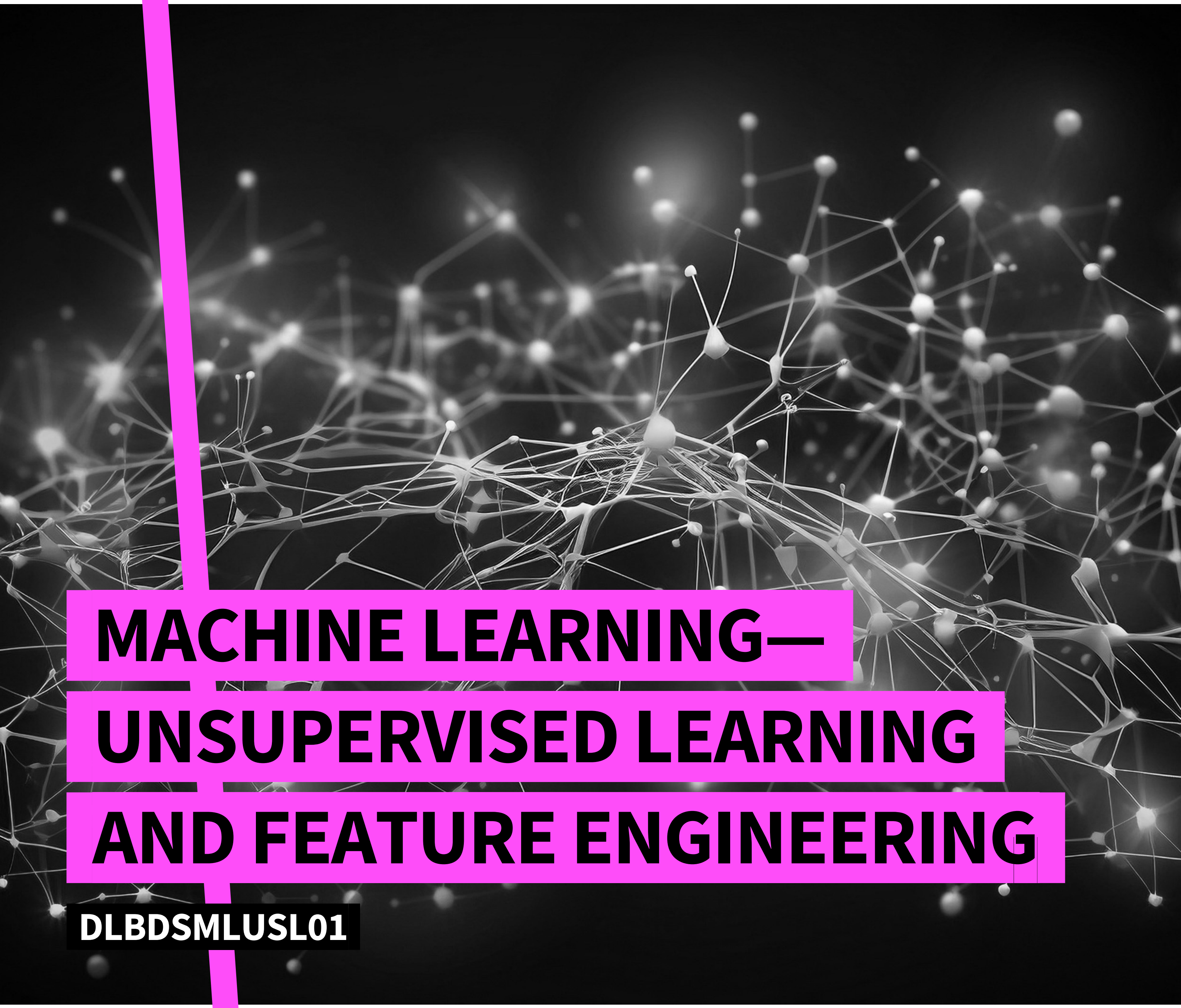
Course Book



MACHINE LEARNING—UNSUPERVISED LEARNING AND FEATURE ENGINEERING

DLBDSMLUSL01

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

MACHINE LEARNING— UNSUPERVISED LEARNING AND FEATURE ENGINEERING

MASTHEAD

Publisher:
IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Mailing address:
Albert-Proeller-Straße 15-19
D-86675 Buchdorf
media@iu.org
www.iu.de

DLBDSMLUSL01
Version No.: 001-2023-0811
N.N.

TABLE OF CONTENTS

INTRODUCTION

WELCOME

SIGNPOSTS THROUGHOUT THE COURSE BOOK

This course book contains the core content for this course. Additional learning materials can be found on the learning platform, but this course book should form the basis for your learning.

The content of this course book is divided into units, which are divided further into sections. Each section contains only one new key concept to allow you to quickly and efficiently add new learning material to your existing knowledge.

At the end of each section of the digital course book, you will find self-check questions. These questions are designed to help you check whether you have understood the concepts in each section.

For all modules with a final exam, you must complete the knowledge tests on the learning platform. You will pass the knowledge test for each unit when you answer at least 80% of the questions correctly.

When you have passed the knowledge tests for all the units, the course is considered finished and you will be able to register for the final assessment. Please ensure that you complete the evaluation prior to registering for the assessment.

Good luck!

SUGGESTED READINGS

GENERAL SUGGESTIONS

Bonaccorso, G. (2019). *Hands-on unsupervised learning with Python: Implement machine learning and deep learning models using Scikit-Learn, TensorFlow, and more*. Packt Publishing Ltd.

Celebi, M. E., & Aydin, K. (Eds.). (2016). *Unsupervised learning algorithms*. Springer International Publishing.

Kane, F. (2017). *Hands-on data science and Python machine learning*. Packt Publishing Ltd.

Patel, A. A. (2019). *Hands-on unsupervised learning using Python: How to build applied machine learning solutions from unlabeled data*. O'Reilly Media.

UNIT 1

Nisioti, A., Mylonas, A., Yoo, P. D., & Katos, V. (2018). From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys & Tutorials*, *20*(4), 3369—3388.

Yao, Q., Wang, M., Chen, Y., Dai, W., Yi-Qi, H., Yu-Feng, L., & Yang, Y. (2018). Taking human out of learning applications: A survey on automated machine learning. *arXiv*.

UNIT 2

Kaufman, L., & Rousseeuw, P. J. (2009). *Finding groups in data: An introduction to cluster analysis* (Vol. 344). Wiley. Chapters 2 and 3

UNIT 3

Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). *Dimensionality reduction: A comparative*. Technical Report TiCC TR 2009-005. Available online

UNIT 4

Dong, G., & Liu, H. (Eds.). (2018). *Feature engineering for machine learning and data analytics*. CRC Press. Chapters 1 and 2

UNIT 5

Solorio-Fernández, S., Carrasco-Ochoa, J. A., & Martínez-Trinidad, J. F. (2020). A review of unsupervised feature selection methods. *Artificial Intelligence Review*, *53*(2), 907—948.

UNIT 6

Das, S., & Cakmak, U. M. (2018). *Hands-on automated machine learning: A beginner's guide to building automated machine learning systems using AutoML and Python*. Packt Publishing Ltd. Chapters 1 and 4

LEARNING OBJECTIVES

Machine learning allows the building or learning of a model that predicts an output (e.g., class or target) based on a set of data points previously gathered in a training or learning set. When the latter does not contain labels indicating the class or target of each data point, the learning is called unsupervised learning. **Machine Learning—Unsupervised Learning and Feature Engineering** will present the most popular methods and techniques used to perform the unsupervised learning.

Using various academic and real examples, you will learn how to design and implement a successful unsupervised learning model according to the problem constraints and challenges. To this end, you will discover how to define features with adequate specific characteristics based on the properties of the input data samples. You will learn how to find, by selecting or transforming, the most efficient features that maximize the performance or success of the designed unsupervised machine learning model. Furthermore, you will become familiar with various popular unsupervised machine learning methods and techniques and gain insight into how to choose between them by discovering their advantages and drawbacks with respect to the problem constraints and challenges.

The recommended reading will add a further dimension to the methods and techniques that constitute this course by introducing you to the most important authors and articles in the field of unsupervised learning. These articles present new case studies, as well as methods and techniques covering the challenges and applications of unsupervised machine learning.

UNIT 1

INTRODUCTION TO UNSUPERVISED MACHINE LEARNING AND FEATURE ENGINEERING

STUDY GOALS

On completion of this unit, you will be able to …

– explain the general principal of unsupervised machine learning and its applications to real-life problems.
– define what features are, their types, their interest for unsupervised machine learning, and their challenges.
– explain the steps of designing an unsupervised machine learning model.
– adapt or transform features for an unsupervised machine learning model.
– evaluate and improve the performance of an unsupervised machine learning model.

1. INTRODUCTION TO UNSUPERVISED MACHINE LEARNING AND FEATURE ENGINEERING

Introduction

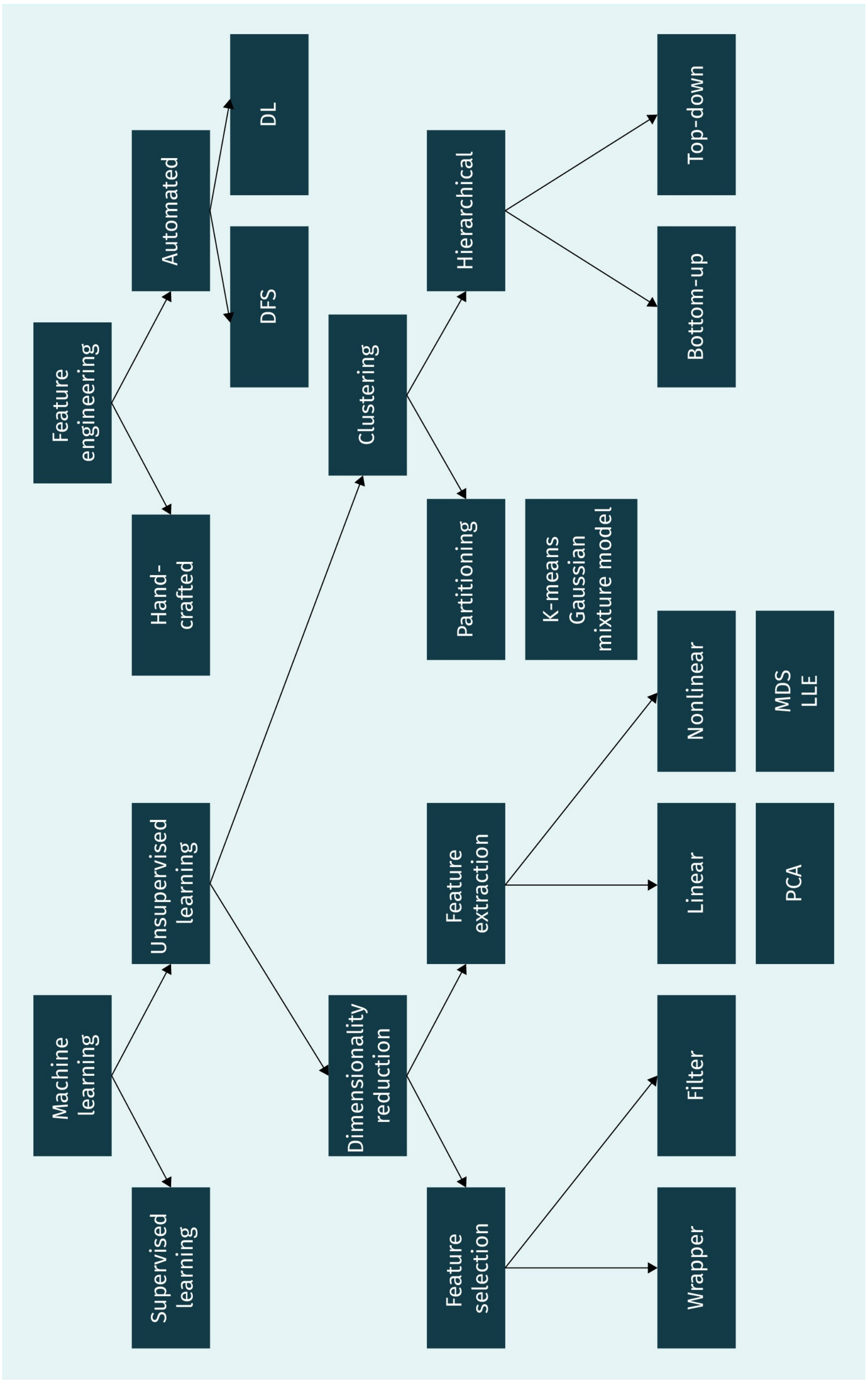
Machine learning is a set of techniques and methods used to build models that are learned from data. These models allows us to predict an outcome, understand, and characterize the data behavior. For example, the model can be used to predict if an email is a spam, if a credit card transaction is genuine or fraudulent, or if the electricity prices will increase or decrease. The model is learned using a **training or learning set**. When the training set contains labels about the prediction or interpretation of each data point, then the model is learned using supervised machine learning. For instance, in the spam example, each email in the training set has a label indicating if it is spam. When the training set does not include labels, then the model is learned using unsupervised machine learning.

Training or learning set
A training or learning set is a set of historical data samples about the phenomenon to be predicted or understood.

Unsupervised machine learning might be considered more complicated than supervised machine learning since the model does not have an output and is learned without a teacher (labels) to instruct the learning algorithm. The task is to extract knowledge (the model output) based only on the input data points. This knowledge is represented by the intrinsic structure of the data points assigned to distinct groups of similar points. For instance, groups gathering products frequently purchased together by the customers can be used to put items of the same group in the same stocking shelves, in sales promotions, in the design of promotional catalogs, and in consumer segmentation.

The quality of the built model (extracted knowledge) depends on the quality of the input data points and the use of meaningful features based on these data points. Indeed, the unsupervised machine learning algorithms require numerical inputs. Hence, the categorical or non-numerical data inputs, such as customer satisfaction (high, medium, and low), need to be transformed into numerical inputs. In addition, these data inputs must be cleaned by dealing with missing values, collinear or redundant input variables, and irrelevant input variables. In addition, it is very useful to derive more meaningful features based on the cleaned or preprocessed input data points, allowing the improvement of the quality of the built model. The techniques used to preprocess, transform, and clean. The input raw data points and extract additional, more meaningful, features are called feature engineering. Feature engineering is essential to build a quality model. However, it is difficult, time-consuming, and requires domain or business problem expert knowledge. The figure below gives an overview of the classification and comparison of the different methods and techniques that will be discussed in this unit.

Figure 1: Overview of Unsupervised Machine Learning and Feature Engineering Models



Source: Sayed-Mouchaweh (2021).

In order to use unsupervised machine learning to extract quality knowledge from a database or training set, the following questions need to be answered:

- How is unsupervised machine learning used to extract the knowledge or build the model?
- What is the suitable unsupervised machine learning method to be used to extract the knowledge or build the model?
- How is the quality of the extracted knowledge or the built model assessed?
- How can input data points be preprocessed, transformed, and cleaned to be suitable for the built model?
- How can the input data points be processed in order to maximize the quality of the extracted knowledge or built model by extracting or coming up with additional and more meaningful features?

1.1 Unsupervised Machine Learning

Unsupervised learning is the process used to train or build a model when the training set is not labeled. The output or target of a learned model is the intrinsic structures or patterns within the data points. Let us pretend that we are trying to estimate the links between eating habits and the risk of a heart attack. If we have a database of different human subjects described by their diet (eating habits), we can see if these human subjects form clusters according to their diet. The goal is to identify, understand or characterize subjects with risk (unhealthy) and no risk (healthy) diet. Since this database does not include a column indicating if the label, or interpretation, of the diet of each subject, the clusters need to be labeled by an expert.

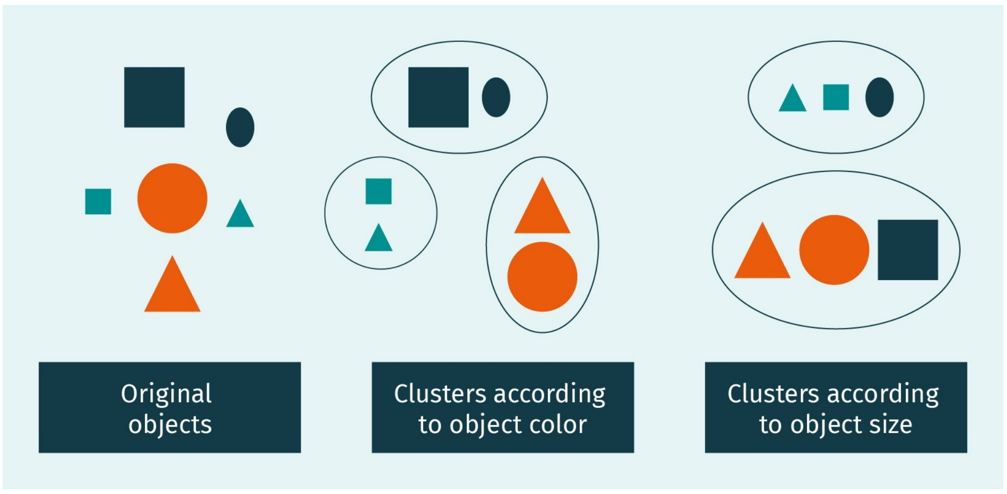
Another example concerns the study of the professional success of individuals according to certain social criteria. The database represents a set of individuals described through these social characteristics (e.g., education level obtained, socioeconomic background, or location). We might obtain two homogeneous groups representing similar individuals. These two groups represent individuals who have been successful in their lives (with respect to the considered factors) and who have not. There are two types of unsupervised machine learning (Müller & Guido, 2016): clustering algorithms and unsupervised transformations (dimensionality reduction).

Clustering Algorithms

Clustering algorithms
The clustering algorithms aim at discovering independent groups, or clusters, in the data points.

Clustering algorithms gather data points into clusters. Data points in each cluster are similar, while they are dissimilar to the data points in the other clusters, according to predefined meaningful similarity/dissimilarity criteria. Clustering can be an efficient technique used to compress the data points by replacing them within each cluster with its statistical characteristics. For instance, if the cluster is compact, i.e., its data points are very close to each other, then this cluster will be represented by its gravity center and its variance. Clustering is unsupervised learning since it is a process of learning by observation, rather than learning by examples (supervised learning). Let us take the example illustrated in the figure below where objects can be divided into clusters according to a predefined similarity measure. If the similarity measure is the color of objects, i.e., each cluster includes objects of the same color, then three clusters can be found; if the similarity measure is the object size, then two clusters can be found.

Figure 2: Clustering by Partitioning


Original objects
Clusters according to object color
Clusters according to object size

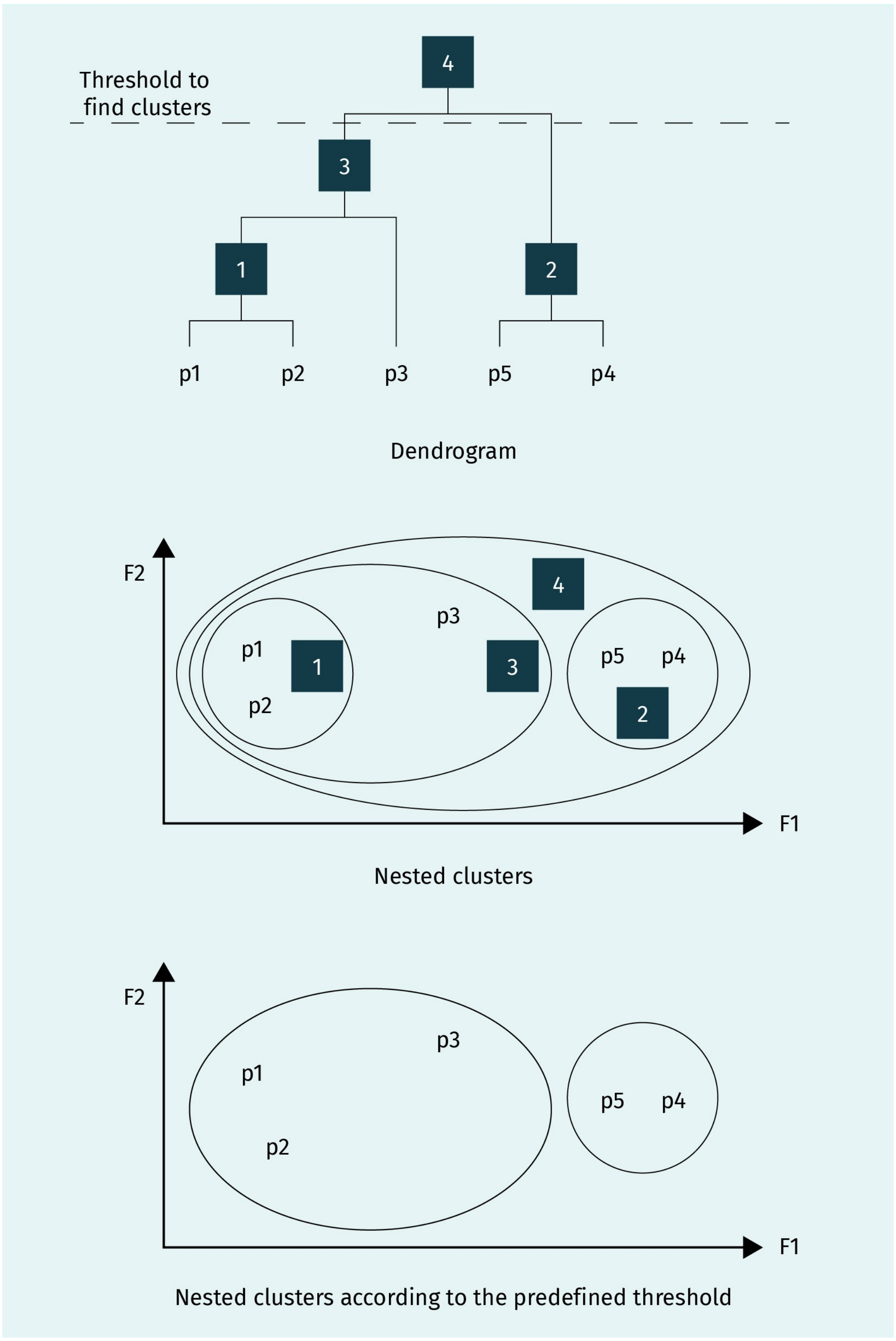
Source: Sayed-Mouchaweh (2021).

Clustering approaches can be divided generally into two main categories (Müller & Guido, 2016): partitioning and hierarchical. Partitioning approaches divide a set of n objects or data points into k partitions or clusters, where $k \leq n$. Therefore, partitioning methods conduct one-level partitioning on datasets. K-means, Gaussian Mixture Model (GMM) clustering, and Density Based Spatial Clustering of Applications with Noise (DBSCAN) are well-known examples of partitioning methods.

Hierarchical methods generate a hierarchical decomposition, a tree or a dendrogram, of the data samples to be clustered. They perform the clustering either as agglomerative merging (starting from the bottom and ending in the top) or divisive splitting (starting from the top and ending down) of data samples. Each merging or splitting process will add a new hierarchical level of decomposition of data samples. The agglomerative methods consider each data sample a separate cluster. Then, the clusters are successively merged by adding new data points or clusters that are close to each other. This merging process will continue until it either reaches one cluster that includes all the data points in the dataset or a stop criterion is satisfied.

The top-down or divisive methods, more popular than agglomerative methods, start with all the objects in the same cluster. Then, in each successive iteration, a cluster is split into smaller clusters until each object forms one cluster or a stop criterion is satisfied. The number of clusters is determined either manually by the user, or by defining a threshold as illustrated in the figure below.

Figure 3: Hierarchical (Agglomerative) Clustering



Threshold to find clusters
4
3
1
2
p1
p2
p3
p5
p4
Dendrogram
F2
F1
4
p3
p1
1
3
p5
p4
p2
2
Nested clusters
F2
F1
p3
p1
p5
p4
p2
Nested clusters according to the predefined threshold

Source: Sayed-Mouchaweh (2021).

There are several differences between these clustering approaches, allowing the choice of one according to the dataset conditions and available prior knowledge. For instance, k-means can be applied to discover hard clusters of spherical shapes. Hard clusters mean that all data points within a cluster belong to it with the same strength. However, normally, data points closer to the cluster center belong to the cluster with more of belief strength than the data points located at the cluster periphery. In addition, k-means requires the determination of the number of clusters to discover, k, in advance.

Gaussian Mixture Model (GMM) clustering is more flexible than k-means since it can discover clusters of an elliptical shape thanks to the use of the **matrix variance-covariance**. Indeed, the correlation (covariance) between each pair of attributes (axes) in the matrix variance-covariance allows us to take into account the elliptical shape of clusters. Moreover, it discovers soft clusters since it is a probabilistic algorithm. In other words, it assigns higher probabilities to data points according to the belief one has that those data points belong to a specific cluster.

Matrix variance-covariance
A matrix variance-covariance indicates the variance of data points along each attribute and the covariance or correlation between each pair of attributes.

DBSCAN has the following advantages: a) it does not require the number of clusters k to be defined in advance, b) it can be applied to discover clusters with arbitrary or complex shape, and c) it can deal with noisy data, i.e., identify data points that do not belong to any cluster. These are very useful advantages since most of the real-world datasets include outliers and missing, unknown, or erroneous data samples. Noisy data impacts the quality of the resulting clusters that the clustering algorithm cannot deal with (are not robust against noises). However, DBSCAN is slower than k-means. This limits its scalability or its ability to be applied to large datasets.

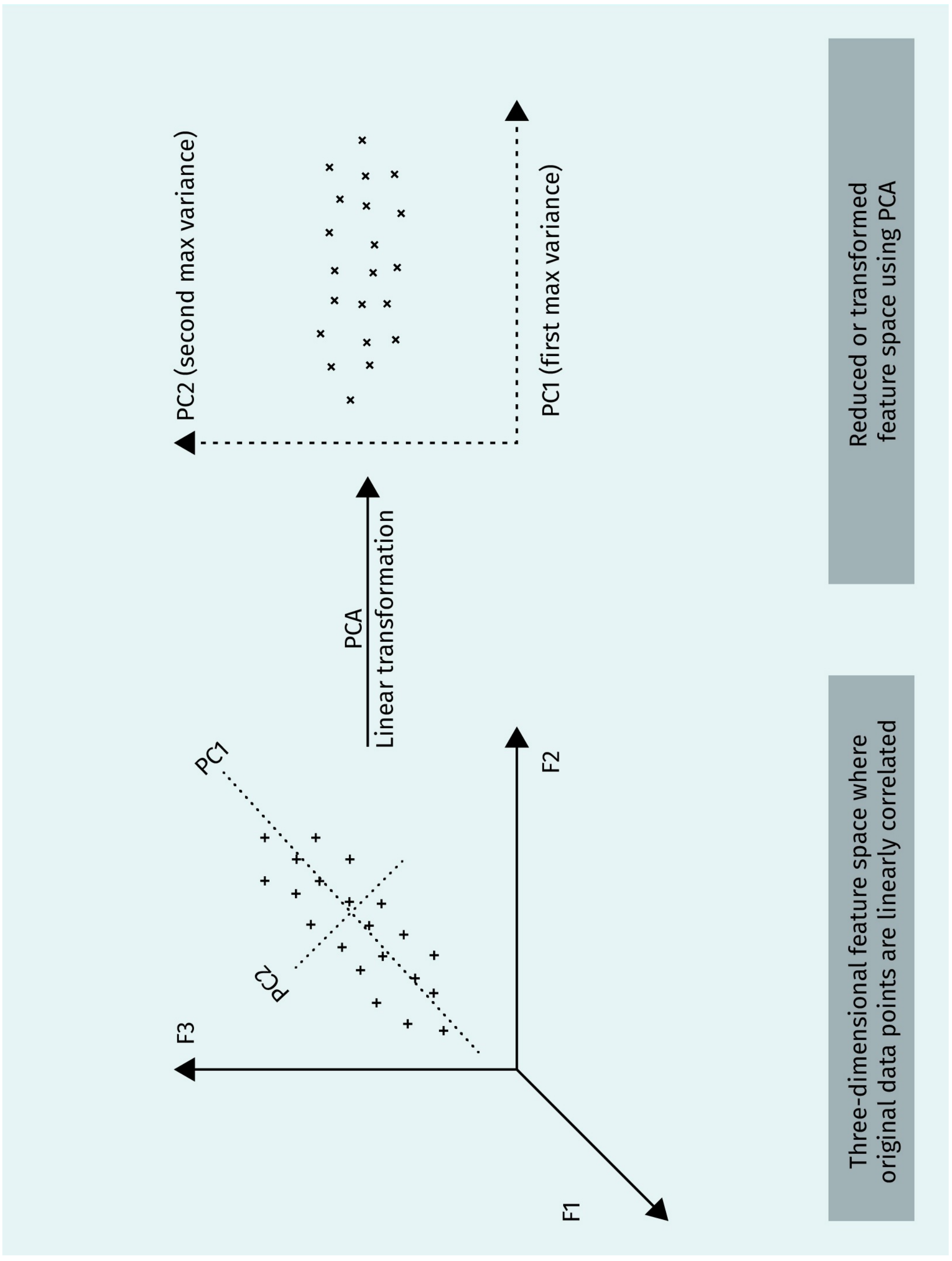
Unsupervised Transformation (Dimensionality Reduction)

The main application of **unsupervised transformations** is the dimensionality reduction where data points described in a high-dimension feature space are represented or transformed in another feature or representation space with much fewer features. The new representation or feature space allows us to conserve certain important properties and characteristics of the transformed data points, such as the distances between them, or their variance. Generally, the original features are reduced to two dimensions in order to present the data points visually in two-dimensional representation space.

Unsupervised transformations
The unsupervised transformations aim at summarizing the main characteristics of high-dimensional data in order to represent it into much smaller dimension space.

Let us take, as an example, one famous method of dimensionality reduction: Principal Component Analysis (PCA). Let us apply this method to the example depicted in the figure below. In this figure, the original data forming a cluster are represented in three-dimensional feature space. The data can be transformed and projected using PCA into another, smaller, feature space composed of two uncorrelated features (principal components PC1 and PC2) that capture the most of the variance of the original data as illustrated in the figure below.

Figure 4: Unsupervised Transformation Using Principal Component Analysis (PCA)

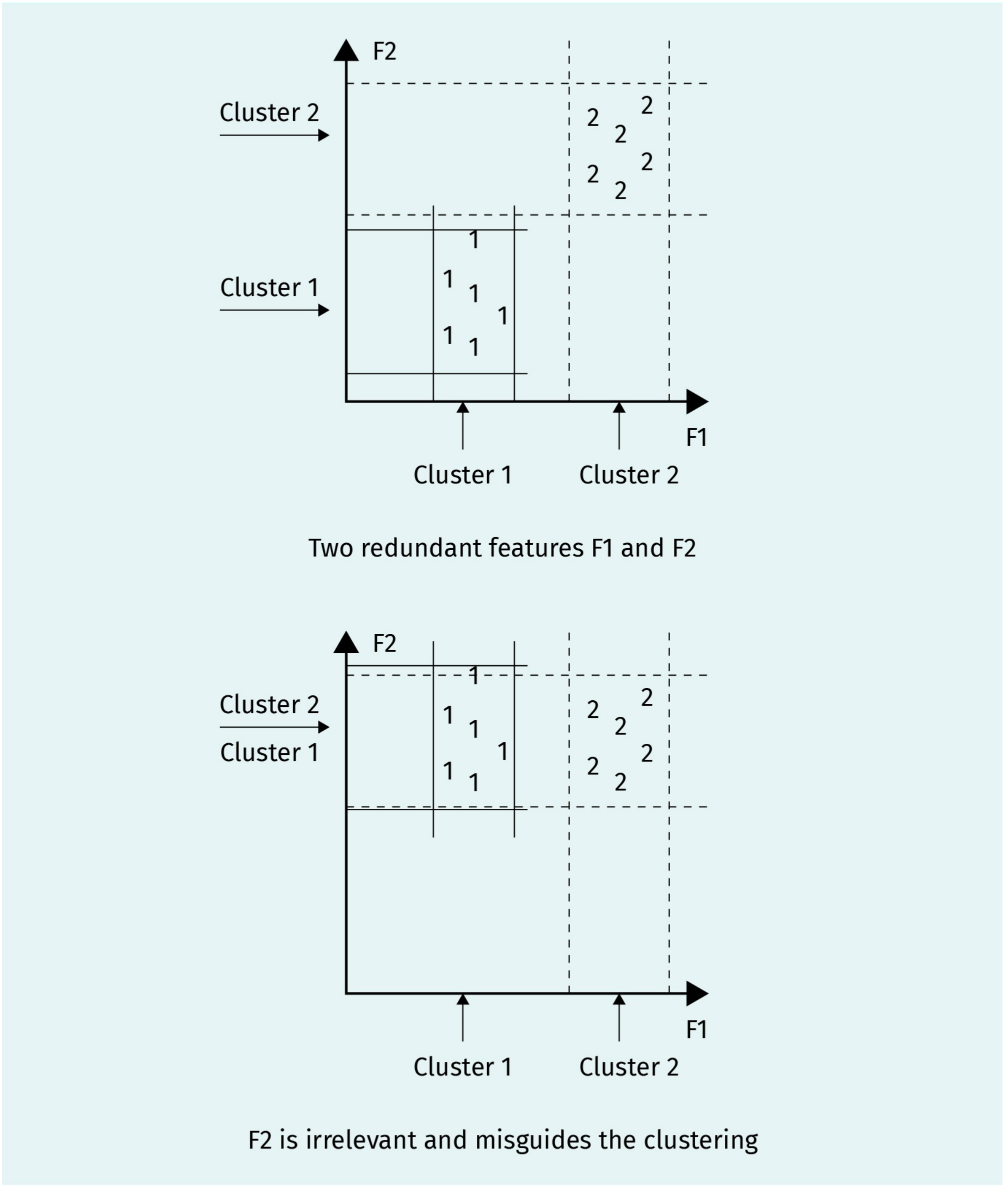

Three-dimensional feature space where original data points are linearly correlated
F1
F2
F3
PC1
PC2
PCA
Linear transformation
PC1 (first max variance)
PC2 (second max variance)
Reduced or transformed feature space using PCA

Source: Sayed-Mouchaweh (2021).

Unsupervised transformations methods are divided into two categories. The first is feature selection (Dy & Brodley, 2004), where a subset of features are selected according to their contribution to the quality of the knowledge extracted by the built model. This contribution can be measured directly according to a specific built model, or indirectly, independently of the built model. The former are called the wrapper methods, and the latter, filter methods. In both cases, the selection process is achieved by removing irrelevant, redundant, or noisy features. Consequently, the feature selection will improve the quality of the built model through higher learning accuracy, lower learning and computation cost, and better model interpretability. Car shopping, for example, could be helpful with feature selection. It could be used with the clustering of cars according to their price. If we have, as the feature, the color of the car, this feature does not contribute to the price of the car, but the brand and model year are informative features used to determine their price.

Let us take the example of the figure below in which two clusters are represented into two-dimensional feature space. It is clear that both features, F1 and F2, provide the same information regarding the discrimination of the data points into two different clusters. Therefore, one feature can be selected to discriminate the data points into two clusters. The figure below shows an example of an irrelevant feature where feature F2 does not contribute to the discrimination of the two clusters naturally existing into the data points. It is worth mentioning that noisy or irrelevant features not only fail to contribute to the discrimination of clusters in the data points, but also misguide the clustering.

Figure 5: Redundant and Irrelevant Features

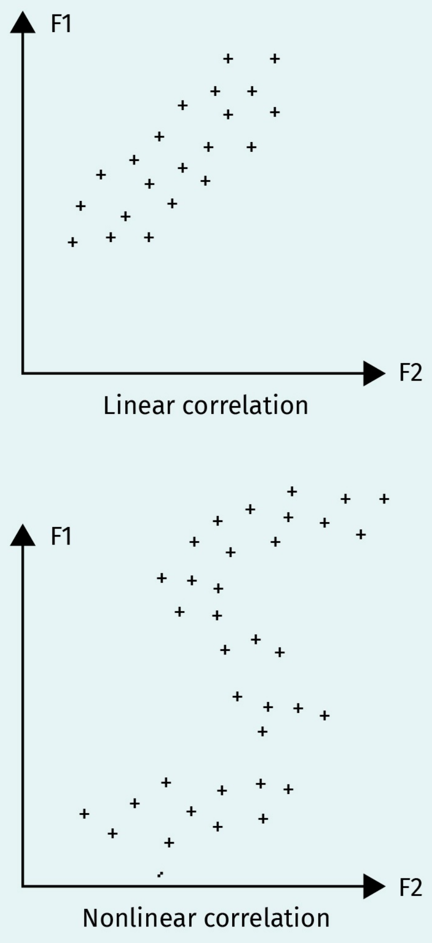


Source: Sayed-Mouchaweh (2021).

The second category is feature extraction (Ding et al., 2012). This reduces the number of features by projecting or transforming the original data into a new smaller feature space. In the latter, the representation of the original data is conserved as faithfully as possible. Therefore, the difference between feature selection and feature extraction is that in the former, the new reduced feature space is based on a selection of the original features, while in the latter, the new reduced feature space is based on new features that are built by transforming the original features.

Feature extraction can be divided into linear dimensionality and nonlinear dimensionality reduction methods. In linear dimensionality reduction methods, such as PCA, Factor Analysis, and Linear Discriminant Analysis, the original data points in the feature space are linearly correlated and thus can be linearly transformed and projected into a reduced new feature space. In nonlinear dimensionality reduction methods or manifold learning methods, such as Multi-Dimensional Scaling (MDS), Locally Linear Embedding (LLE), and Kernel PCA, the original data points are correlated in the feature space in nonlinear way, as illustrated in the figure below. For instance, MDS projects original data points into a lower dimensional space in a such a way that data points close to each other (e.g., in terms of Euclidean distance) in the original high dimensional space also remain close in the transformed, lower, dimensional space. In LLE, each local patch of the nonlinear structure of the original data points can be written, transformed, as a linear, weighted sum of its neighbors given that there are enough data points.

Figure 6: Data Points Linearly and Nonlinearly Correlated in the Original Feature Space



Source: Sayed-Mouchaweh (2021).

Real-Life Applications of Unsupervised Machine Learning

There are an important number of applications of unsupervised machine learning approaches used to solve real-life problems. This number is continuously increasing, particularly within the context of digital transformation. The latter allows the generation of a huge number of data records that can be transformed into knowledge and insights to facilitate the decision-making process and increase efficiency. The following paragraphs present some of the major applications of unsupervised machine learning. Other applications can be found in Bernard et al. (2018), Westermann et al. (2020), Verkerken et al. (2020) and the references therein.

Medical diagnosis (Alashwal et al., 2019)

Unsupervised machine learning, such as clustering approaches, can be used to understand the parameters that cause, entail, or encourage the evolution from normal or mildly impaired subjects toward a certain disease. To this end, a set of data (clinical tests and biomarkers) about the medical situation of healthy and unhealthy human subjects are used. These data are divided into clusters gathering healthy, mildly impaired, and sick patients. These clusters are then used to understand the correlation between the subjects or the identification of parameters leading the subjects to evolve from healthy or mildly impaired to a disease situation. For instance, Alashwal et al. (2019) discuss the use of the unsupervised machine learning (e.g., k-means) in order to discover patterns and structures in unlabeled datasets of the neurological Alzheimer’s disease (AD). As an example, the mild-cognitive impaired (MCI) subjects were clustered into cognitive normal (CN)-like and AD-like. The goal was to observe the MCI subjects that may stay as healthy individuals (CN) and the MCI subjects that may evolve to Alzheimer’s disease (AD). This permitted an early diagnosis of AD at the MCI stage.

Fault diagnosis of industrial systems (Toubakh et al., 2020)

An industrial system, such as a manufacturing system, works under normal operation conditions. Clustering techniques can be applied to the historical data points describing the system’s normal operation conditions in order to divide them into clusters in the feature space. A fault can occur, impacting the normal operation functioning of the system or one of its components, such as a pump or a valve. For instance, the pump may be failed-on or failed-off. In the former, the pump cannot be stopped, leading to the problem of overflow, while in the latter, the pump cannot be turned on and stays off. If the pump is used to cool (reduce the temperature of a generator), then the failed-off fault of the pump can lead to a catastrophic damage in the generator. The fault can be seen as an outlier that does not have a similar characteristic with any of the normal clusters. Data points representing the failed-off or failed-on pump faults can then be gathered in clusters to understand their characteristics and the factors they entail.

Customer segmentation or client profiling (Băcilă et al., 2012)

Marketers and service providers need to target customers with suitable products or services according to their profile or interest. Customer segmentation aims at gathering customers into segments or clusters based on similar demands, needs, consumption, and interests. The goal is to obtain clusters of customers or clients that have a similar profile or response to marketing strategies (e.g., products and services). The obtained clusters enable marketers and service providers to design efficient marketing strategies and planning, thanks to a better understanding of the customer needs and interests, enhancing customer satisfaction. In Băcilă et al. (2012), telecom services subscribers were divided into clusters according to how they spent money on credit recharging, making calls, sending SMS, and navigating the internet. The resulting clusters allowed the characterization of expenditure patterns for certain telecom services (SMS, internet navigation, phone calls, etc.). The telecom company used these patterns to identify the most attractive consumer segments, with a better positioning of services or a prioritization of resources.

Crime and fraud detection (Maddila et al., 2020)

Crime activities can be characterized according to type (e.g., robbery, assault, fraud, or murder) as well as location (e.g., city or neighborhood). Clustering crime data records to gather similar crimes according to their type and location can help give quality insights into crime-prone areas within a city or an area by identifying the crime tendency and the factors that create it. Similarly, fraudulent transactions of any kind (e.g., credit cards, insurance claims, or online shopping) can be detected and characterized using unsupervised machine learning, such as abnormality detection techniques. Genuine transactions are gathered into clusters using a set of features, such as how much money was spent, which types of products were purchased, or where and when (day or hour) these products were purchased. A fraudulent transaction is detected if it does not have similar characteristics to the genuine clusters. With clustering techniques, gathering these fraudulent transactions into clusters allows the understanding of their characteristics and helps improve the security of future transactions.

Throughout these examples of the use of unsupervised machine learning in real-life applications, the obtained knowledge and insights can improve the quality of life and safety of human beings, as well as their health and well-being. However, privacy and virtual equity issues arise due to the use of increasing volumes of data by unsupervised machine learning, related to personal and sensitive information. Indeed, personal and sensitive information can be inferred using unsupervised machine learning. For instance, the health status, consumption behavior or profile, or financial situation of a patient or a customer is personal and sensitive information that might be used against them.

As for predictive policing, there are use cases where unsupervised machine learning can help investigators solve crimes by identifying patterns in criminal behavior. But, there are also many cases where unsupervised machine learning has shown a reproduction of racist narratives. Thus, efforts are made to address these issues when using machine learning approaches. For instance, pre-trained models can be used to avoid having access to the training dataset. Another example is to perform the learning in a decentralized manner. This prevents models from accessing all of the dataset from one point, making it unfeasible to infer sensitive or private information. Moreover, a whole research field focuses on fairness in data science in order to learn from mistakes made in the past. At the end of the day, we decide for what purpose and how these techniques are used and whether they will produce added value for all of us.

1.2 Feature Engineering

A feature, known also as an attribute or predictor, is an individual property or characteristic extracted from a set of data points describing an input variables. Inputs are considered to be independent measurable variable. There are two types of features according to the types of input variables used to define them: numerical and categorical. Numerical variables are measurable variables whose values have a mathematical order, such as the number of children in a family, a person's weight and height, or house prices. Categorical features are defined using categorical input values. The latter can take a fixed and limited number of possible values, such as the gender or home country of a customer. Categorical variables are not meant to be used by statistical methods since it is not possible to compute summary statistics, such as mean value, variance, or median.

Categorical variables can be nominal or ordinal. In nominal variables, the values do not have any kind of mathematical order, such as a customer's gender or the model of their car. In contrast, the values of ordinal features have a meaningful order or scale, such as customer satisfaction (high, medium, and low), and an exam grade (A, B, C, and F). Ordinal variables can exhibit numerical and categorical characters when numbers are used to scale the order of its values, such as customer satisfaction between 0 (not-satisfied at all) and 5 (completely satisfied). However, this should be used with caution since there is danger in using statistical methods which are not meant for ordinal data.

The training set used to build the model using unsupervised machine learning includes raw data organized into a table of n rows and d columns. N indicates the number of data points while d indicates the number of measured features. For example, we have a table indicating the energy consumption of 100 different consumers at each hour of the day for 12 months. We have $\mathrm{n}= 100{\cdot}12{\cdot}360 = 864000$ lines. If we provide, in addition to the energy consumption per hour, the consumer ID, the size of their home in m^2, employment status, age, and marital status, then we have $\mathrm{d}= 6$ features.

We want to build a model to categorize these consumers according to their consumption. The model will be trained using the training set. The used features impact the quality of the extracted knowledge to a large extent, e.g., clusters grouping the energy consumption of consumers. If these features are not meaningful, the extracted knowledge could be poor and useless. Therefore, the more you have useful and meaningful features, the more the built model is efficient and generic. Consequently, the first step in building an efficient unsupervised machine learning model is to represent or extract useful and meaningful features as much as possible. This is done by feature engineering (Müller & Guido, 2016; Ozdemir & Susarla, 2018).

Feature engineering (or feature creation) aims at defining features from the available raw data samples in order to improve the performance of the unsupervised or supervised machine learning model. These features are defined by converting data samples based on a blend of domain expertise, intuition, and mathematics. Useful and meaningful features will not only largely improve the quality of the extracted knowledge (clusters), but will also allow the use of less complex models that can be built or learned faster and interpreted more easily.

Let us take the example of energy consumption clustering (Sayed-Mouchaweh, 2020). Based on domain expertise, we propose adding a feature indicating if the day is a weekend, day off, or a normal working day, since the energy consumption depends on whether the consumer is at home or at work. Another additional feature is the maximum or minimum energy used each day by the consumer. This feature allows us to observe the days when the energy consumption is at the maximum or minimum. Thus, this information identifies the consumers who have similar usage. A utility company can use this information to ensure the balance between consumption and generation. When the consumption is greater than the power generation, the utility company can ask the consumers in the high consumption cluster to stop or reduce their consumption through a demand response strategy (Sayed-Mouchaweh, 2020). Other additional features include the mean value of the consumption of each consumer per day and per month, and the mean value of consumption of all consumers per day and per month. These features allow us to identify consumers who use much more than the average. These consumers may have a problem or anomaly in their installation and need repairs.

Moreover, it may be helpful to add features that represent the interactions (products) of features. For instance, we may add a new feature as the product between the energy consumption and the home size demonstrates the interaction between these two features. This new feature helps distinguish a cluster gathering high, but normal, energy consumption. This can be explained by large home sizes from a cluster gathering high energy consumption because of abnormal or excessive energy need.

Feature Engineering Types and Steps

There are two types of feature engineering used to define features: manual and automated feature engineering generation. In manual feature engineering, the features are defined manually based on the intuition, domain knowledge, and user expertise. Let us take the following simple example of student records composed of both tables below.

Table 1: Student_R

Student_ID	Birth_date
S1	1996-07-14
S2	1997-08-22
S3	1998-05-11

Source: Sayed-Mouchaweh (2021).

The second table includes the student ID and grades for two courses.

Table 2: Courses

Student_ID	Grades
S1	18

S2	11
S3	12
S1	15
S2	19
S3	10

Source: Sayed-Mouchaweh (2021).

We have two tables with a common link: the student identification number. The latter is unique in each row of each table. Let us manually create new features using Python. You need to import the following Python modules:

Code

```
import numpy as np
import pandas as pd
import datetime
```

We will create the two simple tables above, including three students.

Code

```
Student_R =
        'Student_ID':['S1', 'S2', 'S3'],\
        'Birth_date': [datetime.date(1996,7,14),
                      datetime.date(1997,8,22),
                      datetime.date(1998,5,11)]}
Student_R = pd.DataFrame (Student_R, \
     columns = ['Student_ID','Birth_date'])
Courses = { \
    'Student_ID':['S1', 'S2', 'S3', 'S1', 'S2', 'S3'],
    'Grades':[18, 11, 12, 15, 19, 10]}
Courses = pd.DataFrame (Courses, \
    columns = ['Student_ID','Grades'])
```

Now, let us create a simple feature based on a single table. For instance, we can create a feature indicating the birth date of each student.

Code

```
Student_R['year'] =
pd.DatetimeIndex(Student_R['Birth_date']).year
print(Student_R.head())

# console output:
#   Student_ID Birth_date year
# 0        S1 1996-07-14 1996
# 1        S2 1997-08-22 1997
# 2        S3 1998-05-11 1998
```

Now, the Student_R table has one additional column indicating the birth year of each student. Let us now define features requiring more than one table. For example, let us create features about the mean, max, and min of the grades obtained by each student. Finally, let us include all the created features in one target table: Student_R. This table will be used to build the machine learning model.

Code
```
# creation of features by aggregation of grouped values
goper = Courses.groupby('Student_ID')['Grades'].\
   agg(['mean','max','min'])

# rename columns
goper.columns = ['mean_grade','max_grade','min_grade']
print(goper.head())#Creation of features by grouping tables
goper = Courses.groupby('Student_ID')['Grades'].agg(['mean','max'
                                           ,'min'])goper = Courses.groupby('Student_ID')['Grade
       (['mean','max','min'])
goper.columns =['mean_grade','max_grade','min_grade']
print(goper.head())
# Merge with the Student_R dataframe
R = Student_R.merge(goper, left_on = 'Student_ID',
                   right_index=True, how =
                   'left').head()
print('Student-R with the new manual features\n', R)

# merge with the Student_R dataframe
R = Student_R.merge(goper, left_on = 'Student_ID', \
   right_index=True, how = 'left'). \
       head()
```

The new table with the additional features is below.

Table 3: Student_R with Generated New Features

Student_ID	Birth_date	year	mean_grade	max grade	min_grade
S1	1996-07-14	1996	16.5	18	15
S2	1997-08-22	1997	15.0	19	11
S3	1998-05-11	1998	11.0	12	12

Source: Sayed-Mouchaweh (2021).

Manually coming up with useful and meaningful features can often be painstaking, tedious, and time-consuming, and requires expert knowledge. In addition, the defined features are problem-specific and cannot be used for other problems. To avoid these limits, automated feature engineering (Müller & Guido, 2016; Ozdemir & Susarla, 2018) is an alternative; it automatically extracts useful and meaningful features from a set of related data tables with a standard framework that can be used for different problems.

Automated feature engineering helps reduce the required time to define features and create interpretable features. In addition, it prevents data leakage by filtering time-dependent data. These features can be generated automatically using either deep feature synthesis (DFS) or deep learning (DL). DFS helps create multiple **deep features** based on the use of the process called DFS in the featuretools library in Python.

Deep features
The deep features are new features created by transforming features in the same table or aggregating features of different tables using basic operation, such as mean or max.

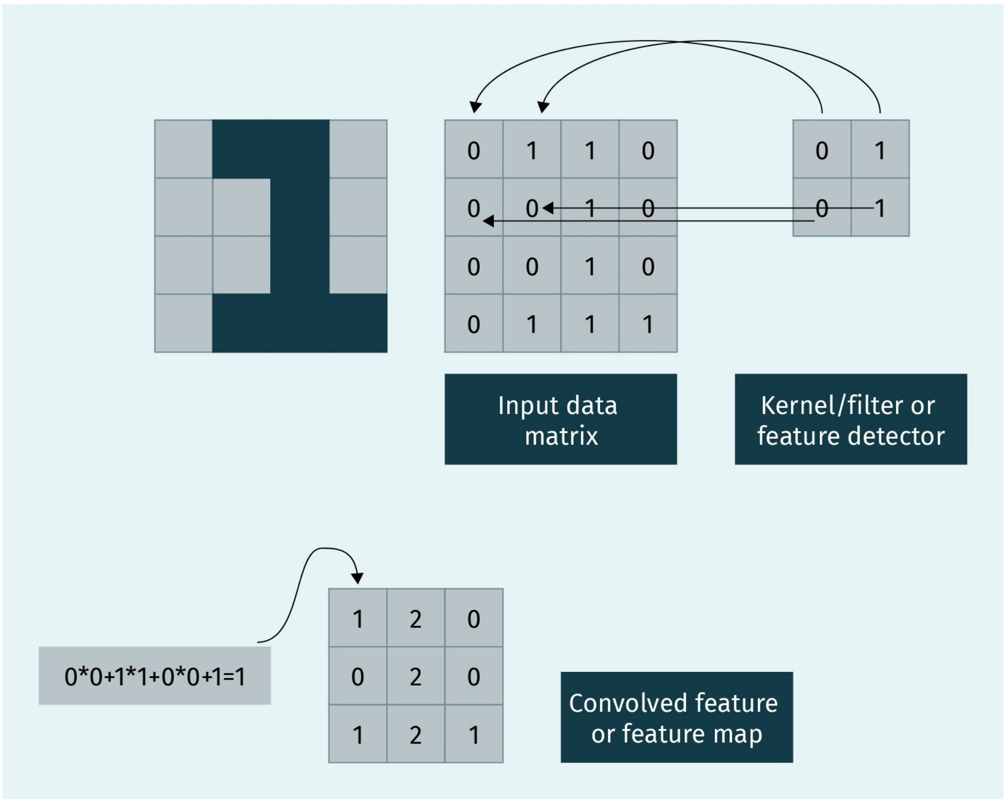
A deep feature is created using a set of primitives, the basic operations (e.g., mean, max, and min) used to form new features. A deep feature can be created either as transformations or aggregations. The transformations are done to one or more columns on a single table, such as the difference between two columns in one table or taking the absolute value of a column. The aggregations are achieved among different primitives applied to several tables, such as finding the maximum obtained grade of each student requiring the use of both the Student_R and Courses tables to create this feature.

Depth
The depth is the number of aggregations that featuretools needs to generate features.

Featuretools generates as many features as we want. All we need to do is define the **depth** to which you want to stack the generated features. For instance, creating a feature using the aggregation operation "Mean" is of depth 1. However, creating a feature as the most recent of the mean values (Last of Mean values) is of depth 2, since two aggregations are required to generate it by finding the mean values and the most recent value of mean values.

Deep learning approaches, such as the convolutional neural network (CNN) (Sewak et al., 2018), also automate feature generation from raw data through matrix multiplication. Let us take the example of digits recognition, 0 to 9. Each digit is represented as a grayscale input matrix of pixels. The figure below exemplifies the digit "1." CNN uses a kernel or filter to define a feature that can be used to recognize a digit. For instance, in the figure below, the filter is used to generate a feature allowing the detection of vertical lines. Similarly, other features can be generated, such as horizontal or left/right diagonal. The multiplication between the input raw data matrix and the vertical filter allows us to generate the feature map indicating if the corresponding feature (vertical lines) exists in the input image. For our example, the value "3" indicates the existence of a vertical line. CNN automatically generates the suitable filters to search for useful features in the raw input matrix. All we need to do is define the number of filters.

Figure 7: Automated Feature Generation by Convolutional Neural Networks (CNN)



Source: Sayed-Mouchaweh (2021).

However, deep learning approaches require a huge number of training data samples in order to learn and train the complex architectures they need to work. Moreover, the extracted features are not interpretable; however, the generated features by featuretools are interpretable since they are based on the combinations of simple primitives (e.g., mean, max, and min) that can be easily described in natural language. In addition, featuretools can generate features from small datasets. It is worth mentioning that raw data points may need to be pre-processed to allow an efficient use of feature engineering. Indeed, raw data may need to be cleaned by removing outliers or by dropping or imputing missing values for some data points in the dataset.

Building a Successful Unsupervised Learning Model from Raw Data

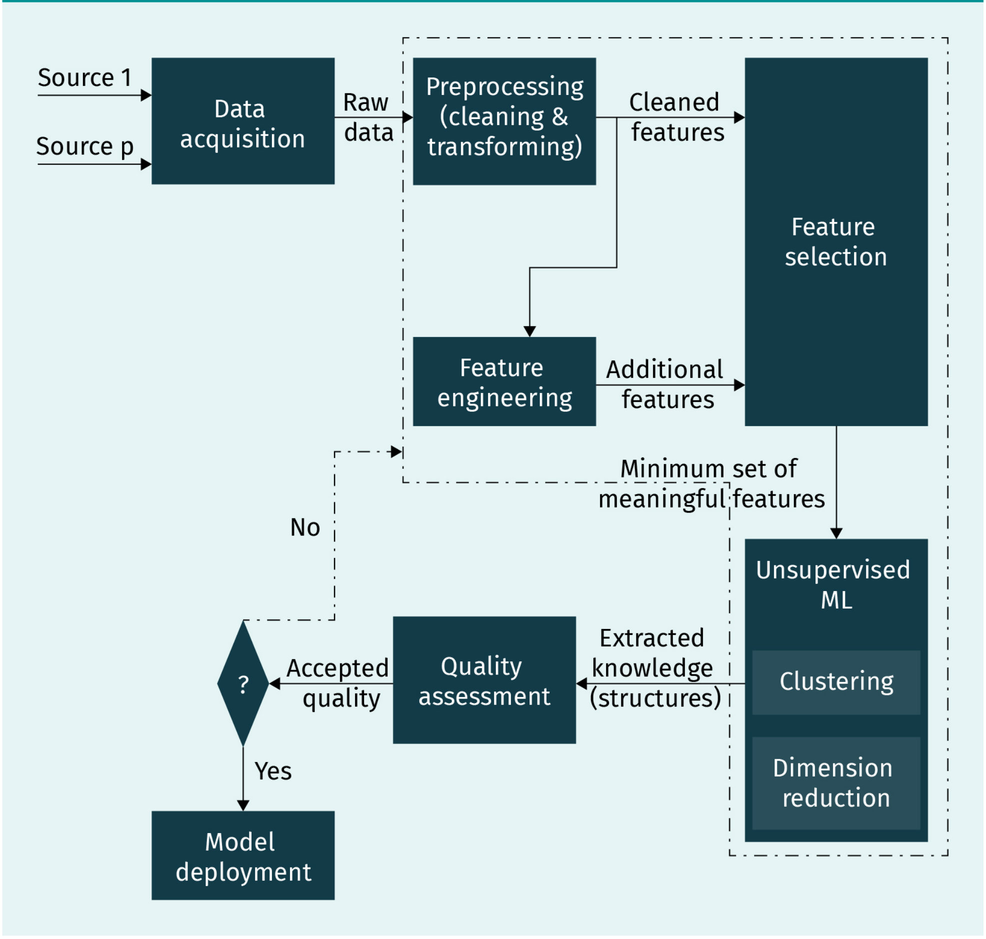
Automated feature engineering, through featuretools in Python, creates many features. However, having too many features can lead to the curse of dimensionality. Indeed, as the number of features increases, the number of data samples required to achieve efficient learning also proportionally increases, i.e., data samples are required to have all combinations of features. The model becomes more complex with a higher number of features. The chances of the model becoming too dependent on the training set, i.e., overfitting, increases when the number of features increases. The danger of introducing features which only pretend to add information also increases.

Consequently, it is important to apply feature selection and extraction as well as dimensionality reduction techniques to avoid overfitting and the curse of dimensionality. This allows us to obtain better quality of extracted knowledge thanks to the use of fewer misleading data, less training time, less storage space, and less complex models.

Finally, most of the unsupervised machine learning approaches are designed to work with numeric data. However, most of the real datasets have other types of data, such as binary, nominal, and ordinal data, or mixtures of these data types. Thus, these other types of data need to be converted into a suitable type (numeric) in order to be adapted for the unsupervised machine learning methods. The figure below shows the different steps required to build an efficient unsupervised machine learning model from raw data to implementation and deployment.

Figure 8

Steps Required to Build an Efficient Unsupervised Machine Learning Model



Source: Sayed-Mouchaweh (2021).

SUMMARY

This unit presented an overview of machine learning methods by dividing them into supervised and unsupervised learning. It focused on the unsupervised learning, when training data points do not have labels, by briefly discussing their functioning principals and comparing their performances with respect to the problem conditions and constraints and the characteristics of available datasets. In addition, this unit presented the definition and motivation of feature engineering, as well as its types, handcrafted and automated feature engineering, in order to build a successful unsupervised machine learning model.

The unit compared the advantages and drawbacks of the different types of feature engineering generation with respect to the required time and mental effort to generate features and the dataset size. The unit ended by presenting the required steps to build a successful unsupervised machine learning model from raw data through to its implementation and deployment. The discussion and comparison were illustrated by several simple examples and real-world applications.

UNIT 2

CLUSTERING

STUDY GOALS

On completion of this unit, you will be able to …

- explain the functioning principal of clustering approaches and how they work.
- implement a clustering approach.
- test and evaluate the quality of the obtained clusters.
- choose the clustering approach with respect to the challenges and constraints of the dataset.

2. CLUSTERING

Introduction

Let us suppose that $X = \{x_1, x_2, ..., x_n\}$ is a set of historical data points or observations about a phenomenon or a dynamic system. Each data point, x_i, $i = 1, ..., n$, is a vector of p dimensions. Each dimension represents a measurement of a certain property of the phenomenon or system under consideration. Clustering aims at discovering the natural groups (clusters) that represent the structure of a set of data points. In general, two criteria are used to assign data points in each group or cluster: the inter-cluster distance (or separability) and the intra-cluster distance (or cohesion). Therefore, data points within a cluster must be similar to each other, and they must be different from the data points in other clusters.

Several challenges face the clustering process, such as the determination of the number of natural clusters in the data samples, missing or noisy (e.g., outliers) data samples, overlapping clusters, complex or non-convex shape of clusters, and convergence to local optimal minima. These challenges impact the quality of the obtained clusters by a clustering algorithm.

There are several algorithms used to perform the clustering. They can be divided into partitioning, such as k-means and Gaussian Mixture Model (GMM) clustering, and hierarchical clustering approaches. Both categories of approaches are based on the same principal: building a model that allows us to discover the clusters based on the use of a similarity measure (e.g., inter-cluster and intra-cluster distance).

To address the aforementioned challenges facing the use and implementation of clustering approaches, the following questions need to be answered:

- What are the drawbacks and advantages of clustering approaches?
- How do we implement the clustering method to allow the discovery of the natural clusters in the data samples?
- What are the criteria used to choose the clustering approach with respect to the constraints of the data samples?
- How do we verify the quality of the discovered clusters?
- How do we determine the suitable number of clusters?

2.1 K-Means

How It Works

K-means uses a similarity measure to find the clusters gathering the data points (objects). The similarity measure is defined as the distance between the data points and the cluster centers, or centroids. Several distance metrics can be used for the similarity measure, such as Euclidian distance, Manhattan distance, or weighted Euclidian distance. A cluster gathers data points that have the smallest distance to its centroid. K-means works using the following six steps:

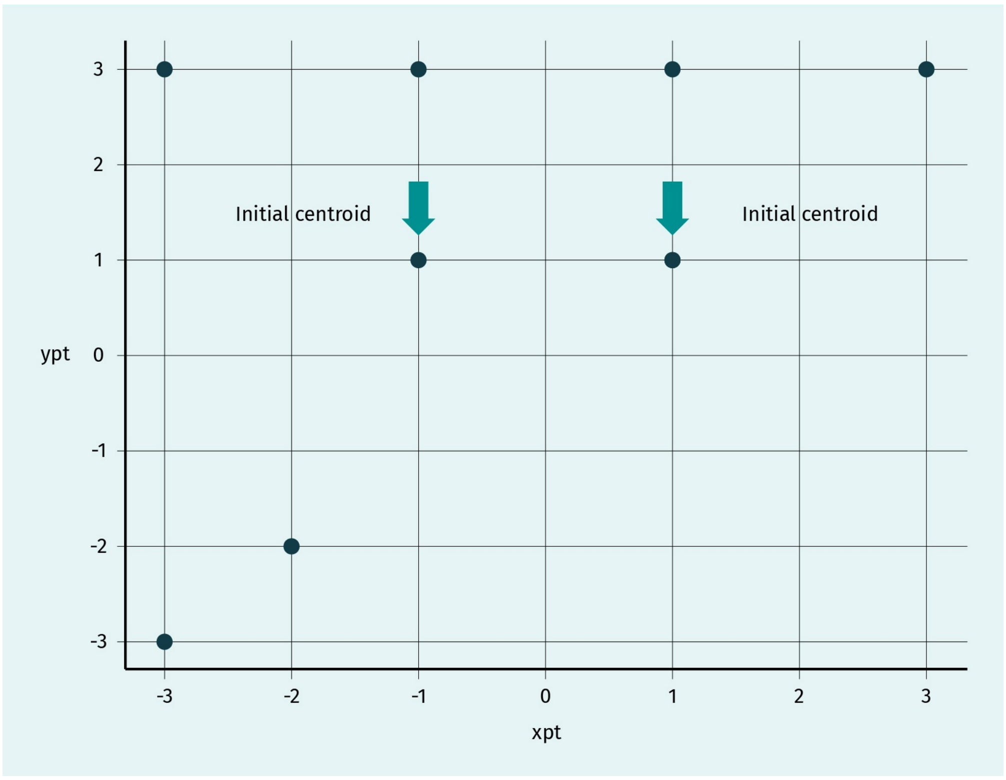
1. Choose the number of clusters, k, to be found since k is specified by the user.
2. Choose random k data points to be the initial centroids (seeds) of the clusters.
3. For each data point x belonging to the training set X, compute the distance in the n-dimensional space from x to each centroid of the k clusters.
4. Assign x to the closest cluster, i.e., the data points that possess the smallest distance with respect to a certain cluster will be assigned to this cluster.
5. Re-compute the centroids of the k clusters using the data points that have been assigned to each cluster, i.e., compute the mean of all the data points belonging to a cluster. Alternatively, a data point that is closest to the cluster's mean can be chosen as a representative of this cluster. This is similar to the cluster's median point and is less prone to outliers.
6. Repeat steps four and five until there is no change in the cluster assignments (convergence criterion), i.e., no or weak re-assignments. This can be achieved by calculating the difference (distance) between each cluster's old and new centroids.

When this difference is less than a predefined threshold, the cluster is considered to be stable. The stop criterion can also be defined by the maximum number of iterations, which is normally defined as 50. It is possible to use the square sum error (SSE) as a stop criterion when the decrease in SSE after an iteration is less than a predefined threshold.

SSE is calculated as follows: 1) compute the sum of the distances from all the data points belonging to a cluster to their corresponding centroid, 2) square this sum, 3) repeat the process for each of the k clusters, and 4) sum all of the computed squaring for the k clusters. This is one way to determine the quality of the clustering. There are also other metrics, such as the Elbow Criterion and the Silhouette Score, which will be discussed later in this unit.

In order to explain k-means, let us use the following simple toy example illustrated in the figure below.

Figure 9: Toy Example of Data Points to Be Clustered Using K-Means



Source: Sayed-Mouchaweh (2021).

Let us set k to 2 and choose randomly from the training data points the initial centroid to each of the cluster (illustrated in the figure above) points [1, 1] and [—1, 1]. In the first iteration, we will look to compute the distances from each of the training data points to each cluster centroid. The Euclidian distance between two points, $a = (a_x, a_y)$ and $b = (b_x, b_y)$, can be calculated by

$$d\left(a, b\right) = sqrt\left[\left(a_x - b_x\right)^2 + \left(a_y - b_y\right)^2\right]$$

The table below (on the left) shows these distances. The data points with the minimum distance to a cluster's centroid will be assigned to this cluster as it is shown in the table below (on the right).

Table 4: Assigned Points to Each of the Two Clusters in the First Iteration of K-Means

Distances to centroids			Assigned points to the clusters		
	[1, 1]	[—1, 1]		[1, 1]	[—1, 1]
[1, 1]	0	2	[1, 1]	x	
[1, 3]	2	2.83	[1, 3]	x	

[—1, 3]	2.83	2	[—1, 3]		x
[3, 3]	2.83	4.47	[3, 3]	x	
[—1, 1]	2	0	[—1, 1]		x
[—2, —2]	4.24	3.16	[—2, —2]		x
[—3, —3]	5.66	4.47	[—3, —3]		x
[—3, 3]	4.47	2.83	[—3, 3]		x

Source: Sayed-Mouchaweh (2021).

Next, the new centroid for each cluster created in the first iteration will be computed as the mean value of the points assigned to each cluster. For instance, the new centroid of cluster with the centroid [1, 1] will be: $[(1\ +\ 1\ +3)/3,\ (1\ +\ 3\ +\ 3)/3]\ =\ [1.67,\ 2.34]$. The distances of the data points according to each of the two new centroids are computed, and those whose distance to a centroid is the minimum are assigned to the corresponding cluster, as illustrated in the table below.

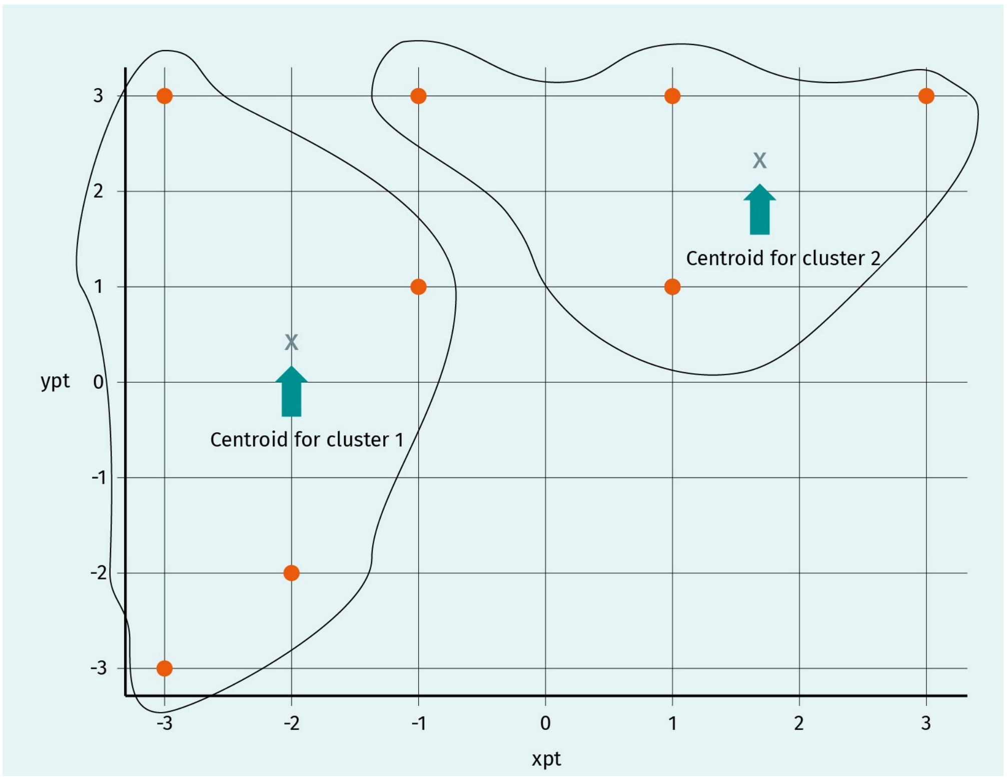
Table 5: Assigned Points to Each of the Two Clusters in the Second Iteration of K-Means

Distances to the two centroids			Assigned points to the clusters		
	[—2, 0.4]	[1.67, 2.34]		[—2, 0.4]	[1.67, 2.34]
[1, 1]	3.06	1.49	[1, 1]		x
[1, 3]	3.97	0.94	[1, 3]		x
[—1, 3]	2.79	2.75	[—1, 3]		x
[3, 3]	5.64	1.49	[3, 3]		x
[—1, 1]	1.17	2.98	[—1, 1]	x	
[—2, —2]	2.40	5.68	[—2, —2]	x	
[—3, —3]	3.54	7.09	[—3, —3]	x	
[—3, 3]	2.79	4.71	[—3, 3]	x	

Source: Sayed-Mouchaweh (2021).

The figure below shows the obtained clusters in the second iteration.

Figure 10: Obtained Clusters with Their Centroids by K-Means in the Second Iteration



Source: Sayed-Mouchaweh (2021).

The same process will be carried out in the third iteration, the new centroids will be computed using the new data points assigned to the two clusters. The new centroids are [—2.25, —0.25] and [1, 2.5]. The points will then be assigned to the cluster for which they have the minimum distance, as illustrated in the table below.

Table 6: Assigned Points to Each of the Two Clusters in the Third Iteration of K-means

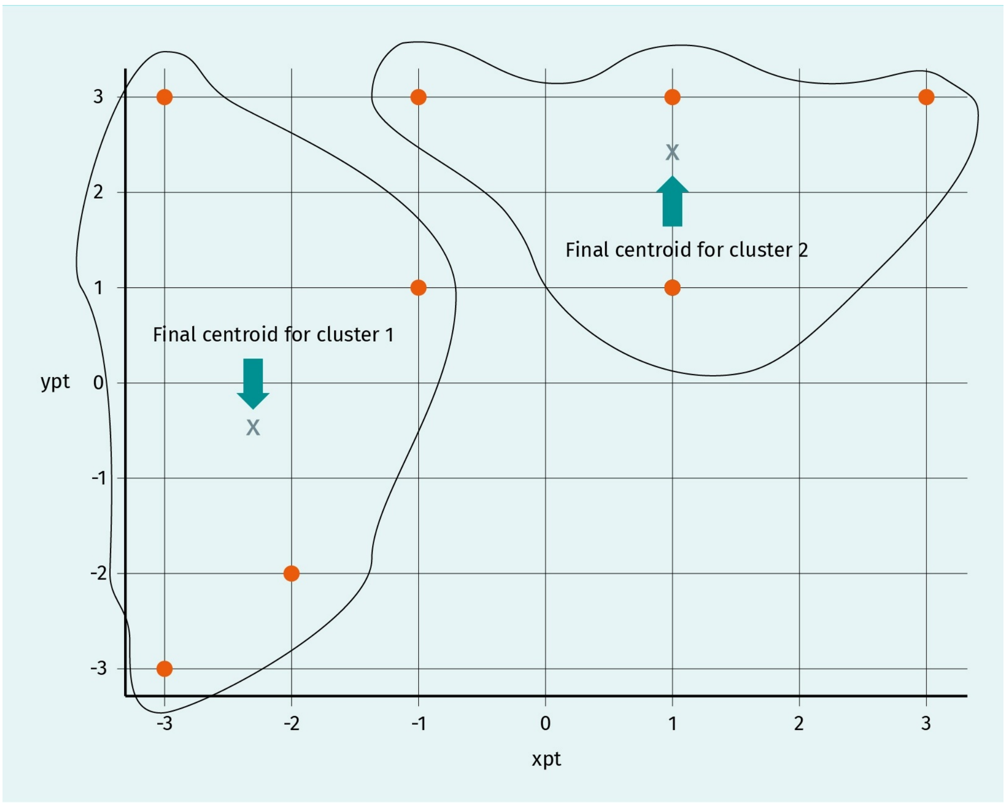
Distances to the two centroids			Assigned points to the clusters		
	[—2.25, —0.25]	[1, 2.5]		[—2.25, —0.25]	[1, 2.5]
[1, 1]	3.48	1.5	[1, 1]		x
[1, 3]	4.6	0.5	[1, 3]		x
[—1, 3]	3.48	2.06	[—1, 3]		x
[3, 3]	6.17	2.06	[3, 3]		x
[—1, 1]	1.77	2.50	[—1, 1]	x	
[—2, —2]	1.77	5.41	[—2, —2]	x	
[—3, —3]	2.85	6.80	[—3, —3]	x	

[—3, 3]	3.34	4.03	[—3, 3]	x	

Source: Sayed-Mouchaweh (2021).

In the fourth iteration, we notice that the cluster centroids and, hence, the assigned points do not change any more. Therefore, k-means stops at this iteration and provides the results clusters depicted in the figure below.

Figure 11: Obtained Clusters and Their Centroids by K-Means in the Third and Last Iteration

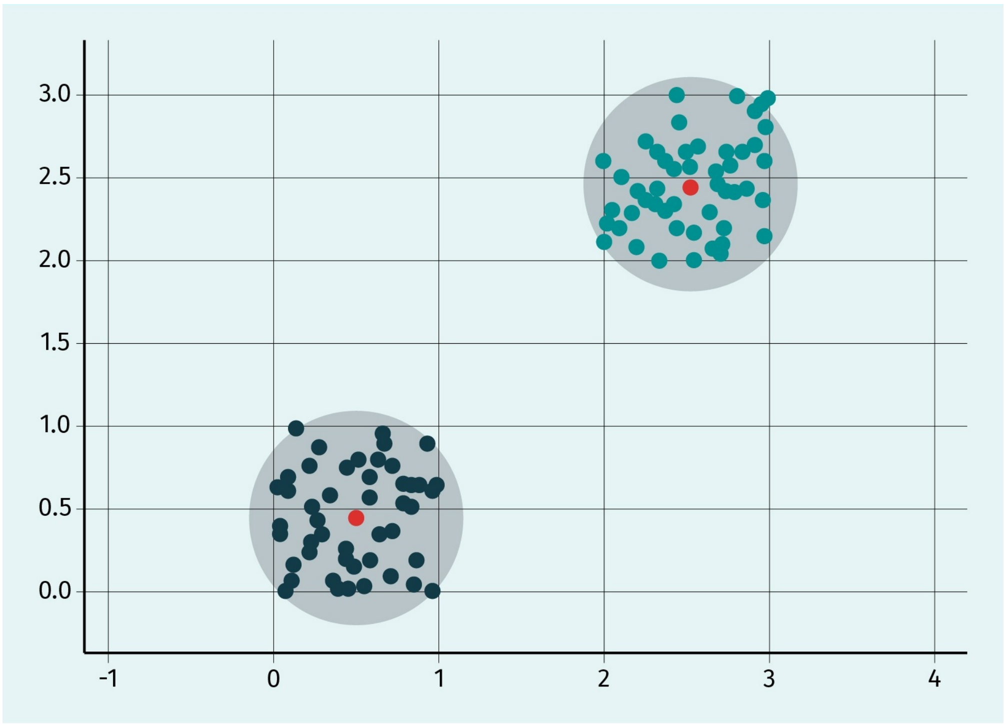


Source: Sayed-Mouchaweh (2021).

Let us now use k-means in sklearn.cluster in Python and let us apply the following code in order to generate two **Gaussian clusters** well separated in two-dimensional space. Applying this code will provide the following clustering illustrated in the figure below.

Gaussian clusters
The Gaussian clusters include data points that are generated by normal probability functions with two parameters: the mean value (or gravity center) and a variance-covariance matrix.

Figure 12: Two Gaussian Clusters with Their Centroids (Red Points) Obtained by K-Means



Source: Sayed-Mouchaweh (2021).

We can also display the clusters zones where data points are considered to be in these clusters. We can see that these zones are hard in the sense that data points belong or do not belong to a cluster whatever their position with respect to the cluster centroid. The steps are demonstrated below:

1. We import the required Python libraries.

   Code
   ```
   import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   from sklearn.cluster import KMeans
   ```

2. We generate data points of two Gaussian clusters, X and Y, well separated. X has a mean value = 0 and standard deviation of value = 1, while Y has a mean value of = 2 and standard deviation of value = 1.

   Code
   ```
   # generate random variables
   X= np.random.rand(50,2)
   Y= 2 + np.random.rand(50,2)
   ```

```
# gather random variable in a dataframe
Z= np.concatenate((X,Y))
df1 = pd.DataFrame(Z, columns=['xpt', 'ypt'])
```

Let us have a glimpse at what the data look like at this point. We generated a dataframe with two columns which are the features, and also the x and y coordinates in the plots we are going to create. The first 50 rows were drawn from one distribution, and the last 50 rows were drawn from a second distribution.

Code

```
df.head()
# console output:
#  xpt     ypt
# 0 0.843469 0.719464
# 1 0.283066 0.002213
# 2 0.327358 0.211112
# 3 0.087454 0.286058
# 4 0.606084 0.568120
```

3. Now, we are ready to apply k-means to the generated data points with k (number of clusters) equal to 2. This application allows us to compute the centroids of the two clusters and their assigned points. The labels assigned to the two clusters are 0 and 1, respectively.

Code

```
# clustering
kmeans = KMeans(n_clusters=2, random_state=0).fit(Z)

# extract centroids of clusters into a dataframe
centers = kmeans.cluster_centers_
centroids = pd.DataFrame(centers,columns=['xpt','ypt'])

# extract cluster labels
lab = kmeans.labels_

# add cluster information to the dataframe
df['lab']=lab
```

Again, let us have a look at what the data look like at this point.

Code

```
df.head()
# console output:
#  xpt     ypt     lab
# 0 0.684751 0.846389 1
# 1 0.572355 0.144059 1
# 2 0.752941 0.768759 1
# 3 0.722927 0.790472 1
# 4 0.105875 0.761570 1
```

We added a new column, which contains the prediction for one of the two clusters ("0" and "1") for each row as it was predicted by the k-means algorithm.

4. Next, we gather the resulting information in order to visualize the found clusters in a plot. We calculate the radius around each cluster centroid, which includes the data points that were predicted to belong to this cluster. We do this by calculating the distance between each data point and the corresponding cluster centroid and choosing the maximum of these distances.

Code
```
radii = [cdist(df[lab == i].iloc[:,[0,1]], [center]).\
   max() \
        for i, center in enumerate(centers)]
```

When we take a look at the results of this calculation, we see that we found the maximum radius around each cluster centroid which includes all corresponding data points.

Code
```
print(radii)
# console output:
# [0.7096052458280815, 0.6578703939400616]
```

5. Finally, we can display the zones of the obtained clusters, where data points are considered to be in these clusters.

Code
```
# create a figure and axes
fig, ax = plt.subplots(figsize=(6,4))

# add data points
ax.scatter(x=df['xpt'], y=df['ypt'], c=df['lab'])

# add cluster centroids
ax.scatter(centroids['xpt'], centroids['ypt'])

# set the axis scale on both axes equally
ax.axis('equal')

# draw a circle around each cluster centroid
for c, r in zip(centers, radii):
   ax.add_patch(plt.Circle(c, r, fc='#CCCCCC', \
                       lw=3, alpha=0.5))
# show the plot
plt.show()
```

Choosing the Number of Clusters k

The number of clusters to discover in the dataset is defined by the user. If the number of clusters is too high, this will lead to the problem of overfitting (eventually, one cluster for each data point). Too small a number of clusters leads to a poor representation of the data structure (high heterogeneity within a cluster). Therefore, it is important to determine the number of clusters correctly, i.e., how the user defines k properly. In general, there are three ways to help define the number of clusters, k. They are as follows:

1. Visualization, where the data points are plotted in order to better set the number of clusters
2. Domain knowledge, where knowledge or expertise in the area of the application can be used to define the number of clusters, e.g., if the dataset is related to spam detection, two clusters, spam or genuine, are the optimal number to discover
3. Data-driven approaches that use a metric to determine the quality of the obtained clusters for different values of k such as the Elbow method and Silhouette score

Elbow method

The elbow method uses an objective function to explain the variation of data points into clusters. When the number of clusters increases, the model will fit or better explain the data variation since it uses more parameters. However, when the number of clusters increases too much, the problem of overfitting will occur, increasing the complexity and computation time while not decreasing the added information to the same extent. This is because the newly created clusters will divide the actual or natural clusters and fit more closely to the noises within the data.

Thus, the Elbow method looks for the number of clusters k for which adding more clusters will not add considerable information to increase the quality of the clustering and give better modeling results with respect to the data variation. To this end, the value of the objective function to explain the variation is computed for different values of k and then the value of k that gives the cutoff or "elbow" of the curve, when plotting the cost against k, is chosen.

There are different measures to explain the variance, such as the **total within-cluster sum of squares (WSS)**. It is computed by

Total within-cluster sum of square (WSS)
This term refers to the sum of the squared distances from all data points of a cluster to its centroid, so this measure indicates the cohesion of data points to their cluster (cluster compactness).

$$WSS = \sum_{j=1}^{k} \sum\nolimits_{x_i \in c_j} \left(x_i - c_j\right)^2$$

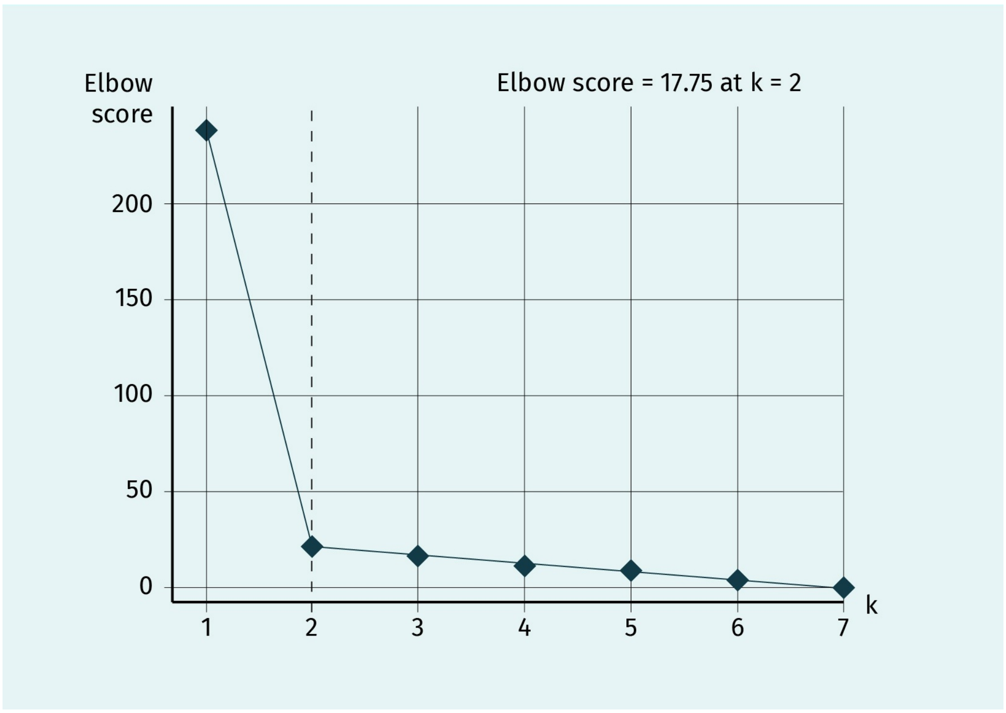
where x_i is a data point assigned to cluster C_j, and c_j is its center of gravity (centroid). The lower the WSS value, the better the clustering. Let us take the example of two Gaussian classes (illustrated in the figure above) and let us use Elbow method to choose the number of clusters k. We can use the tool in the “yellowbrick.clusters” library (visual analysis and diagnostic tools) in Python to generate an Elbow graph and score for different values of k. If it is not already installed, you need to install it using, for instance, `pip install yellowbrick.clusters`. Then import the Python function “KElbowVisualizer” that is used to which is an implementation of the “Elbow” method. “KElbowVisualizer” also displays the amount of time required in order to train the clustering model per k as a dashed green line, but is can be hidden by setting “timings” to “False”. The following code allows us to generate an Elbow graph and score for the example of two Gaussian classes.

Code

```
# import libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

# create sample data
X= np.random.rand(50,2)
Y= 2 + np.random.rand(50,2)
Z= np.concatenate((X,Y))

# create a k-Means model an Elbow-Visualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,8), \
   timings=False)

# fit the visualizer and show the plot
visualizer.fit(Z)
visualizer.show()
```

The figure below shows the Elbow graph and score using the aforementioned code. The elbow score is detected using the “knee point detection algorithm” developed in Satopaa et al. (2011). We can see that $k = 2$ is the Elbow value that corresponds to the data point structure for the example of two Gaussian clusters. It is worth noting that the Elbow score only considers the clusters cohesion or compactness and not their separability. In addition, sometimes it is not possible to find the Elbow score. Therefore, it is recommended to use a metric that considers cohesion and separability, such as the Silhouette score.

Figure 13: Elbow Method Applied to Two Gaussian Clusters to Find the Suitable K



Source: Sayed-Mouchaweh (2021).

Silhouette score

Silhouette is a technique used to assess the quality of obtained clusters. The **silhouette score** for data point x_i in cluster C_i is computed using the following four steps:

Silhouette score
The Silhouette score is a measure of how similar data points are within the same clusters (cohesion) and how dissimilar data points of different clusters are (separation).

1. Compute the mean distance $a(i)$ between data point x_i and all other data points in the same cluster C_i. This can be computed by

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} dist(x_i, x_j)$$

   where $|C_i|$ is the number of data points assigned to cluster C_i, and dist(x_i, x_j) is the distance between data point x_i and data point x_j. $a(i)$ indicates how well x_i is assigned to its cluster C_i. The smaller the value of $a(i)$, the better the assignment of x_i to its cluster C_i.

2. Compute the mean dissimilarity of data point x_i of cluster C_i to other clusters C_j. It is defined as the mean of the distance from x_i to all data points in cluster C_j, $C_j \neq C_i$ and is computed as follows:

$$\frac{1}{|C_j|} \sum_{x_j \in C_j} dist(x_i, x_j)$$

3. Compute the smallest mean distance $\mathrm{b(i)}$ between data point $\mathrm{x_i}$and any other data point in any other cluster as follows:

$$b(i) = \min_{j \neq i} \frac{1}{|C_j|} \sum_{x_j \in C_j} dist\left(x_j, x_j\right)$$

4. Finally, the silhouette score $\mathrm{s(i)}$ for the data point $\mathrm{x_i}$ in cluster $\mathrm{C_i}$ is computed as the combination of $\mathrm{a(i)}$ and $\mathrm{b(i)}$ by

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)}, if a(i) < b\left(i\right) \\ 0, if a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1, if a(i) > b\left(i\right) \end{cases}$$

Therefore, $\mathrm{s(i)}$ is a metric that has its values range from —1 to 1 as follows:

- $\mathrm{s(i)}$ close to 1 means $\mathrm{x_i}$ is appropriately clustered since the similarity measure $\mathrm{a(i)}$ is small and dissimilarity measure $\mathrm{b(i)}$ is large.
- $\mathrm{s(i)}$ close to zero means that $\mathrm{x_i}$ is on the border of two natural clusters.
- $\mathrm{s(i)}$ close to —1 (negative values) means that $\mathrm{x_i}$is badly clustered since its dissimilarity with other clusters is bigger than its similarity with the data points of its proper cluster.

The silhouette will be computed for each data point of each cluster and will be displayed in a figure. The distribution of the silhouette values of the data points in a cluster tells us whether those points are well clustered by observing the silhouette they manifest. It is worth mentioning that the mean of the silhouette can be computed over all the data points in each cluster as follows:

$$\overline{s}\left(C_i\right) = \frac{1}{|C_i|} \sum_{X_i \in C_i} s\left(X_i\right)$$

The maximum value of all the mean values of the silhouette over all the clusters can then be defined as the silhouette score or coefficient (Kaufman & Rousseeuw, 2009) as follows:

$$s(k) = \max_{j = 1, . . , k} \left(\overline{s}\left(C_j\right)\right)$$

It is used to determine the quality of the selected number of clusters k as follows:

- $\mathrm{s(k)}$ close to 1 means data points are well clustered.
- $\mathrm{s(k)}$ close to 0 means clusters are indifferent, i.e. too close to each other.
- $\mathrm{s(k)}$ negative and close to —1 means the data points are badly clustered.

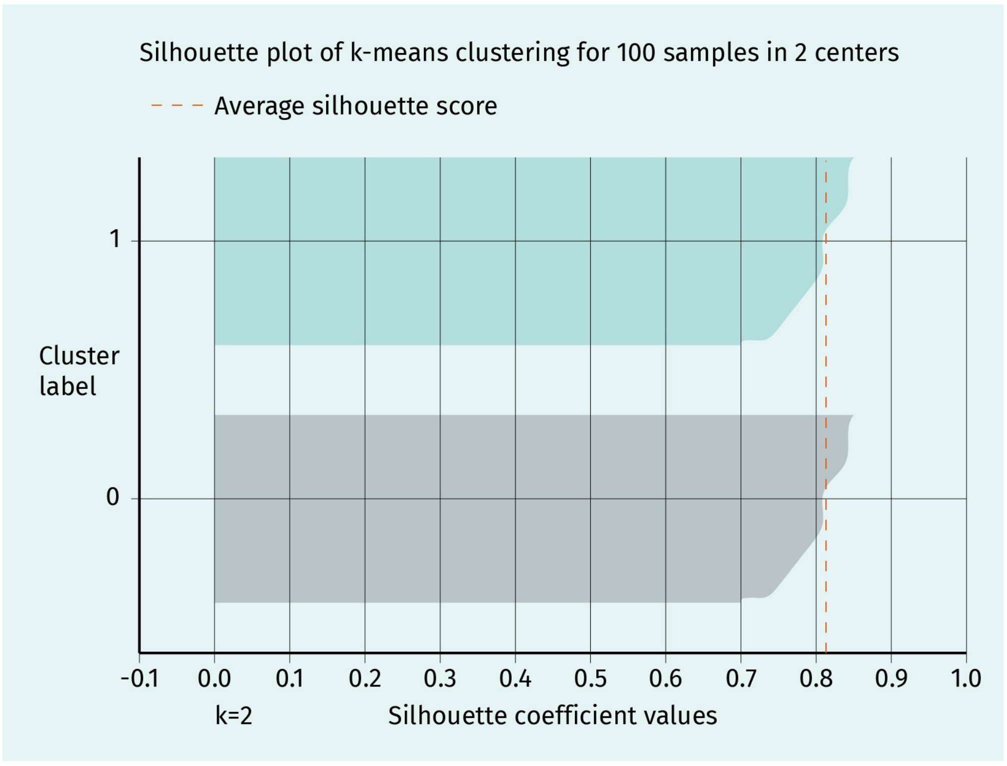
Let us take the example of the two Gaussian clusters and compute the silhouette score $\mathrm{s(k)}$ and display the mean silhouette, $\mathrm{s(C_i)},\ \mathrm{i} = 1,\ \ldots,\ \mathrm{k}$, for each cluster with two values of k : 2 and 3 using Python. The following code allows us to compute $\mathrm{s(k = 2)}$:

Code

```
# import libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.metrics import silhouette_score

# generate sample data
X= np.random.rand(50,2)
Y= 2 + np.random.rand(50,2)
Z= np.concatenate((X,Y))

# conduct a k-Means clustering
model = KMeans(n_clusters=2, random_state=0).fit(Z)

# extract labels, i.e. cluster associations
lab=model.labels_

# calculate the overall Silhouette score
S = silhouette_score(Z, lab)
print(S)
# console output: 0.8123455046726186

# generate, fit and show a Silhouette visualizer
visualizer = SilhouetteVisualizer(model, colors='yellowbrick')
visualizer.fit(Z)
visualizer.show()
```

The figure below shows the mean silhouette $s(C_1)$ and $s(C_2)$ and the silhouette score for $k = 2$ for the example of the two Gaussian clusters. We can see that data points in each cluster match their corresponding cluster very well with a high $s(k = 2) = 0.82$.

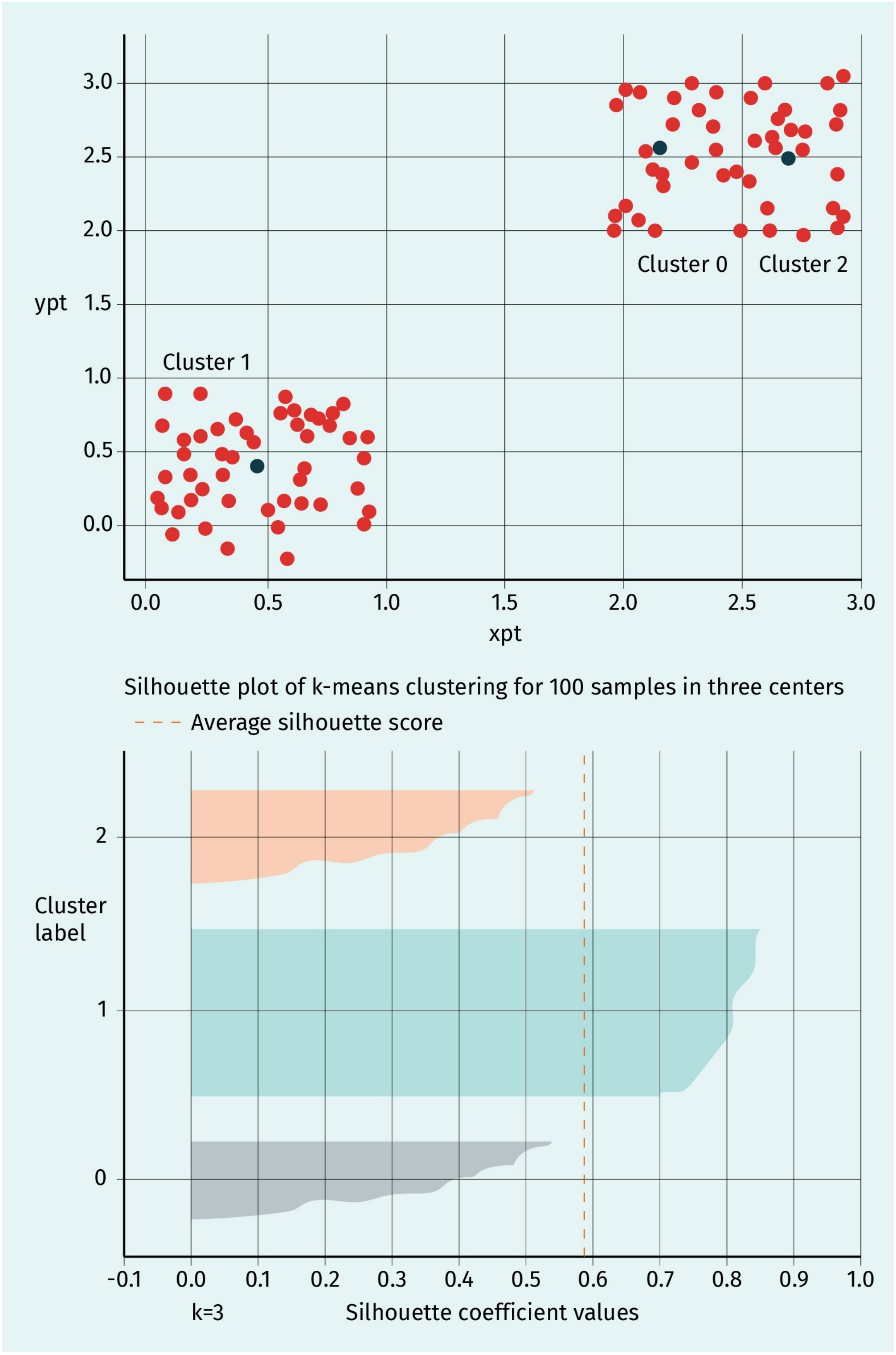
Figure 14: Silhouette Measure for the Example of the Two Gaussian Clusters



Source: Sayed-Mouchaweh (2021).

The figures below shows the mean silhouette $\mathrm{s}(\mathrm{C}_1),\ \mathrm{s}(\mathrm{C}_2)$, and $\mathrm{s}(\mathrm{C}_3)$,and the silhouette score for $\mathrm{k}=3$ for the example of the two Gaussian clusters. Indeed, a natural cluster has been divided into two close clusters as we can see in the figure below. We can see that the silhouette measure for the silhouette of the two divided clusters (0 and 2 in the figure below) decreased largely, indicating that the clusters are not well separated. Moreover, the silhouette score $\mathrm{s}(\mathrm{k})=3$ has been reduced to 0.58, compared to $\mathrm{s}(\mathrm{k}=2)$ which was 0.82, indicating that $\mathrm{k}=2$ is much better than $\mathrm{k}=3$.

Figure 15: Two Gaussian Clusters and Corresponding Silhouette Measure



Source: Sayed-Mouchaweh (2021).

Advantages and drawbacks

K-means is a very popular clustering method. Thanks to its simplicity and efficiency, it is easy to understand and implement. It is considered to be a linear learning algorithm with respect to the size of the dataset (number of data points). Indeed it is time complexity $\mathrm{O(t.k.n)}$ where t is the number of iterations, k is the number of clusters, and n is the number of data points. Since t (<50) and k are small, then the time complexity $\mathrm{O(n)}$ is linear with n. It is worth mentioning that there are extensions of k-means allowing us to deal with large datasets, such as Clustering Large Applications (CLARA) (Popat & Emmanuel, 2014). CLARA applies a sampling method to perform clustering using a small batch of data points and repeats this sampling for a fixed number of times in order to minimize the sampling bias.

However, k-means also suffers from several weaknesses. First, it is sensitive to outliers (i.e., data points that are far away from all other data points) that are generated due to some errors in the recorded data or noises, or simply valid data points that do not fit into a cluster. Therefore, it is useful to clean the dataset by removing data points that are far away from the other data points. It is worth mentioning that there is a robust version of k-means that avoids this weakness: Partition Around Medoids (PAM) (Bishop, 2006). In PAM, the centroids are selected from representative data points. Then, the data points are assigned to clusters by minimizing their sum of pairwise dissimilarities. Thanks to the latter, PAM is robust against outliers and noises, in contrast to k-means, which searches for minimizing the sum of squared distances of data points to centroids. Additionally, it is sensitive to the initial seeds (centroids); if we change the initial seeds, the obtained clusters will be different. Therefore, if the seeds are selected in the wrong way, this can impact the quality of the obtained clusters. Finally, it is not suitable for discovering clusters that are not of hyper-spherical shape.

2.2 Gaussian Mixture Model Clustering

How It Works

K-means is a hard clustering method since it considers that all data points of a cluster belong to this cluster with the same certainty or confidence. For instance, a data point close to a cluster center is considered to belong to this cluster with the same confidence or certainty as a data point located in the cluster's periphery. Therefore, k-means assigns a value of 1 if a data point belong to a cluster, whatever its position in this cluster (close to the cluster centroid or at the cluster periphery) and 0 to other data points that do not belong to this cluster. However, in real world problems, and in order to take into account the uncertainty attached to data points, such as noises, it is important to quantify the probability or the membership value of a data point to a cluster.

This method is particularly useful when clusters are overlapped and a data point may belong to different clusters. To take the cluster overlapping into account, the clustering method must provide a probability membership value of each point to different clusters according to its position with respect to these clusters. This can be done by a soft or probabilistic clustering method, such as Gaussian mixture clustering model (GMM).

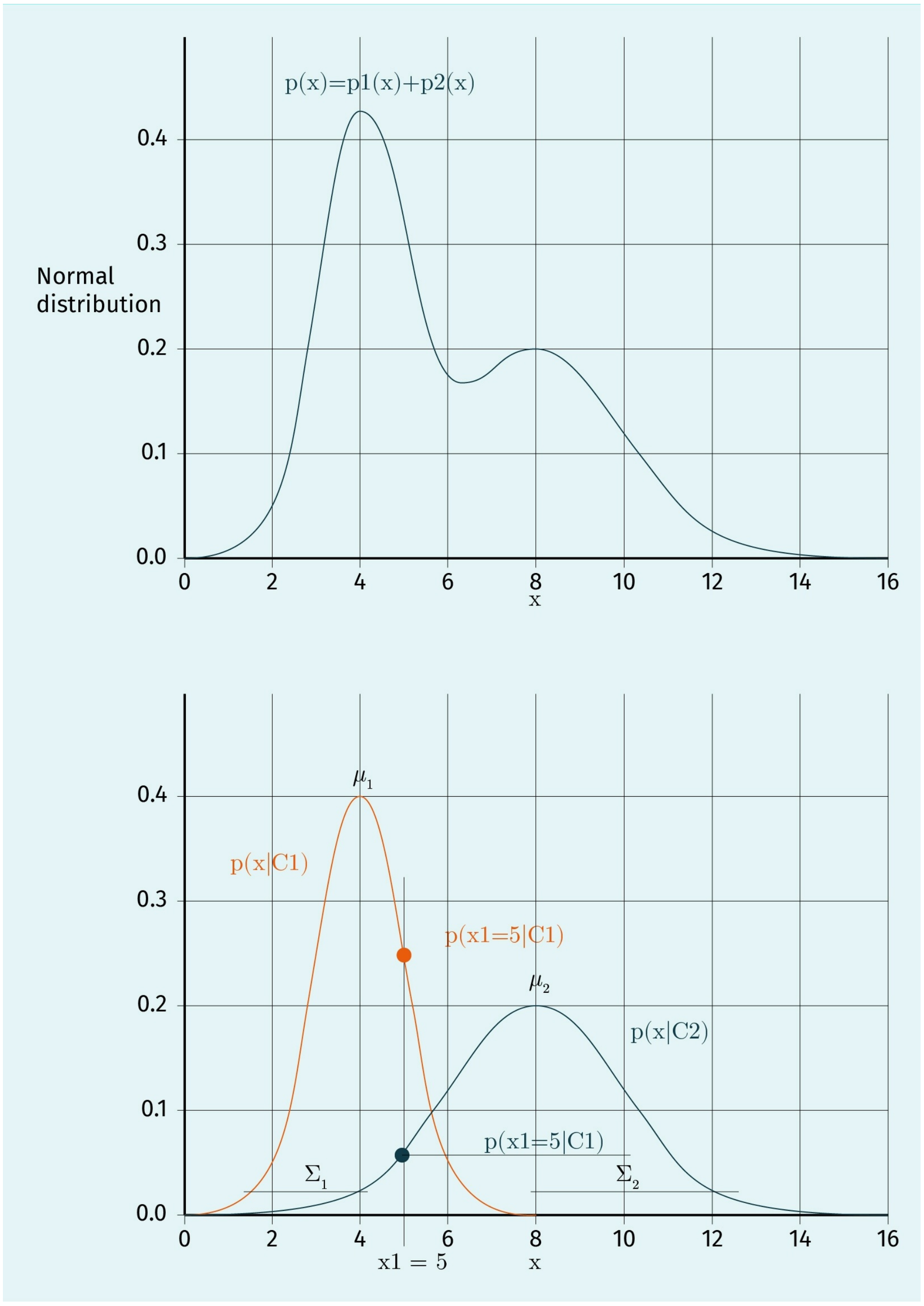
The functioning of GMM is very similar to k-means in the sense that it is a prototype based method where each cluster is represented by a **prototype**. In k-means the prototype is the clusters' centroids, while in GMM, it is a Gaussian or normal probability density $p(x|C_j)$, $j = 1, ..., k$, represented by its two parameters: the mean value μ_j and the variance-covariance matrix $\sum_j$. In GMM, a cluster is not only represented by its mean as in k-means, but also by its variance-covariance matrix. Thus, GMM is considered to be an extension of k-means.

Prototype
A prototype is a (preliminary) representative data point of a cluster, such as its gravity center in the k-means algorithm or its most representative data point in the PAM (Partition Around Medoids) algorithm.

Let us explain how GMM works. Let X be a dataset of n data points, $x_1, x_2, ..., x_n$, described in dimensional d space. Let us suppose that we have k clusters to discover in X using GMM. GMM do that in the following three steps:

1. Represent each cluster C_j, $j = 1, ..., k$, with a Gaussian or normal probability density $p(x|C_j)$, $j = 1, ..., k$, and its prior probability π_j.
   π_j defines the importance of cluster C_j with respect to the other clusters in the dataset. A cluster importance depends on the number of data points belonging to this cluster. The more data points a cluster has, the greater the importance of this cluster. $p(x|C_j)$ is the conditional Gaussian probability that a data point x belongs to a cluster C_j. The figure below shows an example of two clusters, C_1 and C_2, represented as two Normal or Gaussian probability densities, $p(x|C_1)$ and $p(x|C_2)$, in a one-dimensional feature space. In this example, $p(x|C_j)$ can be calculated for any data point x if the cluster's mean value μ_j and variance-covariance matrix $\sum_j$ are known.

Figure 16: Two Clusters Represented by Two Gaussian Probability Densities and Their Mixture Probability Density



Source: Sayed-Mouchaweh (2021).

2. Define the mixture probability $p(x)$ that a data point x belongs to k clusters. It can be represented as weighted sum of the probability densities, $p(x|C_j)$, $j = 1, ..., k$,of the k clusters as follows:

$$p(x) = \sum_{j=1}^{k} \left(p\left(xC_j\right) . \pi_j\right)$$

The figure above shows the mixture probability $p(x)$ of the two clusters with the probability densities $p(x|C_1)$ and $p(x|C_2)$. For this example, this cluster mixture probability is the sum of the individual cluster probabilities:

$$p(x) = p(xC_1) . \pi_1 + p(xC_2) . \pi_2 = p_1(x) + p_2(x)$$

The mixture probability is used because we do not know the cluster of a data point x in advance; thus, the estimation of the clusters' parameters is refined (updated) iteratively.

3. Estimate the normal or Gaussian probability density parameters $(\mu_j, \sum_j, \pi_j)$ for each cluster C_j.
   To this end, GMM uses the expectation-maximization (EM) algorithm to perform this estimation iteratively in two steps: expectation and maximization. This is very similar to the k-means algorithm. First, we define cluster centroids at random. Next, we calculate the mean and, this time, also the variance-covariance matrix over all data points. We then update our first estimates (or guesses) and repeat the process. The difference from k-means is that we calculate both the mean and the "spread" of clusters. We also do this with probabilities belonging to that cluster rather than all-in-or-out clusters as we did with k-means.
   In the expectation step, the parameters $(\mu_j, \sum_j, \pi_j)$ of each cluster C_j are initialized. Then, for each data point x from X, the probability to belong to each cluster, $p(x|C_j)$, $j = 1, ..., k$, is calculated using the initialization parameters for each cluster C_j. The posterior probability of x is then calculated as follows:

$$p\left(C_j x\right) = \frac{p\left(x\middle|C_j\right) . \pi_j}{\sum_{i=1}^{k} \left(p\left(x\middle|C_i\right) . \pi_i\right)}$$

   It is used to assign a data point x to a cluster $C_j$if the posterior probability $p(x|C_j)$ of x in C_j is the biggest one with respect to the other posterior probabilities:

$$p(x\,|\,Ci), i = 1, ..., k, i \neq j$$

   In the maximization step, the parameters $(\mu_j, \sum_j, \pi_j)$ of each cluster C_j, will be updated using the weighted data points by the posterior probabilities as follows:

$$\mu_j = \frac{\sum_{i=1}^{n} p\left(C_j \middle| x_i\right) . x_i}{\sum_{i=1}^{n} p\left(C_j \middle| x_i\right)}$$

$$\Sigma_j = \frac{1}{\sum_{i=1}^{n} p\left(C_j \middle| x_i\right)} \sum_{i=1}^{n} p\left(C_j \middle| x_i\right) . \left(x_i - \mu_j\right)^T . \left(x_i - \mu_j\right)$$

$$\pi_j = \frac{\sum_{i=1}^{n} p\left(C_j \middle| x_i\right)}{n}$$

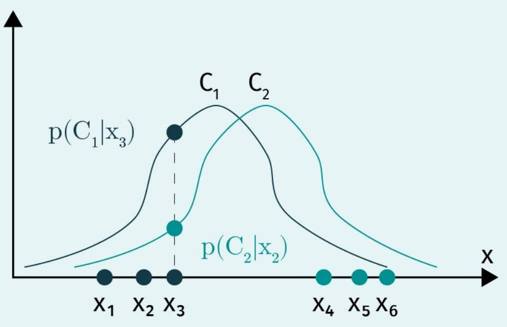
The process will restart with the expectation step until the EM converges to where no change occurs in the parameters, as in the case for k-means.

Let us take the example of the figure below showing two clusters in one-dimenstional space. In the Expectation-step, the posterior probabilities, $p(x|C_1)$ and $p(x|C_2)$ of each data point, $x_1,\ x_2,\ x_3,\ x_4,\ x_5,$ and x_6, in C_1 and C_2, are calculated. For instance, as illustrated in the figure below, $x3$ has $p(x_3|C_1)$ higher than $p(x_3|C_2)$. Therefore, its posterior probability $p(C_1|x_3)$ in C_1 will be higher than its posterior probability $p(C_2|x_3)$ in C_2. Therefore, x_3 will be assigned to C_1 since $p(C_1|x_3) > p(C_2|x_3)$. The same reasoning can be applied for the other data points of the figure below. Hence, x_1 and x_2 will be assigned to C_1 while x_4, $x_5$and x_6 will be assigned to C_2.

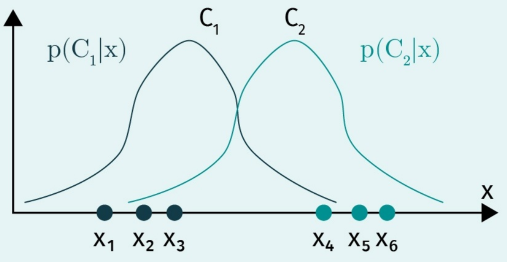
In the maximization step, the parameters $(\mu_1,\ \sum_1,\ \pi_1)$ of C_1 and $(\mu_2,\ \sum_2,\ \pi_2)$ of C_2 will be updated according to the data points assigned to each of them. Then, the expectation step will be repeated using the new calculated parameters, and the data points will be assigned to their corresponding clusters, as illustrated in the figure below. In the second iteration of the maximization step, the cluster parameters will be updated again using the new data points assigned to each of these two clusters. The expectation and maximization steps will be repeated until the stabilization of the both clusters' parameters. In the figure below, we can see that at each iteration of the EM algorithm, the Gaussian model (parameters) for each cluster fits its data points better.

Figure 17: Convergence of the Expectation-Maximization (EM) Algorithm

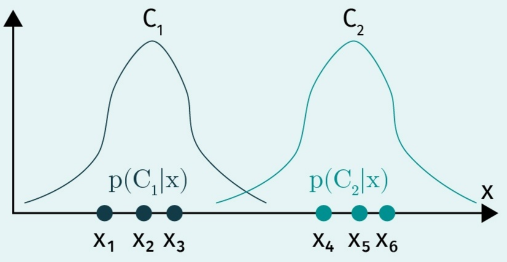
The posterior probabilities of x_1, x_2, and x_3 within C_1 are larger than those in C_2. Therefore, in the first iteration of EM, the Gaussian distribution of C_1 is shifted to the left. The same can be observed for x_4, x_5, and x_6, which reflects the Gaussian distribution of C_2 having been shifted to the right.



The posterior probabilities of x_1, x_2, and x_3 become larger after the parameters of C_1 have been updated during the maximization step. Therefore, in the second iteration of EM, the Gaussian distribution of C_1 is shifted to the left to better fit those points. The same can be observed for x_4, x_5, and x_6, which reflects the Gaussian distribution of C_2 having been shifted to the right.



The Gaussian distributions of C_1 and C_2 fit well to their corresponding points and EM converges. There is also no change in the parameters of both clusters.



Source: Sayed-Mouchaweh (2021).

Let us now use a GMM in Python to identify fuzzy clusters. In this example, we use a two-dimensional feature space. The steps are demonstrated below:

Code

```
Import the necessary libraries.
from sklearn import mixture
import numpy as np
import matplotlib.pyplot as plt
```

1. Generate three Gaussian clusters and display the obtained clusters.

Code

```
# generate sample data
X1 = 4 + np.random.rand(50,2)
X2 = 5 + np.random.rand(50,2)
X3 = 6 + np.random.rand(50,2)
Z = np.concatenate((X1,X2,X3))

# plot the sample data
plt.scatter(Z[:, 0], Z[:, 1], marker='+')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

2. Specify a Gaussian Mixture model, choose “full” for the parameter covariance_type. “full” allows the components to independently adopt any position and shape. Apply GMM in Python in order to obtain the clusters in the figure below. GMM provides the probability to belong to each of the obtained clusters. The code below shows the probability that the first five data points belong to each of the three obtained clusters: [[0. 0. 1.], [0. 0. 1.], [0.001 0. 0.999], [0. 0. 1.], [0. 0. 1.]]. For instance, the third data point has the probability vector [0.001 0. 0.999] of belonging to clusters C_1, C_2, and C_3. We can see that since the probability to belong to cluster C_3 is very high, it will be assigned to this cluster.

Code

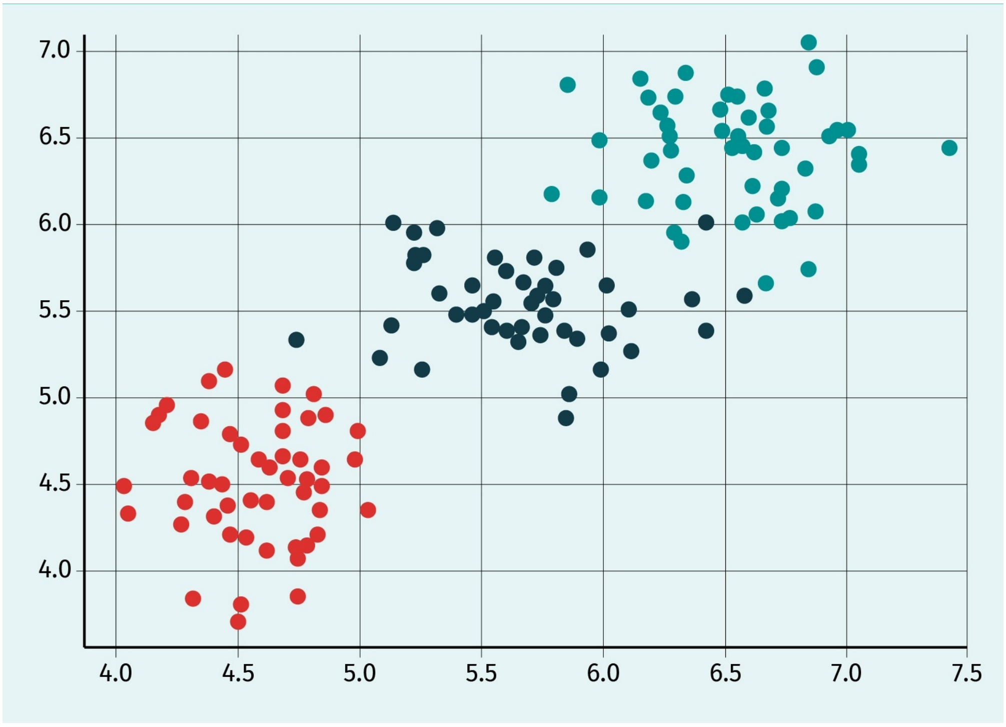
```
# specify Gaussian Mixture Model
gmm = mixture.GaussianMixture(n_components=3,
                              covariance_type='full')

# fit the model
gmm.fit(Z)

# extract the clusters predictions according to
# the highest probability
labels = gmm.predict(Z)

# show the predicted labels
print(labels)
# console output:
# [1 1 1 1 1 1 1 1 1 1 1 1...
# ...2 2 2 2 2 2 2 2 2 2 ...
# ...0 0 0 0 0 0 0 0 0 0 ]

# extract the probabilities to belong to a cluster
probs = gmm.predict_proba(Z)
print(probs)
# console output:
# [[1.85387989e-24 9.99998029e-01 1.97120198e-06]
# [6.54360391e-28 9.99999964e-01 3.56085199e-08]
# ...
# [1.00000000e+00 2.36479665e-39 2.81008490e-10]
# [9.99470972e-01 1.24217497e-24 5.29027653e-04]]
```

3. Display the results from the previous step visually.

Code

```
plt.scatter(x=Z[:,0], y=Z[:,1], c=labels, cmap='viridis')
plt.show()
```

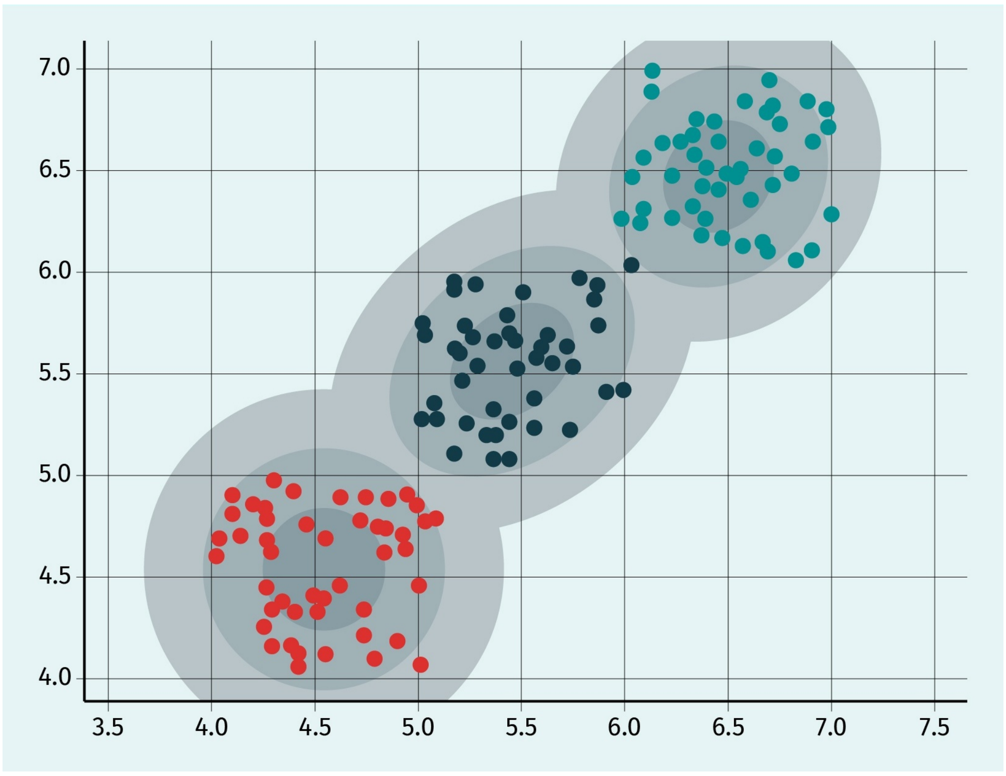
Figure 18: Clustered Obtained by Applying GMM on Data Points Including Three Gaussian Clusters



Source: Sayed-Mouchaweh (2021).

The figure below shows the membership zones created by the GMM for each cluster. We can see that several probability membership zones are created for each cluster by the GMM, allowing us to assign a higher probability membership to data points close to the cluster centroid.

Figure 19: Probability Membership Zones Generated by GMM



Source: Sayed-Mouchaweh (2021).

Choosing the Number of Clusters, Advantages, and Drawbacks

The GMM requires the definition of the number of clusters k in advance. As in the case of k-means, the quality of the obtained clusters can be tested using some meaningful criteria for different values of k. The value of k that maximizes those criteria are selected.

There are two criteria that can be used to test the quality of the obtained clusters: the Silhouette score, which was used for k-means in the previous section, and the Bayesian Information Criterion (BIC). BIC is a balance between the number of observations and the number of a model's parameters against its maximum likelihood value. The model here is GMM, and its parameters are $(\mu_j, \sum_j, \pi_j),\ j = 1, \ldots, k,$ for each of the k clusters. The maximum likelihood function L represents the probability that the model fits or best describes the data points. BIC can be computed by

$$BIC = \ln(n)\,.\,p - 2\ln(L)$$

where n is the number of data samples, p is the number of the model parameters, and L is the maximized value of the likelihood function of the model. The lower the BIC, the better the model to predict the data samples (clusters). The BIC penalizes the model with a large number of clusters in order to avoid overfitting issues.

Let us apply the code in Python to test of the quality of clusters obtained by the GMM for the example of the three Gaussian clusters. The steps are demonstrated below:

1. Import the required Python libraries.

   Code
   ```
   import numpy as np
   import matplotlib.pyplot as plt
   from sklearn import mixture
   from sklearn.metrics import silhouette_score
   ```

2. Generate the three Gaussian clusters.

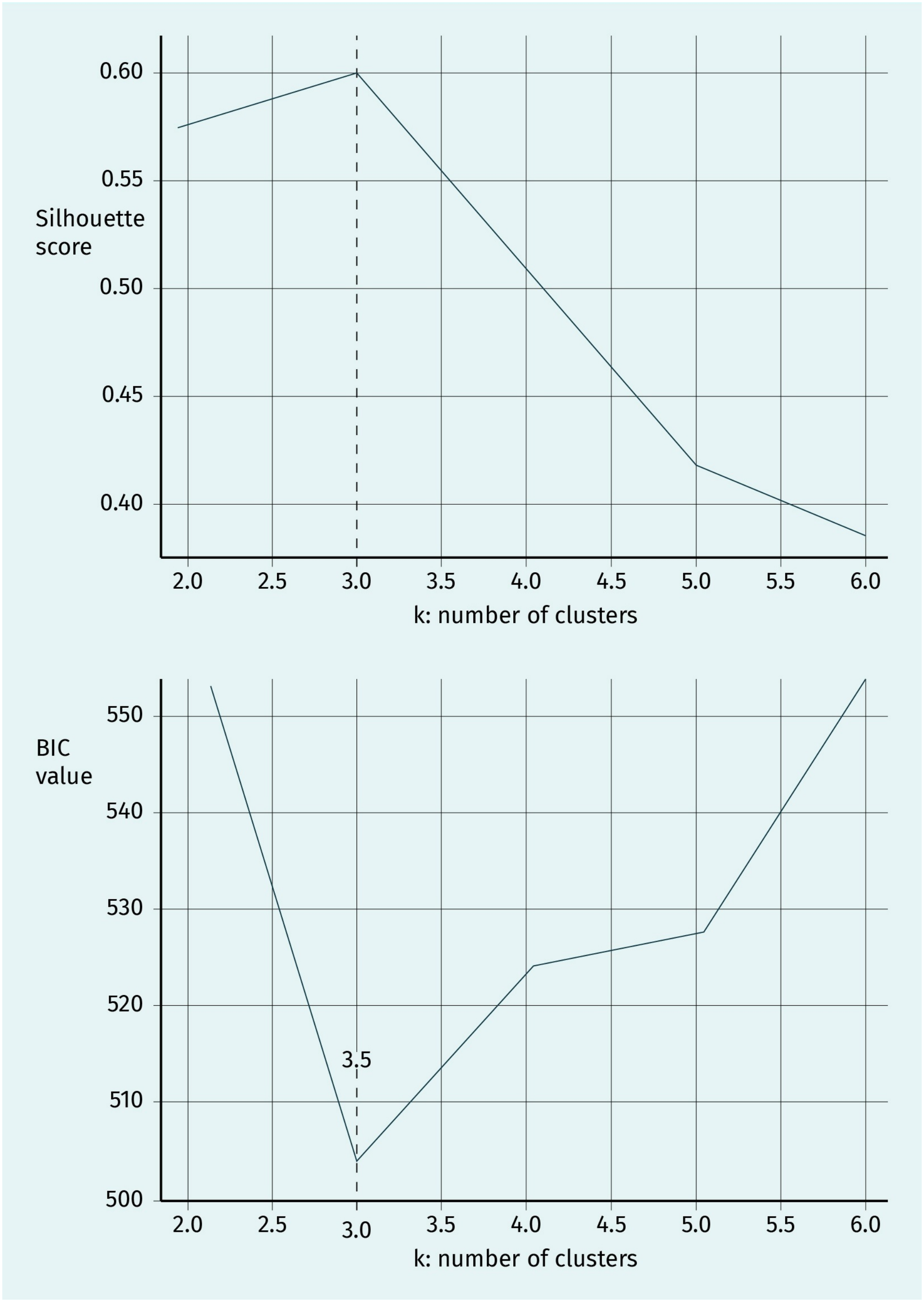
   Code
   ```
   # generate sample data
   X1 = 4 + np.random.rand(50,2)
   X2 = 5 + np.random.rand(50,2)
   X3 = 6 + np.random.rand(50,2)
   Z = np.concatenate((X1,X2,X3))
   ```

3. Compute both the Silhouette score and the BIC for $\mathrm{k} =$ from 2 to 6 and display the corresponding results. The figure below shows the Silhouette score and the BIC generated by this code. We can see that both criteria indicate that $\mathrm{k} = 3$ is the best number of clusters to be selected.

   Code
   ```
   # calculate the Silhouette score and BIC
   # for the number of clusters, k = 2 to 6
   S = []
   bic = []
   n_cluster_range = [2, 3, 4, 5, 6]
   for n_cluster in n_cluster_range:
      gmm = mixture.GaussianMixture(n_components=n_cluster)
      gmm.fit(Z)
      lab = gmm.predict(Z)
      S.append(silhouette_score(Z, lab))
      bic.append(gmm.bic(Z))
   # show the resuls visually
   # figure with two plots
   fig, (ax1, ax2) = plt.subplots(1, 2)

   # first plot: Silhouette score
   ax1.plot(n_cluster_range, S)
   ax1.set_title('Silhouette Score')
   ax1.set(xlabel='Number of clusters', \
      ylabel='Silhouette Score')
   ```

```
# second plot: BIC
ax2.plot(n_cluster_range, bic)
ax2.set_title('BIC')
ax2.set(xlabel='Number of clusters', \
   ylabel='BIC')

plt.show()
```

Figure 20: Silhouette Score and BIC as an Example of Three Gaussian Clusters



Source: Sayed-Mouchaweh (2021).

Generative method
A generative method provides a probability model enabling the estimation of the cluster's conditional probability density and its prior probability for data points in this cluster.

The GMM is a **generative method**. This is very useful for generating new samples or imputing missing data points. For the example of the three Gaussian clusters, we have already built the GMM model with $k = 3$. The following code allows us to generate and display 150 new data points using the built GMM model. The steps are demonstrated below:

1. Import the required Python libraries.

   Code
   ```
   import numpy as np
   import matplotlib.pyplot as plt
   from sklearn import mixture
   ```

2. Generate the three Gaussian clusters.

   Code
   ```
   # generate sample data
   X1= 4+np.random.rand(50,2)
   X2= 5+ np.random.rand(50,2)
   X3 = 6 + np.random.rand(50,2)
   Z= np.concatenate((X1,X2,X3))
   ```

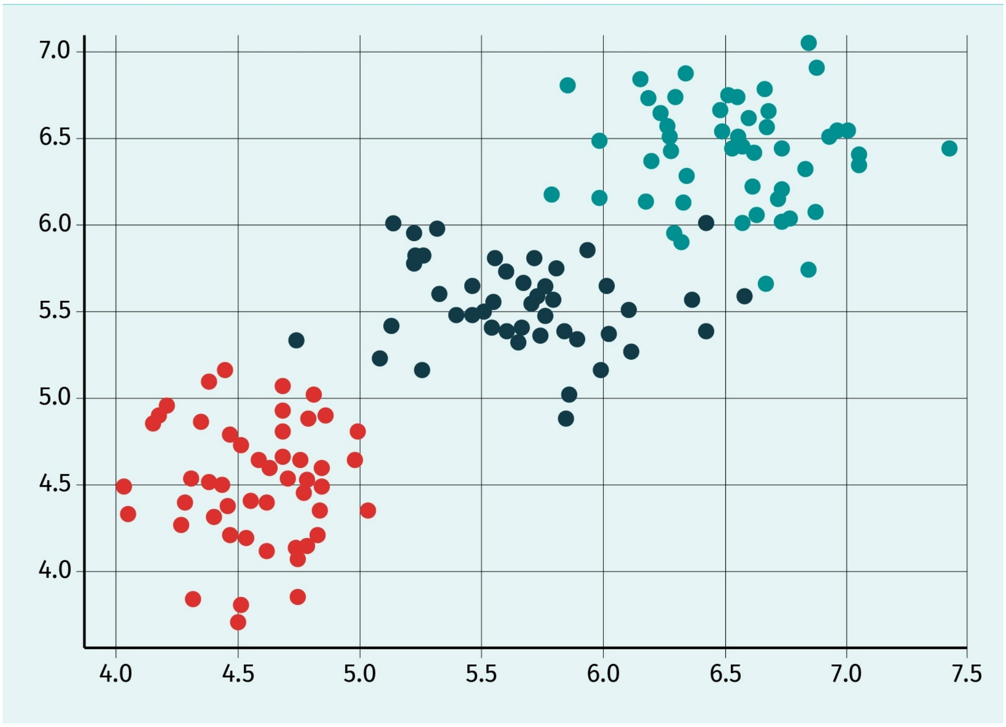
3. Build and it a GMM model for $k = 3$ clusters.

   Code
   ```
   gmm = mixture.GaussianMixture(n_components=3)
   gmm.fit(Z)
   ```

4. Generate 150 new data points using the already built GMM model and display them with their cluster assignment. The figure below shows the data points generated by the GMM.

   Code
   ```
   # generate new samples
   newdata = gmm.sample(150)

   # extract the feature values, i.e. coordinates
   vals = newdata[0]

   # extract the labels
   labs = newdata[1]

   # plot the generated samples
   plt.scatter(x=vals[:,0], y=vals[:,1], c=labs)
   plt.show()
   ```

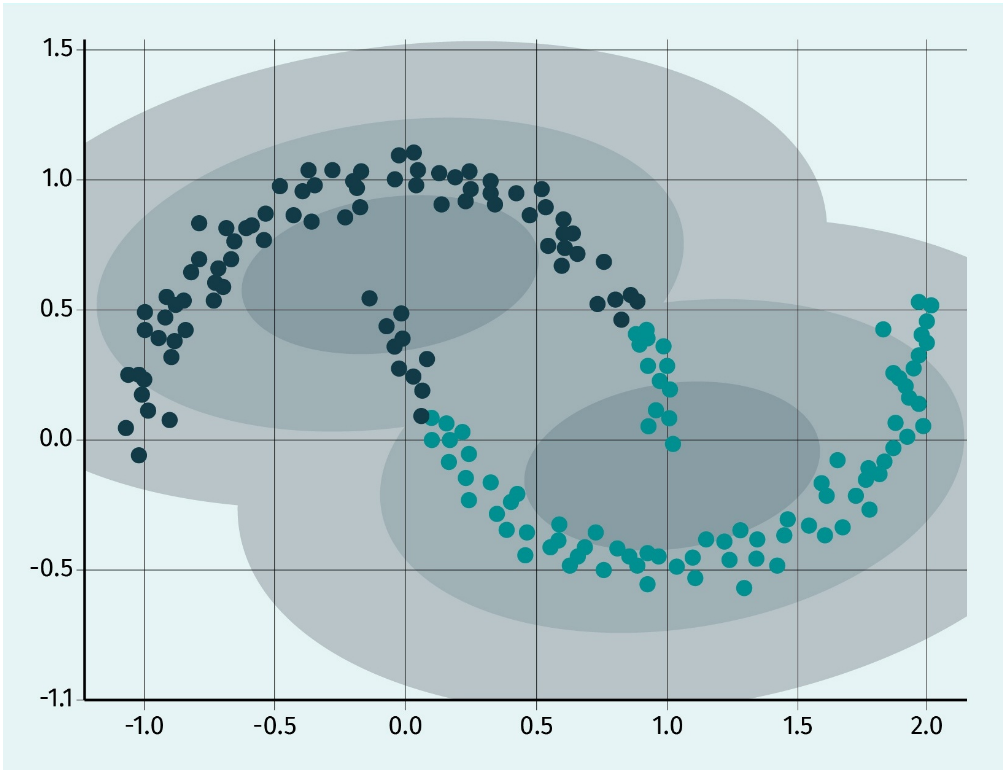
Figure 21: Clustered Obtained by Applying GMM on Data Points Including Three Gaussian Clusters



Source: Sayed-Mouchaweh (2021).

In addition, it is more efficient to represent overlapped clusters than k-means since it provides a probability membership value of each data point to different clusters. Moreover, since the GMM represents clusters, in addition to the mean value, by the variance-covariance matrix, it can discover clusters of non-spherical shape, e.g., elliptical shape, while k-means cannot do that since it uses Euclidian distance. However, it requires the number of clusters k to be known in advance. In addition, it depends, as with k-means, on the initialization of the parameters of each cluster converging toward local optimums. Finally, it cannot discover clusters of complex shape, such as the one in the figure below. We can see that the membership zones for the two moon clusters do not respect their non-convex shape.

Figure 22: Membership Zones Obtained by GMM for Clusters of Non-Convex Shape



Source: Sayed-Mouchaweh (2021).

2.3 Hierarchical Clustering

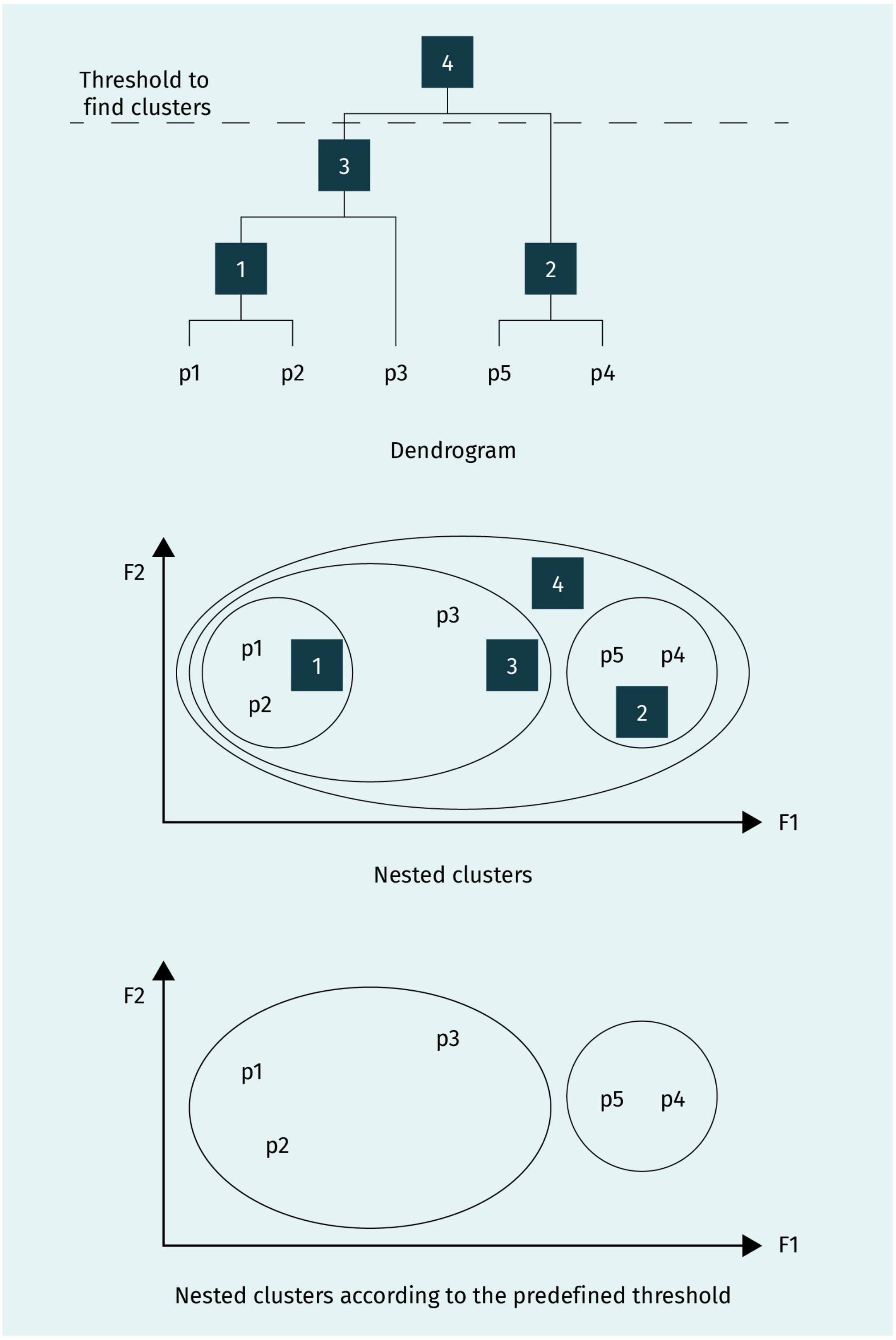
How It Works

Hierarchical clustering is based on two strategies: agglomerative (bottom-up) and divisive (top-down). In agglomerative strategy, each data point is considered to be a cluster at the lowest level of the hierarchy. The data points with smallest dissimilarity distance are merged into one cluster. Therefore, in the upper level of the hierarchy, fewer clusters are formed. At each level, the clusters that have the smallest inter-cluster distance are merged until they reach the highest level where all data points are gathered into one cluster. Divisive strategy follows the same reasoning but starts from the top by considering all the data points in one cluster. It then splits the cluster into two clusters that have the largest inter-cluster distance or dissimilarity. It repeats the splitting at each level until it reaches the lowest level where each cluster includes one data point.

Let us take the simple toy example in the figure below, illustrating a set of data points to be clustered, and apply the agglomerative strategy to perform the clustering. As we can see at each step of the agglomeration, the data points, or clusters, with the smallest inter-cluster dissimilarity are merged into one cluster until they reach one cluster grouping all the data points. Since the dissimilarity in agglomerative strategy increases in a monotone way, the dissimilarity is proportional to the merging or hierarchy level. The figure below shows an example of a **dendrogram**. It is used to decide the horizontal cutting, allowing us to determine the clusters to be provided by the hierarchical (agglomerative) clustering. This horizontal cutting is a dissimilarity threshold that allows us to decide the acceptable dissimilarity among the obtained clusters. The figure below shows the obtained clusters (two clusters) with respect to the predefined threshold (horizontal cutting of the dendrogram).

Dendrogram
A dendrogram is a graphical display of the merging levels with the corresponding obtained clusters. It can be used as a highly interpretable graphical description of the hierarchical clustering. Dendro means tree; gram means drawing.

Figure 23: Hierarchical (Agglomerative) Clustering


Threshold to find clusters
4
3
1
2
p1
p2
p3
p5
p4
Dendrogram
F2
4
p3
p1
1
3
p5
p4
p2
2
F1
Nested clusters
F2
p3
p1
p5
p4
p2
F1
Nested clusters according to the predefined threshold

Source: Sayed-Mouchaweh (2021).

The advantage of hierarchical clustering is that it does not require the specification in advance of the number of clusters to be searched, as is the case for k-means. Instead, it requires the user to define the dissimilarity measure to be used in order to merge or split the disjoint clusters. It is up to the user to decide which level of the hierarchical clustering is the one which resembles the natural structure the most.

Hierarchical clustering does not scale linearly with large size datasets because its complexity is at least $\mathrm{O(n^2)}$, where n is the number of data points. However, there are strategies for tackling this issue, such as the pre-calculation of distance matrices in a parallel and distributed way. In addition, it cannot be applied to clusters of complex shape such as non-convex banana-shaped data points. It is worth mentioning that there are other clustering algorithms, such as Density Based Spatial Clustering of Applications with Noise (DBSCAN) (Ester et al., 1996), which are able to discover clusters with arbitrary or irregular shapes.

Merging in Agglomerative Clustering

Merging two clusters A and B requires a dissimilarity measure. The dissimilarity $\mathrm{diss(A, B)}$ between A and B is based on the dissimilarity between the pairwise data points belonging to these two clusters. Let $\mathrm{diss(x_A,\ x_B)}$ be the dissimilarity between $\mathrm{x_A}$ (the data point in A) and $\mathrm{x_B}$(the data point in B). $\mathrm{diss(A,\ B)}$ can be computed by one of the following three strategies.

The single linkage (SL), or nearest neighbor technique, considers the dissimilarity between two clusters as the smallest dissimilarity between their pairwise data points (i.e., comparing the two data points from different clusters which are closest to each other):

$$diss(A, B) = \min_{x_A \in A, x_B \in B} diss(x_A, x_B)$$

The limit of this strategy is related to its strong tendency to merge clusters at relatively small thresholds. The consequence, called chaining, of this limit can be seen particularly for clusters with a large diameter (high variance or dispersion of their data points) since only one point at the border of each cluster is sufficient to merge both clusters. This entails violating the compactness property of clusters stating that all the data points in the same cluster must be similar.

Complete linkage (CL), or furthest neighbor technique, is the extreme opposite of SL strategy. It considers the inter-cluster dissimilarity between A and B to be the furthest (most dissimilar) of their pairwise data points:

$$iss(A, B) = \max_{x_A \in A, x_B \in B} diss(x_A, x_B)$$

Therefore, A and B are merged only if all their corresponding data points are relatively similar. This will entail compact clusters with a small diameter. For instance, several compact clusters will be formed while they belong to one natural cluster. This will entail violating the cluster compactness property since some data points of a small compact cluster can be much closer to some data points in a neighboring cluster than its proper members in its own cluster.

Finally, group average (GA) clustering considers the dissimilarity between A and B as the average dissimilarity between the set of pairwise members of these two clusters:

$$diss(A, B) = \frac{1}{|A| \,.\, |B|} \sum_{x_A \in A} \sum_{x_B \in B} diss(x_A, x_B)$$

GA can be considered a compromise between SL and CL allowing us to produce relatively compact clusters that are separated relatively enough.

Ward
This term refers to an algorithm used by hierarchical clustering to determine the pair of clusters to merge at each step. It computes the "error sum of squares" (ESS) after merging two clusters into a single cluster. It chooses the clusters to merge by minimizing the increase in ESS at each step.

Let us use the SL strategy that is implemented in the Python library `scipy.cluster.hierarchy` as the function **Ward**. In the same library, the dendrogram function allows us to trace the hierarchical clustering as a dendrogram. The code is structured as follows:

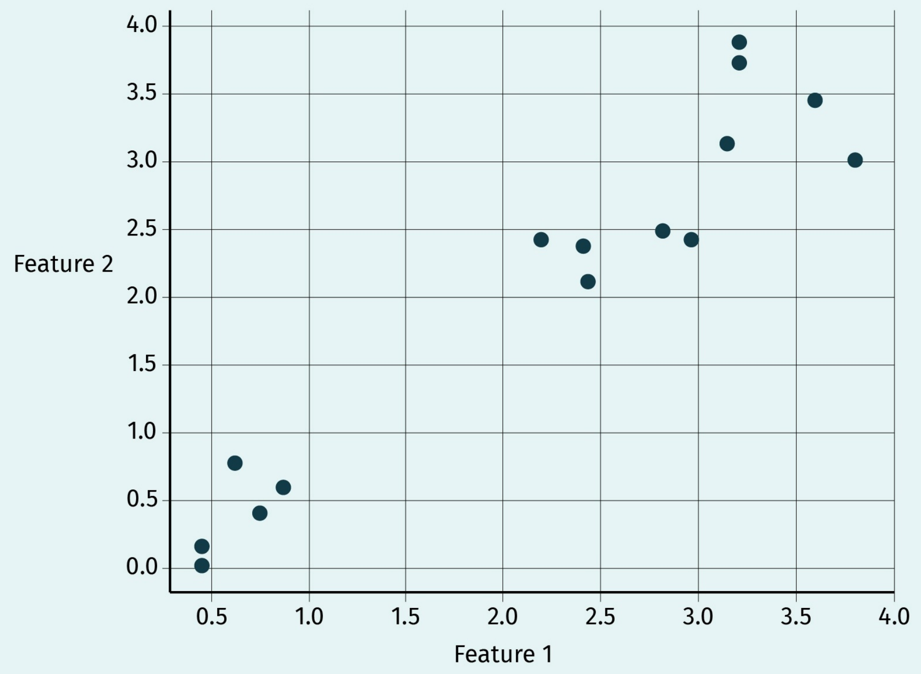
1. Import the required Python libraries, in particular the dendrogram function and the ward clustering function from SciPy.

   Code
   ```
   import matplotlib.pyplot as plt
   import numpy as np
   from scipy.cluster.hierarchy import dendrogram, ward
   ```

2. Generate three clusters, including normally distributed data points, as illustrated in the figure below; two of the three clusters are close to each other.

   Code
   ```
   X1= np.random.rand(5,2)
   X2= 2 + np.random.rand(5,2)
   X3 = 3 + np.random.rand(5,2)
   Z= np.concatenate((X1,X2,X3))
   ```

Figure 24: Simple Toy Example



Three Gaussian groups are generated, where two of them are close.

Source: Sayed-Mouchaweh (2021).

3. Apply the Ward function to the generated data array. It returns an array that specifies the distances between data points.

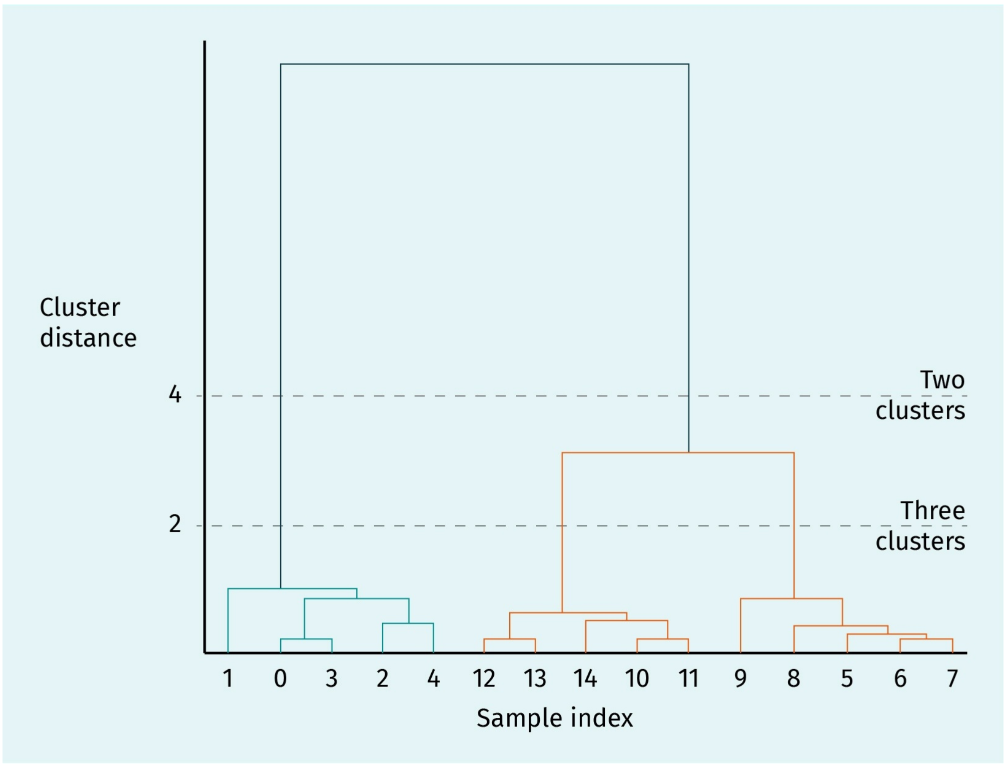
   Code
   ```
   linkage_array = ward(Z)
   ```

4. Generate and plot the dendrogram depicted in the figure below. The dendrogram shows the linkage array or the distances between clusters. We can set the threshold in order to obtain either two or three clusters, as illustrated in the figure below. In the case of setting the threshold in order to obtain two clusters, we notice that the two close clusters (orange clusters in the figure below) are merged.

   Code
   ```
   # create a dendrogram
   dendrogram(linkage_array)
   ax = plt.gca()
   bounds = ax.get_xbound()

   # add the boundary for two/three clusters
   ax.plot(bounds, [4, 4], '--', c='k')
   ax.plot(bounds, [2, 2], '--', c='k')
   ```

```
# add an annotation to the marked boundary
ax.text(bounds[1], 4, ' two clusters', va='center',
fontdict={'size': 15})
ax.text(bounds[1], 2, ' three clusters', va='center',
fontdict={'size': 15})

# label the axes
plt.xlabel("Sample index")
plt.ylabel("Cluster distance")

# show the plot
plt.show()
```

Figure 25: Dendrogram Example



Source: Sayed-Mouchaweh (2021).

We can also specify the number of clusters directly, without the need for a dendrogram, using the function AgglomerativeClustering in Python as follows. The index (0, 1, 2) indicating the cluster of each data point is recorded in the array Assignment. This function provides the cluster index, the variable “Assignment” in the code below, for the data points, variable “Z” in the code below. For our example, “Assignment” will be a matrix of 15 rows and one column. Each row will have the cluster index “0,” “1,” or “2” as follows: Assignment = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 0, 2].

Code

```
# import libraries
from sklearn.cluster import AgglomerativeClustering
import numpy as np

# execute agglomerative clusering with 3 clusters
agg = AgglomerativeClustering(n_clusters=3)

# predict clusters
labs = agg.fit_predict(Z)

print(labs)
# console output: [0 0 0 0 0 2 2 2 2 2 1 1 1 1 1]
```

SUMMARY

This unit presented and discussed the functioning principal and the advantages and drawbacks of the major clustering approaches: k-means, Gaussian mixture model (GMM) clustering, and hierarchical clustering. This unit started by presenting the general principal of clustering approaches. It showed the different steps in implementing the clustering approach, and drawbacks and advantages. The unit focused on how to test the quality of the obtained clusters by presenting meaningful criteria, such as the Elbow method, the Silhouette score, and the Bayesian Information Criterion (BIC).

The unit highlighted the comparison between these clustering approaches with respect to the clustering challenges, such as the number of clusters in advance requirement, the overfitting problem, the complex and non-convex shape of clusters, the size of dataset, and the overlapping clusters. The goal is to guide the choice of the suitable clustering approach with respect to the application (data points) constraints and conditions.

UNIT 3

DIMENSIONALITY REDUCTION

STUDY GOALS

On completion of this unit, you will be able to …

- explain how dimensionality reduction approaches work.
- apply dimensionality reduction approaches.
- choose an appropriate dimensionality reduction approach with respect to the challenges and constraints of the dataset.

3. DIMENSIONALITY REDUCTION

Introduction

Dimensionality reduction aims at projecting the data points described in high dimensional space into a new feature space with much fewer features. Its principal motivation is to avoid the model overfitting. Indeed, a model trained using a dataset in a high dimensional features space becomes increasingly dependent on these data samples. Therefore, its performances, i.e., the quality of the extracted knowledge or clusters, become poor for unseen data samples. In addition, dimensionality reduction helps to compress data, leading to reduced storage space and computation time. Moreover, dimensionality reduction is frequently used to explain a dataset. For instance, if we have to deal with a dataset with many features, exploratory data analysis becomes hard as it is difficult to visualize in a feature space with more than two or three dimensions. Therefore, dimensionality reduction may add to the interpretability of these datasets.

This projection is obtained through linear or nonlinear transformation of the original variables or features. The new, transformed features better explain the variance of the data samples than the original features. This is because the dimensionality reduction allows the elimination of noisy, redundant, or irrelevant features by consolidating features that are highly correlated (hence, better explaining the variance).

The transformation of original features is performed by assigning continuous weights. These weights are determined by searching to optimize an objective function using the available dataset. The objective function aims at optimizing the separation between the clusters gathering the data points.

The application of dimensionality reduction to real-world problems faces several challenges. For instance, enough data samples must be available in order to observe their tendency or variance axes. Moreover, the interpretation of, for example, obtained clusters, requires an inverse transformation from the reduced to the original feature space. Moreover, the transformation must conserve the nonlinearity of correlations and other important properties between data samples in the reduced feature space. Finally, the minimal number of reduced features required to conserve these properties needs to be determined. These challenges affect the quality of the reduced feature space as well as the obtained clusters by a clustering algorithm.

There are several ways to perform dimensionality reduction techniques. They can be divided into linear, such as Principal Component Analysis (PCA), and nonlinear, such as Multi-Dimensionality Scaling (MDS) and Locally Linear Embedding (LLE), feature extractions. The choice of one of these techniques depends on the challenges to address with respect to the available dataset. This unit examines the following questions:

- What are the drawbacks and advantages of dimensionality reduction approaches?
- How do we implement the dimensionality reduction technique by ensuring the best separation between the data samples' natural clusters in the reduced feature space?
- What are the criteria to use when choosing a dimensionality reduction technique with respect to the constraints and challenges of the data samples?

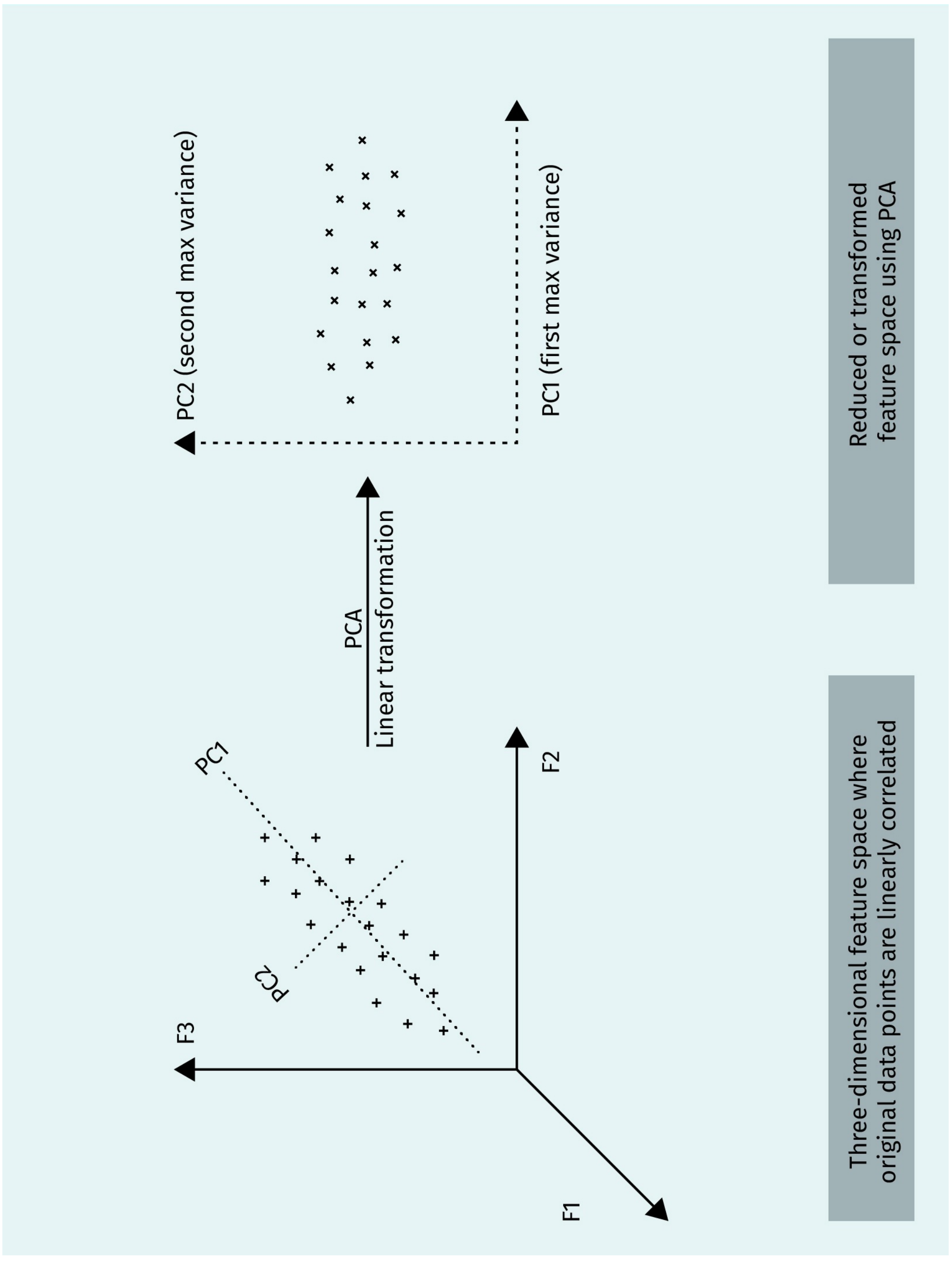
3.1 Principal Component Analysis (PCA)

How It Works

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction approach that allows the transformation of data samples into reduced feature space while conserving the maximum amount of information (variance). It rotates the feature space around orthogonal axes, thereby transforming the data to a feature space. This allows for more separability between variance explanations of features. Therefore, finding orthogonal axes in the feature space and rotating the data around them is central to PCA.

To explain the interest of Principal Component Analysis (PCA), let us take the example of the figure below. Most of the variance of the data samples in the original feature space is, along a single feature, different from the original features. PCA aims at extracting new features that better emphasize the variance in the data samples than the original features. It is worth mentioning that the example of the figure below is just a simple illustration example. PCA is useful when the number of original features is much greater than three.

Figure 26: Unsupervised Transformation Using Principal Component Analysis (PCA)



Three-dimensional feature space where original data points are linearly correlated
F1
F2
F3
PC1
PC2
PCA
Linear transformation
PC1 (first max variance)
PC2 (second max variance)
Reduced or transformed feature space using PCA

Source: Sayed-Mouchaweh (2021).

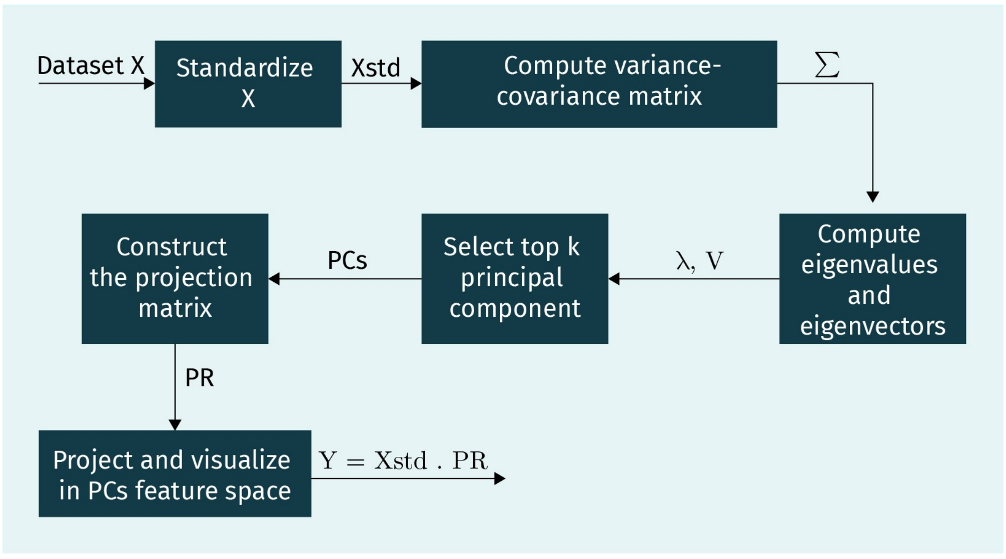
Let us explain the functioning principal of PCA using the Iris dataset as an example. This set includes 150 data samples representing the three species of Iris (*Irissetosa*, *Irisvirginica*, and *Irisversicolor*) described by four features (length and width of sepals and petals). This dataset can be downloaded from datasets in sklearn in Python as follows. We also need to import PCA from sklearn.decomposition:

Code

```
# load libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
# load sample data
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

The steps required in order to implement PCA are illustrated in the figure below.

Figure 27: Implementing PCA


Dataset X
Standardize X
Xstd
Compute variance-covariance matrix
$\sum$
Compute eigenvalues and eigenvectors
λ, V
Select top k principal component
PCs
Construct the projection matrix
PR
Project and visualize in PCs feature space
Y = Xstd . PR

Source: Sayed-Mouchaweh (2021).

The steps are demonstrated below:

1. **Standardize the data points** X, allowing them to have a mean equal to zero and standard deviation equal to 1. The standardization is crucial for PCA since it is sensitive to variance. Sometimes, features may have different scales and units. Therefore, standardizing data samples allows them to have the same range with respect to each feature. The standardization can be obtained by

$$\forall x \in X, x_{std} = \frac{x - \mu}{\sigma}$$

where μ is the mean value and σ is the standard deviation. This can be done using Python by importing StandardScaler from sklearn.preprocessing as follows:

Code

```
X_std = StandardScaler().fit_transform(X)
```

2. **Compute the matrix of variance-covariance** $\sum$, which represents the correlation between two variables (features). It is used to detect which two variables are heavily dependent on or correlated to each other in order to capture the redundant features in the dataset. It can be calculated using the following code:

Code

```
cov_X_std = np.cov(X_std.T)

print(cov_X_std)
# console output:
# [[ 1.00671141 -0.11835884 0.87760447 0.82343066]
# [-0.11835884 1.00671141 -0.43131554 -0.36858315]
# [ 0.87760447 -0.43131554 1.00671141 0.96932762]
# [ 0.82343066 -0.36858315 0.96932762 1.00671141]]
```

3. **Compute eigenvectors and eigenvalues on the variance-covariance matrix.** Indeed, the variance-covariance matrix will be transformed or decomposed by finding its **eigenvectors**V and **eigenvalues**λ by

Eigenvectors
The eigenvectors determine the axes that maximize the variance of data samples.

Eigenvalues
The eigenvalues are scalars that allow us to stretch the eigenvectors.

$$\Sigma . V = \lambda . V$$

This decomposition of the matrix to be represented in terms of its eigenvectors and eigenvalues is called matrix eigen decomposition. The sum of all the eigenvalues represents the overall variance in the entire dataset. Each eigenvector has its associated eigenvalue. The eigenvectors represent the axes, directions, or principal components that form the transformed or reduced feature space. The highest eigenvalue explains where the maximum variance lies in the dataset. The following code computes them:

$$\lambda = [\lambda_1, \lambda_2, \lambda_3, \lambda_4] = [2 .93 , 0 .93 , 0 .14 , 0 .02]$$

Code

```
eig_vals, eig_vecs = np.linalg.eig(Cov_X_std)
print(eig_vals)
# console output: [2.93808505 0.9201649 0.14774182
0.02085386]
print(eig_vecs)
# console output:
# [[ 0.52106591 -0.37741762 -0.71956635 0.26128628]
# [-0.26934744 -0.92329566 0.24438178 -0.12350962]
# [ 0.5804131 -0.02449161 0.14212637 -0.80144925]
# [ 0.56485654 -0.06694199 0.63427274 0.52359713]]
```

4. **Select the principal components to form the reduced feature space.** Since eigenvalues represent the proportion of explained variances in the data samples according to each eigenvector, they are sorted in decreasing order. Therefore, the first eigenvalue is the most significant and thus forms the first principal component candidate.
   Remember that eigenvectors form the axes in the transformed feature space. Most of the variance in the data happens along these transformed axes. To determine the number of principal components that we will keep to form the reduced or transformed feature space, we will use the explained variance criterion that represents the variance (information) conserved by the selected principal components. We will keep the k top eigenvectors, having the k top eigenvalues, that their cumulative sum conserves as the maximum of variance greater than a threshold, normally defined to be 90 percent of the overall data variance. The other eigenvectors will be dropped since they explain the least about the distribution of the data. The following code allows us to compute the explained variance using Python for the Iris dataset. Running this code provides the following results:

   $$exp_{var1} = 72.9\%, exp_{var2} = 22.8\%, exp_{var3} = 3.7\%, exp_{var4} \\ = 0.5\%$$

   The cumulative sum of the first two principal components is $cum\_exp\_var = 95.8\%$, which is enough to ensure that more than 90 percent of the total variance is represented in the reduced dimensional space. Therefore, for the Iris dataset, we will reduce the feature space from four features, in the original space, to a feature space formed by the two first principal components.

   Code
   ```
   # calculating the total sum of Eigenvalues
   tot = sum(eig_vals)
   print(tot)
   # console output: 4.026845637583896

   # sorting Eigenvalues in decreasing order
   exp_var = sorted(eig_vals, reverse=True)

   # present values as percentages
   exp_var = exp_var/tot * 100

   # print the explained variance
   print("Explained variance per PC:", exp_var)
   # console output:
   # Explained variance per PC: [72.96244541 22.85076179
   # 3.66892189 0.51787091]

   # Computing and print the explained cumulative variance
   cum_exp_var = np.cumsum(exp_var)
   print("Cumulative Explained Variance:",cum_exp_var)
   # console output:
   ```

```
# Cumulative Explained Variance: [ 72.96244541 95.8132072
# 99.48212909 100. ]

# display the proportion of variance
# which is explained by the first two PC
print("Explained variance by PC1 and PC2:", \
   sum(exp_var[0:2]))
# console output:
# Explained variance by PC1 and PC2: 95.81320720000163
```

5. **Construct the projection matrix from the selected k eigenvectors.**The projection matrix $\mathrm{PR(dxk)}$ will allow the projection of the Iris data samples from the original feature space ($\mathrm{d}= 4$) into the reduced or transformed feature space formed by the top $\mathrm{k}= 2$ principal components (eigenvectors) with the highest eigenvalues. To do this, we will associate the top eigenvalues with their associated eigenvectors by building the array Eig_pairs. Then, we will choose the top $\mathrm{k}= 2$ pairs of this array in order to build the projection matrix PR. The following code allows us to perform this construction. The np.hstack is used to concatenate the two principal components (two selected eigenvectors), and the obtained matrix is reshaped to be in $\mathrm{d}= 4$ lines and $\mathrm{k}= 2$ columns.

   Code
```
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i])
             for i in range(len(eig_vals))]
PR = np.hstack((eig_pairs[0][1].reshape(4,1),
                eig_pairs[1][1].reshape(4,1)))
```

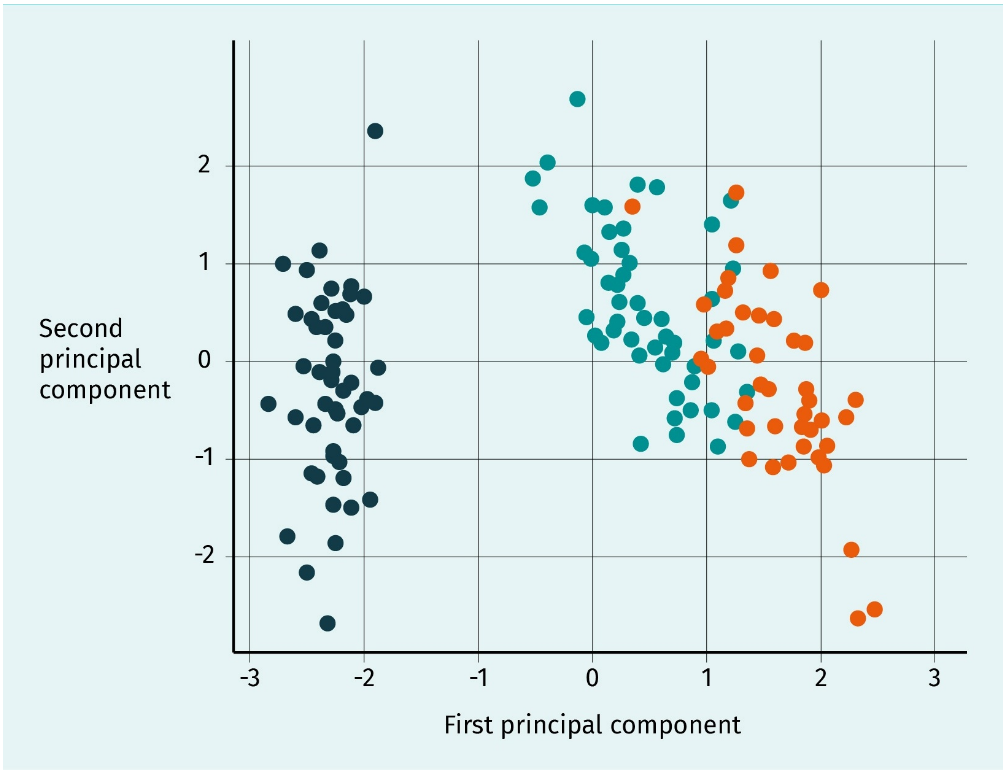
6. **Project and visualize the data samples into the new reduced feature space**.The original standardized data samples $\mathrm{X\_std(150x4)}$ are projected onto the reduced feature space in order to obtain the transformed data samples $\mathrm{Y(150x2)}$ using the projection matrix $\mathrm{PR(4x2)}$. This projection is achieved by applying

$$Y = X\_std\,.\,PR$$

   The figure below shows the Iris data samples projected onto the new, reduced feature space. The following code allows us to achieve this projection and visualization.

   Code
```
# project the original data to the feature
# which is reduced in dimensions
Y = X_std.dot(PR)
YL = np.append(Y, y.reshape(150,1), axis=1)

# plot the projected data
plt.scatter(x=YL[:,0], y=YL[:,1], c=YL[:,2])
plt.show()
```

Figure 28: Projection of the Iris Data in a Reduced Feature Space Formed by the Top Two Principal Components



Source: Sayed-Mouchaweh (2021).

The figure above shows the use of PCA-facilitated description of the Iris dataset in a two-dimensional feature space, while capturing almost all of the characteristics of the original higher four-dimensional feature space. Therefore, it is now easier to work with these data and to explore them visually. Python has the advantage of including a developed package (libraries and functions) in order to use PCA with far fewer lines of code. All previous steps to conduct a PCA can be reduced to the following, much simpler, code:

Code

```
# doing it the easy way
pca = PCA().fit(X_std)

# extract the explaind variance ratios
var_exp = pca.explained_variance_ratio_
print(var_exp)
# console output:
# [0.72962445 0.22850762 0.03668922 0.00517871]

# calculate the explained cumulative variance
cum_var_exp = np.cumsum(var_exp)
print(cum_var_exp)
```

```
# console output:
# [0.72962445 0.95813207 0.99482129 1.]

# extract the Eigenvectors
eig_vecs = pca.components_

# use PCA to project the data to a two-dimensional
# feature space
Y = PCA(n_components=2).fit(X_std).transform(X_std)pca = PCA().fit(X_std)
```

Advantages and Drawbacks

PCA is a powerful unsupervised dimensionality reduction approach that allows the transformation of the data samples into a reduced feature space while conserving the maximum amount of information (variance). It can be used for different applications, such as compressing the dataset into a space formed by fewer transformed features than the original feature space. In addition, PCA is useful to speed up the learning algorithm or to visualize the data points. It can also be used to compute and graphically show the correlations between the transformed features (principal components) and the original features. This can be achieved using the correlation circle. The correlation circle graphically demonstrates the correlations between the original features (sepal length, sepal width, petal length, and petal width for the Iris dataset) and the principal components (e.g., the top $\mathrm{k} = 2$ eigenvectors for the Iris dataset). These correlations are plotted as vectors (one vector for each original feature) on a unit circle. Therefore, for the Iris dataset, four vectors (sepal length, sepal width, petal length, and petal width) are plotted (see the figure below). The axes of the circle are the selected principal components. The projection of each vector on these axes provides the explained variance percentage of this feature with respect to the corresponding principal component.

As shown in the figure below, the explained variance of the feature “sepal_width” is 0.46 according to the first principal component and —0.88, according to the second principal component. In addition, the correlation circle allows the observation of the correlated and uncorrelated features. Per the figure below, the “petal_length” and “petal_width” features are clearly correlated since the angle between them is close to zero. Both are correlated with the third feature “sepal_length.” Meanwhile, these three features are uncorrelated with the fourth feature “sepal_width” since they are almost orthogonal in relation. Therefore, “sepal width” should be kept with one of the three other features. “Petal_length” might be the best choice to keep with “sepal_width” since it is the most uncorrelated feature among the three correlated features with “sepal_width.”

The correlation circle can be computed using the function `plot_pca_correlation_graph()`in Python. The library mlxtend must be installed when using, for example, the following instruction: `pip install mlxtend`. The code below displays the correlation circle for the top $\mathrm{k}= 2$ principal components and the four features of the Iris dataset. Remember that the explained variances for the first principal component, PC_1, and the second principal component, PC_2, are 72.9 percent and 22.8 percent, respectively. This code also displays the values of the correlation matrix “correlation_matrix" for the four original features, as in the table below. This correlation matrix is also called the loading matrix since it represents the correlations between the original variables and the principal components.

Table 7: Correlation or Loading Matrix for the Iris Dataset

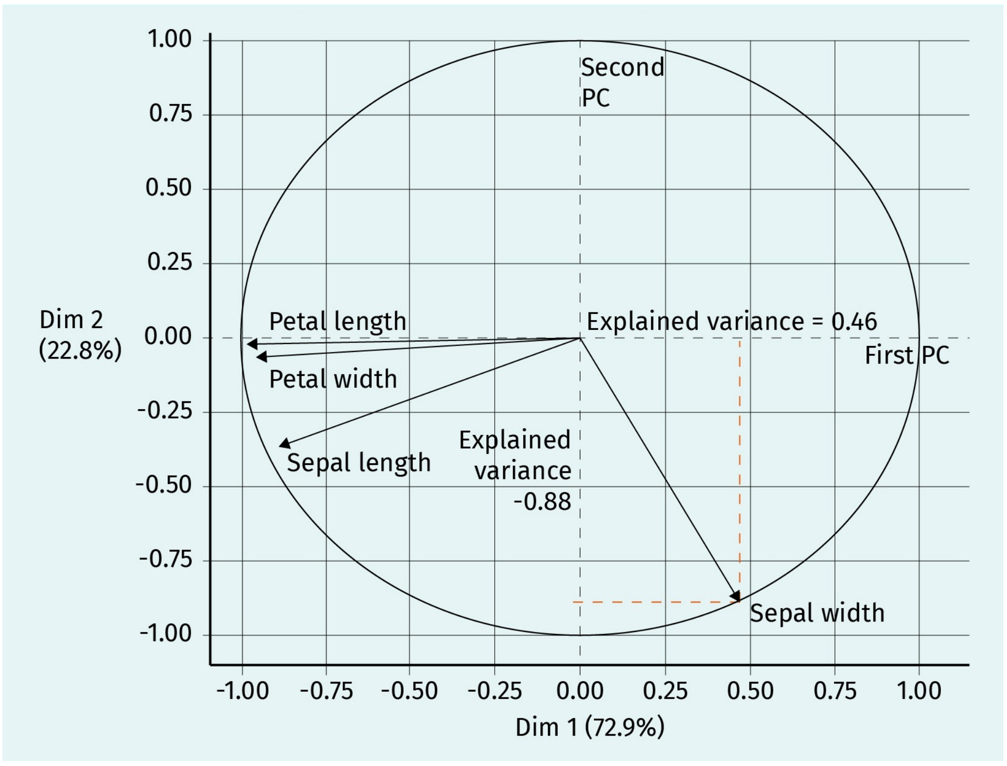
	PC1	PC2
sepal length	—0.89	—0.36
sepal width	0.46	—0.88
petal length	—0.99	—0.02
petal width	—0.96	—0.06

Source: Sayed-Mouchaweh (2021).

Code

```
                    from mlxtend.plotting import plot_pca_correlation_graph

# specify feature names
feature_names = ['sepal length','sepal width',
        'petal length','petal width']


                    #
                    calculate the correlation matrix and
# create a correlation graph
fig, cor_mat = plot_pca_correlation_graph(X_std, \
  feature_names, dimensions=(1, 2), \
  figure_axis_size=10)


                    #
                    show the numbers of the correlation
# matrix for the 4 features
print(cor_mat)
                    #
                    console output:
#         Dim 1  Dim 2
# sepal length -0.890169 -0.360830
# sepal width 0.460143 -0.882716
# petal length -0.991555 -0.023415
# petal width -0.964979 -0.064000
```

Figure 29: Correlation Circle for the Iris Dataset

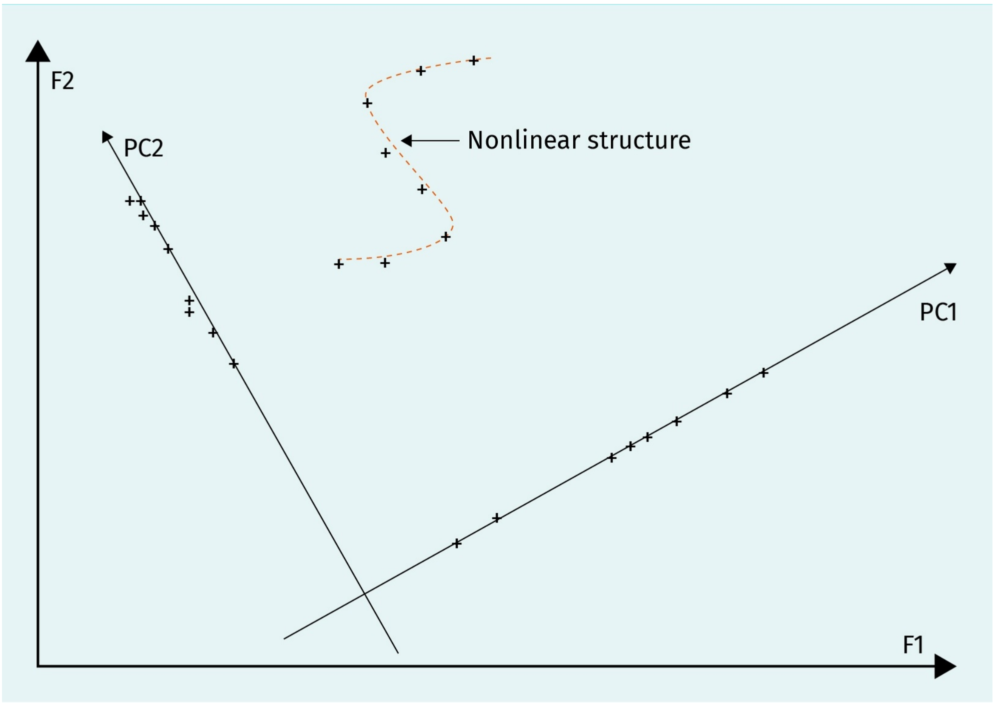


Source: Sayed-Mouchaweh (2021).

However, PCA suffers from several drawbacks. They are as follows:

- PCA performs poorly for nonlinearly correlated data samples as it will not clearly separate traits of the dataset into principal components. For instance, in the figure below, PCA cannot properly capture the nonlinear structure of the data samples by a rotation of the initial axes of the original feature space. Most real-world datasets are characterized by variables or features that are nonlinearly correlated. This limits the use of PCA for real-world applications.

Figure 30: Nonlinearly Correlated Data Points and Their Projection into the Transformed Feature Space Using PCA

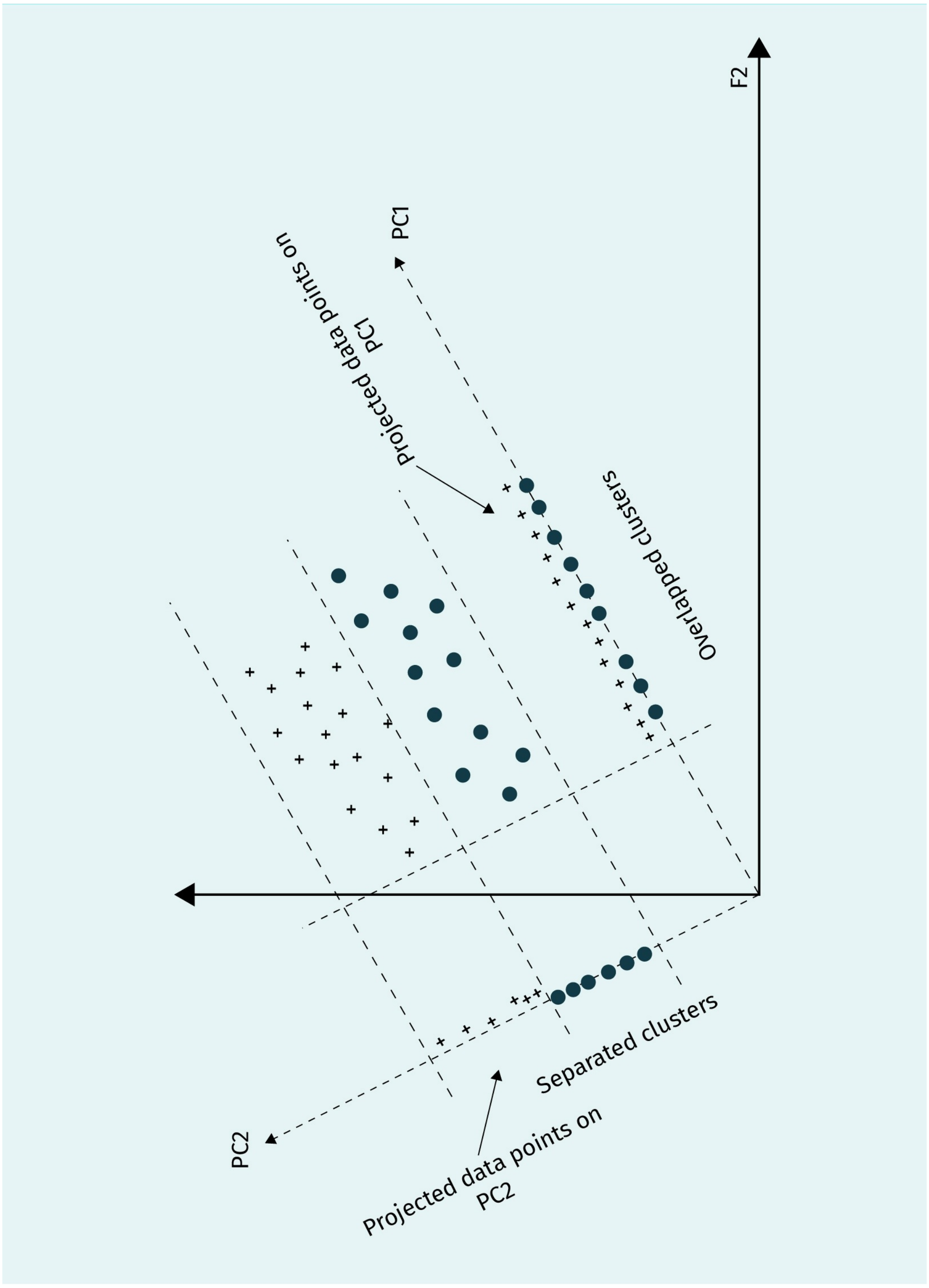


Source: Sayed-Mouchaweh (2021).

- The selection of principal components depends on the application at hand. As in the figure below, the axis PC_1 that holds the maximum data variance is not suitable to keep in order to distinguish between the two clusters in a reduced feature space. Indeed, the axis or the principal component, PC_2 in the figure below, that holds less data variance is the right one to use in order to properly discriminate the two clusters in a reduced feature space.
- PCA can only be applied to numerical features since it is based on the use of their variance. PCA may behave poorly if the dataset contains categorical features, such as nationality, country, gender, or customer satisfaction (high, medium, and low), since their variance may not have a meaning in the case of numerical features. Indeed, although categorical features can be converted into numerical features, the converted features (quantified) need to properly reflect the distance between the different levels of ordinal variables or the different categories of nominal variables.
- The feature space which has been transformed by PCA is not easily interpretable. For example, in the Iris dataset, the “petal length” or “petal width” have a direct meaning and can be easily interpreted. For the top two principal components, the interpretation or meaning is not that apparent. In order to make sense of these principal components, we have to investigate further, for instance, the **vector loadings**, the eigenvalues, and their relative proportions.

Vector loadings
The vector loadings are the eigenvectors multiplied by the square root of eigenvalues. They represent the correlations between the original variables and the eigenvectors.

Figure 31: Selection of PC2 to Separate Clusters Although PC2 Holds Less of Variance than PC1



Source: Sayed-Mouchaweh (2021).

3.2 Multi-Dimensional Scaling

How it Works

Multi-Dimensional Scaling (MDS) allows the projection of data samples in a high-dimensional space in a reduced dimensional space, normally two or three dimensions, by preserving the pairwise sample similarity or dissimilarity. The latter can be represented by a distance metric, often the Euclidian distance, in order to weigh the distances between the different data samples. For instance, if we have three data samples A, B, and C, and if A is close to B but too far from C in the original high dimensional space, then in the reduced two- or three-dimensional space, A and B will also be close together, and will be far from C.

MDS resembles PCA in the sense that both perform a matrix eigen decomposition in order to project data samples into a reduced (transformed) feature space. Remember that the decomposition of a matrix to be represented in terms of its eigenvectors and eigenvalues is called matrix eigen decomposition. However, MDS differs from PCA in the kind of used matrix for eigen decomposition and the objective function that they aim to optimize. Indeed, MDS performs eigen decomposition on a distance or dissimilarity matrix, also called the proximity matrix (Buja et al., 2008). The **dissimilarity matrix** is computed based on the use of the dataset. The two categories of MDS are as follows:

Dissimilarity matrix
This matrix is computed by calculating the distance between each two data points of the dataset. Several distance metrics can be used, such as Euclidian, Manhattan, or weighted Euclidian distance.

s.t.
This abbreviation stands for "such that."

1. Metric MDS, which is used when the dataset is described by numerical or quantitative features. In this case, MDS seeks to preserve the dissimilarity metrics of data samples in the original feature space. Metric MDS seeks to find a monotone distance function or metric f defined in $\mathrm{R^p}$, where$\mathrm{p} = 2 \text{ or } 3$, **s.t.** the distance f between two data points $\mathrm{x_i}$ and $\mathrm{x_j}$ approaches $\mathrm{d_{ij}} = ||\mathrm{x_i} - \mathrm{x_j}||$.
2. Non-metric MDS, which is used when the dataset is described by categorical (ordinal, nominal) features. In this case, MDS seeks to keep the order of dissimilarity metrics of the original data samples. For instance, if $\mathrm{d_{ij}}$ is the dissimilarity between data samples $\mathrm{x_i}$ and $\mathrm{x_j}$ in the original feature space, and $\mathrm{x_i} > \mathrm{x_j}$, then non-metric MDS seeks creating a mapping or a distance metric s.t. $\mathrm{d_i} > \mathrm{d_j}$.

We will demonstrate how to use the Python packages to run the metric MDS using the Iris dataset. The optimized configuration or distance function is obtained using the optimization algorithm Scaling by Majorization of Complicated Function (**SMACOF**). To this end, the steps are as follows:

SMACOF
This term refers to an optimization strategy that aims at minimizing a cost or loss function. The latter measures the squared differences between ideal (original) distances and projected distances in two- or three-dimensional space.

1. We import MDS from the Python library sklearn.manifold. The default parameters for MDS use metric MDS and the Euclidian distance. We also import the MinMaxScaler function from sklearn.preprocessing in order to normalize the data samples between 0 and 1.

   Code
   ```
   import numpy as np
   from sklearn import datasets
   import matplotlib.pyplot as plt
   ```

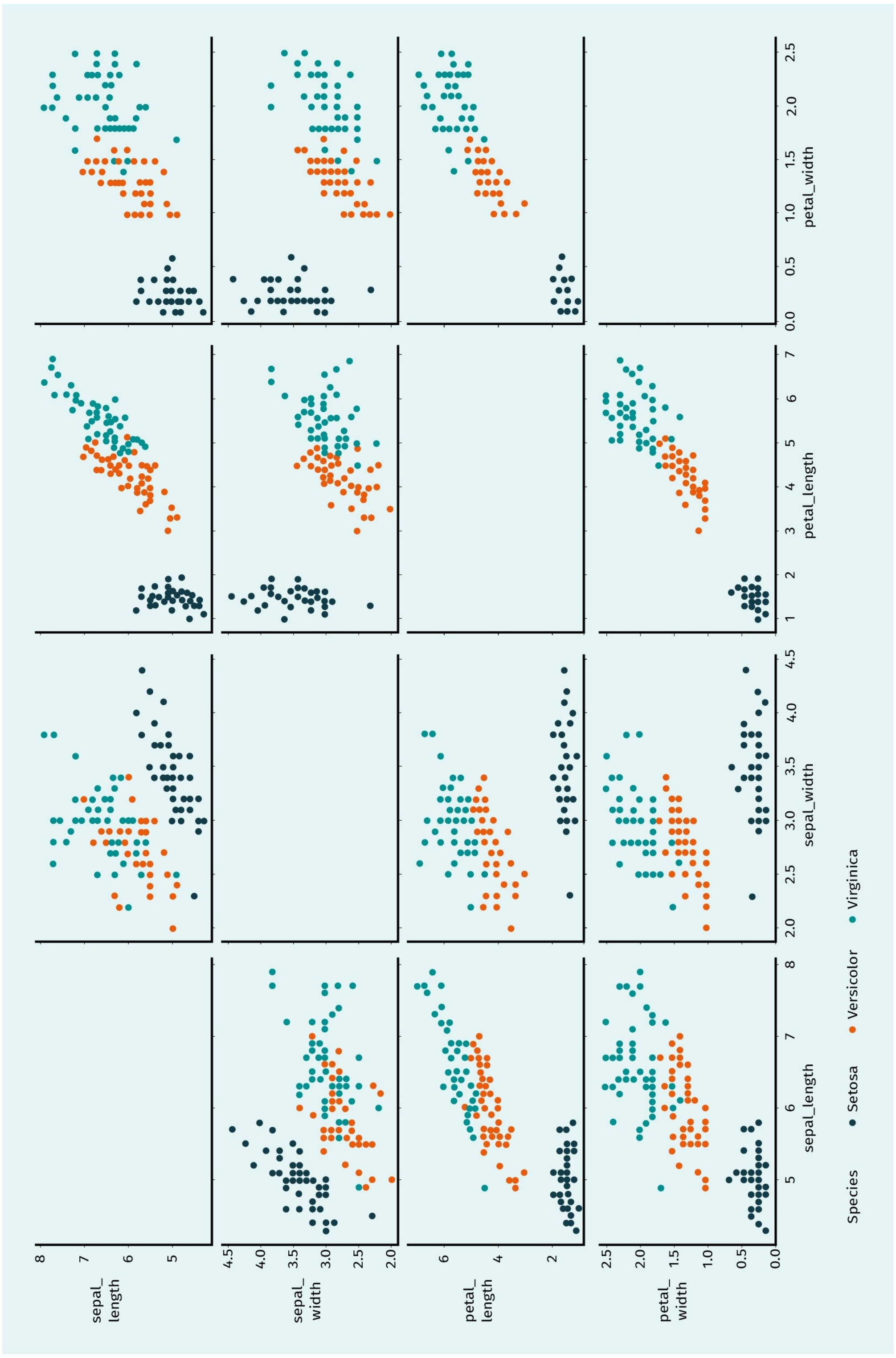
```
from sklearn.manifold import MDS
from sklearn.preprocessing import MinMaxScaler

# load the sample data set
iris = datasets.load_iris()
X = iris.data

# normalize the data
X_scaled = MinMaxScaler().fit_transform(X)
```

2. We run MDS to reduce to two dimensions and visualize the projected Iris data of four original features into the reduced two dimensional space as follows:

   Code
   ```
   # conduct MDS on the data
   mds = MDS(2,random_state=0)
   X_2d = mds.fit_transform(X_scaled)

   # Plot the projected Iris data points into the reduced
   # feature space by MDS
   plt.scatter(x=X_2d[:,0], y=X_2d[:,1], c=iris.target)
   plt.show()
   ```

The figure below shows the Iris data points projected into each of the two-dimensional feature spaces formed by the original features.

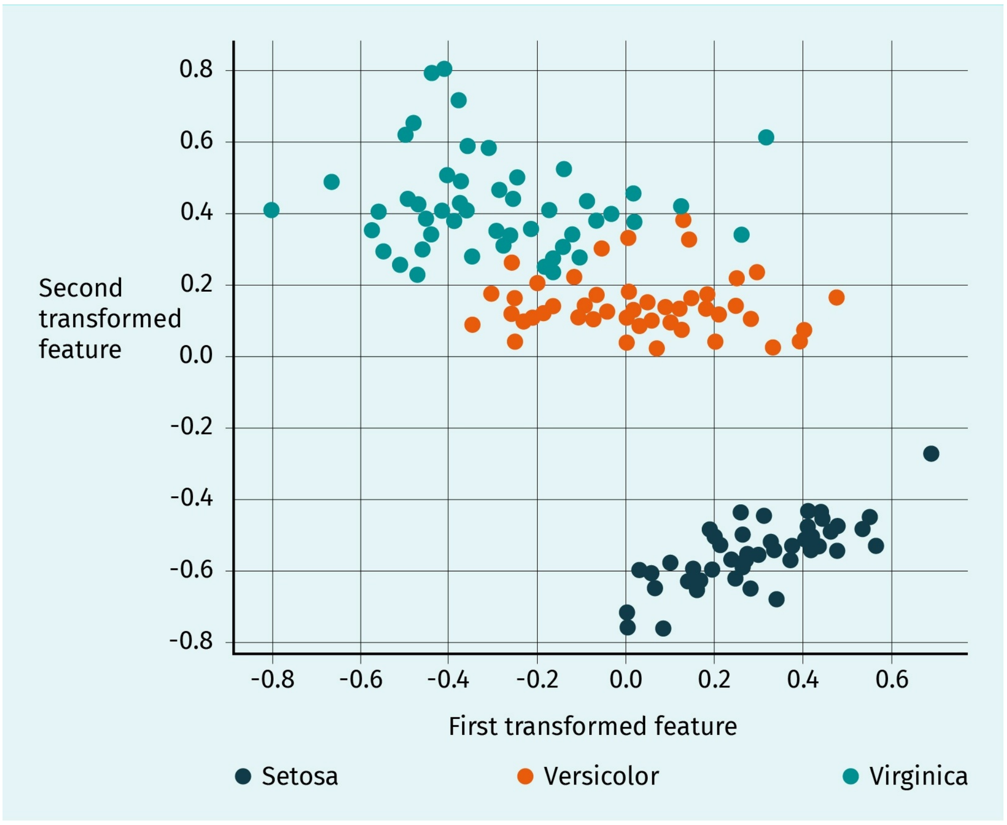
Figure 32: Iris Data Points Projected into a Two-Dimensional Feature Space



Source: Sayed-Mouchaweh (2021).

The figure below shows the projected Iris data samples into the reduced two-dimensional feature space. By comparing the figure above with the figure below, we can see clearly that the pairwise distances have been conserved in the reduced feature space (the data samples in the blue or *setosa* cluster are far away from the data samples of the other two clusters).

Figure 33: Iris Data Points Projected into a Two-Dimensional Feature Space Obtained by Metric MDA



Source: Sayed-Mouchaweh (2021).

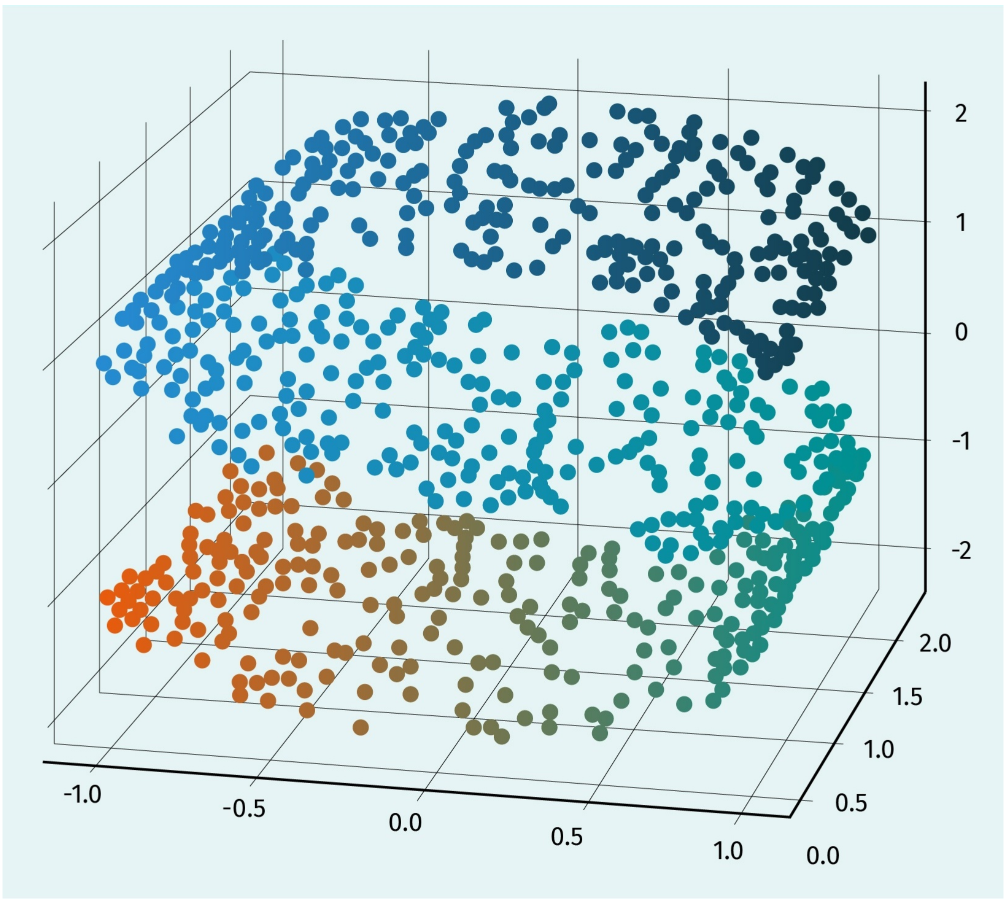
Advantages and Drawbacks

MDS can be used to simplify exploration analysis of multidimensional data samples thanks to the dimensionality reduction. It allows the observation of the distances or dissimilarities between these data samples. This observation is very useful for many applications, such as fault diagnosis or credit card fraud. Indeed, data samples that characterize normal operation conditions or genuine behavior tend to be close to each other through clusters, while data samples characterizing abnormal operation conditions or fraudulent behavior will be far away from, or dissimilar to, normal or genuine data samples. MDS can also be used to clean a dataset by detecting outliers before the clustering step, or to observe if data samples are self-organized into clusters.

In addition, MDS can be used to project data samples that manifest nonlinear correlations. For example, let us use the **Swiss roll dataset** illustrated in the figure below. It is clear that these data samples have a nonlinear structure in the three-dimensional feature space.

Swiss roll dataset
The Swiss roll dataset is a set of data points generated in three-dimensional feature space with a nonlinear structure. These data points can be described in two-dimensional feature space.

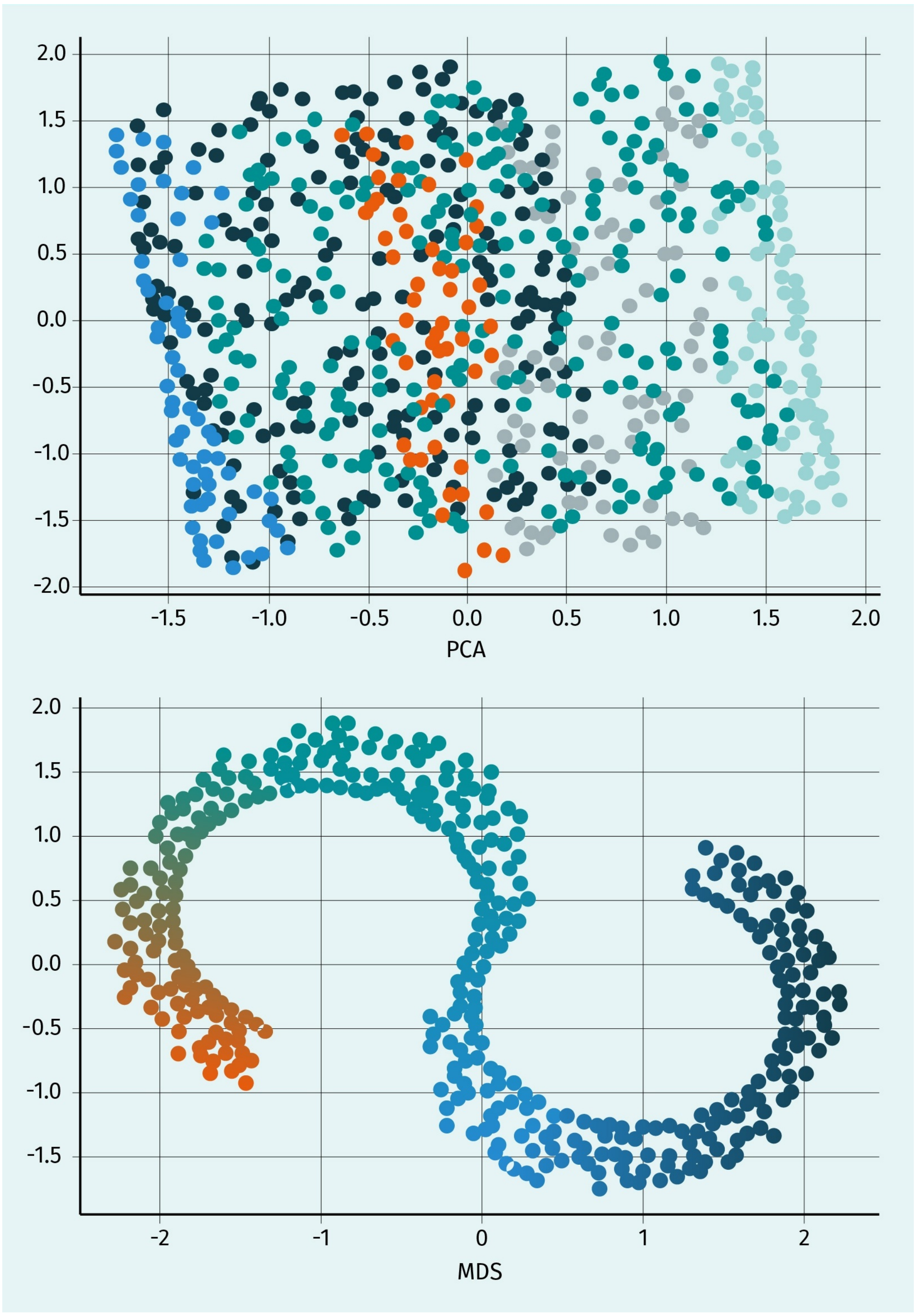
Figure 34: Swiss Roll Dataset with Nonlinear Structure



Source: Sayed-Mouchaweh (2021).

PCA fails to highlight this nonlinear structure by projecting the data samples into a reduced feature space formed by the top two principal components, as illustrated in the figure below. Indeed, since PCA is a linear dimensionality reduction approach, it fails to conserve the neighborhood information between highly dissimilar orange and navy blue points (center). Meanwhile, MDS perfectly conserves neighborhood information due to its conservation of the pairwise distances between projected data samples as illustrated in the figure below.

Figure 35: Projection of Swiss Roll Dataset into Two-Dimensional Feature Space



Source: Sayed-Mouchaweh (2021).

However, the computation complexity of MDS is proportional to the number of data points in the dataset. Therefore, MDS cannot be used for too large datasets because it becomes too slow. For instance, the required time for the Swiss roll dataset including 1,000 data samples is 29.5 seconds, while for PCA it is 0.003 seconds. The time is computed using a single Intel Core i7 microprocessor. The following Python code allows us to generate the obtained results in the figure above. The steps are as follows:

1. Import the required Python libraries.

   Code
   ```
   from sklearn.manifold import MDS
   from mpl_toolkits.mplot3d import Axes3D
   from sklearn import manifold, datasets
   ```

2. Generate 1,000 data points of the Swiss roll and display them.

   Code
   ```
   ax = plt.axes(projection='3d')
   n_pts = 1000
   X, color = datasets.make_s_curve(n_pts, random_state=0)
   ax.scatter3D(X[:, 0], X[:, 1], X[:, 2], c=color)
   plt.show()
   ```

3. Project the 1000 data points of the Swiss roll into a two-dimensional feature space using MDS and calculate the required time for this operation.

   Code
   ```
   n_components = 2
   mds = MDS(2,random_state=0)
   X_2d = mds.fit_transform(X)
   ```

4. Display the obtained projection in the two-dimensional feature space.

   Code
   ```
   plt.scatter(X_2d[:,0], X_2d[:,1], c=color)
   plt.show()
   ```

3.3 Local Linear Embedding

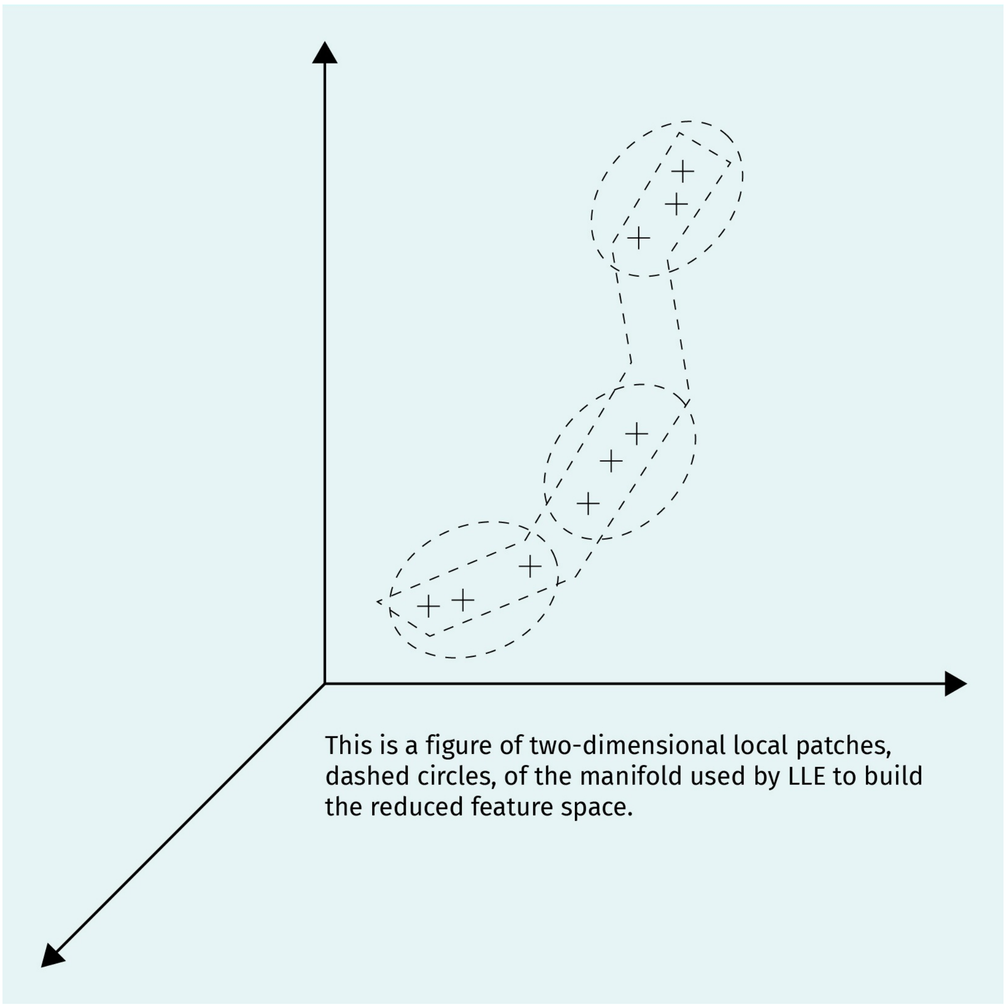
How It Works

Local linear embedding (LLE) (Saul & Roweis, 2000) is an unsupervised dimensional reduction method that allows the projection of datasets with nonlinear structures into reduced feature space. It preserves the distances within local neighborhoods when projecting data samples into the reduced feature space.

Manifold
A manifold is an object embedded in a higher dimensional space (e.g., a two-dimensional plan in a three-dimensional space).

Let us take the example of the figure below. It shows a **manifold** comprising data samples X with nonlinear structure. LLE starts by creating locally linear patches around each data sample x from X in the manifold. Each local patch includes k-nearest neighbors for the corresponding data sample. Each patch can be seen as a flat plan or circle where the data sample x and its k-nearest neighbors lay on the same flat plane or circle. Then each data sample x will be approximated by a weighted linear combination of its k-nearest neighbors. These weights W are selected in such a way that allow us to reconstruct the corresponding data sample x from its nearest neighbors by minimizing a cost function, e.g., the sum of the squared errors between the data sample and the reconstructed data sample by its nearest neighbors.

Figure 36: Manifold

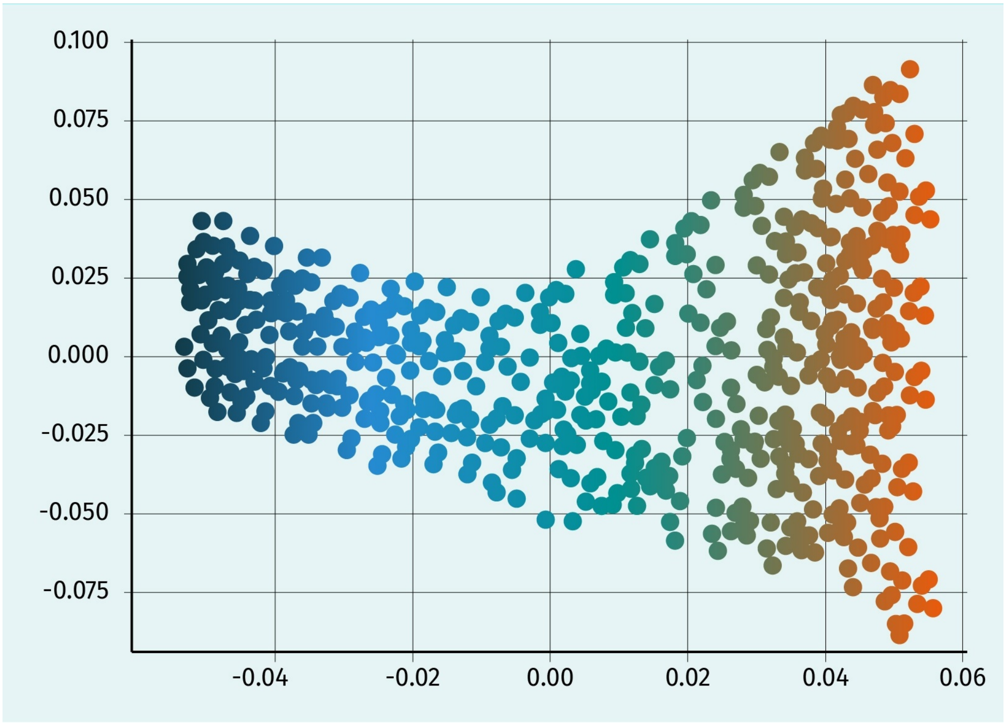

This is a figure of two-dimensional local patches, dashed circles, of the manifold used by LLE to build the reduced feature space.

Source: Sayed-Mouchaweh (2021).

Finally, LLE transforms or maps each data point x of X to a point y in a reduced feature space by using the computed weights W. The transformed data samples Y are computed by minimizing an embedded cost function, e.g., the sum of the squared errors between the transformed data sample y and the reconstructed data sample by its nearest neighbors in the reduced feature space. The embedded cost function is optimized by fixing the weights while optimizing the coordinates of each y in Y (Saul & Roweis, 2000).

Let us run LLE using Python and apply it to the Swiss roll dataset. We need to import the function LocallyLinearEmbedding from the library sklearn.manifold. Let us specify the number of nearest neighbors k to be 12. We project the data points into a two-dimensional transformed feature space, i.e., the number of components is equal to two. Let us compute the required time to perform this projection and plot the projected data samples into this two-dimensional feature space. The figure below shows the projected points by LLE into the two-dimensional space. We can see clearly that the local distances, local neighborhoods, are conserved in the reduced feature space. For instance, in the high dimensional space, the navy blue data samples have the largest distance from the orange data samples. The reduced feature space conserves this distance (dissimilarity) while, at the same time, gathering the data samples of the same color.

Figure 37: Swiss Roll Dataset Projected into Two-Dimensional Feature Space Obtained by LLE



Source: Sayed-Mouchaweh (2021).

The Python code required to provide these results is as follows:

Code

```
# import the required libraries
import matplotlib.pyplot as plt
from sklearn.manifold import LocallyLinearEmbedding
from sklearn import manifold, datasets

# generate 1000 data points of the Swiss roll dataset
n_pts = 1000
X, color = datasets.make_s_curve(n_pts, random_state=0)

# apply LLE to the generated Swiss roll dataset
# to project it into a 2-dimensional feature space
embedding = LocallyLinearEmbedding(n_neighbors=12, \
  n_components=2)
X_2d = embedding.fit_transform(X)
```

```
# display the data points in the
# reduced feature space
plt.scatter(X_2d[:,0], X_2d[:,1], c=color)
plt.show()
```

Advantages and Drawbacks

As we have seen in the figure above, LLE succeeded at unrolling the Swiss roll dataset so the data points of one cluster are not overlapped with the other data points of the other clusters. LLE can be considered an efficient unsupervised dimensional reduction approach for datasets with nonlinear structures. LLE has multiple advantages. These are as follows:

- It is easy to implement and use.
- It requires only the number of nearest neighbors k and the number of dimensions to reduce.
- Its computation time is much lower than the other dimensionality reduction approaches, particularly MDS. Therefore, it is more suitable for large datasets.

However, LLE suffers from its sensitivity to noises and outliers since it is based on the value of the k-nearest neighbors. In addition, some neighbors can be mixed in different patches entailing the creation of short-circuit problems between patches.

SUMMARY

This unit presented and discussed the functioning principal and the advantages and drawbacks of the major dimensionality reduction approaches: Principal Component Analysis (PCA), Multi-Dimensionality Scaling (MDS), and Locally Linear Embedding (LLE). The advantages and drawbacks of these approaches were highlighted in order to identify the suitable conditions of their use.

Two major criteria were proposed to guide the choice of PCA, MDS, and LLE to apply with respect to the dataset's conditions and characteristics. These criteria include the correlation (linear and nonlinear) between the data samples and the dataset's size (small and large). For instance, LLE is the most suitable dimensionality reduction approach to apply when the data have a nonlinear structure and the number of data samples is large.

UNIT 4

FEATURE ENGINEERING

STUDY GOALS

On completion of this unit, you will be able to ...

- explain the difference between numerical, categorical, and text features.
- clean, scale, encode, or transform these features.
- generate new features by transforming, splitting, or grouping existing features as interaction features.

4. FEATURE ENGINEERING

Introduction

Machine learning approaches require input data in order to provide an output, e.g., clusters gathering similar data points. The input data points are generally structured in a table or array where the rows represent the data points and the columns represent their features. The latter describe certain properties of the data points. For instance, the price of a product, the age of a client, etc. The performances, e.g., quality of the obtained clusters or predictive power, depend largely on the quality or reliability of the data features. The more those features represent salient properties of the data, the better the provided knowledge or insights from the machine learning approaches. Therefore, an important effort must be exerted to build "good" features.

Feature engineering is the process that allows us to build features from raw data automatically or manually using domain knowledge. It is worth mentioning that building meaningful features will improve model performances, speed up the model's learning and processing, and facilitate the understanding of the provided output.

Datasets often contain different types of features, such as numerical, categorical, and text features. They present quantitative (natural ordering and range of values) and qualitative (no natural ordering and come from fixed list of values) properties of the data. The way that these features are represented has an enormous effect on the performances of the built machine learning model.

Gender
Please note that in this unit, gender refers to biological sex.

Let us take an example of a dataset representing the energy consumption by clients (see the table below). This dataset is an array of n rows indicating the clients and $d = 6$ columns representing the client identification number, the client **gender** (male or female), their work type (self-employed, state employee, or private), the number of occupants, and their consumption in kilowatts per hour.

Table 8: Energy Consumption Dataset

C-ID	Gender	Work-type	Client-satisfaction	Number-occupants	Consumption
1	M	1	3	2	70
2	F	2	0	4	140
3	M	2	4	2	65
4	F	3	3	1	40
5	F	3	5	2	65

Source: Sayed-Mouchaweh (2021).

We can see that the dataset contains two types of features: categorical and numerical. The client gender is a categorical feature since its possible values do not have a natural ordering. This is the same for the client's work-type. Meanwhile, the number of occupants and amount of energy consumption are numerical features since they have a natural ordering.

Machine learning approaches require a suitable, numerical representation of data features to build a model. To this end, a dataset needs to be pre-processed in order to clean the data samples and convert features into a suitable representation form. For instance, the energy consumption dataset cannot be used in its current state as and input for a machine learning approach. The raw data points need to be preprocessed, their features converted, and new features constructed. This unit presents the different techniques used to preprocess the raw data, convert their features, and construct new features. Therefore, this unit examines the following questions:

- What are the different techniques used to preprocess the raw data?
- What are the different techniques used to construct new features?
- How do we implement the preprocessing and new feature construction techniques in Python?

4.1 Numerical Features

There are two fundamental levels of measurement for numerical features: interval and ratio scales. Interval scales represent ordered elements with a specific interval of the same difference. Consider the temperature degree in a room. Twenty degrees Celsius is not double 10 degrees Celsius since there is no true zero in an interval scale. Indeed, in order to say that 20 degrees Celsius is double of 10 degrees Celsius, zero must be considered the reference. This is not possible since there are negative temperatures. Therefore, the interval between 20 and 10 degrees Celsius is the same as the interval between 20 and 30 degrees Celsius. The interval between 20 and 10 degrees Celsius is the double of the interval between 20 and 25 degrees Celsius.

A ratio scale has all the characteristics of an interval scale and can integrate the value of "zero" into any of its features. For instance, the age is a ratio scale feature. If you are 20 years old, you are twice the age of a ten-year-old. This is because it is possible to have the "zero" as a reference since there is no negative age.

Feature Cleaning (Imputing Missing Values)

Datasets may have missing values for some features (columns) for different reasons, such as discomfort, privacy, or recording errors. For example, some volunteers may not disclose their age, weight, or salary because this is personal information. Missing values can manifest as blanks, by "NA" (not available), or by "NaN" (not a number) in the corresponding columns.

In general, machine learning approaches underperform or do not work at all when missing values are present in the dataset table. Therefore, these missing values must be treated either by removing the corresponding rows or by imputing them. When there is a relatively high number of missing values in a small dataset, removing the rows of those missing data may impact the performances of the machine learning approaches and bias the obtained output. Therefore, it is better to impute them using one of many existing imputing techniques, such as substitution (mean, median, etc.), regression imputation, stochastic regression imputation, interpolation, and extrapolation imputation. As an example, let us take the following simple imputation strategies: imputing by mean value, median value, or the most frequent value (mode value).

Let us use the example of energy consumption. Suppose that the energy consumption of Client 5 is not available, as illustrated in the table below. We can replace this missing value by using either the mean value, the median value, or the most frequent value of the available energy consumptions, as illustrated in the table below.

Table 9: Min-Max Scaling on the Consumption Feature

C-ID	Consumption	Missing value replaced by mean value	Missing value replaced by median value	Missing value replaced by most frequent value
1	70	70	70	70
2	140	140	140	140
3	65	65	65	65
4	NaN	85	67.5	65
5	65	65	65	65

Source: Sayed-Mouchaweh (2021).

Let us apply the imputation of missing values using Python:

1. Import pandas, numPy, and Python function SimpleImputer from Python library sklearn.impute, and create the dataset of energy consumption as a DataFrame with the missing value as it is illustrated in the table above.

   Code
   ```
   Table = {\
      'Customer-ID': [1, 2, 3, 4, 5], \
      'Gender': ['M','F','M','F','F'], \
      'Work-type': [1, 2, 2, 3, 3], \
      'Client-satisfaction':[3, 0, 4, 3, 5], \
   ```

```
    'Number-occupants': [2, 4, 2, 1, 2],
    'Consumption':[70, 140, 65, np.NaN, 65]}
TDF = pd.DataFrame(data=Table)
```

2. Apply the Python function SimpleImputer, and specify the type of missing value (e.g., NaN) and the strategy to impute it (mean, median, or most-frequent value). Here, we chose the "mean" strategy.

   Code
```
input = SimpleImputer(missing_values = np.nan, \
    strategy = 'mean')
imput = imput.fit(TDF[['Consumption']])
imput = imput.transform(TDF[['Consumption']])
TDF['Consumption']= imputimput = SimpleImputer(missing_values = np.nan, strategy =
```

Impute missing values by using mean, median, or mode values, which are also called placeholder values. This is a suitable choice when there are few missing values in a column (feature) and/or the variance of the data points in this column is not too large. If this is not the case, it is better to choose a more sophisticated imputation technique in order to avoid a biased estimation of parameters. The latter arises when the data points are purposefuly missing. Hence, the parameters' estimation of a model may be biased if the missing points are improperly imputed.

To mitigate this problem, one alternative is the use of the regression imputation technique. In this technique, missing data are imputed as functions of the value of the other features (e.g., variables and columns). Several regression models can be used to predict or estimate the missing data points, such as Bayesian ridge regression or extra-trees regressor.

Consider the following example:

$$D = \begin{bmatrix} Var1 & Var2 \\ 10 & 2 \\ NaN & 1 \\ 2 & 0.4 \\ 1 & 0.2 \\ 5 & NaN \end{bmatrix}$$

We can see that values in "Var1" are five times the values in "Var2." The regression model will learn this linear relationship between "Var1" and "Var2" in order to impute the missing values in both "Var1" and "Var2."

$$D = \begin{bmatrix} Var1 & Var2 \\ 10 & 2 \\ 5 & 1 \\ 2 & 0.4 \\ 1 & 0.2 \\ 5 & 1 \end{bmatrix}$$

Let us apply the regression imputation technique using Python:

1. Import the required Python libraries. We will import two different regression models in order to compare the obtained imputation. “BayesianRidge” is the regression method used by default by the function “IterativeImputer.”

   Code
   ```
   from sklearn.experimental import enable_iterative_imputer
   from sklearn.impute import IterativeImputer
   from sklearn.ensemble import ExtraTreesRegressor
   from sklearn.linear_model import BayesianRidge
   ```

2. Create the data table or matrix D, transform it into a DataFrame, then apply the IterativeImputer function using the two regression methods “Bayesian Ridge” and “Extra Trees Regressor,” and display both results.

   Code
   ```
   # generate sample data
   table = {'Var-1': [10, np.NaN, 2, 1, 5],\
           'Var-2': [2, 1, 0.4, 0.2, np.NaN]}
   TDF = pd.DataFrame(data=table)
   TDF
   # console output:
   #  Var-1    Var-2
   # 0 10.0 2.0
   # 1 NaN    1.0
   # 2 2.0    0.4
   # 3 1.0    0.2
   # 4 5.0    NaN

   # apply regression imputation using ‘Bayesian Ridge’
   imputbr = IterativeImputer(BayesianRidge())
   TDF = pd.DataFrame(imputbr.fit_transform(TDF))
   TDF
   # console output:
   #  0       1
   # 0 10.000000 2.000000
   # 1 5.000531 1.000000
   # 2 2.000000 0.400000
   # 3 1.000000 0.200000
   ```

```
# 4 5.000000 0.999973

# apply regression imputation using
# ‘Extra Trees Regressor’
TDF = pd.DataFrame(data=table)
imputetr = IterativeImputer(ExtraTreesRegressor())
TDF = pd.DataFrame(imputetr.fit_transform(TDF))
TDF
# console output:
#  0      1
# 0 10.00    2.000
# 1 5.35     1.000
# 2 2.00     0.400
# 3 1.00     0.200
# 4 5.00     0.898
```

Feature Scaling

Models built by the majority of machine learning approaches, such as k-means, are sensitive to features scales. Machine learning approaches that involve distance calculation, such as clustering approaches, are affected by the magnitude of variation of the input features. Indeed, features with large variation will have much more influence on the distance between two data points than the features with small range of variation. For instance, if we have two features describing employees in a company, their age and their height in meters, it is obvious that the age will be overrepresented with respect to the height. Indeed, the age range of employees is much greater than their height range.

To avoid the impact of different feature ranges on the performances of the built models, we apply feature scaling. It is a technique used to normalize or standardize the range of input features. Feature scaling is done to each independent feature during the preprocessing step of raw data. The various feature scaling techniques are described in the following sections.

Min-max scaling

The first technique is min-max scaling. It aims to compact all the feature values in a dataset to be within the range [0, 1]. Let $\mathrm{x_F}$ be the value of a data point in a dataset X according to a certain feature F. Let $\mathrm{min_F}$and $\mathrm{max_F}$ be, respectively, the minimum and maximum values of all the data points according to the feature F. The min-max scaling of x is

$$\widetilde{x_F} = \frac{x_F - \min\limits_F}{\max\limits_F - \min\limits_F}$$

Let us take the “consumption” feature of the energy consumption example. Min-max scaling on the “consumption” feature is illustrated in the table below.

Table 10: Min-Max Scaling on the Consumption Feature

C-ID	Consumption	Min-max scaling of consumption
1	70	0.3
2	140	1
3	65	0.25
4	40	0
5	65	0.25

Source: Sayed-Mouchaweh (2021).

Let us use the Python package min-max scaling:

1. Import pandas and Python function MinMaxScaler from Python library sklearn.preprocessing, and create the energy consumption dataset as a DataFrame.

   Code
   ```
   # import libraries
   import pandas as pd
   from sklearn.preprocessing import MinMaxScaler

   # generate sample data
   Table = {\
      'Customer-ID': [1, 2, 3, 4, 5], \
      'Gender': ['M', 'F', 'M',' F', 'F'], \
      'Work-type': [1, 2, 2, 3, 3], \
      'Client-satisfaction': [3, 0, 4, 3, 5], \
      'Number-occupants': [2, 4, 2, 1, 2], \
     'Consumption': [70, 140, 65, 40, 65]}
   TDF = pd.DataFrame(data=Table)
   ```

2. Apply the min-max scaling on the feature "consumption" and display the obtained results. In this example, we only apply the scaling to one column. This process can also be conducted on many columns, or all columns of a dataframe if these columns are numeric. Verify that the min-max scaling of "consumption" is between 0 and 1.

   Code
   ```
   # apply Min-Max Scaling on the feature 'Consumption'
   MMS = MinMaxScaler().fit_transform(TDF[['Consumption']])
   TDF['Consumption'] = MMS

   print(TDF['Consumption'])
   # console output:
   #     0.30
   # 1   1.00
   ```

```
# 2   0.25
# 3   0.00
# 4   0.25

print(TDF['Consumption'].describe())
# console output:
# count   5.000000
# mean    0.360000
# std     0.376497
# min     0.000000
# 25%     0.250000
# 50%     0.250000
# 75%     0.300000
# max     1.000000
```

The drawback of min-max scaling could appear if the data points are skewed or if there are outliers. Indeed, compacting the range of data points between 0 and 1 entails the loss of information since outliers create important useless part of the [0, 1] range. This impacts the quality of obtained clusters or the predictive power of the built model.

Standardization (variance scaling)

This is also called Z-score scaling, or normalization, and can be obtained by the following:

$$\frac{x_F - mean_F}{\sigma_F}$$

It removes the mean value, $\mathrm{mean_F}$, from the data points, $\mathrm{x_F}$, and scales them to the unit variance by dividing by the standard deviation, σ_F (square root of the variance). Therefore, this is called variance scaling. The obtained scaled feature has a mean of 0 and variance of 1.

Let us take the “consumption” feature of the energy consumption example. The resulted “consumption” feature after the standardization is illustrated in the table below.

Table 11: Standardization of Consumption Feature

C-ID	Consumption	Standardization of consumption
1	70	—0.18
2	140	1.9
3	65	—0.33
4	40	—1.07
5	65	—0.33

Source: Sayed-Mouchaweh (2021).

Let us apply the standardization using Python:

1. Import Python function StandardScaler from Python library sklearn.preprocessing.

   Code
   ```
   from sklearn.preprocessing import StandardScaler
   ```

2. Apply the standardization on the feature “consumption” and display the obtained results. Verify that the standardization of “consumption” provides a mean value equal to zero and unit variance.

   Code
   ```
   # apply standardization on the feature 'Consumption'
   TDF = pd.DataFrame(data=Table)
   ST = StandardScaler().fit_transform(TDF[['Consumption']])
   TDF['Consumption'] = ST

   print(TDF['Consumption'])
   # console output:
   # 0  -0.178174
   # 1   1.900524
   # 2  -0.326653
   # 3  -1.069045
   # 4  -0.326653

   print(TDF['Consumption'].describe())
   # console output:
   # Name: Consumption, dtype: float64
   # count   5.000000e+00
   # mean   -3.330669e-17
   # std     1.118034e+00
   # min    -1.069045e+00
   # 25%    -3.266526e-01
   # 50%    -3.266526e-01
   # 75%    -1.781742e-01
   # max     1.900524e+00
   ```

The standardization has the same drawback as min-max scaling for sparse data points (e.g., if the data points are skewed or if there are outliers). This impacts the quality of obtained clusters or the predictive power of the built model.

Robust scaling

Robust scaling removes the feature **median** and scales the data points to the InterQuartile Range (IQR). It is calculated as follows:

$$\frac{x_F - Median_F}{IQR_F}$$

Median
The median is the value separating the higher half of data points from the lower half when ordered from lowest to highest.

Indeed, the data points, when ordered from lowest to highest, are divided into four equal parts, each described by a quartile: Q1, Q2, Q3, and Q4. IQR describes the middle 50 percent of data points between Q1 and Q3.

Let us take the "consumption" feature of the energy consumption example (see the table below). The resulting "consumption" feature after applying the robust scaling is illustrated in the table below.

Table 12: Robust Scaling of Consumption Feature

C-ID	Consumption	Robust scaling of consumption
1	70	1
2	140	15
3	65	0
4	40	—5
5	65	0

Source: Sayed-Mouchaweh (2021).

Again, we apply the robust scaling using Python:

1. Import Python function RobustScaler from Python library sklearn.preprocessing.

   Code
   ```
   from sklearn.preprocessing import RobustScaler
   ```

2. Apply robust scaling on the feature "consumption" and display the obtained results.

   Code
   ```
   RS = RobustScaler().fit_transform(TDF[['Consumption']])
   TDF['Consumption'] = RS
   print(TDF)
   ```

Thanks to the use of the median and IQR of data points, rather than minimum and mean, robust scaling has the advantage of being more adapted to scale-sparse data points than min-max scaling and standardization. Let us compare the three feature scaling techniques with respect to a sparse dataset. Let us suppose that in the "consumption" feature, one outlier exists, as illustrated in the table below. Let us apply the three feature scaling techniques to this feature. The obtained results are shown in the table below. We can clearly see that min-max scaling and standardization compress most of the data points to a narrow range, while robust scaling does much better at keeping the spread of the data points.

Table 13: Comparison between Feature Scaling Techniques in the Case of Sparse Dataset

C-ID	Consumption	Min-max scaling of consumption	Standardization of consumption	Robust scaling of consumption
1	70	0.004	—0.50	0
2	140	0.015	—0.47	0.93
3	65	0.003	—0.50	—0.06
4	40	0.000	—0.51	—0.4
5	**6500**	**1**	**2**	**85.73**

Source: Sayed-Mouchaweh (2021).

Feature Generation

New features can be generated as a transformation or combination of existing features. These new features may add domain knowledge to the dataset, improving the performance of the built model by using machine learning approaches. New features can be generated as either transformed or interaction features.

Transformed features

Transforming features by applying mathematical operations allows the generation of new features. The aim of transforming features is to simplify their treatment or their interpretation. For instance, making a **skewed distribution** as normal as possible allows us to meet this assumption (normal distribution) as is required by most statistical models. In addition, transforming features can help to meet the assumption of constant variance in the case of linear modeling by making a nonlinear relationship more linear. There are many transformation approaches. These include:

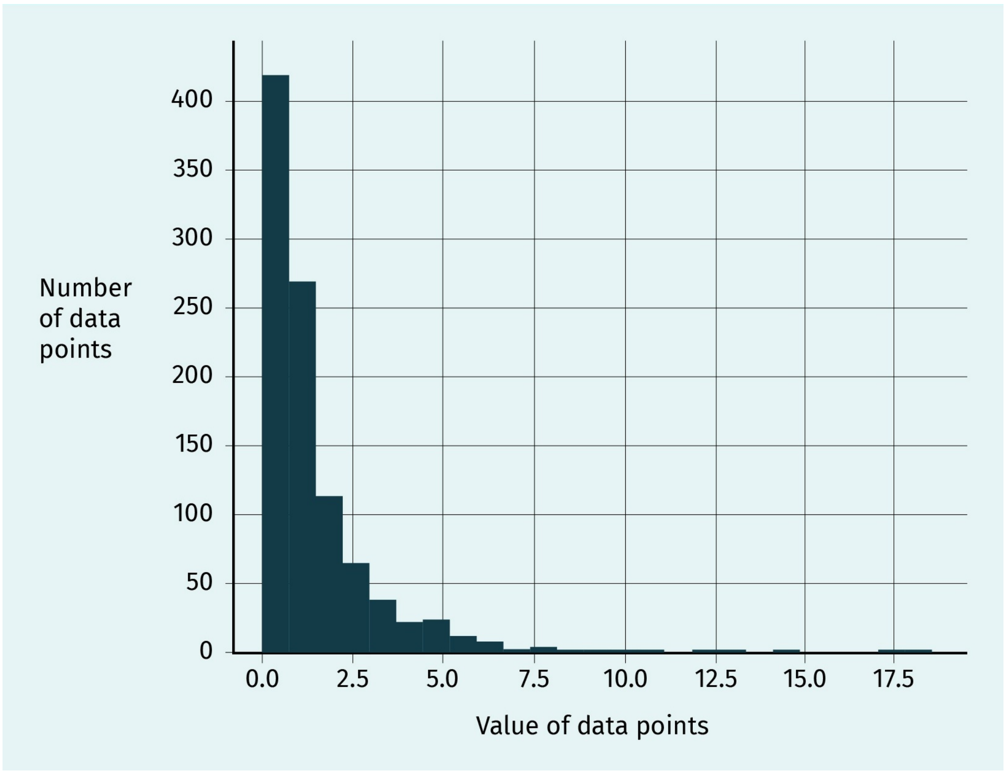
Skewed distribution
This type of distribution is an asymmetric distribution with a long left or right tail. Normal distribution is symmetric distribution around the mean value. Most real-life distributions are skewed.

- min (finding the minimum value for a feature or a column)
- max (finding the maximum value for a feature or a column)
- σ(finding the standard deviations for a feature or a column)
- variance
- mean (finding the average value for a feature or a column)
- median
- sum
- count (count the number of occurrences of an element in a feature or column)
- log transformation (replacing each value of a feature with its logarithm)
- quantile transformation
- exponential transformation
- square root transformation
- reciprocal transformation

The choice of one of them depends on the available dataset and domain application. In some cases, the use of the reciprocal transformation ($1/\mathrm{x}$, where x should not be zero) may ease the interpretation. For example, if x represents the population density (i.e., people per unit area), applying the reciprocal transformation generates a new feature indicating the area per person. The log transformation has an effect on the shape of data distribution. It transforms skewed data distribution to a normal-like shape in order to simplify its treatment by the statistical models. In addition, log transformation helps to make a nonlinear relationship more linear (e.g., $\mathrm{Log}(\mathrm{x}^2) = 2\mathrm{Log}(\mathrm{x})$).

Log transformation is useful for data points generated by skewed distribution. Often, the distribution of data points is not normal, and log transformation allows us to transform skewed data points to approximately conform to normal distribution. The figure below shows an example of a skewed distribution of data points.

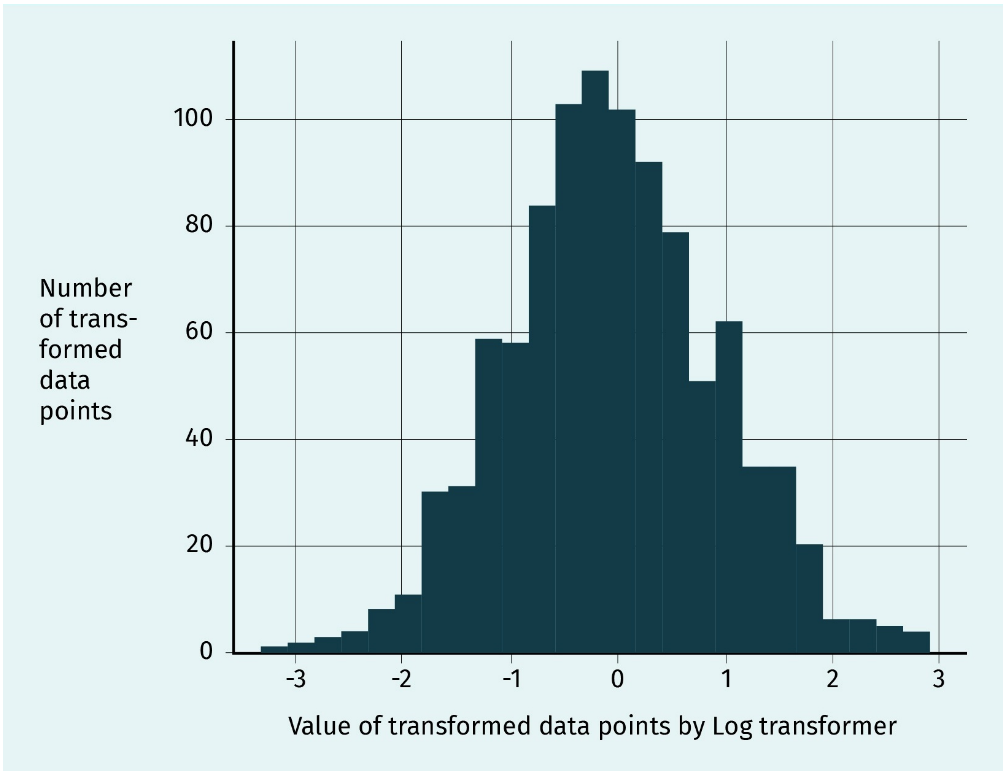
Figure 38: Histogram of a Skewed Distribution of Data Points



Source: Sayed-Mouchaweh (2021).

The figure below shows the histogram of transformed data points (of the figure above) using Log Transformer. We can see that the new transformed distribution follows a normal distribution.

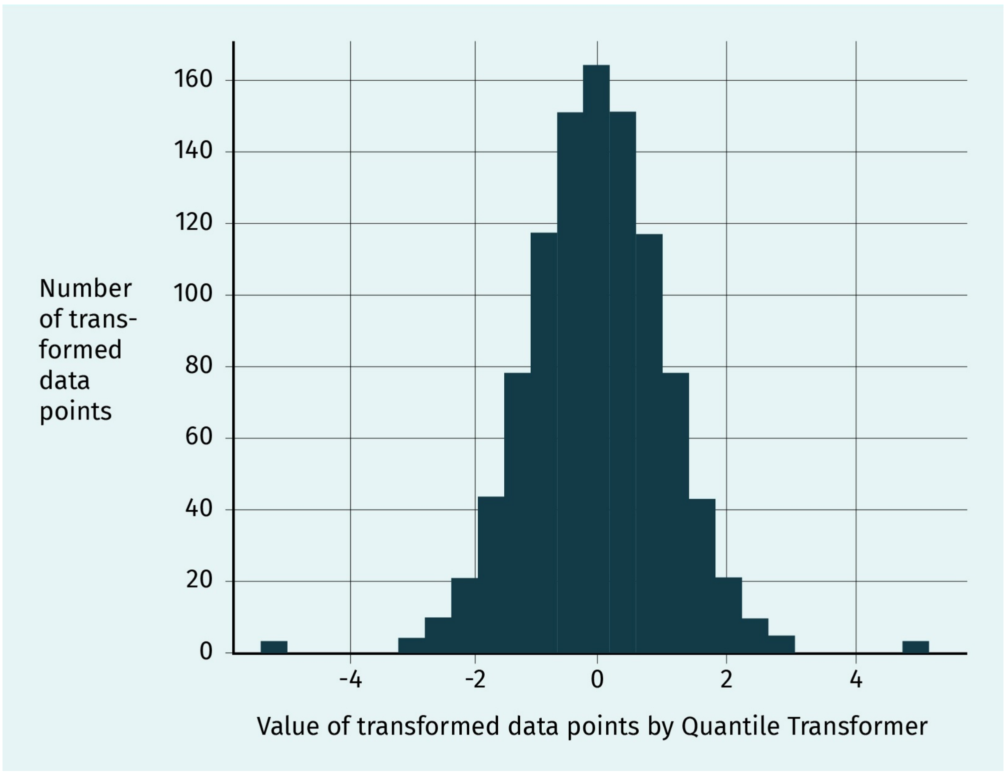
Figure 39: Histogram of Transformed Data Points by Log Transformer



Source: Sayed-Mouchaweh (2021).

The figure below shows the histogram after transforming the skewed distribution using Quantile Transformer. We can see that the new transformed distribution follows a normal distribution.

Figure 40: Histogram of Transformed Data Points by Quantile Transformer



Source: Sayed-Mouchaweh (2021).

Nevertheless, the data points must follow a log-normal distribution, and data points must have only positive values. Another alternative transformation used to approximate a skewed distribution to a normal distribution is the quantile transformation since it can reduce the effect of marginal (skewed) data points. However since it is a nonlinear transformation, it may distort the linear correlation between features.

Let us perform both transformations using Python functions `LogTransformer` and `QuantileTransformer` in order to generate transformed features, as illustrated in the figures above.

1. Install the library `feature-engine` on cmd.

   Code
   ```
   pip install feature-engine
   ```

2. Import libraries and then generate data points from a normal distribution, then add a skew to them in order to generate skewed data points, and display the histogram of the skewed data points. The latter's distribution has a tail.

Code
```
# generate sample data from a normal distribution
dat = randn(999)

# add a skew to the generated data points
dat_skew = exp(dat)
plt.hist(dat_skew, bins = 25)
plt.show()
```

3. Import the `LogTransformer` function, and apply it to the skewed data points, then display the histogram of transformed data. The latter shows that the transformed skewed data points follow a normal distribution.

   Code
   ```
   # convert the data into a dataframe
   dat_skew = dat_skew.reshape((len(dat), 1))
   dat_skew = pd.DataFrame(dat_skew, columns = ['Value'])

   # generate and fit log transformer
   lgt = vt.LogTransformer(variables= ['Value'])
   lgt.fit(dat_skew)

   # apply log transformation
   dat_lg = lgt.transform(dat_skew)

   # plot the distribution of the transformed data
   plt.hist(dat_lg['Value'], bins=25)
   plt.show()
   ```

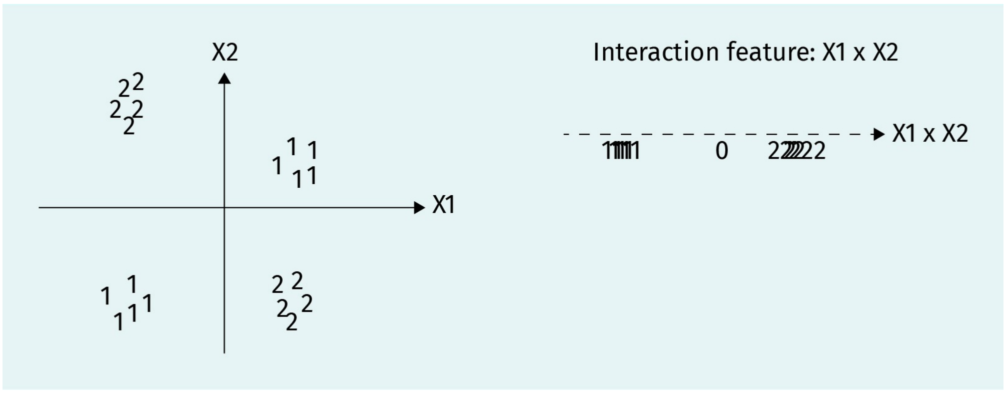
4. Import the functions `QuantileTransformer` and apply it to the skewed data points, then display the histogram of transformed data points. The latter shows that the transformed skewed data points follow a normal distribution.

   Code
   ```
   # generate and fit quantile transformer
   qt = QuantileTransformer(output_distribution='normal')
   qt.fit(dat_skew[['Value']])

   # apply quantile transformation
   dat_q = qt.transform(dat_skew[['Value']])

   # plot the distribution of the transformed data
   plt.hist(dat_q, bins=25)
   plt.show()
   ```

Interaction features

Interaction features, or cross product features, allow the combination of pairs of features through the logical operator AND. Let us take the example of the figure below. We have two different clusters described in two-dimensional feature space. We can see that clusters 1 and 2 have positive and negative values according to each feature. Cluster 1 can be described by the rule "(If a datapoint x has positive value according to the feature $\mathrm{X1}$ AND positive value according to $\mathrm{X2}$), OR if x has a negative value according to $\mathrm{X1}$ AND a negative value according to $\mathrm{X2}$) Then x belongs to cluster 1." Similarly, Cluster 2 can be described by the rule "(If a datapoint x has positive value according to the feature $\mathrm{X1}$ AND negative value according to $\mathrm{X2}$), OR if x has a positive value according to $\mathrm{X1}$ AND a negative value according to $\mathrm{X2}$) then x belongs to cluster 2." Hence, we can see that, in order to describe cluster 1 or cluster 2, we need to take into account the interaction between the two features $\mathrm{X1}$ and $\mathrm{X2}$ using the operator 'AND' as we can see in the figure below.

Figure 41: Use of an Interaction Feature to Discriminate Two Different Clusters

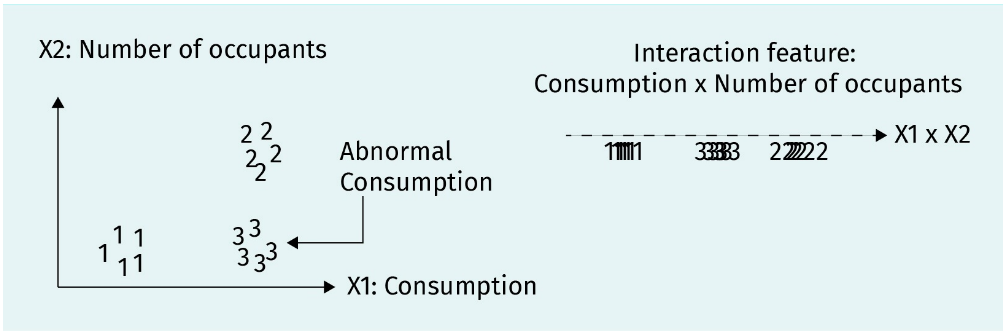


Source: Sayed-Mouchaweh (2021).

Let us take the energy consumption example and build a model that allows us to provide clusters gathering clients of similar consumption. The number of occupants in a home affects the energy consumption of this home. Therefore, there is an interaction between these two features since the impact of the "consumption" on the obtained clusters (clients with similar consumptions) is different at different values of the "number of occupants." Therefore, generating an interaction feature, "consumption" AND "number of occupants," lets the model express what happens when these two features are together in the same row.

The figure below shows three different clusters. Cluster 1 represents the consumption for households of a small number of occupants, while cluster 2 represents the consumption for households of a large number of occupants. Both clusters represent normal consumption. Cluster 3 represents an example of abnormal consumption since the households belonging to this cluster consume a lot with a small number of occupants. Identifying the clients belonging to this cluster can alert them that they need to repai their electrical appliances, such as a washing machine or stove, or to mind their consumption. We can observe from the figure on the left below that neither feature 1 (consumption) nor feature 2 (number of occupants) alone can discriminate cluster 3. Taking into account their interaction (figure in the right below) enables the discrimination of cluster 3.

Figure 42: Interest of the Use of Interaction Features to Discriminate The Abnormal Consumption Cluster



Source: Sayed-Mouchaweh (2021).

The same is true for when we want to observe whether the energy consumption differs with respect to the work type. If the obtained clusters gather clients of similar energy consumption but with different work types, then there is no impact of the work type on the energy consumption.

The table below shows the features “number of occupants” and “consumption” and their cross product feature “number of occupants x consumption.” We can see that the cross product feature allows a model to learn that the consumption depends on the number of occupants. For instance, the second row in the table below has the highest consumption because the number of occupants is the largest, while the fourth row has the smallest consumption because the corresponding flat has the smallest number of occupants. This useful information cannot be learned by the individual features “number of occupants” nor “consumption.”

Table 14: Motivation to Use Cross Product Features for Energy Consumption Example

2	70	140
4	**140**	**560**
2	65	130
1	40	40
2	65	130

Source: Sayed-Mouchaweh (2021).

Generating interaction features can be achieved as all polynomial combinations of the features with degree less than or equal to the polynomial predefined degree. For instance, if we have two features x_{F1} and x_{F2}, a polynomial of degree 2 allows us to generate an interaction feature between them: $x_{F1}x_{F2}$.

Interaction features can be generated using the polynomial expansion function PolynomialFeatures in the Python library sklearn. Let us generate the interaction feature between the features “number occupants” and “consumption” of the energy consumption example. The steps are as follows:

1. Import the function PolynomialFeatures of Python library sklearn.

   Code
   ```
   from sklearn.preprocessing import PolynomialFeatures
   ```

2. Create a polynomial combination of feature “number-occupants” and “consumption” in order to have interaction features indicating consumption of a certain number of occupants, and then display the set of old and new features together.

   Code
   ```
   # create and fit a polynomial feature creator
   pf = PolynomialFeatures(\
      degree = 2, \
      interaction_only=True, \
      include_bias = False).\
          fit(TDF[['Number-occupants', 'Consumption']])

   # apply the polynomial feature creator to the data
   int_feat = pf.transform(TDF[['Number-occupants', \
      'Consumption']])

   print(int_feat)
   # console output:
   # [[ 2. 70. 140.]
   # [ 4. 140. 560.]
   # [ 2. 65. 130.]
   # [ 1. 40. 40.]
   # [ 2. 65. 130.]]

   # convert the generated interaction feature array
   # to a dataframe
   int_feat = pd.DataFrame(int_feat, \
      columns=['Number-occupants', 'Consumption', \
          'nOcc_x_Conspt'])

   # append generated interaction feature to dataframe
   TDF = pd.concat([TDF, int_feat], axis=1)
   ```

4.2 Categorical Features

There are four kinds of feature scales. We have already seen the interval and ratio scales, and now the missing two scale levels come up here. As a reminder, interval scales represent ordered elements with a specific interval which have the same difference, while ratio scale has all the characteristics of interval scale and, in addition, it can integrate the value of "zero" to any of its features.

There are two fundamental levels of measurement for categorical features: nominal and ordinal. In the nominal scale, features are named or labeled, but do not have any kind of mathematical order, such as the gender of a customer or the brand of their car. In ordinal scale, the values of features have a meaningful or specific order, such as customer satisfaction (high, medium, and low) and an exam grade (A, B, C, or F). Ordinal variable can exhibit numerical and categorical characters when numbers are used to scale the order of its values, such as the customer satisfaction between 0 (not satisfied at all) and 5 (completely satisfied). However, this should be used with caution as this inhibits the danger of using statistical methods, which are not meant for ordinal data.

Let us take the example of energy consumption. The energy consumption table includes two nominal features (gender and work type), one ordinal feature described by integers (client satisfaction), one discrete numeric feature (number of occupants), and one continuous numeric feature (consumption). The nominal feature "work type" is represented by integers, where 1 refers to "self-employed," 2 to "state employee," and 3 to "private." The ordinal categorical feature "client satisfaction" is also represented by integers from 0, "completely unsatisfied," to 5, "completely satisfied." This table cannot be presented as input data to a machine learning approach; the categorical features need to be converted into numerical features. This can be done by using feature encoding techniques. In addition, useful new features can be generated to improve the predictive power or the clusters quality of the built models by machine learning approaches.

Feature Cleaning (Imputing Missing Values)

Categorical features may have missing values for some rows. In this case, missing values can be replaced with the maximum occurred value in the corresponding column (feature). Let us take the energy consumption dataset and suppose that there is one missing value in the "gender" feature indicated by NA (not available), illustrated in the table below.

Table 15: Imputing a Missing Value in the Gender Feature of the Energy Consumption Example

C-ID	Gender before imputing	Gender after imputing by replacing by the maximum occurred value
1	M	M
2	NA	M
3	M	M

4	F	F
5	F	F

Source: Sayed-Mouchaweh (2021).

To impute using Python, we first need to check if missing values exist in the dataset, and, if this is the case, determine the features that have missing values and the number of missing values. Then, we replace the missing values in each feature by the maximum occurred value as follows.

1. Create the energy consumption dataset with a missing value in the “gender” feature.

   Code
   ```
   # import libraries
   import pandas as pd
   from sklearn.preprocessing import PolynomialFeatures
   from sklearn.preprocessing import LabelEncoder

   # generate sample data
   Table = { \
      'Customer-ID': [1, 2, 3, 4, 5], \
      'Gender':['M', pd.NA, 'M', 'F', 'F'], \
      'Work-type': [1, 2, 2, 3, 3], \
      'Client-satisfaction':[3, 0, 4, 3, 5], \
      'Number-occupants': [2, 4, 2, 1, 2], \
      'Consumption': [70, 140, 65, 40, 65]}
   TDF = pd.DataFrame(data=Table)
   ```

2. Check for missing values using the Python function `isnull().sum()`. This function provides the number of missing values (null) for each feature (column) of the dataset. For our example, all features have zero, excluding “gender,” which will have one missing value, NA.

   Code
   ```
   print(TDF.isnull().sum())
   # console output:
   # Customer-ID           0
   # Gender                1
   # Work-type             0
   # Client-satisfaction   0
   # Number-occupants      0
   # Consumption           0
   ```

3. Fill the missing values using the function `fillna` by the maximum occurred value. The latter can be found using the function `value_counts()`. Then, check that there are no more missing values using the Python function `isnull().sum()`.

Code
```
TDF = TDF.fillna(TDF['Gender'].value_counts().index[0])
print(TDF.isnull().sum())
```

Feature Encoding

One technique used to convert categorical features into numerical features is label encoding. This alphabetically converts each category in a column to a number between 0 and the number of categories —1. For this example, the “gender” category will take one of two variables M and F, which will be replaced by 1 and 0, respectively. It can be implemented using the scikit-learn function `LabelEncoder` as follows.

Code
```
# encode the categories in the column
# 'Gender' by numbers
TDF['Gender'] =
LabelEncoder().fit_transform(TDF['Gender'])

# show the resulting column
TDF['Gender']
# console output:
# 0   1
# 1   0
# 2   1
# 3   0
# 4   0
```

Although label encoding is straightforward to use and implement, the converted numerical values can be misinterpreted by a machine learning method since, for example, 1 for “M” does not mean it has more weight than “F,” which has the value 0.

An alternative to the misinterpretation of the numerical values provided by label encoding is one-hot encoding. This creates one new feature for each potential value of a categorical feature by replacing it with 1 or zero. In our example, the “gender” feature will be replaced by two one-hot encoding features: “M-gender” and “F-gender.” For “M-gender,” all the male clients will have the value 1, while female clients have the value 0. For the “F-gender,” all female clients will be assigned the value 1, while the male clients will be assigned 0. Similarly, the feature “work type” will be replaced by three one-hot encoding features: work type 1, work type 2, and work type 3, as illustrated in the table below.

Table 16: One-Hot Encoding for Gender and Work-types Features of a Table

Gender-M	Gender-F	Work-type-1	Work-type-2	Work-type-3
1	0	1	0	0
0	1	0	1	0

1	0	0	1	0
0	1	0	0	1
0	1	0	0	1

Source: Sayed-Mouchaweh (2021).

In the following, we apply the one-hot encoding using Python. It is very useful to show the type (integer (int), real (float), string (object), mixed numeric and non-numeric (object)) of each variable in the table. The code below shows this type as follows: Customer-ID int64, Gender object, Work-type int64, Client-satisfaction int64, Number-occupants int64, Consumption int64.

Code
```
print('The type of each column is: \n', TDF.dtypes)
```

1. We can see that both "work-type" and "client-satisfaction" are considered integers. To create one-hot encoding using Python for these two features, they must be converted into string:

   Code
   ```
   TDF['Work-type']=TDF['Work-type'].astype(str)
   TDF['Client-satisfaction']= TDF[
                         'Client-satisfaction'].astype(str)
   print('The type of each column is: \n', TDF.dtypes)
   ```

2. We now apply the Python function `pd.get_dummies` in order to create the one-hot encoding for the three categorical features of "gender," "work type," and "number occupants."

3. **Code**
   ```
   # one-hot-encode categorical features
   TDF_1hot = pd.get_dummies(TDF)

   # print the column names of the resulting
   # dataframe
   print(list(TDF_1hot.columns))

   # console output:
   # ['Customer-ID',
   # 'Number-occupants',
   # 'Consumption',
   # 'Gender_F',
   # 'Gender_M',
   # 'Work-type_1',
   # 'Work-type_2',
   # 'Work-type_3',
   ```

```
    # 'Client-satisfaction_0',
    # 'Client-satisfaction_3',
    # 'Client-satisfaction_4',
    # 'Client-satisfaction_5']
```

One-hot encoding has the advantage of avoiding the unequal weights assigned to a feature's categories. However, it quickly becomes useless when the features have many categories. Therefore, one-hot encoding may dramatically increase the number of features entailed to generate overfitting issues. In addition, it is not guaranteed that these added features improve the performances (quality of obtained clusters or predictive power) of the built model by machine learning approaches.

Feature Generation

There are many techniques used for generating new features. We will present the following frequently used techniques: feature splitting, feature grouping, extracting dates, and cross product features (interaction features).

Feature splitting or spreading

Some features in datasets may include more than one piece of information per cell defined by a certain row in a certain feature or column. This means that these cells cannot be treated by machine learning approaches. Rows of these features may contain grouped information, such as names and dates in the same row, leading to their misinterpretation. In addition, some parts of the grouped information may not be useful for machine learning approaches. Hence, splitting, or spreading, features in order to extract useful features from raw and irrelevant features may improve the performances of machine learning approaches. For instance, if a feature contains the value "France-Paris," this value contains combined information about a country and a city. It cannot be treated (interpreted) by machine learning approaches in this combined form. It is useful to split it into two useful features: the country "France" and the city "Paris." This splitting, or spreading, enables the generation of a **tidy dataset**.

Tidy dataset
A tidy dataset is represented by tables of the following characteristics: 1) each feature forms a column and contains values, 2) each data point forms a row, and 3) each type of observational unit forms a table.

Let us take the example of a feature representing the names of clients illustrated in the table below. We see that each name (row) contains a first name, one or two last names, and a date. We can split this feature into three features: one representing the first name, the second representing the last name, and the third representing the year, as illustrated in the table below.

Table 17: Splitting a Categorical Feature into Three Different Features

Names	FirstName	LastName	Year
Joe B. BARBY 12-05-2019	Joe	BARBY	2019
Juliette KARB 18-08-2018	Juliette	KARB	2018

Lucien VAN 05-07-2017	Lucien	VAN	2017
Danielle G. REB 03-09-2020	Danielle	REB	2020
Lydia HAM 09-07-2018	Lydie	HAM	2018

Source: Sayed-Mouchaweh (2021).

We can apply the feature splitting using Python as follows:

1. Create the dataset with the combined feature to be split “Names” as a DataFrame.

   Code
   ```
   # import libraries
   import pandas as pd
   import numpy as np

   # generate sample data
   Table = { \
      'Customer-ID': [1, 2, 3, 4, 5], \
      'Names': [ \
          'Joe B. BARBY 12-05-2019', \
          'Juliette KARB 18-08-2018', \
          'Lucien VAN 05-07-2017', \
          'Danielle G. REB 03-09-2020', \
          'Lydia HAM 09-07-2018'], \
      'Gender': ['M', 'F', 'M', 'F', 'F'], \
      'Work-type': [1, 2, 2, 3, 3], \
      'Client-satisfaction': [3, 0, 4, 3, 5], \
      'Number-occupants': [2, 4, 2, 1, 2], \
      'Consumption': [70, 140, 65, 40, 65]}
   TDF = pd.DataFrame(data=Table)
   ```

2. Split the feature “names” into three features: “FNF” for the first name, “LNF” for the last name, and “YF” for the year. To achieve this, we need to use the Python functions `split` and `map`. `split` allows us to split a long string according to certain defined separator. For our example, the separator is a space for the first name and last name, and “-” for the date. `map` allows us to map a certain element (first, before last, etc.) of the `split` string. For instance, applying split function with the separator “ ” on “Joe B. BARBY 12-05-2019” generates four different elements: “Joe,” “B.,” “BARBY,” and “12-05-2019.” Applying `map` to select the first element (x[0]) of the split string generates “Joe,” while the before last element (x[-2]) generates “BARBY.”

   Code
   ```
   # split information in one column into three
   TDF['First-Name'] = TDF.Names.str.split(" ").\
      map(lambda x: x[0])
   ```

```
TDF['Second-Name'] = TDF.Names.str.split(" ").\
   map(lambda x: x[-2])
TDF['Birth-Year'] = TDF.Names.\
   str.split("-", n=2, expand=True)[2]
```

Feature grouping

Machine learning approaches work with tidy datasets represented by tables. Each row of the table represents an observation or data point and each column indicates a feature value of this data point. However, datasets may not be tidy. Multiple rows may represent the same data point. As demonstrated in the table below, the energy consumption of the same customer is represented in different rows. Therefore, it is important to convert an untidy dataset to a tidy dataset. This can be achieved using group-by operations through deciding the aggregation function (sum, mean, max, and min, etc.).

Table 18: Untidy Dataset Where the Same Customer Has Different Consumption (Rows)

Customer-ID	Sum-Consumption
1	70
2	50
3	65
2	57
3	69
1	73

Source: Sayed-Mouchaweh (2021).

For the example in the table below, the energy consumptions of customers are grouped by summing them.

Table 19: Grouping the Consumption of the Same Customer Using the Aggregation Operator Sum

Customer-ID	Sum-Consumption
1	143
2	107
3	134

Source: Sayed-Mouchaweh (2021).

Grouping can be applied using Python as follows:

1. Create two tables: The first table is a tidy dataset (dat_tidy), and the second is non-tidy dataset (data_non_tidy). Convert them into DataFrame variables.

   Code
   ```
   dat_tidy = pd.DataFrame({'Customer-ID': [1, 2, 3], \
      'Gender': ['M', 'F', 'M']})
   dat_non_tidy = pd.DataFrame(\
      {'Customer-ID': [1, 2, 3, 2, 3, 1],
       'Consumption': [70, 50, 65, 57, 69, 73]})
   ```

2. Group the consumptions of the same customer based on their ID (customer-ID) using the aggregation function sum, then merge the aggregated (dat_tidy) column (sum-consumption) to the tidy table (TDF) and display the obtained table.

   Code
   ```
   # aggregate consumption per customer
   dat = dat_non_tidy.groupby('Customer-ID')['Consumption'].\
      agg(['sum'])

   # rename column
   dat.columns = ['Sum-Consumption']

   # join tables
   dat = dat_tidy.merge(dat, left_on='Customer-ID', \
      right_index=True, how='left')

   # show the resulting table
   dat
   # console output:

   print(dat)
   #  Customer-ID Gender Sum-Consumption
   # 0 1          M      143
   # 1 2          F      107
   # 2 3          M      134
   ```

Extracting dates

Date columns carry interesting information that machine learning approaches ignore because they combine year, month, and day in the same cell. Therefore, it is useful to extract dates to improve the performance of machine learning methods using one of the following:

- extracting the year, month, or day of the date
- extracting the elapsed time between the current date and the date in the column
- extracting whether the day is a weekday, a weekend, or a holiday

Let us apply these three methods using Python:

1. Import `date` from the Python library `datetime`, and create an example of a dataset containing a date column as a DataFrame.

   Code
   ```
   # import libraries
   import pandas as pd
   from datetime import date

   # generate sample data
   Table = {'Customer-ID': [1, 2, 3, 4, 5], \
      'Date': ['12-05-2019', '18-08-2018', \
          '05-07-2017', '03-09-2020', '09-07-2018'], \
      'Consumption': [70, 140, 65, 40, 65]}
   TDF = pd.DataFrame(data=Table)
   ```

2. Extract information about the date from the date column.

   Code
   ```
   # convert 'Date'-column to date format
   TDF['Date'] = pd.to_datetime(TDF.Date, format="%d-%m-%Y")

   # extract the year
   TDF['year'] = TDF['Date'].dt.year

   # extract the month
   TDF['month'] = TDF['Date'].dt.month

   # extract quarter of the year
   TDF['quarter'] = TDF['Date'].dt.quarter
   ```

3. Extract the past years and months from the date in the date column to today.

   Code
   ```
   # extract passed years
   years_diff = date.today().year - TDF['Date'].dt.year
   TDF['passed_years'] = years_diff

   # extract passed months
   months_diff = (date.today().year - \
      TDF['Date'].dt.year) * 12 + \
          date.today().month - \
              TDF['Date'].dt.month
   TDF['passed_months'] = months_diff
   ```

4. Extract the name of the day of the date and then display the obtained extracted dates.

Code
```
TDF['day_name'] = TDF['Date'].dt.day_name()
```

5. We can also indicate whether the extracted day is a weekend by using the function “dayofweek.” This function returns the values from 0 to 4 for the days Monday to Friday, 5 for Saturday, and 6 for Sunday. Running the following code generates the following results in the table below:

   Code
```
                            # extract the day of the week
TDF['dow'] = pd.to_datetime(TDF['Date']).dt.dayofweek

# extract weekends
TDF['weekend'] = TDF['dow'].\
 map(lambda x: 0 if x < 5 else 1)
```

Table 20: Extracting Whether the Day is a Weekend Day or a Business Day

day-name	dow	weekend
Sunday	6	1
Saturday	5	1
Wednesday	2	0
Thursday	3	0
Monday	0	0

Source: Sayed-Mouchaweh (2021).

Cross products features (interaction features)

We have already seen how to build interaction features between numeric features. Let us now see how to build interaction features between categorical features. Sometimes certain features grouped together can generate useful information that those features cannot provide when they are used individually by machine learning approaches. This is because gathered features may denote an interesting property that allows the improvement of the predictive power or the quality of obtained clusters of the machine learning approaches. For instance, if a medical treatment (yes or no) or obtaining a science grant (yes or no) depends on the gender of a candidate, then there is an interaction between these two features, and it is useful to combine them. This combination allows us to obtain four categories: (male, yes), (male, no), (female, yes), and (female, no). In other words, the combination of two interacted features allows to take into account the effect that one of them has on the level (response) of the other feature.

Let us generate interaction features between categorical features using Python. The table below shows an example of the gender of scientists who received (or did not receive) a science grant. We would like to know whether the gender affects the status of obtaining a science grant.

Table 21: Example Showing Whether the Gender Affects the Status of Receiving a Grant

Gender	Scientific-Grant
M	Y
F	N
M	Y
F	N
F	Y
M	N
F	N
M	N

Source: Sayed-Mouchaweh (2021).

The table below shows the one-hot encoding of the table above.

Table 22: One-Hot-Encoding for Gender and "Work-types" Features of Table

Gender-M	Gender-F	Work-type-1	Work-type-2	Work-type-3
1	0	1	0	0
0	1	0	1	0
1	0	0	1	0
0	1	0	0	1
0	1	0	0	1

Source: Sayed-Mouchaweh (2021).

We can observe that the mean value of not obtaining a grant largely increases when the gender is "female." Therefore, there is an interaction between the gender and the status of obtaining a scientific grant. Therefore, we will run this example using Python as follows:

1. Generate the dataset of energy consumption as a `DataFrame`.

Code

```
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
Table = {'Gender': ['M','F','M','F','F','M',’F’,’M’],
        'Science-Grant': ['Y', 'N', 'Y', 'N',
                          'Y','N',’N’,’N’]}
TDF = pd.DataFrame(data=Table)
```

2. Apply the Python function `pd.get_dummies` to create the OneHotEncoding for the categorical features “gender” and “science-grant.”

Code

```
TDF = pd.get_dummies(TDF)
```

3. Generate the interaction feature between the two categorical features “Work-type-Gender_F” and “Science-Grant_NClient-satisfaction” using Python function PolynomialFeatures. Since we are interested only in the interaction between features, i.e., “Gender_F” x “Science-Grant_N,” the parameter “interaction only” is selected to be “True.” This allows us to avoid generating the features “Gender_F” x “Gender_F” and “Science-Grant_N” x “Science-Grant_N.” The parameter “include bias” is deactivated by assigning the “False” value in order to avoid generating an intercept feature (the feature in which all polynomial powers are zero, i.e., a column of ones). Calculate the mean value of female scientists who did not obtain the grant.

Code

```
# generate interaction features
pf = PolynomialFeatures(degree=2, \
   interaction_only=True, include_bias=False).\
       fit(TDF[['Gender_F','Science-Grant_N']])
int_feat = pf.transform(TDF[['Gender_F',\
'Science-Grant_N']])

print(int_feat)
# console output:
# [[0. 0. 0.]
# [1. 1. 1.]
# [0. 0. 0.]
# [1. 1. 1.]
# [1. 0. 0.]
# [0. 1. 0.]
# [1. 1. 1.]
# [0. 1. 0.]]

# convert the generated interaction feature array
# to a dataframe
TDF = pd.DataFrame(int_feat, \
   columns=['Gender_F','Science-Grant_N', 'FxGrant'])
```

```
# Calculate the mean value of female scientists
# who did not obtain the grant
print(TDF['FxGrant'].mean(0))
# console output:
# 0.375
```

4. Generate the interaction feature between the two categorical features “Gender_M” and “Science-grant_N” using the Python function `PolynomialFeatures`. Calculate the mean value of male scientists who did not receive the grant.

   Code
   ```
   # generate interaction features (male x grant)
   pf = PolynomialFeatures(degree=2, \
      interaction_only=True, include_bias=False).\
          fit(TDF[['Gender_M','Science-Grant_N']])
   int_feat = pf.transform(TDF[['Gender_M', \
      'Science-Grant_N']])

   # convert the generated interaction feature array
   # to a dataframe
   male_x_grant = pd.DataFrame(int_feat, \
      columns=['Gender_M','Science-Grant_N', 'MxGrant'])

   # Calculate the mean value of female scientists
   # who did not obtain the grant
   print(male_x_grant['MxGrant'].mean(0))
   # console output:
   # 0.25
   ```

4.3 Text Features

A text, composed by a set of sentences, such as “Martin broke the cup,” contains a lot of information. To extract the information contained in a text, salient features need to be defined, e.g., “Martin,” “broke,” “cup.” We can do this by parsing the text and removing certain useless words. This process is called tokenization. These words need to be encoded as integers or floating-point values in order to be used as input by machine learning methods. This process is called feature extraction (or vectorization). In the next section, we will see some of the frequently used text feature tokenization and vectorization to extract information from texts.

Bag-of-Words (BoW)

Bag-of-Words (BoW), also called tokenizer or naïve scoring, represents the statistical count of the words contained in a text. For instance, in the sentence “I will call you if I have time,” the word “I” has a count of two, while all the other words have a count of one. BoW represents the occurrence number of a word in a sentence. It is the simplest text feature. It is used in document retrieval since the presence, absence, or the number of times certain words are mentioned are efficient indicators of a document's topic content.

Each unique word in a text or document represents one dimension or feature; therefore a document of n unique words has n dimensions and its BoW is a flat vector of n dimensions. However, BoW does not conserve the order of the words or their hierarchy. Therefore, BoW cannot conserve the text semantic (never mind the context and meaning of a text). For instance, dog toy and toy dog have different semantics while sharing the same BoW. Therefore, the word ordering is one—though not the only—important factor in conserving the sentence meaning. In addition, the link or hierarchy between words, such as dog, cat, and animal, is not conserved by BoW. It is worth mentioning that stop words, such as "a," "an," "the," "they," "where," "etc.," punctuation, white spaces, and so on, must be removed from the text since they deliver information which is not relevant to the analysis.

Let us carry out BoW using Python, with the following example text: "Martin is not bad person. Kevin, Martin's brother, is bad person." Let us compute the BoW for this text. The text feature is ['bad', 'brother', 'is', 'Kevin', 'Martin', 'not', 'person'] and its `BoW = [[1 0 1 0 1 1 1] [1 1 1 1 1 0 1]]`. The steps are demonstrated below:

1. Import `CountVectorizer` from Python module sklearn.feature_extraction.text.

   Code
   ```
   from sklearn.feature_extraction.text
        import CountVectorizer
   ```

2. Create the following text, compute its BoW, and then display the text feature and its BoW. The parameter "lowercase" when it is set to "True" converts all words to lowercase before tokenizing. In order to separate uppercase and lowercase words, such as "Martin" and "martin," "lowercase" is set to "False."' CountVectorizer can enable the text preprocessing before generating the vector representation. Scikit-learn has a built-in list of English stop words in the feature_extraction.text module. This list can be used by CountVectorizer to remove the stop words. The code below uses this list allowing to obtain the text feature ['Kevin', 'Martin', 'bad', 'brother', 'person'] and its `BoW = [[0 1 1 0 1] [1 1 1 1 1]]`.

   Code
   ```
   # generate sample data
   corpus = ['Martin is not a bad person.',
             'Kevin, is the brother of Martin.',
             'Kevin is a bad person.']

   # create tokenizer
   vectorizer = CountVectorizer(lowercase = False,
                                stop_words='english')
   ```

```
    # fit tokenizer
    BoW = vectorizer.fit_transform(corpus)

    # print feature names
    print(vectorizer.get_feature_names())
    # console output:
    # ['Kevin', 'Martin', 'bad', 'brother', 'person']

    # print the number of occurrences of each
    # feature in each text element
    print(BoW.toarray())

    # console output:
    # [[0 1 1 0 1]
    # [1 1 0 1 0]
    # [1 0 1 0 1]]
```

Bag-of-N-Grams

In order to avoid destroying the sentence meaning by breaking it down into individual words, Bag-of-n-Grams is used. It is a sequence of n successive tokens or words. For instance, one word is 1-gram or unigram, two successive words is 2-grams or bigram, and so on. Let us take the text: “Martin is not bad person. Kevin, Martin’s brother, is bad person.” We can create its Bag-of-2-Grams feature text ['Kevin Martin', 'Martin brother', 'Martin is', 'bad person', 'brother is', 'is bad', 'is not', 'not bad'] and its BoW vector [[0 0 1 1 0 0 1 1] [1 1 0 1 1 1 0 0]]. It is worth mentioning that punctuation is treated as a token (word) separator by the function `CountVectorizer`. In addition, the parameter $(\mathrm{n},\ \mathrm{n})$ indicates the lower and upper n-values for word n-grams to be extracted. For instance, $(1,\ 2)$ will extract the unigrams and bigrams of the text while $(2,\ 2)$ will extract only bigrams of the text.

Code
```
# create bag Bag-of-n-Grams tokenizer
vectorizer2 = CountVectorizer(analyzer='word', \
   ngram_range=(2, 2), lowercase=False,
   stop_words='english')

# fit the vectorizer
Bo2G = vectorizer2.fit_transform(corpus)

# print feature names
print(vectorizer2.get_feature_names())
# console output:
# ['Kevin bad', 'Kevin brother', 'Martin bad',
# 'bad person', 'brother Martin']

# print the number of occurrences of each
# feature in each text element
print(Bo2G.toarray())
# console output (extract):
# [[0 0 1 1 0]
# [0 1 0 0 1]
# [1 0 0 1 0]]
```

The n-grams contain more information and better conserve the text semantic. The higher the number of successive words to consider n, the better the conserved information or semantic. However, the computation cost and required memory space become much greater. For a feature text of p individual words, there are p^2 bigrams.

Term Frequency-Inverse Document Frequency (TF-IDF)

Counts of words cannot provide rich or refined information about the document or text content. They cannot capture the meaning of a text, its context, nor, most importantly, its magic. One technique to approaching the context of many documents (on a quantitative level) is to have useful indicators about the content of a collection of documents, which is the Term Frequency-Inverse Document Frequency (TF-IDF). It weights the words by their occurrence frequency in the text. This allows us to detect some meaningful words such as “excellent,” “magnificent,” or “bad.” In addition, TF-IDF considers the words within the whole collection of documents. Therefore, it scales down the frequent words while scaling up the rare words. Hence, a word that appears 100 times will not be 100 times more important than a word appearing only once.

TF-IDF is a normalized count of words in a set of documents. Let $\mathrm{BoW(w{,}d)}$ be the number of times that the word w appeared in a document d; it indicates the frequency term (TF) of word w in document d. Inverse Document Frequency (IDF) is the number of documents N in the dataset divided by the number $\mathrm{n_w}$ that the word w occurred in the documents contained in the dataset. Then, TF-IDF can be obtained by multiplying TF and IDF:

$$TF - IDF = BoW(w,d)\,.\,\frac{N}{n_w} = TF\,.\,IDF$$

Let us take the following simple example of two documents:

1. The first document $\mathrm{d_1}$ is “Martin is good person”
2. The second document $\mathrm{d_2}$ is “Kevin is bad person”

The feature vector for this example is ['Kevin', 'Martin', 'bad', 'good', 'is', 'person']. The term frequencies are as follows:

$\mathrm{TF\ (Kevin{,}d_1)\ {=}0,\ TF\ (Martin{,}d_1)\ {=}1,\ TF\ (bad{,}d_1)\ {=}0,\ TF\ (good{,}d_1)\ {=}1,\ TF\ (person{,}d_1)\ {=}1\ TF\ (Kevin{,}d_2)\ {=}1,\ TF\ (Martin{,}d_2)\ {=}0,\ TF\ (bad{,}d_2)\ {=}1,\ TF\ (good{,}d_2)\ {=}0,\ TF\ (person{,}d_2)\ {=}1}$

The Inverse Document Frequencies for $\mathrm{d_1}$ and $\mathrm{d_2}$ are

$\mathrm{IDF(Kevin,d_1)} = (\mathrm{N}=2/\mathrm{n_w}=1)=2$, $\mathrm{IDF(Martin,d_1)} = (2/1)=2$, $\mathrm{IDF(bad,d1)} = (2/1)=2$, $\mathrm{IDF(good,d_1)} = (2/1)=2$, $\mathrm{IDF(person,d_1)} = (2/2)=1$, $\mathrm{IDF(Kevin,d_2)} = (2/1)=2$, $\mathrm{IDF(Martin,d_2)} = (2/1)=2$, $\mathrm{IDF(bad,d_2)} = (2/1)=2$, $\mathrm{IDF(good,d_2)} = (2/1)=2$, $\mathrm{IDF(person,d_2)} = (2/2)=1$

TF depends on the word in a certain document, while IDF is the same for a certain word in any document, e.g., $\mathrm{IDF(person,d_2)} = \mathrm{IDF(person,d_1)} = 1$.

Often, we take the log of IDF instead of the raw IDF. The logarithm returns 1 when the raw IDF is 0, and makes large values of raw IDF smaller. In addition, in order to avoid dividing by zero when a word does not occur in any document ($\mathrm{n_w} = 0$), 1 is added to the numerator and denominator of the raw IDF as if an extra document were seen containing every word in the feature vector exactly once. Finally, 1 is added to the obtained IDF allowing words with zero IDF, i.e., words that occur in all documents, not to be entirely ignored. Therefore, TF-IDF will be calculated by

$$TF - IDF = TF\,.\,IDF = BoW(w,d)\,.\,\log_2\left(\frac{N+1}{n_w+1} + 1\right)$$

The table below shows FT-IDF using the log formula above, for instance

- $\mathrm{TF(Kevin,d_1)} = 0$, $\mathrm{IDF(Kevin,d_1)} = \log_2((2+1)/(2+1))+1 = 1$, therefore, TF-IDF $= 1$
- $\mathrm{TF(good,d_1)} = 1$, $\mathrm{IDF(good,d_1)} = \log_2((2+1)/(1+1))+1 = 1.4$, TF-IDF $= 1.4$

Table 23: TF-IDF for a Simple Dataset of Two Simple Documents

Word	TF		IDF	TF-IDF	
	d_1 (0)	c		d_1 (0)	d_2 (1)
Kevin (0)	0	1	$\mathrm{Log_2(3/2)+1}$	0	1.4
Martin (1)	1	0	$\mathrm{Log_2(3/2)+1}$	1.4	0
bad (2)	0	1	$\mathrm{Log_2(3/2)+1}$	0	1.4
good (3)	1	0	$\mathrm{Log_2(3/2)+1}$	1.4	0
is (4)	1	1	$\mathrm{Log_2(3/3)+1}$	1	1
person (5)	1	1	$\mathrm{Log_2(3/3)+1}$	1	1

Source: Sayed-Mouchaweh (2021).

We can see from the table above that a word which appears in each single document, such as “is,” has a small TF-IDF, while a word which appears in few documents, such as “good,” has a bigger TF-IDF. Therefore, TF-IDF allows us to distinguish common words, such as “is,” from rare words, such as “good.” Consequently, TF-IDF can be used to remove common words, which are not useful, and select rare or meaningful words. It is worth mentioning that TF-IDF becomes better to be able to detect the frequent words reliably as the number of documents increases.

Let us apply the TF-IDF using Python using the following steps:

1. Import the Python function TfidfVectorizer from Python library `sklearn.feature_extraction` and create the simple dataset of two simple documents.

   Code
   ```
   # import libraries
   from sklearn.feature_extraction.text \
      import TfidfVectorizer

   # generate sample data
   corpus = ['Martin is not a bad person.',
            'Kevin, is the brother of Martin.',
            'Kevin is a bad person.']
   ```

2. Apply the Python function TfidfVectorizer to this dataset, and get the features (words) of this dataset using the Python function `get_feature_names()` and display the result. In the table above, we integrated the index of each word (from 0 to 5) and document (from 0 to 1) since Python displays TF-IDF as a combination of the index of the document and the index of the word. For instance, (0, 1) indicates the TF-IDF for the word “Martin” in the first document d_1. It should be mentioned that the parameter “norm” is deactivated only to make the results comparable with the examples we have seen so far. Thus, the TF-IDF is not scaled.

   Code
   ```
   # create TF-IDF tokenizer without normalization
   TFIDF= TfidfVectorizer(lowercase=False, \
      norm=False, stop_words='english')

   # fit tokenizer
   TFIDFtext = TFIDF.fit_transform(corpus)

   # print feature names
   print(TFIDF.get_feature_names())
   # console output:
   # ['Kevin', 'Martin', 'bad', 'brother', 'person']

   # print the values of each Word
   # (second entry in parenthesis) in each
   # document (first entry in parenthesis)
   print(TFIDFtext)
   # console output:
   ```

```
# (0, 4) 1.2876820724517808
# (0, 2) 1.2876820724517808
# (0, 1) 1.2876820724517808
# (1, 3) 1.6931471805599454
# (1, 0) 1.2876820724517808
# (1, 1) 1.2876820724517808
# (2, 0) 1.2876820724517808
# (2, 4) 1.2876820724517808
# (2, 2) 1.2876820724517808
```

The computed TF-IDF is not scaled or normalized. This means that TF-IDF will depend on the length of each document since a certain word may appear much more frequently in a longer document. Since the length of a document does not contribute to its meaning, it is very useful to scale the TF-IDF to obtain unbiased TF-IDF with respect to the document length. There are several norms used to normalize TF-IDF. One of the most used norms is $l2$ norm. It is simply based on dividing the TF-IDF of each word in a document by the square root of the sum of the squared TF-IDF of all the words in each document:

$$(TF-IDF(w_i,d))_{l2} = TF-IDF\left(w_i,d\right) / \sqrt{\sum_{w_k \in d}(TF-IDF(w_k,d))^2}$$

This can be done in Python by keeping the default value, which is “True,” of the parameter “norm” in the Python function TfidfVectorizer by not specifying it:

Code
```
TFIDF= TfidfVectorizer(lowercase = False)
```

The table below shows the scaled TF-IDF using $l2$ norm. The sum of the normalized TF-IDF of all the features in each document is equal to one. Therefore, if a word is repeated several times in a document, then the normalization reduces the generated bias because of the document length. In the table below, the words “Martin” and “good,” respectively “Kevin and ‘bad,” are identified as important words in d1, respectively d2, while “is” and “person” are identified as less important in both documents. This is logic since the words “Martin,” “good,” “Kevin, and “bad” contribute more to the context and meaning of each document.

Table 24: Normalized TF-IDF According to $l2$ Norm for a Simple Dataset of Two Simple Documents

	Kevin (0)	Martin (1)	bad (2)	good (3)	is (4)	person (5)
d1 (0)	0	0.57	0	0.57	0.40	0.40
d2 (1)	0.57	0	0.57	0	0.40	0.40

Source: Sayed-Mouchaweh (2021).

SUMMARY

This unit explained the techniques used to clean, scale, encode, or transform numerical, categorical, and text features in order to extract and generate features that can be used as input by machine learning algorithms. The goal is to maximize the prediction power and the quality of obtained clusters of machine learning approaches.

The unit discussed the advantages and drawbacks of the different techniques used to preprocess, transform, and generate new features with respect to the facility and simplicity of their treatment or interpretation by machine learning algorithms. The goal is to highlight the suitable conditions for their use and the complementarity between them. For instance, making a skewed distribution as normal as possible allows us to meet this assumption (normal distribution) required by most statistical models, or making a nonlinear relationship between data samples linear. This allows us to simplify the treatment of the dataset by using machine learning approaches. Another example is the interaction features that allow us to take into account the effect that one of them has on the level (response) of the other feature.

UNIT 5

FEATURE SELECTION

STUDY GOALS

On completion of this unit, you will be able to …

- describe the different techniques used to select relevant features.
- explain how to rank features according to certain relevant evaluation criteria.
- select the best features in order to maximize the performances and avoid overfitting of the learned model.

5. FEATURE SELECTION

Introduction

Feature selection is the process of choosing relevant features from the original features in order to optimize an objective function during the model construction. The objective function is defined in such a way that the separation of clusters (for unsupervised machine learning), or the discrimination between classes (in supervised machine learning) is maximized. Therefore, the selected features must capture most of the useful information. To this end, different evaluation criteria are used to determine the information carried out by each of the original features as well as the combinations of them. It is worth mentioning that dimensionality reduction techniques, such as Principal Component Analysis (PCA), differ from feature selection techniques in the fact that the latter transform original features, and not selecting a subset of them, and it requires all original features to perform this feature transformation.

Feature selection becomes necessary for datasets described in high-dimensional feature space. This is because the performances (prediction power for supervised machine learning, quality of obtained clusters for unsupervised machine learning, learning speed, **generalization power**, processing time, etc.) of the learned model and the interpretation power of its output decrease when the number of used features becomes very large. When a model does not generalize well to unseen data, this is known as an overfitting problem, which happens when the number of features is very large, in particular for inherent sparse datasets. Indeed, irrelevant input features may behave as noises, resulting in worse prediction by the learned model. In addition, since clustering approaches consider all features equally important, they may not behave well in the case of a high dimensional feature space. Finally, reducing the number of input features helps to reduce the size of dataset, allowing better data storage, collection, and time processing.

Generalization power
A generalization power of a model represents its capacity to predict unseen data points that were not used to build the model.

Feature selection can be performed manually or with algorithms. When the number of features is small (less than a dozen features), and if you have enough domain knowledge or it is obvious to decide if features are relevant, then you can manually assess the usefulness of each feature to drop the irrelevant or useless ones. For instance, if you want to build a model to predict the weight of a person, features such as eye color or favorite movies are useless since they do not impact weight. Features such as height or proclivity to exercise might be useful. However, manual feature selection is time-consuming, and relevant features are hard to discover, particularly when specialized domain knowledge is required. Therefore, when the number of features is large (hundreds of features) and your domain knowledge is limited, feature selection with algorithms for automation is more efficient and better adapted.

Feature selection algorithms can broadly be divided into two categories: filter and wrapper models. In the filter methods, features are selected by studying their characteristics using some statistical evaluation criteria, such as their variance or correlation. They have the advantage that the features are independently selected before the training of the model starts. Indeed, the features are selected without the need for a model (e.g., a machine learning method), therefore the selected features can be used to train any model. The wrapper methods evaluate the features' relevance using a certain learning approach. In other words, the features that give the best discriminative power, e.g., prediction accuracy or clusters' quality, using a specific learning approach, will be selected. Consequently, when applying wrapper methods for feature selection, the training of the model must be conducted repeatedly. In both categories, the subset of selected features are identified either by an index or by their weight. The index indicates the rank of a feature (e.g., a feature with rank 1 means that it is the best feature with respect to its relevance or importance). The weights indicate the relevance of features: The higher the feature weight, the better its relevance or rank. In both categories, feature selection can be either univariate or multivariate. In univariate feature selection, each feature is independently evaluated, while in multivariate feature selection, each feature is evaluated with respect to other features.

Feature selection can also be divided into Supervised Feature Selection (SFS), when labels (output) of data points are available, and Unsupervised Feature Selection (UFS) when the labels are not available. SFS selects discriminant features that allow separate data points to belong to different classes. UFS is much harder to perform than SFS because defining a feature's relevancy in the absence of output (labels) becomes challenging. It is worth mentioning that SFS is much more developed in the literature than UFS. This is because it is much easier to evaluate the relevance of input features when the response or output variable (labels) is available. The table below summarizes the characteristics of the different feature selection techniques that will be treated in this unit.

Table 25: Overview of the Characteristics of Feature Selection Techniques

	Unsupervised/ Supervised	**Univariate/Multi-ariate**	**Filter/Wrapper**
ANOVA test/Chi-square test	Supervised	Univariate	Filter
Mutual information	Supervised	Univariate	Filter
Feature variance	Unsupervised	Univariate	Filter
Correlation matrix	Unsupervised/Supervised	Uni-variate/Multi-variate	Filter
Permutation feature importance	Supervised	Univariate	Wrapper
Exclusive FS (EFS)	Supervised	Univariate	Wrapper
Sequential Forward FS (SFS)/ Sequential Backward FS (SBS)	Supervised	Multivariate	Wrapper
Recursive Feature Elimination (RFE)	Supervised	Multivariate	Wrapper

Source: Sayed-Mouchaweh (2021).

This unit considers the following questions:

- What are the different techniques used to evaluate the feature relevancy when a dataset is labeled and unlabeled?
- How are features ranked to select the most relevant in univariate and multivariate modes?

5.1 Feature Importance

Features can be ranked according to their relevance to the response variable (output and labels). This can be achieved by assigning a score to features according to their contribution to the prediction of the response. To this end, the correlation between each feature and the response variable is evaluated using some meaningful statistical criteria, such as the Chi-square test for feature selection, the analysis of variance (ANOVA) test, and mutual information. The higher the correlation between an input feature and the response variable, the better the score or the importance of this feature in predicting the response. The feature importance requires the output variable (i.e., response variable or labels) since it is calculated as the correlation between an input feature and the output variable. Therefore, feature importance is primarily a supervised feature selection technique. That being said, the presented techniques can also be used for unsupervised machine learning as the importance of features can be evaluated with respect to their ability, for instance, to separate clusters.

ANOVA Test

ANOVA is a statistical method that compares the variance of one independent feature (input feature) with one dependent feature (output feature) to assess whether they are relevant (i.e., they are from the same population or distribution). It is a hypothesis-testing technique used to determine whether a null hypothesis is accepted or rejected for an alternative.

Let us suppose we have a dataset X containing k input features (columns) and n data points (rows). Let us perform the ANOVA test for X. Let $\mathrm{X_{jm}},\ \mathrm{j} = 1,\ ..,\ \mathrm{k},$ be the mean value of each feature j. Let $\mathrm{X_m}$ be the overall mean of the dataset. We need to calculate the sum of squares between input features (SSB) as follows:

$$SSB = \sum_{j=1}^{k} n \cdot \left(X_{jm} - X_m\right)^2$$

The sum of the squared differences between each data point, $\mathrm{X_{ji}},\ \mathrm{i} = 1,\ ..,\ \mathrm{n},$ according to an input feature, $\mathrm{X_j},\ \mathrm{j} = 1,\ ..,\ \mathrm{k},$ and its corresponding mean, $\mathrm{X_{jm}}$, (SSE) is then calculated as follows:

$$\sum_{j=1}^{k} \sum_{i=1}^{n} \left(X_{ji} - X_{jm}\right)^2$$

SSE is also called the sum of squared errors or residuals. Then, the mean square between input features (MSB) and the mean square of errors (MSE) are calculated as follows:

$$MSB = \frac{SSB}{k-1}, MSE = \frac{SSE}{n-k}$$

$\mathrm{k} - 1$ and $\mathrm{n} - \mathrm{k}$ are called, respectively, the degree of freedom 1, df1, and degree of freedom 2, df2. The F-value used to accept or reject the null hypothesis is then calculated as follows:

$$F = \frac{MSB}{MSE}$$

ANOVA test is a univariate feature technique (Bejani et al., 2014) since it is applied to individual features. It belongs to the filter feature selection category since it is conducted before the actual machine learning takes place. It is used for classification problems (the response variable is categorical) and when the input features are numeric (continuous).

The null hypothesis is that the feature has no relevance to the response (target) variable, while the alternative hypothesis is that the feature has some relevance to the response variable. In order to accept or reject the null hypothesis, p-value probabilities are used. p-values represent the probability or chance that the data points occurred under the null hypothesis. They are decimals from 0 to 1 since they are probabilities. The lower the p-value of a certain feature, the better its chance to have some relevance (i.e., rejecting the null hypothesis for the alternative hypothesis). In order to choose the value of p, we need to choose the confidence level indicating our confidence in rejecting or accepting the null hypothesis. For instance, if we want to be within the confidence interval bigger than or equal to 95 percent and less than 100 percent, we need an **Alpha level** of five percent. The null hypothesis will be rejected if p is smaller than or equal to 0.05. This means that we have strong evidence that the null hypothesis is valid. It is worth mentioning that

Alpha level
The Alpha level can be obtained by subtracting the confidence level from 100. For instance, a confidence level of 95 percent requires an Alpha level of five percent.

- $\mathrm{p} > 5\%$ means “not significant,” shown as “n.s.” in graphics.
- $0.01 < \mathrm{p} <= 5\%$ means “significant,” shown with an asterisk, *, in graphics.
- $0.001 < \mathrm{p} <= 0.01$ means “highly significant,” shown with two asterisks, **, in graphics.
- $\mathrm{p} <= 0.001$ means “very highly significant,” shown with three asterisks, ***, in graphics.

In order to run ANOVA using Python, we need to import two functions from the scikit-learn library: SelectKBest and `f_classif`. SelectKBest is a function that takes two arrays as input: the data points X and target or response variable or column y, and returns a pair of arrays (features scores and p-values) or just the features’ scores array. It return `f_classif` is a function that performs ANOVA test by assigning it a p-value for a classification task. Then, SelectKBest will rank the different features according to their assigned p-value (the lower the better) and will retain the best k features. Hence, k must be predefined by the user.

Let us take the Iris dataset as an example and apply ANOVA to rank the different features in order to select the best k ones. The steps are demonstrated below:

1. Import the required libraries and load the Iris dataset as a DataFrame.

   Code
   ```
   # import libraries
   import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   from sklearn.datasets import load_iris
   from sklearn.feature_selection import SelectKBest
   from sklearn.feature_selection import f_classif

   # load sample data
   iris = load_iris()
   feature_names = load_iris().feature_names
   X = pd.DataFrame(iris.data, columns=iris.feature_names)
   y = iris.target
   ```

2. Choose `f_classif` (F-values in the ANOVA test) as the criterion used in order to select the best $\mathrm{k} = 2$ features.

   Code
   ```
   selector = SelectKBest(score_func=f_classif, k=2)
   ```

3. Train and transform the Iris dataset in order to keep only the selected features. In this example, it should be the best two features according to their test statistics value (F-value for ANOVA) and p-values, and then display the feature names and their assigned p-values. As can be seen from the console output, all features are very highly significant with respect to their importance. Judging by the F-values, “petal length” and “petal width” could be selected as the two most important features in this context.

   Code
   ```
   # fit the selector
   X_new = selector.fit_transform(X, y)

   # print F- and p-values per feature
   pd.DataFrame({'features': feature_names, \
      'Scores': selector.scores_, \
      'p-values': selector.pvalues_})

   # console output:
   #  features          Scores       p-values
   # 0 sepal length (cm)  119.264502    1.669669e-31
   ```

```
# 1 sepal width (cm)   49.160040     4.492017e-17
# 2 petal length (cm)  1180.161182   2.856777e-91
# 3 petal width (cm)   960.007147    4.169446e-85
```

ANOVA ranks the features by estimating the linearity degree between a feature and the response or target variable. Therefore, ANOVA cannot capture nonlinear relationships between input features and the target or output variable.

Chi-Square Test

The Chi-square test allows the evaluation of the independence between two variables. It is calculated based on the difference between the observed O and expected E values for input feature with respect to each category of the response variable as follows:

$$Chi - square = \sum_i \frac{\left(O_i - E_i\right)^2}{E_i}$$

Let us take the table below, called a contingency table, showing the number of patients with respect to their gender (Male/Female) who responded (Yes/No) to medical treatment. The table represents the observed values of each patient gender who responded to the medical treatment (e.g., O(Male/Yes) is the number of "Male" patients who responded ("Yes") to the treatment).

Table 26: Example Showing How a Chi-Square Can Be Calculated

Gender	Responded-Yes	Responded-No	Total
Male	O(Male/Yes)	O(Male/No)	$\mathrm{n_{Male}} = \mathrm{O(Male/Yes)} + \mathrm{O(Male/No)}$
Female	O(Female/Yes)	O(Female/No)	$\mathrm{n_{Female}} = \mathrm{O(Female/Yes)} + \mathrm{O(Female/No)}$
Total	$\mathrm{n_{Yes}} = \mathrm{O(Male/Yes)} + \mathrm{O(Female/Yes)}$	$\mathrm{n_{No}} = \mathrm{O(Male/No)} + \mathrm{O(Female/No)}$	$\mathrm{n} = \mathrm{n_{Male}} + \mathrm{n_{Female}}$

Source: Sayed-Mouchaweh (2021).

If we consider the patient gender (input feature) independent from the treatment response (response variable), then we expect that the probability that the treatment response is "Yes" ("No") when the patient gender is "Male" ("Female") will be the multiplication between the probability that the patient gender is "Male" ("Female") and the probability that the treatment response is "Yes" ("No"). For instance, the expected value for the number of "Male" patients that responded "Yes" to the medical treatment is calculated as follows:

$$E(Male|Yes) = n \cdot p(Male) \cdot p(Yes) = n \cdot \frac{n_{Male}}{n} \cdot \frac{n_{Yes}}{n}$$

Similarly, the other expected values can be calculated as follows:

$$E(Male|No) = n \cdot p(Male) \cdot p(No) = n \cdot \frac{n_{Male}}{n} \cdot \frac{n_{No}}{n}$$

$$E(Female|Yes) = n \cdot p(Female) \cdot p(Yes) = n \cdot \frac{n_{Female}}{n} \cdot \frac{n_{Yes}}{n}$$

$$E(Female|No) = n \cdot p(Female) \cdot p(No) = n \cdot \frac{n_{Female}}{n} \cdot \frac{n_{No}}{n}$$

Then, the Chi-square score is calculated by

$$\begin{aligned}\mathrm{Chi-square} &= \frac{(O(Male|Yes) - E(Male|Yes))^2}{E(Male|Yes)} \\ &+ \frac{(O(Male|No) - E(Male|No))^2}{E(Male|No)} \\ &+ \frac{(O(Female|Yes) - E(Female|Yes))^2}{E(Female|Yes)} \\ &+ \frac{(O(Female|No) - E(Female|No))^2}{E(Female|No)}\end{aligned}$$

The greater the difference between the observed and the expected values, more the Chi-square value. This indicates that the input feature (patient gender) is dependent on the response variable (treatment response). The confidence that we can assign to the calculated Chi-square value can be found using the table of Chi-square statistics. We need to find the degree of freedom, which is calculated as as $(\text{r-1}) \cdot (\text{c-1})$ where r and c are rows and columns of the contingency table. For the table above, the degree of freedom is 1.

In feature selection, thus, the Chi-square test allows us to test the relationship between an input categorical feature with a categorical output or response variable. It is also a hypothesis-testing technique, like ANOVA, but it is used when the input features and the output or response variable are categorical. Chi-square is a univariate feature selection technique since it independently evaluates the importance of a single input feature. It is also a supervised feature selection technique since it requires the response or output (labels) variable.

The null hypothesis is that the input feature is independent of the response variable. The alternative hypothesis is that the feature is dependent on the response variable. High p-values of Chi-square tests indicate that the null hypothesis is not correct and therefore it is rejected in favor of the alternative hypothesis. Therefore, higher values of Chi-square tests indicate that the feature is more dependent on the response variable, i.e., it has more importance. In other words, the importance of an input categorical feature increases with the increase of its Chi-square test, indicating that it has a better relationship, similarity, or relevance to the categorical response variable.

Let us take the following example representing the clients who either left a bank or decided to stay. This dataset is used to predict the bank **churn rate**. The dataset can be uploaded from Dasgupta (2019). It contains 1000 rows and 14 columns. Each row represents a client; the columns represent the description of the client characteristics, such as age, country, ID, name, gender, and estimated salary. The last column is the response or output variable which represents whether the client left the bank (value 1) or decided to stay (value 0). In this example, we will focus on three categorical features (Gender, HasCrCard, IsActiveMember) since Chi-square tests are usually applied for the assessment of the importance of categorical input features. The goal is to rank these features according to their relevance to the response variable. The table below shows the first five rows of the dataset with respect to these three features.

Churn rate
A customer churn is the rate of customers who stopped using a service or product in a given time frame.

Table 27: Churn Rate According to Gender, HasCrCard, and IsActiveMember Features

Gender	HasCrCard	IsActiveMember	Response variable (Exited/Not-Exited)
Female	1	1	1
Female	0	1	0
Female	1	0	1
Female	0	0	0
Female	1	1	0

Source: Sayed-Mouchaweh (2021).

Let us apply a Chi-square test using Python to this example in order to rank these three categorical features. The steps are demonstrated below.

1. Import the required libraries and functions.

   Code
   ```
   import pandas as pd
   from sklearn.feature_selection import SelectKBest, chi2
   from kaggle.api.kaggle_api_extended import KaggleApi
   import zipfile
   from sklearn.preprocessing import LabelEncoder
   ```

2. The dataset is contained in a file called “client.csv” that can be downloaded from Dasgupta (2019). After loading this file in Python, assign the dataset to a DataFrame variable as follows:

   Code
   ```
   # log into kaggle
   api = KaggleApi()
   api.authenticate()

   # download the data
   kaggle_user = 'sonalidasgupta95'
   kaggle_project = 'churn-prediction-of-bank-customers'
   api.dataset_download_files(kaggle_user + '/' + kaggle_project)
   ```

```
# unzip the data
zip = zipfile.ZipFile(kaggle_project + '.zip').\
   extractall()

# load the data
churn_df = pd.read_csv('Churn_Modelling.csv')
```

3. Divide the dataset into two DataFrames: One contains only the input features, and the other contains the response variable. Then, create a new DataFrame containing only the three categorical input features that we want to rank using the Chi-square test.

   Code
   ```
   y = churn_df['Exited']
   X = churn_df[['Gender', 'HasCrCard', 'IsActiveMember']]
   ```

4. Transform the feature "Gender" into a numeric feature using the LabelEncoder technique. LabelEncoder is a scikit-learn function that converts alphabetically each category in a column to a number between 0 and 1. For instance, "Gender" will be replaced by two labels 0 and 1 since "Gender" has two categories "M" and "F." "F" will be concerted to "0" and "M" to "1."

   Code
   ```
   from sklearn.preprocessing import LabelEncoderX['Gender']
   = LabelEncoder().fit_transform(X['Gender'])
   ```

5. Apply the module SelectKBest using the ranking criterion Chi-square test. We set k to be 2 to select the best two features from the three original features.

   Code
   ```
   # create and fit feature selector
   selector = SelectKBest(chi2, k=2)
   selector.fit(X,y)

   # apply feature selector to the data
   X_new = selector.transform(X)
   ```

6. Display the name and the rank of the three features. The Chi-square test uses p-values, as ANOVA, in order to rank the three features (the lower the p-value, the better the feature rank). We will have the following results: "Gender, 7.01e-13," "HasCrCard, 0.69, "IsActiveMember," 1.56e-27. We can see the most relevant feature is "IsActiveMember," then "Gender," and finally "HasCrCard."
   A p-value less than or equal to 0.05 is statistically significant. It indicates strong evidence that the input feature is relevant (dependent) to the response variable, as there is less than or equal to a five percent probability that the independent hypothesis

(null hypothesis) is correct. Therefore, “HasCrCard” does not have a significant relevance with the response variable (exited/not exited) since its p-value is very high (above 0.05).

7. **Code**

```
# print Chi²-statistics- and p-values per feature
pd.DataFrame({'features': X.columns.values, \
   'Scores': selector.scores_, \
   'p-values': selector.pvalues_})

# console output:
#  features    Scores        p-values
# 0 Gender      51.539926     7.015575e-13
# 1 HasCrCard   0.150041     6.984962e-01
# 2 IsActiveMember 118.199414 1.568036e-27
```

Mutual Information

Mutual information (MI) measures the amount of information gained, or the reduction in the uncertainty, about one variable or feature given a known value of another feature. For instance, if variable X is the roll of a fair 6-sided die, and Y indicates if the roll is even, $\mathrm{Y} = 1$, or not, $\mathrm{Y} = 1$. The value of Y provides information about the value of X and vice versa. Indeed, knowing that $\mathrm{Y} = 1$ provides information that X is one of the following values: 2, 4, and 6. Therefore, X and Y share mutual information. The mutual information between two variables X and Y can be calculated as follows:

$$MI(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \cdot \log_2\left(\frac{p(x,y)}{p(x) \cdot p(y)}\right)$$

where $\mathrm{p(x)}$ and $\mathrm{p(y)}$ are the marginal probabilities, and $\mathrm{p(x,\ y)}$ is the joint probability.

Table 28: How to Calculate the Mutual Information (Example)

$\mathrm{X = x1}$	$\mathrm{X = x1}$	$\mathrm{X = x1}$	$\mathrm{X = x1}$	$\mathrm{X = x2}$	$\mathrm{X = x2}$
$\mathrm{Y = 1}$	$\mathrm{Y = 2}$	$\mathrm{Y = 2}$	$\mathrm{Y = 2}$	$\mathrm{Y = 1}$	$\mathrm{Y = 2}$

Source: Sayed-Mouchaweh (2021).

To understand how the mutual information can be calculated, let us take the following example illustrated in the table above. We have $\mathrm{p(x1) = 4/6,\ p(x2) = 2/6 = 0.5,\ p(Y{=}1) = 2/6,\ p(Y = 2) = 4/6,\ p(x1,Y = 1){=}1/6,\ p(x1,Y = 2) = 3/6,\ p(x2,Y = 1) = 1/6,\ and\ p(x2,Y = 2) = 1/6}$. Therefore, mutual information is calculated as follows:

$$MI(X,Y) = p(x1, Y=1)*\log_2\left(\frac{p(x1, Y=1)}{p(x1)*p(Y=1)}\right) + p(x1, Y=2)* \log_2\left(\frac{p(x1, Y=2)}{p(x1)*p(Y=2)}\right) + p(x2, Y=1)*\log_2\left(\frac{p(x2, Y=1)}{p(x2)*p(Y=1)}\right) + p(x2, Y=2)* \log_2\left(\frac{p(x2, Y=2)}{p(x2)*p(Y=2)}\right) = -1 .87$$

Mutual information measures the dependence between two variables. In feature selection, these two variables are an input feature X and the response or output variable y. Therefore, the mutual information $\mathrm{MI(X,\ y)}$ between X and y measures the average amount of information, gain of information, that X transfers about y. The prediction power increases when the value of mutual information $\mathrm{MI(X,\ y)}$ increases. Therefore, MI can be used to rank the input features X with respect to the importance of their dependence with the response variable y. The mutual information has the following properties:

- It is positive $0 \leq \mathrm{MI}$.
- It is symmetric $\mathrm{MI(X,y)} = \mathrm{MI(y,X)}$.
- It is equal to zero when the input feature and the response variable are independent. In this case, the input feature is not relevant and should be dropped.

The advantage of mutual information over ANOVA and the Chi-square test is that it is better suited to capturing nonlinear relationships between X and y because it is based on the use of the logarithmic function, which can help to make a nonlinear relationship appear linear in some cases. Mutual information is a univariate feature selection technique that is used to measure the dependence or relevance between an input feature and the response or output (label) variable. Therefore, it is a supervised feature selection technique since we need to have the output variable. It belongs to the filter feature selection category of techniques since it can be calculated without the need for any machine learning before the selection is conducted.

Let us use Python in order to use the mutual information between the different input features of the Iris dataset and its output or label variable as a selection criterion. We will rank the different input features according to their relevance (mutual information or dependence) to the output variable and then select the best $\mathrm{k} = 2$ features. The steps are demonstrated below:

1. Import the required libraries and functions and import the Iris dataset from `skleran.datasets` as a DataFrame. The `mutual_info_classif` function allows us to compute the mutual information for the classification task.

   Code
   ```
   from sklearn.feature_selection import mutual_info_classif
   from sklearn.feature_selection import SelectKBest
   import pandas as pd
   from sklearn.datasets import load_iris
   iris = load_iris()
   ```

```
feature_names = load_iris().feature_names
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.DataFrame(iris.target)
```

2. Use the `mutual_info_classif` criterion as a score function in SelectKBest and rank the features according to their mutual information. Then, display the name and the mutual information for each of the four features of the Iris dataset.

   Code
   ```
   selector = SelectKBest(score_func=mutual_info_classif, \
      k=2)
   X_new = selector.fit_transform(X, y)
   ```

   Running the code above provides the following rank (from the highest to the lowest) for the four input features of the Iris dataset: “petal length (cm),” 1.00, “petal width (cm) ”0.88, “sepal length (cm),” 0.63, and “sepal width (cm),” 0.32.
3. Display the name and mutual information of the $k = 2$ best selected features according to their mutual information. The two best selected features are “petal length” and “petal width.”

   Code
   ```
   # print mutual information per feature
   pd.DataFrame({'features': X.columns.values, \
      'Scores': selector.scores_})

   # console output:
   #   features          Scores
   # 0 sepal length (cm)  0.508725
   # 1 sepal width (cm)   0.302152
   # 2 petal length (cm)  0.982984
   # 3 petal width (cm)   0.994338
   ```

Permutation Feature Importance

Feature permutation allows us to evaluate a feature’s impact on the model prediction power. The amplitude of this impact represents the importance of the permutated feature. For instance, we build a machine learning model to predict the weight of a person using the description of their characteristics, such as height and eye color. We start by splitting the available data into training and test data. The training dataset is used learn, fit, the model. We then calculate the accuracy rate of this model using the test dataset. After that, we start by randomly permuting the values of the each column (i.e., feature). Then, we calculate the accuracy rate of the already built model. If the permutated feature is not important, then the reduction in accuracy rate of the model will be minor. If this reduction is high, this indicates that the feature is important since it has a big impact on the model prediction accuracy. For instance, permuting the values of the “eye color” will not have an impact on the accuracy rate of the model, while permuting the values of the “height” feature will reduce the accuracy rate considerably.

Permutation feature importance is model agnostic in the sense that this technique can be applied to any machine learning algorithm. It is a supervised feature selection technique since the labels (target variable) are required to calculate the model accuracy.

Let us utilize Python to evaluate the feature importance of the Iris dataset by using the permutation feature importance technique. The steps are demonstrated below:

1. Import the libraries and load the Iris dataset.

   Code
   ```
   # load libraries
   import pandas as pd
   from sklearn.datasets import load_iris
   from sklearn.model_selection import train_test_split
   from sklearn.inspection import permutation_importance
   from sklearn.neighbors import KNeighborsClassifier

   # load sample data
   iris = load_iris()
   ```

2. Divide the Iris dataset into training and test sets.

   Code
   ```
   X_train, X_test, y_train,y_test = train_test_split(\
          iris.data, iris.target)
   ```

3. Build the model using a machine learning approach such as k-nearest neighbors, then train the model using the training set.

   Code
   ```
   # create and fit a KNN model
   model = KNeighborsClassifier()
   model.fit(X_train, y_train)
   ```

4. Perform permutation feature importance using the test dataset and repeat the permutation 10 times.

   Code
   ```
   # assess feature importance by permutation
   feat_imp = permutation_importance(model, \
      X_test, y_test, n_repeats=10, \
          scoring='accuracy')
   ```

5. Display the mean importance of each feature of the Iris dataset.

Code
```
pd.DataFrame({'features': iris.feature_names, \
 'importances_mean': feat_imp['importances_mean'], \
 'importances_std': feat_imp['importances_std']})

# console output:
#  features        importances_mean importances_std
# 0 sepal length (cm) -0.034211       0.016850
# 1 sepal width (cm) -0.021053       0.015789
# 2 petal length (cm) 0.544737       0.110432
# 3 petal width (cm)  0.107895       0.049162
```

The scores in the table above show how much model performance and accuracy rate decreased with a random permutation. Therefore, the most important features are first “petal length,” then “petal width.” The other two features have very little importance since their impact on the accuracy rate is very small. The presented techniques evaluate feature importance for the whole dataset and trained machine learning model.

There are other techniques that consider individual samples for the feature importance evaluation, such as minimum/maximum accuracy samples, Local Interpretable Model-agnostic Explanations (LIME) (Thanh-Hai et al., 2020), and SHapley Additive exPlanations values (SHAP) (Marcílio & Eler, 2020). These methods are all model agnostic in the sense that they can be applied to any kind of machine learning algorithm. A detailed explanation of these techniques is beyond the scope of this section, but the main concepts are briefly presented.

In minimum/maximum accuracy feature importance, the feature importance is based on its contribution to the prediction of the sample in the test dataset with the highest accuracy as well as the sample with the lowest. The model agnostic LIME technique selects one sample x and generates samples around it. LIME weights the generated samples according to their distances to x. A simple machine learning approach, such as linear regression, is then applied to assess the importance of features on the prediction of x by using the regression weights. The model agnostic SHAP explains the prediction of a data point x by calculating the contribution of each feature to its prediction. Therefore, features with large absolute Shapley values are important since they largely contribute to the prediction of the different data points in the dataset.

5.2 Feature Variance

Features with the same value do not carry information and must be removed. Indeed, features that do not vary within themselves do not possess predictive power. The variance of a feature indicates how much it varies within itself. When a feature has one unique value which is the same for all samples, then its variance is equal to zero. Therefore, features with a variance that does not exceed a predefined threshold should be dropped. The other features, possessing a variance greater than this threshold, will be selected. The threshold is defined in order to remove features (columns) that have few unique values with respect to the number of rows (data points) or when the ratio of the frequency of the most common value to the frequency of the second most common value is high.

Feature variance belongs to the filter feature selection category since it filters the features before any machine learning model is trained. In addition, it is an unsupervised feature selection technique since it does not need the response (output or labels) column in order to compute the variance of a feature.

Let us take the example of the table below. We can see that one column, feature F1, has only one unique value, therefore it can be dropped. The fourth column has a small variance, so it will also be dropped as it does not introduce considerable information to the model. Only features F2 and F3 will be selected since they have a large variance. We can see that each feature was independently evaluated by computing its variance. Hence, selecting features with respect to their variance is a univariate feature selection.

Table 29: Selecting High Variance Features (Example)

	F1	F2	F3	F4
	0	2	0	3
	0	3	4	3
	0	5	1	2
Variance	0	1.55	0.88	0.22

Source: Sayed-Mouchaweh (2021).

Let us run this example using Python. We can do this by using the function VarianceThreshold() from the library `sklearn.feature_selection`. This function drops all the features that have zero variance. However, if we want to select only the features with a variance greater than a predefined threshold, then we can set the parameter threshold of this function as we will see in the following code. The steps are demonstrated below:

1. Import the required libraries and function and then create the example of the table above.

   Code
   ```
   # load libraries
   import numpy as np
   import pandas as pd
   from sklearn.feature_selection\
      import VarianceThreshold
   from sklearn.feature_selection import VarianceThreshold
   from sklearn.datasets import load_iris
   from matplotlib import pyplot as plt
   ```

```
# generate sample data
X = np.array([[0, 2, 0, 3], [0, 3, 4, 3], \
   [0, 5, 1, 2]])
```

2. Apply the function VarianceThreshold with a predefined threshold equal to 0.4 in order to drop out features with variance less than 0.4 and then display the variances of the features and the selected features.

   Code
   ```
   # apply variance threshold
   selector = VarianceThreshold(threshold=0.4)
   Xs = selector.fit_transform(X)

   # show the variances per feature
   # (the ones above the threshold were chosen)
   print(selector.variances_)
   # console output:
   # [0. , 1.55555556, 2.88888889, 0.22222222]
   ```

   Let us now use the function VarianceThreshold in order to select the relevant features of Iris dataset.
3. Load the Iris dataset as a DataFrame.

   Code
   ```
   iris = load_iris()
   X = pd.DataFrame(iris.data, columns=iris.feature_names)
   y = iris.target
   ```

4. Create and fit the VarianceThreshold object and use it to transform the data (select features).

   Code
   ```
   selector = VarianceThreshold(threshold=0.6)
   selector.fit_transform(X)
   ```

5. Display the name and variance of each of the four features of the Iris dataset.

   Code
   ```
   # show the variances per feature
   # (the ones above the threshold were chosen)
   pd.DataFrame({'features': iris.feature_names, \
      'variances': selector.variances_})
   ```

   The table below shows the result of running the code above.

Table 30: Name and Variance of Each of the Four Iris Dataset Features

'sepal length (cm)'	0.68
'sepal width (cm)'	0.18
'petal length (cm)'	3.09
'petal width (cm)'	0.57

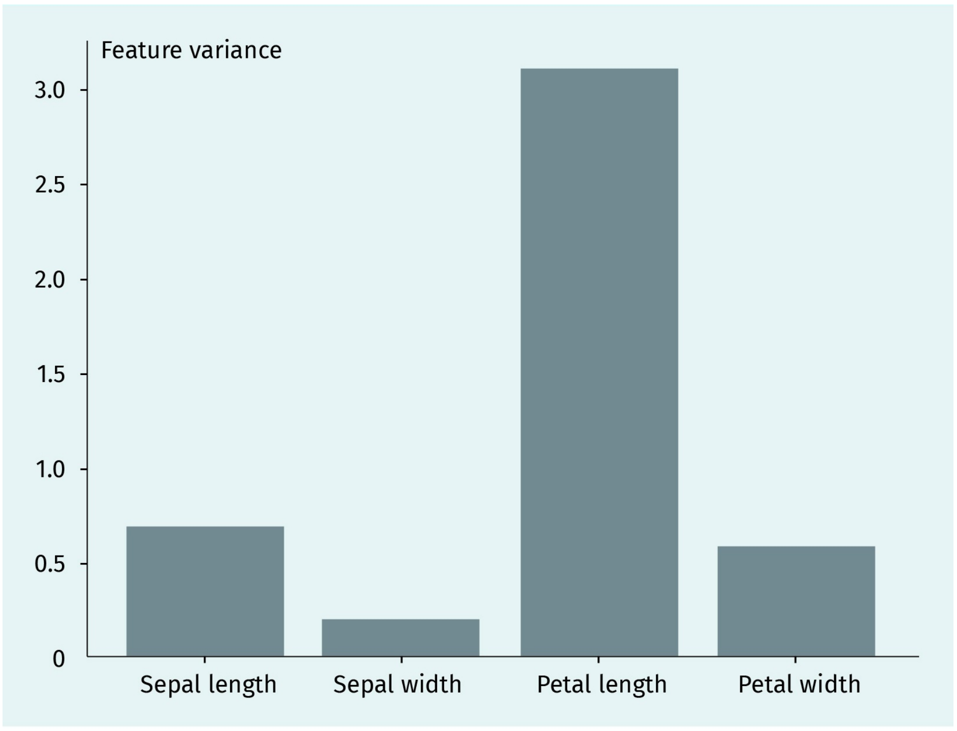
Source: Sayed-Mouchaweh (2021).

6. Create a bar chart to visually compare the features variances. The figure below shows the obtained bar chart for the Iris dataset. Please note that the features in the Iris dataset are ordered as follows: feature 0 is “sepal length,” feature 1 is “sepal width,” feature 2 is “petal length,” and feature 3 is “petal width.”

Code

```
plt.bar(x=feature_names, height=selector.variances_)
plt.ylabel('Feature Variance')
plt.title('Iris features variance comparison')
plt.show()
```

Figure 43: Features Variances of the Iris Dataset (Bar Chart)



Source: Sayed-Mouchaweh (2021).

7. Select features with variance greater than 0.6 and display the names of selected features. The method `get_support()` returns a Boolean array where the value "True" is assigned to the selected features. For instance, the table above showing the features variances, get_support() will provide the following array [True, False, True, False] when 0.6 is used as a threshold (as it was set in the sample code above). This indicates that the first and third features are selected since their variances are greater than 0.6.

Code
```
for selected_feature in \
selector.get_support(indices=True):
print('* ' + feature_names[selected_feature])
# console output:
# * sepal length (cm)
# * petal length (cm)
```

5.3 Correlation Matrix

The correlation between two variables indicates that values in one variable systematically move toward one direction if values in the other variable systematically move in one direction. Therefore, when two variables are correlated, as one variable changes, so does the second variable. However, it is worth mentioning that the correlation between two variables does not necessarily imply a causation between them. This relationship or correlation between the two variables can be positive when they change in the same direction or negative when they change in two different directions. When the correlation is equal to zero, then the two variables do not show this systematic movement.

The correlation, in the feature selection process, can be between two input features or between an input feature and the target or response variable. Therefore, the correlation can be a univariate and multivariable feature selection technique. In the univariate mode, the correlation between an input feature and the response feature is calculated in order to select the relevant features to the output (label) prediction and to remove the irrelevant features. In the multivariate mode, the correlation is calculated between two input features to remove the redundant features, i.e., features highly correlated, thus, carrying the same information. Both cases improve the performance of the trained models (prediction accuracy, quality of obtained clusters, learning time, processing time, etc.) and reduce overfitting issues. Finally, the correlation can be considered as both a supervised and unsupervised feature selection technique since it can be used when the response variable is available or unavailable.

There are several correlation techniques used to measure the correlation between two variables. The first technique is called the covariance and is calculated as follows:

$$Cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \mathrm{mean}_x\right)\cdot\left(y_i - \mathrm{mean}_y\right)$$

where n is the number of rows in X and Y.

The covariance measures a linear relationship between two variables. However, the interpretation of its values is not easy. To overcome this problem, the Pearson’s correlation is commonly used. Pearson’s correlation is the normalization of the covariance between the two variables by dividing the covariance by the product of their standard deviations as follows:

$$\mathrm{Pearson'scorrelationscore}\left(X,Y\right) = \frac{Cov(X,Y)}{\sigma_X\cdot\sigma_Y}$$

The normalization of the correlation allows us to facilitate its interpretation since the Pearson’s correlation score is between —1 and +1. When it is close to —1 or +1, this indicates strong negative or positive correlation between X and Y, respectively. When it is equal to zero, this indicates that both variables X and Y are independent. When the Pearson’s correlation scores are calculated between each pair of the features as well as between each feature and the response variable, the result is a symmetric matrix called a correlation matrix. Each cell of the diagonal of the correlation matrix has a value of 1.0 since each feature perfectly correlates with itself.

Let us use Python to compute the correlation matrix of the Iris dataset and select the best two features according to the Pearson’s correlation score. The steps are demonstrated below:

1. Import the required libraries and upload the Iris dataset as a DataFrame.

   Code
   ```
   # load libraries
   import pandas as pd
   import numpy as np
   from sklearn.datasets import load_iris
   from matplotlib import pyplot as plt
   import seaborn as sns
   from scipy import stats

   # load sample data
   iris = load_iris()
   X = pd.DataFrame(iris.data, \
      columns=iris.feature_names)
   y = pd.DataFrame(iris.target, \
      columns=['Response_variable'])
   XY = pd.concat([X, y], axis=1)
   ```

2. Concatenate the input features X with the response variable y to compute the correlation matrix between each pair of columns (between each pair of input features and between each input feature and the response variable). Then, compute the Pearson's correlation score and display the correlation matrix. The table below shows the Pearson's correlation matrix. Based on this table, we can observe that "petal width" and "petal length" are very relevant to the response variable. However, they are strongly correlated to each other. Therefore, it is better to select one of them since their strong correlation may indicate their redundancy. "sepal width" is the feature least correlated with the other features. Hence, it may be selected as a second feature to retain.

   Code
   ```
   cor_mat = XY.corr(method='pearson')
   ```

Table 31: Pearson's Correlation Matrix for the Iris Dataset

	sepal length	sepal width	petal length	petal width	Response_variable
sepal length	1	—0.117	0.871	0.817	0.782
sepal width	—0.117	1	0.428	— 0.366	—0.426
petal length	0.871	—0.428	1	0.962	0.949
petal width	0.817	—0.366	0.962	1	0.956
Response_variable	0.782	—0.42	0.949	0.956	1

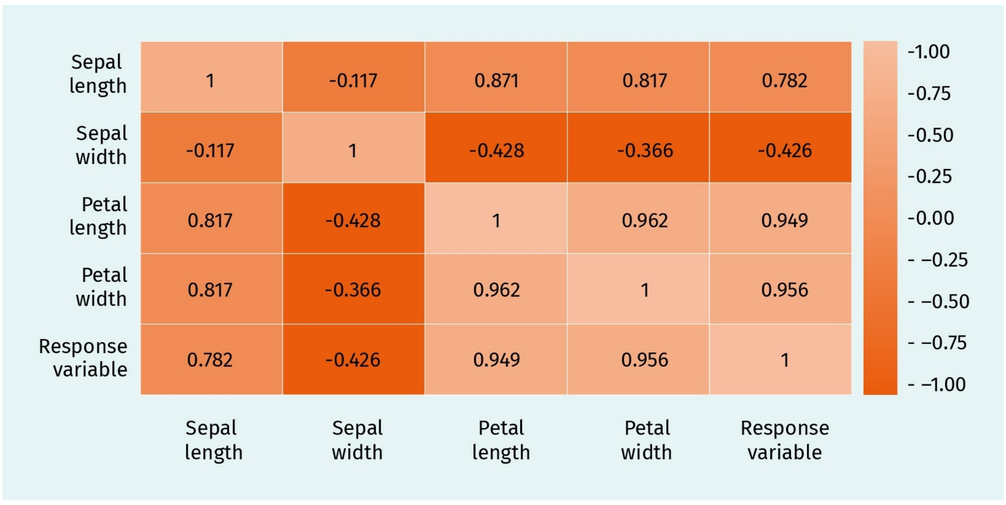
Source: Sayed-Mouchaweh (2021).

3. Visualize the correlation matrix using the function heatmap of the Python library seaborn. The parameter vmin indicates the minimum value of the correlation matrix; vmax indicaates the maximum value of the correlation matrix; and annot is set to True in order to integrate in each cell of the heatmap the corresponding Pearson's correlation score. Since the Pearson's correlation score constitute floating number values, its format is defined as "float" by assigning the value "f" to the parameter "fmt" (number format) of the function heatmap of the Python library seaborn. The figure below shows the Pearson's correlation heatmap for Iris dataset.

Code

```
ax = sns.heatmap(cor_mat, vmin=-1, vmax=1, \
   annot=True, fmt="f")
plt.show()
```

Figure 44: Pearson's Correlation Heatmap for the Iris Dataset



Source: Sayed-Mouchaweh (2021).

The weak point of Pearson's correlation is its assumption that the data points in each column follow a Gaussian or normal probability distribution and the relationships are linear between the different columns or variables. One alternative to overcome this limit is the use of Spearman's correlation.

Spearman's correlation is a statistical measure of the strength of a monotonic relationship between variables or features. A monotonically increasing, respectively decreasing, relationship between two variables x and y means that when x increases, y will never decrease, but will respectively increase. However, a non-monotonic relationship means when x increases, y sometimes decreases and sometimes increases.

Spearman's correlation is similar to Pearson's correlation in the sense that its values is between —1 and 1 where

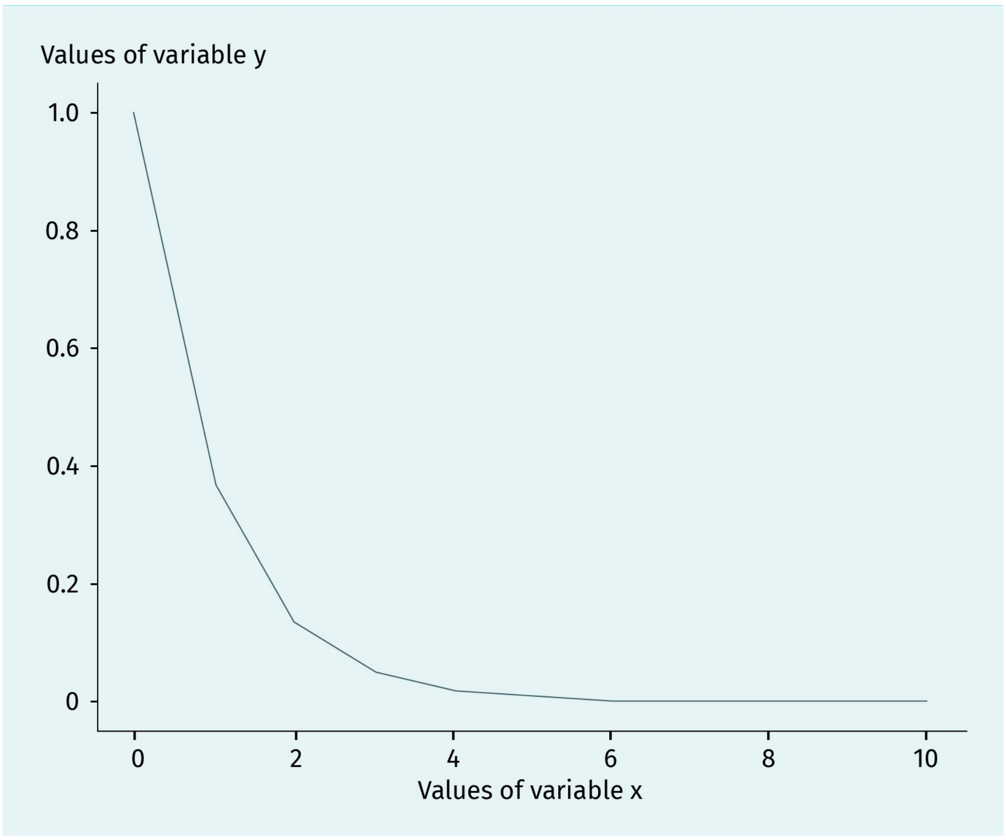
- 0 to ± 0.19 indicates a very weak correlation.
- 0.20 to 0.39 (—0.39 to —0.2) indicates a weak correlation.
- 0.40 to 0.59 (—0.59 to —0.40) indicates a moderate correlation.
- 0.60 to 0.79 (—0.79 to —0.60) indicates a strong correlation.
- 0.80 to 1.0 (—1.0 to —0.8) indicates a very strong correlation.

However, Spearman's correlation is calculated by taking the rank of the two variables' values as follows. The rank of the values of a variable is obtained by sorting these values in decreasing order:

$$Spearman'scorrelationscore\left(X, Y\right) = \frac{Cov(rank(X), rank(Y))}{\sigma_{rank(X)} \cdot \sigma_{rank(Y)}}$$

Let us take the example of a nonlinear relationship between two variables x and y represented as an exponential function $\mathrm{y} = \mathrm{e}^{\mathrm{x}}$ as illustrated in the figure below.

Figure 45: Nonlinear Correlation between Two Variables Represented as an Exponential Relationship



Source: Sayed-Mouchaweh (2021).

The values of x and y are represented in the table below. The values of x are sorted (ranked) in increasing order. The values of y are ranked in decreasing order since the exponential function is a monotonic function (i.e., when x increases, y will decrease, but never decrease). The table also shows the Pearson's correlation and the Spearman's correlation between x and y. We can see that Spearman's correlation better captures the nonlinear correlation between x and y than Pearson's correlation since it provides the perfect correlation value, 1.

Table 32: Pearson's and Spearman's Correlation for a Nonlinear Correlation Represented as an Exponential Relationship between Two Variables

x	y	rank(x)	rank(y)
0	1	0	10
1	0.36	1	9
2	0.13	2	8
3	0.05	3	7
4	0.018	4	6
5	0.006	5	5
6	0.002	6	4
7	0.0009	7	3
8	0.0003	8	2
9	0.0001	9	1
10	0.0000	10	0
Pearson's correlation coefficient		Spearman's correlation coefficient	
—0.69		—1	

Source: Sayed-Mouchaweh (2021).

The code below shows how to compute the Pearson's and Spearman's correlations for the example above using Python. In this example, we also test for normal distribution of the data and see that the data are not normally distributed ($p < 0.05$ in the normality test for y). Therefore, we are not allowed to use Pearson's correlation as this method assumes a normal distribution of the data. We use the Spearman's rank correlation method as this method has less explanatory power since it disregards parts of the information in the data. However, the key with this method is that it does not assume a normal distribution of the data.

Code

```
# generate sample data
dat = pd.DataFrame({'x': np.arange(0,10), \
   'y': np.exp(-np.arange(0,10))})

# compute Pearson’s correlation and
# display the correlation matrix
XY.corr(method='pearson')

# console output:
#   x           y
# x 1.00000     -0.71687
# y -0.71687 1.00000

# test for normal distribution of the data
stats.normaltest(XY['x'])
# console output:
# NormaltestResult(statistic=2.02697581498966,
# pvalue=0.362950830342156)

stats.normaltest(XY['y'])
# console output:
# NormaltestResult(statistic=19.779358749097575,
# pvalue=5.0695197559354735e-05)

# compute Spearman's rank correlation and
# display the correlation matrix
XY.corr(method='spearman')

# console output:
#   x       y
# x 1.0    -1.0
# y -1.0   1.0
```

In order to work efficiently by showing strong evidence-based results, the Spearman's correlation requires a monotonic relationship between the two variables in the sense that when $\mathrm{x1} > \mathrm{x2}$ then $\mathrm{y1} < \mathrm{y2}$. This is because Spearman's correlation is based on the rank of the values of both variables. However, the Spearman's correlation is still an alternative to be used when the data are not normally distributed and there is simply no other choice to evaluate the correlation between abnormally distributed data samples.

5.4 Recursive Feature Selection

Recursive feature selection is a wrapper-type feature selection process that allows us to select relevant features by adding the most relevant or eliminating the least relevant features iteratively. It is a wrapper-type feature selection because it is based on the use of a certain machine learning method to score the features according to the importance of their contribution to the quality or accuracy of the output (e.g., the obtained clusters and prediction). The most and least important features are first added or eliminated, the model is trained again, refitted, using the most important features. This process is repeated until it reaches the specified number of features to be selected or a desirable quality of the model.

Recursive feature selection techniques can be divided into: Exclusive Feature Selection (EFS), Sequential Forward feature selection (SFS), Sequential Backward feature selection (SBS), and Recursive Feature Elimination (RFE). All of these recursive feature selection techniques have two hyper-parameters to set: the number k of features to select and the model or learning algorithm to use to select these features.

Exclusive Feature Selection (EFS)

The exclusive Feature Selection (EFS) technique selects the best combination of k features through all the existing combinations of the original features. It is a supervised multivariate wrapper technique. In the case of 3 features F1, F2, and F3 and, in order to find the $\mathrm{k} = 2$ best features, EFS ranks the following combinations: (F1), (F2), (F3), (F1, F2), (F1, F3), (F2, F3). Then, it selects the $\mathrm{k} = 2$ features allowing us to have the best rank. Let us suppose we have the following rank in decreasing order: (F2, F3), (F1, F2), (F1, F3), (F2), (F3), (F1), then EFS will select the combination of $\mathrm{k} = 2$ features (F2, F3). In order to rank the features, EFS requires a predictive model (regressorclassifier) built using a machine learning method, such as logistic regression, k-nearest neighbors, or tree classification. For each combination of the original features, the performance of the built model is evaluated using an evaluation metric, such as accuracy rate, recall (True Positive Rate), Area Under the Curve or **AUC**, and Receiver Operation Curve or **ROC**.

AUC
The Area Under the Curve (AUC) is a measure of separability between classes. It indicates how much the model is able to discriminate classes. Therefore, the higher the AUC, the better the model at predicting the classes. It is calculated as the area under a ROC curve.

The EFS method is available via the `mlxtend.feature_selection` in scikit-learn. We will show step by step how to use Python in order to select the best features of the Iris dataset using EFS method:

1. Import libraries and load the Iris dataset.

   Code
```
# import libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from mlxtend.feature_selection \
   import ExhaustiveFeatureSelector as EFS
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

# load sample data
iris = load_iris()
x = pd.DataFrame(iris.data, \
   columns=iris.feature_names)
```

2. Import the logistic regression classification method from scikit-learn in order to use it to build the model that will be used by the EFS technique to rank the features. Then create the logistic regression classifier (model).

Code
```
# Create a logistic regression object
lr = LogisticRegression()
```

3. Create an EFS object. In this object, you use the logistic regression model as an estimator. The minimum number and maximum number of features to be selected are 1 and 3, respectively. The scoring metric used to rank the features is the accuracy rate of the model, which is the Logistic Regression classifier, and the method used to compute the accuracy is the cross-validation. The cross-validations is set to be performed in five-folds.

Code
```
# create an EFS object
efs = EFS(estimator=lr,
         min_features=1,
         max_features=3,
         scoring='accuracy',
         cv=5)
```

4. Train the EFS model with the Iris dataset

Code
```
efs = efs.fit(x, iris.target)
```

5. Display the names and the indexes of the best selected features. `efs.best_idx` and `efs.best_features_names` show the index and the names of the features that provide the best accuracy score, respectively.

Code
```
efs.best_feature_names_
# console output:
# ('sepal length (cm)', 'petal length (cm)',
# 'petal width (cm)')
```

6. Display the performances of each combination of Iris features. These performances can be displayed in a pandas DataFrame format using the get_metric_dict method of the SequentialFeatureSelector object. For each feature (row), there are the following columns: `avg_score` (the average score through the five-fold cross validation), `ci_bound` (the confidence interval around the computed cross validation scores), `feature_idx` (feature index), `feature_names`, `std_dev` (the standard deviation of the cross validation scores), and `std_err` (the standard errors of the cross validation scores). The results are displayed in decreasing order with respect to the average score of features.

Code

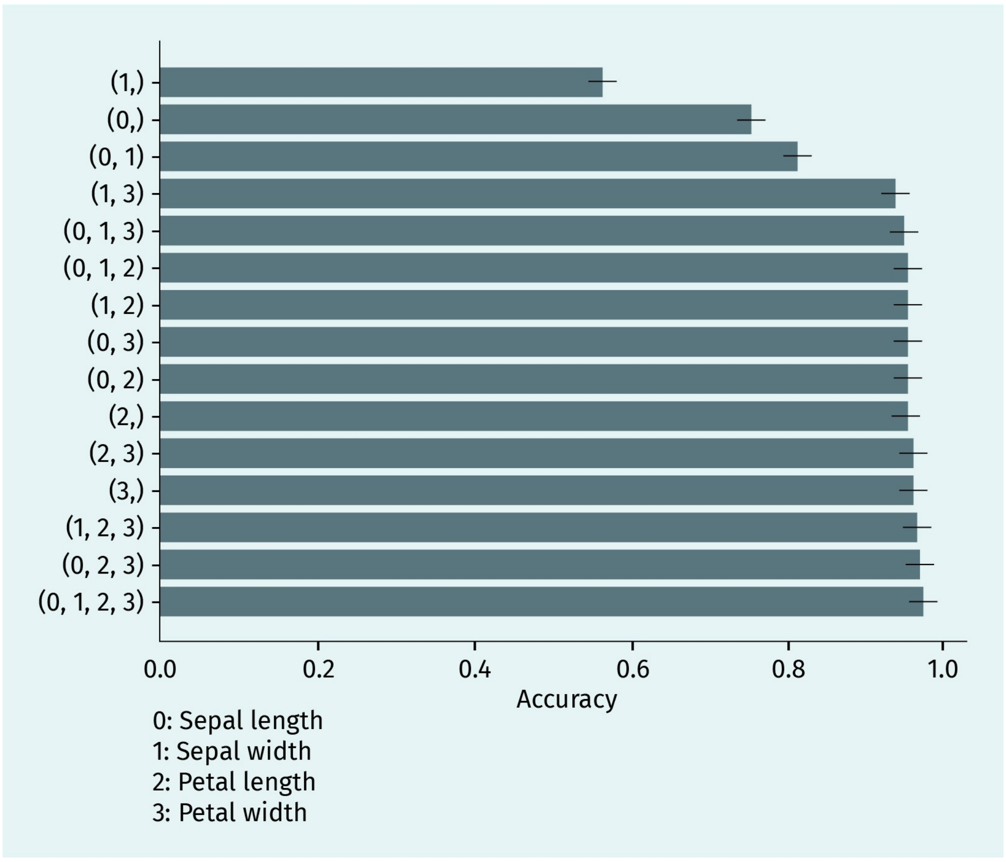
```
# show a full report on the feature selection
efs_results = pd.DataFrame(efs.get_metric_dict()).\
   T. \
   sort_values(by='avg_score', ascending=False)
```

7. Display a horizontal bar chart showing the average accuracy score of each combination of the Iris dataset features. The figure below shows the obtained horizontal bar chart where each bar represents the average accuracy score for each feature combination as well as its standard error.

   Code

   ```
   # create figure and axes
   fig, ax = plt.subplots()

   # plot bars
   y_pos = np.arange(len(efs_results))
   ax.barh(y_pos, efs_results['avg_score'], \
      xerr=efs_results['std_err'])

   # set axis ticks and labels
   ax.set_yticks(y_pos)
   ax.set_yticklabels(efs_results['feature_names'])
   ax.set_xlabel('Accuracy')

   # show the plot
   plt.show()
   ```

Figure 46: Horizontal Bar Chart Obtained by EFS for the Iris Features



Source: Sayed-Mouchaweh (2021).

However, as models have to be trained over and over again, EFS is computationally expensive in particular for datasets described by a large number of features. The following techniques are alternatives to overcome this limitation.

Sequential Forward Feature Selection (SFS)

The Sequential Forward feature Selection (SFS) technique first selects the most relevant feature, then it iteratively adds new relevant features until reaching the predefined number k of features to be selected. In the case of features F1, F2, and F3, and in order to find the $\mathrm{k} = 2$ best features, SFS ranks the individual features using a specific machine learning method as in the case of EFS. Then, the relevance of the combinations between the selected most relevant feature with the other features will be calculated in order to find the best combination. This process will be repeated until reaching the predefined number k of features to be selected. For instance, if in the first iteration, F3 was the most relevant feature, then in the second iteration, all of the combinations, (F1, F3), (F2, F3) between the other features and F3 will be ranked. If the combination (F2, F3) has a better accuracy score than the one for (F1, F3), then it will be selected. Therefore, SFS will provide the combination (F2, F3) as the best $\mathrm{k} = 2$ features.

The SFS method is available via the mlxtend.feature_selection in scikit-learn. We will show step by step how to use Python to select the best features of the Iris dataset using SFS method:

1. Import the necessary libraries and load the Iris dataset.

   Code
   ```
   # import libraries
   import pandas as pd
   import numpy as np
   from sklearn.datasets import load_iris
   from mlxtend.feature_selection \
      import SequentialFeatureSelector as SFS
   from sklearn.linear_model import LogisticRegression
   import matplotlib.pyplot as plt

   # load sample data
   iris = load_iris()
   x = pd.DataFrame(iris.data, \
      columns=iris.feature_names)
   ```

2. Create a logistic regression classifier.

   Code
   ```
   from sklearn.linear_model import LogisticRegression
   # Create a logistic regression model
   lr = LogisticRegression()
   ```

3. Create an SFS object. In this object, you use the logistic regression model as an estimator, consider any feature combination between 1 and 3 and set the parameter `forward` to “True” to perform SFS, the scoring metric used to rank the features is the accuracy rate of the Logistic Regression classifier, and the method used to compute the accuracy is the cross-validation. The cross-validation is set to be performed in five-folds.

   Code
   ```
   sfs = SFS(estimator=lr,
            k_features=(1, 3),
            forward=True,
            scoring='accuracy',
            cv=5)
   ```

4. Train the SFS object with the Iris dataset.

   Code
   ```
   sfs = sfs.fit(x, iris.target)
   ```

5. Display the names and the indexes of the best selected features. `sfs.k_feature_idx_` and `sfs.k_feature_names_` each show the index and the names of the features that provide the best accuracy score, respectively. For the Iris dataset, the indices of the best features are (0, 1, 2, 3), and their corresponding names are “sepal length (cm),” “sepal width (cm),” “petal length (cm),” and “petal width (cm).”

   Code
   ```
   # show the selected features
   sfs.k_feature_names_
   # console output:
   # ('sepal length (cm)', 'petal length (cm)',
   # 'petal width (cm)')
   ```

6. Display the performances of each combination of Iris features in a pandas DataFrame format using the `get_metric_dict` method of the `SequentialFeatureSelector` object. For each feature of these combinations, there are the same columns as with using EFS: `avg_score`, `ci_bound`, `feature_idx`, `feature_names`, `std_dev`, and `std_err`. The results are displayed in decreasing order with respect to the average score of features.

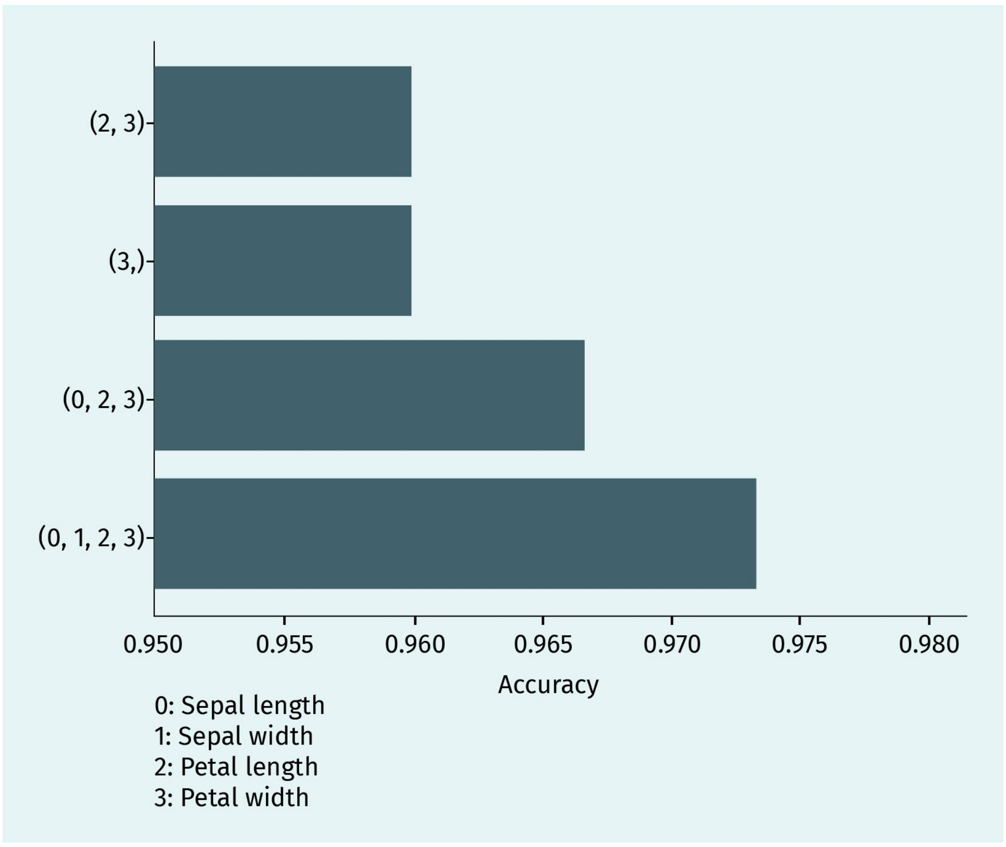
   Code
   ```
   sfs_results = pd.DataFrame(sfs.get_metric_dict()).\
      T. \
      sort_values(by='avg_score', ascending=False)
   ```

7. Again, we display a horizontal bar chart showing the average accuracy score of each combination of the Iris dataset features. The figure below shows the obtained horizontal bar chart where each bar represents the average accuracy score for each feature combination as well as its standard error.

   Code
   ```
   # create figure and axes
   fig, ax = plt.subplots()

   # plot bars
   y_pos = np.arange(len(sfs_results))
   ax.barh(y_pos, sfs_results['avg_score'], \
      xerr=sfs_results['std_err'])

   # set axis ticks and labels
   ax.set_yticks(y_pos)
   ax.set_yticklabels(sfs_results['feature_names'])
   ax.set_xlabel('Accuracy')

   # limit range to overimpose differences
   plt.xlim([0.95, 0.98])
   ```

```
# show the plot
plt.show()
```

Figure 47: Horizontal Bar Chart Obtained by SFS for the Iris Features



Source: Sayed-Mouchaweh (2021).

The figure above shows that the best single feature is feature #3 (“petal width”) since it gives the best accuracy against the other three single features. The best combination of two features (feature #3 and one of the other three features #0, #1, and #2) is the one between feature #3 and feature #1 (“petal length”). Then, the SFS technique searches for the best combination of three features starting from the combination of feature #1 and feature #3. This combination is formed by feature #1, feature #3, and feature #2 (“sepal length”). The last combination giving the best accuracy rate is the one combining all the four features.

Sequential Backward Feature Selection (SBS)

The Sequential Backward feature Selection (SBS) technique is similar to SFS but works in the inverse direction. It starts by considering all the features together and proceeds to rank all the combinations by removing one feature. Next, the combination with the best rank is conserved, and the process is repeated until reaching the predefined number k of features to be selected. In the case of features F1, F2, and F3, and in order to find the $\mathrm{k}=2$ best features, SBS starts by considering all the features. Then, it ranks all the combinations obtained by removing one feature. These combinations are (F2, F3), (F1, F3), and (F1, F2). The ranking is performed, as in the case of EFS and SFS, using a specific machine learning method. The best ranked combination is selected. Let us suppose that (F2, F3) is the best ranked one. Therefore, (F2, F3) will be selected by SBS as the best $\mathrm{k}=2$ features.

SBS is performed in Python in a similar way as using the SFS method. The only difference is that the parameter forward is set to "False." We will show step by step how to use Python to select the best features of the Iris dataset using SBS method:

1. Import libraries and load the Iris dataset as sample data.

   Code
   ```
   # import libraries
   import pandas as pd
   import numpy as np
   from sklearn.datasets import load_iris
   from mlxtend.feature_selection \
      import SequentialFeatureSelector as SBS
   from sklearn.linear_model import LogisticRegression
   import matplotlib.pyplot as plt

   # load sample data
   iris = load_iris()
   x = pd.DataFrame(iris.data, \
      columns=iris.feature_names)
   ```

2. Create the logistic regression classifier (model).

   Code
   ```
   lr = LogisticRegression()
   ```

3. Create an SBS object. It is similar to the creation of SFS object but we need to set the parameter "forward" to "False" in order to perform SBS. All the other parameters are the same as for the creation of an SFS object as we have seen in the code above.

   Code
   ```
   sbs = SBS(estimator=lr,
            k_features=(1, 3),
            forward=False,
            scoring='accuracy',
            cv=5)
   ```

4. Train the SFS object with the Iris dataset.

Code
```
sbs = sbs.fit(x, iris.target)
```

5. Display the names and the indexes of the best selected features. `sbs.k_feature_idx_` and `sbs.k_feature_names_` show the index and the names of the features that provide the best accuracy score, respectively.

Code
```
sbs.k_feature_names_
# console output:
# ('sepal length (cm)', 'petal length (cm)',
# 'petal width (cm)')
```

6. Display the performances of each combination of Iris features in a pandas DataFrame format using the get_metric_dict method of the SequentialFeatureSelector object. The obtained combinations are: (0, 1, 2, 3), (0, 2, 3), (2, 3), and (3). For each feature of these combinations, there are the following columns: `avg_score`, `ci_bound`, `feature_idx`, `feature_names`, `std_dev`, and `std_err`. The results are displayed in decreasing order with respect to the average score of features.

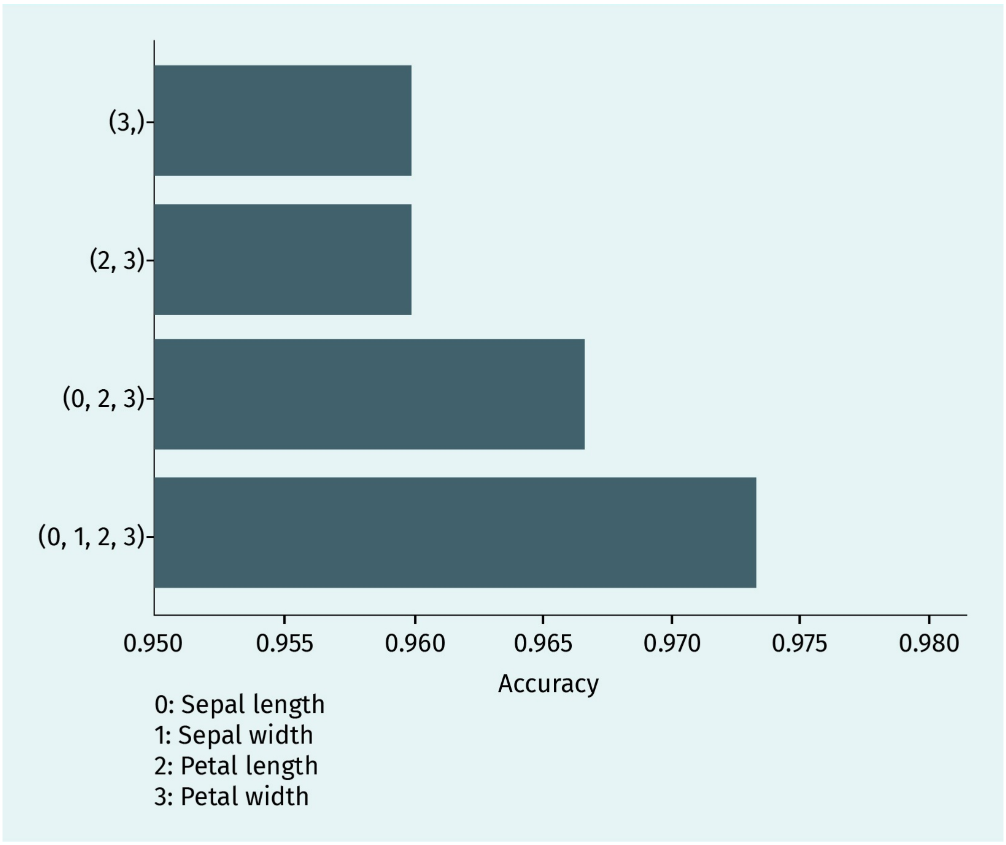
Code
```
sbs_results = pd.DataFrame(sbs.get_metric_dict()).\
  T. \
  sort_values(by='avg_score', ascending=False)
```

7. Display a horizontal bar chart showing the average accuracy score of each combination of the Iris dataset features. The figure below shows the obtained horizontal bar chart where each bar represents the average accuracy score for each feature combination as well as its standard error.

Code
```
# create figure and axes
fig, ax = plt.subplots()

# plot bars
y_pos = np.arange(len(sbs_results))
ax.barh(y_pos, sbs_results['avg_score'], \
   xerr=sbs_results['std_err'])

# set axis ticks and labels
ax.set_yticks(y_pos)
ax.set_yticklabels(sbs_results['feature_names'])
ax.set_xlabel('Accuracy')

# limit range to overimpose differences
plt.xlim([0.95, 0.98])
```

```
    # show the plot
    plt.show()
```

Figure 48: Horizontal Bar Chart Obtained by SBS for the Iris Features



Source: Sayed-Mouchaweh (2021).

The figure above shows the first combination containing the four features of the Iris dataset. This combination provides the best accuracy rate. SBS starts by removing one feature from this combination and computes the corresponding accuracy rate for each obtained combination of three features. The best combination of three features, which provides the best accuracy rate, is conserved. As we see in the figure above, this combination contains the features “sepal length,” “petal length,” and “petal width.” Then, SBS removes one feature from this selected combination to compute the accuracy rate for all the combinations of two features. SBS will conserve the combination of two features that has the best accuracy rate. This combination is the one containing the features “petal length” and “petal width.” Finally, SBS removes one feature from the selected combination of two features in order to compute the accuracy rate for the single feature “petal length” and the single feature “petal width.” The figure above shows that the feature “petal width” is the best-related single feature among the other Iris dataset features.

Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a popular feature selection technique that allows us to select the best or most important k features by eliminating iteratively the least important features. The number k of features to be selected and the model to be used to rank the features must be defined. REF starts by considering all the original features. REF ranks the features according to their importance or weight derived from a machine learning algorithm, such as logistic regression or tree-based approaches. The, REF removes the least important feature and builds a model using a machine learning algorithm with the remaining features. Then, REF calculates the model performance using a performance metric such as accuracy. If the decrease in the model prediction accuracy trained with the remaining features is higher than a predefined threshold, then that feature is important and should be conserved; otherwise, it can be removed. This process is repeated until it reaches the best k features. RFE ranks the features according to their importance coefficient or weight derived from a machine learning algorithm, such as logistic regression or tree-based approaches. Then, it decides to remove or keep a feature using the prediction performance (e.g., accuracy) of a specific machine learning approach such as k-nearest neighbors or support vector machines.

The RFE method is available via the RFE class in scikit-learn. Let us see how to select the best features of the Iris dataset using RFE method. The steps are demonstrated below:

1. Import the Iris dataset and load libraries.

   Code
   ```
   # import libraries
   import pandas as pd
   import numpy as np
   from sklearn.datasets import load_iris
   from sklearn.feature_selection import RFE
   from sklearn.linear_model import LogisticRegression
   from sklearn.feature_selection import RFECV

   # load sample data
   iris = load_iris()
   x = pd.DataFrame(iris.data, \
      columns=iris.feature_names)
   ```

2. Create an object for the logistic regression method to be used as the model to rank the features.

   Code
   ```
   lr = LogisticRegression()
   ```

3. Configure RFE with the chosen model specified via the `estimator` argument and the number of features to select via the `n_features_to_select` argument. The `estimator` is the logistic regression and the `n_features_to_select` is set to 3.

Code
```
rfe = RFE(lr, 3).fit(x, iris.target)
```

4. Display the obtained results that can be seen in the `support_`, `n_features_`, and `ranking_` attributes of the resulting object. The support_ attribute is “True” when the feature is selected and “False” when it is not selected. The `n_features_` and `ranking_` attributes provide the number of selected features and the feature rankings, respectively. For the Iris dataset, the number of selected features is 3, the selected features are [False True True True], and their ranking is [2 1 1 1]. This means that the first Iris feature (sepal length) is not selected.

Code
```
pd.DataFrame({'features': iris.feature_names, \
   'Selected features': rfe.support_, \
   'Feature ranks': rfe.ranking_})

# console output:
#  features        Selected features Feature ranks
# 0 sepal length (cm) False           2
# 1 sepal width (cm) True             1
# 2 petal length (cm) True            1
# 3 petal width (cm) True             1
```

It is possible to find the number of features to be selected using cross validation. The code above needs to be slightly modified as follows in order to tune k automatically using cross validation. The steps for this are as follows:

1. Import the Iris dataset as a DataFrame as we have done in the code above.
2. Import the logistic regression method to be used as the feature ranking model as we have done in the code above.
3. Set the minimum number of features to consider. We will set it to 3. Configure RFECV with the chosen model specified via the `estimator` argument, the number of features to be eliminated at each iteration specified through `step` argument, the number of folds during cross validations specified through the argument `cv`, and the minimum number of features to select via the `min_features_to_select` argument.

Code
```
rfecv = RFECV(estimator=lr, step=1, cv=5, \
   scoring='accuracy', \
   min_features_to_select= 3).\
       fit(x, iris.target)
```

4. Display the number of selected features `n_features_` and their scores that can be seen in the `n_features_` and `grid_scores_` attributes, respectively. The optimal number of folds for the cross validation is 4, which means that the combination of all Iris features provides the best cross validation accuracy score.

Code

```
# show grid scores for selected featues
print(rfecv.grid_scores_)
# console output: [0.96666667 0.97333333]
```

EFS has the advantage of studying all the potential combinations of the original features in order to find the best combination that provides the best prediction performance according to the selected metric, such as the accuracy rate. However, it has the worst computation complexity, particularly when the number of features is large since the number of combinations to evaluate increases exponentially with the number of original features. SFS and SBS have the advantage of reducing the number of feature combinations to evaluate since they add or remove a feature of a previously selected combination in each iteration. Therefore, they have better computation complexity than EFS, in particular for datasets with a high number of features.

However, SFS and SBS are unable to investigate the usefulness of a feature after being added or removed from the feature set. For instance, SFS may add a feature, which is useful in the beginning and becomes useless when other features are selected. RFE has the advantage of being computationally less complex than SFS and SBS. This is because REF uses feature weight coefficients (when, for instance, the model is a linear model) or feature importance (e.g., when the model uses tree-based approaches) in order to remove the least important feature. Therefore, the latter is removed only once, while SBS removes all the features first in order to determine which one is the least important. For instance, if we have four initial features, SBS removes each one of these features in order to determine its importance on the model accuracy rate; RFE eliminates the least important feature by using the feature importance coefficient or weight derived from its model.

SUMMARY

This unit presented the major feature selection techniques used to rank and select the most relevant features. These techniques improve the performance of the trained model by removing irrelevant and noisy features. Conserving the most relevant features allows us to improve the learning speed, generalization power, processing time, data collecting and storage, and interpretation power of the learned model, particularly when the number of original features is very large.

These techniques were categorized into two major families: filters and wrappers. Feature selection techniques belonging to the filter family select the most relevant features based on some meaningful statistical criteria, such as variance, mutual information, and correlation. Therefore, the filter family has the advantage of being independent from the machine learning approach used to build the model. In addition, filter techniques are computationally efficient since they do not need to train a model. Therefore, they are widely used for real applications with large a number of features. Feature selection techniques belonging to the

wrapper family rank the features using a machine learning method. They have the advantage of considering the interaction between features, determining the optimal subset of features for a machine learning model. Consequently, wrapper techniques have proved to be superior to filter techniques in terms of accuracy. However they suffer from high computational cost. This is because they need to repeatedly train machine learning models.

To bridge the gap between filter and wrapper techniques, hybrid feature selection techniques combining the advantages of both filter and wrappers are proposed. The features are ranked according to their importance using filter techniques such as mutual information, Chi-square test, and feature variance. Wrapper techniques, such as EFS, SFS, SBS, are then used to select the best subset of features from the top k features selected by filter techniques. Hybrid feature selection has the advantage of determining the subset of features, allowing us to obtain better prediction accuracy, more flexibility since different machine learning approaches can be used to build the model, and better computational complexity than wrapper techniques since the number of features is reduced during the filter step.

UNIT 6

AUTOMATED FEATURE GENERATION

STUDY GOALS

On completion of this unit, you will be able to ...

- automatically generate transformation features.
- automatically generate aggregation features.
- highlight the advantages and limitations of the techniques used to automatically generate features.

6. AUTOMATED FEATURE GENERATION

Introduction

The performance (prediction accuracy, learning and processing time, generalization capacity, etc.) of a predictive model heavily depends on the information and predictive power of the features in the dataset used to train that model. Therefore, defining meaningful and useful features is essential to building efficient, predictive models.

The process of defining features manually has several limitations. Firstly, it is a problem-specific task. The features defined for one problem cannot be applied to other problems. Secondly, it is time-consuming since defining features manually requires time and hard work, particularly when the dataset is composed of several related tables. Finally, as with any manual process, the manual feature generation is an error-prone process.

Automated feature generation is an efficient alternative to surpass these limitations. It allows us to automatically generate many features based on the use of a set of related tables. In addition, it can be applied to any dataset describing any problem. There are two techniques to automatically generate features from raw datasets: Deep Feature Synthesis (DFS) and Deep Learning (DL). This unit considers the following questions:

- How are features automatically generated by means of the DFS and DL concepts?
- What are the advantages and limitations of DFS and DL in order to automatically generate features?

6.1 Automated Feature Generation

Automated feature generation allows the reduction of the required time to define features since it automatically creates derived features for any dataset. In addition, as for the case of DFS, it creates interpretable features since they are based on the use of basic mathematical operators, such as min, max, and mean. These simple mathematical operators are called primitives.

Transactional datasets
This term describes datasets which contain data that describe events as a result of transactions, such as orders, invoices, payments, and activity records. The transition data always have a time variable since they describe an online, real-time system.

Featuretools is an open source Python library framework allowing the automatic generation of features. It transforms **transactional** and **relationaldatasets** into adapted feature matrices for machine learning. Featuretools works on a concept known as Deep Feature Synthesis (DFS) (Kanter & Veeramachaneni, 2015). DFS allows us to automatically create multiple features either as transformations or aggregations. Transformations are done to one or more columns on a single table, such as the difference between two columns in one table or taking the absolute value of a column. The table below shows some transformation primitives.

Table 33: Transformation Primitives in the Featuretools Library

multiply_boolean	Element-wise multiplication of two lists of Boolean values
year	Determines the year value of a datetime
day	Determines the day of the month based on a datetime
weekday()	Returns the day of the week from a datetime value. Weeks start on Monday (day 0) and run through Sunday (day 6).
divide_by_feature	Divides a scalar by each value in the list
equal	Determines if values in one list are equal to another list

Source: Sayed-Mouchaweh (2021).

Relational datasets
This term describes datasets that contain data points related to one another. These data points are organized in rows and columns where each row represents one data point that is described by the different columns (attributes). Relational datasets can be spread over multiple "relations," i.e., tables.

Aggregations are achieved using different primitives applied to several tables. Primitives are the basic operations, such as mean, max, and min, applied to the values of original columns, in order to form new features. The table below shows some aggregation primitives.

Table 34: Aggregation Primitives in the featuretools Library

all	Calculates if all values are 'True' in a list
std	Computes the standard deviation which is the dispersion relative to the mean value, ignoring `NaN`,
num_unique	Determines the number of distinct values, ignoring `NaN` values
n_most_common	Determines the `n` most common elements
mean	Computes the average for a list of values
num_true	Counts the number of `True` values
median	Determines the middlemost value in a list of values
max	Calculates the highest value, ignoring `NaN` values
time_since_first	Calculates the time elapsed since the first datetime (in seconds)
sum	Calculates the total addition, ignoring `NaN`

Source: Sayed-Mouchaweh (2021).

The complete list of primitives and their description can be found at ("Primitives List," n.d.). Running the following code displays the whole list of transformation and aggregation primitives in the library featuretools.

Code

```
# import libraries
import pandas as pd
import featuretools as ft

# Remove any limit on the number of columns to display
pd.options.display.max_columns = None

# Remove any limit on the number of rows to display
pd.options.display.max_rows = None

# Display the list of primitives
print(ft.list_primitives())
```

We will explain through an example how to apply featuretools using Python to automatically generate features. The featuretools library is based on three major components:

1. Entities
2. Feature primitives
3. Deep Feature Synthesis (DFS)

In the featuretools library, an Entity is a DataFrame table. Each Entity must have a unique index identifying each row (element, sample, or data point) in its DataFrame table. We can create several entities gathered into an Entityset. An Entityset is a large data structure composed of individual entities and the relationships between these entities. Each relationship links an Entity parent to an Entity child. An Entity parent can have several Entity children.

Feature primitives are basic operations that are used to form new features across one entity or several entities. For instance, we have a table titled Students that includes the following columns or features: Students ID number, Student name, and Date of birth. This table defines an Entity that can be called "Students." A new feature can be generated across this Entity indicating the age of each student given their date of birth. This feature is automatically generated by applying the transformation primitive "AGE" to the feature or column Date of birth. This new feature is computed using one table: Students. Let us say we have another table, "Courses," made up of the following columns: Student ID number, Course name, and Grades. The column "Course name" indicates the name of the course, and "Grades" indicates the grade obtained by this student for this course. A second Entity can be defined based on this table and can be called "Courses." A new feature indicating the mean grade obtained by each student for the followed courses using the aggregation primitive "MEAN." This feature is computed based on two tables: Students and Courses.

These features are called deep features. The depth of these features can be variable allowing to generate **complex features**. For instance, creating a feature using the aggregation operation "Mean" is of depth 1. However, creating a feature as the max of the mean values is of depth 2 since two aggregations are required to generate it, first find the mean values, then the max value of mean values. For instance for the example of "Students," the max of the mean grades obtained by the students generates a new feature of depth 2.

Complex features
These are features based on the use of more than one basic operation. For instance, a feature defined using MEAN and MAX is a complex feature.

DFS is a concept allowing us to automatically generate new features from single and multiple entities (dataframes). DFS generates the deep features by grouping data points in the different entities based on the defined relationships between these entities. The grouping is achieved using the different aggregation primitives.

Let us take an example describing the purchases of customers. Each customer orders a certain number of products and each product has a certain price. This description is achieved by the three tables below. The first table, "Customers," contains the customer ID, name, and account creation date. The second table "Orders" represents the customer order ID. The last table "Payments" represents the amount paid by customers for each order. For reasons of simplicity, we only have two customers.

Table 35: Customer Table

Customer_ID	Customer_name	Creation-date
C1	Martin	2018-08-15
C2	Julia	2020-05-05

Source: Sayed-Mouchaweh (2021).

Table 36: Customer Orders

Order ID	Customer ID
1	C1
2	C2
3	C1
4	C1
5	C2

Source: Sayed-Mouchaweh (2021).

Table 37: Customer Payments

Order_ID	Price
1	500
5	200
3	300
4	100

2	900

Source: Sayed-Mouchaweh (2021).

Let us write the Python code to automatically generate new features for this example using the library featuretools. The steps are demonstrated below:

1. Import featuretools in order to automatically generate features using the DFS concept, and import pandas in order to create the tables as DataFrames.

   Code
   ```
   import featuretools as ft
   import pandas as pd
   ```

2. Create the three tables above for our customer purchases example.

   Code
   ```
   Customers = pd.DataFrame({ \
      'C_ID': ['C1', 'C2'], \
      'Name': ['Martin', 'Julia'], \
      'Creation_date': ['2018-08-15', '2020-05-05']}, \
          columns = ['C_ID','Name','Creation_date'])
   Orders = pd.DataFrame({ \
      'Ord_ID': ['1', '2', '3', '4', '5'], \
      'C_ID': ['C1', 'C2', 'C1', 'C1','C2']}, \
          columns = ['Ord_ID','C_ID'])
   Payments = pd.DataFrame({ \
      'Ord_ID':['1', '5', '3', '4', '2'], \
      'Price':[500, 200, 300, 100, 900]}, \
          columns = ['Ord_ID', 'Price'])
   ```

3. Create an Entityset in order to gather three entities, one Entity for each dataframe table. Since “Customers” is the main Entity in the Entityset, we call the Entityset by the same name.

   Code
   ```
   es = ft.EntitySet(id = 'Customers')
   ```

4. Create the Entity “Customers” to represent the table “Customers” within the framework of featuretools. Indeed, an Entity is a table but is defined in DFS. We can see that this table has a unique index “C_ID” identifying each row in it.

Code
```
es = es.entity_from_dataframe( \
  entity_id = 'Customers', \
  dataframe = Customers, \
  index = 'C_ID', time_index = 'Creation_date')
```

5. Create the Entity “Orders” to represent the table of the same name within the framework of featuretools. This table has a unique index “Ord_ID” that identifies each row in it.

Code
```
es = es.entity_from_dataframe( \
  entity_id = 'Orders', \
  dataframe = Orders, \
  index = 'Ord_ID')
```

6. Create the Entity “Payments” to represent the eponymous table within the framework of featuretools. This table does not have a unique index; therefore, we need to create a unique index by using the `make_index=True` command and then specify a name for the index, such as “P_ID.”

Code
```
es = es.entity_from_dataframe(entity_id = 'Payments',
                             dataframe = Payments,
                             make_index = True,
                             index = 'P_ID')
```

7. Create relationships between the different entities. Each table connection must be established between an Entity parent and an Entity child. We model the Entity “Customers” as the parent of Entity ‘Orders’ since each customer from “Customers” can order several orders in “Orders.” This makes it easier for us in terms of **cardinality**. Similarly, we model the Entity “Orders” as the parent of the Entity “Payments’ since each order can have several payments. This is usually not the case, but for the sake of this simple example, let us assume that we offer deferred payment to our customers so that each order can have several payments. The connection between two entities needs a common column between the two entities in order to link them. This linking column is called the “key field.” We call the linking column of a table in question the “primary key” and the respective column in the linked table the “foreign key.” For the relationship between “Customers” and “Orders,” the key field is “C_ID,” while for “Orders” and “Payments,” the key field is “Ord_ID.” The format for defining a table connection is as follows: `Relationship.(<parent_entity>, <primary_key>, <child_entity>, <foreign_key>).`

Cardinality
The cardinality between two tables describes how the rows in these tables are linked to each other. One row in one table might be linked to one or more rows in another table.

Code
```
# Define the relationship between the parent 'Customers'
# and the child 'Orders' linked together by 'C_ID'
r_Cust_Ord = ft.Relationship( \
   es['Customers']['C_ID'], \
   es['Orders']['C_ID'])

# Add the relationship to the entity set
es = es.add_relationship(r_Cust_Ord)

# define relationship between 'Orders'
# and 'Payments'
r_Orders_Payments = ft.Relationship( \
   es['Orders']['Ord_ID'], \
   es['Payments']['Ord_ID'])

# Add the relationship to the entity set
es = es.add_relationship(r_Orders_Payments)
```

8. We can print out the defined entities and entity connections.

Code
```
                        # show entityset
es
# console output:
# Entityset: Customers
#  Entities:
#    Customers [Rows: 2, Columns: 3]
#    Orders [Rows: 5, Columns: 2]
#    Payments [Rows: 5, Columns: 3]
#  Relationships:
#    Orders.C_ID -> Customers.C_ID
#    Payments.Ord_ID -> Orders.Ord_ID
```

9. You can create new features using specified primitives. You can take a look at the list of transformation and aggregation primitives in the library featuretools by running the following code (we display only 15 transformation primitives and 15 aggregation primitives).

Code
```
# show aggregation primitives
primitives = ft.list_primitives()
pd.options.display.max_colwidth = 160
primitives[primitives['type']=="aggregation"].\
    head(15)

# show transformation primitives
primitives[primitives['type']=="transform"].\
    head(15)
```

10. Create new features using the `ft.dfs` function by passing the transformation primitive "year" and the aggregation primitive "sum" applied to the Entity "Customers" as the target entity. The "year" feature will display the account creation year for each customer. It is a transformation feature since it results from using a transformation primitive. The latter is acting on one table "Customers" by creating a new feature out of one existing column, "Creation_Date." The "sum" feature computes the sum of all the prices of the products ordered by each customer. It is an aggregation feature since it is computed by using an aggregation primitive that is acting on columns, Customer_ID, Order_ID, and Price, from several tables (Customers, Orders, and Payments).

    Code
    ```
    features, feature_names = ft.dfs( \
       entityset=es, \
       target_entity='Customers', \
       agg_primitives=['sum'], \
       trans_primitives=['year'])

    features
    # console output:
    #       Name SUM(Payments.Price) YEAR(Creation_date)
    # C_ID
    # C1 Martin   900                2018
    # C2 Julia    1100               2020
    ```

11. Let featuretools also automatically generate new features using the ft.dfs function without specifying any primitives. To determine the number of generated features, set the max-depth of the generated features, the Entityset, and target-entity used to generate these features. Featuretools then automatically generates many combinations of feature primitives (transformation and aggregation primitives) according to the predefined depth.

    Code
    ```
    feats, feat_names = ft.dfs( \
       entityset=es, \
       target_entity='Customers', \
       max_depth = 2)
    ```

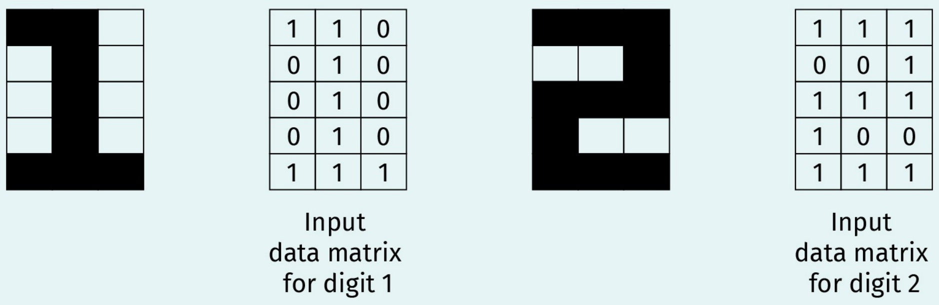
The library featuretools will generate 51 different features, such as `SUM(Payments.Price)`, `MAX(Orders.SUM(Payments.Price))`, or `STD(Payments.Price)` for each customer from the "Customers" Entity. The feature

`SUM(Payments.Price)`, for instance, provides the sum of the prices of products ordered by each customer. Its depth is 1 since it is based on one basic operation, `SUM`. `MAX(Orders.SUM(Payments.Price))` allows us to find the maximum amount paid by each customer for all the purchased products. It is a feature of depth 2 since it is based on two basic operations, `SUM` and `MAX`.

6.2 Feature Engineering versus Deep Learning

Deep learning approaches (Du et al., 2016), such as Convolutional Neural Network (CNN) (Maitra et al., 2015), are commonly used for classification tasks for images, text, and audio. As one part of the to be trained network, kernel filters and pools, i.e., aggregations, are applied to the original input data, usually exposing underlying structures and patterns hidden within the raw input data. These patterns are useful for the distinction between classes; however, they can also be used as input feature to other machine learning models, thereby constituting a technique for automated feature generation. These features are automatically extracted from raw data by matrix multiplication. Let us take the example of digits recognition, 0 to 9. Each digit is represented as a grayscale input matrix of pixels as shown in the figure below.

Figure 49: Input Data Matrix for Digit 1 and Digit 2



1	1	0
0	1	0
0	1	0
0	1	0
1	1	1

Input data matrix for digit 1

1	1	1
0	0	1
1	1	1
1	0	0
1	1	1

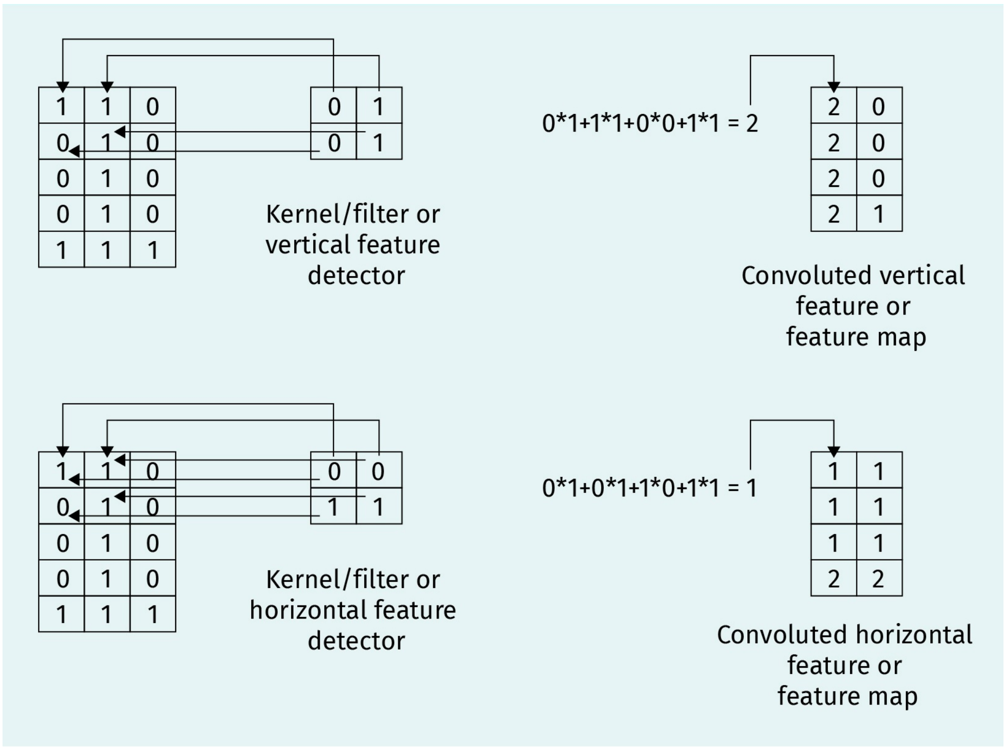
Input data matrix for digit 2

Source: Sayed-Mouchaweh (2021).

CNN uses a kernel or filter to define features that can be used to recognize a digit. For instance, in the figure below, two filters are used to generate a feature allowing detecting vertical lines and another feature to detect horizontal lines. Similarly, other features can be generated, such as left or right diagonal lines.

The multiplication between the input raw data matrix and each filter allows us to generate the feature map indicating if the corresponding feature (vertical or horizontal lines) exists in the input image. In the example below, the value “2” indicates, depending on the applied kernel filter, the existence of a vertical or horizontal line, respectively. The figure below shows how the vertical and horizontal filters are used by CNN in order to build the vertical and horizontal convolved features or features map for digit 1.

Figure 50: Automated Convoluted Features Generation for Digit 1 by CNN

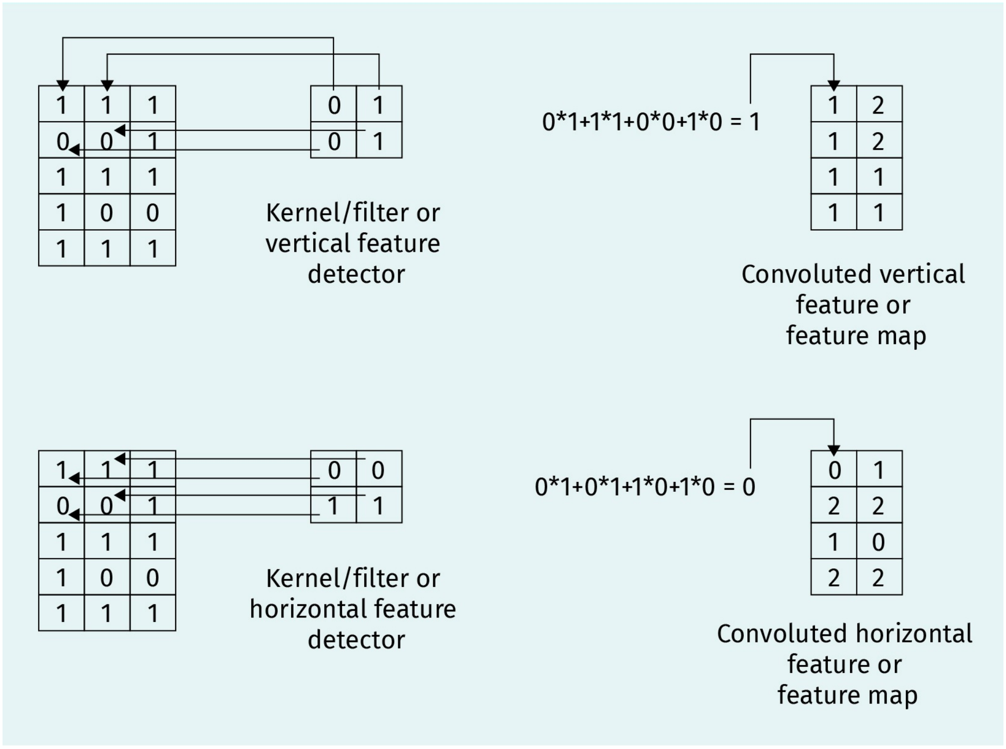


1 1 0
0 1 0
0 1 0
0 1 0
1 1 1
0 1
0 1
Kernel/filter or vertical feature detector
0*1+1*1+0*0+1*1 = 2
2 0
2 0
2 0
2 1
Convoluted vertical feature or feature map
1 1 0
0 1 0
0 1 0
0 1 0
1 1 1
0 0
1 1
Kernel/filter or horizontal feature detector
0*1+0*1+1*0+1*1 = 1
1 1
1 1
1 1
2 2
Convoluted horizontal feature or feature map

Source: Sayed-Mouchaweh (2021).

Likewise, the figure below shows how the vertical and horizontal filters are used by CNN to build the vertical and horizontal convolved features or features map for digit 2.

Figure 51: Automated Convoluted Features Generation for Digit 2 by CNN

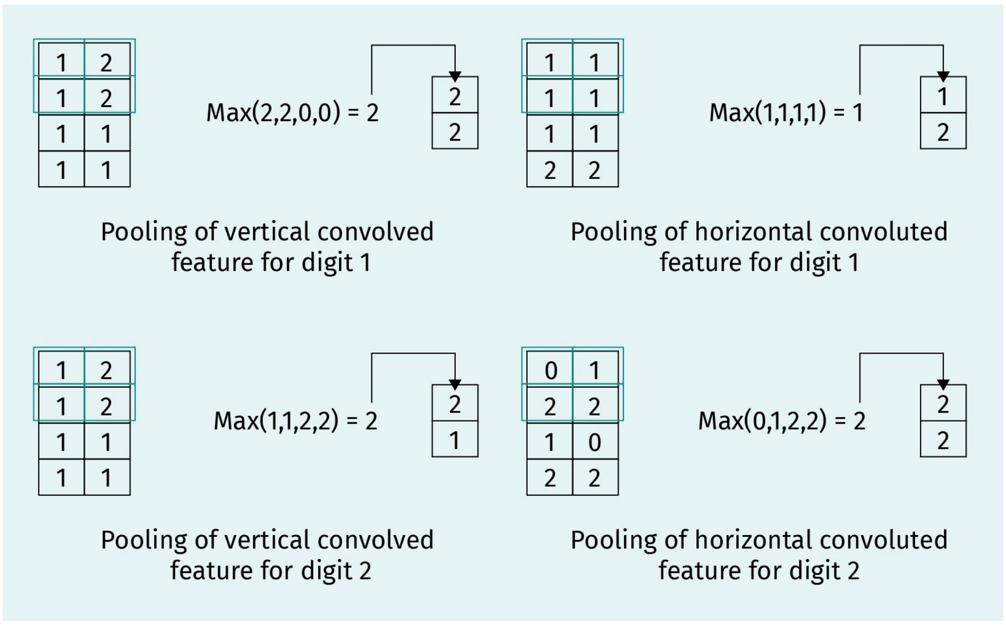

Kernel/filter or vertical feature detector
0*1+1*1+0*0+1*0 = 1
Convoluted vertical feature or feature map
Kernel/filter or horizontal feature detector
0*1+0*1+1*0+1*0 = 0
Convoluted horizontal feature or feature map

Source: Sayed-Mouchaweh (2021).

CNN automatically generates the suitable filters to search for informative features in the raw input matrix. All we need to do is define the number of filters we would like to apply.

Let us see how CNN uses these generated features in order to recognize the digit 1 and digit 2. The vertical and horizontal convoluted features will be pooled with a window, generally of size of 2, and a stride, which is also 2. The goal of the pooling is to shrink the image (input raw data matrix for digits 1 and 2). The figure below shows the pooling of vertical and the horizontal convoluted features for digits 1 and 2. If we put the vertical and horizontal convolved features after pooling into one single list, we have, for digit 1, the list [2 2 1 2] and, for digit 2, the list [2 1 2 2]. Therefore, CNN can distinguish the digit 1 if the second element in the list is equal to 2 and the third element is equal to 1. It can distinguish the digit 2, if the second element in the list is equal to 1, and the third element is equal to 2.

Figure 52: Automated Convoluted Features Generation for Digit 2 by CNN



1 2
1 2
1 1
1 1
Max(2,2,0,0) = 2
2
2
Pooling of vertical convolved feature for digit 1
1 1
1 1
1 1
2 2
Max(1,1,1,1) = 1
1
2
Pooling of horizontal convoluted feature for digit 1
1 2
1 2
1 1
1 1
Max(1,1,2,2) = 2
2
1
Pooling of vertical convolved feature for digit 2
0 1
2 2
1 0
2 2
Max(0,1,2,2) = 2
2
2
Pooling of horizontal convoluted feature for digit 2

Source: Sayed-Mouchaweh (2021).

Although deep learning approaches have been successfully used for many applications, they require a large number of training data samples to learn and train the complex architectures they need to work. Moreover, the extracted features are not interpretable, while the generated features by featuretools are interpretable since they are based on the combinations of simple primitives (e.g., mean, max, and min) that can be easily described in natural language.

SUMMARY

Feature engineering is the process of using domain knowledge to extract features from raw data to improve the performances of machine learning models. Features can be generated manually and automatically. Manual feature engineering is tedious and time-consuming and requires domain knowledge. Automatically generating features is more efficient and faster than manual feature engineering and can be repeated to different datasets belonging to different domain applications. However, the usefulness of the generated features is not guaranteed and some features may be useless and redundant. Therefore, automated feature engineering does not compensate for domain knowledge, expertise, and open and clear communication.

This unit presented two techniques for automated generation, Deep Feature Synthesis (DFS) and Deep Learning (DL). The former was demonstrated by the open Python library featuretools. This library can auto-

matically generate features from a set of related data tables (structured transactional and relational datasets) and can be applied to any machine learning problem. In addition, featuretools can generate features from small datasets.

The Deep Learning technique for automated feature generation was demonstrated by Convolutional Neural Networks (CNNs) which can automatically generate features for image, text, and audio classification. The features are extracted through the multiplication of the input raw data matrix and a set of kernels filters. The generated features are not interpretable and require large datasets.

BACKMATTER

LIST OF REFERENCES

Alashwal, H., El Halaby, M., Crouse, J. J., Abdalla, A., & Moustafa, A. A. (2019). The application of unsupervised clustering methods to Alzheimer's disease. *Frontiers in Computational Neuroscience*, *13,* Article 31. https://doi.org/10.3389/fncom.2019.00031

Băcilă, M. F., Rădulescu, A., & Mărar, I. L. (2012). Prepaid telecom customers segmentation using the k-mean algorithm. *Annals of the University of Oradea, Economic Science Series*, *21*(1), 1112—1118.

Bejani, M., Gharavian, D., & Charkari, N. M. (2014). Audiovisual emotion recognition using ANOVA feature selection method and multi-classifier neural networks. *Neural Computing and Applications*, *24*(2), 399—412.

Bernard, T., Verbunt, M., vom Bögel, G., & Wellmann, T. (2018, May). Non-intrusive load monitoring (nilm): Unsupervised machine learning and feature fusion. *2018 international conference on smart grid and clean energy technologies (ICSGCE)* (pp. 174—180). IEEE. https://doi.org/

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Buja, A., Swayne, D. F., Littman, M. L., Dean, N., Hofmann, H., & Chen, L. (2008). Data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics*, *17*(2), 444—472. https://doi.org/10.1198/106186008X318440

Dasgupta, S. (2019). *Churn prediction of bank customers: Neural network for finding surprising correlations.* [Version 1]. Kaggle. https://www.kaggle.com/sonalidasgupta95/churn-prediction-of-bank-customers

Ding, S., Zhu, H., Jia, W., & Su, C. (2012). A survey on feature extraction for pattern recognition. *Artificial Intelligence Review*, *37*(3), 169—180. https://doi.org/10.1007/s10462-011-9225-y

Du, X., Cai, Y., Wang, S., & Zhang, L. (2016, November). Overview of deep learning. *2016 31st youth academic annual conference of Chinese association of automation (YAC)* (pp. 159—164). IEEE. https://doi.org/

Dy, J. G., & Brodley, C. E. (2004). Feature selection for unsupervised learning. *Journal of Machine Learning Research*, *5* (Aug), 845—889.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databses with noise. In E. Simoudis, J. Han, & U. Fayyad (Eds.), *Proceedings of the second international conference on knowledge discovery and data mining* (pp. 226—231). AAAI Press.

Kanter, J. M., & Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. *2015 IEEE international conference on data science and advanced analytics (DSAA)* (pp. 1—10). IEEE.

Kaufman, L., & Rousseeuw, P. J. (2009). *Finding groups in data: An introduction to cluster analysis* (Vol. 344). John Wiley & Sons.

Maddila, S., Ramasubbareddy, S., & Govinda, K. (2020). Crime and fraud detection using clustering techniques. In H. Saini, R. Sayal, R. Buyya, & G. Aliseri (Eds.), *Innovations in Computer Science and Engineering* (pp. 135—143). Springer.

Maitra, D. S., Bhattacharya, U., & Parui, S. K. (2015). CNN based common approach to handwritten character recognition of multiple scripts. *2015 13th international conference on document analysis and recognition* (ICDAR) (pp. 1021—1025). IEEE. https://doi.org/

Marcílio, W. E., & Eler, D. M. (2020, November). From explanations to feature selection: Assessing SHAP values as feature selection mechanism. *2020 33rd SIBGRAPI conference on graphics, patterns and images (SIBGRAPI*) (pp. 340—347). IEEE. https://doi.org/

Müller, A. C., & Guido, S. (2016). *Introduction to machine learning with Python: A guide for data scientists*. O'Reilly Media.

Ozdemir, S., & Susarla, D. (2018). *Feature engineering made easy: Identify unique features from your dataset in order to build powerful machine learning systems*. Packt Publishing Ltd.

Popat, S. K., & Emmanuel, M. (2014). Review and comparative study of clustering techniques. *International Journal of Computer Science And Information Technologies*, *5*(1), 805—812.

Primitives List. (2021). https://primitives.featurelabs.com/

Satopaa, V., Albrecht, J., Irwin, D., & Raghavan, B. (2011). Finding a "kneedle" in a haystack: Detecting knee points in system behavior. *2011 31st international conference on distributed computing systems workshops* (pp. 166—171). IEEE. https://doi.org/10.1109/ICDCSW.2011.20

Saul, L. K., & Roweis, S. T. (2000). *An introduction to locally linear embedding*. [Unpublished article]. https://cs.nyu.edu/~roweis/lle/papers/lleintro.pdf

Sayed-Mouchaweh, M. (Ed.). (2020). *Artificial intelligence techniques for a scalable energy transition: Advanced methods, digital technologies, decision support tools, and applications.* Springer.

Sewak, M., Karim, M. R., & Pujari, P. (2018). *Practical convolutional neural networks: Implement advanced deep learning models using Python*. Packt Publishing Ltd.

Thanh-Hai, N., Tran, T. B., Tran, A. C., & Thai-Nghe, N. (2020). Feature selection using local interpretable model-agnostic explanations on metagenomic data. In T. K. Dang, J. Küng, M. Takizawa, & T. M. Chung (Eds.), *International conference on future data and security engineering*, (Vol. 1306, pp. 340—357). Springer. https://doi.org/10.1007/978-981-33-4370-2_24

Toubakh, H., Sayed-Mouchaweh, M., Benmiloud, M., Defoort, M., & Djemai, M. (2020). Self adaptive learning scheme for early diagnosis of simple and multiple switch faults in multicellular power converters. *ISA Transactions*. https://doi.org/10.1016/j.isatra.2020.03.025

Westermann, P., Deb, C., Schlueter, A., & Evins, R. (2020). Unsupervised learning of energy signatures to identify the heating system and building type using smart meter data. *Applied Energy*, *264*, Article 114715. https://doi.org/10.1016/j.apenergy.2020.114715

Verkerken, M., D'hooge, L., Wauters, T., Volckaert, B., & De Turck, F. (2020). Unsupervised machine learning techniques for network intrusion detection on modern data. *2020 4th cyber security in networking conference (CSNet)* (pp. 1—8). IEEE. https://doi.org/

LIST OF TABLES AND FIGURES

IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Mailing Address
Albert-Proeller-Straße 15-19
D-86675 Buchdorf

media@iu.org
www.iu.org

Help & Contacts (FAQ)
On myCampus you can always find answers
to questions concerning your studies.