Information Fusion and Data Science
Series Editor: Henry Leung

Haitao Zhao
Zhihui Lai
Henry Leung
Xianyi Zhang

Feature Learning and Understanding

Algorithms and Applications

Springer

Information Fusion and Data Science

Series Editor

Henry Leung, University of Calgary, Calgary, AB, Canada

This book series provides a forum to systematically summarize recent developments, discoveries and progress on multi-sensor, multi-source/multi-level data and information fusion along with its connection to data-enabled science. Emphasis is also placed on fundamental theories, algorithms and real-world applications of massive data as well as information processing, analysis, fusion and knowledge generation.

The aim of this book series is to provide the most up-to-date research results and tutorial materials on current topics in this growing field as well as to stimulate further research interest by transmitting the knowledge to the next generation of scientists and engineers in the corresponding fields. The target audiences are graduate students, academic scientists as well as researchers in industry and government, related to computational sciences and engineering, complex systems and artificial intelligence. Formats suitable for the series are contributed volumes, monographs and lecture notes.

More information about this series at http://www.springer.com/series/15462

Haitao Zhao • Zhihui Lai • Henry Leung
Xianyi Zhang

Feature Learning and Understanding

Algorithms and Applications

Springer

Haitao Zhao
East China University of Science
and Technology
Shanghai, Shanghai, China

Henry Leung
Department of Electrical
& Computer Engineering
University of Calgary
Calgary, AB, Canada

Zhihui Lai
Shenzhen University
Shenzhen, China

Xianyi Zhang
East China University of Science
and Technology
Shanghai, Shanghai, China

ISSN 2510-1528 ISSN 2510-1536 (electronic)
Information Fusion and Data Science
ISBN 978-3-030-40793-3 ISBN 978-3-030-40794-0 (eBook)
https://doi.org/10.1007/978-3-030-40794-0

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Big data is a big opportunity for us in this era. The digital transformation leaves no organization untouched, and all companies have to derive value and insight from data. The theories and applications of mining-specific information and extracting knowledge from massive data are becoming more and more important.

Features are data representatives that can be much easier to understand in the context of a problem, and feature learning is the process of using domain knowledge and special techniques to transform raw data into features. Feature learning is an essential procedure for data analysis and machine intelligence. Readers who are not sure what feature learning is could turn to Chap. 1 or some other works.

This book covers the essential concepts and strategies within traditional and cutting-edge feature learning methods. Each feature learning method has its own dedicated chapter that explains how it is theoretically derived and shows how it is implemented for real-world applications through case studies.

In this book, readers can find not only traditional feature learning methods, such as principal component analysis, linear discriminant analysis, geometrical structure-based methods, and kernel-based learning methods but also advanced feature learning methods, such as sparse learning, low-rank decomposition, tensor-based feature extraction, and deep learning-based feature learning. Some relevant codes and experimental results uploaded at https://github.com/haitaozhao/flu allow readers to reproduce the experiments easily by themselves.

The intended audience of this book are nonspecialists whose needs cannot be satisfied by the black box. It seems that these people will be chiefly interested in the methods themselves – how they are derived and how they can be adapted to particular problems. The aim of this book is to bring the reader to the point where he/she can go to the research literature to augment what is in this book.

Readers are assumed to have a knowledge of elementary analysis and linear algebra and a reasonable amount of programming experience. Strictly speaking, this book is not a textbook. The guiding principle has been that if something is worth explaining, it is worth explaining clearly. This is necessarily restricted to the

scope of the book, but the authors hope the selected feature learning methods in this book will give the reader a good basis for further study or research.

Many people have contributed to this book. We would like to thank the following colleagues and friends for their help: Qianqian Wang, Yuqi Li, Yuru Chen, Ziyan Liao, Zhengwei Hu, Jingchao Peng, and Yaobin Xu. Their suggestions and remarks have contributed significantly to improvements.

Shanghai, China Haitao Zhao
January 2020

Contents

Notation

Numbers and Sets

x	A scalar
$\boldsymbol{x}$	A vector
$\boldsymbol{X}$	A matrix
$\mathbb{X}$	A tensor
x_{ij}	The element of matrix $\boldsymbol{X}$ at row i and column j
$\boldsymbol{X}_r$	The matrices formed by the first r columns of $\boldsymbol{X}$
$\cap$	The intersection of two sets
$\cup$	The union of two sets
$\cup_{i=1}^{n}$	The union of n sets
$\forall$	For all
$\mathcal{R}$	The set of all real numbers
$\mathcal{R}^n$	The set of n-dimensional column vectors of real numbers
$\mathcal{R}^{m\times n}$	The set of matrices of real numbers with m rows and n columns
$\{\boldsymbol{x}_i\}_{i=1}^{n}$	The set consists of samples $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$

Operators and Functions

$(\cdot)^T$	The transpose of a vector or a matrix
$\|\boldsymbol{x}\|_0$	The ℓ_0-norm of a vector $\boldsymbol{x}$
$\|\boldsymbol{x}\|(\|\boldsymbol{x}\|_2)$	The ℓ_2-norm of a vector $\boldsymbol{x}$
$\|\boldsymbol{X}\|_F$	The Frobenius norm of a matrix $\boldsymbol{X}$
$\|\boldsymbol{X}\|_{2,1}$	The $\ell_{2,1}$-norm of a matrix $\boldsymbol{X}$
$\|\boldsymbol{X}\|_*$	The nuclear norm of a matrix $\boldsymbol{X}$: the sum of all its singular values
$\boldsymbol{X}^{-1}$	The inverse of matrix $\boldsymbol{X}$
$\boldsymbol{X}^{+}$	The generalized inverse of matrix $\boldsymbol{X}$
$\det(\boldsymbol{X})$	The determinant of matrix $\boldsymbol{X}$

$\text{rank}(\boldsymbol{X})$	The rank of a matrix $\boldsymbol{X}$: the dimension of the vector space generated (or spanned) by its columns
$\text{trace}(\boldsymbol{X})$	The trace of a matrix $\boldsymbol{X}$: the sum of all its diagonal entries
$\otimes$	The Kronecker product
$\odot$	The Hadamard (elementwise) product
$\lfloor \cdot \rfloor$	Rounding a number to the next smaller integer
$\sum$	Series addition
$\prod$	Series multiplication
$\int f(x)dx$	The indefinite integral of f with respect to x
$\int_a^b f(x)dx$	The definite integral of f from a to b with respect to x
$f(\cdot)$	A function
$\ln(\cdot)$	The natural logarithm
$\exp(\cdot)$	The exponential function

Derivative and Gradient

$\frac{dy}{dx}$	The derivative of y with respect to x
$\frac{\partial y}{\partial x}$	The partial derivative of y with respect to x
∇f	The gradient of function f

Probability and Statistics

$p(x)$	The probability density function of a random variable x
$p(x\|y)$	The conditional probability of the random variable y given x
$P(n)$	The probability distribution of a discrete variable n
$E(\boldsymbol{x})$	The expectation of $\boldsymbol{x}$
$\text{cov}(\boldsymbol{x})$	The covariance matrix of a random vector $\boldsymbol{x}$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	The normal (Gaussian) distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Graph Theory and Symbols

$\mathcal{G}$	A graph
$\mathcal{V}$	The set of nodes
$\mathcal{E}$	The set of edges connecting the points
$\measuredangle$	The Laplace Beltrami operator
$\ll$	a $\ll$ b means a is much less-than b
$\in$	$s \in S$ means s is an element of set S

Chapter 1
A Gentle Introduction to Feature Learning

1.1 Introduction

Data is the new oil, and artificial intelligence and machine learning are powerful tools to convert data into information that fuels humans (Sarkar et al. 2018; Ray 2019). In this age, it is hardly a surprise that big data and machine learning are some of the top words. Due to the rapid development of computer technology, information technology, and internet technology, the cost of data storage has dropped significantly. Over the last 2 years alone, 90% of the data in the world has been generated (Einolander 2019). This data comes from everywhere: sensors that gather shopper information, posts to social media sites, digital pictures and videos, purchase transactions, and cell phone GPS signals, to name a few. The theory and application of mining-specific information and extracting knowledge from massive data are becoming more and more important (Yu and Yan 2018).

Feature learning and understanding is a crucial part of machine learning. The main challenge for today's enterprises and organizations is how to use various techniques to learn from data and use valuable information and insights to make better decisions (Wu et al. 2013). Feature learning can build derived values (features), eliminate irrelevant, redundant, or noisy data, accelerate the speed of data processing, and improve the understanding of the data (Guyon and Elisseeff 2006).

Since the 1970s, feature learning has become an essential research content of machine learning and pattern recognition, and widely used in text analysis (Yan et al. 2009), speech processing (Sivaram et al. 2010), image/video recognition (Jiang et al. 2019; KalaiSelvi et al. 2014), biological information analysis (Hanna and Zaki 2015), and other fields. In this book, we will focus on the theory and methods of feature learning widely studied in machine learning and pattern recognition.

As shown in Fig. 1.1, a typical learning or recognition system consists of two parts: data preparation and algorithm. Generally, the algorithm alone is not intelligent enough to process the raw data from the real world and discover the latent patterns to train the system. Hence, we need data preparation to discover better data

H. Zhao et al., *Feature Learning and Understanding*, Information Fusion and Data Science, https://doi.org/10.1007/978-3-030-40794-0_1

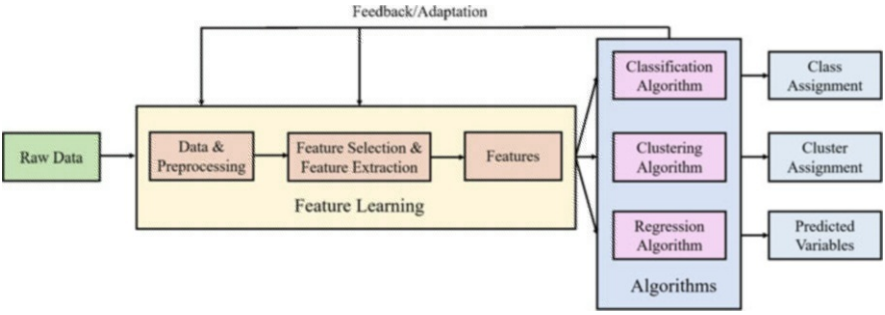

Fig. 1.1 Machine learning pipeline

representatives, or more precisely, features. And the process of using domain knowledge and special techniques to transform raw data into features is called feature learning, which is our area of focus in this book.

Feature is a broad concept in machine learning and pattern recognition. Any attribute could be a feature, as long as it is useful to the following algorithms. The purpose of a feature, other than being an attribute, would be much easier to understand in the context of a problem. A feature is a characteristic that might help when solving the problem.

The initial set of raw data can be noisy and redundant. A fundamental procedure in many applications of machine learning is to select a subset of features or construct a new and informative feature set to facilitate learning and improve generalization and interpretability. When we talk about *feature learning*, it comes with a set of techniques to transform raw data into features.

There are no fixed rules for feature learning. Generally, we generate features from real-world data through domain knowledge and mathematical transformation. With the development of deep learning, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) and other neural network architectures are widely used in feature learning tasks (Bengio et al. 2012; Tetko et al. 2019; Ibrahim and Al-Jumaily 2016). Deep-learning-based methods learn features directly from the original data, which is often called end-to-end feature learning (Goodfellow et al. 2016). As shown in Fig. 1.2, traditional feature learning basically describes data with features without relying on explicit algorithms and then apply a learning algorithm. However, end-to-end feature learning attempts to learn features and optimize the following algorithm together (Wen et al. 2016).

Real-world data such as images, video, and sensor data is usually high dimensional, noisy, and redundant for machine learning algorithms. Building informative, representative or discriminative features is a crucial step for the algorithms in machine learning tasks.

In this chapter, we will go over the life cycle of data, including data collection, data cleaning, data sampling, and feature learning.

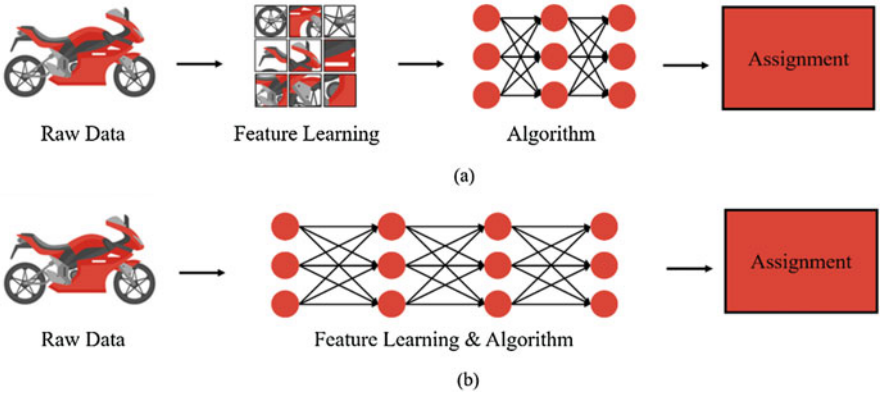

Fig. 1.2 Traditional feature learning (**a**) and end-to-end feature learning (**b**)

1.2 Data and Preprocessing

Raw data refers to numbers or characters collected before being "cleaned" and corrected. Raw data needs to be corrected by eliminating outliers or obvious instruments or data input errors. Data processing is usually carried out in stages, and "processed data" in one stage can be considered as "raw data" in the next stage (Hand 2006; Wu et al. 2013; Kantardzic 2011; Drabas 2016).

1.2.1 Data Collection

Data collection is the process of gathering and measuring information from different sources. These data can be numerical (temperature, loan amount, customer retention rate), categorical (gender, color, highest degree earned), or even free text (doctor's notes or opinion surveys).

Predictive algorithms are only as good as the data from which they are built, so good data collection practices are crucial to developing high-performing algorithms. The data are expected to be error-free and contain relevant information for the task at hand.

There is no specific way to collect data. Deciding what data to use involves using a combination of domain knowledge and business constraints. And it takes a lot of trial and error. For different real-world problems, we can usually think about the following aspects:

- **How can we get the data**: Not all the useful data for the problem can be obtained from real life. For example, it is difficult to know the decision of the top management of a company in time, even if it is useful for predicting the market value of the company.

- **How quickly the data can be accessed online in real-time**: In practice, it is hard to pay for a long wait if the data is not available in time, even if the model is accurate. Therefore, we should try our best to collect data that is easily available.

1.2.2 Data Cleaning

Data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records and refers to identifying incomplete, incorrect, inaccurate, or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data (Wickham 2014). Incorrect or inconsistent data can lead to false conclusions and misdirected investments. Here are some common situations for data cleaning and their corresponding solutions (Idris 2016; Ilyas and Chu 2019).

- **Missing or Incomplete Data**

First, we should determine the range of missing values: calculate the proportion of missing values for each field and then formulate strategies according to the missing proportion and the importance, which we can see clearly in Fig. 1.3 (Graham 2009; Enders 2010).

Then we should remove unnecessary fields by deleting them directly. Here we strongly recommend to do a backup every step of the cleaning or to test the full amount of data on small-scale data. Otherwise, it will be regrettable to delete important data.

Next, we should fill the missing content. Here we provide three common ways to solve the problems. The first is to predict missing values with business knowledge or experience. The second is to fill in missing values with the same metric (mean, median, mode, etc.). The third is to fill in missing values with calculation results for

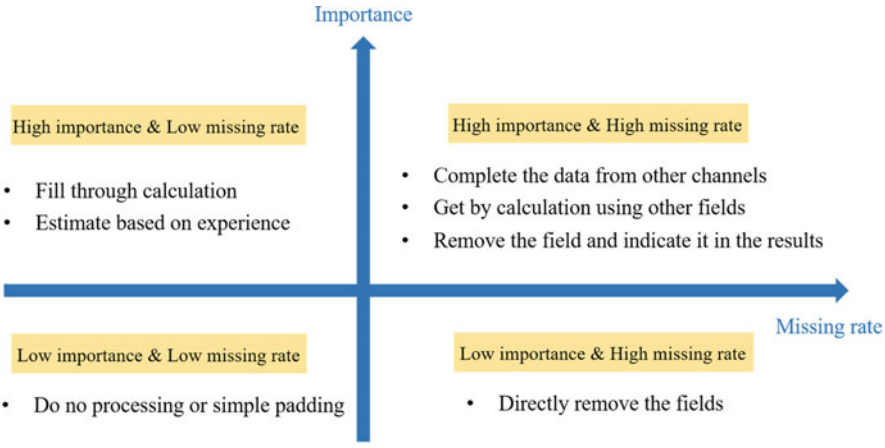

Fig. 1.3 Strategies for missing data based on the importance of the data and the missing rate

different metrics, like to get the date of birth from the ID card field (Bouza-Herrera 2013).

Finally, if some fields are very important and the missing rates are high, then we need to contact people or business personnel to see if there are other channels to get relevant data.

- **Content in Different Format**

Sometimes display formats such as time, date, value, and full-width are inconsistent, which is usually related to the input or integrating multi-source data. And we should process it in a consistent format (Dong and Liu 2018).

Sometimes there are characters in the content that should not exist, such as space at the beginning, end, and middle of the data. In this case, semi-manual methods are needed to identify problems and remove unwanted characters.

Sometimes the content does not match the content that should be, such as the place where the name should be filled now filled with the gender. The reason may be a small mistake when filling in the form, or the front end may not be verified, or the columns are not aligned when the data is imported. We can not simply delete them. Instead, we should identify the problem types in detail and give corresponding solutions (Idris 2016).

- **Logical Error**

The idea here is to remove some data that can be directly found problems using simple logical reasoning to prevent the analysis results from going wrong, usually about duplicate values and unreasonable values.

1.2.3 Data Sampling

Data sampling is a statistical analysis technique used to select, manipulate, and analyze a representative subset of data points to identify patterns and trends in the larger data set being examined. It enables predictive modelers with a small, manageable amount of data to build and run analytical models more quickly, while still producing accurate findings (Thompson 2012; Singh 2013). The common data sampling methods are as follows:

- Simple random sampling: Software is used to select subjects from the whole population randomly.
- Stratified sampling: Subsets of the data sets or population are created based on a common factor, and samples are randomly collected from each subgroup.
- Cluster sampling: The larger data set is divided into subsets (clusters) based on a defined factor, then a random sampling of clusters is analyzed.
- Systematic sampling: A sample is created by setting an interval at which to extract data from the larger population – for example, selecting every tenth row in a spreadsheet of 200 items to create a sample size of 20 rows to analyze.

In many cases, data sampling may result in an imbalance between positive and negative samples (Komori and Eguchi 2019). For example, to study a disease, the number of healthy people will be much larger than the number of sick people. It is important to solve the imbalance of positive and negative samples in data sampling. Usually, if the number of positive samples is much bigger than the negative samples, an effective way to approach this imbalance would be to subsample the positive class randomly. For example, if you have 10,000 positive samples and 100 negative samples, we might consider taking 100 samples out of 10,000 positive samples. The other way may consider taking the following aspects:

- Collect more negative samples if conditions permit
- Oversampling, such as mirroring and rotation in image augmentation
- Increase the weights of the negative samples in the total "loss" of the classification

1.2.4 Data Transformation

Data transformation plays a key role in big data analytics (Dong and Liu 2018). Data often exist in various forms, such as image, text, graph, sequence, and time-series. A common way to represent data objects for data analytics is to use vectors.

Raw data refers to some existing information or knowledge convenient for people to use. Usually, the data we have may not be simply numerical (i.e., age), but categorical (i.e., gender), time type (i.e., 9:30), text type (i.e., sentence) and so on. For different data types, there can be different data transformation methods.

- **Numerical variable**

A numerical variable can be continuous or discrete, generally expressed as a real value, like age, price, height, weight, and so on. Even though numerical data can be directly fed into a machine learning algorithm, important aspects including scale and distribution should still be focused on (Kuhn and Johnson 2019).

- **Standardizing/Normalizing**: Standardizing, which is usually (but not always) the same thing as normalizing means transforming a variable so that it has a mean of 0 and a standard deviation of 1. This is done by subtracting the mean from each value of a variable and then dividing by its standard deviation. Different variables may not have the same unit. Therefore, standardization is needed to eliminate the dimension influence among different variables and bring data into a common format that allows for comparative evaluation (Spivak and Brenner 2001). It should be noted that a new data point also needs to be standardized before testing.
- **Log Transformation**: Log transformation is a common process of taking a mathematical function and applying it to the data (Vittinghoff et al. 2006). Each variable x is replaced with $\log(x)$, where the base of the log is left up to the analyst. Log transformation is useful for compressing the y-axis when plotting

histograms, leading the visualization to be clearer. And it also de-emphasizes outliers and allows us to obtain a bell-shaped distribution potentially.

- **Discretization**: The use of continuous attributes requires large storage. A discretization technique is required to change the continuous value to discrete value (Wohlmuth 2012). It should be noted that the data distribution is often uneven by equidistant segmentation. In this case, the segmentation can be designed according to quantile.

- **Categorical variable**

Categorical variables represent characteristics such as a person's gender, marital status, hometown, etc. Categorical data can take on numerical values (such as "1" indicating male and "2" indicating female), but those numbers do not have mathematical meaning. Some tree-based algorithms can work with categorical data directly, but other machine learning algorithms require all input variables to be numerical.

One-hot encoding is widely used to deal with the categorical feature (Simonoff 2013). One-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0).

In general, we use one-hot if there is no logical relationship between different categories (such as male and female). Taking the "RGB color" variable as an example, there are three categories, and therefore, three binary variables are needed. A "1" value is placed in the binary variable for the color and "0" values for the other colors. So red can be written as "100", blue as "010", and green as "001".

One-hot coding can transform a categorical variable into points in Euclidean space. Moreover, there is no numerical relationship in one-hot coding.

- **Time/date variable**

In machine learning, there is often time type data, such as year, month, day, hour, minute, and so on. In practical application, time is a useful and important feature. We may find that some action is more probable on certain days of the week, or something happens around the same month every year. Two common approaches to deal with the date feature is to transform it into multiple attributes or into a difference between dates.

Time data can be regarded as either continuous data or discrete data. Take CTR (click-through rate) as an example: the time length of the user's single-page browsing time and the time interval between the last click and the current one are continuous data. While in the processing of evaluating the days of the week or the months of the year, the time variable can be viewed as discrete data.

- **Text variable**

Text variables include all the components of a story or an article that is not the main body of the text. These components include the table of contents, index, glossary, headings, bold words, sidebars, pictures and captions, and labeled diagrams. To deal with the unstructured data like text documents, the first challenge is the unpredictable nature of the syntax, format, and content of the documents, which

make it difficult to extract useful information for building models. The second challenge is to transform the textual representations into numerical representations that can be understood by machine learning algorithms.

- **Bag-of-words model**: A bag-of-words is a representation of text that describes the occurrence of words within a document (Dong and Liu 2018; Zheng and Casari 2018). It involves the vocabulary of known words and the frequencies of the known words. For example, if there is a text document including "John likes to watch movies" and "Mary likes movies too", then a list is constructed as follows for each document: "John", "likes", "to", "watch", "movies", "Mary", "likes", "movies", "too". The feature can be written as BoW1 = {"John":1, "likes":2, "to":1, "watch":1, "movies":2, "Mary":1, "too":1}. Each key is the word, and each value is the number of occurrences of that word in the given text document. If phrases or collection of words rather than a single word are taken into account, it is usually called the "Bag-of N-Grams model".
- **TF-IDF model**: Tf-idf stands for term frequency-inverse document frequency, which is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus (Dong and Liu 2018; Zheng and Casari 2018). Typically, the tf-idf weight is composed of two terms: the first computes the normalized term frequency (TF), aka, the number of times a word appears in a document, divided by the total number of words in that document; the second term is the inverse document frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Hence, the product of TF $\times$ IDF (TF$-$IDF) of a word gives a product of how frequent this word is in the document multiplied by how unique the word is w.r.t. The entire corpus of documents. Words in the document with a high TF$-$IDF score frequently occur in the document and provide the most information about that specific document.
- **Hashing Trick**: If a data point is converted into a vector by a hash function, it is usually called "the hashing trick" (Garreta et al. 2017). A hash function can be any function used to map data of arbitrary size to fixed-size values, and one of the most popular hashing algorithms is MURMURHASH3. To do the hashing trick, the first thing is to fix the length (M) of the vector. Then using the hash function to get hash value and let the hash value mod M to make it return a number between 0 and $M - 1$.

Compared with other methods to transform data into numerical types, such as one-hot encoding and the bag-of-words model, the most important advantage of the hashing trick is that the data of arbitrary size is mapped to fixed-size values. Using one-hot encoding, we create N binary variables where N is the number of possible values of the categorical variable. And using the bag-of-words model, we create N binary variables where N is the number of words. Both of them create high-dimensional and very sparse input vectors or matrices.

1.3 Feature Learning

Feature learning is to build informative features from an initial set of training data, ignoring redundant and irrelevant information (Kantardzic 2011). From the mathematical sense, that is to find a mapping $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$, so that a set of n-dimensional original data points can be transformed into a set of m-dimensional features (Ding et al. 2012). It's noteworthy that when $m < n$, the feature space can also be called subspace in subspace learning, submanifold in manifold learning, and embedded space in embedding (Ghojogh et al. 2019). In this case, feature learning is related to dimensionality reduction, which is useful when reducing the number of resources is needed for data processing (Burges 2010). Moreover, dimensionality reduction can also provide an easier visualization of data (Liu et al. 2016). Please note that in certain feature learning methods, m can be larger than n, such as in kernel-based methods and deep-learning-based methods.

Generally, feature learning can be divided into two main categories: supervised feature learning and unsupervised feature learning.

In supervised feature learning, each example is a pair consisting of an input data point and the desired output value (often a label of the data point), i.e. $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$. Supervised features are extracted or constructed by analyzing the training data under certain criteria. Approaches include linear discriminant analysis, supervised neural networks, etc.

In unsupervised feature learning, features are built with unlabeled input data, i.e. $\{\boldsymbol{x}_i\}_{i=1}^n$. Since the main goal of feature learning is to build features, without pre-existing labels, most of the unsupervised feature learning methods concentrate on the reconstruction errors or the preservation of the geometrical structure of the original data. Approaches include principal component analysis, locally linear embedding, and autoencoders, etc.

The difference between supervised and unsupervised feature learning is whether or not using the label information of the data. If label information is available, supervised feature learning methods often try to incorporate this information in their metric designs, criteria, or loss functions (Onwubolu and Babu 2013). For example, the similarities between different data points can be designed according to the label information of the data points. The similarity of data points with different labels can be designed as zero or a negative value, while the similarity of data points of the same label may have a positive value. When label information is unavailable, unsupervised feature learning methods often try to preserve the statistical or geometrical information of the original data. For instance, without pre-existing labels, the similarity between different data points usually designed based on their inner products or other nonlinear functions which emphasize certain local structures of the data.

Although criteria or loss functions may be different between supervised and unsupervised feature learning methods (Onwubolu and Babu 2013). The optimization techniques used in these methods are almost the same. There are three types of optimization techniques that are widely used in feature learning: directly solving

eigenvalue equations (or generalized eigenvalue equations), convex optimization, and gradient descent.

1.3.1 Solutions to Eigenvalue Equations

Using training data points (no matter with labels or not), many feature learning methods need to obtain a mapping or a transformation that can be further used on testing data points. Examples include principal component analysis, locality preserving projection, linear discriminant analysis, etc. (Ghojogh et al. 2019; Bengio et al. 2012). These methods look for linear transformations through which the obtained features can best explain the data or model the difference between the classes of data (Jimenez-Rodriguez et al. 2007). Although the criteria are quite different, the objective functions of these methods are all formed into eigenvalue problems (Bai et al. 2000). The linear transformations of these methods are obtained by solving different eigenvalue equations. For example, principal component analysis can compute an orthogonal linear transformation by the eigendecomposition of the empirical covariance matrix of the data (Jolliffe and Cadima 2016; Sakurai et al. 2018).

There are several advantages of solving eigenvalue equations to obtain the feature learning methods. First, analytical solutions can be obtained with eigendecomposition. Second, theoretical analysis of these equations is simple, and the connections with other methods, such as linear regression and logistic regression, are easy to obtain. Third, the extension of these feature learning methods can be derived effectively based on the formations of the eigenvalue equations. For example, using the kernel trick, principal component analysis can be extended to its nonlinear version, kernel principal component analysis, which can also be obtained by eigenvalue equations.

1.3.2 Convex Optimization

In order to obtain a mapping or transformation with certain properties, such as sparsity or low rank, feature learning methods also add constraints in their optimization problems (Tetko et al. 2019; Gao et al. 2010). In this case, directly analytical solutions cannot be obtained, and convex optimization techniques are widely adopted to solve the constrained optimization problems.

For example, sparse principal component analysis (Zou et al. 2006) extends the classical principal component analysis for feature extraction by introducing sparsity constraints to the input variables (Lee et al. 2007). Semidefinite programming (SDP) (Vandenberghe and Boyd 1996), a widely used convex optimization method, can be used to solve sparse principal component analysis with ℓ_1-norm convex constraint.

Although convex optimization techniques used for feature learning are usually iterative, they still can be solved efficiently in polynomial time. Moreover, global optimization can be achieved with a theoretical guarantee.

1.3.3 Gradient Descent

Feature learning does not always mean dimension reduction. In certain applications, the dimension of constructed features can be larger than the dimension of original training data. For example, in deep-neural-network-based methods (Goodfellow et al. 2016), multiple layers of features can be obtained after the training of deep neural networks with multiple layers. Commonly the number of features is much larger compared with the original number of variables. Through the nonlinear transformations in the neural networks, the original data points are mapped into a much higher-dimensional space in which the transformed data can be more representative than original data according to the loss function of the neural networks. In this case, feature learning can be viewed as "feature augmentation" through the multiple-layer aggregation of the original data (DeVries and Taylor 2017; Liu et al. 2017).

Optimization problems in neural-network-based methods are not convex, which are widely solved by gradient descent or its extensions (Looi 1992). In deep neural networks, backpropagation (BP) is an algorithm widely used in the training of neural networks. Gradient descent with backpropagation can find a local minimum of the loss function, and the convergence to a local minimum can be guaranteed (Goodfellow et al. 2014b).

It should be noted that certain feature learning methods can take more than one optimization technique. For example, in sparse learning methods, eigendecomposition and gradient descent are utilized iteratively. In this case, the criterion or the loss function often have two sets of parameters to be optimized. One set of parameters can be obtained by solving eigenequations, while the other set of parameters can be optimized by gradient descent. Although the convergence can be proved, the solution is also a local minimum.

1.4 Summary

In this chapter, we give a gentle introduction to feature learning. Firstly, we show that feature learning is fundamental to the application of machine learning. Then we briefly reviewed the data preparation, which includes data pre-processing and feature learning. Through this chapter, readers can have a general understanding of feature learning. Please note that in machine learning, feature learning refers to a set of techniques that allows a system to automatically build features, which is different from manual feature engineering.

In the following chapters, we mainly focus on latent semantic analysis (LSA, Chap. 2), principal component analysis (PCA, Chap. 3), manifold-learning-based feature learning (Chap. 4), linear discriminant analysis (LDA, Chap. 5), kernel-based feature learning (Chap. 6), and more advanced feature learning methods, such as sparse learning (Chap. 7), low-rank learning (Chap. 8), tensor-based learning (Chap. 9) and neural networks (Auto-Encoders, Chap. 10; Convolutional neural networks, Chap. 11; Recurrent neural networks, Chap. 12).

Chapter 2
Latent Semantic Feature Extraction

2.1 Introduction

Latent semantic feature extraction (LSFE) (Deerwester et al. 1990) is a dimensional reduction framework to obtain meaningful features from large volumes of data. LSFE identifies the pattern in an unstructured collection of data and finds the relationship between them (Hofmann 2001). LSFE tries to capture the hidden structure using methods from linear algebra (Hofmann 2017). The main idea of LSFE is to project the raw data into a low-dimensional subspace such that the noise in the raw data can be removed (Landauer et al. 1998). In information retrieval, LSFE enables retrieval on the basis of semantic contents (Dumais 2004), instead of directly matching different raw data. LSFE is widely used in search engines (Wieser et al. 2013), recommendation systems (Wu et al. 2008), and textual analysis (Yu et al. 2008).

A Latent-semantic-structure-based information retrieval technique was first proposed in 1988 (Deerwester et al. 1989). Due to its wide applications to concept-based automatic indexing, it is also called latent semantic indexing (LSI) (Zha and Simon 1999). In concept indexing, synonymy and polysemy are two fundamental problems. Synonymy refers to the problem that multiple words express the same concept, while polysemy means that words have multiple meanings.

A latent semantic model can be viewed as an extension of the vector space model for information retrieval (Furnas et al. 1988). In the vector space, the collection of text documents is represented by a co-occurrence matrix that describes the occurrences of terms in documents (Van Rijsbergen 1977). In recommendation systems, the co-occurrence matrix depicts the relationships among different customers and distinct catalog items (Wetzker et al. 2009).

After the construction of the co-occurrence matrix, LSFE tries to find a low-rank approximation to the co-occurrence matrix. The computation of the approximation can be viewed as the extraction of key features that are computationally efficient and effective to reduce noise in the original co-occurrence matrix.

H. Zhao et al., *Feature Learning and Understanding*, Information Fusion and Data Science, https://doi.org/10.1007/978-3-030-40794-0_2

In this chapter, we will give a review of the classical theory of singular value decomposition (SVD), which is well studied as a key method of LSFE. We will also discuss the SVD updating technique, which is designed for incremental feature extraction. Finally, through case studies, we show how SVD can be used in a movie recommendation system.

2.2 Singular Value Decomposition

Let the co-occurrence matrix be $\boldsymbol{A} = [a_{ij}] \in \mathcal{R}^{m\times n}$ whose rank is r, there exist orthogonal matrices $\boldsymbol{P} \in \mathcal{R}^{m\times m}$ and $\boldsymbol{Q} \in \mathcal{R}^{n\times n}$ such that

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Sigma}\boldsymbol{Q}^T \tag{2.1}$$

where

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_r & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \in \mathcal{R}^{m\times n}$$

and $\boldsymbol{\Sigma}_r = \text{ diag } (\sigma_1, \sigma_2, \ldots, \sigma_r)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$.

The diagonal elements of $\boldsymbol{\Sigma}_r$ are called *singular values* of $\boldsymbol{A}$, and the columns of $\boldsymbol{P}$ and $\boldsymbol{Q}$ are called left and right singular vectors of $\boldsymbol{A}$, respectively. Equation (2.1) is the *singular value decomposition* (SVD) of $\boldsymbol{A}$.

It is easy to verify that the left singular vectors are eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$, while the right singular vectors are eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$.

Without loss of generality, consider the case $m \geq n$. We have $\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_n \\ \boldsymbol{0} \end{bmatrix}$ with $\boldsymbol{\Sigma}_n = \begin{bmatrix} \boldsymbol{\Sigma}_r & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \in \mathcal{R}^{n\times n}$. Figure 2.1 illustrates SVD with $m \geq n$.

Following Eq. (2.1), the SVD of $\boldsymbol{A}$ can be written as

$$\boldsymbol{A} = \boldsymbol{P}_r\boldsymbol{\Sigma}_r\boldsymbol{Q}_r{}^T \tag{2.2}$$

where $\boldsymbol{P}_r$ and $\boldsymbol{Q}_r$ are the matrices formed by the first r columns of $\boldsymbol{P}$ and $\boldsymbol{Q}$, respectively. Figure 2.2 illustrates Eq. (2.2) with $m \geq n$ and $r < n$. It is easy to see that $\{\sigma_1, \sigma_2, \ldots, \sigma_r\}$ consists of the nonnegative square roots of the eigenvalues of

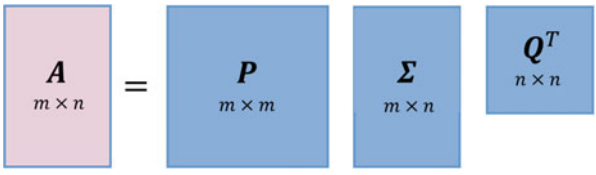

Fig. 2.1 Schematic diagram of SVD with $m \geq n$

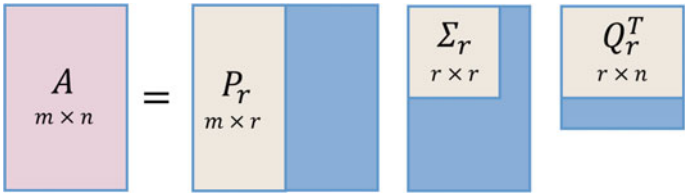

Fig. 2.2 Schematic diagram of SVD with $m \geq n$ and $r < n$

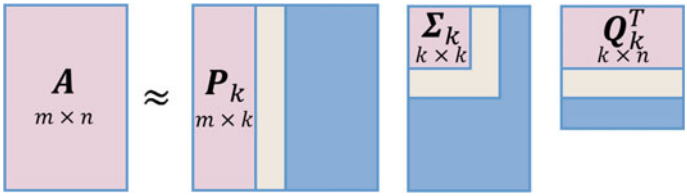

Fig. 2.3 Schematic diagram of SVD with $m \geq n$ and $k < r$

$\boldsymbol{AA}^T$ (or $\boldsymbol{A}^T\boldsymbol{A}$). The singular vectors are not unique, but they are by no means arbitrary. The columns of $\boldsymbol{P}_r$ form an orthonormal basis for the column space of $\boldsymbol{A}$, while the columns of $\boldsymbol{Q}_r$ form an orthonormal basis for the column space of $\boldsymbol{A}^T$.

The *reduced-dimension representation* (RDR) is given by the best rank-k approximation $\boldsymbol{A}_k = \boldsymbol{P}_k\boldsymbol{\Sigma}_k{\boldsymbol{Q}_k}^T$, where $\boldsymbol{P}_k$ and $\boldsymbol{Q}_k$ are composed of the first k ($k < r$) columns of $\boldsymbol{P}$ and $\boldsymbol{Q}$, respectively and $\boldsymbol{\Sigma}_k$ is the k-th leading principal submatrix of $\boldsymbol{\Sigma}$, i.e. $\boldsymbol{\Sigma}_k =$ diag $(\sigma_1, \sigma_2, \ldots, \sigma_k)$. Figure 2.3 illustrates SVD with $m \geq n$ and $k < r$. The approximation error introduced by $\boldsymbol{A}_k$ in Frobenius norm is formed by the ℓ_2-norm of the sum of the remaining singular values,

$$\left\|\boldsymbol{A} - \boldsymbol{P}_k\boldsymbol{\Sigma}_k{\boldsymbol{Q}_k}^T\right\|_F = \left(\sum\nolimits_{i=k+1}^{r}\sigma_i^2\right)^{\frac{1}{2}} \tag{2.3}$$

2.2.1 Feature Extraction by SVD

Matrix $\boldsymbol{A}$ in Eq. (2.1) contains a set of n samples $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n\}$ where $\boldsymbol{a}_i \in \mathcal{R}^m$. Considering Eq. (2.2), we define $\boldsymbol{B}_r = \boldsymbol{\Sigma}_r{\boldsymbol{Q}_r}^T$. It is easy to find that $\boldsymbol{B}_r \in \mathcal{R}^{r \times n}$. Let $\boldsymbol{B}_r = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n]$ and $\boldsymbol{b}_i \in \mathcal{R}^r$. In the following, we show that $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\}$ preserves the geometrical structure of the samples $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n\}$ while r features are extracted from original data of dimension m in the sense that the *distance* and the *angle* (correlation coefficient) between each pair of original samples are preserved, i.e.