(67577) Introduction to Machine Learning October 19, 2009

Lecture 1 – Introduction

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

1 What is learning?

The subject of this course is automated learning, or, as we will more often use, machine learning (ML for short). Roughly speaking, we wish to program computers so that they can "learn".

Before we discuss how machines can learn, or how the process of learning can be automated, let us consider two examples of naturally occurring animal learning. Not surprisingly, some of the most fundamental issues in ML arise already in that context, that we are all familiar with.

1. **Bait Shyness - rats learning to avoid poisonous baits.** It is well known that, when eating a substance is followed by illness an animal may associate the illness with the eaten substance and avoid that food in the future. This is called conditioned taste aversion or "bait shyness". For example, when rats encounter novel food items, they will first eat very small amounts, and subsequent feeding will depend on the flavor of the food and its physiological effect. If the food produces an ill effect, the food will often be associated with the illness, and subsequently, the rats will not eat it. Clearly, there is a learning mechanism in play here- if past experience with some food was negatively labeled, the animal predicts that it will also have a negative label when encountered in the future. Naturally, bait shyness is often a useful survival mechanism.

2. **Pigeon superstition. Something goes wrong.** Another demonstration of naive animal "learning" is the classic "pigeon superstition" experiment. In that experiment, the psychologist B.F. Skinner placed a bunch of hungry pigeons in a cage attached to an automatic mechanism that delivered food to the pigeons "at regular intervals with no reference whatsoever to the bird's behavior." What happens in such an experiment is that the hungry pigeons go around the cage pecking at random objects. When food is delivered, it finds each pigeon pecking at some object. Consequently, each bird tends to spend some more pecking time at that lucky object. That, in turn, increases the chance that the next random food spraying will find each bird at that location of hers. What results is chain of events that reinforces the pigeons' association of the delivery of the food with whatever chance actions they had been performing when it was first delivered. They subsequently continue to perform these same actions diligently (see http://psychclassics.yorku.ca/Skinner/Pigeon/).

What distinguishes learning mechanisms that result in superstition from useful learning? This question is crucial to the development of automated learners. While human learners can rely on common sense to filter out random meaningless learning conclusions, once we export the task of learning to a program, we must provide well defined crisp principles that will protect the program from reaching senseless/useless conclusions. The development of such principles is a central goal of the theory of machine learning. As a first step in this direction, let us have a closer look at the bait shyness phenomenon in rats.

3. **Bait Shyness revisited: rats fail to acquire conditioning between food and electric shock or between sound and nausea.** The bait shyness mechanism is rats turn out to be more complex than what one may expect. In experiments carried out by Garcia, it was demonstrated that if the unpleasant stimulus that follows food consumption is replaced by, say, electrical shock (rather than nausea), then no conditioning occurs. That is, even after repeated trials in which the consumption of some food is followed by the administration of unpleasant electrical shock, the rats do not tend to avoid that food. Similar failure of conditioning occurs when the characteristics of the food that implies nausea

is replaced by a vocal signal (rather than the taste or smell of that food). Clearly, the rats have some "built in" prior knowledge telling them that, while temporal correlation between food and nausea can be causal, it is unlikely that there will be a causal relationship between food consumption and electrical shocks.

Now, we can observe that one distinguishing feature between the bait shyness learning and the pigeon superstition is the incorporation of *prior knowledge* that biases the learning mechanism. The pigeons in the experiment are willing to adopt *any* explanation to the occurrence of food. However, the rats "know" that food cannot cause an electric shock and that the co-occurrence of noise with some food is not likely to effects the nutritional value of that food. The rats learning process is biased towards detecting some kind of patterns while ignoring other temporal correlations between events. It turns out that the incorporation of prior knowledge, biasing the learning process, is inevitable for the success of learning algorithm (this is formally stated and proved as the "No Free Lunch theorem"). The development of tools for expressing domain expertise, translating it into a learning bias, and quantifying the effect of such a bias on the success of learning, is a central theme of the theory of machine learning. Roughly speaking, the stronger the prior knowledge (or prior assumptions) that one starts the learning process with, the easier it is to earn from further examples. However, the stronger these prior assumptions are, the less flexible the learning is - it is bound, a priory, by the commitment to these assumptions. We shall discuss these issues explicitly later in the next lecture.

2 Why automating the process of learning?

Computers are being applied to almost every aspect of our lives. However, with all their innumerable successful applications, there are several inherent limitations to what humanly-written program can do. One limiting factor is the need for explicit, detailed, specification; in order to write a program to carry out some function, the programmer needs to fully understand, down to the level of the smallest details, how that function should be executed. Computers only follow the program's instructions and have no way of handling ambiguity or to adapt to scenarios that are not explicitly addressed by the program's instructions. In contrast to that, taking example from intelligent beings, many of our skills are acquired or refined through *learning* from our experience (rather than following explicit instructions given to us). Machine learning concerns with endowing programs with the ability to "learn" and adapt. Many tasks currently carried out automatically utilize machine learning components.

Consider for example the task of driving. It would have been extremely useful to have reliable automated drivers. They would never doze off, never get drunk or angry at the other driver, they could be sent out to dangerous roads without risking human lives and so on and on. Why don't we see automated drivers around us on the roads (at least not in 2010 when this handouts are being written)? It does not seem to require much intelligence to drive, computers are already routinely performing way more sophisticated tasks. The obstacle is exactly in the gap between what we can provide exact instructions for and what we have to rely on experience to shape up and refine. While we all drive, we acquire much of driving skills through practice. We do not understand our driving decision-making well enough to be able to translate our driving skills into a computer program. Recent significant progress in the development of automated drivers relies on programs that can improve with experience - machine learning programs.

Automated driving, medical research, natural language processing (including speech recognition and machine translation), credit card fraud detection, are only a few of the many areas where machine learning is playing a central role in applying computers to substitute for human "thinking" and decision making.

Two aspects of a problem may call for the use of programs that learn and improve based on their "experience"; the problem's complexity and its "adaptivity".

Tasks that are too complex to program.

- *Tasks performed by animals/humans:* there are numerous tasks that, although we perform routinely, our introspection, concerning how we do them, is not sufficiently elaborate to extract a well defined program. Examples of such tasks include driving, speech recognition, and face recognition. In all of these tasks, state of the art ML programs, programs that "learn from their experience", achieve quite satisfactory results, once exposed to sufficiently many training examples.
- *Tasks beyond human capabilities:* another wide family of tasks that benefit from machine learning techniques are related to the analysis of very large and complex data sets: Astronomical data, turning medical archives into medical knowledge, weather prediction, analysis of genomic data, web search engines, and electronic commerce. With more and more available electronically recorded data, it becomes obvious that there are treasures of meaningful information buried in data archives that are way too large and too complex for humans to make sense of. Learning to detect meaningful patterns in large and complex data sets is a promising domain in which the combination of programs that learn with the almost unlimited memory capacity and processing speed of computers open up new horizons.

Cope with diversity (Adaptivity). One limiting feature of programmed tools is their rigidity - once the program has been written down and installed, it stays unchanged. However, many tasks change over time or from one user to another in a way that requires the way we handle them to adapt. Machine learning tools - programs whose behavior adapts to their input data - offer a solution to such issues; they are, by nature, adaptive to changes in the environment they interact with. Typical successful applications of machine learning to such problems include programs that decode hand written text, where a fixed program can adapt to variations between the handwriting of different users, spam detection programs, adapting automatically to changes in the nature of spam emails, and speech recognition programs (again, a scenario in which a fixed program is required to handle large variability in the type on inputs it is applied to).

2.1 Types of learning

Learning is, of course, a very wide domain. Consequently, the field of machine learning has branched into several subfields dealing with different types of learning tasks. We give a rough taxonomy of learning paradigms, aiming to provide some perspective of where the content of this course sits within the wide field of machine learning.

The first classification we make regards the goal of the learning process. In the discriminative learning setting, the learner uses past experience in order to predict properties of future examples. That is, learning can be defined as the process of *using experience to become an expert*. The goal of the learning process should be well defined in advance, before seeing any examples. Discriminative learning is convenient because there is a clear measure of success – how good the predictions of the learner on future examples are. In non-discriminative learning, the goal of the learner is not well defined in advance. The learner aims to "understand" the data e.g. by learning a generative probabilistic model that fits the data. In this case, learning can be described as the process of finding *meaningful simplicity in the midst of disorderly complexity*. In this course we mostly deal with discriminative learning.

Next, we describe four parameters along which learning paradigms can be further classified.

Supervised vs. Unsupervised Since learning involves an interaction between the learner and the environment, one can divide learning tasks according to the nature of that interaction. The first distinction to note is the difference between supervised and unsupervised learning. As an illustrative example, consider the task of learning to detect spam email versus the task of anomaly detection. For the spam detection task, we consider a setting in which the learner receives training emails for which the label `spam/not-spam` is provided. Based on such training the learner should figure out a rule for labeling a newly arriving email message. In contrast, for the task of anomaly detection, all the learner gets as training is a large body of email messages and the learner's task is to detect "unusual" messages.

More abstractly, viewing learning as a process of "using experience to gain expertise", supervised learning describes a scenario in which the "experience", a training example, contains significant information that is missing in the "test examples" to which the learned expertise is to be applied (say, the `Spam/no-Spam` labels). In this setting, the acquired expertise is aimed to predict that missing information for the test data. In such cases, we can think of the environment as a teacher that "supervises" the learner by providing the extra information (labels). In contrast with that, in unsupervised learning, there is no distinction between training and test data. The learner processes input data with the goal of coming up with some summary, or compressed version of that data. Clustering a data set into subsets of similar objets is a typical example of such a task.

There is also an intermediate learning setting in which, while the training examples contain more information than the test examples, the learner is required to predict even more information for the test examples. For example, one may try to learn a value function, that describes for each setting of a chess board the degree by which White's position is better than the Black's. Such value functions can be learned based on a data base that contains positions that occurred in actual chess games, labeled by who eventually won that game. Such learning framework are mainly investigated under the title of *'reinforcement learning'*.

Active vs. Passive learners Learning paradigms can vary by the role played by the learner. We distinguish between 'active' and 'passive' learners. An active learner interacts with the environment at training time, say by posing queries or performing experiments, while a passive learner only observes the information provided by the environment (or the teacher) without influencing or directing it. Note that, the learner of a spam filter is usually passive - waiting for users to mark the emails arriving to them. In an active setting, one could imagine asking users to label specific emails chosen by the learner, or even composed by the learner to enhance its understanding of what spam is.

Helpfulness of the teacher When one thinks about human learning, of a baby at home, or a student at school, the process often involves a helpful teacher. A teacher trying to feed the learner with the information most useful for achieving the learning goal. In contrast, when a scientist learns about nature, the environment, playing the role of the teacher, can be best thought of as passive - apples drop, stars shine and the rain falls without regards to the needs of the learner. We model such learning scenarios by postulating that the training data (or the learner's experience) is generated by some random process. This is the basic building block in the branch of 'statistical learning'. Finally, learning also occurs when the learner's input is generated by an adversarial "teacher". This may be the case in the spam filtering example (if the spammer makes an effort to mislead the spam filtering designer) or in learning to detect fraud. One also uses an adversarial teacher model as a worst-case-scenario, when no milder setup can be safely assumed. If you can learn against an adversarial teacher, you are guaranteed to succeed interacting any odd teacher.

Online vs. Batch learning protocol The last parameter we mention is the distinction between situations in which the learner has to respond online, throughout the learning process, to settings in which the learner has to engage the acquired expertise only after having a chance to process large amounts of data. For example, a stock broker has to make daily decisions, based on the experience collected so far. He may become an expert over time, but might have made costly mistakes in the process. In contrast, in many data mining settings, the learner - the data miner - has large amounts of training data to play with before having to output conclusions.

In this course we shall discuss only a subset of the possible learning paradigms. Our main focus is on supervised statistical batch learning with a passive learner (like for example, trying to learn how to generate patients' prognosis, based on large archives of records of patients that were independently collected and are already labeled by the fate of the recorded patients). We shall also briefly discuss online learning and batch unsupervised learning (in particular, clustering). Maybe the most significant omission here, at least from the point of view of practical machine learning, is that this course does not address reinforcement learning.

(67577) Introduction to Machine Learning October 20, 2009

Lecture 2 – A gentle start

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In this lecture we give a first formal treatment of learning. We focus on a learning model called the PAC model. We aim at rigorously showing how data can be used for learning as well as how overfitting might happen if we are not careful enough.

3 A statistical learning framework

Imagine you have just arrived in some small Pacific island. You soon find out that papayas are a significant ingredient in the local diet. However, you have never before tasted papayas. You have to learn how to predict whether a papaya you see in the market is tasty or not. There are two obvious features that you can base your prediction on; the papaya's color, ranging from dark green, through orange and red to dark brown, and there is the papaya's softness, ranging from rock hard to mushy. Your input for figuring out your prediction rule is a sample of papayas that you have examined for color and softness and then tasted and found out if they were tasty or not. This is a typical learning problem.

3.1 A Formal Model

The Learner's Input: In the basic statistical learning setting, the learner has access to the following:

- **Domain Set:** An arbitrary set, $\mathcal{X}$. This is the set of objects that we may wish to label. For example, these could be papayas that we wish to classify as tasty or not-tasty, or email messages that we wish to classify as spam or not-spam. Usually, these domain points will be represented by a vector of *features* (like the papaya's color, softness etc.).
- **Label Set:** For our discussion, we will restrict $\mathcal{Y}$ to be a two-element set, usually, $\{0, 1\}$ or $\{-1, +1\}$. In our papayas example, let $+1$ represents being tasty and -1 being not-tasty.
- **Training data:** $S = ((x_1, y_1) \ldots (x_m, y_m))$ is a finite sequence of pairs in $\mathcal{X} \times \mathcal{Y}$. That is, a sequence of labeled domain points. This is the input that the learner has access to (like a set of papayas that have been tasted and their tastiness recorded).

The Learner's Output: The learner outputs a *hypothesis* or a *prediction rule*, $h : \mathcal{X} \to \mathcal{Y}$. This function can be then used to predict the label of new domain points. In our papayas example, it is the rule that our learner will employ to decide whether a future papaya she examines in the farmers market is going to be tasty or not.

A Measure of success: We assume that the data we are interested in (the papayas we encounter) is generated by some probability distribution (say, the environment). We shall define the quality of a given hypothesis as the chance that it has to predict the correct label for a data point that is generated by that underlying distribution. Equivalently, the *error of a hypothesis* quantifies how likely it is to make an error when labeled points are randomly drawn according to that data-generating distribution.

Formally, we model the environment as a probability distribution, $\mathcal{D}$, over $\mathcal{X} \times \mathcal{Y}$. Intuitively, $D(\mathbf{x}, y)$ determines how likely is it to observe a and define the error of a classifier to be:

$$\mathrm{err}_{\mathcal{D}}(h) \stackrel{\mathrm{def}}{=} \mathop{\mathbb{P}}_{(\mathbf{x},y)\sim\mathcal{D}}[h(\mathbf{x}) \neq y] \stackrel{\mathrm{def}}{=} \mathcal{D}(\{(\mathbf{x}, y) : h(\mathbf{x}) \neq y\}) \,. \tag{1}$$

That is, the error of a classifier h is the probability to randomly choose an example $(\mathbf{x}, y)$ for which $h(\mathbf{x}) \neq y$. The subscript $\mathcal{D}$ reminds us that the error is measured with respect to the probability distribution $\mathcal{D}$ over our "world", $\mathcal{X} \times \mathcal{Y}$. We omit this subscript when it is clear from the context. $\mathrm{err}_D(h)$ has several synonymous names such as the *generalization error*, the *test error*, or the *true error* of h.

Note that this modeling of the world allows sampling the same instance with different labels. In the papayas example, this amounts to allowing two papayas with the same color and softness such that one of them is tasty and the other is not. In some situations, the labels are determined deterministically once the instance is fixed. This scenario can be derived from the general model as a special case by imposing the additional requirement that the conditional probability (according to $\mathcal{D}$) to see a label y given an instance $\mathbf{x}$ is either 1 or 0.

i.i.d. assumption: The learner is blind to the underlying distribution $\mathcal{D}$ over the world. In our papayas example, we have just arrived to a new island and we have no clue as to how papayas are distributed. The only way the learner can interact with the environment is through observing the training set. Of course, to facilitate meaningful learning, there must be some connection between the examples in the training set and the underlying distribution. Formally, we assume that each example in the training set is independently and identically distributed (i.i.d.) according to the distribution $\mathcal{D}$. Intuitively, the training set S is a window throughout we look at the distribution $\mathcal{D}$ over the world.

4 Empirical Risk Minimization

Recall that a learning algorithm receives as input a training set S, sampled i.i.d. from an unknown distribution $\mathcal{D}$, and should output a predictor $h_S : \mathcal{X} \to \mathcal{Y}$, where the subscript S emphasizes the fact that the output predictor depends on S. The error of h_S is the probability that h_S errs on a random example sampled according to $\mathcal{D}$. Since we are blind to the distribution over the world, a natural and intuitive idea is to choose one of the predictors that makes a minimal number of mistakes on the training set, S. This learning paradigm is called *Empirical Risk Minimization* or ERM for short.

Formally, We denote the average number of mistakes a predictor makes on a training set S by

$$\mathrm{err}_S(h) = \frac{|\{i : h(\mathbf{x}_i) \neq y_i\}|}{m} .$$

This quantity is also called the *empirical* risk of h or the *training error* of h. Then, the ERM rule finds a predictor h that minimizes $\mathrm{err}_S(h)$. Note that there may be several predictors that minimize the training error and an ERM algorithm is free to choose any such predictor.

4.1 Something goes wrong

Although the ERM rule seems like a natural learning algorithm, as we show next, without being careful this approach can miserably fail.

Recall again the problem of learning the taste of a papaya based on its shape and color. Consider an i.i.d. sample of m examples as depicted in Figure 1. Assume that the probability distribution $\mathcal{D}$ is such that instances are distributed uniformly within the gray square and the labels are determined deterministically to be 1 if the instance is within the blue square and 0 otherwise. The area of the gray square in the picture is 2 and the area of the blue square is 1. Consider the following predictor:

$$h_S(\mathbf{x}) \;=\; \begin{cases} y_i & \text{if } \exists i \text{ s.t. } \mathbf{x}_i = \mathbf{x} \\ 0 & \text{otherwise} \end{cases} . \tag{2}$$

Clearly, no matter what the sample is, $\mathrm{err}_S(h_S) = 0$, and therefore the classifier may be chosen by an ERM algorithm since it is one of the empirical-minimum cost hypotheses. On the other hand, it is clear that the

generalization error of any classifier that predicts the label 1 only on a finite number of instances is $1/2$. Thus, $\mathrm{err}_{\mathcal{D}}(h_S) = 1/2$. This is a clear example of *overfitting*. We found a predictor whose performance on the training set is excellent but whose performance on the true world is very bad. Intuitively, overfitting occurs when we can explain every set of examples. The explanations of someone that can explain everything are suspicious.

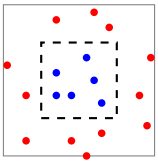


Figure 1: An illustration of a sample for the Papaya taste learning problem.

5 Empirical Risk Minimization with inductive bias

In the example shown previously, we showed that the ERM rule might lead to overfitting – we found a predictor that has excellent performance on the training set but has a very bad performance on the underlying distribution. How can we avoid overfitting?

A possible solution is to apply the ERM learning rule, but restrict the search space. Formally, the learner should choose in advance (before seeing the data) a set of predictors. This set is called a *hypothesis class* and is denoted by $\mathcal{H}$. That is, each $h \in \mathcal{H}$ is a function mapping from $\mathcal{X}$ to $\mathcal{Y}$. After deciding on $\mathcal{H}$, the learner samples a training set S and uses the ERM rule to choose a predictor *out of the hypothesis class*. The learner may choose a hypothesis $h \in \mathcal{H}$, which minimizes the error over the training set. By restricting the learner to choose a predictor from $\mathcal{H}$ we *bias* it toward a particular set of predictors. This preference is often called an *inductive bias*. Since $\mathcal{H}$ is chosen in advance we refer to it as a prior knowledge on the problem.

A fundamental question in learning theory, which we will study later in the course, is what hypothesis classes guarantee learning without overfitting. That is, how the restriction of the search space to those predictors in $\mathcal{H}$ saves us from overfitting. Intuitively, choosing a more restricted hypothesis class better protects us against overfitting but at the same time might cause us a larger inductive bias. We will get back to this fundamental tradeoff later.

Next, we show how a restriction of the search space to a finite hypothesis class prevents overfitting.

5.1 A finite hypothesis class

Let $\mathcal{H}$ be a finite hypothesis class. For example, $\mathcal{H}$ can be the set of all predictors that can be implemented by a C++ program whose length is at most k bits. Or, in our papayas example, $\mathcal{H}$ can be the set of all rectangles whose coordinates are taken from a grid. The learning algorithm is allowed to use the training set for deciding which predictor to choose from the hypothesis class $\mathcal{H}$. In particular, we will analyze the performance of the "biased" ERM learning rule:

$$h_S \in \operatorname*{argmin}_{h \in \mathcal{H}} \mathrm{err}_S(h) \,, \tag{3}$$

where ties are broken in some arbitrary way.

To simplify the analysis of the biased ERM rule we make one additional assumption (that will be relaxed later in this course).

Realizable assumption: Exists $h^\star \in \mathcal{H}$ s.t. $\mathrm{err}_D(h^\star) = 0$. This assumption implies that for any training set S we have $\mathrm{err}_S(h^\star) = 0$ with probability 1.

From the realizable assumption and the definition of the ERM rule given in Eq. (3), we have that $\mathrm{err}_S(h_S) = 0$ (with probability 1). But, what we are interested in is the generalization error of h_S, that is $\mathrm{err}_{\mathcal{D}}(h_S)$. Since $\mathrm{err}_{\mathcal{D}}(h_S)$ depends on the training set it is a random variable and therefore we will analyze the probability to sample a training set for which $\mathrm{err}_{\mathcal{D}}(h_S)$ is not too large. Formally, let ϵ be an accuracy parameter, where we interpret the event $\mathrm{err}_{\mathcal{D}}(h_S) > \epsilon$ as a severe overfitting, while if $\mathrm{err}_{\mathcal{D}}(h_S) \leq \epsilon$ we accept the output of the algorithm to be an approximately correct predictor. Therefore, we are interested in calculating

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_{\mathcal{D}}(h_S) > \epsilon] \,.$$

Let H_B be the set of "bad" hypotheses, that is $H_B = \{h \in \mathcal{H} : \mathrm{err}_{\mathcal{D}}(h) > \epsilon\}$. As mentioned previously, the realizable assumption implies that $\mathrm{err}_S(h_S) = 0$ with probability 1. This also implies that the event $\mathrm{err}_{\mathcal{D}}(h_S) > \epsilon$ can only happen if for some $h \in \mathcal{H}_B$ we have $\mathrm{err}_S(h) = 0$. Therefore, the set $\{S : \mathrm{err}_{\mathcal{D}}(h_S) > \epsilon\}$ is contained in the set $\{S : \exists h \in \mathcal{H}_B, \mathrm{err}_S(h) = 0\}$ and thus,

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_D(h_S) > \epsilon] \;\leq\; \mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\exists h \in H_B : \mathrm{err}_S(h) = 0] \,. \tag{4}$$

Next, we upper bound the right-hand side of the above using the *union bound*, whose proof is trivial.

Lemma 1 (Union bound) *For any two sets A, B we have*

$$\mathbb{P}[A \cup B] \leq \mathbb{P}[A] + \mathbb{P}[B] \,.$$

Since the set $\{S : \exists h \in \mathcal{H}_B, \mathrm{err}_S(h) = 0\}$ can be written as $\cup_{h\in\mathcal{H}_B}\{S : \mathrm{err}_S(h) = 0\}$, we can apply the union bound on the right-hand side of Eq. (4) to get that

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_D(h_S) > \epsilon] \;\leq\; \sum_{h\in\mathcal{H}_B} \mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_S(h) = 0] \,. \tag{5}$$

Next, let us bound each summand of the right-hand side of the above. Fix some bad hypothesis $h \in \mathcal{H}_B$. For each individual element of the training set we have,

$$\mathop{\mathbb{P}}_{(\mathbf{x}_i,y_i)\sim\mathcal{D}}[h(\mathbf{x}_i) = y_i] = 1 - \mathrm{err}_{\mathcal{D}}(h) \leq 1 - \epsilon \,.$$

Since the examples in the training set are sampled i.i.d. we get that for all $h \in \mathcal{H}_B$

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_S(h) = 0] = \mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\forall i, h(\mathbf{x}_i) = y_i] = \prod_{i=1}^{m} \mathop{\mathbb{P}}_{(\mathbf{x}_i,y_i)\sim\mathcal{D}}[h(\mathbf{x}_i) = y_i] \leq (1-\epsilon)^m \,. \tag{6}$$

Combining the above with Eq. (5) and using the inequality $1 - \epsilon \leq e^{-\epsilon}$ we conclude that

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_D(h_S) > \epsilon] \;\leq\; |\mathcal{H}_B|\,(1-\epsilon)^m \;\leq\; |\mathcal{H}|\,e^{-\epsilon\,m} \,.$$

Corollary 1 *Let $\mathcal{H}$ be a finite hypothesis class. Let $\delta \in (0,1)$ and $\epsilon > 0$ and let m be an integer that satisfies*

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon} \,.$$

Then, for any distribution $\mathcal{D}$, for which the realizable assumption holds, with probability of at least $1-\delta$ over the choice of an i.i.d. sample S of size m we have

$$\mathrm{err}_{\mathcal{D}}(h_S) \leq \epsilon \,.$$

A graphical illustration which explains how we used the union bound is given in Figure 2.

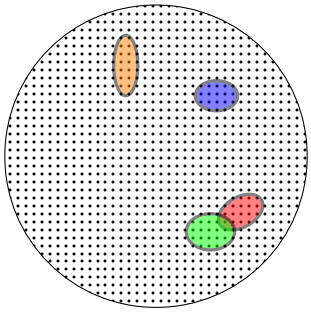


Figure 2: Each point in the large circle represents a possible training sets of m examples. Each colored area represents 'bad' training sets for some bad predictor $h \in \mathcal{H}_B$, that is $\{S : \text{err}_{\mathcal{D}}(h) > \epsilon \wedge \text{err}_S(h) = 0\}$. The ERM can potentially overfit whenever it gets a training set S which is bad for some $h \in \mathcal{H}_B$. Eq. (6) guarantees that for each individual $h \in \mathcal{H}_B$, at most $(1-\epsilon)^m$-fraction of the training sets will be bad. Using the union bound we bound the fraction of training sets which are bad for some $h \in \mathcal{H}_B$. This can be used to bound the set of predictors which might lead the ERM rule to overfit.

6 PAC learning

In the previous section we showed that if we restrict the search space to a finite hypothesis class then the ERM rule will probably find a classifier whose error is approximately the error of the best classifier in the hypothesis class. This is called *Probably Approximately Correct* (PAC) learning.

Definition 1 (PAC learnability) *A hypothesis class $\mathcal{H}$ is PAC learnable if for any $\epsilon > 0, \delta \in (0,1)$ there exists $m = \text{poly}(1/\epsilon, 1/\delta)$ and a learning algorithm such that for any distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, which satisfies the realizability assumption, when running the learning algorithm on m i.i.d. examples it returns $h \in \mathcal{H}$ such that with probability of at least $1-\delta$, $\text{err}_{\mathcal{D}}(h) \leq \epsilon$.*

Few remarks:

- The definition of Probably Approximately Correct learnability contains two approximation parameters. The parameter ϵ is called the accuracy parameter (corresponds to "approximately correct") and the parameter δ is called the confidence parameter (corresponds to "probably"). The definition requires that the number of examples that is required for achieving accuracy ϵ with confidence $1-\delta$ is polynomial in $1/\epsilon$ and in $1/\delta$. This is similar to the definition of Fully Polynomial Approximation Schemes (FPTAS) in the theory of computation.

- We require the learning algorithm to succeed for any distribution $\mathcal{D}$ as long as the realizability assumption holds. Therefore, in this case the prior knowledge we have on the world is encoded in the choice of the hypothesis space and in the realizability assumption with respect to this hypothesis space. Later, we will describe the agnostic PAC model in which we relax the realizability assumption as well. We will also describe learning frameworks in which the guarantees do not hold for any distribution but instead only hold for a specific parametric family of distributions, which implies a much stronger prior belief on the world.

- In the previous section we showed that a finite hypothesis class is PAC learnable. But, there are infinite classes that are learnable as well. Later on we will show that what determines the PAC learnability of a class is not its finiteness but rather a combinatorial measure called VC dimension.

(67577) Introduction to Machine Learning October 26, 2009

Lecture 3 – The Bias-Complexity Tradeoff

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the previous lecture we showed that a finite hypothesis class is learnable in the PAC model. The PAC model assumes that there exists a perfect hypothesis (in terms of generalization error) in the hypothesis class. But, what if this assumption does not hold? In this lecture we present the *agnostic* PAC model in which we do not assume the existence of a perfect hypothesis. We analyze the learnability of a finite hypothesis class in the agnostic PAC model and by doing so demonstrate the *bias-complexity tradeoff*, a fundamental concept in machine learning which analyzes the tension between overfitting and underfitting.

7 Agnostic PAC learning

In the previous lecture we defined the PAC learning model. We now define the agnostic PAC learning model, in which the realizability assumption is not required. Clearly, we cannot hope that the learning algorithm will find a hypothesis whose error is smaller than the minimal possible error, $\min_{h\in\mathcal{H}} \mathrm{err}_{\mathcal{D}}(h)$. Instead, we require that the learning algorithm will find a predictor whose error is not much larger than the best possible error of a predictor in the hypothesis class.

Definition 2 (agnostic PAC learnability) *A hypothesis class $\mathcal{H}$ is agnostic PAC learnable if for any $\epsilon > 0, \delta \in (0,1)$ there exists $m = \mathrm{poly}(1/\epsilon, 1/\delta)$ and a learning algorithm such that for any distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, when running the learning algorithm on m i.i.d. training examples it returns $h \in \mathcal{H}$ such that with probability of at least $1-\delta$ (over the choice of the m training examples),*

$$\mathrm{err}_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} \mathrm{err}_{\mathcal{D}}(h') + \epsilon \,.$$

In this lecture we will prove that a finite hypothesis class is agnostic PAC learnable. To do so, we will show that there exists $m = \mathrm{poly}(1/\epsilon, 1/\delta)$ such that with probability of at least $1-\delta$, for *all* $h \in \mathcal{H}$ we have that $|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| \leq \epsilon/2$. This type of property is called *uniform convergence*. The following simple lemma tells us that whenever uniform convergence holds the ERM learning rule is guaranteed to return a good hypothesis.

Lemma 2 *Let $\mathcal{D}$ be a distribution, S be a training set, and $\mathcal{H}$ be a hypothesis class such that uniform convergence holds, namely*

$$\forall h \in \mathcal{H},\ |\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| \ \leq\ \epsilon/2 \,.$$

Let $h_S \in \arg\min_{h\in\mathcal{H}} \mathrm{err}_S(h)$ be an ERM hypothesis. Then,

$$\mathrm{err}_{\mathcal{D}}(h_S) \ \leq\ \min_{h\in\mathcal{H}} \mathrm{err}_{\mathcal{D}}(h) + \epsilon \,.$$

Proof Since h_S is an ERM hypothesis, we have that for all $h \in \mathcal{H}$,

$$\mathrm{err}_{\mathcal{D}}(h_S) \leq \mathrm{err}_S(h_S) + \tfrac{\epsilon}{2} \leq \mathrm{err}_S(h) + \tfrac{\epsilon}{2} \leq \mathrm{err}_{\mathcal{D}}(h) + \tfrac{\epsilon}{2} + \tfrac{\epsilon}{2} = \mathrm{err}_{\mathcal{D}}(h) + \epsilon \,.$$

The first and third inequalities are because of the uniform convergence and the second inequality is because h_S is an ERM predictor. The lemma follows because the inequality holds for all $h \in \mathcal{H}$. ■

The above lemma tells us that uniform convergence is a sufficient condition for agnostic PAC learnability. Therefore, to prove that a finite hypothesis class is agnostic PAC learnable it suffices to establish that uniform

convergence holds for a finite hypothesis class. To show that uniform convergence holds we follow a two step argument. First, we will argue that $|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)|$ is likely to be small for any fixed hypothesis, which is chosen in advance prior to the sampling of the training set. Since $\mathrm{err}_{\mathcal{D}}(h)$ is the expected value of $\mathrm{err}_S(h)$, the quantity $|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)|$ is the deviation of $\mathrm{err}_S(h)$ from its expectation. When this deviation is likely to be small we say that the measure of $\mathrm{err}_S(h)$ is concentrated. Second, we will apply the union bound (similar to the derivation in the previous lecture) to show that with a slightly larger training set, the random variable $\mathrm{err}_S(h)$ is concentrated around its mean uniformly for all hypotheses in $\mathcal{H}$.

8 Measure Concentration

Let $Z_1, \ldots, Z_m$ be an i.i.d. sequence of random variables and let μ be their mean. In the context of this chapter, one can think on Z_i as being the random variable $|h(\mathbf{x}_i) - y_i|$. The law of large numbers states that when m tends to infinity, the empirical average, $\frac{1}{m}\sum_{i=1}^m Z_i$, converges to the expected value μ, with probability 1. Measure concentration inequalities quantify the deviation of the empirical average from the expectation when m is finite.

We start with an inequality which is called Markov's inequality. Let Z be a non-negative random variable. The expectation of Z can be written as follows (see Exercise ?):

$$\mathbb{E}[Z] = \int_{x=0}^{\infty} \mathbb{P}[Z \geq x] \,. \tag{7}$$

Since $\mathbb{P}[Z \geq x]$ is monotonically non-increasing we obtain

$$\forall a \geq 0, \quad \mathbb{E}[Z] \geq \int_{x=0}^{a} \mathbb{P}[Z \geq x] \geq \int_{x=0}^{a} \mathbb{P}[Z \geq a] = a\, \mathbb{P}[Z \geq a] \,. \tag{8}$$

Rearranging the above yields Markov's inequality:

$$\forall a \geq 0, \quad \mathbb{P}[Z \geq a] \;\leq\; \frac{\mathbb{E}[Z]}{a} \,. \tag{9}$$

Applying Markov's inequality on the random variable $(Z - \mathbb{E}[Z])^2$ we obtain Chebyshev's inequality:

$$\mathbb{P}[|Z - \mathbb{E}[Z]| \geq a] = \mathbb{P}[(Z - \mathbb{E}[Z])^2 \geq a^2] \leq \frac{\mathrm{Var}[Z]}{a^2} \,, \tag{10}$$

where $\mathrm{Var}[Z] = \mathbb{E}[(Z - \mathbb{E}[Z])^2]$ is the variance of Z.

Next, we apply Chebyshev's inequality on the random variable $\frac{1}{m}\sum_{i=1}^m Z_i$. Since $Z_1, \ldots, Z_m$ are i.i.d. we know that (see Exercise ?)

$$\mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^m Z_i\right] \;=\; \frac{\mathrm{Var}[Z_1]}{m} \,.$$

From the above, we obtain the following:

Lemma 3 *Let $Z_1, \ldots, Z_m$ be a sequence of i.i.d. random variables and assume that $\mathbb{E}[Z_1] = \mu$ and $\mathrm{Var}[Z_1] \leq 1$. Then, for any $\delta \in (0,1)$, with probability at least $1 - \delta$ we have*

$$\left|\frac{1}{m}\sum_{i=1}^m Z_i - \mu\right| \leq \sqrt{\frac{1}{\delta\, m}} \,.$$

Proof Applying Chebyshev's inequality we obtain that for all $a > 0$

$$\mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^m Z_i - \mu\right| > a\right] \;\leq \frac{\mathrm{Var}[Z_1]}{m\, a^2} \leq \frac{1}{m\, a^2} \,.$$

The proof follows by denoting the right-hand side δ and solving for a. ■

Lets apply the above lemma for the random variables $Z_i = |h(X_i) - Y_i|$. Since $Z_i \in [0,1]$ we clearly have that $\mathrm{Var}[Z_i] \leq 1$. Choose $\delta = 0.1$, we obtain that for 90% of the training sets we will have

$$|\mathrm{err}_S(h) - \mathrm{err}_{\mathcal{D}}(h)| \leq \sqrt{\frac{10}{m}} \,.$$

In other words we can use $\mathrm{err}_S(h)$ to estimate $\mathrm{err}_{\mathcal{D}}(h)$ and the estimate will be *Probably Approximately Correct*, as long as m is sufficiently large.

We can also ask how many examples are needed in order to obtain accuracy ϵ with confidence $1 - \delta$. Based on Lemma 3 we obtain that if

$$m \;\geq\; \frac{1}{\delta\,\epsilon^2}$$

then with probability of at least $1 - \delta$ the deviation between the empirical average and the mean is at most ϵ.

The deviation between the empirical average and the mean given above depends polynomially on the confidence parameter δ. It is possible to obtain a significantly milder dependence, and this will turn out to be crucial in later sections. We conclude this section with another measure concentration inequality due to Hoeffding in which the dependence on $1/\delta$ is only logarithmic.

Lemma 4 (Hoeffding's inequality) *Let $Z_1, \ldots, Z_m$ be a sequence of i.i.d. random variables and assume that $\mathbb{E}[Z_1] = \mu$ and $\mathbb{P}[a \leq Z_1 \leq b] = 1$. Then, for any $\epsilon > 0$*

$$\mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^{m} Z_i - \mu\right| > \epsilon\right] \;\leq\; 2\,\exp\left(-2\,m\,\epsilon^2/(b-a)^2\right) \,.$$

The proof is left as an exercise.

The advantage of Lemma 4 over Lemma 3 stems from the fact that in the former our confidence improves exponentially fast as the number of examples increases.

Applying Hoeffding's inequality (Lemma 4) with the random variables $Z_i = |h(\mathbf{x}_i) - y_i|$ we obtain:

Corollary 2 *Let $h : \mathcal{X} \to \{0,1\}$ be an arbitrary predictor and let $\epsilon > 0$ be an accuracy parameter. Then, for any distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ we have*

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}\left[|\mathrm{err}_S(h) - \mathrm{err}_{\mathcal{D}}(h)| > \epsilon\right] \;\leq\; 2\,\exp\left(-2\,m\,\epsilon^2\right) \,.$$

9 Finite classes are agnostic PAC learnable

We are now ready to prove that a finite hypothesis class is agnostic PAC learnable.

Theorem 1 *Let $\mathcal{H}$ be a finite hypothesis class. Let $\delta \in (0,1)$ and $\epsilon > 0$ and let m be an integer that satisfies*

$$m \geq \frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2} \,.$$

Then, for any distribution $\mathcal{D}$, with probability of at least $1 - \delta$ over the choice of an i.i.d. training set S of size m we have

$$\mathrm{err}_{\mathcal{D}}(h_S) \;\leq\; \min_{h\in\mathcal{H}} \mathrm{err}_{\mathcal{D}}(h) + \epsilon \,.$$

Proof From Lemma 2 we know that it suffices to prove that with probability of at least $1 - \delta$, for all $h \in \mathcal{H}$ we have $|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| \leq \epsilon/2$. In other words,

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}\left[\exists h \in \mathcal{H},\; |\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| > \tfrac{\epsilon}{2}\right] \;\leq\; \delta \,.$$

Using the union bound we have

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}\left[\exists h\in\mathcal{H},\ |\mathrm{err}_{\mathcal{D}}(h)-\mathrm{err}_S(h)|>\tfrac{\epsilon}{2}\right]\ \leq\ \sum_{h\in\mathcal{H}}\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}\left[|\mathrm{err}_{\mathcal{D}}(h)-\mathrm{err}_S(h)|>\tfrac{\epsilon}{2}\right]\ .$$

Using Corollary 2 we know that each summand on the left-hand side of the above is at most $2\exp(-m\,\epsilon^2/2)$. Therefore,

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}\left[\exists h\in\mathcal{H},\ |\mathrm{err}_{\mathcal{D}}(h)-\mathrm{err}_S(h)|>\tfrac{\epsilon}{2}\right]\ \leq\ 2\,|\mathcal{H}|\,\exp(-m\,\epsilon^2/2)\ .$$

Combining the above with the requirement on m we conclude our proof. ■

It is interesting to compare Theorem 1 to Corollary 1. Clearly, Theorem 1 is more general. On the other hand, whenever the realizable assumption holds, both Theorem 1 to Corollary 1 guarantee that $\mathrm{err}_{\mathcal{D}}(h_S)\leq\epsilon$ but the number of training examples required in Theorem 1 is larger – it grows like $1/\epsilon^2$ while in Corollary 1 it grows like $1/\epsilon$. We note in passing that it is possible to interpolate between the two results but this is out of our scope.

10 Error decomposition

Learning is about replacing expert knowledge with data. In this lecture we have shown how data can be used to estimate the error of a hypothesis using a set of examples. In the previous lecture we also saw that without being careful, the data can mislead us and in particular we might suffer from overfitting. To overcome this problem, we restricted the search space to a particular hypothesis class $\mathcal{H}$. How should we choose $\mathcal{H}$?

To answer this question we decompose the error of the ERM predictor into:

- **The approximation error**—the minimum generalization error achievable by a predictor in the hypothesis class. The approximation error does not depend on the sample size, and is determined by the hypothesis class allowed. A larger hypothesis class can decrease the approximation error.
- **The estimation error**—the difference between the approximation error and the error achieved by the ERM predictor. The estimation error is a result of the training error being only an estimate of the generalization error, and so the predictor minimizing the training error being only an estimate of the predictor minimizing the generalization error. The quality of this estimation depends on the training set size and the size, or complexity, of the hypothesis class.

Formally, let h_S be an ERM predictor, then

$$\mathrm{err}_{\mathcal{D}}(h_S)=\epsilon_{\mathrm{app}}+\epsilon_{\mathrm{est}}\quad\text{where :}\ \ \epsilon_{\mathrm{app}}=\min_{h\in\mathcal{H}}\mathrm{err}_{\mathcal{D}}(h),\ \epsilon_{\mathrm{est}}=\mathrm{err}_{\mathcal{D}}(h_S)-\epsilon_{\mathrm{app}}\ . \tag{11}$$

The first term, the approximation error, measures how much error we have because we restrict ourselves to the hypothesis class $\mathcal{H}$. This is the *inductive bias* we add. It does not depend on the size of the training set. The second term, the estimation error, measures how much extra error we have because we learn a classifier based on a finite training set S instead of knowing the distribution $\mathcal{D}$. As we have shown, for a finite hypothesis class, ϵ_{est} increases with $|\mathcal{H}|$ and decreases with m. We can think on the size of $\mathcal{H}$ as how complex $\mathcal{H}$ is. Later in this course we will define other complexity measures of hypothesis classes.

Since our goal is to minimize the total generalization error, we face a tradeoff. On one hand, choosing $\mathcal{H}$ to be a very rich class decreases the approximation error but at the same time increases the estimation error, as a rich $\mathcal{H}$ might lead to *overfitting*. On the other hand, choosing $\mathcal{H}$ to be a very small set reduces the estimation error but might increase the approximation error, or in other words, might lead to *underfitting*. Of course, a great choice for $\mathcal{H}$ is the class that contains only one classifier – the Bayes optimal classifier (this is the classifier whose generalization error is minimal – see exercise for details). But, the Bayes optimal classifier

depends on the underlying distribution $\mathcal{D}$, which we do not know, and in fact learning was unnecessary had we knew $\mathcal{D}$.

Learning theory studies how rich we can make $\mathcal{H}$ while still maintaining reasonable estimation error. In many cases, empirical research focuses on designing good hypothesis classes for a certain domain. The idea is that although we are not experts and do not know how to construct the optimal classifier, we still have some prior knowledge on the specific problem at hand which enables us to design hypothesis classes for which both the approximation error and the estimation error are not too large. Getting back to our Papayas example, we do not know how exactly the color and smoothness of a Papaya predicts its taste, but we do know that Papaya is a fruit and based on previous experience with other fruits we conjecture that a rectangle may be a good predictor. We use the term bias-complexity tradeoff to denote the tradeoff between approximation error and estimation error.

11 Concluding Remarks

To summarize, in this lecture we introduced the agnostic PAC learning model and showed that finite hypothesis classes are learnable in this model. We made several assumptions, some of them are arbitrary, and we now briefly mention them.

- Why modeling the environment as a distribution and why interaction with the environment is by sampling — this is a convenient model which is adequate to some situations. We will meet other learning models in which there is no distributional assumption and the interaction with the environment is different.
- Why binary classifiers — the short answer is simplicity. In binary classification the definition of error is very clear. We either predict the label correctly or not. Many of the results we will discuss in the course hold for other type of predictors as well.
- Why considering only ERM — the ERM learning rule is very natural. Nevertheless, in some situations we will discuss other learning rules. We will show that in the agnostic PAC model, if a class if learnable then it is learnable with the ERM rule.
- Why distribution free learning — Our goal was to make as few assumptions as possible. The agnostic PAC model indeed make only few assumptions. There are popular alternative models of learning in which we make rather strong distributional assumptions. For example, generative models, which we discuss later in the course.
- Why restricting hypothesis space — we will show that there are other ways to restrict the search space and avoid overfitting.
- Why finite hypothesis classes — we will show learnability results with infinite classes in the next lectures
- We ignore computational issues — for simplicity. In some cases, computational issues make the ERM learning rule infeasible.

12 Exercise

1. **The Bayes error:** Among all classifiers, the Bayes optimal classifier is the one which has the lowest generalization error and the Bayes error is its generalization error. The error of the Bayes optimal classifier is due to the intrinsic non-determinism of our world as reflected in $\mathcal{D}$. In particular, if y is deterministically set given $\mathbf{x}$ then $\mathrm{err}_{\mathcal{D}}(h_{\mathrm{Bayes}}) = 0$. The Bayes optimal classifier depends on the

distribution $\mathcal{D}$, which is unknown to us. Had we known to construct the Bayes optimal classifier, we wouldn't need to learn anything. Show that the Bayes optimal classifier is:

$$h_{\text{Bayes}}(\mathbf{x}) \;=\; \operatorname*{argmax}_{y \in \mathcal{Y}} \mathbb{P}[y|\mathbf{x}] \;. \tag{12}$$

2. Prove Eq. (7)

3. Calculate the variance of sums of independent random variables

4. Prove Hoeffding's inequality

(67577) Introduction to Machine Learning November 2, 2009

Lecture 4 – Minimum Description Length

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the previous lecture we saw that finite hypothesis classes are learnable in the agnostic PAC model. In the case of a finite set of hypotheses we established a uniform convergence results — the training error is close to the generalization error for *all* hypotheses in the finite class. In this lecture we will see that it is possible to learn infinite hypothesis classes even though uniform convergence does not hold. Instead of requiring the algorithm to choose a hypothesis from a finite hypothesis class, we will define weights of hypotheses. The idea is to divide the uncertainty of the sampling error *unevenly* among the different hypotheses, such that the estimation error will be smaller for hypotheses with larger weights. The learner will need to balance between choosing hypotheses with larger weights (thus having a smaller estimation error) to choosing hypotheses that fit well the training set.

13 Generalization bounds for weighted hypothesis classes

Let $\mathcal{H}$ be any hypothesis class, and let $w : \mathcal{H} \to [0, 1]$ be a function such that $\sum_{h\in\mathcal{H}} w(h) \leq 1$. One can think of w as assigning 'weights' to the hypotheses in $\mathcal{H}$. Of course, if $\mathcal{H}$ is finite, then w can treat all members of $\mathcal{H}$ equally, setting $w(h) = 1/|\mathcal{H}|$ for every $h \in \mathcal{H}$. When $\mathcal{H}$ is infinite, such a uniform weighting is not possible but many other weighting schemes may work. For example, when $\mathcal{H}$ is countable, one can pick any one-to-one function, f, from the set of natural numbers, $\mathbb{N}$, to $\mathcal{H}$ and then define, for any h in the range of f, $w(h) = \left(1/f^{-1}(h)\right)^2$. We will see how such a weighting of the members of $\mathcal{H}$ can replace the assumption that $\mathcal{H}$ is finite and allow some form of learnability for that class.

Theorem 2 *Let $\mathcal{H}$ be a hypothesis class, and let $w : \mathcal{H} \to [0, 1]$ be any function satisfying $\sum_{h\in H} w(h) \leq 1$. Then, for every sample size, m, every confidence parameter, $\delta > 0$, and every distribution $\mathcal{D}$ over $\mathcal{X} \times \{0, 1\}$*

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}\left[\exists h \in \mathcal{H} \text{ s.t. } |\mathrm{err}_S(h) - \mathrm{err}_\mathcal{D}(h)| \geq \sqrt{\frac{\ln(1/w(h)) + \ln(2/\delta)}{2m}}\right] \leq \delta$$

.

Proof For all $h \in \mathcal{H}$, define $\epsilon_h = \sqrt{\frac{\ln(1/w(h))+\ln(2/\delta)}{2m}}$. Using the union bound we have

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\exists h \in \mathcal{H} \text{ s.t. } |\mathrm{err}_S(h) - \mathrm{err}_\mathcal{D}(h)| \geq \epsilon_h] \leq \sum_{h\in\mathcal{H}} \mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[|\mathrm{err}_S(h) - \mathrm{err}_\mathcal{D}(h)| \geq \epsilon_h] \ .$$

We can bound each summand using Hoeffding's inequality (Corollary 2):

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[|\mathrm{err}_S(h) - \mathrm{err}_\mathcal{D}(h)| \geq \epsilon_h] \ \leq \ 2\exp(-2m\epsilon_h^2) \ .$$

Combining the above two inequalities and plugging in the definition of ϵ_h we obtain that

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\exists h \in \mathcal{H} \text{ s.t. } |\mathrm{err}_S(h) - \mathrm{err}_\mathcal{D}(h)| \geq \epsilon_h] \leq \delta \sum_{h\in\mathcal{H}} w(h) \ .$$

The theorem now follows from the assumption that $\sum_{h\in\mathcal{H}} w(h) \leq 1$. ■

Note that the generalization bound we established for a finite hypothesis class (Theorem 1) can be viewed as a special case of the above theorem by choosing the uniform weighting: $w(h) = 1/|\mathcal{H}|$ for all $h \in \mathcal{H}$.

13.1 Bound minimization

A direct corollary of Theorem 2 is the following:

Corollary 3 *Under the conditions of Theorem 2, for every sample size, m, every confidence parameter, $\delta > 0$, and every distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$, with probability greater than $1-\delta$ over a training set of size m generated i.i.d. by $\mathcal{D}$,*

$$\forall h \in \mathcal{H},\ \ \mathrm{err}_{\mathcal{D}}(h) \leq \mathrm{err}_S(h) + \sqrt{\frac{\ln(1/w(h)) + \ln(2/\delta)}{2m}}$$

.

Such an error bound suggests a natural learning paradigm: Given a hypothesis class, $\mathcal{H}$, and a weighting function, w, upon receiving a training set, S, search for $h \in \mathcal{H}$ that minimizes the bound on the true error, namely return a hypothesis in

$$\operatorname*{argmin}_{h \in \mathcal{H}} \mathrm{err}_S(h) + \sqrt{\frac{\ln(1/w(h)) + \ln(2/\delta)}{2m}}\ .$$

That is, unlike the ERM paradigm discussed in previous lectures, we no longer just care about the empirical error, but also willing to trade off some of that error (or, if you wish, some bias) for the sake of a better estimation-error term (or, complexity).

The above results can be applied to several natural weighting functions. We shall discuss two such examples; description length and prior probability over hypotheses.

13.2 Description Length and Occam's Razor

Having a hypothesis class, one can wonder about how do we describe, or represent, each function in the class. We naturally fix some description language. This can be English, or a programming language, or some set of mathematical formulas. Any of these languages consists of finite strings of symbols (or characters) drawn from some fixed alphabet. It is worthwhile to formalize these notions.

Let $\mathcal{H}$ be some set (possibly infinite), the members of which we wish to describe. Fix some finite set Σ of symbols (or "characters"), which we call the alpha-bet. For concreteness, we let $\Sigma = \{0,1\}$. A string is a finite sequence of symbols from Σ, for example $\boldsymbol{\sigma} = (0,1,1,1,0)$ is a string of length 5. We denote by $|\boldsymbol{\sigma}|$ the length of a string. The set of all finite length strings is denoted Σ^*. A description language for $\mathcal{H}$ is a function $d : \mathcal{H} \to \Sigma^*$, mapping each member, h of $\mathcal{H}$, to a string $d(h)$. We call $d(h)$ 'the description of h'. We denote by $|h|$ the length of the string $d(h)$.

We shall require that description languages are *prefix-free*. That is, for every distinct h, h', none of $d(h), d(h')$ is a prefix of the other. That is, we do not allow that the string $d(h)$ is exactly the first $|h|$ symbols of the string $d(h')$ and the other way around. Prefix-free collections of strings have the following nice combinatorial property:

Lemma 5 (Kraft inequality) *If $\mathcal{S} \subseteq \{0,1\}^*$ is a prefix-free set of strings, then*

$$\sum_{\boldsymbol{\sigma} \in \mathcal{S}} \frac{1}{2^{|\boldsymbol{\sigma}|}} \leq 1\ .$$

Proof Define a probability distribution over the members of $\mathcal{S}$ as follows: repeatedly toss an unbiased coin, with faces labeled 0 and 1, until the sequence of outcomes is a member of $\mathcal{S}$, at that point, stop. For each $\boldsymbol{\sigma} \in \mathcal{S}$, let $P(\boldsymbol{\sigma})$ be the probability that this process generates the string $\boldsymbol{\sigma}$. Note that since $\mathcal{S}$ is prefix-free, for every $\boldsymbol{\sigma} \in \mathcal{S}$, if the coin toss outcomes follow the bits of $\boldsymbol{\sigma}$ than we will stop only once the sequence of outcomes equals $\boldsymbol{\sigma}$. We therefore get that, for every $\boldsymbol{\sigma} \in \mathcal{S}$, $P(\boldsymbol{\sigma}) = \frac{1}{2^{|\boldsymbol{\sigma}|}}$. Since probabilities add up to at most one, our proof is concluded. ■

In light of Kraft's inequality, any prefix-free description language of a hypothesis class, $\mathcal{H}$, gives rise to a weighting function w over that hypothesis class — we will simply set $w(h) = \frac{1}{2^{|h|}}$. This observation immediately yields the following:

Theorem 3 *Let $\mathcal{H}$ be a hypothesis class and let $d : \mathcal{H} \to \{0,1\}^*$ be a prefix-free description language for $\mathcal{H}$. Then, for every sample size, m, every confidence parameter, $\delta > 0$, and every probability distribution, $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$, with probability greater than $1-\delta$ over an m-size training set generated i.i.d. by $\mathcal{D}$,*

$$\forall h \in \mathcal{H}, \;\; \mathrm{err}_{\mathcal{D}}(h) \leq \mathrm{err}_S(h) + \sqrt{\frac{|h| + \ln(2/\delta)}{2m}} \; .$$

Proof Just substitute $w(h) = 1/2^{|h|}$ in Corollary 3 above. ■

As was the case with Corollary 3, this result suggests a learning paradigm for $\mathcal{H}$ — given a training set, S, search for a hypothesis $h \in \mathcal{H}$ that minimizes the error bound, $\mathrm{err}_S(h) + \sqrt{\frac{|h|+\ln(2/\delta)}{2m}}$. In particular, it suggests trading off training error for short description length.

13.2.1 Occam's razor

Theorem 3 tell us that if we have two hypotheses that have the same training error, for the one that has shorter description our bound on its generalization error is lower. Thus, this result can be viewed as having a philosophical message —

> *A short explanation (that is, a hypothesis that has a short length) tends to be more valid than a long explanation.*

This is a well known principle, called Occam's razor, after William of Ockham, a 14th-century English logician, who is believed to have been the first to phrase it explicitly. Here, we seem to provide a rigorous justification to this principle. The inequality of Theorem 3 shows that the more complex a hypothesis h is (in the sense of having a long description), the larger the sample size it has to fit needed to guarantee that it really has a small generalization error, $\mathrm{err}_{\mathcal{D}}(h)$. The second term in the inequality requires m, the sample size, to be at least in the order of $|h|$, the length of the description of h. Note that, as opposed to the context in which the Occam razor principle is usually invoked in science, here we consider an arbitrary abstract description language, rather than a natural language.

13.2.2 The role of the description language

At a second thought, there may be something bothering about our Occam razor result — the simplicity of a hypothesis is measured by the length of the string assigned to it by the description language. However, this language is quite arbitrary. Assume that we have two hypotheses such that $|h'|$ is much smaller than $|h|$. By the result above, if both have the same error on a given training set, S, then the true error of h may be much higher than the true error of h', so one should prefer h' over h. However, we could have chosen a different description language, say one that assigns a string of length 3 to h and a string of length 100000 to h'. Suddenly it looks like one should prefer h over h'. But these are the same h and h' for which we argued two sentences ago that h' should be preferable. Where is the catch here?

Indeed, there is no inherent generalizability difference between hypotheses. The crucial aspect here is the dependency order between the initial choice of language (or, preference over hypotheses) and the training set. As we know from the basic Hoeffding's bound (Corollary 2), if we commit to any hypothesis *before* seeing the data, then we are guaranteed a rather small generalization error term $\mathrm{err}_{\mathcal{D}}(h) \leq \mathrm{err}_S(h) + \sqrt{\frac{\ln(2/\delta)}{2m}}$. Choosing a description language (or, equivalently, some weighting of hypotheses) is a weak form of committing to a hypothesis. Rather than committing to a single hypothesis, we spread out our commitment

among many. As long as it is done independently of the training sample, our generalization bound holds. Just as the choice of a single hypothesis to be evaluated by a sample can be arbitrary, so is the choice of description language for the description length based bound.

13.3 Incorporating a prior probability over hypotheses

Yet another intuitive aspect of Theorem 2 is gained by viewing the weighting function, w, as a probability distribution over the set of hypotheses, $\mathcal{H}$. This is possible since all weights are non-negative and their sum is at most 1. Under such an interpretation, what the learner starts with is some *prior* probability, evaluating the likelihood of each hypothesis to be the best predictor. We call it "prior" since, as discussed above, that probability (or weighting) must be decided before the learner accesses the training sample. Under this interpretation, Corollary 3 reads as follows:

Theorem 4 *Let $\mathcal{H}$ be a hypothesis class and let P be any probability distribution over $\mathcal{H}$ (the "prior"). For every sample size, m, every probability distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ and every confidence parameter, $\delta > 0$, for every $h \in \mathcal{H}$, with probability greater than $1-\delta$ over an m-size training set generated i.i.d. by $\mathcal{D}$,*

$$\mathrm{err}_{\mathcal{D}}(h) \leq \mathrm{err}_{S}(h) + \sqrt{\frac{\ln(1/P(h)) + \ln(2/\delta)}{2m}} \ .$$

In particular, this result suggests a learning paradigm that searches for a hypothesis $h \in \mathcal{H}$ that balances a tradeoff between having low sample error and having high prior probability. Or, from a slightly different angle, the supporting evidence, in the form of a training sample, that one needs to validate an a priory unlikely hypothesis is required to be larger than what we need to validate a likely hypothesis. This is very much in line with our daily experience.

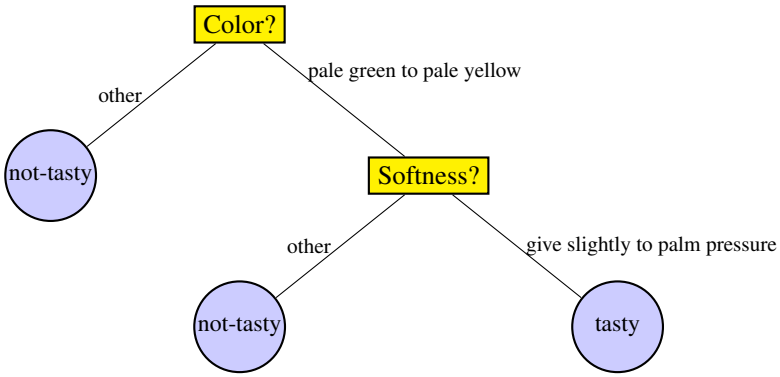
(67577) Introduction to Machine Learning November 3, 2009

Lecture 5 – Decision Trees

Lecturer: Ohad Shamir *Scribe: Ohad Shamir*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

A decision tree is a classifier, $h : \mathcal{X} \to \mathcal{Y}$, that predicts the label associated with an instance $\mathbf{x}$ by traveling from a root node of a tree to a leaf. At each node on the root-to-leaf path, the successor child is chosen based on a splitting of the input space. Usually, the splitting is based on one of the attributes of $\mathbf{x}$ or on a predefined set of splitting rules. A leaf contains a specific label. An example of a decision tree for the papayas example is given below:



To check if a given papaya is tasty or not, the decision tree above first examines the color of the Papaya. If this color is not in the range pale green to pale yellow, then the tree immediately predicts that the papaya is not tasty without additional tests. Otherwise, the tree turns to examine the softness of the papaya. If the softness level of the papaya is such that it gives slightly to palm pressure, the decision tree predicts that the papaya is tasty. Otherwise, the prediction is "not-tasty". The above example underscores one of the main advantages of decision trees — the resulting classifier is simple to understand and easy to interpret.

We can think on a decision tree as a splitting of the instance space, $\mathcal{X}$, into cells, where each leaf of the tree corresponds to one cell. Clearly, if we allow decision trees of arbitrary size, for any training set, S, we can find in general a decision tree that attains a zero training error on S, simply by splitting the instance space into small enough regions such that each region contains a single training example.[1] Such an approach can easily lead to overfitting. To avoid overfitting, we can rely on the Occam's razor principle described in the previous section, and aim at learning a decision tree that on one hand fits the data well while on the other hand is not too large.

For simplicity, we will assume that $\mathcal{X} = \{0,1\}^d$. In other words, each instance is a vector of d bits. We will also assume that our decision tree is always a binary tree, with the internal nodes of the form '$x_i = 0$?' for some $i = \{1, \ldots, d\}$. For instance, we can model the Papaya decision tree above by assuming that a Papaya is parameterized by a two-bit vector $\mathbf{x} = (x_1, x_2)$, where the bit x_1 represents whether the color is 'pale green to pale yellow' or not, and the bit x_2 represents whether the softness is 'give slightly to palm pressure' or not. With this representation the node 'Color?' can be replaced with '$x_1 = 0$?', and the node 'Softness?' can be replaced with '$x_2 = 0$?'. While this is a big simplification, the algorithms and analysis we provide below can be extended to more general cases.

With these simplifying assumptions, the hypothesis class becomes finite, but is still very large. In particular, it is possible to show that any classifier from $\{0,1\}^d$ to $\{0,1\}$ can be represented by a decision tree.

[1] We might have two identical examples in the training set that have different labels, but if the instance space, $\mathcal{X}$, is large enough and the distribution is not focused on few elements of $\mathcal{X}$, such an event will occur with a very low probability.

Therefore, the effective size of the hypothesis class is equal to the number of functions from $\{0,1\}^d$ to $\{0,1\}$, namely 2^{2^d}. If we invoke the standard bound for finite hypothesis classes, we get that we will need at least

$$m \geq \frac{2\log(2|H|/\delta)}{\epsilon^2} = \frac{2(2^d + \log(2/\delta))}{\epsilon^2}$$

examples in order to agnostically PAC learn the hypothesis class. Unless d is very small, this is a huge number of examples. Luckily, the Occam's razor principle tells us that we might need a much smaller number of examples, provided that we content ourselves with a small or 'simple' decision tree.

In order to formally apply the relevant bound, we need to define a description language for decision trees, which is prefix free and requires less bits for small decision trees. Here is one possible way: first, notice that using $\log(d)+1$ bits, it is possible to encode any of the following:

1. A node of the form '$x_i = 0$?' for any $i \in \{1,\ldots,d\}$.
2. A leaf whose value is 0 or 1.

Given a tree, we can describe it as a concatenation of blocks, each of size $\log(d)+1$ bits. Each block represent a node/leaf/end-of-coding, assuming that the order represents a depth-first walk on the tree. Thus, the first block represents the root of the tree, the second block represents the left sub-tree etc.. Assuming each internal node has two children[2], It is not hard to show that this is a prefix-free encoding of the tree, and that the description length of a tree with n nodes is $n(\log(d)+1)$.

Therefore, we have by Theorem 3 that with probability at least $1-\delta$ over a sample of size m, it holds for any n and any decision tree $h \in \mathcal{H}$ with n nodes that

$$\mathrm{err}_{\mathcal{D}}(h) \leq \mathrm{err}_S(h) + \sqrt{\frac{n(\log(d)+1) + \log(2/\delta)}{2m}}. \tag{13}$$

We see that this bound performs a trade-off: on the one hand, we expect larger, more complex decision trees to have a small $\mathrm{err}_S(h)$, but the respective value of n will be large. On the other hand, small decision trees will have a small value of n, but $\mathrm{err}_S(h)$ might be larger. Our hope is that we can find a decision tree with both low empirical error $\mathrm{err}_S(h)$, and with a number of nodes n not too high. Our bound indicates that such a tree will have low generalization error $\mathrm{err}_{\mathcal{D}}(h)$.

14 Decision Tree Algorithms

Ideally, we wish to have a learning algorithm which given a sample, outputs a decision tree whose generalization error $\mathrm{err}_{\mathcal{D}}(h)$ is as small as possible. Unfortunately, it turns out that even designing a decision tree that minimizes the empirical error $\mathrm{err}_S(h)$ is NP-complete. Consequently, practical decision-tree learning algorithms are based on heuristics such as a greedy approach, where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree but tend to work reasonably well in practice.

A general framework for growing a decision tree is as follows. We start with a tree with a single leaf (the root) and assign this leaf a label according to a majority vote among all labels over the training set. We now perform a series of iterations. On each iteration, we examine the effect of splitting a single leaf. We define some 'gain' measure that quantifies the improvement due to this split. Then, among all possible splits, we choose the one that maximizes the gain and perform it. In Algorithm 1, we give a possible implementation. It is based on a popular decision tree algorithm known as 'ID3' (short for 'Iterative Dichotomizer 3'), adapted to our simple setting of binary features. The algorithm works by recursive calls, with the beginning call being $\texttt{ID3}(S, \{1,\ldots,d\})$, and returns a decision tree. The notation $\mathbb{P}_S(A)$ for some predicate A denotes here the percentage of examples in S where A holds.

[2]We may assume this without loss of generality, because if a decision node has only one child, we can replace the node by its child without affecting the behavior of the decision-tree.

Algorithm 1 $\texttt{ID3}(S, A)$

Input: Training set S, feature subset $A \subseteq \{1, \ldots, d\}$.
If all examples in S are labeled by 1, return a leaf 1.
If all examples in S are labeled by 0, return a leaf 0.
If $A = \emptyset$, return a leaf whose value = majority label in S.
Else:
  Let $i_b = \arg\max_{i \in \{1,\ldots,d\}} \texttt{Gain}(S, i)$.
  If $\mathbb{P}_S(x_{i_b} = 0)$ equals 0 or 1, return a leaf whose value = majority label in S.
  Else:
    Let T_1 be the tree returned by $\texttt{ID3}(\{(\mathbf{x}, y) \in S : x_{i_b} = 0\}, A \setminus i_b)$.
    Let T_2 be the tree returned by $\texttt{ID3}(\{(\mathbf{x}, y) \in S : x_{i_b} = 1\}, A \setminus i_b)$.
    Return a tree whose root is '$x_{i_b} = 0$?', T_1 being the subtree corresponding to a positive answer,
      and T_2 being the subtree corresponding to a negative answer.

Different algorithms use different implementations of $\texttt{Gain}(S, i)$. The simplest definition of gain is maybe the decrease in training error. Formally, if we define $C(x) = \min\{x, 1 - x\}$, notice that the training error before splitting on feature i is $C(\mathbb{P}_S(y = 1))$, and the error after splitting on feature i is $\mathbb{P}_S(x_i = 1)C(\mathbb{P}_S(y = 1|x_i = 1)) + \mathbb{P}_S(x_i = 0)C(\mathbb{P}_S(y = 1|x_i = 0))$. Therefore, we can define $\texttt{Gain}$ to be the difference between the two, namely

$$\texttt{Gain}(S, i) := C(\mathop{\mathbb{P}}_S(y = 1)) - \left(\mathop{\mathbb{P}}_S(x_i = 1)C(\mathop{\mathbb{P}}_S(y = 1|x_i = 1)) + \mathop{\mathbb{P}}_S(x_i = 0)C(\mathop{\mathbb{P}}_S(y = 1|x_i = 0))\right).$$

Another popular gain measure that is used in the ID3 and C4.5 algorithms of Quinlan is the information gain. The information gain is the difference between the entropy of the label before and after the split, and is achieved by replacing the function C in the expression above by the entropy function H, where $H(x) = -x\log(x) - (1 - x)\log(1 - x)$. Yet another definition of a gain is based on the Gini index. We will discuss the properties of those measures in the exercise.

The algorithm described above still suffers from a big problem: the returned tree will usually be very large. Such trees may have low empirical error, but their generalization error will tend to be high - both according to our theoretical analysis, and in practice. A common solution is to *prune* the tree after it is built, hoping to reduce it to a much smaller tree, but still with a similar empirical error. Theoretically, according to the bound in Eq. (13), if we can make n much smaller without increasing $\mathrm{err}_S(h)$ by much, we are likely to get a decision tree with better generalization error.

Usually, the pruning is performed by a bottom-up walk on the tree. Each node might be replaced with one of its subtrees or with a leaf, based on some bound or estimate for the generalization error (for example, the bound in Eq. (13)). See algorithm 2 for a common template.

Algorithm 2 Generic Tree Pruning Procedure

Input: Function $f(T, m)$ (bound/estimate for the generalization error of a decision tree T,
  based on a sample of size m), tree T.
Perform a bottom-up walk on T (from leaves to root). At each node j:
  Find T' which minimizes $f(T', m)$, where T' is any of the following:
    The current tree after replacing node j with a leaf 1.
    The current tree after replacing node j with a leaf 0.
    The current tree after replacing node j with its left subtree.
    The current tree after replacing node j with its right subtree.
    The current tree.
  Let $T := T'$.

(67577) Introduction to Machine Learning November 9, 2009

Lecture 6 – VC dimension and No-Free-Lunch

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In previous lectures we saw that learning is possible if we incorporate some prior knowledge in the form of a finite hypothesis class or a weighting over hypotheses. Is prior knowledge really necessary? Maybe there exists some super-learner that can learn without any prior knowledge? We start this lecture by establishing that learning is impossible without employing some prior knowledge. Next, having established that we must have some form of prior knowledge, we will get back to the (agnostic) PAC framework in which the prior knowledge takes the form of a hypothesis class. Our goal will be to exactly characterize which hypothesis classes are learnable in the PAC model. We saw that the finiteness of a hypothesis class is a sufficient condition for its learnability. But, is it necessary? We will see that finiteness is not a necessary condition. Instead, a combinatorial measure, called VC dimension, characterizes learnability of hypothesis classes. Namely, a hypothesis is learnable in the PAC model if and only if its VC dimension is finite.

15 No Free Lunch

The following no-free-lunch theorem states that no learning algorithm can work for all distributions. In particular, for any learning algorithm, there exists a distribution such that the Bayes optimal error is 0 (so there is a perfect predictor), but the learning algorithm will fail to find a predictor with a small generalization error.

Theorem 5 *Let m be a training set size and let $\mathcal{X}$ be an instance space such that there are at least $2m$ distinct instances in $\mathcal{X}$. For every learning algorithm, A, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ and a predictor $f : \mathcal{X} \to \{0,1\}$ such that $\mathrm{err}_{\mathcal{D}}(f) = 0$ but*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}[\mathrm{err}_{\mathcal{D}}(A(S))] \geq 1/4 \,.$$

To prove the theorem we will use the following lemma.

Lemma 6 *Let C be a set of size $2m$ and let $\mathcal{F}$ be the set of all functions from C to $\{0,1\}$. Let $c^{(m)} = (c_1, \ldots, c_m)$ denote a sequence of m elements from C and let $f(c^{(m)}) = (f(c_1), \ldots, f(c_m))$. Let $U(C)$ be the uniform distribution over C and for any $f \in \mathcal{F}$ let $\mathcal{D}_f$ be the distribution over $C \times \{0,1\}$ such that the probability to choose a pair (c, y) is $1/|C|$ if $y = f(c)$ and 0 otherwise. Then, for any function $A : (C \times \{0,1\})^m \to \mathcal{F}$ there exists $f \in \mathcal{F}$ such that*

$$\mathop{\mathbb{E}}_{c^{(m)} \sim U(C)^m} \left[\mathrm{err}_{\mathcal{D}_f} \left(A \left(c^{(m)}, f(c^{(m)}) \right) \right) \right] \;\geq\; 1/4 \,.$$

Proof It suffices to prove that:

$$\max_{f \in \mathcal{F}} \mathop{\mathbb{E}}_{c^{(m)} \sim U(C)^m} \left[\mathrm{err}_{\mathcal{D}_f} \left(A \left(c^{(m)}, f(c^{(m)}) \right) \right) \right] \;\geq\; 1/4 \,.$$

We will prove the stronger result:

$$\mathop{\mathbb{E}}_{f \sim U(\mathcal{F})} \mathop{\mathbb{E}}_{c^{(m)} \sim U(C)^m} \left[\mathrm{err}_{\mathcal{D}_f} \left(A \left(c^{(m)}, f(c^{(m)}) \right) \right) \right] \;\geq\; 1/4 \,, \tag{14}$$

where $U(\mathcal{F})$ is the uniform distribution over $\mathcal{F}$. This proof technique, in which we show the existence of f by analyzing the expectation of f according to some distribution over $\mathcal{F}$ is called *the probabilistic method*. To show that Eq. (14) holds we first note that

$$\mathrm{err}_{\mathcal{D}_f}\left(A\left(c^{(m)}, f(c^{(m)})\right)\right) = \mathop{\mathbb{E}}_{c\sim U(C)}\left[\mathbb{1}_{[A\left(c^{(m)}, f(c^{(m)})\right)(c)\neq f(c)]}\right] ,$$

where $A\left(c^{(m)}, f(c^{(m)})\right)(c)$ is the prediction of the output of A on c when the input of A is the labeled training set, $(c^{(m)}, f(c^{(m)}))$, and for a predicate π the function $\mathbb{1}_{[\pi]}$ is 1 if π holds and 0 otherwise. Plugging the above into the left-hand side of Eq. (14) and using the linearity of the expectation we obtain

$$\begin{aligned}&\mathop{\mathbb{E}}_{f\sim U(\mathcal{F})}\mathop{\mathbb{E}}_{c^{(m)}\sim U(C)^m}\left[\mathrm{err}_{\mathcal{D}_f}\left(A\left(c^{(m)}, f(c^{(m)})\right)\right)\right] \\ &\qquad = \mathop{\mathbb{E}}_{c^{(m)}\sim U(C)^m}\mathop{\mathbb{E}}_{c\sim U(C)}\mathop{\mathbb{E}}_{f\sim U(\mathcal{F})}\left[\mathbb{1}_{[A\left(c^{(m)}, f(c^{(m)})\right)(c)\neq f(c)]}\right] .\end{aligned} \tag{15}$$

Now, fix some $c^{(m)}$. For any sequence of labels $y^{(m)} = (y_1, \ldots, y_m)$, let $\mathcal{F}_{y^{(m)}} = \{f \in \mathcal{F} : f(c_1) = y_1, \ldots, f(c_m) = y_m\}$. For all the functions $f \in \mathcal{F}_{y^{(m)}}$, the output of $A\left(c^{(m)}, f(c^{(m)})\right)$ is the same predictor. Therefore, for any c which is not in $c^{(m)}$, the value of $A\left(c^{(m)}, f(c^{(m)})\right)(c)$ does not depend on which f from $f \in \mathcal{F}_{y^{(m)}}$ is chosen. Therefore, for each c which is not in $c^{(m)}$ we have

$$\mathop{\mathbb{E}}_{f\sim U(\mathcal{F})}\left[\mathbb{1}_{[A\left(c^{(m)}, f(c^{(m)})\right)(c)\neq f(c)]}\right] = \frac{1}{|\mathcal{F}|}\sum_{y^{(m)}\in\{0,1\}^m}\sum_{f\in\mathcal{F}_{y^{(m)}}}\mathbb{1}_{[A\left(c^{(m)}, y^{(m)}\right)(c)\neq f(c)]} = \frac{1}{2} , \tag{16}$$

where the last equality is from a straightforward symmetry argument — half the functions in $\mathcal{F}_{y^{(m)}}$ will predict $f(c) = 1$ and the other half will predict $f(c) = 0$. Since the number of elements $c \in C$ that do not belong to $c^{(m)}$ is at least m, i.e. half the size of C, we obtain that

$$\forall c^{(m)}, \quad \mathop{\mathbb{E}}_{c\sim U(C)}\mathop{\mathbb{E}}_{f\sim U(\mathcal{F})}\left[\mathbb{1}_{[A\left(c^{(m)}, f(c^{(m)})\right)(c)\neq f(c)]}\right] \geq 1/4 .$$

Combining the above with Eq. (15) we conclude our proof. ■

Based on the above lemma, the proof of the No-Free-Lunch theorem easily follows.

Proof [of Theorem 5] The proof follows from Lemma 6 as follows. By assumption, there exists $C \subset \mathcal{X}$ with $2m$ distinct values. To prove the theorem it suffices to find a distribution over $C \times \{0, 1\}$ and we will simply let the probability of instances outside of C to be 0. For any distribution $\mathcal{D}_f$ defined in Lemma 6, we clearly have that $\mathrm{err}_{\mathcal{D}_f}(f) = 0$ and that the probability to choose a training set of m examples from $\mathcal{D}_f$ is the same as the probability to choose m instances i.i.d. from $U(C)$. Additionally, a learning algorithm is a function that receives m instances from C along with their labels according to some f, and returns a predictor over $\mathcal{X}$. But, since we only care about predictions on the set C (because the distribution is focused on C), we can think on the predictor as a mapping from C to $\{0, 1\}$, i.e. an element of the set $\mathcal{F}$ defined in Lemma 6. The claim of Lemma 6 now concludes our proof. ■

Remark 1 *It is easy to see that if C is a set with km distinct instances, with $k \geq 2$, then we can replace the lower bound of $1/4$ in Lemma 6 with $\frac{k-1}{2k} = \frac{1}{2} - \frac{1}{2k}$. Namely, when k is large the lower bound becomes close to $1/2$.*

16 Size does not matter

In the previous section we saw a No-Free-Lunch result, telling us that no algorithm can learn without prior knowledge. In this section we get back to the PAC framework, in which the prior knowledge is in the form

of a hypothesis class. Recall that in the agnostic PAC framework, the learning algorithm should output a predictor whose performance is comparable to the best predictor in a predefined hypothesis class, $\mathcal{H}$. Our goal is to characterize what property of a hypothesis class determines its learnability. Furthermore, we would like to find a "complexity" measure of hypothesis classes that tells us how difficult learning a class $\mathcal{H}$ is.

A natural complexity measure of a hypothesis class is the size of the class. Indeed, we saw in previous lectures that finite size is sufficient for PAC learnability. However, as the following simple example shows, finite size is not a *necessary* requirement for PAC learnability.

Let $\mathcal{H}$ be the set of threshold functions from $\mathbb{R} \to \mathbb{R}$, namely, $\mathcal{H} = \{x \mapsto \mathbb{1}_{[x<a]} : a \in \mathbb{R}\}$. To remind the reader, $\mathbb{1}_{[x<a]}$ is 1 if $x < a$ and 0 otherwise. We will show that $\mathcal{H}$ is learnable in the PAC model. The learning algorithm will be: Given a training set S, define $a_S = \max x : (x, 1) \in S$, and set the output predictor to be $x \mapsto \mathbb{1}_{[x<a_S]}$. That is, we found the largest positive example in the training set and set the threshold to be the value of that example. The following lemma shows that this algorithm is probably approximately correct.

Lemma 7 *Let $\mathcal{H} = \{x \mapsto \mathbb{1}_{[x<a]} : a \in \mathbb{R}\}$ be the class of thresholds. For any $\epsilon, \delta > 0$, let $m = \log(1/\delta)/\epsilon$. Then, for any distribution $\mathcal{D}$ over $\mathbb{R} \times \{0, 1\}$, such that exists $h \in \mathcal{H}$ with $\mathrm{err}_{\mathcal{D}}(h) = 0$, there exists a learning algorithm A such that*

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_{\mathcal{D}}(A(S)) > \epsilon] \leq \delta \ .$$

Proof Let $a^\star$ be a threshold such that the hypothesis $h^\star(x) = \mathbb{1}_{[x<a^\star]}$ achieves 0 generalization error. Let $\mathcal{D}_x$ be the marginal distribution over instances and let $a_0 < a^\star$ be such that

$$\mathop{\mathbb{P}}_{x\sim\mathcal{D}_x}[x \in (a_0, a^\star)] = \epsilon \ .$$

Consider the algorithm that sets the threshold to be $a_S = \max x : (x, 1) \in S$ (and if no example in S is positive we set $a_S = -\infty$) and let $h_S(x) = \mathbb{1}_{[x<a_S]}$. Since we assume $\mathrm{err}_{\mathcal{D}}(h^\star) = 0$ we have that (with probability 1) no positive example in S can be larger than $a^\star$ and thus $a_S < a^\star$. Therefore,

$$\mathrm{err}_{\mathcal{D}}(h_S) = \mathop{\mathbb{P}}_{x\sim\mathcal{D}_x}[x \in (a_S, a^\star)] \ .$$

Thus, $\mathrm{err}_{\mathcal{D}}(h_S) > \epsilon$ if and only if $a_S < a_0$ which will happen if and only if all examples in S are not in the interval $(a_0, a^\star)$. Namely,

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\mathrm{err}_{\mathcal{D}}(h_S) > \epsilon] = \mathop{\mathbb{P}}_{S\sim\mathcal{D}^m}[\forall (x, y) \in S,\ x \notin (a_0, a^\star)] = (1 - \epsilon)^m \leq e^{-\epsilon\, m} \ .$$

Since we assume $m > \log(1/\delta)/\epsilon$ it follows that the above is at most δ and our proof is concluded. ■

17 VC dimension

In the previous section we saw that the size of $\mathcal{H}$ is not a good complexity measure because the class of threshold functions is of infinite size but is PAC learnable. Intuitively, although the class of threshold functions is infinite, when restricting it to a finite set of instances, $C = \{c_1, \ldots, c_m\}$, we obtain a class of small size — there are only $m + 1$ different functions from $C \to \{0, 1\}$ that can be derived from the restriction of $\mathcal{H}$ to C. This number is much smaller than all 2^m potential number of functions from C to $\{0, 1\}$. The restriction of $\mathcal{H}$ to a finite set is an important operation and we give it a dedicated notation.

Definition 3 (Restriction of $\mathcal{H}$ to C) *Let $\mathcal{H}$ be a class of functions from $\mathcal{X}$ to $\{0, 1\}$ and let $C = \{c_1, \ldots, c_m\} \subset \mathcal{X}$. The restriction of $\mathcal{H}$ to C is the set of functions from C to $\{0, 1\}$ that can be derived from $\mathcal{H}$. That is,*

$$\mathcal{H}_C = \{(h(c_1), \ldots, h(c_m)) : h \in \mathcal{H}\} \ ,$$

where we represent each function from C to $\{0, 1\}$ as a vector in $\{0, 1\}^{|C|}$.

Intuitively, if $\mathcal{H}_C$ is all the functions from C to $\{0,1\}$ then no algorithm can learn using a training set from C because $\mathcal{H}$ can explain any sequence of labels over C. This negative result is formalized in the lemma below, whose proof follows easily from Lemma 6.

Lemma 8 *Let $\mathcal{H}$ be a hypothesis class of functions from $\mathcal{X}$ to $\{0,1\}$. Let m be a training set size. Assume that there exists a set $C \subset \mathcal{X}$ of size $2m$ such that the restriction of $\mathcal{H}$ to C is the set of all functions from C to $\{0,1\}$. That is, $|\mathcal{H}_C| = 2^{2m}$. Then, for any learning algorithm, A, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0,1\}$ and a predictor $h \in \mathcal{H}$ such that $\mathrm{err}_{\mathcal{D}}(h) = 0$ but*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}[\mathrm{err}_{\mathcal{D}}(A(S))] \geq 1/4 \,.$$

In the above lemma it is assumed that the restriction of $\mathcal{H}$ to C is the set of all functions from C to $\{0,1\}$. In this case we say that $\mathcal{H}$ *shatters* the set C. Formally:

Definition 4 (Shattering) *A hypothesis class $\mathcal{H}$ shatters a finite set $C \subset \mathcal{X}$ if the restriction of $\mathcal{H}$ to C is the set of all functions from C to $\{0,1\}$. That is, $|\mathcal{H}_C| = 2^{|C|}$.*

Lemma 8 tells us that if $\mathcal{H}$ shatters some set C of size $2m$ then we cannot learn $\mathcal{H}$ using m examples (in the PAC sense). Intuitively, if a set C is shattered by $\mathcal{H}$, and we receive a sample containing half the instances of C, the labels of these instances give us no information about the labels of the rest of the instances in C. This is simply because every possible labeling can be explained by some hypothesis $\mathcal{H}$. Philosophically,

> *If someone can explain every phenomena, his explanations are worthless.*

Shattering leads us directly to the VC dimension — a complexity measure of hypothesis classes defined by Vapnik and Chervonenkis.

Definition 5 (VC dimension) *The VC dimension of a hypothesis class $\mathcal{H}$, denoted $\mathrm{VCdim}(\mathcal{H})$, is the maximal size of a set $C \subset \mathcal{X}$ that can be shattered by $\mathcal{H}$. If $\mathcal{H}$ can shatter sets of arbitrarily large size we say that $\mathcal{H}$ has infinite VC dimension.*

A direct consequence of Lemma 8 is therefore:

Theorem 6 *Let $\mathcal{H}$ be a class with infinite VC dimension. Then, $\mathcal{H}$ is not PAC learnable.*

Proof Since $\mathcal{H}$ has an infinite VC dimension, for any training set size m, there exists a shattered set of size $2m$, and the claim follows directly from Lemma 8. ■

The fact that a class with infinite VC dimension is not learnable is maybe not surprising. After all, as we argued before, if all possible labeling sequences are possible, previously observed examples give us no information on unseen examples. The more surprising fact is that the converse statement is also true:

Theorem 7 *Let $\mathcal{H}$ be a class with finite VC dimension. Then, $\mathcal{H}$ is agnostically PAC learnable.*

The proof of Theorem 7 is based on two claims:

- In the next section, we will show that if $\mathrm{VCdim}(\mathcal{H}) = d$ then even though $\mathcal{H}$ might be infinite, when restricting it to a finite set $C \subset \mathcal{X}$, its "effective" size, $|\mathcal{H}_C|$, is only $O(|C|^d)$. That is, the size of $\mathcal{H}_C$ grows polynomially rather than exponentially with $|C|$.
- In the next lecture, we will present generalization bounds using a technique called Rademacher complexities. In particular, this technique[3] will enable us to extend the "learnability of finite classes" result we established in previous lectures to "learnability of classes with small effective size". By "small effective size" we mean classes for which $|\mathcal{H}_C|$ grows polynomially with $|C|$.

[3] It is possible to prove learnability of classes with small effective size directly, without relying on Rademacher complexities, but the proof that relies on Rademacher complexities is more simple and intuitive.

18 The growth function and Sauer-Shelah lemma

In the previous section we defined the notion of *shattering*, by considering the restriction of $\mathcal{H}$ to a finite set of instances. The growth function measures the maximal "effective" size of $\mathcal{H}$ on a set of m examples. Formally:

Definition 6 (Growth function) *Let $\mathcal{H}$ be a hypothesis class. Then the* growth function *of $\mathcal{H}$, denoted $\tau_{\mathcal{H}} : \mathbb{N} \to \mathbb{N}$, is defined as*

$$\tau_{\mathcal{H}}(m) \;=\; \max_{C \subset \mathcal{X} : |C| = m} \left| \mathcal{H}_C \right| .$$

In words, $\tau_H(m)$ is the number of different functions from C to $\{0,1\}$ that can be obtained by restricting $\mathcal{H}$ to C.

Obviously, if the $\mathrm{VCdim}(\mathcal{H}) = d$ then for any $m \leq d$ we have $\tau_{\mathcal{H}}(m) = 2^m$. In such cases, $\mathcal{H}$ induces all possible functions from C to $\{0,1\}$. The following beautiful lemma, proposed independently by Sauer and Shelah, shows that when m becomes larger than the VC dimension, the growth function increases polynomially rather than exponentially with m.

Lemma 9 (Sauer-Shelah) *Let $\mathcal{H}$ be a hypothesis class with $\mathrm{VCdim}(\mathcal{H}) \leq d < \infty$. Then, for all m, $\tau_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}$. In particular, if $m > d$ then $\tau_{\mathcal{H}}(m) \leq (em/d)^d$.*

Proof To prove the lemma it suffices to prove the following stronger claim: For any $C = \{c_1, \ldots, c_m\}$ we have

$$\forall \mathcal{H}, \quad \left| \mathcal{H}_C \right| \;\leq\; \left| \{ B \subseteq C : \mathcal{H} \text{ shatters } B \} \right| . \tag{17}$$

The reason why Eq. (17) is sufficient to prove the lemma is because if $\mathrm{VCdim}(\mathcal{H}) \leq d$ then no set whose size is larger than d is shattered by $\mathcal{H}$ and therefore

$$\left| \{ B \subseteq C : \mathcal{H} \text{ shatters } B \} \right| \;\leq\; \sum_{i=0}^{d} \binom{m}{i} .$$

When $m > d$ the right-hand side of the above is at most $(em/d)^d$ (proving the latter fact is left as an exercise). We are left with proving Eq. (17) and we do it using an inductive argument. For $m = 1$, no matter what $\mathcal{H}$ is, either both sides of Eq. (17) equal 1 or both sides equal 2 (the empty set is always considered to be shattered by $\mathcal{H}$). Assume Eq. (17) holds for sets of size $k < m$ and let us prove it for sets of size m. Fix $\mathcal{H}$ and $C = \{c_1, \ldots, c_m\}$. Denote $C' = \{c_2, \ldots, c_m\}$ and in addition, define the following two sets:

$$Y_0 = \{ (y_2, \ldots, y_m) : (0, y_2, \ldots, y_m) \in \mathcal{H}_C \vee (1, y_2, \ldots, y_m) \in \mathcal{H}_C \} ,$$

and

$$Y_1 = \{ (y_2, \ldots, y_m) : (0, y_2, \ldots, y_m) \in \mathcal{H}_C \wedge (1, y_2, \ldots, y_m) \in \mathcal{H}_C \} .$$

It is easy to verify that $|\mathcal{H}_C| = |Y_0| + |Y_1|$. Additionally, since $Y_0 = \mathcal{H}_{C'}$, using the induction assumption (applied on $\mathcal{H}$ and C') we have that

$$|Y_0| = |\mathcal{H}_{C'}| \leq |\{ B \subseteq C' : \mathcal{H} \text{ shatters } B \}| = |\{ B \subseteq C : c_1 \notin B \wedge \mathcal{H} \text{ shatters } B \}| .$$

Next, define $\mathcal{H}' \subseteq \mathcal{H}$ to be:

$$\mathcal{H}' = \{ h \in \mathcal{H} : \exists h' \in \mathcal{H} \text{ s.t. } (1 - h'(c_1), h'(c_2), \ldots, h'(c_m)) = (h(c_1), h(c_2), \ldots, h(c_m)) \} .$$

Namely, $\mathcal{H}'$ contains pairs of hypotheses that agree on C' and differ on c_1. Using this definition, it is clear that if $\mathcal{H}'$ shatters a set $B \subseteq C'$ then it also shatters the set $B \cup \{c_1\}$. Combining this with the fact that $Y_1 = \mathcal{H}'_{C'}$ and using the inductive assumption (now applied on $\mathcal{H}'$ and C') we obtain that,

$$\begin{aligned} |Y_1| = |\mathcal{H}'_{C'}| &\leq |\{ B \subseteq C' : \mathcal{H}' \text{ shatters } B \}| = |\{ B \subseteq C' : \mathcal{H}' \text{ shatters } B \cup \{c_1\} \}| \\ &= |\{ B \subseteq C : c_1 \in B \wedge \mathcal{H}' \text{ shatters } B \}| \leq |\{ B \subseteq C : c_1 \in B \wedge \mathcal{H} \text{ shatters } B \}| . \end{aligned}$$

Overall, we have shown that

$$\begin{aligned}|\mathcal{H}_C| &= |Y_0| + |Y_1| \\ &\leq |\{B \subseteq C : c_1 \notin B \wedge \mathcal{H} \text{ shatters } B\}| + |\{B \subseteq C : c_1 \in B \wedge \mathcal{H} \text{ shatters } B\}| \\ &= |\{B \subseteq C : \mathcal{H} \text{ shatters } B\}| \,,\end{aligned}$$

which concludes our proof. ■

19 Examples

In this section we calculate the VC dimension of several hypothesis classes.

19.1 Threshold functions

Let $\mathcal{H}$ be the class of threshold functions over $\mathbb{R}$, namely, $\mathcal{H} = \{x \mapsto \mathbb{1}_{[x<a]} : a \in \mathbb{R}\}$. Take a set $C = \{c_1\}$. Then, $\mathcal{H}$ shatters C and therefore $\mathrm{VCdim}(\mathcal{H}) \geq 1$. Now take a set $C = \{c_1, c_2\}$ and assume without loss of generality that $c_1 < c_2$. Then, the labeling $(1, 0)$ cannot be obtained by a threshold and therefore $\mathcal{H}$ does not shatter C. We therefore conclude that $\mathrm{VCdim}(\mathcal{H}) = 1$.

19.2 Finite classes

Let $\mathcal{H}$ be a finite class. Then, clearly, for any set C we have $|\mathcal{H}_C| \leq |\mathcal{H}|$ and thus C cannot be shattered if $|\mathcal{H}| < 2^{|C|}$. This implies that $\mathrm{VCdim}(h) \leq \log_2(|\mathcal{H}|)$. This shows that the PAC learnability of finite classes follows from the more general statement of PAC learnability of classes with finite VC dimension.

19.3 Axis aligned rectangles

Let $\mathcal{H}$ be the class of axis aligned rectangles. We show below, $\mathrm{VCdim}(\mathcal{H}) = 4$. To prove that $\mathrm{VCdim}(\mathcal{H}) = 4$ we need to find a set of 4 points that are shattered by $\mathcal{H}$ and we also need to show that no set of 5 points can be shattered by $\mathcal{H}$. Finding a set of 4 points that are shattered is easy (see Figure 3). Consider any set $C \subset \mathbb{R}^2$ of 5 points. In C, take a leftmost point (whose first coordinate is the smallest in C), a rightmost point (first coordinate is the largest), a lowest point (second coordinate is the smallest), and a highest point (second coordinate is the largest); let $x \in C$ be the (fifth) point that was not selected. Without loss of generality, denote $C = \{c_1, \ldots, c_5\}$ and let c_5 be the point that was not selected. Now, define the labeling $(1, 1, 1, 1, 0)$. It is impossible to make this labeling by an axis-aligned rectangle. Indeed, such a rectangle must contain $c_1, \ldots, c_4$; but in this case the rectangle contains c_5 as well, because its coordinates are within the intervals spanned by selected points. So, C is not shattered by H, and therefore $\mathrm{VCdim}(H) = 4$.

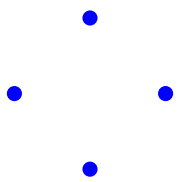


Figure 3: The following 4 points are shattered by axis-aligned rectangles.

20 Exercises

1. Prove that $\sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{e\,m}{d}\right)^d$.

(67577) Introduction to Machine Learning November 16, 2009

Lecture 7 – Rademacher Complexities

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the previous lecture we argued that the VC dimension of a hypothesis class determines its learnability. In this lecture we will describe another, more general, complexity measure of hypothesis classes that is called Rademacher complexities. We will provide generalization bounds based on this measure. Additionally, we shall bound the Rademacher complexity of a hypothesis class based on its VC dimension, and as a result will complete the proof of the learnability of hypothesis classes with finite VC dimension.

21 General loss functions

In the previous lecture we studied the problem of learning binary classifiers. In this section we extend our framework to more general prediction problems. In the general learning model we study now, there is a loss function $\ell(h, \mathbf{z})$ where $h \in \mathcal{H}$ is the hypothesis we wish to learn and $\mathbf{z}$ is a data point. The loss function assesses how good h performs on the example $\mathbf{z}$. As before, we assume there is an unknown distribution $\mathcal{D}$ over examples and our goal is to find $h \in \mathcal{H}$ with a good generalization properties, where now the generalization loss is defined as

$$L(h) \;=\; \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathcal{D}}[\ell(h, \mathbf{z})] \;. \tag{18}$$

To do so, we can get a sample $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m)$ where each $\mathbf{z}_i$ is sampled i.i.d. from $\mathcal{D}$. We denote the average loss of a hypothesis h on the training sample as

$$L_S(h) \;=\; \frac{1}{m}\sum_{i=1}^{m} \ell(h, \mathbf{z}_i) \;. \tag{19}$$

The basic learning question is how we can use S for learning a hypothesis $h \in \mathcal{H}$ with a low generalization loss? And, how does the learnability of $\mathcal{H}$ depend on properties of $\mathcal{H}$ and ℓ ?

This more general learning model encompasses binary classification as a special case by defining $\mathbf{z} = (\mathbf{x}, y)$ and $\ell(h, \mathbf{z}) = \mathbb{1}_{[h(\mathbf{x})\neq y]}$. As before, the goal is to find h which approximately minimizes the generalization error, $L(h) = \mathbb{E}_{z\sim\mathcal{D}}[\ell(h, \mathbf{z})] = \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}[h(\mathbf{x}) \neq y]$.

We can also study other learning problems. For example, in regression problems $\mathbf{z} = (\mathbf{x}, y)$ where now $y \in \mathbb{R}$ is a scalar and each hypothesis is a mapping $h : \mathcal{X} \to \mathbb{R}$. Widely used loss functions for regression are the absolute value loss, $\ell(h, \mathbf{z}) = |h(\mathbf{x}) - y|$, and the squared loss, $\ell(h, \mathbf{z}) = (h(\mathbf{x}) - y)^2$.

We can even study unsupervised learning problems, such as clustering, under this model. For example, in k-means clustering, which will be studied later in this book, each example is a vector in $\mathbb{R}^d$, and each hypothesis is a set of k vectors in $\mathbb{R}^d$, denoted $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$. The loss function is the squared Euclidean distance from a vector $\mathbf{z}$ to the closest vector in $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$,

$$\ell(\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}, \mathbf{z}) \;=\; \min_{i\in[k]} \|\mathbf{z} - \boldsymbol{\mu}_i\|^2 \;.$$

22 Rademacher complexity and data-dependent bounds

Let $\mathcal{H}$ be a class of hypotheses. In the previous lecture we showed how the VC dimension measures the complexity of classes of binary classifiers. In this section we introduce another notion of complexity which is called Rademacher complexity. While the VC dimension is a combinatorial measure, the Rademacher complexity has an algebraic flavour. In particular, we now allow $\mathcal{H}$ to be a class of hypotheses mapping from

$\mathcal{X}$ to $\mathbb{R}$. Furthermore, the loss function also affects the complexity of learning. We therefore talk about the complexity of $\ell \circ \mathcal{H} = \{\mathbf{z} \mapsto \ell(h, \mathbf{z}) : h \in \mathcal{H}\}$.

To motivate the Rademacher complexity measure, recall that an overfitting occurs when the training loss significantly differs from the generalization loss. That is, given a training set $S = \{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$, overfitting might occur if for some hypothesis $h \in \mathcal{H}$ we have that $L(h) - L_S(h)$ is large. We therefore obtain the following measure of the complexity of $\ell \circ \mathcal{H}$ with respect to S:

$$\sup_{h \in \mathcal{H}} L(h) - L_S(h) \,. \tag{20}$$

Now, suppose we would like to base the complexity measure only on S. One simple idea is to split S into two disjoint sets, $S = S_1 \cup S_2$, refer to S_1 as a training set and to S_2 as a validation set. We can then estimate Eq. (20) by

$$\sup_{h \in \mathcal{H}} L_{S_1}(h) - L_{S_2}(h) \,. \tag{21}$$

This can be written more compactly by defining $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m) \in \{\pm 1\}^m$ to be a vector such that $S_1 = \{\mathbf{z}_i : \sigma_i = 1\}$ and $S_2 = \{\mathbf{z}_i : \sigma_i = -1\}$. Then, if we further assume that $|S_1| = |S_2|$ then Eq. (21) can be rewritten as

$$\frac{2}{m} \sup_{h \in \mathcal{H}} \sum_{i=1}^{m} \sigma_i \ell(h, \mathbf{z}_i) \,. \tag{22}$$

The Rademacher complexity measure captures the above idea by considering the expectation of the above with respect to a random choice of $\boldsymbol{\sigma}$. Formally, let the variables in $\boldsymbol{\sigma}$ be distributed i.i.d. according to $\mathbb{P}[\sigma_i = 1] = \mathbb{P}[\sigma_i = -1] = \frac{1}{2}$. Then, the Rademacher complexity of $\ell \circ \mathcal{H}$ with respect to S is defined as follows:

$$R(\ell \circ \mathcal{H} \circ S) \;=\; \frac{1}{m} \mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \{\pm 1\}^m} \left[\sup_{h \in \mathcal{H}} \sum_{i=1}^{m} \sigma_i \ell(h, \mathbf{z}_i) \right] . \tag{23}$$

More generally, given a set of vectors, $A \subset \mathbb{R}^m$, we defined

$$R(A) \;=\; \frac{1}{m} \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \left[\sup_{\mathbf{a} \in A} \sum_{i=1}^{m} \sigma_i a_i \right] . \tag{24}$$

The above definition coincides with Eq. (23) for the set of all possible loss values a hypothesis $h \in \mathcal{H}$ can achieve on a sample S, namely, $\ell \circ \mathcal{H} \circ S = \{(\ell(h, \mathbf{z}_1), \ldots, \ell(h, \mathbf{z}_m)) : h \in \mathcal{H}\}$.

The following lemma formalizes the above intuitive arguments by comparing the expected value of Eq. (20) with the expected value of $R(\ell \circ \mathcal{H} \circ S)$.

Lemma 10 *We have*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} \left[\sup_{h \in \mathcal{H}} L(h) - L_S(h) \right] \;\leq\; 2 \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} R(\ell \circ \mathcal{H} \circ S) \,.$$

Proof Let $S' = \{\mathbf{z}'_1, \ldots, \mathbf{z}'_m\}$ be another i.i.d. sample. Clearly, for all h, $L(h) = \mathbb{E}_{S'}[L_{S'}(h)]$. Using the fact that supremum of expectation is smaller than expectation of supremum we obtain

$$\begin{aligned}
\mathop{\mathbb{E}}_{S} \left[\sup_{h \in \mathcal{H}} L(h) - L_S(h) \right] \;&=\; \mathop{\mathbb{E}}_{S} \left[\sup_{h \in \mathcal{H}} \mathop{\mathbb{E}}_{S'} [L_{S'}(h)] - L_S(h) \right] \\
&\leq\; \mathop{\mathbb{E}}_{S, S'} \left[\sup_{h \in \mathcal{H}} L_{S'}(h) - L_S(h) \right] \\
&=\; \mathop{\mathbb{E}}_{S, S'} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} (\ell(h, \mathbf{z}'_i) - \ell(h, \mathbf{z}_i)) \right] .
\end{aligned} \tag{25}$$

Next, we note that for each j, $\mathbf{z}_j$ and $\mathbf{z}'_j$ are i.i.d. variables. Therefore, we can replace them without affecting the expectation:

$$\begin{aligned}\mathop{\mathbb{E}}_{S,S'}\left[\sup_{h\in\mathcal{H}}\left(\ell(h,\mathbf{z}'_j)-\ell(h,\mathbf{z}_j)\right)+\sum_{i\neq j}\left(\ell(h,\mathbf{z}'_i)-\ell(h,\mathbf{z}_i)\right)\right] \;=\\ \mathop{\mathbb{E}}_{S,S'}\left[\sup_{h\in\mathcal{H}}\left(\ell(h,\mathbf{z}_j)-\ell(h,\mathbf{z}'_j)\right)+\sum_{i\neq j}\left(\ell(h,\mathbf{z}'_i)-\ell(h,\mathbf{z}_i)\right)\right].\end{aligned} \tag{26}$$

Let σ_j be a random variable such that $\mathbb{P}[\sigma_j=1]=\mathbb{P}[\sigma_j=-1]=1/2$. From Eq. (26) we obtain that

$$\begin{aligned}\mathop{\mathbb{E}}_{S,S',\sigma_j}\left[\sup_{h\in\mathcal{H}}\sigma_j\left(\ell(h,\mathbf{z}'_j)-\ell(h,\mathbf{z}_j)\right)+\sum_{i\neq j}\left(\ell(h,\mathbf{z}'_i)-\ell(h,\mathbf{z}_i)\right)\right] \;=\\ \mathop{\mathbb{E}}_{S,S'}\left[\sup_{h\in\mathcal{H}}\left(\ell(h,\mathbf{z}'_j)-\ell(h,\mathbf{z}_j)\right)+\sum_{i\neq j}\left(\ell(h,\mathbf{z}'_i)-\ell(h,\mathbf{z}_i)\right)\right].\end{aligned} \tag{27}$$

Repeating this for all j we obtain that

$$\mathop{\mathbb{E}}_{S,S'}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\left(\ell(h,\mathbf{z}'_i)-\ell(h,\mathbf{z}_i)\right)\right]=\mathop{\mathbb{E}}_{S,S',\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i\left(\ell(h,\mathbf{z}'_i)-\ell(h,\mathbf{z}_i)\right)\right].$$

The right-hand side of the above is at most

$$\mathop{\mathbb{E}}_{S,S',\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i\ell(h,\mathbf{z}'_i)+\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}(-\sigma_i)\ell(h,\mathbf{z}_i)\right].$$

Finally, since for any $\boldsymbol{\sigma}$ we have $\mathbb{P}[\boldsymbol{\sigma}]=\mathbb{P}[-\boldsymbol{\sigma}]$ we obtain that the above is at most $2\,\mathbb{E}_{S',\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i\ell(h,\mathbf{z}'_i)\right]$ and this concludes our proof. ■

As a corollary we obtain that in expectation the ERM rule generalizes well if the Rademacher complexity is low.

Corollary 4 *For each S, let h_S be an ERM hypothesis, $h_S\in\operatorname{argmin}_{h\in\mathcal{H}}L_S(h)$. Then,*

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[L(h_S)-L_S(h_S)\right]\leq 2\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}R(\ell\circ\mathcal{H}\circ S).$$

We can also easily obtain that the ERM rule finds a hypothesis which is close to the optimal hypothesis in $\mathcal{H}$.

Theorem 8 *For each S, let h_S be an ERM hypothesis, $h_S\in\operatorname{argmin}_{h\in\mathcal{H}}L_S(h)$. Let $h^\star$ be a minimizer of the generalization loss, $h^\star\in\operatorname{argmin}_{h\in\mathcal{H}}L(h)$. Then,*

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[L(h_S)-L(h^\star)\right]\;\leq\;2\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}R(\ell\circ\mathcal{H}\circ S).$$

Furthermore, for each $\delta\in(0,1)$ with probability of at least $1-\delta$ over the choice of S we have

$$L(h_S)-L(h^\star)\;\leq\;\frac{2\,\mathbb{E}_{S'\sim\mathcal{D}^m}R(\ell\circ\mathcal{H}\circ S')}{\delta}.$$

Proof The first inequality follows directly from the fact that

$$L(h^\star) = \mathop{\mathbb{E}}_{S}[L_S(h^\star)] \geq \mathop{\mathbb{E}}_{S}[L_S(h_S)] \; .$$

The second inequality follows from Markov inequality by noting that the random variable $L(h_S) - L(h^\star)$ is non-negative. ■

Next, we derive bounds similar to the bounds in Theorem 8 with a better dependence on the confidence parameter δ. To do so, we first introduce the following bounded differences concentration inequality.

Lemma 11 (McDiarmid's Inequality) *Let $V \subset \mathbb{R}$ and let $f : V^m \to \mathbb{R}$ be a function of m variables such that for some $c > 0$, for all $i \in [m]$ and for all $x_1, \ldots, x_m, x_i' \in V$ we have*

$$|f(x_1, \ldots, x_m) - f(x_1, \ldots, x_{i-1}, x', x_{i+1}, \ldots, x_m)| \leq c \; .$$

Let $X_1, \ldots, X_m$ be m independent random variables taking values in V. Then, with probability of at least $1 - \delta$ we have

$$|f(X_1, \ldots, X_m) - \mathbb{E}[f(X_1, \ldots, X_m)]| \leq c \sqrt{\ln\left(\tfrac{2}{\delta}\right) \; m/2} \; .$$

Based on McDiarmid inequality we can derive generalization bounds with a better dependence on the confidence parameter.

Theorem 9 *Assume the conditions stated in Theorem 8 hold. Assume also that for all $\mathbf{z}$ and $h \in \mathcal{H}$ we have that $|\ell(h, \mathbf{z})| \leq c$. Then,*

1. *With probability of at least $1 - \delta$*

$$L_{\mathcal{D}}(h_S) - L_S(h_S) \;\leq\; 2 \mathop{\mathbb{E}}_{S \sim D^m} R(\ell \circ \mathcal{H} \circ S) + c \sqrt{\frac{2 \ln(2/\delta)}{m}} \; .$$

2. *With probability of at least $1 - \delta$*

$$L_{\mathcal{D}}(h_S) - L_S(h_S) \;\leq\; 2 \, R(\ell \circ \mathcal{H} \circ S) + 2 \, c \sqrt{\frac{2 \ln(4/\delta)}{m}} \; .$$

3. *With probability of at least $1 - \delta$*

$$L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^\star) \;\leq\; 2 \, R(\ell \circ \mathcal{H} \circ S) + 3 \, c \sqrt{\frac{2 \ln(8/\delta)}{m}} \; .$$

Proof First note that the random variable $\sup_{h \in \mathcal{H}} L(h) - L_S(h)$ satisfies the bounded differences condition of Lemma 11 with a constant $2c/m$. Combining the bound in Lemma 11 with Lemma 10 we obtain the first inequality. For the second inequality we note that the random variable $R(\ell \circ \mathcal{H} \circ S)$ also satisfies the bounded differences condition of Lemma 11 with a constant $2c/m$. Therefore, the second inequality follows from the first inequality, Lemma 11, and the union bound. Finally, for the last inequality we use the fact that h_S is an ERM to get that

$$\begin{aligned} L(h_S) - L(h^\star) &= L(h_S) - L_S(h_S) + L_S(h_S) - L_S(h^\star) + L_S(h^\star) - L(h^\star) \\ &\leq (L(h_S) - L_S(h_S)) + (L_S(h^\star) - L(h^\star)) \; . \end{aligned} \tag{28}$$

The first summand is bounded by the second inequality in the theorem. For the second summand, we use the fact that $h^\star$ does not depend on S, hence by using Hoeffding's inequality we obtain that with probaility of at least $1 - \delta/2$,

$$L_S(h^\star) - L(h^\star) \;\leq\; c \sqrt{\frac{\ln(4/\delta)}{2m}} \; . \tag{29}$$

Combining the above with the union bound we conclude our proof. ■

The above theorem tells us that if the quantity $R(\ell \circ \mathcal{H} \circ S)$ is small then it is possible to learn the class $\mathcal{H}$ using the ERM rule. It is important to emphasize that the last two bounds given in the above theorem depend on the specific training set S. That is, we use S both for learning a hypothesis from $\mathcal{H}$ as well as for estimating the quality of it. This type of bound is called a *data-dependent bound*.

22.1 Rademacher and VC dimension

In the previous lecture we argued that a hypothesis class with a finite VC dimension is learnable. We are now ready to prove this theorem based on Theorem 9. The following lemma, due to Massart, states that the Rademacher complexity of a finite set grows logarithmically with the size of the set.

Lemma 12 (Massart lemma) *Let $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$ be a finite set of vectors taken from $\mathbb{R}^m$. Then,*

$$R(A) \leq \max_{\mathbf{a} \in A} \|\mathbf{a}\| \frac{\sqrt{2 \log(N)}}{m} .$$

Proof Let $\lambda > 0$ and let $A' = \{\lambda \mathbf{a}_1, \ldots, \lambda \mathbf{a}_N\}$. We upper bound the Rademacher complexity as follows (explanations follow)

$$\begin{aligned}
mR(A') &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \left[\max_{\mathbf{a} \in A'} \sum_{i=1}^{m} \sigma_i a_i \right] \\
&\leq \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \left[\log \left(\sum_{\mathbf{a} \in A'} \exp \left(\sum_{i=1}^{m} \sigma_i a_i \right) \right) \right] && (30) \\
&\leq \log \left(\mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \left[\sum_{\mathbf{a} \in A'} \exp \left(\sum_{i=1}^{m} \sigma_i a_i \right) \right] \right) && (31) \\
&= \log \left(\sum_{\mathbf{a} \in A'} \prod_{i=1}^{m} \mathop{\mathbb{E}}_{\sigma_i} \left[\exp \left(\sigma_i a_i \right) \right] \right) && (32) \\
&= \log \left(\sum_{\mathbf{a} \in A'} \prod_{i=1}^{m} \frac{\exp(a_i) + \exp(-a_i)}{2} \right) && (33) \\
&= \log \left(\sum_{\mathbf{a} \in A'} \prod_{i=1}^{m} \exp \left(\frac{a_i^2}{2} \right) \right) && (34) \\
&= \log \left(\sum_{\mathbf{a} \in A'} \exp \left(\|\mathbf{a}\|^2 / 2 \right) \right) && (35) \\
&\leq \log \left(|A'| \max_{\mathbf{a} \in A'} \exp \left(\|\mathbf{a}\|^2 / 2 \right) \right) && (36) \\
&= \log(|A'|) + \max_{\mathbf{a} \in A'} (\|\mathbf{a}\|^2 / 2) .
\end{aligned}$$

Eq. (30) is because for any sequence of numbers $x_1, \ldots, x_k$, the soft-max is larger than the max, $\log(\sum_i \exp(x_i)) \geq \max_i x_i$. Eq. (31) is an application of Jensen's inequality with the log function (which is concave). Eq. (32) follows from the independence of the Rademacher variables. Eq. (33) is from the definition of the expectation and Eq. (34) is due to the inequality $(e^a + e^{-a})/2 \leq e^{a^2}/2$ which holds for all $a \in \mathbb{R}$. Eq. (35) is from the definition of the Euclidean norm. Finally, Eq. (36) is because a sum of k numbers is always at most k times the largest number.

Since $R(A) = \frac{1}{\lambda} R(A')$ we obtain from the above that

$$R(A) = \frac{\log(|A|) + \lambda^2 \max_{\mathbf{a} \in A} (\|\mathbf{a}\|^2/2)}{\lambda m} \ .$$

Setting $\lambda = \sqrt{2 \log(|A|) / \max_{\mathbf{a} \in A} \|\mathbf{a}\|^2}$ and rearranging terms we conclude our proof. ■

Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a classification training set. Recall that Sauer-Shelah lemma tells us that if $\text{VCdim}(\mathcal{H}) = d$ then

$$|\{(h(\mathbf{x}_1), \ldots, h(\mathbf{x}_m)) : h \in \mathcal{H}\}| \ \leq \ \left(\frac{e\, m}{d}\right)^d \ .$$

Clearly, this also implies that

$$\left|\{(\mathbb{1}_{[h(\mathbf{x}_1) \neq y_1]}, \ldots, \mathbb{1}_{[h(\mathbf{x}_m) \neq y_m]}) : h \in \mathcal{H}\}\right| \ \leq \ \left(\frac{e\, m}{d}\right)^d \ .$$

Combining the above with Lemma 12 and Theorem 9 we get the following:

Theorem 10 *Let $\mathcal{H}$ be a set of binary classifiers with $\text{VCdim}(\mathcal{H}) = d < \infty$. Let S be a training set of m i.i.d. samples from a distribution $\mathcal{D}$. Let $h_S \in \mathcal{H}$ be a classifier that minimizes the training error and let $h^\star \in \mathcal{H}$ be a classifier that minimizes the generalization error. Then, with probability of at least $1 - \delta$ over the choice of S we have*

$$\text{err}_{\mathcal{D}}(h_S) - \text{err}_{\mathcal{D}}(h^\star) \leq O\left(\sqrt{\frac{d \log(m/\delta)}{m}}\right) \ .$$

The above theorem concludes the proof of Theorem 7, namely, it proves that a class with a finite VC dimension is learnable in the agnostic PAC model.

22.2 Rademacher Calculus

Let us now discuss some properties of the Rademacher complexity measure. These properties will help us in deriving some simple bounds on $R(\ell \circ \mathcal{H} \circ S)$ for specific cases of interest.

The following lemma is immediate from the definition.

Lemma 13 *For any $A \subset \mathbb{R}^m$ and scalar c we have $R(\{c\, \mathbf{a} : \mathbf{a} \in A\}) \leq |c|\, R(A)$.*

The following lemma tells us that the convex hull of A has the same complexity as A.

Lemma 14 *Let A be a subset of $\mathbb{R}^m$ and let $A' = \{\sum_{j=1}^N \alpha_j \mathbf{a}^{(j)} : N \in \mathbb{N}, \forall j, \mathbf{a}^{(j)} \in A, \|\boldsymbol{\alpha}\|_1 \leq 1\}$. Then, $R(A') = R(A)$.*

Proof The main idea follows from the fact that for any vector $\mathbf{v}$ we have

$$\sup_{\boldsymbol{\alpha}: \|\boldsymbol{\alpha}\|_1 \leq 1} \sum_{j=1}^N \alpha_j v_j \ = \ \max_j |v_j| \ .$$

Therefore,

$$\begin{aligned}
m\, R(A') &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \sup_{\|\boldsymbol{\alpha}\|_1 \leq 1} \sup_{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(N)}} \sum_{i=1}^m \sigma_i \sum_{j=1}^N \alpha_j a_i^{(j)} \\
&= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \sup_{\|\boldsymbol{\alpha}\|_1 \leq 1} \sum_{j=1}^N \alpha_j \sup_{\mathbf{a}^{(j)}} \sum_{i=1}^m \sigma_i a_i^{(j)} \\
&= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \sup_{\mathbf{a} \in A} \sum_{i=1}^m \sigma_i a_i \\
&= m\, R(A) \ .
\end{aligned}$$

■

The following lemma shows that composing A with a Lipschitz function does not blow up the Rademacher complexity. The proof is due to Kakade and Tewari.

Lemma 15 (Contraction lemma) *For each $i \in [m]$, let $\phi_i : \mathbb{R} \to \mathbb{R}$ be a ρ-Lipschitz function, namely for all $\alpha, \beta \in \mathbb{R}$ we have $|\phi_i(\alpha) - \phi_i(\beta)| \leq \rho\,|\alpha - \beta|$. For $\mathbf{a} \in \mathbb{R}^m$ let $\boldsymbol{\phi}(\mathbf{a})$ denote the vector $(\phi_1(a_1), \ldots, \phi_m(y_m))$. Let $\boldsymbol{\phi} \circ A = \{\boldsymbol{\phi}(\mathbf{a}) : a \in A\}$. Then,*

$$R(\boldsymbol{\phi} \circ A) \leq \rho\, R(A)\ .$$

Proof For simplicity, we prove the lemma for the case $\rho = 1$. The case $\rho \neq 1$ will follow by defining $\phi' = \frac{1}{\rho}\phi$ and then using Lemma 13. Let $A_i = \{(a_1, \ldots, a_{i-1}, \phi_i(a_i), a_{i+1}, \ldots, a_m) : \mathbf{a} \in A\}$. Clearly, it suffices to prove that for any set A and all i we have $R(A_i) \leq R(A)$. Without loss of generality we will prove the latter claim for $i = 1$ and to simplify notation we omit the underscript from ϕ_1. We have

$$\begin{aligned}
mR(A_1) &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{a}\in A_1}\sum_{i=1}^m \sigma_i a_i\right] \qquad (37)\\
&= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{a}\in A}\sigma_1\phi(a_1) + \sum_{i=2}^m \sigma_i a_i\right]\\
&= \frac{1}{2}\mathop{\mathbb{E}}_{\sigma_2,\ldots,\sigma_m}\left[\sup_{\mathbf{a}\in A}\left(\phi(a_1) + \sum_{i=2}^m \sigma_i a_i\right) + \sup_{\mathbf{a}\in A}\left(-\phi(a_1) + \sum_{i=2}^m \sigma_i a_i\right)\right]\\
&= \frac{1}{2}\mathop{\mathbb{E}}_{\sigma_2,\ldots,\sigma_m}\left[\sup_{\mathbf{a},\mathbf{a}'\in A}\left(\phi(a_1) - \phi(a_1') + \sum_{i=2}^m \sigma_i a_i + \sum_{i=2}^m \sigma_i a_i'\right)\right]\\
&\leq \frac{1}{2}\mathop{\mathbb{E}}_{\sigma_2,\ldots,\sigma_m}\left[\sup_{\mathbf{a},\mathbf{a}'\in A}\left(|a_1 - a_1'| + \sum_{i=2}^m \sigma_i a_i + \sum_{i=2}^m \sigma_i a_i'\right)\right]\ ,
\end{aligned}$$

where in the last inequality we used the assumption that ϕ is Lipschitz. Next, we note that the absolute value on $|a_1 - a_1'|$ in the above expression can be omitted since both $\mathbf{a}$ and $\mathbf{a}'$ are from the same set A and the rest of the expression in the supremum is not affected from replacing $\mathbf{a}$ and $\mathbf{a}'$. Therefore,

$$mR(A_1) \leq \frac{1}{2}\mathop{\mathbb{E}}_{\sigma_2,\ldots,\sigma_m}\left[\sup_{\mathbf{a},\mathbf{a}'\in A}\left(a_1 - a_1' + \sum_{i=2}^m \sigma_i a_i + \sum_{i=2}^m \sigma_i a_i'\right)\right]\ . \qquad (38)$$

But, using the same equalities as in Eq. (37) it is easy to see that the right-hand side of Eq. (38) exactly equals to $m\,R(A)$, which concludes our proof. ■

(67577) Introduction to Machine Learning November 23, 2009

Lecture 8(a) – Linear Separators (Halfspaces)

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the following lectures we will study the hypothesis class of linear separators (a.k.a. Halfspaces). Some of the most important machine learning tools are based on learning Halfspaces. Examples include the Perceptron, Support Vector Machines, and AdaBoost. The class of Halfspaces is defined over the instance space $\mathcal{X} = \mathbb{R}^n$ as follows:

$$\mathcal{H} = \{\mathbf{x} \mapsto \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) : \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}\} ,$$

where $\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{j=1}^n w_j x_j$ is the inner-product between the vectors $\mathbf{w}$ and $\mathbf{x}$. We call the vector $\mathbf{w}$ a weight vector and the scalar b a bias. We sometimes discuss unbiased Halfspaces, namely, the case $b = 0$. This is justifiable because we can augment $\mathbf{x}$ with a constant coordinate equals 1 and then learning a Halfspace is equivalent to learning unbiased Halfspaces with the additional constant coordinate. An illustration of a Halfspace in $\mathbb{R}^2$ is given in Figure 4.

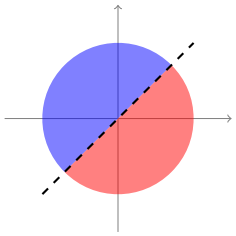


Figure 4: A Halfspace in $\mathbb{R}^2$ corresponding to the weight vector $\mathbf{w} = (-1, 1)$ and bias $b = 0$.

23 Learning Halfspaces

We now discuss the problem of learning Halfspaces. As we will show in the next section, the VC dimension of the class of Halfspaces in $\mathbb{R}^n$ is $n + 1$. Therefore, we can learn this class using the ERM rule. That is, given a training set S, a learning algorithm can set the Halfspace to be

$$\operatorname*{argmin}_{h \in \mathcal{H}} \text{err}_S(h) .$$

We distinguish between two cases.

23.1 Separable training sets

A training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ is called separable if there exists a halfspace $h \in \mathcal{H}$ that perfectly explains S, namely, for all $(\mathbf{x}, y) \in S$ we have that $h(\mathbf{x}) = y$. In this case, the ERM problem can be solved efficiently using *linear programming* as follows.

Note that we seek a halfspace h, parametrized by a vector $\mathbf{w}$ and scalar b, such that $h(\mathbf{x}_i) = y_i$ for all i, that is,

$$\forall i, \quad \text{sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = y_i .$$

Equivalently, we would like to solve the following problem:

$$\text{Find } \mathbf{w} \text{ and } b \text{ such that: } \forall i, \quad y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) > 0 \; . \tag{39}$$

This is a set of linear inequalities in $\mathbf{w}$ and b and therefore is an instance of a generic linear programming problem. There are algorithms that solve linear programming in polynomial time and therefore the problem given in Eq. (39) can be solved using an off-the-shelf algorithm. We will discuss other solutions later in the course.

23.2 Non-Separable training sets

The non-separable case is when no halfspace perfectly explains the entire training set. That is, for any halfspace, there exists some example in the training set such that $h(\mathbf{x}_i) \neq y_i$. Solving the ERM problem in this case is known to be NP hard even to approximate. We deal with this problem using the notion of surrogate loss functions, that will be learned in later lectures.

24 The VC dimension of Halfspaces

In this section we calculate the VC dimension of $\mathcal{H}$. First, we recall the definition of convex hull and prove Radon's lemma.

Definition 7 *A set $A \subseteq \mathbb{R}^n$ is* convex *if for every pair of points $\mathbf{u}, \mathbf{v} \in A$ all the line from $\mathbf{u}$ to $\mathbf{v}$ is in A, namely, $\{\mathbf{u} + \lambda(\mathbf{v} - \mathbf{u}) : \lambda \in [0, 1]\} \subseteq A$. The* convex hull *of $A = \{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ (denoted $\mathrm{conv}(A)$) is*

$$\mathrm{conv}(A) = \left\{ \sum_{i=1}^{m} \lambda_i \mathbf{u}_i : \sum_{i=1}^{m} \lambda_i = 1 \wedge \forall i, \lambda_i \geq 0 \right\} \; .$$

It is possible to show that $\mathrm{conv}(A)$ is the smallest convex set that contains A. Note that a halfspace, $\{\mathbf{x} : h(\mathbf{x}) = 1\}$, as well as its complement, $\{\mathbf{x} : h(\mathbf{x}) = 0\}$, are convex sets.

Lemma 16 (Radon) *For every set $A \subseteq \mathbb{R}^n$, if $|A| \geqslant n + 2$, then there is a subset $B \subseteq A$ such that*

$$\mathrm{conv}(B) \; \cap \; \mathrm{conv}(A \setminus B) \; \neq \; \emptyset.$$

Proof We can assume that $|A| = n + 2$, because if some $A' \subset A$ satisfies the lemma, then A also does. Let $A = \{\mathbf{u}_1, \ldots, \mathbf{u}_{n+2}\}$ and let $\tilde{A} = \{\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_{n+2}\}$, where for all i the vector $\tilde{\mathbf{u}}_i$ is the concatenation of the vector $\mathbf{u}_i$ and the scalar 1. The set $\tilde{A}$ contains $n + 2$ vectors in $\mathbb{R}^{n+1}$ and is therefore a set of linearly dependent vector, which means that there exist a vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{n+2}) \neq \mathbf{0}$ such that

$$\sum_{i=1}^{n+2} \mu_i \, \tilde{\mathbf{u}}_i = (0, \ldots, 0) \; .$$

Without loss of generality, assume that $\mu_1, \ldots, \mu_k > 0$ and $\mu_{k+1}, \ldots, \mu_{n+2} \leq 0$. Note that from the definition of $\tilde{\mathbf{u}}_i$ we have that

$$\sum_{i=1}^{k} \mu_i = \sum_{i=k+1}^{n+2} (-\mu_i) \; ,$$

which implies that $k \geq 1$ (because otherwise, $\boldsymbol{\mu}$ would have been all zeros). Choose $B = \{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$. We have,

$$\sum_{i=1}^{k} \mu_i \mathbf{u}_i = \sum_{i=k+1}^{n+2} (-\mu_i) \mathbf{u}_i \; .$$

Dividing both sides by $\sum_{i=1}^{k} \mu_i$ we obtain that the left-hand side is in $\mathrm{conv}(B)$ and the right-hand side is in $\mathrm{conv}(A \setminus B)$, which concludes our proof. ■

Based on Radon's lemma, we can calculate the VC dimension of halfspaces.

Theorem 11 *Let $\mathcal{H}$ be the hypothesis class of halfspaces in $\mathbb{R}^n$. Then,*

$$\mathrm{VCdim}(\mathcal{H}) \;=\; n+1.$$

Proof First, it is easy to show that the set of points $\{\mathbf{0}, \mathbf{e}_1, \ldots, \mathbf{e}_n\}$, where $\mathbf{e}_i$ is the all zeros vector except 1 in the i'th element, is shattered by $\mathcal{H}$. Suppose that there exists A such that $|A| \geqslant n+2$ and A is shattered by $\mathcal{H}$. Pick B as in Radon's lemma; since A is shattered, there is a halfspace $h \in \mathcal{H}$ such that $\{\mathbf{x} \in A : h(\mathbf{x}) = 1\} = B$. Of course, it is also true that $\{\mathbf{x} \in A : h(\mathbf{x}) = 0\} = A \setminus B$. By Radon's lemma, there is a point $\mathbf{x} \in \mathrm{conv}(B) \cap \mathrm{conv}(A \setminus B)$. The halfspace defined by h is a convex set containing B, so $\mathbf{x} \in \mathrm{conv}(B) \subseteq \{\mathbf{x}' : h(\mathbf{x}') = 1\}$. Similarly, $\mathbf{x} \in \mathrm{conv}(A \setminus B) \subseteq \{\mathbf{x}' : h(\mathbf{x}') = 0\}$, but this leads to a contradiction. ■

(67577) Introduction to Machine Learning November 30, 2009

Lecture 8(b) – Margin and Fuzzy Halfspaces

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the previous lecture we learned about the class of Halfspaces. We saw that the VC dimension of Halfspaces grows with the dimension. In this lecture we will learn about a related class which we call fuzzy Halfspaces. Intuitively, fuzzy Halfspaces formalize the intuitive idea that a Halfspace predictor is less sure about the correct labels of points withing a small *margin* of the separating Hyperplane, that is points which are very close to the decision boundary.

25 Fuzzy Halfspaces and ℓ_p margin

Recall that a Halfspace is a function $\mathbf{x} \mapsto \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$. The sign function is not continuous, and therefore the prediction changes immediately when we pass the halfspace defined by $\mathbf{w}$. This means that a slight perturbation of $\mathbf{x}$ might flip the prediction. Such a non-robust behavior might be inadequate for real-world data. In this section we shall describe a *fuzzy Halfspace* – a classifier that changes its predictions more gradually.

Similarly to a regular Halfspace, a fuzzy Halfspace is parametrized by a weight vector $\mathbf{w}$ and the prediction associated with an instance $\mathbf{x}$ is based on the inner product $\langle \mathbf{w}, \mathbf{x} \rangle$. If $\mathbf{x}$ is far from the hyperplane defined by $\mathbf{w}$, i.e. $|\langle \mathbf{w}, \mathbf{x} \rangle| > \gamma$, for some parameter γ, then the prediction of the fuzzy Halfspace is identical to the prediction of a regular Halfspace, that is $h(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$. However, when $\mathbf{x}$ is close to the hyperplane, the fuzzy Halfspace is less sure about the label of $\mathbf{x}$ and it therefore changes its mind gradually from a negative prediction to a positive prediction. The uncertainty of the fuzzy Halfspace is formalized by outputting a random prediction. Formally, a fuzzy halfspace is a *probabilistic classifier* where $\mathbb{P}[h(\mathbf{x}) = 1] = \tau_\gamma(\langle \mathbf{w}, \mathbf{x} \rangle)$ where $\tau_\gamma : \mathbb{R} \to \mathbb{R}$ is the function

$$\tau_\gamma(a) = \begin{cases} 1 & \text{if } a \geq \gamma \\ 0 & \text{if } a \leq -\gamma \\ \frac{1+a/\gamma}{2} & \text{otherwise} \end{cases} , \tag{40}$$

and γ is called a *margin* parameter. See illustration in Figure 5. We refer to the ares close to the decision

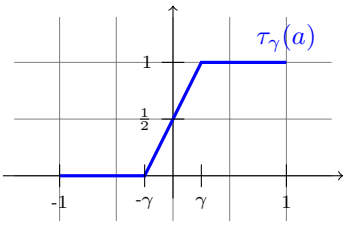


Figure 5: An illustration of the function τ_γ defined in Eq. (40).

boundary as the *margin* of the hyperplane (see Figure 6).

To predict the label of $\mathbf{x}$, the probabilistic classifier h flips a coin whose bias is $\tau_\gamma(\langle \mathbf{w}, \mathbf{x} \rangle)$. If $\mathbf{x}$ is outside a margin γ of the hyperplane defined by $\mathbf{w}$ then $\tau_\gamma(\langle \mathbf{w}, \mathbf{x} \rangle)$ is either 0 or 1, which means that h will predict the label deterministically to be $\text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$. However, when $\mathbf{x}$ starts to get closer to the hyperplane (enters the margin area), we start to be less sure about the prediction and therefore we flip a coin whose bias gradually

changes from 0 to 1. When $\mathbf{x}$ is exactly on the decision boundary, we really don't know what to predict so we should guess a label according to an unbiased coin, and indeed $\tau_\gamma(0) = 1/2$.

We define the loss of a fuzzy halfspace on a labeled example $(\mathbf{x}, y)$ to be the probability of error, where the probability is over the randomness in choosing the prediction. That is, the loss of a fuzzy Halfspace predictor parametrized by $\mathbf{w}$ is

$$\ell(h, (\mathbf{x}, y)) \;=\; \mathop{\mathbb{P}}_{h_{\mathbf{w}}(\mathbf{x}) \sim \tau_\gamma(\langle \mathbf{w}, \mathbf{x} \rangle)} [h_{\mathbf{w}}(\mathbf{x}) \neq y] \;=\; \left| \tau_\gamma(\langle \mathbf{w}, \mathbf{x} \rangle) - \tfrac{y+1}{2} \right| . \tag{41}$$

Note that $\ell(h, (\mathbf{x}, y)) = 0$ iff both $\text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle) = y$ and $|\langle \mathbf{w}, \mathbf{x} \rangle| \geq \gamma$. Given a training set S, a fuzzy Halfspace will have a zero training error if all points in the training set will be on the right side of the hyperplane and furthermore no instances will be within the margin area. In such a case we say that the training set is *separated with a margin* γ.

Naturally, since τ_γ depends on the magnitude of $\langle \mathbf{w}, \mathbf{x} \rangle$, we must normalize $\mathbf{w}$ (otherwise, we can increase the norm of $\mathbf{w}$ to infinity, thus making the fuzzy halfspace behave exactly as a regular halfspace). For the same reason, we also need to normalize $\mathbf{x}$. Two types of normalization are widely used.

25.1 ℓ_2 margin

In ℓ_2 margin we normalize $\mathbf{w}$ to have a unit ℓ_2 norm. The ℓ_2 norm of an n-dimensional vector is:

$$\|\mathbf{w}\|_2 \;=\; \sqrt{\sum_{j=1}^{n} w_j^2} .$$

The class of fuzzy Halfspaces with respect to ℓ_2 margin is therefore:

$$\mathcal{H}_{\gamma,2} \;=\; \{\mathbf{x} \mapsto h_{\mathbf{w}}(\mathbf{x}) : \|\mathbf{w}\|_2 = 1,\; \mathbb{P}[h_{\mathbf{w}}(\mathbf{x}) = 1] = \tau_\gamma(\langle \mathbf{w}, \mathbf{x} \rangle)\} .$$

To ensure that $\langle \mathbf{w}, \mathbf{x} \rangle \in [-1, +1]$ we will also assume that the ℓ_2 norm of each instance is bounded by 1.

25.2 ℓ_1 margin

In ℓ_1 margin we normalize $\mathbf{w}$ to have a unit ℓ_1 norm. The ℓ_1 norm of an n-dimensional vector is:

$$\|\mathbf{w}\|_1 \;=\; \sum_{j=1}^{n} |w_j| .$$

The class of fuzzy Halfspaces with respect to ℓ_1 margin is therefore:

$$\mathcal{H}_{\gamma,1} \;=\; \{\mathbf{x} \mapsto h_{\mathbf{w}}(\mathbf{x}) : \|\mathbf{w}\|_1 = 1,\; \mathbb{P}[h_{\mathbf{w}}(\mathbf{x}) = 1] = \tau_\gamma(\langle \mathbf{w}, \mathbf{x} \rangle)\} .$$

To ensure that $\langle \mathbf{w}, \mathbf{x} \rangle \in [-1, +1]$ we will also assume that the ℓ_∞ norm of each instance is bounded by 1, where ℓ_∞ is defined as:

$$\|\mathbf{x}\|_\infty \;=\; \max_j |x_j| .$$

An illustration of a fuzzy halfspace with respect to ℓ_2 and ℓ_1 margin is given in Figure 6.

26 Generalization bounds for fuzzy Halfspaces

In this section we analyze the Rademacher complexity of fuzzy Halfspaces. As we have shown in previous lectures, a bound on the Rademacher complexity immediately translates into a generalization bound.

To simplify the derivation we first define the following two classes:

$$\mathcal{H}_1 = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle : \|\mathbf{w}\|_1 \leq 1\} \quad , \quad \mathcal{H}_2 = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle : \|\mathbf{w}\|_2 \leq 1\} . \tag{42}$$

The following lemma bounds the Rademacher complexity of $\mathcal{H}_1 \circ S$.

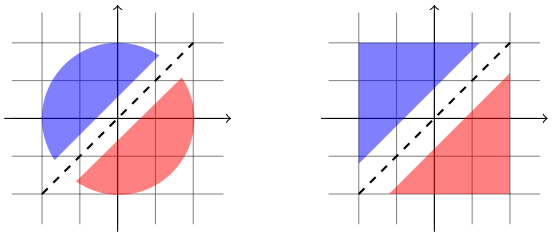


Figure 6: An illustration of fuzzy halfspaces with $\gamma = 0.2$. Left: ℓ_2 margin. Right: ℓ_1 margin.

Lemma 17 *Let $S = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ be vectors in $\mathbb{R}^n$. Then,*

$$R(\mathcal{H}_1 \circ S) \;\leq\; \max_i \|\mathbf{x}_i\|_\infty \sqrt{\frac{2\log(n)}{m}}\,.$$

Proof For each $j \in [n]$, let $\mathbf{v}_j = (x_{1,j}, \ldots, x_{m,j}) \in \mathbb{R}^m$. Note that $\|\mathbf{v}_j\|_2 \leq \sqrt{m}\,\max_i \|\mathbf{x}_i\|_\infty$. Let $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$. Using Massart lemma (Lemma 12) we have that $R(V) \leq \max_i \|\mathbf{x}_i\|_\infty \sqrt{2\log(n)/m}$. Next, we note that for each vector $\mathbf{a} \in \mathcal{H}_1 \circ S$, there exists $\mathbf{w} \in \mathbb{R}^n$, $\|\mathbf{w}\|_1 \leq 1$, such that $\mathbf{a} = \sum_{j=1}^n w_j \mathbf{v}_j$. Therefore, using Lemma 14 we conclude our proof. ■

Next we bound the Rademacher complexity of $\mathcal{H}_2$. In the following lemma, we allow the $\mathbf{x}_i$ to be vectors in any Hilbert space (even infinite dimensional), and the bound does not depend on the dimensionality of the Hilbert space. This property will become useful later when we will introduce kernel methods.

Lemma 18 *Let $S = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ be vectors in a Hilbert space. Define: $\mathcal{H}_2 \circ S = \{(\langle \mathbf{w}, \mathbf{x}_1\rangle, \ldots, \langle \mathbf{w}, \mathbf{x}_m\rangle) : \|\mathbf{w}\|_2 \leq 1\}$. Then,*

$$R(\mathcal{H}_2 \circ S) \;\leq\; \frac{\max_i \|\mathbf{x}_i\|_2}{\sqrt{m}}\,.$$

Proof Using Cauchy-Schwartz inequality we know that for any vectors $\mathbf{w}, \mathbf{v}$ we have $\langle \mathbf{w}, \mathbf{v}\rangle \leq \|\mathbf{w}\|\,\|\mathbf{v}\|$. Therefore,

$$\begin{aligned} m R(\mathcal{H}_2 \circ S) &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{a}\in A_2} \sum_{i=1}^m \sigma_i a_i\right] \\ &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{w}:\|\mathbf{w}\|\leq 1} \sum_{i=1}^m \sigma_i \langle \mathbf{w}, \mathbf{x}_i\rangle\right] \\ &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{w}:\|\mathbf{w}\|\leq 1} \langle \mathbf{w}, \sum_{i=1}^m \sigma_i \mathbf{x}_i\rangle\right] \\ &\leq \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\|\sum_{i=1}^m \sigma_i \mathbf{x}_i\|_2\right]\,. \end{aligned} \tag{43}$$

Next, using Jensen's inequality we have that

$$\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\|\sum_{i=1}^m \sigma_i \mathbf{x}_i\|_2\right] \;=\; \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[(\|\sum_{i=1}^m \sigma_i \mathbf{x}_i\|_2^2)^{1/2}\right] \leq \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\|\sum_{i=1}^m \sigma_i \mathbf{x}_i\|_2^2\right]^{1/2}\,. \tag{44}$$

Finally, since the variables $\sigma_1, \ldots, \sigma_m$ are independent we have

$$\begin{aligned}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\|\sum_{i=1}^{m}\sigma_i\mathbf{x}_i\|_2^2\right] &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sum_{i,j}\sigma_i\sigma_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle\right] \\ &= \sum_{i\neq j}\langle\mathbf{x}_i,\mathbf{x}_j\rangle\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sigma_i\sigma_j\right] + \sum_{i=1}^{m}\langle\mathbf{x}_i,\mathbf{x}_i\rangle\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sigma_i^2\right] \\ &= \sum_{i=1}^{m}\|\mathbf{x}_i\|_2^2 \;\leq\; m\;\max_i\|\mathbf{x}_i\|_2^2\;.\end{aligned}$$

Combining the above with Eq. (43) and Eq. (44) we conclude our proof. ■

Equipped with the above lemmas we are ready to bound the Rademacher complexity of $\mathcal{H}_{\gamma,1}$ and $\mathcal{H}_{\gamma,2}$.

Theorem 12 *Let $\mathcal{H}_{\gamma,1}$ and $\mathcal{H}_{\gamma,2}$ be classes of fuzzy Halfspaces with respect to ℓ_1 and ℓ_2 margin and let ℓ be the loss function given in Eq. (41). Let S be a training set of m examples. Then:*

$$R(\ell\circ\mathcal{H}_{\gamma,1}\circ S) \;\leq\; \frac{\max_i\|\mathbf{x}_i\|_\infty\sqrt{\log(n)}}{\gamma\sqrt{2\,m}} \tag{45}$$

$$R(\ell\circ\mathcal{H}_{\gamma,2}\circ S) \;\leq\; \frac{\max_i\|\mathbf{x}_i\|_2}{2\,\gamma\sqrt{m}} \tag{46}$$

Proof We can rewrite each vector in $\ell\circ\mathcal{H}_{\gamma,1}\circ S$ as $(g(\langle\mathbf{w},\mathbf{x}_1\rangle),\ldots,g(\langle\mathbf{w},\mathbf{x}_m\rangle))$ where $g(a) = |\tau_\gamma(a) - \frac{y+1}{2}|$. Since the function g is $1/(2\gamma)$-Lipschitz the proof follows directly from Lemmas 17-18 using the contraction lemma (Lemma 15). ■

27 Learning fuzzy Halfspaces

In the previous section we derived bounds on the Rademacher complexity of fuzzy Halfspaces and therefore we can learn fuzzy Halfspaces using the ERM principle. This amounts to solving the optimization problem:

$$\min_{\mathbf{w}:\|\mathbf{w}\|_p=1}\;\frac{1}{m}\sum_{i=1}^{m}\left|\tau_\gamma(\langle\mathbf{w},\mathbf{x}\rangle) - \tfrac{y+1}{2}\right|\;, \tag{47}$$

where p is 1 for ℓ_1 margin and 2 for ℓ_2 margin.

Solving the optimization problem in Eq. (47) in the general case seems to be difficult because of the non-convexity of the objective function. However, in the realizable case, in which some $\mathbf{w}$ achieves zero training loss, the problem can be solved efficiently as we show next.

27.1 Separability with margin

Consider a training set for which Eq. (47) is zero and let $\mathbf{w}^\star$ be an ERM. This implies that $\|\mathbf{w}^\star\|_p = 1$ and

$$\forall i\in[m],\;\; \text{sign}(\langle\mathbf{w}^\star,\mathbf{x}_i\rangle) = y_i \wedge |\langle\mathbf{w}^\star,\mathbf{x}_i\rangle| \geq \gamma\;. \tag{48}$$

We can rewrite Eq. (48) more compactly as

$$\forall i\in[m],\;\; y_i\langle\mathbf{w}^\star,\mathbf{x}_i\rangle \geq \gamma\;. \tag{49}$$

A training set that satisfies Eq. (49) is called *separable with margin* γ, since all the instances are classified correctly and there are no instances inside a margin of γ around the decision boundary. So, to solve the ERM problem we need to find a *unit* vector $\mathbf{w}^\star$ that satisfies Eq. (49). Below we show that this can be done in polynomial time.

Consider the following optimization problem, in which instead of finding a vector with a unit norm we look for a vector with minimal norm:

$$\min_{\mathbf{w}} \|\mathbf{w}\|_p \quad \text{s.t.} \quad \forall i \in [m], \;\; y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq \gamma \; . \tag{50}$$

For $p = 1$ the above optimization problem can be solved efficiently using linear programming and for $p = 2$ the problem can be solved efficiently using quadratic programming. The set of constraints in Eq. (50) is non-empty because we assume that $\mathbf{w}^\star$ satisfies the constraints. Let $\mathbf{w}_0$ be an optimum of Eq. (50). Clearly, $\|\mathbf{w}_0\|_p \leq \|\mathbf{w}^\star\|_p = 1$. We now argue that $\hat{\mathbf{w}}_0 = \mathbf{w}_0 / \|\mathbf{w}_0\|_p$ is an ERM. Indeed, $\|\hat{\mathbf{w}}_0\|_p = 1$ and from the linearity of inner products we have that for all $i \in [m]$

$$y_i \langle \hat{\mathbf{w}}_0, \mathbf{x}_i \rangle = \tfrac{1}{\|\mathbf{w}_0\|_p} \, y_i \langle \mathbf{w}_0, \mathbf{x}_i \rangle \geq \frac{\gamma}{\|\mathbf{w}_0\|_p} \; . \tag{51}$$

But, since $1/\|\mathbf{w}_0\|_p \geq 1$ we get that $y_i \langle \hat{\mathbf{w}}_0, \mathbf{x}_i \rangle \geq \gamma$ as required. In summary, we have shown that by solving Eq. (50) and normalizing the solution to have a unit ℓ_p norm we are guaranteed to find an ERM.

27.2 Max Margin

So far we assumed that the margin parameter γ is set in advance (in the terminology we used in previous lectures, this is part of our prior knowledge about the problem). In practice, however, it is preferable to set γ automatically, based on the specific data we have. A general method for tuning parameters is called validation. We shall learn about validation later in the course. Luckily, in our specific case, the parameter γ can be tuned automatically.

Suppose that we have a training set of examples which are linearly separable with margin $\gamma^\star$, but we do not know the value of $\gamma^\star$. Consider the optimization problem:

$$\min_{\mathbf{w}} \|\mathbf{w}\|_p \quad \text{s.t.} \quad \forall i \in [m], \;\; y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1 \; , \tag{52}$$

where, again, p is either 1 or 2. Let $\mathbf{w}_0$ be a solution of Eq. (52) and define

$$\gamma = \frac{1}{\|\mathbf{w}_0\|_p} \quad ; \quad \hat{\mathbf{w}} = \gamma \, \mathbf{w}_0 \; . \tag{53}$$

It is easy to verify that $\hat{\mathbf{w}}$ is an ERM with respect to the margin parameter γ and that $\gamma \geq \gamma^\star$. Therefore, $\hat{\mathbf{w}}$ is also an ERM with respect to the margin parameter $\gamma^\star$. In fact, $\hat{\mathbf{w}}$ is a *max margin* solution, i.e. no weight vector separates the training set with a margin larger than that of $\hat{\mathbf{w}}$.

27.3 Structural Risk Minimization (SRM)

We saw that solving Eq. (52) produces a max margin solution. We now show a generalization bound for the fuzzy Halfspace parametrized by the vector $\hat{\mathbf{w}}$ and the margin γ (as defined in Eq. (53)). For simplicity, we focus on ℓ_2 margin, but the analysis below can be easily adapted to ℓ_1 margin as well.

For each $i = 1, 2, \ldots$ let $\gamma_i = 2^{-i}$ and let $\hat{h}_i$ be the fuzzy Halfspace associated with $\hat{\mathbf{w}}$ and γ_i. Assume that the distribution $\mathcal{D}$ over pairs $(\mathbf{x}, y)$ is such that $\|\mathbf{x}\|_2 \leq 1$ with probability 1. Let $\delta_i = \delta \, 2^{-i}$ for some fixed $\delta \in (0, 1)$. Combining Theorem 12 with Theorem 9 we obtain that with probability of at least $1 - \delta_i$

$$\begin{aligned} L_{\mathcal{D}}(\hat{h}_i) \;&\leq\; L_S(\hat{h}_i) + 2\, R(\ell \circ \mathcal{H}_{\gamma_i, 2} \circ S) + \sqrt{\frac{8 \, \ln(4/\delta_i)}{m}} \\ &\leq\; L_S(\hat{h}_i) + \frac{1}{\gamma_i \sqrt{m}} + \sqrt{\frac{8 \, (\ln(4/\delta) + i \ln(2))}{m}} \; . \end{aligned} \tag{54}$$

Using the union bound and the fact that $\sum_{i=1}^{\infty} \delta_i = \delta$ we obtain from the above that with probability of at least $1-\delta$

$$\forall i, \quad L_{\mathcal{D}}(\hat{h}_i) \;\leq\; L_S(\hat{h}_i) + \frac{1}{\gamma_i \sqrt{m}} + \sqrt{\frac{8\left(\ln(4/\delta) + i\ln(2)\right)}{m}} \,. \tag{55}$$

In particular, the above holds for $i = \lceil -\log_2(\gamma) \rceil$. Since for this value of i we have that $L_S(\hat{h}_i) = 0$ and $1/\gamma_i = 2^i \leq 2^{-\log_2(\gamma)+1} = 2/\gamma$ we obtain the following:

Corollary 5 *Let $\mathcal{D}$ be a distribution over pairs $(\mathbf{x}, y)$ such that $\|\mathbf{x}\|_2 \leq 1$ with probability 1. Let S be a training set and let $\hat{\mathbf{w}}, \gamma$ be as defined in Eq. (53). Let $\hat{h}$ be a fuzzy Halfspace associated with $\hat{\mathbf{w}}$ and $2^{-\lceil \log_2(1/\gamma) \rceil}$. Then, with probability of at least $1-\delta$ we have*

$$L_{\mathcal{D}}(\hat{h}) \;\leq\; \frac{2}{\gamma\sqrt{m}} + \sqrt{\frac{8\left(\ln(4/\delta) + \lceil \log_2(1/\gamma) \rceil \, \ln(2)\right)}{m}} \,. \tag{56}$$

The above corollary gives a formal justification to the large margin principle. It tells us that among all margin parameters that still provides a zero training loss, it is better to choose the margin parameter to be as large as possible, since the generalization bound decreases when γ is increasing.

The analysis we have performed is called *structural risk minimization*. In contrast to ERM, when we only consider the empirical risk, here we also consider the complexity of the hypothesis class, and prefer to choose a hypothesis class with a smaller complexity (i.e. larger margin). This idea is closely related to the Occam's razor principle and MDL bounds we encountered previously in the course.

28 The Aggressive Perceptron

In this section we present the aggressive Perceptron learning algorithm. We show that this very simple algorithm finds a separator whose margin is at least third the margin of the max margin separator.

Algorithm 3 Aggressive Perceptron

`Input:` A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
Initialize: $\mathbf{w} = \mathbf{0}$
While ($\exists i$ s.t. $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle < 1$)
  $\mathbf{w} = \mathbf{w} + y_i \mathbf{x}_i$
`Output:` $\mathbf{w}$

Theorem 13 *Let $\mathbf{w}^\star$ be a minimizer of Eq. (52) with $p = 2$. Assume that for all i we have that $\|\mathbf{x}_i\| \leq 1$. Then, the Aggressive Perceptron algorithm stops after at most $3\,\|\mathbf{w}^\star\|^2$ iterations and when it stops we have*

1. $\forall i \in [m], \;\; y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$
2. $\|\mathbf{w}\| \leq 3\,\|\mathbf{w}^\star\|$

Proof Denote $\mathbf{w}_t$ to be the value of $\mathbf{w}$ at the beginning of the tth iteration of the Aggressive Perceptron. We prove the theorem by monitoring the value of $\langle \mathbf{w}^\star, \mathbf{w}_t \rangle$. At the first iteration, $\mathbf{w}_1 = \mathbf{0}$ and therefore $\langle \mathbf{w}^\star, \mathbf{w}_1 \rangle = 0$. On iteration t, if we update using example $(\mathbf{x}_i, y_i)$ we have that

$$\langle \mathbf{w}^\star, \mathbf{w}_{t+1} - \mathbf{w}_t \rangle = y_i \langle \mathbf{w}^\star, \mathbf{x}_i \rangle \geq 1 \,.$$

Therefore, after t iterations we have that $\langle \mathbf{w}^\star, \mathbf{w}_{t+1} \rangle \geq t$. On the other hand, from Cauchy-Schwartz inequality we have that

$$\langle \mathbf{w}^\star, \mathbf{w}_{t+1} \rangle \leq \|\mathbf{w}^\star\| \, \|\mathbf{w}_{t+1}\| \,.$$

Next, we upper bound the value of $\|\mathbf{w}_{t+1}\|$. Initially, $\|\mathbf{w}_1\| = 0$ and after each update we have

$$\|\mathbf{w}_{t+1}\|^2 - \|\mathbf{w}_t\|^2 = 2y_i\langle \mathbf{w}_t, \mathbf{x}_i\rangle + \|\mathbf{x}_i\|^2 \leq 3 \; .$$

Therefore,

$$\|\mathbf{w}_{t+1}\| \leq \sqrt{3\,t} \; . \tag{57}$$

Overall, we have shown that

$$t \leq \langle \mathbf{w}^\star, \mathbf{w}_{t+1}\rangle \leq \|\mathbf{w}^\star\|\,\|\mathbf{w}_{t+1}\| \leq \sqrt{3\,t}\,\|\mathbf{w}^\star\| \; .$$

Rearranging the above we get that $t \leq 3\,\|\mathbf{w}^\star\|^2$, which together with Eq. (57) concludes our proof. ■

(67577) Introduction to Machine Learning November 30, 2009

Lecture 8(c) – Surrogate loss functions

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the previous lecture we defined the class of fuzzy Halfspaces. We showed that in the realizable case, it is possible to learn a fuzzy Halfspace by using linear programming (for ℓ_1 margin) or quadratic programming (for ℓ_2 margin). In this lecture we consider the non-realizable case. Recall that the loss function of a fuzzy Halfspace, parametrized by $\mathbf{w}$, on an example $(\mathbf{x}, y)$ is defined as

$$\ell(\mathbf{w}, (\mathbf{x}, y)) \;=\; \left|\tau_\gamma(\langle \mathbf{w}, \mathbf{x}\rangle) - \tfrac{y+1}{2}\right| \;.$$

It is easy to verify that an equivalent way to express the loss function is as follows:

$$\ell(\mathbf{w}, (\mathbf{x}, y)) \;=\; \min\left\{1\,,\; \tfrac{1}{2}\,\max\left\{0, 1 - y\langle \mathbf{w}, \mathbf{x}\rangle/\gamma\right\}\right\} \;.$$

That is, we can rewrite the ERM optimization problem as:

$$\min_{\mathbf{w}:\|\mathbf{w}\|_p=1} \sum_{i=1}^{m} f(y_i\langle \mathbf{w}, \mathbf{x}_i\rangle) \;,$$

where $f : \mathbb{R} \to \mathbb{R}$ is the scalar loss function,

$$f(a) = \min\{1, 0.5\max\{0, 1 - a/\gamma\}\} \;.$$

Solving the ERM problem is difficult because the function f is non-convex and the constraint $\|\mathbf{w}\|_p = 1$ is also non-convex. The non convexity of the constraint can be easily circumvented by allowing $\mathbf{w}$ to have a norm of at most 1 (instead of exactly 1). That is, we replace the original constraint, $\|\mathbf{w}\|_p = 1$, with the constraint $\|\mathbf{w}\|_p \leq 1$. This is legitimate since any $\mathbf{w}$ whose norm is at most 1 defines a legitimate fuzzy Halfspace.

To circumvent the non-convexity of f we shall upper bound f by the convex surrogate loss function

$$g(a) = 0.5\max\{0, 1 - a/\gamma\} \;.$$

An illustration of the functions f and g is given in Figure 7. Overall, we obtained the following optimization

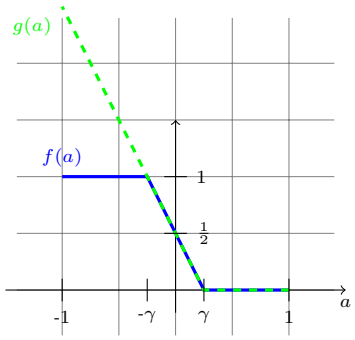


Figure 7: An illustration of the scalar loss function f and its surrogate convex upper bound g.

problem:

$$\min_{\mathbf{w}:\|\mathbf{w}\|_p \leq 1} \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle / \gamma\} \,, \tag{58}$$

or equivalently,

$$\min_{\mathbf{w}:\|\mathbf{w}\|_p \leq 1/\gamma} \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\} \,. \tag{59}$$

The scalar function $a \mapsto \max\{0, 1 - a\}$ is called the *hinge-loss*. This is a convex loss function and therefore Eq. (59) can be solved in polynomial time by various methods. In the next lecture we will discuss several simple generic methods which are adequate for convex loss minimization.

28.1 Regularization

TBA

28.2 Logistic Regression

TBA

(67577) Introduction to Machine Learning December 7, 2009

Lecture 10 – Convex Optimization and Online Convex Optimization

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In previous lectures we cast learning problems as convex optimization problems. Convex optimization can be solved in polynomial time by off-the-shelf tools. Furthermore, for the problems of minimizing the training hinge-loss and logistic loss many dedicated methods have been proposed that use the specific structure of the problem. In this lecture we present specific simple, yet effective, convex optimization procedures for convex loss minimization. After a brief overview of convex analysis we describe a game called online convex optimization. This game is closely related to the online learning framework, which we will learn later on in the course. We will present simple algorithms for online convex optimization and later on use them for deriving stochatic-gradient descent procedures for convex loss minimization.

29 Convexity

A set A is convex if for any two vectors $\mathbf{w}_1, \mathbf{w}_2$ in A, all the line between $\mathbf{w}_1$ and $\mathbf{w}_2$ is also within A. That is, for any $\alpha \in [0,1]$ we have that $\alpha\mathbf{w}_1 + (1-\alpha)\mathbf{w}_2 \in A$. A function $f : A \to \mathbb{R}$ is convex if for all $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ and $\alpha \in [0,1]$ we have

$$f(\alpha\mathbf{u} + (1-\alpha)\mathbf{v}) \;\leq\; \alpha f(\mathbf{u}) + (1-\alpha) f(\mathbf{v}) \;.$$

It is easy to verify that f is convex iff its *epigraph* is a convex set, where $\mathrm{epigraph(f)} = \{(\mathbf{x}, \alpha) : f(\mathbf{x}) \leq \alpha\}$. Throughout this section we focus on convex functions.

Sub-gradients: A vector $\boldsymbol{\lambda}$ is a *sub-gradient* of a function f at $\mathbf{w}$ if for all $\mathbf{u} \in A$ we have that

$$f(\mathbf{u}) - f(\mathbf{w}) \geq \langle \mathbf{u} - \mathbf{w}, \boldsymbol{\lambda} \rangle \;.$$

The *differential set* of f at $\mathbf{w}$, denoted $\partial f(\mathbf{w})$, is the set of all sub-gradients of f at $\mathbf{w}$. For scalar functions, a sub-gradient of a convex function f at x is a slope of a line that touches f at x and is not above f everywhere.

Two useful properties of subgradients are given below:

1. If f is differentiable at $\mathbf{w}$ then $\partial f(\mathbf{w})$ consists of a single vector which amounts to the *gradient* of f at $\mathbf{w}$ and is denoted by $\nabla f(\mathbf{w})$. In finite dimensional spaces, the gradient of f is the vector of partial derivatives of f.

2. If $g(\mathbf{w}) = \max_{i \in [r]} g_i(\mathbf{w})$ for r differentiable functions $g_1, \ldots, g_r$, and $j = \arg\max_i g_i(\mathbf{u})$, then the gradient of g_j at $\mathbf{u}$ is a subgradient of g at $\mathbf{u}$.

Example 1 (Sub-gradients of the logistic-loss) *Recall that the logistic-loss is defined as* $\ell(\mathbf{w}; \mathbf{x}, y) = \log(1 + \exp(-y\langle \mathbf{w}, \mathbf{x} \rangle))$. *Since this function is differentiable, a sub-gradient at* $\mathbf{w}$ *is the gradient at* $\mathbf{w}$*, which using the chain rule equals to*

$$\nabla \ell(\mathbf{w}; \mathbf{x}, y) = \frac{-\exp(-y\langle \mathbf{w}, \mathbf{x} \rangle)}{1 + \exp(-y\langle \mathbf{w}, \mathbf{x} \rangle)}\, y\, \mathbf{x} = \frac{-1}{1 + \exp(y\langle \mathbf{w}, \mathbf{x} \rangle)}\, y\, \mathbf{x} \;.$$

Example 2 (Sub-gradients of the hinge-loss) *Recall that the hinge-loss is defined as* $\ell(\mathbf{w}; \mathbf{x}, y) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$. *This is the maximum of two linear functions. Therefore, using the two propoerties*

above we have that if $1-y\langle \mathbf{w},\mathbf{x}\rangle > 0$ *then* $-y\,\mathbf{x} \in \partial\ell(\mathbf{w};\mathbf{x},y)$ *and if* $1-y\langle \mathbf{w},\mathbf{x}\rangle < 0$ *then* $\mathbf{0} \in \partial\ell(\mathbf{w};\mathbf{x},y)$. *Furthermore, it is easy to verify that*

$$\partial\ell(\mathbf{w};\mathbf{x},y) = \begin{cases} \{-y\mathbf{x}\} & \text{if } 1-y\langle \mathbf{w},\mathbf{x}\rangle > 0 \\ \{\mathbf{0}\} & \text{if } 1-y\langle \mathbf{w},\mathbf{x}\rangle < 0 \\ \{-\alpha y\mathbf{x} : \alpha \in [0,1]\} & \text{if } 1-y\langle \mathbf{w},\mathbf{x}\rangle = 0 \end{cases}$$

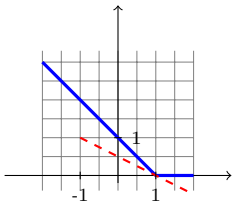


Figure 8: An illustration of the hinge-loss function $f(x) = \max\{0, 1-x\}$ and one of its sub-gradients at $x = 1$.

Lipschitz functions: We say that $f : A \to \mathbb{R}$ is ρ-Lipschitz if for all $\mathbf{u}, \mathbf{v} \in A$

$$|f(\mathbf{u}) - f(\mathbf{v})| \leq \rho \|\mathbf{u} - \mathbf{v}\| .$$

An equivalent definition is that the ℓ_2 norm of all sub-gradients of f at points in A is bounded by ρ.

30 Online Convex Optimization

A convex repeated game is a two players game that is performed in a sequence of consecutive rounds. On round t of the repeated game, the first player chooses a vector $\mathbf{w}_t$ from a convex set A. Next, the second player responds with a convex function $g_t : A \to \mathbb{R}$. Finally, the first player suffers an instantaneous loss $g_t(\mathbf{w}_t)$. We study the game from the viewpoint of the first player.

In offline convex optimization, the goal is to find a vector $\mathbf{w}$ within a convex set A that minimizes a convex objective function, $g : A \to \mathbb{R}$. In online convex optimization, the set A is known in advance, but the objective function may change along the online process. The goal of the online optimizer, which we call the learner, is to minimize the averaged objective value $\frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}_t)$, where T is the total number of rounds.

Low regret: Naturally, an adversary can make the cumulative loss of our online learning algorithm arbitrarily large. For example, the second player can always set $g_t(\mathbf{w}) = 1$ and then no matter what the learner will predict, the cumulative loss will be T. To overcome this deficiency, we restate the learner's goal based on the notion of *regret*. The learner's regret is the difference between his cumulative loss and the cumulative loss of the optimal offline minimizer. This is termed 'regret' since it measures how 'sorry' the learner is, in retrospect, not to use the optimal offline minimizer. That is, the regret is

$$R(T) = \frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}_t) - \min_{\mathbf{w}\in A} \frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}) \ .$$

We call an online algorithm a *low regret* algorithm if $R(T) = o(1)$. Next, we present a simple online convex optimization procedure which guarantees a regret of $O(1/\sqrt{T})$ provided that the functions g_t are Lipschitz.

Online sub-gradient descent The following simple procedure is guaranteed to have $O(1/\sqrt{T})$ regret if the sub-gradients of all functions the learner receive are bounded.

Algorithm 4 Online sub-gradient descent

Initialize: $\mathbf{w}_1 = \mathbf{0}$; Choose $\eta_1 \in \mathbb{R}$
for $t = 1, \ldots, T$
  Predict $\mathbf{w}_t$
  Receive $g_t : A \to \mathbb{R}$
  Choose $\mathbf{v}_t \in \partial g_t(\mathbf{w}_t)$
  Set $\eta_t = \eta_1/\sqrt{t}$
  Update $\mathbf{w}_{t+1} = \arg\min_{\mathbf{w} \in A} \|\mathbf{w} - (\mathbf{w}_t - \eta_t \mathbf{v}_t)\|^2$
end for

We now analyze the regret of Algorithm 4. We start with the following lemma.

Lemma 19 (Projection lemma) *Let A be a convex set, let $\mathbf{u} \in A$, and let $\mathbf{v}$ be the projection of $\mathbf{w}$ on A, i.e.*

$$\mathbf{v} = \operatorname*{argmin}_{\mathbf{x} \in A} \|\mathbf{w} - \mathbf{x}\|^2 .$$

Then,

$$\|\mathbf{w} - \mathbf{u}\|^2 - \|\mathbf{v} - \mathbf{u}\|^2 \geq 0 .$$

Proof Since the desired inequality measures relative distances between $\mathbf{w}, \mathbf{v}, \mathbf{u}$ we can translate everything so that $\mathbf{v}$ will be the zero vector. If $\mathbf{w} \in A$ then the claim is trivial. Otherwise, the gradient of the objective of the optimization problem in the definition of $\mathbf{v}$ must point oustide the set A. Formally,

$$\langle \mathbf{w} - \mathbf{v}, \mathbf{u} - \mathbf{v} \rangle \leq 0 .$$

Thus,

$$\|\mathbf{w} - \mathbf{u}\|^2 - \|\mathbf{v} - \mathbf{u}\|^2 = \|\mathbf{w}\|^2 - \|\mathbf{v}\|^2 - 2\langle \mathbf{w} - \mathbf{v}, \mathbf{u} \rangle \geq \|\mathbf{w}\|^2 - \|\mathbf{v}\|^2 = \|\mathbf{w}\|^2 \geq 0 .$$

■

Next, we bound the regret in terms of the size of the sub-gradients along the online learning process.

Theorem 14 *Let $\rho \geq \max_t \|\mathbf{v}_t\|$ and $U \geq \max\{\|\mathbf{w} - \mathbf{u}\| : \mathbf{w}, \mathbf{u} \in A\}$. Then, for any $\mathbf{u} \in A$ we have*

$$\frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}_t) - \frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{u}) \;\leq\; \frac{1}{\sqrt{T}}\left(\frac{U^2}{2\eta_1} + \rho^2 \eta_1\right) .$$

In particular, setting $\eta_1 = \frac{U}{\rho\sqrt{2}}$ gives

$$\frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}_t) - \frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{u}) \;\leq\; U\rho\sqrt{\frac{2}{T}} .$$

Proof We prove the theorem by analyzing the potnetial $\|\mathbf{w}_t - \mathbf{u}\|^2$. Initially, $\|\mathbf{w}_1 - \mathbf{u}\|^2 = \|\mathbf{u}\|^2$. Let $\mathbf{w}'_t = \mathbf{w}_t - \eta_t \mathbf{v}_t$. Then,

$$\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2 \;=\; \left(\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}'_t - \mathbf{u}\|^2\right) + \left(\|\mathbf{w}'_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2\right) .$$

The second summand is non-negative because of the projection lemma. For the first summand we have

$$\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}'_t - \mathbf{u}\|^2 = 2\eta_t \langle \mathbf{w}_t - \mathbf{u}, \mathbf{v}_t \rangle - \eta_t^2 \|\mathbf{v}_t\|^2 \,.$$

The fact that $\mathbf{v}_t$ is a sub-gradient of g_t at $\mathbf{w}_t$ implies that $\langle \mathbf{w}_t - \mathbf{u}, \mathbf{v}_t \rangle \geq g_t(\mathbf{w}_t) - g_t(\mathbf{u})$. Therefore,

$$\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2 \quad \geq \quad 2\eta_t (g_t(\mathbf{w}_t) - g_t(\mathbf{u})) - \eta_t^2 \, \rho^2 \,.$$

Rearranging the above gives

$$g_t(\mathbf{w}_t) - g_t(\mathbf{u}) \quad \leq \quad \frac{\|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2}{2\eta_t} + \frac{\eta_t}{2} \rho^2 \,.$$

Summing over t and rearranging we obtain

$$\begin{aligned}
\sum_{t=1}^{T} (g_t(\mathbf{w}_t) - g_t(\mathbf{u})) &\leq \|\mathbf{w}_1 - \mathbf{u}\|^2 \tfrac{1}{2\eta_1} + \sum_{t=2}^{T} \|\mathbf{w}_t - \mathbf{u}\|^2 \left(\tfrac{1}{2\eta_t} - \tfrac{1}{2\eta_{t-1}} \right) + \tfrac{\rho^2}{2} \sum_{t=1}^{T} \eta_t \\
&\leq U^2 \left(\tfrac{1}{2\eta_1} + \sum_{t=2}^{T} \left(\tfrac{1}{2\eta_t} - \tfrac{1}{2\eta_{t-1}} \right) \right) + \tfrac{\rho^2}{2} \sum_{t=1}^{T} \eta_t \\
&= U^2 \tfrac{1}{2\eta_T} + \tfrac{\rho^2 \eta_1}{2} \sum_{t=1}^{T} \tfrac{1}{\sqrt{t}} \\
&\leq U^2 \tfrac{\sqrt{T}}{2\eta_1} + \rho^2 \eta_1 \sqrt{T} \,.
\end{aligned}$$

Dividing the above by T we conclude our proof. ■

As a corollary we obtain:

Corollary 6 *Assume that $U \geq \max\{\|\mathbf{w} - \mathbf{u}\| : \mathbf{w}, \mathbf{u} \in A\}$ and that for all t the function g_t is ρ-Lipschitz. Then, Algorithm 4 guarantees*

$$\frac{1}{T} \sum_{t=1}^{T} g_t(\mathbf{w}_t) - \min_{\mathbf{u} \in A} \frac{1}{T} \sum_{t=1}^{T} g_t(\mathbf{u}) \;\leq\; \rho U \sqrt{\frac{2}{T}} \,.$$

That is, the online gradient descent procedure guarantees $O(1/\sqrt{T})$ regret as long as the functions it receives are Lipschitz and that the diameter of A is bounded.

31 Sub-gradient Descent

Consider the problem of minimizing a convex and Lipschitz function $f(\mathbf{w})$ over a convex set A. We can apply the online convex optimization given in the previous procedure while setting $g_t \equiv f$ for all t. Then, Corollary 6 tells us that

$$\frac{1}{T} \sum_{t=1}^{T} f(\mathbf{w}_t) - \min_{\mathbf{u} \in A} \frac{1}{T} \sum_{t=1}^{T} f(\mathbf{u}) \;\leq\; U\rho \sqrt{\frac{2}{T}} \,.$$

Combining the above with Jensen's inequality we obtain:

Corollary 7 *Let A be a convex set s.t. $U \geq \max\{\|\mathbf{w} - \mathbf{u}\| : \mathbf{w}, \mathbf{u} \in A\}$. Let $f : A \to \mathbb{R}$ be a convex and ρ-Lipschitz function and consider running Algorithm 4 with $g_t \equiv f$ for all $t = 1, 2, \ldots, T$. Let $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}_t$. Then,*

$$f(\bar{\mathbf{w}}) - \min_{\mathbf{u} \in A} f(\mathbf{u}) \;\leq\; \rho U \sqrt{\frac{2}{T}} \,.$$

In other words, the above corollary tells us that for any $\epsilon > 0$, to ensure that $f(\bar{\mathbf{w}}) - \min_{\mathbf{u} \in A} f(\mathbf{u}) \leq \epsilon$ it suffices to have

$$T \geq \frac{2\rho^2 U^2}{\epsilon^2} \ .$$

31.1 Max margin revisited

Recall that the max margin optimization problem can be written as:

$$\min_{\mathbf{w}} \|\mathbf{w}\| \ \text{ s.t. } \ \max_{i \in [m]} [1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle]_+ = 0 \ ,$$

where $[a]_+ = \max\{0, a\}$. We have shown that the Aggressive Perceptron finds a 3 approximation to the above problem. Now, let $\mathbf{w}^\star$ be an optimum of the above problem. Had we known the norm of $\mathbf{w}^\star$ we could have found $\mathbf{w}^\star$ by solving the problem

$$\min_{\mathbf{w}: \|\mathbf{w}\|_2 \leq \|\mathbf{w}^\star\|} \ \max_{i \in [m]} \ [1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle]_+ \ .$$

The above problem can be solved using the sub-gradient descent procedure. To do so, we note that to find a sub-gradient of the objective function, it suffices to find i that maximizes the hinge-loss. If for all i the hinge-loss is zero, then the zero vector is a sub-gradient. Otherwise, a sub-gradient is $-y_i \mathbf{x}_i$ for the i that maximizes the hinge-loss. Additionally, the projection step in this case is simply scaling of $\mathbf{w}_t$ to have an ℓ_2 norm of at most $\|\mathbf{w}^\star\|$.

The above will work if we knew the value of $\|\mathbf{w}^\star\|$. Since in practice we do not know this value, we can search it using a binary search.

32 Stochastic Sub-gradient Descent

Let us now consider the problem of minimizing an empirical risk:

$$\min_{\mathbf{w} \in A} \ \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{w}) \ . \tag{60}$$

We assume that $f_1, \ldots, f_m$ be a sequence of convex and ρ-Lipschitz functions from A to $\mathbb{R}$, and that A is convex. To optimize Eq. (60) we can apply the sub-gradient descent procedure. However, the cost of each iteration is $O(m)$. Instead, as we show below, we prefer to perform $O(1)$ operations at each iteration. We do this, by running the online convex optimization procedure where at each round we feed the online procedure a function taken randomly from the set $\{f_1, \ldots, f_m\}$. The resulting optimization procedure is called stochastic sub-gradient descent.

Algorithm 5 Stochastic sub-gradient descent

`Input`: An optimization problem given in Eq. (60)
Initialize: $\mathbf{w}_1 = \mathbf{0}$; Choose $\eta_1 \in \mathbb{R}$
for $t = 1, \ldots, T$
  Choose i uniformly at random from $[m]$
  Set $g_t \equiv f_i$
  Choose $\mathbf{v}_t \in \partial g_t(\mathbf{w}_t)$
  Set $\eta_t = \eta_1 / \sqrt{t}$
  Set $\mathbf{w}_{t+1} = \arg\min_{\mathbf{w} \in A} \|\mathbf{w} - (\mathbf{w}_t - \eta_t \mathbf{v}_t)\|_2$
end for

We now analyze the convergence properties of the Stochastic sub-gradient descent procedure by relying on the regret bound we established for online convex optimization.

Theorem 15 *Assume that the conditions stated in Corollary 6 hold and let* $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}_t$. *Let* $F(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{w})$. *Then,*

$$\mathbb{E}[F(\bar{\mathbf{w}})] \;\leq\; \min_{\mathbf{w}\in A} F(\mathbf{w}) + \rho U\sqrt{2/T}\,,$$

where expectation is w.r.t. the randomness in choosing i *at each iteration.*

Proof Let $\mathbf{w}^\star$ be a minimizer of $F(\mathbf{w})$. Taking expectation of the inequality given in Corollary 6 we obtain

$$E[\frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}_t)] \;\leq\; \mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}^\star)] + \rho U\sqrt{2/T}\,. \tag{61}$$

We now analyze the two expectations given in Eq. (61). Since g_t is chosen randomly to be some f_i, and $\mathbf{w}^\star$ does not depend on the choice of i, we have that for all t, $\mathbb{E}[g_t(\mathbf{w}^\star)] = F(\mathbf{w}^\star)$ and thus

$$\mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}^\star)] = \; F(\mathbf{w}^\star)\,. \tag{62}$$

Next, we analyze the expectation at the left-hand side of Eq. (61). Note that $\mathbf{w}_t$ only depends on the choice of $g_1, \ldots, g_{t-1}$ and not on the choice of g_t. Thus, using the law of total expectation we get that

$$\mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{w}_t)] \;=\; \mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} F(\mathbf{w}_t)]\,. \tag{63}$$

Finally, Jensen's inequality tells us that $F(\bar{\mathbf{w}}) \leq \frac{1}{T}\sum_{t=1}^{T} F(\mathbf{w}_t)$. Combining the above with Equations 61-63 we conclude our proof. ■

Remark 2 *It is also possible to derive a variant of Theorem 15 that holds with high probability by relying on a measure concentration inequality due to Azuma.*

To achieve a solution with expected accuracy of ϵ we need that

$$T \geq \frac{2\rho^2 U^2}{\epsilon^2}\,.$$

Interestingly, the number of iterations required by the stochatic gradient descent procedure does not depend on m, the number of examples. In particular, we can run it on the distribution itself ...

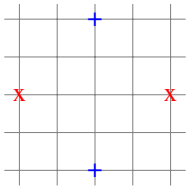
(67577) Introduction to Machine Learning December 14, 2009

Lecture 10 – Kernels

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the previous lectures we talked about the hypothesis class of Halfspaces. Seemingly, the expressive power of Halfspaces is rather restricted – for example, it is impossible to explain the training set below by a Halfspace hypothesis.



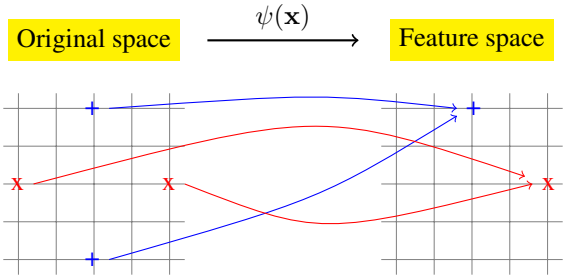
In this lecture we present the concept of *kernels*, which makes the class of Halfspaces much more expressive. The kernel trick has had tremendous impact on machine learning theory and algorithms over the past decade.

33 Mapping to a feature space

To make the class of Halfspaces more expressive, we can first map the original instance space into another space (possibly, of higher dimension) and then learn a Halfspace in that space. For example, consider the example mentioned previously. Instead of learning a Halfspace in the original representation let us first define a mapping $\psi : \mathbb{R}^2 \to \mathbb{R}^2$ as follows:

$$\psi((x_1, x_2)) = (|x_1|, |x_2|) \ .$$

We use the term *feature space* to denote the range of ψ. After applying ψ the data can be easily explained using a Halfspace:



Of course, choosing a good ψ is part of our prior knowledge on the problem. But, there are some generic mappings that enable to enrich the class of Halfspaces. One notable example is polynomial mappings.

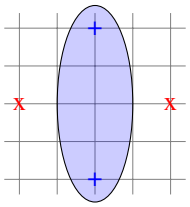
Recall that with a standard Halfspace classifier, the prediction on an instance $\mathbf{x}$ is based on the linear mapping $\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle$. We can generalize linear mappings to a polynomial mapping, $\mathbf{x} \mapsto p(\mathbf{x})$, where p is a polynomial of degree k. For simplicity, consider first the case in which $\mathbf{x}$ is 1 dimensional. In that case, $p(x) = \sum_{j=0}^{k} w_j x^j$, where $\mathbf{w} \in \mathbb{R}^{k+1}$ is the vector of coefficients of the polynomial we need to learn. We can rewrite $p(x) = \langle \mathbf{w}, \psi(x) \rangle$ where $\psi : \mathbb{R} \to \mathbb{R}^{k+1}$ is the mapping $x \mapsto (x^0, x^1, \ldots, x^k)$. That is, learning a k-degree polynomial in $\mathbb{R}$ can be done by learning a linear mapping in the feature space, which is $\mathbb{R}^{k+1}$ in our case.

More generally, a degree k multivariate polynomial from $\mathbb{R}^n$ to $\mathbb{R}$ can be written as

$$p(\mathbf{x}) = \sum_{J \in [n]^r : r \leq k} w_J \prod_{i=1}^{r} x_{J_i} \ . \tag{64}$$

As before, we can rewrite $p(\mathbf{x}) = \langle \mathbf{w}, \psi(\mathbf{x}) \rangle$ where now $\psi : \mathbb{R}^n \to \mathbb{R}^d$ such that for each $J \in [n]^r$, $r \leq k$, the coordinate of $\psi(\mathbf{x})$ associated with J is the monomial $\prod_{i=1}^{r} x_{J_i}$.

Naturally, polynomials-based classifiers yield much richer hypotheses classes than Halfspaces. For example, while the training set given in the beginning of this section cannot be explained by a Halfspace, it can be explained by an ellipse, which is a degree 2 polynomial.



So while the classifier is always linear in the feature space, it can have a highly non-linear behavior on the original space from which instances were sampled.

In general, we can choose any feature mapping ψ that maps the original instances into some *Hilbert space* (namely, a complete[4] inner-product space). The Euclidean space $\mathbb{R}^d$ is a Hilbert space for any finite d. But, there are also infinite dimensional Hilbert space (see next section).

34 The kernel trick

In the previous section we saw how to enrich the class of Halfspaces by first applying a non-linear mapping, ψ, that maps the instance space into a feature space, and then learning a Halfspace in the feature space. If the range of ψ is a high dimensional space we face two problems. First, the VC dimension of Halfspaces is the dimension and therefore we need much more samples in order to learn a Halfspace in the range of ψ. Second, from the computational point of view, performing calculations in the high dimensional space might be too costly. In fact, even the representation of the vector $\mathbf{w}$ in the feature space can be unrealistic.

To overcome the first problem, we can learn a fuzzy Halfspace in the feature space rather than a Halfspace. Recall that the Rademacher complexity of fuzzy Halfspaces (w.r.t. ℓ_2 margin) does not depend on the dimension but only depends on the margin parameter. Therefore, even if the dimensionality of the feature space is high (and even infinite), we can still learn a fuzzy Halfspace in the feature space with a number of examples that only depends on the margin parameter.

Interestingly, as we show below, learning a fuzzy Halfspace w.r.t. ℓ_2 margin enables us to overcome the computational problem as well. To do so, first note that all optimization problems associated with learning a fuzzy Halfspace w.r.t. ℓ_2 margin are special cases of the following general problem:

$$\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} f(y_i \langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle) + R(\|\mathbf{w}\|_2) \ , \tag{65}$$

where $f : \mathbb{R} \to \mathbb{R}$ and $R : \mathbb{R}_+ \to \mathbb{R}$ is monotonically non-decreasing. For example, to write the problem in Eq. (59) we set $f(a) = [1 - a]_+$ and $R(a) = 0$ if $a \leq 1/\gamma$ and $R(a) = \infty$ otherwise.

[4] A space is complete if all Cauchy sequences in the space converge. A sequence, $\mathbf{x}_1, \mathbf{x}_2, \ldots$, in a normed space is a Cauchy sequence if for any $\epsilon > 0$ there exists a large enough n such that for any $i, j > n$ we have $\|\mathbf{x}_i - \mathbf{x}_j\| < \epsilon$. The sequence converges to a point $\mathbf{x}$ if $\|\mathbf{x}_n - x\| \to 0$ as $n \to \infty$. In our case, the norm $\|\mathbf{w}\|$ is defined by the inner product $\sqrt{\langle \mathbf{w}, \mathbf{w} \rangle}$. The reason we require the range of ψ to be in a Hilbert space is because projections in a Hilbert space are well defined. In particular, if M is a linear subset of a Hilbert space, then every $\mathbf{x}$ in the Hilbert space can be written as a sum $\mathbf{x} = \mathbf{u} + \mathbf{v}$ where $\mathbf{u} \in M$ and $\langle \mathbf{v}, \mathbf{w} \rangle = 0$ for all $\mathbf{w} \in M$. We use this fact in the proof of the Wahba's representer theorem given in the next section.

The following theorem tells us that there exists an optimal solution of Eq. (65) that lies in the span of the examples.

Theorem 16 (Wahba's Representer Theorem) *Assume that ψ is a mapping from $\mathcal{X}$ to a Hilbert space. Then, there exists a vector $\boldsymbol{\alpha} \in \mathbb{R}^m$ such that $\mathbf{w} = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$ is an optimal solution of Eq. (65).*

Proof Let $\mathbf{w}^\star$ be an optimal solution of Eq. (65). Because $\mathbf{w}^\star$ is an element of a Hilbert space, we can rewrite $\mathbf{w}^\star$ as

$$\mathbf{w}^\star = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i) + \mathbf{u} \;,$$

where $\langle \mathbf{u}, \psi(\mathbf{x}_i) \rangle = 0$ for all i. Set $\mathbf{w} = \mathbf{w}^\star - \mathbf{u}$. Clearly, $\|\mathbf{w}^\star\|_2^2 = \|\mathbf{w}\|_2^2 + \|\mathbf{u}\|_2^2$, thus $\|\mathbf{w}\|_2 \leq \|\mathbf{w}^\star\|_2$. Since R is non-decreasing we obtain that $R(\|\mathbf{w}\|) \leq R(\|\mathbf{w}^\star\|)$. Additionally, for all i we have that

$$f(y_i \langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle) = f(y_i \langle \mathbf{w} + \mathbf{u}, \psi(\mathbf{x}_i) \rangle) = f(y_i \langle \mathbf{w}^\star, \psi(\mathbf{x}_i) \rangle) \;.$$

We have shown that the objective of Eq. (65) at $\mathbf{w}$ cannot be larger than the objective at $\mathbf{w}^\star$ and therefore $\mathbf{w}$ is also an optimal solution, which concludes our proof. ■

Based on the representer theorem we can optimize Eq. (65) w.r.t. the coefficients $\boldsymbol{\alpha}$ instead of the coefficients $\mathbf{w}$ as follows. Writing $\mathbf{w} = \sum_{j=1}^m \alpha_j \psi(\mathbf{x}_j)$ we have that for all i

$$\langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle = \langle \sum_j \alpha_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \rangle = \sum_{j=1}^m \alpha_j \langle \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \rangle \;.$$

Similarly,

$$\|\mathbf{w}\|_2^2 = \langle \sum_j \alpha_j \psi(\mathbf{x}_j), \sum_j \alpha_j \psi(\mathbf{x}_j) \rangle = \sum_{i,j=1}^m \alpha_i \alpha_j \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle \;.$$

Let $K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ be a function that implements inner products in the feature space. We call K a *kernel* function. Instead of solving Eq. (65) we can solve the equivalent problem

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^m} \frac{1}{m} \sum_{i=1}^m f\left(y_i \sum_{j=1}^m \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) \right) + R\left(\sqrt{\sum_{i,j=1}^m \alpha_i \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)} \right) \;. \tag{66}$$

To solve the optimization problem given in Eq. (66), we do not need any direct access to elements in the features space. The only thing we should know is how to calculate inner-products in the feature space, or equivalently, to calculate the kernel function. In fact, to solve Eq. (66) we solely need to know the value of the $m \times m$ matrix G s.t. $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$, which is often called the *Gram* matrix.

Once we learned the coefficients $\boldsymbol{\alpha}$ we can calculate the prediction on a new instance by:

$$\langle \mathbf{w}, \psi(\mathbf{x}) \rangle = \sum_{j=1}^m \alpha_j \langle \psi(\mathbf{x}_j), \psi(\mathbf{x}) \rangle = \sum_{j=1}^m \alpha_j K(\mathbf{x}_j, \mathbf{x}) \;.$$

Example 3 (Polynomial kernels) *Consider the mapping $\psi : \mathbb{R}^n \to \mathbb{R}^d$ mapping $\mathbf{x}$ to all its monomial of order at most k. That is, for any $r \leq k$ and $J \in [n]^r$ there exists a coordinate $(\psi(\mathbf{x}))_J = \prod_{i=1}^r x_{J_i}$. This is the mapping corresponds to a degree k multivariate polynomial as given in Eq. (64). To implement $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ we note that*

$$\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = \sum_{J \in [n]^r : r \leq k} \left(\prod_{j=1}^r x_{J_j} \right) \left(\prod_{j=1}^r x'_{J_j} \right) = \sum_{J \in [n]^r : r \leq k} \prod_{j=1}^r (x_{J_j} x'_{J_j}) = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k .$$

Therefore, we can learn a degree k multivariate polynomial by solving Eq. (65) with the kernel function

$$K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k \ .$$

Example 4 (Gaussian kernel) *Let the original instance space be $\mathbb{R}$ and consider the mapping ψ where for each non-negative integer $n \geq 0$ there exists an element $\psi(x)_n$ which equals to $\frac{1}{\sqrt{n!}} e^{-\frac{x^2}{2}} x^n$. Then,*

$$\langle \psi(x), \psi(x') \rangle = \sum_{n=0}^{\infty} \left(\frac{1}{\sqrt{n!}} e^{-\frac{x^2}{2}} x^n \right) \left(\frac{1}{\sqrt{n!}} e^{-\frac{(x')^2}{2}} (x')^n \right) = e^{-\frac{x^2+(x')^2}{2}} \sum_{n=0}^{\infty} \left(\frac{(xx')^n}{n!} \right) = e^{-\frac{\|x-x'\|^2}{2}} \ .$$

More generally, given a scalar $\sigma > 0$, the Gaussian kernel is defined to be

$$K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\,\sigma}} \ .$$

It is easy to verify that K implements an inner-product in a space in which for any n and any monomial of order n there exists an element of $\psi(\mathbf{x})$ that equals to $\frac{1}{\sqrt{n!}} e^{-\frac{\|x\|^2}{2}} \prod_{i=1}^{n} x_{j_i}$.

35 Implementing Gradient-based algorithms with Kernels

In the previous section we saw that in order to learn a fuzzy Halfspace w.r.t. ℓ_2 margin, it is possible to solve the optimization problem given in Eq. (66). We now show an even simpler approach. In particular, we show that stochastic gradient descent can be applied using kernels, and without any direct access to individual elements of the vector $\mathbf{w}$ or feature vectors $\psi(\mathbf{x}_i)$.

For concreteness, consider the optimization problem associated with minimizing a margin-based loss (which should be Lipschitz and convex) with a constraint on the ℓ_2 norm of $\mathbf{w}$, namely,

$$\min_{\mathbf{w}:\|\mathbf{w}\| \leq W} \frac{1}{m} \sum_{i=1}^{m} f(y_i \langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle) \ . \tag{67}$$

For example, f can be the hinge-loss, $f(a) = \max\{0, 1-a\}$, or the logistic loss, $f(a) = \log(1+\exp(-a))$. We assume that f is convex and ρ-Lipschitz.

Specifying Algorithm 5 for this case we obtain the following procedure:

Algorithm 6 Stochastic sub-gradient descent for solving Eq. (67)

Initialize: $\mathbf{w}_1 = \mathbf{0}$; Choose $\eta_1 \in \mathbb{R}$
for $t = 1, \ldots, T$
  Choose i uniformly at random from $[m]$
  Let $z_t = y_i \langle \mathbf{w}_t, \psi(\mathbf{x}_i) \rangle$
  Choose $\nu_t \in \partial f(z_t)$
  Set $\mathbf{v}_t = \nu_t y_i \psi(\mathbf{x}_i)$
  Set $\eta_t = \eta_1 / \sqrt{t}$
  Set $\mathbf{w}'_t = \mathbf{w}_t - \eta_t \mathbf{v}_t$
  Set $\mathbf{w}_{t+1} = \min\left\{1, \frac{W}{\|\mathbf{w}'_t\|}\right\} \mathbf{w}'_t$
end for

We next argue that for all t, $\mathbf{w}_t$ can be written as $\sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i)$ for some vector $\boldsymbol{\alpha} \in \mathbb{R}^m$. This is true by a simple inductive argument. Initially, $\mathbf{w}_1 = \mathbf{0}$ so the claim clearly holds (simply set $\alpha_i = 0$ for all i).

On round t, we first construct $\mathbf{w}'_t = \mathbf{w}_t - \eta_t \mathbf{v}_t$. Using the inductive assumption, $\mathbf{w}_t = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$ and therefore by setting $\alpha_i \leftarrow \alpha_i - \eta_t \nu_t y_i$ we get that $\mathbf{w}'_t = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$ as required. Finally,

$$\mathbf{w}_{t+1} = \min\left\{1, \tfrac{W}{\|\mathbf{w}'_t\|}\right\} \mathbf{w}'_t = \min\left\{1, \tfrac{W}{\|\mathbf{w}'_t\|}\right\} \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i) = \sum_{i=1}^m \left(\min\left\{1, \tfrac{W}{\|\mathbf{w}'_t\|}\right\} \alpha_i\right) \psi(\mathbf{x}_i) \ ,$$

which concludes the inductive argument.

The resulting algorithm is given below:

Algorithm 7 Stochastic sub-gradient descent for solving Eq. (67) using kernels

Initialize: $\boldsymbol{\alpha} = \mathbf{0}$; Choose $\eta_1 \in \mathbb{R}$
for $t = 1, \ldots, T$
  Choose i uniformly at random from $[m]$
  Let $z_t = y_i \sum_{j=1}^m \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)$
  Choose $\nu_t \in \partial f(z_t)$
  Set $\eta_t = \eta_1 / \sqrt{t}$
  Set $\alpha_i = \alpha_i - \eta_t \nu_t y_i$
  Set $\|\mathbf{w}'_t\| = \sqrt{\sum_{j,r} \alpha_j \alpha_r K(\mathbf{x}_j, x_r)}$
  Set $\boldsymbol{\alpha} = \min\left\{1, \tfrac{W}{\|\mathbf{w}'_t\|}\right\} \boldsymbol{\alpha}$
end for

36 Support Vector Machines

Recall the max-margin optimization problem:

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|_2^2 \ \text{ s.t. } \ \forall i \in [m], \ y_i \langle \mathbf{w}, \psi(\mathbf{x}_i)\rangle \geq 1 \ . \tag{68}$$

Learning $\mathbf{w}$ using the above rule is called hard Support Vector Machine (hard SVM). Based on the representer theorem we know that an optimal solution of Eq. (68) takes the form $\mathbf{w} = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$. We will show below that we can find such a representation of $\mathbf{w}$ for which $\alpha_i \neq 0$ only if $\langle \mathbf{w}, \psi(\mathbf{x}_i)\rangle = 1$. Put another way, $\mathbf{w}$ is supported by the examples that are exactly at distance $1/\|\mathbf{w}\|$ from the separating hyperplane. These vectors are therefore called *support vectors*.

Fritz John condition Consider the problem:

$$\min_{\mathbf{w}} f(\mathbf{w}) \quad \text{s.t. } \ \forall i \in [m], \ g_i(\mathbf{w}) \leq 0 \ ,$$

where $f, g_1, \ldots, g_m$ are differentiable. Then, if $\mathbf{w}^\star$ is an optimal solution then there exists $\boldsymbol{\alpha} \in \mathbb{R}^m$ such that $\nabla f(\mathbf{w}^\star) + \sum_{i \in I} \alpha_i \nabla g_i(\mathbf{w}^\star) = \mathbf{0}$, where $I = \{i : g_i(\mathbf{w}^\star) = 0\}$.

Applying Fritz John condition on Eq. (68) we obtain:

Corollary 8 *Let $\mathbf{w}^\star$ be a minimizer of Eq. (68) and let $I = \{i : \langle \mathbf{w}^\star, \psi(\mathbf{x}_i)\rangle = 1\}$. Then, there exist coefficients α_i such that*

$$\mathbf{w}^\star = \sum_{i \in I} \alpha_i \psi(\mathbf{x}_i) \ .$$

36.1 Duality

Traditionally, many of the properties we derived for SVM have been obtained by switching to the dual problem. For completeness, we present below how to derive the dual of Eq. (68). We start by rewriting the problem given in Eq. (68) in an equivalent form as follows. Consider the function

$$g(\mathbf{w}) \;=\; \max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \sum_{i=1}^m \alpha_i(1-y_i\langle\mathbf{w},\psi(\mathbf{x}_i)\rangle) \;=\; \begin{cases} 0 & \text{if } \forall i, y_i\langle\mathbf{w},\psi(\mathbf{x}_i)\rangle \geq 1 \\ \infty & \text{otherwise} \end{cases} .$$

We can therefore rewrite Eq. (68) as

$$\min_{\mathbf{w}} \|\mathbf{w}\|_2^2 + g(\mathbf{w}) \; . \tag{69}$$

Rearranging the above we obtain

$$\min_{\mathbf{w}} \max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^m \alpha_i(1-y_i\langle\mathbf{w},\psi(\mathbf{x}_i)\rangle) \; . \tag{70}$$

We have therefore shown that the above problem is equivalent to the hard SVM problem given in Eq. (68). Now suppose that we flip the order of $\min$ and $\max$ in the above equation. It is easy to verify that this can only decrease the value, namely,

$$\min_{\mathbf{w}} \max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^m \alpha_i(1-y_i\langle\mathbf{w},\psi(\mathbf{x}_i)\rangle) \;\geq\; \max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^m \alpha_i(1-y_i\langle\mathbf{w},\psi(\mathbf{x}_i)\rangle) \; .$$

The above inequality is called *weak duality*. It turns out that in our case, *strong duality* also holds, namely, the above inequality holds with equality. The right-hand side is called the *dual* problem, namely,

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^m \alpha_i(1-y_i\langle\mathbf{w},\psi(\mathbf{x}_i)\rangle) \; . \tag{71}$$

We can rewrite the dual problem by noting that once $\boldsymbol{\alpha}$ is fixed, the optimization problem w.r.t. $\mathbf{w}$ is unconstrained and the objective is differentiable, thus, at the optimum, the gradient equals to zero:

$$\mathbf{w} - \sum_{i=1}^m \alpha_i y_i \psi(\mathbf{x}_i) \;=\; 0 \quad\Rightarrow\quad \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \psi(\mathbf{x}_i) \; .$$

This shows us again the representer property from a different angle. Plugging the above into Eq. (71) we obtain that the dual problem can be rewritten as

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \frac{1}{2}\left\|\sum_{i=1}^m \alpha_i y_i \psi(\mathbf{x}_i)\right\|_2^2 + \sum_{i=1}^m \alpha_i(1-y_i\langle\sum_j \alpha_j y_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i)\rangle) \; . \tag{72}$$

Rearranging the above gives the problem

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j y_i y_j \langle\psi(\mathbf{x}_j),\psi(\mathbf{x}_i)\rangle \; . \tag{73}$$

Note that the dual problem only involves inner products between vectors in the feature space an does not require direct access to specific elements of the feature space.

36.2 Soft SVM

In hard SVM, we assume that the data is separable with margin. Since this is a rather strong requirement, it was suggested to replace the hard separability constraint with a penalty on violating this constraint. The resulting task becomes minimization of the hinge-loss plus a squared ℓ_2 regularization term. This is know as *soft Support Vector Machines*. That is, training a soft SVM amounts to solving the following problem:

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m} [1 - y_i\langle \mathbf{w}, \psi(\mathbf{x}_i)\rangle]_+ \ , \tag{74}$$

where λ is a parameter that controls the tradeoff between a low norm and good fit to the data.

(67577) Introduction to Machine Learning December 21, 2009

Lecture 11 – Linear Regression

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

So far, we focused on learning binary classifiers, that is mappings from $\mathcal{X}$ to $\{0,1\}$. In this lecture we consider *regression* problems, in which our goal is to learn a function $h : \mathcal{X} \to \mathbb{R}$. Consider for example the problem of predicting the birth weight of a newborn based on ultra-sound measurements performed several weeks before labor. Both low birth weight and excessive fetal weight at delivery are associated with an increased risk of newborn complications during labor. This is an example of a problem in which the prediction should be a continuous number rather than just a yes/no answer.

37 Loss functions for regression

While in binary classification there is a natural loss measure, i.e. the 0-1 loss, in regression problems there are several methods to measure the mismatch between prediction and true value. The most popular choice is the squared error:

$$\ell(h;(\mathbf{x},y)) = \frac{1}{2}(h(\mathbf{x})-y)^2 \ .$$

Examples of other loss functions that have been proposed in the literature are the absolute loss, $\ell(h;(\mathbf{x},y) = |h(\mathbf{x})-y)|$, the ϵ-insensitive loss, $\ell(h;(\mathbf{x},y) = \max\{0,|h(\mathbf{x})-y)|-\epsilon\}$, and the Huber loss,

$$\ell(h;(\mathbf{x},y) \ = \ \begin{cases} \frac{1}{2}(h(\mathbf{x})-y)^2 & \text{if } |h(\mathbf{x})-y| \leq 1, \\ |h(\mathbf{x})-y| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

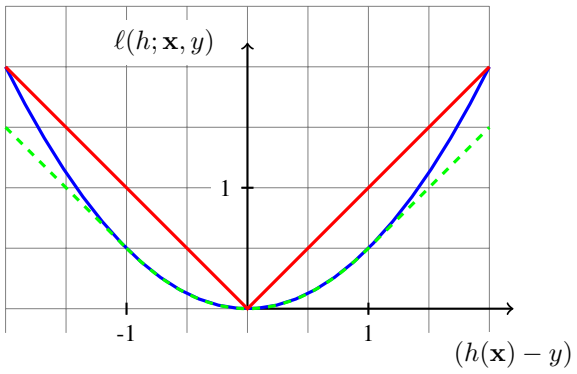


Figure 9: An illustration of squared loss, absolute loss, and Huber loss.

38 Linear regression

A linear regressor is a mapping $\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle$, where we assume that the instance space is a vector space (i.e. $\mathbf{x}$ is a vector) and the prediction is a linear combination of the instance vector $\mathbf{x}$. The problem of learning a regression function with respect to a hypothesis class of linear predictors is called *linear regression*. In the following subsections we describe algorithms for linear regression with respect to the squared loss.

38.1 Least squares

Least squares is the algorithm which solves the ERM problem with respect to the squared loss and the hypothesis class of all linear predictors. Formally, let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a sequence of m training examples where for each i we have $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Consider the class of linear predictors in $\mathbb{R}^d$:

$$\mathcal{H} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle : \mathbf{w} \in \mathbb{R}^d\} .$$

The ERM problem with respect to this class is:

$$\operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{m} \frac{1}{2} (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 .$$

To solve the above problem we calculate the gradient of the objective function and compare it to zero. That is, we need to solve

$$\sum_{i=1}^{m} (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i) \mathbf{x}_i = 0 .$$

We can rewrite the above as the problem $A\mathbf{w} = \mathbf{b}$ where [5]

$$A = \left(\sum_{i=1}^{m} \mathbf{x}_i \, \mathbf{x}_i^T \right) \quad \text{and} \quad \mathbf{b} = \sum_{i=1}^{m} y_i \mathbf{x}_i \ . \tag{75}$$

If the training instances span the entire space $\mathbb{R}^d$ then A is invertible and the solution to the ERM problem is

$$\mathbf{w} = A^{-1} \, \mathbf{b} .$$

If the training instances do not span the entire space then A is not invertible. Nevertheless, we can always find a solution to the system $A\mathbf{w} = \mathbf{b}$ because $\mathbf{b}$ is in the range of A. Indeed, since A is positive semi-definite, we can write it as $A = VDV^T$, where D is a diagonal matrix and V is an orthonormal matrix (that is, $V^T V$ is the identity $n \times n$ matrix). Define D^+ to be the diagonal matrix such that $D^+_{i,i} = 0$ if $D_{i,i} = 0$ and otherwise $D^+_{i,i} = 1/D_{i,i}$. Now, define

$$A^+ = VD^+V^T \quad \text{and} \quad \hat{\mathbf{w}} = A^+ \mathbf{b} .$$

Let $\mathbf{v}_i$ denote the i'th column of V. Then, we have

$$A\hat{\mathbf{w}} = AA^+\mathbf{b} = VDV^TVD^+V^T\mathbf{b} = VDD^+V^T\mathbf{b} = \sum_{i : D_{i,i} \neq 0} \mathbf{v}_i \mathbf{v}_i^T \mathbf{b} .$$

That is, $A\hat{\mathbf{w}}$ is the projection of $\mathbf{b}$ onto the span of those vectors $\mathbf{v}_i$ for which $D_{i,i} \neq 0$. But, those vectors are the linear span of $\mathbf{x}_1, \ldots, \mathbf{x}_m$ and $\mathbf{b}$ is in this span and therefore $A\hat{\mathbf{w}} = \mathbf{b}$, which concludes our argument.

38.2 Tikhonov regularization and Ridge Regression

The least-squares solution we presented before might be highly non-stable – namely, a slight perturbation of the input causes a dramatic change of the output. Consider for example the case when $\mathcal{X} = \mathbb{R}^2$ and the

[5] Note that we can also rewrite A and $\mathbf{b}$ as

$$A = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \ldots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \ldots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix}^T , \quad \mathbf{b} = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \ldots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix} \mathbf{y} .$$

training set contains two examples where the instances are $\mathbf{x}_1 = (1, 0)$ and $\mathbf{x}_2 = (1, \epsilon)$ and the targets are $y_1 = y_2 = 1$. Then, the matrix A becomes

$$A = \begin{pmatrix} 1 & 1 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & \epsilon \end{pmatrix}^T = \begin{pmatrix} 2 & \epsilon \\ \epsilon & \epsilon^2 \end{pmatrix}$$

and its inverse is

$$A^{-1} = \begin{pmatrix} 1 & -\frac{1}{\epsilon} \\ -\frac{1}{\epsilon} & \frac{2}{\epsilon^2} \end{pmatrix} .$$

The vector $\mathbf{b}$ is $(2, \epsilon)$ and therefore the least squares estimator is

$$\hat{\mathbf{w}} = A^{-1}\mathbf{b} = \begin{pmatrix} 1 & -\frac{1}{\epsilon} \\ -\frac{1}{\epsilon} & \frac{2}{\epsilon^2} \end{pmatrix} \begin{pmatrix} 2 \\ \epsilon \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} .$$

Now, lets repeat the above calculation with the slight change in targets: $y_1 = 1 + \epsilon$ and $y_2 = 1$. Now we have $\mathbf{b} = (2 + \epsilon, \epsilon)$ and thus

$$\hat{\mathbf{w}} = A^{-1}\mathbf{b} = \begin{pmatrix} 1 & -\frac{1}{\epsilon} \\ -\frac{1}{\epsilon} & \frac{2}{\epsilon^2} \end{pmatrix} \begin{pmatrix} 2 + \epsilon \\ \epsilon \end{pmatrix} = \begin{pmatrix} 1 + \epsilon \\ -1 \end{pmatrix} .$$

That is, for the same instances, a tiny change in the value of the targets makes a huge change in the least-squares estimator.

A problem suffering from such instability is also called an ill-posed problem. A common solution is to add regularization. Most common regularization is to add $\|\mathbf{w}\|^2$ to the optimization problem, namely, to define the estimator as

$$\operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{m} \frac{1}{2}(\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 , \tag{76}$$

where λ is a regularization parameter. This type of regularization is often called Tikhonov regularization and performing linear regression using Eq. (76) is called ridge regression.

To solve Eq. (76) we again compare the gradient to zero and obtain the set of linear equations

$$(\lambda I + A)\mathbf{w} = \mathbf{b} ,$$

where A and $\mathbf{b}$ are as defined in Eq. (75). Since A is positive semi-definite, the matrix $\lambda I + A$ has all its eigenvalues bounded below by λ. Thus, all the eigenvalues of A^{-1} are bounded above by $1/\lambda$ which guarantees a stable solution.

38.2.1 Ridge regression with kernels

Based on the Representer theorem we know that the solution of Eq. (76) can be written as a linear combination of the instances

$$\mathbf{w} = \sum_{j=1}^{m} \alpha_j \mathbf{x}_j .$$

We can therefore optimize w.r.t. $\boldsymbol{\alpha}$ instead of w.r.t. $\mathbf{w}$. That is, an equivalent problem is

$$\operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^m} \frac{\lambda}{2}\boldsymbol{\alpha}^T G \boldsymbol{\alpha} + \sum_{i=1}^{m} \frac{1}{2}(\langle \boldsymbol{g}_i, \boldsymbol{\alpha} \rangle - y_i)^2 , \tag{77}$$

where G is the Gram matrix, $G_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, and $\boldsymbol{g}_i$ is the i'th column of G. Note that Eq. (76) only access the data through inner products and therefore we can implement ridge regression using kernels.

Comparing the gradient of Eq. (76) w.r.t. $\boldsymbol{\alpha}$ to zero we obtain

$$\left(\lambda G + \sum_{i=1}^{m} \boldsymbol{g}_i \boldsymbol{g}_i^T\right) \boldsymbol{\alpha} = \sum_{i=1}^{m} y_i \boldsymbol{g}_i \; .$$

Equivalently,

$$\left(\lambda G + G G^T\right) \boldsymbol{\alpha} = G \mathbf{y} \; .$$

A sufficient (and also necessary whenever G is invertible) for the above to hold is that

$$(\lambda I + G) \, \boldsymbol{\alpha} = \mathbf{y} \; .$$

38.3 Lasso

Another from of regularization is the ℓ_1 norm. The resulting estimator is called Lasso:

$$\underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \; \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^{m} \frac{1}{2} (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)^2 \; , \tag{78}$$

While there is no closed form solution for the Lasso problem, it can still be solved efficiently by an of-the-shelf convex optimization method. In particular, we can apply the stochastic sub-gradient method for the Lasso problem.

(67577) Introduction to Machine Learning December 21, 2009

Lecture 12 – Nearest Neighbor

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

Good algorithms survive. Nearest Neighbor algorithms are amongst the simplest of all machine learning algorithms. The idea is to memorize the training set and then to predict the label of any new instance based on the labels of its neighbors in the training set. Furthermore, in some situations, the training set is immense but finding a nearest neighbor is extremely fast (for example, when the training set is the entire web and distances are based on links). In such cases, nearest neighbor is a very efficient solution.

In this lecture we describe Nearest Neighbor methods for classification and regression problems. We also analyze the performance of a Nearest Neighbor rule demonstrating its advantages and disadvantages.

39 Nearest Neighbors (NN) rules

Throughout this lecture we assume that $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y}$ is either $\{0,1\}$ (for classification) or $\mathcal{Y} = \mathbb{R}$ (for regression). Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a training set of examples in $\mathcal{X} \times \mathcal{Y}$ sampled i.i.d. according to a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

For each $\mathbf{x}$, let $\pi_1(\mathbf{x}), \ldots, \pi_m(\mathbf{x})$ be a permutation of $\{1, \ldots, m\}$ according to the distance $\|\mathbf{x} - \mathbf{x}_i\|$. That is,

$$\|\mathbf{x} - \mathbf{x}_{\pi_1(\mathbf{x})}\| \leq \|\mathbf{x} - \mathbf{x}_{\pi_2(\mathbf{x})}\| \leq \ldots \leq \|\mathbf{x} - \mathbf{x}_{\pi_m(\mathbf{x})}\| \,.$$

For simplicity, we measure distances using the Euclidean norm, but the NN method can be seamlessly defined using other distance measures as well.

Using the above notation, we consider k-NN rules of the form:

$$h_S(\mathbf{x}) \;=\; \phi\left(\frac{1}{k}\sum_{i=1}^{k} y_{\pi_i(\mathbf{x})}\right) \,,$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is a transfer function. For regression problems, we will take ϕ to be the identity function and then $h_S(\mathbf{x})$ is simply the average labels of the k nearest neighbors of $\mathbf{x}$ in S. We can also use the identity transfer function in classification. In this context, $h_S(\mathbf{x})$ will be the empirical probability to have the label 1 among the k nearest neighbors, and therefore $h_S(\mathbf{x}) \in [0,1]$. We then interpret $h_S(\mathbf{x})$ as the probability to predict the label 1 given the instance $\mathbf{x}$. Another widely used transfer function for classification is $\phi(a) = \mathbb{1}_{[a \geq 1/2]}$. This means that $h_S(\mathbf{x})$ is 1 if the majority of labels of the k neighbors is 1 and otherwise $h_S(\mathbf{x}) = 0$.

When $k = 1$, the two approaches coincides and we have the 1-NN rule:

$$h_S(\mathbf{x}) \;=\; y_{\pi_1(\mathbf{x})} \,.$$

A geometric illustration of the 1-NN rule is given in Figure 10.

40 Analysis

The NN method is different than previous methods we discussed in the course. Previously, we either assumed the existence of a predefined hypothesis class or assumed an order over hypotheses. In contrast, NN rules are completely non-parametric. There is no natural way to define a-priori a hypothesis class with bounded complexity such that the NN rule will be a member of this class.

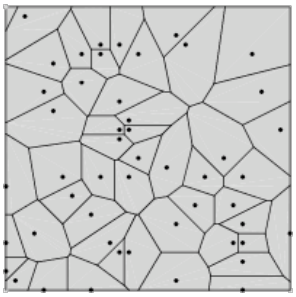


Figure 10: An illustration of the decision boundary of the 1-NN rule. The cells are called Voronoi Tesselation of the space.

Nevertheless, the generalization properties of NN rules have been extensively studied. Most previous results are asymptotic, analyzing the performance of NN rules when $m \to \infty$. As we argue in this course, these type of analyzes are not satisfactory as we would like to learn from a finite sample and to understand the generalization performance as a function of the size of the finite training set. We therefore provide a finite sample analysis of the 1-NN rule, showing how the error decreases as a function of m. In particular, the analysis shows that the generalization error of the 1-NN rule will be bounded above by twice the Bayes error plus a term that tends to zero when m increases.

Seemingly, the latter result contradicts the no-free-lunch principle, which tells us that it is impossible to learn without having some sort of prior knowledge. There is no contradiction here. In fact, our careful finite sample analysis underscores an underlying assumption on the distribution over examples and reveals that NN rules relies on a specific sort of prior knowledge. We demonstrate this fact by building specific distributions for which the 1-NN rule fails.

40.1 A generalization bound for the 1-NN rule

We now analyze the generalization error of the 1-NN rule. We first introduce notation. Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. Let $\mathcal{D}_{\mathbf{x}}$ be the induced marginal distribution over $\mathcal{X}$ and let $\eta : \mathbb{R}^d \to \mathbb{R}$ be the conditional probability over the labels, that is,

$$\eta(\mathbf{x}) = \mathbb{P}[Y = 1 | X = \mathbf{x}] \ .$$

Throughout our analysis we assume that η is a c-Lipschitz function and that the support of $\mathcal{D}_{\mathbf{x}}$ is the set $\{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_\infty \leq 1\}$.

Intuitively, the assumption that η is Lipschitz means that if two vectors are close to each other then their labels are likely to be the same. This leads us to the following lemma which upper bounds the generalization error of the 1-NN rule based on the expected distance between each test instance to its nearest neighbor in the training set.

Lemma 20 *Let $\mathcal{D}$ be a distribution over $\mathbb{R}^d \times \{0, 1\}$ and assume that η is c-Lipschitz. Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be an i.i.d. sample and let h_S be its corresponding 1-NN hypothesis. Let $h^\star$ be the Bayes optimal rule. Then,*

$$\mathop{\mathbb{E}}_{S}[\mathrm{err}(h_S)] \leq 2\,\mathrm{err}(h^\star) + c \mathop{\mathbb{E}}_{X,S}[\|X - \mathbf{x}_{\pi_1(X)}\|] \ .$$

Proof Let $\mathbf{x}, \mathbf{x}'$ be two vectors. Then, the probability to sample random labels corresponding to $\mathbf{x}$ and $\mathbf{x}'$, which are dissimilar from each other, is at most:

$$\begin{aligned} \mathop{\mathbb{P}}_{Y \sim \eta(\mathbf{x}), Y' \sim \eta(\mathbf{x}')}[Y \neq Y'] &= \eta(\mathbf{x}')(1 - \eta(\mathbf{x})) + (1 - \eta(\mathbf{x}'))\eta(\mathbf{x}) \\ &\leq 2\eta(\mathbf{x})(1 - \eta(\mathbf{x})) + c\,\|\mathbf{x} - \mathbf{x}'\| \ , \end{aligned} \tag{79}$$

where the inequality follows from the assumption that η is c-Lipschitz. Since S is sampled i.i.d. we therefore obtain that

$$\mathop{\mathbb{E}}_{S,(X,Y)}[Y \neq Y_{\pi_1(X)}] \;\leq\; \mathop{\mathbb{E}}_{X}[2\eta(X)(1-\eta(X))] + c \mathop{\mathbb{E}}_{S,X}[\|X - \mathbf{x}_{\pi_1(X)}\|] \,.$$

Last, the error of the Bayes optimal classifier is

$$\mathrm{err}(h^\star) \;=\; \mathop{\mathbb{E}}_{X}[\min\{\eta(X), 1-\eta(X)\}] \geq \mathop{\mathbb{E}}_{X}[\eta(X)(1-\eta(\mathbf{x}))] \,.$$

Combining the above two inequalities we conclude our proof. ■

The next obvious step is to bound the expected distance between a random X and its closest element in S. To do so, we first need the following lemma.

Lemma 21 *Let $C_1, \ldots, C_r$ be a sequence of subsets of $\mathbb{R}^d$. Let S be a sequence of m vectors in $\mathbb{R}^d$ sampled i.i.d. according to $\mathcal{D}_{\mathbf{x}}$. Then,*

$$\mathop{\mathbb{E}}_{S}\left[\sum_{i:C_i \cap S = \emptyset} \mathbb{P}[C_i]\right] \;\leq\; \frac{r}{m\,e} \,.$$

Proof From the linearity of expectation, we can rewrite:

$$\mathop{\mathbb{E}}_{S}\left[\sum_{i:C_i \cap S = \emptyset} \mathbb{P}[C_i]\right] \;=\; \sum_{i=1}^{r} \mathbb{P}[C_i] \mathop{\mathbb{E}}_{S}\left[\mathbb{1}_{[C_i \cap S = \emptyset]}\right] \,.$$

Next, for each i we have

$$\mathop{\mathbb{E}}_{S}\left[\mathbb{1}_{[C_i \cap S = \emptyset]}\right] = \mathop{\mathbb{P}}_{S}[C_i \cap S = \emptyset] = (1 - \mathbb{P}[C_i])^m \leq e^{-\,\mathbb{P}[C_i]\,m} \,.$$

Combing the above two equations we get

$$\mathop{\mathbb{E}}_{S}\left[\sum_{i:C_i \cap S = \emptyset} \mathbb{P}[C_i]\right] \;\leq\; \sum_{i=1}^{r} \mathbb{P}[C_i] \mathbf{e}^{-\,\mathbb{P}[C_i]\,m} \;\leq\; r \max_i \mathbb{P}[C_i] \mathbf{e}^{-\,\mathbb{P}[C_i]\,m} \,.$$

Finally, by a standard calculus analysis we have that $\max_a a e^{-ma} \leq \frac{1}{me}$ and this concludes the proof. ■

Equipped with the above lemmas we are now ready to state and prove the main result of this section.

Theorem 17 *Let h_S be the 1-NN rule. Then,*

$$\mathop{\mathbb{E}}_{S}[\mathrm{err}(h_S)] \leq 2\,\mathrm{err}(h^\star) + 4\,c\,\sqrt{d}\,m^{-\frac{1}{d+1}} \,.$$

Proof Fix some $\epsilon > 0$ and let $C_1, \ldots, C_r$ be the cover of the set $\{\mathbf{x} : \|\mathbf{x}\|_\infty \leq 1\}$ using boxes of length ϵ. For each $\mathbf{x}, \mathbf{x}'$ in the same box we have $\|\mathbf{x} - \mathbf{x}'\|_2 \leq \sqrt{d}\,\epsilon$. Otherwise, $\|\mathbf{x} - \mathbf{x}'\|_2 \leq 2\sqrt{d}$. Therefore, using Lemma 21

$$\mathop{\mathbb{E}}_{X,S}[\|X - \mathbf{x}_{\pi_1(X)}\|] \leq \mathop{\mathbb{E}}_{S}\left[\mathbb{P}\left[\bigcup_{i:C_i \cap S = \emptyset} C_i\right] 2\sqrt{d} + \mathbb{P}\left[\bigcup_{i:C_i \cap S \neq \emptyset} C_i\right] \epsilon\sqrt{d}\right] \leq \sqrt{d}\left(\tfrac{2r}{me} + \epsilon\right)$$

Since the number of balls is $r = (2/\epsilon)^d$ we get that

$$\mathop{\mathbb{E}}_{S,X}[\|X - \mathbf{x}_{\pi_1(X)}\|] \leq \sqrt{d}\,\left(\tfrac{2^{d+1}\,\epsilon^{-d}}{m\,e} + \epsilon\right) \,.$$

Combining the above with Lemma 20 we obtain that

$$\mathop{\mathbb{E}}_{S}[\mathrm{err}(h_S)] \leq 2\,\mathrm{err}(h^\star) + c\sqrt{d}\left(\frac{2^{d+1}\,\epsilon^{-d}}{m\,e} + \epsilon\right) .$$

Finally, setting $\epsilon = 2\,m^{-1/(d+1)}$ and noting that

$$\frac{2^{d+1}\,\epsilon^{-d}}{m\,e} + \epsilon = \frac{2^{d+1}\,2^{-d}\,m^{d/(d+1)}}{m\,e} + 2\,m^{-1/(d+1)} = 2m^{-1/(d+1)}(1/e+1) \leq 4m^{-1/(d+1)}$$

we conclude our proof. ■

The above theorem implies that if we first fix the distribution and then let m goes to infinity then the error of the 1-NN rule converges to twice the Bayes error. This asymptotic classic result is due to Cover and Hart (1967).

40.2 Curse of dimensionality and 'no-free-lunch'

Theorem 17 tells us that just by assuming that the conditional distribution, $\eta(\mathbf{x})$, is a Lipschitz function, the generalization error of the NN rule converges to twice the Bayes optimal error. Previously in this course we argued that one must have some prior knowledge in order to be able to learn with a finite sample. Moreover, we established the no-free-lunch principle showing that any algorithm might fail on some distributions. Our goal now is to show the limitations of the NN rule.

An immediate limitation follows directly from the dependence of the upper bound given in Theorem 17 on c (the Lipschitz coefficient of η) and on d (the dimension). In fact, it is easy to see that a necessary condition for the last term in Theorem 17 be smaller than ϵ is that $m \geq (4\,c\sqrt{d}/\epsilon)^{d+1}$. That is, the size of the training set should increase exponentially with the dimension. The following theorem tells us that this is not just an artifact of our upper bound but for some distributions this amount of examples is necessary for learning.

Theorem 18 *For any $c > 1$, there exists a distribution over $[0,1]^d \times \{0,1\}$, such that $\eta(\mathbf{x})$ is c-Lipschitz, the Bayes error is 0, but if $m \leq (c+1)^d/2$ the generalization error of the 1-NN rule is greater than $1/4$.*

Proof For simplicity, assume that c is an integer. Set the distribution over instances, $\mathcal{D}_\mathcal{X}$, to be uniform over the points of a grid on $[0,1]^d$ with distance of $1/c$ between points in the grid. That is, each point on the grid is of the form $(a_1/c, \ldots, a_d/c)$ where a_i is in $\{0, \ldots, c-1, c\}$. Note that no matter how we set $\eta(\mathbf{x})$ we will have that η is a c-Lipschitz function. The number of points on the grid is $(c+1)^d$, hence, if $m < (c+1)^d/2$, the conditions of Lemma 6 hold and this concludes our proof. ■

The exponential dependence on the dimension is known as the *curse of dimensionality*. As we saw, the 1-NN rule might fail if the number of examples is smaller than $\Omega(c^d)$. Therefore, while the 1-NN rule does not restrict itself to a predefined set of hypotheses, it still relies on a prior knowledge – the NN rule assumes that the dimension and the Lipschitz constant of η are not too high.

41 Efficient Implementation

Nearest Neighbor is a learning-by-memorization type of rule. It requires the entire training data set to be stored, and in test time, we need to scan the entire data set in order to find the neighbors. The time of applying the NN rule is therefore $\Theta(d\,m)$. This leads to expensive computation if the data set is large.

When d is small, several results from the field of computational geometry have proposed data structures that enable to apply the NN rule in time $o(d^{(O(1)}\log(m))$. However, the space required by these data structures is roughly $m^{O(d)}$, which makes these methods impractical for larger values of d.

To overcome this problem, it was suggested to improve the search method by allowing *approximate* search. Formally, an r-approximate search procedure is guaranteed to retrieve a point within distance of at most r times the distance to the nearest neighbor. Three popular approximate algorithms for NN are the kd-tree, balltrees, and locality-sensitive hashing (LSH). We refer the reader for example to the book: "Nearest-Neighbor Methods in Learning and Vision: Theory and Practice", Edited by Gregory Shakhnarovich, Trevor Darrell and Piotr Indyk.

(67577) Introduction to Machine Learning December 22, 2009

Lecture 13 – Validation

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In previous lectures we described various machine learning algorithms. The output of a learning algorithm is a predictor and we often would like to estimate the quality of the output predictor based on data. This process is called validation. Previously in the course we derived bounds on the difference between the training error and generalization error of all predictors in a given hypothesis class. In particular, these bounds hold for the output of the learning algorithm and we can therefore use the training error to estimate the generalization error.

Still, there are several reasons to apply a validation process that is different than the learning process. First, for some algorithms, like the Nearest Neighbor rule, the training error is not an indicator of the generalization error. In addition, even if a generalization bound holds for some algorithm, it is often looser than a direct validation bound. Last, in some situations we would like to use a validation process for choosing among several algorithms or for tuning the parameters of some method. This is called *model selection*. In that cases, the validation process is very similar to learning with a finite hypothesis class, except that the hypothesis class is itself a random variable that depends on the training data.

42 Hold out set

The simplest way to estimate the generalization error of a predictor h is by sampling an additional set of examples, independent of the training set, and use the empirical error on the validation set as our estimator. Formally, suppose that we learned h using some method based on a training set S of an arbitrary size. Let $V = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a set of fresh examples which are sampled according to the same distribution $\mathcal{D}$. Using Lemma 4 we have the following:

Theorem 19 *Let h be an arbitrary predictor and assume that the loss function satisfies $a \leq \ell(h, (\mathbf{x}, y)) \leq b$. Then, with probability of at least $1 - \delta$ over the choice of a validation set V we have*

$$|L_V(h) - L_{\mathcal{D}}(h)| \;\leq\; (b-a)\sqrt{\frac{\log(2/\delta)}{2\,|V|}}\,.$$

That is, the error on the validation set can be used to estimate the generalization error of h. We emphasize that the bound in Theorem 19 does not depend on the algorithm or the training set used to construct h. This is the reason why a fresh validation set can give an estimation of the error which is tighter than the error of h on the training set. The price is that we need to sample fresh examples.

Sampling a training set and then sampling an independent validation set is equivalent to randomly partitioning our random set of examples into two parts, using one part for training and the other one for validation. For this reason, the validation set is often referred to as a *hold out* set.

42.1 Validation for model selection

In some situations, we would like to choose among different learning algorithms or to tune the parameters of a learning algorithm. For example, we would like to use k-NN but we are not sure what value of k will give good results. Or, maybe we should use SVM and then what kernel should we employ and how we tune the parameters?

A common solution is to train the different algorithms with different parameters on the training set. Let $\mathcal{H} = \{h_1, \ldots, h_r\}$ be the set of all output predictors of the different algorithms. Now, to choose one predictor

from $\mathcal{H}$ we sample a fresh validation set and choose the predictor that minimizes the error over the validation set.

This process is very similar to learning a finite hypothesis class. The only difference is that $\mathcal{H}$ is not fixed ahead of time but rather depends on the training set. However, since the validation set is independent of the training set we get that it is also independent of $\mathcal{H}$ and therefore the same technique we used to derive bounds for finite hypothesis classes holds here as well. In particular, combining Theorem 19 with a union bound we obtain:

Theorem 20 *Let $\mathcal{H} = \{h_1, \ldots, h_k\}$ be an arbitrary set of predictors and assume that the loss function satisfies $a \leq \ell(h, (\mathbf{x}, y)) \leq b$ for all $h \in \mathcal{H}$ and $(\mathbf{x}, y)$. Assume that a validation set of size m is sampled independent of $\mathcal{H}$. Then, with probability of at least $1 - \delta$ we have*

$$\forall h \in \mathcal{H}, |L_V(h) - L_{\mathcal{D}}(h)| \;\leq\; (b-a)\sqrt{\frac{\log(2|\mathcal{H}|/\delta)}{2\,|V|}}\,.$$

The above theorem tells us that the error on the validation set approximates the generalization error as long as $\mathcal{H}$ is not too large. However, if we try too many methods ($|\mathcal{H}|$ is large) then overfitting might happen.

43 k-fold cross validation

The validation procedure described in the previous section assumes that data is plentiful and we can use part of the data as a fresh validation set. In some situations, data is scarce and we do not want to "waste" data on validation. The k-fold cross validation technique is designed to give an accurate estimate of the generalization error without wasting too much data.

In k fold cross validation the original training set is partitioned into k subsets (folds) of size m/k (for simplicity assume that m/k is an integer). For each fold, a predictor is trained on the other folds and then its error is estimated using the fold. The special case $k = m$, where m is the number of examples, is called *leave-one-out* (LOO).

k-fold cross validation is often used for model selection (or parameter tuning). In that case, the model is chosen by averaging the k estimations over the folds. Once the model is chosen, it is re-trained using the entire data set.

The cross validation method often works very well in practice. However, it can sometime fails as the following artificial example shows. Consider a case in which the label is chosen at random according to $\mathbb{P}[Y = 1] = \mathbb{P}[Y = 0] = 1/2$. Consider a method which outputs the constant predictor $h(\mathbf{x}) = 1$ if the parity of the labels on the training set is 1 and otherwise the method outputs the constant predictor $h(\mathbf{x}) = 0$. Then, the difference between the leave-one-out estimate and the generalization error is always $1/2$ (see Exercise 1)!

Rigorously understanding the exact behavior of cross validation is still an open problem. Rogers and Wagner showed that for k local rule (e.g. k nearest neighbor) the cross validation procedure gives a very good estimate of the generalization error. Other papers show that cross validation works for stable algorithms.

Exercises

1. Prove that the difference between the leave-one-out estimate and the generalization error in the parity example is always $1/2$.

(67577) Introduction to Machine Learning December 28, 2009

Lecture 15 – Dimensionality Reduction

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

Dimensionality reduction is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller. This process is closely related to the concept of (lossy) compression in information theory. There are several reasons to reduce the dimensionality of the data. First, high dimensional data impose computationally efficiency challenge. Moreover, in some situations high dimensionality might lead to poor generalization abilities of the learning algorithm (for example, in Nearest Neighbour classifiers the sample complexity increases exponentially with the dimension). Finally, dimensionality reduction can be used for interpretability of the data, for finding meaningful structure of the data, and for illustration purposes.

In this lecture we describe popular methods for dimensionality reduction. In those methods, the reduction is performed by applying a linear transformation to the original data. That is, if the original data is in $\mathbb{R}^d$ and we want to embed it into $\mathbb{R}^n$ $(n < d)$ then we would like to find a matrix $W \in \mathbb{R}^{n,d}$ that induces the mapping $\mathbf{x} \mapsto W\mathbf{x}$. A natural criterion for choosing W is in a way that will enable a reasonable recovery of the original $\mathbf{x}$. In exercise 1 we show that in the general case exact recovery of $\mathbf{x}$ from $W\mathbf{x}$ is impossible.

The first method we describe is called Principal Component Analysis (PCA). In PCA, the recovery is also a linear transformation and the method finds the compression and recovery linear transformations for which the difference between the recovered vectors and the original vectors is minimal in the least squared sense.

Next, we describe dimensionality reduction using random matrices W. We derive an important lemma, due to Johnson and Lindenstrauss, which analyzes the distortion caused by such a random dimensionality reduction technique.

Last, we show how one can reduce the dimension of a *sparse* vector using again a random matrix. This process is known as compressed sensing. In this case, the recovery process is non-linear but can still be implemented efficiently using linear programming.

44 Principal Component Analysis (PCA)

We would like to reduce the dimensionality of vectors in $\mathbb{R}^n$. A matrix $W \in \mathbb{R}^{n,d}$, where $n < d$, induces a mapping $\mathbf{x} \mapsto W\mathbf{x}$, where $W\mathbf{x} \in \mathbb{R}^n$ is the lower dimensionality representation of $\mathbf{x}$. Then, a second matrix $U \in \mathbb{R}^{d,n}$ can be used to (approximately) recover the original vector $\mathbf{x}$ from its compressed version. That is, for a compressed vector $\mathbf{y} = W\mathbf{x}$, where $\mathbf{y}$ is in the low dimensional space $\mathbb{R}^n$, we can construct $\tilde{\mathbf{x}} = U\mathbf{y}$, so that $\tilde{\mathbf{x}}$ is the recovered version of $\mathbf{x}$ and resides in the original high dimensional space $\mathbb{R}^d$.

Let $\mathbf{x}_1, \ldots, \mathbf{x}_m$ be m vectors in $\mathbb{R}^d$. In PCA, we find the compression matrix W and the recovering matrix U so that the total squared distance between original and recovered vectors is minimal. Namely, we aim at solving the problem:

$$\operatorname*{argmin}_{W \in \mathbb{R}^{n,d}, U \in \mathbb{R}^{d,n}} \sum_{i=1}^{m} \|\mathbf{x}_i - UW\mathbf{x}_i\|_2^2 \,. \tag{80}$$

To solve this problem we first show that the optimal solution takes a specific form.

Lemma 22 *Let (U, W) be a solution to Eq. (80). Then the columns of U are orthonormal (namely, $U^T U$ is the identity matrix of $\mathbb{R}^n$) and $W = U^T$.*

Proof Consider the mapping $\mathbf{x} \mapsto UW\mathbf{x}$. The range of this mapping, $R = \{UW\mathbf{x} : \mathbf{x} \in \mathbb{R}^d\}$, is an n dimensional linear subspace of $\mathbb{R}^d$. Let $\mathbf{z}_1, \ldots, \mathbf{z}_n$ be an orthonormal basis of this subspace. Namely, each $\mathbf{z}_i$ is in R and $\langle \mathbf{z}_i, \mathbf{z}_j \rangle = \mathbb{1}_{[i=j]}$. Consider the matrix Z whose columns are $\mathbf{z}_1, \ldots, \mathbf{z}_n$. Therefore, each vector

in R can be written as $Z\mathbf{y}$ where $\mathbf{y} \in \mathbb{R}^n$. Let $\mathbf{x} \in \mathbb{R}^d$ and let $\tilde{\mathbf{x}} \in R$ such that $\tilde{\mathbf{x}} = Z\mathbf{y}$ for some $\mathbf{y} \in \mathbb{R}^n$. We have

$$\|\mathbf{x} - Z\mathbf{y}\|_2^2 \;=\; \|\mathbf{x}\|^2 + \mathbf{y}^T Z^T Z\mathbf{y} - 2\mathbf{y}^T Z^T \mathbf{x} \;=\; \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\mathbf{y}^T (Z^T \mathbf{x}) \;,$$

where we used the fact that $Z^T Z$ is the identity matrix of $\mathbb{R}^n$. Minimizing the above expression with respect to $\mathbf{y}$ by comparing the gradient with respect to $\mathbf{y}$ to zero gives that $\mathbf{y} = Z^T \mathbf{x}$. Therefore,

$$Z\mathbf{y} = ZZ^T \mathbf{x} = \operatorname*{argmin}_{\tilde{\mathbf{x}} \in R} \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2 \;.$$

This holds for all $\mathbf{x}$ and in particular for $\mathbf{x}_1, \ldots, \mathbf{x}_m$, which concludes our proof. ■

Based on the above lemma, we can rewrite the optimization problem given in Eq. (80) as follows:

$$\operatorname*{argmin}_{U \in \mathbb{R}^{d,n} : U^T U = I} \sum_{i=1}^{m} \|(I - UU^T)\mathbf{x}_i\|_2^2 \;. \tag{81}$$

We further simplify the optimization problem by using the following elementary algebraic manipulations. For each $\mathbf{x} \in \mathbb{R}^d$ and a matrix U with orthonormal columns we have:

$$\begin{aligned}
\|(I - UU^T)\mathbf{x}\|_2^2 &= \mathbf{x}^T (I - UU^T)^T (I - UU^T)\mathbf{x} \\
&= \mathbf{x}^T (I - UU^T)(I - UU^T)\mathbf{x} \\
&= \mathbf{x}^T (I - UU^T - UU^T + UU^T UU^T)\mathbf{x} \\
&= \mathbf{x}^T (I - 2UU^T + UU^T)\mathbf{x} \\
&= \mathbf{x}^T (I - UU^T)\mathbf{x} \\
&= \|\mathbf{x}\|^2 - \operatorname{trace}(UU^T \mathbf{x}\mathbf{x}^T) \;,
\end{aligned} \tag{82}$$

where the trace of a matrix is the sum of its diagonal elements. Since the trace is a linear operator, the above allows us to rewrite Eq. (81) as follows:

$$\operatorname*{argmax}_{U \in \mathbb{R}^{d,n} : U^T U = I} \operatorname{trace}(UU^T \sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^T) \;. \tag{83}$$

Let $A = \sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^T$. The matrix A is symmetric and positive definite. It therefore has an eigenvalues decomposition of the form $A = \sum_{i=1}^d \lambda_i \mathbf{u}_i \mathbf{u}_i^T$ where $\lambda_1 \geq \lambda_2 \geq \ldots \geq 0$ and $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \mathbb{1}_{[i=j]}$. We claim that the solution to Eq. (83) is the matrix U whose columns are $\mathbf{u}_1, \ldots, \mathbf{u}_n$.

Theorem 21 *Let $\mathbf{x}_1, \ldots, \mathbf{x}_m$ be arbitrary vectors in $\mathbb{R}^m$, let $A = \sum_{i=1}^m \mathbf{x}_i \mathbf{x}_i^T$, and let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be n eigenvectors of the matrix A corresponding to the largest n eigenvalues of A. Then, the solution to the PCA optimization problem given in Eq. (80) is to set U to be the matrix whose columns are $\mathbf{u}_1, \ldots, \mathbf{u}_n$ and to set $W = U^T$.*

Proof As discussed previously, it suffices to prove that U solves Eq. (83). Clearly, U satisfies the constraint $U^T U = I$ and the value of the objective at U is

$$\operatorname{trace}(UU^T A) \;=\; \sum_{i=1}^{n} \mathbf{u}_i^T A \mathbf{u}_i \;=\; \sum_{i=1}^{n} \lambda_i \|\mathbf{u}_i\|^2 \;=\; \sum_{i=1}^{n} \lambda_i \;.$$

Take any $V \in \mathbb{R}^{k,n}$ that satisfies $V^T V = I$. We will show that

$$\operatorname{trace}(VV^T A) \;\leq\; \sum_{i=1}^{n} \lambda_i \;,$$

which will conclude our proof. To do so, we use Fan's inequality which states that for any two symmetric matrices B, A with $\rho_1 \geq \rho_2 \geq ... \geq \rho_d$ being the eigenvalues of B and $\lambda_1 \geq \ldots \geq \lambda_d$ being the eigenvalues of A we have

$$\text{trace}(BA) \ \leq \ \sum_{i=1}^{d} \rho_i \lambda_i \ .$$

In our case, $B = VV^T$. Let $\mathbf{v}_i$ be the ith column of V then $VV^T\mathbf{v}_i = \mathbf{v}_i$ and thus the columns of V are eigenvalues of the matrix VV^T with eigenvalues 1. The rest of the eigenvalues of VV^T are zeros (since the rank of VV^T is n). Therefore, $\sum_{i=1}^{d} \rho_i \lambda_i = \sum_{i=1}^{n} \lambda_i$ and this concludes our proof. ■

Remark 3 *The proof of Theorem 21 also tells us that the value of the objective of Eq. (83) is $\sum_{i=1}^{n} \lambda_i$, where λ_i is the i'th eigenvalue of A. Combining this with Eq. (82) and noting that $\sum_{i=1}^{m} \|\mathbf{x}_i\|^2 = \text{trace}(A) = \sum_{i=1}^{d} \lambda_i$ we obtain that the optimal objective value of Eq. (80) is $\sum_{i=n+1}^{d} \lambda_i$.*

Remark 4 *It is a common practice to "center" the examples before applying PCA. That is, we first calculate $\boldsymbol{\mu} = \sum_{i=1}^{m} \mathbf{x}_i$ and then apply PCA on the vectors $(\mathbf{x}_1 - \boldsymbol{\mu}), \ldots, (\mathbf{x}_m - \boldsymbol{\mu})$.*

45 Random Projections and Johnson-Lindenstrauss lemma

In this section we show that reducing the dimension by using a random linear transformation leads to a simple compression scheme with a surprisingly low distortion. The transformation $\mathbf{x} \mapsto W\mathbf{x}$, when W is a random matrix, is often referred to as a random projection. In particular, we provide a variant of a famous lemma due to Johnson and Lindenstrauss, showing that random projections do not distort Euclidean distances too much.

Let $\mathbf{x}_1, \mathbf{x}_2$ be two vectors in $\mathbb{R}^d$. A matrix W has a low distortion if the ratio

$$\frac{\|W\mathbf{x}_1 - W\mathbf{x}_2\|}{\|\mathbf{x}_1 - \mathbf{x}_2\|}$$

is close to 1. This means that the distances between $\mathbf{x}_1$ and $\mathbf{x}_2$ before and after the transformation are almost the same. To show that $\|W\mathbf{x}_1 - W\mathbf{x}_2\|$ is not too far away from $\|\mathbf{x}_1 - \mathbf{x}_2\|$ it suffices to show that W does not distort the norm of the difference vector $\mathbf{x}_1 - \mathbf{x}_2$. Therefore, from now on we focus on the ratio $\frac{\|W\mathbf{x}\|}{\|\mathbf{x}\|}$.

We start with analyzing the distortion caused by applying a random projection on a single vector.

Lemma 23 *Fix some $\mathbf{x} \in \mathbb{R}^d$. Let $W \in \mathbb{R}^{n,d}$ be a random matrix such that each $W_{i,j}$ is an independent normal random variable. Then, for any $\epsilon \in (0, 3)$ we have*

$$\mathbb{P}\left[\left| \frac{\|(1/\sqrt{n})W\mathbf{x}\|^2}{\|\mathbf{x}\|^2} - 1 \right| > \epsilon \right] \ \leq \ 2\, e^{-\epsilon^2 n/6} \ .$$

Proof Without loss of generality we can assume that $\|\mathbf{x}\|^2 = 1$. Therefore, an equivalent inequality is

$$\mathbb{P}\left[(1-\epsilon)n \leq \|W\mathbf{x}\|^2 \leq (1+\epsilon)n\right] \geq 1 - 2e^{-\epsilon^2 n/6} \ .$$

Let $\mathbf{z}_i$ be the ith row of W. The random variable $\langle \mathbf{z}_i, \mathbf{x} \rangle$ is a weighted sum of d independent normal random variables and therefore it is normally distributed with zero mean and variance $\sum_j x_j^2 = \|\mathbf{x}\|^2 = 1$. Therefore, the random variable $\|W\mathbf{x}\|^2 = \sum_{i=1}^{n} (\langle \mathbf{z}_i, \mathbf{x} \rangle)^2$ has a χ^2_n distribution. The claim now follows directly from a measure concentration property of χ^2 random variables stated in Lemma 25 in Section 45.1 below. ■

The Johnson-Lindenstrauss lemma follows from the above using a simple union bound argument.

Lemma 24 (Johnson-Lindenstrauss lemma) *Let Q be a finite set of vectors in $\mathbb{R}^d$. Let $\delta \in (0,1)$ and n be an integer such that*

$$\epsilon = \sqrt{\frac{6\,\ln(2|Q|/\delta)}{n}} \leq 3\,.$$

Then, with probability of at least $1-\delta$ over a choice of a random matrix $W \in \mathbb{R}^{n,d}$ such that each element of W is independently distributed according to $N(0,1/n)$ we have

$$\sup_{\mathbf{x}\in Q}\left|\frac{\|W\mathbf{x}\|^2}{\|\mathbf{x}\|^2}-1\right| < \epsilon\,.$$

Proof Using Lemma 23 and a union bound we have that for all $\epsilon \in (0,3)$:

$$\mathbb{P}\left[\sup_{\mathbf{x}\in Q}\left|\frac{\|W\mathbf{x}\|^2}{\|\mathbf{x}\|^2}-1\right| > \epsilon\right] \leq 2\,|Q|\,e^{-\epsilon^2 n/6}\,.$$

Let δ denote the right-hand side of the above and solve for ϵ we obtain that:

$$\epsilon = \sqrt{\frac{6\,\ln(2|Q|/\delta)}{n}}\,.$$

■

Interestingly, the bound given in Lemma 24 does not depend on the original dimension of $\mathbf{x}$. In fact, the bound holds even if $\mathbf{x}$ is in an infinite dimensional Hilbert space.

45.1 Concentration of χ^2 variables

Let $X_1,\ldots,X_k$ be k independent normally distributed random variables. That is, for all i, $X_i \sim N(0,1)$. The distribution of the random variable X_i^2 is called χ^2 (chi square) and the distribution of the random variable $Z = X_1^2+\ldots+X_k^2$ is called χ_k^2 (chi square with k degrees of freedom). Clearly, $\mathbb{E}[X_i^2]=1$ and $\mathbb{E}[Z]=k$. The following lemma states that X_k^2 is concentrated around its mean.

Lemma 25 *Let $Z \sim \chi_k^2$. Then, for all $\epsilon > 0$ we have*

$$\mathbb{P}[Z \leq (1-\epsilon)k)] \leq e^{-\epsilon^2 k/6}\,,$$

and for all $\epsilon \in (0,3)$ we have

$$\mathbb{P}[Z \geq (1+\epsilon)k)] \leq e^{-\epsilon^2 k/6}\,.$$

Finally, for all $\epsilon \in (0,3)$,

$$\mathbb{P}\left[(1-\epsilon)k \leq Z \leq (1+\epsilon)k\right] \geq 1-2e^{-\epsilon^2 k/6}\,.$$

Proof Let us write $Z = \sum_{i=1}^k X_i^2$ where $X_i \sim N(0,1)$. To prove both bounds we use Chernoff's bounding method. For the first inequality, we first bound $\mathbb{E}[e^{-\lambda X_1^2}]$, where $\lambda > 0$ will be specified later. Since $e^{-a} \leq 1-a+\frac{a^2}{2}$ for all $a \geq 0$ we have that

$$\mathbb{E}[e^{-\lambda X_1^2}] \;\leq\; 1-\lambda\,\mathbb{E}[X_1^2]+\frac{\lambda^2}{2}\,\mathbb{E}[X_1^4]\,.$$

Using the well known equalities, $\mathbb{E}[X_1^2]=1$ and $\mathbb{E}[X_1^4] \;=\; 3$, and the fact that $1-a \leq e^{-a}$ we obtain that

$$\mathbb{E}[e^{-\lambda X_1^2}] \;\leq\; 1-\lambda+\frac{3}{2}\lambda^2 \leq e^{-\lambda+\frac{3}{2}\lambda^2}\,.$$

Now, applying Chernoff's bounding method we get that

$$\begin{aligned}
\mathbb{P}[-Z \geq -(1-\epsilon)k)] &= \mathbb{P}\left[e^{-\lambda Z} \geq e^{-(1-\epsilon)k\lambda}\right] && (84)\\
&\leq e^{(1-\epsilon)k\lambda}\ \mathbb{E}\left[e^{-\lambda Z}\right] && (85)\\
&= e^{(1-\epsilon)k\lambda}\ \left(\mathbb{E}\left[e^{-\lambda X_1^2}\right]\right)^k && (86)\\
&\leq e^{(1-\epsilon)k\lambda}\, e^{-\lambda k+\frac{3}{2}\lambda^2 k} && (87)\\
&= e^{-\epsilon k\lambda+\frac{3}{2}k\lambda^2}\ . && (88)
\end{aligned}$$

Choose $\lambda = \epsilon/3$ we obtain the first inequality stated in the lemma.

For the second inequality, we use a known closed form expression for the moment generating function of a χ_k^2 distributed random variable:

$$\forall \lambda < \tfrac{1}{2},\ \ \mathbb{E}\left[e^{\lambda Z^2}\right] \ = \ (1-2\lambda)^{-k/2}\ . \tag{89}$$

Based on the above and using Chernoff's bounding method we have:

$$\begin{aligned}
\mathbb{P}[Z \geq (1+\epsilon)k)] &= \mathbb{P}\left[e^{\lambda Z} \geq e^{(1+\epsilon)k\lambda}\right] && (90)\\
&\leq e^{-(1+\epsilon)k\lambda}\ \mathbb{E}\left[e^{\lambda Z}\right] && (91)\\
&= e^{-(1+\epsilon)k\lambda}\ (1-2\lambda)^{-k/2} && (92)\\
&\leq e^{-(1+\epsilon)k\lambda}\, e^{k\lambda} = e^{-\epsilon k\lambda}\ , && (93)
\end{aligned}$$

where the last inequality is because $(1-a) \leq e^{-a}$. Setting $\lambda = \epsilon/6$ (which is in $(0, 1/2)$ by our assumption) we obtain the second inequality stated in the lemma.

Finally, the last inequality follows from the first two inequalities and the union bound. ■

46 Compressed Sensing

Compressed sensing is a dimensionality reduction technique which utilizes a prior assumption that the original vector is sparse in some basis. To motivate compressed sensing, consider a vector $\mathbf{x} \in \mathbb{R}^d$ that has at most s non-zero elements. That is,

$$\|\mathbf{x}\|_0 \stackrel{\text{def}}{=} |\{i : x_i \neq 0\}| \leq s\ .$$

Clearly, we can compress $\mathbf{x}$ by representing it using s (index,value) pairs. Furthermore, this compression is lossless – we can reconstruct $\mathbf{x}$ exactly from the s (index,value) pairs. Now, lets take one step forward and assume that $\mathbf{x} = U\boldsymbol{\alpha}$, where $\boldsymbol{\alpha}$ is a sparse vector, $\|\boldsymbol{\alpha}\|_0 \leq s$, and U is a fixed orthonormal matrix. That is, $\mathbf{x}$ has a sparse representation in another basis. It turns out that many natural vectors are (at least approximately) sparse in some representation. In fact, this assumption underlies many modern compression schemes. For example, the JPEG-2000 format for image compression relies on the fact that natural images are approximately sparse in a wavelet basis.

Can we still compress $\mathbf{x}$ into roughly s numbers? Well, one simple way to do this is to multiply $\mathbf{x}$ by U^T, which yields the sparse vector $\boldsymbol{\alpha}$, and then represent $\boldsymbol{\alpha}$ by its s (index,value) pairs. However, this requires to first 'sense' $\mathbf{x}$, to store it, and then to multiply it by U^T. This raises a very natural question: Why go to so much effort to acquire all the data when most of what we get will be thrown away? Can't we just directly measure the part that won't end up being thrown away?

Compressed sensing is a technique that simultaneously acquire and compress the data. The key result is that a random linear transformation can compress $\mathbf{x}$ without loosing information. The number of measurements needed is order of $s\log(d)$. That is, we roughly acquire only the important information about the

signal. As we will see later, the price we pay is a slower reconstruction phase. In some situations, it makes sense to save time in compression even at the price of a slower reconstruction. For example, a security camera should sense and compress a large amount of images while most of the time we do not need to decode the compressed data at all. Furthermore, in many practical applications, compression by a linear transformation is advantageous because it can be performed efficiently in hardware. For example, a team led by Baraniuk and Kelly have proposed a camera architecture that employs a digital micromirror array to perform optical calculations of a linear transformation of an image. In this case, obtaining each compressed measurement is as easy as obtaining a single raw measurement. Another important application of compressed sensing is medical imaging, in which requiring less measurements translates to less radiation for the patient.

We start by defining the so-called Restricted Isoperimetry Property (RIP) of matrices. A matrix that satisfies this property is guaranteed to have a low distortion of the norm of any sparse representable vector.

Definition 8 (RIP) *Let U be an orthonormal matrix $d \times d$ matrix. A matrix $W \in \mathbb{R}^{n,d}$ is (ϵ, s, U)-RIP if for all $\mathbf{x} \neq 0$ that can be written as $\mathbf{x} = U\boldsymbol{\alpha}$ such that $\|\boldsymbol{\alpha}\|_0 \leq s$ we have*

$$\left| \frac{\|W\mathbf{x}\|_2}{\|\mathbf{x}\|_2} - 1 \right| \leq \epsilon \ .$$

If U is the identity function we use the shorthand (ϵ, s)-RIP.

In Section 46.2 we show that a random matrix is (ϵ, s, U) RIP with high probability if $n = \tilde{\Omega}(s)$. The following theorem establishes that RIP matrices yield a lossless compression scheme for sparse representable vectors. It also provides a (non-efficient) reconstruction scheme.

Theorem 22 *Let $\epsilon < 1$ and let W be a $(\epsilon, 2s, U)$-RIP matrix. Let $\mathbf{x} = U\boldsymbol{\alpha}$ be a vector where $\|\boldsymbol{\alpha}\|_0 \leq s$. Let $\mathbf{y} = W\mathbf{x}$ be the compression of $\mathbf{x}$ and let*

$$\tilde{\mathbf{x}} \in \operatorname*{argmin}_{\mathbf{v}: W\mathbf{v}=\mathbf{y}} \|U^T\mathbf{v}\|_0$$

be a reconstructed vector. Then, $\tilde{\mathbf{x}} = \mathbf{x}$.

Proof We prove the theorem by assuming the contrary, namely assuming that $\tilde{\mathbf{x}} \neq \mathbf{x}$. Since $\mathbf{x}$ satisfies the constraints in the optimization problem that defines $\tilde{\mathbf{x}}$ we clearly have that $\|U^T\tilde{\mathbf{x}}\|_0 \leq \|U^T\mathbf{x}\|_0 = \|\boldsymbol{\alpha}\|_0 \leq s$. Consider the difference vector $\mathbf{x} - \tilde{\mathbf{x}}$. We can write it as

$$\mathbf{x} - \tilde{\mathbf{x}} = U(\boldsymbol{\alpha} - U^T\tilde{\mathbf{x}}) \ .$$

Since $\|\boldsymbol{\alpha} - U^T\tilde{\mathbf{x}}\|_0 \leq 2s$ we can apply the RIP inequality on the vector $\mathbf{x} - \tilde{\mathbf{x}}$. But, since $W(\mathbf{x} - \tilde{\mathbf{x}}) = \mathbf{0}$ we get that $|0 - 1| \leq \epsilon$, which leads to a contradiction. ■

It is important to emphasize that the reconstruction given in Theorem 22 does depend on U. In fact, different matrices U will lead to different reconstructed vectors.

The reconstruction scheme given in Theorem 22 seems to be non-efficient because we need to minimize a combinatorial objective (the sparsity of $U^T\mathbf{v}$). Quite surprisingly, it turns out that we can replace the combinatorial objective, $\|U^T\mathbf{v}\|_0$, with a convex objective, $\|U^T\mathbf{v}\|_1$, which leads to a linear programming problem that can be solved efficiently. This is stated formally in the following theorem, adapted from Candes and Tao, "Decoding by linear programming".

Theorem 23 *Let $\epsilon < 0.1$ and let W be a $(\epsilon, 4s, U)$-RIP matrix. Let $\mathbf{x} = U\boldsymbol{\alpha}$ be a vector where $\|\boldsymbol{\alpha}\|_0 \leq s$ and let $\mathbf{y} = W\mathbf{x}$ be the compression of $\mathbf{x}$. Then,*

$$\mathbf{x} \ = \ \operatorname*{argmin}_{\mathbf{v}: W\mathbf{v}=\mathbf{y}} \|U^T\mathbf{v}\|_0 \ = \ \operatorname*{argmin}_{\mathbf{v}: W\mathbf{v}=\mathbf{y}} \|U^T\mathbf{v}\|_1 \ .$$

Proof For simplicity, we prove the theorem for the case that U is the identity matrix. Let $\text{supp}(\mathbf{x}) = \{i \in [d] : x_i \neq 0\}$ be the support of $\mathbf{x}$. Denote $\tilde{\mathbf{x}}$ to be a solution of $\min_{\mathbf{v}:W\mathbf{v}=\mathbf{y}} \|\mathbf{v}\|_1$. We need to show that $\tilde{\mathbf{x}} = \mathbf{x}$.

Let $\mathbf{w}_1, \ldots, \mathbf{w}_d$ denote the columns of W. We use the following claim, whose proof can be found in Section 46.1 below.

Claim 1 *There exists $\mathbf{v} \in \mathbb{R}^n$ that satisfies:*

1. $\forall i \in \text{supp}(\mathbf{x}),\ \langle \mathbf{v}, \mathbf{w}_i \rangle = \textit{sign}(x_i)$
2. $\forall i \notin \text{supp}(\mathbf{x}),\ |\langle \mathbf{v}, \mathbf{w}_i \rangle| < 1$

Based on the above claim, we have:

$$\begin{aligned}
\|\tilde{\mathbf{x}}\|_1 &= \sum_{i \in \text{supp}(\mathbf{x})} |x_i + \tilde{x}_i - x_i| + \sum_{i \notin \text{supp}(\mathbf{x})} |\tilde{x}_i| && (94)\\
&\geq \sum_{i \in \text{supp}(\mathbf{x})} \text{sign}(x_i)(x_i + \tilde{x}_i - x_i) + \sum_{i \notin \text{supp}(\mathbf{x})} |\tilde{x}_i| && (95)\\
&= \sum_{i \in \text{supp}(\mathbf{x})} |x_i| + \sum_{i \in \text{supp}(\mathbf{x})} \text{sign}(x_i)(\tilde{x}_i - x_i) + \sum_{i \notin \text{supp}(\mathbf{x})} |\tilde{x}_i| && (96)\\
&= \|\mathbf{x}\|_1 + \sum_{i \in \text{supp}(\mathbf{x})} \langle \mathbf{v}, \mathbf{w}_i \rangle (\tilde{x}_i - x_i) + \sum_{i \notin \text{supp}(\mathbf{x})} |\tilde{x}_i| && (97)\\
&\geq \|\mathbf{x}\|_1 + \sum_{i \in \text{supp}(\mathbf{x})} \langle \mathbf{v}, \mathbf{w}_i \rangle (\tilde{x}_i - x_i) + \sum_{i \notin \text{supp}(\mathbf{x})} \tilde{x}_i \langle \mathbf{v}, \mathbf{w}_i \rangle && (98)\\
&= \|\mathbf{x}\|_1 + \langle \mathbf{v}, W\tilde{\mathbf{x}} - W\mathbf{x} \rangle && (99)\\
&= \|\mathbf{x}\|_1 && (100)\\
&\geq \|\tilde{\mathbf{x}}\|_1 . && (101)
\end{aligned}$$

This implies that all the inequalities in the above hold with equality. Moreover, since for $i \notin \text{supp}(\mathbf{x})$ we have that $|\langle \mathbf{v}, \mathbf{w}_i \rangle|$ is strictly less than 1 we obtain from Eq. (98) that $\tilde{x}_i$ must be 0 for all $i \notin \text{supp}(\mathbf{x})$. But, this implies that $\tilde{\mathbf{x}}$ is also s sparse (its support is a subset of the support of $\mathbf{x}$) and thus from the RIP condition we must have that $\tilde{\mathbf{x}} = \mathbf{x}$ (as in the proof of Theorem 22). ■

46.1 Proof of Claim 1

Given a matrix W and a set $I \subset [d]$, the notation W_I denotes the sub-matrix of W whose columns are taken from the set I. Similarly, given a vector $\boldsymbol{\alpha}$, the notation $\boldsymbol{\alpha}_I$ denote the sub-vector of $\boldsymbol{\alpha}$ whose elements are taken from I. We first need the following lemma which shows that RIP also implies approximate orthogonality.

Lemma 26 *Let W be an $(\epsilon, 4s)$-RIP matrix. Then, for any two disjoint sets J, J', both of size at most $2s$, we have that $\|W_{J'}^T W_J\| \leq 2\epsilon$, where $\|\cdot\|$ is the spectral norm.*

Proof Let $\sigma = \|W_{J'}^T W_J\|$. Since σ is the largest singular value of $W_{J'}^T W_J$, it means that there exist unit vectors, $\mathbf{u}$ and $\mathbf{u}'$ s.t.

$$(\mathbf{u}')^T W_{J'}^T W_J \mathbf{u} = \sigma .$$

In other words,

$$\sigma = \langle W_{J'}\mathbf{u}', W_J\mathbf{u} \rangle = \frac{\|W_{J'}\mathbf{u}' + W_J\mathbf{u}\|^2 - \|W_{J'}\mathbf{u}' - W_J\mathbf{u}\|^2}{4} .$$

But, since $|J \cup J'| \leq 4s$ we get from the RIP condition that $\|W_{J'}\mathbf{u}' + W_J\mathbf{u}\|^2 \leq (1+\epsilon)(\|\mathbf{u}\|^2 + \|\mathbf{u}'\|^2) = 2(1+\epsilon)$ and that $-\|W_{J'}\mathbf{u}' - W_J\mathbf{u}\|^2 \leq -(1-\epsilon)(\|\mathbf{u}\|^2 + \|\mathbf{u}'\|^2) = -2(1-\epsilon)$, which concludes our proof. ■

To prove Claim 1, we shall describe a process that generates $\mathbf{v}$ that satisfies the requirements of the claim. The basic building block is the following lemma.

Lemma 27 *Let W be an $(\epsilon, 4s)$-RIP matrix. Let $I \subset [d]$, $|I| \leq 2s$, and let $\boldsymbol{\alpha} \in \mathbb{R}^d$ be a vector s.t. $\mathrm{supp}(\boldsymbol{\alpha}) \subseteq I$. Let $\mathbf{v} = W_I(W_I^T W_I)^{-1}\boldsymbol{\alpha}_I$. Then, there exists $J \subset [d]$, disjoint from I, of size $|J| \leq s$ such that:*

1. $\|W_J^T \mathbf{v}\| \leq \frac{2\epsilon}{1-\epsilon}\|\boldsymbol{\alpha}\|$
2. *For all $i \notin I \cup J$ we have $|\langle \mathbf{v}, \mathbf{w}_i\rangle| \leq \frac{2\epsilon}{1-\epsilon} \cdot \frac{1}{\sqrt{s}} \cdot \|\boldsymbol{\alpha}\|$*
3. *For all $i \in I$ we have $\langle \mathbf{v}, \mathbf{w}_i\rangle = \alpha_i$*

Proof From the RIP condition we know that the eigenvalues of $W_I^T W_I$ are in $1 \pm \epsilon$. Therefore, $W_I^T W_I$ is invertible and thus

$$W_I^T \mathbf{v} = W_I^T W_I (W_I^T W_I)^{-1}\boldsymbol{\alpha}_I = \boldsymbol{\alpha}_I \,.$$

This proves the third claim of the lemma. Next, we show that for all sets J', disjoint from I and of size at most s we have $\|W_{J'}^T \mathbf{v}\| \leq \frac{2\epsilon}{1-\epsilon}\|\boldsymbol{\alpha}\|$. Indeed,

$$\|W_{J'}^T \mathbf{v}\| = \|W_{J'}^T W_I (W_I^T W_I)^{-1}\boldsymbol{\alpha}_I\| \leq \|W_{J'}^T W_I\| \, \|(W_I^T W_I)^{-1}\| \, \|\boldsymbol{\alpha}\| \leq \tfrac{2\epsilon}{1-\epsilon}\|\boldsymbol{\alpha}\| \,. \tag{102}$$

Finally, let

$$J = \{i \notin I \,:\, |\langle \mathbf{v}, \mathbf{w}_i\rangle| > \tfrac{2\epsilon}{1-\epsilon} \cdot \tfrac{1}{\sqrt{s}} \cdot \|\boldsymbol{\alpha}\|\} \,,$$

so it is left to show that $|J| \leq s$. But, this must be true since otherwise we could take $J' \subset J$ of size s and obtain that $\|W_{J'}^T \mathbf{v}\| > \frac{2\epsilon}{1-\epsilon}\|\boldsymbol{\alpha}\|$ which contradicts Eq. (102). ■

Equipped with Lemma 27 we now turn to prove claim 1. We first apply the lemma with the vector $\boldsymbol{\alpha}$ s.t. $\alpha_i = \mathrm{sign}(x_i)$ (where we use the convention $\mathrm{sign}(0) = 0$). Denote by $\mathbf{v}_1$ the resulting vector and by J_1 the "error set". Note that $\|\boldsymbol{\alpha}\| = \sqrt{\|\mathbf{x}\|_0} = \sqrt{s}$. Therefore, $\mathbf{v}_1$ 'almost' gives us the desired results — the value of $\langle \mathbf{v}_1, \mathbf{w}_i\rangle$ satisfies the required constraints whenever i is not in J_1. This is because,

1. $\|W_{J_1}^T \mathbf{v}_1\| \leq \frac{2\epsilon}{1-\epsilon}\sqrt{s}$
2. For all $i \notin \mathrm{supp}(\mathbf{x}) \cup J_1$ we have $|\langle \mathbf{v}_1, \mathbf{w}_i\rangle| \leq \frac{2\epsilon}{1-\epsilon}$
3. For all $i \in \mathrm{supp}(\mathbf{x})$ we have $\langle \mathbf{v}_1, \mathbf{w}_i\rangle = \mathrm{sign}(x_i)$

Next, we will gradually "correct" the vector $\mathbf{v}_1$ by applying Lemma 27 again, this time with $\boldsymbol{\alpha}$ be the vector whose value is $-\langle \mathbf{v}_1, \mathbf{w}_i\rangle$ for $i \in J_1$ and the rest of its elements are set to zero. Note that $\|\boldsymbol{\alpha}\| = \|W_{J_1}^T \mathbf{v}_1\| \leq \frac{2\epsilon}{1-\epsilon}\sqrt{s}$. Therefore, Lemma 27 guarantees the existence of $\mathbf{v}_2$ and a set J_2 such that

1. $\|W_{J_2}^T \mathbf{v}_2\| \leq \left(\frac{2\epsilon}{1-\epsilon}\right)^2 \sqrt{s}$
2. For all $i \notin \mathrm{supp}(\mathbf{x}) \cup J_1 \cup J_2$ we have $|\langle \mathbf{v}_2, \mathbf{w}_i\rangle| \leq \left(\frac{2\epsilon}{1-\epsilon}\right)^2$
3. For all $i \in J_1$ we have $\langle \mathbf{v}_2, \mathbf{w}_i\rangle = -\langle \mathbf{v}_1, \mathbf{w}_i\rangle$
4. For all $i \in \mathrm{supp}(\mathbf{x})$ we have $\langle \mathbf{v}_2, \mathbf{w}_i\rangle = 0$

Continuing this k times we obtain that for each $t \in \{2, \ldots, k\}$ we have

1. $\|W_{J_t}^T \mathbf{v}_t\| \leq \left(\frac{2\epsilon}{1-\epsilon}\right)^t \sqrt{s}$
2. For all $i \notin \text{supp}(\mathbf{x}) \cup J_{t-1} \cup J_t$ we have $|\langle \mathbf{v}_t, \mathbf{w}_i \rangle| \leq \left(\frac{2\epsilon}{1-\epsilon}\right)^t$
3. For all $i \in J_{t-1}$ we have $\langle \mathbf{v}_t, \mathbf{w}_i \rangle = -\langle \mathbf{v}_{t-1}, \mathbf{w}_i \rangle$
4. For all $i \in \text{supp}(\mathbf{x})$ we have $\langle \mathbf{v}_t, \mathbf{w}_i \rangle = 0$

Therefore, setting $\mathbf{v} = \sum_{t=1}^k \mathbf{v}_t$ and denoting $a = \sum_{t=1}^k \left(\frac{2\epsilon}{1-\epsilon}\right)^t$, we obtain that $\mathbf{v}$ satisfies the following:

1. For all $i \in J_k$ we have $|\langle \mathbf{v}, \mathbf{w}_i \rangle| \leq \|W_{J_k}^T \mathbf{v}_k\| + a \leq \left(\frac{2\epsilon}{1-\epsilon}\right)^k \sqrt{s} + a$
2. For all $i \in \text{supp}(\mathbf{x})$ we have $\langle \mathbf{v}, \mathbf{w}_i \rangle = \text{sign}(x_i)$
3. For all $i \notin \text{supp}(\mathbf{x}) \cup J_k$ we have $|\langle \mathbf{v}, \mathbf{w}_i \rangle| \leq a$

Finally, since

$$a \leq \frac{2\epsilon}{1-\epsilon} \frac{1}{1 - \frac{2\epsilon}{1-\epsilon}} = \frac{2\epsilon}{1 - (\epsilon + 2\epsilon)} < 1/2 \,,$$

if we set k to be large enough we obtain that all the conditions of the claim are satisfied and this concludes our proof.

46.2 Random Matrices are likely to be RIP

Theorem 24 *Let U be an orthonormal $d \times d$ matrix, $\epsilon \in (0,1)$ be a scalar and s be an integer. A random matrix $W \in \mathbb{R}^{n,d}$ satisfies the (ϵ, s, U) RIP condition with probability of at least $1 - \delta$ if the following holds:*

$$n \geq s \cdot \frac{24 \, (\ln(d) + \ln(2/\delta) + \ln(20/\epsilon))}{\epsilon^2} \,.$$

To prove this theorem we follow the approach of Baraniuk, Davenport, DeVore, and Wakin, "A simple proof of the RIP for random matrices". The idea is to combine Johnson-Lindenstrauss lemma with a simple covering argument.

We start with a covering property of the unit ball.

Lemma 28 *Let $\epsilon \in (0,1)$. There exists a finite set $Q \subset \mathbb{R}^d$ of size $|Q| \leq \left(\frac{5}{\epsilon}\right)^d$ such that*

$$\sup_{\mathbf{x}: \|\mathbf{x}\| \leq 1} \min_{\mathbf{v} \in Q} \|\mathbf{x} - \mathbf{v}\| \leq \epsilon \,.$$

Proof Let k be an integer and let

$$Q' = \{\mathbf{x} \in \mathbb{R}^d : \forall j, \exists i \in \{-k, -k+1, \ldots, k\} \text{ s.t. } x_j = \tfrac{i}{k}\} \,.$$

Clearly, $|Q'| = (2k+1)^d$. We shall set $Q = Q' \cap B_2(1)$, where $B_2(1)$ is the unit L_2 ball of $\mathbb{R}^d$. Since the points in Q' are distributed evenly on the unit cube, the size of Q is the size of Q' times the ratio between the volumes of the unit L_2 ball and the unit cube. The volume of the unit cube is 1 and the volume of $B_2(1)$ is

$$\frac{\pi^{d/2}}{\Gamma(1 + d/2)} \,.$$

For simplicity, assume that d is even and therefore

$$\Gamma(1+d/2) \;=\; (d/2)! \;\geq\; \left(\tfrac{d/2}{e}\right)^{d/2} \;,$$

where in the last inequality we used Stirling's approximation. Overall we obtained that

$$|Q| \leq (2k+1)^d \, (\pi/e)^{d/2} \, (d/2)^{-d/2} \;. \tag{103}$$

Now lets specify k. For each $\mathbf{x} \in B_2(1)$ let $\mathbf{v} \in Q$ be the vector whose ith element is $\text{sign}(x_i) \, \lfloor |x_i| \, k \rfloor$. Then, for each element we have that $|x_i - v_i| \leq 1/k$ and thus

$$\|\mathbf{x} - \mathbf{v}\| \leq \frac{\sqrt{d}}{k} \;.$$

To ensure that the right-hand side of the above will be at most ϵ we shall set $k = \lceil \sqrt{d}/\epsilon \rceil$. Plugging this value into Eq. (103) we conclude that

$$|Q| \leq (3\sqrt{d}/\epsilon)^d \, (\pi/e)^{d/2} \, (d/2)^{-d/2} = \left(\tfrac{3}{\epsilon}\sqrt{\tfrac{2\pi}{e}}\right)^d \leq \left(\tfrac{5}{\epsilon}\right)^d \;.$$

■

Let $\mathbf{x}$ be a vector that can be written as $\mathbf{x} = U\boldsymbol{\alpha}$ with U being some orthonormal matrix and $\|\boldsymbol{\alpha}\|_0 \leq s$. Combining the covering property above and the JL lemma (Lemma 24) enables us to show that a random W will not distort any such $\mathbf{x}$.

Lemma 29 *Let U be an orthonormal $n \times d$ matrix and let $I \subset [d]$ be a set of indices of size $|I| = s$. Let S be the span of $\{U_i : i \in I\}$, where U_i is the ith column of U. Let $\delta \in (0,1)$, $\epsilon \in (0,1)$, and n be an integer such that*

$$n \geq 24 \, \frac{\ln(2/\delta) + s \, \ln(20/\epsilon)}{\epsilon^2} \;.$$

Then, with probability of at least $1-\delta$ over a choice of a random matrix $W \in \mathbb{R}^{n,d}$ such that each element of W is independently distributed according to $N(0, 1/n)$ we have

$$\sup_{\mathbf{x} \in S} \left| \frac{\|W\mathbf{x}\|}{\|\mathbf{x}\|} - 1 \right| < \epsilon \;.$$

Proof It suffices to prove the lemma for all $\mathbf{x} \in S$ of unit norm. We can write $\mathbf{x} = U_I\boldsymbol{\alpha}$ where $\boldsymbol{\alpha} \in \mathbb{R}^s$, $\|\boldsymbol{\alpha}\|_2 = 1$, and U_I is the matrix whose columns are $\{U_i : i \in I\}$. Using Lemma 28 we know that there exists a set Q of size $|Q| \leq (20/\epsilon)^s$ such that

$$\sup_{\boldsymbol{\alpha}:\|\boldsymbol{\alpha}\|=1} \min_{\mathbf{v} \in Q} \|\boldsymbol{\alpha} - \mathbf{v}\| \leq (\epsilon/4) \;.$$

But, since U is orthogonal we also have that

$$\sup_{\boldsymbol{\alpha}:\|\boldsymbol{\alpha}\|=1} \min_{\mathbf{v} \in Q} \|U_I\boldsymbol{\alpha} - U_I\mathbf{v}\| \leq (\epsilon/4) \;.$$

Applying Lemma 24 on the set $\{U_I\mathbf{v} : \mathbf{v} \in Q\}$ we obtain that for n satisfying the condition given in the lemma, the following holds with probability of at least $1-\delta$:

$$\sup_{\mathbf{v} \in Q} \left| \frac{\|WU_I\mathbf{v}\|^2}{\|U_I\mathbf{v}\|^2} - 1 \right| \leq \epsilon/2 \;,$$

This also implies that

$$\sup_{\mathbf{v}\in Q}\left|\frac{\|WU_I\mathbf{v}\|}{\|U_I\mathbf{v}\|}-1\right|\leq \epsilon/2\,.$$

Let a be the smallest number such that

$$\forall \mathbf{x}\in S,\ \ \frac{\|W\mathbf{x}\|}{\|\mathbf{x}\|}\leq 1+a\,.$$

Clearly $a<\infty$. Our goal is to show that $a\leq\epsilon$. This follows from the fact that for any $\mathbf{x}\in S$ of unit norm there exists $\mathbf{v}\in Q$ such that $\|\mathbf{x}-U_I\mathbf{v}\|\leq\epsilon/4$ and therefore

$$\|W\mathbf{x}\|\leq\|WU_I\mathbf{v}\|+\|W(\mathbf{x}-U_I\mathbf{v})\|\leq 1+\epsilon/2+(1+a)\epsilon/4\,.$$

Thus,

$$\forall \mathbf{x}\in S,\ \ \frac{\|W\mathbf{x}\|}{\|\mathbf{x}\|}\leq 1+(\epsilon/2+(1+a)\epsilon/4)\ .$$

But, the definition of a implies that

$$a\leq\epsilon/2+(1+a)\epsilon/4\ \ \Rightarrow\ \ a\leq\frac{\epsilon/2+\epsilon/4}{1-\epsilon/4}\leq\epsilon\,.$$

This proves that for all $\mathbf{x}\in S$ we have $\frac{\|W\mathbf{x}\|}{\|\mathbf{x}\|}-1\leq\epsilon$. The other side follows from this as well since

$$\|W\mathbf{x}\|\geq\|WU_I\mathbf{v}\|-\|W(\mathbf{x}-U_I\mathbf{v})\|\geq 1-\epsilon/2-(1+\epsilon)\epsilon/4\geq 1-\epsilon\,.$$

■

The proof of Theorem 24 follows from the above by a union bound over all choices of I.

Exercises

1. In this exercise we show that in the general case, exact recovery of a linear compression scheme is impossible.
   (a) let $A\in\mathbb{R}^{n,d}$ be an arbitrary compression matrix where $n\leq d-2$. Show that there exists $\mathbf{u},\mathbf{v}\in\mathbb{R}^n$, $\mathbf{u}\neq\mathbf{v}$ such that $A\mathbf{u}=A\mathbf{v}=\mathbf{0}$.
   (b) Show that for any function $f:\mathbb{R}^n\to\mathbb{R}^d$ we have
   $$\|\mathbf{u}-f(A\mathbf{u})\|^2+\|\mathbf{v}-f(A\mathbf{v})\|^2>0\,.$$
   (c) Conclude that exact recovery of a linear compression scheme is impossible.

(67577) Introduction to Machine Learning January 4, 2010

Lecture 16 – Clustering

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

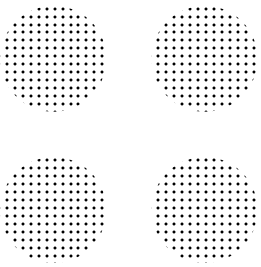
Clustering is one of the most widely used techniques for exploratory data analysis. Across all disciplines, from social sciences over biology to computer science, people try to get a first intuition about their data by identifying meaningful groups among the data points. Examples of tasks to which clustering is applied include:

- Computational biologists cluster genes according to similarities based on their expression in different experiments.
- Retailers cluster customers, based on their customer profiles, for the purpose of targeted marketing.
- Astronomers cluster stars based on their spacial proximity.

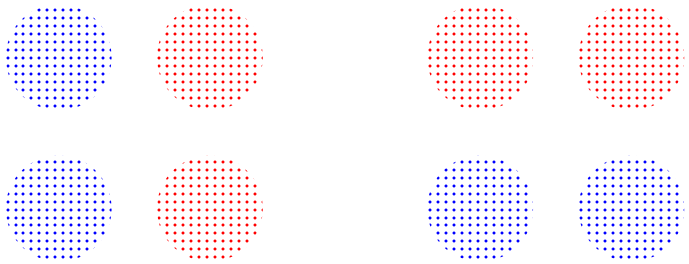
The first point that one should clarify is, naturally, what is clustering? Intuitively, clustering is the task of grouping a set of objects such that similar objects end up in the same group and dissimilar objects are separated into different groups. Quite surprisingly, it is not at all clear how to come up with a more rigorous definition.

There are several sources for this difficulty. One basic problem is that the two objectives mentioned in the above statement are quite different and, in many cases, contradict each other. Mathematically speaking, similarity (or proximity) is not a transitive relation, while cluster sharing is an equivalence relation and, in particular, it is a transitive relation. More concretely, it may be the case that there is a long sequence of objects, $x_1, \ldots, x_m$ such that each x_i is very similar to its two neighbors, x_{i-1} and x_{i+1}, but x_1 and x_m are very dissimilar. If we wish to make sure that whenever two elements are similar they share the same cluster, then we must put all of the elements of the sequence in the same cluster. However, in that case, we end up with dissimilar elements (x_1 and x_m) sharing a cluster, thus violating the second requirement.

Another basic problem is the lack of "ground truth" for clustering. Consider for example the following set of points in $\mathbb{R}^2$



and suppose we are required to cluster them into two clusters. We have two well justifiable solutions:



This phenomenon is not artificial but occurs in real applications. E.g., clustering recordings of speech by the accent of the speaker vs. clustering them by content, clustering movie reviews by movie topic vs. clustering them by the review sentiment, clustering paintings by topic vs. clustering them by style, etc.

Previously in the course, we dealt with *supervised* learning, e.g. the problem of learning a classifier. When we learn a classifier the goal is clear - we wish to minimize the error (or, more generally, the loss) of our classifier. Furthermore, a supervised learner can estimate the success of its hypothesis classifier against the labeled training data (or a hold out subset of that). In contrast with that, no such success evaluation procedure is available for clustering algorithms. Learning can be viewed as a process by which one tries to deduce properties of some data distribution from samples of that distribution. For clustering, however, even on the basis of full knowledge of the underlying data distribution, it is not clear what is the "correct" clustering for that data or how to evaluate a proposed clustering. This aspect is often referred to by the term "*un-supervised learning*".

Given a data set there may be several very different conceivable clustering solutions for that data. Consequently, there is a wide variety of clustering algorithms that are likely to output very different clusterings for a single given input. Most of the common clustering algorithms are defined for the following setup:

Input — a set of elements, $\mathcal{X}$, and a distance function over it. That is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ that is symmetric, satisfies $d(x, x) = 0$ for all $x \in \mathcal{X}$ and often also satisfies the triangle inequality (alternatively, the input can come in the form of a similarity function $s : \mathcal{X} \times \mathcal{X} \to [0, 1]$ that is symmetric and satisfies $s(x, x) = 1$ for all $x \in \mathcal{X}$). Additionally, some clustering algorithms also require an input parameter k (determining the number of required clusters).

Output — a partition of the domain set $\mathcal{X}$ into k subsets. That is, $C = (C_1, \ldots C_k)$ where $\bigcup_{i=1}^{k} C_i = \mathcal{X}$ and for all $i \neq j$, $C_i \cap C_j = \emptyset$. In some situations the clustering is "soft", namely, the partition of $\mathcal{X}$ into the different clusters is probabilistic where $p(\mathbf{x} \in C_i)$ is the probability that $\mathbf{x}$ belongs to class C_i. Another possible output is a clustering *dendrogram* (from Greek dendron = tree, gramma = drawing), which is a hierarchical tree of domain subsets, having the singleton sets in its leaves, and the full domain as its root.

We list below some of the most common clustering methods.

47 Linkage-based clustering algorithms

Linkage-based clustering is probably the simplest and most straightforward paradigm of clustering. These algorithms proceed in a sequence of rounds. They start from the trivial clustering that has each data point as a single-point cluster. Then, repeatedly, these algorithm merge the two closest clusters of the previous clustering. Consequently, the number of clusters decreases with each such round. If kept going, such algorithms would eventually result in the trivial clustering in which all of the domain points share one large cluster. There are two points that need to be more clearly defined. First, we have to decide how to measure (or define) the distance between clusters. recall that the input to a clustering algorithm is a between-points distance, d. There are three common ways of doing that:

1. Single Linkage clustering, in which the between clusters distance if defined by the minimum distance between members of the two clusters. Namely,

$$D(A, B) \stackrel{def}{=} \min\{d(x, y) : x \in A,\ y \in B\}$$

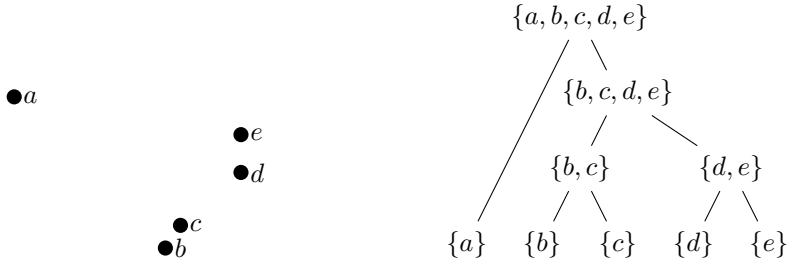
2. Average Linkage clustering, in which the distance between two clusters is defined to be the average distance between a point in one of the clusters and a point in the other. Formally,

$$D(A, B) = \frac{1}{|A||B|} \sum_{x \in A,\ y \in B} d(x, y)$$

3. Max Linkage clustering, in which the distance between two clusters is defined as the maximum distance between their elements. Namely,

$$D(A,B) \stackrel{def}{=} \max\{d(x,y) : x \in A,\ y \in B\}.$$

The linkage based clustering algorithms are *agglomerative* in the sense that they start from the data being completely fragmented and keep building larger and larger clusters as they proceed. The output is a clustering *dendrogram*, which is a tree of domain subsets, having the singleton sets in its leaves, and the full domain as its root. For example, if the input is the elements $\mathcal{X} = \{a, b, c, d, e\} \subset \mathbb{R}^2$ with the Euclidean distance as depicted on the left, then the resulting dendrogram is the one depicted on the right:



It is possible to show that the tree produced by the single linkage clustering is a minimal spanning tree on the full graph whose vertices are elements of $\mathcal{X}$ and the weight of an edge (x, y) is the distance $d(x, y)$.

If one wishes to turn a method that returns a dendrogram into a partition of the space, one needs to define a *stoping criteria*. Common stoping criteria include

- Fixed number of clusters - fix some parameter, k, and stop merging clusters as soon as the number of clusters is k.
- Distance upper bound - fix some $r \in \mathbb{R}^+$. Stop merging as soon as all the between clusters distances are smaller than r. We can also set r to be $\alpha \max\{d(x, y) : x, y \in \mathcal{X}\}$ for some $\alpha < 1$. In that case the stopping criterion is called "scaled distance upper bound".

48 k-means and variants

Another popular approach to clustering defines a cost function over a parameterized set of possible clusterings and the goal of the clustering algorithm is to find a partitioning (clustering) of minimal cost. Under this paradigm, the clustering problem is turned into an optimization problem. An objective function defines a *goal* for clustering, but in order to reach that goal, one has to apply some appropriate search algorithm. As it turns out, most of the resulting optimization problems are NP-hard, and some are even NP-hard to approximate. Consequently, when people talk about, say, k-Means clustering they often refer to some particular common approximation algorithm rather than the cost function or the corresponding exact solution of the minimization problem. Examples of objective functions:

The k-Means objective function is one of the most popular clustering objective. In k-means the data is partitioned into disjoint sets $C_1, \ldots, C_k$ where each C_i is represented by a centroid μ_i. It is assumed that the input set $\mathcal{X}$ is embedded in some larger metric space $(\mathcal{X}', d)$ (so that $\mathcal{X} \subseteq \mathcal{X}'$) and centroids are members of $\mathcal{X}'$. A point $x \in \mathcal{X}$ is associated with cluster C_i if $d(x, \mu_i)$ is minimal. The k-mean objective function measures the squared distance between each point in $\mathcal{X}$ to the centroid of its cluster:

$$\min_{\mu_1,\ldots\mu_k \in \mathcal{X}'} \sum_{x \in \mathcal{X}} \min_{i \in [k]} d(x, \mu_i)^2$$

The k-medoids objective function is similar to the k-means objective, except that it requires the cluster centroids to be members of the input set. The objective function is defined by

$$\min_{\mu_1,\ldots\mu_k\in\mathcal{X}} \sum_{x\in\mathcal{X}} \min_{i\in[k]} d(x,\mu_i)^2$$

The k-Median objective function is quite similar to the k-means objective, except that the "distortion" between a data point and the centroid of its cluster is measured by distance, rather than by the square of the distance:

$$\min_{\mu_1,\ldots\mu_k\in\mathcal{X}'} \sum_{x\in\mathcal{X}} \min_{i\in[k]} d(x,\mu_i)$$

An example where such an objective makes sense is the *facility location* problem. Consider the task of locating k-many fire stations in a city. One can model houses as data points and aim to place the stations so as to minimize the average distance between a house and its closest fire station.

48.1 The k-Means algorithm

The k-Means objective function is quite popular in practical applications of clustering. However, it turns out that finding the optimal k-Means solution is often computationally infeasible (the problem is NP-hard, and even NP-hard to approximate to within some constant). As an alternative, the following simple iterative algorithm is often used. So often that, in many cases, the term k-Means Clustering refers to the outcome of this algorithm rather than to the clustering that minimizes the k-Means objective cost. We describe the algorithm w.r.t. the distance function $d(x,y) = \|x-y\|$.

Algorithm 8 k-Means

Input: $\mathcal{X} \subset \mathbb{R}^n$; Number of clusters k
Initialize: Randomly choose initial centroids $\mu_1, \ldots, \mu_k$
Repeat until convergence
$\forall i \in [k]$ set $C_i = \{x \in \mathcal{X} : \|x-\mu_i\| = \min_j \|x-\mu_j\|\}$
$\forall i \in [k]$ update $\mu_i = \frac{1}{|C_i|}\sum_{x\in C_i} x$

Lemma 30 *The k-Means stops after a finite number of iterations.*

Proof We first show that each iteration decreases the k-Means objective function. Indeed, let $\mu_1,\ldots,\mu_k$ be the centroids before the update and let $C_1,\ldots,C_k$ be the corresponding clusters. For each i, let μ_i' be the new centroid. Since $\mu_i' = \operatorname{argmin}_\mu \sum_{x\in C_i}\|x-\mu\|^2$ it follows that

$$\sum_{x\in C_i}\min_{j\in[k]}\|x-\mu_j'\|^2 \le \sum_{x\in C_i}\|x-\mu_i'\|^2 \le \sum_{x\in C_i}\|x-\mu_i\|^2 = \sum_{x\in C_i}\min_{j\in[k]}\|x-\mu_j\|^2 .$$

Since the above holds for all i we obtain that the objective is non-increasing. Now, if we didn't stop, then the partition is strictly different after the iteration, which decreasing whenever we make an actual change of the partition function. The proof now follows from the fact that the number of different partitions of the data is finite, so the algorithm will not visit the same partition twice in its run. ■

We note however that the number of iterations required to reach convergence can be exponentially large, and furthermore, there it no any non-trivial lower bound on the gap between the value of the k-Means objective of the algorithm's output and the minimum possible value of that objective function. To improve the results of k-Means it is often recommended to repeat the procedure several times with different randomly chosen initial centroids (e.g., we can choose the centroids to be points from the data).

49 Spectral clustering

A nice way of representing the data points $x_1, \ldots, x_m$ is by a *similarity graph*; Each vertex represents a data point x_i, every two vertices are connected by an edge whose weight is their similarity, $W_{i,j} = s(x_i, x_j)$, where $W \in \mathbb{R}^{m,m}$. For example, we can set $W_{i,j} = \exp(-d(x_i, x_j)^2/\sigma^2)$, where $d(\cdot,\cdot)$ is a distance function and σ is a parameter. The clustering problem can now be formulated as follows: we want to find a partition of the graph such that the edges between different groups have low weights and the edges within a group have high weights.

In the clustering objectives described previously, the focus was on one side of our intuitive definition of clustering — making sure that points in the same cluster are similar. We now present objectives that focus on the other requirement — points separated into different clusters should be non similar.

49.1 Graph cut

Given a graph represented by a similarity matrix W, the simplest and most direct way to construct a partition of the graph is to solve the mincut problem, which chooses a partition $C_1, \ldots, C_k$ which minimizes the objective

$$\text{cut}(C_1, \ldots, C_k) = \sum_{i=1}^{k} \sum_{r \in C_i, s \notin C_i} W_{r,s} \; .$$

For $k = 2$, the mincut problem can be solved efficiently. However, in practice it often does not lead to satisfactory partitions. The problem is that in many cases, the solution of mincut simply separates one individual vertex from the rest of the graph. Of course this is not what we want to achieve in clustering, as clusters should be reasonably large groups of points.

Several solutions to this problem has been suggested. The simplest solution is to normalize the cut and define the normalized mincut objective as follows:

$$\text{RatioCut}(C_1, \ldots, C_k) = \sum_{i=1}^{k} \frac{1}{|C_i|} \sum_{r \in C_i, s \notin C_i} W_{r,s} \; .$$

The above objective take smaller values if the clusters are not too small. Unfortunately, introducing the balancing makes the problem hard to solve. Spectral clustering is a way to relax the problem of minimizing RatioCut.

49.2 Graph Laplacian and Relaxed Graph Cuts

The main mathematical object for spectral clustering is the graph Laplacian matrix. Unfortunately, there are several different definitions of graph Laplacian in the literature. Below we describe one particular definition. For a more complete overview see the excellent tutorial on spectral clustering by Ulrike von Luxburg.

Definition 9 (Unnormalized Graph Laplacian) *The* unnormalized graph Laplacian *is the matrix $L = D - W$ where D is a diagonal matrix with $D_{i,i} = \sum_{j=1}^{m} W_{i,j}$. The matrix D is called the* degree matrix.

The following lemma underscores the relation between RatioCut and the Laplacian matrix.

Lemma 31 *Let $C_1, \ldots, C_k$ be a clustering and let $H \in \mathbb{R}^{m,k}$ be the matrix such that*

$$H_{i,j} = \tfrac{1}{|C_j|} \mathbb{1}_{[i \in C_j]} \; .$$

Then, the columns of H are orthonormal to each other and

$$\textit{RatioCut}(C_1, \ldots, C_k) = \text{trace}(L\, H H^T) \; .$$

Proof Let $\mathbf{h}_1, \ldots, \mathbf{h}_k$ be the columns of H. The fact that these vectors are orthonormal is trivial. Next, note that $\mathrm{trace}(L\,HH^T) = \sum_{i=1}^{k} \mathbf{h}_i^T L \mathbf{h}_i$ and that for any vector $\mathbf{v}$ we have

$$\mathbf{v}^T L \mathbf{v} = \frac{1}{2}\left(\sum_i D_{i,i} v_i^2 - 2\sum_{i,j} v_i v_j W_{i,j} + \sum_j D_{j,j} v_j^2\right) = \frac{1}{2}\sum_{i,j} W_{i,j}(v_i - v_j)^2 \ .$$

Since $(h_{i,r} - h_{i,s})^2$ is non-zero only if $r \in C_i, s \notin C_i$ or the other way around, we obtain that,

$$\mathbf{h}_i^T L \mathbf{h}_i \ = \ \frac{1}{|C_i|} \sum_{r \in C_i, s \notin C_i} W_{r,s} \ .$$

■

Therefore, to minimize RatioCut we can search for a matrix H whose columns are orthonormal and such that each $H_{i,j}$ is either 0 or $1/|C_j|$. Unfortunately, this is an integer programming problem which we cannot solve. Instead, we relax the latter requirement and simply search an orthonormal matrix $H \in \mathbb{R}^{m,k}$ that minimizes $\mathrm{trace}(L\,HH^T)$. As we have shown in the lecture about PCA (particularly, the proof of Theorem 21), the solution to this problem is to set U to be the matrix whose columns are the eigenvectors corresponding to the k minimal eigenvalues of L. The resulting algorithm is called unnormalized spectral clustering.

49.3 Unnormalized spectral clustering

Algorithm 9 Unnormalized spectral clustering

Input: $W \in \mathbb{R}^{m,m}$; Number of clusters k
Initialize: Compute the unnormalized graph Laplacian L
Let $U \in \mathbb{R}^{m,k}$ be the matrix whose columns are the eigenvectors of L corresponding to minimal k eigenvalues
Let $\mathbf{v}_1, \ldots, \mathbf{v}_m$ be the rows of U
Cluster the points $\mathbf{v}_1, \ldots, \mathbf{v}_m$ using k-Means
Output: Clusters $C_1, \ldots, C_K$ of the k-Means algorithm

The spectral clustering algorithm starts with finding the matrix H of the first k eigenvectors of the graph Laplacian matrix and representing points according to rows of H. It is due to the properties of the graph Laplacians that this change of representation is useful. In many situations, this change of representation enables the simple k-Means algorithm to detect the clusters seamlessly. Intuitively, if H is as defined in Lemma 31 then each point in the new representation is an indicator vector whose value is non-zero only on the element corresponds to the cluster it belongs to.

50 Information bottleneck

The information bottleneck method is a clustering technique introduced by Tishby, Pereira, and Bialek. To illustrate the method, consider the problem of clustering text documents where each document is represented as a bag-of-words, namely, each document is a vector $\mathbf{x} = \{0,1\}^n$, where n is the size of the dictionary and $x_i = 1$ iff the word corresponding to index i appears in the document. Given a set of m documents, we can interpret the bag-of-words representation of the m documents as a joint probability over a random variable X, indicating the identity of a document (thus taking values in $[m]$), and a random variable Y indicating the identity of a word in the dictionary (thus taking values in $[n]$).

With this interpretation, the information bottleneck refers to the identity of a clustering as another random variable, denoted C, that takes values in $[k]$ (where k will be set by the method as well). Once we formulated X, Y, C as random variable, we can use tools from information theory to express a clustering objective. In particular, the information bottleneck objective is

$$\min_{p(C|X)} I(X;C) - \beta I(C;Y) \ ,$$

where $I(\cdot;\cdot)$ is the mutual information between two variables[6], β is a parameter, and the minimization is over all possible probabilistic assignments of points to clusters. Intuitively, we would like to achieve two contradictory goals. On one hand, we would like the mutual information between the identity of the document and the identity of the cluster to be as small as possible. This reflects the fact that we would like a strong compression of the original data. On the other hand, we would like a high mutual information between the clustering variable and the identity of the words, which reflects the goal that the "relevant" information about the document (as reflected by the words that appear in the document) is retained. This generalizes the classical notion of minimal sufficient statistics[7] used in parametric statistics to arbitrary distributions.

Solving the optimization problem associated with the information bottleneck principle is hard in the general case. Some of the proposed methods rely on the EM principle (which we will discuss in the next lecture).

[6] That is, $I(X;C) = \sum_x \sum_c p(x,c) \log\left(\frac{p(x,c)}{p(x)p(c)}\right)$, where the sum is over all values X can take and all values C can take.

[7] A sufficient statistic is a function of the data which has the property of sufficiency with respect to a statistical model and its associated unknown parameter, meaning that "no other statistic which can be calculated from the same sample provides any additional information as to the value of the parameter". For example, if we assume that a variable is distributed normally with a unit variance and an unknown expectation, then the average function is a sufficient statistic.

(67577) Introduction to Machine Learning January 11, 2009

Lecture 17 – Generative Models

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*
Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

We started this course with a *distribution free* learning framework, namely, we did not impose any assumptions on the underlying distribution over the data. Furthermore, we followed a *discriminative* approach in which our goal is not to learn the underlying distribution but rather to learn an accurate predictor. In this lecture we describe a *generative* approach, in which it is assumed that the underlying distribution over the data has a specific parametric form and our goal is to estimate the parameters of the model. This task is called parametric density estimation.

The discriminative approach has the advantage of directly optimizing the quantity of interest (the prediction accuracy) instead of learning the underlying probability. This was phrased by Vladimir Vapnik's in his principle for solving problems using a restricted amount of information: *When solving a given problem, try to avoid a more general problem as an intermediate step*.

Of course, if we succeed to learn the underlying distribution accurately, we are considered to be 'experts' in the sense that we can design the Bayes optimal classifier. The problem is that it is usually more difficult to learn the underlying distribution than to learn an accurate predictor. However, in some situations, it is reasonable to adopt the generative learning approach. For example, sometimes it is easier to estimate the parameters of the model than to learn a discriminative predictor. Additionally, in some cases we do not have a specific task at hand but rather would like to model the data either for making predictions at a later time without having to re-train a predictor or for the sake of interpretability of the data.

We start with a popular statistical method for estimating the parameters of the data which is called the maximum likelihood principle. Next, we describe two generative assumptions which greatly simplify the learning process. We also describe the EM algorithm for calculating the maximum likelihood in the presence of latent variables. We conclude with a brief description of Bayesian reasoning.

51 Maximum Likelihood Estimator

Let us start with a simple example. A drug company developed a new drug to treat some deadly disease. We would like to estimate the probability of survival when using the drug and when not using the drug. To do so, the drug company sampled a training set of m people and gave them the drug. Let $S = (x_1, \ldots, x_m)$ denotes the training set, where for each i, $x_i = 1$ if the ith person survived and $x_i = 0$ otherwise. We can model the underlying distribution using a single parameter $\theta \in [0, 1]$ where $\mathbb{P}[X = 1] = \theta$.

We now would like to estimate the parameter θ based on the training set S. Since θ is the expected number of ones in S, a natural idea is to use the empirical number of ones in S as an estimator. That is,

$$\hat{\theta} \;=\; \frac{1}{m} \sum_{i=1}^{m} x_i \;. \tag{104}$$

Clearly, $\mathbb{E}_S[\hat{\theta}] = \theta$. That is, $\hat{\theta}$ is an *unbiased estimator* of θ. Furthermore, since $\hat{\theta}$ is the average of m i.i.d. binary random variables we can use Hoeffding's inequality to get that with probability of at least $1 - \delta$ over the choice of S we have that

$$|\hat{\theta} - \theta| \;\leq\; \sqrt{\frac{\log(2/\delta)}{2\,m}} \;. \tag{105}$$

Another interpretation of $\hat{\theta}$ is as the *Maximum Likelihood Estimator*, as we formally explain now. We

first write the probability of S:

$$\mathbb{P}[S = (x_1, \ldots, x_m)] \;=\; \prod_{i=1}^{m} \theta^{x_i} \, (1-\theta)^{1-x_i} \;=\; \theta^{\sum_i x_i} \, (1-\theta)^{\sum_i (1-x_i)} \; .$$

We defined the *likelihood* of S, given the parameter θ, as the log of the above expression:

$$L(S;\theta) \;=\; \log\left(\mathbb{P}[S = (x_1, \ldots, x_m)]\right) \;=\; \sum_i x_i \log(\theta) + \sum_i (1 - x_i) \log(1-\theta) \; .$$

The Maximum Likelihood (ML) estimator is to choose a parameter that maximizes the likelihood:

$$\hat{\theta} \in \operatorname*{argmax}_{\theta} L(S;\theta) \; . \tag{106}$$

Next, we show that Eq. (104) is a maximum likelihood estimator. To see this, we take the derivative of $L(S;\theta)$ with respect to θ and compare it to zero:

$$\frac{\sum_i x_i}{\theta} - \frac{\sum_i (1 - x_i)}{1-\theta} \;=\; 0 \; .$$

Solving the above for θ we obtain the estimator given in Eq. (104).

51.1 Continuous random variable and density of a distribution

Let X be a continuous random variable. Then, for most $x \in \mathbb{R}$ we have $\mathbb{P}[X = x] = 0$ and therefore the definition of likelihood as given before trivialized. To overcome this technical problem we define the Likelihood as log of the density of the probability of X at x. As an example, consider a Gaussian random variable. That is, the density function of X is parametrized by $\theta = (\mu, \sigma)$ and is defined as follows:

$$\mathcal{P}_\theta(x) \;=\; \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \; .$$

Given an i.i.d. training set $S = (x_1, \ldots, x_m)$ sampled according to a density distribution $\mathcal{P}_\theta$ we define the likelihood of S given θ as

$$L(S;\theta) \;=\; \log\left(\prod_{i=1}^{m} \mathcal{P}_\theta(x_i)\right) \;=\; \sum_{i=1}^{m} \log(\mathcal{P}_\theta(x_i)) \; .$$

As before, the maximum likelihood estimator is a maximizer of $L(S;\theta)$ with respect to θ.

Getting back to our Gaussian distribution, we can rewrite the likelihood as

$$L(S;\theta) = -\frac{1}{2\sigma^2} \sum_{i=1}^{m} (x_i - \mu)^2 - m \log(\sigma \sqrt{2\,\pi}) \; .$$

To find a parameter $\theta = (\mu, \sigma)$ that optimizes the above we take the derivative of the likelihood w.r.t. μ and w.r.t. σ and compare it to 0. We obtain the following two equations:

$$\begin{aligned}
\frac{d}{d\mu} L(S;\theta) \;&=\; \frac{1}{\sigma^2} \sum_{i=1}^{m} (x_i - \mu) \;=\; 0 \\
\frac{d}{d\sigma} L(S;\theta) \;&=\; \frac{1}{\sigma^3} \sum_{i=1}^{m} (x_i - \mu)^2 - \frac{m}{\sigma} \;=\; 0
\end{aligned}$$

Solving the above equations we obtain the ML estimate:

$$\hat{\mu} = \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{and} \quad \hat{\sigma} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - \hat{\mu})^2}$$

Note that the ML estimate is not always an unbiased estimator. For example, while $\hat{\mu}$ is unbiased, it is possible to show that the estimate $\hat{\sigma}$ of the variance is biased (Exercise 1).

Simplifying notation To simplify our notation, we use $\mathcal{P}[X = x]$ in this lecture to describe both the probability that $X = x$ (for discrete random variables) and the density of distribution at x (for continuous variables).

51.2 Maximum Likelihood and Empirical Risk Minimization

The ML estimator shares some similarity with the Empirical Risk Minimization (ERM) principle, which we studied extensively in this book. Recall that in the ERM principle we have a hypothesis class $\mathcal{H}$ and we use the training set for choosing a hypothesis $h \in \mathcal{H}$ that minimizes the empirical loss. We now show that the ML estimator is an ERM for a particular loss function.

Given a parameter θ and an observation x, we define the loss of θ on x as

$$\ell(\theta, x) = -\log(\mathcal{P}_\theta[x]) \,. \tag{107}$$

That is, $-\ell(\theta, x)$ is the log-likelihood of the observation x assuming the data is distributed according to $\mathcal{P}_\theta$. This loss function is often referred to as the log-loss. Based on this definition it is immediate that the ML principle is equivalent to minimizing the empirical risk with respect to the loss function given in Eq. (107). That is,

$$\hat{\theta} \in \operatorname*{argmin}_\theta \sum_{i=1}^{m} (-\log(\mathcal{P}_\theta[x])) \,.$$

Assuming that the data is distributed according to a distribution $\mathcal{P}$ (not necessarily of the parametric form we employ), the generalization error of a parameter θ becomes

$$\begin{aligned} \mathop{\mathbb{E}}_{x}[\ell(\theta, x)] &= -\sum_x \mathcal{P}[x] \log(\mathcal{P}_\theta[x]) \\ &= \underbrace{\sum_x \mathcal{P}[x] \log\left(\frac{\mathcal{P}[x]}{\mathcal{P}_\theta[x]}\right)}_{D_{\mathrm{RE}}[\mathcal{P}||\mathcal{P}_\theta]} + \underbrace{\sum_x \mathcal{P}[x] \log\left(\frac{1}{\mathcal{P}[x]}\right)}_{H(\mathcal{P})} \,, \end{aligned} \tag{108}$$

where D_{RE} is called the relative entropy divergence, and H is called the entropy function. The relative entropy is a divergence measure between two probabilities. For discrete variables, it is always non-negative and is equal to 0 only if the two distributions are the same.

The expression given in Eq. (108) underscores how our generative assumption effects our density estimation, even in the limit of infinite data. It shows that if the underlying distribution is indeed of a parametric form, then by choosing the correct parameter we can make the generalization error to be the entropy of the distribution. However, if the distribution is not of the assumed parametric form, even the best parameter leads to an inferior model and the suboptimality is measured by the relative entropy divergence.

51.3 Generalization Analysis

How good is the ML estimator when we learn from a finite data set ? To answer this question we need to define how we assess the quality of an approximated solution of the density estimation problem. Based on the previous subsection, a natural candidate is the expected log-loss as given in Eq. (108).

In some situations, it is easy to prove that the ML principle guarantees low generalization error as well. For example, consider the problem of estimating the mean of a Gaussian variable. We saw previously that the ML estimator is the average: $\hat{\mu} = 1/m \sum_i x_i$. Let $\mu^\star$ be the optimal parameter. Then,

$$\begin{aligned}
\mathop{\mathbb{E}}_{x}[\ell(\hat{\mu}, x) - \ell(\mu^\star, x)] &= \mathop{\mathbb{E}}_{x} \log\left(\frac{\mathcal{P}_{\mu^\star}[x]}{\mathcal{P}_{\hat{\mu}}[x]}\right) \\
&= \mathop{\mathbb{E}}_{x}\left(-\frac{1}{2}(x-\mu^\star)^2 + \frac{1}{2}(x-\hat{\mu})^2\right) \\
&= \frac{\hat{\mu}^2}{2} - \frac{(\mu^\star)^2}{2} + (\mu^\star - \hat{\mu}) \mathop{\mathbb{E}}_{x}[x] \\
&= \frac{1}{2}(\hat{\mu} - \mu^\star)^2 \,,
\end{aligned} \tag{109}$$

where the last equality is because $\mathbb{E}_x[x] = \mu^\star$. Next, we note that $\hat{\mu}$ is the average of m Gaussian variables and therefore it is also distributed normally with mean $\mu^\star$ and variance $\sigma^\star/m$. From this fact we can derive bound of the form: with probability of at least $1-\delta$ we have that $|\hat{\mu} - \mu^\star| \leq \epsilon$ where ϵ depends on $\sigma^\star/m$ and on δ.

In some situations, the ML estimator clearly overfits. For example, consider a Bernoulli random variable X and let $\mathcal{P}[X=1] = \theta^\star$. As we saw previously, using Hoeffding inequality we can easily derive a guarantee on $|\theta^\star - \hat{\theta}|$ that holds with high probability (see Eq. (105)). However, if our goal is to obtain a small value of the expected loss function as defined in Eq. (108) we might fail. For example, assume that $\theta^\star$ is non-zero but very small. Then, it is likely that no element in the sample will be 1 and therefore the ML rule will set $\hat{\theta} = 0$. But, the generalization error for this estimate is ∞. This simple example shows that we should be careful in applying the ML principle.

To overcome the overfitting, we can use the variety of tools we encountered previously in the book. A simple regularization technique is outlined in Exercise 2.

52 Naive Bayes

The Naive Bayes classifier is a classical demonstration of how generative assumptions and parameter estimations simplify the learning process. Consider the problem of predicting a label $y \in \{0,1\}$ based on a vector of features $\mathbf{x} = (x_1, \ldots, x_d)$, where for simplicity assume that each x_i is in $\{0,1\}$. Recall that the Bayes optimal classifier is

$$h_{\text{Bayes}}(\mathbf{x}) \;=\; \operatorname*{argmax}_{y \in \{0,1\}} \mathcal{P}[Y = y | X = \mathbf{x}] \,.$$

The number of parameters that is required to describe $\mathcal{P}[Y = y|X = \mathbf{x}]$ is order of 2^d. This implies that the number of examples we need grows exponentially with the number of features.

In the Naive Bayes approach we make the (rather naive) generative assumption that given the label, the features are independent of each other. That is,

$$\mathcal{P}[X = \mathbf{x}|Y = y] \;=\; \prod_{i=1}^{d} \mathcal{P}[X_i = x_i | Y = y] \,.$$

With this assumption and using Bayes rule, the Bayes optimal classifier can be further simplifies:

$$\begin{aligned}
h_{\text{Bayes}}(\mathbf{x}) \;&=\; \operatorname*{argmax}_{y \in \{0,1\}} \mathcal{P}[Y = y] \mathcal{P}[X = \mathbf{x}|Y = y] \\
&=\; \operatorname*{argmax}_{y \in \{0,1\}} \mathcal{P}[Y = y] \prod_{i=1}^{d} \mathcal{P}[X_i = x_i | Y = y] \,.
\end{aligned} \tag{110}$$

That is, now the number of parameters we need to estimate is only $2d+1$. That is, the generative assumption we made reduced significantly the number of parameters we need to learn.

When estimating the parameter using the maximum likelihood principle, the resulting classifier is called the *Naive Bayes* classifier.

53 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is another demonstration of how generative assumptions simplify the learning process. As in the Naive Bayes classifier we consider again the problem of predicting a label $y \in \{0,1\}$ based on a vector of features $\mathbf{x} = (x_1, \ldots, x_d)$. But, now the generative assumption is as follows. First, we assume that $\mathcal{P}[Y=1] = \mathcal{P}[Y=0] = 1/2$. Second, we assume that the conditional probability of $\mathbf{x}$ given y is a Gaussian distribution. Finally, the covariance matrix of the Gaussian distribution is the same for both values of y. Formally, let $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbb{R}^d$ and let Σ be a covariance matrix. Then, the density distribution is given by

$$\mathcal{P}[X=\mathbf{x}|Y=y] \;=\; \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_y)^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_y)\right) .$$

As we shown in the previous section, using Bayes rule we can write

$$h_{\text{Bayes}}(\mathbf{x}) \;=\; \operatorname*{argmax}_{y\in\{0,1\}} \mathcal{P}[Y=y]\mathcal{P}[X=\mathbf{x}|Y=y] .$$

This means that we will predict $h_{\text{Bayes}}(\mathbf{x}) = 1$ iff

$$\log\left(\frac{\mathcal{P}[Y=1]\mathcal{P}[X=\mathbf{x}|Y=1]}{\mathcal{P}[Y=0]\mathcal{P}[X=\mathbf{x}|Y=0]}\right) > 0 .$$

The ratio without the log is often called the *Likelihood ratio*.

In our case, the log-likelihood ratio becomes

$$\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_0)^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_0) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)$$

We can rewrite the above as $\langle \mathbf{w}, \mathbf{x}\rangle + b$ where

$$\mathbf{w} \;=\; \frac{1}{2}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)^T\Sigma^{-1} \quad \text{and} \quad b = \frac{1}{2}\left(\boldsymbol{\mu}_0^T\Sigma^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1^T\Sigma^{-1}\boldsymbol{\mu}_1\right) . \tag{111}$$

As a result of the above derivation we obtain that under the aforementioned generative assumption, the Bayes optimal classifier is a linear classifier. Additionally, one may train the classifier by estimating the parameter $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$ and Σ from the data, using for example the maximum likelihood estimator. With those estimators at hand, the values of $\mathbf{w}$ and b can be calculated as in Eq. (111). Note, however, that even if the generative assumption hold, this does not mean that the resulting predictor obtained by substituting maximum likelihood estimated values is optimal in any sense.

54 Latent variables and the EM algorithm

In generative models we assume that the data is generated by sampling from a specific parametric distribution over our space $\mathcal{X}$. Sometimes, it is convenient to express this distribution using latent random variables. A natural example is mixture of k Gaussian distributions. That is, $\mathcal{X} = \mathbb{R}^d$ and we assume that each $\mathbf{x}$ is generated as follows. First, we choose a random number in $\{1, \ldots, k\}$. Let Y be a random variable corresponding to this choice. Second, we choose $\mathbf{x}$ based on the value of Y according to a Gaussian distribution

$$\mathcal{P}[X=\mathbf{x}|Y=y] = \frac{1}{(2\pi)^{d/2}|\Sigma_y|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_y)^T\Sigma_y^{-1}(\mathbf{x}-\boldsymbol{\mu}_y)\right) . \tag{112}$$

Therefore, the density of X can be written as:

$$\mathcal{P}[X = \mathbf{x}] \;=\; \sum_{y=1}^{k} \mathcal{P}[Y = y]\mathcal{P}[X = \mathbf{x}|Y = y] \;.$$

Note that Y is a hidden variable that we do not observe in our data. Nevertheless, we introduce Y since it helps us to describe the parametric form of the probability of X.

Our goal in this section is to describe a procedure for estimating the parameters of a model with latent variables. We assume that we have m observations which are sampled i.i.d. according to the distribution over X and that the log-likelihood of X can be written in a parametric form using a random variable Y:

$$\log(\mathcal{P}[X = \mathbf{x}]) \;=\; \log\left(\sum_{y} \mathcal{P}_{\boldsymbol{\theta}}[Y = y]\mathcal{P}_{\boldsymbol{\theta}}[X = \mathbf{x}|Y = y]\right) \;.$$

For example, for mixture of k Gaussians we have that $\boldsymbol{\theta}$ is a triplet $(\mathbf{c}, \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}, \{\Sigma_1, \ldots, \Sigma_k\})$ where $\mathcal{P}_{\boldsymbol{\theta}}[Y = y] = c_y$ and $\mathcal{P}_{\boldsymbol{\theta}}[X = \mathbf{x}|Y = y]$ is as given in Eq. (112).

Given a data $S = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ we wish to find $\boldsymbol{\theta}$ that maximizes the log-likelihood. Denote by $X_{1:m}$ a sequence of random variables $X_1, \ldots, X_m$ and by $\mathbf{x}_{1:m}$ an instantiation of them. Similarly, denote by $Y_{1:m}$ and $y_{1:m}$ a sequence of hidden random variables and their instantiation. Our goal is to find a solution to the optimization problem

$$\max_{\boldsymbol{\theta}} \; \log\left(\sum_{y_{1:m}} \mathcal{P}_{\boldsymbol{\theta}}[Y_{1:m} = y_{1:m}, X_{1:m} = \mathbf{x}_{1:m}]\right) \;.$$

In many situations, it is computationally hard to solve the above optimization problem because the summation inside the log is over exponential number of assignments to $y_{1:m}$. The *Expectation-Maximization* (EM) algorithm, due to Dempster, Laird and Rubin, is a heuristic procedure that often works very well in practice. Furthermore, under mild conditions, the EM algorithm is guaranteed to converge to a local maximum of the optimization problem (see exercise 3).

The intuitive idea is that we have a "chicken and egg" problem. On one hand, if we knew the latent variables, the optimization problem would have become easy to solve and we can find the maximum likelihood estimator of $\boldsymbol{\theta}$. On the other hand, if we knew the parameters $\boldsymbol{\theta}$ we could have find a good assignment to the latent variables. The EM algorithm is an iterative method which alternates between finding $\boldsymbol{\theta}$ and finding a good distribution over the latent variables. Formally, EM finds a sequence of solutions $\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \ldots$ where at iteration t, we construct $\boldsymbol{\theta}^{t+1}$ from $\boldsymbol{\theta}^t$ by performing two steps.

- **E**xpectation step: Evaluate the distribution over the latent variables $y_{1:m}$ given the current parameter $\boldsymbol{\theta}^t$ and the observed data $\mathbf{x}_{1:m}$, that is $\mathcal{P}_{\boldsymbol{\theta}^t}[Y_{1:m} = y_{1:m}|X_{1:m} = \mathbf{x}_{1:m}]$. Using Bayes rule this distribution is:

  $$\frac{\mathcal{P}_{\boldsymbol{\theta}^t}[Y_{1:m} = y_{1:m}]\,\mathcal{P}_{\boldsymbol{\theta}^t}[X_{1:m} = \mathbf{x}_{1:m}|Y_{1:m} = y_{1:m}]}{\mathcal{P}_{\boldsymbol{\theta}^t}[X_{1:m} = \mathbf{x}_{1:m}]} \;.$$

  Because of the independence assumption the above simplifies to:

  $$\frac{\prod_{i=1}^{m} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y_i]\,\mathcal{P}_{\boldsymbol{\theta}^t}[X = \mathbf{x}_i|Y = y_i]}{\mathcal{P}_{\boldsymbol{\theta}^t}[X_{1:m} = \mathbf{x}_{1:m}]} \;.$$

- **M**aximization step: Suppose we had a training set in which the variables $y_1, \ldots, y_m$ were also observed. Then, the maximum likelihood principle set $\boldsymbol{\theta}$ to maximize

  $$\sum_{i=1}^{m} \log\left(\mathcal{P}_{\boldsymbol{\theta}}[Y = y_i, X = x_i]\right) \;.$$

Since we do not observe the variables $y_1, \ldots, y_m$, we take the expectation of the above expression w.r.t. the distribution over the latent variables calculated in the E step and set the new $\boldsymbol{\theta}$ to maximize the resulting expression. That is, in the M step we set $\boldsymbol{\theta}^{t+1}$ to be a maximizer of:

$$\sum_{y_{1:m}} \mathcal{P}_{\boldsymbol{\theta}^t}[Y_{1:m} = y_{1:m} | X_{1:m} = \mathbf{x}_{1:m}] \sum_{i=1}^{m} \log\left(\mathcal{P}_{\boldsymbol{\theta}}[Y = y_i, X = \mathbf{x}_i]\right) \ . \tag{113}$$

Furthermore, since we assume that the sequence of pairs (x_i, y_i) is i.i.d. the above simplifies to (see exercise 4)

$$\sum_{i=1}^{m} \sum_{y \in \mathcal{Y}} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y | X = \mathbf{x}_i] \log\left(\mathcal{P}_{\boldsymbol{\theta}}[Y = y, X = \mathbf{x}_i]\right) \ . \tag{114}$$

The initial value $\boldsymbol{\theta}^1$ is usually chosen at random and the procedure terminates after the improvement in the likelihood value stops to be significant.

Below we specify the EM algorithm for the important special case of mixture of Gaussians.

54.1 EM for Mixture of Gaussians (soft k-means)

Consider the case of mixture of k Gaussians in which $\boldsymbol{\theta}$ is a triplet $(\mathbf{c}, \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}, \{\Sigma_1, \ldots, \Sigma_k\})$ where $\mathcal{P}_{\boldsymbol{\theta}}[Y = y] = c_y$ and $\mathcal{P}_{\boldsymbol{\theta}}[X = \mathbf{x} | Y = y]$ is as given in Eq. (112). For simplicity, we assume that $\Sigma_1 = \Sigma_2 = \ldots = \Sigma_k = I$, where I is the identity matrix. Specifying the EM algorithm for this case we obtain the following:

- **E**xpectation step: For each $i \in [m]$ and $y \in [k]$ we have that

$$\begin{aligned} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y | X = \mathbf{x}_i] &= \frac{1}{Z_i} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y] \, \mathcal{P}_{\boldsymbol{\theta}^t}[X = \mathbf{x}_i | Y = y] \\ &= \frac{1}{Z_i} c_y^t \exp\left(-\frac{1}{2} \|\mathbf{x}_i - \boldsymbol{\mu}_y^t\|^2\right) \ , \end{aligned} \tag{115}$$

  where Z_i is a normalization factor which ensures that $\sum_y \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y | X = \mathbf{x}_i]$ sums to 1.

- **M**aximization step: We need to set $\boldsymbol{\theta}^{t+1}$ to be a maximizer of Eq. (114), which in our case amounts to maximizing the following expression w.r.t. $\mathbf{c}$ and $\boldsymbol{\mu}$:

$$\sum_{i=1}^{m} \sum_{y=1}^{k} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y | X = \mathbf{x}_i] \left(\log(c_y) - \frac{1}{2} \|\mathbf{x}_i - \boldsymbol{\mu}_y\|^2\right) \ . \tag{116}$$

  Comparing the derivative of Eq. (116) w.r.t. μ_y to zero and rearranging terms we obtain:

$$\boldsymbol{\mu}_y = \frac{\sum_{i=1}^{m} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y | X = \mathbf{x}_i] \, \mathbf{x}_i}{\sum_{i=1}^{m} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y | X = \mathbf{x}_i]} \ .$$

  That is, $\boldsymbol{\mu}_y$ is a weighted average of the $\mathbf{x}_i$ where the weights is according to the probabilities calculated in the E step. To find the optimal $\mathbf{c}$ we need to be more careful since we must ensure that $\mathbf{c}$ is a probability vector. In exercise 5 we show that the solution is:

$$c_y = \frac{\sum_{i=1}^{m} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y | X = \mathbf{x}_i]}{\sum_{y'=1}^{k} \sum_{i=1}^{m} \mathcal{P}_{\boldsymbol{\theta}^t}[Y = y' | X = \mathbf{x}_i]} \ . \tag{117}$$

It is interesting to compare the above algorithm with the k-means algorithm described in the previous lecture. In the k-means, we first assign each example to a cluster according to the distance $\|\mathbf{x}_i - \boldsymbol{\mu}_y\|$. Then, we update each center $\boldsymbol{\mu}_y$ according to the average of the examples assigned to this cluster. In the EM approach, we instead determine the probability that each example belongs to each cluster. Then, we update the centers based on a weighted sum over the entire examples. For this reason, the EM approach for k-means is sometimes called "soft k-means".

55 Bayesian Reasoning

The maximum likelihood estimator follows a frequentist approach. This means that we refer to the parameter θ as a fixed parameter and the only problem is that we do not know what its value is. A different approach to parameter estimation is called Bayesian reasoning. In the Bayesian approach, our uncertainty about θ is also modeled using probability theory. That is, we think on θ as a random variable as well and refer to the distribution $\mathcal{P}[\theta]$ as a *prior distribution*. As its name indicates, the prior distribution should be defined by the learner prior to observing the data.

As an example, lets consider again the drug company which developed a new drug. Based on past experience of the statisticians at the drug company, they believe that whenever a drug arrives to a level of a clinic experiment on people, it is likely to be effective. They model this prior belief by defining a density distribution on θ such that

$$\mathcal{P}[\theta] = \begin{cases} 0.8 & \text{if } \theta > 0.5 \\ 0.2 & \text{if } \theta \leq 0.5 \end{cases} \tag{118}$$

As before, given a specific value of θ, it is assumed that the conditional probability, $\mathcal{P}[X = x|\theta]$, is known. In the drug company example, X takes values in $\{0, 1\}$ and $\mathcal{P}[X = x|\theta] = \theta^x(1-\theta)^{1-x}$.

Once the prior distribution over θ and the conditional distribution over X given θ are defined, we again have a complete knowledge on the distribution over X. This is because we can write the probability over X as a marginal probability

$$\mathcal{P}[X = x] \;=\; \sum_{\theta} \mathcal{P}[\theta]\, \mathcal{P}[X = x, \theta] \;=\; \sum_{\theta} \mathcal{P}[\theta]\; (\mathcal{P}[\theta]\mathcal{P}[X = x|\theta]) \;,$$

where the last equality follows from Bayes rule. If θ is continuous we replace $\mathcal{P}[\theta]$ with the density function and the sum becomes an integral:

$$\mathcal{P}[X = x] \;=\; \int_{\theta} \mathcal{P}[\theta]\; (\mathcal{P}[\theta]\mathcal{P}[X = x|\theta])\, d\theta \;.$$

Seemingly, once we know $\mathcal{P}[X = x]$, a training set $S = (x_1, \ldots, x_m)$ tells us nothing as we are already experts that know the distribution over a new point X. However, the Bayesian view introduces dependency between S and X. This is because we now refer to θ as a random variable. A new point X and the previous points in S are independent *only* conditioned on θ. This is different than the frequentist philosophy in which θ is a parameter that we might don't know, but since it's just a parameter of the distribution, a new point X and previous points S are always independent.

In the Bayesian framework, since X and S are not independent any more, what we would like to calculate is the probability of X given S which by the chain rule can be written as follows:

$$\mathcal{P}[X = x|S] = \sum_{\theta} \mathcal{P}[X = x|\theta, S]\, \mathcal{P}[\theta|S] = \sum_{\theta} \mathcal{P}[X = x|\theta]\, \mathcal{P}[\theta|S] \;.$$

The second inequality follows from the assumption that X and S are independent when we condition on θ. Using Bayes rule and the assumption that points are independent conditioned on θ, we can write

$$\mathcal{P}[\theta|S] = \frac{\mathcal{P}[S|\theta]\, \mathcal{P}[\theta]}{\mathcal{P}[S]} \;=\; \frac{1}{\mathcal{P}[S]} \prod_{i=1}^{m} \mathcal{P}[X = x_i|\theta]\, \mathcal{P}[\theta] \;.$$

We therefore obtain the following expression for Bayesian prediction:

$$\mathcal{P}[X = x|S] = \frac{1}{\mathcal{P}[S]} \sum_{\theta} \mathcal{P}[X = x|\theta] \prod_{i=1}^{m} \mathcal{P}[X = x_i|\theta]\, \mathcal{P}[\theta] \;. \tag{119}$$

Getting back to our drug company example, we can rewrite $\mathcal{P}[X = x|S]$ as

$$\mathcal{P}[X = x|S] = \frac{1}{P[S]} \int \theta^{x+\sum_i x_i} (1-\theta)^{1-x+\sum_i (1-x_i)} \mathcal{P}[\theta] d\theta$$

It is interesting to note that when $\mathcal{P}[\theta]$ is uniform we obtain that

$$\mathcal{P}[X = x|S] \propto \int \theta^{x+\sum_i x_i} (1-\theta)^{1-x+\sum_i (1-x_i)} \, d\theta \; .$$

Solving the above integral (using integration by parts) we obtain

$$\mathcal{P}[X = 1|S] = \frac{\sum_i x_i + 2}{m+2} \; .$$

Recall that the prediction according to the Maximum Likelihood principle in this case is $\mathcal{P}[X = 1|\hat{\theta}] = \frac{\sum_i x_i}{m}$. The Bayesian prediction with uniform prior is rather similar to the Maximum Likelihood prediction, except it adds 'pseudo-examples' to the training set, thus biasing the prediction toward the uniform prior.

Maximum A-Posteriori In many situations, it is difficult to find a closed form solution to the integral given in Eq. (119). Several numerical methods can be used to approximate this integral. Another popular solution is to find a single θ which maximizes $\mathcal{P}[\theta|S]$. The value of θ which maximizes $\mathcal{P}[\theta|S]$ is called the *Maximum A-Posteriori* (MAP) estimator. Once this value is found, we can calculate the probability that $X = x$ given the MAP estimator and independently on S.

Generalization properties Predictors which are derived using a Bayesian approach can be analyzed using the PAC-Bayes formulation.

Exercises

1. Prove that the ML estimator of the variance of a Gaussian variable is biased.
2. Regularization for ML: Consider the following regularized loss minimization

$$\frac{1}{m} \sum_{i=1}^{m} \log(1/\mathcal{P}_\theta[x_i]) + \frac{1}{m} \left(\log(1/\theta) + \log(1/(1-\theta)) \right) \; .$$

   - Show that the above objective is equivalent to the usual empirical error had we add two pseudo-examples to the trainings set. Conclude that the regularized ML estimator will be

$$\hat{\theta} = \frac{1}{m+2} \left(1 + \sum_{i=1}^{m} x_i \right) \; .$$

   - Derive a high probability bound on $|\hat{\theta} - \theta^\star|$. Hint, rewrite the above as $|\hat{\theta} - \mathbb{E}[\hat{\theta}] + \mathbb{E}[\hat{\theta}] - \theta^\star|$ and then use the triangle inequality and Hoeffding inequality.
   - Use the above to bound the generalization error. Hint: Use the fact that now $\hat{\theta} \geq \frac{1}{m+2}$ to relate $|\hat{\theta} - \theta^\star|$ to the relative entropy.
3. Prove that EM converges to local maximum. Guidance:

- Let S be the observed variables and $\mathbf{y}$ be the hidden. Let the objective of EM be

$$L(\boldsymbol{\theta}) = \log\left(\sum_{\mathbf{y}} \mathcal{P}_{\boldsymbol{\theta}}[X, \mathbf{y}]\right) .$$

  Let q be an arbitrary vector in the simplex corresponds to the dimensionality of $\mathbf{y}$. Define the auxiliary function

$$Q(q, \boldsymbol{\theta}) \;=\; \sum_{\mathbf{y}} q(\mathbf{y}) \log \frac{\mathcal{P}_{\boldsymbol{\theta}}[S, \mathbf{y}]}{q(\mathbf{y})}$$

  Use Jensen's inequality to prove that

$$L(\boldsymbol{\theta}) = \log\left(\sum_{\mathbf{y}} q(\mathbf{y}) \frac{\mathcal{P}_{\boldsymbol{\theta}}[S, \mathbf{y}]}{q(\mathbf{y})}\right) \;\geq\; Q(q, \boldsymbol{\theta}) .$$

- Show that $L(\boldsymbol{\theta}^t) = Q(\mathcal{P}_{\boldsymbol{\theta}^t}[\cdot|S], \boldsymbol{\theta}^t)$.
- Conclude that

$$\operatorname*{argmax}_q Q(q, \boldsymbol{\theta}^t) = \mathcal{P}_{\boldsymbol{\theta}^t}[\mathbf{y}|S]$$

- Show that the EM algorithm is equivalent to an algorithm with the following iterations:

$$\text{E-step:} \quad q^{t+1} \;=\; \operatorname*{argmax}_q Q(q, \boldsymbol{\theta}^t)$$
$$\text{M-step:} \quad \boldsymbol{\theta}^{t+1} = \operatorname*{argmax}_{\boldsymbol{\theta}} Q(q^{t+1}, \boldsymbol{\theta})$$

- Conclude that the value of Q monotonically non-decreases
- Combine this with the fact that $Q(q^{t+1}, \boldsymbol{\theta}^t) = L(\boldsymbol{\theta}^t)$ to get that the value of $L(\boldsymbol{\theta}^t)$ is monotonically non-decreasing

4. Prove that Eq. (113) and Eq. (114) are equal.

5. - Consider a general optimization problem of the form:

$$\max_{\mathbf{c}} \sum_{y=1}^{k} \nu_y \log(c_y) \quad \text{s.t.} \quad c_y > 0, \sum_y c_y = 1 ,$$

  where $\boldsymbol{\nu} \in \mathbb{R}_+^k$ is a vector of non-negative weights. Verify that the M step of soft k-means involves solving such an optimization problem.
- Let $\mathbf{c}^\star = \frac{1}{\sum_y \nu_y} \boldsymbol{\nu}$. Show that $\mathbf{c}^\star$ is a probability vector.
- Show that the optimization problem is equivalent to the problem:

$$\min_{\mathbf{c}} D_{\mathrm{RE}}(\mathbf{c}^\star || \mathbf{c}) \quad \text{s.t.} \quad c_y > 0, \sum_y c_y = 1 .$$

- Using properties of the relative entropy, conclude that $\mathbf{c}^\star$ is the solution to the optimization problem.

(67577) Introduction to Machine Learning January 12, 2010

Lecture 18 – Boosting

Lecturer: Ohad Shamir *Scribe: Ohad Shamir*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In the supervised learning framework we have considered so far, it was implicitly assumed that we can design a hypothesis class and representation for the data, such that a good classifier can be found based on a small training sample. However, this task can sometimes be very difficult in practice. Quite often, we can only come up with marginally useful and reliable classifiers for our learning problem: Namely, classifiers than are only slightly better than random. Given a learning algorithm which produces such 'weak' classifiers, can we use it to obtain 'strong' classifiers, which achieve low error with high probability?

The question of whether weak learning algorithms can be 'boosted' into strong learning algorithms was first raised in the late 80's, and solved in 1990 by Robert Schapire, then a graduate student at MIT. However, the proposed mechanism was not very practical. In 1995, Robert Schapire and Yoav Freund proposed the Adaboost algorithm, which was the first truly practical implementation of boosting. This simple and elegant algorithm (which we present and analyze later on) became hugely popular, and Freund & Schapire's work has been recognized by numerous awards.

In fact, boosting is a great example for the practical impact of learning theory. While boosting originated as a purely theoretical problem, it has lead to popular and widely used algorithms. For example, a face recognition algorithm based on boosting (due to Viola & Jones) is widely used in digital cameras sold today.

56 The Problem of Boosting

For simplicity, we will focus here solely on binary classification, where our goal is to predict binary labels (either -1 or 1). Also, we will assume the *realizable assumption*, which was used when we discussed PAC learnability: namely, we constrain ourselves to distributions $\mathcal{D}$ over the examples $\mathcal{X} \times \{-1, 1\}$, for which there exist $h^\star$ in the hypothesis class $\mathcal{H}$, such that $\mathrm{err}_{\mathcal{D}}(h^\star) = 0$.

Under these assumptions, we define the following variant of PAC-learnability, which we will denote as *strong learnability*:

Definition 10 (Strong Learnability) *A learning algorithm A is a* strong learner*, if for any distribution $\mathcal{D}$ which satisfies the realizable assumption, and $\forall \delta, \epsilon > 0$, the following holds: if we run A on a sample of $m = poly\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$ i.i.d. examples from $\mathcal{D}$, it returns a hypothesis h such that $\mathrm{err}_{\mathcal{D}}(h) \leq \epsilon$ with probability at least $1 - \delta$.*

Compared to the definition of PAC-learnability, the only difference is that we do not require h to be a member of $\mathcal{H}$: it can be any function from the domain $\mathcal{X}$ to labels $\{-1, 1\}$. This relaxation (sometimes denoted as *improper* learning) is needed because boosting algorithms usually do not return a hypothesis in $\mathcal{H}$.

We will now define *weak learnability*, which captures the kind of weak learning we discussed informally above.

Definition 11 (Weak Learner) *A learning algorithm A is a* weak learner*, if there are fixed parameters $\gamma > 0$, $m_0 > 0$, $\delta_0 < 1$, such that for any distribution $\mathcal{D}$ which satisfies the realizable assumption, the following holds: if we run A on a sample of m_0 i.i.d. examples from D, it returns a hypothesis $h \in H$ such that $\mathrm{err}_D(h) \leq \frac{1}{2} - \gamma$ with probability at least $1 - \delta_0$.*

Note that in weak learning, we do require the returned hypothesis to be a member of $\mathcal{H}$. Other than that, comparing the two definitions, we see that they differ in two aspects: first, strong learnability provides guarantees with arbitrarily high probability $1 - \delta$ over the training set, while weak learnability provides

guarantees only with some positive probability $1-\delta_0$. Secondly, strong learnability implies the ability to find an arbitrarily good classifier (with error rate at most ϵ for an arbitrarily small $\epsilon > 0$). On the other, in weak learnability we are only guaranteed to get a hypothesis whose error rate is slightly better than what a random labeling would give us (e.g. $1/2$).

Clearly, any strong learner is also a weak learner. The main theoretical premise of boosting is that under certain assumptions, given a weak learner, one can use it to form a strong learner. This is achieved using boosting algorithms.

To show that weak learnability implies strong learnability, we need to show two things: first, how to *boost the confidence*, i.e. convert an algorithm with a fixed confidence parameter δ_0 into an algorithm whose confidence parameter is an arbitrarily small $\delta > 0$. Second, we need to show how to *boost the accuracy*, i.e. convert an algorithm with a fixed accuracy parameter $1/2 - \gamma$ for some $\gamma > 0$, into an algorithm whose returned hypothesis has arbitrarily small error $\epsilon > 0$. In this lecture, we will first show how to boost the confidence. Then, we will show how to boost the accuracy, in the sense of reducing the *training error* on the training set to an arbitrarily small value. Showing how the *generalization error* can be reduced is a somewhat trickier issue, which is out of the scope of this course.

57 Boosting the Confidence

In this section, we will show the following result: suppose we are given a learning algorithm A, which after running on m training examples, returns a hypothesis with error rate ϵ with some fixed probability $1-\delta_0$ (where $1-\delta_0$ might be close to 0). Then it is possible to build another learning algorithm A' which achieves error rate $\epsilon + \lambda$ (for $\lambda > 0$ as small as we wish) with a probability $1-\delta$, where $\delta > 0$ is as small as we wish.

To achieve this, we construct the algorithm A' as follows (where n and k in the pseudo-code are parameters to be determined later on):

1. Sample k samples of size m.
2. Apply A on each of the k samples, resulting in hypotheses $h_1, h_2, \ldots, h_k$.
3. Sample another sample S of size n, and return $\arg\min_{i \in \{1,\ldots,k\}} \mathrm{err}_S(h_i)$.

The guarantee on A' is formalized in the following theorem:

Theorem 25 *For any desired $\delta > 0$, if we let $k = \frac{\log(\delta)}{\log(2\delta_0)}$ and $n = \frac{2\log(2k/\delta)}{\lambda^2}$, then with probability at least $1-\delta$ over the samples, A' returns a hypothesis h such that $\mathrm{err}_{\mathcal{D}}(h) \leq \epsilon + \lambda$.*

Proof For each $i \in \{1, \ldots, k\}$, it holds that $\Pr(\mathrm{err}_{\mathcal{D}}(h_i) > \epsilon) \leq \delta_0$. Since each h_i was selected based on an independent sample,

$$\Pr(\forall i \;\; \mathrm{err}_{\mathcal{D}}(h_i) > \epsilon) = \prod_{i=1}^{k} \Pr(\mathrm{err}_{\mathcal{D}}(h_i) > \epsilon) \leq \delta_0^k.$$

So with probability at least $1-\delta_0^k$, there is a hypothesis $h^\star \in \{h_1, \ldots, h_k\}$ such that

$$\mathrm{err}_{\mathcal{D}}(h^\star) \leq \epsilon. \tag{120}$$

We can think of $h_1, \ldots, h_k$ as a finite hypothesis class of size k. The last step of A' consists of performing ERM with respect to this hypothesis class. From results we have obtained earlier in the course about learnability of finite hypothesis classes, we know that with probability at least $1-\delta_0^k$, if we sample $n \geq 2\log(2k/\delta_0^k)/\lambda^2$ examples for S (where $\lambda > 0$ is some parameter) and apply the ERM, then with probability at least $1-\delta_0^k$, the returned hypothesis h satisfies

$$\mathrm{err}_{\mathcal{D}}(h) \leq \mathrm{err}_{\mathcal{D}}(h^\star) + \lambda. \tag{121}$$

Combining Eq. (120) and Eq. (121) with a union bound, it holds with probability at least $1 - 2\delta_0^k$ (over the sampling performed by algorithm A') that A' returns a hypothesis h such that

$$\mathrm{err}_{\mathcal{D}}(h) \leq \epsilon + \lambda.$$

Finally, for any desired confidence parameter δ, we just need to set k large enough so that $2\delta_0^k = \delta$. This happens if we let $k \geq \frac{\log(\delta)}{\log(2\delta_0)}$. ■

58 Boosting the Accuracy: the AdaBoost Algorithm

After dealing with boosting the confidence, we now turn to the question of how to boost the accuracy. We will see how this can be achieved using the AdaBoost algorithm (short for 'adaptive boosting') , which we present and analyze in this section.

The AdaBoost algorithm receives as input a training set of examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ where for all $i \in [m]$, $\mathbf{x}_i$ is taken from an instance domain $\mathcal{X}$, and y_i is a binary label, $y_i \in \{+1, -1\}$. The boosting process proceeds in a sequence of consecutive rounds. At round t, the booster first defines a distribution, denoted $\mathbf{d}_t$, over the set of examples. Then, the booster passes the training set along with the distribution $\mathbf{d}_t$ to the weak learner. The weak learner is assumed to return a hypothesis $h_t : \mathcal{X} \to \{+1, -1\}$ whose average error is slightly smaller than $\frac{1}{2}$, for any distribution[8] . In particular, it should also work for some distribution over the finite set of examples in the training set. Therefore, there exists a constant $\gamma > 0$ such that,

$$\epsilon_t \stackrel{\text{def}}{=} \sum_{i=1}^{m} d_{t,i} \frac{1 - y_i h_t(\mathbf{x}_i)}{2} \leq \frac{1}{2} - \gamma \ . \tag{122}$$

The goal of the boosting algorithm is to invoke the weak learner several times with different distributions, and to combine the hypotheses returned by the weak learner into a final, so-called strong hypothesis whose error is small. The final hypothesis is a weighted linear combination of the T hypotheses returned by the weak learner. The pseudo-code of AdaBoost is presented below.

Algorithm 10 The AdaBoost Algorithm

Input: Training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, weak learner A, desired number of rounds T.
Initialize $\mathbf{d}_1 = (\frac{1}{m}, \ldots, \frac{1}{m})$.
For $t = 1, \ldots, T$:
  Invoke A with the distribution $\mathbf{d}_i$ on the training set, receive hypothesis h_t.
  Compute ϵ_t as defined in Eq. (122).
  Let $\omega_t := \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
  Let $d_{t+1,i} := d_{t,i} \frac{\exp(-\omega_t y_i h_t(\mathbf{x}_i))}{Z_t}$ for all $i = 1, \ldots, m$.
    // Here, $Z_t := \sum_{i=1}^{k} d_{t,i} \exp(-\omega_t y_i h_t(\mathbf{x}_i))$ is a normalization factor.
Output the hypothesis $h_f(\mathbf{x}) := \mathrm{sign}\left(\sum_{t=1}^{T} \omega_t h_t(\mathbf{x})\right)$.

Theorem 26 *Under the assumption given in Eq. (122), the error of the final strong hypothesis is at most*

$$\exp(-2\,\gamma^2\,T) \ .$$

The proof is based on the following lemma.

[8] Strictly speaking, the weak learner only returns such a hypothesis with some positive probability. However, since we already know how to boost the confidence to an arbitrarily high level, we will assume for simplicity that the learner returns such a hypothesis with probability 1. It is not hard to modify the analysis to deal with the case where this probability is strictly less than 1.

Lemma 32 *Consider an arbitrary boosting algorithm that satisfies:*

1. *The distribution is set according to:* $d_{t,i} \propto e^{-y_i \sum_{j<t} \omega_j h_j(\mathbf{x}_i)}$.
2. $\omega_t \in \{\omega : e^{-\omega}(1-\epsilon_t) + e^{\omega}\epsilon_t \leq e^{-2\gamma^2}\}$.

Then:

$$\frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{T} \omega_t \, h_t(\mathbf{x}_i)} \;\leq\; e^{-2\,\gamma^2\,T} \;,$$

Proof Consider the ratio

$$R_t \;=\; \frac{\frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{j=1}^{t} \omega_j \, h_j(\mathbf{x}_i)}}{\frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{j=1}^{t-1} \omega_j \, h_j(\mathbf{x}_i)}} \;.$$

We have

$$\prod_{t=1}^{T} R_t \;=\; \frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{T} \omega_t \, h_t(\mathbf{x}_i)} \;.$$

On the other hand,

$$R_t \;=\; \sum_{i=1}^{m} d_{t,i}\, e^{-y_i \omega_t h_t(\mathbf{x}_i)} \;=\; e^{-\omega_t}(1-\epsilon_t) + e^{\omega_t}\epsilon_t \;\leq\; e^{-2\gamma^2} \;,$$

where the first equality follows from the first assumption, the second equality follows from the definition of ϵ_t, and the last inequality is the second assumption. Thus $\prod_t R_t \leq e^{-2\gamma^2 T}$ and our proof is concluded. ■

Based on the above lemma we now turn to the proof of Theorem 26.

Proof [of Theorem 26] Plugging the definition of $\omega_t = \frac{1}{2}\log((1-\epsilon_t)/\epsilon_t)$ we obtain

$$e^{-\omega_t}(1-\epsilon_t) + e_t^{\omega}\epsilon_t \;=\; 2\sqrt{\epsilon_t(1-\epsilon_t)} \;.$$

In fact, it is easy to verify that in AdaBoost the value of ω_t is the maximizer of the expression $e^{-\omega}(1-\epsilon_t) + e^{\omega}\epsilon_t$. The expression $\epsilon(1-\epsilon)$ is monotonically increasing in $[0, 1/2]$. Combining this with the weak learnability assumption we obtain

$$e^{-\omega_t}(1-\epsilon_t) + e_t^{\omega}\epsilon_t \;\leq\; 2\sqrt{(1/2-\gamma)(1/2+\gamma)} = \sqrt{1-4\gamma^2} \leq e^{-2\gamma^2} \;,$$

Therefore, we can apply Lemma 32, and get that

$$\frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{T} \omega_t \, h_t(\mathbf{x}_i)} \;\leq\; e^{-2\,\gamma^2\,T} \;.$$

To finish the proof, we note that $\mathbb{1}_{[a<0]} \leq e^{-a}$ for all a, so we can lower bound the left hand side of the inequality above by

$$\frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{[-y_i \sum_{t=1}^{T} \omega_t \, h_t(\mathbf{x}_i) < 0]} \;=\; \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{[y_i \neq \mathrm{sign}\left(\sum_{t=1}^{T} \omega_t h_t(\mathbf{x})\right)]} \;,$$

which is exactly the training error of the hypothesis returned by the Adaboost algorithm. ■

(67577) Introduction to Machine Learning January 12, 2009

Lecture 19 – Online Learning

Lecturer: Shai Shalev-Shwartz *Scribe: Shai Shalev-Shwartz*

Based on a book by Shai Ben-David and Shai Shalev-Shwartz (in preparation)

In this lecture we describe a different model of learning which is called *online* learning. Online learning takes place in a sequence of consecutive rounds. To demonstrate the online learning model, consider again the papaya tasting problem. On each online round, the learner first receives an instance (the learner buys a papaya and knows its shape and color, which form the instance). Then, the learner is required to predict a label (is the papaya tasty?). At the end of the round, the learner gets the correct label (he tastes the papaya and then knows if it's tasty or not). Finally, the learner uses this information to improve his future predictions.

Previously, we used the batch learning model in which we first use a batch of training examples to learn a hypothesis and only when learning is completed the learned hypothesis is tested. In our papayas learning problem, we should first buy bunch of papayas and taste them all. Then, we use all of this information to learn a prediction rule that determines the taste of new papayas. In contrast, in online learning there is no separation between a training phase and a test phase. The same sequence of examples is used both for training and testing and the distinguish between train and test is through time. In our papaya problem, each time we buy a papaya, it is first considered a *test* example since we should predict if it's going to taste good. But, after we take a byte from the papaya, we know the true label, and the same papaya becomes a *training* example that can help us improve our prediction mechanism for future papayas.

The goal of the online learner is simply to make few prediction mistakes. By now, the reader should know that there are no free lunches – we must have some prior knowledge on the problem in order to be able to make accurate predictions. As in previous lectures, we encode our prior knowledge on the problem using some representation of the instances (e.g. shape and color) and by assuming that there is a class of hypotheses, $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y}\}$, and on each online round the learner uses a hypothesis from $\mathcal{H}$ to make his prediction.

To simplify our presentation, we start the lecture by describing online learning algorithms for the case of a finite hypothesis class.

59 Online Learning and Mistake Bounds for Finite Hypothesis Classes

Throughout this section we assume that $\mathcal{H}$ is a finite hypothesis class. On round t, the learner first receives an instance, denoted $\mathbf{x}_t$, and is required to predict a label. After making his prediction, the learner receives the true label, y_t.

The most natural learning rule is to use a consistent hypothesis at each online round. If there are several consistent hypotheses, it makes sense to choose one uniformly at random, as there is no reason to prefer one consistent hypothesis over another. This leads to the following algorithm.

Algorithm 11 `RandConsistent`

INPUT: A finite hypothesis class $\mathcal{H}$

INITIALIZE: $V_1 = \mathcal{H}$

FOR $t = 1, 2, \ldots$

Receive $\mathbf{x}_t$

Choose some h from V_t uniformly at random

Predict $\hat{y}_t = h(\mathbf{x}_t)$

Receive true answer y_t

Update $V_{t+1} = \{h \in V_t : h(\mathbf{x}_t) = y_t\}$

The `RandConsistent` algorithm maintains a set, V_t, of all the hypotheses which are consistent with $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{t-1}, y_{t-1})$. It then chooses a hypothesis uniformly at random from V_t and predicts according to this hypothesis.

Recall that the goal of the learner is to make few mistakes. The following theorem upper bounds the expected number of mistakes `RandConsistent` makes on a sequence of examples. To motivate the bound, consider a round t and let α_t be the fraction of hypotheses in V_t which are going to be correct on the example $(\mathbf{x}_t, y_t)$. Now, if α_t is close to 1, it means we are likely to make a correct prediction. On the other hand, if α_t is close to 0, we are likely to make a prediction error. But, on the next round, after updating the set of consistent hypotheses, we will have $|V_{t+1}| = \alpha_t |V_t|$, and since we now assume that α_t is small, we will have a much smaller set of consistent hypotheses in the next round. To summarize, if we are likely to err on the current example then we are going to learn a lot from this example as well, and therefore be more accurate in later rounds.

Theorem 27 *Let $\mathcal{H}$ be a finite hypothesis class, let $h^\star$ be some hypothesis in $\mathcal{H}$ and let $(\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_T, h^\star(\mathbf{x}_T))$ be an arbitrary sequence of examples. Then, the expected number of mistakes the `RandConsistent` algorithm makes on this sequence is at most $\ln(|\mathcal{H}|)$, where expectation is with respect to the algorithm's own randomization.*

Proof For each round t, let $\alpha_t = |V_{t+1}|/|V_t|$. Therefore, after T rounds we have

$$1 \leq |V_{T+1}| = |\mathcal{H}| \prod_{t=1}^{T} \alpha_t \,.$$

Using the inequality $b \leq e^{-(1-b)}$, which holds for all b, we get that

$$\begin{aligned} 1 \leq |\mathcal{H}| \prod_{t=1}^{T} e^{-(1-\alpha_t)} &= |\mathcal{H}|\, e^{-\sum_{t=1}^{T}(1-\alpha_t)} \\ \Rightarrow \sum_{t=1}^{T}(1-\alpha_t) &\leq \ln(|\mathcal{H}|) \,. \end{aligned} \tag{123}$$

Finally, since we predict $\hat{y}_t$ by choosing $h \in V_t$ uniformly at random, we have that the probability to make a mistake on round t is

$$\mathbb{P}[\hat{y}_t \neq y_t] = \frac{|\{h \in V_t : h(\mathbf{x}_t) \neq y_t\}|}{|V_t|} = \frac{|V_t| - |V_{t+1}|}{|V_t|} = (1 - \alpha_t) \,.$$

Therefore, the expected number of mistakes is

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}_{[\hat{y}_t \neq y_t]}] = \sum_{t=1}^{T} \mathbb{P}[\hat{y}_t \neq y_t] = \sum_{t=1}^{T}(1 - \alpha_t) \,.$$

Combining the above with Eq. (123) we conclude our proof. ■

It is interesting to compare the guarantee in Theorem 27 to guarantees on the generalization error in the PAC model (see Corollary 1). In the PAC model, we refer to the T examples in the sequence as a training set. Then, Corollary 1 implies that with probability of at least $1-\delta$, our average error on new examples is guaranteed to be at most $\ln(|\mathcal{H}|/\delta)/T$. In contrast, Theorem 27 tells us a much stronger guarantee. We do not need to first train the model on T examples, in order to have error rate of $\ln(|\mathcal{H}|)/T$. We can have this same error rate immediately on the first T examples we observe. In our papayas example, we don't need to first buy T papayas, taste them all, and only then to be able to classify new papayas. We can start making predictions from the first day, each day trying to buy a tasty papaya, and we know that our performance after T days will be the same as our performance had we first trained our model using T papayas !

Another important difference between the online model and the batch model is that in the latter we assume that instances are sampled i.i.d. from some underlying distribution, but in the former there is no such an assumption. In particular, Theorem 27 holds for any sequence of instances. Removing the i.i.d. assumption is a big advantage. Again, in the papayas problem, we are allowed to choose a new papaya every day, which clearly violates the i.i.d. assumption. On the flip side, we only have a guarantee on the total number of mistakes but we have no guarantee that after observing T examples we will identify the 'true' hypothesis. Indeed, if we observe the same example on all the online rounds, we will make few mistakes but we will remain with a large set V_t of hypotheses, all of them are potentially the true hypothesis.

Note that the `RandConsistent` algorithm is a specific variant of the general `Consistent` learning paradigm (i.e., ERM) and that the bound in Theorem 27 relies on the fact that we use this specific variant. This stands in contrast to the results we had before for the PAC model in which it doesn't matter how we break ties, and any consistent hypothesis is guaranteed to perform well. In some situations, it is computationally harder to sample a consistent hypothesis from V_t while it is less demanding to merely find one consistent hypothesis. Moreover, if $\mathcal{H}$ is infinite, it is not well defined how to choose a consistent hypothesis uniformly at random. On the other hand, as mentioned before, the results we obtained for the PAC model assume that the data is i.i.d. while the bound for `RandConsistent` holds for any sequence of instances. If the data is indeed generated i.i.d. then it is possible to obtain a bound for the general `Consistent` paradigm.

Theorem 27 bounds the *expected* number of mistakes. Using martingale measure concentration techniques, one can obtain a bound which holds with extremely high probability. A simpler way is to explicitly derandomize the algorithm. Note that `RandConsistent` predicts 1 with probability greater than $1/2$ if the majority of hypotheses in V_t predicts 1. A simple derandomization is therefore to make a deterministic prediction according to a majority vote of the hypotheses in V_t. The resulting algorithm is called `Halving`.

Algorithm 12 `Halving`

INPUT: A finite hypothesis class $\mathcal{H}$

INITIALIZE: $V_1 = \mathcal{H}$

FOR $t = 1, 2, \ldots$

Receive $\mathbf{x}_t$

Predict $\hat{y}_t = \operatorname{argmax}_{r \in \{\pm 1\}} |\{h \in V_t : h(\mathbf{x}_t) = r\}|$

(In case of a tie predict $\hat{y}_t = 1$)

Receive true answer y_t

Update $V_{t+1} = \{h \in V_t : h(\mathbf{x}_t) = y_t\}$

Theorem 28 *Let $\mathcal{H}$ be a finite hypothesis class, let $h^\star$ be some hypothesis in $\mathcal{H}$ and let $(\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_T, h^\star(\mathbf{x}_T))$ be an arbitrary sequence of examples. Then, the number of mistakes the `Halving` algorithm makes on this sequence is at most $\log_2(|\mathcal{H}|)$.*

Proof We simply note that whenever the algorithm errs we have $|V_{t+1}| \leq |V_t|/2$. (Hence the name Halving.) Therefore, if M is the total number of mistakes, we have

$$1 \leq |V_{T+1}| \leq |\mathcal{H}|\, 2^{-M} \ .$$

Rearranging the above inequality we conclude our proof. ■

A guarantee of the type given in Theorem 28 is called a *Mistake Bound*. Theorem 28 states that `Halving` enjoys a mistake bound of $\log_2(|\mathcal{H}|)$. In the next section, we relax the assumption that all the labels are generated by a hypothesis $h^\star \in \mathcal{H}$.

60 Weighted Majority and Regret Bounds

In the previous section we presented the `Halving` algorithm and analyze its performance by providing a mistake bound. A crucial assumption we relied on is that the data is realizable, namely, the labels in the sequence are generated by some hypothesis $h^\star \in \mathcal{H}$. This is a rather strong assumption on our data and prior knowledge. In this section we relax this assumption.

Recall that the mistake bounds we derived in the previous section do not require the data to be sampled i.i.d. from some distribution. We allow the sequence to be deterministic, stochastic, or even adversarially adaptive to our own behavior (for example, this is the case in spam email filtering). Clearly, learning is impossible if there is no correlation between past and present examples.

In the realizable case, future and past examples are tied together by the common hypothesis, $h^\star \in \mathcal{H}$, that generates all labels. In the non-realizable case, we analyze the performance of an online learner using the notion of *regret*. The learner's regret is the difference between his number of prediction mistakes and the number of mistakes the optimal fixed hypothesis in $\mathcal{H}$ makes on the same sequence of examples. This is termed 'regret' since it measures how 'sorry' the learner is, in retrospect, not to have followed the predictions of the optimal hypothesis. We again use the notation $h^\star$ to denote the hypothesis in $\mathcal{H}$ that makes the least number of mistakes on the sequence of examples. But, now, the labels are not generated by $h^\star$, meaning that for some examples we might have $h^\star(\mathbf{x}_t) \neq y_t$.

As mentioned before, learning is impossible if there is no correlation between past and future examples. However, even in this case, the algorithm can have low regret. In other words, having low regret does not necessarily mean that we will make few mistakes. It only means that the algorithm will be competitive with an oracle, that knows in advance what the data is going to be, and chooses the optimal $h^\star$. If our prior knowledge is adequate, then $\mathcal{H}$ contains a hypothesis that (more or less) explains the data, and then a low regret algorithm is guaranteed to have few mistakes.

We now present an online learning algorithm for the non-realizable case, also called the agnostic case. As in the previous section, we assume that $\mathcal{H}$ is a finite hypothesis class and denote $\mathcal{H} = \{h_1, \ldots, h_d\}$. Recall that the `RandConsistent` algorithm maintains the set V_t of all hypotheses which are consistent with the examples observed so far. Then, it samples a hypothesis from V_t uniformly at random. We can represent the set V_t as a vector $\mathbf{w}_t \in \mathbb{R}^d$, where $w_{t,i} = 1$ if $h_i \in V_t$ and otherwise $w_{t,i} = 0$. Then, the `RandConsistent` algorithm chooses h_i with probability $w_{t,i}/(\sum_j w_{t,j})$, and the vector $\mathbf{w}$ is updated at the end of the round by zeroing all elements corresponding to hypotheses that err on the current example.

If the data is non-realizable, the weight of $h^\star$ will become zero once we encounter an example on which $h^\star$ errs. From this point on, $h^\star$ will not affect the prediction of the `RandConsistent` algorithm. To overcome this problem, one can be less aggressive and instead of zeroing weights of erroneous hypotheses, one can just diminish their weight by scaling down their weight by some $\beta \in (0, 1)$. The resulting algorithm is called `Weighted-Majority`.

Algorithm 13 `Weighted-Majority`

INPUT: Finite hypothesis class $\mathcal{H} = \{h_1, \ldots, h_d\}$; Number of rounds T

INITIALIZE: $\beta = e^{-\sqrt{2\,\ln(d)/T}}$; $\mathbf{w}_1 = (1, \ldots, 1) \in \mathbb{R}^d$

FOR $t = 1, 2, \ldots, T$

Let $Z_t = \sum_{j=1}^{d} w_{t,j}$

Sample a hypothesis $h \in \mathcal{H}$ at random according to $\left(\frac{w_{t,1}}{Z_t}, \ldots, \frac{w_{t,d}}{Z_t}\right)$

Predict $\hat{y}_t = h(\mathbf{x}_t)$

Receive true answer y_t

Update: $\forall j,\;\; w_{t+1,j} = \begin{cases} w_{t,j}\,\beta & \text{if } h_j(\mathbf{x}_t) \neq y_t \\ w_{t,j} & \text{else} \end{cases}$

The following theorem provides an expected regret bound for the algorithm.

Theorem 29 *Let $\mathcal{H}$ be a finite hypothesis class, and let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ be an arbitrary sequence of examples. If we run* `Weighted-Majority` *on this sequence we have the following expected regret bound*

$$\mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}_{[\hat{y}_t \neq y_t]}\right] - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \mathbb{1}_{[h(\mathbf{x}_t) \neq y_t]} \;\leq\; \sqrt{0.5\,\ln(|\mathcal{H}|)\,T}\,,$$

where expectation is with respect to the algorithm own randomization.

Proof Let $\eta = \sqrt{2\,\ln(d)/T}$ and note that $w_{t+1,i} = w_{t,i} e^{-\eta\,\mathbb{1}_{[h_i(\mathbf{x}_t) \neq y_t]}}$. Therefore,

$$\ln \frac{Z_{t+1}}{Z_t} = \ln \sum_i \frac{w_{t,i}}{Z_t}\, e^{-\eta \mathbb{1}_{[h_i(\mathbf{x}_t) \neq y_t]}}\,.$$

Hoeffding inequality tells us that if X is a random variable over $[0, 1]$ then

$$\ln \mathbb{E}[e^{-\eta X}] \leq -\eta\,\mathbb{E}[X] + \frac{\eta^2}{8}\;\;.$$

Since $\mathbf{w}_t / Z_t$ is a probability vector and $\mathbb{1}_{[h_i(\mathbf{x}_t) \neq y_t]} \in [0, 1]$, we can apply Hoeffding's inequality to obtain:

$$\ln \frac{Z_{t+1}}{Z_t} \;\leq\; -\eta \sum_i \frac{w_{t,i}}{Z_t} \mathbb{1}_{[h_i(\mathbf{x}_t) \neq y_t]} + \frac{\eta^2}{8} \;=\; -\eta\,\mathbb{E}[\mathbb{1}_{[\hat{y}_t \neq y_t]}] + \frac{\eta^2}{8}\,.$$

Summing the above inequality over t we get

$$\ln(Z_{T+1}) - \ln(Z_1) = \sum_{t=1}^{T} \ln \frac{Z_{t+1}}{Z_t} \leq -\eta \sum_{t=1}^{T} \mathbb{E}[\mathbb{1}_{[\hat{y}_t \neq y_t]}] + \frac{T\,\eta^2}{8}\,.$$

Next, we lower bound Z_{T+1}. For each i, let M_i be the number of mistakes h_i makes on the entire sequence of T examples. Therefore, $w_{T+1,i} = e^{-\eta M_i}$ and we get that

$$\ln Z_{T+1} = \ln\left(\sum_i e^{-\eta M_i}\right) \geq \ln\left(\max_i e^{-\eta M_i}\right) = -\eta \min_i M_i\,.$$

Combining the above and using the fact that $\ln(Z_1) = \ln(d)$ we get that

$$-\eta \min_i M_i - \ln(d) \;\leq\; -\eta \sum_{t=1}^{T} \mathbb{E}[\mathbb{1}_{[\hat{y}_t \neq y_t]}] + \frac{T\,\eta^2}{8}\,,$$

which can be rearranged as follows:

$$\sum_{t=1}^{T} \mathbb{E}[\mathbb{1}_{[\hat{y}_t \neq y_t]}] - \min_{h \in \mathcal{H}} M_i \;\leq\; \frac{\ln(d)}{\eta} + \frac{\eta\, T}{8}\,.$$

Plugging the value of η into the above concludes our proof. ■

Comparing the regret bound of `Weighted-Majority` in Theorem 29 to the mistake bound of `RandConsistent` given in Theorem 27 we note several differences. First, the result for `Weighted-Majority` holds for any sequence of examples while the mistake bound of `RandConsistent` assumes that the labels are generated by some $h^\star \in \mathcal{H}$. As a result, the bound of `Weighted-Majority` is relative to the minimal number of mistakes a hypothesis $h \in \mathcal{H}$ makes. Second, dividing both bounds by the number of rounds T, we get that the error rate in Theorem 27 decreases as $\ln(|\mathcal{H}|)/T$ while the error rate in Theorem 29 decreases as $\sqrt{\ln(|\mathcal{H}|)/T}$. This is similar to the results we had for PAC learning in the realizable and non-realizable cases.