Comuter Vision – ENGN6528

Richard Hartley
Australian National University,
Canberra, Australia.

2020

1

Lecture-1: background requirements

Notation

Matrices will be denoted in this font: $\mathtt{A}$, $\mathtt{B}$. Vectors will be denoted as $\mathbf{x}$, $\mathbf{y}$. Written as such, a vector is intended to mean a column vector. The corresponding row vector will be represented by $\mathbf{x}^\mathsf{T}$. Multiplication of vectors by matrices will therefore be written as $\mathbf{y} = \mathtt{A}\mathbf{x}$ or $\mathbf{y}^\mathsf{T} = \mathbf{x}^\mathsf{T}\mathtt{B}$. The superscript $^\mathsf{T}$ means transpose.

Scalars (elements of $\mathbb{R}$, the real numbers, or $\mathbb{C}$, the complex numbers) will be written thus: a, b, z, λ.

1.1 Linear Algebra

First the linear algebra requirements for taking this course will be give. In most cases, the description given will be brief. Extra information is available from Wikipedia pages and forms part of the course.

1.1.1 Orthogonal matrices

A real-valued square matrix $\mathtt{A}$ is called *orthogonal* (not orthonormal) if $\mathtt{A}^\mathsf{T}\mathtt{A} = \mathtt{I}$, where $\mathtt{I}$ is the identity matrix. This implies that $\mathtt{A}^\mathsf{T}$ is the inverse of $\mathtt{A}$:

$$\mathtt{A}^\mathsf{T} = \mathtt{A}^{-1} \ .$$

In addition, if $\mathtt{A}^\mathsf{T}\mathtt{A} = \mathtt{I}$, then multiplying on the left by $\mathtt{A}$ and on the right by $\mathtt{A}^{-1}$ shows that

$$\mathtt{A}\mathtt{A}^\mathsf{T}\mathtt{A}\mathtt{A}^{-1} = \mathtt{A}\mathtt{I}\mathtt{A}^{-1} = \mathtt{I}$$

from which

$$\mathtt{A}\mathtt{A}^\mathsf{T} = \mathtt{I}.$$

An orthogonal matrix can be further characterized by the condition that the rows of $\mathtt{A}$ are of unit length and orthogonal (two vectors $\mathbf{x}$ and $\mathbf{y}$ are orthogonal if $\mathbf{x}^\mathsf{T}\mathbf{y} = 0$). The same is true of the columns.

Fig. 1.1. *Charles Hermite, 1822 - 1901.*

Multiplication of a vector by an orthogonal matrix $\mathtt{U}$ represents rotation of the vector (n-dimensional rotation in $\mathbb{R}^n$) if $\det(\mathtt{U}) = 1$ or rotation plus reflection if $\det(\mathtt{U}) = -1$.

Complex case. The complex case: a matrix $\mathtt{U}$ is called *unitary* if $\mathtt{U}^*\mathtt{U} = \mathtt{I}$, where $\mathtt{U}^*$ represents the complex conjugate transpose of $\mathtt{U}$.

Wikipedia:

`https://en.wikipedia.org/wiki/Orthogonal_matrix}`

1.1.2 Symmetric and Hermitian matrices

A real-valued matrix is symmetric if $\mathtt{A}^\mathsf{T} = \mathtt{A}$. According to this definition, it must be square.

This definition generalizes to complex matrices as follows. A complex matrix is called *Hermitian* (after Charles Hermite, 1822 - 1901) figure 1.1, if it has the property that

$$\mathtt{A} = \mathtt{A}^*$$

where $\mathtt{A}^*$ is the conjugate transpose of $\mathtt{A}$.

A symmetric matrix is therefore by definition Hermitian, and most properties of symmetric matrices can be described more generally for Hermitian matrices.

(i) The eigenvalues of a Hermitian $n \times n$ matrix (or, therefore, a real symmetric matrix) are real. Even for symmetric matrices, this is not immediately obvious.

(ii) The eigenvectors of a Hermitian matrix corresponding to distinct eigenvalues are orthogonal, in the sense that $\mathbf{v}^*\mathbf{v} = 1$ and $\mathbf{v}^*\mathbf{w} = 1$.

(iii) For a real symmetric matrix, eigenvalues and eigenvectors are all real.

(iv) There exist a set of n eigenvectors that are all orthogonal, and hence form an orthogonal basis for $\mathbb{R}^n$.

(v) Hence, if the eigenvectors of a Hermitian matrix are written as the columns of an $n \times n$ matrix, $\mathtt{U}$, then $\mathtt{U}$ is orthogonal.

(vi) The eigenvalue decomposition of a Hermitian matrix $\mathtt{A}$ is given by

$$\mathtt{A} = \mathtt{U}\mathtt{D}\mathtt{U}^*$$

where $\mathtt{D}$ is a diagonal matrix with real entries, and $\mathtt{U}$ is the unitary matrix of eigenvectors.

Wikipedia:

`https://en.wikipedia.org/wiki/Hermitian_matrix`

1.1.3 Positive definite matrices

Normally, we are only interested in positive-definite matrices if they are symmetric (or Hermitian).

This will describe real-valued matrices. In the complex case, we replace transpose (where it occurs) with conjugate-transpose, and the same results apply to complex-valued matrices.

A (square) matrix $\mathtt{A}$ is called positive-definite if $\mathbf{v}^\mathsf{T}\mathtt{A}\mathbf{v} > 0$ whenever $\mathbf{v} \neq 0$. (If $\mathbf{v} = 0$, then obviously this product is zero.) It is called positive-semidefinite if $\mathbf{v}^\mathsf{T}\mathtt{A}\mathbf{v} \geq 0$.

Exercises.

(i) If $\mathbf{v}^*$ is the conjugate transpose of a non-zero vector $\mathbf{v}$, then $\mathbf{v}^*\mathbf{v} > 0$ (and it is real).

(ii) If $\mathtt{A}$ is a Hermitian matrix, then $\mathbf{v}^*\mathtt{A}\mathbf{v}$ is real. Thus, the additional fact about positive-definite matrices is that $\mathbf{v}^*\mathtt{A}\mathbf{v}$ is a *positive* real matrix.

There are other characterizations of positive definite matrices.

(i) The eigenvalues of a positive definite Hermitian (or symmetric) matrix are all positive.

(ii) If $\mathtt{A}$ is any matrix then $\mathtt{A}^\mathsf{T}\mathtt{A}$ is positive-semidefinite. (Same for $\mathtt{A}^*\mathtt{A}$ if $\mathtt{A}$ is complex.) If in addition $\mathtt{A}$ is of dimension $m \times n$ and has rank n, then $\mathtt{A}^\mathsf{T}\mathtt{A}$ (an $n \times n$ matrix) is positive-definite. This applies when $\mathtt{A}$ is a square non-singular matrix (i.e. $\mathtt{A}$ has an inverse).

(iii) If $\mathtt{A}$ and $\mathtt{B}$ (of the same dimension) are positive definite, then $\mathtt{A} + \mathtt{B}$ and $\mathtt{A}^{-1}$ are positive-definite.

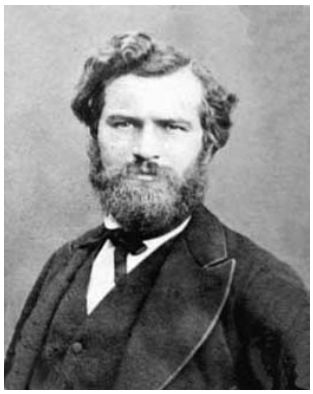

Fig. 1.2. *Eugenio Beltrami (1835 - 1900) and Camille Jordan (1838 - 1922) discovered the Singular Value Decomposition around 1873 – 1874.*

1.1.4 Singular value decomposition

Let $\mathtt{A}$ be a square $n \times n$ real-valued matrix. Then it has a *Singular value decomposition*

$$\mathtt{A} = \mathtt{U}\Sigma\mathtt{V}^{\mathsf{T}}$$

where Σ is a **diagonal** matrix with **positive real elements**, and $\mathtt{U}$ and $\mathtt{V}$ are orthogonal matrices (all $n \times n$ matrices). Let $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, then it is always possible (and usual) to do this decomposition such that $\sigma_i \geq \sigma_{i+1}$ for all i, so, diagaonal entries of Σ are in decreasing order. Note that it is conventional always to write $\mathtt{V}^{\mathsf{T}}$ in this decomposition.

In the case where $\mathtt{A}$ is not square, this is slightly more complicated. Suppose that $\mathtt{A}$ is of dimension $m \times n$ (m rows and n columns) with $m > n$ – hence, a tall skinny matrix. Then

$$\mathtt{A} = \mathtt{U}\Sigma\mathtt{V}^{\mathsf{T}}$$

where $\mathtt{U}$ is an $m \times n$ matrix with orthonormal columns (that is, the columns are orthogonal and of unit length), and Σ are of dimension $n \times n$.

If $\mathtt{A}$ is of dimension $m \times n$ with $m < n$ – a short fat matrix, then $\mathtt{A} = \mathtt{U}\Sigma\mathtt{V}^{\mathsf{T}}$ in which $\mathtt{U}$ and Σ have dimension $m \times m$, and $\mathtt{V}$ is of dimension $n \times m$ with orthogonal columns.

The non-zero (diagonal) entries of Σ are called the singular values of $\mathtt{A}$. The columns of $\mathtt{V}$ are called the *singular vectors* of $\mathtt{A}$.

Here are some properties:

(i) The singular values of $\mathtt{A}$ are always positive (or zero), irrespective of whether the matrix has positive eigenvalues.
(ii) If all the singular values are non-zero, then the matrix has full rank (minimum of m and n), and conversely.

(iii) If $\mathtt{A}$ is a matrix, then the singular values of $\mathtt{A}$ are equal to the square-roots of the eigenvalues of $\mathtt{A}^\mathsf{T}\mathtt{A}$. (Prove this.) Note that $\mathtt{A}^\mathsf{T}\mathtt{A}$ is symmetric positive-semi-definite, so its eigenvalues are real and non-negative (so the square root exists and is real).

Let $\mathtt{A}$ be an $n \times n$ matrix with SDV $\mathtt{A} = \mathtt{U\Sigma V}^\mathsf{T}$. The existence of the SVD has the interpretation that the linear transformation corresponding to $\mathtt{A}$ can be broken down into 3 parts.

(i) Consider multiplication of a vector $\mathbf{x}$ by $\mathtt{A}$, giving $\mathtt{A}\mathbf{x} = \mathtt{U\Sigma V}^\mathsf{T}\mathbf{x}$.
(ii) First, $\mathbf{x}$ is rotated by multiplication by the orthogonal matrix $\mathtt{V}^\mathsf{T}$.
(iii) Then it is scaled (by the singular values) in the n orthogonal axial directions.
(iv) Then it is rotated (multiplication by $\mathtt{U}$) back to a final position.

Another way of thinking of this is that it is scaled (by the singular values) in orthogonal directions, then rotated, or first rotated then scaled. Any linear transform of $\mathbb{R}^n$ to itself can be decomposed this way.

1.1.5 Eigenvalue decomposition

If $\mathtt{A}$ is a square matrix, then an eigenvector of $\mathtt{A}$ is a vector such that $\mathtt{A}\mathbf{x} = \lambda\mathbf{x}$, where $\lambda \in \mathbb{C}$ is a constant, called the corresponding eigenvalue.

(i) The eigenvalue λ may be complex of real, even for real matrices.
(ii) If $\mathtt{A}$ is symmetric (or Hermitian) then the eigenvalues are real, and eigenvectors corresponding to different eigenvalues are orthogonal. (This is not true of arbitrary non-symmetric matrices.)
(iii) If $\mathtt{A}$ is an $n \times n$ matrix **with n different eigenvalues**, and $\mathtt{P}$ is a matrix whose columns are the distinct eigenvectors of $\mathtt{A}$, then

$$\mathtt{A} = \mathtt{P\Lambda P}^{-1}$$

where $\mathtt{\Lambda}$ is a diagonal matrix containing the corresponding eigenvalues of $\mathtt{A}$. The matrix $\mathtt{A}$ is said to be *diagonizable*. (A matrix is diagonizable under some other conditions, but not always).
(iv) An orthogonal or Hermitian matrix is always diagonizable (over $\mathbb{C}$) and the eigenvalues all have length 1.

1.1.6 Norms and inner products.

Norms and inner products are defined on vector spaces.

Vector spaces. A vector space consists of a pair (K, V), where K is a field and V are a set of vectors. For instance the real and complex numbers are examples of fields. Correspondingly one calls them real or complex vector spaces, or vector spaces over $\mathbb{R}$ or $\mathbb{C}$.

Elements of K are called scalars; elements of V are called vectors. In a vector space, there are two things you can do. You can add two vectors, and you can multiply a vector by a scalar. Various axioms apply.

Wikipedia:

`https://en.wikipedia.org/wiki/Field_(mathematics)}`
`https://en.wikipedia.org/wiki/Vector_space}`

The most common example of a real vector space is $\mathbb{R}^n$, consisting of n-tuples of real numbers. These can be added and multiplied by a scalar in obvious ways.

There are other sorts of vector spaces (in fact, they come up all over the place). For instance, the set of functions (or continuous, or differentiable functions on $\mathbb{R}$, or on an interval such as $[0, 1]$), form a vector space, in that you can add them or multiply them by a constant in an obvious way.

Norms. A *norm* is a length (or size) measure on a vector space, denoted by $\|\mathbf{v}\|$, which measures a size of a vector. It must obey certain axioms, the most important being the triangle inequality

$$\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\| \ .$$

A norm allows us to measure distances between two vectors

$$d(\mathbf{v}, \mathbf{w}) = \|\mathbf{v} - \mathbf{w}\|$$

Wikipedia:

`https://en.wikipedia.org/wiki/Normed_vector_space`

Common norms. A common norm is the 2-norm $\|\cdot\|_2$ defined on $\mathbb{R}^n$ by

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2} .$$

where $\mathbf{x} = (x_1, x_n, \ldots, x_n)$ with $x_i \in \mathbb{R}$.

Another common norm is the L_1-norm

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

where $|x_i|$ is the absolute value of x_i.

These are both examples of the L_p norm, defined by

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} .$$

Wikipedia:

`https://en.wikipedia.org/wiki/Lp_space`

Special cases are the L_∞ norm given by

$$\|\mathbf{x}\|_\infty = \max_{i=1}^{n} |x_i| .$$

In addition the L_0 norm $\|\|_0$ (not actually a norm) is defined to equal the number of non-zero elements x_i in $\mathbf{x}$.

A norm on the space of functions defined on the interval $[0, 1]$ is given by

$$\|f\|_2 = \left(\int_0^1 f(x)^2 \, dx \right)^{1/2}$$

Similarly, L_p norms may be defined. For this to be correct, we need to restrict what functions are being considered, so that this integral is finite. One possibility is to consider the set of continuous functions on $[0, 1]$.

Inner products. Sometimes, vector spaces have inner-products defined on them. Suppose this is a vector space over $\mathbb{R}$. An inner product of two vectors is a mapping $\langle \cdot \rangle : V \times V \to \mathbb{R}$, satisfying various simple axioms.

Wikipedia:

`https://en.wikipedia.org/wiki/Inner_product_space`

The most common inner product on the vector space $\mathbb{R}^n$ is

$$\langle \mathbf{v}, \mathbf{w} \rangle = \sum_{i=1}^{n} v_i w_i .$$

This is also called the *dot product* of the two vectors, or if in $\mathbb{R}^3$, it is called the *scalar product* of the two vectors (to distinguish from the *vector product* , otherwise known as the *cross-product* on $\mathbb{R}^3$. (Note that the dot product is defined on all $\mathbb{R}^n$, but the cross-product is defined only on $\mathbb{R}^3$.)

An inner product gives two things:

(i) A norm. Given an inner product on a vector space, one can define a norm by

$$\|\mathbf{x}\| = \langle \mathbf{x}, \mathbf{x} \rangle^{1/2} .$$

It is necessary to verify that this defines a norm.


Fig. 1.3. *Augustin-Louis Cauchy (1789 - 1857) and Karl Hermann Amandus Schwarz (1843 – 1921)*

(ii) An angle θ between two vectors $\mathbf{x}$ and $\mathbf{y}$, defined by

$$\cos(\theta) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \, \|\mathbf{y}\|}$$

For this to make sense, it is necessary to verify that the right-hand-side is between -1 and 1. This is guaranteed by the *Cauchy-Schwartz inequality*.

An inner product gives rise to a norm, but every norm does not come from an inner product in this way. For it to do so, it is necessary that the norm satisfy the *parallelogram law* (see the Wikipedia page).

In $\mathbb{R}^n$, the various L_p norms are defined. Of these, the L_2 norm comes from the standard inner product in this way, whereas the other L_p norms, such as the L_1 norm do not, and in fact, do not derive from any inner product.

Positive-definiteness property. An inner product is said to be *positive definite*. This means that if $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ are N vectors in the space and $\mathtt{A}$ is an $N \times N$ matrix defined by

$$\mathtt{A}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

then $\mathtt{A}$ is positive-semi-definite. (Recall the definition for matrices, given previously). Note, in this definition, the matrix needs only to be *semi-definite*.

Mahalanobis inner product. The standard inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i$ on $\mathbb{R}^n$ is not the only inner product. In fact, if $\mathtt{M}$ is any symmetric positive-definite $n \times n$ matrix, then one may define

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathtt{M}} = \mathbf{x}^{\mathsf{T}} \mathtt{M}^{-1} \mathbf{y} \ .$$

It may be verified that this gives an inner-product on $\mathbb{R}^n$, whereby it is essential that $\mathtt{M}$ be positive-definite.

Fig. 1.4. *David Hilbert and Stefan Banach*

Similarly $\mathtt{M}$ induces a norm on $\mathbb{R}^n$ defined by $\|\mathbf{x}\|_{\mathtt{M}} = \langle \mathbf{x}, \mathbf{x} \rangle_{\mathtt{M}}$, and a distance, called the *Mahalanobis distance* between two vectors defined as

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_{\mathtt{M}} = (\mathbf{x} - \mathbf{y})^{\mathsf{T}} \mathtt{M}^{-1} (\mathbf{x} - \mathbf{y}) \ .$$

Wikipedia:

`https://en.wikipedia.org/wiki/Mahalanobis_distance`

1.1.7 Hilbert and Banach spaces

A Hilbert space is a vector space with an inner product – sometimes called an inner-product space. A Banach space is a vector space with a norm, sometimes called a normed vector space.

Wikipedia:

`https://en.wikipedia.org/wiki/Hilbert_space`
`https://en.wikipedia.org/wiki/Banach-space`

Usually, when we talk of Hilbert of Banach space, we are more interested in infinite-dimensional spaces, but in fact, finite-dimensional spaces, such as $\mathbb{R}^n$, are Hilbert or Banach spaces with the standard inner product or norm.

Actually, a Hilbert space and a Banach space must have a further property called *completeness*, which says that any Cauchy sequence in the space (where Cauchy is defined in terms of the norm) must have a limit in the space. This is always true for finite-dimensional Hilbert spaces. In fact, any finite-dimensional inner-product space is isometric (that is, essentially equivalent to) $\mathbb{R}^n$, and is therefore complete, and a Hilbert space.

The concept of completeness will (probably) not be used in this course, and the Wikipedia pages probably give much more information than is required.

For more information, the book on Hilbert spaces by Elias Stein is very much recommended, and even more, the book on Hilbert spaces by Rod Kennedy (just retired from ANU Engineering) is very much recommended.

1.1.8 Matrix norms

Matrices of a given dimension form a vector space (you can add them and multiply by a scalar). In fact, the set of $m \times n$ matrices is essentially the same, as a vector space, as $\mathbb{R}^{m \times n}$.

(Here, the fact that you can multiply matrices of the right dimension is not relevant – it just gives some extra structure.)

Any norm on $\mathbb{R}^{m \times n}$ corresponds to a norm on the set of (real) matrices of dimension $m \times n$. However, sometimes an extra property of norms defined on square matrices is required, so that it is a so-called *matrix norm*. This is the property that

$$\|\mathtt{AB}\| \leq \|\mathtt{A}\| \, \|\mathtt{B}\| \ .$$

Wikipedia:

`https://en.wikipedia.org/wiki/Matrix_norm`

Many of the standard norms defined on $\mathbb{R}^n$ lead to matrix norms on the set $\mathbb{R}^{m \times n}$ of real matrices of dimension $m \times n$. The most important of these (perhaps) is the *Frobenius norm* of a matrix, given by

$$\|\mathtt{A}\|_F = \Big(\sum_{ij} A_{ij}^2 \Big)^{1/2} \ .$$

which is just the standard L_2 norm of the vector of elements of matrix $\mathtt{A}$.

Related is the *Frobenius inner product* of two matrices, given by

$$\big\langle \mathtt{A}, \mathtt{B} \big\rangle_F = \sum_{ij} A_{ij} \, B_{ij} \ ,$$

which is just the standard inner product of $\mathtt{A}$ and $\mathtt{B}$, considered as vectors.

Another important one is the *nuclear norm* of a matrix, which is defined as the sum of the singular values of the matrix. Thus, if $\mathtt{A} = \mathtt{U\Sigma V}^{\mathsf{T}}$, then $\|\mathtt{A}\|_\rho = \mathrm{trace}(\boldsymbol{\sigma})$ is the nuclear norm.

The *spectral norm* of a matrix $\mathtt{A}$ is the largest singular value of $\mathtt{A}$.

1.1.9 Metrics

As seen before norms on a vector space can be used to define distance between two vectors, by

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| \ .$$


Fig. 1.5. *Ferdinand Georg Frobenius (1849 – 1917) and Prasanta Chandra Mahalonobis (1893 - 1972)*

An example is the usual L_2 distance between the two vectors, when the standard inner product is used, or the Mahalanobis distance (relative to some matrix $\mathtt{M}$) when the Mahalanobis norm $\|\mathbf{x}\|_{\mathtt{M}} = (\mathbf{x}^{\mathsf{T}}\mathtt{M}^{-1}\mathbf{x})^{1/2}$ is used.

However, metrics are an important concept even when vector spaces are not involved.

A *metric space* is a pair (X, d) where X is a set, and d is a distance: $d : X \times X \to \mathbb{R}$. This is intended to measure the distance between two points in X, and hence must satisfy obvious requirements

$$\begin{aligned} d(x,x) &= 0 \\ d(x,y) &\geq 0 \\ d(x,y) &= d(y,x) \\ d(x,z) &\leq d(x,y) + d(y,z)\ . \end{aligned}$$

The last of these is called the *triangle inequality.*

Wikipedia:

`https://en.wikipedia.org/wiki/Metric_space`

The distance d defined by a norm on a vector space by

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$$

is an example of a metric.

Manifolds. An important example of a space where metrics may be defined is the space of manifolds.

A manifold is best envisaged as a smooth surface in some $\mathbb{R}^n$. Examples of manifolds are:

James Chester Manifold (1867-1918), by unknown photographer, 1910s
National Library of Australia, nla.pic-an23358149

Fig. 1.6. *James Chester Manifold (1867 – 1918) – politician and philanthropist.*

Fig. 1.7. *Geodesics on various manifolds.*

- $\mathbb{R}^n$ itself.
- The sphere $S^n = \{\mathbf{x} \in \mathbb{R}^{n+1} \mid \|\mathbf{x}\| = 1\}$.
- Torus (doughnut), cylinder or other surface embedded in some $\mathbb{R}^n$.
- A lower-dimensional "surface" formed by data in $\mathbb{R}^n$.

Definition 1.1. A geodesic is a curve on a manifold that is locally the shortest path between points. The *geodesic* distance between two points on a manifold is the length of the shortest path joining them.

See figure 1.7 for examples.

1.1.10 Approximation theorems

Procrustes' theorem. Given a matrix $\mathtt{A}$, how do we find the closest orthogonal matrix?

Theorem 1.2 (Procrustes theorem). *Given a square matrix* $\mathtt{A}$*, with Singular Value Decomposition (SVD)* $\mathtt{A} = \mathtt{UDV}^\mathsf{T}$*, the closest orthogonal matrix to it under Frobenius norm is given by* $\tilde{\mathtt{A}} = \mathtt{UV}^\mathsf{T}$*.*

Hence, to find the closest orthogonal matrix, simply take the SVD, and drop the diagonal matrix $\mathtt{D}$.

Fig. 1.8. *Procrustes was killed by Theseus, because of his poor hospitality.*

Eckart-Young-Mirsky Theorem

Wikipedia:

`https://en.wikipedia.org/wiki/Low-rank_approximation`

Given an $m \times n$ matrix $\mathtt{A}$, what is the closest matrix (under Frobenius norm distance) with rank $d < \min(m, n)$?

Theorem 1.3 Eckart-Young-Mirsky. *Let* $\mathtt{A}$ *be a matrix with Singular Value Decomposition (SVD)* $\mathtt{A} = \mathtt{U}\Sigma\mathtt{V}^{\mathsf{T}}$*, with* Σ *containing the singular values in descending order. The matrix of rank* d *or less that is closest under Frobenius norm is given by*

$$\hat{\mathtt{A}} = \mathtt{U}\hat{\Sigma}\mathtt{V}^{\mathsf{T}}$$

where $\hat{\Sigma}$ *is obtained from* $\boldsymbol{\sigma}$ *by setting all singular values, after the first* d*, to zero.*

Thus, to do the low-rank approximation, take the SVD, set small singular values (after the d-th largest) to 0, and multiply back out.

1.2 Linear regression

Suppose that we want to fit a line to a set of points. Thus, there are pairs $(x_i, y_i), i = 1, \ldots, n$. For now, suppose that the x_i and y_i are simply real numbers. We want to solve the equation: $mx_i + b = y_i$ (one equation for each i) for a and b. If there are just two points ($n = 2$) then the solution is unique (there is just one line passing through two points). In the case where there are three or more points, then we can not find a solution. We choose to find the "least squares" solution.

The equations can be set up as follows:

$$\begin{bmatrix} x_1 & 1 \\ x_1 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{1.1}$$

With more than two points, this is n equations in two unknowns. We can write this set of equations as $\mathtt{X}\mathbf{a} = \mathbf{y}$, where a is the set of parameters to be found. The "least squares solution" is the one that minimizes

$$L(m, b) = \sum_{i=1}^{n} (mx_i + b - y_i)^2$$

In matrix form, this is:

$$L(\mathbf{a}) = \|\mathtt{X}\mathbf{a} - \mathbf{y}\|^2 \ .$$

This needs to be minimized over the vector $\mathbf{a}$. The solution is

$$\mathtt{X}^\mathsf{T}\mathtt{X}\mathbf{a} = \mathtt{X}^\mathsf{T}\mathbf{y}$$

or

$$\mathbf{a} = (\mathtt{X}^\mathsf{T}\mathtt{X})^{-1}\mathtt{X}^\mathsf{T}\mathbf{y}$$

Another way to see this is:

$$\begin{aligned} \|\mathtt{X}\mathbf{a} - \mathbf{y}\|^2 &= \langle \mathtt{X}\mathbf{a} - \mathbf{y}, \mathtt{X}\mathbf{a} - \mathbf{y} \rangle \\ &= \langle \mathtt{X}\mathbf{a}, \mathtt{X}\mathbf{a} \rangle - 2\langle \mathtt{X}\mathbf{a}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle \\ &= \mathbf{a}^\mathsf{T}\mathtt{X}^\mathsf{T}\mathtt{X}\mathbf{a} - 2\mathbf{a}^\mathsf{T}\mathtt{X}^\mathsf{T}\mathbf{y} + \mathbf{y}^\mathsf{T}\mathbf{y} \end{aligned}$$

Take derivatives with respect to $\mathbf{a}$, and setting to zero gives

$$2\mathtt{X}^\mathsf{T}\mathtt{X}\mathbf{a} - 2\mathtt{X}^\mathsf{T}\mathbf{y} = 0 \ .$$

(It is important to learn how to take derivatives involving matrices and vectors.) This gives the same thing.

Orthogonal regression. Suppose we are working in $\mathbb{R}^n$. In linear regression, we try to minimize the "orthogonal" distance between the plane and the measurements $\mathbf{y}_i, i = 1, \ldots, K$, which are points in $\mathbb{R}^n$. In *orthogonal regression* we wish to compute the orthogonal distance from the points to a dimension $n-1$ hyperplane.

Simplification: suppose we are solving for a line through the origin that minimizes orthogonal distance. The line is determined by its normal direction. Let this be $\mathbf{v}$ (an n vector of length 1). Then, we are trying to minimize the following function:

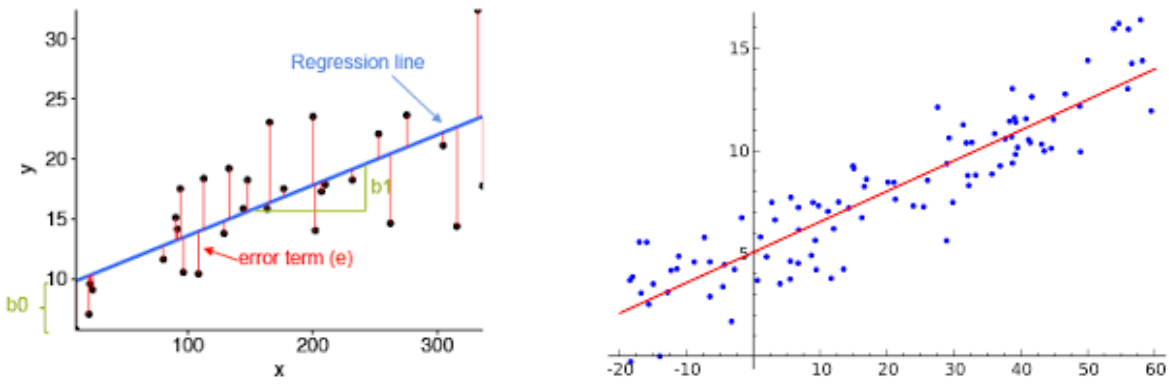

Fig. 1.9. *Linear regression fits a line to points.*

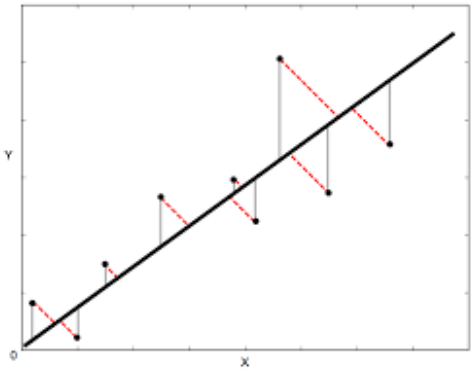

Fig. 1.10. *Orthogonal regression fits a line to points, minimizing the orthogonal distance.*

$$
\begin{aligned}
L(\mathbf{v}) &= \sum_{i=1}^{K} (\mathbf{v}^{\mathsf{T}} \mathbf{y}_i)^2 \\
&= \sum_i \mathbf{v}^{\mathsf{T}} \mathbf{y}_i \mathbf{y}_i^{\mathsf{T}} \mathbf{v} \\
&= \mathbf{v}^{\mathsf{T}} \sum_i (\mathbf{y}_i \mathbf{y}_i^{\mathsf{T}}) \mathbf{v} \\
&= \mathbf{v}^{\mathsf{T}} \mathtt{Y} \mathbf{v}
\end{aligned}
$$

where $\mathtt{Y} = \sum_i (\mathbf{y}_i \mathbf{y}_i^{\mathsf{T}})$ is a symmetric positive definite matrix, called the "scatter matrix". This is to be minimized over all unit vectors $\mathbf{v}$. (Please verify the steps of this derivation.)

The matrix Y can also be written as follows. Suppose that $\mathtt{Y}'$ is the matrix $[\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K]$, whose columns are the points $\mathbf{y}_i$. Then $\mathtt{Y} = \mathtt{Y}'\mathtt{Y}'^{\mathsf{T}}$. It follows that $\mathtt{Y}$ is symmetric and positive-semi-definite.

Let's look at the eigenvalue decomposition of $\mathtt{Y}$. This is $\mathtt{Y} = \mathtt{U}\Lambda\mathtt{U}^\mathsf{T}$ where $\mathtt{U}$ is an orthogonal matrix and the eigenvalues are real and positive (because $\mathtt{Y}$ is positive-semi-definite). Assume that the eigenvalues are in descending order. So, $L(\mathbf{v}) = \mathbf{v}^\mathsf{T}\mathtt{U}\Lambda\mathtt{U}^\mathsf{T}\mathbf{v}$.

Define $\mathbf{v}' = \mathtt{U}^\mathsf{T}\mathbf{v}$. If $\mathbf{v}$ is a unif vector, then so is $\mathbf{v}'$. Then

$$L(\mathbf{v}) = \mathbf{v}'^\mathsf{T}\Lambda\mathbf{v}' = \sum_{i=1}^{n} \lambda_i v_i'^2 \ ,$$

since Λ is a diagonal matrix. The unit vector $\mathbf{v}'$ that minimizes this is $\mathbf{v}' = \mathbf{e}_n$, where $\mathbf{e}_n$ is the n-th basis vector (with 1 in the n-th, or last position).

Finally $\mathbf{v} = \mathtt{U}\mathbf{e}_n$, which is the last column of $\mathtt{U}$, namely the eigenvector of $\mathtt{Y}$ corresponding to the smallest eigenvector.

Check this:

$$\begin{aligned}
\mathtt{Y}\mathbf{v} &= \mathtt{Y}\mathtt{U}\mathbf{e}_n \\
&= \mathtt{U}\Lambda\mathtt{U}^\mathsf{T}\mathtt{U}\mathbf{e}_n \\
&= \mathtt{U}\Lambda\mathbf{e}_n \\
&= \mathtt{U}\lambda_n\mathbf{e}_n \\
&= \lambda_n\mathbf{v} \ .
\end{aligned}$$

To recapitulate: the normal to the best hyperplane fit of the data is the smallest eigenvector (i.e. the eigenvector corresponding to the smallest eigenvalue) of the scatter matrix $\mathtt{Y}$.

Steps of the orthogonal regression algorithm.

(i) Translate the points so that their centroid is at the origin (subtract $\bar{\mathbf{y}} = (1/k)\sum_{i=1}^{k} \mathbf{y}_i$ from all points).
(ii) Form the scatter matrix $\mathtt{Y}$.
(iii) Let $\mathbf{v}$ be the eigenvalue of $\mathtt{Y}$ corresponding to the least eigenvector.
(iv) The best hyperplane fit to the points $\{\mathbf{y}_i\}$ is given by the points $\mathbf{x}$ such that $\mathbf{v}^\mathsf{T}(\mathbf{x} - \bar{\mathbf{y}}) = 0$.

Principal Component Analysis. Principal Component Analysis (PCA) is an extension to orthogonal regression, except that we wish to fit an affine subspace of lower dimension than n to some points $\mathbf{y}_i$ in $\mathbb{R}^n$. See figure 1.11.

Generally, an affine subspace of dimension m in $\mathbb{R}^n$ is a set of points of the form

$$\mathbf{y} = \mathtt{A}\mathbf{x} + \mathbf{b}$$

where $\mathtt{A}$ is a matrix of dimension $n \times m$ and $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{x} \in \mathbb{R}^m$.

The steps of PCA are very much the same as those of orthogonal regression. Given k points $\mathbf{y}_i \in \mathbb{R}^n$,

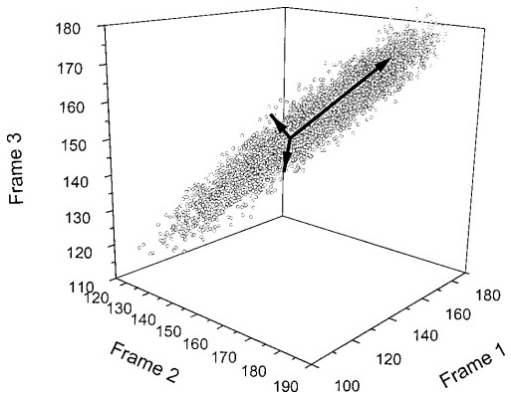

Fig. 1.11. *Principal component analysis seeks to fit a subspace of dimension less than n to a set of points in* $\mathbb{R}^n$.

(i) Translate the points so that their centroid is at the origin (subtract $\bar{\mathbf{y}} = (1/k)\sum_{i=1}^{k} \mathbf{y}_i$ from all points.
(ii) Form the scatter matrix $\mathtt{Y}$.
(iii) Let $\mathtt{A}$ be the matrix consisting of the eigenvectors corresponding to the m **largest** eigenvalues of $\mathtt{Y}$.
(iv) The best hyperplane fit to the points $\{\mathbf{y}_i\}$ is the affine subspace of points of the form $\mathbf{x}' = \mathtt{A}\mathbf{x} + \bar{\mathbf{y}}$, for $\mathbf{x} \in \mathbb{R}^m$.
(v) This can also be written as the points $\mathbf{x}'$ satisfying the equation

$$(\mathtt{A}^{\perp})^{\mathsf{T}}(\mathbf{x}' - \bar{\mathbf{y}}) = \mathbf{0}$$

where $\mathtt{A}^{\perp}$ is the matrix consisting of the columns of $\mathtt{Y}$ corresponding to the $n - m$ **smallest** eigenvalues.

For brevity, we can say " smallest eigenvectors" instead of "eigenvectors corresponding to the smallest eigenvalues".

Note, what happens in higher dimensions. The "normal" to a subspace is no longer a single vector, but a complete linear subspace itself, of dimension $n - m$. In this case, it is spanned by the basis given by the $n - m$ smallest eigenvalues of $\mathtt{Y}$. These are the columns of the matrix $\mathtt{A}^{\perp}$. The subspace itself is spanned by the m largest (principal) eigenvectors, namely the columns of $\mathtt{A}$, offset by $\bar{\mathbf{y}}$.

2

Lecture-2: Statistics and Model Fitting

2.1 Probabilities

Let Ω be a finite set with n elements. A probability distribution on the set Ω is an assignment of a real number $p(x_i)$ to each $x_i \in \Omega$, with the properties

$$p(x) \geq 0 \quad \text{for all} \quad x \in \Omega$$
$$\sum_{x \in \Omega} p(x) = 1$$

The idea of probability is extended to finite subsets of Ω as follows. For a subset $S \subset \Omega$,

$$p(S) = \sum_{x \in S} p(x) \ .$$

As a particular case, it follows that $P(\Omega) = 1$.

Probabilities on infinite sets. The idea of probability can be extended to probabilities defined on infinite sets Ω. However, in this case, we cannot approach it by assigning probabilities to single elements, since the probability of most single element $x \in \Omega$ will be zero, otherwise, $\sum_{x \in \Omega} p(x) = 1$ cannot hold. Therefore, a probability on a set Ω is defined by assigning a probability $p(S)$ to subsets S of Ω, subject to certain conditions.

In this case, let $\mathcal{F} \subset 2^{\Omega}$ be a collection of subsets of Ω (technically called a σ-algebra)[1] . (The notation 2^{Ω} is defined to mean the set of all subsets of Ω, or alternatively, the set of all mappings from Ω to the set $\mathcal{B} = \{0, 1\}$.) A probability function $P : \mathcal{F} \to \mathbb{R}^+$ (non-negative real numbers) assigns a probability to each subset, with the following properties.

[1] Technical point: it turns out that it is not possible to define a probability $P(S)$ for all sets $S \in 2^{\Omega}$, if certain obvious properties are desired. For this reason, probabilities are defined only on *some* subsets, the elements of $\mathcal{F}$.

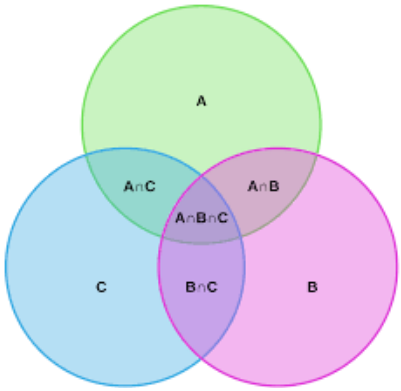

Fig. 2.1. *Representation of probabilities by Venn diagrams.*

(i) Countable additivity: if $\{S_i\}$ is a countable collection of **disjoint** sets $S_i \in \mathcal{F}$, then

$$P(\cup S_i) = \sum_i P(S_i)$$

(ii) Positivity: $P(S) \geq 0$ for all S.
(iii) Total probability is 1, that is $P(\Omega) = 1$.

It is convenient to represent probabilities (and occasionally to argue about them) using a Venn diagram. See figure 2.1.

Probability distributions. For probabilities defined on real numbers $\mathbb{R}$, or $\mathbb{R}^n$, one can define probability distributions.[1] An element consisting of a single point $x \in \mathbb{R}$ can be associated with a set, $\{x\}$, called a singleton. This should have a probability. However, because of countable additivity, the probability of a singleton is often (usually) zero.

Instead, we look at the probability of a finite interval I, containing a point x, denoted by $P(I)$. As the interval gets smaller, the probability of the set decreases. We can define

$$p(x) = \lim_{|I| \to 0} P(I)$$

where the I are intervals containing x, and $|I|$ means the length of the interval.

Then $p(x)$ is a continuous function (as long as singletons have probability zero), called a *probability distribution*. We use p to mean a probability distribution, and P to mean a probability function $P : \mathcal{F} \to \mathbb{R}$.

A probability distribution on the real line $\mathbb{R}$ is a function $p(x)$ satisfying

(i) Positivity: $p(x) \geq 0$ for all x.

[1] It is possible to define probability distributions on other spaces as well, such as Riemannian manifolds, such as S^n and others.

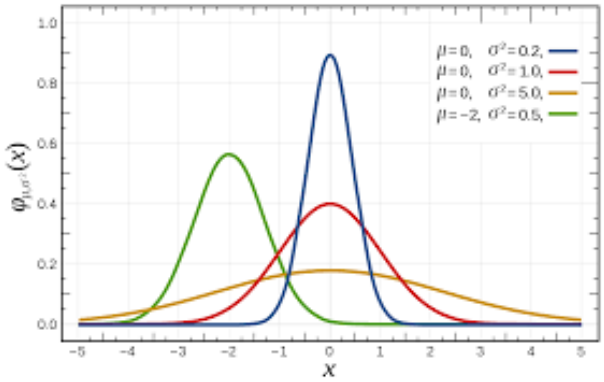

Fig. 2.2. *Gaussian (normal) distributions with differing mean and variance.*

(ii) Unit integrability: $\int_{-\infty}^{\infty} p(x)\,dx = 1$.

Examples of continuous distributions are

(i) Uniform distribution on a given finite interval. Let $[a, b]$ be an interval. The *uniform distribution* is given by $p(x) = \alpha$ (a constant) for all $x \in [a, b]$, and $p(x) = 0$ otherwise. Because of the unit integrability, the constant $\alpha = 1/(b - a)$.

(ii) Gaussian (or *normal*) distribution:

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

Here μ is known as (in fact is) the mean of the distribution, and σ^2 is the variance. See figure 2.2.

(iii) Others.

Note. Probabilities must always lie between 0 and 1. On the other hand, values of probability distributions $p(x)$ do not always lie between 0 and 1 (though they must always be non-negative). The whole distribution must integrate (or sum) to 1, but $p(x)$ is defined as a ratio of probability on an interval to length of interval, which need not be bounded (by 1, or at all).

2.1.1 Random variables

There is a mathematical definition of random variable, but we ignore it for now. A random variable X is conveniently thought of as a variable taking values in a given set (often the reals, $\mathbb{R}$, but not necessarily). The difference between this and an ordinary variable is that a random variable takes any value in the set of values, $\mathbb{R}$, taking any value with a certain probability. (Or more properly, having values in some set, with a given probability).

We write: $X : \Omega \to E$, where X takes values in E. Often $E = \mathbb{R}$.

We can write $P(X = x)$ to mean *the probability that* random variable X takes value $x \in \mathbb{R}$. One way of thinking of this is that a random variable and a probability distribution are the same thing, but this is a little circular.

It is also possible to write things like: $P(X \in T)$, to mean the probability that the value random variable X lies in a set T.

(A more formal definition of this probability associated with a random variable X is as follows. The "probability space" Ω has an associated probability. In particular, any set $S \subset \Omega$ has a probability function $P : \mathcal{F} \to \mathbb{R}^+$, where $\mathcal{F}$ is the σ-algebra of subsets of Ω. Given the random variable $X : \Omega \to E$, and a subset $T \subset E$, the probability $P(X \in T)$ is defined to equal $P(X^{-1}(T))$, where $X^{-1}(T) = S = \{s \in \Omega \mid X(s) \in T\}$.)

Also, if X and Y are two random variables (with values in $\mathbb{R}$), then XY is also a random variable got by multiplying the values of X and Y together. (Note that this works for random variables $X : \Omega \to E$ only when multiplication is defined on E.) Likewise, if $f : \mathbb{R} \to \mathbb{R}$ is a function, then $f(X)$ is also a random variable.

As with probabilities, a random variable gives rise to a probability distribution. If X is a random variable with values in $\mathbb{R}$, then there is a probability distribution $p_X : \Omega \to \mathbb{R}$ by

$$p_X(x) = \lim_{|I| \to 0} \frac{P(X \in I)}{|I|} \ .$$

2.1.2 Moments

Expected value. The expected value of a random variable X with distribution p (with values in $\mathbb{R}$) is defined as

$$E[X] = \int_{-\infty}^{\infty} p(x)\, x\, dx$$

This is otherwise known as the mean of the random variable, or the *first moment*. Note that the integration takes place in the space where the random variable takes its values, that is, in $\mathbb{R}$.

Variance. The variance of a random variable X, denoted by $\sigma^2(X)$, with mean $\mu = E[X]$, is given by

$$\begin{aligned} \sigma^2(X) &= \int p(x)(x-\mu)^2\, dx \\ &= \int p(x)x^2\, dx - \mu^2 \\ &= E[(X-\mu)^2] \end{aligned}$$

If you are not familiar with the above formulas for variance, then you should try to prove that they are all equal.

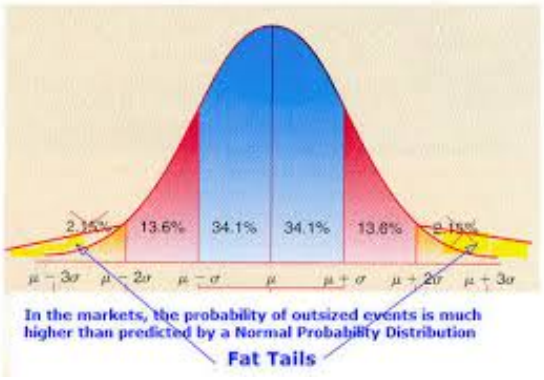

Fig. 2.3. *A probability distribution with a bad case of kurtosis. The condition of kurtosis is characterized by having fat tails. It is known to be extremely contagious among Australian marsupials. (Figure from https://kenjeyaretnam.com/market/fat-tails-and-unknown-unknowns/)*

The variance is sometimes called the *second moment*. The *standard deviation* is σ, the square-root of the variance.

There are also higher-order moments, defined in the obvious way. The *third moment* is called the *skew*, and the fourth moment is the *kurtosis* (Wikipedia: kurtosis, moment). See figure 2.3.

Finite formulas For distributions on a finite set with n elements, the formulas above are as follows:

$$\begin{aligned} E[X] &= \sum_{i=1}^{n} x_i p(x_i) \\ \sigma^2(X) &= \sum_{i=1}^{n} p(x_i)(x_i - \mu)^2 \\ &= \sum_{i=1}^{n} p(x_i) x_i^2 - \mu^2 \ . \end{aligned}$$

Moments of random variables with values in $\mathbb{R}^n$. If X takes values in $\mathbb{R}^n$, then the formulas need to be modified as follows. We write $\mathbf{x}$ (in bold font) to remind ourselves that it is a vector (in $\mathbb{R}^n$).

$$E[X] = \int_{\mathbb{R}^n} p(\mathbf{x})\, \mathbf{x}\, d\mathbf{x} \ .$$

Here, $p(\mathbf{x})$ is in $\mathbb{R}$, but $\mathbf{x} \in \mathbb{R}^n$. The result is a vector $\mu \in \mathbb{R}^n$.

As for variance, we cannot simply multiply two elements of $\mathbb{R}^n$ together. (What

would multipication of two vectors mean?) The formula for the variance is

$$\begin{aligned}\sigma^2(X) &= \int p(\mathbf{x})(\mathbf{x}-\mu)\,(\mathbf{x}-\mu)^\mathsf{T}\,d\mathbf{x} \\ &= \int p(\mathbf{x})\mathbf{x}\mathbf{x}^\mathsf{T}\,d\mathbf{x} - \mu\mu^\mathsf{T}\end{aligned}$$

This directly generalizes the definition for the case of real-valued random variables.

Note that this defines a matrix $\sigma^2(X)$, which is symmetric and positive-definite. It is called the *covariance matrix* of the (vector-valued) random variable.

The eigenvectors of this matrix are therefore orthogonal, and are called the *principal axes* of the distribution.

For vector-valued random variables defined on finite sets Ω, the formulas become

$$\begin{aligned}E[X] &= \sum_{i=1}^{n} p(\mathbf{x}_i)\mathbf{x}_i \\ \sigma^2(X) &= \sum_{i=1}^{n} p(x_i)(\mathbf{x}_i-\mu)(\mathbf{x}_i-\mu)^\mathsf{T} \\ &= \sum_{i=1}^{n} p(\mathbf{x}_i)\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - \mu\mu^\mathsf{T}\ .\end{aligned} \tag{2.1}$$

Empirical mean and variance In some cases, we do not know the probability $p(x)$, but instead, we have a set of samples from the distribution. Suppose that $\mathbf{x}_i$ are a set of samples taken from a distribution $p(\mathbf{x})$. Then, the empirical mean and covariance can be computed empirically[1] as:

$$\begin{aligned}E[X] &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \\ \sigma^2(X) &= \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i-\mu)(\mathbf{x}_i-\mu)^\mathsf{T} \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^\mathsf{T} - \mu\mu^\mathsf{T}\ .\end{aligned} \tag{2.2}$$

Comparing this with (2.1) we we see that the probabilities $p(\mathbf{x}_i)$ are missing in (2.2). This is because the different samples appear in this sum with frequencies proportional to their probabilities. The matrix $\sigma^2(X)$ is the empirical covariance matrix, or the "scatter matrix", which we have encountered previously in discussion of PCA.

[1] These are the empirical mean and variance of the measured data $\mathbf{x}_i$. It is not claimed that these are the best estimates of unknown mean and variance of the actual probability distribution. Nevertheless, as the number of trials increases to infinity, the empirical mean and variance converge to the true mean and variance of the distribution

2.1.3 Independent and uncorrelated distributions

The *covariance* of two random variable is given by (See Wikipedia: Correlation and dependence):

$$\begin{aligned}\mathrm{cov}(X,Y) &= E[(X-\mu_X)(Y-\mu_Y)] \\ &= E[XY] - \mu_X\mu_Y\end{aligned}$$

(prove these are the same).

The *correlation*, or *normalized covariance* is

$$\mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X\sigma_Y} \ . \tag{2.3}$$

Exercise Prove that $\mathrm{corr}(X,Y) = \mathrm{corr}(aX,bY)$ for any non-zero constants a and b.

Two variables X and Y are *uncorrelated* if $\mathrm{corr}(X,Y) = 0$.

2.1.4 Joint probability distributions

(See Wikipedia:"Joint probability distribution").

Let $X : \Omega \to E_X$ and $Y : \Omega \to E_Y$ be two random variables defined on the same space, Ω. Together, they make a function $(X,Y) : \Omega \to E_X \times E_Y$, If $X = x$ and $Y = y$, then $(X,Y) = (x,y)$. For instance, height h and weight w are two random variables defined on the set of all people, with values in $\mathbb{R}$. The pairs (h,w) form a pair of joint random variables.

The joint variable (X,Y) comes with an associated probability distribution $P(X,Y)$, where $P(X=x, Y=y)$ is the probability that both $X = x$ and $Y = y$ occur together.

Independent variables. Two random variables X and Y are *independent* if $P(X,Y) = P(X)P(Y)$. This is a stronger condition than being uncorrelated. Two independent variables are uncorrelated, but not vice-versa. See Wikipedia:"Correlation and dependence") for a proof.

It is a common mistake to think that uncorrelated and independent mean the same thing – don't make this mistake.

`See: Wikpedia: ‘‘Normally_distributed_and_uncorrelated_does_not_imply_independent’’`

Conditional probabilities. Given a joint probability distribution defined by random variables (X,Y), one can define conditional probabilities $P(X \mid Y)$, read "probability of X given Y". This is itself a random variable, defined on the same space as X, defined by

$$P(X \mid Y) = \frac{P(X,Y)}{P(X)\,P(Y)} \ .$$

This formula may need some explanation. When a formula is written in this way, involving random variables, then things like $P(X)$ and $P(X, Y)$ do not represent numbers, but probability distributions. What is meant is that such a formula should hold for all possible values of the variables, X and Y. This means:

$$P(X = x \mid Y = y) = \frac{P(X = x, Y = y)}{P(X = x)P(Y = y)} \ .$$

This can be read: The probability that $X = x$ given that $Y = y$ is (defined as) the probability that $X = x$ and $Y = y$ divided by the probability that $X = x$ times probability that $Y = y$. Thus, $\mid$ is read as "given" and the comma "," is read as "and".

Another way this can be written is:

$$P(X, Y) = P(X \mid Y)P(Y)$$

Exercise. Two random variables X and Y were defined as *independent* if $P(X, Y) = P(X)\,P(Y)$. Show that this is the same as saying: $P(X \mid Y) = P(X)$, in other words, the probability of X given Y is the same as probability of X – in other words, knowing the value of Y gives no information about the probability of X.

Also, show that if X and Y are independent, then $E[XY] = E[X]E[Y]$.

Bayes' rule. Clearly $P(X, Y) = P(Y, X)$. Using this, we see that

$$P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X) \ . \tag{2.4}$$

from which we get "Bayes' rule ":

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)} \tag{2.5}$$

Although Bayes' rule has a fancy name, it is nothing more that a simple consequence of the definition of conditional probability.

Marginal probabilities. If a joint probability $P(X, Y)$ is given, we sometimes talk of $P(X)$, the probability associated with the single random variable X. This is known as the *marginal probability*. For instance $P(X)$ and $P(Y)$ appear in Bayes' rule.

More exactly, we could write $P(X = x)$, how frequently does the event $X = x$ occur. Suppose for now that both X and Y are random variables taking their values in finite sets, for instance $\mathcal{L}_X$ for X and $\mathcal{L}_Y$ for Y.

For instance, suppose that there are objects in an urn, that come in two colours, black and white, denote by X, and two shapes square and round, denoted by Y. There is a mixture of types, and the probability of drawing an object with attributes (x, y) from the urn is $p(x, y)$. Now, what is the probability of drawing a black object from the urn. There are round black objects and square black objects.

Write $P(X = \text{black}, Y = \text{square}) = P(b, s)$ and $P(X = \text{black}, Y = \text{round}) = P(b, r)$. It follows that the probability of getting a black object is given by $P(b) = P(b, s) + P(b, r)$.

In general, one can write

$$\begin{aligned} P(X = x) &= \sum_{y \in \mathcal{L}_Y} P(X = x, Y = y) \\ &= \sum_{y \in \mathcal{L}_Y} P(X = x \mid Y = y)\, P(Y = y) \end{aligned} \tag{2.6}$$

or, for brevity

$$P(X) = \sum_{y \in \mathcal{L}_Y} P(X, y) = \sum_{y \in \mathcal{L}_Y} P(X \mid y) P(y) \ . \tag{2.7}$$

Both $P(X)$ and $P(Y)$ appearing here are marginal probabilities. We could remind ourselves of this by writing

$$P_M(x) = \sum_{y \in \mathcal{L}_Y} P_J(x, y) = \sum_{y \in \mathcal{L}_Y} P_C(x \mid y) P_M(y) \ . \tag{2.8}$$

where J indicates that it is a joint probability, C indicates a conditional probability, and M denotes a marginal probability. However, this is not usuall done, and one must be alert to interpret this properly.

In the case of probability distributions defined on an infinite set, such as $\mathbb{R}$, the correct formula is

$$p(x) = \int_y p(x, y)\, dy \ .$$

The two basic formula of joint probabilities. If you are going to memorize two formulas to do with joint probabilities, they would be the following. Given a joint probability distribution $P_J(X, Y)$:

(i) Marginal probability:

$$P_M(x) = \sum_{y \in \mathcal{L}_Y} P_J(x, y) \ .$$

(ii) Conditional probability:

$$P_C(x \mid y) = \frac{P_J(x, y)}{P_M(y)} = \frac{P_J(x, y)}{\sum_{x' \in \mathcal{L}_X} P_J(x', y)} \ .$$

(iii) In the continuous case:

$$p_C(x \mid y) = \frac{p_J(x, y)}{p_M(y)} = \frac{p_J(x, y)}{\int p_J(x', y)\, dx'} \ .$$

2.1.5 Likelihood

Likelihood is usually described in terms of *models* . For instance, suppose that a person has a model for rainfall as follows.

Let temperature (on a given day) be T (Kelvin), humidity be H (percent) and rainfall R (in millimetres). A model may be that

$$R = (3T + H^2)/HT \ .$$

(Don't worry if this makes no sense.)[1] This model will scarcely ever be exactly right. Instead, a better model may be

$$R = (\alpha T + \beta H^2)/HT + \eta \ ,$$

where η represents randomness or variability, and is a random variable with some distribution. For instance, a common assumption may be that η is a Gaussian random variable with mean 0 and variance σ^2, meaning that the model is essentially correct, but subject to noise, or random perturbations.

Suppose that σ is known, as are α and β. These are the parameters, and will be denoted by θ. Represent the rainfall by y (meaning the same as R here). Let the measurements T and H be denoted by x. Further, let this function be denoted by

$$y = f(x, \theta) + \eta(\sigma) \ .$$

Now, suppose on a given day, T, H and R are all measured. The question is, if we assume that the model is correct, then what is the probability of the outcome (y), given the measurements x and the parameters θ. This can be calculated as follows:

$$P(y, x \mid \theta) = p_\eta(y - f(x, \theta)) \ .$$

For the case where the distribution p_η is Gaussian, this is

$$P(y, x \mid \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-(y - f(x, \theta))/2\sigma^2) \ .$$

Normally, the model will be evaluated given several measurement, denoted $\mathbf{x}$ and $\mathbf{y}$. Each day's measurements is called a *trial*. A usual assumption is that all the trials are independent (given the measurements), and the random model is the same for each trial. This is called "independent and identically distributed" noise model, or iid noise.

In this case, if there are n trials, then the probability of the set of outcomes $\mathbf{y}$ given measurements $\mathbf{x}$ is equal to

$$P(\mathbf{y}, \mathbf{x} \mid \theta) = \prod_{i=1}^{n} p_\eta(y_i - f(x_i, \theta)) \ .$$

[1] A saying in statistics is that all models are wrong.

In the case of Gaussian p_η, this is

$$P(\mathbf{y}, \mathbf{x} \mid \theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp(-(y_i - f(x_i, \theta))^2 / 2\sigma^2) \ .$$

Log probability. This can be simplified by taking logarithms, changing products to sums, resulting in

$$\log P(\mathbf{y}, \mathbf{x} \mid \theta) = \sum_{i=1}^{n} \left(-\log(\sqrt{2\pi}) - \log(\sigma) - \frac{(y_i - f(x_i, \theta))^2}{2\sigma^2} \right) \ . \tag{2.9}$$

If we write

$$r^2 = \sum_{i=1}^{n} (y_i - f(x_i, \theta))^2 \ ,$$

which is the sum-of-squared *residuals*, then finally

$$\log P(\mathbf{y}, \mathbf{x} \mid \theta) = C - n \log(\sigma) - \frac{r^2}{2\sigma^2} \ . \tag{2.10}$$

What we have calculated here is the probability of the measurement $(\mathbf{y}, \mathbf{x})$, given the model with parameters θ. This can be written as $P(\mathbf{d} \mid \theta)$, where $\mathbf{d}$ stands for the *data*. Then $\log P(\mathbf{d} \mid \theta)$ is known as the *log-probability* of the data given the model.

The probability of the data given a model gives an indication of what model (or parametrization of the model) should be chosen. Models or parametrizations that give high probability are more likely. Hence, it is customary to define

$$\mathcal{L}(\theta \mid \mathbf{d}) = P(\mathbf{d} \mid \theta)$$

and call this the *likelihood* of the model, given the data. Similarly, $\log \mathcal{L}(\theta \mid \mathbf{d}) = \log P(\mathbf{d} \mid \theta)$ is called the *log-likelihood* of the model. Typically, one will want to choose a parametrization that *maximizes* the log-likelihood, or equivalently, one that *minimizes* the *negative log-likelihood* of the model. This is known as the *Maximum Likelihood* estimate (of the model), often abbreviated to MLE.

It is important to note that $\mathcal{L}(\theta \mid \mathbf{d})$ is **not** the probability of the model θ given the data $\mathbf{d}$. In fact, by Bayes' Rule,

$$P(\theta \mid \mathbf{d}) = \frac{P(\mathbf{d} \mid \theta) P(\theta)}{P(\mathbf{d})}$$

In minimizing this quantity, the value of $P(\mathbf{d})$ can be ignored, since it is constant (with respect to θ) and has been observed to be true. This gives that

$$P(\theta \mid \mathbf{d}) \equiv \mathcal{L}(\theta \mid \mathbf{d})\, P(\theta)$$

(meaning "proportional"). Thus the probability and likelihood differ by the probability of the model, (called the *a priori* or *prior* probability of the model). Often

there is no reasonable way to assign this prior probability (without reference to any data), and hence $P(\theta \mid \mathbf{d})$ cannot be computed. If a prior probability $P(\theta)$ is available, then $P(\theta \mid \mathbf{d})$ can be computed and used to estimate the most probable model, called the *Maximum a-priori probability*, or MAP estimate.

Minimizing over the variance. Consider the case where the noise η is zero-mean Gaussian with unknown variance σ. In finding the ML model estimate, we can separate the parameters of the function f and the parameter σ of the Gaussian.

In this case, the log-likelihood is given by (2.10). To minimize with respect to σ, we take the derivative:

$$\frac{\partial \log \mathcal{L}(\theta \mid \mathbf{d})}{\partial \sigma} = -\frac{n}{\sigma} + \frac{r^2}{\sigma^3}$$

Setting this to zero gives $\sigma = \sqrt{r^2/n}$, which is the RMS (root-mean-squared) residual. Finally, substituting this into the equation (2.10) for log-likelihood gives

$$-\log \mathcal{L}(\theta \mid \mathbf{d}) = -C + \frac{n}{2}\log(r^2/n) + \frac{n}{2} \tag{2.11}$$

for the optimal value of σ. Since n is fixed, minimizing the negative log-likelihood is equivalent to minimizing $\log(r^2)$, which is equivalent to minimizing r^2 (since log is an increasing function).

Summary: to find the most likely model in terms of the parameters θ and σ, the steps are:

(i) Minimize the sum-of-square residuals

$$r^2 = \sum_{i=1}^{n} (y_i - f(x_i, \theta))^2$$

with respect to the parameters θ.

(ii) The value of σ is equal to the RMS residual $\sqrt{r^2/n}$.

2.2 Fitting models to data

We now consider the case where there is a choice between different models to fit to data, and illustrate it on the situation where polynomials of different degrees are fitted to some noisy points. As will be seen, as the number degree of the polynomial increases, the better the fit to the data points will be. In fact, a degree $n-1$ polynomial (which has n independent coefficients) will fit n data points (x_i, y_i). exactly, irrespective of the degree of the polynomial used to generate the points.

Thus, we consider a function $y_i = f(x_i)$ where f is polynomial of a given degree d. For each x_i, a ground-truth value $\bar{y}_i$ is generated. Then Gaussian noise with a certain variance is added to produce noisy values, y_i. The data points (x_i, y_i) are then used to find the best fitting polynomial.

Training residual. When the polynomial is fitted to the points (x_i, y_i) it will not fit exactly, and there will be a certain residual $r_i = |f(x_i) - y_i|$. A suitable measure of the goodness of the fit is the sum of squares residual

$$\sum_{i=1}^{n}(f(x_i) - y_i)^2 = \sum_{i=1}^{n} r_i^2$$

which we have denoted by r^2. The RMS error is given by $\delta_{\text{RMS}} = \sqrt{r^2/n}$. This measures the fit to the measured (noisy) data, also called the training data.

Ground truth residual At the same time it is interesting to ask (and this is presumably the main goal) how well the curve fits the ground-truth data $(x_i, \hat{y}_i)$. This is given by

$$\sum_{i=1}^{n}(f(x_i) - \bar{y}_i))^2 = \sum_{i=1}^{n} \bar{r}_i^2 = \bar{r}^2$$

where $\bar{r}_i$ represents the residual with repect to ground truth. The RMS ground truth residual is represented by $\sqrt{\bar{r}^2/n}$. Normally, the ground-truth is not known, so the ground-truth RMS cannot be measured exactly. Later on it will be seen that it can be estimated, however.

Test residual. To test the generalizability of the found model, a good way is to see how well it fits some test data, which is generated for this purpose. To do this synthetically, one generates further pairs $(\hat{x}_i, \hat{y}_i)$ (without added noise). Then the test-data sum of squares error is given by

$$\sum_{i=1}^{m}(f(\hat{x}_i) - \hat{y}_i))^2 = \sum_{i=1}^{m} \hat{r}_i^2 = \hat{r}^2$$

where m is the number of test samples. The test sum-of-squared residual is denoted by $\hat{r}^2$, and the RMS residual is given by $\sqrt{\hat{r}^2/m}$.

Various examples of trying to fit polynomial curves to data are given in figure 2.4, figure 2.6 and figure 2.8. The values of the various RMS residuals are given in figure 2.5, showing how as the degree of the polynomial increases beyond the ground-truth degree, training error continues to decrease, whereas the ground-truth and test errors increase. When the degree of the polynomial (9) reaches $n - 1$, the training fit is perfect, but test error becomes very large.

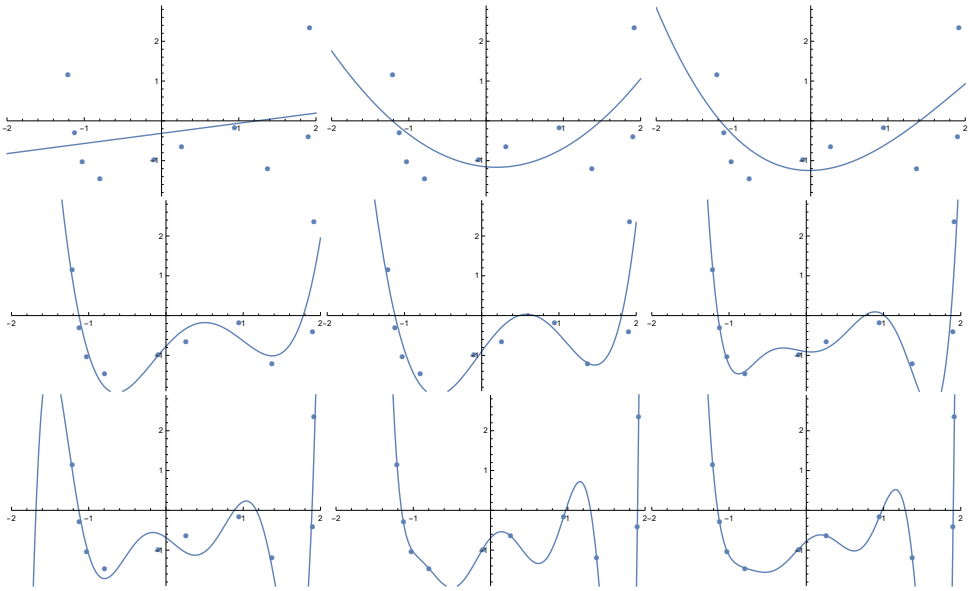

Fig. 2.4. *Fitting polynomials to points. Ten points are generated by a degree* 2 *polynomial.*

degree	truth	train	test	aic	aicc
0	0.471	1.112	0.704	4.128	4.128
1	0.515	1.07	0.891	5.361	5.361
2	0.258	0.89	0.201	3.677	4.677
3	0.272	0.886	0.39	5.582	6.582
4	0.683	0.627	5.603	0.669	2.669
5	0.692	0.617	3.568	2.333	6.333
6	0.77	0.516	23.618	0.754	7.754
7	0.8	0.467	7.976	0.787	17.787
8	0.926	0.028	110.817	-53.62	0
9	0.927	0	44.421	-607.025	0

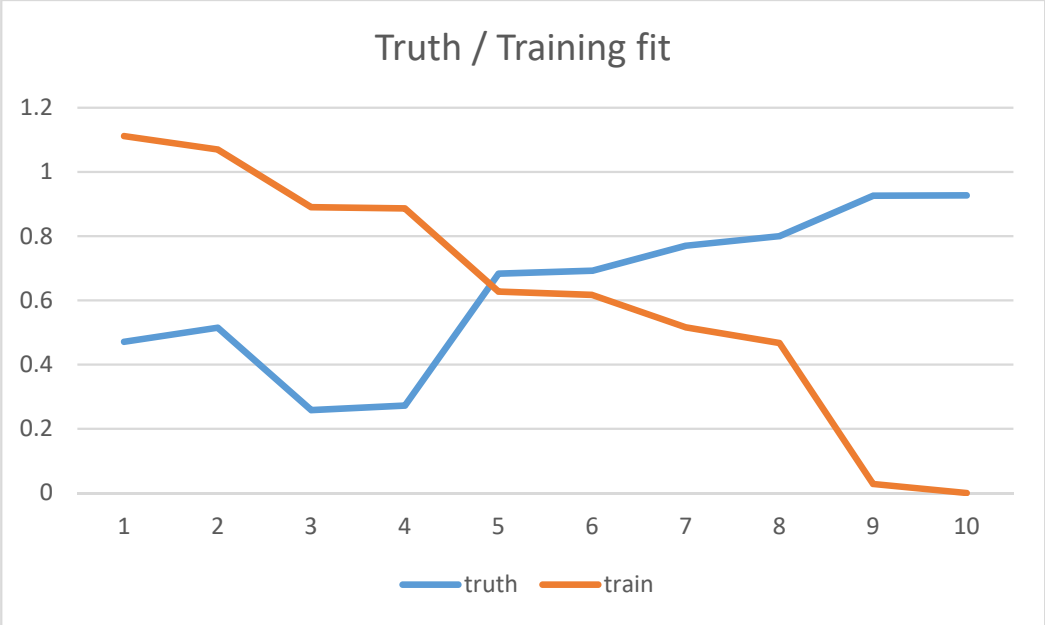

Fig. 2.5. *This shows the residual errors when a set of points are fitted to a polynomial of differeing degrees. The true value of the degree is* 3 *and there are* 10 *points.*

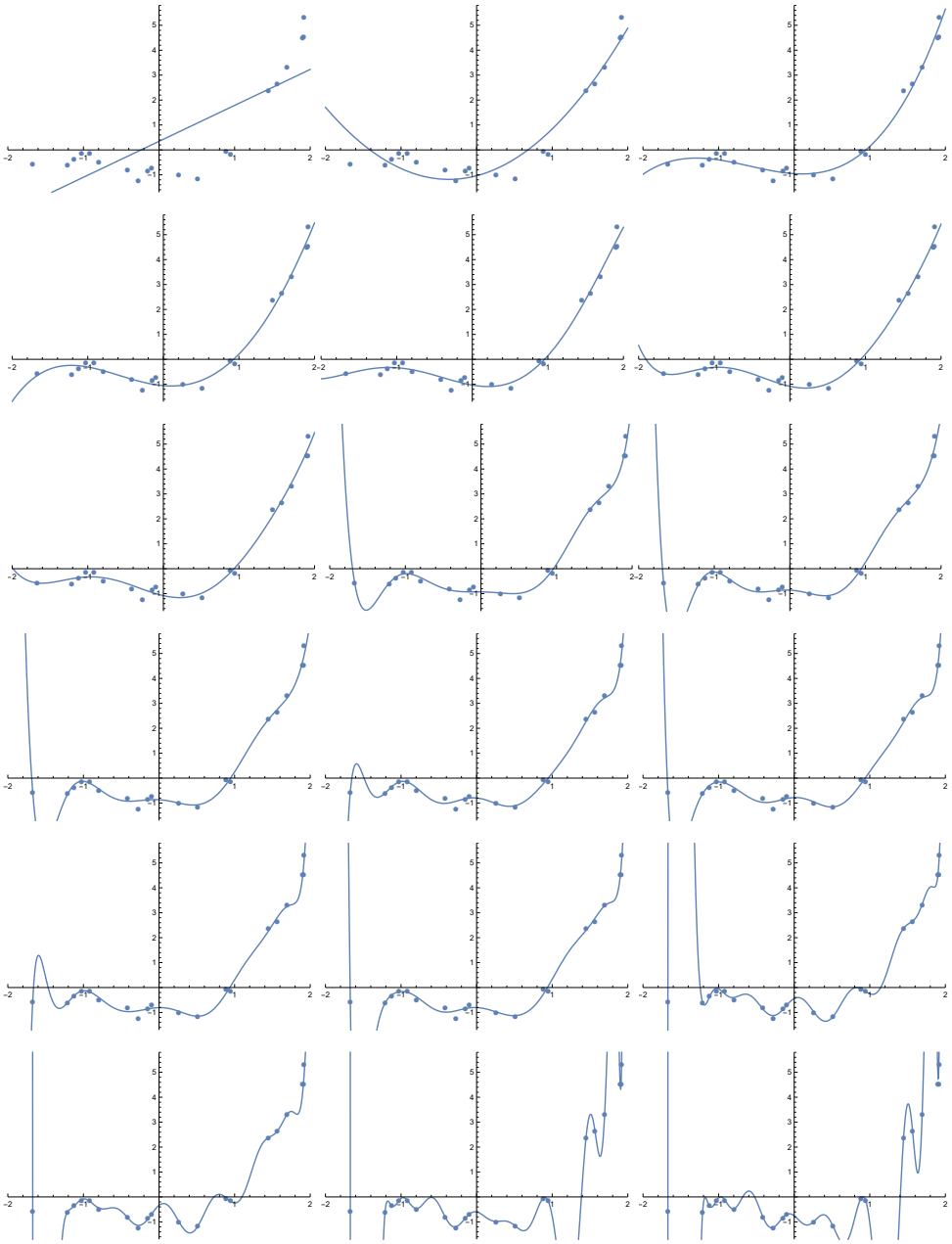

Fig. 2.6. *Fitting polynomials to points. Points are generated by a degree* 3 *polynomial. Degrees of polynomials* 1 – 19 *fitting* 20 *points.*

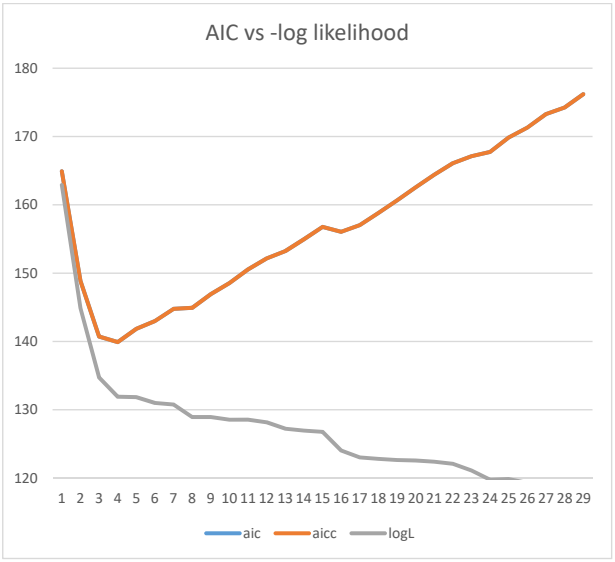

Fig. 2.7. *Akaike Information Criterion (AIC) finds the best model. Here AIC and the negative log-likelihood are plotted.*

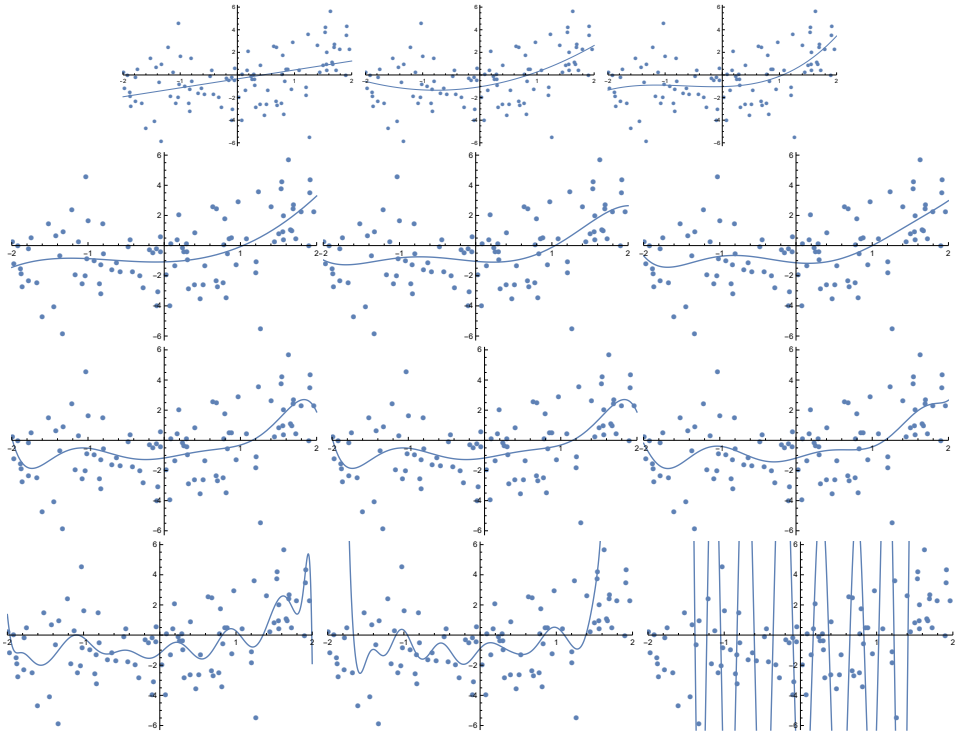

Fig. 2.8. *Fitting polynomials to points. Points are generated by a degree* 3 *polynomial. Degrees of polynomials* 1 – 9 *and (bottom row)* 20, 30, 40. *This shows how chaotic fitting of large degree polynomials to data can be. Although the curve fits the data well, it performs badly on the test data (poor generalization).*

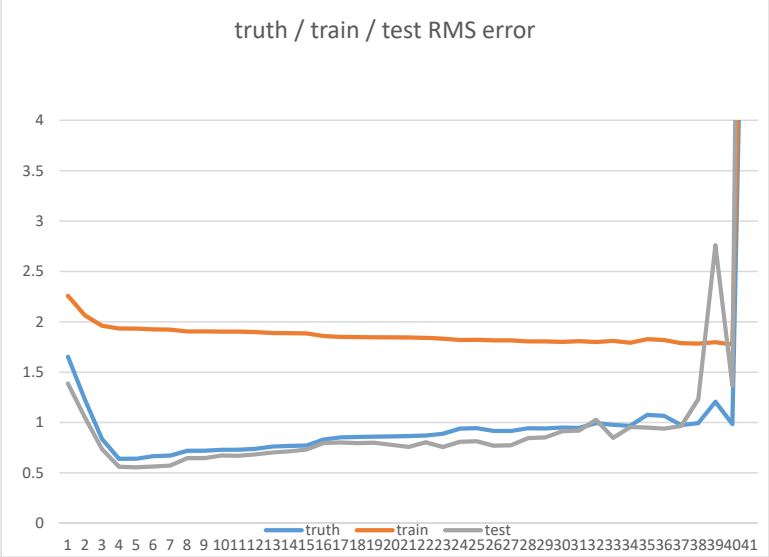

Fig. 2.9. *Training, ground truth and test RMS errors for fitting a large number of noisy points. As seen, as the degree of the polynomial increases, the error decreases until around the value of the true degree of the data. After that, training error continues to decrease, whereas both test and ground-truth error start to increase. For very high degrees (around 40) the test RMS error goes completely off the rails, because of extreme over-fitting (as seen in figure 2.8.*

2.2.1 Estimating ground-truth and training errors

It is possible to estimate the expected training and ground-truth errors, at least in the case of Gaussian errors. One needs to know the following things:

(i) The noise-level σ in the original data (values y_i).
(ii) The number of degrees of freedom, k, of the model being fitted.
(iii) The ground-truth has to fit the model for some value of the parameters.

Thus, in the case where the ground-truth data $(\bar{x}_i, \bar{y}_i)$ are generated by a polynomial model of degree $\bar{d}$. This is used to generate ground-truth data, to which Gaussian noise with variance σ^2 is added. Finally, the model is fitted with a polynomial model of degree d. Assuming that $d \geq \bar{d}$, the the third condition above is satisfied (ground-truth fits the fitted model), since the ground truth fits exactly to a degree $\bar{d}$-degree model, and hence also to a degree d model. The number of degrees of freedom of the fitting model is equal to $k = d + 1$.

This does not hold in the case where the degree of the fitting polynomial is less than the ground-truth degree, so the results to be considered only hold if $d \geq \hat{d}$.

2.2.2 Estimating the expected fitting error.

This section is a little repetive. It is adapted from the book Hartley-Zisserman, Multiview Geometry.

We consider general sorts of models here, not just the polynomial models considered so far. It is assumed that there is a set of measurements $(\mathbf{x}_i, \bar{\mathbf{y}}_i)$ and a function f_θ such that $f_\theta(\mathbf{x}_i) = \bar{\mathbf{y}}_i$, ideally. The bar on the $\bar{\mathbf{y}}_i$ indicatex that this is the "ground truth" value of the measured values. The variables $\mathbf{x}_i$ are not important, except to generate the values $\mathbf{y}_i$, via some function $\mathbf{y}_i = f_\theta(\mathbf{x}_i)$, so we will sometimes ignore them.

In general, we will assume that $\bar{\mathbf{y}}_i$ is a vector in some $\mathbb{R}^m$. If there are n of them, then they can be put all together to form a vector $\bar{\mathbf{Y}}$ in some larger space $\mathbb{R}^N$, where $N = mn$. In this case, we can write $\bar{\mathbf{Y}} = f_\theta(\mathbf{X})$, or allowing for noise, it is $\mathbf{Y} = f_\theta(\mathbf{X}) + \eta$, where η is a multi-dimensional Gaussian, in $\mathbb{R}^N$ (or more generally some probability distribution corresponding to iid measurements).

Given $(\mathbf{X}, \bar{\mathbf{Y}})$, and ML estimation algorithm is then run to compute the parameters θ of the model f_θ. A model (and an algorithm to compute the model) is then evaluated according to how closely the computed

values $\hat{\mathbf{Y}}$ match the (noisy) measured data $\mathbf{Y}$.

A model $\hat{f}_\theta$ with parameters θ is estimated to fit the data, in the sense that $\hat{f}_\theta(\mathbf{X}) = \mathbf{Y} + \eta$. The model $\hat{f}$ is not assumed to be the same one that generated the ground truth data $(\mathbf{X}, \bar{\mathbf{Y}})$, though it is assumed that for some set of the parameters, $\bar{\theta}$, the model $f_{\bar{\theta}}$ will produce the ground truth.

The RMS (root-mean-squared) residual error

$$\epsilon_{\text{res}} = \left(\frac{1}{mn} \sum_{i=1}^{n} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2 \right)^{1/2} = \|\mathbf{Y} - \hat{\mathbf{Y}}\| / \sqrt{mn} \tag{2.12}$$

expresses the per-measurement average difference between the noisy ($\mathbf{Y}$) and the estimated points $\hat{\mathbf{Y}}$. It is therefore appropriately called *residual error*. It measures how well the computed model matches the input data, and as such is a suitable quality measure for the estimation procedure.

The value of the residual error is *not* in itself a suitable measure of the quality of the model, however. For instance, consider the problem of fitting a polynomial of degree n to $n+1$ points. This can be fitted exactly, and hence the residual error is zero. One cannot expect any better performance from an algorithm than this.

Note that $\hat{f}$ matches to the the input data $\mathbf{Y}$, and not to the original noise-free data, $\bar{\mathbf{Y}}$. As was seen the model can fit the input data well, but not fit the true noise-free data well at all.

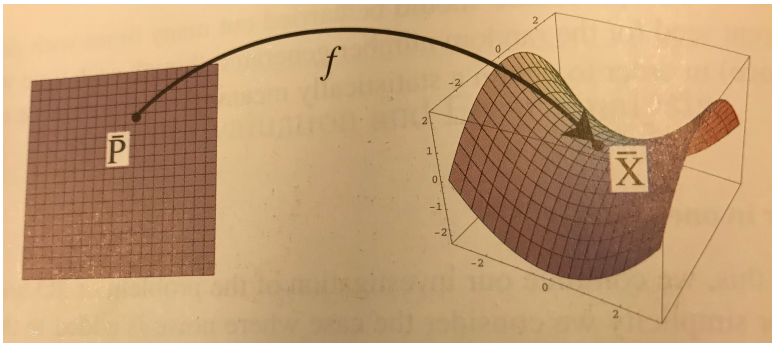

Fig. 2.10. *As the values of the parameters θ vary, the function image traces out a surface S_{M} through the true value $\bar{\mathbf{Y}}$. (Warning, in this diagram, $\bar{\mathbf{P}}$ should be θ, and $\mathbf{X}$ should be $\bar{\mathbf{Y}}$.)*

2.2.3 Optimal estimators (MLE)

Bounds for estimation performance will be considered in a general framework. The goal is to derive formulae for the expected residual error of the Maximum Likelihood Estimate (MLE).

We abstract the problem a little further. A general estimation problem is concerned with a function f from $\mathbb{R}^M$ to $\mathbb{R}^N$ where $\mathbb{R}^M$ is a parameter space, and $\mathbb{R}^N$ is a space of measurements (any hidden variables, such as $\mathbf{X}$ are not considered here). Consider now how the point $\bar{\mathbf{Y}} = f(\theta)$ varies as θ varies over the set of parameters, $\mathbb{R}^M$. As the value of the parameter vector θ varies in a neighbourhood of the ground truth parameter point $\bar{\theta}$, the value of the function $f(\theta)$ traces out a surface S_{M} in $\mathbb{R}^N$ through the point $\bar{\mathbf{Y}}$. This is illustrated in figure 2.10. The surface S_{M} is given by the range of f. The dimension of the surface as a submanifold of $\mathbb{R}^N$ is equal to d, where d is the number of essential parameters (that is the number of degrees of freedom, or minimum number of parameters). Often $d = M$, but if there are redundant parameters, then $d < M$.

A measurement $\mathbf{Y}$ is generated by adding noise to the ground truth measurement point $\bar{\mathbf{Y}}$. This may (generally will) lie off the surface S_M.

Now, given a measurement vector $\mathbf{Y}$, the maximum likelihood (ML) estimate $\hat{\mathbf{Y}}$ is the point on S_{M} closest to $\mathbf{Y}$. The ML estimator is the one that returns this closest point to $\mathbf{Y}$ that lies on this surface. Denote this ML estimate by $\hat{\mathbf{Y}}$. Refer to figure 2.11.

We now assume that in the neighbourhood of $\bar{\mathbf{Y}}$, the surface is essentially planar and is well approximated by the tangent surface – at least for neighbourhoods around $\bar{\mathbf{Y}}$ of the order of magnitude of noise variance. In this linear approximation, the ML estimate $\hat{\mathbf{Y}}$ is the foot of the perpendicular from $\mathbf{Y}$ onto the tangent plane. The residual error is the distance from the point $\mathbf{Y}$ to the estimated value $\hat{\mathbf{Y}}$. Furthermore, the distance from $\hat{\mathbf{Y}}$ to (the unknown) $\bar{\mathbf{Y}}$ is the distance from the optimally estimated value to the true value as seen in figure 2.11. Our task is to

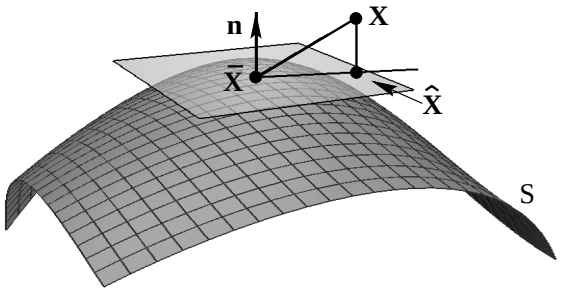

Fig. 2.11. *Geometry of the errors in measurement space using the tangent plane approximation to* S_{M} *. (***Note.** *to conform to the text, the symbols* $\mathbf{X}$ *should be* $\mathbf{Y}$*.) The estimated point* $\hat{\mathbf{Y}}$ *is the closest point on* S_{M} *to the measured point* $\mathbf{Y}$*. The residual error is the distance between the measured point* $\mathbf{Y}$ *and* $\hat{\mathbf{Y}}$*. The estimation error is the distance from* $\hat{\mathbf{Y}}$ *to the true point* $\bar{\mathbf{Y}}$*.*

compute the expected value of these errors.

Computing the expected ML residual error has now been abstracted to a geometric problem as follows. The *total variance* of an N-dimensional Gaussian distribution is the trace of the covariance matrix, that is the sum of variances in each of the axial directions. This is, of course, unchanged by a change of orthogonal coordinate frame. For an N-dimensional isotropic Gaussian distribution with independent variances σ^2 in each variable, the total variance is $N\sigma^2$. Now, given an isotropic Gaussian random variable defined on $\mathbb{R}^N$ with total variance $N\sigma^2$ and mean the true point $\bar{\mathbf{Y}}$, we wish to compute the expected distance of the random variable from a dimension d hyperplane passing through $\bar{\mathbf{Y}}$. The projection of a Gaussian random variable in $\mathbb{R}^N$ onto the d-dimensional tangent plane gives the distribution of the *estimation error* (the difference between the estimated value and the true result). Projection onto the $(N-d)$-dimensional normal to the tangent surface gives the distribution of the residual error.

By a rotation of axes if necessary, one may assume, without loss of generality, that the tangent surface coincides with the first d coordinate axes. Integration over the remaining axial directions provides the following result.

Result 2.4. *The projection of an isotropic Gaussian distribution defined on* $\mathbb{R}^N$ *with total variance* $N\sigma^2$ *onto a subspace of dimension* s *is an isotropic Gaussian distribution with total variance* $s\sigma^2$*.*

The proof of this is straightforward, and is omitted. We apply this in the two cases where $s = d$ and $s = N - d$ to obtain the following results.

Result 2.5. *Consider an estimation problem where* N *measurements are to be modelled by a function depending on a set of* d *essential parameters. Suppose the mea-*

surements are subject to independent Gaussian noise with standard deviation σ in each measurement variable.

(i) *The RMS* **residual error** *(distance of the measured from the estimated value) for the ML estimator is*

$$\epsilon_{\text{res}} = E[\|\hat{\mathbf{Y}} - \mathbf{Y}\|^2/N]^{1/2} = \sigma(1 - d/N)^{1/2} \tag{2.13}$$

(ii) *The RMS* **estimation error** *(distance of the estimated from the true value) for the ML estimator is*

$$\epsilon_{\text{est}} = E[\|\hat{\mathbf{Y}} - \bar{\mathbf{Y}}\|^2/N]^{1/2} = \sigma(d/N)^{1/2} \tag{2.14}$$

where $\mathbf{Y}$, $\hat{\mathbf{Y}}$ and $\bar{\mathbf{Y}}$ are respectively the measured, estimated and true values of the measurement vector.

Result 2.5 follows directly from result 2.4 by dividing by N to get the variance per measurement, then taking a square root to get standard deviation, instead of variance.

These values give lower bounds for residual error against which a particular estimation algorithm may be measured.

2D homography – error in one image. These two examples relate to matching sets of points in two images (image points are considered to be elements of $\mathbb{R}^2$). The images are supposed to be related by a mapping with 8 degrees of freedom (called a *projectivity* or *homography*). So the problem is one of estimating a $2D$ projectivity, given a set of points in two images. There are two cases – ones where the points are considered to have noise in one image, or in both. For the 2D projectivity estimation problem considered in this chapter, assuming error in the second image only, we have $d = 8$ and $N = 2n$, where n is the number of point matches. This is the number of measured quantities. Thus, we have for this problem

$$\begin{aligned} \epsilon_{\text{res}} &= \sigma\,(1 - 4/n)^{1/2} \\ \epsilon_{\text{est}} &= \sigma\,(4/n)^{1/2}\,. \end{aligned} \tag{2.15}$$

Graphs of these errors as n varies are shown in figure 2.12.

Error in both images. In this case, $N = 4n$ and $d = 2n+8$. As before, assuming linearity of the tangent surface in the neighbourhood of the true measurement vector $\hat{\mathbf{Y}}$, result 2.5 gives the following expected errors.

$$\begin{aligned} \epsilon_{\text{res}} &= \sigma\left(\frac{n-4}{2n}\right)^{1/2} \\ \epsilon_{\text{est}} &= \sigma\left(\frac{n+4}{2n}\right)^{1/2}. \end{aligned} \tag{2.16}$$

Graphs of these errors as n varies are also shown in figure 2.12.

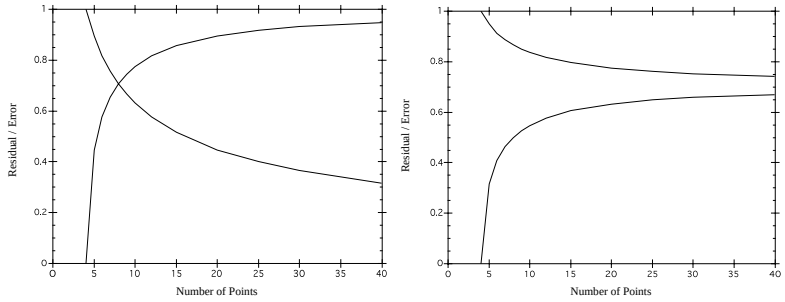

Fig. 2.12. *Optimal error when noise is present in (a) one image, and in (b) both images as the number of points varies. An error level of one pixel is assumed. The descending curve shows the estimation error* ϵ_{est} *and the ascending curve shows the residual error* ϵ_{res}.

An interesting observation to be made from this graph is that the asymptotic error with respect to the true values is $\sigma/\sqrt{2}$, and not 0 as in the case of error in one image. This result is expected, since in effect, one has two measurements of the position of each point, one in each image, related by the projective transformation. With two measurements of a point the variance in the estimate of the point position decreases by a factor of $\sqrt{2}$. By contrast, in the previous case where errors occur in one image only, one has one exact measurement for each point (i.e. in the first image). Thus, as the transformation $\mathtt{H}$ is estimated with greater and greater accuracy, the exact position of the point in the second image becomes known with uncertainty asymptotically approaching 0.

Mahalanobis distance. The formulae quoted above were derived under the assumption that the error distribution in measurement space was an isotropic Gaussian distribution, meaning that errors in each coordinate were independent. This assumption is not essential. We may assume any Gaussian distribution of error, with covariance matrix Σ. The formulae of result 2.5 remain true with ϵ being replaced with the expected Mahalanobis distance $E[\|\hat{\mathbf{Y}} - \mathbf{Y}\|^2_{\Sigma}/N]^{1/2}$. The standard deviation σ also disappears, since it is taken care of by the Mahalanobis distance.

This follows from a simple change of coordinates in the measurement space $\mathbb{R}^N$ to make the covariance matrix equal to the identity. In this new coordinate frame, Mahalanobis distance becomes the same as Euclidean distance.

2.2.4 Determining the correct convergence of an algorithm

The relations given in (2.13) and (2.14) give a simple way of determining correct convergence of an estimation algorithm, without the need to determine the number of degrees of freedom of the problem.

As seen in figure 2.11, the measurement space corresponding to the model specified by the parameter vector θ forms a surface S_{M}. If near the noise-free data $\bar{\mathbf{Y}}$ the surface is nearly planar, then it may be approximated by its tangent plane, and the

three points $\hat{\mathbf{Y}}$, $\mathbf{Y}$ and $\bar{\mathbf{Y}}$ form a right-angled triangle. In most estimation problems this assumption of planarity will be very close to correct at the scale set by typical noise magnitude. In this case, the Pythagorean equality may be written as

$$\|\mathbf{Y} - \bar{\mathbf{Y}}\|^2 = \|\mathbf{Y} - \hat{\mathbf{Y}}\|^2 + \|\bar{\mathbf{Y}} - \hat{\mathbf{Y}}\|^2 \tag{2.17}$$

In evaluating an algorithm with synthetic data, this equality allows a simple test to see whether the algorithm has converged to the optimal value. If the estimated value $\hat{\mathbf{Y}}$ satisfies this equality, then it is a strong indication that the algorithm has found the true global minimum. Note that it is unnecessary in applying this test to determine the number of degrees of freedom of the problem. A few more properties are listed:

- This test can be used to determine on a run-by-run basis whether the algorithm has succeeded. Thus, with repeated runs, it allows an estimate of the percentage success rate for the algorithm.
- This test can only be used for synthetic data, or at least data for which the true measurements $\bar{\mathbf{Y}}$ are known.
- The equality (2.17) depends on the assumption that the surface S_{M} consisting of valid measurements is locally planar. If the equality is not satisfied for a particular run of the estimation algorithm, then this is because the surface is not planar, or (far more likely) because the algorithm is failing to find the best solution.
- The test (2.17) is a test for the algorithm finding the global, not a local solution. If $\hat{\mathbf{Y}}$ settles to a local cost minimum, then the right-hand-side of (2.17) is likely to be much larger than the left-hand-side. The condition is unlikely to be satisfied entirely by chance if the algorithm finds the incorrect point $\hat{\mathbf{Y}}$.

2.3 Choosing between models - the AIC

AIC stands for the Akaike Information Criterion. As stated by Akaike in "A New Look at the Statistical Model Identification", *IEEE Transactions on Automatic Control*, December 1974, "IC stands for information criterion and A is added so that similar statistics, BIC, DIC etc. , may follow." In fact, BIC is Bayesian Information Criterion and DIC is Deviance Information Criterion.

The problem being considered is which of many models may be used to fit some set of data. For instance, if a set of data should be fitted to a polynomial model, what should be the degree of the polynomial? It is assumed that measurements to which the model should be fitted are subject to some noise perturbance, and we shall usually consider here that this is Gaussian iid noise, though the variance may be unknown. Given a model, it is assumed that the ML estimate $\hat{\theta}$ is found. That is

$$\hat{\theta} = \mathrm{argmax}_{\theta} \mathcal{L}(\theta \mid \mathbf{Y}) \ .$$

The Akaike Information Criterions (AIC) defines

$$\mathrm{AIC}(\hat{\theta}) = -2\log(\mathcal{L}(\hat{\theta} \mid \mathbf{Y})) + 2k \tag{2.18}$$

where k is the number of free parameters of the model, namely the dimension d of the measurement surface in figure 2.11.

Minimizing the first term, the *negative log-likelihood*, was shown to be equivalent to maximizing the likelihood $\mathcal{L}(\hat{\theta} \mid \mathbf{Y}) = P(\mathbf{Y} \mid \hat{\theta})$, and hence is a reasonable thing to do. Furthermore, it was shown in (2.11 – p 30) that for Gaussian noise models the negative log-likelihood is given by

$$-\log \mathcal{L}(\theta \mid \mathbf{d}) = C + \frac{n}{2}\log(r^2/n) + \frac{n}{2} \ .$$

Since n is constant, the number of measurements, this is equal, apart from constants, to $n/2\ \log(r^2)$. Ignoring n, then, for models with a Gaussian noise model, the AIC is given by

$$\mathrm{AIC} = n\log(r^2/n) + 2k \ .$$

If the AIC is computed for a number of different models, we then choose the model that gives the smallest value of the AIC.

Let's see how this works. This is shown in figure 2.9 and figure 2.5 (p 33).

2.3.1 Reduced sample AIC – AICc

It is observed that if the number of degrees of freedom, k of the model is large with respect to N, the number of measurements, then the value of AIC needs to be corrected.

This is particularly significant when $N = k$, for then the model can fit the data exactly, and so the term $\log(r^2)$ or $-\log(\mathcal{L}(\hat{\theta}))$ becomes $-\infty$, and hence this will represent the minimum value. Normally, however, this will give a poor result (over-fitting).

To correct this, a modified version of AIC, known as AICc is proposed.

$$\mathrm{AICc}(\hat{\theta}) = -2\log(\mathcal{L}(\hat{\theta} \mid \mathbf{Y})) + \frac{2kn}{n-k-1} \ . \tag{2.19}$$

Note that this correction term becomes infinite when $k = n-1$, and so this corrects for the fact that $\mathcal{L}(\theta \mid \mathbf{Y})$ becomes negative-infinite. Example of the AICc is given in figure 2.5 (p 33).

2.3.2 Justification of the AIC

It may look like simply adding the number of parameters of the model is entirely heuristic (a fudge). However, there is an argument that this is a sensible thing to do, based on geometry. We assume a Gaussian distribution of noise.

The situation is slightly different from, but similar to that considered in figure 2.11. In this case we assume that a measurement vector $\mathbf{Y}$ is observed and a model $\hat{\mathbf{Y}}$ is found. Under the assumption that the true measurement $\mathbf{Y}$ lies on the surface S_M (meaning that the data is generated by some parametrization of the model), ??(**??** – *p* **??**) gives

$$\epsilon^2_{\text{res}}/n = E[\|\hat{\mathbf{Y}} - \mathbf{Y}\|^2]/n = \sigma^2(1 - k/n) \tag{2.20}$$

On the other hand, the distance of the measured data $\mathbf{Y}$ from the ground truth $\bar{\mathbf{Y}}$ is

$$\epsilon^2_{\text{meas}}/n = E[\|\bar{\mathbf{Y}} - \mathbf{Y}\|^2]/n = \sigma^2 \ . \tag{2.21}$$

We wish to evaluate the model, not by how well it fits the measured data, but by how well the measured data $\mathbf{Y}$ fits the ground-truth data, $\hat{\mathbf{Y}}$, which gives a measure of the probability of the measured data $\mathbf{Y}$. We cannot estimate ϵ_{meas} directly, since the ground-truth data is not known.

$$E[\|\bar{\mathbf{Y}} - \mathbf{Y}\|^2] = \frac{r^2}{n} \ \frac{1}{1 - k/n} \ . \tag{2.22}$$

Taking logarithms gives

$$\log(E[\|\bar{\mathbf{Y}} - \mathbf{Y}\|^2]) = \log(r^2/n) - \log(1 - k/n) \ . \tag{2.23}$$

We now use the first term of approximation of the series expansion of logarithm. $\log(1 - x) \approx -x$ for small x. This gives finally

$$n \log(E[\|\bar{\mathbf{Y}} - \mathbf{Y}\|^2]) = n \log(r^2/n) + k \ . \tag{2.24}$$

This identifies the AIC as an estimate of $n \log(E[\|\bar{\mathbf{Y}} - \mathbf{Y}\|^2])$, which is proportional to the log-probability of the estimated squared error of the measurements.

Another way of getting the same result is to write (2.25) as

$$n \log(E[\|\bar{\mathbf{Y}} - \mathbf{Y}\|^2]) = n \log(r^2/n) - \log((1 - k/n)^n) \ . \tag{2.25}$$

Now, $\lim_{n\to\infty}(1 - k/n)^n = e^{-k}$. So, in the limit, as $n \to \infty$, this formula converges to $n \log(r^2/n) - \log(e^{-k}) = n \log(r^2/n) + k$.

For some reason this only has a correction term of k, when we require that the correction term be $2k$.

2.3.3 More material

More stuff on model fitting is given in this set of slides:
http://users.monash.edu/~dschmidt/ModelSectionTutorial1_SchmidtMakalic_2008.pdf

A paper on geometric model selection: https://www.sciencedirect.com/science/article/pii/S0923045996800322

Paper by Phil Torr on rigid motion model selection: https://ieeexplore.ieee.org/document/609296

Akaike's paper: http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/MiscDocs/Akaike_1974.pdf

3

Lecture-3: Robust model fitting

3.1 Example of model fitting: Computer Vision

Understanding (and modelling) the geometric structure of the world through images taken with a camera requires some understanding of camera geometry.

The basic pinhole model. Most commonly, cameras are modelled in terms of the pinhole, or central projection model, where points in space are projected through a pinhole onto an image plane. We consider the central projection of points in space onto a plane. Let the centre of projection be the origin of a Euclidean coordinate system, and consider the plane $\mathrm{Z} = f$, which is called the *image plane* or *focal plane*. Under the pinhole camera model, a point in space with coordinates $\mathbf{X} = (\mathrm{X}, \mathrm{Y}, \mathrm{Z})^\mathsf{T}$ is mapped to the point on the image plane where a line joining the point $\mathbf{X}$ to the centre of projection meets the image plane. This is shown in figure 3.1. By similar triangles, one quickly computes that the point $(\mathrm{X}, \mathrm{Y}, \mathrm{Z})^\mathsf{T}$ is mapped to the point $(f\mathrm{X}/\mathrm{Z}, f\mathrm{Y}/\mathrm{Z}, f)^\mathsf{T}$ on the image plane. Ignoring the final image coordinate, we see that

$$(\mathrm{X}, \mathrm{Y}, \mathrm{Z})^\mathsf{T} \mapsto (f\mathrm{X}/\mathrm{Z}, f\mathrm{Y}/\mathrm{Z})^\mathsf{T} \tag{3.1}$$

describes the central projection mapping from world to image coordinates. This is a mapping from Euclidean 3-space $\mathbb{R}^3$ to Euclidean 2-space $\mathbb{R}^2$.

The centre of projection is called the *camera centre*. It is also known as the *optical centre*. The line from the camera centre perpendicular to the image plane is called the *principal axis* or *principal ray* of the camera, and the point where the principal axis meets the image plane is called the *principal point*. The plane through the camera centre parallel to the image plane is called the *principal plane* of the camera.

Central projection using homogeneous coordinates. If the world and image points are represented by homogeneous vectors, then central projection is very simply expressed as a linear mapping between their homogeneous coordinates. In

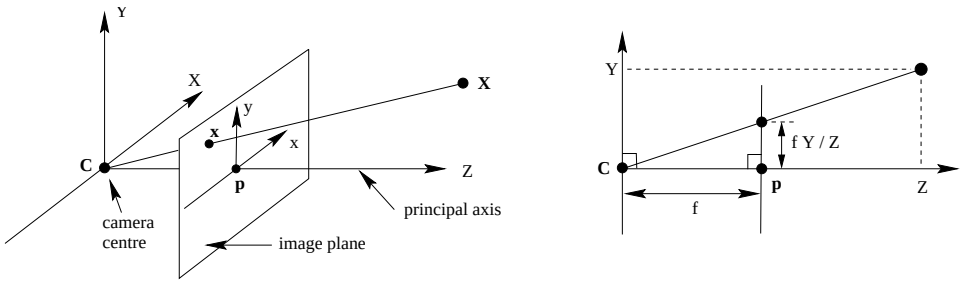

Fig. 3.1. **Pinhole camera geometry.** **C** *is the camera centre and* **p** *the principal point. The camera centre is here placed at the coordinate origin. Note the image plane is placed in front of the camera centre.*

particular, (3.1) may be written in terms of matrix multiplication as

$$\begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f\mathrm{X} \\ f\mathrm{Y} \\ \mathrm{Z} \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix}. \tag{3.2}$$

The matrix in this expression may be written as $\mathrm{diag}(f, f, 1)[\mathtt{I} \mid \mathbf{0}]$ where $\mathrm{diag}(f, f, 1)$ is a diagonal matrix and $[\mathtt{I} \mid \mathbf{0}]$ represents a matrix divided up into a 3×3 block (the identity matrix) plus a column vector, here the zero vector.

We now introduce the notation $\mathbf{X}$ for the world point represented by the homogeneous 4-vector $(\mathrm{X}, \mathrm{Y}, \mathrm{Z}, 1)^\mathsf{T}$, $\mathbf{x}$ for the image point represented by a homogeneous 3-vector, and $\mathtt{P}$ for the 3×4 homogeneous *camera projection matrix*. Then (3.2) is written compactly as

$$\mathbf{x} = \mathtt{P}\mathbf{X}$$

which defines the camera matrix for the pinhole model of central projection as

$$\mathtt{P} = \mathrm{diag}(f, f, 1)\,[\mathtt{I} \mid \mathbf{0}].$$

Principal point offset. The expression (3.1) assumed that the origin of coordinates in the image plane is at the principal point. In practice, it may not be, so that in general there is a mapping

$$(\mathrm{X}, \mathrm{Y}, \mathrm{Z})^\mathsf{T} \mapsto (f\mathrm{X}/\mathrm{Z} + p_x, f\mathrm{Y}/\mathrm{Z} + p_y)^\mathsf{T}$$

where $(p_x, p_y)^\mathsf{T}$ are the coordinates of the principal point. See figure 3.2. This equation may be expressed conveniently in homogeneous coordinates as

$$\begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f\mathrm{X} + \mathrm{Z}p_x \\ f\mathrm{Y} + \mathrm{Z}p_y \\ \mathrm{Z} \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix}. \tag{3.3}$$

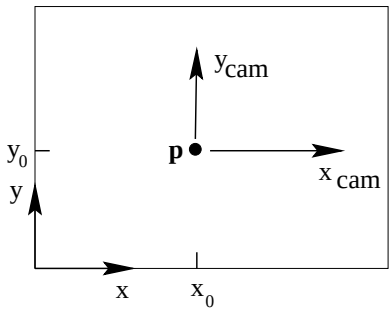

Fig. 3.2. *Image* (x, y) *and camera* $(x_{\text{cam}}, y_{\text{cam}})$ *coordinate systems.*

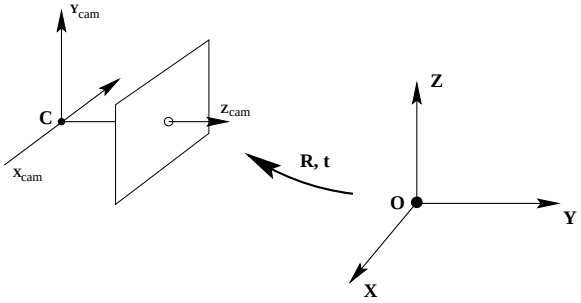

Fig. 3.3. *The Euclidean transformation between the world and camera coordinate frames.*

Now, writing

$$\mathtt{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{3.4}$$

then (3.3) has the concise form (in homogeneous coordinates)

$$\mathbf{x} = \mathtt{K}[\mathtt{I} \mid \mathbf{0}]\mathbf{X}_{\text{cam}}. \tag{3.5}$$

The matrix $\mathtt{K}$ is called the *camera calibration matrix*. In (3.5) we have written $(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, 1)^\mathsf{T}$ as $\mathbf{X}_{\text{cam}}$ to emphasize that the camera is assumed to be located at the origin of a Euclidean coordinate system with the principal axis of the camera pointing straight down the $\mathtt{Z}$-axis, and the point $\mathbf{X}_{\text{cam}}$ is expressed in this coordinate system. Such a coordinate system may be called the *camera coordinate frame*.

Camera rotation and translation. In general, points in space will be expressed in terms of a different Euclidean coordinate frame, known as the *world coordinate frame*. The two coordinate frames are related via a rotation and a translation. See figure 3.3. If $\tilde{\mathbf{X}}$ is an inhomogeneous 3-vector representing the coordinates of a point in the world coordinate frame, and $\tilde{\mathbf{X}}_{\text{cam}}$ represents the same point in the camera coordinate frame, then we may write $\tilde{\mathbf{X}}_{\text{cam}} = \mathtt{R}(\tilde{\mathbf{X}} - \tilde{\mathbf{C}})$, where $\tilde{\mathbf{C}}$ represents the coordinates of the camera centre in the world coordinate frame, and $\mathtt{R}$ is a 3×3

rotation matrix representing the orientation of the camera coordinate frame. This equation may be written in homogeneous coordinates as

$$\mathbf{X}_{\text{cam}} = \begin{bmatrix} \mathtt{R} & -\mathtt{R}\tilde{\mathbf{C}} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix} \begin{pmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{pmatrix} = \begin{bmatrix} \mathtt{R} & -\mathtt{R}\tilde{\mathbf{C}} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix} \mathbf{X}. \tag{3.6}$$

Putting this together with (3.5) leads to the formula

$$\mathbf{x} = \mathtt{K}\mathtt{R}[\mathtt{I} \mid -\tilde{\mathbf{C}}]\mathbf{X} \tag{3.7}$$

where $\mathbf{X}$ is now in a world coordinate frame. This is the general mapping given by a pinhole camera. One sees that a general pinhole camera, $\mathtt{P} = \mathtt{K}\mathtt{R}[\mathtt{I} \mid -\tilde{\mathbf{C}}]$, has 9 degrees of freedom: 3 for $\mathtt{K}$ (the elements f, p_x, p_y), 3 for $\mathtt{R}$, and 3 for $\tilde{\mathbf{C}}$. The parameters contained in $\mathtt{K}$ are called the *internal* camera parameters, or the *internal orientation* of the camera. The parameters of $\mathtt{R}$ and $\tilde{\mathbf{C}}$ which relate the camera orientation and position to a world coordinate system are called the *external* parameters or the *exterior orientation*.

It is often convenient not to make the camera centre explicit, and instead to represent the world to image transformation as $\tilde{\mathbf{X}}_{\text{cam}} = \mathtt{R}\tilde{\mathbf{X}} + \mathbf{t}$. In this case the camera matrix is simply

$$\mathtt{P} = \mathtt{K}[\mathtt{R} \mid \mathbf{t}] \tag{3.8}$$

where from (3.7) $\mathbf{t} = -\mathtt{R}\tilde{\mathbf{C}}$.

CCD cameras. The pinhole camera model just derived assumes that the image coordinates are Euclidean coordinates having equal scales in both axial directions. In the case of CCD cameras, there is the additional possibility of having non-square pixels. If image coordinates are measured in pixels, then this has the extra effect of introducing unequal scale factors in each direction. In particular if the number of pixels per unit distance in image coordinates are m_x and m_y in the x and y directions, then the transformation from world coordinates to pixel coordinates is obtained by multiplying (3.4) on the left by an extra factor $\text{diag}(m_x, m_y, 1)$. Thus, the general form of the calibration matrix of a CCD camera is

$$\mathtt{K} = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \tag{3.9}$$

where $\alpha_x = f m_x$ and $\alpha_y = f m_y$ represent the focal length of the camera in terms of pixel dimensions in the x and y direction respectively. Similarly, $\tilde{\mathbf{x}}_0 = (x_0, y_0)$ is the principal point in terms of pixel dimensions, with coordinates $x_0 = m_x p_x$ and $y_0 = m_y p_y$. A CCD camera thus has 10 degrees of freedom.

Finite projective camera. For added generality, we can consider a calibration matrix of the form

$$\mathtt{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}. \tag{3.10}$$

The added parameter s is referred to as the *skew* parameter. The skew parameter will be zero for most normal cameras. However, in certain unusual instances which are described in ??section **??** (*p* **??**), it can take non-zero values.

A camera

$$\mathtt{P} = \mathtt{KR}[\mathtt{I} \mid -\tilde{\mathbf{C}}] \tag{3.11}$$

for which the calibration matrix $\mathtt{K}$ is of the form (3.10) will be called a *finite projective* camera. A finite projective camera has 11 degrees of freedom. This is the same number of degrees of freedom as a 3×4 matrix, defined up to an arbitrary scale.

Note that the left hand 3×3 submatrix of $\mathtt{P}$, equal to $\mathtt{KR}$, is non-singular. Conversely, any 3×4 matrix $\mathtt{P}$ for which the left hand 3×3 submatrix is non-singular is the camera matrix of some finite projective camera, because $\mathtt{P}$ can be decomposed as $\mathtt{P} = \mathtt{KR}[\mathtt{I} \mid -\tilde{\mathbf{C}}]$. Indeed, letting $\mathtt{M}$ be the left 3×3 submatrix of $\mathtt{P}$, one decomposes $\mathtt{M}$ as a product $\mathtt{M} = \mathtt{KR}$ where $\mathtt{K}$ is upper-triangular of the form (3.10) and $\mathtt{R}$ is a rotation matrix. This decomposition is essentially the RQ matrix decomposition, The matrix $\mathtt{P}$ can therefore be written $\mathtt{P} = \mathtt{M}[\mathtt{I} \mid \mathtt{M}^{-1}\mathbf{p}_4] = \mathtt{KR}[\mathtt{I} \mid -\tilde{\mathbf{C}}]$ where $\mathbf{p}_4$ is the last column of $\mathtt{P}$. In short

- *The set of camera matrices of finite projective cameras is identical with the set of homogeneous* 3×4 *matrices for which the left hand* 3×3 *submatrix is non-singular.*

General projective cameras. The final step in our hierarchy of projective cameras is to remove the non-singularity restriction on the left hand 3×3 submatrix. A *general projective* camera is one represented by an arbitrary homogeneous 3×4 matrix of rank 3. It has 11 degrees of freedom. The rank 3 requirement arises because if the rank is less than this then the range of the matrix mapping will be a line or point and not the whole plane; in other words not a 2D image.

A heirarchy of camera models. We consider cameras modelled by a camera matrix $\mathtt{P}$, which is a 3×4 matrix, acting on homogeneous coordinates of 3D and $2D$ points. $\mathtt{P}$ may be decomposed as

$$\mathtt{P} = \mathtt{K}[\mathtt{R} \mid \mathbf{t}]$$

where $\mathtt{K}$ is an upper-triangular "calibration matrix", and $\mathtt{R}$ and $\mathbf{t}$ represent rotation and translation.

A possible hierarchy of models (with increasing numbers of parameters, or degrees of freedom) is

(i) Calibrated camera: the matrix $\mathtt{K}$ is known. This model has 6 degrees of freedom, 3 for $\mathbb{R}$ and 3 for $\mathbf{t}$.

(ii) Unknown focal length: the matrix $\mathtt{K}$ is diagonal, $\mathtt{K} = \mathrm{diag}(f, f, 1)$, where f is the focal length. The model has 7 parameters.

(iii) Unknown principal points (centre of the image). This may occur if the image is cropped. The matrix $\mathtt{K}$ is of the form

$$\mathtt{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}$$

and the model has 9 parameters.

(iv) General projective camera. In this case, $\mathtt{P}$ is an arbitrary 3×4 matrix. This model has 11 degrees of freedom (because multiplying $\mathtt{P}$ by a constant has no effect).

(v) Radial distortion: some cameras do not conform exactly to a pinhole model (such as fish-eye lenses). They have radial distortion, where points are moved radially some distance. This radial distortion is commonly modelled as a polynomial model, with two coefficients, sometimes called κ_2 and κ_4. This adds to parameters to the model.

3.1.1 Modelling tasks in camera geometry

.

There are numerous modelling problems that arise in image geometry (sometimes called multiview geometry). Here are some:

Camera calibration. One is given the position of n points $\mathbf{X}_i$ in the 3D world, and the coordinates of their images $\mathbf{x}_i$ in an image (sometimes several images). The task is to find the parameters of the camera, using one of the hierarchy of camera models. The model to be optimized is then of the form

$$\mathbf{x}_j = \mathtt{P}\mathbf{X}_j + \eta \ .$$

In this case, η is commonly modelled as a 2-dimensional Gaussian distribution, representing 2D displacement of the point in the image plane, due to limited accuracy of measuring points' positions in the image.

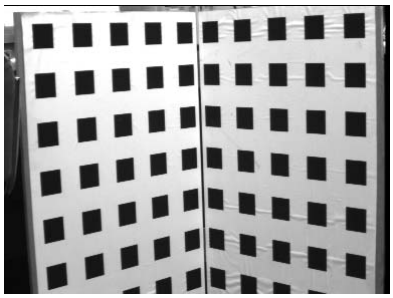

Fig. 3.4. *An image of a typical calibration object. The black and white checkerboard pattern (a "Tsai grid") is designed to enable the positions of the corners of the imaged squares to be obtained to high accuracy. A total of 197 points were identified and used to calibrate the camera.*

Relative pose. Given point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ between two cameras with matrices $\mathtt{P}$ and $\mathtt{P}'$, usually assumed to have the same parameters, except for their pose, find their relative displacement. A corresponding point pair $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ corresponds to an unknown 3D point $\mathbf{X}_i$ in space. It is required to find the rigid displacement, represented by a rotation and translation that takes one camera to the other. (This has 6 degrees of freedom, but only 5 may be determined.)

This problem is modelled as

$$\mathbf{x}_i = \mathtt{P}\mathbf{X} + \eta$$
$$\mathbf{x}'_i = \mathtt{P}\mathtt{T}\mathbf{X} + \eta$$

where $\mathtt{T}$ represents a 3D rigid transformation.

The number of measurements is $4n$, where n is the number of points, namely two image coordinates in each image. The number of parameters is $5 + 3n$, as follows. The 5 represents the degrees of freedom of the rigid transformation T (actually 6, but we cannot get scale, so call it 5). The $3n$ represent coordinates of the n points $\mathbf{X}_i$, which also need to be estimated. Thus, this task is possible if $4n \geq 5 + 3n$, or $n \geq 5$. (There is a well-known algorithm, the 5-point algorithm https://dl.acm.org/citation.cfm?id=987623, or https://dl.acm.org/citation.cfm?id=1171943. A different solution that uses 8 points is given in https://en.wikipedia.org/wiki/Eight-point_algorithm

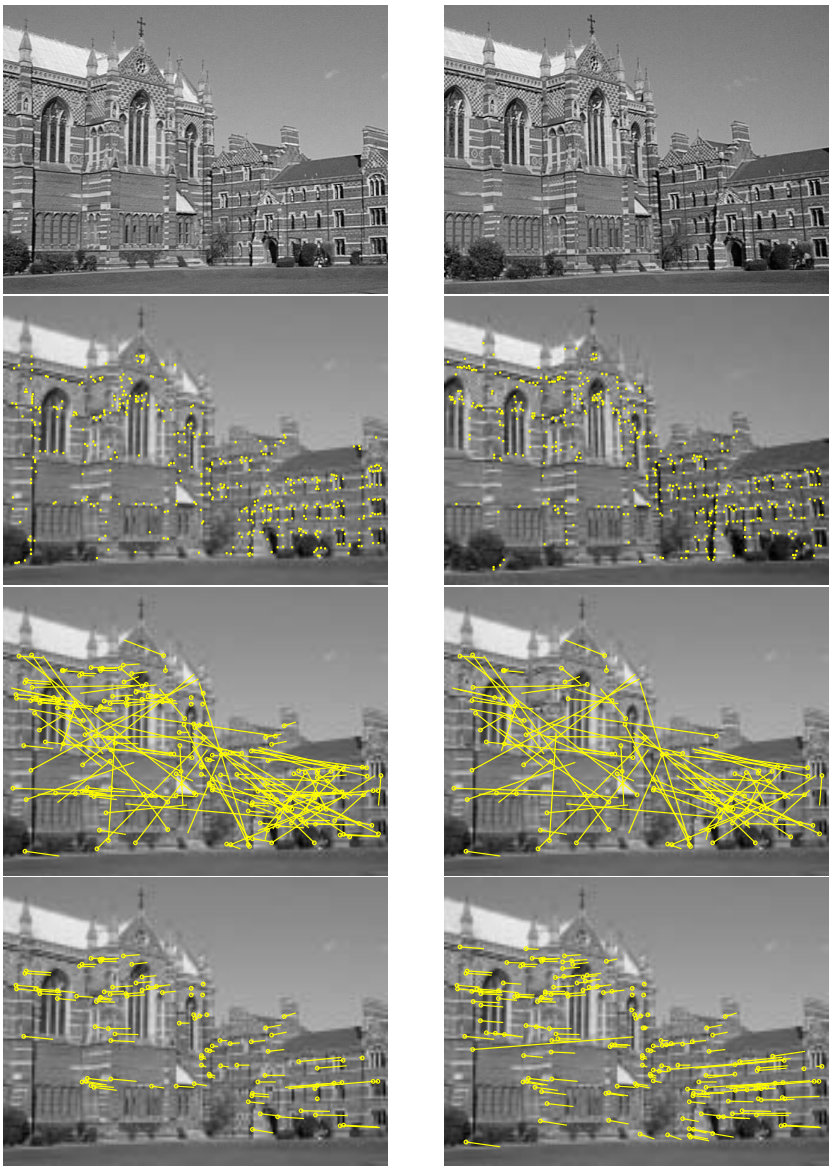

Fig. 3.5. **Automatic computation of relative pose between two images using RANSAC.** *Row 1: left and right images of Keble College, Oxford. The motion between views is a translation and rotation. The images are* 640×480 *pixels. Row 2: detected corners superimposed on the images. There are approximately 500 corners on each image. The following results are superimposed on the left image: Row 3, left: 188 putative matches shown by the line linking corners, note the clear mismatches; Row 3, right: outliers – 89 of the putative matches. Row 4, left: inliers – 99 correspondences consistent with the estimated* `F`*; Row 5, right: final set of 157 correspondences after guided matching and MLE. There are still a few mismatches evident, e.g. the long line on the left.*

Homography estimation. A homography is a mapping between two images (or in general between two 2-dimensional "projective space"). Thus, given points $\mathbf{x}$ and $\mathbf{x}'$ in the two images that are both images of the same 3D point (unknown), then they are related by a mapping:

$$\mathbf{x}' = \mathtt{H}\mathbf{x} \ .$$

The mapping has 8 degrees of freedom, in general.

There is a homography accurately mapping one image to another accurately, under two notable circumstances.

(i) The two images are taken from the same location at an unchanged scene. Thus the camera simply rotates, but does not move laterally. (It is possible that the camera can zoom, however). There is a homography relating the regions of the two images visible in both. In this case, the homography is of a particular type that can be parametrized by fewer than 8 parameters. In fact, if the camera is fully callibrated then the homography $\mathtt{H}$ has only 3 degrees of freedom, corresponding to the three degrees of freedom of a rotation. In fact, in this case, the mapping is known as a "perspectivity".

(ii) A moving camera takes two images of a planar scene, for instance a chessboard pattern, a wall or floor. The two images will look substantially different, but will be related by a homography.

(iii) If the scene (world) is a long distance from the cameras (at infinity), such as a picture of the sky, relative to the motion of the camera, then the mapping from one image to another is approximated by a homography.

See figure 3.16 (*p* 81) for an example of homography estimation.

Full 3D reconstruction. The full 3D construction problem is a modelling problem in which images $\mathbf{x}_{ij}$ are given, which are the coordinates of an unknown i-th 3D point in a j-th image. The task is to find the positions of all the points $\mathbf{X}_j$ and the poses of all the cameras, represented by $\mathtt{P}_i$. Not all x_{ij} are known, since some points are not found in some images. Once again, the problem can be modelled as

$$\mathbf{x}_{ij} = \mathtt{P}_i\mathbf{X}_j + \eta$$

In this case, it is not generally assumed that all the cameras $\mathtt{P}_i$ are the same, except in case of *visual odometry* (finding the motion of a moving camera).

3.2 Robust model fitting

This material has been adapted (modified) from book Hartley-Zisserman, Multiview Geometry.

In fitting a model

$$y_i = f_\theta(x_i) + \eta$$

Fig. 3.6. *3D reconstruction of San Marco, Venice, downloaded from* https://www.researchgate.net/figure/Our-dense-reconstruction-of-Piazza-San-Marco-Venice-fr fig1_221363626

it is often appropriate to assume some different distribution than Gaussian for the noise model η. For a specific instantiation of the model, f_θ with parameters θ, the *residual* is defined by

$$\delta_i = y_i - f_\theta(x_i) \ ,$$

the residual for the i-th measurement. The residual δ_i is explained by the probability distribution η, and η defines the probability of an error δ_i according to some function $p_\eta(\delta)$, or simply $p(\delta)$, expressing the probability of an error δ.

In the previous lecture, it was assumed that η is a zero-mean Gaussian distribution, and η may itself have parameters, just as a zero-mean Gaussian distribution has the additional parameter σ^2, the variance.

In this section, it will be investigated how a Gaussian distribution is not always the best choice of noise model. There are two common reasons for choosing a Gaussian distribution:

(i) Because of the Central Limit Theorem https://en.wikipedia.org/wiki/Central_limit_theorem, under broad conditions, the sum of repeated trials of a random variable converge to a normal distribution. This leads to some

sort of vague idea that the normal distribution will always be a good choice, since it comes up so frequently.

(ii) (Convenience). It was seen in the previous lecture, that maximizing the probability of the fit of data to a model with Gaussian noise is equivalent to minimizing a sum of squares of residuals. Thus, the probability can be maximized (the negative-log-probability is minimized) by solving either a linear or non-linear least-squares problem. Generally, least-squares problems are easier to solve – linear least-squares problems are solvable in closed form, and non-linear least squares is well studied: see for instance

- Gauss-Newton https://en.wikipedia.org/wiki/Gauss-Newton_algorithm
- Levenberg-Marquardt https://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm
- or conjugate gradient https://en.wikipedia.org/wiki/Conjugate_gradient_method optimization methods.

However, I **do not recommend** the use of Gaussian distributed noise model, unless you are clear that this is a good idea, or you cannot solve the problem any other way. Noise in measurements rarely conform to a Gaussian noise model.

Outliers. Very often measured data contain outliers, which simply means that they do not conform to the distribution of 'correctly" measured data. Outliers can occur because of a number of things.

(i) The experimenter was drunk when he took the measurements.
(ii) The measurements were extracted by a computer program, which commonly makes mistakes.
(iii) The thing being measured was disturbed by a chance and unlikely event, for instance, the experimental aparatus was struck by an meteorite just as the measurement was being taken.
(iv) There was an ambiguity as to the correct measured value, and the wrong data was measured.

Most important here, perhaps, is the second case, where a computer program makes mistakes, perhaps because of ambiguity. Examples are as follows:

- Point matching in images: Finding corresponding points between two images is a first step in building a 3D computer model of a scene seen in a number of images. This is not an easy task, because of ambiguities of matching, differing scales and orientation of points in the images. The number of outliers (incorrectly matched points) can often be very large (of the order of 50%). Typically, when the right match is made, the points are matched with less than a pixel standard-deviation of error. Outliers, on the other hand can be arbitrary far from the correct matched pair. See ??figure **??** (*p* **??**).

- Seizmic activity: the earth's surface always has some seismic activity from minor activity, or even everyone jumping up and down when their team scores a goal at the World Cup. If an attempt is being made to model this activity, perhaps tracing its variation with time of day or month of year, then events such as major earthquakes can entirely skew a model. Such extraordinary events should be treated as outliers.

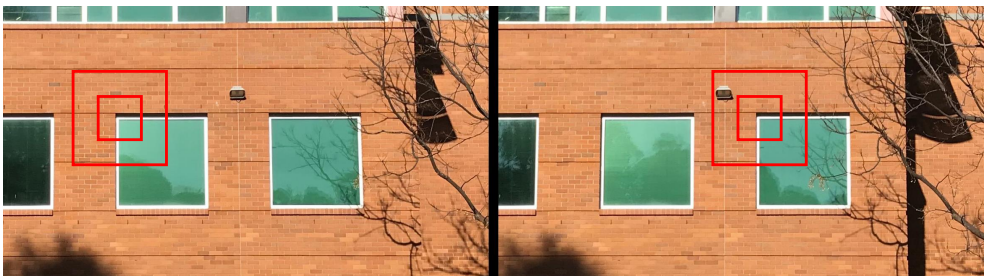

Fig. 3.7. *Local matching methods are not able to distinguish the correct match among repeated structures. In a small window, the two corners look very similar. Looking further in a larger window shows that they are a false match.*

Getting back to model fitting, if all measurements are assumed to be independent, and $p(\delta)$ is the probability distribution of an error δ in the measurement, then the probability of a set of measurement with errors δ_i is given by

$$p(\delta_1, \ldots, \delta_n) = \prod_{i=1}^{n} p(\delta_i) \ .$$

Taking the negative logarithm gives

$$-\log(p(\delta_1, \ldots, \delta_n)) = -\sum_{i=1}^{n} \log(p(\delta_i)) \tag{3.12}$$

and the right-hand side of this expression is a suitable cost function for a set of measurements. It is usually appropriate to set the cost of an exact measurement to be zero, by subtracting $\log(p(0))$, though this is not strictly necessary, if our purpose is cost minimization. Graphs of various specific cost functions to be discussed next are shown in figure 3.8.

Thus, associated with the chosen noise distribution, $p(\delta)$, we associate a cost $C(\delta)$ given by

$$C(\delta_i) = -\log(p(\delta_i))$$

and then we need to find the model f_θ that minimizes the total cost

$$C(f_\theta) = \sum_{i=1}^{n} C(f_\theta(x_i) - y_i) \ .$$

We can turn this expression around and write

$$\begin{aligned} p(\delta) &= \exp(-C(\delta)) \\ p(\delta_1, \ldots, \delta_n) &= \exp\Big(-\sum_{i=1}^{n} C(\delta_i)\Big) \end{aligned} \tag{3.13}$$

It is common to think in terms of choosing the cost function C first and then relating it to the corresponding probability distribution according to (3.13). Various different possible cost functions will be considered next.

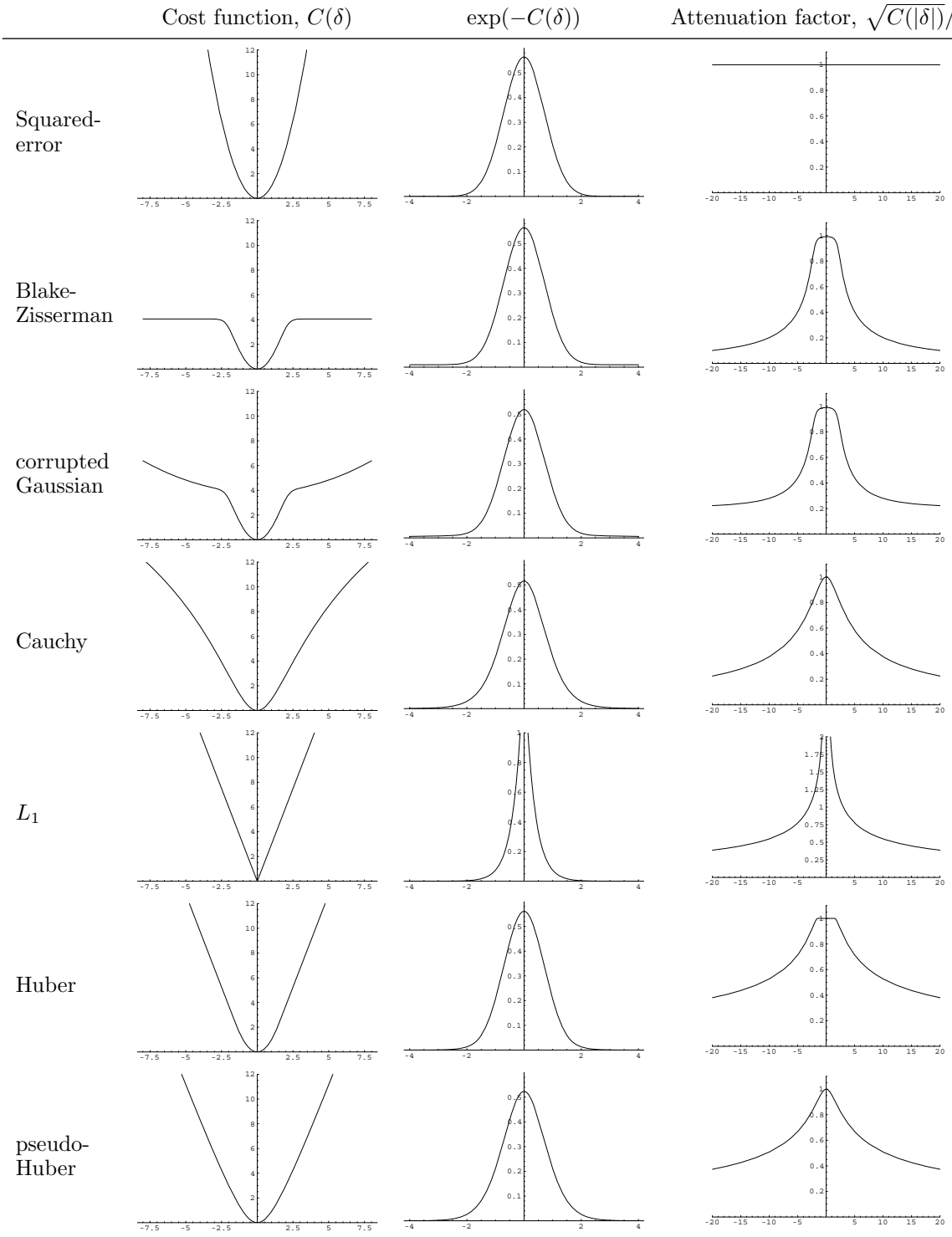

Fig. 3.8. *A comparison of different cost functions, $C(\delta)$, for robust estimation. Their corresponding* pdfs*, $\exp(-C(\delta))$, and attenuation factors ($w = C(\delta)^{1/2}/\delta$ see text) are also shown.*

Statistically based cost functions. Determination of a suitable cost function may be approached by estimating or guessing the distribution of errors for the particular measurement process involved, such as point extraction in an image. In the following list, for simplicity, we ignore the normalization constant for Gaussian distributions $(2\pi\sigma^2)^{-1/2}$, and assume that $2\sigma^2 = 1$.

(i) **Squared error.** Assuming the data is Gaussian distributed, the Probability Distribution Function (pdf) is $p(\delta) = \exp(-\delta^2)$ which leads to a cost function

$$C(\delta) = \delta^2.$$

(ii) **Blake-Zisserman.** The data is assumed to have a Gaussian distribution for inliers with a uniform distribution of outliers. The pdf is taken to be of the form $p(\delta) = \exp(-\delta^2) + \epsilon$. This is not actually a pdf, since it integrates to infinity. Nevertheless, it leads to a cost function of the form

$$C(\delta) = -\log(\exp(-\delta^2) + \epsilon).$$

For inliers (small δ), this approximates δ^2, whereas for outliers (large δ) the asymptotic cost is $-\log\epsilon$. Thus, the crossover point from inliers to outliers is given approximately by $\delta^2 = -\log\epsilon$. The actual cost function used by Blake and Zisserman in [**?**] was $\min(\delta^2, \alpha^2)$ and $\epsilon = \exp(-\alpha^2)$.

(iii) **Corrupted Gaussian.** The previous example has the theoretical disadvantage that it is not actually a pdf. An alternative is to model the outliers by a Gaussian with larger standard deviation, leading to a mixture model probability distribution of the form $p(\delta) = \alpha\ \exp(-\delta^2) + (1-\alpha)\exp(\delta^2/w^2)/w$ where w is the ratio of standard deviations of the outliers to the inliers, and α is the expected fraction of inliers. Then

$$C(\delta) = -\log(\alpha\exp(-\delta^2) + (1-\alpha)\exp(-\delta^2/w^2)/w).$$

Heuristic cost functions. We consider cost functions justified more by heuristics and required noise-immuneness properties than by adherence to a specific noise-distribution model. For this reason they will be introduced directly as a cost function, rather than a pdf.

(i) **Cauchy cost function.** The cost function is given by

$$C(\delta) = b^2\log(1 + \delta^2/b^2)$$

for some constant b. For small values of δ, this curve approximates δ^2, and the value of b determines for what range of δ this approximation is close. The cost function is derived from the Cauchy distribution $p(\delta) = 1/(\pi(1+\delta^2))$, which is a bell-curve similar to the Gaussian, but with heavier tails.

(ii) **The L_1 cost function.** Instead of using the sum of squares, we use the sum of absolute errors. Thus,

$$C(\delta) = 2b|\delta|$$

where $2b$ is some positive constant (which normally could just be 1). This cost function is known as the *total variation*.

(iii) **Huber cost function.** This cost function is a hybrid between the L_1 and least-squares cost function. Thus, we define

$$\begin{aligned} C(\delta) &= \delta^2 \quad \text{for} \quad |\delta| < b \\ &= 2b|\delta| - b^2 \quad \text{otherwise} \end{aligned}$$

This cost function is continuous with continuous first derivative. The value of the threshold b is chosen approximately to equal the outlier threshold.

(iv) **Pseudo-Huber cost function.** The cost function

$$C(\delta) = 2b^2(\sqrt{1 + (\delta/b)^2} - 1)$$

is very similar to the Huber cost function, but has continuous derivatives of all orders. Note that it approximates δ^2 for small δ and is linear with slope $2b$ for large δ.

3.2.1 Properties of the different cost functions

Squared error. The most basic cost function is the squared error $C(\delta) = \delta^2$. Its main drawback is that it is not robust to outliers in the measurements, as we shall see. Because of the rapid growth of the quadratic curve, distant outliers exert an excessive influence, and can draw the cost minimum well away from the desired value.

Non-convex cost functions. The Blake-Zisserman, corrupted Gaussian and Cauchy cost functions seek to mitigate the deleterious effect of outliers by giving them diminished weight. As is seen in the plot of the first two of these, once the error exceeds a certain threshold, it is classified as an outlier, and the cost remains substantially constant. The Cauchy cost function also seeks to deemphasize the cost of outliers, but this is done more gradually. These three cost functions are non-convex, which has important effects as we will see.

Asymptotically linear cost functions. The L_1 cost function measures the absolute value of the error. The main effect of this is to give outliers less weight compared with the squared error. The key to understanding the performance of this cost function is to observe that it acts to find the *median* of a set of data. Consider a set of real valued data $\{a_i\}$ and a cost function defined by $C(x) = \sum_i |x - a_i|$. The minimum of this function is at the median of the set $\{a_i\}$. To see this, note

that the derivative of $|x - a_i|$ with respect to x is $+1$ when $x > a_i$ and -1 when $x < a_i$. Thus, the derivative is zero when there are as many values of a_i less than x as there are greater than x. Thus, the cost is minimized at the median of the values a_i. Note that the median is immune to changes in the values of data a_i that lie far from the median. The value of the cost function changes, but not the position of its minimum.

For higher dimensional data $\mathbf{a}_i \in \mathbb{R}^n$, the minimum of the cost function $C(\mathbf{x}) = \sum_i \|\mathbf{x} - \mathbf{a}_i\|$ has similar stability properties. Note that $\|\mathbf{x} - \mathbf{a}_i\|$ is a *convex* function of x, and therefore so is a sum of such terms, $\sum_i \|\mathbf{x} - \mathbf{a}_i\|$. Consequently, this cost function has a single minimum (as do all convex functions).

The Huber cost function takes the form of a quadratic for small values of the error, δ, and becomes linear for values of δ beyond a given threshold. As such, it retains the outlier stability of the L_1 cost function, while for inliers it reflects the property that the squared-error cost function gives the Maximum Likelihood estimate.

The Pseudo-Huber cost function is also near-quadratic for small δ, and linear for large δ. Thus, it may be used as a smooth approximation to the Huber cost function, and gives similar results. It is important to note that each of these three cost functions has the very desirable property of being convex.

3.2.2 Performance of the different cost functions

To illustrate the properties of the different cost functions we will evaluate the cost $\sum_i C(x - a_i)$ for two synthetic example data sets $\{a_i\}$. Of the group of asymptotically linear cost functions, only the Huber cost function will be shown, since the other two (L_1 and pseudo-Huber) give very similar results.

The data $\{a_i\}$ may be thought of as the outcome of an experiment to measure some quantity, with repeated measurements. The measurements are subject to random Gaussian noise, with outliers. The purpose of the estimation process is to estimate the value of the quantity by minimizing a cost function. The experiments and the results for the two data sets are described in the captions of figure 3.9 and figure 3.10.

Summary of findings. The squared-error cost function is generally very susceptible to outliers, and may be regarded as unusable as long as outliers are present. If outliers have been thoroughly eradicated, using for instance RANSAC, then it may be used.

The non-convex cost functions, though generally having a stable minimum, not much effected by outliers have the significant disadvantage of having local minima, which can make convergence to a global minimum chancy. The estimate is not strongly attracted to the minimum from outside of its immediate neighbourhood.

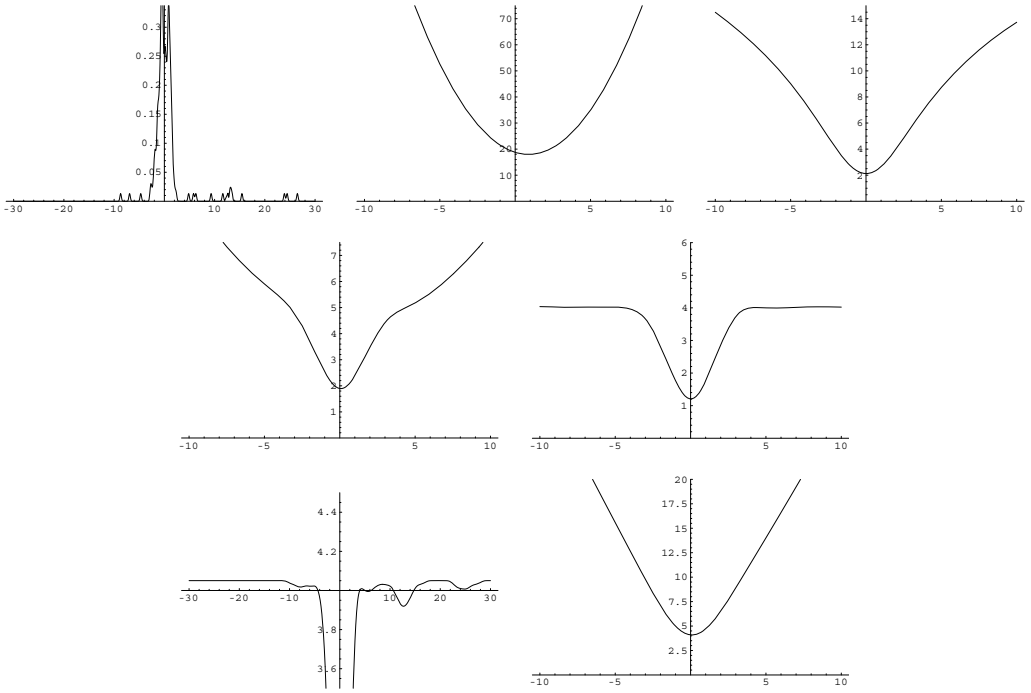

Fig. 3.9. *The data* $\{a_i\}$ *(illustrated in the top left graph) consists of a set of measurements centred around 0 with unit Gaussian noise, plus 10% of outliers biased towards the right of the true value. The graphs of* $\sum_i C(|x - a_i|)$ *correspond (left-to-right and top-to-bottom) to the cost functions Squared error, Cauchy, corrupted-Gaussian, Blake-Zisserman, a zoom of the Blake-Zisserman, and Huber cost functions. Note that the minimum of the squared-error cost function is pulled significantly to the right by the outliers, whereas the other cost-functions are relatively outlier-independent.*
The Blake-Zisserman cost function, which is based most nearly on the distribution of the data, has a very clear minimum. However, close examination (the zoomed plot) shows an undesirable characteristic, which is the presence of local minima near each of the outliers. An iterative method to find the cost minimum will fail if it is initiated outside the narrow basin of attraction surrounding the minimum.
By contrast, the Huber cost function is convex, which means the estimate will be drawn towards the single minimum from any initialization point.

Thus, they are not useful, unless (or until) the estimate is close to the final correct value.

The Huber cost function has the pleasant property of being convex, which makes convergence to a global minimum more reliable. The minimum is quite immune to the baleful influence of outliers since it represents a compromise between the Maximum Likelihood estimate of the inliers and the median of the outliers. The pseudo-Huber cost function is a good alternative to Huber, but use of L_1 should be approached with care, because of its non-differentiablity at the origin.

These findings were illustrated on one-dimensional data, but they carry over to higher dimensional data also.

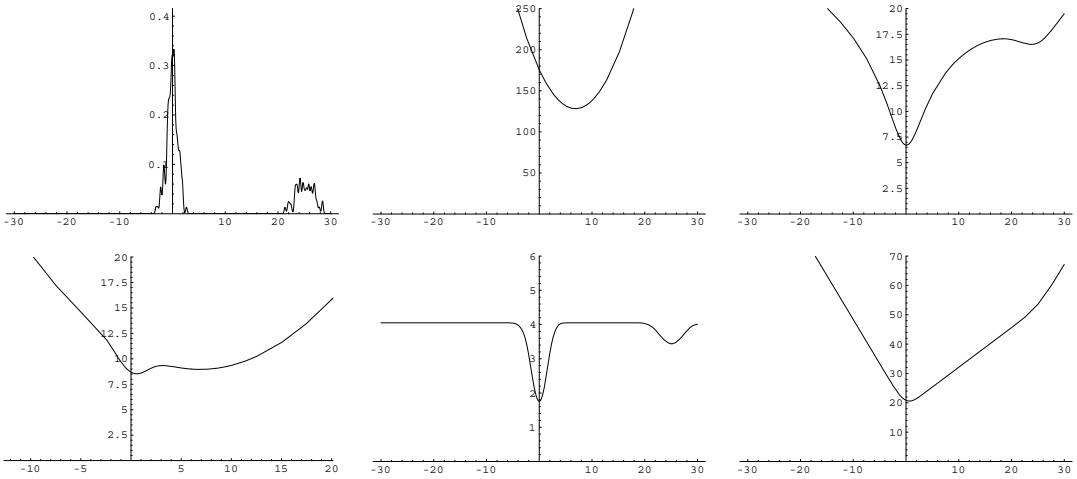

Fig. 3.10. *In this experiment, as in figure 3.9, the main part of the data (70%) is centred at the origin, with 30% of "outliers" concentrated in a block away from the origin (see top left graph). This type of measurement distribution is quite realistic in many imaging scenarios, for instance where point or edge measurement is confused by ghost edges. The cost functions in order from the top are: Square error, Cauchy, corrupted Gaussian, Blake-Zisserman and Huber.*
The Squared error cost function finds the mean of the measurement distribution, which is significantly pulled to the right by the block of outliers. The effect of the outlier block on the non-convex cost functions is also shown clearly here. Because of the non-convexity, the total cost function does not have a single minimum, but rather two minima around the separate blocks of measurements. Because of its convexity, the Huber cost function has a single minimum, which is located close to the median of the data, and is hardly influenced by the presence of the 30% of outliers.

Parameter minimization. We have seen that the Huber and related cost functions are convex, and hence have a single minimum. We refer here to the cost $C(\delta)$ as a function of the error δ. In general in problems such as structure from motion, the error δ itself is a non-linear function of the parameters (such as camera and point positions). For this reason, the total cost expressed as a function of the motion and structure parameters can not be expected to be convex, and local minima are inevitable. Nevertheless, an important principle is:

- *choose a parameterization in which the error is as close as possible to being a linear function of the parameters, at least locally.*

Observing this principle will lead to simpler cost surfaces with fewer local minima, and generally quicker convergence.

Attenuation factor. The purpose of the different cost functions considered here is to minimize the effect of outliers. To see how this is done, it is instructive

to compare the various cost functions with the squared-error cost function, which corresponds to a Gaussian noise model.

It turns out that all the cost functions considered here can be written in terms of a weighted sum-of-squares cost.

Thus, consider a cost function $f(\delta) = -\log(p(\delta)$. The Gaussian distribution corresponds to the squared-error cost function $f(\delta) = \delta^2$. Now, consider a different cost $g(\delta)$, one of the ones considered above, for instance. We can then write:

$$\begin{aligned} g(\delta) &= \left(\delta \frac{\sqrt{g(\delta)}}{\delta}\right)^2 \\ &= (w\,\delta)^2 \end{aligned}$$

where

$$w(\delta) = \frac{\sqrt{g(\delta)}}{\delta}$$

may be thought of as a weighting factor applied to δ. Then

$$\sum_{i=1}^{n} g(\delta_i) = \sum_{i=1}^{n} (w_i \delta_i)^2$$

and w_i is the weighte applied to the i-th measurement. Hence the cost function $\sum_i g(\delta_i)$ can be thought of as a weighted sum-of-squares cost function, where the weights themselves depend on the residual δ_i. The idea is that if δ is large (possibly an outlier), then it will be down-weighted accordingly.

Usually cost functions of the type we have discussed are used in the context of fitting a parametrized model. Commonly, the fitting procedure seek to minimize the norm of some vector $\boldsymbol{\Delta}$ depending on a set of parameters $\boldsymbol{\theta}$.

Since minimization of a squared vector norm $\|\boldsymbol{\Delta}\|^2$ is built into most implementations of Levenberg-Marquardt, we need to see how to apply the robust cost function in this case. The answer is to replace each vector $\boldsymbol{\delta}_i$ by a weighted vector $\boldsymbol{\delta}'_i = w_i \boldsymbol{\delta}_i$ such that

$$\|\boldsymbol{\delta}'_i\|^2 = w_i^2 \|\boldsymbol{\delta}_i\|^2 = C(\|\boldsymbol{\delta}_i\|)$$

for then $\sum_i C(\|\boldsymbol{\delta}_i\|) = \sum_i \|\boldsymbol{\delta}_i\|^2$ as desired. From this equality, we find

$$w_i = C(\|\boldsymbol{\delta}_i\|)^{1/2} / \|\boldsymbol{\delta}_i\|. \tag{3.14}$$

Thus, the minimization problem is to minimize $\|\boldsymbol{\Delta}'\|^2$ where $\boldsymbol{\Delta}'$ is the vector obtained by concatenating the vectors $\boldsymbol{\delta}'_i = w_i \boldsymbol{\delta}_i$, and each w_i is computed from (3.14). Note that w_i is a function of $\|\boldsymbol{\delta}_i\|$, which normally seeks to attenuate the cost of the outliers. This attenuation function is shown in the final column of figure 3.8 for the different cost functions. For the Squared error cost function, the attenuation factor is 1, meaning no attenuation occurs. For the other cost functions, there is little

attenuation within an inlier region, and points outside this region are attenuated to different degrees.

3.3 L_1 PCA

The standard PCA suffers from the usual defects of methods that minimize a sum-of-squares (otherwise known as L_2) error. They are extremely susceptible to outliers.

The solution to this is to use a form of L_1 PCA, where one fits a subspace to a set of data in a way so as to minimize the distance of the points from the subspace, n the L_1 sense. In other words, if a lower-dimensional subspace is chosen to closely fit a set of data, then the distances $d_i = d(x_i, S)$, the distance of the point from the surface, is measured. The loss (or cost) function is then

$$L(S) = \sum_{i=1}^{n} d(x_i, S) \ .$$

This differs from standard L_2 PCA in which $d(x_i, S)^2$ appears in the sum.

There are (at least) two ways to formulate an L_1 of PCA. To understand this, let's look at what normal PCA does.

Let $\{\mathbf{x}_i \ ; \ i = 1, \ldots, n\}$ be a set of points in $\mathbb{R}^D$. As a simplyifying assumption we can assume that the points are centred, in the sense that

$$\sum_{i=1}^{n} \mathbf{x}_i = \mathbf{0} \ .$$

The sum of their squares $\sum_{i=1}^{n} \|\mathbf{x}_i\|_2^2$ may be thought of as their *energy*.[1] Here, $\|\cdot\|_2$ is the 2-norm of a vector, namely the square root of sum of squares of its elements – see lecture 1.

Now, suppose that we wish to reduce the dimension of the points $\mathbf{x}_i$ by projecting (orthogonally) onto a subspace S of lower dimension d than the ambient dimension D. The subspace is to be chosen so that the total energy remaining in the projecte signal is maximized. So, if $\hat{\mathbf{x}}_i$ are the projected points, then the subspace S is chosen so that $\sum_{i=1}^{n} \|\hat{\mathbf{x}}_i\|_2^2$ is maximized.

This is the same thing as minimizing the sum-of-squared distances of the points $\mathbf{x}_i$ from the subspace, for if $\mathbf{x}_i^{\perp} = \mathbf{x}_i - \hat{\mathbf{x}}_i$, then $\mathbf{x}_i^{\perp}$ is orthogonal to the subspace (since $\hat{\mathbf{x}}_i$ is the orthogonal projection of $\mathbf{x}_i$). By Pythagoras' theorem, $\|\mathbf{x}_i\|_2^2 = \|\hat{\mathbf{x}}_i\|_2^2 + \|\mathbf{x}_i^{\perp}\|_2^2$. Summing over i gives

$$\sum_{i=1}^{n} \mathbf{x}_i^{\perp 2} = \sum_{i=1}^{n} \mathbf{x}_i^2 - \sum_{i=1}^{n} \hat{\mathbf{x}}_i^2$$

[1] This interpretation of the sum of squares as being the energy of the data is motivated by its meaning for signals. In particular, if the x_i come from a sampled signal, where x_i is the amplitude of the signal at evenly spaced moments of time t_i, then $\sum_{i=1}^{n} x_i^2$ is indeed a measure of the energy in the signal. In the case where they are continuous signals, the quantity $\int_I x(t)^2 \, dt$ measures the energy in a signal, or averaged over the length of the interval I, it is the power in the signal.

Therefore, minimizing $\sum_{i=1}^{n} \mathbf{x}_i^{\perp 2}$ is the same as maximizing $\sum_{i=1}^{n} \hat{\mathbf{x}}_i^2$.

L_1 PCA. Analogously, we may wish to project points $\mathbf{x}_i$ onto a subspace so as to maximize the L_1-energy of the projected points, namely

$$\sum_{i=1}^{n} \|\hat{\mathbf{x}}_i\|_2 \tag{3.15}$$

where $\|\hat{\mathbf{x}}_i\|_2$ is the Euclidean distance, in other words, the Euclidean 2-norm. Notice that this is the 1-norm of the vector consisting of the 2-norms of each $\hat{\mathbf{x}}_i$. For this reason (3.15) may be called the $(2,1)$-norm of the points $\hat{\mathbf{x}}_i$. Maximizing this quantity finds the subspace so that the projected data retains as much L_1-energy as possible.

An alternative is to try to minimize

$$\sum_{i=1}^{n} \|\mathbf{x}_i^{\perp}\|_2 \tag{3.16}$$

which is the $(2,1)$-norm, measuring the sum of distances of the $\mathbf{x}_i$ from the subspace S. **Note:** In this case, minimizing (3.16) is **not** the same thing as maximizing (3.15).

A further alternative is to maximize

$$\sum_{i=1}^{n} \|\hat{\mathbf{x}}_i\|_1 \tag{3.17}$$

which can be called the $(1,1)$ norm. Or else, we can minimize

$$\sum_{i=1}^{n} \|\mathbf{x}_i^{\perp}\|_1 \tag{3.18}$$

which can be called the $(1,1)$ norm. Finding the subspace S that optimizes (minimizes of maximizes, as appropriate) any one of the costs (3.15), (3.16) or (3.18) may reasonably be called L_1 PCA.

In the context of fitting data to points in a way that the fit is robust to outliers (wrong points), it makes best sense to minimize the L_1 norm of distances of points from S. There

Algorithms. There are different algorithms, depending on which version of L_1-PCA you want to implement.

For instance https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3746759/ gives an algorithm for minimizing (3.16). This is the problem that I think you should be solving.

Also, the Wikipedia page https://en.wikipedia.org/wiki/L1-norm_principal_component_analysis gives an algorithm to solve (maximize) (3.18). (Well, it actually does not solve this problem exactly.)

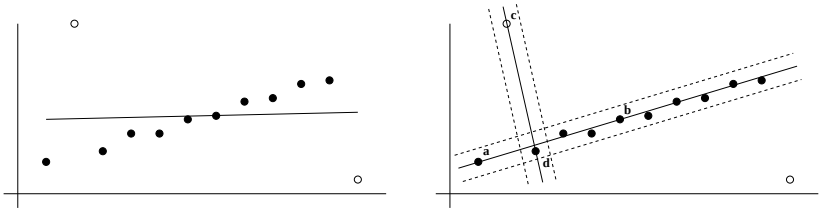

Fig. 3.11. **Robust line estimation.** *The solid points are inliers, the open points outliers. (a) A least-squares (orthogonal regression) fit to the point data is severely affected by the outliers. (b) In the RANSAC algorithm the support for lines through randomly selected point pairs is measured by the number of points within a threshold distance of the lines. The dotted lines indicate the threshold distance. For the lines shown the support is 10 for line* $\langle \mathbf{a}, \mathbf{b} \rangle$ *(where both of the points* $\mathbf{a}$ *and* $\mathbf{b}$ *are inliers); and 2 for line* $\langle \mathbf{c}, \mathbf{d} \rangle$ *where the point* $\mathbf{c}$ *is an outlier.*

3.4 RANSAC

Up to this point it has been assumed that we have been presented with a set of correspondences, $\{\mathbf{x}_i \leftrightarrow \mathbf{x}'_i\}$, where the only source of error is in the measurement of the point's position, which follows a Gaussian distribution. In many practical situations this assumption is not valid because points are mismatched. The mismatched points are *outliers* to the Gaussian error distribution. These outliers can severely disturb the estimated homography, and consequently should be identified. The goal then is to determine a set of *inliers* from the presented "correspondences" so that the homography can then be estimated in an optimal manner from these inliers using the algorithms described in the previous sections. This is *robust estimation* since the estimation is robust (tolerant) to outliers (measurements following a different, and possibly unmodelled, error distribution).

3.4.1 RANSAC

We start with a simple example that can easily be visualized – estimating a straight line fit to a set of 2-dimensional points. This can be thought of as estimating a 1-dimensional affine transformation, $x' = ax + b$, between corresponding points lying on two lines.

The problem, which is illustrated in figure 3.11a, is the following: given a set of 2D data points, find the line which minimizes the sum of squared perpendicular distances (orthogonal regression), subject to the condition that none of the valid points deviates from this line by more than t units. This is actually two problems: a line fit to the data; and a classification of the data into inliers (valid points) and outliers. The threshold t is set according to the measurement noise (for example $t = 3\sigma$), and is discussed below. There are many types of robust algorithms and which one to use depends to some extent on the proportion of outliers. For example, if it is known that there is only one outlier, then each point can be deleted in turn and the line estimated from the remainder. Here we describe in detail a general

and very successful robust estimator – the RANdom SAmple Consensus (RANSAC) algorithm of Fischler and Bolles [**?**]. The RANSAC algorithm is able to cope with a large proportion of outliers.

The idea is very simple: two of the points are selected randomly; these points define a line. The *support* for this line is measured by the number of points that lie within a distance threshold. This random selection is repeated a number of times and the line with most support is deemed the robust fit. The points within the threshold distance are the inliers (and constitute the eponymous *consensus* set). The intuition is that if one of the points is an outlier then the line will not gain much support, see figure 3.11b.

Furthermore, scoring a line by its support has the additional advantage of favouring better fits. For example, the line $\langle \mathbf{a}, \mathbf{b} \rangle$ in figure 3.11b has a support of 10, whereas the line $\langle \mathbf{a}, \mathbf{d} \rangle$, where the sample points are neighbours, has a support of only 4. Consequently, even though both samples contain no outliers, the line $\langle \mathbf{a}, \mathbf{b} \rangle$ will be selected.

More generally, we wish to fit a *model*, in this case a line, to data, and the random *sample* consists of a minimal subset of the data, in this case two points, sufficient to determine the model. If the model is a planar homography, and the data a set of 2D point correspondences, then the minimal subset consists of four correspondences. The application of RANSAC to the estimation of a homography is described below.

As stated by Fischler and Bolles [**?**] "The RANSAC procedure is opposite to that of conventional smoothing techniques: Rather than using as much of the data as possible to obtain an initial solution and then attempting to eliminate the invalid data points, RANSAC uses as small an initial data set as feasible and enlarges this set with consistent data when possible".

The RANSAC algorithm is summarized in algorithm 3.12. Three questions immediately arise:

1. What is the distance threshold? We would like to choose the distance threshold, t, such that with a probability α the point is an inlier. This calculation requires the probability distribution for the distance of an inlier from the model. In practice the distance threshold is usually chosen empirically. However, if it is assumed that the measurement error is Gaussian with zero mean and standard deviation σ, then a value for t may be computed. In this case the square of the point distance, $d_\perp^2$, is a sum of squared Gaussian variables and follows a χ_m^2 distribution with m degrees of freedom, where m equals the codimension of the model. For a line the codimension is 1 – only the perpendicular distance to the line is measured. If the model is a point the codimension is 2, and the square of the distance is the sum of squared x and y measurement errors. The probability that the value of a χ_m^2 random variable is less than k^2 is given by the cumulative chi-squared distribution, $F_m(k^2) = \int_0^{k^2} \chi_m^2(\xi) d\xi$. Both of these distributions are described in

Objective

Robust fit of a model to a data set S which contains outliers.

Algorithm

(i) Randomly select a sample of s data points from S and instantiate the model from this subset.
(ii) Determine the set of data points S_i which are within a distance threshold t of the model. The set S_i is the consensus set of the sample and defines the inliers of S.
(iii) If the size of S_i (the number of inliers) is greater than some threshold T, re-estimate the model using all the points in S_i and terminate.
(iv) If the size of S_i is less than T, select a new subset and repeat the above.
(v) After N trials the largest consensus set S_i is selected, and the model is re-estimated using all the points in the subset S_i.

Fig. 3.12. *The RANSAC robust estimation algorithm. A minimum of s data points are required to instantiate the free parameters of the model. The three algorithm thresholds t, T, and N are discussed in the text.*

??section **??** (*p* **??**). From the cumulative distribution

$$\begin{cases} \text{inlier} & d_\perp^2 < t^2 \\ \text{outlier} & d_\perp^2 \geq t^2 \end{cases} \quad \text{with } t^2 = F_m^{-1}(\alpha)\sigma^2. \tag{3.19}$$

Usually α is chosen as 0.95, so that there is a 95% probability that the point is an inlier. This means that an inlier will only be incorrectly rejected 5% of the time. Values of t for $\alpha = 0.95$ and for the models of interest in this book are tabulated in table 3.1.

Codimension m	Model	t^2
1	line, fundamental matrix	3.84 σ^2
2	homography, camera matrix	5.99 σ^2
3	trifocal tensor	7.81 σ^2

Table 3.1. *The distance threshold $t^2 = F_m^{-1}(\alpha)\sigma^2$ for a probability of $\alpha = 0.95$ that the point (correspondence) is an inlier.*

2. How many samples? It is often computationally infeasible and unnecessary to try every possible sample. Instead the number of samples N is chosen sufficiently high to ensure with a probability, p, that at least one of the random samples of s points is free from outliers. Usually p is chosen at 0.99. Suppose w is the probability that any selected data point is an inlier, and thus $\epsilon = 1 - w$ is the probability that it is an outlier. Then at least N selections (each of s points) are required, where

$(1-w^s)^N = 1-p$, so that

$$N = \log(1-p)/\log(1-(1-\epsilon)^s). \tag{3.20}$$

Table 3.2 gives examples of N for $p = 0.99$ for a given s and ϵ.

Sample size	Proportion of outliers ϵ						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Table 3.2. *The number N of samples required to ensure, with a probability $p = 0.99$, that at least one sample has no outliers for a given size of sample, s, and proportion of outliers, ϵ.*

Example 3.6. For the line-fitting problem of figure 3.11 there are $n = 12$ data points, of which two are outliers so that $\epsilon = 2/12 = 1/6$. From table 3.2 for a minimal subset of size $s = 2$, at least $N = 5$ samples are required. This should be compared with the cost of exhaustively trying every point pair, in which case $\binom{12}{2} = 66$ samples are required (the notation $\binom{n}{2}$ means the number of choices of 2 among n, specifically, $\binom{n}{2} = n(n-1)/2$). △

Note

(i) The number of samples is linked to the proportion rather than number of outliers. This means that the number of samples required may be smaller than the number of outliers. Consequently the computational cost of the sampling can be acceptable even when the number of outliers is large.

(ii) The number of samples increases with the size of the minimal subset (for a given ϵ and p). It might be thought that it would be advantageous to use more than the minimal subset, three or more points in the case of a line, because then a better estimate of the line would be obtained, and the measured support would more accurately reflect the true support. However, this possible advantage in measuring support is generally outweighed by the severe increase in computational cost incurred by the increase in the number of samples.

3. How large is an acceptable consensus set? A rule of thumb is to terminate if the size of the consensus set is similar to the number of inliers believed to be in the data set, given the assumed proportion of outliers, i.e. for n data points

$T = (1 - \epsilon)n$. For the line-fitting example of figure 3.11 a conservative estimate of ϵ is $\epsilon = 0.2$, so that $T = (1.0 - 0.2)12 = 10$.

Determining the number of samples adaptively. It is often the case that ϵ, the fraction of data consisting of outliers, is unknown. In such cases the algorithm is initialized using a worst case estimate of ϵ, and this estimate can then be updated as larger consistent sets are found. For example, if the worst case guess is $\epsilon = 0.5$ and a consensus set with 80% of the data is found as inliers, then the updated estimate is $\epsilon = 0.2$.

This idea of "probing" the data via the consensus sets can be applied repeatedly in order to adaptively determine the number of samples, N. To continue the example above, the worst case estimate of $\epsilon = 0.5$ determines an initial N according to (3.20). When a consensus set containing more than 50% of the data is found, we then know that there is at least that proportion of inliers. This updated estimate of ϵ determines a reduced N from (3.20). This update is repeated at each sample, and whenever a consensus set with ϵ lower than the current estimate is found, then N is again reduced. The algorithm terminates as soon as N samples have been performed. It may occur that a sample is found for which ϵ determines an N less than the number of samples that have already been performed. In such a case sufficient samples have been performed and the algorithm terminates. In pseudo-code the adaptive computation of N is summarized in algorithm 3.13.

- $N = \infty$, sample_count= 0.
- While $N >$ sample_count Repeat
  - Choose a sample and count the number of inliers.
  - Set $\epsilon = 1 -$ (number of inliers)/(total number of points)
  - Set N from ϵ and (3.20) with $p = 0.99$.
  - Increment the sample_count by 1.
- Terminate.

Fig. 3.13. *Adaptive algorithm for determining the number of RANSAC samples.*

This adaptive approach works very well and in practice covers the questions of both the number of samples and terminating the algorithm. The initial ϵ can be chosen as 1.0, in which case the initial N will be infinite. It is wise to use a conservative probability p such as 0.99 in (3.20). ??Table **??** (p **??**) on page **??** gives example ϵ's and N's when computing a homography.

3.4.2 Robust Maximum Likelihood estimation

The RANSAC algorithm partitions the data set into inliers (the largest consensus set) and outliers (the rest of the data set), and also delivers an estimate of the model, M_0, computed from the minimal set with greatest support. The final step

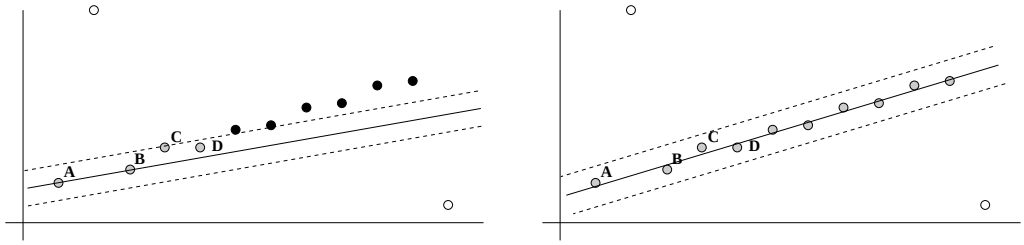

Fig. 3.14. **Robust ML estimation.** *The grey points are classified as inliers to the line. (a) A line defined by points* $\langle \mathbf{A}, \mathbf{B} \rangle$ *has a support of four (from points* $\{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$*). (b) The ML line fit (orthogonal least-squares) to the four points. This is a much improved fit over that defined by* $\langle \mathbf{A}, \mathbf{B} \rangle$*. 10 points are classified as inliers.*

of the RANSAC algorithm is to re-estimate the model using all the inliers. This re-estimation should be optimal and will involve minimizing a ML cost function, as described in ??section **??** (*p* **??**). In the case of a line, ML estimation is equivalent to orthogonal regression, and a closed form solution is available. In general, though, the ML estimation involves iterative minimization, and the minimal set estimate, M_0, provides the starting point.

The only drawback with this procedure, which is often the one adopted, is that the inlier–outlier classification is irrevocable. After the model has been optimally fitted to the consensus set, there may well be additional points which would now be classified as inliers if the distance threshold was applied to the new model. For example, suppose the line $\langle \mathbf{A}, \mathbf{B} \rangle$ in figure 3.14 was selected by RANSAC. This line has a support of four points, all inliers. After the optimal fit to these four points, there are now 10 points which would correctly be classified as inliers. These two steps: optimal fit to inliers; re-classify inliers using (3.19); can then be iterated until the number of inliers converges. A least-squares fit with inliers weighted by their distance to the model is often used at this stage.

Robust cost function. An alternative to minimizing $\mathcal{C} = \sum_i d_{\perp i}^2$ over the inliers is to minimize a robust version including all data. A suitable robust cost function is

$$\mathcal{D} = \sum_i \gamma\left(d_{\perp i}\right) \qquad \text{with} \quad \gamma(e) = \begin{cases} e^2 & e^2 < t^2 \quad \text{inlier} \\ t^2 & e^2 \geq t^2 \quad \text{outlier} \end{cases} \tag{3.21}$$

Here $d_{\perp i}$ are point errors and $\gamma(e)$ is a robust cost function [**?**] where outliers are given a fixed cost. The χ^2 motivation for the threshold is the same as that of (3.19), where t^2 is defined. The quadratic cost for inliers arises from the Gaussian error model, as described in ??section **??** (*p* **??**). The constant cost for outliers in the robust cost function arises from the assumption that outliers follow a diffuse or uniform distribution, the log-likelihood of which is a constant. It might be thought that outliers could be excluded from the cost function by simply thresholding on

$d_{\perp i}$. The problem with thresholding alone is that it would result in only outliers being included because they would incur no cost.

The cost function $\mathcal{D}$ allows the minimization to be conducted on all points whether they are outliers or inliers. At the start of the iterative minimization $\mathcal{D}$ differs from $\mathcal{C}$ only by a constant (given by 4 times the number of outliers). However, as the minimization progresses outliers can be redesignated inliers, and this typically occurs in practice. A discussion and comparison of cost functions is given in appendix 3.2 (*p* 57).

3.4.3 Other robust algorithms

In RANSAC a model instantiated from a minimal set is scored by the number of data points within a threshold distance. An alternative is to score the model by the median of the distances to all points in the data. The model with least median is then selected. This is Least Median of Squares (LMS) estimation, where, as in RANSAC, minimum size subset samples are selected randomly with the number of samples obtained from (3.20). The advantage of LMS is that it requires *no* setting of thresholds or *a priori* knowledge of the variance of the error. The disadvantage of LMS is that it fails if more than half the data is outlying, for then the median distance will be to an outlier. The solution is to use the proportion of outliers to determine the selection distance. For example if there are 50% outliers then a distance below the median value (the quartile say) should be used.

Both the RANSAC and LMS algorithms are able to cope with a large proportion of outliers. If the number of outliers is small, then other robust methods may well be more efficient. These include case deletion, where each point in turn is deleted and the model fitted to the remaining data; and iterative weighted least-squares, where a data point's influence on the fit is weighted inversely by its residual. Generally these methods are **not** recommended. Both Torr [**?**] and Xu and Zhang [**?**] describe and compare various robust estimators for estimating the fundamental matrix.

3.5 Automatic computation of a homography

This section describes an algorithm to automatically compute a homography between two images. The input to the algorithm is simply the images, with no other *a priori* information required; and the output is the estimated homography together with a set of interest points in correspondence. The algorithm might be applied, for example, to two images of a planar surface or two images acquired by rotating a camera about its centre.

The first step of the algorithm is to compute interest points in each image. We are then faced with a "chicken and egg" problem: once the correspondence between the interest points is established the homography can be computed; conversely, given the homography the correspondence between the interest points can easily be

Objective

Compute the 2D homography between two images.

Algorithm

(i) **Interest points:** Compute interest points in each image.
(ii) **Putative correspondences:** Compute a set of interest point matches based on proximity and similarity of their intensity neighbourhood.
(iii) **RANSAC robust estimation:** Repeat for N samples, where N is determined adaptively as in algorithm 3.13:
  (a) Select a random sample of 4 correspondences and compute the homography `H`.
  (b) Calculate the distance $d_\perp$ for each putative correspondence.
  (c) Compute the number of inliers consistent with `H` by the number of correspondences for which $d_\perp < t = \sqrt{5.99}\,\sigma$ pixels.

  Choose the `H` with the largest number of inliers. In the case of ties choose the solution that has the lowest standard deviation of inliers.
(iv) **Optimal estimation:** re-estimate `H` from all correspondences classified as inliers, by minimizing the ML cost function ??(**??** – p **??**) using the Levenberg–Marquardt algorithm of ??section **??** (p **??**).
(v) **Guided matching:** Further interest point correspondences are now determined using the estimated `H` to define a search region about the transferred point position.

The last two steps can be iterated until the number of correspondences is stable.

Fig. 3.15. *Automatic estimation of a homography between two images using RANSAC.*

established. This problem is resolved by using robust estimation, here RANSAC, as a "search engine". The idea is first to obtain by some means a set of putative point correspondences. It is expected that a proportion of these correspondences will in fact be mismatches. RANSAC is designed to deal with exactly this situation – estimate the homography and also a set of inliers consistent with this estimate (the true correspondences), and outliers (the mismatches).

The algorithm is summarized in algorithm 3.15, with an example of its use shown in figure 3.16, and the steps described in more detail below. Algorithms with essentially the same methodology enable the automatic computation of the fundamental matrix and trifocal tensor directly from image pairs and triplets respectively. This computation is described in ??chapter **??** and ??chapter **??**.

Determining putative correspondences. The aim, in the absence of any knowledge of the homography, is to provide an initial point correspondence set. A good proportion of these correspondences should be correct, but the aim is not perfect matching, since RANSAC will later be used to eliminate the mismatches. Think of these as "seed" correspondences. These putative correspondences are obtained by detecting interest points independently in each image, and then matching these interest points using a combination of proximity and similarity of intensity neigh-

bourhoods as follows. For brevity, the interest points will be referred to as 'corners'. However, these corners need not be images of physical corners in the scene. The corners are defined by a minimum of the image auto-correlation function.

For each corner at (x, y) in image 1 the match with highest neighbourhood cross-correlation in image 2 is selected within a square search region centred on (x, y). Symmetrically, for each corner in image 2 the match is sought in image 1. Occasionally there will be a conflict where a corner in one image is "claimed" by more than one corner in the other. In such cases a "winner takes all" scheme is applied and only the match with highest cross-correlation is retained.

A variation on the similarity measure is to use Squared Sum of intensity Differences (SSD) instead of (normalized) Cross-Correlation (CC). CC is invariant to the affine mapping of the intensity values (i.e. $I \mapsto \alpha I + \beta$, scaling plus offset) which often occurs in practice between images. SSD is not invariant to this mapping. However, SSD is often preferred when there is small variation in intensity between images, because it is a more sensitive measure than CC and is computationally cheaper.

RANSAC for a homography. The RANSAC algorithm is applied to the putative correspondence set to estimate the homography and the (inlier) correspondences which are consistent with this estimate. The sample size is four, since four correspondences determine a homography. The number of samples is set adaptively as the proportion of outliers is determined from each consensus set, as described in algorithm 3.13.

There are two issues: what is the "distance" in this case? and how should the samples be selected?

(i) **Distance measure**: The simplest method of assessing the error of a correspondence from a homography $\mathtt{H}$ is to use the symmetric transfer error, i.e. $d^2_{\text{transfer}} = d(\mathbf{x}, \mathtt{H}^{-1}\mathbf{x}')^2 + d(\mathbf{x}', \mathtt{H}\mathbf{x})^2$, where $\mathbf{x} \leftrightarrow \mathbf{x}'$ is the point correspondence. A better, though more expensive, distance measure is the reprojection error, $d^2_{\perp} = d(\mathbf{x}, \hat{\mathbf{x}})^2 + d(\mathbf{x}', \hat{\mathbf{x}}')^2$, where $\hat{\mathbf{x}}' = \mathtt{H}\hat{\mathbf{x}}$ is the perfect correspondence. This measure is more expensive because $\hat{\mathbf{x}}$ must also be computed. A further alternative is Sampson error.

(ii) **Sample selection**: There are two issues here. First, degenerate samples should be disregarded. For example, if three of the four points are collinear then a homography cannot be computed; second, the sample should consist of points with a good spatial distribution over the image. This is because of the extrapolation problem – an estimated homography will accurately map the region straddled by the computation points, but the accuracy generally deteriorates with distance from this region (think of four points in the very top corner of the image). Distributed spatial sampling can be implemented

by tiling the image and ensuring, by a suitable weighting of the random sampler, that samples with points lying in different tiles are the more likely.

Robust ML estimation and guided matching. The aim of this final stage is two-fold: first, to obtain an improved estimate of the homography by using all the inliers in the estimation (rather than only the four points of the sample); second, to obtain more inlying matches from the putative correspondence set because a more accurate homography is available. An improved estimate of the homography is then computed from the inliers by minimizing an ML cost function. This final stage can be implemented in two ways. One way is to carry out an ML estimation on the inliers, then recompute the inliers using the new estimated `H`, and repeat this cycle until the number of inliers converges. The ML cost function minimization is carried out using the Levenberg–Marquardt algorithm described in ??section **??**-(*p* **??**). The alternative is to estimate the homography and inliers simultaneously by minimizing a robust ML cost function of (3.21) as described in section 3.4.2. The disadvantage of the simultaneous approach is the computational effort incurred in the minimization of the cost function. For this reason the cycle approach is usually the more attractive.

3.6 Iteratively reweighted least-squares – IRLS

Iteratively reweighted least-squares optimization is a way of fitting models in cases where the noise η does not conform to a Gaussian model. We have in mind the kind of cost function, such as Huber, or L_1 estimation considered above.

IRLS is useful in cases where there is a simple least-squares solution (L_2 solution) to the optimization problem. The method consists of a series of weighted least-squares optimizations, followed by reweighting of the data point, according to the current residual for each point.

In section 3.2.2 it was pointed an "attenuation" factor was defined, which weights data points according to some changing weight factor, in order to down-weight measurements currently deemed to be outliers. This suggests an iterative procedure that alternates between estimating the model, and reweighting the data points according to the current residual. This is essentially the IRLS algorithm. It is known to converge to a desired cost minimum for a wide variety of costs.

For now, the following paper is included – it explains the method and proves convergence in many instances.

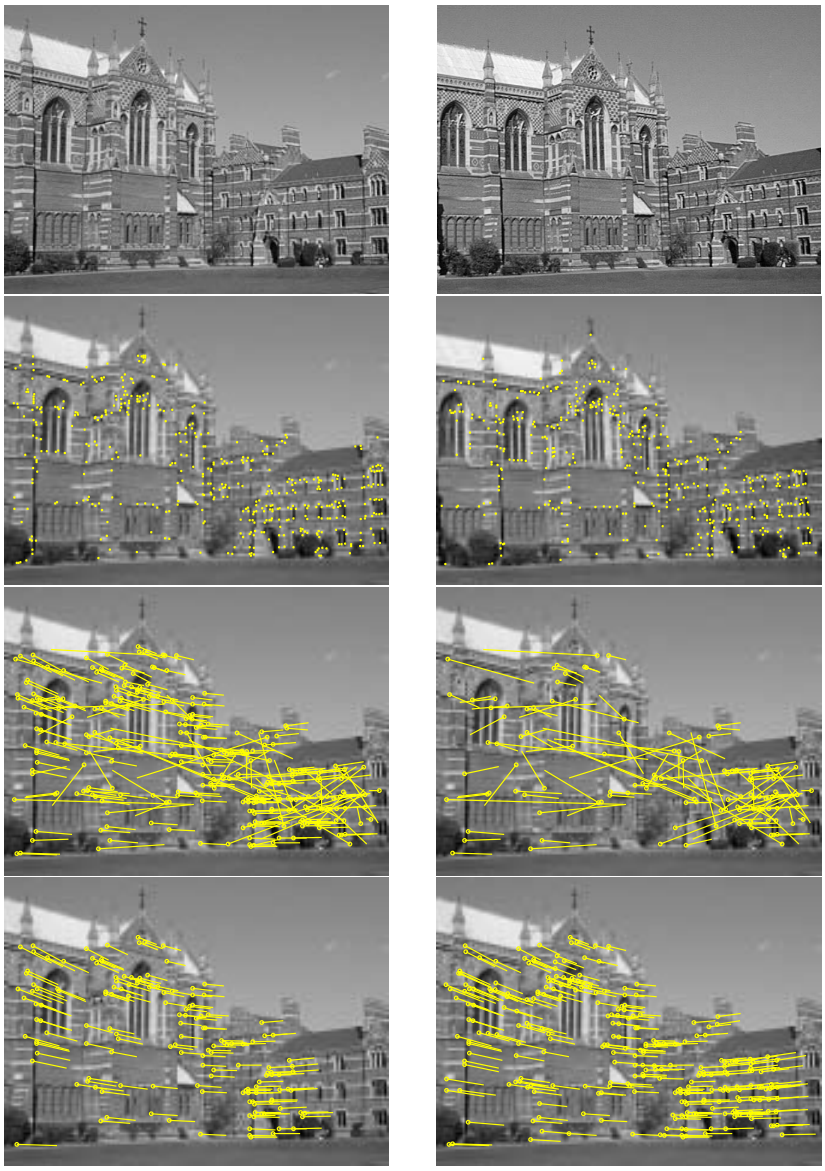

Fig. 3.16. **Automatic computation of a homography between two images using RANSAC.** *The motion between views is a rotation about the camera centre so the images are exactly related by a homography. Row 1: left and right images of Keble College, Oxford. The images are* 640×480 *pixels. Row 2: detected corners superimposed on the images. There are approximately 500 corners on each image. The following results are superimposed on the left image: Row 3, left: 268 putative matches shown by the line linking corners, note the clear mismatches; Row3, right: outliers – 117 of the putative matches; Row 4, left: inliers – 151 correspondences consistent with the estimated* `H`*; Row 4 right: final set of 262 correspondences after guided matching and MLE.*

4

Lecture-4: Clustering

4.1 Fitting Gaussians

4.1.1 Multidimensional Gaussians

A Gaussian distribution on the line is given by the formula

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

for $x \in \mathbb{R}$.

A Gaussian distribution on $\mathbb{R}^n$ is given by

$$g(\mathbf{x}) = \frac{1}{\sqrt{2\pi \det(\Sigma)}} \exp\left(-\frac{(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}}{2}\right) \tag{4.1}$$

Fig. 4.1. *Carl Friedrich Gauss (1777 - 1855)*

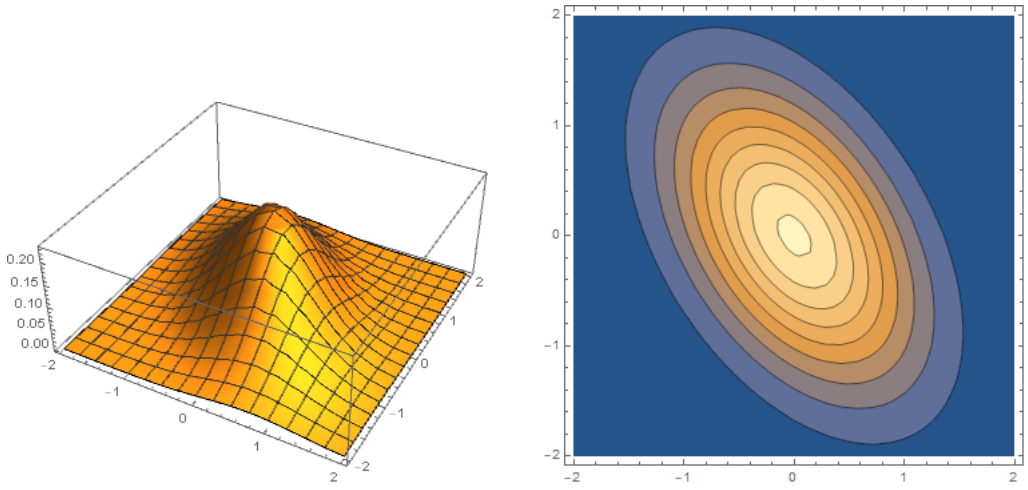

Fig. 4.2. *A 2D Gaussian distribution. On the left a 2D plot of a non-isotropic (i.e. non-circular) Gaussian. On the right its contour plot. As seen, a Gaussian is obtained from an isotropic Gaussian by stretching in n orthogonal directions.*

for $\mathbf{x} \in \mathbb{R}^n$. Here Σ is the *covariance matrix*, and $\boldsymbol{\mu}$ is the mean of the distribution.

See https://en.wikipedia.org/wiki/Multivariate_normal_distribution

Exercise: In the case of a 1-dimensional Gaussian, the covariance matrix is a 1×1 matrix, namely a real number, representing the variance of the distribution. See that the formula (4.1) corresponds to the definition, given previously for the Gaussian distribution.

The matrix Σ is the covariance matrix given by

$$\begin{aligned}\Sigma &= E((\mathbf{x}-\mu)(\mathbf{x}-\mu)^\mathsf{T}) \\ &= \int_{\mathbb{R}^n} g(\mathbf{x})(\mathbf{x}-\mu)(\mathbf{x}-\mu)^\mathsf{T}\, d\mathbf{x}\ .\end{aligned}$$

Since Σ is positive definite (prove) and symmetric, it can be written in its eigenvalue decomposition as $\Sigma = \mathtt{U}^\mathsf{T}\Lambda\mathtt{U}$. Then

$$\mathbf{x}^\mathsf{T}\Sigma^{-1}\mathbf{x} = (\mathtt{U}\mathbf{x})^\mathsf{T}\Lambda^{-1}(\mathtt{U}\mathbf{x})\ .$$

By a change of coordinates, $\mathbf{x}' \leftarrow \mathtt{U}\mathbf{x}$, this can be written as

$$\mathbf{x}'^\mathsf{T}\Lambda^{-1}\mathbf{x}' = \sum_{i=1}^{n} \lambda_i^{-1} x_i'^2\ .$$

Thus, the Gaussian is obtained by stretching an isotropic Gaussian by factors λ_i (the eigenvalues) in the orthogonal eigenvalue directions.

4.1.2 Fitting a Gaussian to data

.

Suppose that a set of points $\mathbf{x}_i \in \mathbb{R}^m,\ i = 1, \ldots, n$ are given, that we wish to fit to a Gaussian model. Denote the set of all points by $\mathbf{x}$.

Let $\theta = \{\boldsymbol{\mu}, \Sigma\}$ be a set of parameters for a multidimensional Gaussian. The problem considered is to find the maximum likelihood estimate of the Gaussian, given the data. That is, the set of parameters that will maximize

$$\begin{aligned} \log \mathcal{L}(\theta \mid \mathbf{x}) &= \log(P(\mathbf{x} \mid \theta)) \\ &= -(1/2) \sum_{i=1}^{n} (\mathbf{x}_i - \mu)^{\mathsf{T}} \Sigma^{-1} (\mathbf{x}_i - \mu) \ . \end{aligned}$$

It may be seen (similarly to the calculation given for $1D$ Gaussians), that the ML solution is given by

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \\ \Sigma &= \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^{\mathsf{T}} \ . \end{aligned} \tag{4.2}$$

Exercise. Verify this.

Weighted data. Change this problem a little to a case where each data point $\mathbf{x}_i$ comes with a weight w_i. The problem is now to find the ML estimate of parameters that minimizes a weighted negative log-likelihood. We think of a weight w_i as meaning that the data point $\mathbf{x}_i$ appears w_i times. This makes good sense if w_i is an integer, but we assume it makes good sense if w_i is fractional also.

Thus, we wish to minimize

$$\begin{aligned} \log \mathcal{L}(\theta \mid \mathbf{x}) &= \log(P(\mathbf{x} \mid \theta)) \\ &= -(1/2) \sum_{i=1}^{n} w_i (\mathbf{x}_i - \mu)^{\mathsf{T}} \Sigma^{-1} (\mathbf{x}_i - \mu) \ . \end{aligned}$$

The formula (4.2) is then generalized to

$$\begin{aligned} \boldsymbol{\mu} &= \frac{\sum_{i=1}^{n} w_i \mathbf{x}_i}{\sum_{i=1}^{n} w_i} \\ \Sigma &= \frac{\sum_{i=1}^{n} w_i (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^{\mathsf{T}}}{\sum_{i=1}^{n} w_i} \end{aligned} \tag{4.3}$$

A natural way to weight data is as follows. Suppose that there are K different classes, each with its own probability distribution $p_k(\mathbf{x})$. Let w_{ik} denote the probability that data $\mathbf{x}$ belongs to class k. In other words, set

$$w_{ik} = T_{ki} = P(Z = k \mid X = \mathbf{x}_i) \ .$$

Then, we can compute set of parameters for probability distribution p_k by weighting

each point $\mathbf{x}$ by the probability that it belongs to the class k. Then,

$$\begin{aligned} \boldsymbol{\mu}_k &= \frac{\sum_{i=1}^n T_{ki}\mathbf{x}_i}{\sum_{i=1}^n T_{ki}} \\ \Sigma_k &= \frac{\sum_{i=1}^n T_{ki}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^\mathsf{T}}{\sum_{i=1}^n T_{ki}} \end{aligned} \tag{4.4}$$

4.2 Model fitting

We have talked about ML model fitting in the context of fitting a model with probability $p(x)$. Thus, given data $\mathbf{x}_i$, the task is to find the model that maximizes the probability of the data, given the model:

$$P(\mathbf{x} \mid \theta) = \prod_{i=1}^n p_\theta(\mathbf{x}_i) \ .$$

This assumes that the data is iid.

It is simple to consider the negative log-probability of the data, given the model, otherwise known as the negative log-likelihood of the model.

$$-\log \mathcal{L}(\theta \mid \mathbf{x}) = -\log P(\mathbf{x} \mid \theta) = -\sum_{i=1}^n \log p_\theta(\mathbf{x}_i) \ .$$

Another way of thinking of this is that the data $\mathbf{x}$ is sampled from some probability distribution itself, call this $q(\mathbf{x})$. In this case, the finite sums can be thought of as approximations to the expected value over taking many samples of data.

Thus, we can speak of the *expected* negative-log likelihood of the model

$$E_q(\log p_\theta(\mathbf{x})) = -\int q(\mathbf{x}) \log(p_\theta(\mathbf{x}))\, dx$$

Thus, the model fitting task is defined as choosing the model that has the maximum expected log-likelihood.

4.3 K-means algorithm

The K-means algorithm is a simple method of clustering in which data in $\mathbb{R}^m$ is clustered into several different (K) classes.

Let $\mathcal{L} = \{1, 2, \ldots, K\}$ be labels for the K different classes. The algorithm starts with a set of n data points $\mathbf{x}_i \in \mathbb{R}^m$ and the task is to assign a label $z_i \in \mathcal{L}$ to each data point $\mathbf{x}_i$.

It is a very simple alternating algorithm which models clusters by their cluster centres. The algorithm starts with K cluster centres, $\mu_1, \ldots, \mu_k$ in $\mathbb{R}^m$, initially chosen at random. It finds class assignments $z_i \in \mathcal{L}$. Then it alternates between the following steps until convergence:

(i) **Assignment of data to clusters.** Assign each data element $\mathbf{x}_i$ to class k whose cluster centre is closest.

$$z_i = \arg\min_k \|\mathbf{x}_i - \mu_k\|$$

(ii) **Redefine cluster centres.** Redefine each cluster centre μ_k as the centroid of the data points $\mathbf{x}_i$ that are labelled with $z_i = k$.

$$\mu_k = \frac{\sum_{i=1}^n \mathbf{x}_i T_{ki}}{\sum_{i=1}^n T_{ki}} \ .$$

where T_{ki} is an indicator function for data item $\mathbf{x}_i$ being in class k. Thus $T_{ki} = 1$ if $z_i = k$, and 0 otherwise.

4.4 The EM algorithm

This is described in https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm. There are lots of interesting animations of the algorithm. For instance, the animation from https://commons.wikimedia.org/w/index.php?curid=20494862. However, the page is not easy to understand in general.

Criticism. In my view, part of the confusion in reading the Wikipedia page derives from the fact that given data samples $\mathbf{x}_i$, it defines separate random variables X_i and hidden variables Z_i for each sample. In my view, this is **wrong**. It causes confusion with expressions such as $P(X_i = \mathbf{x}_i)$ or the like. Since $\mathbf{x}_i$ is a measurement, one is tempted to say that $P(X_i = \mathbf{x}_i)$ is 1, the i-th measurement **is** $\mathbf{x}_i$. On the contrary, there is a single random variable X operating here, not separate X_i. Similarly, there is a single random variable Z operating, and $\mathbf{x}_i$ and z_i are just instances of these random variables. In this case, expressions such as $P(X = \mathbf{x}_i)$ make a lot more sense, representing the probability that a given measurement $\mathbf{x}_i$ occurs. This can be expressed in terms of the probability distribution p_X of the random variable X. The individual measurements $\mathbf{x}_i$ can be considered as instances of the random variable X.

The EM algorithm may be seen as a version of k-means where instead of assigning measurements $\mathbf{x}_i$ to specific classes in a hard manner, we use the probability that a measurement is assigned to a given class. Thus, with K classes $\mathcal{L} = \{1, 2, \ldots, K\}$ any measurement $\mathbf{x}$ is assigned to one of the k classes.

Another way of thinking of the EM algorithm is that it is a model-fitting algorithm, fitting a set of data, to a probability "mixture model," namely a mixture of different probability distributions p_k. As is usual with model fitting, the EM algorithm minimizes the expected log-likelihood of the mixture model.

4.4.1 Mixture models.

https://en.wikipedia.org/wiki/Mixture_model

Sometimes, data is drawn from different populations, each with its own statistics. It is not always known which population a given sample is drawn from. This leads to the concept of a mixture model.

Suppose $k \in \mathcal{L} = \{1, \ldots, K\}$, a finite set of "classes". Associated with each class k is a probability distribution denoted $p_k(\mathbf{x})$ or $p_{\theta_k}(\mathbf{x})$, defined, say, on $\mathbb{R}^m$ – hence $\mathbf{x} \in \mathbb{R}^m$. Further, let τ_k represent the prior probability of a data element belonging to class k. These values τ_k are called the *class membership probabilities*. Of course,

$$\sum_{k \in \mathcal{L}} \tau_k = 1 \ ,$$

assuming that each data element belongs to exactly one class.

As an example, let the two classes be men and women, $k = 1$ and 2 and suppose there is a group of subjects consisting of n_1 men and n_2 women. Then, the probability of a person, selected at random, belonging to class k is given by

$$\tau_k = \frac{n_k}{\sum_{j=1}^{K} n_j} \ .$$

Suppose that their heights and weights are measured, consisting of a pair $\mathbf{x} = (h, w)$ of measurements, which is an instance of a random variable $X = (H, W)$. For each of the two classes, there is a probability distribution $p_k(\mathbf{x})$.

There is a further random variable, denoted Z, representing the class, and having values in the finite set $\mathcal{L}$. The pair X and Z represent a joint probability distribution – each subject instance i instantiates each of these random variables giving a pair $(X = \mathbf{x}_i, Z = z_i)$. In the height/weight example, X itself consists of two variables H and W (height and weight), so the i-th subject corresponds to an instantiation $(X = (h_i, w_i), Z = \text{gender}_i)$.

This situation defines a mixture-type probability distribution, having parameters $\{\tau_k, \theta_k; k = 1, \ldots, K\}$ consisting of the class probabilities τ_k along with the distribution probabilities θ_k for each class.

The mixture probability distribution. It is easy to compute the probability of a given measurement $\mathbf{x}$, using Bayes' Rule. Denoting by $p_M(\mathbf{x})$ the mixture probability distribution,

$$\begin{aligned} p_M(\mathbf{x}) = P(X = \mathbf{x}) &= \sum_{k \in \mathcal{L}} P(X = \mathbf{x} \mid Z = k) P(Z = k) \\ &= \sum_{k \in \mathcal{L}} p_k(\mathbf{x}) \tau_k \end{aligned} \tag{4.5}$$

The point of this calculation is to show that the probability distribution $P(X = \mathbf{x})$ can be computed from the class membership probabilities τ_k and the individual

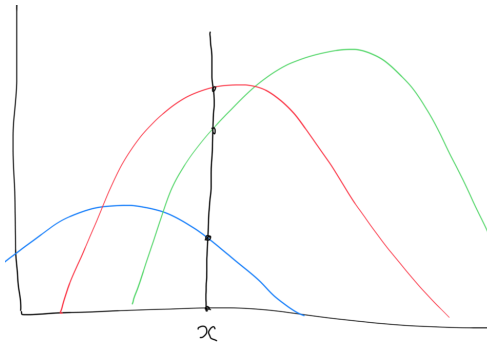

Fig. 4.3. *Given histograms of occurrences of the data in each class, one can find the probability that a data point x lies in a given class k. This is given by $P(Z=k) = \frac{H_k(x)}{\sum_{k=1}^{K} H_k(x)}$ where $H_k(x) = \tau_k p_k(x)$ and τ_k is the expected number of samples in class k.*

class distributions $p_k(\mathbf{x})$. Otherwise stated, the distributions p_k along with the class probabilities τ_k define the mixture probability distribution.

Conditional probabilities. The probability distribution for each class is given by $p_k(\mathbf{x})$. By definition, this is

$$P(X = \mathbf{x} \mid Z = k) = p_k(\mathbf{x}) \ .$$

The reverse conditional probability may be calculted using Bayes' Rule. Denoting $P(Z = k \mid X = x)$ by $T_k(\mathbf{x})$, we have

$$\begin{aligned} T_k(\mathbf{x}) &= \frac{P(X = \mathbf{x} \mid Z = k)\, P(Z = k)}{P(X = \mathbf{x})} \\ &= \frac{p_k(\mathbf{x})\, \tau_k}{\sum_{k \in \mathcal{L}} p_k(\mathbf{x})\, \tau_k} \end{aligned} \tag{4.6}$$

where we used (4.5). See figure 4.3.

Formula for membership probabilities. Conversely, it is also possible to compute the class membership probabilities, knowing the mixture probability $P(X = x)$ and the conditional probabilities $P(Z = k \mid X = \mathbf{x})$. Indeed

$$\tau_k = P(Z = k) = \int_{\mathbf{x}} P(Z = k \mid X = \mathbf{x}) P(X = \mathbf{x}) = \int_{\mathbf{x}} T_k(\mathbf{x}) P(X = \mathbf{x})\, dx \ .$$

This last integral is the expected value of $T_k(\mathbf{x})$ under the distribution $P(X = \mathbf{x})$, and it may be evaluated empirically as the sum

$$\tau_k = E(T_k(\mathbf{x})) = \frac{1}{n} \sum_{i=1}^{n} T_k(\mathbf{x}_i) \ , \tag{4.7}$$

which may be thought of as a measure of the number of measurements in class k.

This is a little round-about, since to compute $T_k(\mathbf{x})$, we need to know τ_k, but

τ_k is defined in terms of $T_k(\mathbf{x})$. The resolution to this is that at each step in an alternating algorithm, τ_k^{t+1} is computed in terms of $T_k(\mathbf{x})^t$, the values at iterations $t+1$ and t.

In other words: **if the parameters of the probability distributions** $p_k(\mathbf{x})$ **are known** then we can compute the probability of a data point $\mathbf{x}$ belonging to any class k.

Thus, given the assignments T_{ki} of data $\mathbf{x}_i$ to cluster k, it is possible to find the parameters $\theta_k = (\mu_k, \Sigma_k)$ of each of the Gaussian probability distributions $p_k(\mathbf{x})$ that maximize the log-likelihood of the model given by parameters $\theta = \{\theta_k\}$.

4.4.2 Steps of the EM algorithm

From (4.6) it is seen that the conditional class membership probabilities $T_k(\mathbf{x}_i)$ of data elements $\mathbf{x}_i$ can be computed if $p_k(\mathbf{x})$ and τ_k are known. On the other hand, from (4.7) and (4.4) , τ_k and p_k can be computed as long as $T_k(\mathbf{x}_i)$ is known.

This suggests an iterative algorithm, with successive estimates of these parameters. Indicating by a superscript (t) the estimates of the parameters at time step t, the iteration can be denoted by

$$\{\tau_k^{(t)}, \theta_k^{(t)}\} \stackrel{E}{\Longrightarrow} \{T_k(\mathbf{x}_i)^{(t)}\}^{(t)} \stackrel{M}{\Longrightarrow} \{\tau_k^{(t+1)}, \theta_k^{(t+1)}\} \ ,$$

where θ_k represents the parameters of the distribution $p_k(\mathbf{x}) = p_{\theta_k}(\mathbf{x})$. In the case of Gaussian distributions, these are the mean μ_k and the covariance Σ_k.

The starting point can be a set of randomly chosen probability distributions and class membership probabilities. The number of classes K is also assumed known. Thus, for each k, the parameters $\theta_k = (\mu_k, \Sigma_k)$ are initialized, and τ_k is initialized to $1/K$. Then,

(i) **E-step.** Compute the conditional class probabilities T_{ki}. From (4.6):

$$T_k(\mathbf{x}_i) = \frac{p_k(\mathbf{x}_i)\,\tau_k}{\sum_{k\in\mathcal{L}} p_k(\mathbf{x}_i)\,\tau_k} \tag{4.8}$$

(ii) **M-step.** Compute the parameters θ_k of each distribution and the class membership probabilities. These are the ML estimation of the parameters of the mixture model, given $T_k(\mathbf{x}_k)$. From (4.7):

$$\tau_k = \frac{1}{n}\sum_{i=1}^{n} T_k(\mathbf{x}_i) \ , \tag{4.9}$$

and assuming multi-dimensional Gaussian distributions p_k, (4.4) gives

$$\begin{aligned} \boldsymbol{\mu}_k &= \frac{\sum_{i=1}^n T_k(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^n T_k(\mathbf{x}_i)} \\ \Sigma_k &= \frac{\sum_{i=1}^n T_k(\mathbf{x}_i)(\mathbf{x}_i-\mu_k)(\mathbf{x}_i-\mu_k)^\mathsf{T}}{\sum_{i=1}^n T_k(\mathbf{x}_i)} \end{aligned} \tag{4.10}$$

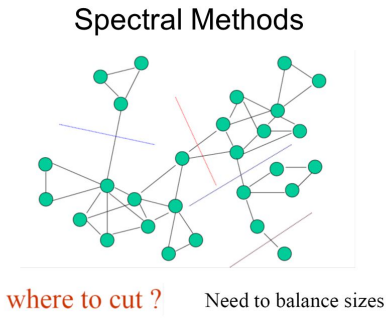

Fig. 4.4. *Similarity graph from* https://slideplayer.com/slide/5008353/

4.5 Spectral clustering

Spectral clustering is a way of clustering data elements into different classes based on "similarity measures."

A good reference for spectral clustering is https://arxiv.org/abs/0711.0189, the paper "A tutorial on spectral clustering" by Ulrike von Luxburg.

The steps of spectral clustering are:

(i) Form a similarity graph, which has weights w_{ij}.
(ii) Form the (sparse – usually) similarlty matrix in which $\mathtt{W}_{ij} = w_{ij}$.
(iii) Form its graph Laplacian.

$$\mathtt{L} = \mathtt{D} - \mathtt{W}$$

(iv) Find the k smallest eigenvalues of $\mathtt{L}$, and their corresponding eigenvalues.
(v) Arrange the eigenvectors in an $n \times k$ matrix $\mathtt{Y}$. The rows of this matrix may be thought of as n different k-vectors. Hence, this provides a projection of the data into a k-dimensional space $\mathbb{R}^k$. (A bit similar to PCA.) Let $\mathbf{y}_i$ be the i-th row of $\mathtt{Y}$.
(vi) Cluster the n vectors $\mathbf{y}_i$ into classes using k-means.

Fig. 4.5. *Pierre-Simon Marquis de Laplace. (1749 – 1827). Author of "Méchanique Céleste". "I had no need of that hypothesis." (Images from Wikipedia.)*

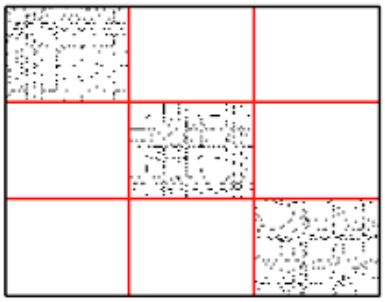

Fig. 4.6. *If data lies in disconnected blocks, then the similarity graph breaks into blocks.*

4.5.1 Why does this work?

The reasoning behind subspace clustering is as follows.

(i) The vector $\mathbf{1} = (1, 1, \ldots, 1)$ is an eigenvector of the Laplacian, with eigenvalue 0. This is because of the definition of the Laplacian $\mathtt{L} = \mathtt{D} - \mathtt{W}$, so $\mathtt{W}\mathbf{1} = \mathtt{D}\mathbf{1}$. Hence

$$\mathtt{L}\mathbf{1} = \mathtt{D}\mathbf{1} - \mathtt{W}\mathbf{1} = \mathbf{0} \ .$$

(ii) If the graph has k disjoint components, then it has k distinct and orthogonal eigenvectors, corresponding to indicator vectors $\mathbf{1}_k$ which is 1 for those data i in cluster k.

(iii) Therefore, in spectral clustering, data $\mathbf{x}_i$ in cluster j will be represented by the k-vector $(0, 0, 1, 0, 0)$ where the 1 appears in the j-th position. Hence, they are easily separated by k-means.

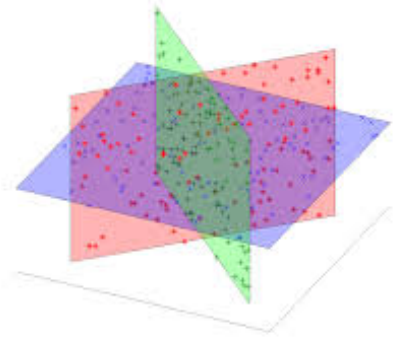

Fig. 4.7. *Data lying on different subspaces.*

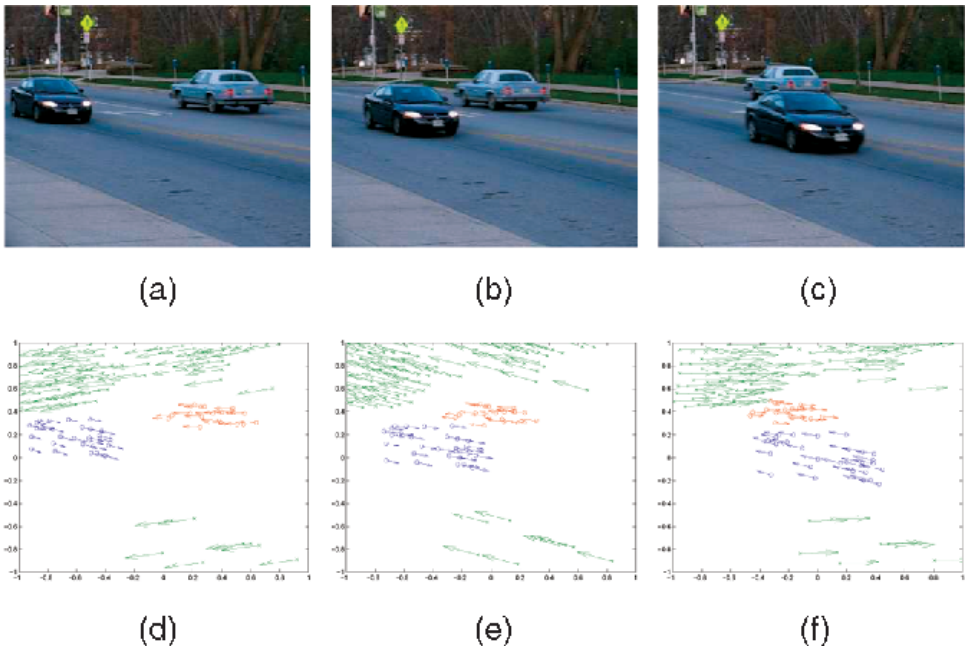

Fig. 4.8. *Segmentation of separately moving objects using subspace clustering.*

4.5.2 Subspace clustering

Data in $\mathbb{R}^n$ may lie on a set of hyperplanes. See book by Rene Vidal.

Application: multibody motion segmentation. This method has been used in computer vision for lots of things. In particular, multi-body segmentation of videos.

4.5.3 Normalized Cuts

This is a method used in Computer Vision (Shi and Malik). https://scholar.google.com.au/citations?hl=en&user=Sm14jYIAAAAJ

It uses a slightly different normalized version of the Laplacian, the normalized laplacian. See the tutorial cited above: https://arxiv.org/abs/0711.0189.

5

Lecture-5: Dimensionality Reduction

5.1 Recap of PCA

Principal Component Analysis (PCA) is an analogy to orthogonal regression, except that we wish to fit an affine subspace of lower dimension than n to some points $\mathbf{y}_i$ in $\mathbb{R}^n$.

(i) Translate the points so that their centroid is at the origin (subtract $\bar{\mathbf{y}} = (1/k)\sum_{i=1}^{k} \mathbf{y}_i$ from all points.

(ii) Form the scatter matrix $\mathtt{Y}$.

(iii) Let $\mathtt{A}$ be the matrix consisting of the eigenvectors corresponding to the m largest eigenvalues of $\mathtt{Y}$. Thus, let $\mathtt{Y} = \mathtt{U}^{\mathsf{T}}\Lambda\mathtt{U}$. Then $\mathtt{A}$ consists of the first r columns of $\mathtt{U}$.

(iv) The best hyperplane fit to the points $\{\mathbf{y}_i\}$ is the affine subspace $\mathtt{Y}\mathbf{x} + \bar{\mathbf{y}}$, for $\mathbf{x} \in \mathbb{R}^m$, namely the space that is spanned by the columns of $\mathtt{Y}$.

5.1.1 Eigenfaces.

https://en.wikipedia.org/wiki/Eigenface A use of PCA for face recognition.

Look on the web and find out how Eigenfaces works.

5.1.2 Forensic Face Reconstruction.

The following paper shows an example of use of PCA.

http://scholar.google.com.au/scholar?q=Face+reconstruction+using+flesh+deformation+modes&hl=en&as_sdt=0&as_vis=1&oi=scholart

5.2 Linear discriminant analysis

https://en.wikipedia.org/wiki/Linear_discriminant_analysis

5.3 Logistic Regression

https://en.wikipedia.org/wiki/Logistic_regression

5.4 Multidimensional Scaling (MDS)

https://en.wikipedia.org/wiki/Multidimensional_scaling

We talk about classical MDS.

The purpose of MDS is to embed a set of n points $\mathbf{x}_i \in \mathbb{R}^m$ into a lower dimensional space while preserving (as much as possible) their distances. Thus, we find points $\mathbf{y}_i \in \mathbb{R}^n$ with $k < m$ such that

$$\|\mathbf{y}_i - \mathbf{y}_j\| = \|\mathbf{x}_i - \mathbf{x}_j\|$$

as closely as possible, for all i, j. Of course, if the points $\mathbf{x}_i$ already lie in an affine sub-space (generalization of a plane or straight line) of dimension k in $\mathbb{R}^m$, then this will be an exact equality.

The steps of the algorithm are

(i) Form the matrix $\mathtt{D}$ with entries $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$. This is a square symmetric matrix of dimension $n \times n$. Alternatively, we simply start with a matrix D giving desired distances between points.

(ii) Multiply by the centering matrix $\mathtt{H} = \mathtt{I} - \mathbf{1}\mathbf{1}^\mathsf{T}/n$. This matrix has the effect of centering the data, as will be explained below. https://en.wikipedia.org/wiki/Centering_matrix.

In particular, let

$$\mathtt{B} = -(1/2)\mathtt{HDH} \ .$$

Since $\mathtt{D}$ has been multiplied by $\mathtt{H}$ on both sides, it is said to be *double-centred*.

(iii) Take the eigenvalue expansion of $\mathtt{B}$. Since $\mathtt{B}$ is symmetric, it has real eigenvalues and orthogonal eigenvectors. So we can write

$$\mathtt{B} = \mathtt{U}\Lambda\mathtt{U}^\mathsf{T}$$

where $\mathtt{U}$ is an orthogonal matrix. we now write

$$\mathtt{X} = \mathtt{U}\Lambda^{1/2}$$

so that

$$\mathtt{B} = \mathtt{X}\mathtt{X}^\mathsf{T}$$

Note that if some eigenvector $\boldsymbol{\lambda}$ are very small, then the corresponding rows of $\mathtt{X}$ are very small. What if we simply ignore these small rows, and write $\mathtt{X}_k$ to be the (tall thin) matrix consisting of the first k columns of the matrix $\mathtt{X}$?

(iv) The rows of $\mathtt{X}_k$ (which are k-vectors) are the desired points $\mathbf{y}_i$.

5.4.1 Justification and explanation

We first explain what the double-centering operation is all about.

The centering matrix H. The centering matrix is defined as $\mathtt{H} = \mathtt{I} - \mathbf{1}\mathbf{1}^\mathsf{T}/n$, where n is the dimension of H and $\mathbf{1}$ is the vector of all 1 entries, and hence $\mathbf{1}\mathbf{1}^\mathsf{T}$ is the matrix with all entries equal to 1.

The main property of the centering matrix is that it centres a set of vectors.

Lemma 5.7. *If* X *is a matrix of dimension $m \times n$ with columns equal to $\mathbf{x}_i$, then*

$$\mathtt{X}_c = \mathtt{X}(\mathtt{I} - \mathbf{1}\mathbf{1}^\mathsf{T}/n) = \mathtt{XH}$$

is the matrix with columns $\mathbf{x}_i - \bar{\mathbf{x}}$, where $\bar{\mathbf{x}}$ is mean of the vectors $\mathbf{x}_i; i = 1, \ldots, m$.

The proof if this is simple, and left as an exercise.

Another useful property of the centering matrix is:

$$\mathtt{H}\mathbf{1} = (\mathtt{I} - \mathbf{1}\mathbf{1}^\mathsf{T}/n)\mathbf{1} = \mathbf{0}\ , \tag{5.1}$$

which says (in other words) that the vector $\mathbf{1}$ is an eigenvector of the matrix $\mathtt{H} = \mathtt{I} - \mathbf{1}\mathbf{1}^\mathsf{T}/n$ with zero-eigenvalue.

Double centering. So what does double-centering do?

Suppose that we are given n points $\mathbf{x}_i$ in a Euclidean space $\mathbb{R}^k$. Let D be the $n \times n$ matrix whose (i,j)-th element is equal to $\|\mathbf{x}_i - \mathbf{x}_j\|^2$, the squared Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Typically, we do not know the points $\mathbf{x}_i$, just their squared-distance matrix D.

Let X be the $k \times n$ matrix whose columns are the vectors $\mathbf{x}_i$. As before, we do not necessarily know what this matrix X is.

We start with a simple computation:

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \langle \mathbf{x}_i - \mathbf{x}_j, \mathbf{x}_i - \mathbf{x}_j \rangle = \langle \mathbf{x}_i, \mathbf{x}_i \rangle + \langle \mathbf{x}_j, \mathbf{x}_j \rangle - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle\ .$$

The matrix D with entries $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ may then be written as

$$\mathtt{D} = -2 \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \ldots & \langle \mathbf{x}_1, \mathbf{x}_n \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_1 \rangle & \ldots & \langle \mathbf{x}_n, \mathbf{x}_n \rangle \end{bmatrix} + \begin{bmatrix} \|\mathbf{x}_1\|^2 & \ldots & \|\mathbf{x}_1\|^2 \rangle \\ \vdots & \ddots & \vdots \\ \|\mathbf{x}_n\|^2 & \ldots & \|\mathbf{x}_n\|^2 \rangle \end{bmatrix} + \begin{bmatrix} \|\mathbf{x}_1\|^2 & \ldots & \|\mathbf{x}_n\|^2 \rangle \\ \vdots & \ddots & \vdots \\ \|\mathbf{x}_1\|^2 & \ldots & \|\mathbf{x}_n\|^2 \rangle \end{bmatrix}\ . \tag{5.2}$$

Denoting the vector $(\|\mathbf{x}_1\|^2, \|\mathbf{x}_2\|^2, \ldots, \|\mathbf{x}_n\|^2)^\mathsf{T}$ by $\mathbf{z}$, we see that this may be written as

$$\mathtt{D} = -2\mathtt{X}^\mathsf{T}\mathtt{X} + \mathbf{z}\mathbf{1}^\mathsf{T} + \mathbf{1}\mathbf{z}^\mathsf{T}\ .$$

Here, it is key to notice that the matrix $\mathtt{X}^\mathsf{T}$ has the elements $\mathbf{x}_i^\mathsf{T}$ as its rows, and X has $\mathbf{x}_i$ as its columns. The (i,j)-th entry of $\mathtt{X}^\mathsf{T}\mathtt{X}$ is therefore $\mathbf{x}_i^\mathsf{T}\mathbf{x}_j$ which is $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$.

Now, we can calculate

$$\mathtt{HDH} = -2\mathtt{H}\mathtt{X}^{\mathsf{T}}\mathtt{X}\mathtt{H} + \mathtt{H}\mathbf{1}\mathbf{z}^{\mathsf{T}}\mathtt{H} + \mathtt{H}\mathbf{z}\mathbf{1}^{\mathsf{T}}\mathtt{H} \ .$$

But, according to (5.1), $\mathtt{H}\mathbf{1} = \mathbf{1}^{\mathsf{T}}\mathtt{H} = \mathbf{0}$, so this collapses (along with division by -2) to

$$\mathtt{B} = -(1/2)\mathtt{HDH} = (\mathtt{XH})^{\mathsf{T}}(\mathtt{XH}) \ .$$

Finally, define $\mathtt{X}_c = \mathtt{XH}$. Then,

$$\mathtt{B} = -\mathtt{HDH}/2 = \mathtt{X}_c^{\mathsf{T}}\mathtt{X}_c \ . \tag{5.3}$$

According to lemma 5.7, $\mathtt{X}_c = \mathtt{X} - \bar{\mathbf{x}}\mathbf{1}^{\mathsf{T}}$ is the matrix whose columns are the centred vectors $\mathbf{x}_i - \bar{\mathbf{x}}$. Hence, this last equation may be written (representing the entries of $\mathtt{B}$ by b_{ij}) as

$$b_{ij} = \left\langle \mathbf{x}_i - \bar{\mathbf{x}}, \mathbf{x}_j - \bar{\mathbf{x}} \right\rangle \ .$$

So, ultimately, the double-centering operation has the effect of replacing the distances $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ by the inner products of the centred vectors $\mathbf{x}_i - \bar{\mathbf{x}}$.

If $\mathbf{y}_i$ is defined as $\mathbf{x}_i - \bar{\mathbf{x}}$, then clearly $\|\mathbf{x}_i - \mathbf{x}_j\| = \|\mathbf{y}_i - \mathbf{y}_j\|$, since $\mathbf{x}_i$ and $\mathbf{y}_i$ differ only by translation by $\bar{\mathbf{x}}$.

This discussion may be summarized in the following result.

Theorem 5.8. *If $\{\mathbf{x}_i\}$ are n points in $\mathbb{R}^k$ and $\mathtt{D}$ is the $n \times n$ squared distance matrix with entries $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, then $\mathtt{B} = -\mathtt{HDH}/2$ has entries*

$$b_{ij} = \left\langle \mathbf{x}_i - \bar{\mathbf{x}}, \mathbf{x}_j - \bar{\mathbf{x}} \right\rangle \ .$$

Moreover, there exists a $k \times n$ matrix $\mathtt{X}_c$ (with $\mathtt{X}_c\mathbf{1} = \mathbf{0}$) such that $\mathtt{B} = \mathtt{X}_c^{\mathsf{T}}\mathtt{X}_c$.

Define $\mathbf{y}_i$ to be the i-th column of $\mathtt{X}_c$, then $\|\mathbf{y}_i - \mathbf{y}_j\| = \|\mathbf{x}_i - \mathbf{x}_j\|$.

The important consequence of this theorem is that if the points $\mathbf{x}_i$ are not known, and it is even unknown what is the dimension of the space $\mathbb{R}^k$ in which they sit, then it is still possible to find the points $\mathbf{y}_i$, starting only from the squared-distance matrix $\mathtt{D}$.

Gramian matrices. Any matrix $\mathtt{B}$ whose entries $b_{ij} = \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle$ for some points $\{\mathbf{x}_i\}$ in some $\mathbb{R}^k$ is called the *Gramian* of the points. A Gramian is always positive semi-definite. Any symmetric positive semi-definite matrix $\mathtt{B}$ can be written as

$$\mathtt{B} = \mathtt{U}^{\mathsf{T}}\Lambda\mathtt{U} = (\Lambda^{1/2}\mathtt{U})^{\mathsf{T}}(\Lambda^{1/2}\mathtt{U}) = \mathtt{X}^{\mathsf{T}}\mathtt{X}$$

and so $b_{ij} = \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle$, where $\mathbf{x}_i$ is the i-th column of $\mathtt{X}$. Hence, any positive semi-definite matrix is a Gramian matrix.

5.4.2 Backwards operation.

Suppose now that one starts with a matrix $\mathtt{D}$, with entries d_{ij}, which is intended to be a squared-distance matrix. Such a matrix must satisfy some obvious properties, namely

(i) $d_{ii} = 0$ for all i.
(ii) $d_{ij} = d_{ji}$ – the matrix $\mathtt{D}$ is symmetric.

There are other properties that $\mathtt{D}$ must satisfy, but we assume only these two properties, and call such a matrix $\mathtt{D}$ a hollow symmetric matrix.

Suppose that $\mathtt{D}$ is a hollow symmetric matrix and $\mathtt{B}$ is obtained from $\mathtt{D}$ by double-centering:

$$\mathtt{B} = -\mathtt{HDH}/2 \ .$$

Given $\mathtt{B}$, is it possible to retrieve $\mathtt{D}$?

Let $\mathbf{e}_i$ be the i-th basis vector in $\mathbb{R}^n$, namely the vector with 1 in place i and 0 elsewhere.

Lemma 5.9. *Let $\mathtt{D}$ be a hollow symmetric matrix and let $\mathtt{B} = -\mathtt{HDH}/2$. (Here $\mathtt{H}$ is the centering matrix.) Then*

$$d_{ij} = (\mathbf{e}_i - \mathbf{e}_j)^{\mathsf{T}}\mathtt{B}(\mathbf{e}_i - \mathbf{e}_j) \ .$$

Proof It is easily seen that $\mathbf{1}^{\mathsf{T}}(\mathbf{e}_i - \mathbf{e}_j) = 0$, so

$$\mathtt{H}(\mathbf{e}_i - \mathbf{e}_j) = (\mathtt{I} - \mathbf{1}\mathbf{1}^{\mathsf{T}})(\mathbf{e}_i - \mathbf{e}_j) = \mathbf{e}_i - \mathbf{e}_j \ .$$

Therefore

$$\begin{aligned}(\mathbf{e}_i - \mathbf{e}_j)^{\mathsf{T}}\mathtt{B}(\mathbf{e}_i - \mathbf{e}_j) &= -(\mathbf{e}_i - \mathbf{e}_j)^{\mathsf{T}}\mathtt{HDH}(\mathbf{e}_i - \mathbf{e}_j)/2 \\ &= -(\mathbf{e}_i - \mathbf{e}_j)^{\mathsf{T}}\mathtt{D}(\mathbf{e}_i - \mathbf{e}_j)/2 \\ &= -(d_{ii} + d_{jj} - d_{ij} - d_{ji})/2 \\ &= d_{ij} \ .\end{aligned}$$

This last step follows since $\mathtt{D}$ is symmetric and $d_{ii} = d_{jj} = 0$. □

Notation: Denote the mapping $\mathtt{D} \to \mathtt{B} = -\mathtt{HDH}/2$ by $\mathtt{B} = \tau(\mathtt{D})$.

The next theorem gives the condition that a matrix $\mathtt{D}$ is a squared-distance matrix.

Theorem 5.10. *Let $\mathtt{D}$ be hollow symmetric matrix and let $\mathtt{B} = \tau(\mathtt{D})$. There exist points $\mathbf{x}_i$ in some space $\mathtt{R}^k$ such that $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, if and only if $\mathtt{B}$ is positive-semi-definite.*

In this case, the smallest possible k satisfying this condition is equal to $\mathrm{rank}(\mathtt{B})$.

Proof Suppose that points $\mathbf{x}_i \in \mathbb{R}^k$ exist such that $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$. Then theorem 5.8 states that $\mathtt{B}$ can be written as $\mathtt{B} = \mathtt{X}_c^{\mathsf{T}}\mathtt{X}_c$. This shows that $\mathtt{B}$ is positive semi-definite, since $\mathbf{x}^{\mathsf{T}}\mathtt{B}\mathbf{x}^{\mathsf{T}} = (\mathtt{X}_c\mathbf{x})^{\mathsf{T}}(\mathtt{X}_c\mathbf{x}) \geq 0$.

Conversely, let $\mathtt{B} = \tau(\mathtt{D})$ be positive-semi-definite. If $\text{rank}(\mathtt{B}) = k$, then there exists an $n \times k$ matrix $\mathtt{X}$ such that $\mathtt{B} = \mathtt{X}^\mathsf{T}\mathtt{X}$ (obtained using the eigenvalue expansion of $\mathtt{B}$). Let $\mathbf{x}_i$ be the i-th column of matrix $\mathtt{X}$. This means that $b_{ij} = \mathbf{x}_i^\mathsf{T}\mathbf{x}_j = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. In particular $b_{ii} = \|\mathbf{x}_i\|^2$, and

$$\begin{aligned}(e_i - e_j)^\mathsf{T}\mathtt{B}(e_i - e_j) &= b_{ii} - b_{ij} - b_{ji} + b_{jj} \\ &= \|\mathbf{x}_i\|^2 - 2\langle \mathbf{x}_i, \mathbf{x}_j \rangle + \|\mathbf{x}_j\|^2 \\ &= \|\mathbf{x}_i - \mathbf{x}_j\|.\end{aligned}$$

However, according to lemma 5.9, $(e_i - e_j)^\mathsf{T}\mathtt{B}(e_i - e_j) = d_{ij}$. Therefore, $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$ as required. □

This theorem provides a necessary and sufficient condition for a matrix $\mathtt{D}$ to be a matrix of squared distances.

(i) $\mathtt{D}$ is hollow: $d_{ii} = 0$ for all i.
(ii) $\mathtt{D}$ is symmetric: $d_{ij} = d_{ji}$
(iii) $\mathtt{HDH}$ is negative semi-definite.

(Note here that the condition that $\mathtt{B}$ is posititive semi-definite has been replaced by the equivalent condition that $\mathtt{HDH} = -2\mathtt{B}$ is negative semi-definite.)

The condition that $\mathtt{HDH}$ is negative semi-definite can be replaced by a condition of $\mathtt{D}$ itself, according to the following lemma, which is given simply for interest.

Lemma 5.11. *A matrix* $\mathtt{B} = -\mathtt{HDH}/2$ *is positive semi-definite if and only if* $\mathtt{D}$ *satisfies* $\mathbf{x}^\mathsf{T}\mathtt{D}\mathbf{x} \leq 0$ *whenever* $\mathbf{1}^\mathsf{T}\mathbf{x} = 0$.

A matrix $\mathtt{D}$ satisfying this property is called *conditionally negative definite*. Thus, we have shown:

Theorem 5.12. *A matrix* $\mathtt{D}$ *is a squared-distance matrix if and only if it is hollow, symmetric and conditionally negative definite.*

5.4.3 Implementation examples

```
(* Now, we look at MDS *)
Clear[B, H, dim, A, on];
dim = 3;
tdim = 2;  (* The target dimension *)
PP = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

(* Form matrix of distances *)
A = Array[0 &, {dim, dim}];
For [i = 1, i <= dim, i++,
  For[j = 1, j <= dim, j++,
    A[[i, j]] = (PP[[i]] - PP[[j]]).(PP[[i]] - PP[[j]]);
    ];
  ];

Print["distances is", A // MatrixForm];

(* Form double projection matrix *)
on = Array[1 &, dim];
H = IdentityMatrix[dim] -  Outer[Times, on, on]  / dim;
Print["H is ", H // MatrixForm]

(* Perform the double projection *)
B = - H . A . H  / 2;
Print["B is",  B // MatrixForm];

(* Take the eigenvector decomposition and show it *)
{eval, evec} = Eigensystem[B];
Print["Check orthogonal eigenvectors: ", evec. Transpose[evec] // MatrixForm]
Print["Eigenvectors = ", evec // MatrixForm , "  Eigenvalues = ", eval // MatrixForm];

(* Find the result matrix *)
XX = Array[0 &, {dim, dim}];
For[i = 1, i <= dim, i++,
  XX[[i]] = Sqrt[eval[[i]]] evec[[i]];
  ];
XX = Transpose[XX];

(* Take only the first tdim dimensions *)
XX = XX[[All, 1 ;; tdim]];
Print["XX = ", XX // MatrixForm];

(* Print out the differences *)
A2 = Array[0 &, {dim, dim}];
For [i = 1, i <= dim, i++,
  For[j = 1, j <= dim, j++,
    A2[[i, j]] = (XX[[i]] - XX[[j]]).(XX[[i]] - XX[[j]]);
    ];
  ];

(* Print out the results *)
Print["Result distances = ", A2 // MatrixForm];
```

Fig. 5.1. Implementation of the classical MDS algorithm. Input is a set of three points in $\mathbb{R}^3$. Since three points always lie in a plane, the points should be embeddable in $\mathbb{R}^2$ without error. It will be seen that this algorithm (though essentially correct) gives the wrong results.

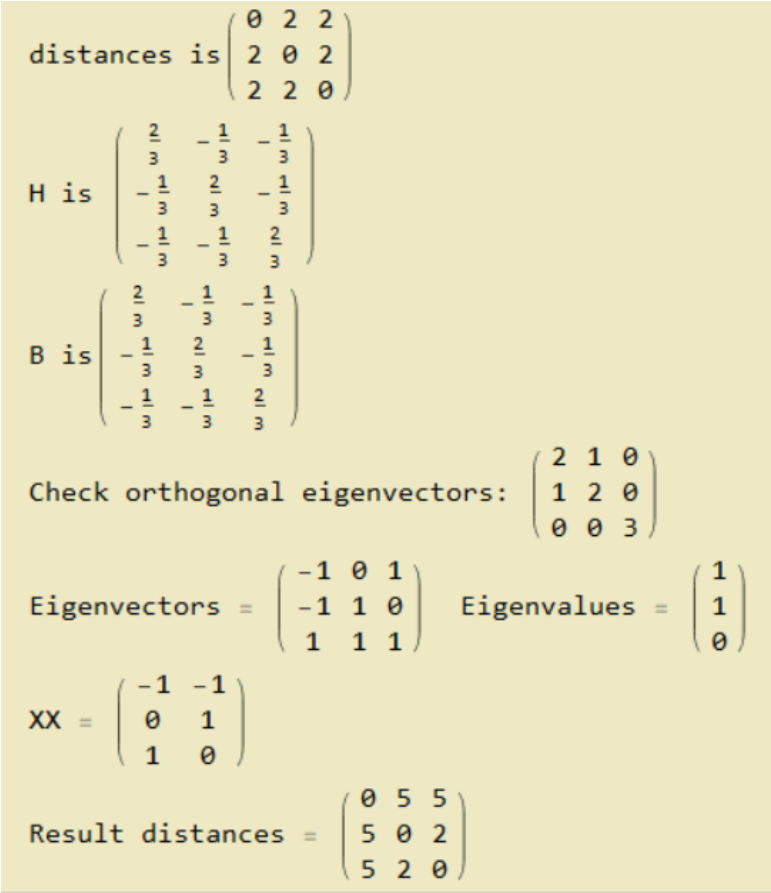

Fig. 5.2. *MDS output showing the wrong results. The algorithm has failed. Can you work out why?*

```
(* Now, we look at MDS *)
Clear[B, H, dim, A, on];
dim = 3;
tdim = 2;  (* The target dimension *)
PP = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1.00001}};
```

Fig. 5.3. The same input, but just the definition of the points PP has changed. Now it will be seen that the results are correct.

```
distances is {{0, 2, 2.00002}, {2, 0, 2.00002}, {2.00002, 2.00002, 0.}}

H is {{2/3, -1/3, -1/3}, {-1/3, 2/3, -1/3}, {-1/3, -1/3, 2/3}}

B is {{0.666669, -0.333331, -0.333338}, {-0.333331, 0.666669, -0.333338}, {-0.333338, -0.333338, 0.666676}}

Check orthogonal eigenvectors: {{1., 5.55112*10^-17, 1.11022*10^-16}, {5.55112*10^-17, 1., 5.55112*10^-17}, {1.11022*10^-16, 5.55112*10^-17, 1.}}

Eigenvectors = {{-0.408248, -0.408248, 0.816497}, {0.707107, -0.707107, 0.}, {-0.57735, -0.57735, -0.57735}}  Eigenvalues = {1.00001, 1., 4.44089*10^-16}

XX = {{-0.408251, 0.707107}, {-0.408251, -0.707107}, {0.816502, 0.}}

Result distances = {{0., 2., 2.00002}, {2., 0., 2.00002}, {2.00002, 2.00002, 0.}}
```

Fig. 5.4. *MDS output showing correct results. The rows of the matrix XX are the 2-dimensional points required. Note that they are not the same as the original points, but this is not required, just that the distances between points should be the same.*

5.5 ISOmap algorithm

See the paper by Tenenbaum et al.
https://web.mit.edu/cocosci/Papers/sci_reprint.pdf

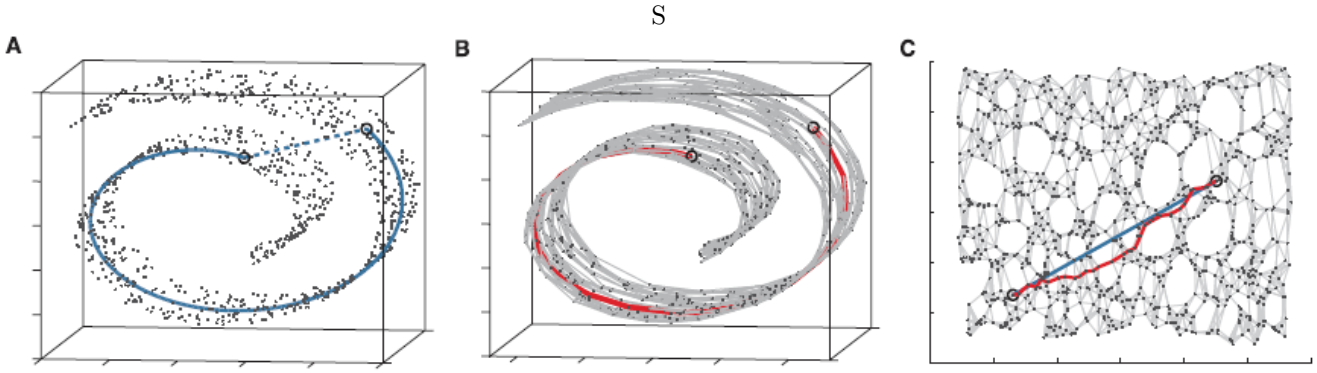

Fig. 5.5. *Figure from ISOmap paper illustrating the method.*

6

Lecture-6: SVM and Kernel methods

6.1 Kernels

A (real-valued) *kernel* k is a function

$$k : X \times X \to \mathbb{R}$$

where X is any set (for instance, but not always, Euclidean space). For instance, $k(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$ is an example of a kernel on $\mathbb{R}^k$, and so is $k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle$. These will be called the *distance kernel* and *inner product kernel* respectively.

We shall be particularly interested in symmetric kernels, satisfying $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$ for any two points $\mathbf{x}$ and $\mathbf{y}$ in X. We shall assume implicitly that any kernel being discussed is symmetric.

Given a kernel k and any set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in X$, one may form the matrix $\mathtt{K}$ whose entries are $k(\mathbf{x}_i, \mathbf{x}_j)$. This is a symmetric matrix.

A kernel k is called a *positive-definite* kernel, if the matrix $\mathtt{K}$ is always *positive semi-definite*, for any choice of the points $\mathbf{x}_i$. It may be objected that this should be called a positive semi-definite kernel, but this is the standard terminology.

A kernel k is called a *negative-definite kernel* if the matrix $\mathtt{K}$ is *conditionally negative-definite* for all choices of the $\mathbf{x}_i$. It may be objected once again that this ought to be called a *conditionally* negative semi-definite kernel, but this is the standard terminology.

Result 6.13. *The distance kernel $k(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$ and the inner-product kernel, $k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle$ on $\mathbb{R}^k$ are negative-definite and positive-definite kernels, respectively.*

Proof Consider any points $\{\mathbf{x}_i\}$ in $\mathbb{R}^k$. The matrix $\mathtt{K}$ corresponding to the inner-product kernel has entries $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$, and hence it can be written as $\mathtt{X}^\mathsf{T}\mathtt{X}$ where $\mathtt{X}$ is the matrix with columns $\mathbf{x}_i$. Consequently, $\mathtt{K}$ is positive-semi-definite.
In the case of the distance kernel, $k(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$, the matrix $\mathtt{K}$ is of the same form as the matrix $\mathtt{D}$, considered in theorem 5.8. According to that theorem, the matrix $\mathtt{B} = -\mathtt{HDH}/2$ is positive semi-definite, and consequently, $\mathtt{D}$ is conditionally

negative semi-definite. In other words, the kernel k is negative definite, by definition. □

For properties of kernels: www.haralick.org/ML/kernel_trick.pdf

6.2 Kernels and Hilbert Spaces

We describe here the standard way of introducing the connection between kernels on X and mapping X to a Hilbert space, or inner-product space.

Everyone quotes the result that a kernel gives rise a mapping into a Hilbert space, but it is not often explained which Hilbert space this is. Here is a simple (as simple as possible) explanation of this.

Suppose that there is a kernel $k : X \times X \to \mathbb{R}$. For each $x \in \mathrm{X}$, one can define a function $\varphi_x : X \to \mathbb{R}$ by

$$\varphi_x(\cdot) = k(\cdot, x) \ .$$

That is, the function $X \to \mathbb{R}$ obtained by fixing the second argument of $k(\cdot, \cdot)$ to x. Another way of looking at this is that

$$\varphi_x(y) = k(y, x) \ .$$

Thus, for each x, there is a function $\varphi_x : \mathrm{X} \to \mathbb{R}$.

Let Φ be the map $x \mapsto \varphi_x$. Then Φ maps X to $\mathbb{R}^X$, where $\mathbb{R}^X$ means the set of functions from X to $\mathbb{R}$.

$$\Phi : X \to \mathbb{R}^X \ .$$

Note that $\mathbb{R}^X$ is a vector space (over $\mathbb{R}$). Given two functions $\alpha, \beta : X \to \mathbb{R}$, one can add them $(\alpha + \beta)$ and multiply by a constant $r \in \mathbb{R}$ forming $r\alpha$ in the obvious way.

However, we do not normally consider the whole set $\mathbb{R}^X$, but simply the subset of $\mathbb{R}^X$ generated (spanned in the vector space) by the functions $\{\varphi_x \mid x \in X\}$. Let this subspace be called $\mathcal{H}'$. Later, we will define $\mathcal{H}$, which is an extension of $\mathcal{H}'$.

An example: RBF kernels. As an example, let k be the usual RBF kernel. Thus

$$k(\mathbf{x}, \mathbf{y}) = \exp(-\alpha \|\mathbf{x} - \mathbf{y}\|^2) \ .$$

We illustrate this particularly in the case where this is a 2-dimensional RBF kernel, so $X = \mathbb{R}^2$. See figure 6.1.

Let $\mathbf{x}$ be given, and write $\mathbf{x} = \mu$. Then

$$\varphi_{\mathbf{x}}(\mathbf{y}) = k(\mathbf{x}, \mathbf{y}) = \exp(-\alpha \|\mathbf{y} - \mu\|^2) \ .$$

Thus, φ_x is a simple Gaussian function with mean at μ and variance given by $1/(2\sigma^2) = \alpha$.

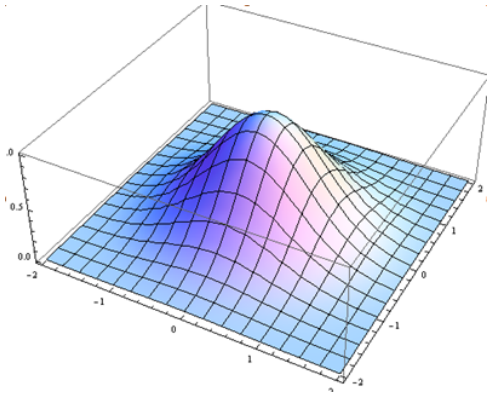

Fig. 6.1. *Simple RBF kernel.*

The vector space spanned by functions of this type consists of all functions $\mathbb{R}^2 \to \mathbb{R}$ that can be written in the form

$$f(\mathbf{y}) = \sum_{i=1}^{n} a_i \exp(-\alpha \|y - \mu\|^2) \ ,$$

for some (varying) n.

It is clear that a sum of two such functions is also of the same form, as is a multiplication by a scalar.

Thus $\mathcal{H}'$ is equal to the vector space of all functions $\mathbb{R}^2 \to \mathbb{R}$ that are the sum of a finite number of weighted Gaussians, all with the same variance $\sigma^2 = 1/(2\alpha)$. Such a function is shown in figure 6.2 (and also in figure 6.7 (*p* 123) – figure 6.9 (*p* 124)).

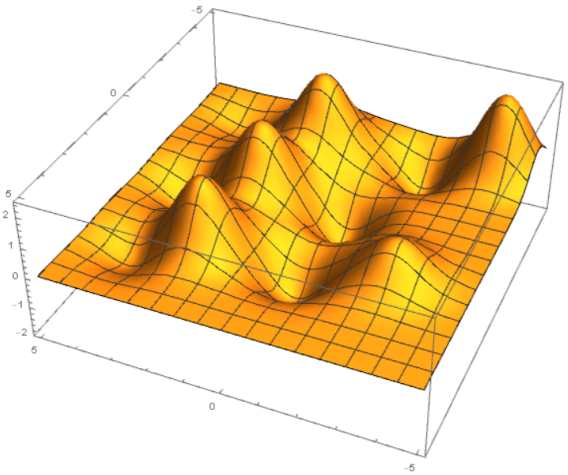

Fig. 6.2. *For the RBF kernel, a member of $\mathcal{H}'$ is a finite sum of weighted Gaussians.*

Inner product. To turn $\mathcal{H}'$ into an inner-product space, we need to define an inner-product. This is done by defining

$$\big\langle \varphi_x, \varphi_y \big\rangle_{\mathcal{H}'} = k(x,y) \ ,$$

for any two $x, y \in X$. (In the case of RBF kernels, $X = \mathbb{R}^2$.) This can then be extended linearly to any pair of elements in $\mathcal{H}'$ by

$$\big\langle \sum_{i=1}^{m} a_i \varphi_{x_i}, \sum_{j=1}^{n} b_j \varphi_{y_j} \big\rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i b_j \left\langle \varphi_{x_i}, \varphi_{y_j} \right\rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i b_j \, k(x_i, y_j) \ ,$$

where the x_i and y_i are any elements of X.

It needs to be verified that the inner product, so defined, satisfies the condition of being an inner product. This is left as an exercise. It depends on the fact that k is a positive-definite kernel.

Extension to a Hilbert space. The space $\mathcal{H}'$, consisting of finite weighted sums of kernel functions φ_{x_i} is an inner-product space, sometimes called a *pre-Hilbert* space. It lacks the condition of *completeness* that is part of the condition for a Hilbert space.

To complete it to a Hilbert space one needs to extend $\mathcal{H}$ to be a sum of countably (infinitely) many kernel functions φ_x, satisfying a square-summable property. We define elements of $\mathcal{H}$ to be elements of the form

$$\mathbf{z} = \sum_{i=1}^{\infty} a_i \varphi_{x_i} = \sum_{i=1}^{\infty} k(\cdot, x_i) \ ,$$

where

$$\sum_{i=1}^{\infty} a_i^2 < \infty \ .$$

This square-summability condition is required so that the inner-product-induced norm

$$\|\mathbf{z}\|^2 = \big\langle \mathbf{z}, \mathbf{z} \big\rangle = \sum_{i,j=1}^{\infty} a_i a_j k(x_i, x_j) \tag{6.1}$$

is finite for any countable set of elements x_i.

Then, an inner product can be defined for elements $\mathbf{z}_1 = \sum_{i=1}^{\infty} a_i \varphi_{x_i}$ and $\mathbf{z}_2 = \sum_{j=1}^{\infty} b_j \varphi_{x_j}$ may be defined by

$$\big\langle \mathbf{z}_1, \mathbf{z}_2 \big\rangle = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} a_i b_j \, k(x_i, x_j) \ , \tag{6.2}$$

where this sum is finite (assuming that the kernel is bounded).

Exercises.

(i) If real numbers $a_i, i = 1, \ldots, \infty$ satisfy $\sum_{i=1}^{\infty} a_i^2 < \infty$, then $\sum_{i,j=1}^{\infty} |a_i a_j| < \infty$.
(ii) Thus, if $k(x, y)$ is bounded (always less than some number Ω for all $x, y \in X$) then the expression (6.1) is always finite.
(iii) If $\sum_{i=1}^{\infty} a_i^2 < \infty$ and $\sum_{j=1}^{\infty} b_j^2 < \infty$, then $\sum_{i,j=1}^{\infty} |a_i b_j| < \infty$. Thus, the square-summable condition ensures that the inner-product defined by (6.2) is finite.
(iv) If the kernel k is positive-definite, then the inner product defined in this way satisfies $\langle \mathbf{z}, \mathbf{z} \rangle > 0$, unless $\mathbf{z} = 0$. (Investigate this. There is some condition missing here.)

Hilbert space or pre-Hilbert space. For most purposes, we may assume that we are working in the pre-Hilbert space $\mathcal{H}'$ instead of $\mathcal{H}$. Rarely, if ever, is it necessary to assume or require the completeness property.

6.3 Support Vector Machines

We now talk about Support Vector Machines (SVMs).

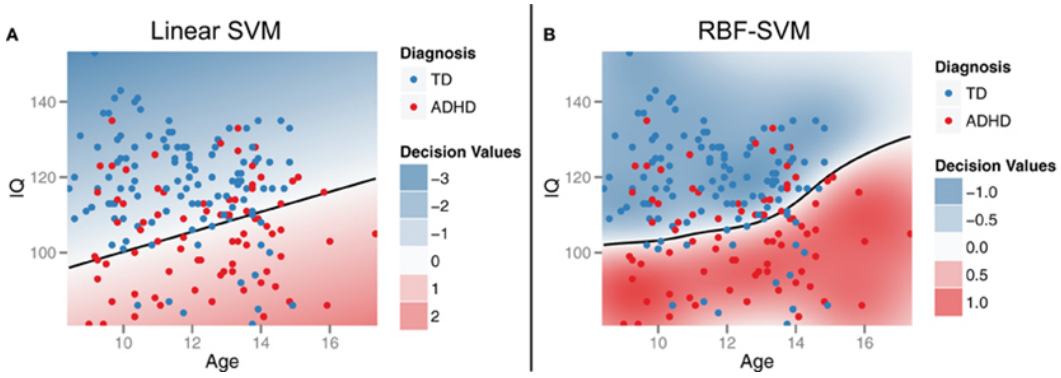

Fig. 6.3. *Illustration of linear and kernel SVM. A linear SVM finds a separating hyperplane between two different classes of data. If the data is not strictly separable, it finds a hyperplane that minimizes a suitable loss-function. In many cases, data is not separable by simple hyperplanes. The technique of kernel SVM is used to solve this problem.*

As shown in figure 6.3 a linear SVM finds a hyperplane that separates data into two classes. It takes a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ as input, and outputs a value y, defined by

$$y = \langle \mathbf{x}, \mathbf{w} \rangle + b$$

where a vector $\mathbf{w}$ and offset b are learnt during a training process. This provides a classifier for the input $\mathbf{x}$; the vector $\mathbf{x}$ is assigned to class 1 or -1 depending on whether $y > 0$ or $y < 0$.

Note that the equation $\langle \mathbf{x}, \mathbf{w} \rangle + b = 0$ defines a hyperplane (affine space of dimension $m-1$ in $\mathbb{R}^m$) that separates $\mathbb{R}^m$ into two parts, according to whether $\langle \mathbf{x}, \mathbf{w} \rangle + b$ is positive or negative. The vector $\mathbf{w}$ is a vector perpendicular to the hyperplane, pointing from the negative to the positive side.

Given elements $\{\mathbf{x}_i\}$ in some $\mathbb{R}^m$, and for each $\mathbf{x}_i$ a corresponding $y_i \in \{-1, 1\}$, indicating which of two classes a date point $\mathbf{x}_i$ belongs to, we wish to separate them into two classes, in such a way that is generalizable to further (unseen) data points.

In linear SVMs, in which the data lies in some $\mathbb{R}^m$, this is done by finding a hyperplane that separates the data belonging to the two classes. There are two details.

(i) If the data is separable, we normally wish to find the "max-margin" hyperplane, namely the one that separates the two classes, with the maximum possible distance between the hyperplane and the data. This is so that further unseen data can then be classified more accurately.

(ii) Frequently, it will prove impossible to separate the data completely by a hyperplane. Instead, one defines a (convex) loss function that penalizes data that is incorrectly classified by a hyperplane. One then selects the hyperplane that minimizes the loss.

Generally speaking, linear SVMs are too weak to separate data into two given classes. This is where *kernel SVMs* come into play. In figure 6.3 it is shown how data can be separated into two parts by curved surfaces.

Aside: CNNs. A linear SVM is identical with a final *fully-connected* layer in a CNN, used as a classifier. In fact, when used as a classifier, with a final fully-connected layer, a CNN may be thought as divided into two parts: one part (all layers but the last) implementing a non-linear function $\Phi : \mathbb{R}^m \to \mathbb{R}^n$ (where m is the dimension of the input of the complete CNN): the second part, implemented by the last fully-connected layer, is simply a linear SVM used for classification.

Of course, in a CNN, the SVM and the preceding non-linear function f are trained together to implement a desired classification task. The process of training is distinct from the task of training a stand-alone linear SVM, which uses convex optimization methods. The consequence of this is that the function Φ implemented by the CNN is customized to the particular problem at hand, and is crafted for the training data, to obtain the best classification results.

6.4 Kernel SVMs

6.4.1 Mapping induced by a kernel

The purpose of this section is to give a different perspective on kernel SVMs.

It is usual in the literature on kernels to speak of positive-definite kernels inducing a map of the space X, on which the kernel is defined, into some Hilbert space. To

recall, a Hilbert space is simply a vector space with an inner product, and an additional property called *completeness*. Every finite-dimensional vector space can be given an inner product, and is complete. Normally, the completeness property will not be too important.

Here, we take a non-traditional route to explaining the mapping into a Hilbert space. Let k be a positive definite kernel. This means that for all sets of points $\{\mathbf{x}_i\}$ in X, the matrix $\mathtt{K}$ having entries $k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semi-definite.

Now, consider such a set of points $\{\mathbf{x}_i, i = 1, \ldots, n\}$. This may consist of all the elements from some "training set" of data in X, or if X is finite, it can consist of all the points in X. Bear in mind that X is a set of arbitrary type, not necessarily a vector space, or $\mathbb{R}^m$. The matrix $\mathtt{K}$ is positive semi-definite (assuming that k is a positive-definite kernel) and so can be decomposed as

$$\mathtt{K} = \mathtt{Y}^\mathsf{T}\mathtt{Y} \ ,$$

where $\mathtt{Y}$ is a matrix of dimension $r \times n$, where r is the rank of matrix $\mathtt{K}$ and n is the number of points $\mathbf{x}_i$.[1]

Letting $\mathbf{y}_i$ be the i-th column of $\mathtt{Y}$ (hence, the i-th *row* of matrix $\mathtt{Y}^\mathsf{T}$), then the (i,j)-th entry of $\mathtt{K}$ is equal to $\mathbf{y}_i^\mathsf{T}\mathbf{y}_j = \langle \mathbf{y}_i, \mathbf{y}_j \rangle$. Each vector $\mathbf{y}_i$ is an r-vector. Then,

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathtt{K})_{ij} = \langle \mathbf{y}_i, \mathbf{y}_j \rangle \ .$$

We denote the correspondence $\Phi : \mathbf{x}_i \mapsto \mathbf{y}_i$.

What we have shown here is that the points $\mathbf{x}_i$ are mapped by Φ onto points $\mathbf{y}_i$ in the Euclidean space $\mathbb{R}^r$, and the following condition holds:

$$\langle \mathbf{y}_i, \mathbf{y}_j \rangle = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle = k(\mathbf{x}_i, \mathbf{x}_j) \ . \tag{6.3}$$

In other words, the points $\mathbf{x}_i$, which in general have nothing to do with any vector space, or inner product, can be mapped to points $\mathbf{y}_i = \Phi(\mathbf{x}_i)$ in a Euclidean inner-product space $\mathbb{R}^r$, so that the important relation $\langle \mathbf{y}_i, \mathbf{y}_j \rangle = k(\mathbf{x}_i, \mathbf{x}_j)$ holds.

This can be stated as an important result.

Result 6.14. *If $k : X \times X \to \mathbb{R}$ is a positive-definite kernel, and $\{\mathbf{x}_i\}$ are a (finite) set of points in X, then there exist points $\mathbf{y}_i$ in a Euclidean space $\mathbb{R}^r$ and a mapping $\Phi : \mathbf{x}_i \mapsto \mathbf{y}_i$ such that*

$$\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle = \langle \mathbf{y}_i, \mathbf{y}_j \rangle = k(\mathbf{x}_i, \mathbf{x}_j) \ .$$

[1] Recall, that we can get the matrix $\mathtt{Y}$ by taking the eigenvalue expansion $\mathtt{U}^\mathsf{T}\Lambda\mathtt{U}$ of $\mathtt{K}$ and factoring this as $\mathtt{Y}^\mathsf{T}\mathtt{Y}$, where $\mathtt{Y} = \sqrt{\Lambda}\mathtt{U}$. Note how this fact relies on the fact that $\mathtt{K}$ is positive-semidefinite, so the eigenvalues are non-negative, and that a symmetric matrix $\mathtt{K}$ has eigenvectors that are orthogonal (so $\mathtt{U}$ is orthogonal).

6.4.2 Ways of getting positive-definite kernels

Linear kernel. The inner product kernel is positive definite, since the Gram matrix (or Gramian), the matrix of inner products, is positive definite. This is not a very interesting example of a kernel, by itself, since it leads to nothing new.

From negative-definite kernels. Recall that a negative-definite kernel is one such that the matrix $\mathtt{K}$ is *conditionally* negative-definite, meaning that $\mathbf{x}^\mathsf{T}\mathtt{K}\mathbf{x} \leq 0$ for any $\mathbf{x}$ such that $\mathbf{x}^\mathsf{T}\mathbf{1} = \sum_{i=1}^m x_i = 0$. Consequently, it does *not* follow that the negative of a negative-definite kernel is positive definite.

A matrix $\mathtt{D}$ with entries $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, for points $\mathbf{x}_i \in \mathbb{R}^m$ is conditionally negative-definite because the matrix

$$\mathtt{B} = -\mathtt{HDH}/2 \ ,$$

where $\mathtt{H}$ is the centering matrix, was shown in theorem 5.8 (*p* 98) to be positive-semi-definite, since it equals the Gram matrix of the centred vectors $\mathbf{x}_i - \bar{\mathbf{x}}$.

This allows us to characterize negative-definite kernels.

Definition 6.15. A symmetric kernel $k : X \times X \to \mathbb{R}$ is defined to be a *distance kernel* if there is a mapping $\Phi : X \to \mathcal{H}$ into a Hilbert space such that $k(\mathbf{x}_i, \mathbf{x}_j) = \|\Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\|^2_{\mathcal{H}}$ for all pairs $(\mathbf{x}_i, \mathbf{x}_j) \in X \times X$.

Result 6.16. *A kernel k is a distance kernel if and only if it is negative definite and $k(\mathbf{x}, \mathbf{x}) = 0$ for all $\mathbf{x}$.*

Proof. The condition that k is a negative definite symmetric kernel with $k(\mathbf{x}, \mathbf{x}) = 0$ is the same as stating that the matrix $\mathtt{K}$ defined by any points $\mathbf{x}_i$ is hollow, symmetric and conditionally negative semi-definite. Then this result follows from theorem 5.12-(*p* 100). □

This, along with result 6.14 shows that a positive-definite kernels is essentially the same as an inner-product, whereas a negative-definite kernels is essentially the same as squared-distance function.

Radial basis function kernels. An important kernel is the following, defined on $\mathbb{R}^m$

$$k(\mathbf{x}, \mathbf{y}) = \exp(-\alpha\|\mathbf{x} - \mathbf{y}\|^2) \ .$$

This is called the RBF (radial basis function) kernel figure 6.1.

Result 6.17. *For every $\alpha > 0$, the RBF kernel is positive-definite.*

Proof A proof for this relies on two facts, which will not shown here.

(i) The product of two positive-definite kernels is positive definite. Thus, if $k_1 : X \times X \to \mathbb{R}$ and similarly k_2 are positive definite kernels, then $(k_1 \cdot k_2)$, defined by $(k_1 \cdot k_2)(\mathbf{x}) = k_1(\mathbf{x})\, k_2(\mathbf{x})$, is also positive definite. (This is not altogether trivial to prove).

(ii) The limit of a convergent sequence of positive-definite kernels is positive definite. Thus, if (k_i) is a sequence of kernels and k^* defined by $k^*(\mathbf{x}) = \lim_{i\to\infty} k_i(\mathbf{x})$ (if the limit exists) is positive definite. (This is relatively straight-forward).

(You can no doubt find a proof of this on the internet).
From this you can prove that

$$\exp(2\alpha\langle \mathbf{x}, \mathbf{y}\rangle) = \sum_{i=0}^{\infty} (2\alpha\langle \mathbf{x}, \mathbf{y}\rangle)^n / n!$$

is positive definite, since $\langle \mathbf{x}, \mathbf{y}\rangle$ is.
Finally,

$$\exp(-\alpha\|\mathbf{x}-\mathbf{y}\|^2) = \exp(2\alpha\langle \mathbf{x}, \mathbf{y}\rangle)\, \exp(-\alpha\|\mathbf{x}\|^2)\, \exp(-\alpha\|\mathbf{y}\|^2)$$

is positive definite. (A little bit to fill in here – it can be left as an exercise.) □

For lots of other ways to get positive-definite kernels, see Haralick's slides: www.haralick.org/ML/kernel_trick.pdf

6.4.3 Kernel SVM

As with linear SVMs, we are given elements $\mathbf{x}_i$ in some set X, and for each $\mathbf{x}_i$ a corresponding $y_i \in \{-1, 1\}$, indicating which of two classes a data point $\mathbf{x}_i$ belongs to. We wish to separate them into two classes, in such a way that is generalizable to further (unseen) data points.

In kernel SVM, the first step is to define a suitable kernel. This can be an RBF kernel if the data lie in some Euclidean space $\mathbb{R}^m$, however, there are other choices. Sometimes, the data do not lie in a Euclidean space at all, in any natural way. Let k be a positive definite kernel. There is a map Φ that maps each $\mathbf{x}_i$ to $\mathbf{y}_i = \Phi(\mathbf{x}_i)$, where $\mathbf{y}_i$ lies in some Hilbert space $\mathcal{H}$ (or inner-product space, or Euclidean space).

Then, the SVM will seek to find a hyperplane in $\mathcal{H}$ that separates (or most nearly separates) the data according to which side of the hyperplane a data point $\mathbf{y}_i$ lies on.

Linear SVM. A hyperplane in $\mathbb{R}^m$ is defined as the set of points $\mathbf{z} \in \mathcal{H}$ that satisfies the equation

$$\langle \mathbf{w}, \mathbf{z}\rangle + b = 0$$

where $\mathbf{w}$ is a vector in $\mathbb{R}^m$ and b is a scalar.

If $\mathbf{w}$ and b are known, a data point $\mathbf{x} \in X$ is classified according to the sign of

$$\langle \mathbf{w}, \mathbf{x} \rangle + b \ .$$

Given a set of training data $\{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, the parameters $\mathbf{w} \in \mathbb{R}^m$ and $b \in \mathbb{R}$ of the hyperplane are chosen to minimize a loss of the form

$$L(\mathbf{w}, b \mid \{(\mathbf{x}_i, y_i)\}) = \sum_{i=1}^{n} f\big([\langle \mathbf{w}, \mathbf{x}_i \rangle + b]\, y_i\big) \tag{6.4}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a decreasing function, such as in figure 6.4. The function f penalizes data wrongly classified by the hyperplane (parametrized by $\mathbf{w}$ and b).

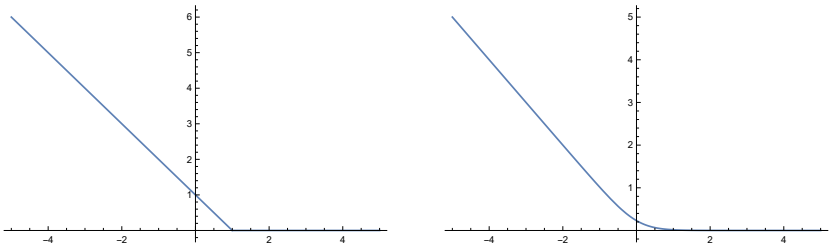

Fig. 6.4. *The* relu *function (and a smooth version). This function penalizes negative values (misclassifications), but positive values, as long as they exceed a threshold, incur no penalties. This minimum threshold has the effect of encouraging the maximum possible margin. In other words, it tries to ensure that data is as far on the right side of the separating hyperplane as possible. (To understand this loss function, it is a good excercise to see what happens if both* $\mathbf{w}$ *and* b *are multiplied by a constant greater than or less than* 1*.)*

Kernel SVM. A hyperplane in a Hilbert space $\mathcal{H}$ is defined in much the same way as one in $\mathbb{R}^m$, as the set of points $\mathbf{z} \in \mathcal{H}$ that satisfies the equation

$$\langle \mathbf{w}, \mathbf{z} \rangle + b = 0$$

where $\mathbf{w}$ is an element of $\mathcal{H}$ and b is a scalar.

If $\mathbf{w} \in \mathcal{H}$ and b are known, a data point $\mathbf{x} \in X$ is classified according to the sign of

$$\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b \ .$$

The parameters $\mathbf{w}$ and b of the hyperplane are chosen to minimize a loss of the form

$$L(\mathbf{w}, b \mid \{(\mathbf{x}_i, y_i)\}) = \sum_{i=1}^{n} f\big(y_i[\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle + b]\big) \tag{6.5}$$

where the pairs $(\mathbf{x}_i, y_i), i = 1, \ldots, n)$ are a set of training data.

A useful (and important) observation is that the optimal parameter (hyperplane normal) $\mathbf{w}$ lies in the span of the "feature vectors", $\Phi(\mathbf{x}_i)$.

Theorem 6.18 (Representer theorem – Schölkopf, Smola, et al). *The optimal* $\mathbf{w}$ *that minimizes the loss function (6.5) is of the form*

$$\mathbf{w} = \sum_{i=1}^{n} w_i \Phi(\mathbf{x}_i)$$

with $w_i \in \mathbb{R}$.

See https://en.wikipedia.org/wiki/Representer_theorem. The theorem is proved there in slightly more generality. See also https://alex.smola.org/papers/2001/SchHerSmo01.pdf which gives a good alternative (to these notes) account of elementary kernel theory and proves a general representer theorem.

In other words, the vector $\mathbf{w}$ we are looking for is in the span of the $\Phi(\mathbf{x}_i)$.

Now, we play a little notational trickery. Represent a vector $\sum_{i=1}^{n} a_i \Phi(\mathbf{x})$ by the vector $\mathbf{a} \in \mathbb{R}^n$. The element $\Phi(\mathbf{x}_i)$ is therefore represented by the vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ with 1 in place i. Denote this by $\mathbf{e}_i \in \mathbb{R}^n$. Effectively, what this is saying is that the data elements $\mathbf{x}_i$ are represented by the basis elements $\mathbf{e}_i$ in $\mathbb{R}^n$, so, the corners of the standard regular simplex. https://en.wikipedia.org/wiki/Simplex.

The inner product defined by the kernel leads to an inner product, denoted $\langle \cdot, \cdot \rangle_{\mathtt{K}}$ on $\mathbb{R}^n$, defined by

$$\langle \mathbf{e}_i, \mathbf{e}_j \rangle_{\mathtt{K}} = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathcal{H}} = k(\mathbf{x}_i, \mathbf{x}_j) \ .$$

This is extended to an inner product on $\mathbb{R}^n$ given by

$$\langle \mathbf{a}, \mathbf{b} \rangle_{\mathtt{K}} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i \, k(\mathbf{x}_i, \mathbf{x}_j) \, b_j = \mathbf{a}^{\mathsf{T}} \mathtt{K} \mathbf{b} \ .$$

This may be seen as a sort of Mahalanobis inner product[1] of $\mathbf{a}$ and $\mathbf{b}$ defined by the positive definite matrix $\mathtt{K}$. The mapping

$$\mathbf{a} = (a_1, \ldots a_n) \mapsto \sum_{i=1}^{n} a_i \Phi(\mathbf{x}_i)$$

defines a isometry (inner-product-preserving linear mapping)

$$\left(\mathbb{R}^n, \langle \cdot, \cdot \rangle_{\mathtt{K}}\right) \to \left(\mathcal{H}, \langle \cdot, \cdot \rangle_{\mathcal{H}}\right)$$

from $\mathbb{R}^n$ with the Mahalanobis inner product to $\mathcal{H}$ with the inner product defined by the kernel. It maps $\mathbb{R}^n$ onto the subspace of $\mathcal{H}$ spanned by the $\Phi(\mathbf{x}_i)$.

Thus, we mayidentify the span of the $\Phi(\mathbf{x}_i)$ in $\mathcal{H}$ with $\mathbb{R}^n$ equipped with the Mahalanobis inner product, defined by the kernel matrix $\mathtt{K}$ with entries $k(\mathbf{x}_i, \mathbf{x}_j)$. This

[1] The term Mahalanobis norm and Mahalanobis inner product should perhaps be reserved to describe an inner product $\langle \cdot, \cdot \rangle_{\Sigma}$ defined in terms of a covariance matrix of some data. However, it will be used here to describe any inner product $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathtt{K}} = \mathbf{x}^{\mathsf{T}} \mathtt{K} \mathbf{y}$ where $\mathtt{K}$ is a positive-definite matrix.

is very convenient, because it allows us to work in $\mathbb{R}^n$ using an inner Mahalanobis inner product.

Using this notation, we can write (6.5)in terms of a vector $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ as

$$\begin{aligned} L(\mathbf{w}, b \mid \{(\mathbf{x}_i, \mathbf{y}_i)\}) &= \sum_{i=1}^{n} f\left(y_i\left[\left\langle \sum_{j=1}^{n} w_j \Phi(\mathbf{x}_j), \Phi(\mathbf{x}_i)\right\rangle_{\mathcal{H}} + b\right]\right) \\ &= \sum_{i=1}^{n} f\left(y_i\left[\left\langle \mathbf{w}, \mathbf{e}_i\right\rangle_{\mathtt{K}} + b\right]\right) \ . \end{aligned} \tag{6.6}$$

This should be compared with the formulation of linear SVM, which minimizes a misclassification loss

$$L(\mathbf{w}, b \mid \{(\mathbf{x}_i, \mathbf{y}_i)\}) = \sum_{i=1}^{n} f\left(y_i\left[\left\langle \mathbf{w}, \mathbf{x}_i\right\rangle_2 + b\right]\right) \tag{6.7}$$

where $\left\langle \cdot, \cdot \right\rangle_2$ is the standard inner product on $\mathbb{R}^n$. Since $\mathbf{e}_i$ in (6.6) represents the data point $\mathbf{x}_i$, it may be seen that the essential difference between kernel SVM and linear SVM is that kernel-SVM minimizes a loss based on the Mahalanobis inner product $\left\langle \cdot, \cdot \right\rangle_{\mathtt{K}}$, where $\mathtt{K}$ is defined in terms of the training samples, instead of the standard inner-product.

Embedding X into $\mathbb{R}^n$. We give a further formulation of kernel-SVM. Once more, let $\{\mathbf{x}_i, i = 1, \ldots, n\}$ be a set of training points in X and let $\{y_i\} \in \{1, -1\}$ be given ground-truth classifications.

A map $\varphi : X \to \mathbb{R}^n$ is defined by

$$\mathbf{x} \mapsto (k(\mathbf{x}, \mathbf{x}_1), k(\mathbf{x}, \mathbf{x}_2), \ldots, k(\mathbf{x}, \mathbf{x}_n)^{\mathsf{T}} \ .$$

This mapping depends directly on the training set $\{\mathbf{x}_i\}$. The vector $\varphi(\mathbf{x}_i)$ is simply the i=th column of the matrix $\mathtt{K}$, equal to $\mathtt{K}\mathbf{e}_i$. In this case,

$$\left\langle \mathbf{w}, \mathbf{e}_i\right\rangle_{\mathtt{K}} = \mathbf{w}^{\mathsf{T}}\mathtt{K}\mathbf{e}_i = \mathbf{w}^{\mathsf{T}}\varphi(\mathbf{x}_i) \ .$$

Then, the loss (6.6) can be written as

$$L(\mathbf{w}, b \mid \{(\mathbf{x}_i, \mathbf{y}_i)\}) = \sum_{i=1}^{n} f\left(y_i(\mathbf{w}^{\mathsf{T}}\varphi(\mathbf{x}_i) + b)\right) \ . \tag{6.8}$$

Now, this is just the same as (6.7), with $\varphi(\mathbf{x}_i)$ replacing $\mathbf{x}_i$.

Classification. The purpose of SVM is to classify unseen data, that is, data that does not belong to a training set $\{\mathbf{x}_i\}$. Let $\mathbf{y} \in X$ be a further data element previously unseen. Using the trained parameters $\mathbf{w} \in \mathbb{R}^n, b \in R$, (found by minimizing

(6.6)), we classify $\mathbf{x}$ according to the sign of

$$\begin{aligned}\Big\langle \sum_{j=1}^{n} w_j \Phi(\mathbf{x}_j), \Phi(\mathbf{x}) \Big\rangle_{\mathcal{H}} + b &= \sum_{j=1}^{n} w_j \big\langle \Phi(\mathbf{x}_j), \Phi(\mathbf{x}) \big\rangle_{\mathcal{H}} + b \\ &= \sum_{j=1}^{n} w_j k(\mathbf{x}_j, \mathbf{x}) + b \\ &= \mathbf{w}^{\mathsf{T}} \varphi(\mathbf{x}) + b\end{aligned}$$

This shows that both the training and classification using kernel-SVM is equivalent to linear SVM carried out on data mapped by $\varphi : X \to \mathbb{R}^n$.

Why do we need a positive-definite kernel? Under the formulation just given, let $k : X \times X \to \mathbb{R}$ be any function. Given some training points $\mathbf{x}_i$, a mapping $\varphi : X \to \mathbb{R}^n$ is defined by $\mathbf{x} \mapsto (k(\mathbf{x}, \mathbf{x}_1, \ldots, k(\mathbf{x}, \mathbf{x}_n)^{\mathsf{T}}$. Then, one can train a linear SVM defined on $\mathbb{R}^n$ in terms of the training set $\{(\varphi(\mathbf{x}_i), y_i\}$. This produces parameters $\mathbf{w}$ and b of a hyperplane in $\mathbb{R}^n$, and then one classifies test data according to the sign of $\mathbf{w}^{\mathsf{T}} \varphi(\mathbf{x}) + b$.

As has been seen, this is exactly what kernel-SVM does, so there does not seem to be any necessity to use a positive-definite kernel.

If there is a very large training set, then $\mathbb{R}^n$ is very high-dimensional. It is possible that one can implement kernel-SVM without explicitly computing the embedded data points $\varphi(\mathbf{x}_i)$, but some kernel-SVM implementations at least do compute this mapping explicitly. (This requires the computation of n vectors in $\mathbb{R}^n$. If n is very large, this can run into implementation problems.)

6.4.4 The kernel trick

The "kernel trick" is often referred to. What is this? It is basically what we have just seen.

Essentially, the kernel trick is to use a positive-definite kernel defined on a space X containing data elements $\mathbf{x}_i$ to map these data elements into a Hilbert space (or inner-product space), equipped with the inner product defined by the kernel. One then carries out all appropriate calculations in this Hilbert space.

In the cases where the representer theorem is valid, in some form, the mapping Φ may be seen as the mapping that takes every data point $\mathbf{x}_i$ in some training set to the elements $\mathbf{e}_i$ of the standard basis for $\mathbb{R}^n$, and the Mahalanobis inner-product $\langle \cdot, \cdot \rangle_{\mathtt{K}}$ is taken as the inner-product on $\mathbb{R}^n$. Calculations then are carried out in $\mathbb{R}^n$ with this inner-product.

Note that once we have an inner-product on a finite-dimensional real vector space, such as $\mathbb{R}^n$, we effectively can carry out all algorithms as if they were in $\mathbb{R}^n$, since

along with the inner product comes a notion of distance, orthogonality of vectors, orthogonal basis, and so on.

Nearly positive-definite kernels. If the kernel is not quite positive-definite then some slight modifications must be made.

For instance, it was seen that the RBF kernel

$$k(\mathbf{x}, \mathbf{y}) = \exp(-\alpha \|\mathbf{x} - \mathbf{y}\|^2)$$

is positive definite for all $\alpha > 0$. This relies on $\|\cdot\|$ being a Euclidean distance in some inner product space (a normed-vector space without inner-product is not enough). For instance, in the case where a distance $d(\mathbf{x}, \mathbf{y})$ is determined empirically, such as the distance through a graph in the case of the ISOmap algorithm, the RBF kernel

$$k(\mathbf{x}, \mathbf{y}) = \exp(-\alpha d(\mathbf{x}, \mathbf{y})^2)$$

may not be exactly positive definite. The kernel defined by $d(\mathbf{x}, \mathbf{y})2$ may not be exactly negative semi-definite, corresponding to Euclidean distances, and hence the kernel $k(\mathbf{x}, \mathbf{y})$ defined in this way will not be positive definite. However, if the matrix `K` got by applying k to the data has some negative eigenvalues, hopefully small, one may then proceed by setting the negative eigenvalues (along with any small positive eigenvalues) to zero and then proceeding as before. In effect, this applies a small correction to the matrix `K`, and hence to the kernel, so that the matrix becomes positive-semi-definite.

6.4.5 SVM and Parzen windows

The standard kernel SVM approach, is to use some imagined non-linear function $\Phi : \mathbb{R}^m \to \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space, in the usual description of kernel SVMs. This is just a fancy way of saying that $\mathcal{H}$ is a vector space in which we can take inner products. Sometimes $\mathcal{H}$ is finite-dimensional, and sometimes infinite, but in the case where $\mathcal{H}$ is of infinite dimension, the complete training set $\{\mathbf{x}_i\}$ is always mapped into a finite-dimensional subset, so Φ may be considered simply as a mapping $\Phi : \mathbb{R}^m \to \mathbb{R}^n$.

The justification for introducing the mapping Φ in the description of kernel SVMs is to highlight the essential similarity with linear SVMs. In reality, neither the mapping Φ nor the space $\mathcal{H}$ is constructed explicitly, and they may be thought of as being a convenient fiction. All that is needed is the kernel mapping $K : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$. The way that this relates to the feature mapping Φ is through the formula

$$k(\mathbf{x}_1, \mathbf{x}_2) = \big\langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \big\rangle_{\mathcal{H}} \ . \tag{6.9}$$

To use a kernel SVM, one need only define a suitable kernel k. In the most general case, k is simply a mapping $k : X \times X \to \mathbb{R}$ (or more generally still, a mapping

to the complex numbers) where X is any set. In order for (6.9) to define a proper inner-product, the kernel mapping should be positive-definite.

A commonly used kernel is the Gaussian Radial Basis Function (RBF) kernel, defined by

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp\big(-\alpha\|\mathbf{x}_1 - \mathbf{x}_2\|^2\big) \tag{6.10}$$

where $\|\cdot\|$ is the standard Euclidean norm in $\mathbb{R}^m$, and $\alpha > 0$ is some chosen scale factor. If $\mathbf{x}_1$ is fixed, then as a function of $\mathbf{x}_2$ this is just a circularly symmetric Gaussian function centred at $\mathbf{x}_1$, with width determined by α. See figure 6.1 (*p* 108)

Now, given an input $\mathbf{x} \in \mathbb{R}^m$, we may carry out two-label classification by forming the expression

$$y = \big\langle \mathbf{w}, \Phi(x) \big\rangle_{\mathcal{H}} + b \tag{6.11}$$

where $\mathbf{w} \in \mathcal{H}$ and $b \in \mathbb{R}$. Then $\mathbf{x}$ is classified as -1 or 1 depending on whether $y < 0$ or $y > 0$. The elements $\mathbf{w}$ and b are learnt during a training process.

Normally, however, specifying a general value of $\mathbf{w}$ is problematic, because $\mathcal{H}$ may be infinite-dimensional, or else its structure may not be well understood. Therefore, one usually makes the simplifying assumption that $\mathbf{w}$ lies in the span of the $\Phi(\mathbf{x}_i)$. Thus,

$$\mathbf{w} = \sum_{i=1}^{k} a_i\, \Phi(\mu_i)\ .$$

where the μ_i are chosen from among the training samples $\{\mathbf{x}_i\}$, and are called the *support vectors.* With this assumption, we may write

$$\begin{aligned}
\mathbf{y} &= \big\langle \mathbf{w}, \Phi(\mathbf{x}) \big\rangle_{\mathcal{H}} + b \\
&= \Big\langle \sum_{i=1}^{k} a_i \Phi(\mu_i), \Phi(\mathbf{x}) \Big\rangle_{\mathcal{H}} + b \\
&= \sum_{i=1}^{k} a_i \big\langle \Phi(\mu_i), \Phi(\mathbf{x}) \big\rangle_{\mathcal{H}} + b \\
&= \sum_{i=1}^{k} a_i k(\mu_i, \mathbf{x}) + b\ .
\end{aligned}$$

If $k(\mu_i, \mathbf{x})$ is the standard inner-product in $\mathbb{R}^m$, then this is just the linear SVM. For different kernel functions it gives something different. For example, in the case of the RBF kernel, this becomes

$$y(\mathbf{x}) = \sum_{i=1}^{k} a_i \exp\big(-\alpha\|\mathbf{x} - \mu_i\|^2\big)\ . \tag{6.12}$$

In normal practice with SVMs, the Gaussian centres, μ_i, are chosen from among the training samples, but considered simply as a classifier, based on whether (6.12) is

positive or negative, this is not necessary. One may consider the process of training the classifier simply as selecting some number k of centres μ_i, and weights a_i, and classifying the sample $\mathbf{x}$ according to the value of the function $y(\mathbf{x})$ defined here. (One may also envisage the possibility of allowing different values of α for each i).

Although a RBF kernel is discussed here, there is no reason not to use other functions in its place. As long as the function depends on some set of parameters (such as the Gaussian defined by centres μ_i and scales α_i) which may be learnt through some training process, any function may be used. In particular, there is no need for the kernel function to be positive definite.

Why it works. Although a linear SVM can only classify data based on its separation by a hyperplane, it is easy to see that a kernel-SVM can classify data based on more complex separation surfaces. For instance, consider a set of data divided into two classes, such as shown in figure 6.5.

Although this data is not accurately separable by a line (hyperplane), it may be separated by placing sufficiently many Gaussians, centred inside the red area with positive weight, and within the blue area with negative weight. Then evaluating their sum and classifying points as "red" if the combined Gaussian function evaluates positive, or "blue" if it evaluates negative, one obtains a classifier that separates the plane in the required manner.

It is interesting to note that a linear SVM can be obtained as a special case of an RBF-kernel SVM using just two Gaussians. In particular, the function

$$f(x) = \exp(-\|\mathbf{x} - \mathbf{x}_1\|^2) - \exp(-\|\mathbf{x} - \mathbf{x}_2\|^2)$$

will be positive for points closer to $\mathbf{x}_1$ than to $\mathbf{x}_2$, and non-positive otherwise. These two sets are separated by a hyperplane placed mid-way between $\mathbf{x}_1$ and $\mathbf{x}_2$.

The approach of placing Gaussians in the plane, or Euclidean space is similar to the idea of Parzen windows (see internet), in which distributions are approximated by summing normal Gaussian distributions placed at locations within the support region of the distribution. In the method suggested here using RBF-kernel SVMs, an approximation to the support regions of each of the two classes is formed by summing Gaussians placed within each regions, and classifying test points according to the values of the approximation functions to the two different support regions. Thus, RBF-kernel SVMs may be understood in this simple way, without the machinery of Hilbert spaces, or the requirement of positive-definiteness of the kernel.

Example. We consider some data that is to be divided into two classes. Look at the data in figure 6.6.

We take a 20×20 grid of samples across the domain of the function, and generate $N = 400$ pairs $(\mathbf{x}_i, y_i)$, where $\mathbf{x}$ contains the $2D$-coordinates of the point and $\mathbf{y}_i$ is

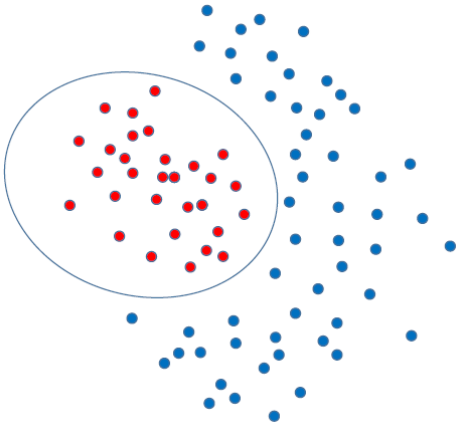

Fig. 6.5. *Data not separable by a hyperplane (a line in dimension 2). Although this data cannot be accurately separated by a line, a kernel-SVM with RBF kernel may work. Gaussians with positive weight are placed, centred in the red region, and with negative weight in the blue region. The combined function provides a classifier for the data.*

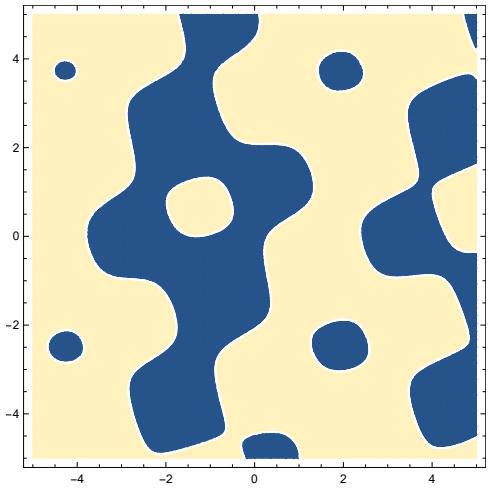

Fig. 6.6. *This shows 2-dimensional data that is to be separated into two classes. The two classes are separated by a relatively complex separation curve. This diagram may be thought of as the "ground truth" classification of the 2-dimensional data, which is to be approximated (we hope) by a kernel SVM.*

1 or -1 indicating which class $\mathbf{x}_i$ belongs to. Now we form a simple function

$$y(\mathbf{x}) = \sum_{i=1}^{N} y_i \exp\left(-\alpha\|\mathbf{x}_i - \mathbf{x}\|^2\right) = \sum_{i=1}^{N} y_i k(\mathbf{x}_i, \mathbf{x}) \ . \tag{6.13}$$

In other words, this is the linear combination of the RBF kernels, of the type given in (6.12), centred at means $\mu_i = \mathbf{x}_i$ and having coefficients given by $\alpha_i = y_i = \pm 1$

The linear combination (6.13) involves all the data points. It is possible to sep-

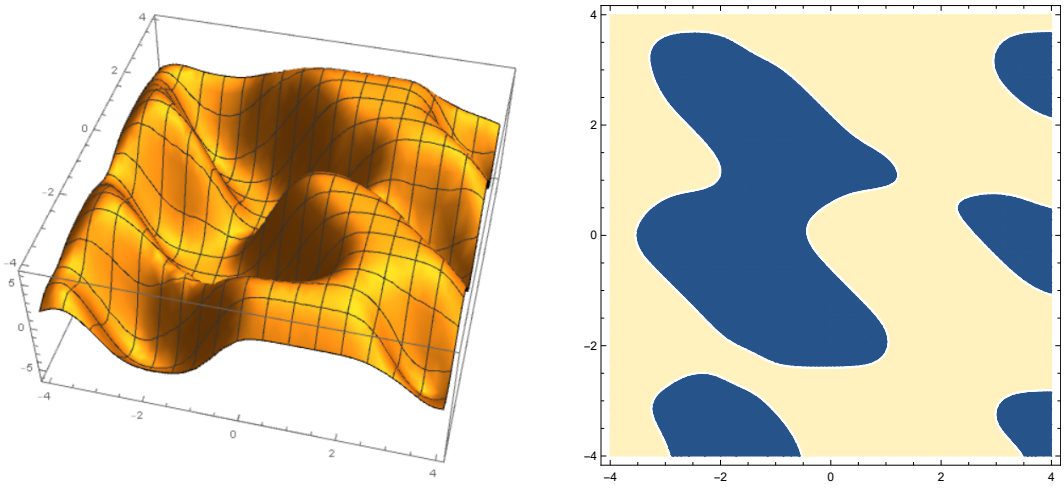

Fig. 6.7. *This shows the classification of the data given by the function $y(\mathbf{x})$ defined in (6.13). The value of α used is $1/(2\sigma^2)$, where σ is the separation between adjacent data points. (All the data in this example are on an equally spaced grid.) On the left the function $y(\mathbf{x})$. On the right the classification according to whether $y(\mathbf{x})$ is positive or negative. This is a reasonable approximation of the ground-truth given in figure 6.6.*

arate the data by a function that involves combinations of fewer RBF functions. Taking only samples $\mathbf{x}_i$ that lie at the boundary between the two classes (in other words, they have a nearest neighbour that is in a different class) will be sufficient to separate the classes efficiently. In this case, there are 198 samples that lie at the class boundary. This is shown in figure 6.8.

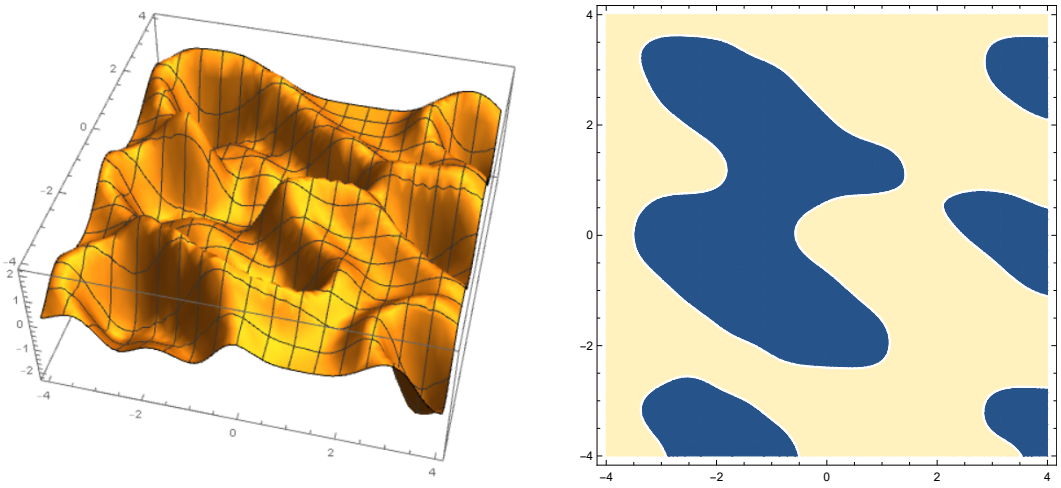

Fig. 6.8. *This shows the classification given by the function $y(\mathbf{x}) = \sum_{k=1}^{198} y_{i_k} \exp\left(-\alpha\|\mathbf{x}_{i_k} - \mathbf{x}\|^2\right)$ where the $\mathbf{x}_{i_k}$ are the 198 boundary points between the two classes. On the left is shown the classification function $y(\mathbf{x})$, on the right the segmentation according to whether $y(\mathbf{x})$ is positive of negative.*

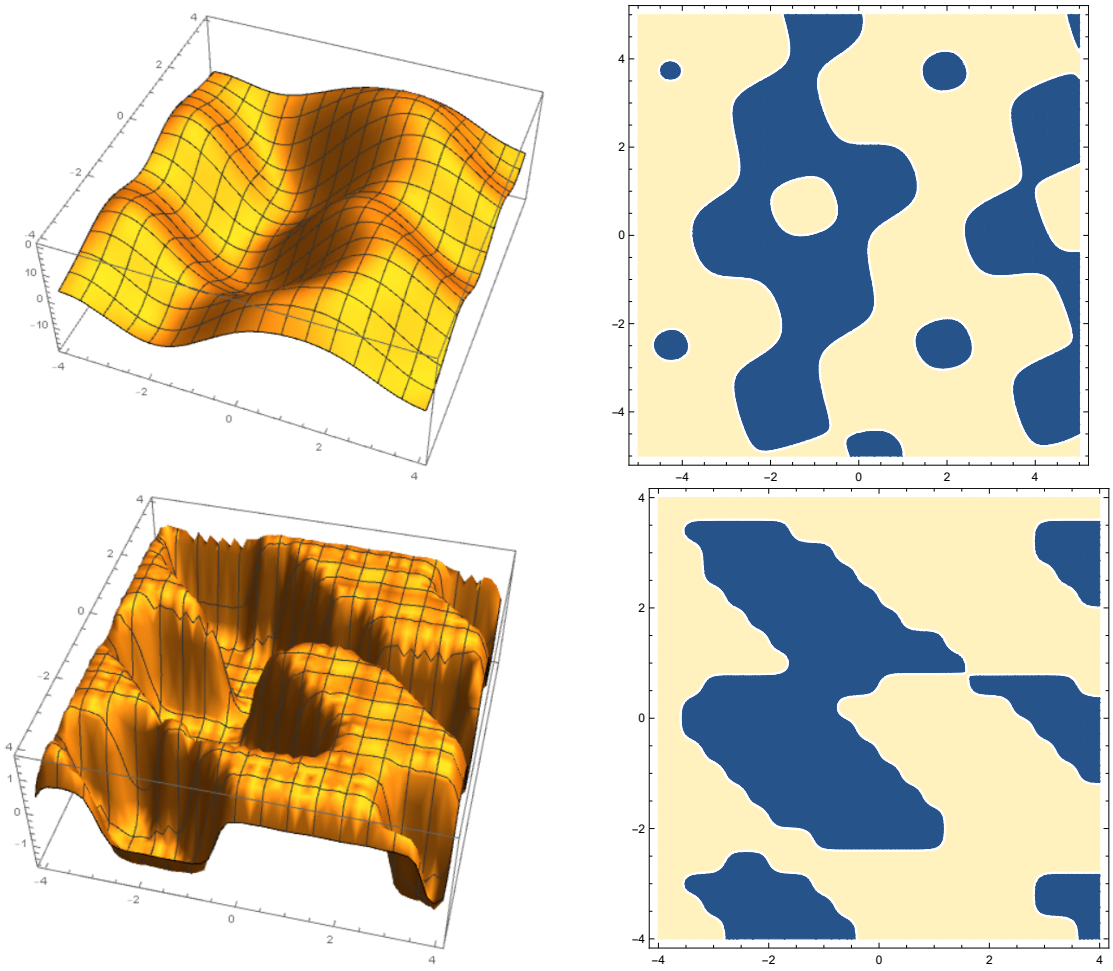

Fig. 6.9. *Effect of α. The parameter $\alpha = 1/(2\sigma^2)$ where σ^2 is the variance of the RBF kernel, controlling its width.. This figure shows the effect of changing the value of α. At the top $\sigma = 2d$. At the bottom $\sigma = d/2$, where d is the distance between adjacent training points $\mathbf{x}_i$. As is seen, too large a value of σ lead to over-smoothing, too small values can lead to irregular contours (overfitting).*
If the RBF kernels are very narrow, centred on all the data points, then clearly the data can be separated exactly, but this solution may not generalize well to other unseen data.

These diagrams are for illustration only. It is not implied that kernel SVM works in this way. In fact, kernel SVM will determine the optimum number of non-zero coefficients α_i in the linear combination (6.12). It will also not be the case that all the coefficients α_i will be ± 1, as here. These examples are simply given to show the way that kernel SVMs can separate data along complex boundaries.

6.5 SVM kernel as deblurring

A further interpretation of kernel-SVM is discussed now. This section contains some relatively specialized Mathematics, and is given with the purpose of showing a different interpretation of kernel SVMs, for interest and enlightenment only.

As seen previously, for each point $\mathbf{x} \in X$, there is a mapping $\Phi_{\mathbf{x}} = k(\cdot, \mathbf{x}) : X \to \mathbb{R}$. We specialize to the case where $X = \mathbb{R}^m$ (or at least some space in which addition can take place – an abelian semigroup), and suppose that the kernel has the form

$$k(\mathbf{x}, \mathbf{y}) = f(\mathbf{x} - \mathbf{y}) \ .$$

The exponential RBF kernel is of this type. To avoid using a new symbol (f) we denote it as $k : X \to \mathbb{R}$, and instead of writing $k(\mathbf{x}, \mathbf{y})$, we write $k(\mathbf{x} - \mathbf{y})$.

We have seen that given a set of n training points $\mathbf{x}_i \in X$, there exists a vector $\varphi(\mathbf{y})$ where $\varphi(\mathbf{y})_i = k(\mathbf{y} - \mathbf{x}_i)$. This is the same thing as the function $\Phi(\mathbf{y})$ sampled at the points $\mathbf{x}_i$ to form an n-vector.

Furthermore, in kernel-SVM, we find a vector $\mathbf{w}$ and a classifying hyperplane $\mathbf{w}^\mathsf{T}\varphi(\mathbf{x}) + b = 0$. This can also be written as

$$\sum_{i=1}^{n} w_i \, k(\mathbf{y} - \mathbf{x}_i) + b \ .$$

Let us write this as

$$\sum_{i=1}^{n} w(\mathbf{x}_i) k(\mathbf{y} - \mathbf{x}_i) + b \ .$$

This notation is justified, since the value w_i is multiplied with the value of $\varphi(\mathbf{x}_i)$ for the training sample $\mathbf{x}_i$, and hence is associated with the i-th training sample.

The value of this should be positive if $\mathbf{x}$ belongs to class 1 and negative if it belongs to class -1. Suppose we define a *classifying function* $C(\mathbf{x})$, where $C(\mathbf{x})$ is the class that data point $\mathbf{x}$ should belong to. Then, the required condition is

$$\sum_{i=1}^{n} w(\mathbf{x}_i) k(\mathbf{y} - \mathbf{x}_i) + b = C(\mathbf{y}) \ .$$

Well, in reality, we do not need the sum to equal $C(\mathbf{y})$, but simply that they have the same sign.

As stated before, the terms $w(\mathbf{x}_i)$ and $k(\mathbf{y} - \mathbf{x}_i)$ can be considered as samples (at the points $\mathbf{x}_i$) of functions w and $k(\mathbf{y}, \cdot)$ sampled at the points $\mathbf{x}_i$.

Now, let the number of sampled points $\mathbf{x}_i$ increase towards infinity. Dropping the constant b (for no apparent reason), the summation becomes an integral, and this expression becomes

$$\int w(\mathbf{x}) \, k(\mathbf{y} - \mathbf{x}) \, d\mathbf{x} = C(\mathbf{y}) \ . \tag{6.14}$$

As the number of training samples increases, the summation is an approximation of the integral.

The formula (6.14) is a convolution of function w and kernel k, both functions $\mathbb{R}^m \to \mathbb{R}$. Therefore, given a classifying function $C(\mathbf{y})$ and a kernel k, the task of kernel SVM is to find a function w such that

$$w * k = C \ ,$$

where $*$ denotes convolution. Thinking of convolution as blurring (certainly convolution with a Gaussian kernel is a sort of low-pass filtering, or blurring), finding w is equivalent to deblurring the classification function C.

The standard way to do deblurring is to use the Fourier transform. In this case, from the properties of convolution, $\mathcal{F}(w)(\omega) = \mathcal{F}(k)(\omega)\mathcal{F}(C)(\omega)$, where the functions are multiplied pointwise. Here ω is the frequency-domain variable, not to be confused with w. Hence,

$$w(\mathbf{x}) = \mathcal{F}^{-1}(\mathcal{F}(C)(\omega)/\mathcal{F}(k)(\omega))(\mathbf{x})$$

This method of deblurring is effective as long as $\mathcal{F}(k)(\omega)$ is never zero, otherwise division by zero results.

Here, there is a beautiful connection with a theorem of Harmonic Analysis, Bochner's theorem. In simple form, Bochner's theorem states:

Theorem 6.19. *If $k : R^m \to \mathbb{R}$ is a positive-definite kernel then its Fourier transform $\mathcal{F}(k)$ is a positive distribution on $\mathbb{R}^m$.*

For instance, the Fourier transform of the Gaussian kernel $k(\mathbf{x}) = \exp(\|\mathbf{x}\|^2/2)$ is equal to itself – in general, the Fourier transform of a Gaussian is a Gaussian, which is therefore non-zero at all points. This gives some insight into why positive-definite kernels are of particular interest for kernel SVMs.

The condition in this theorem is actually an if-and-only-if condition, and the theorem holds on an locally-compact semi-group, which is a sort of object that Harmonic analysts are interested in. However, the theorem specializes to the form given above. https://en.wikipedia.org/wiki/Bochner%27s_theorem.

6.6 Kernel SVM in a CNN

This general approach may be used to replace the final linear SVM in a CNN classifier with a kernel SVM. The reasoning behind this is that both a CNN and a kernel SVM can be seen as applying a non-linear function Φ to the input. The CNN applies a customized non-linear function to the input, whereas a kernel SVM applies a given fixed feature function Φ defined by the kernel k. Although commonly outperformed by deep CNNs, kernel SVMs have proven very effective in implementing simple classifiers with a relatively small number of parameters. The motive for the present work is to demonstrate that by appending a kernel SVM to a CNN and

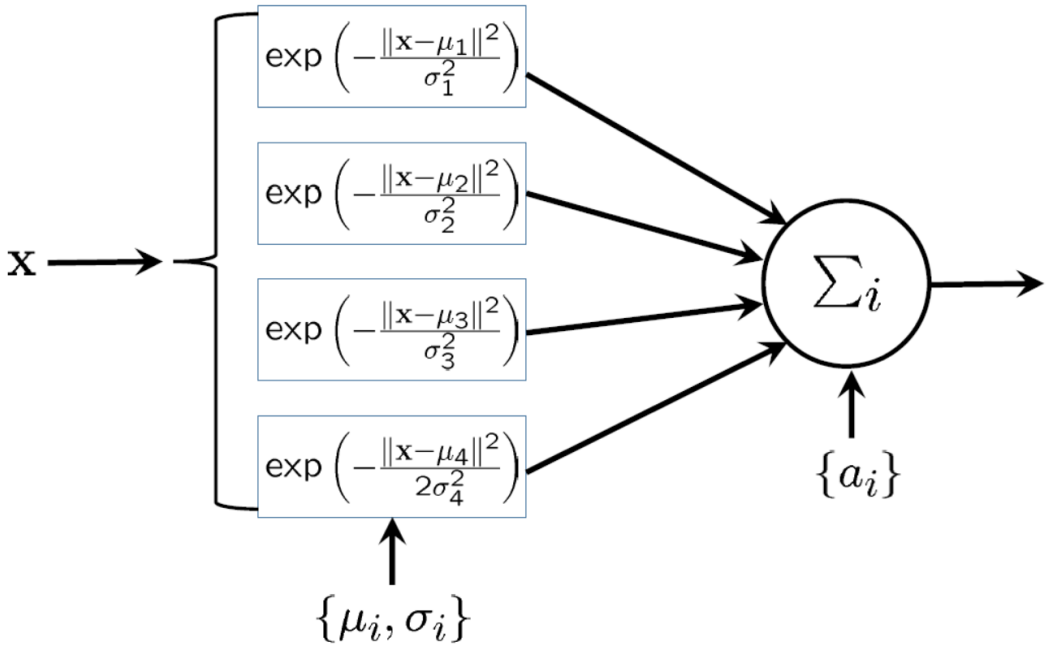

Fig. 6.10. *Final layer of a CNN implementing an RBF-kernel SVM.*

training them together, one may obtain the benefits of both approaches, namely the high classification performance of the CNN with the compact parametrization of the SVM. The hope is that one may obtain equal performace to a deep network with large numbers of parameters with a relatively simpler network involving a kernel SVM as the last stage.

The SVM is implemented by defining a new final layer for the CNN, as shown in figure 6.10. As seen in the diagram, the layer computes and sums Gaussian functions with centres μ_i and width specified by $\alpha_i = 1/\sigma_i^2$. The complete CNN with kernel-SVM is shown in figure 6.6.

Loss layer. In figure 6.6, the loss is denoted by $L(\mathbf{x}_0, \mathbf{y}_0, \theta)$, where θ represents the ensemble of all parameters of the CNN, namely the weights w_i and the parameters μ_i, σ_i, a_i of the kernel SVM layer. The loss is a function of the training vector $\mathbf{x}$, its corresponding ground truth $\mathbf{y}_0$ and the parameters of the CNN.

The loss layer takes the ground truth $\mathbf{y}_0$ for the training set and evaluates it against the output $f(x)$ of the complete CNN. For of classification, the input $\mathbf{x}_0$ is deemed to belong to class -1 if $f(x) < 0$ and to class 1 otherwise. Thus, if $f(x) > 0$ and $\mathbf{y}_0 = 1$, then there is no loss; similarly this is so if $f(x) < 0$ and $\mathbf{y}_0 = -1$. Therefore, a suitable loss function is

$$L(\mathbf{x}_0, \mathbf{y}_0, \theta) = \max(0, \mathbf{y}_0 f(x)) \ ,$$

namely, the relu loss, or hinge loss commonly used in CNNs or SVMs.

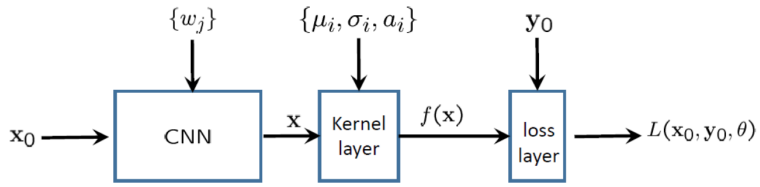

Fig. 6.11. *Complete CNN including a final nonlinear kernel layer, and loss-layer. The loss-layer is removed after training. The weights of the leading layers of the CNN, and the parameters of the function $f(x)$ are learned by back-propagation, all together. of these functions, such as taking a maximum*

Fig. 6.12. *Carl Gustav Jacob Jacobi (1804 – 1851). Jacobi is known for many contributions to Mathematics, such as the Jacobian, the theory of elliptic functions, and Jacobi fields (in Riemannian geometry).*

Generalization. Examining figure 6.10, it is evident that it may be generalized by replacing the Gaussian functions by arbitrary parametrized function, or by replacing the summation by some alternative way of combining the outputs

6.7 Numerics of finding eigenvalues.

Generally, finding eigenvalues of arbitrary matrices can be difficult. Well, it is easy in that you just use some standard library routine. However, what this means is that the results may not be entirely reliable in some cases.

On the othe hand, finding the eigenvalues of a symmetric matrix is significantly easier (provided that the routine used knows that the matrix is symmetric). A good algorithm is the Jacobi algorithm https://en.wikipedia.org/wiki/Jacobi_eigenvalue_algorithm.

The problem is also a lot easier if it is required only to find a small number of largest (or smallest) eigenvalues. Given a symmetric positive-definite matrix $\mathtt{M}$ of dimension $n \times n$, the problem of finding a matrix $\mathtt{X}$ such that $\mathtt{M} \approx \mathtt{X}^{\mathsf{T}}\mathtt{X}$ is known as finding a low-rank approximation of $\mathtt{M}$.

The standard method for finding a low-rank approximation to a matrix $\mathtt{M}$ is to find its eigenvalue decomposition, but this can be expensive. The Power Method (Golub and Van Loan) is a useful method for finding the dominant eigenvector of a matrix $\mathtt{M}$, converging rapidly if the largest eigenvalue sufficiently dominates the next largest. Starting from a random vector $\mathbf{u}_0$ the method is to repeatedly apply $\mathtt{M}$ to $\mathbf{u}$ and normalize the result:

$$\mathbf{u}_{k+1} = \mathtt{M}\mathbf{u}_k / \|\mathtt{M}\mathbf{u}_k\|$$

The Power Method may be extended for finding the dominant subspace of dimension r (i.e. the subspace spanned by the first r eigenvectors). Start with a random matrix $\mathtt{Y}_0$ with r orthonormal columns and repeatedly multiply by $\mathtt{M}$ and and re-orthonormalize columns:

$$\mathtt{Y}_k = \mathtt{M}\mathtt{Y}_{k-1}\mathtt{N}_k \tag{6.15}$$

where $\mathtt{N}_k$ is an upper triangular matrix that makes the columns of $\mathtt{Y}_k$ orthonormal. This normalizing matrix $\mathtt{N}_k$ is found by the Gram-Schmidt process, equivalent to QR decomposition of $\mathtt{M}\mathtt{Y}_{k-1}$.

https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

The resulting algorithm is known as *orthogonal power iteration* (Golub Van-Loan). Note the stopping point of this algorithm. If $\mathtt{M} = \mathtt{Y}_k\mathtt{Y}_k^{\mathsf{T}}$, then

$$\mathtt{Y}_{k+1} = \mathtt{M}\mathtt{Y}_k\mathtt{N}_{k+1} = \mathtt{Y}_k(\mathtt{Y}_k^{\mathsf{T}}\mathtt{Y}_k)\mathtt{N}_k = \mathtt{Y}_k\mathtt{N}_{k+1} \ ,$$

since $\mathtt{Y}_k$ has orgonal columns ($\mathtt{Y}^{\mathsf{T}}\mathtt{Y} = \mathtt{I}$). However, then $\mathtt{N}_{k+1}$ is the identity, since already $\mathtt{Y}_k$ has orthogonal columns. Hence, $\mathtt{Y}_{k+1} = \mathtt{Y}_k$ and the algorithm stops.

Sparse matrices. This process is particularly rapid in the case where $\mathtt{M}$ is a sparse matrix. The matrix $\mathtt{Y}$ cannot be expected to be sparse, however if $r \ll n$, then $\mathtt{Y}$ is small compared with $\mathtt{M}$. The algorithm involves two steps, repeated.

(i) Computation of $\mathtt{M}\mathtt{Y}_{k-1}$. This is rapid as long as $\mathtt{M}$ is sparse.

(ii) Gram-Schmidt process applied to the $n \times r$ matrix $\mathtt{M} \times \mathtt{Y}_{k-1}$. In this case $\mathtt{N}_k$ is an $r \times r$ matrix, and the operation requires $2nr^2$ operations, hence linear in n.
https://stackoverflow.com/questions/27986225/computational-complexity-of-gram-schmidt-orthogonalization-algorithm.