Machine Learning 1

Professor Erik Bekkers
LaTeX by Samuele Punzo
University of Amsterdam

Winter Semester

Contents

Chapter 1

Intro and theory recap

1.1 Introduction

> ”A computer program is said to learn from **experience E** with respect to some class of **tasks T**, and **performance P**, if its performance at tasks in T, as measured by P, improves with experience E.” — Tom Mitchell

There are many different kinds of Machine Learning algorithms, depending on the nature of the task T, the nature of the performance measure P, and the nature of the training signal or experience E.
Some examples of these tasks are: Regression, Clustering and classification, but many other exists (e.g image recognition ecct...)

The machine learning algorithms divide roughly in 3 areas:

- **Supervised Learning** is the most common form of ML, in this problem the training data comprises input vectors ($x \in X$) along with their corresponding target vector ($y \in Y$). Cases in which the aim is to assign each input vector to one of a finite number of discrete categories are called $classification$ problems, if the desired output consists of one or more continous variables, then the task is called $regression$.

- **Unsupervised Learning** In these type of problems the training data consists of a set of input vectors without any corresponding target vector, so the goal is to discover groups of similar examples within the data (*clustering*), or to determine the distribution of data within the input space (*density estimation*), or to project the data from a high-dimensional space down to two or three dimensions (*dimensionality reduction*).

- **Reinforcement Learning** These techniques are concerned with the problem of finding suitable actions to take, in a given situation, in order to maximize a reward. Here the learning algorithm is not given examples of optimal outputs, in contrast to supervised learning, but must instead discover them by a process of trial and error. Typically there is a sequence of states and actions in which the learning algorithm is interacting with its environment. In many cases, the current action not only affects the immediate reward but also has an impact on the reward at all subsequent time steps.

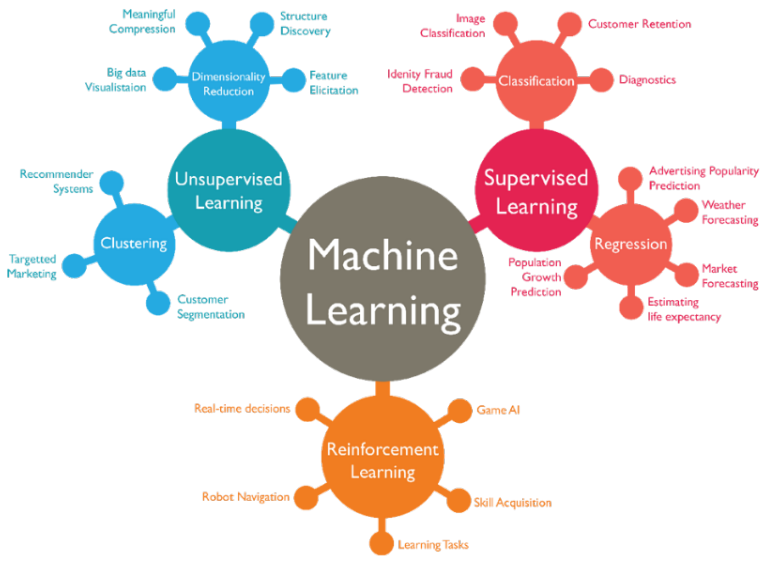


Figure 1.1: Machine Learnng areas

In general we will use a large set of N input vectors $x_1, ..., x_N$ called *training set* to tune the parameters of an adaptive model. In *supervised learning* these vectors are labeled, so we dispose of a *target vector* containing all the input vector's label. The result of training a machine learning algorithm can be expressed as a function $y(x)$ which takes a new sample x as input and generates a label as output, encoded in the same way as in the target vector. The ability to correctly categorize new examples that differ from those used for training is known as *generalization*. In practical applications, the variability of the samples will be such that the training data can comprise only a tiny fraction of all possible samples, and so generalization is a central goal in pattern recognition.

Note that for most practical applications, the original input variables are typically *preprocessed* to transform them into some new space of variables, where, it is hoped, the pattern recognition problem will be easier to solve. pre-processing might also be performed in order to speed up computation.

Let's now introduce an example of a simple regression problem. Suppose that we observe a real-valued input variable x and wish to use this observation to predict the value of a real-valued target variable t. The data for this example are generated from the function $sin(2\pi x)$ with random noise included in the target values. Now suppose that we are given a training set comprising N observations of x, written as $\mathbf{x} = (x_1, ..., x_n)^T$, together with corresponding observations of the values of t, denoted $\mathbf{t} = (t_1, ..., t_n)^T$. The input data set $\mathbf{x}$ in was generated by choosing values of x_i, for $i = 1, ..., N$, spaced uniformly in range $[0, 1]$, and the target data set $\boldsymbol{t}$ was obtained by first computing the corresponding values of the function $sin(2\pi x)$ and then adding a small level of random noise having a Gaussian distribution. Our goal is to exploit this training set in order to make predictions of the value

$\hat{t}$ of the target variable for some new value $\hat{x}$ of the input variable (spoiler: this involves implicitly trying to discover the underlying function $sin(2\pi x)$). There are many ways to do this but for the moment we shall consider a simple approach based on curve fitting.

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{1.1}$$

Where M is the order of the polynomial.
Note that, although the polynomial function $y(x, w)$ is a nonlinear function of x, it is a **linear function of the coefficients w**. Functions, such as the polynomial, which are linear in the unknown parameters have important properties and are called **linear models**. The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing an error function that measures the error between the function values $y(x, w)$, for any given value of w, and the training set data points. One simple choice of error function, which is widely used, is given by the sum of the squares of the errors between the predictions $y(x_i, w)$ for each data point x_i and the corresponding target values t_i, so that we minimize:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=i}^{N} \{y(x_i, \mathbf{w}) - t_i\}^2 \tag{1.2}$$

We can solve the curve fitting problem by choosing the value of **w** for which $E(\mathbf{w})$ is as small as possible. Because the error function is a **quadratic function** of the coefficients **w**, its derivatives with respect to the coefficients will be linear in the elements of **w**, and so the minimization of the error function has a unique solution, denoted by **w***, which can be found in **closed form**. The resulting polynomial is given by the function $y(x, \mathbf{w*})$.
There remains the problem of choosing the order M of the polynomial, and as we shall see this will turn out to be an example of an important concept called model comparison or **model selection**.

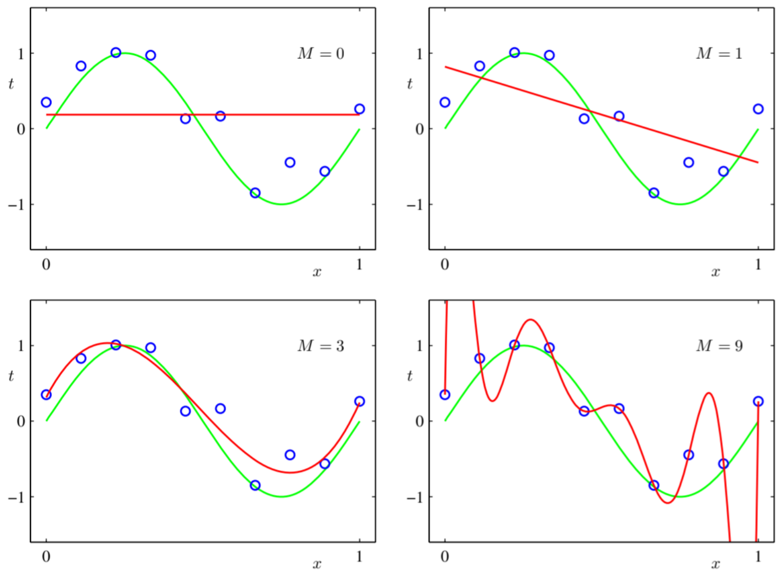


Figure 1.2: Polynomials having various orders M (red curves), fitted to the data set

We notice that the constant ($M = 0$) and first order ($M = 1$) polynomials give rather poor fits to the data and consequently rather poor representations of the function $sin(2\pi x)$. The third order ($M = 3$) polynomial seems to give the best fit to the function $sin(2\pi x)$ of our samples. When we go to a much higher order polynomial ($M = 9$), we obtain an excellent fit to the training data. In fact, the polynomial passes exactly through each data point and $E(\mathbf{w*}) = 0$. However, the fitted curve oscillates wildly and gives a very poor representation of the function $sin(2\pi x)$. This latter behaviour is known as **over-fitting**.

1.2 Probability Theory

"Probability theory is nothing but common sense reduced to calculation." — Pierre Laplace

There are actually two different interpretations of probability. One is called the **frequentist** interpretation. In this view, probabilities represent long run frequencies of **events** that can happen multiple times. The other interpretation is called the **Bayesian** interpretation of probability. In this view, probability is used to quantify **uncertainty** or ignorance about something; hence it is fundamentally related to information rather than repeated trials. One big advantage of the Bayesian interpretation is that it can be used to model our uncertainty about one-off events that do not have long term frequencies.

The uncertainty in our predictions can arise for two fundamentally different reasons. The first is due to our ignorance of the underlying hidden causes or mechanism generating our data. This is called **epistemic uncertainty** (aka model uncertainty). The second kind of uncertainty arises from intrinsic variability, which cannot be reduced even if we collect more data. This is sometimes called **aleatorics uncertainty** (aka data uncertainty).

Probability theory provides a consistent framework for the quantification and manipulation of uncertainty and forms one of the central foundations for pattern recognition. When combined with decision theory, it allows us to make optimal predictions given all the information available to us, even though that information may be incomplete or ambiguous.

Definition 1.2.1: Event

We define an event, denoted by the binary variable A as some state of the world that either holds or does not hold.

The expression $Pr(A)$ denotes the probability with which you believe event A is true. We require that $0 \leq Pr(A) \leq 1$, where $Pr(A) = 0$ means the event definitely will not happen, and $Pr(A) = 1$ means the event definitely will happen.

Definition 1.2.2: Random variables

Suppose X represents some unknown quantity of interest. If the value of X is unknown and/or could change, we call it a **random variable** or **rv**. The set of possible values, denoted X, is known as the **sample space** or **state space**. An **event** is a set of outcomes from a given sample space.

If the sample space X is finite or countably infinite, then X is called a **discrete random variable**. In this case, we denote the probability of the event that X has value x by $Pr(X = x)$. We define the **probability mass function** or **pmf** as a function which computes the probability of events which correspond to setting the rv to each possible value $p(x) \triangleq Pr(X = x)$.

Property 1.2.3: pmf properties

- $0 \leq p(x) \leq 1$
- $\sum_{x \in X} p(x) = 1$.

If $X \in \mathbb{R}$ is a real-valued quantity, it is called a **continuous random variable**. In this case, we can no longer create a finite (or countable) set of distinct possible values it can take on. However, there are a countable number of *intervals* which we can partition the real line into. So, we can associate events with X being in each one of these intervals, using the methods discussed above for discrete random variables.
In general, we define the **cumulative distribution function** or **cdf** or the rv X as follows: $P(x) \triangleq Pr(X \leq x)$ (**note that** we use a capital P to represent the cdf).
Using this, we can compute the probability of being in any interval as follows:

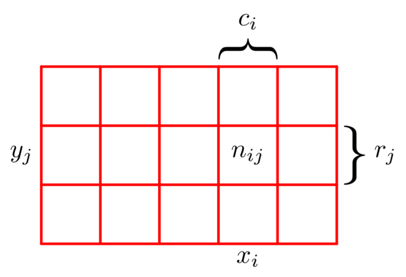
$$Pr(a \leq X \leq b) = P(b) - P(a) \tag{1.3}$$

We define the **probability density function** or **pdf** as the derivative of the cdf: $\frac{\mathrm{d}}{\mathrm{d}x} P(x)$ (**note that** this derivative not always exist, in which case the pdf is not defined).

Given a pdf, we can compute the probability of a continuous variable being in a finite interval as follows:

$$Pr(a < X \leq b) = \int_a^b p(x)dx = P(b) - P(a) \tag{1.4}$$

Let's know take a look at sets of related random variables; suppose, to start, that we have two discrete random variables, X and Y. We shall suppose that X can take any values x_i where $i = 1, ..., M$ and Y can take the values y_j where $j = 1, ..., L$.
Consider a total of N trials in which we sample both of the variables X and Y, and let the number of such trials in which $X = x_i$ and $Y = y_j$ be n_{ij}. Also, let the number of trials in which X takes the values x_i (irrespective of the values Y takes) be denoted by c_i and similarly for Y let the number of trials in which Y takes the values y_j be denoted by r_j.



We can define the **joint distribution** of two random variables using

$$p(x_i, y_j) = p(X = x_i, Y = y_j) \tag{1.5}$$

for all possible values of X and Y.

This probability is given by the number of points falling in the cell i, j as a fraction of the total number of points, and hence $p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$. Similarly, the probability that X takes the value x_i, irrespective of the value of Y is written as $p(X = x_i)$ and it is called **marginal distribution**, this is given by the fraction of the total number of points that fall in column i, so that $p(X = x_i) = \frac{c_i}{N}$. Because the number of instances in column i is just the sum of the number of instances in each cell of that column, we have $c_i = \sum_j n_j$ and therefore:

$$p(X = x_i) = \sum_{j=1}^{L} p(X = x_i, Y = y_j) \tag{1.6}$$

or more in general

$$p(X = x) = \sum_{y} p(X = x, Y = y) \tag{1.7}$$

this is called **sum rule** (or **rule of total probability**).

If we consider only those instances for which $X = x_i$, then the fraction of such instances for which $Y = y_j$ is written as

$$p(Y = y_j|X = x_i) \tag{1.8}$$

and is called the **conditional probability** of $Y = y_j$ given $X = x_i$. It is obtained by finding the fraction of those points in column i that fall in cell i, j and hence is given by $p(Y = y_j|X = x_i) = \frac{n_{ij}}{c_i}$. From the previous formulas we can derive the following relationship:

$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} = p(Y = y_j|X = x_i)p(X = x_i) \tag{1.9}$$

which is knows as the **product rule** or probability.
By extending the product rule to D variables, we get the **chain rule of probability**

$$p(\mathbf{x}_{1:D}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|X_1, x_2, x_3)...p(x_D|\mathbf{x}_{1:D-1}) \tag{1.10}$$

This provides a way to create a high dimensional joint distribution from a set of conditional distributions.

Definition 1.2.4: The Rules of Probability

$$\textbf{sum rule} \qquad p(X) = \sum_{Y} p(X, Y) \tag{1.11}$$

$$\textbf{product rule} \qquad p(X, Y) = p(Y|X)P(X) \tag{1.12}$$

$$\textbf{chain rule} \qquad p(\mathbf{x}_{1:D}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)...p(x_D|\mathbf{x}_{1:D-1}) \tag{1.13}$$

Here $p(X, Y)$ is a joint probability and is verbalized as “the probability of X and Y”. Similarly, the quantity $p(Y|X)$ is a conditional probability and is verbalized as “the probability of Y given X”, whereas the quantity $p(X)$ is a marginal probability.

1.2.1 Bayes's theorem

From the product rule, together with the symmetry property $p(X,Y) = p(Y,X)$, we immediately obtain the following relationship:

$$p(X,Y) = p(Y|X)P(X) \qquad \text{(Product rule)} \tag{1.14}$$

$$p(X,Y) = p(Y,X) \qquad \text{(Symmetry)} \tag{1.15}$$

$$p(Y|X)P(X) = p(X|Y)P(Y) \tag{1.16}$$

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \qquad \text{(Bayes's theorem)} \tag{1.17}$$

"Bayes's theorem is to the theory of probability what Pythagoras's theorem is to geometry." — Sir Harold Jeffreys

Using the sum rule, the denominator in Baye's theorem can be expressed in terms of the quantities appearing in the numerator.

$$p(X) = \sum_{Y} p(X|Y)p(Y) \tag{1.18}$$

We can view the denominator in Baye's theorem as being the normalization constant required to ensure that the sum of the conditional probability on the left-hand side of Equation 1.17 over all values of Y equals one.

Bayes' rule itself is very simple: it is just a formula for computing the probability distribution over possible values of an unknown (or hidden) quantity Y given some observed data $X = x$. In Equation 1.17 the term $p(Y)$ represents what we know about possible values of X before we see any data; this is called **prior probability** (or distribution). The term $p(X|Y = y)$ represents the distribution over the possible outcomes Y we expect to see if $Y = y$; this is called the **observation distribution**. When we evaluate this at a point corresponding to the actual observations, x, we get the function $p(X = x|Y = y)$, which is called the **likelihood**. (Note that this is a function of y, since x is fixed, but it is not a probability distribution, since it does not sum to one.) Multiplying the prior distribution $p(Y = y)$ by the likelyhood function $p(X = x|Y = y)$ for each y gives the unnormalized joint distribution $p(X = x, Y = y)$. We can convert this into a normalized distribution by dividing by $p(X = x)$, which is known as the **marginal likelihood**, since it is computed by marginalizing over the unknown Y:

$$p(X = x) = \sum_{y' \in Y} p(Y = y')p(X = x|Y = y') = \sum_{y' \in Y} p(Y = y', X = x) \tag{1.19}$$

Normalizing the joint distribution by computing $\frac{p(Y=y,X=x)}{p(X=x)}$ for each y gives the **posterior distribution** $p(Y = y|X = x)$; this represent our new **belief state** about the possible values of Y.

We can summarize Bayes rule in words as follows: *posterior* $\propto$ *prior* $\times$ *likelihood*

Here we use the symbol $\propto$ to denote "proportional to", since we are ignoring the denominator, which is just a constant, independent of Y. Using Bayes rule to update a distribution over unknown values of some quantity of interest, given relevant observed data, is called

Bayesian inference, or posterior inference.

Theorem 1.2.5: Bayes rule

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- $p(Y)$ the **prior probability** of $Y = y$
- $p(X = x|Y = y)$ **likelihood**, recall that this IS NOT a probability
- $p(X)$ the **evidence** for $X = x$
- $p(Y = y|X = x)$ the **posterior probability** of $Y = y$

Finally, we note that if the joint distribution of two variables factorizes into the product of the marginals, so that $p(X, Y) = p(X)p(Y)$, then X and Y are said to be **independent**. From the product rule, we see that $p(Y|X) = p(Y)$, and so the conditional distribution of Y given X is indeed independent of the value of X.

1.2.2 Expected value and Variance

One of the most important operations involving probabilities is that of finding weighted averages of functions. The average value of some function $f(x)$ under a probability distribution $p(x)$ is called *expectation* of $f(x)$ and will be denoted by $\mathbb{E}[f]$.
For a discrete distribution, it is given by

$$\mathbb{E}[f] = \sum_x p(x)f(x) \tag{1.20}$$

so that the average is weighted by the relative probabilities of the different values of x. In the case of continuous variables, expectations are expressed in terms of an integration with respect to the corresponding probability density

$$\mathbb{E}[f] = \int p(x)f(x)\,dx \tag{1.21}$$

In either case, if we are given a finite number N of points drawn from the probability distribution or probability density, then the expectation can be approximated as a finite sum over these points

$$\mathbb{E}[f] \simeq \frac{1}{N}\sum_{n=1}^{N} f(x_n) \tag{1.22}$$

Sometimes we will be considering expectations of functions of several variables, in which case we can use a subscript to indicate which variable is being averaged over, so that for instance $\mathbb{E}_x[f(x, y)]$ denotes the average of the function $f(x, y)$ with respect to the distribution of x. (**Note that** $\mathbb{E}_x[f(x, y)]$ will be a function of y).
We can also consider a conditional expectation with respect to a conditional distribution, so that

$$\mathbb{E}_x[f|y] = \sum_x p(x|y)f(x) \tag{1.23}$$

with analogous definition for continuous variables.

Property 1.2.6: Linearity of the Expected value

$$\mathbb{E}[f(x)+g(x)] = \mathbb{E}[f(x)] + \mathbb{E}[g(x)] \tag{1.24}$$

$$\mathbb{E}[cf(x)] = c\,\mathbb{E}[f(x)] \tag{1.25}$$

$$\mathbb{E}[c] = c \tag{1.26}$$

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i] \tag{1.27}$$

If we have N independent variables then:

$$\mathbb{E}[\prod_{i=1}^{n} X_i] = \prod_{i=1}^{n} \mathbb{E}[X_i] \tag{1.28}$$

The *variance* of $f(x)$ (often denoted as σ^2) is defined by

$$Var[f] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \tag{1.29}$$

and provides a measure of how much variability there is in f(x) around its mean value $\mathbb{E}[f(x)]$. Expanding out the square, we see that the variance can also be written in terms of the expectations of $f(x)$ and $f(x)^2$.

$$\begin{aligned} Var[f] &= \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \\ &= \mathbb{E}[f(x)^2 - 2f(x)\,\mathbb{E}[f(x)] + \mathbb{E}[f(x)]^2] \\ &= \mathbb{E}[f(x)^2] - 2f(x)\,\mathbb{E}[f(x)]\,\mathbb{E}[f(x)] + \mathbb{E}[f(x)]^2 \\ &= \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2 \end{aligned} \tag{1.30}$$

In particular, we can consider the variance of the variable x itself, which is given by

$$Var[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \tag{1.31}$$

The **standard deviation** is defined as

$$std[X] \triangleq \sqrt{Var[X]} = \sigma \tag{1.32}$$

The s.d is useful since it has the same units as X itself.

Property 1.2.7: Some variance properties

$$Var[cX] = c^2 Var[X] \tag{1.33}$$

$$Var[b] = 0 \textit{ where } b \textit{ is a constant} \tag{1.34}$$

If we have N independent variables then:

$$Var[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} Var[X_i] \tag{1.35}$$

$$\begin{aligned} Var\left[\prod_{i=1}^{n} X_i\right] &= \mathbb{E}\left[(\prod_i X_i)^2\right] - (\mathbb{E}\left[\prod_i X_i\right])^2 \\ &= \mathbb{E}\left[\prod_i X_i^2\right] - (\prod_i \mathbb{E}[X_i])^2 \\ &= \prod_i \mathbb{E}[X_i^2] - \prod_i (\mathbb{E}[X_i])^2 \\ &= \prod_i (Var[X_i] + (\mathbb{E}[X_i])^2) - \prod_i (\mathbb{E}[X_i])^2 \\ &= \prod_i (\sigma_i^2 + \mu_i^2) - \prod_i \mu_i^2 \end{aligned} \tag{1.36}$$

For two vectors of random variables $\mathbf{x}$ and $\mathbf{y}$, the *covariance* is a matrix and it is defined by

$$cov[\mathbf{x}, \mathbf{y}] = \mathbb{E}_{\mathbf{x},\mathbf{y}}[\{\mathbf{x} - \mathbb{E}[\mathbf{x}]\}\{\mathbf{y}^T - \mathbb{E}[\mathbf{y}^T]\}] = \mathbb{E}_{\mathbf{x},\mathbf{y}}[\mathbf{x}\mathbf{y}^T] - \mathbb{E}[\mathbf{x}]\,\mathbb{E}[\mathbf{y}^T] \tag{1.37}$$

If we consider the covariance of the components of a vector x with each other, then we use a slightly simpler notation $cov[\mathbf{x}] \equiv cov[\mathbf{x}, \mathbf{x}]$.

Remember that if two random variables X, Y are independent then their covariance is 0, but the contrary is NOT true.

1.2.3 Gaussian distribution

The most widely used distribution of continuous random variables $y \in \mathbb{R}$ is the **Gaussian distribution**, also called the **normal distribution**.

For the case of a single real-valued variable x, the **pdf** of the Gaussian distribution is defined by

$$\mathcal{N}\left(x|\mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \tag{1.38}$$

which is governed by two parameters: μ the mean, and σ^2 the variance. $\sqrt{2\pi\sigma^2}$ is the normalization constant needed to ensure that the density integrated to 1.
Remember: The square root of the variance σ, is the standard deviation, and the reciprocal of the variance, $\beta = 1/\sigma^2$ is the precision.

The cdf od the Gaussian is defined by

$$\Phi(x;\mu,\sigma^2) \triangleq \int_0^y \mathcal{N}(x|\mu,\sigma^2)\ dx \tag{1.39}$$

Note that the cdf of the Gaussian is often implemented using $\Phi(X;\mu,\sigma^2) = \frac{1}{2}[1 + erf[z/\sqrt{2}]$where $z = (y-\mu)/\sigma$ and $erf(u)$ is the **error function**, defined as

$$erf(u) \triangleq \frac{2}{\pi}\int_0^u e^{-t^2}\ dt \tag{1.40}$$

When $\mu = 0$ and $\sigma = 1$, the Gaussian is called the **standard normal** distribution.

Let's see how to use the pdf to compute the expected value of the distribution:

$$\begin{aligned}
\mathbb{E}_{x\sim\mathcal{N}(x|\mu,\sigma^2)}[x] &= \int_{-\infty}^{\infty} x\mathcal{N}(x|\mu,\sigma^2)\ dx \\
&= \int_{-\infty}^{\infty} x\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\ dx \\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} xe^{-\frac{(x-\mu)^2}{2\sigma^2}} \\
&\text{let's apply the substitution } \ z = \frac{1}{\sqrt{2\pi\sigma^2}}(x-\mu) \ \text{ with } \ dz = \frac{1}{\sqrt{2\pi\sigma^2}}\ dx \\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} \sqrt{2\sigma^2}(\sqrt{2\sigma^2}z+\mu)e^{-z^2}\ dz \\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} (2\sigma^2 z+\sqrt{2\sigma^2}\mu)e^{-z^2}\ dz \\
&\text{now, we split the integral} \\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} 2\sigma^2 ze^{-z^2}\ dz + \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} \sqrt{2\sigma^2}\mu e^{-z^2}\ dz \\
&\text{the left integral is odd so it is 0} \\
&= \frac{\sqrt{2\sigma^2}\mu}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} e^{-z^2}\ dz \qquad \text{recall that } \int_{-\infty}^{\infty} e^{-x^2}\ dx = \sqrt{\pi} \\
&= \frac{\sqrt{2\pi\sigma^2}\mu}{\sqrt{2\pi\sigma^2}} = \mu
\end{aligned} \tag{1.41}$$

The variance is defined as follows:

$$\begin{aligned}
Var[X] \triangleq \mathbb{E}[(X-\mu)^2] &= \int_{-\infty}^{\infty} (x-\mu)^2 p(x)\ dx \\
&= \int_{-\infty}^{\infty} (x^2-2x\mu+\mu^2)p(x)\ dx \\
&= \int_{-\infty}^{\infty} y^2 p(x)\ dx - 2\mu\int_{-\infty}^{\infty} xp(x)\ dx + \mu^2\int_{-\infty}^{\infty} p(x)\ dx \\
&= \mathbb{E}[X^2] - 2\mu^2 + \mu^2 = \mathbb{E}[X^2] - \mu^2
\end{aligned} \tag{1.42}$$

from which we derive the useful result

$$\mathbb{E}[X^2] = \sigma^2 + \mu^2 \tag{1.43}$$

Note that the variance could also be calculated by solving explicitly $\int_{-\infty}^{\infty}(x-\mu)^2\mathcal{N}(x|\mu,\sigma^2)\,dx$

We are also interested in the Gaussian distribution defined over a D-dimensional vector **x** of continuous variables, called the **multivariate Gaussian** or **multivariate normal** (**MVN**). The MVN density is defined by the following:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{1.44}$$

where $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}] \in \mathbb{R}$ is the mean vector, and $\boldsymbol{\Sigma} = Cov[\boldsymbol{x}]$ is the $D \times D$ **covariance matrix**, defined as follows:

$$Cov[\boldsymbol{x}] \triangleq \mathbb{E}[(\boldsymbol{x}-\mathbb{E}[\boldsymbol{x}])(\boldsymbol{x}-\mathbb{E}[\boldsymbol{x}])^T] \tag{1.45}$$

where

$$Cov[X_i, X_j] \triangleq \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])] = \mathbb{E}[X_iX_j] - \mathbb{E}[X_i]\,\mathbb{E}[X_j] \tag{1.46}$$

and $Var[X_i] = Cov[X_i, X_i]$.
From Equation Equation 1.45, we get the important result $\mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] = \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^T$

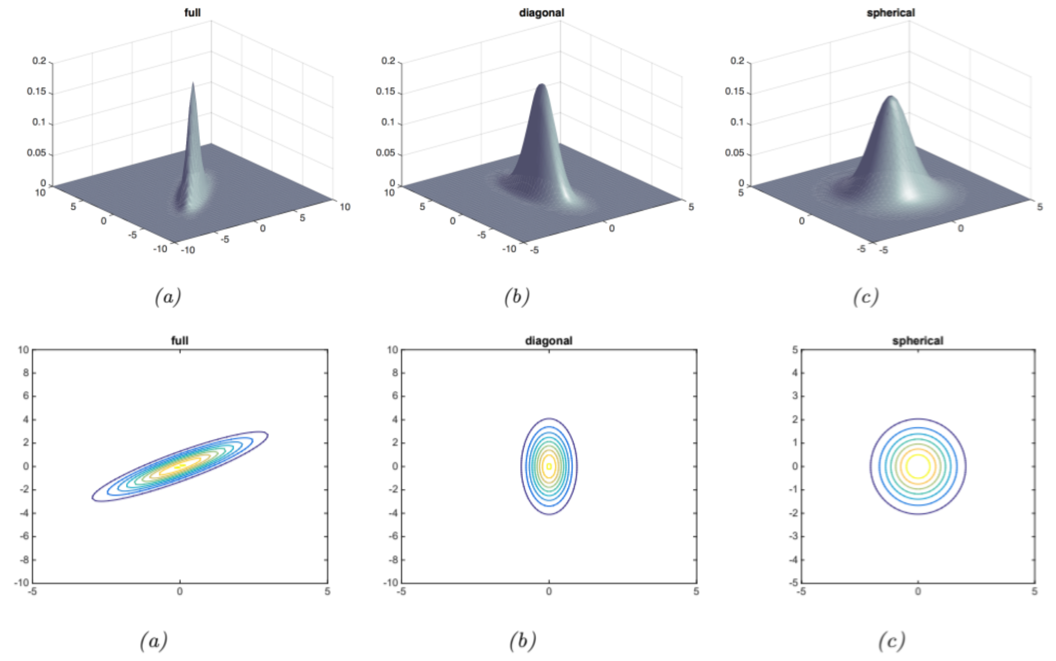


Figure 1.3: Visualization of a 2d Gaussian density as a surface plot. (a) Distribution using a full covariance matrix can be oriented at any angle. (b) Distribution using a diagonal covariance matrix must be parallel to the axis. (c) Distribution using a spherical covariance matrix must have a symmetric shape.

Those figure shows some MVN densiities in 2d for three different kinds of covariance matrices. A **full covariance matrix** (a) has $D(D+1)/2$ parameters, where we divide by 2 since $\mathbf{\Sigma}$ is symmetric. A **diagonal covariance matrix** (b) has D parameters, and has 0s in the off-diagonal terms. A **spherical covariance matrix** (c), also called **isotropic covariance matrix**, has the form $\mathbf{\Sigma} = \sigma^2 \boldsymbol{I}_D$, so it only has one free parameter, namely σ^2.

Marginalization property

Given two random variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ that are jointly Gaussian distributed:

$$p(x_1, x_2) = \mathcal{N}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} | \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \tag{1.47}$$

then the marginals are given by:

$$p(x_1) = \mathcal{N}(x_1|\mu_1, \Sigma_{11})$$
$$p(x_2) = \mathcal{N}(x_2|\mu_2, \Sigma_{22})$$

Conditioning property

Given two random variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ that are jointly Gaussian distributed:

$$p(x_1, x_2) = \mathcal{N}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} | \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \tag{1.48}$$

then the conditional is:

$$p(x_1|x_2) = \mathcal{N}(\mu_{1|2}, \Sigma_{1|2}) \tag{1.49}$$

with

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$
$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

Summing Gaussians

The sum of two **independent** Gaussian random variables $x \sim \mathcal{N}(\mu, \Sigma)$ $y \sim \mathcal{N}(\mu', \Sigma')$ is also a Gaussian random variable

$$z = x + y \quad \rightarrow \quad z \sim \mathcal{N}(\mu + \mu', \Sigma + \Sigma') \tag{1.50}$$

If we have sampled a vector $\boldsymbol{x}$ of **uncorrelated** Gaussian Variables $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and if $\boldsymbol{y} = \boldsymbol{\mu} + \boldsymbol{a}\boldsymbol{x}$ then $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{A}\boldsymbol{A}^T) \quad with \quad \mathbf{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T$ so if we have access to a sampler of uncorrelated Gaussian variables, we can create correlated samples for a given mean $\boldsymbol{\mu}$ and covariance $\mathbf{\Sigma}$.

Note that $\boldsymbol{A}$ can be determined by solving the eigen decomposition

$$\mathbf{\Sigma} = \boldsymbol{U}\mathbf{\Lambda}\boldsymbol{U}^T \quad where \quad \boldsymbol{A} = \boldsymbol{U}\mathbf{\Lambda}^{1/2}$$

1.2.4 Parameter estimation

The most common approach to parameter estimation is to pick the parameters that assign the highest probability to the training data; this is called **maximum likelihood estimation** or **MLE**.

Definition 1.2.8: Maximum Likelihood Estimator

We define the MLE as follows:

$$\hat{\boldsymbol{\theta}} \triangleq \underset{\theta}{argmax}\ p(D|\theta) \tag{1.51}$$

We usually assume the training examples are independently sampled from the same distribution, so the (conditional) likelihood becomes:

$$p(D|\theta) = \prod_{n=1}^{N} p(\boldsymbol{y_n}|\boldsymbol{x_n},\boldsymbol{\theta}) \tag{1.52}$$

This is known as the iid assumption, which stands for "independent and identically distributed". We usually work with the log-likelihood to avoid numerical underflow/overflow problems. The log-likelihood is given by

$$l(\boldsymbol{\theta}) \triangleq \log p(D|\boldsymbol{\theta}) = \log\left(\prod_{n=1}^{N} p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta})\right) = \sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta}) \tag{1.53}$$

This decomposes into a sum of terms, one per example. Thus the MLE is given by

$$\hat{\boldsymbol{\theta}}_{ML} = \underset{\theta}{argmax} \sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta}) \tag{1.54}$$

Since most optimization algorithms are designed to minimize cost functions, we can redefine the **objective function** to be the (conditional) **negative log likelihood** or **NLL**:

$$NLL(\theta) \triangleq -\log p(D|\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta}) \tag{1.55}$$

$$\hat{\boldsymbol{\theta}}_{ML} = \underset{\theta}{argmin} - \log p(D|\boldsymbol{\theta}) = \underset{\theta}{argmin} - \sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n,\boldsymbol{\theta}) \tag{1.56}$$

Suppose $X \sim \mathcal{N}(\mu, \sigma^2)$ and let $D = \{x_i : i = 1 : N\}$ be an iid sample of size N.
We can estimate the parameters $\boldsymbol{\theta} = (\mu, \sigma^2)$ using MLE as follows. First, we derive the log-likelihood, which is given by

$$\begin{aligned} l(\mu, \sigma^2) &= \sum_{i=1}^{N} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)\right] \\ &= \sum_{i=1}^{N}\left(-log\left(\sqrt{2\pi\sigma^2}\right) + \frac{(x_i-\mu)^2}{2\sigma^2}\right) \\ &= -\frac{N}{2} log\left(2\pi\sigma^2\right) + \sum_{i=1}^{N} -\frac{(x_i-\mu)^2}{2\sigma^2} \\ &= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 \end{aligned} \tag{1.57}$$

The maximum (or the minimum for the NLL) of this function must satisfy the following conditions:

$$\frac{\mathrm{d}}{\mathrm{d}\mu} l(\mu, \sigma^2) = 0 \qquad \frac{\mathrm{d}}{\mathrm{d}\sigma^2} l)\mu, \sigma^2) = 0 \tag{1.58}$$

So all we have to do is to find this stationary point.
First for μ:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\mu} l(\mu, \sigma^2) &= -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\frac{\mathrm{d}}{\mathrm{d}\mu}(x_i-\mu)^2 \\ &= -\frac{1}{2\sigma^2}\sum_{i=1}^{N} -2(x_i-\mu) \\ &= \frac{1}{\sigma^2}\left(\sum_{i=1}^{N} x_i - \sum_{i=1}^{N}\mu\right) \\ & \text{now set the derivative to 0} \\ & \frac{1}{\sigma^2}\left(\sum_{i=1}^{N} x_i - \sum_{i=1}^{N}\mu\right) = 0 \\ & \sum_{i=1}^{N} x_i - \sum_{i=1}^{N}\mu = 0 \\ & \sum_{i=1}^{N} x_i - N\mu = 0 \\ & \mu_{ML} = \frac{1}{N}\sum_{i=1}^{N} x_i \end{aligned} \tag{1.59}$$

Then for σ^2:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\sigma^2} l(\mu, \sigma^2) &= -\frac{N}{2}\frac{1}{2\pi\sigma^2}4\pi\sigma + \frac{1}{2\sigma^4}2\sigma\sum_{i=1}^{N}(x_i-\mu)^2 \\
&= -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{N}(x_i-\mu)^2 \\
&\text{we set it to 0} \\
&-\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{N}(x_i-\mu)^2 = 0 \\
&-N\sigma^2 + \sum_{i=1}^{N}(x_i-\mu)^2 = 0 \\
&\sigma^2_{ML} = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2
\end{aligned} \tag{1.60}$$

Biased estimators

An estimator is a procedure applied to data which returns an estimand. Let $\hat{\boldsymbol{\theta}}()$ be the estimator, and $\hat{\boldsymbol{\theta}}(D)$ the estimand. In frequentist statistics, we treat the data as a random variable, drawn from some true but unknown distribution, $p^*(D)$; this induces a distribution over the estimand, $p^*(\hat{\boldsymbol{\theta}}(D))$, known as the sampling distribution. In this section, we discuss two key properties of this distribution, its bias and its variance.

The **bias** of an estimator is defined as

$$bias(\theta(\hat{\cdot})) \triangleq \mathbb{E}\left[\hat{\theta}(D) - \theta^*\right] \tag{1.61}$$

where θ^* is the true parameter value, and the expectation is with reference to "nature's distribution" $p(D|\theta^*)$. If the bias is zero the estimator is called **unbiased**. For example the MLE for a Gaussian mean is unbiased:

$$bias(\hat{\mu}) = \mathbb{E}[\bar{x}-\mu] = \mathbb{E}[\bar{x}]-\mu = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}x_i\right]-\mu = \frac{1}{N}\sum_{i=1}^{N}(\mathbb{E}[x_i])-\mu = \frac{N\mu}{N}-\mu = 0 \tag{1.62}$$

where $\bar{x}$ is the sample mean.

However the MLE for a Gaussian variance, $\sigma^2_{mle} = \frac{1}{N}\sum_{i=1}^{N}(x_n-\bar{x})^2$, is not an unbiased estimator of σ^2. In fact, one can show that

$$\begin{aligned}
\mathbb{E}[\sigma^2_{mle}] &= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^2\right] \\
&= \frac{1}{N}\mathbb{E}\left[\sum_{i=1}^{N}(x_i^2-2x_i\bar{x}+\bar{x}^2)\right] \\
&= \frac{1}{N}\mathbb{E}\left[\sum_{i=1}^{N}x_i^2 - 2\bar{x}\sum_{i=1}^{N}x_i + \sum_{i=1}^{N}\bar{x}^2\right]
\end{aligned}$$

$$
\begin{aligned}
&\text{we can write} \sum_{i=1}^{N} x_i \;\text{ as }\; N\bar{x} \;\text{ and }\; \sum_{i=1}^{N} \bar{x}^2 \;\text{ as }\; N\bar{x}^2 \\
&= \frac{1}{N}\,\mathbb{E}\left[\sum_{i=1}^{N} x_i^2 - 2N\bar{x}^2 + N\bar{x}^2\right] \\
&= \frac{1}{N}\,\mathbb{E}\left[\sum_{i=1}^{N} x_i^2 - N\bar{x}^2\right] \\
&= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}(x_i^2) - \bar{x}^2\right] \\
&= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}(x_i^2)\right] - \mathbb{E}[\bar{x}^2] \\
&= \mathbb{E}\left[x^2\right] - \mathbb{E}[\bar{x}^2] \qquad\qquad \text{we expand the right term} \\
&= \mathbb{E}\left[x^2\right] - \mathbb{E}\left[\left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)\left(\frac{1}{N}\sum_{j=1}^{N} x_j\right)\right] \\
&= \mathbb{E}\left[x^2\right] - \mathbb{E}\left[\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} x_i x_j\right)\right] \\
&\text{we split the sum in two sub-series one where } i == j \;\text{ (so that }\; x_i x_j = x_i^2) \\
&\text{and the other where } i \neq j \\
&= \mathbb{E}\left[x^2\right] - \frac{1}{N^2}\,\mathbb{E}\left[\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbb{1}(i=j)\; x_i x_j \;+\; \sum_{i=1}^{N}\sum_{j=1}^{N} \mathbb{1}(i\neq j)\; x_i x_j\right] \\
&= \mathbb{E}\left[x^2\right] - \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbb{1}(i=j)\;\; \mathbb{E}[x_i x_j] \;+\; \sum_{i=1}^{N}\sum_{j=1}^{N} \mathbb{1}(i\neq j)\;\; \mathbb{E}[x_i x_j] \\
&\text{we derive from covariances that } \mathbb{E}[x_i x_j] = \begin{cases} \mu^2 + \sigma^2 \; if \; i = j \\ \mu^2 \; if \; i \neq j \end{cases} \\
&\text{so from the left sum we obtain } N(\mu^2+\sigma^2) \text{ while from the right one, since we are} \\
&\text{summing the squared expected value of all the elements except for the diagonal} N(N-1)\mu^2 \\
&\text{the sum results in } N(N-1)\mu^2 \text{ we also know that } \mathbb{E}[x^2] = \mu^2+\sigma^2 \text{ so} \\
&= (\mu^2+\sigma^2) - \frac{1}{N}(\mu^2+\sigma^2) \;-\; \frac{1}{N^2}N(N-1)\mu^2 \\
&= \frac{N-1}{N}(\mu^2+\sigma^2) \;-\; \frac{N-1}{N}\mu^2 \\
&= \frac{N-1}{N}\sigma^2 \qquad\qquad \text{which is different from zero, so the estimator is indeed biased.}
\end{aligned}
\tag{1.63}
$$

Note that the unbiased estimator of the variance is $\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})^2$

Maximum A Posteriori estimate

The MAP estimate chooses the most probable $\boldsymbol{w}$ given the data, so:

Definition 1.2.9: MAP estimate

$$\boldsymbol{w}_{MAP} = \underset{\boldsymbol{w}}{argmax}\ p(\boldsymbol{w}|D) \tag{1.64}$$

With the posterior $p(\boldsymbol{w}|D)$ obtained via Baye's rule. In order to calculate the posterior we are going to assume a distribution over the model parameter, our prior $p(\boldsymbol{w})$.

$$p(\boldsymbol{w}|D) = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)} \tag{1.65}$$

Note that, in general, it is difficult to calculate p(D) and also w can be very high dimensional.

$$\begin{aligned}
\boldsymbol{w}_{MAP} &= \underset{\boldsymbol{w}}{argmax}\ p(\boldsymbol{w}|D) \\
&= \underset{\boldsymbol{w}}{argmax}\ \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{p(D)} \\
&= \underset{\boldsymbol{w}}{argmax}\ p(D|\boldsymbol{w})p(\boldsymbol{w}) \\
&= \underset{\boldsymbol{w}}{argmin} - log(p(D|\boldsymbol{w})) - log(p(\boldsymbol{w}))
\end{aligned}$$

1.3 Lagrange Multipliers

It is common to want to find the maximum / minimum of a function $f(\boldsymbol{x})$ subject to some constrain $\underbrace{g(\boldsymbol{x}) = c}_{level\ set}$. A useful property that we can exploit is that $\bigtriangledown g(\boldsymbol{x})$ is perpendicular to the constraint surface. At constrained maximum (minimum) $\bigtriangledown f(\boldsymbol{x})$ must also be perpendicular to constraint surface. Therefore:

$$\bigtriangledown f(\boldsymbol{x}) + \lambda \bigtriangledown g(\boldsymbol{x}) = 0 \tag{1.66}$$

where λ takes the name of *Lagrangian multiplier.*

It is helpful to introduce the **Lagrangian function**:

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda(g(\boldsymbol{x}) - c) \tag{1.67}$$

The solution to the original problem is given by the stationary points of $L(\boldsymbol{x}, \lambda)$

$$\frac{\partial}{\partial \boldsymbol{x}} L(\boldsymbol{x}, \lambda) = 0 \qquad \frac{\partial}{\partial \lambda} L(\boldsymbol{x}, \lambda) = 0 \tag{1.68}$$

Chapter 2

Regression

2.1 Introductory concepts

The goal of regression is to predict the value of one or more continuous target variables t given the value of a D-dimensional vector $\boldsymbol{x}$ of input variables. The simplest form of linear regression models are linear functions of the input variables. However, we can obtain a much more useful class of functions by taking linear combinations of a fixed set of nonlinear functions of the input vector $\boldsymbol{x}$, known as *basis functions.*

$$y(\boldsymbol{x},\boldsymbol{w}) = w_0 + w_1\phi_1(\boldsymbol{x}) + ... + w_D\phi_D(\boldsymbol{x}) = \sum_{i=0}^{D} w_i\phi_i(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) \qquad \text{with} \qquad \phi_0(\boldsymbol{x}) = 1 \tag{2.1}$$

Note that w_0 is usually called the bias.
More generally, from a probabilistic perspective, we aim to model the predictive distribution $p(t|\boldsymbol{x})$ because this expresses our uncertainty about the value of t for each value of $\boldsymbol{x}$. From this conditional distribution we can make predictions of t, for any new value of $\boldsymbol{x}$, in such a way as to minimize the expected value of a suitably chosen loss function.

The example of polynomial regression considered at the beginning of these notes is a particular example of this model in which there is a single input variable x, and the basis functions take the form of powers of x so that $\phi_j(\boldsymbol{x}) = x^j$. One limitation of polynomial basis functions is that they are global functions of the input variable, so that changes in one region of input space affect all other regions. There are many other possible choices for the basis functions, for example the Gaussian basis function, with $\boldsymbol{x} \in \mathbb{R}^D$:

$$\phi_i(\boldsymbol{x}) = exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right) \tag{2.2}$$

Another possibility is the sigmoidal basis function of the form, with $x \in \mathbb{R}$:

$$\phi_i(x) = \sigma\left(\frac{x-\mu_i}{s_i}\right) \tag{2.3}$$

where $\sigma(a)$ is the logistics sigmoid function defined by

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{2.4}$$

2.1.1 Maximum likelihood and least squares

In the introduction of these notes, we fitted polynomial functions to data sets by minimizing the sum of squares error function. We also showed that this error function could be derived as the maximum likelihood solution under an assumed Gaussian noise model. Let's know go back to it and see this relation in more detail. As before, we assume that the target variable $\boldsymbol{t}$ is given by a deterministic function $y(\boldsymbol{x}, \boldsymbol{w})$ with additive Gaussian noise so that

$$t = y(\boldsymbol{x}, \boldsymbol{w}) + \epsilon \text{ where } \epsilon \propto \mathcal{N}(0, \beta^{-1}) \tag{2.5}$$

In the case of a Gaussian conditional distribution, the conditional mean is

$$\mathbb{E}[\boldsymbol{t}|\boldsymbol{x}] = \int tp(t|x)\ dt = y(\boldsymbol{x}, \boldsymbol{w}) \tag{2.6}$$

Note that the Gaussian noise assumption implies that the conditional distribution of t given x is unimodal, which may be inappropriate for some applications.

Let's consider now a set of inputs $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with corresponding target values $t_1, ..., t_n$. The data distribution (likelihood) is given by:

$$p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\boldsymbol{w}^T\phi(\boldsymbol{x}_n), \beta^{-1}) \tag{2.7}$$

Taking the logarithm of the likelihood function, and making use of the standard form of the Gaussian:

$$\ln p(\boldsymbol{t}|\boldsymbol{w}, \beta) = \sum_{n=1}^{N}\left(\ln\sqrt{\frac{\beta}{2\pi}} + e^{-\frac{\beta}{2}(t_i - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))^2}\right) = \frac{N}{2}\ln\beta - \frac{N}{2}\log 2\pi - \frac{\beta}{2}\sum_{n=1}^{N}(t_i - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))^2 \tag{2.8}$$

Having written down the likelihood function, we can use maximum likelihood to determine $\boldsymbol{w}$ and β. Let's consider first the maximization with respect to $\boldsymbol{w}$.

Note that we can assume to find a local minima (and not a maxima) because the function is **convex**.

The gradient of the log likelihood with respect to $\boldsymbol{w}$ is obtained as follow:

$$\begin{aligned}\frac{\partial}{\partial\boldsymbol{w}}\log p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) &= -\beta\frac{\partial}{\partial\boldsymbol{w}}\frac{1}{2}\sum_{n=1}^{N}(t_i - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))^2 \\ &= -\beta\sum_{n=1}^{N}(t_i - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))\phi(\boldsymbol{x}_n)^T \end{aligned} \tag{2.9}$$

Then we set this to $\mathbf{0}^T$

$$\begin{aligned}
-\beta\sum_{n=1}^{N}(t_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))\phi(\boldsymbol{x}_n)^T &= \mathbf{0}^T \\
\sum_{n=1}^{N}(t_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))\phi(\boldsymbol{x}_n)^T &= \mathbf{0}^T \\
\sum_{n=1}^{N}\boldsymbol{w}^T\phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^T &= \sum_{n=1}^{N}t_n\phi(\boldsymbol{x}_n)^T \text{ let's transpose both sides} \\
\left(\sum_{n=1}^{N}\phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^T\right)\boldsymbol{w} &= \sum_{n=1}^{N}\phi(\boldsymbol{x}_n)t_n \\
\text{We need to introduce the design matrix } \mathbf{\Phi} &= \begin{pmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \dots & \phi_D(\boldsymbol{x}_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \dots & \phi_D(\boldsymbol{x}_1) \end{pmatrix} \\
\text{with the design matrix we obtain: } \quad (\mathbf{\Phi}^T\mathbf{\Phi})\boldsymbol{w} &= \mathbf{\Phi}^T\boldsymbol{t} \\
\text{so our estimator is } \quad \boldsymbol{w}_{ML} &= (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\boldsymbol{t}
\end{aligned}$$

We can also maximize the log likelihood function with respect to the noise precision parameter β, giving:

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}_{ML}^T\phi(\boldsymbol{x}_n)\}^2 \tag{2.10}$$

2.1.2 Regularization

In the regression example at the beginning of these notes we saw that we could increment the hyperparameter D (the degree of the polynomial) at our choice, until we could perfectly interpolate the data, but we also showed that this is not in our best intention. In fact with a suitably flexible model, we can drive the training loss to zero (assuming no label noise), by simply memorizing the correct output for each input. But what we care about is prediction accuracy on new data, which are not part of the training set. A model that perfectly fits the training data, but which is too complex, is said to suffer from **overfitting**. One solution is to reduce the degree of the polynomial. However, a more general solution is to penalize the magnitude of the weights (regression coefficients). We can do this by adding a penalty term to the NLL. Thus we optimize an objective of the form:

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) = \left[\frac{1}{N}\sum_{n=1}^{N} l(\boldsymbol{y}_n, \boldsymbol{\theta}; \boldsymbol{x}_n)\right] + \lambda C(\boldsymbol{\theta}) \tag{2.11}$$

where $\lambda \geq 0$ is the **regularization parameter**, and $C(\boldsymbol{\theta})$ is some form of **complexity penalty**. A common complexity penalty is to use $C(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta})$ where $p(\boldsymbol{\theta})$ is the **prior** for $\boldsymbol{\theta}$.

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) = \frac{1}{N}\sum_{n=1}^{N} l(\boldsymbol{y}_n, \boldsymbol{\theta}; \boldsymbol{x}_n) + \lambda\log(p(\boldsymbol{\theta})) \tag{2.12}$$

Maximizing this is equivalent to maximizing the log posterior, leading us to the **MAP estimation**.

Example: weight decay Before we observed how using polynomial regression with too high of a degree can result in overfitting. A general way to avoid this is to to penalize the magnitude of the weights (regression coefficients). We can do this by using a zero-mean Gaussian prior on the weights, $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \lambda^{-1}\boldsymbol{I})$ The resulting MAP estimate is given by:

$$\begin{aligned}\boldsymbol{w}_{map} &= \underset{\boldsymbol{w}}{argmin}\; NLL(\boldsymbol{w}) + \lambda||\boldsymbol{w}||_2^2 \\ &= \underset{\boldsymbol{w}}{argmin}\; \frac{\beta}{2}\sum_{i=1}^{N}(y(x_i, \boldsymbol{w}) - t_i)^2 + \lambda \boldsymbol{w}^T\boldsymbol{w} \end{aligned} \tag{2.13}$$

The above equation is called l_2 regularization or **weight decay**. The larger the value of λ, the more the parameters are penalized for being “large” (deviating from the zero-mean prior), and thus the less flexible is the model. In the case of linear regression, this kind of penalization scheme (with the l_2 *norm*) is called **ridge regression**. It has the advantage that the error function remains a quadratic function of $\boldsymbol{w}$, and so its exact minimizer can be found in closed form. Specifically, setting the gradient with respect to $\boldsymbol{w}$ to zero, and solving for $\boldsymbol{w}$ we obtain:

$$\boldsymbol{w} = (\lambda\boldsymbol{\mathcal{I}} + \boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} \tag{2.14}$$

Instead of using the euclidean (l_2) norm we can generalize the penalty term:

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\{t_i - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i)\}^2 + \frac{\lambda}{2}\sum_{i=1}^{M-1}|w_i|^q \tag{2.15}$$

If we set $q = 1$ we obtain the lasso regularization, it has the effect of preferring 0-weights and so it induces a sparsification effect.

2.1.3 Stochastic gradient descent

For $N \gg 1$ the solution $\boldsymbol{w}_{ML} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$ is very costly to compute. It needs to process all data $\boldsymbol{X}$ at once and to invert a $M \times M$ matrix $O(M^3)$. The loss is the sum of the error terms for each datapoint:

$$\begin{aligned} E_D(\boldsymbol{w}) &= \sum_{n=1}^{N} E(\boldsymbol{x}_n, \boldsymbol{t}_n, \boldsymbol{w}) \\ E(\boldsymbol{x}_n, \boldsymbol{t}_n, \boldsymbol{w}) &= \frac{1}{2}(\boldsymbol{t}_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))^2 \end{aligned}$$

The simplest approach to using gradient information is to choose a weight update to comprise a small step in the direction of the negative gradient, so that

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} - \eta \bigtriangledown E_D(\boldsymbol{w}^{(i)}) \tag{2.16}$$

where the parameter $\eta > 0$ is known as the *learning rate*. It begins by choosing a random point $\boldsymbol{w}_0$ and after each update, the gradient is re-evaluated for the new weight vector and the process repeated. Note that the error function is defined with respect to a training set, and so each step requires that the entire training set be processed in order to evaluate

$\bigtriangledown E_D$. Techniques that use the whole data set at once are called *batch methods.* At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as gradient descent or steepest descent. In order to find a sufficiently good minimum, it may be necessary to run a gradient-based algorithm multiple times, each time using a different randomly chosen starting point, and comparing the resulting performance on an independent validation set.

There is, however, an on-line version of gradient descent that has proved useful in practice, the idea is to approximate E_D with few datapoints. The gradient $\bigtriangledown_{\boldsymbol{w}} E$ encodes all the directional derivatives via scalar product: $(\bigtriangledown_{\boldsymbol{w}} E)V$. The gradient always points in the direction of steepest ascent and it is always perpendicular to the contours of the function.

The stochastic gradient descent (SGD):

- Initialize $\boldsymbol{w}^{(0)}$, choose learning rate η
- Iterate over data points, and update

$$\begin{aligned}\boldsymbol{w}^{(r+1)} &= \boldsymbol{w}^{(r)} - \eta(\bigtriangledown_{\boldsymbol{w}} E(\boldsymbol{x}_n, \boldsymbol{t}_n, \boldsymbol{w}^{(r)})^T \\ &= \boldsymbol{w}^{(r)} - \eta(\boldsymbol{t}_n - \phi(\boldsymbol{x}_n)^T \boldsymbol{w}^{(r)} \phi(\boldsymbol{x}_n))\end{aligned} \tag{2.17}$$

It is common to pick a small batch of samples and compute the gradient on this batch instead of a single sample. If $E_D(\boldsymbol{w})$ is convex in $\boldsymbol{w}$ and η is small enough the SGD converges. Note that if the learning rate is too small the we have slow convergence, if it is too high we oscillate around local minimum and could have no convergence at all. One way is to use a **learning rate scheduling** with smaller learning rate over time. Gradient descent compared to stochastic GD requires less iterations but each iteration is slow to compute (we are computing the error on the whole dataset).

2.2 Probabilistic linear regression

Consider a dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\} = \{\boldsymbol{x}, \boldsymbol{t}\}$, assume that the data are generated through an underlying model: $t = y(x, \boldsymbol{w}) + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \beta^{-1})$. We define the precision as

$$\beta = \frac{1}{\sigma^2} \tag{2.18}$$

Under these model assumptions we have a target distribution:

$$p(t|, x, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(x, \boldsymbol{w}), \beta^{-1}) \tag{2.19}$$

where $y(x, \boldsymbol{w})$ is the mean function $\mu(x)$.

As we did before, it is convenient to minimize the negative log-likelihood function.

$$\begin{aligned}-\log p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) &= \frac{\beta}{2}\sum_{i=1}^{N}\{y(x_n, \boldsymbol{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \\ &= \log\prod_{i=1}^{N}\mathcal{N}(t_i|y(x_i, \boldsymbol{w}), \beta^-1)\end{aligned}$$

$$\begin{aligned} &= \frac{\beta}{2}\sum_{i=1}^{N}\{y(x_n,\boldsymbol{w})-t_n\}^2 + \frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\beta) \\ &= \frac{1}{2}\sum_{i=1}^{N}\{y(x_n,\boldsymbol{w})-t_n\}^2 \end{aligned} \tag{2.20}$$

We can omit the last two terms since they do not depend on w and they will disappear in the derivative.
Also, we note that scaling the log likelihood by a positive constant coefficient does not alter the location of the maximum with respect to $\mathbf{w}$, and so we can replace the coefficient $\beta/2$ with $1/2$.

We therefore see that minimizing the negative likelihood is equivalent to minimizing the sum of squares error (least sqaures fit). We can also use maximum likelihood to determine the precision parameter β of the Gaussian conditional distribution. Minimizing Equation 2.20 with respect to β gives

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{i=1}^{N}\{y(x_i,\boldsymbol{w}_{ML})-t_i\}^2 \tag{2.21}$$

We can first determine the parameter vector $\boldsymbol{w}_{ML}$ governing the mean and subsequently use this to find the precision β_{ML} as we did for the simple Gaussian.

2.2.1 MAP for the univariate Gaussian distribution

This is our likelihood:

$$p(t|x,\boldsymbol{w},\beta) = \mathcal{N}(t|y(x,\boldsymbol{w}),\beta^{-1}) = \sqrt{\frac{\beta}{2\pi}}\exp\left[-\frac{\beta}{2}(t-y(x,\boldsymbol{w})^2\right] \tag{2.22}$$

Now, we define the prior over $\boldsymbol{w}$ as:

$$p(\boldsymbol{w}|\alpha) = \prod_{i=1}^{N}\mathcal{N}(w_i|0,\alpha^{-1}) = \left(\frac{\alpha}{2\pi}\right)^{\frac{N}{2}}\prod_{I=1}^{N}\exp\left[-\frac{\alpha}{2}w_iw_i\right] = \left(\frac{\alpha}{2\pi}\right)^{\frac{N}{2}}e^{-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}} \tag{2.23}$$

The MAP solution is obtained by solving:

$$\begin{aligned} \boldsymbol{w}_{MAP} &= \underset{w}{argmin}\ \frac{\beta}{2}\sum_{i=1}^{N}\{y(x_i,\boldsymbol{w})-t_i\}^2 - \log p(\boldsymbol{w}|\alpha) \\ &= \underset{w}{argmin}\ \frac{\beta}{2}\sum_{i=1}^{N}\{y(x_i,\boldsymbol{w})-t_i\}^2 + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} \end{aligned} \tag{2.24}$$

So for Baye's theorem:

$$p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{t},\alpha,\beta) \propto p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w},\beta)p(\boldsymbol{w}|\alpha) \tag{2.25}$$

2.3 Bayesian Approach

Although we have included a prior distribution $p(\boldsymbol{w}|\alpha)$, we are so far still making a point estimate of $\boldsymbol{w}$ and so this does not yet amount to a Bayesian treatment. The frequentist approach is to search for one optimal estimate of $\boldsymbol{w}$, while the Bayesian given a prior belief over $\boldsymbol{w}$ gives an entire posterior distribution over $p(\boldsymbol{w}|D)$, this probability reflects the plausibility of different $\boldsymbol{w}$ given our prior knowledge and how likely our data is generate using $\boldsymbol{w}$.

$$p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t}) = \frac{p(\boldsymbol{t}, \boldsymbol{X}|\boldsymbol{w})p(\boldsymbol{w})}{\int p(\boldsymbol{t}, \boldsymbol{X}|\boldsymbol{w}')p(\boldsymbol{w}')\, d\boldsymbol{w}'} \tag{2.26}$$

The denominator $p(\boldsymbol{t}, \boldsymbol{X})$ is called the marginal likelihood (or evidence), since it is computed by marginalizing over (or integrating out) the unknown $\boldsymbol{w}$. This can be interpreted as the average probability of the data, where the average is wrt the prior. Note, however, that $p(\boldsymbol{t}, \boldsymbol{X})$ is a constant, independent of $\boldsymbol{w}$, so we will often ignore it when we just want to infer the relative probabilities of $\boldsymbol{w}$ values.

Definition 2.3.1: Conjugate Prior

We say that a prior $p(\boldsymbol{w}) \in \mathcal{F}$ is a **conjugate prior** for a likelihood function $p(D|\boldsymbol{w})$ if the posterior distribution is in the same parametrized family as the prior, i.e $p(\boldsymbol{w}|D) \in \mathcal{F}$.

Once we have computed the posterior over the parameters, we can compute the **posterior predictive distribution** over outputs given inputs by marginalizing out the unknown parameters. In fact, if we consider both x and $\boldsymbol{w}$ to be random variables. $p(x', \boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t})$ dives the probability of observing a new x' and model parameters $\boldsymbol{w}$ given the already observed data $(\boldsymbol{X}, \boldsymbol{t})$. In the predictive setting we are only interested in x' and thus we compute the marginal $p(x'|\boldsymbol{X}, \boldsymbol{t})$ by integration out $\boldsymbol{w}$. In the curve fitting problem, we are given the training data $\boldsymbol{X}$ and $\boldsymbol{t}$, along with a new test point x', and our goal is to predict the value of t'. We therefore wish to evaluate the predictive distribution $p(t'|x', \boldsymbol{X}, \boldsymbol{t})$. Here we shall assume that the parameters α and β are fixed and known in advance. A Bayesian treatment simply corresponds to consider:

$$p(t'|x', \boldsymbol{X}, \boldsymbol{t}) = \int p(t'|x', \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t})\, d\boldsymbol{w} \tag{2.27}$$

Where $p(t'|x', \boldsymbol{w})$ is given by Equation 2.22 and it is based on our initial assumption over the target distribution (note that we have omitted the dependence on α and β to simplify the notation). While $p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t})$ is the posterior distribution over parameters, and can be found by normalizing the right-hand side of Equation 2.25.

2.3.1 Bayesian Linear Regression

In studying maximum likelihood estimation for linear regression, we observed that model complexity (driven by the number of basis functions) must be carefully balanced with dataset size. Regularization addresses this by introducing a coefficient that helps control complexity, though the selection of basis functions remains crucial. However, relying solely on maximum likelihood leads to overly complex models and overfitting. While hold-out validation can guide complexity, it is inefficient and data-intensive. To overcome these issues, we turn to a Bayesian approach to linear regression, which inherently mitigates overfitting and provides automatic, data-driven methods for selecting model complexity.

Given a dataset of i.i.d samples $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)^T$, and a target vector $\boldsymbol{t} = (t_1, ..., t_n)^T$ we are interested in making prediction of t for new values of x. This requires that we evaluate the *predictive distribution* defined by Equation 2.27 which we expand below:

$$p(t'|\boldsymbol{x}', \boldsymbol{w}, \boldsymbol{X}, \boldsymbol{t}, \alpha, \beta) = \int \underbrace{p(t'|\boldsymbol{x}', \boldsymbol{w}, \beta)}_{likelihood} \underbrace{p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t}, \alpha, \beta)}_{posterior} \, d\boldsymbol{w} = \mathcal{N}(t'|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}'), \beta^{-1}) \quad (2.28)$$

The conditional distribution $p(t'|\boldsymbol{x}', \boldsymbol{w}, \beta)$ of the target variable is given by Equation 2.7 and the posterior weight distribution is obtained through:

$$p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{t}, \alpha, \beta) = \frac{p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \alpha, \beta)p(\boldsymbol{w})}{p(\boldsymbol{t}|\boldsymbol{X}, \alpha, \beta)} = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N, \boldsymbol{S}_N) \quad (2.29)$$

where $\boldsymbol{S}_N = \boldsymbol{S}_0^{-1} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$ and $\boldsymbol{m}_N = \boldsymbol{S}_N(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \beta \boldsymbol{\Phi}^T \boldsymbol{t})$ with $\boldsymbol{m}_0 = 0$ and $\boldsymbol{S}_0 = \alpha^{-1}\boldsymbol{I}$ so $\boldsymbol{S}_N = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$ and $\boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}$. we define now our likelihood and prior as:

$$\begin{aligned} p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \alpha, \beta) &= \prod_{n=1}^{N} \mathcal{N}(t_n|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}) = \mathcal{N}(\boldsymbol{t}| \underbrace{\boldsymbol{\Phi}\boldsymbol{w}}_{\boldsymbol{\mu}}, \underbrace{\beta^{-1}\boldsymbol{I}}_{\boldsymbol{\Sigma}}) \\ p(\boldsymbol{w}) &= \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_0, \boldsymbol{S}_0) \end{aligned} \quad (2.30)$$

If we assume an infinitely broad prior *i.e* $(p(\boldsymbol{w}|\alpha = (\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I})$ *with* $\alpha \to 0)$ this is equivalent to having no restriction on $\boldsymbol{w}$ we can see that our MAP becomes:

$$\lim_{\alpha \to 0} \boldsymbol{m}_N = \lim_{\alpha \to 0} \beta(\alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi} \boldsymbol{t} = \boldsymbol{w}_{ML} \quad (2.31)$$

If instead we assume an infinitely narrow prior $(\alpha \to \infty)$:

$$\begin{aligned} \lim_{\alpha \to \infty} \boldsymbol{m}_N &= \lim_{\alpha \to \infty} \beta(\alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} = \lim_{\alpha \to \infty} \frac{\beta}{\alpha} \boldsymbol{\Phi}^T \boldsymbol{\Phi} = \boldsymbol{0} = \boldsymbol{m}_0 \\ \lim_{\alpha \to \infty} \boldsymbol{S}_N &= \lim_{\alpha \to \infty} \frac{1}{\alpha} \boldsymbol{I} = [\boldsymbol{0}]_{n \times n} \end{aligned} \quad (2.32)$$

The Maximum A Posteriori estimate for $\boldsymbol{w}$ results in:

$$\boldsymbol{w}_{MAP} = \underset{\boldsymbol{w}}{argmax} \; p(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{X}) = \boldsymbol{m}_N \quad (2.33)$$

2.3.2 Sequential Bayesian Learning

To introduce the concept of sequential Bayesian Learning let's introduce a particular form of Gaussian prior, the zero-mean isotropic Gaussian, which is governed by a single precision parameter α so that:

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}) \quad (2.34)$$

In sequential bayesian learning we consider a sequence of data points, spaced in time, so that at each stage we can use the posterior distribution computed on the $i-1$ samples as a prior for the $i-th$ observation.

$$p(\boldsymbol{w}|D_N) = \frac{p(\boldsymbol{x}_N|\boldsymbol{w})p(\boldsymbol{w}|D_{N-1})}{p(\boldsymbol{x}_N)} \tag{2.35}$$

Let's see and example for $N = 2$:

$$\begin{aligned}
p(\boldsymbol{w}|\boldsymbol{x}_2, \boldsymbol{x}_1) &= \frac{p(\boldsymbol{x}_2, \boldsymbol{x}_1|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{x}_2, \boldsymbol{x}_1)} \\
&= \frac{p(\boldsymbol{x}_2|\boldsymbol{x}_1, \boldsymbol{w})p(\boldsymbol{x}_1|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{x}_2)p(\boldsymbol{x}_1)} \qquad \text{since they are i.i.d} \\
&= \frac{p(\boldsymbol{x}_2|\boldsymbol{w})p(\boldsymbol{x}_1|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{x}_2)p(\boldsymbol{x}_1)} \\
&\text{now I can write } \frac{p(\boldsymbol{x}_1|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{x}_1)} \text{ as the posterior } p(\boldsymbol{w}|\boldsymbol{x}_1) \\
&= \frac{p(\boldsymbol{x}_2|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{x}_1)}{p(\boldsymbol{x}_2)}
\end{aligned} \tag{2.36}$$

As shown in Figure 2.1 at each step the previous posterior distribution forms the new prior. The first row of this figure corresponds to the situation before any data points are observed and shows a plot of the prior distribution in $\boldsymbol{w}$ space together with six samples of the function $y(\boldsymbol{x}, \boldsymbol{w})$ in which the values of $\boldsymbol{w}$ are drawn from the prior. In the second row, we see the situation after observing a single data point. The location (x, t) of the data point is shown by a blue circle in the right column. In the left column there is a plot of the **likelihood function** $p(t|x, w)$ for this data point as a function of w. Note that the likelihood function provides a soft constraint that the line must pass close to the data point, where close is determined by the noise precision β. Then we multiply this likelihood function by the **prior from the top row**, and normalize, we obtain the **posterior distribution** shown in the middle plot on the second row. Samples of the regression function $y(x, w)$ obtained by drawing samples of w from this posterior distribution are shown in the right-hand plot. Note that these sample lines all pass close to the data point. The third row of this figure shows the effect of observing a second data point, again shown by a blue circle in the plot in the right-hand column. The corresponding likelihood function for this second data point alone is shown in the left plot. When we multiply this likelihood function by the posterior distribution from the second row, we obtain the posterior distribution shown in the middle plot of the third row. Note that this is exactly the same posterior distribution as would be obtained by combining the original prior with the likelihood function for the two data points. This posterior has now been influenced by two data points, and because two points are sufficient to define a line this already gives a relatively compact posterior distribution. Samples from this posterior distribution give rise to the functions shown in red in the third column, and we see that these functions pass close to both of the data points. The fourth row shows the effect of observing a total of 20 data points. The left-hand plot shows the likelihood function for the 20 th data point alone, and the middle plot shows the resulting posterior distribution that has now absorbed information from all 20 observations. Note how the posterior is much sharper than in the third row. In the limit of an infinite

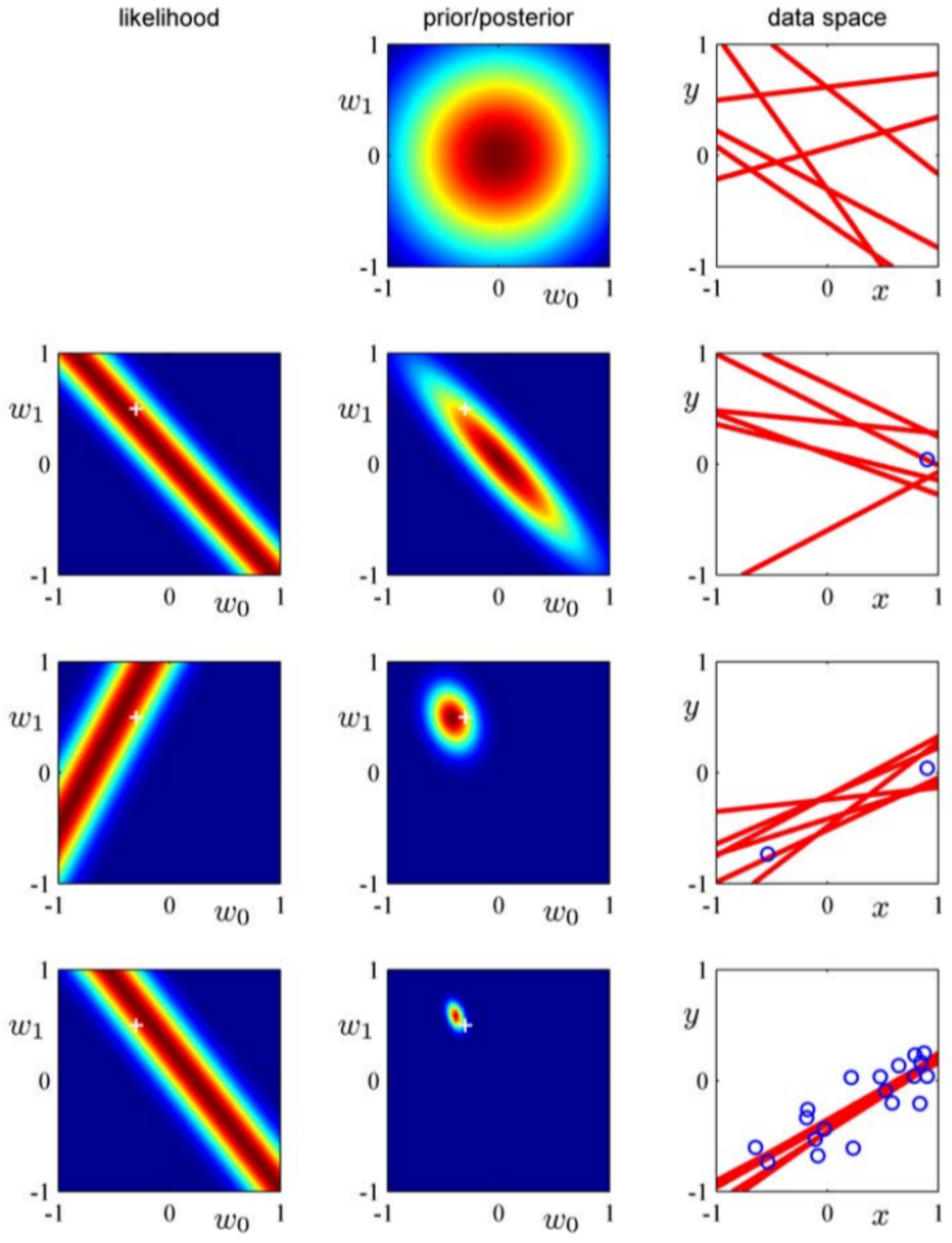


Figure 2.1: Example of sequential bayesian learning.

number of data points, the posterior distribution would become a delta function centered on the true parameter values, shown by the white cross.

An interesting fact is that after infinite amount of data ($N \to \infty$) ML = MAP = Bayesian since in the first equality prior plays no role and in the latter the posterior is sharp:

- $\lim_{N\to\infty}[\boldsymbol{\Phi}^T\boldsymbol{\Phi}]_{ij} = \lim_{N\to\infty}\sum_{n=1}^{N}\boldsymbol{\phi}_i(\boldsymbol{x}_n)\boldsymbol{\phi}_j(x_n) \propto N$
- $\lim_{N\to\infty}, \boldsymbol{S}_N = [\mathbf{0}]_{n\times n}$
- $\lim_{N\to\infty}\boldsymbol{m}_N = \lim_{N\to\infty}\beta(\alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}\boldsymbol{t}$

From the Bayesian predictive distribution Equation 2.28 we obtain that:

$$\begin{aligned} p(t'|\boldsymbol{x}',\boldsymbol{w},\boldsymbol{X},\boldsymbol{t},\alpha,\beta) &= \int \mathcal{N}(t'|\boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{w},\beta^{-1})\mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_N,\boldsymbol{S}_N)\, d\boldsymbol{w} \\ &= \mathcal{N}(t'|\boldsymbol{x}',\boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{m}_N,\sigma_N^2(\boldsymbol{x}')) \qquad (2.37) \end{aligned}$$

$$\boldsymbol{m}_N = \beta\boldsymbol{S}_N\boldsymbol{\Phi}^T\boldsymbol{t} \qquad \sigma_N^2(\boldsymbol{x}') = \frac{1}{\beta} + \boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}') \qquad \boldsymbol{S}_N^{-1} = \alpha\boldsymbol{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi} \qquad (2.38)$$

where $\boldsymbol{m}_N$ is the MAP solution.

Let's see now how this can be formulated using Kernels. Given the predictive distribution Equation 2.37, the predictive mean is:

$$y(\boldsymbol{x}',\boldsymbol{m}_N) = \boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{m}_N == \sum_{n=1}^{N} k(\boldsymbol{x}',\boldsymbol{x}_n)t_n \qquad (2.39)$$

A weighted sum of training samples, no parameters! The function k is known as *smoother matrix* or *equivalent kernel*. Using as kernel $k(\boldsymbol{x}',\boldsymbol{x}) = \beta\boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_n)$ we can give a proof for the previous equivalence:

$$\begin{aligned} \sum_{n=1}^{N} k(\boldsymbol{x}',\boldsymbol{x}_n)t_n &= \sum_{n=1}^{N}\beta\boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_n)t_n = \beta\boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{S}_N\sum_{n=1}^{N}\boldsymbol{\Phi}_{n,:}^T t_n \\ &= \beta\boldsymbol{\phi}(\boldsymbol{x}')\boldsymbol{S}_N\boldsymbol{\Phi}^T\boldsymbol{t} = \boldsymbol{\phi}(\boldsymbol{x}')^T\boldsymbol{m}_N = y(\boldsymbol{x}',\boldsymbol{m}_N) \end{aligned}$$

Regression functions, such as this, which make predictions by taking linear combinations of the training set target values are known as *linear smoothers*. The formulation of linear regression in terms of a kernel function suggests an alternative approach to regression as follows. Instead of introducing a set of basis functions, which implicitly determines an equivalent kernel, we can instead define a localized kernel directly and use this to make predictions for new input vectors x, given the observed training set. This leads to a practical framework for regression (and classification) called Gaussian processes.
Using the gaussian basis function the training points $\boldsymbol{x}_n$ close to $\boldsymbol{x}'$ contribute more to the model predictions! In fact the covariance between predictions is $cov[t_1,t_2|\boldsymbol{x}_1,\boldsymbol{x}_2] = \boldsymbol{\phi}(\boldsymbol{x}_1)^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_2))$ let's derive it:

$$\begin{aligned} cov[t_1,t_2|\boldsymbol{x}_1,\boldsymbol{x}_2] &= cov_{\boldsymbol{w}}[y(\boldsymbol{x}_1,\boldsymbol{w}),y(\boldsymbol{x}_2,\boldsymbol{w})] = cov_{\boldsymbol{w}}[\boldsymbol{\phi}(\boldsymbol{x}_1)^T\boldsymbol{w},\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_2)] \\ &= \mathbb{E}_{\boldsymbol{w}}[\boldsymbol{\phi}(\boldsymbol{x}_1)^T\boldsymbol{w},\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_2)] - \mathbb{E}_{\boldsymbol{w}}[\boldsymbol{\phi}(x_1)^T\boldsymbol{w}]\,\mathbb{E}_{\boldsymbol{w}}[\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_2)] \\ &= \boldsymbol{\phi}(\boldsymbol{x}_1)^T(\mathbb{E}_{\boldsymbol{w}}[\boldsymbol{w}\boldsymbol{w}^T] - \mathbb{E}[\boldsymbol{w}]\,\mathbb{E}[\boldsymbol{w}]^T)\boldsymbol{\phi}(\boldsymbol{x}_2) = \boldsymbol{\phi}(\boldsymbol{x}_1)^T cov[\boldsymbol{w},\boldsymbol{w}]\boldsymbol{\phi}(\boldsymbol{x}_2) \\ &= \boldsymbol{\phi}(\boldsymbol{x}_1)^T\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_2) \qquad (2.40) \end{aligned}$$

Chapter 3

Model Selection

The questions that we want to answer in a supervised learning settings are: How can we reliably estimate the model performance properly for unknown data? How can we choose the optimal hyperparameters?

One common method to select the best set of hyperparameters is through model selection. Starting from a dataset $D = \{(\boldsymbol{x}_1, t_1), ..., (\boldsymbol{x}_N, t_N)\}$ we divide it into 3 subsets: the **Training**, the **Validation** and the **Test**, following an approximate 80/10/10% partition (sometimes is useful to have the test set a bit bigger than this). The Training and Validation set allows us to find the optimal $\boldsymbol{w}^*$. We do that by minimizing the error $(E(y(\boldsymbol{x}, \boldsymbol{w}), t)$ for every $(\boldsymbol{x}, t) \in D_{train}$ and comparing different $\boldsymbol{w}$-s over the validation set. This last evaluation is truly important as it allow us to *estimate* the generalization error $E(y(\boldsymbol{x}_{val}, \boldsymbol{w}^*), t_{val})$ over every $(\boldsymbol{x}_{val}, t_{val}) \in D_{val}$ for different set of hyperparameters $\boldsymbol{w}$.
In the end, we calculate our true generalization error $E(y(\boldsymbol{x}_{test}, \boldsymbol{w}^*), t_{test})$ over the Test set $(\boldsymbol{x}_{test}, t_{test}) \in D_{test}$.
Note that the test set should **NEVER** be used for model selection.

What happens if we have only a small amount of data available?
To strengthen our generalization previsions and in general the model generalization capabilities, we use a technique called K-fold Cross-Validation. It consists of splitting the data $D = \{(\boldsymbol{x}_1, t_1), ..., (\boldsymbol{x}_N, t_N)\}$ into $\boldsymbol{K}$ folds, train the $i-th$ model on $\boldsymbol{K}-1$ folds and evaluate its performances on the left-out fold. The same model is trained (and evaluated) over all the different fold combinations and its validation error is calculated as the average over the left-out folds.

$$CV(\hat{y}) = \frac{1}{N} \sum_{i=1}^{N} E(\hat{y}_{\hat{\boldsymbol{w}}}^{-k(i)}(\boldsymbol{x}_i), t) \tag{3.1}$$

Where $\hat{y}$ is the current model we are evaluating (corresponding to a specific hyperparameter configuration and trained over the K-1 folds), $k(i)$ is a map from $\{1, ..., N\} \to \{1, ..., K\}$ that tells us whether the $i-th$ sample is in the validation set or not. The best hyperparameter configuration (model) is obtained by computing:

$$\boldsymbol{w}^* = \underset{\hat{\boldsymbol{w}}}{argmax}\ CV(\hat{y}_{\hat{\boldsymbol{w}}}) \tag{3.2}$$

After having selected the best hyperparameter configuration $\boldsymbol{w}^*$ we evaluate the model over the hold-out test set. Here we could run into the same problem as before, what happens if we have only few sample to evaluate the model on?

We could use a Nested Cross-Validation, in order to evaluate $\boldsymbol{K}$ different models over $\boldsymbol{K}$ hold-out sets, where each corresponding training set was in turn divided by a $\boldsymbol{H}$-fold Cross-Validation.

Chapter 4

Decision Theory

4.1 Regression

Consider a dataset of observations $D = \{(\boldsymbol{x}_1, t_1), ..., (\boldsymbol{x}_N, t_N)\}$ our goal is to predict t' given a new value for $\boldsymbol{x}'$. The joint probability distribution $p(\boldsymbol{x}, t)$ provides a complete summary of the uncertainty associated with these variables. The determination of $p(\boldsymbol{x}, t)$ from a set of training data is an example of inference and is typically a very difficult problem. In a practical application, we must often make a specific prediction for the value of t, or more generally take a specific action based on our understanding of the values t is likely to take, and this aspect is the subject of decision theory.

Every time we make an observation of a random variables $(\boldsymbol{x}, t)$ we make a different error. Let's now consider the expected loss:

$$\mathbb{E}_{(\boldsymbol{x},t)\sim p(\boldsymbol{x},t)}\lfloor L(t, y(\boldsymbol{x}))\rfloor = \int\int (t - y(\boldsymbol{x}))^2 p(\boldsymbol{x}, t) \;\; d\boldsymbol{x}dt \tag{4.1}$$

Where L is the loss function we're currently using $L = \{y(\boldsymbol{x}) - t\}^2$.

The best model y we can possibly have is $y(\boldsymbol{x}) = \mathbb{E}[t|\boldsymbol{x}]$

The Figure 4.1 shows how the function $y(\boldsymbol{x})$ which minimizes the expected squared loss, is given by the mean of the conditional distribution $p(t|\boldsymbol{x})$. We now analyze the expected

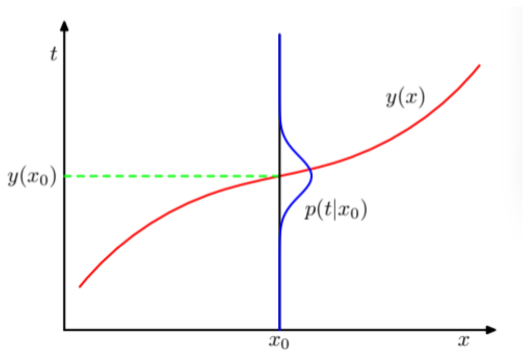


Figure 4.1: Example of regression function

loss relatively to the regression function $\mathbb{E}[t|\boldsymbol{x}] \coloneqq \mathbb{E}_{t\sim p(t|\boldsymbol{x})}[t|\boldsymbol{x}]$

$$\mathbb{E}[L] = \int\int (y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}] + \mathbb{E}[t|\boldsymbol{x}] - t)^2 \; p(\boldsymbol{x}, t) \; dt d\boldsymbol{x}$$

I write out the square taking as terms $a = (y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])$ *and* $b = (\mathbb{E}[t|\boldsymbol{x}] - t)$

$$= \int\int (y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2 p(\boldsymbol{x}, t) \; dt d\boldsymbol{x}$$
$$+ \int\int 2(y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])(\mathbb{E}[t|\boldsymbol{x}] - t) p(\boldsymbol{x}, t) \; dt d\boldsymbol{x}$$
$$+ \int\int (\mathbb{E}[t|\boldsymbol{x}] - t)^2 p(\boldsymbol{x}, t) \; dt d\boldsymbol{x}$$

I note that the middle term is 0, now I use the product rule $p(\boldsymbol{x}, t) = p(t|\boldsymbol{x})p(\boldsymbol{x})$

$$= \int\int (y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2 p(t|\boldsymbol{x})p(\boldsymbol{x}) \; dt d\boldsymbol{x} + \int\int (\mathbb{E}[t|\boldsymbol{x}] - t)^2 p(t|\boldsymbol{x})p(\boldsymbol{x}) \; dt d\boldsymbol{x}$$

recognize that $(\mathbb{E}[t|\boldsymbol{x}] - t)^2$ is the variance $var[t|\boldsymbol{x}]$

and that $\int p(t|\boldsymbol{x}) \; dt = 1$ so we obtain

$$= \underbrace{\int (y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2 p(\boldsymbol{x}) \; d\boldsymbol{x}}_{\text{sub-optimal y}} + \underbrace{\int var[t|\boldsymbol{x}] p(\boldsymbol{x}) \; d\boldsymbol{x}}_{\text{inherent noise}} \tag{4.2}$$

Note that the term relative to the inherent noice cannot be reduced. The best possible model is $y(\boldsymbol{x}) = \mathbb{E}[t|\boldsymbol{x}]$, in practice we approximate it with fits of $y_D = \underset{y}{argmin} \sum_{(\boldsymbol{x},t)\in D} L(t, y(\boldsymbol{x}))$.

Indeed, we can identify three distinct approaches to solving regression problems given, in order of decreasing complexity, by:

- First solve the inference problem of determining the joint density $p(\boldsymbol{x}, t)$. Then normalize to find the conditional density $p(t|\boldsymbol{x})$ and finally marginalize to find the conditional mean given by:

$$\frac{\delta \, \mathbb{E}[L]}{\delta y(\boldsymbol{x})} = 2 \int \{y(\boldsymbol{x}) - t\} p(\boldsymbol{x}, t) \, dt = 0 \tag{4.3}$$

  Solving for $y(\boldsymbol{x})$, and using the sum and product rules of probability, we obtain

$$y(\boldsymbol{x}) = \frac{\int t \, p(\boldsymbol{x}, t) \, dt}{p(\boldsymbol{x})} = \int t \, p(t|\boldsymbol{x}) \, dt = \mathbb{E}_t[t|\boldsymbol{x}] \tag{4.4}$$

  Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as *generative models*, because by sampling from them it is possible to generate synthetic data points in the input space.

- Another way is to solve the inference problem of determining the conditional density $p(t|\boldsymbol{x})$, and then subsequently marginalize to find the conditional mean given by Equation 4.4. Approaches that model the posterior probabilities directly are called *discriminative models.*

- Find a regression function $y(\boldsymbol{x})$, called a discriminant function, directly from the training data which maps each input $\boldsymbol{x}$ directly onto a target t. In this case, probabilities play no role.

Let us consider the relative merits of these three alternatives. The first approach is the most demanding because it involves finding the joint distribution over both $\boldsymbol{x}$ and $\boldsymbol{t}$. For many applications, $\boldsymbol{x}$ will have high dimensionality, and consequently we may need a large training set in order to be able to determine the class-conditional densities to reasonable accuracy. Note that the target priors $p(\boldsymbol{t})$ can often be estimated simply from the fractions of the training set data points in each of the classes. One advantage of the first approach, however, is that it also allows determine the marginal density of data $p(\boldsymbol{x})$. This can be useful for detecting new data points that have low probability under the model and for which the predictions may be of low accuracy, which is known as *outlier detection* or novelty detection. However, if we only wish to make classification decisions, then it can be wasteful of computational resources, and excessively demanding of data, to find the joint distribution $p(\boldsymbol{x}, \boldsymbol{t})$ when in fact we only really need the posterior probabilities $p(\boldsymbol{t}|\boldsymbol{x})$, which can be obtained directly through the second approach. An even simpler approach is the latter, in which we use the training data to find a discriminant function $y(\boldsymbol{x})$ that maps each $\boldsymbol{x}$ directly onto a class label, thereby combining the inference and decision stages into a single learning problem. With this option, however, we no longer have access to the posterior probabilities $p(\boldsymbol{t}|\boldsymbol{x})$. There are many powerful reasons for wanting to compute the posterior probabilities, even if we subsequently use them to make decisions. These include: minimizing risk, reject option and combining models.

Average Expected Loss

Let's now analyze the performances of a learning algorithm by averaging the expected loss over learned y_D for different datasets D.

$$\begin{gathered}
\mathbb{E}_D[\mathbb{E}[L]] = \int \mathbb{E}_D[(y_D(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2]p(\boldsymbol{x})dx + \int var[t|\boldsymbol{x}]p(\boldsymbol{x})d\boldsymbol{x} \\
\text{we're going to analyze it relatively to the average model } \mathbb{E}_D[y_D(\boldsymbol{x})] \\
\mathbb{E}_D[(y_D(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2] = \mathbb{E}_D[(y_D(\boldsymbol{x}) - \mathbb{E}_D[y_D(\boldsymbol{x})] + \mathbb{E}_D[y_D(\boldsymbol{x})] - \mathbb{E}[t|\boldsymbol{x}])^2] \\
\text{as before I'm going to expand the squares} \\
= \mathbb{E}_D[(y_D(\boldsymbol{x}) - \mathbb{E}_D[y_D(\boldsymbol{x})])^2] + \mathbb{E}_D[(\mathbb{E}_D[y_D(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2] \\
+2\,\mathbb{E}_D[(y_D(\boldsymbol{x}) - \mathbb{E}_D[y_D(\boldsymbol{x})])(\mathbb{E}_D[y_D(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])] \\
\text{this last term cancels out, so we end up with} \\
= \mathbb{E}_D[(y_D(\boldsymbol{x}) - \mathbb{E}_D[y_D(\boldsymbol{x})])^2] + (\mathbb{E}_D[y_D(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2 \qquad (4.5)
\end{gathered}$$

On average (over datasets D) our model y_D will make three types of errors:

$$\begin{aligned}
\mathbb{E}_D[\mathbb{E}[L]] &= \int \mathbb{E}_D[(y_D(\boldsymbol{x}) - \mathbb{E}_D[y_D(\boldsymbol{x})])^2]d\boldsymbol{x} && \text{Variance} \\
&= + \int (\mathbb{E}_D[y_D(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}])^2 p(\boldsymbol{x})d\boldsymbol{x} && \text{Bias}^2 \\
&= + \int var[t|\boldsymbol{x}]p(\boldsymbol{x})d\boldsymbol{x} && \text{Noise} \qquad (4.6)
\end{aligned}$$

Our goal is to minimize the expected loss, which we have decomposed into the sum of a (squared) bias, a variance, and a constant noise term. As we shall see, there is a trade-off

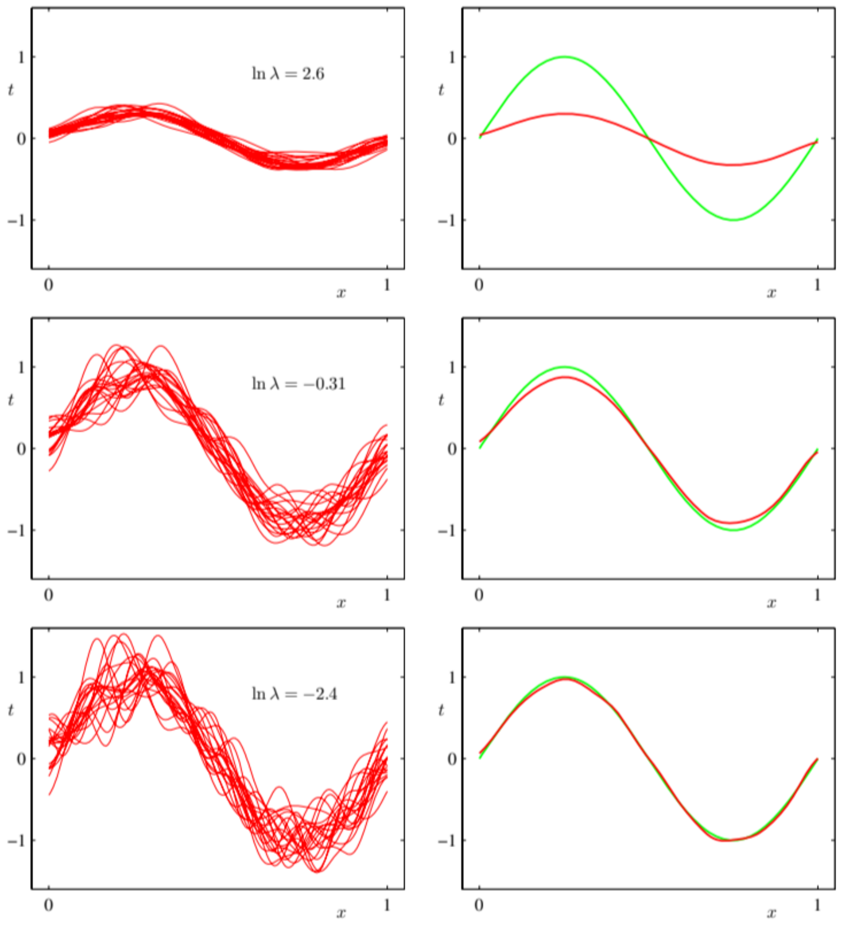


Figure 4.2: Examples of models with different variance/bias balancing.

between bias and variance, with very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance. The model with the optimal predictive capability is the one that leads to the best balance between bias and variance.

Example

Imagine to generate 100 dataset, each containing N=50 data points, each sampled from $\mathbb{E}[t|x] = \sin(2\pi x) + e_i \quad where \quad e \sim \mathcal{N}(0, \alpha^{-1})$. Now we fit a model with 24 Gaussian basis functions by minimizing the regularized error function $E_D = \frac{1}{2}\sum_{i=1}^{N}\{t_i - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})\}^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$ to give a prediction function $y^{(l)}(x)$, one for each dataset. As we can see from Figure 4.2 in the first row, when the variance is low (the red curves overlap) the bias is high (we're fitting a sinusoid that is not exactly $\mathbb{E}[t|x]$). Conversely on the last row, if we sent λ very small, our model is not well regularized and so we obtain a large variance (each one of the l models tries to fit all the training points) but low bias. In practice, we see that small values of λ allow the model to become finely tuned to the noise on each individual data set leading to large variance. On the contrary, larger values of λ pulls the weight parameters towards zero leading to large bias.

Although the bias-variance decomposition may provide some interesting insights into

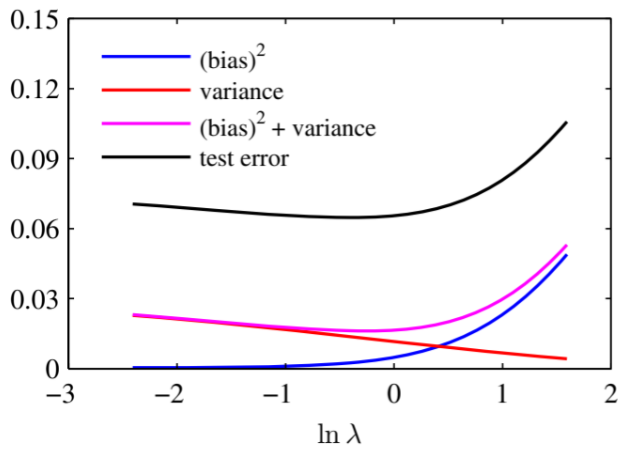


Figure 4.3: Bias-Variance tradeoff.

the model complexity issue from a frequentist perspective, it is of limited practical value, because the bias-variance decomposition is based on averages with respect to ensembles of data sets, whereas in practice we have only the single observed data set.

4.2 Classification

Let's now analyze the problem of classification through decision regions. The goal in classification is to take an input vector $\boldsymbol{x}$ and to assign it to one of K discrete classes C_k where $K = 1, ..., K$. In the most common scenario, the classes are taken to be disjoint, so that each input is assigned to one and only one class. The input space is thus divided into *decision regions* whose boundaries are called *decision boundaries* or *decision surfaces*. The decision surfaces are **linear functions** of the input vector $\boldsymbol{x}$ and hence are defined by $(D-1)$-dimensional hyperplanes within the D-dimensional input space (linear classification consider only decision linear boundaries). Data sets whose classes can be separated exactly by linear decision surfaces are said to be *linearly separable*.

For probabilistic models, in the case of a two-class problem, the most convenient representation is the binary one. In which there is a single target variable $t \in \{0, 1\}$ such that $t = 1$ represents class C_1 and $t = 0$ represents class C_2. We can interpret the value of t as the probability that the class is C_1 with the values of probability taking only the extreme values of 0 and 1. For $K > 2$ classes, it is convenient to use a type of coding scheme called **one-hot encoding** in which $\boldsymbol{t}$ is a vector of length K such that if the class is C_j, then all elements t_k of $\boldsymbol{t}$ are zero except element t_j , which takes the value 1. For instance, if we have $K = 5$ classes, then a pattern from class 2 would be given the target vector: $\boldsymbol{t} = (0, 1, 0, 0, 0)^T$. Our goal is to divide the input space $\mathbb{R}^D$ into K decision regions R_k representing each a class C_k as show in Figure 4.4. The boundaries of decision regions are called *decision boundaries/surfaces*.

What happens if we have $K > 2$ classes? We might be tempted be to build a K-class discriminant by combining a number of two-class discriminant functions. However, this leads to some serious difficulties. Consider the use of $K-1$ classifiers each of which solves a two-class problem of separating points in a particular class C_k from points not in that class.

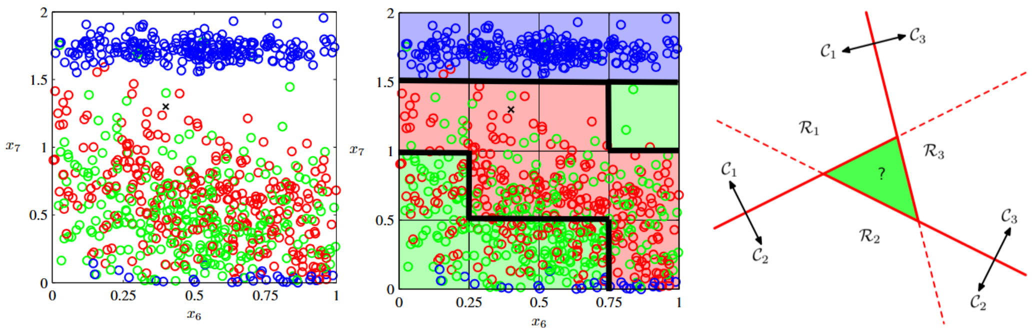


Figure 4.4: Examples of decision regions and boundaries.

This is known as a **one-versus-all classifier**. An alternative is to introduce $K(K-1)/2$ binary discriminant functions, one for every possible pair of classes. This is known as a **one-versus-one** classifier. Each point is then classified according to a **majority vote** amongst the discriminant functions. However, this too runs into the problem of ambiguous regions, as illustrated in the Figure 4.5.
A useful tool for classification are confusion matrices, they compare the ground truth classes vs the predicted one. On the diagonal we count the elements correctly classified, while off the diagonal the misclassified ones.

$$\begin{array}{c} \\ C_1 \\ C_2 \\ \vdots \\ C_K \end{array}\begin{array}{c} \begin{array}{cccc} R_1 & R_2 & \dots & R_K \end{array} \\ \begin{bmatrix} 6 & 1 & \dots & 0 \\ 5 & 3 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 2 & 0 & \dots & 8 \end{bmatrix} \end{array}$$

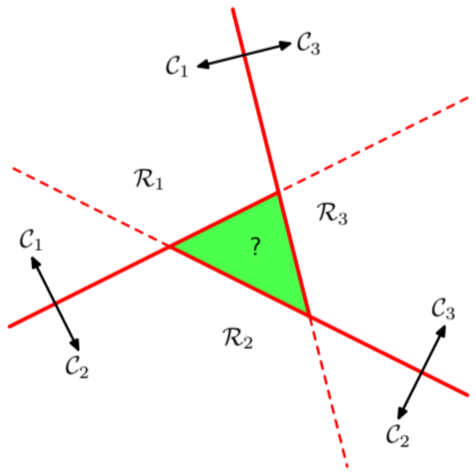


Figure 4.5: Example of multiple decision boundaries.

Chapter 5

Linear Models for Classification

Previously in these notes, we identified three distinct approaches to the classification problem. The simplest involves constructing a discriminant function that directly assigns each vector $\boldsymbol{x}$ to a specific class. A more powerful approach, however, models the conditional probability distribution $p(C_k|\boldsymbol{x})$ in an inference stage, and then subsequently use this distribution to make optimal decisions. By separating inference and decision, we gain numerous benefits, as previously discussed. There are actually two different approaches to determining the conditional probabilities $p(C_k|\boldsymbol{x})$. One technique is to model them directly, for example by representing them as parametric models and then optimizing the parameters using a training set. Alternatively, we can adopt a **generative approach** in which we model the class-conditional densities given by $p(\boldsymbol{x}|C_k)$, together with the prior probabilities $p(C_k)$ for the classes, and then we compute the required posterior probabilities using Bayes' theorem.

Let's consider a classification model of the following form:

$$p(C_k|\boldsymbol{x},\boldsymbol{\theta}) = \frac{p(\boldsymbol{x}|C_k,\boldsymbol{\theta})p(C_k|\boldsymbol{\theta})}{p(\boldsymbol{x},C_k|\boldsymbol{\theta})} \tag{5.1}$$

The term $p(C_k|\boldsymbol{\theta})$ is the prior over class labels, and the term $p(\boldsymbol{x}|C_k,\boldsymbol{\theta})$ is the class conditional density for class k. The overall model is called a **generative classifier**, since it specifies a way to generate the features $\boldsymbol{x}$ for each class k, by sampling from $p(\boldsymbol{x}|C_k,\boldsymbol{\theta})$. By contrast, a **discriminative classifier** directly models the class posterior $p(C_k|\boldsymbol{x},\boldsymbol{\theta})$. If we choose the class conditional densities in a special way, we will see that the resulting posterior over classes is a linear function of $\boldsymbol{x}$ i.e $\log p(y=c|\boldsymbol{x},\boldsymbol{\theta}) = \boldsymbol{w}^T\boldsymbol{x} + const$, where $\boldsymbol{w}$ is derived from $\boldsymbol{\theta}$. Thus the overall method is classed **linear discriminant analysis** or **LDA**.

Note that C_k can be seen also as $\boldsymbol{t} = c$ where c is a specific class.

Why are we interested in joint or posterior?
Decision theory tells us that the best prediction for input $\boldsymbol{x}$ is to choose the class with highest joint $p(\boldsymbol{x},C_k) = p(C_k|x)p(x)$, or equivalently to choose the class with the highest posterior $p(C_k|\boldsymbol{x})$. The decision boundaries between classes C_k and C_j are at $p(C_k|\boldsymbol{x}) = p(C_j|\boldsymbol{x})$

5.1 Probabilistic generative models

5.1.1 Linear Discriminant Analysis

Here we shall adopt a **generative approach** in which we model the class-conditional densities $p(\boldsymbol{x}|C_k)$, as well as the class priors $p(C_k)$, and then use these to compute posterior probabilities $p(C_k|\boldsymbol{x})$ through Bayes' theorem. Assuming to have only 2 classes ($K = 2$) the posterior distribution Equation 5.1 for the class C_1 can be written as:

$$p(C_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_1)p(C_1) + p(\boldsymbol{x}|C_2)p(C_2)} = \frac{1}{1 + \frac{p(\boldsymbol{x}|C_2)p(C_2)}{p(\boldsymbol{x}|C_1)p(C_1)}}$$

now I can define

$$a = \log\left(\frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_2)p(C_2)}\right) = \log\left(\left(\frac{p(\boldsymbol{x}|C_2)p(C_2)}{p(\boldsymbol{x}|C_1)p(C_1)}\right)^{-1}\right) = -\log\left(\frac{p(\boldsymbol{x}|C_2)p(C_2)}{p(\boldsymbol{x}|C_1)p(C_1)}\right)$$

$$\text{so } -a = \log\left(\frac{p(\boldsymbol{x}|C_2)p(C_2)}{p(\boldsymbol{x}|C_1)p(C_1)}\right) \text{ from which}$$

$$\frac{1}{1 + \frac{p(\boldsymbol{x}|C_2)p(C_2)}{p(\boldsymbol{x}|C_1)p(C_1)}} = \frac{1}{1 + e^{-a}} = \sigma(a) \tag{5.2}$$

The term 'sigmoid' means S-shaped. This type of function is sometimes also called a 'squashing function' because it maps the whole real axis into a finite interval. It satisfies the following properties:

Property 5.1.1: Sigmoid's properties

- $\sigma(-a) = 1 - \sigma(a)$
- $a = \ln\left(\frac{\sigma(a)}{1-\sigma(a)}\right)$
- $\sigma'(a) = \sigma(a)(1 - \sigma(a))$

Note that the inverse of the sigmoid is knwon as the logit function and it represents the log of the ratio of probabilities for the two classes, also known as the *log odds*.

$$\textit{log odds} = \ln\left(\frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_2)p(C_2)}\right) \tag{5.3}$$

For multiple classes (general K):

$$p(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{\sum_{j=1}^{K} p(\boldsymbol{x}|C_j)p(C_j)} = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}} \tag{5.4}$$

which is known as the normalized exponential and can be regarded as a multiclass generalization of the logistic sigmoid. Here the quantities a_k are defined by

$$a_k = \ln(p(\boldsymbol{x}|C_k)p(C_k)) \tag{5.5}$$

The normalized exponential is also known as the softmax function, as it represents a smoothed version of the 'max' function because, if $a_k \gg a_j$ for all $j \neq k$, then $p(C_k|\boldsymbol{x}) \simeq 1$

and $p(C_j|\boldsymbol{x}) \simeq 0$.

Definition 5.1.2: Softmax function

$$\sigma(\boldsymbol{z})_i = \frac{exp(z_i)}{\sum_{j=1}^{K} exp(z_j)} \tag{5.6}$$

Let's assume that the class conditional densities are multivariate Gaussians:

$$p(\boldsymbol{x}|C_k, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{(Det(\boldsymbol{\Sigma}_k))^{1/2}} exp(\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k)) \tag{5.7}$$

To simplify things we consider the covariance matrices shared across classes $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_k$. For 2 classes the corresponding class posterior are given as:

$$p(C_1|\boldsymbol{x}) = \frac{1}{1 + exp(-a)} = \sigma(a) \text{ with}$$

$$a = \ln \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_2)p(C_2)} = \ln \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) - \ln(\boldsymbol{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}) + \ln \frac{p(C_1)}{p(C_2)}$$

$$= \cancel{-\frac{1}{2}\ln|\boldsymbol{\Sigma}|} - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_1) + \cancel{\frac{1}{2}\ln|\boldsymbol{\Sigma}|} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_2) + \ln \frac{p(C_1)}{p(C_2)}$$

$$= \underbrace{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)T\boldsymbol{\Sigma}^{-1}\boldsymbol{x}}_{\boldsymbol{w}^T\boldsymbol{x}} \underbrace{- \frac{1}{2}\boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln \frac{p(C_1)}{p(C_2)}}_{-w_0} \tag{5.8}$$

We see that the quadratic terms in $\boldsymbol{x}$ from the exponents of the Gaussian densities have cancelled (due to the assumption of common covariance matrices) leading to a linear function of x in the argument of the logistic sigmoid, $a(x)$ is a linear model, so the the **generalized linear model** is $p(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0)$. Note that σ it's not linear but we use it only to discretize the outputs of the model.

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$
$$w_0 = \frac{1}{2}\boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln \frac{p(C_1)}{p(C_2)}$$

The resulting decision boundaries correspond to surfaces along which the posterior probabilities $p(C_k|\boldsymbol{x})$ are constant and so will be given by linear functions of $\boldsymbol{x}$, and therefore the decision boundaries are linear in input space. The prior probabilities $p(C_k)$ enter only through the bias parameter w_0 so that changes in the priors have the effect of making parallel shifts of the decision boundary and more generally of the parallel contours of constant posterior probability.

For the general case of K classes we have one linear model per class defined by:

$$a_k(\boldsymbol{x}) = \boldsymbol{w}_k^T \boldsymbol{w} + w_{k0}$$

$$\text{where} \quad \boldsymbol{w}_k = \Sigma^{-1}\boldsymbol{\mu}_k \quad \text{and} \quad w_{k0} = -\frac{1}{2}\boldsymbol{\mu}_k^T \Sigma^{-1}\boldsymbol{\mu}_k + \ln p(C_k) \tag{5.9}$$

The resulting decision boundaries, corresponding to the minimum misclassification rate are $p(C_k|\boldsymbol{x}) = p(C_j|\boldsymbol{x}) \Rightarrow a_k(\boldsymbol{x}) = a_j(\boldsymbol{x})$.

Note that if all the covariance matrices were different $\boldsymbol{\Sigma}_k \neq \boldsymbol{\Sigma}_j$ then $a_k(\boldsymbol{x})$ would also have contained a quadratic term in $\boldsymbol{x}$.

Model fitting

Once we have specified a parametric functional form for the class-conditional densities $p(\boldsymbol{x}|C_k)$, we can then determine the values of the parameters $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, together with the prior class probabilities $p(C_k)$, using maximum likelihood. Let q parametrize the prior probabilities with $p(C_1) = q$ and $p(C_2) = 1 - q$. So for $\boldsymbol{x}_n$ with $t_n = 1$ we have $p(\boldsymbol{x}_n, C_1) = p(\boldsymbol{x}_n|C_1)p(C_1) = q\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ while for $t_n = 0$ the gaussian is multiplied by $1 - q$.

The likelihood function is defined as:

$$\begin{aligned} p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) &= \prod_{n=1}^{N} p(\boldsymbol{x}_n, t_n) = \prod_{n=1}^{N} p(\boldsymbol{x}_n|t_n)p(t_n) \\ &= \prod_{n=1}^{N} \left[p(\boldsymbol{x}_n|C_1)p(C_1)\right]^{t_n} \left[p(\boldsymbol{x}_n|C_2)p(C_2)\right]^{1-t_n} \\ &= \prod_{n=1}^{N} \left[q\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right]^{t_n} \left[(1-q)\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\right]^{1-t_n} \end{aligned} \tag{5.10}$$

Note that in case of a multiclass classification problem with the target vector $\boldsymbol{t}_n$ one-hot encoded, the likelihood function is defined as follows:

$$p(D|\boldsymbol{\theta}) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(t_{nk} = 1|\boldsymbol{\theta}) p(\boldsymbol{x}_n|\boldsymbol{\theta}_k)^{t_{nk}} \tag{5.11}$$

The log-likelihood for the two-class problem is:

$$\ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} t_n \ln q + t_n\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) + (1-t_n)\ln(1-q) + (1-t_n)\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$$

we cancel the terms that does not depend on q thus obtaining

$$= \sum_{n=1}^{N} t_n \ln q + (1-t_n)\ln(1-q) \tag{5.12}$$

We now estimate q solving the MLE.

$$\frac{\partial}{\partial q} \ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = 0$$

$$\frac{\partial}{\partial q} \ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \frac{t_n}{q} - \frac{1-t_n}{1-q} = \sum_{n=1}^{N} \frac{t_n(1-q) - (1-t_n)q}{q(1-q)} = \sum_{n=1}^{N} \frac{t_n - q}{q(1-q)}$$

solving for q

$$\sum_{n=1}^{N} \frac{t_n - q}{q(1-q)} = 0 \Rightarrow (q \neq 0, 1) \sum_{n=1}^{N} q = \sum_{n=1}^{N} t_n \iff q = \frac{1}{N}\sum_{n=1}^{N} t_n \tag{5.13}$$

The MLE solution to $p(C_1) = q$ is the fraction of points with label $t_n = 1$ as expected.

Let's now consider the maximization with respect to $\boldsymbol{\mu}_1$, this time we pick out of the log-likelihood the terms that depend on $\boldsymbol{\mu}_1$:

$$\ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} t_n \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) \tag{5.14}$$

We now estimate $\boldsymbol{\mu}_1$ solving the MLE:

$$\frac{\partial}{\partial \boldsymbol{\mu}_1} \sum_{n=1}^{N} t_n \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) = \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\mu}_1} \sum_{n=1}^{N} t_n (\boldsymbol{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_1)$$

$$\text{since } \boldsymbol{\Sigma} \text{ is symmetric so the derivative is } = \sum_{n=1}^{N} t_n (\boldsymbol{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}$$

$$\text{solving for } \boldsymbol{\mu}_1 \quad \sum_{n=1}^{N} t_n (\boldsymbol{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} = \boldsymbol{0}^T$$

$$\text{since } \boldsymbol{\Sigma}^{-1} \text{ is positive definite the inverse always exists}$$

$$\sum_{n=1}^{N} t_n \boldsymbol{\mu}_1 = \sum_{n=1}^{N} t_n \boldsymbol{x}_n \iff N_1 \boldsymbol{\mu}_1 = \sum_{n=1}^{N} t_n \boldsymbol{x}_n$$

$$\text{so } \boldsymbol{\mu}_{1\,ML} = \frac{1}{N_1} \sum_{n=1}^{N} t_n \boldsymbol{x}_n \quad \text{which is the average for } \boldsymbol{x}_n \text{ for the case } t_n = 1 \tag{5.15}$$

The MLE for $\boldsymbol{\mu}_2$ is specular.

Finally, consider the maximum likelihood solution for the shared covariance matrix $\boldsymbol{\Sigma}$. The log-likelihood is:

$$\ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} t_n \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) + (1 - t_n) \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}) \tag{5.16}$$

We now estimate $\boldsymbol{\mu}_1$ solving the MLE:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \ln p(\boldsymbol{t}, \boldsymbol{X}|q, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) =$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \left[-\frac{1}{2} \sum_{n=1}^{N} t_n \ln |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{n=1}^{N} t_n (\boldsymbol{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_1) \right.$$

$$\left. - \sum_{n=1}^{N} (1 - t_n) \ln |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{n=1}^{N} (1 - t_n) (\boldsymbol{x}_n - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_2) \right]$$

$$\text{in the end we obtain}$$

$$\boldsymbol{\Sigma}_{ML} = \frac{N_1}{N} \left[\underbrace{\frac{1}{N_1} \sum_{n=1}^{N} t_n (\boldsymbol{x}_n - \boldsymbol{\mu}_{1\,ML})^T (\boldsymbol{x}_n - \boldsymbol{\mu}_{1\,ML})}_{\text{sample covariance of class 1}} \right] +$$

$$\frac{N_2}{N}\left[\underbrace{\frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)(\boldsymbol{x}_n-\boldsymbol{\mu}_{2\ ML})^T(\boldsymbol{x}_n-\boldsymbol{\mu}_{2\ ML})}_{\text{sample covariance of class 2}}\right] \tag{5.17}$$

This result is easily extended to the K class problem to obtain the corresponding maximum likelihood solutions for the parameters in which each class-conditional density is Gaussian with a shared covariance matrix.
Note that the approach of fitting Gaussian distributions to the classes is not robust to outliers, because the maximum likelihood estimation of a Gaussian is not robust.

For the joint probabilities:

$$p(\boldsymbol{x}, C_1) = p(\boldsymbol{x}|C_1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{1\ ML}, \boldsymbol{\Sigma})q_{ML}$$
$$p(\boldsymbol{x}, C_2) = p(\boldsymbol{x}|C_2) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{2\ ML}, \boldsymbol{\Sigma})(1-q_{ML})$$

For each new datapoint $\boldsymbol{x}'$ we evaluate $p(C_1|\boldsymbol{x}') = \sigma(\boldsymbol{w}_{ML}^T\boldsymbol{x}' + W_{0\ ML})$ using the linear model given by maximum-likelihood. We assign $\boldsymbol{x}'$ to C_1 if $p(C_1|\boldsymbol{x}') > \frac{1}{2}$. The disadvantages of LDA are that Gaussian distribution is sensitive to outliers, linearity/handcrafted features restrict its application and most important Maximum likelihood is prone to overfitting (we are not doing any regularization here).

5.1.2 Naive Bayes

Let us now consider the case of discrete feature values $\boldsymbol{x}_i$. For simplicity, we will consider binary feature values $\boldsymbol{x}_i \in \{0, 1\}$ If there are D inputs, then a general distribution would correspond to a table of 2^D numbers for each class, containing $2^D - 1$ independent variables (due to the summation constraint). Because this grows exponentially with the number of features, we might seek a more restricted representation. Here we will make the naive Bayes assumption in which the feature values are treated as independent, conditioned on the class C_k . Thus we have class-conditional distributions of the form:

$$p(\boldsymbol{x}|C_k) = p(x_1, x_2, ..., x_D|C_k) = p(x_1|D)\cdot\ldots\cdot(x_D|C_k) = \prod_{i=1}^{D} p(x_i|C_k) = \prod_{i=1}^{D}\pi_{ki}^{x_i}(1-\pi_{ki})^{(1-x_i)}$$

which contain D class conditional parameters π_{ki}.

The class posterior is:

$$p(C_k|\boldsymbol{x}) = \frac{exp(a_k(\boldsymbol{x}))}{\sum_{j=1}^{K} exp(a_j(\boldsymbol{x}))} \tag{5.18}$$

And the prior $p(C_k) = q_k$ where

$$\begin{aligned} a_k &= \ln p(\boldsymbol{x}|C_k)p(C_k) = \ln p(\boldsymbol{x}|C_k) + \ln p(C_k) \\ &= \sum_{i=1}^{D} x_i \ln \pi_{ki} + (1-x_i)\ln(1-\pi_{ki}) + \ln p(C_k) \end{aligned} \tag{5.19}$$

which again are linear functions of the input values x_i . For the case of $K = 2$ classes, we can alternatively consider the logistic sigmoid formulation. Analogous results are obtained for discrete variables each of which can take $M > 2$.

5.2 Discriminant functions

The discriminant function performs a direct mapping of input to target, they are functions $f : \mathbb{R}^D \to C_k$.

We consider now the **linear** model:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

The negative of the bias is sometimes called a threshold. In the case of K=2 an input vector $\boldsymbol{x}$ is assigned to class C_1 if $y(x) \geq 0$ and to class C_2 otherwise. The decision boundary takes the form $y(\boldsymbol{w}, \boldsymbol{x}) = 0$, which corresponds to a (D-1)-dimensional hyperplane within the D-dimensional input space, if we consider any 2 datapoints $(\boldsymbol{x}_a, \boldsymbol{x}_b)$ on the decision boundary we have:

$$\boldsymbol{w}^T \boldsymbol{x}_a + w_0 = \boldsymbol{w}^T \boldsymbol{x}_b + w_0$$
$$y(\boldsymbol{x}_a) = y(\boldsymbol{x}_b) = 0$$
$$\boldsymbol{w}^T (\boldsymbol{x}_b - \boldsymbol{x}_a) = 0 \text{ so } (\boldsymbol{x}_b - \boldsymbol{x}_a) \perp \boldsymbol{w}$$

Note that $\boldsymbol{w}$ determines the orientation of the decision boundary, the distance between the decision boundary and the origin is computed as:

$$d = \frac{-w_0}{||\boldsymbol{w}||}$$

Intuitively, w_0 shifts the decision boundary away from the origin.

Being r the distance of a general point $\boldsymbol{x}'$ (not on the decision boundary) we can define $\boldsymbol{x}'$ as

$$\boldsymbol{x}' = \boldsymbol{x}'_\perp + r \frac{\boldsymbol{w}}{||\boldsymbol{w}||}$$

Plugging $\boldsymbol{x}'$ in the model function $y(\boldsymbol{x})$ we obtain:

Note that

$\boldsymbol{w}^T \boldsymbol{x}_\perp = -\boldsymbol{w}_0$

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}' + w_0 = \cancel{\boldsymbol{w}^T \boldsymbol{x}'_\perp} + r \frac{\overbrace{\boldsymbol{w}^T \boldsymbol{w}}^{||\boldsymbol{w}||^2}}{||\boldsymbol{w}||} + \cancel{w_0} = r||\boldsymbol{w}|| \tag{5.20}$$

The distance r of a point to the decision boundary is $r = \frac{y(\boldsymbol{x}')}{||\boldsymbol{w}||}$.

For multiple classes we assign $\boldsymbol{x}$ to C_k if $\forall_{j \neq k} : y_k(\boldsymbol{x}) > y_j(\boldsymbol{x})$, where the decision boundaries between R_k and R_j are defined as $y_k(\boldsymbol{x}) = y_j(\boldsymbol{x})$ and hence corresponds to a (D - 1)-dimensional hyperplane defined by

$$(\boldsymbol{w}_k - \boldsymbol{w}_j)^T \boldsymbol{x} + (w_{k0} - w_{j0}) = 0 \tag{5.21}$$

The decision regions of such a discriminant are always singly connected and convex.

To see this, consider two points $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ both of which lie inside decision region R_k. Any point $\hat{\boldsymbol{x}}$ that lies on the line connecting $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ can be expressed in the form

$$\hat{\boldsymbol{x}} = \lambda \boldsymbol{x}_a + (1 - \lambda)\boldsymbol{x}_b \tag{5.22}$$

where $0 \leq \lambda \leq 1$.

5.2.1 Least squares for classification

Before in these notes we considered models that were linear functions of the parameters, and we saw that the minimization of a sum-of-squares error function led to a simple closed-form solution for the parameter values. It is therefore tempting to see if we can apply the same formalism to classification problems. Consider a general classification problem with K classes, with a 1 hot encoded binary coding scheme for the target vector $\boldsymbol{t}$. Here, each class C_k is described by its own linear model:

$$y_k(\boldsymbol{x}) = \boldsymbol{w}_k^T \boldsymbol{x} + w_{k0} \tag{5.23}$$

We can conveniently group these together using vector notation so that

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{W}^T \boldsymbol{x} \tag{5.24}$$

where $\boldsymbol{W}$ is a matrix whose k^{th} column comprises the $D+1$-dimensional vector $\boldsymbol{w}_k$. We determine this parameter matrix $\boldsymbol{W}$ by minimizing a sum of squares error function.

$$E_D(\boldsymbol{W}) = \sum_{n=1}^{N} \sum_{k=1}^{K} (\boldsymbol{w}_k^T \boldsymbol{x}_n - t_{nk})^2 = \frac{1}{2} \boldsymbol{Tr} \left\{ (\boldsymbol{X}\boldsymbol{W} - \boldsymbol{T})^T (\boldsymbol{X}\boldsymbol{W} - \boldsymbol{T}) \right\} \tag{5.25}$$

Setting the derivative with respect to $\boldsymbol{W}$ to zero, we obtain

$$\boldsymbol{W}_{LS} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{T} = \boldsymbol{X}^\dagger \boldsymbol{T} \tag{5.26}$$

We obtain the discriminant function in the form:

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{W}^T \boldsymbol{x} = \boldsymbol{T}^T (\boldsymbol{X}^\dagger)^T \boldsymbol{x} \tag{5.27}$$

The decision boundaries are very sensitive to outliers, moreover the components of $\boldsymbol{y}_{LS}(\boldsymbol{x})$ are not probabilities, even though they sum to 1. he sum-of-squares error function penalizes predictions that are 'too correct' in that they lie a long way on the correct side of the decision boundary. **This happens because the output of logistic regression is a real value that can be arbitrarly large**

5.2.2 Perceptron

Another example of a linear discriminant model is the perceptron, it corresponds to a two-class model in which the input vector $\boldsymbol{x}$ is first transformed using a fixed nonlinear transformation to give a feature vector $\boldsymbol{\phi}(\boldsymbol{x})$, and this is then used to construct a generalized linear model of the form

$$y(\boldsymbol{x}) = f(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})) \tag{5.28}$$

where the nonlinear activation function $f(\cdot)$ is given by this step function:

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases} \tag{5.29}$$

For the perceptron, instead of the one hot encoding scheme it is more convenient to use target values $+1$ for class C_1 and -1 for class C_2, which matches the choice of activation function. The algorithm used to determine the parameters w of the perceptron can most easily be motivated by error function minimization. A natural choice of error function would be the total number of misclassified patterns. However, this does not lead to a simple learning algorithm because the error is a piecewise constant function of w, with discontinuities wherever a change in w causes the decision boundary to move across one of the data points. We therefore consider an alternative error function known as **the perceptron criterion**. To derive this, we note that we are seeking a weight vector $\boldsymbol{w}$ such that patterns $\boldsymbol{x}_n$ in class C_1 will have $\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) > 0$, whereas patterns in class C_2 will have $\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) < 0$, using the previously described target coding scheme we have that all correct classified patterns must satisfy $\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{t}_n > 0$. The perceptron criterion is given by:

With number of error as error function it becomes a piecewise function which is non differentiable, so it has problems when combined with gradient based optimization

$$E_P(\boldsymbol{w}) = -\sum_{n\in M} \boldsymbol{w}^T\boldsymbol{\phi}_n t_n \tag{5.30}$$

where M denotes the set of all misclassified patterns. We now apply the stochastic gradient descent algorithm to this error function in order to find the best $\boldsymbol{w}$. At each iteration the contribution to the error from a misclassified pattern will be reduced but the change in the weight vector may cause some previously correctly classified patterns to become misclassified. the perceptron learning rule is not guaranteed to reduce the total error function at each stage. However, the perceptron convergence theorem states that if there exists an exact solution (in other words, if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps. However, the number of steps required to achieve convergence could still be substantial, and in practice, until convergence is achieved, we will not be able to distinguish between a non-separable problem and one that is simply slow to converge. Even when the data set is linearly separable, there may be many solutions, and which one is found will depend on the initialization of the parameters $\boldsymbol{w}^{(0)}$ and on the order of presentation of the data points.

5.3 Probabilistic discriminative models

For the multiclass classification problem, we have seen that the posterior probability of class C_K is given by a softmax transformation of a linear function of $\boldsymbol{x}$. For specific choices of the class-conditional densities $p(\boldsymbol{x}|C_k)$, we have used maximum likelihood to determine the parameters of the densities as well as the class priors $p(C_k)$ and then used Bayes' theorem to find the posterior class probabilities. The indirect approach to finding the parameters of a generalized linear model, by fitting class-conditional densities and class priors separately and then applying Bayes' theorem, represents an example of **generative modeling**, because we could take such a model and generate synthetic data by drawing values of $\boldsymbol{x}$ from the marginal distribution $p(\boldsymbol{x})$. In the direct approach, we are maximizing a likelihood function defined through the conditional distribution $p(C_k|\boldsymbol{x})$, which represents a form of discriminative training. One advantage of the discriminative approach is that there will typically be fewer adaptive parameters to be determined. Sometimes in these chapter we have considered classification models that work directly with the original input vector $\boldsymbol{w}$. However, all of the algorithms are equally applicable if we first make a fixed nonlinear transformation of the inputs using a vector of basis functions $\boldsymbol{\phi}(\boldsymbol{x})$. The resulting decision boundaries will be linear in the feature space $\boldsymbol{\phi}$, and these correspond to nonlinear decision boundaries in the original $\boldsymbol{x}$ space. Some limitations of Fixed basis functions are that they are fixed and not learned by the model (as will happen with Neural Network), and that to cover growing dimensions D of input vectors, the number of basis functions needs to grow rapidly /exponentially.

5.3.1 Logistic Regression

Let's begin our treatment of generalized linear models by considering the problem of two class classification. As a discriminative linear model our aim is to model the posterior probabilities $p(C_k|\boldsymbol{\phi})$ as non-linear function of some **linear input**. One example that we have already seen is the logistic sigmoid:

$$p(C_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}) \tag{5.31}$$

In the terminology of statistics, this model is known as logistic regression, although it should be emphasized that this is a model for classification rather than regression. For an M-dimensional feature space, this model has M adjustable parameters. By contrast, if we had fitted Gaussian class conditional densities using maximum likelihood, we would have used 2M parameters for the means and $M(M+1)/2$ parameters for the (shared) covariance matrix. For a dataset $\{\boldsymbol{\phi}_n, t_n\}$, where $t_n \in \{0,1\}$ with $n = 1,...,N$ the likelihood function can be written as

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}\{1-y_n\}^{1-t_n} \tag{5.32}$$

As usual, we can define an error function by taking the negative logarithm of the likelihood, which gives the **cross-entropy** error function in the form

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{t}|\boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \tag{5.33}$$

where $y_n = \sigma(a_n)$ and $a_n = \boldsymbol{w}^T\boldsymbol{\phi}_n$.

Taking the gradient of the error function w.r.t $\boldsymbol{w}$, we obtain

$$\bigtriangledown E(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi_n \tag{5.34}$$

We see that the factor involving the derivative of the logistic sigmoid has cancelled, leading to a simplified form for the gradient of the log likelihood. In particular, the contribution to the gradient from data point n is given by the error $y_n - t_n$ between the target value and the prediction of the model, times the basis function vector $\boldsymbol{\phi}_n$. It is worth noting that maximum likelihood can exhibit severe over-fitting for data sets that are linearly separable. This arises because the maximum likelihood solution occurs when the hyperplane corresponding to $\sigma = 0.5$, equivalent to $\boldsymbol{w}^T\boldsymbol{\phi} = 0$, separates the two classes and the magnitude of w goes to infinity. In this case, the logistic sigmoid function becomes infinitely steep in feature space, corresponding to a Heaviside step function, so that every training point from each class k is assigned a posterior probability of 1. Maximum likelihood provides no way to favor one such solution over another, and which solution is found in practice will depend on the choice of optimization algorithm and on the parameter initialization. Note that the problem will arise even if the number of data points is large compared with the number of parameters in the model, so long as the training data set is linearly separable. The singularity can be avoided by inclusion of a prior and finding a MAP solution for $\boldsymbol{w}$, or equivalently by adding a regularization term to the error function.

5.3.2 Iterative reweighted least squares

The cross-entropy error function is convex but there is no closed-form solution, this is due to the nonlinearity of the logistic sigmoid function. However, thanks to its convexity the error function can be minimized by an efficient iterative techniques based on the *Newton-Raphson* iterative optimization scheme, which uses a **local** quadratic approximation of the error function thanks to a second order approximation. Algorithm:

- Initial guess $\boldsymbol{w}^{(0)}$
- For $\tau = 1, ...$
  - Approximate $E(\boldsymbol{w})$ with a quadratic function $\tilde{E}(\boldsymbol{w})$ around $\boldsymbol{w}^{(\tau-1)}$
  - Construct $\boldsymbol{w}^(\tau)$ such that it minimize $\tilde{E}(\boldsymbol{w})$
  - Stop when $\|\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^{(\tau)}\| = 0$
- You have found $\boldsymbol{w}^*$ such that $\frac{\partial}{\partial \boldsymbol{w}} E(\boldsymbol{w}^*) = 0$

Given our old estimate $\boldsymbol{w}^{(\tau-1)}$ we approximate $E(\boldsymbol{w})$ with a **second order Taylor expansion** around $\boldsymbol{w}^{(\tau-1)}$

$$E(\boldsymbol{w}) \approx \tilde{E}(\boldsymbol{w}^{(n-1)} + \triangle\boldsymbol{w}) = E(\boldsymbol{w}^{(n-1)}) + (\triangle\boldsymbol{w})^T \bigtriangledown^T E(\boldsymbol{w}^{(n-1)}) + \frac{1}{2}(\triangle\boldsymbol{w})^T \boldsymbol{H} \triangle\boldsymbol{w} \quad (5.35)$$

where $\boldsymbol{H}$ is the Hessian matrix which is symmetric. Now, we choose $\triangle\boldsymbol{w}$ such that the next extimate

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} + \triangle\boldsymbol{w} \quad (5.36)$$

minimize $\tilde{E}(\boldsymbol{w})$

$$\frac{\partial}{\partial \triangle\boldsymbol{w}} \tilde{E}(\boldsymbol{w}^{(\tau-1)} + \triangle\boldsymbol{w}) = \bigtriangledown E + (\triangle\boldsymbol{w})^T \boldsymbol{H} \quad (5.37)$$

setting it to 0

$$\begin{aligned} &\bigtriangledown E + (\triangle\boldsymbol{w})^T \boldsymbol{H} = 0 \\ &\boldsymbol{H}\triangle\boldsymbol{w} = -\bigtriangledown^T E \\ &\triangle\boldsymbol{w} = -\boldsymbol{H}^{-1} \bigtriangledown^T E \end{aligned} \quad (5.38)$$

the update rule is

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \boldsymbol{H}^{-1} \bigtriangledown^{T} E(\boldsymbol{w}^{(\tau-1)}) \tag{5.39}$$

the hessian is computed as

$$\boldsymbol{H} = \bigtriangledown \bigtriangledown E(\boldsymbol{w}) = \sum_{n=1}^{N} y_n(1-y_n)\phi_n\phi_n^T = \boldsymbol{\Phi}^T R \boldsymbol{\Phi}$$

$$where \;\; R_{nn} = y_n(1-y_n) \;\; and \;\; R_{nm} = 0 \;\; if \;\; n \neq m$$

The error function $E(\boldsymbol{w})$ is convex when its Hessian is positive definite, meaning

$$\forall_{\boldsymbol{w}\neq 0 \in \mathbb{R}^M} : \boldsymbol{w}^T \boldsymbol{H} \boldsymbol{w} > 0 \tag{5.40}$$

The Hessian of $E(\boldsymbol{w})$ is given by $\boldsymbol{H} = \boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi}$ with $\boldsymbol{R} = diag_{n\times n}\{y_n(1-y_n)\}$

$$\begin{aligned} \boldsymbol{w}^T \boldsymbol{H} \boldsymbol{w} &= \boldsymbol{w}^T \boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi} \boldsymbol{w} \\ &= \boldsymbol{w}^T \boldsymbol{\Phi}^T \boldsymbol{R}^{1/2} \boldsymbol{R}^{1/2} \boldsymbol{\Phi} \boldsymbol{w} \\ &= (\boldsymbol{R}^{1/2} \boldsymbol{\Phi} \boldsymbol{w})^T (\boldsymbol{r}^{1/2} \boldsymbol{\Phi} \boldsymbol{w}) \\ &= \|\boldsymbol{R}^{1/2} \boldsymbol{\Phi} \boldsymbol{w}\|^2 > 0 \end{aligned} \tag{5.41}$$

so indeed my hessian is positive definite, and so my error function convex.

Now we have everything in order to actually compute a step of the update rule 5.39 with gradient $\bigtriangledown E(\boldsymbol{w} = \boldsymbol{\Phi}^T(\boldsymbol{y}-\boldsymbol{t})$ and Hessian $\boldsymbol{H} = \boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi}$

$$\begin{aligned} \boldsymbol{w}^{(\tau)} &= \boldsymbol{w}^{(\tau-1)} - (\boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T (\boldsymbol{y}-\boldsymbol{t}) \\ &= (\boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi})^{-1} \{\boldsymbol{\Phi} \boldsymbol{R} \boldsymbol{\Phi} \boldsymbol{w}^{(\tau-1)} - \boldsymbol{\Phi}^T(\boldsymbol{y}-\boldsymbol{t})\} \\ &= (\boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi})^{-1} \{\boldsymbol{\Phi}^T \boldsymbol{R}(\boldsymbol{z} + \boldsymbol{R}^{-1}(\boldsymbol{y}-\boldsymbol{t})) - \boldsymbol{\Phi}^T(\boldsymbol{y}-\boldsymbol{t})\} \\ &\text{let's call} \quad \boldsymbol{\Phi} \boldsymbol{w}^{(\tau-1)} - \boldsymbol{R}^{-1}(\boldsymbol{y}-\boldsymbol{t}) = \boldsymbol{z} \\ &= (\boldsymbol{\Phi} \boldsymbol{R} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{z} \end{aligned} \tag{5.42}$$

observe that this is very similar to the ML solution obtained for linear regression $((\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t})$. Compared to SGD, Newton-Raphson does not require any step-size, converge faster but we need to compute the Hessian and its inverse.

Chapter 6

Neural Networks

In the previous chapters we considered models for regression and classification that comprised linear combinations of **fixed** basis functions. We saw that such models have useful analytical and computational properties but that their practical applicability was limited by the curse of dimensionality. In order to apply such models to large scale problems, it is necessary to adapt the basis functions to the data. A possible approach is to fix the number of basis functions in advance but allow them to be adaptive, in other words to use parametric forms for the basis functions in which the parameter values are adapted during training. The most successful model of this type in the context of pattern recognition is the feed-forward neural network, also known as the *multilayer perceptron*. Actually, 'multilayer perceptron' is really a misnomer, because the model comprises multiple layers of logistic regression models (with continuous nonlinearities) rather than multiple perceptrons (with discontinuous nonlinearities). For many applications, the resulting model can be significantly more compact, and hence faster to evaluate, than a support vector machine having the same generalization performance. The price to be paid for this compactness, as with the relevance vector machine, is that the likelihood function, which forms the basis for network training, is no longer a convex function of the model parameters.

the sigmoid is continuous while the perceptron uses as activation a step function

A basic Neural Network model with 2 layers can be described as a series of functional transformations, in particular we construct M linear combination of the input variables $x_1, ..., x_D$ in the form:

$$\phi_m(\boldsymbol{x}, \boldsymbol{w}_m^{(1)}) = h(\boldsymbol{w}_m^{(1)T}\boldsymbol{x}) = \underbrace{h(\underbrace{\sum_{d=0}^{D} w_{md}^{(1)} x_d}_{a_m\ activation})}_{\phi_m(\boldsymbol{x}, \boldsymbol{w}_1^{(1)})} \tag{6.1}$$

where $m = 1, ..., M$ and the superscript (1) indicates that the corresponding parameters are in the first "layer" of the network. The quantities inside h are known as **activations**, where each of the is then transformed using a differentiable, **nonlinear** *activation function* $h(\cdot)$ to give:

$$z_m = h(a_m) \tag{6.2}$$

These quantities correspond to the outputs of the basis functions that, in the context of neural networks, are called hidden units. These values are then again linearly combined to give **output unit activation**:

$$a_k = \sum_{m=0}^{M} w_{km}^{(2)} z_m \tag{6.3}$$

This transformation corresponds to the second layer of the network. Finally, the output unit activations are transformed using an appropriate **activation function** to give a set of network outputs y_k. The choice of activation function is determined by the nature of the data and the assumed distribution of target variables. Thus for standard regression problems, the activation function is the identity so that $\boldsymbol{y_k = a_k}$. Similarly, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function $\boldsymbol{\sigma(a_k)}$. For regression our function looks like the following:

$$y(\boldsymbol{x}, \boldsymbol{W}^1, \boldsymbol{w}^{(2)}) = \underbrace{\sum_{m=1}^{M} w_m^{(2)} \underbrace{h(\underbrace{\sum_{d=0}^{D} w_{md}^{(1)} x_d}_{a_m})}_{z_m}}_{a_k} = \boldsymbol{w}^{(2)T} \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{w}_1^{(1)}) \tag{6.4}$$

For classification:

$$y(\boldsymbol{x}, \boldsymbol{W}^1, \boldsymbol{w}^{(2)}) = f(\boldsymbol{w}^{(2)T} \boldsymbol{h}(\boldsymbol{W}^{(1)T} \boldsymbol{x})) \tag{6.5}$$

Where f is the logistics sigmoid / softmax.

Multilayer perceptron

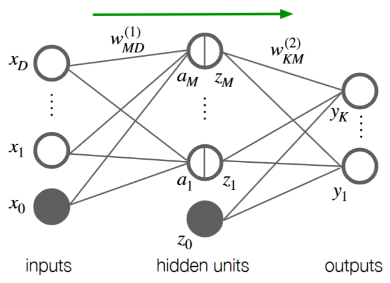


Figure 6.1: Network diagram for the two layer neural network

We can combine these various stages to give the overall network function:

$$y_k(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}) = h^{(2)} \left(\sum_{m=0}^{M} w_{km}^{(2)} h^{(1)} \left(\sum_{d=0}^{D} w_{md}^{(1)} x_d \right) \right) \tag{6.6}$$

Thus the neural network model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector $\boldsymbol{w}$ of adjustable parameters. The process of evaluating this function can then be interpreted as a forward propagation of information through the network. As can be seen from Figure 6.1, the neural network model comprises two stages of processing, each of which resembles the perceptron model, and for this reason the neural network is also known as the multilayer perceptron, or MLP. A key difference compared to the perceptron, however, is that the neural network uses continuous sigmoidal nonlinearities in the hidden units, whereas the perceptron uses step-function nonlinearities. This means that the neural network function is differentiable with respect to the network parameters, and this property will play a central role in network training.

Why activation functions are needed? Without activation function to induce non linearity then for any such network we can always find an equivalent network without hidden units. This follows from the fact that the composition of successive linear transformations is itself a linear transformation. Some popular activation functions are:

- Logistic sigmoid:
$$h(a) = \sigma(a) = \frac{1}{1+e^{-a}} \tag{6.7}$$

- Hyperbolic tangent:
$$h(a) = tanh(a) \tag{6.8}$$

- Rectified linear unit:
$$h(a) = ReLu(a) = max(0, a) \tag{6.9}$$

Another way to represent a feed-forward neural networks is the following:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = h^{(4)}(a^{(4)}(h^{(3)}(a^{(3)}...h^{(1)}(a^{(1)})))) \tag{6.10}$$

Another generalization of the network architecture is to include skip-layer connections, each of which is associated with a corresponding adaptive parameter. For instance, in a two-layer network these would go directly from inputs to outputs. In principle, a network with sigmoidal hidden units can always mimic skip layer connections (for bounded input values) by using a sufficiently small first-layer weight that, over its operating range, the hidden unit is effectively linear, and then compensating with a large weight value from the hidden unit to the output. In practice, however, it may be advantageous to include skip-layer connections explicitly. Furthermore, the network can be sparse, with not all possible connections within a layer being present.

Theorem 6.0.1: Universal Approximators

Let f be any continuous function on a compact of $\mathbb{R}^D$, let h be any fixed analytic function which is **not** polynomial. Given any small number $\epsilon > 0$ of an acceptable error, we can find a number M and weights $\boldsymbol{w}^{(2)} \in \mathbb{R}^M$ and $\boldsymbol{W}^{(1)} \in \mathbb{R}^{MxD}$ such that:

$$|f(\boldsymbol{x}) - y(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{w}^{(2)})| < \epsilon \tag{6.11}$$

with a 2-layer NN:

$$y_k(\boldsymbol{x}, \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}) = \sum_{m=0}^{M} w_m^{(2)} h \left(\sum_{d=0}^{D} w_{md}^{(1)} x_d \right) \tag{6.12}$$

Caution: for smaller ϵ we usually need larger M

If we take 2 Neural nets, one with L layers and the other with $L' < L$ layers, we can approximate the deep neural net with shallow neural net up to error ϵ, usually the number of required units $M(\epsilon)$ of the shallow net scales exponentially for decreasing ϵ.

Which is the expressive power of ReLU networks? The expressive power of ReLU-DNN is computed as the number of linear regions, where $\#regions \approx width^{depth*D}$ which is polynomial in width, but exponential with depth. With fixed network capacity $\#parameters \approx width^2 * depth$, most expressive power is gained by going deeper with less neurons per layer respect than staying shallow with more neurons per layer. One property of feed-forward networks, which will play a role when we consider Bayesian model comparison, is that multiple distinct choices for the weight vector $\boldsymbol{w}$ can all give rise to the same mapping function from inputs to outputs.

6.1 Network Training

So far, we have viewed neural networks as a general class of parametric nonlinear functions from a vector x of input variables to a vector y of output variables. A simple approach to the problem of determining the network parameters is to make an analogy with the discussion of polynomial curve fitting, and therefore to minimize a sum-of-squares error function. Given a input dataset $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)^T, \ \ with \ \ \boldsymbol{x}_n \in \mathbb{R}^D$ and targets $\boldsymbol{t} = (t_1, ...t_N)^T$, we use a probabilistic interpretation of the network outputs to choose:

- Number of outputs
- Output activation function
- Loss function

We start by discussing regression problems, and for the moment we consider a single target variable t that can take any real value. As previously seen in these notes, we assume that t has a Gaussian distribution with an independent mean, which is given by the output of the neural network, so that:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(t|y(\boldsymbol{x}; \boldsymbol{w}), \beta^{-1}) \tag{6.13}$$

For this conditional distribution, it is sufficient to take the output unit activation function to be the identity $y(\boldsymbol{x}, \boldsymbol{w}) = h^{(L)}(a^{(out)}) = a^{(out)}$, because such a network can approximate any continuous function from $\boldsymbol{x}$ to y. In particular, we can construct the corresponding likelihood function:

$$p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} p(t_n|\boldsymbol{x}_n, \boldsymbol{w}, \beta) \tag{6.14}$$

taking the negative logarithm, we obtain the error function:

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \tag{6.15}$$

which can be used to learn the parameters $\boldsymbol{w}$ and β. Note that in the neural networks literature, it is usual to consider the minimization of an error function rather than the maximization of the log-likelihood. Consider first the determination of $\boldsymbol{w}$, minimizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\boldsymbol{x}_n - \boldsymbol{w}) - t_n\}^2 \tag{6.16}$$

In practice, the nonlinearity of the network function $y(\boldsymbol{x}_n, \boldsymbol{w})$ causes the error $E(\boldsymbol{w})$ to be nonconvex, and so in practice local maxima of the likelihood may be found, corresponding to local minima of the error function. Having found $\boldsymbol{w}_{ML}$, the value of β can be found by minimizing the negative log-likelihood to give:

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\boldsymbol{x}_n - \boldsymbol{w}_{ML}) - t_n\}^2 \tag{6.17}$$

Note that this can be evaluated once the iterative optimization required to find $\boldsymbol{w}_{ML}$ is completed. If we have multiple target variables, and we assume that they are independent conditional on $\boldsymbol{x}$ and $\boldsymbol{w}$ with shared noise precision β, then the conditional distribution of the target values is given by

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{t}|\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1}\mathcal{I}) \tag{6.18}$$

Following the same argument as for a single target variable, we see that the maximum likelihood weights are determined by minimizing the sum-of-squares error function. The noise precision is then given by:

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{k=1}^{K}||y(\boldsymbol{x}_n - \boldsymbol{w}_{ML}) - t_n||^2 \tag{6.19}$$

where K is the number of target variables. The assumption of independence can be dropped at the expense of a slightly more complex optimization problem. Recall from previous sections that there is a natural pairing of the error function (given by the negative log likelihood) and the output unit activation function. In the regression case, we can view the network as having an output activation function that is the identity, so that $y_k = a_k$. The corresponding sum-of-squares error function has the property:

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{6.20}$$

Now consider the case of binary classification in which we have a single target variable t such that $t = 1$ denotes class C_1 and $t = 0$ denotes class C_2. We consider a network having a single output whose activation function is a logistic sigmoid so that $0 \leq y(\boldsymbol{x}, \boldsymbol{w}) \leq 1$. We can interpret $y(\boldsymbol{x}, \boldsymbol{w}) = \sigma(a^{out})$ as the conditional probability $p(C_1|\boldsymbol{x})$, with $p(C_2|\boldsymbol{x})$ given by $1 - y(\boldsymbol{x}, \boldsymbol{w})$. The conditional distribution of targets given inputs is then a Bernoulli distribution of the form:

$$p(t|\boldsymbol{x}, \boldsymbol{w}) = y(\boldsymbol{x}, \boldsymbol{w})^t \{1 - y(\boldsymbol{x}, \boldsymbol{w})\}^{1-t} \tag{6.21}$$

If we consider a training set of independent observations, then the error function, which is given by the negative log likelihood, is then a **cross-entropy** error function of the form:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{6.22}$$

where y_n denotes $y(\boldsymbol{x}_n, \boldsymbol{w})$. Note that there is no analogue of the noise precision β because the target values are assumed to be correctly labelled.

If we have K separate binary classifications to perform, then we can use a network having K outputs each of which has a logistic sigmoid activation function. Associated with each output is a binary class label $t_k \in \{0, 1\}$, where $k = 1, ..., K$. If we assume that the class labels are independent, given the input vector, then the conditional distribution of the targets is:

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}) = \prod_{k=1}^{K} y_k(\boldsymbol{x}, \boldsymbol{w})^{t_k} [1 - y_k(\boldsymbol{x}, \boldsymbol{w})]^{1-t_k} \tag{6.23}$$

Taking the negative logarithm of the corresponding likelihood function then gives the following error function

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\} \tag{6.24}$$

where y_{nk} denotes $y_k(\boldsymbol{x}_n, \boldsymbol{w})$. Again, the derivative of the error function with respect to the activation for a particular output unit takes the previous seen form just as in the regression case.

Finally, we consider the standard multiclass classification problem in which each input is assigned to one of K mutually exclusive classes. The binary target variables have a 1-of-K coding scheme indicating the class, and the network outputs are interpreted as $y_k(\boldsymbol{x}, \boldsymbol{w}) = h^{(L)}(\boldsymbol{a}^{out}) = \frac{exp(a_K^{out})}{\sum_{j=1}^{K} exp(a_j^{out})}$ leading to the following error function:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln(y_k(\boldsymbol{x}_n, \boldsymbol{w})) \tag{6.25}$$

In summary, there is a natural choice of both output unit activation function and matching error function, according to the type of problem being solved. For regression we use **linear outputs** and a **sum-of-squares error**, for (multiple independent) binary classifications we use **logistic sigmoid outputs** and a **cross-entropy error** function, and for multiclass classification we use **softmax outputs** with the corresponding **multiclass cross-entropy error** function. For classification problems involving two classes, we can use a single logistic sigmoid output, or alternatively we can use a network with two outputs having a softmax output activation function.

6.2 Parameter optimization

For each task we have a different loss function $E(\boldsymbol{w})$ that we want to minimize. In particular we want to find the weight vector $\boldsymbol{w}^*$ such that $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{argmin} E(\boldsymbol{w})$. The error function can be viewed as a surface, sitting over the weight space. Note that if we make a small step in the weight space from $\boldsymbol{w}$ to $\boldsymbol{w} + \delta\boldsymbol{w}$ then the change in the error function is $\delta E \approx \delta\boldsymbol{w}^T \bigtriangledown E(\boldsymbol{w})$ where the vector $\bigtriangledown E(\boldsymbol{w})$ points in the direction of greatest rate of increase of the error function. Because the error $E(\boldsymbol{w})$ is a **smooth continuous** function of $\boldsymbol{w}$, its smallest value will occur at a point in weight space such that the gradient of the error function vanishes, so that $\bigtriangledown E(\boldsymbol{w}) = 0$ as otherwise we could make a small step in the direction of $- \bigtriangledown E(\boldsymbol{w})$ and thereby further reduce the error. Points at which the gradient vanishes are called stationary points, and may be further classified into minima, maxima, and saddle points. Our goal is to find a vector w such that $E(\boldsymbol{w})$ takes its smallest value. However, the error function typically has a highly nonlinear dependence on the weights and bias parameters, and so there will be many points in weight space at which the gradient vanishes (or is numerically very small). For a successful application of neural networks, it may not be necessary to find the global minimum (and in general it will not be known whether the global minimum has been found) but it may be necessary to compare several local minima in order to find a sufficiently good solution (we usually test different random $\boldsymbol{w}^{(0)}$ to report unvertainties on performance). Because there is clearly no hope of finding an analytical solution to the equation $\bigtriangledown E(\boldsymbol{w}) = 0$ we resort to iterative numerical procedures. The optimization of continuous nonlinear functions is a widely studied problem and there exists an extensive literature on how to solve it efficiently. Most techniques (like stochastic gradient descent) involve choosing some initial value $\boldsymbol{w}^{(0)}$ for the weight vector and then moving through weight space in a succession of steps.

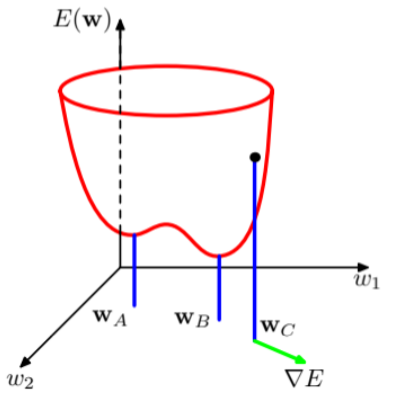


Figure 6.2: Shape of the error function

6.3 Backpropagation

We now derive the backpropagation algorithm for a general network having arbitrary feed-forward topology, arbitrary differentiable nonlinear activation functions, and a broad class of error function. Many error functions of practical interest, for instance those defined by maximum likelihood for a set of i.i.d. data, comprise a sum of terms, one for each data point in the training set, so that

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}) \tag{6.26}$$

First, from the beginning of this chapter recall that:

- The j-th nodes in the layer (l) computes a weighted sum of its inputs of the form:

$$a_j = \sum_i w_{ji}^{(l)} z_i^{(l-1)} \tag{6.27}$$

- This sum is then transformed by a nonlinear function $h(\cdot)$ to give:

$$z_j = h(a_j) \tag{6.28}$$

For each pattern in the training set, we shall suppose that we have supplied the corresponding input vector to the network and calculated the activations of all of the hidden and output units in the network by successive application of the above formulas. This process is often called **forward propagation** because it can be regarded as a forward flow of information through the network. Now consider the evaluation of the derivative of E_n with respect to a weight w_{ji}. The outputs of the various units will depend on the particular input pattern n. However, in order to keep the notation uncluttered, we shall omit the subscript n from the network variables. First we note that E_n depends on the weight $w_{ji}^{(l)}$ only via the summed input $a_j^{(l)}$ to unit j . We can therefore apply the chain rule for partial derivatives to give:

$$\frac{\partial}{\partial w_{ji}^{(l)}} E_n = \frac{\partial E_n}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{6.29}$$

Let's define the "node error" as

$$\delta_j^{(l)} \equiv \frac{\partial E_n}{\partial a_j^{(l)}}$$

from the definition of activations (a_j) we obtain

$$\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$$

thus, putting all together

$$\frac{\partial}{\partial w_{ji}^{(l)}} E_n = \delta_j^{(l)} z_i^{(l-1)}$$

This last equation tells us that the required derivative is obtained simply by multiplying the value of δ for the unit at the output end of the weight by the value of z for the unit at the input end of the weight (where z = 1 in the case of a bias). In order to evaluate the derivatives, we need only to calculate the value of δ_j for each hidden and output unit in the network, and then apply the formula above. As we have seen already, for the output units, we have

$$\delta_k = y_k - t_k \tag{6.30}$$

provided we are using the canonical link as the output-unit activation function. To evaluate the $\delta's$ for hidden units, we again make use of the chain rule for partial derivatives

$$\begin{aligned}\delta_j^{(l)} = \frac{\partial E_n}{\partial a_j^{(l)}} &= \sum_m \frac{\partial E_n}{\partial a_m^{(l+1)}} \frac{\partial a_m^{(l+1)}}{\partial a_j^{(l)}} \\ &= \sum_m \delta_m \frac{\partial a_m^{(l+1)}}{\partial a_j^{(l)}} \\ &= \sum_m \delta_m \frac{\partial}{\partial a_j^{(l)}} \left(\sum_j w_{mj}^{(l+1)} h(a_j^{(l)}) \right) \\ &= h(a_j^{(l)}) \sum_m \delta_m w_{mj}^{(l+1)} \end{aligned} \tag{6.31}$$

which tells us that the value of δ for a particular hidden unit can be obtained by propagating the $\delta's$ backwards from units higher up in the network. Note that the last summation is taken over the first index on w_{mj} (corresponding to backward propagation of information through the network), whereas in the forward propagation equation it is taken over the second index. Because we already know the values of the $\delta's$ for the output units, it follows that by recursively applying this last formula we can evaluate the $\delta's$ for all of the hidden units in a feed-forward network, regardless of its topology.
Summary of error backpropagation:

1. Apply an input vector $\boldsymbol{x}_n$ to the network and forward propagate through the network using 6.27 and 6.28 to find the activations of all the hidden and output units.

2. Evaluate the δ_m for all the output units using 6.30.

3. Backpropagate the $\delta's$ using 6.31 to obtain δ_j for each hidden unit in the network.

4. Use (5.53) to evaluate the required derivatives.

Chapter 7

Mixture models

While in supervised learning our goal was from a dataset $D = \{\boldsymbol{X}, \boldsymbol{T}\}$ infer a $f(\boldsymbol{x}) \approx t,\ \ p(\boldsymbol{t}|\boldsymbol{x})$, in unsupervised learning we have a dataset comprising of only samples unlabelled $D = \{\boldsymbol{X}\}$ and our goal is to determine $p(\boldsymbol{x}), p(\boldsymbol{z}|\boldsymbol{x}), p(\boldsymbol{x}|\boldsymbol{z})$ through methods like density estimation, clustering (discrete) or dimensionality reduction (continuous). Note that in this case $\boldsymbol{z}$ is called **latent variable**. Let's introduce the **latent variable models**; if we define a joint distribution over observed $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$, the corresponding distribution of the observed variables alone is obtained by marginalization. This allows relatively complex marginal distributions over observed variables to be expressed in terms of more tractable joint distributions over the expanded space of observed and latent variables. The introduction of latent variables thereby allows complicated distributions to be formed from simpler components.

- Discrete:
$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z})$$

- Continuous:
$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \int p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} \tag{7.1}$$

As well as providing a framework for building more complex probability distributions, mixture models can also be used to cluster data. We therefore begin our discussion of mixture distributions by considering the problem of finding clusters in a set of data points, which we approach first using a non probabilistic technique called the K-means algorithm. Then we introduce the latent variable view of mixture distributions in which the discrete latent variables can be interpreted as defining assignments of data points to specific components of the mixture.

7.1 K-means Clustering

We begin by considering the problem of identifying groups, or clusters, of data points in a multidimensional space. Suppose we have a dataset $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ consisting of N observations of a random D-dimensional variable $\boldsymbol{x}$. Our goal is to partition the data set into some number K of clusters (a discrete latent variable), where we shall suppose for the moment that the value of K is given. We can formalize this notion by first introducing a set of D-dimensional vectors $\boldsymbol{\mu}_k$, where $k = 1, ..., K$, in which $\boldsymbol{\mu}_k$ is a prototype associated with the k-th cluster. As we shall see shortly, we can think of the $\boldsymbol{\mu}_k$ as representing the centers of the clusters. Our goal is then to find an assignment of data points to clusters, as well as a set of vectors $\boldsymbol{\mu}_k$, such that the sum of the squares of the distances of each data point to its closest vector $\boldsymbol{\mu}_k$, is a minimum. For each data point $\boldsymbol{x}_n$, we introduce a 1-hot-encoding scheme with $z_{nk} = 1$ if and only if point n is assigned to cluster k. We can then define an objective function, sometimes called a distortion measure, given by:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \tag{7.2}$$

which represents the sum of the squares of the distances of each data point to its assigned vector $\boldsymbol{\mu}_k$. Our goal is to find values for the $\{z_{nk}\}$ and the $\{\boldsymbol{\mu}_k\}$ so as to minimize J. We can do this through an iterative procedure in which each iteration involves two successive steps corresponding to successive optimizations with respect to the z_{nk} and the $\boldsymbol{\mu}_k$. First we choose some initial random values for the $\boldsymbol{\mu}_k$. Then in the first phase we minimize J with respect to the z_{nk} , keeping the $\boldsymbol{\mu}_k$ fixed. In the second phase we minimize J with respect to the $\boldsymbol{\mu}_k$, keeping z_{nk} fixed. This two-stage optimization is then repeated until convergence. We shall see that these two stages of updating z_{nk} and updating $\boldsymbol{\mu}_k$ correspond respectively to the E (expectation) and M (maximization) steps of the EM algorithm, and to emphasize this we shall use the terms E step and M step in the context of the K-means algorithm.

Note that J is a linear function of z_{nk} so this optimization can be performed easily to give a closed form solution.
The E-step can be summarized like this:

$$z_{nk} = \begin{cases} 1 & if \;\; k = \underset{j}{argmin} ||\boldsymbol{x}_n - \boldsymbol{\mu}_j||^2 \\ 0 & otherwise \end{cases} \tag{7.3}$$

The M-step:

$$\boldsymbol{\mu}_k = \frac{\sum_n z_{nk} \boldsymbol{x}_n}{\sum_n z_{nk}} \tag{7.4}$$

The denominator in this expression is equal to the number of points assigned to cluster k, and so this result has a simple interpretation, namely set $\boldsymbol{\mu}_k$ equal to the mean of all of the data points $\boldsymbol{x}_n$ assigned to cluster k. For this reason, the procedure is known as the K-means algorithm. Because each phase reduces the value of the objective function J, convergence of the algorithm is assured. However, it may converge to a local rather than global minimum of J. Since J is a convex function the M step is derived by differentiating w.r.t $\boldsymbol{\mu}_k$ and setting the gradient to 0.

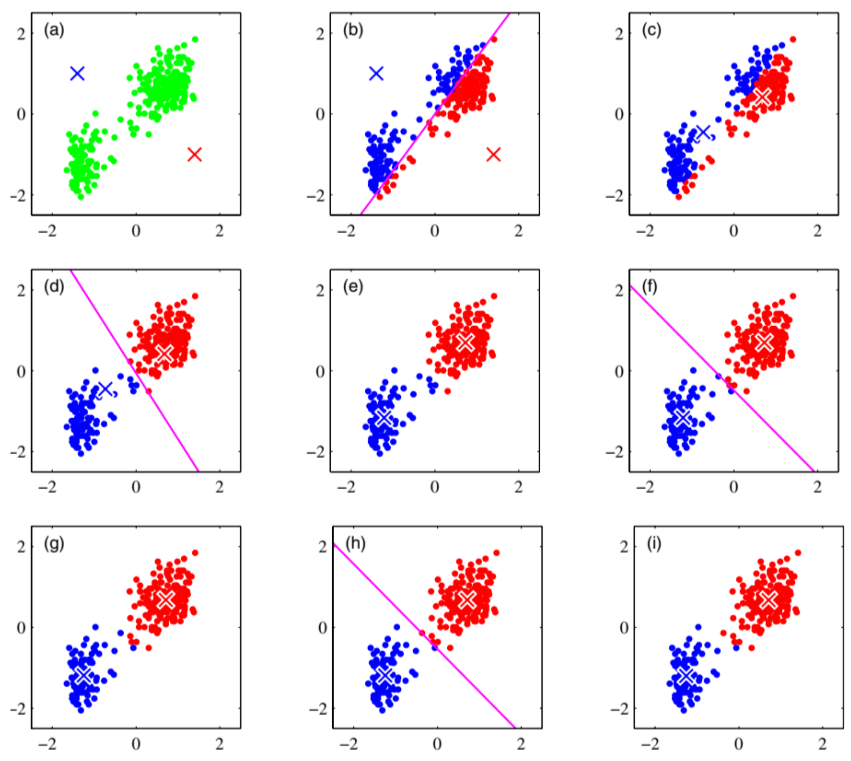


Figure 7.1: K means example

So far, we have considered a batch version of K-means in which the whole data set is used together to update the prototype vectors. We can also derive an on-line stochastic algorithm by applying the Robbins-Monro procedure to the problem of finding the roots of the regression function given by the derivatives of J with respect to $\boldsymbol{\mu}_k$. This leads to a sequential update in which, for each data point $\boldsymbol{x}_n$ in turn, we update the nearest prototype $\boldsymbol{\mu}_k$ using

$$\boldsymbol{\mu}_k^{new} = \boldsymbol{\mu}_k^{old} + \eta_n(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{old}) \tag{7.5}$$

where η_n is the learning rate parameter, which is typically made to decrease monotonically as more data points are considered. The K-means algorithm is based on the use of squared Euclidean distance as the measure of dissimilarity between a data point and a prototype vector. Not only does this limit the type of data variables that can be considered (it would be inappropriate for cases where some or all of the variables represent categorical labels for instance), but it can also make the determination of the cluster means non-robust to outliers.

We can generalize the K-means algorithm by introducing a more general dissimilarity measure $V(\boldsymbol{x}, \boldsymbol{x}')$ between two vectors x and x and then minimizing the following distortion measure

$$\tilde{J} = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk} V(\boldsymbol{x}_n, \boldsymbol{\mu}_k) \tag{7.6}$$

which gives the K-medoids algorithm.

Some pros of the K-means algorithm are that it is simple to implement and fast. The cons are that it can get stuck in local minima, it can model only spherical clusters, it is sensible to feature scale and the number of clusters need to be chosen in advance.

7.2 Gaussian Mixture Model

The Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{7.7}$$

In the GMM a discrete (latent) random variable $\boldsymbol{z}$ picks the cluster (the mixture component) and points in the cluster $\boldsymbol{x}$ are Gaussian distributed. Let's introduce the one-hot encoded K-dimensional binary random variable $\boldsymbol{z}$ which represent our latent variable. The values of z_k satisfy $z_k \in \{0, 1\}$ and $\sum_k z_k = 1$. We define now the marginal distribution $p(\boldsymbol{x})$ in terms of the joint distribution, thus obtaining:

$$p(\boldsymbol{x}) = \sum_{z} p(\boldsymbol{x}, z) = \sum_{z} p(\boldsymbol{x}|z)p(z) \tag{7.8}$$

The marginal distribution over $\boldsymbol{z}$ is specified in terms of the mixing coefficient π_k such that:

$$p(z_k = 1) = \pi_k \tag{7.9}$$

The conditional distribution of $\boldsymbol{x}$ given a particular value for $\boldsymbol{z}$ is a Gaussian

$$p(\boldsymbol{x}|z_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{7.10}$$

(This means that the clusters are Gaussians with different parameters).

In the end we obtain the full **generative model**:

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{7.11}$$

We have therefore found an equivalent formulation of the Gaussian mixture involving an explicit latent variable. Our goal is to partition the data into K clusters by maximizing the likelihood of the probabilistic model $p(\boldsymbol{x})$. Another quantity that will play an important role is the conditional probability of $\boldsymbol{z}$ given $\boldsymbol{x}$. We shall use $\gamma(z_k)$ to denote $p(z_k = 1|\boldsymbol{x})$, whose value can be found using Bayes' theorem:

$$\gamma(z_k) = p(z_k = 1|\boldsymbol{x}) = \frac{p(z_k = 1)p(\boldsymbol{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\boldsymbol{x}|z_j = 1)} = \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{7.12}$$

We shall view π_k as the prior probability of $z_k = 1$, and the quantity $\gamma(z_k)$ as the corresponding posterior probability once we have observed $\boldsymbol{x}$. As we will see later, $\gamma(z_k)$ can also be viewed as the responsibility that component k takes for 'explaining' the observation $\boldsymbol{x}$.

A key distinction between GMM and K-means is that $p(\boldsymbol{z}|\boldsymbol{x})$ represents the probability that a sample belongs to each cluster, whereas K-means assigns each sample to a single cluster deterministically. This probabilistic assignment in GMM is known as **soft clustering**.

The GMM are **generative models** because we can use the technique of ancestral sampling to generate random samples distributed according to the Gaussian mixture model. To do this, we first generate a value for $\boldsymbol{z}$, which we denote $\hat{\boldsymbol{z}}$, from the marginal distribution $p(\boldsymbol{z})$ and then generate a value for $\boldsymbol{x}$ from the conditional distribution $p(\boldsymbol{x}|\boldsymbol{z})$. We can depict samples from the joint distribution $p(\boldsymbol{x}, \boldsymbol{z})$ by plotting points at the corresponding values of $\boldsymbol{x}$ and then colouring them according to the value of $\boldsymbol{z}$, in other words according to which Gaussian component was responsible for generating them. Similarly samples from the marginal distribution $p(\boldsymbol{x})$ are obtained by taking the samples from the joint distribution and ignoring the values of $\boldsymbol{z}$.

Given a dataset of observations $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}, \boldsymbol{x}_i \in \mathbb{R}^D$, if we assume that the data points are drawn independently from the distribution, then we can derive the log-likelihood in the following form:

$$\begin{aligned} \ln p(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \ln \prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= \sum_{n=1}^{N} \ln p(\boldsymbol{x}_n|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned} \tag{7.13}$$

We need to maximize the likelihood with respect to $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \ \ \forall k = 1, ..., K$, but he problem is non-convex so there is not a closed form solution, stationary points depends on the posterior $\gamma(z_{nk})$. An elegant and powerful method for finding maximum likelihood solutions for models with latent variables is called the **expectation-maximization algorithm**, or EM algorithm. It alternates 2 phases: the updates of the (**expected**) posterior $\gamma(z_{nk})$ and the **maximization** for $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$.
The trick is to solve with $\gamma(z_{nk})$ fixed; first we choose some initial value for the means, covariances and mixing coefficients. Then we alternate between the **update of the expected posterior** $\gamma(z_{nk})$ and **the maximization** of $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ which result will depend on the updated expected posterior. We shall show that each update to the parameters resulting from an E step followed by an M step is guaranteed to increase the log likelihood function. In practice, the algorithm is deemed to have converged when the change in the log likelihood function, or alternatively in the parameters, falls below some threshold. The Figure 7.2 illustrates the EM algorithm for a mixture of two Gaussians.

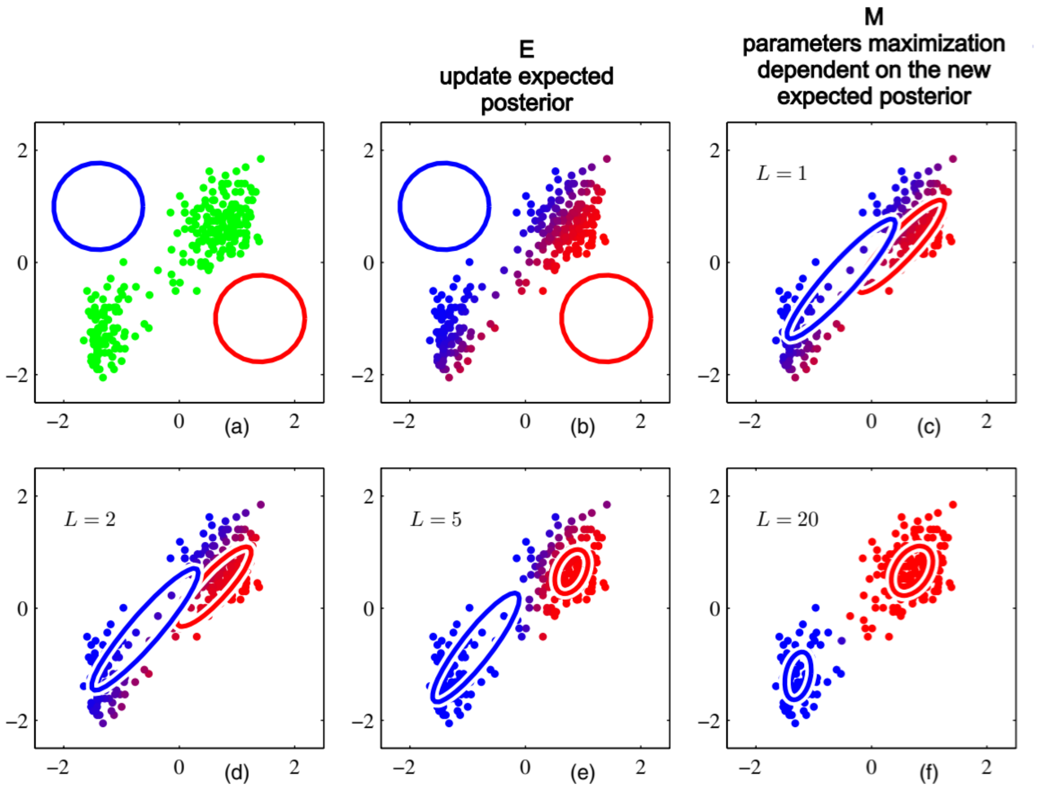


Figure 7.2: Illustration of the EM algorithm

Note that the EM algorithm takes many more iterations to reach (approximate) convergence compared with the K-means algorithm, and that each cycle requires significantly more computation. It is therefore common to run the K-means algorithm in order to find a suitable initialization for a Gaussian mixture model that is subsequently adapted using EM. The covariance matrices can conveniently be initialized to the sample covariances of the clusters found by the K-means algorithm, and the mixing coefficients can be set to the fractions of data points assigned to the respective clusters. As with gradient-based approaches for maximizing the log likelihood, techniques must be employed to avoid singularities of the likelihood function in which a Gaussian component collapses onto a particular data point. It should be emphasized that there will generally be multiple local maxima of the log likelihood function, and that EM is not guaranteed to find the largest of these maxima.

Let us now see in detail the maximization (and derivations) required by the EM algorithm:

Maximization with respect to $\boldsymbol{\mu}_k$

Set the derivative w.r.t $\boldsymbol{\mu}_k$ of the log-likelihood to 0:

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\mu}_k} \sum_{n=1}^{N} \log p(\boldsymbol{x}_n | \{\pi_k\}, \{\boldsymbol{\mu}_k\}, \{\boldsymbol{\Sigma}_k\}) \\
&\quad = \sum_{n=1}^{N} \frac{1}{p(\boldsymbol{x}_n | \{\pi_k\}, \{\boldsymbol{\mu}_k\}, \{\boldsymbol{\Sigma}_k\})} \frac{\partial}{\partial \boldsymbol{\mu}_k} p(\boldsymbol{x}_n | \{\pi_k\}, \{\boldsymbol{\mu}_k\}, \{\boldsymbol{\Sigma}_k\}) \\
&\quad = \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}_{\gamma_{z_{nk}}} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}
\end{aligned}$$

$$\sum_{n=1}^{N} \gamma_{z_{nk}} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} = 0 \quad \Longrightarrow \quad \boldsymbol{\mu}_k^{new} = \frac{\sum_{n=1}^{N} \gamma(z_{nk} \boldsymbol{x}_n)}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{7.14}$$

Maximization with respect to π_k

Set the derivative w.r.t π_k of the log-likelihood to 0, with the constraint that $\underbrace{\sum_k \pi_k}_{g(x)} = \underbrace{1}_{c}$

$$\frac{\partial}{\partial \pi_k} \underbrace{\sum_{n=1}^{N} \log p(\boldsymbol{x}_n | \{pi_k\}, \{\boldsymbol{\mu}_k^{new}\}, \{\boldsymbol{\Sigma}_k^{new}\})}_{f(x)} + \underbrace{\lambda \sum_{j=1}^{K} (\pi_j - 1)}_{\lambda(g(x)-c)}$$

$$= \sum_{n=1}^{N} \underbrace{\frac{\mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k^{new}, \boldsymbol{\Sigma}_k^{new})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_j^{new}, \boldsymbol{\Sigma}_j^{new})}}_{\frac{1}{\pi_k}\gamma(z_{nk})} + \lambda$$

$$\lambda + \sum_{n=1}^{N} \frac{1}{\pi_k} \gamma(z_{nk}) = 0 \quad \Longrightarrow \quad \pi_k = -\frac{1}{\lambda} \sum_{n=1}^{N} \gamma(z_{nk}) \tag{7.15}$$

$$\frac{\partial}{\partial \lambda} L(\{\pi_k\}), \lambda) = \sum_{j=1}^{K} \pi_j - 1 = -\frac{1}{\lambda} \sum_{j=1}^{K} \sum_{n=1}^{N} \gamma(z_{nj}) - 1 = 0$$

$$\lambda = -N \tag{7.16}$$

Note that, π_k is the fraction of points for which cluster k takes responsibility. If we define the "effective number of points in cluster k as

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{7.17}$$

From Equation 7.15 we obtain

$$\pi_k = \frac{N_k}{N} \tag{7.18}$$

Maximization with respect to $\boldsymbol{\Sigma}_k$

Set the derivative w.r.t $\boldsymbol{\Sigma}_k$ of the log-likelihood to 0:

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})(\boldsymbol{x}_n - \boldsymbol{\mu}_j^{new})^T \tag{7.19}$$

Chapter 8

Continuous latent variables

In the previous chapter we discussed probabilistic models having discrete latent variables, such as the mixture of Gaussians. We now explore models in which some, or all, of the latent variables are continuous. An important motivation for such models is that many data sets have the property that the data points all lie close to a manifold of much lower dimensionality than that of the original data space, this is called the **manifold hypothesis**.

Theorem 8.0.1: Manifold hypothesis

This hypothesis states that high dimensional data is actually (often, in theory) "embedded" in a lower dimensional manifold, and machine learning models are trying to learn the geometry of this manifold. This can essentially be restated as "there exists a lower dimensional representation of the data which preserves proximity relationships between observations, and in which observations are potentially more accurately discriminated relative to the target variable.

Example Let's consider an artificial data set constructed by taking one of the off-line digits, represented by a 100 x 100 pixel grey-level image, in which the location and orientation of the digit is varied at random. Each of the resulting images is represented by a point in the 100 x 100 = 10,000-dimensional data space. However, across a data set of such images, there are only three degrees of freedom of variability, corresponding to the vertical and horizontal translations and the rotations. The data points will therefore live on a subspace of the data space whose intrinsic dimensionality is three. Note that the manifold will be nonlinear because, for instance, if we translate the digit past a particular pixel, that pixel value will go from zero (white) to one (black) and back to zero again. which is clearly a nonlinear function of the digit position. In this example. the translation and rotation parameters are latent variables because we observe only the image vectors and are not told which values of the translation or rotation variables were used to create them. A more realistic dataset of images will have more degrees of freedom in the latent space, such as: Scaling, Digits from 0-9, Colors, Different hand-writing styles, etc... but still much fewer than 100x100!

In this chapter we will begin with a standard, non probabilistic treatment of PCA, and then we show how PCA arises naturally as the maximum likelihood solution to a particular form of linear-Gaussian latent variable model. This probabilistic reformulation brings many advantages, such as the use of EM for parameter estimation, principles extension to mixtures of PCA models, and Bayesian formulations that allow the number of principal components to be determined automatically from the data.

8.1 Principal Component Analysis

Principal Component Analysis or PCA is a technique that is widely used for applications such as dimensionality reduction, lossy data compression, feature extraction, and data visualization. There are two commonly used definitions of PCA that give rise to the same algorithm. PCA can be defined as the **orthogonal projection** of the data **onto a lower dimensional linear space**, known as the *principal subspace*, such that the variance of the projected data is maximized. Equivalently, it can be defined as the linear projection that minimizes the average projection cost, defined as the mean squared distance between the data points and their projections.

8.1.1 Maximum variance formulation

Given a dataset of observations $\{\boldsymbol{x}_n\}\ \ with\ \ n = 1, ..., N$, our goal is to project the data onto a space having dimensionality $M < D$ while maximizing the variance of the projected data. For the moment, we shall assume that the value of M is given. To begin with, consider the projection onto a one-dimensional space ($M = 1$) (the first latent dimension). We can define the direction of this space using a D-dimensional vector $\boldsymbol{u}_1 \in \mathbb{R}^D$, which for convenience (and without loss of generality) we shall choose to be a unit vector so that $\boldsymbol{u}_1^T\boldsymbol{u}_1 = 1$ (note that we are only interested in the direction defined by $\boldsymbol{u}_1$, not in the magnitude of $\boldsymbol{u}1$ itself). Each data point $\boldsymbol{x}_n$ is then projected onto a scalar value $z_{n1} = \boldsymbol{u}_1^T\boldsymbol{x}_n$. The mean of the projected data is $\boldsymbol{u}_1^T\bar{\boldsymbol{x}}$ where $\bar{\boldsymbol{x}}$ is the sample mean given by

$$\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n \tag{8.1}$$

and the variance of the projected data is given by

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}\{\boldsymbol{u}_1^T\boldsymbol{x}_n - \boldsymbol{u}_1^T\bar{\boldsymbol{x}}\}^2 &= \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{u}_1^T(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^2 \\ &= \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{u}_1^T(\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^T\boldsymbol{u}_1 \\ &= \boldsymbol{u}_1^T\left(\underbrace{\frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^T}_{\boldsymbol{S}}\right)\boldsymbol{u}_1 \\ &= \boldsymbol{u}_1^T\boldsymbol{S}\boldsymbol{u}_1\end{aligned} \tag{8.2}$$

where $\boldsymbol{S}$ is the data covariance matrix defined by:

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^T \tag{8.3}$$

which is symmetric and positive definite thus $\forall j \ : \ \lambda_j \geq 0$

We now maximize the projected variance $\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1$ with respect to $\boldsymbol{u}_1$ with the constraint that $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$, to do so we define the Lagrangian:

$$L(\boldsymbol{\mu}_1, \lambda_1) = \boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 + \lambda_1(\boldsymbol{u}_1^T \boldsymbol{u}_1 - 1) \tag{8.4}$$

Solving for $\boldsymbol{u}_1$

$$\frac{\partial}{\partial \boldsymbol{u}_1} L(\boldsymbol{u}_1, \lambda_1) = \boldsymbol{S}\boldsymbol{u}_1 - \lambda_1 \boldsymbol{u}_1$$

$$\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1 \tag{8.5}$$

So $\boldsymbol{u}_1$ and λ_1 are respectively an eigenvector and eigenvalue of $\boldsymbol{S} \in \mathbb{R}^{D\times D}$, $\boldsymbol{u}_1$ is called **principal component**, the variance of the projected data is $\boldsymbol{u}_1^T \boldsymbol{S} \boldsymbol{u}_1 = \lambda_1$. Maximizing the variance means searching for the eigenvector with the largest eigenvalue.

We can define additional principal components in an incremental fashion by choosing each new direction to be the one which maximizes the projected variance amongst all possible directions orthogonal to those already considered. If we consider the general case of an M -dimensional projection space, the optimal linear projection for which the variance of the projected data is maximized is now defined by the projection $U_M = [\boldsymbol{u}_1, ..., \boldsymbol{u}_M] \in \mathbb{R}^{D\times M}$ which is the set of the M eigenvectors of the data covariance matrix $\boldsymbol{S}$ corresponding to the M largest eigenvalues.

The total variance of the projected data is $Tr[Cov[\boldsymbol{z}]] = \sum_{j=1}^{M} \lambda_j$

How do we obtain the eigenvectors of the covariance matrix? When the matrix is symmetric positive semi-definite we can perform the eigen-decomposition (for symmetric positive definite the SVD is equivalent).

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T \quad with \quad \boldsymbol{\Lambda} = diag\{\lambda_1, ..., \lambda_D\} \tag{8.6}$$

The eigenvectors are **orthonormal** and are stored in $\boldsymbol{U} = [\boldsymbol{u}_1, ..., \boldsymbol{u}_D]$, all eigenvalue are **non-negative** and are the elements of the diagonal matrix $\boldsymbol{\Lambda}$. The full eigen-decomposition is expensive $O(D^3)$ while we need only up to the M^{th} component: $O(MD^2)$.

How to choose M?

We can measure the discarded variance, for example to preserve the 90% of the variance we select M such that $\frac{\sum_{j=1}^{M}\lambda_j}{Tr(\boldsymbol{S})} > 0.9$, where the fract term is the proportion of explained variance. When data is defined in high dimension (large D) we want to project down to lower dimension because it reduce time and storage space required, and for classification/regression our model will have less parameters, thus we need less data points for learning.

A good side effect of PCA is that features have **no correlation** in the projected space, the covariance matrix of the projected data is thus **diagonal**.

$$\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{z}_n\boldsymbol{z}_n^T = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{U}_M^T(\boldsymbol{x}_n-\bar{\boldsymbol{x}})(\boldsymbol{x}_n-\bar{\boldsymbol{x}})^T\boldsymbol{U}_M$$
$$= \boldsymbol{U}_M^T\boldsymbol{S}\boldsymbol{U}_M = \boldsymbol{U}_M^T\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T\boldsymbol{U}_M = \boldsymbol{\Lambda} \tag{8.7}$$

Applications

Before applying learning algorithms we usually do some pre-processing: e.g. standardization: subtract the mean and divide by the standard deviation. With PCA we can whiten the data, one step more: Centre and de-correlate the features:

$$\boldsymbol{z} = \boldsymbol{U}_M^T(\boldsymbol{x}-\bar{\boldsymbol{x}}) \tag{8.8}$$

Then we can cast features to **unit standard deviation** by rescaling:

$$\boldsymbol{z} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}_M^T(\boldsymbol{x}-\bar{\boldsymbol{x}}) \tag{8.9}$$

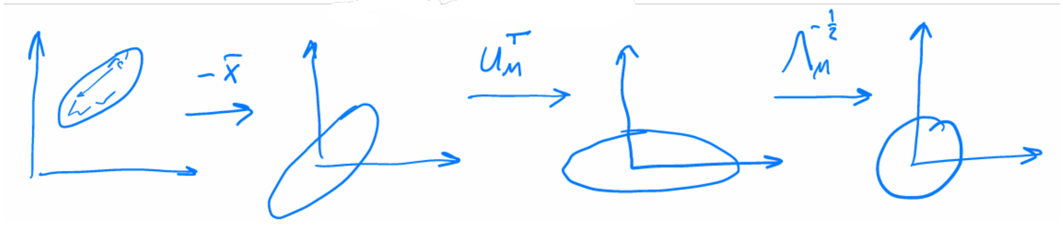


Figure 8.1: Example of pre-processing with PCA

8.1.2 Minimum error formulation

We now discuss an alternative formulation of PCA based on projection error minimization. We represent the points in a different orthonormal basis (unknown for now), $\{\boldsymbol{u}_i : \boldsymbol{u}_i^T\boldsymbol{u}_i = 1\}_{i=1}^{D}$ *with* $\boldsymbol{u}_j^T\boldsymbol{u}_i = 1$ *iff* $i = j$, $\boldsymbol{u}_j^T\boldsymbol{u}_i = 0$ *otherwise.* Our goal is to find a transformation that minimizes:

$$\frac{1}{N}\sum_{n=1}^{N}||\boldsymbol{x}_n-\tilde{\boldsymbol{x}}_n|| \tag{8.10}$$

with

$$\tilde{\boldsymbol{x}}_n = \boldsymbol{U}_M\boldsymbol{z}_n + \bar{\boldsymbol{x}} \tag{8.11}$$

where $\boldsymbol{z}_n$ is our latent variable / basis coefficients.

Because the basis is complete, each data point can be represented exactly by a linear combination of the basis vectors

$$\boldsymbol{x}_n = \sum_{i=1}^{D}\alpha_{ni}\boldsymbol{u}_i \tag{8.12}$$

where the coefficients α_{ni} will be different for different data points. This simply corresponds to a rotation of the coordinate system to a new system defined by the $\{\boldsymbol{u}_i\}$, where the original D components $\{x_{n1}, ..., x_{nD}\}$ are replaced by an equivalent set $\{\alpha_{n1}, ..., \alpha_{nD}\}$. This coefficient are found by multiplying with $\boldsymbol{u}_j^T$ on both sides:

$$\alpha_{ni} = \boldsymbol{x}_n^T \boldsymbol{u}_i \tag{8.13}$$

Therefore we get the projection onto the new basis

$$\boldsymbol{x}_n = \sum_{i=1}^{D} \underbrace{(\boldsymbol{x}_n^T \boldsymbol{u}_i)}_{\alpha_{ni}} \boldsymbol{u}_i \tag{8.14}$$

Our goal, however, is to approximate this data point using a representation involving a restricted number $M < D$ of variables corresponding to a projection onto a lower-dimensional subspace. The M-dimensional linear subspace can be represented, without loss of generality, by the first M of the basis vectors, and so we approximate each data point $\boldsymbol{x}_n$ by

$$\tilde{\boldsymbol{x}}_n = \sum_{i=1}^{M} z_{ni} \boldsymbol{u}_i + \sum_{i=M+1}^{D} b_i \boldsymbol{u}_i \tag{8.15}$$

where the $\{z_{ni}\}$ depend on the particular data point, whereas the $\{b_i\}$ are constants that are the same for all data points. The difference/error is given by

$$\begin{aligned} \boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n &= \underbrace{\sum_{i=1}^{D} (x_n^T \boldsymbol{u}_i) \boldsymbol{u}_i}_{\boldsymbol{x}_n} - \underbrace{\sum_{i=1}^{M} (x_n^T \boldsymbol{u}_i) \boldsymbol{u}_i + \sum_{i=M+1}^{D} b_i \boldsymbol{u}_i}_{\tilde{\boldsymbol{x}}_n} \\ &= \sum_{i=M+1}^{D} (\boldsymbol{x}_n^T \boldsymbol{u}_i) \boldsymbol{u}_i - \sum_{i=M+1}^{D} b_i \boldsymbol{u}_i \\ &= \sum_{i=M+1}^{D} (\boldsymbol{x}_n^T \boldsymbol{u}_i) \boldsymbol{u}_i - b_i \boldsymbol{u}_i \end{aligned} \tag{8.16}$$

If we choose to minimize the mean squared error we obtain as error function

$$\frac{1}{N} \sum_{n=1}^{N} ||\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n||^2 = \frac{1}{N} \sum_{n=1}^{N} || \sum_{i=M+1}^{D} (\boldsymbol{x}_n^T \boldsymbol{u}_i) \boldsymbol{u}_i - b_i \boldsymbol{u}_i||^2 \tag{8.17}$$

which we have to minimize bot for $\boldsymbol{u}_i$ and b_i. The solution for b_i is given by

$$b_i = \bar{\boldsymbol{x}}^T \boldsymbol{u}_i \tag{8.18}$$

Substituting Equation 8.18 in Equation 8.17 the solution for $\boldsymbol{u} : i$ is given by:

$$\begin{aligned} &\frac{1}{N} \sum_{n=1}^{N} \left|\left| \sum_{i=M+1}^{D} ((\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^T \boldsymbol{u}_i) \boldsymbol{u}_i \right|\right|^2 \\ &= \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{i=M+1}^{D} ((\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n)^T \boldsymbol{u}_i) \boldsymbol{u}_i \right)^T \left(\sum_{i=M+1}^{D} ((\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n)^T \boldsymbol{u}_i) \boldsymbol{u}_i \right) \end{aligned}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\sum_{j=M+1}^{D}((\boldsymbol{x}_n-\tilde{\boldsymbol{x}})^T\boldsymbol{u}_i)\underbrace{\boldsymbol{u}_i^T\boldsymbol{u}_j}_{0\ \forall\ i\neq j}((\boldsymbol{x}_n-\tilde{\boldsymbol{x}})^T\boldsymbol{u}_j)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\boldsymbol{u}_i^T(\boldsymbol{x}_n-\tilde{\boldsymbol{x}})(\boldsymbol{x}_n-\tilde{\boldsymbol{x}})^T\boldsymbol{u}_i = \sum_{i=M+1}^{D}\boldsymbol{u}_i^T\boldsymbol{S}\boldsymbol{u}_i \tag{8.19}$$

In the end we got

$$\frac{1}{N}\sum_{n=1}^{N}||\boldsymbol{x}_n-\tilde{\boldsymbol{x}}_n||^2 = \sum_{i=M+1}^{D}\boldsymbol{u}_i^T\boldsymbol{S}\boldsymbol{u}_i = \sum_{i=M+1}^{D}\lambda_i \tag{8.20}$$

There remains the task of minimizing J with respect to the $\{\boldsymbol{u}_i\}$, which must be a constrained minimization ($\boldsymbol{u}_i^T\boldsymbol{u}_i = 1$) otherwise we will obtain the vacuous result $\boldsymbol{u}_i = 0$. The constraints arise from the orthonormality conditions and, as we shall see, the solution will be expressed in terms of the eigenvector expansion of the covariance matrix.

8.1.3 Probabilistic PCA

The formulation of PCA discussed in the previous section was based on a linear projection of the data onto a subspace of lower dimensionality than the original data space. We now show that PCA can also be expressed as the maximum likelihood solution of a probabilistic latent variable model. This reformulation of PCA, known as probabilistic PCA, brings several advantages compared with conventional PCA:

- Probabilistic PCA represents a constrained form of the Gaussian distribution in which the number of free parameters can be restricted while still allowing the model to capture the dominant correlations in a data set.
- We can derive an EM algorithm for PCA that is computationally efficient.
- The combination of a probabilistic model and EM allows us to deal with missing values in the data set.
- Probabilistic PCA forms the basis for a Bayesian treatment of PCA in which the dimensionality of the principal subspace can be found automatically from the data.
- The existence of a likelihood function allows direct comparison with other probabilistic density models.
- Probabilistic PCA can be used to model class-conditional densities and hence be applied to classification problems.
- The probabilistic PCA model can be run generatively to provide samples from the distribution.

Probabilistic PCA is a simple example of the linear-Gaussian framework, in which both latent and observed variables are Gaussian. We can formulate probabilistic PCA by first introducing an explicit latent variable $\boldsymbol{z}$ corresponding to the principal-component subspace.

The goal as before is, given a dataset $\boldsymbol{X}$ learn a $M < D$ **continuous latent space** by maximizing the likelihood of the probabilistic model M given:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \int p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} \tag{8.21}$$

Specifically the **generative model** works as follows

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \tag{8.22}$$

with $\boldsymbol{\mu} \in \mathbb{R}^D$ and the **continuous latent variable** $\boldsymbol{z} \in \mathbb{R}^M$ with a Gaussian **prior**

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|0, 1) \tag{8.23}$$

The independent noise is also Gaussian

$$p(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{\epsilon}|\boldsymbol{0}, \sigma^2 \boldsymbol{I}) \tag{8.24}$$

The matrix $\boldsymbol{W} \in \mathbb{R}^{D \times M}$ transforms the latent variables into observed variables. The conditional distribution of the observed variable $\boldsymbol{x}$ conditioned on the latent variable $\boldsymbol{z}$ **is still** a Gaussian, given by

$$p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2 \boldsymbol{\mathcal{I}}) \tag{8.25}$$

Note that this factorizes with respect to the elements of $\boldsymbol{x}$, in other words this is an example of the naive Bayes model. As we shall see shortly, the columns of $\boldsymbol{W}$ span a linear subspace within the data space that corresponds to the principal subspace. The other parameter in this model is the scalar a σ^2 governing the variance of the conditional distribution.

We can view the probabilistic PCA model from a generative viewpoint in which a sample's value of the observed variable is obtained by first choosing a value for the latent variable and then sampling the observed value conditioned on this latent value. This framework is based on mapping from latent space to data space. The reverse mapping, from data space to latent space can be obtained using Bayes' theorem. To write down the likelihood function, we need an expression for the marginal distribution $p(\boldsymbol{x})$ of the observed variable. This is expressed, from the sum and product rules of probability in the form

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \int p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} = \mathcal{N}(\boldsymbol{x}|\,\mathbb{E}[\boldsymbol{z}], Cov[\boldsymbol{x}]) \tag{8.26}$$

which is still a Gaussian so

$$\begin{aligned}\mathbb{E}[\boldsymbol{x}] &= \mathbb{E}[\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}] \\ &= \boldsymbol{W}\,\mathbb{E}[\boldsymbol{z}] + \boldsymbol{\mu} + \mathbb{E}[\boldsymbol{\epsilon}] \\ &= \boldsymbol{\mu}\end{aligned} \tag{8.27}$$

and

$$\begin{aligned}Cov[\boldsymbol{x}] &= \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T] \\ &= \mathbb{E}[(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon} - \boldsymbol{\mu})(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon} - \boldsymbol{\mu})^T]\end{aligned}$$

$$= \mathbb{E}[\boldsymbol{W}\boldsymbol{z}\boldsymbol{z}^T\boldsymbol{W}^T + 2\boldsymbol{W}\,\mathbb{E}[\boldsymbol{z}\boldsymbol{\epsilon}^T] + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T]]$$

$$\boldsymbol{z}, \boldsymbol{\epsilon} \text{ are independent so } Cov[\boldsymbol{z}, \boldsymbol{\epsilon}] = \mathbb{E}[\boldsymbol{z}\boldsymbol{\epsilon}^T - \underbrace{\mathbb{E}[\boldsymbol{z}]}_{0}\underbrace{\mathbb{E}[\boldsymbol{\epsilon}]}_{0} = 0$$

$$= \boldsymbol{W}\underbrace{\mathbb{E}[\boldsymbol{z}\boldsymbol{z}^T]}_{\boldsymbol{I}}\boldsymbol{W}^T + \underbrace{2\boldsymbol{W}\,\mathbb{E}[\boldsymbol{z}\boldsymbol{\epsilon}^T]}_{0} + \underbrace{\mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T]}_{\sigma^2\boldsymbol{I}}$$

$$= \boldsymbol{W}\boldsymbol{W}^T + \sigma^2\boldsymbol{I} = \boldsymbol{C} \tag{8.28}$$

Therefore

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{C}) \tag{8.29}$$

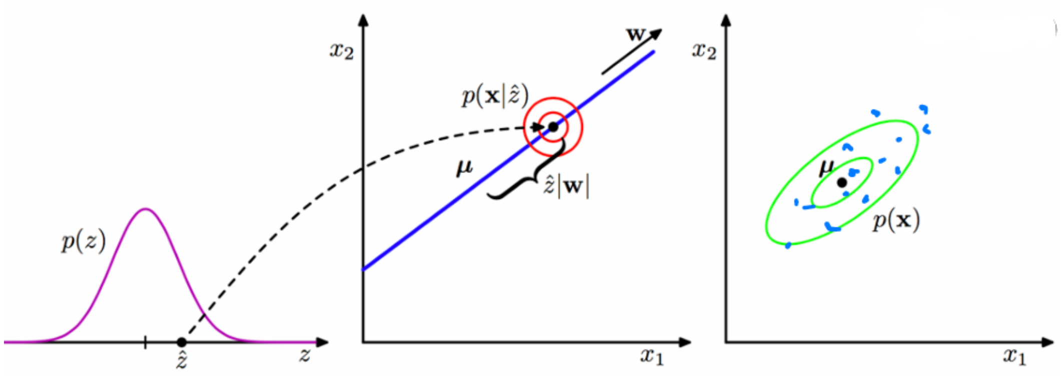


Figure 8.2: Illustration or probabilistic PCA

We next consider the determination of the model parameters using maximum likelihood. Gived a dataset i.i.d the corresponding log likelihood function is given by

$$\ln p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{W}, \sigma^2) = \sum_{n=1}^{N} \ln p(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{W}, \sigma^2) = \sum_{n=1}^{N} \ln \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{C})$$

$$= -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\boldsymbol{C}| - \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{\mu})^T\boldsymbol{C}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}) \tag{8.30}$$

Setting the derivative of the log likelihood with respect to $\boldsymbol{\mu}$ equal to zero gives the expected result

$$\sum_{n=1}^{N} \boldsymbol{C}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}) = 0 \Longrightarrow \boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n = \bar{\boldsymbol{x}} \tag{8.31}$$

Back-substituting we can then write the log likelihood function in the form

$$\ln p(\boldsymbol{X}|\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2) = -\frac{N}{2}\{D\ln(2\pi) + \ln|\boldsymbol{C}| + Tr(\boldsymbol{C}^{-1}\boldsymbol{S})\} \tag{8.32}$$

where $\boldsymbol{S}$ is the covariance matrix. Because the log likelihood is a quadratic function of $\boldsymbol{\mu}$, this solution represents the unique maximum and can be computed in a closed-form. All of the stationary points of the log likelihood function can be written as

$$\boldsymbol{W}_M = \boldsymbol{U}_{ML}(\boldsymbol{L}_M - \sigma^2\boldsymbol{I})^{1/2}\boldsymbol{R} \tag{8.33}$$

where U_M is a $D \times M$ matrix whose columns are given by any subset (of size M) of the eigenvectors of the data covariance matrix $\boldsymbol{S}$, the $M \times M$ diagonal matrix $\boldsymbol{L}_M$ has elements given by the corresponding eigenvalues λ_i, and $\boldsymbol{R}$ is an arbitrary $M \times M$ orthogonal matrix, which (WLOG) we can take to be $\boldsymbol{R} = \boldsymbol{I}$. Furthermore, it has been shown that the maximum of the likelihood function is obtained when the M eigenvectors are chosen to be those related to the M largest eigenvalues (all other solutions being saddle points).

Again, we shall assume that the eigenvectors have been arranged in order of decreasing values of the corresponding eigenvalues, so that the M principal eigenvectors are $\boldsymbol{u}_1, ..., \boldsymbol{u}_M$. In this case, the columns of $\boldsymbol{W}$ define the principal subspace of standard PCA. The corresponding maximum likelihood solution for σ^2 is then given by

$$\sigma^2 = \frac{1}{D-M} \sum_{j=M+1}^{D} \lambda_j \tag{8.34}$$

Because $\boldsymbol{R}$ is orthogonal, it can be interpreted as a rotation matrix in the $M \times M$ latent space. If we substitute the solution for $\boldsymbol{W}$ into the expression for $\boldsymbol{C}$, and make use of the orthogonality property $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$, we see that $\boldsymbol{C}$ is independent of $\boldsymbol{R}$. This simply says that the predictive density is unchanged by rotations in the latent space as discussed earlier. For the particular case of $\boldsymbol{R} = \boldsymbol{I}$, we see that the columns of $\boldsymbol{W}$ are the principal component eigenvectors scaled by the variance parameters $\lambda_i - \sigma^2$. The interpretation of these scaling factors is clear once we recognize that for a convolution of independent Gaussian distributions (in this case the latent space distribution and the noise model) the variances are additive. Thus the variance λ_i in the direction of an eigenvector $\boldsymbol{u}_i$ is composed of the sum of a contribution $\lambda_i - \sigma^2$ from the projection of the unit-variance latent space distribution into data space through the corresponding column of $\boldsymbol{W}$, plus an isotropic contribution of variance σ^2 which is added in all directions by the noise model.

8.2 Non-linear PCA

Previously in these notes we saw how the technique of kernel substitution allows us to take an algorithm expressed in terms of scalar products of the form $\boldsymbol{x}^T\boldsymbol{x}'$ and generalize that algorithm by replacing the scalar products with a nonlinear kernel. Here we apply this technique of kernel substitution to principal component analysis, thereby obtaining a nonlinear generalization called **kernel PCA**. The first step is to express conventional PCA in such a form that the data vectors $\{\boldsymbol{x}_n\}$ appear only in the form of the scalar products. Recalling that the principal components are defined by the eigenvectors $\boldsymbol{u}_i$ of the covariance matrix $\boldsymbol{S}$.

$$\boldsymbol{S} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^T \tag{8.35}$$

with the normalized eigenvectors such that $\boldsymbol{u}_i^T \boldsymbol{u}_i = 1$. Now consider a nonlinear transformation $\boldsymbol{\phi}(\boldsymbol{x})$ into an M-dimensional feature space (set of basis functions $\boldsymbol{\phi}(\boldsymbol{x}_n)$), so that each data point $\boldsymbol{x}_n$ is thereby projected onto a point $\boldsymbol{\phi}(\boldsymbol{x}_n)$. We can now perform standard PCA in the feature space, which implicitly defines a nonlinear principal component model in the original data space as can be seen from the figure below.

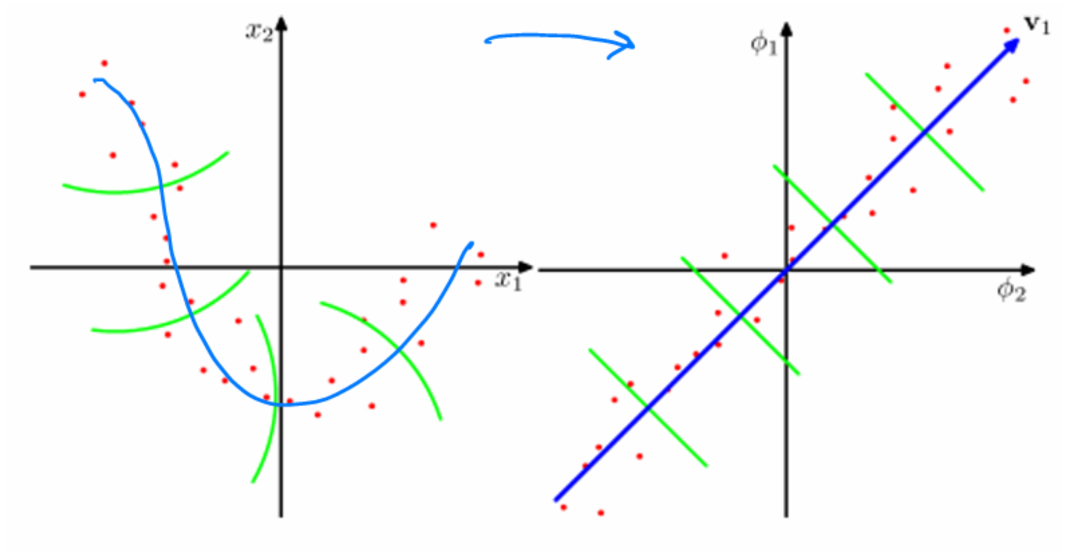


Figure 8.3: Schematic illustration fo kernel PCA

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n\boldsymbol{x}_n^T \quad \Longrightarrow \quad \boldsymbol{C} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T \tag{8.36}$$

Let $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j)$ be the kernel associated with the basis functions, let $\boldsymbol{u}_i$ be the principal components of $\boldsymbol{C} \in \mathbb{R}^{M\times M}$, let $z_i(\boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{u}_i$ be the projection onto the i-th component, with $\boldsymbol{a}_i$ the i-th eigenvector of $\boldsymbol{K} = \boldsymbol{\Phi}^T\boldsymbol{\Phi} \in \mathbb{R}^{N\times N}$, then

$$z_i(\boldsymbol{x}) = \sum_{n=1}^{N} a_{in}k(\boldsymbol{x}, \boldsymbol{x}_n) \tag{8.37}$$

The projection is purely in terms of the other data points via k! In the original D-dimensional $\boldsymbol{x}$ space there are D orthogonal eigenvectors and hence we can find at most D linear principal components. The dimensionality M of the feature space, however, can be much larger than D (even infinite), and thus we can find a number of nonlinear principal components that can exceed D. Note, however, that the number of non-zero eigenvalues cannot exceed the number N of data points, because (even if $M > N$) the covariance matrix in feature space has rank at most equal to N. This is reflected in the fact that kernel PCA involves the eigenvector expansion of the $N \times N$ matrix $\boldsymbol{K}$.

Kernel trick: use some defined kernel $k(\boldsymbol{x}, \boldsymbol{x}_j)$ without explicitly defining the basis functions. Note that this is very powerful because implicitly works with infinite dimensional features.

8.3 Autoencoders

Autoencoders are auto-associative neural networks used for dimensionality reduction, the idea is: given an input x to map it to a latent variable z, the neural network which does this is called **encoder**, there will be another NN called **decoder** that will allow me to reconstruct the original data from its latent representation. For autoencoders similarly to minimal reconstruction error analysis, we are going to optimize the weights of the NNs in order to minimize the reconstruction error.

$$\frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \frac{1}{N}\|\boldsymbol{x}_n - g_{\boldsymbol{W}'}(f_{\boldsymbol{W}}(\boldsymbol{x}_n))\|^2 \tag{8.38}$$

This function is non-linear so there is no closed form solution, we have to solve it via SGD. Recall the minimum error viewpoint of PCA

$$f(\boldsymbol{x}) = \boldsymbol{U}_M^T(\boldsymbol{x} - \bar{\boldsymbol{x}}), \quad g(\boldsymbol{x}) = \mathcal{U}_M \boldsymbol{z} + \bar{\boldsymbol{x}} \tag{8.39}$$

where $\boldsymbol{U}_M$ of f can be seen as the encoder parameter matrix, and $\mathcal{U}_M$ as the parameter matrix of the decoder. Bonus: We can use the decoder part $\tilde{\boldsymbol{x}} = g(\boldsymbol{z})$ as a generator of fake data (images), we could train the autoencoder to get $\tilde{\boldsymbol{x}} = g(\boldsymbol{z})$, sample a random $\boldsymbol{z}^*$ feed it into the generator to get fake image $\tilde{x} = g(\boldsymbol{z}^*)$

Chapter 9

Kernel Methods

Through these notes we have considered linear **parametric models** for regression and classification in which the from of the mapping $y(\boldsymbol{x}, \boldsymbol{w})$ is governed by a parameter $\boldsymbol{w}$. During the learning phase, a set of training data is used either to obtain a point estimate of the parameter vector or to determine a posterior distribution over this vector. The training data is then discarded, and predictions for new inputs are based purely on the learned parameter vector $\boldsymbol{w}$. This approach is also used in nonlinear parametric models such as neural networks. Many linear parametric models can be re-casted into an equivalent 'dual representation' in which the predictions are also based on linear combinations of a kernel function evaluated at the training data points, these models are called **non-parametric models**, this models use (subset of) training points for predictions and they are useful if $M >> N$. As we shall see, for models which are based on a fixed nonlinear feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$, the kernel function is given by the relation

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}) \tag{9.1}$$

From this definition, we see that the kernel is a symmetric function of its arguments so that $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$. Non parametric models are models with no explicitly defined parameters (but which implicitly still work with (finite) or infinite number of parameters), this allow them to directly work in possibly infinite dimensional function spaces. The concept of a kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well-known algorithms by making use of the kernel trick, also known as kernel substitution. The general idea is that, if we have an algorithm formulated in such a way that the input vector x enters only in the form of scalar products, then we can replace that scalar product with some other choice of kernel. For instance, the technique of kernel substitution can be applied to principal component analysis in order to develop a nonlinear variant of PCA. There are numerous forms of kernel functions in common use, as we shall shortly. Many have the property of being a function only of the difference between the arguments, so that $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x} - \boldsymbol{x}')$ which are known as stationary kernels because they are invariant to translations in input space. A further specialization involves homogeneous kernels, also known as radial basis functions, which depend only on the magnitude of the distance (typically Euclidean) between the arguments so that $k(\boldsymbol{x}, \boldsymbol{x}') = k(||\boldsymbol{x} - \boldsymbol{x}')||$.

9.1 Dual representation

Many linear models for regression and classification can be reformulated in terms of a dual representation in which the kernel function arises naturally. Here we consider a linear regression model whose parameters are determined by minimizing a regularized sum-of-squares error function given by

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\}^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{9.2}$$

where $\lambda \geq 0$. If we set the gradient of $J(\boldsymbol{w})$ with respect to $\boldsymbol{w}$ equal to zero, we see that the solution for $\boldsymbol{w}$ takes the form of a linear combination of the vectors $\boldsymbol{\phi}(\boldsymbol{x}_n)$ with coefficient that are function of $\boldsymbol{w}$, of the form

$$\begin{aligned}
\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} &= \sum_{n=1}^{N}\{\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\}\boldsymbol{\phi}(\boldsymbol{x}_n)^T + \lambda\boldsymbol{w}^T\boldsymbol{I} \\
&= \boldsymbol{w}^T\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T + \lambda\boldsymbol{I}\right) - \sum_{n=1}^{N}t_n\boldsymbol{\phi}(\boldsymbol{x}_n)^T \\
&\text{setting this to 0} \\
&\boldsymbol{w}^T\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T + \lambda\boldsymbol{I}\right) = \sum_{n=1}^{N}t_n\boldsymbol{\phi}(\boldsymbol{x}_n)^T \\
&\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \sum_{n=1}^{N}t_n\boldsymbol{\phi}(\boldsymbol{x}_n) \\
&\boldsymbol{w} = \left(\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T + \lambda\boldsymbol{I}\right)^{-1}\sum_{n=1}^{N}t_n\boldsymbol{\phi}(\boldsymbol{x}_n) \\
&\boldsymbol{w} = \underbrace{(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}}_{M\times M}\boldsymbol{\Phi}^T\boldsymbol{t}
\end{aligned} \tag{9.3}$$

Using the matrix inversion lemma, it allows us to alternatively obtain $\boldsymbol{w}$ via:

$$\boldsymbol{w} = \boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{t} \tag{9.4}$$

let's call $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$ the Gramm matrix, obtained via $\boldsymbol{K}_{ij} = \boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j)$. In this way the minimization w.r.t $\boldsymbol{w}$ becomes:

$$\boldsymbol{w} = \boldsymbol{\Phi}^T(\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{t} \tag{9.5}$$

Thus we see that the dual formulation allows the solution to the least-squares problem to be expressed entirely in terms of the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$. This is known as a dual formulation because, by noting that the solution for $\boldsymbol{a}$ can be expressed as a linear combination of the elements of $\boldsymbol{\phi}(\boldsymbol{x})$, we recover the original formulation in terms of the parameter vector $\boldsymbol{w}$. Note that the prediction at $\boldsymbol{x}$ is given by a linear combination of the target value from the training set.

The primal variable is $\boldsymbol{w} = \boldsymbol{\Phi}^T \boldsymbol{a}$, the dual variable $\boldsymbol{a} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{t}$. The predictive mean of the two viewpoints are:

- primal viewpoint $y(\boldsymbol{x}', \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}')$
- sual viewpoint $y(\boldsymbol{x}', \boldsymbol{a}) = \sum_{n=1}^{N} a_n k(\boldsymbol{x}_n, \boldsymbol{x}')$

In the dual formulation, we determine the parameter vector a by inverting an $N \times N$ matrix, whereas in the original parameter space formulation we had to invert an $M \times M$ matrix in order to determine $\boldsymbol{w}$. Because N is typically much larger than M, the dual formulation does not seem to be particularly useful. However, the advantage of the dual formulation, as we shall see, is that it is expressed entirely in terms of the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$. We can therefore work directly in terms of kernels and avoid the explicit introduction of the feature vector $\boldsymbol{\phi}(\boldsymbol{x})$, which allows us implicitly to use feature spaces of high, even infinite, dimensionality.

9.2 Constructing Kernels

First we need to formulate our optimization problem in such a way that the input vectors $\boldsymbol{x}_n$ enter only in the from of scalar products, then we replace all instances $\boldsymbol{x}_n^T \boldsymbol{x}_m$ with a kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \boldsymbol{K}_{nm}$, the kernel k corresponds to a scalar product in some (possibly infinite dimensional) feature space.
Note that a kernel to be **valid** has to be **symmetric positive semi definite** for all possible choices of $\{\boldsymbol{x}_n\}_{n=1}^{N}$.

One approach to build a valid kernel is to choose a feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$ and then use this to tind the corresponding kernel.

Theorem 9.2.1

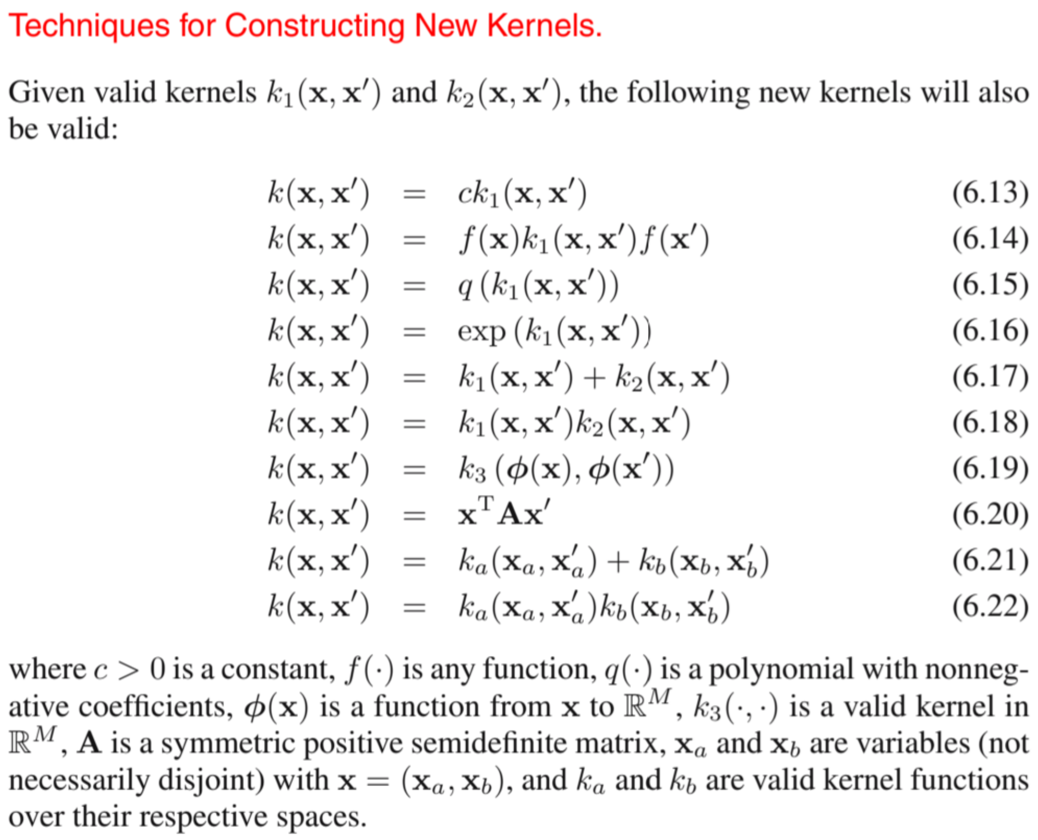
For every positive definite kernel there exists $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^M$ such that

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{9.6}$$

Depending on the kernel, M can be infinite, and in general it is difficult to retrieve the corresponding $\boldsymbol{\phi}(\boldsymbol{x})$ for a given kernel.
Some kernel examples:

- Generalized polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = (c + \boldsymbol{x}^T \boldsymbol{x}')^M$
- Gaussian kernel (infinite dim space!): $k(\boldsymbol{x}, \boldsymbol{x}') = exp(-\frac{1}{2\sigma^2}||\boldsymbol{x} - \boldsymbol{x}'||^2)$

One powerful technique for constructing new kernels is to build them out of simpler kernels as building blocks. This can be done using the following properties:



Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= ck_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.22)
\end{aligned}$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Figure 9.1: Enter Caption

Chapter 10

Support Vector Machines

We will now explore in detail the support vector machines (SVM), which are very useful for problems like classification, regression, and novelty detection. An important property of support vector machines is that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum. The SVM is a decision machine and so does not provide posterior probabilities. SVM leverages Kernel methods with sparse solutions, where prediction for new inputs depend only on kernel function evaluated at a **subset** of the training points.

- Primal viewpoint: $y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$
- Dual viewpoint: $y(\boldsymbol{x}, \boldsymbol{a}) = \sum_{n=1}^{N} a_n k(\boldsymbol{x}, \boldsymbol{x}_n)$

We define our classifer as:

$$y(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{x}_n + b \tag{10.1}$$

where the label is assigned as

$$\begin{cases} t_n = +1 \;\; if \;\; y(\boldsymbol{x}_n) \geq 0 \\ t_n = -1 \;\; if \;\; y(\boldsymbol{x}_n) < 0 \end{cases} \tag{10.2}$$

If $\boldsymbol{x}'$ lies on the decision boundary then $y(\boldsymbol{x}') = \boldsymbol{w}^T \boldsymbol{x}' + b = 0$. Recall that the distance from $\boldsymbol{x}$ to the decision boundary is

$$r = \frac{|y(\boldsymbol{x}_n)|}{||\boldsymbol{w}||} = \frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||} = \frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{||\boldsymbol{w}||} \tag{10.3}$$

We shall assume for the moment that the training data set is linearly separable in feature space, so that by definition there exists at least one choice of the parameters w and b such that our model function satisfies

- $y(\boldsymbol{x}_n) \geq 0 \;\; if \;\; t_n = +1$
- $y(\boldsymbol{x}_n) < 0 \;\; if \;\; t_n = -1$

so that for all $n = 1, ...N \quad t_n y(\boldsymbol{x}_n) \geq 0$.

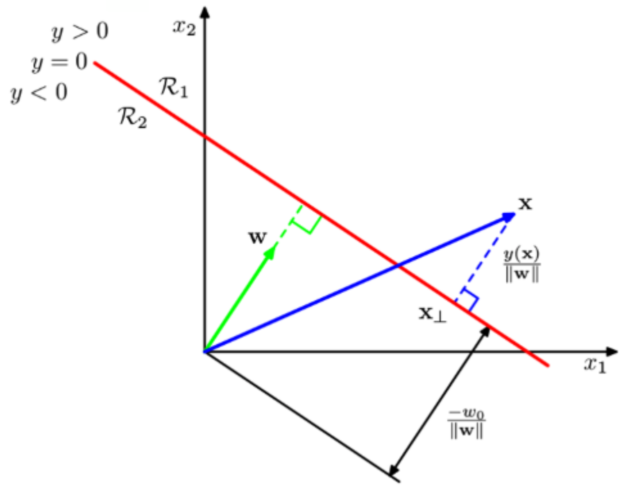


Figure 10.1: Projection on the decision boundary

10.1 Hard margin

There may of course exist many such solutions that separate the classes exactly. Previously in these notes, we described the perceptron algorithm that is guaranteed to find a solution in a finite number of steps. The solution that it finds, however, will be dependent on the (arbitrary) initial values chosen for w and b as well as on the order in which the data points are presented. If there are multiple solutions all of which classify the training data set exactly, then we should try to find the one that will give the smallest generalization error. The support vector machine approaches this problem through the concept of the margin, which is defined to be the smallest distance between the decision boundary and any of the samples. In support vector machines the decision boundary is chosen to be the one for which the margin is maximized. The margin is defined as the perpendicular distance from decision boundary to the closest point $\boldsymbol{x}_n$.

$$\underset{n}{min}\frac{t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)}{||\boldsymbol{w}||} = \underset{n}{min}\frac{t_n(k\boldsymbol{w}^T\boldsymbol{x}_n + kb)}{||\boldsymbol{w}||} \tag{10.4}$$

Note that if we rescale with k, then the distance from any point $\boldsymbol{x}_n$ to the decision surface is unchanged. We can use this freedom to set

$$t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) = 1 \tag{10.5}$$

for the point that is closest to the surface. In this case, all data points will satisfy the constraints

$$t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 \tag{10.6}$$

Given the size of the margin $\frac{1}{||\boldsymbol{w}||}$ and the set of constraints $t_n(\boldsymbol{w}^T\boldsymbol{x}_n + n) \geq 1$ we wish to maximize the margin $(||\boldsymbol{w}||^{-1}$, which is equivalent to minimizing $||\boldsymbol{w}||^2$, and so we have to solve the optimization problem

$$\underset{\boldsymbol{w},b}{argmin}\frac{1}{2}||\boldsymbol{w}||^2 \quad \textit{subject to N constraints} \;\; t_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 \tag{10.7}$$

where the factor of 1/2 in is included for later convenience. In the case of data points for which the equality holds, the constraints are said to be active, whereas for the remainder they are said to be inactive. By definition, there will always be at least one active constraint, because there will always be a closest point, and once the margin has been maximized there will be at least two active constraints. This is an example of a quadratic programming problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints. It appears that the bias parameter b has disappeared from the optimization. However, it is determined implicitly via the constraints, because these require that changes to $\boldsymbol{w}$ be compensated by changes to b.

In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geq 0$, with one multiplier a_n for each constraints, giving the **primal Lagrangian** function

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \underbrace{\frac{1}{2}||\boldsymbol{w}||^2}_{f(\boldsymbol{w})} - \sum_{n=1}^{N} \underbrace{a_n}_{a_n} \{\underbrace{t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b)}_{g(\boldsymbol{w})} - 1\} \tag{10.8}$$

with KKT conditions:

- primal feasibility $t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b) - 1 \geq 0$ for $n = 1, ..., N$
- dual feasibility $a_n \geq 0$ for $n = 1, ..., N$
- complimentary slackness $a_n(t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b) - 1) = 0$ for $n = 1, ..., N$

Note the minus sign in front of the Lagrange multiplier term, because we are minimizing with respect to $\boldsymbol{w}$ and b, and maximizing with respect to $\boldsymbol{a}$.

The dua lagrangian is obtained via the stationarity conditions

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0 \;\; \frac{\partial L}{\partial b} = 0$$

$$L(\boldsymbol{a}) = \underset{\boldsymbol{x},b}{min}\; L(\boldsymbol{x}, b, \boldsymbol{a}) \tag{10.9}$$

$$\textbf{\textit{solution}}: \quad \boldsymbol{a}^* = \underset{\boldsymbol{a}}{argmax}\; \tilde{L}(\boldsymbol{a}) \implies \boldsymbol{w}^*, b^* = \underset{\boldsymbol{w},b}{argmin}\; L(\boldsymbol{w}, v, \boldsymbol{a}^*) \tag{10.10}$$

Computing the derivatives

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}} &= \boldsymbol{w}^T - \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n^T = 0 \rightarrow \boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n \\ \frac{\partial L}{\partial b} &= -\sum_{n=1}^{N} a_n t_n = 0 \rightarrow \sum_{n=1}^{N} a_n t_n = 0 \end{aligned} \tag{10.11}$$

Eliminating $\boldsymbol{w}$ and b from L then gives the dual representation

$$\begin{aligned}
\tilde{L}(\boldsymbol{a}) &= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{n=1}^{N} a_n\{t_n(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+b)-1\} \\
&= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{n=1}^{N}(a_nt_n\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)+a_nt_nb-a_n) \\
&= \boldsymbol{w}^T\left(\frac{1}{2}\boldsymbol{w} - \sum_{n=1}^{N} a_nt_nx_n\right) - \sum_{n=1}^{N} a_nt_nb + \sum_{n=1}^{N} a_n \\
&= -\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - b\underbrace{\sum_{n=1}^{N} a_nb_n}_{0} + \sum_{n=1}^{N} a_n \\
&= \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_na_mt_nt_m\boldsymbol{x}_n^T\boldsymbol{x}_m \\
&\mathit{with}\ a_n \geq 0\ \mathit{for}\ n=1,...,N\ \mathit{and\ with}\ \sum_{n=1}^{N} a_nt_n = 0
\end{aligned} \tag{10.12}$$

We now apply the kernel trick on the dual representation fo the maximum margin, replacing $\boldsymbol{x}_n^T\boldsymbol{x}_m$ with $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_na_mt_nt_mk(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

The advantage is that we can now learn complex nonlinear decision boundaries, without having to specify an initial set of basis functions. The solution to a quadratic programming problem in M variables in general has computational complexity that is $O(M^3)$. In going to the dual formulation we have turned the original optimization problem, which involved minimizing over M variables, into the dual problem, which has N variables. For a fixed set of basis functions whose number M is smaller than the number N of data points, the move to the dual problem appears disadvantageous. However, it allows the model to be reformulated using kernels, and so the maximum margin classifier can be applied efficiently to feature spaces whose dimensionality exceeds the number of data points, including infinite feature spaces With the kernel trick our model function becomes.

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} a_nt_nk(\boldsymbol{x}_n, \boldsymbol{x}) + b \tag{10.13}$$

This formulation satisfies the KKt conditions, thus for every data point, either $a_n = 0$ or $t_ny(\boldsymbol{x}_n) = 1$. Any data point for which $a_n = 0$ will not appear in the sum in Equation 10.13 and hence plays no role in making predictions for new data points. The remaining data points are called support vectors, and because they satisfy $t_ny(\boldsymbol{x}_n) = 1$, they correspond to points that lie on the maximum margin hyperplanes in feature space. This property is central to the practical applicability of support vector machines. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors considered.

Having solved the quadratic programming problem and found a value for $\boldsymbol{a}$, we can then determine the value of the threshold parameter b by noting that any support vector $\boldsymbol{x}_n$ satisfies $t_n y(\boldsymbol{x}_n) = 1$. Using Equation 10.13 and S the set of indices of the support vectors, we obtain

$$\begin{aligned}
t_n \left(\sum_{m \in S} a_m t_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) + b \right) &= 1 \\
t_n^2 \left(\sum_{m \in S} a_m t_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) + b \right) &= t_n \\
\sum_{m \in S} a_m t_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) + b &= t_n \\
b &= t_n - \sum_{m \in S} a_m t_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) \qquad (10.14)
\end{aligned}$$

In practice it is more stable to average over all support vectors, (this because depending on optimizer, a_n may not be perfect) $b = \frac{1}{N} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) \right)$

10.2 Soft margin

So far, we have assumed that the training data points are linearly separable in the feature space $\boldsymbol{\phi}(\boldsymbol{x})$. The resulting support vector machine will give exact separation of the training data in the original input space $\boldsymbol{x}$, although the corresponding decision boundary will be nonlinear. In practice, however, the class-conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization. We therefore need a way to modify the support vector machine so as to allow some of the training points to be misclassified, with a penalty proportional to the distance to the margin boundary. In order to do so we introduce the *slack variables* $\varepsilon_n \geq 0$ where $n = 1, ..., N$, with one slack variable for each training data point. These are defined by $\varepsilon_n = 0$ for data points that are on the correct side of the margin and $\varepsilon_n = |t_n - y(\boldsymbol{x}_n)|$ for the ones on the wrong side of the margin. With this formulation the soft margin constrain for correct classification becomes:

The general definition of the slack variable is:

$\xi_n = \max(0, 1 - t_n y(x_n))$

$$t_n y(\boldsymbol{x}_n) \geq 1 - \varepsilon_n \;\; with \;\; n = 1, ..., N \qquad (10.15)$$

Points for which $0 < \varepsilon_n \leq 1$ lie inside the margin, but on the correct side of the decision boundary, and those data points for which $\varepsilon_n > 1$ lie on the wrong side of the decision boundary and are misclassified. This is sometimes described as relaxing the hard margin constraint to give a soft margin and allows some of the training set data points to be misclassified. Note that while slack variables allow for overlapping class distributions, this framework is still sensitive to outliers because the penalty for misclassification increases linearly with ε.

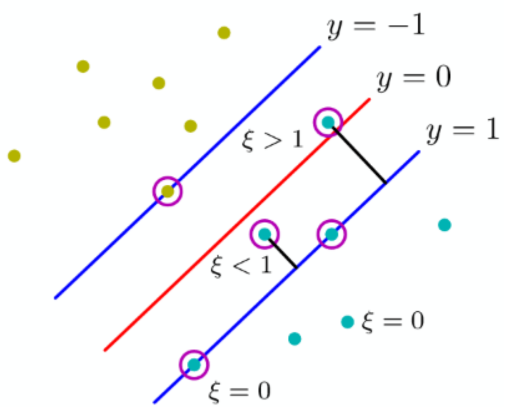


Figure 10.2: Illustration of the soft margin SVM.

Our goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. We therefore minimize

$$\underset{\boldsymbol{w},\, b,\, \varepsilon_n}{argmin}\ \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{n=1}^{N}\varepsilon_n$$

subject to constraints

$$t_n y(\boldsymbol{x}_n) \geq 1 - \varepsilon_n \quad \varepsilon_n \geq 0 \quad for \quad n = 1, ..., N$$

where the parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin. Because any point that is misclassified has $\varepsilon > 1$ it follows that $\sum_n \varepsilon_n$ is an upper bound on the number of misclassified points. The parameter C is therefore analogous to the inverse of a regularization coefficient because it controls the trade-off between minimizing training errors and controlling model complexity. In the limit $C \to \infty$, we will recover the hard margin SVM, while for $C \to 0$ we would obtain a possibly infinite margin, where every point becomes a support vector. Since we wish to minimize this new constrained function, we formulate the corresponding Lagrangian:

$$L(\boldsymbol{w}, b, \boldsymbol{\varepsilon}, \boldsymbol{a}, \boldsymbol{\mu}) = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{n=1}^{N}\varepsilon_n - \sum_{n=1}^{N} a_n\{t_n(\boldsymbol{w}^T\boldsymbol{w}_n + b) - 1 + \varepsilon_n)\} - \sum_{n=1}^{N}\mu_n\varepsilon_n \tag{10.16}$$

where the Lagrange multipliers are $a_n \geq 0$ and $\mu_n \geq 0$. The corresponding set of KKT conditions are given by:

$$\begin{aligned} a_n &\geq 0 \\ t_n(\boldsymbol{w}^T\boldsymbol{w}_n + b) - 1 + \varepsilon_n &\geq 0 \\ a_n(t_n(\boldsymbol{w}^T\boldsymbol{w}_n + b) - 1 + \varepsilon_n) &= 0 \\ \mu_n &\geq 0 \\ \varepsilon_n &\geq 0 \\ \mu_n\varepsilon_n &= 0 \end{aligned} \tag{10.17}$$

where $n = 1, .., N$ for a total of 6N KKT conditions. We now minimize L w.r.t primal variables $\boldsymbol{w}, b, \varepsilon_n$ and use the KKt conditions to eliminate $\boldsymbol{w}, b, \varepsilon_n$ from Lagrangian to obtain dual formulation.

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}} &= \boldsymbol{w}^T - \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n^T = 0 \quad \rightarrow \quad \boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{x}_n \\ \frac{\partial L}{\partial b} &= -\sum_{n=1}^{N} a_n t_n = 0 \quad \rightarrow \quad \sum_{n=1}^{N} a_n t_n = 0 \\ \frac{\partial L}{\partial \varepsilon_n} &= C - a_n - \mu_n = 0 \quad \rightarrow \quad a_n = C - \mu_n \end{aligned} \tag{10.18}$$

where the first two are the same as before but we have obtained a new constraint. Using these results to eliminate $\boldsymbol{w}, b$ and $\{\varepsilon_n\}$ from the Lagrangian, we obtain the dual Lagrangian in the form

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{x}_n^T \boldsymbol{x}_m \tag{10.19}$$

which is identical to the hard margin one, except that the constraints are somewhat different. To see what these constraints are, we note that $a_n \geq 0$ is required because these are Lagrange multipliers. Furthermore, the new constraint together with $\mu_n \geq 0$ implies $a_n \leq C$. We therefore have to minimize dual Lagrangian with respect to the dual variables $\{a_n\}$ subject to

$$\begin{aligned} &0 \leq a_n \leq C \quad (box - constraints) \\ &\sum_{n=1}^{N} a_n t_n = 0 \end{aligned} \tag{10.20}$$

As before, we can make use of the kernel trick, thus obtaining

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) \tag{10.21}$$

and for prediction

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} a_n t_n k(\boldsymbol{x}_n, \boldsymbol{x}) + b \tag{10.22}$$

We can now interpret the resulting solution, As before, a subset of the data points may have $a_n = 0$, in which case they do not contribute to the predictive model. The remaining data points constitute the support vectors. These have $a_n > 0$ and hence from one of the previous constraints they must satisfy

$$t_n y(\boldsymbol{x}_n) = 1 - \varepsilon_n \tag{10.23}$$

If $a_n < C$, then $\mu_n > 0$ so $\varepsilon_n = 0$ and hence such points lie on the margin. Points with $a_n = C$ can lie inside the margin and can either be correctly classified if $\varepsilon_n \leq 1$ or misclassified if $\varepsilon_n > 1$.

Chapter 11

Gaussian Processes

When consider Bayesian Linear regression we discussed a linear regression models of the form$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$ in which $\boldsymbol{w}$ is a vector of parameters and $\boldsymbol{\phi}(\boldsymbol{x})$ is a vector of fixed nonlinear basis functions that depend on the input vector $\boldsymbol{x}$. We showed that a prior distribution over $\boldsymbol{w}$ induced a corresponding prior distribution over functions $y(\boldsymbol{x}, \boldsymbol{w})$. Given a training data set, we then evaluated the posterior distribution over $\boldsymbol{w}$ and thereby obtained the corresponding posterior distribution over regression functions, which in turn (with the addition of noise) implies a predictive distribution $p(t|\boldsymbol{x})$ for new input vectors $\boldsymbol{x}$.

In the Gaussian process viewpoint, we dispense with the parametric model and instead define a prior probability distribution over functions directly. At first sight, it might seem difficult to work with a distribution over the uncountably infinite space of functions. However, as we shall see, for a finite training set we only need to consider the values of the function at the discrete set of input values $\boldsymbol{x}_n$ corresponding to the training set and test set data points, and so in practice we can work in a finite space. So, with GP insted of sampling (finite dimensional vectors) $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we sample (infinite dimensional) functions $f(\cdot) \sim GP(\boldsymbol{\mu}(\cdot), k(\cdot, \cdot))$ where $\boldsymbol{\mu}(\cdot)$ is the mean function and $k(\cdot, \cdot)$ the kernel.

Definition 11.0.1: Gaussian Process

A Gaussian Process is a collection of random variables indexed with time or space, any finite number of which is jointly Gaussian distributed.
It is thus a distribution for random functions

$$f(\cdot) \sim GP(m(\cdot), k(\cdot, \cdot)) \tag{11.1}$$

with

$$\begin{aligned} &\mathbb{E}[f(\boldsymbol{x})] = m(\boldsymbol{x}) \\ &cov(f(\boldsymbol{x}), f(\boldsymbol{x}')) = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))] = k(\boldsymbol{x}, \boldsymbol{x}') \end{aligned} \tag{11.2}$$

Taking any finite set $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with corresponding random variables $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$ then we can sample from a GP as

$$p\left(\begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x}_1) \\ \vdots \\ m(\boldsymbol{x}_N) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_N) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix}\right) \tag{11.3}$$

Consistency requirement: any finite subset of $\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}$ is Gaussian distributed. We can think of the function $f(\cdot)$ drawn from a GP as an extremely high-dimensional vector drawn from an extremely high-dimensional multivariate Gaussian distribution.

Example of Bayesian Linear Regression

We defined the Bayesian linear models

$$f(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{w} \tag{11.4}$$

with prior on $\boldsymbol{w}$

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \boldsymbol{\Sigma}_p) \tag{11.5}$$

then $f(\boldsymbol{x})$ is a Gaussian process

$$\begin{gathered} \mathbb{E}[f(\boldsymbol{x})] = \boldsymbol{\phi}(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}] = \boldsymbol{0} \\ cov(f(\boldsymbol{x}), f(\boldsymbol{x}')) = \mathbb{E}[f(\boldsymbol{x})f(\boldsymbol{x}')] = \boldsymbol{\phi}(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^T]\boldsymbol{\phi}(\boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\Sigma}_p \boldsymbol{\phi}(\boldsymbol{x}') \end{gathered} \tag{11.6}$$

Thus, $f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)$ for any N are Gaussian. This model provides us with a particular example of a Gaussian process. In general, a Gaussian process is defined as a probability distribution over functions $y(\boldsymbol{x})$ such that the set of values of $y(\boldsymbol{x})$ evaluated at an arbitrary set of points $\{\boldsymbol{x}_1, ...\boldsymbol{x}_N\}$ jointly have a Gaussian distribution.

Example of Drawing functions from GP's

Specifying a kernel determines the characteristics over functions drawn from the GP. Let's consider the kernel

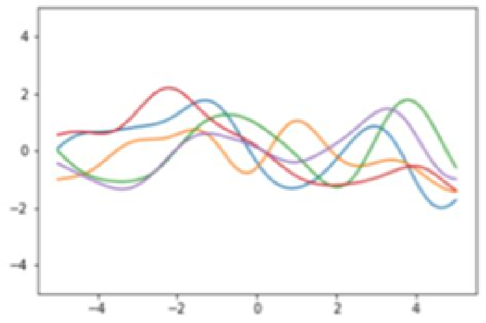
$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 exp\left(-\frac{1}{2\theta_1}\|\boldsymbol{x}_n - \boldsymbol{x}_m\|^2\right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m \tag{11.7}$$

and a finite grid of points $\{\boldsymbol{x}_1, ...\boldsymbol{x}_N\}$, we compute the gram matrix $\boldsymbol{K} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{T}}$ and use a reparametrization trick for sampling:

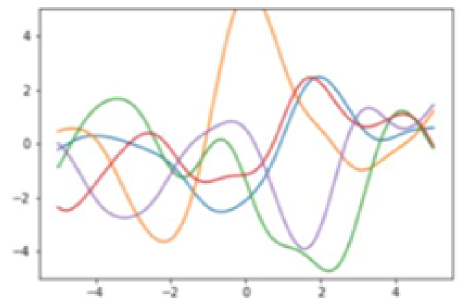
- Sample a random vector of size N: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_N)$
- Sample $\boldsymbol{f} = \begin{bmatrix} f(\boldsymbol{x}_1) \\ \vdots \\ f(\boldsymbol{x}_N) \end{bmatrix} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K})$ by computing $\boldsymbol{f} = \boldsymbol{L}\boldsymbol{z}$

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{1}{2\theta_1^2}||\mathbf{x}_n - \mathbf{x}_m||^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m$$

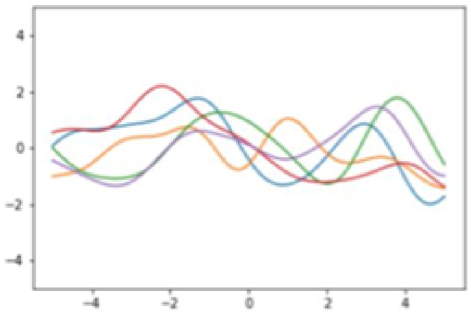
$\theta_0 = 1 \quad \theta_1 = 1 \quad \theta_2 = 0 \quad \theta_3 = 0$



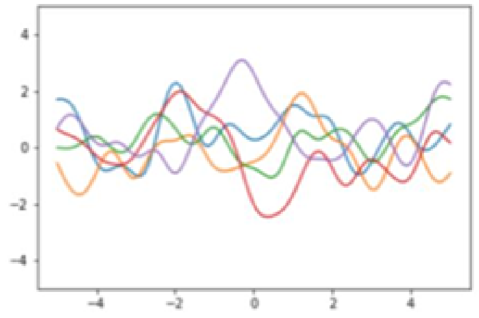
$\theta_0 = 4 \quad \theta_1 = 1 \quad \theta_2 = 0 \quad \theta_3 = 0$



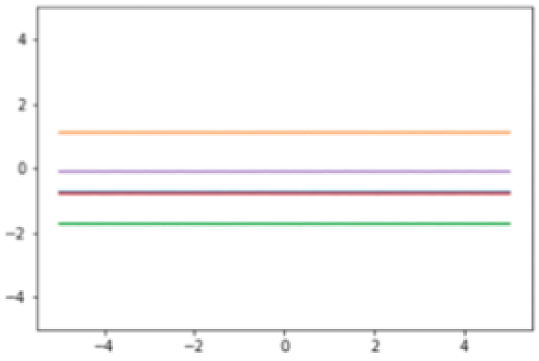
$\theta_0 = 1 \quad \theta_1 = 1 \quad \underline{\theta_2 = 0 \quad \theta_3 = 0}$



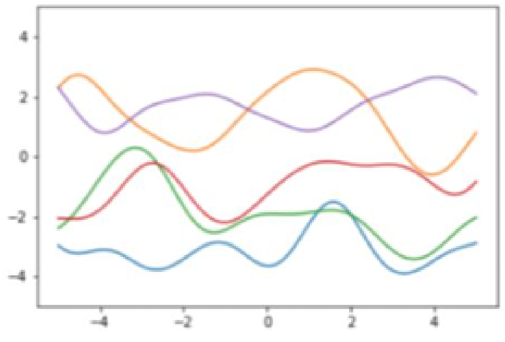
$\theta_0 = 1 \quad \theta_1 = 0.5 \quad \underline{\theta_2 = 0 \quad \theta_3 = 0}$



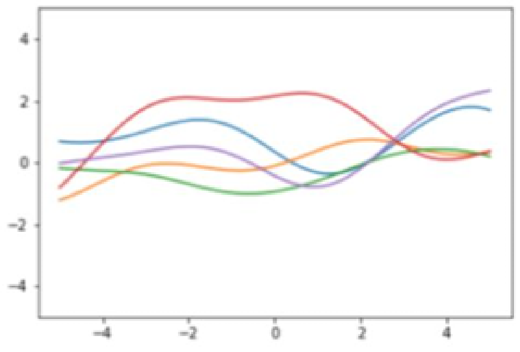
$\underline{\theta_0 = 0 \quad \theta_1 = 0} \quad \theta_2 = 5 \quad \underline{\theta_3 = 0}$



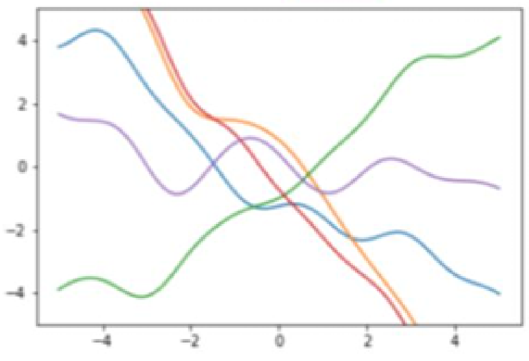
$\theta_0 = 1 \quad \theta_1 = 1 \quad \theta_2 = 5 \quad \underline{\theta_3 = 0}$



$\theta_0 = 1 \quad \theta_1 = 2 \quad \underline{\theta_2 = 0 \quad \theta_3 = 0}$



$\theta_0 = 1 \quad \theta_1 = 1 \quad \underline{\theta_2 = 0} \quad \theta_3 = 1$



11.1 Gaussian processes for regression

In order to apply Gaussian process models to the problem of regression, we need to take account of the noise on the observed target values, which are given by

$$f_n = f(\boldsymbol{x}_n) = y(\boldsymbol{x}_n) + \varepsilon_n \quad with \quad \varepsilon_n \sim \mathcal{N}(0, \beta^{-1}) \tag{11.8}$$

Assuming that we have a GP for $\boldsymbol{y}(\boldsymbol{x})$ any vector of observations is a Gaussian random variable from the definition of GP the distribution $p(\boldsymbol{y})$ is given by a Gaussian whose mean is zero and whose covariance is defined by the Gram matrix $\boldsymbol{K}$ so that

$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{K}) \tag{11.9}$$

The kernel function that determines $\boldsymbol{K}$ is typically chosen to express the property that, for points $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ that are similar, the corresponding values $y(\boldsymbol{x}_n)$ and $y(\boldsymbol{x}_m)$ will be more strongly correlated than for dissimilar points. Let's note that $\boldsymbol{y}$ and $\boldsymbol{\varepsilon}$ are two independent Gaussian distributed random variables, so their sum $\boldsymbol{f} = \boldsymbol{y} + \boldsymbol{\varepsilon}$ is also Gaussian distributed, thus

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \beta^{-1}\boldsymbol{I}) \tag{11.10}$$

The joint distribution of test points $\boldsymbol{f'}$ *at* $\boldsymbol{X'}$ and $\boldsymbol{f}$ (train points), according to our GP is given by

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f'} \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \beta^{-1}\boldsymbol{I} & \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X'}) \\ \boldsymbol{K}(\boldsymbol{X'}, \boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X'}, \boldsymbol{X'}) + \beta^{-1}\boldsymbol{I} \end{bmatrix}\right) \tag{11.11}$$

then

$$\begin{gathered} p(\boldsymbol{f'}|\boldsymbol{X'}, \boldsymbol{X}, \boldsymbol{f}) = \mathcal{N}(\boldsymbol{\mu'}, \boldsymbol{\Sigma'}) \\ with \\ \boldsymbol{\mu'} = \boldsymbol{K}(\boldsymbol{X'}, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \beta^{-1}\boldsymbol{I})^{-1}\boldsymbol{f} \\ \boldsymbol{\Sigma'} = \boldsymbol{K}(\boldsymbol{X'}, \boldsymbol{X'}) + \beta^{-1}\boldsymbol{I} - \boldsymbol{K}(\boldsymbol{X'}, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \beta^{-1}\boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X'}) \end{gathered} \tag{11.12}$$

11.2 How to choose kernel parameters?

The kernel parameters $\theta_0, \theta_1, ...$ are hyperparameters, the simplest approach is to take training observations, for which we know

$$\begin{gathered} \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{0}, C_\theta(\boldsymbol{X}, \boldsymbol{X})) = \frac{1}{(2\pi)^{N/2}|C_\theta|^{1/2}} exp\left(-\frac{1}{2}\boldsymbol{f}^T\boldsymbol{C}_\theta^{-1}\boldsymbol{f}\right) \\ with \quad \boldsymbol{C}_\theta(\boldsymbol{X}, \boldsymbol{X}) = \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \beta^{-1}\boldsymbol{I} \end{gathered} \tag{11.13}$$

is to make a maximum likelihood estimate by solving numerically for θ

$$\underset{\theta}{max} \ln p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\theta}) = \underset{\theta}{max} - \frac{1}{2}\ln|\boldsymbol{C}_\theta| - \frac{1}{2}\boldsymbol{f}^T\boldsymbol{C}_\theta^{-1}\boldsymbol{f} - \frac{N}{2}\ln 2\pi \tag{11.14}$$