4 Probabilistic regression and maximum likelihood

4.1 Probabilistic motion models

We are now ready for formalize supervised learning and to demonstrate this with a probabilistic motion model. In supervised learning we consider training data that consist of example inputs and corresponding labels, that is, pairs of values $(\mathbf{x}^{(i)}\mathbf{y}^{(i)})$, where the index $i = 1, ..., m$ labels each of m training example. As an example, let us consider the estimation of the motion model for the tribot. To automate the collection of data we can use the ultrasonic sensor to measure the distance to a wall while driving the tribot for different amount of time forward and backward. In Fig. 4.1A we show several measurements of the distance traveled.

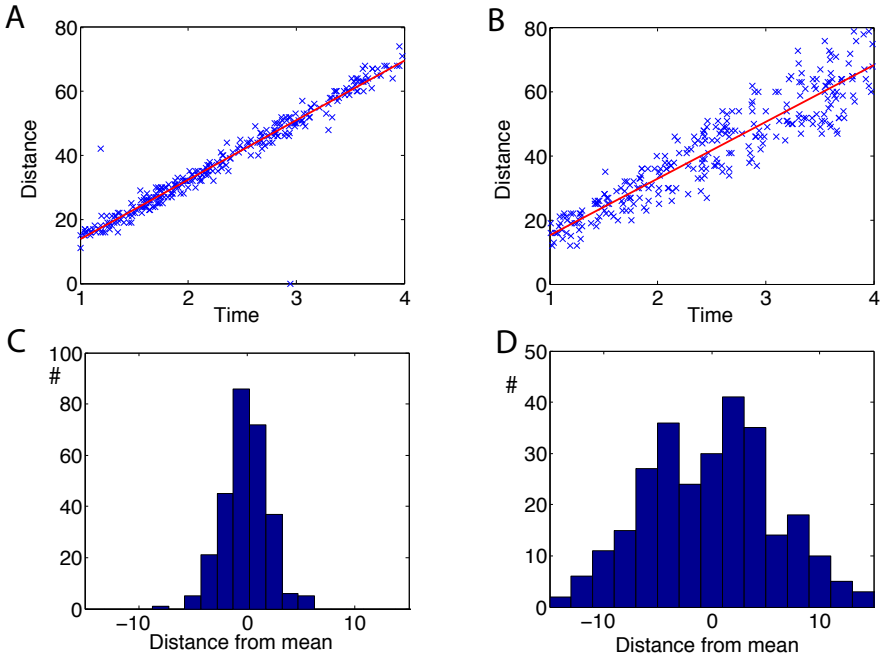


Fig. 4.1 (A) Measurements of distance travelled by the tribot when running the motor for different number of milliseconds with constant motor power. (B) Same as (A) with random motor power. (C,D) Corresponding histogram of differences between data and hypothesis.

The data clearly reveal some systematic relation between the time of running the

motor and the distance traveled, the general trend being that the traveled distance increases with increasing running time of the motors. While there seems to be some noise in the data, the outliers and the noise can not hide a linear trend for most of the data. This hypothesis can be quantified as a parameterized function,

$$h(x;\theta) = \theta_0 + \theta_1 x. \tag{4.1}$$

This notation means that the hypothesis h is a function of the quantity x, and the hypothesis includes all possible straight lines, where each line can have a different offset θ_0 (intercept with the y-axis), and slope θ_1.

We typically collect parameters in a **parameter vector** θ. We only considered one input variable x above, but we can easily generalize this to higher dimensional problems where more input **attributes** are given. For example, we could not only vary the time the motor is running, let us label this attribute now with x_1, but also push the robot forward by hand for a certain time, labeled with x_2. As the effects of these manipulations are independent on the results, we can independently add the effects of higher dimensions to our hypothesis. To compress our notations further we also introduce here the convention that we consider a constant input, $x_0 = 1$ as the first component of the input vector, so that the corresponding parameter encodes the offset of the function. The state vector can then be written as,

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}. \tag{4.2}$$

With this convention we can write the hypothesis as

$$h(\mathbf{x};\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2. \tag{4.3}$$

It is then even easy to write a linear hypothesis with n attributes as

$$h(\mathbf{x};\theta) = \theta_0 x_0 + ... + \theta_n x_n = \sum_i \theta_i x_i = \theta^T \mathbf{x}, \tag{4.4}$$

where the superscript T indicates the transpose of a vector.

Another factor that influences the distance traveled is the power setting of the motor. Of course, the distance traveled within a certain time does depend on the power and is not just an independent additive effect on the travelled distance. Results of the experiment for different power settings and different travel times are show in Fig. 4.2. Fig. 4.2A also includes a fit to Eq. 4.3. However, these data are better described by a bilinear hypothesis,

$$h(\mathbf{x};\theta) = \theta_0 x_0 + \theta_1 x_1 x_2. \tag{4.5}$$

The corresponding fit of the same data is shown in Figure 4.2B. How to perform these fits is discussed further below.

While we had to make a good guess for the functional form of the trend in the data, the actual parameters have so far not been specified. Thus, we made a **hypothesis** in the form of a parameterized function, $h(\mathbf{x};\theta)$, and the learning part boils down to determining appropriate values for the parameters from the sample data. After learning

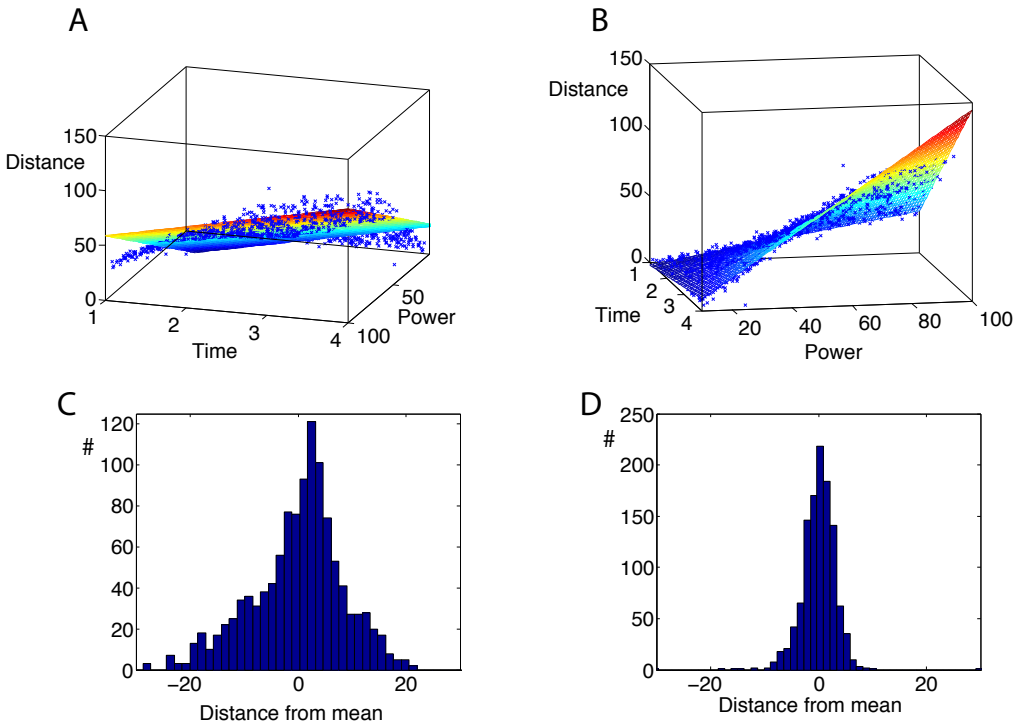


Fig. 4.2 (A) Measurements of distance travelled by the tribot when running the motor for different number of milliseconds and various power settings. Fit according to equation 4.3 (B) Same as (A) with fit according to equation 4.5. (C,D) Corresponding histogram of differences between data and hypothesis.

these parameters we can then use this function to predict specific reactions of a plant even for motor commands for which no training examples were given. The remaining question is how we find appropriate values for the parameters. However, before we do this we need to be more faithful to the data and acknowledge fluctuation around our initial hypothesis.

So far, we have only modelled the trend of the data, and we should investigate more the fluctuations around this trend. Of course, we expect several sources of noise such as the accuracy of the ultrasonic sensor and the tendency of the tribot to sometimes turn due to wheel slippage. Indeed, while gathering these data we have fixed the follower wheel to minimize turning when moving forward and backward. We also started new trials when the robot went too much off track. Thus, we already try to minimize error due to a careful setup of the experiment to gather the data.

However, what I did not tell you is that I run the experiment in Figure 4.1A with different powers of the motor between 40 and 60. Such hidden information can also contribute to uncertainties in the environment. Data for doing the same experiment as before but with a fixed motor power of 50 is shown in Figure 4.1B. These data vary less but are still noisy due other sources of uncertainty.

Figures 4.2C and D are plots of the histogram of the differences between the actual data and the hypothesis regression line. The histogram of plots of Figures 4.2C and D look a bit Gaussian, which according to the central limit theorem is a likely finding for additive and independent noise sources. In any case, we should revise our hypothesis by acknowledging the stochastic nature of the data and writing a down a specific functional form of a conditional density function for the quantity y given some input values $\mathbf{x}$. Similar to before, we also allow this probabilistic hypothesis to depend on some parameters,

$$p(y|\mathbf{x};\theta). \tag{4.6}$$

For our specific example of the tribot we assume here that the data follow our previous deterministic hypothesis $h(\mathbf{x};\theta)$ with **additive Gaussian noise**, or with other words, that the data in Figure 4.1B are Gaussian distributed with a mean $\mu = h(x)$ depends linearly on the value of x,

$$p(y|x;\theta,\sigma) = N(\mu = h(x),\sigma) \tag{4.7}$$

$$= \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y-\theta^T\mathbf{x})^2}{2\sigma^2}\right) \tag{4.8}$$

This functions specifies the probability of values for y, given an input x and the parameters $\theta = (\mu,\sigma)^T$.

Specifying a model with a density function is an important step in modern modelling and machine learning. In this type of thinking, we treat data from the outset as fundamentally stochastic, that is, data can be different even in situations that we deem identical. This randomness may come from an **irreducible indeterminacy**, that is, true randomness in the world that can not be penetrated by further knowledge, or this noise might represent **epistemological limitations** such as the lack of knowledge of hidden processes or limitations in observing states directly. The only important fact for us is that we have to live with these limitations. This acknowledgement together with the corresponding language of probability theory has helped to make large progress in the machine learning area.

4.2 Parameter estimates

4.2.1 Maximum A Posteriori (MAP)

Coming up with a (probabilistic) hypothesis in form of an appropriate parameterized (density) function is the hard problem in machine learning, and one that we have to discuss further. However, for now we assume that we have a parameterized hypothesis, and what is left is simply to use the data to estimate the parameters. More specifically, we want to know the probability of the parameters, given the data. That is

$$p(\theta|x,y) \tag{4.9}$$

However, as we have seen before, we usually make the hypothesis in the form of a probability function for the data given the parameters,

$$p(x,y|\theta) \tag{4.10}$$

Of course, these are related by Bayes' rule

$$p(\theta|x,y) = \frac{p(x,y|\theta)p(\theta)}{p(x,y)}, \tag{4.11}$$

The rule considers the likelihood of the data given specific parameters, $p(x,y|\theta)$. Each of these likelihoods should be weighted (multiplied) by our initial, or prior, knowledge of the possible parameter, $p(\theta)$. And of course, this product needs to be proper normalized by the probability of having the data, $p(x,y)$ to yield a proper probability.

Now. let us consider that we have a certain prior for the parameters, $p(\theta)$. The prior is in this situation sometimes called the **regularizer**, restricting possible values in a specific domain. And let us also consider a specific model $p(x,y|\theta)$. We are still missing an expression for the marginal probability of having the data, $p(x,y)$. However, this denominator does not depend on the parameters θ and the most probable values for the parameters can thus be calculated without this term. This **maximum a posteriori (MAP)** estimate is given by,

$$\theta^{\mathrm{MAP}} = arg\max_{\theta} p(x,y|\theta)p(\theta). \tag{4.12}$$

This is, in a Bayesian sense, the most likely value for the parameters.

Note that the MAP is a **point estimate**, a single answer of the most likely values of the parameters. This is often useful as a first guess and is commonly used to **make decisions** about which actions to take. However, it is possible that other sets of parameters values might have only a little smaller likelihood value, and should therefore also be considered. Thus, one limit of the estimation methods discussed here is that they do not take distribution of answers into account.

4.2.2 Maximum Likelihood Estimate (MLE)

Let us make the procedure of estimating the parameters of a model even more concrete by discussing a further special case of MAP and also by considering the specific example our model 4.7. It is common that we do not have any prior knowledge of the parameters in which case it is common to consider all possible values of the parameters as equally likely. The MAP estimate in this case is then given simply by maximizing the likelihood

$$\theta^{\mathrm{ML}} = arg\max_{\theta} p(x,y|\theta). \tag{4.13}$$

We will now demonstrate how to use data to make an estimate of the likelihood and thus can make a concrete estimate for the parameters of the model.

Let us now consider that the training data a provided as input-output pairs. Thus, the training set is given by $\{(x^{(i)},y^{(i)}); i = 1...m\}$. Note that the quantities $x^{(i)}$ and $y^{(i)}$ represent specific values. We are asking now which parameters would make the specific set of samples most likely, assuming that these are typical (unbiased) data from the assumed distribution. To do this we need first to write down the corresponding model for m random variable. We assume that the observations are independent so

that we can write the joint probability of several observations as the product of the individual probabilities,

$$p(y_1, y_2, ..., y_m|x_1, x_2, ..., x_m; \theta) = \Pi_i^m p(y_i|x_i; \theta). \tag{4.14}$$

Note that y_i are random variables at this point, not the observations). We now use our training examples as specific observations for each of these random variables, and introduce the **Likelihood function**

$$L(\theta) = \Pi_i^m p(\theta; x^{(i)}, y^{(i)}). \tag{4.15}$$

The p on the right hand side is now not a density function, but it is a regular function of the parameters θ for the given values $y^{(i)}$ and $x^{(i)}$. The form of this function is the same as the density function for the likelihood, but the notation should make it clear that the θ is now a regular variable and that $x^{(i)}$ and $y^{(i)}$ are specific values.

Instead of evaluating the Likelihood function which is a large product, it is common to use the logarithm of the likelihood function, so that we can use the sum over the training examples,

$$l(\theta) = \log L(\theta) = \sum_i^m \log(p(\theta; x^{(i)}, y^{(i)})). \tag{4.16}$$

Since the log function increases monotonically, the maximum of L is also the maximum of l. The maximum (log-)likelihood can thus be calculated from the examples as

$$\theta^{\mathrm{MLE}} = arg \max_\theta l(\theta). \tag{4.17}$$

We might be able to calculate this analytically or use one of the search algorithms to find a maximum from this function.

Let us apply this to the model of a Gaussian model with linear mean discussed above (eq. 4.7). The log-likelihood function for this example is

$$l(\theta) = \log \Pi_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T \mathbf{x^{(i)}})^2}{2\sigma^2}\right) \tag{4.18}$$

$$= \sum_{i=1}^m \left(\log \frac{1}{\sqrt{2\pi}\sigma} - \frac{(y^{(i)} - \theta^T \mathbf{x^{(i)}})^2}{2\sigma^2}\right) \tag{4.19}$$

$$= -\frac{m}{2} \log 2\pi\sigma - \sum_{i=1}^m \frac{(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2}{2\sigma^2}. \tag{4.20}$$

Thus, the log was chosen so that we can use the sum in the estimate instead of dealing with big numbers based on the product of the examples.

Let us now consider the special case in which we assume that the constant σ, the variance of the data, is the same for all x and thus has a fixed value given to us. We can thus concentrate on the estimation of the other parameters θ. Since the first term in

the expression 4.20, $-\frac{m}{2}\log 2\pi\sigma$, is independent of θ, maximizing the log-likelihood function is equivalent to minimizing a quadratic error term

$$E = \frac{1}{2}\left(y - h(\mathbf{x};\theta)\right)^2 \Longleftrightarrow p(y|\mathbf{x};\theta) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{\left(y - h(\mathbf{x};\theta)\right)^2}{2}) \tag{4.21}$$

This **error function** or **cost function** was a frequently used criteria called **Least Mean Square (LMS)** regression for parameters estimation when considering deterministic hypothesis. In terms of our probabilistic view, the LMS regression is equivalent to maximum likelihood estimate (MLE) for Gaussian data with constant variance. When the variance is a free parameter, then we need to minimize equation 4.20. instead.

We have discussed Gaussian distributed data in most of this section, but one can similarly find corresponding error functions for other distributions. For example, a **polynomial error function** correspond more generally to a density model of the form

$$E = \frac{1}{p}||y - h(\mathbf{x};\theta)||^p \Longleftrightarrow p(y|\mathbf{x};\theta) = \frac{1}{2\Gamma(1/p)}\exp(-||y - h(\mathbf{x};\theta)||^p). \tag{4.22}$$

Later we will specifically discuss and use the ϵ**-insensitive error function**, where errors less than a constant ϵ do not contribute to the error measure, only errors above this value,

$$E = ||y - h(\mathbf{x};\theta)||_\epsilon \Longleftrightarrow p(y|\mathbf{x};\theta) = \frac{p}{2(1-\epsilon)}\exp(-||y - h(\mathbf{x};\theta)||_\epsilon). \tag{4.23}$$

Since we already acknowledged that we do expect that data are noisy, it is somewhat logical to not count some deviations form the expectation as errors. It also turns out that this error function is much more robust than other error functions.

The final part that we need to discuss to fully implement probabilistic regression are methods to find extrema. For this it is also useful to refresh our memory of basic calculus.

4.3 Basic calculus and minimization

4.3.1 Differences and sums

We are often interested how a variable change with time. Let us consider the quantity $x(t)$ where we indicated that this quantity depends on time. The change of this variable from time t to time $t' = t + \Delta t$ is then

$$\Delta x = x(t + \Delta t) - x(t). \tag{4.24}$$

The quantity Δt is the finite difference in time. For a continuously changing quantity we could also think about the instantaneous change value, $\mathrm{d}x$, by considering an infinitesimally small time step. Formally,

$$\mathrm{d}x = \lim_{\Delta t \to 0} \Delta x = \lim_{\Delta t \to 0} (x(t + \Delta t) - x(t)). \tag{4.25}$$

The infinitesimally small time step is often written as $\mathrm{d}t$. Calculating with such infinitesimal quantities is covered in the mathematical discipline of calculus, but on the

computer we have always finite differences and we need to consider very small time steps to approximate continuous formulation. With discrete time steps, differential become differences and integrals become sums

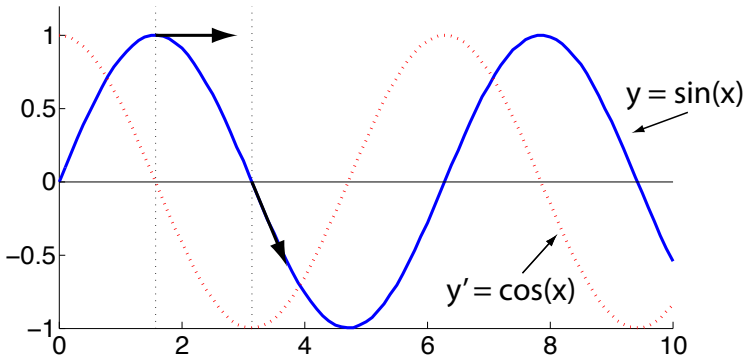
$$\mathrm{d}x \rightarrow \Delta x \tag{4.26}$$

$$\int \mathrm{d}x \rightarrow \Delta x \sum \tag{4.27}$$

Note the factor of Δx in front of the summation in the last equation. It is easy to forget this factor when replacing integrals with sums.

4.3.2 Derivatives

The derivative of a quantity y that depends on x is the slope of the function $y(x)$. This derivative can be defined as the limiting process equation 4.25 and is commonly written as $\frac{\mathrm{d}y}{\mathrm{d}x}$ or as y'.



It is useful to know some derivatives of basic functions.

$$y = e^x \rightarrow y' = e^x \tag{4.28}$$

$$y = sin(x) \rightarrow y' = cos(x) \tag{4.29}$$

$$y = x^n \rightarrow y' = nx^{n-1} \tag{4.30}$$

$$y = log(x) \rightarrow \frac{1}{x} \tag{4.31}$$

as well as the chain rule

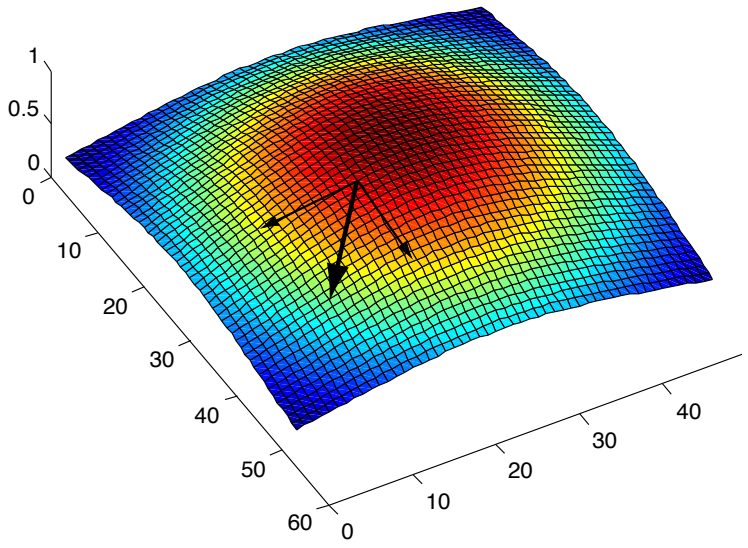
$$y = f(x) \rightarrow y' = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}f}\frac{\mathrm{d}f}{\mathrm{d}x}. \tag{4.32}$$

It is also useful to point out that the slope of a function is zero at an extremum (minimum or maximum). A minimum of a function $f(x)$ is hence characterized by $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$, and $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} > 0$. The second requirement, that the change of the slope is positive, specifies that this is a minimum rather than a maximum at which point this change in slope would be negative.

4.3.3 Partial derivative and gradients

A function that depends on more than one variable is a higher dimensional function. An example is the two-dimensional function $z(x, y)$. The slope of the function in the direction x (keeping y constant) is defined as $\frac{\partial z}{\partial x}$ and in the direction of y (keeping x constant) as $\frac{\partial z}{\partial y}$. The **gradient** is the vector that point in the direction of the maximal slope and has a length proportional to the slope,

$$\nabla z = \begin{pmatrix} \frac{\partial z}{\partial x} \\ \frac{\partial z}{\partial y} \end{pmatrix}. \tag{4.33}$$



Exercise

Calculate the gradient of a two dimensional Gaussian and find the minimum.

4.4 Minimization and gradient descent

We have discussed the important principle of maximum likelihood estimation to learn parameters from data so that the data are most likely under our hypothesis. This principle tells us how to use the training examples to come-up with some reasonable values for the parameters. To execute these principles we have to find the maximize of a (log)likelihood function.

Let us for now concentrate again on the maximum likelihood estimation of the parameters θ for the mean of the Gaussian data. The maximum likelihood estimate can the be found by minimizing the **mean square error** (MSE) function

$$E(\theta) = \frac{1}{2}(y - h(x;\theta))^2 \tag{4.34}$$

$$\approx \frac{1}{2m}\sum_i (y^{(i)} - h(x^{(i)};\theta))^2. \tag{4.35}$$

Note that the objective function is a function of the parameters. Also, note that while we wrote the general form of the objective function in line 4.34, we consider its maximum likelihood estimation from independent data in line 4.35. With this objective function, we reduced the learning problem to a search problem of finding the parameter values that minimize this objective function,

$$\theta = \arg\min_\theta E(\theta) \tag{4.36}$$

We will demonstrate practical solutions to this search problem with three important methods. The first method is an analytical one. You might remember from section 4.3.2 that finding a minimum of a function $f(x)$ is characterized by $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$, and $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} > 0$. Here we have a vector function since the cost function depends on several parameters. The derivative then becomes a gradient

$$\nabla_\theta E(\theta) = \begin{pmatrix} \frac{\partial E}{\partial \theta_0} \\ . \\ . \\ . \\ \frac{\partial E}{\partial \theta_n} \end{pmatrix}. \tag{4.37}$$

It is useful to collect the training data in a large matrix for the x values, and a vector for the y values,

$$X = \left(\mathbf{x}^{(1)} ... \ \mathbf{x}^{(m)}\right) \qquad Y = \left(y^{(1)} ... \ y^{(m)}\right). \tag{4.38}$$

Now let us use again the specific linear hypothesis of the data above. We can then write the cost function as

$$E(\theta) = \frac{1}{2m}(Y - X\theta)(Y - X\theta)^T. \tag{4.39}$$

Some straight forward calculation will then provide the parameters for which the gradient is zero,

$$\theta = (XX^T)^{-1}XY^T, \tag{4.40}$$

which is also known as the **normal equation**. We have still to make sure these parameters are the minimum and not a maximum value, but this can easily be done and is also obvious when plotting the result. This analytic methods is optimal in the requirement of computational time. However, it requires that we can analytically solve an equation system. This was easy in the linear case but fails in general for more complex functions. The next methods are more widely applicable.

The second method I want to mention here is random search, which is included mainly for illustrative purposes and to have a baseline to compare different algorithms. In this algorithm, new random values for the parameters are tried, and these new parameters replace the old ones if the new values result in a smaller error value (see

Table 4.1 Program randomSearch.m

```
%% Linear regression with random search
clear; clf; hold on;

load healthData;
E=1000000;
for trial=1:100
    thetaNew=[100*rand()+50, 3*rand()];
    Enew=0.5*sum((y-(thetaNew(1)+thetaNew(2)*x)).^2);
    if Enew<E; E=Enew; theta=thetaNew; end
end

plot(x,y,'*')
plot(120:260,theta(1)+theta(2)*(120:260))
xlabel('weight (pounds)')
ylabel('time of one-mile run (seconds)')
```

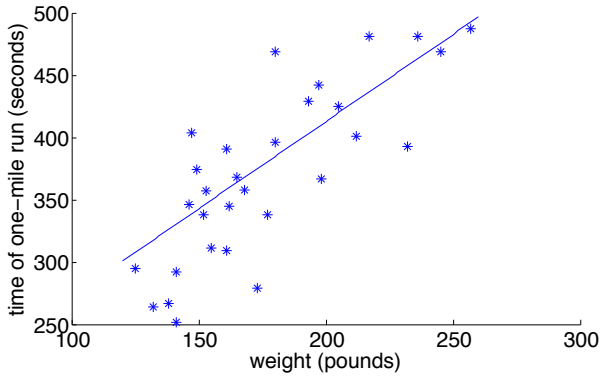


Fig. 4.3 Health data with linear least-mean-square (LMS) regression from random search.

Matlab code in Tab.4.3). The benefit of this method is that it can be applied to any function. Indeed, this methods is even guarantied to find a solution, but in most of our applications it usually takes too long to find a solution in reasonable time.

The final method discussed here for finding a minimum of a function $E(\theta)$ is **Gradient Descent**. This method will often be used in the following and it will thus be reviewed here in more detail.

Gradient Descent starts at some initial value for the parameters, θ, and improves the values iteratively by making changes to the parameters along the negative gradient of the cost function,

$$\theta \leftarrow \theta - \alpha \nabla_\theta E(\theta). \tag{4.41}$$

The constant α is called a **learning rate**. The principle idea behind this method is illustrated for a general cost function with one parameter in Fig.4.4.

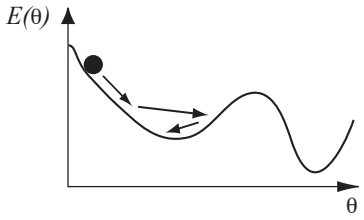


Fig. 4.4 Illustration of error minimization with a gradient descent method on a one-dimensional error surface $E(\theta)$.

The gradient is simple the slope (local derivative) for a function with one variable, but with functions in higher dimensions (more variables), the gradient is the local slope along the direction of the steepest accent, and since we are interested here in minimizing the cost function we make changes along the negative gradient. For large gradients, this method takes large steps, whereas the effective step-width becomes smaller near a minimum. Gradient descent works often well for local optimization, but it can get stuck in local minima. The corresponding update rule in case of the LMS error function,

$$E(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(y^{(i)} - h(x^{(i)}; \theta) \right)^2, \tag{4.42}$$

with a general hypothesis function $h(x^{(i)}; \theta)$ is given by

$$\theta_k \leftarrow \theta_k - \frac{\alpha}{m} \sum_{i=1}^{m} (y^{(i)} - h(x^{(i)}; \theta)) \frac{\partial h(\theta)}{\partial \theta_k}. \tag{4.43}$$

For a linear regression function, the update rule for the two parameters θ_0 and θ_1 are therefore:

$$h(x^{(i)}; \theta) = \theta_0 + \theta_1 x^{(i)} \tag{4.44}$$

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{\partial E(\theta)}{\partial \theta_0} \tag{4.45}$$

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{\partial E(\theta)}{\partial \theta_1} \tag{4.46}$$

$$\frac{\partial E(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \theta_0 - \theta_1 x^{(i)})(-1) \tag{4.47}$$

$$\frac{\partial E(\theta)}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \theta_0 - \theta_1 x^{(i)})(-x^{(i)}), \tag{4.48}$$

which lead to the final rule:

$$\theta_0 \leftarrow \theta_0 + \frac{\alpha}{m} \sum_{i=1}^{m} (y_i - \theta_0 - \theta_1 x_i) \tag{4.49}$$

$$\theta_1 \leftarrow \theta_1 + \frac{\alpha}{m}\sum_{i=1}^{m}(y_i - \theta_0 - \theta_1 x_i)x_i. \tag{4.50}$$

Note that the learning rate α has to be chosen small enough for the algorithm to converge. An example is show in Fig.4.5, where the dashed line shows the initial hypothesis, and the solid line the solution after 100 updates.

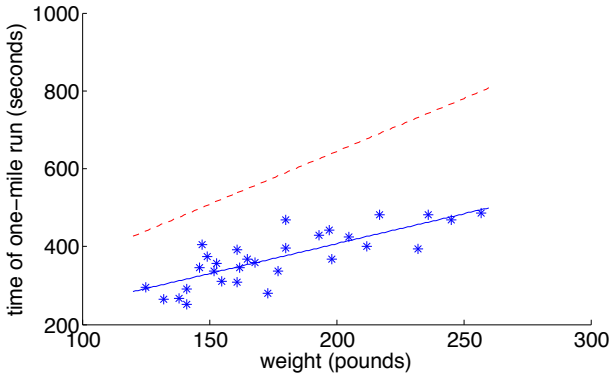


Fig. 4.5 Health data with linear least-mean-square (LMS) regression with gradient descent. The dashed line shows the initial hypothesis, and the solid line the solution after 100 updates.

In the algorithm above we calculate the average gradient over all examples before updating the parameters. This is called a **batch algorithm** or **synchronous update** since the whole batch of training data is used for each updating step and the update is only made after seeing all training data. This might be problematic in some applications as the training examples have to be stored somewhere and have to be recalled continuously. A much more applicable methods, also thought to be more biological realistic, is to use each training example when it comes in and disregards it right after. In this way we do not have to store all the data. Such algorithms are called **online algorithms** or **asynchronous update**. Specifically, in the example above, we calculate the change for each training examples and update the parameters for this training example before moving to the next example. While we might have to run through the short list of training examples in this specific example, it can still be considered online since we need only one training example at each training step and a list could be supplied to us externally. There are also variations of this algorithms depending on the order we use the training examples (e.g. random or sequential), although this should not be crucial for the examples discussed here.

The simple gradient descent as outlined above should highlight the general idea of gradient-based minimization. There are several problems with this simple approach when the function to be minimized is strongly non-linear. To illustrate this better, let us consider the Taylor expansion of a function

$$f(x+\epsilon) = f(x) + \epsilon^T \nabla_x f + O(\epsilon^2) \tag{4.51}$$

The simple gradient term is a good approximation if the higher order terms are small. However, if these are large than we make considerable errors and the method becomes

unstable. You can experience this when using a large learning rate. Of course, we could then consider higher order approximations like

$$f(x+\epsilon) = f(x) + \epsilon^T \nabla_x f + \frac{1}{2}\epsilon^T \mathbf{H}\epsilon + O(\epsilon^3), \tag{4.52}$$

where $\mathbf{H}$ is the Hessian matrix. Minimization based on this second-order term is called **Newton method**. While the Newton method converges after much faster than the simple gradient method, it is often difficult or time consuming to calculate or approximate the Hessian. There are further methods that are mainly used in practice such as the **Levenberg-Marquardt algorithm** or the **Natural Gradient**. We will not discuss this here in more detail, but you should consider these algorithms in practical applications.

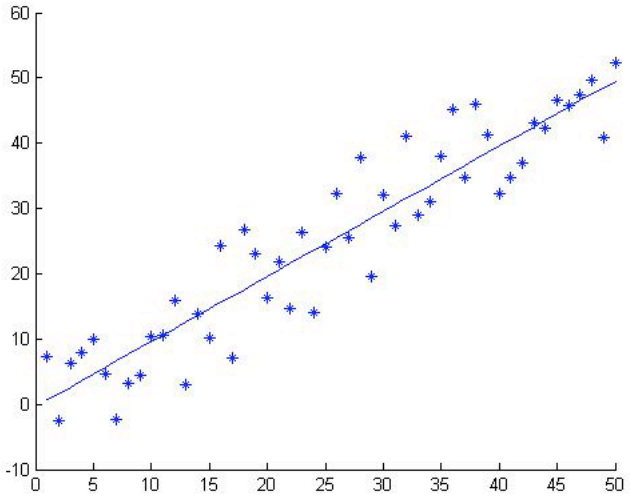
4.4.1 LinearRegressionExampleCode

```
%% Linear regression with gradient descent
clear all; clc; hold on;

load SampleRegressionData; m=50; alpha=0.001;
theta1=rand*100*((rand<0.5)*2-1);
theta2=rand*100*((rand<0.5)*2-1);

for trial=1:50000
    sum1 = sum(y - theta1 - theta2 *x);
    sum2 = sum((y - theta1 - theta2 *x).* x);
    theta1 = theta1 + (2*alpha/m) * sum1;
    theta2 = theta2 + (2*alpha/m) * sum2;
    sum1 = 0; sum2 = 0;
end

plot(x, y, '*');
plot(x, x*theta2+theta1);
hold off;
```



4.5 Classification

4.5.1 Logistic regression

We have grounded supervised learning in probabilistic function regression and maximum likelihood estimation. An important special case of supervised learning is **classification** where the possible y-values have only discrete values such as $y = \{1, 2, 3, \}$. We often call the different y-values simply **labels**.

An important example of classification is that of **binary classification** which are data that have only two possible labels such as $y = (0, 1)$. For example, let us consider the simplest case where the label does not depend on any feature value x. More formally, let us consider a random number which takes the value of 1 with probability ϕ and the value 0 with probability $1 - \phi$ (the probability of being either of the two choices has to be 1.). Thus, out probabilistic model of the data is a **Bernoulli distribution**

$$p(y) = \phi^y (1 - \phi)^{1-y} \tag{4.53}$$

Tossing a coin is a good example of a process that generates a Bernoulli random variable, but our data could be from a biased coin that might favour the head to the tail. We can again use maximum likelihood estimation to estimate the parameter ϕ from such trials. That is, let us consider m tosses in which h heads have been found. The log-likelihood of having h heads ($y = 1$) and $m - h$ tails ($y = 0$) is

$$\begin{aligned} l(\phi) &= \log(\phi^h (1 - \phi)^{m-h}) && (4.54) \\ &= h \log(\phi) + (m - h) \log(1 - \phi). && (4.55) \end{aligned}$$

To find the maximum with respect to ϕ we set the derivative of l to zero,

$$\frac{\mathrm{d}l}{\mathrm{d}\phi} = \frac{h}{\phi} - \frac{m - h}{1 - \phi} = 0 \tag{4.56}$$

$$\rightarrow \quad \phi = \frac{h}{m} \tag{4.57}$$

As you might have expected, the maximum likelihood estimate of the parameter ϕ is the fraction of heads in m trials.

Now let us discuss the case when the probability of observing a head or tail, the parameter ϕ, depends on an attribute x, as usual in a stochastic (noisy) way. For example, the data tend to fall into one class when the feature value is below a threshold θ, or into the other class when above. An example is illustrated in Fig.4.6 with 100 examples plotted with star symbols. The data suggest that it is far more likely that the class is $y = 0$ for small values of x and that the class is $y = 1$ for large values of x, and the probabilities are more similar in-between. The most difficult situation for such classification is around the threshold value since small changes in this value might trigger one versus the other class. It is then appropriate to make a hypothesis for the probability of $p(y = 1|x)$ which is small for x ($x << \theta$), around 0.5 for $x \approx \theta$, and approaching one for large x ($x >> \theta$). We further suggest that the transition between the low and high probability region is smooth. We qualify this hypothesis as parameterized density function known as a **logistic** (sigmoidal) function

$$\phi = \frac{1}{1 + \exp(-\theta^T \mathbf{x})}. \tag{4.58}$$

As before, we can then treat this density function as a function of the parameters θ for the given data values (likelihood function), and use maximum likelihood estimation to estimate values for the parameters so that the data are most likely.

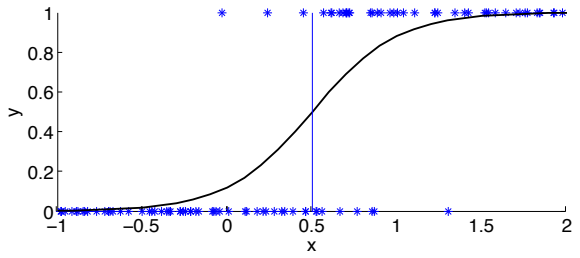


Fig. 4.6 Binary random numbers (stars) drawn from the density $p(y = 1) = \frac{1}{1+\exp(-\theta_1 x - \theta_0)}$ (solid line) with offset $\theta_0 = 2$ and slope $\theta_1 = 4$.

Specifically, let us again consider a pattern with m samples. A density function of m independent random variables of model 4.58 is

$$p(y_1, y_2, , y_m|x_1, x_2, , x_m; \theta) = \prod_i \phi(x_i; \theta)^{y_i} (1 - \phi(x_i; \theta))^{(1-y_i)} \tag{4.59}$$

and, using the data, the corresponding log-likelihood function

$$l(\theta) = \sum_i y^{(i)} \phi(\theta, x^{(i)}) + (1 - y^{(i)})(1 - \phi(\theta; x^{(i)})) \tag{4.60}$$

To find the maximum of the log-likelihood function we should again calculate the derivative of this function with respect to the parameters. We will not do this straight

forward calculation here, but it is useful to note that the derivative of the logistic function can be expressed again as a function of logistic functions

$$g(x) = \frac{1}{1+e^{-x}} \Rightarrow \frac{\partial g}{\partial x} = g(x)(1-g(x)) \tag{4.61}$$

Using this we find the following gradient for each data point,

$$\frac{\partial l}{\partial \phi} = (y - \phi(x,\theta))x. \tag{4.62}$$

Note that the learning procedure looks equivalent to the linear regression discussed above, but also note that the probabilistic model is quite different here.

How can we use the knowledge (estimate) of the density function to do classification? The obvious choice is to predict the class with the higher probability, given the input attribute. This **bayesian decision point**, x_d, or **dividing hyperplane** in higher dimensions, is given by

$$p(y=1|x_d) = p(y=0|x_d) = 0.5 \rightarrow x_d \theta^T \mathbf{x_d} = 0. \tag{4.63}$$

We have here considered binary classification with linear decision boundaries as logistic regression, and we can also generalize this method to problems with non-linear decision boundaries by considering hypothesis with different functional forms of the decision boundary. However, coming up with specific functions for boundaries is often difficult in practice, and we will discuss much more practical methods for binary classification later in this chapter.

4.6 Generative models and discriminant analysis

In the previous chapter we have introduced the idea that understanding the world should be based on a model of the world in a probabilistic sense. That is, building knowledge about the world really means estimating a large density function about the world. So far we have used such stochastic model mainly for a recognition model that take feature values $\mathbf{x}$ and make a prediction of an output y. Given the stochastic nature of the systems we want to model, the models where formulated as parameterized functions that represent the conditional probability $p(y|\mathbf{x};\theta)$. Of course, learning such models is a big task. Indeed, we had to assume that we know already the principle form of the distribution, and we used only simple model with low-dimensional feature vectors. The learning tasks of humans to be able to function in the real world seems much more daunting, and even training a robot in more restricted environment seems still beyond our current ability. While the previous models illustrate the principle problem in supervised learning, much of the rest of this course discusses more practical methods.

At the end of the last chapter we discussed a classification task where the aim of the model was to discriminate between classes based on the feature values. Such models are called **discriminative models** because they try to discriminate between possible outcomes based on the input values. Building a discriminative model directly from example data can be a daunting task as we have to learn how each item is distinguished from every other possible item. A different strategy, which seems much

more resembling human learning, is to learn first about the nature of specific classes and then use this knowledge when faced with a classification task. For example, we might first learn about chairs, and independently about tables, and when we are shown pictures with different furnitures we can draw on our knowledge to classify them. Thus, in this chapter we start discussing **generative models** of individual classes, given by $p(\mathbf{x}|y;\theta)$.

Generative models can be useful in their own right, and are also important to guide learning as discussed later, but for now we are mainly interested in using these models for classification. Thus, we need to ask how we can combine the knowledge about the different classes to do classification. Of course, the answer is provided by Bayes' theorem. In order to make a discriminative model from the generative models, we need to the **class priors know**, e.g. what the relative frequencies of the classes is, and can then calculate the probability that an item with features $\mathbf{x}$ belong to a class y as

$$p(y|\mathbf{x};\theta) = \frac{p(\mathbf{x}|y;\theta)p(y)}{p(x)}. \tag{4.64}$$

We can use this directly in the case of classification. The Bayesian decision criterion of predicting the class with the largest posterior probability is then:

$$\arg\max_y p(y|\mathbf{x};\theta) = \arg\max_y \frac{p(\mathbf{x}|y;\theta)p(y)}{p(x)} \tag{4.65}$$

$$= \arg\max_y p(\mathbf{x}|y;\theta)p(y), \tag{4.66}$$

where we have used the fact that the denominator does not depend on y and can hence be ignores. In the case of binary classification, this reads:

$$\arg\max_y p(y|\mathbf{x};\theta) = \arg\max_y (p(\mathbf{x}|y=0;\theta)p(y=0) + p(\mathbf{x}|y=1;\theta)p(y=1). \tag{4.67}$$

While using generative models for classification seem to be much more elaborate, but we will see later that generative models are attractive for machine learning.

The following example of using generative models in classification goes back to a paper by R. Fisher in 1936. In the following examples we consider that there are k classes, and we first assume that each class has members which are Gaussian distribution over the n feature value. An example for $n = 2$ is shown in Fig. 4.7A.

Each of the classes have a certain class prior

$$p(y = k) = \phi_k \tag{4.68}$$

, and each class itself is multivariate Gaussian distributed, generally with different means, μ_k and variances, Σ_k,

$$p(\mathbf{x}|y = k) = \frac{1}{\sqrt{2\pi}^n \sqrt{|\Sigma_0|}} e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T \Sigma_k^{-1}(\mathbf{x}-\mu_k)} \tag{4.69}$$

$$\tag{4.70}$$

Since we have supervised data with examples for each class, we can use maximum likelihood estimation to estimate the most likely values for the parameters

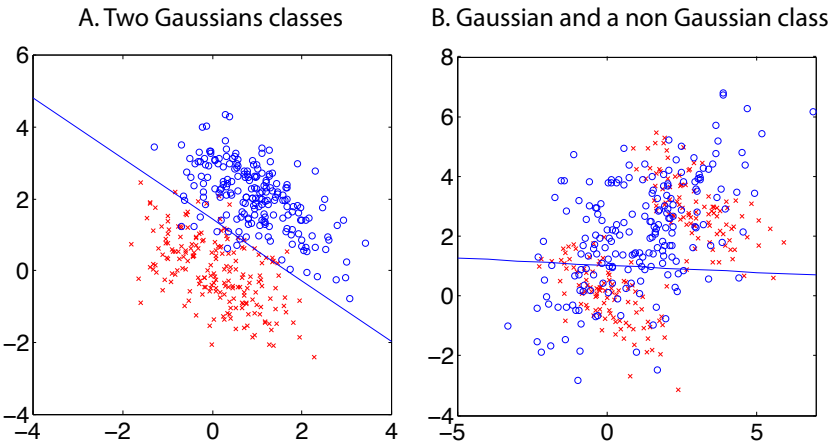


Fig. 4.7 Linear Discriminant analysis on a two class problem with different class distributions.

$\theta = (\phi_k, \mu_k, \Sigma_k)$. For the class priors, this is simply the relative frequency of the training data,

$$\phi_k = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = k) \tag{4.71}$$

where the function $\mathbb{1}(y^{(i)} = k) = 1$ if the i^{th} example belongs to class k, and $\mathbb{1}(y^{(i)} = k) = 0$ otherwise. The estimates of the means and variances within each class are given by

$$\mu_k = \frac{\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = k)\mathbf{x}^{(i)}}{\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = k)} \tag{4.72}$$

$$\Sigma_k = \frac{\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = k)(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T}{\sum_{i=1}^{m} \mathbb{1}(y^{(i)} = k)}. \tag{4.73}$$

With these estimates, we can calculate the optimal (in a Bayesian sense) decision rule, $G(x;\theta)$, as a function of $\mathbf{x}$ with parameters θ, namely

$$G(x) = arg\max_k p(y = k|\mathbf{x}) \tag{4.74}$$

$$= arg\max_k [p(\mathbf{x}|y = k;\theta)p(y = k)] \tag{4.75}$$

$$= arg\max_k [log(p(\mathbf{x}|y = k;\theta)p(y = k))] \tag{4.76}$$

$$= arg\max_k [-log(\sqrt{2\pi}^n \sqrt{|\Sigma_0|}) - \frac{1}{2}(\mathbf{x} - \mu_k)^T\Sigma_k^{-1}(\mathbf{x} - \mu_k) + log(\phi_k)] \tag{4.77}$$

$$= arg\max_k [-\frac{1}{2}\mathbf{x}^T\Sigma_k^{-1}\mathbf{x} - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k + \mathbf{x}^T\Sigma_k^{-1}\mu_k + log(\phi_k)], \tag{4.78}$$

since the first term in equation 4.77 does not depend on k and we can multiply out the other terms. With the maximum likelihood estimates of the parameters, we have all we need to make this decision.

In order to calculate the decision boundary between classes l and k, we make the common additional assumption that the covariance matrices of the classes are the same,

$$\Sigma_k =: \Sigma. \tag{4.79}$$

The decision boundary is then

$$log(\frac{\phi_k}{\phi_l}) - \frac{1}{2}(\mu_k - \mu_l)^T\Sigma^{-1}(\mu_k - \mu_l) - \mathbf{x}\Sigma^{-1}(\mu_k - \mu_l) = 0. \tag{4.80}$$

The first two terms do not depend on x and can be summarized as constant $\mathbf{a}$. We can also introduce the vector

$$\mathbf{w} = -\Sigma^{-1}(\mu_k - \mu_l). \tag{4.81}$$

With these simplifying notations is it easy to see that this decision boundary is a linear,

$$\mathbf{a} + \mathbf{w}\mathbf{x} = 0, \tag{4.82}$$

and this method with the Gaussian class distributions with equal variances is called **Linear Discriminant Analysis (LDA)**. The vector $\mathbf{w}$ is perpendicular to the decision surface. Examples are shown in Figure 4.7. If we do not make the assumption of equal variances of the classes, then we have a quadratic equation for the decision boundary, and the method is then called **Quadratic Discriminant Analysis (QDA)**. With the assumptions of LDA, we can calculate the contrastive model directly using Bayes rule,

$$p(y = k|\mathbf{x};\theta) = \frac{\frac{\phi_k}{|2\pi\Sigma|}e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)}}{\frac{\phi_k}{|2\pi\Sigma|}e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)} + \frac{\phi_l}{|2\pi\Sigma|}e^{-\frac{1}{2}(\mathbf{x}-\mu_l)^T\Sigma_l^{-1}(\mathbf{x}-\mu_l)}} \tag{4.83}$$

$$= \frac{1}{1 + \frac{\phi_l}{\phi_k}exp^{-\theta^T x}}, \tag{4.84}$$

where θ is an appropriate function of the parameters μ_k, μ_l, and Σ. Thus, the contrastive model is equivalent to logistic regression discussed in the previous chapter, although we use parametrisations and the two methods will therefore usual give different results on specific data sets.

So which method should be used? In LDA we made the assumption that each class is Gaussian distributed. If this is the case, then LDA is the best method we can use. Discriminant analysis is also popular since it often works well even when the classes are not strictly Gaussian. However, as can be seen in Figure 4.7B, it can also produce quite bad results if the data are multimodal distributed. Logistic regression is somewhat more general since it does not make the assumption that the class distributions are Gaussian. However, so far we have mainly looked at linear models and logistic regression would have also problems with the data shown in Figure 4.7B.

Finally, we should note that Fisher's original method was slightly more general than the examples discussed here since he did not assume Gaussian distributions. Instead considered within-class variances compared to between-class variances, something which resembles a signal-to-noise ratio. In **Fisher discriminant analysis (FDA)**, the separating hyperplane is defined as

$$\mathbf{w} = -(\Sigma_k + \Sigma_l)^{-1}(\mu_k - \mu_l). \tag{4.85}$$

which is the same as in LDA in the case of equal covariance matrices.

4.7 Non-linear regression and the bias-variance tradeoff

The role of supervised learning is to determine the parameters of a model that parameterizes our hypothesis about the relations between data.

change

We have only considered binary models where each Bernoulli variable is characterized by a single parameter ϕ. However, the density function can be much more complicated and introduce many more parameters. A major problem in practice is thus to have enough training examples with labels to restrict useful learning appropriately. This is one important reason for unsupervised learning as we have usually many unlabelled data that can be used to represent the problem appropriately. But we still need to understand the relations between free parameters and the number of training data.

We already discussed the bias-variance tradeoff in the first section. Finding the right function that describe nonlinear data is one of the most difficult tasks in modelling, and there is not a simple algorithm that can give us the answer. This is why more general learning machines, which we will discuss in the next section, are quite popular. To evaluate the generalization performance of a specific model it is useful to split the training data into a **training set**, which is used to estimate the parameters of the model, and a **validation set**, which is used to study the **generalization performance** on data that have not been used during training the model.

A important question is then how many data we should keep to validate versus train the model. If we use too many data for validation, than we might have too less data for accurate learning in the first place. On the other hand, if we have to few data for validation than this might not be very representative. In practice we are often using some **cross-validation** techniques to minimize the tradeoff. That is, we use the majority of the data for training, but we repeat the selection of the validation data several times to make sure that the validation was not just a result of outliers. The repeated division of the data into a training set and validation set can be done in different ways. For example, in **random subsampling** we just use random subsample for each set and repeat the procedure with other random samples. More common is k**-fold cross-validation**. In this technique we divide the data set into k-subsamples and use $k-1$ subsamples for training and one subsample for validation. In the next round we use another subsample for validating the training. A common choice for the number of subsamples is $k = 10$. By combining the results for the different runs we can often reduce the variance of our prediction while utilizing most data for learning.

We can sometimes help the learning process further. In many learning examples it turns out that some data are easy to learn while others are much harder. In some techniques called **boosting**, data which are hard to learn are over-sampled in the learning set so that the learning machine has more opportunities to learn these examples. A popular implementation of such an algorithm is **AdaBoost** (adaptive Boosting).

Before proceeding to general non-linear learning machines, I would like to outline a point that was recently made very eloquently by Doug Tweet in a course module that we shared last summer in a computational neuroscience course in Kingston, Canada. As discussed above, supervised learning is best phrased in terms of regression and that many applications are nonlinear in nature. It is common to make a nonlinear hypothesis in form of $y = h(\theta^T \mathbf{x})$, where θ is a parameter vector and h is a nonlinear hypothesis function. A common example of such a model is an artificial perceptrons

with a sigmoidal transfer function such as $h(x) = \tanh(\theta x)$. However, as nicely stressed by Doug, there is no reason to make the functions nonlinear in the parameters which then result in a non-linear optimization problem. Support vector machines that are reviewed next are a good example where the optimization problem is only quadratic in the parameters. The corresponding convex optimization has no local minima that plagued multilayer perceptrons. The different strategies might be summarized with the following optimization functions:

$$\text{Linear Perceptron } E \propto \left(y - \theta^T \mathbf{x}\right)^2 \tag{4.86}$$

$$\text{Nonlinear Perceptron } E \propto \left(y - h(\mathbf{x};\theta)\right)^2 \tag{4.87}$$

$$\text{Linear in Parameter (LIP) } E \propto \left(y - \theta^T \phi(\mathbf{x})\right)^2 \tag{4.88}$$

$$\text{Linear SVM } E \propto \alpha_i \alpha_j y_i y_j \mathbf{x}^T \mathbf{x} + \text{constraints} \tag{4.89}$$

$$\text{nonlinear SVM } E \propto \alpha_i \alpha_j y_i y_j \phi(\mathbf{x})^T \phi(\mathbf{x}) + \text{constraints} \tag{4.90}$$

The LIP (linear in parameters) model is more general than a linear model in that it considers functions of the form $y = \theta^T \phi(\mathbf{x})$ with some mapping function $\phi(\mathbf{x})$. In light of this review, the transformation $\phi(\mathbf{x})$ can be seen as re-coding a sensory signal into a more appropriate form with unsupervised learning methods as discussed above.

From previous version

So far, we have always discussed linear regression which assume that there is a linear relation between the variables. This is of course not the only possible relations between data. In Fig.4.8A we plotted the number of transistors of microprocessors against the year the processors were introduced. The plot also includes a linear fit, suggesting that we should assume some other functions.

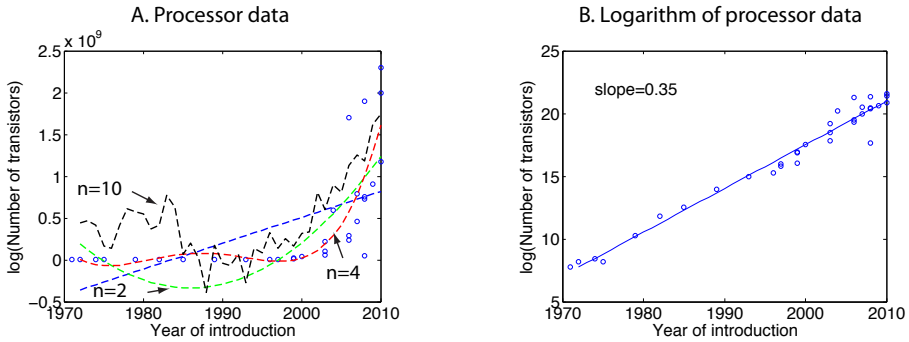


Fig. 4.8 Data from showing the number of transistors in microprocessors plotted the year that the processor was introduced. (A) Data and some linear and polynomial fits of the data. (B) Logarithm of the data and linear fit of these data.

Finding the right function is one of the most difficult tasks, and there is not a simple algorithm that can give us the answer. This task is therefore an important area where experience, a good understanding of the problem domain, and a good

understanding of scientific methods are required. This section does therefore try to give some recommendations when generalizing regression to the non-linear domain. These comments are important to understand for applying machine learning techniques in general.

It is often a good idea to visualize data an various ways since the human mind is often quite advanced in 'seeing' trends and patterns. Domain-knowledge thereby very valuable as specialists in the area from which the data are collected can give important advice of or they might have specific hypothesis that can be investigated. It is also good to know common mechanisms that might influence processes. For example, the rate of change in basic growth processes is often proportional to the size of the system itself. Such an situation lead to exponential growth. (Think about why this is the case). Such situations can be revealed by plotting the functions on a logarithmic scale or the logarithm of the function as shown in Fig.4.8B. A linear fit of the logarithmic values is also shown, confirming that the average growth of the number of transistors in microprocessors is exponential.

But how about more general functions. For example, we can consider a polynomial of order n, that can be written as

$$y = \theta_0 x^0 + \theta_1 x^1 + \theta_2 x^2 + ... + \theta_n x^n \tag{4.91}$$

We can again use LMS regression to determine the parameters from the data by minizing the LMS error function between the hypothesis and the data. This is implemented in Matlab as function `polyfit(x,y,n0)`. The LMS regression of the transistor data to a polynomials for orders $n = 2, 4, 10$ are shown in Fig.**??**A as dashed lines.

A major question when fitting data with fairly general non-linear functions is the order that we should consider. The polynomial of order $n = 4$ seem to somewhat fit the data. However, notice there are systematic deviations between the curve and the data points. For example, all the data between years 1980 and 1995 are below the fitted curve, while earlier data are all above the fitted curve. Such a systematic **bias** is typical when the order of the model is too low. However when we increase the order, then we usually get large fluctuations, or **variance**, in the curves. This fact is also called **overfitting** the data since we have typically too many parameters compared to the number of data points so that our model starts describing individual data points with their fluctuations that we earlier assumed to be due to some noise in the system. This difficulty to find the right balance between these two effects is also called the **bias-variance tradeoff**.

The bias-variance tradeoff is quite important in practical applications of machine learning because the complexity of the underlying problem is often not know. It then becomes quite important to study the performance of the learned solutions in some detail. For this it is useful to split the data set into **training set**, which is used to estimate the parameters of the model, and a **validation set** that can be used to study the performance on data that have not been used in the training procedure, that is, how the machine performs in **generalizes**. A schematic figure showing the bias-variance tradeoff is shown in Fig.4.9. The plot shows the error rate as evaluated by the training data (dashed line) and validation curve (solid line) when considering models with different complexities. When the model complexity is lower than the true complexity of the problem, then it is common to have a large error both in the training set and

in the evaluation due to some systematic bias. In the case when the complexity of the model is larger than the generative model of the data, then it is common to have a small error on the training data but a large error on the generalization data since the predictions are becoming too much focused on the individual examples. Thus varying the complexity of the data, and performing experiments such as training the system on different number of data sets or for different training parameters or iterations can reveal some of the problems of the models.

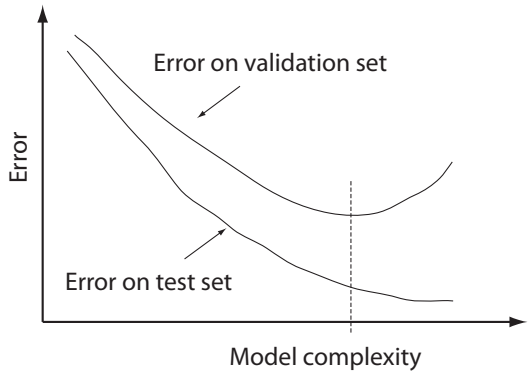


Fig. 4.9 Illustration of bias-variance tradeoff.

Using some of the data to validate the learning model is essential for many machine learning methods. A important question is then how many data we should keep to validate versus train the model. If we use too many data for validation, than we might have too less data for accurate learning in the first place. On the other end, if we have to few data for validation than this might not be very representative. In practice we are often using some **cross-validation** techniques to minimize the trade-off. That is, we use the majority of the data for training, but we repeat the selection of the validation data several times to make sure that the validation was not just a result of outliers. The repeated division of the data into a training set and validation set can be done in different ways. For example, in **random subsampling** we just use random subsample for each set and repeat the procedure with other random samples. More common is k**-fold cross-validation**. In this technique we divide the data set into k-subsamples samples and use $k-1$ subsamples for training and one subsample for validation. In the next round we use another subsample for validating the training. A common choice for the number of subsamples is $k = 10$. By combining the results for the different runs we can often reduce the variance of our prediction while utilizing most data for learning.

We can sometimes even help the learning process further. In many learning examples it turns out that some data are easy to learn while others are much harder. In some techniques called **boosting**, data which are hard to learn are over-sampled in the learning set so that the learning machine has more opportunities to learn these examples. A popular implementation of such an algorithm is **AdaBoost** (adaptive Boosting).