Machine Learning

Summary by Ari Feiglin (ari.feiglin@gmail.com)

Contents

1 Bayes Decision Theory

1.1 Formalizing the Theory

Let us imagine the following scenario: you, a computer science student, are preparing to leave your house for the first time in a while. Should you or should you not take your umbrella? If you take your umbrella and it doesn't rain then you've inconvenienced yourself, but if you don't and it rains then you'll end up getting wet. You look outside, see that clouds are grey, and decide to take your umbrella.

Here you are trying to decide between some possible actions, each with their own cost. These actions are based on the state of the world, of which you have some set of observations.

The problem can be formalized as follows, you have

(1) a set of world states: $\{\omega_i\}_{i\in I}$, which are disjoint and exhaustive: $\Omega = \bigcup_{i\in I}\omega_i$ (where Ω is the entire space).

(2) a set of observations: $S_n = \{x_1, \dots, x_n\}$.

(3) a probabilistic model: conditionals $\mathbb{P}(S_n \mid \omega)$ and priors $\mathbb{P}(\omega)$.

(4) a set of possible actions $A = \{\alpha_1, \dots, \alpha_k\}$.

(5) a set of cost functions $\Lambda = \{\lambda(\alpha_k \mid \omega_j)\}_{j,k}$.

So in our example above, we have two world states: raining and not raining, our observation is that the clouds are grey, we have some prior belief about the probabilities of each world state as well as the probability of observing our observation given a world state, our actions are to take and to not take an umbrella, each has an associated cost.

So suppose we've made an observation x, we'd like to compute the new probability of a world state ω given this observation. This can be done via Bayes's law:

$$\mathbb{P}(\omega \mid x) = \frac{\mathbb{P}(x \mid \omega)}{\mathbb{P}(x)} \cdot \mathbb{P}(\omega)$$

$\mathbb{P}(\omega \mid x)$ is called the *posterior*. But $\mathbb{P}(x)$ is not known, so how do we compute it? Using total probability:

$$\mathbb{P}(x) = \sum_{i\in I} \mathbb{P}(x \mid \omega_i)\,\mathbb{P}(\omega_i)$$

As we make more and more observations, we can employ Baye's law over and over to refine the posterior.

1.2 Minimizing Risk

Our goal in Bayes decision theory is to define a strategy $\alpha(S_n)$ which determines which action α to take so that it minimizes our expected costs, given observations S_n.

Definition 1.2.1

Given an action α, we define its **conditional risk** given an observation x to be

$$R(\alpha \mid x) = \sum_{i\in I} \lambda(\alpha \mid \omega_i)\,\mathbb{P}(\omega_i \mid x)$$

Where the posterior $\mathbb{P}(\omega_i \mid x)$ is computed as above using Bayes's law. If we define the random variable $\lambda(\alpha)$ to be $\lambda(\alpha \mid \omega)$ under the state ω, then this is just $\mathbb{E}[\lambda(\alpha) \mid x]$.

Example 1.2.2

Suppose our actions $\alpha_1, \dots, \alpha_k$ are to guess the world state $\omega_1, \dots, \omega_k$. The cost we pay is 1 if we are wrong and 0 if we are correct, meaning $\lambda(\alpha_k \mid \omega_j) = 1 - \delta_{kj}$ where $\delta_{kj} = 1$ when $k = j$ and 0 otherwise.

Then the conditional risk is

$$R(\alpha_k \mid x) = \sum_{j=1}^{k} \lambda(\alpha_k \mid \omega_j)\,\mathbb{P}(\omega_j \mid x) = \sum_{j \neq k} \mathbb{P}(\omega_j \mid x) = 1 - \mathbb{P}(\omega_k \mid x)$$

So we minimize the conditional risk when we take α_k such that the posterior $\mathbb{P}(\omega_k \mid x)$ is maximal.

Example 1.2.3

Suppose we have two world states ω_1, ω_2 and two actions α_1, α_2. Then our costs form a 2×2 matrix: $\lambda_{kj} = \lambda(\alpha_k \mid \omega_j)$. And by definition

$$R(\alpha_i \mid x) = \lambda_{i1}\,\mathbb{P}(\omega_1 \mid x) + \lambda_{i2}\,\mathbb{P}(\omega_2 \mid x)$$

We choose α_1 if $R(\alpha_1 \mid x) < R(\alpha_2 \mid x)$, which is equivalent to

$$(\lambda_{12} - \lambda_{22})\,\mathbb{P}(\omega_2 \mid x) < (\lambda_{21} - \lambda_{11})\,\mathbb{P}(\omega_1 \mid x)$$

which is equivalent to

$$\iff \frac{\mathbb{P}(\omega_1 \mid x)}{\mathbb{P}(\omega_2 \mid x)} > \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}} \iff \frac{\mathbb{P}(x \mid \omega_1)}{\mathbb{P}(x \mid \omega_2)} > \frac{\mathbb{P}(\omega_2)}{\mathbb{P}(\omega_1)} \cdot \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}}$$

The left-hand side is called the **likelihood ratio** and the right-hand side is called the **decision boundary**.

If we have many observations $x_1, \dots, x_n$ then this becomes

$$\overset{\text{Likelihood ratio}}{\frac{\mathbb{P}(x_1, \dots, x_n \mid \omega_1)}{\mathbb{P}(x_1, \dots, x_n \mid \omega_2)}} > \overset{\text{Decision boundary}}{\frac{\mathbb{P}(\omega_2)}{\mathbb{P}(\omega_1)} \cdot \frac{\lambda_{22} - \lambda_{12}}{\lambda_{11} - \lambda_{21}}} = \Theta$$

If the observations $x_1, \dots, x_n$ are independent then this becomes

$$\frac{\prod_i \mathbb{P}(x_i \mid \omega_1)}{\prod_i \mathbb{P}(x_i \mid \omega_2)} > \Theta$$

taking the log of both sides gives

$$\sum_i \log \frac{\mathbb{P}(x_i \mid \omega_1)}{\mathbb{P}(x_i \mid \omega_2)} > \log \Theta = \Theta'$$

the left-hand side is called the **log-likelihood ratio**.

1.3 Parameter Estimation

Suppose we know the distribution of some random variable X up to some parameter θ, i.e. we know the function $\mathbb{P}(X \mid \theta)$. For example, X is the number of heads in n coin tosses where the coin has a bias of θ, then $X \mid \theta \sim \mathrm{Bin}(n, \theta)$.

We use Bayes decision theory to estimate θ. Suppose we have a prior $\mathbb{P}(\theta)$, we use this to estimate θ. Our actions will be choosing some prediction of θ, $\hat{\theta}$. Define a cost function $\lambda(\hat{\theta}, \theta)$. Then given a sequence of observations $S_n = \{x_1, \dots, x_n\}$, we want to minimize the expected cost

$$\mathbb{E}[\lambda(\hat{\theta}) \mid S_n] = \int \lambda(\hat{\theta}, \theta)\,\mathbb{P}(\theta \mid S_n)\,d\theta$$

We then define the *Bayes estimator* to be the prediction θ^* which minimizes this:

$$\theta^* = \mathrm{argmin}_{\hat{\theta}}\,\mathbb{E}[\lambda(\hat{\theta}) \mid S_n]$$

To find θ^* we differentiate $\mathbb{E}[\lambda(\hat{\theta}) \mid S_n]$ and compare it with zero. Hand-waving away all the technical details because this is computer science, we can swap the order of differentiation and integration and so

$$\frac{d}{d\hat{\theta}} \mathbb{E}[\lambda(\hat{\theta}) \mid S_n] = \frac{d}{d\hat{\theta}} \int \lambda(\hat{\theta}, \theta) \, \mathbb{P}(\theta \mid S_n) \, d\theta = \int \frac{d}{d\hat{\theta}} \lambda(\hat{\theta}, \theta) \, \mathbb{P}(\theta \mid S_n) \, d\theta$$

Example 1.3.1 (Square Loss)

Suppose we use a **square loss** cost function: $\lambda(\hat{\theta}, \theta) = (\theta - \hat{\theta})^2$. Then the Bayes estimator is

$$\mathbb{E}[\lambda(\hat{\theta})] = \int \lambda(\hat{\theta}, \theta) \, \mathbb{P}(\theta \mid S_n) \, d\theta = \int (\theta - \hat{\theta})^2 \, \mathbb{P}(\theta \mid S_n) \, d\theta$$

Differentiating gives that we want

$$\int \theta \, \mathbb{P}(\theta \mid S_n) \, d\theta = \hat{\theta} \int \mathbb{P}(\theta \mid S_n) \, d\theta = \hat{\theta}$$

(since $\int \mathbb{P}(\theta \mid S_n) = 1$.) So we get that our estimator is

$$\hat{\theta}_{SE} = \int \theta \, \mathbb{P}(\theta \mid S_n) \, d\theta = \mathbb{E}[\theta \mid S_n]$$

Example 1.3.2 (Maximum Aposteriori)

Suppose we use a **zero-one** cost function: $\lambda(\hat{\theta}, \theta) = 1 - \delta_{\hat{\theta}\theta}$. We have already showed that to minimize the cost, we must maximize $\operatorname{argmax} \mathbb{P}(\theta \mid S_n)$. By Bayes, this is just $\operatorname{argmax} \mathbb{P}(S_n \mid \theta) \frac{\mathbb{P}(\theta)}{\mathbb{P}(S_n)} = \operatorname{argmax} \mathbb{P}(S_n \mid \theta) \, \mathbb{P}(\theta)$. Since the observations in S_n are independent, we see that this is then just equal to $\operatorname{argmax} \prod_i \mathbb{P}(x_i \mid \theta) \, \mathbb{P}(\theta)$. Finally we take the log, which is monotonic, to get that the estimator

$$\hat{\theta}_{MAP} = \operatorname{argmax} \sum_i \log \mathbb{P}(x_i \mid \theta) + \log \mathbb{P}(\theta)$$

as n grows, the last term becomes negligible and this just becomes the maximum log likelihood function.

Exercise 1.3.3

Compute the MAP estimator for $X \sim \operatorname{Exp}(\theta)$, i.e. $f_X(x) = \theta \exp(-\theta x)$, with the prior $\mathbb{P}(\theta) = \exp(-\theta)$. We want to compute

$$\sum_i \log \mathbb{P}(x_i \mid \theta) + \log \mathbb{P}(\theta) = \sum_i \log\bigl(\theta \exp(-\theta x_i)\bigr) + \log \exp(-\theta) = n \log \theta - \theta \sum_i x_i - \theta$$

Differentiating with respect to θ gives

$$\frac{n}{\theta} - \sum_i x_i - 1 = 0 \implies \hat{\theta}_{MAP} = \frac{n}{\sum_i x_i + 1}$$

Here the prior behaves similarly to a sample, it can be viewed as a “pseudo-sample”.

Example 1.3.4 (Maximum Likelihood)

We know that

$$\hat{\theta}_{MAP} = \operatorname{argmax} \sum_i \log \mathbb{P}(x_i \mid \theta) + \log \mathbb{P}(\theta)$$

A common approach is to approximate this by dropping the prior, giving

$$\hat{\theta}_{ML} = \operatorname{argmax} \sum_i \log \mathbb{P}(x_i \mid \theta)$$

Asymptotically this is efficient.

2 PAC Learning

2.1 Definitions

We now define *Probably Approximately Correct (PAC)* Learning.

Definition 2.1.1

Let $\mathcal{X}$ be the **input space**, the set of all possible examples or instances. The set of all **labels** or **target values** is $\mathcal{Y}$. For now, we restrict our view to be binary: $\mathcal{Y} = \{0, 1\}$. A **concept** is a map $c\colon \mathcal{X} \longrightarrow \mathcal{Y}$, or equivalently a subset of $\mathcal{X}$ (the set $\{x \in \mathcal{X} \mid c(x) = 1\}$). A **concept class** is a class of concepts which we would like to learn (approximate) and is denoted $\mathcal{C}$.

The idea of PAC learning is as follows: the learner considers a fixed set of concepts $\mathcal{H}$ called the *hypothesis set*, which may or may not coincide with $\mathcal{C}$. The learner then receives a sequence of samples $S = (x_1, \dots, x_n)$ which are independent and distribute according to some distribution $\mathcal{D}$. The learner also receives labels $(c(x_1), \dots, c(x_n))$ according to some concept $c \in \mathcal{C}$ which it is tasked with learning. Using this information the learner attempts to choose a hypothesis $h_S \in \mathcal{H}$ which minimizes the *generalization error* (or *risk*):

Definition 2.1.2

Given a hypothesis $h \in \mathcal{H}$, target concept $c \in \mathcal{C}$, and an underlying distribution $\mathcal{D}$, the **generalized error** (or **risk**) is:

$$R(h) = \mathbb{P}(h(x) \neq c(x) \mid x \sim \mathcal{D}) = \mathbb{E}[\mathbf{1}\{h(x) \neq c(x)\} \mid x \sim \mathcal{D}]$$

i.e. it is the probability that $h(x)$ differs from $c(x)$ when x is chosen randomly with a distribution of $\mathcal{D}$.

But the generalized error cannot be known to the learner, as it knows not the target concept nor the underlying distribution. So instead the learner minimizes the *empirical error* (or *risk*):

Definition 2.1.3

Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and a sample $S = (x_1, \dots, x_n)$, define the **empirical error** (or **risk**) to be:

$$\widehat{R}_S(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{h(x_i) \neq c(x_i)\}$$

Notice that

$$\begin{aligned}\mathbb{E}[\widehat{R}_S(h) \mid S \sim \mathcal{D}^n] &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[\mathbf{1}\{h(x_i) \neq c(x_i)\} \mid S \sim \mathcal{D}^n] \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[\mathbf{1}\{h(x) \neq c(x)\} \mid x \sim \mathcal{D}] = \frac{1}{n}\sum_{i=1}^{n} R(h) = R(h)\end{aligned}$$

We now formally define what PAC learning is. Let n be a number such that the size of every $x \in \mathcal{X}$ can be represented in $O(n)$ space, for $c \in \mathcal{C}$ let *size* c be the maximal computational cost of c. We focus on algorithms $\mathcal{A}$ which take as input a sample S and return a hypothesis h_S.

Definition 2.1.4

A concept class $\mathcal{C}$ is **PAC-learnable** if there exists an algorithm $\mathcal{A}$ and a polynomial function $\mathit{poly}(\bullet, \bullet, \bullet, \bullet)$ such that for all $\varepsilon, \delta > 0$, distribution $\mathcal{D}$ on $\mathcal{X}$ and target concept $c \in \mathcal{C}$, for every sample size $n \geq \mathit{poly}(1/\varepsilon, 1/\delta, n, \mathit{size}\, c)$,

$$\mathbb{P}(R(h_S) \leq \varepsilon \mid S \sim \mathcal{D}^n) \geq 1 - \delta$$

If $\mathcal{A}$ runs in $\mathit{poly}(1/\varepsilon, 1/\delta, n, \mathit{size}\, c)$ time then $\mathcal{C}$ is **efficiently PAC-learnable**. An algorithm $\mathcal{A}$, if one exists, is called a **PAC-learning algorithm** for $\mathcal{C}$.

The intuition is as follows: a concept class $\mathcal{C}$ is PAC-learnable if there exists an algorithm $\mathcal{A}$ where given a sample size at least polynomial in $1/\varepsilon$ and $1/\delta$, it returns a hypothesis with an error bound by ε at least $1-\delta$ of the time. Note that if the running time is polynomial in $1/\varepsilon$ and $1/\delta$, then assuming the total input is read by the algorithm, the input must too be polynomial in $1/\varepsilon$ and $1/\delta$.

2.2 The Finite Case

Given a concept $c \in \mathcal{C}$ and a sample $S = (x_1, \dots, x_n)$, call an hypothesis $h \in \mathcal{H}$ *consistent* if $h(x_i) = c(x_i)$ for all $1 \le i \le n$. Equivalently, $\widehat{R}_S(h) = 0$. We will assume that our hypotheses are consistent, and so we can always assume that our target concept is in $\mathcal{H}$.

Theorem 2.2.1

Let $\mathcal{H}$ be a finite set of hypotheses, and $\mathcal{A}$ an algorithm such that for any target concept $c \in \mathcal{H}$, $\mathcal{A}$ returns a consistent hypothesis h_S for any input sample S (where $S \sim \mathcal{D}^n$). Then for every every $\varepsilon, \delta > 0$, the inequality $\mathbb{P}(R(h_S) \le \varepsilon \mid S \sim \mathcal{D}^n) \ge 1 - \delta$ holds if

$$n \ge \frac{1}{\varepsilon} \cdot \log \frac{|\mathcal{H}|}{\delta}$$

Proof: let $\varepsilon > 0$, and define $\mathcal{H}_\varepsilon = \{h \in \mathcal{H} \mid R(h) > \varepsilon\}$. The probability that $h \in \mathcal{H}_\varepsilon$ is consistent is

$$\begin{aligned}\mathbb{P}(\widehat{R}_S(h) = 0) &= \mathbb{P}(h(x_1) = c(x_1), \dots, h(x_n) = c(x_n) \mid x_i \sim \mathcal{D}) \\ &= \prod_{i=1}^{n} \mathbb{P}(h(x_i) = c(x_i) \mid x_i \sim \mathcal{D}) = \mathbb{P}(h(x) = c(x) \mid x \sim \mathcal{D})^n = \big(1 - \mathbb{P}(h(x) \ne c(x) \mid x \sim \mathcal{D})\big)^n\end{aligned}$$

this is since the samples are independent and distributively equal. Now recall that by definition, we have that $\mathbb{P}(h(x) \ne c(x) \mid x \sim \mathcal{D}) = R(h)$ which is greater than ε by definition, so

$$\mathbb{P}(\widehat{R}_S(h) = 0) \le (1 - \varepsilon)^n$$

Thus the probability that a "bad" hypothesis tricks us by being consistent is bound by $(1-\varepsilon)^n$. Now, we want to show that the probability that there exists a bad consistent hypothesis is bound by δ. This means that with probability at least $1-\delta$, there exist no bad consistent hypotheses and so h_S is necessarily a "good" hypothesis with $R(h_S) \le \varepsilon$. The probability of there existing a consistent bad hypothesis is

$$\mathbb{P}(\exists h \in \mathcal{H}_\varepsilon.\ \widehat{R}_S(h) = 0) \le \sum_{h \in \mathcal{H}_\varepsilon} \mathbb{P}(\widehat{R}_S(h) = 0) \le \sum_{h \in \mathcal{H}_\varepsilon} (1-\varepsilon)^n \le |\mathcal{H}|(1-\varepsilon)^n$$

We also know that $(1-\varepsilon)^n \le e^{-n\varepsilon}$, so this probability is bound by $|\mathcal{H}|e^{-n\varepsilon}$. If $n \ge \frac{1}{\varepsilon} \cdot \log \frac{|\mathcal{H}|}{\delta}$ then this is bound by δ, as required. ■

Since $\frac{1}{\varepsilon} \cdot \log \frac{|\mathcal{H}|}{\delta}$ is bound by some polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}$, this means that when $\mathcal{H}$ is finite a consistent algorithm $\mathcal{A}$ is PAC-learning. But what if our algorithm isn't consistent, i.e. the target concept is not in $\mathcal{H}$?

Theorem 2.2.2

For every $\varepsilon, \delta > 0$ if $n > \frac{1}{\varepsilon^2} \log \frac{2|\mathcal{H}|}{\delta}$ then

$$\mathbb{P}\left(R(h_S) - \min_{h \in \mathcal{H}} R(h) \le \varepsilon\right) \ge 1 - \delta$$

Proof: similar to before, but using Hoeffding's inequality $\mathbb{P}(S - \mathbb{E}[S] \ge \varepsilon) \le \exp(-2\varepsilon^2/n)$. ■

This is a generalization of the previous theorem, since if c is taken from $\mathcal{H}$ then $\min_{h \in \mathcal{H}} R(h) = 0$. Call a concept class $\mathcal{C}$ which satisfies this probability bound *Agonistically PAC-learnable*:

Definition 2.2.3

A concept class $\mathcal{C}$ is **agonistically PAC-learnable** if there exists an **agnostic PAC-learner** $\mathcal{A}$ such that for every $\varepsilon, \delta > 0$ distribution $\mathcal{D}$ and $n \geq poly(1/\varepsilon, 1/\delta, n, size\, c)$ for some $poly$,

$$\mathbb{P}\left(R(h_S) - \min_{h \in \mathcal{H}} R(h) \leq \varepsilon\right) \geq 1 - \delta$$

So assuming the hypothesis class is finite, every concept class is PAC-learnable in the agnostic sense.

2.3 The Infinite Case

Suppose now that $\mathcal{H}$ is infinite, and we don't necessarily have that the target concept is in $\mathcal{H}$. Define

$$h_S := \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_S(h), \qquad h_{\mathcal{D}} := \operatorname*{argmin}_{h \in \mathcal{H}} R_{\mathcal{D}}(h)$$

Then we have

Lemma 2.3.1

$$R_{\mathcal{D}}(h_S) - R_{\mathcal{D}}(h_{\mathcal{D}}) \leq 2 \sup_{h \in \mathcal{H}} |\widehat{R}_S(h) - R_{\mathcal{D}}(h)|$$

Proof: let $\varepsilon = \sup_{h \in \mathcal{H}} |\widehat{R}_S(h) - R_{\mathcal{D}}(h)|$, then

$$\begin{aligned} R_{\mathcal{D}}(h_S) &\leq \widehat{R}_S(h_S) + \varepsilon && \text{since } h_S \in \mathcal{H} \\ &\leq \widehat{R}_S(h_{\mathcal{D}}) + \varepsilon && \text{since } h_S \text{ minimizes } \widehat{R}_S \\ &\leq R_{\mathcal{D}}(h_{\mathcal{D}}) + 2\varepsilon && \text{since } h_{\mathcal{D}} \in \mathcal{H} \end{aligned}$$

■

Definition 2.3.2

A hypothesis class $\mathcal{H}$ **shatters** a finite set $C = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ if for any labels $y_1, \ldots, y_n \in \{0, 1\}$ there exists an hypothesis $h \in \mathcal{H}$ such that $h(x_i) = y_i$ for all i.

In other words, $\mathcal{H}$ shatters C if for every function $f\colon C \longrightarrow \mathcal{Y}$, there exists an hypothesis h such that $h\big|_C = f$. So if we define $\mathcal{H}_C = \left\{h\big|_C \;\middle|\; h \in \mathcal{H}\right\}$, then $\mathcal{H}$ shatters C if and only if $|\mathcal{H}_C| = 2^{|C|}$.

Definition 2.3.3

The **VC dimension** of an hypothesis class $\mathcal{H}$, denoted $\mathrm{VCdim}(\mathcal{H})$, is the size of the largest set $C \subseteq \mathcal{X}$ shattered by $\mathcal{H}$.

Theorem 2.3.4

Let $\mathcal{H}$ be an hypothesis class with VC dimension $d < \infty$. Then there exists a constant $c > 0$ such that for every $\varepsilon, \delta \in (0, 1)$ there is a polynomial such that for every $n \geq poly(1/\varepsilon, 1/\delta)$ for which

$$\mathbb{P}\left(\sup_{h \in \mathcal{H}} |\widehat{R}_S(h) - R_{\mathcal{D}}(h)| < c\sqrt{\frac{d}{n}} \;\middle|\; S \sim \mathcal{D}^n\right) \geq 1 - \delta$$

The proof of this is beyond the scope of the class.

3 Linear Regression

3.1 Linear Regression

Suppose our input space $\mathcal{X}$ is a subset of $\mathbb{R}^d$ for some d, and $\mathcal{Y}$ is $\mathbb{R}$. Given a sample $\{(\vec{x}_i, y_i)\}_{1\le i\le n}$, we would like to find the best linear approximation $h\colon \mathbb{R}^d \longrightarrow \mathbb{R}$ which approximates the relation between the inputs and their labels. Thus the hypothesis class is then the set of all linear functions

$$\mathcal{H} = \left\{ f(\vec{x}) = \vec{x}^\top \vec{w} + b \;\middle|\; \vec{w} \in \mathbb{R}^d, b \in \mathbb{R} \right\}$$

For a hypothesis f, let us denote $\hat{y}_i = f(x_i) = \vec{x}_i^\top \vec{w} + b$, notice that

$$\begin{pmatrix} \text{------} & \vec{x}_1 & \text{------} & 1 \\ & \vdots & & \vdots \\ \text{------} & \vec{x}_n & \text{------} & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_d \\ b \end{pmatrix} = \begin{pmatrix} \vec{x}_1^\top \vec{w} + b \\ \vdots \\ \vec{x}_n^\top \vec{w} + b \end{pmatrix}$$

To simplify notation, we can just assume that the final coordinate of each $\vec{x}_i$ is 1 and so we can view each hypothesis f as its vector $\vec{w}$ (where the last coefficient is b). Thus

$$X\vec{w} = \hat{y}$$

Where X is the matrix whose rows are $\vec{x}_i$.

We want to minimize the difference between $\hat{y}$ and $\vec{y}$. We can do this by taking the mean square error (MSE): $\frac{1}{n}\|\hat{y} - \vec{y}\|^2 = \frac{1}{n}\|X\vec{w} - \vec{y}\|^2$. So we define

$$\mathrm{MSE}(\vec{w}) = \frac{1}{n}\|X\vec{w} - \vec{y}\|^2 = \frac{1}{n}(X\vec{w} - \vec{y})^\top (X\vec{w} - \vec{y})$$

We take the gradient and compare with zero:

$$\nabla \mathrm{MSE}(\vec{w}) = \frac{2}{n} X^\top (X\vec{w} - \vec{y})$$

comparing with zero gives

$$\nabla \mathrm{MSE}(\vec{w}) = 0 \iff X^\top X \vec{w} - X^\top \vec{y} = 0 \iff \vec{w}_{\min} = (X^\top X)^{-1} X^\top \vec{y}$$

This must be a minimum since MSE is quadratic in $\vec{w}$.

So now if we have new data X_{new}, our estimator for the new values will be

$$\hat{y}_{\text{new}} = X_{\text{new}} \vec{w}_{\min} = X_{\text{new}} (X^\top X)^{-1} X^\top \vec{y}$$

Furthermore, notice that the error $\vec{y} - \hat{y}$ is orthogonal to $\hat{y}$:

$$\begin{aligned} (\vec{y} - \hat{y})^\top \hat{y} = (\vec{y} - X\vec{w})^\top \hat{y} &= \vec{y}^\top \hat{y} - \vec{w}^\top X^\top X \vec{w} \\ &= \vec{y}^\top X\vec{w} - \vec{w}^\top X^\top X (X^\top X)^{-1} X^\top \vec{y} \\ &= \vec{y}^\top X \vec{w} - \vec{w}^\top X^\top \vec{y} \\ &= \vec{y}^\top X\vec{w} - (X\vec{w})^\top \vec{y} = 0 \end{aligned}$$

3.2 Polynomial Fitting

Suppose we have n points $\{(x_i, y_i)\}_{1\le i\le n}$ which we would like to approximate using a k-degree polynomial $\hat{y} = p(x) = w_0 x^0 + \cdots + w_k x^k$. We can do so using linear regression: define

$$X = \begin{pmatrix} x_1^0 & x_1^1 & \cdots & x_1^k \\ \vdots & \vdots & & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^k \end{pmatrix}$$

Then doing linear regression gets us the best vector $\vec{w}$ such that

$$X\vec{w} = \begin{pmatrix} \sum_{i=0}^{k} w_i x_1^i \\ \vdots \\ \sum_{i=0}^{k} w_i x_n^i \end{pmatrix} = \begin{pmatrix} p(x_1) \\ \vdots \\ p(x_n) \end{pmatrix}$$

is closest to $\vec{y}$, as required.

3.3 Ridge Regression

What if $X^\top X$ is not invertible? Which can happen if we have too few samples, i.e. $n < d$. Notice that $X^\top X$ is positive semi-definite: $v^\top X^\top X v = (Xv)^\top (Xv) \geq 0$, and thus all its eigenvalues are nonnegative. Then for any $\lambda > 0$, $(X^\top X + \lambda I)v = \mu v$ if and only if $X^\top X v = (\mu - \lambda)v$ which must mean that $\mu - \lambda \geq 0$ and in particular $\mu > 0$. So all of $X^\top X + \lambda I$'s eigenvalues are positive, and in particular non-zero, meaning it is invertible.

So using this knowledge, let us define the *ridge estimator* to be

$$\widehat{w}_{\text{ridge}} = \underset{\vec{w}}{\operatorname{argmin}} \|X\vec{w} - \vec{y}\|^2 + \lambda \|\vec{w}\|^2$$

Taking the gradient and equating to zero gives that

$$\widehat{w}_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y$$

which, as explained above, exists no matter what, for any $\lambda > 0$.

4 Binary Classification

Suppose we'd like to learn the class of halfspaces: $\mathcal{X} = \mathbb{R}^d, \mathcal{Y} = \{-1, 1\}$ and the hypothesis class is

$$\mathcal{H}S_d = \left\{\mathbf{x} \mapsto \operatorname{sign}(\langle \mathbf{w}, \mathbf{x}\rangle + b) \;\middle|\; \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\right\}$$

The idea here is that we split our space by the hyperplane $\langle \mathbf{w}, \mathbf{x}\rangle = -b$, and we classify elements as being either above or below the hyperplane depending on the sign of $\langle \mathbf{w}, \mathbf{x}\rangle + b$. Let us assume the realizable case, then we'd like to learn this hypothesis class.

Notice that

$$\langle \mathbf{w}, \mathbf{x}\rangle + b = \langle (b, \mathbf{w}), (1, \mathbf{x})\rangle$$

so we can assume that $b = 0$ and the input $\mathbf{x}$'s first index is 1.

4.1 The Perceptron Algorithm

```
1.  function PERCEPTRON($(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$)
2.      $\mathbf{w}^{(1)} \leftarrow (0, \dots, 0)$
3.      for ($t = 1, 2, 3, \dots$)
4.          if (There exists an $i$ such that $y_i\langle \mathbf{w}^{(t)}, \mathbf{x}_i\rangle \le 0$)
5.              $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i\mathbf{x}_i$
6.          else
7.              return $\mathbf{w}^{(t)}$
8.          end if
9.      end for
10. end function
```

Notice that the check $y_i\langle \mathbf{w}^{(t)}, \mathbf{x}\rangle \le 0$ is the same as $\operatorname{sign}\langle \mathbf{w}^{(t)}, \mathbf{x}\rangle \neq y_i$. So this algorithm checks if the current $\mathbf{w}^{(t)}$ mislabels any input, and if so it alters $\mathbf{w}^{(t)}$ to fix this. Notice that $w^{(t+1)} = \mathbf{w}^{(t)} + y_i\mathbf{x}_i$ will increase $y_i\langle \mathbf{w}, \mathbf{x}_i\rangle$:

$$y_i\left\langle \mathbf{w}^{(t+1)}, \mathbf{x}_i\right\rangle = y_i\left\langle \mathbf{w}^{(t)} + y_i\mathbf{x}_i, \mathbf{x}_i\right\rangle = y_i\left\langle \mathbf{w}^{(t)}, \mathbf{x}_i\right\rangle + \|\mathbf{x}_i\|^2$$

so this will make the current hypothesis hopefully closer to objective concept.

Indeed this algorithm does terminate, but the proof of this is not relevant to this course.

4.2 Gradient Descent

Suppose we want to minimize some loss function $\ell(\mathbf{w})$. Since $\nabla\ell$ is the direction in which the loss function increases the most, if we go in the opposite direction we hope that this decreases the function. So if we start with an initial guess for the minimum point of the loss function $\mathbf{w}^{(1)}$, then at each iteration we just compute

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta\nabla\ell(\mathbf{w}^{(t)})$$

where $\eta > 0$ is some constant. This iterative procedure is called *gradient descent.* Assuming ℓ is convex and $\nabla\ell$ is Lipschitz-continuous, and $0 < \eta < 1/L$ where L is the Lipschitz constant for $\nabla\ell$, then gradient descent is ensured to converge.

4.3 Stochastic Gradient Descent

When we have a collection of samples $\{(x_i, y_i)\}_{i=1}^n$, then we want to minimize $Err(w) = \sum_{i=1}^n \ell(w; x_i, y_i)$ (i.e. ℓ is parameterized, and we want to minimize the mean over the samples). Then our iteration becomes

$$w^{(t+1)} \leftarrow w^{(t)} - \eta\nabla Err(w^{(t)})$$

But computing the gradient of n parameterizations is costly, so we can approximate this by defining

$$Batch\text{-}Err(w) = \sum_{i=1}^{b} \ell(w; x_i, y_i)$$

where $b \ll n$ and the parameters x_i, y_i are chosen randomly from the set of samples (the samples are chosen uniformly at each iteration). When $b = 1$ this is called *pure stochastic gradient descent.*

4.4 Stochastic Gradient Descent Learning

Let us return to the problem of learning halfspaces. Recall that our samples are of the form $\{(\mathbf{x}_i, y_i)\}$ where $y_i = \pm 1$ depending on whether or not $\mathbf{x}_i$ is in or out of the halfspace. The perceptron method works for the realizable case, but in the case that there is some noise in the sampling, it will not work. So let us use SGD to learn it, to do so we take a good loss function, for example the *hinge loss function*:

$$\ell_{hinge}(w; x, y) = \begin{cases} -yw^\top x & yw^\top x < 0 \\ 0 & \text{else} \end{cases}$$

Then the total loss function is

$$Err(w) = \sum_{i=1}^{n} \ell_{hinge}(w; \mathbf{x}_i, y_i)$$

We compute:

$$\nabla \ell_{hinge}(w; x, y) = \begin{cases} -yx & yw^\top x < 0 \\ 0 & \text{else} \end{cases}$$

So using pure SGD we get the following algorithm for learning halfspaces:

```
1. function SGD-Learner($(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n), \eta$)
2.     $\mathbf{w}^{(1)} \leftarrow (0, \ldots, 0)$
3.     while (There exists an $i$ such that $y_i(\mathbf{w}^{(t)})^\top x_i < 0$)
4.         $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta y_i \mathbf{x}_i$
5.     end while
6. end function
```

4.5 Logistic Regression

The issue with the previous methods is that they check $\text{sign}(\mathbf{w}^\top \mathbf{x}_i)$, which is binary and therefore very rigid. Instead, we can define the hypothesis class

$$\mathcal{H}_{\text{sig}} = \left\{ \mathbf{x} \mapsto \sigma(\langle \mathbf{w}, \mathbf{x} \rangle) \;\middle|\; \mathbf{w} \in \mathbb{R}^d \right\}$$

where σ is the *sigmoid function*:

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

The shape of this graph looks a lot like $f(x) = 0$ for $x \ll 0$ and $f(x) = 1$ for $x \gg 0$. For a hypothesis $h_\mathbf{w}$, we can view $h_\mathbf{w}(\mathbf{x})$ as the probability that the label of $\mathbf{x}$ is 1.

We want to define a loss function which should determine how bad it is to predict $h_\mathbf{w}(\mathbf{x}) \in [0, 1]$ given that the true label is $y = \pm 1$. Suppose we have a sample (x, y) then our model outputs $\mathbb{P}(y = 1 \mid x)$, and for a given weight $\mathbf{w}$, this is $\mathbb{P}(y = 1 \mid x) = \sigma(\mathbf{w}^\top \mathbf{x})$. For a given set of independent samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we want to maximize

$$\mathcal{L}(\mathbf{w}) = \prod_{y_i = 1} \mathbb{P}(y = 1 \mid \mathbf{x}_i) \cdot \prod_{y_i = 0} \mathbb{P}(y = 0 \mid \mathbf{x}_i)$$

Since $x^0 = x$ for all x, we can write this as

$$= \prod_{i=1}^{n} \mathbb{P}(y = 1 \mid \mathbf{x}_i)^{y_i} \cdot \prod_{i=1}^{n} \mathbb{P}(y = 0 \mid \mathbf{x}_i)^{1-y_i} = \prod_{i=1}^{n} \sigma(\mathbf{w}^\top \mathbf{x}_i)^{y_i} \cdot \prod_{i=1}^{n} \big(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)\big)^{1-y_i}$$

Maximizing this is the same as minimizing $-\log \mathcal{L}(\mathbf{w})$ (because we can then use gradient descent to find the minimum). And so our total loss function is

$$Err(\mathbf{w}) = -\log \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + \sum_{i=1}^{n} (1 - y_i) \log\big(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)\big)$$

Let us write σ for $\sigma(\mathbf{w}^\top \mathbf{x}_i)$, then computing the gradient gives

$$\nabla Err(\mathbf{w}) = -\sum_{i=1}^{n} \left(y_i \frac{1}{\sigma}(1 - \sigma)\mathbf{x}_i + (1 - y_i) \frac{-1}{1 - \sigma} \sigma(1 - \sigma)\mathbf{x}_i \right) = -\sum_{i=1}^{n} (y_i(1 - \sigma)\mathbf{x}_i - (1 - y_i)\sigma \mathbf{x}_i)$$

And so we get that

$$\nabla Err(\mathbf{w}) = -\sum_{i=1}^{n} \big(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)\big)\mathbf{x}_i$$

Thus using pure stochastic gradient descent, we have the update rule

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \mathbf{x}_i \big(\sigma(\mathbf{w}^\top \mathbf{x}_i) - y_i\big)$$

5 Multiclass Classification

5.1 Softmax Regression

Suppose we have K different classes we want to classify our input into (e.g. airplane, automobile, bird, cat, etc.). One way to do this is to define K binary classifiers and then return the class identified as true. Though this will of course not work if two binary classifiers both return true, and furthermore this is expensive.

Instead what we can do is have our output space be probability tuples, i.e. it is the K-simplex $\Delta^K = \{(p_1, \ldots, p_K) \mid p_i \geq 0, \sum_i p_i = 1\}$. Suppose we have an input vector $\mathbf{x}$, then we can multiply it with weights $\mathbf{w}_1, \ldots, \mathbf{w}_m$ to get $\mathbf{w}_1^\top \mathbf{x}, \ldots, \mathbf{w}_m^\top \mathbf{x}$. Since the sum of these outputs is not necessarily 1, we can normalize these and return

$$p_i = \frac{\mathbf{w}_i^\top \mathbf{x}}{\sum_j \mathbf{w}_j^\top \mathbf{x}}$$

But this is not good enough; what if $\mathbf{w}_i^\top \mathbf{x}$ isn't positive? We will use the *softmax* function: a continuous function which approximates the maximum of a vector:

Definition 5.1.1

We define the **softmax** function to be a function softmax: $\mathbb{R}^n \longrightarrow \mathbb{R}^n$ with components $\text{softmax}_i\colon \mathbb{R}^n \longrightarrow \mathbb{R}$:

$$\text{softmax}_i(\mathbf{z}) = \frac{\exp(\mathbf{z}_i)}{\sum_j \exp(\mathbf{z}_j)}$$

we can then define the softmax of $\mathbf{z}$ to be $\text{softmax}(\mathbf{z}) = (\text{softmax}_1(\mathbf{z}), \ldots, \text{softmax}_n(\mathbf{z}))$.

Notice that trivially the sum of $\text{softmax}_i(\mathbf{z})$ is 1, and $\text{softmax}_i(\mathbf{z}) > 0$ for all i. Thus we will define

$$p_i = \text{softmax}_i(\mathbf{w}_i^\top \mathbf{x}) = \frac{\exp(\mathbf{w}_i^\top \mathbf{x})}{\sum_j \exp(\mathbf{w}_j^\top \mathbf{x})}$$

In matrix form

$$\mathbf{p} = \text{softmax}(\mathbf{W}\mathbf{x})$$

where the rows of $\mathbf{W}$ are the weight vectors $\mathbf{w}_i$.

p_i is the probability that the class of $\mathbf{x}$ is i. We will also write it as $p_i = \mathbb{P}(y = i \mid \mathbf{x})$, i.e. the probability that the output class is i given the input x. If the variable y is in use, I will also use $\mathbb{P}(o = i \mid \mathbf{x})$ (o for output).

Definition 5.1.2

Suppose we are given a sequence of independent identically distributed training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. Then the **likelihood** of a weight matrix $\mathbf{W}$ is defined to be

$$\mathcal{L}(\mathbf{W}) = \prod_{i=1}^n \prod_{k=1}^K \mathbb{P}(o_i = k \mid \mathbf{x}_i)^{\mathbf{1}\{y_i = k\}} = \prod_{i=1}^n \mathbb{P}(o_i = y_i \mid \mathbf{x}_i)$$

Then the **log-likelihood** is

$$\log \mathcal{L}(\mathbf{W}) = \sum_{i=1}^n \sum_{k=1}^K \mathbf{1}\{y_i = k\} \log \mathbb{P}(o_i = k \mid \mathbf{x}_i) = \sum_{i=1}^n \sum_{k=1}^K \mathbf{1}\{y_i = k\} \log \frac{\exp(\mathbf{w}_k^\top \mathbf{x}_i)}{\sum_j \exp(\mathbf{w}_j^\top \mathbf{x}_i)}$$

We can then perform stochastic gradient descent where the loss function is the log-likelihood.

5.2 Support Vector Machines (SVM)

Suppose we want to solve the problem of binary classification, i.e. we want to split our space into halfspaces. There can be many halfspaces which do so (think of the simplest problem, where we have two points), so we want to find some optimal halfspace. Let us define the *margin* of a class to a half space to be the minimum distance between the dividing hyperplane and a point from the class.

We need a quick result from linear algebra:

Proposition 5.2.1

Let $\mathcal{H}$ be the space defined by $\mathbf{w}^\perp$ for $\mathbf{w} \neq 0$. Show that for any $\mathbf{x}$, $\mathbf{x} - \pi_{\mathcal{H}}(\mathbf{x}) = \frac{\langle \mathbf{x}, \mathbf{w}\rangle}{\langle \mathbf{w}, \mathbf{w}\rangle}\mathbf{w}$.

Proof: First let us assume that $\mathbf{w}$ is normalized, then let us extend $\mathbf{w}$ to an orthonormal basis, $\{\mathbf{w}, \mathbf{w}_1, \dots, \mathbf{w}_n\}$. Then $\{\mathbf{w}_1, \dots, \mathbf{w}_n\}$ is an orthonormal basis for $\mathcal{H}$. Now suppose that $\mathbf{x} = \alpha\mathbf{w} + \sum_i \alpha_i \mathbf{w}_i$, then $\pi_{\mathcal{H}}(\mathbf{x}) = \sum_i \alpha_i \mathbf{w}_i$ and so $\mathbf{x} - \pi_{\mathcal{H}}(\mathbf{x}) = \alpha\mathbf{w}$. And $\langle \mathbf{x}, \mathbf{w}\rangle = \alpha$, so we have that $\mathbf{x} - \pi_{\mathcal{H}}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w}\rangle\mathbf{w}$ as required (since $\mathbf{w}$'s norm is 1).

Now let $\widehat{\mathbf{w}}$ be the normalization of $\mathbf{w}$, i.e. $\widehat{\mathbf{w}} = \mathbf{w}/\|\mathbf{w}\|$. Then by above

$$\mathbf{x} - \pi_{\mathcal{H}}(\mathbf{x}) = \langle \mathbf{x}, \widehat{\mathbf{w}}\rangle\widehat{\mathbf{w}} = \frac{\langle \mathbf{x}, \mathbf{w}\rangle}{\langle \mathbf{w}, \mathbf{w}\rangle}\mathbf{w}$$

as required. ∎

If our hyperplane is given by $\mathcal{H} = \{\mathbf{x} \mid \langle \mathbf{w}, \mathbf{x}\rangle = 0\} = \mathbf{w}^\perp$ then the distance between a point $\mathbf{x}$ and $\mathcal{H}$ is just $\|\mathbf{x} - \pi_{\mathcal{H}}(\mathbf{x})\|$, and by the above proposition this is just

$$\left\|\frac{\langle \mathbf{x}, \mathbf{w}\rangle}{\langle \mathbf{w}, \mathbf{w}\rangle}\mathbf{w}\right\| = \frac{\left|\mathbf{w}^\top\mathbf{x}\right|}{\|\mathbf{w}\|}$$

So the margin of a class $\mathcal{C}$ is just

$$\min_{\mathbf{x}\in\mathcal{C}} \frac{\mathbf{w}^\top\mathbf{x}}{\|\mathbf{w}\|}$$

In our situation we have two classes: those $\mathbf{x}$s where $\mathbf{w}^\top\mathbf{x} > 0$ and those where $\mathbf{w}^\top\mathbf{x} < 0$. So the margin of the first class is just $\min_{\mathbf{w}^\top\mathbf{x}>0} \frac{\mathbf{w}^\top\mathbf{x}}{\|\mathbf{w}\|}$, and the margin of the second class is $\min_{\mathbf{w}^\top\mathbf{x}<0} -\frac{\mathbf{w}^\top\mathbf{x}}{\|\mathbf{w}\|} = -\max_{\mathbf{w}^\top\mathbf{x}<0} \frac{\mathbf{w}^\top\mathbf{x}}{\|\mathbf{w}\|}$. That means that $\min_{\mathbf{w}^\top\mathbf{x}>0} \mathbf{w}^\top\mathbf{x} = -\max_{\mathbf{w}^\top\mathbf{x}<0} \mathbf{w}^\top\mathbf{x} = \varepsilon$.

So $y_i = 1$ ($\mathbf{x}_i$ is in the positive class) if and only if $\mathbf{w}^\top\mathbf{x}_i \geq \varepsilon$ and $y_i = -1$ if and only if $\mathbf{w}^\top\mathbf{x}_i \leq -\varepsilon$. That is we have that $y_i\mathbf{w}^\top\mathbf{x}_i \geq \varepsilon$. These form two new parallel hyperplanes, and their distance is $\frac{2\varepsilon}{\|\mathbf{w}\|}$. So to maximize the distance between these hyperplanes, we want to minimize $\|\mathbf{w}\|$, equivalently $\|\mathbf{w}\|^2$. Note that if $\mathbf{w}$ is an optimal solution so is $\alpha\mathbf{w}$ (since they define the same hyperplanes), so we can scale $\mathbf{w}$ so that $\varepsilon = 1$. Meaning we want to solve the problem of minimizing $\|\mathbf{w}\|^2$ with respect to the constraints $y_i\mathbf{w}^\top\mathbf{x}_i \geq 1$ for every sample $\mathbf{x}_i$.

5.3 Constrained Optimization

Suppose we have a problem, like in the previous subquestion, where we want to minimize some function $f(\mathbf{x})$ with respect to some constraints $g_i(\mathbf{x}) \leq 0$:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad \forall i\colon g_i(\mathbf{x}) \leq 0$$

This is equivalent to minimizing the function $J(\mathbf{x}) = f(\mathbf{x}) + \sum_i \text{penalty}(g_i(\mathbf{x}))$ where penalty $= I$ is the infinite step function: $I(x) = \infty$ if $x > 0$ and zero otherwise. But this is hard to minimize since penalty here is not continuous. Instead we can approximate I linearly, using the *Lagrangian*:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x})$$

Proposition 5.3.1

For every $\mathbf{x}$, $\max\limits_{\boldsymbol{\lambda}\geq 0} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = J(\mathbf{x})$.

Proof: if $f_i(\mathbf{x}) \leq 0$ for all i then $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$ is maximized when $\boldsymbol{\lambda} = 0$ and then $\mathcal{L}(\mathbf{x}, 0) = f(\mathbf{x}) = J(\mathbf{x})$. If $f_i(\mathbf{x}) > 0$ for some i then taking $\lambda_i \to \infty$ we get $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \to \infty$, so the maximum is ∞, and $J(\mathbf{x}) = \infty$. ∎

Thus the problem is equivalent to $\min_{\mathbf{x}} \max_{\boldsymbol{\lambda}\geq 0} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$. Suppose we flip the minimum and maximum: $\max_{\boldsymbol{\lambda}\geq 0} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$. Let us define

$$g(\boldsymbol{\lambda}) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}), \qquad h(\mathbf{x}) = \max_{\boldsymbol{\lambda}\geq 0} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$$

Notice then that for all $\mathbf{x}, \boldsymbol{\lambda}$:

$$g(\boldsymbol{\lambda}) \leq \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \leq h(\mathbf{x})$$

Thus we can maximize $\boldsymbol{\lambda}$ and minimize $\mathbf{x}$ and we get

$$\max_{\boldsymbol{\lambda}\geq 0} g(\boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}\geq 0} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \leq \min_{\mathbf{x}} \max_{\boldsymbol{\lambda}\geq 0} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \min_{\mathbf{x}} h(\mathbf{x})$$

Definition 5.3.2

The **Lagrangian dual program** of the primal program

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad \forall i \colon g_i(\mathbf{x}) \leq 0$$

is the program of maximizing $g(\boldsymbol{\lambda})$: $\max_{\boldsymbol{\lambda} \geq 0} g(\boldsymbol{\lambda})$ where $g(\boldsymbol{\lambda}) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$.

We have just shown

Theorem 5.3.3 (Weak Duality Principle)

The Lagrangian dual program gives a lower bound to the optimal solution of the primal program.

5.4 Solving Support Vector Machines

Recall that an SVM is of the form (we minimize $\frac{1}{2}\|\mathbf{w}\|^2$ so the gradient is nicer):

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad \forall i \colon y_i \mathbf{w}^\top \mathbf{x}_i \geq 1$$

which can be equivalently written as

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad \forall i \colon 1 - y_i \mathbf{w}^\top \mathbf{x}_i \leq 0$$

The Lagrangian is then

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^n \lambda_i (1 - y_i \mathbf{w}^\top \mathbf{x}_i)$$

Taking the gradient only with respect to $\mathbf{w}$ gives

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \mathbf{w} - \sum_{i=1}^n \lambda_i y_i \mathbf{x}_i$$

This is zero when

$$\mathbf{w} = \sum_{i=1}^n \lambda_i y_i \mathbf{x}_i$$

Notice then that

$$\|\mathbf{w}\|^2 = \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

So plugging this back into the Lagrangian we get

$$g(\boldsymbol{\lambda}) = \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \boldsymbol{\lambda}) = \frac{1}{2} \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^n \lambda_i - \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j = \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

Taking the partial derivative of $g(\boldsymbol{\lambda})$ with respect to λ_i gives

$$\frac{\partial}{\partial \lambda_i} g(\boldsymbol{\lambda}) = 1 - \frac{1}{2} \sum_{j=1}^n \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

So we can use this to compute the gradient ascent of (since we want to increase) $g(\boldsymbol{\lambda})$.

5.5 Solving Noise

Suppose our data is not separable; there is noise. That means that there is no halfspace which correctly classifies our training data. For each $\mathbf{x}_i$ we define a parameter ξ_i which will measure how far away $\mathbf{x}_i$ is from the correct class (the margin).

If we have that $y_i\mathbf{w}^\top\mathbf{x}_i \geq 1$ then $\mathbf{x}_i$ is properly classified and so ξ_i should be zero. Otherwise we want to measure how far away $\mathbf{x}_i$ is from the margin $\mathbf{w}^\top\mathbf{x} = 1$. The distance to the halfspace $\mathbf{w}^\top\mathbf{x} = 0$ is $\frac{|\mathbf{w}^\top\mathbf{x}_i|}{\|\mathbf{w}\|}$, and the distance from $\mathbf{w}^\top\mathbf{x} = 1$ to $\mathbf{w}^\top\mathbf{x} = 0$ is $\frac{1}{\|\mathbf{w}\|}$. If $\mathbf{x}_i$ is misclassified, then the first distance is $-\frac{y_i\mathbf{w}^\top\mathbf{x}_i}{\|\mathbf{w}\|}$, so the total distance to the correct margin is $\frac{1}{\|\mathbf{w}\|}(1 - y_i\mathbf{w}^\top\mathbf{x}_i)$. And if $\mathbf{x}_i$ is classified correctly but not beyond the margin, then we want the difference between the two distances, which is the same. So all in all we should have something like $\xi_i = \frac{1}{\|\mathbf{w}\|}(1 - y_i\mathbf{w}^\top\mathbf{x}_i)$. $\|\mathbf{w}\|$ is just a constant so we can simply set $\xi_i = 1 - y_i\mathbf{w}^\top\mathbf{x}_i$.

Notice that in the second case we have that $y_i\mathbf{w}^\top\mathbf{x}_i < 1$ and so $1 - y_i\mathbf{w}^\top\mathbf{x}_i > 0$ so we can just define

$$\xi_i = \max\{0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i\}$$

So to solve noise what we can do is minimize $\frac{1}{2}\|\mathbf{w}\|^2 + c\sum_i \xi_i$ where c is the cost of a misclassification, according to the constraints $y_i\mathbf{w}^\top\mathbf{x}_i \geq 1 - \xi_i$.

5.6 Nonlinear SVMs

Suppose our data is not linearly separable: perhaps it is a circle. The idea to solve this is we can perform a transformation of the input space into some larger dimension, and hope that the inputs are now linearly classifiable in this larger dimension. So we define a *feature map* which is a function from the input space to a feature space, $\varphi\colon \mathbb{R}^d \longrightarrow \mathbb{R}^D$ where $D \gg d$. Recall that the gradient of $g(\boldsymbol{\lambda})$ has components of the form

$$\frac{\partial}{\partial\lambda_i} g(\boldsymbol{\lambda}) = 1 - \frac{1}{2}\sum_{j=1}^{n} \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$$

So if we now replace $\mathbf{x}_i$ with the feature $\varphi(\mathbf{x}_i)$, we get

$$\frac{\partial}{\partial\lambda_i} g(\boldsymbol{\lambda}) = 1 - \frac{1}{2}\sum_{j=1}^{n} \lambda_j y_i y_j \varphi(\mathbf{x}_i)^\top \varphi(\mathbf{x}_j)$$

Notice that φ always appears in an inner product, so we can define the *kernel* to be $k(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x})^\top\varphi(\mathbf{y})$, and then we have that

$$\frac{\partial}{\partial\lambda_i} g(\boldsymbol{\lambda}) = 1 - \frac{1}{2}\sum_{j=1}^{n} \lambda_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

For some types of feature maps computing k is simpler than φ. For example:

$$\varphi(\mathbf{x}) = \varphi\begin{pmatrix} x1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \\ x_1 \\ x_2 \\ 1 \end{pmatrix}$$

Then a computation gives us

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top\mathbf{z} + 1)^2$$

So we can focus on kernels instead of feature maps, in fact we have the following result due to Moore-Aronszajn:

Theorem 5.6.1 (Moore-Aronszajn)

Every positive semi-definite function k defines a valid kernel.

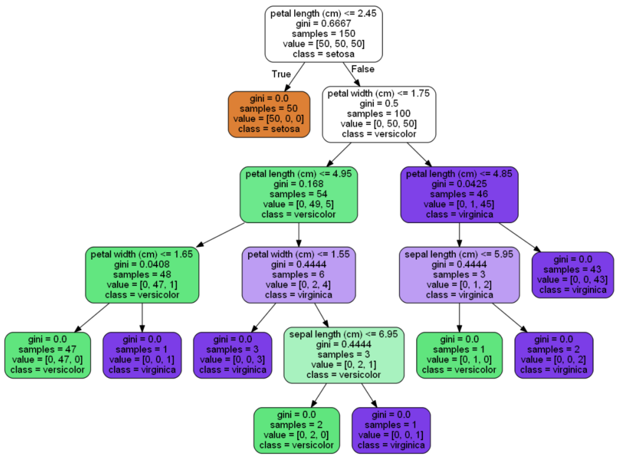
So for example, we can define

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

and this is positive semi-definite so it defines a kernel.

5.7 Decision Trees

A *decision tree* is an iterative procedure where at each iteration data is split according to some cutoff on a certain feature. The features and cutoff points are chosen to minimize either variance: $\frac{1}{n}\sum_y (y - \bar{y})^2$ (for numeric/real y), or entropy: $-\sum_y p(y)\log p(y)$ (for discrete y) where $p(y)$ is the probability of a sample belong to class y. Take for example the following decision tree to classify flowers:


petal length (cm) <= 2.45 gini = 0.6667 samples = 150 value = [50, 50, 50] class = setosa
True
False
gini = 0.0 samples = 50 value = [50, 0, 0] class = setosa
petal width (cm) <= 1.75 gini = 0.5 samples = 100 value = [0, 50, 50] class = versicolor
petal length (cm) <= 4.95 gini = 0.168 samples = 54 value = [0, 49, 5] class = versicolor
petal length (cm) <= 4.85 gini = 0.0425 samples = 46 value = [0, 1, 45] class = virginica
petal width (cm) <= 1.65 gini = 0.0408 samples = 48 value = [0, 47, 1] class = versicolor
petal width (cm) <= 1.55 gini = 0.4444 samples = 6 value = [0, 2, 4] class = virginica
sepal length (cm) <= 5.95 gini = 0.4444 samples = 3 value = [0, 1, 2] class = virginica
gini = 0.0 samples = 43 value = [0, 0, 43] class = virginica
gini = 0.0 samples = 47 value = [0, 47, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica
gini = 0.0 samples = 3 value = [0, 0, 3] class = virginica
sepal length (cm) <= 6.95 gini = 0.4444 samples = 3 value = [0, 2, 1] class = versicolor
gini = 0.0 samples = 1 value = [0, 1, 0] class = versicolor
gini = 0.0 samples = 2 value = [0, 0, 2] class = virginica
gini = 0.0 samples = 2 value = [0, 2, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica

Each node has the following information:

(1) The feature and cutoff: the features here are petal width and sepal length, and the cutoffs are the conditions at the top of each (non-leaf) node.

(2) Gini: an alternative for entropy and variance, ignore this.

(3) The number of samples: the number of samples remaining to be classified.

(4) The value: an array showing how many of the current samples belong to each class.

(5) The class: the current class the set of samples are believed to belong to.

Here we need to use entropy and not variance since the classes are discrete (and not numeric). The entropy of the first node is $-\frac{1}{3}\log\frac{1}{3}-\frac{1}{3}\log\frac{1}{3}-\frac{1}{3}\log\frac{1}{3}=\log 3$, since the probability of being in a class is $\frac{1}{3}$ (by the values array). The entropy of its left child is $-1\log 1-0\log 0-0\log 0=0$ (since there is no unknown, the entropy should be zero). The entropy of its right child is $-\frac{1}{2}\log\frac{1}{2}-\frac{1}{2}\log\frac{1}{2}=1$.

5.8 Boosting and Gradient Boosting

Boosting is an ensemble-based approach to machine learning (meaning it uses many other learning algorithms to improve performance). Suppose we have a labeled sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ which are independent and identically distributed. Furthermore we have a loss function $\ell(\bullet,\bullet)$. We want to learn a function f which minimizes the empirical risk (as this is an approximation to minimizing generalized risk):

$$\mathcal{L}(f)=\frac{1}{n}\sum_i \ell(y_i, f(\mathbf{x}_i))$$

But there are far too many functions to choose from, so instead we consider a class $\mathcal{H}$ of simple base learners (typically small decision trees), and focus on functions of the form

$$f(\mathbf{x})=\sum_i \gamma_i h_i(\mathbf{x}),\qquad h_i\in\mathcal{H}$$

Typically in gradient descent we have an update rule of the form $\theta_{t+1}=\theta_t-\gamma_t\frac{\partial}{\partial\theta}\mathcal{L}(\theta_t)$. Here our update rule will be of the form

$$f_{t+1}=f_t-\gamma_{t+1}\frac{\partial}{\partial f}\mathcal{L}(f_t)$$

Our iterations are of the form $f_t=\sum_{i=1}^t\gamma_i h_i$, and so $f_{t+1}=f_t+\gamma_{t+1}h_{t+1}$. Thus we want to learn $h_{t+1}\in\mathcal{H}$ to fit $-\frac{\partial}{\partial f}\mathcal{L}(f_t)$, so that f_{t+1} is an approximation to the step we got in gradient descent. And we choose γ_{t+1} to minimize the overall loss:

$$\gamma_{t+1}=\operatorname*{argmin}_{\gamma}\mathcal{L}(f_t+\gamma h_{t+1})=\operatorname*{argmin}_{\gamma}\sum_{i=1}^n \ell(y_i, f_t(\mathbf{x}_i)+\gamma h_{t+1}(\mathbf{x}_i))$$

The gradient boosting algorithm is as follows:

```
1.  function GRADIENT BOOSTING($\{\mathbf{x}_i, y_i\}_{i=1}^n$)
```

```
2.      $f_1 \leftarrow c$ for some constant $c$
3.      **for** $(t = 1, \dots, M)$
4.          compute the residuals: $r_i = \frac{\partial}{\partial f(\mathbf{x})} \ell(y_i, f_t(\mathbf{x}_i))$
5.          learn $h_{t+1} \in \mathcal{H}$ to the new training data $\{\mathbf{x}_i, r_i\}_{i=1}^n$
6.          find $\gamma_{t+1}$ which minimizes $\mathcal{L}(f_t + \gamma_{t+1} h_{t+1})$
7.          $f_{t+1} \leftarrow f_t + \gamma_{t+1} h_{t+1}$
8.      **end for**
9.  **end function**
```

Some common loss functions include:

- Continuous y: MSE: $\ell(y, f(\mathbf{x})) = \frac{1}{2}(y - f(\mathbf{x}))^2$. Then the residual is $\frac{\partial}{\partial \ell}(y_i, f_t(\mathbf{x}_i)) = -(y_i - f_t(\mathbf{x}_i))$.
- Categorical/discrete y: consider $y = \pm 1$

  (**1**) Bernoulli loss: $\ell(y, f(\mathbf{x})) = \log(1 + \exp(-2yf(\mathbf{x})))$.

  (**2**) Adaboost loss: $\ell(y, f(\mathbf{x})) = \exp(-2yf(\mathbf{x}))$.