Mathematics for Machine Learning

NYU Paris, CSCI-UA 9473

Augustin Cosse

This note provides a summary of the mathematics needed for an introductory class in machine learning. The version is temporary. Please direct any comments or questions to acosse@nyu.edu or acosse@ens.fr

1 Notations

$\mathbb{R}$	Real numbers
$\langle\cdot,\cdot\rangle$	Inner product, $\langle \boldsymbol{v}, \boldsymbol{w}\rangle = \sum_{i=1}^n v_i w_i$, $\langle \boldsymbol{A}, \boldsymbol{B}\rangle = \sum_{i=1}^n \sum_{j=1}^m A_{ij}B_{ij}$
$\text{Tr}(\boldsymbol{A})$	Trace of a matrix $\text{Tr}(\boldsymbol{A}) = \sum_{i=1}^n A_{ii}$
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ, variance σ^2 (see below)
$A \propto B$	A Proportional to B. We will use this symbol often when considering Bayes theorem with uniform priors. For example we will often write $p(\theta\|x) \propto P(x\|\theta)p(\theta)$ where we neglect the normalizing constant $P(x)$.
$\delta(x = k)$	Indicator function, $\delta(x = k) = 1$ if $x = k$ and 0 otherwise
$\mathcal{D}$	Usually used to represent a dataset, $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}$
$\phi(\boldsymbol{x})$	For a given prototype $\boldsymbol{x}$, we usually reserve the notation $\phi(\boldsymbol{x})$ to denote the corresponding feature vector of $\boldsymbol{x}$
$\frac{\partial f}{\partial x_i}$	partial derivative of $f(x_1, \dots, x_N)$ with respect to the variable x_i
$\nabla f(\boldsymbol{x})$	gradient, $\nabla f(\boldsymbol{x}) = (\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n})$

Table 1: Notations

Contents

2 Linear algebra

Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$,

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \ddots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{1}$$

we define its transpose as the matrix $\boldsymbol{A}^T \in \mathbb{R}^{n \times m}$ whose j^{th} column is defined as the j^{th} row of $\boldsymbol{A}$, i.e.

$$\boldsymbol{A}^T = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \ddots & & \\ a_{1m} & \dots & a_{n-1,m} & a_{nm} \end{pmatrix} \tag{2}$$

If $\boldsymbol{A}$ and $\boldsymbol{B}$ are $m \times n$ matrices, we have the following properties

- $(\boldsymbol{A}^T)^T = \boldsymbol{A}$
- $(\boldsymbol{A} + \boldsymbol{B})^T = \boldsymbol{A}^T + \boldsymbol{B}^T$
- $(\alpha \boldsymbol{A})^T = \alpha \boldsymbol{A}^T$
- $(\boldsymbol{A}\boldsymbol{B})^T = \boldsymbol{B}^T \boldsymbol{A}^T$

Definition 1 (PseudoInverse)**.** *Given a matrix $\boldsymbol{A}$, the pseudo inverse is defined as the matrix $\boldsymbol{A}^+$ that satisfies the following conditions*

$$\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{A} \tag{3}$$
$$\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}^+ \tag{4}$$
$$(\boldsymbol{A}\boldsymbol{A}^+)^T = \boldsymbol{A}\boldsymbol{A}^+ \tag{5}$$
$$(\boldsymbol{A}^+\boldsymbol{A})^T = \boldsymbol{A}^+\boldsymbol{A} \tag{6}$$

When the matrix $\boldsymbol{A}^T\boldsymbol{A}$ is invertible, the pseudo inverse can be defined as $(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$

2.1 Norms and inner products

Given two vectors $\boldsymbol{v} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^n$, one can define an inner product as

$$\boldsymbol{v} \cdot \boldsymbol{w} = \langle \boldsymbol{v}, \boldsymbol{w} \rangle = \sum_{i=1}^{n} v_i w_i \tag{7}$$

Genrally speaking, a function $\langle \cdot, \cdot \rangle$ is an inner product if it satisfies the following properties

- $\langle x, x \rangle \geq 0$, $\langle x, x \rangle = 0 \iff x = 0$ (positivity)
- $\langle x, y \rangle = \langle y, x \rangle$ (symmetry)
- $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$ (additivity)
- $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$, for all $\alpha \in \mathbb{R}$ (homogeneity)

So far, we have discussed inner products for vectors, but it is also possible to define an inner product on matrices. A standard inner product can be defined in a similar way on matrices $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{m\times n}$,

$$\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \mathrm{Tr}(\boldsymbol{X}^T \boldsymbol{Y}) = \sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij} Y_{ij} \tag{8}$$

Here we used the notation $\mathrm{Tr}(\boldsymbol{A})$ to denote the trace of a square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, i.e. $\mathrm{Tr}(\boldsymbol{X}) = \sum_{i=1}^{n} A_{ii}$.

Another important function is the norm. A norm is a function $f \ : \ \mathbb{R}^n \mapsto \mathbb{R}$ anf which has the following properties

- $f(\boldsymbol{x}) \geq 0$, $f(\boldsymbol{x}) = 0 \iff \boldsymbol{x} = 0$ (positivity)
- $f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x}), \quad \forall \alpha \in \mathbb{R}$ (homogeneity)
- $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$ (triangle inequality)

Examples of norms include

- The ℓ_2 norm: $\|\boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$
- The ℓ_1 norm: $\|\boldsymbol{v}\|_1 = \sum_{i=1}^{n} |v_i|$
- The ℓ_∞ norm: $\|\boldsymbol{v}\|_\infty = \max_i |v_i|$
- More generally, the ℓ_p norms are defined for $p \geq 1$ as $\|\boldsymbol{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p}$

Just as for vectors, one can defined norms on the vectors. Those norms will satisfy the same properties. The most popular one is the Frobenius norm

$$\|\boldsymbol{A}\|_F = \|\boldsymbol{A}\|_2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}^2} = \sqrt{\mathrm{Tr}(\boldsymbol{A}^T \boldsymbol{A})} \tag{9}$$

In fact given any inner product (on vectors or matrices), one can define an "induced" norm by letting $f(\boldsymbol{v}) = \|\boldsymbol{v}\| = \sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle}$

3 Differential Calculus

Given a multivariate scalar function $f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^n$, one defines the gradient of the function as the vector encoding the derivatives of the function f with respect to each of the component of $\boldsymbol{x}$, i.e.,

$$\nabla f(\boldsymbol{x}) = \frac{\partial f}{\partial \boldsymbol{x}} = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n}\right) \tag{10}$$

The Hessian matrix of a multivariate function f $\mathbb{R}^n \ \mapsto \ \mathbb{R}^n$, is the matrix encoding all second order partial derivatives of f,

$$\nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \ddots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix} \tag{11}$$

Compactly we can write

$$H = \nabla^2 f = \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{12}$$

3.1 Matrix differentiation

Consider the inner product $\langle \boldsymbol{v}, \boldsymbol{w} \rangle$. When considering the derivative of a function $f(\boldsymbol{A})$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with respect to any matrix $\boldsymbol{A}$, we will use the abridged notations $\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}}$ to denote the $m \times n$ matrix encoding the first order derivatives of the function f with respect to the entries of the matrix $\boldsymbol{A}$, i.e.

$$\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}} = \begin{pmatrix} \frac{\partial f(\boldsymbol{A})}{\partial a_{11}} & \frac{\partial f(\boldsymbol{A})}{\partial a_{12}} & \cdots & \frac{\partial f(\boldsymbol{A})}{\partial a_{1m}} \\ \frac{\partial f(\boldsymbol{A})}{\partial a_{21}} & \frac{\partial f(\boldsymbol{A})}{\partial a_{2,2}} & \cdots & \frac{\partial f(\boldsymbol{A})}{\partial a_{2m}} \\ \frac{\partial f(\boldsymbol{A})}{\partial a_{31}} & & \ddots & \vdots \\ \frac{\partial f(\boldsymbol{A})}{\partial a_{m1}} & \cdots & \frac{\partial f(\boldsymbol{A})}{\partial a_{m,n-1}} & \frac{\partial f(\boldsymbol{A})}{\partial a_{mn}} \end{pmatrix} \tag{13}$$

4 Statistics and probability

4.1 Probability

Definition 2. *Given a sample space $\mathcal{S}$ and an associated algebra $\mathcal{B}$, a probability function is a function P with a domain $\mathcal{B}$ that satisfies*

- $P(A) \geq 0$ *for all* $A \in \mathcal{B}$
- $P(S) = 1$
- *If* $A_1, A_2, \ldots \in \mathcal{B}$ *are pairwise disjoint, then* $P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$

Theorem 1. *If P is a probability function and A is any set in $\mathcal{B}$, then*

- $P(\emptyset) = 0$, *where* $\emptyset$ *is the empty set*
- $P(A) \leq 1$
- $P(A^c) = 1 - P(A)$

Theorem 2. *if P is a probability function and A and B are any sets in $\mathcal{B}$, then*

- $P(B \cap A^c) = P(B) - P(A \cap B)$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- *If* $A \subset B$, *then* $P(A) \leq P(B)$.

Definition 3. *If A and B are events in S, and $P(B) > 0$, then the conditional probability of A given B, written $P(A|B)$ is*

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{14}$$

Given definition (3), it is easy to see that $P(A \cap B) = P(A|B)P(B) = P(B \cap A) = P(B|A)P(A)$ (this idea is sometimes known as the chain rule). The generalization of this gives Bayes' rule:

Theorem 3 (Bayes' rule)**.** *Let $A_1, A_2, \ldots$ be a partition of a sample space, and let B be any set. Then, for each $i = 1, 2, \ldots$,*

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{\infty} P(B|A_j)P(A_j)} \tag{15}$$

(the denominator is simply the expansion $P(B) = \sum_{j=1}^{\infty} P(B|A_j)P(A_j)$)

Definition 4 (Independence)**.** *Two events A and B are statistically independent if*

$$P(A \cap B) = P(A)P(B) \tag{16}$$

Definition 5. *A collection of events $A_1, \dots, A_n$ are mutually independent if for any subcollection $A_{i_1}, \dots, A_{i_k}$, we have*

$$P\left(\bigcap_{j=1}^{k} A_{i_j}\right) = \prod_{j=1}^{k} P(A_{i_j}) \tag{17}$$

4.2 Random variables

The formal definition of a random variable is as a function from a sample space into the real numbers.

Definition 6 (Random variable, informal)**.** *A random variable is a function from a sample space into the real numbers*

It is important to understand that the set of random variables includes the samples themselves as one can of course choose the random variable to match a sample onto the value of this sample. However, the probability is only defined on the sample space. For example, a simple Gaussian random variable X is a function from the sample space $\mathcal{S} \equiv (-\infty, \infty)$ into this same sample space (or if you prefer the value of the sample). I.e. $X(s_i) = s_i$ in this case. But the definition is more general and one could for example define another random variable $Y = X^2$ now the random variable is not the identity anymore. It a function which take samples from the sample space $\mathcal{S}$ and raise them to the power 2. The connection to the sample space is important because the notion of probability P is defined only on this sample space. I.e we only know how likely an event s_i is, we don't know about $X(s_i)$. It is however possible to define an induced probability on the random variable X, by using P,

$$P_X(X \in A) = P(\{s \in \mathcal{S} \ : \ X(s) \in A\}) \tag{18}$$

In words, we look at the samples for which $X(s) \in A$ and we define the probability of the event $X \in A$ from the probability of the corresponding sample.

We are now ready to introduce the notions of cumulative distribution function (cdf), probability mass function (pmf) and probability density function (pdf).

Definition 7. *The cumulative distribution function or cdf of a random variable X of a random variable X denotes $F_X(x)$, is defined by*

$$F_X(x) = P(X \leq x), \quad \textit{for all } x \tag{19}$$

Definition 8. *The probability mass function (pmf) of a discrete random variable X is given by*

$$f_X(x) = P(X = x), \quad \textit{for all } x \tag{20}$$

Definition 9. *The probability density function or pdf, $f_X(x)$ of a continuous random variable X is the function that satisfies*

$$F_X(x) = \int_{-\infty}^{x} f_x(t)\, dt, \quad \textit{for all } x \tag{21}$$

4.3 Classical PDFs

We start by listing some important distributions:

- The Gaussian Distribution is the most popular distribution in statistics as it is used to

- The Binomial distribution (which we write $\text{Bin}(k|n, \boldsymbol{\theta})$) is used to represent the probability of observing the positive/successful outcomes (the probability of success is given by θ) that repeats itself k times over n successive trials.

$$\text{Bin}(k|n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{22}$$

  The first factor encode the number of possibilities of choosing k elements among the n without replacement.

$$\binom{n}{k} = \frac{n!}{(n-k)!k!} \tag{23}$$

  The mean and variance for this distribution are given by $\mu = n\theta$ and $\sigma^2 = n\theta(1-\theta)$

- The Bernoulli distribution. The Bernoulli distribution is defined on a binary variable. I.e if we consider a single experiment (tossing a coin) with only two possible outcomes which are encoded by the variable $y = \{1, 0\}$ and appear with probability p and $1-p$ respectively. to encode the probability of an outcome 0 or 1 we can thus just raise the probability p to the power y, as show below. This gives the Bernoulli pdf

$$\text{Ber}(x|\theta) = \theta^y (1-\theta)^{1-y} \tag{24}$$
$$\tag{25}$$

  This is also sometimes written

$$\text{Ber}(x|\theta) = \theta^{\delta(y=0)} (1-\theta)^{\delta(y=1)} \tag{26}$$
$$\tag{27}$$

  Where $\delta(\mathcal{C}) = 1$ if the condition $\mathcal{C}$ is satisfied.

- Categorical distribution or Multinoulli distribution. When considering a variable with K possible states (e.g. a dice with 6 faces), we will often encodes those states through dummy variables. This means that for example for a variable that can take three different values, $y = 1$, $y = 2$ or $y = 3$ (we will see later that this applied to features or classes in classification problems), we would encode each of these feature through the 3-tuples $\boldsymbol{t} = (1, 0, 0)$ $(y = 1)$, $\boldsymbol{t} = (0, 1, 0)$ $(y = 2)$ and $\boldsymbol{t} = (0, 0, 1)$ $(y = 3)$. If we use $\theta_j = p_j$, $j = 1, \ldots, 3$ to denote the probability to get each feature, the total probability density function can now read compactly as

$$\text{Cat}(x|\boldsymbol{\theta}) = \text{Mu}(\boldsymbol{x}|\theta) = p(x|\theta) = \prod_{j=1}^{K} \theta_j^{t_j} = \theta_1^{t_1} \ldots \theta_K^{t_K} = \boldsymbol{\theta}^{\boldsymbol{t}} \tag{28}$$

- The Laplace distribution is characterized by "fatter" tails than the Gaussian distribution (i.e the probability of getting values that are very different from the mean will be higher in the Laplace distribution). The pdf of the Laplace distribution reads as

$$p(x|\mu, \beta) = \frac{1}{2\beta} \exp(-\frac{|x-\mu|}{\beta}) \tag{29}$$

  one way to see this is that in the Laplace distribution the argument of the exponential is the absolute value whether in the the Normal distribution, the argument is squared which implies a faster decrease in the pdf for large values of the deviation to the mean, $|x - \mu|$.

4.4 MAP and MLE

This part gives a very basic introduction to the difference between the Frequentist and Bayesian notions of estimators. For more details see [1]

Definition 10. *A point estimator is any function of the sample $W(x_1, \dots, x_n)$*

The most popular method to derive an estimator on the parameters of a distribution is the method of maximum likelihood. For a given probability distribution $p(x_i|\theta_1, \dots, \theta_k)$ parametrized by $\boldsymbol{\theta}$, the likelihood function is defined as

$$L(\theta|x) = \prod_{i=1}^{n} f(x_i|\theta_1, \dots, \theta_n) \tag{30}$$

This function encodes the "likelihood" of observing the samples x_i for a particular choice of parameters θ. Given this function, it seems reasonable to look for the parameters $\theta_1, \dots, \theta_n$ which give the highest probability of observing the sample (i.e the distribution which gives the highest probability of observing the x_i is likely to be the distribution of those x_i). This idea leads to the Maximum likelihood estimator summarized by definition **??** below

Definition 11. *For each sample point $\boldsymbol{x}$, let $\hat{\theta}(\boldsymbol{x})$ be a parameter value at which $L(\theta|\boldsymbol{x})$ attains its maximum as a function of θ, with $\boldsymbol{x}$ held fixed. A maximum likelihood estimator (MLE) of the parameter θ based on a sample X is $\hat{\theta}(X)$.*

The MLE is considered a Frequentist estimator as it only relies on the sample itself without making any prior assumption on the parameters.

The Bayesian approach to statistics is fundamentally different. So far, we assumed that θ was an unknown but fixed quantity. I.e. A random sample $X_1, \dots, X_n$ was drawn from a population and based on those observations, some knowledge on θ was obtained. In the Bayesian approach, θ is described by a probability distribution (the prior distribution). We thus assume some level of uncertainty on θ. The prior is a subjective distribution which is set by the observer.

The idea is that the observer updates his original prior based on the sample that he draws from the population. The update is done by means of Bayes' rule,

$$p(\theta|\boldsymbol{x}) \propto p(\boldsymbol{x}|\boldsymbol{\theta})p(\theta) \tag{31}$$

Just as in the MLE setting, we can then estimate the parameter θ by finding the value which maximizes the new prior, $p(\theta|\boldsymbol{x})$.

4.5 Exponential family

Many of the probability density functions discussed above are part of a general family of distributions known as the exponential family.

A probability density function, $f(\boldsymbol{x}|\boldsymbol{\theta})$ is said to be in the exponential family if it is of the form

$$\begin{aligned} f(\boldsymbol{x}|\boldsymbol{\theta}) &= h(\boldsymbol{x})c(\boldsymbol{\theta})\exp(\sum_{i=1}^{k} w_i(\boldsymbol{\theta})t_i(\boldsymbol{x})) && (32)\\ &= \frac{1}{Z(\theta)} h(\boldsymbol{x})\exp(\boldsymbol{\theta}^T \phi(\boldsymbol{x})) && (33)\\ &= h(\boldsymbol{x})\exp(\boldsymbol{\theta}^T \phi(\boldsymbol{x}) - A(\boldsymbol{\theta})) && (34) \end{aligned}$$

where

$$\begin{aligned} Z(\boldsymbol{\theta}) &= \int_{\mathcal{X}^m} h(\boldsymbol{x})\exp(\boldsymbol{\theta}^T \phi(\boldsymbol{x}))\, d\boldsymbol{x} && (35)\\ A(\boldsymbol{\theta}) &= \log Z(\boldsymbol{\theta}) && (36) \end{aligned}$$

$Z(\boldsymbol{\theta})$ is called the partition function and $A(\boldsymbol{\theta})$ is called the log partition function. $\boldsymbol{\theta}$ are called the natural parameters or canonical parameters.

5 Probabilistic classifiers

When discussing classification, we often either look at classifiers as geometrical objects (i.e separating planes, maximal margin hyperplanes,..) or as probabilistic objects based on Bayes Theorem. In the last framework, we consdider two main groups of classifiers

- Generative classifiers
- Discriminative classifiers

The first class defines a model for the joint probability distribution $p(\boldsymbol{x}, y)$ (or equivalently for the class conditional density $p(\boldsymbol{x}|y)$) and hence provides a way to generate new samples $\boldsymbol{x}$. The second class defines a model for the probability distribution $p(y|\boldsymbol{x})$ (i.e the class posterior) and hence only provides a way to discriminate between classes. Below we discuss the two most popular discriminative models: The Naive Bayes classifier (generative) and the Logistic regression classifier (discriminative)

5.1 Naive Bayes classifier

The Naive Bayes classifier is a generative classifier (i.e we learn a model for $p(\boldsymbol{x}|y)$ or $p(\boldsymbol{x}, y)$). We consider a classification problem in which the feature vectors are D dimensional. Moreover, each of the D features can take K values. I.e, $\boldsymbol{x} \in \{1, \ldots, K\}^D$. If we assume that the features are independent within a class c (note that this is usually not the case), we can write the "class conditional density" as

$$p(\boldsymbol{x}|y = \mathcal{C}_\ell, \boldsymbol{\theta}) = \prod_{j=1}^{D} p(x_j|y = c, \boldsymbol{\theta}_{j\ell}) \tag{37}$$

In the equation above $\theta_{j\ell}$ is the parameter associated to feature j in class $\mathcal{C}_\ell$ (see below).

Possible choices for the probability distribution the include

- the Gaussian distribution

$$p(\boldsymbol{x}|y = \mathcal{C}_\ell; \boldsymbol{\theta}) = \prod_{j=1}^{D} \mathcal{N}(x_j|\mu_{j\ell}, \sigma^2_{j,\ell}) \tag{38}$$

  Here $\mu_{j,\ell}$ and $\sigma^2_{j,\ell}$ are the mean and variance for the j^{th} feature in class $\mathcal{C}_\ell$.

- When the features are encoded through binary variables $\beta = \{0, 1\}$, the multivariate distribution is often used. In this case, we have

$$p(\boldsymbol{x}|y = \mathcal{C}_\ell, \boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Ber}(x_j|\mu_{j,\ell}) \tag{39}$$

  Again, here $\mu_{j,\ell}$ is the probability of getting feature j in class ℓ.

5.1.1 Deriving the parameters through Maximum Likelihood

We want to train a classification algorithm from a set of pairs $(\boldsymbol{x}_i, c_i)$. The prototypes $\boldsymbol{x}_i$ are represented by D-dimensional feature vectors $x_i = (x_{i1}, \ldots, x_{iD})$. From this, if we assume that the probabilities of observing the features are independent, the probability of observing a given pair $(\boldsymbol{x}_i, c_i)$ reads as

$$p(\boldsymbol{x}_i, c_i|\boldsymbol{\theta}) = \prod_{j=1}^{D} p(x_{ij}|c_i, \boldsymbol{\theta}_j) p(c_i|\boldsymbol{\pi}) \tag{40}$$

Here $p(c_i|\boldsymbol{\pi})$ encodes the probability of observing the class i and $\boldsymbol{\pi} = (\pi_1, \dots, \pi_N)$ encode the probabilities of getting a particular class $\mathcal{C}_1$ to $\mathcal{C}_N$. Using the δ notation, we can thus write (40) as

$$p(\boldsymbol{x}_i|y_i|\boldsymbol{\theta}) = \prod_{c=1}^{C} \pi_c^{\delta(y_i=c)} \prod_{j=1}^{D} \prod_{c=1}^{N} p(x_{ij}|\boldsymbol{\theta}_{c,j})^{\delta(y_i=c)} \tag{41}$$

Taking the product over all pairs and then the log, the log likelihood can read as

$$\log(\mathcal{D}|\boldsymbol{\theta}) = \sum_{c=1}^{C} N_c \log(\pi_c) + \sum_{j=1}^{D} \sum_{c=1}^{N} \sum_{i \text{s.t.} y_i=c} \log p(x_{ij}|\boldsymbol{\theta}_{jc}) \tag{42}$$

Maximizing the log likelihood with respect to the parameters of the model then gives

$$\hat{\pi}_c = \frac{N_c}{N} \tag{43}$$

as an estimate for the class prior (the probability of being in Class c). This estimate is thus the ratio between the number of samples in class c and the total number of training samples.

If we assume that all N features $x_{1i}, \dots, x_{1,N}$ are binary and that all the vectors within a class follow the same distribution, then a good choice for $p(x_{ij}|\theta_{jc})$ is to take a Bernoulli distribution with parameter θ_{jc}. In this case we can show the MLE for the parameter θ_{jc} is given by the ratio of the number of vectors from class c which have their j^{th} feature equal to 1 over the total number of points from class c,

$$\hat{\theta}_{jc} = \hat{p}(\boldsymbol{x}_{ij} = 1|y = c) = \frac{N_{jc}}{N_c} \tag{44}$$

Once we have computed the parameters of the model, the model can be used to classify new points by relying again on Bayes (neglecting the normalizing constant), we have

$$p(y = c|\boldsymbol{x}, \mathcal{D}) \propto p(y = c|\mathcal{D}) \prod_{j=1}^{D} p(x_j|y = c, \mathcal{D}) \tag{45}$$

$$\propto p(y = c|\mathcal{D}) p(\boldsymbol{x}|y = c, \mathcal{D}) \tag{46}$$

and $p(y = c|\mathcal{D})$ can be estimated from the data as $\hat{p}(y = c|\mathcal{D}) = \frac{N_c}{C}$, i.e the number of training samples from class c over the total number of samples.

5.2 Logistic regression

Logistic regression is perhaps the most popular discriminative classifier. The probabilistic model is defined as

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \text{Ber}(y|\text{sigm}(\boldsymbol{w}^T \boldsymbol{x})) \tag{47}$$

where the sigmoid $\text{sigm}(\cdot)$ is defined as

$$\text{sigm}(x) = \frac{1}{1 + \exp(-x)} \tag{48}$$

We thus have

$$p(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \frac{\exp(\boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(\boldsymbol{w}^T \boldsymbol{x})} \tag{49}$$

$$p(y = 0|\boldsymbol{x}, \boldsymbol{w}) = 1 - \frac{\exp(\boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(\boldsymbol{w}^T \boldsymbol{x})} \tag{50}$$

The interesting aspect of logistic regression is that the ratio between the probability of each class (which can be used to do classification) is linear in the parameters of the model. You can indeed verify that

$$\log(\frac{p(y=1|\boldsymbol{x},\boldsymbol{w})}{p(y=0|\boldsymbol{x},\boldsymbol{w})}) = \boldsymbol{w}^T\boldsymbol{x} \tag{51}$$

This model in fact falls in the class of "genralized" linear models for that precise reason. This idea easily extends to multiple class by introducing "log-odd ratio" between the probability of each class and the last. For K classes, we have $K-1$ such ratios (the last probabilitiy is fixed through $\sum_i p(\mathcal{C}=i|x)=1$)

$$\log(\frac{P(\mathcal{C}=1|x)}{P(\mathcal{C}=K|x)}) = \boldsymbol{w}_1^T\boldsymbol{x} \tag{52}$$

$$\log(\frac{P(\mathcal{C}=1|x)}{P(\mathcal{C}=K|x)}) = \boldsymbol{w}_1^T\boldsymbol{x} \tag{53}$$

$$\vdots \tag{54}$$

$$\log(\frac{P(\mathcal{C}=K-1|x)}{P(\mathcal{C}=K|x)}) = \boldsymbol{w}_{K-1}^T\boldsymbol{x} \tag{55}$$

The posterior class probabilities are then defined as

$$P(\mathcal{C}=k|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^T\boldsymbol{x})}{1+\sum_{\ell=1}^{K-1}\exp(\boldsymbol{w}_\ell^T\boldsymbol{x})}, \quad k=1,\dots,K-1 \tag{56}$$

$$P(\mathcal{C}=K|\boldsymbol{x}) = \frac{1}{1+\sum_{\ell=1}^{K-1}\exp(\boldsymbol{w}_\ell^T\boldsymbol{x})} \tag{57}$$

The model (and the associated parameters) can be learned through Maximum likelihood, by writing the likelihood function for the N observation (noting c_n the class of $\boldsymbol{x}_n$),

$$\ell(\boldsymbol{w}_\ell, \ell=1,\dots,K-1) = \sum_{n=1}^N \log(p(\mathcal{C}=c_n|\boldsymbol{x})) \tag{58}$$

And then minimizing the function through gradient descent.

6 Optimization

Finding the global minimizer of sufficiently complex functions is usually hard because iterative algorithms will get trapped in "local minimas". When solving optimization problems, we will often be interested in understanding whether the particular point to which our algorithm converges is a local or a global minimum (resp maximum) of the function. Under some conditions (smoothness of the function), the general characterization of the local extremas of a function is relatively easy as shown by the following two propositions

Proposition 1 (First order necessary conditions)**.** *If $\boldsymbol{x}^*$ is a local minimizer and f is continuously differentiable in an open neighborhood of $\boldsymbol{x}^*$, then $\nabla f(\boldsymbol{x}^*)=0$.*

Proposition 2 (Second Order Necessary conditions)**.** *If x^* is a local minimizer of f and $\nabla^2 f$ is continuous in an open neighborhood of x^*, then $\nabla f(x^*)=0$ and $\nabla^2 f(x^*)$ is positive semidefinite.*

Given a function f, there also exist sufficient conditions under which one can guarantee that a point is a minimizer. This idea is summarized by proposition 3 below

Proposition 3 (Second Order Sufficient conditions)**.** *Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*)=0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.*

6.1 Convexity

Definition 12 (Convex set)**.** *A set of points, $\mathcal{C}$ is convex if the line segment between any two points in the set is included in the set, that is to say for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{C}$, and any scalar $\alpha \in \mathbb{R}$ with $0 \leq \alpha \leq 1$, we have*

$$\alpha \boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2 \in \mathcal{C} \tag{59}$$

When a set is not convex, it is always possible to define the smallest convex set that contains $\mathcal{C}$. This idea in particular plays an important role when replacing non convex function (which therefore cannot be optimized efficiently) by convex approximation whose solution can be found and studied by iterative algorithms.

Definition 13 (convex hull)**.** *Given a set of points $\mathcal{S}$, the convex hull of $\mathcal{S}$ is defined as*

$$\text{conv}\mathcal{S} = \{\alpha_1 \boldsymbol{x}_1 + \ldots, \alpha_k \boldsymbol{x}_k | \boldsymbol{x}_i \in \mathcal{S}, \alpha_i \geq 0, i = 1, \ldots, k, \alpha_1 + \ldots + \alpha_k = 1\} \tag{60}$$

I.e., conv$\mathcal{S}$ *is the set defined from all possible convex combinations of points from $\mathcal{S}$.*

6.2 Constrained problems and Lagrangian

When considering a basic mathematical programming problem of the form

$$\begin{aligned} \min \quad & f_0(\boldsymbol{x}) \\ \text{subject to} \quad & f_\ell(\boldsymbol{x}) \leq 0, \ell = 1, \ldots L. \end{aligned} \tag{61}$$

We will use K to denote the set of feasible points for the program (61) (i.e. the set of points $\boldsymbol{x}$ satisfying the constraints in (61)), i.e. $K = \{\boldsymbol{x} \mid f_\ell(\boldsymbol{x}) \leq 0\}$

We can write the Lagrangian by introducing positive multipliers $\lambda_\ell \in \mathbb{R}$ for each of the constraints $f_\ell(x) \leq 0$. As those constraints are negative, the Lagrangian then reads as

$$L = f_0(x) - \sum_{\ell=1}^{L} \lambda_\ell f_\ell(\boldsymbol{x}) \tag{62}$$

The following Theorem then gives necessary conditions (on the Lagrangian) for a point $\boldsymbol{x}$ to be a minimizer from the feasible set, K.

Theorem 4 (Kuhn-Tucker conditions)**.** *Assume that $f_\ell(\boldsymbol{x})$ ($\ell = 1, \ldots, L$) are all differentiable. If the function $f_0(\boldsymbol{x})$ attains a local minimum at a point $\boldsymbol{x}^*$ which belongs the feasible set K, then there exists a vector of multipliers, λ_ℓ^*, $\ell = 1, \ldots, L$, such that the following conditions are satisfied*

$$\frac{\partial f_0(\boldsymbol{x}^*)}{\partial x_j} + \sum_{\ell=1}^{L} \lambda_\ell^* \frac{\partial f_\ell(\boldsymbol{x}^*)}{\partial x_j} = 0, \quad (j = 1, \ldots, J) \tag{63}$$

$$f_\ell(\boldsymbol{x}^*) \leq 0, \quad (\ell = 1, \ldots, L) \tag{64}$$

$$\lambda_\ell^* f_\ell(\boldsymbol{x}^*) = 0, \quad (\ell = 1, \ldots, L) \tag{65}$$

$$\lambda_\ell^* \geq 0, \quad (\ell = 1, \ldots, L) \tag{66}$$

Proof. We can always replace the inequality constraints in (61), by introducing slack variables y_ℓ. I.e., we always have the equivalence

$$f_\ell(\boldsymbol{x}) \leq 0 \iff f_\ell(\boldsymbol{x}) + y_\ell^2 = 0 \tag{67}$$

for some variables y_ℓ.

For those slack variables, the Lagrangian reads as

$$L = f_0(\boldsymbol{x}) + \sum_{\ell=1}^{L} \lambda_\ell (f_\ell(\boldsymbol{x}) + y_\ell^2) \tag{68}$$

Any minimum of the Lagrangian must satisfy the zero gradient condition, so we have

$$\frac{\partial L}{\partial x_j} = \frac{\partial f_0(\boldsymbol{x}^*)}{\partial x_j} + \sum_{\ell=1}^{L} \lambda_\ell \frac{\partial (f_\ell(\boldsymbol{x}^*) + (y_\ell^*)^2)}{\partial x_j} = 0, \quad (\ell = 1, \dots, L) \tag{69}$$

$$\frac{\partial L}{\partial y_\ell} = 2\lambda_\ell y_\ell = 0 \tag{70}$$

$$\frac{\partial L}{\partial \lambda_\ell} = f_\ell(\boldsymbol{x}) + y_\ell^2 = 0 \tag{71}$$

Condition (70) above is equivalent to the "complementary slackness" condition (65). To see this, simply note that $\lambda_\ell y_\ell = 0$ implies either $\lambda_\ell = 0$ or $y_\ell = 0$.

In the second case, we have $y_i = 0$ and hence $f_\ell(x) + y_\ell^2 = 0$ which implies $f_\ell(\boldsymbol{x}) = 0$. For the reverse implication, simply note that $\lambda_\ell \neq 0$ implies $f_\ell = 0$ and hence $y_\ell^2 = -f_\ell = 0$ which finally gives $y_\ell \lambda_\ell = 0$

If $y_\ell \neq 0$ (equivalently, $\lambda_\ell = 0$), on the other hand, then (71) implies $y_\ell \lambda_\ell = -f_\ell \lambda_\ell = 0$ and the equivalence is straightforward.

Condition (71) is equivalent to (63) (i.e the slack variables can be eliminated from the derivative)

We are thus left with showing that conditions (69) to (71) imply the non negativity of the multipliers λ_ℓ. For a general minimization problem,

$$\min \quad f_0(\boldsymbol{x}) \tag{72}$$

$$\text{subject to} \quad f_i(\boldsymbol{x}) \leq b_i, \quad (i = 1, 2, \dots, m) \tag{73}$$

if we denote the optimal solution corresponding to a given value of the bounds b_i, as $\boldsymbol{x}_0(\boldsymbol{b})$, we have

$$\frac{\partial f_0(\boldsymbol{x}_0(\boldsymbol{b}))}{\partial b_i} = -\lambda_i \tag{74}$$

But on the other any increase in the b_i leads to a larger feasible set and hence the variation in the value of f_0 for such a small increase in b_i must always be negative (i.e we can only do better when we increase the feasible set), following from this, we must have

$$\frac{\partial f_0(x_0(\boldsymbol{b}))}{\partial b_i} \leq 0 \tag{75}$$

which implies $\lambda_i \geq 0$

$\square$

7 Halfspaces and hyperplanes

Given a vector $\boldsymbol{w} \in \mathbb{R}^N$, one can define a(n) (affine) hyperplane as the set of all points $\boldsymbol{x} \in \mathbb{R}$ satisfying the relation $\boldsymbol{w}^T\boldsymbol{x} + b = 0$.

The vector $\boldsymbol{w}$ encodes the normal to the hyperplane, which also means that for any two points $\boldsymbol{x}_1, \boldsymbol{x}_2$ that lie on the hyperplane, we necessarily have

$$\boldsymbol{w}^T\boldsymbol{x}_1 + b = \boldsymbol{w}^T\boldsymbol{x}_2 + b = 0 \tag{76}$$

(i.e the two points belong to the hyperplane) and hence,

$$\boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0 \tag{77}$$

Every hyperplane naturally defines a splitting of the space into two halfspaces. The set of points which are lying above the plane (i.e formally, the points for which $\boldsymbol{w}^T\boldsymbol{x} + b > 0$), and the set of points which are lying under the plane ($\boldsymbol{w}^T\boldsymbol{x} + b < 0$). Every hyperplane can thus be used as a natural classifier,

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b \tag{78}$$

where we put the point $\boldsymbol{x}$ in class $\mathcal{C}_0$ if $y(\boldsymbol{x}) > 0$ and in class $\mathcal{C}_1$ if $y(\boldsymbol{x}) < 0$.

7.1 Distance of a point to a hyperplane

When discussing the robustness of a classifier (and in particular when introducing the notion of margin), we will need the notion of distance of a point to a hyperplane.

Consider Fig 1 below. We consider the plane in red defined as $y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$. For any point $\mathbf{x}$ (shown in blue), one can consider the decomposition

$$\mathbf{x} = \mathbf{x}_\perp + d\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{79}$$

That is we write $\mathbf{x}$ as the combination of its projection onto the plane and a contribution of length d along the normal vector to the plane, $\boldsymbol{w}$. d thus encodes the distance of $\mathbf{x}$ to the hyperplane. Now if we multiply (79) by $\mathbf{w^T}$,

$$\mathbf{w^T x} = \mathbf{w^T x}_\perp + \boldsymbol{w}^T d\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{80}$$

and add the bias, we get

$$\mathbf{w^T x} + b = \mathbf{w^T x}_\perp + b + \boldsymbol{w}^T d\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{81}$$

The left handside is simnply $y(\boldsymbol{x})$. The first term on the right handside is 0 as $\mathbf{x}_\perp$ lies along the plane (hence is orthogonal to $\mathbf{w}$). We are thus left with

$$y(\boldsymbol{x}) = \|\mathbf{w}\|\mathbf{d} \tag{82}$$

from which follows

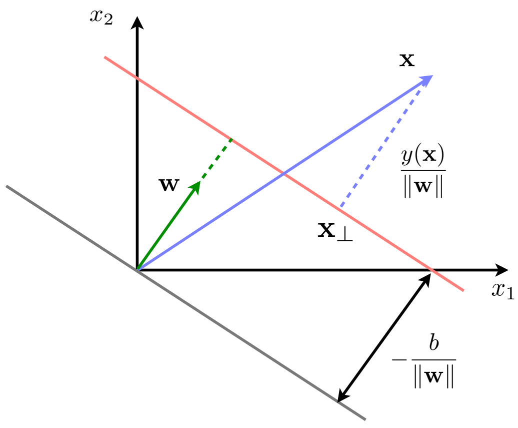
$$d = \frac{y(\boldsymbol{x})}{\|\mathbf{w}\|} \tag{83}$$

So the signed (because $\mathbf{d}$ can be both positive and negative here) distance, is given by the ratio of the prediction $y(\boldsymbol{x})$ over the norm of the normal vector $\boldsymbol{w}$. We will use this when discussing Maximal margin classifiers.

8 Regression and regularization

The simplest regression model, and one of the most widely used, is the linear regression model. In linear regression we want to learn a plane $(\boldsymbol{w}, b)$ that describes the data as well as possible. The idea here is that if we can show that such a plane is a good representation of the relation between $y(\boldsymbol{x}_n)$ and $\boldsymbol{x}_n$ for the training data we have, then it might give reliable predictions on new data $\boldsymbol{x}_n$

To do so, for a given set of training points, $(\boldsymbol{x}_n, y_n)$, we minimize the sum of squared errors between the predictions $y(\boldsymbol{x}_n)$ from the plane and the exact target values , t_n,



x_2
$\mathbf{x}$
$\frac{y(\mathbf{x})}{\|\mathbf{w}\|}$
$\mathbf{w}$
$\mathbf{x}_\perp$
x_1
$-\frac{b}{\|\mathbf{w}\|}$

Figure 1: Distance of a point to a hyperplane

$$\begin{aligned}
&\underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} |\hat{y}(\boldsymbol{x}^n;\boldsymbol{w},b) - t^n|^2 && (84)\\
&= \underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} |\boldsymbol{w}^T\boldsymbol{x}^n + b - t^n|^2 && (85)\\
&= \underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} |\sum_{i=1}^{D} \boldsymbol{x}_i^n \boldsymbol{w}_i + b - t_n|^2 && (86)
\end{aligned}$$

The model remains linear if we replace the original points by a representation in some feature space, $\boldsymbol{\phi}(\boldsymbol{x})$. In this case, the model reads as

$$y(\boldsymbol{x}^n;\boldsymbol{w};b) = b + \sum_{j=1}^{D} w_j \phi_j(\boldsymbol{x}^n) \tag{87}$$

and, including the bias b in the weight vector $\boldsymbol{w}$, $\tilde{\boldsymbol{w}} = [1, \boldsymbol{w}]$ and letting $\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) = [1,\ \boldsymbol{\phi}(\boldsymbol{x})]$, we have

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \sum_{n=1}^{N} |y(\boldsymbol{x}^n;\tilde{\boldsymbol{w}}) - t^n|^2 = \sum_{n=1}^{N} \left|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 \tag{88}$$

The solution to problem (88) can be computed in closed form by setting the derivatives to 0 and then solving for $\boldsymbol{w}$ and b. Setting the derivative of (84) (on the generic feature formulation) to 0, we get

$$\sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right)\boldsymbol{\phi}(\boldsymbol{x}_n)^T = 0 \tag{89}$$

(Verify this using the matrix derivatives). When solving for $\boldsymbol{w}$, we thus have

$$\sum_{n=1}^{N} t_n\boldsymbol{\phi}(\boldsymbol{x}_n)^T - \boldsymbol{w}^T\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right) = 0 \tag{90}$$

which gives

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} \tag{91}$$

Here $\boldsymbol{t}$ is the vector concatenating the targets $\boldsymbol{t} = (t_1, \dots, t_n)$, and $\boldsymbol{\Phi}$ is the matrix whose columns encode the feature vectors $\boldsymbol{\phi}(\boldsymbol{x}_n)$, i.e.

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\boldsymbol{x}_0) & \phi_1(\boldsymbol{x}_0) & \dots & \phi_M(\boldsymbol{x}_0) \\ \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \dots & \phi_M(\boldsymbol{x}_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & \dots & \phi_0(\boldsymbol{x}_N) \end{pmatrix} \tag{92}$$

The matrix $\boldsymbol{\Phi}^+ = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T$ is precisely the Moore Penrose Pseudo inverse which was introduced in Definition 1.

This solution highlights an important aspect of linear regression: For the solution of problem (88) to be well defined, we need the Gram matrix $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ to be invertible.

When this matrix is not invertible, it means that columns or rows, of that matrix are linearly dependent and hence, that a given subset of the feature vectors $\boldsymbol{\phi}(\boldsymbol{x}_n)$ can be written from the knowledge of all the remaining features. The issue with such redundancy in the representation of the data is that one can define a perfectly valid classifier for the dataset, but for which the weights can vary arbitrarily and in particular, be arbitrarily high as they can cancel each other. Indeed, asssume that for all training points, features $\phi_1(\boldsymbol{x}_n)$ can be obtained as the combination $\phi_2(\boldsymbol{x}_n) + 2\phi_3(\boldsymbol{x}_n)$. Then for any given classifier,

$$y(\boldsymbol{x}) = b + w_1\phi_1 + w_2\phi_2 + w_3\phi_3 \tag{93}$$

adding a term

$$\alpha w\phi_1 - w(\phi_2 + 2\phi_3) \tag{94}$$

will not change the regression of the training points, for any $w \in \mathbb{R}$, since at all those training points we have

$$\phi_1(\boldsymbol{x}_n) = \phi_2(\boldsymbol{x}_n) + 2\phi_3(\boldsymbol{x}_n) \tag{95}$$

In particular, we could take w arbitrarily large and get a classifier that "fits" the training data perfectly well. The issue however is that for new data (or test data), the two classifiers will give very different results. In practice, we therefore want to avoid such situations and we will add a penalty on the coefficients $(w_1, \dots, w_N)$ and bias b.

8.1 Regularizers

There are three popular approaches at regularizing the linear regression problem (88).

- ℓ_2 (Ridge regression). Here we simply minimize the sum of squared weights (squared ℓ_2 norm),

$$\underset{\boldsymbol{w},b}{\text{argmin}} \quad \sum_{n=1}^{N} \left|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 + \lambda \sum_{i=1}^{D} |\boldsymbol{w}_i|^2 \tag{96}$$

- ℓ_1 (LASSO). Here we minimize the sum of the absolute value of the weights,

$$\underset{\boldsymbol{w},b}{\text{argmin}} \quad \sum_{n=1}^{N} \left|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 + \lambda \sum_{i=1}^{D} |w_i| \tag{97}$$

- Finally in Best Subset Selection (which can be throught of as a minimization of the number of non zero weights), we found for each K, the best size-K subset of regression coefficients $w_1, \dots, w_N$.

In all of those approaches, the optimal choice of λ or for best subset selection, the optimal size of the subset of regression coefficients, is fixed through cross validation, by computing the regression coefficients associated to a particular value of λ for a subset $\mathcal{S}_1$ of the whole dataset, then testing the model on the remaining $\mathcal{S} \setminus \mathcal{S}_1$ data. And then selecting the λ that leads to the smallest generalization error. The two formulations (96) and (97) both admit constrained variations,

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} \left|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 \tag{98}$$

$$\text{subject to} \quad \sum_{i=1}^{D} |\boldsymbol{w}_i|^2 \leq t \tag{99}$$

and

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} \left|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 \tag{100}$$

$$\text{subject to} \quad \sum_{i=1}^{D} |\boldsymbol{w}_i| \leq t \tag{101}$$

respectively.

Finally, note that other regularization terms are possible. In particular, any ℓ_p norm, $p \geq 1$

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{D} w_i^p\right)^{1/p} \tag{102}$$

can be used as a regularizer.

In the case of the ridge regression formulation (96), it remains possible to derive a closed form solution as follows. We consider the unconstrained formulation, take the derivatives with respect to $(\boldsymbol{w}, b)$ and set those derivatives to 0, we get

$$\boldsymbol{w}^{\text{ridge}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{t} \tag{103}$$

or in the case of feature vectors $\boldsymbol{\phi}(\boldsymbol{x})$,

$$\boldsymbol{w}^{\text{ridge}} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} \tag{104}$$

Hence you see that we have replaced the matrix $(\boldsymbol{\Phi}^T\boldsymbol{\Phi})$ by a better conditioned version of this matrix as it is now combined to the identity which is invertible.

9 Kernels

Let $\beta_0 \in \mathbb{R}$ to denote the bias and $\boldsymbol{\beta}_1 \in \mathbb{R}^D$ to denote the vector of weights. In practice, we often have to deal with prototypes, $\{\boldsymbol{x}_\mu\}_{\mu=1}^N$ which are not linearly separable in their original space. One approach then consists in introducing features and to "map" the original prototypes $\boldsymbol{x}_\mu$ into a space (the feature space) in which those prototypes become linearly separable. We use ϕ to denote the underlying transformation so that for a given datapoint $\boldsymbol{x}$, $\phi(\boldsymbol{x})$ denotes the feature vector of $\boldsymbol{x}$.

Example 1. *Consider the dataset shown in Fig. 2 (left). In the original $\mathbb{R}^2$ space, the data given by pairs (x_1, x_2) is clearly not linearly separable. However, it is possible to introduce a transformation $\phi : \boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ defined as $\phi(\boldsymbol{x}) = \phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$.*

Here, introducing an additional dimension, and setting this dimension to be equal to the radius of the points in the original space, results in the purple points being placed above the red points, thus leaving some space for a separating plane.

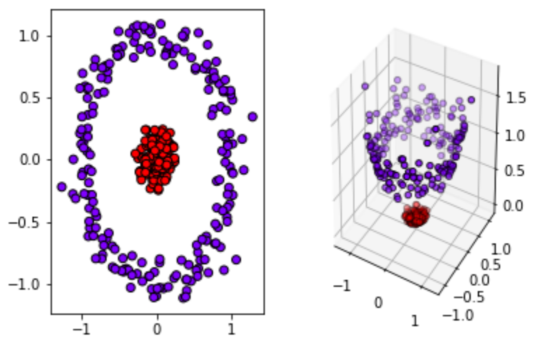


Figure 2: From non linearly separable data to linearly separable data via higher dimensional feature space

Relying on feature vectors to learn classifiers is at the core of machine learning. However, such an approach often requires to map the data to a much higher dimensional space (i.e higher than the number of prototypes $\boldsymbol{x}_\mu$). In this case, it is more interesting to avoid computing the feature vectors explicitely and to rely instead on a measure of similarity between the points in the feature space. After all, the important information here is really how the points compare to each other in the feature space. This information is encoded through kernels

Mathematically, a kernel is nothing else than a function that measure similarity between points. This is summarized by the definition below

Definition 14 (Kernel)**.** *We define a kernel to be the real valued function of two arguments $\boldsymbol{x}$ and $\boldsymbol{x}'$ from a space $\mathcal{X}$. Typically this function is taken to be symmetric $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$ and non negative, $k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$. Those two properties enable us to interpret the kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ as a measure of similarity between points.*

Examples of popular kernels are listed below

- The Gaussian kernel
$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}'))$$
- When the covariance matrix $\boldsymbol{\Sigma}$ is isotropic/spherical, we get the isotropic kernel
$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2})$$
- The cosine similarity:
$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{\boldsymbol{x}^T\boldsymbol{x}'}{\|\boldsymbol{x}\|_2\|\boldsymbol{x}'\|_2}$$
- The (non stationnary) polynonial kernel:
$$k(\boldsymbol{x}, \boldsymbol{x}') = (\gamma\boldsymbol{x}^T\boldsymbol{x}' + r)^M$$
where $r > 0$. This kernel corresponds to a feature vector $\phi(\boldsymbol{x})$ that contains all the monomials up to degree M.

The isotropic kernel is an example of Radial basis function (a function whose distance decreases or increases monotonically with respect to a central point). Those functions are multivariate (i.e they are defined on $\boldsymbol{x} \in \mathbb{R}^D$) but they reduce to a scalar function of the Euclidean norm $\|\boldsymbol{x}\|^2$ of their argument $\boldsymbol{x}$, i.e.,

$$F(\boldsymbol{x}) = f(\|\boldsymbol{x}\|_2) = \phi(r), \quad \boldsymbol{x} \in \mathbb{R}^D \tag{105}$$

Examples of radial basis functions include

- The Gaussian RBF: $F(\|\boldsymbol{x}-\boldsymbol{x}'\|) = \exp(-\alpha^2\|\boldsymbol{x}-\boldsymbol{x}'\|^2)$
- The Multiquadratic RBF: $F(\|\boldsymbol{x}-\boldsymbol{x}'\|) = \sqrt{1+\alpha^2\|\boldsymbol{x}-\boldsymbol{x}'\|^2}$

As soon as one has access to the feature vector $\phi(\boldsymbol{x})$ (for example when this feature is finite dimensional), one can build a valid kernel by defining $k(\boldsymbol{x},\boldsymbol{x}')$ as the inner product of the feature vectors of $\boldsymbol{x}$ and $\boldsymbol{x}'$, i.e.,

$$k(\boldsymbol{x},\boldsymbol{x}') = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x}') \tag{106}$$

9.1 From linear regression to kernel regression

When the dataset is not linearly separable in the original space, we turn to a formulation on the feature space and try to find the plane defined by $(\beta_0,\boldsymbol{\beta}_1)$ that minimizes $J(\boldsymbol{\beta})$, i.e.,

$$\min J(\boldsymbol{\beta}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\boldsymbol{\beta}_1^T\boldsymbol{\phi}(\boldsymbol{x})+\beta_0-t_n\right\}^2+\frac{\lambda}{2}\|\boldsymbol{\beta}\|^2 \tag{107}$$

Here the separating plane is defined through the vector of parameters $\boldsymbol{\beta}=(\beta_0,\boldsymbol{\beta}_1)$, $\beta_0\in\mathbb{R}$, $\boldsymbol{\beta}_1\in\mathbb{R}^D$. The solution for β can be obtained by computing the derivative of $J(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ and settting it to zero. For pairs $\{(\boldsymbol{x}_n,t_n)\}_{n=1}^N$, we have

$$\frac{\partial J(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}}=0\Rightarrow\boldsymbol{\beta}=-\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\boldsymbol{\beta}^T\tilde{\boldsymbol{\phi}}(\boldsymbol{x}_n)-t_n\right\}\tilde{\boldsymbol{\phi}}(\boldsymbol{x}_n) \tag{108}$$

Now introducing the notation a_n for the quantities

$$a_n=-\frac{1}{\lambda}\left\{\boldsymbol{\beta}^T\tilde{\boldsymbol{\phi}}(\boldsymbol{x})-t_n\right\} \tag{109}$$

Then the solution for $\boldsymbol{\beta}$ can read as

$$\boldsymbol{\beta}=\sum_{n=1}^{N}a_n\boldsymbol{\phi}(\boldsymbol{x}_n)=\boldsymbol{\Phi}^T\boldsymbol{a} \tag{110}$$

Where $\boldsymbol{\Phi}$ is the matrix whose i^{th} row is given by $\boldsymbol{\phi}^T(x_i)$. Now substituting this expression into (108), we get

$$J(\boldsymbol{a})=\frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a}-\boldsymbol{a}\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{t}+\frac{1}{2}\boldsymbol{t}^T\boldsymbol{t}+\frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a} \tag{111}$$

From the discussion above, you see that the product $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ is actually encoding all inner products $\langle\boldsymbol{\phi}(\boldsymbol{x}_n),\boldsymbol{\phi}(\boldsymbol{x}_m)\rangle$ and we can thus replace the product $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ with the Kernel $K(\boldsymbol{x},\boldsymbol{x}')$. Doing this yields an objective, or energy function that only depends on the kernel $K(\boldsymbol{x},\boldsymbol{x}')$,

$$J(\boldsymbol{a})=\frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a}-\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t}+\frac{1}{2}\boldsymbol{t}^T\boldsymbol{t}+\frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a} \tag{112}$$

We can then solve for $\boldsymbol{a}$ or simply substitute the expression (110) we found for $\boldsymbol{\beta}$ in (109), to find

$$\boldsymbol{a}=(\boldsymbol{K}+\lambda\boldsymbol{I})^{-1}\boldsymbol{t} \tag{113}$$

Once we have this expression (note that so far everything can be expressed with the Kernel), the classifier simply reads as

$$y(\boldsymbol{x})=\boldsymbol{\beta}^T\boldsymbol{\phi}(\boldsymbol{x})=\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x})=\sum_{n=1}^{N}a_n\boldsymbol{K}(\boldsymbol{x}_n,\boldsymbol{x}) \tag{114}$$

10 Support Vector Machines (SVM)

Support vector machine, which are also known as sparse kernel machines, or Maximal margin classifiers, extend the notion of separating hyperplane to find a separating plane that reduces as much as possible the "risk" of misclassifying new points. To achieve this, the separating plane is defined as the plane maximizing the distance to its closest point(s).

Using the discussion from section 7.1, we can then write the derivation of such maximal margin hyperplane as

$$\underset{\mathbf{w},\mathbf{b}}{\text{argmax}} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n(\mathbf{w}^{\mathbf{T}}(\phi(\boldsymbol{x}_{\mathbf{n}}) + \mathbf{b})) \right] \right\} \tag{115}$$

That is we look for the $\boldsymbol{w}$ and b (which define the plane) such that the distance of the closest point (minimization on $\boldsymbol{x}_n$) to the plane is maximized.

Formulation (115) is invariant under any rescaling of the pair $(\mathbf{w}, \mathbf{b})$. I.e as $\mathbf{w}$ and/or b both appear at the numerator and at the denominator, replacing those quantities by $(\alpha\mathbf{w}, \alpha\mathbf{b})$ does not affect the solution. In fact any such pair, for any α will be a valid solution.

In order to simplify the formulation, we can therefore fix the value of α and decide to take this value equal to the value satisfying

$$t_n(\mathbf{w}^{\mathbf{T}}\phi(\boldsymbol{x}_{\mathbf{n}}) + \mathbf{b}) = \mathbf{1} \tag{116}$$

for the closest point $\boldsymbol{x}_n$. By assumption, any other point, must satisfy $t_n(\mathbf{w}^{\mathbf{T}}\phi(\boldsymbol{x}_{\mathbf{n}}) + \mathbf{b}) \geq \mathbf{1}$, and the problem reads as

$$\begin{aligned} &\underset{\mathbf{w},\mathbf{b}}{\text{argmin}} && \frac{1}{2}\|\mathbf{w}\|^{\mathbf{2}} \\ &\text{subject to} && t_n(\mathbf{w}^{\mathbf{T}}\phi(\boldsymbol{x}_{\mathbf{n}}) + \mathbf{b}) \geq \mathbf{1} \end{aligned} \tag{117}$$

Note that we have used $\underset{x}{\text{argmax}}\|\boldsymbol{x}\|^{-1} = \underset{x}{\text{argmin}}\|\boldsymbol{x}\|$. Problem (117) is a quadratic program that is convex, hence, has a single basin of attraction and can be solved efficiently.

To solve this problem, we write is as an unconstrained formulation (Lagrangian) by introducing multipliers, λ_i for each constraints (see section 6.2)

$$L(\mathbf{w}, \mathbf{b}, \lambda) = \frac{\mathbf{1}}{\mathbf{2}}\|\mathbf{w}\|^{\mathbf{2}} - \sum_{\mathbf{n=1}}^{\mathbf{N}} \lambda_{\mathbf{n}} \left\{ \mathbf{t_n}(\mathbf{w}^{\mathbf{T}}\phi(\boldsymbol{x}_{\mathbf{n}}) + \mathbf{b}) - \mathbf{1} \right\} \tag{118}$$

To find the solutions for $\mathbf{w}$ and b, we compute the derivatives of L with respect to those variables and set the derivatives to 0. From this, we get

$$\mathbf{w} = \sum_{n=1}^{N} \lambda_n t_n \phi(\boldsymbol{x}_n) \tag{119}$$

$$0 = \sum_{n=1}^{N} \lambda_n t_n \tag{120}$$

Now recall that the general form of the classifier we are after is given by

$$y(\boldsymbol{x}) = \mathbf{w}^{\mathbf{T}}\boldsymbol{\phi}(\boldsymbol{x}) + \mathbf{b} \tag{121}$$

If we substitute the expression for $\mathbf{w}$ that we derived from the zero derivatives in the expression of the classifier, we get

$$\begin{aligned} y(\mathbf{x}) &= \sum_{n=1}^{N} \lambda_n t_n \langle \boldsymbol{\phi}(\boldsymbol{x}_n), \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b \\ &= \sum_{n=1}^{N} \lambda_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b \end{aligned} \tag{122}$$

Now besides the maximization of the margins, the other key idea of SVM is the fact that we don't need to keep track of all the training samples in (122). Indeed from Theorem 4, we know that the solution $(\mathbf{w}, \mathbf{b})$ of problem (117) has to satisfy the conditions

$$\lambda_n \geq 0 \tag{123}$$

$$t_n y(\boldsymbol{x}_n) - 1 \geq 0 \tag{124}$$

$$\lambda_n (t_n y(\boldsymbol{x}_n) - 1) = 0 \tag{125}$$

From the last condition in particular, we can see that for every training sample, $n = 1, \dots, N$, either we have $\lambda_n = 0$ or we have $t_n y(\boldsymbol{x}) - 1 = 0$. The second case corresponds to points that are located the closest to the maximal margin hyperplane as the distance to each points to the hyperplane is always lower bounded by

$$\frac{t_n y(\boldsymbol{x}_n)}{\|\mathbf{w}\|} \geq \frac{1}{\|\mathbf{w}\|} \tag{126}$$

In other words, for every training point that appears in (122):

- Either that point does not contribute to the expression (122) of the classifier (as $\lambda_n = 0$)
- Or it is lying on the margins.

Using this idea to simplify (122), if we introduce the notation $\mathcal{S}$ to denote the set of "support vectors", that is the points that are lying on the margins, or located the closest to the plane, the expression of the classifier reduces to

$$y(\boldsymbol{x}) = \sum_{n \in \mathcal{X}} \lambda_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b \tag{127}$$

Furthermore, we can use condition (125) for the support vectors, (i.e. those for which we have $t_n(y(\boldsymbol{x}_n) - 1) = 0$) to write b. indeed note that substituting (127) into condition (125) gives

$$t_n \left(\sum_{n \in \mathcal{S}} \lambda_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b \right) = 1, \quad \text{for every } \boldsymbol{x}_n \in \mathcal{S} \tag{128}$$

To obtain a robust estimate of b, we can then just sum all those equation and divide by N_S. This gives

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{n' \in \mathcal{S}} \lambda_{n'} t_{n'} k(\boldsymbol{x}', \boldsymbol{x}_n) \right) \tag{129}$$

11 Neural Networks

The general formulation of a (two hidden layers) neural network is as follows,

$$y_k = y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma^{(2)} \left(\sum_{j=1}^{M} w_{kj}^{(2)} \sigma^{(1)} \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{130}$$

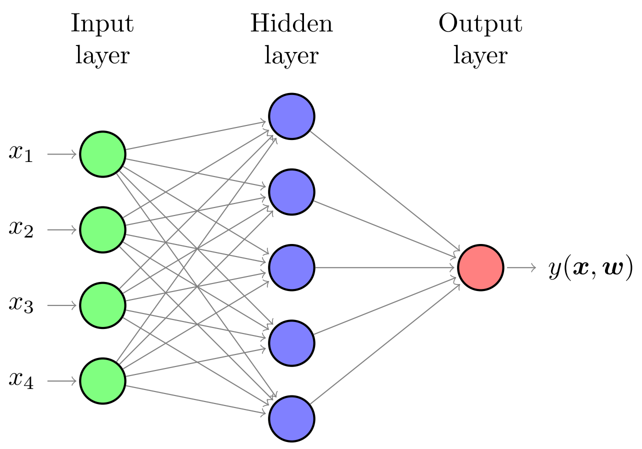


Figure 3: Graphical representation of a one layer neural network. The arrows represent multiplication by the weights w_{ij}. The hidden units (shown in blue) represent the application of the activation function.

When training neural networks, we usually don't take all the training samples into account but rather rely on a small subset of those training samples at each gradient iteration. These subsets are known as minibatches. There are three main approaches at training a network

- Batch gradient descent. Here all the training samples are taken into account and the weight are updated only after all the error has been evaluated at all the training samples (Batch gradient descent is just the regular gradient descent)
- Stochastic gradient descent (SGD). Here, the algorithm computes the error for a given training sample and update the weights immediately after (i.e the weights are updated for each training sample)
- Finally, the tradeoff between batch gradient descent and SGD, MiniBatch gradient descent updates the weight after evaluating the error/loss on a subset of the training samples.

Among the possible regularization approaches for neural networks, the most popular are the following

- Activity regularization. For a given network, we call activations, the outputs of the activation functions, i.e $\sigma^{(\ell)}(\cdot)$. Placing a regularizer on the activation will result in training a network in which only a fraction of the neurons fire at the same time (at least for data similar to the training data).
- Weight regularization. Here we add a penalty (e.g. ℓ_2 or ℓ_1 norm) on the weights.
- Early stopping. When training neural networks, one often observe a decrease in both the training and validation error for the first iterations. Then, after a sufficient number of gradient iterations, the validation error starts increasing [2]. One way to avoid overfitting is then to get the values of the parameters at each iteration and return the values of the parameters that give the lowest validation error (we stop the iterations when no improvement on the validation error has been obtained for some iterations)
- Dropout, which shares some similarities with activity regularizationin fact it can be considered as some sort of subset selection alternative to the ℓ_1 or ℓ_2 penalties used in activity regularization, can be considered as optimizing over a family of "sparse" networks. When training a network with dropout, we select some random subset of the neurons and only update the weights associated to those neurons. Mathematically, when performing SGD, we thus compute the forward pass (i.e the output of the network) by considering a sparse

version of the original network [3]. We choose a probability p. Recall that for a traditional neural network, the activity of each unit can be defined as

$$z_i^{(\ell+1)} = \langle \boldsymbol{w}_i^{(\ell+1)}, \boldsymbol{y}\rangle + b_i^{(\ell+1)} \tag{131}$$

$$y_i^{(\ell+1)} = \sigma(z_i^{\ell+1}) \tag{132}$$

With dropout propagation through the network now reads as

$$r_j^{(\ell)} \sim \text{Bernoulli}(p)^1 \tag{133}$$

$$\tilde{\boldsymbol{y}}^{(\ell)} = \boldsymbol{r}^{(\ell)} \odot \boldsymbol{y}^{(\ell)} \tag{134}$$

$$z_i^{(\ell+1)} = \boldsymbol{w}_i^{(\ell+1)} \tilde{\boldsymbol{y}}^{(\ell)} + b_i^{(\ell+1)} \tag{135}$$

$$y_i^{(\ell+1)} = f(z_i^{(\ell+1)}) \tag{136}$$

So that we select only some of the activations y_j and only update the weights associated to those activations.

- Weight Sharing. The idea of weight sharing is especially meaningful in convolutional neural networks (CNN) where filters are applied to an image to extract information from this image. As the name indicates, weights sharing requires all the neurons within a particular subgroup to have identical weights. Weights sharing is implicitely present in convolutional neural. In those network, each neuron can be thought as computing a local average across the neighborhood of a pixel. The point of such averages is to extract features from a part of the image. Usually, when using convolutional neural networks, we are interested in detecting a particular object in the input images. Hence the features we are looking for should be the same regardless of the part of the image on which the filter is applied. AS a consequence, the weights are constants across each layers. Equality of the weights can be enforced in a hard (exact equality such as in CNNs) or in a soft way. Soft weight sharing is done by assuming a Gaussian distribution of the weights within a given group of neurons.

  In the simplest framework, we assume that all the weights w_i in the group $\mathcal{G}$ are all relatively close to some mean value μ (with variance σ^2) and we for example assume the distribution of those weights to be Gaussian

$$p(w_i) = \mathcal{N}(w_i|\mu, \sigma^2) \tag{137}$$

  We can make this model slightly more general by assuming that the weights have a high probability to take a few distinct values. In this case, we assume for example that the distribution of those weights is given by a mixture of Gaussian, i.e.,

$$p(w_i) = \sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2) \tag{138}$$

  We then use this distribution as a regularization in the objective used to train the network. I.e, just as we did for the MAP and MLE estimators, we want the training step to select weights that have a high probability in the model (138). This corresponds to maximizing (138) over the weights, or (as we did for the MAP and MLE) to minimizing the negative log likelihood

$$\Omega(w) = -\log(\sum_{j=1}^{M} \mathcal{N}(w|\mu_j, \sigma_j^2)) \tag{139}$$

  When there are several ($N_{\mathcal{G}}$) weights in the group $\mathcal{G}$, we simply sum over the contribution of each weight,

$$\Omega(\boldsymbol{w}) = -\sum_{i=1}^{N_{\mathcal{G}}} \log\left(\sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2)\right) \tag{140}$$

  The loss function that is minimized when training the network is then given by

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \tag{141}$$

11.1 Backpropagation

Consider the two hidden layers NN of (130). The extension of this simpler model to L layers can read as

$$y_k = y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma^{(L)}\left(\sum_{j=1}^{M} w_{kj}^{(L)} \sigma^{(L-1)}\left(\dots \sigma^{(1)}\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) \dots\right) + w_{k0}^{(L)}\right) \tag{142}$$

In order to train such a network, we minimize the loss

$$\sum_{n=1}^{N}\sum_{k=1}^{K} (y_k(\boldsymbol{x}_n) - t_k^n)^2 \tag{143}$$

if we put ourselves in a stochastic gradient descent framework, where we only consider a single sample at each iteration, we can drop the sum over n in (143) and consider the minimization

$$L = \sum_{k=1}^{K} (y_k(\boldsymbol{x}_n) - t_k^n)^2 \tag{144}$$

In order to apply one SGD iteration on this objective, and to update the weights

$$w_{ij}^{\ell} \leftarrow w_{ij}^{\ell} - \eta \frac{\partial L}{\partial w_{ij}^{\ell}} \tag{145}$$

we need to compute the expression of the derivatives $\frac{\partial L}{\partial w_{ij}^{\ell}}$. To do so, we will rely extensively on the chain rule.

Computing the derivative with the outer weights (the weights that are the closes to the output of the network) is easy. But computing the derivatives with respect to the most inner weights, i.e. w_{ij}^1, can be quite involved.

Backpropagation is an efficient way to compute the gradient of neural networks. First note that

$$\frac{\partial L}{\partial y_k} = 2(y_k - t_k) \tag{146}$$

We then want to compute all derivatives of the form

$$\frac{\partial L}{\partial w_{kj}^{\ell}} \tag{147}$$

For this, we use the chain rule a first time, splitting (147) into

$$\frac{\partial L}{\partial w_{kj}^{\ell}} = \frac{\partial L}{\partial a_k^{\ell}} \frac{\partial a_{kj}^{\ell}}{\partial w_{kj}^{\ell}} \tag{148}$$

that is we start from the weight we want to compute and we go up one step towards the output of the network by considering the quantitites

$$a_k^{\ell} = \sum_{j=1}^{M} w_{k,j}^{\ell} z_j^{\ell-1} \tag{149}$$

where $z_j^{\ell-1} = \sigma^{\ell-1}(\ldots)$.

Note that the second factor in (148) is simply $z_j^{\ell-1} = \sigma^{\ell-1}(\ldots)$ which is computed when we pass the data through the network. To get the first factor, we apply the chain rule again, going one more level towards the output of the network, y_k. We have

$$\frac{\partial L}{\partial a_k^\ell} = \sum_j \frac{\partial L}{\partial a_j^{\ell+1}} \frac{\partial a_j^{\ell+1}}{\partial a_k^\ell} \tag{150}$$

This last equation is the equation that really encodes backpropagation. To see this, first note that from (142), we have

$$\begin{aligned} a_j^{\ell+1} &= \sum_m w_{jm}^\ell z_m &(151)\\ &= \sum_m w_{jm}^\ell \sigma^\ell(a_m) &(152) \end{aligned}$$

and we can thus write

$$\frac{\partial a_j^{\ell+1}}{\partial a_k^\ell} = (\sigma^\ell)' w_{jm}^\ell \tag{153}$$

Putting this back into (150), we get

$$\frac{\partial L}{\partial a_k^\ell} = \sum_j \frac{\partial L}{\partial a_j^{\ell+1}} (\sigma^\ell)' w_{jm}^\ell \tag{154}$$

Now let $\delta_k^\ell = \frac{\partial L}{\partial a_k^\ell}$. From (154), we can write

$$\delta_k^\ell = \sum_j \delta_j^{\ell+1} (\sigma^\ell)' w_{jm}^\ell \tag{155}$$

From (155), you see that the $\delta^{\ell+1}$ can be "backpropagated" through the network by multiplication with the weights, to obtain the δ^ℓ (we move from the output a^L to the most inner part of the network, a^1, hence the term "backpropagation"). Once we have computed all the δ^ℓ, the derivatives are simply given by using (148) and noting that $\frac{\partial a_k^\ell}{\partial w_{kj}^\ell} = z_j^{\ell-1}$ (Eq. (149)).

The "backpropagation" algorithm can thus read as follows

- First send the training data through the network and compute all the $z_j^\ell(\ldots)$ which are the outputs of every neuron.
- Then compute all the intermediate derivatives $\left[\sigma^\ell(a^{\ell-1})\right]'$ a the activations $a^{\ell-1}$ given by your training point.
- Once you have all those values, start backpropagating the δ with

$$\begin{aligned} \delta^L = \frac{\partial L}{\partial a_k^L} &= \frac{\partial (y_k - t_k)^2}{\partial y_k} \frac{\partial y_k}{\partial a_k^L} &(156)\\ &= 2(y_k - t_k)\left[\sigma^L(a_k^L)\right]' &(157) \end{aligned}$$

  and then

$$\delta_k^\ell = \sum_j \delta_j^{\ell+1} (\sigma^\ell)' w_{jm}^\ell \tag{158}$$

- Finally, once you have all the δ_k^ℓ, compute the derivatives as

$$\frac{\partial L}{\partial w_{kj}^\ell} = \frac{\partial L}{\partial a_k^\ell} z_j^\ell \tag{159}$$

11.2 Clustering Algorithms

Discuss empty clusters

When the total number of clusters is known to be K and one of the clusters ends up being empty. In this case we restart the initialization by taking the centroid to be for example the point that is located the farthest away from all the centroids. Proceeding like this will eliminate the points that contributes the most to error. Another approach is restart the algorithm by keeping the non zero clusters and resampling a centroid from the cluster that has the largest intra cluster SSE. This will reduce the general error.

The process can be repeated when there are multiple empty clusters.

References

[1] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[3] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.