Basic Linear Algebra

Machine Learning

Linear Algebra

Matrix Miscellany I

(Adapted from J.A. Richards Gi and X. Jia, *Remote Sensing Digital Image Analysis*)

- Addition and subtraction of matrices are additions and subtractions of corresponding entries. For example:

$$M \pm N = \begin{bmatrix} m_{11} \pm n_{11} & m_{12} \pm n_{12} \\ m_{21} \pm n_{21} & m_{22} \pm n_{22} \end{bmatrix}$$

  (and similarly for higher dimension matrices)

- Multiplication of an $m \times n$ matrix by an $n \times k$ matrix is a $m \times k$ matrix consisting of "sums of row-column products". For example:

$$MN = \begin{bmatrix} m_{11}n_{11} + m_{12}n_{21} & m_{11}n_{12} + m_{12}n_{22} \\ m_{21}n_{11} + m_{22}n_{21} & m_{21}n_{12} + m_{22}n_{22} \end{bmatrix}$$

  That is, entry 11 of the product is the sum of products from Row 1, Col 1; entry 12 is the sum of products from Row 1, Col 2 and so on.

Matrix Miscellany II

- New vectors (matrices) can be obtained from old ones by linear combination of the components. For example, the new vector $\mathbf{y}$ with components:

$$\begin{aligned} y_1 &= m_{11}x_1 + m_{12}x_2 \\ y_2 &= m_{21}x_1 + m_{22}x_2 \end{aligned}$$

is a *linear transformation* of the vector $\mathbf{x}$ and can be written in matrix notation as:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

or:

$$\mathbf{y} = M\mathbf{x}$$

This kind of transformation is an example of a *rotational* transformation

Matrix Miscellany III

- The *inverse* of a matrix M is the matrix M^{-1} s.t.

$$MM^{-1} = I$$

  where I is the *identity* matrix (a matrix with 1's along the diagonal, and 0's everywhere else). The inverse of a matrix may not always exist, and is not always easy to compute. But it will usually be done by a program, rather than by hand
- Calculation of the inverse requires knowledge of the determinant $|M|$ or $det(M)$. In 2 dimensions:

$$|M| = det(M) = \begin{vmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{vmatrix} = m_{11}m_{22} - m_{12}m_{21}$$

- Division of matrices is not defined: the inverse of a matrix takes the place of division

Matrix Miscellany IV

- Multiplication of a vector $\mathbf{x}$ by a matrix M transforms $\mathbf{x}$ to a vector $\mathbf{y}$. Sometimes, the same effect can be achieved by multiplying $\mathbf{x}$ by a number (which may be a complex number) λ. That is:

$$M\mathbf{x} = \lambda\mathbf{x}$$

  or:

$$(M - \lambda I)\mathbf{x} = \mathbf{0}$$

- It can be shown that for the above to hold, either $\mathbf{x} = \mathbf{0}$ or $det(M - \lambda I) = 0$. This is a polynomial equation in λ. The solutions of λ for which $det(M - \lambda I) = 0$ are the *eigenvalues* of M. The resulting values of $\mathbf{x}$ that satisfy $M\mathbf{x} = \lambda\mathbf{x}$ are called the *eigenvectors* of M

Matrix Miscellany V

- The transpose of a matrix M, denoted by M^T is obtained by "rotating" the matrix. That is, rows are made into columns. For example:

$$\text{if} \quad \mathbf{x} \;=\; \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{then} \quad \mathbf{x}^T \;=\; \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

- If:

$$M^T \;=\; M$$

then M is said to be symmetric

Matrix Miscellany VI

- Given a vector $\mathbf{x}$, the operation $\mathbf{x}\mathbf{x}^T$ results in in a matrix. For example:

$$\mathbf{x}\mathbf{x}^T = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} x_1^2 & x_1 x_2 \\ x_1 x_2 & x_2^2 \end{bmatrix}$$

But the operation $\mathbf{x}^T\mathbf{x}$ results in a number (scalar):

$$\mathbf{x}^T\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + x_2^2$$

In general, provided the vectors are compatible, $\mathbf{x}\mathbf{y}^T$ is a matrix and $\mathbf{x}^T\mathbf{y}$ is a scalar. The latter is also called the *dot* or *inner* product $\mathbf{x} \cdot \mathbf{y}$ of the vectors. If the dot product of a pair of vectors is 0, then the vectors are said to be *orthogonal*

Matrix Miscellany VII

- An *orthogonal* matrix M is a $n \times n$ matrix whose rows and columns are orthogonal. That is:

$$M^T M = M M^T = I$$

or:

$$M^T = M^{-1}$$

- Let M be a matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. Then

$$M\mathbf{x}_i = \lambda_i \mathbf{x}_i$$

Matrix Miscellany VIII

- Let D be the matrix comprised of the eigenvectors of M. That is, $D = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$. If the *diagonal form* of M is:

$$\Lambda = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix}$$

Then:

$$D\Lambda = MD$$

and:

$$\Lambda = D^{-1}MD$$

D is said to *diagonalise* M. If D is orthogonal, then:

$$\Lambda = D^T MD$$

Matrix Miscellany IX

- If M is a symmetric $n \times n$ matrix, then the eigenvectors corresponding to distinct eigenvalues are orthogonal. Specifically, if the eigenvalues are n distinct real numbers, then there will be n orthogonal eigenvectors
- If M is a symmetric $n \times n$ matrix whose entries are all real numbers, then there exists an orthogonal matrix D whose columns are the eigenvectors of M, such that $D^T MD$ is a diagonal matrix. The entries of the diagonal matrix are the eigenvalues of M.

Data are Matrices I

- Data are in the form of rows of values for each of d dimensions. Each dimension is sometimes also called a *feature* or an *attribute* or an *independent variable*. In addition, there may be an additional value y, denoting a *dependent variable*
- Here, we will only look at cases where the x_i and y values are all numbers. For example:

x_1	x_2	$\cdots$	x_d	y
1.2	5.7	$\cdots$	23.8	2.1
0.95	3.9	$\cdots$	12.3	0.7
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
2.1	3.9	$\cdots$	16.7	4.2

Data are Matrices II

- The values of the attributes in each data row can be represented by the vector of of values:

$$\mathbf{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,d} \end{bmatrix}$$

  or simply $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ to represent any specific $\mathbf{x}$. Each such $\mathbf{x}$ can be taken as the value of a random variable $X = [X_1, X_2, \ldots, X_d]$, whose values are vectors in $\Re^d$. The N rows are then taken to be *independent* observations (a *sample*) of the r.v. X.
- Recall vectors are special $m \times n$ matrices with $n = 1$

Data are Matrices III

- If the data are assumed to bw indendent observations of a d-dimensional random variable X, then their mean is:

$$\mu_X = E[X].$$

  This is simply the mean values of the individual dimensions:

$$\mu_X = [\mu_1, \mu_2, \ldots, \mu_d]^T$$

  where:

$$\mu_i = E[X_i]$$

Data are Matrices IV

- Given N values of X (that is, a sample) $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, an estimate of the mean vector is:

$$\mathbf{m}_x = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$$

This is simply the estimate:

$$\mathbf{m}_x = [m_1, m_2, \ldots, m_d]^T$$

where m_i is the estimated mean of the i^{th} dimension:

$$m_i = \frac{1}{N}\sum_{j=1}^{N} x_{ij}$$

Data are Matrices V

- The variance of values in the i^{th} dimension is:

$$Var(X_i) \;\; \sigma_i^2 \;=\; E[(X_i - \mu_i)^2]$$

with an unbiased estimator:

$$s_i^2 \;=\; \frac{1}{N-1}\sum_{j=1}^{N}(x_{ij} - m_i)^2$$

- The scatter of values of the entire dataset about the the mean μ_x is given by:

$$\Sigma_X = E[(\mathbf{X} - \mu_X)(\mathbf{X} - \mu_X)^T]$$

Data are Matrices VI

We know from "Matrix Miscellany" above that the term inside the $E[\ldots]$ is a matrix. This is the covariance matrix, and it's entries are:

$$\begin{bmatrix} \sigma_1^2 & \sigma_{12}^2 & \ldots & \sigma_{1d}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & \ldots & \sigma_{2d}^2 \\ \vdots & \ldots & & \vdots \\ \sigma_{d1}^2 & \ldots & & \sigma_{dd}^2 \end{bmatrix}$$

The matrix is symmetric and the diagonal terms are the variance of the individual X_i. The off-diagonal term σ_{ij}^2 is the co-variation of X_i and X_j.

Data are Matrices VII

- In a matrix representation:

$$\Sigma_X = E[(\mathbf{X} - \mu_X)(\mathbf{X} - \mu_X)^T]$$

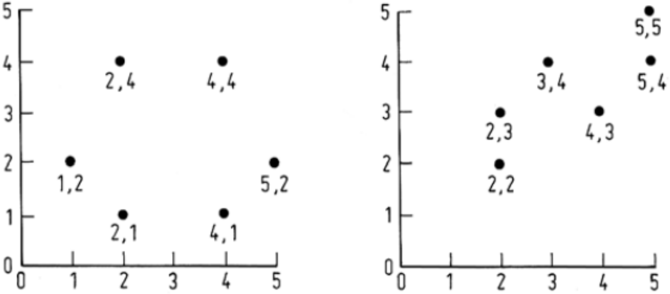
Given a sample $\mathbf{x}_1, \ldots, \mathbf{x}_N$ for $\mathbf{X}$, an estimate of Σ_X is given by:

$$S_x = \frac{1}{N-1} \sum_{i=1}^{N} (\mathbf{x}_i - \mathbf{m}_x)(\mathbf{x}_i - \mathbf{m}_x)^T$$

We will use Σ_X and S_x notation interchangeably, with the understanding that the latter is an estimate of the former. Similarly with μ and $\mathbf{m}$

Data are Matrices VIII

- The covariance matrix is related to correlations in the data. For example, given two datasets (from Richards and Jia, *Remote Sensing Image Analysis*):



the data-values for X_1 and X_2 are said to be *uncorrelated* on the left, and *correlated* on the right.

- The mean vector for the data on the left is:

$$\mathbf{m} = \begin{bmatrix} 3.00 \\ 2.33 \end{bmatrix}$$

and the covariance matrix is estimated as follows:

x	$x - m$	$[x - m]\,[x - m]'$
$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} -2.00 \\ -0.33 \end{bmatrix}$	$\begin{bmatrix} 4.00 & 0.66 \\ 0.66 & 0.11 \end{bmatrix}$
$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -1.00 \\ -1.33 \end{bmatrix}$	$\begin{bmatrix} 1.00 & 1.33 \\ 1.33 & 1.77 \end{bmatrix}$
$\begin{bmatrix} 4 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1.00 \\ -1.33 \end{bmatrix}$	$\begin{bmatrix} 1.00 & -1.33 \\ -1.33 & 1.77 \end{bmatrix}$
$\begin{bmatrix} 5 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 2.00 \\ -0.33 \end{bmatrix}$	$\begin{bmatrix} 4.00 & -0.66 \\ -0.66 & 0.11 \end{bmatrix}$
$\begin{bmatrix} 4 \\ 4 \end{bmatrix}$	$\begin{bmatrix} 1.00 \\ 1.67 \end{bmatrix}$	$\begin{bmatrix} 1.00 & 1.67 \\ 1.67 & 2.79 \end{bmatrix}$
$\begin{bmatrix} 2 \\ 4 \end{bmatrix}$	$\begin{bmatrix} -1.00 \\ 1.67 \end{bmatrix}$	$\begin{bmatrix} 1.00 & -1.67 \\ -1.67 & 2.79 \end{bmatrix}$
		$\Sigma_x = \begin{bmatrix} 2.40 & 0 \\ 0 & 1.87 \end{bmatrix}$

Data are Matrices X

- A *correlation matrix* can be obtained from the covariance matrix with entries:

$$\rho_{ij} = \frac{\sigma_{ij}}{\sigma_{ii}\sigma_{jj}}$$

The correlation matrix for the data on the left is:

$$R = \begin{bmatrix} 1.00 & 0 \\ 0 & 1.00 \end{bmatrix}$$

- For the data on the right, the covariance matrix estimate is (calculate this!):

$$\Sigma = \begin{bmatrix} 1.9 & 1.1 \\ 1.1 & 1.1 \end{bmatrix}$$

Data are Matrices XI

and the correlation matrix is:

$$R = \begin{bmatrix} 1.00 & 0.761 \\ 0.761 & 1.00 \end{bmatrix}$$

That is, dimensions X_1 and X_2 are 76% correlated

- In general, if the data values in any one dimension have no correlation with the data values in any other dimensions, then the covariance (and correlation) matrix will be diagonal

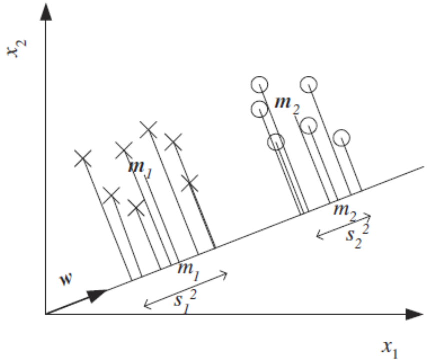
Compressing Data (Special Case) I

(Adapted from Alpaydin, *Introduction to Machine Learning*)

- Consider the special case when the data are known beforehand to be from some groups (or classes). We will only look at the case were data are from 2 classes (dependent variable values"1" and "0"). In general, each data point is a

Compressing Data (Special Case) II

point in d-
dimensional space. A two-dimensional example is shown below:



Compressing Data (Special Case) III

- In Linear Discriminant Analysis (LDA): not to be confused with Latent Dirichlet Allocation), we want to find a direction **w** s.t. when the data are projected onto **w** they are well separated
- LDA thus reduces the d-dimensional data to 1 dimension that allows the best separation into the classes
- The projection of a vector **x** onto a vector **w** is

$$\mathbf{z} = \frac{1}{|\mathbf{w}|}\mathbf{w}^T\mathbf{x}$$

- In the figure m_1 is the projection of $\mathbf{m}_1$ and m_2 is the projection of $\mathbf{m}_2$ onto **w**. We can take **w** to be a unit-vector, since we are only interested in the magnitude of the projections in direction (angle) of **w** That is:

$$m_i = \mathbf{w}^T\mathbf{m}_i$$

Compressing Data (Special Case) IV

- Assuming Class 1 is represented by $y = 1$ and Class 2 is represented by $y = 0$, then, given data with N instances:

$$m_1 = \frac{\sum_{i=1}^{N} \mathbf{w}^T \mathbf{x}_i y_i}{\sum_{i=1}^{N} y_i}$$

and:

$$m_2 = \frac{\sum_{i=1}^{N} \mathbf{w}^T \mathbf{x}_i (1 - y_i)}{\sum_{i=1}^{N} (1 - y_i)}$$

- The scatter of the projected instances about their (projected) means is:

$$s_1^2 = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - m_1)^2 y_i$$

Compressing Data (Special Case) V

and:

$$s_2^2 = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - m_1)^2 (1 - y_i)$$

- We want to find a **w** s.t. m_1 and m_2 are as far apart as possible, and s_1^2 and s_2^2 are as small as possible. That is, we want to maximize:

$$f(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

Compressing Data (Special Case) VI

- The numerator is:

$$\begin{aligned}(m_1 - m_2)^2 &= (\mathbf{w}^T\mathbf{m}_1 - \mathbf{w}^T\mathbf{m}_2)^2 \\ &= \mathbf{w}^T(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T\mathbf{w} \\ &= \mathbf{w}^T \mathrm{S}_{1,2}\mathbf{w}\end{aligned}$$

and the denominator is:

$$\begin{aligned}s_1^2 &= \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - m_1)^2 \\ &= \sum_i \mathbf{w}^T(\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^T\mathbf{w} \\ &= \mathbf{w}^T\mathrm{S}_1\mathbf{w}\end{aligned}$$

where $\mathrm{S}_1 = \sum_i(\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^T$.

Compressing Data (Special Case) VII

Similarly:

$$s_2^2 \;=\; \mathbf{w}^T \mathbf{S}_2 \mathbf{w}$$

and the denominator is:

$$s_1^2 + s_2^2 \;=\; \mathbf{w}^T (\mathrm{S}_1 + \mathrm{S}_2) \mathbf{w}$$

- Let $\mathrm{S} = \mathrm{S}_1 + \mathrm{S}_2$. Recall we want to minimise:

$$f(\mathbf{w}) \;=\; \frac{\mathbf{w}^T \mathrm{S}_{1,2} \mathbf{w}}{\mathbf{w} \mathrm{S} \mathrm{w}}$$

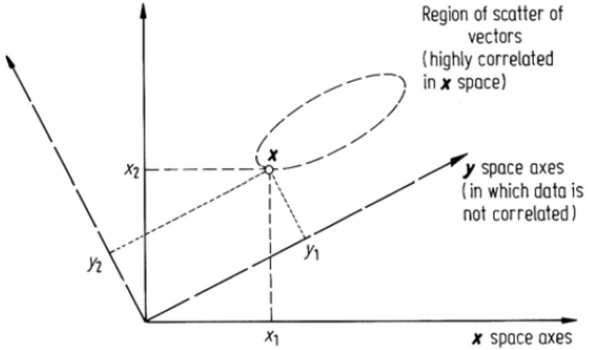
- It can be shown that if the partial derivation of f w.r.t. $\mathbf{w}$ is 0 then:

$$\mathbf{w} = \mathbf{S}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$$

Compressing Data (General Case) I

- We now consider the problem of reducing the dimensions when classes are not known (or do not exist)
- If data are correlated, then correlated dimensions represent redundancy. We can try to remove redundant dimensions by defining a new coordinate system of orthogonal axes. For example, in the $\mathbf{y}$ coordinate system below, all the variation is largely along y_1, and y_2 can be taken to be mostly "noise":

Compressing Data (General Case) II



Compressing Data (General Case) III

The new $\mathbf{y}$ coordinate system is a rotation (i.e. a linear combination) of the old coordinates. For example, the point (x_1, x_2) in $\mathbf{x}$-space is the point (y_1, y_2) in $\mathbf{y}$-space, where:

$$y_1 = w_{11}x_1 + w_{12}x_2$$
$$y_2 = w_{21}x_1 + w_{22}x_2$$

- In general, assume the directions of the new coordinate space are $\mathbf{w}_1$ and $\mathbf{w}_2, \ldots \mathbf{w}_d$. Then, given any point $\mathbf{x}$ in the original coordinates the i^{th} dimension in the new coordinates is:

$$y_i \;=\; w_{i1}x_1 + \cdots + w_{id}x_d \;=\; \mathbf{w}_i^T\mathbf{x}$$

More generally:

$$Y_i = \mathbf{w}_i^T\mathbf{X}$$

Compressing Data (General Case) IV

and:

$$\mathbf{Y} = W^T \mathbf{X}$$

where $W = [\mathbf{w}_1, \ldots, \mathbf{w}_d]$

- We want to rotate the $\mathbf{x}$-axes s.t. the data are uncorrelated in $\mathbf{y}$-space.
- That is, we want to find a matrix W s.t.

$$\mathbf{Y} = W^T \mathbf{X}$$

  and the covariance matrix Σ_Y is diagonal
- Now, we know:

$$\Sigma_Y = E[(\mathbf{Y} - \mu_Y)(\mathbf{Y} - \mu_Y)^T]$$

Compressing Data (General Case) V

- If $\mathbf{Y} = W^T\mathbf{X}$, then:

$$\mu_y = E[W^T\mathbf{X}] \;=\; W^T\mu_X$$

and:

$$\begin{aligned}
\Sigma_Y &= E[(W^T\mu_X - W^T\mu_X)(W^T\mu - W^T\mu_X)^T] \\
&= W^T\;E[(\mathbf{X}-\mu_X)(\mathbf{X}-\mu_X)^T](W^T)^T \\
&= W^T\Sigma_X W
\end{aligned}$$

- Now: (a) Since the data are uncorrelated in **y**-space, Σ_Y is diagonal; and (b) Since Σ_X is a symmetric matrix with real values, We know that there exists an orthogonal matrix W consisting of the eigenvectors of Σ_X that results in a diagonal matrix containing the eigenvalues of Σ_X

Compressing Data (General Case) VI

- So, the matrix W consisting of the eigenvectors of Σ_X as columns, and the transformation

$$\mathbf{Y} = W^T \mathbf{X}$$

  will result in new axes in which the covariance matrix is diagonal
- By definition of a covariance matrix, the diagonal elements of Σ_Y are the variances of data along the corresponding dimension in the new coordinate space.
- Dimensions with low variance (in the new space) contain little information and can discarded
- Ordering dimensions by decreasing eigenvalues of Σ_X thus gives one way of identifying useful dimensions (in the new coordinate space). These useful dimensions are called the *principal components*

Compressing Data (General Case) VII

- Continuing the example from Richards and Jia, the graph on the right seen earlier, recall the covariance matrix was:

$$\Sigma_X = \begin{bmatrix} 1.9 & 1.1 \\ 1.1 & 1.1 \end{bmatrix}$$

- The eigenvectors of Σ_X are solutions to equations arising from:

$$det(\Sigma_X - \lambda I) = 0$$

or:

$$\begin{vmatrix} 1.9 - \lambda & 1.1 \\ 1.1 & 1.1 - \lambda \end{vmatrix} = 0$$

That is:

$$\lambda_1^2 - 3\lambda + 0.88 = 0$$

Compressing Data (General Case) VIII

or $\lambda = \lambda_1 = 2.67$ and $\lambda = \lambda_2 = 0.33$. The eigenvector corresponding to λ_1 is the vector $\mathbf{w}_1 = [w_{11}, w_{21}]^T$ that satisfies:

$$(\Sigma_X - \lambda_1 I)\mathbf{w}_1 = 0$$

This gives 2 equations:

$$\begin{aligned} -0.77w_{11} + 1.1w_{21} &= 0 \\ 1.1w_{11} - 1.57w_{21} &= 0 \end{aligned}$$

In addition, if we impose the constraint for the eigenvectors to be unit-vectors, then $w_{11}^2 + w_{21}^2 = 1$. Solving all these constraints simultaneously gives:

$$\mathbf{w}_1 = \begin{bmatrix} 0.82 \\ 0.57 \end{bmatrix}$$

Compressing Data (General Case) IX

Similarly, using λ_2:

$$\mathbf{w}_2 = \begin{bmatrix} -0.57 \\ 0.82 \end{bmatrix}$$

- With $W = [\mathbf{w}_1, \mathbf{w}_2]$, the transformation matrix is:

$$\text{``}W^T = \begin{bmatrix} 0.82 & -0.57 \\ 0.57 & 0.82 \end{bmatrix}^T = \begin{bmatrix} 0.82 & 0.57 \\ -0.57 & 0.82 \end{bmatrix}$$

That is, the *principal components transformation* is:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.82 & 0.57 \\ -0.57 & 0.82 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Describing Data (Special Case) I

(Adapted from Rogers and Girolami, *A First Course in Machine Learning*)

- We will now look at the special case of describing a set of data points using a linear model with the following structure:

$$y = f(x_1, x_2, \ldots, x_d) = w_0 + w_1 x_1 + \cdots + w_d x_d$$

- What should the values of the w_i be to describe the data well? One possible solution is to select the w_i to minimise the errors made by describing the data using the model. In general, this is a form of optimisation that minimises a *loss function* $\mathcal{L}$. A common loss function to measure error made by a model is the squared-loss function.

Describing Data (Special Case) II

- To find the w_i that minimise any loss function $\mathcal{L}$ will require obtaining the partial derivatives of $\mathcal{L}$ w.r.t. the w_i, and setting each partial derivative to 0. With d-dimensional data, this will result in d equations. Solving these d equations simultaneously, will give us the values of the w_i. But:
  - Solving the equations will require values from the data
  - With large d, this representation is cumbersome
- We now consider vectors of the form $\mathbf{x} = [1, x_1, x_2, \ldots, x_d]^T$ (this is an *extended* form of the vector representation of a row of data $[x_1, x_2, \ldots, x_d]^T$)

Describing Data (Special Case) III

- An extended representation of the data is then:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ \dots & \dots & \dots & \dots & \dots \\ N & x_{N,1} & x_{N,2} & \dots & x_{N,d} \end{bmatrix}$$

  This is a $N \times (d+1)$ matrix.

- The values of the linear model for each data instance can be written as:

$$f(x_1, x_2, \dots, x_d) = \mathbf{Xw}$$

  The r.h.s. is a $N \times 1$ matrix since $\mathbf{w}$ is a $(d+1) \times 1$ matrix i.e. $\mathbf{w} = [w_0, w_1, w_2, \dots, w_d]$.

Describing Data (Special Case) IV

- The difference between y's and the model's value for each data instance can can be represented compactly as:

$$\mathbf{y} - \mathbf{X}\mathbf{w} = \begin{bmatrix} y_1 - (w_0 + w_1 x_{1,1} + \cdots + w_d x_{1,d}) \\ y_2 - (w_0 + w_1 x_{2,1} + \cdots + w_d x_{2,d}) \\ \cdots y_N - (w_0 + w_1 x_{N,1} + \cdots + w_d x_{N,d}) \end{bmatrix}$$

  This is also a $N \times 1$ matrix (or a N-dimensional vector). The mean squared-loss incurred by the model is then

$$\mathcal{L} \;=\; \frac{1}{N} \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_{i,1} + \cdots + w_d x_{i,d}))^2$$

Describing Data (Special Case) V

- For an N-dimensional vector

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}$$

Recall:

$$\mathbf{v} \cdot \mathbf{v} = \mathbf{v}^T \mathbf{v} = \sum_{1}^{N} v_i^2$$

where $\mathbf{u} \cdot \mathbf{v}$ denotes the inner-product of the vectors $\mathbf{u}$ and $\mathbf{v}$

Describing Data (Special Case) VI

- Then, the mean-squared loss is

$$\mathcal{L} = \frac{1}{N}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

This is also the same as:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Describing Data (Special Case) VII

- We can simplify the squared-loss expression further:

$$
\begin{aligned}
\mathcal{L} &= \frac{1}{N}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w}) \\
&= \frac{1}{N}(\mathbf{X}\mathbf{w}-\mathbf{y})^T(\mathbf{X}\mathbf{w}-\mathbf{y}) \\
&= \frac{1}{N}((\mathbf{X}\mathbf{w})^T-\mathbf{y}^T)(\mathbf{X}\mathbf{w}-\mathbf{y}) \\
&= \frac{1}{N}\left[(\mathbf{X}\mathbf{w})^T\mathbf{X}\mathbf{w}-\mathbf{y}^T\mathbf{X}\mathbf{w}-(\mathbf{X}\mathbf{w})^T\mathbf{y}+\mathbf{y}^T\mathbf{y}\right] \\
&= \frac{1}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}-\frac{2}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{y}+\frac{1}{N}\mathbf{y}^T\mathbf{y}
\end{aligned}
$$

(The terms $\mathbf{y}^T\mathbf{X}\mathbf{w}$ and $\mathbf{w}^T\mathbf{X}^T\mathbf{y}$ are transposes of each other and scalars, and therefore equal)

Describing Data (Special Case) VIII

- Differentiating w.r.t $\mathbf{w}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}} \left[\frac{1}{N} \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{w}^T \mathbf{X}^T \mathbf{y} + \frac{1}{N} \mathbf{y}^T \mathbf{y} \right]$$

- Here are some useful identities:

f	$\partial f / \partial \mathbf{w}$
$\mathbf{w}^T \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^T \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^T \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^T \mathbf{A} \mathbf{w}$	$2\mathbf{A}\mathbf{w}$

- Then:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{2}{N} \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^T \mathbf{y}$$

Describing Data (Special Case) IX

- Equating to $\mathbf{0}$ gives:

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

or:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Dimensionality Reduction using PCA I

Overview:

- Let d be the number of features in each datapoint.
- Dimensionality Reduction: Find fewer features (less than d) such that they are representative of the original d features.
- Principal Component Analysis (PCA) gives us new features (principal components) that are linear combinations of the original features.
- Principal components can be sorted by the amount of variance in the data that they can explain.
- Principal components form an orthonormal basis of the d dimensional feature space.
- p principal components ($p < d$) are chosen such that maximum variance in the data is captured.

Dimensionality Reduction using PCA II

- Let $\mathbf{X}$ be an $n \times d$ data matrix that contains n datapoints having d feature dimensions.
- Mean centering: The average value of each of the d features is subtracted from the rows of $\mathbf{X}$. This produces a dataset such that the mean of the rows is $\vec{0}$.
- We will assume that the data matrix $\mathbf{X}$ is mean centered.
- Covariance between features f_1 and f_2:
  $COV(f_1, f_2) = \frac{1}{n-1} \sum_{i=1}^{n} (f_1 - \bar{f_1})(f_2 - \bar{f_2})$
- Step 1: Find covariance matrix $cov = \frac{1}{n-1} \mathbf{X^T X}$
  Note: If the scales of the features vary widely, then compute the correlation matrix instead of covariance matrix in Step 1.
- The $(i, j)^{th}$ element of cov matrix will contain the covariance between i^{th} and j^{th} features.

Dimensionality Reduction using PCA III

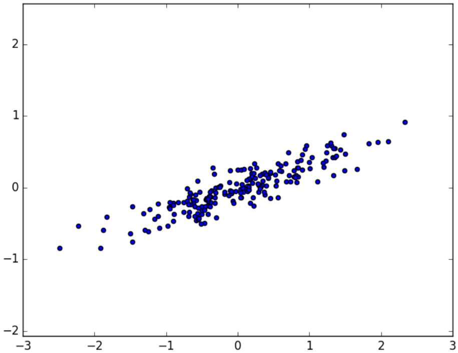
- $rank(cov) = rank(\mathbf{X}) = r$ (because $rank(\mathbf{X}^\mathsf{T}\mathbf{X}) = rank(\mathbf{X})$).
- eigenvectors of cov will be orthogonal (because cov is symmetric).
- Step 2: Find the eigenvalues and orthonormal eigenvectors of cov. Order the eigenvectors by their eigenvalues from highest to lowest.
- The eigenvectors corresponding to large eigenvalues are more significant, and capture more variation in the data.
- Step 3: Select p eigenvectors (principal components) corresponding to the p largest eigenvalues.

Dimensionality Reduction using PCA IV

- Step 4: Construct a matrix $\mathbf{V}$ with p orthonormal eigenvectors as its columns.
- Step 5: Project the data matrix $\mathbf{X}$ on to the space spanned by the p eigenvectors in $\mathbf{V}$.
  $\mathbf{X_{new}} = \mathbf{XV}$
- $\mathbf{X_{new}}$ is the new data matrix with each datapoint having p dimensions.

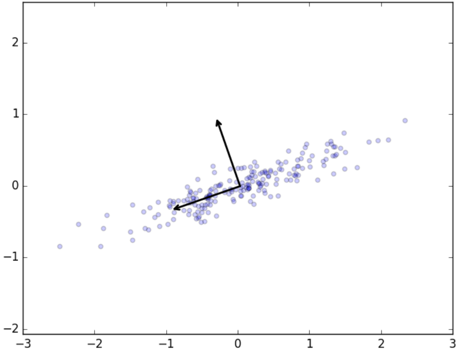
Dimensionality Reduction: An Example I

- Let $\mathbf{X}$ be mean centered 200×2 datamatrix. Each datapoint has two dimensions.



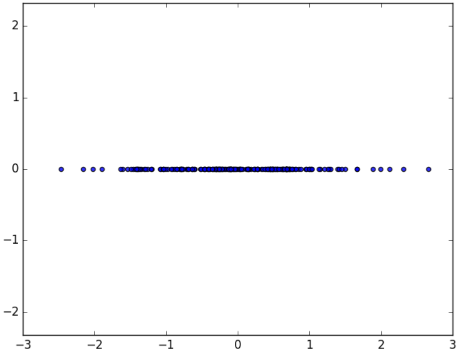
Dimensionality Reduction: An Example II

- Figure shows the orthonormal eigenvectors of *cov* matrix.



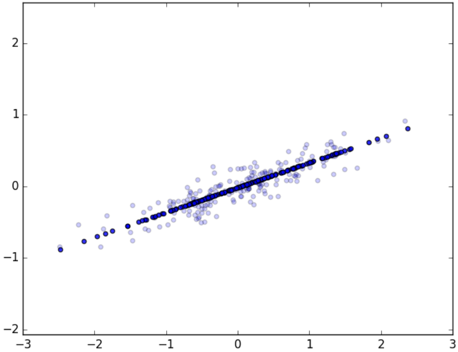
Dimensionality Reduction: An Example III

- Figure shows vectors in $\mathbf{X_p} = \mathbf{XV}$, where $\mathbf{V}$ is 2×1 matrix containing the eigenvector corresponding to the largest eigenvalue. The dimensionality is reduced while retaining maximum possible variation in the data.



Dimensionality Reduction: An Example IV

- The reduced data $\mathbf{X_p}$ can be moved back to the original two dimensional feature space using $\mathbf{X_p V^T}$. The figure shows datapoints in $\mathbf{X_p V^T}$ as dark dots, and those in the original data matrix $\mathbf{X}$ as light dots.



Linear Algebra in Programming

Next, we look at how linear algebra allows us to write vectorised computer programs for machine- and deep learning, which exploit the power of SIMD/MIMD architectures such as CPUs and GPUs.

Scalars, Vectors, Matrices, Tensors I

Scalars A scalar is just a single number.

Example 1

$x = 3.5$. Here, x is a scalar; $x \in \mathbb{R}$.

Example 2

Similarly, Iris dataset has $d = 4$ features. Here, $d \in \mathbb{N}$ is a scalar.

Sometimes, we write a scalar as $(a)_{1\times 1}$.

Scalars, Vectors, Matrices, Tensors II

Vectors A vector is an array of number.

Example

$\mathbf{x} = [1.3, -4.0, 11.7, 4]^{\mathsf{T}}$ is a vector; $\mathbf{x} \in \mathbb{R}^4$.

A d-dimensional vector is written as $\mathbf{x} \in \mathbb{R}^d$ and in matrix form as: $(\cdots)_{d \times 1}$.

In a programming convention, we refer $(\cdots)_{d \times 1}$ as an 1-D array with d-elements.

Scalars, Vectors, Matrices, Tensors III

Matrices A matrix is a 2-D array of numbers.

Example

$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 4 & 6 \\ 3 & 6 & 8 \end{bmatrix}$ is a 3×3 matrix.

We write a matrix as $\mathbf{A} \in \mathbb{R}^{m \times n}$.

We read this as: $\mathbf{A}$ is a real-valued matrix of order $m \times n$, where m is number of rows, n is number of columns.

An element of a matrix is referenced as $a_{i,j} \in \mathbf{A}$, where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

Tensors In some cases we will need an array with more than two axes (dimensions). A tensor is an array of numbers arranged on a regular grid with a variable number of axes.

Example

A photo that you clicked is stored as a tensor: (3-depth; RGB) or (4-depth; CMYK).

In an image, the first two axes are *height* and *width* of the image and the third axis refers to *depth* (no. of color channels).

Norms I

- Norm measures the size of a vector.

Norms I

- Norm measures the size of a vector.
- Formally, L^p norm is given as

$$||\mathbf{x}||_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

for $p \in \mathbb{R}$, $p \geq 1$.

Norms I

- Norm measures the size of a vector.
- Formally, L^p norm is given as

$$||\mathbf{x}||_p = \left(\sum_i |x_i|^p\right)^{\frac{1}{p}}$$

  for $p \in \mathbb{R}$, $p \geq 1$.
- Norm is a function that maps a vector to a non-negative value.

Norms I

- Norm measures the size of a vector.
- Formally, L^p norm is given as

$$||\mathbf{x}||_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

  for $p \in \mathbb{R}$, $p \geq 1$.
- Norm is a function that maps a vector to a non-negative value.
- Intuitively, norm measures the distance of a vector $\mathbf{x}$ from origin.

Norms II

Norm is any function f that satisfies the following properties:

- $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$

Norms II

Norm is any function f that satisfies the following properties:

- $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$
- $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ (*triangle inequality*)

Norms II

Norm is any function f that satisfies the following properties:

- $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$
- $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ (*triangle inequality*)
- $\forall a \in \mathbb{R}, f(a\mathbf{x}) = |a| f(\mathbf{x})$

Norms III

Euclidean norm Most frequently, we will be using L^2 norm in ML. It is enoted as $||\mathbf{x}||$.

$$||\mathbf{x}|| = \mathbf{x}^\mathsf{T}\mathbf{x}$$

L^1 **norm** There will be situations in ML where difference between a zero and a non-zero element is very important. In such cases, we will use L^1 norm.

$$||\mathbf{x}||_1 = \sum_i |x_i|$$

i.e. every time an element x_i moves ϵ-away from 0, the norm increases by ϵ.

Max norm It is called L^∞ norm, which is calculated as

$$||\mathbf{x}||_\infty = \max_i |x_i|$$

Norms IV

Frobenius norm It measures the size of a matrix.

This norm is used most frequently in deep learning.

$$||\mathbf{A}||_F = \left(\sum_{i,j} a_{i,j}^2 \right)^{\frac{1}{2}}$$

Relationship between dot product and norm The dot product of two vectors **x** and **y** can be written as

$$\mathbf{x}^\mathsf{T}\mathbf{y} = ||\mathbf{x}||_2 ||\mathbf{y}||_2 \cos\theta$$

where, θ is angle between the two vectors.

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)
- A multi-core processor is MIMD. A Graphics Processing Unit (GPU) is SIMD.

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)
- A multi-core processor is MIMD. A Graphics Processing Unit (GPU) is SIMD.
- Deep Learning is a problem for which SIMD is well-suited.

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)
- A multi-core processor is MIMD. A Graphics Processing Unit (GPU) is SIMD.
- Deep Learning is a problem for which SIMD is well-suited.
- Example: You want to compute non-linear activation of a matrix:
  1. Either you can call the `transform()` operation for each element. Total $m \times n$ calls.
  2. Or, call the `transform()` operation once for whole matrix.

Parallelism II

Given a matrix $\mathbf{Z}$, compute its non-linear activation $\sigma(\mathbf{Z})$.

1. Using explicit for loop:

```
A = np.zeros(Z.shape)
for i in range(0, Z.shape[0]):
    for j in range(0, Z.shape[1]):
        A[i][j] = 1 / (1 + np.exp(-Z[i][j]))
```

2. Using **vectorisation**:

```
A = 1 / (1 + np.exp(-Z))
```

Vectorisation I

Let $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{w} \in \mathbb{R}^d$, $b \in R$. We want to compute $z = \mathbf{w}^\mathsf{T}\mathbf{x} + b$.

1. Without vectorisation

```
z = 0.
for i in range(d):
    z += W[i] * X[i]
z += b
```

2. With vectorisation

```
z = np.dot(W, X) + b
```

Vectorisation II

Computation time (64GB RAM, 16-core Intel Xeon CPU, 3.10GHz):

d	non-vec(s)	vec(s)
10^3	6.1×10^{-4}	2.6×10^{-5}
10^6	0.622	0.003
10^7	6.099	0.008
10^8	55.191	0.081
10^9	–	0.487

– : couldn't wait!

Vectorisation III

Vectorised matrix multiplication ($A, B \in \mathbb{R}^{d \times d}$). Same machine with 8GB GPGPU. Comparing CPU and GPU:

d	CPU(s)	GPU(s)
10^4	5.600	0.001

*Both results are obtained using PyTorch.