ECE 403/503

Optimization for Machine Learning

Wu-Sheng Lu

Department of Electrical and Computer Engineering

University of Victoria

Online link: http://www.ece.uvic.ca/~wslu/403/index.html

Office: EOW 427

E-mail: wslu@ece.uvic.ca

May 2019

Instructor	**Office Hours**
Wu-Sheng Lu	Days: Wednesdays
Phone: 721-8692	Time: 14:00 – 16:00
E-mail: wslu@ece.uvic.ca	Location: EOW 427
URL: http://www.ece.uvic.ca/~wslu	

Lectures

Days: Tuesday, Wednesday, Friday

Time: 12:30 – 1:20pm

Location: David Turpin Building A120

Labs for ECE 403 & 503

B01 T, 2:00-4:50 pm, ELW326, May 28, June 11, June 25, July 16

B04 F, 2:00-4:50 pm, ELW326, June 7, June 21, July 12, July 26

Required Text

Course Notes

Assessment

Assignments:	10%	
Labs (for ECE 403)	15%	
Labs & Project (for ECE 503)	15%	
Mid-term exam	20%	Date: June 26 (Wednesday)
Final exam	55%	

Other Things to Note

- Course web site: http://www.ece.uvic.ca/~wslu/403/index.html
  You need the user name and password to access "Course Material".
- About assignments:
  - There will be 7 assignments.
  - Only paper copies are accepted.
  - Please clearly print your name and V-number on the first page of your work.
  - Submit your assignment on the due day by 4:00 pm.

    2 submission options:
    - ◊ Submit your work to the instructor at class; or
    - ◊ Slide it into the instructor's office EOW 427.

  NOTE: ECE403-503 do **not** use drop box for **assignments**.
- **The classes on May 28 (Tuesday) and May 29 (Wednesday) will be cancelled.**

A make-up class of 100 minutes is scheduled on May 15, Wednesday, 3:00 - 4:40pm.

References

[1] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223-311, 2018.

[2] Y. LeCun, C. Cortes, and C. J. C. Burges, "The MNIST database," see the web link: http://yann.lecun.com/exdb/mnist/

[3] UCI Machine Learning, http://archive.ics.uci.edu/ml, University of California Irvine, School of Information and Computer Science.

[4] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, 1989.

[6] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, pp. 788–791, 21Oct. 1999.

[7] N. Gillis, "The why and how of nonnegative matrix factorization," in *Regularization, Optimization, Kernels, and Support Vector Machines*, J. A. K. Suykens, M. Signoretto, and A. Argyriou Eds., pp. 257-291, CRC Press, 2014.

[8] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer, 2009.

[9] A. Antoniou and W.-S. Lu, *Practical Optimization – Algorithms and Engineering Applications*, Springer 2007.

[10] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[11] Y. Nesterov, "A method for solving a convex programming problem with rate of convergence $O(1/k^2)$," *Soviet Math. Doklady*, vol. 269, no. 3, pp. 543–547, 1983, in Russian.

[12] Y. Nesterov, *Introductory Lectures on Convex Optimization — A Basic Course*, Norwell, MA.: Kluwer Academic Publishers, 2004.

[13] M. Grant and S. Boyd, *CVX: MATLAB Software for Disciplined Convex Programming*, version 2.0, Sept. 2013, online available: http://cvxr.com/cvx

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[15] W. N. Street, W. H. Wolberg, and O. L. Mangasarian, "Nuclear feature extraction for breast tumor diagnosis," in *IS?T/SPIE Int. Symp. Electronic Imaging: Science and Technology*, vol. 1905, pp. 861-870, San Jose, CA., 1993.

[16] O. L. Mangasarian, W. N. Street, and W. H. Wolberg, "Breast cancer diagnosis and prognosis via linear programming," AAAI Tech. Report SS-94-01, 1994.

• Requirements

(a) The mathematical literacy for taking this cause includes

◊ basic linear algebra (MATH 101)

◊ calculus (MATH 100, 101, and 200)

◊ basic probability theory (STAT 254)

(b) Software

◊ Familiarity with MATLAB is essential for better understanding course material, doing homework, performing laboratory experiments, conducting projects, and so on. Occasionally we use CVX. CVX is a MATLAB software for disciplined convex optimization. Assuming you have properly installed MATLAB on your PC, you can download and install and use CVX on your computer as a MATLAB toolbox.

◊ To download CVX, go to the web link http://cvxr.com/cvx/download/ , read the instructions carefully, choose an appropriate download option and follow the installation steps exactly. An informative user guide for version 2.1 of the software is available from the web link below: http://web.cvxr.com/cvx/doc/

◊ Both MATLAB and CVX are available on the PCs in the laboratory room. It is a great idea to install these software packages on your personal computer as well for the convenience of using the software elsewhere.

Chapter 1 Optimization Problems in Machine Learning

1.1 Optimization Problems

The world is so much quantified these days. And we are told that we now live in a "big data" era as search engines, recommendation platforms, and speech and image recognition software become an indispensable part of modern society. Based on statistical techniques and by capitalizing on increasingly powerful computing resources and availability of datasets of immense size, machine learning (ML) has demonstrated a dramatic rise by playing a key role in many recent technological advances that yield undeniable performance superiority as well as societal, economic, and scientific impacts [1].

From a technical perspective, ML focuses on the development of computer programs that can access data and use it to learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future (often in terms of unseen data), based on the examples provided. In brief words, ML allows computers to learn automatically without human intervention or assistance and adjust actions accordingly.

One of the pillars of ML is *mathematical optimization* which, in this context, involves numerical computations of parameters for a system designed to make decisions based on available data. During the optimization process, these parameters are chosen to be optimal with respect to a given learning problem [1]. ML applications deal with a large variety of data sizes, types and structures, as well as a suite of criteria for performance evaluations. As a result, the problem sizes as well as the formulations and solution techniques can vary wildly from standard least-squares to highly nonlinear and nonconvex optimization methods. Nevertheless, practically all optimization problems for ML applications fit into the general formulation

$$\text{minimize} \quad f(\boldsymbol{x}) \tag{1.1a}$$

$$\text{subject to:} \quad a_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, 2, \ldots, p \tag{1.1b}$$

$$c_j(\boldsymbol{x}) \le 0 \quad \text{for } j = 1, 2, \ldots, q \tag{1.1c}$$

where $f(\boldsymbol{x})$ is an *objective* (or loss) *function*, $a_i(\boldsymbol{x}) = 0$ for $i = 1, 2, \ldots, p$ are *equality constraints*, $c_j(\boldsymbol{x}) \le 0$ for $j = 1, 2, \ldots, q$ are *inequality constraints*. According to (1.1), optimization is an algorithmic procedure for identifying a best set of parameters as vector $\boldsymbol{x}$ that reduces the loss (measure by function $f(\boldsymbol{x})$) to minimum while meeting all the requirements that are set for $\boldsymbol{x}$ to satisfy. This course is about basic concepts and optimization methods that are found particularly useful for ML applications. Our primary goal is to address the following two questions:

1. How do optimization problems arise in ML applications?
2. What are the most successful optimization methods for ML problems of various scales?

The first question is addressed in the rest of this chapter, while the second question is studied in Chapters 2 and 3.

1.2 Types of Data Sets

One thing in common in all optimization models for ML applications is the involvement of real-world data. The data types and sizes are problem-dependent and vary widely from problem to problem. It is widely realized that new problems of increasingly larger scales emerge virtually from all fields of human activities at a much faster rate than the growth of available physical and mental resources. In this course, we will be selective to focus on the most common types of data that are encountered in ML applications.

Data with Labels

A data set with labels assumes the form $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, 2, ..., N\}$, where each $\boldsymbol{x}_n$ is an example from input space X and $\boldsymbol{y}_n \in \mathcal{Y}$ is a label associated with $\boldsymbol{x}_n$. It is the labels that makes the data extremely valuable in the context of learning: datasets with labels provide examples we can trust.

Example 1.1 This example is about the MNIST (Modified National Institute of Standards and Technology) database for handwritten digits recognition [2]. The database provides 60,000 handwritten digits for training, which constitute 10 roughly equal-size data sets for 10 numerals from 0 to 9; each digit is represented by a 8-bit gray-scale image of size 28 by 28. The database also provides a separate 10,000 handwritten digits for test purposes. Fig. 1.1 displays several sample digits from the database.

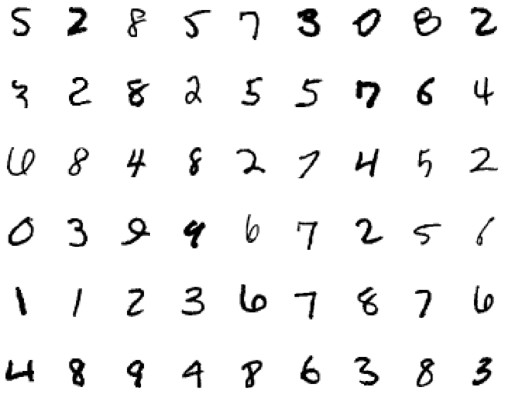


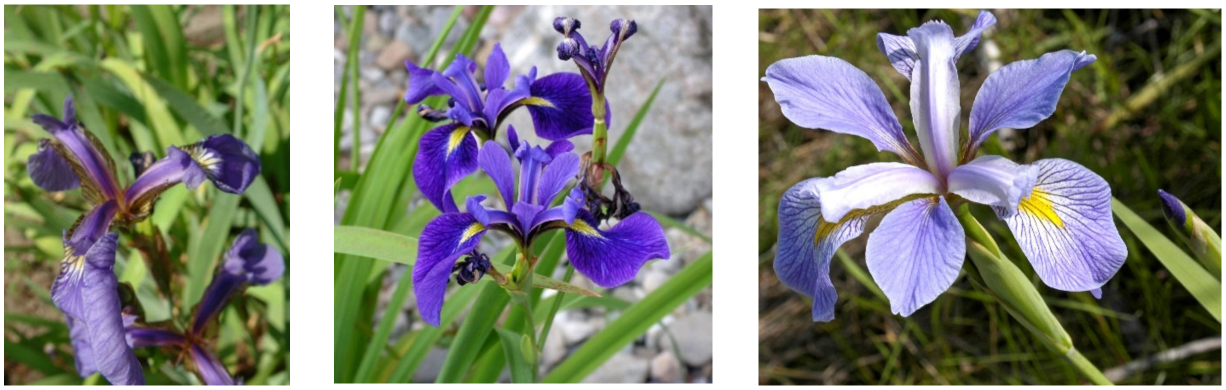
Figure 1.1. Samples of handwritten digits from MNIST database.

The training data from MNIST fit into the structure $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ with N = 60,000, where label y_n indicates one of the ten numerals between 0 and 9. A natural label assignment is to choose a $y_n \in \{0, 1, ..., 9\}$ that coincides the numeral sample $\boldsymbol{x}_n$ represents. Each $\boldsymbol{x}_n$ is a matrix of size 28 by 28 with components in the range [0, 1] with 0 and 1 denoting most white and most black pixels, respectively. Alternatively, by stacking such a matrix column by column or row by row, each x_n becomes a vector of dimension 784. Consequently, a digit sample from MNIST can be regarded as a “point” in the 784-dimensinal Euclidean space. Often times, vectorizing an image (matrix) is a good idea as it allows us to represent a set of arbitrarily many (say, m) images by a *single* matrix of the form

$$\boldsymbol{X} = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_m]$$

Obviously, the entire train data $\boldsymbol{X}$ from MNIST database has a size of 784×60000. ■

Example 1.2 In 1936 R. A. Fisher created a data set of iris plants, which is now available from UCI Machine Learning Repository [3]. It includes three classes, each consists of 50 instances (examples), for three species known as Iris Setosa, Iris Versicolor, and Iris Virginica, see Fig. 1.2 for their appearances.



Iris Setosa — Iris Versicolor — Iris Virginica

Figure 1.2. Three species of Iris.

Each training sample, namely vector $\boldsymbol{x}_n$, consists of four features representing the sepal length, sepal width, petal length, and petal width of the flower (see Fig. 1.3 for their definitions), respectively, all in centimetres. And each training sample $\boldsymbol{x}_n$ is associated with a label y_n, which is assigned to value 1 or 2 or 3, depending on whether it is Setosa or Versicolor or Virginica.

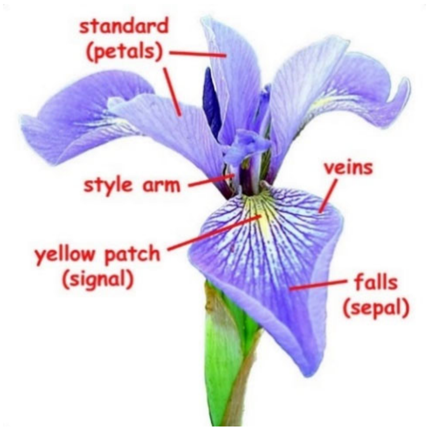


Figure 1.3. Features of Iris.

Thus the data set assumes the form $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ with $\boldsymbol{x}_n \in R^4$ and $N = 150$ that can be arranged so that the first 50 pairs $\{(\boldsymbol{x}_n, y_n)\}$ are for Iris Setosa with labels $y_n = 1$, the next 50 pairs $\{(\boldsymbol{x}_n, y_n)\}$ are for Iris Versicolor with labels $y_n = 2$, and the last 50 pairs $\{(\boldsymbol{x}_n, y_n)\}$ are for Iris Virginica with labels $y_n = 3$. Below is a summary of the basic data statistics

	Min	Max	Mean	SD	Class Correlation
Sepal length:	4.3	7.9	5.84	0.83	0.7826
Sepal width:	2.0	4.4	3.05	0.43	–0.4194
Petal length:	1.0	6.9	3.76	1.76	0.9490
Petal width:	0.1	2.5	1.20	0.76	0.9565

■

Example 1.3 The database CBCL from the Center for Biological and Computational Learning at MIT contains 2429 8-bit gray-scale facial images of 19×19 pixels, see Fig. 1.4 for the first 49 images from the database.

If each image from the dataset is vectorized as a column of dimension 361, then the entire dataset can be put together as a matrix $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_{2429} \end{bmatrix} \in R^{361 \times 2429}$, where $\boldsymbol{x}_i$ denotes the ith vectorized image. ■

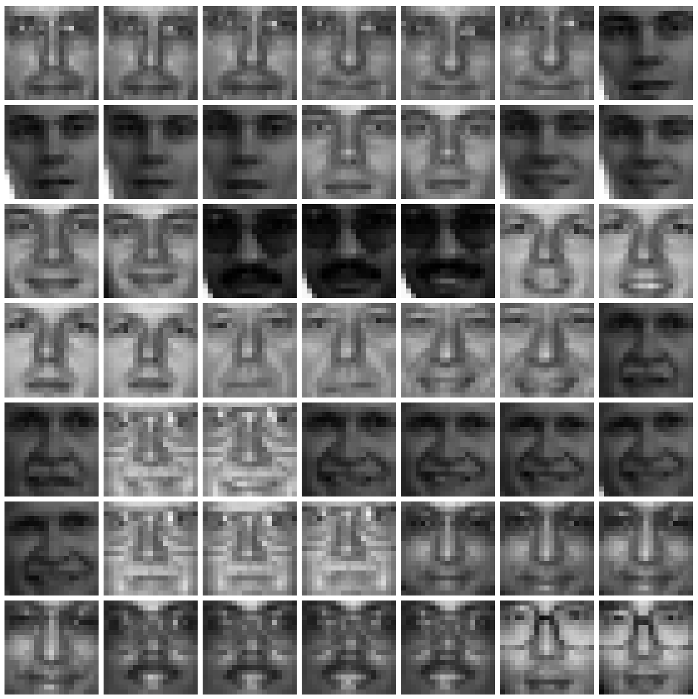


Figure 1.4. The first 49 facial images from CBCL.

Example 1.4 There is a data set of eight building parameters versus energy efficiency of buildings in terms of heating load and cooling load. It was created by A. Xifara and processed by A. Tsanas in UK in 2012 based on an energy analysis using 12 different building shapes with respect to 8 features including relative compactness, surface area, wall area, roof area, overall height, orientation, glazing area, and glazing area distribution [3]. The data set comprises 768 samples, and for each sample the data set also provides the associated heating load and cooling load as two output labels. Clearly, the data set in this case assumes the form $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, 2, ..., N\}$ with $N = 768$, $\boldsymbol{x}_n \in R^8$ and $\boldsymbol{y}_n \in R^2$. The data set can be useful in constructing a model (function) to predict a building's energy efficiency based on its feature values. ■

Example 1.5 A text document may be represented by a feature vector $\boldsymbol{x} \in R^d$ whose components are associated with a prescribed set of vocabulary words in terms of the number of times (or the frequency) the words appear [1]. The standard dataset RCV1 for document classification, for example, uses a vocabulary of $d = 47{,}152$ words to represent news stories that typically contain less than $1{,}000$ words. As a result, vector $\boldsymbol{x}$ is very sparse. To compensate for differences in document lengths, $\boldsymbol{x}$ is usually normalized to have $\|\boldsymbol{x}\|_2 = 1$. Here we see a high-dimensional and highly sparse data set. ■

Data without Labels

In unsupervised learning, we are given a set of samples $\mathcal{D} = \{\boldsymbol{x}_n, n = 1, 2, \ldots, N\}$, but their labels $\boldsymbol{y}_n$ are not available, and $\mathcal{D}$ in such a case is called an unlabeled data set. Typical learning problems in this case are related to pattern classifications and structure identification of unlabeled datasets. For example, one might be interested in dividing the data as *clusters*, and finding the sizes and centres of the clusters, etc. A method known as *K*-means clustering algorithm for unlabeled datasets will be discussed in Sec. 1.4.

1.3 Feature Extraction

A problem to work with data sets in a high-dimensional space is that a seemingly large data set (e.g. $60{,}000$ training digits from MNIST) would become so sparse that when it comes to classify a new digit, the chances to find training data in a vicinity of the new digit are extremely rare, making the inference on the new digit difficult. The problem, known as the ***curse of dimensionality*** (R. Bellman, 1961), turns out to be one of the major technical barriers facing many problems in ML applications.

A pre-processing step to help alleviate the problem is to extract from input data more effective features that reside in a space of much lower dimension, and utilize these features rather than the (raw) input data to train the machine [2]. In this section, we introduce two general techniques, namely principal component analysis (PCA) and nonnegative matrix factorization (NMF), that are widely considered among the most effective.

There is a general view point under which the two techniques to be introduced below can be better understood as particular instances of *low-rank approximation* of a given matrix. Let $\boldsymbol{A}$ be a given real-valued matrix of size $d \times m$. A low-rank approximation of $\boldsymbol{A}$ refers to an approximate factorization

$$A \approx U \cdot H \text{ with } U \in R^{d\times q} \text{ and } H \in R^{q\times m} \tag{1.2}$$

where q is the rank of both U and H, and $q \ll \min\{d, m\}$. Note that factorization (1.2) is not unique, because if $A \approx U \cdot H$ holds true, then so does $A \approx \tilde{U} \cdot \tilde{H}$ for $\tilde{U} = UQ$ and $\tilde{H} = Q^{-1}H$ with *any* nonsingular $Q \in R^{q\times q}$. In ML applications, we are interested in finding a factorization (1.2) for a data matrix A, where factors U and H are *interpretable*. It turns out such meaningful factorizations do exist, especially when certain constraints are imposed on factor U or on both factors U and H. In fact, PCA is essentially a low-rank approximation problem where the columns of matrix U are constrained to be orthonormal, while NMF fits into (1.2) where data matrix A is nonnegative and both U and H are constrained to be nonnegative.

1.3.1 *Principal Component Analysis*

We begin with a review of a matrix decomposition method from linear algebra known as *singular value decomposition* [5], [Appendix A.9].

Singular value decomposition of a matrix

Let A be a real-valued data matrix of size $d \times m$, the singular value decomposition (SVD) of A refers to the decomposition

$$A = U\Sigma V^T \tag{1.3}$$

where $U \in R^{d\times d}$ and $V \in R^{m\times m}$ are *orthogonal* matrices, namely $UU^T = U^TU = I_d$ and $VV^T = V^TV = I_m$; and $\Sigma \in R^{d\times n}$ is a diagonal matrix of the form

$$\Sigma = \begin{bmatrix} S_r & 0 \\ 0 & 0 \end{bmatrix}_{d\times m} \tag{1.4}$$

with $S_r = \text{diag}\{\sigma_1, \sigma_2, \cdots, \sigma_r\}$ and $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > 0$ where r denotes the rank of matrix A. The σ_i's are called the *singular values* of A, and the columns of U and V are called the left- and right-singular vectors of A, respectively.

Computing SVD

From (1.3) and (1.4), it is straightforward to verify that

$$AA^T = U\begin{bmatrix} S_r^2 & 0 \\ 0 & 0 \end{bmatrix}U^T, \quad A^TA = V\begin{bmatrix} S_r^2 & 0 \\ 0 & 0 \end{bmatrix}V^T \tag{1.5}$$

Consequently, the left- and right-singular vectors of A (i.e., the columns of U and V) are the eigenvectors of AA^T and A^TA, respectively, and the singular values $\{\sigma_1, \sigma_2, \cdots, \sigma_r\}$ are the square roots of the nonzero eigenvalues of AA^T (as well as A^TA). The MATLAB function **`svd`** provides a convenient way to perform SVD: **`[U,S,V] = svd(A)`** .

***Compact Form of SVD and Low-Rank Approximations of** A*

When the rank of matrix A is strictly smaller than both the number of its rows and number of its columns, namely $r < \min\{d, m\}$, only the first r left- and righ-singular vectors are relevant, and the SVD of A has a more compact form as

$$A = U_r S_r V_r^T \tag{1.6}$$

where $U_r = \begin{bmatrix} u_1 & \cdots & u_r \end{bmatrix}$ and $V_r = \begin{bmatrix} v_1 & \cdots & v_r \end{bmatrix}$ are composed of the first r columns of U and V, which are associated with the r nonzero singular values, respectively. The amount of computations required to calculate U_r and V_r depends on the sizes of AA^T and A^TA, namely d and m, respectively. If $d < m$, then computing U_r is less expensive. Once U_r is obtained, the columns of V_r can be easily calculated using

$$v_i = \sigma_i^{-1} A^T u_i \quad \text{for } i = 1, 2, \ldots, r \tag{1.7a}$$

If $m < d$, then we compute V_r, and the columns of U_r are obtained using

$$u_i = \sigma_i^{-1} A v_i \quad \text{for } i = 1, 2, \ldots, r \tag{1.7b}$$

see Problem 1.3.

From (1.6), a rank-q (for *any* positive integer q less than r) approximation of A is obtained by retaining the q *largest singular values* $\{\sigma_1, \sigma_2, \cdots, \sigma_q\}$ and the first q left- and right-singular vectors, while discarding the rest. This yields the rank-q approximation of A as

$$A \approx U_q S_q V_q^T \tag{1.8a}$$

where

$$U_q = \begin{bmatrix} u_1 & \cdots & u_q \end{bmatrix} \in R^{d\times q},\ S_q = \text{diag}\{\sigma_1, \sigma_2, \cdots, \sigma_q\},\ \text{and}\ V_q = \begin{bmatrix} v_1 & \cdots & v_q \end{bmatrix} \tag{1.8b}$$

If we let $H_q = S_q V_q^T$ which is of size $q \times m$, then (1.8a) becomes

$$A \approx U_q H_q \tag{1.9}$$

On comparing (1.9) with (1.2), we see that the SVD of a data matrix naturally leads to a suite of low-rank approximations, where matrix U_q consists of q orthogonal basis vectors of unit length. The real value of (1.9) lies in its use in *low-dimensional feature extraction*. To see this, we write data matrix A and matrix H_q explicitly in terms of their columns as $A = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}$ and $H_q = \begin{bmatrix} h_1 & h_2 & \cdots & h_m \end{bmatrix}$ where $h_i \in R^{q\times 1}$, thus (1.9) becomes an interpretable factorization od a data matrix:

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix} \approx U_q H_q = \underbrace{\begin{bmatrix} u_1 & u_2 & \cdots & u_q \end{bmatrix}}_{\substack{\text{These are } q \text{ orthonomal} \\ \text{basis vectors.}}} \cdot \underbrace{\begin{bmatrix} h_1 & h_2 & \cdots & h_m \end{bmatrix}}_{\substack{\text{Each } h_i \text{ is a } q\text{-dim feature} \\ \text{vector for the } i\text{th data point } a_i. \\ \text{In fact, using } h_i \text{ to linearly} \\ \text{combine basis vectors } \{u_k\} \\ \text{approximately reproduces } a_i.}} \tag{1.10}$$

Clearly, each vector h_i is a q-dimensional vector that serves as a *feature* for the ith data point a_i because using h_i to linearly combine basis vectors $\{u_j,\ j = 1, 2, \ldots, q\}$ approximately reproduces a_i.

By multiplying (1.10) with U_q^T and using the fact that the columns of U_q are orthonormal, namely $U_q^T U_q = I_q$, we obtain

$$U_q^T \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix} \approx U_q^T U_q H_q = H_q = \begin{bmatrix} h_1 & h_2 & \cdots & h_m \end{bmatrix}$$

This gives

$$\begin{bmatrix} \boldsymbol{U}_q^T \boldsymbol{a}_1 & \boldsymbol{U}_q^T \boldsymbol{a}_2 & \cdots & \boldsymbol{U}_q^T \boldsymbol{a}_m \end{bmatrix} \approx \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_m \end{bmatrix}$$

which provides a convenient way to compute the q-dimensional features using

$$\boldsymbol{h}_i \approx \boldsymbol{U}_q^T \boldsymbol{a}_i = \begin{bmatrix} \boldsymbol{u}_1^T \boldsymbol{a}_i \\ \boldsymbol{u}_2^T \boldsymbol{a}_i \\ \vdots \\ \boldsymbol{u}_q^T \boldsymbol{a}_i \end{bmatrix} \qquad \text{for } i = 1, 2, \ldots, m \tag{1.11}$$

In words, the feature vector for the ith data point $\boldsymbol{a}_i$ is approximately equal to $\boldsymbol{U}_q^T \boldsymbol{a}_i$ which, in the language of linear algebra, is the orthogonal projection of data point $\boldsymbol{a}_i$ onto the basis vectors $\{\boldsymbol{u}_j, j = 1, 2, \ldots, q\}$. In the literature,

$$\boldsymbol{f}_i = \boldsymbol{U}_q^T \boldsymbol{a}_i \quad \text{for } i = 1, 2, \ldots, m \tag{1.12}$$

are called *principal components* of data points $\{\boldsymbol{a}_i, i = 1, 2, \ldots, m\}$, while orthonormal singular vectors $\{\boldsymbol{u}_j, j = 1, 2, \ldots, q\}$ are called *principal axes* of dataset $\boldsymbol{A}$. We follow the above analysis to reiterate that $\{\boldsymbol{f}_i, i = 1, 2, \ldots, m\}$ are approximately the same as $\{\boldsymbol{h}_i, i = 1, 2, \ldots, m\}$, therefore can be used as low-dimensional features of a high-dimensional dataset.

Applying PCA to a Centralized Data Set

In order to apply PCA to extract low-dimensional features from a data set $X = \{\boldsymbol{x}_i, i = 1, 2, \ldots, m\}$ in an ML application, the data need to be centralized around the origin first. This is because in geometric terms PCA essentially rotates (and scales) the data in order to obtain uncorrelated axes, and this can be done effectively only when any shift (i.e., linear translation) of the data from the origin has been removed. To this end, we calculate mean of the data set as

$$\boldsymbol{\mu} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i \tag{1.13}$$

to generate a centralized data set as $\hat{X} = \{\boldsymbol{x}_i - \boldsymbol{\mu}, i = 1, 2, \ldots, m\}$. If X contains multiple data classes, the above pre-processing step has to be applied to each and every individual data class separately. Next we define data matrix

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{\mu} & \boldsymbol{x}_2 - \boldsymbol{\mu} & \cdots & \boldsymbol{x}_m - \boldsymbol{\mu} \end{bmatrix} \tag{1.14}$$

PCA-based feature extraction is now summarized as follows.

PCA for feature extraction

Input: $X = \{\boldsymbol{x}_i, i = 1, 2, \ldots, m\}$ and a target dimension q.

Step 1 Use (1.13) and (1.14) to compute $\boldsymbol{\mu}$ and centralize the input data.

Step 2 Compute q eigenvectors $\{\boldsymbol{u}_j, j = 1, 2, \ldots, q\}$ of covariance matrix $\boldsymbol{C} = \frac{1}{m} \boldsymbol{A}\boldsymbol{A}^T$ corresponding to its q largest eigenvalues, and construct matrix $\boldsymbol{U}_q$ using (1.8b).

Step 3 Calculate q-dim features $\{\boldsymbol{f}_i, i = 1, 2, \ldots, m\}$ using $\boldsymbol{f}_i = \boldsymbol{U}_q^T (\boldsymbol{x}_i - \boldsymbol{\mu})$ for $i = 1, 2, \ldots, m$.

Example 1.6 Apply PCA to Fisher's data set for Iris plants (see Example 1.2) to extract 2-dimensional features of the data set.

Solution

Fisher's dataset contains 3 classes, each has 50 samples. We denote the data classes as $\mathcal{D}_k = \{\boldsymbol{x}_i^{(k)}, i = 1, 2, \ldots, 50\}$ for k = 1, 2, 3 and apply PCA with q = 2 to each class to obtain its mean and principal axes, namely $\{\boldsymbol{\mu}^{(k)}, \boldsymbol{U}_2^{(k)}\}$ for k = 1, 2, 3. The two-dimensional feature vectors were computed using $\boldsymbol{f}_i^{(k)} = \boldsymbol{U}_2^{(k)T}\boldsymbol{x}_i^{(k)}$ for k = 1, 2, 3 and i = 1, 2, …, 50. Note that here the principal components were calculated not using $\boldsymbol{f}_i^{(k)} = \boldsymbol{U}_2^{(k)T}(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}^{(k)})$. This is because the effect of removing the mean of each data class to perform PCA needs to be compensated. The compensation is done by adding the projected mean to the principal components, that is

$$\underbrace{\boldsymbol{U}_2^{(k)T}(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}^{(k)})}_{\text{principal component}} + \underbrace{\boldsymbol{U}_2^{(k)T}\boldsymbol{\mu}^{(k)}}_{\substack{\text{projected mean}\\ \text{of the data class}}} = \boldsymbol{U}_2^{(k)T}\boldsymbol{x}_i^{(k)}$$

The 2-dimensional features of the 150 samples are shown in Fig. 1.5 in blue, red, and black for Setosa, Versicolour, and Viginica, respectively. It is observed that the three data classes were well separated when represented by their low-dimensional features. ■

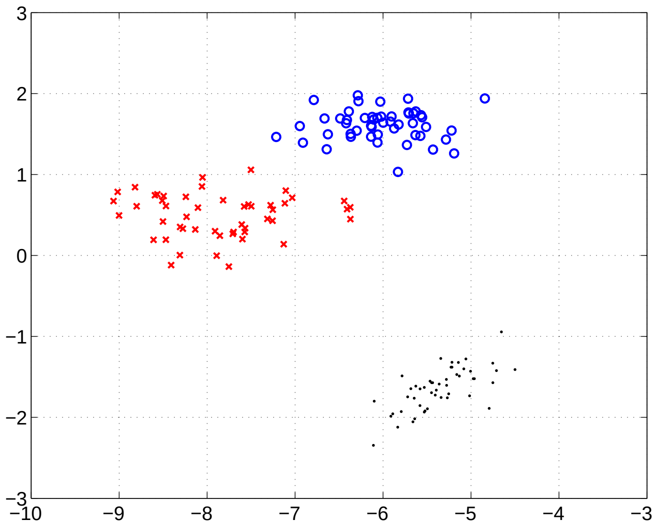


Figure 1.5. PCA-based two-dimensional features of Fisher's Iris plants. Points in blue, red, and black represent Setosa, Versicolour, and Viginica Iris, respectively.

1.3.2 *Nonnegative Matrix Factorization*

A matrix is said to be nonnegative if its components are all nonnegative. In practice many datasets are nonnegative matrices, these includes digital images, probabilities, five-star product or movie ratings, etc. Nonnegative matrix factorization (NMF) is a relatively recent technique for feature extraction in addition to its many other uses. The technique applies to a *nonnegative* dataset $\boldsymbol{X} \in R^{d\times m}$ and finds two low-rank *nonnegative* matrices $\boldsymbol{U} \in R^{d\times q}$ and $\boldsymbol{H} \in R^{q\times m}$ with

$q \ll \min\{d, m\}$ such that $\boldsymbol{X} \approx \boldsymbol{U} \cdot \boldsymbol{H}$ in a certain sense. In a landmark work by D. D. Lee and H. S. Seung [6], NMF is shown to be able to learn parts of objects from input data, the parts might be eyes, noses, and lips etc. if the input is a set of facial images of humans, or they might be semantic features such as the most frequent words in a text if the input is a set of text documents.

Unlike PCA where the columns of matrix $\boldsymbol{U}$ are constrained to be orthonormal so they can be regarded as basis vectors to represent the data effectively, NMF imposes the constraints that both $\boldsymbol{U}$ and $\boldsymbol{H}$ are nonnegative matrices. As a result, only additive combinations of multiple basis images (i.e., the columns in $\boldsymbol{U}$) are allowed and no subtractions can occur. In this way, NMF is compatible with the intuitive notion of combining parts to form a whole, and this is how NMF learns a parts-based representation [6].

A popular approach to address the NMF problem is to formulate it as the optimization problem

$$\underset{\boldsymbol{U}, \boldsymbol{H}}{\text{minimize}} \quad \| \boldsymbol{X} - \boldsymbol{U} \cdot \boldsymbol{H} \|_F \tag{1.15a}$$

$$\text{subject to:} \quad \boldsymbol{U} \ge \mathbf{0}, \boldsymbol{H} \ge \mathbf{0} \tag{1.15b}$$

where $\| \boldsymbol{E} \|_F$ denotes the Frobenius norm of matrix $\boldsymbol{E} = \{e_{i,j}\}$, which is defined by

$$\| \boldsymbol{E} \|_F = \left(\sum\nolimits_i \sum\nolimits_j e_{i,j}^2 \right)^{1/2},$$

and constraints $\boldsymbol{U} \ge \mathbf{0}, \boldsymbol{H} \ge \mathbf{0}$ signify nonnegativeness of these matrices [7]. A variant of (1.15), which is often shown to provide improved solutions, is the *regularized* problem

$$\underset{\boldsymbol{U}, \boldsymbol{H}}{\text{minimize}} \quad \| \boldsymbol{X} - \boldsymbol{U} \cdot \boldsymbol{H} \|_F + \mu_1 \sum_{i=1}^{d} \sum_{j=1}^{q} | u_{i,j} | + \mu_2 \sum_{i=1}^{q} \sum_{j=1}^{m} | h_{i,j} | \tag{1.16a}$$

$$\text{subject to:} \quad \boldsymbol{U} \ge \mathbf{0}, \boldsymbol{H} \ge \mathbf{0} \tag{1.16b}$$

where the second and third terms in (1.16a) are included to promote solution sparsity, and parameters $\mu_1 > 0$ and $\mu_2 > 0$ are used to control the trade-off between approximation fidelity and solution sparsity.

Suppose an NMF of a data set $\boldsymbol{X}$ has been obtained by solving the constrained problem (1.15) or (1.16) and the approximation error $\| \boldsymbol{X} - \boldsymbol{U} \cdot \boldsymbol{H} \|_F$ is found to be reasonably small, then we denote the data set $\boldsymbol{X}$ and matrix $\boldsymbol{H}$ explicitly as

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_m \end{bmatrix}, \ \boldsymbol{H} = \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_m \end{bmatrix} \text{ with } \boldsymbol{x}_i \in R^{d \times 1} \text{ and } \boldsymbol{h}_i \in R^{q \times 1}$$

and claim that

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_m \end{bmatrix} \approx \boldsymbol{U} \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{U}\boldsymbol{h}_1 & \boldsymbol{U}\boldsymbol{h}_2 & \cdots & \boldsymbol{U}\boldsymbol{h}_m \end{bmatrix}$$

which implies that

$$\boldsymbol{x}_i \approx \boldsymbol{U}\boldsymbol{h}_i \quad \text{for } i = 1, 2, \ldots, m \tag{1.17}$$

Since the q columns of nonnegative matrix $\boldsymbol{U}$ can be interpreted as “parts” of the “objects” (i.e. data points) in $\boldsymbol{X}$, (1.17) simply says that the ith object $\boldsymbol{x}_i$ can be approximately re-produced by using the q components of $\boldsymbol{h}_i$ as weights to linearly combine the parts $\{\boldsymbol{u}_k, k = 1, 2, \ldots, q\}$. For this, the columns $\{\boldsymbol{h}_i, i = 1, 2, \ldots, m\}$ of $\boldsymbol{H}$ can be used as q-dimensional features to represent the corresponding d-dimensional data points $\{\boldsymbol{x}_i, i = 1, 2, \ldots, m\}$.

Example 1.7 Extract 2-dimensional features of Fisher's data set for Iris plants (see Example 1.2) by solving an NMF problem with $q = 2$.

Solution

With Fisher's data set as a matrix $\boldsymbol{X}$ of 4×150 and $q = 2$, NMF of $\boldsymbol{X}$ was performed by solving optimization problem (1.15). The columns of the optimal $\boldsymbol{H}$ were taken as 2-dimensional features of the data samples. Fig. 1.6 shows these points in blue, red, and black for Setosa, Versicolor, and Virginica, respectively. It is observed that the 2-dim features for Setosa and Versilcolor are well separated while the 2-dim features for Versilcolor and Virginica are not linearly separated (two sets of points are said to be linearly separable if there exists a straight line (hyperplane) that separates the two sets). ■

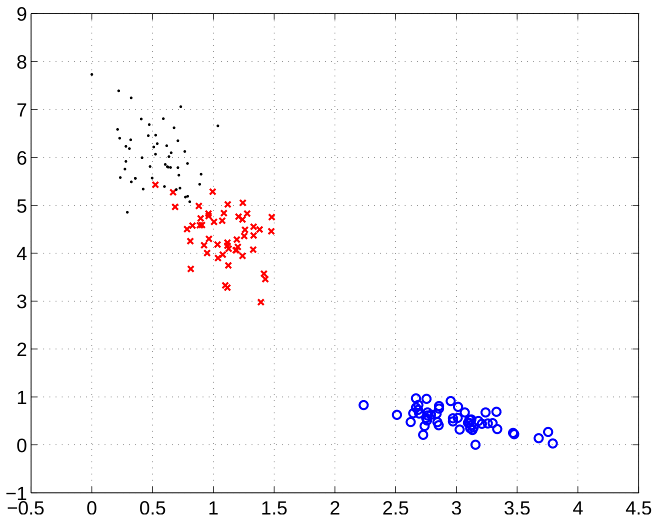
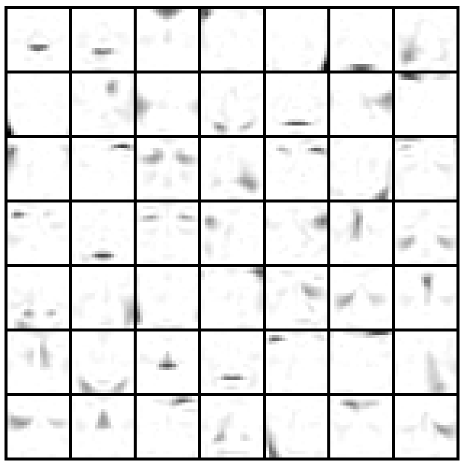


Figure 1.6. NMF-based two-dimensional features of Fisher's Iris plants. Points in blue, red, and black represent Setosa, Versicolor, and Virginica Iris, respectively.

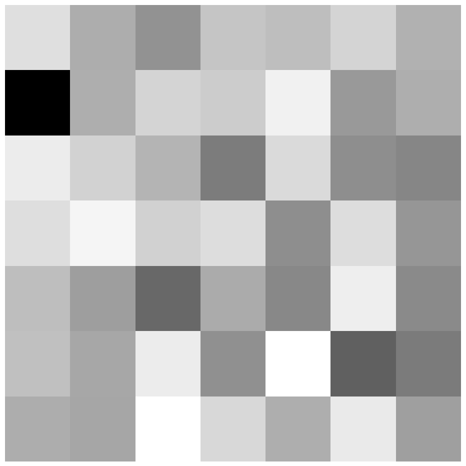
Example 1.8 Apply NMF to the facial images from database CBCL (see Example 1.3).

Solution

In this case we are dealing with a dataset $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_{2429}\end{bmatrix} \in R^{361\times2429}$, so the NMF problem in question has a size of $d = 361$ and $m = 2429$. With $q = 49$, NMF of $\boldsymbol{X}$ was performed by solving optimization problem (1.16) where the regularization parameters were set to $\mu_1 = 0.015$ and $\mu_2 = 1$. The 49 columns of the optimized matrix $\boldsymbol{U}$ are shown in Fig. 1.7(a) where each column was converted into an image of 19×19 pixels. These images provide several versions of parts of human face. Fig. 1.7(b) displays the values of a typical column of the optimized matrix $\boldsymbol{H}$, where the column has been converted into an image of 7×7 pixels with larker ones representing larger numerical values. Figure 1.8 provides visual comparison between the first 49 facial images of the database with their approximations by NMF. ■

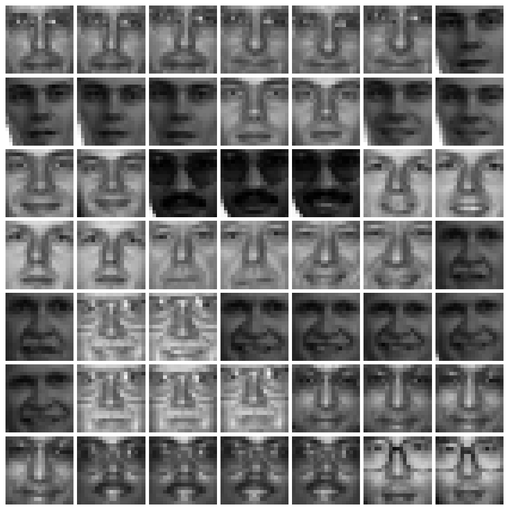


(a)

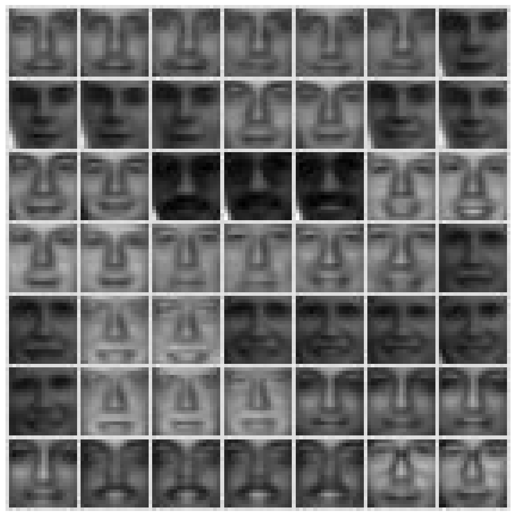


(b)

Figure 1.7. (a) The columns of optimized matrix $\boldsymbol{U}$, here each column is converted into a gray-scale image of 19 x 19 pixels; (b) a typical column of optimized matrix $\boldsymbol{H}$, here the column has been converted into an image of 7 x 7 pixels where darker pixels represent larger numerical values.



(a)



(b)

Figure 1.8. (a) The first 49 facial images from database CBCL (see Example 1.5) and (b) their approximations by NMF.

1.4 Examples of Unsupervised Learning

In *unsupervised learning*, we are given a set of samples $\mathcal{D} = \{\boldsymbol{x}_n, n = 1, 2, \ldots, N\}$, but their labels y_n are *not* available. Typical learning problems in this case are pattern classifications and structure identification, such as clustering, of unlabeled datasets.

As Chris Bishop said, "the most general, and most natural, framework in which to formulate solutions to pattern recognition problems is a statistical one, which recognizes the probabilistic nature both of the information we seek to process, and of the form in which we should express the results [4].” Here we illustrate his point by considering a general classification problem where we are given a training dataset containing K data classes $\{\boldsymbol{C}_k, k = 1, 2, \ldots, K\}$ and the goal is to classify a new input sample $\boldsymbol{x}$ in such a way as to minimize the probability of misclassification. In theory, the solution is that $\boldsymbol{x}$ belongs to class k^* if the *conditional probability* $P(\boldsymbol{C}_{k^*} \mid \boldsymbol{x})$, which by definition is the probability that the class is $\boldsymbol{C}_{k^*}$ given input data $\boldsymbol{x}$, reaches the maximum among all $\{P(\boldsymbol{C}_k \mid \boldsymbol{x}), k = 1, 2, \ldots, K\}$, that is

$$\text{class } k^* = \arg \max_{1 \le k \le K} P\left(\boldsymbol{C}_k \mid \boldsymbol{x}\right) \tag{1.18}$$

In the literature, $P(\boldsymbol{C}_k \mid \boldsymbol{x})$ is called *posterior* probability because it is about probability that the class is $\boldsymbol{C}_k$ *after* data $\boldsymbol{x}$ has been given. In practice, however, solving a classification problem based on (1.18) is not a trivial matter because calculating or estimating conditional probability $P(\boldsymbol{C}_k \mid \boldsymbol{x})$ can be challenging.

Another type of unsupervised learning is *clustering*. Fig. 1.9 illustrates an instance where a set of 100 unlabeled random data points are grouped into two clusters whose centroids are marked with an “**x**”.

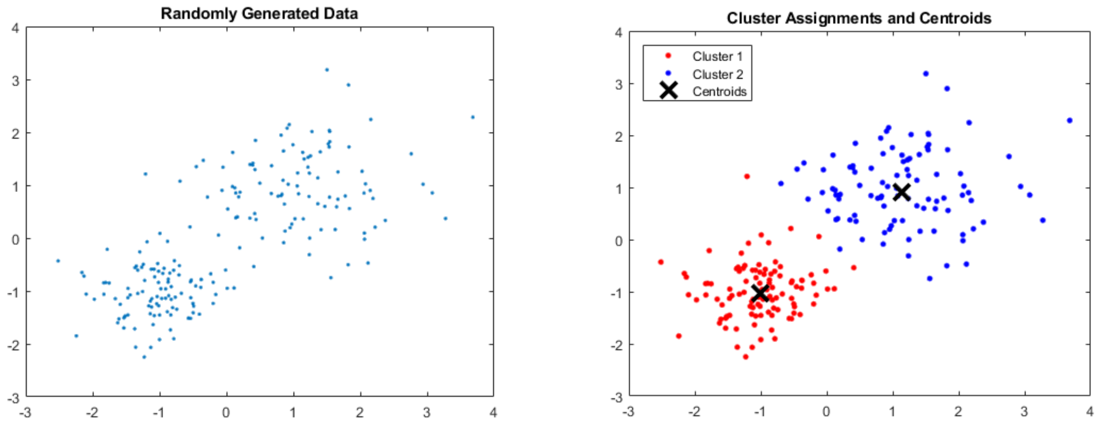


Figure 1.9. An example where random data points on the left are clustered by K-means algorithm.

In general, suppose we are given a dataset $\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ with $\boldsymbol{x}_n \in R^d$, and we want to divide the dataset into K groups, called *clusters*. Each cluster is supposed to have a “centre” and hence there are K centres, denoted by $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K\}$. Below we describe a popular descent clustering method known as the K-means algorithm.

***K*-means clustering**

It is rational to argue that a data point belongs to the *k*th cluster if it is closest to centre $\boldsymbol{\mu}_k$ in Euclidean distance among all centres. The goal of the K-means algorithm [8] is to find a set of centers such that the sum of the squares of the Euclidean distance of all data points to their respective centres reaches a minimum. Obviously this is an optimization problem. The K-means algorithm quantifies the problem by defining its objective function as

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{n,k} \| \boldsymbol{x}_n - \boldsymbol{\mu}_k \|_2^2 \tag{1.19}$$

where $\{r_{n,k}\}$ are scalars which obey the convention that, among the K scalars $\{r_{n,1}, r_{n,2}, \ldots, r_{n,K}\}$ for each index n, there is one and only one scalar (say, $r_{n,j}$ some j) to be assigned to unity while the others are all set to zero. Consequently, for any given n the sum $\sum_{k=1}^{K} r_{n,k} \| \boldsymbol{x}_n - \boldsymbol{\mu}_k \|_2^2$ as seen in (1.19) actually reduces to a single term $\| \boldsymbol{x}_n - \boldsymbol{\mu}_j \|_2^2$ for some j. In this way, the objective function in (1.19) is actually sum of the squares of the Euclidean distance of all data points to their respective "centres", although these centres are yet to be optimized.

The K-means clustering is an iterative algorithm that carries out several rounds of alternating minimization of function J : It minimizes function J with respect to $\{r_{n,k}\}$ while the centres $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K\}$ are held fixed, then it minimizes function J w.r.t. $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K\}$ while $\{r_{n,k}\}$ are held fixed. The procedure repeats until both $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K\}$ and $\{r_{n,k}\}$ remain unaltered, and the final $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K\}$ is taken as the optimal centres. The dataset $\boldsymbol{X}$ can then be clustered easily according to their distances to the optimal centres. Below are technical details in implementing these two steps in each iteration.

1. Hold $\{\boldsymbol{\mu}_k, k = 1, 2, ..., K\}$ fixed while minimizing J w.r.t. $\{r_{n,k}\}$. This is done set by set for a total of N sets of scalars of the form $\{r_{n,1}, r_{n,2}, \ldots, r_{n,K}\}$ where $n = 1, 2, \ldots, N$ and N is the total number of input samples. By inspection, the scalars $\{r_{n,1}, r_{n,2}, \ldots, r_{n,K}\}$ for a given index n that minimize function J is given by

$$\{r_{n,k}\} = \begin{cases} 1 \quad \text{if } \| \boldsymbol{x}_n - \boldsymbol{\mu}_k \|_2 \text{ is the minimum of } \{ \| \boldsymbol{x}_n - \boldsymbol{\mu}_j \|_2 \text{ for } j = 1, ..., K\} \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \tag{1.20}$$

where $k = 1, 2, ..., K$.

Having optimized $\{r_{n,k}, k = 1, 2, ..., K; n = 1, 2, ..., N\}$, the current K clusters are identified as

$$\text{Cluster } C_k = \{\text{all } \boldsymbol{x}_n\text{'s with } r_{n,k} = 1\} \quad \text{for } k = 1, 2, \ldots, K \tag{1.21}$$

2. Hold $\{r_{n,k}\}$ fixed while minimizing function J w.r.t. $\{\boldsymbol{\mu}_k, k = 1, 2, \ ..., K\}$. Because J is quadratic and convex w.r.t. $\{\boldsymbol{\mu}_k\}$, optimal $\{\boldsymbol{\mu}_k\}$ can be found by setting the gradient of J w.r.t. $\{\boldsymbol{\mu}_k\}$ to zero. From (1.19), we compute the gradient of J w.r.t. $\boldsymbol{\mu}_k$:

$$\nabla_{\boldsymbol{\mu}_k} J = 2 \sum_{n=1}^{N} r_{n,k} (\boldsymbol{x}_n - \boldsymbol{\mu}_k).$$

By letting $\nabla_{\boldsymbol{\mu}_k} J = \boldsymbol{0}$ and the equation for $\{\boldsymbol{\mu}_k\}$, we obtain

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{n,k}\boldsymbol{x}_n}{\sum_n r_{n,k}} \quad \text{for } k = 1, 2, ..., K \tag{1.22a}$$

From the definition of $r_{n,k}$ in (1.20), it follows that the numerator in (1.22a) is equal to the sum of all data points in cluster C_k while the denominator in (1.22a) is equal to the number of data points in cluster C_k. Therefore, formula (1.22a) says that the optimal centre $\boldsymbol{\mu}_k$ is equal to the *mean* of the points $\{\boldsymbol{x}_n\}$ contained in cluster C_k, that is,

$$\boldsymbol{\mu}_k = \text{mean of } \{\boldsymbol{x}_n \in C_k\} \quad \text{for } k = 1, 2, ..., K \tag{1.22b}$$

and this explains the term "K-means" for the algorithm.

We remark that the algorithm assures its convergence because as the iteration proceeds the objective function J decreases monotonically, and J is bounded from below as it is always a positive quantity. However, the solution obtained may not be the global minimizer because, with respect to both $\{\boldsymbol{\mu}_k\}$ and $\{r_{n,k}\}$, function J in (1.19) is *nonconvex*. In other words, with different assignment of the initial centres the algorithm may converge to a different group of clusters. The algorithm is now summarized as follows.

***K-means clustering* algorithm**

Step 1 Input: dataset $\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, number of clusters K, initial centres $\{\boldsymbol{\mu}_1^{(0)}, \boldsymbol{\mu}_2^{(0)}, \ldots, \boldsymbol{\mu}_K^{(0)}\}$, and convergence tolerance ε. Set $t = 0$.

Step 2 Compute $\{r_{n,k}\}$ for $n = 1, 2, ..., N$ and $k = 1, 2, ..., K$ using (1.20).

Step 3 Using (1.21) to regroup the data points into K clusters $\{C_k, k = 1, 2, ..., K\}$.

Step 4 Using (1.22) to update the centres to $\{\boldsymbol{\mu}_1^{(t+1)}, \boldsymbol{\mu}_2^{(t+1)}, \ldots, \boldsymbol{\mu}_K^{(t+1)}\}$.

Step 5 If $\sum_{k=1}^{K} \|\boldsymbol{\mu}_k^{(t+1)} - \boldsymbol{\mu}_k^{(t)}\|_2 < \varepsilon$, output $\{\boldsymbol{\mu}_1^{(t+1)}, \boldsymbol{\mu}_2^{(t+1)}, \ldots, \boldsymbol{\mu}_K^{(t+1)}\}$ as the solution centres, use them in (1.20) to update $\{r_{n,k}\}$, then use (1.21) to obtain K final clusters $\{C_k, k = 1, 2, ..., K\}$ and stop; otherwise, set $t := t + 1$ and repeat from Step 2.

Example 1.9 Depicted in Fig. 1.10(a) is a dataset $\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ with $d = 2$ and $N = 300$. Visual inspection of $\mathcal{D}$ suggests that it is rational to divide the dataset into $K = 3$ clusters. With $\varepsilon = 10^{-6}$ and initial centers $\boldsymbol{\mu}_1 = [-2 \quad 2]^T$, $\boldsymbol{\mu}_2 = [2 \quad -2]^T$, and $\boldsymbol{\mu}_3 = [3 \quad -1]^T$ which give initial value of the objective $J = 2.5370 \times 10^3$, it took the k-means algorithm 4 iterations to converge to 3 clusters with centers $\boldsymbol{\mu}_1 = [0.0028 \quad 0.0875]^T$, $\boldsymbol{\mu}_2 = [-1.4721 \quad -1.5018]^T$, and $\boldsymbol{\mu}_3 = [1.4559 \quad 1.6394]^T$. With the optimized centres, the objective was reduced to $J = 144.9881$. Figs. 1.10(b)-(e) show the 3 clusters obtained with 1, 2, 3, and 4 iterations, respectively, while Fig. 1.10(f) shows a profile of the objective J versus number of iterations. ■

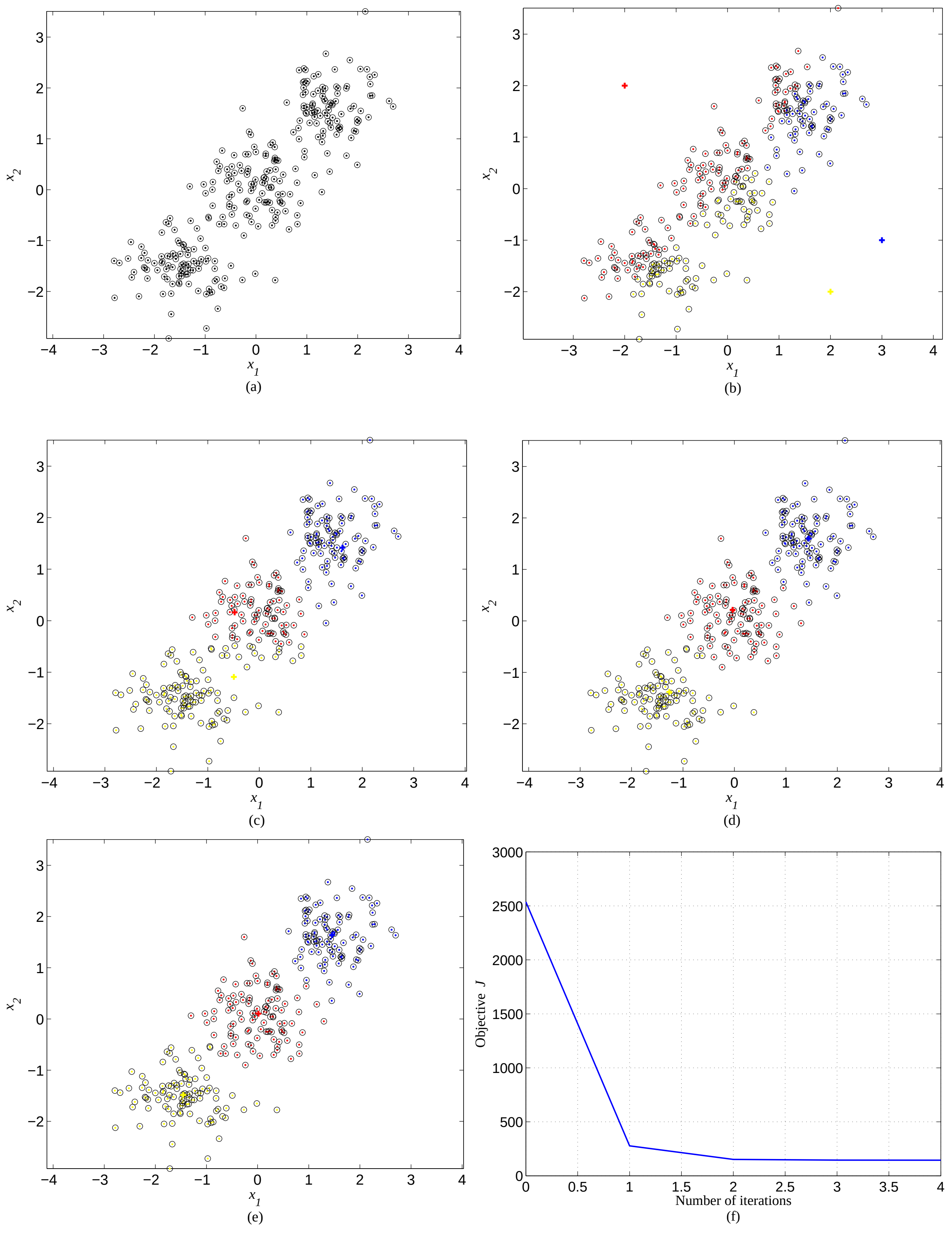


Figure 1.10. Clustering data points by k-means with $k = 3$ for Example 1.9

1.5 Examples of Supervised Learning

In supervised learning we are given a dataset with labels that typically assumes the form $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, 2, ..., N\}$ where $\boldsymbol{x}_n$ are called *examples* (or *samples* or *input data points* etc.) from space $\mathcal{X}$ and $\boldsymbol{y}_n \in \mathcal{Y}$ is a label associated with $\boldsymbol{x}_n$. A common viewpoint in contemporary ML is that data $\mathcal{D}$ were produced by an *unknown* "*governing law*", and we want to utilize the data to infer this governing law. From a statistical perspective, this governing law is seen as an *unknown conditional probability distribution* $P(\boldsymbol{y} \mid \boldsymbol{x})$, or an *unknown target function* $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ denotes random noise.

A distinct feature of supervised learning that differs from unsupervised learning is that supervised learning takes full advantages of the availability of data labels to facilitate a *generalization* process that infers a new (previously unseen) data sample $\boldsymbol{x}$ *outside* $\mathcal{D}$. In effect the term *supervised learning* came from the fact that learning is supervised with a *labeled* training dataset.

Below we present some examples to further illustrate the concept of supervised leaning.

Optimizing a Linear Model for Prediction

Given a training dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, 2, ..., N\}$ with $\boldsymbol{x}_n \in R^{d\times 1}$ and $\boldsymbol{y}_n \in R^{l\times 1}$, we consider using a *linear model*

$$f(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b} \tag{1.23}$$

to predict or classify new data, where weight $\boldsymbol{W} \in R^{d\times l}$ and bias $\boldsymbol{b} \in R^{l\times 1}$ are parameters to be optimized to achieve best prediction accuracy $f(\boldsymbol{x}_n, \boldsymbol{W}, \boldsymbol{b}) \approx \boldsymbol{y}_n$ for n = 1, 2, …, N in least-squares (LS) sense. We start by introducing an objective (loss) function that measures assumes the form $\sum_{n=1}^{N} \| f(\boldsymbol{x}_n, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{y}_n \|_2^2$ to measure the prediction error over the entire training set, where parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ are to be optimized by minimizing this loss function, that is

$$\underset{\boldsymbol{W},\boldsymbol{b}}{\text{minimize}} \ \sum_{n=1}^{N} \| f(\boldsymbol{x}_n, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{y}_n \|_2^2 \tag{1.24}$$

To address problem (1.24), we express parameters $\boldsymbol{W}$, $\boldsymbol{b}$, and label $\boldsymbol{y}_n$ explicitly as

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_l \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_l \end{bmatrix}, \text{ and } \boldsymbol{y}_n = \begin{bmatrix} y_1^{(n)} \\ y_2^{(n)} \\ \vdots \\ y_l^{(n)} \end{bmatrix} \tag{1.25}$$

and use (1.23) to write the nth term in (1.24) as

$$\| f(\boldsymbol{x}_n, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{y}_n \|_2^2 = \| \boldsymbol{W}^T \boldsymbol{x}_n + \boldsymbol{b} - \boldsymbol{y}_n \|_2^2 = \left\| \begin{bmatrix} \boldsymbol{w}_1^T \boldsymbol{x}_n \\ \boldsymbol{w}_2^T \boldsymbol{x}_n \\ \vdots \\ \boldsymbol{w}_l^T \boldsymbol{x}_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_l \end{bmatrix} - \begin{bmatrix} y_1^{(n)} \\ y_2^{(n)} \\ \vdots \\ y_l^{(n)} \end{bmatrix} \right\|_2^2 = \left\| \begin{bmatrix} \boldsymbol{w}_1^T \boldsymbol{x}_n + b_1 - y_1^{(n)} \\ \boldsymbol{w}_2^T \boldsymbol{x}_n + b_2 - y_2^{(n)} \\ \vdots \\ \boldsymbol{w}_l^T \boldsymbol{x}_n + b_l - y_l^{(n)} \end{bmatrix} \right\|_2^2 = \sum_{i=1}^{l} \left(\boldsymbol{w}_i^T \boldsymbol{x}_n + b_i - y_i^{(n)} \right)^2$$

Therefore, we can write the loss function in (1.24) as

$$\sum_{n=1}^{N} \| f(\boldsymbol{x}_n, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{y}_n \|_2^2 = \sum_{n=1}^{N} \sum_{i=1}^{l} \left(\boldsymbol{w}_i^T \boldsymbol{x}_n + b_i - y_i^{(n)} \right)^2 = \sum_{i=1}^{l} \sum_{n=1}^{N} \left(\boldsymbol{w}_i^T \boldsymbol{x}_n + b_i - y_i^{(n)} \right)^2$$

where the internal sum can be expressed as

$$\sum_{n=1}^{N} \left(\boldsymbol{w}_i^T \boldsymbol{x}_n + b_i - y_i^{(n)} \right)^2 = \left\| \hat{\boldsymbol{X}} \hat{\boldsymbol{w}}_i - \hat{\boldsymbol{y}}_i \right\|_2^2 \quad \text{for } i = 1, 2, \ldots, l$$

with

$$\hat{\boldsymbol{X}} = \begin{bmatrix} \hat{\boldsymbol{x}}_1^T \\ \hat{\boldsymbol{x}}_2^T \\ \vdots \\ \hat{\boldsymbol{x}}_N^T \end{bmatrix} \quad \text{where } \hat{\boldsymbol{x}}_n = \begin{bmatrix} \boldsymbol{x}_n \\ 1 \end{bmatrix} \text{ for } n = 1, 2, \ldots, N, \tag{1.26a}$$

and

$$\hat{\boldsymbol{w}}_i = \begin{bmatrix} \boldsymbol{w}_i \\ b_i \end{bmatrix}, \quad \hat{\boldsymbol{y}}_i = \begin{bmatrix} y_i^{(1)} \\ y_i^{(2)} \\ \vdots \\ y_i^{(N)} \end{bmatrix} \tag{1.26b}$$

As a result, problem (1.24) is now reduced to

$$\underset{\boldsymbol{W}, \boldsymbol{b}}{\text{minimize}} \ \sum_{i=1}^{l} \left\| \hat{\boldsymbol{X}} \hat{\boldsymbol{w}}_i - \hat{\boldsymbol{y}}_i \right\|_2^2 \tag{1.27}$$

where $\{\hat{\boldsymbol{w}}_i, i = 1, 2, ..., l\}$ are variables while $\hat{\boldsymbol{X}}$ and $\{\hat{\boldsymbol{y}}_i, i = 1, 2, ..., l\}$ are known data. To solve (1.27), note that each of the l unknown parameter vectors $\{\hat{\boldsymbol{w}}_j, j = 1, 2, ..., l\}$ appears only in *one* single term in (1.27) and hence (1.27) can be solved by minimizing each $\hat{\boldsymbol{w}}_i$ separately, i.e. by solving

$$\underset{\hat{\boldsymbol{w}}_i}{\text{minimize}} \ \left\| \hat{\boldsymbol{X}} \hat{\boldsymbol{w}}_i - \hat{\boldsymbol{y}}_i \right\|_2^2 \quad \text{for } i = 1, 2, \ldots, l \tag{1.28}$$

The objective function (1.28) is quadratic and convex with regard to $\hat{\boldsymbol{w}}_i$, hence its global minimizer can be obtained by solving the equation $\nabla_{\hat{\boldsymbol{w}}_i} \left\| \hat{\boldsymbol{X}} \hat{\boldsymbol{w}}_i - \hat{\boldsymbol{y}}_i \right\|_2^2 = \boldsymbol{0}$, that is,

$$\hat{\boldsymbol{X}}^T \hat{\boldsymbol{X}} \hat{\boldsymbol{w}}_i = \hat{\boldsymbol{X}}^T \hat{\boldsymbol{y}}_i \quad \text{for } i = 1, 2, \ldots, l \tag{1.29}$$

The equations in (1.29) can be solved together as

$$\begin{bmatrix} \hat{\boldsymbol{w}}_1^* & \hat{\boldsymbol{w}}_2^* & \cdots & \hat{\boldsymbol{w}}_l^* \end{bmatrix} = \left(\hat{\boldsymbol{X}}^T \hat{\boldsymbol{X}} \right)^{-1} \hat{\boldsymbol{X}}^T \begin{bmatrix} \hat{\boldsymbol{y}}_1 & \hat{\boldsymbol{y}}_2 & \cdots & \hat{\boldsymbol{y}}_l \end{bmatrix} = \left(\hat{\boldsymbol{X}}^T \hat{\boldsymbol{X}} \right)^{-1} \hat{\boldsymbol{X}}^T \begin{bmatrix} \boldsymbol{y}_1^T \\ \boldsymbol{y}_2^T \\ \vdots \\ \boldsymbol{y}_N^T \end{bmatrix} \tag{1.30}$$

If we denote each solution vector in (1.30) as $\hat{\boldsymbol{w}}_i^* = \begin{bmatrix} \boldsymbol{w}_i^* \\ b_i^* \end{bmatrix}$, then the optimal $\boldsymbol{W}^*$ and $\boldsymbol{b}^*$ can be constructed as

$$\boldsymbol{W}^* = \begin{bmatrix} \boldsymbol{w}_1^* & \boldsymbol{w}_2^* & \cdots & \boldsymbol{w}_l^* \end{bmatrix} \text{ and } \boldsymbol{b}^* = \begin{bmatrix} b_1^* \\ b_2^* \\ \vdots \\ b_l^* \end{bmatrix} \tag{1.31}$$

and the optimal linear model becomes

$$f(\boldsymbol{x}, \boldsymbol{W}^*, \boldsymbol{b}^*) = \boldsymbol{W}^{*T}\boldsymbol{x} + \boldsymbol{b}^*$$

which sometimes is simply written as

$$f(\boldsymbol{x}) = \boldsymbol{W}^{*T}\boldsymbol{x} + \boldsymbol{b}^* \tag{1.32}$$

We are now in a position to introduce two important concepts, known as regression and classification. Let us cite M. J. Garbade who says the following: "Regression and classification are categorized under the same umbrella of supervised machine learning. The main difference between them is that the output variable in regression is numerical (or continuous) while that for classification is categorical (or discrete)." It follows that (1.32) is a *linear regression model* because the output $f(\boldsymbol{x})$ is a continuous (actually linear) function of the input $\boldsymbol{x}$. In ML, $f(\boldsymbol{x})$ in (1.32) is often called *discriminant function* which is of use both in prediction and classification, as explained below.

(i) *Use* $f(\boldsymbol{x}) = \boldsymbol{W}^{*T}\boldsymbol{x} + \boldsymbol{b}^*$ *for prediction*

If one takes the view that the optimized model (1.32) extracts some sort of intrinsic essence from the correspondences between data $\{\boldsymbol{x}_n, n = 1, 2, \ldots, N\}$ and their labels $\{\boldsymbol{y}_n, n = 1, 2, \ldots, N\}$, then it is justified to use the model to *predict* a label $\boldsymbol{y}$ for a *new* input $\boldsymbol{x}$ as

$$\boldsymbol{y} = \boldsymbol{W}^{*T}\boldsymbol{x} + \boldsymbol{b}^* \tag{1.33}$$

Example 1.11 (see below) illustrates the usefulness of (1.33) for multi-category classification problems.

(ii) *Use* $f(\boldsymbol{x}) = \boldsymbol{W}^{*T}\boldsymbol{x} + \boldsymbol{b}^*$ *for classification*

The problem of data classification may be considered as a special type of prediction problem where the algorithm receives a new data point $\boldsymbol{x}$ outside the training set and predicts a class that point $\boldsymbol{x}$ belongs to. Below we illustrate this point by considering a special case of what we had addressed above, where the labels of the data set are scalars, namely the dimension of space $\mathcal{Y}$ is $l = 1$. In this case, the data set assumes the form $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$ with $\boldsymbol{x}_n \in R^{d\times 1}$ and $y_n \in R$. Accordingly, matrix $\boldsymbol{W}$ is reduced to a column vector $\boldsymbol{w}$ of dimension d, and bias b is a scalar. As a result, (1.29) is reduced to a single equation

$$\hat{\boldsymbol{X}}^T \hat{\boldsymbol{X}} \hat{\boldsymbol{w}} = \hat{\boldsymbol{X}}^T \boldsymbol{y} \tag{1.34}$$

where

$$\hat{\boldsymbol{w}} = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

and the discriminant function in (1.32) becomes

$$f(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x} + b^* \tag{1.35}$$

where $\boldsymbol{w}^*$ and b^* are obtained by solving linear equation (1.34).

The linear regression model in (1.35) is useful for *two-category classification* problems where the training dataset contains *two classes* of examples, denoted by $\mathcal{P}$ and $\mathcal{N}$, and assumes the form $\mathcal{D}$ = $\{(\boldsymbol{x}_n, y_n), n = 1,2,\ldots,N\}$ where $y_n = 1$ if $\boldsymbol{x}_n$ belongs to class $\mathcal{P}$ and $y_n = -1$ if $\boldsymbol{x}_n$ belongs to class $\mathcal{N}$. Under these circumstances, classification of a new input $\boldsymbol{x}$ *outside* dataset $\mathcal{D}$ can be carried out as

$$\begin{cases} \boldsymbol{x} \text{ belongs to class } \mathcal{P} & \text{if } \boldsymbol{x}^T\boldsymbol{w}^* + b^* > 0 \\ \boldsymbol{x} \text{ belongs to class } \mathcal{N} & \text{if } \boldsymbol{x}^T\boldsymbol{w}^* + b^* < 0 \end{cases} \tag{1.36}$$

and the label of a data point $\boldsymbol{x}$ predicted according to (1.36) is given by

$$y = \text{sign}\left(\boldsymbol{w}^{*T}\boldsymbol{x} + b^*\right) \tag{1.37}$$

where by definition $y = \text{sign}(\cdot)$ yields $y = 1$ for any positive input value and $y = -1$ for any negative input value. Equation (1.37) analytically explains how a linear regression model can be of use in two-category classification problems.

An important concept related to this classification procedure is *decision boundary* which is defined as a trajectory produced by setting the discriminant function to zero. From (1.35), the decision boundary is defined by the *linear* equation

$$f(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x} + b^* = 0 \tag{1.38}$$

In geometrical term, (1.38) defines a flat "plane" that divides the entire input space into two parts, and at the points on one side of the plane the values of $f(\boldsymbol{x})$ have the *same sign*, and at the points on the other side of the plane the values of $f(\boldsymbol{x})$ all have the opposite sign.

Example 1.10 The classification method described above was applied to the first two data classes of the dataset "Iris" (see Example 1.2), namely Iris Setosa and Iris Versicolor, for classification. In each class, we select 40 examples at random for training, leaving the rest 10 examples for testing. The labels of Iris Setosa and Iris Versicolor were assigned to 1 and -1, respectively, thus vector $\boldsymbol{y}$ in (1.34) was constructed as `y = [ones(40,1); -ones(40,1)]`. The data matrix $\hat{\boldsymbol{X}}$ is of size 80×5. The unique solution of equation (1.34) was found to be

$$\hat{\boldsymbol{w}}^* = \begin{bmatrix} 0.4408 & 0.0389 & 0.3105 & -0.4237 & -0.5052 \end{bmatrix}^T$$

hence

$$\boldsymbol{w}^* = [0.0389 \;\; 0.3105 \;\; -0.4237 \;\; -0.5052]^T \quad \text{and} \quad b^* = 0.4408$$

Using (1.37), the 20 Iris in the test data were all classified correctly. Fig. 1.11 depicts the values of $\boldsymbol{w}^{*T}\boldsymbol{x}+b^*$ for the 20 test samples. ■

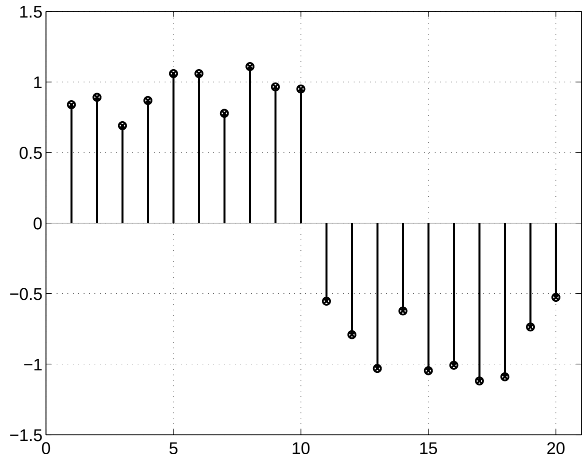


Figure 1.11 Values of $\boldsymbol{w}^{*T}\boldsymbol{x}+b^*$ for the test data, the first 10 samples were from Iris Setosa and the rest 10 samples from Iris Versicolor. Using (1.37), all test samples were classified correctly.

As the next example will illustrate, the linear regression model (1.33) is useful for multi-category classification.

Example 1.11 Classify the Iris plants in Example 1.2 as a 3-class dataset using linear regression model (1.33).

Solution

For the dataset from Example 1.2, we assign labels $\boldsymbol{y}_n = [1 \quad 0 \quad 0]^T$, $\boldsymbol{y}_n = [0 \quad 1 \quad 0]^T$, and $\boldsymbol{y}_n = [0 \quad 0 \quad 1]^T$ for Iris Setosa, Versicolor, and Virginica, respectively. This type of labels is considered more appropriate relative to intuitive and seemingly more natural assignments such $y_n = 1$, or 2, or 3, especially when the labels are involved in *numerical calculations* of optimal parameters $\boldsymbol{W}^*$ and $\boldsymbol{b}^*$ as seen in (1.30). In effect, label assignments such as $y_n = 1$, or 2, or 3 tend to associate a kind of "degree of importance" with each individual data classe while the vectorized labels given above eliminate such association so that every data sample is treated equally fairly during the training. We remark that this is accomplished at the cost of going from scalar (i.e. one-dimensional) labels to l-dimensional labels (in this example we have $l = 3$). The good news is that the regression model (1.33), namely $\boldsymbol{y} = \boldsymbol{W}^{*T}\boldsymbol{x} + \boldsymbol{b}^*$, does offer a *vector* output for an input $\boldsymbol{x}$ and hence the model fits nicely into the problem at hand.

As in Example 1.10, from each class we select 40 examples at random for training while leaving the rest 10 examples for testing. With 3 data classes, matrix $\hat{\boldsymbol{X}}$ is of size 120×5, and the labels for the first, second, and third 40 data samples were assigned to $\boldsymbol{y}_n = [1 \quad 0 \quad 0]^T$, $\boldsymbol{y}_n = [0 \quad 1 \quad 0]^T$, and $\boldsymbol{y}_n = [0 \quad 0 \quad 1]^T$, respectively. This specifies the label matrix in (1.30) as

$$\begin{bmatrix} \boldsymbol{y}_1^T \\ \boldsymbol{y}_2^T \\ \vdots \\ \boldsymbol{y}_{120}^T \end{bmatrix} = \begin{bmatrix} \mathbf{1}_{40} & \boldsymbol{z}_{40} & \boldsymbol{z}_{40} \\ \boldsymbol{z}_{40} & \mathbf{1}_{40} & \boldsymbol{z}_{40} \\ \boldsymbol{z}_{40} & \boldsymbol{z}_{40} & \mathbf{1}_{40} \end{bmatrix}_{120\times 3}$$

where $\mathbf{1}_{40}$ and $\boldsymbol{z}_{40}$ denote all-one and all-zero columns of length 40, respectively. Using (1.30) and (1.31), the optimal model parameters were found to be

$$\boldsymbol{W}^* = \begin{bmatrix} 0.0502 & -0.0160 & -0.0341 \\ 0.2442 & -0.4257 & 0.1815 \\ -0.2150 & 0.1929 & 0.0221 \\ -0.0641 & -0.4326 & 0.4967 \end{bmatrix}, \quad \boldsymbol{b}^* = \begin{bmatrix} 0.1751 \\ 1.5257 \\ -0.7008 \end{bmatrix}$$

Having obtained $\boldsymbol{W}^*$ and $\boldsymbol{b}^*$, the classification of a new data point $\boldsymbol{x}$ is performed by the following two steps:

Step 1: Compute $\boldsymbol{y} = \boldsymbol{W}^{*T}\boldsymbol{x} + \boldsymbol{b}^*$

Step 2: If the jth component of $\boldsymbol{y}$ is the largest among its three components, then classify point $\boldsymbol{x}$ to class j.

Classification performance is usually evaluated using a *confusion matrix* of size $K \times K$ where K denotes the number of data classes involved. If we look at a confusion matrix column by column, the sum of all components in jth column tell us how many samples from the jth data class are classified; and the ith component in the jth column indicates how many samples from class j have been classified to class i. For the present example, the confusion matrix is a 3 by 3 matrix. Shown below is the confusion matrix that summarizes the test results obtained.

			Actual Classes	
		Setosa (j = 1)	Versicolor (j = 2)	Virginica (j = 3)
	Setosa (i =1)	10	0	0
Predicted	Versicolor (i = 2)	0	9	0
Classes	Virginica (i = 3)	0	1	10

It is observed that all 10 Iris Setosa and all 10 Iris Virginica in the test data were classified correctly. For the 10 test samples of Versicolor, 9 of them were classified correctly, but one Iris Versicolor was misclassified to the class of Iris Virginica. ■

Problems

1.1 Consider a data matrix

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 1 \end{bmatrix}$$

(a) Compute the eigenvalues of $\boldsymbol{AA}^T$ without any software.

(b) Compute the nonzero eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ with minimum amount of software: compute matrix $\boldsymbol{A}^T\boldsymbol{A}$ by hand, and use MATLAB function **`eig`** to get the eigenvalues.

(c) Use the results from part (a) or (b) to compute the nonzero singular values of $\boldsymbol{A}$.

(d) Use MATLAB function **`svd`** to compute the SVD of matrix $\boldsymbol{A}$ (see Eq. (1.3) of the course notes). Then use the first pair of singular vectors $\boldsymbol{u}_1$ = **`U(:,1)`** and $\boldsymbol{v}_1$ = **`V(:,1)`** and first singular value σ_1 to construct a rank-1 approximation of $\boldsymbol{A}$ as $\boldsymbol{A}_1 = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T$. Report the numerical results.

(e) Compare matrix $\boldsymbol{A}_1$ with the original data matrix $\boldsymbol{A}$ by inspection, and comment on how close they are. An objective evaluation of the approximation error can be made by computing the Frobenius norm of $\boldsymbol{A} - \boldsymbol{A}_1$: the Frobenius norm of a matrix $\boldsymbol{H} = \{h_{i,j}\} \in R^{n\times m}$ is defined by

$$\|\boldsymbol{H}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} h_{i,j}^2}$$

and it can be calculated using MATLAB as **`norm(H,'fro')`**. Use the result from part (d) to compute $\|\boldsymbol{A} - \boldsymbol{A}_1\|_F$ and report your numerical result.

1.2 Download image **`building256.mat`** from the course web site, then in MATLAB normalize the image to **`A = building256/255;`**

(a) Compute the SVD of $\boldsymbol{A}$ (see Eq. (1.3) of the course notes) using MATLAB function **`svd`**.

(b) Denote the ith left- and right-singular vectors of $\boldsymbol{A}$ by $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$, respectively. Construct five rank-q approximations of $\boldsymbol{A}$ with $q = 1, 2, 3, 4, 5$ as follows:

$$\boldsymbol{A}_k = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \qquad \text{for } k = 1, 2, 3, 4, 5$$

(c) Compute the relative approximation errors $e_k = \dfrac{\|\boldsymbol{A}_k - \boldsymbol{A}\|_F}{\|\boldsymbol{A}\|_F}$ for k = 1, 2, 3, 4, 5 and report the numerical results.

(d) Plot images **`A1`**, **`A3`**, and **`A5`** together with the original **`A`** in a single figure for comparison using the code below:

```
figure(1)
subplot(221)
imshow(A)
title('original building256')
subplot(222)
imshow(A1)
```

```
title('rank-1 approximation')
subplot(223)
imshow(A3)
title('rank-3 approximation')
subplot(224)
imshow(A5)
title('rank-5 approximation')
```

(e) Image **building256** (hence matrix A) is of size 256×256, hence you need to save $256^2 = 65,536$ numbers to keep the image. Alternatively, if you happen to like what image **A5** offers, how many numbers do you need to save to keep **A5**? Report your numerical result.

Now if we denote that number by n_5, compute and report $\frac{256^2}{n_5}$ as the "compression ratio" .

1.3 Let A be a data matrix of size $d \times m$, and the compact version of the SVD of A of rank r be given by

$$A = U_r S_r V_r^T$$

where S_r is a diagonal matrix with diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, $U_r = [\boldsymbol{u}_1 \quad \cdots \quad \boldsymbol{u}_r]$ and $V_r = [\boldsymbol{v}_1 \quad \cdots \quad \boldsymbol{v}_r]$ are composed of the first r columns of U and V, which are associated with the r nonzero singular values, respectively. See Eq. (1.6) of the course notes.

(a) Show that if we have already computed the left-singular vectors $\{\boldsymbol{u}_i,\ i = 1, 2, \ldots, r\}$, then the right-singular vectors can be found by using $\boldsymbol{v}_i = \sigma_i^{-1} A^T \boldsymbol{u}_i$ for $i = 1, 2, \ldots, r$.

(b) Similarly, show that if we have already computed the right-singular vectors $\{\boldsymbol{v}_i,\ i = 1, 2, \ldots, r\}$, then the left-singular vectors can be found by using $\boldsymbol{u}_i = \sigma_i^{-1} A \boldsymbol{v}_i$ for $i = 1, 2, \ldots, r$.

1.4 Given an input data set consisting of four data points:

$$X = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \boldsymbol{x}_3 \quad \boldsymbol{x}_4] = \begin{bmatrix} 0.6 & 1 & 1.6 & 2 \\ 0.4 & 1.2 & 1.3 & 2.3 \end{bmatrix}$$

(a) Find mean $\boldsymbol{\mu}$ of the dataset.

(b) Construct centralized dataset $A = [\boldsymbol{x}_1 - \boldsymbol{\mu} \quad \boldsymbol{x}_2 - \boldsymbol{\mu} \quad \boldsymbol{x}_3 - \boldsymbol{\mu} \quad \boldsymbol{x}_4 - \boldsymbol{\mu}]$

(c) Evaluate covariance matrix $C = \frac{1}{4} A A^T$ and find the eigenvector $\boldsymbol{u}_1$ of matrix C corresponding to its largest eigenvalue using MATLAB function **eig**.

(d) Use the result from part (c) to compute the first principal component for each data point.

1.5 With $K = 2$, $\boldsymbol{\mu}_1^{(0)} = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$, and $\boldsymbol{\mu}_2^{(0)} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, apply the *K*-means algorithm to cluster dataset

$$\mathcal{D} = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right\}$$

(a) The first iteration starts with the initial centres $\{\boldsymbol{\mu}_1^{(0)}, \boldsymbol{\mu}_2^{(0)}\}$ and use (1.20) to perform

Step 2 of the algorithm, namely, to compute $\{r_{n,k}\}$ for $n = 1, 2, ..., 6$ and $k = 1$ and 2. It is a good idea to prepare $\{r_{n,k}\}$ as a matrix of size 6×2, and complete the matrix row by row using Eq. (1.20). The matrix $\{r_{n,k}\}$ so completed can then be used together with Eq. (1.21) to identify which points are associated to which centre so as to specify two clusters C_1 and C_2. And this completes Step 3. Next, use (1.22) to update the centres to $\left\{\boldsymbol{\mu}_1^{(1)}, \boldsymbol{\mu}_2^{(1)}\right\}$, and this completes the first iteration. Report the updated clusters C_k for $k = 1$ and 2, and your numerical results of $\left\{\boldsymbol{\mu}_1^{(1)}, \boldsymbol{\mu}_2^{(1)}\right\}$.

(b) Repeat the process in part (a) and report the updated clusters C_k for $k = 1$ and 2, and your numerical results of $\left\{\boldsymbol{\mu}_1^{(2)}, \boldsymbol{\mu}_2^{(2)}\right\}$.

(c) Display in a figure the locations of the six data points as well as the two centres $\left\{\boldsymbol{\mu}_1^{(2)}, \boldsymbol{\mu}_2^{(2)}\right\}$ with two colors (one for each cluster and its centre). Compare the initial centres and clusters with those after two iterations and comment on the results you obtained.

1.6 Repeat Example 1.10, but this time the linear regression method is applied to the classification of the second and third data classes, namely Iris Versicolor and Iris Virginica. As in Example 1.10, in each class 40 samples are selected at random for training, leaving the rest 10 examples for testing. To complete the problem, it is required to compute optimal parameters $\boldsymbol{w}^*$ and b^* for the regression model (1.35) and use the 20 test samples to evaluate the classification performance in terms of a confusion matrix. Report the numerical values of $\boldsymbol{w}^*$, b^*, and the confusion matrix.

To ensure consistent results, use the steps below to prepare the dataset:

1. Download matrix `D_iris.mat` from the course web site.
2. In MATLAB, use `D = D_iris(1:4,51:150)` to keep only the second (Versicolor) and third (Virginica) data classes.
3. Use the code below to prepare 80 samples (40 samples from each class) for training and 20 samples (10 samples from each class) for testing:

```
D1 = D(:,1:50);
D2 = D(:,51:100);
rand('state',15)
r1 = randperm(50);
D1train = D1(:,r1(1:40));
D1test = D1(:,r1(41:50));
rand('state',16)
r2 = randperm(50);
D2train = D2(:,r2(1:40));
D2test = D2(:,r2(41:50));
Dtrain = [D1train D2train];
Dtest = [D1test D2test];
```

NOTE: Here `Dtrain` includes 80 samples for training, with the first 40 samples from Versicolor and the rest 40 samples from Virginica; `Dtest` includes 20 samples for testing, with the first 10 samples from Versicolor and the rest 10 samples from Virginica.