CHAPTER 14

Nearest Neighbour Classification

Often when faced with a classification problem it is useful to first employ a simple method to produce a baseline against which more complex methods can be compared. In this chapter we discuss the simple nearest neighbour method. The nearest neighbour methods are extremely popular and can perform surprisingly well. We also discuss how these methods are related to probabilistic mixture models.

14.1 Do As Your Neighbour Does

Successful prediction typically relies on smoothness in the data – if the class label can change as we move a small amount in the input space, the problem is essentially random and no algorithm will generalise well. In machine learning one constructs appropriate measures of smoothness for the problem at hand and hopes to exploit this to obtain good generalisation. Nearest neighbour methods are a useful starting point since they readily encode basic smoothness intuitions and are easy to program.

In a classification problem each input vector $\mathbf{x}$ has a corresponding class label, $c^n \in \{1, \ldots, C\}$. Given a dataset of N train examples, $\mathcal{D} = \{\mathbf{x}^n, c^n\}, n = 1, \ldots, N$, and a novel $\mathbf{x}$, we aim to return the correct class $c(\mathbf{x})$. A simple, but often effective, strategy for this supervised learning problem can be stated as: for novel $\mathbf{x}$, find the nearest input in the train set and use the class of this nearest input, algorithm(14.1). For vectors $\mathbf{x}$ and $\mathbf{x}'$ representing two different datapoints, we measure 'nearness' using a *dissimilarity function* $d(\mathbf{x}, \mathbf{x}')$. A common dissimilarity is the *squared Euclidean distance*

$$d(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^{\mathsf{T}}(\mathbf{x} - \mathbf{x}') \tag{14.1.1}$$

which can be more conveniently written $(\mathbf{x} - \mathbf{x}')^2$. Based on the squared Euclidean distance, the decision boundary is determined by the lines which are the perpendicular bisectors of the closest training points with different training labels, see fig(14.1). This partitions the input space into regions classified equally and is called a *Voronoi tessellation*.

The nearest neighbour algorithm is simple and intuitive. There are, however, some issues:

- How should we measure the distance between points? Whilst the Euclidean square distance is popular, this may not always be appropriate. A fundamental limitation of the Euclidean distance is that it does not take into account how the data is distributed. For example if the length scales of $\mathbf{x}$ vary greatly the largest length scale will dominate the squared distance, with potentially useful class-specific information in other components of $\mathbf{x}$ lost. The *Mahalanobis distance*

  $$d(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x} - \mathbf{x}'\right)^{\mathsf{T}} \mathbf{\Sigma}^{-1} \left(\mathbf{x} - \mathbf{x}'\right) \tag{14.1.2}$$

  where $\mathbf{\Sigma}$ is the covariance matrix of the inputs (from all classes) can overcome some of these problems since it rescales all length scales to be essentially equal.

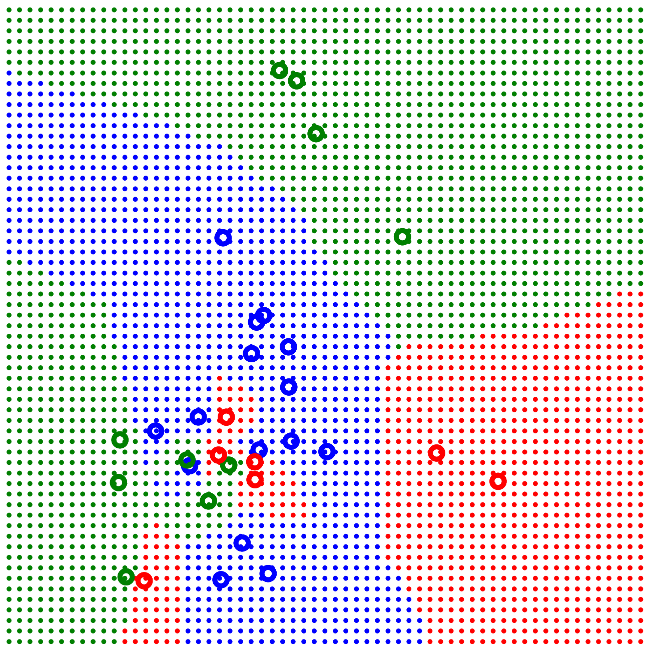


Figure 14.1: In nearest neighbour classification a new vector is assigned the label of the nearest vector in the training set. Here there are three classes, with training points given by the circles, along with their class. The dots indicate the class of the nearest training vector. The decision boundary is piecewise linear with each segment corresponding to the perpendicular bisector between two datapoints belonging to different classes, giving rise to a Voronoi tessellation of the input space.

Algorithm 14.1 Nearest neighbour algorithm to classify a vector $\mathbf{x}$, given train data $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$:

```
1: Calculate the dissimilarity of the test point $\mathbf{x}$ to each of the train points, $d^n = d(\mathbf{x}, \mathbf{x}^n)$, $n = 1, \ldots, N$.
2: Find the train point $\mathbf{x}^{n^*}$ which is nearest to $\mathbf{x}$ :
       $n^* = \underset{n}{\text{argmin}}\ d(\mathbf{x}, \mathbf{x}^n)$
3: Assign the class label $c(\mathbf{x}) = c^{n^*}$.
4: In the case that there are two or more nearest neighbours with different class labels, the most numerous class
   is chosen. If there is no one single most numerous class, we use the $K$-nearest-neighbours.
```

- The whole dataset needs to be stored to make a classification since the novel point must be compared to all of the train points. This can be partially addressed by a method called data editing in which datapoints which have little or no effect on the decision boundary are removed from the training dataset. Depending on the geometry of the training points, finding the nearest neighbour can also be accelerated by examining the values of each of the components x_i of $\mathbf{x}$ in turn. Such an axis-aligned space-split is called a *KD-tree*[205] and can reduce the possible set of candidate nearest neighbours in the training set to the novel $\mathbf{x}^*$, particularly in low dimensions.
- Each distance calculation can be expensive if the datapoints are high dimensional. Principal Components Analysis, see chapter(15), is one way to address this and replaces $\mathbf{x}$ with a low dimensional projection $\mathbf{p}$. The Euclidean distance of two datapoints $\left(\mathbf{x}^a - \mathbf{x}^b\right)^2$ is then approximately given by $\left(\mathbf{p}^a - \mathbf{p}^b\right)^2$, see section(15.2.4). This is both faster to compute and can also improve classification accuracy since only the large scale characteristics of the data are retained in the PCA projections.
- It is not clear how to deal with missing data or incorporate prior beliefs and domain knowledge.

14.2 *K*-Nearest Neighbours

If your neighbour is simply mistaken (has an incorrect training class label), or is not a particularly representative example of his class, then these situations will typically result in an incorrect classification. By including more than the single nearest neighbour, we hope to make a more robust classifier with a smoother decision boundary (less swayed by single neighbour opinions). If we assume the Euclidean distance as the dissimilarity measure, the K-Nearest Neighbour algorithm considers a hypersphere centred on the test point $\mathbf{x}$. The radius of the hypersphere is increased until it contains exactly K train inputs. The class label $c(\mathbf{x})$ is then given by the most numerous class within the hypersphere.

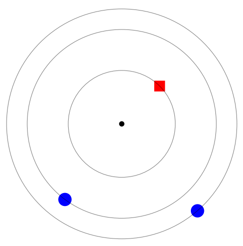


Figure 14.2: In K-nearest neighbours, we centre a hypersphere around the point we wish to classify (here the central dot). The inner circle corresponds to the nearest neighbour. However, using the 3 nearest neighbours, we find that there are two round-class neighbours and one square-class neighbour– and we would therefore classify the central point as round-class. In the case of a tie, one may increase K until the tie is broken.

Choosing K

Whilst there is some sense in making $K > 1$, there is certainly little sense in making $K = N$ (N being the number of training points). For K very large, all classifications will become the same – simply assign each novel $\mathbf{x}$ to the most numerous class in the train data. This suggests that there is an optimal intermediate setting of K which gives the best generalisation performance. This can be determined using cross-validation, as described in section(13.2.2).

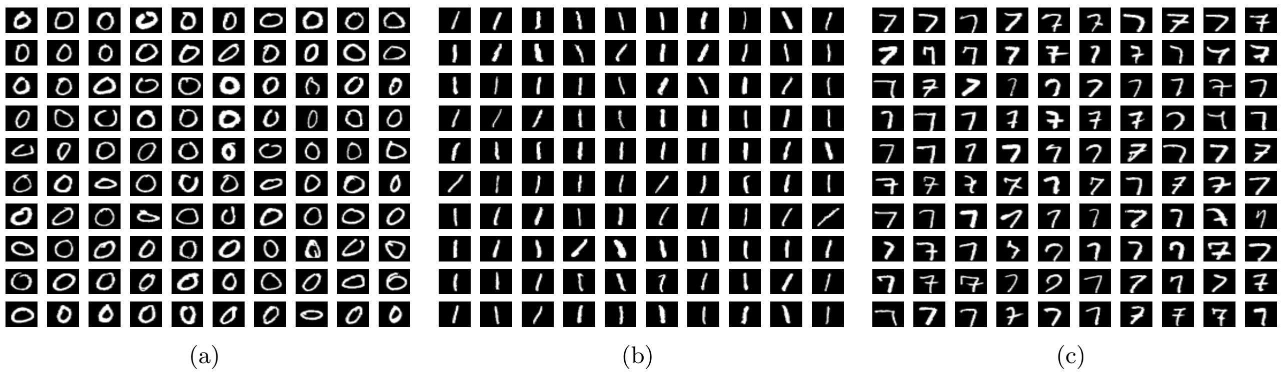


Figure 14.3: Some of the train examples of the digit zero and (a), one (b) and seven (c). There are 300 train examples of each of these three digit classes.

Example 14.1 (Handwritten Digit Example)**.** Consider two classes of handwritten digits, zeros and ones. Each digit contains $28 \times 28 = 784$ pixels. The train data consists of 300 zeros, and 300 ones, a subset of which are plotted in fig(14.3a,b). To test the performance of the nearest neighbour method (based on Euclidean distance) we use an independent test set containing a further 600 digits. The nearest neighbour method, applied to this data, correctly predicts the class label of all 600 test points. The reason for the high success rate is that examples of zeros and ones are sufficiently different that they can be easily distinguished.

A more difficult task is to distinguish between ones and sevens. We repeat the above experiment, now using 300 training examples of ones, and 300 training examples of sevens, fig(14.3b,c). Again, 600 new test examples (containing 300 ones and 300 sevens) were used to assess the performance. This time, 18 errors are found using nearest neighbour classification – a 3% error rate for this two class problem. The 18 test points on which the nearest neighbour method makes errors are plotted in fig(14.4). If we use $K = 3$ nearest neighbours, the classification error reduces to 14 – a slight improvement. As an aside, the best machine learning methods classify real world digits (over all 10 classes) to an error of less than 1% (`yann.lecun.com/exdb/mnist`) – better than the performance of an 'average' human.

14.3 A Probabilistic Interpretation of Nearest Neighbours

Consider the situation where we have data from two classes – class 0 and class 1. We make the following mixture model for data from class 0:

$$p(\mathbf{x}|c=0) = \frac{1}{N_0} \sum_{n \in \text{ class } 0} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \sigma^2\mathbf{I}\right) = \frac{1}{N_0}\frac{1}{(2\pi\sigma^2)^{D/2}} \sum_{n \in \text{ class } 0} e^{-(\mathbf{x}-\mathbf{x}^n)^2/(2\sigma^2)} \tag{14.3.1}$$

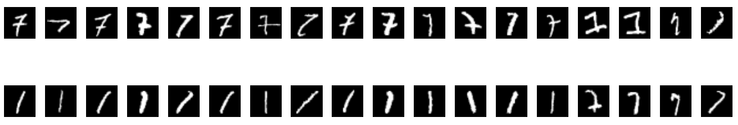


Figure 14.4: '1' versus '7' classification using the NN method. (Top) The 18 out of 600 test examples that are incorrectly classified; (Bottom) the nearest neighbours in the training set corresponding to each test-point above.

where D is the dimension of a datapoint $\mathbf{x}$ and N_0 are the number of train points of class 0, and σ^2 is the variance. This is a *Parzen estimator*, and models the data as a uniform weighted sum of distributions centred on the training points, fig(14.5).

Similarly, for data from class 1:

$$p(\mathbf{x}|c=1) = \frac{1}{N_1}\sum_{n\in \text{ class } 1} \mathcal{N}\left(\mathbf{x}|\mathbf{x}^n, \sigma^2\mathbf{I}\right) = \frac{1}{N_1}\frac{1}{\left(2\pi\sigma^2\right)^{D/2}}\sum_{n\in \text{ class } 1} e^{-(\mathbf{x}-\mathbf{x}^n)^2/(2\sigma^2)} \tag{14.3.2}$$

To classify a new datapoint $\mathbf{x}^*$, we use Bayes' rule

$$p(c=0|\mathbf{x}^*) = \frac{p(\mathbf{x}^*|c=0)p(c=0)}{p(\mathbf{x}^*|c=0)p(c=0) + p(\mathbf{x}^*|c=1)p(c=1)} \tag{14.3.3}$$

The Maximum Likelihood setting of $p(c=0)$ is $N_0/(N_0+N_1)$, and $p(c=1) = N_1/(N_0+N_1)$. An analogous expression to equation (14.3.3) holds for $p(c=1|\mathbf{x}^*)$. To see which class is most likely we may use the ratio

$$\frac{p(c=0|\mathbf{x}^*)}{p(c=1|\mathbf{x}^*)} = \frac{p(\mathbf{x}^*|c=0)p(c=0)}{p(\mathbf{x}^*|c=1)p(c=1)} \tag{14.3.4}$$

If this ratio is greater than one, we classify $\mathbf{x}^*$ as 0, otherwise 1.

Equation(14.3.4) is a complicated function of $\mathbf{x}^*$. However, if σ^2 is very small, the numerator, which is a sum of exponential terms, will be dominated by that term for which datapoint $\mathbf{x}^{n_0}$ in class 0 is closest to the point $\mathbf{x}^*$. Similarly, the denominator will be dominated by that datapoint $\mathbf{x}^{n_1}$ in class 1 which is closest to $\mathbf{x}^*$. In this case, therefore,

$$\frac{p(c=0|\mathbf{x}^*)}{p(c=1|\mathbf{x}^*)} \approx \frac{e^{-(\mathbf{x}^*-\mathbf{x}^{n_0})^2/(2\sigma^2)}p(c=0)/N_0}{e^{-(\mathbf{x}^*-\mathbf{x}^{n_1})^2/(2\sigma^2)}p(c=1)/N_1} = \frac{e^{-(\mathbf{x}^*-\mathbf{x}^{n_0})^2/(2\sigma^2)}}{e^{-(\mathbf{x}^*-\mathbf{x}^{n_1})^2/(2\sigma^2)}} \tag{14.3.5}$$

Taking the limit $\sigma^2 \to 0$, with certainty we classify $\mathbf{x}^*$ as class 0 if $\mathbf{x}^*$ has a point in the class 0 data which is closer than the closest point in the class 1 data. The nearest (single) neighbour method is therefore recovered as the limiting case of a probabilistic generative model, see fig(14.5).

The motivation for K nearest neighbours is to produce a classification that is robust against unrepresentative single nearest neighbours. To ensure a similar kind of robustness in the probabilistic interpretation, we may use a finite value $\sigma^2 > 0$. This smoothes the extreme probabilities of classification and means that more points (not just the nearest) will have an effective contribution in equation (14.3.4). The extension to more than two classes is straightforward, requiring a class conditional generative model for each class.

By using a richer generative model of the data we may go beyond the Parzen estimator approach. We will examine such cases in some detail in later chapters, in particular chapter(20).

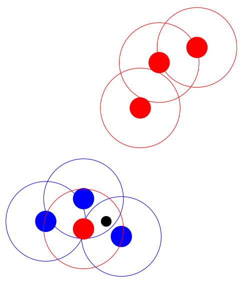


Figure 14.5: A probabilistic interpretation of nearest neighbours. For each class we use a mixture of Gaussians to model the data from that class $p(\mathbf{x}|c)$, placing at each training point an isotropic Gaussian of width σ^2. The width of each Gaussian is represented by the circle. In the limit $\sigma^2 \to 0$ a novel point (black) is assigned the class of its nearest neighbour. For finite $\sigma^2 > 0$ the influence of non-nearest neighbours has an effect, resulting in a soft version of nearest neighbours.

14.3.1 When your nearest neighbour is far away

For a novel input $\mathbf{x}^*$ that is far from all training points, Nearest Neighbours, and its soft probabilistic variant will confidently classify $\mathbf{x}^*$ as belonging to the class of the nearest training point. This is arguably opposite to what we would like, namely that the classification should tend to the prior probabilities of the class based on the number of training data per class. A way to avoid this problem is, for each class, to include a fictitious mixture component at the mean of all the data with large variance, equal for each class. For novel inputs close to the training data, this extra fictitious datapoint will have no appreciable effect. However, as we move away from the high density regions of the training data, this additional fictitious component will dominate. Since the distance from $\mathbf{x}^*$ to each fictitious class point is the same, in the limit that $\mathbf{x}^*$ is far from the training data, the effect is that no class information from the position of $\mathbf{x}^*$ occurs. See section(20.3.3) for an example.

14.4 Code

`nearNeigh.m`: K Nearest Neighbour

14.4.1 Utility Routines

`majority.m`: Find the majority entry in each column of a matrix

14.4.2 Demonstration

`demoNearNeigh.m`: K Nearest Neighbour Demo

14.5 Summary

- Nearest neighbour methods are general classification methods.
- The NN method can be understood as a class conditional mixture of Gaussians in the limit of a vanishingly small covariance for each mixture component model.

14.6 Exercises

Exercise 14.1. *The file* `NNdata.mat` *contains training and test data for the handwritten digits 5 and 9. Using leave one out cross-validation, find the optimal K in K-nearest neighours, and use this to compute the classification accuracy of the method on the test data.*

Exercise 14.2. *Write a routine* `SoftNearNeigh(xtrain,xtest,trainlabels,sigma)` *to implement soft nearest neighbours, analogous to* `nearNeigh.m`*. Here* `sigma` *is the variance σ^2 in equation (14.3.1). The file* `NNdata.mat` *contains training and test data for the handwritten digits 5 and 9. Using leave one out cross-validation, find the optimal σ^2 and use this to compute the classification accuracy of the method on the test data. Hint: you may have numerical difficulty with this method. To avoid this, consider using the logarithm, and how to numerically compute $\log\left(e^a + e^b\right)$ for large (negative) a and b. See also* `logsumexp.m`*.*

Exercise 14.3. *The editor at YoMan! (a 'mens' magazine) has just had a great idea. Based on the success of a recent national poll to test IQ, she decides to make a 'Beauty Quotient' (BQ) test. She collects as many images of male faces as she can, taking care to make sure that all the images are scaled to roughly* *the same size and under the same lighting conditions. She then gives each male face a BQ score from 0 ('Severely Aesthetically Challenged') to 100 ('Generously Aesthetically Gifted'). Thus, for each real-valued D-dimensional image* $\mathbf{x}$*, there is an associated value* b *in the range 0 to 100. In total she collects* N *images and associated scores,* $\{(\mathbf{x}^n, b^n), n = 1, \ldots, N\}$*. One morning, she bounces into your office and tells you the good news : it is your task to make a test for the male nation to determine their Beauty Quotient. The idea, she explains, is that a man can send online an image of their face* $\mathbf{x}^*$*, to YoMan! and will immediately receive an automatic BQ response* b^**.*

1. *As a first step, you decide to use the* K *nearest neighbour method (KNN) to assign a BQ score* b^* *to a novel test image* $\mathbf{x}^*$*.*

   Describe how to determine the optimal number of neighbours K *to use.*

2. *Your line manager is pleased with your algorithm but is disappointed that it does not provide any simple explanation of Beauty that she can present in a future version of YoMan! magazine.*

   To address this, you decide to make a model based on linear regression. That is

   $$b = \mathbf{w}^{\mathsf{T}}\mathbf{x} \tag{14.6.1}$$

   where $\mathbf{w}$ *is a parameter vector chosen to minimise*

   $$E(\mathbf{w}) = \sum_n \left(b^n - \mathbf{w}^{\mathsf{T}}\mathbf{x}^n\right)^2$$

   (a) *After training (finding a suitable* $\mathbf{w}$*), how can YoMan! explain to its readership in a simple way what facial features are important for determining one's BQ?*

   (b) *Describe fully and mathematically a method to train this linear regression model.*

   (c) *Discuss any implications of the situation* $D > N$*.*

   (d) *Discuss any advantages/disadvantages of using the linear regression model compared with using the KNN approach.*

CHAPTER 15

Unsupervised Linear Dimension Reduction

High-dimensional data is prevalent in machine learning and related areas. Indeed, there often arises the situation in which there are more data dimensions than there are data examples. In such cases we seek a lower dimensional representation of the data. In this chapter we discuss some standard methods which can also improve the prediction performance by removing 'noise' from the representation.

15.1 High-Dimensional Spaces – Low Dimensional Manifolds

In machine learning problems data is often high dimensional – images, bag-of-word descriptions, gene-expressions *etc.* In such cases we cannot expect the training data to densely populate the space, meaning that there will be large parts in which little is known about the data. For the hand-written digits from chapter(14), the data is 784 dimensional and for binary valued pixels the number of possible images is $2^{784} \approx 10^{236}$. Nevertheless, we would expect that only a handful of examples of a digit should be sufficient (for a human) to understand how to recognise a 7. Digit-like images must therefore occupy a highly constrained volume of the 784 dimensions and we expect only a small number of degrees of freedom to be required to describe the data to a reasonable accuracy. Whilst the data vectors may be very high dimensional, they will therefore typically lie close to a much lower dimensional 'manifold' (informally, a two-dimensional manifold corresponds to a warped sheet of paper embedded in a high dimensional space), meaning that the distribution of the data is heavily constrained. Here we concentrate on computationally efficient linear dimension reduction techniques in which a high dimensional datapoint $\mathbf{x}$ is projected down to a lower dimensional vector $\mathbf{y}$ by

$$\mathbf{y} = \mathbf{F}\mathbf{x} + \mathsf{const}. \tag{15.1.1}$$

The non-square matrix $\mathbf{F}$ has dimensions $\dim(\mathbf{y}) \times \dim(\mathbf{x})$, with $\dim(\mathbf{y}) < \dim(\mathbf{x})$. The methods in this chapter are largely non-probabilistic, although many have natural probabilistic interpretations. For example, PCA is closely related to Factor Analysis, as described in chapter(21).

15.2 Principal Components Analysis

If data lies close to a hyperplane, as in fig(15.1) we can accurately approximate each data point by using vectors that span the hyperplane alone. In such cases we aim to discover a low dimensional co-ordinate system in which we can approximately represent the data. We express the approximation for datapoint $\mathbf{x}^n$ as

$$\mathbf{x}^n \approx \mathbf{c} + \sum_{i=1}^{M} y_i^n \mathbf{b}^i \equiv \tilde{\mathbf{x}}^n \tag{15.2.1}$$

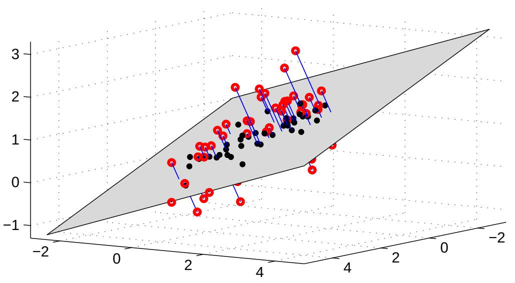


Figure 15.1: In linear dimension reduction a hyperplane is fitted such that the average distance between datapoints (red rings) and their projections onto the plane (black dots) is minimal.

Here the vector $\mathbf{c}$ is a constant and defines a point in the hyperplane and the $\mathbf{b}^i$ are 'basis' vectors that span the hyperplane (also known as 'principal component coefficients' or 'loadings'). Collectively we can write $\mathbf{B} = \left[\mathbf{b}^1, \ldots, \mathbf{b}^M\right]$. The y_i^n are the low dimensional co-ordinates of the data. Equation(15.2.1) expresses how to find the reconstruction $\tilde{\mathbf{x}}^n$ given the lower dimensional representation $\mathbf{y}^n$ (which has components $y_i^n, i = 1, \ldots, M$). For a data space of dimension $\dim(\mathbf{x}) = D$, we hope to accurately describe the data using only a small number $M \ll D$ of co-ordinates $\mathbf{y}$.

To determine the best lower dimensional representation it is convenient to use the square distance error between $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$:

$$E(\mathbf{B}, \mathbf{Y}, \mathbf{c}) = \sum_{n=1}^{N} \sum_{i=1}^{D} [x_i^n - \tilde{x}_i^n]^2 \tag{15.2.2}$$

It is straightforward to show that the optimal bias $\mathbf{c}$ is given by the mean of the data $\sum_n \mathbf{x}^n/N$, exercise(15.1). We therefore assume that the data has been centred (has zero mean $\sum_n \mathbf{x}^n = \mathbf{0}$), so that we can set $\mathbf{c}$ to zero, and concentrate on finding the optimal basis $\mathbf{B}$ below.

15.2.1 Deriving the optimal linear reconstruction

To find the best basis vectors $\mathbf{B}$ (defining $[B]_{i,j} = b_i^j$) and corresponding low dimensional coordinates $\mathbf{Y} = \left[\mathbf{y}^1, \ldots, \mathbf{y}^N\right]$, we wish to minimize the sum of squared differences between each vector $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$:

$$E(\mathbf{B}, \mathbf{Y}) = \sum_{n=1}^{N} \sum_{i=1}^{D} \left[x_i^n - \sum_{j=1}^{M} y_j^n b_i^j \right]^2 = \text{trace}\left((\mathbf{X} - \mathbf{B}\mathbf{Y})^{\mathsf{T}} (\mathbf{X} - \mathbf{B}\mathbf{Y}) \right) \tag{15.2.3}$$

where $\mathbf{X} = \left[\mathbf{x}^1, \ldots, \mathbf{x}^N\right]$.

Consider an invertible transformation $\mathbf{Q}$ of the basis $\mathbf{B}$ so that $\tilde{\mathbf{B}} \equiv \mathbf{B}\mathbf{Q}$ is an orthonormal matrix, $\tilde{\mathbf{B}}^{\mathsf{T}}\tilde{\mathbf{B}} = \mathbf{I}$. Since $\mathbf{Q}$ is invertible, we may write $\mathbf{B}\mathbf{Y} = \tilde{\mathbf{B}}\mathbf{Q}^{-1}\mathbf{Y} \equiv \tilde{\mathbf{B}}\tilde{\mathbf{Y}}$, which is of then same form as $\mathbf{B}\mathbf{Y}$, albeit with an orthonormality constraint on $\tilde{\mathbf{B}}$. Hence, without loss of generality, we may consider equation (15.2.3) under the orthonormality constraint $\mathbf{B}^{\mathsf{T}}\mathbf{B} = \mathbf{I}$, namely that the basis vectors are mutually orthogonal and of unit length.

By differentiating equation (15.2.3) with respect to y_k^n we obtain (using the orthonormality constraint)

$$-\frac{1}{2}\frac{\partial}{\partial y_k^n} E(\mathbf{B}, \mathbf{Y}) = \sum_i \left[x_i^n - \sum_j y_j^n b_i^j \right] b_i^k = \sum_i x_i^n b_i^k - \sum_j y_j^n \underbrace{\sum_i b_i^j b_i^k}_{\delta_{jk}} = \sum_i x_i^n b_i^k - y_k^n$$

The squared error $E(\mathbf{B}, \mathbf{Y})$ therefore has zero derivative when

$$y_k^n = \sum_i b_i^k x_i^n, \qquad \text{which can be written as} \quad \mathbf{Y} = \mathbf{B}^{\mathsf{T}}\mathbf{X} \tag{15.2.4}$$

We now substitute this solution into equation (15.2.3) to write the squared error as a function of **B** alone. Using

$$\sum_j y_j^n b_i^j = \sum_{j,k} b_i^j b_k^j x_k^n = \sum_{j,k} B_{i,j} B_{k,j} x_k^n = [\mathbf{B}\mathbf{B}^\mathsf{T}\mathbf{x}^n]_i \tag{15.2.5}$$

The objective $E(\mathbf{B})$ becomes

$$E(\mathbf{B}) = \sum_n \left(\left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)\mathbf{x}^n\right)^2 \tag{15.2.6}$$

Since $\left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)^2 = \mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}$, (using $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$)

$$E(\mathbf{B}) = \sum_n (\mathbf{x}^n)^\mathsf{T}\left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)\mathbf{x}^n = \text{trace}\left(\sum_n (\mathbf{x}^n)(\mathbf{x}^n)^\mathsf{T}\left(\mathbf{I} - \mathbf{B}\mathbf{B}^\mathsf{T}\right)\right) \tag{15.2.7}$$

Hence the objective becomes

$$E(\mathbf{B}) = (N-1)\left[\text{trace}\left(\mathbf{S}\right) - \text{trace}\left(\mathbf{S}\mathbf{B}\mathbf{B}^\mathsf{T}\right)\right] \tag{15.2.8}$$

where **S** is the sample covariance matrix of the data[1]. Since we assumed the data is zero mean, this is

$$\mathbf{S} = \frac{1}{N-1}\sum_{n=1}^N \mathbf{x}^n(\mathbf{x}^n)^\mathsf{T} \tag{15.2.9}$$

More generally, for non-zero mean data, we have

$$\mathbf{S} = \frac{1}{N-1}\sum_{n=1}^N (\mathbf{x}^n - \mathbf{m})(\mathbf{x}^n - \mathbf{m})^\mathsf{T}, \qquad \mathbf{m} = \frac{1}{N}\sum_{n=1}^N \mathbf{x}^n \tag{15.2.10}$$

To minimise equation (15.2.8) under the constraint $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{I}$ we use a set of Lagrange multipliers **L**, so that the objective is to minimize

$$-\text{trace}\left(\mathbf{S}\mathbf{B}\mathbf{B}^\mathsf{T}\right) + \text{trace}\left(\mathbf{L}\left(\mathbf{B}^\mathsf{T}\mathbf{B} - \mathbf{I}\right)\right) \tag{15.2.11}$$

(neglecting the constant prefactor $N-1$ and the trace $(\mathbf{S})$ term). Since the constraint is symmetric, we can assume that **L** is also symmetric. Differentiating with respect to **B** and equating to zero we obtain that at the optimum

$$\mathbf{S}\mathbf{B} = \mathbf{B}\mathbf{L} \tag{15.2.12}$$

We need to find matrices **B** and **L** that satisfy this equation. One solution is given when **L** is diagonal in which case this is a form of eigen-equation and the columns of **B** are the corresponding eigenvectors of **S**. In this case, trace $\left(\mathbf{S}\mathbf{B}\mathbf{B}^\mathsf{T}\right)$ = trace $(\mathbf{L})$, which is the sum of the eigenvalues corresponding to the eigenvectors forming **B**.

$$\frac{1}{N-1}E(\mathbf{B}) = -\text{trace}\left(\mathbf{L}\right) + \text{trace}\left(\mathbf{S}\right) = -\sum_{i=1}^M \lambda_i + \mathsf{const.} \tag{15.2.13}$$

Since we wish to minimise $E(\mathbf{B})$, we therefore define the basis using the eigenvectors with largest corresponding eigenvalues. If we order the eigenvalues $\lambda_1 \geq \lambda_2, \ldots$, the squared error is then given by, from equation (15.2.8)

$$\frac{1}{N-1}E(\mathbf{B}) = \text{trace}\left(\mathbf{S}\right) - \text{trace}\left(\mathbf{L}\right) = \sum_{i=1}^D \lambda_i - \sum_{i=1}^M \lambda_i = \sum_{i=M+1}^D \lambda_i \tag{15.2.14}$$

Whilst the solution to this eigen-problem is unique, this only serves to define the solution subspace since one may rotate and scale **B** and **Y** such that the value of the squared loss is exactly the same (since the least squares objective depends only on the product **BY**). The justification for choosing the non-rotated eigen solution is given by the additional requirement that the principal components corresponds to directions of maximal variance, as explained in section(15.2.2).

[1] Here we use the unbiased sample covariance, simply because this is standard in the literature. If we were to replace this with the sample covariance as defined in chapter(8), the only change required is to replace $N-1$ by N throughout, which has no effect on the form of the solutions found by PCA.

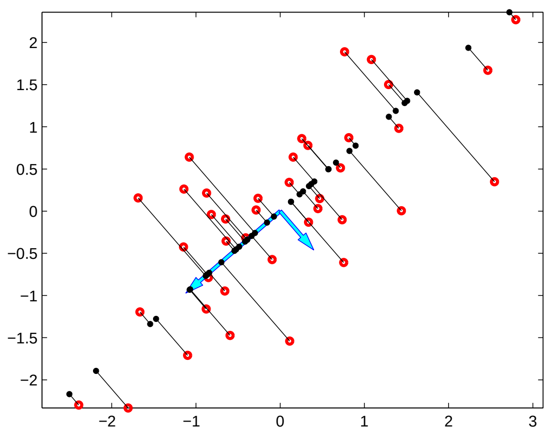


Figure 15.2: Projection of two dimensional data using one dimensional PCA. Plotted are the original datapoints $\mathbf{x}$ (larger rings) and their reconstructions $\tilde{\mathbf{x}}$ (small dots) using 1 dimensional PCA. The lines represent the orthogonal projection of the original datapoint onto the first eigenvector. The arrows are the two eigenvectors scaled by the square root of their corresponding eigenvalues. The data has been centred to have zero mean. For each 'high dimensional' datapoint $\mathbf{x}$, the 'low dimensional' representation $\mathbf{y}$ is given in this case by the distance (possibly negative) from the origin along the first eigenvector direction to the corresponding orthogonal projection point.

15.2.2 Maximum variance criterion

To break the invariance of least squares projection with respect to rotations and rescaling, we need an additional criterion. One such is given by first searching for the single direction $\mathbf{b}$ such that the variance of the data projected onto this direction is maximal amongst all possible such projections. Using equation (15.2.4) for a single vector $\mathbf{b}$ we have

$$y^n = \sum_i b_i x_i^n \tag{15.2.15}$$

The projection of a datapoint onto a direction $\mathbf{b}$ is $\mathbf{b}^\mathsf{T}\mathbf{x}^n$ for a unit length vector $\mathbf{b}$. Hence the sum of squared projections is

$$\sum_n \left(\mathbf{b}^\mathsf{T}\mathbf{x}^n\right)^2 = \mathbf{b}^\mathsf{T}\left[\sum_n \mathbf{x}^n \left(\mathbf{x}^n\right)^\mathsf{T}\right]\mathbf{b} = (N-1)\mathbf{b}^\mathsf{T}\mathbf{S}\mathbf{b} = \lambda(N-1) \tag{15.2.16}$$

Ignoring constants, this is the negative of equation (15.2.8) for a single retained eigenvector $\mathbf{b}$ (with $\mathbf{S}\mathbf{b} = \lambda\mathbf{b}$). Hence the optimal single $\mathbf{b}$ which maximises the projection variance is given by the eigenvector corresponding to the largest eigenvalue of $\mathbf{S}$. Under the criterion that the next optimal direction $\mathbf{b}^{(2)}$ should be orthonormal to the first, one can readily show that $\mathbf{b}^{(2)}$ is given by the second largest eigenvector, and so on. This explains why, despite the squared loss equation (15.2.8) being invariant with respect to arbitrary rotation (and scaling) of the basis vectors, the ones given by the eigen-decomposition have the additional property that they correspond to directions of maximal variance. These maximal variance directions found by PCA are called the *principal directions*.

15.2.3 PCA algorithm

The routine for PCA is presented in algorithm(15.1). In the notation of $\mathbf{y} = \mathbf{F}\mathbf{x}$, the projection matrix $\mathbf{F}$ corresponds to $\mathbf{E}^\mathsf{T}$. Similarly for the reconstruction equation (15.2.1), the coordinate $\mathbf{y}^n$ corresponds to $\mathbf{E}^\mathsf{T}\mathbf{x}^n$ and $\mathbf{b}^i$ corresponds to $\mathbf{e}^i$. The PCA reconstructions are orthogonal projections of the data onto the subspace spanned by the eigenvectors corresponding to the M largest eigenvalues of the covariance matrix, see fig(15.2).

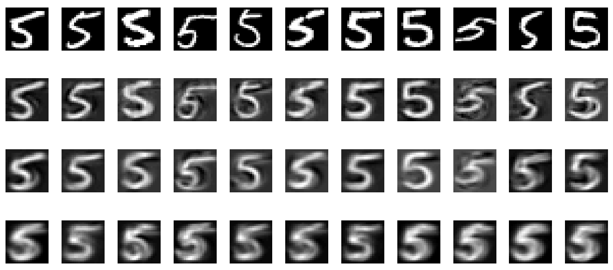


Figure 15.3: Top row : a selection of the digit 5 taken from the database of 892 examples. Plotted beneath each digit is the reconstruction using 100, 30 and 5 eigenvectors (from top to bottom). Note how the reconstructions for fewer eigenvectors express less variability from each other, and resemble more a mean 5 digit.

Algorithm 15.1 Principal Components Analysis to form an M-dimensional approximation of a dataset $\{\mathbf{x}^n, n = 1, \ldots, N\}$, with $\dim \mathbf{x}^n = D$.

```
1: Find the $D \times 1$ sample mean vector and $D \times D$ covariance matrix
       $$\mathbf{m} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^n, \quad \mathbf{S} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}^n - \mathbf{m})(\mathbf{x}^n - \mathbf{m})^{\mathsf{T}}$$
2: Find the eigenvectors $\mathbf{e}^1, \ldots, \mathbf{e}^M$ of the covariance matrix $\mathbf{S}$, sorted so that the eigenvalue of $\mathbf{e}^i$ is larger
   than $\mathbf{e}^j$ for $i < j$. Form the matrix $\mathbf{E} = [\mathbf{e}^1, \ldots, \mathbf{e}^M]$.
3: The lower dimensional representation of each data point $\mathbf{x}^n$ is given by
       $$\mathbf{y}^n = \mathbf{E}^{\mathsf{T}}(\mathbf{x}^n - \mathbf{m}) \quad (15.2.17)$$
4: The approximate reconstruction of the original datapoint $\mathbf{x}^n$ is
       $$\mathbf{x}^n \approx \mathbf{m} + \mathbf{E}\mathbf{y}^n \quad (15.2.18)$$
5: The total squared error over all the training data made by the approximation is
       $$\sum_{n=1}^{N}(\mathbf{x}^n - \tilde{\mathbf{x}}^n)^2 = (N-1)\sum_{j=M+1}^{D}\lambda_j \quad (15.2.19)$$
   where $\lambda_{M+1} \ldots \lambda_N$ are the eigenvalues discarded in the projection.
```

Example 15.1 (Reducing the dimension of digits)**.** We have 892 examples of handwritten 5's, where each image consists of 28×28 real-values pixels, see fig(15.3). Each image matrix is stacked to form a 784 dimensional vector, giving a 784×892 dimensional data matrix $\mathbf{X}$. The covariance matrix of this data has eigenvalue spectrum as plotted in fig(15.4), where we plot only the 100 largest eigenvalues. Note how after around 40 components, the mean squared reconstruction error is small, indicating that the data lies close to a 40 dimensional hyperplane. The eigenvalues are computed using `pca.m`.

The reconstructions using different numbers of eigenvectors (100, 30 and 5) are plotted in fig(15.3). Note how using only a small number of eigenvectors, the reconstruction more closely resembles the mean image.

Example 15.2 (Eigenfaces)**.** In fig(15.5) we present example images for which we wish to find a lower dimensional representation. Using PCA the first 49 'eigenfaces' are presented along with reconstructions of the original data using these eigenfaces, see fig(15.6).

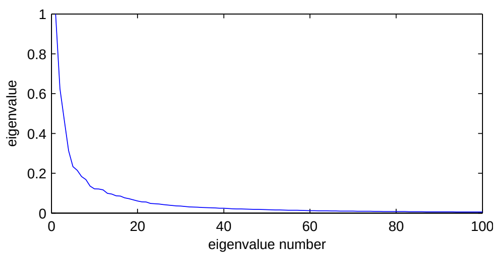


Figure 15.4: For the digits data consisting of 892 examples of the digit 5, each image being represented by a 784 dimensional vector. Plotted as the largest 100 eigenvalues (scaled so that the largest eigenvalue is 1) of the sample covariance matrix.

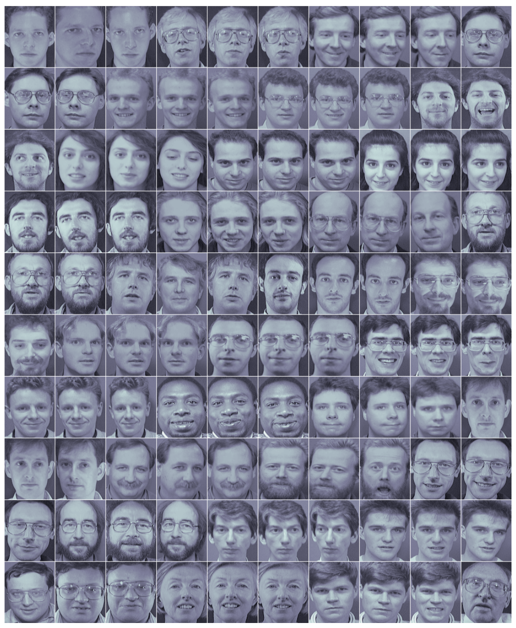
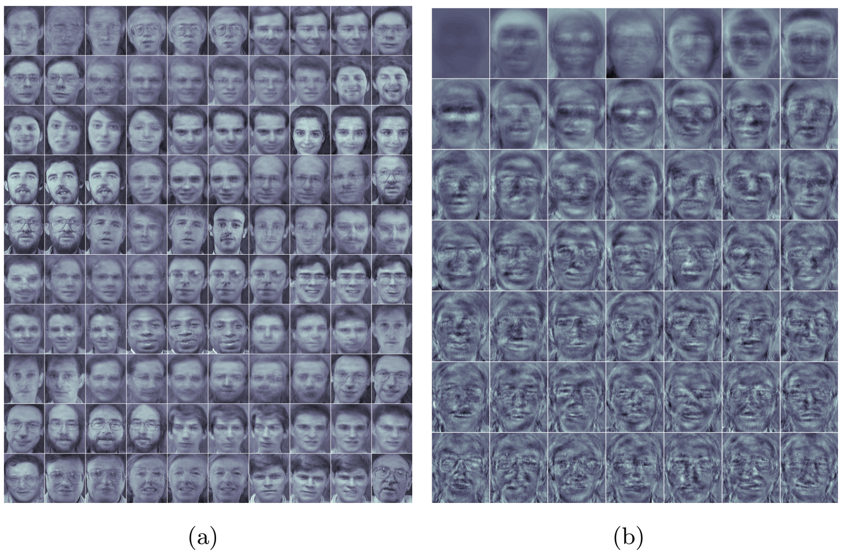


Figure 15.5: 100 of the 120 training images (40 people, with 3 images of each person). Each image consists of $92 \times 112 = 10304$ greyscale pixels. The train data is scaled so that, represented as an image, the components of each image sum to 1. The average value of each pixel across all images is 9.70×10^{-5}. This is a subset of the 400 images in the full Olivetti Research Face Database.



(a) (b)

Figure 15.6: **(a)**: SVD reconstruction of the images in fig(15.5) using a combination of the 49 eigen-images. **(b)**: The eigen-images are found using SVD of the fig(15.5) and taking the 49 eigenvectors with largest eigenvalue. The images corresponding to the largest eigenvalues are contained in the first row, and the next 7 in the row below, *etc.* The root mean square reconstruction error is 1.121×10^{-5}, a small improvement over PLSA (see fig(15.15)).

15.2.4 PCA and nearest neighbours

In nearest neighbours we need to compute the distance between datapoints. For high-dimensional data computing the squared Euclidean distance between vectors can be expensive, and also sensitive to noise. It is therefore often useful to project the data to a lower dimensional representation first. For example, in making a classifier to distinguish between the digit 1 and the digit 7, example(14.1), we can form a lower dimensional representation and use this as a more robust representation of the data. To do so we first by ignore the class label to make a dataset of 1200 training points. Each of the training points $\mathbf{x}^n$ is then projected to a lower dimensional PCA representation $\mathbf{y}^n$. Subsequently, any distance calculations $(\mathbf{x}^a - \mathbf{x}^b)^2$ are replaced by $(\mathbf{y}^a - \mathbf{y}^b)^2$. To justify this, consider

$$\begin{aligned}
(\mathbf{x}^a - \mathbf{x}^b)^\mathsf{T}(\mathbf{x}^a - \mathbf{x}^b) &\approx (\mathbf{E}\mathbf{y}^a + \mathbf{m} - \mathbf{E}\mathbf{y}^b - \mathbf{m})^\mathsf{T}(\mathbf{E}\mathbf{y}^a + \mathbf{m} - \mathbf{E}\mathbf{y}^b - \mathbf{m}) \\
&= (\mathbf{y}^a - \mathbf{y}^b)^\mathsf{T}\mathbf{E}^\mathsf{T}\mathbf{E}(\mathbf{y}^a - \mathbf{y}^b) \\
&= (\mathbf{y}^a - \mathbf{y}^b)^\mathsf{T}(\mathbf{y}^a - \mathbf{y}^b) \qquad (15.2.20)
\end{aligned}$$

where the last equality is due to the orthonormality of eigenvectors, $\mathbf{E}^\mathsf{T}\mathbf{E} = \mathbf{I}$.

Using 19 principal components (see example(15.3) as to why this number was chosen) and the nearest neighbour rule to classify 1's and 7's gave a test-set error of 14 in 600 examples, compared to 18 from the standard method on the non-projected data. A plausible explanation for this improvement is that the new PCA representation of the data is more robust since only the large scale change directions in the space are

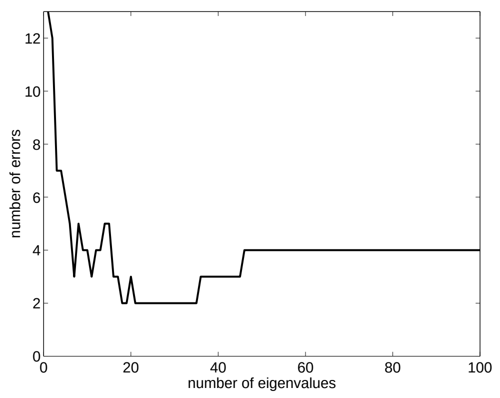


Figure 15.7: Finding the optimal PCA dimension to use for classifying hand-written digits using nearest neighbours. 400 training examples are used, and the validation error plotted on 200 further examples. Based on the validation error, we see that a dimension of 19 is reasonable.

retained, with low variance directions discarded.

Example 15.3 (Finding the best PCA dimension)**.** There are 600 examples of the digit 1 and 600 examples of the digit 7. We will use half the data for training and the other half for testing. The 600 training examples were further split into a training set of 400 examples and a separate validation set of 200 examples. PCA was used to reduce the dimensionality of the inputs, and then nearest neighbours used to classify the 200 validation examples. Different reduced dimensions were investigated and, based on the validation results, 19 was selected as the optimal number of PCA components retained, see fig(15.7). The independent test error on 600 independent examples using 19 dimensions is 14.

15.2.5 Comments on PCA

The 'intrinsic' dimension of data

How many dimensions should the linear subspace have? From equation (15.2.14), the reconstruction error is proportional to the sum of the discarded eigenvalues. If we plot the eigenvalue spectrum (the set of eigenvalues ordered by decreasing value), we might hope to see a few large values and many small values. If the data does lie close to an M dimensional hyperplane, we would see M large eigenvalues with the rest being very small. This gives an indication of the number of degrees of freedom in the data, or the intrinsic dimensionality. Directions corresponding to the small eigenvalues are then interpreted as 'noise'.

Non-linear dimension reduction

In PCA we are presupposing that the data lies close to a hyperplane. Is this really a good description? More generally, we would expect data to lie on low dimensional curved manifolds. Also, data is often clustered – examples of handwritten 4's look similar to each other and form a cluster, separate from the 8's cluster. Nevertheless, since linear dimension reduction is computationally relatively straightforward, this is one of the most common dimensionality reduction techniques.

15.3 High Dimensional Data

The computational complexity of computing the eigen-decomposition of a $D \times D$ matrix is $O\left(D^3\right)$. You might be wondering therefore how it is possible to perform PCA on high dimensional data. For example, if we have 500 images each of $1000 \times 1000 = 10^6$ pixels, the covariance matrix will be a $10^6 \times 10^6$ square matrix. It would appear be significant computational challenge to compute the eigen-decomposition of this matrix directly. In this case, however, since there are only 500 such vectors, the number of non-zero eigenvalues cannot exceed 500. One can exploit this fact to bound the complexity by $O\left(\min(D,N)^3\right)$, as described below.

15.3.1 Eigen-decomposition for $N < D$

First note that for zero mean data, the sample covariance matrix can be expressed as

$$[\mathbf{S}]_{ij} = \frac{1}{N-1}\sum_{n=1}^{N} x_i^n x_j^n \tag{15.3.1}$$

In matrix notation this can be written

$$\mathbf{S} = \frac{1}{N-1}\mathbf{X}\mathbf{X}^{\mathsf{T}} \tag{15.3.2}$$

where the $D \times N$ matrix $\mathbf{X}$ contains all the data vectors:

$$\mathbf{X} = \left[\mathbf{x}^1, \ldots, \mathbf{x}^N\right] \tag{15.3.3}$$

Since the eigenvectors of a matrix $\mathbf{M}$ are equal to those of $\gamma\mathbf{M}$ for scalar γ, one can consider more simply the eigenvectors of $\mathbf{X}\mathbf{X}^{\mathsf{T}}$. Writing the $D \times N$ matrix of eigenvectors as $\mathbf{E}$ (this is a non-square thin matrix since there will be fewer eigenvalues than data dimensions) and the eigenvalues as an $N \times N$ diagonal matrix $\mathbf{\Lambda}$, the eigen-decomposition of the scaled covariance $\mathbf{S}$ is

$$\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{E} = \mathbf{E}\mathbf{\Lambda} \Rightarrow \mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{X}^{\mathsf{T}}\mathbf{E} = \mathbf{X}^{\mathsf{T}}\mathbf{E}\mathbf{\Lambda} \Rightarrow \mathbf{X}^{\mathsf{T}}\mathbf{X}\tilde{\mathbf{E}} = \tilde{\mathbf{E}}\mathbf{\Lambda} \tag{15.3.4}$$

where we defined $\tilde{\mathbf{E}} = \mathbf{X}^{\mathsf{T}}\mathbf{E}$. The final expression above represents the eigenvector equation for $\mathbf{X}^{\mathsf{T}}\mathbf{X}$. This is a matrix of dimensions $N \times N$ so that calculating the eigen-decomposition takes $O\left(N^3\right)$ operations, compared with $O\left(D^3\right)$ operations in the original high-dimensional space. We then can therefore calculate the eigenvectors $\tilde{\mathbf{E}}$ and eigenvalues $\mathbf{\Lambda}$ of this matrix more easily. Once found, we use the fact that the eigenvalues of $\mathbf{S}$ are given by the diagonal entries of $\mathbf{\Lambda}$ and the eigenvectors by

$$\mathbf{E} = \mathbf{X}\tilde{\mathbf{E}}\mathbf{\Lambda}^{-1} \tag{15.3.5}$$

15.3.2 PCA via Singular value decomposition

An alternative to using an eigen-decomposition routine to find the PCA solution is to make use of the *Singular Value Decomposition* (SVD) of an $D \times N$ dimensional matrix $\mathbf{X}$. This is given by

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}} \tag{15.3.6}$$

where $\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}_D$ and $\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{I}_N$ and $\mathbf{D}$ is a diagonal matrix of the (positive) singular values. We assume that the decomposition has ordered the singular values so that the upper left diagonal element of $\mathbf{D}$ contains the largest singular value. The matrix $\mathbf{X}\mathbf{X}^{\mathsf{T}}$ can then be written as

$$\mathbf{X}\mathbf{X}^{\mathsf{T}} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}\mathbf{V}\mathbf{D}\mathbf{U}^{\mathsf{T}} = \mathbf{U}\mathbf{D}^2\mathbf{U}^{\mathsf{T}} \tag{15.3.7}$$

Since $\mathbf{U}\mathbf{D}^2\mathbf{U}^{\mathsf{T}}$ is in the form of an eigen-decomposition, the PCA solution is equivalently given by performing the SVD decomposition of $\mathbf{X}$, for which the eigenvectors are then given by $\mathbf{U}$, and corresponding eigenvalues by the square of the singular values.

Equation(15.3.6) shows that PCA is a form of matrix decomposition method:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}} \approx \mathbf{U}_M\mathbf{D}_M\mathbf{V}_M^{\mathsf{T}} \tag{15.3.8}$$

where $\mathbf{U}_M$, $\mathbf{D}_M$, $\mathbf{V}_M$ correspond to taking only the first M singular values of the full matrices.

15.4 Latent Semantic Analysis

In the document analysis literature PCA is also called Latent Semantic Analysis and is concerned with analysing a set of N documents. Each document is represented by a vector

$$\mathbf{x}^n = \left(x_1^n, \ldots, x_D^n\right)^{\mathsf{T}} \tag{15.4.1}$$