582669 Supervised Machine Learning

lectures in Spring 2011, period III

Jyrki Kivinen

Relationship to other courses etc.

- elective master's level course in specialisation area *Algorithms and Machine Learning*, continues from *Introduction to Machine Learning*
- *Introduction to Machine Learning* is not a necessary pre-requisite but helps a lot in (a) understanding the basic concepts and (b) seeing the larger context.
- Students are assumed to have basic knowledge of linear algebra, probability theory and calculus.
- We will need a little bit of multivariate calculus but explain the needed concepts when we get there.
- We don't need any sophisticated data structures or algorithmic techniques.
- Homework will include small computer exercises. Familiarity with, e.g., Matlab, Octave or R will help.
- other courses in this area include *Unsupervised Machine Learning* and *Probabilistic Models*

Practicalities

- Grade consists of homework (20%) and exam (80%).
- Homework needs to be turned in in writing each week (details to be announced soon).
- There is no textbook. Lecture notes will appear on the course web page. In addition you may be required to read some original articles.

Contents (general)

Supervised machine learning is a wide area. We cover a variety of topics chosen partly based on the lecturer's personal preferences.

Our point of view is largely that of computational learning theory: we are interested in provable performance guarantees for learning algorithms.

We consider mainly (binary) classification.

See lecture notes of *Introduction to Machine Learning* for

- other types of learning (unsupervised learning, reinforcement learning, ...)
- other aspects of supervised learning and different algorithms (regression, decision trees, nearest neighbour, ...)
- techniques one needs in addition to a learning algorithm (preprocessing, cross-validation, ...)
- typical applications.

Table of Contents (preliminary)

1. Introduction

We discuss briefly the general setting of machine learning and some closely related disciplines.

On a more technical level, we get familiar with some basic concepts:

- supervised learning, classification, loss function
- online learning: the basic scenario and some introductory results
- statistical learning: the basic scenario and some introductory results.

What is supervised machine learning?

For the purposes of this course,

- learning is how a system improves its performance in some task based on some observations
- in machine learning this is performed by some algorithm
- in supervised machine learning the task is assigning labels to given objects.

As a representative supervised learning task, consider the following:

- We wish to build a system that inputs a digitized image and decides whether the image represents a human face.
- Here the objects are images and labels are "face" and "non-face."
- We train (teach) the system by giving it a large set of images that have been labelled ("face" or "non-face") by a human "expert."
- After training, the system hopefully can correctly label also images that were not in the training data.

We shall soon give a more precise definition of the supervised machine learning task, and in particular what are the criteria for evaluating learning algorithms.

Before that, let's briefly look at machine learning from a more general point of view.

Machine learning and related fields

Historically, machine learning is part of artificial intelligence.

Artificial intelligence is sometime characterized as the study of computational problems on which humans still outperform computers. The classical "machine learning approach" to such problems is to have a human create examples of desired behoviour, and let the computer come up with a general algorithm.

Typical applications include

- finding meaningful information from low-level observation data (machine vision)
- building complicated logical models, such as expert systems
- planning in games.

Learning has of course been studied intensively in fields such as psychology, neurobiology and cognitive science

- goal: find out how humans and other organisms actually learn.
- very difficult, only very limited things can be directly measured
- different from the machine learning problem of finding out what is a good way to learn (for machines)
- models from neurobiology etc. can still provide inspiration for machine learning
- even "biologically motivated" machine learning approaches (such as so-called neural networks) are quite different from actual biological learning.

Similar problems arise in other areas, too:

Signal processing

- need to adapt to changes in channel properties
- very specific problems and constraints
- well-developed theory, established technologies
- similar to online learning.

Statistics

- a lot of classical statistics concerns hypothesis testing etc.
- emphasis often in getting results in form that humans can understand
- however there is a significant body of statistics research where at least the basic setting is similar to machine learning
- increase in computational power enables new kinds of methods.

Data mining

- emerged much later than statistics and signal processing
- starting point often in data bases: what extra value can we get from our huge masses of data?
- usually want results that are intelligible to humans
- problem setting often quite loose ("are there any interesting patterns in the data?").

Generally there is no point in trying to define exact boundaries between these disciplines:

- often just different emphasis in similar problems
- theoretical frameworks, new algorithms etc. propagate from one area to another
- however there may be significant communication difficulties between researches from different fields.

Supervised learning—basic concepts

The most basic components of a supervised learning scenario are

- an input space X, the elements of which are also called instances
- an output space Y, the elements of which are often called labels.

Typically, the learning algorithm

- receives as input a sample (also called the training set) $((x_1, y_1), \ldots, (x_m, y_m))$ of m labelled instances, where $x_i \in X$ and $y_i \in Y$ for all i
- outputs a hypothesis $h\colon X \to Y$.

Intuitively, if we then receive further examples (the test set) $(x_i, y_i) \in X \times Y$, $i = m + 1, \ldots$, from the same source that produced the sample, we expect that the predictions $h(x_i)$ of our hypothesis are close to the correct labels y_i. We will soon make this more precise.

Loss functions

We use a loss function L to quantitatively evaluate the goodness of the hypothesis. If we predict with $\widehat{y} = h(x)$, but the correct label is y, we incur loss $L(y, \widehat{(y)})$. Usually $L(y, y) = 0$ (no loss if prediction was exactly correct) and $L(y, \widehat{y}) > 0$ if $\widehat{y} \neq y$.

We naturally need different loss functions for different output spaces Y. In classification, Y is a small finite set and the labels $y \in Y$ are not assumed to have any further structure or meaning. The labels are called classes. The most basic loss function for classification is the discrete or zero-one loss

$$L_{0-1}(y, \widehat{y}) = \begin{cases} 0 & \text{if } y = \widehat{y} \\ 1 & \text{if } y \neq \widehat{y}. \end{cases}$$

That is, we simply count mistakes.

The case $|Y| = 2$ is called binary classification. We usually choose as labels $Y = \{-1, 1\}$ or $Y = \{0, 1\}$, whichever allows for most convenient notations.

Binary classification is (in particular in computational learning theory and artificial intelligence) often also called concept learning. The explanation is that subsets $c \subseteq X$ of the input space are traditionally called concepts, and a concept $c \subseteq X$ can naturally be identified with the binary classifier

$$f(x) = \begin{cases} 1 & \text{if } x \in c \\ 0 & \text{otherwise.} \end{cases}$$

Many classification algorithms produce besides a prediction of the most likely label also some estimate of how reliable the prediction is.

For example, with $Y = \{0, 1\}$, if the algorithm knows it is just guessing, it could signal this with output $\widehat{y} = 1/2$. However, the discrete loss function does not support this.

One possible refinement (for $Y = \{0, 1\}$) is to allow continuous-valued predictions $0 \leq p \leq 1$ and use absolute loss

$$L_{\text{abs}}(y, p) = |y - p| \,.$$

Notice that $L_{\text{abs}}(y, p)$ is the expectation of the discrete loss $L_{\text{abs}}(y, \widehat{y})$ if we choose $\widehat{y} = 1$ with probability p and $\widehat{y} = 0$ with probability $1 - p$.

We will later see how using the absolute loss may lead to more meaningful performance bounds.

Another possibility for $y \in \{0, 1\}$ and $0 \leq p \leq 1$ is the logarithmic loss

$$L_{\log}(y, p) = \begin{cases} -\ln(1-p) & \text{if } y = 0 \\ -\ln p & \text{if } y = 1. \end{cases}$$

(The base of the logarithm does not really matter. Here we use natural logarithm ln; another popular choice is $\log_2$.)

The logarithmic loss comes from information theory. If we observe an event A that had probability p of occuring, we assign $-\ln p$ as the information content of the observation. Thus, for $y \in \{0, 1\}$, the logarithmic loss $L_{\log}(y, p)$ is the information content of observing y when the probability of 1 is p and probability of 0 is $1 - p$.

There are various reasons why

$$-\ln P(A)$$

is a good measure for the information content of event A. Let us here just notice that if A and B are two independent events, then $P(A \cap B) = P(A)P(B)$ and

$$-\ln P(A \cap B) = -\ln P(A) - \ln P(B).$$

In other words, with this definition the information content of two independent events together is the sum of the information contents of the individual events. This is a nice and intuitive property to have.

Both absolute and logarithmic loss can be generalised to multi-class classification (i.e. case $|Y| > 2$).

Let $Y = \{ b_1, \ldots, b_k \}$.

We now allow a prediction $\boldsymbol{p}$ to be a probability distribution over Y. In other words, $\boldsymbol{p} = (p_1, \ldots, p_k)$ where $p_i \geq 0$ for all i and $\sum_i p_i = 1$. We interpret p_i as the probability given by the algorithm for the label b_i.

Analogously, we represent the label $y = b_i$ by a vector $\boldsymbol{y}$ where $y_i = 1$ and $y_j = 0$ for $j \neq i$. This $\boldsymbol{y}$ can be interpreted as a probability distribution where the label b_i occurs with probability 1.

We can now take as loss $L(\boldsymbol{y}, \boldsymbol{p})$ any distance measure between the distribution $\boldsymbol{y}$ and $\boldsymbol{p}$. Popular distance measures include the variation distance

$$d_{\mathsf{var}}(\boldsymbol{y}, \boldsymbol{p}) = \frac{1}{2} \sum_i |y_i - p_i|$$

and Kullback-Leibler divergence (or relative entropy)

$$d_{\mathsf{KL}}(\boldsymbol{y}, \boldsymbol{p}) = \sum_i y_i \ln \frac{y_i}{p_i}.$$

In the special case $k = 2$, variation distance becomes absolute loss and Kullback-Leibler divergence becomes logarithmic loss. (We interpret the prediction $0 \le p \le 1$ used for absolute and logarithmic loss as the probability distribution $(1 - p, p)$ over the set $\{ 0, 1 \}$.)

We use the term hard classification to denote the case where the hypothesis is required to output just a single label (so we use zero-one loss).

Soft classification is the case where the hypothesis outputs probability distributions.

Besides classification, the other major type of supervised learning is regression.

In regression, $Y = \mathbb{R}$ (or $Y = [a, b]$ for some $a, b \in \mathbb{R}$). The most commonly used loss function is the squared loss

$$L_{\text{sq}}(y, \hat{y}) = (y - \hat{y})^2.$$

On this course we will not say much about regression.

Linear classification

We mainly consider the case where instances are real vectors, i.e. $X \subseteq \mathbb{R}^n$ for some n (including the special cases $X = \{0,1\}^n$ and $X = \{-1,1\}^n$). Linear classifiers are a very useful class of hypotheses for binary classification of real vectors.

A linear classifier $f\colon X \to Y$ has the form

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}),$$

where

- $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ is the weight vector
- $\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{i=1}^{n} w_i x_i$
- $\text{sign}(z) = 1$ if $z \geq 0$ and $\text{sign}(z) = -1$ otherwise (here we have taken $Y = \{-1,1\}$).

Often we include as linear classifiers also those with a slightly more general form

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} - b),$$

where we have included an extra parameter $b \in \mathbb{R}$ called the bias or threshold.

If instead of hard classifications $\widehat{y} = \pm 1$ we want probabilities $0 \leq p \leq 1$, we can replace the sign function for example with the logistic sigmoid:

$$f(\boldsymbol{x}) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} - b),$$

where

$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

We will later see several examples of learning algorithms that produce (usually hard) linear classifiers.

Supervised learning—summary of basic concepts

- A supervised learning algorithm produces a hypothesis h, which maps instances $x \in X$ into labels $h(x) \in Y$.
- The hypothesis is based on a sample $((x_1, y_1), \ldots, (x_m, y_m))$, where $x_i \in X$ and $y_i \in Y$.
- The performance of hypothesis h on an example (x, y) is evaluated in terms of the loss $L(y, h(x))$.
- Typical loss functions include zero-one loss (discrete loss) for hard classification, absolute loss and logarithmic loss for soft classification and square loss for regression.
- Linear classifiers are a useful class of hypotheses for binary classification of real vectors.

We now have sufficient concepts to state formally our two main learning models: online and statistical learning. We start with the online model which is mathematically simpler. We illustrate both models by some very basic algorithms.

Online learning: basics

In online learning, we do not produce just a single hypothesis based on a sample. The sample points are given one by one, and the hypothesis is updated after each of them.

More precisely, learning proceeds as a sequence of trials. At trial (or time step) t, for $t = 1, \ldots, T$ where the horizon T may or may not be known in advance, the following takes place

1. the algorithm receives the input $x_t \in X$
2. the algorithm outputs the prediction $\widehat{y}_t = h_t(x_t) \in Y$, where the hypothesis h_t comes from trial $t - 1$ (and h_1 is some arbitrary initial hypothesis)
3. the algorithm receives the correct label $y_t \in Y$ and incurs the loss $L(y_t, \widehat{y}_t)$
4. the algorithm updates the hypothesis h_t into h_{t+1} based on x_t, y_t and possibly some information remembered from previous trials.

For an online learning algorithm A on example sequence $S = ((x_1, y_1), \ldots, (x_T, y_T))$, we define the total loss in terms of a loss function L as

$$L^{\leq T}(S, A) = \sum_{t=1}^{T} L(y_t, \hat{y}_t).$$

This gives a well-defined numerical performance measure, but it is not at all obvious what values of the total loss can be considered "good" performance and what are "poor."

We resolve this issue by considering relative loss bounds, where we relate the total loss of the algorithm to the total losses of some fixed hypothesis $h \colon X \to Y$, defined analogously as

$$L^{\leq T}(S, h) = \sum_{t=1}^{T} L(y_t, h(x_t)).$$

This becomes clearer after some examples.

Online learning: the halving algorithm

For a relative loss bound, we first fix some finite class of hypotheses H, where each $h \in H$ is a function $X \to Y$. It does not matter what the hypotheses $h \in H$ are. However the following analysis is most interesting if $|H|$ is fairly large and we suspect that at least one $h \in H$ is fairly accurate for the data we have at hand, while most of them are likely to be quite bad. We present an algorithm for finding the "good" hypothesis from among the "bad" ones.

We consider binary classification with $Y = \{-1, 1\}$. To get started, assume a simplified scenario where one of the hypotheses $h_* \in H$ is actually perfect: we have $y_t = h_*(x_t)$ for all t. Don't worry too much about this assumption, we shall soon remove it. However the point is that the algorithm does not know which of the hypotheses is the perfect one.

Consider the following algorithm.

Algorithm 1.1 (The Halving Algorithm):

Initialise $V \leftarrow C$.
For $t = 1, \ldots, T$:

1. Input $x_t \in X$.
2. $V^- \leftarrow \{ h \in V \mid h(x_t) = -1 \}$ and
   $V^+ \leftarrow \{ h \in V \mid h(x_t) = 1 \}$
3. If $|V^-| > |V^+|$ then output $\widehat{y}_t = -1$
   else output $\widehat{y}_t = 1$.
4. Input y_t.
5. If $y_t = -1$ then $V \leftarrow V^-$, else $V \leftarrow V^+$.

The set V maintained by the algorithm is called the version space. It consists of hypotheses that are consistent, i.e., have not made a mistake yet.

In Step (3) the prediction is decided by a majority vote among the consistent hypotheses.

In Step (5) the version space is updated. The assumption that there is a perfect hypothesis guarantees that the version space never gets empty.

Theorem 1.2: If there is $h_* \in H$ such that $y_t = h_*(x_t)$ for all t, then the Halving Algorithm (HA) makes at most $\log_2 |H|$ mistakes, i.e.,

$$L_{0-1}^{\leq T}(S, \mathrm{HA}) \leq \log_2 |H| \,.$$

Proof: HA predicts as the majority of the version space. If the HA made a mistake, so did at least half of the consistent hypotheses, so the size of the version space is reduced at least by a factor $1/2$.

Hence, after HA has made M mistakes, the size $|V|$ of the version space is at most $|H|\,(1/2)^M$.

However the version space never gets empty, so $|H|\,(1/2)^M \geq 1$. This yields $M \leq \log_2 |H|$. $\square$

Notice a fundamental idea used in online learning: we cannot control when a mistake happens, but we can make sure that when one happens, we "learn" a lot.

Online learning: the Weighted Majority algorithm

In the more realistic case that no perfect hypothesis exists, the Halving Algorithm is useless, because the version space tends to get empty.

The solution is to use a "soft" version space. We maintain a weight for each hypothesis $h \in H$. Initially all weights are one, intuitively meaning that all hypotheses are fully in the version space. When a hypothesis h makes a mistake, we multiply its weight by a factor $0 < \beta < 1$, intuitively meaning that we reduce its presence in the "soft" version space. When voting for the prediction, the influence of a hypotheses is based on its current weight.

Thus we rely more on those hypotheses that up to now have not made too many mistakes.

We get the following.

Algorithm 1.3 (Weighted Majority, WM): Let $H = \{ h_1, \ldots, h_n \}$. At time t, hypothesis h_i has weight $w_{t,i} \in [0, 1]$. The algorithm has learning rate $0 < \beta < 1$ as parameter.

Initialise $w_{1,i} = 1$ for all i.
Repeat for $t = 1, \ldots, T$:

1. Input $x_t \in X$.
2. Let $P = \{ i \mid h_i(x_t) = 1 \}$ and $M = \{ i \mid h_i(x_t) = -1 \}$.
3. Calculate $W_t^+ = \sum_{i \in P} w_{t,i}$
   and $W_t^- = \sum_{i \in M} w_{t,i}$.
4. If $W_t^- > W_t^+$ then $\widehat{y}_t = -1$,
   else $\widehat{y}_t = 1$
5. For $i = 1, \ldots, n$:
   if $h_i(x_t) = y_t$ then $w_{t+1,i} = w_{t,i}$
   else $w_{t+1,i} = \beta w_{t,i}$.

Notice that using $\beta = 0$ would give the Halving Algorithm.

We analyse the performance of WM in terms a potential function $P_t = c \ln W_t$ where $W_t = W_t^+ + W_t^- = \sum_i w_{t,i}$ and the constant $c > 0$ will be fixed later.

Since W_t is intuitively the size of our soft version space, a drop in the potential corresponds to a reduction in the version space and thus learning. Again, we wish to show that whenever we make a mistake, we learn a lot.

This approach is somewhat similar to the use of potential function in amortised analysis.

Lemma 1.4: If

$$c = \left(\ln \frac{2}{1+\beta} \right)^{-1},$$

then for all t we have

$$L_{0-1}(y_t, \hat{y}_t) \leq c \ln \frac{W_t}{W_{t+1}} = -(P_{t+1} - P_t).$$

In other words, whenever the algorithm makes a mistake, the potential decreases by at least 1.

Proof: Since the potential never increases, the case $y_t = \widehat{y}_t$ is clear.

Consider for example $\widehat{y}_t = -1$ and $y_t = 1$, so $L_{0-1}(y_t, \widehat{y}_t) = 1$. Then $W_t^+ \leq W_t/2$, and

$$\begin{aligned} W_{t+1} &= W_t^+ + \beta W_t^- \\ &= (1-\beta) W_t^+ + \beta (W_t^+ + W_t^-) \\ &\leq (1-\beta)\frac{W_t}{2} + \beta W_t \\ &= \frac{1+\beta}{2} W_t, \end{aligned}$$

so the claim follows. □

Theorem 1.5: For all $0 < \beta < 1$ and all hypotheses $h \in H$ we have

$$L_{0-1}(S, \mathrm{WM}) \leq c\eta L_{0-1}(S, h) + c \ln n,$$

where again $n = |H|$,

$$c = \left(\ln \frac{2}{1+\beta} \right)^{-1} \quad \text{and} \quad \eta = \ln \frac{1}{\beta}.$$

Remark: Naturally the bound is tightest if we compare against the best hypothesis:

$$L_{0-1}(S, \mathrm{WM}) \leq c\eta \min_{h \in H} L_{0-1}(S, h) + c \ln n.$$

Notice that the theorem makes no assumption about the sequence S or the hypotheses $h \in H$. It simply states that if at least one hypothesis is good, then so is the algorithm (albeit worse by a constant factor $c\eta$ and term $c \ln n$). On the other hand, if all the hypotheses in H are bad, the bound is not very useful. However in such a situation it would not be reasonable to require good performance from a learning algorithm based on H, anyway.

We can optimise the bound by tuning β:

- slow learning:

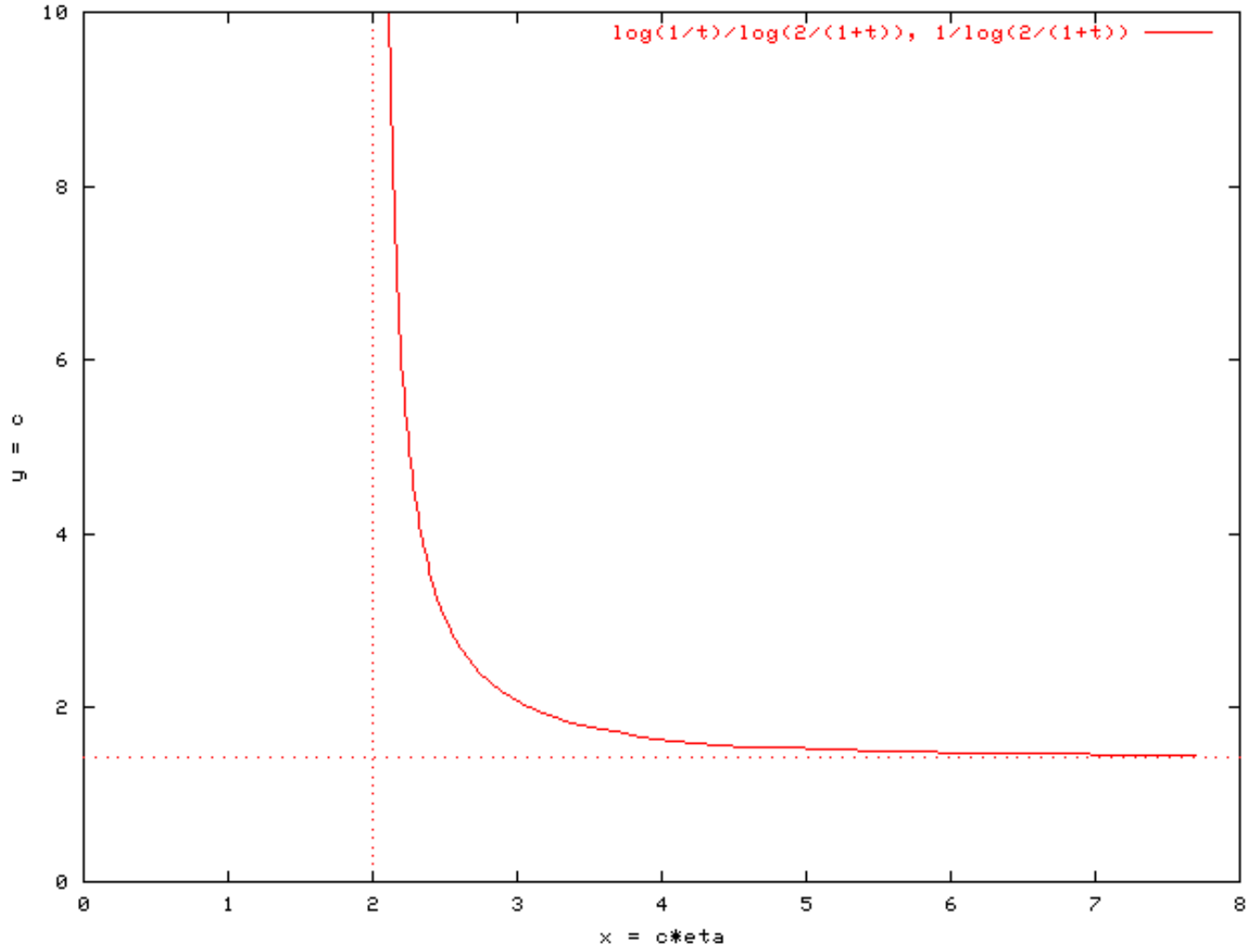
$$\lim_{\beta \to 1-} c\eta = 2 \quad \text{and} \quad \lim_{\beta \to 1-} c = \infty.$$

- fast learning:

$$\lim_{\beta \to 0+} c\eta = \infty \quad \text{and} \quad \lim_{\beta \to 0+} c = \frac{1}{\ln 2} \approx 1{,}44.$$

$\Rightarrow$ if all experts are bad, choose $\beta \approx 1$;
in one expert is really good, choose $\beta \approx 0$

Choosing the optimal β requires either (usually unrealistic) advance knowledge (how much is $\min_h L_{0-1}(S, h)$) or complicated tuning "on the fly". We shall return to this.



Curve $(x, y) = (c\eta, c)$, $0 < \beta < 1$; asymptotes $x = 2$ and $y = 1/\ln 2$.

Proof of Theorem: By summing the estimates from previous lemma for $t = 1, \ldots, T$ we get

$$\begin{aligned} L_{0-1}(S, \mathsf{WM}) &\leq \sum_{t=1}^{T} -(P_{t+1} - P_t) \\ &= P_1 - P_{T+1} \\ &= c \ln n - c \ln W_{T+1}. \end{aligned}$$

The claim follows, since for all $h_i \in H$ we have

$$\begin{aligned} W_{T+1} &\geq w_{t+1,i} \\ &= \beta^{L_{0-1}(S,h_i)} \\ &= \exp(-\eta L_{0-1}(S, h_i)). \end{aligned}$$

□

Weighted Majority vs. Bayes

We consider the following scenario:

- There are n hypotheses: $H = \{ h_1, \ldots, h_n \}$.
- The instances $x_1, \ldots, x_T$ are obtained somehow (doesn't matter how).
- We pick a target hypothesis $h_* \in H$ at random so that each h_i has probability $1/n$ of being picked.
- The labels $Y_1, \ldots, Y_T \in \{ 0, 1 \}$ are defined by h_* but subject to classification noise at some rate $0 < \nu < 1/2$:

$$Y_t = \begin{cases} h_*(x_t) & \text{with probability } 1 - \nu \\ 1 - h_*(x_t) & \text{with probability } \nu. \end{cases}$$

(We capitalise Y_i to emphasise that it is a random variables and use y_i to denote some particular value it gets.)

Let $M(t,i) = L_{0-1}^{\leq t}(S, h_i)$ be the number of mistakes made by h_i in the first t trials. If h_i is the target, then the probability of a mistake by h_i at any given trial is the same as the noise rate ν. Therefore

$$\Pr[Y_1 = y_1, \ldots, Y_t = y_t \mid h_* = h] = \nu^{M(t,i)}(1-\nu)^{t-M(t,i)} = \left(\frac{\nu}{1-\nu}\right)^{M(t,i)}(1-\nu)^t.$$

We write $\beta = \nu/(1-\nu)$, $\boldsymbol{Y} = (Y_1, \ldots, Y_t)$ and $\boldsymbol{y} = (y_1, \ldots, y_t)$ and apply Bayes's formula:

$$\begin{aligned}
\Pr[h_* = h_i \mid \boldsymbol{Y} = \boldsymbol{y}] &= \frac{\Pr[\boldsymbol{Y} = \boldsymbol{y} \mid h_* = h_i]\Pr[h_* = h_i]}{\sum_j \Pr[\boldsymbol{Y} = \boldsymbol{y} \mid h_* = h_j]\Pr[h_* = h_j]} \\
&= \frac{\beta^{M(t,i)}(1-\nu)^t(1/n)}{\sum_j \beta^{M(t,j)}(1-\nu)^t(1/n)} \\
&= \frac{\beta^{M(t,i)}}{\sum_j \beta^{M(t,j)}} \\
&= \frac{w_{t,i}}{\sum_j w_{t,j}}
\end{aligned}$$

where $w_{t,j}$ is the weight used in the WM algorithm.

Thus, the normalised weight

$$v_{t,i} = \frac{w_{t,i}}{\sum_j w_{t,j}}$$

of the WM algorithm is actually the posterior probability of hypothesis h_i being the target.

Further, the WM algorithm predicts 0 at trial t if the total posterior probability of hypotheses predicting 0 is larger than the total posterior probability of hypotheses predicting 1. This is the "Bayesian" thing to do, for hard classification.

(The algorithm actually compares unnormalised weights $w_{t,i}$, but that makes no difference since the normalisation factor $\sum_j w_{t,j}$ is the same for all hypotheses.)

We have seen that in this particular case, our online algorithm has a nice Bayesian interpretation. However this is not the case for online algorithms in general. Still, we can consider this as an additional motivation for the multiplicative update rule of the algorithm. (The other motivation is the loss bounds we can prove.)

Statistical learning: the basics

Fix some input space X, label set Y, hypothesis class H and loss function L. Suppose there is some fixed but unknown probability measure P on $X \times Y$. The true risk of hypothesis $h \in H$ is defined as

$$R(h) = \mathsf{E}_{(x,y)\sim P}[L(y, h(x))]$$

where $\mathsf{E}_{(x,y)\sim P}[\cdot]$ denotes expectation when (x, y) is drawn according to P.

Example 1.6: Let $X \subseteq \mathbb{R}^n$ and $Y = \mathbb{R}$, and let p_1 be some probability density function over X. Fix some function $h\colon X \to Y$ and a parameter $\sigma > 0$. We could now define a probability measure P over $X \times Y$ by a density function

$$p(x, y) = p_1(x)\phi(y; h(x), \sigma),$$

where $\phi(y; \mu, \sigma)$ is the density function of the normal distribution with mean μ and variance σ. That is, drawing (x, y) according to P consists of first drawing x according to p_1 and then drawing y from a normal distribution centered at $h(x)$ and with variance σ.

This is a fairly common model, but our scenario is more general since it allows the noise to have different form at different points x. □

In statistical learning,

- the input is a sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ of m examples $(x_i, y_i) \in X \times Y$ that are assumed to be drawn independently from P
- the output is some hypothesis $\widehat{h} \in H$
- the goal is find $\widehat{h}$ with as small true risk as possible.

This type of learning, where examples are obtained at the same time, is called batch learning, as opposed to online learning.

The problem is of course that we don't know P. However the sample gives us some partial information we can use.

Statistical learning: Empirical Risk Minimisation

Without knowing P, we cannot evaluate the true risk of a hypothesis. It is natural idea to approximate the true risk of h by the empirical risk

$$\hat{R}(h) = \frac{1}{m} \sum_{i=1}^{m} L(y_i, h(x_i)).$$

For any given h, the expectation of the empirical risk is the true risk, and for reasonably large m the variance is fairly small (for nice loss functions).

Empirical risk minimisation (ERM) means simply choosing as hypothesis $\hat{h}$ the one from H that has the smallest empirical risk.

The problem is that for large H, it's still quite possible that for some hypothesis h the empirical risk happens to be much smaller than the true risk, and such h might well be picked by ERM. This leads to a bad hypothesis, and an over-optimistic estimate of its performance.

Hence, some restriction on H is needed for ERM to work.

Statistical learning: noise-free PAC model

As with online learning, we consider here the case where H is finite, with $|H| = n$. Later in the course we consider infinite hypothesis spaces. Let $Y = \{0, 1\}$ and, for simplicity, let X be finite.

Again, we start with the unrealistic noise-free setting. We assume there is an unknown probability measure P_1 on X and an unknown target $h_* \in H$ such that

$$P(x, y) = \begin{cases} P_1(x) & \text{if } y = h_*(x) \\ 0 & \text{otherwise.} \end{cases}$$

With this assumption, there is always at least one hypothesis (namely the target h_*) that has zero empirical error. Hence, the ERM algorithm becomes

> Pick as $\widehat{h}$ any consistent $h \in H$ (i.e. one that makes no mistakes on the training sample).

Theorem 1.7: Let $\varepsilon > 0$ and $\delta > 0$ be arbitrary. In the noise-free case, if

$$m \geq \frac{1}{\varepsilon} \ln \frac{|H|}{\delta}$$

then with probability at least $1 - \delta$ any consistent hypothesis has true risk at most ε.

Comment: We call ε the accuracy parameter and δ the confidence parameter. The idea is that we consider it acceptable to have a true risk of ε or smaller. However, we must also allow that with probability δ we get an unrepresentative sample and the true risk of our hypothesis may be much higher.

Since $1/\delta$ appears only logarithmically in the bound, we can achieve high confidence with quite reasonable sample sizes. Decreasing ε is much more expensive. This happens also more generally in the bounds for statistical learning.

Proof: We say that a sample is bad if there is a hypothesis $h \in H$ such that

- h is consistent with S but
- h has true error larger than ε.

We need to show that the probability of bad samples is at most δ.

Consider first any fixed $h \in H$ that has true risk larger than ε. That is, the probability of drawing an example (x, y) such that $h(x) = y$ is less than $1 - \varepsilon$. If h is consistent with S, we have drawn m such examples. The probability of this is less than

$$(1 - \varepsilon)^m \leq \exp(-\varepsilon m) \leq \exp(-\ln(|H|/\delta)) = \frac{\delta}{|H|}.$$

There are at most $|H|$ different hypotheses h for which this could happen. Hence the probability that it happens to at least one of them is at most

$$|H| \cdot \frac{\delta}{|H|} = \delta.$$

(This is known as the union bound.) $\square$

The noise-free model is the basic version of PAC learning (Probably Approximately Correct) which was introduced by Valiant in 1983. Notice that in the theorem

- the sample size m grows polynomially in $1/\varepsilon$ and $1/\delta$.
- the probability measure P is unknown and arbitrary.

The PAC model has been very important in stimulating computational learning theory. However the assumption about no noise is usually quite unrealistic, and the sample size bounds one gets at least by basic techniques are impractically large.

Statistical learning: agnostic PAC model

In the noise-free PAC model, we assumed that the target had zero true risk. Our hypothesis had with high confidence risk at most ε.

Consider now a more realistic situation with no "perfect" target. We still have a hypothesis class H and a fixed but unknown probability measure P over $X \times Y$. In agnostic PAC learning, our goal is to find a hypothesis $\hat{h}$ such that

$$R(\hat{h}) \leq \min_{h \in H} R(h) + \varepsilon$$

holds with high confidence.

In other words, choosing the hypothesis $\hat{h}$ based purely on the sample increases the true risk only by ε compared to picking the optimal hypothesis using full knowledge of P.

Notice how this is somewhat similar in spirit to relative loss bounds in online learning.

The basic setting works for any loss function, but again we consider only classification with discrete loss in more detail.

Theorem 1.8: Assume $\hat{h} \in H$ is chosen by ERM and

$$m \geq \frac{2}{\varepsilon^2} \ln \frac{|H|}{\delta}.$$

Then with probability at least $1 - \delta$ we have

$$R(\hat{h}) \leq \min_{h \in H} R(h) + \varepsilon.$$

Proof: Let

$$h_* = \arg \min_{h \in H} R(h)$$

be the hypothesis which achieves minimal true risk. It is sufficient to show that with probability at least $1 - \delta$ we have $\hat{R}(h_*) \leq R(h_*) + \frac{\varepsilon}{2}$ and $\hat{R}(h) \geq R(h) - \frac{\varepsilon}{2}$ for all $h \in H - \{ h_* \}$. Namely, if these conditions hold and $R(h) > R(h_*) + \varepsilon$, then

$$\hat{R}(h) \geq R(h) - \frac{\varepsilon}{2} > R(h_*) + \varepsilon - \frac{\varepsilon}{2} \geq \hat{R}(h_*),$$

so ERM cannot choose h.

We use the following Hoeffding's Inequalities:

Lemma 1.9: Let $S = \sum_{i=1}^{m} X_i$, where the X_i are independent random variables with $a_i \leq X_i \leq b_i$ for some $a_i, b_i \in \mathbb{R}$. Then

$$\begin{aligned} \Pr[S \geq \mathrm{E}[S] + t] &\leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{m}(b_i - a_i)}\right) \\ \Pr[S \leq \mathrm{E}[S] - t] &\leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{m}(b_i - a_i)}\right). \end{aligned}$$

In our case, $m\hat{R}(h)$ is the sum of m independent random variables $L_{0-1}(y_i, h(x_i))$ that take values 0 and 1, and $\mathrm{E}[m\hat{R}(h)] = mR(h)$.

Now

$$\begin{aligned}
\Pr[\hat{R}(h) \leq R(h) - \varepsilon/2] &= \Pr[m\hat{R}(h) \leq mR(h) - m\varepsilon/2] \\
&\leq \exp\left(-\frac{2m^2\varepsilon^2/4}{m}\right) \\
&= \exp\left(-\frac{m\varepsilon^2}{2}\right) \\
&\leq \frac{\delta}{|H|}.
\end{aligned}$$

We apply this for $h \neq h_*$. Similarly we see that

$$\Pr\left[\hat{R}(h) \geq R(h) + \frac{\varepsilon}{2}\right] \leq \frac{\delta}{|H|},$$

and we apply this for $h = h_*$. The union bound now gives the desired result. □

By slightly modifying the proof of Theorem 1.8 we get the following uniform convergence result.

Theorem 1.10: Let

$$m \geq \frac{1}{2\varepsilon^2} \ln \frac{2|H|}{\delta}.$$

Then with probability at least $1 - \delta$ we have

$$\left|\widehat{R}(h) - R(h)\right| \leq \varepsilon \qquad \text{for all } h \in H.$$

In other words, with probability at least $1 - \delta$ we have

$$\max_{h \in H} \left|\widehat{R}(h) - R(h)\right| \leq \varepsilon.$$

Remark: Notice that here and in Theorem 1.8 the bound for m is $O(1/\varepsilon^2)$, while for the noise-free PAC model (Theorem 1.7) it was only $O(1/\varepsilon)$. This is a real difference between the noise-free and agnostic model, not just some slackness in our analysis.

Remark: The interesting part is getting similar results for infinite H.

Proof: From Hoeffding's Inequality we have

$$\begin{aligned}
\Pr[\hat{R}(h) \leq R(h) - \varepsilon] &= \Pr[m\hat{R}(h) \leq mR(h) - m\varepsilon] \\
&\leq \exp\left(-\frac{m^2\varepsilon^2}{m}\right) \\
&= \exp\left(-2m\varepsilon^2\right) \\
&\leq \frac{\delta}{2\,|H|},
\end{aligned}$$

and similarly

$$\Pr[\hat{R}(h) \geq R(h) + \varepsilon] \leq \frac{\delta}{2\,|H|}.$$

We apply the union bound to these $2\,|H|$ inequalities. □

2. Online learning

We see more examples of online learning algorithms:

- the Aggregating Algorithm that generalises the Weighted Majority
- the Perceptron Algorithm for learning linear classifiers.

We get more familiar with the basic analysis techniques

- relative loss bounds
- potential functions
- margin-based bounds for linear classifiers.

Combining expert advice

First we want to slightly generalise the scenario underlying the Weighted Majority algorithm.

Consider the following (unrealistic) example:

On each day t of the year, we want to predict whether the stock market goes up ($y_t = 1$) or down ($y_t = -1$).

There are n experts $\mathcal{E}_i$, $i = 1, \ldots, n$, who are also predicting this. Let $\mathcal{E}_{i,t} \in \{-1, 1\}$ be the prediction of expert $\mathcal{E}_i$ for day t.

On day t, for $t = 1, \ldots, T$, we

1. receive the experts' predictions $\mathcal{E}_{1,t}, \mathcal{E}_{2,t}, \ldots, \mathcal{E}_{n,t}$
2. make our own prediction $\widehat{y}_t \in \{-1, 1\}$ based on the expert advice we just received
3. receive the outcome $y_t \in \{-1, 1\}$, suffer loss $L_{0-1}(y_t, \widehat{y}_t)$ and possibly update our rule for producing $\widehat{y}$ in Step 2.

Notice that the original scenario of the Weighted Majority algorithm is a special case. There we had $\mathcal{E}_{i,t} = h_i(x_t)$. Now we allow arbitrary $\mathcal{E}_{i,t}$.

In our stock market setting, it could be that

- most experts are analysts making their predictions based on news they gather from different public sources
- some experts are analysts who have different sources of inside information
- a few experts might be just tossing a coin
- one expert might be an evil mastermind manipulating the whole market.

It makes no difference where the experts' predictions come from, as long as we have them all available every morning.

Almost realistic examples

Task: managing a stock portfolio
Experts: "invest all money in stock x", for different stocks x

Task: disk spin-down in a laptop
Experts: "spin down after x seconds idle" for a set of values x

Task: virtual memory paging
Experts: RAND, FIFO, LIFO, LRU, MRU, ...

"Almost realistic" means that the algorithms we shall soon see have worked well in realistic simulations.

All these examples have additional complications because the real-world cost of false predictions is not just L_{0-1}.

Notice that h_i and x_t appear in the original Weighted Majority algorithm only in combination $h_i(x_t)$. By replacing $h_i(x_t) \leftarrow \mathcal{E}_{t,i}$ we get the following algorithm for the experts setting.

Initialise $w_{1,i} = 1$ for all i.
Repeat for $t = 1, \ldots, T$:

1. Input $x_t \in X$.
2. Let $P = \{\, i \mid \mathcal{E}_{i,t} = 1 \,\}$ and $M = \{\, i \mid \mathcal{E}_{i,t} = -1 \,\}$.
3. Calculate $W_t^+ = \sum_{i \in P} w_{t,i}$
   and $W_t^- = \sum_{i \in M} w_{t,i}$.
4. If $W_t^- > W_t^+$ then $\widehat{y}_t = -1$,
   else $\widehat{y}_t = 1$
5. For $i = 1, \ldots, n$:
   if $\mathcal{E}_{i,t} = y_t$ then $w_{t+1,i} = w_{t,i}$
   else $w_{t+1,i} = \beta w_{t,i}$.

Let us reconsider also the loss bound (Theorem 1.5) for Weighted Majority:

$$L_{0-1}(S, \mathrm{WM}) \leq c\eta L_{0-1}(S, h_i) + c \ln n.$$

for all i.

Rewriting $L_{0-1}(S, h_i) = \sum_{t=1}^{T} L_{0-1}(y_t, h_i(x_t))$, we see that in the proof, too, the hypotheses h_i and inputs x_t appear only in combination $h_i(x_t)$. Writing again $h_i(x_t) = \mathcal{E}_{i,t}$, we get the bound

$$L_{0-1}(S, \mathrm{WM}) \leq c\eta \sum_{t=1}^{T} L_{0-1}(y_t, \mathcal{E}_{i,t}) + c \ln n.$$

We conclude that the Weighted Majority algorithm satisfies the loss bound

$$L_{0-1}(S, \mathrm{WM}) \leq c\eta \min_{1 \leq i \leq n} \sum_{t=1}^{T} L_{0-1}(y_t, \mathcal{E}_{i,t}) + c \ln n.$$

also in the expert setting, where the experts' predictions $\mathcal{E}_{i,t}$ can be anything. As long as at least one expert has small loss, so has the WM algorithm.

Lower bound for Weighted Majority

We noticed earlier that in the bound

$$L_{0-1}(S, \mathrm{WM}) \leq c\eta \min_{1 \leq i \leq n} \sum_{t=1}^{T} L_{0-1}(y_t, \mathcal{E}_{i,t}) + c \ln n$$

the coefficients $c\eta$ and c are such that

$$c\eta \geq 2 \qquad \text{and} \qquad c \geq \frac{1}{\ln 2}.$$

Can we improve upon these constants? We prove a (partial) negative answer.

Theorem 2.1: Let A be any online prediction algorithm for combining expert advice. There are arbitrarily large values n and M, and sequences of experts' predictions $\mathcal{E}_{i,t}$ and outcomes y_t, such that for which it is possible that

- there are n experts and
- the best expert makes at most M mistakes but
- the algorithm A makes at least

$$2M + \log_2 \frac{n}{2} = 2M + \frac{1}{\ln 2} \ln n - 1$$

mistakes.

Proof: Let $n = 2^{k+1}$ for some $k \geq 1$, and $T = 2M + k$. Choose T predictions $\mathcal{E}_{i,t}$ for n experts as follows:

- For $1 \leq t \leq k$ let $\mathcal{E}_{i,t} = \mathcal{E}_{i+2^k,t} = 1$ if bit t in the binary representation of i is 1, and $\mathcal{E}_{i,t} = \mathcal{E}_{i+2^k,t} = -1$ otherwise.

- For $k + 1 \leq t \leq T$ let $\mathcal{E}_{i,t} = 1$ and $\mathcal{E}_{i+2^k,t} = -1$.

Now for any sequence of outcomes $(y_1, \ldots, y_T)$

- there is one i such that both $\mathcal{E}_i$ and $\mathcal{E}_{i+2^k}$ predict the first k answers correctly and

- for all i at least one of $\mathcal{E}_i$ or $\mathcal{E}_{i+2^k}$ get at least half the answers $k + 1, \ldots, T$ right.

Hence, the best expert makes at most M mistakes.

On the other hand, for given A and $\mathcal{E}_i$ we can always pick an answer sequence such that A always makes a mistake (take $y_t = -\widehat{y}_t$). □

Example 2.2: $k = 2$, $n = 2^{k+1} = 8$, $M = 3$; $T = k + 2M = 8$.

The predictions of the experts are as follows:

t	1	2	3	4	5	6	7	8
$\mathcal{E}_1$	-1	-1	1	1	1	1	1	1
$\mathcal{E}_2$	1	-1	1	1	1	1	1	1
$\mathcal{E}_3$	-1	1	1	1	1	1	1	1
$\mathcal{E}_4$	1	1	1	1	1	1	1	1
$\mathcal{E}_5$	-1	-1	-1	-1	-1	-1	-1	-1
$\mathcal{E}_6$	1	-1	-1	-1	-1	-1	-1	-1
$\mathcal{E}_7$	-1	1	-1	-1	-1	-1	-1	-1
$\mathcal{E}_8$	1	1	-1	-1	-1	-1	-1	-1

Suppose the correct answers are

$$(y_1, \ldots, y_8) = (1, -1, 1, -1, -1, -1, -1, 1).$$

After 2 rounds, $\mathcal{E}_2$ and $\mathcal{E}_6$ have no mistakes.

After the whole sequence, $\mathcal{E}_6$ has done $2 \leq M$ mistakes (and $\mathcal{E}_2$ $4 = 2M - 2$ mistakes).

Thus, our upper bound is fairly tight in worst case. How realistic is "worst case"?

Intuitively the term $(\ln 2)^{-1} \ln n = \log_2 n$ seems right: to find the best out of n experts we need at least $\log_2 n$ bits of information.

On the other hand, the factor 2 in $2L_{0-1}(\mathcal{E}_i)$ seems a bit dubious. In our lower bound it clearly is a result of looking very hard for a worst-case y_t and forcing the algorithm to make hard predictions.

For the learning result, the factor 2 implies that even with large amounts of data the algorithm is not guaranteed to converge towards the best expert, which is bad. We address this by considering soft classification with absolute loss.

The Aggregating Algorithm

We generalise the Weighted Majority algorithm for loss functions that can in principle be arbitrary, and show that this actually works for the absolute loss.

First notice that the update rule

$$w_{t+1,i} = \begin{cases} w_{t,i} & \text{if } y_t = \mathcal{E}_{i,t} \\ \beta w_{t,i} & \text{if } y_t \neq \mathcal{E}_{i,t} \end{cases}$$

can be written as

$$w_{t+1,i} = w_{t,i} \exp(-\eta L_{0-1}(y_t, \mathcal{E}_{i,t}))$$

where $\eta = -\ln \beta$. We can now obviously replace L_{0-1} with some other loss function.

The fundamental part of the analysis was defining a potential function

$$P_t = c \ln W_t = c \ln \sum_{i=1}^{n} w_{t,i}$$

and showing that for a suitable c the majority vote $\widehat{y}$ satisfies

$$L_{0-1}(y_t, \widehat{y}_t) \leq P_t - P_{t+1} = c \ln \frac{W_t}{W_{t+1}}.$$

Now we are doing soft classification and wish to allow $0 \leq \widehat{y} \leq 1$. Can we find $\widehat{y}$ such that

$$L_{\mathsf{abs}}(y_t, \widehat{y}_t) \leq c \ln \frac{W_t}{W_{t+1}}$$

holds for some c? Notice that we still assume $y_t \in \{0, 1\}$. However we can allow also the experts to produce soft classifications $0 \leq \mathcal{E}_{t,i} \leq 1$.

Lemma 2.3: Choose any $\eta > 0$ and

$$c \geq \left(2 \ln \frac{2}{1 + e^{-\eta}} \right)^{-1}.$$

Then for any set of weights $w_{t,i}$, $i = 1, \ldots, n$, and experts' predictions $\mathcal{E}_{t,i}$ there is a prediction $\widehat{y}_t$ such that

$$L_{\mathsf{abs}}(y_t, \widehat{y}_t) \leq c \ln \frac{W_t}{W_{t+1}}$$

holds for any $y_t \in \{ 0, 1 \}$.

Remark: At the start of the trial, we know $w_{t,i}$ and $\mathcal{E}_{i,t}$, but of course not y_t. Knowing $w_{t,i}$, we also know W_t, but not W_{t+1}. However, if we knew y_t, we could use $\mathcal{E}_{i,t}$ to calculate W_{t+1} and thus see for which values $\widehat{y}_t$ the inequality holds. The idea is to simply consider both possible values $y_t \in \{ 0, 1 \}$ separately and combine the conditions.

In the proof we know a basic result known as Jensen's inequality.

Theorem 2.4 (Jensen's Inequality): Let X be an arbitrary real-valued random variable and f a convex real-valued function defined on an interval that includes the range of X (which need not be bounded). Then

$$f(\mathrm{E}X) \leq \mathrm{E}f(X)$$

where E denotes expectation.

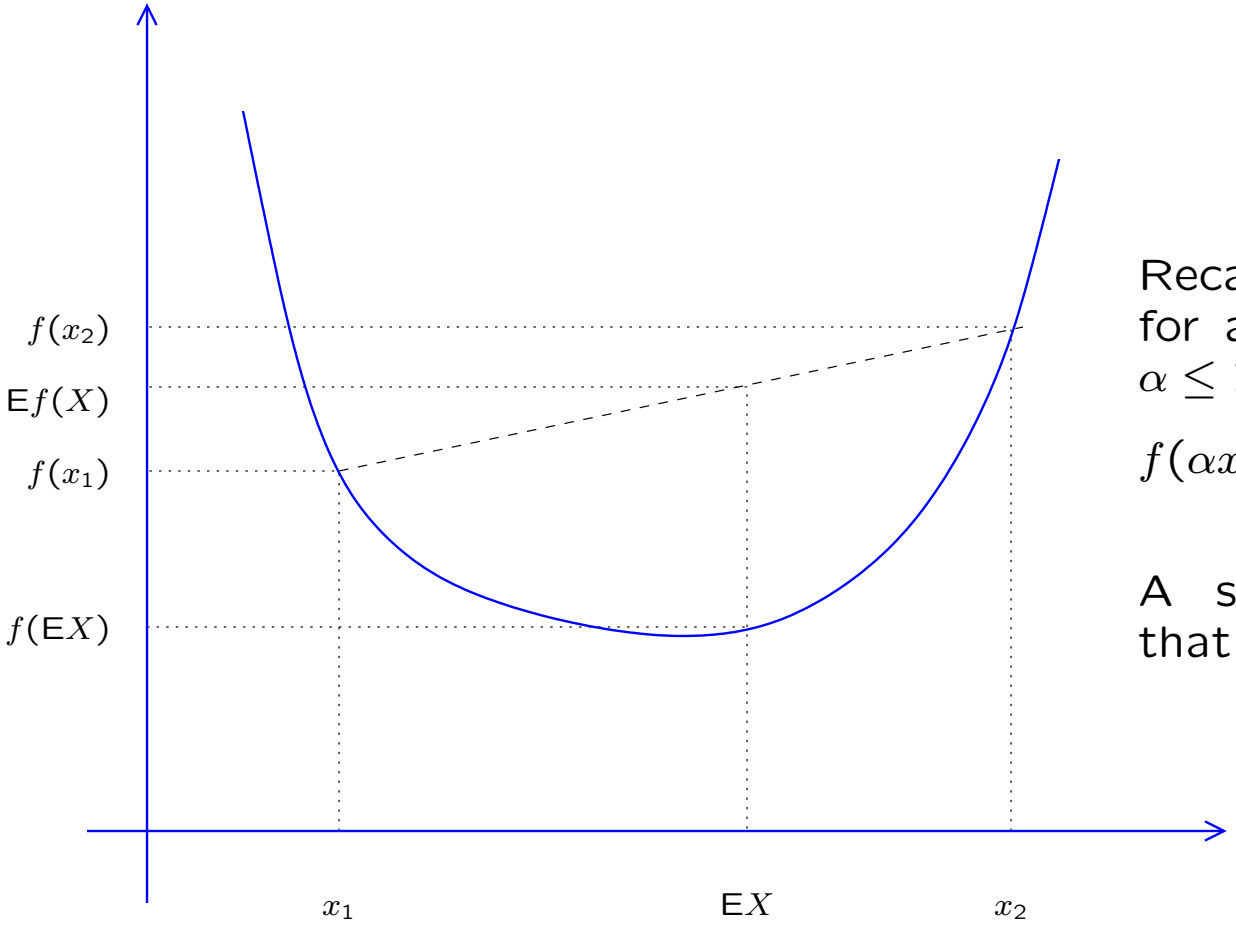
"Proof": The case where X has two possible values x_1 and x_2 follows directly from the definition of convexity. (See picture on next page.) More general case by induction and continuity. □

Corollary 2.5: If f is convex then

$$f\left(\sum_i v_i x_i\right) \leq \sum_i v_i f(x_i)$$

for all $\boldsymbol{x}$ and $\boldsymbol{b}$ such that $\sum_i v_i = 1$ and $v_i \geq 0$. □

If f is concave, the inequality flips the other way (and the best way to remember the right way is to remember the picture on next page).



Recall that f is convex if for all x_1 and x_2 and $0 \leq \alpha \leq 1$ we have

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2).$$

A sufficient condition is that $f''(x) \geq 0$ for all x.

Proof of Lemma: For $y \in \{0, 1\}$, let

$$W_{t+1}(y) = \sum_{i=1}^{n} w_{t,i} \exp(-\eta L_{\text{abs}}(y, \mathcal{E}_{i,t}))$$

be the value that W_{t+1} would take if $y_t = y$. Define normalised weights $v_{t,i} = w_{t,i}/W_t$, for which $\sum_i v_i = 1$. Then

$$\ln \frac{W_t}{W_{t+1}(y)} = -\ln \frac{W_{t+1}(y)}{W_t} = -\ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta L_{\text{abs}}(y, \mathcal{E}_{i,t})).$$

Since $L_{\text{abs}}(0, \widehat{y}) = \widehat{y}$ and $L_{\text{abs}}(1, \widehat{y}) = 1 - \widehat{y}$, we need to have

$$\begin{aligned} \widehat{y} &\leq -c \ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta \mathcal{E}_{i,t}) \\ 1 - \widehat{y} &\leq -c \ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta(1 - \mathcal{E}_{i,t})). \end{aligned}$$

Let $\beta = e^{-\eta}$. Since the function $z \mapsto \beta^z$ is convex, for $0 \le z \le 1$ we have

$$\beta^{(1-z)a+zb} \le (1-z)\beta^a + z\beta^b$$

for all $a, b \in \mathbb{R}$. Choosing $a = 0$ and $b = 1$ gives us

$$\beta^z \le 1 - z + z\beta.$$

We get

$$\begin{aligned} -c \ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta \mathcal{E}_{i,t}) &\ge -c \ln \sum_{i=1}^{n} v_{t,i}(1 - \mathcal{E}_{i,t} + \beta \mathcal{E}_{i,t}) \\ &= -c \ln(1 - (1-\beta) r_t) \end{aligned}$$

and

$$\begin{aligned} -c \ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta(1 - \mathcal{E}_{i,t})) &\ge -c \ln \sum_{i=1}^{n} v_{t,i}(1 - (1 - \mathcal{E}_{i,t}) + \beta(1 - \mathcal{E}_{i,t})) \\ &= -c \ln(1 - (1-\beta)(1 - r_t)) \end{aligned}$$

where $r_t = \sum_{i=1}^{n} v_{t,i} \mathcal{E}_{i,t}$.

Thus, it is sufficient to show for all $0 \leq r_t \leq 1$ that

$$\begin{aligned} \widehat{y} &\leq -c \ln(1-(1-\beta)r_t) \\ 1-\widehat{y} &\leq -c \ln(1-(1-\beta)(1-r_t)). \end{aligned}$$

We combine the conditions as

$$1 + c \ln(1-(1-\beta)(1-r_t)) \leq \widehat{y} \leq -c \ln(1-(1-\beta)r_t).$$

Obviously a suitable $\widehat{y}$ exists, if and only if

$$1 + c \ln(1-(1-\beta)(1-r_t)) \leq -c \ln(1-(1-\beta)r_t).$$

In this case one suitable choice would be

$$\widehat{y} = \frac{1}{2}\left(1 + c \ln(1-(1-\beta)(1-r_t)) - c \ln(1-(1-\beta)r_t)\right).$$

We could also go back to the original conditions for $\widehat{y}$ and choose

$$\widehat{y} = \frac{1}{2}\left(1 + c \ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta(1-\mathcal{E}_{i,t})) - c \ln \sum_{i=1}^{n} v_{t,i} \exp(-\eta \mathcal{E}_{i,t})\right).$$

Finally, to verify

$$1 + c \ln(1 - (1-\beta)(1-r_t)) \leq -c \ln(1 - (1-\beta) r_t),$$

first use Jensen's inequality and the concavity of ln to see that

$$\begin{aligned} 2c \cdot \left(\frac{1}{2} \ln(1 - (1-\beta)(1-r_t)) + \frac{1}{2} \ln(1 - (1-\beta) r_t) \right) & \\ \leq \quad 2c \ln \left(\frac{1 - (1-\beta)(1-r_t) + 1 - (1-\beta) r_t}{2} \right) & \\ = \quad -2c \ln \frac{2}{1+\beta} & \end{aligned}$$

so with our choice

$$c = \left(2 \ln \frac{2}{1+\beta} \right)^{-1}$$

we get

$$1 + c \ln(1 - (1-\beta)(1-r_t)) + c \ln(1 - (1-\beta) r_t) \leq 1 - 2c \ln \frac{2}{1+\beta} = 0$$

so the condition is satisfied. □

The generalisation of the Weighted Majority algorithm is the following.

Algorithm 2.6 (Aggregating Algorithm (AA)):

Given: loss function L, learning rate $\eta > 0$, factor $c > 0$

Initialise $w_{1,i} = 1$ for all i.
Repeat for $t = 1, \ldots, T$:

1. Receive all expert predictions $\mathcal{E}_{i,t}$.
2. Let $W_t = \sum_{i=1}^{N} w_{t,i}$ and, for $y \in \{0, 1\}$, $W_t(y) = \sum_{i=1}^{N} w_{t,i} \exp(-\eta L(y, \mathcal{E}_{i,t}))$.
3. Predict with any $\widehat{y}_t$ satisfying $L(y, \widehat{y}_t) \leq c \ln(W_t / W_{t+1}(y))$ both for $y = 0$ and $y = 1$. (If no such $\widehat{y}_t$ exists, the algorithm **fails**.)
4. Receive y_t and for all i let $w_{t+1,i} = w_{t,i} \exp(-\eta L(y_t, \mathcal{E}_{i,t}))$.

Remark: Previous lemma shows precisely that with $L = L_{\text{abs}}$ and $c = \left(2 \ln(2/(1 + \mathrm{e}^{-\eta}))\right)^{-1}$, the Aggregating Algorithm never fails. Other such parameter combinations include $L = L_{\log}$ and $c = \eta = 1$, and $L = L_{\text{sq}}$, $\eta = 2$ and $c = 1/2$, but we will not show the proofs here.

Theorem 2.7: If the Aggregating Algorithm never fails, it satisfies for any trial sequence S the loss bound

$$L(S, \mathsf{AA}) \le c\eta \min_{1 \le i \le N} L(S, \mathcal{E}_i) + c \ln n,$$

where $L(S, \mathcal{E}_i) = \sum_{t=1}^{T} L(y_t, \mathcal{E}_{i,t})$.

Proof: Basically the same as for the Weighted Majority. By assumption we have

$$L(y_t, \widehat{y}_t) \le P_t - P_{t+1}$$

for all t, where $P_t = c \ln W_t$. Summing over t, we get

$$L(S, \mathsf{AA}) \le P_1 - P_{T+1}.$$

We have $P_1 = c \ln n$ and

$$\begin{aligned} -P_{T+1} &= -c \ln W_{T+1} \\ &\le -c \ln w_{t+1,i} \\ &= -c \ln \exp\left(-\eta \sum_{t=1}^{T} L(y_t, \mathcal{E}_{i,t})\right) \\ &= c\eta L(S, \mathcal{E}_i) \end{aligned}$$

for any $1 \le i \le n$. $\square$

Corollary 2.8: Let $\eta > 0$ be arbitrary and $\beta = e^{-\eta}$. With $c = (2\ln(2/(1+\beta)))^{-1}$, the Aggregating Algorithm for absolute loss achieves

$$L_{\text{abs}}(S, \text{AA}) \leq \frac{\eta}{2\ln(2/(1+\beta))} \min_{1 \leq i \leq n} L_{\text{abs}}(S, \mathcal{E}_i) + \frac{1}{2\ln(2/(1+\beta))} \ln n.$$

Remark Comparing this to the Weighted Majority bound

$$L_{0-1}(S, \text{WM}) \leq \frac{\eta}{\ln(2/(1+\beta))} \min_{1 \leq i \leq n} L_{0-1}(\mathcal{E}_i) + \frac{1}{\ln(2/(1+\beta))} \ln n$$

and remembering the interpretation of absolute loss as expected number of mistakes, we see that allowing randomised (soft) predictions saves a factor 2 in the loss bound.

Tuning the learning rate

The bound of the previous corollary contains the learning rate $\eta = -\ln \beta$ as a free parameter. The choice of such parameters often has a large impact on the performance of the algorithm, both in practice and in theoretical analysis. In batch learning, the most common approach to parameter tuning is cross-validation. (See *Introduction to Machine Learning*.) In online learning, it is not clear what could replace cross-validation, and anyway we also want to have some insight into the theory, too.

Let us denote the loss of the best expert by $L_* = \min_i L_{0-1}(S, \mathcal{E}_i)$ The basic idea is that if L_* and $\ln n$ are known, the bound from previous corollary is simply a function of η. We then choose η so as to minimise the bound (or some more convenient approximation of it).

In reality things are not so simple, since η must be fixed at the beginning, when one would not usually know L_*.

The first approach to this problem is to assume that at least some upper bound $K \geq L_*$ is available in advance.

Theorem 2.9: Define $h(z) = 1 + 2\sqrt{z} + z/\ln 2$. If K is such that $L_* \leq K$ and $\eta = \ln h((\ln n)/K)$, then

$$L_{\text{abs}}(S, \text{AA}) \leq L_* + \sqrt{K \ln n} + \frac{\ln n}{2 \ln 2}.$$

(Proof: Plug in value of η to previous corollary and crank; details omitted.)

Of course if we knew L_* in the beginning, we would get the best bound by choosing $K = L_*$.

If we know T in advance, which is a much more reasonable assumption, we could choose $K = T$.

Further, if we assume that H is closed with respect to complementation, i.e. for all $h \in H$ there is $\bar{h} \in H$ such that $\bar{h}(x) = 1 - h(x)$, we could choose $K = T/2$. If H is originally not closed with respect to complementation, we can always make it so by at most doubling $|H| = n$. This only increases the term $\ln n$ to $\ln n + \ln 2$ in the bound.

There are still problems with using the upper bound $L_* \leq T/2$:

- This might still be a very loose estimate.
- We might not know even T in advance.

Before briefly discussing these issues, consider first the bound we get with $K = T/2$. We rewrite it as

$$L_{\mathsf{abs}}(S, \mathsf{AA}) - L_* \leq \sqrt{(T/2) \ln n} + \frac{\ln n}{2 \ln 2}.$$

The quantity $L_{\mathsf{abs}}(S, \mathsf{AA}) - L_*$ is called the regret of the Aggregating Algorithm. Intuitively, this is how much we afterwards regret that we did not know the best expert in advance. Thus, the algorithm has regret $O(T^{1/2})$ (in this context we consider n a constant). Equivalently,

$$\frac{L_{\mathsf{abs}}(S, \mathsf{AA})}{T} - \frac{L_*}{T} \leq \sqrt{\frac{2 \ln n}{T}} + \frac{\ln n}{2T \ln 2},$$

so the regret per time unit goes to zero at rate $O(T^{-1/2})$. This suggests that in some sense the algorithm is converging towards the best expert.

The interpretation of the sublinear regret

$$L_{\text{abs}}(S, \text{AA}) - L_* = O\left(T^{1/2}\right)$$

as some kind of convergence has one major flaw. The bound requires that η is chosen suitably as a function of T and fixed in the beginning. Therefore it is **not** the case that for any single instance of the Aggregating Algorithm the regret per time $(L_{\text{abs}}(S, \text{AA}) - L_*)/T$ would go to zero as T increases. Rather, we must consider a different instance of the AA, with different η, for each value T.

The two major methods for dealing with this are

- the doubling trick
- self-confident tuning.

We explain only the basic ideas. For details (which are complicated) see Auer, Cesa-Bianchi and Gentile, *Adaptive and self-confident learning algorithms.*

The doubling trick

Suppose we do not know even T in advance. We can dynamically adjust η as follows.

1. Let $B \leftarrow B_0$ for some suitable initial value $B_0 > 0$.
2. Choose η according to Theorem 2.9 with $K = B$. Let M be the resulting mistake bound.
3. Run the algorithm until one of the following happens:
   - Mistake count exceeds M. Set $B \leftarrow 2B$ ("doubling"). Reset the weights of the algorithm, and the mistake count. Go to Step 2.
   - We finish without exceeding M mistakes, and are happy.

Thus, the run of the algorithm consists of stages. During Stage i, we act as if we knew $L_* \leq 2^i B_0$.

Since the estimate B increases, the value η decreases as we go from one stage to the next one. Therefore it really is necessary to reset the weights, otherwise we might never recover from the effect of too large learning rate in some early stage.

By carefully choosing B and summing up all the stages we get for this algorithm

$$L_{0-1}(S, A) \leq L_* + a\sqrt{L_* \ln n} + b \ln n$$

for some $a, b > 0$. Very roughly, the idea is the following:

- In the last stage, the estimate B is at most double the loss of the best expert (for that stage).
- Hence, the regret for the last stage is at most some constant times the optimum.
- Because of how doubling works, total regret in all earlier stages is at most the same as (the upper bound for) the last stage.

Doubling is probably **not** a good strategy in practice, since the weights must be reset at the beginning of each stage, losing anything we learned previously.

Nevertheless, one should be aware of this concept, since it is often mentioned in the theory literature (also in the context of non-machine learning algorithms).

Self-confident algorithms

The learning rate given in Theorem 2.9 was

$$\eta = \ln\left(1 + 2\sqrt{z} + \frac{z}{\ln 2}\right)$$

where $z = (\ln n)/K$ and $K \geq L_*$. The interesting case is large L_*, i.e. small z. Since $\eta = 2\sqrt{z} + O(z)$, we approximate it $\eta = d\sqrt{(\ln n)/K}$ for some $d > 0$.

The idea of self-confident algorithms is that instead of trying to bound L_* in the beginning, we allow η to change and use increasingly accurate estimates for L_*. Hence, at time t we use learning rate

$$\eta_t = d\sqrt{\frac{\ln n}{L_*^{\leq t}}}$$

where $d > 0$ is a suitable constant and

$$L_*^{\leq t} = \min_{1 \leq i \leq n} \sum_{j=1}^{t} L_{0-1}(y_j, \mathcal{E}_{i,j})$$

is the loss of the best expert up to time t which can easily be observed.

Intuitively, we are always using slightly wrong learning rate, but the effect of this on any trial can be bounded so that we again can bound the total loss as

$$L_{0-1}(S, A) \leq L_* + a\sqrt{L_* \ln n} + b \ln n$$

for some $a, b > 0$.

This technique can be used also for example in linear classification algorithms (of which we will soon see some examples). There $L_*^{\leq t}$ is not as easy to evaluate exactly as in the expert setting, so we use estimates. Possible estimates include

- Ct where C is an upper bound for loss in single trial
- the loss of the algorithm $\sum_{j=1}^{t} L_{0-1}(y_j, \widehat{y}_j)$.

The first estimate leads to $\eta = C'/\sqrt{t}$ for some C'. This is simple and fairly popular in practice.

The second estimate is more theoretical, but it gives the name of this approach: the algorithm is confident that its own loss is pretty close to L_*.

Online learning of linear classifiers

Let $X = \mathbb{R}^d$ and $Y = \{-1, 1\}$. We use the notation $f(\cdot; \boldsymbol{w}, b)$ for the linear classifier

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} - b) = \begin{cases} -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} < b \\ 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq b. \end{cases}$$

We use notation $f(\cdot; \boldsymbol{w})$ for the important special case where b is fixed to zero.

The normalised margin of example $(\boldsymbol{x}, y)$ with respect to $f(\cdot; \boldsymbol{w}, b)$ is

$$\frac{y(\boldsymbol{w} \cdot \boldsymbol{x} - b)}{\|\boldsymbol{w}\|_2}$$

where $\|\boldsymbol{w}\|_2 = \left(\sum_{i=1}^{d} |w_i|^2\right)^{1/2}$ is the Euclidean norm.

The margin is positive (or zero) if $f(\boldsymbol{x}; \boldsymbol{w}, b) = y$ and negative otherwise. The absolute value of the margin is the geometric distance of $\boldsymbol{x}$ from the hyperplane $\{\boldsymbol{z} \mid \boldsymbol{w} \cdot \boldsymbol{z} = b\}$, i.e. the decision boundary.

We call $y(\boldsymbol{w} \cdot \boldsymbol{x} - b)$ the unnormalised margin.

We sometime include the threshold value b as part of the weight vector $\boldsymbol{w}$, so we do not need to consider it separately in the learning algorithm.

- Suppose we have an algorithm A that learns linear classifiers of the form $f(\cdot; \boldsymbol{w})$ (with threshold fixed to zero).
- Based on a sample $s = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \in (\mathbb{R}^d \times \{-1, 1\})^m$, we wish to learn a classifier $f(\cdot; \boldsymbol{w}, b)$ where the threshold b may be non-zero.
- For $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$, let $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_d, 1) \in \mathbb{R}^{d+1}$, and similarly $\tilde{S} = ((\tilde{\boldsymbol{x}}_1, y_1), \ldots, (\tilde{\boldsymbol{x}}_m, y_m)) \in (\mathbb{R}^{d+1} \times \{-1, 1\})^m$.
- We apply the algorithm A with sample $\tilde{S}$, obtaining a weight vector $\tilde{\boldsymbol{w}} = (\tilde{w}_1, \ldots, \tilde{w}_n, \tilde{w}_{d+1}) \in \mathbb{R}^{d+1}$.
- The d dimensional classifier $f(\cdot; \boldsymbol{w}, b)$ is obtained as $\boldsymbol{w} = (\tilde{w}_1, \ldots, \tilde{w}_d)$ ja $b = -\tilde{w}_{d+1}$.

Now $f(\boldsymbol{x}; \boldsymbol{w}, b) = f(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{w}})$.

The Perceptron Algorithm

This is the most basic algorithm for linear classification. We fix threshold $b = 0$; to get the more general case, use the reduction on previous page.

Algorithm 2.10 (The Perceptron Algorithm):

Initialise $\boldsymbol{w}_1 \leftarrow \mathbf{0}$.
For $t = 1, \ldots, T$ do the following:

1. Get the instance $\boldsymbol{x}_t \in \mathbb{R}^d$.
2. Predict $\widehat{y}_t = \text{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x}_t) \in \{-1, 1\}$.
3. Get the correct answer $y_t \in \{-1, 1\}$.
   Let $\sigma_t = 1$ if $y_t \neq \widehat{y}_t$ and $\sigma_t = 0$ if $y_t = \widehat{y}_t$.
4. Update $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \sigma_t y_t \boldsymbol{x}_t$.

In other words, if no mistake is made, $\sigma_t = 0$ and the weight vector remains unchanged.

Theorem 2.11 (Perceptron Convergence Theorem): Let $B > 0$ be such that $\|\boldsymbol{x}_t\|_2 \leq B$ for all t. Assume that for some $\boldsymbol{u} \in \mathbb{R}^d$ the classifier $f(\cdot; \boldsymbol{u})$ has normalised margin at least $\gamma > 0$ on all examples $(\boldsymbol{x}_t, y_t)$ (i.e., the sample is linearly separable with margin γ). Then the Perceptron Algorithm makes at most

$$\sum_{t=1}^{T} \sigma_t \leq \frac{B^2}{\gamma^2}$$

mistakes.

Remark: The mistake bound does not explicitly depend on T or d. They could even be infinite (if the notations and definitions are generalised suitably; we omit the details).

Remark: We could equivalently formulate the separability assumption as

- some $\boldsymbol{u}$ with $\|\boldsymbol{u}\|_2 = 1$ satisfies $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \gamma$ for all t, or
- some $\boldsymbol{u}$ with $\|\boldsymbol{u}\|_2 \leq 1/\gamma$ satisfies $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq 1$ for all t.

Before the proof, consider briefly the implications.

Suppose we have a sample of m examples

$$s = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \in (\mathbb{R}^d \times \{-1, 1\})^m.$$

For $k = 1, 2, 3, \ldots$, let s_k be the sample of mk examples obtained by taking k copies of s and putting them one after another.

Running the Perceptron on s_k, if there ever are m consecutive time steps during which no mistake was made, the algorithm has learned to predict all the examples correctly. Hence there will be no more updates, and the algorithm has converged.

Suppose some $\boldsymbol{u}$ has margin at least $\alpha > 0$ on s. The margin is of course the same for s_k, for all k. If

$$r = \frac{\max_t \|\boldsymbol{x}_t\|_2^2}{\alpha^2},$$

then for any k the Perceptron Algorithm makes at most r mistakes on s_k.

In particular, if we take $k = r + 1$, we know that in s_k there must be some sequence of m consecutive examples during which there were no mistakes.

Hence, after repeating the sample at most $r + 1$ times the algorithm has converged to a consistent hypothesis ($\boldsymbol{w}_t$ such that $y_j \boldsymbol{w}_t \cdot \boldsymbol{x}_j > 0$ for all j).

The previous remark is related to applying online algorithms to the statistical learning setting. We shall return to this later in more detail. For now, just some brief remarks:

- Linear classifiers in high-dimensional spaces are a very rich concept class. In modern machine learning applications it is common to have $n \gg m$. (This is often the result of using feature maps, of which more soon.) In this case running the Perceptron through the sample sufficiently many times **will** produce a consistent hypothesis, **no matter what** the actual data is (barring some pathological special cases). This is an example of overfitting.

- Various methods exist to avoid overfitting. They include early stopping and keeping the weights "small" by regularisation.

- Algorithmically, if the margin is small then the problem of finding a consistent linear classifier may be more efficiently solved by linear programming.

Proof of the Perceptron Convergence Theorem: Without loss of generality we can take $\|\boldsymbol{u}\|_2 = 1$. Then $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \gamma$ for all t.

The idea is to show that $\boldsymbol{w}_t$ converges towards $c\boldsymbol{u}$ where $c > 0$ is a suitable constant. (Recall that $\boldsymbol{u}$ and $c\boldsymbol{u}$ for $c > 0$ define the same classifier.)

We define a potential function

$$P_t = \frac{1}{2} \|c\boldsymbol{u} - \boldsymbol{w}_t\|_2^2 ,$$

where $c > 0$ is to be fixed later. Initially $P_1 = c^2 \|\boldsymbol{u}\|_2^2 / 2 = c^2/2$. Always $P_t \geq 0$.

Next we lower bound the drop of potential at time t as

$$P_t - P_{t+1} \geq \left(c\gamma - \frac{B^2}{2} \right) \sigma_t .$$

For $\sigma_t = 0$ the claim is clear. Assume $\sigma_t = 1$, so $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq 0$.

By simply writing the squared norms as dot products it is easy to verify

$$\frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}\|_2^2 - \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}'\|_2^2 = (\boldsymbol{w}' - \boldsymbol{w}) \cdot (\boldsymbol{u} - \boldsymbol{w}) - \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}'\|_2^2 .$$

for all $\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{w}' \in \mathbb{R}^d$. (Notice that since the dot product on the right-hand side can be negative, this shows that the squared Euclidean distance does not satisfy the triangle inequality.)

By plugging in $\boldsymbol{u} \leftarrow c\boldsymbol{u}$, $\boldsymbol{w} \leftarrow \boldsymbol{w}_t$ and $\boldsymbol{w}' \leftarrow \boldsymbol{w}_{t+1}$, and noticing $\boldsymbol{w}_{t+1} - \boldsymbol{w}_t = y_t \boldsymbol{x}_t$, we get

$$P_t - P_{t+1} = y_t \boldsymbol{x}_t \cdot (c\boldsymbol{u} - \boldsymbol{w}_t) - \frac{1}{2}\|\boldsymbol{x}_t\|_2^2 .$$

Since $\|\boldsymbol{x}_t\|_2 \leq X$, $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \gamma$ and $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq 0$, we get

$$P_t - P_{t+1} \geq c\gamma - \frac{B^2}{2}.$$

By summing over t we get

$$\begin{aligned}\frac{c^2}{2} &\geq P_1 - P_{T+1} \\ &\geq \sum_{t=1}^{T} (P_t - P_{t+1}) \\ &\geq \left(c\gamma - \frac{X^2}{2}\right) \sum_{t=1}^{T} \sigma_t\end{aligned}$$

which for $c \geq X^2/(2\gamma)$ becomes

$$\sum_{t=1}^{T} \sigma_t \leq \frac{c^2}{2c\gamma - X^2}.$$

We get the desired bound by choosing $c = X^2/\gamma$. (A straightforward differentiation shows that this choice of c maximises

$$\frac{2c\gamma - X^2}{c^2}$$

and thus gives the best bound.) □

To get some geometrical intuition, denote by $\varphi(\boldsymbol{u}, \boldsymbol{x})$ the angle between vectors $\boldsymbol{u}$ and $\boldsymbol{x}$. That is,

$$\cos\varphi(\boldsymbol{u}, \boldsymbol{x}) = \frac{\boldsymbol{u} \cdot \boldsymbol{x}}{\|\boldsymbol{u}\|_2 \, \|\boldsymbol{x}\|_2}.$$

Do the standard simplification trick of replacing $(\boldsymbol{x}_t, y_t)$ by $(\tilde{\boldsymbol{x}}_t, 1)$ where $\tilde{\boldsymbol{x}}_t = y_t \boldsymbol{x}_t$. The condition $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \gamma$ then becomes $\boldsymbol{u} \cdot \tilde{\boldsymbol{x}}_t \geq \gamma$.

For simplicity, consider the special case $\|\boldsymbol{x}_t\|_2 = 1$ for all t. The condition becomes $\cos\varphi(\boldsymbol{u}, \tilde{\boldsymbol{x}}_t) \geq \gamma$. Thus for all $\tilde{\boldsymbol{x}}_t$ we have

$$\varphi(\boldsymbol{u}, \tilde{\boldsymbol{x}}_t) \leq \arccos\gamma = \frac{\pi}{2} - \theta$$

for some constant $\theta > 0$. All the $\tilde{\boldsymbol{x}}_t$ are in a certain cone opening around the vector $\boldsymbol{u}$ in angle $\pi/2 - \theta$.

Suppose we now simply want to find any $\boldsymbol{w}$ such that $\boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_t > 0$ for all t. Thus any positive margin is acceptable. Then we can pick any $\boldsymbol{w}$ in the interior of the cone opening in an angle θ around $\boldsymbol{u}$.

The idea of the Perceptron Convercence Theorem is that every mistake twists $\boldsymbol{w}_t$ towards $\boldsymbol{u}$ by a fixed amount, and after a finite number of mistakes $\boldsymbol{w}_t$ is in the cone and no further mistakes occur.

Example 2.12: Let g be a k-literal conjunction over n variables. That is,

$$g(\boldsymbol{x}) = \tilde{x}_{i_1} \wedge \ldots \wedge \tilde{x}_{i_k},$$

where each literal $\tilde{x}_j$ is either x_j or $\overline{x_j}$.

For $i = 1, \ldots, d$, define $u_j = 1$ if g contains the literal x_j and $u_{i_j} = -1$ if g contains the literal $\overline{x_j}$. Further, introduce an extra variable x_{d+1} that is always set to 1, and let $u_{d+1} = -k + 1$. For all other i we set $u_i = 0$. Then

$$g(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{d+1} u_i x_i\right)$$

and this linear classifier has unnormalised margin 1. Since $\|\boldsymbol{u}\|_2 = \sqrt{k \cdot 1^2 + (d-k) \cdot 0^2 + (k-1)^2}$, the normalised margin is $(2k^2 - 2k + 1)^{-1/2}$.

Assume now that the sequence $((\boldsymbol{x}_t, y_t)) \in (\{-1, 1\}^d \times \{-1, 1\})^T$ is such that $y_t = g(\boldsymbol{x}_t)$ for all t. Then $\|\boldsymbol{x}_t\|_2^2 = d + 1$ for all t. Plugging this and the margin to the Perceptron Convergence Theorem, we get

$$\sum_{i=1}^{T} \sigma_t \leq d(2k^2 - 2k + 1).$$

In practical applications, we of course never know that the target is a k-literal conjunction. Nevertheless bounds like this do give useful intuition about what affects the performance of the algorithm.

Notice in particular that the mistake bound is linear in d even if the target is extremely simple, say $k = 1$. By experimenting, we can see that this is how the algorithm actually behaves, not just some loose upper bound.
□(Example)

Feature mappings

Consider learning classifiers over instance base X. In principle a feature map is any function $\boldsymbol{\psi}\colon X \to \mathbb{R}^r$, for some r. The r components $\psi_i(x)$ are called features of x, and $\mathbb{R}^r$ is the feature space.

Given a feature map ψ and a linear prediction algorithm A (e.g. the Perceptron), we can learn classifiers $X \to \{-1, 1\}$ by doing the following at time t:

1. Get input x_t.
2. Give $\boldsymbol{\psi}(x_t)$ as input to A.
3. Get prediction $\widehat{y}_t$ from A and give it as your prediction.
4. Get correct answer y_t and give to A.

Thus, the linear algorithm A sees a transformed sequence $((\boldsymbol{\psi}(x_t), y_t)) \in (\mathbb{R}^r \times \{-1, 1\})^T$. Ideally, we hope to choose $\boldsymbol{\psi}$ such that this sequence is linearly separable with large margin.

The original instance base X need not be a subset of $\mathbb{R}^n$ for any n. However, having $X = \mathbb{R}^n$ with $n \ll r$ is an important special case.

Example 2.13: Consider learning k-clause l-CNF formulas over n variables. Such formulas are of the form $\phi_1 \wedge \ldots \wedge \phi_k$, where each clause ϕ_i is a disjunction of l literals (variables or negated variables). We choose as features all disjunctions of at most l literals. Thus, for $n = 4$ and $l = 2$ we have

$$\begin{aligned}\boldsymbol{\psi}(\boldsymbol{x}) \quad = \quad & (\mathbf{false}, \mathbf{true}, x_1, x_2, x_3, x_4, \overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \\ & x_1 \vee x_2, x_1 \vee x_3, x_1 \vee x_4, x_1 \vee \overline{x_2}, x_1 \vee \overline{x_3}, x_1 \vee \overline{x_4}, \\ & \overline{x_1} \vee x_2, \overline{x_1} \vee x_3, \overline{x_1} \vee x_4, \overline{x_1} \vee \overline{x_2}, \overline{x_1} \vee \overline{x_3}, \overline{x_1} \vee \overline{x_4}, \\ & x_2 \vee x_3, x_2 \vee x_4, x_2 \vee \overline{x_3}, x_2 \vee \overline{x_4}, \overline{x_2} \vee x_3, \overline{x_2} \vee x_4, \overline{x_2} \vee \overline{x_3}, \overline{x_2} \vee \overline{x_4}, \\ & x_3 \vee x_4, x_3 \vee \overline{x_4}, \overline{x_3} \vee x_4, \overline{x_3} \vee \overline{x_4}).\end{aligned}$$

(The ordering of the features is of course unimportant.) Here $r = 34$. In general we can roughly estimate estimate $r \leq (2n+1)^l$.

Assume now $y_t = g(\boldsymbol{x}_t)$ where $\boldsymbol{x}_t \in \{-1, 1\}^n$ and g is a k-clause l-CNF formula. Then $y_t = \tilde{g}(\boldsymbol{\psi}(\boldsymbol{x}_t))$ where $\tilde{g}$ is a k-literal conjunction. Based on the previous example, we know that using the Perceptron Algorithm with this feature map will make at most

$$r(2k^2 - 2k + 1) \leq (2n+1)^l(2k^2 - 2k + 1)$$

mistakes. □

Example 2.14: A monomial over (real-valued) variables $x_1, \ldots, x_n$ is a product $x_1^{k_1} \ldots x_n^{k_n}$. The degree of the monomial is $k_1 + \cdots + k_n$. Let r be the number of degree k monomials over n variables, and let ψ_i, $i = 1, \ldots, r$, be these monomials. A degree k polynomial over $\boldsymbol{x}$ is a sum

$$\sum_{i=1}^{r} a_i \psi_i(\boldsymbol{x})$$

where $a_i \in \mathbb{R}$. A degree k polynomial classifier is a classifier g that can be represented as $g(\boldsymbol{x}) = \text{sign}(p(\boldsymbol{x}))$ for some degree k polynomial p. The feature map $\boldsymbol{\psi}$ reduces learning degree k polynomial classifiers to learning linear classifiers in r dimensions.

It can be shown that $r = \binom{n+k}{k}$. As a useful estimate, we have

$$\left(\frac{n}{k} + 1\right)^k \leq \binom{n+k}{k} \leq \left(\frac{\mathrm{e}n}{k} + \mathrm{e}\right)^k.$$

As a practical application, consider classifying images given as pixel vectors. Using a linear classifier would allow us to give weights to individual pixels, which is often not very informative. Using degree 2 polynomials allows us to consider correlations between pair of pixels. In general, degree k polynomials consider interactions between groups of k pixels. □

The Kernel Trick

This is an important technique that makes feature mapping particularly attractive for linear learning.

The feature spaces $\mathbb{R}^r$ of interesting feature mappings tend to have **very** large r. This is a computational problem, since it looks like we would need to do a lot of manipulation of r-dimensional vectors.

A kernel function for feature map $\boldsymbol{\psi}$ is a function $k\colon X^2 \to \mathbb{R}$ such that $k(x,z) = \boldsymbol{\psi}(x) \cdot \boldsymbol{\psi}(z)$ for all $x, z \in X$. Here $\cdot$ is the dot product in the r-dimensional feature space. Often the kernel is much simpler to compute than the actual feature map (examples follow).

To apply to the Perceptron, notice that with feature map $\boldsymbol{\psi}$ we have $\boldsymbol{w}_t = \sum_{j=1}^{t-1} \sigma_j y_j \boldsymbol{\psi}(x_j)$. Therefore

$$\boldsymbol{w}_t \cdot \boldsymbol{\psi}(\boldsymbol{x}_t) = \sum_{j=1}^{t-1} \sigma_j y_j \boldsymbol{\psi}(x_j) \cdot \boldsymbol{\psi}(x_t) = \sum_{j=1}^{t-1} \sigma_j y_j k(x_j, x_t).$$

We get the following algorithm:

Algorithm 2.15 (Kernelised Perceptron):

For $t = 1, \ldots, T$ do the following:

1. Get the instance $x_t \in X$.
2. Let $p_t = \sum_{j=1}^{t-1} \sigma_j y_j k(x_j, x_t)$. Predict $\widehat{y}_t = \mathrm{sign}(p_t)$.
3. Get the correct answer $y_t \in \{-1, 1\}$.
4. If $y_t p_t \leq 0$, set $\sigma_t = 1$ and store y_t and x_t.
   Otherwise $\sigma_t = 0$ and x_t can be discarded.

Instead of storing (and manipulating) an explicit feature space weight vector $\boldsymbol{w}_t$, which would have r components, we store $O(T)$ instances x_t and coefficients σ_t. For large enough T, this can be a computational problem, too.

Theorem 2.16: Let $\boldsymbol{\psi}\colon X \to \mathbb{R}^r$ be a feature map with kernel k, and write $B^2 = \max_t k(x_t, x_t)$. If there is a vector $\boldsymbol{u} \in \mathbb{R}^r$ such that $\|\boldsymbol{u}\|_2 = 1$ and $y_t \boldsymbol{u} \cdot \boldsymbol{\psi}(x_t) \geq \gamma > 0$ for all t, then the Kernelised Perceptron makes at most

$$\frac{B^2}{\gamma^2}$$

mistakes on sequence $((x_t, y_t))$.

Proof: This is a direct corollary the the original Perceptron Convergence Theorem applied to the sequence $((\boldsymbol{\psi}(x_t), y_t))$. Notice that $\|\boldsymbol{\psi}(x)\|_2^2 = \boldsymbol{\psi}(x) \cdot \boldsymbol{\psi}(x) = k(x, x)$. □

The vector $\boldsymbol{u}$ in the theorem can be any vector in the feature space. We do *not* require $\boldsymbol{u} = \boldsymbol{\psi}(z)$ for some $z \in X$.

Next we consider some basic examples of kernels.

Example 2.17 (Monomial kernel): Consider $X \subseteq \mathbb{R}^n$ and the kernel

$$k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x} \cdot \boldsymbol{z})^q \,.$$

This corresponds to an r dimensional feature space, where $r = \binom{n+q-1}{q}$ is the number of monomials over n variables with degree exactly q. (For constant q, we have $r = \Theta(n^q)$.)

To see this, denote the monomials by $\psi_1(x_1, \ldots, x_n), \ldots, \psi_r(x_1, \ldots, x_n)$. By simply multiplying out, we get

$$\begin{aligned}
(\boldsymbol{x} \cdot \boldsymbol{z})^q &= (x_1 z_1 + \cdots + x_n z_n)^q \\
&= \sum_{j=1}^{r} c_j \psi_j(x_1 z_1, \ldots, x_n z_n) \\
&= \sum_{j=1}^{r} c_j \psi_j(x_1, \ldots, x_n) \psi_j(z_1, \ldots, z_n)
\end{aligned}$$

for some constants c_j (that depend on n and q). For example, in case $n = q = 2$ we get

$$(x_1 z_1 + x_2 z_2)^2 = (x_1 z_1)^2 + 2 x_1 z_1 x_2 z_2 + (x_2 z_2)^2.$$

If we define $\tilde{\psi}_j(\boldsymbol{x}) = c_j^{1/2} \psi_j(\boldsymbol{x})$, we see that $k(\boldsymbol{x}, \boldsymbol{z}) = \tilde{\boldsymbol{\psi}}(\boldsymbol{x}) \cdot \tilde{\boldsymbol{\psi}}(\boldsymbol{z})$.

Hence, the kernel k corresponds to features which are the monomials ψ_j with some weights $c_j^{1/2}$. We have reduced computing the dot product in $\Theta(n^q)$ dimensional feature space to computing the dot product in n dimensional space and taking a power which does not depend on n. $\square$

Example 2.18 (Polynomial kernel): The degree q polynomial kernel, again for $X \subseteq \mathbb{R}^n$, is given by

$$k_q(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x} \cdot \boldsymbol{z} + c)^q$$

where $c > 0$ is some suitable constant. It can be shown that the dimension of the feature space is $r = \binom{n+q}{q}$ and each feature is one of the r monomials of degree at most q multiplied by some constant.

The values of these constants can be determined by writing

$$k_q(\boldsymbol{x}, \boldsymbol{z}) = \sum_{j=0}^{q} \binom{q}{j} c^{q-j} (\boldsymbol{x} \cdot \boldsymbol{z})^j$$

and rewriting $(\boldsymbol{x} \cdot \boldsymbol{z})^j$ as in previous example. For practical purposes this is not really interesting:

> We do not really care about the details of the feature map, since we never need to compute it and the kernel tells us all we need to know.

The expansion above does indicate that the larger c, the less weight is given to high-degree monomials. □

Example 2.19 (All subsets kernel): Again $X \subseteq \mathbb{R}^n$. We take as features the functions ψ_A for all $A \subseteq \{1, \ldots, n\}$ where

$$\psi_A(\boldsymbol{x}) = \prod_{i \in A} x_i.$$

In other words, we have all monomials where each individual variable may have degree at most 1. There are 2^n such monomials, and the kernel can be written as

$$k(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{n} (1 + x_i z_i).$$

This has an interesting application to Boolean functions. We encode **true** as 1 and **false** as 0 (instead of the ± 1 encoding we have been using). Then for $\boldsymbol{x} \in \{0, 1\}^n$, the features ψ_A are exactly the monotone conjunctions over n variables. Further, we can include non-monotone ones by using

$$k'(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{n} (1 + x_i z_i)(1 + (1 - x_i)(1 - z_i))$$

or equivalently replacing $\boldsymbol{x} \in \{0, 1\}^n$ by $(x_1, \ldots, x_n, 1 - x_1, \ldots, 1 - x_n) \in \{0, 1\}^{2n}$.

Denote the feature map corresponding to k' by $\boldsymbol{\psi}'$. It has 4^n features that for $\boldsymbol{x} \in \{0,1\}^n$ become all the conjunctions, with **false** having several representations.

An arbitrary l-term DNF formula can be represented as $\text{sign}(\boldsymbol{u} \cdot \boldsymbol{\psi}'(\boldsymbol{x}))$ where $\|\boldsymbol{u}\|_2^2 = l$ and the unnormalised margin is $1/2$. Thus, it would seem that with this kernel the Perceptron could learn arbitrary Boolean formulas with $O(n)$ time per update.

Unfortunately, this will not work, since $\max_{\boldsymbol{x}} k'(\boldsymbol{x}, \boldsymbol{x}) = 2^n$ so the mistake bound becomes

$$\frac{2^n}{1/(2\sqrt{l})^2} = l2^{n+2}$$

which is larger than $|X| = 2^n$. Since this also means that the number of non-zero σ_t can be $\Omega(2^n)$, also the computational efficiency becomes questionable. □

Example 2.20 (Gaussian kernel): For $X \subseteq \mathbb{R}^n$, we define

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|_2^2}{2\sigma^2}\right)$$

where $\sigma > 0$ is a suitably chosen parameter. (The "suitable" values depend on application and may not be trivial to find.) This is also known as the radial basis function (RBF) kernel.

In this case the feature space is actually not $\mathbb{R}^n$ for any finite n but an infinite dimensional Hilbert space. Still, the computations can be done in finite time using kernels and the mistake bound analysis can be done using some inner product in some feature space. We ignore the formal details for now.

Since now $k(\boldsymbol{x}, \boldsymbol{x}) = 1$ for all $\boldsymbol{x} \in \mathbb{R}^n$, we can interpret the kernel as a similarity measure:

$$\begin{aligned}
\|\boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{z})\|_H^2 &= \langle \boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{z}), \boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{z}) \rangle_H \\
&= \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{x}) \rangle_H - 2 \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{z}) \rangle_H + \langle \boldsymbol{\psi}(\boldsymbol{z}), \boldsymbol{\psi}(\boldsymbol{z}) \rangle_H \\
&= k(\boldsymbol{x}, \boldsymbol{x}) - 2k(\boldsymbol{x}, \boldsymbol{z}) + k(\boldsymbol{z}, \boldsymbol{z}) \\
&= 2 - 2k(\boldsymbol{x}, \boldsymbol{z})
\end{aligned}$$

where $\|\cdot\|_H$ and $\langle \cdot, \cdot \rangle_H$ refer to the norm and inner product in the Hilbert space.

A large number of easy-to-compute kernels are known, and more are being developed. We shall not try to list them here, but it should be noted that kernels exist for a wide variety of instance classes X (trees, graphs, strings, text documents, . . .).

The recent interest in kernels among the machine learning community is largely due to the success of Support Vector Machines (SVMs) that are a statistical learning algorithm based on kernels and large margins. Kernels themselves are an old idea in mathematics, and their use in "machine learning" goes back to 1960's.

We return to SVMs later, and consider some theoretical issues we ignored here (such as, given $k(\cdot, \cdot)$, is it really a kernel for some feature map).

Despite its simplicity, the Kernelised Perceptron is surprisingly good. More sophisticated algorithms (such as SVM) can be more accurate, but often not by much, and they are computationally much more expensive.

Online learning with non-separable data

Often there is no $\boldsymbol{u}$ with positive margin on all the examples:

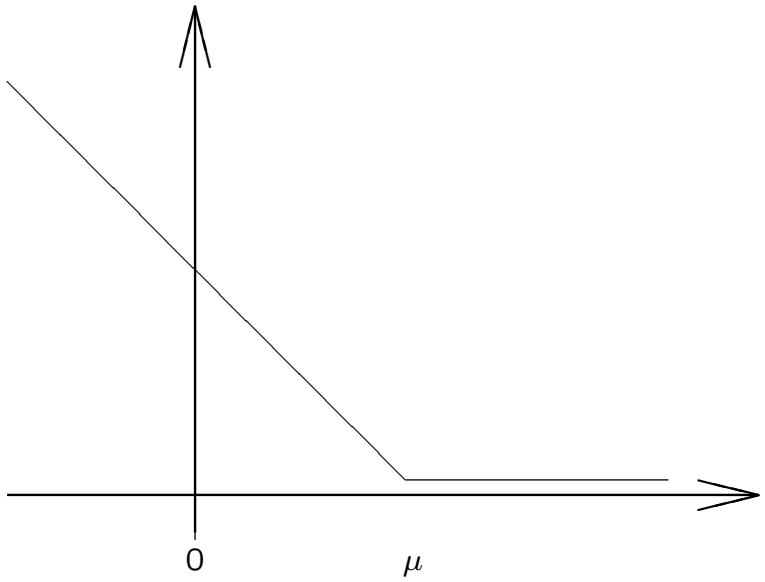
- the examples may contain errors
- the target classifier may be non-linear
- a target classifier may not exist at all (or the target is "probabilistic")

We generalise our bounds for this scenario by considering the hinge loss (or soft margin loss).

For any margin parameter $\mu \geq 0$ we define the hinge loss as

$$L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) = \max\{0, \mu - y\boldsymbol{u} \cdot \boldsymbol{x}\}.$$

If $y\boldsymbol{u} \cdot \boldsymbol{x} \leq 0$, then $L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) \geq \mu$.
If $y\boldsymbol{u} \cdot \boldsymbol{x} \geq \mu$, then $L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) = 0$.



Given some sequence $S = ((\boldsymbol{x}_t, y_t)) \in (\mathbb{R}^n \times \{-1, 1\})^T$, define for any linear prediction algorithm A and any fixed weight vector $\boldsymbol{u} \in \mathbb{R}^n$

$$L_\mu(S, A) = \sum_{t=1}^{T} L_\mu(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t) \quad \text{and} \quad L_\mu(S, \boldsymbol{u}) = \sum_{t=1}^{T} L_\mu(\boldsymbol{u}, \boldsymbol{x}_t, y_t).$$

Define similarly the discrete losses $L_{0-1}(A)$ and $L_{0-1}(\boldsymbol{u})$.

If $y\boldsymbol{u} \cdot \boldsymbol{x} \leq 0$, then $L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) \geq \mu$. Therefore

$$L_{0-1}(\boldsymbol{u}) \leq \frac{1}{\mu} L_\mu(\boldsymbol{u}).$$

Ideally, we would wish to have bounds of the form

$$L_{0-1}(S, A) \leq a L_{0-1}(S, \boldsymbol{u}) + b$$

for some constants a and b (that might depend on norms of $\boldsymbol{x}_t$ etc.). However, this seems very difficult. We settle for weaker bounds of the form

$$L_{0-1}(S, A) \leq \frac{a}{\mu} L_\mu(S, \boldsymbol{u}) + b.$$

We obtain the desired bound for Perceptron by analysing the following more general algorithm in terms of hinge loss and then considering some special cases.

Algorithm 2.21 (Marginalised Perceptron (MP)):
parameters: learning rate $\eta > 0$, margin parameter $\rho \geq 0$.

Initialise $\boldsymbol{w}_1 = \mathbf{0} \in \mathbb{R}^n$.
Repeat for $t = 1, \ldots, T$:
1. Get input $\boldsymbol{x}_t \in \mathbb{R}^n$.
2. Predict $\widehat{y}_t = \mathrm{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)$.
3. Get correct answer $y_t \in \{-1, 1\}$.
4. If $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq \rho$ then $\tilde{\sigma}_t = 1$.
   Otherwise $\tilde{\sigma}_t = 0$.
5. Let $\boldsymbol{w}'_{t+1} = \boldsymbol{w}_t + \eta \tilde{\sigma}_t y_t \boldsymbol{x}_t$.
6. If $\left\|\boldsymbol{w}'_{t+1}\right\|_2 > 1$ then $\boldsymbol{w}_{t+1} = \boldsymbol{w}'_{t+1} / \left\|\boldsymbol{w}'_{t+1}\right\|_2$.
   Otherwise $\boldsymbol{w}_{t+1} = \boldsymbol{w}'_{t+1}$.

We get the usual Perceptron by taking $\rho = 0$ and omitting the normalisation (step 6). The value η is then irrelevant.

If $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq \rho$, we say that a margin error occurred, which we denote by $\tilde{\sigma}_t = 1$. Step 6 forces the hypothesis to stay inside the unit ball.

Remarks on the marginalised Perceptron

The original Perceptron algorithm is conservative: it only updates its hypothesis when a mistake occurs. Choosing $\rho > 0$ makes the algorithm aggressive: updates may occur even when there was no actual error. This may lead to faster learning.

Assuming linearly separable data, the original Perceptron algorithm stops as soon as a consistent hypothesis is achieved. Using $\rho > 0$ also has the effect of forcing the algorithm to go on until margin ρ is achieved. We combine this with a normalisation step in order to get a normalised margin ρ.

We start by stating a bound on margin errors.

Theorem 2.22: Assume $\|\boldsymbol{u}\|_2 \leq 1$ and $\|\boldsymbol{x}_t\|_2 \leq X$ for all t. Then for any $0 \leq \rho < \mu$ and $0 < \eta < 2(\mu - \rho)/X^2$ the Marginalised Perceptron has

$$\sum_{t=1}^{T} \tilde{\sigma}_t \leq \frac{L_\mu(\boldsymbol{u})}{\mu - \rho - \eta X^2/2} + \frac{1}{2\eta(\mu - \rho - \eta X^2/2)}.$$

Intuitively μ is our idea of the margin with which the examples "should" be linearly separable. If the examples actually are not separable with margin μ, we want to consider the difference as "noise" that makes learning harder but ideally should not affect the end result.

Then ρ is the goal we set to our algorithm. We would wish to set ρ as close to μ as possible, but this will cost more margin errors.

After we have decided our goal by fixing ρ, the learning rate η can be tuned to get the fastest learning.

This becomes clearer by considering two extreme examples.

Example 2.23: Choose $\rho = 0$, so $\tilde{\sigma}_t = \sigma_t$. By straightforward calculus, we can find η such that the bound is minimised. The optimised bound is

$$\begin{aligned}\sum_{t=1}^{T} \sigma_t &\leq \frac{L_\mu(\boldsymbol{u}, S)}{\mu} + \frac{X^2}{2\mu^2} + \left(\frac{2L_\mu(\boldsymbol{u}, S)}{\mu} + \frac{X^2}{2\mu^2}\right)^{1/2} \left(\frac{X^2}{2\mu^2}\right)^{1/2} \\ &= \frac{L_\mu(\boldsymbol{u}, S)}{\mu} + O\left(\sqrt{L_\mu(\boldsymbol{u}, S)}\right)\end{aligned}$$

if we consider X and μ as constants. This gives a bound for how much the performance of the algorithm degrades, if the data are not linearly separable.

If we take $\rho = 0$ and omit the normalisation step, we get a bound for the usual Perceptron when the data is not linearly separable. The fact that our bound would seem to suggest a particular value η is an artefact of our proof technique. □

Example 2.24: Assume that $\boldsymbol{u}$ actually separates $((\boldsymbol{x}_t, y_t))$ with margin μ, so $L_\mu(\boldsymbol{u}, S) = 0$. Choose $\rho = (1 - \varepsilon)\mu$ for some $0 < \varepsilon \leq 1$.

The bound is minimised for $\eta = (\mu - \rho)/X^2$, giving

$$\sum_{t=1}^{T} \tilde{\sigma}_t \leq \frac{X^2}{\varepsilon^2 \mu^2}.$$

Setting $\varepsilon = 1$ gives the familiar Perceptron mistake bound.

More generally, if we know the optimal margin μ, we can force the algorithm to find a hypothesis with margin within ε of optimal, but the number of updated needed for that grows as $O(1/\varepsilon^2)$.

Knowing μ in advance is often not realistic. There are more sophisticated algorithms that can approximate the optimal margin without knowing it in advance. □

Proof of Marginalised Perceptron bound: We consider the potential

$$P_t = \frac{1}{2} \|\boldsymbol{u} - \boldsymbol{w}_t\|_2^2 .$$

This time we use the learning rate η to take care of proper scaling.

We decompose the potential decrease as

$$\begin{aligned} P_t - P_{t+1} &= \frac{1}{2} \|\boldsymbol{u} - \boldsymbol{w}_t\|_2^2 - \frac{1}{2} \left\|\boldsymbol{u} - \boldsymbol{w}'_{t+1}\right\|_2^2 \\ &\quad + \frac{1}{2} \left\|\boldsymbol{u} - \boldsymbol{w}'_{t+1}\right\|_2^2 - \frac{1}{2} \|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|_2^2 . \end{aligned}$$

Since $\boldsymbol{u}$ is within the unit ball $B = \{ \boldsymbol{w} \mid \|\boldsymbol{w}\|_2 \leq 1 \}$, and $\boldsymbol{w}_{t+1}$ is the projection of $\boldsymbol{w}'_{t+1}$ onto that ball, we can estimate

$$\frac{1}{2} \left\|\boldsymbol{u} - \boldsymbol{w}'_{t+1}\right\|_2^2 - \frac{1}{2} \|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|_2^2 \geq 0.$$

As in the Perceptron convergence proof, we have

$$\begin{aligned}\frac{1}{2}\left\|\boldsymbol{u}-\boldsymbol{w}_t\right\|_2^2-\frac{1}{2}\left\|\boldsymbol{u}-\boldsymbol{w}'_{t+1}\right\|_2^2 &= \left(\boldsymbol{w}'_{t+1}-\boldsymbol{w}_t\right)\cdot\left(\boldsymbol{u}-\boldsymbol{w}_t\right)-\frac{1}{2}\left\|\boldsymbol{w}_t-\boldsymbol{w}'_{t+1}\right\|_2^2 \\ &= \eta\tilde{\sigma}_t y_t(\boldsymbol{u}-\boldsymbol{w}_t)\cdot\boldsymbol{x}_t-\frac{1}{2}\eta^2\tilde{\sigma}_t\left\|\boldsymbol{x}_t\right\|_2^2 .\end{aligned}$$

We estimate $\tilde{\sigma}_t y_t \boldsymbol{w}_t\cdot\boldsymbol{x}_t\le\tilde{\sigma}_t\rho$ and $\tilde{\sigma}_t y_t\boldsymbol{u}\cdot\boldsymbol{x}_t\ge\tilde{\sigma}_t\mu-L_\mu(\boldsymbol{u},\boldsymbol{x}_t,y_t)$. Combining with the previous, we get

$$P_t-P_{t+1}\ge-\eta L_\mu(\boldsymbol{u},\boldsymbol{x}_t,y_t)+\tilde{\sigma}_t\eta\left(\mu-\rho-\eta X^2/2\right).$$

The result follows by summing over $t=1,\ldots,T$ and noticing $P_1=\frac{1}{2}\left\|\boldsymbol{u}\right\|_2^2$ and $P_{T+1}\ge 0$. □

Online prediction: summary We were interested in relative loss bounds or regret bounds of the form

$$L(A,S) \leq (1+o(1)) \min_{f \in F} L(f,S)$$

where F is some comparison class of predictors (e.g., a set of experts, or all norm-bounded linear predictors). We did not always get exactly this form, in particular with discrete loss.

The algorithms are in general simple to implement and run fast.

Analysis is based on potential functions. With linear prediction, margins are important. Mathematical tools include Jensen's inequality, but in general we do not need any deep results from mathematics.

Some interesting topics that we omitted

- the Perceptron is not attribute-efficient: it does not really take advantage of the situation when the "target" classifier is sparse. The attribute-efficient counterpart is called Winnow.
- There is a family of p-norm algorithms ($1 \leq p \leq 2$) that interpolate between the Perceptron and Winnow.
- The ideas about linear classifier learning (attribute-efficient and otherwise) can be generalised to linear and logistic regression.
- The online prediction model can be extended to model situations where the "target" changes over time.
- All this is a special case of online convex optimisation. A central tool is Bregman divergences, which include squared Euclidean distance and relative entropy.
- In the so-called bandit model, the algorithm gets only partial feedback about its predictions.
- Follow the Perturbed Leader is another algorithm for combining expert advice. It is particularly interesting for special cases where the experts come from a known class of combinatorial objects (say, trying to learn a minimum spanning tree online).

After this part of the course you are able to . . .

- interpret relative loss bounds and explain the role of potential functions in obtaining them
- reproduce the loss bound proofs for the Aggregating Algorithm and the Perceptron Algorithm and apply their basic ideas on related problems
- apply Jensen's inequality in different situations
- explain and apply the basic idea of kernels to linear online classification.

3. Statistical learning

We start by introducing

- the Support Vector Machine (SVM)

which is a very popular batch-learning algorithm for linear classifiers. It is usually used with kernels to implement a feature map, and uses quadratic optimisation to maximise a margin, or more generally to minimise margin-based cost function.

We continue by introducing complexity measures of hypothesis classes that can be used to prove uniform convergence of empirical risks to true risks:

- Vapnik-Chervonenkis (VC) dimension, which is a fairly simple measure of just the hypothesis class
- Rademacher complexity, which incorporates the input distribution, making things more complicated but allowing more accurate bounds, in particular in terms of margins.

Support Vector Machine: basics

It is quite possible to use SVMs as plain linear classifiers, without a feature mapping. However the possibility of kernelising the SVM is so central in both applications and theory that from the beginning we assume that our examples are of the form $(\boldsymbol{\psi}(x_j), y_j)$ where $\boldsymbol{\psi}$ is the feature mapping.

The algorithm then produces a classifier of the form

$$g(x) = \text{sign}\left(\sum_{j=1}^{m} \alpha_j y_j k(x, x_j)\right),$$

where $k(\cdot, \cdot)$ is the kernel corresponding to $\boldsymbol{\psi}$ and $\alpha_j \geq 0$ are coefficients determined by an optimisation procedure.

Typically most of the coefficients α_j are zero. The vectors $\boldsymbol{\psi}(x_j)$ with $\alpha_j > 0$ are called support vectors.

There are two main versions of SVM classifier: hard margin SVM which enforces a positive margin on all sample points, and soft margin SVM which minimises a continuous cost function based on the hinge loss.

There are also SVM-based algorithms for regression and unsupervised learning.

A note on feature spaces

Assume that the feature mapping ψ is a function from X to $\mathcal{H}$. In most examples up to now, the feature space $\mathcal{H}$ has been just $\mathbb{R}^d$ for some $d \in \mathbb{N}$, it has been possible to write out explicitly the features $\psi_i(x)$, $i = 1, \ldots, d$, and the kernel function $k(x, x')$ gives the usual dot product $\psi(x) \cdot \psi(x')$.

However, the only thing we really have required is that the feature space $\mathcal{H}$ is a vector space (over $\mathbb{R}$) that also has an inner product $\langle \cdot, \cdot \rangle$ satisfying the usual conditions

positive definiteness: $\langle x, x \rangle \geq 0$ for all $x \in \mathcal{H}$, with equality only for $x = 0$

symmetricity: $\langle x, y \rangle = \langle y, x \rangle$ for all $x, y \in \mathcal{H}$,

linearity: $\langle ax + by, z \rangle = a \langle x, z \rangle + b \langle y, z \rangle$ for all $x, y, z \in \mathcal{H}$ and $a, b \in \mathbb{R}$.

Using the inner product we can as usual define the norm

$$\|x\| = (\langle x, x \rangle)^{1/2}.$$

In addition to taking inner products, we shall be doing optimisation over H. Therefore, we also need H to be complete.

An infinite sequence $(x_1, x_2, \ldots)$ in a norm space is a Cauchy sequence if

$$\lim_{n\to\infty} \sup_{m>n} \|x_m - x_n\| = 0.$$

The space A is complete, if for any Cauchy sequence (x_n) there is a point $x \in A$ such that $\lim_{n\to\infty} x_n = x$.

Thus, we are going to require that H is a complete inner product space. (Completeness is with respect to the norm defined by the inner product.) Such spaces are called Hilbert spaces.

We shall not go further into the theory of Hilbert spaces. Just notice that the most common examples are $\mathbb{R}^d$ equipped with the usual dot product, and

$$\ell^2 = \left\{ (x_1, x_2, x_3, \ldots) \in \mathbb{R}^\infty \mid \sum_{i=1}^{\infty} |x_i| < \infty \right\}$$

with the inner product

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{\infty} x_i y_i.$$

Optimising margins

Assume now that we have chosen a kernel k on X, and thus implicitly a feature map $\boldsymbol{\psi}\colon X \to \mathcal{H}$ for some Hilbert space $(\mathcal{H}, \langle\cdot,\cdot\rangle)$. We consider functions $f\colon X \to \mathbb{R}$ of the form

$$f(x) = \langle \boldsymbol{w}, \boldsymbol{\psi}(x)\rangle - b$$

for some $\boldsymbol{w} \in \mathcal{H}$ and $b \in \mathbb{R}$. Notice that we now explicitly include the bias b. As usual, we get a classifier as $g(x) = \mathrm{sign}(f(x))$.

Recall that

$$\frac{|f(x)|}{\|\boldsymbol{w}\|}$$

is the geometrical distance between the point $\boldsymbol{\psi}(x)$ and the hyperplane $f(x) = 0$ (in Hilbert space $\mathcal{H}$). Further, $yf(x) > 0$ means that $g(x) = y$.

Suppose now that the sample is linearly separable in the feature space, i.e., some $\boldsymbol{w} \in \mathcal{H}$ satisfies $y_j f(x_j) > 0$ for all j. For doing empirical risk minimisation, it would be sufficient to pick any such $\boldsymbol{w}$ as our hypothesis. However, here we actually pick the one that maximises the margin. This seems intuitively natural, and also turns out to result in a good bound for the true risk.

For notational convenience, we formalise this as a minimisation problem.

Optimisation 3.1: Variables $\boldsymbol{w} \in \mathcal{H}$, $b \in \mathbb{R}$, $\mu \in \mathbb{R}$

$$\begin{array}{ll} \textbf{minimise} & -\mu \\ \textbf{subject to} & \mu \geq 0 \\ & y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) - \mu \geq 0 \text{ for } i = 1, \ldots, m \\ & \|\boldsymbol{w}\|^2 \leq 1. \end{array}$$

Remarks: A variable assignment is feasible if it satisfies all the constraints. In our case, if $(\boldsymbol{w}, b, \mu)$ is feasible, then $(r\boldsymbol{w}, rb, r\mu)$ is feasible for all $0 < r \leq 1/\|\boldsymbol{w}\|$. This means that

1. If feasible solutions exist, the optimum occurs for $\|\boldsymbol{w}\| = 1$.

2. We would get the same solutions, up to rescaling, by minimising $\|\boldsymbol{w}\|^2$ subject to constraints $y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle - b) \geq 1$. This formalisation is a quadratic optimisation problem.

Brief introduction to convex optimisation

Consider the following problem where f, p_i and q_i are functions $\mathbb{R}^n \to \mathbb{R}$.

Optimisation 3.2: Variables $\boldsymbol{x} \in \mathbb{R}^n$

$$\begin{array}{ll} \textbf{minimise} & f(\boldsymbol{x}) \\ \textbf{subject to} & p_i(\boldsymbol{x}) \leq 0 \text{ for } i = 1, \ldots, m \\ & q_j(\boldsymbol{x}) = 0 \text{ for } j = 1, \ldots, l. \end{array}$$

The problem is convex if f is convex, p_i is convex for $i = 1, \ldots, m$, and q_j is affine for $j = 1, \ldots, l$ (i.e., $q_j(\boldsymbol{x}) = \boldsymbol{u}_j \cdot \boldsymbol{x} + b_j$ for some $\boldsymbol{u}_j \in \mathbb{R}^n$, $b_j \in \mathbb{R}$).

Notice that Optimisation 3.1 is convex (with some trivial rewriting of the constraints).

The feasible set is the set of $\boldsymbol{x}$ such that the constraints $p_i(\boldsymbol{x}) \leq 0$ and $q_j(\boldsymbol{x}) = 0$ are satisfied. For a convex problem, the feasible set is convex (i.e., if $\boldsymbol{x}$ and $\boldsymbol{y}$ are feasible, so is $(1 - v)\boldsymbol{x} + v\boldsymbol{y}$ for all $0 \leq v \leq 1$).

The gradient is the generalisation of the derivative for functions of several variables.

Consider a function $f\colon \mathbb{R}^d \to \mathbb{R}$. We obtain the partial derivative of f with respect to x_i by differentiating f with respect to x_i while keeping the rest of the variables constant:

$$\frac{\partial f(x_1, \ldots, x_d)}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_d) - f(x_1, \ldots, x_i, \ldots, x_d)}{h}.$$

The gradient $\nabla f(\boldsymbol{x})$ is just the vector of the partial derivatives:

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{x_d} \right).$$

If the gradient is defined everywhere, i.e. f is everywhere differentiable, then at any $\boldsymbol{x}$ that is a local optimum (minimum or maximum) of f, the gradient $\nabla f(\boldsymbol{x})$ is zero. (If, say, $\partial f(\boldsymbol{x})/\partial x_3$ is positive, then $f(\boldsymbol{x})$ could be increased by increasing x_3 and decreased by decreasing x_3.)

Consider now $\boldsymbol{g}\colon \mathbb{R} \to \mathbb{R}^d$ and $f\colon \mathbb{R}^d \to \mathbb{R}$. We have a chain rule

$$\frac{df(\boldsymbol{g}(z)}{dz} = \sum_{i=1}^{d} g_i'(z) \frac{\partial f(\boldsymbol{y})}{\partial y_i} = \boldsymbol{g}'(z) \cdot \nabla f(\boldsymbol{y})$$

where the partial derivatives of f are evaluated at $\boldsymbol{y} = \boldsymbol{g}(z)$.

In particular, consider how fast f increases when we move from point $\boldsymbol{x}$ in direction $\boldsymbol{r}$:

$$\frac{df(\boldsymbol{x} + h\boldsymbol{r})}{dh} = \boldsymbol{r} \cdot \nabla f(\boldsymbol{x})$$

where we substituted $h \leftarrow 0$ on the right-hand side. Thus, keeping the norm of $\boldsymbol{r}$ constant, we see that f increases the fastest when $\boldsymbol{r}$ is parallel to $\nabla f(\boldsymbol{x})$, and remains constant (as a first-order approximation) if $\boldsymbol{r}$ is orthogonal to $\nabla f(\boldsymbol{x})$.

In other words, $\nabla f(\boldsymbol{x})$ is perpendicular to the level curve $\{\, \boldsymbol{y} \mid f(\boldsymbol{y}) = f(\boldsymbol{x}) \,\}$ at $\boldsymbol{x}$, and points to the direction of steepest ascent.

Suppose now that we have just one inequality constraint:

minimise $f(\boldsymbol{x})$
over $\boldsymbol{x} \in \mathbb{R}^n$
subject to $p(\boldsymbol{x}) \leq 0$.

To get a physical representation of the situation, consider $f(\boldsymbol{x})$ as the potential (say, elevation) at point $\boldsymbol{x}$, so the negative gradient $-\nabla f(\boldsymbol{x})$ is the force acting on a particle at point $\boldsymbol{x}$.

Further, along the surface $p(\boldsymbol{x}) = 0$ there is a fence. For a particle at point $\boldsymbol{x}$ with $p(\boldsymbol{x}) = 0$, the fence exerts a contraint force that is just sufficient to keep the particle on the right side. The constraint force is perpendicular to the fence and points invards, so it is $-\lambda \nabla p(\boldsymbol{x})$ for some $\lambda > 0$.

Since we assume an infinitely strong fence, we have no prior knowledge of how large λ might be. However, if the particle does not touch the fence, the constraint force is zero: if $p(\boldsymbol{x}) \neq 0$, then $\lambda = 0$, which we write as $\lambda p(\boldsymbol{x}) = 0$.

The solution to the optimisation problem is the point $\boldsymbol{x}^*$ at which the particle is at balance, i.e., the net force acting on it is zero. Summarising the conditions, we get

$$\begin{aligned} \nabla f(\boldsymbol{x}^*) + \lambda \nabla p(\boldsymbol{x}^*) &= 0 \\ \lambda &\geq 0 \\ \lambda p(\boldsymbol{x}^*) &= 0. \end{aligned}$$

If there are several inequality constraints, then each fence $p_i(\boldsymbol{x})$ can exert its own constraint force $-\lambda_i \nabla p_i(\boldsymbol{x})$, again subject to $\lambda_i \geq 0$ and $\lambda_i p_i(\boldsymbol{x}) = 0$.

For equality constraints, the particle is prevented from leaving the surface $q_j(\boldsymbol{x}) = 0$. The constraint force is still perpendicular to the surface, but can be to either direction, so it is given by $\beta_j \nabla q_j(\boldsymbol{x})$ where $\beta_j \in \mathbb{R}$.

By combining all the above, we get the Karush-Kuhn-Tucker (KKT) conditions for optimisation problem 3.2:

$$\begin{aligned}
\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla p_i(\boldsymbol{x}^*) + \sum_{j=1}^{l} \beta_j \nabla q_j(\boldsymbol{x}^*) &= 0 \\
p_i(\boldsymbol{x}^*) &\leq 0 \qquad \text{for } i = 1, \ldots, m \\
q_j(\boldsymbol{x}^*) &= 0 \qquad \text{for } j = 1, \ldots, l \\
\lambda_i &\geq 0 \qquad \text{for } i = 1, \ldots, m \\
\lambda_i p_i(\boldsymbol{x}^*) &= 0 \qquad \text{for } i = 1, \ldots, m.
\end{aligned}$$

If the problem is convex (and satisfies some regularity conditions), there are no local minima, and the KKT conditions can be proved to be necessary and sufficient for $\boldsymbol{x}^*$ to be the solution of Optimisation 3.2.

If the problem is not convex, the situation is more complicated, but often the KKT conditions are still necessary.

A further important notion is Lagrange duality (or simply duality).

Given Optimisation 3.2, we define the Lagrangian $L\colon \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^l \to \mathbb{R}$ by

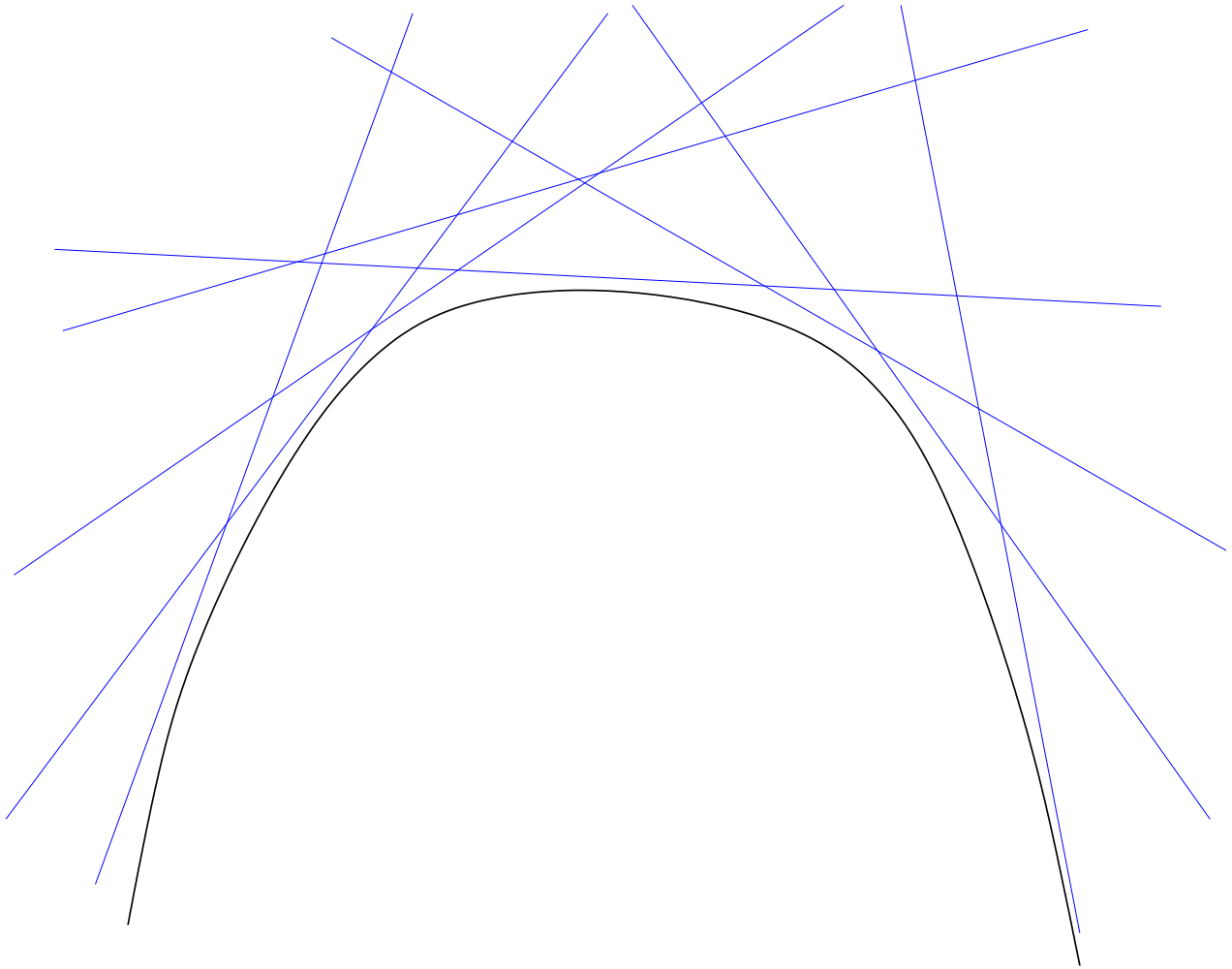
$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_j q_j(\boldsymbol{x}).$$

(Compare with KKT conditions.) The new variable λ_i is called the Lagrange multiplier for the constraint $p_i(\boldsymbol{x}) \leq 0$, and similarly for β_j.

The dual function is $g\colon \mathbb{R}^m \times \mathbb{R}^l \to \mathbb{R} \cup \{-\infty\}$ where

$$g(\boldsymbol{\lambda}, \boldsymbol{\beta}) = \inf_{\boldsymbol{x} \in \mathbb{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta}).$$

Since g is a pointwise infimum of affine functions, it is concave even if the original (primal) problem is not convex.



The infimum of affine functions is concave.

Let $\boldsymbol{x}$ be feasible (i.e., satisfy the constraints), $\lambda_i \geq 0$ for all i, and $\beta_j \in \mathbb{R}$ arbitrary. Since $p_i(\boldsymbol{x}) \leq 0$, we have

$$\sum_{i=1}^{m} \lambda_i p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_i q_i(\boldsymbol{x}) \leq 0.$$

Therefore

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_i q_i(\boldsymbol{x}) \leq f(\boldsymbol{x})$$

which implies

$$g(\boldsymbol{\lambda}, \boldsymbol{\beta}) = \inf_{\boldsymbol{x}' \in \mathbb{R}^n} L(\boldsymbol{x}', \boldsymbol{\lambda}, \boldsymbol{\beta}) \leq f(\boldsymbol{x}).$$

We can now define the dual problem of the primal problem 3.2 as the following.

Optimisation 3.3: Variables: $\boldsymbol{\lambda} \in \mathbb{R}^m$, $\boldsymbol{\beta} \in \mathbb{R}^l$

$$\begin{array}{ll} \textbf{maximise} & g(\boldsymbol{\lambda}, \boldsymbol{\beta}) \\ \textbf{subject to} & \lambda_i \geq 0 \text{ for } i = 1, \ldots, m. \end{array}$$

We call $(\boldsymbol{\lambda}, \boldsymbol{\beta}) \in \mathbb{R}^m \times \mathbb{R}^l$ dual feasible if $\lambda_i \geq 0$ for all i.

Let $\boldsymbol{x}^*$ be a solution to the primal 3.2 and $(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ to the dual 3.3. (They need not be unique.)

Given a feasible $\boldsymbol{x}$ and dual feasible $(\boldsymbol{\lambda}, \boldsymbol{\beta})$, the remark on previous page gives

$$g(\boldsymbol{\lambda}, \boldsymbol{\beta}) \leq g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*) \leq f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}).$$

Many optimisation algorithms produce a sequence of dual-primal feasible $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$, from which we thus get immediately an estimate of how far we are from the optimal values.

Given primal-dual feasible $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$, the quantity $f(\boldsymbol{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\beta})$ is called the duality gap.

The quantity $f(\boldsymbol{x}^*) - g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ is the optimal duality gap. The fact that it is always non-negative, as we just saw, is called weak duality. If it is actually zero, i.e., $f(\boldsymbol{x}^*) = g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$, we say that strong duality holds.

Convexity with some additional conditions is a sufficient (but not necessary) condition for strong duality. Requiring the existence of "strictly feasible" solutions is one way of setting the extra conditions.

Theorem 3.4 (Slater): If Optimisation 3.2 is convex and there is some $\boldsymbol{x} \in \mathbb{R}^n$ such that $p_i(\boldsymbol{x}) < 0$ for $1 \leq i \leq m$ and $q_j(\boldsymbol{x}) = 0$ for $1 \leq j \leq l$, then strong duality holds.

Proof can be found in optimisation text books. □

Assume that $\boldsymbol{x}^*$ is primal and $(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ dual optimal, and strong duality holds. Then

$$\begin{aligned} f(\boldsymbol{x}^*) &= g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*) \\ &= \inf_{\boldsymbol{x} \in \mathbb{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*) \\ &\leq f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* p_i(\boldsymbol{x}^*) + \sum_{j=i}^{l} \beta_j^* q_j(\boldsymbol{x}^*) \\ &\leq f(\boldsymbol{x}^*) \end{aligned}$$

where the last step follows from primal and dual feasibility. Thus the inequalities hold as equalities, and in particular $\sum_{i=1}^{m} \lambda_i^* p_i(\boldsymbol{x}^*) = 0$. Since each term in the sum in non-negative, we must have

$$\lambda_i^* p_i(\boldsymbol{x}^*) = 0 \qquad \text{for } 1 \leq i \leq m.$$

We already gave an intuitive motivation for this condition known as complementary slackness. It means that at the optimum, the Lagrange coefficient can be non-zero only for constraints that are active.

Another important observation is that $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \mathbb{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$.

Recall that the KKT conditions for $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$ are

$$\nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \nabla p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_j \nabla q_j(\boldsymbol{x}) = 0 \tag{1}$$

$$\begin{aligned} p_i(\boldsymbol{x}) &\leq 0 \quad \text{for } i = 1, \ldots, m && (2)\\ q_j(\boldsymbol{x}) &= 0 \quad \text{for } j = 1, \ldots, l && (3)\\ \lambda_i &\geq 0 \quad \text{for } i = 1, \ldots, m && (4)\\ \lambda_i p_i(\boldsymbol{x}) &= 0 \quad \text{for } i = 1, \ldots, m. && (5) \end{aligned}$$

Theorem 3.5 (Karush-Kuhn-Tucker): Let $\boldsymbol{x}^*$ be primal and $(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ dual optimal, and assume that strong duality holds. Assume futher that f and p_i are differentiable. Then $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ satisfies the KKT conditions. Further, if the primal problem is convex, then any $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$ that satisfies the KKT conditions is optimal.

Proof: Let $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ be as assumed. Conditions (2)–(4) are the feasibility constraints, and (5) is the complementary slackness we just verified.

Finally, since $\boldsymbol{x} = \boldsymbol{x}^*$ minimises $L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$, the gradient of $L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ w.r.t. $\boldsymbol{x}$ must be zero at $\boldsymbol{x} = \boldsymbol{x}^*$.

Assume now that the problem is convex and $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}})$ satisfy the KKT conditions. In particular, this implies feasibility. Since $\tilde{\lambda}_i \geq 0$, we see that $L(\boldsymbol{x}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}})$ is convex in $\boldsymbol{x}$ and therefore attains its minimum when the gradient is zero. Using (1), this happens for $\boldsymbol{x} = \tilde{\boldsymbol{x}}$. We conclude

$$\begin{aligned} g(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}}) &= L(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}}) \\ &= f(\tilde{\boldsymbol{x}}) + \sum_{i=1}^{m} \tilde{\lambda}_i p_i(\tilde{\boldsymbol{x}}) + \sum_{j=1}^{l} \tilde{\beta}_j q_j(\tilde{\boldsymbol{x}}) \\ &= f(\tilde{\boldsymbol{x}}) \end{aligned}$$

where the last step uses (3) and (5). Hence, we have zero duality gap, which implies optimality. □

We are now ready to tackle the hard margin SVM optimisation 3.1. Clearly the problem is convex, and if a positive margin is possible then Slater's condition is satisfied.

It should be noticed that we are (conceptually) optimising over $\boldsymbol{w} \in \mathcal{H}$ for some Hilbert space, and not $\boldsymbol{w} \in \mathbb{R}^n$. However, it is easy to see that all the required properties still hold.

First, we set up the Lagrangian

$$L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda) = -\mu - \sum_{i=1}^{m} \alpha_i (y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) - \mu) + \lambda(\|\boldsymbol{w}\|^2 - 1)$$

where the constraints for the Lagrange coefficients are $\alpha_i \geq 0$ and $\lambda \geq 0$.

We have omitted the constraint $\mu \geq 0$ since we can just check afterwards whether it holds. If not, then the problem is not separable.

We get the dual by optimising away the primal variables. The derivatives are

$$\begin{aligned}
\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)}{\partial \boldsymbol{w}} &= -\sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\psi}(x_i) + 2\lambda \boldsymbol{w} \\
\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)}{\partial b} &= \sum_{i=1}^{m} \alpha_i y_i \\
\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)}{\partial \mu} &= -1 + \sum_{i=1}^{m} \alpha_i .
\end{aligned}$$

Setting them all to zero gives us

$$\begin{aligned}
\boldsymbol{w} &= \frac{1}{2\lambda} \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\psi}(x_i) \\
\sum_{i=1}^{n} \alpha_i y_i &= 0 \\
\sum_{i=1}^{n} \alpha_i &= 1.
\end{aligned}$$

We now simplify the dual

$$g(\boldsymbol{\alpha}, \lambda) = \inf_{\boldsymbol{w}, b, \mu} L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)$$

assuming $\sum_{i=1}^{m} \alpha_i y_i = 0$ and $\sum_{i=1}^{m} \alpha_i = 1$. (If these conditions do not hold, then $g(\boldsymbol{\alpha}, \lambda) = -\infty$.)

Plugging also the condition for $\boldsymbol{w}$ into the Lagrangian gives us

$$\begin{aligned} g(\boldsymbol{\alpha}, \lambda) &= -\sum_{i=1}^{m} \alpha_i y_i \langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle + \lambda \|\boldsymbol{w}\|^2 - \lambda \\ &= \left(-\frac{1}{2\lambda} + \frac{1}{4\lambda} \right) \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) - \lambda \\ &= \frac{1}{4\lambda} W(\boldsymbol{\alpha}) - \lambda \end{aligned}$$

where

$$W(\boldsymbol{\alpha}) = -\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) = -\|2\lambda \boldsymbol{w}\|^2 .$$

Maximising w.r.t. lambda gives

$$\max_{\lambda \geq 0} g(\boldsymbol{\alpha}, \lambda) = -(-W(\boldsymbol{\alpha}))^{1/2}$$

with the maximum attained at $\lambda = \frac{1}{2}(-W(\boldsymbol{\alpha}))^{1/2}$. Hence, the dual solution $(\boldsymbol{\alpha}^*, \lambda^*)$ is obtained as

$$\boldsymbol{\alpha}^* = \arg \max_{\boldsymbol{\alpha}} \left(- \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \right) = \arg \max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}),$$

where the optimisation is subject to $\sum_{i=1}^{m} \alpha_i y_i = 0$ and $\sum_{i=1}^{m} \alpha_i = 1$, and

$$\lambda^* = \frac{1}{2} \left(\sum_{i,j=1}^{m} \alpha_i^* \alpha_j^* y_i y_j k(x_i, x_j) \right)^{1/2}.$$

We still need to recover the primal variables. Since $-\mu$ is the objective function, strong duality implies

$$\mu^* = -g(\boldsymbol{\alpha}^*, \lambda^*) = (-W(\boldsymbol{\alpha}^*))^{1/2} = 2\lambda^*.$$

We already noticed

$$\boldsymbol{w}^* = \frac{1}{2\lambda^*} \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{\psi}(x_i).$$

To solve b^*, we notice that by complementary slackness we have $y_i(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i) \rangle - b^*) = \mu^*$ for any i such that $\alpha_i^* \neq 0$.

For simplicity, we choose as the final output $(\boldsymbol{w}, b)$ of the algorithm a scaled version $(2\lambda^* \boldsymbol{w}^*, 2\lambda^* b^*)$ of the solution to the original optimisation.

We summarise the preceding observations.

Algorithm 3.6 (Hard Margin SVM): Input: sample $((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times \{-1, 1\})^m$

1. Obtain $\boldsymbol{\alpha}^* \in \mathbb{R}^m$ as maximiser of

$$W(\boldsymbol{\alpha}) = \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

subject to $\sum_{i=1}^m \alpha_i = 1$, $\sum_{i=1}^m \alpha_i y_i = 0$ and $\alpha_i \geq 0$ for all i.

2. Let $\mu^* = (-W(\boldsymbol{\alpha}^*))^{1/2}$. Choose any i such that $\alpha_i^* \neq 0$ and set

$$b = \sum_{j=1}^{m} \alpha_j^* y_j k(x_i, x_j) - y_i (\mu^*)^2.$$

3. Output the classifier $\operatorname{sign}(f(\cdot))$ where

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i^* y_i k(\cdot, x_i) - b.$$

The support vectors are those feature vectors $\boldsymbol{\psi}(x_i)$ for which $\alpha_i^* > 0$. By the KKT conditions, the support vectors satisfy

$$y_i(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i) \rangle - b^*) = \mu^*.$$

In other words, the optimal classifier has margin exactly μ^* on the support vectors.

(To understand the name of the method, visualise a strut of lenght μ^* from each support vector perpendicularly to the optimal separating hyperplane.)

Typically only a small proportion of the examples end up as support vectors. This is important from a computational point of view. Notice that to represent the hypothesis

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i^* y_i k(\cdot, x_i)$$

we need to store only those x_i, y_i and α_i^* for which $\alpha_i^* > 0$.

If we omit the (ultimately redundant) constraint $\mu \geq 0$, then for any $\widehat{\boldsymbol{w}}$ with $\|\widehat{\boldsymbol{w}}\| = 1$ there is $\widehat{b} \in \mathbb{R}$ and $\widehat{\mu} \in \mathbb{R}$ such that $(\widehat{\boldsymbol{w}}, \widehat{b}, \widehat{\mu})$ is feasible for the primal. The largest possible such $\widehat{\mu}$ is given by

$$\widehat{\mu} = \frac{1}{2}\left(\min\left\{\, \langle \widehat{\boldsymbol{w}}, \boldsymbol{\psi}(x_i)\rangle \mid y_i = 1 \,\right\} - \max\left\{\, \langle \widehat{\boldsymbol{w}}, \boldsymbol{\psi}(x_i)\rangle \mid y_i = -1 \,\right\}\right).$$

Given $\widehat{\boldsymbol{\alpha}}$ that is feasible for the dual, let

$$\widehat{\boldsymbol{w}} = \frac{1}{-W(\widehat{\boldsymbol{\alpha}})^{1/2}} \sum_{i=1}^{m} \widehat{\alpha}_i y_i \boldsymbol{\psi}(x_i)$$

as earlier. Then duality theory gives a bound

$$-(-W(\widehat{\boldsymbol{\alpha}}))^{1/2} \leq -\mu^* \leq -\widehat{\mu},$$

and if equality holds then we are at the optimum.

Soft margin SVM

Instead of having a hard constraint that the hinge losses $L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i)$ must be zero, we take them as a part of the function to be minimised.

Since the non-differentiable "hinge" would be a problem in the optimisation, we use the standard trick of introducing free variables ξ_i, the so-called slack variables, and constraining them suitably.

Optimisation 3.7: Variables $\boldsymbol{w} \in \mathcal{H}$, $b \in \mathbb{R}$, $\boldsymbol{\xi} \in \mathbb{R}^m$, $\mu \in \mathbb{R}$

$$\begin{array}{ll} \textbf{minimise} & -\mu + C \sum_{i=1}^m \xi_i \\ \textbf{subject to} & \|\boldsymbol{w}\|^2 - 1 \leq 0 \\ & -\xi_i \leq 0 \text{ for } i = 1, \ldots, m \\ & \mu - \xi_i - y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) \leq 0 \text{ for } i = 1, \ldots, m \end{array}$$

The parameter $C > 0$ is an arbitrary positive constant. In practice a suitable value is determined by cross-validation.

Notice that after minimising w.r.t. ξ_i, the constraints imply
$\xi_i = \min\{\, 0, \mu - y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) \,\} = L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i)$.

The Lagrangian now becomes

$$L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = -\mu + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle - b) - \mu + \xi_i) - \sum_{i=1}^{m}\beta_i\xi_i + \lambda(\|\boldsymbol{w}\|^2 - 1)$$

where $\alpha_i \geq 0$, $\beta_i \geq 0$ and $\lambda \geq 0$. Differentiating w.r.t. the primal variables, we get

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial \boldsymbol{w}} = -\sum_{i=1}^{m}\alpha_i y_i \boldsymbol{\psi}(x_i) + 2\lambda \boldsymbol{w} \quad (6)$$

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial b} = \sum_{i=1}^{m}\alpha_i y_i \quad (7)$$

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial \mu} = -1 + \sum_{i=1}^{m}\alpha_i \quad (8)$$

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial \xi_i} = C - \alpha_i - \beta_i. \quad (9)$$

Consider the dual

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda).$$

If one of the derivatives (2)–(4) is non-zero, then $g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = -\infty$. Assume this is not the case.

Since L is convex in $\boldsymbol{w}$, we obtain the minimum by making (1) zero. This happens for

$$\boldsymbol{w} = \frac{1}{2\lambda} \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\psi}(x_i).$$

Further taking (2)–(4) to be zero gives us

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = \frac{1}{4\lambda} W(\boldsymbol{\alpha}) - \lambda$$

where

$$W(\boldsymbol{\alpha}) = -\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) = -\left\| 2\lambda \boldsymbol{w} \right\|^2$$

like in the hard margin case.

As in the hard margin case, we maximise w.r.t. λ to obtain

$$\max_{\lambda \geq 0} g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = -(-W(\boldsymbol{\alpha}))^{1/2}$$

with the optimum at $\lambda = \frac{1}{2}(-W(\boldsymbol{\alpha}))^{1/2}$. The solution to the dual is given by

$$\boldsymbol{\alpha}^* = \arg \max_{\boldsymbol{\alpha}} -(-W(\boldsymbol{\alpha}))^{1/2} = \arg \max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}).$$

Given the dual solution $(\boldsymbol{\alpha}^*, \lambda^*)$, we already know that

$$\boldsymbol{w}^* = \frac{1}{2\lambda^*} \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{\psi}(x_i) = \frac{1}{(-W(\boldsymbol{\alpha}^*))^{1/2}} \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{\psi}(x_i).$$

The remaining question is, how to get b^* and μ^*.

The constraint $\beta_i \geq 0$ and the KKT condition $C - \alpha_i - \beta_i = 0$ together imply the box constraint

$$0 \leq \alpha_i \leq C.$$

Intuitively, increasing α_i increases the margin of $\boldsymbol{w}$ on the example $(\boldsymbol{\psi}(x_i), y_i)$. In the hard margin case, we increase α_i as much as needed to push the slack variable to zero. In the soft margin case, we stop increasing α_i at some stage and allow the slack variable to get positive.

More formally, the complementary slackness conditions are

$$\begin{aligned} \alpha_i(y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle - b) - \mu + \xi_i) &= 0 \\ (C - \alpha_i)\xi_i &= 0. \end{aligned}$$

If $\xi_i > 0$, then $\alpha_i = C$. If $\alpha_i = 0$, then $\xi_i = 0$.

Assume first that there are some i and j such that $y_i = -1$, $y_j = 1$ and $0 < \alpha_i^*, \alpha_j^* < C$. Then $\xi_i = \xi_j = 0$, and

$$y_i(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle - b^*) - \mu^* = 0 = y_j(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle - b^*) - \mu^*$$

so

$$\begin{aligned} b^* &= \frac{1}{2}\left(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle + \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle\right) \\ \mu^* &= \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle - b^*. \end{aligned}$$

Further,

$$\begin{aligned} \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle &= \min\{\, \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k)\rangle \mid \alpha_k > 0, y_k = -1 \,\} \\ \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle &= \max\{\, \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k)\rangle \mid \alpha_k > 0, y_k = 1 \,\}. \end{aligned}$$

We can use these latter conditions to pick i and j, and then calculate b^* and μ^*, even in the special case that i and j such that $0 < \alpha_i^*, \alpha_j^* < C$ does not exist.

We summarise the preceding observations.

Algorithm 3.8 (1-norm Soft Margin SVM): Parameter: $C > 0$
Input: sample $((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times \{-1, 1\})^m$

1. Obtain $\boldsymbol{\alpha}^* \in \mathbb{R}^m$ as maximiser of $W(\boldsymbol{\alpha}) = \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j k(x_i, x_j)$ subject to $\sum_{i=1}^m \alpha_i = 1$, $\sum_{i=1}^m \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$ for $i = 1 \leq i \leq m$.

2. Let $\lambda^* = \frac{1}{2}(-W(\boldsymbol{\alpha}^*))^{1/2}$ and $\boldsymbol{w}^* = \frac{1}{2\lambda^*} \sum_{i=1}^m \alpha_i^* y_i \boldsymbol{\psi}(x_i)$

3. Choose i and j such that
$\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i) \rangle = \min \{ \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k) \rangle \mid \alpha_k > 0, y_k = -1 \}$ and
$\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j) \rangle = \max \{ \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k) \rangle \mid \alpha_k > 0, y_k = 1 \}$.

4. Let $b^* = \frac{1}{2} (\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i) \rangle + \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j) \rangle)$ and $\mu^* = \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j) \rangle - b^*$.

5. Output the classifier $\operatorname{sign}(f(\cdot))$ where

$$f(\cdot) = \frac{1}{2\lambda^*} \sum_{i=1}^m \alpha_i^* y_i k(\cdot, x_i) - b^*.$$

The ν SVM is just the soft margin SVM with $C = 1/(\nu m)$ for some $0 < \nu \leq 1$. This parameterisation has an intuitive interpretation.

Theorem 3.9: Run the ν SVM on sample S to obtain $\boldsymbol{w}^*$, b^* and μ^*. Then at most νm examples in S have margin less than μ^*, and at most $(1-\nu)m$ examples in S have margin larger than μ^*.

Proof: Recall that $\sum_{i=1}^{m} \alpha_i = 1$.

If $\xi_i > 0$, then $\alpha_i = 1/(\nu m)$. Since $\alpha_i \geq 0$, this can happen at most νm times.

Similarly, since $\alpha_i \leq 1/(\nu m)$, we need at least νm examples with $\alpha_i > 0$. They all have margin at most μ^*. □

Intutively, we can pick ν to correpond to the "noise rate" in the data. We allow a fraction ν of the data to fail to have a large margin. (In practice, cross-validation is still needed.)

Vapnik-Chervonenkis dimension

Theorem 1.10 gave a uniform convergence bound that

$$\max_{h \in H} \left| \hat{R}(h) - R(h) \right| \leq \varepsilon$$

holds with probability at least $1 - \delta$, where $R(h)$ is the true risk and $\hat{R}(h)$ the empirical risk on sample of size

$$m \geq \frac{1}{2\varepsilon^2} \ln \frac{2\,|H|}{\delta}.$$

The proof was based on Hoeffding's inequality and the union bound. The bound is rather crude. In particular it becomes vacuous if H is infinite, for example linear classifiers in $\mathbb{R}^d$.

Our first approach to providing bounds for infinite H is based on the Vapnik-Chervonenkis (VC) dimension.

Let H be a class of binary classifiers with instance space X. Let (A, B) be a partitioning of some $D \subseteq X$. In other words, $A \cup B = D$ and $A \cap B = \emptyset$. We say that the class H realises the partitioning (A, B) if there is a classifier $h \in H$ such that

$$\begin{aligned} h(x) &= -1 \qquad \text{if } x \in A \\ h(x) &= 1 \qquad \text{if } x \in B. \end{aligned}$$

We say that H shatters a set $D \subseteq X$, if every partitioning (A, B) of D is realised by some classifier $h \in H$.

The Vapnik-Chervonenkis dimension of H, denoted by $\text{VCdim}(H)$, is the largest m such that H shatters a set D with $|D| = m$. If H shatters arbitrarily large sets, we write $\text{VCdim}(H) = \infty$.

Example 3.10: If $\text{VCdim}(H) = d$, then $|H| \geq 2^d$. □

Example 3.11: Let $X = \mathbb{R}^n$ and H the class of linear classifiers where we allow a bias term. We show that $\text{VCdim}(H) = n + 1$.

To see $\text{VCdim}(H) \geq n + 1$, let $\boldsymbol{x}_{n+1} = \mathbf{0}$, and for $i = 1, \ldots, n$ let $\boldsymbol{x}_i$ be the ith unit vector: $x_{ij} = \delta_{ij}$ where δ_{ij} is Kronecker delta,

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

We claim that H shatters $D = \{ \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1} \}$. To see that, let $(y_1, \ldots, y_{n+1}) \in \{ -1, 1 \}^{n+1}$ be arbitrary. By defining $w_i = y_i$ for $i = 1, \ldots, n$, and $b = -y_{n+1}/2$, we get

$$\text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}_i - b) = y_i$$

for all i.

To see VCdim$(H) \leq n+1$, suppose for contradiction that H shatters $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+2}\} \subset \mathbb{R}^n$. Then, by the familiar reduction, we see that there is a set $\{\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_{n+2}\} \subset \mathbb{R}^{n+1}$ that is shattered by linear classifiers with bias 0. At least one $\tilde{\boldsymbol{x}}_i$ is a linear combination of the others. Without loss of generality, assume

$$\tilde{\boldsymbol{x}}_{n+2} = \sum_{i=1}^{n+1} a_i \boldsymbol{x}_i.$$

Choose now $y_i = \text{sign}(a_i)$ for $i = 1, \ldots, n+1$, and $y_{n+2} = -1$. Let $\boldsymbol{w}$ be such that $\text{sign}(\boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_i) = y_i$ for $1 \leq i \leq n+1$. Then

$$\text{sign}(\boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_{n+2}) = \text{sign}\left(\sum_{i=1}^{n+1} a_i \boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_i\right) = 1 \neq y_{n+2}.$$

Thus the labeling $(y_1, \ldots, y_{n+2})$ cannot be realised by a linear classifier. □

Although we will not prove it here, we remark in passing the key combinatorial result that makes the VC dimension important also outside learning theory. Let $\Pi_H(D)$ be the number of partitionings of D realised by H, and

$$\Pi_H(m) = \max\{\,\Pi_H(D) \mid |D| = m\,\}.$$

Thus, $\mathrm{VCdim}(H)$ is the largest m such that $\Pi_H(m) = 2^m$. The function $\Pi_H(\cdot)$ is called the growth function.

Lemma 3.12: If $\mathrm{VCdim}(H) = d < \infty$, we have

$$\Pi_H(m) \leq \sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{\mathrm{e}m}{d}\right)^d.$$

□

This is known as Sauer's Lemma. Hence, there are two cases:

- $\mathrm{VCdim}(H) = d$ is finite, and $\Pi_H(m)$ is bounded by a polynomial of degree d
- $\mathrm{VCdim}(H) = \infty$ and $\Pi_H(m) = 2^m$ for all m.

We are now ready to state one of the classical results in computational learning theory.

Theorem 3.13 (Vapnik and Chervonenkis, 1971): Assume $\mathrm{VCdim}(H) = d < \infty$. There is an absolute constant C such that

$$\Pr_{S \sim P^m} \left(\sup_{h \in H} \left| R(h) - \hat{R}(h) \right| > \varepsilon \right) \leq \delta$$

holds whenever

$$m \geq \frac{C}{\varepsilon^2} \left(d \ln \frac{2}{\varepsilon} + \ln \frac{2}{\delta} \right).$$

Here $\Pr_{S \sim P^m}(\phi)$ denotes the probability of ϕ when S is obtained by m independent draws from distribution P.

The original bound by Vapnik and Chervonenkis actually had an extra $\log d$ factor, which has since been removed by considerable technical effort.

We have omitted some measurability assumptions that technically should be in the theorem. They basically say that certain sets appearing in the proof must be measurable. We are not giving a proof for the theorem, and therefore don't bother writing out the assumptions. The only known examples where the conditions do not hold are extremely contrived.

The bound is usually too loose to give practically useful numerical error estimates. Despite its looseness, the bound gives useful intuition: in order to avoid overfitting, one could control the VC dimension of the hypothesis class.

One reason for the looseness of the bound is its worst-case nature. In particular, it is worst case with respect to P. We now go to Rademacher complexity that does take P into account. In particular, this will allow bounds based on quantities such as the margin that depend on the sample points.

For convenience, we will consider slightly more general uniform convergence bounds of the form

$$\Pr_{S\sim P^m}\left(\sup_{f\in F}\left|\mathop{\mathsf{E}}_{z\sim P}[f(z)]-\mathop{\mathsf{E}}_{z\sim S}[f(z)]\right|>\varepsilon\right)\leq\delta$$

where

- F is a class of functions $Z\to\mathbb{R}$
- $\mathsf{E}_{z\sim P}[f(z)]$ is the expectation of $f(z)$ when $z\in Z$ is drawn from P
- $S=(z_1,\ldots,z_m)$ is a sequence of m points drawn independently from P
- $\mathsf{E}_{z\sim S}[f(z)]=\frac{1}{m}\sum_{i=1}^m f(z_i)$ is the empirical average of f.

The special case we are interested in for a set of classifiers H is obtained by choosing $Z=X\times\{-1,1\}$ and letting $F=\{\ell_h\mid h\in H\}$ consist of functions

$$\ell_h(x,y)=\begin{cases}0 & \text{if } y=f(x)\\ 1 & \text{if } y\neq f(x).\end{cases}$$

This class F is the loss class corresponding to H. Here we have considered the 0-1 loss, but the results apply to other loss functions, too.

Let r_i, $i = 1, \ldots, m$, be independent Rademacher random variables, i.e., $r_i \in \{-1, 1\}$ and $\Pr(r_i = -1) = \Pr(r_i = 1) = 1/2$.

For a class F of functions $Z \to \mathbb{R}$ and a sample $S = (z_1, \ldots, z_m) \in Z^m$, define the empirical Rademacher complexity as

$$\widehat{\mathrm{Rad}}_m(F, S) = \mathop{\mathsf{E}}_{r_i \in \{-1,1\}} \left[\sup_{f \in F} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f(z_i) \right| \right].$$

The sum $\sum_i r_i f(z_i)$ is a measure of correlation between f and the labels $\boldsymbol{r}$.

If $\widehat{\mathrm{Rad}}_m(F, S)$ is high, then a random labelling of points $z_1, \ldots, z_m$ can be accurately matched by choosing a function from F. This means that F is rich enough that we are likely to find spurious regularities in the sample even if in reality there are none to be found. Thus we should not trust our empirical estimates too much.

Usually the empirical Rademacher complexity is denoted by $\hat{R}_m$, but we wish to avoid confusion with our notation for the empirical risk.

Finally, given a probability measure P over Z, define the Rademacher complexity of F as

$$\mathrm{Rad}_m(F) = \mathop{\mathsf{E}}_{S\sim P^m}\left[\widehat{\mathrm{Rad}}_m(F,S)\right].$$

Notice that unlike the VC dimension, the Rademacher complexity depends on P.

This leads to more accurate estimates when P is "easy", but makes it usually impossible to determine $\mathrm{Rad}_m(F)$ exactly in closed form.

Fortunately, for large m we have with high probability

$$\widehat{\mathrm{Rad}}_m(F,S) \approx \mathrm{Rad}_m(F)$$

so in practice it is sufficient to determine the empirical Rademacher complexity for a sample S from P^m.

The main result is the following.

Theorem 3.14: Let F be a class of functions $Z \to [0,1]$, and let P be a probability measure on Z. Let $0 < \delta \leq 1$. For $S = (z_1, \ldots, z_m)$ drawn from P^m, with probability at least $1 - \delta$ we have

$$\sup_{f \in F} \left| \mathop{\mathsf{E}}_{z \sim P}[f(z)] - \mathop{\mathsf{E}}_{z \sim S}[f(z)] \right| \leq \mathrm{Rad}_m(F) + \sqrt{\frac{1}{2m} \ln \frac{2}{\delta}}.$$

Assuming we know how to evaluate $\mathrm{Rad}_m(F)$ where F is the loss class we are interested in, we get a bound

$$R(h) \leq \hat{R}(h) + \mathrm{Rad}_m(F) + \sqrt{\frac{1}{2m} \ln \frac{2}{\delta}}$$

which holds with probability $1 - \delta$ for all hypotheses $h \in H$, in particular for the one chosen by Empirical Risk Minimisation (or any other learning algorithm).

Consider the case where F contains all functions from Z to $\{0,1\}$. Then $\mathrm{Rad}_m(F) = 1$, and the bound is trivial (as expected).

However, we see later that for many interesting classes F we have $\mathrm{Rad}_m(F) = O(m^{-1/2})$. Then we can apply the following.

Corollary 3.15: Assume that $\mathrm{Rad}_m(F) \leq cm^{-1/2}$ for all m. Then for

$$m \geq \frac{1}{\varepsilon^2}\left(2c^2 + \ln\frac{2}{\delta}\right)$$

we have

$$\sup_{f\in F}\left|\mathop{\mathsf{E}}_{z\sim P}[f(z)] - \mathop{\mathsf{E}}_{z\sim S}[f(z)]\right| \leq \varepsilon$$

with probability at least $1-\delta$.

Proof: Directly from the previous theorem using

$$\frac{1}{2}(\sqrt{a}+\sqrt{b}) \leq \sqrt{\frac{a+b}{2}}$$

(Jensen). □

Our proof of Theorem 3.14 is based on the following concentration inequality.

Theorem 3.16 (McDiarmid): Let $X_1, \ldots, X_m$ be independent random variables taking values in some set A. Assume that $f\colon A^m \to \mathbb{R}$ is such that for $i = 1, \ldots, m$ we have a constant c_i such that

$$\left|f(x_1, \ldots, x_m) - f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, x_m)\right| \leq c_i$$

for all $x_1, \ldots, x_m, x_i' \in A$. Write $\bar{f} = \mathsf{E}[f(X_1, \ldots, X_m)]$. Then for $\varepsilon > 0$ we have

$$\Pr(f(X_1, \ldots, X_m) - \bar{f} \geq \varepsilon) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^m c_i^2}\right).$$

□

As an example, we derive a version of the Hoeffding bound using McDiarmid's inequality.

Corollary 3.17: For $i = 1, \ldots, m$, let X_i be independent random variables with $a_i \leq X_i \leq b_i$ and $\mathsf{E}[X_i] = \bar{X}_i$. Let $S = \sum_{i=1}^{m} X_i$. Then

$$\Pr\left(\left|S - \sum_{i=1}^{m} \bar{X}_i\right| \geq \varepsilon\right) \leq 2\exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^{m}(b_i - a_i)^2}\right).$$

Proof: We apply McDiarmid separately to S and $-S$. The union bound gives the factor 2. □

Proof of Theorem 3.14: For $f \in F$ we have

$$\mathop{\mathsf{E}}_{z\sim P}[f(x)] \leq \mathop{\mathsf{E}}_{z\sim S}[f(z)] + p(S)$$

where

$$p(S) = \sup_{g\in F}\left(\mathop{\mathsf{E}}_{z\sim P}[g(z)] - \mathop{\mathsf{E}}_{z\sim S}[g(z)]\right).$$

We apply McDiarmid's inequality to function p. Since $g(z) \in [0,1]$, changing one z_i in S can change $\mathsf{E}_{z\sim S}[g(z)]$ by at most $1/m$. We solve for ε such that

$$\exp\left(-2m\varepsilon^2\right) = \frac{\delta}{2}$$

and plug into McDiarmid to get

$$p(S) \leq \mathop{\mathsf{E}}_{S\sim P^m}[p(S)] + \sqrt{\frac{1}{2m}\ln\frac{2}{\delta}}$$

with probability at least $1 - \delta/2$.

The key to the whole proof is now estimating $\mathsf{E}_{S\sim P^m}[p(S)]$.

First we express $\mathsf{E}_{z\sim P}[g(z)]$ in terms of a hypothetical sample $S' = (z'_1, \ldots, z'_m)$:

$$\begin{aligned}
\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] &= \mathop{\mathsf{E}}_{S\sim P^m}\left[\sup_{g\in F}\left(\mathop{\mathsf{E}}_{z\sim P}[g(z)] - \mathop{\mathsf{E}}_{z\sim S}[g(z)]\right)\right] \\
&= \mathop{\mathsf{E}}_{S\sim P^m}\left[\sup_{g\in F}\left(\mathop{\mathsf{E}}_{S'\sim P^m}\left[\frac{1}{m}\sum_{i=1}^m g(z'_i)\right] - \frac{1}{m}\sum_{i=1}^m g(z_i)\right)\right].
\end{aligned}$$

We then apply the fact $\sup_f \mathsf{E}[f] \le \mathsf{E}[\sup_f f]$ to get

$$\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] \le \mathop{\mathsf{E}}_{S,S'\sim P^m}\left[\sup_{g\in F}\left(\frac{1}{m}\sum_{i=1}^m (g(z'_i) - g(z_i))\right)\right].$$

Let now $r_1, \ldots, r_m$ be independent Rademacher random variables. Since $g(z'_i)$ and $g(z'_i)$ have same distribution, $g(z'_i) - g(z_i)$ has the same distribution as $r_i(g(z'_i) - g(z_i))$, and we get

$$\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] \le \mathop{\mathsf{E}}_{r_i\in\{-1,1\}}\mathop{\mathsf{E}}_{S,S'\sim P^m}\left[\sup_{g\in F}\left(\frac{1}{m}\sum_{i=1}^m r_i(g(z'_i) - g(z_i))\right)\right].$$

Since $\sup_x(f(x)+g(x)) \le \sup_x f(x) + \sup_x g(x)$, we get

$$\begin{aligned}
\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] &\le \mathop{\mathsf{E}}_{r_i\in\{-1,1\}} \mathop{\mathsf{E}}_{S,S'\sim P^m}\left[\sup_{g\in F}\left(\frac{1}{m}\sum_{i=1}^m r_i(g(z_i')-g(z_i))\right)\right] \\
&\le \mathop{\mathsf{E}}_{r_i\in\{-1,1\}} \mathop{\mathsf{E}}_{S,S'\sim P^m}\left[\sup_{g\in F}\left(\frac{1}{m}\sum_{i=1}^m r_i g(z_i')\right)+\sup_{g\in F}\left(-\frac{1}{m}\sum_{i=1}^m r_i g(z_i)\right)\right] \\
&\le 2\mathop{\mathsf{E}}_{r_i\in\{-1,1\}} \mathop{\mathsf{E}}_{S\sim P^m}\left[\sup_{g\in F}\left|\frac{1}{m}\sum_{i=1}^m r_i g(z_i)\right|\right] \\
&= \mathsf{Rad}_m(F).
\end{aligned}$$

By applying the same argument to $-f$ and then taking the union bound we see that

$$\left|\mathop{\mathsf{E}}_{z\sim P}[f(x)] - \mathop{\mathsf{E}}_{z\sim S}[f(z)]\right| \le \mathsf{Rad}_m(F) + \sqrt{\frac{1}{2m}\ln\frac{2}{\delta}}$$

holds with probability $1-\delta$. □

To apply the bound, we of course need to be able to estimate $\mathsf{Rad}_m(F)$.

Theorem 3.18: For a fixed $S \in Z^m$ we have

$$\mathrm{Rad}_m(F) \leq \widehat{\mathrm{Rad}}_m(F, S) + \sqrt{\frac{2}{m} \ln \frac{1}{\delta}}$$

with probability at least $1 - \delta$ over the random choice of S.

We also have

$$\mathrm{Rad}_m(F) \leq \sup_{f \in F} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f(z_i) \right| + \sqrt{\frac{8}{m} \ln \frac{1}{\delta}}$$

with probability at least $1 - \delta$ over the random choice of $\boldsymbol{r}$ and S.

This shows that instead of calculating expectations, we can just solve a single maximisation and have an upper bound with high confidence.

Proof: We prove the second estimate. The first one is proven similarly.

Let $q_i = (r_i, z_i) \in \{-1, 1\} \times Z$, and define

$$p(q_1, \ldots, q_m) = \sup_{f \in F} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f(z_i) \right|.$$

Since we assume $f(z_i) \in [0, 1]$, we can apply McDiarmid's inequality with $c_i = 4/m$. Solving for ε such that

$$\exp\left(-m\varepsilon^2/8\right) = \delta$$

gives

$$\varepsilon = \sqrt{\frac{8}{m} \ln \frac{1}{\delta}}.$$

□

Assume now that F is the 0-1 loss class for some class H of classifiers. In other words, $F = \{ \ell_h \mid h \in H \}$ where

$$\ell_h(x, y) = \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{if } y \neq f(x). \end{cases}$$

Recall that when F is the 0-1 loss class for H, the bound of Theorem 3.14 becomes

$$\sup_{h \in H} \left| R(h) - \widehat{R}(h) \right| \leq \mathrm{Rad}_m(F) + \sqrt{\frac{1}{2m} \ln \frac{2}{\delta}}.$$

We show how empirical risk minimisation can be used to estimate $\mathrm{Rad}_m(F)$.

Let $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times Y)^m$ and $\boldsymbol{r} \in \{-1, 1\}^m$ be fixed. Write $S' = ((x_1, r_1 y_1), \ldots, (x_m, r_m y_m))$. Since $f_h(x, y) = L_{0-1}(y, h(x)) = \frac{1}{2}(1 - yh(x))$, we have

$$\begin{aligned}
2\sum_{i=1}^{m} r_i f_h(x_i, y_i) &= \sum_{i=1}^{m} r_i - \sum_{i=1}^{m} r_i y_i h(x_i) \\
&= \sum_{i=1}^{m} r_i - \sum_{i=1}^{m} (1 - 2L_{0-1}(r_i y_i, h(x_i))) \\
&= \sum_{i=1}^{m} r_i + m - 2\sum_{i=1}^{m} (1 - L_{0-1}(r_i y_i, -h(x_i))) \\
&= \sum_{i=1}^{m} r_i + m - 2m\hat{R}'(-h)
\end{aligned}$$

where $\hat{R}'$ is empirical risk with respect to sample S'.

Similarly

$$-2\sum_{i=1}^{m} r_i f_h(x_i, y_i) = -\sum_{i=1}^{m} r_i + m - 2m\hat{R}'(h).$$

Write $\hat{\varepsilon} = \inf_{h\in H} \hat{R}'(h)$, which by our assumption means also $\hat{\varepsilon} = \inf_{h\in H} \hat{R}'(-h)$. We have

$$\sup_{h\in H} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f_h(x_i, y_i) \right| = 1 - 2\hat{\varepsilon} + \frac{1}{m} \left| \sum_{i=1}^{m} r_i \right|.$$

Thus, we get an estimate for $\mathrm{Rad}_m(F)$ by computing $\hat{\varepsilon}$ by empirical risk minimisation and plugging the above to Theorem 3.18.

Unfortunately, empirical risk minimisation is computationally intractable for most interesting classes H. Also in practice, the problem of evaluating $\mathrm{Rad}_m(F)$ seriously restricts the applicability of Rademacher complexities.

Rademacher complexities can be crudely estimated in terms of the VC dimension. This of course gives only an upper bound that is worst case with respect to P.

Theorem 3.19: Let F be the 0-1 loss class for H where $\mathrm{VCdim}(H) = d < \infty$, and let $m \geq d$. Then

$$\mathrm{Rad}_m(F) \leq 5\sqrt{\frac{d+1}{m}\left(\ln\frac{m}{d}+1\right)}.$$

Remark: A stronger result $\mathrm{Rad}_m(F) = O(\sqrt{d/m})$ is also known. Plugging that into Corollary 3.15 gives the sample complexity

$$m = O\left(\frac{1}{\varepsilon^2}\left(d + \ln\frac{1}{\delta}\right)\right).$$

Here we only give the proof for the weaker bound of Theorem 3.19 which still illustrates the connection between combinatorial and statistical parameters.

Proof of Theorem 3.19: For $0 < \delta \leq 1$, let

$$\rho_\delta = \sqrt{\frac{8}{m}\left(d\left(\ln\frac{m}{d}+1\right)+\ln\frac{2}{\delta}\right)}.$$

Write also $S = ((x_1,y_1),\ldots,(x_m,y_m))$ and

$$Q(S,\boldsymbol{r}) = \sup_{f\in F}\left|\frac{2}{m}\sum_{i=1}^{m} r_i f(x_i,y_i)\right|.$$

We show that with probability at least $1-\delta$ over random choice of $(x_i,y_i)\sim P^m$ and $r_i \in \{-1,1\}$, we have

$$Q(S,\boldsymbol{r}) \leq \rho_\delta.$$

Since always $Q(S,\boldsymbol{r}) \leq 2$, choosing $\delta = d/m$ will then give the desired bound

$$\begin{aligned}
\mathop{\mathrm{E}}_{S,\boldsymbol{r}}[Q(S,\boldsymbol{r})] &\leq (1-\delta)\rho_\delta + \delta\cdot 2 \\
&\leq \sqrt{\frac{8}{m}\left(d\left(\ln\frac{m}{d}+1\right)+\ln\frac{m}{d}+\ln 2\right)}+\frac{2d}{m} \\
&\leq 5\sqrt{\frac{d+1}{m}\left(\ln\frac{m}{d}+1\right)}.
\end{aligned}$$

Consider a fixed $S = ((x_1, y_1), \ldots, (x_m, y_m))$.

We call any sequence $\boldsymbol{\ell} \in \{0, 1\}^m$ a label sequence, and say that a label sequence $\boldsymbol{\ell}$ is valid if for some $h \in H$ we have $\ell_i = f_h(x_i, y_i)$.

Fix $\rho \geq 0$. Now $Q(S, \boldsymbol{r}) \geq \rho$ holds iff for some valid label sequence $\boldsymbol{\ell}$ we have

$$\left| \frac{2}{m} \sum_{i=1}^{m} r_i \ell_i \right| \geq \rho.$$

In this case we say that $\boldsymbol{\ell}$ covers $\boldsymbol{r}$.

Intutively, if $\boldsymbol{\ell}$ covers $\boldsymbol{r}$, then $\boldsymbol{\ell}$ "explains" why the class F gets a good correlation with that particular $\boldsymbol{r}$. We show that most $\boldsymbol{r}$ remain uncovered, so the expected correlation is low.

Clearly the number of different valid label sequences is the same as the number $\Pi_H(\{x_1, \ldots, x_m\})$ of different partitionings of $\{x_1, \ldots, x_m\}$ that can be realised by a classifier in H. By Sauer's lemma, this is at most

$$\Pi_H(m) \leq \left(\frac{\mathrm{e}m}{d}\right)^d.$$

Consider first a fixed label sequence $\boldsymbol{\ell}$, and let $I = \{\, i \in \{\, 1, \ldots, m \,\} \mid \ell_i = 1 \,\}$. Then

$$\Pr_{\boldsymbol{r} \in \{\, -1,1 \,\}^m} \left(\frac{2}{m} \sum_{i=1}^{m} r_i \ell_i \geq \rho \right) = \Pr_{\boldsymbol{r} \in \{\, -1,1 \,\}^m} \left(\sum_{i \in I} \frac{1 + r_i}{2} \geq \frac{|I|}{2} + \frac{m\rho}{4} \right).$$

Similarly

$$\Pr_{\boldsymbol{r} \in \{\, -1,1 \,\}^m} \left(\frac{2}{m} \sum_{i=1}^{m} r_i \ell_i \leq -\rho \right) = \Pr_{\boldsymbol{r} \in \{\, -1,1 \,\}^m} \left(\sum_{i \in I} \frac{1 + r_i}{2} \leq \frac{|I|}{2} - \frac{m\rho}{4} \right).$$

If $I = \emptyset$, then $\sum_i r_i \ell_i = 0$ for all $\boldsymbol{r}$. Otherwise the random variable $\sum_{i \in I} (1 + r_i)/2$ has binomial distribution with parameters $|I|$ and $1/2$. By Hoeffding's inequality, the probability of drawing $\boldsymbol{r}$ that is covered by $\boldsymbol{\ell}$ is

$$\begin{aligned} \Pr_{\boldsymbol{r} \in \{\, -1,1 \,\}^m} \left(\left| \frac{2}{m} \sum_{i=1}^{m} r_i \ell_i \right| \geq \rho \right) &= \Pr_{\boldsymbol{r} \in \{\, -1,1 \,\}^m} \left(\left| \frac{1}{|I|} \sum_{i \in I} \frac{1 + r_i}{2} - \frac{1}{2} \right| \geq \frac{m\rho}{4|I|} \right) \\ &\leq 2 \exp\left(-2|I| (m\rho/(4|I|))^2 \right) \\ &\leq 2 \exp\left(-m\rho^2/8 \right). \end{aligned}$$

Therefore, a single label sequence $\boldsymbol{\ell}$ covers at most a proportion $2\exp(-m\rho^2/8)$ of sequences $\boldsymbol{r} \in \{-1,1\}^m$.

Since there are at most $(\mathrm{e}m/d)^d$ valid sequences, the proportion of all covered sequences $\boldsymbol{r}$ is at most

$$2\left(\frac{em}{d}\right)^d \exp(-m\rho^2/8).$$

By choosing $\rho = \rho_\delta$ this becomes δ.

We have shown

$$\Pr_{\boldsymbol{r}\in\{-1,1\}^m}[Q(S,\boldsymbol{r}) \geq \rho_\delta] \leq \delta$$

for any fixed S. This implies

$$\Pr_{S\sim P^m}\Pr_{\boldsymbol{r}\in\{-1,1\}^m}[Q(S,\boldsymbol{r}) \geq \rho_\delta] \leq \delta.$$

(As usual, we ignore some measurability assumptions.) □

We now use the Rademacher theory to develop a margin-based risk bound for linear classifiers.

For $B > 0$, let $\mathcal{F}_B$ be the function class

$$\mathcal{F}_B = \{ (\boldsymbol{x}, y) \mapsto y\boldsymbol{w} \cdot \boldsymbol{x} \mid \|\boldsymbol{w}\|_2 \leq B \}.$$

Theorem 3.20: For any $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m))$ we have

$$\widehat{\mathrm{Rad}}_m(\mathcal{F}_B, S) \leq \frac{2B}{m} \sqrt{\sum_{i=1}^{m} \|\boldsymbol{x}_t\|_2^2}.$$

Notice that if $\|\boldsymbol{x}_t\|_2 \leq X$, the bound becomes $2BX/\sqrt{m}$. However, this does not yet give a loss bound, since

- functions $f \in \mathcal{F}_B$ are not bounded to $[0, 1]$, and
- $\mathcal{F}$ is not a loss class for any interesting L.

Proof: If $f(\boldsymbol{x}, y) = y\boldsymbol{w} \cdot \boldsymbol{x}$ and $\|\boldsymbol{w}\|_2 \leq B$, we have

$$\left|\sum_{i=1}^{m} r_i f(\boldsymbol{x}_i, y_i)\right| = \left|\sum_{i=1}^{m} r_i y_i \boldsymbol{w} \cdot \boldsymbol{x}_i\right| = \left|\boldsymbol{w} \cdot \left(\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right)\right| \leq B \left\|\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right\|_2$$

by Minkowski's inequality $|\boldsymbol{w} \cdot \boldsymbol{x}| \leq \|\boldsymbol{w}\|_2 \|\boldsymbol{x}\|_2$. Therefore

$$\begin{aligned}
\widehat{\mathrm{Rad}}_m(\mathcal{F}, S) &= \mathop{\mathsf{E}}_{\boldsymbol{r} \in \{-1,1\}^m} \left[\sup_{f \in \mathcal{F}_B} \left|\frac{2}{m} \sum_{i=1}^{m} r_i f(\boldsymbol{x}_i)\right|\right] \\
&\leq \frac{2B}{m} \mathop{\mathsf{E}}_{\boldsymbol{r} \in \{-1,1\}^m} \left[\left\|\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right\|_2\right] \\
&= \frac{2B}{m} \mathop{\mathsf{E}}_{\boldsymbol{r} \in \{-1,1\}^m} \left[\sqrt{\left(\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right) \cdot \left(\sum_{j=1}^{m} r_j y_j \boldsymbol{x}_j\right)}\right].
\end{aligned}$$

Since $\sqrt{\cdot}$ is concave, we can apply Jensen to get

$$\begin{aligned}
&\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sqrt{\left(\sum_{i=1}^m r_i y_i \boldsymbol{x}_i\right)\cdot\left(\sum_{j=1}^m r_j y_j \boldsymbol{x}_j\right)}\right] \\
&\qquad\leq \sqrt{\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\left(\sum_{i=1}^m r_i y_i \boldsymbol{x}_i\right)\cdot\left(\sum_{j=1}^m r_j y_j \boldsymbol{x}_j\right)\right]} \\
&\qquad= \sqrt{\sum_{i,j=1}^m \mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}[r_i r_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j]}.
\end{aligned}$$

Since $\mathsf{E}[r_i r_j] = \delta_{ij}$ and $y_i \in \{-1,1\}$, the claim follows. □

A function $f : \mathbb{R} \to \mathbb{R}$ has Lipschitz constant c if it satisfies

$$|f(x) - f(y)| \leq c\,|x - y|$$

for all $x, y \in \mathbb{R}$.

The class we are ultimately interested here is the discrete loss class for norm-bounded linear classifiers

$$T \circ \mathcal{F}_B = \{\, (\boldsymbol{x}, y) \mapsto T(f(\boldsymbol{x}, y)) \mid f \in \mathcal{F}_B \,\}$$

where $T(p) = 0$ if $p > 0$ and $T(p) = 1$ if $p \leq 0$.

For technical reasons we introduce for $\mu \geq 0$ the function A_μ with

$$A_\mu(p) = \begin{cases} 1 & \text{if } p \leq 0 \\ 1 - p/\mu & \text{if } 0 < p < \mu \\ 0 & \text{if } \mu \leq p. \end{cases}$$

Then A_μ has Lispschitz constant $1/\mu$, and $A_\mu(p) \geq T(p)$ for all p.

Theorem 3.21: If function G has Lipschitz constant c and $G(0) = 0$, then

$$\widehat{\mathrm{Rad}}_m(G \circ \mathcal{F}, S) \leq 2c\widehat{\mathrm{Rad}}_m(\mathcal{F}, S)$$

for all S. □

(The proof of this theorem is technical and we will not go into it.)

In particular, if we consider $G = A_\mu - 1$ to get $G(0) = 0$, we see that

$$\begin{aligned} \widehat{\mathrm{Rad}}_m((A_\mu - 1) \circ \mathcal{F}_B, S) &\leq \frac{2}{\mu}\widehat{\mathrm{Rad}}_m(\mathcal{F}_B, S) \\ &\leq \frac{4B}{\mu m}\sqrt{\sum_{i=1}^{m} \|\boldsymbol{x}_i\|_2^2}. \end{aligned}$$

Recall the hinge loss $L_\mu(\boldsymbol{w}, \boldsymbol{x}, y) = \max\{0, \mu - y\boldsymbol{w} \cdot \boldsymbol{x}\}$.

Theorem 3.22: Fix $\mu > 0$ and $B > 0$ and a probability measure P over $\mathbb{R}^d \times \{-1, 1\}$. With probability at least $1 - \delta$ over drawing $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \sim P^m$, we have

$$\begin{aligned} \Pr_{(x,y)\sim P}(\text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}) \neq y) &= \mathop{\mathsf{E}}_{(x,y)\sim P}[T(y\boldsymbol{w} \cdot \boldsymbol{x})] \\ &\leq \frac{1}{\mu m}\sum_{i=1}^{m} L_\mu(\boldsymbol{w}, \boldsymbol{x}_i, y_i) \\ &\quad + \frac{4B}{\mu m}\sqrt{\sum_{i=1}^{m}\|\boldsymbol{x}_i\|_2^2} + 3\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}} \end{aligned}$$

for all $\boldsymbol{w}$ with $\|\boldsymbol{w}\|_2 \leq B$.

Remarks:

- If $\boldsymbol{w}$ has margin μ, we have $L_\mu(\boldsymbol{w}, \boldsymbol{x}_i, y_i) = 0$ for all i, and the bound becomes

$$\frac{4B}{\mu m}\sqrt{\sum_{i=1}^{m} \|\boldsymbol{x}_i\|_2^2} + 3\sqrt{\frac{1}{2m} \ln \frac{4}{\delta}}.$$

- The bound depends only on empirical quantities, with no assumptions about P. If we are "lucky" and get a sample that allows a large margin, we also get a good bound.

- The bound can be kernelised. Suppose we use kernel k associated with feature map $\boldsymbol{\psi}$ and $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{\psi}(x_i)$ for some α_i. The relevant quantities in the bound become

$$\sum_{i=1}^{m} \|\boldsymbol{\psi}(x_i)\|_2^2 = \sum_{i=1}^{m} k(x_i, x_i) \qquad \text{and} \qquad \|\boldsymbol{w}\|_2^2 = \sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j).$$

Proof: Since $T - 1 \leq A_\mu - 1$, we have

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[T(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1] \leq \mathop{\mathsf{E}}_{(x,y)\sim P}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1].$$

From Theorem 3.14, we have

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1] \leq \mathop{\mathsf{E}}_{(x,y)\sim S}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1] + \mathsf{Rad}_m((A_\mu - 1)\circ\mathcal{F}_B) + \sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

for all $\boldsymbol{w}$ with probability $1 - \delta/2$. From Theorem 3.18, we have

$$\mathsf{Rad}_m((A_\mu - 1)\circ\mathcal{F}_B) \leq \widehat{\mathsf{Rad}}_m((A_\mu - 1)\circ\mathcal{F}_B, S) + 2\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

with probability $1 - \delta/2$.

Since $\mathsf{E}_{(x,y)\sim P}[-1] = \mathsf{E}_{(x,y)\sim S}[-1] = -1$, we get

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[T(y\boldsymbol{w}\cdot\boldsymbol{x})] \quad \leq \quad \mathop{\mathsf{E}}_{(x,y)\sim S}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x})] + \widehat{\mathsf{Rad}}_m((A_\mu - 1)\circ\mathcal{F}_B, S) + 3\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

with probability $1-\delta$. We notice that

$$\mu A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) \leq L_\mu(\boldsymbol{w},\boldsymbol{x},y)$$

and plug in the estimate for $\widehat{\mathsf{Rad}}_m((A_\mu - 1)\circ\mathcal{F}_B, S)$. □

Again, one should not expect this bound to give numerical estimates that are useful in practice. Getting practically useful bounds is still work in progress. Currently the so-called "PAC-Bayesian" approach seem one promising approach, but we will not go into it here.

However, margin bounds like this are currently the main explanation for why kernel based methods, such as support vector machines, work so well.

4. Final remarks

The theme of the course has been linear binary classification algorithms, both online and batch, and their analysis using margins.

Besides the personal preferences of the lecturer, this choice of theme from among the huge body of literature on supervised learning does have some additional motivation:

- linear methods have a sound theoretical foundation
- in particular combined with kernels they are quite useful in practice, too
- this approach provides some common ground for online learning, which is currently a very active research topic, and statistical (batch) learning, which is more common in applications.
- both the algorithms and their analysis are relatively simple (at least on the online side).

Important topics that would have fitted the theme but did not quite fit in:

Linear algorithms that do not kernelise. These include multiplicative online algorithms such as Winnow, and L_1 regularisation in batch algorithms. (SVM does a form of L_2 regularisation.) Such algorithm favour classifiers with a sparse weight vector.

Conversion of online algorithms into batch ones, in particular so that the online loss bound becomes a bound on the true risk.

Boosting, which is a general batch-learning method that can be analysed in terms of margins.

Some links will be provided on the course web page for those interested.

The emphasis on the course was on theory. In the programming exercises we used toy data to make it easy to illustrate the behaviour of the algorithms.

However, unless we mentioned otherwise, the techniques are applicable also in practical problems.

It should be noted that the literature on computational learning theory also includes work that is purely theoretical (unrealistic assumptions about the data, running times that are high-order polynomials etc.).

What we have learned

1. Basic tools and concepts

- the basics of online and statistical learning: how to interpret the theoretical bounds, deriving them for a finite hypothesis class
- linear classifiers, margins, kernels and their application in linear classification
- basic mathematical tools: Jensen, Hoeffding

2. Online learning

- proving relative loss bounds using potential functions
- the Aggregating Algorithm and its analysis
- the Perceptron Algorithm, its marginalised version and analysis

3. Batch learning

- convex optimisation, convex duality and its application to margins
- Vapnik-Chervonenkis dimension, Rademacher complexity and the related main results
- proofs for (some of) the main results
- applying Rademacher complexity to large margin classifiers

About the exam

You are in principle expected to know all the material in lecture notes and in the homework (but of course the exam will not have any programming tasks). However the following are not included (since they were skipped in class):

- the proof of Theorem 2.1, Example 2.2
- the proof of Theorem 3.19.

Possible types of questions include

- define/explain/describe some term/concept/algorithm
- basic mathematical problems: prove an inequality using Jensen; obtain the dual of a convex optimisation problem; . . .
- prove something: either apply some basic proof technique from lectures/homework, or reproduce a (perhaps more difficult) proof directly from the lectures/homework
- design an algorithm.

There will be no trick questions. You don't need to memorise numerical constants in formulas, or irrelevant technical details.

However there **are** technical details that are (in the lecturer's opinion) relevant. This includes in particular the proofs of the main results.

Please provide feedback using the link on the course home page!

The end