SpringerBriefs in Computer Science

M. N. Murty · M. Avinash

Representation in Machine Learning

Springer

SpringerBriefs in Computer Science

SpringerBriefs present concise summaries of cutting-edge research and practical applications across a wide spectrum of fields. Featuring compact volumes of 50 to 125 pages, the series covers a range of content from professional to academic.

Typical topics might include:

- A timely report of state-of-the art analytical techniques
- A bridge between new research results, as published in journal articles, and a contextual literature review
- A snapshot of a hot or emerging topic
- An in-depth case study or clinical example
- A presentation of core concepts that students must understand in order to make independent contributions

Briefs allow authors to present their ideas and readers to absorb them with minimal time investment. Briefs will be published as part of Springer's eBook collection, with millions of users worldwide. In addition, Briefs will be available for individual print and electronic purchase. Briefs are characterized by fast, global electronic dissemination, standard publishing contracts, easy-to-use manuscript preparation and formatting guidelines, and expedited production schedules. We aim for publication 8–12 weeks after acceptance. Both solicited and unsolicited manuscripts are considered for publication in this series.

**Indexing: This series is indexed in Scopus, Ei-Compendex, and zbMATH **

M. N. Murty • M. Avinash

Representation in Machine Learning

Springer

M. N. Murty
Department of CS and Automation
Indian Institute of Science Bangalore
Bangalore, India

M. Avinash
Indian Institute of Technology Madras
Chennai, India

ISSN 2191-5768 ISSN 2191-5776 (electronic)
SpringerBriefs in Computer Science
ISBN 978-981-19-7907-1 ISBN 978-981-19-7908-8 (eBook)
https://doi.org/10.1007/978-981-19-7908-8

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Overview

This book deals with the most important issue of representation in machine learning (ML). While learning class/cluster abstractions from the data using a machine, it is important to represent the data in a form suitable for effective and efficient machine learning. In this book, we propose to cover a wide variety of representation techniques that are important in both theory and practice.

In practical applications of current interest, the data typically is high dimensional. These applications include image classification, information retrieval, problem solving in AI, biological and chemical structure analysis, and social network analysis. A major problem with such high-dimensional data analysis is that most of the popular tools like the k-nearest neighbor classifier, decision tree classifier, and several clustering algorithms that depend on interpattern distance computations fail to work well. So, representing the data in a lower-dimensional space is inevitable.

Popularly used dimensionality reduction techniques may be categorized as follows:

1. Feature selection schemes: Here an appropriate subset of the given feature set is identified and used in learning.
2. Feature extraction schemes: Here linear or nonlinear combinations of the given features are used in learning.

Some of the popular linear feature extractors are based on principal components, random projections, and nonnegative matrix factorization. We cover all these techniques in the book. There are some misconceptions in the literature on representing the data using a subset of principal components. It is typically believed that the first few principal components make the right choice for classifying the data. We argue and show practically, in the book, how such a practice may not be correct.

It is argued in the literature that deep learning tools are the ideal choices for nonlinear feature selection; also they can learn the representations automatically. These tools include autoencoders and convolutional neural networks. We discuss

these tools in the book. Further, we argue that it is difficult even for the deep learners to automatically learn the representations.

We present experimental results on some benchmark data sets to illustrate various ideas.

Audience

The coverage is meant for both students and teachers and helps practitioners in implementing ML algorithms. It is intended for senior undergraduate and graduate students and researchers working in machine learning, data mining, and pattern recognition. We present material in this book so that it is accessible to a wide variety of readers with some basic exposure to undergraduate-level mathematics. The presentation is intentionally made simpler to make the reader feel comfortable.

Organization

This book is organized as follows: Chapter 1 deals with a generic introduction to machine learning (ML) and various concepts including feature engineering, model selection, model estimation, model validation, and model explanation. Two important tasks in ML are classification and clustering. So, Chap. 2 deals with the representation of data items, classes, and clusters.

Nearest neighbor finding algorithms play an important role in several ML tasks. However, finding nearest neighbors in high-dimensional spaces can be both time consuming and inaccurate. In Chap. 3, we deal with nearest neighbor finding algorithms using fractional norms and approximate nearest neighbor computation using locality-sensitive hashing. We illustrate using several benchmark data sets.

Chapter 4 deals with feature selection and linear feature extraction schemes. It includes discussion on principal components, random projections, and nonnegative matrix factorization. Nonlinear feature extraction schemes are gaining importance because of the deep learning architectures based on autoencoders and multilayer perceptrons. These topics are examined in Chap. 5.

Bangalore, India M. N. Murty
Chennai, India M. Avinash

Contents

Acronyms

AE	Autoencoder
AI	Artificial intelligence
BP	Backpropagation
DTC	Decision tree classifier
JC	Jaccard coefficient
KLD	Kullback-Leibler divergence
KMA	K-means algorithm
KNNC	K-nearest neighbor classifier
LDA	Latent Dirichlet allocation
LSH	Locality sensitive hashing
MDC	Minimal distance classifier
MI	Mutual information
ML	Machine learning
MLP	Multilayer perceptron
MNIST	Modified National Institute of Standards and Technology
NBC	Naïve Bayes classifier
NMF	Nonnegative matrix factorization
NNC	Nearest neighbor classifier
ORL	Olivetti Research Laboratory
PCA	Principal component analysis
RP	Random projections
SOM	Self-organizing map
SVD	Singular value decomposition
SVM	Support vector machine

Chapter 1
Introduction

Abstract In this chapter we provide a brief introduction to machine learning (ML). We introduce some of the important steps in ML including data acquisition, feature engineering, model selection, model estimation, model evaluation, and model explanation.

1.1 Machine Learning (ML) System

Machine Learning (ML) has been an active and important area of study for more than six decades. The phenomenal growth in ML may be attributed, among other things, to the availability of large datasets and improved computing power. It is important to consider Deep Learning (DL) as an offshoot of ML; this is because the generic framework for DL is based on neural networks which were widely studied under the umbrella of ML for more than six decades. In fact, perceptron classifier is the earliest popular ML tool and perceptron forms the basic building block of DL architectures. In the combined space of ML and DL, building a learning system involves the following crucial steps as illustrated in Fig. 1.1.

1. *Data Acquisition:* It depends to a large extent on the knowledge of the application domain. For example, to distinguish between *adults* and *children*, measurements of their *height* and *weight* are adequate; however, to distinguish between *normal* and *abnormal* health status of humans their *body temperature* and *blood test* results might be more important to measure. Further, it is likely that in some domains, there could be missing data possibility. This could be a consequence of being unable to measure the feature value or the entries in the physical records being illegible. There are some statistical tools to deal with the missing values.
2. *Feature Engineering:* This step involves a combination of data preprocessing and representation or dimensionality reduction. Specifically,

M. N. Murty, M. Avinash, *Representation in Machine Learning*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-981-19-7908-8_1

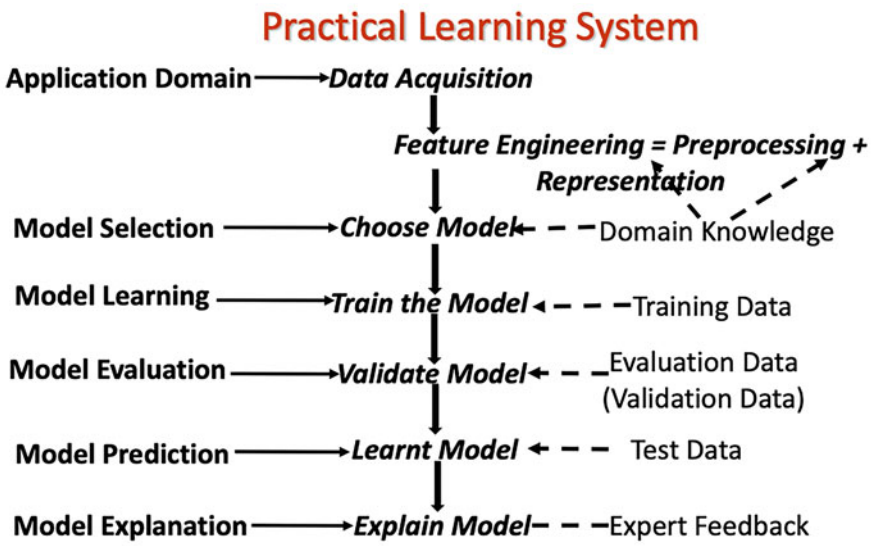

Fig. 1.1 Important steps in machine learning

- **Preprocessing:** Typically there are three different difficulties encountered in data collection.
  a. **Missing Data:** It is possible that some values in the data may be missing. This could be because there was no measurement taken or data was not recorded.
  b. **Data from Different Domains:** The scales behind different features could be highly different. This will bias the proximity measure to be dependent on features that have larger size domains, ignoring the contributions of features with smaller ranges.
  c. **Presence of Outliers:** An outlier is a data item that is either noisy or erroneous. Presence of an outlier could be because of noise in the measuring instruments or erroneous data entries.

3. *Model Selection:* It depends upon the types of data and relevant domain knowledge. For some types of data only a subset of the ML models can be used. For example, if some features are numerical and others are categorical then classifiers based on perceptrons and support vector machines (SVM) are not suitable whereas Bayes classifier and decision-tree based classifiers are ideally suited to deal with such data.
4. *Model Estimation:* It may also be called *model learning*. This learning is dependent on the size and type of the training data. In practice a subset of the labelled data is used as training data and another subset is used for model validation.

5. *Model Evaluation:* This step is also called *model validation*. This step requires a specifically earmarked data called *validation data*. It is possible that the ML model works well on the training data; then we say that the model is well trained. However, it may not work well on the validation data. In such a case, we say that the ML model overfits the training data.
6. *Model Explanation:* This step is important to get expert's attention and feedback from the expert. Explanation had an important role earlier in expert systems and other artificial intelligence (AI) systems. However, explanation has become all the more important in the era of Deep Learning (DL). It is because DL systems typically employ neural networks and their functioning cannot be easily explained at a level of detail that can be appreciated by the domain expert. Such an opaque behaviour has motivated the need for explainable AI.

   It is often stated in DL literature that feature engineering is important in ML, but not in DL. This is a highly debatable issue. It is possible that, in some application domains, DL systems can avoid the *representation* step explicitly. But preprocessing, including handling missing data and eliminating outliers is still an essential part of any DL system. Even though representation is not explicit, it is implicitly handled based on the number of layers and number of neurons each layer of the neural network.

We consider these steps in detail in the subsequent sections.

1.2 Main Steps in an ML System

An ML system is built using several steps. The main steps in the systems are detailed in this section.

1.2.1 Data Collection/Acquisition

The input to an ML system is a set of *data instances or data items*; so data collection is an important step. However, it highly depends on the application domain. For example, consider classification of villages into those which have *fertile lands* versus those which have *barren lands*. In this application, data related to the name of the village, how far away it is from the capital city, what is the height of tallest hillock in the village, etc. may not be so important. On the other hand, data corresponding to the soil moisture content, yield of various crops in the past, cost of a hectare of land, etc. in the village may be more important.

Similarly, in classifying documents into *politics* and *sports* classes, data pertaining to the age of the document, length of the document, and document file type like whether it is a pdf file, a doc file, or an excel file may not be important. However.

frequency of occurrence of some related words like *election, party, candidate, ball, bat, soccer, etc.* are more relevant.

So, the kind of data that needs to be collected or acquired is application/problem specific. In dealing with such practical applications, one will face several difficulties including:

- *Missing Data:* It is possible that in an application domain all the data that one would like to collect is not available. For example, for one book we may have the entire soft copy in the form of a pdf and for another book that is currently out-of-stock, it is possible that a soft copy of only the table of contents is available.

  In the *education* domain, it is possible that students in section A are tested on their understanding of concept 1, whereas students in section B are tested on some other concept, say concept 2, but not on concept 1. So, when this data is used in analysing the performance of students across both the sections, there is a case of missing data as depicted in Table 1.1. Consider the first row in the table; it corresponds to student 1 who is in section A and is not evaluated on concept 2; so, the entry corresponding to concept 2 is NA. Similarly, in the second row corresponding to student 2 from section B, information under concept 1 is not available; so the corresponding entry is NA. There could be other students who are assigned to one of the two sections and so there will be one NA entry in each row for each such student.

  In general, there can be zero or more missing entries corresponding to each item represented as a vector or some other structure like a string, a tree or a graph.
- *Different Types of Data:* There are several applications where we encounter different types of data in the same application. For example, in the agriculture domain consider the application of predicting the yield of soybean crop. A possible set of different data types acquired to build an ML model is shown in Table 1.2. Even in this simple example, one can see that there are different types of data: *types of soybean seeds* is a *binary* variable in this case with two possible

Table 1.1 Example of missing data

Student	Section	Concept 1	Concept 2
Student 1	Section A	90	NA
Student 2	Section B	NA	30
Student 3	Section B	NA	75
	⋮		

Table 1.2 Example of missing data

Attribute	Type of data	Domain
Type of soybean seed	Binary	{type1, type2}
Year	Numerical	[2000, 2015]
Rainfall	Numerical	[83 mm,11871 mm]
Temperature	Numerical	[1.9 °C, 51 °C]
Relative Humidity	Numerical	[50%, 73%]
Soybean Yield	Numerical	[580, 1080]

values type1 and type2 seeds. The variable *Year* indicates the year corresponding to the yield; here the possible values are integers in the range [2000, 2015]. The variables *Temperature, Relative Humidity, and Soybean Yield* are also numerical but their values are floating point numbers. Note that temperature is represented in degrees Celsius, relative humidity in percentage, and soybean yield in quintals per hectare.

In general data can be either:

1. *Categorical:* These are divided into either *nominal* or *ordinal* types. In the case of nominal data, there is no order among the elements of the domain. For example, for geometric shapes the domain is {circle, ellipse, triangle, rectangle, square}. This data is of categorical type and the elements of the domain are not ordered. In the case of ordinal data, there is an inherent order among the values of the domain. For example, the domain of the variable *size* could be {very small, small, medium, large, very large}; here, an ordering among the elements of the domain is observed.
2. *Numerical:* In the case of numerical data, the domain of values of the data type could be either a set of integers or a set of real numbers. For example, in Table 1.2, the domain of *Year* is the set $\{2000, 2001, \cdots, 2015\}$ which is a collection of integer values. Further, the domain of *Relative Humidity* is a collection of floating point numbers in the range [50, 73].

   It is possible to have binary values in the domain for either categorical or numerical. For example domain of *status* could be {Pass, Fail} and this variable is nominal; an example binary ordinal type is {short, tall} of humans based on their height. A very popular binary numerical type is $\{0, 1\}$; also in the classification context, the *class label* data can have the domain $\{-1, +1\}$ where -1 stands for the label of the negative class and $+1$ stands for the label of the positive class.

- *Difference in Domain Sizes:* In an application, if we encounter different types of data, then it is quite likely to have different domain sizes. For example, for the data types shown in Table 1.2, we have binary data (seed type), integer data (Year), and different sized floating point data types (rainfall, temperature, relative humidity, and yield). These size and type differences can create a problem while matching different data items using some proximity score in ML.

  Consider for example, classification of objects into one of two classes: *human* or *chair*. Let the objects be represented by *height* in meters and *weight* in grams. Consider an adult represented by the vector (1.7, 65,000) and a chair represented by the vector (1, 5000) where the heights of the adult and the chair in meters are 1.7 and 1 respectively and the weights of the adult and the chair in grams are 65,000 and 5000 respectively. Note that the domain of *height* can be [0.5, 2.5] and the domain of *weight* can be [2000, 200,000] in this example. So, there is a large difference in the sizes of these two domains.

  In ML, matching is done by using a *proximity measure* which can be either a distance/dissimilarity measure or a similarity measure. Two data items, represented as l-dimensional vectors X and Y, match better when the distance

between them is smaller or when the similarity between them is larger. A popular distance measure is the *euclidean distance* and a popular similarity measure is the *cosine of the angle between vectors*. The euclidean distance is given by

$$d(X, Y) = \sqrt{\sum_{i=1}^{l}(X_i - Y_i)^2}.$$

The cosine similarity is given by

$$cos(X, Y) = \frac{X^t Y}{||X||||Y||},$$

where $X^t Y$ is the dot product between the vectors X and Y, and $||X||$ is the euclidean distance between X and the origin.

Now the euclidean distance between the adult and chair vectors given above is

$$\sqrt{1.7 - 1)^2 + (65{,}000 - 5000)^2} = \sqrt{0.49 + 3.6 \times 10^9} = 60{,}000.000004 \approx 60{,}000.$$

Similarly, cosine of the angle between adult and chair vectors is

$$\frac{1.7 + 325 \times 10^6}{\sqrt{25{,}000{,}001} \times \sqrt{4{,}225{,}000{,}002.89}} = 0.9999999849 \approx 1.0.$$

Note that the proximity values computed between the two vectors, whether it is the euclidean distance or the cosine of the angle between the two vectors are dependent largely upon only one of the two features, that is $weight$ and the contributions of $height$ are negligible. This is because of the difference in the magnitudes of the two data types. This example illustrates how the magnitudes/ranges of values of different data types contribute differently to the overall proximity. A solution to this problem is provided by normalizing the data. A simple $normalization$ scheme scales the magnitude of values such that all the features have the same range. For example, a popular scheme is to transform the data vectors so that they are zero mean unit variance vectors.

- *Presence of Outliers:* A common problem across various applications is the presence of outliers. A data item is usually called an *outlier* if it

  1. assumes values that are *far away* from those of the average data items
  2. *deviates* from the normally behaving data item
  3. is not connected/similar to any other object in terms of its characteristics.

  Outliers can occur because of different reasons including

1. *Noisy measurements:* The measuring instruments may malfunction and may lead to recording of noisy data. It is possible that the recorded value lies outside the domain of the data type.
2. *Erroneous data entry:* Outlying data can occur at the data entry level itself. For example, it is very common to introduce spelling mistakes while entering names. Further, it is common to introduce errors in the names because of typing the neighboring characters, instead of the intended character, on the key board. Also, it is possible to enter the salary erroneously as 2,000,000 instead of 200,000 by typing an extra zero (0).
3. *Evolving systems:* It is possible to encounter data items in sparse regions during the evolution of a system. For example, it is common to encounter isolated entities during the early times of a social network. Such isolated entities may or may not be outliers.
4. *Very naturally:* Instead of viewing an outlier as a noisy or unwanted data item, it may be very useful in several applications. For example, a novel idea or a breakthrough in a scientific discipline, a highly paid sportsperson, an expensive car can all be useful and influential outliers.

An outlying data item can be either out-of-range or with-in-range. For example, consider an organization in which the salary values are

$$\{10{,}000,\ 15{,}0000,\ 225{,}000,\ 300{,}000\}.$$

In this case an entry like 2,250,000 is an out-of-range outlier that occurs because of an erroneous zero (0). Also if there are only 500 people drawing 10,000, 400 drawing 150,000, 300 at the salary level 225,000 and 175 drawing 300,000, then an entry like 270,000 could be a with-in-range outlier.

There are different schemes for detecting outliers. They are based on density around various points in the data. If a data point is located in a sparse region, then it could be a possible outlier. It is possible to use clustering to locate such outliers. It does not matter whether it is within range or out of range. If the clustering output has a singleton cluster, that is a one element cluster, then it could be a possible outlier.

1.2.2 Feature Engineering and Representation

In the previous subsection, we have seen several issues associated with data items. It is convenient and customary to view these items as vectors. *Feature engineering* (FE) is concerned with addressing several problems associated with data vectors. In this section we will examine how these problems could be handled.

- *Missing Data:* There are some ML algorithms that can work even when there are missing data values and in such cases there is no need to do any pre-processing. However there are a large number of other cases where the classifiers cannot

handle missing values. So, there is a need to examine techniques for dealing with missing data. Some possibilities are:

– *Use the nearest neighbour:* Let X be a d-dimensional data vector that has its ith component x_i missing. Let $\mathcal{X} = \{X^1, X^2, \cdots, X^n\}$ be the set of n training pattern vectors. Let $X^i \in \mathcal{X}$ be the nearest neighbour of X based on the remaining $l-1$ (excluding the ith) components. The nearest neighbour of X, $NN(X)$ is defined as

$$NN(X) = argmin_{X^j \in \mathcal{X}} d(X, X^j),$$

where X^j is the jth training pattern and $d(X, X^j)$ is the euclidean distance between X and X^j. Intuitively, $NN(X)$ is in the proximity of X; so $NN(X)$ is at a minimum euclidean distance from X and is maximally similar to X. Predict the value of x_i to be x_i^p, that is if the ith component, x_i, of X is missing, then use the ith component of $X^p = NN(X)$ instead.

– *Use a larger neighbourhood:* It is well-known that the NN-based scheme can be easily affected by outliers. For example, if $NN(X)$ is an outlier, then the value predicted based on $NN(X)$, that is an outlier pattern can be erroneous. Even if $NN(X)$ is not an outlier, it is advisable to use more neighbors of X for predicting the missing value x_i in a robust manner. So, we use K nearest neighbors (KNNs) of X to predict the missing x_i. Let the KNNs of X, using the remaining $l-1$ components, from $\mathcal{X}$ be $X1, X2, \cdots, XK$. Now the predicted value of x_i is the average of the ith components of these KNNs. That is the predicted value of x_i is

$$\frac{1}{K}\sum_{j=1}^{K} Xj_i.$$

– *Cluster the data and locate the nearest cluster:* This approach is based on clustering the training data and locating the cluster to which X belongs based on the remaining $l-1$ components. Let X with its x_i value missing belong to cluster C^q. Let μ^q be the centroid of C^q. Then the predicted value of x_i is μ_i^q, the ith component of μ^q.

Note that the approaches mentioned above to deal with missing data are all applicable when the components of the vectors are all numerical. However when the data vectors have categorical entries, then one needs to use median or mode instead of centroid. We consider a distance function that can deal with vectors having missing component values next.

- *Combination of categorical and numerical features:* Let the vectors be l-dimensional out of which let m be categorical and the remaining $l-m$ be numerical. Without loss of generality, let $x_1, x_2, \cdots, x_m$ be m categorical features and $x_{m+1}, \cdots, x_l$ be numerical features. There are some classifiers that can deal with vectors that have a mix of categorical and numerical components.

Also, a more recent view has been to represent categorical attribute values using one-hot representation that is popularly used in neural networks. If the number of distinct values of a variable is p, then each value is represented as a binary vector with one 1 and remaining $p-1$ values being zero (0). The ith distinct value is represented as a binary vector with a 1 in the ith location and remaining zeros.

However, both distance based and dot product based classifiers will face difficulties in dealing with such mixed data. In the case of proximity based classifiers, it is not possible to use euclidean distance type distances or cosine like similarity measures. Some of the possible distances are:

1. *Gower Distance:* It is defined between two vectors X^i and X^j as

$$d(X^i, X^j) = \sum_{p=1}^{m} mat(x_p^i, x_p^j) + \sum_{q=m+1}^{l} \frac{|x_q^i - x_q^j|}{range(q)},$$

where

- $mat(x_p^i, x_p^j) = 0$ if $x_p^i = x_p^j$ else it is 1.
- range(q) is the range of the domain of the numerical feature.

Another normalizing variant involves dividing the sum by the dimensionality l.

We can illustrate this using the data given in Table 1.3. Note that the Gower distance between X^i and X^j is

$$d(X^i, X^j) = 1 + 1 + 0 + \frac{1}{2} + \frac{100}{1000} = 2.6.$$

Its normalized variant is $\frac{2.6}{5} = 0.52$.

2. *Heterogeneous Value Distance Metric* ($HVDM$)*:* It can deal with missing values also. However, it is applicable in classification contexts only. It is defined as

$$HVDM(X^i, X^j) = \sqrt{\sum_{p=1}^{l} d^2(x_p^i, x_p^j)},$$

Table 1.3 Mixed type data used to illustrate gower distance

	Categorical 1	Categorical 2	Categorical 3	Numerical 1	Numerical 2
X^i	a	1	0	1.5	600
X^j	b	0	0	0.5	700
Range	NA	NA	NA	2	1000

where

$$d(x_p^i, x_p^j) = \begin{cases} 1 & if\ x_p^i,\ or\ x_p^j\ is\ unknown \\ nvdm(x_p^i, x_p^j) & if\ p\ is\ categorical \\ ndiff(x_p^i, x_p^j) & if\ p\ is\ numerical \end{cases}$$

where

- $nvdm(x_p^i, x_p^j) = \sqrt{\sum_{c=1}^{C}(P(c|x_p^i) - P(c|x_p^j))^2}$ if there are C classes and $P(c|x_p^i)$ is the posterior probability of class c given that x_p^i is the value of the pth component of X^i, and
- $ndiff(x_p^i, x_p^j) = \frac{|x_p^i - x_p^j|}{4\sigma_p}$, where σ_p is the standard deviation of the values of the pth component.

So, Gower distance can deal with the vectors even if they have both categorical and numerical components. Further, $HVDM$ can deal with vectors that have missing values also; however, it requires computation of posterior probabilities for each of the categorical variables.

- *Varying domain sizes:* In applications where different components of the vectors have different domain sizes, it is possible for some components to dominate in contributing to the distance between any pair of patterns. This can be handled by scaling different components differently and such a process of scaling is called *normalization*. We have seen such a scaling in the computation of both the Gower distance and $HVDM$. There are typically two types of popular normalization schemes:

  1. *Scaling using the range:* On any categorical data type, the values of two patterns either match or mismatch, respectively the distance contribution is either zero (0) or 1. Note that both the Gower distance and $HVDM$ use this. So, the contribution of any categorical variable is an element of the set $\{0, 1\}$. To be consistent, in the case of any numerical feature we want the contribution to be in the range [0,1]. This is achieved by scaling the difference by the range of the variable. So, if pth component is of numerical type, then its contribution to the distance between X^i and X^j is

  $$\frac{|x_p^i - x_p^j|}{R_p},$$

  where R_p is the range of the pth component. Note that the minimum value of this term is 0 (zero) and the maximum value is 1 when $|x_p^i - x_p^j| = R_p$. Such a scaling will ensure the required behaviour and the contribution, to the distance, of a categorical or a numerical component will be in the range [0, 1].
  2. *Standardization:* Here, each numerical variable value is scaled so that the mean value of the variable is 0 (zero) and the standard deviation is 1. This

is motivated by the standard normal distribution that has zero mean and unit variance.

This may be illustrated using the following example.

- Let there be 5 l-dimensional data vectors and without loss of generality, let the lth components of the 5 vectors be 600, 800, 200, 1000, and 400.
- The mean of this collection is

$$\frac{600 + 800 + 200 + 1000 + 400}{5} = 600.$$

- We get zero mean data by subtracting this mean from each of the 5 data items to get 0, 200, -400, 400, -200. Note that this is a zero mean data.
- To make the standard deviation of this data 1, we divide each of the zero mean data values by the standard deviation of the data.
- Note that the variance of the zero mean data is

$$\frac{0 + 200^2 + (-400)^2 + 400^2 + (-200)^2}{5} = 80{,}000$$

  and the standard deviation is 282.84.
- So, the scaled data is 0, 0.707, -1.414, 1.414, -0.707. Note that this data has zero mean and unit variance.

This is illustrated further using the following 4 two-dimensional vectors. Let

$$X_1 = (1, 100{,}000), X_2 = (2, 100{,}000), X_3 = (1, 200{,}000), X_4 = (2, 2{,}000{,}000)$$

be four vectors with the first feature taking values in $[1, 2]$ and the second feature assuming values from [100,000, 200,000]. The mean of these four vectors is (1.5, 150,000). By subtracting the sample mean vector from the four patterns gives us four mean normalized data points which are

$$(-0.5, -50, 000), (0.5, -50, 000), (-0.5, 50{,}000), (0.5, 50{,}000).$$

Now we can make it unit variance data by dividing each value by the sample standard deviation value. Note that the standard deviation for the first feature is $\frac{1}{2}$ and for the second feature, it is 50,000. By dividing the values by their respective standard deviations, we get the vectors

$$(-1, -1), (1, -1), (-1, 1), (1, 1).$$

Note that these four points lie on the surface of a unit circle centered around the origin. In a higher dimensional space, the normalized patterns fall on the surface of a unit hypersphere.

- *Presence of outliers:* In the previous section, we have examined outliers and the reasons for their presence. We will see how the data can be processed to detect and eliminate outliers. Some possibilities are:
  - *Out-of-range outliers:* Some ways of dealing with this kind of outliers are:

    Based on statistics: If the data is normally distributed, an outlier can be characterized based on how far away it is from the mean in terms of the variance/covariance structure. For example, one can fit a normal distribution with mean μ and variance σ^2 to one-dimensional data. It is well-known that in a uni-variate normal distribution, the values in the range $[\mu - 3\sigma, \mu + 3\sigma]$ account for more than 99%. So, any object/value falling outside this range may qualify to be an outlier as it is away from the mean. One may collect all such data points and look for possible outliers and deal with them.
    Knowledge-based approach: knowledge-based If the domain of a variable is known beforehand, then it is easy to identify out-of-range data points and eliminate or correct such entries. For example, removing an extra zero (0) in some numeric entry or using a spell correction software package belong to this category.
  - *within-range outliers:* Typically these outliers can be detected based on the sparseness of the region to which the point belongs. Some possible schemes are:
    - *Clustering based:* One can use a clustering algorithm to cluster the data and identify clusters that have a small number of data items. One can concentrate on such small size clusters and examine for possible outliers. There are clustering algorithms that can generate clusters by taking into account the density of data.
    - *Frequency based:* There are applications like *information retrieval* where it is observed that frequent words are not helpful in classification. So, such frequent words are eliminated. Further, *rare terms* that are infrequent are not important to several users of the search engines or information retrieval systems. Such rare words could be viewed as some kind of outliers as they could be of interest to a very tiny fraction of users.

An important step in ML is to represent the data items as vectors appropriately. While using such vectors, one will encounter several difficulties.

- *Problems with high-dimensional vectors:* Matching based algorithms like NNC *fail in dealing with high-dimensional vectors*. This is because as the dimensionality increases the notion of similarity gets affected. If X is a pattern, then its nearest neighbor $NN(X)$ and its farthest neighbor $FN(X)$ will be located such that as the dimensionality $\rightarrow \infty$, $d(X, NN(X)) \rightarrow d(X, FN(X))$. That is $NN(X)$ and $FN(X)$ lie at the same distance from X corrupting the notion of NN in high dimensional spaces. So, representing the vectors in a lower-dimensional

space is important for the matching based algorithms to work properly. Some of the solutions to this problem are:

1. *Approximate matching:* An approximate matching algorithm based on random subspace matching is used to achieve the approximation. It is based on hashing that is sensitive to nearer or local regions. This is studied under the paradigm of locality sensitive hashing (LSH). We study LSH in detail in a subsequent chapter.

   There are other random subspace classifiers based on building one classifier for each random subset of features. Such multiple classifiers are combined to form the overall composite classifier.
2. *Reducing the dimensionality:* Another approach is based on reducing the dimenisonality and then building the ML model on the reduced dimensional patterns. Some popular schemes under this category are:

   a. *Feature Selection:* Here, we select a subset of l features from the given $L(> l)$ features. This subset selection is carried out in a variety of ways. A popular scheme, in the case of categorical features, is based on *mutual information*. We will discuss mutual information based feature selection in a subsequent chapter.
   b. *Linear Feature Extraction:* In this type of schemes, we extract l out of $L(> l)$ features where each of the extracted l features is a linear combination of the L features. There are different schemes under this category. Some popular ones are:

      i. *Principal Components* (PCs)*:* These are the leading eigenvector directions of the covariance matrix of the data. These PCs, or the leading eigenvectors, capture the maximum variance directions in the data. The first PC is in the maximum variance direction, the next PC captures the second largest variance direction, and so on. Also these PCs are uncorrelated pairwise. We will analyse the behaviour of PCs in classification in a later chapter.
      ii. *Non-negative matrix factorization* (NMF)*:* Here, the data (n items) in L dimensional space is represented as a matrix of size $n \times L$; it is called the *data matrix*. Let this matrix be A. It is factorized into a product of two matrices B and C such that

      $$A_{n \times L} \approx B_{n \times l} C_{l \times L}.$$

      Typically, the entries in A are non-negative. The factorization is carried out so that the resulting approximate factors B and C also have non-negative entries. Each of the l columns of the matrix B characterizes a feature extracted. We will consider it in detail in a later chapter.

   c. *Non-Linear Feature Extraction:* In this category, from the given collection of L features, a smaller set of l features is extracted where each of the l extracted features is a non-linear combination of the original L features.

Such non-linear feature extractors are easily achieved with the help of neural networks. Autoencoder is a popular architecture for realizing non-linear feature extraction. We will examine autoencoders in detail later.

- *Visualization:* It is important to verify whether the patterns represented in the low-dimensional $l(< L)$ space preserve some structural properties present in the original L-dimensional space. Visualization tools help us project the data, present either in the L or the l dimensional space, to a two-dimensional space so that we can analyse the projected data manually. There are schemes including the t-distributed stochastic neighbor embedding (t-SNE) plot which is a nonlinear projection based statistical scheme.

1.2.3 Model Selection

Feature engineering is an important step in ML. Once the data is represented in an appropriate manner, then one can select the model appropriately. For example, if there are both numerical and categorical entities, then it is possible to use classifiers based on decision trees, and Bayes classifier; we will not be able to exploit support vector machine and neural network based models directly on such vectors. So, we need to select a model based on the application considered and the training data types involved.

1.2.4 Model Estimation

Once a model for ML is selected, one needs to estimate the parameters associated with the model using the *training data*. For example, a linear discriminant function based classifier can be characterized using W^tX where W and X are l-dimensional vectors. Learning the weight vector W is done using the training data. Given any test pattern X, X is assigned to class1 if $W^tX < 0$, else X is assigned to class2. Similarly, in the case of Bayes classifier one needs to estimate the probability structure, the prior probabilities and likelihood values, underlying the learning problem.

1.2.5 Model Validation

It is important to tune the parameters of the estimated model using additional data called *validation data*. This step is called model validation. The validation data is used to fine tune the model estimation. Typically the available training data is split into train and validation parts in practice. One of the most popular schemes is K-

fold cross-validation. Here, the training data set is partitioned into K equal size sets. Out of these K, $K-1$ sets are used for training and the remaining one is used for validation. This process is done so that every one of the K subsets participates in validation. Once the parameters are fine tuned, the ML model is ready for use.

1.2.6 Model Explanation

In several practical applications, an expert would typically be involved in providing the underlying domain knowledge, training data, etc. Further, the learnt ML model is to be used by the domain expert, for example in domains like medicine, security, transport, and education. So, it would be possible for the expert to provide an appropriate *feedback* to improve the model if the learning process is *transparent* and easy for the expert to appreciate. In order to meet this requirement, it is essential to provide an explanation that could be easily understood by the expert. For example, decision tree based classifiers are the simplest and are easy for providing an explanation to the expert. ML models based on neural networks fall at the other end of the spectrum as they are typically opaque.

1.3 Data Sets Used

We make use of two data sets to conduct experiments and present results in various chapters of the book. These are:

1. *MNIST Handwritten digits data set:* There are 10 classes (corresponding to digits 0, 1, ..., 9) and each digit is viewed as an image of size 28×28 (= 784) pixels; each pixel having values in the range 0–255. There are around 6000 digits as training patterns and around 1000 test patterns in each class and the class label is also provided for each of the digits. Visit http://yann.lecun.com/exdb/mnist/ for more details.
2. *The ORL Face Data Set:* Ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). Each image is of size 112×92. Visit https://ai.stanford.edu/~marinka/nimfa/nimfa.examples.orl_images.html for more details.

1.4 Summary

Machine learning is an important topic and has affected research practices in both science and engineering significantly. Important steps in building an ML system are:

- Data acquisition that is domain application dependent.
- Feature Engineering that involves both data preprocessing and representation.
- Selecting a model based on the type of data and the knowledge of the domain.
- Learning the model based on the training data.
- Evaluating the learnt model based on validation data.

This book deals with **representation** that is the most important part of ML.

References

1. Murphy, K.P.: Machine Learning - A Probabilistic Perspective. MIT Press (2012)
2. Murty, M.N., Biswas, A.: Centrality and Diversity in Search: Roles in A.I., Machine Learning, Social Networks, and Pattern Recognition. Springer Briefs in Intelligent Systems (2019)
3. Aggarwal, M., Murty, M.N.: Machine Learning in Social Networks - Embedding Nodes, Edges, Communities, and Graphs. Springer (2021)
4. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press (2005)
5. Murty M.N., Devi V.S.: Introduction to Pattern Recognition and Machine Learning. World-Scientific (2020)
6. Kabán, A.: Fractional norm regularization: learning with very few relevant features. IEEE Trans. Neural Netw. Learn. Syst. **24**(6), 953–963 (2013)
7. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM **51**(1), 117–122 (2008)

Chapter 2
Representation

Abstract Representation is an important step in building ML models. This chapter introduces how data items, classes and clusters are represented. It also discusses the importance of representation in both ML and AI. In the process it deals with both feature selection and feature extraction and introduces different categories of dimensionality reduction.

2.1 Introduction

In the previous chapter, we provided a brief introduction to machine learning. We have identified several steps involved in building an ML system. Specifically we have highlighted the importance of *feature engineering* in ML model building. Feature engineering has two important components:

1. *Pre-processing the data:* This step involves dealing with missing data, combination of categorical and numerical attributes, normalizing the data appropriately, etc.
2. *Representation:* This step is very important in ML. We can illustrate its importance by using the data in Table 2.1. In this table, there are two independent features, *height of the tallest hillock* and *average height of people* in a village and the dependent feature is the crop yield in the village. Note that if a test village is reported to have average height of its people to be 4 feet 8 inches, then an ML model can help us make a prediction of the crop yield. However, such a prediction of the algorithm may not be meaningful as we know very well that average height of people in a village is **not good** in helping us predict the crop yield in the village.

   So, representation is an important component of the ML system. An arbitrary representation also might be adequate to build an ML model. However, the predictions made using such a model may not be meaningful. This book deals primarily with **representation**.

M. N. Murty, M. Avinash, *Representation in Machine Learning*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-981-19-7908-8_2

Table 2.1 Representation is important

Height of the tallest hillock	Average height of people	Crop yield
2 m	5 feet	Low
5 cm	5 feet 2 inches	High
1 m	6 feet	Low
10 m	4 feet 5 inches	High
7.5 m	5 feet 1 inch	Low

2.2 Representation in Problem Solving

Representation is important not only in *problem solving in ML*, it is also important in *artificial intelligence* (*AI*). in order to illustrate the importance of representation in *AI*, we consider the following example from automated deduction. Consider the following statements.

- Gold ornaments are expensive.
- Necklace is a gold ornament.

From these two statements, we would like to infer that *necklace is expensive.* For such a reasoning we would use first-order predicate calculus (*FOPC*). In *FOPC*, the given statements are represented as:

1. $\forall X\ (gold - ornament(X) \rightarrow expensive(X))$.
2. $gold - ornament(necklace)$

There is an inference rule called *modus ponens* (*MP* associated with *FOPC*. It takes statements of the form $\forall X(P(X) \rightarrow Q(X))$ and $P(a)$ to infer $Q(a)$, where P and Q are predicates, X is a variable and a is a constant. By associating the following way:

- The predicate *gold-ornament* with P,
- The predicate *expensive* with Q,
- The constant $necklace$ with a,

and using *MP* we infer $expensive(necklace)$. This inference is simple because of reasoning based on *FOPC*. The corresponding representation is very compact. It is possible to represent these statements in propositional logic (*PL*) also. Assuming that the domain $\mathcal{D}$ has some n objects, that is $\mathcal{D} = \{a_1, a_2, \cdots, a_n\}$, we can represent the above statements in *PL* as follows:

1. $gold - ornament(a_1) \rightarrow expensive(a_1) \wedge gold - ornament(a_2) \rightarrow \wedge expensive(a_2) \cdots gold - ornament(a_n) \rightarrow expensive(a_n)$.
2. $gold - ornament(necklace)$

Now using *MP*, we can infer $expensive(necklace)$, where $necklace$ is one of the objects in the domain. We can get the same inference using *PL* also. However, representation using *PL* can be highly unwieldy. The length of the conjunction can

keep increasing with the size of the domain. We can also have a situation where inference itself can be affected by the representation. For example, consider the following statements:

- Humans are widely distributed over the earth.
- Vidya is human.

We can represent these two statements in $FOPC$ as follows:

1. $\forall X(Human(X) \rightarrow widely-distributed-over-theearth(X))$.
2. $human(Vidya)$

From these two statements and MP we infer $widely-distributed-over-the-earth(Vidya)$ which is not correct. So, here representation of the statements makes the inference erroneous.

In AI, representation can impact both the number of steps required in solving a problem and the complexity in terms of storage/space requirement. Also the representation can impact the inference. In ML also we encounter similar difficulties: representation has impact on the performance of the model and space and time requirements.

2.3 Representation of Data Items

Even though there are other paradigms for ML, the most active and currently popular paradigm is *statistical machine learning*. Here, each data item is represented as a *vector*. Typically we consider addition of vectors, multiplication of a vector by a scalar, and the dot product between a pair of vectors as important operations on the set of vectors. This *vector space model* is the most exploited model over the past three decades. It is popular in ML, *information retrieval* (IR), and *soft computing* (SC). In most of the practical applications, the size of the vectors, L, can be very large. For example, some practical natural language systems have been trained on a vocabulary of 30,000 or more words. Google Ngrams are around 500 billion. In this case, the dimensionality of the vectors is the vocabulary size or the number of Ngrams; so, the dimensionality could be very large. Such high dimensional data are common in bioinformatics, information retrieval, satellite imagery, etc.

Some of the difficulties associated with high-dimensional vectors are:

1. *Space requirement:* Consider a collection of n data vectors in the L-dimensional space. This data can be viewed as a matrix of size $n \times L$. In order to store such a data matrix, we require to $n \times L$ entries. So, naturally space required to store the data matrix increases linearly with L.
2. *Time requirement:* Recall that proximity between a pair of vectors is calculated using a distance measure like the euclidean distance or some similarity measure like the cosine of the angle between the vectors.

(a) In order to compute the euclidean distance between 2 L-dimensional vectors, X^i and X^j, one needs to compute

$$\sqrt{\sum_{p=1}^{L}(x_p^i - x_p^j)^2}.$$

This operation requires L multiplications, L subtractions and $L-1$ additions and a square root operation. So, as L increases time required to compute distances will also increase.

(b) In the case of cosine similarity, we need to compute

$$\frac{X^{iT}X^j}{||X^i||||X^j||}.$$

Note that this operation requires L multiplications for the numerator, 2 L multiplications in the denominator and around the same number of additions. So, this computation also requires more time as L is increased. The time here also is linear in L.

So, computation time increases with the dimensionality.

3. *Performance of the model:* It is well-known that as the dimensionality L increases, we require a larger training data set to build an ML model. There is a result, popularly called under the name *peaking phenomenon*, that shows that as the dimensionality L keeps increasing the accuracy of a classification model increases until some value and beyond that value the accuracy keeps decreasing. This may be attributed to the well-known concept called overfitting. The model will tend to remember the training data and fails to work on validation data. With a larger training data set we can afford to have a larger value of L and still avoid overfitting.

Even though the dimensionality of the data set in an application is large, it is possible that the number of training vectors is small. It could be because acquiring training data is expensive like in crowd sourcing or the nature of the application does not permit. In such cases a popular technique in ML is to reduce the dimensionality so that the learnt model does not overfit the available data. However in the case of big data, overfitting problem is not an issue; this is aptly exploited by the deep learning (DL) models. Deep learning is more appropriate to exploit a larger data set.

In the case of dimensionality reduction based approach the vectors are represented using one of the following schemes:

a. *Feature selection:* Let $\mathcal{F} = \{f_1, f_2, \cdots, f_L\}$ be the set of L features. In the feature selection approaches, we would like to select a subset F_l of $\mathcal{F}$ having $l(< L)$ features such that F_l maximizes the performance of the ML model. There are three different approaches to feature selection. They are:

i. *Filter Methods:* In filter methods, we rank L features using a fitness measure that exploits the class labels but does not use any classifier. Based on the ranking, the top l features are selected and used in building the ML model. There are different fitness measures to evaluate and rank the features. They are based on:

   A. *Distance:* We prefer values of a feature from the same class to be such that the intra-class distances are smaller and inter-class distances are larger.
   B. *Dependency:* Here, the correlation between a feature and a class is exploited in ranking. If the correlation is larger for a class and smaller for the other classes, then the feature in ranked better.
   C. *Mutual information* (MI:) If a feature and a class have a larger MI value, then the feature is good. It is seen that MI based feature selection works well on high-dimensional datasets. Feature selection scheme based on MI will be considered later in the book.

ii. *Wrapper Methods:* In this case, the selection of a subset of features is based on the performance of a classifier on the selected subset. The subset of features which gives the maximum classification performance is selected. This subset of features is used in designing the ML model. There are a variety of approaches including *genetic algorithm* based approaches that employ this method. These schemes can be more expensive compared to the filter based schemes because the number of subsets is much larger than the number of features.

iii. *Embedded Methods:* Here, an ML model is built using the L features and the model directly selects/indicates the relevant subset of features. Several classifiers including ones based on decision trees, support vector machines, and Bayes classifier can be exploited to realize such an embedded scheme.

b. *Feature extraction:* Here from the set $\mathcal{F}$ of L features, a set $\mathcal{G} = \{g_1, g_2, \cdots, g_l\}$ of l features is extracted. It is possible to categorize these schemes into

   i. *Linear schemes:* In this case,

$$g_j = \sum_{i=1}^{L} \alpha_{ij} f_i.$$

   That is each element of $\mathcal{G}$ is a linear combination of the original features. Note that feature selection is a specialization of feature extraction. Some prominent schemes under this category are:

      A. *Principal Components* (PCs): Consider the dataset of n vectors in L-dimensional space; this may be represented as a matrix A of size $n \times L$. The mean of the rows in A is subtracted from each row of A to get zero-mean data. Without loss of generality, we assume that A

is zero-mean data matrix. Now the covariance matrix of the data is given by

$$\sigma_{L\times L} = A^t_{L\times n} A_{n\times L}.$$

Because σ is the *covariance matrix*, it is a *symmetric* matrix and its *eigenvalues capture variances* in the data in the directions of their respective eigenvectors. Let $\lambda_1 \geq \lambda_2 \geq \cdots, \geq \lambda_L$ be the eigenvalues of σ. Let $V_1, V_2, \cdots, V_L$ be the respective eigenvectors of σ. Then V_1 gives us the direction of the first PC, V_2 the direction of the second PC, and so on till V_l the direction of the lth PC. If $\lambda_i \neq \lambda_j$, then V_i is orthogonal to V_j due to the symmetry property of σ. Because of these properties it is possible to show that the projections of the data onto these eigenvector directions give us data directions that are *uncorrelated*.

The representation of the data points using these eigenvector (PC) directions is popular in ML.

B. *Non-negative matrix factorization* (NMF): Even when the data is non-negative, it is possible that PCs have negative entries. However, it is useful to have representations using non-negative entries; NMF is such a factorization of $A_{n\times L}$ into a product of $B_{n\times l}$ and $C_{l\times L}$. Its use is motivated by the hope that NMF can be used to extract objects in an image represented by A.

In NMF, A is factorized into B and C such that the Frobenius norm between A and BC is minimized. That is

$$||A - BC||_F$$

is minimized. Here, A is known, further if we know either B or C, we can get the other factor in an easy manner iteratively. Now considering $B_{n\times l}$, we can see the connection with the original/given n vectors in L-dimensional space. B consists of, as its rows, the same n vectors in a lower dimensional, ($l < L$), space. The matrix $C_{l\times L}$ describes how each of the l features is a weighted combination of the original features, the weights being provided by the rows of C. For example, the weights for the ith feature, ($1 \leq i \leq l$), is given by the ith row of C.

We will examine, in detail, both these schemes in a later chapter.

c. *Nonlinear feature extraction:* Here, we represent using $\mathcal{H} = \{h_1, \cdots, h_l\}$, such that

$$h_i = t(f_1, f_2, \cdots, f_L),$$

where t is a nonlinear function of the features. For example, if $\mathcal{F} = \{f_1, f_2\}$, then $h_1 = af_1 + bf_2 + cf_1f_2$ is one such nonlinear combination; it is nonlinear because we have term of the form f_1f_2 in h_1. These may be categorized as

i. *Explicit schemes:* There are a variety of schemes under this category. The basic idea is to represent the data in a low-dimensional space explicitly.

   A. One popular scheme is based on:

      - Let X and Y be a pair of patterns in the L-dimensional space.
      - Let X^* and Y^* be the corresponding l-dimensional patterns.
      - We would like to get X^* and Y^* such that

$$d(X, Y) \approx d(X^*, Y^*)$$

      - There are several ways of ensuring that $d(X, Y) \approx d(X^*, Y^*)$. A popular way is to pose the problem as a minimization of a squared difference in an iterative manner.

   B. *Autoencoders:* Here, a neural network is used which has an encoder and a decoder. The middle layer may be viewed as a bottleneck layer that has l neurons so that the l outputs from the middle layer give an $l(< L)$-dimensional representation of the L-dimensional pattern that is input to the autoencoder. Note that the encoder encodes or represents the L- dimensional pattern in the l-dimensional space and the decoder decodes or converts the l-dimensional pattern into the L- dimensional space. Note that it is called *autoencoder* because with the input and output layers the same L-dimensional pattern is associated. Here also the weights in the network are updated so that some distance between the pattern presented at the input layer and the pattern obtained at the output layer is minimized. In an ideal learning situation, we would like to have a pattern input at the input layer is decoded at the output layer exactly.

ii. *Implicit schemes:* We use some function that implicitly computes similarity that considers nonlinear terms. These are applicable even when the data is low-dimensional. Some popular schemes are:

   A. *Use a kernel function:* There are a variety of kernel functions that capture the similarity between a pair of patterns. Consider, for example, a pair of patterns X and Y such that the similarity

$$K(X, Y) = (1 + X^tY)^2.$$

   Here, it is possible to view the kernel function performing a dot product based on a nonlinear representation of patterns.

For example, let $X = (x_1, x_2)^t$ and $Y = (y_1, y_2)^t$ be two two-dimensional patterns. Then $K(X, Y) = (1 + x_1y_1 + x_2y_2)^2$ which could be seen as a dot product between vectors that have nonlinear components of the form $X' = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$ and $Y' = (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, \sqrt{2}y_1y_2)$.

B. *Use a nonlinear classifier:* A nonlinear classifier that perceives a nonlinear combination of the components of the vectors is a member of this category. For example, logistic regression employs a nonlinear function of the linear discriminant $W^tX + b$, where W and b are learnt based on the training data. The specific nonlinear form is

$$h(W^tX + b) = \frac{1}{1 + exp(-(W^tX + b))}.$$

d. *Other schemes:* There are other feature extraction schemes

i. *Schemes for visualization:* Several times it is important for a human to visualize the data in a two or three dimensional space. There are different schemes for doing this. One of them is a neural network based on Kohonen's self-organizing Map. It maps data in an L-dimensional space into a two or three dimensional space while ensuring that topological structure in the data is preserved in the reduced dimensional space.

ii. *Fisher's discriminant:* Another popular scheme is based on reducing the dimensionality of vectors to a one-dimensional space. In a two-class problem, a vector X in an L-dimensional space is projected to a one-dimensional space by using W^tX, where W is learnt using the labelled vectors. It employs a vector W whose direction is given by

$$direction - of(W) = Max_W \frac{W^tS_BW}{W^tS_WW},$$

where S_B and S_W are matrices of size $L \times L$ and are given by

- m_1 is the mean of the vectors in class C_1 and m_2 is the mean of the vectors in class C_2
- $S_B = (m_1 - m_2)(m_1 - m_2)^t$ (Between class scatter matrix) and
- $S_W = \sum_{X_i \in C_1}(X_i - m_1)(X_i - m_1)^t + \sum_{X_i \in C_2}(X_i - m_2)(X_i - m_2)^t$ (within-class scatter matrix)

It is possible to show that required direction of W is characterized by

$$S_BW = \lambda S_WW \Rightarrow direction - of(W) = S_W^{-1}(m_1 - m_2),$$

because $S_BW = \alpha(m_1 - m_2)$, for some constant α. Fisher's discriminant performs a linear transformation.

2.4 Representation of Classes

There are a variety of classifiers and each might have its own abstraction of the classes. They may be categorized based on using:

1. *Exemplars:* Here there is no need to represent classes. All the training patterns or prototypes obtained from the training set are directly used to classify the test patterns. Some popular classifiers of this variety are:

   a. *Nearest neighbor classier* (NNC): Here a test pattern X is assigned the class label of its nearest neighbor, $NN(X)$. So, for each test pattern, one has to compute n distances if there are n training patterns. So, it could be computationally expensive and there are different schemes that are used to reduce the number of training patterns.
   b. *k-Nearest neighbor classifier* ($KNNC$): Depending on a single nearest neighbor may not be robust if there are outliers present in the data. If $NN(X)$ is an outlier, then X can be erroneously labeled. In order to correct it, in $KNNC$, $k(> 1)$ neighbors of X are considered and X is given the label of *majority of these k neighbors*. This is more robust and in an asymptotic sense leads to an optimal classifier. In $KNNC$, the value of the parameter k needs to be found based on the validation data.

      There are variants to $KNNC$. Specifically, $k = 1$ leads to NNC. When $k > 1$, it is possible to think of giving different weights to different neighbors of X. This is done to ensure that neighbors in the close vicinity of X contribute more to the class label of X.

   Some of the problems associated with the $KNNC$ based classifiers are:

   a. Classification of each test pattern requires $O(Kn)$ time and space.
   b. If the dimensionality L is large, then the $KNNC$ may fail to classify the test pattern correctly. This is because neighborhood may not be a meaningful notion in high-dimensional spaces. There is a solution to this problem based on locality sensitive hashing (LSH) that can be useful in obtaining an approximate NN's of X in a high-dimensional space. We discuss LSH in detail in the next chapter.

2. *Probability structure:* Bayes classifier, an optimal classifier, classifies a test pattern using the posterior probability. If there are classes C_1 and C_2, and if $P(C_1|X) > P(C_2|X)$, then X is assigned to C_1, else to C_2. It is possible to classify, based on the largest posterior, even when there are more than two classes. Bayes classifier in this setting can be shown to minimize the probability of error in classification. It is well-known based on Bayes rule that

$$P(C|X) \propto P((x_1, x_2, \cdots, x_L)|C)$$

   when $X = (x_1, x_2, \cdots, x_L)$ is an L-dimensional vector and C is a class; $P((x_1, x_2, \cdots, x_L)|C)$ is the likelihood of the vector X given that the class is C.

When the data is not large, it is difficult to make estimates of the probabilities involved. In order to simplify the estimates, one popular assumption is the *class-conditional independence*. Under this assumption, given the class all the components are assumed to be independent. This may be explained as follows:

- Let the test pattern X be such that $(x_1 = v_1, x_2 = v_2, \cdots, x_L = v_L)$.
- The Bayes classifier needs to compute the likelihood given by $P((x_1 = v_1, x_2 = v_2, \cdots, x_L = v_L)|C)$.
- Under the class-conditional independence we need to compute $P(x_1 = v_1|C)P(x_2 = v_2|C)\cdots P(x_L = v_L|C)$. This is a product of probabilities of the form $P(x_i = v_i|C)$ and it is possible to estimate these quantities even when the data set is not large.
- Bayes classifier under the assumption of class-conditional independence is called the *naïve bayes classifier* (NBC).
- So, in the Bayes classifier a class is represented by the class-conditional density of the data vectors, that is $P(X|C)$; in the NBC it is based on the probabilities $P(x_i = v_i|C)$ (of the individual components).

3. *Decision trees:* In the decision tree classifier (DTC), the training data is represented using a decision tree. In a decision tree all the leaf/terminal nodes represent *classes* and the non-terminal/internal nodes of the tree are called *decision nodes*. They are typically of the form $x_i < v_i$ where the left branch under the node might characterize all the data vectors with $x_i < v_i$ and the right branch corresponds to those data vectors that have $x_i \geq v_i$. So a class C is represented by all the paths that exist, in the decision tree, from the root to the leaves that belong to class C. The decision tree classifier (DTC) is simple and is popular because of its transparent nature in providing links between the feature values and classes.
4. *Linear discriminants:* Here, from the two-class training data, a vector W and a scalar b are learnt such that for all $X \in C_1$ $W^tX_1 + b < 0$ and for all the patterns in class C_2, $W^tX + b > 0$. So, the entire space of patterns is split into two regions based on whether $W^tX + b$ is less than zero (0) or not. So, a class is represented as a region characterized by the polarity of $W^tX + b$.

2.5 Representation of Clusters

Different clustering algorithms are grouped into *partitional* and *hierarchical* categories.

1. *Partitional algorithms:* Given a dataset $\mathcal{X} = \{X^1, X^2, \cdots, X^n\}$ of n vectors, the problem is to partition $\mathcal{X}$ into K clusters, $C_1, C_2, \cdots C_K$. Such a partition satisfies the property that

$$C_i \neq \phi;\ C_i \cap C_j = \phi;\ \textit{and}\ \cup_{i=1}^{K} C_i = \mathcal{X}.$$

If we relax the condition, $C_i \cap C_j = \phi$, we get a *soft partition*. Some of the popular partitional categories are:

- *K-Prototype algorithms:* The most popular algorithm is the K-means algorithm that starts with K initial centroids, one for each of the K clusters and updates the centroids iteratively based on assigning patterns nearest to each centroid to the respective cluster. This algorithm is popular because it requires linear time and space. In this case each cluster is represented by its centroid. Centroid need not be a member of the cluster; medoid is the most centrally located pattern that is a member of the cluster. If the data is categorical, one can have K-modes or K-medians representing the K clusters.
- *Spectral clustering:* In spectral clustering, the data points are represented as nodes of a graph and an edge is added between a pair of nodes if the corresponding data vectors are similar. Adjacency matrix of such a graph or some normalized versions of the adjacency matrix are used to realize the clustering based on the eigenvalues and eigenvectors of the matrix considered. The name spectral comes out of the view that the collection of eigenvalues is the *spectrum*. Here, the eigenvectors form the basis for the resulting cluster structure and so clusters are captured by the eigenvectors corresponding to the K- smallest eigenvalues.

K-means algorithm is the most popular clustering algorithm to generate hard partitions. There are soft versions that generate soft partitions ($C_i \cap C_j \phi$). Among them are the expectation maximization (EM) that may be viewed as a probabilistic variant of the K-means algorithm, a fuzzy K-means algorithm that is based on membership values, a rough K-means algorithm that exploits non-discriminative properties of equivalence classes of patterns. Currently, EM and variants that exploit it are popular; one popular variant is the *latent Dirichlet allocation* (LDA). It is possible to view LDA as generating a matrix factorization, $A_{n\times L} \approx B_{n\times l} C_{l\times L}$, where each column of B is a cluster/topic and each row is a pattern vector in a lower ($l < L$) dimensional space. Because of the inherent probabilistic structure, the entries in B and C matrices will be probabilities unlike the conventional NMF.

2. *Hierarchical clustering:* In the partitional (sot or hard) clustering, a single partition of $\mathcal{X}$ is obtained. On the contrary, in hierarchical clustering, a hierarchy of partitions is obtained. There are two categories:

   a. *Agglomerative algorithms:* Here each element of $\mathcal{X}$ is considered as a cluster, there are n clusters, to start with. This is a partition of size n. These smaller sized clusters are merged to get bigger size clusters iteratively. There are different schemes for merging the smaller sized clusters. Typically partitions of different sizes, starting with n, $n-1$, $n-2$, so on till all the patterns are in 1 cluster. So, a hierarchy of partitions of size n to size 1 are obtained. A specific partition of K clusters is chosen based on a predefined criterion. So, here clusters are subtrees below the specified level of the hierarchy.

b. *Divisive algorithms:* In this case, we start with all the elements in a single cluster and it is split into two clusters based on some criterion. To partition further, one of these two clusters is chosen and split further into two clusters. This splitting/division process goes on till there is no scope to split the existing clusters further. This scheme also gives a hierarchy of partitions of size 1, 2, and so on till we have n clusters. Here also each cluster is a tree below the chosen level.

2.6 Summary

Representation is the most important step in building ML models. It is an important component of feature engineering. Representation deals with both feature selection where a subset of features is selected based on different schemes and feature extraction. In feature extraction both linear and non-linear schemes are considered. Further, both implicit and explicit extraction schemes are considered. It is also important to consider schemes for representing classes and clusters of patterns.

References

1. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. **31**(3), 264–323 (1999)
2. Murty, M.N., Susheela Devi, V.: Introduction to Pattern Recognition and Machine Learning. World Scientific/IISc Press (2015)
3. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. John Wiley and Sons (2000)
4. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press (2005)
5. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson (2015)

Chapter 3
Nearest Neighbor Algorithms

Abstract Most of the practical data sets are high-dimensional. A major difficulty with classifying such data is involved not only in terms of the computational demands but also in terms of classification performance. It is very obvious when the learning algorithms are dependent on distances. In this chapter, we present the difficulties and possible solutions to deal with such high-dimensional data classification.

3.1 Introduction

There are several classifiers and clustering algorithms that inherently depend upon some distance between a pair of patterns. These include

1. **Nearest Neighbour Classifier** (NNC)**:** Here a test pattern X is classified based on its nearest neighbour (NN) in the training data. Specifically, let

$$\mathcal{X} = \{(X_1, \theta_1), (X_2, \theta_2), \cdots, (X_n, \theta_n)\}$$

be the labelled training data set of n patterns. Let each pattern be a vector in some L dimensional space. Here, $X_i, i = 1, 2, \cdots, n$ is the ith training pattern and θ_i is its class label. So, if there are p classes with their labels coming from the set

$$\mathcal{L} = \{C_1, C_2, \cdots, C_p\},$$

then $\theta_i \in \mathcal{L}, \ \ for\ i = 1, 2, \cdots, n$. Now the nearest neighbor of X is given by

$$NN(X) = argmin_{X_j \in \mathcal{X}} d(X, X_j),$$

where X_j is the jth training pattern and $d(X, X_j)$ is the distance between X and X_j. Intuitively, $NN(X)$ is in the proximity of X; so $NN(X)$ is at a minimum distance from X and is maximally similar to X.

M. N. Murty, M. Avinash, *Representation in Machine Learning*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-981-19-7908-8_3

2. **K-Nearest Neighbour Classifier** ($KNNC$)**:** In $KNNC$, we find the K nearest neighbours (KNN's) of the test pattern X from the training data $\mathcal{X}$ and assign the majority class label among the KNN's to X. So, we need to compute distance between X and each of the n training patterns, in $\mathcal{X}$ in the L dimensional space, to find the KNN's of X. The class label of X is decided based on the majority class label of the KNN's. Here also we need to compute the distance between X and each of the training patterns.
3. **Clustering Algorithms:** There are clustering algorithms that depend upon distances. One of the most popular clustering algorithms is the K-Means algorithm (KMA). This algorithm is an iterative algorithm. It starts with a set of K initial centroids, one centroid per cluster; KMA computes the distance between a data vector and each of the K centroids and assigns the pattern to the cluster with the nearest centroid. Once all the patterns are assigned to their clusters, based on the nearest centroid for each pattern, the centroid of each cluster is recomputed based on the current cluster assignment. This process of assigning patterns and updating the centroids is repeated till there is no change in the centroids. So, KMA needs to compute, during each iteration, the distance between every pattern and each element of the set of K centroids to locate the nearest centroid.

So, all the above algorithms need to compute the distance between an L-dimensional vector X and each element of a set of L-dimensional vectors to find one or more nearest neighbors. Computation of $NN(X)$ involves the usage of some distance between a pair of L-dimensional patterns.

A major problem with the NN search in high-dimensional spaces (L is large) is that it may not be stable. This is related to the *concentration problem*. Several training patterns will concentrate around X in the L-dimensional space, when L is large. The consequence of this concentration problem is that the discrimination between even the $NN(X)$ and $FN(X)$ diminishes as the dimensionality increases, where $FN(X) \in \mathcal{X}$ is the farthest neighbour of X.

3.2 Nearest Neighbors in High-Dimensional Spaces

Let $d(X, NN(X))$ be the distance of the nearest neighbor, $NN(X)$, of a pattern X. The nearest neighbor search becomes *unstable*, as the dimensionality increases, because the distances from the pattern X to most of the training data points lie close to $d(X, NN(X))$. Further, it can be shown that, under certain acceptable conditions, the probability that the ratio of $d(X, NN(X))$ and $d(X, FN(X))$ will tend to 1 in higher dimensional spaces. A consequence of this property is that it is difficult to obtain the nearest neighbour of a point X from the other neighbours that may include even the farthest neighbour, of X.

Let us consider the MNIST Data set, where the nearest neighbor search is seen to perform well. The Training set of classes labeled 7 and 9 from the MNIST data set consists of 12,214 images, each of dimension $28 \times 28 (= 784)$. Viewing each image

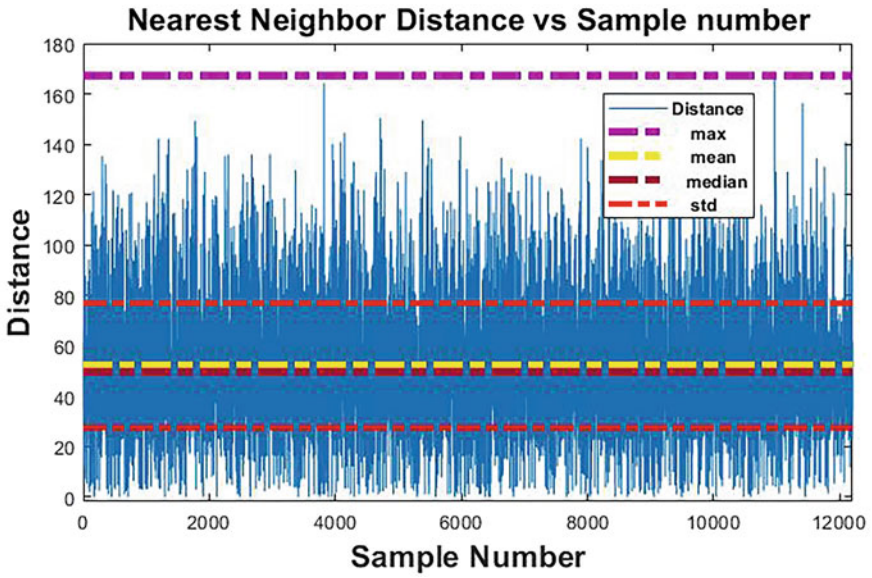

Fig. 3.1 Nearest neighbor distance for each of 12,214 samples in the classes labeled 7 *and* 9 of the MNIST data set

as a vector of size 784, the training data matrix becomes $12,214 \times 784$ (where, each row is an image). For every sample point (each image) X in the training set, we find the nearest neighbor $NN(X)$, from the rest of the training patterns and its distance $d(X, NN(X))$ from $NN(X)$. We show the plot of these distances for all the 12,214 patterns in $\mathcal{X}$ in Fig. 3.1.

The Histogram of the 12,214 distances is shown in Fig. 3.2. From the Histogram, we can observe some outliers (X's) in the data set, whose $d(X, NN(X))$ value is falling beyond 3σ (three standard deviations) from its mean. There are 66 such outliers. It is possible to use some other technique to detect and remove outliers; some of the other outlier detection techniques are discussed in the previous chapters. After removing these 66 outliers, we get a cleaned training set of size 12,148. We consider the corresponding data matrix of size $12,148 \times 784$.

For a pattern X from the training set, the number of training patterns which fall within the distance bound of 1.1 times $d(X, NN(X))$ is calculated (it is $1+\epsilon$ bound where $\epsilon = 0.1$) and its relative percentage (number of points within the distance bound with respect to the total number of 12,147 patterns excluding X) is plotted in Fig. 3.3. We use a value of $\epsilon = 0.1$ in our experiments listed below.

We can observe that on an average only 0.0221% of the points lie within the bound for the X considered and a maximum of 0.3293% points lie within the bound for the cleaned MNIST training data set corresponding to classes labeled 7 and 9.

Now, we increase the dimensionality of the training Data set. Each sample/image in the training Data set consists of 28×28 pixels. The pixel values range from 0 to

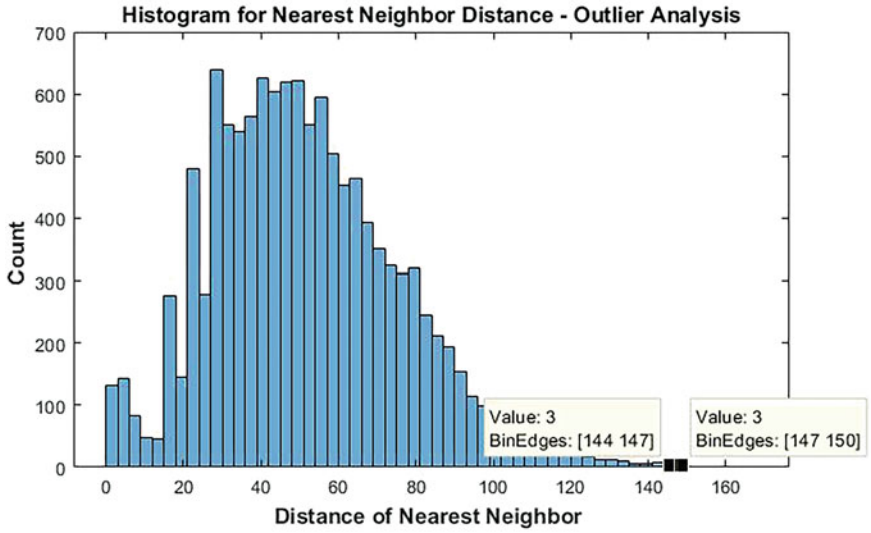

Fig. 3.2 Histogram of nearest neighbor distances $d(X, NN(X))$

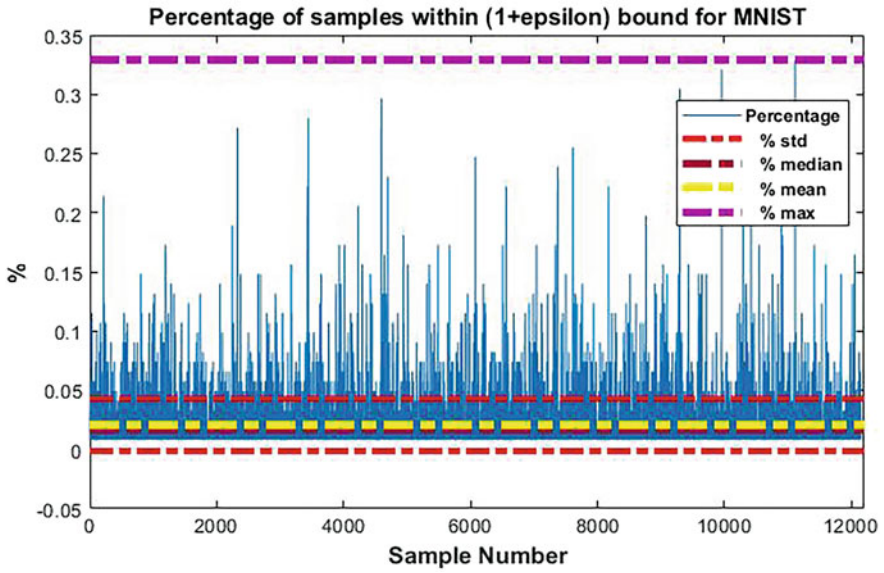

Fig. 3.3 Percentage of sample points falling within $(1.1 * d(X, NN(X))$ for each sample point X in the MNIST data set

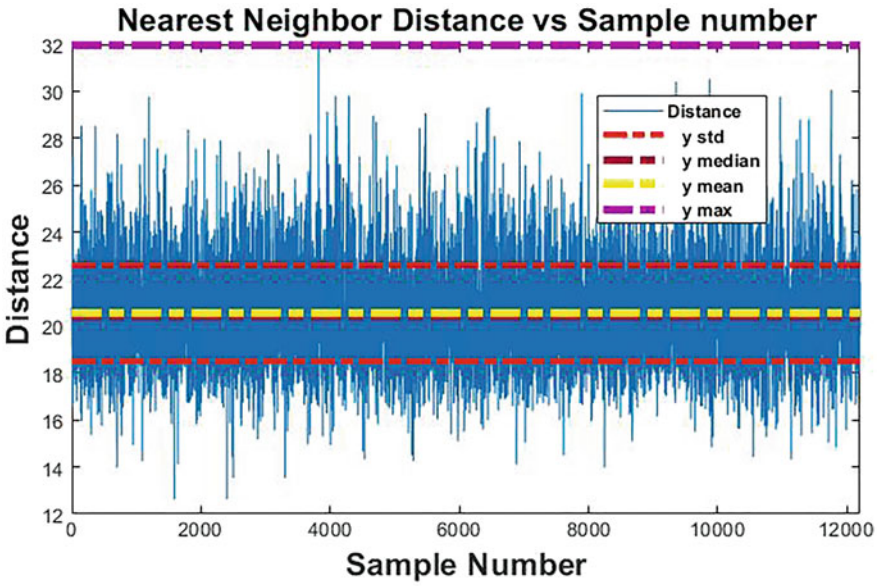

Fig. 3.4 Nearest neighbor distance for each of the 12,214 samples in the 6272 dimensional space for the MNIST data set

255. Converting each pixel value (using 8 bits) into a binary string of size 8 results in a total of $8 * 28 * 28 = 6272$ binary pixels per image. Viewing these elements as a one dimensional row vector of size 6272 results in the training data matrix, based on 12,214 training patterns from the classes labeled 7 and 9, of size 12214×6272 (where, each row is a binary vector of length 6272 bits corresponding to the image).

We repeat the experiments on the high dimensional data set (with dimension increased from 784 to 6272). Observe the plot of Nearest neighbor distances, for each of the patterns in the training data, in Fig. 3.4.

we can find that most of the distances lie between 18 and 22 units (one standard deviation bound) for the 6272 dimensional data set. Recall that for the 784 dimensional data set, the distances vary mostly from 30 and 80 units; so as the dimensionality increases, observe the average reduction in the spread of distance values. Like in the previous case, we identified and removed 93 outliers (X's) whose $d(X, NN(X))$ values are beyond 3 σ units from the mean. We depict the number of points falling within 1.1 times the distance to the nearest neighbor of each pattern in the 6272 dimensional MNIST Data set in Fig. 3.5.

We can observe that on an average 0.0361% lie within the $(1.1) \times d(X, NN(X))$ bound for the point X and maximum of 0.0395% for the cleaned 6272 dimensional MNIST Training Data set of classes labeled 7 and 9. **So, as the dimensionality increases, we observe reduction in the average value of** $d(X, NN(X))$ **and increase in the number of points within the** 1.1 **distance bound.** So, the Nearest Neighbor search becomes unstable. The increase in the number of points within the

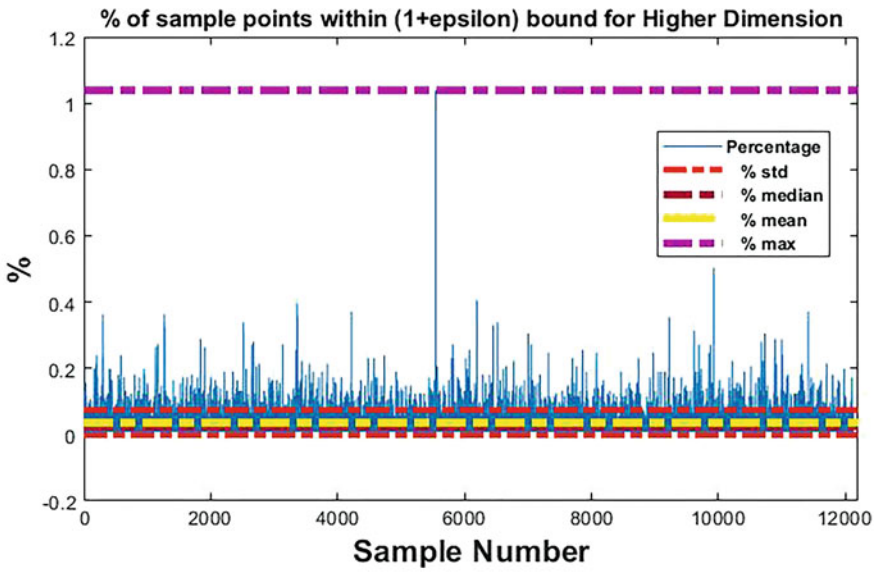

Fig. 3.5 Percentage of sample points within the $1.1 * d(X, NN(X))$ bound for each sample point in the 6272 dimensional MNIST data set

bound can be observed more clearly when we extend our analysis to a very sparse data set with a much higher dimensionality, like an image data set.

Let us consider the ORL-Face Data set. The data set has a total of 400 face images, (40 persons, 10 images/samples per person, each image is of dimension 112×92). So, the data matrix is of size $400 \times 10{,}304$. By repeating the experiments on this data, we get the Nearest Neighbor distance for each Sample as shown in Fig. 3.6 and percentage of points within the 1.1 bound for the ORL-Data set is plotted in Fig. 3.7. We can observe that on an average 1.36% lie within the 1.1 bound for a point X and a maximum of 6.51% in the case of the ORL Face Data set.

Now, we increase the dimensionality of the ORL Face Data set as earlier (each pixel value is expressed as a binary vector of size 8), so now the dimension of ORL Data set becomes 82,432 ($10{,}304 * 8 = 82{,}432$). The data matrix is of size $400 \times 82{,}432$. Now, consider the plots showing the Nearest Neighbor distance vs Sample in Fig. 3.8 and the percentage of points within the 1.1 bound for the ORL-Data set in Fig. 3.9.

We observe that the value of $d(X, NN(X))$ for the ORL Data set of lower dimension (10,304) varies from 500 units to 750 units. As the dimensionality increases, that is for the 82,432 dimensional data set, the value of $d(X, NN(X))$ is shrunk and varies from 175 units to 185 units. Interestingly, we observe from Fig. 3.9, that on an average about 50% of the samples in the data set lie within the 1.1 bound. **So we can abstract that, as the dimensionality increases the number of sample points within the** 1.1 **bound increases and the Nearest Neighbor finding**

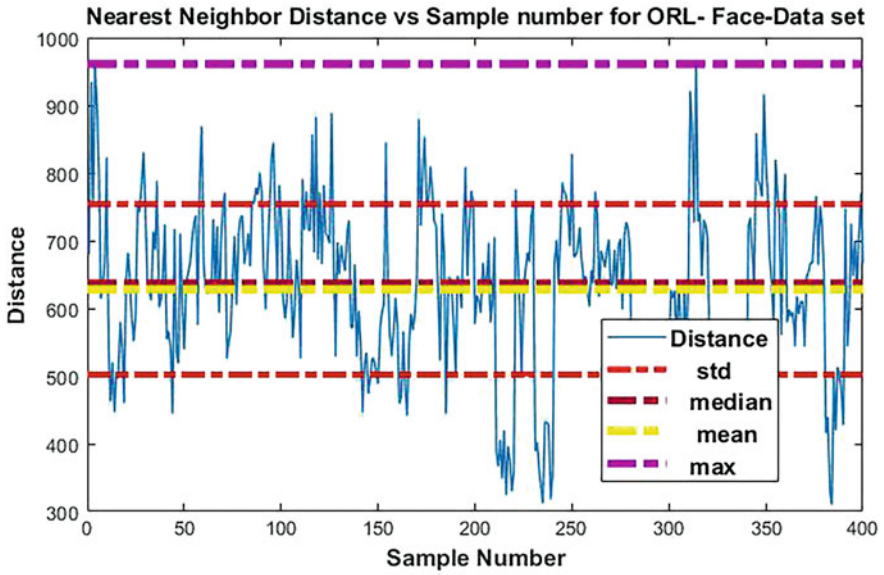

Fig. 3.6 Nearest neighbor distance for each of the 400 samples in the ORL Face Data set

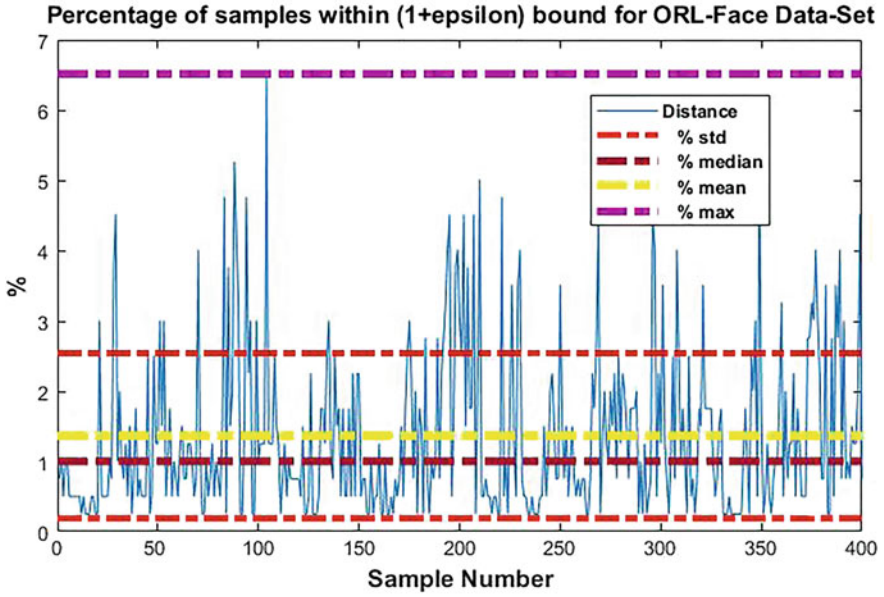

Fig. 3.7 Percentage of sample points within $1.1 * d(X, NN(X))$ bound for each sample point in ORL Face Data set

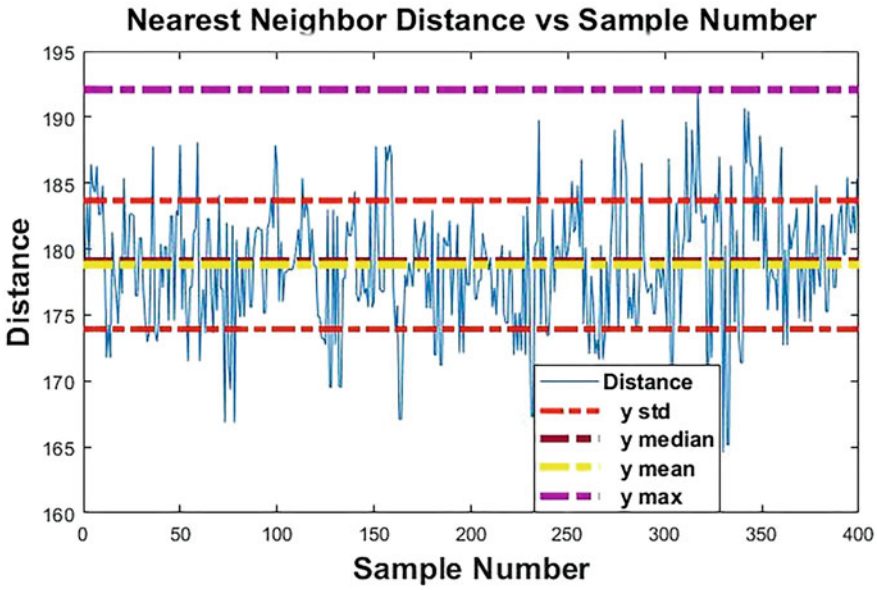

Fig. 3.8 Nearest neighbor distance for each of the 400 samples in the ORL Face Data set of dimension 82,432

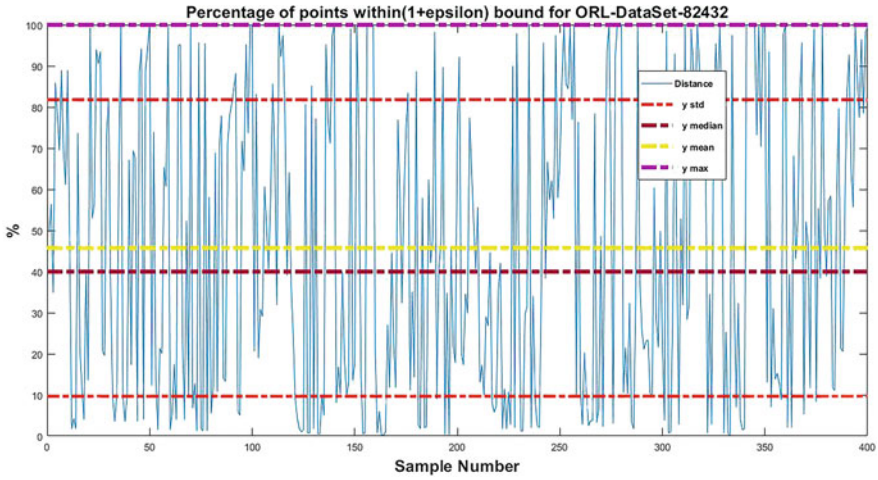

Fig. 3.9 Percentage of sample points within $1.1 * d(X, NN(X))$ bound for each sample point in the ORL Face Data set of dimension 82,432

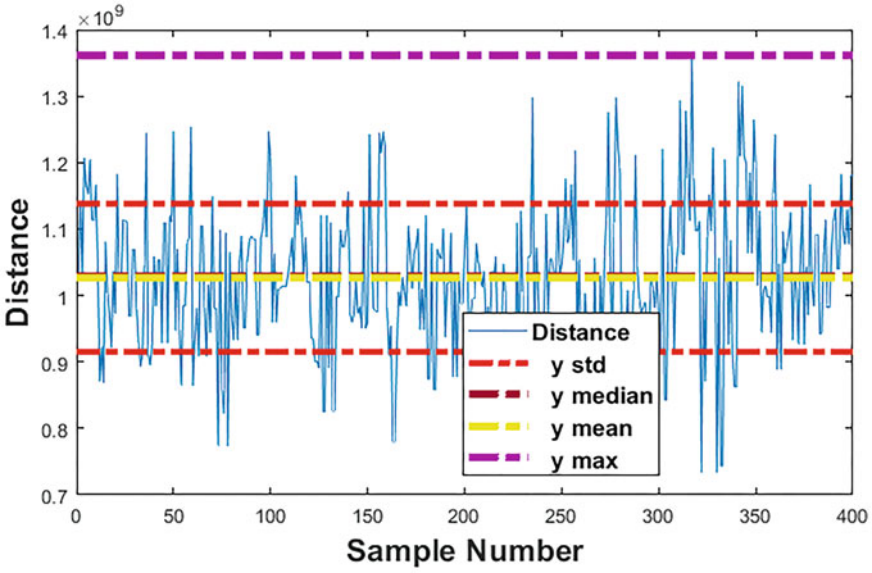

Fig. 3.10 Nearest neighbor distance for each of the 400 samples in the ORL Face Data set of dimension 82,432 using the L_r-Norm parameter with $r = 0.5$

algorithm becomes unstable, which means, that on an average, there is not much discrimination between the nearest neighbor and the farthest neighbor of a pattern X in a high dimensional space.

The performance of K-Nearest Neighbor Classifier on the ORL-Face Data Set can be worse because of the concentration effect. Let us repeat the experiments on the ORL Data Set with dimension 82,432 and using *Minkowski* distance measure with parameter $r = 0.5$ (fractional norm).

The results of the Nearest Neighbor distance vs Sample and percentage of points within the 1.1 bound are depicted in Figs. 3.10 and 3.11 respectively. From Fig. 3.11, we can observe that for *Minkowski* distance measure with parameter $r = 0.5$, the average number of samples in the data set which lie within the 1.1 bound is reduced to about less than 3%, so it may be observed that in higher dimensions, the fractional norm has a lesser concentration effect and so it gives a better discrimination.

There are two popular schemes to deal with this problem arising due to the high-dimensionality of the data.

1. To use *fractional norms* to compute the distances in high-dimensional spaces. The reason for using the fractional norms is that they provide a *larger dynamic range* for the distance values compared to the integer norms.
2. To compute an approximate nearest neighbour by combining multiple lower dimensional locality sensitive mappings.

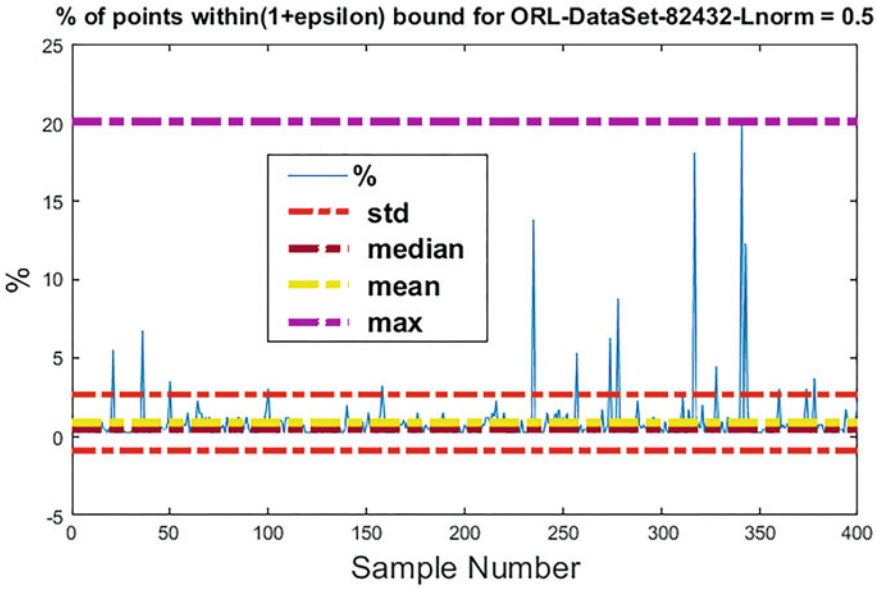

Fig. 3.11 Percentage of sample points within $1.1 * d(X, NN(X))$ bound for each sample point in ORL Face Data set of dimension 82,432 and L-norm with parameter $r = 0.5$

We discuss these solution directions in this chapter. Of course, it is important to note that these schemes may demand more computational resources.

3.3 Fractional Norms

A popular class of distance measures, between a pair of L dimensional vectors P and Q is the *Minkowski distance* given by

$$d(P, Q) = \left(\sum_{i=1}^{L} |P_i - Q_i|^r \right)^{\frac{1}{r}}.$$

Some instantiations based on selecting the value of r are:

1. L_∞ Norm: Here, $r = \infty$ and $d(P, Q) = maximum_i(|P_i - Q_i|), i = 1, \cdots, L$.
2. L_2 norm: In this case, $r = 2$ and $d(P, Q) = \left(\sum_{i=1}^{L} |P_i - Q_i|^2 \right)^{\frac{1}{2}}$ is the *euclidean distance and is the most popular.*
3. L_1 norm: In this case, $r = 1$ and $d(P, Q) = \sum_{i=1}^{L} |P_i - Q_i|$ is the *city-block distance.*

4. It is possible that r is a fraction. In such a case the resulting distance is called the *fractional norm.* It is not a metric as it violates the triangle inequality.

Example 3.3.1 Consider the patterns $X = (4, 2)^t$ and $X_1 = (2, 3)^t$, $X_2 = (2, 4)^t$. Then the various distance values are:

1. $r = \infty$:
   - $d(X, X_1) = Maximum(2, 1) = 2$.
   - $d(X, X_2) = Maximum(2, 2) = 2$.
   - The difference between $d(X, X_1)$ and $d(X, X_2)$ is 0.
2. $r = 2$:
   - $d(X, X_1) = (2^2 + 1^2)^{\frac{1}{2}} = 2.236$.
   - $d(X, X_2) = ((2^2 + 2^2)^{\frac{1}{2}} = 2.828$.
   - The difference between $d(X, X_1)$ and $d(X, X_2)$ is 0.592.
3. $r = 1$:
   - $d(X, X_1) = (2 + 1) = 3$.
   - $d(X, X_2) = (2 + 2) = 4$.
   - The difference between $d(X, X_1)$ and $d(X, X_2)$ is 1.
4. $r = 0.5$:
   - $d(X, X_1) = (\sqrt{2} + 1)^2 = 5.828$.
   - $d(X, X_2) = (\sqrt{2} + \sqrt{2})^2 = 8$.
   - The difference between $d(X, X_1)$ and $d(X, X_2)$ is 2.172.
5. $r = 0.1$:
   - $d(X, X_1) = (2^{0.1} + 1)^{10} = 1456.87$.
   - $d(X, X_2) = (2^{0.1} + 2^{0.1})^{10} = 2048$.
   - The difference between $d(X, X_1)$ and $d(X, X_2)$ is 591.13.

   Note that as the value of r keeps decreasing

- the distance value between a pair of patterns keeps *increasing.*
- the difference between $d(X, X_1)$ and $d(X, X_2)$ also keeps increasing illustrating the *improvement in the dynamic range.*

Next we present the results based on experimentation on two practical data sets.

1. *MNIST data Classes 7 and 9:* Here we have classified the test data corresponding to the classes 7 and 9. We have experimented with different Minkowski norms based on r values in the set $\{2, 1, 0.9, \cdots, 0.1\}$. The classifier used is the $KNNC$ with the value of K in $\{1, 3, 4, 5, 10\}$. We show the results of using different norms in Fig. 3.12. Some of the observations are:
   - This figure shows the results of using L_2, L_1 and a variety of fractional norms with the fractions going from 0.9 to 0.1 with a decrements of 0.1.

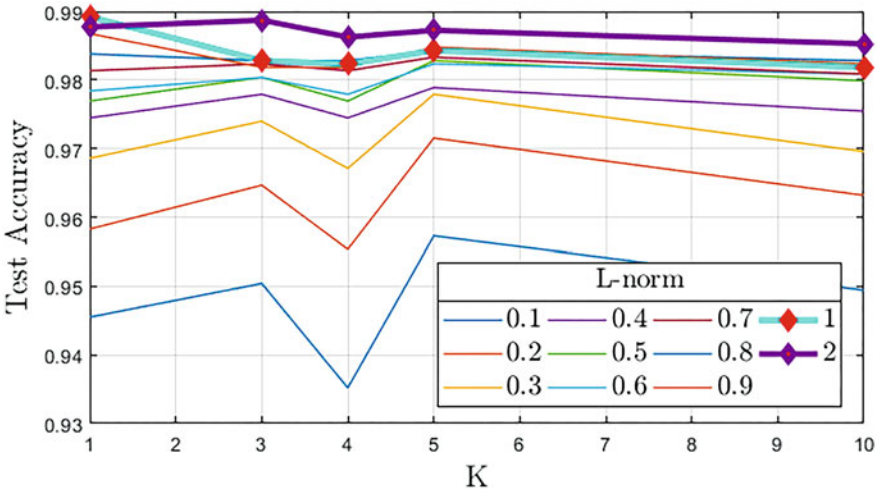

Fig. 3.12 Application of fractional norms on MNIST data set—class label 7 and 9, with different 'K' values in KNNC and different norms

- The X-axis depicts the value of K and the Y-axis shows the test accuracy using the $KNNC$.
- On an average, L_2 and L_1 norms are doing better. It could be because the dimensionality is not very high and also the number of training patterns is large.
- There is a dip in the accuracy in all the cases when $K = 4$.
- Another important property is that using the L_2 norm, the best performance is obtained at the value of $K = 3$. However, all the fractional norms perform the best when $K = 5$. Hence, they are able to extend the dynamic range and can effectively accommodate 5 nearest neighbors in making the decision.

We conduct the experiment with a reduced number of training patterns. We use 10 training patterns, 5 each from classes 7 and 9. We show the results of using different norms on this smaller data set in Fig. 3.13. The results may be summarized as follows:

- Observe that the fractional norm based on $r = 0.1$ is the best, on an average, across different values of K. This is because of the concentration effect that is less effective when small fractions are used.
- For larger values of K, the L_2 norm based result is the worst followed by the L_1 norm. This is because they are unable to accommodate a good number of neighbors from the correct class in the top K.

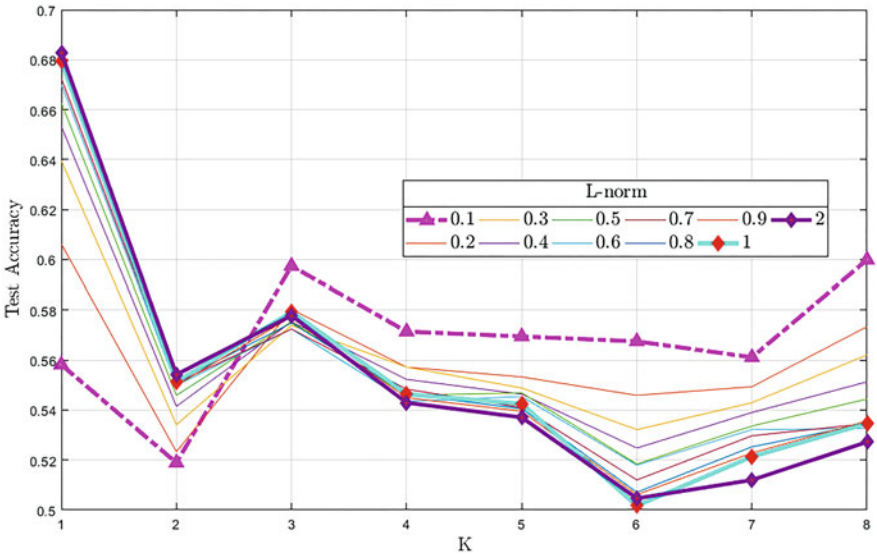

Fig. 3.13 Application of fractional norms on MNIST data set with a small size training data—class label 7 and 9, with different 'K' values in KNNC and different norms

2. Now we consider the ORL face data set having 400 images in 92×112 (10,304) dimensional space. The results are shown in Fig. 3.14.
   - Here 320 face images, 8 images per person, are used for training and the remaining 80 images, 2 images person, are used for testing using the $KNNC$.
   - Here also we have the value of K on the X-axis and the test accuracy is depicted on the Y-axis.
   - It is easy to see that the fractional norms are consistently outperforming the L_1 and L_2 norms.
   - The worst performer is the L_2 norm which fails to break the concentration effect in high dimensions.
   - Note that the dimensionality is much larger compared to the number of training patterns. The ratio is 10,304:320 which could be the reason for the superior performance of the fractional norms.

3.4 Locality Sensitive Hashing (LSH) and Applications

In many practical higher dimensional data sets, performance of the Nearest Neighbor based algorithms is poor. As the dimensionality increases, decision making using the nearest neighbor gets affected as the discrimination between the nearest and farthest neighbors of a pattern X diminishes. **LSH** performs well in finding the NN of a point X in higher dimensional space. We achieve this by matching

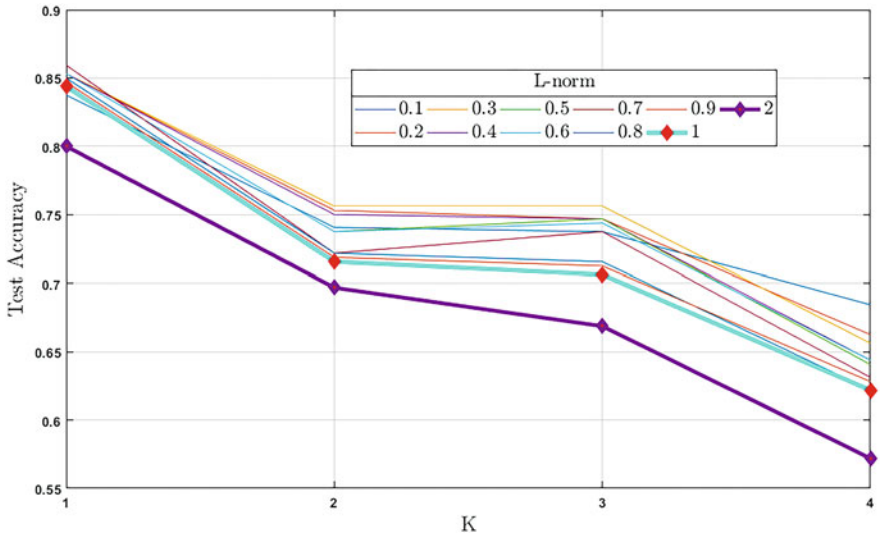

Fig. 3.14 Application of fractional norms on ORL data set, with different 'K' values using KNNC

on some randomly chosen q of the L dimensions l times. Based on these partial matchings multiple times, we find the approximate NN of a test pattern X. A larger sized training data set can be pre-processed, such that only the points which lie in the same bucket as that of X are examined further to find the approximate NN. So, one or more nearest neighbors of X are obtained quickly using LSH. *Hashing* is the process of exploiting one or more functions to map data in higher dimensional space to a lower dimensional space.

Example 3.4.1 Consider a data set having *binary vectors* corresponding to positive integers from *0 to 15* using 4 bits. Let b_4, b_3, b_2, b_1, represent their bit patterns. So, i takes value from the set $\{4, 3, 2, 1\}$, where $i = 1, 4$, represent *msb and lsb respectively*. So, by choosing different values of i, the 4-bit binary patterns will be hashed to two values (buckets) of *binary* 0 and *binary* 1 as shown in Table 3.1.

If we choose the LSB (i = 1) for partial matching, then all the even valued numbers hash to bucket 0 and the odd valued ones will be hashed to bucket 1. Similarly, if we select the MSB (i = 4) for partial matching, then the numbers in the range 0–7 are mapped to bucket 0 and the remaining go to bucket 1. A sample test pattern, based on the partial matching, is mapped to the respective bucket and

Table 3.1 Simple example for hashing

ith bit of X	Bucket for *binary* 0	Bucket for *binary* 1
1	0,2,4,6,8,10,12,14	1,3,5,7,9,11,13,15
4	0,1,2,3,4,5,6,7	8,9,10,11,12,13,14,15

further exploration happens only in the selected bucket. **Locality Sensitive Hashing (LSH)** is hashing (mapping) of data points in a higher dimensional space to a lower dimensional space, such that data points which are similar/closer in the higher dimensional space are mapped to the same bucket in the low dimensional space with a very high probability.

Let us analyse the behaviour of LSH using binary vectors.

- Let the ith point, X_i, be an L-bit binary (either 0 or 1) string/vector. Let X_i^j, $j = 1, \cdots, L$ be the jth bit of X_i.
- Let us use hamming distance between the test pattern X and a training pattern X_i (both are L-bit strings).
- For L-bit strings X and X_i, if $Prob[X_i^j = X^j] \geq r$, then patterns X_i and X have identical values in at least rL of the L bit locations.
- This means that the hamming distance between X and X_i is less than or equal to $L - rL$ and r is the probability that X_i and X are equal on a randomly selected bit.
- If we select q out of L bits randomly, then probability that X and X_i have the same values on all these q bits will be at least r^q.
- So, probability that they fail to match on at least one of the q bits will be less than $1 - r^q$.
- If we select randomly q bits l times independently, then probability that X_i and X fail to match on at least one of the q bits in all the l selections is less than or equal to $(1 - r^q)^l$.
- So, probability that X_i and X share all the q bits in at least one of the l selections is greater than or equal to

$$1 - (1 - r^q)^l.$$

- If X_i and X have identical values on one of the l selections of q bits, then X and X_i match partially (at least on q of the L bits); as the value of q increases the match between X and X_i will get better.
- However, if q gets larger then r^q gets smaller and $(1 - r^q)^l$ tends to 1.
- In order to exploit the matching based on these random choices, we need $(1-r^q)^l$ to tend to 0. This can happen if q is large (and still $1 - r^q \leq 1$, so that X and X_i match better) and l also is large so that $(1 - r^q)^l$ tends to 0.
- So, in LHS we make l choices (tables) each of size q bits. We can extend the same idea to even non-binary vectors of size L.

Now we report our experimental results using LSH on the $MNIST$ data. The training set consists of the first five patterns of class labelled 7 and the first five patterns of class labelled 9 (from the total training set). Similarly, the test set used consists of the first five patterns of classes labelled 7 and 9 (from the total test set). These smaller sets of training (10 patterns) and test (10) patterns are used for performing classification using the Nearest Neighbor classifier and the LSH based approximate NN classification experiments. The results are plotted in Fig. 3.15.

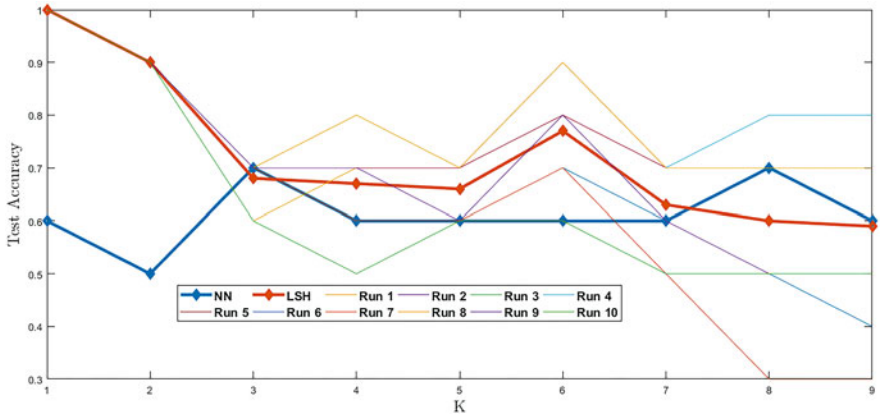

Fig. 3.15 Application of LSH on MNIST data set with a small size training data—class label 7 and 9, with different 'K' values in KNNC

The LSH experiment is performed 10 times and the average values are shown in the figure. From Fig. 3.15, we can observe that on an average LSH outperformed the Nearest Neighbor based classifier.

3.5 Summary

In this chapter, we have examined the role of NNC and its variants in classifying high-dimensional data sets. Some of the important comments are:

1. The NNC and $KNNC$ are inadequate to classify the test patterns properly due to the *concentration effect.*
2. We have observed experimentally that most of the points in the data set fall within a distance bound of 1.1 that of the $d(X, NN(X)$ making it difficult for the NNC in the high-dimensional cases.
3. There are two different solution directions that we considered in this chapter. They are based on either fractional norms or using LSH.
4. We have observed that the fractional norms offer a wider dynamic range for the distances to overcome the concentration problem. This was experimentally shown on two different data sets.
5. The LSH scheme is explained and is used on the $MNIST$ data set to show its superiority over the NNC and its variants on an average.
6. We will consider another option to deal with the concentration problem, using dimensionality reduction, in the later chapters.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM **51**(1), 117–122 (2008)
2. LSH Algorithm and Implementation (E2LSH): https://www.mit.edu/~andoni/LSH/
3. Murty, M.N., Susheela devi, V.: Introduction to Pattern Recognition and Machine Learning. World Scientific/IISc Press (2015)

Chapter 4
Representation Using Linear Combinations

Abstract This chapter deals with linear representations. Here, each new feature is a linear combination of the given input features. Even feature selection may be viewed as a special case of the linear representation schemes. We specifically consider, in this chapter, various types of feature selection schemes, principal components, random projections and non-negative matrix factorization.

4.1 Introduction

Linear representation is a very popular and relatively simpler scheme for representing patterns. Here, a feature g_j is extracted such that it is a linear combination of the given L features, $f_1, \cdots, f_L$. So,

$$g_j = \sum_{i=1}^{L} \alpha_{ij} f_i,$$

where α_{ij} is a real number and it indicates the contribution of feature f_i to the extracted feature g_j. Further, we typically extract l features $g_1, g_2, \cdots, g_l$ where $l < L$.

4.2 Feature Selection

Note that it is possible to view feature selection as a special case of such a linear combination. For example, g_j can be the same as f_p, if $\alpha_{pj} = 1$ and $\alpha_{ij} = 0$ if $i \neq p$. Further, we have seen in Chap. 2 that the feature selection schemes are categorized further into

1. *Filter Methods:* In filter methods, we rank the L features using a fitness measure that exploits the class labels but does not use any classifier. Based on the ranking,

M. N. Murty, M. Avinash, *Representation in Machine Learning*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-981-19-7908-8_4

the top l features are selected and used in building the ML model. There are different fitness measures to evaluate and rank the features. They are based on:

- *Distance:* We prefer values of a feature from the same class to be such that the intra-class distances are smaller and inter-class distances are larger.
- *Dependency:* Here, the correlation between a feature and a class is exploited in ranking. If the correlation is larger for a class and smaller for the other classes, then the feature in ranked better.
- *Mutual information* (MI:) If a feature and a class have a larger MI value, then the feature is good. It is seen that MI based feature selection works well on high-dimensional datasets.

2. *Wrapper Methods:* In this case, the selection of a subset of features is based on the performance of a classifier on the selected subset. The subset of features which gives the maximum classification performance is selected. This subset of features is used in designing the ML model. There are a variety of approaches including *genetic algorithm* based approaches that employ this method. These schemes can be more expensive compared to the filter based schemes because the number of subsets is much larger than the number of features.
3. *Embedded Methods:* Here, an ML model is built using the L features and the model directly selects/indicates the relevant subset of features. Several classifiers including ones based on decision trees, support vector machines, and Bayes classifier can be exploited to realize such an embedded scheme.

We will present some experimental results based on these schemes in this chapter.

1. Filter methods:
   Let us consider the patterns of classes labelled 7 and 9 from the *MNIST* data set. Each pattern is a 28×28 image, whose pixel values range from 0 to 255. Every pattern is converted into a binary row vector of dimension 1×784, such that a pixel value greater than 127 is replaced by 1 and pixel value less than or equal to 127 is substituted by 0. These row converted binary patterns are stacked to form the training set of size 12,214 and test set of size 2037 respectively. Each pattern is a binary vector of length 784.

   For each feature, the euclidean distance between the feature and its class label is calculated and shown in Fig. 4.1. Similarly *Mutual Information* and absolute value of *correlation coefficient* of each feature is calculated and shown in Figs. 4.2 and 4.3 respectively. The *correlation coefficient* of features whose value does not change throughout the training data set is set as $zero$ (since *MATLAB* returns **NAN** for those constant features).

   Features which represented pixel values in the top, bottom, left and right extremes of the image are mostly of same value (zeros) so they will not contribute to classification, which is observed from the *Mutual information* values in Fig. 4.2. The *Mutual information* becomes zero for feature number less than 100 and greater than 700 which correspond to the top and bottom pixels in the image. Similarly, *Mutual information* minima (touching 0) in Fig. 4.2, represents the left

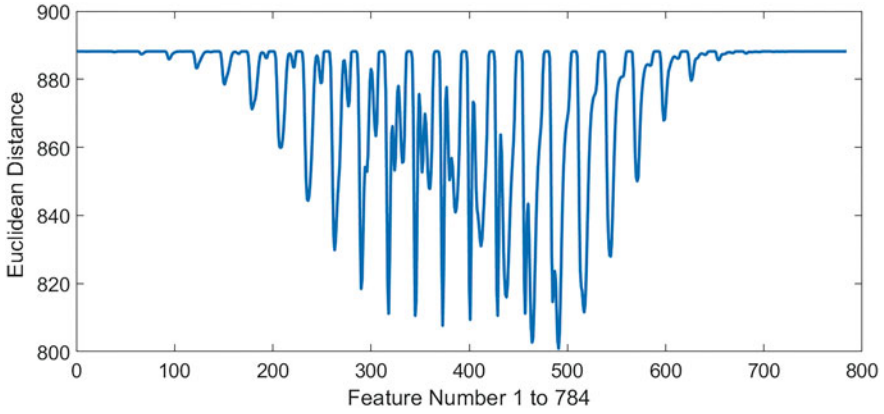

Fig. 4.1 Euclidean distance of each feature to the Class label

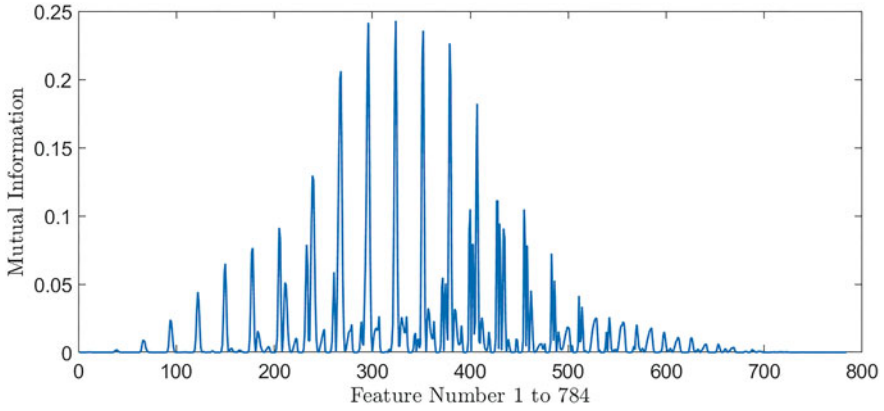

Fig. 4.2 Mutual information of each feature

and right extreme pixels of the image (whose values are constant throughout the training data set).

Based on the above mentioned *Filter methods*, 80 features are selected out of 784 features. These 80 features are used for classification by *Decision tree* classifier with Maximum number of splits, $N = 9$ and *k-NNC* classifier with $K = 10$. The results are shown in the bar (Fig. 4.4).

From Fig. 4.4, we observe that the accuracy of *Decision Tree* with 80 selected features and all 784 features, remains almost the same, since *Decision tree* automatically selects the important features and the *test accuracy* of *Decision tree* is low compared to that of *KNNC*, and further since the classes 7 and 9 are having highly non linear decision boundary, we need a deeper tree to classify correctly. Since, *KNNC* is a non linear classifier, it is able to perform well. Out of the *filter*

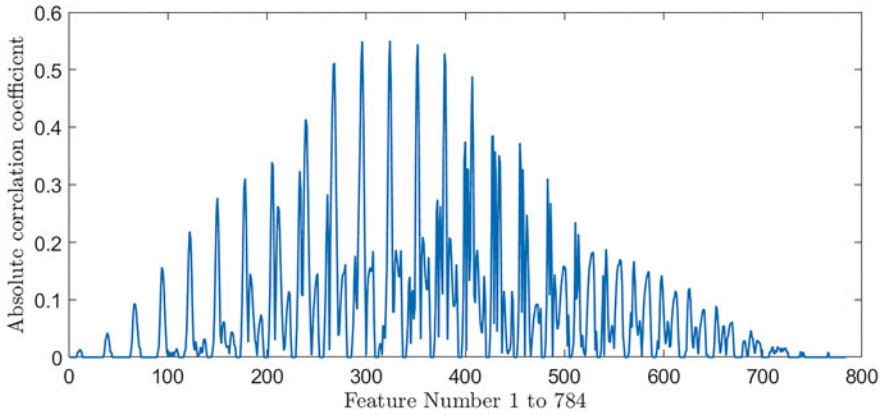

Fig. 4.3 Absolute value of correlation coefficient of each feature

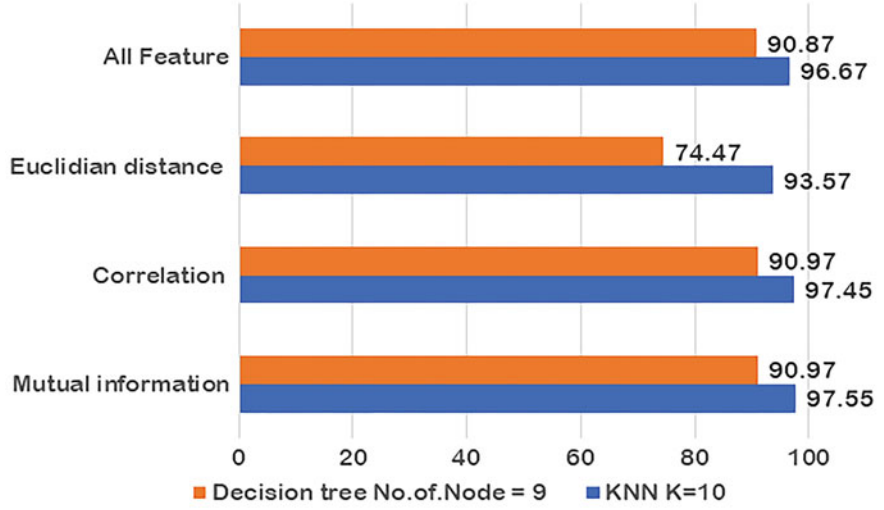

Fig. 4.4 Comparison of various filter methods

methods considered, *Mutual Information* based feature selection performed the best on this data set.

2. Wrapper methods:
   Refer to Sect. 4.3, where we have used *Genetic Algorithm* to select the optimum subset of 100 Principal components from the total set of 10,304 Principal components for the **ORL** face data set, having the *test accuracy* of *KNNC* classifier as the fitness criterion for selecting features.
3. Embedded methods:
   Let us consider the *Iris* data set. It has four features: *Petal length*, *Petal Width*, *Sepal length* and *Sepal width*. We have used *Decision Tree* to classify the three

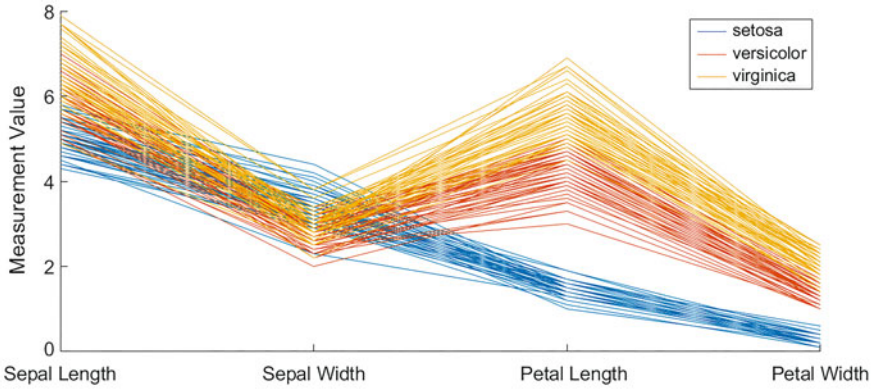

Fig. 4.5 Data visualization of Iris data set using parallel coordinate plot

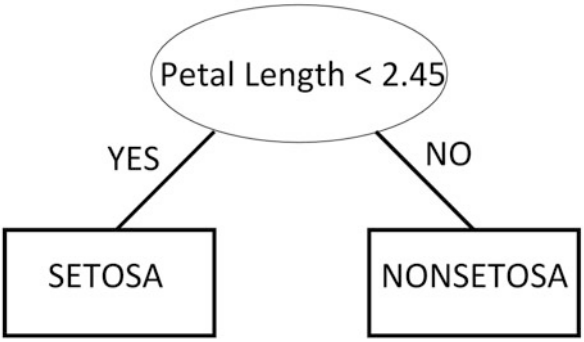

Fig. 4.6 Decision tree on Iris data set

flower species (*setosa*, *vericolor* and *virginica*) based on these four features. We used *Parallel coordinates plot* from **MATLAB**, to visualize the *Iris* data set in Fig. 4.5.

From Fig. 4.5, feature *Petal Length* clearly classifies class label *setosa*. So, the *Decision Tree* classifier has given the first preference to this feature and keep it at the *Root* of the *Decision Tree*, as shown in Fig. 4.6.

Let us calculate the *predictor importance*. It is calculated by summing changes in the mean squared error due to splits on every predictor and dividing the sum by the number of branch nodes. The estimates of *predictor importance* are shown in Fig. 4.7. From Fig. 4.7, we observe that predictor *Petal length* is the most importance feature.

Root of the *Decision Tree* is shown in Fig. 4.6 and the rule captured by the tree is as follows:

(1) if PetalLength $<$ 2.45 then setosa else (PetalLength $\geq$ 2.45) nonsetosa

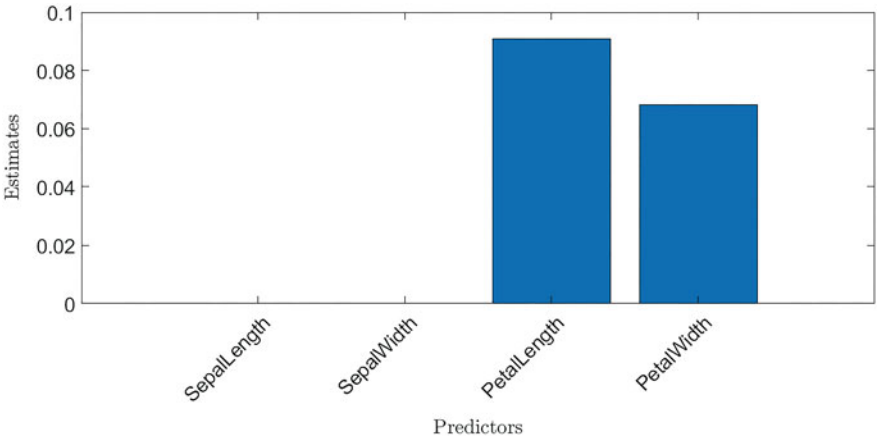

Fig. 4.7 Predictor importance of predictors-decision tree

4.3 Principal Component Analysis

The most popular linear representation scheme is based on principal components (PCs). PCs are the leading eigenvectors of the covariance matrix of the data. Eigenvectors are considered in the decreasing (non-increasing) order of their corresponding eigenvalues. The variance in different directions, that is present in the data, is characterized by the eigenvalues. The covariance matrix is symmetric; so, it is possible to select the eigenvectors to be orthonormal. The top l eigenvectors are used in the analysis where the value of l is chosen based on total variance explained by the top l eigenvalues.

Let us consider the patterns of MNIST data set from Classes labelled 7 and 9. The dimensionality of the data set is 784 (28×28 pixels in each image). From the covariance matrix of the Training data patterns, 784 eigenvalues and their eigenvectors are calculated. The eigenvectors are sorted in descending order based on their corresponding eigenvalues. The dimensionality of both the training and test sets is reduced based on the projections of the data points on the top l eigenvectors. The K-Nearest Neighbor classifier with K = 3, (three-nearest neighbor classifier) is used for classification. The results are plotted in Fig. 4.8. The X-axis depicts the value of l, the number of top eigenvectors used for classification and the Y-axis shows the test accuracy using the $KNNC$ with K = 3.

From Fig. 4.8, we observe that feature selection based on first few Principal components gave good test accuracy. It is to be noted that first few PCs will only be useful for discriminating between groups if within- and between-group variation have the same dominant directions.

We will show that first few PCs may not always give the best representation for classification; that is best test accuracy. In practice, it is possible that the selection of

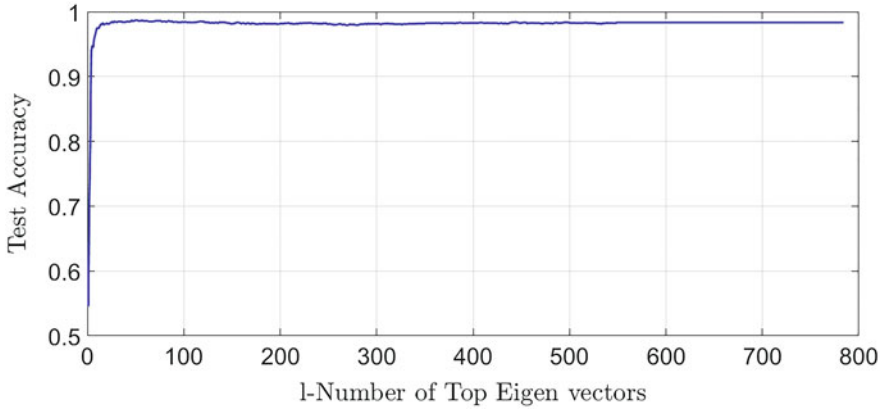

Fig. 4.8 Application of PCA on MNIST data set class label 7 and 9, with 'l' top principal components

last few PCs may provide a better discrimination. So, the feature selection should be based on subset of PCs from the entire spectrum of PCs. We will discuss this next.

For the experiments we use the ORL face Data set. Out of 400 images, 320 images are used for training and remaining 80 images for testing. The data set is divided, such that for each person, eight patterns are used for training and two patterns for testing. The dimension of the data set is 10,304 (92×112). From the covariance matrix of the training set, 10,304 Principal components and the corresponding eigenvalues are calculated, then these Principal components (PC) are sorted based on their eigenvalues (in descending order), such that the first PC corresponds to the highest eigenvalue and the last PC is based on the least eigenvalue.

Dimensionality reduction of both the training and test sets is based on using the first l PCs and KNNC, with K = 3 is used as the classifier. From Fig. 4.9 for the first 108 PCs, we got the maximum test accuracy of 0.95 and for the first 100 PCs, the test accuracy is 0.9375, When we use all the 10,304 PCs, the accuracy drops to 0.925.

To illustrate the point that first few PCs will not always give a better representation in the lower dimensional space, we randomly selected 100 PCs from different sections of the entire spectrum of PCs as shown in Table 4.1.

We present the results in Fig. 4.10.

From the figure, we observe that, for some random selection of 100 PCs, we got test accuracy of 0.9625, which is better than the test accuracy obtained from the first 108 PCs and first 100 PCs.

To find the optimum sub set selection of PCs from the entire spectrum of 10,304 PCs, we used Genetic Algorithm (a stochastic search technique), with population size 150 and maximum of 200 generations and show the results in Fig. 4.11. Here X axis represents the lth PC (as we have selected 100 PCs, it varies from 1 to 100) and

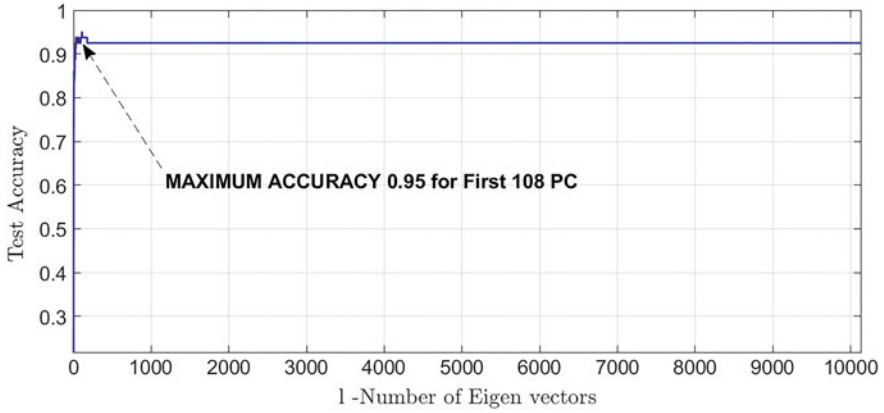

Fig. 4.9 Application of PCA on MNIST data set class label 7 and 9, with 'l' top principal components

Table 4.1 Random selection of PCs from the set of 10,304 PCs

Section Name	Number of PCs
First 100 [1–100]	60
[1000–3000]	10
[4000–6000]	10
[7000–9000]	10
Last [10,204–10,304]	10

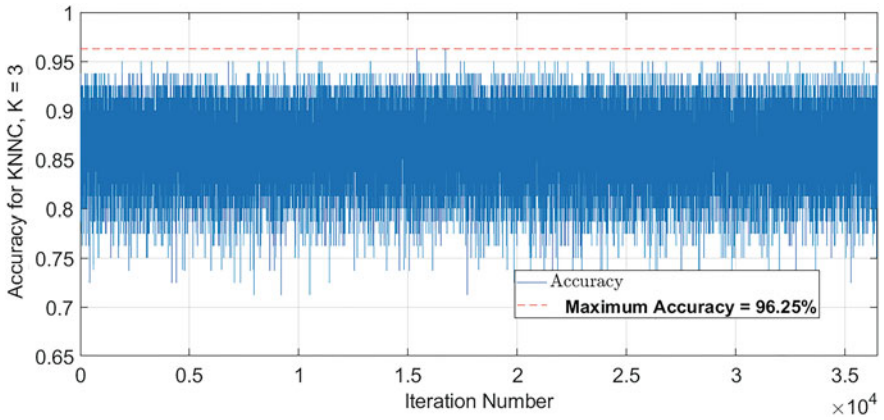

Fig. 4.10 Application of PCA on ORL data set, with random selection of 100 principal components

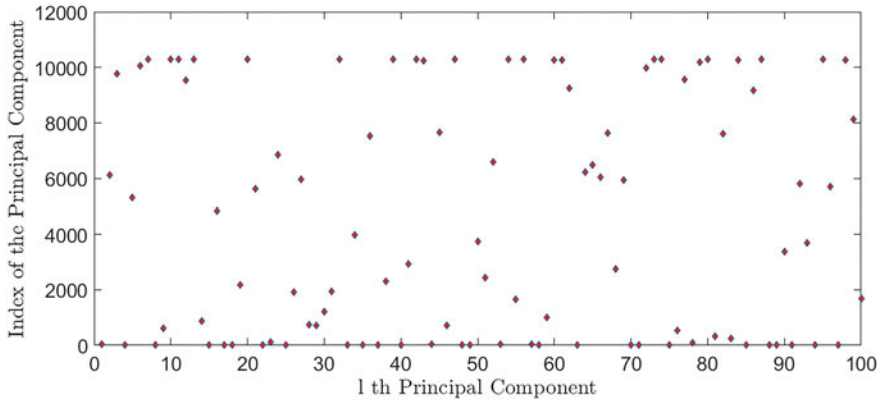

Fig. 4.11 Application of PCA on ORL data set, with 100 principal components given by GA

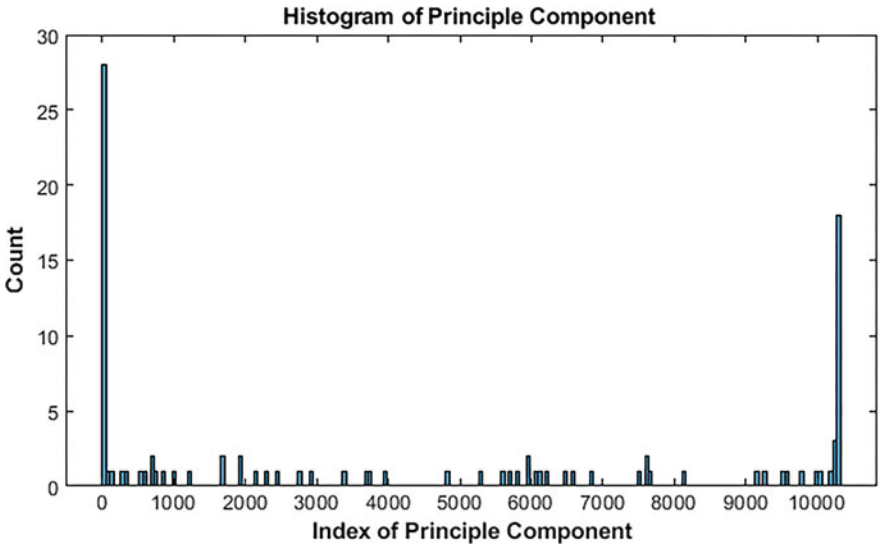

Fig. 4.12 Histogram of Index values of 100 Principal Components given by GA

Y-axis represents the index of the selected PC from the set of 10,304 PCs, where index 1 represents the first PC and index 10,304 represents the last PC.

From Fig. 4.11, we observe that the optimum sub set of 100 PCs, which gave a test accuracy of 0.9625 has indices in the entire spectrum (from 1 to 10,304). So, neither the first few PCs nor the last few PCs may always give better discrimination in the lower dimensional space. The optimum sub set of PCs fall in the entire spectrum, which is evident from the Histogram in Fig. 4.12.

From the Histogram in Fig. 4.12, we also observe that the Last few PCs also play a significant role in discrimination and the optimum choice of PCs selects from the entire spectrum.

4.4 Random Projections

We have discussed in Chap. 2 that in the case of random projections, we reduce the dimensionality of the data from the L-dimensional space to a lower-dimensional space of dimension l, where l can be much smaller than L. This is achieved by using a matrix R that has random entries.

Some of the properties of random projections are:

- $B_{n\times l} = A_{n\times L} R_{L\times l}$
- The value of l can be much smaller than L.
- If the entries in R are independently selected from a zero mean and unit variance distribution, then it is possible to show that

$$E[|| b_i ||^2] = || a_i ||^2,$$

  where b_i is the ith row of B and a_i is the ith row of A.
- Further, by selecting the value of $l = \mathcal{O}(logn/\epsilon^2)$ we can approximately preserve pairwise distances upto a factor of $(1 \pm \epsilon)$, where ϵ is a small real number.
- Similarly it is possible to preserve dot products. That is

$$E[b_i^t b_j] = a_i^t a_j$$

Let us consider the MNIST data set, the subset of patterns belonging to class labels $\{0, 1, 7, 9\}$ from both training set and test set are collected. Let us call this subset as '$\mathbf{X}$'. Then, $\mathbf{X}$ consist of 29,031(= 24,879 + 4152) patterns where first 24,879 patterns belong to the training set and the next 4152 belong to the test set. The dimension of $\mathbf{X}$ is $29{,}031 \times 784$, where each row represents a pattern.

The dimensionality of $\mathbf{X}$ is reduced to $\mathbf{l}$ using *Gaussian Random Projection* model from *python sklearn* package. After, the dimensionality is reduced, the training set patterns (24,879 patterns) and test set patterns (4152 patterns) are separated. The *KNNC* classifier with different $\mathbf{K}$ values is used for classification. The experiment is repeated for five different seed values $\{1, 2, 3, 4, 5\}$ and the average training set and test set accuracies obtained are shown in Table 4.2. The same experiment is repeated using *Sparse Random Projection* model which uses a Sparse Random matrix with density parameter as $1/3.0$, the results are tabulated in Tables 4.3 and 4.4 respectively.

Let us consider the ORL face data set. The Data set consist of 400 faces each with dimension 112×92. The original dimension of the data set is 10,304. The

Table 4.2 Gaussian Random Projection based dimensionality reduction on MNIST data set

l	eps	K	Mean train accuracy	Mean test accuracy
100	0.1	3	0.993432212	0.982996146
100	0.1	5	0.990779372	0.981358382
300	0.1	3	0.99462197	0.984537572
300	0.1	5	0.992555971	0.983815029
500	0.1	3	0.995023916	0.98473025
500	0.1	5	0.992756944	0.98371869

Table 4.3 Sparse random projection based dimensionality reduction: MNIST data set

l	eps	K	Mean train accuracy	Mean test accuracy
100	0.1	3	0.992869488	0.981888247
100	0.1	5	0.989959404	0.980684008
300	0.1	3	0.994372764	0.985356455
300	0.1	5	0.992089714	0.983140655
500	0.1	3	0.994597854	0.985308285
500	0.1	5	0.992379115	0.984104046

Table 4.4 Achlioptas random projection based dimensionality reduction: MNIST data set

l	eps	K	Mean train accuracy	Mean test accuracy
100	0.1	3	0.993335745	0.981743738
100	0.1	5	0.990956228	0.980828516
300	0.1	3	0.994469231	0.984007707
300	0.1	5	0.992113831	0.982755299
500	0.1	3	0.995023916	0.984104046
500	0.1	5	0.992580088	0.983188825

dimensionality of the data set is reduced to l components (selected for different values of *eps*) using the following Projection matrices.

- *Gauss Random Projection matrix*
- *Sparse Random Projection Matrix*
- *Achlioptas Random Projection matrix*

The minimum size of the reduced dimension l for a given *eps* is selected based on *Johnson-Lindenstrauss lemma*.

The experiment is performed for different values of *eps* ranging from 0.1 to 0.95 in the steps of 0.05. For each value *eps*, the minimum value of N is selected based on *Johnson-Lindenstrauss lemma*. Once the dimension of the data set is reduced, it is divided into training and test set such that for each person, eight patterns are selected for training and the remaining two patterns for testing .*KNNC* is used as the classifier for different values of K which range from 1 to 8. The results of the experiments are plotted in Figs. 4.13, 4.14, and 4.15 respectively.

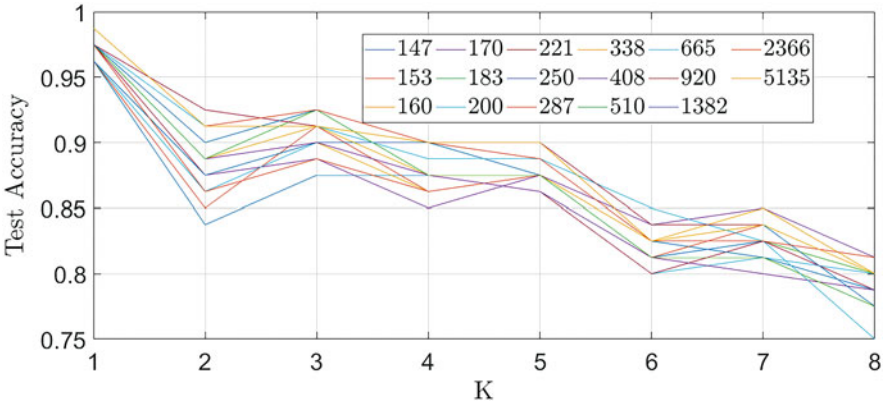

Fig. 4.13 Dimensionality reduction based on gauss Random Projection matrix: ORL Face Data set

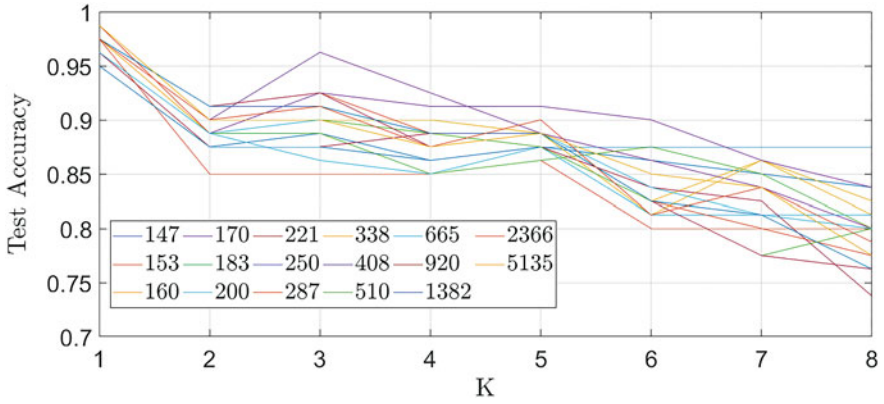

Fig. 4.14 Dimensionality reduction based on sparse Random Projection matrix: ORL Face Data set

4.5 Non-negative Matrix Factorization

Let us consider the NMIST Data set, the training and test patterns belonging to class labels 0, 1, 7, 9 are used for the experiments. The *NMF* module from *SKLearn* is used for dimensionality reduction. The tuning parameters for the modules are *alpha*, which is the constant that multiples the regularization term and *L1 ratio*, which controls the contribution of $L1$-$term$ errors in the regularization term. After reducing the dimensionality, *KNNC* classifier with $K = 5$ is used as the classifier. The NMF may be viewed as factorizing the data matrix A into W and H matrices. So, if A is of size $n \times l$, then we have

$$A_{n \times l} = W_{n \times K} * H_{K \times l}.$$

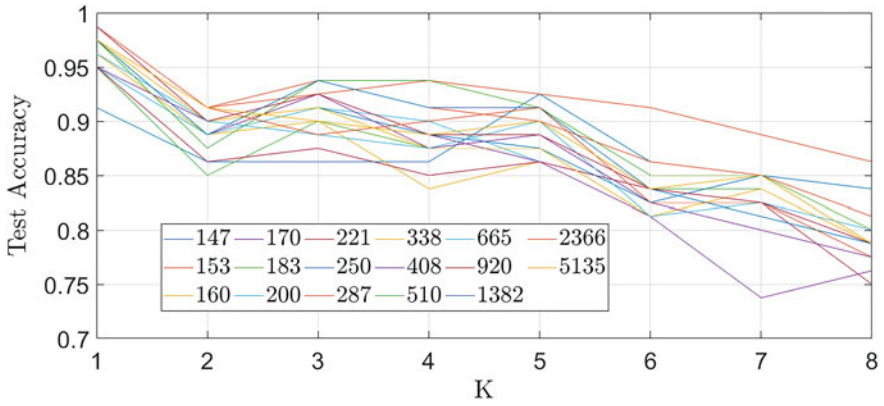

Fig. 4.15 Dimensionality reduction based on Achlioptas Random Projection matrix: ORL Face Data set

Table 4.5 NMF on MNIST Data set for class label 0, 1, 7, 9

l	alpha	L1	Train accuracy	Test accuracy
350	10	0	0.986414245	0.945086705
350	5	0	0.98548977	0.933526012
350	50	0	0.990112143	0.971820809
350	15	0	0.988705334	0.957129094
350	25	0	0.990152337	0.967003854
350	75	0	0.990272921	0.972302505
350	100	0	0.991076812	0.9727842

The experiment is repeated for various values of *alpha*, *L1 ratio* and l (the number of reduced components after dimensionality reduction). By Trail and error, the following results are obtained and tabulated in Table 4.5. Let us consider the smaller version of ORL-Data set, which consists of 4096 features instead of 10,304 features. The total data set consists of 400 faces. The Data matrix is of size 400×4096, where each row represents a face pattern.

NMF function from **MATLAB** is used for dimensionality reduction experimentation. After dimensionality reduction, the data set is divided into 320 Training patterns and 80 test patterns. *KNNC* classifier, with K values range from 1 to 8 is used for classification. The initialization value for W(*W0*) matrix plays a crucial role in determining the best representation (W) and test accuracy (making all the other parameters to take their default values and Maximum iteration step is taken as 100).

For the Random initialization of W (*W0*) and H, the results are shown in Fig. 4.16. Now, for the 400 features, Mutual information is calculated. Based on the Mutual information values, features are sorted and W is initialized with top l features (where, l is the number of reduced components). The *Multiplicative Update* algorithm is modified such that H is updated based on the initialization of W as the first step.

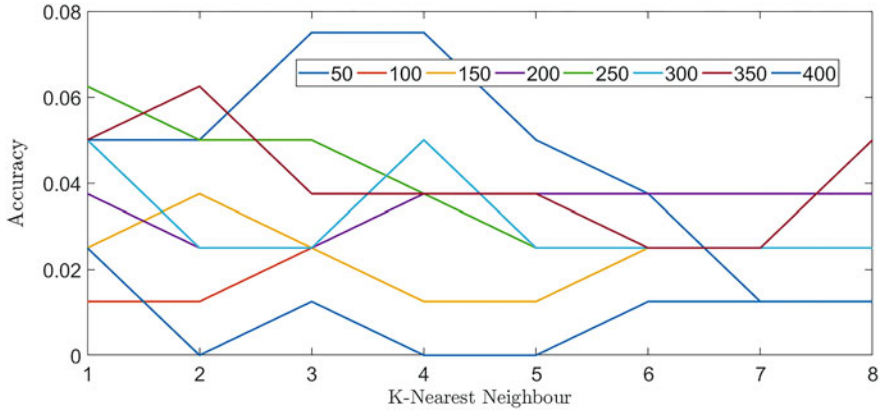

Fig. 4.16 Dimensionality reduction based on NMF on reduced ORL Face Data set with random initialization of W

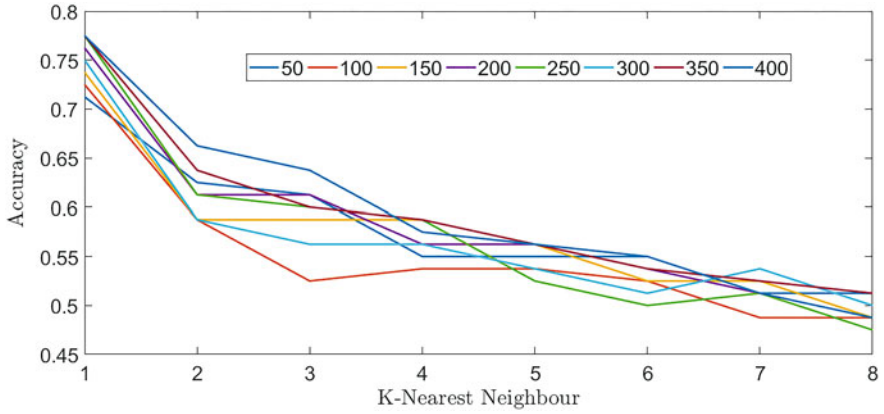

Fig. 4.17 Dimensionality Reduction based on NMF on reduced ORL Face Data set with Mutual information based initialization of W

The results for the *Mutual information* based feature initialization W are shown in Fig. 4.17. The *Root Mean Square of Residual Error* between data matrix and its approximation $W \times H$ for various methods of W initialization are shown in Fig. 4.18. From Fig. 4.18, we observe that improper selection of initial values of W results in reaching local optima.

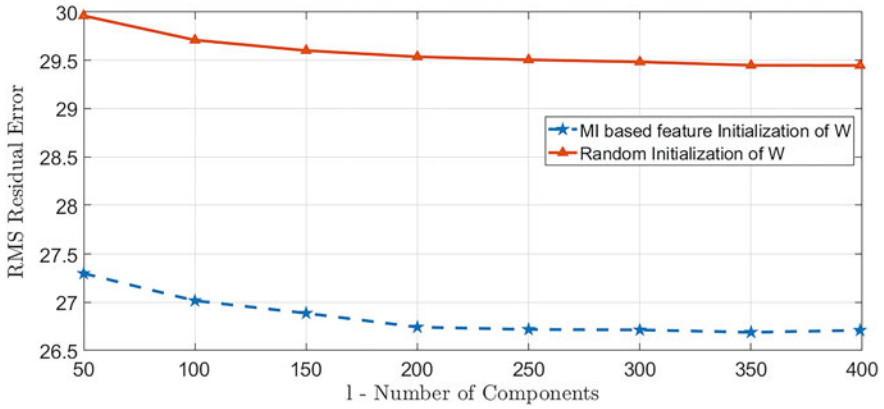

Fig. 4.18 RMS Error Comparison for various methods of initialization of W

4.6 Summary

In this chapter, we have examined various feature selection and linear feature extraction techniques in terms of their applicability. The feature selection schemes include:

1. **Filter methods:** We have considered distance, correlation and mutual information (MI) based methods. We have observed that the MI based scheme is good to reduce the dimensionality of large dimensional data sets.
2. **Wrapper methods:** We have used subsets of PCs to represent the data and $KNNC$ is used in classification in the process of selecting features using a GA.
3. **Embedded methods:** It is shown using the Iris data set that Decision Tree classifier can automatically select a subset of the features as a part of classification.

Subsequently, we have considered linear feature extraction schemes. These include:

1. PC based,
2. Random Projections based, and
3. NMF based:

It is observed that the top PCs may not be the best for representing the data used in classification. So, it is not correct to use the top few PCs for discrimination or classification. Random projections provide a good alternative to deal with high-dimensional data sets. NMF based reduction needs a better initialization scheme to avoid local-minima problems.

References

1. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Mining Knowl. Discov. **2**(2), 121–167 (1998)
2. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press (2001)
3. Rifkin, R.M.: Multiclass Classification. Lecture Notes, Spring08. MIT, USA (2008)
4. Witten, I.H., Frank, E., Hall, M.A.: Data Mining, 3rd edn. Morgan Kauffmann (2011)
5. Prakash, M., Murty, M.N.: A genetic approach for selection of (near-) optimal subsets of principal components for discrimination. Pattern Recogn. Lett. **16**, 781–787. Elsevier (1995)
6. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: NIPS'00: Proceedings of the 13^{th} International Conference on Neural Information Processing Systems, pp. 535–541 (2000)
7. MNIST Dats Set: https://www.tensorflow.org/datasets/catalog/mnist
8. ORL Face Dats Set: https://www.kaggle.com/datasets/tavarez/the-orl-database-for-training-and-testing
9. Scikit-Machine Learning in Python: https://scikit-learn.org/stable/

Chapter 5
Non-linear Schemes for Representation

Abstract In this chapter we deal with various nonlinear feature extraction schemes. In nonlinear feature extraction, the extracted features may be viewed as nonlinear combinations of the originally given features. Two popular neural network architectures employed are self-organizing map (SOM) and autoencoder (AE). We examine both of them in this chapter.

5.1 Introduction

We have studied different schemes for representation in the earlier chapters. These include:

- *Feature Selection:* Here, we select a subset of size l from the given set of $L(> l)$ features.
- *Linear Feature Extraction:* Under this category we represent each pattern using a collection of $l(< L)$ features. Each of these l features is obtained by using an appropriate linear combination of the given L features.

In this chapter, we will deal with extracting features that may be viewed as nonlinear combinations of the given L features.

5.2 Optimization Schemes for Representation

Let X_i and X_j be two L-dimensional vectors. Let P_i and P_j be the respective vectors in the l-dimensional space. Let d_{ij} be the euclidean distance between P_i and P_j and let d^*_{ij} be the distance between X_i and X_j. The idea behind the optimization is to minimize a function that captures the difference between d_{ij} and d^*_{ij} over all the pairs of patterns.

M. N. Murty, M. Avinash, *Representation in Machine Learning*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-981-19-7908-8_5

The specific function used by the popular Sammon's mapping algorithm is

$$\frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}.$$

In other words, this nonlinear mapping algorithm employs a nonlinear optimization algorithm that starts with an initial representation and keeps on updating by using a gradient descent approach. The idea behind the choice of the objective function is to preserve some structure of the data, given in high-dimensional space, after mapping the data to a lower-dimensional space.

If X is an L-dimensional vector and $l = 2$, then one possible initial representation of P, corresponding to X, in the two-dimensional space is given by

$$P = \begin{pmatrix} P_1 \\ P_2 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{L} X_i \\ \sum_{i=1}^{L} X_i^2 \end{pmatrix}.$$

This initial representation in the l-dimensional space ($l = 2$ here) is updated iteratively till some stopping/termination criterion is satisfied.

5.3 Visualization

1. Representation using Faces: Here, a pattern is represented using a face for easy visualization. It may be explained using the face shown in Fig. 5.1.

   A three-dimensional pattern ($L = 3$) is represented as a face in the figure. The first feature f_1 is used to depict the height of the face. The second feature f_2 is used to represent the width of the face and feature f_3 depicts the length of the mouth. Note that it is possible to incorporate more features. For example, f_4 can be used to capture the size of the eyes; f_5 for the length of the nose; f_6 for the distance between the eyes; f_7 for the distance between the eyes and the nose; f_8 for the distance between the nose and mouth, and so on. In addition one can use other parts like ears to represent more features.

   So, such a representation of patterns using faces can help the users in discriminating between patterns from different classes.
2. Self-Organizing Map (SOM): Kohonen's SOM is the most popular neural network model to represent a collection of L-dimensional vectors in a two- or three-dimensional space. Consider the simple two-dimensional network shown in Fig. 5.2.

   This network may be viewed as having the input and output layers. The input layer is an L-dimensional layer. An L-dimensional vector X is presented at the input layer. The L components of X are $x_1, x_2, \cdots, x_L$ as shown in the figure.

   The output layer typically is a two-dimensional grid of $M \times N$ cells. in the figure the values of M and N are 7 and 10 respectively. Every cell is connected

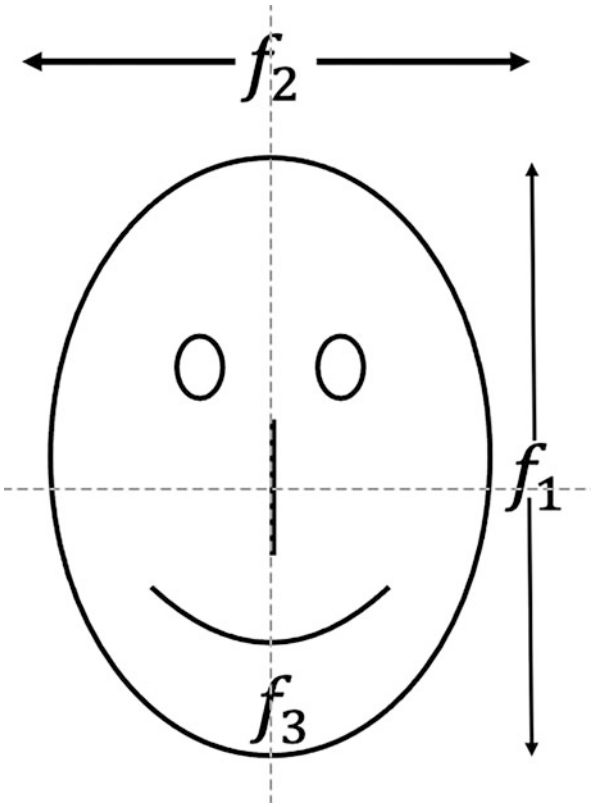

Fig. 5.1 An example face image representation of a pattern

to the input layer with L weights. For example, if the top leftmost cell is numbered as 1, then the input x_j is linked to this cell with a weight $W_{j,1}$ for $j = 1, 2, \cdots, L$. Similarly ith cell (shown here using the third row from the top and the sixth column from the left) is connected using a collection of L weights. For example, the Lth input node is connected using the weight $W_{L,i}$ as shown in the figure. We have shown weights connected to 3 out of 70 cells for the sake of clarity in the figure.

So, if there are L inputs and $M \times N$ cells in the output layer, then there will be $M \times N \times L$ weights to train. These weights are initially chosen randomly and are updated based on the assignment, of data presented, to the cells. For example, if a data vector X^p is more similar to the weight vector, $W(j) = (W_{1,j}, W_{2,j}, \cdots, W_{L,j})$, then X^P is associated with the jth cell and the weight vector, $W(j)$, associated with the winner cell (jth cell) is updated to align more with X^p. Such an update ensures that when X^p is presented next time it will match better with the weight vector $W(j)$.

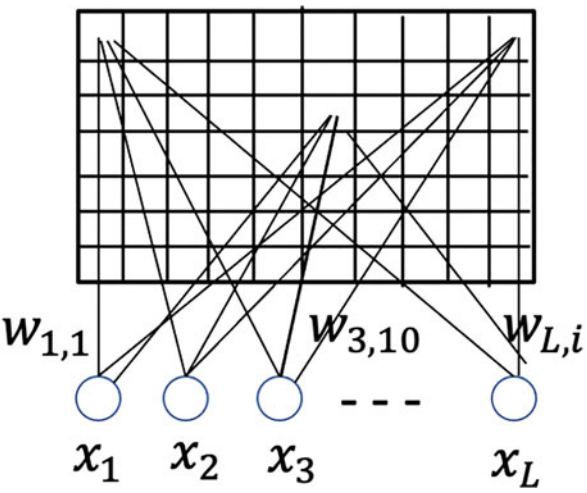

Fig. 5.2 Kohonen's SOM

Typically, the input data will be a collection of n data vectors, $X^1, X^2, \cdots, X^n$. These n data vectors are presented in a sequence at the input layer and the weight vector associated with the winning neuron is updated after assigning each pattern. These input vectors are repeatedly presented at the input layer multiple times (epochs). It is continued till either there are no updates to the weight vectors in an epoch or till a pre-specified number of epochs is completed. This may be viewed as associating each of the data vectors with a cell that is matching better in terms of its current weight vector.

It is possible to view the entire activity as clustering the n patterns into clusters corresponding to the output cells. There are several variants that can be used:

- In the procedure specified, weights of only the winning cell/neuron are updated. It is possible to update the weights associated with the neurons located within a window around the winning cell. Different window types were specified.
- It is popularly used to cluster unlabeled patterns. However, it can be used to cluster labelled patterns also.
- It may be viewed as mapping L-dimensional vectors into a two-dimensional map. It is shown to achieve the mapping so that topological properties of the data are preserved.

Consider the first 250 patterns of class label 1 and 7 in the MNIST Test Data set. We set the *default* seed to the random generator, using the command *rng default*. We used **MATLAB**, *selforgmap* function with the following parameters

- *dimensions* = [12, 12]
- Number of training steps for initial covering of the input space, *coverSteps* = 100

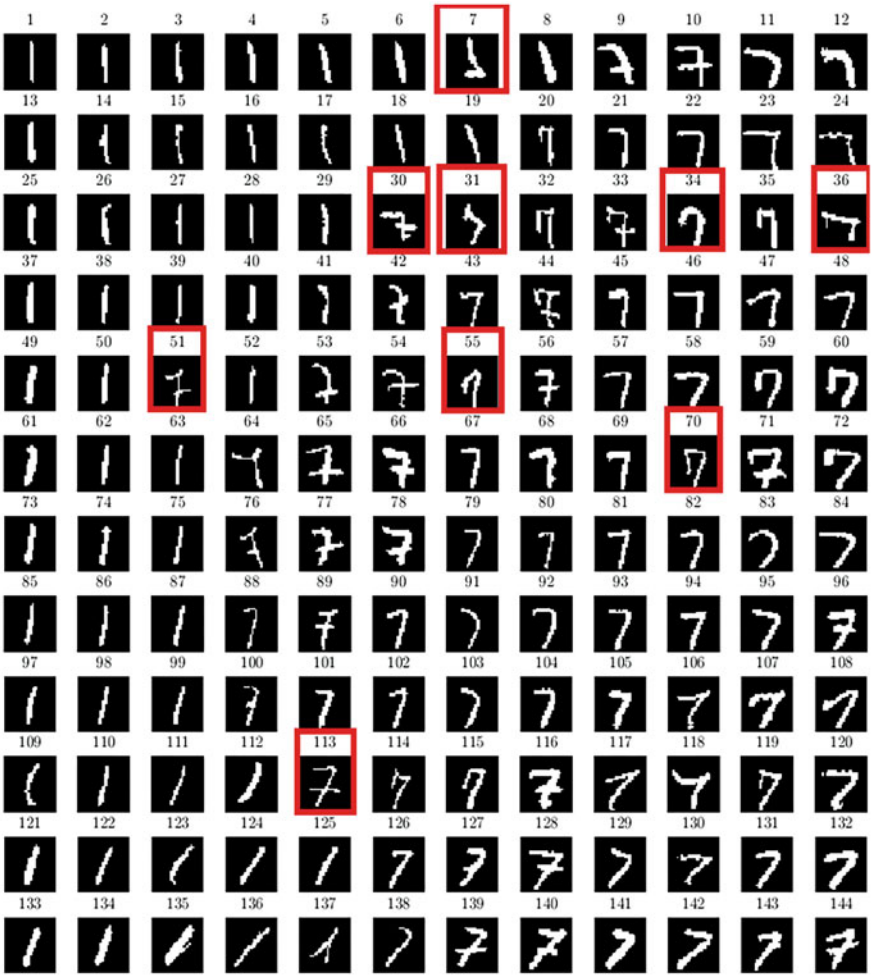

Fig. 5.3 SOM for first 250 patterns in classes labelled 1 and 7 in MNIST Test dataset

- Initial neighborhood size, *initNeighbor* = 3
- Layer topology function (default = 'hextop')
- Neuron distance function (default = 'linkdist')

The obtained SOM is shown in Fig. 5.3. The 144 grid/neurons (12 × 12) are numbered on the top in Fig. 5.3. At each grid point, the patterns which are falling in the grid, are averaged (For example, if three patterns fall in a grid, those three images are added and divided by 3, to get the average 28 × 28 image/pattern). The averaged pattern is converted into a binary vector (if pixel value greater than 127, it is replaced by 1 else replaced by 0). We observe the following in Fig. 5.3,

- In the last three rows, (grid numbered 109 to 144) the patterns are mostly inclined to the right.
- Pattern in grid numbers 4–12 and 16–19 are mostly inclined to the left.
- Patterns which are marked in red box are actually outlier patterns in the actual data set, not resulted from the addition of patterns falling in their respective grids.
- There is a clear transition from 1 to 7, as we move from left to right.
- It is able to capture the characteristics of different ones and sevens and clusters them together.

3. T-Distributed Stochastic Neighbor Embedding (T-SNE) Plot :
   This is a very popular visualization tool that employs a nonlinear mapping scheme to reduce the dimensionality of the data from l to 2. Given the data in the l-dimensional space, it estimates a probability structure over pairs of points. The probability is larger if the corresponding pair of points are similar and it is smaller if the two points are dissimilar. Similarly it computes the probability structure in the reduced dimensional space. KL-divergence between these two probability distributions is minimized to achieve the mapping.

   Let us consider all the test patterns of MNIST data set. Figure 5.4 shows T-SNE plot for all the test patterns of MNIST data set with *Perplexity* = 30 (for many values of *Perplexity* we got similar plots) and *Euclidean Distance* as distance measure using **MATLAB**-*tsne* function. We can observe that there are many overlaps between many class labels. For example, patterns from classes 1 and 7 show significant overlaps, similarly classes labelled 3 and 8 show overlaps. Let us consider only the patterns of classes labelled 1 and 7 respectively. Figure 5.5 shows the T-SNE for class labelled 1 and 7. We can clearly observe the overlapping patterns. From Fig. 5.5, we picked some overlapping patterns and displayed their corresponding high dimensional image values in Fig. 5.6.

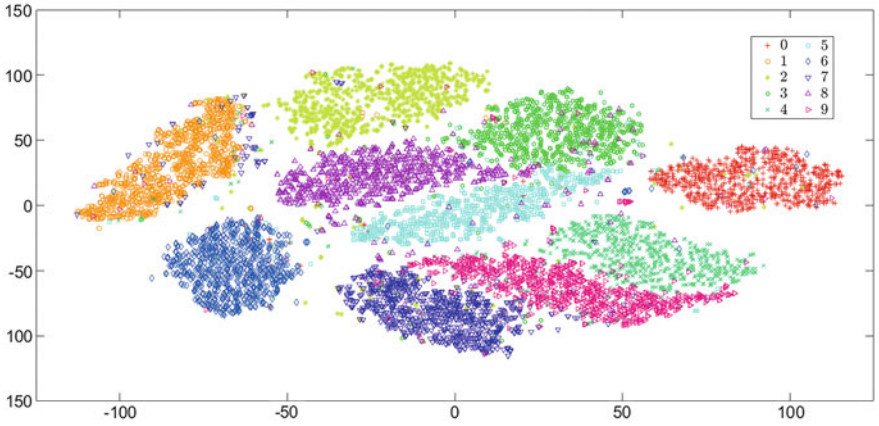

Fig. 5.4 T-SNE for all test patterns in MNIST Data set

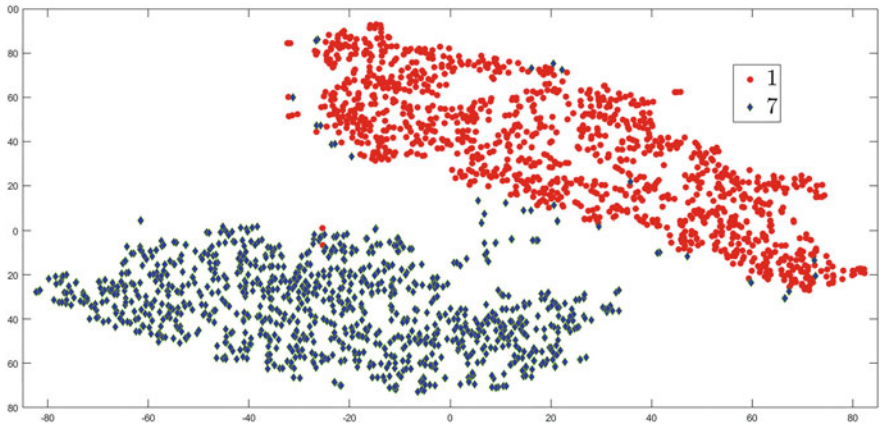

Fig. 5.5 T-SNE for patterns labelled 7 and 1 in MNIST Data-set

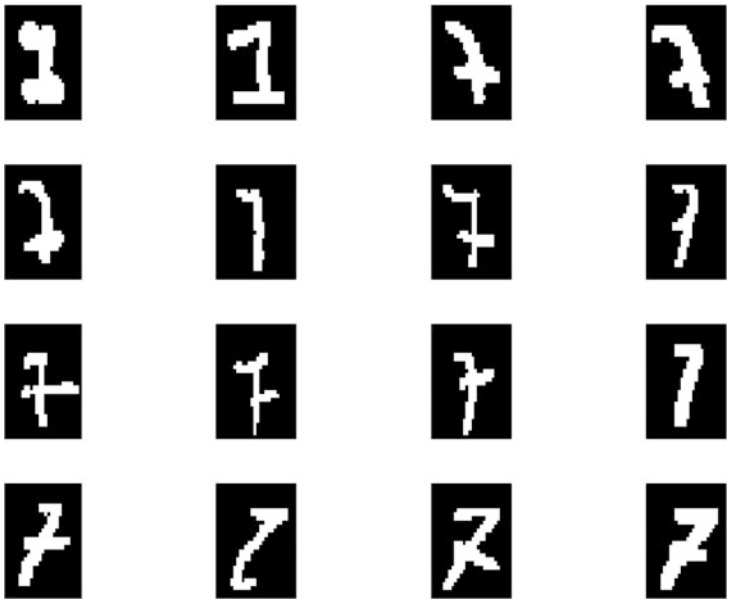

Fig. 5.6 Some overlapping pattern of class labelled 7 and 1 in MNIST Data set

In Fig. 5.6, the first two patterns (in position (1,1) and (1,2) respectively), are actually labelled as pattern 1, but they occur in the region of label 7 and similarly other patterns with actual label as 7, but in the T-SNE plot in Fig. 5.5, they occur in the region of label 1 respectively.

We took six misplaced patterns in the T-SNE plot in Fig. 5.5. For these six patterns, we found three nearest neighbors in the lower dimensional embedding and we plotted their corresponding higher dimensional input in Fig. 5.7. In

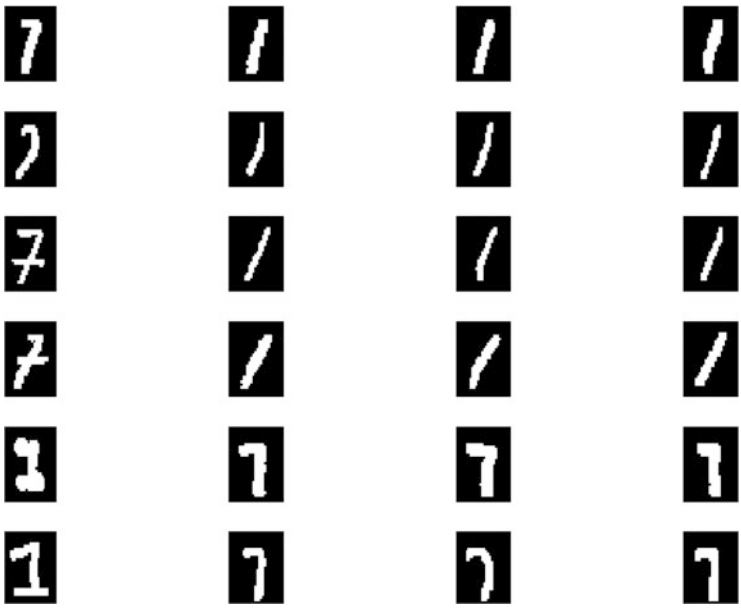

Fig. 5.7 Three nearest neighbours of misplaced patterns

Fig. 5.7, the first column represents the misplaced patterns and other columns correspond to their three nearest neighbors respectively.

For the same misplaced patterns (column 1 in Fig. 5.7), we find all the three nearest neighbors belonging to the same class. For example, for the pattern at position (1,1) in Fig. 5.7, the actual label of the pattern is label 7, we find the corresponding neighbors among all the patterns belonging to class label 7 and we plotted them as the first row in Fig. 5.8 and similarly for other patterns.

The T-SNE plots for test patterns labelled 7 and 1 show misplacement of patterns. So, in order to measure the effect of misplacement, in the lower dimensional embedding space a *KNN classifier* with nine nearest neighbors is used. For each pattern in the lower dimensional space, nine nearest neighbors are found and using *Majority vote* the label of the pattern is predicted from its nine nearest neighbors. If the predicted label is the actual label of the pattern, then that pattern is not misplaced in the T-SNE plot, else, the pattern is declared as a misplaced pattern. By the above mentioned criterion, the train accuracy (A) may give the measure of misplacement by the T-SNE plot.

Various L-norms with parameter r, are used to compute the dissimilarity for the T-SNE plot. The Training Accuracy (A) for various values of parameter is tabulated in Table 5.1. From the table, we observe that for fractional norms and lower values of parameter r, there are many misplaced patterns in the T-SNE plots and as the value of r increase, the misplacement is getting reduced, which is reflected in terms of the increase in the training accuracy A.

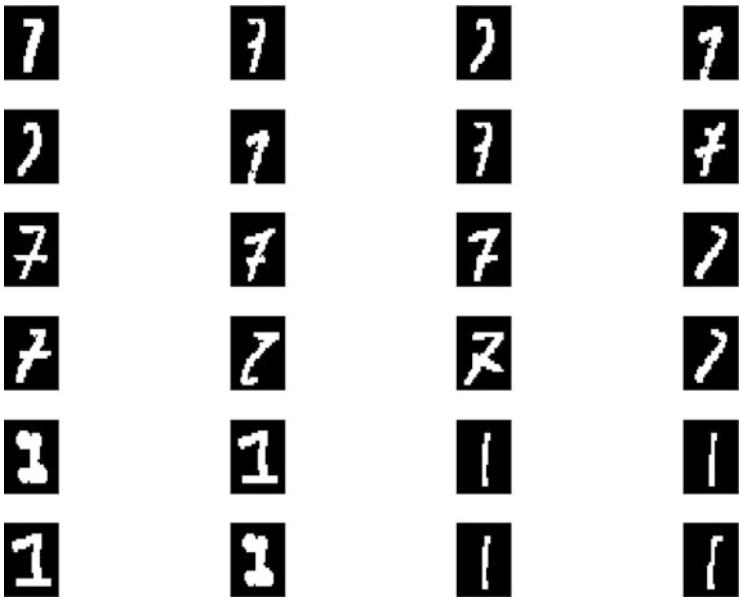

Fig. 5.8 Three nearest neighbours in the same class of misplaced patterns

Table 5.1 T-SNE Error analysis for different values of r using Minkowski distance

r	A (%)
0.2	97.1798
0.5	97.8733
1	98.1045
2	98.5668
50	98.8904
60	98.9829
80	98.9367
90	99.0754
95	99.2141
100	99.0754
∞	98.1507

From Figs. 5.9 and 5.10, we observe the following:

- For lower values of r, the distance measure in higher dimensions is large, but in the lower dimensional space, distance is small and the clusters are tightly packed. As the value of r increases, we can observe the reduction in within cluster distance and the clusters are becoming loose.

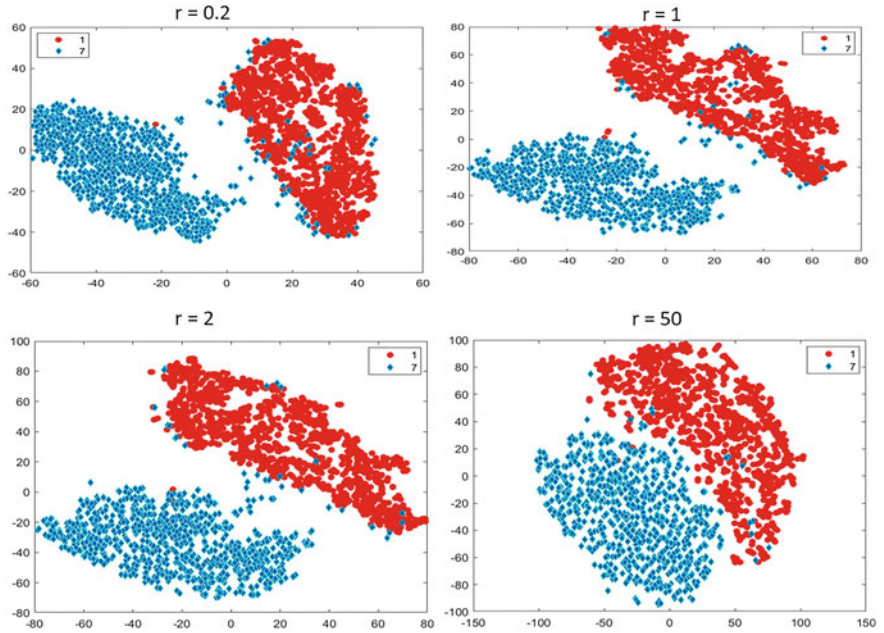

Fig. 5.9 T-SNE for various L-Norms

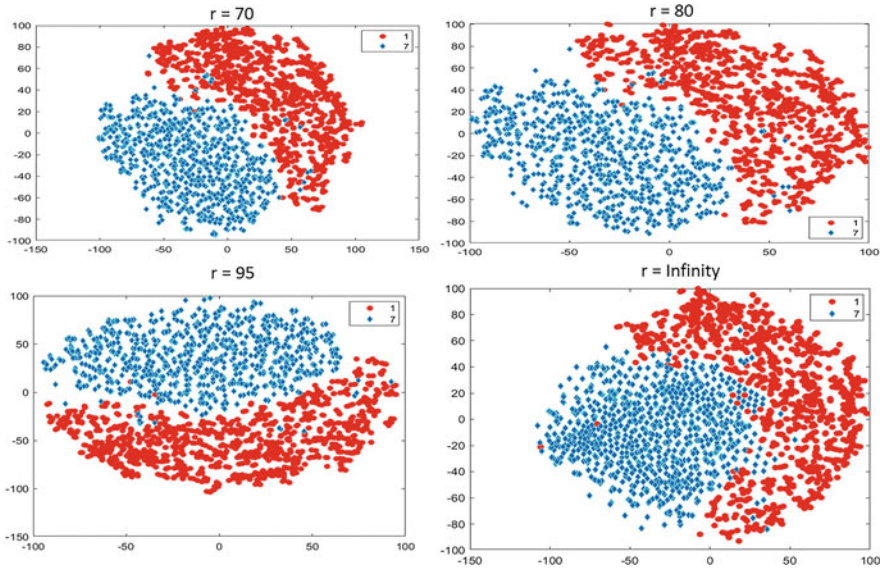

Fig. 5.10 T-SNE for various L-Norms

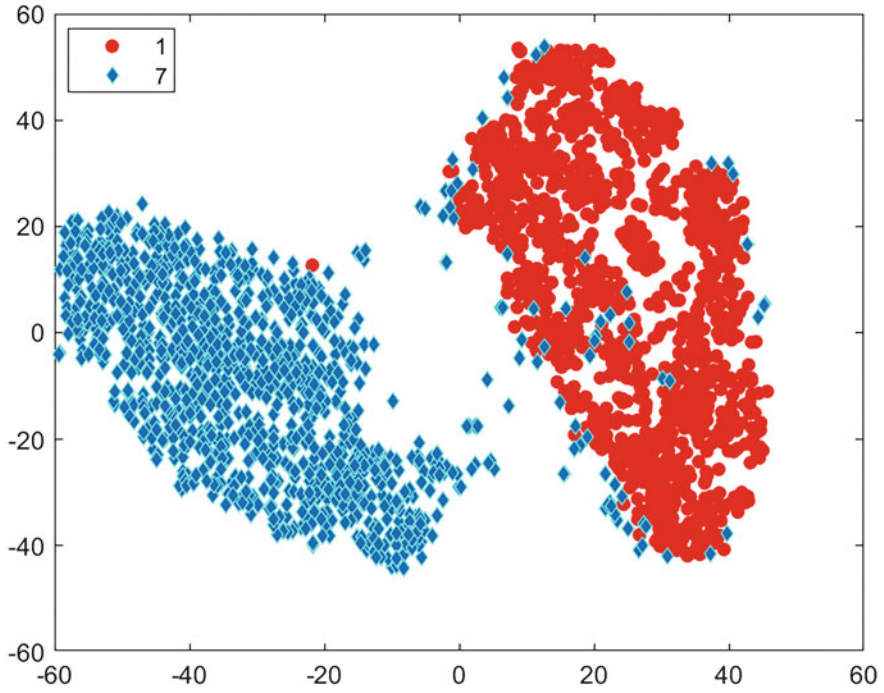

Fig. 5.11 T-SNE Plot with L-Norm r = 0.2

- As the value of r increases, the clear separation between the two clusters is reducing.
- The penetration of pattern labelled 7 (blue) into the region of 1 (red) is getting reduced as the parameter r increases.
- The occurrence of pattern labelled 7 (blue) in far extreme end in the regions of 1 (red) is getting reduced as the value of r increases.
- As r increases, the misplacement of patterns is getting reduced.

Consider Figs. 5.11 and 5.12 for further insights

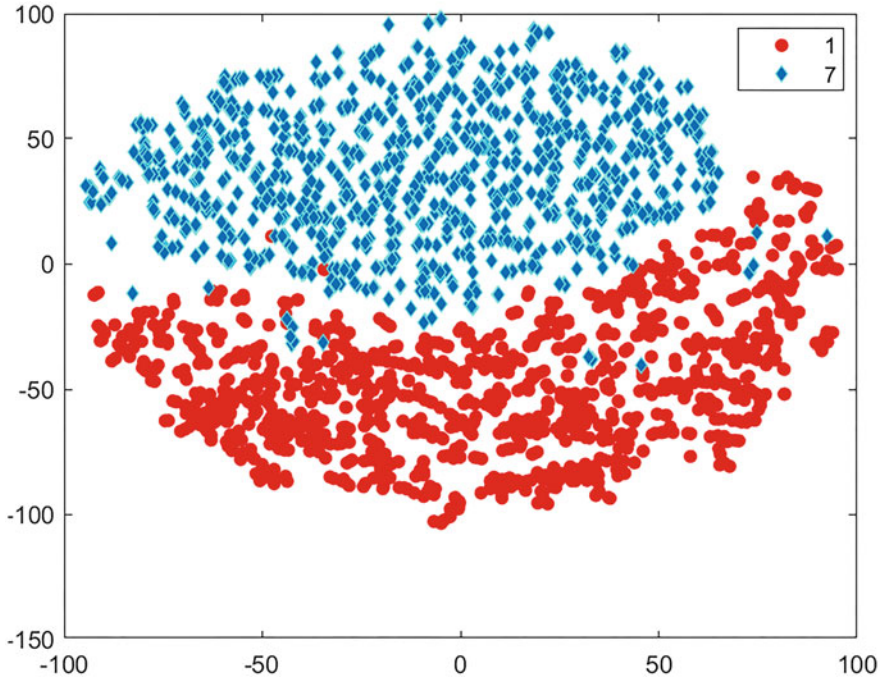

Fig. 5.12 T-SNE Plot with L-Norm r = 95

5.4 Autoencoders for Representation

let us consider the ORL face Data set. The Data set consists of 400 face images. The entire set of 400 faces is divided into 320 Train patterns and 80 test patterns, such that for each class label 8 patterns are assigned to the Training set and two patterns are assigned to the test set respectively. The image dimension is 112 by 92. Each image is converted into a row vector. So, the number of features of the Data set is $10{,}304 = 112 * 92$. For the sake of simplicity and computation capacity constrains, we consider a single hidden layer *Autoencoder* and employed the following procedure for feature extraction and classification

- Each image is reshaped as 100×100 (instead of original dimension 112×92).
- Each reshaped image is divided into four quadrants, $\{C1, C2, C3, C4\}$ of dimension 50×50 each.
- Four different single hidden layer *Autoencoders*$\{A1, A2, A3, A4\}$ are used; each one extracts features from each quadrant,$\{C1, C2, C3, C4\}$ of the image. The number of features is the number of neurons in the single hidden layer.
- The four sets of features $\{f1, f2, f3, f4\}$ extracted from the four different *Autoencoders*$\{A1, A2, A3, A4\}$, are used for classification using a *softmax*

classifier for each *Autoencoder*. (Note: Each *Autoencoder* and *softmax*-Pair are trained separately for their respective quadrants).
- For a given test pattern, its four quadrants are given as input to their corresponding *Autoencoder-softmax Pair*$\{A1, A2, A3, A4\}$ and four class labels are predicted one for each quadrant.
- Using *Majority voting* among the four predicted class labels from the four individual *Autoencoder-softmax-pairs*, the class label for the test pattern is predicted.
- The *Test Accuracy* of each quadrant and for the entire image (predicted using *Majority voting*) is noted and the experiments are repeated for various values of the parameters of the *Autoencoder*.

Parameters of *Autoencoder*

Let us start with a brief introduction of a single layer *Autoencoder*. Consider the following notation,

- A single input vector $\mathbf{x}$ with dimension $\mathbf{L}$, $\mathbf{x} \in \mathbb{R}^L$.
- The total number of training patterns, $\mathbf{N}$
- The entire training data matrix $\mathbf{X} \in \mathbb{R}^{L \times N}$ of size $L \times N$, where patterns are stacked column wise.
- The first layer of *Autoencoder* has $\mathbf{l}$, hidden units (neurons).
- The weight matrix $W^{(p)}$ and bias vector $b^{(p)}$ at level (layer) $\mathbf{p}$. (where $\mathbf{p}$ represents the number of levels/layers. We have considered only a single layer *Autoencoder*; so, we have two levels, level = 1 encoder level and level = 2 decoder level respectively).
- Let $\mathbf{z} \in \mathbb{R}^l$ be the latent representation of single input vector $\mathbf{x}$, with dimension $l \times 1$
- Let $\mathbf{Z} \in \mathbb{R}^{l \times N}$ be the latent representation of the entire input training data matrix, of size $l \times N$ (a matrix of size $l \times N$).
- Let $h(.)$ represent the non linear function at the encoder and decoder level which introduces the non linearity.

At the first level of *Autoencoder*, which is the *encoder* level, the single input vector $x \in \mathbb{R}^L$ is converted into its lower dimensional latent representation $z \in \mathbb{R}^l$ (where $l < L$) as follows:

$$z^{(1)} = h\left(W^{(1)} \cdot x + b^{(1)}\right)$$

$$z^{(1)} = h\left(\begin{bmatrix} w^{(1)}_{1\,1} & w^{(1)}_{1\,2} & \dots & w^{(1)}_{1\,L} \\ \vdots & \ddots & \ddots & \vdots \\ w^{(1)}_{l\,1} & \dots & \dots & w^{(1)}_{l\,L} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_L \end{bmatrix} + \begin{bmatrix} b^{(1)}_1 \\ \vdots \\ b^{(1)}_l \end{bmatrix}\right)$$

where superscript (1) represents the first level (Encoder level) and $W^{(1)}$ and $b^{(1)}$ are the corresponding weight vector and bias terms at the encoder level.

At the second level, which is the decoder level, an estimate of the input vector $\hat{\mathrm{x}}$, is obtained as follows:

$$\hat{\mathrm{x}} = h\left(W^{(2)} \cdot z^{(1)} + b^{(2)}\right)$$

$$\hat{\mathrm{x}} = h\left(\begin{bmatrix} w_{1\,1}^{(2)} & w_{1\,2}^{(2)} & \dots & w_{1\,l}^{(2)} \\ \vdots & \ddots & \ddots & \vdots \\ w_{L\,1}^{(2)} & \dots & \dots & w_{L\,l}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} z_1^{(1)} \\ \vdots \\ z_l^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ \vdots \\ b_L^{(2)} \end{bmatrix}\right)$$

where superscript (2) represents the second level (decoder level) and $W^{(2)}$ and $b^{(2)}$ are the corresponding weight vector and bias terms at the decoder level. So, the *Autoencoder* takes an input vector of dimension **L** and reduces to a lower dimensional latent vector of dimension **l**; then it tries to reconstruct the estimate of input vector from the lower dimensional latent representation.

For illustration, consider an example single hidden layer *Autoencoder* in Fig. 5.13. In Fig. 5.13, we observe the following,

- The input nodes represents the input vector **x** with dimension $\mathbf{L} = 8$.
- The middle hidden layer represents the latent vector z with dimension $\mathbf{l} = 4$
- The output nodes represents the reconstructed estimate of input vector $\hat{x}$ with dimension $\mathbf{L} = 8$.

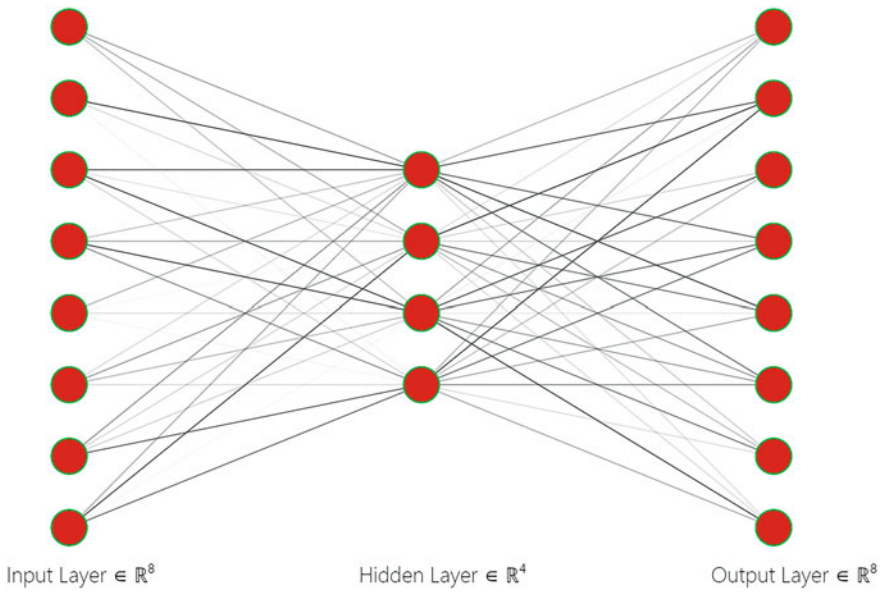

Fig. 5.13 An example single hidden layer *Autoencoder*

- The edge weights are randomly selected and their thickness is propositional to their weights

Now, consider the following parameters, which are tuned for the experiments.

1. *Hidden Layer Size (l)*:
   The number of hidden units (neurons) in the single layer *Autoencoder* is tuned by this parameter, *Hidden layer size, l*. As the number of hidden units increases the number of weights and bias parameters increases at both the encoder and decoder levels respectively.
2. *Encoder-Decoder Transfer Function [TF]*:
   We used non linear transfer functions for the encoder and decoder as follows:
   - *Logistic sigmoid function,* TF_1

$$h(k) = \frac{1}{1 + e^{-k}}$$

   - *Positive saturating linear transfer function,* TF_2

$$h(k) = \begin{cases} 0 & \text{if } k \leq 0 \\ k & \text{if } 0 < k < 1 \\ 1 & \text{if } k \geq 1 \end{cases}$$

   The *Autoencoder* reconstructs the input from the decoder level; so, the range of the input must match the range of the decoder transfer function. Hence, the inputs are scaled to match the range of decoder transfer function accordingly.
3. *Maximum number of training epochs [MaxEpochs]*
   The maximum number of training epochs/iterations is fixed as 500 because of computational constraints.
4. *Cost Function, [CF]*:
   The *Autoencoder* tries to reduce the *Mean Square Error* between the input vector $x \subset \mathbb{R}^L$ and the reconstructed estimate $\hat{x} \subset \mathbb{R}^L$; we can also add magnitude constraint to the weights and Sparsity constraint by employing a suitable Cost Function as follows:

$$CF = MSE + \lambda \cdot \Omega_{weights} + \beta \cdot \Omega_{Sparsity}$$

5. *Mean Square Error [MSE]*

$$MSE = \frac{1}{N} \sum_{i}^{N} \sum_{j}^{L} \left(x_{ij} - \hat{x}_{ij}\right)$$

Where N represents the total number of Training patterns. L is the dimensionality of the training patterns and $\hat{x}$ represents the reconstructed estimate of the actual training pattern x by the *Autoencoder*.

6. L_2 *Weight Regularization* $\Omega_{weights}$:
   As the *Autoencoder* reduces the error between the actual training patterns and their estimates, we need to have the following:

   - a mechanism to control the magnitude of the weights.
   - to prevent the *Autoencoder* from remembering the training patterns and over fitting.

   Hence the following weight regularization term is used:

$$\Omega_{weights} = \sum_{l}^{P} \sum_{i} \sum_{j} \left(W_{i,j}^{(l)} \right)^2$$

   where P is the total number of levels of the *Autoencoder*, as we have considered a single layer *Autoencoder*, we have $P = 2$. These are *Encoder level* and *Decoder level*. The number of elements of the Weight matrix W at each level varies. The *Encoder* level weight matrix $W^{(1)}$ has a dimension $L \times l$ and *Decoder* level weight matrix $W^{(2)}$ has a dimension $l \times L$. These weight matrices can be modified to incorporate the bias terms too.
7. L_2 *Weight Regularization term* λ:
   Parameter which varies the contribution of L_2 Weight Regularization term $\Omega_{weights}$ in the Cost function.
8. *Sparsity Regularization*, $\Omega_{Sparsity}$:
   We can control the firing of a neuron by controlling its average output activation value. If the average output activation value of a neuron is low, it means that the neuron is responding (firing) only to the features present in a subset of training examples. The average output activation value of ith neuron, $\hat{\rho}_i$, is:

$$\hat{\rho}_i = \frac{1}{N} \sum_{j=1}^{N} h(W_i^{(p)} \cdot x_j + b_i^{(p)})$$

   where,

   - $h(\cdot)$ represents the Nonlinear *Encoder-Decoder Activation function*
   - $W_i^{(p)}$ represents the ith row of the weight matrix of level (p)
   - b_i represents the ith element of the bias term at level (p)

   So, we are calculating the average output of the activation function of ith neuron for all the training examples. We need to add a *Kullback–Leibler divergence* (KL) term to the Cost Function which results in a large value whenever the actual average output activation value of ith neuron deviates from the desired value ρ, by which we can introduce Sparsity in the latent representation.

$$\Omega_{Sparsity} = \sum_i KL\left(\rho||\rho_i\right) \quad = \sum_i \rho \cdot \log\left(\frac{\rho}{\rho_i}\right) + (1-\rho) \cdot \log\frac{1-\rho}{1-\rho_i}$$

9. *Sparsity Regularization term, β*:
   Parameter which varies the contribution of Sparsity Regularization term $\Omega_{Sparsity}$ in the Cost function.

The above mentioned parameters are varied and the results are tabulated in the next section.

5.5 Experimental Results: ORL Data Set

Let the notation for test accuracy of the four quadrants be TA1, TA2, TA3, and TA4 and final test accuracy for the entire image by *Majority voting* be TA. We have varied the parameter λ, Transfer function and the results are tabulated in Table 5.2. From Table 5.2, we observe that *Positive saturating linear function* (TF_2) performs well when compared to *Logistic sigmoid function* (TF_1) in terms of final test accuracy TA.

Let us tune the parameter ρ, Transfer function and perform the experiment. The results are tabulated in Table 5.3. From Table 5.3, we again observe that TF_2 outperforms TF_1 in terms of final test accuracy TA.

Now, we alter the number of hidden neurons l along with ρ and Transfer function. The results are tabulated in Table 5.4.

From the results we can observe the following:

- When the inputs are scaled properly, the *Positive saturating linear function* (TF_2) performs well when compared to *Logistic sigmoid function* (TF_1) in terms of final test accuracy TA.
- Increasing the number of dimensions, l, in the latent representation may not necessarily result in a better representation.

Table 5.2 λ Experimentation for ORL data set

TF	λ	β	ρ	l	TA1	TA2	TA3	TA4	TA
1	0.1	1	0.75	1000	0.875	0.925	0.912	0.887	0.95
1	0.01	1	0.75	1000	0.875	0.925	0.912	0.9	0.95
1	0.001	1	0.75	1000	0.875	0.912	0.9	0.887	0.938
1	0.0001	1	0.75	1000	0.863	0.887	0.887	0.9	0.938
2	0.1	1	0.75	1000	0.838	0.875	0.9	0.912	0.975
2	0.01	1	0.75	1000	0.85	0.887	0.887	0.9	0.963
2	0.001	1	0.75	1000	0.863	0.875	0.9	0.925	0.975
2	0.0001	1	0.75	1000	0.838	0.863	0.887	0.9	0.975

Table 5.3 ρ Experimentation for ORL data set

TF	λ	β	ρ	l	TA1	TA2	TA3	TA4	TA
1	0.01	1	0.05	1000	0.85	0.912	0.9	0.887	0.938
1	0.01	1	0.1	1000	0.863	0.912	0.9	0.887	0.938
1	0.01	1	0.2	1000	0.863	0.912	0.9	0.887	0.938
1	0.01	1	0.5	1000	0.875	0.912	0.887	0.9	0.938
1	0.01	1	0.75	1000	0.875	0.912	0.9	0.887	0.95
2	0.01	1	0.05	1000	0.85	0.875	0.887	0.912	0.975
2	0.01	1	0.1	1000	0.85	0.875	0.9	0.9	0.975
2	0.01	1	0.2	1000	0.85	0.863	0.9	0.875	0.95
2	0.01	1	0.5	1000	0.825	0.863	0.9	0.9	0.975
2	0.01	1	0.75	1000	0.85	0.887	0.887	0.9	0.963

Table 5.4 Hidden layer p—Experimentation for ORL data set

TF	λ	β	ρ	l	TA1	TA2	TA3	TA4	TA
1	0.01	1	0.75	600	0.85	0.887	0.9	0.9	0.925
1	0.01	1	0.75	800	0.863	0.887	0.887	0.887	0.95
1	0.01	1	0.75	1000	0.875	0.912	0.9	0.887	0.95
1	0.01	1	0.75	1200	0.863	0.925	0.9	0.9	0.938
1	0.01	1	0.75	1500	0.875	0.925	0.9	0.887	0.95
2	0.01	1	0.5	600	0.875	0.875	0.887	0.925	0.988
2	0.01	1	0.5	800	0.85	0.863	0.887	0.887	0.95
2	0.01	1	0.5	1000	0.825	0.863	0.9	0.9	0.975
2	0.01	1	0.5	1200	0.85	0.863	0.887	0.875	0.95
2	0.01	1	0.5	1500	0.838	0.863	0.9	0.875	0.95

- By proper tuning of parameters, we can achieve better representation even with a smaller value of l, dimension of latent representation. For Example, in Table 5.4, for $l = 600$, we got better Test accuracy TA, than higher values of l.

5.6 Experimental Results: MNIST Data Set

Let us consider the patterns of classes labelled 7 and 9 from the *MNIST* data set. Each pattern is a 28×28 image, whose pixel values range from 0 to 255. Every pattern is converted into a binary row vector of dimension 1×784, such that a pixel value greater than 127 is replaced by 1 and pixel value less than or equal to 127 is substituted by 0. These row converted binary patterns are stacked to form the training set of size 12,214 and test set of size 2037 respectively. Each pattern is a binary vector of length 784.

A single hidden layer *Autoencoder* is used as a feature extractor and *Softmax* layer is used as the classifier. The experiment mentioned in Sect. 5.4 is repeated for

the MNIST Data set. As the dimension of the input vector is less (784 in the case of MNIST Data set), we have directly used the input vectors without pursuing any divide and conquer strategy as mentioned in Sect. 5.4.

The input image x (28×28 image as 1×784 vector) is fed into the *Autoencoder* and the reconstructed estimate $\hat{x}$ is obtained. The *Structural Similarity Index, SSIM* between the input image x and its estimate $\hat{x}$ is calculated for all the training and test patterns and its values are averaged. The *SSIM* is a good measure of reconstructing capability of the *Autoencoder*. The output parameters like *Test Accuracy*, averaged *SSIM* of training set and averaged *SSIM* of test set are tabulated for different experiments. The parameter *MaxEpochs* = 200 is fixed for all the experiments.

Fixing the parameter l, all other parameters are varied and the results are tabulated in Table 5.5. Similarly, fixing all other parameters as constant, parameter l is varied and the results are tabulated in Table 5.6.

Table 5.5 ρ Experimentation for MNIST data set

TF	λ	β	ρ	l	TA (%)	Train set average SSIM	Test set average SSIM
1	0.01	1	0.05	100	98.72361	0.471434	0.47556
1	0.01	1	0.1	100	98.52725	0.480052	0.483841
1	0.01	1	0.2	100	98.67452	0.495541	0.499057
1	0.01	1	0.5	100	98.72361	0.386587	0.389973
1	0.01	1	0.8	100	98.8218	0.362159	0.365737
1	0.01	1	0.95	100	98.7727	0.326204	0.329523
1	0.001	1	0.05	100	99.01816	0.566713	0.565715
1	0.001	1	0.1	100	99.06726	0.482927	0.479977
1	0.001	1	0.2	100	99.01816	0.473056	0.469951
1	0.001	1	0.5	100	98.7727	0.579093	0.5807
1	0.001	1	0.8	100	98.91998	0.632808	0.634788
1	0.001	1	0.95	100	98.91998	0.599323	0.601631
2	0.01	1	0.05	100	99.01816	0.819241	0.820994
2	0.01	1	0.1	100	99.06726	0.850315	0.852342
2	0.01	1	0.2	100	98.96907	0.880027	0.881356
2	0.01	1	0.5	100	98.67452	0.885968	0.885967
2	0.01	1	0.8	100	98.7727	0.844688	0.843249
2	0.01	1	0.95	100	99.01816	0.733145	0.731035
2	0.001	1	0.05	100	98.96907	0.828054	0.829163
2	0.001	1	0.1	100	99.11635	0.85617	0.858287
2	0.001	1	0.2	100	98.91998	0.890277	0.890806
2	0.001	1	0.5	100	98.67452	0.891055	0.890241
2	0.001	1	0.8	100	98.8218	0.84853	0.845882
2	0.001	1	0.95	100	98.96907	0.753597	0.751189

Table 5.6 Hidden layer l—Experimentation for MNIST data set

TF	λ	β	ρ	l	TA (%)	Train set average SSIM	Test set average SSIM
1	0.01	1	0.1	50	98.52725	0.471213	0.475623
1	0.01	1	0.1	250	98.67452	0.525936	0.529374
1	0.01	1	0.1	500	98.87089	0.529627	0.53246
1	0.01	1	0.1	600	98.67452	0.554979	0.557877
1	0.001	1	0.1	50	98.96907	0.477558	0.477894
1	0.001	1	0.1	250	98.96907	0.539827	0.535098
1	0.001	1	0.1	500	98.67452	0.560211	0.553098
1	0.001	1	0.1	600	98.33088	0.580901	0.573728
2	0.01	1	0.1	50	98.91998	0.805224	0.810006
2	0.01	1	0.1	250	99.01816	0.893267	0.89295
2	0.01	1	0.1	500	98.13451	0.920173	0.917866
2	0.01	1	0.1	600	97.79087	0.924482	0.922242
2	0.01	1	0.1	50	98.87089	0.815872	0.8199
2	0.01	1	0.1	250	98.91998	0.917327	0.914497
2	0.01	1	0.1	500	98.42906	0.91927	0.917235
2	0.01	1	0.1	600	97.34904	0.942189	0.937611

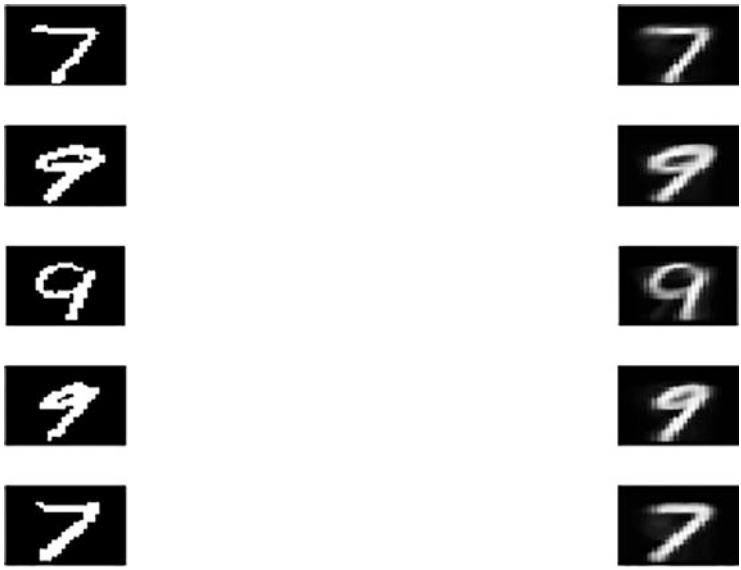

Fig. 5.14 Reconstructed images using Logistic sigmoid function, TF_1

From Tables 5.5 and 5.6, we can again observe that the *Positive saturating linear function* (TF_2) performs well when compared to *Logistic sigmoid function* (TF_1) in terms of *SSIM*.

Figure 5.14 shows the reconstructed images of some patterns, where the *Autoencoder* used TF_1 with $\rho = 0.5$ and $\lambda = 0.01$. Similarly, Fig. 5.15 shows the

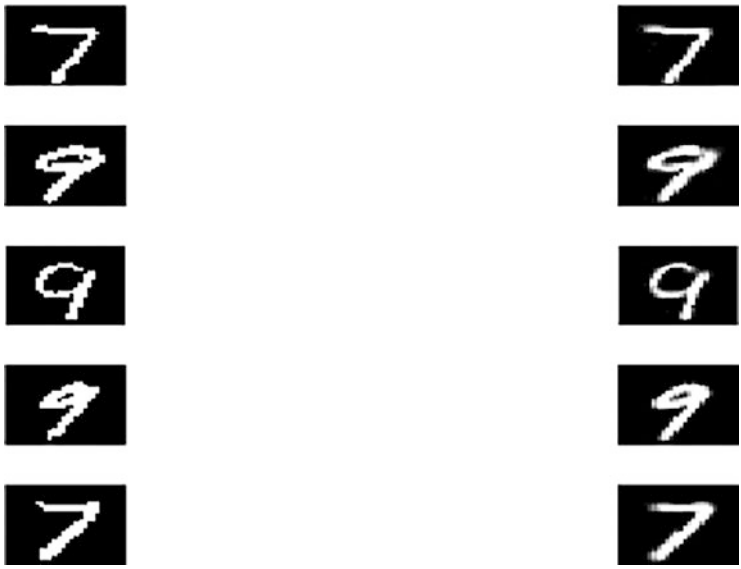

Fig. 5.15 Positive saturating linear transfer function, TF_2

reconstructed images of some patterns using Positive saturating linear transfer function, TF_2. From Figs. 5.14 and 5.15, we observe a better reconstruction capability of Positive saturating linear transfer function (Close to the famous ReLu function).

In order to understand how the weight matrix in the *Encoder* level represents certain features and how the parameter ρ affects the weight matrix, consider Figs. 5.16 and 5.17.

For an *Autoencoder* of 100 hidden neurons, each neuron is represented in a 10×10 grid in Figs. 5.16 and 5.17. Each grid unit represents the 784 weights for each neuron as an 28×28 image. For $\rho = 0.05$ in Fig. 5.16, the average output activation values is low so the weight images 28×28 in each grid is nearer to black region representing smaller values and the neurons will fire only for certain features.

Similarly, for $\rho = 0.95$, in Fig. 5.17, the weights are having higher values nearer to the white regions and no sharp characteristics when compared to Fig. 5.16.

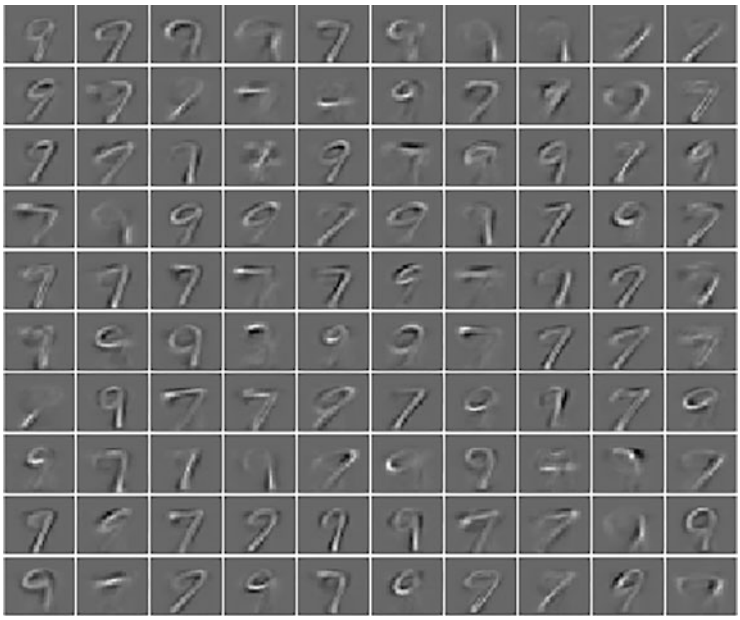

Fig. 5.16 Weight matrix with $\rho = 0.05$

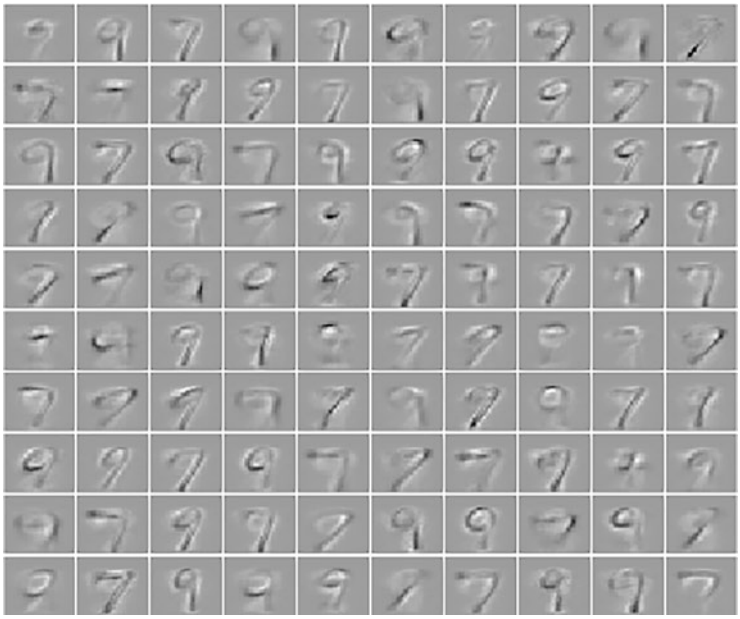

Fig. 5.17 Weight matrix with $\rho = 0.95$

5.7 Summary

In this chapter, we have considered nonlinear feature extraction. Here each extracted feature may be viewed as a nonlinear combination of the given L features. We have discussed and examined three different schemes; they are useful in both dimensionality reduction and visualization. Specifically, we have considered

1. Self-organizing neural network: It is popularly called the self-organizing map (SOM). It can be used to map high-dimensional data vectors into a two or a three dimensional space. It has some nice properties:
   - It preserves the topological structure present in the data.
   - It can be viewed as a neural network architecture for clustering. The clusters are perceived based on the self-organization property exploited by the training algorithm.
   - It is helpful in visualizing the data in a two-dimensional space that is easy for human consumption.
2. T-Distributed Stochastic Neighbor Embedding (T-SNE) Plot: It is popularly known as the t-SNE plot. It is a nonlinear technique for dimensionality reduction and visualization of high-dimensional data. Some of its features are:
   - It is the most popular embedding technique for visualization of high-dimensional data.
   - It is popular in analysing the embedding techniques employed in complex networks including social networks, chemical and biological networks.
3. Autoencoder: It is also a neural network architecture primarily used in nonlinear dimensionality reduction. Some of its features are:
   - Based on the type of activation function used by various neurons, it can work like a nonlinear feature extractor (nonlinear activation) or like PCA to perform linear feature extraction (linear activation).
   - It is found to be an apt tool when the classes are not linearly separable in the original feature space.

In each of the above cases, we have conducted experiments and reported the results using two benchmark data sets, the MNIST handwritten digits data set and the ORL face data set.

References

1. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Mining Knowl. Discov. **2**(2), 121–167 (1998)
2. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press (2001)
3. Rifkin, R.M.: Multiclass Classification, Lecture Notes, Spring08. MIT, USA (2008)
4. Witten, I.H., Frank, E., Hall, M.A.: Data Mining, 3rd edn. Morgan Kauffmann (2011)

5. Kohonen, T., Honkela, T.: Kohonen network. Scholarpedia **2**(1), 1568 (2007)
6. van der Maaten, L.J.P., Hinton, G.E.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)

Chapter 6
Conclusions

Abstract In this chapter, we provide a brief summary of various representation schemes discussed in the book. Then we provide conclusions for each of the dimensionality reduction techniques discussed and used in our experiments.

It is well-known that representation forms the most important step in machine learning. It can have impact on both the computational resources required to implement the ML algorithm and also the performance measures of the algorithm. Even though there are other schemes for representing patterns/data items, vector space representations have been the state-of-the-art. Here, each item is represented as a vector in a multi-dimensional space. Based on this view, a collection of items is represented as a data matrix of size $n \times L$ where there are n training patterns and L features.

Dimensionality reduction is an important tool for achieving a lower-dimensional representation of the data. This is important especially in situations where distance between a pair of patterns is computed by the ML algorithm. The nearest neighbour algorithms are essentially dependent on computing distance between a test pattern and the training patterns to find out the nearest neighbour. It is well-known that the nearest neighbour classifier and its variants suffer in high-dimensional spaces because of the concentration effect. Related problems are discussed in Chap. 3. Solutions to this problem in the form of using Fractional norms to compute inter-pattern distances and locality sensitive hashing are examined in the same chapter. Extensive experiments were carried out the results are analysed in Chap. 3.

There are three different approaches for dimensionality reduction. They are:

1. Feature selection schemes: Here a subset of l features is selected from the given set of L features where L » l. There are different approaches to feature selection including filter methods, wrapper methods, and embedded methods. We have examined various feature selection schemes in Chap. 4. It is observed that mutual information based feature selection is ideally suited to deal with high-dimensional data sets; it is popularly used in information retrieval where the text documents are represented as vectors in a very high-dimensional space. Among the contenders, feature selection based on correlation also works well. Further,

M. N. Murty, M. Avinash, *Representation in Machine Learning*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-981-19-7908-8_6

it is observed that classifiers like decision tree classifier are good for embedded methods as it can select relevant features in the process of building the classifier.

2. Feature extraction schemes: Here a set of l features are derived out of the given set of L features where $l << L$. Let $f_1.f_2, \cdots, f_L$ be the given L features. The extracted features may be viewed as $f^1, f^2, \cdots, f^l$, where

$$f^i = g_i(f_1, f_2, \cdots, f_L) \; for \; i = 1, 2, \cdots, l,$$

where g_i is a function of the given L features. Based on the nature of $g_i's$ we can divide the feature extraction schemes into either linear or nonlinear schemes.

   a. Linear Feature extraction: Here,

$$f^i = c_{1i} f_1 + c_{2i} f_2 + \cdots + c_{Li} f_l, \;\; for \; i = 1, 2, \cdots, l.$$

   That means each extracted feature is a linear combination of the given L features. It is possible to view this kind of feature extraction as a consequence of matrix multiplication. We can represent the operation as a matrix vector multiplication as follows:

$$\begin{pmatrix} f^1 \\ f^2 \\ \vdots \\ f^l \end{pmatrix} = \begin{bmatrix} c_{11}, & c_{21}, & \cdots, & c_{L1} \\ c_{12}, & c_{22}, & \cdots, & c_{L2} \\ & & \vdots & \\ c_{1L}, & c_{2L}, & \cdots, & c_{LL} \end{bmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_L \end{pmatrix}$$

   We have considered principal components, random projections, and features extracted using NMF under this category. We have discussed in detail how these linear schemes work on some benchmark data sets in Chap. 4. A major misconception in terms of the use of principal components in classification is that the first few PCs corresponding to the leading eigenvalues are good for classification. We have demonstrated, in Chap. 4, how such a notion can fail in classification/discrimination.

   b. Nonlinear Feature Extraction: Here the extracted feature f^i is a nonlinear combination of the given features, $f_1, f_2, \cdots, f_L$. One may appreciate the power of nonlinear features using a simple example of xor (exclusive or) computation. We can abstract the functionality of xor using the following truth table. It is True (1) only when exactly one of either a or b is True (1), False (0) otherwise. It is not possible to represent xor as a linear combination of a and b. It means that it is not possible to express $xor(a, b) = \alpha a + \beta b$ where α and β. are any real numbers. However, if we have a nonlinear feature ab in addition to a and b, then we can represent $xor(a, b)$ as a linear combination as given by

$$xor(a, b) = a + b - 2ab.$$

Table 6.1 Truth table of $xor(a, b)$

a	b	$xor(a, b)$
0	0	0
0	1	1
1	0	1
1	1	0

(Please check by plugging in the values for a, b, and ab (Table 6.1).)

So, nonlinear features can be highly expressive and can help us in realizing simpler classifiers by using nonlinear feature extraction. We have considered three popular nonlinear feature extraction schemes in Chap. 5. These are SOM, t-SNE features and Autoencoders. These schemes are useful in both dimensionality reduction and visualization. We have conducted experimental studies using these schemes on benchmark data sets. We demonstrated how SOM preserves the topological properties and how various parameter values affect the working of autoencoders.

References

1. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Mining Knowl. Discov. **2**(2), 121–167 (1998)
2. Schölkopf, B., Smola, A. J.: Learning with Kernels. MIT Press (2001)
3. Rifkin, R.M.: Multiclass Classification, Lecture Notes, Spring08. MIT, USA (2008)
4. Witten, I.H., Frank, E., Hall, M.A.: Data Mining, 3rd edn. Morgan Kauffmann (2011)
5. Murty, M.N., Susheela devi, V.: Introduction to Pattern Recognition and Machine Learning. World Scientific/IISc Press (2015)

Index

M. N. Murty, M. Avinash, *Representation in Machine Learning*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-981-19-7908-8