COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Rob Schapire

Lecture #1
February 4, 2014

1 What is Machine Learning?

Machine learning studies computer algorithms for learning to do stuff. We might, for instance, be interested in learning to complete a task, or to make accurate predictions, or to behave intelligently. The learning that is being done is always based on some sort of observations or data, such as examples (the most common case in this course), direct experience, or instruction. So in general, machine learning is about learning to do better in the future based on what was experienced in the past.

The emphasis of machine learning is on *automatic* methods. In other words, the goal is to devise learning algorithms that do the learning automatically without human intervention or assistance. The machine learning paradigm can be viewed as "programming by example." Often we have a specific task in mind, such as spam filtering. But rather than program the computer to solve the task directly, in machine learning, we seek methods by which the computer will come up with its own program based on examples that we provide.

Machine learning is a core subarea of artificial intelligence. It is very unlikely that we will be able to build any kind of intelligent system capable of any of the facilities that we associate with intelligence, such as language or vision, without using learning to get there. These tasks are otherwise simply too difficult to solve. Further, we would not consider a system to be truly intelligent if it were incapable of learning since learning is at the core of intelligence.

Although a subarea of AI, machine learning also intersects broadly with other fields, especially statistics, but also mathematics, physics, theoretical computer science and more.

2 Examples of Machine Learning Problems

There are many examples of machine learning problems. Much of this course will focus on classification problems in which the goal is to categorize objects into a fixed set of categories. Here are several examples:

- optical character recognition: categorize images of handwritten characters by the letters represented
- face detection: find faces in images (or indicate if a face is present)
- spam filtering: identify email messages as spam or non-spam
- spoken language understanding: within the context of a limited domain, determine the meaning of something uttered by a speaker to the extent that it can be classified into one of a fixed set of categories
- medical diagnosis: diagnose a patient as a sufferer or non-sufferer of some disease

- classifying or predicting customer/user behavior: predict, for instance, which customers will respond to a particular promotion, or which users will click on a particular ad
- weather prediction: predict, for instance, whether or not it will rain tomorrow.

(In this last case, we most likely would actually be more interested in estimating the *probability* of rain tomorrow.)

Although much of what we will talk about will be about classification problems, there are other important learning problems. In classification, we want to categorize objects into fixed categories. In regression, on the other hand, we are trying to predict a real value. For instance, we may wish to predict *how much* it will rain tomorrow. Or, we might want to predict how much a house will sell for.

A richer learning scenario is one in which the goal is actually to behave intelligently, or to make intelligent decisions. For instance, a robot needs to learn to navigate through its environment without colliding with anything. To use machine learning to make money on the stock market, we might treat investment as a classification problem (will the stock go up or down) or a regression problem (how much will the stock go up), or, dispensing with these intermediate goals, we might want the computer to learn directly how to decide to make investments so as to maximize wealth. A final example is game playing where the goal is for the computer to learn to play well through experience.

3 Goals of Machine Learning Research

The primary goal of machine learning research is to develop general purpose algorithms of practical value. Such algorithms should be efficient. As usual, as computer scientists, we care about time and space efficiency. But in the context of learning, we also care a great deal about another precious resource, namely, the amount of data that is required by the learning algorithm.

Learning algorithms should also be as general purpose as possible. We are looking for algorithms that can be easily applied to a broad class of learning problems, such as those listed above.

Of primary importance, we want the result of learning to be a prediction rule that is as accurate as possible in the predictions that it makes.

Occasionally, we may also be interested in the interpretability of the prediction rules produced by learning. In other words, in some contexts (such as medical diagnosis), we want the computer to find prediction rules that are easily understandable by human experts.

As mentioned above, machine learning can be thought of as "programming by example." What is the advantage of machine learning over direct programming? First, the results of using machine learning are often more accurate than what can be created through direct programming. The reason is that machine learning algorithms are data driven, and are able to examine large amounts of data. On the other hand, a human expert is likely to be guided by imprecise impressions or perhaps an examination of only a relatively small number of examples.

Also, humans often have trouble expressing what they know, but have no difficulty labeling items. For instance, it is easy for all of us to label images of letters by the character represented, but we would have a great deal of trouble explaining how we do it in precise terms.

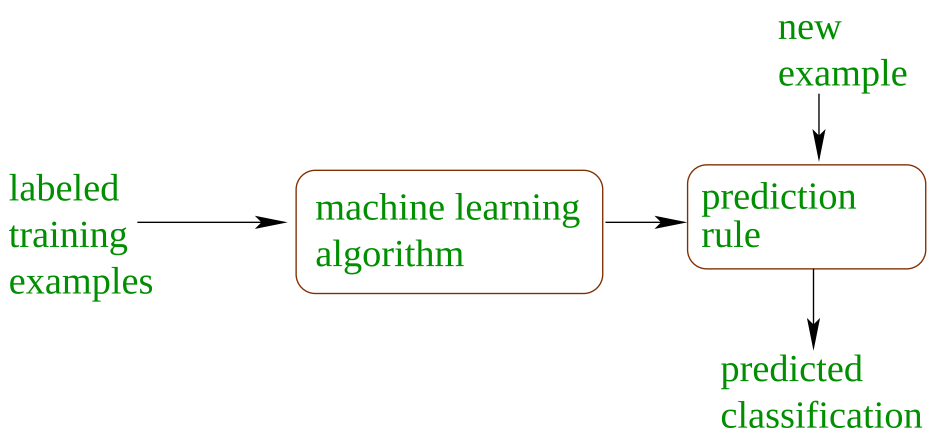


Figure 1: Diagram of a typical learning problem.

Another reason to study machine learning is the hope that it will provide insights into the general phenomenon of learning. Some of the questions that might be answered include:

- What are the intrinsic properties of a given learning problem that make it hard or easy to solve?
- How much do you need to know ahead of time about what is being learned in order to be able to learn it effectively?
- Why are "simpler" hypotheses better?

This course is focused on *theoretical* aspects of machine learning. Theoretical machine learning has much the same goals. We still are interested in designing machine learning algorithms, but we hope to analyze them mathematically to understand their efficiency. It is hoped that theoretical study will provide insights and intuitions, if not concrete algorithms, that will be helpful in designing practical algorithms. Through theory, we hope to understand the intrinsic difficulty of a given learning problem. And we also attempt to explain phenomena observed in actual experiments with learning algorithms.

This course emphasizes the study of mathematical models of machine learning, as well as the design and analysis of machine learning algorithms. Topics include:

- the number of random examples needed to learn;
- the theoretical understanding of practical algorithms, including boosting and support-vector machines;
- online learning from non-random examples (including portfolio selection);
- estimating a probability distribution from samples;
- game theory and its connection to learning.

The two most closely related computer science courses at Princeton are 402 (focusing on core areas of artificial intelligence) and 424 (focusing on techniques for the effective use of data, including machine learning, statistics and data mining). In comparison to 511 which focuses only on the theoretical side of machine learning, both of these offer a broader and more general introduction to machine learning — broader both in terms of the topics covered, and in terms of the balance between theory and applications.

example	**label**
train	
ant	−
bat	+
dolphin	−
leopard	+
sea lion	−
zebra	+
shark	−
mouse	+
chicken	−
test	
tiger	
tuna	
platypus	

Figure 2: A tiny learning problem.

4 A Typical Learning Problem

In a typical learning problem, such as optical character recognition, our goal is to create a system that can read handwritten characters. But rather than attacking the problem directly, we start by gathering a large collection of examples of images of characters which are labeled as to the letter or digit that they represent. We then feed these examples to a general purpose learning algorithm which in turn produces a prediction rule capable (we hope) of classifying new images. The entire process is depicted in Figure 1.

To get a feeling for learning, we looked first at the learning problem shown in Figure 2. Here, examples are labeled positive ("+") or negative ("−"). In this case, the pattern is that "land mammals" are positive, and others negative, although many other suggestions were made in class. Actually, the pattern is kind of hazy, since it might be, for instance, that the positive examples are animals that don't live in the ocean and don't lay eggs. Thus, test examples "tiger" and "tuna" are positive and negative, respectively, but we can only guess the correct label of "platypus" (an egg-laying mammal that spends much of its life in streams and rivers).

The second example is shown in Figure 3. Initially, examples were presented as animal names, and the problem seemed difficult. When the animal names are rewritten in terms of the position in the alphabet of each letter of each name, the pattern becomes more evident, namely, that an example is positive if and only if the third letter has an odd position in the alphabet (for instance, the third letter of "elephant" is "e" which is the fifth letter of the alphabet, so "elephant" is a positive example).

There are some things to notice about this experiment. First of all, more data is better. For instance, we were unsure whether platypus should be a positive or negative example. However, if we had more training data, including, for instance, live-bearing non-mammalian animals (like a scorpion), or other mammals that live in rivers rather than the ocean, we might have been able to better define what the unknown pattern is. Also, many of the other competing suggestions for the first example problem could have been quickly eliminated with more data.

example										**label**
train										
aardvark	→	1	1	18	4	22	1	18	11	−
cow	→	3	15	23						+
giraffe	→	7	9	18	1	6	6	5		−
termite	→	20	5	18	13	9	20	5		−
oyster	→	15	25	19	20	5	18			+
dove	→	4	15	22	5					−
spider	→	19	16	9	4	5	18			+
dog	→	4	15	7						+
elephant	→	5	12	5	16	8	1	14	20	+
test										
rabbit	→	18	1	2	2	9	20			
frog	→	6	18	15	7					
kangaroo	→	11	1	14	7	1	18	15	15	

Figure 3: A second toy learning problem. Examples were intially presented as animal names as in the left column, but later rewritten with the corresponding number of each letter of each animal name, as shown to the right of each name.

Second, it seems very natural to try to find a pattern that is consistent with the data, i.e., a rule that "explains" all of the observed training data. In practice, of course, it might not be possible to find a rule that is perfectly consistent with all of the data, but we still might find it desirable to find a rule that makes as few mistakes as possible.

Thirdly, it seems natural to seek a rule that is not only consistent, but also as simple as possible. For instance, no one was satisfied with a rule that says, on the first problem, that an animal is positive if and only if it begins with a "b" or an "l" or a "z" or an "m" (which was not far from one of the suggestions for this problem). The notion of "simplicity" is tied up with our prior beliefs about what kind of rule we are expecting. When the point of view on the second problem was changed by altering the representation, the kind of rules that seemed simple and natural also changed.

So in sum, there are three conditions that must be met for learning to succeed. First, we need enough data. Second, we need to find a rule that makes a low number of mistakes on the training data. And third, we need that rule to be as simple as possible. Note that the last two requirements are typically in conflict with one another: we sometimes can only find a rule that makes a low number of mistakes by choosing a rule that is more complex, and conversely, choosing a simple rule can sometimes come at the cost of allowing more mistakes on the training data. Finding the right balance is perhaps the most central problem of machine learning.

The notion that simple rules should be preferred is often referred to as "Occam's razor."

5 Learning Models

To study machine learning mathematically, we need to formally define the learning problem. This precise definition is called a *learning model*. A learning model should be rich enough to capture important aspects of real learning problems, but simple enough to study the problem mathematically. As with any mathematical model, simplifying assumptions are unavoidable.

A learning model should answer several questions:

- What is being learned?
- How is the data being generated? In other words, where does it come from?
- How is the data presented to the learner? For instance, does the learner see all the data at once, or only one example at a time?
- What is the goal of learning in this model?

6 Definitions

Before getting to our first learning model, we will need some definitions. An *example* (sometimes also called an *instance*) is the object that is being classified. For instance, in OCR, the images are the examples.

Usually, an example is described by a set of *attributes*, also known as *features* or *variables* or *dimensions*. For instance, in medical diagnosis, a patient might be described by attributes such as gender, age, weight, blood pressure, body temperature, etc.

The *label* or *class* is the category that we are trying to predict. For instance, in OCR, the labels are the possible letters or digits being represented. During training, the learning algorithm is supplied with *labeled examples*, while during testing, only *unlabeled examples* are provided.

To make things as simple as possible, we will often assume that only two labels are possible that we might as well call 0 and 1. We also will make the simplifying assumption that there is a mapping from examples to labels. This mapping is called a *concept*. Thus, a concept is a function of the form $c : X \to \{0, 1\}$ where X is the space of all possible examples called the *domain* or *instance space*. A collection of concepts is called a *concept class*. We will often assume that the examples have been labeled by an unknown concept from a *known* concept class.

7 The Consistency Model

Our first learning model, called the consistency model, is rather unrealistic, but it is intuitive, and it is a good place to start. Many of the ideas that come up will also be of value later. The model captures the intuitive idea that the prediction rule that is derived from a set of examples should be consistent with their observed labelings.

We say that a concept class $\mathcal{C}$ is learnable in the consistency model if there is an algorithm A which, when given any set of labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$ and $y_i \in \{0, 1\}$, finds a concept $c \in \mathcal{C}$ that is consistent with the examples (so that $c(x_i) = y_i$ for all i), or says (correctly) that there is no such concept. Moreover, we are especially interested in finding efficient algorithms in this model.

Next time, we will look at examples of concept classes that are or are not learnable in this model.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #2
Scribe: Scott Yak — February 6, 2014

---

1 Review of definitions

We are now mostly concerned with classification problems. For a particular classification problem, we have each *example/instance* coming from X, a much larger space of all possible examples called the *domain space* or *instance space*. Each example is associated with a label, which we denote as y. For simplicity, we assume there are only two possible labels.

1.1 The Concept and Concept class

A *concept*, which is what we are trying to learn, is an unknown function that assigns labels to examples. It takes the form $c : X \to \{0, 1\}$.

A collection of concepts is called a *concept class.* We will often assume that the examples have been labeled by an unknown concept from a *known* concept class.

1.2 The Consistency Model

We say that a concept class $\mathcal{C}$ is learnable in the consistency model if there is an algorithm[1] A which, when given any set of labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$ and $y_i \in \{0, 1\}$, finds a concept $c \in \mathcal{C}$ that is consistent with the examples (so that $c(x_i) = y_i$ for all i), or says (correctly) that there is no such concept.

In the following sections, we'll look at some concept classes, and see if they are learnable.

2 Examples from Boolean Logic

For this section, we consider the instance space of n-bit vectors:

$$X = \{0, 1\}^n, \quad \mathbf{x} = (x_1, \ldots, x_n)$$

Each n-bit vector represents the values of n boolean variables. Examples of boolean variables could be the following:

$$\begin{aligned} x_1 &= \text{"is a bird"} \\ x_2 &= \text{"is a mammal"} \\ x_3 &= \text{"lives on land"} \\ &\vdots \\ &\textit{etc.} \end{aligned}$$

[1]One caveat is that this algorithm should be reasonably "efficient", and by "efficient" we mean something like having a polynomial runtime bound. This caveat is important, because if we do not require the algorithm to be efficient, then we can always come up with an algorithm that tries every possible concept in the concept class, and we would be able to make the uninteresting claim that all finite concept classes are learnable, making this learning model uninteresting. In any case, we are mostly interested in learning algorithms that are efficient enough to be practical.

2.1 Monotone conjunctions

Suppose our concept class is the set of all *monotone conjunctions*. A monotone conjunction is the AND of some of our boolean variables, where we do not allow the boolean variables to be negated. One possible concept in $\mathcal{C}$ would be $c(\mathbf{x}) = x_2 \wedge x_5 \wedge x_7$, which is a function that takes in an n-bit vector and outputs either 0 (or $-$) or 1 (or $+$).

example	**label**
0 1 1 0 1	$+$
1 1 0 1 1	$+$
0 0 1 0 1	$-$
1 1 0 0 1	$+$
1 1 0 0 0	$-$

Given the labelled examples above, what is the monotone conjunction consistent with it? In this small example, the answer is $x_2 \wedge x_5$. But is there a general algorithm for finding a consistent monotone conjunction?

Consider this algorithm: We start off by only considering the positive examples, and we keep only the columns where all the entries are '1'. For example, only the second column and the fifth column are all ones, so our monotone conjunction is $x_2 \wedge x_5$.

example	**label**
0 1 1 0 1	$+$
1 1 0 1 1	$+$
1 1 0 0 1	$+$
0 1 0 0 1	

We then take this monotone conjunction and evaluate it with our negative examples. For example, $x_2 \wedge x_5$ evaluated on 00101, one of our negative examples, is $0 \wedge 1 = 0$. If our monotone conjunction evaluates to 0 on all negative examples, then the algorithm outputs that as the concept; otherwise the algorithm says that no consistent concept exists.

Is this algorithm correct? We know that any concept the algorithm outputs must be consistent because it has already been checked against the examples, but when the algorithm says that there is no consistent concept, how can we be sure that it is really the case?

We note that any monotone conjunction that is consistent with all positive examples can only contain the boolean variables output by the algorithm. This is because all other boolean variables were rejected for taking the value of 0 for at least one positive example, so if we included any of those boolean variables, the concept would wrongly label that positive example as a negative example. In the above example, we could not have included x_1 into the conjunction, since including x_1 would cause the concept to mislabel the positive example "01101" as negative. For similar reasons, we could not have included x_3 or x_4 either. This means that only subsets of the boolean variables that the algorithm selects have monotone conjunctions that are consistent with all positive examples, in this case, $\{\}, \{x_2\}, \{x_5\}$, and $\{x_2, x_5\}$.

Now that we have all possible concepts that are consistent with all positive examples, we would like to choose one that is also consistent with all the negative examples, if possible. Note that if any negative example were consistent with the monotone conjunction of any subset of the selected boolean variables, it would also be consistent with the monotone conjunction of all the selected boolean variables. For instance, any bit-vector that gets evaluated to '0' with x_2 must also get evaluated to '0' with $x_2 \wedge x_5$. This implies that if the monotone conjunction of all the selected boolean variables is not consistent with any of the negative examples, then none of the monotone conjunctions that are consistent with all the positive examples would be consistent with that negative example either, so no consistent concept exists. In other words, if a consistent concept exists, the algorithm would have considered it, so it would only state that there are no consistent concepts if there really aren't any.

Since we have a learning algorithm for the concept class of monotone conjunctions, we can conclude that the concept class of monotone conjunctions is learnable.

2.2 Monotone disjunctions

Now let's consider the concept class of monotone disjunctions. A monotone disjunction is the OR of some of our boolean variables, where we do not allow the boolean variables to be negated. One possible concept in $\mathcal{C}$ would be $c(\mathbf{x}) = x_2 \vee x_5 \vee x_{20}$. Is this concept class learnable?

Consider the same set of examples again. Using De Morgan's rule, $\overline{x_1 \vee x_2} = \overline{x}_1 \wedge \overline{x}_2$, we can convert the examples into the monotone conjunction form by negating both the example bit-vector and the labels:

example	**label**		**example**	**label**
0 1 1 0 1	$+$		1 0 0 1 0	$-$
1 1 0 1 1	$+$	$\longrightarrow$	0 0 1 0 0	$-$
0 0 1 0 1	$-$		1 1 0 1 0	$+$
1 1 0 0 1	$+$		0 0 1 1 0	$-$
1 1 0 0 0	$-$		0 0 1 1 1	$+$

Now we just need to find the consistent monotone *conjunction* for the converted examples, and flip the conjunctions to disjunctions to obtain the consistent monotone disjunction for the original examples. This means that we have reduced the problem of learning from the monotone disjunction concept class to the already solved problem of learning from the monotone conjunction concept class, so the monotone disjunction concept class is also learnable.

2.3 Conjunctions

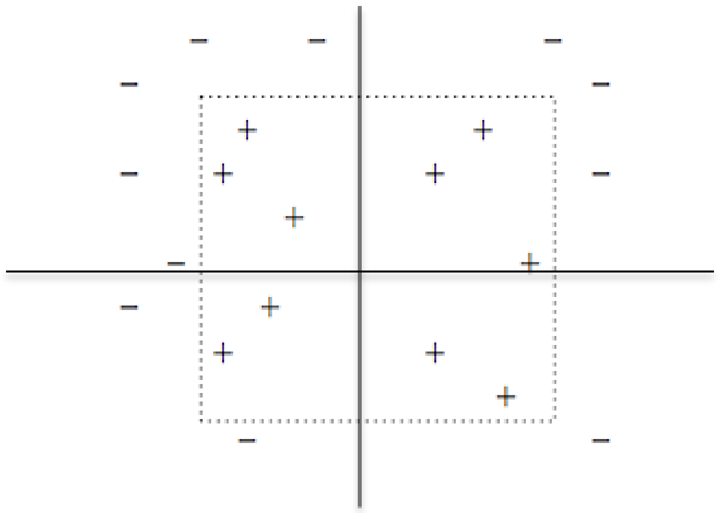
Now let the concept class be the set of conjunctions, not necessarily monotone. This means that we allow the boolean variables to be negated. For example, one possible conjunction would be $x_3 \wedge \overline{x}_7 \wedge x_{10}$, where x_7 is the negated variable. Is this concept class learnable?

Once again, we observe that this problem can also be reduced to the monotone conjunction problem. For each variable x_i in the bit-vector, we add a new variable $z_i = \overline{x}_i$, and we tack it on to the examples' bit-vectors, so $(x_1, \ldots, x_m) \to (x_1, \ldots, x_m, z_1, \ldots, z_m)$. We obtain the consistent monotone conjunction for this converted set of examples. Now suppose we obtain the monotone conjunction $x_3 \wedge z_7 \wedge x_{10}$; we would be able to convert it back to a general conjunction, $x_3 \wedge \overline{x}_7 \wedge x_{10}$. Thus, the concept class of conjunctions is also learnable.

3 Examples from Geometry

3.1 Axis-aligned rectangles

Now let's consider points on a 2-D plane, $X = \mathbb{R}^2$. Each point is assigned a label $+$ or $-$. We consider the concept class of axis-aligned rectangles — each concept is a boolean function where the points in the rectangle are labelled '+', and the points outside of the rectangle are labelled '$-$'. So the learning problem is this: Given a set of labelled points on a plane, find an axis-aligned rectangle such that all the points inside are '+' and all the points outside are '$-$' (such as the dotted-line box in the following diagram).

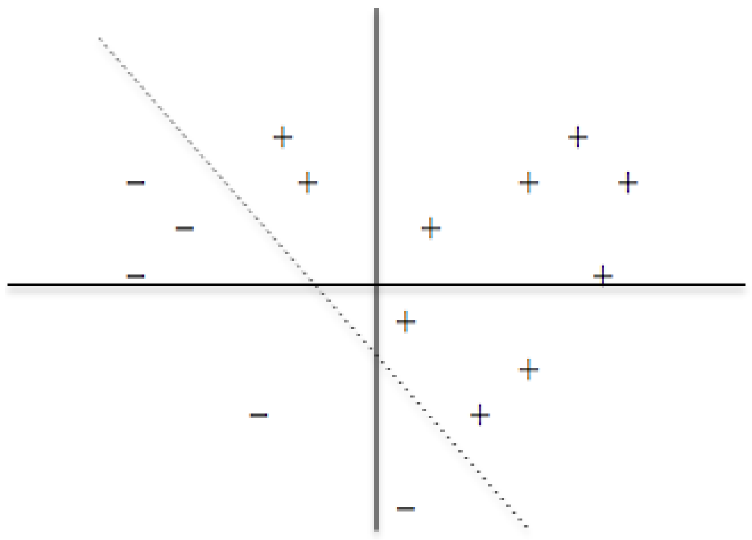


One algorithm would be to scan through all the positive examples, and use the topmost, bottommost, leftmost, and rightmost coordinates to define the smallest enclosing rectangle for the positive examples. Then we check if any of the negative examples are contained in this rectangle — if there is, we state there are no consistent concepts; otherwise we output that rectangle as the consistent concept [2]. So the concept class of axis-aligned rectangles is learnable.

3.2 Half-spaces

Let's consider points in an n-dimensional space, $X = \mathbb{R}^n$. Again, each point is assigned a label $+$ or $-$. We consider the concept class of half-spaces or linear threshold functions — each concept is a boolean function where the points on one side of a linear hyperplane are labelled '+', and the points on the other side are labelled '$-$'. The concept we want to find is a linear hyperplane that separates the positive from the negative examples (such as the dotted-line in the diagram on the next page).

[2] Notice the general trick of finding the concept that includes as little of the domain space as possible while still remaining consistent. This trick was previously used for the monotone conjunctions



We can define a hyperplane as

$$\mathbf{w} \cdot \mathbf{x} = b$$

where $\mathbf{w}$ and b are fixed values that determine the hyperplane. We denote $\mathbf{x}_i$ as the i^{th} example, and y_i as the corresponding label. Algebraically, what we are trying to do is:

$$\begin{aligned}
&\text{find} \\
&\qquad \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R} \\
&\text{such that } \forall\ i, \\
&\qquad \mathbf{w} \cdot \mathbf{x}_i > b \text{ if } y_i = 1 \\
&\qquad \mathbf{w} \cdot \mathbf{x}_i < b \text{ if } y_i = 0
\end{aligned}$$

This turns out to be a linear programming problem, which is known to be efficiently solvable (there are math packages that can solve such linear programs well). This shows that the concept class of half-spaces is learnable.

4 More examples from Boolean logic

Once again, our domain space is the set of n-bit vectors.

4.1 k-CNF

A formula in the *conjunctive normal form*(CNF) is the conjunction of disjunctions, or the AND of ORs. For example, the following formula is a CNF:

$$(x_1 \vee x_3) \wedge (x_4 \vee \overline{x}_2) \wedge (x_2 \vee \overline{x}_3 \vee x_6) \wedge (x_7)$$

It contains *literals*, which are boolean variables either in their negated or unnegated form, and consists of *clauses* (literals ORed together) which are ANDed together. For a k-CNF, each clause can contain at most k literals, and there are no restrictions on the number of clauses in the formula. k is considered a constant. The example above would be a 3-CNF, because the longest clause contains 3 literals.

Now consider the concept class of k-CNFs. Is there an algorithm that can come up with a consistent concept? [3]

For simplicity, let's first consider 2-CNFs. One way to find a consistent concept would be to list all the possible pairs of literals (which include both the negated and unnegated forms of the boolean variables), name the OR of that as new variables, and tack them to the examples' bit-vectors (similar to what we did before with non-monotone conjunctions). This converts the CNF into a monotone conjunction[4], which we know how to solve, and once we get the solution, we can convert it back to a 2-CNF by substituting back the pairs that were named as variables. For the more general k-CNF, our new variables would include the disjunction of all subsets of the set of literals up to size k. Thus, the concept class of k-CNF is also learnable.

4.2 2-term DNF

A formula in the *disjunctive normal form*(DNF) is a disjunction of conjunctions, or an OR of ANDs. It is made up of a conjunction of *terms*, which is a conjunction of an unrestricted number of literals - the size of each term can be as large as desired. Note the difference from CNF — here, it's the number of "parentheses", and not the number of things in the parentheses, that is restricted. A k-term DNF, as the name suggests, can only contain up to k-terms. For instance, $(x_1 \wedge x_3 \wedge x_4) \vee (\overline{x}_2 \wedge \overline{x}_3 \wedge x_6 \wedge x_7)$ is a 2-term DNF.

Now we consider the concept class of 2-term DNFs. Is this class learnable?

We note the distributive property of conjunction over disjunction — we can "expand" the 2-term DNF in a way analogous to how we might multiply two polynomials, but we treat the ORs like multiplication signs, and the ANDs like additions. So,

$$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x}_2 \wedge x_4) = (x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_4) \wedge (x_2 \vee \overline{x}_2) \wedge (x_2 \vee x_4) \wedge (x_3 \vee \overline{x}_2) \wedge (x_3 \vee x_4)$$

This means that we can convert any 2-term DNF to a 2-CNF. Great! Does it mean that we can reduce 2-term DNF problem to 2-CNF? We could use the 2-CNF algorithm to produce a consistent 2-CNF, but there is no guarantee that it would yield a 2-CNF that can to factorized into a 2-term DNF! The fact that we can convert any 2-term DNF to a 2-CNF only shows that the 2-term DNFs form a concept class that is a subset of the concept class of 2-CNFs. Even if we find a consistent 2-CNF, it doesn't mean that we get a consistent 2-term DNF.

In fact, the 2-term DNF learning problem is NP-hard, so an efficient learning algorithm is unlikely to exist. This is an example of a concept class that is not learnable.

4.3 General DNF

Now we consider the concept class of general DNFs, where there are no restrictions on the number of terms in the DNF. Is this learnable?

Consider the following algorithm: For each positive example, we take the negated form of the boolean variable if the value is zero, or taking the unnegated form of the boolean variable if the value is 1, and AND all these literals to form a term. For example:

[3]Note that this is different from the CNF satisfaction problem. In the CNF satisfaction problem, we start with a known CNF and try to come up with a positive example. In the learning problem here, we start off with a bunch of examples, and we are trying to find the unknown CNF that is consistent with all the examples.

[4]We might get unnecessary terms such as $x_1 \vee \overline{x}_1$, but it doesn't matter because we can just set it to 0.

example	**label**	DNF term
0 1 1 0 1	$+$	$\overline{x}_1 \wedge x_2 \wedge x_3 \wedge \overline{x}_4 \wedge x_5$
1 1 0 1 1	$+$	$x_1 \wedge x_2 \wedge \overline{x}_3 \wedge x_4 \wedge x_5$
0 0 1 0 1	$-$	-

Once we obtain the terms for all the positive examples, we OR them together to form a DNF. This DNF is automatically consistent with all the positive examples by construction. We then take this DNF and check if it is consistent with the negative examples, and if it is, the algorithm outputs the DNF; otherwise it outputs ”no consistent DNF”. Since this algorithm works, the general DNF concept class is learnable.

However, this algorithm seems to be performing a somewhat unsatisfactory kind of “learning”. For one, the DNF that this algorithm outputs is going to label any instance it has not seen before as negative! Also, the DNF that the algorithm outputs is of a size comparable to the size of the training data. So in effect, this “learning” algorithm is really nothing more than a lookup table.

5 Problems with the Consistency model

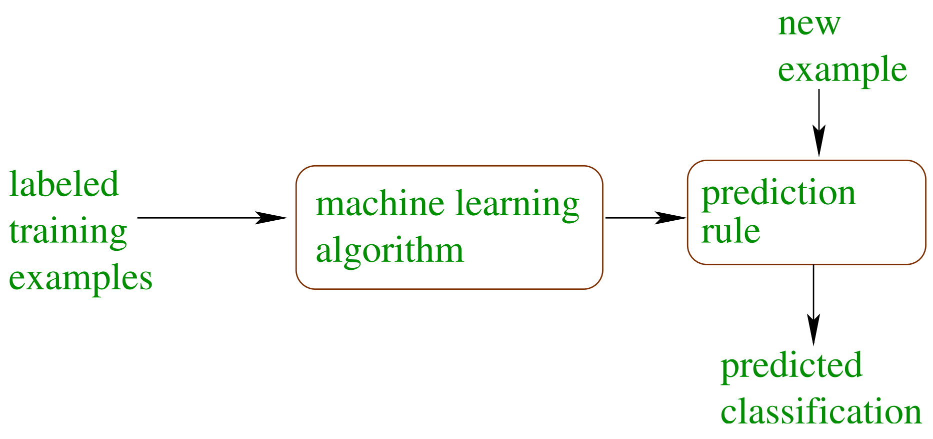
One problem we saw with the DNF example is that the consistency model doesn't say anything about how the concept that the algorithm learns generalizes to new data. Even though the concept class is apparently learnable under the consistency model, it does so in an unsatisfactory way that seems unrelated to what we typically mean when we say “learning”. It seems, then, that the consistency model doesn't really say much about learning at all! This suggests that we need a new model.

6 Quick review of probability concepts

event	—	Something that either happens or not. A probabilistic outcome.
random variable	—	Variable that takes values probabilistically. Has a distribution.
distribution	—	$Pr[X = x]$, and $\sum_x Pr[X = x] = 1$
expected value	—	$E[X] = \sum_x Pr[X = x] \cdot x$ $E[f(X)] = \sum_x Pr[X = x] \cdot f(x)$
linearity of expectations	—	$E[X + Y] = E[X] + E[Y]$. $E[cX] = cE[X]$. Always.
conditional probability	—	$Pr[a\|b] = \frac{Pr[a \wedge b]}{Pr[b]}$
independence	—	a, b are independent if $Pr[a \wedge b] = Pr[a] \cdot Pr[b]$ $\forall \text{r.v.} A, B, \forall a, b : Pr[A = a \wedge B = b] = Pr[A = a] \cdot Pr[B = b]$ $(\textit{shorthand} : Pr[A \wedge B] = Pr[A] \cdot Pr[B])$
If X, Y independent,	$\Rightarrow$	$E[XY] = E[X] \cdot E[Y]$
union bound	—	$Pr[a \vee b] \leq Pr[a] + Pr[b]$

7 Introduction to the PAC learning model

We aim to capture some notion of learning in this new model. This means that this model should be able to say something about generalizing from a smaller set of data to a larger set of instances.



Our goal now is to develop a hypothesis that is as accurate as possible — we are not as concerned about discovering the underlying truth. An important question we need to consider is — where do the examples come from? We need to assume that the examples are in some way related — otherwise we cannot learn anything. In this model, we would make the assumption that the examples are generated randomly from the same distribution (not necessarily uniform).

Since our goal is to obtain an accurate hypothesis, we need to be precise about what we mean by "accurate". We define the generalization/true error as follows:

$$Pr_{x\sim D}[h(x) \neq c(x)] = err_D(h)$$

where x is some example that comes from an unknown target distribution D that generates each example, and h is a hypothesis. We typically assume that the different examples are independent of each other (even though that is not always realistic, for instance, in the case of an email spam filter). We also assume that the training examples and test examples come from the same source — i.e., they are drawn from the same distribution D.

As for the labels, we assume for simplicity that there are only two possible values. The labels are applied in accordance to some unknown concept, $c : X \to \{0, 1\}$, that comes from a known concept class $\mathcal{C}$.

As an overview:

$$\begin{tikzcd}
 & & \begin{array}{c}\text{test example } x\\ x \sim D\end{array} \arrow[d] \\
\begin{array}{c}\text{Labelled}\\ \text{training}\\ \text{examples}\\ (x_1, c(x_1))\\ \vdots\\ (x_m, c(x_m))\end{array} \arrow[r] & \begin{array}{c}\text{Learning algorithm}\\ \text{A}\end{array} \arrow[r] & \begin{array}{c}\text{Hypothesis}\\ \text{H}\end{array} \arrow[d] \\
x_i \sim D & & \text{label } h(x)
\end{tikzcd}$$

We reiterate that in this learning model, we aim to formulate a hypothesis that has $err_D(h)$ that is as small as possible. We no longer view the goal of the learning algorithm as that of obtaining a hypothesis that is consistent on the training set. Rather, the goal now is to obtain a hypothesis that does well on test data, and that is at least "approximately correct". But since the training examples are considered to be drawn randomly from an unknown distribution, there is always a chance that the training set that is drawn is very unrepresentative of the source distribution. For instance, there is a non-zero probability that an OCR program never sees the letter ‘t’ up till a time we test it on a normal piece of text. Unlikely, but possible. For us to be able to say anything useful about the hypothesis, then, we would need to disregard extremely unlikely events that can completely mess up the machine learning algorithm. Thus, the accurate guarantees are “probabilistic”. This sets us up for the “Probably Approximately Correct” (PAC) learning model.

Next time, we will elaborate more on the PAC learning model.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Kevin Lai

Lecture #3
February 11, 2014

1 The Probably Approximately Correct (PAC) Model

A target concept class $\mathcal{C}$ is **PAC-learnable** by a hypothesis space $\mathcal{H}$ if there exists an algorithm A such that for all $c \in \mathcal{C}$, any target distribution D, and any positive ϵ and δ, A uses a training set $S = \langle (x_1, c(x_1)), (x_2, c(x_2)), ..., (x_m, c(x_m)) \rangle$ consisting of $m = \text{poly}(\frac{1}{\epsilon}, \frac{1}{\delta}, ...)$ examples taken i.i.d. from D and produces $h \in \mathcal{H}$ such that $\Pr[\text{err}_D(h) \leq \epsilon] \geq 1 - \delta$.

A few comments on notation. ϵ is called the accuracy parameter, and we call h "ϵ-good" if $\text{err}_D(h) \leq \epsilon$, where $\text{err}_D(h)$ is called the **true error** or the **generalization error**. δ is the confidence parameter. ϵ and δ are user-specified parameters (eg. 5% and 1%). The name "Probably Approximately Correct" comes from the fact that we want a hypothesis that is approximately correct (ϵ-good) with high probability (namely $1-\delta$). The probability is taken over the choice of S, which will determine which h the algorithm chooses. This is a reasonable goal because there is always a small chance that the test data will be very unrepresentative of D.

We assume that the training set and the test data are drawn from the same distribution D. In general $\mathcal{H}$ will not necessarily be the same as $\mathcal{C}$. Finally, we may also want m to be polynomial in the size of each example or in the size of the target concept c.

2 Learning positive half-lines

We will now look at a series of examples of PAC-learnable concept classes, starting with the class of positive half-lines. In this example, the domain $\mathcal{X}$ is the real line, and $\mathcal{C} = \mathcal{H} = \{\text{positive half lines}\}$. A positive half-line is defined by a threshold (a real number): all points to the left of the threshold are labeled negative, while all points to the right of the threshold are labeled positive.

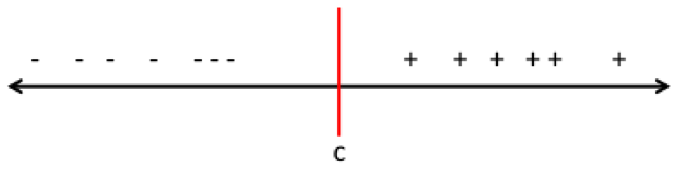


Figure 1: The target concept c is a half-line

To find a hypothesis, we will simply pick some h that is consistent with the test data. We can do this by scanning the test data in ascending sorted order until we find the greatest negatively labeled point and the smallest positively labeled point. We then set the threshold of our half-line anywhere in this interval. Note that we can always find a consistent h because $\mathcal{H} = \mathcal{C}$.

As shown in Figure 3, the generalization error of h will be the probability mass that falls between the target concept c and our hypothesis h. Points in this region will be labeled

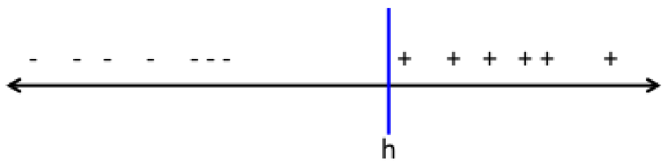


Figure 2: Half-line h outputted by our algorithm based on test data

differently by h and c. Any points that are either to the right or to the left of both h and c will be labeled the same by h and c.

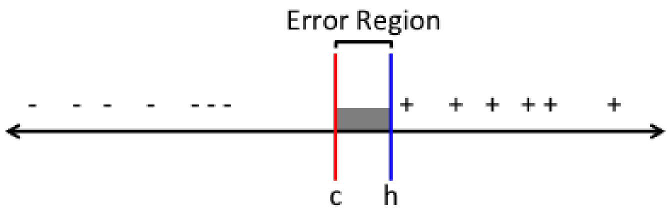


Figure 3: The generalization error of h is the probability mass between h and c

We need to show that the generalization error is low with high probability. Equivalently, we can show that $\Pr[\mathrm{err}_D(h) > \epsilon] \le \delta$. There are two cases where $\mathrm{err}_D(h) > \epsilon$. Let B_+ be the event that h is more than ϵ probability mass to the right of c, and let B_- be the event that h is more than ϵ probability mass to the left of c.

To determine the probability that B_+ or B_- occur, we define two points on the number line, as shown in Figure 4. Let r_+ be a point to the right of c such that the interval $[c, r_+]$ has ϵ probability mass. Likewise, let r_- be a point to the left of c such that the interval $[r_-, c]$ has ϵ probability mass.

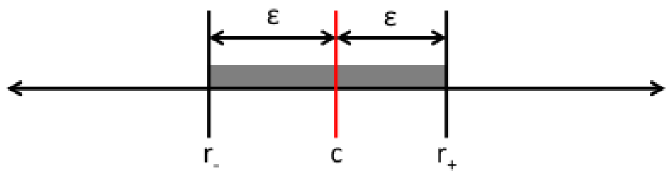


Figure 4: r_+ and r_- are both ϵ probability mass away from c

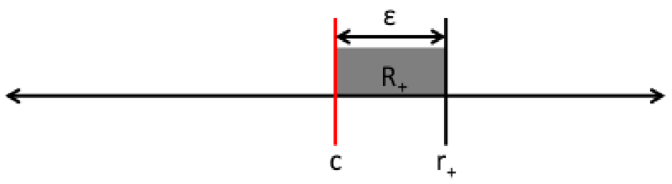


Figure 5: R_+ has probability mass ϵ

We will first calculate the probability of B_+. Let R_+ be the interval $[c, r_+]$, which has probability mass ϵ. Now note that B_+ can only occur if h is to the right of r_+, which only occurs if all $x_i \notin R_+$. If any of the training points were in R_+, h would be to the left of r_+

(since the training point would have a positive label). Observe now that $\Pr[x_i \notin R_+] \leq 1-\epsilon$ because R_+ has probability mass ϵ. Then:

$$\Pr[B_+] = \Pr[x_1 \notin R_+ \wedge x_2 \notin R_+ \wedge x_m \notin R_+] \leq (1-\epsilon)^m \tag{1}$$

where the last inequality follows by independence of the x_i's. Note also that while the x_i's are random, R_+ is fixed because it depends on a fixed c. Also, by symmetry, $\Pr[B_-] \leq (1-\epsilon)^m$.

Now we can bound the probability that $\mathrm{err}_D(h) > \epsilon$:

$$\begin{aligned}
\Pr[\mathrm{err}_D(h) > \epsilon] &\leq \Pr[B_+ \vee B_-] && (2)\\
&\leq \Pr[B_+] + \Pr[B_-] \qquad \text{(union bound)} && (3)\\
&\leq 2(1-\epsilon)^m && (4)\\
&\leq 2e^{-\epsilon m} \qquad (\forall x, 1-x \leq e^{-x}) && (5)
\end{aligned}$$

We want $\Pr[\mathrm{err}_D(h) > \epsilon] \leq \delta$, so we set (5) $\leq \delta$ and solve for m to get $m \geq \frac{1}{\epsilon}\ln\frac{2}{\delta}$. This shows that $\mathcal{C}$ is PAC-learnable by $\mathcal{H}$.

If we set $\delta =$ (5), we can write this equivalent statement: with probability at least $1-\delta, \mathrm{err}_D(h) \leq \frac{1}{m}\ln\frac{2}{\delta}$.

3 Learning intervals

Our next concept class $\mathcal{C}$ is the set of intervals on the real line. This will also be our hypothesis space $\mathcal{H}$. Each $c \in \mathcal{C}$ specifies an interval on the real line that will be labeled positive. All points outside of the interval are labeled negative.

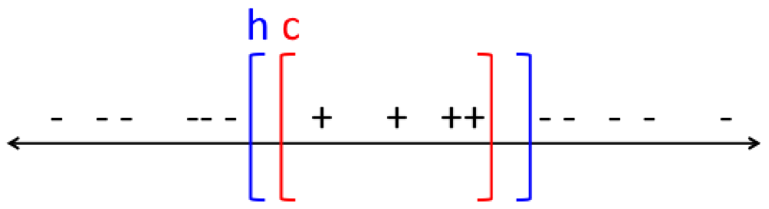


Figure 6: Test data generated by a target concept interval c. h is a consistent hypothesis interval

Our algorithm will simply pick a consistent $h \in \mathcal{H}$. We can then prove that the generalization error of h is low with high probability by an analogous argument to the half-line case. That is, $\mathcal{C}$ is PAC-learnable by $\mathcal{H}$.

We will briefly summarize the argument. Let c_l be the left boundary of c. We make an interval of probability mass $\frac{\epsilon}{2}$ on both sides of c_l. Then the analysis will be the same as in the half-line case, except with a constant factor difference. We repeat this argument for the right boundary of c. At the end, we will have a union bound over four bad events, each with probability $e^{-\frac{\epsilon m}{2}}$, so $m \geq \frac{2}{\epsilon}\ln\frac{4}{\delta}$.

4 Learning axis-aligned rectangles

Our last example will be learning the concept class $\mathcal{C}$ of axis-aligned rectangles. We will use the algorithm we had in the previous lecture, which finds the smallest axis-aligned rectangle consistent with the data.

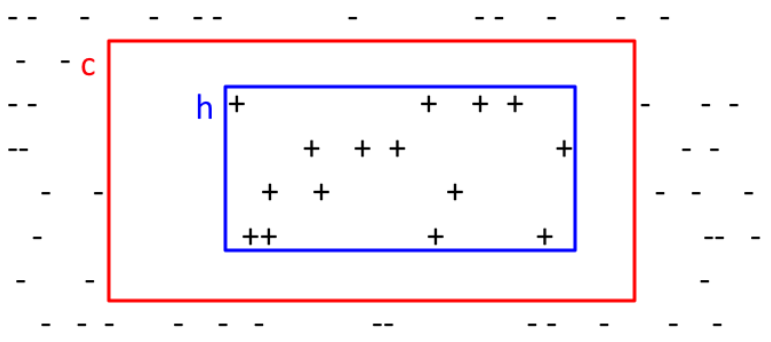


Figure 7: Test data generated by a target concept rectangle c. h is the smallest consistent rectangle

To bound the probability that the generalization error is greater than ϵ, we create four "bad" regions, each created by growing a rectangle from the interior of one wall of c until the region has probability mass $\frac{\epsilon}{4}$.[1] It does not matter that these rectangles overlap. Call these regions $R_1, ..., R_4$. Next, we use an argument that is analogous to the half-line case to bound the probability that no points land in each region individually. Let $B_1, ..., B_4$ be the events that no points land in regions $R_1, ..., R_4$ respectively. Then:

$$\begin{aligned}
\Pr[\mathrm{err}_D(h) > \epsilon] &\leq \Pr[B_1 \vee B_2 \vee B_3 \vee B_4] && (6)\\
&\leq \Pr[B_1] + \Pr[B_2] + \Pr[B_3] + \Pr[B_4] \qquad \text{(union bound)} && (7)\\
&\leq 4(1 - \epsilon/4)^m && (8)\\
&\leq 4e^{-\frac{\epsilon m}{4}} && (9)
\end{aligned}$$

Then we get $m \geq \frac{4}{\epsilon} \ln \frac{4}{\delta}$. As a final note, observe that we can extend this argument to axis-aligned hyperrectangles in an arbitrary number of dimensions.

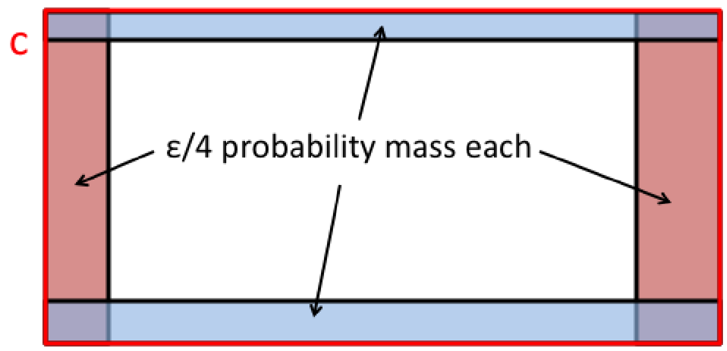


Figure 8: Regions analogous to R_+ in the half-line case. We bound the probability that no points fall in each of these regions. Each region has probability mass $\frac{\epsilon}{4}$ and the regions overlap

[1] If the region hits the opposite wall of c before the region reaches probability mass $\frac{\epsilon}{4}$, then we simply stop growing the region. Thus, each region has probability mass at most $\frac{\epsilon}{4}$.

5 Proving PAC results in general

We just saw several examples of proving concept classes are PAC-learnable using proofs tailored to each problem. However, it would be more convenient if we had some way to prove PAC results in general. Fortunately, such a way exists for finite hypothesis spaces (i.e. $|\mathcal{H}| < \infty$).

Theorem 1. *Suppose an algorithm A always finds a hypothesis $h_A \in \mathcal{H}$ consistent with m examples where $m \geq \frac{1}{\epsilon}(\ln |\mathcal{H}| + \ln \frac{1}{\delta})$. Then $\Pr[\text{err}_D(h_A) > \epsilon] \leq \delta$.*

The proof of Theorem 1 also will lead to this similar statement: with probability at least $1 - \delta$, if h_A is consistent, then $\text{err}_D(h_A) \leq \frac{1}{m}(\ln |\mathcal{H}| + \ln \frac{1}{\delta})$. This statement is essentially saying that if an algorithm finds a consistent hypothesis and uses enough data relative to the complexity of the hypothesis space and δ, the generalization error will be low.

6 Complexity and $|\mathcal{H}|$

The $\ln |\mathcal{H}|$ term in Theorem 1 is a measure of complexity of the hypothesis space. Intuitively, we can understand the logarithmic term as essentially the number of bits required to uniquely label each $h \in \mathcal{H}$, which would be the base two logarithm of $|\mathcal{H}|$.

We did an exercise in class to demonstrate the relevance of $|\mathcal{H}|$. Everyone wrote down 20 bits of their choosing, corresponding to the output of a hypothesis on the domain $\{1, 2, ..., 20\}$. Professor Schapire then revealed the "training" data, which was a set of 10 bits, corresponding to the output of the target concept for {1,2,...,10}. We selected the hypothesis with the lowest training error in the class. Then we evaluated that hypothesis's performance against the 10 bits output by the target concept for {11,12,...,20}. In our example, the best hypothesis had a training error of 10% and a test error of 10%. However, as Professor Schapire determined his bits using random coin tosses, in general the hypothesis with the best training error will have an expected test error of 50%. In a large class, it is likely to find a hypothesis with low training error, but that hypothesis will still have an expected test error of 50%.

The purpose of the exercise was to demonstrate that with a bigger hypothesis space, there is a greater chance for a poor hypothesis to match the training set well, despite having poor generalization error. Thus, complexity in the hypothesis space will tend to increase the probability of choosing a hypothesis that fits the training set well, simply by chance, but which actually performs poorly on test data.

Example 1. Suppose $\mathcal{C}$ is the set of monotone conjunctions in n-dimensions. This will also be our hypothesis space. $|\mathcal{H}| = 2^n$ because each variable can either be included or not included in our hypothesis. Since our algorithm from last lecture finds a consistent hypothesis, Theorem 1 implies that if $m \geq \frac{1}{\epsilon}(n \ln 2 + \ln \frac{1}{\delta})$, then $\Pr[\text{err}_D(h_A) > \epsilon] \leq \delta$. Since m only needs to be at most polynomial in the size of $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and n, we have shown that the class of monotone conjunctions is PAC-learnable.

Example 2. Now suppose that $\mathcal{C}$ is the set of all DNFs, and let this be our hypothesis space as well. Now since we can think of every DNF as a boolean formula (given values for n variables, the DNF outputs one of two answers), and every boolean formula can be represented by a DNF, $|\mathcal{H}|$ is just the number of boolean formulas on n variables, which is 2^{2^n}. Using Theorem 1, we get that $m \geq \frac{1}{\epsilon}(\ln 2^{2^n} + \ln \frac{1}{\delta}) = \frac{1}{\epsilon}(2^n \ln 2 + \ln \frac{1}{\delta})$, which is exponential in the number of variables. This is consistent with our conclusions from last lecture that DNFs likely can't be learned efficiently with a small amount of data.

An informal way to see that the class of DNFs likely can't be learned efficiently is to look at our algorithm from last lecture for learning DNFs. In that algorithm, we included a clause for each of the positive examples in our training set. In the worst case, this could give us a hypothesis of size $O(mn)$. If we approximate $\ln |\mathcal{H}|$ with the size of each hypothesis, then the error is $\epsilon \geq \frac{O(mn)+\ln \frac{1}{\delta}}{m} \geq cn$, where c is some constant. This is too large, as we would like our error to be less than 1. So our previous algorithm does not efficiently learn DNFs.

In general, we see that if our hypothesis space is not too complex, finding a consistent hypothesis will allow us to achieve low generalization error with a reasonably small amount of data. However, this conclusion does not say anything about computational efficiency. Even if only a small amount of data is required, it may be difficult to design efficient learning algorithms.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #4
Scribe: Akshay Mittal — February 13, 2013

1 Proof of learning bounds

For intuition of the following theorem, suppose there exists a hypothesis h which is ϵ-bad and makes at least one mistake with a few 100 examples. Therefore, since h is ϵ-bad, then with high probability, it is going to be eliminated and will not be picked up by the algorithm. By the union bound, we will then show that all of the ϵ-bad hypotheses are inconsistent with that training set.

Theorem. *Say algorithm A always finds hypothesis $h_A \in \mathcal{H}$ consistent with m examples where*

$$m \geq \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right)$$

then

$$Pr[err_D(h_A) > \epsilon] \leq \delta$$

The underlying assumption is that the hypothesis space is finite, *i.e.* $|\mathcal{H}| < \infty$ and that the m examples are i.i.d. with respect to the distribution $\mathcal{D}$. The theorem provides a upper bound on the amount of training data m needed to achieve a low error ϵ with a confidence of at least $1 - \delta$.

Proof. We aim to bound the probability that h_A is both consistent and ϵ-bad, *i.e.* the generalization error of h_A is greater than ϵ. Let $\mathcal{B} = \{h \in \mathcal{H} : h\ \epsilon\text{-bad}\}$ be the set of all ϵ-bad hypotheses in $\mathcal{H}$. (Here $\mathcal{B}$ is a fixed set and not a random variable. The concept c and distribution $\mathcal{D}$ are fixed, thus, the hypotheses h having error on $\mathcal{D}$ are fixed. The only random variable is h_A which depends on the sample. The consistency of h_A is also random.)

$$\begin{aligned}
&Pr[h_A \text{ is } \textit{consistent} \text{ and } \epsilon\text{-bad}] \\
&\quad \leq Pr[\exists h \in \mathcal{H} : h\ \textit{cons.}\ \&\ \epsilon\text{-bad}] && (\because \text{ if } A \Rightarrow B, \text{ then } Pr[A] \leq Pr[B]) \\
&\quad = Pr[\exists h \in \mathcal{B} : h\ \textit{cons.}] \\
&\quad = Pr\left[\bigvee_{h \in \mathcal{B}} (h\ \textit{cons.})\right] \\
&\quad \leq \sum_{h \in \mathcal{B}} Pr[h\ \textit{cons.}] && \text{(by union bound)} \\
&\quad = \sum_{h \in \mathcal{B}} Pr[h(x_1) = c(x_1) \wedge \ldots \wedge h(x_m) = c(x_m)] && \text{(by defn. of consistency)} \\
&\quad = \sum_{h \in \mathcal{B}} \prod_{i=1}^{m} Pr[h(x_i) = c(x_i)] && \text{(by independence)}
\end{aligned}$$

$$\begin{aligned}
&\leq \sum_{h \in \mathcal{B}} (1-\epsilon)^m && (\because h \in \mathcal{B} \Rightarrow Pr[h(x_i) \neq c(x_i)] \geq \epsilon) \\
&= |\mathcal{B}|(1-\epsilon)^m \\
&\leq |\mathcal{H}|(1-\epsilon)^m && (\because \mathcal{B} \subseteq \mathcal{H}) \\
&\leq |\mathcal{H}|e^{-\epsilon m} && (\because \forall x, (1+x) \leq e^x) \\
&\leq \delta && (\text{follows by choice of } m)
\end{aligned}$$

□

The negation of $Pr[\exists h \in \mathcal{H} : h \ cons. \ \& \ \epsilon\text{-bad}]$ leads us to conclude that with probability $\geq (1-\delta)$ and $\forall h \in \mathcal{H}$, if h is consistent, then

$$err_{\mathcal{D}}(h) \leq \epsilon = \frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{m} \tag{1}$$

Equation 1 captures the bound on the generalization error in terms of the learning performance δ, the size of the hypothesis space $|\mathcal{H}|$ and the number of training sample m.

An Alternate Proof

We can attempt to get rid of the dependence of the generalization error on the number of hypotheses $|\mathcal{H}|$ as follows

$$\begin{aligned}
&Pr[err_{\mathcal{D}}(h_A) > \epsilon \mid h_A \ cons.] \\
&\quad = \frac{Pr[h_A \ cons. \mid err(h_A) > \epsilon] \ Pr[err(h_A) > \epsilon]}{Pr[h_A \ cons.]} && (\text{by Bayes rule}) \\
&\quad = Pr[h_A \ cons. \mid err(h_A) > \epsilon] \ Pr[err(h_A) > \epsilon] && (\because Pr[h_A \ cons.] = 1) \\
&\quad \leq Pr[h_A \ cons. \mid err(h_A) > \epsilon] && (\because Pr[err(h_A) > \epsilon] \leq 1) \\
&\quad = Pr[h_A(x_1) = c(x_1) \wedge \ldots \wedge h_A(x_m) = c(x_m) \mid err(h_A) > \epsilon] && (\text{by defn. of consistency}) \\
&\quad = \prod_{i=1}^{m} Pr[h_A(x_i) = c(x_i) \mid err(h_A) > \epsilon] && (\text{by conditional independence}) \\
&\quad \leq (1-\epsilon)^m && (\because Pr[h(x_i) \neq c(x_i)] \geq \epsilon) \\
&\quad \leq e^{-\epsilon m} && (\because \forall x, (1+x) \leq e^x) \\
&\quad \leq \delta && (\text{if } m \geq \tfrac{ln\frac{1}{\delta}}{\epsilon})
\end{aligned}$$

The argument above seems plausible, but it is actually incorrect. In the first proof, h is not a random variable, since we had picked it before the sample $\mathcal{S}$ was picked, hence use of *independence* is valid in that case. However in this alternate proof, the hypothesis h_A is generated from the sample $\mathcal{S}$, and therefore is a random variable that depends on the sample $\mathcal{S}$. Since h_A depends on the sample $\mathcal{S}$, given h_A is ϵ-bad, the samples from $\mathcal{S}$ are no longer i.i.d. Thus, use of conditional independence in the above proof is incorrect, *i.e.*

$$Pr[h_A \ cons. \mid err(h_A) > \epsilon] \neq \prod_{i=1}^{m} Pr[h_A(x_i) = c(x_i) \mid err(h_A) > \epsilon]$$

Moreover, $Pr[h_A \; cons. \mid err(h_A) > \epsilon]$ should be 1, because we assume that h_A is always consistent. Therefore, care must be taken to pick the hypothesis (for which the generalization error is being analysed) before the sample space $\mathcal{S}$ is selected.

2 Consistency via PAC

In the previous section, we have seen that if we can learn in the consistency model, then we can learn in the PAC-model, provided $|\mathcal{H}|$ is not too huge. A concept class $\mathcal{C}$ is said to be *properly* PAC learnable by $\mathcal{H}$ if the hypotheses space $\mathcal{H}$ is the same as the concept class $\mathcal{C}$. We will now take the situation considering the case vice-versa.

Proposition. *Given*

- *algorithm A that properly PAC-learns $\mathcal{C}$,* i.e. *given a set of random examples, A finds a hypothesis $h \in \mathcal{C}$, such that with high probability, the hypothesis has generalization error at most ϵ.*
- *a sample $\mathcal{S} = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$*

we can use A as a subroutine to find $c \in \mathcal{C}$ consistent with $\mathcal{S}$ (if one exists).

Intuitively, since a PAC learning algorithm must have examples from a random distribution and $\mathcal{S}$ is *not* a random set, we construct a distribution for it and sample the examples (for feeding to algorithm A) from it. We then use algorithm A to get a hypothesis h such that $err_{\mathcal{D}}(h) \leq \epsilon$ and use it to show that h is consistent.

Proof. Given m examples $\mathcal{S}$, we construct a distribution $\mathcal{D}$ that is *uniform* over the m examples in $\mathcal{S}$. We choose $\epsilon = \frac{1}{2m}$ and any desired value of $\delta > 0$. We then run algorithm A on $m' = poly(\frac{1}{\epsilon}, \frac{1}{\delta})$ examples chosen from the distribution $\mathcal{D}$. Here m' is the number of examples required by A to attain the desired accuracy $(1 - \epsilon)$ with high probability $1 - \delta$. If A outputs the algorithm h, we check whether h is consistent with $\mathcal{S}$. If h is consistent with $\mathcal{S}$, then we output the hypothesis h (thus proving the proposition), else (or if A failed to generate a hypothesis), then we say "nothing consistent". Mathematically, if there exists $c \in \mathcal{C}$ consistent with $\mathcal{S}$, then with probability at least $(1 - \delta)$ (since A is PAC-learning algorithm), we have

$$\begin{aligned} err_{\mathcal{D}}(h) &\leq \epsilon \\ &= \frac{1}{2m} \\ &< \frac{1}{m} \end{aligned} \tag{2}$$

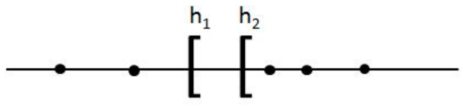
Since $\mathcal{D}$ is uniform, the probability assigned to each example is $\frac{1}{m}$ and therefore the generalization error is an integer multiple of $\frac{1}{m}$. By Equation 2, this leads to the conclusion that $err_{\mathcal{D}}(h) = 0$ and h is consistent. If, however, there does not exist a $c \in \mathcal{C}$ which is consistent, then algorithm A would fail somehow *i.e.* either give a hypothesis h which is inconsistent or terminate by saying that "nothing consistent". □

Consistency and PAC-learnability are closely related concepts.

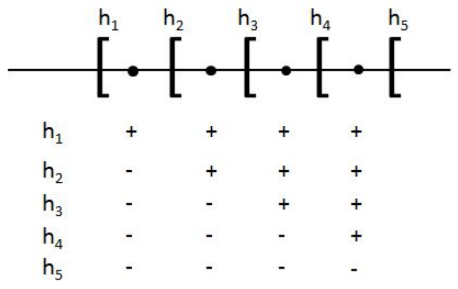
3 Learnability of Infinite Hypothesis Space

The result shown above holds only for the finite hypothesis spaces. There are still various examples, such as positive half-lines, rectangles, etc. that allow us to learn even though they have infinite hypothesis spaces. We will now discuss the characteristics of these hypothesis spaces to determine what makes a concept class $\mathcal{C}$ PAC-learnable.

Example 3.1. Positive Half-lines



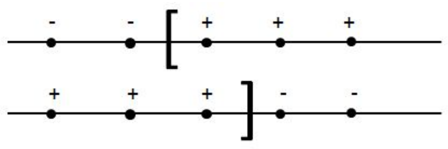
Given any unlabeled dataset of points on the x-axis, h_1 and h_2 behave exactly the same on the dataset. Although there are infinitely many hypotheses possible for this example, using the similarity of multiple hypotheses, we can divide the hypothesis into finite distinct equivalence classes. For instance, below are the 5 possible labelings/dichotomies/behaviors for a set of 4 unlabeled examples.



h_1	+	+	+	+
h_2	-	+	+	+
h_3	-	-	+	+
h_4	-	-	-	+
h_5	-	-	-	-

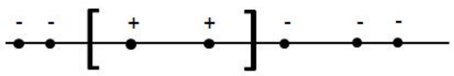
Therefore, in general, for m unlabeled examples we have $(m+1)$ possible equivalence classes compared to the 2^m possible labelings (there are infinitely many hypotheses but only finitely many labelings) for the unlabeled dataset $[m+1 \ll 2^m]$. The fact that the number of equivalence classes/labellings/dichotomies is so small, makes this concept class of positive half-lines PAC-learnable. Even though the hypothesis space is infinitely large, the *effective* hypothesis space is small $\mathcal{O}(m)$.

Example 3.2. Positive/Negative Half-lines



This case is similar to Example 3.1 except that the half-line can label the examples positive or negative on either side of the marker point. To compute the effective hypothesis space, we double the result of Example 3.1 and subtract 2 (to account for the double counting of all positive labels and all negative labels). Thus, the effective hypothesis space $= 2(m+1)-2 = 2m \equiv \mathcal{O}(m)$.

Example 3.3. Intervals
The concept class $\mathcal{C}$ consists of concepts which classify points, on the real axis, inside an interval (specific to every concept) as positive and those outside the interval as negative.



With m points, there are $(m+1)$ ways to place a marker for an interval boundary, thus the number of ways to select an interval, of this format, is $\binom{m+1}{2}$. To account for the empty interval (which can be placed between any two points), one extra value needs to be added, thus totaling the effective hypothesis cardinality to be $\binom{m+1}{2} + 1 \equiv \mathcal{O}(m^2)$. Alternatively, one could first pick pair of points $\binom{m}{2}$, then singleton points m and lastly the empty case 1, once again totaling to $\binom{m}{2} + m + 1$ which computes to be the same result.

Generalizing the characteristics of the aforementioned examples, we have a hypothesis space $\mathcal{H}$ and a set of unlabeled examples $\mathcal{S} = \langle x_1, x_2, \ldots, x_m \rangle$. We can thus define the set of all possible behaviors/dichotomies/labelings of $\mathcal{H}$ on $\mathcal{S}$ as

$$\Pi_{\mathcal{H}}(\mathcal{S}) = \{\langle h(x_1), \ldots, h(x_m) \rangle : h \in \mathcal{H}\}$$

Thus, for Example 3.1 we get $|\Pi_{\mathcal{H}}(\mathcal{S})| = 5$. We can now define a *growth function* over m samples to capture how complex the hypothesis space grows as we see more samples

$$\Pi_{\mathcal{H}}(m) = \max_{|\mathcal{S}|=m} |\Pi_{\mathcal{H}}(\mathcal{S})|$$

Intuitively, we would like to use $\Pi_{\mathcal{H}}(\mathcal{S})$ as an effective hypothesis space and thus replace $\ln |\mathcal{H}|$ in the error bound with $\ln |\Pi_{\mathcal{H}}(m)|$. The error bound depends on the growth function, which means that if the growth function grows as 2^m, *i.e.* $\forall m,\ \Pi_{\mathcal{H}}(m) = 2^m$, then we reduce the bound as $m \geq \frac{m + \ln \frac{1}{\delta}}{\epsilon}$, which is not useful. In this case, learning is impossible because we are working with something like all possible functions. However, we will see that the only other possible case for all hypothesis spaces is that the growth function grows as a polynomial in m, *i.e.* $\Pi_{\mathcal{H}}(m) = \mathcal{O}(m^d)$. Here d is called the *VC-dimension* and the error bound does not blow up

$$\lim_{m \to \infty} \frac{d \ln m + \ln \frac{1}{\delta}}{m} \longrightarrow 0$$

Thus we observe that for any $\mathcal{H}$, either $\Pi_{\mathcal{H}}(m) = 2^m$ for all m, or $\Pi_{\mathcal{H}}(m) = \mathcal{O}(m^d)$ for some constant d.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture # 5
Scribe: Yi-Hsien (Stephen) Lin — February 18, 2014

Recap

Last lecture, we proved Occam's Razor, which is that with probability at least $1-\delta$, $\forall h \in \mathcal{H}$, if h is consistent with all m examples that are sampled independently from distribution D, then the generalization error $err_D(h) \leq \frac{ln|\mathcal{H}|+\ln\frac{1}{\delta}}{m}$. However, this equation only applies to finite hypothesis spaces since we are using the cardinality of $\mathcal{H}$. This led us to briefly discuss about the generalization of Occam's Razor to infinite hypothesis spaces at the end of last week's lecture.

Sample Complexity for Infinite Hypothesis Space

In order to generalize Occam's Razor to infinite hypothesis spaces, we have to somewhat replace the $|\mathcal{H}|$. Here we first introduce some new concepts and notations which would simplify the later proof and discussion.

$$S = \langle x_1, \cdots, x_m\rangle \qquad \text{(sample set)}$$
$$\Pi_{\mathcal{H}}(S) = \{\langle h(x_1), \cdots, h(x_m)\rangle : h \in \mathcal{H}\} \qquad \text{(set of all possible dichotomies of } \mathcal{H} \text{ on } S)$$
$$\Pi_{\mathcal{H}}(m) = \max_{S:|S|=m} |\Pi_{\mathcal{H}}(S)| \qquad \text{(growth function)}$$

The growth function denotes the maximum number of distinct ways in which m points can be classified using hypotheses in $\mathcal{H}$, which provides another measure of the complexity of the hypothesis set $\mathcal{H}$. We will prove later that $\forall\mathcal{H}$ either:

- $\Pi_{\mathcal{H}}(m) = 2^m$ (impossible for PAC, can't get enough data)

  or
- $\Pi_{\mathcal{H}}(m) = \mathcal{O}(m^d)$ (possible for PAC)

Recall that our goal is to replace the cardinality of $|\mathcal{H}|$ in Occam's Razor. It is now clear that a growth function with a form similar to $\Pi_{\mathcal{H}}(m)$ is a good candidate. Therefore, our goal is to modify Occam's Razor to the following generalized version:

Theorem:
with probability at least $1-\delta$, $\forall h \in \mathcal{H}$, *if* h *is consistent with all* m *examples that are sampled independently from distribution* D*, then the generalization error* $err_D(h) \leq \mathcal{O}(\frac{\ln\Pi_{\mathcal{H}}(2m)+\ln\frac{1}{\delta}}{m})$.

Before the proof, we first introduce some definitions. Let D denote our target distribution, and $S = \langle x_1, \cdots, x_m\rangle$ denote a sample of $m > 8/\epsilon$ points chosen independently from D. We also introduce a "ghost sample" $S'\langle x'_1, \cdots, x'_m\rangle$ that consists of m points drawn i.i.d.

from D. By creating this "ghost sample", we are using the "double-sample trick" to take the mistakes on S' as a proxy for a hypothesis's generalization error. More importantly, doing so helps us avoid dealing with the potentially infinite space of instances, yet being able to make claims about a hypothesis. S' is called the "ghost sample" because it never actually exists and is not provided to the learning algorithm. We also define:

- $M(h, S) =$ number mistakes h makes on S
- $B \equiv [\exists h \in \mathcal{H} : (h \text{ is consistent on } S) \wedge (err_D(h) > \epsilon)]$
- $B' \equiv [\exists h \in \mathcal{H} : (h \text{ is consistent on } S) \wedge (M(h, S') \geq \frac{m\epsilon}{2})]$

Proof

Our goal is to prove that $Pr[B] \leq \delta$

Step 1: $Pr[B'|B] \geq 1/2$

In order to show this, suppose B holds, which is that there exists h consistent on S and $err_D(h) > \epsilon$. Since $err_D(h) > \epsilon$, the expectation value of $M(h, S')$, which is simply the number of examples times the probability of making an error would be at least $m\epsilon$. By Chernoff bounds (to be discussed later in the course) we can show that $Pr[M(h, S') < \frac{m\epsilon}{2}] \leq \frac{1}{2}$. Therefore, we can conclude that $Pr[B'|B] \geq 1/2$.

Step 2: $Pr[B] \leq 2Pr[B']$

From $A \wedge B \Rightarrow A$, we can show that:

$$\begin{aligned} Pr[B'] &\geq Pr[B \wedge B'] \\ &= Pr[B]Pr[B'|B] && \text{(by product rule)} \\ &\geq \frac{1}{2}Pr[B] && \text{(by step 1)} \end{aligned}$$

Now we have reduced the original problem to finding an upper bound for $Pr[B']$.

Now, consisder two experiments to generate S and S'
Experiment 1: Choose S, S' as usual (i.i.d. from D)
Experiment 2: First choose S, S' as usual (i.i.d. from D), but for $i \in \{1, 2, \cdots, m\}$ swap the example x_i in S with x'_i in S' with 0.5 probability and call the resulting samples as T and T'.

Notice that T, T' have the exact same distribution as S, S' since they are drawn from i.i.d., so experiment 1 and experiment 2 are actually identical. Also, we define:

- $B'' \equiv [\exists h \in \mathcal{H} : (h \text{ is consistent on } T) \wedge (M(h, T') \geq \frac{m\epsilon}{2})]$
  $\equiv [\exists h \in \mathcal{H} : (M(h, T) = 0) \wedge (M(h, T') \geq \frac{m\epsilon}{2})]$

Step 3: $Pr[B''] = Pr[B']$

Becuase the distributions for T, T' are exactly the same as those for S, S', $Pr[B''] = Pr[B']$.

Define $b(h) \equiv [h$ is consistent with T and $M(h, T') \geq \frac{m\epsilon}{2}]$

Step 4: $Pr[b(h)|S, S'] \leq 2^{-m\epsilon/2}$

Let us identify each example x in S and S' with a bit which is 0 if $h(x) = c(x)$ and 1 if $h(x) \neq c(x)$. In this step, we want to bound the probability of constructing a set T that is consist of only example 0's and T' that is consist of only example 1's given S and S' that is selected from the standard procedure (drawing i.i.d. from D). We denote this as $b(h)$:

$$b(h) \equiv (M(h, T) = 0) \wedge (M(h, T') \geq \frac{m\epsilon}{2})$$

Let r denote the number of pairs of points from S and S' that has exactly one 1 labeled. $Pr[b(h)|S, S'] \leq 2^{-m\epsilon/2}$ can then be shown by the following three cases:

Case 1: $\exists x_i, x'_i$ with both of them labeled as 1

In this case, no matter how the examples in S and S' are swapped by experiment 2, there will always be an error in T. Therefore, $Pr[M(h, T) = 0] = 0 \Rightarrow Pr[b(h)|S, S'] = 0$

$$\begin{array}{r|l} S & 1\,1\,0\,0\,1 \\ S' & 0\,1\,0\,1\,0 \end{array}$$

We can see from the above example that, no matter how the example in S are swapped with the example in S' below it, the minimum number of 1 labeled in S will be 1 since there are two 1's in colum 2.

$$\begin{array}{r|l} S & 0\,1\,0\,0\,0 \\ S' & 1\,1\,0\,1\,1 \end{array}$$

Case 2: $r < \frac{m\epsilon}{2}$

In this case $Pr[b(h)|S, S']$ is also 0. This is because in order for $b(h)$ to be true, all r errors have to occur in T' and the total number of errors labeled in T' have to exceed $\frac{m\epsilon}{2}$, which is impossible since there is only one error in each pair in r and $r < \frac{m\epsilon}{2}$.

Case 3: $r \geq \frac{m\epsilon}{2}$

Now, the total number of errors exceeds $\frac{m\epsilon}{2}$ so there is a probability that $b(h)$ is true. As mentioned above, experiment 2 would swap examples in S and S' with probability 0.5. Since these events are independent, $Pr[b(h)|S, S'] = (\frac{1}{2})^r \leq 2^{-m\epsilon/2}$.

Now, we can derive the bound of $Pr[b(h)|S, S']$ as follows:

$$Pr[b(h)|S, S'] \leq 2^{-m\epsilon/2}$$

Step 5: $Pr[B''|S,S'] \le \Pi_{\mathcal{H}}(2m)2^{-m\epsilon/2}$

Let $\mathcal{H}'$ denote the space of "representative" hypotheses for each labeling of S, S', which is finite. We can see that $|\mathcal{H}'| = |\Pi_{\mathcal{H}}(S,S')| \le \Pi_{\mathcal{H}}(2m)$.

We can now prove $Pr[B''|S,S'] \le \Pi_{\mathcal{H}}(2m)2^{\frac{-m\epsilon}{2}}$ as follows:

$$\begin{aligned}
Pr[B''|S,S'] &= Pr[\exists h \in \mathcal{H} : b(h)|S,S'] \\
&= Pr[\exists h \in \mathcal{H}' : b(h)|S,S'] \\
&\le \sum_{h \in \mathcal{H}'} Pr[b(h)|S,S'] && \text{(by union bound)} \\
&\le |\mathcal{H}'|2^{-m\epsilon/2} && \text{(from step4)} \\
&\le \Pi_{\mathcal{H}}(2m)2^{-m\epsilon/2}
\end{aligned}$$

Notice that the second step above is true because if $b(h)$ holds for some $h \in \mathcal{H}$, it will also hold for some $h \in \mathcal{H}'$ since they behave the same on S and S' ($b(h)$ only depends on S and S').

Step 6: $Pr[B''] \le \Pi_{\mathcal{H}}(2m)2^{-m\epsilon/2}$

By marginalization ($Pr[a] = \mathbb{E}_X[Pr[a|X]]$) we can show that:

$$\begin{aligned}
Pr[B''] &= \mathbb{E}_{S,S'}[Pr[B''|S,S']] \\
&\le \Pi_{\mathcal{H}}(2m)2^{-m\epsilon/2} && \text{(by marginalization)}
\end{aligned}$$

By the above six steps, we can finally show that:

$$\begin{aligned}
Pr[B''] &\le 2Pr[B'] = 2Pr[B''] \\
&\le 2\Pi_{\mathcal{H}}(2m)2^{-m\epsilon/2} \\
&\le \delta
\end{aligned}$$

Now, by solving the above inequality for ϵ, we can see that the inequality above holds when $\epsilon \le \frac{2}{m}(\lg \Pi_{\mathcal{H}}(2m) + \lg \frac{1}{\delta} + 1) = \mathcal{O}(\frac{\ln \Pi_{\mathcal{H}}(2m) + \ln \frac{1}{\delta}}{m})$, which is the error bound we are trying to prove for the generalized Occam's Razor.

By replacing $|\mathcal{H}|$ with the growth function $\Pi_{\mathcal{H}}(2m)$, we have now proved a bound on generalization in terms of the growth function. When the growth function has the form of $\mathcal{O}(m^d)$, we have a useful bound. We will later see when this form of growth function happens.

VC-Dimension

At the end of the class, we also briefly discussed the VC-dimension (Vapnik-Chervonenkis dimension). In order to define the VC-dimension of a hypothesis set $\mathcal{H}$, we first need to introduce the concept of "*shattering*". A set S of size m is *shattered* by $\mathcal{H}$ if all labelings of S can be realized by hypotheses in $\mathcal{H}$, that is when $|\Pi_{\mathcal{H}}(S)| = \Pi_{\mathcal{H}}(m) = 2^m$. And the VC-dimension of a hypothesis set $\mathcal{H}$ is the cardinality of the largest set that can be fully shattered by $\mathcal{H}$:

$$VCdim(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\}$$

We now look at an example of $\mathcal{H} = \{\text{intervals}\}$:

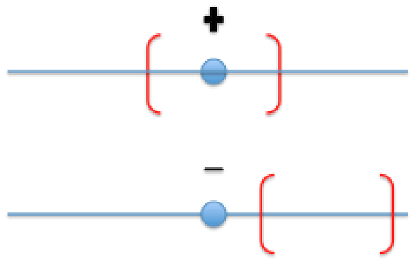


Figure 1: $\mathcal{H}$ contains hypotheses that produce evey possible labeling of the 1 point in S. Therefore, $\mathcal{H}$ shatters S, VCdim ≥ 1

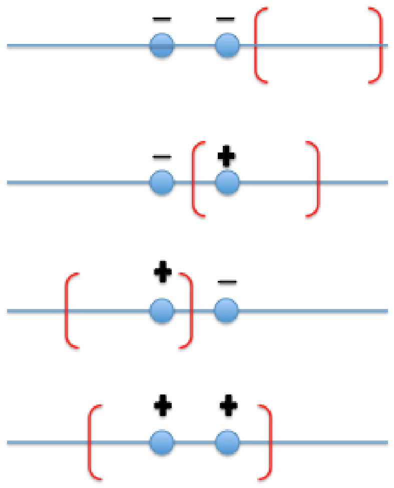


Figure 2: $\mathcal{H}$ contains hypotheses that produce evey possible labeling of the 2 points in S. Therefore, $\mathcal{H}$ shatters S, VCdim ≥ 2

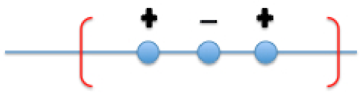


Figure 3: When S is a set of 3 points, $\mathcal{H}$ does not contain a hypothesis that can label this situation. Therefore, $\mathcal{H}$ does not shatter S, VCdim < 3

We can see from the example that $\mathcal{H}$ shatters S when S contains a single point and two points, but not three. Therefore, $\text{VCdim}(\mathcal{H}) = 2$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #6
Scribe: Zeyu Jin — February 20, 2014

1 VC Dimension

Last time we proved the theorem that with high probability $1-\delta$, the generalization error is given by

$$\mathrm{err}(h) \leq \mathrm{O}\left(\frac{\ln \Pi_{\mathcal{H}}(2m) + \ln 1/\delta}{m}\right) \tag{1}$$

where $\Pi_{\mathcal{H}}(m)$ is the growth function. We also defined the concept of shattering where S is shattered by $\mathcal{H}$ if $|\Pi_{\mathcal{H}}(S)| = 2^{|S|}$. Finally, we defined VC-dimension, or Vapnik-Chervonenkis dimension, as the cardinality of the largest shattered set. In this lecture, we are going to derive the bounds of the growth function, which is either $O(m^d)$ or 2^m.

1.1 Examples of VC-dimension

Here are some general results:

- VC-dim(intervals) $= 2$
- VC-dim(Axis-aligned rectangles) $= 4$
- VC-dim(Hyper-rectangles in $\mathbb{R}^n$) $= 2n$
- VC-dim(LTF in $\mathbb{R}^n$) $= n+1$

Note that LTF means linear threshold function (or perceptron), which is defined as a half space with parameters $\mathbf{w}$ where every points in this space is defined as "+". Formally,

$$c_w(x) = \begin{cases} 1 & \text{if } \mathbf{w}\cdot\mathbf{x} \geq b \\ 0 & \text{if } \mathbf{w}\cdot\mathbf{x} < b \end{cases} \tag{2}$$

The dot sign means inner product. If b is forced to be 0, the VC-dimension reduces to n. It is often the case that the VC-dimension is equal to the number of free parameters of a concept (for example, a rectangle's parameters are its topmost, bottommost, leftmost and rightmost bounds, and its VC-dimension is 4). However, it is not always true; there exists concepts with 1 parameter but an infinite VC-dimension.

There is also an inequality relationship between VC-dimension and the cardinality of $\mathcal{H}$. If the VC-dimension is d, then there exists a shattered set of size d on which $\mathcal{H}$ realizes all possible labelings. Because for every labeling there must be a corresponding hypothesis, we have $|\mathcal{H}| \geq 2^d$, which gives us:

$$\text{VC-dim}(\mathcal{H}) \leq \lg |\mathcal{H}| \tag{3}$$

1.2 Determining VC-dimension

In the last section, we claimed VC-dim(Axis-aligned rectangles) = 4. Now we show how to prove it. The proof involves two steps: first, we show the VC-dimension is at least 4 by showing that there exists a 4-point set shattered by the concept set (it's worth noting that not every 4-point configuration can be shattered, but we only need one to make the statement). Then, we show that there is no 5-point set that can be shattered.

Proof (1) An example 4-point set is shown in Figure 1 with all typical labelings and the corresponding realization. So we have VC-dim≥ 4.

(2) For any 5-point set, we can construct a data assignment in this way: pick the topmost, bottommost, leftmost and rightmost points and give them the label "+". Because there are 5 points, there must be at least one point left to which we assign "−". Any rectangle that contains all the "+" points must contains the "−" point, which is a case where shattering is not possible. This proves that VC-dim< 5.

In sum, VC-dim(axis aligned rectangle)$= 4$.

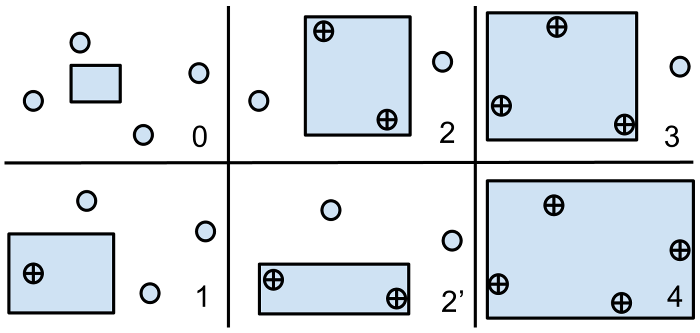


Figure 1: Proving that rectangle concept space shatters at least 4 points

2 Sauer's Lemma

Sauer's Lemma provides an upper bound for $\Pi_{\mathcal{H}}(m)$ parameterized by d, the VC-dimension of $\mathcal{H}$. It also leads to the proof that the growth function is either $O(m^d)$ or 2^m. In this section, we are going to use these definition and facts in binomial coefficients:

$$\binom{m}{k} = 0 \text{ if } k < 0 \text{ or } k > m \tag{4}$$

$$\binom{m}{k} = \binom{m-1}{k-1} + \binom{m-1}{k} \tag{5}$$

$$(a+b)^m = \sum_{k=0}^{m} \binom{m}{k} a^k b^{m-k} \tag{6}$$

Lemma 2.1 (Sauer's Lemma) *Let $\mathcal{H}$ be a hypothesis set with VC-dim($\mathcal{H}$) $= d$. Then, for all $m \in \mathcal{N}$, the following inequality holds*

$$\Pi_{\mathcal{H}}(m) \leq \Phi_d(m) \equiv \sum_{i=0}^{d} \binom{m}{i} \tag{7}$$

Proof The proof is by induction on $m + d$. The base cases are as follows:

- When $d = 0$, for any m points, there is only a single label possible for every point in the space. So in this case $\Pi_{\mathcal{H}}(m) = 1 = \binom{m}{0} = \Phi_0(m)$.

- When $m = 0$, for any d, there is only one labeling. So $\Pi_{\mathcal{H}}(0) = 1 = \sum_{i=0}^{d} \binom{0}{i} = \Phi_d(0)$

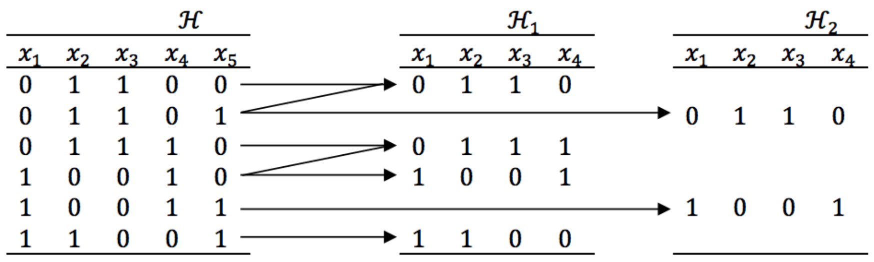
When $m \geq 1$ and $d \geq 1$, assume the lemma holds for any m' and d' if $m' + d' < m + d$. Suppose $S = \{x_1, ..., x_m\}$; we now prove $\Pi_{\mathcal{H}}(S) \leq \Phi_d(m)$. We start by creating two other hypothesis spaces: first, we construct $\mathcal{H}_1$ by restricting the set of concepts in $\mathcal{H}$ to the set $S' = \{x_1, ..., x_{m-1}\}$. Figure 2 shows an example of the construction: suppose there is a concept in $\mathcal{H}$ maps $(x_1, x_2, x_3, x_4, x_5)$ to $(0, 1, 1, 0, 1)$, by restricting this concept on the domain (x_1, x_2, x_3, x_4), we create a new concept in $\mathcal{H}_1$ that maps (x_1, x_2, x_3, x_4) to $(0, 1, 1, 0)$.

In this construction, some pairs of concepts may collapse into single concepts in $\mathcal{H}_1$. The second hypothesis space $\mathcal{H}_2$ is obtained by including all these collapsed concepts in constructing $\mathcal{H}_1$. As illustrated in Figure 2, when concept $(0, 1, 1, 0, 1)$ and $(0, 1, 1, 0, 0)$ both collapse into a concept $(0, 1, 1, 0)$ in $\mathcal{H}_1$, we add another copy of $(0, 1, 1, 0)$ to $\mathcal{H}_2$. Note that both $\mathcal{H}_1$ and $\mathcal{H}_2$ are both defined on the domain $(x_1, ..., x_{m-1})$.

We now derive bounds on the size of these two new hypothesis spaces.

- Any subset $T \subseteq S$ shattered by $\mathcal{H}_1$ is also shattered by $\mathcal{H}$. So VC-dim$(\mathcal{H}_1) \leq$ VC-dim $(\mathcal{H}) = d$. By inductive hypothesis, $|\mathcal{H}| = |\Pi_{\mathcal{H}_1}(S')| \leq \Phi_d(m-1)$

- Also notice that VC-dim$(\mathcal{H}_2) \leq d - 1$ since for any $T \subseteq S'$ shattered by $\mathcal{H}_2$, $T \cup \{x_m\}$ is shattered by $\mathcal{H}_1$. So $|\mathcal{H}_2| = |\Pi_{\mathcal{H}_2}(S')| \leq \Phi_{d-1}(m-1)$



x_1	x_2	x_3	x_4	x_5
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
1	0	0	1	0
1	0	0	1	1
1	1	0	0	1

x_1	x_2	x_3	x_4
0	1	1	0
0	1	1	1
1	0	0	1
1	1	0	0

x_1	x_2	x_3	x_4
0	1	1	0
1	0	0	1

Figure 2: Constructing $\mathcal{H}_1$ and $\mathcal{H}_2$ from $\mathcal{H}$: each table represents the content of hypothesis space; each row corresponds to a hypothesis and each row corresponds to a point x_i. The values are the labeling of a point given the row hypothesis. The arrow shows which hypothesis in $\mathcal{H}$ is used to construct a new hypothesis in $\mathcal{H}_1$ and $\mathcal{H}_2$.

In summary,

$$\begin{aligned}
|\Pi_{\mathcal{H}}(S)| &= |\mathcal{H}_1| + |\mathcal{H}_2| \le \Phi_d(m-1) + \Phi_{d-1}(m-1) \\
&= \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d-1} \binom{m-1}{i} \\
&= \sum_{i=0}^{d} \left[\binom{m-1}{i} + \binom{m-1}{i-1} \right] && \text{(Equation 4)} \\
&= \sum_{i=0}^{d} \binom{m}{i} && \text{(Equation 5)}
\end{aligned}$$

which completes the proof. □

Now we show an upper bound of $\Phi_d(m)$: if $m \ge d \ge 1$, then

$$\Phi_d(m) \le (\frac{em}{d})^d = O(m^d)$$

Proof According to the definition, $\Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}$. We multiply $(\frac{d}{m})^d$ on both sides and thus,

$$\begin{aligned}
\left(\frac{d}{m}\right)^d \Phi_d(m) &\le \sum_{i=0}^{d} \binom{m}{i} \left(\frac{d}{m}\right)^d \\
&\le \sum_{i=0}^{d} \binom{m}{i} \left(\frac{d}{m}\right)^i && (d/m \le 1) \\
&= \sum_{i=0}^{d} \binom{m}{i} \left(\frac{d}{m}\right)^i 1^{m-i} \\
&\le \sum_{i=0}^{m} \binom{m}{i} \left(\frac{d}{m}\right)^i 1^{m-i} && (d \le m) \\
&= \left(1 + \frac{d}{m}\right)^m \le e^d && \text{(Binomial theorem, Equation 6)}
\end{aligned}$$

It follows that $\Phi_d(m) \le (\frac{em}{d})^d$. □

Corollary 2.2 *Let $\mathcal{H}$ be a hypothesis space with VC-dim($\mathcal{H}$) $= d$. Then for all $m \ge d \ge 1$*

$$\Pi_{\mathcal{H}}(m) \le (\frac{em}{d})^d = O(m^d)$$

The proof directly follows from Sauer's lemma where $\Pi_{\mathcal{H}}(m) \le \Phi_d(m)$ and the fact we just proved. With this corollary, we can show that the growth function only exhibits two types of behavior: either VC-dim($\mathcal{H}$) $= d < \infty$, in which case $\Pi_{\mathcal{H}}(m) = O(m^d)$, or VC-dim($\mathcal{H}$) $= \infty$, in which case $\Pi_{\mathcal{H}}(m) = 2^m$ for all $m \ge 1$

Finally, we are able to express the generalization bound using VC-dimension:

Theorem 2.3 *Let $\mathcal{H}$ be a hypothesis space with VC-dim($\mathcal{H}$) $= d$. With probability at least $1-\delta$, for all $h \in \mathcal{H}$, if h is consistent with all m examples ($m \geq d \geq 1$) sampled independently from a distribution D, then the generalization error is*

$$err_D(h) \leq \frac{2}{m}\left(d \lg \frac{2em}{d} + \lg \frac{1}{\delta} + 1\right)$$

It directly follows from Corollary 2.2 and our earlier bound.

3 The lower bound?

VC-dimension also provides necessary conditions for learnability. In this sense, it is also possible to prove lower bounds on the number of examples needed to learn in the PAC model to a given accuracy. The difference is that instead of looking at the hypothesis space $\mathcal{H}$, we evaluate the VC dimension over the concept class $\mathcal{C}$.

Let's suppose we are working with concept class $\mathcal{C}$ and VC-dim($\mathcal{C}$) $= d$, which means there exists a set of size d, $\{z_1, ..., z_d\}$, shattered by concept class $\mathcal{C}$. A natural lower bound is d. The intuition is that even if we are given $z_1, ..., z_{d-1}$, we still lack the information conveyed by the last point; both labelings of the last point are still possible. To prove it rigorously, we need to go back to the definition of PAC learnable.

Claim 3.1 *For any algorithm A that PAC-learns the concept class $\mathcal{C}$, if given $d/2$ examples,then $err(h_A)$ has large generalization error with high probability.*

An incorrect proof is given below:

In the PAC learning setting, there is a target distribution D. To make things as bad as possible, we can choose whatever distribution we want. So we define D as a uniform distribution over the shattered set $z_1, ..., z_d$. Then we run some candidate algorithm A on $d/2$ samples chosen randomly from D. This algorithm will output hypothesis h_A. Pick $c \in \mathcal{C}$ which is consistent with the labels on the training set S. Let the remaining samples be labelled incorrectly, that is choose $c(x)$ so that $c(x) \neq h_A(x)$ for all $x \notin S$. Then err(h_A) is $1/2$ since h_A misclassifies at least half the points in the shattered set.

This is wrong because the proof cheats by choosing the concept c after the training samples are selected (we can do that for h but not for c). We will show a correct proof next time.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #7
Scribe: Huiwen Chang — February 25, 2014

1 A Lower Bound on Sample Complexity

In the last lecture, we proved an upper bound about how many examples are needed for PAC learning involving the VC dimension. Then we started talking about that the VC dimension also provides a lower bound for learning. While the upper bound gives the sufficient condition for PAC learning, the lower bound gives the necessary condition which says if the examples is too small, then the concept is not PAC learnable.

Last time, we gave a false proof of the lower bound. We generate a random training set, and then choose a concept from $\mathcal{C}$ which labels exactly the opposite to the prediction given by our hypothesis. This is cheating because the concept has to be chosen before the training set is generated. In the following, we give a correct proof of the lower bound:

Theorem 1. *Let $d =$ VC-dim$(\mathcal{C})$. For any algorithm A, there exists distribution $\mathcal{D}$ and a concept $c \in \mathcal{C}$, such that if A is given sample $\mathcal{S}$ of $m \leq d/2$ examples, then*

$$Pr[err(h_A) > \frac{1}{8}] \geq \frac{1}{8}$$

*An alternative(rougher) statement: if $\epsilon \leq 1/8$ and $\delta \leq 1/8$ then we need more than $d/2$ examples for PAC learning.

Proof. According to the definition of the VC dimension, we know that there exist d points $z_1, ..., z_d$ shattered by $\mathcal{C}$. Let D be uniform over $z_1, ..., z_d$. Let $\mathcal{C}' \subseteq \mathcal{C}$ have one representative for every labeling of the shattered points $z_1, ..., z_d$. Then we know $|\mathcal{C}'| = 2^d$. Then we choose c uniformly at random from $\mathcal{C}'$.

Let's think about two experiments about how to generate variables:

- Experiment 1:
  c is chosen uniformly at random from $\mathcal{C}'$.
  $\mathcal{S}$ is chosen at random (according to D) and labeled by c.
  h_A is computed from $\mathcal{S}$.
  The test point x is chosen (from D).
  We try to measure: $Pr[h_A(x) \neq c(x)]$.

- Experiment 2:
  Unlabeled part of $\mathcal{S}$ is chosen.
  Random labels $c(x_i)$ are chosen for $x_i \in \mathcal{S}$.
  h_A is computed from labeled $\mathcal{S}$.
  The test point x is chosen.
  If $x \notin \mathcal{S}$ then label $c(x)$ is chosen uniformly at random.
  We try to measure: $Pr[h_A(x) \neq c(x)]$.

Though the order is flipped in the two experiments above, we claim that they produce the same distribution of random variables and same probability measure. This is because the unlabeled sample S is generated independently of the choice of the labels, and the label for x is also chosen independently of the samples S, labels of other points, and the prediction of hypotheses. So in both experiments, the probability is given over random variables concept c, sample S, and the test point x. We denote it as $Pr_{c,S,x}[h_A(x) \neq c(x)]$. Let's work on experiment 2, and we have

$$\begin{aligned} & Pr_{c,S,x}[h_A(x) \neq c(x)] \\ \geq\ & Pr_{c,S,x}[x \notin \mathcal{S} \wedge h_A(x) \neq c(x)] \\ =\ & \underbrace{Pr_{c,S,x}[x \notin \mathcal{S}]}_{\geq 1/2 \texttt{ because } m \leq d/2 \texttt{ and } x \texttt{ is uniform chosen}} \cdot \underbrace{Pr_{c,S,x}[h_A(x) \neq c(x) | x \notin \mathcal{S}]}_{=1/2 \texttt{ because } c \texttt{ is a random guess}} \\ \geq\ & \frac{1}{4} \end{aligned}$$

According to marginalization $Pr[a] = E_X[Pr[a|X]]]$, we have

$$Pr_{c,S,x}[h_A(x) \neq c(x)] = E_c[Pr_{S,x}[h_A(x) \neq c(x)]]$$

By the fact that if $E[X] \geq b$, then there exists $x \in X$ such that $x \geq b$, we can know there exists $c \in \mathcal{C}' \subseteq \mathcal{C}$ such that

$$Pr_{S,x}[h_A(x) \neq c(x)] \geq \frac{1}{4}$$

Using marginalization again, we can get

$$\begin{aligned} \frac{1}{4} &\leq Pr_{S,x}[h_A(x) \neq c(x)] = E_S[Pr_x[h_A(x) \neq c(x)]] \\ &= E_S[err(h_A)] \\ &\leq Pr_S[err(h_A) \leq \frac{1}{8}] \cdot \frac{1}{8} + Pr_S[err(h_A) > \frac{1}{8}] \qquad (1) \\ &\leq \frac{1}{8} + Pr_S[err(h_A) > \frac{1}{8}] \end{aligned}$$

The inequality (1) comes as follows: for $X \in [0,1]$,

$$\begin{aligned} E[X] &= \sum_{x \in X} Pr(x) \cdot x \\ &= \underbrace{\sum_{x: x \leq 1/8} Pr(x)}_{Pr[X \leq 1/8]} \cdot \underbrace{x}_{\leq 1/8} + \underbrace{\sum_{x: x > 1/8} Pr(x)}_{Pr[X > 1/8]} \cdot \underbrace{x}_{\leq 1} \\ &\leq Pr[X \leq \frac{1}{8}] \cdot \frac{1}{8} + Pr[X > \frac{1}{8}] \end{aligned}$$

□

2 Inconsistent Model Hypotheses

So far we have only dealt with the situation in which the hypotheses is consistent, and we focused on the samples needed for learning in that space, but what to do if you cannot find a consistent hypotheses? There are several reasons it may not be consistent as follows:

- The concept $c \notin \mathcal{H}$;($\mathcal{H}$ is not powerful enough to represent the truth.)
- $c \in \mathcal{H}$, but it's just a too hard computational problem to find it;
- c may not exist. (We always assume that there's a target concept c that is a functional mapping that maps each instance to a label, but the reality is not that case. Consider weather prediction—the forecaster only estimates and reports the probability of snowing tomorrow. We believe there is an intrinsic randomness, or say it's too hard to *model* in a deterministic form by requiring so much knowledge.)

So now we work with a more realistic model where there might not exist the functional relationship. We can generalize our usual model: We assume we have examples (x, y) where $x \in X, y \in \{0, 1\}$. Now we let (x, y) be random according to distribution D on $X \times \{0, 1\}$. (Unlike our earlier model, the label y here is random). It follows from the chain rule that:

$$Pr_D[(x, y)] = Pr_D[x] \cdot Pr_D[y|x]$$

We can think the process as x being first generated by $Pr_D[x]$ and then y being generated according to its conditional distribution $Pr_D[y|x]$. In the PAC model where the label is deterministic, $Pr[y|x]$ is either 0 or 1, while in this new model, it's between 0 and 1. We also need to modify the generalization error from $err_D(h) = Pr_{x\sim D}[h(x) \neq c(x)]$ to

$$err_D(h) = Pr_{(x,y)\sim D}[h(x) \neq y]$$

Now, the first question is that, "With complete knowledge of distribution D, how small can the generalization error be?" Let's start with a simpler problem — tossing a coin with a known bias. The coin comes up heads with probability p. In this case, to minimize the probability of an incorrect prediction, our strategy is

$$\begin{cases} Head, & p > \frac{1}{2}; \\ Tail, & p < \frac{1}{2}; \\ arbitrary, & p = \frac{1}{2}. \end{cases}$$

Consider each x has a coin to flip, so the optimal decision rule is similar with before:

$$h_{opt}(x) = \begin{cases} 1, & Pr_D[y = 1|x] > \frac{1}{2}; \\ 0, & Pr_D[y = 1|x] < \frac{1}{2}; \end{cases}$$

The optimal prediction rule is called the "*Bayes Optimal Classifier*" or "*Bayes optimal decision rule*", and the optimal possible error $err_D(h_{opt}) = \min_h err_D(h)$ is called the "*Bayes error*". It provides a lower bound on the error over all hypotheses regardless of computational power.

Now, our goal is to minimize $err_D(h)$ over $h \in H$. We then introduce a natural approach: Given examples $(x_1, y_1), ..., (x_m, y_m)$ chosen independently at random from D, we try to minimize the *training error* with indicator variables(1 if $h(x_i) \neq y_i$):

$$\widehat{err}(h) = \frac{1}{m}\sum_{i=1}^{m} 1\{h(x_i) \neq y_i\} \tag{2}$$

So suppose you could find $\hat{h} = arg\min_{h\in H} \widehat{err}(h)$. We also suppose you could show that, with probability $1 - \delta$, for any $h \in \mathcal{H}$,

$$|\widehat{err}(h) - err_D(h)| \leq \epsilon \tag{3}$$

Then for all $h \in H$:

$$\begin{aligned} err_D(\widehat{h}) &\leq \widehat{err}(\widehat{h}) + \epsilon \\ &\leq \widehat{err}(h) + \epsilon \\ &\leq err_D(h) + 2\epsilon \end{aligned}$$

Therefore, the hypotheses $\widehat{h}$ will have a generalization error close to the lower bound of the error for all hypotheses in $\mathcal{H}$:

$$err_D(\widehat{h}) \leq \min_{h \in H} err_D(h) + 2\epsilon$$

But, this approach has two things to deal with:

- The computational problem about how to minimize the training error in (2);
- The statistical problem in (3) which implies the training error is a good approximation of true error for all hypotheses in $\mathcal{H}$.

The bound in (3) is also called a "*uniform convergence bound*". We also name $err(h)$ as *true/generalization error* or *true risk*, $\widehat{err}(h)$ as *training/empirical error* or *empirical risk*, and the approach of minimizing the training error as *empirical risk minimization*.

In order to prove a uniform convergence bound, we first move to a more abstract setting. Define random variables $X_1, ...X_m$, i.i.d $X_i \in [0, 1]$ for all $i = 1, \ldots, m$. Let $p = E[X_i]$, $\widehat{p} = \frac{1}{m} \sum_{i=1}^m X_i$. In fact, if we denote X_i to be $1\{h(x_i) \neq y_i\}$, we can see $\widehat{p}$ is the training error and p is the generalization error. We want to show how close $\widehat{p}$ is with respect to the mean p.

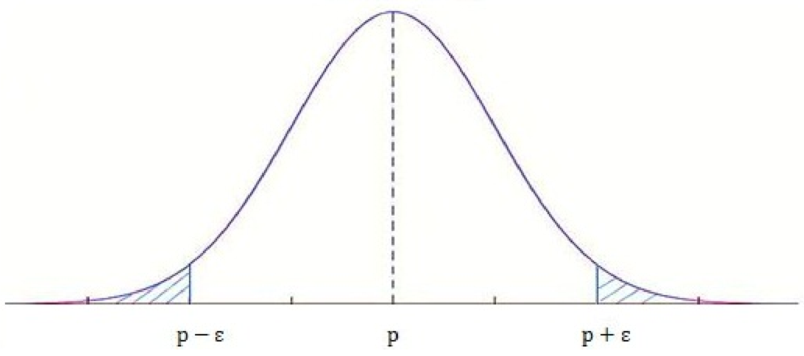


Figure 1: Illustration of concentration inequality or tail bound on $\widehat{p}$

Let's look at the distribution of $\widehat{p}$ in Figure 1, next time, we will show the tail $Pr[\widehat{p} > p + \varepsilon]$ and $Pr[\widehat{p} < p - \varepsilon]$ are really small, which is called "tail bounds" or "concentration inequalities". In the next lecture, we will provide a proof of a general bound – the Chernoff bound.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Ugne Klibaite

Lecture #8
February 27, 2013

1 Review From Last Time:

Last class, we discussed the relationship between training and generalization error, and how we could relate the two.

$$\begin{aligned}(x, y) &\sim D \\ err_D(h) &= Pr_{(x,y)\sim D}[h(x) \neq y] \\ e\hat{r}r(h) &= \frac{1}{m}\sum_{i=1}^{m} 1\{h(x_i) \neq y_i\}\end{aligned}$$

Here, err_D is our true or generalization error, which is the error over all samples in the distribution D. $e\hat{r}r(h)$, on the other hand, is the training error and is calculated by finding the number of incorrectly labeled examples in the training data. We are interested in relating the two in order to see how a given hypotheses will perform on unseen data after traning on a sample set.

If we specify that the training set is $(x_1, y_1), ...(x_m, y_m)$ and **if** we can show that with probability $\geq 1 - \delta$:

$$\begin{gathered}\forall h \in (H) : |err(h) - e\hat{r}r(h)| \leq \epsilon \\ \text{and } \textbf{if} \text{ we can find } \hat{h} = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\ e\hat{r}r(h) \\ \textbf{then}\text{: } err(\hat{h}) \leq \min_{h\in\mathcal{H}} err(h) + 2\epsilon\end{gathered}$$

If our assumptions hold then this means that if we can find the hypothesis in $\mathcal{H}$ that minimizes the training error, we can bound the true or generalization error of this hypothesis to the minimum true error over all hypotheses in $\mathcal{H}$ plus 2ϵ. In order to prove this we will use a special case of Chernoff bound, Hoeffding's inequality.

For now we are focusing on the statistical problem of showing that the training error is close to the generalization error without worrying about computation.

As mentioned at the end of last class, let's define the random variables $X_1, ..., X_m$, i.i.d. $X_i \in [0, 1]$ for all $i = 1, ..., m$:

$$\begin{aligned}p &= E[X_i] \\ \hat{p} &= \frac{1}{m}\sum_{i=1}^{m} X_i\end{aligned}$$

2 Hoeffding's Inequality

Hoeffding's Inequality states that:

$$Pr[\hat{p} \geq p + \epsilon] \leq e^{-2\epsilon^2 m} \tag{1}$$
$$Pr[\hat{p} \leq p - \epsilon] \leq e^{-2\epsilon^2 m} \tag{2}$$

Combining these two inequalities using the union bound:

$$Pr[|\hat{p} - p| > \epsilon] \leq 2e^{-2\epsilon^2 m} = \delta \tag{3}$$

Another way to say this is:
with probability $\geq 1 - \delta$,

$$|\hat{p} - p| = |e\hat{r}r(h) - err(h)| \leq \sqrt{\frac{\ln\left(\frac{2}{\delta}\right)}{2m}}$$

which means that the difference between training and generalization error decays by some constant over $\sqrt{m}$ as a function of the number of samples.

3 Chernoff Bounds

We will now prove a stronger form of Chernoff bounds, and Hoeffding's Inequality will follow as a corollary of this proof. This better bound says:

$$Pr[\hat{p} \geq p + \epsilon] \leq e^{-RE(p+\epsilon||p)m} \leq e^{-2\epsilon^2 m} \tag{4}$$
$$Pr[\hat{p} \leq p - \epsilon] \leq e^{-RE(p-\epsilon||p)m} \leq e^{-2\epsilon^2 m} \tag{5}$$

3.1 Relative Entropy

In order to understand Chernoff bounds we have to take a detour and first define the concept of relative entropy, also known as the Kullback-Leibler divergence.

The notion of KL divergence comes from information theory and can be illustrated using a simple example from class. Imagine that person A is trying to send a message to person B. Can we calculate the number of bits necessary to send this message? The obvious initial approach is to say that there are 26 letters in the alphabet and since k bits can describe 2^k different messages, we simply use $\log_2(26)$ bits. Rounding this up to 5, we can encode the alphabet with 5 bits where each letter takes exactly 5 bits (A = 0 0 0 0 0, B = 0 0 0 0 1, C = 0 0 0 1 0, ...).

However, not all letters are equally likely and we can come up with a more efficient encoding system where more likely letters take fewer bits and unlikely letters take more (A = 0 0, Q = 0 1 1 0 1 0 1, ...). This system would use less bits on average to send a message.

If letters are drawn from a probability distribution P, and $P(x)$ is the probability of sending a letter x, the optimal way of coding messages from P is to use $\log_2(\frac{1}{P(x)})$ bits for x. Then:

$$E[message\ length] = \sum_x P(x) \log_2 \frac{1}{P(x)}$$

This value is called the entropy of the distribution P and it is possible to show that this is maximized when P is a uniform distribution.

Now suppose that person A makes a mistake and uses the distribution Q to send the message where $\log_2(\frac{1}{Q(x)})$ bits are used. Now

$$\begin{aligned} E[message\ length] &= \sum_x P(x) \log_2 \frac{1}{Q(x)} \\ expected - optimal &= \sum_x P(x) \log_2 \frac{1}{Q(x)} - \sum_x P(x) \log_2 \frac{1}{P(x)} \\ &= \sum_x P(x) \log_2 (\frac{P(x)}{Q(x)}) \\ &= RE(P||Q) \end{aligned}$$

Note: we will be using natural log base instead of base 2.
RE between two numbers will be written with the shorthand:

$$RE(p||q) = p \ln \frac{p}{q} + (1-p) \ln(\frac{1-p}{1-q})$$

3.2 Markov's Inequality

Markov's inequality is a simple inequality used to compare the expected value of a random variable to the probability of that variable being very large in relation to the expected value.

Say $X \geq 0$, Markov's Inequality states:

$$Pr[X] \geq t] \leq \frac{E[X]}{t}$$

Proof:

$$\begin{aligned} E[X] &= Pr[X \geq t] \cdot E[X | X \geq t] + Pr[X < t] \cdot E[X | X < t] \\ &\geq Pr[X \geq t] \cdot t \end{aligned}$$

3.3 Proof

We are now ready to prove equation 4. As a first attempt, we let $q = p + \epsilon$, where p and ϵ, and therefore q are fixed. We are trying to upper bound $Pr[\hat{p} \geq q]$. Using Markov's inequality we show:

$$Pr[\hat{p} \geq q] \leq \frac{E[\hat{p}]}{q} = \frac{p}{q} = \frac{p}{p+\epsilon} \tag{6}$$

This is equal to less than one but is very weak and doesn't have the dependence on m we are looking for. Next, we try a clever trick to make this happen and pass both sides of our original inequality through a monotonically increasing function to get an equivalent inequality to work with. This is possible because if f is strictly increasing then $\hat{p} \geq q \Leftrightarrow f(\hat{p}) \geq f(q)$.

We will use the form $f(x) = e^{\lambda m x}$ where $\lambda > 0$. Our new inequality becomes:

$$\begin{aligned}
Pr[\hat{p} \geq q] &= Pr[e^{\lambda m \hat{p}} \geq e^{\lambda m q}] \\
&\leq \frac{E[e^{\lambda m \hat{p}}]}{e^{\lambda m q}} && \text{Markov's Ineq.} \\
&= e^{-\lambda m q} \cdot E[exp(\lambda \sum_{i=1}^{m} X_i)] \\
&= e^{-\lambda m q} \cdot \prod_{i=1}^{m} E[e^{\lambda X_i}] \\
&\leq e^{-\lambda m q} \cdot \prod_{i=1}^{m} E[(1 - X_i) + e^{\lambda} X_i] && \text{bound } e^{\lambda x} \text{ by a line} \\
&= e^{-\lambda m q} \cdot \prod_{i=1}^{m} [1 - p + e^{\lambda} p] \\
&= e^{-\lambda m q} \cdot [1 - p + e^{\lambda} p]^m \\
&= (e^{-\lambda q}[1 - p + e^{\lambda} p])^m \\
&= e^{\phi(\lambda) m}
\end{aligned}$$

We set $\phi(\lambda) = \ln[e^{-\lambda q} \cdot (1 - p + e^{\lambda} p)]$ to get our final result.

Minimizing over λ we find:

$$\begin{aligned}
\lambda_{min} &= \ln\left(\frac{q(1-p)}{(1-q)p}\right) \\
\phi(\lambda_{min}) &= -RE(q||p)
\end{aligned}$$

This is true for all λ, true for λ_{min}, and gives the earlier bound.

4 McDiarmid's Inequality

Hoeffding's inequality shows that the average of a set of random variables is connected to the expected value of individual random variables. Hoeffding's Inequality as a corollary of the bound involving relative entropy can be proven using Taylor's theorem.

$$\frac{1}{m}\sum_{i=1}^{m} x_i \rightarrow E[\frac{1}{m}\sum_{i=1}^{m} x_i] AVG(x_1, ..., x_m) \rightarrow E[AVG(x_1, ..., x_m)] \tag{7}$$

This tells us the rate at which the average converges to the expected value. What other functions could the average be replaced with and still give these results? Whenever f has the property that changing one argument does not change f by a lot, that is:

$$\forall(x_1, ..., x_m, x_i') : |f(x_1, ..., x_i, ..., x_m) - f(x_1, ..., x_i', ..., x_m)| \leq c_i.$$

Let $X_1, ..., X_m$ be independent, but not necessarily identical. Then:

$$Pr[f(X_1, ..., X_m) \geq E[f(X_1, ..., X_m)] + \epsilon] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^m c_i^2}\right)$$

$c_i = \frac{1}{m}$ for AVG, and we get Hoeffding's inequality for this special case.

4.1 Putting it Together

Theorem: Given $m = O(\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{\epsilon^2})$ examples, then with probability $\geq 1 - \delta$,

$$\forall h \in \mathcal{H} : |err(h) - e\hat{r}r(h)| \leq \epsilon$$

Proof: for any particular $h \in \mathcal{H}$, we have already argued that

$$Pr[|err(h) - e\hat{r}r(h)| > \epsilon] \leq 2e^{-2m\epsilon^2}$$
$$Pr[\exists h \in \mathcal{H} : |err(h) - e\hat{r}r(h)| > \epsilon] \leq 2|\mathcal{H}|e^{-2m\epsilon^2} \leq \delta \qquad \text{for given } m$$

We can take these bounds and rewrite them by solving for ϵ, where:

$$|err(h) - e\hat{r}r(h)| \leq O\left(\sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{m}}\right) \tag{8}$$

This combines the number of training examples, complexity of hypotheses, and how well the hypothesis fits the training samples.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #9
Scribe: Bebe Shi — March 4, 2013

1 Techniques that Handle Overfitting

- Cross Validation:
  Hold out part of the training data and use it as a proxy for the generalization error
  Disadvatages: 1. Wastes data. 2. Time-consuming because a lot of the variants of cross validation involve doing multiple splits on data for training and validation and running the algorithm multiple times.

- Structural Risk Minimization:
  Earlier, we found an upper bound on the generalization error in the following form.
  Under usual assumptions, with probability at least 1 - δ, $\forall h \in \mathcal{H}$ and $|\mathcal{H}| < \infty$,
  $err(h) \leq \hat{err}(h) + O\sqrt{\frac{ln|\mathcal{H}|+ln(\frac{1}{\delta})}{m}}$
  This technique tries to minimize the entire right-hand side of the inequality.

- Regularization
  This general family of techniques is closely related to structural risk minimization.
  It minimizes expressions of the form $\hat{err}$ + constant $\times$ "complexity"

- Algorithms that tend to resist overfitting

2 Rademacher Complexity

We have already learned about using the growth function and VC-dimesion as complexity measures for infinite hypothesis spaces. Today, we are going to introduce a more modern and elegant complexity measure called the Rademacher complexity. This technique subsumes the previous techniques in the sense that the previous bounds we found using $|\mathcal{H}|$, the growth function or the VC-dimesion would fall out as special cases of the new measure.

2.1

We start by laying down the setups of Rademacher complexity.

Sample $\mathcal{S} = \langle (x_1, y_1), ..., (x_m, y_m) \rangle$, $y_i \in \{-1, 1\}$. We are using $\{-1, 1\}$ here instead of $\{0, 1\}$, in order to make the math come out nicer.

hypothesis $h : X \rightarrow \{-1, 1\}$
Here, we're providing an alternative definition for training error.

$$\hat{err}(h) = \frac{1}{m} \sum_{i=1}^{m} 1\{h(x_i) \neq y_i\} \tag{1}$$

$$\hat{err}(h) = \frac{1}{m}\sum_{i=1}^{m}\frac{1-y_ih(x_i)}{2} = \frac{1}{2} - \frac{1}{2m}\sum_{i=1}^{m} y_ih(x_i) \tag{2}$$

Equation (2) is reached because $y_ih(x_i)$ equals 1 when $y_i = h(x_i)$ and $y_ih(x_i)$ equals -1 when $y_i \neq h(x_i)$.

$$\frac{1}{m}\sum_{i=1}^{m} y_ih(x_i) = 1 - 2\hat{err}(h) \tag{3}$$

Training error is a reasonable measure of how well a single hypothesis fits the data set. From equation (3), we can see that in order to minimize the training error, we can simply maximize $\frac{1}{m}\sum\limits_{i=1}^{m} y_ih(x_i)$.

2.2

Now, let us introduce a random label for data i, which we name σ_i and which is also known as a Rademacher random variable.

$$\sigma_i = \begin{cases} -1, & \text{with probability } 1/2. \\ +1, & \text{with probability } 1/2. \end{cases} \tag{4}$$

We can use this random label to form a complexity measure for $\mathcal{H}$ that is independent of the real labels of $\mathcal{S}$.

$$E_\sigma[\max_{h\in\mathcal{H}} \frac{1}{m}\sum_i \sigma_ih(x_i)] \tag{5}$$

Equation (5) intuitively measures the complexity of $\mathcal{H}$. Notice that we can find the range of this measure using two extreme cases.

- $\mathcal{H} = \{h_0\}$: because there is only one hypothesis, max is not used. We then arrive at the expectation of 0.
- $\mathcal{S}$ is shattered by $\mathcal{H}$: In this case, we can always find a hypothesis that matches all σ_i. Thus, the expected value is 1.

We now know that this measure ranges from 0 to 1.

2.3

We now replace $\mathcal{H}$ with $\mathcal{F}$, a family of functions $f\colon \mathcal{Z} \to \mathcal{R}$. This generalizes our hypotheses to real-valued functions.

Sample $\mathcal{S} = \langle z_1, ..., z_m \rangle$, $z_i \in \mathcal{Z}$.

The definition for the *empirical Rademacher complexity* is

$$\hat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = E_\sigma[\sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i f(z_i)] \tag{6}$$

Notice we replaced max by sup (supremum) because max might not exist when taken over an infinite number of functions. Supremum takes the least upper bound. For example, $\sup\{.9, .99, .999, ...\} = 1$.

In order to find a measure with respect to the distribution $\mathcal{D}$ over $\mathcal{Z}$, we take the expected value of the *empirical Rademacher complexity* and arrive at the definition for the *expected Rademacher complexity*, i.e., *Rademacher complexity* — equation (7).

$$\mathcal{R}_m(\mathcal{F}) = E_s[\hat{\mathcal{R}}_s(\mathcal{F})] \tag{7}$$

$\mathcal{S} = \langle z_1, ..., z_m \rangle$, $z_i \sim \mathcal{D}$

3 Generalization Bounds Based on Rademacher Complexity

Theorem

Let $\mathcal{F}$ be a family of functions $f : \mathcal{Z} \to [0, 1]$. Assume $\mathcal{S} = \langle z_1, ..., z_m \rangle$, i.i.d and $z_i \sim \mathcal{D}$. Define $\hat{E_{\mathcal{S}}}[f] = \frac{1}{m} \sum\limits_i f(z_i)$, $\mathrm{E}[f] = E_{z \sim \mathcal{D}}[f(z)]$. ($\hat{E_{\mathcal{S}}}[f]$ is similar to the idea of the training error and $\mathrm{E}[f]$ is similar to the idea of the generalization error)

With probability at least $1 - \delta$, $\forall f \in \mathcal{F}$,

$$E[f] \leq \hat{E_{\mathcal{S}}}[f] + 2\mathcal{R}_m(\mathcal{F}) + O\sqrt{\frac{ln(\frac{1}{\delta})}{m}} \tag{8}$$

$$E[f] \leq \hat{E_{\mathcal{S}}}[f] + 2\hat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) + O\sqrt{\frac{ln(\frac{1}{\delta})}{m}} \tag{9}$$

Proof

We want to bound the following random variable:

$$\Phi(\mathcal{S}) = \sup_{f \in \mathcal{F}}(E[f] - \hat{E_{\mathcal{S}}}[f]) \tag{10}$$

Step 1

Using the definitions, we get:

$$\Phi(\mathcal{S}) = \sup_{f \in \mathcal{F}}(E[f] - \hat{E_{\mathcal{S}}}[f]) = \sup_{f \in \mathcal{F}}(E[f] - \frac{1}{m}\sum_i f(z_i)) \tag{11}$$

Since $f(z_i) \in [0,1]$, changing any z_i value to z_i' can only change $\frac{1}{m}\sum\limits_i f(z_i))$ by at most $\frac{1}{m}$, and therefore $\Phi(\mathcal{S})$ by at most $\frac{1}{m}$. This means that $\Phi(\mathcal{S})$ satisfies the condition for McDiarmid's inequality, in that $|\Phi(z_1, ..., z_i, ..., z_m) - \Phi(z_1, ..., z_i', ..., z_m)| \le c_i$, where $c_i = \frac{1}{m}$.

McDiarmid's inequality states that with probability at least $1 - \delta$
$Pr[f(x_1, ..., x_m) - E[f(X_1, ..., X_m)] \ge \epsilon] \le \exp(\frac{-2\epsilon^2}{\sum_{i=1}^m c_i^2})$

Applying McDiarmid's inequality, we get:
With probability at least $1 - \delta$

$$\Phi(\mathcal{S}) \le E_{\mathcal{S}}[\Phi(\mathcal{S})] + \sqrt{\frac{ln(\frac{1}{\delta})}{2m}} \tag{12}$$

Step 2

Let us define a ghost sample $\mathcal{S}' = \langle z_1', ..., z_m' \rangle$, $z_i' \sim \mathcal{D}$. We aim to show that $E[\Phi(\mathcal{S})] \le E_{\mathcal{S},\mathcal{S}'}[\sup\limits_{f \in \mathcal{F}}(\hat{E_{\mathcal{S}'}}[f] - \hat{E_{\mathcal{S}}}[f])]$.

$$E_{\mathcal{S}'}[\hat{E_{\mathcal{S}'}}[f]] = E[f] \tag{13}$$

Equation (13) is true because the expected value of the random variable $\hat{E_{\mathcal{S}'}}[f]$ over all samples $\mathcal{S}'$ is $E[f]$.

$$E_{\mathcal{S}'}[\hat{E_{\mathcal{S}}}[f]] = \hat{E_{\mathcal{S}}}[f] \tag{14}$$

Equation (14) is true because the random variable $\hat{E_{\mathcal{S}}}[f]$ is independent of $\mathcal{S}'$.

Therefore,

$$\begin{aligned} E[\Phi(\mathcal{S})] &= E_{\mathcal{S}}[\sup_{f \in \mathcal{F}}(E[f] - \hat{E_{\mathcal{S}}}[f])] \\ &= E_{\mathcal{S}}[\sup_{f \in \mathcal{F}}(E_{\mathcal{S}'}[\hat{E_{\mathcal{S}'}}[f] - \hat{E_{\mathcal{S}}}[f]])] \\ &\le E_{\mathcal{S},\mathcal{S}'}[\sup_{f \in \mathcal{F}}(\hat{E_{\mathcal{S}'}}[f] - \hat{E_{\mathcal{S}}}[f])] \end{aligned}$$

The last inequality is true because the expected value of the max of some function is at least the max of the expected value of the function.

Step 3

Continuing the ghost sampling technique, we now try to obtain two new samples $\mathcal{T}$ and $\mathcal{T}'$ by running through the following mechanism on $\mathcal{S}$ and $\mathcal{S}'$.

```
for $i = 1, ..., m$
    with probability 1/2: swap $z_i$, $z'_i$
    else: leave alone
$\mathcal{T}$, $\mathcal{T}'$ = resulting samples
```

$$\hat{E_{\mathcal{T}'}}[f] - \hat{E_{\mathcal{T}}}[f] = \frac{1}{m}\sum_i \begin{cases} (f(z_i) - f(z'_i)), & \text{with probability } 1/2 \\ (f(z'_i) - f(z_i)), & \text{with probability } 1/2. \end{cases} \tag{15}$$

$\implies$

$$\hat{E_{\mathcal{T}'}}[f] - \hat{E_{\mathcal{T}}}[f] = \frac{1}{m}\sum_i \sigma_i(f(z'_i) - f(z_i)) \tag{16}$$

We know that $\mathcal{T}$, $\mathcal{T}' \sim \mathcal{S}$, $\mathcal{S}'$ (equally distributed) because $\mathcal{S}$, $\mathcal{S}'$ are i.i.d samples from the distribution $\mathcal{D}$.
Therefore, $\sup\limits_{f\in\mathcal{F}}(\hat{E_{\mathcal{S}'}}[f] - \hat{E_{\mathcal{S}}}[f]) \sim \sup\limits_{f\in\mathcal{F}}(\frac{1}{m}\sum\limits_i \sigma_i(f(z'_i) - f(z_i)))$.
Then, if we take the expected values of the two expressions over $\mathcal{S}$, $\mathcal{S}'$ and σ_i, the values should equal to each other.

Equation (17) shows the conclusion for step 3.

$$E_{\mathcal{S},\mathcal{S}'}[\sup_{f\in\mathcal{F}}(\hat{E_{\mathcal{S}'}}[f] - \hat{E_{\mathcal{S}}}[f])] = E_{\mathcal{S},\mathcal{S}',\sigma}[\sup_{f\in\mathcal{F}}(\frac{1}{m}\sum_i \sigma_i(f(z'_i) - f(z_i)))] \tag{17}$$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: José Simões Ferreira

Lecture #10
March 06, 2013

In the last lecture the concept of Rademacher complexity was introduced, with the goal of showing that for all f in a family of functions $\mathcal{F}$ we have $\hat{\mathrm{E}}_S[f] \approx \mathrm{E}[f]$. Let us summarize the definitions of interest:

$$\mathcal{F} \text{ family of functions } f : Z \to [0,1]$$

$$S = \langle z_1, \ldots, z_m \rangle$$

$$\hat{R}_S(\mathcal{F}) = \mathrm{E}_\sigma \left[\sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^m \sigma_i f(z_i) \right]$$

$$R_m(\mathcal{F}) = \mathrm{E}_S \left[\hat{R}_S(\mathcal{F}) \right].$$

We also began proving the following theorem:

Theorem. *With probability at least* $1-\delta$ *and* $\forall f \in \mathcal{F}$*:*

$$E[f] \le \hat{E}_S[f] + 2R_m(\mathcal{F}) + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right)$$

$$E[f] \le \hat{E}_S[f] + 2\hat{R}_S(\mathcal{F}) + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

Which we now prove in full.

Proof. Let us define:

$$\Phi(S) = \sup_{f \in \mathcal{F}} \left(\mathrm{E}[f] - \hat{\mathrm{E}}_S[f] \right)$$

$$\mathrm{E}[f] = \mathrm{E}_{z \sim D}[f(z)]$$

$$\hat{\mathrm{E}}_S[f] = \frac{1}{m} \sum_{i=1}^m f(z_i)$$

Step 1 $\Phi(S) \le \mathrm{E}_S[\Phi(S)] + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right)$

This was proven last lecture and follows from McDiarmid's inequality.

Step 2 $\mathrm{E}_S[\Phi(S)] \le \mathrm{E}_{S,S'}\left[\sup_{f\in\mathcal{F}}\left(\hat{\mathrm{E}}_{S'}[f] - \hat{\mathrm{E}}_S[f]\right)\right]$

This was also shown last lecture. We also considered generating new samples T, T' by flipping a coin, i.e. running through $i = 1, \ldots, m$ we flip a coin, swapping z_i with z_i' if heads, and doing nothing otherwise. We then claimed that the distributions thus generated are distributed the same as S and S', and we noted

$$\hat{\mathrm{E}}_{S'}[f] - \hat{\mathrm{E}}_S[f] = \frac{1}{m} \sum_i \left(f(z_i') - f(z_i) \right)$$

which means we can write

$$\hat{\mathrm{E}}_{T'}\left[f\right] - \hat{\mathrm{E}}_{T}\left[f\right] = \frac{1}{m}\sum_i \sigma_i \left(f(z_i') - f(z_i)\right)$$

which is written in terms of Rademacher random variables, σ_i. We now proceed with the proof.

Step 3 We first claim

$$\mathrm{E}_{S,S'}\left[\sup_{f\in\mathcal{F}}\left(\hat{\mathrm{E}}_{S'}\left[f\right] - \hat{\mathrm{E}}_{S}\left[f\right]\right)\right] = \mathrm{E}_{S,S',\sigma}\left[\sup_{f\in\mathcal{F}}\left(\frac{1}{m}\sum_i \left(f(z_i') - f(z_i)\right)\sigma_i\right)\right].$$

To see this, note that the right hand side is effectively the same expectation as the left hand side, but with respect to T and T', which are identically distributed to S and S'. Now we can write

$$\begin{aligned}\mathrm{E}_{S,S',\sigma}\left[\sup_{f\in\mathcal{F}}\left(\frac{1}{m}\sum_i \left(f(z_i') - f(z_i)\right)\sigma_i\right)\right] \le\ & \mathrm{E}_{S,S',\sigma}\left[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_i \sigma_i f(z_i')\right] \\ &+ \mathrm{E}_{S,S',\sigma}\left[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_i (-\sigma_i) f(z_i)\right]\end{aligned}$$

where we are just maximizing over the sums separately. We now note two points:

1. The random variable $-\sigma_i$ has the same distribution as σ_i;
2. The expectation over S is irrelevant in the first term, since the term inside the expectation does not depend on S. Similarly, the expectation over S' is irrelevant in the second term.

Therefore

$$\begin{aligned}\mathrm{E}_{S,S',\sigma}\left[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_i \sigma_i f(z_i')\right] &= \mathrm{E}_{S',\sigma}\left[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_i \sigma_i f(z_i')\right] \\ &= \mathrm{E}_{S'}\left[\mathrm{E}_{\sigma}\left[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_i \sigma_i f(z_i')\right]\right] \\ &= R_m(\mathcal{F})\end{aligned}$$

and, similarly

$$\mathrm{E}_{S,S',\sigma}\left[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_i (-\sigma_i) f(z_i)\right] = R_m(\mathcal{F}).$$

Step 4 We have thus shown

$$\mathrm{E}_{S,S'}\left[\sup_{f\in\mathcal{F}}\left(\hat{\mathrm{E}}_{S'}[f]-\hat{\mathrm{E}}_S[f]\right)\right]\leq 2R_m(\mathcal{F}).$$

Chaining our results together, we obtain

$$\Phi(S)=\sup_{f\in\mathcal{F}}\left(\mathrm{E}[f]-\hat{\mathrm{E}}_S[f]\right)\leq 2R_m(\mathcal{F})+\mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

We conclude that with probability at least $1-\delta$ and $\forall f\in\mathcal{F}$

$$\mathrm{E}[f]-\hat{\mathrm{E}}_S[f]\leq\Phi(S)\leq 2R_m(\mathcal{F})+\mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

Therefore, with probability at least $1-\delta$ and $\forall f\in\mathcal{F}$

$$\mathrm{E}[f]\leq\hat{\mathrm{E}}_S[f]+2R_m(\mathcal{F})+\mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

This is one of the results we were seeking. Proving the result for $\hat{R}_S(\mathcal{F})$ is just a matter of applying McDiarmid's inequality to obtain, with probability at least $1-\delta$

$$\hat{R}_S(\mathcal{F})\leq R_m(\mathcal{F})+\mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

$\square$

1 Motivation

The original motivation behind the theorem above was to obtain a relationship between generalization error and training error. We want to be able to say that, with probability at least $1-\delta$, $\forall h\in\mathcal{H}$

$$err(h)\leq\hat{err}(h)+\text{ small term.}$$

We note that $err(h)$ is evocative of $\mathrm{E}[f]$ and $\hat{err}(h)$ is evocative of $\hat{\mathrm{E}}_S[f]$, which appear in our theorem. Let us write

$$err(h)=\Pr_{(x,y)\sim D}[h(x)\neq y]=\mathrm{E}_{(x,y)\sim D}[\mathbf{1}\{h(x)\neq y\}]$$

$$\hat{err}(h)=\frac{1}{m}\sum_i\mathbf{1}\{h(x_i)\neq y_i\}=\hat{\mathrm{E}}_S[\mathbf{1}\{h(x)\neq y\}]$$

as per our definitions. We see that, to fit our definition, we must work with functions f which are indicator functions. Let us define

$$Z=X\times\{-1,+1\}$$

and for $h\in\mathcal{H}$:

$$f_h(x,y)=\mathbf{1}\{h(x)\neq y\}.$$

Now we can write:

$$\mathrm{E}_{(x,y)\sim D}\left[\mathbf{1}\{h(x)\neq y\}\right] = \mathrm{E}\left[f_h\right]$$

$$\hat{\mathrm{E}}_S\left[\mathbf{1}\{h(x)\neq y\}\right] = \hat{\mathrm{E}}_S\left[f_h\right]$$

$$\mathcal{F}_{\mathcal{H}} = \{f_h : h\in\mathcal{H}\}.$$

This allows us to use our theorem to state that:

With probability $\geq 1-\delta$
$\forall h\in\mathcal{H}$

$$err(h) \leq e\hat{r}r(h) + 2R_m(\mathcal{F}_{\mathcal{H}}) + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right)$$

$$err(h) \leq e\hat{r}r(h) + 2\hat{R}_S(\mathcal{F}_{\mathcal{H}}) + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

We want to write the above in terms of the Rademacher complexity of $\mathcal{H}$, which we can do by looking at the definition of Rademacher complexity. We have

$$\hat{R}_S(\mathcal{F}_{\mathcal{H}}) = \mathrm{E}_\sigma\left[\sup_{f_h\in\mathcal{F}_{\mathcal{H}}}\frac{1}{m}\sum_i \sigma_i f_h(x_i,y_i)\right].$$

Now, our functions f_h are just indicator functions and can be written $f_h(x_i,y_i) = \frac{1-y_ih(x_i)}{2}$. Further, we are indexing each function by a function $h\in\mathcal{H}$. Therefore, we can just index the supremum with $h\in\mathcal{H}$ instead of $f_h\in\mathcal{F}_{\mathcal{H}}$. Writing this out gives

$$\begin{aligned}\hat{R}_S(\mathcal{F}_{\mathcal{H}}) &= \mathrm{E}_\sigma\left[\sup_{h\in\mathcal{H}}\frac{1}{m}\sum_i \sigma_i\left(\frac{1-y_ih(x_i)}{2}\right)\right]\\ &= \frac{1}{2}\mathrm{E}_\sigma\left[\frac{1}{m}\sum_i\sigma_i + \sup_{h\in\mathcal{H}}\frac{1}{m}\sum_i(-y_i\sigma_i)h(x_i)\right].\end{aligned}$$

Because σ_i is a Rademacher random variable, its expectation is just 0. For the second term, we note that because the sample S is fixed, the y_i's are fixed, and therefore the term $-y_i\sigma_i$ is distributed the same as σ_i. Hence, we conclude

$$\hat{R}_S(\mathcal{F}_{\mathcal{H}}) = 0 + \frac{1}{2}\mathrm{E}_\sigma\left[\sup_{h\in\mathcal{H}}\frac{1}{m}\sum_i\sigma_ih(x_i)\right] = \frac{1}{2}\hat{R}_S(\mathcal{H}).$$

We have therefore shown

$$err(h) \leq e\hat{r}r(h) + R_m(\mathcal{H}) + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right)$$

$$err(h) \leq e\hat{r}r(h) + \hat{R}_S(\mathcal{H}) + \mathcal{O}\left(\sqrt{\frac{\ln 1/\delta}{m}}\right).$$

2 Obtaining other bounds

It was alluded to in class that obtaining the above bounds in terms of Rademacher complexity subsumes other bounds previously shown, which can be demonstrated with an example. We first state a simple theorem (a slightly weaker version of this theorem will be proved in a later homework assignment).

Theorem. *For $|\mathcal{H}| < \infty$:*

$$\hat{R}_S(\mathcal{H}) \leq \sqrt{\frac{2\ln|\mathcal{H}|}{m}}.$$

Now consider again the definition of empirical Rademacher complexity:

$$\hat{R}_S(\mathcal{H}) = \mathrm{E}_\sigma\left[\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m}\sigma_i h(x_i)\right].$$

We see that it only depends on how the hypothesis behaves on the fixed set S. We therefore have a finite set of behaviors on the set.

Define $\mathcal{H}' \subseteq \mathcal{H}$, where $\mathcal{H}'$ is composed of one representative from $\mathcal{H}$ for each possible labeling of the sample set S by $\mathcal{H}$. Therefore

$$|\mathcal{H}'| = |\Pi_\mathcal{H}(S)| \leq \Pi_\mathcal{H}(m).$$

Since the complexity only depends on the behaviors on S, we claim

$$\hat{R}_S(\mathcal{H}) = \mathrm{E}_\sigma\left[\sup_{h\in\mathcal{H}'} \frac{1}{m}\sum_{i=1}^{m}\sigma_i h(x_i)\right] = \hat{R}_S(\mathcal{H}').$$

We can now use the theorem stated above to write

$$\hat{R}_S(\mathcal{H}') \leq \sqrt{\frac{2\ln|\Pi_\mathcal{H}(S)|}{m}}.$$

Finally, we recall that after proving Sauer's lemma, we showed $\Pi_\mathcal{H}(m) \leq \left(\frac{em}{d}\right)^d$, for $m \geq d \geq 1$. Therefore

$$\hat{R}_S(\mathcal{H}) \leq \sqrt{\frac{2d\ln\left(\frac{em}{d}\right)}{m}}.$$

We have thus used the Rademacher complexity results to get an upper bound for the case of infinite $|\mathcal{H}|$ in terms of VC-dimension.

3 Boosting

Up until this point, the PAC learning model we have been considering requires that we be able to learn to arbitrary accuracy. Thus, the problem we have been dealing with is:

Strong learning $\mathcal{C}$ is *strongly* PAC-learnable if

$\exists$ algorithm A
$\forall$ distributions $\mathcal{D}$
$\forall c \in \mathcal{C}$
$\forall \epsilon > 0$
$\forall \delta > 0$
A, given $m = poly\,(1/\epsilon, 1/\delta, \ldots)$ examples, computes h such that

$$\Pr\,[err(h) \leq \epsilon] \geq 1 - \delta.$$

But what if we can only find an algorithm that gives slightly better than an even chance of error (e.g. 40%)? Could we use it to develop a better algorithm, iteratively improving our solution to arbitrary accuracy? We want to consider the following problem:

Weak learning $\mathcal{C}$ is *weakly* PAC-learnable if

$\exists \gamma > 0$
$\exists$ algorithm A
$\forall$ distributions $\mathcal{D}$
$\forall c \in \mathcal{C}$
$\forall \delta > 0$
A, given $m = poly\,(1/\epsilon, 1/\delta, \ldots)$ examples, computes h such that

$$\Pr\left[err(h) \leq \frac{1}{2} - \gamma\right] \geq 1 - \delta.$$

We note that in this problem we no longer require arbitrary accuracy, but only that the algorithm picked be able to do slightly better than random guessing, with high probability. The natural question that arises is whether weak learning is equivalent to strong learning.

Consider first the simpler case of a fixed distribution $\mathcal{D}$. In this case, the answer to our question is no, which we can illustrate through a simple example.

Example: For fixed $\mathcal{D}$, define
$X = \{0,1\}^n \cup \{z\}$
$\mathcal{D}$ picks z with probability 1/4 and with probability 3/4 picks uniformly from $\{0,1\}^n$
$\mathcal{C} = \{$ all concepts over X $\}$.

In a training sample, we expect to see z with high probability, and therefore z will be correctly learned by the algorithm. However, the remaining points are exponential in m, so that with only $poly(1/\epsilon, 1/\delta, \ldots)$ number of examples, we are unlikely to do much better than even chance on the rest of the domain. We therefore expect the error to be given roughly by

$$err(h) \approx \frac{1}{2} \cdot \frac{3}{4} + 0 \cdot \frac{1}{4} = \frac{3}{8}$$

in which case $\mathcal{C}$ is weakly learnable, but not strongly learnable.

We wish to prove that in the general case of an arbitrary distribution the following theorem holds:

Theorem. *Strong and weak learning are equivalent under the PAC learning model.*

The way we will reach this result is by developing a *boosting algorithm* which constructs a strong learning algorithm from a weak learning algorithm.

3.1 The boosting problem

The challenge faced by the boosting algorithm can be defined by the following problem.

Boosting problem *Given:*

$(x_1, y_1), \ldots, (x_m, y_m)$ with $y_i \in \{-1, +1\}$
access to a weak learner A:
  $\forall$ distributions D
  given examples from D
  computes h such that

$$\Pr\left[err_D(h) \leq \frac{1}{2} - \gamma\right] \geq 1 - \delta$$

Goal: find H such that with high probability $err_{\mathcal{D}}(H) \leq \epsilon$ for any fixed ϵ.

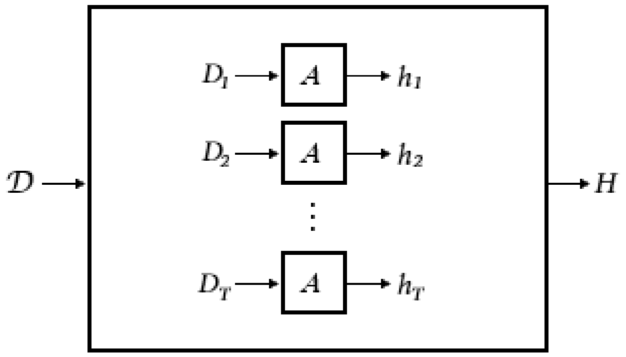


Figure 1: Schematic representation of boosting algorithm.

The main idea behind the boosting algorithm is to produce a number of different distributions D from $\mathcal{D}$, using the sample provided. This is necessary because running A on the same sample alone will not, in general, be enough to produce an arbitrarily accurate hypothesis (certainly so if A is deterministic). A boosting algorithm will therefore run as follows:

```
Boosting algorithm
    for $t = 1, \ldots, T$
        run $A$ on $D_t$ to get weak hypothesis $h_t : X \to \{-1, +1\}$
        $\epsilon_t = err_{D_t}(h_t) = \frac{1}{2} - \gamma_t$, where $\gamma_t \geq \gamma$
    end
    output $H$, where $H$ is a combination of the weak hypotheses $h_1, \ldots, h_T$.
```

In the above, the distributions D_t are distributions on the indices $1, \ldots, m$, and may vary from round to round. It is by adjusting these distributions that the boosting algorithm will be able to achieve high accuracy. Intuitively, we want to pick the distributions D_t such that, on each round, they provide us with more information about the points in the sample that are “hard” to learn. The boosting algorithm can be seen schematically in Figure 1

Let us define: $D_t(i) = D_t(x_i, y_i)$. We pick the distribution as follows:

$$\forall i : D_1(i) = \frac{1}{m}$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \\ e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \end{cases}$$

where $\alpha_t > 0$.

Intuitively, all our examples are considered equally in the first round of boosting. Going forward, if an example is misclassified, its weight in the next round will increase, while the weights of the correctly classified examples will decrease, so that the classifier will focus on the examples which have proven harder to classify correctly.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #11
Scribe: Eric Denovitzer — March 11, 2014

1 AdaBoost

Algorithm 1 AdaBoost

```
$\forall i : D_1(i) = \frac{1}{m}$
for $t = 1..T$ do
    $h_t \leftarrow$ Run A on $D_t$
    $\epsilon_t = err_{D_t}(h_t) = \frac{1}{2} - \gamma_t$
    $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    $\forall i : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \\ e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \end{cases}$
return $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
```

In this algorithm, Z_t represents a normalizing factor since D_{t+1} is a probability distribution.

1.1 Bounding the training error.

In the previous class, we gave the basic intuition behind the AdaBoost algorithm. Now, having defined the value for α_t, we tracked the three rounds of the algorithm in a toy example (see slides on the course website).

Theorem 1.1. *The training error is bounded by the following expression:*

$$\begin{aligned}
\hat{err}(H) &\leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \\
&= \exp\left(-\sum_t RE(\frac{1}{2} \mid\mid \epsilon_t)\right) && \text{(By definition of RE)} \\
&= \prod_t \sqrt{1-4\gamma_t^2} && \left(\epsilon_t = \frac{1}{2} - \gamma_t\right) \\
&\leq \exp\left(-2\sum_t \gamma_t^2\right) && (1 + x \leq e^x) \\
&\text{Consdering the weak learning assumption: } \gamma_t \geq \gamma > 0 \\
&\leq e^{-2\gamma^2 T}
\end{aligned}$$

Step 1: $D_{T+1}(i) = \frac{\exp[-y_i F(x_i)]}{m \prod\limits_t Z_t}$, $F(x) = \sum\limits_t \alpha_t h_t(x)$

Proof.
$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times e^{-\alpha_t y_i h_t(x_i)} = D_t(i)\frac{e^{-y_i \alpha_t h_t(x_i)}}{Z_t}$

Then, we can find this expression for $t = T$, and solve recursively:

$$\begin{aligned}
D_{T+1} &= D_1(i)\frac{e^{-y_i\alpha_1 h_1(x_i)}}{Z_1} \ \dots \ \frac{e^{-y_i\alpha_T h_T(x_i)}}{Z_T} \\
&= \frac{1}{m}\frac{\exp\left(-y_i \sum_t \alpha_t h_t(x_i)\right)}{\prod_t Z_t} \\
&= \frac{\exp\left[-y_i F(x_i)\right]}{m \prod_t Z_t}
\end{aligned}$$

□

Step 2: $\hat{err}(H) \leq \prod_t Z_t$

Proof.

$$\begin{aligned}
\hat{err}(H) &= \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{y_i \neq H(x_i)\} && (1)\\
&= \frac{1}{m}\sum_i \mathbf{1}\{y_i F(x_i) \leq 0\} && (2)\\
&\leq \frac{1}{m}\sum_i e^{-y_i F(x_i)} && (3)\\
&= \frac{1}{m}\sum_i D_{T+1}(i) m \prod_t Z_t && (4)\\
&= \prod_t Z_t \sum_i D_{T+1}(i) && (5)\\
&= \prod_t Z_t && (6)
\end{aligned}$$

(3) follows since $e^{-y_i F(x_i)} > 0$ if $-y_i F(x_i) > 0$ and $e^{-y_i F(x_i)} \geq 1$ if $-y_i F(x_i) \leq 0$. (4) follows from Step 1. (6) follows from the fact that we are adding all values over distribution D_{T+1} so we are getting 1. □

Step 3: $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$

Proof.

$$\begin{aligned}
Z_t &= \sum_i D_t(i) \times \begin{cases} e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \\ e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \end{cases} && (1)\\
&= \sum_{i: y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i: y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} && (2)\\
&= \epsilon_t e^{\alpha_t} + (1-\epsilon_t) e^{-\alpha_t} && (3)
\end{aligned}$$

(2) follows from just decomposing the sum for the two cases. (3) follows from the fact that e^{α_t} or $e^{-\alpha_t}$ can be taken outside of the sum, and $\sum\limits_{i:y_i \neq h_t(x_i)} D_t(i) = \epsilon_t$ and $\sum\limits_{i:y_i = h_t(x_i)} D_t(i) = 1 - \epsilon_t$.

We choose α_t to minimize the empirical error, so we get:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$$

*This is how we choose α_t in the algorithm. □

1.2 Bounding the generalization error.

Of the many tools we have used over the past classes, we choose the growth function to bound the generalization error.

$$H(x) = sign\left(\sum_t \alpha_t h_t(x) \right) \tag{1}$$

$$= g(h_1(x), \ldots, h_T(x)) \tag{2}$$

We defined $g(z_1, z_2, \ldots, z_t) = sign(\sum\limits_t \alpha_t z_t) = sign(\mathbf{w} \cdot \mathbf{z})$, with $\mathbf{w} = \langle \alpha_1, \alpha_2, \ldots, \alpha_T \rangle$, which represents linear threshold functions in $\mathbb{R}^T$. Let us define now the following spaces:

$\mathcal{J} = \{\text{LTFs in } \mathbb{R}^T\}$
$\mathcal{H} =$ weak hypothesis space
$\mathcal{F} =$ all functions f (as above), where $g \in \mathcal{J}$, $h_1, h_2, \ldots, h_T \in \mathcal{H}$

As proved in problem 2 of Homework 2, we can set the following bound:

$$\Pi_{\mathcal{F}}(m) \leq \Pi_{\mathcal{J}}(m) \prod_{t=1}^{T} \Pi_{\mathcal{H}}(m) \tag{3}$$

$$= \Pi_{\mathcal{J}}(m) \left[\Pi_{\mathcal{H}}(m) \right]^T \tag{4}$$

We have that VC-dim($\mathcal{J}$) $= T$ since we are considering linear threshold functions going through the origin in $\mathbb{R}^T$, and we define VC-dim($\mathcal{H}$) $= d$. Then, using Sauer's Lemma:

$$\Pi_{\mathcal{J}}(m) \leq \left(\frac{em}{T} \right)^T$$

$$\Pi_{\mathcal{H}}(m) \leq \left(\frac{em}{d} \right)^d$$

Plugging the above inequalities in equation (4):

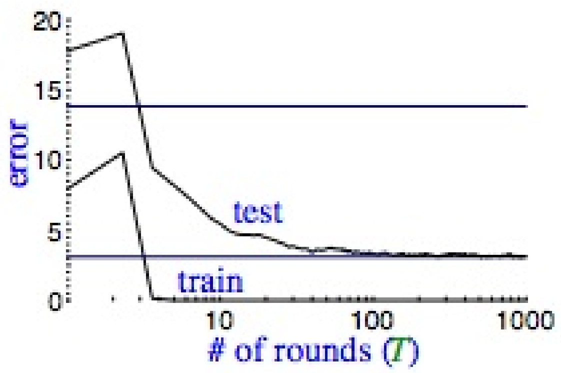
$$\Pi_{\mathcal{F}}(m) \leq \left(\frac{em}{T} \right)^T \left(\frac{em}{d} \right)^{dT} \tag{5}$$

Using "soft-oh" notation (not only hides constant but also log factors), given m examples, with probability at least $1 - \delta, \forall H \in \mathcal{F}$:

$$err(H) \leq e\hat{r}r(H) + \tilde{\mathcal{O}} \left(\sqrt{\tfrac{Td + \ln 1/\delta}{m}} \right)$$

1.3 Margin

Contrary to what we would expect based on the previous equation, as we increase T (the complexity) we do not always get a worse generalization error even when the training error is already 0. The following image is the one in the slides from class that represents this behavior:



Graph I : Error versus # of rounds of boosting

The reason behind this behavior is that, as we keep increasing the number of rounds, the classifier becomes more "confident". This confidence translates into a lower generalization error. We have:

$$H(x) = sign\left(\sum_{t=1}^{T} a_t h_t(x)\right), \text{ where } a_t = \frac{\alpha_t}{\sum_{t'=1}^{T} \alpha_{t'}}$$

In this way, we are normalizing the weights for each hypothesis, having $a_t \geq 0, \sum a_t = 1$. We define the margin as the difference between the weighted fraction of h_t's voting correctly and the fraction corresponding to those voting incorrectly. Then for an example x with correct label y, the margin is:

$$\begin{aligned} \text{margin} &= \sum_{t:h_t(x)=y} a_t - \sum_{t:h_t(x)\neq y} a_t \\ &= \sum_t a_t y h_t(x) \\ &= y \sum_t a_t h_t(x) \\ &= y f(x) \qquad \text{where } f(x) = \sum_t a_t h_t(x) \end{aligned}$$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #12
Scribe: Jun Wei Ho — March 13, 2014

1 Margin Theory for Boosting

Recall from the earlier lecture that we may write our hypothesis $H(x) = \text{sign}\left(\sum_{t=1}^{T} a_t h_t(x)\right)$, where $a_t = \alpha_t / \sum_s \alpha_s$ (so that $\sum_t a_t = 1$) and $h_1, \ldots, h_T$ are the weak hypotheses that we obtained over T iterations of AdaBoost.

Writing $f(x) = \sum_{t=1}^{T} a_t h_t(x)$, we define $\text{marg}_f(x, y) = yf(x)$ to be the *margin* of f for a training example (x, y). In the last lecture, we have seen that this quantity represents the weighted fraction of h_t's that voted correctly, minus the weighted fraction of h_t's that voted incorrectly, for the class y when given the data x.

A few remarks about the margin:

– $yf(x)$ takes values in the interval $[-1, 1]$

– $yf(x) > 0$ if and only if $H(x) = y$

– The magnitude $|yf(x)|$ represents the degree of 'confidence' for the classification $H(x)$. A number substantially far from zero implies high confidence, whereas a number close to zero implies low confidence.

It is therefore desirable for the margin $yf(x)$ to be 'large', since this represents a correct classification with high confidence. We will see that under the usual assumptions, AdaBoost is able to increase the margins on the training set and achieve a positive lower bound for these margins. In particular, this means that the training error will be zero, and we will see that larger margins help to achieve a smaller generalization error.

In this lecture, we aim to show that:

1. Boosting tends to increase the margins of training examples. Moreover, a bigger edge will result in larger margins after boosting.

2. Large margins on our training set leads to better performance on our test data (and this is independent of T, the number of rounds of boosting)

Notation

$\mathcal{S}$	Training set $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$
$\mathcal{H}$	Weak hypothesis space
d	$\text{VCdim}(\mathcal{H})$
$co(\mathcal{H})$	Convex hull of $\mathcal{H}$, the set of functions given by $\left\{ f(x) = \sum_{t=1}^{T} a_t h_t(x) \ : \ a_1, \ldots, a_T \geq 0, \ \sum_t a_t = 1, \ h_1, \ldots, h_T \in \mathcal{H}, \ T \geq 1 \right\}$
$Pr_{\mathcal{D}}$	Probability with respect to the true distribution $\mathcal{D}$
$E_{\mathcal{D}}$	Expectation with respect to the true distribution $\mathcal{D}$
$\widehat{Pr}_{\mathcal{S}}$	Empirical probability with respect to $\mathcal{S}$
$\widehat{E}_{\mathcal{S}}$	Empirical expectation with respect to $\mathcal{S}$

1.1 Boosting Increases Margins of Training Examples

We will show that given sufficient rounds of boosting, we can guarantee that $y_i f(x_i) \geq \gamma \; \forall \; i$, where $\gamma > 0$ is the edge in our weak learning assumption. In particular, this means that $H(x)$ will classify each training example correctly, and do so with confidence at least γ. The main result we will use is the following.

Theorem 1. *For $\theta \in [-1, 1]$, we have*

$$\widehat{Pr}_{\mathcal{S}}[\, yf(x) \leq \theta \,] \leq \prod_{t=1}^{T} \left[2\sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \right] \tag{1}$$

Moreover, if $\epsilon_t \leq \frac{1}{2} - \gamma$ for $t = 1, \ldots, T$, then

$$\widehat{Pr}_{\mathcal{S}}[\, yf(x) \leq \theta \,] \leq \left[\sqrt{(1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta}} \right]^T \tag{2}$$

Proof. Recall from the last lecture that

$$\frac{1}{m}\sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) \right) = \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

where we had set $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ to obtain the last equality.

Using a similar argument as before,

$$\begin{aligned}
\widehat{Pr}_{\mathcal{S}}[\, yf(x) \leq \theta \,] &= \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{y_i f(x_i) \leq \theta\} \\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) \leq \theta \sum_{t=1}^{T} \alpha_t\} \\
&\leq \frac{1}{m}\sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) + \theta \sum_{t=1}^{T} \alpha_t \right) \\
&= \exp\left(\theta \sum_{t=1}^{T} \alpha_t \right) \frac{1}{m}\sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) \right) \\
&= \exp\left(\theta \sum_{t=1}^{T} \alpha_t \right) \prod_{t=1}^{T} Z_t \\
&= \prod_{t=1}^{T} e^{\theta \alpha_t} Z_t \\
&= \prod_{t=1}^{T} \left[2\sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \right]
\end{aligned}$$

where the inequality follows from $\mathbb{1}\{x \leq 0\} \leq e^{-x}$, and the final equality is achieved by setting $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$.

The second result uses the fact that if $\epsilon_t \leq \frac{1}{2} - \gamma$, then

$$\begin{aligned} e^{\theta\alpha_t} Z_t &= e^{\theta\alpha_t}\left(\epsilon_t e^{\alpha_t} + (1-\epsilon_t)e^{-\alpha_t}\right) \\ &\leq e^{\theta\alpha_t}\left[\left(\frac{1}{2}-\gamma\right)e^{\alpha_t} + \left(\frac{1}{2}+\gamma\right)e^{-\alpha_t}\right] \\ &= \sqrt{(1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta}} \end{aligned}$$

by setting $\alpha_t = \frac{1}{2}\ln\left(\frac{\frac{1}{2}+\gamma}{\frac{1}{2}-\gamma}\right)$. The reader should verify the inequality and work out the details. $\square$

Remark. *By setting $\theta = 0$ in the above result, we recover the bound on training error proven in the previous lecture. Moreover, it is possible to show that for any $0 < \theta \leq \gamma$, the term $(1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta} < 1$, hence as $T \to \infty$ the RHS of (2) goes to zero. As an easy consequence, we have the following:*

Corollary. *If the weak learning assumption holds, then given sufficiently large T, we have $y_i f(x_i) \geq \gamma \;\forall\, i$.*

1.2 Large Margins on Training Set Reduce Generalization Error

Previously, we have shown that with probability at least $1-\delta$,

$$err(H) \leq \widehat{err}(H) + \tilde{O}\left(\sqrt{\frac{Td + \ln(1/\delta)}{m}}\right)$$

We can rewrite this equivalently as

$$Pr_{\mathcal{D}}[\,yf(x) \leq 0\,] \leq \widehat{Pr}_{\mathcal{S}}[\,yf(x) \leq 0\,] + \tilde{O}\left(\sqrt{\frac{Td + \ln(1/\delta)}{m}}\right)$$

We will now prove a variant of this result where the upper bound does not depend on T, but instead on a parameter θ that we can relate to the margin.

Theorem. *For $0 < \theta \leq 1$, with probability at least $1-\delta$,*

$$Pr_{\mathcal{D}}[\,yf(x) \leq 0\,] \leq \widehat{Pr}_{\mathcal{S}}[\,yf(x) \leq \theta\,] + \tilde{O}\left(\sqrt{\frac{d/\theta^2 + \ln(1/\delta)}{m}}\right).$$

Before we prove the theorem, we will first introduce two lemmas.

Recall that for $\mathcal{S} = \langle z_1, \ldots, z_m \rangle$ and $\mathcal{F} = \{f : Z \to \mathbb{R}\}$, the empirical Rademacher complexity of $\mathcal{F}$ is given by

$$\widehat{R}_{\mathcal{S}}(\mathcal{F}) = E_\sigma\left[\sup_{f\in\mathcal{F}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i f(z_i)\right]$$

In the last lecture, we've seen that $\widehat{R}_{\mathcal{S}}(\mathcal{H}) = \tilde{O}\left(\sqrt{\frac{d}{m}}\right)$. The following lemma tells us how $\widehat{R}_{\mathcal{S}}(co(\mathcal{H}))$ relates to $\widehat{R}_{\mathcal{S}}(\mathcal{H})$.

Lemma 1. *The Rademacher complexity of $\mathcal{H}$ is equal to the Rademacher complexity of its convex hull. In other words, $\widehat{R}_{\mathcal{S}}(co(\mathcal{H})) = \widehat{R}_{\mathcal{S}}(\mathcal{H})$.*

Proof. Since $\mathcal{H} \subset co(\mathcal{H})$, it is clear that $\widehat{R}_{\mathcal{S}}(\mathcal{H}) \leq \widehat{R}_{\mathcal{S}}(co(\mathcal{H}))$. Moreover,

$$\begin{aligned}
\widehat{R}_{\mathcal{S}}(co(\mathcal{H})) &= E_\sigma\left[\sup_{f\in co(\mathcal{H})} \frac{1}{m}\sum_{i=1}^m \sigma_i \sum_t a_t h_t(x_i)\right] \\
&= E_\sigma\left[\sup_{f\in co(\mathcal{H})} \frac{1}{m}\sum_t a_t \sum_{i=1}^m \sigma_i h_t(x_i)\right] \\
&\leq E_\sigma\left[\sup_{f\in co(\mathcal{H})} \frac{1}{m}\sum_t a_t \sup_{h\in\mathcal{H}} \sum_{i=1}^m \sigma_i h(x_i)\right] \\
&= E_\sigma\left[\sup_{f\in co(\mathcal{H})} \frac{1}{m} \sup_{h\in\mathcal{H}} \sum_{i=1}^m \sigma_i h(x_i)\right] \\
&= E_\sigma\left[\frac{1}{m} \sup_{h\in\mathcal{H}} \sum_{i=1}^m \sigma_i h(x_i)\right] \\
&= \widehat{R}_{\mathcal{S}}(\mathcal{H})
\end{aligned}$$

To obtain the fourth line we had used the fact that $\sum_t a_t = 1$, and for the fifth line we note that the expression in $\sup_f(..)$ does not depend on f, so we could omit the $\sup_f$ function. We therefore conclude that $\widehat{R}_{\mathcal{S}}(co(\mathcal{H})) = \widehat{R}_{\mathcal{S}}(\mathcal{H})$. □

Next, for any function $\phi : \mathbb{R} \to \mathbb{R}$, and $f : Z \to \mathbb{R}$, we define the composition $\phi \circ f : Z \to \mathbb{R}$ by $\phi \circ f(z) = \phi(f(z))$. We also define the space of composite functions $\phi \circ \mathcal{F} = \{\phi \circ f : f \in \mathcal{F}\}$.

Lemma 2. *Suppose ϕ is Lipschitz-continuous, that is, $\exists\ L_\phi > 0$ such that $\forall\ u, v \in \mathbb{R}$, $|\phi(u) - \phi(v)| \leq L_\phi |u - v|$. Then $\widehat{R}_{\mathcal{S}}(\phi \circ \mathcal{F}) \leq L_\phi \widehat{R}_{\mathcal{S}}(\mathcal{F})$.*

Proof. See Mohri et al. □

Equipped with the two lemmas, we are now ready to prove the main theorem. We will state the result once more:

Theorem 2. *For $0 < \theta \leq 1$, with probability at least $1 - \delta$,*

$$Pr_{\mathcal{D}}[\,yf(x) \leq 0\,] \leq \widehat{Pr}_{\mathcal{S}}[\,yf(x) \leq \theta\,] + \tilde{O}\left(\sqrt{\frac{d/\theta^2 + \ln(1/\delta)}{m}}\right).$$

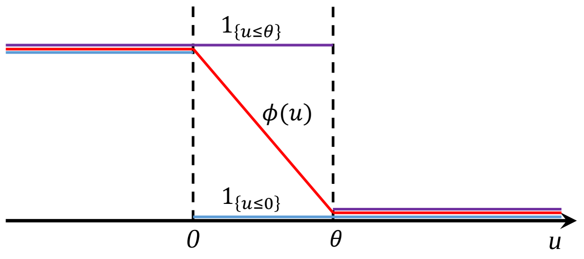
Proof. Write $\text{marg}_f(x, y) = yf(x)$. Define $\mathcal{M} = \{\text{marg}_f : f \in co(\mathcal{H})\}$. Then

$$\begin{aligned}
\widehat{R}_{\mathcal{S}}(\mathcal{M}) &= E_\sigma\left[\sup_{f\in co(\mathcal{H})} \frac{1}{m}\sum_{i=1}^m (\sigma_i y_i) f(x_i)\right] \\
&= \widehat{R}_{\mathcal{S}}(co(\mathcal{H})) \\
&= \widehat{R}_{\mathcal{S}}(\mathcal{H}) \qquad \text{(by Lemma 1)}
\end{aligned}$$

Next, we define the function $\phi : \mathbb{R} \to [0,1]$ by

$$\phi(u) = \begin{cases} 1 & \text{if } u \le 0 \\ 1 - u/\theta & \text{if } 0 < u \le \theta \\ 0 & \text{if } u > \theta \end{cases}$$

A plot of $\phi(u)$ is shown in the diagram below:



Note that for all $u \in \mathbb{R}$, we have

$$\mathbb{1}\{u \le 0\} \le \phi(u) \le \mathbb{1}\{u \le \theta\}$$

Moreover, ϕ is clearly Lipschitz-continuous with $L_\phi = \frac{1}{\theta}$. Therefore, Lemma 2 gives us

$$\widehat{R}_{\mathcal{S}}(\phi \circ \mathcal{M}) \;\le\; \frac{1}{\theta}\widehat{R}_{\mathcal{S}}(\mathcal{M}) \;=\; \frac{1}{\theta}\widehat{R}_{\mathcal{S}}(\mathcal{H}) \;\le\; \tilde{O}\left(\sqrt{\frac{d/\theta^2}{m}}\right)$$

Using the result from a previous lecture[1] and the results above, we have

$$\begin{aligned} Pr_{\mathcal{D}}[\,yf(x) \le 0\,] &= E_{\mathcal{D}}[\mathbb{1}\{yf(x) \le 0\}] \\ &\le E_{\mathcal{D}}[\phi \circ (yf)(x)] \\ &\le \widehat{E}_{\mathcal{S}}[\phi \circ (yf)(x)] + 2\widehat{R}_{\mathcal{S}}(\phi \circ \mathcal{M}) + O\left(\sqrt{\frac{\ln(1/\delta)}{m}}\right) \\ &\le \widehat{E}_{\mathcal{S}}[\mathbb{1}\{yf(x) \le \theta\}] + \tilde{O}\left(\sqrt{\frac{d/\theta^2}{m}}\right) + O\left(\sqrt{\frac{\ln(1/\delta)}{m}}\right) \\ &= \widehat{Pr}_{\mathcal{S}}[\,yf(x) \le \theta\,] + \tilde{O}\left(\sqrt{\frac{d/\theta^2 + \ln(1/\delta)}{m}}\right) \end{aligned}$$

as desired. $\square$

Remark. *The larger the value of θ we use, the smaller the $\tilde{O}(...)$ term on the RHS. With larger margins on the training set, we are able to choose larger values of θ while keeping the $\widehat{Pr}_{\mathcal{S}}[\,yf(x) \le \theta\,]$ term zero (or close to zero), and this will give us a sharper upper bound on the generalization error. This suggests that by increasing the margin on the training set, we may expect to see a smaller generalization error.*

[1]In an earlier lecture, we had proved that with probability at least $1 - \delta$, $\forall\, f \in \mathcal{F}$,

$$E_{\mathcal{D}}[f] \le \widehat{E}_{\mathcal{S}}[f] + 2\widehat{R}_{\mathcal{S}}(\mathcal{F}) + O\left(\sqrt{\frac{\ln(1/\delta)}{m}}\right)$$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #13
Scribe: Charles Marsh — March 25, 2014

1 Introduction to SVMs

1.1 Motivation

As we saw, the boosting technique wasn't defined so as to maximize margins, but we did end up using margins to analyze its performance. This begs the question: What if we derive an algorithm whose goal is, in fact, to maximize margins?

1.2 Assumptions

With boosting, we required the weak learning assumption. In this setting, we instead require that all features can be converted to real numbers, regardless of whether they're categorical, discrete, continuous, etc. This allows us to deal with feature vectors as if they're represented by points in Euclidean space, which we will assume for the remainder of the notes.

1.3 Intuition

To start, consider m labeled points $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ points in the plane, with $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$, as seen in Figure 1.

To classify these points, the tendency is to draw some line between them and consider "above the line" to be labeled $+1$ and "below the line", -1. But there are an infinite number of such lines that cleanly divide the data, two of which are seen in Figure 1. Which do we prefer?

Some of these lines are intuitively "better" than others. For example, while both the green and orange lines cleanly separate the data, the green line appears to be "better" than the orange because it provides wider margins. For example, if we receive a new data point located at the black dot, the orange line will classify it as a negative example when in fact it's quite close to a positive example. In other words, the orange line just barely separates the data consistently, while the green line leaves some margin for error.

Intuitively, if we consider a ball of radius δ around every example, we'd hope that all points in this δ-ball are classified with the same label as the given example. Thus, we want a hyperplane that not only classifies the training data correctly, but is also as far away as possible from all of the training points.

2 The Support Vector Machines Algorithm

The goal is to maximize the *margin* δ between the linear separator and the training data (note that in this setting, margins are not exactly the same as in boosting, although the two concepts are connected). There will always be examples that are exactly δ away from the separator, and these are known as the **support vectors**. In Figure 2, the support vectors are bolded with respect to the dotted linear separator.

Figure 1: Labeled examples in the plane

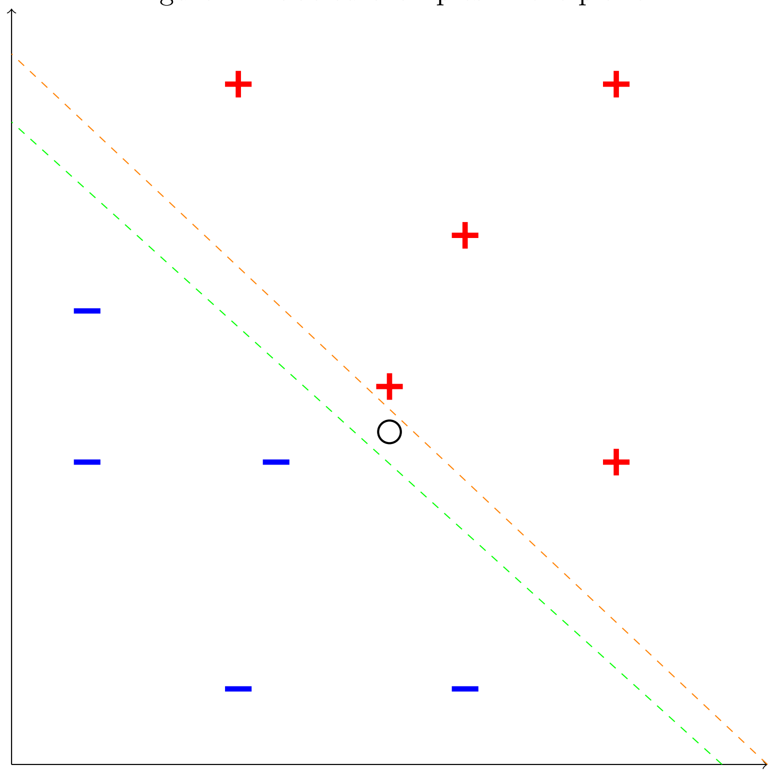
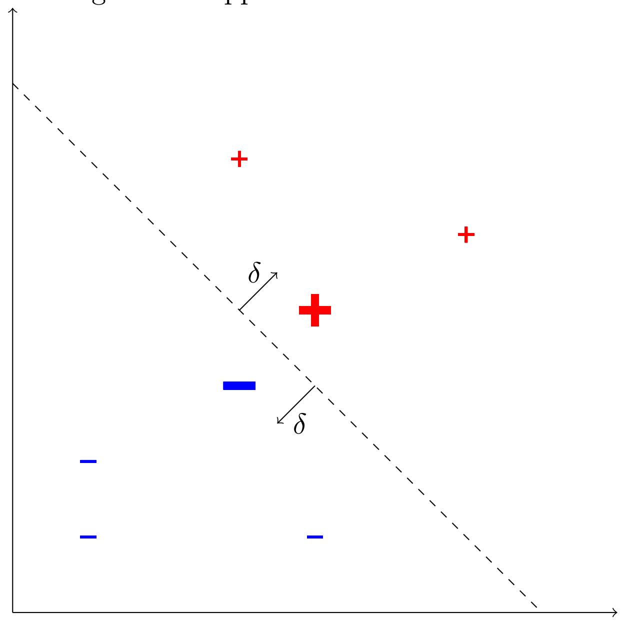


Figure 2: Support vectors with an LTF



2.1 VC-Dimension

Why is this a good approach? Can we find a more formal justification for maximizing the margin?

Recall that the VC-Dimension of an n-dimensional linear threshold function (LTF) through the origin is n. This may lead to acceptable generalization error bounds in low dimensions, but with SVMs, we'll often be working in spaces with hundreds of thousands of dimensions, so we'd like to do a lot better.

Assuming that all of our examples are contained within the unit ball ($\forall i : ||\mathbf{x}_i|| \leq 1$), we have the following nice result:

$$\text{VC-Dim(LTF with margin } \delta) \leq \frac{1}{\delta^2}$$

Note that if all our examples are contained with a ball of size R, we can normalize the data and the bound above becomes $(\frac{R}{\delta})^2$.

Why is this a nice result?

- It proves to us that larger margins leads to a smaller VC-Dimension, which in turn leads to better bounds on the generalization error.
- It's independent of the number of dimensions.

The proof of the bound on VC-Dimension will be given in a later lecture. An alternative bound on the generalization error can be achieved using Rademacher Complexity, the proof of which is very similar to that which was given for boosting.

3 How do we find these hyperplanes?

We start by formalizing the problem. Given that we're assuming our hyperplanes go through the origin, they can be defined by a single unit normal $\mathbf{v}$. That is, the hyperplane will be defined as all points orthogonal to this vector $\mathbf{v}$, with the additional constraint that $||\mathbf{v}|| = 1$.

For any given point $\mathbf{x}$, we'll also be interested in solving for the distance from the hyperplane to $\mathbf{x}$, which is simply:

$$\mathbf{v} \cdot \mathbf{x}$$

The scenario is laid out in Figure 3.

Next, note that this is a signed distance, such that:

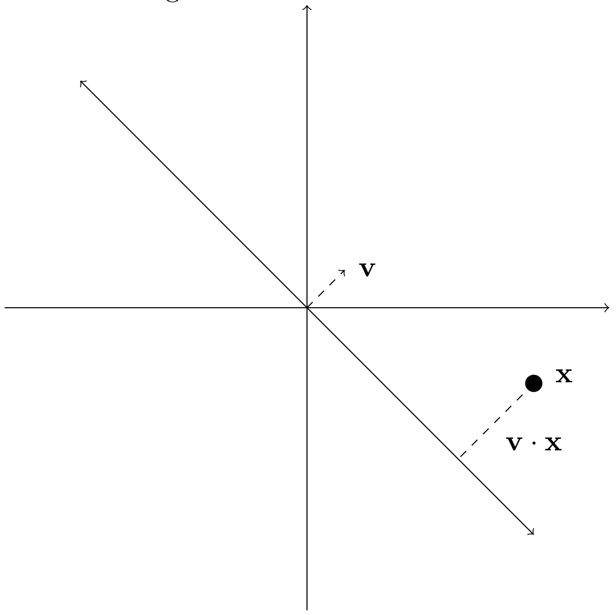
$$\begin{aligned} \mathbf{v} \cdot \mathbf{x} &> 0 \text{ if } \mathbf{x} \text{ is above the plane} \\ &< 0 \text{ if } \mathbf{x} \text{ is below the plane} \\ &= 0 \text{ if } \mathbf{x} \text{ is on the plane} \end{aligned}$$

As a result, we can classify a new point $\mathbf{x}$ simply by returning the sign of $\mathbf{v} \cdot \mathbf{x}$. Formally:

$$h(\mathbf{x}) = sign(\mathbf{v} \cdot \mathbf{x})$$

So under this setting, what problem are we trying to solve?

Figure 3: The unit normal



3.1 Formalizing the Problem

The problem statement is as follows:

Given $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ with $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$, maximize δ such that:

$$\begin{aligned} ||\mathbf{v}|| &= 1 \\ \forall i : \mathbf{v} \cdot \mathbf{x}_i &\geq \delta \quad \text{if } y_i = +1 \\ \mathbf{v} \cdot \mathbf{x}_i &\leq -\delta \text{ if } y_i = -1 \end{aligned}$$

We'll re-write this optimization problem several times.

3.1.1 Rewrite #1

First, we can combine the two primary constraints and rewrite the problem as follows:

Maximize δ such that:

$$\forall i : y_i(\mathbf{v} \cdot \mathbf{x}_i) \geq \delta$$
$$||\mathbf{v}|| = 1$$

3.1.2 Rewrite #2

Next, define $\mathbf{w} = \frac{\mathbf{v}}{\delta}$. This implies that $||\mathbf{w}|| = \frac{1}{\delta}$ and, subsequently, $\delta = \frac{1}{||\mathbf{w}||}$. Maximizing δ is now akin to minimizing $||\mathbf{w}||$. Thus, the optimization problem can now be written as:

Minimize $||\mathbf{w}||$ such that:

$$\forall i : y_i(\frac{\mathbf{v}}{\delta} \cdot \mathbf{x}_i) \geq \frac{\delta}{\delta}$$
$$y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$$

This is ideal because the constraint on $\mathbf{v}$ can now be ignored. Further, the problem is defined solely in terms of a single vector $\mathbf{w}$ and has no dependence on δ.

To avoid dealing with square roots, we redefine the objective to be:

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$
$$\text{Such that } \forall\, i : y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$$

3.2 The Lagrangian

As this problem is defined on a convex function subject to convex constraints, it is known as a "Convex Program". We'll proceed using standard techniques for solving such programs.

First, we rewrite our constraints as:

$$b_i(\mathbf{w}) \geq 0$$
$$\text{where: } b_i(\mathbf{w}) = y_i(\mathbf{w} \cdot \mathbf{x}_i) - 1$$

Next, we define the Lagrangian function $L(\mathbf{w}, \boldsymbol{\alpha})$ such that:

$$L(\mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i b_i(\mathbf{w})$$

Claim. *The previous formulation is identical to solving:*

$$\min_{\mathbf{w} \in \mathbb{R}^n} [\max_{\alpha_i \geq 0} [L(\mathbf{w}, \boldsymbol{\alpha})]]$$

Proof. Consider the optimization problem to be a game between two players Mindy and Max. Mindy's goal is to minimize the above function $L(\mathbf{w}, \boldsymbol{\alpha})$, and she gets to pick $\mathbf{w}$. Max, knowing Mindy's choice of $\mathbf{w}$, aims to maximize the above function by picking $\boldsymbol{\alpha}$ subject to the constraint that all $\boldsymbol{\alpha}_i \geq 0$.

Say Mindy picks $\mathbf{w}$ such that $b_i(\mathbf{w}) < 0$ for some i. Then, Max will pick $\alpha_i = \infty$, as he wants to maximize the function and $-b_i(\mathbf{w})\infty = \infty$ in this case. But this would be a terrible result for Mindy, who wants to minimize the function. As such, she will never choose $b_i(\mathbf{w}) < 0$, lest she allow Max the opportunity to earn a payoff of ∞.

Consider the two remaining cases:

- $b_i(\mathbf{w}) = 0 \implies \alpha_i$ is irrelevant as it contributes nothing to the Lagrangian function.
- $b_i(\mathbf{w}) > 0 \implies \alpha_i = 0$. Recall that $\alpha_i \geq 0$ by definition. Max's only choices are to play $\alpha_i = 0$ or $\alpha_i > 0$. If he chooses $\alpha_i > 0$, he'll be decreasing the value of the function. Thus, the best he can do is play $\alpha_i = 0$.

In either case, it holds that $\alpha_i b_i(\mathbf{w}) = 0$. Thus, the sum on the right will disappear and the optimization problem will resolve to minimizing $\frac{1}{2}||\mathbf{w}||^2$, just as in the previous formulation. □

3.3 The Convex Dual

For any function on two parameters, it can be shown that:

$$\min_{\mathbf{w}}[\max_{\boldsymbol{\alpha}}[L(\mathbf{w},\boldsymbol{\alpha})]] \geq \max_{\boldsymbol{\alpha}}[\min_{\mathbf{w}}[L(\mathbf{w},\boldsymbol{\alpha})]]$$

Under certain conditions, we can go further and show that the two expressions are in fact equal. The most important of these conditions is that the function L is convex in $\boldsymbol{\alpha}$ and concave in $\mathbf{w}$. As it turns out, our choice of L indeed satisfies this along with other necessary conditions, giving us equality:

$$\min_{\mathbf{w}}[\max_{\boldsymbol{\alpha}}[L(\mathbf{w},\boldsymbol{\alpha})]] = \max_{\boldsymbol{\alpha}}[\min_{\mathbf{w}}[L(\mathbf{w},\boldsymbol{\alpha})]]$$

The RHS of this equation is known as the **convex dual**. It too is a convex optimization problem. The dual is often easier to solve, but more importantly, it can reveal important facts about the structure of the problem itself.

Define:

$$\begin{aligned}\boldsymbol{\alpha}^* &= \arg\max_{\boldsymbol{\alpha}}[\min_{\mathbf{w}}[L(\mathbf{w},\boldsymbol{\alpha})]]\\ \mathbf{w}^* &= \arg\min_{\mathbf{w}}[\max_{\boldsymbol{\alpha}}[L(\mathbf{w},\boldsymbol{\alpha})]]\end{aligned}$$

Then, we have the following series of equations:

$$\begin{aligned}L(\mathbf{w}^*,\boldsymbol{\alpha}^*) &\leq \max_{\boldsymbol{\alpha}} L(\mathbf{w}^*,\boldsymbol{\alpha})\\ &= \min_{\mathbf{w}}\max_{\boldsymbol{\alpha}} L(\mathbf{w},\boldsymbol{\alpha})\\ &= \max_{\boldsymbol{\alpha}}\min_{\mathbf{w}} L(\mathbf{w},\boldsymbol{\alpha})\\ &= \min_{\mathbf{w}} L(\mathbf{w},\boldsymbol{\alpha}^*)\\ &\leq L(\mathbf{w}^*,\boldsymbol{\alpha}^*)\end{aligned}$$

3.4 Saddle Points

Examining two of the equations above, we have:

$$\min_{\mathbf{w}} L(\mathbf{w},\boldsymbol{\alpha}^*) = L(\mathbf{w}^*,\boldsymbol{\alpha}^*)$$

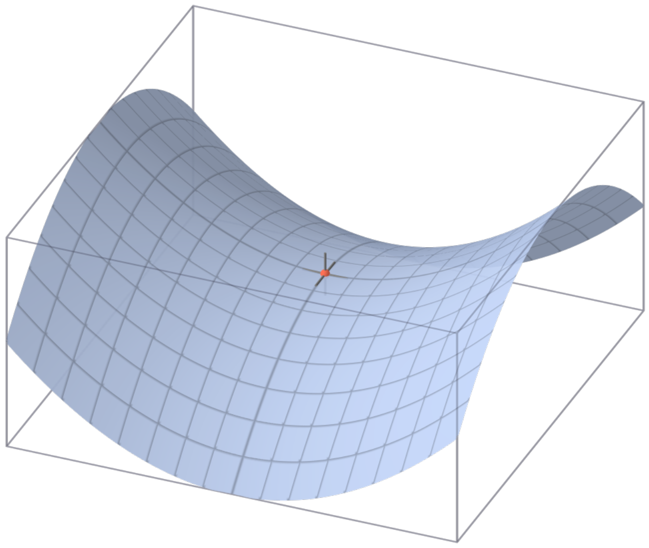
In other words, $\mathbf{w}^*$ minimizes L when the choice of $\boldsymbol{\alpha}*$ is fixed. Similarly, $\boldsymbol{\alpha}^*$ maximizes L when the choice of $\mathbf{w}$ is fixed. This implies that the pair of solutions $(\mathbf{w}^*,\boldsymbol{\alpha}^*)$ form a saddle point, as seen in Figure 4.

3.5 KKT Conditions

We already saw that:

$$\begin{aligned}\forall i : b_i(\mathbf{w}^*) &\geq 0\\ \alpha_i^* &\geq 0\\ \alpha_i^* b_i(\mathbf{w}^*) &= 0\end{aligned}$$

Figure 4: Saddle point



In addition, if $\mathbf{w}^*$ is in fact the minimizer, then the partial derivatives of the Lagrangian with respect to each component w_j should also be zero. This imposes an extra condition:

$$\forall j : \frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*)}{\partial w_j} = 0$$

Together, these are known as the Karush-Kuhn-Tucker (KKT) conditions.

We can solve this expression as follows:

$$\begin{aligned}\frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*)}{\partial w_j} &= 0 \\ &= w_j - \sum_{i=1}^{m} \alpha_i y_i x_{ij} \\ \implies \mathbf{w} &= \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i\end{aligned}$$

The $\mathbf{w}$ derived above is the weight vector that minimizes L for *fixed* $\boldsymbol{\alpha}$. The simple formula implies that for any $\boldsymbol{\alpha}$, we can find the appropriate weight vector. In particular, if we can find $\boldsymbol{\alpha}^*$, we can find $\mathbf{w}^*$.

As a next step, we can plug $\mathbf{w}$ back into L, which gives us the following:

$$\begin{aligned}L(\mathbf{w}, \boldsymbol{\alpha}) &= L(\sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \boldsymbol{\alpha}) \\ &= \sum_{i} \alpha_i - \frac{1}{2} \sum_{i}^{m} \sum_{j}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x_j})\end{aligned}$$

This represents the minimum over $\mathbf{w}$. To finish solving, we need to take the maximum over $\boldsymbol{\alpha}$ of the above expression subject to $\alpha_i \geq 0$. In effect, this represents the dual of the previous problem.

To find $\boldsymbol{\alpha}^*$, we can use iterative methods along the lines of the above until we reach the desired saddle point. And, as previously demonstrated, finding $\mathbf{w}^*$ given $\boldsymbol{\alpha}^*$ is merely a matter of plugging $\boldsymbol{\alpha}^*$ into the equation $\mathbf{w}^* = \sum_{i=1}^m \alpha_i^* y_i \mathbf{x}_i$.

3.6 Support Vectors & Generalization Error

All the conditions outlined above will still hold at our solution $(\mathbf{w}^*, \boldsymbol{\alpha}^*)$. In particular:

$$\alpha_i^* b_i(\mathbf{w}^*) = 0$$
$$\alpha_i^* [y_i(\mathbf{w}^* \cdot \mathbf{x}_i) - 1] = 0$$

Therefore, if $\alpha_i^* \neq 0$, it follows that $y_i(\mathbf{w}^* \cdot \mathbf{x}_i) = 1$. This implies that example $(\mathbf{x}_i, y_i)$ is a support vector, as we can substitute $\mathbf{w} = \frac{\mathbf{v}}{\delta}$ to get:

$$y_i(\mathbf{v} \cdot \mathbf{x}_i) = \delta$$

This satisfies exactly the definition of a support vector in that example $(\mathbf{x}_i, y_i)$ is exactly δ away from the hyperplane.

Further, if $(\mathbf{x}_i, y_i)$ is *not* a support vector, then working backwards, you can see that $b_i(\mathbf{w}^*) \neq 0$, which implies that $\alpha_i = 0$. As a result, the solution $\mathbf{w}$ will be a linear combination of the support vectors! Therefore, our output hypothesis can be defined just in terms of these support vectors, a subset of the overall training data.

Say there are k such support vectors and our algorithm outputs hypothesis h. As our output hypothesis is a function of a subset of the training data, we can use the result of Homework 2, Problem 4 to bound the generalization error:

$$err(h) \leq \tilde{O}(\frac{k + \ln(\frac{1}{\delta})}{m})$$

This result is particularly appealing as it is independent of the number of dimensions.

4 Linear Inseparability: Soft Margins

Until now, we've assumed that our data is linearly separable. That is, we've assumed that there exists a valid half plane that can divide the data consistently. Often, however, this assumption does not hold.

It might be the case, instead, that we have a few examples that are incorrectly classified by our LTF, as in Figure 5. We'd like to somehow *move* the bad examples to the other side of the hyperplane. But for this, we'd have to pay a price.

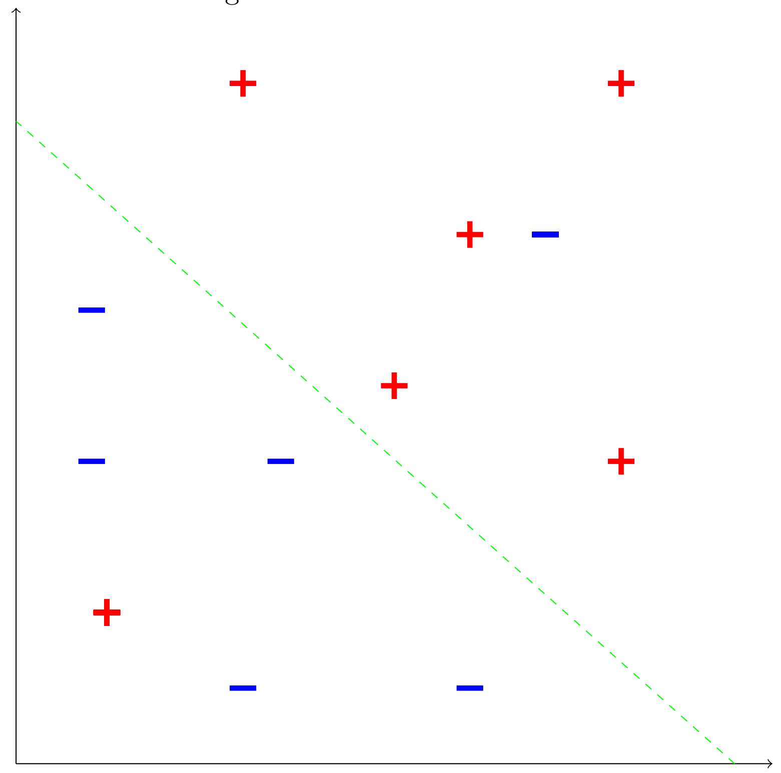
In the previous setting, our goal was to minimize $\frac{1}{2}||\mathbf{w}||^2$ subject to the constraint:

$$\forall i: \; y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$$

However, in this new setting (in which we're allowed to nudge our examples around slightly), we pay a different price. The problem setting is as follows:

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^m \xi_i$$
$$\text{Such that } \forall\, i: \; y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

Figure 5: An Inconsistent LTF



Essentially, ξ_i is the amount which we move example i, and C is some positive constant. This approach to linear inseparability is known as the "Soft Margins" technique.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Li-Fang (Fanny) Cheng

Lecture #14
March 27, 2014

1 SVM

1.1 A brief review

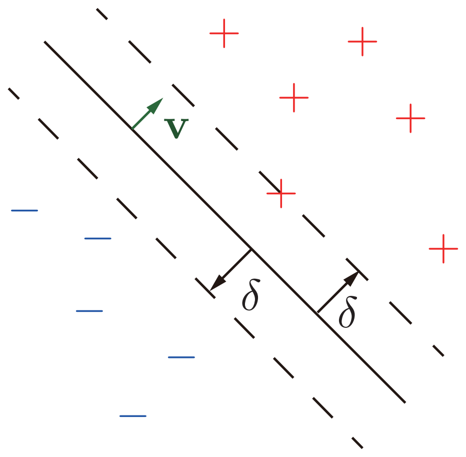


Figure 1: An illustration of the key idea of SVM for linearly separable data.

As discussed in the previous lecture, the key idea of support vector machine (SVM) is to find a hyperplane that can separate the given data. Figure 1 illustrates this idea when the data is linearly separable. The hyperplane is defined by a unit normal vector $\mathbf{v}$, and if we suppose the hyperplane passes through the origin, we can formulate the prediction of a data point $\mathbf{x}$ as

$$y = \text{sign}\left(\mathbf{v} \cdot \mathbf{x}\right)$$

The distance from a data point to the hyperplane in the "right" direction is called the margin:

$$\text{margin}(\mathbf{x}, y) = y\left(\mathbf{v} \cdot \mathbf{x}\right)$$

For a given labeled data set $(\mathbf{x}_i, y_i)$, $i = 1, \cdots m$, we define the smallest margin δ as

$$\delta = \min_i y_i\left(\mathbf{v} \cdot \mathbf{x}_i\right)$$

Our objective is to find a hyperplane with maximized δ by solving convex optimization problems. Figure 2 gives a summary of both the primal convex optimization problem and its dual form.

The next question we would like to ask is what kinds of operations do we need for each sample when solving the optimization problems. According to Equation 1, we can tell that the dot product, $(\mathbf{x}_i \cdot \mathbf{x}_j)$, is the only computation we need for each sample when solving the dual SVM problem.

Goal of SVM problem: find $\mathbf{v}$ to maximize δ

$$\text{define} \quad \mathbf{w} = \frac{\mathbf{v}}{\delta} \Rightarrow \mathbf{v} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Primal Form (Linearly Separable):

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & \forall i : y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 \end{aligned}$$

Dual Form (Linearly Separable):

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\begin{aligned} \max \quad & \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \qquad (1) \\ \text{s.t.} \quad & \forall i : \alpha_i \geq 0 \end{aligned}$$

Primal Form (Soft Margin):

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \sum_i \xi_i & (2) \\ \text{s.t.} \quad & \forall i : y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 - \xi_i & (3) \\ & \xi_i \geq 0 & (4) \end{aligned}$$

Figure 2: The primal and dual SVM problems.

Sometimes there is a case that the data is linearly inseparable. If the data is "almost" linearly separable, we can use the soft margin SVM. In this case, we allow the hyperplane to make a few mistakes in classification by moving some data points slightly. The optimization problem is then reformulated in Equation 2 to 4, as discussed in the previous lecture.

1.2 More on Linearly Inseparable Data

What can we do if the data is just too far from linearly separable as is the case in Figure 4? In this situation, we have to look for another solution. We map the data to a higher dimensional space where the data can be linearly separable. Here follows an example.

Suppose the original data is in 2-dimensional space; we can use the following method to map it into 6 dimensional space:

$$\mathbf{x} = (x_1, x_2) \mapsto \psi(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2) \qquad (5)$$

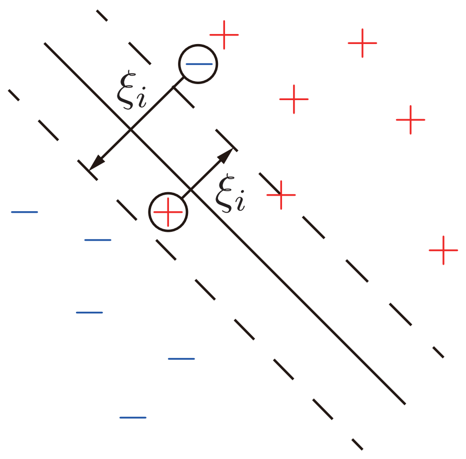


Figure 3: An illustration of the soft margin SVM for nearly linearly separable data.

The new hyperplane consists of the points that $\mathbf{v} \cdot \psi(\mathbf{x}) = 0$, which is written as:

$$a + bx_1 + cx_2 + dx_1x_2 + ex_1^2 + fx_2^2 = 0 \tag{6}$$

where $\mathbf{v} = (a, b, c, d, e, f)$. Equation 6 defines a hyperplane in the 6-dimensional space, while in the original 2-dimensional space, it is the general equation for a conic section, that is, a line, circle, ellipse, parabola or hyperbola. Therefore, by this mapping, we are able to separate the 2-dimensional data in the 6-dimensional space. Figure 5 shows a possible 6-dimensional hyperplane that has the form of an ellipse in the 2-dimensional space for classification.

The method described above can be generalized. If we start with n dimensional space, by adding up all terms of degree at most k, we can have $O(n^k)$ dimensional space.

Next, we have to notice possible problems when adopting this approach. There are a statistical problem and a computational problem:

- Statistical Problem: According to the results above, if we start from 100 dimensions, it is possible that we reach more than a trillion dimensions. That is, we will have more parameters to train and the complexity of the hypothesis will be higher. In this case, we generally need more samples to achieve better fitting results. If we have too few samples, the algorithm overfits easily.

- Computational Problem: The storage space of the data is in proportion to the number of dimensions. It would take too much time to even read the data if the dimension after mapping is too high.

However, SVM can overcome both kinds of problems. First, for the statistical problem, we use the result that

$$\text{VC-dimension} \le \left(\frac{R}{\delta}\right)^2, \tag{7}$$

where R is the radius of the sphere that contains all the data, and δ is the margin. We should notice that VC-dimension does not depend on the number of dimensions of the data. Therefore, although increasing the dimension of data might increase R, generally δ also gets larger. That is, we can expect that VC-dimension is not growing so fast.

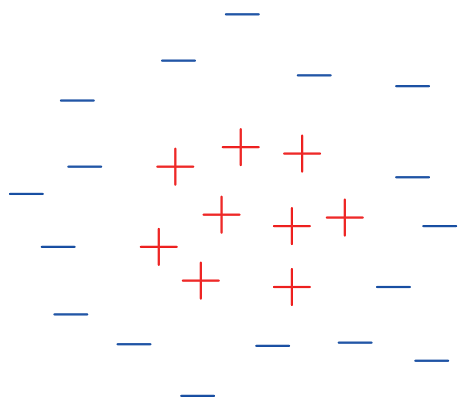


Figure 4: A non-linearly separable data set in $\mathbb{R}^2$.

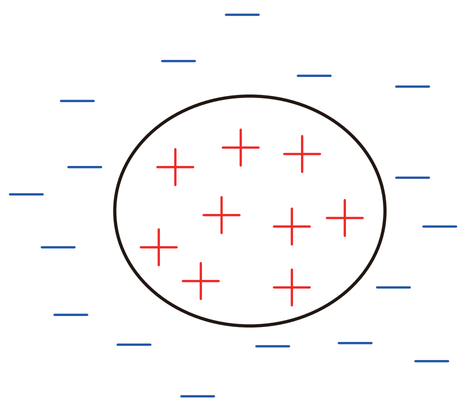


Figure 5: Finding an ellipse to separate the data by mapping from $\mathbb{R}^2$ to $\mathbb{R}^6$.

For the computational problem, recall the previous observation that for each sample, we only need to compute the inner products. Therefore after the mapping, the computation for two samples $\mathbf{x}$ and $\mathbf{z}$ is:

$$\psi(\mathbf{x}) \cdot \psi(\mathbf{z})$$

However, based on the numbers we gave above (mapping from 100 dimensions to a trillion dimensions), the computation could become really slow if the dimension gets too high. We will talk about how to relieve this problem in SVM.

1.3 The kernel trick

We first revisit the mapping function described in Equation 5. Suppose now we modify the mapping function $\psi(\mathbf{x})$ as:

$$\mathbf{x} = (x_1, x_2) \mapsto \psi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2) \tag{8}$$

Since we only change the constant for some terms, this does not affect the hyperplane we can represent after mapping. However, the inner product for $\psi(\mathbf{x})$ and $\psi(\mathbf{z})$ becomes:

$$\begin{aligned}\psi(\mathbf{x})\cdot\psi(\mathbf{z}) &= 1+2x_1z_1+2x_2z_2+2x_1z_1x_2z_2+(x_1z_1)^2+(x_2z_2)^2\\ &= (1+x_1z_1+x_2z_2)^2\\ &= (1+\mathbf{x}\cdot\mathbf{z})^2\end{aligned} \tag{9}$$

That is, if original dimension is n, by adding all terms of degree at most k, we have

$$\psi(\mathbf{x})\cdot\psi(\mathbf{z}) = (1+\mathbf{x}\cdot\mathbf{z})^k = K(\mathbf{x},\mathbf{z}) \tag{10}$$

The computation complexity is now $O(n)$ instead of $O(n^k)$, and the computational problem can be relieved.

The result shown above indicates that it is possible to calculate the inner product in higher dimensional space using only the inner product in lower dimensional space under some specific mapping. Generally, we define a kernel function $K(\mathbf{x},\mathbf{z}) = \psi(\mathbf{x})\cdot\psi(\mathbf{z})$ for the mapping ψ with this property. By replacing the inner product $(\mathbf{x}_i\cdot\mathbf{x}_j)$ in Equation 1 with $K(\mathbf{x}_i,\mathbf{x}_j)$, we can obtain the higher dimensional hyperplane as the solution. This method is called the kernel trick. There are different kinds of kernels in practice. The kernel function shown in Equation 10 is called the polynomial kernel. Another popular kernel is Gaussian radial basis function (RBF) kernel $K(\mathbf{x},\mathbf{z}) = \exp\left(-c\cdot\|\mathbf{x}-\mathbf{z}\|^2\right)$, whose dimension is infinite. Another thing we should notice is that the input to the kernel function $\mathbf{x}$ and $\mathbf{z}$ are not necessarily vectors. For instance, $\mathbf{x}$ and $\mathbf{z}$ can be entirely different kinds of objects, such as strings or trees, as long as the kernel function provides the mapping of inner product for them.

In summary, our objective for solving SVM problem is to maximize the margin δ. When the data is linearly inseparable, we can deal with it by combining the kernel trick and the soft margin approach.

1.4 Comparison of SVM and boosting

We can now compare SVM with boosting. In SVM, we treat the input data as points in Euclidean space: $\mathbf{x}\in\mathbb{R}^n$. As discussed in the previous lecture, it is natural to assume $\|\mathbf{x}\|_2 \le 1$. In boosting, we never really touch the data. Instead, what we manipulate are the weak hypotheses. To make things simple, suppose we use a finite weak hypothesis space $\mathcal{H} = \{g_1(x),\cdots,g_N(x)\}$ and the input can be viewed as the vector $\mathbf{h}(x) = \langle g_1(x),\cdots,g_N(x)\rangle$. Recall that the infinity norm for a vector $\mathbf{z}$ is defined as $\|\mathbf{z}\|_\infty = \max_j |z_j|$. Therefore, we have $\|\mathbf{h}(x)\|_\infty = \max_j |g_j(x)| = 1$ since $g_j(x)\in\{-1,+1\}$.

Next, we compare the coefficients to compute and the predictions. In SVM, the algorithm computes the unit normal vector $\mathbf{v}$ for the hyperplane, and the prediction is $\text{sign}(\mathbf{v}\cdot\mathbf{x})$. In boosting, the algorithm computes the coefficient $\alpha_t \ge 0$ for the weak hypothesis $h_t\in\mathcal{H}$. Suppose we run the boosting algorithm for T times, the prediction $\text{sign}\left(\frac{\sum_{t=1}^T \alpha_t h_t(x)}{\sum_t^T \alpha_t}\right)$ is a convex combination of h_t. Since each $h_t\in\mathcal{H} = \{g_1(x),\cdots,g_N(x)\}$, we can rewrite the prediction into another convex combination of $g_j(x)$ by finding the corresponding weight a_j:

$$\left(\frac{\sum_{t=1}^T \alpha_t h_t(x)}{\sum_t^T \alpha_t}\right) = \sum_{j=1}^N a_j g_j(x) = \mathbf{a}\cdot\mathbf{h}(x),$$

where $\mathbf{a} = \langle a_1,\cdots,a_N\rangle$. It should be noted that $\sum_{j=1}^N |a_j| = \|\mathbf{a}\|_1 = 1$, where $a_j \ge 0$. Therefore, the goal of boosting can be viewed as finding a_j for each g_j, and the prediction for sample x is $\text{sign}\left(\mathbf{a}\cdot\mathbf{h}(x)\right)$.

	SVM	AdaBoost
input	$\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_2 \leq 1$	$\mathbf{h}(x)$, $\|\mathbf{h}(x)\|_\infty = 1$
finds	$\|\mathbf{v}\|_2 = 1$	$\|\mathbf{a}\|_1 = 1$, $\mathbf{a} = \langle a_1, \cdots, a_N \rangle$
prediction	$\text{sign}(\mathbf{v} \cdot \mathbf{x})$	$\text{sign}\left(\mathbf{a} \cdot \mathbf{h}(x)\right) = \text{sign}\left(\sum_{j=1}^N a_j g_j(x)\right)$
margin	$y(\mathbf{v} \cdot \mathbf{x})$	$y\left(\mathbf{a} \cdot \mathbf{h}(x)\right)$

Figure 6: Comparison of SVM and Boosting.

Finally, we compare the margin of both SVM and boosting algorithms. The margin of SVM is $y\left(\mathbf{v} \cdot \mathbf{x}\right)$, while in boosting it is $y\left(\mathbf{a} \cdot \mathbf{h}(x)\right)$. The goal of both SVM and AdaBoost is to maximize the margin, but the norms that are used here are different: in SVM's, $\|v\|_2 = 1$ and $\|x\|_2 = 1$, while in boosting, $\|a\|_1 = 1$ and $\|h(x)\|_\infty = 1$.

The summary of the comparisons described above are listed in Figure 6.

2 Online Learning

2.1 Introduction

So far we have focused on the PAC learning model. We assume there is a fixed distribution for both the training and testing data, and the training samples are selected randomly. The algorithms we have discussed are batch learning algorithms, which means that after training, the hypothesis is fixed and then used for all future testing samples. Now we move on to online learning. The following are some properties of online learning. First, both training and testing happen at the same time in online learning. The learner gets one training sample at a time, makes the prediction, and then gets the true result as feedback. An example is to predict the stock market. In the morning the online learner makes a prediction about whether the price will go up or down, and then after one day it can receive the true situation and adjust future prediction. Second, the online learning algorithms tend to be simple. Third, online learning model makes no assumption about the distribution of the data, and can even completely drop the assumption that the data is generated randomly. In the following and future lectures, we will show that even without these assumptions, we can still analyze online learning algorithms in a meaningful way.

2.2 Learning with expert advice

We start from looking at an example of the stock market. Figure 7 gives the setting of the example. Suppose there are four experts who will make predictions for the price every morning, and the learner makes the prediction based on the four predictions. The goal of the learner is to provide the performance as good as, or at least not too much worse, than the best expert after a certain amount of time. This general setting is called "learning with expert advice" and is formulated as below:

```
$N = \#$ of experts
for $t = 1, \cdots, T$
  each expert $i$ predicts $\xi_i \in \{0, 1\}$ ,
  learner predicts $\hat{y} \in \{0, 1\}$ ,
  observe outcome $y \in \{0, 1\}$ (mistake if $\hat{y} \neq y$).
```

	Experts				Learner (Master)	Outcome
	1	2	3	4		
day 1	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
day 2	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
# of mistakes	37	12	67	50	18	

Figure 7: The stock market example.

We would like to relate the number of mistakes of the learner to the number of mistakes of the best expert. However, the learner does not know which expert is the best. In the case assuming at least one expert is perfect, we can adopt a simple algorithm described as below:

$$\hat{y} = \text{ majority vote of experts with no mistakes so far}$$

This method is called the "Halving algorithm". To give a more concrete idea about how it works, we revisit the stock market example in Figure 7. In the first day, expert 3 made a mistake, so the learner does not take the prediction of expert 3 into account starting from the second day.

Now we can calculate the mistake bound of the Halving algorithm as follows. Let W be the number of experts that make no mistake so far, or we say the number of *active* experts. Initially we have $W = N$. After the learner made one mistake, we have $W \leq \frac{1}{2}N$ because there are at least half of the active experts that made this mistake. Similarly, after the learner made the second mistake, $W \leq \frac{1}{4}N$, and so on. After the learner made m mistakes, we have $W \leq \frac{1}{2^m}N$. Due to the assumption that at least one expert is perfect, we have $W \geq 1$. That is,

$$1 \leq W \leq \frac{1}{2^m}N \Rightarrow m \leq \lg(N).$$

Finally, we can consider a special case in which we view each expert as one hypothesis. Suppose we have a hypothesis space $\mathcal{H} = \{h_1, \cdots, h_N\}$, and the target concept $c \in \mathcal{H}$. By adopting the Halving algorithm, each round the learner gets one sample x, makes the prediction $\hat{y} \in \{0, 1\}$, and then observes the true result $y = c(x)$. We then have the following mistake bound:

$$\#\text{ of mistakes } \leq \lg(N) = \lg(|\mathcal{H}|).$$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture # 15
Scribe: Jieming Mao — April 1, 2013

1 Brief review

1.1 Learning with expert advice

Last time, we started to talk about "learning with expert advice". This model is very different from models we saw earlier in the semester. In this model, the learner can see only one example at a time, and has to make predictions based on the advice of "experts" and the history. Formally, the model can be written as

- $N = \#$ of experts
- for $t = 1, \cdots, T$
  - each expert i predicts $\xi_i \in \{0, 1\}$
  - learner predicts $\hat{y} \in \{0, 1\}$
  - learner observes $y \in \{0, 1\}$
  - (mistake if $y \neq \hat{y}$)

In the previous lecture, we gave a "Halving algorithm" which works when there exists a "perfect" expert. In this case, the number of mistakes that the "Halving algorithm" makes is at most $\lg(N)$.

1.2 Connection with PAC learning

Now we consider an "analogue of the PAC learning model":

- hypothesis space $\mathcal{H} = \{h_1, \cdots, h_N\}$
- target concept $c \in \mathcal{H}$
- on each round
  - get x
  - predict $\hat{y}$
  - observe $c(x)$

This problem is a very natural online learning problem. However, we can relate this problem to "learning with expert advice" by considering each hypothesis h_i as an expert. Since the target concept is chosen inside the hypothesis space, we always have a "perfect" expert. Therefore, we can apply the "Halving algorithm" to solving this problem and the "Halving algorithm" will make at most $\lg(N) = \lg(|\mathcal{H}|)$ mistakes.

2 Bounding the number of mistakes

In the previous section, it is natural to ask whether the "Halving algorithm" is the best possible. In order to answer this problem, we have to define what is "best". Let A be any deterministic algorithm, define

$$M_A(\mathcal{H}) = \max_{c,x}(\#\text{ mistakes made by } A)$$

$M_A(\mathcal{H})$ is the number of mistakes A will make on hypothesis space $\mathcal{H}$ in the worst case. An algorithm is considered good if its $M_A(\mathcal{H})$ is small. Now define

$$\mathit{opt}(\mathcal{H}) = \min_A M_A(\mathcal{H}).$$

We are going to lower bound $\mathit{opt}(\mathcal{H})$ by $VCdim(\mathcal{H})$ as the following theorem:

Theorem 1. $VCdim(\mathcal{H}) \leq \mathit{opt}(\mathcal{H})$

Proof: Let A be any deterministic algorithm. Let $d = VCdim(\mathcal{H})$. Let $x_1, ..., x_d$ be shattered by $\mathcal{H}$. Now we can construct an adversarial strategy that forces A to make d mistakes as following:

- for $t = 1, \cdots, d$
  - present x_t to A
  - $\hat{y}_t = A$'s prediction
  - choose $y_t \neq \hat{y}_t$.

Since $x_1, ..., x_d$ is shattered by $\mathcal{H}$, we know there exists a concept $c \in \mathcal{H}$, such that $c(x_t) = y_t$ for all t. In addition, since A is deterministic, we can simulate it ahead of time. Therefore such adversarial construction is possible. Thus there exists an adversarial strategy to force A to make d mistakes. □

To sum up this section, we have the following result,

$$VCdim(\mathcal{H}) \leq \mathit{opt}(\mathcal{H}) \leq M_{Halving}(\mathcal{H}) \leq \lg(|\mathcal{H}|).$$

3 Weighted majority algorithm

Now we are going back to "learning with expert advice" and we want to devise algorithms when there is no "perfect" expert. In this setting, we will compare the number of mistakes of our algorithm and the number of mistakes of the best expert. Here we modify the "Halving algorithm" to get an algorithm when there is no "perfect" expert. Instead of discarding experts, we will keep a weight for each expert and lower it when this expert makes a mistake. We call this algorithm weighted majority algorithm:

- w_i = weight on expert i
- Parameter $0 \leq \beta < 1$
- $N = \#$ of experts

- Initially $\forall i, w_i = 1$
- for $t = 1, \cdots, T$
  - each expert i predicts $\xi_i \in \{0,1\}$
  - $q_0 = \sum_{i:\xi_i=0} w_i$, $q_1 = \sum_{i:\xi_i=1} w_i$.
  - learner predicts $\hat{y} \in \{0,1\}$
  - $\hat{y} = \begin{cases} 1 & \text{if } q_1 > q_0 \\ 0 & \text{else} \end{cases}$ (weighted majority vote)
  - learner observes $y \in \{0,1\}$
  - (mistake if $y \neq \hat{y}$)
  - for each i, if $\xi_i \neq y$, then $w_i \leftarrow w_i\beta$

Now we are going to analyze weighted majority algorithm(WMA) by proving the following theorem:

Theorem 2. *(# of mistakes of WMA) $\leq a_\beta$(# of mistakes of the best expert) $+c_\beta \lg(N)$, where $a_\beta = \frac{\lg(1/\beta)}{\lg(\frac{2}{1+\beta})}$ and $c_\beta = \frac{1}{\lg(\frac{2}{1+\beta})}$.*

Before proving this theorem, let's first try to understand what this theorem implies. The following table gives a good understanding of the parameter:

β	a_β	c_β
$1/2$	≈ 2.4	≈ 2.4
0	$+\infty$	1
$\rightarrow 1$	2	$+\infty$

If we divide both sides of the inequality by T, we get

$$\frac{(\text{\# of mistakes of WMA})}{T} \leq \frac{a_\beta(\text{\# of mistakes of the best expert})}{T} + \frac{c_\beta \lg(N)}{T}$$

When $T \rightarrow +\infty$, $\frac{c_\beta \lg(N)}{T} \rightarrow 0$. Then this theorem means that the rate that WMA makes mistakes is bounded by a constant times the rate that the best expert makes mistakes.
Proof: Define W as the sum of the weights of all the experts: $W = \sum_{i=1}^{N} w_i$. Initially, we have $W = N$. Now we consider how W changes in some round. On some round, without loss of generality, we assume that $y = 0$. Then

$$\begin{aligned} W_{new} &= \sum_{i=1}^{N} w_i^{new} \\ &= \sum_{i:\xi_i=1} w_i\beta + \sum_{i:\xi_i=0} w_i \\ &= q_1\beta + q_0 \\ &= q_1\beta + (W - q_1) \\ &= W - (1-\beta)q_1 \end{aligned}$$

Now suppose WMA makes a mistake. Then

$$\begin{aligned}\hat{y} \neq y &\Rightarrow \hat{y} = 1 \Rightarrow q_1 \geq q_0 \\ &\Rightarrow q_1 \geq \frac{W}{2} \\ &\Rightarrow W_{new} \leq W - (1-\beta)\frac{W}{2} = (\frac{1+\beta}{2})W\end{aligned}$$

So if WMA makes a mistake, the sum of weights W will decrease by multiplying $\frac{1+\beta}{2}$. Therefore, after m mistakes, we get an upper bound of the sum of weights as

$$W \leq N \cdot (\frac{1+\beta}{2})^m.$$

Define L_i to be the number of mistakes that expert i makes. Then we have

$$\forall i, w_i = \beta^{L_i} \leq W \leq N \cdot (\frac{1+\beta}{2})^m.$$

Solving this inequality, and noting that it holds for all experts i, we get

$$m \leq \frac{(\min_i L_i)\lg(1/\beta) + \lg(N)}{\lg(\frac{2}{1+\beta})}.$$

□

4 Randomized weighted majority algorithm

The previous proof has shown that, at best, the number of mistakes of WMA is at most twice the number of mistakes of the best expert. This is a weak result if the best expert actually makes a substantial number of mistakes. In order to get a better algorithm, we have to introduce randomness. The next algorithm we are going to introduce is called randomized weighted majority algorithm. This algorithm is very similar to weighted majority algorithm. The only difference is that rather than making prediction by weighted majority vote, in each round, the algorithm picks an expert with some probability according to its weight, and uses that randomly chosen expert to make its prediction. The algorithm is as following:

- w_i = weight on expert i
- Parameter $0 \leq \beta < 1$
- $N = \#$ of experts
- Initially $\forall i, w_i = 1$
- for $t = 1, \cdots, T$
  - each expert i predicts $\xi_i \in \{0, 1\}$
  - $q_0 = \sum_{i:\xi_i=0} w_i$, $q_1 = \sum_{i:\xi_i=1} w_i$.
  - learner predicts $\hat{y} \in \{0, 1\}$

- $\hat{y} = \begin{cases} 1 \quad \text{with probability } \dfrac{\sum_{i:\xi_i=1} w_i}{W} = \dfrac{q_1}{W} \\ 0 \quad \text{with probability } \dfrac{\sum_{i:\xi_i=0} w_i}{W} = \dfrac{q_0}{W} \end{cases}$
- learner observes $y \in \{0, 1\}$
- (mistake if $y \neq \hat{y}$)
- for each i, if $\xi_i \neq y$, then $w_i \leftarrow w_i\beta$

Now let's analyze randomized weight majority algorithm(RWMA) by proving the following theorem:

Theorem 3. *E(# of mistakes of RWMA)* $\leq a_\beta$*(# of mistakes of the best expert)* $+c_\beta \ln(N)$*, where* $a_\beta = \frac{\ln(1/\beta)}{1-\beta}$ *and* $c_\beta = \frac{1}{1-\beta}$.

Notice here we are considering the expectation of number of mistakes RMWA makes because RMWA is a randomized algorithm. When $\beta \to 1$, $a_\beta \to 1$. This means RMWA can do really close to the optimal.

Proof: Similar to the proof of the previous theorem, we prove this theorem by considering the sum of weights W. On some round, define l as the probability that RWMA makes a mistake:

$$l = Pr[\hat{y} \neq y] = \frac{\sum_{i:\xi_i \neq y} w_i}{W}.$$

Then the weight in this round is changed as

$$\begin{aligned} W_{new} &= \sum_{i:\xi_i \neq y} w_i\beta + \sum_{i:\xi_i = y} w_i \\ &= lW\beta + (1-l)W \\ &= W(1 - l(1-\beta)) \end{aligned}$$

Then we can write the sum of weights at the end as

$$\begin{aligned} W_{final} &= N \cdot (1 - l_1(1-\beta)) \cdots (1 - l_t(1-\beta)) \\ &\leq N \prod_{t=1}^{T} \exp(-l_t(1-\beta)) \\ &= N \cdot \exp(-(1-\beta)\sum_{t=1}^{T} l_t) \end{aligned}$$

Then we have

$$\beta^{L_i} \leq w_i \leq W_{final} \leq N \cdot \exp(-(1-\beta)\sum_{t=1}^{T} l_t).$$

Define $L_A = \sum_{t=1}^{T} l_t$ as the expected number of mistakes RWMA makes. We have

$$L_A = \sum_{t=1}^{T} l_t \leq \frac{\ln(1/\beta)}{1-\beta}(\text{\#of mistakes of the best expert}) + \frac{1}{1-\beta}\ln N.$$

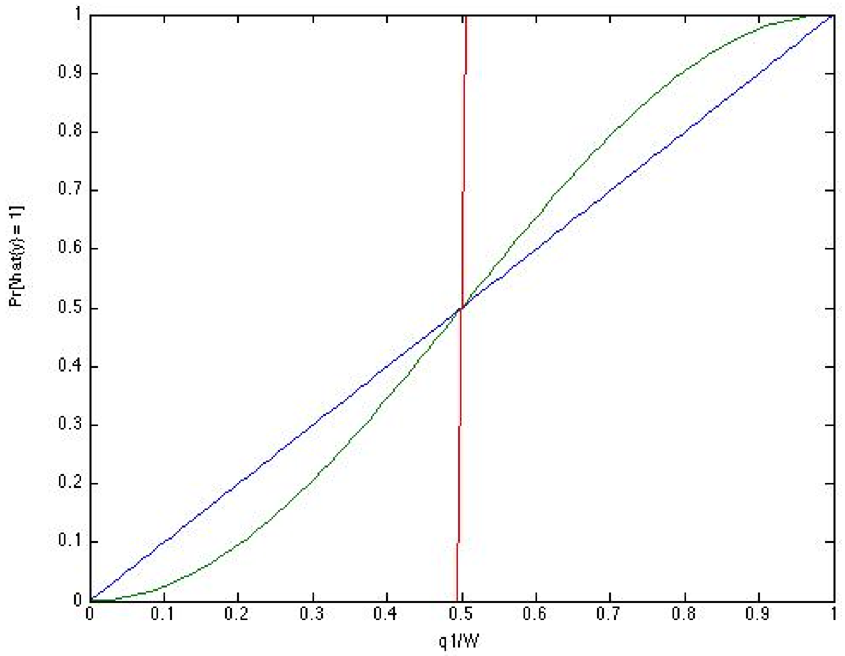
□

5 Variant of RWMA

Let's first discuss how to choose the parameter β for RWMA. Suppose we have an upper on the number of mistakes that the best expert makes as $\min_i L_i \le K$, then we can choose β as $\frac{1}{1+\sqrt{\frac{2\ln(N)}{K}}}$. Then we have

$$L_A \le \min_i L_i + \sqrt{2K\ln(N)} + \ln(N).$$

It is natural to ask whether we can do better than RWMA. The answer is yes. The high level idea is to work in the middle of WMA and RWMA. The following figure shows how this algorithm works. The y-axis is the probability of predicting $\hat{y}$ as 1. And the x-axis is the weighted fraction of experts predicting 1. The red line is WMA, the blue line is RWMA, and the green line is the new algorithm.



The new algorithm can reach $a_\beta = \frac{\ln(1/\beta)}{2\ln(\frac{2}{1+\beta})}$ and $c_\beta = \frac{1}{2\ln(\frac{2}{1+\beta})}$. These are exactly half of those of WMA. And if we make the assumption that $\min_i L_i \le K$ again, the new algorithm has

$$L_A \le \min_i L_i + \sqrt{K\ln(N)} + \frac{\lg(N)}{2}.$$

If we set $K = 0$ which means there exists a "perfect" expert, we have

$$L_A \le \frac{\lg(N)}{2}$$

Thus we know this algorithm is better than the "Halving algorithm".

We usually can assume that $K = \frac{T}{2}$ by adding two experts: one always predicts 1 and the other always predicts 0. Then we have

$$L_A \le \min_i L_i + \sqrt{\frac{T\ln(N)}{2}} + \frac{\lg(N)}{2}.$$

If we divide both sides by T, we get

$$\frac{L_A}{T} \le \frac{\min_i L_i}{T} + \sqrt{\frac{\ln(N)}{2T}} + \frac{\lg(N)}{2T}.$$

When T gets large, the right hand side of the inequality is dominated by the term $\sqrt{\frac{\ln(N)}{2T}}$.

6 Lower bound

Finally we are going to give a lower bound for "learning with expert advice". This lower bound matches the upper bound we get from the previous section even for constant factors. So this lower bound is tight and we cannot improve the algorithm in the previous section.

To prove the lower bound, we consider the following scenario:

- On each round, the adversary chooses the prediction of each expert randomly, $\xi_i = \begin{cases} 1 & \text{w.p. } 1/2 \\ 0 & \text{w.p. } 1/2 \end{cases}$
- On each round, the adversary chooses the feedback randomly, $y = \begin{cases} 1 & \text{w.p. } 1/2 \\ 0 & \text{w.p. } 1/2 \end{cases}$

For any learning algorithm A,

$$E_{y,\xi}[L_A] = E_\xi[E_y[L_A|\xi]] = T/2.$$

For any expert i, we have

$$E[L_i] = T/2.$$

However, we can prove that

$$E[\min_i L_i] \approx \frac{T}{2} - \sqrt{\frac{T\ln(N)}{2}}.$$

Thus,

$$E[L_A] - E[\min_i L_i] \ge \sqrt{\frac{T\ln(N)}{2}}.$$

The term $\sqrt{\frac{T\ln(N)}{2}}$ meets the dominating term in the previous section. Also, in the proof of the lower bound, the adversary only uses entirely random expert predictions and outcomes. So we would not gain anything by assuming that the data is random rather than adversarial. The bounds we proved in an adversarial setting are just as good as could possibly be obtained even if we assumed that the data are entirely random.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #16
Scribe: Yannan Wang — April 3, 2014

1 Introduction

The goal of our online learning scenario from last class is C comparing with best expert and do as well as the best expert.

An alternative scenario is there is no single good expert. But we could form a committee of experts and they might be much better.

We'll formalize this as follows:

- We have N experts.
- For $t = 1, ..., T$ we get $\mathbf{x}_t \in \{1, -1\}^N$
  Note: $\mathbf{x}_t$: a set of predictions
  N: dimension
  i_{th} component: prediction of expert i
- In each round, learner predicts $\hat{y_t} \in \{1, -1\}$
- In each round, we observe the outcome $y_t \in \{1, -1\}$
- The above is the same; what we changed is the assumption of data. We assume that there is a perfect committee, i.e. a weighted sum of experts that are always right. Formally, this means that $\mathbf{u} \in \mathbb{R}^N$,

$$\forall t : y_t = sign(\sum_{i=1}^{N} u_i x_{t,i}) = sign(\mathbf{x}_t \cdot \mathbf{u})$$

$$\iff y_t(\mathbf{u} \cdot \mathbf{x}_t) > 0$$

  Geometrically, the perfect committee means that there is a linear threshold that separates the 1 points and -1 points, generated by the appropriate weighted sum of the experts.

2 How to do updates

We are focusing on $\mathbf{w}_t$, the prediction of $\mathbf{u}$. It is sort of a "guess" of the correct weighting of the experts. We will update the weighting on each round. Today we are looking at two algorithms. For each algorithm, we only need to focus on

$$(1) initialize$$

$$(2) update$$

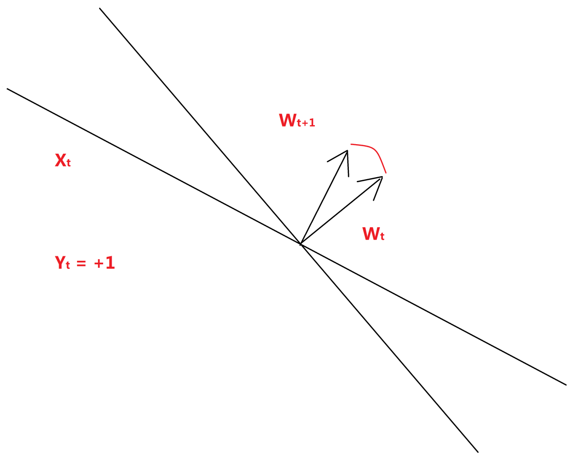


Figure 1: Perceptron geometric intuition: tiping the hyperplane

2.1 Perceptron

The first way to update weights will give us an algorithm called Perceptron. The update rules are as follows:

- *Initialize*: $\mathbf{w}_1 = \mathbf{0}$
- *Update*: If mistake($\iff \hat{y}_t \neq y_t \iff y_t(\mathbf{w}_t \cdot \mathbf{x}_t) \leq 0$),

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_t\mathbf{x}_t$$

  else,

$$\mathbf{w}_{t+1} = \mathbf{w}_t$$

  Not adjusting the weights when there are no mistakes makes the algorithm *conservative*; the algorithm ignores the correctly classifying samples.

The intuition is that in case of a wrong answer we "shift" the weights on all the experts in the direction of the correct answer. Figure 1 gives a geometrical intuition of the Perceptron algorithm. Here $y_t = +1$, when $(\mathbf{x}_t, y_t)$ is classified incorrectly, then we add $\mathbf{x}_t y_t$ to $\mathbf{w}_t$ to such a direction that is more likely to correctly classify $(\mathbf{x}_t, y_t)$ next time; we are shifting the hyperplane defined by $\mathbf{w}_t$ in such a direction that we are more likely to correctly classify $\mathbf{x}_t$.

Now let's state a theorem to formally analyze the performance of the Perceptron algorithm. However, first we will make a a few assumptions:

- Mistakes happens in every round. This is because no algorithmic change happens during other rounds. So: $T = \#$ of rounds $= \#$ of mistakes.
- We normalize the vector of predictions $\mathbf{x}_t$, so that $\|\mathbf{x}_t\|_2 \leq 1$.

- We normalize the vector of weights for the perfect committee, so that$\|\mathbf{u}\|_2 = 1$. (This is fine because the value of the sign function will not be affected by this normalization.)
- We make the assumption that the points are linearly separable with margin at least δ: $\exists \delta, \mathbf{u} \in \mathrm{I\!R}^{\mathrm{N}}, \forall \mathrm{t} : \mathrm{y_t}(\mathbf{u} \cdot \mathbf{x}_\mathrm{t}) \geq \delta > 0$. Note that this assumption is with loss of generality.

Theorem 2.1 *Under the assumptions above, $T = \#$ mistakes, we have*

$$T \leq \frac{1}{\delta^2}$$

Proof : In order to prove this, we will find some quantity that depends on the state of the algorithm at time t, upper bound and lower bound it, and derive a bound from there. The quantity here is Φ_t, which is cosine of the angle Θ between $\mathbf{w}_t$ and $\mathbf{u}$. More formally,

$$\Phi_t = \frac{\mathbf{w}_t \cdot \mathbf{u}}{\|\mathbf{w}_t\|_2} = \cos \Theta \leq 1$$

Now for the lower bound, we will prove that

$$\Phi_{T+1} \geq \sqrt{T}\delta$$

We will do this in two parts — by lower bounding the numerator of Φ_t and by upper bounding the denominator.

- **step 1: $\mathbf{w}_{T+1} \cdot \mathbf{u} \geq T\delta$:**

$$\mathbf{w}_{t+1} \cdot \mathbf{u} = (\mathbf{w}_t + y_t\mathbf{x}_t) \cdot \mathbf{u} = \mathbf{w}_t \cdot \mathbf{u} + y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \mathbf{w}_t \cdot \mathbf{u} + \delta$$

  The inequality is by the $4th$ assumption above. Initially we have set $\mathbf{w_1} \cdot \mathbf{u} = 0$, thus the above bound implies that $\mathbf{w}_{T+1} \cdot \mathbf{u} \geq T\delta$.

- **step 2: $\|\mathbf{w}_{T+1}\|^2 \leq T$:**

$$\|\mathbf{w}_{t+1}\|^2 = \mathbf{w}_{t+1} \cdot \mathbf{w}_{t+1} = (\mathbf{w}_t + y_t\mathbf{x}_t) \cdot (\mathbf{w}_t + y_t\mathbf{x}_t) = \|\mathbf{w}_t\|_2^2 + 2y_t(\mathbf{x}_t \cdot \mathbf{w}_t) + \|\mathbf{x}_t\|^2$$

Since we have made the assumption that we get a mistake at each round, $y_t(\mathbf{x}_t \cdot \mathbf{w}_t) \leq 0$, and from the normalization assumption, $\|\mathbf{x}_t\|_2^2 \leq 1$, so that we get $\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_t\|_2^2 + 1$. Initially we have set $\|\mathbf{w}_1\|_2^2 = 0$, so we get $\|\mathbf{w}_{T+1}\|_2^2 \leq T$

Now we put step 1 and step 2 together, $1 \geq \Phi_{T+1} \geq \frac{T\delta}{\sqrt{T}}$, i.e. $T \leq \frac{1}{\delta^2}$. $\square$

Let $\mathcal{H}$ be the hypothesis space and $M_{perceptron}(\mathcal{H})$ be the number of mistakes made by the Perceptron algorithm. As a simple consequence of the above, since the VC dimension of the hypothesis space is upper bounded by the number of mistakes the algorithm makes, we get the VC dimension of threshold functions with margin at least δ is at most $\frac{1}{\delta^2}$:

$$VC\text{-}dim(\mathcal{H}) \leq opt(\mathcal{H}) \leq M_{perceptron}(\mathcal{H}) \leq \frac{1}{\delta^2}$$

Now consider a scenario where the target $\mathbf{u}$ consists of 0s and 1s, and the number of 1s in the vector is k.

$$\mathbf{u} = \frac{1}{\sqrt{k}}(0 \quad 1 \quad 0 \quad 0 \quad 1 \quad ...)$$

Note that here $\frac{1}{\sqrt{k}}$ is for normalization. Think of k as being small compared to N, the number of experts, i.e. it could be a very sparse vector. This is also one example of the problems we earlier examined — the k experts are the "perfect" committee. We have,

$$\mathbf{x}_t = \frac{1}{\sqrt{N}}(+1, \quad -1, \quad -1, \quad +1, \quad ...)$$

$$y_t = sign(\mathbf{u} \cdot \mathbf{x}_t)$$

$$y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \frac{1}{\sqrt{kN}}$$

Note that here $\frac{1}{\sqrt{N}}$ is for normalization. So using $\frac{1}{\sqrt{kN}}$ as δ, by Theorem 2.1, the Perceptron algorithm would make at most kN mistakes. However this is not good — consider interpreting the experts as features, and we have millions of irrelevant features, and the committee is the important (maybe a dozen) features. We get a linear dependencies on N, which is usually large.

Motivated by this example, we present another update algorithm, called the Winnow algorithm, which will get a better bound.

2.2 Winnow Algorithm

- **Initialize**:

  $$\forall i, w_{1,i} = \frac{1}{N}$$

  we start with a uniform distribution over all experts.

- **Update:** If we make a mistake,

  $$\forall i : w_{t+1,i} = \frac{w_{t,i} \cdot e^{\eta y_t x_t}}{Z_t}$$

  Here η is a parameter we will define later, and Z_t is a normalization factor. Else,

  $$\mathbf{w}_{t+1} = \mathbf{w}_t$$

This update rule is like exponential punishment for the experts that are wrong. If we ignore the normalization factors, the above update is equivalent to $w_{t+1,i} = w_{t,i}e^{\eta}$, if i predicts correctly, and $w_{t+1,i} = w_{t,i}e^{-\eta}$ otherwise. Ignoring the normalization factor, we could see it as $w_{t+1,i} = w_{t,i}$, if i predicts correctly, and $w_{t+1,i} = w_{t,i}e^{-2\eta}$ otherwise. This is the same as the weighted majority vote.

Before stating the formal theorem for the Winnow algorithm, we make a few assumptions without loss of generality:

- We make mistake at every round.
- $\forall t : \|\mathbf{x}_t\|_\infty \leq 1$.
- $\exists \delta, \mathbf{u} : \forall t : y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \delta > 0$.
- $\| \mathbf{u} \|_1 = 1$ and $\forall i$: $u_i \geq 0$.

Notice here we used L_1 and L_∞ norm here instead of the L_2 norm that we used in Perceptron algorithm.

Theorem 2.2 *Under the assumptions above, We have the following upper bound on the number of mistakes:*

$$T \leq \frac{\ln N}{\eta\delta + \ln(\frac{2}{e^{\eta}+e^{-\eta}})}$$

If we choose an optimal η to minimize the bound , we get when $\eta = \frac{1}{2}\ln(\frac{1+\delta}{1-\delta})$,

$$T \leq \frac{2\ln N}{\delta^2}$$

Proof The approach is similar to the previous one. We use a quantity Φ_t, which we both upper and lower-bound. The quantity we use here is $\Phi_t = RE(\mathbf{u} \parallel \mathbf{w}_t)$. Immediately we have, $\Phi_t \geq 0$ for all t.

$$\begin{aligned} \Phi_{t+1} - \Phi_t &= \sum_i u_i \ln(\frac{u_i}{w_{t+1,i}}) - \sum_i u_i \ln(\frac{u_i}{w_{t,i}}) \\ &= \sum_i u_i \ln(\frac{w_{t,i}}{w_{t+1,i}}) \\ &= \sum_i u_i \ln(\frac{Z_t}{e^{\eta y_t x_{t,i}}}) \\ &= \sum_i u_i \ln Z_t - \sum_i u_i \ln e^{\eta y_t x_{t,i}} \\ &= \ln Z_t - \eta y_t(\mathbf{u}\cdot\mathbf{x}_t) \\ &\leq \ln Z_t - \eta\delta \end{aligned} \tag{1}$$

The last inequality follows from the margin property we assumed. Now let's approximate Z_t. We know that Z is the normalization factor and can be computed as:

$$Z = \sum_i w_i e^{\eta y x_i} \tag{2}$$

Note that here we are dropping the subscript t for simplicity; Z and w_i are same as Z_t and $w_{t,i}$. We will bound the exponential term by a linear function, as illustrated in figure 2:

$$e^{\eta x} \leq (\frac{1+x}{2})e^{\eta} + (\frac{1-x}{2})e^{-\eta}, \quad for -1 \leq x \leq 1.$$

Using this bound, we have:

$$\begin{aligned} Z &= \sum_i w_i e^{\eta y x_i} \\ &\leq \sum_i w_i(\frac{1+yx_i}{2})e^{\eta} + \sum_i w_i(\frac{1-yx_i}{2})e^{-\eta} \\ &= \frac{e^{\eta}+e^{-\eta}}{2}\sum_i w_i + \frac{e^{\eta}-e^{-\eta}}{2} y \sum_i w_i x_i \\ &= \frac{e^{\eta}+e^{-\eta}}{2} + \frac{e^{\eta}+e^{-\eta}}{2} y(\mathbf{w}\cdot\mathbf{x}) \\ &\leq \frac{e^{\eta}+e^{-\eta}}{2} \end{aligned} \tag{3}$$

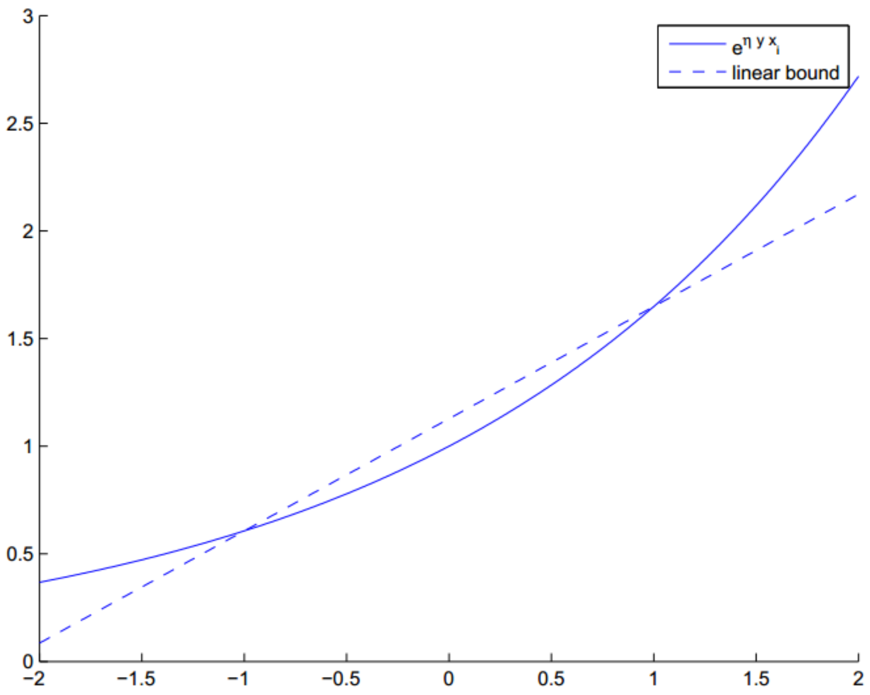


Figure 2: Using linear function to bound exponential function

The last inequality comes from the assumption that the expert makes a wrong prediction every time, so the second term is less than 0. So we have,

$$\begin{aligned}\Phi_{t+1} - \Phi_t &\leq \ln Z_t - \eta\delta \\ &\leq \ln(\frac{e^{\eta} + e^{-\eta}}{2}) - \eta\delta = -C\end{aligned} \tag{4}$$

Note that here $\ln(\frac{e^{\eta}+e^{-\eta}}{2}) - \eta\delta$ is a constant and let's make it equals $-C$. So for each round Φ_t is decreasing by at least $C = \ln(\frac{2}{e^{\eta}+e^{-\eta}}) + \eta\delta$.

In the next class, we will finish the proof of Theorem 2.2 and we will study a modified version of Winnow Algorithm called Balanced Winnow Algorithm that gets rid of the assumption that $\forall i : u_i \geq 0$.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture # 17
Scribe: Madhuvanthi Jayakumar — April 08, 2013

1 Review of Winnow Algorithm

Algorithm 1 Winnow.

```
procedure WINNOW
    $\forall i : w_{1,i} = \frac{1}{N}$
    for $t = 1, \ldots, T$ do
        Get $\mathbf{x}_t \in \mathbb{R}^N$
        Predict $\hat{y}_t = sign(\mathbf{w}_t \cdot \mathbf{x}_t)$
        Get $y_t \in \{-1, +1\}$
        Update, if mistake: $w_{t+1,i} = \frac{w_{t,i}}{Z_t} e^{\eta y_t x_{t,i}}$
```

Comments about algorithm:

- $\mathbf{w}_t$ is the weight vector. We can view it as a probability distribution (non-negative values, and sums up to 1). Initially it is uniformly distributed.
- $\mathbf{x}_t$ can be thought of as a point in space or as an N-dimensional vector
- If there is no mistake, the weight is carried over to the next iteration: $w_{t+1} = w_t$

2 Upperbounding Number of Mistakes for Winnow

Assumptions:

1. A mistake is made on every round.
2. $\forall t : \|\mathbf{x}_t\|_\infty \leq 1$
3. $\exists \delta, \mathbf{u}, \forall t : y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \delta > 0$
4. $\forall t : \|\mathbf{u}\|_1 = 1$
5. $\forall i : u_i \geq 0$

Theorem 2.1. *Under the assumptions made above, we can bound the maximum number of* *mistakes that can be made by the algorithm to* $\frac{2\ln N}{\delta^2}$ *if* $\eta = \frac{1}{2}\ln\frac{1+\delta}{1-\delta}$

$$\begin{aligned}\#\ \textit{mistakes} &\leq \frac{\ln N}{\eta\delta + \ln\left(\frac{2}{e^{\eta}+e^{-\eta}}\right)} \\ &\leq \frac{2\ln N}{\delta^2} \\ &\textit{if} \quad \eta = \frac{1}{2}\ln\left(\frac{1+\delta}{1-\delta}\right)\end{aligned} \tag{1}$$

Proof. Last time, we showed:

$$\Phi_t = RE(\mathbf{u}||\mathbf{w}) \geq 0 \tag{2}$$

$$\Phi_{t+1} - \Phi_t \leq -c \tag{3}$$

$$\text{where } c = \eta\delta + \ln\frac{2}{e^{\eta}+e^{-\eta}}$$

Today, we will bound the number of mistakes by bounding the total change in potential. We first upperbound Φ_1, the potential on the first round.

$$\begin{aligned}\Phi_1 &= RE(\mathbf{u}||\mathbf{w}) \\ &= \sum u_i \ln(u_i N) \\ &\leq \sum u_i \ln(N) \\ &= \ln N\end{aligned} \tag{4}$$

At this point we have upper bounded the potential for the first round. We also know the following two things:

1. The minimum change in each potential is c, and since there are T rounds
   - This imiplies that the minimum total change in potential has to be at least cT.
2. The potential can never be negative (and $\Phi_1 \leq \ln N$).
   - This implies that the maximum total change in potential can be at most $\ln N$.

Hence, we can upperbound the maximum number of iterations of the algorithm in terms of N and c in the following way.

$$cT \leq \text{total change} \leq \ln N$$
$$cT \leq \ln N$$
$$T \leq \frac{\ln N}{c}$$

Since our first assumption is that there is a mistake on every round, the number of mistakes is also bounded by this value. We have thus proved Theorem 2.1 to bound the number of mistakes in the algorithm which does not depend on probability assumptions.

Minimizing bound: We would like to set the value of c such that the bound is minimized. To do this, we get the value of η that will result in the minimum bound by taking the derivative and setting it equal to 0. Solving for η we get:

$$\eta = \frac{1}{2}\ln\frac{1+\delta}{1-\delta}$$

Plugging this back into our equation for c, we get that:

$$c = RE(\tfrac{1}{2} - \tfrac{\delta}{2} || \tfrac{1}{2})$$

Since we are taking the Relative Entropy between two bernoulli items, we can reduce it (to 2 times the difference between the two bernoulli items) to get:

$$c \geq 2 \cdot (\tfrac{\delta}{2})^2 = \tfrac{\delta^2}{2}$$

We have thus proved the bound in the theorem:

$$T \leq \frac{\ln N}{c} \leq \frac{2\ln N}{\delta^2}$$

□

3 Balanced Winnow

Now we would like to remove the fifth assumption that $\forall i : u_i \geq 0$

The quick and dirty way of doing this is to modify the two vectors, **x** and **u**. Say we have vectors x and u as given below. We modify **x** to get $\mathbf{x}'$ by creating two copies of **x** and modifying the second copy such that their signs are negated. We modify **u** to get $\mathbf{u}'$ by creating two copies of u and modifying the second copy such that all the negative components are zero.

$\mathbf{x} = (1 \quad 0.7 \quad -0.4) \quad \rightarrow \quad \mathbf{x}' = (1 \quad .7 \quad -.4 \quad | \quad -1 \quad -.7 \quad .4)$

$\mathbf{u} = (0.5 \quad -0.2 \quad 0.3) \quad \rightarrow \quad \mathbf{u}' = (0.5 \quad 0 \quad .3 \quad | \quad 0 \quad .2 \quad 0)$

We've doubled the number of components but the other three previously made assumptions still hold, as shown:

Assumptions:

1. A mistake is made on every round: yes.
2. $\forall t : \|\mathbf{x'}_t\|_\infty \leq 1$
   - $\|\mathbf{x'}_t\|_\infty$ is also unchanged since we haven't changed the maximum absolute value.
3. $\exists \delta, \mathbf{u}', \forall t : y_t(\mathbf{u}' \cdot \mathbf{x'}_t) \geq \delta > 0$
   - Inner product is unchanged: $\mathbf{x}' \cdot \mathbf{u}' = \mathbf{x} \cdot \mathbf{u}$. The dot product on non-negative portion stays the same. The dot product of the negative portion cancels out because it is positive but taking the inner product with negative of $\mathbf{x}_t$.
4. $\forall t : \|\mathbf{u}'\|_1 = 1$
   - $\|\mathbf{u}'\|_1$ is also unchanged since we take the absolute value of each component exactly once.

4 Compare Perceptron and Winnow/WMA

Perceptron	Winnow/WMA
Additive Update	Multiplicative update
$\|\mathbf{x}_t\|_2 \leq 1$	$\|\mathbf{x}_t\|_\infty \leq 1$
$\|\mathbf{u}\|_2 = 1$	$\|\mathbf{u}\|_1 \leq 1$
SVM	Adaboost

5 Regression

Until now, we have been learning to classify objects as labels 0/1 or -1/1. Our goal was to minimize the number of mistakes made by the algorithm or minimze the probability of making mistakes. We talked about how PAC outputs a hypothesis whose probability of making a mistake we wanted to be low. With online learning algorithms, we wanted the number of mistakes made by the algorithm to be low. We were able to evaluate these by looking at the labels. Now we want to switch focus and have a different goal which is not just to get the correct label.

5.1 Example

We will introduce this topic with an example. Say we are looking to hire Alice or Bob to predict the weather. We ask each of them to predict the probability that it rains. Alice says the probability of it raining is 70% and Bob says it is 80%. We see the outcome (that it rains) but we don't know the underlying probability, so we can't say whether Alice or Bob was closer to the actual probability. In the following sections, we explore how to come up with a percentage that is close to the true probabilities even though we can never observe true probabilities.

First we will formally state the problem. We will then create a model for scoring the prediction and state and prove a theorem that shows why the model works.

5.2 Formal Statement of Problem

We are given the weather conditions $\mathbf{x}$, and we would like to predict the value of y, which is 1 if it rains, and 0 otherwise. We obtain both $\mathbf{x}$ and y from distribution D, $(\mathbf{x}, y) \sim D$. Our goal is to estimate $Pr[y = 1|\mathbf{x}]$. We never get to observe this, we only get the outcome y. We define $p(\mathbf{x}) = Pr[y = 1|\mathbf{x}] = \mathbb{E}[y|\mathbf{x}]$. We use expectation to have a more general problem statement, that allows y to be any real number and not necessarily restrict it to 0 or 1. This problem is called regression. We may define $h_A(\mathbf{x})$ as Alice's estimate of the porbability of rain given x, and $h_B(\mathbf{x})$ as Bob's estimate of the probability of rain given $\mathbf{x}$.

5.3 Model

We define **square/quadratic loss** as $(h(x) - y)^2$ and we use this to score how good the prediction is. Taking the difference of the hypothesis and outcome is a natural way to characterize this and squaring gives nicer mathematical properties (differentiable, etc). We define risk to be the expected loss, $\mathbb{E}[(h(x)-y)^2]$, and choose h that minimizes the expected value over x, y. We define the risk with respect to the true distribution D as the true risk. We will show how to estimate this from samples, by looking at a set of predictions and outcomes and taking the average loss. Theorem 5.1 and its associated proof shows why minimizing this expectation leads to a good prediction.

First we fix $\mathbf{x}$ and define the following:

$$h = h(\mathbf{x}) \text{ and } p = p(\mathbf{x}) = Pr[y = 1|\mathbf{x}]$$

Then we have that:

$$\mathbb{E}[(h - y)^2] = p(h - 1)^2 + (1 - p)h^2.$$

This comes from the definition of expectation: Pr[y=1] · resulting loss + P[y=0] · resulting loss.
To minimize the expectation, we take the derivative with respect to h and set it equal to 0.

$$\frac{d\mathbb{E}}{dh} = 2(h - p) = 0, \text{ resulting in } h = p.$$

This implies that the loss is minimized when we set $h = p$. We cannot directly observe p, but this result shows that minimizing the loss will lead us to choose h which is equal to p. Now we will state a theorem that is more general and stronger that applies to any value of $\mathbf{x}$.

Theorem 5.1. $\mathbb{E}_x[(h(\mathbf{x}) - p(\mathbf{x}))^2] = \mathbb{E}_{\mathbf{x},y}[(h(x) - y)^2] - \mathbb{E}_{x,y}[(p(\mathbf{x}) - y)^2]$

Note:

- The first term is the loss/risk we can measure to estimate from data.
- The second term measures the intrinsic noise of y.
- Our goal is to minimize $\mathbb{E}[(h(\mathbf{x}) - p(\mathbf{x}))^2]$.
  - Since the variance of y does not depend on h, we have that $\mathbb{E}[(h(\mathbf{x}) - p(\mathbf{x}))^2] = \mathbb{E}[(h(\mathbf{x}) - y)^2] -$constant
  - Therefore minimizing $\mathbb{E}[(h(\mathbf{x}) - y)^2]$ also minimizes $\mathbb{E}[(h(\mathbf{x}) - p(\mathbf{x}))^2]$ and this justifies the use of square loss to get the best prediciton.

Proof. We will prove Theorem 5.1 for fixed $\mathbf{x}$. Then we can use linearity of expectations to show that it holds for any expectation of $\mathbf{x}$. We define $h = h(\mathbf{x})$, and $p = p(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \mathbb{E}[y]$. We will modify the LHS and RHS of the equations to show that they are both equal to each other.

$$\text{LHS} = (h - p)^2.$$

Explanation: we can remove the expectation because x is fixed.

$$\begin{aligned} \text{RHS} &= \mathbb{E}[(h - y)^2] - \mathbb{E}[(p - y)^2] \\ &= \mathbb{E}[(h^2 - 2hy + y^2) - (p^2 - 2py + y^2)] \\ &= h^2 - 2h\mathbb{E}[y] - p^2 + 2p\mathbb{E}[y] \\ &= h^2 - 2hp + p^2 \\ &= (h - p)^2 \\ &= \text{LHS} \end{aligned} \tag{5}$$

□

6 Linear Regression

We can estimate the smallest value of $\mathbb{E}[(h(x)-y)^2]$ by looking at the empirical average of the given sample. Given m samples, $(x_1, y_1)...(x_m, y_m) \sim D$:

$$\hat{\mathbb{E}}[(h(x)-y)^2] = \tfrac{1}{m}\sum_{i=1}^{m} h(x_i - y_i)^2 = \mathbb{E}[L_h].$$

This is the empirical risk that can be measured using training data. We define

$$L_h(x,y) = (h(x)-y)^2$$

Now we do the following two things:

1. Prove/argue w.h.p. $\forall h \in H$, $\hat{\mathbb{E}}[L_h] \approx \mathbb{E}[L_h]$. This is a uniform convergence problem.
2. Minimize $\hat{\mathbb{E}}[L_h]$. This is a computational problem.

6.1 Example

Suppose we are given m examples, $(x_1, y_1)...(x_m, y_m)$ with $x_i \in \mathbb{R}^n$ and labels $y_i \in R$
Our hypothesis is of the form, $h(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x}$, which is linear for some fixed vector $\mathbf{w}$
We then have:

$$\begin{aligned}\hat{\mathbb{E}}[L_h] &= \frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}_i)-y_i)^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}(\mathbf{w}\cdot\mathbf{x}_i - y)^2 \qquad (6)\end{aligned}$$

We would like to find the $\mathbf{w}$ that minimizes this quantity. This is called linear regression. We will work with this in matrix form, defining M to be the matrix of vectors $\mathbf{x}$.

$$\begin{pmatrix} \leftarrow & \mathbf{x}_1^T & \rightarrow \\ \leftarrow & \mathbf{x}_2^T & \rightarrow \\ & ... & \\ \leftarrow & \mathbf{x}_m^T & \rightarrow \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ ... \\ w_n \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{pmatrix}$$

$$M\mathbf{w} - \mathbf{b}$$

$$\left\| \begin{matrix} \mathbf{w}\cdot\mathbf{x}_1 - y_1 \\ \mathbf{w}\cdot\mathbf{x}_2 - y_2 \\ ... \\ \mathbf{w}\cdot\mathbf{x}_m - y_m \end{matrix} \right\|_2^2 = ||M\mathbf{w} - \mathbf{b}||_2^2$$

The Euclidean length squared will give us the sum of squared errors, $\sum_{i=1}^{m}(\mathbf{w}\cdot\mathbf{x}_i - y_i)^2$
Now we find $\mathbf{w}$ that minimizes this. To do this, we compute the gradient and set it equal to 0 and solve for $\mathbf{w}$.

$$\nabla\Phi = 2M^T(M\mathbf{w} - \mathbf{b}) = 0$$
$$\mathbf{w} = (M^TM)^{-1}M^T\mathbf{b}$$

$(M^TM)^{-1}M^T$is known as the pseudo inverse of M^T.

7 Combining Regression with Online Learning

Now we take a look at what regression looks like in an online learning setting.

Algorithm 2 Regression with Online Learning

```
procedure REGRESSION
    Initialize $\mathbf{w}_1$
    for $t = 1, \dots, T$ do
        get $\mathbf{x}_t \in \mathbb{R}^N$
        predict $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t \in R$
        get $y_t \in R$
        loss = $(\hat{y}_t - y_t)^2$
        update $\mathbf{w}_t$
```

Our goal is to minimize loss, $L_A = \sum_{t=1}^{T} (\hat{y}_t - y_t)^2$

We are interested in updating the weight $\mathbf{w}_t$ and using it in a linear way to make predictions without making any randomness assumptions. We analyze the loss suffered using one particular weight vector $u \in \mathbb{R}^N$:

- Predict: $\mathbf{u} \cdot \mathbf{x}_t$
- Loss: $(\mathbf{u} \cdot \mathbf{x}_t - \mathbf{y}_t)^2$
- Cumulative loss $\sum_{t=1}^{T} (\mathbf{u} \cdot \mathbf{x}_t - y_t)^2$

We would like to achieve the result:

$$L_A \leq minL_u + \text{small amount: "regret"}$$

$$\text{where } L_u = \sum_{t=0}^{T} (\mathbf{u} \cdot \mathbf{x}_t - y_t)^2$$

L_u is the loss of a linear predictor $\mathbf{u}$. The above inequality for L_A says that if there exists any predictor that gives good predictions, then our algorithm performs close to that predictor.

8 Widrow-Hoff (WH) Algorithm

Towards the end of class, we introduced the Widrow Hoff Algorithm, which is a particular kind of online regression algorithm. This algorithm uses the weight vector in the following way:

- intialize: $\mathbf{w}_1 = 0$
- update: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta(\mathbf{w}_t \cdot \mathbf{x}_t - y_t)x_t$

There are two main motivations for this algorithm. We discuss the first one here and will continue the rest during the next lecture.
We define our loss function as:

$$L(\mathbf{w}, \mathbf{x}, y) = (\mathbf{w} \cdot \mathbf{x} - y)^2$$

The gradient descent of this is:

$$\nabla_{\mathbf{w}} L = 2(\mathbf{w} \cdot \mathbf{x} - y) \cdot \mathbf{x}$$

We have $\mathbf{w_t}$ and we use $\mathbf{x_t}$ and y_t to improve $\mathbf{w}_{t+1}$ so that loss will be slightly smaller on the example that we just observed. The equation below moves $\mathbf{w}$ in the direction of the gradient, which minimizes the loss function.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \tfrac{1}{2}\eta \nabla_{\mathbf{w}} L(\mathbf{w}_t, \mathbf{x}_t, y_t)$$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Shaoqing Yang

Lecture # 18
April 10, 2014

1 Widrow-Hoff Algorithm

First let's review the Widrow-Hoff algorithm that was covered from last lecture:

Algorithm 1: Widrow-Hoff Algorithm

```
Initialize parameter $\eta > 0$, $\mathbf{w}_1 = \mathbf{0}$
for $t = 1 \dots T$
    get $\mathbf{x}_t \in \mathbb{R}^n$
    predict $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t \in \mathbb{R}$
    observe $y_t \in \mathbb{R}$
    update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta(\mathbf{w}_t \cdot \mathbf{x}_t - y_t) \cdot \mathbf{x}_t$
```

And we define the loss functions as $L_A = \sum_{t=1}^{T}(\hat{y}_t - y_t)^2$. And $L_{\mathbf{u}} = \sum_{t=1}^{T}(\mathbf{u} \cdot \mathbf{x}_t - y_t)$. What we want is

$$L_A \leq \min_{\mathbf{u}} L_{\mathbf{u}} + \mathit{small}$$

There are 2 goals in choosing the update function to be $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta(\mathbf{w}_t \cdots \mathbf{x}_t - y_t) \cdot \mathbf{x}_t$: (1) Want loss of $\mathbf{w}_{t+1}$ on $\mathbf{x}_t$, y_t to be small. This means we want to minimize $(\mathbf{w}_{t+1} \cdot \mathbf{x}_t - y_t)^2$ (2) Want $\mathbf{w}_{t+1}$ close to $\mathbf{w}_t$ so that we do not forget everything we learnt so far. And this means we want to minimize $\|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2$.

Therefore to sum up, we want to minimize

$$\eta(\mathbf{w}_{t+1} \cdot \mathbf{x}_t - y_t)^2 + \|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2$$

If we take the derivative of the above equation and set it to zero, we have

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta(\mathbf{w}_{t+1} \cdot \mathbf{x}_t - y_t) \cdot \mathbf{x}_t$$

Instead of solving $\mathbf{w}_{t+1}$, we approximate the term $\mathbf{w}_{t+1}$ inside the parenthesis and change it to $\mathbf{w}_t$. The reason we can do this is because $\mathbf{w}_{t+1}$ does not change much from $\mathbf{w}_t$. Therefore we have

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta(\mathbf{w}_t \cdot \mathbf{x}_t - y_t) \cdot \mathbf{x}_t$$

which is the update function stated in the algorithm.

Now let's state a theorem:

Theorem 1.1 If we assume on every round t, $\|\mathbf{x}_t\|_2 \leq 1$, then:

$$L_{WH} \leq \min_{\mathbf{u} \in \mathbb{R}^n} [\frac{L_{\mathbf{u}}}{1 - \eta} + \frac{\|\mathbf{u}\|_2^2}{\eta}]$$

From this theorem, we have $\forall \mathbf{u}$:

$$L_{WH} \le \frac{1}{1-\eta} \cdot L_{\mathbf{u}} + \frac{\|\mathbf{u}\|_2^2}{\eta}$$

If we divide T on both side, we have:

$$\frac{L_{WH}}{T} \le \frac{1}{1-\eta} \cdot \frac{L_{\mathbf{u}}}{T} + \frac{\|\mathbf{u}\|_2^2}{\eta T}$$

The term $\frac{\|\mathbf{u}\|_2^2}{\eta T}$ goes to 0 when T gets large. And we can choose η small enough to make $\frac{1}{1-\eta}$ to be close to 1. Therefore we have the rate that the algorithm is suffering loss is close to rate that $L_{\mathbf{u}}$ is suffering loss.

Now let's prove the theorem:

Proof: Pick any $\mathbf{u} \in \mathbb{R}^n$. First let's define some terms:

$$\begin{aligned}
\Phi_t &= \|\mathbf{w}_t - \mathbf{u}\|_2^2 && \text{(measure of progess)}\\
l_t &= \mathbf{w}_t \cdot \mathbf{x}_t - y_t = \hat{y}_t - y_t && \text{(notice } l_t^2 \text{ is the loss of WH on round } t\text{)}\\
g_t &= \mathbf{u} \cdot \mathbf{x}_t - y_t && \text{(} g_t^2 \text{ is the loss of } \mathbf{u} \text{ on round } t\text{)}\\
\boldsymbol{\Delta}_t &= \eta(\mathbf{w}_t \cdot \mathbf{x}_t - y_t) \cdot \mathbf{x}_t = \eta l_t \mathbf{x}_t\\
\mathbf{w}_{t+1} &= \mathbf{w}_t - \boldsymbol{\Delta}_t
\end{aligned}$$

Our main claim is that the change of potential is:

$$\Phi_{t+1} - \Phi_t \le -\eta l_t^2 + \frac{\eta}{1-\eta} \cdot g_t^2 \tag{1}$$

This shows that l_t^2 tends to drive potential down while g_t^2 tends to drive potential up.

Now assume (1) holds. Notice that total change in potential should be non-negative. And also we initialize $\mathbf{w}_1 = \mathbf{0}$. So we have the following inequality:

$$\begin{aligned}
-\|\mathbf{u}\|_2^2 = -\Phi_1 &\le \Phi_{T+1} - \Phi_1\\
&= \Phi_{t+1} - \Phi_t + \Phi_t - \Phi_{t-1} + \cdots + \Phi_2 - \Phi_1\\
&= \sum_{t=1}^{T} (\Phi_{t+1} - \Phi_t)\\
&\le \sum_{t=1}^{T} [-\eta l_t^2 + \frac{\eta}{1-\eta} g_t^2]\\
&= -\eta \sum_t l_t^2 + \frac{\eta}{1-\eta} \sum_t g_t^2\\
&= -\eta L_{WH} + \frac{\eta}{1-\eta} L_{\mathbf{u}}
\end{aligned}$$

Now we solve for L_{WH}, we get

$$L_{WH} \le \frac{1}{1-\eta} L_{\mathbf{u}} + \frac{\|\mathbf{u}\|_2^2}{\eta}$$

And since this inequality holds for all $\mathbf{u}$, we have:

$$L_{WH} \le \min_{\mathbf{u}\in\mathbb{R}}[\frac{1}{1-\eta}L_{\mathbf{u}} + \frac{\|\mathbf{u}\|_2^2}{\eta}]$$

which is the theorem.

Now let's go back to prove (1):

$$\begin{aligned}
\Phi_{t+1} - \Phi_t &= \|\mathbf{w}_{t+1} - \mathbf{u}\|^2 - \|\mathbf{w}_t - \mathbf{u}\|^2 \\
&= \|\mathbf{w}_t - \mathbf{u} - \mathbf{\Delta}_t\|^2 - \|\mathbf{w}_t - \mathbf{u}\|^2 \\
&= \|\Delta_t\|^2 - 2(\mathbf{w}_t - \mathbf{u})\cdot\mathbf{\Delta}_t + \|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_t - \mathbf{u}\|^2 \\
&= \|\Delta_t\|^2 - 2(\mathbf{w}_t - \mathbf{u})\cdot\mathbf{\Delta}_t \\
&= \|\Delta\|^2 - 2(\mathbf{w} - \mathbf{u})\cdot\mathbf{\Delta} \quad \text{(dropping subscript } t \text{ since it doesn't affect the proof)} \\
&= \eta^2 l^2\|\mathbf{x}\|^2 - 2\eta l\mathbf{x}\cdot(\mathbf{w} - \mathbf{u}) \\
&= \eta^2 l^2\|\mathbf{x}\|^2 - 2\eta l(\mathbf{w}\cdot\mathbf{x} - \mathbf{u}\cdot\mathbf{x} - y + y) \\
&= \eta^2 l^2\|\mathbf{x}\|^2 - 2\eta l[(\mathbf{w}\cdot\mathbf{x} - y) - (\mathbf{u}\cdot\mathbf{x} - y)] \\
&= \eta^2 l^2\|\mathbf{x}\|^2 - 2\eta l[l - g] \\
&= \eta^2 l^2\|\mathbf{x}\|^2 - 2\eta l^2 + 2\eta lg \\
&\le \eta^2 l^2 - 2\eta l^2 + 2\eta lg && (\|\mathbf{x}\|^2 \le 1) \\
&\le (\eta^2 - 2\eta)l^2 + \frac{2\eta[\frac{g^2}{1-\eta} + l^2(1-\eta)]}{2} && (ab \le \tfrac{a^2+b^2}{2}) \\
&= (\eta^2 - 2\eta)l^2 + \eta[\frac{g^2}{1-\eta} + l^2(1-\eta)] \\
&= -\eta l^2 + \frac{\eta}{1-\eta}g^2
\end{aligned}$$

2 Families of Online Algorithm

The two goals of the learning algorithm are minimizing the loss of $\mathbf{w}_{t+1}$ on $\mathbf{x}_t$ and y_t, and minimizing the distance between $\mathbf{w}_{t+1}$ and $\mathbf{w}_t$. So to generalize, we are trying to minimize

$$\eta L(\mathbf{w}_{t+1}, \mathbf{x}_t, y_t) + d(\mathbf{w}_{t+1}, \mathbf{w}_t)$$

So if we use the Euclidean norm as our distance measurement, then the above function becomes:

$$\eta L(\mathbf{w}_{t+1}, \mathbf{x}_t, y_t) + \|\mathbf{w}_t - \mathbf{w}_{t+1}\|^2$$

So if we try to optimize the above function, we have the update equation:

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta\nabla_{\mathbf{w}}L(\mathbf{w}_{t+1}, \mathbf{x}_t, y_t) \\
&\approx \mathbf{w}_t - \eta\nabla_{\mathbf{w}}L(\mathbf{w}_t, \mathbf{x}_t, y_t)
\end{aligned}$$

Notice that we use $\mathbf{w}_t$ to approximate $\mathbf{w}_{t+1}$ when we calculate $\mathbf{w}_{t+1}$. This is called the Gradient Descent Algorithm.

Alternatively, we can use relative entropy as a measure of distance. So $d(\mathbf{w}_t, \mathbf{w}_{t+1}) = RE(\mathbf{w}_t \| \mathbf{w}_{t+1})$. Now we can have the update function as

$$w_{t+1,i} = \frac{w_{t,i} \cdot \exp(\eta \frac{\partial L(\mathbf{w}_{t+1}, \mathbf{x}_t, y_t))}{\partial w_i})}{\mathcal{Z}_t}$$

This is called the Exponentiated Gradient Algorithm, or "EG" algorithm. We need to change the norm: $\|\mathbf{x}_t\|_\infty \leq 1$ and $\|\mathbf{u}\|_1 = 1$. It's also possible to prove a bound on this update equation, but we skip it in this class.

3 Online Algorithm in a Batch Setting

We can modify the online algorithms slightly so that we can use them in the batch learning settings. Let's take a look at one example in a linear regression setting. In a linear regression setting, training and test samples are drawn i.i.d from a fixed distribution $\mathcal{D}$. So we have $\mathcal{S} = \langle (\mathbf{x}_1, y_1) \ldots (\mathbf{x}_m, y_m) \rangle$ where $(x_i, y_i) \sim \mathcal{D}$. Our goal is to find $\mathbf{v}$ with low risk, where risk is defined to be

$$R_\mathbf{v} = E_{(\mathbf{x},y) \sim \mathcal{D}}[(\mathbf{v} \cdot \mathbf{x} - y)^2]$$

We want to find $\mathbf{v}$ such that $R_\mathbf{v}$ is small compared to $\min_\mathbf{u} R_\mathbf{u}$.

Now we can apply WH algorithm to the data as follows:
(1) run WH on $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, and calculate $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$.

(2) Combine the vectors:

$$\mathbf{v} = \frac{1}{m} \sum_{t=1}^{m} \mathbf{w}_t$$

and output $\mathbf{v}$. We choose to output the average of all the $\mathbf{w}_t$'s because we can prove something theoretically good about it, which is not necessarily the case for the last vector $\mathbf{w}_m$.

Now let's state another theorem:

Theorem 3.1

$$E_\mathcal{S}[R_\mathbf{v}] \leq \min_{\mathbf{u} \in \mathbb{R}^n} [\frac{R_\mathbf{u}}{1-\eta} + \frac{\|\mathbf{u}\|^2}{\eta m}]$$

If we divide T on both side of the equation above and if η is chosen to be small, we can see that $\frac{R_\mathbf{v}}{T}$ will be close to $\frac{R_\mathbf{u}}{T}$ when T is large. **Proof**:

There are three observations needed in the proof:

(1):
Let $\mathbf{x}, y$ be a random test example from $\mathcal{D}$. Then we have

$$(\mathbf{v} \cdot \mathbf{x} - y)^2 \leq \frac{1}{m} \sum_{t=1}^{m} (\mathbf{w}_t \cdot \mathbf{x}_t - y)^2$$

Proof for (1):

$$\begin{aligned}
(\mathbf{v}\cdot\mathbf{x}-y)^2 &= [(\frac{1}{m}\sum_{t=1}^{m}\mathbf{w}_t)\cdot\mathbf{x}-y]^2 \\
&= [(\frac{1}{m}\sum_{t=1}^{m}\mathbf{w}_t\cdot\mathbf{x})-y]^2 \\
&= [\frac{1}{m}\sum_{t=1}^{m}(\mathbf{w}_t\cdot\mathbf{x}-y)]^2 \\
&\le \frac{1}{m}\sum_{t}(\mathbf{w}_t\cdot\mathbf{x}-y)^2 && \text{(convexity of } f(x)=x^2\text{)}
\end{aligned}$$

(2):

$$E[(\mathbf{u}\cdot\mathbf{x}_t-y_t)^2]=E[(\mathbf{u}\cdot\mathbf{x}-y)^2]$$

The above expectation is with respect to the random choice of $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m)$ and $(\mathbf{x},y)$. This is because $(\mathbf{x}_t,y_t)$and $(\mathbf{x},y)$ are from the same distribution.

(3):

$$E[(\mathbf{w}_t\cdot\mathbf{x}_t-y_t)^2]=E[(\mathbf{w}_t\cdot\mathbf{x}-y)^2]$$

This is because $\mathbf{w}_t$ only depends on the first $t-1$ samples but doesn't depend on $(\mathbf{x}_t,y_t)$.

Now let's start the proof:

$$\begin{aligned}
E_{\mathcal{S}}[R_{\mathbf{v}}] &= E_{\mathcal{S},(\mathbf{x},y)}[(\mathbf{v}\cdot\mathbf{x}-y)^2] \\
&\le E[\frac{1}{m}\sum_t(\mathbf{w}_t\cdot\mathbf{x}-y)^2] && \text{(using observation (1))} \\
&= \frac{1}{m}\sum_t E[(\mathbf{w}_t\cdot\mathbf{x}-y)^2] \\
&= \frac{1}{m}\sum_t E[(\mathbf{w}_t\cdot\mathbf{x}_t-y_t)^2] && \text{(observation (3))} \\
&= \frac{1}{m}E[\sum_t(\mathbf{w}_t\cdot\mathbf{x}_t-y_t)^2] \\
&\le \frac{1}{m}E[\frac{\sum_t(\mathbf{u}\cdot\mathbf{x}_t-y_t)^2}{1-\eta}+\frac{\|\mathbf{u}\|^2}{\eta}] && \text{(by WH bound)} \\
&= \frac{1}{m}[\frac{\sum_t E[(\mathbf{u}\cdot\mathbf{x}_t-y_t)^2]}{1-\eta}+\frac{\|\mathbf{u}\|^2}{\eta}] \\
&= \frac{1}{m}[\frac{\sum_t E[(\mathbf{u}\cdot\mathbf{x}-y)^2]}{1-\eta}]+\frac{\|\mathbf{u}\|^2}{\eta m} && \text{(by observation (2))} \\
&= \frac{R_{\mathbf{u}}}{1-\eta}+\frac{\|\mathbf{u}\|^2}{\eta m}
\end{aligned}$$

and we have completed the proof.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #19
Scribe: Bianca M. Dumitrascu — April 15, 2014

1 Introduction to Probability Density Estimation

1.1 Summary

In this lecture we consider the framework of **Probability Density Estimation**. In our previous approaches to learning (Classification, Regression) we are given m labeled points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ according to a random distribution $\mathcal{P}$, and, disregarding this distribution, we want to predict with minimal error the label of a new point x. Such approaches belong to the class of **Discriminative Approaches**. The name comes from the fact that, from a probabilistic perspective, we attempt to find the conditional distribution $\Pr(\mathbf{y}|\mathbf{x})$ which helps us *discriminate* between different label values. An alternative is provided by **Generative Approaches** which constitutes the focus of our current discussion.

1.2 Intuition

In the generative approach setting, when presented with m labeled training samples (x_1, y_1), (x_2, y_2), …, (x_m, y_m), we assume the data points x are generated from an unknown distribution $\mathcal{P}$. We aim to model this distribution using the conditional density estimation of the quantity $\Pr(\mathbf{x}|\mathbf{y})$. To illustrate this, consider a dataset where the training samples give information about the distribution of height in a population. The training samples take the form (height, gender) where the labels y represent the gender. Given the height of a person we might want to guess the person's gender. Previously, we might have modeled this problem using a discriminative approach by looking for a threshold or for a separating hyperplane that would give us a decision rule according to which the height of interest belongs to a man or to a woman. In the new set-up, we are interested in estimating the distribution of heights separately for women and men such that we can calculate how *likely* it is for a new sample to have been generated from either of these distributions. In general, for the rest of this lecture we assume there is a probability distribution over the values x (in the example above, we assume there is a distribution of heights for women that is different from the height distribution of men), and our goal is to model this distribution for the purpose of learning and inference.

2 Maximum Likelihood

To formalize the above learning setting, consider we are given m samples $x_1, x_2, \ldots, x_m$ drawn from a probability distribution $\mathcal{P}$ over a finite domain $\mathcal{X}$ (generalizable to the continuous setting, the assumption over the domain is merely for making calculations easy). The goal is to estimate $\mathcal{P}$ by finding a model that while not too complex, is able to fit the data. To this end, let $\mathcal{Q}$ be a family of possibly infinitely many density functions q. It is among these functions that we will search for the best model to explain our data.

Definition. Let $x_1, x_2, ...x_m$ be m points sampled *iid.* from a distribution $\mathcal{P}$ and let $q \in \mathcal{Q}$ be a density function. The *likelihood* of $x_1, x_2, \ldots, x_m$ under q is the quantity:

$$\prod_{i=1}^{m} q(x_i) \tag{1}$$

Notice that this quantity is exactly the probability of generating the m independent samples given that the underlying model is given by the probability density q.

Example (Coin Toss) Consider flipping a coin with probability q of landing *heads.* Consider random variables x which take the value 1 if the coin lands heads, and 0 otherwise. Then given m coin flips, let the sequence $x_1, x_2, \ldots, x_m$ record the sequence of tosses and notice that the number of heads is nothing else but $h = \sum_{i=1}^{m} x_i$. With this notation the *likelihood* of the data under q is equal to $q^h(1-q)^{m-h}$.

Naturally, we are interested in the probability function q that performs closest to the real probability distribution $\mathcal{P}$ and which makes the likelihood quantity *more likely.* As we want to maximize the probability that the sequence $x_1, x_2, \ldots, x_m$ is generated from q we notice that:

$$\max_{q \in \mathcal{Q}} \prod_i q(x_i) = \max \log \prod_i q(x_i) = \max \sum_i \log(q(x_i)) = \min \frac{1}{m} \sum_i [-\log q(x_i)] \tag{2}$$

where $-\log q(x)$ is the log loss of q on x and where $\frac{1}{m}\sum_i[-\log q(x_i)]$ is the empirical risk, or the average log loss. We use this empirical risk as a proxy for the real expectation. The true value of the loss or true risk is given by the quantity $E_{x\sim\mathcal{P}}[-\log q(x)]$ which can be iteratively written as:

$$\begin{aligned}
&E_{x\sim\mathcal{P}}[-\log q(x)] \\
&= -\sum_{x\in\mathcal{X}} \mathcal{P}(x)\log q(x) \\
&= -\sum_{x\in\mathcal{X}} \mathcal{P}(x)\log q(x) + \sum_{x\in\mathcal{X}} \mathcal{P}(x)\log \mathcal{P}(x) - \sum_{x\in\mathcal{X}} \mathcal{P}(x)\log \mathcal{P}(x) \\
&= -\sum_{x\in\mathcal{X}} \mathcal{P}(x)\log \frac{\mathcal{P}(x)}{q(x)} - \sum_{x\in\mathcal{X}} \mathcal{P}(x)\log \mathcal{P}(x) \\
&= RE(\mathcal{P}||q) + H(\mathcal{P})
\end{aligned}$$

where RE denotes the relative entropy and H is the Shannon entropy.
Equation (3) justifies the intuition that minimizing (2) would give us the probability density function closest to the true distribution, where the chosen metric is relative entropy. Going back to the **coin toss** example, estimating the bias of the coin from the sequence of tosses is simply finding the term that maximizes the log likelihood, namely $q = \frac{h}{m}$.

3 Maximum Entropy Formulation

Consider now the related problem of modeling the distribution of interest given multiple constraints, or features of the data. As before, we are given the samples $x_1, x_2, \ldots, x_m$ generated from some unknown distribution $\mathcal{P}$ over a finite set $\mathcal{X}$ of cardinality N, with $N \gg m$. In addition, for each such x we are given a set of n functions $f_1, f_2, \ldots, f_n$ where $f_j : \mathcal{X} \to \mathbb{R}$. We call these functions features and we think of them as constraints over the distribution $\mathcal{P}$. The goal is the same as before: estimating $\mathcal{P}$ subject to the constraints induced by the features.

A natural way to approach the problem is to use the additional information encoded into the feature. To this end, notice that we can approximate the expectations $E_{\mathcal{P}}[f_j]$ over the true distribution for features f_j using the empirical average taken over the given samples:

$$E_{\mathcal{P}}[f_j] \approx \widehat{E}[f_j] = \frac{1}{m} \sum_{j=1}^{m} f_j(x_i) \tag{3}$$

The problem can be recast as the problem of finding q such that for all i from 1 to n, $E_q[f_j] = \widehat{E}[f_j]$. As there could be many probability density functions satisfying this constraint, we will pick the one that minimizes $\mathrm{RE}(q||U)$ where U is the uniform distribution over X:

$$\begin{aligned} q = argmin_q RE(q||U) &= \sum_{x \in X} q(x) \ln \frac{q(x)}{1/N} \\ &= \ln N + \sum_{x \in X} q(x) \ln(q(x)) \\ &= \ln N - H(q) \end{aligned} \tag{4}$$

With these in mind let $\mathcal{P}$ be the set of probability densities constrained by their features:

$$\mathcal{P} = \{q \mid E_q[f_j] = \widehat{E}[f_j], \forall j\}. \tag{5}$$

$$\tag{6}$$

The probability of interest is therefore the solution to the following formulation, called *maximum entropy*:

$$\begin{aligned} &\textbf{maximize } H(q) \\ &\textbf{subject to } q \in \mathcal{P} \end{aligned} \tag{7}$$

Notice however that we can also think about the problem using the maximum likelihood framework developed in the previous section. To solve this in practice we need to make the search tractable by restricting the set of density functions $\mathcal{Q}$ over which we maximize (or minimize if we talk about the log loss). One solution to this is to consider the set of probability distributions q:

$$q(x) \propto \exp\Big(\sum_{j=1}^{n} \lambda_j f_j(x)\Big) \tag{8}$$

where $\lambda_j \in \mathbb{R}$. This family of distribution functions is an example of what is often referred to as an *exponential family.* For this particular family we will use the name *Gibbs distribution.* The maximum likelihood problem becomes:

$$\begin{aligned}&\textbf{maximize } \sum_{i=1}^{m} \log q(x_i)\\ &\textbf{subject to } q \in \bar{\mathcal{Q}}\end{aligned} \tag{9}$$

where $\bar{\mathcal{Q}}$ is the closure of $\mathcal{Q}$.

The following theorem brings together the two approaches stating that they are, in fact, equivalent.

Theorem 1. Duality between Maximum Entropy and Maximum Likelihood.
Let q^* be a probability distribution. Then, the following identities are equivalent:
1. $q^* = \arg\ max_{q \in \mathcal{P}} H(q)$,
2. $q^* = \arg\ max_{q \in \bar{\mathcal{Q}}} \sum_i \log q(x_i)$,
3. $q^* \in \mathcal{P} \cap \bar{\mathcal{Q}}$

Furthermore, any of these properties uniquely determine q^*

Without proving this result, let us consider the Lagrangian form of the likelihood function, obtaining the following identity in which $q(x)$ for $x \in \mathcal{X}$ are primal variables and λ_j and γ are dual variables or Lagrange multipliers:

$$L = \sum_{x \in \mathcal{X}} q(x) \log q(x) + \sum_{j=1}^{n} \lambda_j \big(\widehat{E}[f_j] - \sum_{x \in \mathcal{X}} q(x) f_j(x)\big) + \gamma\big(\sum_{x \in \mathcal{X}} q(x) - 1\big). \tag{10}$$

$$\tag{11}$$

Setting $\frac{\partial L}{\partial q(x)}$ to zero, that is

$$0 = 1 + \log q(x) + \sum_j \lambda_j f_j(x) + \gamma, \tag{12}$$

we obtain a closed form expression for $q(x)$ that resembles the form of the optimal solution given by the Gibbs distribution, namely:

$$q(x) = \frac{\exp\big(\sum_{j=1}^{n} \lambda_j f_j(x)\big)}{e^{\gamma+1}}. \tag{13}$$

By letting $Z = e^{\gamma+1}$ and plugging it back into L, we notice that the expression simplifies to be the log likelihood of q:

$$
\begin{aligned}
L &= \sum_{x\in\mathcal{X}} q(x)\big(\sum_{j=1}^{n}\lambda_j f_j(x) - \log Z\big) - \sum_{j=1}^{n}\lambda_j \sum_{x\in\mathcal{X}} q(x) f_j(x) + \sum_{j=1}^{n}\lambda_j \widehat{E}[f_j] \\
&= -\log Z + \frac{1}{m}\sum_{j=1}^{n}\lambda_j \sum_{i=1}^{m} f_j(x_i) \\
&= \frac{1}{m}\sum_{i=1}^{m}\big(\sum_{j=1}^{n}\lambda_j f_j(x_i) - \log Z\big) \\
&= \frac{1}{m}\sum_{i=1}^{m}\log q(x_i)
\end{aligned}
$$

which suggests that the dual problem is to maximize the log likelihood of q as a function of the Lagrange multipliers λ.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture # 20
Scribe: Elena Sizikova — April 17, 2013

Last time, we began discussing how to learn a probability distribution D from samples. Suppose we are given:

1. a large but finite space X with $|X| = N$
2. D - an (unknown) distribution over X
3. $x_1, \ldots, x_m \sim D$ iid samples
4. $f_1, \ldots f_n$ features, where each feature is a function $f_j : X \to \mathbb{R}$

We would like to estimate the distribution D. Using the example of butterfly observance patterns from before, we could be estimating the distribution of butterfly positions from position features such as average temperature, average annual rainfall, altitude etc.

We have discussed two possible approaches. The first is to directly find a distribution q^* that maximizes the relative entropy:

$$P = \{q \ |\forall j: \ \mathbf{E}_q[f_j] = \hat{\mathbf{E}}[f_j]\}$$
$$q^* = \operatorname*{argmax}_{q \in P} H(q) \qquad \text{where } H \text{ is the entropy function}$$

Above, $\mathbf{E}_q[f_j]$ and $\hat{\mathbf{E}}[f_j]$ are defined as before:

$$\mathbf{E}_q[f_j] = \mathbf{E}_{x\sim q}[f_j(x)] \quad \text{and} \quad \hat{\mathbf{E}}[f_j] = \frac{1}{m}\sum_{i=1}^{m} f_j(x_i)$$

Analytically, it is usually difficult to find a maximizer for H directly. The second approach is to use a parametric form, i.e. find a parameter $\boldsymbol{\lambda} = \langle \lambda_1, \ldots, \lambda_m \rangle$ in order to minimize:

$$L(\boldsymbol{\lambda}) = \underbrace{-\frac{1}{m}\sum_{i=1}^{m} \ln(q_{\boldsymbol{\lambda}}(x_i))}_{\text{form of log loss}} \tag{1}$$

Above, $q_{\boldsymbol{\lambda}}$ is defined to be:

$$q_{\boldsymbol{\lambda}}(x) = \frac{\exp\left(\sum_j \lambda_j f_j(x)\right)}{Z_\lambda}$$

where Z_λ is a normalization constant. We can rewrite q as:

$$q_{\boldsymbol{\lambda}}(x) = \frac{\exp(g_\lambda(x))}{Z_\lambda} \quad \text{where we use} \quad g_{\boldsymbol{\lambda}}(x) := \sum_{j=1}^{n} \lambda_j f_j(x)$$

The usual approach of setting the derivative of $q_{\boldsymbol{\lambda}}$ with respect to λ_j to 0 does not yield an easily solvable system of equations, so instead we will use an iterative method of finding the minimum:

```
Choose $\boldsymbol{\lambda}_1$
For $t = 1, 2, \ldots$
    compute $\boldsymbol{\lambda}_{t+1}$ from $\boldsymbol{\lambda}_t$
```

i.e. we are trying to find a sequence of $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3, \ldots$ such that:

$$\lim_{t \to \infty} L(\boldsymbol{\lambda}_t) = \inf_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda})$$

Above, we have assumed that the features are arbitrary functions, but in practice, we make the following assumptions (without loss of generality):

$$\forall x \forall j : f_j(x) \geq 0 \tag{2}$$

$$\forall x \quad : \sum_{j=1}^{n} f_j(x) = 1 \tag{3}$$

It is straightforward to justify why (2) and (3) come w.l.o.g. Adding a constant to the feature function does not affect distribution q, so we can assume (2). We can also scale the features to have range: $f_j : X \to \left[0, \frac{1}{n}\right]$, without affecting the distribution:

$$\sum_j f_j(x) \leq 1$$

Finally, we create a dummy feature f_0 defined by:

$$f_0(x) = 1 - \sum_{j=1}^{n} f_j(x) \quad \text{which again doesn't alter } q$$

we obtain (3) for the set of all features.

Consider the difference of loss functions after each iteration of the algorithm:

$$\Delta L = L(\boldsymbol{\lambda}_{t+1}) - L(\boldsymbol{\lambda}_t)$$

We will derive a tractable approximation to ΔL and minimize it, since minimizing the loss at each step is equivalent to minimizing the overall loss. Let us focus on a particular round $\boldsymbol{\lambda} = \boldsymbol{\lambda}_t$ and $\boldsymbol{\lambda'} = \boldsymbol{\lambda}_{t+1}$. We have:

$$\begin{aligned}
\Delta L &= L(\boldsymbol{\lambda'}) - L(\boldsymbol{\lambda}) \\
&= \frac{1}{m} \sum_{i=1}^{m} \left[-\ln\left(\frac{\exp(g_{\lambda'}(x_i))}{Z_{\lambda'}}\right) + \ln\left(\frac{\exp(g_{\lambda}(x_i))}{Z_{\lambda}}\right) \right] \\
&= \frac{1}{m} \sum_i \left[g_\lambda(x_i) - g_{\lambda'}(x_i)\right] + \ln\left(\frac{Z_{\lambda'}}{Z_\lambda}\right) \qquad (4)
\end{aligned}$$

For the first term in (4), write the update to λ as $\lambda'_j = \lambda_j + \alpha_j$. Then this term becomes:

$$\begin{aligned}
\frac{1}{m}\sum_i [g_\lambda(x_i) - g_{\lambda'}(x_i)] &= \frac{1}{m}\sum_i \sum_j \left(\lambda_j f_j(x_j) - \lambda'_j f_j(x_j)\right) \\
&= \frac{-1}{m}\sum_i \sum_j \alpha_j f_j(x_i) \\
&= -\sum_j \alpha_j \underbrace{\left(\frac{1}{m}\sum_i f_j(x_i)\right)}_{\text{empirical average of } f_j} \\
&= -\sum_j \alpha_j \hat{\mathbb{E}}[f_j]
\end{aligned}$$

Now, rewriting the second term:

$$\begin{aligned}
\frac{Z_{\lambda'}}{Z_\lambda} &= \frac{\sum_{x\in X} \exp\left(\sum_j \lambda'_j f_j(x)\right)}{Z_\lambda} \\
&= \sum_{x\in X} \underbrace{\frac{\exp\left(\sum_j \lambda_j f_j(x)\right)}{Z_\lambda}}_{q_\lambda(x)} \cdot \exp\left(\sum_j \alpha_j f_j(x)\right) \\
&= \sum_x q_\lambda(x) \exp\left(\sum_j \alpha_j f_j(x)\right)
\end{aligned}$$

Note that for each x, the feature values $f_1(x), \ldots, f_n(x)$ form a distribution by our assumptions (2) and (3). Also $\sum_j \alpha_j f_j(x)$ is a weighted average of the α_js. Using convexity of the exponential function, we have:

$$\begin{aligned}
\frac{Z_{\lambda'}}{Z_\lambda} &\le \sum_x q_\lambda(x) \sum_j f_j(x) e^{\alpha_j} \\
&= \sum_j e^{\alpha_j} \underbrace{\sum_x q_\lambda(x) f_j(x)}_{\mathbb{E}_{q_\lambda}[f_j]}
\end{aligned}$$

Finally, going back to (4), we have:

$$\begin{aligned}
\Delta L &= \frac{1}{m}\sum_i [g_\lambda(x_i) - g_{\lambda'}(x_i)] + \ln\left(\frac{Z_{\lambda'}}{Z_\lambda}\right) \\
&\le -\sum_j \alpha_j \hat{\mathbf{E}}[f_j] + \ln\left(\sum e^{\alpha_j} \mathbf{E}_{q_\lambda}(f_j)\right) \\
&= -\sum_j \alpha_j \hat{\mathbf{E}}_j + \ln\left(\sum_j e^{\alpha_j} \mathbf{E}_j\right) \qquad (5)
\end{aligned}$$

where we define $\hat{\mathbf{E}}_j := \hat{\mathbf{E}}[f_j]$ and $\mathbf{E}_j := \mathbf{E}_{q_\lambda}(f_j)$. Notice that we can now optimize the RHS of (5) directly, by taking partial derivatives:

$$0 = \frac{\partial}{\partial \alpha_j} = -\hat{\mathbf{E}}_j + \frac{\mathbf{E}_j e^{\alpha_j}}{\sum_j \mathbf{E}_j e^{\alpha_j}}$$

Notice that if α_j is a solution to the above, then so is $\alpha_j + c$ for a constant c, and so we choose c so that the denominator $\sum_j \mathbf{E}_j e^{\alpha_j}$ is equal to zero. We thus find a solution where $\alpha_j = \ln\left(\hat{\mathbf{E}}_j/\mathbf{E}_j\right)$. It follows that the algorithm's iterative update on round t is:

$$\lambda_{t+1,j} = \lambda_{t,j} + \ln\left(\frac{\hat{\mathbf{E}}[f_j]}{\mathbf{E}_{q_{\boldsymbol{\lambda}_t}}(f_j)}\right)$$

we hope that this process converges to the optimal value of $\boldsymbol{\lambda}$.

Thus, it remains to prove convergence. Define $p_t = q_{\lambda_t}$.
Definition. A function A : { probability distributions over X} $\rightarrow \mathbb{R}$ is an *auxiliary* function if it satisfies the following requirements:

1. A is continuous
2. $L(\boldsymbol{\lambda_{t+1}}) - L(\boldsymbol{\lambda_t}) \leq A(p_t) \leq 0$. (We want ≤ 0 so that the loss is always decreasing.)
3. If for some distribution p, $A(p) = 0$, then $\mathbf{E}_p[f_j] = \hat{\mathbf{E}}[f_j]$ for all j. In other words, $p \in P$.

Theorem. $p_t \rightarrow q^*$.
We first prove that if an auxiliary function A exists, then the theorem statement holds.

Suppose A is an auxiliary function. We know that $L \geq 0$ by properties of $\ln(x)$ and definition of L. By the second property of auxiliary functions, the loss L is decreasing and bounded below by 0, so $L(\boldsymbol{\lambda_{t+1}}) - L(\boldsymbol{\lambda_t}) \rightarrow 0$, and thus $A(p_t) \rightarrow 0$ as $t \rightarrow \infty$.

Now, we consider what happens at the limit of t. Suppose $p = \lim_{t\rightarrow\infty} p_t$. Since for all t, $p_t \in \overline{Q}$, where $\overline{Q}$ is the closure of Q, we have that $p \in \overline{Q}$. Also, since A is continuous,

$$A(p) = A(\lim_{t\rightarrow\infty} p_t) = \lim_{t\rightarrow\infty} A(p_t) = 0$$

Thus, $p \in P$. But now we have proved that $p \in P$ and $p \in \overline{Q}$, so $p \in P \cap \overline{Q}$. As we have stated (without proof) a theorem that $P \cap \overline{Q} = \{q^*\}$, it follows that $p = q^*$.
(This assumes that the limit $\lim_{t\rightarrow\infty} p_t$ exists. If it does not exist, applying general results from real analysis (which are slightly beyond the scope of this class), we know that $\{p_t | t = 0, 1, \ldots\}$ belong to a compact subspace of $\mathbb{R}^n$, and so there is a convergent subsequence of p_t's. By the same proof just given, this subsequence must converge to q^*. Thus, the only limit point of this subsequence is q^*. Therefore, by general results from real analysis, the entire sequence p_t converges to q^*.)

We now have:

$$\begin{aligned}\Delta L &\leq -\sum_j \alpha_j \hat{\mathbf{E}}_j + \ln\left(\sum_j e^{\alpha_j}\mathbf{E}_j\right) \\ &= -\sum_j \hat{\mathbf{E}}_j \ln\frac{\hat{\mathbf{E}}_j}{\mathbf{E}_j} + \ln\left(\sum_j \hat{\mathbf{E}}_j\right) \quad \text{using } \alpha_j = \ln\left(\hat{\mathbf{E}}_j/\mathbf{E}_j\right) \qquad (6)\end{aligned}$$

Now,for any distribution q,

$$\sum_j \mathbf{E}_q[f_j] = \mathbf{E}_q\left[\sum_j f_j(x)\right] = \mathbf{E}_q[1] = 1$$

and therefore $\mathbf{E}_q[f_1], \ldots, \mathbf{E}_q[f_n]$ forms a distribution. In particular, this means that $\hat{\mathbf{E}}_j$ and $\mathbf{E}_j$ form distributions.

So in (6), we find that the second term simplifies to:

$$\ln\left(\sum_j \hat{\mathbf{E}}_j\right) = \ln 1 = 0$$

Hence we can rewrite (6) in terms of relative entropy:

$$\Delta L \leq -RE\left(\left\langle \hat{\mathbf{E}}[f_1], \ldots, \hat{\mathbf{E}}[f_n]\right\rangle \,||\, \langle \mathbf{E}_{p_t}(f_1), \ldots, \mathbf{E}_{p_t}(f_n)\rangle\right)$$

Now, define:

$$A(p) := -RE\left(\left\langle \hat{\mathbf{E}}[f_j]\right\rangle \,||\, \langle \mathbf{E}_p(f_j)\rangle\right)$$

where $\left\langle \hat{\mathbf{E}}[f_j]\right\rangle := \left\langle \hat{\mathbf{E}}[f_1], \ldots, \hat{\mathbf{E}}[f_n]\right\rangle$ and $\langle \mathbf{E}_p(f_j)\rangle := \langle \mathbf{E}_p(f_1), \ldots, \mathbf{E}_p(f_n)\rangle$.

It remains to verify that A is an auxiliary function. Clearly A satisfies properties 1 and 2 (continuity and non-positivity) of auxiliary functions, by properties of relative entropy. Now, relative entropy is zero iff two distributions are indentical, so $A(p) = 0$ implies $\hat{\mathbf{E}}[f_j] = \mathbf{E}_p(f_j)$ for all j, i.e. $p \in P$. $\square$

Observe that we have not addressed over how quickly does the given algorithm converge, but this is out of scope of the lecture.

Next: The above algorithm applies to the batch setting. The following is an outline of an analogous algorithm for the online setting, that we will explore next time:

```
For round $t = 1, \ldots, T$:
    Each expert $i$ chooses distribution $p_{t,i}$ over X
    Master combines the distribution into its own distribution $q_t$
    Observe $x_t \in X$
    Evaluate loss $= -\ln q_t(x_t)$
```

We want:

$$\sum_{t=1}^{T} -\ln q_t(x_t) \leq \min_i \underbrace{\sum_{t=1}^{T} -\ln p_{t,i}(x_t)}_{\text{loss of expert } i} + \text{small regret}$$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire — Lecture #21
Scribe: Amy Tai — April 22, 2014

1 Online Learning with Log-Loss

1.1 Problem

Recall the online learning problem from last time:

```
$i \in N$ experts
for $t = 1, \dots, T$ do
    each expert $i$ chooses distribution $p_{t,i}$ over $X$
    master combines into distribution $q_t$ over $X$

    observe $x_t \in X$
end for
```

At the end of this learning procedure, we want the following bound on the log-loss of the master:

$$-\sum_{t=1}^{T} \ln q_t(x_t) \le \min_i \left(-\sum_{t=1}^{T} \ln p_{t,i}(x_t) \right) + \epsilon \tag{1}$$

where ϵ is small, and the minimization term is interpreted as the cumulative loss of the best expert, in hindsight.

1.2 Universal Compression

Before we develop an algorithm that allows us to arrive at the desired bound in Equation 1, we take an aside to motivate online learning with log-loss.

Here is a problem from coding theory: Suppose we have a sender, Alice, and a receiver, Bob. Suppose Alice wants to send a message to Bob, and suppose the message is comprised of letters from alphabet X and $p(x)$ is the probability of choosing $x \in X$. Then for messages of length 1 letter, the optimal way of coding the message is $-\lg p(x)$ bits.

But the more interesting problem is extending this encoding to messages of arbitrary length, $x_1, x_2, \dots$. We could just assume that the x_t are independently drawn from the same distribution p, in which case the optimal encoding simply has length $\sum_t -\lg p(x_t)$.

However, independence is a poor assumption. In English, for example, knowing that the message so far is "`I am goi`", basically tells us that the next letter is going to be `n`. However, using the typical distribution of letters over the English alphabet, the strawman approach would guess that `e` has the highest likelihood of being the next letter.

From this example, we learn that ideally, we want p_t to be the probability of the next letter, x_t, *given* the context $\equiv \langle x_1, \dots, x_{t-1} \rangle$. Denote this context as x_1^{t-1}. If we have such p_t, then the length of the optimal encoding will be $\sum_t -\lg p_t(x_t)$ bits.

However, it is nearly impossible to learn p_t, because of the nearly limitless number of contexts. Instead, we consider a learning algorithm that takes a bunch of candidate methods for encoding the messages, and combine them into a master method. For example, a sample candidate method would only look 1 letter back, and have contexts of length 1 letter.

Formally, suppose the ith method does the following: Given x_1^{t-1}, encodes x_t using $-\lg p_{t,i}(x_t)$ bits, where $p_{t,i}$ is method i's estimated distribution of x_t, given x_1^{t-1}.

Then we combine these into the master method, which encodes x_t with $-\lg q_t(x_t)$ bits. Then the master uses a total of $\sum_{t=1}^{T} -\lg q_t(x_t)$ bits to encode the entire message. We want:

$$\sum_{t=1}^{T} -\lg q_t(x_t) \leq \min_i \sum_{t=1}^{T} -\lg p_{t,i}(x_t) + \epsilon$$

Namely, we want the total number of bits used by the master to be at most the number of bits used by the best method, plus some small ϵ. This is exactly what the online learning model that we presented in Section 1.1 does.

2 Bayes Algorithm

Now we go back to describing an algorithm for arriving at Equation 1. To derive such an algorithm, we *pretend* that the data is random. However, we will see in the next section that the proof of the algorithm's bounds holds for any arbitrary data sequence and does not depend on the randomness assumption. Then suppose $x_1, \ldots, x_T$ is random and generated as follows:

- Expert i^* chosen at random (assume uniformly across all experts). So $Pr[i^* = i] = \frac{1}{N}$
- Generate x_t according to distribution p_{t,i^*}. Then $Pr[x_t | x_1^{t-1}, i^* = i] = p_i(x_t | x_1^{t-1})$

We also denote $p_{t,i}(x_t) = p_i(x_t | x_1^{t-1})$ and $q_t(x_t) = q(x_t | x_1^{t-1})$. Furthermore, by definition, $q(x_t | x_1^{t-1}) = Pr[x_t | x_1^{t-1}]$. Then we have:

$$\begin{aligned}
q(x_t | x_1^{t-1}) &= Pr[x_t | x_1^{t-1}] && \text{By definition} \\
&= \sum_i Pr[x_t, i^* = i | x_1^{t-1}] && \text{Marginalizing over all experts} \\
&= \sum_i Pr[i^* = i | x_1^{t-1}] \cdot Pr[x_t | i^* = i, x_1^{t-1}] && \text{Product rule} \\
&= \sum_i w_{t,i} \cdot p_i(x_t | x_1^{t-1})
\end{aligned}$$

where we denote $w_{t,i} = Pr[i^* = i | x_1^{t-1}]$. Then we seek to specify how $w_{t,i}$ is updated from round to round. First, we observe that $w_{1,i} = Pr[i^* = i | \emptyset] = Pr[i^* = i] = \frac{1}{N}$.

Now suppose that we have $w_{t,i}$ and want $w_{t+1,i}$. Then:

$$\begin{aligned}
w_{t+1,i} &= Pr[i^* = i | x_1^t] \\
&= Pr[i^* = i | x_t, x_1^{t-1}] \\
&= \frac{Pr[i^* = i | x_1^{t-1}] \cdot Pr[x_t | i^* = i, x_1^{t-1}]}{Pr[x_t | x_1^{t-1}]} && \text{Bayes' Rule} \\
&= \frac{w_{t,i} \cdot p_i(x_t | x_1^{t-1})}{\text{norm}} && \text{Consider } Pr[x_t | x_1^{t-1}] \text{ as a normalization}
\end{aligned}$$

Because $Pr[x_t|x_1^{t-1}]$ is independent of i, we can consider it as a normalization factor. Then we observe that we have a simple update algorithm for $w_{t,i}$. We then modify and fill in the learning problem from Section 1.1 with the following algorithm:

```
**Bayes' Algorithm**
$i \in N$ experts
**Initialize** $w_{1,i} = \frac{1}{N}$
**for** $t = 1, \dots, T$ **do**
    each expert $i$ chooses distribution $p_{t,i}$ over $X$
    $q_t(x) = \sum_i w_{t,i} p_{t,i}(x)$
    **Update** $w_{t+1,i} = \frac{w_{t,i} \cdot p_{t,i}(x_t)}{\text{norm}}$
    observe $x_t \in X$
**end for**
```

We also observe that this algorithm is very similar to the weight-update online learning algorithms that we saw earlier, in which $w_{t+1,i} = \frac{w_{t,i} \cdot \beta^{\text{loss}}}{Z_t}$. In this case, loss is the log-loss, or $-\ln p_{t,i}(x_t)$. If we let $\beta = e^{-1}$, then $\beta^{\text{log-loss}}$ is precisely the update factor in Bayes' algorithm.

Now we prove that this algorithm works on any given data $x_1, \dots, x_T$, and achieves the log-loss bound presented in Equation 1.

3 Bounding Results of Bayes' Algorithm

First, we extend the definition of q so it is defined for entire sequences $x_1, \dots, x_T$.

Define:

$$\begin{aligned} q(x_1^T) &= q(x_1) \cdot q(x_2|x_1) \cdot q(x_3|x_2, x_1) \dots \\ &= \prod_t q(x_t|x_1^{t-1}) \\ &= \prod_t Pr[x_t|x_1^{t-1}] \end{aligned}$$

Similarly, define:

$$\begin{aligned} p_i(x_1^T) &= \prod_t p_i(x_t|x_1^{t-1}) \\ &= \prod_t Pr[x_1^T|i^* = i] \end{aligned}$$

Then:

$$\begin{aligned} -\sum_t \ln q_t(x_t) &= -\sum_t \ln q(x_t|x_1^{t-1}) \\ &= -\ln \prod_t q(x_t|x_1^{t-1}) \\ &= -\ln q(x_1^T) \end{aligned}$$

Similarly, $-\sum_t \ln p_{t,i}(x_t) = -\ln p_i(x_1^T)$. Then:

$$\begin{aligned} q(x_1^T) = Pr[x_1^T] &= \sum_i Pr[i^* = i] \cdot Pr[x_1^T | x^* = i] \\ &= \frac{1}{N} \sum_i p_i(x_1^T) \\ \Rightarrow \text{Log-loss of } q = -\ln q(x_1^T) &= -\ln(\frac{1}{N} \sum_i p_i(x_1^T)) \\ &\leq -\ln \frac{1}{N} p_i(x_1^T) \qquad\qquad \sum_i p_i(x_1^T) \geq p_i(x_1^T) \end{aligned}$$

Because this last inequality is true for all i, it must be true that the Log-loss of $q \leq \min_i(-\ln p_i(x_1^T)) + \ln N = \min_i(-\sum_t \ln p_{t,i}(x_t)) + \ln N$, which is exactly as we desired in Equation 1.

We note here that Alice could have used an offline algorithm to determine the best encoding of a message. Namely, Alice runs all i methods on her message, determines the encoded message with optimal length, which would be $\min_i(-\sum_t \lg p_{t,i}(x_t))$ bits, and sends this across to Bob, along with a specification of the method she used for the encoding. The encoding of the message takes $\lg N$ bits, if there are N methods. Hence, using an offline method, Alice achieves the same bounds on the length of her optimal message. However, using the online method, Alice can encode the message as it comes in a stream, and does not have to store as much data for her calculations.

We also note that we could have used a different prior probability other than the uniform distribution. We could use $Pr[i^* = i] = \pi_i$, instead of $\frac{1}{N}$. Then the only modification we need to make to the original Bayes' algorithm is the initialization of the weights, which we change to $w_{1,i} = \pi_i$. Then the bound becomes

$$-\sum_{t=1}^{T} \ln q_t(x_t) \leq \min_i \left(-\sum_{t=1}^{T} \ln p_{t,i}(x_t) - \ln \pi_i \right)$$

4 An Example

Suppose $X = \{0, 1\}$. Expert p predicts 0 with probability $1 - p$ and 1 with probability p. We have an expert for all $p \in [0, 1]$, hence we have infinite experts.

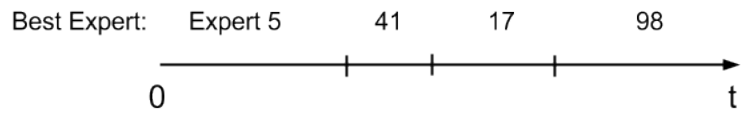
Hence $w_{t,p} = \frac{1}{\text{norm}} \prod_{t=1}^{t-1} \begin{cases} p & \text{if } x_t = 1 \\ 1-p & \text{otherwise} \end{cases}$. Suppose there are h 1's so far. Then $w_{t,p} = \frac{p^h(1-p)^{t-1-h}}{\text{norm}}$.

Then $q_t = \sum_p w_{t,p} \cdot p = \int_0^1 w_{t,p} \cdot p dp = \frac{h+1}{(t-1)+2}$. This is known as Laplace smoothing.

But in this case, $N = \infty$, so the $\ln N$ from Bayes' algorithm is not helpful. In a later lecture, we will get general bounds that can be applied to this particular case.

5 Switching Experts

Next time, we will cover a new learning model. Hitherto, we have assumed that there is one single, best expert. But this is not always a reasonable assumption. Instead, this new model assumes that the best expert switches based on data that the model sees, and this may change as time goes on. For example in the following diagram, the best expert changes during each epoch:



We will also assume that the number of switches is bounded.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Nicolas Crowell

Lecture 22
April 24, 2014

1 Bayes' algorithm review

Last lecture, we derived an on-line algorithm for estimating a probability distribution for a sequence of elements. We developed the algorithm as a general framework that can incorporate and choose among distinct individual methods for estimating distributions, so as to take into account the diversity of problem-specific methods that exist for estimating probability distributions in particular settings.

To repeat the formal problem setting: we are given a set of methods for estimating a probability distribution over a space. We often call these methods "experts". We observe a sequence of elements from the space (which may be chosen adversarially), and estimate on-line a probability distribution based on everything we've seen so far. Our goal is to develop an on-line algorithm that estimates a distribution by incorporating the outputs of the predetermined set of estimation methods. We will want the algorithm to perform almost as well as the best *single* expert on the observed sequence of elements, with a corresponding theoretical bound proving this "almost as good" comparison.

In this setting, we will measure performance by minimizing *log loss*. Given a probability distribution and an observed element, we define loss to be the negative of the logarithm of the probability of the observed element under the probability distribution, i.e. $-\log p(x)$.

With these goals in mind, we derived the Bayes algorithm, together with the following performance comparison bound bound:

$$-\sum_{t=1}^{T} \ln q_t(x_t) \leq \left(-\sum_{t=1}^{T} \ln p_{i,t}(x_t) \right) + \ln N, \ \forall \text{ experts } i \tag{1}$$

where q_t is the probability distribution chosen by Bayes' algorithm at time t, $p_{i,t}$ is the probability distribution chosen by expert i at time t, and N is the number of experts. We also showed that the $\ln N$ term corresponded to the special case of a slightly more general derivation, where instead of assuming a uniform distribution over the experts at the beginning of the derivation, we use a different distribution that incorporates prior knowledge about the different choices of expert, where π_i is the new prior probability of expert i. The more general bound was then the following:

$$-\sum_{t=1}^{T} \ln q_t(x_t) \leq \left(-\sum_{t=1}^{T} \ln p_{i,t}(x_t) \right) - \ln \pi_i, \ \forall \text{ experts } i \tag{2}$$

Switching experts

A more general problem

Now we consider a slightly more general problem: what if different methods of estimating the distribution perform better during different parts of the sequence? For example, for

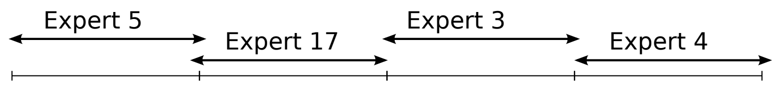


Figure 1: An observation sequence generated by 4 experts respectively choosing probability distributions at different parts of the sequence.

the first 83 time steps of the on-line sequence, expert 1 might have the best guesses, while expert 2 might be more accurate during the next 47 observations.

In the derivation of the bound for Bayes' algorithm, we pretended that the data was generated randomly according to some expert's distribution at every time step, and the expert whose distribution was followed was chosen randomly at the *beginning* of the process. Now we change the model by further pretending that at arbitrary times t, the chosen expert is *switched* to a different expert. In this new setting, algorithms that do comparably well to the best *single* expert are no longer satisfying: we want to do almost as well as the best *switching sequence* of experts.

Our original motivating example from coding theory hypothesized that someone wanted to send a message using as few bits as possible, so they needed to find an encoding that would take advantage of the implicit probability distribution of the possible elements of the message to minimize the number of bits used to encode the message. In this new setting, we can imagine that the content of the message takes several different forms. For example, the first part of the message might be written in English, and the second part of the message might be written in Spanish, which presumably have different distributions over characters. Generalizing the example even further, the rest of the message might switch between sending pictures, and then maybe music, or even completely random bits. The point is that these different kinds of message content will likely have very different distributions of elements, so we will want to switch between different methods of encoding in order to take advantage of each distribution separately.

A strategy for solving the switching experts problem

So we want to do almost as well as the best switching sequence of experts, and we already have a way to do almost as well as a single best expert. Our idea for an algorithm to address this new problem setting will be to instead consider "meta-experts" representing a switching sequence of *base* experts. A meta-expert makes predictions based on which base expert it considers to be the "true" expert at a given time. Specifically, we can have a meta-expert for *every* switching sequence with k switches. Then we could just apply Bayes' algorithm to this huge family of meta-experts built from the N base experts. We can count these meta experts:

$$M \triangleq \# \text{ of meta experts} = N^{k+1}\binom{T+k}{k}$$

And if we want to plug this size into our error term in the original bound for Bayes' algorithm with a uniform prior, we just need to compute $\ln M$:

$$\ln M \approx \ln N + k\ln N + k\ln(T/k) \approx \ln N + [k(\ln N + \ln T)] \tag{3}$$

The inner term $(\ln N + \ln T)$, which is multiplied by the number of switches k, conceptually corresponds to paying a price in loss for every time a meta-expert switches base experts, in addition to the original cost of Bayes' algorithm.

Making meta-experts tractable

The problem with this approach is that the number of meta-experts is exponential in N. We need a computational trick that will allow us to consider the very large space of meta-experts without computationally interacting with each one directly.

Another inelegant part of our meta-expert idea as stated is that it uses an external parameter, namely the precise number of times that the base expert switches. To deal with both this rather arbitrary parameter and the computational problem posed by the large number of meta-experts, we will take advantage of the flexibility of Bayes' algorithm, which works for *any* prior distribution over the space of experts, along with a clever choice of prior distribution.

Let's think of a meta-expert as a vector $\mathbf{e}$ of T elements representing which base expert is predicting at time t.

$$e_t \in \{1, \ldots, N\} \quad \mathbf{e} \in \mathbb{M} \triangleq \{1, \ldots, N\}^T$$

Every meta-expert corresponds to a vector in the space $\mathbb{M}$. Notice that we've represented *every* possible meta expert, including one that switches at every time step! This seems undesirable, as such a meta-expert represents a very complicated way of generating data. Reflecting our general preference in machine learning for simpler hypotheses, we will want to focus on "nice" blocky sequences, where the meta-expert doesn't switch very often. To embody this preference, we can weight the prior distribution over meta-experts to give higher weight to meta-experts with fewer switches.

To find such a prior, we will come up with a random process to generate meta-experts in such a way as to embody the simplicity preference. The random process will then exactly define our prior distribution over the meta-experts, and we can write:

$$\Pr[\mathbf{e}^* = \mathbf{e}] = \pi(\mathbf{e})$$

where $\pi(\mathbf{e})$ is the probability of a meta-expert under the random process, and $\mathbf{e}^*$ is the chosen "true" meta-expert.

A prior distribution over meta-experts

The basic heuristic we want to follow is that a higher number of switches should correspond to a lower probability. Our random process will work by generating one component of a vector representing a meta-expert at a time. Let the initial component be chosen uniformly at random.

$$e_1^* = \text{uniform} \implies \Pr[e_1^* = i] = 1/N$$

And the key step is how to find component $t+1$ in accordance with our heuristic, which is that we want consecutive components to be the same with relatively high probability:

$$e_{t+1}^* = \begin{cases} e_t^* \text{ with probability } 1-\alpha \\ \text{some other expert chosen uniformly, with probability } \alpha \end{cases}$$

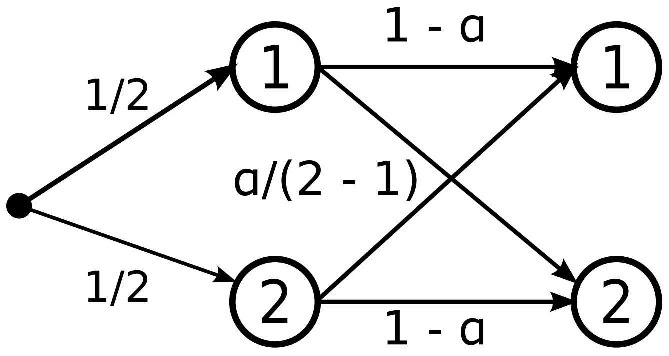


Figure 2: We randomly generate the generating meta-expert via a Markov chain of base experts.

Given this, we can compute the corresponding conditional probability that some base expert is the true expert at time $t+1$ given the identity of the true expert at time t:

$$\Pr[e^*_{t+1} \mid e^*_t] = \begin{cases} 1-\alpha & \text{if } e^*_{t+1} = e^*_t \\ \alpha/(N-1) & \text{otherwise} \end{cases}$$

This means that we defined a Markov chain of base experts, which we illustrate in Figure (2).

So meta-experts are defined by a path in the Markov sequence graph, and their probability is just the product of the edge probabilities. This defines our prior distribution on meta-experts, so we can now analyze our original bound.

Analyzing loss of meta-expert algorithm

Say that some meta-expert $\mathbf{e}$ has k switches. Then we can compute the prior probability that it is the generating expert, and thus compute the error term in our original bound on the error of Bayes' algorithm compared to any expert, including the best one:

$$\begin{aligned} -\ln(\pi(\mathbf{e})) &= -\ln\left[(1/N)\left(\frac{\alpha}{N-1}\right)^k (1-\alpha)^{T-1-k}\right] \\ &= \ln N + k\ln\frac{N-1}{k} + (T-k-1)\ln(1/(1-\alpha)) \end{aligned}$$

If we "approximately minimize" this with respect to α, we find $\alpha = k/(T-1)$. Then we have:

$$-\ln(\pi(\mathbf{e})) \approx \ln N + k(\ln N + \ln T)$$

which is (roughly) the same bound as Equation (3), which was for the space of all meta-experts with exactly k experts. It's a nice result that in moving from considering only experts with precisely k switches between base experts to those with any number of switches, we can still bound the loss in terms of the number of switches made by the generating expert!

Efficient implementation of meta-expert algorithm

The number of meta-experts with *any* number of switches is much larger than just those with k switches for some k, so we need to find a way of dealing with all of them at once without performing computation directly proportional to the population size.

Recall that at every time step of Bayes' algorithm, the "master" receives probability distributions from each expert and combines them into a single distribution before observing the next data point. The combined distribution is produced by weighting each expert's distribution and normalizing. The weights are updated every round by multiplying each expert's old weight by the probability of the observed data point under its chosen distribution for that round.

Before diving into how Bayes' algorithm will work with meta-experts, we define some notation and make explicit what is signified in the terms of the problem by a few specific random events.

$x_1^t \triangleq \langle x_1, \ldots, x_t \rangle$	the sequence of observations up to time t	(4)
$\Pr[\mathbf{e}^* = \mathbf{e}] = \pi(\mathbf{e})$	the prior probabilty that $\mathbf{e}$ is the generating meta-expert	(5)
$\Pr[x_t \vert x_1^{t-1}, \mathbf{e}^* = \mathbf{e}]$	the prediction of $\mathbf{e}$ on x_t, i.e. the prediction of base expert $\mathbf{e}_t$	(6)

$$\Pr[x_t|x_1^{t-1}, e_t^* = i] \triangleq p_i(x_t|x_1^{t-1}) \tag{7}$$

(prediction of base expert selected by generating meta-expert at time t)

Also recall that when we derived/analyzed Bayes' algorithm, we pretended that one of the experts is chosen at the beginning of the observation sequence by a random process, and that at every time step the observed data point is generated according to the generating expert, which was randomly chosen to begin with. We also define the probability distribution produced by the master to be the probability of the next data point under this pretend random process given all of the observed data so far, which is implicitly marginalized with respect to the experts, one of which was randomly chosen to be the "generating" expert.

$$\begin{aligned}
q(x_t \mid x_1^{t-1}) &= \Pr[x_t \mid x_1^{t-1}] \\
&= \sum_{i=1}^{N} \Pr[x_t, e_t^* = i \mid x_1^{t-1}] && \text{(marginalization)} \\
&= \underbrace{\Pr[e_t^* = i \mid x_1^{t-1}]}_{v_{t,i}} \cdot \underbrace{\Pr[x_t \mid e_t^* = i, x_1^{t-1}]}_{p_i(x_t \mid x_1^{t-1})} && \text{(defn. of conditional probability)}
\end{aligned}$$

We've separated the two quantities above to deal with them separately below. The probability $v_{t,i}$ is the probability of the event that the t-th generating expert is base expert i, given all of the observations through x_{t-1}, which can also be described as the distribution of base experts at time t given the previous observations. The probability $p_i(x_t \mid x_1^{t-1})$ is the distribution of observations at time t, given all of the previous observations *and* the identity of the generating base expert at time t.

First, we examine $v_{t,i}$. At the first time step, we have $v_{1,i} = \Pr[e_1^* = i] = 1/N$. For time step $t + 1$, we will first explicitly marginalize with respect to the experts at the previous time step.

$$\begin{aligned}\forall i,\ v_{t+1,i} = \Pr[e^*_{t+1} = i \mid x_1^t] &= \sum_{j=1}^{N} \Pr[e^*_{t+1} = i, e^*_t = j \mid x_1^t] \\ &= \sum_{j=1}^{N} \underset{(B)}{\Pr[e^*_{t+1} = i \mid e^*_t = j,\ x_1^t]} \cdot \underset{(A)}{\Pr[e^*_t = j \mid x_1^t]} \qquad (8)\end{aligned}$$

Note that (B) is defined directly by the Markovian process we defined early:

$$\Pr[e^*_{t+1} = i \mid e^*_t = j, x_1^t] = \Pr[e^*_{t+1} = i \mid e^*_t = j] = \begin{cases} 1-\alpha \text{ when } i = j \\ \alpha/(N-1) \text{ otherwise}\end{cases}$$

Diving into (A):

$$\begin{aligned}\Pr[e^*_t = j \mid x_t, x_1^{t-1}] &= \frac{\Pr[x_t \mid e^*_t = j, x_1^{t-1}] \cdot \Pr[e^*_t = j, x_1^{t-1}]}{\Pr[x_t \mid x_1^{t-1}]} \qquad \text{(Bayes' rule)} \\ &= \frac{v_{t,j} \cdot p_j(x_t \mid x_1^{t-1})}{q(x_t \mid x_1^{t-1})} \qquad (9)\end{aligned}$$

Given these simplifications for the quantities (A) and (B), we can write a computable expression for $v_{t+1,i}$

$$v_{t+1,i} = \sum_j \frac{v_{t,j} \cdot p_j(x_t \mid x_1^{t-1})}{q_t(x_t \mid x_1^{t-1})} \cdot \begin{cases} 1-\alpha \text{ when } i = j \\ \alpha/(N-1) \text{ otherwise}\end{cases} \qquad (10)$$

So we see that $v_{t+1,i}$ can be computed using the sum of N terms from Equation (10). This means we can compute $v_{t+1,i}$ for all $i \in \{1, \ldots, N\}$ in $\mathcal{O}(N^2)$ time. In fact, we can algebraically simplify to compute this in $\mathcal{O}(N)$ time without much more work:

$$\begin{aligned}v_{t,j} = \Pr[e^*_t = j \mid x_1^{t-1}] &= \sum_j \frac{v_{t,j} p_j(x_t \mid x_1^{t-1})}{q_t(x_t \mid x_1^{t-1})} \cdot \begin{cases} 1-\alpha \text{ when } i = j \\ \alpha/(N-1) \text{ otherwise}\end{cases} \\ &= \sum_j c_j \left[\frac{\alpha}{N-1} + \left(1 - \alpha - \frac{\alpha}{N-1}\right) \cdot \mathbb{1}\{i = j\}\right] \\ &= \left(1 - \alpha - \frac{\alpha}{N-1}\right) c_i + \frac{\alpha}{N-1} \sum_{j=1}^{N} c_j \\ &= \frac{\alpha}{N-1} + \left(1 - \alpha - \frac{\alpha}{N-1}\right) c_i \qquad (11)\end{aligned}$$

where c_i is the quantity (A), from equations (8) and (9), and is the posterior probability of the generating expert being base expert i at time t. This means we can compute the weight update for one of the experts at one time step in constant time, so each time step will only need $\mathcal{O}(N)$ computation to update a set weights over the base experts, even though there are a huge number of meta-experts.

2 Stock investment, game theory

How do you choose to apportion money between different investment instruments? We will set up some notation and a simple mathematical framework to be expanded upon during the next lecture.

There are N stocks. Every "day" (pick your favorite unit of time), you need to decide how to reapportion wealth among the stocks during the morning. At the end of each day t, we find out the price shift for each stock i:

$$p_t(i) = \frac{\text{price at end of day } t}{\text{price at beginning of day } t}$$

For example, if stock i goes up 5% on day t, then $p_t(i) = 1.05$.

Let S_t be our total wealth at the start of day t, and let $S_1 = 1$. Let $w_t(i) \triangleq$ the fraction of our wealth which we choose to invest in stock i on day t. Then our notation allows us to write the amount of our wealth at the beginning of day t invested in a particular stock i as $S_t \cdot w_t(i)$, and the corresponding amount for the end of day t is $S_t \cdot w_t(i) \cdot p_t(i)$.

Our total wealth at the end of day t is the following:

$$S_{t+1} = \sum_{i=1}^{N} S_t w_t(i) p_t(i) = S_t \cdot (\mathbf{w}_t \cdot \mathbf{p}_t)$$

And at the end of T time steps:

$$S_T = S_1 \cdot \prod_{t=1}^{T} (\mathbf{w}_t \cdot \mathbf{p}_t) = \prod_{t=1}^{T} (\mathbf{w}_t \cdot \mathbf{p}_t)$$

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Kevin Lee

Lecture #23
April 29, 2014

1 Recap

Last lecture, we briefly introduced the setup for portfolio selection. We assume that every time period, there are N stocks available to us, and we want to figure out how to best allocate our money among the stocks at the beginning of every investment period. We define:

$$p_t(i) = \frac{\text{price of stock } i \text{ at end of day } t}{\text{price of stock } i \text{ at start of day } t}$$

as the price relative which is how much a stock goes up or down in a single day.

S_t denotes the amount of wealth we have at the start of day t and we assume $S_1 = 1$. We denote $w_t(i)$ to be the fraction of our wealth that we have in stock i at the beginning of day t which can be viewed as a probability distribution as $\forall i, w_t(i) \geq 0$ and $\sum_i w_t(i) = 1$. We can then derive the total wealth in stock i at the start of day t to be $S_t w_t(i)$ and the total wealth in stock i at the end of day t to be $S_t w_t(i) p_t(i)$. We can use a simple summation over the i stocks to find our total wealth at the end of day t as:

$$S_{t+1} = \sum_{i=1}^{N} S_t w_t(i) p_t(i) = S_t(\mathbf{w}_t \cdot \mathbf{p}_t)$$

The total wealth after T time periods is then:

$$S_{T+1} = \prod_{t=1}^{T} (\mathbf{w}_t \cdot \mathbf{p}_t)$$

2 Portfolio Selection

Our goal is to make as much money as possible. This is done by maximizing $\prod_{t=1}^{T}(\mathbf{w}_t \cdot \mathbf{p}_t)$. This is the same as maximizing the log of the expression, which is $\sum_{t=1}^{T} \ln(\mathbf{w}_t \cdot \mathbf{p}_t)$. This is also the same as minimizing the negative of the log of the expression, $\sum_{t=1}^{T} -\ln(\mathbf{w}_t \cdot \mathbf{p}_t)$. We thus notice that maximizing wealth is equivalent to minimizing an expression that looks like log loss, so we can view investing as an online learning problem:

```
for $t = 1, ..., T$:
    learner/investor chooses $\mathbf{w}_t$
    stock market/nature chooses $\mathbf{p}_t$
    loss $= -\ln(\mathbf{w}_t \cdot \mathbf{p}_t)$
```

In each round, the learner chooses w_t which is how he invests money, and the stock market responds with each stock going up or down that day as represented by $\mathbf{p}_t$. Our goal is then to minimize the cumulative loss.

A natural starting point is to aim to do almost as well as the best individual stock. We can do this by massaging our problem so that we can apply Bayes algorithm which is designed around log loss. We define the Bayes algorithm outcome space to be $X = \{0, 1\}$ and choose $C \geq p_t(i)\ \forall t, i$. For each timestep t of the algorithm, each expert i needs to come up with a probability distribution over the outcome space X by coming up with a probability for outcome 1, $p_{t,i}(1)$, and trivially $p_{t,i}(0) = 1 - p_{t,i}(1)$. Let the outcomes $x_t = 1\ \forall t$. We can then let $p_{t,i}(1) = \frac{p_t(i)}{C}$. Applying Bayes algorithm will give us back a weight vector $w_{t,i}$, and we use these weights for investing by setting $w_t(i) = w_{t,i}$. Now observe that:

$$q_t(x_t) = q_t(1) = \sum_i w_{t,i} p_{t,i}(1) = \sum_i \frac{w_{t,i} p_t(i)}{C} = \frac{\mathbf{w}_t \cdot \mathbf{p}_t}{C}$$

The bound on log loss guaranteed by Bayes algorithm says:

$$-\sum_t \ln q_t(x_t) \leq \min_i -\sum_t \ln p_{t,i}(x_t) + \ln N$$
$$-\sum_t \ln\left(\frac{\mathbf{w}_t \cdot \mathbf{p}_t}{C}\right) \leq \min_i -\sum_t \ln \frac{p_t(i)}{C} + \ln N$$
$$-\sum_t \ln(\mathbf{w}_t \cdot \mathbf{p}_t) \leq \min_i -\sum_t \ln p_t(i) + \ln N$$

This essentially says:

$$-\ln(\text{wealth of the algorithm}) \leq -\ln(\text{wealth of best stock}) + \ln N$$

We then remove the logs to find:

$$\text{wealth of the algorithm} \geq \frac{1}{N}(\text{wealth of best stock})$$

The algorithm is actually equivalent to the "buy and hold" strategy where on day 1 we invest $\frac{1}{N}$ of our wealth into each stock and then just leave it there. Since this means we will invest $\frac{1}{N}$ of our wealth into the best stock, the bound on the wealth of the algorithm naturally says that we will make at least $\frac{1}{N}$ of the wealth of the best stock, even in the case that we lose all our money in the other stocks. When we invest our money, ideally we want the money to grow exponentially at the rate c^t where c is a constant that is to be maximized. The bound on the wealth of the algorithm implies that the constant c that we get from the algorithm will be asymptotically at least as good as that of the best underlying stock.

3 Constant Rebalanced Portfolio

Instead of comparing with the best individual stock, we now look towards comparing with constant rebalanced portfolios (CRP). In a CRP, we decide ahead of time on fixed allocations among the different stocks and, everyday, rebalance the different portfolios so that they always have those fixed allocations. The simplest kind of CRP is a uniform CRP (UCRP) where everyday we rebalance equally among the N stocks. CRP is a very common strategy, as it is natural to constantly rebalance your portfolio so that you have, for example, 60% in

day	stock 1 price	stock 2 price	stock 1 price relative	stock 2 price relative
1	1	1	1	0.5
2	1	0.5	1	2
3	1	1	1	0.5
4	1	0.5	1	2
5	1	1	1	0.5

Figure 1: Stock 1 and 2 behavior

stock, 30% in bonds, and 10% in cash. If stocks go up and bonds go down, then CRP will have you sell stock to buy more bonds. This thus encourages buying low and selling high.

We present a concrete example where CRP is a good idea. Imagine there is stock 1 and stock 2 that both start at \$1. The price of stock 1 never changes, and the price of stock 2 is \$1 on odd days and \$0.50 on even days. The behavior of these two stocks is depicted for 5 days in Figure 1.

The buy and hold strategy will never earn money when applied to these two stocks. We then look at how UCRP performs:

$$\begin{aligned}
&S_1 = 1\\
&S_2 = S_1(\frac{1}{2}\cdot 1 + \frac{1}{2}\cdot\frac{1}{2}) = S_1\cdot\frac{3}{4}\\
&S_3 = S_2(\frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 2) = S_2\cdot\frac{3}{2} = S_1\cdot\frac{3}{4}\cdot\frac{3}{2}
\end{aligned}$$

More generally, if we have S_t on day t then $S_{t+2} = S_t\cdot\frac{3}{4}\cdot\frac{3}{2} = S_t\cdot\frac{9}{8}$. Thus every two days our wealth grows by 12.5%, so it grows exponentially.

4 Universal Portfolio Algorithm

We now return to making an algorithm to try to do almost as well as the best CRP instead of the best individual stock. Let us say that each CRP is a vector $\mathbf{b} = \langle b_1, ..., b_N\rangle$ which forms a valid distribution over the N stocks, and using the CRP means using $w_t(i) = b_i$. We then would want to reapply Bayes algorithm as we previously did, but instead of splitting wealth amongst stocks, we split it amongst all possible CRP's. There are uncountably infinite possible CRPs, so for each CRP, $\mathbf{b}$, we give it an infinitesimally small piece of our wealth, $d\mu(\mathbf{b})$. At the start of day t:

$$\text{wealth in CRP } \mathbf{b} = \prod_{s=1}^{t-1}(\mathbf{b}\cdot\mathbf{p}_s)d\mu(\mathbf{b})$$

$\prod_{s=1}^{t-1}(\mathbf{b}\cdot\mathbf{p}_s)$ is how much wealth we would have at the start of day t if we had started with \$1 invested in the CRP, and $d\mu(\mathbf{b})$ scales this wealth down as we only in fact invested an infinitesimally small amount into the CRP. We can simply integrate over the set of all possible CRP's $\mathbf{b}$ to find the total wealth at the start of day t:

$$\text{total wealth} = S_t = \int\prod_{s=1}^{t-1}(\mathbf{b}\cdot\mathbf{p}_s)d\mu(\mathbf{b})$$

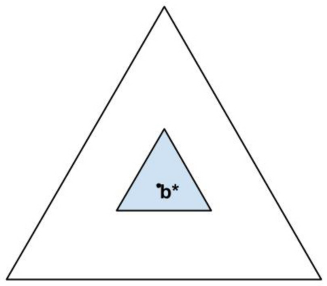


Figure 2: Simplex

We can use another integral over the set of all possible CRP's $\mathbf{b}$ to find the total wealth invested in stock i at the start of day t, where b_i is the fraction of our wealth in i, as:

$$\text{total wealth in stock } i = \int b_i \prod_{s=1}^{t-1} (\mathbf{b} \cdot \mathbf{p}_s) d\mu(\mathbf{b})$$

We can then calculate the fraction of our wealth we need to invest in stock i, $w_t(i)$ as:

$$w_t(i) = \frac{\int b_i \prod_{s=1}^{t-1} (\mathbf{b} \cdot \mathbf{p}_s) d\mu(\mathbf{b})}{\int \prod_{s=1}^{t-1} (\mathbf{b} \cdot \mathbf{p}_s) d\mu(\mathbf{b})}$$

Using this $w_t(i)$ to rebalance our portfolio at each time step is known as the Universal Portfolio (UP) Algorithm or Cover's algorithm.

5 Bounds on UP Algorithm

Theorem 1. *Wealth of UP algorithm* $\geq (\frac{1}{(T+1)^{N-1}})$*(Wealth of best CRP)*

While $(\frac{1}{(T+1)^{N-1}})$ may appear to be a small fraction, it holds for any stock market and still says the rate of exponential growth of this algorithm will eventually match the rate of exponential growth of the best CRP.

We will proceed to prove a slightly weaker version of this theorem in two steps. Let the best CRP be $\mathbf{b}^*$. Fortunately in our algorithm, we put part of our money into $\mathbf{b}^*$, but unfortunately we only put in an infinitesimally small amount. We note that the CRP's are essentially just probability vectors and so they live in the space of all probability vectors which is known as the simplex. We illustrate this simplex in the case that we have 3 stocks in Figure 2. $\mathbf{b}^*$ is just a point in the simplex and we consider the neighborhood around $\mathbf{b}^*$, which consists of CRPs that are close to $\mathbf{b}^*$, as illustrated by the inner shaded triangle in Figure 2. We then want to argue two points. In step 1, we want to argue that all of the CRPs in the neighborhood of $\mathbf{b}^*$ attain wealth close to $\mathbf{b}^*$. Then in step 2, we want to show that the overall size of the neighborhood is large. Let us define

$$\Delta = \{\text{all CRPs}\} = \{\mathbf{b} : b_i \geq 0, \sum_i b_i = 1\}$$

and we define the neighborhood of $\mathbf{b}^*$ as:

$$N(\mathbf{b}^*) = \{(1-\alpha)\mathbf{b}^* + \alpha\mathbf{z} : \mathbf{z} \in \Delta\}$$

where α is a small positive number and we are essentially mixing $\mathbf{b}^*$ with some other distribution $\mathbf{z}$.

Proof. Step 1:
Let us say that $\mathbf{b}$ is one of the points in the neighborhood of $\mathbf{b}^*$ and so
$\mathbf{b} = (1-\alpha)\mathbf{b}^* + \alpha\mathbf{z}$. We can derive the amount of wealth that $\mathbf{b}$ gains at time t to be:

$$\mathbf{b} \cdot \mathbf{p}_t = (1-\alpha)\mathbf{b}^* \cdot \mathbf{p}_t + \alpha\mathbf{z} \cdot \mathbf{p}_t$$

$\mathbf{z} \cdot \mathbf{p}_t \geq 0$ as price relatives are never negative. Thus, after T timesteps:

$$\text{wealth of } \mathbf{b} \geq (1-\alpha)^T (\text{wealth of } \mathbf{b}^*)$$

□

Proof. Step 2:

$$\text{Vol}(N(\mathbf{b}^*)) = \text{Vol}(\{(1-\alpha)\mathbf{b}^* + \alpha\mathbf{z} : \mathbf{z} \in \Delta\})$$

where Vol() denotes volume. We now note that $(1-\alpha)\mathbf{b}^*$ is a fixed quantity, so the total volume will be the same if we shift the simplex and remove this quantity:

$$\begin{aligned} \text{Vol}(N(\mathbf{b}^*)) &= \text{Vol}(\{\alpha\mathbf{z} : \mathbf{z} \in \Delta\}) \\ &= \text{Vol}(\Delta) \cdot \alpha^{N-1} \end{aligned}$$

The last equality holds as the simplex is an $N-1$ dimensional object and each dimension is being scaled by α. □

We can now combine the results of the two steps to show:

$$\begin{aligned} \text{wealth of UP algorithm} &\geq (\text{fraction of CRPs in } N(\mathbf{b}^*))(\text{minimum wealth of any CRP in } N(\mathbf{b}^*)) \\ &\geq \alpha^{N-1}(1-\alpha)^T (\text{wealth of } \mathbf{b}^*) \\ &\geq \frac{1}{e(T+1)^{N-1}} (\text{wealth of } \mathbf{b}^*) \end{aligned}$$

where the last inequality holds if we choose $\alpha = \frac{1}{T+1}$. Thus, we have proved a slightly weaker version of the theorem.

6 Game Theory

Game Theory is a field that studies games and is really about interactions between players of all kinds. There is a natural connection to learning, as in learning there is often an interaction between a teacher and a student or between a learner and nature. For our purposes, a game is defined by a matrix. The game matrix for rock, paper, scissors is shown in Figure 3. The rows of the matrix are actions that the row player Mindy can take, and the columns are actions that the column player Max can take. Mindy chooses one of the rows, and Max chooses one of the columns. The entry that is defined by these choices

	R	P	S
R	0.5	1	0
P	0	0.5	1
S	1	0	0.5

Figure 3: Rock, Paper, Scissors Game Matrix

in the matrix is the loss suffered by Mindy. Max tries to maximize this loss while Mindy tries to minimize it. In general we will always have a game matrix M. To play, Mindy chooses row i, and Max chooses column j. Individual rows i and columns j are called pure strategies. The corresponding entry in the matrix $M(i, j)$ is the loss suffered by Mindy. In principle, any two-person zero-sum game can be put into this form.

COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Jordan Ash

Lecture #24
May 1, 2014

1 Review of Game Theory:

Let M be a matrix with all elements in $[0, 1]$. Mindy (called the row player) chooses row i while Max (called the column player) chooses column j. In this case, from Mindy's expected loss is:

$$Loss = M(i, j)$$

Alternatively, Mindy could select a move randomly from a distribution P over the rows and Max could select a move randomly from a distribution Q over the columns. Here, the expected loss for Mindy is:

$$Loss = \sum_{i,j} P(i)M(i, j)Q(j) = P^T MQ = M(P, Q)$$

P and Q are called "mixed strategies," while i and j are called "pure strategies."

2 Minimax Theorem:

In some games, such as Rock, Paper, Scissors, players move at exactly the same time. In this way, both players have the same information available to them at the time of moving. Now we suppose that Mindy plays first, followed by Max. Max knows the P that Mindy chose, and further knows $M(P, Q)$ for any Q he chooses. Consequently, he chooses a Q that maximizes $M(P, Q)$. Because Mindy knows that Max will choose $Q = \arg\max_Q M(P, Q)$ for any P she chooses, she selects a P that minimizes $\max_Q M(P, Q)$. Thus, if Mindy goes first, she could expect to suffer a loss of $\min_P \max_Q M(P, Q)$. Overall, it may initially seem like the player to go second has an advantage because she has more information available to her. From Mindy's perspective again, this leads to:

$$\max_Q \min_P M(P, Q) \leq \min_P \max_Q M(P, Q)$$

So Mindy playing after Max seems to be better than if the two play in reverse order. However, John von Neumann showed that the expected outcome of a game is always the same, regardless of the order in which players move.

$$v = \max_Q \min_P M(P, Q) = \min_P \max_Q M(P, Q)$$

Here, v denotes the value of the game. This may seem counterintuitive, because the player that goes second has more information available to her at the time of choosing a move. We will prove the above statement using an online learning algorithm. Let:

$$P^* = \arg\min_P \max_Q M(P,Q)$$
$$Q^* = \arg\max_Q \min_P M(P,Q)$$

Then,

$$\forall Q : M(P^*, Q) \leq v \tag{1}$$
$$\forall P : M(P, Q^*) \geq v \tag{2}$$

In other words, for some optimal P^*, the maximum loss that Max could cause is bounded by v and Mindy's loss is at least v, regardless of the particular strategies they choose.

If we had knowledge of M, we might be able to find P^* by employing techniques from linear programming. However, we don't necessarily have this knowledge, and even if we did, M could be massively large. Further, P^* applies here only for opponents that are perfectly adversarial, so it doesn't account for an opponent that might make mistakes. Thus, it makes sense to try to learn M and Q iteratively.

We do this with the following formulation:

```
for $t = 1, \dots, T$
    Mindy chooses $P_t$
    Max chooses $Q_t$ (with knowledge of $P_t$)
    Mindy observes $M(i, Q_t) \forall i$
    Loss = $M(P_t, Q_t)$
end
```

Clearly, the total loss of this algorithm is simply $\sum_{t=1}^{T} M(P_t, Q_t)$. We want to be able to compare this loss to the best possible loss that could have been achieved by fixing any single strategy for all T iterations. In other words, we want to show:

$\sum_{t=1}^{T} M(P_t, Q_t) \leq \min_P \sum_{t=1}^{T} M(P, Q_t) +$ [Small Regret Term]

2.1 Multiplicative Updates

Suppose we use a multiplicative weight algorithm that updates weights in the following way, where n is the number of rows in matrix M:

$$\beta \in [0,1) \tag{3}$$

$$P_1(i) = \frac{1}{n}\forall i \tag{4}$$

$$P_{t+1}(i) = \frac{P_t(i)\beta^{M(i,Q_t)}}{\text{Normalizing Constant}} \tag{5}$$

Our algorithm is similar to the weighted majority algorithm. The idea is decrease the probability of choosing a particular row proportionally to the loss suffered by selecting that row. After making an argument using potentials, we could use this algorithm to obtain the following bound:

$$\sum_{t=1}^{T} M(P_t, Q_t) \leq \alpha_\beta \min_P \sum_{t=1}^{T} M(P, Q_t) + c_\beta \ln(n) \tag{6}$$

where $\alpha_\beta = \frac{ln(\frac{1}{\beta})}{1-\beta}$ and $c_\beta = \frac{1}{1-\beta}$.

2.2 Corollary

We can choose β such that:

$$\frac{1}{T}\sum_{t=1}^{T} M(P_T, Q_T) \leq \min_P \frac{1}{T}\sum_{t=1}^{T} M(P, Q_t) + \Delta_T \tag{7}$$

where $\Delta_T = O(\sqrt{\frac{ln(n)}{T}})$, which goes to zero for large T. In other words, the loss suffered by Mindy per round approaches the optimal average loss per round. We'll use this result to prove the Minimax theorem.

2.3 Proof

Suppose that Mindy uses the above algorithm to choose P_t, and that Max chooses Q_t such that $Q_t = \arg\max_Q M(P_T, Q)$, maximizing Mindy's loss. Also, let:

$$\bar{P} = \frac{1}{T}\sum_{t=1}^{T} P_t \tag{8}$$

$$\bar{Q} = \frac{1}{T}\sum_{t=1}^{T} Q_t \tag{9}$$

We also know intuitively, as mentioned before, that $\max_Q \min_P M(P,Q) \leq \min_P \max_Q M(P,Q)$, because as stated earlier, the player that goes second has more information available to her.

To show equality, which would prove the Minimax theorem stated earlier, it's enough to show that $\max_Q \min_P M(P,Q) \geq \min_P \max_Q M(P,Q)$ also.

$$\min_P \max_Q P^T MQ \leq \max_Q \bar{P}^T MQ$$

By definition of $\bar{P}$:

$$= \max_Q \frac{1}{T} \sum_{t=1}^{T} P_t^T MQ$$

By convexity:

$$\leq \frac{1}{T} \max_Q \sum_{t=1}^{T} P_t^T MQ$$

By definition of Q_t:

$$= \frac{1}{T} \sum_{t=1}^{T} P_t^T MQ_t$$

By corollary 2.2:

$$\leq \min_P \frac{1}{T} \sum_{t=1}^{T} P^T MQ_t + \Delta_T$$

By definition of $\bar{Q}$:

$$= \min_P P^T M\bar{Q} + \Delta_T$$

$$\leq \max_Q \min_P P^T MQ + \Delta_T$$

The proof is finished because Δ_T goes to zero as T gets large. This proof also shows that:

$$\max_Q \bar{P}^T MQ \leq v + \Delta_T$$

where $v = \max_Q \min_P P^T MQ$. If we take the average of the P_t terms computed at each round of the algorithm, we get something within Δ_T of the optimal value. Because Δ_T goes to zero for large values of T, we can get closer to the optimal strategy by simply increasing T. In other words, this strategy becomes more and more optimal as the number of rounds T increases. For this reason, $\bar{P}$ is called an approximate min max strategy. A similar argument could be made to show that $\bar{Q}$ is an approximate max min strategy.

3 Relation to Online Learning

In order to project our analysis into an online learning framework, consider the following problem setting:

```
for $t = 1, \ldots, T$
    Observe $x_t$ from $X$
    Predict $\hat{y_t} \in \{0, 1\}$
    Observe true label $c(x_t)$
end
```

Here we consider each hypothesis h as being an expert from the set of all hypotheses H. We want to show that:

number of mistakes $\leq$ number of mistakes of best h + [Small Regret Term]

We set up a game matrix M where $M(i, j) = M(h, x) = 1$ if $h(x) \neq c(x)$ and 0 otherwise. Thus, the size of this matrix is $|H| \cdot |X|$. Given an x_t, the algorithm must choose some P_t, a distribution used to predict x_t's label. h is chosen according to the distribution P_t, and then $\hat{y_t}$ is chosen as $h(x_t)$. Q_t in this context is the distribution concentrated on x_t (is 1 at x_t and 0 at all other $x \in X$). Consequently:

$$\begin{aligned}
&\sum_{t=1}^{T} M(P_t, x_t) \\
&= \text{E[number of mistakes]} \\
&\leq \min_h \sum_{t=1}^{T} M(h, x_t) + \text{[Small Regret Term]}
\end{aligned}$$

Notice that $\min_h \sum_{t=1}^{T} M(h, x_t)$ is equal to the number of mistakes made by the best hypothesis h. If we substitute $M(P_t, x_t)$ with $\sum_h P_t(h) \cdot 1\{h(x_t) \neq c(x_t)\} = Pr[h(x) \neq c(x)]$ above, we obtain the same bound as was found in the previous section.

4 Relation to Boosting

We could think of boosting as a game between the boosting algorithm and the weak learner it calls. Consider the following problem:

```
for $t = 1, \ldots, T$
    The boosting algorithm selects a distribution $D_t$ over the training set samples $X$
    The weak learner chooses a hypothesis $h_t$
end
```

Here we assume that all weak learners h_t obey the weak learning assumption, i.e. that $\Pr_{(x,y)\sim D_t}[h_t(x) \neq y)] \leq \frac{1}{2} - \gamma$ and $\gamma > 0$. We could define the game matrix M' in terms of the matrix M used in the last section. However, here we want a distribution over the X samples rather than over the hypotheses, so we need to transpose and normalize M.

$M' = 1 - M^T$

In other words, $M'(i,j) = M'(x,h) = 1$ if $h(x) = c(x)$ and 0 otherwise. Here, $P_t = D_t$, and Q_t is a distribution fully concentrated on the particular h_t chosen by the weak learner. We could apply our same analysis from the multiplicative weights algorithm:

$\frac{1}{T}\sum_{t=1}^{T} M'(P_t, h_t) \le \min_x \frac{1}{T}\sum_{t=1}^{T} M'(x, h_t) + \Delta_T$

Also,

$M'(P_t, h_t) = \sum_x P_T(x) \cdot 1\{h(x) = c(x)\} = Pr[h_t(x) = c(x)] \ge \frac{1}{2} + \gamma$

Combining these facts:

$$\begin{aligned}\frac{1}{2} + \gamma &\le \frac{1}{T}\sum_{t=1}^{T} M'(x, h_t) \\ &\le \min_x \frac{1}{T}\sum_{t=1}^{T} M'(x, h_t) + \Delta_T\end{aligned}$$

Rearranging,

$$\forall x : \frac{1}{T}\sum_{t=1} TM'(x, h_t) \ge \frac{1}{2} + \gamma - \Delta_T > \frac{1}{2}$$

which is again true because Δ_T approaches 0 as T gets large. In other words, we have found that over $\frac{1}{2}$ of the weak hypotheses correctly classify any x when T gets sufficiently large. Because the final hypothesis is just a majority vote of these weak learners, we have proven that the boosting algorithm drives training error to zero when enough weak learners are employed.