Machine learning notes

Francesco Boi

Contents

1 Preliminary definitions

1.1 Matrix properties and definitions

1.1.1 Trace of a matrix

The **trace** of a square matrix $\mathbf{A}$, denoted as $\mathbf{Tr}(\mathbf{A})$ is the sum of diagonal elements:

$$\mathbf{Tr}(\mathbf{A}) = \sum_{d=1}^{p} A_{dd} \tag{1.1}$$

It follows that $\mathrm{Tr}(\mathbf{I_d}) = p$. Also $\mathbf{Tr}(\mathbf{AB}) = \mathbf{Tr}(\mathbf{AB})$ and $\mathbf{Tr}(\mathbf{x}^T\mathbf{x}) = \mathbf{x}^T\mathbf{x}$ the latter being a scalar.

1.1.2 Exponential matrices

Definition 1.1. Exponential matrix The matrix exponential is a matrix function on square matrices analogous to the ordinary exponential function. Let X be an $N \times N$ real or complex matrix. Its exponential denoted by e^X is the $N \times N$ matrix given by the power series:

$$e^X = \sum_{k=0}^{\infty} \frac{1}{k!} X^k \tag{1.2}$$

where $X^0 = I_N$.

1.1.3 Similar matrices

Definition 1.2. Similar matrices In linear algebra, two $N \times N$ matrices A and B are called **similar** if there exists a $N \times N$ invertible matrix P such that:

$$B = P^{-1}AP \tag{1.3}$$

Example Similar matrices represent the same linear map under two (possibly) different bases, with P being the change of basis matrix. For example, a rotation around an axis not aligned with the coordinate axis might be difficult to express. If the axis of rotation is aligned with the positive z-axis, then the rotation is simply $S = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$. So in these new space such a rotation is $y' = Sx'$ where x' and y' are respectively the points before and after the rotation in the new more convenient space. In the original space the transformation is $y = Tx$. Then assuming P is the change of base matrix, T can be found as $y' = Py = PTx = Sx' \Rightarrow T = P^{-1}SP$.

1.1.4 Diagonalizable matrices

Definition 1.3. Diagonalizable matrices In linear algebra, a matrix is called **diagonalizable** if it is similar 1.1.3 to a diagonal matrix, i.e., if there exists an invertible matrix P and a diagonal matrix D such that $P^{-1}AP = D$.

1.2 Eigenvalue decomposition

Definition 1.4. Eigenvalues and eigenvector A (non-zero) vector $\vec{v}$ is an **eigenvector** of a square matrix $A \in \mathcal{R}^{N\times N}$ if it satisfies the linear equation:

$$A\vec{v} = \lambda\vec{v} \tag{1.4}$$

where λ is a scalar, termed **eigenvalue** associated to the eigenvector $\vec{v}$.

That is, the eigenvectors are the vectors that the linear transformation A merely elongates or shrinks, and the amount that they elongate/shrink by is the eigenvalue.

From this we get an equation for the eigenvalues.

Definition 1.5. characteristic polynomial and **characteristic equation**

$$p(\lambda) = \det(A - \lambda I) = 0 \tag{1.5}$$

$p(\lambda)$ is called **characteristic polynomial** and the equation **characteristic equation**.

$p(\lambda)$ is a N-th order polynomial equation in λ. The set of solution is called **spectrum of** A. $p(\lambda)$ can be refactored as:

$$p(\lambda) = (\lambda - \lambda_1)^{n_1}(\lambda - \lambda_2)^{n_2} \cdots (\lambda - \lambda_\lambda)^{n_\lambda} \tag{1.6}$$

where n_i is termed the **algebraic multiplicity of the eigenvalue** λ_i. The number of distinct eigenvalue is $0 \leq N_\lambda \leq N$.

For each eigenvalue we have a specific eigenvalue equation:

$$(A - \lambda_i I)\vec{v} = 0 \tag{1.7}$$

and there will be $0 \leq m_i \leq n_i$ linearly independent solutions to each eigenvalue equation. The term m_i is termed **geometric multiplicity of** λ_i, with $m_i \leq n_i$, the algebraic multiplicity.

1.3 Decomposition

Eigendecomposition or sometimes spectral decomposition is the factorization of a matrix into a canonical form, whereby the matrix is represented in terms of its eigenvalues and eigenvectors. Only diagonalizable matrices can be factorized in this way.

Let A be a $N \times N$ square matrix with N linearly independent eigenvectors q_i where $i = 1, \cdots, n$. Then A can be factorized as

$$A = Q\Lambda Q^{-1} \tag{1.8}$$

where Q is the square $n \times n$ matrix whose $i-th$ column is the eigenvector q_i of A, and Λ is the diagonal matrix whose diagonal elements are the eigenvalues. The q_i eigenvector are generally normalized but they need not be: the magnitudes of the eigenvectors in Q get cancelled by Q^{-1}:

$$A\vec{v} = \lambda\vec{v} \Rightarrow AQ = Q\Lambda \Rightarrow A = Q\Lambda Q^{-1} \tag{1.9}$$

The eigenvalue decomposition gives an easy way to compute power of matrices or exponential matrices:

$$\begin{aligned} A^{-1} &= Q\Lambda^{-1}Q^{-1} \\ A^{n} &= Q\Lambda^{n}Q^{-1} \\ e^{A} &= Q\left[\mathrm{diag}\left(e^{\lambda}_{ii}\right)\right]Q^{-1} \end{aligned} \tag{1.10}$$

1.4 Singular Value Decomposition

It is a factorization of a real or complex matrix that generalizes of the eigendecomposition of a square normal matrix to any $m \times n$ matrix. Given a $m \times n$ real or complex matrix M,

$$M = U\Sigma V^{*} \tag{1.11}$$

where U is a $m \times m$ real or complex unitary matrix, Σ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers termed **singular values** on the diagonal and V is a real or complex $n \times n$ unitary matrix. If M is real, U and $V* = V$ are orthonormal matrices, i.e., $U^{T}U = I$ and $V^{T}V = I$.

The number of non-zero singular values is equal to the rank of M. The column of U and V are called **left singular vectors** and **right singular vectors**. The SVD is not unique. However, it is always possible to choose the decomposition in order to have the singular values in descending order: $\sigma_1 \geq \sigma_2 \geq \sigma_3 \cdots$. In this way Σ is uniquely determined, but **not** U and V.

In the special case when M is an $m \times m$ real square matrix, the matrices U and $V*$ can be chosen to be real $m \times m$ matrices too. In that case, "unitary" is the same as "orthonormal". Then, interpreting each such matrix A as the linear transformation $x \rightarrow Ax$ of the space $\mathcal{R}^{m}$, the matrices U and $V*$ represent rotations or reflection of the space, while Σ represent the scaling of each coordinate x_i by the factor σ_i. Thus the SVD decomposition breaks down any invertible linear transformation of R^{m} into a composition of three geometrical transformations: a rotation or reflection (V^{*}), followed by a coordinate-by-coordinate scaling (Σ), followed by another rotation or reflection (U).

The singular values can be interpreted as the magnitude of the semiaxis of an ellipse in 2D. Singular values encode magnitude of the semiaxis, while singular vectors encode direction. the columns of U, U^{*}, V, and V^{*} are orthonormal bases. When the **M** is a normal matrix, U and V^{*} reduce to the unitary used to diagonalize **M**. However, when **M** is not normal but still

diagonalizable, its eigendecomposition and singular value decomposition are distinct.

1.4.1 Relationship with eigendecomposition

$$\begin{aligned} M^*M &= V\Sigma^*U^*U\Sigma V^* = V\Sigma^*\Sigma V^* \\ MM^* &= U\Sigma V^*V\Sigma^*U^* = U\Sigma\Sigma^*U^* \end{aligned} \tag{1.12}$$

The right-hand sides of these relations describe the eigenvalue decompositions of the left-hand sides. The columns of V (right-singular vectors) are eigenvectors of M^*M. The columns of U (left-singular vectors) are eigenvectors of MM^*. The non-zero elements of Σ (non-zero singular values) are the square roots of the non-zero eigenvalues of M^*M or MM^*.

Defining the **matrix of principal component variables**

$$Z = U\Sigma \tag{1.13}$$

we can find Z from the eigenvalue-decomposition of $M^*M = WDW^{-1}$.Σ is found as explained above. If W is chosen to be orthogonal, then we can set $U = W$, hence we have Z.

1.5 Expectation

Definition 1.6. Expectation Let X be a random variable with a finite number of outcomes $x_1, x_2, \cdots, x_k$ occurring respectively with probabilities $p_1, p_2, \cdots, p_k$. The expectation value is the summation of each outcome times its probability.

$$\mathbf{E}[X] = \sum_k x_k \cdot p_k \tag{1.14}$$

In case of an infinite number of outcomes the summation is replaced with the integral:

$$\mathbf{E}[X] = \int x \cdot p(x)\, dx \tag{1.15}$$

As explained here, when many random variables are involved, and there is no subscript in the E symbol, the expected value is taken with respect to their joint distribution:

$$E[h(X,Y)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y) f_{XY}(x,y)\, dx\, dy$$

When a subscript is present, in some cases it tells us on which variable we should condition. So

$$\mathrm{E}_{\mathrm{X}}[\mathrm{h}(\mathrm{X},\mathrm{Y})] = \mathrm{E}[\mathrm{h}(\mathrm{X},\mathrm{Y}) \mid \mathrm{X}] = \int_{-\infty}^{\infty} \mathrm{h}(\mathrm{x},\mathrm{y}) \mathrm{f}_{\mathrm{h}(\mathrm{X},\mathrm{Y})|\mathrm{X}}(\mathrm{h}(\mathrm{x},\mathrm{y}) \mid \mathrm{x})\, \mathrm{dh}$$

...But in other cases, it tells us which density to use for the "averaging"

$$\mathrm{E}_{\mathrm{X}}[\mathrm{h}(\mathrm{X},\mathrm{Y})] = \int_{-\infty}^{\infty} \mathrm{h}(\mathrm{x},\mathrm{y}) \mathrm{f}_{\mathrm{X}}(\mathrm{x})\, \mathrm{dx}$$

1.6 Variance

Definition 1.7. Variance The variance of a random variable X is the expected value of the squared deviation from the mean of X:

$$\mathrm{Var}\,(\mathrm{X}) = \mathrm{E}\left[(\mathrm{X} - \mu)^2\right] \tag{1.16}$$

Variance can be expressed in another way recalling $\mu = \mathrm{E}\,[\mathrm{X}]$ and using the linearity property:

$$\begin{aligned} \mathrm{Var}\,(\mathrm{X}) = \mathrm{E}\left[(\mathrm{X} - \mu)^2\right] &= \mathrm{E}\left[(\mathrm{X} - \mathrm{E}\,[\mathrm{X}])^2\right] = \\ &= \mathrm{E}\left[\mathrm{X}^2 - 2 \cdot \mathrm{X} \cdot \mathrm{E}\,[\mathrm{X}] + \mathrm{E}\,[\mathrm{X}]^2\right] \\ &= \mathrm{E}\left[\mathrm{X}^2\right] - 2 \cdot \mathrm{E}\,[\mathrm{X} \cdot \mu] + \mathrm{E}\left[\mu^2\right] \\ &= \mathrm{E}\left[\mathrm{X}^2\right] - 2 \cdot \mu \cdot \mathrm{E}\,[\mathrm{X}] + \mu^2 \\ &= \mathrm{E}\left[\mathrm{X}^2\right] - \mu^2 = \mathrm{E}\left[\mathrm{X}^2\right] - \mathrm{E}\,[\mathrm{X}]^2 \end{aligned} \tag{1.17}$$

1.7 Median

Definition 1.8. Median For any probability distribution on the real line $\mathbb{R}$ with cumulative distribution function F, regardless of whether it is any kind of continuous probability distribution, in particular an absolutely continuous distribution (which has a probability density function), or a discrete probability distribution, a median is by definition any real number m that satisfies

the inequalities:

$$P(x \leq m) \geq \frac{1}{2} \quad \text{and} \quad P(x \leq m) \geq \frac{1}{2} \tag{1.18}$$

$$\tag{1.19}$$

or equivalently the inequalities

$$\int_{-\infty}^{m} F(x)dx \geq \frac{1}{2} \quad \text{and} \quad \int_{m}^{\infty} F(x)dx \geq \frac{1}{2} \tag{1.20}$$

1.8 Median as the minimizer of L_1 norm

Assume that S is a finite set, with say k elements. Line them up in order, as $s_1 < s_2 < \cdots < s_k$.

If k is even there are (depending on the exact definition of median) many medians. $|x-s_i|$ is the *distance* between x and s_i, so we are trying to minimize the sum of the distances. For example, we have k people who live at various points on the x-axis. We want to find the point(s) x such that the sum of the travel distances of the k people to x is a minimum.

Imagine that the s_i are points on the x-axis. For clarity, take $k = 7$. Start from well to the left of all the s_i, and take a tiny step, say of length ϵ, to the right. Then you have gotten ϵ closer to every one of the s_i, so the sum of the distances has decreased by 7ϵ.

Keep taking tiny steps to the right, each time getting a decrease of 7ϵ. This continues until you hit s_1. If you now take a tiny step to the right, then your distance from s_1 increases by ϵ, and your distance from each of the remaining s_i decreases by ϵ. So there is a decrease of 6ϵ, and an increase of ϵ, for a net decrease of 5ϵ in the sum.

This continues until you hit s_2. Now, when you take a tiny step to the right, your distance from each of s_1 and s_2 increases by ϵ, and your distance from each of the five others decreases by ϵ, for a net decrease of 3ϵ.

This continues until you hit s_3. The next tiny step gives an increase of 3ϵ, and a decrease of 4ϵ, for a net decrease of ϵ.

This continues until you hit s_4. The next little step brings a total increase of 4ϵ, and a total decrease of 3ϵ, for an increase of ϵ. Things get even worse when you travel further to the right. So the minimum sum of distances is reached at s_4, the median.

The situation is quite similar if k is even, say $k = 6$. As you travel to the right, there is a net decrease at every step, until you hit s_3. When you

are between s_3 and s_4, a tiny step of ϵ increases your distance from each of s_1, s_2, and s_3 by ϵ. But it decreases your distance from each of the three others, for no net gain. Thus any x in the interval from s_3 to s_4, including the endpoints, minimizes the sum of the distances.

In the even case, Some people prefer to say that any point between the two "middle" points is a median. So the conclusion is that the points that minimize the sum are the medians. Other people prefer to define the median in the even case to be the average of the two "middle" points. Then the median does minimize the sum of the distances, but some other points also do.

In formulas consider two x_i's x_1 and x_2, with $x_2 > x_1$

- $$x_1 \leq a \leq x_2$$
$$\sum_{i=1}^{2} |x_i - a| = |x_1 - a| + |x_2 - a| = a - x_1 + x_2 - a = x_2 - x_1 \tag{1.21}$$

- $$a < x_1$$
$$\sum_{i=1}^{2} |x_i - a| = x_1 - a + x_2 - a = x_1 + x_2 - 2a \geq x_1 + x_2 - 2x_1 \tag{1.22}$$
$$= x_2 - x_1$$

- $$a \geq x_2$$
$$\sum_{i=1}^{2} |x_i - a| \quad = -x_1 + a - x_2 + a = -x_1 - x_2 + 2a \geq -x_1 - x_2 + 2x_2 =$$
$$= x_2 - x_1 \tag{1.23}$$

$\Longrightarrow$ for any two x_i's the sum of the absolute values of the deviations is minimum when $x_1 \leq a \leq x_2$ or $a \in [x_1, x_2]$.

When n is odd,

$$\sum_{i=1}^{n} |x_i - a| = |x_1 - a| + |x_2 - a| + \cdots + \left|x_{\frac{n-1}{2}} - a\right| + \left|x_{\frac{n+1}{2}} - a\right| + \\ + \left|x_{\frac{n+3}{2}} - a\right| + \cdots + |x_{n-1} - a| + |x_n - a| \tag{1.24}$$

consider the intervals $[x_1, x_n], [x_2, x_{n-1}], [x_3, x_{n-2}], \ldots, \left[x_{\frac{n-1}{2}}, x_{\frac{n+3}{2}}\right]$. If a is a member of all these intervals. i.e, $\left[x_{\frac{n-1}{2}}, x_{\frac{n+3}{2}}\right]$,

using the above theorem, we can say that all the terms in the sum except $\left|x_{\frac{n+1}{2}} - a\right|$ are minimized. So

$$\sum_{i=1}^{n} |x_i - a| = (x_n - x_1) + (x_{n-1} - x_2) + (x_{n-2} - x_3) + \cdots + \\ + \left(x_{\frac{n+3}{2}} - x_{\frac{n-1}{2}}\right) + \left|x_{\frac{n+1}{2}} - a\right| = \left|x_{\frac{n+1}{2}} - a\right| + \text{costant} \tag{1.25}$$

To minimize also the term $\left|x_{\frac{n+1}{2}} - a\right|$ it is clear we have to choose $a = x_{\frac{n+1}{2}}$ to get 0 but this is the definition of the median.

$\Longrightarrow$ When n is odd,the median minimizes the sum of absolute values of the deviations.

When n is even,

$$\sum_{i=1}^{n} |x_i - a| = |x_1 - a| + |x_2 - a| + \cdots + |x_{\frac{n}{2}} - a| + \\ + |x_{\frac{n}{2}+1} - a| + \cdots + |x_{n-1} - a| + |x_n - a| \tag{1.26}$$

If a is a member of all the intervals $[x_1, x_n], [x_2, x_{n-1}], [x_3, x_{n-2}], \ldots, \left[x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\right]$, i.e, $a \in \left[x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\right]$,

$$\sum_{i=1}^{n} |x_i - a| = (x_n - x_1) + (x_{n-1} - x_2) + (x_{n-2} - x_3) + \cdots + \\ + \left(x_{\frac{n}{2}+1} - x_{\frac{n}{2}} \right) \tag{1.27}$$

$\implies$ When n is even, any number in the interval $[x_{\frac{n}{2}}, x_{\frac{n}{2}+1}]$, i.e, including the median, minimizes the sum of absolute values of the deviations. For example consider the series:$2, 4, 5, 10$, median, $M = 4.5$.

$$\sum_{i=1}^{4} |x_i - M| = 2.5 + 0.5 + 0.5 + 5.5 = 9$$

If you take any other value in the interval $\left[x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\right] = [4, 5]$, say 4.1

$$\sum_{i=1}^{4} |x_i - 4.1| = 2.1 + 0.1 + 0.9 + 5.9 = 9$$

Taking for example 4 or 5 yields the same result:

$$\sum_{i=1}^{4} |x_i - 4| = 2 + 0 + 1 + 6 = 9$$

$$\sum_{i=1}^{4} |x_i - 5| = 3 + 1 + 0 + 5 = 9$$

This is because when summing the distance from a to the two middle points, you end up with the distance between them: $a - x_{\frac{n}{2}} + (x_{\frac{n}{2}+1} - a) = x_{\frac{n}{2}+1} - x_{\frac{n}{2}}$

For any value outside the interval $\left[x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\right] = [4, 5]$, say 5.2

$$\sum_{i=1}^{4} |x_i - 5.2| = 3.2 + 1.2 + 0.2 + 4.8 = 9.4$$

1.9 Correlation

In statistics, dependence or association is any statistical relationship, whether causal or not, between two random variables or bivariate data. In the broadest sense correlation is any statistical association, though it commonly refers to the degree to which a pair of variables are linearly related.

Definition 1.9. Correlation The most familiar measure of dependence between two quantities is the Pearson product-moment correlation coefficient, or "Pearson's correlation coefficient", commonly called simply "the correlation coefficient". It is obtained by dividing the covariance of the two variables by the product of their standard deviations. The population correlation coefficient $\rho_{\mathrm{X,Y}}$ between two random variables X and Y with expected values μ_{X} and μ_{Y} and standard deviations σ_{X} and σ_{Y} is defined as

$$\rho_{\mathrm{X,Y}} = \frac{\mathrm{Cov}(\mathrm{X},\mathrm{Y})}{\sigma_{\mathrm{X}}\sigma_{\mathrm{Y}}} = \frac{\mathbf{E}\left[(\mathrm{X}-\mu_{\mathrm{X}})(\mathrm{Y}-\mu_{\mathrm{Y}})\right]}{\sigma_{\mathrm{X}}\sigma_{\mathrm{Y}}} = \frac{\mathbf{E}\left[(\mathrm{X}-\mu_{\mathrm{X}})(\mathrm{Y}-\mu_{\mathrm{Y}})\right]}{\sigma_{\mathrm{X}}\sigma_{\mathrm{Y}}} = \\ = \frac{\sum_{\mathrm{i}}^{\mathrm{N}}(\mathrm{X_i}-\mu_{\mathrm{X}})(\mathrm{Y_i}-\mu_{\mathrm{Y}})}{\sqrt{\sum_{\mathrm{i}}^{\mathrm{N}}(\mathrm{X_i}-\mu_{\mathrm{X}})^2}\sqrt{\sum_{\mathrm{i}}^{\mathrm{N}}(\mathrm{Y_i}-\mu_{\mathrm{Y}})^2}} \tag{1.28}$$

An alternative formula purely in terms of moments is

$$\rho_{\mathrm{X,Y}} = \frac{\mathbf{E}\left[\mathrm{XY}\right]\mathbf{E}\left[\mathrm{X}\right]\mathbf{E}\left[\mathrm{Y}\right]}{\sqrt{\mathbf{E}\left[\mathrm{X}^2\right]-\mathbf{E}\left[\mathrm{X}\right]^2}\sqrt{\mathbf{E}\left[\mathrm{Y}^2\right]-\mathbf{E}\left[\mathrm{Y}\right]^2}} \tag{1.29}$$

The correlation coefficient is symmetric: $\mathrm{corr}(\mathrm{X},\mathrm{Y}) = \mathrm{corr}(\mathrm{Y},\mathrm{X})$. This is verified by the commutative property of multiplication. It is a corollary of the Cauchy–Schwarz inequality that the absolute value of the Pearson correlation coefficient is not bigger than 1. The correlation coefficient is +1 in the case of a perfect direct (increasing) linear relationship (correlation), –1 in the case of a perfect decreasing (inverse) linear relationship (anticorrelation), and some value in the open interval $(-1, 1)$ in all other cases, indicating the degree of linear dependence between the variables. As it approaches zero there is less of a relationship (closer to uncorrelated). The closer the coefficient is to either –1 or 1, the stronger the correlation between the variables. If the variables are independent, Pearson's correlation coefficient is 0, but the converse is not true because the correlation coefficient detects only linear dependencies between two variables.

For example suppose $Y = X^2$. Y is completely determined by X so that they are perfectly dependent but they are uncorrelated. In the special case where X and Y are jointly normal then "uncorrelatedness" is equivalent to independence.

So correlation might completely miss out non-linear relationships (1.1) and furthermore it has nothing to do with the slope.

1.10 Gaussian functions and Gaussian distributions

Definition 1.10. Gaussian function A Gaussian function is a mathematical function in the form:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{1.30}$$

The Gaussian function has three parameters a,b and c. The graph of a Gaussian function is the Bell curve. The parameter a is the height of the curve's peak, b is the position of the peak and c the *standard deviation* controls the width of the bell.

An important property of the Gaussian function is that the product of two Gaussian functions is still a Gaussian function:

$$\begin{aligned}
f(x)\cdot g(x) &= ae^{-\frac{(x-b)^2}{2c^2}} a_1 e^{-\frac{(x-b_1)^2}{2c_1^2}} = (a\cdot a_1)e^{-\frac{x^2-2bx+b^2+x^2-2b_1x+b_1^2}{2c^2c_1^2}} = \\
&= (a\cdot a_1)e^{-\frac{2x^2-2x(b+b_1)+b^2+b_1^2}{2c^2c_1^2}} = (a\cdot a_1)e^{-\frac{x^2-2x\frac{b+b_1}{2}+\frac{b^2+b_1^2}{2}}{2c^2c_1^2}} = \\
&= (a\cdot a_1)e^{-\frac{x^2-2x\frac{b+b_1}{2}+\frac{b^2+b_1^2}{2}}{2c^2c_1^2}} e^{\frac{\left(\frac{b+b_1}{2}\right)^2-\left(\frac{b+b_1}{2}\right)^2}{2c^2c_1^2}} = \\
&= (a\cdot a_1)e^{-\frac{x^2-2x\frac{b+b_1}{2}+\left(\frac{b+b_1}{2}\right)^2}{2c^2c_1^2}} e^{\frac{\left(\frac{b+b_1}{2}\right)^2-\frac{b^2+b_1^2}{2}}{2c^2c_1^2}} = \\
&= (a\cdot a_1)e^{-\frac{\left(x-\frac{b+b_1}{2}\right)^2}{2c^2c_1^2}} e^{\frac{\left(\frac{b+b_1}{2}\right)^2-\frac{b^2+b_1^2}{2}}{2c^2c_1^2}} = \\
&= a_2 e^{-\frac{\left(x-\frac{b+b_1}{2}\right)^2}{2c^2c_1^2}} = a_2 e^{-\frac{(x-b_2)^2}{2c_2^2}}
\end{aligned} \tag{1.31}$$

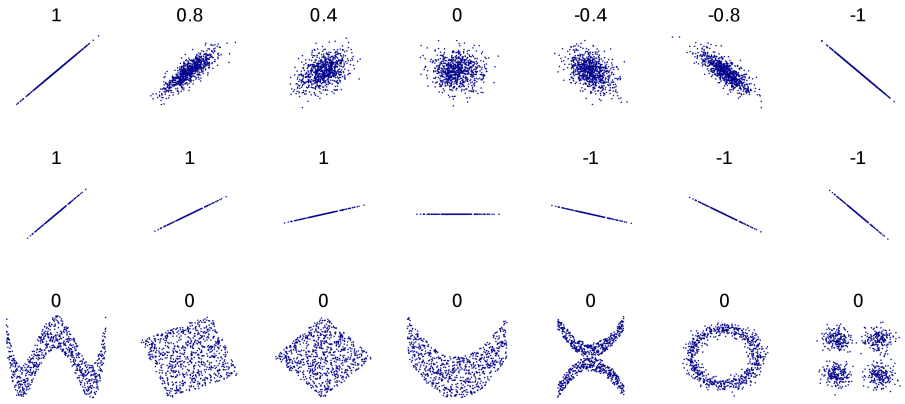


Figure 1.1: Plots of correlated and non correlated variables.

Definition 1.11. Gaussian distribution A normalized Guassian function or normal distribution is a Gaussian function with an area under the curve equal to 1. Hence it can be interpreted as a probability distribution.

To find the formula let us force the curve to have area 1:

$$\int_{-\infty}^{\infty} \mathrm{a} e^{-\frac{(\mathrm{y}-\mathrm{b})^2}{2\mathrm{c}^2}} \mathrm{dy} = \mathrm{a} \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{(\mathrm{y}-\mathrm{b})^2}{2\mathrm{c}^2}} \mathrm{dy} \tag{1.32}$$

and performing a change of integration variable:

$$\begin{aligned} &\frac{\mathrm{y}-\mathrm{b}}{\sqrt{2}\mathrm{c}} = \mathrm{x} \Rightarrow \mathrm{dx} = \mathrm{dy}\frac{1}{\sqrt{2}\mathrm{c}}\mathrm{dx} \Rightarrow \mathrm{dy} = \sqrt{2}\mathrm{cdx} \\ &\mathrm{a}\int_{-\infty}^{\infty} \mathrm{e}^{-\frac{(\mathrm{y}-\mathrm{b})^2}{2\mathrm{c}^2}} \mathrm{dy} = \sqrt{2}\mathrm{ac}\int_{-\infty}^{\infty} \mathrm{e}^{-\mathrm{x}^2}\mathrm{dx} \end{aligned} \tag{1.33}$$

where $\sqrt{2}\mathrm{c}$ is a constant so that in can be moved out of the integral. From now on we will focus just on the integral.

Unfortunately the integral has not solutions with elementary functions (Louiville theorem, see Abstract Algebra). However the definite integral exists (demonstration is skipped) and it is:

$$\int_{-\infty}^{\infty} \mathrm{e}^{-\mathrm{x}^2}\mathrm{dx} = \sqrt{\pi} \tag{1.34}$$

To have area equal to 1 we must have:

$$\mathrm{ac}\sqrt{2\pi} = 1 \Rightarrow \mathrm{a} = \frac{1}{\sqrt{2\pi}\mathrm{c}} \tag{1.35}$$

So now the function is defined by just 2 parameters.

Normally $b = \mu$ and is the mean of the function while $c = \sigma$ is the standard deviation.

As opposed to the Gaussian function, multiplying two of these functions we do not get another gaussian distribution, but a scaled guassian distribution. If the Gaussian functions have 0-mean and $\sigma^2 = 1$ then the product is still Guassian distribution with 0-mean and $\sigma^2 = 1$.

Definition 1.12. Standard Normal Gaussian distribution A Standard Normal Gaussian distribution is a Gaussian distribution with 0 mean and standard deviation 1.

1.11 Hermitian function

Definition 1.13. Hermitian function An hermitian function is a complex function for which its complex conjugate is equal to the original function with the variable changed of sign:

$$\bar{f(x)} = f(-x) \tag{1.36}$$

where the bar indicates the complex conjugate. From this definition it follows immediately that f is a Hermitian function if and only if:

- the real part of f is an even function;
- the imaginary part of f is an odd function.

1.12 Non-negative definite kernels

Definition 1.14. Non-negative definite kernels K is said to be non-negative definite (or positive semidefinite) iff

$$\sum_{i=1}^{n}\sum_{j=1}^{n} K(x_i, x_j)c_i c_j \geq 0 \tag{1.37}$$

for all finite sequences of points $\forall x_i, x_j \in \mathcal{X}$ and $\forall c_i, c_j \in \mathcal{R}$.

2 Statistical Decision Theory

2.1 Expected prediction error

Let $X \in \mathbb{R}^p$ denote a real valued random input vector, and $Y \in \mathbb{R}^p$ a real valued random output variable, with joint distribution $\Pr(X, Y)$. We seek a function $f(X)$ for predicting Y given values of the input X. This theory requires a loss function $L(Y, f(X))$ for penalizing errors in prediction, and by far the most common and convenient is squared error loss: $L(Y, f(X)) = (Y - f(X))^2$.

The estimated prediction error is

$$\begin{aligned} \mathrm{EPE}(f) &= \mathbf{E}\left[(Y - f(X))^2\right] = \int (y - f(x))^2 \, p(x, y) dx dy \\ &= \int_x \int_y (y - f(x))^2 \, p(x, y) dx dy \end{aligned} \tag{2.1}$$

Recalling $p(x, y) = p(y|x)p(x)$:

$$\begin{aligned} \mathrm{EPE}(f) &= \int_x \int_y (y - f(x))^2 \, p(y|x)p(x) dx dy \\ &= \int_x \int_y \left((y - f(x))^2 \, p(y|x) dy\right) p(x) dx \\ &= \int_x \mathbf{E}_{Y|X}\left[(y - f(X))^2 \, |X = x\right] p(x) dx \\ &= \mathbf{E}_X\left[\mathbf{E}_{Y|X}\left[(y - f(X))^2 \, |X = x\right]\right] \end{aligned} \tag{2.2}$$

So to minimize the prediction error:

$$f(x) = \arg\min_c \mathbf{E}_{Y|X}\left[(y - c(x))^2 \, |X = x\right] \quad \Rightarrow f(x) = \mathbf{E}\left[Y|X = x\right] \tag{2.3}$$

The *Nearest Neighbour* algorithm, assigns labels to points by counting and averaging the labels of the points belonging to a given neighbourhood:

$$\hat{f}(x) = \mathrm{Ave}\left(y_i | x_i \in N_k(x)\right) \tag{2.4}$$

where $N_k(x)$ contains the k points closest to x. This presents **two approximations**

- expectation is approximated by averaging
- conditioning at a point is relaxed to conditioning on some region centred at the target point.

With k sufficiently large, the average gets more stable and with large N the points will be more likely close to x. If $k, N \to \infty$ with $k/N \to 0$ the average becomes the expectation and we have the best possible estimator.

Unfortunately often we do not have so much data and some other times we might want exploit the supposed structure of data (linear, polynomial etc.).

However there is even a bigger problem when there are too many dimensions (i.e., p is large). Consider a uniformly distributed input in a p dimensional unit hypercube. Consider a hypercube neighbourhood around the target point capturing a fraction r of the total observations distributed among the unit hypercube. The edge of the neighbour hypercube will be $e_p(r) = r^{1/p}$. In 10 dimensions, using a neighborhood capturing 1% of the data we have $r(0.01) = 0.63$ so we must use 63% of the total data for one target.

On the contrary the linear regression is a **model-based approach**, i.e., one assumes that the function f(x) is approximately linear:

$$f(x) \approx x^T \beta \tag{2.5}$$

Putting this in the EPE equation:

$$\begin{aligned} f(x) &= \arg\min_c \mathbf{E}_{Y|X} \left[(y - c(x))^2 \, | X = x \right] \\ &= \arg\min_\beta \mathbf{E}_{Y|X} \left[\left(y - x^T \beta \right)^2 | X = x \right] \\ \Rightarrow \beta &= \left[\mathbf{E} \left[X \cdot X^T \right] \right]^{-1} \cdot \mathbf{E} \left[X \cdot Y \right] \end{aligned} \tag{2.6}$$

The minimum of a quadratic function is given by deriving and setting its derivative to 0.

If instead of a L_2 loss function we use L_1

$$\begin{aligned} \mathrm{EPE}(f) = \mathbf{E} \left[|Y - f(X)| \right] &= \int |y - f(x)| \, p(x, y) dx dy \\ &= \int_x \int_y |y - f(x)| \, p(x, y) dx dy \end{aligned} \tag{2.7}$$

Recalling $p(x, y) = p(y|x)p(x)$:

$$\begin{aligned} \text{EPE}(f) &= \int_x \int_y |y - f(x)|\, p(y|x)p(x)dxdy \\ &= \int_x \int_y |(y - f(x)|\, p(y|x)dy)\, p(x)dx \\ &= \int_x \mathbf{E}_{Y|X} \left[|y - f(X)|\, |X = x\right] p(x)dx \\ &= \mathbf{E}_X \left[\mathbf{E}_{Y|X} \left[|y - f(X)|\, |X = x\right]\right] \end{aligned} \tag{2.8}$$

$$f(x) = \arg\min_c \mathbf{E}_{Y|X} \left[|y - c(x)|\, |X = x\right] \tag{2.9}$$

and as already seen in 1.8, the minimizer for the sum of distances is the median.

2.1.1 Loss function for categorical variables

For categorical output variables $\mathbb{G}_k$ we have:

$$\text{EPE} = \mathbf{E}\left[L\left(G, \hat{G}(X)\right)\right] = \mathbf{E}_x \sum_{k=1}^{K} L\left[\mathbb{G}_k, \hat{G}(X)\right] \Pr\left(\mathbb{G}_k|X\right) \tag{2.10}$$

where the expectation is again taken with respect to the joint distribution $\Pr(G, X)$. Conditioning again we can write:

$$\text{EPE} = \mathbf{E}_x \sum_{k=1}^{K} L\left[\mathbb{G}_k, \hat{G}(X)\right] \Pr\left(\mathbb{G}_k|X\right) \tag{2.11}$$

where the integral over y has been substituted with the summation due to the categorical nature of the variable.

The minimizer is given by:

$$\hat{G}(x) = \arg\min_{g \in \mathbb{G}} \sum_{k=1}^{K} L\left(\mathbb{G}_k, g\right) \Pr\left(\mathbb{G}_k|X = x\right) \tag{2.12}$$

Often the *zero-one loss function* is used for categorical variables and the above simplifies to:

$$\hat{G}(x) = \arg\min_{g \in \mathbb{G}} \left[1 - \Pr\left(g|X = x\right)\right] = \arg\max_{g \in \mathbb{G}} \left[\Pr\left(g|X = x\right)\right] \tag{2.13}$$

This is known as *Bayes classifier* because **it classifies to the most probable class, using conditional discrete probability distribution**.

Note: When models or loss functions use additional parameters that penalize complexity (Lasso, Ridge and others) we cannot use the training data to determine these parameters, since we would pick those that gave interpolating fits with zero residuals but it will be unlikely to predict future data.

2.2 Bias-Variance trade-off

$$\begin{aligned} \mathrm{EPE}(f) &= \mathbf{E}\left[(Y - f(X))^2\right] = \\ &= \mathbf{E}\left[Y^2\right] - 2\mathbf{E}\left[Y\right]\mathbf{E}\left[f(X)\right] + \mathbf{E}\left[f(X)^2\right] \\ &= Y^2 - 2Y\mathbf{E}\left[f(X)\right] + \mathbf{E}\left[f(X)^2\right] \end{aligned} \tag{2.14}$$

Recalling

$$\begin{aligned} &\mathrm{BIAS}\left(Y, \mathbf{E}\left[f(X)\right]\right) = \left|Y - \mathbf{E}\left[f(X)\right]\right| \\ &\Rightarrow \mathrm{BIAS}\left(Y, f(X)\right)^2 = \left(Y - \mathbf{E}\left[f(X)\right]\right)^2 \\ &= Y^2 - 2Y\mathbf{E}\left[f(X)\right] + \mathbf{E}\left[f(X)\right]^2 \end{aligned} \tag{2.15}$$

EPE can be expressed using also the Variance definition in 1.6

$$\begin{aligned} \mathrm{EPE}(f) &= Y^2 - 2Y\mathbf{E}\left[f(X)\right] + \mathbf{E}\left[f(X)^2\right] + \mathbf{E}\left[f(X)\right]^2 - \mathbf{E}\left[f(X)\right]^2 \\ &= \mathrm{BIAS}(Y, f(X)^2 + \mathrm{Var}(Y, f(x)) \end{aligned} \tag{2.16}$$

The bias is given by the distance of our predictions from real points. Complex models have more degrees of freedom and are able to fit closer real points hence they to tend to have low bias. However, they present higher variance. On the contrary simple models (i.e., linear) have lower variance but higher bias.

The error due to variance is the amount by which the prediction, over one training set, differs from the expected predicted value, over all the training sets. As with bias, you can repeat the entire model building process multiple times. To paraphrase Manning et al (2008), variance measures how inconsistent are the predictions from one another, over different training sets, not whether they are accurate or not. A learning algorithm with low bias must be "flexible" so that it can fit the data well. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance.

For linear models fit by ordinary least squares, the estimation bias is zero. For restricted fits, such as ridge regression, it is positive, and we trade it off with the benefits of a reduced variance.

3 Taxonomy of machine learning algorithms

3.1 Supervised vs Unsupervised

The first distinction is between Supervised and Unsupervised learning. Some in-between methods exist too:

- **supervised learning**: the training data you feed to the algorithm includes the desired solutions, called labels. Typical algorithms are Linear regression, K-Nearest Neighbours, Logistic Regression, Support Vector Machines, Decision Trees and Random Forests, Neural Networks. Tasks of supervised learning are **classification** and **prediction**, i.e., **regression**.
- **unsupervised learning**: no label is passed as input and typical the algorithm must apply label or group data somehow. Its tasks are
  - **clustering**: whose goals is to group together similar groups. This is typically done by defining a measure of similarity (or dissimilarity) between points in the hyperspace. Popular algorithms are K-means, Hierarchical clustering Analysis, Expectation Maximization;
  - **Visualization and dimensionality reduction**: for visualization applications, one feeds them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted. The goal of dimensionality reduction instead is to simplify the data without losing too much information by merging correlated features. Popular algorithms are Principal component analysis (PCA), Kernel PCA, Locally-Linear Embedding, t-distributed Stochastic Neighbour Embedding
  - **Association Rule Learning**: Apriori, Eclat.
- **semi-supervised learning**: Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person.
- **Reinforcement learning**: The learning system, called an agent in this context, can observe the environment, select and perform actions,

and get rewards or penalties in return. It must then learn by itself what is the best strategy, called a **policy**, to get the most reward over time.

3.2 Batch and online Learning

Another criterion used to classify Machine Learning systems is whether or not the system can learn incrementally from a stream of incoming data.

- **batch-learning**: the system is incapable of learning incrementally: it must be trained using all the available data. this is called **offline learning**.
- **online learning**: you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini-batches. Online learning is great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously. It is also a good option if you have limited computing resources.

3.3 Instance-based vs Model based

- **instance-based learning**: the system learns the examples by heart, for example by using a measure of similarity such as number of similar words in two emails to classify spam emails.
- **Model-based learning**: it builds a model to perform predictions

4 Hands on datasets

Before looking at the machine learning algorithms, let us focus on how to prepare data to be fed to the algorithms. The focus is on the *scikit-learn* and *Pandas* libraries in Python and .

4.1 Looking at data

After having loaded the data in a Pandas dataframe, there are methods of this data structure that allow to get some information from the data:

- ***df.head()***: this method allows to get a gist on the general structure of the dataframe by looking at the value of the first 5 rows. More rows can be shown by passing as input the desired number to be shown;
- ***df.info()***: it tells the number of entries and for each column its type and the number of non-null elements. Note that string type is considered in a more general type as *object* type;
- ***df['field'].value_counts()***: this is useful when *field* is a categorical. It shows all the possible values in the dataframe for the column and for each one the number of entries having that column value;
- ***df.describe()***: this method performs some statistics on numerical attributes (null values are ignored). Particularly, it calculates the number of entries (with valid values), the mean, standard deviation, minimum and maximum and the $25\%, 50\%, 75\%$ percentiles.
- ***df.hist(bins=20)***: it plots the histogram of each numerical value. The number of bins tells how many bars (i.e., how many groups) the histogram should have;
- ***df.corr()['attribute'].sort_values(ascending=False)***: shows the correlation between the selected attribute and all the others. Remember that the correlation an indication of linear relationships only;
- ***scatter_matrix(h[attributes], figsize=(12,8))***: plots every numerical attribute against every other numerical attribute (carefulness is needed when the number of attribute is large). To give some sense to the main-diagonal plots, a histogram is plotted by the function itself. See 4.4.

Some inferences can be made from the histogram. First of all the units and scales; there even might be some hints if values have been cupped (for example when there is a natural decreasing trend but the last bin has a value significantly bigger than the others).

4.2 Train and test set

The first thing to do is to split the set between train and test. A good idea is to have the same split across different runs of the algorithms. One solution is to save the data-set and load it next times. Another solution is to use random generator seeds so that the pseudo-random sequence starts every-time from the same point. But these solutions will break next time one fetches an updated dataset.

A common solution is to use each instance's identifier to decide whether or not it should go in the test set (assuming instances have a unique and immutable identifier). For example, you could compute a hash of each instance's identifier, keep only the last byte of the hash, and put the instance in the test set if this value is lower or equal to 51 (20% of 256). This ensures that the test set will remain consistent across multiple runs, even if you refresh the dataset. It is assuming a uniform distribution by hash algorithm. This can be implemented with the following functions:

```
def split_train_test_by_id(data, test_ratio, id_column,
                           hash=hashlib.md5):
    ids = data[id_column];
    in_test_set = ids.apply(lambda _id: test_set_check(_id,
                            test_ratio, hash));
    return data.loc[~in_test_set], data.loc[in_test_set];

# The check on the id is to see the value of the last byte
# If this value is less then test_ratio*256 (assuming uniform
# distribution)
# then will return true otherwise false. The fraction of true
# will be  equal to test_ratio
def test_set_check(identifier, test_ratio, hash):
    limit = test_ratio*256; #1byte
     #digest will calculate the hash value
    res = hash(np.int64(identifier)).digest()[-1];
    return res<limit;
```

Listing 1: Example of function to split according to the last by of the hash on the id.

However, there is a **further problem** if there is not an identifier column. The solution is to create one. It is possible to use the row index but the one has make sure new data are appended in order and no data should not be deleted.If this is not possible, then you can try to use the most stable features to build a unique identifier. For example, a district's latitude and longitude are guaranteed to be stable for a few million years, so one could combine them into an ID like so and their values are unique:

```
df["id"] = df['longitude']*1000+df['latitude']
```

Listing 2: "Creating a column ID from stable features such as latitude and longitude.

4.2.1 Stratified sampling

Consider a given value, either categorical or numerical. Generally, they are not uniformly distributed among the entries. As the simplest possible example consider that in a data set of people women are the 60%, and men 40%. When splitting between train and test set it is better to keep this ratio, especially when some values are rarer.

This is true also for numerical value: in this case, one first needs to create a corresponding categorical attribute. It is important to have sufficient data for each category and at the same time categories must be representatives. First a range size is chosen (the range of the new categorical value) and it is used to divide the original values. Then the resulting valued are ceiled. Clapping is required.

After that, it is possible to stratify-sampling. *scikit-learn* has the function *StratifiedShuffleSplit* to perform it:

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
                    random_state=42);
```

4.3 Plotting the dataset

Visualing data can give other insights. Pandas dataframe has a plot method:

```
df.plot(kind="scatter", x="longitude", y="latitude", alpha=0.2);
```

Listing 4: Example of Pandas dataframe plot method.

In this way one gets darker areas according to the number of elements. It is also to make more complex graphs: for example we can use colormaps to represent the intensity of output and circle radius to represent another feature of the data:

```
h.plot(kind="scatter",x="longitude", y="latitude", alpha=0.4,
       s = h['population']/100,  label="popultion",
       c="median_house_value", cmap = plt.get_cmap('jet'),
       colorbar=True, figsize=(15,8));
plt.legend();
```

Listing 5: Example of Pandas dataframe plot with colormap and variable radius of scattered circles.

4.1 shows the prices of Californian houses using a colormap in terms of geographical position and population. This image tells you that the housing prices are very much related to the location (e.g., close to the ocean) and to the population density, as you probably knew already. It will probably be useful to use a clustering algorithm to detect the main clusters, and add new features that measure the proximity to the cluster centres.

4.4 Interpretation of scatter plot

This method is important: it gives an idea of which variable are related: if the graph looks uniform then maybe there is no relation. On the contrary other, other patterns might be visible. For example, from 4.2 it is possible to see a strong correlation between *median_house_value* and *median_income*.

Looking closer at it in 4.3,

4.3 shows these two variables are really correlated. We also notice the horizontal line at $\$500,000$ since data prices are capped.

However, there are other horizontal lines at $\$450,000$, at $\$380,000$ and maybe one at $\$280,000$. Maybe it is better to remove these districts so that the algorithm does not learn to reproduce them.

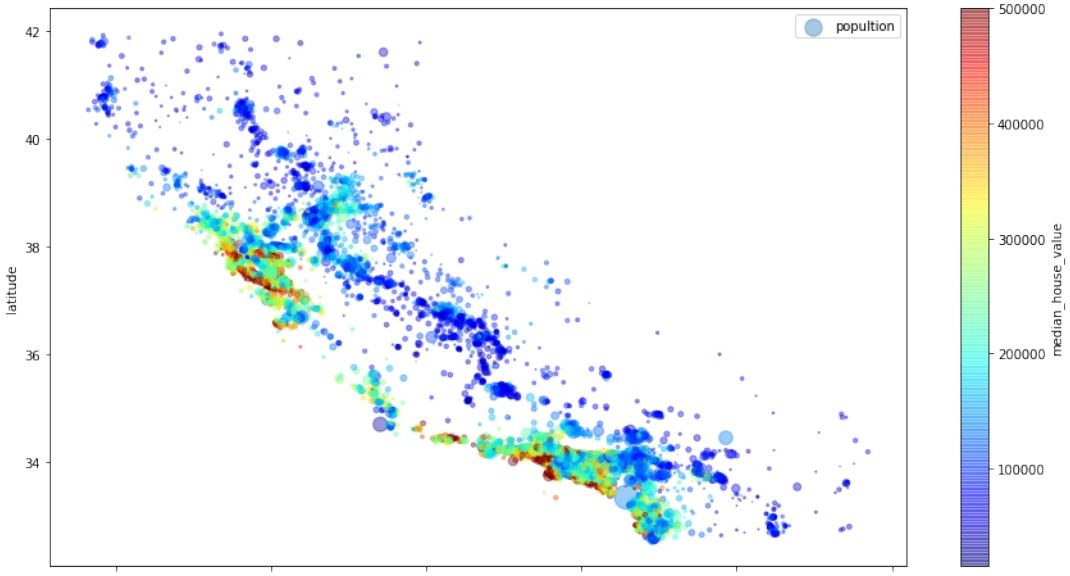


Figure 4.1: Example of plotting a dataframe. The latitude is plotted over the longitude. The color tells the intensity of the output variable while the circle radius tells are related to another input variable (population).

4.5 Data manipulation

Some values need to be properly interpreted. In the California houses data set the attribute *total_bedrooms* is the total number of rooms in a district. Rather than that, it is better to consider the approximated number of bedrooms per house, obtained by dividing the former by the total number of houses in that district. This new attribute will be more correlated to the output than the original one. Apparently houses with a lower bedroom/room ratio tend to be more expensive. In the same way, the number of rooms per household is also more informative than the total number of rooms in a district - obviously the larger the houses, the more expensive they are.

4.5.1 Convert categorical values to numerical values

Many machine algorithms work with numerical attributes and not with categorical values. It is a good idea to convert these categorical value to a numerical value. *SciKit-learn* offers the *LabelEncoder* class, which convert each categorical value to a number:

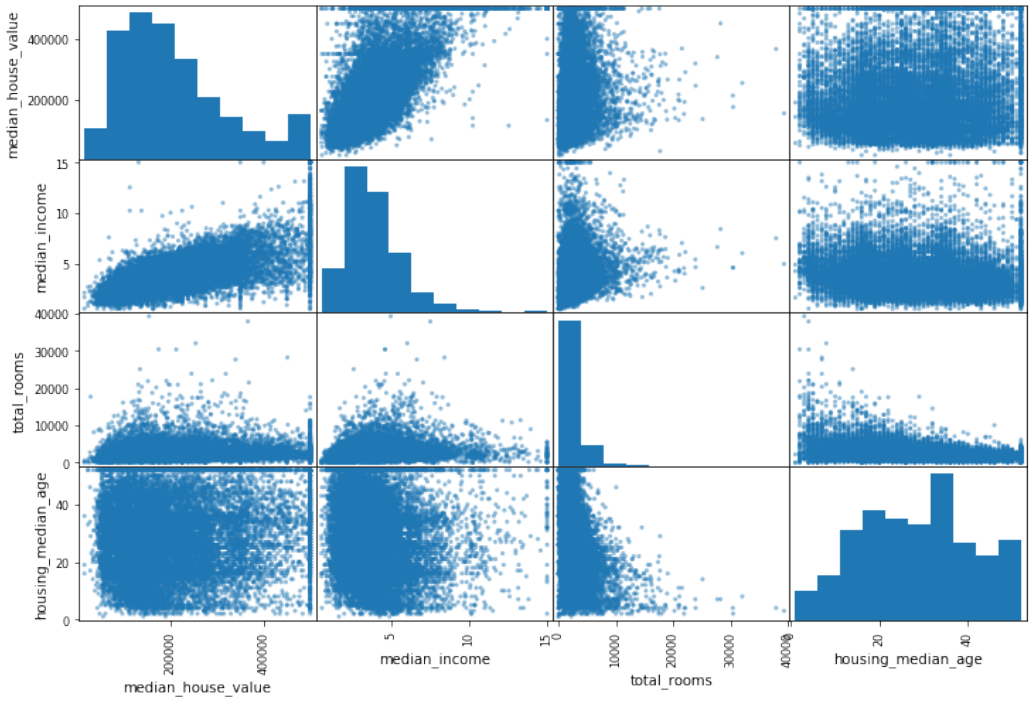


Figure 4.2: Example of scatterplot for some attributes of the California houses dataset.

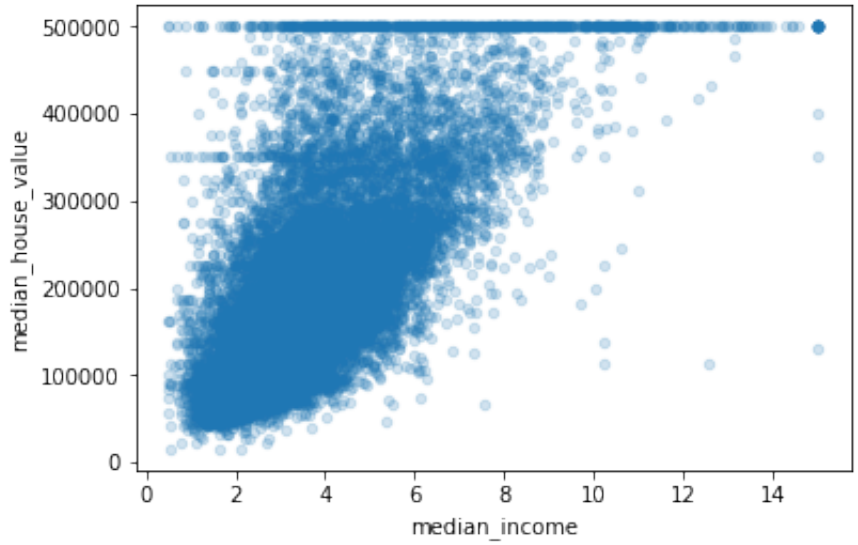


Figure 4.3: Scatterplot of the attributes *median_house_value* and *median_income*.

```
from sklearn.preprocessing import LabelEncoder
encoder = LableEncoder();
encoder = LabelEncoder();
h_cat = h['ocean_proximity'];
h_cat_encoded = encoder.fit_transform(h_cat);
```

Listing 6: Example of usage of *LabelEncoder* class

where *ocean_proximity* is a categorical attribute. However, there is a problem with this representation: ML algorithms will assume that two nearby values are more similar than two distant values. Many times this is not guaranteed to be the case. To fix this issue, a common solution is to create one binary attribute for each possible categorical value. This is called **one-hot encoding**, because only one attribute will be equal to 1 (hot), while the others will be 0 (cold).

SciKit-learn provides *OneHotEncoder* to convert integer categorical value into one-hot vectors. This function seems to be deprecated though.

```
from sklearn.compose import ColumnTransformer
h_copy = h.copy()
ct = ColumnTransformer([("onehot",OneHotEncoder(),[h_copy.columns.get_loc('ocean_pro
```

```
onehot = ct.fit_transform(h_copy.values).astype('int').toarray();
onehot
Out:
array([[1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1],
       ...,
       [0, 1, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0]])
```

Listing 7: Usage of *ColumnTransformer*

Apparently *ColumnTransformer* has no inverse so we are loosing the references to the category We create a dataframe containing the encoding of the categorical variables and then we merge it to the original dataframe:

```
from sklearn.preprocessing import LabelBinarizer
# sparse_output=True compresses the matrix not to use too much mem,
# toarray() allows to get back the array
h_copy = h.copy();
enc = LabelBinarizer(sparse_output=True);
onehot = enc.fit_transform(h_copy['ocean_proximity']);
enc.inverse_transform(np.identity(onehot.shape[1])).reshape(-1,);
ocean_prox_df = pd.DataFrame(
    data=onehot.toarray().astype('int'),
    columns=enc.inverse_transform(np.identity(onehot.shape[1])));

# Now we concatenate the original and one-hot encoded datafram
h_cat = pd.concat([h_copy.drop('ocean_proximity', axis=1),
                   ocean_prox_df], axis=1);
h_cat
```

Listing 8: Usage of *LabelBinarizer*

Alternatively, Pandas dataframe data structure has its own simpler method *pandas.get_dummies(df.categorical_attribute*:

```
h_copy = h.copy();
ocean_prox_dummies = pd.get_dummies(h_copy.ocean_proximity);
h_res = pd.concat([h_copy, ocean_prox_dummies], axis=1);
h_res.drop('ocean_proximity', axis=1, inplace=True);
```

Listing 9: Usage of *pd.get_dummies()*

4.6 Data cleaning

One big problem are missing values. These can be replaced with different techniques:

- ***df.dropna(subset=["total_bedrooms"]);***: gets rid of the entries having null for this column(s);
- ***df.drop("total_bedrooms", axis=1);***: gets rid of the entire attribute;
- ***df["total_bedrooms].fillna(median);***: set the missing value to the specified value (0, mean, median).The value (*median* in this case) has been computed with the training set so it must be saved

If more than one column has missing values, we can use the SciKitLearn class *Imputer*. Its constructor takes as input a string parameter called *strategy* that specifies the value replacement. An example is the following:

```
1 from sklearn.preprocessng import SimpleImputer
2 imputer = SimpleImputer(strategy="median");
3 h_num = h.drop('ocean_proximity', axis=1);
4 imputer.fit(h_num);
5 imputer.statistics_;
6 X = imputer.transform(h_num); #output is a numpy array
7 h_tr = pd.DataFrame(X, columns =h_num.columns)
```

Listing 10: Usage of *SimpleImputer to clean data*

4.7 Custom Transformers

Scikit allows you to create your own transformers using **ducking type programming style**. In ducking type, an object passed into a function must support all methods and attributes it s expected to have at runtime. The object type itself does not matter: in this sense, it is very different from inheritance.

For *Transformers*, three methods must be implemented:

- *fit();*: this method just calculates the parameter and it is run on the training set. The parameters are saved as internal object state.

- *transform();*: it applies the transformation to a particular set of examples. it can be called right after *fit()* on the training set or on the test set (on which one does not call *fit*)

- *fit_transform();*: joins these two steps and is used for the initial fitting of parameters on the training set, but it also returns a transformed set. Internally, it just calls first *fit()* and then *transform()* on the same data.

We can get the last one for free exploiting inheritance from the base class ***TransformerMixin***. By adding also ***BaseEstimator*** as a base class and avoiding *args and *kargs in the constructor, we will get the extra methods get_params and ti**set_params** that will be useful for automatic hyperparameter tuning.

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6;

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room = add_bedrooms_per_room;

    def fit(self, X, y=None):
        return self;

    def transform(self, X, y=None):
        rooms_per_household = X[:,rooms_ix]/X[:, household_ix];
        population_per_household = X[:,population_ix]/X[:, household_ix];
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:,bedrooms_ix]/X[:, rooms_ix];
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_
        else:
            return np.c_[X, rooms_per_household, population_per_household];

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False);
h_extra_attribs= attr_adder.transform(h_copy.values);
```

Listing 11: Example of how to create a custom transformer

4.8 Feature scaling

In general, inputs have very different scales. ML algorithms work better if inputs have the same scale. For this reason, they are scaled before applying the algorithm. Tow popular scaling methods are the following:

- ***min-max*** or ***normalization***: values are scaled to be in range $[0, 1]$. Minimum value is subtracted and then it is scaled by the maximum value. *Scikit* provides *MinMaxScaler* with *feature_range* property to change the range.
- ***standardization***: it subtracts the mean value (0-mean) and then divides by the variance to have a unit variance distribution. Standardization is much less affected by outliers but might be a problem for some algorithms (such as NN) expecting values to be within a given range.

4.9 Scikit-learn transformation pipeline

The *pipeline* data structure allows to collect the transformation to be applied to the dataset. All but last estimator must be transformers, i.e., they must have the method *fit_transform*. When one calls the pipeline's *fit()*method, it calls *fit_transform()* sequentially on all transformers, passing the output of each call as the parameter to the next call, until it reaches the final estimator, for which it just calls the *fit()* method and then *transform()*.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
num_pipeline = Pipeline([('imputer', SimpleImputer(strategy='median')),
                         ('attribs_adder', CombinedAttributesAdder()),
                          ('std_scaler', StandardScaler())
                         ]);
h_num_tr = num_pipeline.fit_transform(h.drop('ocean_proximity',axis=1));
```

4.9.1 Perform parallel transformations

The scaling of numerical values and the transformation of categorical values to many binary attributes are independent, hence they can be performed in parallel.

Scikit provides ***FeatureUnion*** class. You give it a list of transformers (which can be entire transformer pipelines), and when its ***transform()*** method is called, it runs each transformer's ***transform()*** method in parallel, waits for their output, and then concatenates them and returns the result (and of course calling its ***fit()*** method calls all each transformer's ***fit()*** method).

Before looking at the *FutureUnion*, we needed two more pieces. First of all, each subpipeline starts with a selector transformer: it simply transforms the data by selecting the desired attributes (numerical or categorical), dropping the rest, and converting the resulting DataFrame to a NumPy array. There is nothing in Scikit-Learn to handle Pandas DataFrames, so we need to write a simple custom transformer for this task. The code in 13 does this.

```
from sklearn.base import BaseEstimator, TransformerMixin

class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names;
    def fit(self, X, y=None):
        return self;
    def transform(self, X):
        return X[self.attribute_names].values
```

Listing 13: Transformer to select the desired attributes of a Pandas dataframe

Secondly, the method *fit_transform()* of *LabelBinarizer* changed from SciKit-learn version 0.18 to 0.19. The problem is that now it accepts just two inputs: the instance and the array to be encoded: *lb.fit_transform(['yes', 'no', 'no', 'yes'])*, while the pipeline requires three. We create a new class *LabelBinarizerPipelineFriendly* which inherits from *LabelBinarizery* overload *fit_transform()* method to accept another parameter *y* that actually it is not used. The method first calls *fit* and then *transform()*. See 14.

```
class LabelBinarizerPipelineFriendly(LabelBinarizer):
    def fit(self, X, y=None):
        """this would allow us to fit the model based on the X input."""
        super(LabelBinarizerPipelineFriendly, self).fit(X)
    def transform(self, X, y=None):
        return super(LabelBinarizerPipelineFriendly, self).transform(X)

    def fit_transform(self, X, y=None):
        return super(LabelBinarizerPipelineFriendly, self).fit(X).transform(X)
```

Listing 14: Custom LabelBinarizer class that can be used in a pipeline

Finally we can use *FutureUnion* as in 15.

```
from sklearn.pipeline import FeatureUnion;
num_attr = list(h.drop('ocean_proximity',axis=1))
cat_attr = ['ocean_proximity']
num_pipeline = Pipeline([
                        ('selector', DataFrameSelector(num_attr)),
                        ('imputer', SimpleImputer(strategy='median')),
                        ('attribs_adder', CombinedAttributesAdder()),
                        ('std_scaler', StandardScaler())
                        ]);
cat_pipeline = Pipeline([
                        ('selector'       , DataFrameSelector(cat_attr)),
                        ('label_binarizer', LabelBinarizerPipelineFriendly())#using
                        ]);
full_pipeline = FeatureUnion(
            transformer_list =[
                               ("num_pipeline", num_pipeline),
                               ("cat_pipeline", cat_pipeline)
                               ]);
h_prepared = full_pipeline.fit_transform(h);
```

Listing 15: Usage of *FutureUnion*

5 Linear Regression Models

We start from the *univariate linear regression*, i.e., each output consists of a single value while the input is a vector of values.

5.1 Univariate linear regression

Univariate means single output, i.e, y is a number. The basic form is

$$\mathrm{f(X)} = \beta_0 + \sum_{\mathrm{j–1}}^{\mathrm{p}} \mathrm{X_j}\beta_\mathrm{j} \tag{5.1}$$

The most popular estimation method for a linear model is the least square:

$$\mathrm{RSS}(\beta) = \sum_{\mathrm{i=1}}^{\mathrm{p}} (\mathrm{y_i} - \mathrm{f(x_i)})^2 = (\mathbf{y} - \mathbf{X}\beta)^{\mathrm{T}} (\mathbf{y} - \mathbf{X}\beta) \tag{5.2}$$

where $\mathbf{X}$ is a $\mathrm{N} \times (\mathrm{p}+1)$ matrix with each row being an input vector, $\mathbf{y}$ a N vector (we must have N input-output pairs, in this case the output is considered mono-dimensional).

By minimizing we get:

$$\hat{\beta} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \mathbf{X}^{\mathrm{T}}\mathrm{Y} \tag{5.3}$$

Geometrically we are projecting y onto the hyperplane spanned by X and the projections is referred to as $\hat{\mathrm{y}}$:

$$\hat{\mathrm{y}} = \mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \mathbf{X}^{\mathrm{T}}\mathrm{Y} = \mathrm{HY} \tag{5.4}$$

where H is called the *hat* matrix.

5.2 Equivalence of Ordinary least squares and maximum likelihood

We are using an additive model, assuming a Gaussian white noise:

$$\mathrm{y} = \beta^{\mathrm{T}}\mathbf{x} + \epsilon \tag{5.5}$$

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{5.6}$$

Adding a constant to a Gaussian random variable is equivalent to another Gaussian random variable with the mean shifted:

$$\Pr(\mathrm{y}) \sim \mathcal{N}(\beta^{\mathrm{T}}\mathbf{x}, \sigma^2) \tag{5.7}$$

Considering the matrix $\mathbf{X}$ and the output vector $\mathbf{y}$ representing the training set, used to estimate the coefficients, we have:

$$\Pr(\mathbf{y}|\mathbf{X}, \beta, \sigma^2) = \prod_{\mathrm{i}=1}^{\mathrm{N}} \Pr(\mathrm{y_i}|\mathbf{x}, \beta, \sigma^2) = \prod_{\mathrm{i}=1}^{\mathrm{N}} \mathcal{N}(\beta^{\mathrm{T}}\mathbf{x}, \sigma^2) \tag{5.8}$$

where we have assumed each observation is independent. A product of univariate Gaussian can be rewritten as a multivariate Gaussian:

$$\begin{aligned}
\Pr(\mathbf{y}|\mathbf{X}, \beta, \sigma^2) &= \prod_{\mathrm{i}=1}^{\mathrm{N}} \mathcal{N}(\beta^{\mathrm{T}}\mathbf{x}, \sigma^2) = \prod_{\mathrm{i}=1}^{\mathrm{N}} \frac{1}{(2\pi)^{\frac{1}{2}}\sigma} \mathrm{e}^{-\frac{\left(\mathrm{y_i}-\beta^{\mathrm{T}}\mathbf{x_i}\right)^2}{2\sigma^2}} = \\
&= \frac{1}{(2\pi)^{\frac{\mathrm{p}}{2}}\sigma} \prod_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{e}^{-\sum_{\mathrm{i}=1}^{\mathrm{N}} \frac{\left(\mathrm{y_i}-\beta^{\mathrm{T}}\mathbf{x_i}\right)^2}{2\sigma^2}} = \\
&= \frac{1}{(2\pi)^{\frac{\mathrm{p}}{2}}\sigma|\mathbf{I}|} \mathrm{e}^{-\frac{1}{2}(\mathbf{y}-\beta^{\mathrm{T}}\mathbf{X})^{\mathrm{T}}(\sigma^2\mathbf{I})^{-1}(\mathbf{y}-\beta^{\mathrm{T}}\mathbf{X})} = \mathcal{N}(\beta^{\mathrm{T}}\mathbf{X}, \sigma^2\mathbf{I})
\end{aligned} \tag{5.9}$$

If the variables are not independent the more general form is:

$$\mathcal{N}(\beta^{\mathrm{T}}\mathbf{X}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{\frac{\mathrm{p}}{2}}|\Sigma|^{\frac{1}{2}}} \mathrm{e}^{-\frac{1}{2}(\mathbf{y}-\beta^{\mathrm{T}}\mathbf{X})^{\mathrm{T}}\Sigma^{-1}(\mathbf{y}-\beta^{\mathrm{T}}\mathbf{X})} \tag{5.10}$$

Definition of likelihood The quantity $\Pr(\mathrm{y}|\mathbf{x}, \beta, \sigma^2)$ is called **likelihood** and tell us how much it is likely the outcome $\mathrm{y_i}$ in the dataset given the input $\mathbf{x}$ and the parameters.

A different approach to find a model that fits the data is to maximize the *likelihood* of the whole dataset:

$$\mathrm{L} = \Pr(\mathbf{y}|\mathbf{X}, \beta, \sigma^2) = \frac{1}{(2\pi)^{\frac{\mathrm{p}}{2}}|\Sigma|^{\frac{1}{2}}} \mathrm{e}^{-\frac{1}{2}(\mathbf{y}-\beta^{\mathrm{T}}\mathbf{X})^{\mathrm{T}}\Sigma^{-1}(\mathbf{y}-\beta^{\mathrm{T}}\mathbf{X})} \tag{5.11}$$

Actually maximizing the likelihood is equivalent to maximizing its logarithmic, the *log-likelihood*:

$$\begin{aligned}\log \mathrm{L} &= \sum_{\mathrm{i}=1}^{\mathrm{N}} \log \left[\frac{1}{(2\pi)^{\frac{1}{2}}\sigma} \mathrm{e}^{-\frac{\left(\mathrm{y_i}-\beta^{\mathrm{t}}\mathbf{x_i}\right)^2}{2\sigma^2}} \right] \\ &= \sum_{\mathrm{i}=1}^{\mathrm{N}} -\frac{1}{2}\log 2\pi - \log\sigma - \frac{\left(\mathrm{y_i}-\beta^{\mathrm{T}}\mathbf{x_i}\right)^2}{2\sigma^2} = \\ &= -\frac{1}{2}\log 2\pi - \log\sigma - \frac{1}{2\sigma^2}\sum_{\mathrm{i}=1}^{\mathrm{N}} \left(\mathrm{y_i}-\beta^{\mathrm{T}}\mathbf{x_i}\right)^2\end{aligned} \tag{5.12}$$

As already done for OLS, taking the derivative w.r.t. β and setting it to 0:

$$\begin{aligned}&\frac{\partial \log \mathrm{L}}{\partial \beta} = -\frac{1}{2\sigma^2}(-2)\left(\mathbf{y_i}-\beta^{\mathrm{T}}\mathbf{x_i}\right) = 0 \\ &\Rightarrow \left(\mathbf{y}-\beta^{\mathrm{T}}\mathbf{X}\right) = 0 \Rightarrow \beta = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\end{aligned} \tag{5.13}$$

This is the same solution of the OLS: the two models are equivalent.
The two models are equivalent assuming a normal distribution.

5.3 Expectation of the parameter estimation: unbiased estimator

Computing the expectation of $\hat{\beta}$:

$$\begin{aligned}&\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\hat{\beta}\right] = \sum \hat{\beta}\Pr(\mathbf{y}|\mathrm{X},\beta,\sigma^2) = \\ &= \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\sum \mathbf{y}\Pr(\mathbf{y}|\mathrm{X},\beta,\sigma^2) = \\ &= \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}\right] = \\ &= \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta = \beta\end{aligned} \tag{5.14}$$

This is an **unbiased estimator**.

Now let us calculate the covariance matrix:

$$\begin{aligned}
&\mathrm{Cov}\left[\hat{\beta}\right] = \\
&= \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\left(\hat{\beta} - \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\beta\right]\right)\left(\hat{\beta} - \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\beta\right]\right)^{\mathrm{T}}\right] = \\
&= \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\left(\hat{\beta} - \beta\right)\left(\hat{\beta} - \beta\right)^{\mathrm{T}}\right] = \\
&= \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\hat{\beta}\hat{\beta}^{\mathrm{T}}\right] - \beta\beta^{\mathrm{T}}
\end{aligned} \tag{5.15}$$

and

$$\begin{aligned}
&\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\hat{\beta}\hat{\beta}^{\mathrm{T}}\right] = \\
&= \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\left(\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right)\left(\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right)^{\mathrm{T}}\right] = \\
&= \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}\mathbf{y}^{\mathrm{T}}\right]\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}
\end{aligned} \tag{5.16}$$

and recalling from 5.9 $\Pr(\mathbf{y}) \sim \mathcal{N}(\beta^{\mathrm{T}}\mathbf{X}, \sigma^2\mathbf{I})$

$$\begin{aligned}
&\mathrm{Cov}\left[\mathbf{y}\right] = \sigma^2\mathbf{I} = \\
&= \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}\mathbf{y}^{\mathrm{T}}\right] - \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}\right]\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\right]
\end{aligned} \tag{5.17}$$

Rearranging

$$\begin{aligned}
&\Rightarrow \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}\mathbf{y}^{\mathrm{T}}\right] = \sigma^2\mathbf{I} + \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}\right]\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\right] = \\
&= \sigma^2\mathbf{I} + \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{X}\beta\right]\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\right] = \\
&= \sigma^2\mathbf{I} + \mathbf{X}\beta\beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}
\end{aligned} \tag{5.18}$$

Substituting:

$$\begin{aligned}
&\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\hat{\beta}\hat{\beta}^{\mathrm{T}}\right] = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\left(\sigma^2\mathbf{I} + \mathbf{X}\beta\beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\right)\mathrm{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} = \\
&= \sigma^2\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}
\end{aligned} \tag{5.19}$$

and finally variance-covariance matrix of the least square parameters is

$$\mathrm{Cov}\left[\hat{\beta}\right] = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\sigma^2 \tag{5.20}$$

$$\mathrm{Var}(\hat{\beta}) = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\sigma^2 \tag{5.21}$$

5.4 Noise variance estimation

We can find an estimation of the noise variance from the maximum likelihood model using the same procedure used to find the parameters, i.e., taking the derivative and equating it to 0:

$$\begin{aligned}
&\frac{\partial \log \mathrm{L}}{\partial \sigma} = \sum_{\mathrm{i}=1}^{\mathrm{N}} -\frac{1}{\sigma} + \frac{1}{\sigma^3}\left(\mathrm{y_i} - \mathbf{x}_{\mathrm{i}}^{\mathrm{T}}\beta\right)^2 = 0 \Rightarrow \frac{\mathrm{N}}{\sigma} + \frac{1}{\sigma^3}\sum_{\mathrm{i}=1}^{\mathrm{N}}\left(\mathrm{y_i} - \mathbf{x}^{\mathrm{T}}\beta\right)^2 = 0 \\
&\Rightarrow \hat{\sigma^2} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\left(\mathrm{y_i} - \mathbf{x}_{\mathrm{i}}^{\mathrm{T}}\beta\right)^2
\end{aligned} \tag{5.22}$$

This can be re-expressed as

$$\begin{aligned}
&\Rightarrow \hat{\sigma^2} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\left(\mathrm{y_i} - \mathbf{x}^{\mathrm{T}}\hat{\beta}\right)^2 = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\left(\mathrm{y_i} - \mathbf{x}_{\mathrm{i}}^{\mathrm{T}}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right)^2 = \\
&= \frac{1}{\mathrm{N}}\left(\mathbf{y} - \mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right)^{\mathrm{T}}\left(\mathbf{y} - \mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right) = \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{y} - 2\mathbf{y}^{\mathrm{T}}\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} + \mathbf{y}^{\mathrm{T}}\mathbf{X}\cancel{\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \left(\mathbf{y}^{\mathrm{T}}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{X}\hat{\beta}\right)
\end{aligned} \tag{5.23}$$

Taking the expectation w.r.t. $\Pr(\mathbf{y}|\mathbf{X}, \beta, \sigma^2)$:

$$\begin{aligned}
&\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\hat{\sigma}^2\right] = \frac{1}{\mathrm{N}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{X}\hat{\beta}\right] = \\
&= \frac{1}{\mathrm{N}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\mathbf{y}\right] - \frac{1}{\mathrm{N}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right]
\end{aligned} \tag{5.24}$$

Suppose $\mathbf{t} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$, then $\mathbf{E}_{\mathrm{p}(\mathbf{t})}\left(\mathbf{t^T A t}\right) = \mathbf{Tr}(\mathbf{A\Sigma}) + \mu^{\mathbf{T}}\mathbf{A}\mu$ with $\mu = \mathbf{X}\beta$

$$
\begin{aligned}
&\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\hat{\sigma}^2\right] = \frac{1}{\mathrm{N}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\mathbf{y}\right] + \\
&-\frac{1}{\mathrm{N}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right] = \\
&= \frac{1}{\mathrm{N}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\mathbf{I}_{\mathrm{N}}\mathbf{y}\right] + \\
&-\frac{1}{\mathrm{N}}\mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\mathbf{y}^{\mathrm{T}}\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}\right] = \\
&= \frac{1}{\mathrm{N}}\left(\mathbf{Tr}\left(\sigma^2\mathbf{I}_{\mathrm{N}}\right) + \beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta\right) + \\
&-\frac{1}{\mathrm{N}}\left(\mathbf{Tr}\left[\sigma^2\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\right] + \beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta\right) = \\
&= \frac{1}{\mathrm{N}}\left(\mathrm{N}\sigma^2 + \beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta\right) - \frac{1}{\mathrm{N}}\left(\sigma^2\mathbf{Tr}\left[\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\right] + \beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta\right) = \\
&= \sigma^2 + \cancel{\frac{1}{\mathrm{N}}\beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta} - \frac{\sigma^2}{\mathrm{N}}\mathbf{Tr}\left[\mathbf{X}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\right] \cancel{-\frac{1}{\mathrm{N}}\beta^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathrm{X}\beta} = \\
&= \sigma^2 - \frac{\sigma^2}{\mathrm{N}}\mathbf{Tr}\left[\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}\right] = \sigma^2 - \frac{\sigma^2}{\mathrm{N}}\mathbf{Tr}\left[\mathbf{I}_{\mathrm{p}}\right] \\
&\Rightarrow \mathbf{E}_{\Pr(\mathbf{y}|\mathbf{X},\beta,\sigma^2)}\left[\hat{\sigma}^2\right] = \sigma^2\left(1 - \frac{\mathrm{p}}{\mathrm{N}}\right)
\end{aligned}
\tag{5.25}
$$

where lastly we have used the product property of the trace (see 1.1.1).

Normally $\mathrm{p} < \mathrm{N}$ hence the estimate of the variance is smaller than the true variance, so this estimator is **biased**. The estimate gets closer to the real value when p/N is small, i.e., assuming p is fixed, increasing the samples used.

This result might be strange. First of all notice from 5.22 that the closer the model gets to the data, the smaller $\hat{\sigma}^2$. By definition the parameter estimates are the ones that minimise the noise and hence $\hat{\sigma}^2$. As a consequence, when using the true parameters we would get equal or higher variance.

To estimate the true variance we can use the following formula:

$$\hat{\sigma}^2 = \frac{1}{\mathrm{N-p-1}} \sum_{\mathrm{i}=1}^{\mathrm{N}} (\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i})^2 \tag{5.26}$$

$\mathrm{N-p-1}$ makes this estimation unbiased i.e., $\frac{1}{\mathrm{N-p-1}}\mathbf{E}[\hat{\sigma}^2] = \sigma^2$.

5.5 Interpretation of covariance

Consider a covariance matrix of size 2×2 for a two parameter model (i.e., a line on the plane) and suppose the first diagonal element, corresponding to the variable $\hat{\beta}_0$ is much bigger than the second one corresponding to $\hat{\beta}_1$. This means that we can change $\hat{\beta}_0$ a little without affecting too much the model. On the contrary if the variance is small, small changes will affect significantly the model. Sometimes this happens when one variable has a much higher absolute value.

If the values on the off-diagonals are negative, then when increasing one coefficient i.e., $\hat{\beta}_0$, the other must be decreased to have the line to pass as close as possible to all points. For example in 2D, increasing $\hat{\beta}_0$ reduces the coefficient value: if it is positive, the line will be "more horizontal", if negative it becomes steeper, "more vertical".

5.6 Z-score

Let us assume data were really generated by a linear model but were corrupted by Gaussian noise with 0 mean and variance σ^2:

$$\mathrm{Y} = \beta_0 + \sum_{\mathrm{i}}^{\mathrm{p}} \beta_\mathrm{i} \mathrm{X_i} + \epsilon \tag{5.27}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The estimated parameters will still be a normal distribution:

$$\hat{\beta} \sim \mathcal{N}(\beta, (\mathbf{X}^\mathrm{T}\mathbf{X})^{-1}\sigma^2) \tag{5.28}$$

Definition 5.1. Z-score A Z-score, is a numerical and statistical measurement of a value's relationship to the mean (average) of a group of values, measured in terms of standard deviations from the mean. If a Z-score is 0, it indicates that the data point's score is identical to the mean score. A Z-score of 1.0 would indicate a value that is one standard deviation from the mean. Z-scores may be positive or negative, with a positive value indicating the score is above the mean and a negative score indicating it is below the mean. Z-scores are measures of an observation's variability

$$\mathrm{z_j} = \frac{\mathrm{x} - \mu}{\sigma} \tag{5.29}$$

In machine learning the z-value is generally intended as the regression coefficient divided by its standard error. It is also sometimes called the z-statistic. If the z-value is too big in magnitude (i.e., either too positive or too negative), it indicates that the corresponding true regression coefficient is not 0 and the corresponding X-variable matters. A good rule of thumb is to use a cut-off value of 2 which approximately corresponds to a two-sided hypothesis test with a significance level of $\alpha = 0.05$.

Z-values are computed as the test statistic for the hypothesis test that the true corresponding regression coefficient β_j is 0. In hypothesis testing, one assumes the null hypothesis is true, and then see if data provide evidence against it. So in this case, β is assumed to be 0. That is, the expectation of the fitted regression coefficient $\hat{\beta}$ is assumed to be 0:

$$\mathrm{z_j} = \frac{\hat{\beta}_\mathrm{j}}{\hat{\sigma}\sqrt{\mathrm{v_j}}} \tag{5.30}$$

where the denominator is the standard deviation (root square of the variance) of the parameter (from 5.28) with $\mathrm{v_j}$ being the diagonal element of $(\mathrm{X^T X})^{-1}$.

Being the parameters normally distributed by assumption, the standard error tells you how far on average sample parameters tend to deviate from the mean population parameter, i.e., from the true parameter β.

Theorem 1 (**The Gauss-Markov theorem**)**.** Among all linear unbiased estimators, the least square estimates of the parameters are the ones having smallest variance and consequently from 2.16 is the one with the smallest mean squared error (the bias-squared term for unbiased estimator is by definition 0).

Proof. Let $\hat{\beta} = \mathrm{Cy}$ be another linear estimator of β with $\mathrm{C} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}} + \mathrm{D}$

$$\mathbf{E}\left[\hat{\beta}\right] = \mathbf{E}\left[\mathrm{Cy}\right] = \mathbf{E}\left[\left(\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}} + \mathrm{D}\right)(\mathrm{X}\beta + \epsilon)\right] = \tag{5.31}$$

$$= \left(\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{X}\beta + \mathrm{DX}\beta\right) + \cancel{\left(\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}} + \mathrm{D}\right)\mathbf{E}[\epsilon]} = \tag{5.32}$$

$$= (\beta + \mathrm{DX}\beta) = (\mathrm{I} + \mathrm{DX})\,\beta \tag{5.33}$$

$$\tag{5.34}$$

where $\mathbf{E}[\epsilon] = 0$.

To be an unbiased estimator $\mathrm{DX} = 0$, then

$$\begin{aligned}
\mathrm{Var}(\hat{\beta}) &= \mathrm{Var}(\mathrm{Cy}) = \mathrm{C}\mathrm{Var}(\mathrm{y})\mathrm{C}^{\mathrm{T}} = \sigma^2\mathrm{C}\mathrm{C}^{\mathrm{T}} = \\
&= \sigma^2\left(\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}} + \mathrm{D}\right)\left(\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}} + \mathrm{D}\right)^{\mathrm{T}} = \\
&= \sigma^2\left(\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}} + \mathrm{D}\right)\left(\mathrm{X}\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1} + \mathrm{D}^{\mathrm{T}}\right) = \\
&= \sigma^2(\cancel{\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{X}}\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1} + \left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{D}^{\mathrm{T}} + \\
&\quad + \mathrm{DX}\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1} + \mathrm{D}\mathrm{D}^{\mathrm{T}}) = \\
&= \sigma^2(\cancel{\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{X}}\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1} + \left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}(\mathrm{DX})^{\mathrm{T}} + \\
&\quad + \mathrm{DX}\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1} + \mathrm{D}\mathrm{D}^{\mathrm{T}})
\end{aligned}$$

$$\begin{aligned}
&\mathrm{DX} = 0 \\
\Rightarrow\ &\mathrm{Var}(\tilde{\beta}) = \sigma^2\left(\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1} + \cancel{\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}(\mathrm{DX})^{\mathrm{T}}} + \cancel{\mathrm{DX}\left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}} + \mathrm{D}\mathrm{D}^{\mathrm{T}}\right) \\
&\mathrm{Var}(\tilde{\beta}) = \mathrm{Var}(\hat{\beta}) + \sigma^2\mathrm{D}\mathrm{D}^{\mathrm{T}}
\end{aligned} \tag{5.35}$$

□

5.7 Orthogonalization

Normally inputs are not perpendicular but can be orthogonalized. The goal is to define a new orthogonal basis for the data. The procedure, named **Grand-**

Schmidt procedure consists of building up a new matrix of our data starting from **X** matrix, whose columns are orthogonal. The new columns are obtained as linear combinations of the columns of **X**, hence they span the same space. The procedure can be summarized in the following steps: The

---

initialize $z_0 = x_0 = \mathbf{1}$ where $\mathbf{1}$ is a vector of all ones;
For $j = 1, \cdots, p$ regress x_j on $z_0, \cdots, z_{p-1}$ to produce the coefficients $\hat{\gamma}_{lj} = \frac{\langle z_l, x_j \rangle}{\langle z_l, z_l \rangle}$ for $l = 0, \cdots, j-1$;
Calculate the residual vectors as $z_j = x_j - \sum_{k=0}^{j-1} \hat{\gamma}_{kj} z_k$;
Regress $\mathbf{y}$ on the residual z_p to get the estimate $\hat{\beta}_p = \frac{\langle z_p, y \rangle}{\langle z_p, z_p \rangle}$

---

first row of Γ is $(1, 0, 0, \cdots 0)$ because of the initialization of x_0, z_0 as vectors of ones in the first step of the algorithm. From a mathematical point of view we are building orthogonal vectors that describe the same space of the one described by the column of **X**. To start the orthogonalization procedure, we take the first column vector of **X** as it is, in other words, the first direction remains the same and we are not rescaling the vector. At the second step, we take the second column vector of **X**, we project it onto z_0 to get the coefficient $\Gamma_{1,0}$, and then we get the second direction orthogonal to the first one as the vectorial difference of the column of **X** and its projection.

When inputs are correlated, the residual will be close to zero generating instabilities in the coefficients $\hat{\beta}_j$ and the z-score will be small.

The algorithm in matrix form is

$$\mathbf{X} = \mathbf{Z\Gamma}$$

where z_j are the column vectors of Z and Γ is upper triangular. Introducing the diagonal matrix D with j-th diagonal element $D_{jj} = ||z_j||$ we get:

$$\mathbf{X} = \mathbf{ZD^{-1}D\Gamma} = \mathbf{QR} \tag{5.36}$$

where Q is an orthogonal, $Q^T Q = I$, $N \times (p+1)$ matrix and R is a $(p+1)(p+1)$ upper triangular matrix. The Q matrix contains an orthonormal basis spanning the same space spanned by **X**. The R matrix contains the coefficients that multiply the orthonormal basis to get back the original matrix X. Consider the first step of the orthogonalization procedure. It takes the first

column of $\mathbf{X}$ and just rescales it to have norm 1. So to get back the first column of vector just a coefficient is needed. Hence, the first column of R is just a column of 0s except the first term. Then the second column of $\mathbf{X}$ is projected onto q_1 to get the first multiplier: this is the first element of the second column of R.The residual is perpendicular to q_1 and can be used as second axis after normalization: the scale factor is the second element of the column of R, all the other elements will be 0. That is why R is upper triangular.

The least square solution becomes

$$\begin{aligned} \hat{\beta} &= R^{-1}Q^T y \\ \hat{y} &= QQ^T y \end{aligned} \tag{5.37}$$

5.8 Multivariate output

We can rewrite the equation in matrix form:

$$Y = XB + E \tag{5.38}$$

where Y is a $N \times K$ matrix, X is $N \times (p+1)$, B is $(p+1) \times K$, E has the same dimensions of Y. The root squared error is

$$\begin{aligned} RSS(B) &= \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f(x_i))^2 = tr\left[(Y - XB)^T (Y - XB)\right] \\ \Rightarrow B &= \left(X^T X\right)^{-1} X^T Y \end{aligned} \tag{5.39}$$

Multiple outputs do not affect one another's least square estimates.

5.9 Subset selection

Least square estimates sometimes show low bias but high variance resulting in a non-satisfactory prediction accuracy. This can be improved by setting some coefficients to 0 to sacrifice a little of bias to reduce significantly the variance.

Other times it is useful to reduce the input dimensionality for easiness of interpretation or for computation purposes.

5.9.1 Best subset selection

This algorithm finds for each $k \in 0, 1, 2, \cdots p$, the subset of size k that gives the smallest residual sum of squares. Note that if a variable is in the best subset of size m, it might not be in the subsets of larger size (and of course neither in the smallest ones).

5.9.2 Forward stepwise selection

Searching for all the subsets is too computationally intensive (and infeasible for $p > 40$). The *forward stepwise* algorithm starts with the intercept and then sequentially adds to the model the predictor that most improves the fit. QR decomposition can be exploited to choose the next candidate. Particularly, we start from the intercept so that the first normal vector, q_1 is derived from the column of 1s by normalization. At each step we will have q vector features that have already been used to form a partial basis Q_q of size $n \times q$, and the remaining $p - q$ features that still must be exploited. For each remaining feature, calculate the residual:

$$r_k = x_k - \left(x_k^T Q_q Q_q^T\right)^T = x_k - Q_q Q_q^T x_k \tag{5.40}$$

The possible orthonormal vector is the normalized residual, $\tilde{q}_{q+1}$. The column x_k to be considered to form the new direction q_{q+1} is the one for which the product $y^T \tilde{q}_{q+1}$ is maximized. From this best candidate and the matrix Q_q, we build the matrix Q_{q+1}:

$$Q_{q+1} = \left[Q_q, q_{q+1}\right] \tag{5.41}$$

The R matrix must be updated too: first a row of zeros is appended to the end. Then a column containing the coefficients is obtained by projection of the vector x_k onto Q_{q+1}:

$$R_{q+1} = \left[\begin{array}{c|c} R_q & Q_{q+1}^T x_k \\ \hline 0 \quad \cdots \quad 0 & \end{array}\right] = \left[\begin{array}{c|c} R_q & \begin{array}{c} \langle q_1, x_k \rangle \\ \langle q_2, x_k \rangle \\ \vdots \\ \langle q_{q+1}, x_k \rangle \end{array} \\ \hline 0 \quad \cdots \quad 0 & \end{array}\right] \tag{5.42}$$

This algorithm is a greedy sub-optimal algorithm. Statistically it will have lower variance but higher bias.

5.9.3 Backward stepwise selection

Backward stepwise selection starts with the full model and sequentially removes the predictor having least impact on the model, i.e., having the smallest Z – score.

Note: this algorithm can only be applied when N > p while *forward stepwise* can always be used.

5.9.4 Implementations

[From ESLII pg. 60] Some software packages implement hybrid stepwise-selection strategies that consider both forward and backward moves at each step, and select the "best" of the two. For example in the R package the step function uses the AIC criterion for weighing the choices, which takes proper account of the number of parameters fit; at each step an add or drop will be performed that minimizes the AIC score.

Other more traditional packages base the selection on F-statistics, adding "significant" terms, and dropping "non-significant" terms. These are out of fashion.

5.9.5 Forward stagewise regression

As forward stepwise, it starts with the intercept. At each step the algorithm identifies the variable most correlated with the residual and for the chosen variable it computes the linear regression coefficient using the residual (instead of the full target) itself and just the chosen variable (instead of the full input matrix). This scalar value is added to the old value of the coefficient.

This is continued till none of the variables have correlation with the residual. The correlation can be calculated as a scalar product, it must be remembered to use absolute values when comparing correlations.

At each step only one coefficient is updated and the update might not give the final coefficient so that the number of steps is bigger than p. This slow-fitting pays in high dimensions.

Note: forward stagewise is very competitive in

5.10 Shrinkage methods

Subset selection methods either keep or remove a predictor. It has higher variance. Shrinkage or regularization methods are more continuous. They

force the model to keep the weights as small as possible.

5.10.1 Ridge regression

Ridge regression shrinks the coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squared errors

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{\text{i}=1}^{\text{N}} \left(\text{y}_\text{i} - \beta_0 - \sum_{\text{j}=1}^{\text{p}} \text{x}_\text{ij}\beta_\text{j} \right)^2 + \lambda \sum_{\text{j}=1}^{\text{p}} \beta_\text{j}^2 \right\} \tag{5.43}$$

where λ is a parameter controlling the amount of shrinkage: the bigger the value the greater the amount of shrinkage. This concept is also used in the Neural Networks. Another way to express 5.43 is the following:

$$\begin{aligned} \hat{\beta}^{\text{ridge}} &= \arg\min_{\beta} \left\{ \sum_{\text{i}=1}^{\text{N}} \left(\text{y}_\text{i} - \beta_0 - \sum_{\text{j}=1}^{\text{p}} \text{x}_\text{ij}\beta_\text{j} \right)^2 \right\} \\ \text{subject to} &\sum_{\text{j}=1}^{\text{p}} \beta_\text{j}^2 \leq \text{t} \end{aligned} \tag{5.44}$$

where there is a one-to-one correspondence between λ and t. Note that the penalization term does not consider β_0 otherwise the procedure will depend on the chosen origin for Y.

This algorithm solves the problem of high variance in case of correlated inputs when big coefficients of correlated variables can be cancelled out. With a constraints on the coefficients this problem is alleviated.

The coefficients are not preserved when the input is scaled. Generally inputs are standardized before applying the algorithm.

$$\text{RSS}(\lambda, \beta) = (\text{y} - \text{X}\beta)^\text{T} (\text{y} - \text{X}\beta) + \lambda\beta^\text{T}\beta \tag{5.45}$$

$$\hat{\beta}^{\text{ridge}} = \left(\text{X}^\text{T}\text{X} + \lambda\text{I}\right)^{-1} \text{X}^\text{T}\text{y} \tag{5.46}$$

Now even if X is not full rank, the problem is non singular (the inverse exists). In case of orthonormal inputs, the ridge coefficients are the same of least square but scaled: $\hat{\beta}^{\text{ridge}} = \frac{\hat{\beta}}{1+\lambda}$.

The parameter λ can also be derived assuming a prior distribution $y_i \sim \mathcal{N}\left(\beta_0 + x_i^T\beta, \sigma^2\right)$ and the parameters β_j are distributed as $\mathcal{N}\left(0, \tau^2\right)$. Then from 5.43 $\lambda = \frac{\sigma^2}{\tau^2}$.

Applying the SVD decomposition of the matrix $\mathbf{X} = \mathbf{UDV^T}$ where X is N × p, U is N × pand V is p × p, the latter two both orthogonal with the columns of U spanning the column space of X and the columns of V spanning the row space of X. D is a p × p diagonal matrix with the elements $d_1 \geq d_2 \geq \cdots \geq d_p \geq 0$ called singular value decomposition of X. If any $d_j = 0$ then X is singular.

The least squares equation can be rewritten as

$$\mathbf{X}\hat{\beta}^{ls} = \mathbf{X}\left(\mathbf{X^T X}\right)^{-1}\mathbf{X^T y} = \mathbf{UU^T}y \tag{5.47}$$

In case of the ridge regression, the coefficients are

$$\begin{aligned}\mathbf{X}\hat{\beta}^{ridge} &= \mathbf{X}\left(\mathbf{X^T X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X^T y} = \mathbf{UU^T y} = \\ &= \sum_{j-1}^{p}\mathbf{u}_j\frac{d_j^2}{d_j^2 + \lambda}\mathbf{u_j}^T y\end{aligned} \tag{5.48}$$

where u_j are the column vectors. So ridge regression first computes the coordinates of y with respect to the orthonormal basis **U**, it then shrinks those coordinates since $\lambda \geq 0$. A greater amount of shrinkage is applied to the coordinates of basis vector with smaller d_j, corresponding to elements with small variance.

5.10.2 Lasso regression

The lasso regression is similar to ridge regression but it uses a L_1 penalization instead of L_2.

$$\hat{\beta}^{lasso} = \arg\min_{\beta}\left\{\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p}x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right\} \tag{5.49}$$

or equivalently

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{\text{i}=1}^{\text{N}} \left(\text{y}_\text{i} - \beta_0 - \sum_{\text{j}=1}^{\text{p}} \text{x}_{\text{ij}} \beta_\text{j} \right)^2 \right\}$$
$$\text{subject to} \sum_{\text{j}=1}^{\text{p}} |\beta_\text{j}| \leq \text{t} \tag{5.50}$$

If $\text{t} > \text{t}_0 = \sum_1^{\text{p}} |\beta_\text{j}|$ where β_j are the least square coefficients, then the lasso coefficients are the same of the least squares ones. If $\text{t} = \text{t}_0/2$ then the least square coefficients are shrunk by 50% on average. Making t sufficiently big will cause some of the coefficients to be exactly 0.

In 5.1, the blue areas show the constraints for the estimates of ridge and lasso regressions, while the ellipses are contours of the residual sum of squares, centred at β^{ls}, in case of only two coefficients. For the lasso one of the coefficients is 0 when it hits one of the corner. In higher dimensions the figure becomes a rhomboid with many corners (and faces) so it becomes easy to hit a corner.

Also other penalization factors can be chosen, 5.2

5.10.3 Least angle regression

It is similar to forward stepwise. At first it identifies the variable most correlated with the response but instead of fitting it completely, it moves the variable toward its least squares value. As soon as another value gets correlated to the residual as the previous variable, the process is paused. The second variable joins the active set and their coefficients are moved together in a way that keeps their correlations tied and decreasing. The process is continued until all the variables are in the model. After $\min(\text{N} - 1, \text{p})$ steps we we arrive at the full least squares solution. The coefficient profile evolves as

$$\beta_{\mathcal{A}_\text{k}} = \beta_{\mathcal{A}_\text{k}} + \alpha\delta_\text{k} \tag{5.52}$$

If to the LAR algorithm we add the following rule i.e.,

If a non-zero coefficient hits 0*, drop its variable from the active set of variables and recompute the current joint least squares direction.*

we get the same coefficient path of the lasso and this is called LAR(lasso). So this become an efficient solution to compute the Lasso problem, especially

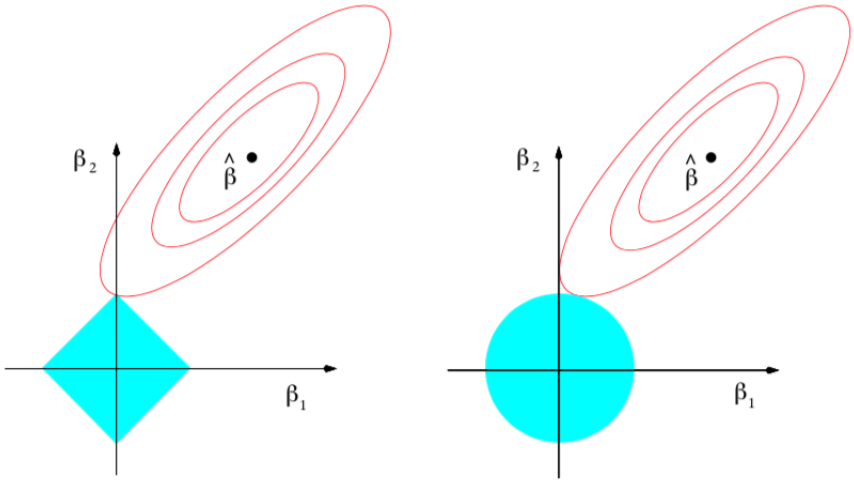


Figure 5.1: Constraints of Ridge and Lasso regressions.

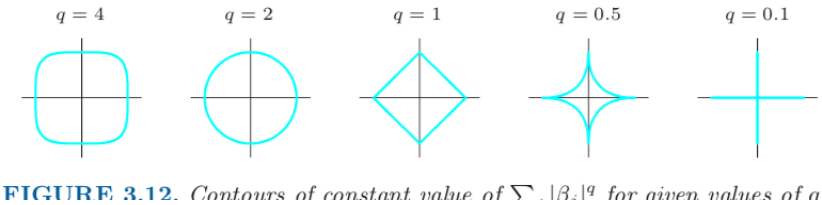


FIGURE 3.12. *Contours of constant value of* $\sum_j |\beta_j|^q$ *for given values of* q.

Figure 5.2: Shapes of different penalization factors.

standardize the predictors to have 0 mean and unit norm;
$\mathcal{A}_k \leftarrow 0$;
$r \leftarrow y - \bar{y}$, $\beta_i = 0$ for $i \neq 0$;
while $|\mathcal{A}_k| < p$ **do**
  find the predictor most correlated to r;
  insert the predictor in the active set $\mathcal{A}_k$;
  move the coefficients of the predictors in the active set to the direction defined by their joint least squares coefficient of the current residual, i.e.,

$$\delta_k = \left(X_{\mathcal{A}_k}^T X_{\mathcal{A}_k}\right)^{-1} X_{\mathcal{A}_k}^T r_k \tag{5.51}$$

  (where $\mathcal{A}_k$ is the current active set of variables) until some other competitor x_l has as much correlation with the current residual;
  $r_k \leftarrow y - X_{\mathcal{A}_k}\beta_{\mathcal{A}_k}$

with $N >> p$ since Lasso can take more than p steps while LAR require p steps and it is efficient since it requires the same complexity as that of a single least squares fit using the p predictors.

5.11 Derived input directions methods

When a large number of inputs is present, often there is a high correlation among them. In this case is convenient to regress on a new set inputs obtained form a linear combination of the original input.

5.11.1 Principal component analysis

The first input must be standardized since this analysis depends on the scaling. The principal components are defined as

$$z_i = Xv_i \tag{5.53}$$

where v_i are the column vectors of V from the SVD decomposition of $X = U\Sigma V^T$ (recall that z_m are orthogonal)(1.4). The algorithm then regress X on $z_1, \cdots, z_M$ for $M \leq p$ and we have:

$$\hat{\mathrm{y}}^{\mathrm{pcr}}_{(\mathrm{M})} = \bar{\mathrm{y}}\mathbf{1} + \sum_{\mathrm{m}=1}^{\mathrm{M}} \hat{\theta}_{\mathrm{m}} \mathrm{z}_{\mathrm{m}} \tag{5.54}$$

where $\hat{\theta}_{\mathrm{m}} = \langle \mathrm{z}_{\mathrm{m}}, \mathrm{y} \rangle / \langle \mathrm{z}_{\mathrm{m}}, \mathrm{z}_{\mathrm{m}} \rangle$. Since z_{m} are linear combination of the original predictors x_{j}, we can express the solution as

$$\hat{\beta}^{\mathrm{pcr}}(\mathrm{M}) = \sum_{\mathrm{m}=1}^{\mathrm{M}} \hat{\theta}_{\mathrm{m}} \mathrm{v}_{\mathrm{m}} \tag{5.55}$$

With M = p we get the usual least squares. Principal component analysis discards the p – M smallest eigenvalue components.

The value M is suggested by cross-validation.

Theorem 2. PCA seeks principal directions of highest variance under the unit-norm constraint

Proof. Consider an input space $\mathbf{X}$ endowed with the inner product operation $\langle \mathbf{x}_{\mathrm{i}}, \mathbf{x}_{\mathrm{j}} \rangle = \mathbf{x}_{\mathrm{i}}^{\mathrm{T}} \mathbf{x}_{\mathrm{j}}$. Let $\{\mathbf{x}_1\}, \mathbf{x}_2, \cdots, \mathbf{x}_{\mathrm{n}}$ denote a set of available data (observations) from $\mathbf{X}$. Let us call the axes that maximise the variance $\psi_1, \cdots, \psi_{\mathrm{k}}$. Consider the first axis ψ_1 and project data onto it. The mean of the projection is $\langle \psi, \bar{\mathrm{x}} \rangle$ with $\bar{\mathrm{x}} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathbf{x}_{\mathrm{i}}$. By applying the variance definition:

$$\begin{aligned}
\sigma^2 &= \mathbf{E}\left[\left(\psi_1^{\mathrm{T}}\mathbf{x} - \mathbf{E}\left[\psi^{\mathrm{T}}\mathbf{x}\right]\right)^2\right] = \\
&= \mathbf{E}\left[\left(\psi_1^{\mathrm{T}}(\mathbf{x} - \mathbf{E}\left[\mathbf{x}\right])\right)^2\right] = \\
&= \mathbf{E}\left[\left(\psi_1^{\mathrm{T}}(\mathbf{x} - \bar{\mathbf{x}})\right)^2\right] = \\
&= \mathbf{E}\left[\psi_1^{\mathrm{T}}(\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}}(\mathbf{x} - \bar{\mathbf{x}})\psi_1\right] = \\
&= \psi_1^{\mathrm{T}}\mathbf{E}\left[(\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}}(\mathbf{x} - \bar{\mathbf{x}})\right]\psi_1 = \psi_1^{\mathrm{T}}\mathrm{S}\psi_1
\end{aligned} \tag{5.56}$$

where S is the covariance matrix, i.e., the measure of the directional relationship between two random variables. The variance must be maximized. A simple way would be to make psi_1 as big as possible but that is not an

interesting solution. That is why is module is restricted: $\psi_1^{\mathrm{T}}\psi_1 = 1$. To combine the objective and constraint the Lagrange multipliers are used:

$$\begin{aligned} \mathrm{L} &= \psi_1^{\mathrm{T}}\mathrm{S}\psi_1 - \alpha_1\psi_1^{\mathrm{T}}\psi - 1 \\ \Rightarrow \frac{\partial \mathrm{L}}{\partial \psi_1} &= 2\mathrm{S}\psi - 2\alpha_1 = 0 \\ \Rightarrow \alpha_1\psi_1 &= \mathrm{S}\psi_1 \end{aligned} \tag{5.57}$$

So the pair (α_1, ψ_1) is an eigenvalue-eigenvector pair. We can define additional principal components in an incremental fashion by choosing each new direction to be that which maximizes the projected variance amongst all possible directions orthogonal to those already considered (eigenvectors are not always perpendicular). In order to do so, one might consider to remove from the covariance matrix the component corresponding to the projection of the previously found axis and maximise the covariance of the residual. Suppose at step l one found l axis: then instead of using S one will use: $\mathrm{S} - [\psi_1, \cdots, \psi_l, 0, \cdots 0]^{\mathrm{T}}\mathrm{S}$.

These are the axes that maximize the variance. Recalling that PCA is obtained through SVD and recalling the property of SVD in 1.4.1, it is clear that PCA maximize the variance. □

Eigenvectors lie in the span of data and eigenvectors associated to the largest eigenvalues provides a low-dimensional subspace:

$$
\begin{aligned}
\psi_{\mathrm{l}} &= \frac{1}{\lambda_{\mathrm{l}}}\mathrm{S}\psi_{\mathrm{l}} = \frac{1}{\lambda_{\mathrm{l}}}\left(\frac{1}{\mathrm{n}}\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)\psi_{\mathrm{l}} = \frac{1}{\mathrm{n}\lambda_{\mathrm{l}}}\left[\mathbf{x}_1, \cdots, \mathbf{x}_{\mathrm{n}}\right]\begin{bmatrix}\mathbf{x}_1^{\mathrm{T}}\\ \vdots \\ \mathbf{x}_{\mathrm{n}}^{\mathrm{T}}\end{bmatrix}\psi_{\mathrm{l}} = \\
&= \frac{1}{\mathrm{n}\lambda_{\mathrm{l}}}\begin{bmatrix} \mathrm{x}_{11}^2 + \cdots + \mathrm{x}_{\mathrm{n}1}^2 & \mathrm{x}_{11}\mathrm{x}_{12} + \cdots + \mathrm{x}_{\mathrm{n}1}\mathrm{x}_{\mathrm{n}2} & \cdots & \mathrm{x}_{11}\mathrm{x}_{1\mathrm{k}} + \cdots + \mathrm{x}_{\mathrm{n}1}\mathrm{x}_{\mathrm{n}\mathrm{k}} \\ \mathrm{x}_{12}\mathrm{x}_{11} + \cdots + \mathrm{x}_{\mathrm{n}2}\mathrm{x}_{\mathrm{n}1} & \mathrm{x}_{12}^2 + \cdots + \mathrm{x}_{\mathrm{n}2}^2 & \cdots & \mathrm{x}_{12}\mathrm{x}_{1\mathrm{k}} + \cdots + \mathrm{x}_{\mathrm{n}2}\mathrm{x}_{\mathrm{n}\mathrm{k}} \\ \vdots & \vdots & \cdots & \vdots \\ \mathrm{x}_{1\mathrm{k}}\mathrm{x}_{11} + \cdots + \mathrm{x}_{\mathrm{n}\mathrm{k}}\mathrm{x}_{\mathrm{n}1} & \mathrm{x}_{1\mathrm{k}}\mathrm{x}_{12} + \cdots + \mathrm{x}_{\mathrm{n}\mathrm{k}}\mathrm{x}_{\mathrm{n}2} & \cdots & \mathrm{x}_{1\mathrm{k}}^2 + \cdots + \mathrm{x}_{\mathrm{n}\mathrm{k}}^2 \end{bmatrix}\psi_{\mathrm{l}} = \\
&= \frac{1}{\mathrm{n}\lambda_{\mathrm{l}}}\begin{bmatrix} \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}1}^2 & \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}1}\mathrm{x}_{\mathrm{i}2} & \cdots & \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}1}\mathrm{x}_{\mathrm{i}\mathrm{k}} \\ \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}2}\mathrm{x}_{\mathrm{i}1} & \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}2}^2 & \cdots & \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}2}\mathrm{x}_{\mathrm{i}\mathrm{k}} \\ \vdots & \vdots & \cdots & \vdots \\ \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}\mathrm{k}}\mathrm{x}_{\mathrm{i}1} & \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}\mathrm{k}}\mathrm{x}_{\mathrm{i}2} & \cdots & \sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x}_{\mathrm{i}\mathrm{k}}^2 \end{bmatrix}\psi_{\mathrm{l}} = \sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\mathbf{x}_{\mathrm{i}}\mathbf{x}_{\mathrm{i}}^{\mathrm{T}}\right)\psi_{\mathrm{l}} = \\
&= \sum_{\mathrm{i}=1}^{\mathrm{n}}\left(\mathbf{x}_{\mathrm{i}}\mathbf{x}_{\mathrm{i}}^{\mathrm{T}}\psi_{\mathrm{l}}\right) = \sum_{\mathrm{i}=1}^{\mathrm{n}}\langle\mathbf{x}_{\mathrm{i}}, \psi_{\mathrm{l}}\rangle\mathbf{x}_{\mathrm{i}}
\end{aligned}
\tag{5.58}
$$

Principal components can be written as linear combinations of input data:

$$
\psi_{\mathrm{l}} = \sum_{\mathrm{i}=1}^{\mathrm{n}}\alpha_{\mathrm{i}\mathrm{l}}\mathbf{x}_{\mathrm{i}} \tag{5.59}
$$

5.11.2 Partial least squares

This technique use a set of linear combinations of y in addition to X for the construction. It is not scale invariant so each $\mathbf{x}_{\mathrm{j}}$ must be standardized.

Since PLS use y to construct its directions, its solution path is not linear in y. It seeks direction with high variance and high correlation with the response while PCR only with high variance.

5.12 Multioutput shrinkage and selection

To apply selection and shrinkage methods in the multiple output case, one could apply a univariate technique individually to each outcome or si- multaneously to all outcomes, i.e., different λ in Ridge or Lasso can be used for each output or the same value can be adopted.

standardize x_j to 0 mean and 1 variance;
$\hat{y}^{(0)} \leftarrow \bar{y}\mathbf{1}$;
$x_j^{(0)} \leftarrow x_j$;
for $m = 1, \cdots, p$ **do**
  $z_m = \sum_{j=1}^{p} \hat{\phi}_{mj} x_j^{(m-1)}$ where $\hat{\phi}_{mj} = \langle x_j^{(m-1)}, y \rangle$;
  $\hat{\theta}_m = \frac{\langle z_m, y \rangle}{\langle z_m, z_m \rangle}$;
  $\hat{y}^m = \hat{y}^{(m-1)} + \hat{\theta}_m z_m$;
  orthogonalize each x_j^{m-1} w.r.t.
    $z_m : x_j^{(m)} = x_j^{(m-1)} - \frac{\langle z_m, x_j^{(m-1)} \rangle}{\langle z_m, z_m \rangle} z_m$, $j = 1, \cdots, p$;
Output the sequence of fitted vectors $\{\hat{y}^m\}_1^p$. Since z_{l1}^m are linear in x_j, so is $\hat{y}^{(m)} = X\hat{\beta}^{pls}(m)$. These coefficients can be recovered from the sequence of PLS transformations.

5.13 Other derived algorithms

5.13.1 Incremental forward stagewise

$r \leftarrow y$;
$\beta_i = 0$ for $i \neq 0$;
find the (standardized) predictor x_j most correlated with the residual ;
$\beta_j \leftarrow \beta_j + \delta_j$ where $\delta_j = \epsilon \text{sign}\left[\langle x_j, r \rangle\right]$ and $\epsilon > 0$ small;
$r \leftarrow r - \delta_j x_j$;
Repeat the steps many times until the residuals are uncorrelated with the predictors.

5.13.2 The Dantzig selector

...

5.13.3 The Grouped Lasso

...

6 Hands on linear regression

The data are taken from

wget https://web.stanford.edu/ hastie/ElemStatLearn/datasets/prostate.data

and are the same used in the book *Elements of statistical learning*. They come from a study by Stamey et al. (1989) and examine the correlation between the level of prostate-specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy. The variables are log cancer volume (`lcavol`), log prostate weight (`lweight`), `age`, log of the amount of benign prostatic hyperplasia (`lbph`), seminal vesicle invasion (`svi`), log of capsular penetration (`lcp`), Gleason score (`gleason`), and percent of Gleason scores 4 or 5 (`pgg45`).

6.1 Analyzing data

We can read the data and analyse the correlation:

```
import numpy as np;
import pandas as pd;
import scipy;
from sklearn.preprocessing import StandardScaler;
from sklearn import preprocessing as pp;
from sklearn.pipeline import Pipeline;
df = pd.read_csv("prostate.data",header=0, delimiter="\t",);
#drop first column: it is repeating the index
df.drop([df.columns[0]],inplace=True, axis='columns');
std_scaler = StandardScaler();
y = df['lpsa'];
X = df.drop(['lpsa'],axis=1);
X_train = X[X['train']=="T"];
y_train = y[X['train']=="T"];
X_test = X[X['train']=="F"];
y_test = y[X['train']=="F"];
X_train = X_train.drop('train',axis=1)
X_scaled = pp.scale(X_train.astype('float'));
# or alternatively:
X_scaled2 = std_scaler.fit_transform(X_train.astype('float'));
#put together scaled data and output to see the correlation
df_scaled = pd.concat([pd.DataFrame(X_scaled), y_train], axis=1, );
#-1 because df has still 'train' column
df_scaled.columns = df.columns[:-1];
```

The last line will show the correlation (since pandas dataframe class has the method Dataframe.tolatex() to print the elements as in the following table, we did: Dataframe.corr().round(3).tolatex()):

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa
lcavol	1.000	0.300	0.286	0.063	0.593	0.692	0.426	0.483	0.727
lweight	0.300	1.000	0.317	0.437	0.181	0.157	0.024	0.074	0.598
age	0.286	0.317	1.000	0.287	0.129	0.173	0.366	0.276	0.345
lbph	0.063	0.437	0.287	1.000	-0.139	-0.089	0.033	-0.030	0.396
svi	0.593	0.181	0.129	-0.139	1.000	0.671	0.307	0.481	0.452
lcp	0.692	0.157	0.173	-0.089	0.671	1.000	0.476	0.663	0.435
gleason	0.426	0.024	0.366	0.033	0.307	0.476	1.000	0.757	0.332
pgg45	0.483	0.074	0.276	-0.030	0.481	0.663	0.757	1.000	0.368
lpsa	0.727	0.598	0.345	0.396	0.452	0.435	0.332	0.368	1.000

As it can be seen, since data are real-valued, it is symmetric and of course square. It shows many strong correlations. Before calculating it, data have been standardized and centred such that they have zero mean. From the correlation matrix we see that both lcavol and lcp show a strong relationship with the response lpsa, and with each other.

6.2 Fitting the ordinary least square model

We can directly calculate the coefficients by ourselves using 5.3:

```
from numpy.linalg import inv;
X_b = np.c_[np.ones((len(X_scaled), 1)), X_scaled];
print(np.mean(X_b, axis=0));
theta = inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y_train);
```

The estimated coefficients are:

intercept	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45
2.452	0.711	0.290	-0.141	0.210	0.307	-0.287	-0.021	0.275

sklearn offers the LinearRegression class to fit a linear model:

```
from sklearn.linear_model import LinearRegression;
lr = LinearRegression();
lr.fit( X_scaled, y_train);
print(lr.coef_, lr.intercept_);
```

yielding the same coefficient estimation.

We can also use the stochastic gradient estimation technique:

```
n_iter = 1000;
np.random.seed(42);
eta = 0.1
theta_gd = np.random.randn(X_b.shape[1],1);
for ii in range(n_iter):
    gradient =  -2/len(X_b)*X_b.T.dot(np.c_[y_train] - X_b.dot(theta_gd));
    theta_gd = theta_gd - eta*gradient;
print(theta_gd);
```

According to the number of iterations, in this case the coefficients might be slightly different since it is an approximated solution. For $n_iter = 1000$ one gets the same coefficients:

intercept	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45
2.452	0.711	0.29	-0.141	0.21	0.307	-0.287	-0.021	0.275

6.3 Predicting values

We can predict the values by matrix mulitplication. For example in the mean square error on the training set is calculated as:

```
from sklearn.metrics import mean_squared_error;
y_pred_man = X_b.dot(theta);
print(np.sum(np.square(y_train-y_pred_man))/len(y_pred));
print(mean_squared_error(y_train, y_pred_man));
```

where the error has been calculated both manually and using the $mean_squared_error$ sklearn function. The value is 0.4392.

With the sklearn LinearRegression class we can use the method predict:

```
from sklearn.metrics import mean_squared_error;
y_pred = lr.predict(X_scaled);
error = mean_squared_error(y_train, y_pred);
```

sklearn does not support statistical inference. To calculate the Z−scores we can create a function or we can extend the LinearRegression class. In the latter case:

```
class customLinearRegression(LinearRegression):
    def __init__(self):
        super(customLinearRegression, self).__init__();
    def z_scores(self, y, X, bias=False):
        p = len(self.coef_);
        if (bias):
            y_pred = self.predict(X[:,1:]);
            X_with_bias = X;
        else:
            y_pred = self.predict(X);
            X_with_bias = np.c_[np.ones((X.shape[0], 1)), X];
        sigma_hat_sq = np.sum(np.square(y_train-y_pred))/
            (len(y_train)-p-1);
        std_err  = np.sqrt(np.diag(inv(
            X_with_bias.T.dot(X_with_bias)))*sigma_hat_sq);
        z_scores = np.multiply(np.append(self.intercept_, self.coef_),
            1/std_err);
        return std_err, z_scores;
    def get_intercept_coef(self):
        return np.append(self.intercept_, self.coef_);

std_err_lr, z_scores_lr = lr.z_scores(y_train, X_scaled);
pd.DataFrame({"Coefficient": lr.get_intercept_coef(),
      "Std. error" : std_err_lr,
              r"$Z$ score" : z_scores_lr},
             index=pd.Index(["intercept"]).append(X_train.columns)
            ).round(2).head(10)
\label{CustomLinearRegression}
```

We get the following values:

	Coefficient	Std. error	Z score
intercept	2.45	0.09	28.18
lcavol	0.71	0.13	5.37
lweight	0.29	0.11	2.75
age	-0.14	0.10	-1.40
lbph	0.21	0.10	2.06
svi	0.31	0.12	2.47
lcp	-0.29	0.15	-1.87
gleason	-0.02	0.14	-0.15
pgg45	0.28	0.16	1.74

The predictor lcavol shows the strongest effect, with lweight and svi also strong. Notice that lcp is not significant, once lcavol is in the model (when used in a model without lcavol, lcp is strongly significant).

6.4 Subset selection

First we define to classes. One is supposed work as interface for all subset selection models:

```
class best_submodel_abstract:
    def __init__(self):
        self._standardInit();
        self.pipeline = Pipeline([
                ('Scaler', StandardScaler()),
                ('LinearRegression',  CustomLinearRegression())]);

    def get_scaler(self):
        return self.pipeline.named_steps['Scaler'];
    def get_model(self):
        return self.pipeline.named_steps['LinearRegression'];

    def _standardInit(self):
        self.X_small = None;
        self.num_features  = None;
        self.feature_names = None;
        self.best_subset_idx = None; #wrt X_with_bias_coeffs
        self.best_error    = None;
        self.all_errors    = error_sequence();
        self.skl_model     = None;

    def _removeBias(self, X, num_features, isItwithoutBias=True):
        self.num_features = num_features;
```

```
        if (not isItwithoutBias):
            print("removing bias");
            X_small = X[:,1:];
        else:
            X_small = X;
        return X_small;

    def fit(self, X, y,num_features, isItwithoutBias=True):
        return
```

The initialization consists of defining a pipeline made up by a scaler and a linear regressor as defined in **??** so that we can use the z-score metric. The initializer also calls a private method $_{s}$tandardInit() that initialize all attributes to None. Elements of the pipeline are easily accessed through the methods get$_{s}$caler and get$_{m}$odel. The model also has a $_{r}$emoveBias() method so that if input data have the column vector of 1 for the intercept, it can be removed.

The second class to stores the errors associated to given sets of features and prints them:

```
class error_sequence:
    def __init__(self, errors=tuple(), sequence=tuple()):
        self.errors = errors;
        self.names  = sequence;

    def printErrs(self):
        for err, el in zip(self.errors, self.names):
            print(str(el) + ": " + str(err));
```

Finally the best subset selection model is the following:

```
class best_submodel_sk(best_submodel_abstract):
    def __init__(self):
        super(best_submodel_sk, self).__init__();
                self.errors = {"features":list(), "errors":list()};

    def _update_best_model(self, pipeline, subset, mse, col_names):
        self.pipeline        = pipeline;
        self.best_subset     = subset;
        self.best_subset_idx = subset; #wrt X_with_bias_coeffs
        self.best_error      = mse;
```

```
        self.model            = self.get_model();
        self.scaler           = self.get_scaler();
        if col_names:
            self.feature_names = col_names;
        return

    def fit(self, X, y, num_features, isItwithoutBias=True):
        column_names = None;
        self.num_features = num_features;
        if isinstance(X, pd.DataFrame):
            column_names = list(X.columns);
            X = X.values;
        X_no_bias    = super()._removeBias(X,num_features, isItwithoutBias);
        err = np.infty;
        all_idx = [i for i in range(X_no_bias.shape[1])];
        errors = {"features":list(), "errors":list()};
        if num_features==0:
            X_small = np.ones((X_no_bias.shape[0],1));
            temp_pipe = clone(self.pipeline);
            temp_pipe.fit(X_small, y); #probably separate scaler for transform
            #print(temp_pipe.named_steps['LinearRegression'].intercept_)
            y_pred_sk  = temp_pipe.predict(X_small);
            mse_sk     = mean_squared_error(y_train,y_pred_sk);
            if column_names:
                errors["features"].append(('intercept',));
            else:
                errors["features"].append(0);
            errors['errors'].append(mse_sk);
            self._update_best_model(temp_pipe, None, mse_sk, tuple(('intercept',)));
            self.all_errors = error_sequence(tuple(errors['errors']),
                                             tuple(errors["features"]));
            return;
        elif num_features>X_no_bias.shape[1]:
            num_features=X_no_bias.shape[1];
        for ff in list(itertools.combinations(all_idx, num_features)):
            X_small = X_no_bias[:,ff];
            temp_pipe = clone(self.pipeline);
            temp_pipe.fit(X_small, y); #probably separate scaler for transform
            y_pred_sk  = temp_pipe.predict(X_small);
            mse_sk     = mean_squared_error(y_train,y_pred_sk);
            if column_names:
                errors["features"].append(X_train.columns[list(ff)].values);
            else:
                errors["features"].append(ff);
            errors['errors'].append(mse_sk);
```

```
56              self.all_errors = error_sequence(tuple(errors['errors']),
57                                               tuple(errors["features"]));
58              if mse_sk < err:
59                  err = mse_sk;
60                  if column_names:
61                      names = ["intercept"] + [column_names[ii] for ii in ff]
62                  self._update_best_model(temp_pipe, ff, mse_sk, names);
```

The initializer calls the parent initializer (the one of the class defined above). The calculation is in the fit() method, which takes input and target data and the number of features to select. If the selected number of features is 0 we have only the intercept, if it is bigger the features in input data, then it is set to the features in the input matrix. itertools.combinations($\text{all}_\text{i}\text{dx}, \text{num}_\text{f}\text{eatures}$) returns all possible combinations of sub-groups of the features having size $\text{num}_\text{e}\text{lements}$.

Note : to copy the structure of the pipeline of the instance, clone is used, otherwise it is copied by reference and when it is changed at the next iteration, also the variable to which it is assigned are changed.

The pipeline is fit to the data, outputs are predicted and the RSE is calculated. The $\text{all}_\text{e}\text{rrors}$ is updated and if this is the best error, data are stored as the best parameters in the $_\text{u}\text{pdate}_\text{b}\text{est}_\text{m}\text{odel}$ private method.

7 Bayesian inference

7.1 Introduction to Bayes' theorem

Bayes' theorem is a formula that describes how to update the probabilities of hypotheses when given evidence. It follows simply from the axioms of conditional probability, but can be used to powerfully reason about a wide range of problems involving belief updates.

Given a hypothesis H and evidence E, Bayes' theorem states that the relationship between the probability of the hypothesis Pr(H), before getting the evidence, and the probability of the hypothesis after getting the evidence Pr(H|E) is

$$\Pr(\mathrm{H}|\mathrm{E}) = \frac{\Pr(\mathrm{E}|\mathrm{H})\Pr(\mathrm{H})}{\Pr(\mathrm{E})} \tag{7.1}$$

Often there are competing hypothesis and the task is to determine which is the most probable.

This formula relates the probability of the hypothesis before getting the evidence Pr(H), to the probability of the hypothesis after getting the evidence, Pr(H|E): this term is generally what we want to know. For this reason, Pr(H) is called the **prior probability**, while Pr(H|E) is called the **posterior probability**. The factor that relates the two, $\frac{\Pr(\mathrm{E}|\mathrm{H})}{\Pr(\mathrm{E})}$, is called the **likelihood ratio** and Pr(E|H) is called **likelihood** which indicates the compatibility of the evidence with the given hypothesis. P(H|E) and Pr(E|H) are called conditional probabilities. A conditional probability is an expression of how probable one event is given that some other event occurred (a fixed value) and Bayes' theorem centers on relating different conditional probabilities.

Pr(E) is sometimes called **marginal likelihood** or model evidence this factor is the same for all the hypotheses being considered.

7.2 Bayes inference

Bayesian inference is a method of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis as more evidence or information becomes available. Bayesian inference assumes the data were generated by a model with unknown parameters. From this, it tries to come up with beliefs about the likely "true values" of the parameters of the model.

The Bayesian approach differs from the standard ("frequentist") method for inference in its use of a prior distribution to express the uncertainty present before seeing the data, and to allow the uncertainty remaining after seeing the data to be expressed in the form of a posterior distribution.

For this reason when writing down the Bayes theorem in these cases, formally also the conditional on the choice of the model should be written down:

$$\Pr(\theta|\mathbf{X}, \mathrm{M}) = \frac{\Pr(\mathbf{X}|\theta, \mathrm{M})\Pr(\theta|\mathrm{M})}{\Pr(\mathbf{X}|\mathrm{M})} \tag{7.2}$$

Here the hypothesis described in the previous paragraph is represented by a set of values for the parameters.

Once the model is stipulated, $\Pr(\mathrm{X}|\theta)$ can be evaluated for any given set of parameters: in this sense the likelihood is fixed once the model is fixed. $\Pr(\mathbf{X}|\mathrm{M})$ can be rexpressed as $\Pr(\mathbf{X}|\mathrm{M}) = \int \Pr(\mathbf{X}|\theta, \mathrm{M})\Pr(\theta|\mathrm{M})\mathrm{d}\theta$. In this way it can be re-thought as a normalization factor of the sets of parameters since it is a summation over the all parameter space. However note that it **does depend** on the choice of the model. It can be seen also as asking the question *how much is it probable to see the data we have seen given that the model* M*, without any claim about its parameters, generated those data*?

7.3 Types of estimation

It is the moment to clarify different types of estimation. We have already seen the Ordinary Least Square (OLS) estimation, and other types of estimation based on the definition of an error function.

We have also seen the **Maximum Likelihood Estimation (MLE)** approach, which maximizes the likelihood of the Bayes expression and how it is equivalent to OLS estimation in case of linear regression. (To be precise almost always the log-likelihood is maximized, first of all because of analytical convenience since many times we deal with exponentials coming from Gaussian distribution. Secondly for numerical precision: we are dealing with probabilities, numbers between 0 and 1 and since the range is quite small, underflow might be a problem.) The logarithmic instead extends the range by mapping numbers close to 0 to ∞, resulting in a better precision.

The **Maximum A-Priori (MAP)** estimation maximizes the numerator of the Bayes expression, i.e., the likelihood times the prior, which means the

likelihood is weighted by weights coming from the prior. When using a uniform distribution for the prior, MAP turns into MLE since we are assigning equal weights for each possible value. For example suppose we can assign six possible values to β and assume $\mathrm{P}(\beta_\mathrm{i}) = 1/6$:

$$\begin{aligned}\beta_{\mathrm{MAP}} &= \arg\max_{\beta} \sum_{\mathrm{i}} \Pr(\mathbf{x}_\mathrm{i}|\beta) + \log \mathrm{P}(\beta) = \\ &= \arg\max_{\beta} \sum_{\mathrm{i}} \Pr(\mathbf{x}_\mathrm{i}|\beta) + \mathrm{const} = \arg\max_{\beta} \sum_{\mathrm{i}} \Pr(\mathbf{x}_\mathrm{i}|\beta) = \beta_{\mathrm{MLE}}\end{aligned} \tag{7.3}$$

When using a different prior, i.e., the simplification does not hold anymore.

7.4 Conjugate distributions

Definition 7.1. Conjugate distributions In Bayesian probability theory, if the posterior distribution $\Pr(\theta|\mathbf{X}, \mathrm{M})$ is in the same probability space as the prior probability distribution $\Pr(\theta|\mathrm{M})$, then the prior and posterior are called **conjugate distributions**, and the prior is called **conjugate prior for the likelihood function**.

As example consider the Gaussian distribution: **the Gaussian family is conjugate to itself (or self-conjugate) with respect to a Gaussian likelihood function**. If the likelihood is Gaussian, choosing Gaussian prior will ensure that also the posterior distribution is a Gaussian (see 1.10). This means that the Gaussian distribution is a conjugate prior for the likelihood that is also Gaussian.

Consider the general problem of inferring a (continuous) distribution for a parameter θ given some datum or data x. From Bayes' theorem, the posterior distribution is equal to the product of the likelihood function $\mathrm{p}(\mathrm{x}|\theta, \mathrm{M})$ and the prior $\mathrm{p}(\theta|\mathrm{M})$. Let the likelihood function be considered fixed; the likelihood function is usually well-determined from a statement of the data-generating process. It is clear that different choices of the prior distribution $\mathrm{p}(\theta|\mathrm{M})$ may make the integral more or less difficult to calculate, and the product $\mathrm{p}(\mathrm{x}|\theta, \mathrm{M}) \times \mathrm{p}(\theta|\mathrm{M})$ may take one algebraic form or another. For certain choices of the prior, the posterior has the same algebraic form as the prior (generally with different parameter values). Such a choice is a conjugate prior. A conjugate prior is an algebraic convenience, giving a closed-form expression for the posterior; otherwise numerical integration may be necessary.

Further, conjugate priors may give intuition, by more transparently showing how a likelihood function updates a prior distribution. All members of the exponential family have conjugate priors.

In case of **classification** we are given a training set with input and output data. Once we have stipulated a model that could have generated output data, we want to find the best parameters of the model such that when those inputs are fed to the model we get those outputs. So the question becomes *what is the best set of parameters θ for the model* M *that could have generated the output* $\mathbf{y}$ *when the model has been fed with input data* $\mathbf{X}$*?*

In formula:

$$\Pr(\theta|\mathbf{y}, \mathbf{X}, \mathrm{M}) = \frac{\Pr(\mathbf{y}|\theta, \mathbf{X}, \mathrm{M})\Pr(\theta|\mathbf{X}, \mathrm{M})}{\Pr(\mathbf{y}|\mathbf{X}, \mathrm{M})} \tag{7.4}$$

7.4.1 Dataset likelihood

7.4.1 To be precise the likelihood is the likelihood of an entire dataset, since we are interested in all $\mathbf{y}$ and not in a single value y. $\Pr(\mathbf{y}|\theta, \mathbf{X}, \mathrm{M})$ is then a joint density over all the responses in our dataset: $\Pr(\mathrm{y}_1, \mathrm{y}_2, \cdots, \mathrm{y}_\mathrm{n}|\theta, \mathbf{X}, \mathrm{M})$. Evaluating this density at the observed points gives a single likelihood value for the whole dataset. Assuming that the noise at each data point is independent we can factorize as:

$$\Pr(\mathbf{y}|\theta, \mathbf{X}, \mathrm{M}) = \prod_{\mathrm{n}=1}^{\mathrm{N}} \Pr(\mathrm{y}_\mathrm{n}|\mathbf{x}_\mathrm{n}, \theta) \tag{7.5}$$

We have not say that y_n's are completely independent as otherwise it would not be worth trying to model the data at all. Rather, they are **conditionally independent** given a value θ, i.e., the deterministic model. Basically the model incorporates the dependency. Consider the following example, we have a set of data $(\mathbf{y}, \mathbf{X})$ from which we want to predict the output y_n given a new $\mathbf{x}_\mathrm{n}$. Recalling from conditional probability that $\mathrm{P}(\mathrm{A}|\mathrm{B}) = \frac{\Pr(\mathrm{A}\cap\mathrm{B})}{\Pr(\mathrm{B})}$ we have:

$$\mathrm{P}(\mathrm{y}_\mathrm{n}|\mathbf{y}, \mathbf{x}_\mathrm{n}, \mathbf{X}) = \frac{\Pr(\mathrm{y}_\mathrm{n} \cap \mathbf{y}|\mathbf{x}_\mathrm{n}, \mathbf{X})}{\Pr(\mathbf{y}|\mathbf{x}_\mathrm{n}, \mathbf{X})} \tag{7.6}$$

Note that actually $\mathbf{y}$ does not depend on $\mathbf{x}_\mathrm{n}$: $\Pr(\mathbf{y}|\mathbf{x}_\mathrm{n}, \mathbf{X}) = \Pr(\mathbf{y}|\mathbf{X})$. Using independence:

$$\mathrm{P}(\mathrm{y}_\mathrm{n}|\mathbf{y}, \mathbf{x}_\mathrm{n}, \mathbf{X}) = \frac{\Pr(\mathrm{y}_\mathrm{n} \cap \mathbf{y}|\mathbf{x}_\mathrm{n}, \mathbf{X})}{\Pr(\mathbf{y}|\mathbf{X})} = \frac{\Pr(\mathbf{y}|\mathbf{X})\Pr(\mathrm{y}_\mathrm{n}|\mathbf{x}_\mathrm{n}, \mathbf{X})}{\Pr(\mathbf{y}|\mathbf{X})} = \tag{7.7}$$

[TO BE CONTINUED WAITING ON ANSWER ON CROSS-VALIDATED STACK EXCHANGE]

8 Linear Classification

For classification problem, in this section we assume the classification boundaries are linear, i.e., in the input hyperspace the points belonging to different classes can be separated by hyperplanes.

8.1 Linear regression of an Indicator matrix

Suppose we have K classes. For a single output we build a vector $\mathbf{y} = (y_1, \cdots, y_k)$ where $y_k = 1$ if the class it belongs is the k class. As output we will have the matrix $\mathbf{Y}$ of 0 and 1 with each row having a single 1. We fit a linear regression model to each of the columns of $\mathbf{Y}$ simultaneously:

$$\hat{\mathbf{Y}} = \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

We get a coefficient vector for each response column y_k, and hence a $(p+1)\times K$ matrix $\hat{\mathbf{B}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$,where $\mathbf{X}$ will have $p+1$ columns with a leading column of 1 for the intercept.

Suppose we are given a new input x. Then the classification problem becomes:

compute the output $\hat{f}(x)^T = (1, x^T)\hat{\mathbf{B}}$ *which is a* k *vector identify the largest component* $\hat{G}(x) = \arg\max_{k\in\mathcal{G}} \hat{f}_k(x)$.

With this approach we are basically estimating a conditional expectation, i.e., given the inputs $\mathbf{x}$ what is the probability the output is of class k? Mathematically $\mathbf{E}(y_k|X = x) = \Pr(G = k|X = x)$ since $y_k = 1$.

Although the linear model guarantees $\sum_{k\in\mathcal{G}} \hat{f}_k = 1$, as long as there is an intercept in the model, $\hat{f}_k(x)$ can be negative or bigger than one, especially when making predictions outside the hull of training data. Although this fact, this approach still works in many cases.

An important limitation is when $K \geq 3$. Even if the classes can still be separated by more than one linear boundaries, linear regression cannot find linear boundaries (because they are more than one?).

Quadratic regression might solve the problem, but a general rule is that if $k \geq 3$ classes are lined up (their centroids are in the same line), a polynomial term with degree up to $k-1$ is needed to solve the problem, and since the direction is arbitrary, cross-products terms might be needed too.

8.2 Linear Discriminant analysis

For optimal classification we have to know the class posteriors $\Pr(\mathrm{G}|\mathrm{X})$. Let π_k be the prior probability of class k with $\sum_\mathrm{k}^\mathrm{K} \pi_\mathrm{k} = 1$. Suppose $\mathrm{f_k(x)}$ is the class conditional density. From the Bayes theorem (?) we get

$$\Pr(\mathrm{G} = \mathrm{k}|\mathrm{X} = \mathrm{x}) = \frac{\Pr(\mathrm{X} = \mathrm{x}|\mathrm{G} = \mathrm{k}) \Pr(\mathrm{G} = \mathrm{k})}{\Pr(\mathrm{X} = \mathrm{x})} \tag{8.1}$$

$\Pr(\mathrm{G} = \mathrm{k})$ is the prior probability π_k, $\Pr(\mathrm{X} = \mathrm{x}|\mathrm{G} = \mathrm{k})$ is the class conditional density while the denominator can be rewritten using the **Total Probability Theorem** as

$$\Pr(\mathrm{X} = \mathrm{x}) = \sum_{\mathrm{l}=1}^{\mathrm{K}} \Pr(\mathrm{X} = \mathrm{x}|\mathrm{G} = \mathrm{l}) \Pr(\mathrm{G} = \mathrm{l}) = \sum_{\mathrm{l}=1}^{\mathrm{K}} \Pr(\mathrm{X} = \mathrm{x}|\mathrm{G} = \mathrm{l}) \pi_\mathrm{l} \tag{8.2}$$

$$\Rightarrow \Pr(\mathrm{G} = \mathrm{k}|\mathrm{X} = \mathrm{x}) = \frac{\mathrm{f_k(x)} \pi_\mathrm{k}}{\sum_{\mathrm{l}=1}^{\mathrm{K}} \mathrm{f_l(x)} \pi_\mathrm{l}} \tag{8.3}$$

The goodness of classification mostly rely on $\mathrm{f_k(x)}$ and many techniques use models for class densities:

- linear and quadratic discriminant analysis use Gaussian densities;
- mixtures of Gaussians allow for non-linear boundaries;
- Naive Bayes models assume that each class density is a product of marginal densities i.e., inputs are conditionally independent in each class.

Modelling each class density as a multivariate Gaussian we have

$$\mathrm{f_k(x)} = \frac{1}{(2\pi)^{\frac{\mathrm{p}}{2}} |\sum_\mathrm{k}|^{\frac{1}{2}}} \mathrm{e}^{-\frac{1}{2}(\mathrm{x}-\mu_\mathrm{k})^\mathrm{T} \Sigma_\mathrm{k}^{-1} (\mathrm{x}-\mu_\mathrm{k})} \tag{8.4}$$

Linear Discriminant analysis assumes equal covariance matrices for all classes. Taking as comparison between two classes the log-ratio, we

have

$$\begin{aligned}
\log\frac{\Pr(\mathrm{G}=\mathrm{k}|\mathrm{X}=\mathrm{x})}{\Pr(\mathrm{G}=\mathrm{l}|\mathrm{X}=\mathrm{x})} &= \log\frac{\mathrm{f_k(x)}\pi_\mathrm{k}}{\mathrm{f_l(x)}\pi_\mathrm{l}} = \log\frac{\pi_\mathrm{k}}{\pi_\mathrm{l}} + \log\frac{\mathrm{f_k(x)}}{\mathrm{f_l(x)}} = \\
&= \log\frac{\pi_\mathrm{k}}{\pi_\mathrm{l}} + \log\frac{\mathrm{e}^{-\frac{1}{2}(\mathrm{x}-\mu_\mathrm{k})^\mathrm{T}\Sigma^{-1}(\mathrm{x}-\mu_\mathrm{k})}}{\mathrm{e}^{-\frac{1}{2}(\mathrm{x}-\mu_\mathrm{l})^\mathrm{T}\Sigma^{-1}(\mathrm{x}-\mu_\mathrm{l})}} = \\
&= \log\frac{\pi_\mathrm{k}}{\pi_\mathrm{l}} - \frac{1}{2}(\mathrm{x}-\mu_\mathrm{k})^\mathrm{T}\Sigma^{-1}(\mathrm{x}-\mu_\mathrm{k}) - \left(-\frac{1}{2}\right)(\mathrm{x}-\mu_\mathrm{l})^\mathrm{T}\Sigma^{-1}(\mathrm{x}-\mu_\mathrm{l}) = \\
&= \log\frac{\pi_\mathrm{k}}{\pi_\mathrm{l}} - \frac{1}{2}\left[(\mathrm{x}-\mu_\mathrm{k})^\mathrm{T}\Sigma^{-1}(\mathrm{x}-\mu_\mathrm{k}) + (\mathrm{x}-\mu_\mathrm{l})^\mathrm{T}\Sigma^{-1}(\mu_\mathrm{l}-\mathrm{x})\right] = \\
&= \log\frac{\pi_\mathrm{k}}{\pi_\mathrm{l}} - \frac{1}{2}\left[(2\mathrm{x}-\mu_\mathrm{k}-\mu_\mathrm{l})^\mathrm{T}\Sigma^{-1}(\mu_\mathrm{l}-\mu_\mathrm{k})\right] = \\
&= \log\frac{\pi_\mathrm{k}}{\pi_\mathrm{l}} + \frac{1}{2}\mathrm{x}^\mathrm{T}\Sigma^{-1}(\mu_\mathrm{k}-\mu_\mathrm{l}) - \frac{1}{2}(\mu_\mathrm{k}+\mu_\mathrm{l})^\mathrm{T}\Sigma^{-1}(\mu_\mathrm{k}-\mu_\mathrm{l}) =
\end{aligned} \tag{8.5}$$

which is linear in x, so all decision boundaries are linear (i.e., they are hyperplanes in p dimensions). If the common covariance matrix is spherical, i.e., $\Sigma = \sigma^2\mathrm{I}$ and the class priors are equal, each boundary that separates two classes is the perpendicular bisector of the segment joining the centroids of the two classes.

The linear discriminant functions of each class are

$$\delta_\mathrm{k}(\mathrm{x}) = \mathrm{x}^\mathrm{T}\Sigma^{-1}\mu_\mathrm{k} - \frac{1}{2}\mu_\mathrm{k}^\mathrm{T} + \log\pi_\mathrm{k} \tag{8.6}$$

We do not know the parameters of the Gaussian distribution and we must estimate them from the training data:

$$\hat{\pi}_\mathrm{k} = \frac{\mathrm{N_k}}{\mathrm{N}} \tag{8.7}$$

$$\hat{\mu}_\mathrm{k} = \sum_{\mathrm{g_i}=\mathrm{k}} \frac{\mathrm{x_i}}{\mathrm{N_k}} \tag{8.8}$$

$$\hat{\Sigma} = \sum_{\mathrm{k}=1}^{\mathrm{K}} \frac{(\mathrm{x_i}-\hat{\mu}_\mathrm{k})(\mathrm{x_i}-\hat{\mu}_\mathrm{k})^\mathrm{T}}{(\mathrm{N}-\mathrm{K})} \tag{8.9}$$

Note that LDA does not use Gaussian assumption for the features.

8.2.1 Decision rule

Consider two classes 1 and 2. LDA classifies to class 1 if

$$\mathrm{x}^{\mathrm{T}}\hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_1 - \frac{1}{2}\hat{\mu}_1^{\mathrm{T}}\hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_1 + \log\hat{\pi}_1 > \mathrm{x}^{\mathrm{T}}\hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_2 - \frac{1}{2}\hat{\mu}_2^{\mathrm{T}}\hat{\mathbf{\Sigma}}^{-1}\hat{\mu}_2 + \log\hat{\pi}_2 \quad (8.10)$$

to class 2 if $<$ holds. In such case there is a correspondence between LDA and linear regression classification if the two classes are coded with +1 and –1. In this case the coefficient vector from least squares is proportional to the LDA direction. However unless $\mathrm{N}_1 = \mathrm{N}_2$, the intercepts are different and so are the decision rules.

With more than 2 classes, linear regression is not able to classify correctly while LDA does.

8.3 Quadratic Discriminant analysis

If we do not assume equal covariance, the squared term in x does not cancel out and we get quadratic discriminant functions:

$$\delta_{\mathrm{k}}(\mathrm{x}) = -\frac{1}{2}\log|\Sigma_{\mathrm{k}}| - \frac{1}{2}\left(\mathrm{x}-\mu_{\mathrm{k}}\right)^{\mathrm{T}}\mathbf{\Sigma}_{\mathrm{k}}^{-1}\left(\mathrm{x}-\mu_{\mathrm{k}}\right) + \log\pi_{\mathrm{k}} \quad (8.11)$$

The decision boundaries between two classes are quadratic functions.

Note: QDA does not differ much from LDA applied the enlarged quadratic polynomial input space but generally QDA is preferred in this case.

The estimates are similar but the covariance matrix must be estimated for each class. When p is large, this means a dramatic increase in the number of parameters, considering we only need the differences $\delta_{\mathrm{k}}(\mathrm{x})-\delta_{\mathrm{l}}(\mathrm{x})$. LDA needs $(\mathrm{K}-1)\times(\mathrm{p}+1)$ parameters, while QDA needs $(\mathrm{K}-1)\times(\mathrm{p}\,(\mathrm{p}+3)/2+1)$.

8.4 Regularized discriminant analysis

This method shrinks the separate covariances of QDA towards a common covariance as in LDA. In a way it is similar to ridge regression. The regularized covariance matrices have the form:

$$\hat{\mathbf{\Sigma}}_{\mathrm{k}}(\alpha) = \alpha\hat{\mathbf{\Sigma}}_{\mathrm{k}} + (1-\alpha)\hat{\mathbf{\Sigma}} \quad (8.12)$$

with $\alpha \in [0,1]$, the two extremes being LDA and QDA. α can be chosen on the validation data or by cross-validation.

Similarly we can allow $\hat{\mathbf{\Sigma}}$ to be shrunk toward the scalar covariance:

$$\hat{\mathbf{\Sigma}}(\gamma) = \gamma\hat{\mathbf{\Sigma}} + (1-\gamma)\hat{\sigma}^2\mathrm{I} \tag{8.13}$$

with $\gamma \in [0,1]$ so we get a more general family of covariances

$$\hat{\mathbf{\Sigma}}_\mathrm{k}(\alpha,\gamma) = \alpha\hat{\mathbf{\Sigma}}_\mathrm{k} + (1-\alpha)\left(\gamma\hat{\mathbf{\Sigma}} + (1-\gamma)\,\hat{\sigma}^2\mathrm{I}\right) \tag{8.14}$$

8.5 Computation

Computation of LDA and QDA is simplified by diagonalizing the covariance matrices with the singular value decomposition $\hat{\mathbf{\Sigma}}_\mathrm{k} = \mathrm{U_k D_k U_k^T}$. The terms in 8.11 become

$$\begin{aligned}&(\mathrm{x}-\mu_\mathrm{k})^\mathrm{T}\,\mathbf{\Sigma}_\mathrm{k}^{-1}\,(\mathrm{x}-\mu_\mathrm{k}) = (\mathrm{x}-\mu_\mathrm{k})^\mathrm{T}\left(\mathrm{U_k D_k U_k^T}\right)^{-1}(\mathrm{x}-\mu_\mathrm{k}) = \\ &= (\mathrm{x}-\mu_\mathrm{k})^\mathrm{T}\,\mathrm{U_k D_k^{-1} U_k^T}\,(\mathrm{x}-\mu_\mathrm{k}) = \left[\mathrm{U_k^T}\,(\mathrm{x}-\mu_\mathrm{k})\right]^\mathrm{T}\mathrm{D_k^{-1}}\left[\mathrm{U_k^T}\,(\mathrm{x}-\mu_\mathrm{k})\right]\end{aligned} \tag{8.15}$$

$$\log|\hat{\mathbf{\Sigma}}_\mathrm{k}| = \sum_\mathrm{l}\log \mathrm{d_{kl}} \tag{8.16}$$

Considering the above steps, LDA classifier can be seen as performing the following steps:

- sphere the data w.r.t. the common covariance estimate: $\mathrm{X}* \leftarrow \mathrm{D}^{-\frac{1}{2}}\mathrm{U}^\mathrm{T}\mathrm{X}$. The common covariance estimate fof X$*$ will now be the identity.
- Classify to the closest centroid in the transformed space, modulo the effect of the class prior probability π_k(?)

8.6 Regularized-rank linear discriminant analysis

Consider K = 2 with two centroids and the input is 2, i.e., input points are on a plane. Given an input point, for classification purposes what matters is not the distance in the p space of such point from the two centroids but rather the distance from the two centroids of the projection of this point on the line joining them (8.1). So basically instead of using the 2 dimensions, we are calculating the distance in one dimension, a line. If K = 3 then the points are projected onto a plane (2d), of course in this case it is convenient if p > 2.

More generally the K centroids in p-dimensional input, lie in an affine subspace of dimensions $\leq K-1$ and if p > is much larger than K this will be a considerable drop in dimension.

If $K > 3$ we can look for a $L < K-1$ dimensional subspace optimal for LDA. Fisher defined *optimal* such that the projected centroids were spread out as much as possible in terms of variance. This problem, finding the principal component subspaces of the centroids, involves the following steps:

- compute the $k \times p$ matrix of class centroids M and the common covariance matrix W for within-class covariance;
- compute $M* = MW^{-\frac{1}{2}}$ using the eigen-value decomposition;
- compute B$*$, the covariance matrix of M$*$ (B for between class covariance) and its eigenvalue decomposition $B^* = V^* D_B V^{*T}$. The columns v_l^* from the first to the last define the coordinates of optimal subspaces.

The lth discriminant variable is given by $Z_l = v_l^T X$ with $v_l = W^{-\frac{1}{2}} v_l^*$. Although the direction joining the centroids separates the means as much as possible(maximizes the between class covariance), there is an overlap between the projected classes due to the nature of covariances. Taking the covariances into account reduce the overlap and that is what we are doing (8.2).

The between-class variance Z is $a^T Ba$ and the within class variance is $a^T Wa$, with $B + W = T$, the total covariance matrix of X. Fisher's problem maximizes the *Rayleigh quotient*:

$$\max_a \frac{a^T Ba}{a^T Wa} \tag{8.17}$$

This is a generalized eigenvalue problem, with a given by the largest eigenvalue. Similarly one can find the next direction a_2, orthogonal in W to a_1, such that $a_2^T Ba_2 / a_2^T Wa_2$ is maximized; the solution is $a_2 = v_2$, and so on. a_l are the *discriminant coordinates* or *canonical variates*, different from discriminant functions.

The reduced subspaces can be used both for visualization and classification by limiting the distance between centroids to the chosen subspace. However, when doing this, due to the Gaussian classification, a correction factor of $\log \pi_k$ is needed. The misclassification is given by the overlapping area in 8.2 between the two densities. When both classes have the same

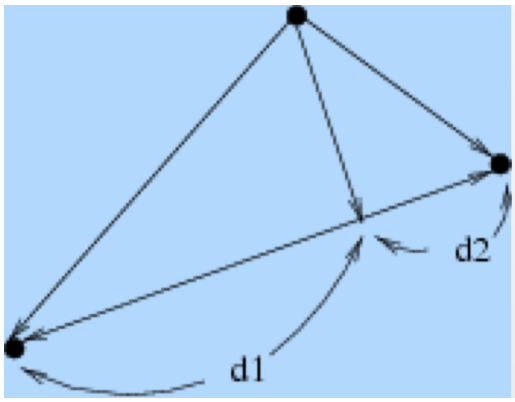


Figure 8.1: ...

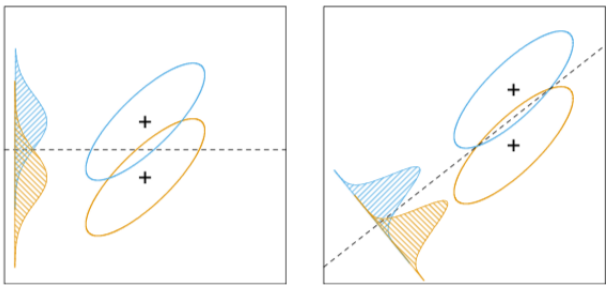


Figure 8.2: Covariance.

priors π_k as in the figure, the optimal cut-point is the midway between projected means, if not the cut-point is moved towards the smaller class to have a better error rate.

For 2 classes one can derive the linear rule using LDA, and then choosing the cut-point to minimize misclassification error.

8.7 Logistic regression

The idea behind logistic regression is to still exploit a linear model $x^T\beta$ but having its output representing a probability, i.e., constrained between 0 and 1. This part is performed using the sigmoid function

$$p = \sigma(q) = \frac{1}{1 + e^{-q}} \tag{8.18}$$

Inverting the terms we get

$$q = -\log \frac{1-p}{p} = \log \frac{p}{1-p} \tag{8.19}$$

8.19 is called **logit function**. As q increases to ∞, the output of the sigmoid gets closer to 1; instead when it diverges to $-\infty$ we get 0. Suppose we have just 2 classes or equivalently a binary classifier that tells the probability of an event to happen: $Y_n = 1$ when the event happens and $Y_n = 0$ when it does not. We can express the probabilities output by our classifier when new input data x_{new} are observed as:

$$\begin{aligned} P(G = 1|x_n, \beta) &= \frac{1}{1 + e^{-\beta^T \mathbf{x}_{new}}} \\ P(G = 0|x_n, \beta) &= 1 - \frac{1}{1 + e^{-\beta^T \mathbf{x}_{new}}} = \frac{e^{-\beta^T \mathbf{x}_{new}}}{1 + e^{-\beta^T \mathbf{x}_{new}}} \end{aligned} \tag{8.20}$$

These equations can be combined in a single equation:

$$P(G = g|x_n, \beta) = P(G = 1|x_n, \beta)^g P(G = 0|x_n, \beta)^{g-1} \tag{8.21}$$

Taking the log-ratio between the two probabilities we have:

$$\log \frac{P(Y_n = 0|x_n, \beta)}{P(Y_n = 1|x_n, \beta)} = \log \frac{\frac{e^{-\beta^T \mathbf{x}_{new}}}{\cancel{1+e^{-\beta^T \mathbf{x}_{new}}}}}{\frac{1}{\cancel{1+e^{-\beta^T \mathbf{x}_{new}}}}} = -\beta^T \mathbf{x}_{new} \tag{8.22}$$

So we are using lines (or hyperplanes) to separate the two classes.

8.7.1 Multinomial logistic regression: more than 2 classes

Now suppose we have more than two classes and we still want to separate those classes with linear functions, which means we will have a hyperplane separating two classes. We have K possible outcomes. We can think to run $\mathrm{K}-1$ independent binary logistic regressions with one class chosen as **pivot**, generally the one corresponding to class K, and with the other $\mathrm{K}-1$ classes separately regressed against the pivot:

$$\begin{aligned}
\log \frac{\Pr(\mathrm{G}=1|\mathrm{X}=\mathrm{x})}{\Pr(\mathrm{G}=\mathrm{K}|\mathrm{X}=\mathrm{x})} &= \beta_1^{\mathrm{T}}\mathrm{x} \\
\log \frac{\Pr(\mathrm{G}=2|\mathrm{X}=\mathrm{x})}{\Pr(\mathrm{G}=\mathrm{K}|\mathrm{X}=\mathrm{x})} &= \beta_2^{\mathrm{T}}\mathrm{x} \\
&\vdots \\
\log \frac{\Pr(\mathrm{G}=\mathrm{K}-1|\mathrm{X}=\mathrm{x})}{\Pr(\mathrm{G}=\mathrm{K}|\mathrm{X}=\mathrm{x})} &= \beta_{(\mathrm{K}-1)}^{\mathrm{T}}\mathrm{x}
\end{aligned} \tag{8.23}$$

The choice of the pivot class used as denominator is arbitrarily and the estimates are equivalent under different choices.

Summing the probability of each class we must get 1:

$$\begin{aligned}
&\sum_{\mathrm{l}=1}^{\mathrm{K}} \Pr(\mathrm{G}=\mathrm{l}|\mathrm{X}=\mathrm{x}) = 1 \Rightarrow \Pr(\mathrm{G}=\mathrm{K}|\mathrm{X}=\mathrm{x}) + \sum_{\mathrm{l}=1}^{\mathrm{K}-1} \Pr(\mathrm{G}=\mathrm{l}|\mathrm{X}=\mathrm{x}) = 1 \\
&\Rightarrow \Pr(\mathrm{G}=\mathrm{K}|\mathrm{X}=\mathrm{x}) + \sum_{\mathrm{l}=1}^{\mathrm{K}-1} \Pr(\mathrm{G}=\mathrm{K}|\mathrm{X}=\mathrm{x})\, \mathrm{e}^{\beta_{\mathrm{l}}^{\mathrm{T}}\mathrm{x}} = 1 \\
&\Rightarrow \Pr(\mathrm{G}=\mathrm{K}|\mathrm{X}=\mathrm{x}) = \frac{1}{1+\sum_{\mathrm{l}=1}^{\mathrm{K}-1} \mathrm{e}^{\beta_{\mathrm{l}}^{\mathrm{T}}\mathrm{x}}}
\end{aligned} \tag{8.24}$$

So we can re-express the probabilities as:

$$\Pr(\mathrm{G}=\mathrm{k}|\mathrm{X}=\mathrm{x}) = \frac{\mathrm{e}^{\beta_{\mathrm{k}}^{\mathrm{T}}\mathrm{x}}}{1+\sum_{\mathrm{l}=1}^{\mathrm{K}-1} \mathrm{e}^{\beta_{\mathrm{l}}^{\mathrm{T}}\mathrm{x}}} \tag{8.25}$$

Softmax function We do not have anymore the sigmoid function, instead we have used another function named **softmax** that as the sigmoid takes

any real value as input and outputs a value between 0 and 1. The difference mostly relies on the denominator used for normalization factor since the sigmoid is a 2D curve while the softmax can have higher dimensionality (for 2D it corresponds to the sigmoid).

8.7.2 Fitting logistic regression

From now on we will consider a logistic regression for just two classes.

As seen in 7.4, we want to find the best parameters for the chosen model according to some criteria. First let us apply the Bayes theorem:

$$\Pr(\beta|\mathbf{y}, \mathbf{X}, \mathrm{M}) = \frac{\Pr(\mathbf{y}|\beta, \mathbf{X}, \mathrm{M}) \Pr(\beta, \mathrm{M})}{\Pr(\mathbf{y}|\mathbf{X}, \mathrm{M})} \tag{8.26}$$

From now on we will not write the conditional on the model for seek of brevity but we know it exists.

This formula tells we want to find the coefficients given some input and output data. Let us analyse each term:

- $\Pr(\beta)$ (**prior distribution**): this is the prior belief about the parameters without seeing the data. As prior, we will use a Gaussian distribution with 0 mean: $\mathcal{N}(0, \sigma^2\mathbf{I})$. In this way the parameters will depend on σ^2: more formally we should write $\Pr(\beta|\sigma)$ but we will skip. Although the choice of the Gaussian distribution with 0 mean is analytically convenient as seen in 7.4, we will not rely on conjugacy.

- $\Pr(\mathbf{y}|\beta, \mathbf{X})$ (**likelihood**): we will assume y are conditionally independent (7.4.1):

  $$\Pr(\mathbf{y}|\mathbf{X}, \beta) = \prod_{\mathrm{n}=1}^{\mathrm{N}} \Pr(\mathrm{y_n}|\mathbf{x_n}, \beta) \tag{8.27}$$

  Since in this case we have a binary variable, instead of a Gaussian random variable, suitable for real values distribution, we will consider a binary random variable $\mathrm{Y_n}$ characterised by the probability of the second class:

  $$\Pr(\mathbf{y}|\mathbf{X}, \beta) = \prod_{\mathrm{n}=1}^{\mathrm{N}} \Pr(\mathrm{Y_n} = \mathrm{y_n}|\mathbf{x_n}, \beta) \tag{8.28}$$

- $\Pr(\mathbf{y}|\mathbf{X})$(**marginal likelihood**): It can be expressed as

$$\Pr(\mathbf{y}|\mathbf{X}) = \int \Pr(\mathbf{y}|\beta, \mathbf{X}) \Pr(\beta) \mathrm{d}\beta \tag{8.29}$$

So the numerator can be calculated and results in a Gaussian function not in standard form (1.10). The denominator, or marginal likelihood, as we have seen can be expressed as $\Pr(\mathbf{y}|\mathbf{X}) = \int \Pr(\mathbf{y}|\beta, \mathbf{X}) \Pr(\beta) \mathrm{d}\beta$. Mathematically this integral has no close solution since (again see 1.10) and the impossibility of the integral of $\mathrm{e}^{-\mathrm{x}^2}$), unless an ad-hoc prior distribution is chosen.

When we cannot directly compute the posterior density (due to the denominator), we have three options:

1. find the single value β that corresponds to the highest value of the posterior. This is equivalent to find the value β that maximize the numerator since the denominator is not a function of β but a numerical value.

2. Approximate $\Pr(\beta|\mathbf{X}, \mathbf{y})$ with some other density that we can compute analytically.

3. Sample directly from the posterior $\Pr(\beta|\mathbf{X}, \mathbf{y})$ knowing only the numerator.

The first method is simple and hence popular but it is not very "Bayesian", since we will make predictions of new data based on a single value and not a distribution. The whole Bayesian theory is based on making an hypothesis and getting feedback from the data that can validate or not that hypothesis. Also when using a distribution we get also measures on how confident we are about the estimated model. For example when estimating a Gaussian distribution, we estimate the coefficients μ and σ^2: the smallest the latter value the more confident we are about the model. When using a single value we loose this piece of information.

With the second method we get a density easy to work with but if the chosen density is very different from the posterior our model will not be very reliable.

The third method samples from the posterior and hence to get a good approximation but it can be difficult.

8.7.3 First method: point estimation, the MAP solution

We have seen that in some cases we cannot compute the posterior but we can compute the numerator of the expression, i.e., the prior multiplied by the likelihood. The value that maximizes the posterior is also the value that maximizes the numerator. We have already seen a likelihood maximization procedure in **??**; here we are maximizing the likelihood times the prior. This solution is the **maximum a posteriori (MAP)** estimate.

The Newton-Raphson method The Newton-Raphson method finds the points where $f(x) = 0$. Starting from an estimation x_n of such points, the estimation is updated by moving to the point where the tangent to the function at x_n passes throw the x-axis. This point is computed by approximating the gradient as a change in $f(x)$ divided by a change in x:

$$\begin{aligned} f'(x_n) &= -\frac{f(x_n) - 0}{x_n - x_{n+1}} \\ \Rightarrow x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \end{aligned} \tag{8.30}$$

Instead of finding the point for which $f(x) = 0$, we want to find the point where its derivative is 0. Hence we substitute $f(x)$ with $f'(x)$ and $f'(x)$ with $f''(x)$. In this way the method can be used to find points where the gradient passes through 0, i.e., minima, maxima and points of inflections:

$$\begin{aligned} f''(x_n) &= -\frac{f'(x_n) - 0}{x_n - x_{n+1}} \\ \Rightarrow x_{n+1} &= x_n - \frac{f'(x_n)}{f''(x_n)} \end{aligned} \tag{8.31}$$

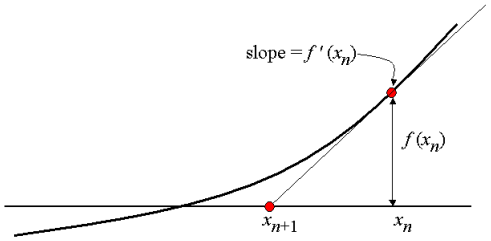


Figure 8.3: Newton-Raphsone example.

When dealing with vectors, $f'(x)$ is replaced by the vector of partial derivatives evaluated at x_n and $\frac{1}{f''(x_n)}$ is replaced by the inverse of the Hessian matrix $\frac{-\partial^2 f(x)}{\partial \mathbf{x} \partial \mathbf{x}^T}$.

Derivation As already done, instead of maximizing the function itself, we maximize its logarithmic. Apart from the mathematical convenience this is also the advantage of avoiding underflow: we are dealing with probabilities, numbers between 0 and 1 and these might be too small to have a sufficient numerical precision. As the numbers go to 0,the logarithmic makes the number goes to infinity, guaranteeing a better numerical precision. Since we cannot compute the derivative and set it to 0, we use the Newton-Raphson procedure. Let us call the numerator $g(\beta, \mathbf{X}, \mathbf{y}) = \Pr(\mathbf{y}|\beta, \mathbf{X})\Pr(\beta|\mathbf{X})$:

$$\beta' = \beta - \left(\frac{\partial^2 \log g(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta \partial \beta^T} \right)^{-1} \frac{\partial \log g(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta} \tag{8.32}$$

The procedure is iterative and stops when the gradient is 0. To check the point we have converged to corresponds to a minimum we check the Hessian matrix is negative definite.

Before computing the derivatives we re-express $g(\beta, \mathbf{X}, \mathbf{y})$.

$$\begin{aligned} &\log g(\beta, \mathbf{X}, \mathbf{y}) = \sum_{n=1}^{N} \log \Pr(Y_n = y_n|\mathbf{x}_n, \beta) + \log \Pr(\beta|\sigma^2) = \\ &= \sum_{n=1}^{N} \log \left[\left(\frac{1}{1 + e^{-\beta^T \mathbf{x}_n}} \right)^{y_n} \left(\frac{e^{-\beta^T \mathbf{x}_n}}{1 + e^{-\beta^T \mathbf{x}_n}} \right)^{1-y_n} \right] + \log \Pr(\beta|\sigma^2) = \end{aligned} \tag{8.33}$$

We denote $P_n = P(Y_n = 1|\beta, \mathbf{x}_n) = \frac{1}{1+e^{-\beta^T \mathbf{x}_n}}$:

$$\log g(\beta, \mathbf{X}, \mathbf{y}) = \sum_{n=1}^{N} \log P_n^{y_n} + \log(1 - P_n)^{1-y_n} + \log \Pr(\beta|\sigma^2) = \tag{8.34}$$

Racalling

$$\begin{aligned}\Pr(\beta|\sigma^2) =& \frac{1}{\sqrt{(2\pi)^{\mathrm{D}}|\Sigma|}}\mathrm{e}^{-\frac{(\beta-\mu)^{\mathrm{T}}(\beta-\mu)}{2\sigma 2}} \\ \Rightarrow \log \mathrm{g}(\beta, \mathbf{X}, \mathbf{y}) =& \sum_{\mathrm{n}=1}^{\mathrm{N}} \log \mathrm{P}_{\mathrm{n}}^{\mathrm{y_n}} + \log(1-\mathrm{P_n})^{1-\mathrm{y_n}} - \frac{\mathrm{D}}{2}\log(2\pi) - \mathrm{D}\log\sigma + \\ & - \frac{1}{2\sigma^2}\beta^{\mathrm{T}}\beta\end{aligned} \tag{8.35}$$

Now we can take the derivative:

$$\begin{aligned}\frac{\partial \log \mathrm{g}(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta} =& -\frac{1}{\sigma^2}\beta + \sum_{\mathrm{n}=1}^{\mathrm{N}} \frac{\mathrm{y_n}}{\mathrm{P_n}}\frac{\partial \mathrm{P_n}}{\partial \beta} + \frac{1-\mathrm{y_n}}{1-\mathrm{P_n}}\frac{\partial(1-\mathrm{P_n})}{\partial \beta} = \\ =& \frac{\partial \log \mathrm{g}(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta} = -\frac{1}{\sigma^2}\beta + \sum_{\mathrm{n}=1}^{\mathrm{N}} \frac{\mathrm{y_n}}{\mathrm{P_n}}\frac{\partial \mathrm{P_n}}{\partial \beta} - \frac{1-\mathrm{y_n}}{1-\mathrm{P_n}}\frac{\partial \mathrm{P_n}}{\partial \beta}\end{aligned} \tag{8.36}$$

Now we must calculate $\frac{\partial \mathrm{P_n}}{\partial \beta}$:

$$\begin{aligned}\frac{\partial \mathrm{P_n}}{\partial \beta} =& \frac{\partial}{\partial \beta}\frac{1}{1+\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}} = \frac{\partial}{\partial \beta}\left(1+\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}\right)^{-1} = \\ =& \frac{1}{\left(1+\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}\right)^2}\frac{\partial}{\partial \beta}\left(1+\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}\right) = \\ =& -\mathbf{x_n}\frac{\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}}{\left(1+\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}\right)^2} = -\mathrm{x_n}\frac{1}{\left(1+\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}\right)}\frac{\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}}{\left(1+\mathrm{e}^{-\beta^{\mathrm{T}}\mathbf{x_n}}\right)} = \\ =& -\mathbf{x_n}\mathrm{P_n}(1-\mathrm{P_n})\end{aligned} \tag{8.37}$$

Substituting in 8.36:

$$\begin{aligned}\frac{\partial \log \mathrm{g}(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta} &= -\frac{1}{\sigma^2}\beta + \sum_{\mathrm{n}=1}^{\mathrm{N}} \frac{\mathrm{y_n}}{\mathrm{P_n}} \frac{\partial \mathrm{P_n}}{\partial \beta} - \frac{1-\mathrm{y_n}}{1-\mathrm{P_n}} \frac{\partial \mathrm{P_n}}{\partial \beta} = \\ &= -\frac{1}{\sigma^2}\beta + \sum_{\mathrm{n}=1}^{\mathrm{N}} \frac{\mathrm{y_n}}{\mathrm{P_n}} \left[-\mathbf{x_n}\mathrm{P_n}(1-\mathrm{P_n})\right] + \\ &\quad - \frac{1-\mathrm{y_n}}{1-\mathrm{P_n}} \left[-\mathbf{x_n}\mathrm{P_n}(1-\mathrm{P_n})\right] = \\ &= -\frac{1}{\sigma^2}\beta + \sum_{\mathrm{n}=1}^{\mathrm{N}} -\mathrm{y_n}\mathbf{x_n}(1-\mathrm{P_n}) - (1-\mathrm{y_n})(-\mathbf{x_n}\mathrm{P_n}) = \\ &= -\frac{1}{\sigma^2}\beta + \sum_{\mathrm{n}=1}^{\mathrm{N}} \mathbf{x_n}(\mathrm{y_n}-\mathrm{P_n})\end{aligned} \tag{8.38}$$

Now we must compute the Hessian matrix:

$$\begin{aligned}&\frac{\partial^2 \log \mathrm{g}(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta \partial \beta^{\mathrm{T}}} = \frac{\partial}{\partial \beta^{\mathrm{T}}} \left(-\frac{1}{\sigma^2}\beta + \sum_{\mathrm{n}=1}^{\mathrm{N}} \mathbf{x_n}(\mathrm{y_n}-\mathrm{P_n}) \right) = \\ &= -\frac{1}{\sigma^2}\mathbf{I} - \sum_{\mathrm{n}=1}^{\mathrm{N}} \mathbf{x_n} \frac{\partial \mathrm{P_n}}{\partial \beta^{\mathrm{T}}} = -\frac{1}{\sigma^2}\mathbf{I} - \sum_{\mathrm{n}=1}^{\mathrm{N}} \mathbf{x_n}\mathbf{x_n^{\mathrm{T}}}\mathrm{P_n}(1-\mathrm{P_n})\end{aligned} \tag{8.39}$$

where we used the result from 8.37.

Note that the 2 terms are negative definite, hence the Hessian is always negative definite. Therefore there can only be one optimum and it must be the minimum.

The decision boundary is the one for which $\Pr(\mathrm{Y_n} = 1|\mathbf{x}, \hat{\beta} = 0.5)$.

The steps above can be done for any prior and likelihood combination. In some cases the posterior might have several maxima and or some minima and it become difficult to know if the maximum is a global optimum.

The *Elements of Statistical Learning* book instead of maximizing posterior (or equivalently the numerator), it maximizes the likelihood. Let us express $\Pr(\mathrm{G} = \mathrm{k}|\mathrm{X} = \mathrm{x}) = \mathrm{p_k}(\mathrm{x}, \theta)$ with $\theta = \left\{\beta_{10}, \beta_1^{\mathrm{T}}, \cdots, \beta_{(\mathrm{K}-1)0}, \beta_{\mathrm{K}-1}^{\mathrm{T}}\right\}$

and let us define the log-likelihood

$$\ell(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i, \theta) \tag{8.40}$$

Consider just two classes with responses 0, 1 and let $p_1(x, \theta) = p(x, \theta)$ and $p_2(x, \theta) = 1 - p(x, \theta)$. Recall that having two classes $\sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_l^T x_i} = e^{\beta_{10} + \beta_1^T x_i}$. The log-likelihood can be written as:

$$\begin{aligned}
\ell(\beta) &= \sum_{i=1}^{N} \{y_i \log p(x_i, \beta) + (1 - y_i) \log(1 - p(x_i, \beta))\} = \\
&= \sum_{i=1}^{N} y_i \log \frac{e^{\beta_{10} + \beta_1^T x_i}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_l^T x_i}} + \\
&\quad + \sum_{i=1}^{N} (1 - y_i) \log \left(1 - \frac{e^{\beta_{10} + \beta_1^T x_i}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0} + \beta_l^T x_i}}\right) = \\
&= \sum_{i=1}^{N} y_i \log \frac{e^{\beta_{10} + \beta_1^T x_i}}{1 + e^{\beta_{10} + \beta_1^T x_i}} + \\
&\quad + \sum_{i=1}^{N} (1 - y_i) \log \left(1 - \frac{e^{\beta_{10} + \beta_1^T x_i}}{1 + e^{\beta_{10} + \beta_1^T x_i}}\right) = \\
&= \sum_{i=1}^{N} y_i \left(\log e^{\beta_{10} + \beta_1^T x_i} - \log\left(1 + e^{\beta_{10} + \beta_1^T x_i}\right)\right) + \\
&\quad + \sum_{i=1}^{N} (1 - y_i) \log \frac{1}{1 + e^{\beta_{10} + \beta_1^T x_i}} = \\
&= \sum_{i=1}^{N} y_i \left(\log e^{\beta_{10} + \beta_1^T x_i} - \log\left(1 + e^{\beta_{10} + \beta_1^T x_i}\right)\right) + \\
&\quad - \sum_{i=1}^{N} (1 - y_i) \log\left(1 + e^{\beta_{10} + \beta_1^T x_i}\right) = \\
&= \sum_{i=1}^{N} \left\{y_i \left(\beta_{10} + \beta_1^T x_i\right) - \log\left(1 + e^{\beta_{10} + \beta_1^T x_i}\right)\right\}
\end{aligned} \tag{8.41}$$

Here $\beta = [\beta_{10}, \beta_1]$. To maximize the log-likelihood we set the derivative to 0:

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{x_i}(\mathrm{y_i} - \mathrm{p}(\mathrm{x_i}, \beta) = 0 \tag{8.42}$$

Using the *Newton-Raphson* algorithm that requires the second-derivative:

$$\frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^{\mathrm{T}}} = -\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{x_i} \mathrm{x_i^T} \mathrm{p}(\mathrm{x_i}, \beta)(1 - \mathrm{p}(\mathrm{x_i}, \beta) \tag{8.43}$$

$$\Rightarrow \beta^{\mathrm{new}} = \beta^{\mathrm{old}} - \left(\frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^{\mathrm{T}}} \right)^{-1} \frac{\partial \ell(\beta)}{\partial \beta} \tag{8.44}$$

Using the matrix notation,

$$\frac{\partial \ell(\beta)}{\partial \beta} = \mathrm{X}^{\mathrm{T}}(\mathrm{y} - \mathrm{p}) \tag{8.45}$$

$$\frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^{\mathrm{T}}} = -\mathrm{X}^{\mathrm{T}}\mathrm{W}\mathrm{X} \tag{8.46}$$

So the Newton step becomes

$$\begin{aligned} \beta^{\mathrm{new}} &= \beta^{\mathrm{old}} + (\mathrm{X}^{\mathrm{T}}\mathrm{W}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}(\mathrm{y} - \mathrm{p}) = \\ &= (\mathrm{X}^{\mathrm{T}}\mathrm{W}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{W}\left(\mathrm{X}\beta^{\mathrm{old}} + \mathrm{W}^{-1}(\mathrm{y} - \mathrm{p})\right) = \\ &= \left(\mathrm{X}^{\mathrm{T}}\mathrm{W}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{W}\mathrm{z} \end{aligned} \tag{8.47}$$

with

$$\mathrm{z} = \mathrm{X}\beta^{\mathrm{old}} + \mathrm{W}^{-1}(\mathrm{y} - \mathrm{p}) \tag{8.48}$$

sometimes known as the adjusted response. This algorithm is known as **iteratively reweighted least squares (IRLS)** since each iteration solves the weighted least square problem:

$$\beta^{\mathrm{new}} \leftarrow \arg\min_{\beta} (\mathrm{z} - \mathrm{X}\beta)^{\mathrm{T}} \mathrm{W} (\mathrm{z} - \mathrm{X}\beta) \tag{8.49}$$

$\beta = 0$ seems a good starting value. Convergence is never guaranteed but typically the algorithm does converge, since the log-likelihood is concave but

overshooting can occur. In the rare cases that the log-likelihood decreases, step size halving will guarantee convergence.

For $K > 2$ we still use the iteration procedure but we will have a $K-1$ vector response and a non-diagonal weight matrix per observation. In this case it is better to work with the vector θ directly.

8.7.4 Second method: Laplace approximation

There are many approximation methods but the most common is the Laplace approximation. The idea is to approximate the density of interest with a Gaussian.

Choosing a Gaussian means choosing a proper variance and mean value. The Laplace approximation method fixes one of the two parameters, i.e., the mean to the value maximizing the posterior, β. We approximate $\log g(\beta, \mathbf{X}, \mathbf{y})$ with the Taylor expansion around $\hat{\beta}$:

$$\begin{aligned}\log g(\beta, \mathbf{X}, \mathbf{y}) \approx & \log g(\hat{\beta}, \mathbf{X}, \mathbf{y}) + \cancel{\frac{\partial \log g(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta}\bigg|_{\hat{\beta}} \left(\beta - \hat{\beta}\right)} + \\ & + \frac{\partial^2 \log g(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta^2}\bigg|_{\hat{\beta}} \left(\beta - \hat{\beta}\right)^2 = \\ & = \log g(\hat{\beta}, \mathbf{X}, \mathbf{y}) + \frac{\partial^2 \log g(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta^2}\bigg|_{\hat{\beta}} \frac{\left(\beta - \hat{\beta}\right)^2}{2}\end{aligned} \tag{8.50}$$

where the second term is the gradient evaluated at the maximum point that therefore must be 0.

The Gaussian density and its log are the following:

$$\begin{aligned}& \frac{1}{2\pi} e^{-\frac{(\beta-\hat{\beta})^2}{2\sigma^2}} \\ \Rightarrow & \log \text{const} - \frac{1}{2\sigma^2}(\beta - \mu)^2\end{aligned} \tag{8.51}$$

which is similar to 8.50. By analogy of the two equations:

$$\begin{aligned}\mu &= \hat{\beta} \\ -\frac{1}{\sigma^2} &= -\frac{\partial^2 \log g(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta^2}\bigg|_{\hat{\beta}}\end{aligned} \tag{8.52}$$

It can be applied also to multivariate densities $\Pr(\beta, \mathbf{X}, \mathbf{y}) \approx \mathcal{N}(\mu, \mathbf{\Sigma})$:

$$\begin{aligned} \mu &= \hat{\beta} \\ \Sigma^{-1} &= -\frac{\partial^2 \log \mathrm{g}(\beta, \mathbf{X}, \mathbf{y})}{\partial \beta \partial \beta^{\mathrm{T}}}\bigg|_{\hat{\beta}} \end{aligned} \tag{8.53}$$

Consider a multinomial 2D Gaussian function (dimensionaity of β is 2) as in 8.4 and suppose to project on the plane the points of the curve having the same value (the ellipses in the figure). The ellipses will be the combination of coefficients giving the same function value.

In figure 8.5 the Laplace approximation of the posterior (darker lines) is shown while the lighter lines are the true unnormalised posterior. The centre point corresponds to the maximum point $\hat{\beta}$. Note how the approximation is good around $\hat{\beta}$ and it diverges going further from it.

We use the approximate posterior to compute predictions. But now we have a density and not a single value: the prediction is computed by averaging over this density: it is like averaging over all possible values of β. We should calculate the expected value $\Pr(\mathrm{Y}_{\mathrm{new}} = 1|\mathbf{x}_{\mathrm{new}}, \beta)$ with respect to the approximate posterior denoted as $\mathcal{N}(\mu, \mathbf{\Sigma})$:

$$\Pr(\mathrm{Y}_{\mathrm{new}} = 1|\mathbf{x}_{\mathrm{new}}, \mathbf{X}, \beta) = \mathbf{E}_{\mathcal{N}(\mu, \mathbf{\Sigma})} \left\{ \Pr(\mathrm{T}_{\mathrm{new}} = 1|\mathbf{x}_{\mathrm{new}}, \beta) \right\} \tag{8.54}$$

However, we cannot compute the integral (of the expectation), but we can sample from $\mathcal{N}(\mu, \mathbf{\Sigma})$ and approximate the expectation with a sum:

$$\Pr(\mathrm{Y}_{\mathrm{new}} = 1|\mathbf{x}_{\mathrm{new}}, \mathbf{X}, \beta) = \frac{1}{\mathrm{N_s}} \sum_{\mathrm{s}=1}^{\mathrm{N_s}} \frac{1}{1 + \mathrm{e}^{-\beta_{\mathrm{s}}^{\mathrm{T}} \mathbf{x}_{\mathrm{new}}}} \tag{8.55}$$

Comparison between the decision boundaries of MAP and Laplace method. In case of MAP estimation we get a single separating line as in 8.6a for the example of two classes. In case of Laplace approximation we don't have a single separating line but a density distribution. 8.6b shows 20 boundaries (i.e., set of coefficients) sampled randomly from the distribution. All or almost all separates the classes quite well but in the area from the graph further from the classes (or from the centroids or clusters) there is a lot of variability. This variability represents the uncertainty of the classifier in those area far from the classes, where no event (in the sense of data entry) was observed.

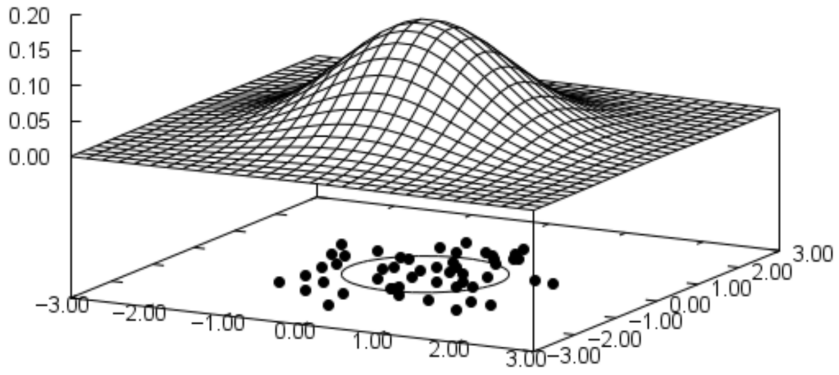


Figure 8.4: Example of multinomial Gaussian.

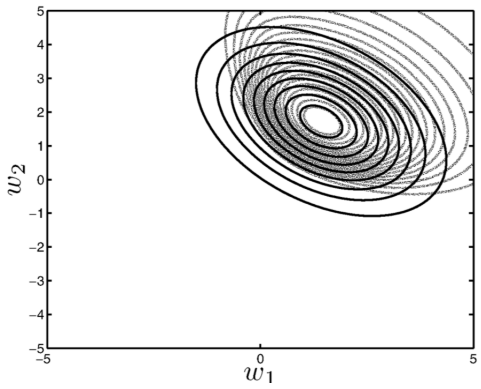


Figure 8.5: The axis are the parameters. The Laplace approximation of the posterior (darker lines) is shown while the lighter lines are the true unnormalised posterior. The centre point corresponds to the maximum point $\hat{\beta}$.

This is made clearer by looking at the decision boundaries in 8.6c and 8.6d. In case of MAP it is quite obvious: the probability increases or decreases just moving close to or far from to the boundary. In case of Laplace approximation the contours are no longer straight lines. The probability are now close to 0.5 in all the area except those closes to the two classes. It is like the classifier is unable to take a decision in those areas, that are the ones where no event has been noted.

On the contrary the classifier resulting from MAP approximation is always sure about its work: it can only classify either with one or the other class except in on the points on the boundaries. This result from the fact that it is the result of a single point and not a distribution: i.e., the amount of confidence on the result is left out.

8.7.5 Third method: sampling technique

The reason of estimating the posterior density with Laplace approximation or any other method is to take into account uncertainties in β when making predictions. Using the Gaussian approximation we were able to sample as in equation 8.55 directly from data. When deciding we take the conditional probability by averaging all over the potential values β by taking the expectation:

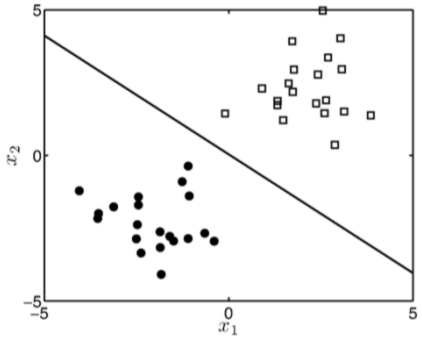
$$\Pr(\mathrm{T}_{\mathrm{new}} = 1|\mathrm{x}_{\mathrm{new}}, \mathbf{X}, \mathbf{y}, \sigma^2) = \mathbf{E}_{\Pr(\beta|\mathbf{X},\mathbf{y},\sigma^2)}(\Pr(\mathrm{T}_{\mathrm{new}} = 1|\mathrm{x}_{\mathrm{new}}, \beta) \quad (8.56)$$

In these types of approximations we cut off the approximation step and sample directly from the posterior. A set of samples from the true posterior could be substituted directly into equation 8.55 to compute the desired predictive probability $\Pr(\mathrm{y_i}|\mathbf{x_i}, \mathbf{X}, \beta)$. A popular sampling technique is the **Metropolis-Hastings** algorithm.

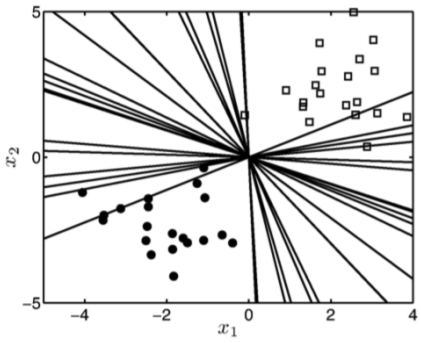
Drawing from the posterior does not mean we can write it down but that we sample directly the system (physical or whatever) generating the data.

The metropolis-Hastings algorithms The objective is to sample from $\Pr(\beta|\mathbf{X}, \mathbf{y}, \sigma^2)$ to approximate the following expectation:

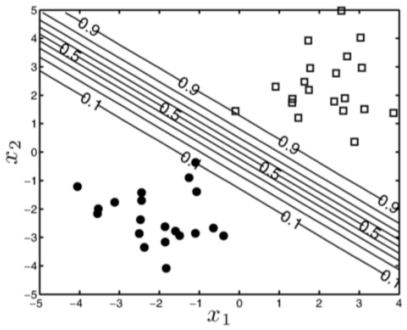
$$\begin{aligned} \Pr(\mathrm{Y}_{\mathrm{new}} = 1|\mathrm{x}_{\mathrm{new}}, \mathbf{X}, \mathbf{y}, \sigma^2) &= \mathbf{E}_{\Pr(\beta|\mathbf{X},\mathbf{y},\sigma^2)}\left[\Pr(\mathrm{T}_{\mathrm{new}} = 1|\mathrm{x}_{\mathrm{new}}, \beta)\right] \\ &= \int \Pr(\mathrm{Y}_{\mathrm{new}} = 1|\mathrm{x}_{\mathrm{new}}, \beta)\Pr(\beta|\mathbf{X}, \mathbf{y}, \sigma^2)\mathrm{d}\beta \end{aligned} \quad (8.57)$$



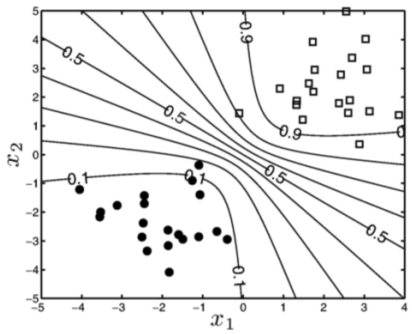
(a) Decision boundary for MAP estimation.



(b) Decision boundary for Laplace approximation estimation.



(c) Contours of probability of belonging to class 1 in case of MAP estimation.



(d) Contours of probability of belonging to class 1 in case of Laplace approximation estimation computed with a sample based approximation.

Figure 8.6: Comparison between MAP and Laplace approximation contours and probability densities.

with

$$\Pr(\mathrm{Y_{new}} = 1|\mathrm{x_{new}}, \mathbf{X}, \mathbf{y}, \sigma^2) \approx \frac{1}{\mathrm{N_s}} \sum_{\mathrm{s}=1}^{\mathrm{N_s}} \Pr(\mathrm{Y_{new}} = 1|\mathrm{x_{new}}, \beta_{\mathrm{s}}) \tag{8.58}$$

The algorithm generates a sequence of samples $\beta_1, \beta_2, \cdots$. First all the algorithm is independent from the starting point, as long as we sample long enough: it is guaranteed the sequence converges.

The generation of new samples happens in this way. Suppose we have the sample s – 1, we will propose a new sample $\tilde{\beta}_{\mathrm{s}}$ and define the density $\Pr(\tilde{\beta}_{\mathrm{s}}|\beta_{\mathrm{s-1}})$. This density is unrelated with the posterior $\Pr(\beta|\mathbf{X}, \mathbf{y}, \sigma^2)$ and we can define it as we please but it will affect the convergence time. A common choice is to use a Gaussian centred on the current sample, $\beta_{\mathrm{s-1}}$:

$$\Pr(\tilde{\beta}_{\mathrm{s}}|\beta_{\mathrm{s-1}}, \Sigma) = \mathcal{N}(\beta_{\mathrm{s-1}}, \Sigma) \tag{8.59}$$

Generally σ is taken diagonal with same values. The smaller the elements on the diagonal, the smaller the distance at each step.

Such a sequence creates a **random walk**. The choice of the Gaussian is justified bye the easy of sampling from the Gaussian and by its symmetry: moving from $\tilde{\beta}_{\mathrm{s-1}}$ to $\tilde{\beta}_{\mathrm{s}}$ is just as likely to move from $\tilde{\beta}_{\mathrm{s}}$ to $\tilde{\beta}_{\mathrm{s-1}}$:

$$\Pr(\tilde{\beta}_{\mathrm{s}}|\beta_{\mathrm{s-1}}, \Sigma) = \Pr(\tilde{\beta}_{\mathrm{s-1}}|\beta_{\mathrm{s}}, \Sigma) \tag{8.60}$$

Accepting or rejecting the candidate $\tilde{\beta}_{\mathrm{s}}$ is performed by calculating the following quantity:

$$\mathrm{r} = \frac{\Pr(\tilde{\beta}_{\mathrm{s}}|\mathbf{X}, \mathbf{y}, \sigma^2)}{\Pr(\beta_{\mathrm{s-1}}|\mathbf{X}, \mathbf{y}, \sigma^2)} \frac{\Pr(\beta_{\mathrm{s-1}}|\tilde{\beta}_{\mathrm{s}}, \Sigma)}{\Pr(\hat{\beta}_{\mathrm{s}}|\beta_{\mathrm{s-1}}, \Sigma)} \tag{8.61}$$

The expression is the product of the ratio of the posterior density at the proposed sample to that at the old sample times the ratio of the proposed densities. For the Gaussian symmetry discussed above, this term is 1 when using Gaussian densities. Note that we cannot compute exactly the densities, but being a ratio the normalisation constant (the denominator in Bayes expression) simplifies and only the likelihoods times the priors are left:

$$\mathrm{r} = \frac{\Pr(\tilde{\beta}_{\mathrm{s}}|\sigma^2)}{\Pr(\beta_{\mathrm{s-1}}|\sigma^2)} \frac{\Pr(\mathbf{y}|\mathbf{X}, \tilde{\beta}_{\mathrm{s}}, \sigma^2)}{\Pr(\mathbf{y}|\mathbf{X}, \beta_{\mathrm{s-1}}, \sigma^2)} \tag{8.62}$$

If r $>$ 1, i.e, we get a higher posterior density, we accept the candidate otherwise we accept the candidate with probability equal to r. The complete algorithm is depicted in 8.7 where a uniform distribution in $[0, 1]$ is used as decision rule in case r $<$ 1. Being a uniform distribution, the probability that u $\leq$ r is r.

8.8 shows a 10-iteration process of the algorithms where solid lines are accepted coefficients, dashed lines are the rejected ones. Note that even the third sample causes a decrease in the posterior r $<$ 1, nevertheless in this specific case the decision rule accepted it. On the contrary the 4-th sample causes a huge decrease hence it is very unlikely it is accepted and in fact it is not, so $\beta_4 = \beta_3$. After N samples we compute the sample based approximations to the mean and covariance

$$\begin{aligned} \mu' &= \frac{1}{\mathrm{N_s}} \sum_{\mathrm{s}=1}^{\mathrm{N_s}} \beta_{\mathrm{s}} \\ \mathrm{S}' &= \frac{1}{\mathrm{N_s}} \sum_{\mathrm{s}=1}^{\mathrm{N_s}} (\beta_{\mathrm{s}} - \mu')(\beta_{\mathrm{s}} - \mu')^{\mathrm{T}} \end{aligned} \tag{8.63}$$

Definition 8.1. Burn-in It is the time interval between the starting of the algorithm and the convergence.

It cannot be determined: to overcome this problem, a method for determining convergence (to a distribution, not to a value) should be established. For example, in the algorithm above, we do not know if the starting point belongs to an area from which we are supposed to sample: it might be very far from the posterior. Including these samples in the approximation might result in a not good value. That is why the first samples (ranging from few samples to thousands) should be discarded.

A popular method is to start several samplers simultaneously from different starting points. When all the samplers are generating samples with similar mean and variance, it suggests they converged all to the same distributions.

Definition 8.2. convergence In this case we are talking about the convergence to a given distribution, not a single point. The convergence to a distribution is characterized to the convergence of its parameters: in case of the Gaussian the mean and variance.

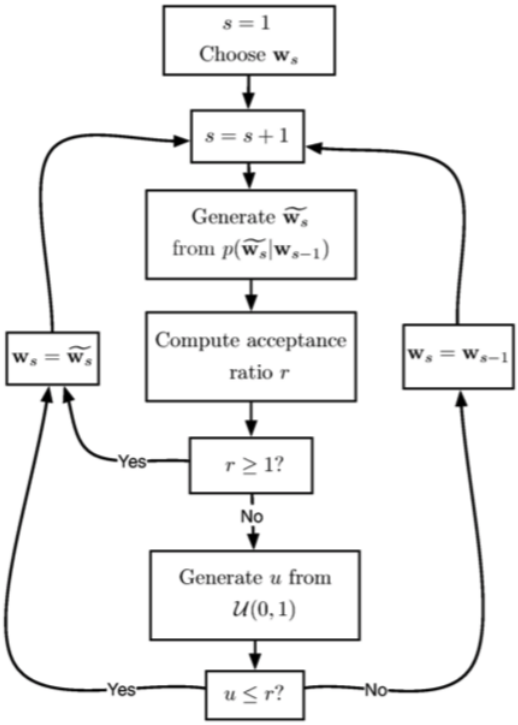
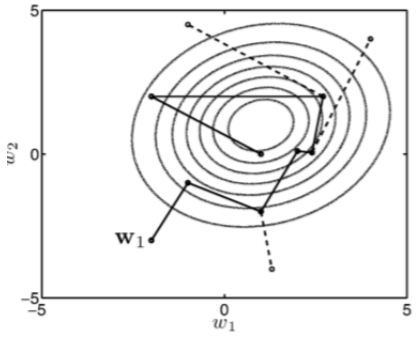


Figure 8.7: The steps of the Metropolis-Hastings algorithm. U is a uniform distribution in $[0, 1]$



(e) After ten samples.

Figure 8.8: Example of Metropolis-Hastings iterations: solid lines are accepted coefficients, dashed lines are the rejected ones.

We can even look at each coefficient independently:

$$\Pr(\beta_1|\mathbf{X},\mathbf{y},\sigma^2) = \int \Pr(\beta_1,\beta_2|\mathbf{X},\mathbf{y},\sigma^2)\mathrm{d}\beta_2 \tag{8.64}$$

To calculate the predictive probability using the obtained set of samples, we can do what already done with the Laplace approximation:

$$\Pr(\mathrm{Y}_{\mathrm{new}} = 1|\mathbf{x}_{\mathrm{new}},\mathbf{X},\mathbf{y},\sigma^2) = \frac{1}{\mathrm{N_s}}\sum_{\mathrm{s}=1}^{\mathrm{N_s}} \frac{1}{1+\mathrm{e}^{-\beta_{\mathrm{s}}^{\mathrm{T}}\mathbf{x}_{\mathrm{new}}}} \tag{8.65}$$

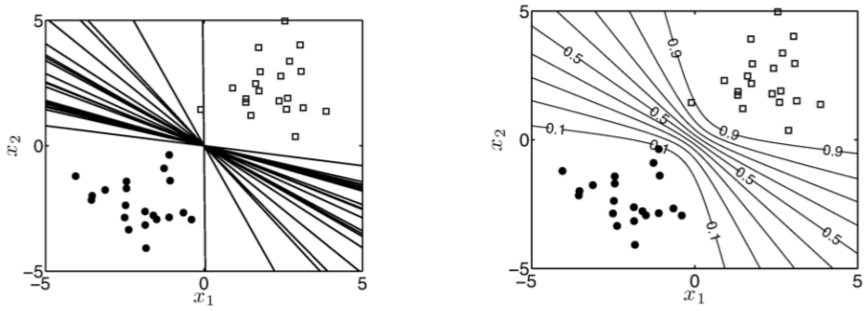
8.9a and 8.9b show an example of possible shapes of contours: it does not look too different from the Laplace ones. The only difference is that these contours are not so tight as the Laplace's ones. This suggests the probability decreases more slowly.

Limitations The difficult mostly lies in the unknown shape of the density. When a density has two or more modes, MH moves towards the modes as these moves increase the posterior density and hence are always accepted. When close to a mode, many steps are required to move from one mode to another and this is very unlikely. We might end up exploring a mode without even knowing the other exists (see 8.10a). Another problem arises when the variables are strongly correlated (see 8.10b). Let us pick any position and propose a movement from a Gaussian with diagonal covariance (i.e., having circular contours). The shapes are very different and many samples will be rejected: the majority of moves that we sample from our proposal will involve moving steeply down the probability gradient. There are even other problems.

8.7.6 Usage

LR is used as data analysis tool where the goal is to understand the role of the input variables in explaining the outcome. Typically many models are fit in a search for a parsimonious model involving a subset of the variables, possibly with some interactions terms.

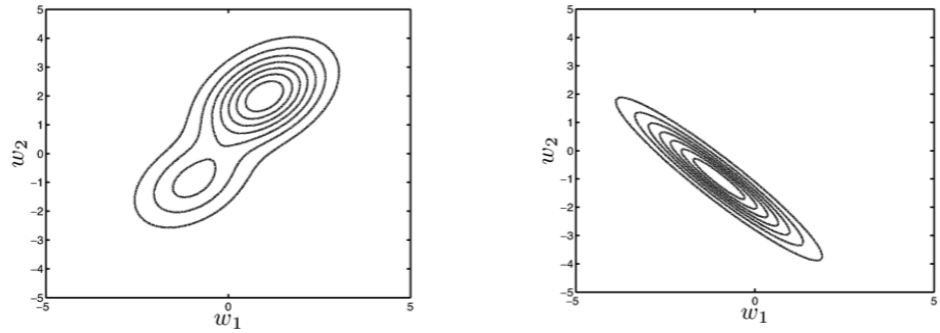
It is widely used in biostatistical applications where binary responses (two classes) occur quite frequently. For example, patients survive or die, have heart disease or not, or a condition is present or absent.



(a) Predictive probability contours of classifying objects as square.

(b) Decision boundaries created from randomly selected MH samples.

Figure 8.9: Example of Metropolis-Hastings algorithm results.



(a) A bi-modal density.

(b) Highly correlated parameter density.

Figure 8.10: Example of Metropolis-Hastings algorithm results.

8.8 Regularized Logistic regression

We can use the L_1 penalty for variable selection and shrinkage:

$$\underset{\beta_0,\beta_1}{\arg\max} \left\{ \sum_{i=1}^{N} y_i \left(\beta_0 + \beta^T x_i\right) - \log\left(1 + e^{\beta_0+\beta^T x_i}\right) - \lambda \sum_{j=-1}^{p} |\beta_j| \right\} \quad (8.66)$$

This function is concave and can be solved using a nonlinear programming method.

8.9 Logistic vs LDA

The difference between the models relies on how the linear coefficients are estimated. The logistic regression model is more general since it makes less assumptions.

LDA is not robust to outliers since observations far from the decision boundary are used to estimate the common covariance matrix, while they are scaled down in the Logistic regression.

8.10 Perceptron learning algorithm

It tries to find a separate hyperplane by minimizing the distance of misclassified points to the decision boundary. If a response $y_i = 1$ is misclassified, then $x_i^T\beta + \beta_0 < 0$, and the opposite for a misclassified response with $y_i = -1$. The goal is to minimize

$$D(\beta, \beta_0) = -\sum_{i\in\mathcal{M}} y_i \left(x_i^T\beta + \beta_0\right) \quad (8.67)$$

where $\mathcal{M}$ is the set of misclassified points. The gradient is

$$\frac{\partial D(\beta, \beta_0}{\partial\beta} = -\sum_{i\in\mathcal{M}} y_i x_i \quad (8.68)$$

$$\frac{\partial D(\beta, \beta_0}{\partial\beta_0} = -\sum_{i\in\mathcal{M}} y_i \quad (8.69)$$

where the algorithm uses the stochastic gradient descent where the coefficients are updated by the gradient value weighted by a step ρ. There are many problems though:

- when data are separable, there are many solutions which depend on the starting value;
- many steps might be required;
- when data are not separable, the algorithm will not converge;

8.11 Optimal separating hyperplanes

See 15.1.

9 Basis expansion and regularization

To extend the flexibility of linear model, a transformation of $\mathbf{X}$ can be performed and then use a linear model to the derived input features. Denoting the transformation $h_m(\mathbf{X}) : \mathcal{R}^p \rightarrow \mathcal{R}$ for a single input m, we have the model:

$$f(\mathbf{X}) = \sum_{m=1}^{M} \beta_m h_m(\mathbf{X}) \tag{9.1}$$

h() are called **basis function** Any kind of transformation can be applied: square-roots, squared (on single inputs i.e, x_i^2 or among different inputs $x_i x_j$), logarithmic, power functions, trigonometric, etc.

Polynomials are more widely used but they have a limitation due to its global nature: when tweaking the parameters to achieve a desired behaviour in a specific region, problems might arise in another region.

A solution is to consider **piecewise-polynomials** and **splines**.

9.1 Piecewise Polynomials and splines

For the moment X is assumed one-dimensional. A piecewise polynomial function is obtained by dividing the domain of X into contiguous intervals and use a different f in each interval.

Given a sequence $a = t_0 < t_1 < \cdots < t_{m+1} = b$, the piecewise polynomials are delimited by an adjacent pair in the sequence. So we have the following sequence of intervals:

$$I_l = [t_{l-1}, t_l) \ \textit{for}\ 1 \leq l \leq m \ \textit{and}\ [t_m, t_{m+1}] \tag{9.2}$$

So the piecewise polynomials, having basis functions: $h_0(X) = 1, h_1(X) = X, h_2(X) = X^2$ and so on, is:

$$f(x) = \begin{cases} g_0(x) = \beta_{0,0} + \beta_{0,1}X + \cdots + \beta_{0,k-1}X^{k-1} & X \in I_1 \\ \vdots \\ g_{m-1}(x) = \beta_{m-1,0} + \beta_{m-1,1}X + \cdots + \beta_{m-1,k-1}X^{k-1} & X \in I_{k+1} \end{cases} \tag{9.3}$$

In this way we get discontinuities at the boundaries. Note that discontinuity means we get completely different values for almost the same input.

More often we would like to have continuity at the boundaries or knots: $f(\xi^+) = f(\xi^-)$ which implies the condition:$\beta_1 + \xi\beta_2 = \beta_3 + \xi_1\beta_4$ which removes one degree of freedom.

These constrains, one for each knot, can be embedded in additional basis functions of the type $h(X) = (X - \xi)_+$ where $(\cdot)_+ = \max(\cdot, 0)$ i.e., the basis function is 0 for $X - \xi < \xi$, $X - \xi$ otherwise

$$h(X) = (X - \xi)_+ = \max(X - \xi, 0) = \begin{cases} 0 & X - \xi < 0 \\ X - \xi & X - \xi \geq 0 \end{cases} \tag{9.4}$$

Using the previous and these basis functions we can reexpress the global function as:

$$\begin{aligned} f(x) =& \beta_{0,0} + \beta_{0,1}X + \cdots + \beta_{0,k-1}X^{k-1} + \\ & + \beta_{1,0}(X - \xi_0)^0_+ + \beta_{1,1}(X - \xi_0)^1_+ + \cdots + \beta_{1,k-1}(X - \xi_0)^{k-1}_+ + \\ & + \vdots \\ & + \beta_{m-1,0}(X - \xi_{m-1})^0_+ + \cdots + \beta_{m-1,k-1}(X - \xi_{m-2})^{k-1}_+ \end{aligned} \tag{9.5}$$

To force continuity between two regions we set the terms $\beta_{1,0}(X - \xi_j)^0_+$ to 0:

For example forcing continuity at the first knot:

$$\begin{aligned} & \beta_{1,0}(X - \xi_0)^0_+ = \beta_{0,0} + \beta_{0,1}\xi_0 \\ & \Rightarrow \beta_{1,0} = \beta_{0,0} + \beta_{0,1}\xi_0 \end{aligned} \tag{9.6}$$

In this way we don't have a jump at the boundaries:

$$\begin{aligned} f(x) =& \beta_{0,0} + \beta_{0,1}X + \cdots + \beta_{0,k-1}X^{k-1} + \\ & + \beta_{1,1}(X - \xi_0)^1_+ + \cdots + \beta_{1,k-1}(X - \xi_0)^{k-1}_+ + \\ & + \vdots \\ & + \beta_{m-1,1}(X - \xi_{m-1})^1_+ + \cdots + \beta_{m-1,k-1}(X - \xi_{m-1})^{k-1}_+ \end{aligned} \tag{9.7}$$

In this way we have removed $m - 1$ parameters.

There is a difference in this representation: before we have to select a polynomial function for a specific region where the fit to the global function was given by the fit of the parameters specific to that region, now we have

a global function that works a little differently. In the first region, the only non-null terms are $\beta_{0,\mathrm{j}}$ as before. When reaching the second region, the terms $\beta_{1,\mathrm{j}}$ becomes non-null. Now the coefficients of the polynomial are the sum of the coefficients of these two region: i.e., in the second region we have:

$$\begin{aligned} \mathrm{f(x)} =& \beta_{0,0} - (\beta_{1,1} + \cdots + \beta_{1,\mathrm{k-1}})\xi_0 + (\beta_{0,1} + \beta_{1,1})\mathrm{X} + \\ &+ \cdots + (\beta_{0,\mathrm{k-1}} + \beta_{1,\mathrm{k-1}})\mathrm{X}^{\mathrm{k-1}} \end{aligned} \tag{9.8}$$

9.1 shows the global function, solid line, given by the sum of basis functions, dashed lines, when traversing the graph from left to right. At each knot, a new basis function is activated and summed to the global function.

The basis functions can be seen as a correction to the global polynomial for that specifc region and the knots are those points where the correction starts. At the knot $\mathrm{X} = \xi_{\mathrm{j}}$ so the correction starts very gently and the continuity is preserved.

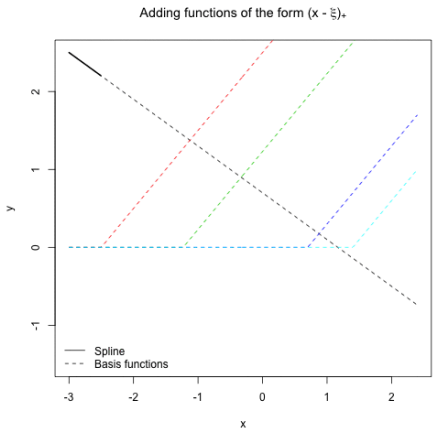
Generally, smoother functions are preferred and these are achieved by increasing the order of the local polynomials and increasing the order of continuity (continuity first derivative, second derivative). It is claimed that cubic splines are the lowest-order spline for which the discontinuity is not visible to the human eye and seldom higher degrees are needed, unless smooth derivatives are required.

The spline order, number of knots and their positions must be selected. Generally the position of the knots is chosen according the observation, while the order is chosen a-priori.

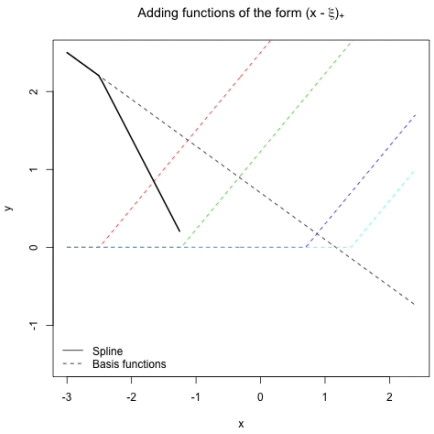
9.2 Natural cubic splines

Polynomial fits to data tends to be erratic at the boundaries and additionally splines behave even more widely than the global polynomial beyond the boundaries. This is shown by the point-wise variance (the variance of the estimator at one point: $\mathrm{Var}[\hat{\mathrm{f}}(\mathrm{x}_0)]$ in 9.3. Note the variance explosion close to the boundaries, especially for cubic splines.

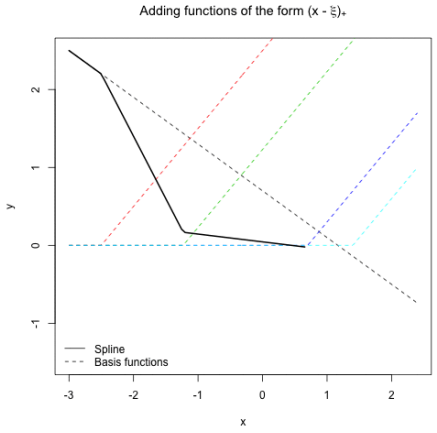
A **natural cubic spline** adds additional constraints by forcing linearity beyond the boundary knots. These frees up four degrees of freedom (2 in both boundary regions), that can be used more profitably by sprinkling more knots in the interior region. There will be a price to pay in terms of bias near the boundaries, but in these regions we have less information anyway so assuming the function is linear here is reasonable.



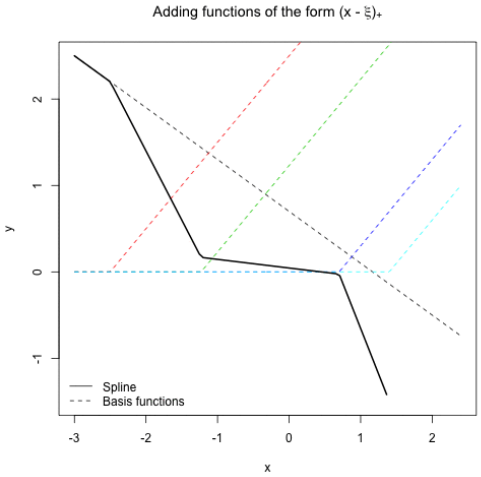
(a) First region.



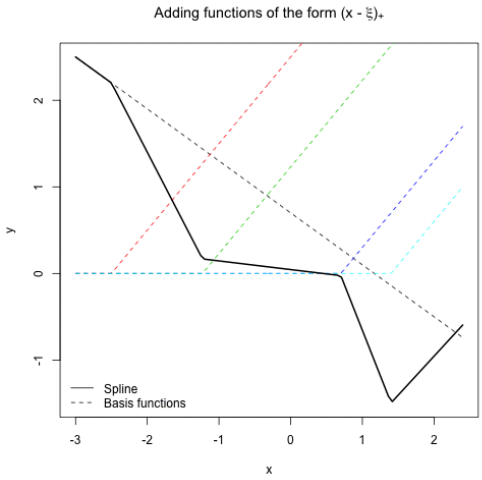
(b) Second region.



(c) Third region.



(d) Fourth region.



(e) Fifth region.

Figure 9.1: Linear splines. The solid line is the global function while the dashed lines are the basis functions.

A natural cubic spline with K knots is represented by K basis functions. One can start from the basis of a cubic spline and then derive the reduce basis by imposing the boundary constraints.

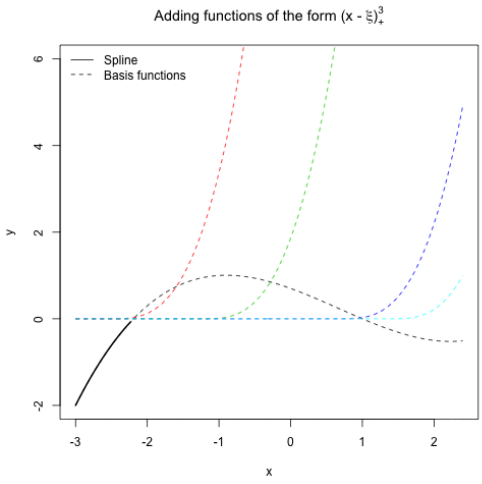
Starting from the general continuous basis for four regions (four knots):

$$\begin{aligned} &h_1(X) = 1, \qquad h_2(X) = X, \quad h_3(X) = X^2, \quad h_4(X) = X^3, \\ &h_5(X) = X - \xi_0, \quad h_6(X) = (X - \xi_0)^2, \quad , h_7(X) = (X - \xi_0)^3 \\ &h_8(X) = X - \xi_1, \quad h_9(X) = (X - \xi_1)^2, \quad h_{10}(X) = (X - \xi_1)^3 \\ &h_9(X) = X - \xi_2, \quad h_10(X) = (X - \xi_1)^2, \quad h_{11}(X) = (X - \xi_2)^3 \end{aligned} \tag{9.9}$$

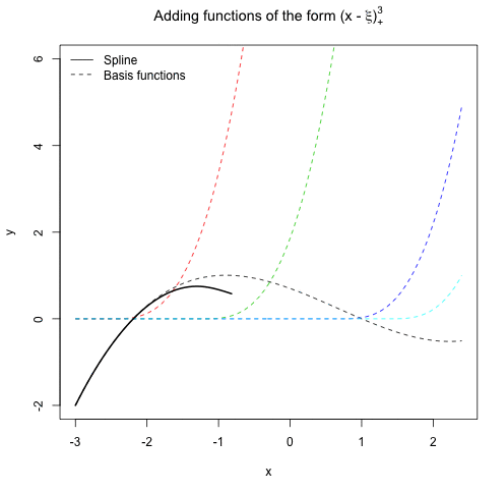
Enforcing also continuity of the first and second derivatives at the knots we reduce the basis to:

$$\begin{aligned} &h_1(X) = 1, \quad h_2(X) = X, \quad h_3(X) = X^2, \quad h_4(X) = X^3, \\ &h_5(X) = (X - \xi_0)^3, \quad h_6(X) = (X - \xi_1)^3, \quad h_7(X) = (X - \xi_2)^3. \end{aligned} \tag{9.10}$$

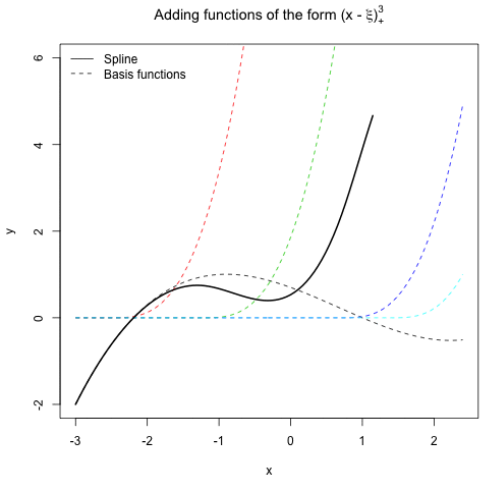
Enforcing linearity beyond the boundaries, we eliminate $h_3(X)$ and $h_4(X)$. We must also keep the cubic expression in the intermediate regions but somehow make the cubic part cancel out when reaching the fifth region since here



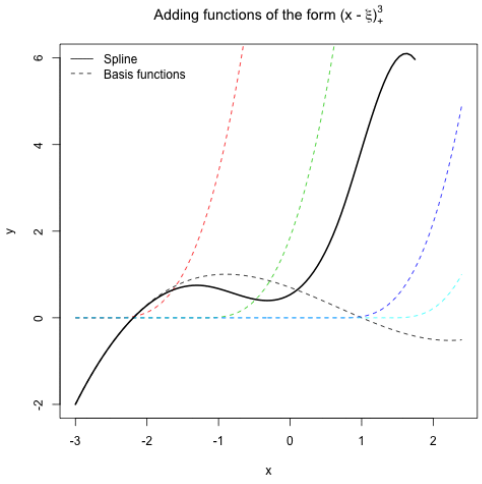
(a) First region.



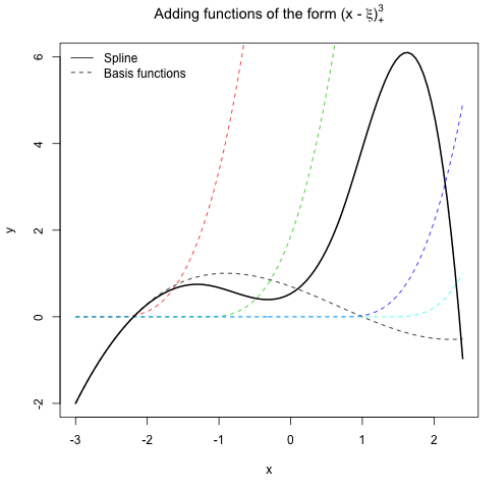
(b) Second region.



(c) Third region.



(d) Fourth region.



(e) Fifth region.

Figure 9.2: Cubic splines. The solid line is the global function while the dashed lines are the basis functions.

it must be linear. We can add for each region in the interval $[1, K-2]$ the following basis function:

$$\begin{aligned} & h_k(X) = d_k(X) - d_{K-1}(X) \\ \textit{where} \quad & d_k(X) = \frac{(X-\xi_k)^3_+ - (X-\xi_K)^3_+}{\xi_K - \xi_k} \end{aligned} \tag{9.11}$$

In the first region we have:

$$\hat{f}_1(X) = \beta_0 + \beta_1 X \tag{9.12}$$

In the second region we have:

$$\begin{aligned} & \hat{f}_2(X) = \beta_0 + \beta_1 X + \beta_2 \frac{(X-\xi_0)^3}{\xi_2 - \xi_0} = \\ & = \beta_0 + \beta_1 X + \beta_2 \frac{X^3 - 3\xi_0 X^2 + 3\xi_0^2 X - \xi_0^3}{\xi_2 - \xi_0} = \\ & = \frac{\beta_2}{\xi_2 - \xi_0} X^3 - 3\beta_2 \frac{\xi_0}{\xi_2 - \xi_0} X^2 + (\beta_1 + 3\beta_2 \frac{\xi_0^2}{\xi_2 - \xi_0}) X + \beta_0 - \beta_2 \frac{\xi_0^3}{\xi_2 - \xi_0} \end{aligned} \tag{9.13}$$

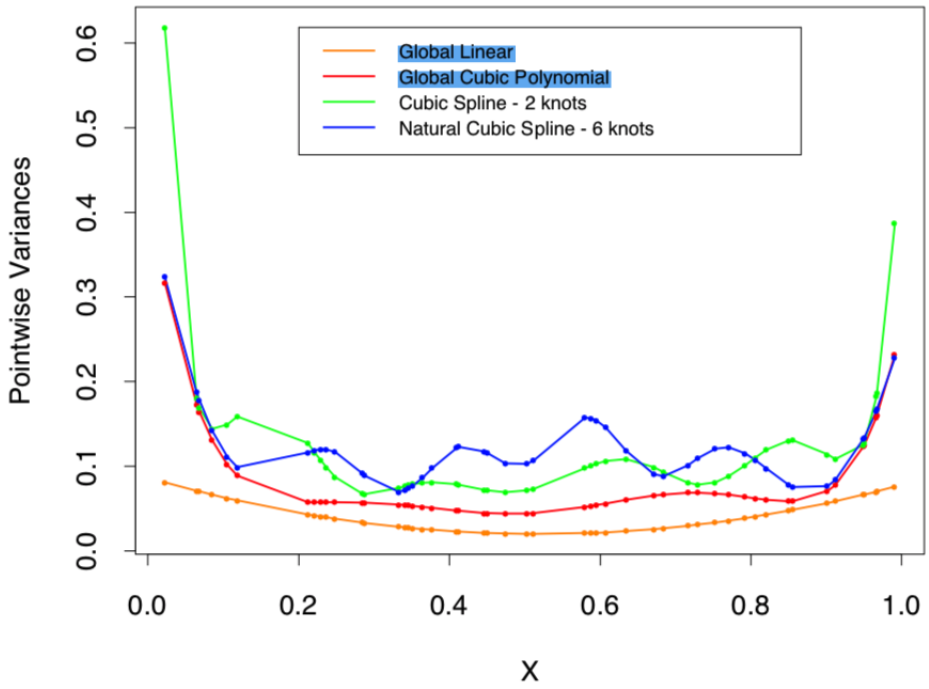


Figure 9.3: X consists of 50 points drawn randomly from U[0, 1] and an assumed error model of with constant variance. Both cubic splines have 6 degrees of freedom. The cubic spline has two knots at 0.33 and at 0.66 while the natural cubic spline has boundary knots at 0.1 and 0.9, and four interior knots uniformly spaced between them.

In the third region we have:

$$
\begin{aligned}
&\hat{\mathrm{f}}_3(\mathrm{X}) = \beta_0 + \beta_1\mathrm{X} + \beta_2\mathrm{h}_2(\mathrm{X}) = \beta_0 + \beta_1\mathrm{X} + \beta_2\left[\frac{(\mathrm{X}-\xi_0)^3}{\xi_2-\xi_0} - \frac{(\mathrm{X}-\xi_1)^3}{\xi_2-\xi_1}\right] \\
&= \beta_0 + \beta_1\mathrm{X} + \beta_2\left[\frac{\mathrm{X}^3 - 3\xi_0\mathrm{X}^2 + 3\xi_0^2\mathrm{X} - \xi_0^3}{\xi_2-\xi_0} - \frac{\mathrm{X}^3 - 3\xi_1\mathrm{X}^2 + 3\xi_1^2\mathrm{X} - \xi_1^3}{\xi_2-\xi_1}\right] = \\
&= \beta_0 + \beta_1\mathrm{X} + \beta_2\frac{(\xi_2-\xi_1-\xi_2+\xi_0)\mathrm{X}^3 - 3\left[\xi_0(\xi_2-\xi_1) - \xi_1(\xi_2-\xi_0)\right]\mathrm{X}^2}{(\xi_2-\xi_0)(\xi_2-\xi_1)} + \\
&\quad + \frac{3\left[\xi_0^2(\xi_2-\xi_1) - \xi_1^2(\xi_2-\xi_0)\right]\mathrm{X} - \left[(\xi_0^3(\xi_2-\xi_1) - \xi_1^3(\xi_2-\xi_0)\right]}{(\xi_2-\xi_0)(\xi_2-\xi_1)} = \\
&= \beta_0 + \beta_1\mathrm{X} + \beta_2\frac{(-\xi_1+\xi_0)\mathrm{X}^3 - 3\xi_2(\xi_0-\xi_1)\mathrm{X}^2}{(\xi_2-\xi_0)(\xi_2-\xi_1)} + \\
&\quad + \frac{3\left[\xi_2(\xi_0-\xi_1)(\xi_0+\xi_1) - \xi_0\xi_1(\xi_0-\xi 1)\right]\mathrm{X}}{(\xi_2-\xi_0)(\xi_2-\xi_1)} + \\
&\qquad - \frac{\left[\xi_2(\xi_0-\xi_1)(\xi_0^2-\xi_0\xi_1+\xi_1^2) - \xi_0\xi_1(\xi_0-\xi 1)(\xi_0+\xi_1)\right]}{(\xi_2-\xi_0)(\xi_2-\xi_1)} = \\
&= \beta_0 + \beta_1\mathrm{X} - \beta_2(\xi_1-\xi_0)\frac{\mathrm{X}^3 - 3\xi_2\mathrm{X}^2}{(\xi_2-\xi_0)(\xi_2-\xi_1)} + \\
&\qquad + \frac{3\left[\xi_2(\xi_0+\xi_1) - \xi_0\xi_1\right]\mathrm{X} - \left[\xi_2(\xi_0^2-\xi_0\xi_1+\xi_1^2) - \xi_0\xi_1(\xi_0+\xi_1)\right]}{(\xi_2-\xi_0)(\xi_2-\xi_1)} = \\
&= -\beta_2\frac{(\xi_1-\xi_0)}{(\xi_2-\xi_0)(\xi_2-\xi_1)}\mathrm{X}^3 + 3\beta_2\frac{\xi_2(\xi_1-\xi_0)}{(\xi_2-\xi_0)(\xi_2-\xi_1)}\mathrm{X}^2 + \\
&\qquad + (\beta_1 - 3\beta_2(\xi_1-\xi_0)\frac{\xi_2(\xi_0+\xi_1) - \xi_0\xi_1}{(\xi_2-\xi_0)(\xi_2-\xi_1)})\mathrm{X} + \\
&\qquad + \beta_0 + \beta_2(\xi_1-\xi_0)\frac{\xi_2(\xi_0^2-\xi_0\xi_1+\xi_1^2) - \xi_0\xi_1(\xi_0+\xi_1)}{(\xi_2-\xi_0)(\xi_2-\xi_1)}
\end{aligned}
\tag{9.14}
$$

In the fourth region:

$$
\begin{aligned}
&\hat{f}_3(X) = \beta_0 + \beta_1 X + \beta_2 h_2(X) = \\
&= \beta_0 + \beta_1 X + \beta_2 \left[\frac{(X-\xi_0)^3 - (X-\xi_2)^3}{\xi_2 - \xi_0} - \frac{(X-\xi_1)^3 - (X-\xi_2)^3}{\xi_2 - \xi_1} \right] \\
&= \beta_0 + \beta_1 X + \beta_2 \frac{X^3 - 3\xi_0 X^2 + 3\xi_0^2 X - \xi_0^3 - X^3 + 3\xi_2 X^2 - 3\xi_2^2 X + \xi_2^3}{\xi_2 - \xi_0} + \\
&\qquad - \beta_2 \frac{X^3 - 3\xi_1 X^2 + 3\xi_1^2 X - \xi_1^3 - X^3 + 3\xi_2 X^2 - 3\xi_2^2 X + \xi_2^3}{\xi_2 - \xi_1} = \\
&= \beta_0 + \beta_1 X + \beta_2 \frac{3(\xi_2 - \xi_0) X^2 - 3(\xi_2^2 - \xi_0^2) X + \xi_2^3 - \xi_0^3}{\xi_2 - \xi_0} + \\
&\qquad - \beta_2 \frac{3(\xi_2 - \xi_1) X^2 - 3(\xi_2^2 - \xi_1^2) X + \xi_2^3 - \xi_1^3}{\xi_2 - \xi_1} = \\
&= \beta_0 + \beta_1 X + \beta_2 \frac{\cancel{3(\xi_2 - \xi_1)(\xi_2 - \xi_0) X^2} - 3(\xi_2 - \xi_1)(\xi_2^2 - \xi_0^2) X + (\xi_2 - \xi_1)(\xi_2^3 - \xi_0^3)}{(\xi_2 - \xi_0)(\xi_2 - \xi_1)} + \\
&\qquad - \beta_2 \frac{\cancel{3(\xi_2 - \xi_0)(\xi_2 - \xi_1) X^2} - 3(\xi_2 - \xi_0)(\xi_2^2 - \xi_1^2) X + (\xi_2 - \xi_0)(\xi_2^3 - \xi_1^3)}{(\xi_2 - \xi_0)(\xi_2 - \xi_1)} = \\
&= \beta_0 + \beta_1 X + \beta_2 \frac{-3(\xi_2 - \xi_1)(\xi_2^2 - \xi_0^2) X + (\xi_2 - \xi_1)(\xi_2^3 - \xi_0^3)}{(\xi_2 - \xi_0)(\xi_2 - \xi_1)} + \\
&\qquad - \beta_2 \frac{-3(\xi_2 - \xi_0)(\xi_2^2 - \xi_1^2) X + (\xi_2 - \xi_0)(\xi_2^3 - \xi_1^3)}{(\xi_2 - \xi_0)(\xi_2 - \xi_1)} = \\
&= \beta_0 + \beta_1 X + \beta_2 \frac{3\left[-\xi_2(\xi_1^2 - \xi_0^2) + \xi_2^2(\xi_1 - \xi_0) + \xi_0\xi_1(\xi_1 - \xi_0)\right] X + }{(\xi_2 - \xi_0)(\xi_2 - \xi_1)} \\
&\quad \frac{-\xi_2(\xi_1^3 - \xi_0^3) - \xi_2^3(\xi_1 - \xi_0) - \xi_0\xi_1(\xi_1 - \xi_0)}{(\xi_2 - \xi_0)(\xi_2 - \xi_1)}
\end{aligned}
\tag{9.15}
$$

Note that the fourth region is linear.

9.3 Smoothing splines

We can avoid knot selection by using a maximal set of knots, controlled by a regularization process. The problem is: *Among all functions* $f(x)$ *with two continuous derivatives, find the one that minimizes the penalized residual sum*

of squares:

$$\mathrm{RSS}(\mathrm{f},\lambda) = \sum_{\mathrm{i}=1}^{\mathrm{N}} [\mathrm{y_i} - \mathrm{f}(\mathrm{x_i})]^2 + \lambda \int \mathrm{f}''(\mathrm{t})\mathrm{dt} \tag{9.16}$$

The first term measures closeness to the data, while the second term penalizes curvature in the function. If $\lambda = 0$ f can be any function that interpolates the data so we can achieve smoothness. If $\lambda = \infty$ simple least squares line fit, since no second derivative can be tolerated and we get a "rough" function. 9.16 has an explicit finite-dimensional unique minimizer which is a natural cubic spline with knots at the unique values of the $\mathrm{x_i}$, $\mathrm{i} = 1, ..., \mathrm{N}$, so there are as many as N knots, hence N degrees of freedom. The penalty translates to a penalty on the spline coefficients, which are shrunk to the linear fit.

Being the solution a natural spline, we can write it as:

$$\mathrm{f}(\mathrm{x}) = \sum_{\mathrm{j}=1}^{\mathrm{N}} \mathrm{N_j}(\mathrm{x})\theta_\mathrm{j} \tag{9.17}$$

where $\mathrm{N_j}(\mathrm{x})$ are the N-dimensional set of basis functions for representing this family of natural splines:

$$\mathrm{RSS}(\theta,\lambda) = (\mathbf{y} - \mathbf{N}\theta)^\mathrm{T}(\mathbf{y} - \mathbf{N}\theta) + \lambda\theta^\mathrm{T}\mathbf{\Sigma_N}\theta \tag{9.18}$$

where $\mathrm{N_{i,j}} = \mathrm{N_j}(\mathrm{x_i})$ and

$$\mathbf{\Sigma_N}_{\mathrm{j,k}} = \int \mathrm{N_j''}(\mathrm{t})\mathrm{N_k''}(\mathrm{t})\mathrm{dt}$$

. The solution is :

$$\hat{\theta} = \left(\mathbf{N}^\mathrm{T}\mathbf{N} + \lambda\mathbf{\Sigma}_\mathrm{N}\right)^{-1}\mathbf{N}^\mathrm{T}\mathbf{y} = \mathbf{S}_\lambda\mathbf{y} \tag{9.19}$$

which is a generalized ridge regression. $\mathbf{S}_\lambda$ is positive-definite, symmetric and $\mathbf{S}_\lambda\mathbf{S}_\lambda \leq \mathbf{S}_\lambda$. Since it is symmetric and positive-definite it has a real eigenvalue decomposition. Let us re-write $\mathbf{S}_\lambda = (\mathbf{I} - \lambda\mathbf{K})^{-1}$. The eigenvalue decomposition of $\mathbf{S}_\lambda$ is:

$$\begin{aligned} \mathbf{S}_\lambda &= \sum_{\mathrm{k}=1}^{\mathrm{N}} \rho_\mathrm{k}(\lambda)\mathbf{u_k}\mathbf{u_k^\mathrm{T}} \\ \rho_\mathrm{k}(\lambda) &= \frac{1}{1+\lambda\mathrm{d_k}} \end{aligned} \tag{9.20}$$

The degrees of freedom are $\mathrm{df}_\lambda = \mathrm{Tr}(\mathbf{S}_\lambda)$.

The eigenvectors are not affected by changes in λ, and hence the whole family of smoothing splines for a particular sequence of x, indexed by λ have the same eigenvectors. $\mathbf{S}_\lambda \mathbf{y} = \sum_{\mathrm{k}=1}^{\mathrm{N}} \mathbf{u}_\mathrm{k} \rho_\mathrm{k}(\lambda) \langle \mathbf{u}_\mathrm{k}, \mathbf{y} \rangle$: the smoothing spline decomposes $\mathbf{y}$ w.r.t. the complete basis $\mathbf{u}_\mathrm{k}$ and differently shrinking the contributions using $\rho_\mathrm{k}(\lambda)$. So for example suppose $\mathbf{B}_\mathrm{x}$i is a matrix of cubic-splines basis functions. Then the vector of fitted spline values is given by $\hat{\mathrm{f}} = \mathbf{B}_\xi (\mathbf{B}_\xi^\mathrm{T} \mathbf{B}_\xi)^{-1} \mathbf{B}_\xi^\mathrm{T} \mathbf{y} = \mathbf{H}_\xi \mathbf{y}$.In such a basis regression method the contributions are either left alone or shrunk to 0. The first two eigenvalues are always 1 and they correspond to the two-dimensional eigenspace of functions linear in x which are never shrunk. The eigenvalues $\rho_\mathrm{k}(\lambda) = 1/(1 + \lambda \mathrm{dk})$ are an inverse function of the eigenvalues d_k of the penalty matrix $\mathbf{K}$, moderated by λ; λ controls the rate at which the $\rho_\mathrm{k}(\lambda)$ decrease to zero. $\mathrm{d1} = \mathrm{d2} = 0$ and again linear functions are not penalized.

9.4 Multidimensional splines

Suppose $\mathrm{X} \in \mathcal{R}^2$ and suppose we have two sets: $\mathrm{h}_{1,\mathrm{k}}(\mathrm{X}_1), \mathrm{k} = 1, \cdots, \mathrm{M}_1$ for representing functions of coordinate X_1, and a set of M_2 functions $\mathrm{h}_{2,\mathrm{k}}(\mathrm{X}_2)$ for coordinate X_2. Then the$\mathrm{M}_1 \times \mathrm{M}_2$ dimensional tensor product basis defined by:

$$\mathrm{g}_{\mathrm{j,k}} = \mathrm{h}_{1,\mathrm{j}}(\mathrm{X}_1) \mathrm{h}_{2,\mathrm{k}}(\mathrm{X}_2), \quad \mathrm{j} = 1, \cdots, \mathrm{M}_1, \quad \mathrm{k} = 1, \cdots, \mathrm{M}_2 \tag{9.21}$$

can be used for representing a two-dimensional function:

$$\mathrm{g}(\mathrm{X}) = \sum_{\mathrm{j}=1}^{\mathrm{M}_1} \sum_{\mathrm{k}=1}^{\mathrm{M}_2} \theta_{\mathrm{jk}} \mathrm{g}_{\mathrm{jk}}(\mathrm{X}) \tag{9.22}$$

The coefficients can be fit with least squares. This can be generalized to d dimensions, but note that the dimension of the basis grows exponentially fast.

One-dimensional smoothing splines (via regularization) generalize to higher dimensions as well. Suppose we have pairs $\mathrm{y}_\mathrm{i}, \mathrm{x}_\mathrm{i}$ with $\mathrm{x}_\mathrm{i} \in \mathcal{R}^\mathrm{d}$, and we seek a d-dimensional regression function $\mathrm{f}(\mathrm{x})$. The idea is to set up the problem:

$$\mathrm{min}_\mathrm{f} \sum_{\mathrm{i}=1}^{\mathrm{N}} \left[\mathrm{y}_\mathrm{i} - \mathrm{f}(\mathrm{x}_\mathrm{i}) \right]^2 + \lambda \mathrm{J}\left[\mathrm{f} \right] \tag{9.23}$$

where J is an appropriate penalty functional for stabilizing a function f in $\mathcal{R}^{\mathrm{d}}$. For example:

$$\mathrm{J}\left[\mathrm{f}\right] = \int\int_{\mathcal{R}^2} \left[\left(\frac{\partial^2 \mathrm{f}(\mathrm{x})}{\partial \mathrm{x}_1^2} \right)^2 + 2 \left(\frac{\partial^2 \mathrm{f}(\mathrm{x})}{\partial \mathrm{x}_1 \partial \mathrm{x}_2} \right)^2 + \left(\frac{\partial^2 \mathrm{f}(\mathrm{x})}{\partial \mathrm{x}_2^2} \right)^2 \right] \mathrm{dx}_1 \mathrm{dx}_2 \quad (9.24)$$

If $\lambda \to 0$ the solution approaches an interpolating function, as λ the solution approaches the least squares plane. The solution has the form:

$$\mathrm{f}(\mathrm{x}) = \beta_0 + \beta^{\mathrm{T}} \mathrm{x} + \sum_{\mathrm{j}=1}^{\mathrm{N}} \alpha_{\mathrm{j}} \mathrm{h}_{\mathrm{j}}(\mathrm{x}) \quad (9.25)$$

Additive spline model can be represented in this formulation too: there exists a penalty J[f] that guarantees that the solution has the from $\mathrm{f}(\mathrm{X}) = \alpha + \mathrm{f}_1(\mathrm{X}_1) + \cdots + \mathrm{f}_{\mathrm{d}}(\mathrm{X}_{\mathrm{d}})$ and that each of the functions f_{j} are univariate splines. In this case it is better to assume f is additive and impose an additional penalty and impose an additional penalty on each of the component functions:

$$\mathrm{J}[\mathrm{f}] = \mathrm{J}[\mathrm{f}_1 + \mathrm{f}_2 + \cdots + \mathrm{f}_{\mathrm{d}}] = \sum_{\mathrm{j}=1}^{\mathrm{d}} \int \mathrm{f}_{\mathrm{j}}''(\mathrm{t}_{\mathrm{j}}) \mathrm{dt}_{\mathrm{j}} \quad (9.26)$$

These are naturally extended to ANOVA spline decomposition:

$$\mathrm{f}(\mathrm{X}) = \alpha + \sum_{\mathrm{j}} \mathrm{f}_{\mathrm{j}}(\mathrm{X}_{\mathrm{j}}) + \sum_{\mathrm{j}<\mathrm{k}} \mathrm{f}_{\mathrm{jk}(\mathrm{X}_{\mathrm{j}}, \mathrm{X}_{\mathrm{k}}} + \cdots \quad (9.27)$$

where each of the components are splines of the required dimension. There are many choices to be made:

- maximum order of interaction (the one above is up to order 2);
- which term to include
- the representation to use (regression splines with a small number of basis functions and their tensor product for interaction or a complete basis as in smoothing splines and include appropriate regularizers for each term in the expansion.

When the number of potential dimensions is large automatic methods are desirable (for example MARS and MART procedures).

9.5 Wavelet smoothing

...

10 Kernel smoothing methods

10.1 One-dimensional kernel smoothers

We have seen the *k-nearest neighbourhood* average as an estimate of the regression function $\mathbf{E}(\mathrm{Y}|\mathrm{X}=\mathrm{x})$:

$$\hat{\mathrm{f}} = \mathrm{Ave}(\mathrm{y_i}|\mathrm{x_i} \in \mathrm{N_k}(\mathrm{x})) \tag{10.1}$$

where $\mathrm{N_k}(\mathrm{x})$ is the set of k – points nearest to x in squared distance. An example of the resulting fitting curve is shown in green on the left of 10.1. The green curve is bumpy, since $\hat{\mathrm{f}}(\mathrm{x})$ is discontinuous in x. As we move $\mathrm{x_0}$ from left to right, the k-nearest neighborhood remains constant, until a point $\mathrm{x_i}$ to the right of $\mathrm{x_0}$ becomes closer than the furthest point $\mathrm{x_{i'}}$ in the neighbourhood to the left of $\mathrm{x_0}$, at which time $\mathrm{x_i}$ replaces $\mathrm{x_{i'}}$.

Rather than give all the points in the neighbourhood equal weights, we can assign weights that die off smoothly with distance from the target point. The right panel in 10.1 shows an example of this, using the so-called Nadaraya–Watson kernel-weighted average:

$$\hat{\mathrm{f}}(\mathrm{x}) = \frac{\mathrm{K}_\lambda(\mathrm{x_0}, \mathrm{x_i})\mathrm{y_i}}{\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{K}_\lambda(\mathrm{x_0}, \mathrm{x_i})} \tag{10.2}$$

with the Epanechnikov quadratic kernel:

$$\mathrm{K}_\lambda(\mathrm{x_0}, \mathrm{x_i}) = \mathrm{D}\left(\frac{|\mathrm{x}-\mathrm{x_0}|}{\lambda}\right) = \begin{cases} \frac{3}{4}\left[1 - \left(\frac{|\mathrm{x}-\mathrm{x_0}|}{\lambda}\right)^2\right] & \text{if } \frac{|\mathrm{x}-\mathrm{x_0}|}{\lambda} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{10.3}$$

Now the function is continuous. As we move the target from left to right, points enter the neighborhood initially with weight zero, and then their contribution slowly increases. Note that λ dictates the width of the window which is fixed. On the contrary, for the *k-nearest neighbourhood* the window size depends on the density since anyway k points must be considered. 10.2 shows other kernels. Another popular one is the tri-cube function:

$$\mathrm{D}\left(\frac{|\mathrm{x}-\mathrm{x_0}|}{\lambda}\right) = \begin{cases} \left[1 - \left(\frac{|\mathrm{x}-\mathrm{x_0}|}{\lambda}\right)^3\right]^3 & \text{if } \frac{|\mathrm{x}-\mathrm{x_0}|}{\lambda} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{10.4}$$

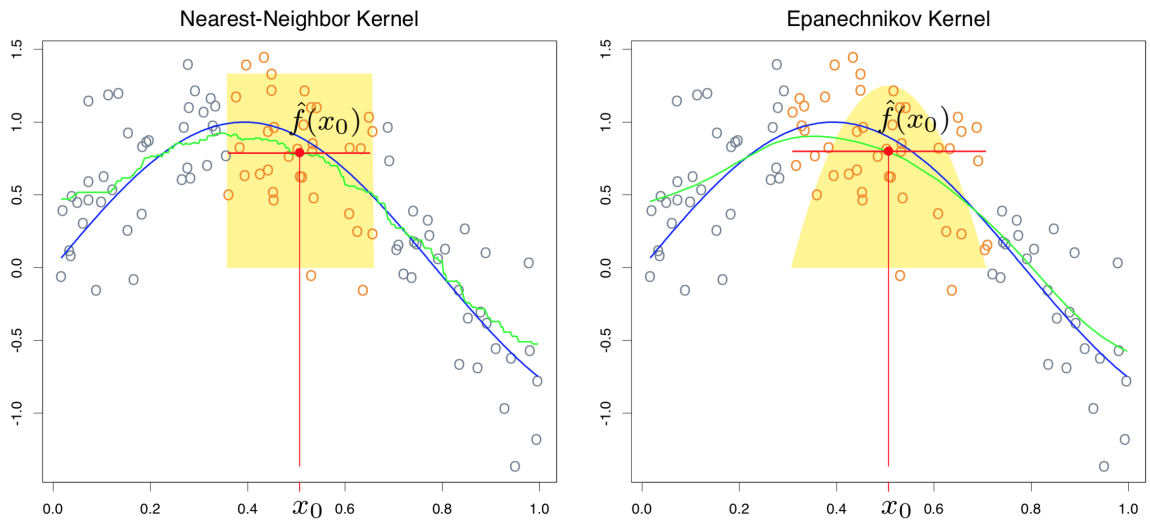


Figure 10.1: In green the fitting curves. On the left the one resulting from *k-nearest neighbourhood*, on the right the metric neighbourhood.

Both are compact functions. A common non-compact function is the Gaussian function.

Generalizing we can use a width function $h_\lambda(x_0)$ that determines the width of the neighbourhood at x_0 (previously we have used $h_\lambda(x_0) = \lambda$.

There are a number of details to be clarified:

- the value of λ which selects the window size. Large values imply lower variance but higher bias;
- $h_\lambda(x_0)$ can be constant (i.e., $h_\lambda(x_0) = \lambda$) or variable. Constant values (**metric window widths** keep the bias constant but the variance is inversely proportional to the local density. Nearest neighbourhood has opposite behaviour: the variance stays constant and the absolute bias varies inversely with local density.
- boundary issues: the metric neighbourhood tend to contain less points on the boundaries while the nearest-neighbourhood get wider.

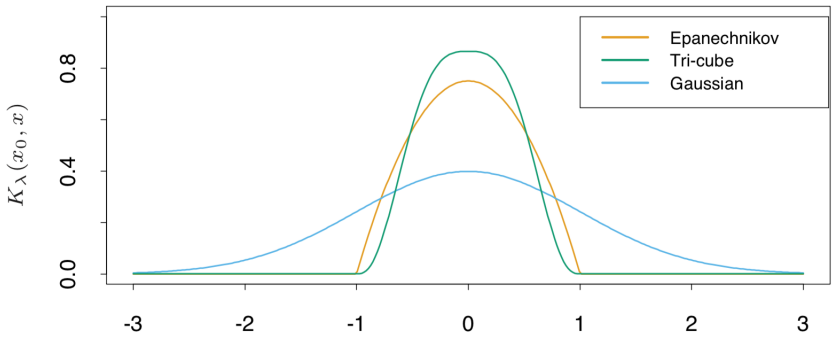


Figure 10.2: A comparison of three popular kernels for local smoothing. Each has been calibrated to integrate to 1. The tri-cube kernel is compact and has two continuous derivatives at the boundary of its support, while the Epanechnikov ker- nel has none. The Gaussian kernel is continuously differentiable, but has infinite support.

10.1.1 Local linear regression

Smooth kernel fit still has a problem close to the boundaries: because of the asymmetry in this region weighted averages can be badly biased (left in 10.3). By fitting straight lines rather than constants locally, we can remove this bias effect. Actually, this bias can be present in the interior of the domain as well, if the X values are not equally spaced. Locally weighted regression solves a separate weighted least squares problem at each target point x_0:

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_1)\,[y_i - \alpha(x_0) - \beta(x_0)x_i]^2 \tag{10.5}$$

and the estimate becomes:

$$\hat{f}(x_0) = \hat{\alpha}(x_0) + \hat{\beta}(x_0)x_0 \tag{10.6}$$

Notice that although we fit an entire linear model to the data in the region, we only use it to evaluate the fit at the single point x_0.

Define the vector-valued function $b(x)^T = (1, x)$. Let B be the $N \times 2$ regression matrix with i-th row $b(x_i)^T$, and $W(x_0)$ the $N \times N$ diagonal matrix with $i-th$ diagonal element $K_\lambda(x_0, x_i)$. Then

$$\hat{f}(x_0) = b(x_0)^T(\mathbf{B}^T\mathbf{W}(x_0)\mathbf{B})^{-1}\mathbf{B}^T\mathbf{W}(x_0)\mathbf{y} = \sum_{i=1}^{N} l_i(x_0)y_i \tag{10.7}$$

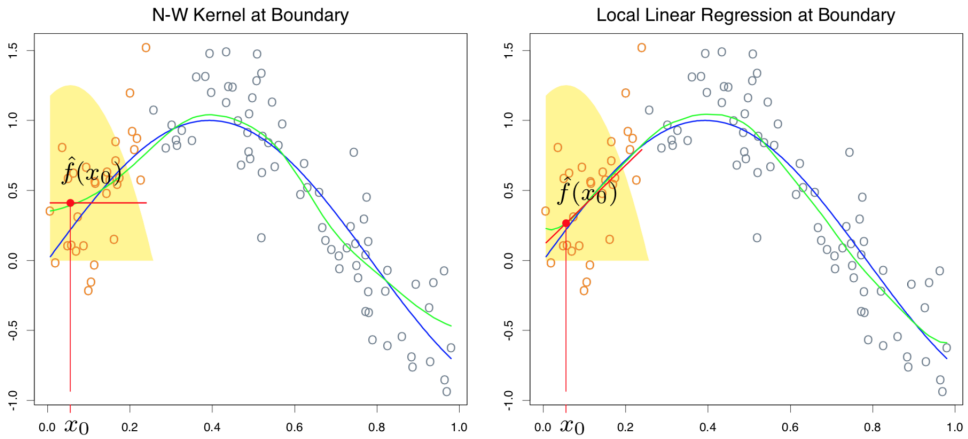


Figure 10.3: On the left it can be noticed the bias effect at the boundaries.

The weights $l_i(x_0)$ combine the weighting kernel $K_\lambda(x_0, \cdot)$ and the least squares operations and are referred to as the equivalent kernel.Local linear regression automatically modifies the kernel to correct the bias exactly to first order, a phenomenon dubbed as **automatic kernel carpentry**.

10.1.2 Local polynomial regression

We can fit local polynomial fits of any degree d:

$$\begin{aligned} &\min_{\alpha(x_0),\beta_j(x_0),j=1,\cdots,d} \sum_{i=1}^{N} K_\lambda(x_0, x_i) \left[y_i - \alpha_0 - \sum_{j=1}^{d} \beta_j(x_0) x_i^j \right]^2 \\ &\hat{f} = \hat{\alpha}(x_0) + \sum_{j=1}^{d} \hat{\beta}_j(x_0) x_0^j \end{aligned} \tag{10.8}$$

Local linear fits tend to be biased in regions of curvature of the true function, a phenomenon referred to as **trimming the hills** and **filling the valleys**. Local quadratic regression is generally able to correct this bias. There is of course a price to be paid for this bias reduction, and that is increased variance. The fit in the right panel of 10.4 is slightly more wiggly, especially in the tails. Summarizing, local linear fits can help bias dramatically at the boundaries at a modest cost in variance. Local quadratic fits do little at the boundaries for bias, but increase the variance a lot. Local quadratic fits

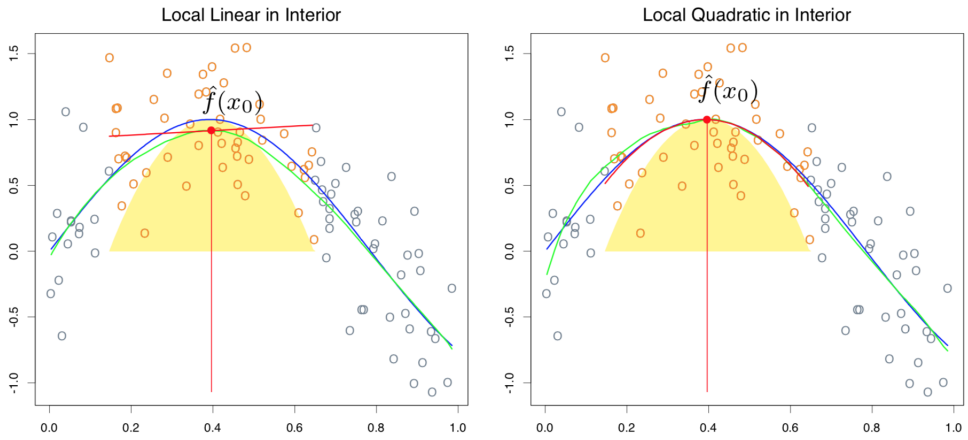


Figure 10.4: Local linear fits exhibit bias in regions of curvature of the true function. Local quadratic fits tend to eliminate this bias.

tend to be most helpful in reducing bias due to curvature in the interior of the domain. Asymptotic analysis suggest that local polynomials of odd degree dominate those of even degree. This is largely due to the fact that asymptotically the MSE is dominated by boundary effects.

10.2 Local regression in $\mathcal{R}^p$

Let b(X) be a vector of polynomial terms in X of maximum degree d. For example, with d = 1 and p = 2 we get $b(X) = (1, X_1, X_2)$; with d = 2 we get $b(X) = (1, X_1, X2, X_1^2, X_2^2, X_1X_2)$; and trivially with d = 0 we get b(X) = 1. At each $x_0 \in \mathcal{R}^p$ solve:

$$\min_{\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_1) \left(y_i - b(x_i)^T \beta(x_0)\right)^2 \tag{10.9}$$

to produce the fit

$$\hat{f}(x_0) = b(x_0)^T \hat{\beta}(x_0) \tag{10.10}$$

While boundary effects are a problem in one-dimensional smoothing, they are a much bigger problem in two or higher dimensions, since the fraction of points on the boundary is larger. In fact, one of the manifestations of the curse of dimensionality is that the fraction of points close to the boundary

increases to one as the dimension grows. Directly modifying the kernel to accommodate two-dimensional boundaries becomes very messy, especially for irregular boundaries. Directly modifying the kernel to accommodate two-dimensional boundaries becomes very messy, especially for irregular boundaries. Local regression becomes less useful in dimensions much larger than 2.

For high dimensionality it is impossible to simultaneously maintain localness (i.e., low bias) and a sizable sample in the neighbourhood (low variance) as dimensions increase, without the total sample size increasing exponentially with p.

10.3 Structured local regression models in $\mathcal{R}^{\mathrm{p}}$

When the dimension to sample-size ratio is unfavorable, local regression does not help us much, unless we are willing to make some structural assumptions about the model.

10.3.1 Structured kernels

The default spherical kernel gives equal weight to each coordinate and we can standardize the variables to unit deviation.

Another way is to use a positive definite matrix A to weigh the different coordinates:

$$\mathrm{K}_{\lambda}(\mathrm{x}_0, \mathrm{x}) = \mathrm{D}\left(\frac{(\mathrm{x}-\mathrm{x}_0)^{\mathrm{T}}\mathbf{A}(\mathrm{x}-\mathrm{x}_0)}{\lambda}\right) \tag{10.11}$$

Entire coordinates or directions can be downgraded or omitted by imposing appropriate restrictions on $\mathbf{A}$. For example, if $\mathbf{A}$ is diagonal, then we can increase or decrease the influence of individual predictors X_{j} by increasing or decreasing A_{jj}.

10.3.2 Structured regression function

We are trying to fit a regression function $\mathbf{E}[\mathrm{Y}|\mathrm{X}] = \mathrm{f}(\mathrm{X}_1, \mathrm{X}_2, \cdots, \mathrm{X}_{\mathrm{p}})$ in $\mathrm{CMcalR}^{\mathrm{p}}$ in which every level of interaction is potentially present. It is natural to consider analysis-of-variance (ANOVA) decompositions of the form

$$\mathrm{f}(\mathrm{X}_1, \mathrm{X}_2, \cdots, \mathrm{X}_{\mathrm{p}}) = \alpha + \sum_{\mathrm{j}} \mathrm{g}_{\mathrm{j}}(\mathrm{X}_{\mathrm{j}}) + \sum_{\mathrm{k}<\mathrm{l}} \mathrm{g}_{\mathrm{kl}}(\mathrm{X}_{\mathrm{k}}, \mathrm{X}_{\mathrm{l}}) + \cdots \tag{10.12}$$

and then introduce structure by eliminating the higher-order terms. Additive models assume only main effect terms: $f(X) = \alpha + \sum_{j=1}^{p} g_j(X_j)$, second order models will have terms with interactions of order at most two and so on. The important detail is that at any stage, one-dimensional local regression is all that is needed. The same ideas can be used to fit low-dimensional ANOVA decompositions.

10.4 Local likelihood and other models

Any parametric model can be made local.

For example we can use likelihood local to x_0, with the parameters $\theta(x_0) = x_0^T \beta(x_0)$:

$$l(\beta(x_0)) = \sum_{i=1}^{N} K_\lambda(x_0, x_i) l\Big(y_i, x_i^T \beta(x_0)\Big) \tag{10.13}$$

Also time-series can be made local by fitting by local least squares with a kernel in order to vary according to the short-term history of the series.

10.4.1 Local multiclassifier linear logistic regression

Consider categorical responses $g_i \in 1, 2, \cdots, J$. The global linear model is:

$$\Pr(G = j | X = x) = \frac{e^{\beta_{j0} + \beta_j^T x}}{1 + \sum_{k=1}^{J-1} e^{\beta_{k0} + \beta_k^T x}}$$

The local log-likelihood for the class J can be written as:

$$\begin{aligned} \sum_{i=1}^{N} K_\lambda(x_0, x_i) \Bigg\{ & \beta_{g_i 0}(x_0) + \beta_{g_i}(x_0)^T (x_i - x_0) + \\ & - \log \left[1 + \sum_{k=1}^{J-1} e^{\beta_{k0}(x_0) + \beta_k(x_0)^T (x_i - x_0)} \right] \Bigg\} \end{aligned} \tag{10.14}$$

Notice that $\beta_{J0} = \beta_J = 0$ i.e., the last class is taken as reference. Also, we have centred the local regression at x_0, so that the fitted posterior probabilities at x_0 are simply:

$$\hat{\Pr}(G = j, X = x_0) = \frac{e^{\hat{\beta}_{j0}(x_0)}}{1 + \sum_{k=1}^{J-1} e^{\hat{\beta}_{k0}(x_0)}} \tag{10.15}$$

10.5 Kernel density classification

One can use nonparametric density estimates for classification in a straightforward fashion using Bayes' theorem:

$$\hat{\Pr}(G = k|X = x_0) = \frac{\hat{\pi}_k \hat{f}_k(x_0)}{\sum_{j=1}^{J} \hat{\pi}_k \hat{f}_j(x_0)} \tag{10.16}$$

with $\hat{f}_j$ being a fitted nonparametric density estimates for each class separately.

10.6 Naive-Bayes classifier

It assumes that given a class $G = j$, the features X_k are independent:

$$f_j(X) = \prod_{k=1}^{p} f_{jk}(X_k) \tag{10.17}$$

While this assumption is generally not true, it does simplify the estimation dramatically since f_{jk} can be estimated separately using one dimensional kernel density estimates. Despite these rather optimistic assumptions, Naive-Bayes classifiers often outperform far more sophisticated alternatives. The reasons are that although the individual class density estimates may be biased, this bias might not hurt the posterior probabilities as much, especially near the decision regions. In fact, the problem may be able to withstand considerable bias for the savings in variance such a "naive" assumption earns.

10.7 Radial basis functions

Radial basis functions combine the idea of basis methods and kernel functions by treating $K_\lambda(\xi, x)$ as a basis function:

$$f(x) = \sum_{j=1}^{M} K_{\lambda_j}(\xi_j, x)\beta_j = \sum_{j=1}^{M} D\left(\frac{\|x - \xi_j\|}{\lambda_j}\right)\beta_j \tag{10.18}$$

A popular choice for D is the standard Gaussian density function. There are several methods to estimate the parameters, among which one of the

most common is the least squares. We can optimize the sum-of-squares with respect to all parameters:

$$\min_{\{\lambda_j,\xi_j,\beta_j\}_1^M} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{M} \beta_j e^{-\frac{(x_i-\xi_j)^T(x_i-\xi_j)}{\lambda_j^2}} \right)^2 \tag{10.19}$$

This model is referred to as RBF network, with ξ_J and λ_j playing the role of weights. This criterion is non-convex with multiple local minima.

Another way is to estimate $\{\lambda_j, \xi_j\}$ separately from β_j. Often, the kernel parameters λ_j and ξ_j are chosen in an unsupervised way using the X distribution alone. One of the method is to fit a Gaussian mixture density model to the training x_i, which provides both the centres ξ_j and the scales λ_j. Other approaches use cluster methods to locate ξ_j and treat $\lambda_j = \lambda$ as a hyper-parameter.

Using a constant λ might create holes-regions (regions that are not fitted by any gaussian) where no kernels has appreciable support. Renormalized radial basis functions solve this problem:

$$h_j(x) = \frac{D\left(\frac{\|x-\xi_j\|}{\lambda}\right)}{\sum_{k=1}^{M} D\left(\frac{\|x-\xi_k\|}{\lambda}\right)} \tag{10.20}$$

10.8 Mixture Models for Density Estimation and Classification

The mixture model can be viewed as a kind of kernel method:

$$\begin{aligned} f(x) &= \sum_{m=1}^{M} \alpha_m \phi(x, \mu_m, \Sigma_m) \\ \sum_m \alpha_m &= 1 \end{aligned} \tag{10.21}$$

It is possible to use any density instead of the Gaussian densities.

The parameters are usually fit by maximum likelihood, using the EM algorithm. If the covariance matrix is force to be scalar $\Sigma_m = \sigma_m \mathbf{I}$, then 10.21 has the form of a radial basis expansion. If $\sigma_m = \sigma > 0$ and $M \uparrow$

11 Model assessment and selection

Take a look also at this: https://machinelearningmastery.com/probabilistic-model-selection-measures/

Definition 11.1. Training error: it is the average loss over the training samples:

$$\bar{\text{err}} = \frac{1}{\text{N}} \sum_{\text{i}=1}^{\text{N}} \text{L}(\text{y}_\text{i}, \hat{\text{f}}(\text{x}_\text{i})) \tag{11.1}$$

For categorical variables it is:

$$\bar{\text{err}} = -\frac{2}{\text{N}} \sum_{\text{i}=1}^{\text{N}} \log \hat{\text{p}}_\text{k}(\text{x}_\text{i}) \tag{11.2}$$

The "–2" in the definition makes the log-likelihood loss for the Gaussian distribution match squared-error loss.

Definition 11.2. Test error: with the term *test error* or *generalization error* we indicate the prediction error over an independent test sample different from the one used for the training and it is indicated as

$$\text{Err}_\tau \left[\text{L}(\text{Y}, \hat{\text{f}}(\text{X}) | \tau \right] \tag{11.3}$$

where τ indicates the training set and X and Y are drawn randomly from their joint distribution.

Definition 11.3. Expected prediction error: or **expected test error** is the expectation of the test.

$$\text{Err} = \mathbf{E} \left[\text{L}(\text{Y}, \hat{\text{f}}(\text{X})) \right] = \mathbf{E} \left[\text{Err}_\tau \right] \tag{11.4}$$

11.1 shows the prediction errors of the same type of model trained with different training sets in light red and their expectation (average) in dark red.

To judge the prediction capability of a model we need to estimate Err_τ. As the model becomes more and more complex, it uses the training data more and is able to adapt to more complicated underlying structures. Hence there is a decrease in bias but an increase in variance. There is some intermediate

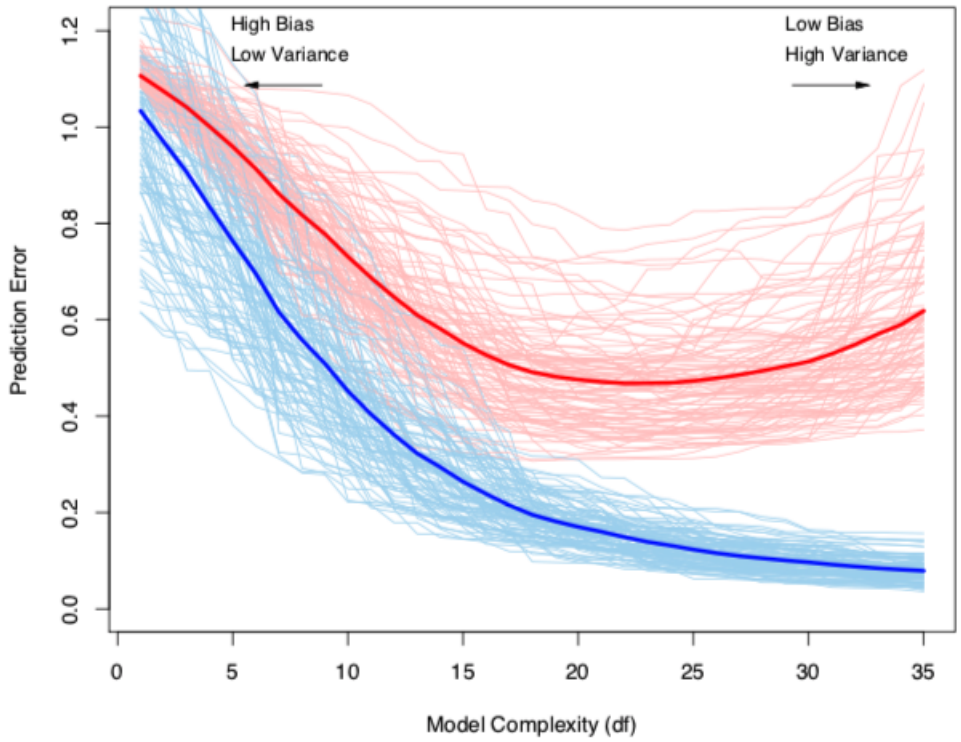


Figure 11.1: Behaviour of test sample and training sample error as the model complexity is varied. The light blue curves show the training error $\bar{\text{err}}$, while the light red curves show the conditional test error $\text{Err}_{\mathcal{T}}$ for 100 training sets of size 50 each, as the model complexity is increased. The solid curves show the expected test error Err and the expected training error $\text{E}[\bar{\text{err}}]$.

model complexity that gives minimum expected test error. The training error is not reliable because it always decreases with the model complexity and with too much complexity it looses the capability to generalize.

The same holds for categorical variables (introduced in 11.1). Typical loss functions are:

$$\begin{aligned} &\mathrm{L}(\mathrm{G}, \hat{\mathrm{G}}) = \mathrm{I}(\mathrm{G} \neq \hat{\mathrm{G}}(\mathrm{X})) \qquad \textit{0-1 loss}, \\ &\mathrm{L}(\mathrm{G}, \hat{\mathrm{p}}(\mathrm{X})) = -2 \sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{I}(\mathrm{G} = \mathrm{k}) \log \hat{\mathrm{p}}_{\mathrm{k}}(\mathrm{X}) \qquad \textit{-2} \times \textit{ log-likelihood}, \end{aligned} \tag{11.5}$$

where the quantity $-2 \times \log$ –likelihood is generally referred as deviance. The "–2", as already said, in the definition makes the log-likelihood loss for the Gaussian distribution match squared-error loss. A number of methods for estimating the expected test error for a model exists. Typically our model will have a tuning parameter or parameters α and so we can write our predictions as $\hat{\mathrm{f}}(\mathrm{x})_\alpha$ The tuning parameter varies the complexity of our model, and we wish to find the value of α that minimizes error, that is, produces the minimum of the average test error curve. There are two goals that one might want to achieve:

Definition 11.4. model selection: estimating the performance of different models in order to choose the best one.

Definition 11.5. model assessment: having chosen a final model, estimate its prediction error on new data.

If we are in a data-rich situation, the best approach for both problems is to randomly divide the dataset into three parts: a training set, a validation set, and a test set. The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model. Ideally, the test set should be kept in a "vault", and be brought out only at the end of the data analysis. It is difficult to give a general rule on how to choose the number of observations in each of the three parts, as this depends on the signal-to- noise ratio in the data and the training sample size. A typical split might be 50% for training, and 25% each for validation and testing.

The following methods allows to work in a situation with insufficient data. The methods can be distinguished in analytical (AIC, BIC, MDL, SRM) or by efficiently sample re-use (cross-validation and boot-strap). Typically the

more complex we make the model f, the lower the (squared) bias but the higher the variance.

11.1 Optimism of the training error rate

Given a training set τ the generalization error of a model is:

$$\mathrm{Err}_\tau = \mathbf{E}_{\mathrm{X}^0,\mathrm{Y}^0}\left[\mathrm{L}(\mathrm{Y}^0, \hat{\mathrm{f}}(\mathrm{X}^0))|\mathrm{tau}\right] \tag{11.6}$$

where the point $(\mathrm{X}^0, \mathrm{Y}^0)$ is a new test data point. Averaging over training sets τ (i.e., averaging over the same type of models $\hat{\mathrm{f}}$ but trained with different trainsets belonging to the same distribution):

$$\mathrm{Err} = \mathbf{E}_\tau \mathbf{E}_{\mathrm{X}^0,\mathrm{Y}^0}\left[\mathrm{L}(\mathrm{Y}^0, \hat{\mathrm{f}}(\mathrm{X}^0))|\mathrm{tau}\right] \tag{11.7}$$

The training error $\bar{\mathrm{err}} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{L}(\mathrm{y_i}, \hat{\mathrm{f}}(\mathrm{x_i}))$ will be less than the true error, because the same data is being used to fit the method and assess its error. A fitting method typically adapts to the training data, and hence the apparent or training error $\bar{\mathrm{err}}$ will be an overly optimistic estimate of the generalization error Err_τ, which can be thought as *extra-sample error*.

The nature of the optimism in the training error can be explained considering the *in-sample error*:

$$\mathrm{Err}_{\mathrm{in}} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{E}_{\mathrm{Y}^0}\left[\mathrm{L}(\mathrm{Y}^0_{\mathrm{i}}, \hat{\mathrm{f}}(\mathrm{x_i}))|\tau\right] \tag{11.8}$$

To explain it better consider the same input points $\mathrm{x_i}, \quad \mathrm{i} = 1, \cdots, \mathrm{N}$. In the training set for these points we got a given set of values, while on new data we get a slightly different values Y^0 due to the random process involved (for example the noise considering the additive model). The model $\hat{\mathrm{f}}$ have been trained using the training set, so for its prediction for those points will be close to the values seen in the training set (influenced by randomness) and it cannot foresee the non-deterministic part in the new samples. We define the optimism as the difference between the real $\mathrm{Err}_{\mathrm{in}}$ and the training error:

$$\mathrm{op} \equiv \mathrm{Err_in} - \bar{\mathrm{err}} \tag{11.9}$$

and this is typically positive since the second term is generally biased downward. Averaging the optimism:

$$\omega = \mathbf{E}_{\mathrm{y}}(\mathrm{op}) \tag{11.10}$$

where the notation $\mathbf{E}_{\mathrm{y}}$ has been used instead of $\mathbf{E}_{\tau}$ because the predictors (the inputs) are fixed, while the outputs vary.

For squared error, 0-1 loss functions and other loss functions one can show:

$$\omega = \frac{2}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{Cov}(\hat{\mathrm{y}}_{\mathrm{i}}, \mathrm{y}_{\mathrm{i}}) \tag{11.11}$$

Thus the amount by which err underestimates the true error depends on how strongly y_{i} affects its own prediction. The harder we fit the data, the greater the covariance will be, thereby increasing the optimism. We have the relationship:

$$\mathbf{E}_{\mathrm{y}}(\mathrm{Err}_{\mathrm{in}}) = \mathbf{E}_{\mathrm{y}}(\bar{\mathrm{err}}) + \frac{2}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{Cov}(\hat{\mathrm{y}}_{\mathrm{i}}, \mathrm{y}_{\mathrm{i}}) \tag{11.12}$$

This expression simplifies if $\hat{\mathrm{y}}_{\mathrm{i}}$ is obtained by a linear fit with d inputs or basis functions. For example for the additive model $\mathrm{Y} = \mathrm{f}(\mathrm{X}) + \epsilon$:

$$\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{Cov}(\hat{\mathrm{y}}_{\mathrm{i}}, \mathrm{y}_{\mathrm{i}}) = \mathrm{d}\sigma_{\epsilon}^2 \tag{11.13}$$

and so

$$\mathbf{E}_{\mathrm{y}}(\mathrm{Err}_{\mathrm{in}}) = \mathbf{E}_{\mathrm{y}}(\bar{\mathrm{err}}) + \frac{2\mathrm{d}}{\mathrm{N}}\sigma_{\epsilon}^2 \tag{11.14}$$

We can use these expressions to select the effective number of parameters. The optimism increases with the number d of inputs or basis functions we use, but decreases as the training sample size increases

An obvious way to estimate prediction error is to estimate the optimism and then add it to the training error. The methods such as Cp, AIC, BIC and others work in this way, for a special class of estimates that are linear in their parameters. In contrast, cross-validation and bootstrap methods are direct estimates of the extra-sample error. These general tools can be used with any loss function, and with nonlinear, adaptive fitting techniques. In-sample error is not usually of direct interest since future values of the features

are not likely to coincide with their training set values. But for comparison between models, in-sample error is convenient and often leads to effective model selection. The reason is that the relative (rather than absolute) size of the error is what matters.

11.2 C_p metric

One can estimate the in-sample error as

$$\hat{\mathrm{Err}} = \bar{\mathrm{err}} + \hat{\omega} \tag{11.15}$$

where $\hat{\omega}$ is an estimate of the average optimism. Using 11.14:

$$C_p = \bar{\mathrm{err}} + \frac{2d}{N}\hat{\sigma}_\epsilon^2 \tag{11.16}$$

where $\hat{\sigma}_\epsilon^2$ is an estimate of the noise variance.

The model with lowest C_p should be selected.

11.3 Akaike Information Criterion

Suppose we have models $\mathcal{M}_1, \cdots, \mathcal{M}_k$ where each model is a set of densities:

$$\mathcal{M}_j = \left\{ p(y; \theta_j) : \theta_j \in \Theta_j \right\}. \tag{11.17}$$

and we have data $Y_1, \cdots, Y_n$ drawn from an unknown density f that we do not know if it is in any of the models. Let $\hat{\theta}_j$ be the Maximum Likelihood estimation from the model j. Using this we can estimate f with $\hat{p}_j(y) = p(y, \hat{\theta}_j)$. The quality of $\hat{p}_j(y)$ (i.e., its distance form f) can be measured by the **Kullback-Leibler distance**:

$$\begin{aligned} K(p, \hat{p}_j) &= \int p(y) \log \frac{p(y)}{\hat{p}_j(y)} dy = \\ &= \int p(y) \log p(y) dy - \int p(y) \log \hat{p}_j(y) dy \end{aligned} \tag{11.18}$$

We want to minimize this distance. The first term is constant while we can act on $\hat{p}_j(y)$. Since it is a probability (i.e. $\hat{p}_j(y)$ is at most one, minimizing the distance is equivalent to maximizing the second term:

$$K_j = \int p(y) \log \hat{p}_j(y) dy \tag{11.19}$$

Intuitively one estimate of K_j might be:

$$\bar{K}_j = \frac{1}{N}\sum_{i=1}^{N} \log p_j(Y_i, \hat{\theta}_j) = \frac{\ell_j(\hat{\theta}_j)}{N} \tag{11.20}$$

i.e., assuming a uniform distribution. This estimation is very biased because data are being used twice: once for the MLE and the second one to calculate the integral. So we have $\bar{K}_j \neq K_j$. We want to find such difference.

Let us drop the index j (i.e., we are considering just one model and not a set) and just use K and $\bar{K}$. Let θ_0 be the parameters that minimize $K(f, p(\cdot, \theta))$. So $p(y, \theta_0)$ is the closest density in the model to the true density. Let $\ell(y, \theta) = \log p(y, \theta)$ and let us define the **score** as:

$$s(y, \theta) = \frac{\partial \log p(y, \theta)}{\partial \theta} \tag{11.21}$$

and let $H(y, \theta)$ be the matrix of second derivatives. Let

$$Z_N = \sqrt{N}\Big(\hat{\theta} - \theta_0\Big) \tag{11.22}$$

and recall (?) that :

$$Z_N \to \mathcal{N}(0, J^{-1}VJ^{-1}) \tag{11.23}$$

where

$$\begin{aligned} J &= -\mathbf{E}\left[H(Y, \theta_0\right] \\ V &= \mathrm{Var}(s(Y, \theta_0)) \end{aligned} \tag{11.24}$$

Let

$$S_N = \frac{1}{N}\sum_{i=1}^{N} s(Y_i, \theta_0) \tag{11.25}$$

By the **Central Limit Theorem**, and recalling that θ_0 is the value that minimizes K (if it is a convex function then for this value the score is 0 i.e., 0 mean):

$$\sqrt{N}S_N \to \mathcal{N}(0, V) \tag{11.26}$$

since

$$\begin{aligned}\mathrm{Var}\left[\sqrt{\mathrm{N}}\mathrm{S_n}\right] &= \mathbf{E}\left[\left(\sqrt{\mathrm{N}}\mathrm{S_n} - \cancel{\mu_{\mathrm{S_n}}}\right)^2\right] = \\ &= \mathrm{N}\mathbf{E}\left[\left(\frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\mathrm{s}(\mathrm{Y_i},\theta_0)\right)^2\right] = \frac{1}{\mathrm{N}}\mathbf{E}\left[\left(\sum_{\mathrm{i}=1}^{\mathrm{N}}\mathrm{s}(\mathrm{Y_i},\theta_0)\right)^2\right] = \\ &= \frac{1}{\mathrm{N}}\left(\sum_{\mathrm{i}=1}^{\mathrm{N}}\mathbf{E}\left[\mathrm{s}(\mathrm{Y_i},\theta_0)\right]^2\right) = \frac{1}{\mathrm{N}}\mathrm{NV} = \mathrm{V}\end{aligned} \tag{11.27}$$

Now J is a scalar, considering the distribution $\mathrm{JZ_N}$ where $\mathrm{Z_N} \to \mathcal{N}(0, \mathrm{J^{-1}VJ^{-1}})$, the result is still a distribution with 0. For the variance let us use the property:

$$\begin{aligned}&\mathrm{Var}\left[\mathrm{JZ_N}\right] = \mathrm{J}(\mathrm{J^{-1}VJ^{-1}})\mathrm{J^T} = \mathrm{V} \\ &\Rightarrow \mathrm{JZ_N} \sim \mathcal{N}(0, \mathrm{V})\end{aligned} \tag{11.28}$$

By using **Taylor series expansion**:

$$\mathrm{K} \approx \int \mathrm{p(y)}\left(\log \mathrm{p}(\mathrm{y},\theta_0) + \cancel{\left(\hat{\theta} - \theta_0\right)^{\mathrm{T}}\mathrm{s}(\mathrm{y},\theta_0)} + \frac{1}{2}\left(\hat{\theta} - \theta_0\right)^{\mathrm{T}}\mathrm{H}(\mathrm{y},\theta_0)\left(\hat{\theta} - \theta_0\right)\right)\mathrm{dy} \tag{11.29}$$

The second term $\left(\hat{\theta} - \theta_0\right)^{\mathrm{T}}\mathrm{s}(\mathrm{y},\theta_0)$ is 0 because the score has 0 mean. Defining

$$\mathrm{K_0} = \int \mathrm{p(y)} \log \mathrm{p}(\mathrm{y},\theta_0)\mathrm{dy} \tag{11.30}$$

we have:

$$\begin{aligned}\mathrm{K} &\approx \mathrm{K_0} + \frac{1}{\sqrt{\mathrm{N}}}\mathrm{Z_N^T}\int \mathrm{H}(\mathrm{y},\theta_0)\mathrm{p(y)dy}\frac{1}{\sqrt{\mathrm{N}}}\mathrm{Z_N} = \\ &= \mathrm{K_0} + \frac{1}{\sqrt{\mathrm{N}}}\mathrm{Z_N^T}\int \mathrm{H}(\mathrm{y},\theta_0)\mathrm{p(y)dy}\frac{1}{\sqrt{\mathrm{N}}}\mathrm{Z_N} = \mathrm{k_0} - \frac{1}{2\mathrm{N}}\mathrm{Z_N^T}\mathrm{JZ_N}\end{aligned} \tag{11.31}$$

Now applying the **Taylor series expansion** to hatK:

$$\bar{\mathrm{K}} \approx \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\left(\ell(\mathrm{Y_i},\theta_0) + \left(\hat{\theta} - \theta_0\right)^{\mathrm{T}}\mathrm{s}(\mathrm{Y_i},\theta_0)\right) + \frac{1}{2}\left(\hat{\theta} - \theta_0\right)^{\mathrm{T}}\mathrm{H}(\mathrm{Y_i},\theta_0)\left(\hat{\theta} - \theta_0\right) \tag{11.32}$$

Defining

$$\begin{aligned} \mathrm{A_N} &= \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} \ell(\mathrm{Y_i}, \theta_0) - \mathrm{K_0} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} (\ell(\mathrm{Y_i}, \theta_0) - \mathrm{K_0}) \\ \mathrm{J_N} &= -\frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{H}(\mathrm{Y_i}, \theta_0) - \xrightarrow{\mathrm{P}} \mathrm{J} \end{aligned} \tag{11.33}$$

we have

$$\begin{aligned} \bar{\mathrm{K}} &\approx \mathrm{K_0} + \mathrm{A_N} + (\hat{\theta} - \theta_0)\mathrm{S_N} - \frac{1}{2\mathrm{N}}\mathrm{Z_N^T J_N Z_N} \approx \\ &\approx \mathrm{K_0} + \mathrm{A_N} + \frac{\mathrm{Z_N^T}}{\mathrm{N}}\sqrt{\mathrm{N}}\mathrm{J}^{-1}\mathrm{J S_N} - \frac{1}{2\mathrm{N}}\mathrm{Z_N^T J Z_N} = \\ &= \mathrm{K_0} + \mathrm{A_N} + \frac{\mathrm{Z_N^T J Z_N}}{\mathrm{N}} - \frac{1}{2\mathrm{N}}\mathrm{Z_N^T J Z_N} \end{aligned} \tag{11.34}$$

where 11.28 has been used again ($\mathrm{J}^{-1}\mathrm{S_N} \rightarrow \mathrm{J}^{-1}\mathcal{N}(0, \mathrm{V}) = \mathcal{N}(0, \mathrm{J}^{-1}\mathrm{V}\mathrm{J}^{-1}) = \mathrm{Z_N}$). We need another property. Let ϵ be a random vector with mean μ and variance Σ and let

$$\mathrm{Q} = \epsilon^{\mathrm{T}}\mathrm{A}\epsilon \tag{11.35}$$

with Q called a quadratic form. Then

$$\mathbf{E}\left[\mathrm{Q}\right] = \mathrm{trace}(\mathrm{A}\Sigma) + \mu^{\mathrm{T}}\mathrm{A}\mu \tag{11.36}$$

Now taking the difference:

$$\begin{aligned} \mathbf{E}\left[\mathrm{hat K} - \mathrm{K}\right] = \mathbf{E}\left[\mathrm{A_N}\right] + \mathbf{E}\left[\frac{\mathrm{Z_N^T J Z_N}}{\mathrm{N}}\right] = 0 + \frac{\mathrm{trace}(\mathrm{J}\mathrm{J}^{-1}\mathrm{V}\mathrm{J}^{-1})}{\mathrm{N}} = \\ \frac{\mathrm{trace}(\mathrm{J}^{-1}\mathrm{V}\mathrm{J}^{-1}\mathrm{J})}{\mathrm{N}} = \frac{\mathrm{trace}(\mathrm{J}^{-1}\mathrm{V})}{\mathrm{N}} \end{aligned} \tag{11.37}$$

If the model is correct then $\mathrm{J}^{-1} = \mathrm{V}$ so that $\mathrm{trace}(\mathrm{J}^{-1}\mathrm{V}) = \mathrm{trace}(\mathrm{I}) = \mathrm{d}$:

$$\mathrm{K} \approx \hat{\mathrm{K}} = \bar{\mathrm{K}} - \frac{\mathrm{d}}{\mathrm{N}} \tag{11.38}$$

So the bias is approximately $\frac{\mathrm{d}}{\mathrm{N}}$. Considering the model j:

$$\begin{aligned} \hat{\mathrm{K}}_\mathrm{j} &= \bar{\mathrm{K}}_\mathrm{j} - \frac{\mathrm{d_j}}{\mathrm{N}} \\ \mathrm{AIC(j)} &= 2\mathrm{N}\hat{\mathrm{K}}_\mathrm{j} = \ell_\mathrm{j}(\hat{\theta}_\mathrm{j}) - 2\mathrm{d_j} \end{aligned} \tag{11.39}$$

Maximizing $\hat{\mathrm{K}}_{\mathrm{j}}$ is the same as maximising AIC(j). Constants are not important in the maximization process. With this representation we choose the model with the highest AIC. Other books use the following representation:

$$\mathrm{AIC(j)} = -\frac{2}{\mathrm{N}}\ell_{\mathrm{j}}(\hat{\theta}_{\mathrm{j}}) + 2\frac{\mathrm{d_j}}{\mathrm{N}} \tag{11.40}$$

and with this representation the best model is the one that minimizes the AIC, since the signs have been inverted.

For the Gaussian model, with variance $\sigma_\epsilon^2 = \hat{\sigma}_\epsilon^2$, the AIC statistic is equivalent to $\mathrm{C_p}$.

For non-linear and other complex models we replace d with a measure of the model complexity. Given a set of models f_α, $\bar{\mathrm{err}}(\alpha)$ is the training error and $\mathrm{d}(\alpha)$ the number of parameters:

$$\mathrm{AIC}(\alpha) = \bar{\mathrm{err}}(\alpha) + 2\frac{\mathrm{d}\alpha}{\mathrm{N}}\hat{\sigma}_\epsilon^2. \tag{11.41}$$

11.4 Effective number of parameters

A linear fitting method (linear regression, derived basis sets and smoothing methods with quadratic shrinkage such as ridge regression and cubic smoothing splines) have the form:

$$\hat{\mathbf{y}} = \mathbf{S}\mathbf{y} \tag{11.42}$$

where $\mathbf{S}$ is a $\mathrm{N} \times \mathrm{N}$ matrix. The effective number of parameters is defined as:

$$\mathrm{df}(\mathbf{S}) = \mathrm{trace}(\mathbf{S}) = \mathrm{M} \tag{11.43}$$

since $\mathbf{S}$ is an orthogonal projection matrix onto a basis set spanned by M features. Then instead of d we use trace($\mathbf{S}$).

For models like NN in which we minimize an error function with weight decay penalty (regularization) $\alpha \sum_{\mathrm{m}} \mathrm{w}_{\mathrm{m}}^2$, the effective number of parameters has ethe form:

$$\mathrm{df}(\alpha) = \sum_{\mathrm{m}=1}^{\mathrm{M}} \frac{\theta_{\mathrm{m}}}{\theta_{\mathrm{m}} + \alpha} \tag{11.44}$$

where θ_{m} are the eigenvalues of the Hessian matrix $\partial \mathrm{R}(\omega)/\partial\omega\partial\omega^{\mathrm{T}}$.

11.5 Bayesian Information Criterion

Known also as *Schwarz Criterion*, it is equivalent to the MDL. We choose j to maximize

$$\mathrm{BIC(j)} = \ell_{\mathrm{j}}(\hat{\theta}_{\mathrm{j}}) - \frac{\mathrm{d_j}}{2}\log \mathrm{n} \tag{11.45}$$

It is similar to the AIC but it has a stronger penalization and tends to choose simpler models.

Using priors $\pi_{\mathrm{j}}(\theta_{\mathrm{j}})$ on the parameters θ_{j} and a probability $\mathrm{p_j}$ indicating the probability that $\mathcal{M}_{\mathrm{j}}$ is the true model, by Bayesian theory, indicating with $\mathbf{Z}$ the training set, we have:

$$\mathrm{P}(\mathcal{M}_{\mathrm{j}}|\mathbf{Z}) \propto \mathrm{p}(\mathbf{Z}|\mathcal{M}_{\mathrm{j}})\mathrm{p_j} \tag{11.46}$$

Furthermore

$$\begin{aligned} \mathrm{p}(\mathbf{Z}|\mathcal{M}_{\mathrm{j}}) &= \int \mathrm{p}(\mathbf{Z}|\mathcal{M}_{\mathrm{j}}, \theta_{\mathrm{j}})\pi_{\mathrm{j}}(\theta_{\mathrm{j}})\mathrm{d}\theta_{\mathrm{j}} = \\ &= \int \mathrm{L}(\theta_{\mathrm{j}})\pi_{\mathrm{j}}(\theta_{\mathrm{j}})\mathrm{d}\theta_{\mathrm{j}} \end{aligned} \tag{11.47}$$

So equivalently we choose j to maximize:

$$\log \int \mathrm{L}(\theta_{\mathrm{j}})\pi_{\mathrm{j}}(\theta_{\mathrm{j}})\mathrm{d}\theta_{\mathrm{j}} + \log \mathrm{p_j} \tag{11.48}$$

Using Taylor expansion:

$$\begin{aligned} \log \int \mathrm{L}(\theta_{\mathrm{j}})\pi_{\mathrm{j}}(\theta_{\mathrm{j}})\mathrm{d}\theta_{\mathrm{j}} + \log \mathrm{p_j} &\approx \ell_{\mathrm{j}}(\hat{\theta}_{\mathrm{j}}) - \frac{\mathrm{d_j}}{2}\log \mathrm{n} \\ \Rightarrow \mathrm{BIC(j)} &= \ell_{\mathrm{j}}(\hat{\theta}_{\mathrm{j}}) - \frac{\mathrm{d_j}}{2}\log \mathrm{n} \end{aligned} \tag{11.49}$$

The prior has been ignored because it can be shown that the terms involving the prior are lower orders than the ones appearing in 11.49. Again in some books the signs are inverted.

The difference from AIC and Cross-Validation is that BIC assumes that one of the models is true and that we are trying to find the model most likely to be true in a Bayesian sense. AIC and Cross-Validation tries to find the model that predicts the best.

Under the Gaussian model, assuming the variance σ_ϵ^2 is known, with a squared error loss:

$$\begin{aligned} \bar{\text{err}} &= \sum_{\text{i}} \Big(\text{y}_\text{i} - \hat{\text{f}}(\text{x}_\text{i})\Big)^2 \\ 2\ell_\text{j}(\hat{\theta}_\text{j}) &= -\text{N}\frac{\bar{\text{err}}}{\sigma_\epsilon^2} \\ \text{BIC(j)} &= -\frac{\text{N}}{2\sigma_\epsilon^2}\bar{\text{err}} - \frac{\text{d}_\text{j}}{2}\log \text{n} = \frac{\text{N}}{2\sigma_\epsilon^2}\left[\frac{\text{d}_\text{j}}{\text{N}}\sigma_\epsilon^2 \log \text{N} - \bar{\text{err}}\right] \end{aligned} \tag{11.50}$$

Hence BIC is proportional to AIC with the factor 2 replaced by $\log \text{N}$.

11.6 Minimum Description Length

The minimum description length (MDL) principle is a formalization of Occam's razor in which the best hypothesis (a model and its parameters) for a given set of data is the one that leads to the best compression of the data. The Minimum Description Length is the minimum number of bits, or the minimum of the sum of the number of bits required to represent the data and the model. $\cdots$

11.7 Cross-Validation

It is the simplest and most widely used widely used method for estimating prediction error is cross-validation. This method directly estimates the expected extra-sample error. It consists of dividing data set in parts: a training set and using an independent test sample from the joint distribution. So one set is used for training and the other to avoid overfitting (to validate the model).

11.7.1 K-fold cross-validation

K-fold cross-validation uses part of the available data to fit the model, and a different part to test it. For the $\text{k}-\text{th}$ part, we fit the model using the other $\text{K}-1$ parts for training and we calculate the prediction error of the fitted model on the $\text{k}-\text{th}$ part. We do this for all $\text{k} = 1, \cdots, \text{K}$, fitting the model resulting from the previous step using the new partition of data. Typical choices of K are 5 or 10. $\text{K} = \text{N}$ case is known as live-one-out

cross-validation. The higher value of K leads to less biased model (but large variance might lead to overfit), where as the lower value of K is similar to the train-test split approach we saw before.

11.7.2 Cross validation according to "Hands on Machine Learning"

Generally a model can be evaluated using a training set and a test set. When one has to choose a hyperparameter a **validation set** of data is used. Models with different parameters are trained using the training set, the model and hyperparameters that perform best according to the validation test are selected and one is satisfied by the performances, finally run a single test on the test set.

When evaluating the hyperparameter on the test-set directly, the problem is that the test-set is being used multiple times: to select the hyperparameters and to finally evaluate the model. The hyperparameters were chosen for this specific test-set so it is clear the error on this specific set is reduced.

This however requires lots of data. To avoid this cross validation is used: the training set is split into complementary subsets, and each model is trained against a different combination of these subsets and validated against the remaining parts. Once the model type and hyperparameters have been selected, a final model is trained using these hyperparameters on the full training set, and the generalized error is measured on the test set.

11.8 Bootstrap Methods

The general idea is to randomly draw datasets with replacement from training data. Samples are constructed by drawing observations from a large data sample one at a time and returning them to the data sample after they have been chosen. This allows a given observation to be included in a given small sample more than once. This approach to sampling is called sampling with replacement. This is done B times, producing B datasets.

For the process we need to decide the number of bootstrap samples and a sample size. Then for each boostrap sample, we refit the model to each of the bootstrap datasets and examine the behaviour of the fits over the B replications.

[From wiki:] As an example, assume we are interested in the average (or mean) height of people worldwide. We cannot measure all the people in the

global population, so instead we sample only a tiny part of it, and measure that. Assume the sample is of size N; that is, we measure the heights of N individuals. From that single sample, only one estimate of the mean can be obtained. In order to reason about the population, we need some sense of the variability of the mean that we have computed. The simplest bootstrap method involves taking the original data set of N heights, and, using a computer, sampling from it to form a new sample (called a 'resample' or bootstrap sample) that is also of size N. The bootstrap sample is taken from the original by using sampling with replacement (e.g. we might 'resample' 5 times from [1,2,3,4,5] and get [2,5,4,4,1]), so, assuming N is sufficiently large, for all practical purposes there is virtually zero probability that it will be identical to the original "real" sample. This process is repeated a large number of times (typically 1,000 or 10,000 times), and for each of these bootstrap samples we compute its mean (each of these are called bootstrap estimates). We now can create a histogram of bootstrap means. This histogram provides an estimate of the shape of the distribution of the sample mean from which we can answer questions about how much the mean varies across samples.

12 Additive models

A linear model has the form $\alpha + f_1(X_1) + f_2(X_2) + f_3(X_3) + \cdots$, where X are the predictors.

12.1 Tree based methods

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one.

We restrict attention to recursive binary partitions. We first split the space into two regions, and model the response by the mean of Y in each region. We choose the variable and split-point to achieve the best fit. Then one or both of these regions are split into two more regions, and this process is continued, until some stopping rule is applied.

For example we first split at $X_1 = t_1$. Then the region $X_1 \leq t_1$ is split at $X_2 = t_2$ and the region $X_1 > t_1$ is split at $X_1 = t_3$ and so on. To each sub-region we assign a constant as output:

$$\hat{f}(X) = \sum_{m=1}^{M} c_m I\{(\mathbf{x} \in \mathcal{R}_m\} \tag{12.1}$$

An advantage of this representation is its interpretability. When more than 2 inputs are involved, it is difficult to represent the partition of the input space.

12.1.1 Regression Trees

The algorithm needs to decide automatically the splitting variables and points. We can start defining the values of the constant of each region:

$$\hat{c}_m = \text{ave}(y_i | x_i \in \mathcal{R}_m) \tag{12.2}$$

Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm. Consider a splitting variable j and split point s and define the pair of half-planes:

$$R_1(j, s) = \{X | X_j \leq s\}. \quad R_2(j, s) = \{X | X_j > s\} \tag{12.3}$$

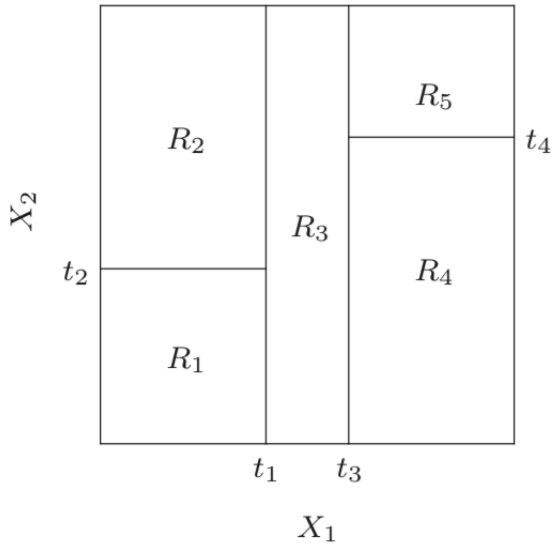


Figure 12.1: 2D-input space binary tree partition.

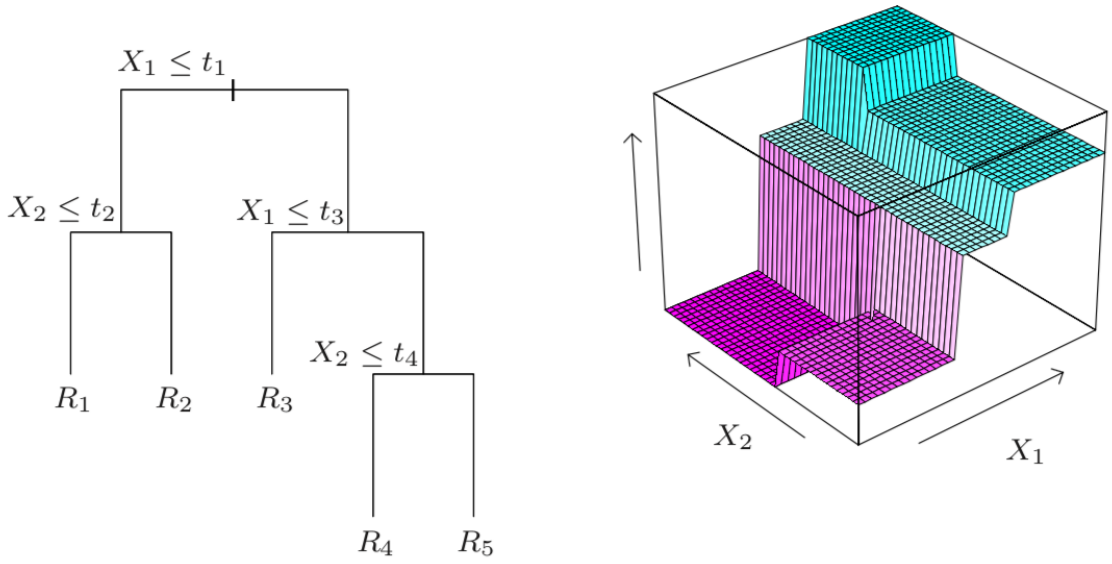


Figure 12.2: Another representation of the binary tree on the left. On the right the 3D resulting shape of the function.

Then we seek the splitting variable j and split point s that solve:

$$\begin{aligned}
&\min_{\mathrm{j,s}}\left[\min_{\mathrm{c_1}}\sum_{\mathrm{x_i}\in\mathcal{R}_1(\mathrm{j,s})}(\mathrm{y_i}-\mathrm{c_1})^2+\min_{\mathrm{c_2}}\sum_{\mathrm{x_i}\in\mathcal{R}_2(\mathrm{j,s})}(\mathrm{y_i}-\mathrm{c_2})^2\right] \\
&\hat{\mathrm{c}}_1=\mathrm{ave}(\mathrm{y_i}|\mathrm{x_i}\in\mathrm{R_1(j,s)}) \\
&\hat{\mathrm{c}}_2=\mathrm{ave}(\mathrm{y_i}|\mathrm{x_i}\in\mathrm{R_2(j,s)})
\end{aligned} \tag{12.4}$$

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j,s) is feasible. Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. Then this process is repeated on all of the resulting regions.

Tree size is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data. One approach would be to split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold. This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it. The preferred strategy is to grow a large tree $\mathrm{T_0}$, stopping the splitting process only when some minimum node size (say 5) is reached. Then this large tree is pruned using **cost-complexity pruning**.

The **cost-complexity pruning** is the following. We define a subtree T of $\mathrm{T_0}$. obtained by collapsing any number of its internal nodes. We index the terminal nodes by m. let $|\mathrm{T}|$ denote the number of terminal nodes in T and

$$\begin{aligned}
\mathrm{N_m}&=\#\{\mathrm{x_i}\in\mathcal{R}_\mathrm{m}\} \\
\hat{\mathrm{c}}_\mathrm{m}&=\frac{1}{\mathrm{N_m}}\sum_{\mathrm{x_i\ in}\mathcal{R}_\mathrm{m}}\mathrm{y_i} \\
\mathrm{Q_m(T)}&=\frac{1}{\mathrm{N_m}}\sum_{\mathrm{x_i}\in\mathrm{R_m}}(\mathrm{y_i}-\hat{\mathrm{c}}_\mathrm{m})^2
\end{aligned} \tag{12.5}$$

We define the cost complexity criterion:

$$\mathrm{C}_\alpha(\mathrm{T})=\sum_{\mathrm{m}=1}^{|\mathrm{T}|}\mathrm{N_m}\mathrm{Q_m(T)}+\alpha|\mathrm{T}| \tag{12.6}$$

The idea is to find for each α the subtree T_α of T_0 to minimize such a function. For each α there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$. To find T_α we use the **weakest link pruning**: we collapse the internal node that produces the smallest per-node increase and we continue umtil we produce the single-node (root) tree. So we have a sequence of subtrees that must contain T_α. We choose α to minimize the cross-validated sum of squares, obtaining the tree T_α.

12.2 Classification trees

The procedure is very similar. In a node m representing the region R_m with N_m observations, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{12.7}$$

We classify the observations in node m to class

$$k(m) = \text{argmax}_k \hat{p}_{mk}, \tag{12.8}$$

the majority class in node m. Different measures $Q_m(T)$ of node impurity exist

$$\begin{aligned}
&\textbf{Misclassification error} && \frac{1}{N_m} \sum_{i \in \mathcal{R}_m} I(y_i \neq k(m)) = 1 - \hat{p}_{km} \\
&\textbf{Gini index} && \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \\
&\textbf{Cross-entropy or deviance} && -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}
\end{aligned} \tag{12.9}$$

These trees are referred as CART (classification and regression trees).

12.2.1 Linear Combination splits

Rather than using splits of the type $X_j \leq s$ we can use splits of the type $\sum a_j X_j \leq s$. The weights a_j and split point s are optimized to minimize the relevant criterion such as the Gini index.

12.2.2 Limitations of trees

One major problem of trees is **the instability (i.e., high variance)**. A small change in the data can result in a very different series of splits, making interpretation somewhat precarious. The major reason is the hierarchical nature of the process: the effect of an error in the top split is propagated down to all the next splits. The problem can be attenuated but not completely solved.

Another problem of trees is the **lack of smoothness**. This is not a problem in classification but it is in regression.

Also they are not good at capturing additive structures, such as $Y = c_1(X_1 < t_1) + c_1(X_1 < t_1) + \epsilon$ where the noise is 0 mean. Then a binary tree might make its first split on X_1 near t_1. At the next level down it would have to split both nodes X_2 at t_2 in order to capture the additive structure. This is achieved with sufficient data, but it it is a fragile mechanism and can fail in case of many additive effects.

12.3 Patient rule induction method (PRIM)

Also PRIM finds boxes in the feature space, but it seeks boxes in which the response average is high. Hence it looks for maxima in the target function, an exercise known as bump hunting. The main box construction method in PRIM works starting with a box containing all of the data. The box is compressed along one face by a small amount, and peeling off a constant portion of data denoted by α. Typically α is 0.05 and 0.1 of the remaining points at each stage. The face chosen for compression is the one resulting in the largest box mean, after the compression is performed. Then the process is repeated, stopping when the current box contains some minimum number of data points.

After the top-down sequence is computed, PRIM reverses the process, expanding along any edge, if such an expansion increases the box mean. This is called pasting. The result of these steps is a sequence of boxes, with different numbers of observation in each box. Cross-validation, combined with the judgement of the data analyst, is used to choose the optimal box size. Denote by B_1 the points in this first box. These points are now removed from the training set and we repeat the two processes (top-down peeling and bottom-up pasting) several times, obtaining for each iteration a new box B_i.

Each box is defined by a set of rules involving a subset of predictors such as:

$$(a_1 \leq X_1 \leq b_1) \qquad (b_1 \leq X_3 \leq b_2) \tag{12.10}$$

PRIM can handle a categorical predictor by considering all partitions of the predictor, as in CART. Missing values are also handled in a manner similar to CART. PRIM is designed for regression

---

Start with all of the training data, and a maximal box containing all of the data;

Consider shrinking the box by compressing one face, so as to peel off the proportion α of observations having either the highest values of a predictor X_j or the lowest. Choose the peeling that produces the highest response mean in the remaining box;

Repeat step 2 until some minimal number of observations (say 10) remain in the box;

Expand the box along any face, as long as the resulting box mean increases;

Steps 1-4 give a sequence of boxes, with different numbers of observations in each box. Use cross-validation to choose a member of the sequence. Call the box B_1.

Remove the data in the box B_1 from the dataset and repeat steps 2-5 to obtain a second box. and continue to get as many boxes as desired.

PRIM is designed for binary regression. There is no simple way to deal with $k > 2$ classes simultaneously: it is possible to run PRIM separately for each class versus a baseline class.In any case, the ability of PRIM to be more patient should help the top-down greedy algorithm find a better solution.

12.4 Multivariate Adaptive Regression Splines (MARS)

It is a procedure well-suited for high-dimensional problems. MARS uses expansions in piecewise linear basis functions of the form $(x-t)_+$ and $(t-x)_+$.

Each function is piecewise linear, with a knot at the value t_1. We call the two functions a **reflected pair**. The idea is to form reflected pairs for each input X_j with knots at each observed value x_{ij} of that input. If all of the input values are distinct, there are 2Np basis functions altogether.

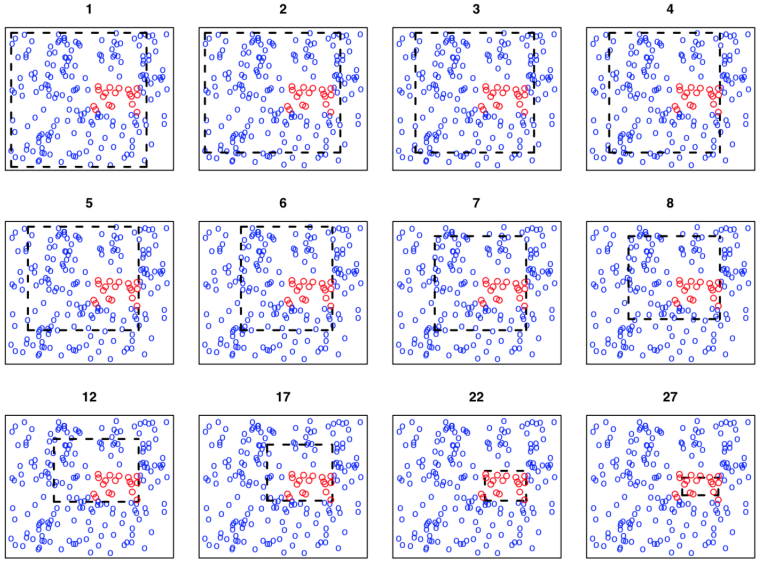


Figure 12.3: Steps in PRIM.

The model-building strategy is like a forward stepwise linear regression, but instead of using the original inputs, we are allowed to use functions from the set $\mathcal{C}$ and their products:

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}_{t \in \{x_{1j}, \cdots, x_{Nj}\}, j=1, \cdots, p} \tag{12.11}$$

Thus the model has the form:

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X) \tag{12.12}$$

where each $h_m(X)$ is a function in $\mathcal{C}$ or a product of two or more such functions. Given the choice for $h_m(X)$, the coefficients are estimated with the residual sum of squares. We start with only a constant $h_0(X) = 1$ and all the functions are candidate functions. At each step we consider as a new basis function pair all products of a function h_m in the model set $\mathcal{M}$ with one of the reflected pairs in $\mathcal{C}$. We add to the model $\mathcal{M}$ the term of the form:

$$\hat{\beta}_{M+1} h_\ell(X)(X_j - t)_+ + \hat{\beta}_{M+2} h_\ell(X)(t - X_j)_+, h_\ell \in \mathcal{M} \tag{12.13}$$

that produces the largest decrease in training error. For example, at the first stage we consider adding to the model a function of the form: $\hat{\beta}_1(X_2 - x_{72})_+ +$

$\hat{\beta}_2(x_{72} - X_2)_+$. Then this pair of basis functions is added to the set $\mathcal{M}$ and at the next stage we consider including a pair of products in the form:

$$h_m(X)(X_j - t)_+ \qquad \text{and} \qquad h_m(X)(t - X_j)_+, t \in \{xij\} \tag{12.14}$$

where for h_m we have the choices:

$$\begin{aligned} &h_0(X) = 1 \\ &h_1(X) = (X_2 - x_{72})_+, \quad or \\ &h_2(X) = (x_{72} - X_2)_+ \end{aligned} \tag{12.15}$$

At the end of this process we have a large model. This model typically overfits the data, and so a backward deletion procedure is applied. The term whose removal causes the smallest increase in residual squared error is deleted from the model at each stage, producing an estimated best model $\hat{f}_\lambda$ of each size (number of terms) λ. One could use cross-validation to estimate the optimal value of λ, but for computational savings the MARS procedure instead uses generalized cross-validation:

$$GCV(\lambda) = \frac{\sum_{i=1}^{N}(y_i - \hat{f}_\lambda(x_i))^2}{(1 - \frac{M(\lambda)}{N})^2} \tag{12.16}$$

The value $M(\lambda)$ is the effective number of parameters in the model: this accounts both for the number of terms in the models, plus the number of parameters used in selecting the optimal positions of the knots.

Some mathematical and simulation results suggest that one should pay a price of three parameters for selecting a knot in a piecewise linear regression. Thus if there are r linearly independent basis functions in the model, and K knots were selected in the forward process, the formula is $M(\lambda) = r + cK$, where $c = 3$.

The MARS method and algorithm can be extended to handle classification problems. Several strategies have been suggested. For two classes, one can code the output as 0/1 and treat the problem as a regression; we did this for the spam example. For more than two classes, one can use the indicator response approach. One codes the K response classes via 0/1 indicator variables, and then performs a multi-response MARS regression. There are, however, potential masking problems with this approach.

12.5 Hierarchical Mixture of Experts

This can be viewed as a variant of tree-based methods. The main difference is that the tree splits are not hard decisions but rather soft probabilistic ones. At each node an observation goes left or right with probabilities depending on its input values. In an HME, a linear (or logistic regression) model is fit in each terminal node, instead of a constant as in CART (the split can be multiway, not only binary).

The terminal nodes are called **experts**, and the non-terminal nodes are called **gating networks**. The idea is that each expert provides an opinion (prediction) about the response, and these are combined together by the gating networks. The model is formally a mixture model, and the two-level model in the figure can be extend to multiple levels, hence the name hierarchical mixtures of experts.

12.5.1 How HME works

Consider the data $(\mathrm{x_i}, \mathrm{y_i}), \quad \mathrm{i} = 1, \cdots, \mathrm{N}$ either for regression or classification problem. The **top-gating** network has the output:

$$\mathrm{g_j}(\mathbf{x}, \gamma_\mathrm{j}) = \frac{\mathrm{e}^{\gamma_\mathrm{j}^\mathrm{T}\mathbf{x}}}{\sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{e}^{\gamma_\mathrm{k}^\mathrm{T}\mathbf{x}}}, \qquad \mathrm{j} = 1, \cdots, \mathrm{K} \tag{12.17}$$

where γ_j is a vector of unknown parameters. Each $\mathrm{g_j}$ is the probability of assigning an observation with feature vector $\mathbf{x}$ to the j–th branch (soft K-way split). At second level the **gating networks** have similar form:

$$\mathrm{g}_{\ell|\mathrm{j}}(\mathbf{x}, \gamma_{\mathrm{j}|\ell}) = \frac{\mathrm{e}^{\gamma_{\mathrm{j}|\ell}^\mathrm{T}\mathbf{x}}}{\sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{e}^{\gamma_{\mathrm{j}|\mathrm{k}}^\mathrm{T}\mathbf{x}}}, \qquad \ell = 1, \tag{12.18}$$

This is the probability of assignment to the l – th branch, given assignment to the j – th branch at the level above. At each expert (terminal node), we have a model for the response variable of the form:

$$\mathrm{Y} \sim \Pr(\mathrm{y}|\mathbf{x}, \theta_{\mathrm{j}\ell} \tag{12.19}$$

which is different according to the type of problem. For **regression** the Gaussian linear model is used with $\theta_{\mathrm{j}\ell} = (\beta_{\mathrm{j}\ell}, \sigma^2_{\mathrm{jl}}$:

$$\mathrm{Y} = \beta_{\mathrm{j}\ell}^\mathrm{T}\mathrm{x} + \mathrm{epsilon}, \qquad \mathcal{N} \sim (0, \sigma^2_{\mathrm{jl}}). \tag{12.20}$$

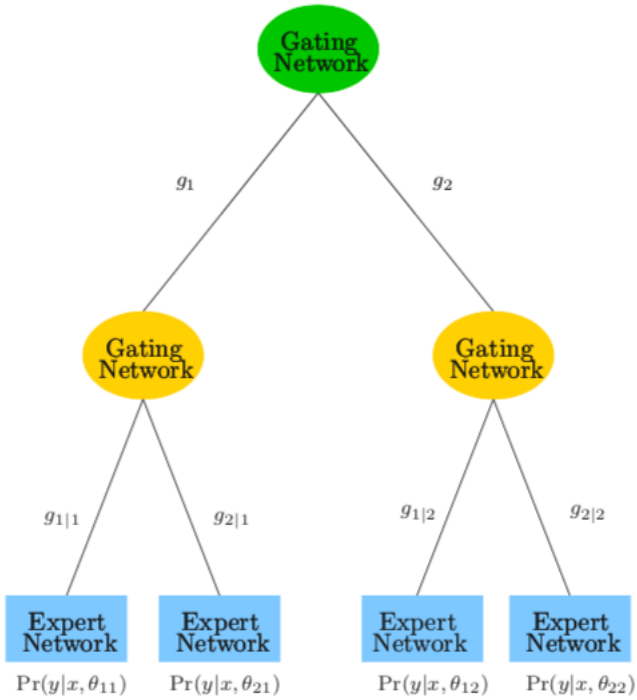


Figure 12.4: A two-level HME model.

For **classification** the linear logistic regression is used:

$$\Pr(\mathrm{Y}=1|\mathbf{x},\theta_{\mathrm{j}\ell}=\frac{1}{1+\mathrm{e}^{-\theta_{\mathrm{j}\ell}^{\mathrm{T}}\mathbf{x}}} \tag{12.21}$$

Denoting with $\Phi=\{\gamma_{\mathrm{j}},\gamma_{\mathrm{j}\ell},\theta_{\mathrm{j}\ell}\}$ the collections of all parameters, the total probability that $\mathrm{Y}=\mathrm{y}$ is:

$$\Pr(\mathrm{y}|\mathbf{x},\Phi)=\sum_{\mathrm{k}=1}^{\mathrm{K}}\mathrm{g}_{\mathrm{j}}(\mathrm{x},\gamma_{\mathrm{j}})\sum_{\ell=1}^{\mathrm{K}}\mathrm{g}_{\ell|\mathrm{j}}(\mathbf{x},\gamma_{\mathrm{j}\ell})\Pr(\mathrm{y}|\mathbf{x},\theta_{\mathrm{j}\ell}) \tag{12.22}$$

To estimate the parameters we maximize the log-likelihood of the data over Φ. The most convenient method is the EM algorithm.

12.6 Missing data

The usual approach is to fill-in missing values. Roughly speaking, data are missing at random if the mechanism resulting in its omission is independent of its (unobserved) value. A more precise definition is given in Little and Rubin (2002). Suppose y is the response vector and X is the $\mathrm{N}\times\mathrm{p}$ matrix of inputs (some of which are missing). Denote by $\mathrm{X}_{\mathrm{obs}}$ the observed entries in X and let $\mathbf{Z}=(\mathrm{y},\mathrm{X})$, $\mathbf{Z}_{\mathrm{obs}}=(\mathbf{y},\mathbf{X}_{\mathrm{obs}})$. Finally, if $\mathbf{R}$ is an indicator matrix with $\mathrm{ij}-\mathrm{thentry}$ 1 if x_{ij} is missing and zero otherwise, then the data is said to be missing at random (MAR) if the distribution of $\mathbf{R}$ depends on the data $\mathbf{Z}$ only through $\mathrm{Z}_{\mathrm{obs}}$:

$$\Pr(\mathbf{R}|\mathbf{Z},\theta)=\Pr(\mathbf{R}|\mathbf{Z}_{\mathrm{obs}},\theta) \tag{12.23}$$

Data are said to be **missing completely at random (MCAR)** if the distribution$\mathbf{R}$ does not depend on the observed or missing data:

$$\Pr(\mathbf{R}|\mathbf{Z},\theta)=\Pr(\mathbf{R}|\theta) \tag{12.24}$$

For example, if a patient's measurement was not taken because the doctor felt he was too sick, that observation would not be MAR or MCAR.

Assuming the features are missing completely at random, there are a number of ways of proceeding:

1. Discard observations with any missing values.

2. Rely on the learning algorithm to deal with missing values in its training phase.

3. Impute all missing values before training.

Approach 1 can be used if the relative amount of missing data is small, but otherwise should be avoided. Regarding 2, CART is one learning algorithm that deals effectively with missing values, through surrogate splits. MARS and PRIM use similar approaches. In generalized additive modeling, all observations missing for a given input feature are omitted when the partial residuals are smoothed against that feature in the backfitting algorithm, and their fitted values are set to zero. Since the fitted curves have mean zero (when the model includes an intercept), this amounts to assigning the average fitted value to the missing observations. For most learning methods, the imputation approach 3 is necessary. The simplest tactic is to impute the missing value with the mean or median of the non-missing values for that feature.

If the features have at least some moderate degree of dependence, one can do better by estimating a predictive model for each feature given the other features and then imputing each missing value by its prediction from the model, distinctly from the method used to predict y from $\mathbf{x}$.

13 Boosting and Additive Trees

13.1 Boosting methods

It was originally designed for classification problems but it can profitably be extended to regression as well. The motivation for boosting was a procedure that combines the outputs of many "weak" classifiers to produce a powerful "committee".

A weak classifier is one whose error rate is only slightly better than random guessing. The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_(x), \quad m = 1, \cdots, M$. Consider a two-class problem with the output variable coded as $Y \in \{-1, 1\}$. Given a vector of predictors $\mathbf{X}$, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$. The predictions from all of them are then combined

to through a weighted majority of vote to produce the final prediction:

$$G(x) = \text{sign}\left(\sum_{i=1}^{M} \alpha_m G_m(x)\right) \tag{13.1}$$

This is the **AdaBoost** procedure. Here $\alpha_1, \cdots, \alpha_m$ are computed by the boosting algorithm and weight the single contribution. The data modifications at each boosting step consist of applying weights $w_1, \cdots, w_N$ to each of the training observations. Initially all the weights are $1/N$. For successive iterations, the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations. At step m the observations that were misclassified by the classifier $G_{m-1}(x)$ have their weights increased, whereas the weights are decreased for those that were classified correctly. Thus as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence. Each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence.

The **AdaBoost.M1** algorithm, aka **Discrete Adaboost**, is shown in 1. If the base classifier instead returns a real-valued prediction (e.g., a probability mapped to the interval $[-1, 1]$), AdaBoost can be modified appropriately and becomes **Real Adaboost**. The power of AdaBoost to dramatically

Algorithm 1: AdaBoost.M1 algorithm.

$w_i = 1/N, \quad i = 1, \cdots, N$

for $(m = 1 : M)$ **do**

Fit a classifier $G_m(x)$ to training data using w_i;

Compute the training error

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}; \tag{13.2}$$

Compute $\alpha_m = \log \frac{1-err_m}{err_m}$;

$w_i \leftarrow w_i e^{\alpha_m I(y_i \neq G_m(x_i))}, \quad i = 1, \cdots, N$;

return $G(x) = \text{sign}\left(\sum_{i=1}^{M} \alpha_m G_m(x)\right)$

increase the performance of even a very weak classifier.

13.2 Boosting fits an additive model

Boosting is a way of fitting an additive expansion in a set of basis-functions where the basis functions are the individual classifiers. The basis function expansion takes the form:

$$\mathrm{f}(\mathrm{x}) = \sum_{\mathrm{m}=1}^{\mathrm{M}} \beta_{\mathrm{m}} \mathrm{b}(\mathrm{x}, \gamma_{\mathrm{m}}) \tag{13.3}$$

Typically these models are fit by minimizing a loss function averaged over the training data, such as the squared error or a likelihood-based loss function:

$$\min_{\{\beta_{\mathrm{m}}, \gamma_{\mathrm{m}}\}_1^{\mathrm{M}}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{L}\left(\mathrm{y}_{\mathrm{i}}, \sum_{\mathrm{m}=1}^{\mathrm{M}} \beta_{\mathrm{m}} \mathrm{b}(\mathrm{x}_{\mathrm{i}}, \gamma_{\mathrm{m}})\right) \tag{13.4}$$

This is sometimes computationally infeasible and it is approximated by:

$$\min_{\{\beta_{\mathrm{m}}, \gamma_{\mathrm{m}}\}_1^{\mathrm{M}}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{L}(\mathrm{y}_{\mathrm{i}}, \beta \mathrm{b}(\mathrm{x}_{\mathrm{i}}, \gamma)) \tag{13.5}$$

13.3 Forward Stagewise Additive Modeling

Forward stagewise modelling approximates the solution of 13.4 by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added. At each iteration m one solves for the optimal basis function $\mathrm{b}(\mathrm{x}, \gamma_{\mathrm{m}})$ and corresponding coefficient β_{m} to add the current expansion $\mathrm{f}_{\mathrm{m}-1}(\mathrm{x})$. This produces $\mathrm{f}_{\mathrm{m}}(\mathrm{x})$ and the process is repeated. For squared error loss one has:

$$\begin{aligned} \mathrm{L}(\mathrm{y}_{\mathrm{i}}, \mathrm{f}_{\mathrm{m}-1}(\mathrm{x}_{\mathrm{i}}) + \beta \mathrm{b}(\mathrm{x}_{\mathrm{i}}, \gamma)) &= (\mathrm{y}_{\mathrm{i}} - \mathrm{f}_{\mathrm{m}-1}(\mathrm{x}_{\mathrm{i}}) - \mathrm{f}_{\mathrm{m}-1}(\mathrm{x}_{\mathrm{i}}) - \beta \mathrm{b}(\mathrm{x}_{\mathrm{i}}, \gamma))^2 = \\ &= (\mathrm{r}_{\mathrm{i}} - \beta \mathrm{b}(\mathrm{x}_{\mathrm{i}}, \gamma))^2 \end{aligned} \tag{13.6}$$

The term that best fits the residual is added to the expansion at each step.

Stagewise additive modelling is equivalent to AdaBoost.M1 using the exponential loss-function:

$$\mathrm{L}(\mathrm{y}, \mathrm{f}(\mathrm{x})) = \exp(-\mathrm{y}\mathrm{f}(\mathrm{x})) \tag{13.7}$$

The principal attraction of exponential loss in the context of additive modeling is computational; it leads to the simple modular reweighting AdaBoost algorithm.

14 Procedure for datamining

In data mining applications, usually only a small fraction of the large number of predictor variables that have been included in the analysis are actually relevant to prediction. Also, unlike many applications such as pat- tern recognition, there is seldom reliable domain knowledge to help create especially relevant features and/or filter out the irrelevant ones, the inclu- sion of which dramatically degrades the performance of many methods. In addition, data mining applications generally require interpretable mod- els. It is not enough to simply produce predictions. It is also desirable to have information providing qualitative understanding of the relationship between joint values of the input variables and the resulting predicted re- sponse value. Thus, black box methods such as neural networks, which can be quite useful in purely predictive settings such as pattern recognition, are far less useful for data mining.

An "off-the-shelf" method is one that can be directly applied to the data without requiring a great deal of time- consuming data preprocessing or careful tuning of the learning procedure. Decision trees come closest to meeting the requirements for serving as an off-the-shelf procedure for data mining. They are relatively fast to construct and they produce interpretable models (if the trees are small). They naturally incorporate mixtures of numeric and categorical predictor variables and missing values. They are invariant under (strictly monotone) transformations of the individual predictors. As a result, scaling and/or more general transformations are not an issue, and they are immune to the effects of predictor outliers. They perform internal feature selection as an integral part of the procedure. They are thereby resistant, if not completely immune, to the inclusion of many irrelevant predictor variables. These properties of decision trees are largely the reason that they have emerged as the most popular learning method for data mining. Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely inaccuracy. They seldom provide predictive accuracy comparable to the best that can be achieved with the data at hand.

14.1 Boosting trees

The boosted tree model is a sum of tress induced in a forward stagewise manner:

$$\begin{aligned} f_M(x) &= \sum_{m=1}^{M} T(x, \Theta_m) \\ T(x, \Theta_m) &= \sum_{j=1}^{J} \gamma_j I(x \in R_j) \end{aligned} \tag{14.1}$$

where Θ_m is the vector of parameters. Considering a single tree, the parameters are found by minimizing the empirical risk:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) \tag{14.2}$$

Due to the combinatorial problem, we usually settle for suboptimal solutions, dividing the problem into two parts:

- **Finding γ_j given R_j**: this is typically trivial and often $\hat{\gamma}_j = \bar{y}_j$, the mean of y_i falling in region R_j.
- **Finding R_j**: this is the difficult part. A typical strategy is to use a greedy top-down recursive partitioning algorithm to find R_j.

The boosted tree is induced in a forward stagewise manner. At each step one must solve:

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m)) \tag{14.3}$$

for the region set and constants $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ for the next tree, given the current model $f_{m-1}(x)$. Given R_{jm}, finding the optimal constants γ_{jm} in each region is typically straightforward:

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \tag{14.4}$$

Finding the regions is more difficult, and even more difficult than for a single tree.

Defining the loss function as

$$\mathrm{L(f)} = \sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{L(y_i, f(x_i))} \tag{14.5}$$

where f is a sum of trees. So, $\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} \mathrm{L}(\mathbf{f})$, $\mathbf{f} = \{\mathrm{f(x_i}1), \mathrm{f}(x_2), \cdots, \mathrm{f(x_N)}\}$. Numerical procedures exist to find $\hat{\mathbf{f}}$ as a sum of component vectors:

$$\mathbf{f_M} = \sum_{\mathrm{m=0}}^{\mathrm{M}} \mathbf{h_m}, \quad \mathbf{h_m} \in \mathrm{CMcalR^N} \tag{14.6}$$

with $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess.

One optimization is the **Steepest descent**: $\mathbf{h_m} = -\rho_\mathrm{m}\mathbf{g_m}$, $\mathbf{g}$ being the gradient of L(f) evaluated at $\mathbf{f}_{\mathrm{m-1}}$.

Another way is the **Gradient Boosting**. The gradient is trivial to calculate for any differentiable loss function but it is defined only at the training data points $\mathrm{x_i}$ whereas the ultimate goal is to generalize it to new data not in the training set.

A possible resolution to this dilemma is to induce a tree $\mathrm{T(x, \Theta_m)}$ at the m-th iteration whose predictions $\mathrm{t_m}$ are as close as possible to the negative gradient. Using squared error loss to measure closeness:

$$\widetilde{\Theta}_\mathrm{m} = \arg\min_{\Theta} \sum_{\mathrm{i=1}}^{\mathrm{N}} (-\mathrm{g_{im}} - \mathrm{T(x_i, \Theta)})^2 \tag{14.7}$$

The algorithm for classification is similar. Two basic tuning parameters are the number of iterations M and the sizes of each of the constituent trees $\mathrm{J_m}, \mathrm{m} = 1, 2, ..., \mathrm{M}$.

14.2 Right-sized Trees for boosting

[skipped, at least for now]

14.3 Regularization

Besides the size of the constituent trees, J, the other meta-parameter of gradient boosting is the number of boosting iterations M. Each iteration

Algorithm 2: Gradient Tree Boosting algorirthm.

$f_0 \leftarrow \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$;
for $(m = 1 : M)$ **do**
  for $(i = 1 : N)$ **do**
    $r_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f-f_{m-1}}$;
  Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, \quad j = 1, \quad , J_m$;
  for $(i = 1 : J_m)$ **do**
    $\gamma_{jm} = \arg\min_{\gamma} \sum_{x_j \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$;
  $f_m(x) \leftarrow f_{m-1} + \sum_{j=1}^{J_m} L(y_i, f_{m-1}(x_i) + \gamma)$;
return $\hat{f} = f_M(x)$

usually reduces the training risk $L(f_M)$, so that for M large enough this risk can be made arbitrarily small. However, fitting the training data too well can lead to overfitting, which degrades the risk on future predictions. Thus, there is an optimal number $M*$ minimizing future risk that is application dependent. A convenient way to estimate $M*$ is to monitor prediction risk as a function of M on a validation sample. The value of M that minimizes this risk is taken to be an estimate of $M*$.

14.3.1 Shrinkage

Controlling the value of M is not the only possible regularization strategy. Shrinkage techniques can be employed as well. The simplest implementation of shrinkage in this case is to scale the contribution of each tree by a factor $0 < \nu < 1$ when it is added to the current approximation:

$$f_m(X) = f_{m-1}(x) + \nu \sum_{i=1}^{J} \gamma_{jm} I(x \in R_{jm}). \tag{14.8}$$

Smaller values of ν (more shrinkage) result in larger training risk for the same number of iterations M. Empirically it has been found (Friedman, 2001) that smaller values of ν favor better test error, and require correspondingly larger values of M. The best strategy appears to be to set ν to be very small

($\nu < 0.1$) and then choose M by early stopping, however more iterations are needed so the computational cost is higher.

15 Support Vector Machines and Flexible Discriminants

The optimal separating hyperplane separates the two classes and maximizes the distance to the closest point from either class (Vapnik, 1996). Not only does this provide a unique solution to the separating hyperplane problem, but by maximizing the margin between the two classes on the training data, this leads to better classification performance on test data.

The optimal hyperplane (or hard margin SVM) focuses more on points that counts, i.e., close to the border, that are called **support vector**. For this reason it is more robust. LDA depends on all the data, even points faraway. If classes are really Gaussian LDA is optimal and separating hyperplane will pay a price for focusing on noisier data. For logistic regression, if a separate hyperplane exists, since the log-likelihood can be driven to 0.

When data are not separable there will be no feasible solution and an alternative formulation is needed. In this case the we allow some points to be beyond the margin and the problem is generally referred to as *soft margin classification*.

First we analyze the hard margin SVM classification problem with the following steps, then extend it to the non perfectly separable case with the soft margin classification and finally show how it can work for regression as well.

15.1 Hard margin Support Vector Machine classification

https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html https://www.youtube.com/watch?v=_PwhiWxHK8o

We will follow the following steps:

Consider two classes on a space (for example a two dimensional space as in 15.1) and suppose the two classes are perfectly separable. We want to divide the two classes with a straight line. The problem becomes to define what is the best straight line since such lines exist. The solution is to choose the line that separates the classes as much as possible, lying as faraway as possible from both the classes. Such line can be thought as the middle line of a street separating the two classes; then we want to fit the widest possible street between the two classes (see 15.2). The closest occurrences of the two

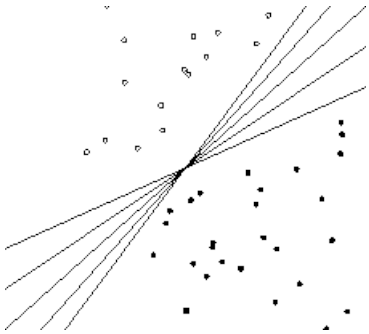


Figure 15.1: Example of infinite lines separating two perfectly separable classes.

classes closest to the separator defines the margins, the gutters of the street (the dashed lines in 15.2).

Let us consider a vector $\vec{w}$ of any length, perpendicular to the gutters of the street and consider an unknown instance to be classified (15.2) called $\vec{u}$. To see if this unknown is on the left or right side of the street, we project its vector $\vec{u}$ onto the vector $\vec{w}$ that is perpendicular to the street. The bigger this product, the more likely the point is to be on the right side of street, hence the more likely it is to be of class +.

Mathematically:

$$\vec{u} \cdot \vec{w} \geq c \Leftrightarrow \vec{u} \cdot \vec{w} + b \geq 0 \Rightarrow \text{Class } 1 \qquad \text{with } b = -c \tag{15.1}$$

This is the **decision rule**. The problem is that both $\vec{w}$ and b are unknown; only the direction of $\vec{w}$ is known but not its length.

Other constraints must be added to calculate $\vec{w}$ and b. Now suppose one positive and one negative instances x_+ and x_- are given. We want:

$$\begin{aligned} \vec{w} \cdot \vec{x}_+ + b &\geq 1 \\ \vec{w} \cdot \vec{x}_- + b &\leq -1 \end{aligned} \tag{15.2}$$

Just for mathematical convenience we introduce a new variable y_i for each instance such that:

$$\begin{aligned} y_i &= 1 \quad \text{if } \vec{x}_+ \\ y_i &= -1 \quad \text{if } \vec{x}_- \end{aligned} \tag{15.3}$$

In this way:

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \tag{15.4}$$

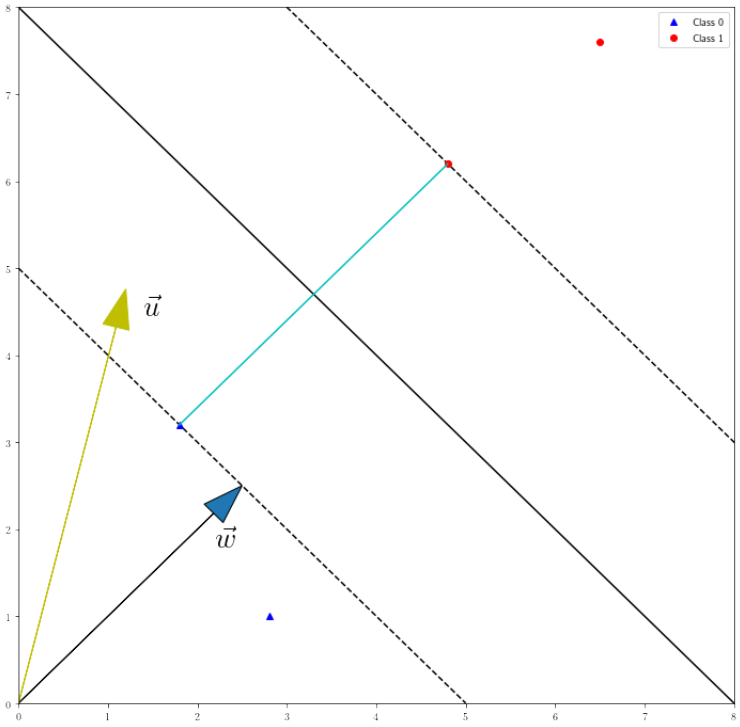


Figure 15.2: Example of best separating line maximizing the margin.

that can be rearranged as:

$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0 \tag{15.5}$$

Now **another constraint** is going to be added: **the previous equation is forced to be** 0 **for the points in the gutter**, i.e., on the margins:

$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 = 0 \tag{15.6}$$

So this equation holds for the points x_+ and x_- that are closest to the separator (the black solid line on the previous plots), i.e., the ones that decide and limit the width of the street. Now we need to express the width of the street. Consider the difference between these two points $(\vec{x}_+ - \vec{x}_-)$ (15.3) and project it onto the normal: **this is exactly the width of the street**.

$$\text{width} = 2M = (\vec{x}_+ - \vec{x}_-) \cdot \frac{\vec{w}}{\|\vec{w}\|} \tag{15.7}$$

For $\vec{x}_+$, $y_i = 1$. From 15.6:

$$\vec{w}\vec{x}_+ + b = 1 \Rightarrow \vec{w}\vec{x}_+ = 1 - b \tag{15.8}$$

for the same reason, considering x_-:

$$-\vec{w}\vec{x}_- - b = 1 \Rightarrow -\vec{w}\vec{x}_+ = 1 + b \tag{15.9}$$

So substituting in 15.7:

$$\text{width} = \frac{2}{\|\vec{w}\|} \tag{15.10}$$

Now we have to maximize the width:

$$\max \frac{2}{\|\vec{w}\|} \Leftrightarrow \max \frac{1}{\|\vec{w}\|} \Leftrightarrow \min \|\vec{w}\| \Leftrightarrow \min \frac{1}{2} \|\vec{w}\|^2 \tag{15.11}$$

We have the expression 15.11 for which we want to find the minimum, its extreme, and we have the constraint 15.6. By **Lagrange**, if we want to find the extreme of a function with constraints, then we have to use **Lagrange multipliers** which will result in a new expression that can be maximized or minimized directly without any constraint.

$$L(\vec{w}, b) = \frac{1}{2} \|\vec{w}\|^2 - \sum_i \alpha_i \left[y_i \left(\vec{w} \cdot \vec{x}_i + b \right) - 1 \right] \tag{15.12}$$

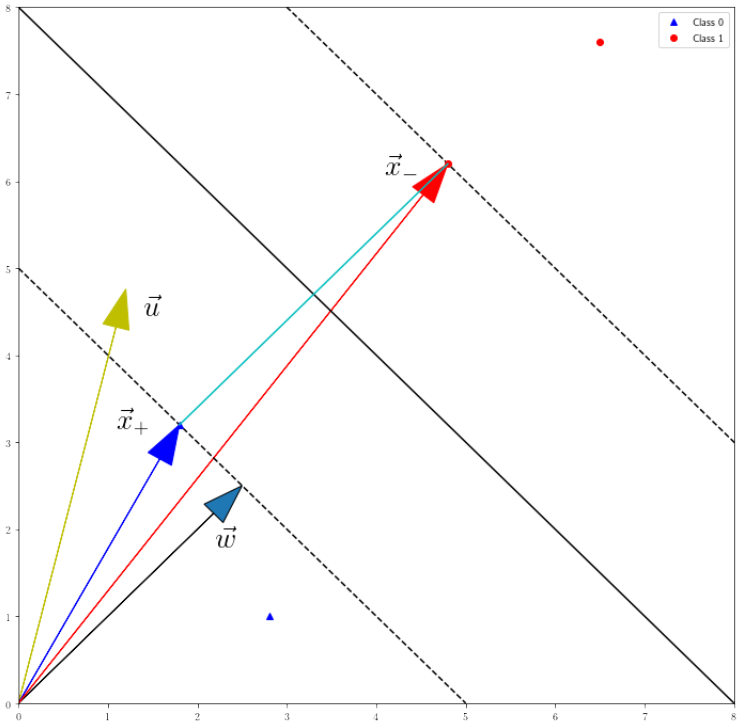


Figure 15.3: Derivation of the width.

where the second term is a sum of all the constraints weighted by α_i, the Lagrange multipliers. These multipliers will be different from 0 for only the points in the gutter; if the same class has more than one point on the margin then all of them will be part of the expression. We need to find the extreme of this expression. To do that, we take the derivatives and set them to 0.

$$\frac{\partial L}{\partial \vec{w}} = w - \sum_i \alpha_i y_i \vec{x}_i = 0 \Rightarrow \quad \vec{w} = \sum_i \alpha_i y_i \vec{x}_i \tag{15.13}$$

This tells that the vector $\vec{w}$ is a linear combination of all the sample vectors lying on the margin, the **support vector**.

Differentiating with respect to b:

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0 \Rightarrow \sum_i \alpha_i y_i = 0 \tag{15.14}$$

We try to substitute these results in 15.12 to find its extreme:

$$\begin{aligned} L &= \frac{1}{2}\left(\sum_i \alpha_i y_i \vec{x}_i\right)\left(\sum_j \alpha_j y_j \vec{x}_j\right) - \sum_i \alpha_i y_i \vec{x}_i \cdot \left(\sum_j \alpha_j y_j \vec{x}_j\right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i = \\ &= \frac{1}{2}\left(\sum_i \alpha_i y_i \vec{x}_i\right)\left(\sum_j \alpha_j y_j \vec{x}_j\right) - \sum_i \alpha_i y_i \vec{x}_i \cdot \left(\sum_j \alpha_j y_j \vec{x}_j\right) - b\cancel{\sum_i \alpha_i y_i} + \sum_i \alpha_i = \\ &= \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \end{aligned} \tag{15.15}$$

where b has been taken out of the sum since it is a constant. This expression allows to see what the value of the extreme point depends on with respect to these vectors. It depends only on the dot product of pairs of samples.

Let us see how the decision rule is affected:

$$\sum_i \alpha_i y_i \vec{x}_i \cdot \vec{u} + b \geq 0 \Rightarrow \vec{u} \text{ belongs to class } + \tag{15.16}$$

Note that the space we are working on is a convex space, so there is no chance to get stuck in a local minimum/maximum, as opposite to neural nets.

This process does not work with linear inseparable problems. In these cases we can switch to another space: we call this transformation $\phi(\vec{x})$. The maximization process depends only on the dot product, so we just need to find $\phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$ in order to maximize in the new space. So actually the mapping function $\phi(\cdot)$ is not needed to be known, we need to know a function such as

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i) \cdot \phi(\vec{x}_j) \tag{15.17}$$

Such a function is called **Kernel function**. Popular choices of kernels are

- linear kernel $(\vec{u} \cdot \vec{v} + 1)^n$
- $e^{-\frac{\|\vec{x}_i - \vec{x}_j\|}{\sigma}}$: be careful to choose a not too small σ the points are shrunk around and we get overfitting.

We have seen **Optimal separating hyperplabes** in 8.11. Here we will see the case for non-separable classes, where the classes overlap.

15.2 Soft Margin Support Vector Machine

The hard margin SVM works well when data do not contain outliers or noisy samples close to the separator. In such cases, the outlier can make the margin too small or move the separator toward the other class and this might cause misclassification. It is generally preferable to have a fat margin allowing to some noisy samples to be inside or on the wrong side of the margin.

We still want to maximize the margins M allowing some points to be on the wrong side. For each misclassified point we pay a cost proportional to its distance from the margin. To do this, we define a slack variable $\xi = (\xi_1, \xi_2, \cdots, \xi_N)$ with $\xi_i \geq 0$, whose value is not null for the points beyond the margin:

$$\xi_i = \max(0, 1 - y_i(\vec{w}\vec{x_i} + b)) \tag{15.18}$$

where $1 - y_i(\vec{w}\vec{x_i} + b)$ is proportional to the distance from the margin. The decision rule hence becomes:

$$\begin{aligned} &y_i(\vec{w}\vec{x}_i + b) \geq 1 - \xi_i \Rightarrow y_i(\vec{w}\vec{x}_i + b) - 1 + \xi_i \geq 0 \\ &\xi_i \geq 0 \end{aligned} \tag{15.19}$$

However, we want to limit the number of misclassified points so the objective becomes

$$\begin{aligned} &\min \frac{1}{2}\|\vec{w}\| + C\sum_i \xi_i \\ &\textit{subject to } \begin{cases} y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \end{aligned} \tag{15.20}$$

The width remains the same as the one in 15.1.

The problem is quadratic with linear inequalities, hence it is still convex. The Lagrange primal function becomes:

$$L_P = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left[y_i(\vec{w}\cdot\vec{x}_i + b) - (1-\xi_i)\right] - \sum_{i=1}^{N}\mu_i\xi_i \tag{15.21}$$

where μ_i are the Lagrange multipliers for the newly introduced constraint $\xi_i \geq 0$. We minimize w.r.t. $\vec{w}, b, \xi_i$. The first two have already been calculated in 15.13 and 15.14:

$$\frac{\partial L_P}{\partial \xi_i} \Longrightarrow \alpha_i = C - \mu_i \quad \forall i \tag{15.22}$$

Substituting in 15.21, we obtain the Lagrangian (Wolfe) dual objective function:

$$\begin{aligned} L_D &= \frac{1}{2}\|\vec{w}\|^2 + C\cancel{\sum_{i=1}^{N}\xi_i} - \sum_{i=1}^{N}\alpha_i\left[y_i(\vec{w}\cdot\vec{x}_i + b) - 1\right] + \cancel{\alpha_i(\xi_i)} - \cancel{\sum_{i=1}^{N}\mu_i\xi_i} = \\ &= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i\vec{x}_i \cdot \sum_{j=1}^{N}\alpha_j y_j\vec{x}_j - \sum_{i=1}^{N}\alpha_i y_i\vec{x}_i\vec{w} - b\cancel{\sum_{i=1}^{N}\alpha_i y_i} + \sum_{i=1}^{N}\alpha_i = \\ &= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i\vec{x}_i \cdot \sum_{j=1}^{N}\alpha_j y_j\vec{x}_j - \sum_{i=1}^{N}\alpha_i y_i\vec{x}_i\sum_{j=1}^{N}\alpha_j y_j\vec{x}_j + \sum_{i=1}^{N}\alpha_i = \\ &= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i\vec{x}_i\sum_{j=1}^{N}\alpha_j y_j\vec{x}_j = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i y_i\alpha_j y_j\vec{x}_i\vec{x}_j \end{aligned} \tag{15.23}$$

where we used the properties that α_i, y_i are scalar and that $\vec{w}, b$ can be taken out. Notice that $\vec{w}\cdot\vec{w} = \sum_{i=1}^{N}\alpha_i y_i\vec{x}_i \cdot \sum_{j=1}^{N}\alpha_j y_j\vec{x}_j = \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i y_i\alpha_j y_j\vec{x}_i\vec{x}_j \neq$

$\sum_{i=1}^{N}(\alpha_i y_i)^2 \vec{x}_i \vec{x}_i$. In fact consider for example: $\vec{w} = c_1\vec{x}_1 + c_2\vec{x}_2$ with $\vec{x}_i = [x_{1i}, x_{2i}]^T$. $\vec{w} = [c_1x_{11} + c_2x_{12}, c_1x_{21} + c_2x_{22}]^T$ and $\vec{w} \cdot \vec{w} = (c_1x_{11} + c_2x_{12})^2 + (c_1x_{21} + c_2x_{22})^2$, so we are missing some cross-products. The Lagrangian (Wolfe) dual objective function gives a lower bound on the objective function for any feasible point. We maximize L_D subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i$. In addition to the derivatives, the **Karush-kuhn-Tucker** conditions include constraints:

$$\begin{aligned} \alpha_i \left[y_i(\vec{w} \cdot \vec{x}_i + b) - (1 - \xi_i) \right] &= 0 \\ \mu_i \xi_i &= 0 \\ y_i(\vec{x}_i \vec{w} + b) - (1 - \xi_i) &\geq 0 \end{aligned} \tag{15.24}$$

for $i = 1, \cdots, N$. These constraints and the ones originating from the derivatives uniquely characterize the primal and dual problem.

From $\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i$, we can see:

$$\hat{\vec{w}} = \sum_{i=1}^{N} \hat{\alpha}_i y_i \vec{x} \tag{15.25}$$

with $\hat{\alpha}_i \neq 0$ only for those observations i for which the constraints in the last equation of 15.24 are exactly met. These observations are called **support vectors**, since $\hat{\vec{w}}$ is represented in terms of them alone. Among these support points, some will lie on the edge of the margin ($\hat{\xi}_i = 0$) and hence will be characterized by $0 < \hat{\alpha}_i < C$: the remainder ($\hat{\xi}_i > 0$) have $\hat{\alpha}_i = C$. We can see that any of these margin points ($0 < \hat{\alpha}_i$, $\hat{\xi}_i = 0$), can be used to solve for b, and we typically use an average of all the solutions for numerical stability. Maximizing the dual is a simpler convex quadratic programming problem than the primal and can be solved with standard techniques such (Murray et. al, 1981).

Points on the wrong side of the boundary are support vectors. In addition, points on the correct side of the boundary but close to it (in the margin), are also support vectors.

Given the solutions $\hat{\vec{w}}, \hat{b}$, the decision function can be written as:

$$\hat{G}(x) = \text{sign}\Big(\hat{f}(x)\Big) = \text{sign}\Big(\hat{\vec{w}} \cdot \vec{x}_i + b\Big) \tag{15.26}$$

The tuning parameter of this procedure is the cost parameter C. The optimal value for C can be estimated by cross-validation.

15.3 Support vector machines and Kernels

We can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines. Generally linear boundaries in the enlarged space achieve better training-class separation, and translate to nonlinear boundaries in the original space.

15.3.1 Computing SVM for classification

The Lagrange dual function has the form:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i \alpha_j y_j \left\langle h(x_i), h(x_j) \right\rangle \tag{15.27}$$

So f(x) can be written as

$$f(x) = h(x) \cdot \vec{w} + b = \sum_{i=1}^{N} \alpha_i y_i \left\langle h(x), h(x_i) \right\rangle + b \tag{15.28}$$

As before, given α_i, b can be determined by solving $y_i f(x_i) = 1$ for any (or all) x_i for which $0 < \alpha_i < C$. So h(x0) is involved only in terms of its inner product: **we do not need to specify the transformation but only the knowledge of the kernel function is required**:

$$K(x, x') = \left\langle h(x), h(x') \right\rangle \tag{15.29}$$

K should be symmetric positive (semi-) definite. Possible popular choices are:

$$\begin{aligned} \textbf{dth-Degree polynomial:} \quad & K(x, x') = \left(1 + \left\langle x, x' \right\rangle\right)^d \\ \textbf{Radial basis:} \quad & K(x, x') = e^{-\gamma |x - x'|^2} \\ \textbf{Neural Network:} \quad & K(x, x') = \tanh\left(k_1 \left\langle x, x' \right\rangle + k_2\right) \end{aligned} \tag{15.30}$$

15.4 SVM for Regression

SVM can be adapted for regression. The linear regression model is in the form

$$f(x) = x^T \beta + \beta_0 \tag{15.31}$$

and then handle non-linear generalizations. To estimate β we consider minimization of:

$$\begin{aligned} H(\beta, \beta_0) &= \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}|\beta|^2 \\ V_\epsilon &= \begin{cases} 0 & \textit{if} \quad |r| < 1 \\ |r| - \epsilon & \textit{otherwise} \end{cases} \end{aligned} \tag{15.32}$$

This is an "ϵ-insensitive" error measure, ignoring errors of size less than ϵ (left panel of 15.4). There is a rough analogy with the support vector classification setup, where points on the correct side of the decision boundary and far away from it, are ignored in the optimization. In regression, these "low error" points are the ones with small residuals.

The support vector error measure has linear tails (beyond ϵ), but in addition it flattens the contributions of those cases with small residuals.

If β, β_0 are the minimizers of H, the solution function can be shown to have the form:

$$\begin{aligned} \hat{\beta} &= \sum_{i=1}^{N} (\hat{\alpha}_i^* - \hat{\alpha}_i) x_i \\ \hat{f}(x) &= \sum_{i=1}^{N} (\hat{\alpha}_i^* - \hat{\alpha}_i) \langle x, x_i \rangle + \beta_0 \end{aligned} \tag{15.33}$$

where $\hat{\alpha}_i^*, \hat{\alpha}_i$ are positive and solve the quadratic problem:

$$\begin{aligned} &\min_{\alpha_i, \alpha_i^*} \epsilon \sum_{i=1}^{N} (\alpha_i^* + \alpha_i) - \sum_{i=1}^{N} y_i (\alpha_i^* - \alpha_i) + \frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) \langle x_i, x_j \rangle \\ &\textit{subject to} \begin{cases} 0 \le \alpha_i, \alpha_i^* \le \frac{1}{\lambda} \\ \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) = 0 \\ \alpha_i, \alpha_i^* = 0 \end{cases} \end{aligned} \tag{15.34}$$

Due to the nature of these constraints, typically only a subset of the solution values $(\hat{\alpha}_i^* - \hat{\alpha}_i^*)$ are nonzero, and the associated data values are called the support vectors. There are two parameters: ϵ is a parameter of the loss function. λ is a regularization parameter and can be estimated by cross validation.

The SVM can be extended to multiclass problems by solving many two-class problems. A classifier is built for each pair of classes and the final

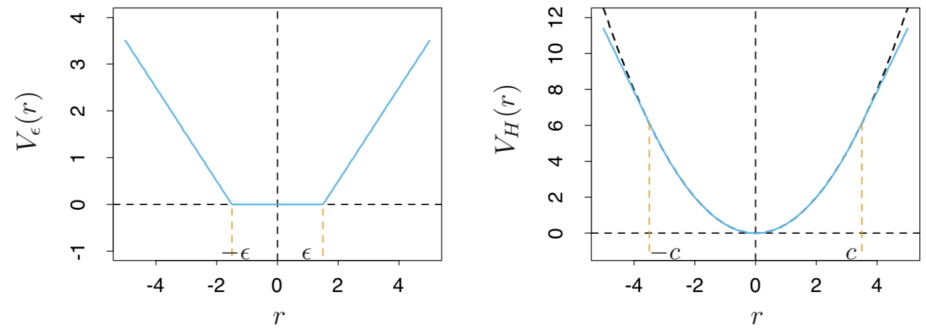


Figure 15.4: The left panel shows the ϵ-insensitive error function used by the support vector regression machine. The right panel shows the error function used in Huber's robust regression (blue curve) used in statistics. Beyond |c|, the function changes from quadratic to linear reducing the contribution of outliers.

classifier is the one that dominates the most.

15.5 Generalized LDA

LDA is a simple prototype classifier which classifies points closest to the centroid. LDA is the estimated Bayes classifier if the observations are multivariate Gaussian in each class, with common covariance matrix, which is very unlikely. LDA has linear boundaries and provides natural low-dimensional views of data. Despite these properties, it fails in a number of situations. One is when more flexibility at for the boundaries is required; in this case QDA works. It does not work well neither with too many correlated predictors, for example in case of digitalized analogue signals and images, since in this case it uses too many parameters. Finally, LDA assumes a single centroid (prototype) per class (and a common covariance matrix) to describe the spread of the data in each class. Sometimes several centroids are more appropriate.

Three alternatives exist to these three situations:

- **recast the LDA problem as a linear regression problem**, which in turn can be converted to a non-parametric form of regression, achieving more flexibility. These are called **FDA**. The regression procedures

can be seen to identify an enlarged set of predictors via basis expansions. FDA amounts to LDA in this enlarged space.

- When there are too many predictors, we do not want to expand the set. The second idea is to fit LDA but penalize its coefficients to be smooth or otherwise coherent in the spatial domain, such as an image. This is called **Penalized Discriminant Analysis (PDA)**.
- **Model each class by a mixture of two or more Gaussians with different centroids sharing the same covariance matrix**. This allows for more complex decision boundaries and allows for subspace reduction. This is called **Mixture Discriminant Analysis (MDA)**.

15.5.1 Flexible Discriminant Analysis (FDA)

This technique performs LDA using linear-regression on derived responses. Assume we have observations with a quantitative response G falling into one of K classes. Suppose $\theta : \mathcal{G} \to \mathcal{R}^1$ is a function that assigns scores to the classes such that the transformed class labels are optimally predicted by linear regression, producing the derived responses. Then, having as training examples $(\mathrm{g_i}, \mathrm{x_i})$, we solve:

$$\min_{\theta,\beta} \sum_{\mathrm{i}=1}^{\mathrm{N}} \Big(\theta(\mathrm{g_i}) - \mathrm{x_i^T}\beta\Big)^2 \tag{15.35}$$

with restrictions on θ to avoid trivial solutions(mean zero and unit variance over the training data).

More generally, we can find up to $\mathrm{L} \leq \mathrm{K}-1$ sets of independent scorings for the class labels, $\theta_1, \theta_2, \cdots, \theta_{\mathrm{L}}$, and L corresponding linear maps $\eta_{\mathrm{l}}(\mathrm{X}) = \mathrm{X^T}\beta_{\mathrm{l}}, \mathrm{l} = 1, \cdots, \mathrm{L}$, chosen to be optimal for multiple regression in $\mathcal{R}^{\mathrm{p}}$. So the mapping function for one class does not to be the same for the others.

The scores $\theta_{\mathrm{l}}(\mathrm{g})$ and maps β_{l} are chosen to minimize the average squared residual:

$$\mathrm{ASR} = \frac{1}{\mathrm{N}} \sum_{\mathrm{l}=1}^{\mathrm{N}} \left[\Big(\theta_{\mathrm{l}}(\mathrm{g_i}) - \mathrm{x_i^T}\beta_{\mathrm{k}}\Big)^2 \right] \tag{15.36}$$

The set of scores are assumed to be mutually orthogonal and normalized with respect to an appropriate inner product to prevent trivial zero solutions.

The Mahalanobis distance of a test point x to the k – th class centroid $\hat{\mu}_{\mathrm{k}}$ is given by:

$$\delta_{\mathrm{j}}(\mathrm{x}, \hat{\mu}_{\mathrm{k}}) = \sum_{\ell=1}^{\mathrm{K–1}} \mathrm{w}_{\ell}\Big(\hat{\eta}_{\ell}(\mathrm{x}) - \eta_{\ell}^{-\mathrm{k}}(\mathrm{x})\Big) + \mathrm{D}(\mathrm{x}) \tag{15.37}$$

where $\eta_{\ell}^{-\mathrm{k}}(\mathrm{x})$s is the mean of $\hat{\eta}_{\ell}(\mathrm{x})$ in the k – th class, and D(x) does not depend on k and w_{l} are defined in terms of the mean squared residual r_{ℓ}^2 f the ℓth optimally scored fit:

$$\mathrm{w}_{\ell} = \frac{1}{\mathrm{r}_{\ell}^2(1 - \mathrm{r}_{\ell}^2} \tag{15.38}$$

To summarize, LDA can be performed by a sequence of linear regressions, fol- lowed by classification to the closest class centroid in the space of fits.

The power is that we can replace the linear regression with more flexible non-parametric fits, achieving a more flexible classifier. We can use generalized additive fits, spline functions, MARS and others. In this more general form the regression problems are defined by the criterion:

$$\mathrm{ASR}\Big(\{\theta_{\ell}, \eta_{\ell}\}_{\ell=1}^{\mathrm{L}}\Big) = \frac{1}{\mathrm{N}} \sum_{\ell=1}^{\mathrm{L}} \left[\sum_{\mathrm{i}=1}^{\mathrm{N}} (\theta_{\ell}(\mathrm{g_i}) - \eta_{\ell}(\mathrm{x_i}))^2 + \lambda \mathrm{J}(\eta_{\ell}) \right] \tag{15.39}$$

where J is a regularizer appropriate for some forms of nonparametric regression.

The computations of FDA coordinates can be simplified in cases when the nonparametric procedure can be represented as a linear operator $\mathbf{S}_{\lambda}$, that is $\hat{\mathbf{y}} = \mathbf{S}_{\lambda}\mathbf{y}$. Additive splines have this property if the smoothing parameters are fixed, as well as MARS when the basis functions are selected. The subscript λ denotes the entire set of smoothing parameters. If $\mathbf{S}_{\lambda} = \mathbf{H}_{\mathrm{X}}$, the linear regression projection operator, then we are back to LDA.

We create $\mathrm{N} \times \mathrm{K}$ *indicator response matrix* $\mathbf{Y}$ from the response $\mathrm{g_i}$, such that $\mathrm{y_{ik}} = 1$ if $\mathrm{g_i} = \mathrm{k}$, 0 otherwise. Each row in $\mathbb{Y}$ has only one 1. The steps are the following:

1. **Multivariate nonparametric regression**: fit a multiresponse, adaptive nonparametric regression of $\mathbf{Y}$ on $\mathbf{X}$, giving the fitted values $\hat{\mathbf{Y}}$. Let $\mathbf{S}_{\lambda}$ be the linear operator that fits the final chosen model and $\eta^*(\mathrm{x})$ be the vector of fitted regression functions .

2. **Optimal scores**: compute the eigen-decomposition of $\mathbf{Y}^{\mathrm{T}}\hat{\mathbf{Y}} = \mathbf{Y}^{\mathrm{T}}\mathbf{S}_{\lambda}\mathbf{Y}$ where the eigenvectors $\mathbf{\Phi}$ are normalized: $\mathbf{\Phi}^{\mathrm{T}}\mathbf{D}_{\pi}\mathbf{\Phi}^{\mathrm{T}}$. here $\mathbf{D}_{\pi} = \mathbf{Y}^{\mathrm{T}}\mathbf{Y}/\mathrm{N}$ is a diagonal matrix of the estimated class prior probabilities.

3. **Update** the model from step 1 using the optimal scores $\eta(\mathrm{x}) = \Phi^{\mathrm{T}}\eta^{*}(\mathrm{x})$.

15.5.2 Penalized discriminant analysis

Although FDA is motivated by generalizing optimal scoring, it can also be viewed directly as a form of regularized discriminant analysis. Suppose the regression procedure used in FDA amounts to a linear regression onto a basis expansion h(X), with a quadratic penalty on the coefficients:

$$\mathrm{ASR}\Big(\{\theta_{\ell},\beta_{\ell}\}_{\ell=1}^{\mathrm{L}}\Big) = \frac{1}{\mathrm{N}}\sum_{\ell=1}^{\mathrm{L}}\left[\sum_{\mathrm{i}=1}^{\mathrm{N}}\Big(\theta_{\ell}(\mathrm{g_i}) - \mathrm{h}^{\mathrm{T}}(\mathrm{x_i})\beta_{\ell}\Big)^2 + \lambda\beta_{\ell}^{\mathrm{T}}\Omega\beta_{\ell}\right] \quad (15.40)$$

The choice of Ω depends on the problem. if $\eta_{\ell}(\mathrm{x}) = \mathrm{h}(\mathrm{x})\beta_{\ell}$ is an expansion on spline basis functions, Ω might constrain η_{ℓ} to be smooth over $\mathcal{R}^{\mathrm{p}}$. In the case of additive splines, there are N spline basis functions for each coordinate, resulting in a total of $\mathrm{N_p}$ basis functions in h(x); Ω in this case is $\mathrm{N_p} \times \mathrm{N_p}$ and block diagonal.

The steps in FDA can then be viewed as a generalized form of LDA, which we call penalized discriminant analysis, or PDA:

- Enlarge the set of predictors X via a basis expansion h(X).
- Use (penalized) LDA in the enlarged space where the Mahalanobis distance is given by:

$$\mathrm{D}(\mathrm{x},\mu) = (\mathrm{h}(\mathrm{x}) - \mathrm{h}(\mu))^{\mathrm{T}}(\Sigma_{\mathrm{W}} + \lambda\Sigma)^{-1}(\mathrm{h}(\mathrm{x}) - \mathrm{h}(\mu)) \quad (15.41)$$

  where Σ_{W} is the within-class covariance matrix of the derived varaibles $\mathrm{h}(\mathrm{x_i})$.

- Decompose the classification subspace using a penalized metric:

$$\max \mathrm{u}^{\mathrm{t}}\Sigma_{\mathrm{Bet}}\mathrm{u} \qquad \textit{subject to} \quad \mathrm{u}^{\mathrm{T}}(\Sigma_{\mathrm{W}} + \lambda\Sigma)\mathrm{u} = 1 \quad (15.42)$$

For some classes of problems, the first step, involving the basis expansion, is not needed; we already have far too many (correlated) predictors. An example is when the objects are digitalized analogue signals. Neighboring pixel values will tend to be correlated, being often almost the same. This implies that the pair of corresponding LDA coefficients for these pixels can be wildly different and opposite in sign, and thus cancel when applied to similar pixel values. Positively correlated predictors lead to noisy, negatively correlated coeffi- cient estimates, and this noise results in unwanted sampling variance. A reasonable strategy is to regularize the coefficients to be smooth over the spatial domain, as with images. This is what PDA does. The computations proceed just as for FDA, except that an appropriate penalized regression method is used.

15.5.3 Mixture Discriminant Analysis

Linear discriminant analysis can be viewed as a prototype classifier. Each class is represented by its centroid, and we classify to the closest using an appropriate metric. In many situations a single prototype is not sufficient to represent inhomogeneous classes and mixture models are more appropriate.

A Gaussian mixture model for the k – th class has density:

$$\mathrm{P}(\mathrm{X}|\mathrm{G}=\mathrm{k}) = \sum_{\mathrm{r}=1}^{\mathrm{R_k}} \pi_{\mathrm{kr}} \phi(\mathrm{X}, \mu_{\mathrm{kr}}, \Sigma) \tag{15.43}$$

where the **mixing proportions**π_{kr} sum to one. The class posterior probabilities are given by

$$\mathrm{P}(\mathrm{G}=\mathrm{k}|\mathrm{X}=\mathrm{x}) = \frac{\sum_{\mathrm{r}=1}^{\mathrm{R_k}} \pi_{\mathrm{kr}} \phi(\mathrm{X}, \mu_{\mathrm{kr}}, \Sigma)\Pi_{\mathrm{k}}}{\sum_{\ell=1}^{\mathrm{K}} \sum_{\mathrm{r}=1}^{\mathrm{R}_\ell} \pi_{\ell\mathrm{r}} \phi(\mathrm{X}, \mu_{\ell\mathrm{r}}, \Sigma)\Pi_\ell} \tag{15.44}$$

where $\mathrm{Pi_k}$ are the class priors.

As in LDA we estimate the parameters by MLE, using the joint log-likelihood based on $\mathrm{P}(\mathrm{G}, \mathrm{X})$:

$$\sum_{\mathrm{k}=1}^{\mathrm{K}} \sum_{\mathrm{g_i}=\mathrm{k}} \log \left[\sum_{\mathrm{r}=1}^{\mathrm{R_k}} \pi_{\mathrm{kr}} \phi(\mathrm{x_i}, \mu_{\mathrm{kr}}, \Sigma)\Pi_{\mathrm{k}} \right] \tag{15.45}$$

The sum within the log makes this a rather messy optimization problem if tackled directly. The classical and natural method for computing the

maximum-likelihood estimates (MLEs) for mixture distributions is the EM algorithm. EM alternates between the two steps:

- **E-step** Given the current parameters, compute the **responsability** of subclass c_{kr} within class k for each of the class-k observations ($g_i = k$):

$$W(c_{kr}|x_i, g_i) = \frac{\pi_{kr}\phi(x_i, \mu_{kr}, \Sigma)}{\sum_{\ell=1}^{R_k} \pi_{k\ell}\phi(x_i, \mu_{k\ell}, \Sigma)} \tag{15.46}$$

- **M-step**: compute the weighted MLEs for the parameters of each of the component Gaussians within each of the classes, using the weights from the E-step.

In the E-step, the algorithm apportions the unit weight of an observation in class k to the various subclasses assigned to that class. If it is close to the centroid of a particular subclass, and far from the others, it will receive a mass close to one for that subclass. On the other hand, observations halfway between two subclasses will get approximately equal weight for both.

In the M-step, an observation in class k is used R_k times, to estimate the parameters in each of the R_k component densities, with a different weight for each.

16 Prototype methods and Nearest-Neighbours

There exist model-free methods for classification and pattern recognition. Because they are highly unstructured, they typically are not useful for understanding the nature of the relationship between the features and class outcome but they work more in a black-box manner.

Prototype methods represent points in a feature space. Each prototype has an associated class label and classification of a point is made to the class of the closest prototype, generally in terms of Euclidean distance in a feature space, after each feature has been standardized (0-mean, unit variance).

16.1 K-means clustering

It is a method to find clusters and cluster centres in a set of unlabeled data. Once chooses the desired number of cluster centres, R, and the K-means

procedure iteratively moves the centres to minimize the total within cluster variance (K is the number of classes, R of clusters).

Given an initial set of centres, the algorithm alternates two steps:

- for each centre we identify the subset of training points (its cluster) that is closer to it than any other center;
- the means of each feature for the data points in each cluster are computed and this mean vector becomes the new centre for that cluster.

These steps are iterated until convergence. Typically the initial R centres are randomly chosen from the training set.

To use K-means clustering for classification of labeled data, the steps are:

- apply K-means clustering to the training data in each class separately, using R prototypes;
- assign a class label to each of the K $\times$ R prototypes;
- classify a new feature x to the class of the closest prototype.

Prototypes near the class boundaries might lead to misclassification errors for points near these boundaries. This results from the fact that for each class, the other classes do not have a say in the positioning of the prototypes for that class.

16.2 Learning Vector Quantization

LVQ places prototypes in strategical position w.r.t. decision boundaries. It is an online algorithm: observations are processed one at a time.

The idea is that the training points attract prototypes of the correct class, and repel other prototypes. When the iterations settle down, prototypes should be close to the training points in their class. The learning rate oϵ is decreased to zero with each iteration. The prototypes tend to move away from the decision boundaries, and away from prototypes of competing classes. A drawback is that they are defined by algorithms, rather than optimization

of some fixed criteria; this makes it difficult to understand their properties.

Algorithm 3: Learning Vector Quantization

Chose R initial prototypes for each class: $m_1(k), \cdots, m_R(k)$, $k = 1, \cdots, K$, for example by sampling R training points at random from each class;

Sample a training point (x_i, g_i) randomly and let (j, k) index the closest prototype $m_j(k)$ to x_i;

if $g_i = k$ **then**

  $m_j(k) \leftarrow m_j(k) + \epsilon(x_i - m_j(k))$: move the prototype towards the the training point;

else

  $m_j(k) \leftarrow m_j(k) - \epsilon(x_i - m_j(k))$: move the prototype away from the training point;

end

Repeat step 2, decreasing the learning rate ϵ with each iteration towards 0.

16.3 Gaussian Mixtures

Each cluster is described in terms of a Gaussian density with mean and covariance matrix. The two steps of the alternating EM algorithm are very similar to the two steps in K-means:

- In the E-step, each observation is assigned a responsibility or weight for each cluster, based on the likelihood of each of the corresponding Gaussians. Observations close to the center of a cluster will most likely get weight 1 for that cluster, and weight 0 for every other cluster. Observations half-way between two clusters divide their weight accordingly.
- In the M-step, each observation contributes to the weighted means (and covariances) for every cluster.

The Gaussian mixture model is often referred to as a soft clustering method, while K-means is hard. When Gaussian mixture models are used to represent the feature density in each class, it produces smooth posterior probabilities while the classification rule is $\hat{G}(x) = \arg\max_k \hat{p}_k(x)$.

Although the decision boundaries are roughly similar, those for the mixture model are smoother (although the prototypes are in approximately the same positions).

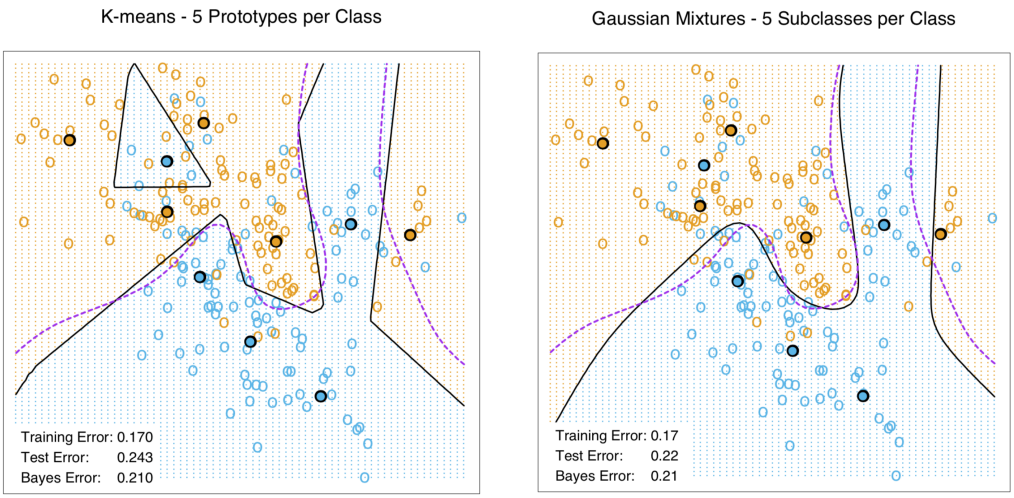


Figure 16.1: Comparison between k-means and Gaussian Mixture model with the same prototypes.

16.4 K-nearest neighbour classifier

These classifiers are memory-based, and require no model to be fit. Given a query point x_0, we find the k training points $x(r), r = 1, ..., k$ closest in distance to x_0, and then classify using majority vote among the k-neighbors. Ties are broken at random.Typically we first standardize each of the features to have mean zero and variance 1, since it is possible that they are measured in different units. Because it uses only the training point closest to the query point, the bias of the 1-nearest-neighbor estimate is often low, but the variance is high.

16.5 Adaptive Nearest Neighbour

Implicit in near-neighbor classification is the assumption that the class probabilities are roughly constant in the neighborhood, and hence simple averages give good estimates. However, in the example of 16.2 the class probabilities vary only in the horizontal direction. If we knew this, we would stretch the neighborhood in the vertical direction, as shown by the tall rectangular region. This will reduce the bias of our estimate and leave the variance the same.

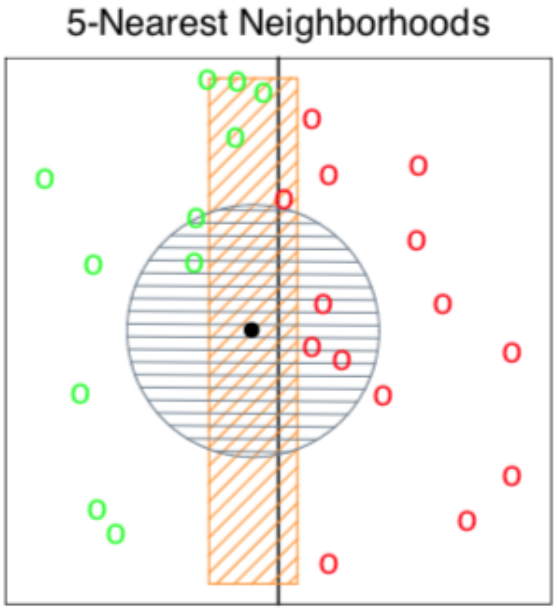


Figure 16.2: The points are uniform in the cube, with the vertical line separating class red and green. The vertical strip denotes the 5-nearest-neighbor region using only the horizontal coordinate to find the nearest-neighbors for the target point (solid dot). The sphere shows the 5-nearest-neighbor region using both co- ordinates, and we see in this case it has extended into the class-red region (and is dominated by the wrong class in this instance).

This suggests the idea of adapting metrics, so that the resulting neighbourhood stretch out in directions for which the class probabilities do not change much. One of these methods is the **Discriminant Adaptive Nearest-Neighbourhood (DANN)**. At each query point a neighborhood of say 50 points is formed, and the class distribution among the points is used to decide how to deform the neighborhood—that is, to adapt the metric. The adapted metric is then used in a nearest-neighbor rule at the query point. Thus at each query point a potentially different metric is used.

The DANN metric at a query x_0 is defined by:

$$\begin{aligned} D(x, x_0) &= (x - x_0)^T \Sigma (x - x_0) \\ \Sigma &= \mathbf{W}^{-\frac{1}{2}} \left[\mathbf{W}^{-\frac{1}{2}} \mathbf{B} \mathbf{W}^{-\frac{1}{2}} + \epsilon \mathbf{I} \right] \mathbf{W}^{-\frac{1}{2}} = \\ &= \mathbf{W}^{-\frac{1}{2}} \left[\mathbf{B}^* + \epsilon \mathbf{I} \right] \mathbf{W}^{-\frac{1}{2}} \\ \mathbf{W} &= \sum_{k=1}^{K} \pi_k \mathbf{W}_k \\ \mathbf{B} &= \sum_{k=1}^{K} \pi_k (\bar{x}_k - \bar{x})(\bar{x}_k - \bar{x})^T \end{aligned} \tag{16.1}$$

$\mathbf{W}$ is the pooled within-class covariance matrix and $\mathbf{B}$ is the between class covariance matrix, both computed with the K nearest neighbours around x_0. The formula first spheres the data with respect to $\mathbf{W}$ and then stretches the neighbourhood in the zero-eigenvalue directions of $\mathbf{B}^*$. This makes sense, since locally the observed class means do not differ in these directions. The ϵ parameter rounds the neighborhood, from an infinite strip to an ellipsoid, to avoid using points far away from the query point. The value of $\epsilon = 1$ seems to work well in general.

16.3 shows the resulting neighborhoods for a problem where the classes form two concentric circles. Notice how the neighborhoods stretch out orthogonally to the decision boundaries when both classes are present in the neighborhood. In the pure regions with only one class, the neighborhoods remain circular: in these cases the between matrix $\mathbf{B} = 0$, and the Σ is the identity matrix.

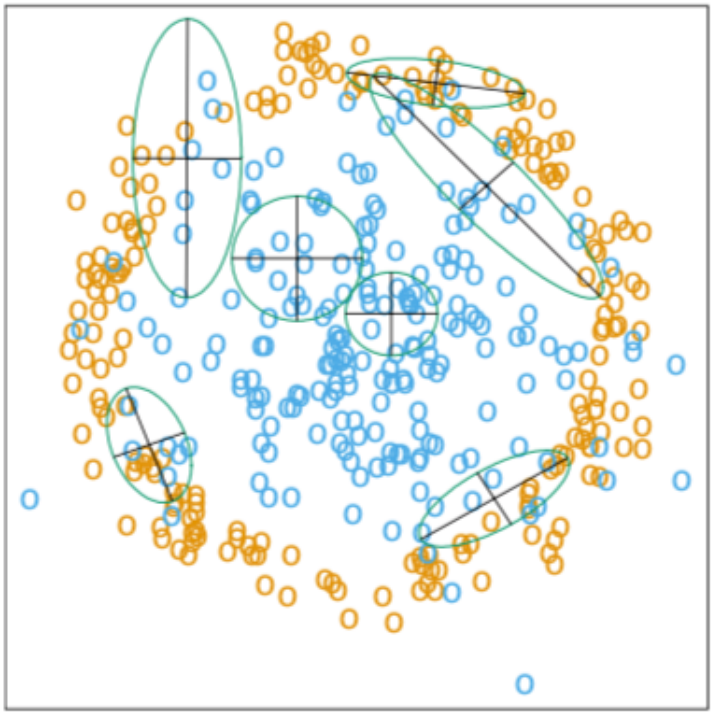


Figure 16.3: Neighborhoods found by the DANN procedure, at various query points (centers of the crosses). There are two classes in the data, with one class surrounding the other. 50 nearest-neighbors were used to estimate the local metrics. Shown are the resulting metrics used to form 15-nearest-neighborhoods.

16.5.1 Global Dimension Reduction for Nearest Neighbours

The discriminant-adaptive nearest-neighbor method carries out local dimension reduction, that is, dimension reduction separately at each query point. In many problems we can also benefit from global dimension reduction, that is, apply a nearest-neighbor rule in some optimally chosen subspace of the original feature space.

At each training point x_i, the between-centroids sum of squares matrix $\mathbf{B}_i$ is computed, and then these matrices are averaged over all training points:

$$\bar{\mathbf{B}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{B}_i \tag{16.2}$$

Let $e_1, \cdots, e_p$ be the eigenvectors of the matrix $\mathbf{B}$ decreasingly ordered. These eigenvectors span the optimal subspaces for global subspace reduction. The rank-L approximation to $\bar{\mathbf{B}}$ is:

$$\bar{\mathbf{B}}_L = \sum_{\ell=1}^{L}\theta_\ell e_\ell e_\ell^T \tag{16.3}$$

and it solves the least squares problem

$$\min_{\text{rank}(\mathbf{M})=L}\sum_{i=1}^{N}\text{trace}\left[(\mathbf{B}_i - \mathbf{M})^2\right] \tag{16.4}$$

$\mathbf{B}_i$ contains information on the local discriminant subspace and the strength of discriminant in that subsapce, it can be seen as a way of finding the best approximating subspace of dimension L to a series of N subspaces by weighted least squares.

For example, when some eigenvalues turn out to be large (having eigenvectors nearly spanning the interesting subspace), and the remaining six are near zero, we project the data into the leading four-dimensional subspace, and then carry out nearest neighbour classification.

16.5.2 Computational considerations

One drawback of nearest-neighbour is the computational load, both in finding the neighbours and storing the entire training set.

Reducing the storage requirements is difficult, and various editing and condensing procedures have been proposed. The idea is to isolate a subset of the training set that suffices for nearest-neighbour predictions, and throw away the remaining training data. Intuitively, it seems important to keep the training points that are near the decision boundaries and on the correct side of those boundaries, while some points far from the boundaries could be discarded. The multi-edit algorithm of Devijver and Kittler (1982) divides the data cyclically into training and test sets, computing a nearest neighbour rule on the training set and deleting test points that are misclassified. The idea is to keep homogeneous clusters of training observations.

The condensing procedure of Hart (1968) goes further, trying to keep only important exterior points of these clusters. Starting with a single randomly chosen observation as the training set, each additional data item is processed one at a time, adding it to the training set only if it is misclassified by a nearest-neighbour rule computed on the current training set.

17 Unsupervised learning

In unsupervised learning one is given N observations x_1, x_N of a random p-vector X having joint density Pr(X). The goal is to directly infer the properties of this probability density without the help of a supervisor providing correct answers or degree of error.

The dimension of X is sometimes much higher than in supervised learning, and the properties of interest are often more complicated than simple location estimates. These factors are somewhat mitigated by the fact that X represents all of the variables under consideration; one is not required to infer how the properties of Pr(X) change, conditioned on the changing values of another set of variables.

In the context of unsupervised learning, there is no such direct measure of success. It is difficult to ascertain the validity of inferences drawn from the output of most unsupervised learning algorithms. One must resort to heuristic arguments not only for motivating the algorithms, as is often the case in supervised learning as well, but also for judgements as to the quality of the results.

17.1 Association rule

The goal is to find joint values of the variables $X = (X_1, X_2, \cdots, X_p)$ that appear most frequently in the data base. It is mostly applied to binary-valued data $X_j = \{0, 1\}$, where it is referred to as **Market basket analysis**. n this context the observations are sales transactions, such as those occurring at the checkout counter of a store. The variables represent all of the items sold in the store. For observation i, each variable X_j is assigned 1 if the item is purchased, 0 otherwise.

More generally, the basic goal of association rule analysis is to find a collection of prototype X-values $v_1, \cdots, v_L$ for the feature vector X, such that the probability density $Pr(v_l)$ evaluated at each of those values is relatively large. This basically means find any kind of pattern that tells us how things go together. In this way the problem can be seen as **mode finding** or **bump hunting**.

A natural estimation of $Pr(v_l)$ is the number of occurrences for which $X = v_l$. With a large number of variables however, such observations will be to few to be reliable. The problem can be restated in another way.

Instead of seeking the values x where Pr(x) is large, one seeks region of the X-space with high probability content relative to their size or support.

Definition 17.1. Support The support $\mathcal{S}_|$ of a variable jis the set of all possible values for the variable.

Let $\mathcal{S}_|$ represent the set of all possible values of the $j-th$ variable (its support) and let $s_j \subseteq \mathcal{S}_|$ be a subset of these values. The modified goal can be stated as attempting to find subsets of variable values $s_1, \cdots, s_p$ such that the probability of each of the variables is simultaneously assuming a value within its respective subset

$$\Pr\left[\bigcap_{j=1}^{p}\left(X_j \in s_j\right)\right] \tag{17.1}$$

is relatively large. $\bigcap_{j=1}^{p}\left(X_j \in s_j\right)$ is the intersection of subsets and it is called **conjunctive rule**.

17.1.1 Market basket analysis

For commercial databases $p \sim 10^4$, $N \sim 10^8$. In this cases many existing algorithm are not feasible and market basket analysis is applied.

Some simplifications are required: either s_j consists of a single value of X_j ($s_j = v_{0j}$), or it consists of the entire set of values of X_j ($s_j = S_j$). So this simplifies to the problem of identifying the subset for which s_j is a single value and and this set is named $\mathcal{J} \subset \{1, \cdots, p\}$ and for each element in this subset find the corresponding value v_{0j} such that:

$$\Pr\left[\bigcap_{j \in \mathcal{J}} (X_j = v_{0j})\right] \tag{17.2}$$

is large.

One can apply the technique of dummy variables to turn 17.2 into a problem involving only binary-valued variables. Here we assume that the support S_j is finite for each variable X_j. Specifically, a new set of variables $Z_1, \cdots, Z_K$ is created, one such variable for each of the values v_{lj} attainable by each of the original variables $X_1, \cdots, X_p$. The number of dummy variables K is

$$K = \sum_{j=1}^{p} |\mathcal{S}_j| \tag{17.3}$$

where $|\mathcal{S}_j|$ is the number of distinct values attainable by X_j. Each dummy variable is assigned the value $Z_k = 1$ (with Z_k being associated to v_{lj}), if the variable with which it is associated takes on the corresponding value to which Z_k is assigned (i.e., Z_k is associated with v_{lj}; $Z_k = 1$ if $X_j = v_{lj}$), and $Z_k = 0$ otherwise. This corresponds to find a subset of integers $\mathcal{K} \subset \{1, \cdots, K\}$ such that:

$$\Pr\left[\bigcap_{k \in \mathcal{K}} (Z_k = 1)\right] = \Pr\left[\prod_{k \in \mathcal{K}} Z_k = 1\right] \tag{17.4}$$

is large.

This is the standard formulation of the market basket problem. The set K is called an **item set**. The number of variables Z_k in the item set is called its "size" (note that the size is no bigger than p).

The estimated value of the previous expression is taken to be the fraction of observations in the database for which the conjunction is true:

$$\hat{\Pr}\left[\bigcap_{k\in\mathcal{K}}(Z_k = 1)\right] = \frac{1}{N}\sum_{i=1}^{N}\prod_{k\in\mathcal{K}} z_{ik} \tag{17.5}$$

where z_{ik} is the value of Z_k in the i-th case. This is called support or prevalence $T(\mathcal{K})$ if the item-set $\mathcal{K}$.

In association rule, a lower support bound t is specified and one seeks all sets $\mathcal{K}_l$ that can be formed from the variables Z_k with support in the data base greater than this lower bound t.

17.1.2 Apriori algorithm

For a given support threshold t The cardinality $|\{\mathcal{K}|T(\mathcal{K})\}|$ is relatively small. Any item set consisting of a subset of the items $\mathcal{K}$ must have support greater than or equal to that of $\mathcal{K}$: $\mathcal{L} \subseteq \mathcal{K} \Rightarrow T(\mathcal{L}) \geq T(\mathcal{K})$.

The first pass over the data computes the support of all single-item sets. Those whose support is less than the threshold are discarded. The second pass computes the support of all item sets of size two that can be formed from pairs of the single items surviving the first pass. Each successive pass over the data considers only those item sets that can be formed by combining those that survived the previous pass with those retained from the first pass. Passes over the data continue until all candidate rules from the previous pass have support less than the specified threshold.

The items Z_k, $k \in K$, are partitioned into two disjoint subsets, $A \cup B = K$, and written $A \Rightarrow B$. A is called antecedent and B consequent.

Association rules are defined to have several properties based on the prevalence of the antecedent and consequent item sets in the data base. The support of the rule $T(A \Rightarrow B)$ is the fraction of observations in the union of the antecedent and consequent, which is just the support of the item set K from which they were derived. It can be viewed as an estimate of the probability of simultaneously observing both item sets Pr(AandB) in a randomly selected market basket. The "confidence" or "predictability" $C(A \Rightarrow B)$ of the rule is its support divided by the support of the antecedent:

$$C(A \Rightarrow B) = \frac{T(A \Rightarrow B)}{T(A)} \tag{17.6}$$

which can be viewed as an estimate Pr(A|B).

The "expected confidence" is defined as the support of the consequent T(B), which is an estimate of the unconditional probability Pr(B). Finally, the "lift" of the rule is defined as the confidence divided by the expected confidence:

$$L(A \Rightarrow B) = \frac{C(A \Rightarrow B)}{T(B)} \tag{17.7}$$

This is an estimate of the association measure Pr(AandB)/Pr(A)Pr(B).

Suppose the item set $K = \{\text{peanutbutter}, \text{jelly}, \text{bread}\}$ and consider the rule $\{\text{peanutbutter}, \text{jelly}\} \Rightarrow \{\text{bread}\}$. A support value of 0.03 for this rule means that peanut butter, jelly, and bread appeared together in 3% of the market baskets. A confidence of 0.82 for this rule implies that when peanut butter and jelly were purchased, 82% of the time bread was also purchased. If bread appeared in 43% of all market baskets then the rule $\{\text{peanutbutter}, \text{jelly}\} \Rightarrow \{\text{bread}\}$ would have a lift of 1.95.

A confidence threshold c is set, and all rules that can be formed from those item sets with confidence greater than this value:

$$\{A \Rightarrow B | C(A \Rightarrow B) > c\} \tag{17.8}$$

The output of the entire analysis is a collection of association rules (14.7) that satisfy the constraints:

$$T(A \Rightarrow B) > t \quad \text{and} \quad C(A \Rightarrow B) > c \tag{17.9}$$

17.2 Generalized association rule

...

17.3 Cluster Analysis

The goal is grouping or segmenting a collection of objects into subsets or "clusters", such that those within each cluster are more closely related to one another than objects assigned to different clusters. Objects can be described by a set of measurements, or by relations to other objects. In addition, the goal is sometimes to arrange the clusters into a natural hierarchy. This involves successively grouping the clusters themselves.

A clustering method attempts to group the objects based on the definition of similarity supplied to it.

K-means clustering starts with guesses for the three cluster centers. Then it alternates the following steps until convergence:

- for each data point, the closest cluster center (in Euclidean distance) is identified;
- each cluster center is replaced by the coordinate-wise average of all data points that are closest to it.

This is a top-down procedure.

17.3.1 Proximity Matrices and dissimilarities

In social science experiments, participants are asked to judge by how much certain objects differ from one another. Dissimilarities can then be computed by averaging over the collection of such judgements.

This type of data can be represented by an N × N matrix D, where N is the number of objects, and each element $d_{ii'}$ records the proximity between the i-th and i′-th objects. Most algorithms use $d_{ii} = 0$ and non negative weights.

Some algorithms require a dissimilarity matrix. Given $d_j(x_{ij}, x_{i'j}$ a dissimilarity between values of the j-the attribute, we define the dissimilarity between objects i and i′ as:

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} d_j(x_{ij}, x_{i'j}) \tag{17.10}$$

Most likely d_j is the squared distance. For quantitative variables other common choice are the absolute error or correlation:

$$\rho(x_i, x_i) = \frac{\sum_j (x_{ij} - \bar{x}_i)(x_{i'j} - \bar{x}_i)}{\sqrt{\sum_j (x_{ij} - \bar{x}_i)^2 (x_{i'j} - \bar{x}_i)^2}} \tag{17.11}$$

Ordinal variables are often represented as contiguous integers. Examples are academic grades (A, B, C, D, E, F), degree of preference and so on. **Categorical or nominal variables** one can build a symmetric matrix with M values s.t. $L_{ij} = L_{ji} \geq 0$ and $L_{ii} = 0$. A popular choice is $L_{ii} = 0$.

To calculate **dissimilarity between objects** generally a weighted average is used:

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} w_j d_j(x_{ij}, x_{i'j}) \tag{17.12}$$

Let us define the average dissimilarity on the j-th attrbute as:

$$\bar{d}_j = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{i'=1}^{N} d_j(x_{ij} x_{i'j}) \tag{17.13}$$

We might define $w_j \sim \frac{1}{\bar{d}_j}$would give all attributes equal influence in characterizing overall dissimilarity between objects. In general, setting $w_j = \frac{1}{\bar{d}_j}$ for all attributes, irrespective of type, will cause each one of them to equally influence the overall dissimilarity between pairs of objects. However, if the goal is to segment the data into groups of similar objects, all attributes may not contribute equally to the (problem-dependent) notion of dissimilarity between objects.

17.3.2 Combinatorial Algorithms

Combinatorial algorithms work directly on the observed data with no direct reference to an underlying probability model. They assign each observation to a group or cluster without regard to a probability model describing the data. Each observation is uniquely labeled by an integer $i \in \{1, \cdots, N\}$.

A prespecified number of clusters $K < N$ is assumed. Each observation is assigned to one and only one cluster. Given the dissimilarities between every pair of observations we want to assign each point to a cluster by the algorithm outputting a value corresponding to the cluster.

One approach is to directly specify a mathematical loss function and attempt to minimize it through some combinatorial optimization algorithm.Since the goal is to assign close points to the same cluster, a natural loss (or "energy") function would be:

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'}) \tag{17.14}$$

where C(i) = k is an encoder assigning someway the point i to the class k. This criterion characterizes the extent to which observations assigned to the same cluster tend to be close to one another.

W(C) is referred to as **within-cluster point scatter** while

$$\mathrm{B(C)} = \frac{1}{2}\sum_{\mathrm{k=1}}^{\mathrm{K}} \sum_{\mathrm{C(i)=k}} \sum_{\mathrm{C(i')\neq k}} \mathrm{d}(\mathrm{x_i}, \mathrm{x_{i'}} \tag{17.15}$$

as the **between-cluster point scatter** and

$$\mathrm{T} = \frac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{N}} \sum_{\mathrm{i'=1}}^{\mathrm{N}} \mathrm{d_{ii'}} = \mathrm{W(C)} + \mathrm{B(C)} = \mathrm{const} \tag{17.16}$$

is the **total point scatter** which is a constant given the data. B(C) tends to be large when observations assigned to different clusters are far apart. So we have W(C) = T – B(C) minimizing W(C) is equivalent to maximizing B(C).

Such optimization by complete enumeration is feasible only for very small data sets. The number of distinct assignments is

$$\mathrm{S(N,K)} = \frac{1}{\mathrm{K!}}\sum_{\mathrm{k=1}}^{\mathrm{K}} (-1)^{\mathrm{K-k}} \binom{\mathrm{K}}{\mathrm{k}} \mathrm{k^N} \tag{17.17}$$

For example, $\mathrm{S}(10, 4) = 34105$ which is still feasible, but $\mathrm{S}(19, 4) \approx 10^{10}$. For this reason, practical clustering algorithms are able to examine only a very small fraction of all possible encoders k = C(i). The goal is to identify a small subset that is likely to contain the optimal one, or at least a good suboptimal partition. Such feasible strategies are based on iterative greedy descent.

17.3.3 K-means

The K-means algorithm is one of the most popular iterative descent clustering methods. It is intended for situations in which all variables are of the quantitative type, and squared Euclidean distance. The within-point scatter

can be written as:

$$\begin{aligned} W(C) &= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k}|x_i - x_{i'}|^2 = \sum_{k=1}^{K} N_k \sum_{C(i)=k}|x_i - \bar{x}_k|^2 \\ N_k &= \sum_{i=1}^{N} I(C(i)=k) \end{aligned} \tag{17.18}$$

where $\bar{x}_k$ is the mean vector associated with the kth-cluster. So the N observations are assigned in a way such that within each cluster the average dissimilarity of the observations from the cluster mean is minimized.

Algorithm 4: K-means algorithm

For a given cluster assignment C, the total cluster variance is minimized w.r.t. $m_1, \cdots, m_K$, yielding the means of the currently assigned clusters;

Given a current set of means the total cluster variance is minimized by assigning each observation to the closest cluster mean: $C(i) = \arg\min_{1\leq k\leq K} |x_i - m_k|^2$;

Iterate the previous steps until assignment do not change.

17.3.4 Gaussian Mixtures as Soft K-means Clustering

The K-means clustering procedure is closely related to the EM algorithm for estimating a certain Gaussian mixture model. The E-step of the EM algorithm assigns "responsibilities" for each data point based in its relative density under each mixture component, while the M-step recomputes the component density parameters based on the current responsibilities. Suppose we specify K mixture components, each with a Gaussian density having scalar covariance matrix $\sigma^2\mathbf{I}$. Then the relative density under each mixture component is a monotone function of the Euclidean distance between the data point and the mixture center. Hence in this setup EM is a "soft" version of K-means clustering, making probabilistic (rather than deterministic) assignments of points to cluster centres. With $\sigma^2 \to 0$ the models coincide.

17.4 Vector quantization

Used in image processing to compress images at a cost of loss in quality. VQ breaks the image into small blocks of pixels 2×2 for example. Suppose the image is 1024×1024 pixels, then we get 512×512 blocks of four pixels, i.e. a vector in $\mathcal{R}^4$. A k-means clustering algorithm is run in this space. Each block is approximated with its closest cluster, known as codeword. The bigger K the less lossy is the compression, and it is big enough we can achieve a lossless compression. The clustering process is called the encoding step, and the collection of centroids is called the codebook. To name and identify the codebook entries approximating the blocks will require $\log_2(\mathrm{K})$ bits per block. We also need to supply the codebook itself, which is $\mathrm{K} \times 4$ real numbers.

17.4.1 K-medoids

Squared Euclidean distance places the highest influence on the largest distances. This causes the procedure to lack robustness against outliers that produce very large distances. In the most common form, centres for each cluster are restricted to be one of the observations assigned to the cluster. This algorithm assumes attribute data, but the approach can also be applied to data described only by proximity matrices.

17.4.2 Number of cluster

Suggestions range from simple random selection to a deliberate strategy based on forward stepwise assignment. At each step a new center $\mathrm{i_k}$ is chosen to minimize the given criterion, given the centres $\mathrm{i_1}, \cdots, \mathrm{i_{k-1}}$ chosen at the previous steps. This continues for K steps, thereby producing K initial centres with which to begin the optimization algorithm.

For data segmentation K is usually defined as part of the problem. For example, a company may employ K sales people, and the goal is to partition a customer database into K segments, one for each sales person, such that the customers assigned to each one are as similar as possible.

In other cases it must be estimated typically by examining the within-cluster dissimilarity $\mathrm{W_K}$ as a function of the number of clusters K. Separate solutions are obtained for $\mathrm{K} \in \{1, \cdots, \mathrm{K_{max}}\}$. Generally $\mathrm{W_k}$ decreases with increasing K, even when the criterion is evaluated on an independent test set.

Algorithm 5: K-medoids

For a given cluster assignment C find the observation in the cluster minimizing total distance to other points in that cluster:

$$i_k^* = \underset{\{i:C(i)=k\}}{\arg\min} \sum_{C(i')=k} D(x_i, x_{i'}) \tag{17.19}$$

Then $m_k = x_{i_k^*}$, $k = 1, \cdots, K$ are the current estimates of the cluster centres;

Given a current set of clusters centres $\{m_1, \cdots, m_K\}$ minimize the total error by assigning each observation to the closest current cluster centre:

$$C(i) = \underset{1 \le k \le K}{\arg\min} D(x_i, m_k) \tag{17.20}$$

;

Iterate the previous steps until the assignment do not change.

17.4.3 Hierarchical clustering

Hierarchical clustering methods require the user to specify a measure of dissimilarity between (disjoint) groups of observations, based on the pairwise dissimilarities among the observations in the two groups. They produce hierarchical representations in which the clusters at each level of the hierarchy are created by merging clusters at the next lower level. At the lowest level, each cluster contains a single observation. At the highest level there is only one cluster containing all of the data.

Strategies for hierarchical clustering divide into two basic paradigms: agglomerative (bottom-up,) and divisive (top-down).

In agglomerative methods, the pair chosen for merging consist of the two groups with the smallest intergroup dissimilarity. For divise-methods, the split is chosen to produce two new groups with the largest between-group dissimilarity. With both paradigms there are $N - 1$ levels in the hierarchy.

Recursive binary splitting/agglomeration can be represented by a rooted binary tree.

17.4.4 Agglomerative clustering

Agglomerative clustering algorithms begin with every observation repre- senting a singleton cluster. At each of the N – 1 steps the closest two (least dissimilar) clusters are merged into a single cluster, producing one less cluster at the next higher level. Therefore, a measure of dissimilarity between two clusters (groups of observations) must be defined.

Given two groups G and H and two observations $i \in G$, $i' \in H$ the pairwise observation dissimilarity between these two points is $d_{ii'}$. The dissimilarity bbtw G and H is computed from the set of all dissimilarities.

Single linkage agglomerative clustering takes the intergroup dissimilarity to be that of the closest (least dissimalar) pair:

$$D_{SL} = \min_{i \in G, i' \in H} d_{ii'} \tag{17.21}$$

This is often called **nearest neighbour technique**. **Complete linkage** agglomerative clustering takes the intergroup dissimilarity to be that of the furthest pair (most dissimilar pair):

$$d_{CL}(G, H) = \max_{i \in G, i' \in H} d_{ii'} \tag{17.22}$$

Group average clustering uses the average:

$$d_{GA} = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{i' \in H} \tag{17.23}$$

where N are the num. observations in each group. If the data dissimilarities exhibit a strong clustering tendency, with each of the clusters being compact and well separated from others, then all three methods produce similar results.

Single linkage only requires that a single dissimilarity be small for two groups to be considered close together, irrespective of the other observation dissimilarities between the groups. It will therefore have a tendency to combine, at relatively low thresholds, observations linked by a series of close intermediate observations. This phenomenon, referred to as **chaining**, is often considered a defect of the method.

Complete linkage represents the opposite extreme. Two groups G and H are considered close only if all of the observations in their union are relatively similar. It will tend to produce compact clusters with small diameters.

However, it can produce clusters that violate the "closeness" property. That is, observations assigned to a cluster can be much closer to members of other clusters than they are to some members of their own cluster. Group average clustering represents a compromise between the two extremes of single and complete linkage. It attempts to produce relatively compact clusters that are relatively far apart. However, its results depend on the numerical scale on which the observation dissimilarities are measured.

17.4.5 Divisive clustering

The divisive paradigm can be employed by recursively applying any of the combinatorial methods such as K-means or K-medoids , with K = 2, to perform the splits at each iteration.

A better approach is by Smith et. al(1965). It begins by placing all observations in a single cluster G. It then chooses that observation whose average dissimilarity from all the other observations is largest. This observation forms the first member of a second cluster H. At each successive step that observation in G whose average distance from those in H, minus that for the remaining observations in G is largest, is transferred to H. This continues until the corresponding difference in averages becomes negative. That is, there are no longer any observations in G that are, on average, closer to those in H.

17.4.6 Self-organizing maps

...

17.5 Principal components, curves and surfaces

The principal components of a set of data in $\mathcal{R}^{\mathrm{p}}$ provide a sequence of best linear approximations to that data, of all ranks $\mathrm{q} \leq \mathrm{p}$. Given the observations $\mathrm{x}_1, \cdots, \mathrm{x}_{\mathrm{N}}$, the linear approximation is:

$$\mathrm{f}(\lambda) = \mu + \mathbf{V}_{\mathrm{q}}\lambda \tag{17.24}$$

where $\mathbf{V}_{\mathrm{q}}$ is a $\mathrm{p} \times \mathrm{q}$ matrix with q orthogonal unit vectors as columns, λ a q vector of parameters and μ a location vector in $\mathcal{R}^{\mathrm{p}}$. This is a p hyperplane.

Fitting such a model corresponds to minimize the reconstruction error:

$$\min_{\mu,\{\lambda_i\},\mathbf{V}_q} \sum_{i=1}^{N} \|x_i - \mu - \mathbf{V}_q \lambda_u\|^2 \tag{17.25}$$

We can partially optimize to obtain:

$$\begin{aligned} \hat{\mu} &= \bar{x} \\ \hat{\lambda}_i &= \mathbf{V}_q^T (x_i - \bar{x}) \end{aligned} \tag{17.26}$$

so we must find the matrix $\mathbf{V}_q$:

$$\min_{\mathbf{V}_q} \sum_{i=1}^{N} \|(x_i - \bar{x}) - \mathbf{V}_q \mathbf{V}_q^T (x_i - \bar{x})\|^2 \tag{17.27}$$

For convenience we assume $\bar{x} = 0$, if not true then one replaces with $\tilde{x}_i = x_i = \bar{x}$. The matrix

$$\mathbf{H}_q = \mathbf{V}_q \mathbf{V}_q^T \tag{17.28}$$

is a projection matrix so that each point x_i is mapped onto its rank-q reconstruction $\mathbf{H}_q x_i$, the orthogonal projection of x_i spanned by the columns $\mathbf{V}_q$.

Consider the matrix $\mathbf{X} = \left(\mathbf{x}_1^T, \cdots, \mathbf{x}_N^T\right)$ of the observations and perform the singular value decomposition:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{17.29}$$

$\mathbf{U}$ is a $N \times p$ orthogonal matrix $\mathbf{U}^T\mathbf{U} = \mathbf{I}_p$ whose columns $\mathbf{u}_j$ are called **left singular vectors**. $\mathbf{V}$ is a $p \times p$ orthogonal matrix ($\mathbf{V}^T\mathbf{V} = \mathbf{I}_p$) with columns $\mathbf{v}_j$ called the **right singular vectors**. $\mathbf{D}$ is a diagonal $p \times p$ matrix with $d_1 \geq d_2 \geq \cdots \geq d_p \geq 0$ known as **singular values**.

For each rank q, (i.e., the chosen dimensionality on which the observations are projected), the solution $\mathbf{V}_q$ consists of the first q columns of $\mathbf{V}$. The columns $\mathbf{UD}$ are called the **principal components** of $\mathbf{X}$. The N optimal $\hat{\lambda}_i$ are given by the first q principal components (the N rows of the $N \times q$ matrix $\mathbf{U}_q\mathbf{D}_q$.

17.1 shows the one-dimensional principal component line in a $\mathcal{R}^2$ input space ($p = 2$, $q = 1$). For each observation point x_i there is a closest point on the line given by $u_{i1}d_1v_1$ where v_1 is the direction of the line (v_1 is $p = 2$ dimensional vector). $\hat{\lambda}_1 = d_{i1}d_1$ measures distance along the line from the origin.

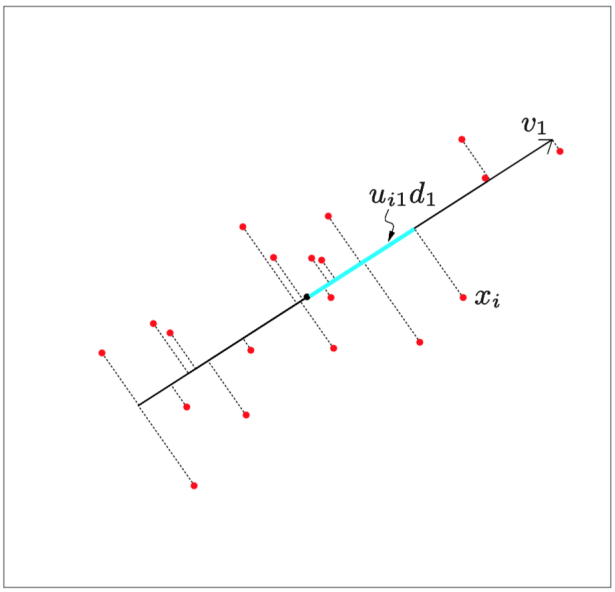


Figure 17.1: Example of SVD mapping of data from a $\mathcal{R}^2$ space (i.e., a plane) onto a line $\mathcal{R}$

17.5.1 Applications

Principal components are a useful tool for dimension reduction and compression, for example for hand-written digits. If the inputs are 16×16 images, inputs rely in a $\mathcal{R}^{256}$ space.

Consider PCA on hand-written digits of a single number, for example a 3, and perform two principal components analysis $\mathrm{u_{i1}d_1}$ and $\mathrm{u_{i2}d_2}$ of each observation and calculate the quantiles $5\%, 25\%, 50\%75\%, 95\%$ indicated by the grid in 17.2. Red circles indicate those images close to the vertices of the grid where distance is in terms of the projected coordinates (but some weight is given to the components in the orthogonal space (?)). On the right of 17.2 the images corresponding to the red circles are shown and this allows to visualize the nature of the first two PC. $\mathrm{v_1}$, the horizontal direction, mainly accounts for the length of lower "tail" of the "3", $\mathrm{v_2}$ the vertical axis accounts for character thickness. The parametrized model is:

$$\hat{\mathrm{f}}(\lambda) = \bar{\mathrm{x}} + \lambda_1 \mathrm{v}_1 + \lambda_2 \mathrm{v}_2 \tag{17.30}$$

that in terms of images is something like the one in 17.3.

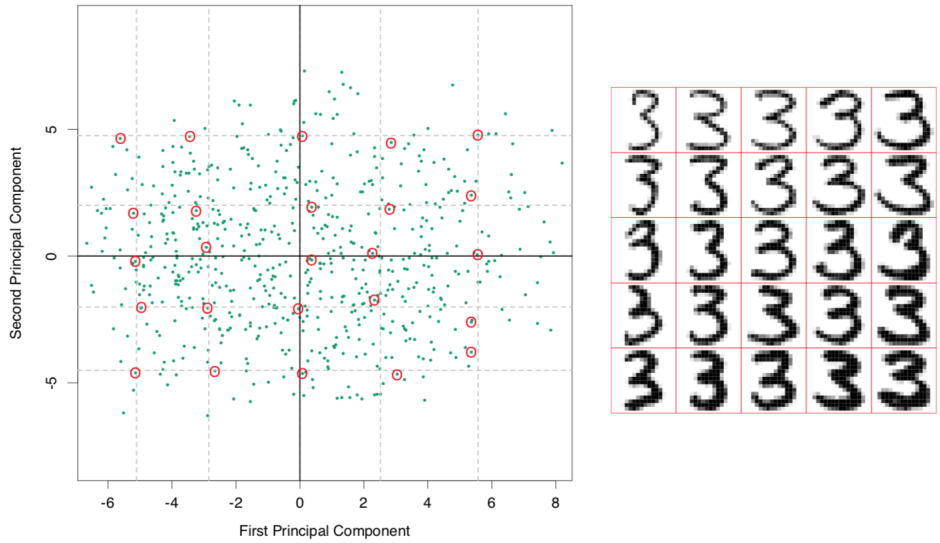


Figure 17.2: 2 Principal components analysis applied on the 3 hand-written digits.

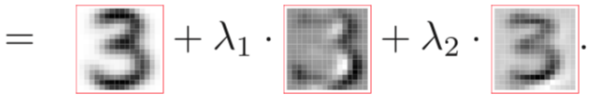


Figure 17.3: Equivalence of the parametrized model using image representation.

Although there are a possible 256 principal components, approximately 50 account for 90% of the variation in the threes, 12 account for 63%.

A relatively small subset of the principal components serve as excellent lower-dimensional features for representing the high-dimensional data.

The same technique, with some further calculations, can be applied to digital signature.

17.5.2 Prinipal curves and surfaces

Principal curves generalize the concept of the principal component line to obtain smoother curves.

Let $\mathrm{f}(\lambda)$ be a parametrized smooth curve in $\mathcal{R}^{\mathrm{p}}$. It is a vector function of with p coordinates, each a smooth function of the single parameter λ. For example λ can be chosen to be the arc-length along the curve from a fixed origin. For each data value x, let $\lambda_{\mathrm{f}}(\mathrm{x})$ define the closest point on the curve to x. Then $\mathrm{f}(\lambda)$ is called a principal component curve for the distribution of the random vector X if:

$$\mathrm{f}(\lambda) = \mathbf{E}(\mathrm{X}|\lambda_{\mathrm{f}}(\mathrm{X}) = \lambda) \tag{17.31}$$

So $\mathrm{f}(\lambda)$ is the average of all data points that project to it (**self-consistency** property). Although in practice, continuous multivariate distributes have infinitely many principal curves (Duchamp and Stuetzle, 1996), we are interested mainly in the smooth ones.

A principal curve is defined by its coordinate functions: $\mathrm{f}(\lambda) = [\mathrm{f}_1(\lambda), \mathrm{f}_{\mathrm{p}}(\lambda)]$ and let $\mathrm{X}^{\mathrm{T}} = (\mathrm{X}_1, \cdots, \mathrm{X}_{\mathrm{p}})$. Consider the following alternate steps:

$$\begin{aligned} &1) \quad \hat{\mathrm{f}}_{\mathrm{j}}(\lambda) \leftarrow \mathbf{E}\big(\mathrm{X}_{\mathrm{j}}|\lambda(\mathrm{X}) = \lambda\big); \quad \mathrm{j} = 1, \cdots, \mathrm{p} \\ &2) \quad \hat{\lambda}_{\mathrm{f}}(\mathrm{x}) \leftarrow \arg\min_{\lambda'} \|\mathrm{x} - \hat{\mathrm{f}}(\lambda')\|^2 \end{aligned} \tag{17.32}$$

In the first λ is fixed and it enforces the self-consistency requirement. In the second the curve is fixed and it finds the closest point on the curve to each data point.

With finite data, the principal curve algorithm starts with the linear principal component, and iterates the two steps until convergence. A scatterplot smoother is used to estimate the conditional expectations in the frist step by smoothing each X_{j} as a function of the arc-length $\lambda(\mathrm{X})$, and the projection

in the second step is done for each of the observed data points. Proving convergence in general is difficult, but one can show that if a linear least squares fit is used for the scatterplot smoothing, then the procedure converges to the first linear principal component, and is equivalent to the power method for finding the largest eigenvector of a matrix.

Principal surfaces have exactly the same form as principal curves, but are of higher dimension. The mostly commonly used is the two-dimensional principal surface, with coordinate functions

$$f(\lambda_1, \lambda_2) = \left[f_1(\lambda_1, \lambda_2), \cdots, f_p(\lambda_1, \lambda_2)\right] \tag{17.33}$$

Principal surfaces of dimension greater than two are rarely used, since the visualization aspect is less attractive, as is smoothing in high dimensions.

17.5.3 Spectral Clustering

Traditional clustering methods like K-means use a spherical or elliptical metric to group data points. Hence they will not work well when the clusters are non-convex, such as the concentric circles.

Spectral clustering solves this problem by firstly building a $N \times N$ matrix of pairwise similarities $s_{ij} \geq 0$ between all observation pairs.We represent the observations in an undirected similarity graph $G = \langle V, E \rangle$. The N vertices v_i are the observations and pairs of vertices are connected by an edge if their similarity is positive or exceeds some threshold. Edges are weighted by s_{ij}.

Clustering now is a graph-partition problem where we identify connected components with clusters. We want to partition the graph such that edges between different groups have low weight and within a group have high weight.

Consider N points $x_i \in \mathcal{R}^p$ and let d_{ij} be the Euclidean distance between x_i and x_j. We will use as similarity matrix the radial-kernel matrix:

$$s_{ij} = \epsilon^{-\frac{d_{ij}^2}{c}} \tag{17.34}$$

where $c > 0$ is a scale parameter. A popular way to define the similarity matrix is the **mutual K-nearest-neighbour graph**. Define $\mathcal{N}_k$ the symmetric set of nearby pairs of points: a pair (i, j) is in $\mathcal{N}_k$ if i is in the K-neighbourhood of j or viceversa. Then we connect all NN and give them edge-weight $w_{ij} = s_{ij}$, otherwise the weight is 0. Alternatively, a fully connected graph includes all pairwise edges with weights $w_{ij} = s_{ij}$, and the local

behaviour is controlled by the scale parameter c. The matrix $\mathbf{W} = \{w_{ij}\}$ is called **adjacency matrix**; the **degree of vertex** i is:

$$g_i = \sum_j w_{ij} \tag{17.35}$$

Let $\mathbf{G}$ be a diagonal matrix of d_{ij}. The **unnormalized graph Laplacian** is

$$\mathbf{L} = \mathbf{G} - \mathbf{W} \tag{17.36}$$

Spectral clustering finds the m eigenvectors $\mathbf{Z}_{N\times m}$ corresponding to the m smallest eigenvalues of $\mathbf{G}$. Using a standard method like K-means, we then cluster the rows of $\mathbf{Z}$ to yield a clustering of the original data points.

Spectral clustering works because for any vector $\mathcal{V}$:

$$\mathcal{V}^T\mathbb{L}\mathcal{V} = \sum_{i=1}^{N} g_i f_i^2 - \sum_{i=1}^{N}\sum_{j=1}^{N} f_i f_j w_{ij} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}(f_i - f_j)^2 \tag{17.37}$$

A small value of $\mathcal{V}^T\mathbb{L}\mathcal{V}$ will be achieved for coordinates that are close together. $\mathbb{1}^T\mathbb{L}\mathbb{1} = 0$ for any graph: the trivial constant vector corresponding to the 0 eigenvalue. If the graph is connected $\mathbb{1}^T\mathbb{L}\mathbb{1}$ is the only 0 eigenvector.

With m connected components, the nodes can be reordered so that L is block diagonal with a block for each connected component. Then L has m eigenvectors of eigenvalue 0, and the eigenspace of eigenvalue 0 is spanned by the indicator vectors of the connected components. In practice one has strong and weak connections, so 0 eigenvalues are approximated by small eigenvalues.

Spectral clustering is an interesting approach for finding non-convex clusters.

17.5.4 Kernel Principal components (kernel-PCA or kPCA)

Support Vector Machines Kernel trick maps instances into a very high-dimensional space (called the feature space), enabling non-linear classification and regression. Recall that a linear decision boundary in the high-dimensional feature space corresponds to a complex nonlinear decision boundary in the original space. It turns out that the same trick can be applied to PCA, making it possible to perform complex non-linear projections for dimensionality reduction. Application of a kernel means expanding the

features-space by non-linear transformation. On this new higher dimensional space we can apply PCA.

Let $\phi()$ be a non-linear transformation mapping data from the input space $\mathcal{X}$ to some feature space $\mathcal{H}$. Then, the problem is the same of 2, the only difference is that the covariance matrix is the one calculated in the feature space, not in the input space. Now ψ_{l} lie in the span of the mapped data, with the coefficient given by the $\mathrm{l}-$ th eigenvector.

$$\lambda_\ell \psi_\ell = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \langle \phi(\mathbf{x}_{\mathrm{i}}), \psi_\ell \rangle \phi(\mathbf{x}_{\mathrm{i}}) \tag{17.38}$$

As in 5.59 we can express the principal components as a linear combination, in this case not in terms of input vectors but in terms of feature vectors:

$$\psi_\ell = \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \phi(\mathbf{x}_{\mathrm{i}}) \tag{17.39}$$

and substituting into the previous equation:

$$\begin{aligned} \lambda_\ell \psi_\ell = \lambda_\ell \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \phi(\mathbf{x}_{\mathrm{i}}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \langle \phi(\mathbf{x}_{\mathrm{i}}), \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \phi(\mathbf{x}_{\mathrm{i}}) \rangle \phi(\mathbf{x}_{\mathrm{i}}) \Rightarrow \\ \lambda_\ell \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \phi(\mathbf{x}_{\mathrm{i}}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \phi(\mathbf{x}_{\mathrm{i}}) \langle \phi(\mathbf{x}_{\mathrm{i}}), \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \phi(\mathbf{x}_{\mathrm{i}}) \rangle \Rightarrow \\ \Rightarrow \lambda_\ell \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \phi(\mathbf{x}_{\mathrm{i}}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \phi(\mathbf{x}_{\mathrm{i}}) \phi(\mathbf{x}_{\mathrm{i}})^{\mathrm{T}} \sum_{\mathrm{j}=1}^{\mathrm{n}} \alpha_{\mathrm{j}\ell} \phi(\mathbf{x}_{\mathrm{j}}) \Rightarrow \end{aligned} \tag{17.40}$$

The key step is now to express this in terms of the kernel function $\mathrm{k}(\mathbf{x}_{\mathrm{i}}, \mathbf{x}_{\mathrm{j}}) = \phi(\mathbf{x}_{\mathrm{n}})^{\mathrm{T}} \phi(\mathbf{x}_{\mathrm{m}})$, which we do by multiplying both sides by $\phi(\mathbf{x}_{\mathrm{p}})^{\mathrm{T}}$ to give

$$\begin{aligned} \lambda_\ell \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \phi(\mathbf{x}_{\mathrm{p}})^{\mathrm{T}} \phi(\mathbf{x}_{\mathrm{i}}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \phi(\mathbf{x}_{\mathrm{p}})^{\mathrm{T}} \phi(\mathbf{x}_{\mathrm{i}}) \sum_{\mathrm{j}=1}^{\mathrm{n}} \alpha_{\mathrm{j}\ell} \phi(\mathbf{x}_{\mathrm{i}})^{\mathrm{T}} \phi(\mathbf{x}_{\mathrm{j}}) \Rightarrow \\ \Rightarrow \lambda_\ell \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}\ell} \mathrm{k}(\mathbf{x}_{\mathrm{p}}, \mathbf{x}_{\mathrm{i}}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{k}(\mathbf{x}_{\mathrm{p}}, \mathbf{x}_{\mathrm{i}}) \sum_{\mathrm{j}=1}^{\mathrm{n}} \alpha_{\mathrm{j}\ell} \mathrm{k}(\mathbf{x}_{\mathrm{i}}, \mathbf{x}_{\mathrm{j}}) \end{aligned} \tag{17.41}$$

This can be rewritten in matrix notationas

$$\mathbf{K}^2 \alpha_{\mathrm{i}} = \lambda_{\mathrm{i}} \mathrm{N} \mathbf{K} \alpha_{\mathrm{i}} \tag{17.42}$$

where α_{i} is a N-dimensional column vector with elements $\alpha_{\ell\mathrm{i}}$ and K is the *Gram matrix* with entries $\langle\phi(\mathbf{x}_{\mathrm{i}}), \phi(\mathbf{x}_{\mathrm{j}})\rangle$.

One can find solutions for α_{i} by solving the following eigenvalue problem:

$$\mathbf{K}\alpha_{\ell} = \lambda_{\ell}\mathrm{N}\alpha_{\mathrm{i}} \tag{17.43}$$

So the expansions require only the inner products, without the need to know exactly the mapping function.

In 1.4.1 we have seen how to compute the matrix Z = UD. If X is not centred:

$$\begin{aligned} \tilde{\mathrm{X}} &= (\mathrm{I} - \mathrm{M})\mathrm{X} \\ \mathrm{M} &= \frac{1}{\mathrm{N}}\mathbf{1}\mathbf{1}^{\mathrm{T}} \end{aligned} \tag{17.44}$$

where M is mean operator. Thus, we compute the eigenvectors of $(\mathrm{I} - \mathrm{M})\mathrm{K}(\mathrm{I} - \mathrm{M})$:

$$\begin{aligned} &\tilde{\mathrm{K}} = \tilde{\mathrm{X}}\tilde{\mathrm{X}}^{\mathrm{T}} = (\mathrm{I} - \mathrm{M})\mathrm{X}\mathrm{X}^{\mathrm{T}}(\mathrm{I} - \mathrm{M})^{\mathrm{T}} = (\mathrm{I} - \mathrm{M})\mathrm{K}(\mathrm{I} - \mathrm{M}) = \mathrm{U}\Sigma^{2}\mathrm{U}^{\mathrm{T}} \\ &\tilde{\mathrm{K}} = \mathrm{Q}\Lambda\mathrm{Q}^{-1} \end{aligned} \tag{17.45}$$

Kernel PCA interprets the kernel matrix $\mathrm{K} = \mathrm{K}(\mathrm{xi}, \mathrm{xi}')$ as an inner-product matrix of the implicit features $\langle\Phi(\mathrm{x}_{\mathrm{i}}), \Phi(\mathrm{x}'_{\mathrm{i}})\rangle$ and finding its eigenvectors. The elements of the m-th component z_{m} (m-th column of Z) can be written (up to centering) as $\mathrm{z}_{\mathrm{im}} = \sum_{\mathrm{j}=1}^{\mathrm{N}} \alpha_{\mathrm{jm}}\mathrm{K}(\mathrm{x}_{\mathrm{i}}, \mathrm{x}_{\mathrm{j}})$, where $\alpha_{\mathrm{jm}} = \mathrm{u}_{\mathrm{jm}}/\mathrm{d}_{\mathrm{m}}$.

17.5.5 Sparse Principal components

...

17.6 The pre-image problem

The kernel trick exploits the fact that a great number of data processing techniques do not depend explicitly on the data itself, but rather on a similarity measure between them and one of these measures is the inner product. In the kernel trick, the the inner product is replaced with a reproducing kernel (i.e., a positive definite symmetric function). The kernel is a non-linear transformation that maps input data onto a feature space: applying the kernel in the input space is equivalent to applying the inner product in this new space as a similarity measure. The advantage is that the resulting non-linear algorithms show significant performance improvements while keeping the same computational complexity.

The reverse transformation from the feature space back to the input space is called **reverse mapping** and is important in kernel PCA for signal and image processing. Unfortunately, **the reverse mapping generally does not exist** and only most elements in the feature space, including the equivalent of denoised signals, do not have a valid pre-image in the input-space. **The pre-image problem consists of finding an approximate solution** by identifying data in the input space based on their corresponding features that are mapped close enough to desired data in the feature space. The pre-image problem is essentially a dimensionality reduction problem.

Most well-known statistical linear techniques can be formulated as inner product between pairs of data. Thus, applying any non-linear transformation to the data can only impact the values of the resulting inner products. Therefore one does not need to compute such transformation explicitly but needs only to replace the inner product operator with an appropriate kernel, i.e., a **symmetric hermitian function** 1.13 (with symmetric meaning $\mathrm{k}(\mathrm{x},\mathrm{y}) = \mathrm{k}(\mathrm{y},\mathrm{x})$. The only restriction is that the latter defines an inner product in some space. A sufficient condition for this is given by t e**Mercer's theorem ??** which states that any continuous positive definite function on $\mathcal{X}$ can be expressed as an inner product in some space.

Furthermore, Moore-Aronszajn theorem states that to any positive semidefinite kernel k it corresponds a unique reproducing kernel Hilbert space (rkHs) whose inner product $\langle\cdot\rangle$ usually called reproducing kernel is k itself.

A rkHs is a Hilbert space of functions for which point evaluations are bounded, and where the existence and uniqueness of the reproducing kernel is guaranteed by the Riesz representation theorem. In fact, let $\mathcal{H}$ be a Hilbert space of functions defined on some compact $\mathcal{X}$, for which the evaluation $\psi(\mathrm{x})$ of the function $\psi \in \mathrm{H}$ is bounded for all $\mathrm{x} \in \mathcal{X}$. By this theorem, there exists a unique function $\psi(\mathrm{x}) \in \mathcal{H}$ such as $\psi(\mathrm{x}) = \langle \psi, \phi(\mathrm{x})\rangle_{\mathcal{H}}$. This function has the following property:

$$\|(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j}) = \langle \phi(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j})\rangle_{\mathcal{H}} \tag{17.46}$$

Distances can be evaluated using the kernel trick:

$$\begin{aligned} \|\phi(\mathbf{x}_\mathrm{i}) - \phi(\mathbf{x}_\mathrm{j})\|^2_{\mathcal{H}} &= \langle \phi(\mathbf{x}_\mathrm{i}) - \phi(\mathbf{x}_\mathrm{j}), \phi(\mathbf{x}_\mathrm{i}) - \phi(\mathbf{x}_\mathrm{j})\rangle_{\mathcal{H}} = \\ &= \|(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{i}) - 2\|(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j}) + \|(\mathbf{x}_\mathrm{j}, \mathbf{x}_\mathrm{j}) \end{aligned} \tag{17.47}$$

Most of the kernels can be divided into two groups:

- projective kernels are functions of inner product (for example the polynomial kernel);
- radial kernels or isotropic kernels are functions of distance (for example the Gaussian kernel).

These kernels map the data onto a higher dimensional space, even infinite in the case of the Gaussian kernel.

Theorem 3. Representer theorem For any function $\psi \in$ CMcalH minimizing a regularized cost function in the form of

$$\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{f}(\mathrm{y_i}, \psi(\mathbf{x_i})) + \eta \mathrm{g}\Big(\|\psi\|_{\mathcal{H}}^2\Big) \tag{17.48}$$

with $\mathrm{f}(\cdot)$ some loss function and $\mathrm{g}(\cdot)$ a strictly monotonic increasing function on CMcalR_+ can be written as an expansion in terms of available data:

$$\psi = \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_\mathrm{i} \phi(\mathbf{x_i}) \tag{17.49}$$

So even in an infinite rkHs, one only needs to work in the subspace spanned by the n images of the training data.

17.6.1 Solving the pre-image problem

A problem is ill-posed if at least one of the following conditions is violated:

- a solution exist;
- it is unique
- it depends continuously on the data.

Identifying the pre-image problem is typically an ill-posed problem due to the higher dimensionality of the feature space. As so, many elements might not have a pre-image in the input space and if it exists it might not be unique. The goal becomes to look for an approximate solution whose map is as close as possible.

The pre-image problem can be seen as

$$\mathrm{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} \left| \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}} \phi(\mathbf{x}_{\mathrm{i}}) - \phi(\mathbf{x}) \right|_{\mathcal{H}}^2 \tag{17.50}$$

Using equation 17.46:

$$\mathrm{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} \|(\mathbf{x},\mathbf{x}) - 2\sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}} \|(\mathbf{x},\mathbf{x}_{\mathrm{i}}) + \left(\sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}} \|(\mathbf{x}_{\mathrm{i}},\mathbf{x}_{\mathrm{i}}) \right)^2 \tag{17.51}$$

where the last term can be discarded because it does not depend on $\mathbf{x}$. This is equivalent to the minimization of the distance.

17.6.2 Exact pre-image

Suppose there exists an exact pre-image ψ, i.e., x^* such that $\phi(\mathrm{x}^*) = \psi$. Then, the minimization problem is equivalent to finding the pre-image. The pre-image can be computed when the kernel is an invertible function of $\langle \mathbf{x}_{\mathrm{i}} \mathbf{x}_{\mathrm{j}} \rangle$, such as some projective kernels including odd degree polynomial kernel and the sigmoid kernel. Let $\mathrm{h} : \mathcal{R} \to \mathcal{R}$ defines the inverse function such that $\mathrm{h}\big(\mathrm{k}(\mathbf{x}_{\mathrm{i}},\mathbf{x}_{\mathrm{j}})\big) = \langle \mathbf{x}_{\mathrm{i}},\mathbf{x}_{\mathrm{j}} \rangle$. Given an orthonormal base in the input space $\{\mathrm{e}_1, \mathrm{e}_2, \cdots\}$, every $\mathbf{x} \in \mathcal{X}$ can be written as

$$\mathbf{x} = \sum_{\mathrm{j}=1}^{\mathrm{N}} \langle \mathrm{e}_{\mathrm{j}}, \mathbf{x} \rangle \mathrm{e}_{\mathrm{j}} = \sum_{\mathrm{j}=1}^{\mathrm{N}} \mathrm{h}\big(\mathrm{k}(\mathrm{e}_{\mathrm{j}}, \mathbf{x})\big) \mathrm{e}_{\mathrm{j}} \tag{17.52}$$

So, the exact pre-image x^* of $\psi = \phi(\mathrm{x}^*)$ can be expanded as

$$\mathbf{x}^* = \sum_{\mathrm{j}=1}^{\mathrm{N}} \mathrm{h}\left(\sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}} \mathrm{k}(\mathrm{e}_{\mathrm{j}}, \mathbf{x}_{\mathrm{i}}) \right) \mathrm{e}_{\mathrm{j}} \tag{17.53}$$

Likewise, when the kernel is an invertible function of the distance, such as radial kernels, a similar expression can be derived $4\langle \mathbf{x}^*, \mathrm{e}_{\mathrm{j}} \rangle = \|\mathbf{x}^* + \mathrm{e}_{\mathrm{j}}\|^2 - \|\mathbf{x}^* + \mathrm{e}_{\mathrm{j}}\|^2$.

Unfortunately for a large class of kernels the pre-image does not exist. The following are approximate solutions.

17.6.3 Gradient descent techniques

It is an iterative procedure where the current guess $\mathbf{x}_t^*$ is updated by taking a step in the opposite direction of the gradient of the objective function:

$$\mathbf{x}_{t+1}^* = \mathbf{x}_t^* - \eta_t \nabla_{\mathbf{x}} J(\mathbf{x}_t^*) \tag{17.54}$$

where η_t is a step size parameter. As an alternative to the gradient descent, one may use more sophisticated techniques such as the Newton's method (8.7.3). Unfortunately, the objective function is non-linear and non-convex, so the procedure must be run many times with several different starting values to avoid local minima solutions.

17.7 Fixed point iteration method

For most kernels, the expression in 17.51 has a closed-form expression. By setting this expression to 0, this greatly simplifies the optimization scheme resulting into a fixed-point iterative technique. Taking for example the Gaussian kernel the objective function becomes:

$$\Xi(\mathbf{x}) = -2 \sum_{i=1}^{n} \alpha_i e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}} \tag{17.55}$$

and its gradient

$$\nabla_{\mathbf{x}} \Xi(\mathbf{x}) = -\frac{2}{\sigma^2} \sum_{i=1}^{n} \alpha_i e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}} (\mathbf{x} - \mathbf{x}_i) \tag{17.56}$$

We get the pre-image setting this gradient to 0:

$$x_{t+1}^* = \frac{\sum_{i=1}^{n} \alpha_i k(\mathbf{x}_t^*, \mathbf{x}_i) \mathbf{x}_i}{\sum_{i=1}^{n} \alpha_i k(\mathbf{x}_t^*, \mathbf{x}_i)} \tag{17.57}$$

with $k(\mathbf{x}_t^*, \mathbf{x}_i) = e^{-\frac{\|\mathbf{x}_t^* - \mathbf{x}_i\|^2}{2\sigma^2}}$. Instead for the polynomial kernel with degree p:

$$x_{t+1}^* = \sum_{i=1}^{n} \alpha_i \left(\frac{\langle \mathbf{x}_t^*, \mathbf{x}_i \rangle + c}{\langle \mathbf{x}_t^*, \mathbf{x}_t^* \rangle + c} \right)^{p-1} \mathbf{x}_i \tag{17.58}$$

Such an iterative method still suffers from local minima and tends to be unstable, for example when the denominator goes to 0. However, regularized solutions exist.

An interesting fact is that the solution lies in the span of available data, taking the form $\mathbf{x}^* = \sum_{\mathrm{i}=1}^{\mathrm{n}} \beta_{\mathrm{i}} \mathbf{x}_{\mathrm{i}}$. So the search space is controlled as opposed to the gradient descent which explores the entire space.

17.7.1 Learning pre-image map

Even a learning machine cab be constructed and trained on the elements from the feature space and estimated (target points) on the input space points. We seek a function $\Gamma^*(\phi(\mathbf{x}_{\mathrm{i}})) = \mathbf{x}_{\mathrm{i}}$. The function must be defined on a vector space. Also, one can decompose the function in order to have functions outputting a single scalar output for every dimension in $\mathcal{X}$:

$$\begin{aligned} \Gamma^* &= \{\Gamma_1^*, \Gamma_2^*, \cdots, \Gamma_{\dim \mathcal{X}}^*\} \\ \Gamma_{\mathrm{m}}^* &: \mathcal{R}^{\mathrm{k}} \Rightarrow \mathcal{R} \end{aligned} \tag{17.59}$$

Each of these functions is obtained by solving the optimization problem:

$$\Gamma_{\mathrm{m}}^* = \arg\min_{\Gamma} \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{f}([\mathbf{x}_{\mathrm{i}}], \Gamma(\psi)) + \eta \mathrm{g}\Big(\|\mathrm{Gamma}\|^2\Big) \tag{17.60}$$

where f is the loss function. For example by taking the distance as the loss-function:

$$\Gamma_{\mathrm{m}}^* = \arg\min_{\Gamma} \sum_{\mathrm{i}=1}^{\mathrm{n}} \| [\mathbf{x}_{\mathrm{i}}]_{\mathrm{m}} - \Gamma(\psi)\|^2 + \eta \mathrm{g}\Big(\|\mathrm{Gamma}\|^2\Big) \tag{17.61}$$

This optimization problem can be solve by matrix inversion.

17.7.2 Multidimensional scaling (MDS)

MDS techniques embed data in a low-dimensional space by pairwise distances. In the rkHs the distance is

$$\delta_{\mathrm{i}} = \|\psi - \phi(\mathbf{x}_{\mathrm{i}})\|_{\mathcal{H}} \tag{17.62}$$

and in the input space it is $\|\mathbf{x}^* - \mathbf{x}_{\mathrm{i}}\|$. Ideally, the distances are preserved:

$$\|\psi - \phi(\mathbf{x}_{\mathrm{i}})\|_{\mathcal{H}}^2 = \|\mathbf{x}^* - \mathbf{x}_{\mathrm{i}}\|^2 \tag{17.63}$$

. If $\psi = \phi(\mathbf{x}^*)$, then $\mathbf{x}^* = \mathbf{x}_\mathrm{i}$. One way to solve this is to minimize the mean square error between these distances:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \left| \|\mathbf{x} - \mathbf{x}_\mathrm{i}\|^2 - \|\psi - \phi(\mathbf{x}_\mathrm{i})\|_{\mathcal{H}}^2 \right|^2 \tag{17.64}$$

To solve this optimization problem, a fixed-point iteration method can be used by setting the gradient to 0:

$$\mathbf{x}^* = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}} \big(\|\mathbf{x}^* - \mathbf{x}_\mathrm{i}\|^2 = \delta_\mathrm{i}^2\big)\mathbf{x}_\mathrm{i}}{\sum_{\mathrm{i}=1}^{\mathrm{n}} \big(\|\mathbf{x}^* - \mathbf{x}_\mathrm{i}\|^2 = \delta_\mathrm{i}^2\big)} \tag{17.65}$$

Another approach to solve this problem is to consider separately the identities 17.63 recalling the definition 17.62 and solving n: equations:

$$2\langle \mathbf{x}^*, \mathbf{x}_\mathrm{i} \rangle = \langle \mathbf{x}^*, \mathbf{x}^* \rangle + \langle \mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{i} \rangle - \delta_\mathrm{i}^2 \tag{17.66}$$

$\langle \mathbf{x}^*, \mathbf{x}^* \rangle$ can be identified in the case of centred data by taking the average of both sides:

$$\langle \mathbf{x}^*, \mathbf{x}^* \rangle = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \Big(\delta_\mathrm{i}^2 - \langle \mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{i} \rangle \Big) \tag{17.67}$$

Let ϵ be the vector having entries equal to $\frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \big(\delta_\mathrm{i}^2 - \langle \mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{i} \rangle \big)$, then:

$$\begin{aligned} 2\mathbf{X}^\mathrm{T}\mathbf{x}^* &= \mathrm{diag}\mathbf{X}^\mathrm{T}\mathbf{X} - \Big[\delta_1^2, \delta_2^2, \cdots, \delta_\mathrm{n}^2\Big]^\mathrm{T} + \epsilon \\ \Rightarrow \mathbf{x}^* &= \frac{1}{2}\Big(\mathbf{X}\mathbf{X}^\mathrm{T}\Big)^{-1}\mathbf{X}\Big(\mathrm{diag}\mathbf{X}^\mathrm{T}\mathbf{X} - \Big[\delta_1^2, \delta_2^2, \cdots, \delta_\mathrm{n}^2\Big]^\mathrm{T} + \not{\epsilon}\Big) \end{aligned} \tag{17.68}$$

where the term $\Big(\mathbf{X}\mathbf{X}^\mathrm{T}\Big)^{-1}\mathbf{X}\epsilon$ goes to 0 thanks to the assumption of centred data.

To keep this technique tractable in practice, only a certain neighbourhood is considered in the pre-image estimation.

17.7.3 Conformal map approach

One may also propose a pre-image method to preserve the product measures, which also preserves the angular measures since $\mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{x}_\mathrm{j}/\big(\|\mathbf{x}_\mathrm{i}\|\|\mathbf{x}_\mathrm{j}\|\big)$ defines the cosine of the angle.

A coordinate system oin the rkHs is constructed with an isometry with respect to the input space. The model is not coupled with any constraint on the coordinate functions as opposed to the orthogonality constraint between the functions in the kernel PCA.

Each of the n-coordinate functions can be rewritten as a linear expansion of the available imagesL $\Phi_\ell = \sum_{\mathrm{i}=1}^{\mathrm{n}} \theta_{\ell,\mathrm{i}} \phi(\mathbf{x}_\mathrm{i})$, with unknown weights. Rearranging:

$$\mathbf{\Phi}_{\mathbf{x}_\mathrm{i}} = [\langle \Phi_1, \phi(\mathbf{x}_\mathrm{i}) \rangle, \langle \Phi_\mathrm{w}, \phi(\mathbf{x}_\mathrm{i}) \rangle \cdots] \tag{17.69}$$

Preservation of the inner products yields:

$$\mathbf{\Phi}_{\mathbf{x}_\mathrm{i}}^{\mathrm{T}} \mathbf{\Phi}_{\mathbf{x}_\mathrm{i}} = \mathbf{x}_\mathrm{i}^{\mathrm{T}} \mathbf{x}_\mathrm{j} \tag{17.70}$$

This can be minimized by minimizing the fitness error:

$$\min_{\Psi_1, \cdots, \Psi_\mathrm{n}} \sum_{\mathrm{i,j}=1}^{\mathrm{n}} \| \mathbf{x}_\mathrm{i}^{\mathrm{T}} \mathbf{x}_\mathrm{j} - \Psi_{\mathbf{x}_\mathrm{i}}^{\mathrm{T}} \Psi_{\mathbf{x}_\mathrm{j}} \|^2 + \eta \sum_{\ell=1}^{\mathrm{n}} \| \Psi_\ell \|_{\mathcal{H}}^2 \tag{17.71}$$

where the second term is the regularization term. Rewritten in matrix form, it becomes:

$$\min \Theta \frac{1}{2} \sum_{\mathrm{i,j}=1}^{\mathrm{n}} \| \mathbf{X}^{\mathrm{T}} \mathbf{X} - \mathbf{K} \mathbf{\Theta}^{\mathbf{T}} \mathbf{\Theta} \mathbf{K} \|_{\mathrm{F}}^2 + \eta \mathrm{tr} \Big(\mathbf{\Theta}^{\mathbf{T}} \mathbf{\Theta} \mathbf{K} \Big) \tag{17.72}$$

where $\| \cdot \|_\mathrm{F}$ is the Frobenius norm, i.e., the root of sum of squared (absolute) values of all its elements: $\| \mathrm{M} \|_\mathrm{F}^2 = \mathrm{tr}(\mathrm{M}^{\mathrm{T}} \mathrm{M})$. By taking the derivative of this expression with respect to $\Theta^{\mathrm{T}} \Theta$ one obtains

$$\mathbf{\Theta}^{\mathbf{T}} \mathbf{\Theta} = \mathbf{K}^{-\mathbf{1}} \Big(\mathbf{X}^{\mathbf{T}} \mathbf{X} - \eta \mathbf{K}^{-\mathbf{1}} \Big) \mathbf{K}^{-\mathbf{1}} \tag{17.73}$$

Now one can determine the pre-image of $\psi = \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_\mathrm{i} \phi(\mathbf{x}_\mathrm{i})$. Given the system of coordinates

17.8 Non-negative matrix factorization

It assumes non-negative data, i.e. images. Given $\mathbf{X} \in \mathcal{R}^{\mathrm{N} \times \mathrm{p}}$, it can be approximated with

$$\mathbf{X} \sim \mathbf{W} \mathbf{H} \tag{17.74}$$

where $\mathbf{W} \in \mathcal{R}^{N \times r}$ and $\mathbf{H} \in \mathcal{R}^{r \times p}$, $r \leq \max(N, p)$. We assume $x_{ij}, w_{ik}, h_{kj} \geq 0$. $\mathbf{W}$ and $\mathbf{H}$ are found by maximizing

$$L(\mathbf{W}, \mathbf{H}) = \sum_{i=1}^{N} \sum_{j=1}^{p} \left[x_{ij} \log(\mathbf{WH}_{ij}) - \mathbf{WH}_{ij} \right] \tag{17.75}$$

This is the log-likelihood from a model in which x_{ij} has a Poisson distribution with mean $(\mathbf{WH})_{ij}$ -quite reasonable for positive data.

The following alternating algorithm (Lee and Seung, 2001) converges to a local maximum of $\mathbf{WH}$:

$$\begin{aligned} w_{ik} &\leftarrow w_{ik} \frac{\sum_{j=1}^{p} \frac{h_{kj} x_{ij}}{(\mathbf{WH})_{ij}}}{\sum_{j=1}^{p} h_{kj}} \\ h_{kj} &\leftarrow h_{kj} \frac{\sum_{i=1}^{p} \frac{w_{ik} x_{ij}}{(\mathbf{WH})_{ij}}}{\sum_{j=1}^{p} h_{kj}} \end{aligned} \tag{17.76}$$

Unlike VQ and PCA, NMF learns to represent faces with a set of basis images resembling parts of faces. Donoho and Stodden (2004) point out a potentially serious problem with non-negative matrix factorization. Even in situations where $X = \mathbf{WH}$ holds exactly, the decomposition may not be unique.

For example, data points in 17.4 lie in$p = 2$ dimensions, and there is "open space" between the data and the coordinate axes. We can choose the basis vectors h_1 and h_2 anywhere in this open space, and represent each data point exactly with a nonnegative linear combination of these vectors. This non-uniqueness means that the solution found by the above algorithm depends on the starting values, and it would seem to hamper the interpretability of the factorization.

17.8.1 Archetypal Analysis

...

17.9 Independent component analysis and Exploratory Projection Pursuit

Multivariate data are often viewed as multiple indirect measurements arising from an underlying source. Variables that cannot be directly observed are

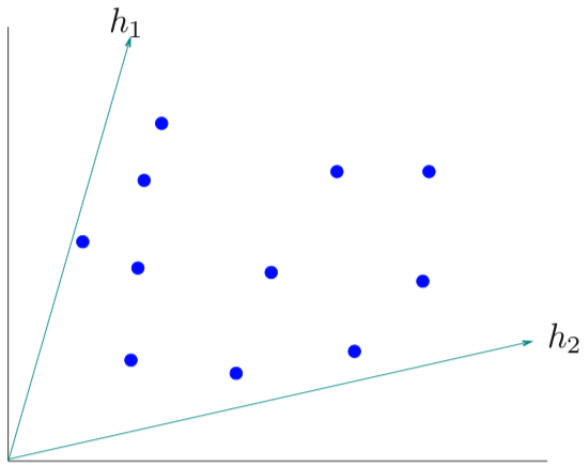


Figure 17.4: Any choice of the basis vectors h1 and h2 in the open space between the coordinate axes and data, gives an exact reconstruction of the data.

called **latent variables**. Example are psychological IQ tests: answers are the multivariate variables (data) and the intelligence, is the latent variable we want to measure and that generated those data. The same for the EEG brain scans to measure neuronal activity: electromagnetic signals are measured by sensors placed at different points. Other examples exist.

17.9.1 Latent variables and Factor analysis

Factor analysis is a statistical technique that aims to find latent sources mostly exploiting Gaussian distributions. Consider the SVD:

$$\begin{aligned} \mathbf{X} &= \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}} \\ \mathbf{S} &= \sqrt{\mathrm{N}}\mathbf{U} \\ \mathbf{A}^{\mathrm{T}} &= \frac{\mathbf{D}\mathbf{V}^{\mathrm{T}}}{\sqrt{\mathrm{N}}} \\ \Rightarrow \mathbf{X} &= \mathbf{S}\mathbf{A}^{\mathrm{T}} \end{aligned} \tag{17.77}$$

and hence each of the columns of $\mathbf{X}$ is a linear combination of the columns of $\mathbf{S}$. Now since $\mathbf{U}$ is orthogonal, and assuming as before that the columns of $\mathbf{X}$ (and hence $\mathbf{U}$) each have mean zero, this implies that the columns of $\mathbf{S}$ have zero mean, are uncorrelated and have unit variance.

However this representation is not unique: given the orthogonal matrix $\mathbf{R}$: $\mathbf{X} = \mathbf{A}\mathbf{R}^{\mathbf{T}}\mathbf{R}\mathbf{S}$. It is therefore impossible to identify any particular latent variables as unique underlying sources.

With $q < p$ the classical factor analysis is $X = AS + \epsilon$. Here $\mathbf{S}$ is a vector of $q < p$ underlying latent variables or factors, A is a $p \times q$ matrix of factor loadings, and the ϵ_j are uncorrelated zero-mean disturbances. The idea is that the latent variables S_ℓ are common sources of variation amongst the X_j, and account for their correlation structure, while the uncorrelated ϵ_j are unique to each X_j and pick up the remaining unaccounted variation. Typically the S_ℓ and the ϵ_j are modeled as Gaussian random variables, and the model is fit by maximum likelihood.

The parameters all reside in the covariance matrix

$$\Sigma = AA^T + D_\epsilon \tag{17.78}$$

Lack of uniqueness still remains.

17.9.2 Independent Component analysis

ICA assumes S_ℓ to be statistically independent rather than uncorrelated. Intuitively, lack of correlation determines the second-degree cross-moments (covariances) of a multivariate distribution, while in general statistical independence determines all of the cross-moments. These extra moment conditions allow us to identify the elements of $\mathbf{A}$ uniquely.

The multivariate Gaussian distribution is determined by its second moments alone. Hence identifiability problems can be avoided if we assume that the S_ℓ are independent and non-Gaussian.

ICA can be used to separate two mixed signals in the classical **cocktail party problem** where different microphones $\mathbf{X}_j$ pick up mixtures of different independent sources S_ℓ. ICA is able to perform **blind source separation** by exploiting independence and non-Gaussianity of the original sources.

Defining the entropy as

$$H(Y) = -\int g(y) \log g(y) dy, \tag{17.79}$$

a result in information Theory states that among all random variables with equal variance, Gaussian variables have the maximum entropy. The **mutual**

information I(Y) between the components of the random vector Y is a natural measure of dependence:

$$I(Y) = \sum_{j=1}^{p} H(Y_j) - H(Y) \tag{17.80}$$

where I(Y) is called **Kullback-Leibler distance** between the density g(y) of Y and its independence version $\prod_{j=1}^{p} g_j(y_j)$ where $g_j(y_j)$ is the marginal density of Y_J.

Now if $\mathbf{X}$ has covariance $\mathbf{I}$, and $Y = A^T\mathbf{X}$ with A orthogonal, then it is easy to show that:

$$I(Y) = \sum_{j=1}^{p} H(Y_j) - H(X) - \log|\det(A)| = \sum_{j=1}^{p} H(Y_j) - H(X) \tag{17.81}$$

Finding an A to minimize I(Y) looks for the orthogonal transformation that leads to the most independence between its components. For convenience, rather than using the entropy $H(Y_j)$, the negentropy measure is sometimes used $J(Y_j)$ defined by:

$$J(Y_j) = H(Z_j) - H(Y_j) \tag{17.82}$$

where Z_j is a Gaussian random variable with the same variance as Y_j . Negentropy is non-negative, and measures the departure of Y_j from Gaussianity. ICA solutions use the approximation:

$$\begin{aligned} J(Y_j) &\sim \left[EG(Y_j) - EG(Z_j)\right] \\ G(u) &= \frac{1}{a}\log\cosh(au) \qquad 1 \le a \le 2 \end{aligned} \tag{17.83}$$

In summary then, ICA applied to multivariate data looks for a sequence of orthogonal projections such that the projected data look as far from Gaussian as possible.

ICA has become an important tool in the study of brain dynamics.

17.9.3 Exploratory Projection Pursuit

It is graphical exploration technique for visualizing high-dimensional data. It assumes projections of high- dimensional data look Gaussian. Interesting

structure, such as clusters or long tails, would be revealed by non-Gaussian projections.

It proposes a number of projection indices for optimization, each focusing on a different departure from Gaussianity. Typically with projection pursuit, the directions a_j are not constrained to be orthogonal.

MDS requires only the dissimilarities d_{ij}.

Multidimensional scaling seeks values $z_1, \cdots, z_N \in \mathcal{R}^k$ to minimize the so-called stress function:

$$S_M(z_1, \cdots, z_N) = \sum_{i \neq j} \big(d_{ij} - \|z_i - z_j\|\big)^2 \tag{17.84}$$

This is known as **least squares** or **Kruskal-Shepherd scaling**. The idea is to find a lower-dimensional representation of the data that preserves the pairwise distances as well as possible. Notice that the approximation is in terms of the distances rather than squared distances. A gradient descent algorithm is used to minimize SM. A variation on least squares scaling is the so-called **Sammon mapping** which minimizes:

$$S_{S_m}(z_1, \cdots, z_N = \sum_{i \neq j} \frac{\big(d_{ij} - \|z_i - z + j\|\big)^2}{d_{ij}} \tag{17.85}$$

Here more emphasis is put on preserving smaller pairwise distances.

In classical scaling, we instead start with similarities s_{ij} : often we use the centered inner product $s_{ij} = \langle x_i - \bar{x}, x_j - \bar{x} \rangle$. The problem then is to minimize

$$S_C(z_1, \cdots, z_N) = \sum_{i,j} \big(s_{ij} - \langle z_i - \bar{z}, z_j - \bar{z} \rangle\big)^2 \tag{17.86}$$

This is attractive because there is an explicit solution in terms of eigenvectors. If we have Euclidean distances instead of inner products we can convert them to centered inner-products. If the similarities are in fact centered inner-products, classical scaling is exactly equivalent to principal components, an inherently linear dimension- reduction technique. Classical scaling is not equivalent to least squares scaling; the loss functions are different, and the mapping can be nonlinear.

Least squares and classical scaling are referred to as **metric scaling methods**, in the sense that the actual dissimilarities or similarities are approximated. **Shephard-Kruskal nonmetric scaling** effectively uses only

ranks:

$$S_{NM}(z_1, \cdots, z_N) = \frac{\sum_{i \neq j} \left[\|z_i - z_j\| - \theta(d_{ij}) \right]^2}{\sum_{i \neq j} \|z_i - z_j\|} \tag{17.87}$$

over z_i and an arbitrary increasing function θ. With θ fixed, we minimize over z_i by gradient descent. With the z_i fixed, the method of isotonic regression is used to find the best monotonic approximation $\theta(d_{ij})$ to $\|z_i - z_j\|$. These steps are iterated until the solutions stabilize.

18 Random forests

Random forests builds a large collection of decorrelated trees and then averages them. It is an alternative to boosting.

18.1 Definition

The idea is to average many noise but unbiased models to reduce the variance. Trees are ideal candidates for bagging, since they can capture complex interaction structures. The **decision rule for regression** is:

Algorithm 6: Random Forests

for $b \leftarrow 1$ **to** B **do**
- Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data;
- Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached;
- (i) Select m variables at random from p variables;
- (ii) Pick the best variable/split-point among m;
- (iii) Split the node into two daughter nodes.

Output the ensemble of trees $\{T_b\}_1^B$;

$$\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x) \tag{18.1}$$

The **decision rule for classification** is:

$$\hat{C}_{rf}^B(x) = \textit{majority vote}\{\hat{C}_b\}_1^B \tag{18.2}$$

Trees are famous to be noisy so they benefit from the averaging. Moreover since each tree is identically distributed, the expectation of an average of B such trees is the same as the expectation of anyone of them. So the bias remains the same and the only improvement is on the variance reduction. In boosting trees are not identically distributed. Summing such B trees each of variance σ^2, the final variance is reduced to $\frac{1}{B}\sigma^2$.

If the variables are simply identically distributed but not necessarily independent with positive pairwise correlation ρ, the variance of the average is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{18.3}$$

As B increases, the second term disappears while the first remains. Correlation limits the benefits of averaging.

The idea of Random forests is to improve variance reduction of bagging by reducing the correlation between trees, without increasing variance too much.

Specifically, when growing a tree on a bootstrapped dataset, before each split, select $m \leq p$ of the input variables at random as candidates for splitting. Typical values are $m = \sqrt{p}$, or even 1.

For classification, the default value for m is $\lfloor\sqrt{p}\rfloor$ and the minimum node size is 1, for regression, the default value for m is $\lfloor p/3\rfloor$ and the minimum node size is five.

After B such trees are grown the predictor is

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x, \Theta_b) \tag{18.4}$$

Θ_b characterizes the b – th random forest tree in terms of split variables, cutpoints at each node, and terminal-node values. Not all estimators can be improved by shaking up the data like this. It seems that highly nonlinear estimators, such as trees, benefit the most.

Random forests do remarkably well, with very little tuning required. A random forest classifier achieves 4.88% misclassification error on the spam test data, which compares well with all other methods, and is not significantly worse than gradient boosting at 4.5%. Bagging achieves 5.4% which is significantly worse than either , so it appears on this example the additional randomization helps.

18.1.1 Overfitting

When the number of variables is large, but the fraction of relevant variables small, random forests are likely to perform poorly with small m. At each split the chance can be small that the relevant variables will be selected. When the number of relevant variables increases, the performance of random forests is surprisingly robust to an increase in the number of noise variables. This robustness is largely due to the relative insensitivity of misclassification cost to the bias and variance of the probability estimates in each tree.

Another claim is that random forests "cannot overfit" the data. It is certainly true that increasing B does not cause the random forest sequence to overfit. The distribution of Θ here is conditional on the training data. However, this limit can overfit the data; the average of fully grown trees can result in too rich a model, and incur unnecessary variance. Using full-grown trees seldom costs much, and results in one less tuning parameter.

18.1.2 Variance analysis

Taking the limit for $B \to \infty$ of the random forest regression estimator:

$$\hat{f}_{rf}(x) = \mathbf{E}_{\Theta|Z} T(x\Theta(Z) \tag{18.5}$$

At a single target point:

$$\mathrm{Var}\hat{f}_{rf}(x) = \rho(x)\sigma^2(x) \tag{18.6}$$

$\rho(x)$ is the sampling correlation between any pair of trees used in averaging:

$$\rho(x) = \mathrm{corr}\left[T(x, \Theta_1(Z)), T(x, \Theta_2(Z))\right] \tag{18.7}$$

Where $T(x, \Theta_1(Z))$ and $T(x, \Theta_2(Z))$ are a randomly drawn pair of random forest trees grown to the randomly sampled Z. σ^2 is the sampling variance of any single randomly drawn tree:

$$\sigma^2(x) = \mathrm{Var}T(x, \Theta(Z)) \tag{18.8}$$

$\rho(x)$ **is not** the average correlation between fitted trees in a given random-forest ensemble. It is the theoretical correlation between a pair of random-forest trees evaluated at x, induced by repeatedly making training sample draws Z from the population, and then drawing a pair of random forest

trees. In statistical jargon, this is the correlation induced by the sampling distribution of Z and Θ.

The variability averaged over in the calculations in18.7 and 18.8 is both conditional on Z for the bootstrap sampling and feature sampling at each split and a result of the sampling variability of Z itself.

$$\textit{Total variance} = \mathrm{Var}_{\mathrm{Z}}\hat{\mathrm{f}}_{\mathrm{rf}}(\mathrm{x}) + \textit{within-}\mathbf{Z}\textit{ variance} \tag{18.9}$$

The second term increases as m decreases. The first term is in fact the sampling variance of the random forest ensemble, which decreases as m decreases. The variance of the individual trees does not change appreciably over much of the range of m, the variance of the ensemble is dramatically lower than this tree variance:

$$\mathrm{Var}_{\Theta,\mathrm{Z}}\mathrm{T}(\mathrm{x},\Theta(\mathrm{Z})) = \mathrm{Var}_{\mathrm{Z}}\mathbf{E}_{\Theta|\mathrm{Z}}\mathrm{T}(\mathrm{x},\Theta(\mathrm{X})) + \mathrm{E}_{\mathrm{Z}}\mathrm{Var}_{\Theta|\mathrm{Z}}\mathrm{T}(\mathrm{x},\Theta(\mathrm{X})) \tag{18.10}$$

18.1.3 Bias analysis

As in bagging, the bias is the same for any individual sampled tree.

$$\mathrm{Bias}(\mathrm{x}) = \mu(\mathrm{x}) - \mathbf{E}_{\mathrm{Z}}\hat{\mathrm{f}}_{\mathrm{rf}}(\mathrm{x}) = \mu(\mathrm{x}) - \mathbf{E}_{\mathrm{Z}}\mathrm{E}_{\Theta|\mathrm{Z}}\mathrm{T}(\mathrm{x},\Theta(\mathrm{Z})) \tag{18.11}$$

This is also typically greater (in absolute terms) than the bias of an unpruned tree grown to Z, since the randomization and reduced sample space impose restrictions.

19 Ensemble Learning

...

20 Undirected Graphical Models

A graph consists of a set of vertices (nodes), along with a set of edges joining some pairs of the vertices. In graphical models, each vertex represents a random variable, and the graph gives a visual way of understanding the joint distribution of the entire set of random variables. They can be useful for either unsupervised or supervised learning. In an undirected graph, the edges have no directional arrows. In undirected graphical models, also known

as Markov random fields or Markov networks, the absence of an edge between two vertices has a special meaning: the corresponding random variables are conditionally independent, given the other variables.

The main challenges in working with graphical models are model selection (choosing the structure of the graph), estimation of the edge parameters from data, and computation of marginal vertex probabilities and expectations, from their joint distribution. The last two tasks are sometimes called learning and inference in the computer science literature.

There is a large and active literature on directed graphical models or Bayesian networks; these are graphical models in which the edges have directional arrows (but no directed cycles). Directed graphical models rep- resent probability distributions that can be factored into products of conditional distributions, and have the potential for causal interpretations.

A graph G consists of a pair (V, E), where V is a set of vertices and E the set of edges (defined by pairs of vertices). Two vertices X and Y are called adjacent if there is a edge joining them; this is denoted by $\mathrm{X} \approx \mathrm{Y}$. A path $\mathrm{X}_1, \mathrm{X}_2, \cdots, \mathrm{X}_\mathrm{n}$ is a set of vertices that are joined, that is $\mathrm{X}_{\mathrm{i}-1} \approx \mathrm{X}_\mathrm{i}$ fori $= 2, \cdots, \mathrm{n}$. A complete graph is a graph with every pair of vertices joined by an edge. A subgraph $\mathrm{U} \in \mathrm{V}$ is a subset of vertices together with their edges. In a Markov graph G, the absence of an edge implies that the corresponding random variables are conditionally independent given the variables at the other vertices. This is expressed with the following notation:

$$\textit{No edge joining } \mathrm{X} \textit{ and } \mathrm{Y} \Longleftrightarrow \mathrm{X} \perp \mathrm{Y}|\mathrm{rest} \tag{20.1}$$

where *rest* refers to the other vertices in the graph (first figure in 20.1). These are known as *pairwise Markov independencies* of G.

If A, B and C are subgraphs, then C is said to separate A and B if every path between A and B intersects a node in C. For example, Y separates X and Z in 20.1(a) and 20.1(d), and Z separates Y and W in 20.1(d). In '20.1(b) Z is not connected to X,Y,W so we say that the two sets are separated by the empty set. In 20.1(c), $\mathrm{C} = \{\mathrm{X}, \mathrm{Z}\}$ separates Y and W.

Separators have the nice property that they break the graph into conditionally independent pieces. Specifically, in a Markov graph G with subgraphs A, B and C,

$$\textit{if } \mathrm{C} \textit{ separates } \mathrm{A} \textit{ and } \mathrm{B} \Rightarrow \mathrm{A} \perp \mathrm{B}|\mathrm{C} \tag{20.2}$$

These are known as the **global Markov properties** of G.

The global Markov property allows us to decompose graphs into smaller more manageable pieces and thus leads to essential simplifications in computation and interpretation. For this purpose we separate the graph into cliques. A **clique** is a complete subgraph a set of vertices that are all adjacent to one another; it is called **maximal**, if it is a clique and no other vertices can be added to it and still yield a clique. The maximal cliques for the graphs of 20.1 are

- $\{X, Y\}$, $\{Y, Z\}$;
- $\{X, Y, W\}$, $\{Z\}$;
- $\{X, Y\}$, $\{Y, Z\}, \{Z, W\}, \{X, W\}$;
- $\{X, Y\}$, $\{Y, Z\}, \{X, W\}$.

A probability density function f over a Markov graph G can be can represented as

$$(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \Psi_C(x_C) \tag{20.3}$$

where C is the set of maximal cliques, and the positive functions $\Psi C()$ are called clique potentials. These are not in general density functions, but rather are affinities that capture the dependence in X_C by scoring certain instances X_C higher than others. The quantity

$$X = \sum_{x \in \mathcal{X}} \prod_{c \in \mathcal{C}} \mathbf{X}_C x_C \tag{20.4}$$

is the normalizing constant, also known as the partition function

....

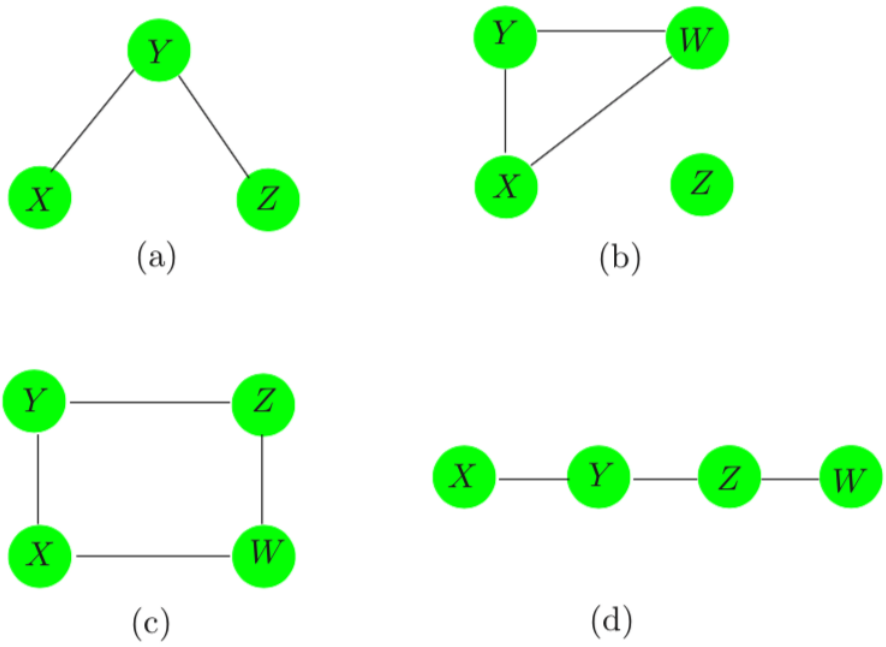


Figure 20.1: Examples of undirected graphical models or Markov networks. Each node or vertex represents a random variable, and the lack of an edge between two nodes indicates conditional independence. For example, in graph (a), X and Z are conditionally independent, given Y. In graph (b), Z is independent of each of X, Y, and W.

Index