

PYTHON

MACHINE LEARNING

FOR BEGINNERS STARTING FROM
SCRATCH



AHMED PH. ABBASI

AHMED PH. ABBASI

A seamless introduction to the machine learning world

Table of Contents

[Chapter One](#)

[Introduction](#)

[1.1 What is Machine Learning](#)

[1.2 Machine Learning and Classical Programming](#)

[1.3 Machine Learning Categories](#)

[1.3.1 Supervised Learning](#)

[1.3.2 Unsupervised Learning](#)

[1.4 Benefits of Applying Python in Machine Learning Programming](#)

[Chapter 2](#)

[Data Scrubbing and Data Preparation](#)

[2.1 Data Scrubbing](#)

[2.1.1 Noisy Data](#)

[2.1.2 Missing Data](#)

[2.1.3 Inconsistent Data](#)

[2.2 Missing Data](#)

[2.2.1 Missing Data Healing, Sacramento real estate transactions as A Case Study](#)

[2.3 Data Preparation](#)

[2.3.1 Data integration](#)

[2.3.2 Data Transformation](#)

[2.3.3 Data Reduction](#)

[2.4 Cross Validation Using K-fold Technique](#)

[2.4.1 K value selection](#)

[2.4.2 How to fold, using Python](#)

[Chapter Three](#)

[Supervision Learning: Regression Analysis & Classification](#)

[3.1 First: Regression Analysis:](#)

[3.1.2 The Equation of the Linear Regression](#)

[3.1.2 Testing with \$R^2\$ correlation:](#)

[3.2 Classification Using Decision Tree](#)

[3.2.1 Introduction to the Decision Tree](#)

[3.2.2 Decision Tree Construction](#)

[3.2.3 Building and Visualization of a Basic Decision Tree in Python](#)

[Chapter Four](#)

[Clustering](#)

[4.1 The k-means clustering algorithm](#)

[1. Bias and Variance](#)

[1.1 Error due to Bias](#)

[1.2 Error due to Variance](#)

[1.3 Graphical Explanation of the Bias and Variance](#)

[1.4 Trade-Off Between Bias and Variance](#)

[Chapter Four](#)

[Artificial Neuron Networks](#)

[4.1 Introduction](#)

[4.2 Artificial Neural Networks](#)

[4.3 The Flow in the Neural Network](#)

[4.4 activation function](#)

[4.5 Working Example](#)

[4.6 The learning process \(How the weights work\)](#)

[4.7 How the back propagation is performed?!](#)

[4.7.1 Gradient Descent](#)

[4.7.2 Stochastic Gradient Decent](#)

[4.8 Understanding Machine Learning Categories in the Context of Neural Networks](#)

[4.9 Neural Networks Applications](#)

Chapter One

Introduction

1.1 What is Machine Learning

I still remember a story from my first year in primary school named: **“Operation Mastermind”**^[1]. In that story, a master computer controls all the systems in the island. Finally, he decides to get rid of human control and to seize all power himself. He begins to manage the island and its systems depending on his own decisions! Although the current situation of machine is still far of this to happen, people believe that science fiction always comes true!

Human power is normally limited. The heaviest weight ever lifted by a human being was 6,270 lb (2,840 Kg)^[2]. That was a great record compared to the average human power. However, it is nothing when compared to the power of machines, which had been invented by human himself to lift tens of tons of kilograms. This is a simple analogy to the realization of **“machine learning”** power and its capabilities. To imagine the situation, it is known that the data analysis and processing capabilities of a well-trained human is limited in terms of the amount of data being processed, time consumption and also the probability of making errors. On the other hand, the machines/computers designed, built and programmed by human can process a massive amount of data in much less time than human with almost no errors. Besides, electronic machine never takes a break and never let its own opinion affect its analyzing process and results.

To grasp the concept of machine learning, take a corporate or a governmental distributed building for example, in which seeking the optimal energy consumption is the main goal. The building consists of walls, floors, doors, windows, furniture, a roof, etc., which are the general architecture elements of a building. These elements consist normally of different kinds of materials and show different reactions to energy, daylight absorption and reflection. Also, the building encounters different amount of sun radiation, sun positions, wind and weather conditions that varies on hourly basis. Now,

consider that the Energy and Electrical Engineers have decided to construct a photovoltaic system on the building. The optimal design in this case would be when they consider the previous aspects, beside those that are related to choosing the optimal places, orientation, shadowing and angles, considering the directions of the sun on hourly basis for the whole year. Last but not least, the building energy requirements for heating, cooling, lighting, etc. has to be clearly estimated. This is a complex and a massive amount of data considering that this is collected on hourly basis, as mentioned above. What the corporation aspire to achieve is predicting the optimal model of their building design that maximizes the renewable power production and minimizes the energy consumption. This kind of data changes according to changes in time and geographic location, which makes the job very hard for classical ways of programming. Machine learning, on the other hand, is the solution when it is related to variable and large amount of data. The main goal of machine learning is to develop a suitable and optimal algorithm that leads to the best decisions.

1.2 Machine Learning and Classical Programming

It is very common to know a programmer who implements an algorithm via a programming language. The programmer gives the chip/machine specific program commands, containing the input parameters and the expected kind of outputs. The program then runs and processes the data, while being restricted by the code entered by the programmer. This kind of programming does not contain the realization of **“Learning”** which means the ability to develop solution based on background examples, experience or statics. A machine equipped with a learning algorithm is able to take different decisions that is suitable for every situation.

Practically, in machine learning, the computer concludes automatically to an algorithm that can process the dataset to produce the desired output. whereas, the concept is different in classical machine programming. Take, for example, the sorting algorithm. We have already many sorting algorithms that can deal with our inputs and give us a sorted output. Our mission here is just to choose the best sorting algorithm that can do the work efficiently. On the other hand, in machine learning, there exists many applications in which we do not have classical algorithms that are totally ready to give us the desired output. Instead, we have what is called: example data. In the machine learning era, no instructions are given to the computers telling them what to do. The computers have to interact with datasets, develop algorithms and make their own decisions like when a human analysis a problem, but with much more scenarios and faster processing!

1.3 Machine Learning Categories

The categorization of the machine learning algorithms depends normally on the purpose of processing a specific data. Therefore, it is important to identify the learning categories to be able to choose the more suitable way. The machine learning algorithms are generally divided into two main categories; supervised and unsupervised learning algorithms.

1.3.1 Supervised Learning

When a machine has a set of inputs that lead to an output, in which the output nature had been determined by a supervisor, this machine basically follows a supervised learning algorithm. The supervision term does not necessarily mean human intervention. It means that the computer has a target and he needs to create and tune functions that pave the way to this target. This kind of algorithms is also called: **predictive algorithm**. This is because there is an output that is being predicted depending on set of inputs. Being predictive does not only relate to future talk. It also includes the cases when an algorithm infers a current or even a previous event. An obvious example about the present case is the traffic lights, which can be optimally controlled depending on a related dataset following a predictive algorithm. In the past events prediction scenario, for example, doctors can predict the specific date of a pregnancy knowing the mother's current level of hormone.

Common algorithms of supervised machine learning are **Classification** and **Regression**. In classification, the mission is to determine what category a group of data or observations should belong to. The use of classification is very wide in machine learning problems. For example, in cellular communication system, the classifier technique can be used to divide a geographical area into femtocell, picocell or micro cell according to the number of users, nature of the area, number of building, signal fading, etc.

Regression, on the other hand, is employed whenever we want to predict

a numeric data. This way of supervised learning can be utilized, for example, to predict certain test scores, lab results, the price of second-hand cars, etc. Figures 1 and 2 shows examples of classification and regression schemes. Classification and regression algorithms will be discussed later in this book.

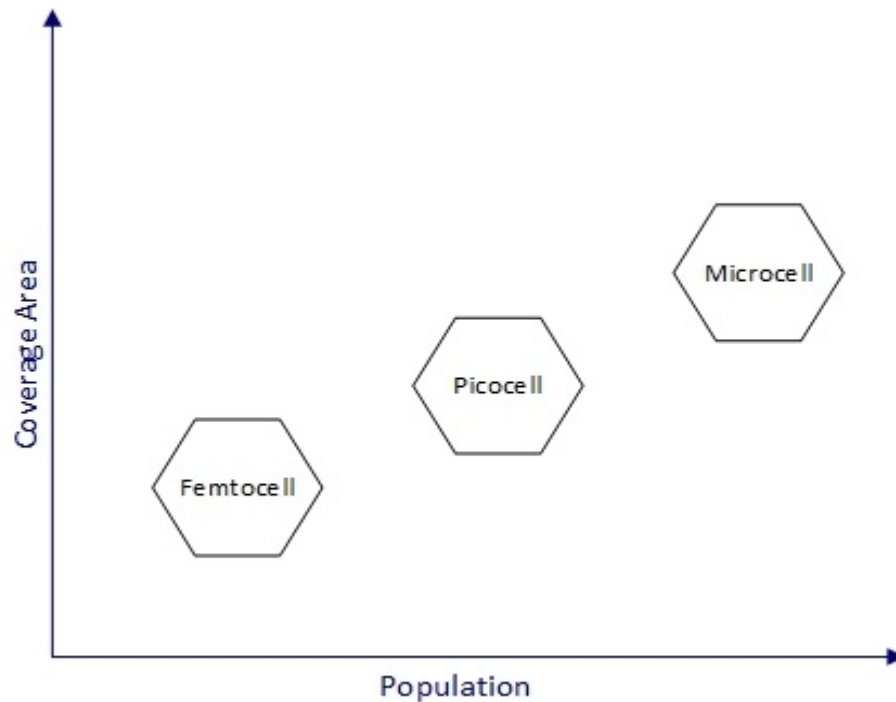


Figure 1.1 Mobile Network classification based on population and coverage area

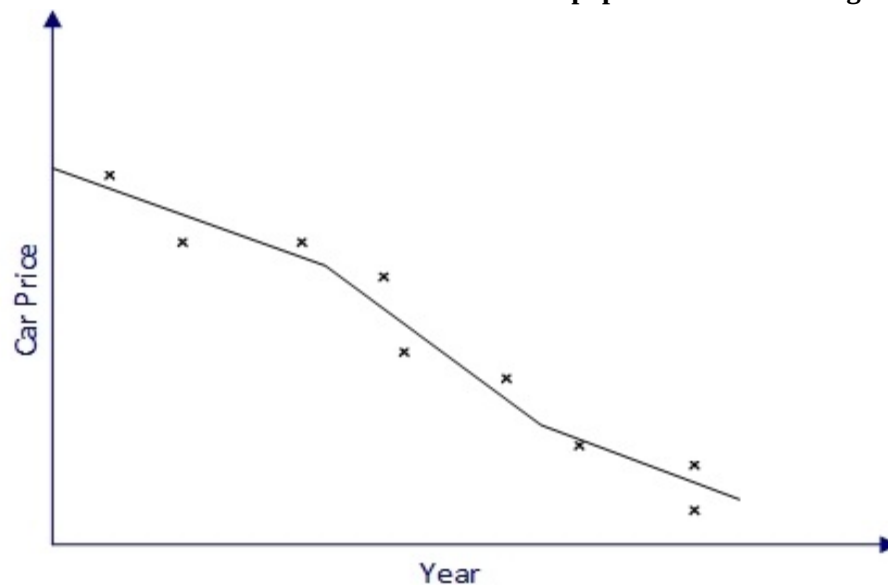


Figure 1.2 Second-handed car price expectation following regression

1.3.2 Unsupervised Learning

It happens that we have a massive amount of data and we want to make a kind of regulation for such an input. This is the main aim of the of the unsupervised learning. A dataset can be processed, following unsupervised algorithm, to tell what patterns happen more frequently than others, what occurs and what does not. This kind of learning models is also sometimes called: **descriptive model**. This is because an unsupervised algorithm makes a summarizing output that describes the data and presents it in a new way. Compared with supervised learning, where a prediction of a target is the main aim, there is not a single feature that is more important than another in the descriptive model. This type of learning is applied in data mining application.

Common algorithms of unsupervised learning are: clustering, density estimation, compressing and data reduction. *Clustering* algorithm is employed to a dataset to have each collection of similar data grouped together. A famous method of clustering is the K-mean algorithm, which will be discussed in details later in this book. The second algorithm, *density estimation*, is applied to a dataset when the goal is to produce statistical output of that dataset. Sometimes the aim is to visualize and present a large dataset information in a summarized and informative way. This brings up the compressing and data reduction algorithms that learn and produce the required summaries without the loss of information. In this algorithm, the key feature is to reduce the dimensionality of a dataset. Reducing the dimensionality of a dataset is also useful in reducing the complexity of other inferring algorithms that may applied later on.

Table 1.1 shows some common supervised and unsupervised machine learning algorithms used to perform the corresponding tasks.

Learning Category	Learning method
-------------------	-----------------

Supervised Learning	
K-Nearest Neighbor	Classification
Decision Trees	Classification
Classification Rule Learners	Numeric prediction
Linear Regression	Numeric prediction
Regression Trees	Numeric prediction
Unsupervised Learning	
Association Rules	Pattern detection
k-means clustering	Clustering

Table 1.1 Common algorithms used to perform supervised and unsupervised machine learning

1.4 Benefits of Applying Python in Machine Learning Programming

For machine learning algorithms, we need a programming language that is understandable and clear for large portion of data researchers and scientists. A language with libraries that are useful for different types of work and in matrix math in specific will be preferable. Moreover, it is of a very good advantage to use a language with a large number of active developers. These features make the arrow point to the Python as the best choice. The main advantages of Python can be summarized in the following points:

- Has clear syntax.
- Makes text manipulation extremely easy.
- A large number of people and communities use Python.
- Possibility of programing in different styles: object-oriented, procedural, functional, etc.
- Ideal for processing non-numeric data.
- Ability to extract data from HTML.
- Common in the scientific and also the financial communities. Therefore, there is a seamless connection between the two fields especially in the machine learning as the financial field is one of the main sources of the datasets.
- Contains a number of useful scientific libraries such as SciPy and NumPy which enables us to perform vector and matrix operations. The installation of Python and also adding these libraries and other, are shown in Appendix A.

Chapter 2

Data Scrubbing and Data Preparation

2.1 Data Scrubbing

Data scrubbing, also called data cleansing, is a very important step before applying any machine learning algorithm. Datasets are normally drawn from real world sources which produce large amounts of messy datasets. Examples of sources are statistics, or massive amounts of records generated from organizations that work in data-intensive fields such as banking, communication systems, insurance, or transportation. These messy data might have noisy entries, missing data or have records that contradict with each other. There are several reasons for why this may happen. Noise usually affects the data because of the hardware limitations and problems, as these can act as noise sources. For example, if a blood sample is tested by a medical device that encounters a problem, the device measurements may be affected and vary on each run for the same sample. If the device is connected to a database via a network, the unstable readings may be transferred automatically to the database regardless of the erroneous that happened. Noisy data can also be generated by human when he makes faults. Missing data problem, on the other hand, may occur due to technical issues, such as a server or a network hang during data transfer, or because of manual data entry. These errors combined; noisy data and missing data problems, may lead also to a third class of messy data, called: data contradiction. Based on that, the data scrubbing step is very important before starting the learning process.

2.1.1 Noisy Data

When there is a difference between the model and the measurements, there exists a “noise” that causes an error or variance. Examples of methods that deal with noisy datasets are as follows^{[3],[4]}: **Binning methods:** In this method, the data is sorted and then smoothed by considering its neighborhood or surrounding values. This is called a local smoothing.

Clustering: This method is useful in detecting and removing the outliers.

Clustering each set of similar values means that the values located in one cluster are different from those in the other clusters. The remaining outliers can then be easily detected.

Machine learning Algorithms: Regression, one of the basic supervised machine learning algorithms, can be used to smooth data by fitting it into regression functions. This algorithm will be discussed in details in chapter three.

Human inspection: Human can interfere to manually detect and eliminate outliers or smooth noise.

In section, 2.3, details on how to deal and eliminate noisy data will be presented.

2.1.2 Missing Data

The followings, are some common ways to deal with the missing data^[5]: **Ignore the tuple:** This method of healing a dataset with missing values is considered efficient, when a record of data contains many attributes with missing values. However, when the amount of missing values per attribute varies significantly, this scheme is no more effective. Efficiency of this method decreases when a regulation of data is produced by an undefined source to the machine. The machine will deal with this kind of data as missing data. For example, when a patient is admitted according to unusual condition.

Determine and fill in the missing value manually: This scheme might have high accuracy. However, it is infeasible mainly in terms of time consumption. Also, it is tiring and tedious.

Making Expectations: Usually, there are ways that one can use to predict a missing attribute. For instance, the average of the values nearby of the missing values. Although this way can cause a bias in the data because of mis-predictions, but it can be utilized to check and compare its results, to the results obtained by the first method, ignoring the tuples.

Inferring-based algorithms: This kind of algorithms are employed when a missing value is filled by the most probable value. Examples algorithms are:

Bayesian formula and decision tree.

In the next section, 2.2, more details about missing data sources, what are the possible solutions for handling missing data and a practical step-by-step example of a real estate transactions dataset with missing values will be presented and solved using Python.

2.1.3 Inconsistent Data

The real life massive amounts of data are susceptible to contain discrepancies or duplications in codes, names or any other values. The inconsistency caused by data entry can be fixed manually. Also, some expert system tools can be adopted to detect and fix contradicts in data. For example, data entered by thousands of individuals, like in health care organizations, are automatically recorded by cloud stored databases. This type of datasets is an obvious case of datasets that are full of inconsistencies. For instance, the same information may be entered in different formats by different sources.

2.2 Missing Data

Most of statistical models cannot be processed unless they are complete without lost or missing variables. Therefore, it is of great importance to solve the missing data problem, by deleting incomplete variables or fill those missing with estimated values, as indicated in the subsection 2.1.1.

First, it is important to specify if the missing data is really missing data. i.e., you need to relate the current available data to its attribute and determine if it makes sense or not. Another important step is to identify the source that causes missingness, as it leads to the best choice of the suitable imputation technique. A value can be missing because: it was forgotten, lost, did not match the instance or it was of no interest relevance importance to the instance.

In this section, we will figure out how to deal with missing data using Python, which is the programming language that will be used throughout this book. Therefore, it is important to refer first to appendix A, where notes are presented on how to install Python and the packages needed to go further in the coming example.

2.2.1 Missing Data Healing, Sacramento real estate transactions as A Case Study

The main aim of this example is to illustrate how to:

- Mark the corrupted or deleted values as: missing.
- Delete records with missing data from the dataset.
- Fill the missing values with the average of the nearby values in the dataset.

First: Downloading and checking the dataset:

The dataset file used in this example, is a real estate dataset that contains a list of 985 real estate transactions in the Sacramento area reported over a five-day period, as reported by the Sacramento Bee^[6]. The dataset .csv file can be downloaded from the link provided in the margin below^[7].

This dataset contains 985 record observations. The first step to process any

dataset is to understand it. Here, we have eleven variables, the longitude and latitude are considered as a one joint variable. The variables declaration is as follows:

- 0. Street name and number.
- 1. city.
- 2. Zip code.
- 3. state.
- 4. Number of bedrooms.
- 5. Number of bathrooms.
- 6. The estate area in square feet.
- 7. The real estate type.
- 8. The sale date.
- 9. The price of the estate
- 10 & 11. Location (Latitude and Longitude).

This dataset contains some missing values. We can print sample data records out of this dataset, following these steps and Python code:

- 1- Open IDLE Python, can be accessed from the start menu, then typing: “Python”. Choose the “IDLE (Python 3.6)” result.
- 2- Create a new file and save it in your work directory, e.g. Name here is: Real_estate_transactions.py
- 3- Place the real_estate_transactions.csv file in the same working file directory contains the previous Python file.
- 4- In the Real_estate_transactions.py file type the following code:

```
from pandas import read_csv
dataset = read_csv('real_estate_transactions.csv', header=None)
# print the first 10 rows of data
print(dataset.head(10))
```

The output is as follows:

	0	...	11
0	3526 HIGH ST	...	-121.434879
1	51 OMAHA CT	...	-121.431028
2	2796 BRANCH ST	...	-121.443839
3	2805 JANETTE WAY	...	-121.439146
4	6001 MCMAHON DR	...	-121.435768
5	5828 PEPPERMILL CT	...	-121.327813
6	6048 OGDEN NASH WAY	...	-121.351705
7	2561 19TH AVE	...	-121.481367
8	11150 TRINITY RIVER DR Unit 114	...	-121.270555
9	7325 10TH ST	...	-121.442979

If we look at the variables in this dataset, we find that the main variables are 4,5,6, and 9. The number of bedrooms, number of bathrooms, the area in square feet and the estate price, respectively. These attributes cannot contain zero values, as zero is considered invalid or lost data. Therefore, in the following step we are going to check the number of zeros in each of these attributes.

Second: Marking the missing values:

The second step is to identify and mark the missing values. A summary statics can be printed to tell us the number of zeros in each attribute. As said, there are columns/attributes in which zero values do not make sense and considered as an invalid or missing data. First, we mark every missing data, i.e. zero number, as True. Then we count the number of “Trues” per column. The following piece of python code performs this task:

```
from pandas import read_csv
dataset = read_csv('real_estate_transactions.csv',
header=None)
# print the first 10 rows of data
print((dataset[[4,5,6,9]] == 0).sum())
```

Code 0-1: Printing the number of missing values

The output summary statistics on each attribute, sum of zeros/missing data, is given as:

4	108
5	108
6	171
9	0

It can be noticed from the output above that there are no zero values in the attribute number 9. However, there are a lot of zero values in the 4,5, and 6 attributes. This means that our initial assumption about the invalidity of having zero values in these records is true. Since this dataset contains real values about real estates, the best way to deal with the missing values problem is to delete the empty records. Although we will first apply the deletion method, other ways will be applied later on for learning purpose. In general, whenever the deletion method is applied, it is important to pay attention for the amount of records that are sufficient to build a predictive model, and that is why, in some cases, deleting large amounts of data is considered un-preferable solution.

We now need to place “marks” instead of the zero values to indicate that there is a missing value in that specific place. To do that, we can utilize the packages we installed to Python, see Appendix A, to place a NaN instead of each zero value. The benefit of this operation is that the NaNs are usually ignored by operations on the dataset such as count and sum. The following code does the process of replacing, using the function: *replace()*. Then, we print the output to check our work. The function: *isnull()* is used to count the number of NaNs.

```
from pandas import read_csv
import numpy
dataset = read_csv('real_estate_transactions.csv',
header=None)
# mark zero values as missing or NaN
dataset[[4,5,6,9]] = dataset[[4,5,6,9]].replace(0, numpy.NaN)
# count the number of NaN values in each column
print(dataset.isnull().sum())
```

The *numpy* package has been imported in the previous example to be able to use “*isnull()*” and hence be able to count the number of NaNs. To check if the zero values have been successfully replaced by NaNs, we compare the following output with the previous one, the number of zeros. It can be easily noted that they have the same numbers. The printed-out output is as follows:

0	0
1	0
2	0
3	0
4	108
5	108
6	171
7	0
8	0
9	0
10	0
11	0

Note: Additional investigation can be done by printing a portion of the new marked dataset, for example: the records from 80 to 100, using the command line:

```
print(dataset.loc[80:100, [4,5,6,9]])
```

This should show that some records have been replaced by NaNs, as follows:

	4	5	6	9
80	3.0	2.0	1478.0	231477
81	3.0	2.0	1287.0	234697
82	4.0	2.0	1277.0	235000
83	4.0	2.0	1448.0	236000
84	4.0	3.0	2235.0	236685
85	3.0	2.0	2093.0	237800
86	3.0	2.0	1193.0	240122
87	3.0	2.0	2163.0	242638
88	3.0	2.0	1269.0	244000
89	NaN	NaN	NaN	244500
90	3.0	1.0	958.0	244960
91	5.0	3.0	2508.0	245918
92	3.0	2.0	1305.0	250000
93	4.0	2.0	1591.0	250000
94	2.0	2.0	1326.0	250134
95	3.0	2.0	1843.0	254200
96	4.0	2.0	1921.0	254200
97	5.0	3.0	2790.0	258000
98	3.0	2.0	1541.0	260000
99	3.0	1.0	1018.0	260014
100	NaN	NaN	NaN	263500

Third: Removing records with missing values

Code 3: Test the dataset with Linear Discriminant Analysis

```

from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
dataset = read_csv('real_estate_transactions.csv', header=None)
# mark zero values as missing or NaN
dataset[[4,5,6,9]] = dataset[[4,5,6,9]].replace(0, numpy.NaN)
# split dataset into inputs and outputs
X = dataset.iloc[:,4,5,6,9].values
y = dataset.iloc[:,12].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

Having a

dataset with missing values may limit the ability to apply machine learning algorithms on it and may produce errors. To clarify this problem, before proceeding to the records removal, we apply the Linear Discriminant Analysis (LDA) algorithm on the dataset. This algorithm does not work if the dataset contains missing values. The following code is a Part of a classification step. At the end of the code we check if the LDA algorithm had been applied or not by printing out the values of X_test set.

As dictated, because of the missing values, the python platform shows a pop-up message like in the following shot, or may show red lines telling you that something was wrong which is preventing the print out process of any output.

Unexpected EOF while Parsing!

The previous code marks the missing values in the dataset, like what had been done before, then it was supposed to calculate the LDA. However, The algorithm failed to work because of the NaNs existed in the dataset. Therefore, a

method to handle the missing variables has to be applied. At this stage, we apply a removal scheme, which is the simplest way to deal with missing data, by deleting records with missing data.

Code 4 Removing missing data records

```
from pandas import read_csv
import numpy
dataset = read_csv('real_estate_transactions.csv', header=None)
# mark zero values as missing or NaN
dataset[[4,5,6,9]] = dataset[[4,5,6,9]].replace(0, numpy.NaN)
# drop rows with missing values
dataset.dropna(inplace=True)
# removes the number of rows and returns to the dataset
```

Utilizing the Pandas packages, we use the *dropna()* function to eliminate records, columns or rows with missing values. This can be done by the following code:

The resulted dataset out of this code has all the records containing NaN deleted. Therefore, a very large cut is done. The resulted dataset is of size (814, 12), which has been clearly reduced to be less than the 985 records in the original dataset. However, we can now apply the LDA algorithm to the new reduced dataset, using the *code 3* sequence, but adding the command line:

```
dataset.dropna(inplace=True)
```

before the command: `X =dataset.iloc[:,[4,5,6,9]].values`. This time, the program will print out the first 5 records of `X_test`, as follows:

```
[ [-0.25247425]
 [ 1.3618153 ]
 [-0.40772602]
 [-0.39940427]
 [-0.29251235]]
```

Fourth: Imputing Missing Values

The deletion process of records with missing values reduced significantly the size of the dataset. Therefore, other imputing methods may be used to replace the NaN/zero values with appropriate values. Some of the common

methods to do that, are as follows:

- 1- A value from another randomly selected record.
- 2- A constant value which makes sense within the domain.
- 3- A mean or mode value for the record.
- 4- A value imported from another model, in which some of its values can be utilized to solve the missing data problem in the current dataset.

```
from pandas import read_csv
import numpy
dataset = read_csv('real_estate_transactions.csv', header=None)
# mark zero values as missing or NaN
dataset[[4,5,6,9]] = dataset[[4,5,6,9]].replace(0, numpy.NaN)
# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# count the number of NaN values in each column
print(dataset.isnull().sum())
```

In this example, we will replace the missing data with the mean value of each column. The “*fillna*” function, provided by the Panda package, is applied for this task, as in the following code:

Code 5 Filling missing values with mean of the column

The code is then counts the number of the missing values, if there is, after updating the dataset with the mean values. The following there are no more missing values in the dataset.

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0

It is left as exercise for the reader to use the updated dataset, with mean values, in the LDA algorithm code to test for dataset validity.

Good to mention that not all algorithms fail to deal with datasets with missing values. Some algorithms do have the ability to overcome the problem of missing data existence, such as K-Nearest Neighbors, which ignore distance measurements for column with missing values. Classification and Regression consider a missing value as a unique value when building a predictive model. In the next chapter, classification and regression algorithms will be discussed in details.

2.3 Data Preparation

In the previous two sections, we talked about the first step in data preparation, which is data scrubbing. However, further preprocessing steps have to be applied to the data, before heading to the learning process. In this section we describe briefly the common preparation methods.

2.3.1 Data integration

In this process, the data derived from multiple resources, such as different datasets or distributed records, that belongs to a one place is gathered and integrated into one dataset. During the data integration process, some considerations have to be taken into account. For example, for the same real-world entity, attribute values from several data sources may be different. This happens because of different representations or scales, e.g., metric vs. British units. Moreover, data sources can sometimes refer to different standards. For instance, some variables can be referred by means of different IDs, in more than one source.

An important problem that may appear when integrating data from different sources is the redundant data. The common reasons are:

- There exists an attribute that is a derived attribute in another dataset, e.g., annual revenue.
- The same attribute has different names in different datasets.

A correlation analysis can be applied to solve the redundant data problem. The accurate data integration produces a dataset with a reduced number of redundancies and inconsistencies. This is of great benefit as it enhances the learning speed and quality. Data integration step, along with the previous data scrubbing steps, produces a dataset with reliable entries.

2.3.2 Data Transformation

Sometimes it is more convenient to deal with dataset in a specific format, rather than another one. Here where the data transformation comes to the picture. The goal of the data transformation is to convert the data values into other format or unit that makes the data analysis more meaningful (e.g. transform non-linear model into a linear one). Depending on the nature of the analysis required, it is decided whether a transformation is needed or not. Common transformation ways are:

- **Aggregation:** It is a way of summarization, in which two or more data values of the same attribute can be aggregated into one value. For example, multiple categories can be grouped into one category.
- **Normalization:** This way of data transformation is followed when we want to express a group of data using a specific range. i.e., the data is scaled to fall within a small, specified range. This way is used, for example, in the communication systems where the Engineers prefer sometimes to express the base station's power in a normalized power form. They scale the power values to lie within the range of 0 and 1. Common normalization ways are: 1) min-max normalization, 2) z-score normalization, and 3) normalization by decimal scaling. Choosing whether to normalize the data or not and the best ways of normalization depends on the nature of data you are dealing with.
- **Generalization:** It can be considered as another way of aggregation, from the summarization concept view. The idea is that the attributes that fall in low level are transformed to higher-level attributes. This process is also called: the hierarchy climbing.
- **Attribute construction:** To simplify analysis, new attributes can be extracted from the already given attributes. This is what meant by attribute construction.

2.3.3 Data Reduction

The final step of data preparation is the data reduction. The aim of any

dataset analysis is to get meaningful outputs that describe the large amount of dataset's information, in simple words, or even in numerical values that can be then employed in the applications of interest. Large datasets analysis is complex and may take very long time. Sometimes, infeasible results may be produced. That is why the principle of data reduction arises. The input data is reduced by means of a more effective representation of the dataset without affecting the accuracy of the original data. Data reduction makes data easier to be analyzed by removing noise and the redundant records, so that, the task of the learning algorithms becomes more effective. The data reduction step has some pros and cons that have to be taken into account:

- **Pros:** Decreases the data complexity, focus on the most important attributes, reduces computational complexity of many algorithms and makes the results easier to be understood.
- **Cons:** Depending on the nature of the dataset, the data reduction may not be needed. Another main disadvantage is that some of the main data may be thrown during the reduction process, such as multiple values being represented as an average value.

Last words that can be said about the data preparation step is that there are three basic rules which enable us to judge if a dataset is clean and prepared to be processed by a machine learning algorithm or not. 1) Each variable forms a column, 2) Each observation forms a row and 3) Each value has its own cell. Clean and prepared dataset are more easily to be visualized and processed by statistical analysis.

2.4 Cross Validation Using K-fold Technique

Cross-validation is a statistical step applied on a data sample in order to check the validity of a machine learning model. In machine learning, the cross-validation technique is mainly utilized to evaluate and expect the performance of a machine learning model. The benefit of using this method is to exploit a small sample of data to expect how the machine learning algorithm is going to behave when applied to a new data that has not been used in the model training step.

In the K-Fold cross validation technique, a dataset is divided into a testing and a training data. The dataset is partitioned into k folds of approximately equal size. The first fold is considered as a testing set, whereas the remaining $k - 1$ folds are the training sets. The k -fold cross validation method is preferred over other validation method, since It is: 1) Easy to implement. 2) Easy to understand. 3) Its results have lower bias than other methods.

The k parameter represents the number of sets that the dataset is to be divided into. That is why this method got the name K-fold. When a certain value of k is selected, it may replace the k so the name becomes: for example, 10-fold cross validation if we choose k to be 10. Now, the problem in the division process is that each fold, testing and training folds, seeks to contain the max. number of data points, as seen in figure X.X. when the number of data points in the testing set increase, the validation becomes better. Similarly, for the training set in which we get better learning results, when the number of its data points increases. Therefore, a trade-off exists between the both of sets. So that, each data point taken out of the training set into the testing set is a lost for the training set, and vice versa. This is where the cross-validation comes to the seen.

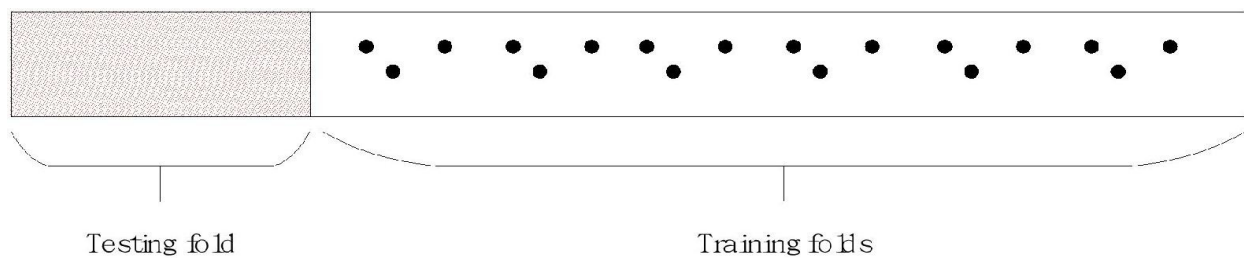


Figure X.X splitting dataset into testing and training folds

The basic idea is to partition the dataset into K sets of equal sizes. For example, if we have a dataset of size 400 and $k = 10$ folds, then the number of data points in each of the 10 folds is 40. As seen in figure X.XX, we pick one of the folds as a testing fold and the other then as a training fold. In K -fold cross-validation, we run k separate learning experiments. In each of those k subsets, we pick one as a testing set. The remaining $k-1$ sets are put together into the training set. After that, we train out a machine algorithm and then the performance is tested on the testing set.

The key feature in cross-validation is that we run the algorithm multiple times, 10 times in the previous example. Then we average the ten different testing set performances for the ten different hold-out sets. i.e. we average the test results from those k experiments. This obviously takes more computing time since we have to run k separate learning experiments, but the evaluation of the learning algorithm will be more accurate. Moreover, all data have been used for training and also for testing. This means that each sample of data is given the chance to be used in the testing set 1 time and used to train the model $k-1$ times.

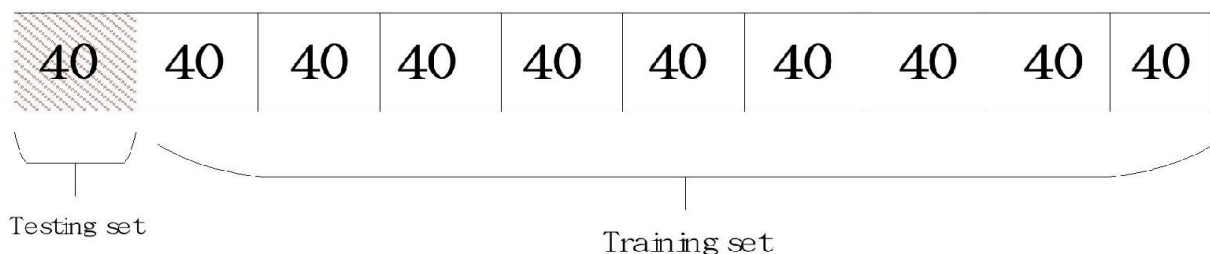


Figure X.XX 10-fold cross-validation example

2.4.1 K value selection

The value of k has to be chosen carefully to avoid having unclear idea of the machine learning model performance. The followings are common ways to determine the value of k :

- 1- Choosing the value of k in a way that guarantee that every train/test set is sufficiently large to represent, statistically, the original full dataset that has to be processed.
- 2- It is found, from empirical results, that choosing $k = 10$ folds gives model performance predictions with low bias.
- 3- Choose the value of k to be equal to the dataset size. This gives each testing set the chance to be used as a training set and vice versa. This way of cross validation is known as: leave-one-out.

The k value choice affects the algorithm bias and variance. In other word, the larger is the value of k , the smaller is the size difference between the training group and the resampling sub-groups. The algorithm's bias reduces as that difference becomes small. This means that there exists a trade-off between choosing the value of k and the algorithm bias and variance behavior.

2.4.2 How to fold, using Python

Suppose that we have the following set of observations:

[0 2 4 6 8 10 12 14 16 18]

In order to divide this set, we need first to choose a value for k . for this example, we will choose a value of $k=5$. This means that we are going to divide the data into 5 partitions, in which each group will contain 2 observations, as follows:

- 1- Fold#1: [0 2]
- 2- Fold#2: [4 6]
- 3- Fold#3: [8 10]
- 4- Fold#4: [12 14]
- 5- Fold#5: [16 18]

Therefore, with each fold, five models are trained and tested. Each fold has the chance to be a testing fold for the 5 separate learning experiments. That is, if we set fold#1 as testing set, then we have folds#2:5 as training set. In the next run, we have, for example, fold#2 as the testing set, so that, folds#1 & folds#3:5 are the training set, and so on. Now, these steps can be simply programmed using Python, utilizing the scikit-learn library. The following code performs the previously explained steps:

Code 6: 5-Fold data division

```
# 5-fold sample division
from numpy import array
from sklearn.model_selection import KFold
# data
sample = array([0, 2, 4, 6, 8, 10, 12, 14, 16, 18])
# Setting-up the folds cross validation
kfold = KFold(5, False)
# Divisions list
```

In the previous code, the KFold command was imported from the scikit-learn library. It took two arguments: The value of k, 5; and if a shuffle will be performed to the data. In our case we set the shuffle option to “False”. If a shuffle is to be done, means “True”, a third argument has to be added to indicate the seed of the pseudorandom number generator. The “split” function is called repeatedly to return the groups of testing and training sets on each iteration. The output of the five folding process is as follows:

```
training: [ 4  6  8 10 12 14 16 18], testing: [0 2]
training: [ 0  2  8 10 12 14 16 18], testing: [4 6]
training: [ 0  2  4  6 12 14 16 18], testing: [ 8 10]
training: [ 0  2  4  6  8 10 16 18], testing: [12 14]
```

Output: Five folds of the data sample

It is advisable to use the *KFold* to partition a dataset before modeling. All of the models shall then exploit the same partitions of the data. This is of main

concern if we are processing massive amounts of data. Moreover, the employment of the same data partitions across the models is good for future statistical work that may be done later on the data.

Chapter Three

Supervision Learning: Regression Analysis & Classification

3.1 First: Regression Analysis:

Regression is a supervised machine learning algorithm. In regression, given an input, when the output is a numeric value, what we would like to learn is not a class, $C(x) \in \{0, 1\}$, but is a numeric function. Regression is exploited to derive equations that best describe the data. The goal is then to utilize this model to generate estimations. The common and the simplest way of regression is the linear regression. The linear regression is preferred to find a predictive function when we have a correlation coefficient indicates that the data can predict upcoming events. Moreover, if a scatter plot of data seems to be a straight line, the linear regression is also preferred.

Linear regression is a method to tell how two variables are related. Recalling from algebra, the famous line equation is $y = mX + b$. This is the famous line equation from elementary algebra. However, we can describe the analogy from the machine learning and linear regression concepts perspective. We have: y as a variable that is dependent, a function of, another variable, X as an independent variable, m is the slope and b is the Y -intercept. We need first, to find the linear regression equation, to decide how the two variables, X and y , are related. In a dataset, we normally have a list of values in columns format, which can be filled as the X and y values.

Note, before proceeding to the regression equation: when we say that there is a relationship between two variables, this does not necessarily mean that one is the cause of the other. Therefore, it is recommended to firstly, generate a scatter graph in order to decide if the data somehow makes a line prior to calculate the equation of the linear regression.

3.1.2 The Equation of the Linear Regression

The formula of the linear regression equation is: $y = mX + b$ (1)

Which is the famous linear equation formula. The slope, m and the intercept, b are calculated from the following two equations, respectively:

$$m = \frac{n \sum xy - \left(\sum x * \sum y \right)}{n \sum x^2 - \left(\sum x \right)^2} \quad (2)$$

$$b = \frac{\left(\sum y * \sum x^2 \right) - \left(\sum x * \sum xy \right)}{n \sum x^2 - \left(\sum x \right)^2} \quad (3)$$

Where, ⁿ is the number of records. Now, to understand how this work, we suppose that we have the following data, shown in the table X.X below:

	X	Y
	45	91
	23	67
	27	81
	41	70
	49	83
	60	77

Table X.X Age and Glucose relationship The dataset above relates two attributes; X: is the age and Y: is the Glucose level. To make the evaluation easier, we make the following table X.XX:

	X	Y	XY	X²	Y²
	45	91	4095	2025	8281
	23	67	1541	529	4489
	27	81	2187	729	6561
	41	70	2870	1681	4900
	49	83	4067	2401	6889
	60	77	4620	3600	5929
Σ	245	469	19380	10965	37049

Table X.XX Age and Glucose relationship Our mission now is to use the values in the table above to calculate the slope and the intercept from equations (2) and (3), respectively. The values of ^m and ^b are found to be: - ^m = 0.2385

$$b = 68.428$$

Now we insert these values in the linear regression equation, which represented as follows: $y = 0.2385X + 68.43$ (4)

Equation (4) is known as a regression equation. The values 0.2385 and 68.43 are called regression weights. The process of calculating these weights is known as regression. Now, the algorithm's learning realization is that once we calculated the regression weights, predicting new values given a dataset is trivial. All we have to do is to multiply the inputs by the regression weights and add them together to get a prediction. To grasp what have been done, you may export this table into an excel sheet, then generate the scatter plot of dataset in table X.X. Then, the linear equation is plotted. The resulted plot is as follows, in

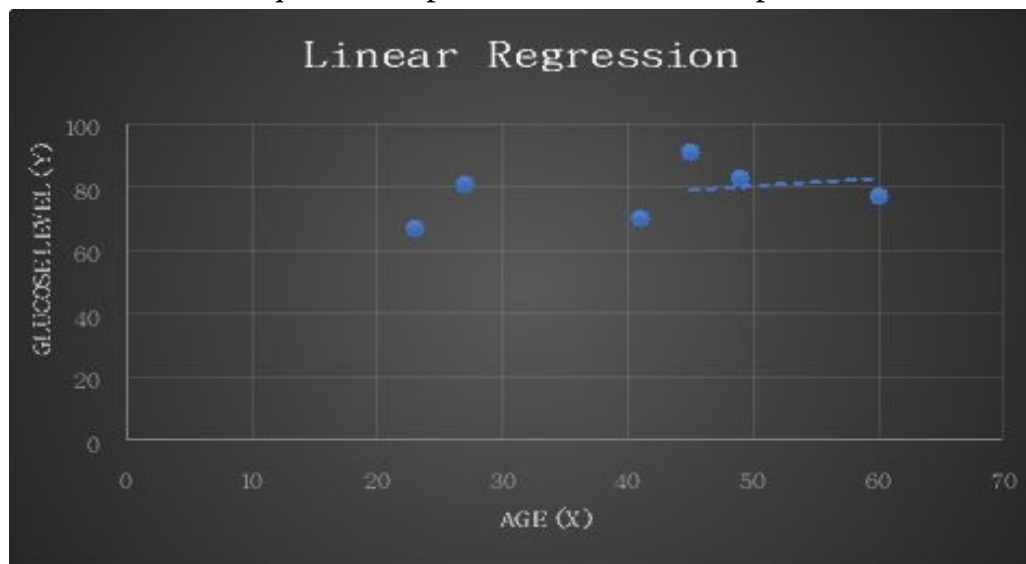


figure X.X:

Figure X.X The scatter data and the linear regression equation plot The line generated by this equation is called: the trend line. An online linear regression calculator is available on the link in the lower margin^[8].

3.1.2 Testing with R^2 correlation:

Correlation is an estimate of how close the trend line to fit the data points that you have in your dataset. In other words, an estimation of the success of the model. It can be somehow estimated by eye from the scatter plot, as we mentioned before, but to make an exact decision we utilize the correlation coefficient; named: R^2 . Normally, the closer R^2 is to 1, the better the line fits the

data points. If R^2 is far from 1, the line definitely, will not fit the data. The correlation factor equation is:

$$R^2 = \frac{\left(n \sum xy - \sum x \sum y\right)^2}{\left(n \sum x^2 - \left(\sum x\right)^2\right)\left(n \sum y^2 - \left(\sum y\right)^2\right)} \quad (5)$$

Utilizing this equation helps engineer or data scientist to make an exact expectation of the accuracy of applying the regression algorithm.

3.2 Classification Using Decision Tree

A lot of resources discuss the classification and concentrate on the K-nearest neighbor method. We will talk about it briefly whenever we need it in this book. However, in this chapter that cares mainly about the supervised learning we are presenting deeply the decision tree classification as an example of the supervised machine learning algorithms

3.2.1 Introduction to the Decision Tree

Have you ever accessed to the “Ask Akinator” online game? For those who do not have a close understanding of the datasets and the data science concept, this game will appear like a kind of magic or paranormal power! In this game, you have to think about a specific character. Then, the Blue Genni will start to ask you a series of questions like: “Is your character a YouTuber”? “Does your character really exist?” and so on. What the game do is a successful splitting of the set of characters it can deduce. If you repeat the game for different characters you will notice that things are working in a branching principal. i.e., every answer leads to a specific set of questions that shrink the circle around the possible answer and cancel a new portion of possibilities. For example, if you answer the first question with “Yes”, the game will exclude the Non-YouTuber persons from its potential decision. The decision tree technique works just like the Akinator game; you give it a set of data and it generates answers to the game.

The decision tree is a famous and a widely used classification technique. The decision tree concept is simple and the “How it works” can be illustrated even to people with almost no knowledge about the machine learning. You may have already seen a decision tree application without knowing it. In figure X.XX a flow chart example of the decision tree. You can notice that the flow goes in a branching manner, where finally you reach to a decision. In this particular example, the algorithm checks for the domain of the sent email and classifies it depending on its domain. This example is a hypothetical system. So that, if the domain is equal to MyOffice.com, the algorithm will classify it as “Urgent”. If it

is not, it makes another check to ensure if the email includes the word “University”. If the text contains the word “University”, then this email is classified as “Important”. If not, the algorithm checks for the word: “Advertisement”. If the email contains it, the email will be classified as “Spam”. Else, it will end to the category: “To be Read later”.

In this section, our decision tree algorithm will have the ability to deal with the input datasets and draw, implicitly, a tree like shown in figure X.XX. The decision tree makes a good job of translating general data into a useful knowledge. The decision tree machine learning algorithm creates rules that classify the unfamiliar datasets. Decision trees are common in expert systems, and they give results that are comparable to those from a human expert with decades of experience in a given field.

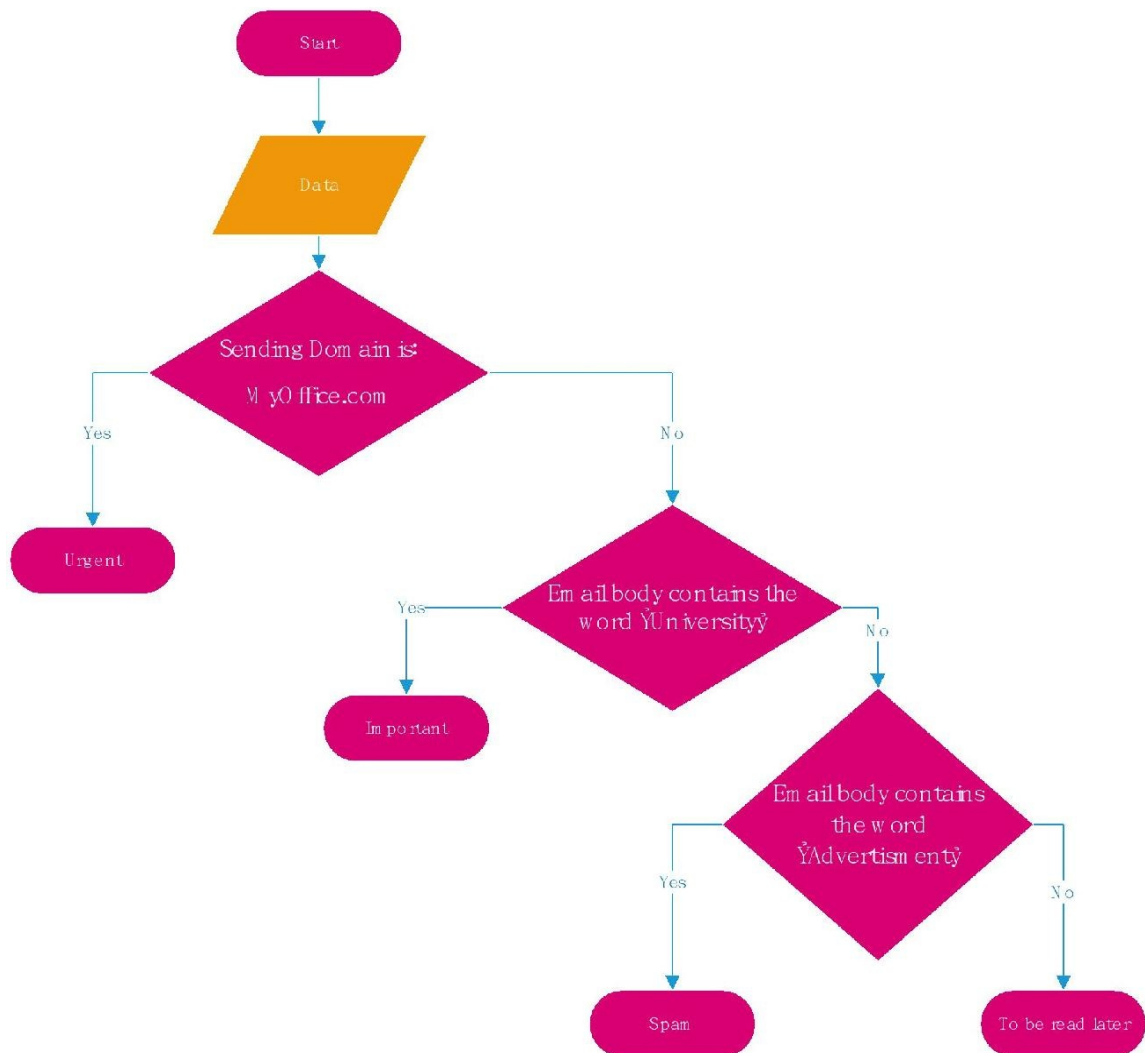


Figure X.XX Decision tree flow chart Note: In this figure, the diamonds are decisions that are upon some conditions. The ovals are terminations.

3.2.2 Decision Tree Construction

Before applying the decision tree algorithm to a dataset, we need first to specify the feature the splitting process will be based on. If you do not have a clear idea about the most feature that rule the dataset, you may try different suggested features until you get the best results. This results in subsets out of the original dataset. Practically, each subset represents a desired feature. Therefore, the subsets keep splitting depending on the features as they travel down the branches. When the data on the branches is the same class, we have classified it and we do not have to continue the splitting process. Else, the algorithm has to be repeated until we have classified all the data. The decision tree algorithm is

summarized in table X.XX. From the table, you can notice the recursive nature of the algorithm; as it calls itself until all data is classified. A python code will be written throughout our talk about the decision tree section.

DECISION TREE ALGORITHM
- EACH OBJECT IN THE DATASET IS IN THE SAME CLASS.
- YES, RETURN THE LABEL OF THE CLASS
- ELSE, TRY ANOTHER FEATURE TO SPLIT THE DATA
- CREATE A BRANCH NODE
- FOR EACH SPLIT ITERATION CALL THE “DECISION TREE ALGORITHM” AND ADD THE RESULT TO THE BRANCH NODE RETURN BRANCH NODE

Table X.XX Summary of the Decision Tree Algorithm A common way of splitting data in the decision tree is the binary split. But suppose that we have followed the binary way, but we still have other possible features, for much more splits. Therefore, in this section we will utilize the ID3 algorithm^[9]. This algorithm helps in making rules about how to split the data and when to stop.

The decision tree algorithm works with both numeric and nominal values. It is simple in terms of computational complexity and of understanding the learnt results. Moreover, it is robust against missing values and can handle features that are irrelevant. However, the main disadvantage of the decision tree algorithm is that it is prone to overfitting. Overfitting is a common defect in the decision tree and other classification algorithms. It occurs when make multiple hypothesis about the training data, i.e. assigning meaningful values to noise. This indeed reduces the errors in the training dataset but at the expense of the testing dataset. In other words, an overfit model is too specific to its training data. So that, when it deals with a new data, its prediction error in this new data is at risk of being very high.

3.2.3 Building and Visualization of a Basic Decision Tree in Python

In this section, we will learn how to use *scikit-learn* tool in python to build a basic decision tree. Moreover, we will show you how to visualize the decision tree using other helping libraries and packages like *matplotlib* and *scipy*.

Chapter Four

Clustering

In the previous chapter, we talked about regression as an example of the supervised learning algorithms. In this chapter we deal with an unsupervised learning algorithm, which is clustering. In unsupervised learning, we don't have a target variable. Instead, we need to get a description about input dataset. For example, in supervised learning, the main aim is to get an output Y for a set of data, X . on the other hand, in unsupervised learning, we are interested in what the machine can tell us about the dataset X . For instance, we may ask the machine to tell us about the best three clusters that we can make out of a given dataset or what are the most repeated four features in the dataset. In this chapter, we will study one type of clustering algorithm called k-means algorithm. The reason that it is called "k-means" because the algorithm tries to form k unique clusters, in which the center of every cluster is the mean value in that cluster.

Clustering is a type of unsupervised learning that finds groups of similar observation. In fact, everything can be clustered. Clustering is more efficient when the most similar items are located in the same cluster. The key difference between the classification, supervised learning, and the clustering algorithms is that in classification we are looking for ways that distinguish pre-classified groups. In clustering, on the other hand, no classes are available, and we want to make some natural classing of instances. Clustering is sometimes called: unsupervised classification. This is because it finally produces the same result as classification but without having predefined classes.

Some applications of the clustering: - **Genomics:** Forming clusters of genes with similar expressions.

- **Astronomy:** Stars, for example, can be distinguished based on their colors, sizes, distances, materials, etc. clustering can be applied to find groups of

similar stars and also galaxies.

- **Earthquake investigations:** Detected earth quake epicenters should be clustered along continent cracks.
- **Marketing:** Identify customer groups and utilize this for targeted marketing and re-organization.

4.1 The k-means clustering algorithm

k-means is an algorithm that tries to form k clusters for a given dataset. The number of groups/clusters k is defined by user. Every cluster is described by its centroid which is a single point at the center of all the points in the cluster, and here where the name comes from; k-mean. The k-means algorithm works with numeric values, means symbolic are excluded. The main advantages and disadvantages of the algorithm is as follows: **Advantages:**

- Easy to implement.
- Simple, quick, and understandable.
- The items are automatically assigned to clusters.

Disadvantages:

- The number of clusters has to be picked in advance. This can be solved by trying the algorithm, k-mean, with different k 's, then choose the clustering that has the best quality.
- Results can alter significantly depending on initial choice of seeds. Also, the algorithm can converge at local minima. Suggested solution is to restart clustering a number of times with different random seeds.
- Slow on very large datasets. This can be solved using sampling.
- Too sensitive to outliers.
- Very large values can skew the means. One suggested solution is to use medians instead of means.

The k-means method approach can be summarized in the following three steps:

1- Choose, randomly, a K number of centers for the cluster.

2- Assign every point in the dataset to its closest cluster center. The assignment is done by finding the closest centroid using methods such as Euclidean distance.

3- The centroids are all then updated by moving each of them to the average of its assigned points.

4- The steps two and three are repeated, till a convergence reached. i.e. when

the cluster assignment change becomes less than a specific threshold. The flow chart in figure X.X explains how the pseudo-code of the algorithm would look.

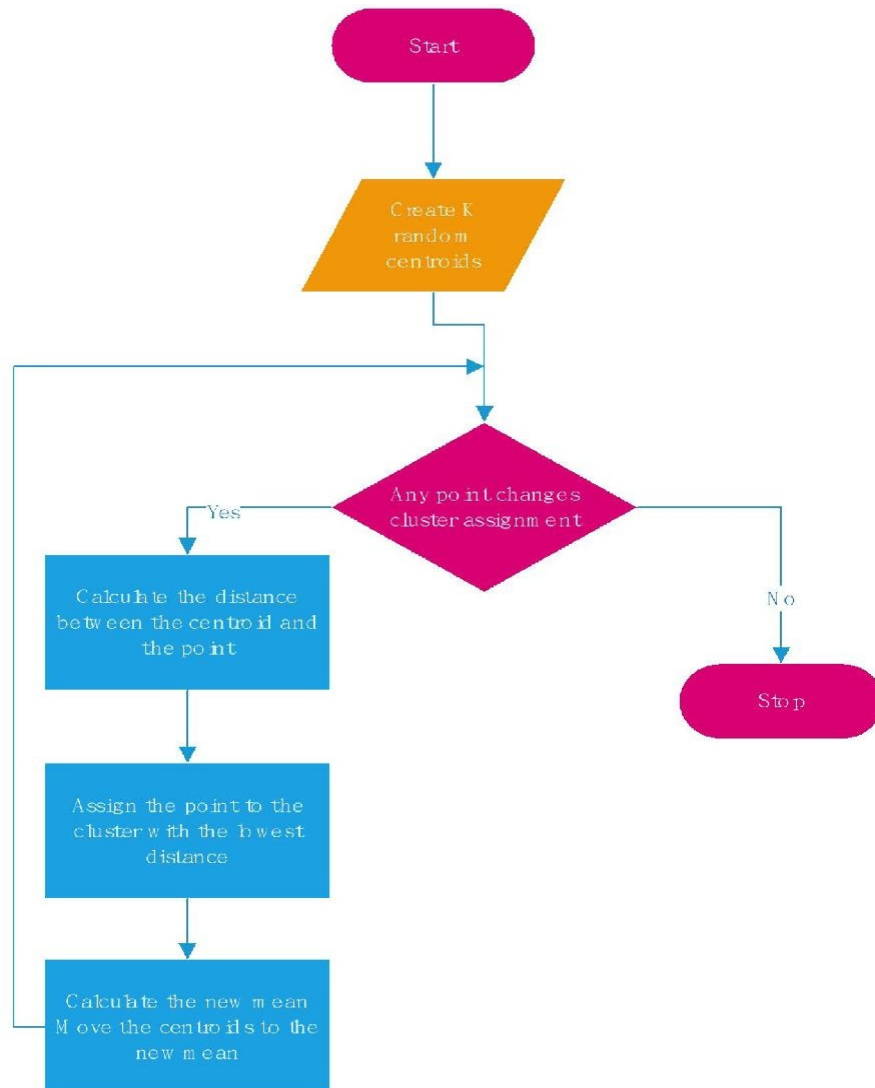


Figure X.X K-mean clustering algorithm, flow chart

Note: if the book capacity allows, a detailed design example using python will be discussed in this chapter.

1. Bias and Variance

Prediction models' errors can be caused by three main components: bias, variance and irreducible errors. In large datasets, there might be a number of undefined variables. These variable may affect the input to output mapping relationship. Therefore, the irreducible error is resulted. As can be understood from its name, there is no way to decrease the irreducible error, whatever learning algorithm is employed. Therefore, we care here mainly about: bias and variance errors, since there is a tradeoff between a model's capability to minimize bias and variance. investigating the two error types help us in diagnosing model results and avoid the mistake of over- or under-fitting.

1.1 Error due to Bias

The bias error is the difference between the average prediction of our model and the correct value which we are seeking to know. If we have only one model, the speak about average means that we have to run the model multiple times, then compare the average of the resulted values out of those runs to the correct value we are trying to predict. In general, machine learning algorithms which concern about numeric analysis and results, called parametric algorithms, encounter high bias. These kind of algorithms are simple and learn fast. On the other hand, in complex problems, the parametric algorithms have poor predictive abilities, which make them less flexible. The difference between low and high bias algorithms, is that the first makes low number of assumptions about the required output. On the other hand, the latter makes more assumptions. Common examples of the low bias learning algorithms are: the k-Nearest Neighbors and the Decision Tree. Linear Regression and Discriminant Analysis are examples of the high bias learning algorithms.

1.2 Error due to Variance

Imagine that we can repeat a whole model building process multiple times. The variance is how much the prediction of the target function is going to vary between different realizations of the model for the same given point. A machine learning algorithm is applied to predict the target function from the training dataset. Therefore, it is expected that the algorithm will show variance. If the machine learning algorithm has a good ability to discover the implicit relationship between the input dataset and the output variables, the target function should not change too much from one training dataset to the next. The difference between low and high variances machine learning algorithms can be cleared as follows:

- **Low Variance Algorithm:** In this algorithm, when changes are made to the training dataset, small variations occur to the prediction of the target function.
- **High Variance Algorithms:** In this algorithm, when changes are made to the training dataset, large variations occur to the prediction of the target function.

In general, non-numeric machine learning algorithms, also called non-parametric algorithms, show high flexibility. However, they are high variance algorithms. Common examples of the low-variance learning algorithms are: Linear Regression and Discriminant Analysis. The k-Nearest Neighbors and the Decision Tree are examples of the high-variance learning algorithms.

1.3 Graphical Explanation of the Bias and Variance

A friendly and simple graphical explanation of bias and variance can be created using a bulls-eye diagram. Let the center of the eye represent the correct values of a specific models. The further we go from the eye center, the worse are the prediction results. In the bull-eyes shown in figure X.X below, every black dot represents an individual run of the model. In case of low bias, the model realizations, dots, are very close to the center of the bull-eye. On the other hand, in case of low variance, the model realizations are very close to each other, regardless of their distance from the center of the bull-eye. When we have a good distributed training data, we predict very well and we are close to the bulls-eye. In turn, when the training data full of outliers or noisy data, the predictions will be far from the bull's eye.

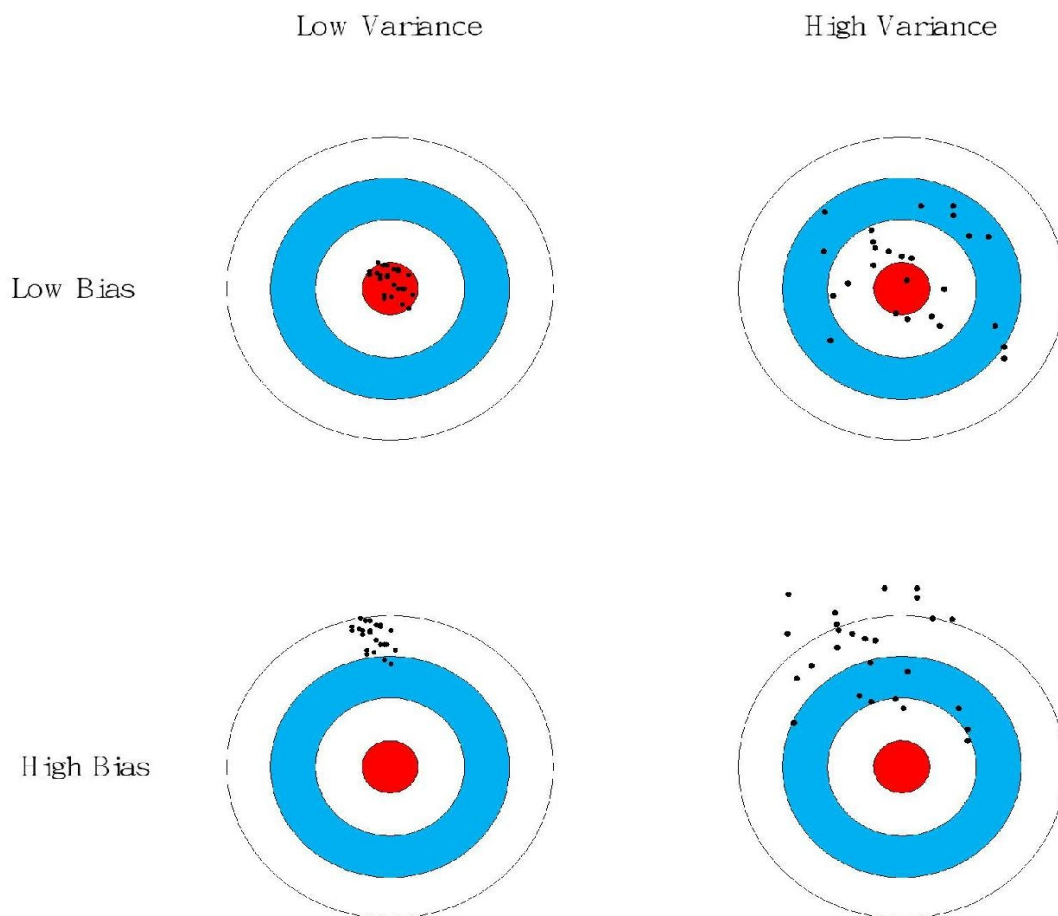


Figure X.X Graphical illustration of bias and variance[\[10\]](#)

1.4 Trade-Off Between Bias and Variance

The machine learning algorithms aim to attain low bias and variance outputs. However, as we seen in sections one and two of this chapter, there is a trade of between the variance and the bias in the machine learning algorithm. Figure X.XX summarized this idea.

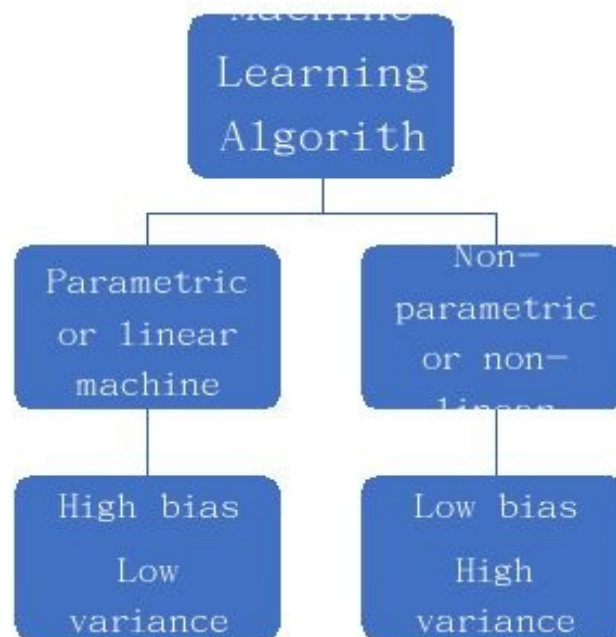


Figure X.XX Trade-off between variance and bias

In machine learning, there is no escaping the relationship between bias and variance. That is, as shown in Figure X.XX, increasing the bias will decrease the variance and vice versa. Therefore, the process of choosing the suitable parameters for a machine learning algorithms is usually a struggle to balance between bias and variance. As an example of the bias-variance trade-off for the k-nearest neighbors algorithms is as follows:

- The k-nearest neighbors (KNN) is a common supervised machine learning algorithm. It shows low bias and high variance. However, if the value of the k neighbors increases, the trade-off situation becomes different. In other words, the increase in the number of neighbors increases the contribution to the prediction process and hence increases the bias of the

model.

This indicates that a careful configuration should be done to the selected machine learning algorithm in order to achieve a balance in this trade-off for the learning problem. Bias and variance are important methods to have an acceptable guide and impression about the predictive performance of machine learning algorithms.

Chapter Four

Artificial Neuron Networks

4.1 Introduction

Neural networks are probably one of the very recent topics. Currently, large corporations and young startups alike are all go rushing into this state-of-the-art field. Deep learning is a term that is associated to the neural networks. Both, deep learning and neural networks have great ability to learn from data and the environment. Therefore, they are the first and preferred choice for machine learning. Common application in which neural networks employed are:

- Cars auto-driver.
- Image processing and recognition.
- Consult and recommender systems.

Artificial neural networks are powerful scheme that show great adaptive ability to wide range of data types. Being artificial brings to the surface the origin of the naming. Artificial neural networks mimic the human being nervous system. Figure 4.1 shows the human neuron. A neuron consists of the following main parts:

- **Dendrites:** These are the recipients that receive the input electrical impulses from other connected neurons. The decision is then taken by the cell body, which concludes to the action needed from those inputs.
- **The axon and the axon endings:** These are the neuron transmitters. They transmit the output decisions as electrical impulses to other neutrons.

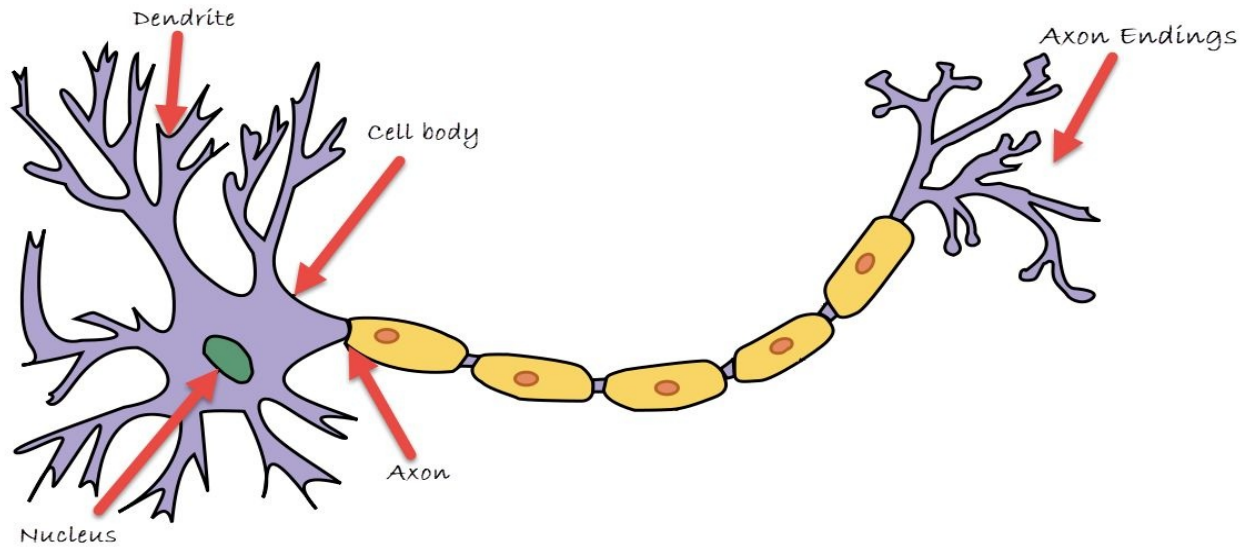


Figure 4.1 Human Neuron

The neuron can be divided into three parts/stages: inputs, processing, and output. This is the key to state the analogy between the biological and artificial neural networks.

4.2 Artificial Neural Networks

An Artificial Neural Network (ANN), is a computational model benefits the idea of the biological neural networks, as depicted previously. The artificial neural network is a machine learning way that mimics the human intelligence in dealing with data. Neural networks are a remarkable class of supervised machine learning algorithms. They are employed in both regression and classification. In this chapter, we will discuss the basics of neural networks, and show a practical example at the last section. Other known types, uses according to the data to be analyzed, are convolutional and recurrent neural networks. The first showed very good performance in image processing. Recurrent networks, on the other hand, achieved a very well level of performance in sequential data related problems such as dynamic system modeling and natural language analysis.

There are, mainly, three different layers in neural networks:

1. The input Layer.
2. The hidden Layers: In this layer, the inputs received from the previous layer are processed. The hidden layer can contain more than one layer.
3. The output Layer: After having the data processed, it becomes available in this layer.

Figure 4.1 shows a graphical representation of the neural networks' three layers. It can be noticed that the variables are represented by nodes. The network layers are distinguished via colors: red, green, blue, for input, hidden and output layers, respectively. Let us define X_0 and H_0 as dummy variables. They always take the value of one. The dummy variables are shaded in figure 4.1.

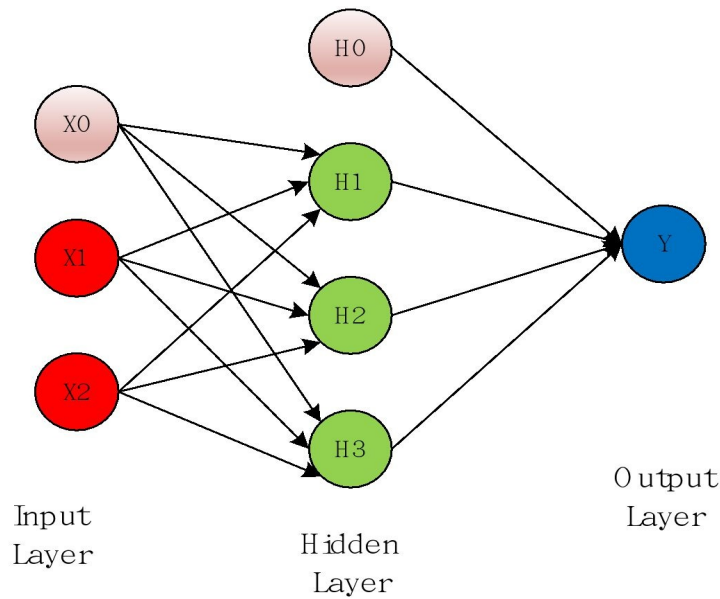


Figure 4.2 Neural Network layers

1- Input Layer:

The Input layer rule is to interact with the external environment. It deals with the inputs come from the data sources. The input is then transferred to the hidden layer. Each input represents an independent variable which affects the output of the network. The inputs are some features such as: Height, Age, weight, image data or hours of sleep.

2- Hidden Layer

The hidden layer is the intermediate layer, lies between the input and the output layers. It consists of the group of neurons that has activation function applied on it. The job of the hidden layer is to deal with and process the inputs got from the previous layer. Therefore, its responsibility is to extract features of interest from the input data. The hidden layer can include more than one hidden layer. So that, one of the problems to be considered is the choice of the number of the hidden layers depending on the learning category. If the input data can be linearly separated, then we can skip the hidden layer. This is because the activation function can be designed to input layer that is able to solve the problem. In the problems that have complex decisions to make, three to five

hidden layers can be used. In the case of multiple hidden layers, the symbol H will have a superscript that indicates the hidden layer number. A neural network with more than one hidden layer is also referred to as deep learning or a deep neural network

The choice of the number of hidden layers depends on the problem complexity or on the level of the required accuracy. Of course, this does not mean that if one keeps increasing the number of hidden layers makes the neural network becomes more accurate. The algorithm will reach a level in which adding extra hidden layer will not affect its steady state output, or sometimes may cause the results to fall below the best reached performance. This means that the accurate calculation of the number of neurons per network is a very important step, either to meet the dataset's complexity or to avoid overfitting.

3- Output layer

The output layer is the stage where we got the values that we want to predict. A good benefit of the neural networks is that the output pattern can be traced back to the input layer. This means that there should be a logical relation between the number of neurons in output layer and the type of work performed by the neural network. The number of neurons in the output layer is related to the intended use of the neural network.

- Latent variables

In the neural networks, the latent variable term is common. However, it is not very new to the reader of this book. Recalling from the Regression chapter, when we looked at the relationship between the dependent and independent variables. Latent variables are functions of the inputs. This relation is expressed by edges, in neural networks. each edge flows from left to right, which means that every variable that receives an edge is a function of the variable from which the edge comes and also the weight associated with this edge. From figure 4.1, we can notice that each variable in the

hidden and the output layers is a function of all previous layer variables, except the bias variable H_0 which has the value of 1.

- Deep Learning and Deep Neural Networks

As said before, the number of the hidden layers may be more than one layer. This results in, mainly, two types of neural networks depending on the number of hidden layers: deep and shallow neural networks. Of course, the deep category refers to the neural networks that have more than one hidden layer. This distinguish is essential because we cannot use the non-linear transformations that fit the shallow neural networks in deep networks; as that may result in poor performance. In the hidden layer, it is common to use functions that are known as activation functions. In section 4.4, we present famous activation functions. As will be noticed, they are all almost non-linear function of the weighted sum of the variables in the layer. Using these functions to produce a hidden layer means that a non-linear transformation is needed. In deep neural networks, we surely have more than one hidden layer which means that we have to perform some feature transformations on the original input features.

4.3 The Flow in the Neural Network

Every neuron, in the ANN, represents an activation node. The activation node is connected to the input nodes in order to apply learning algorithms to calculate the weighted sum. The weighted sum is then passed to an activation function which leads to predict the results. Here where we have the concept of “a perceptron” arises. Perceptron means things that take many inputs that lead to one output.

In figure 4.3, we have inputs that are direct independent variables or outputs from other neurons. Every input X has a weight of W . The importance of weights is that they are indications of the significance of inputs. In other words, as a weight goes higher, the input has a greater contribution to the output results, and vice-versa. As can be seen from figure 4.3, every perceptron has a bias that is an indication of the flexibility of a perceptron. The bias can be compared to the constant B in the famous line equation: $Y = AX + B$, as it enables us to adapt the line either up or down to have better prediction results. As known from simple elementary algebra rules, if we do not have the constant B , the line will always pass through the origin point and in our case in machine learning, we will have poor results.

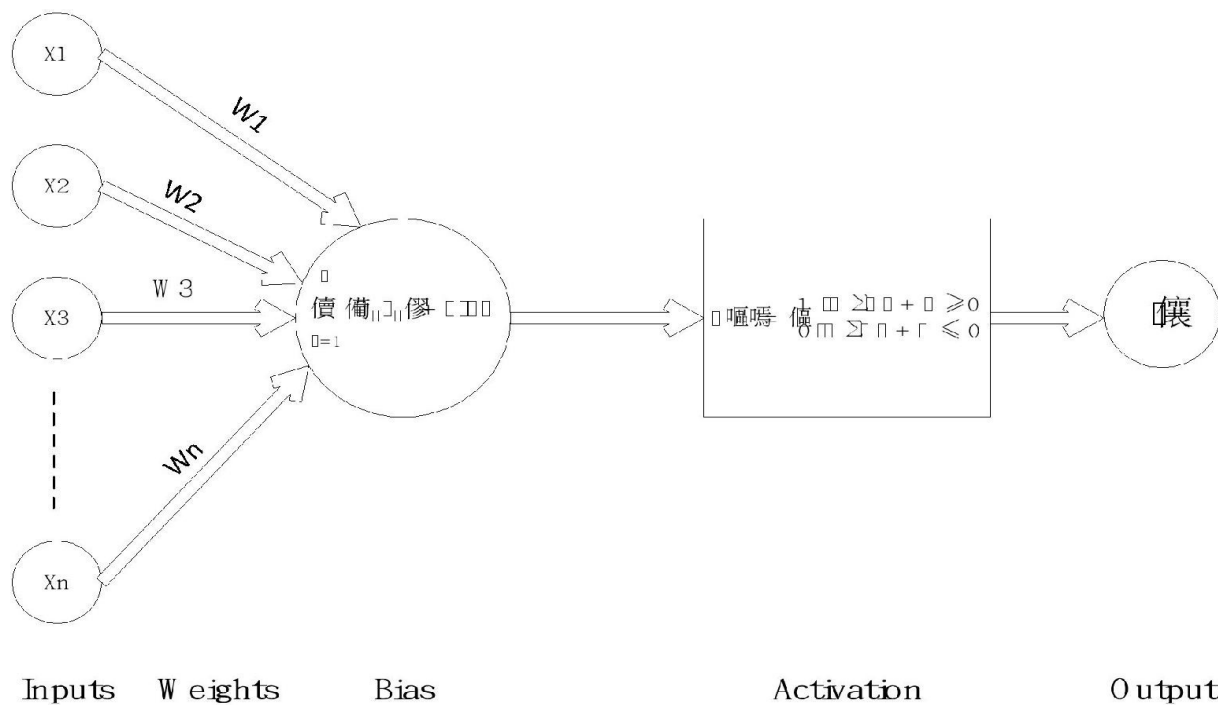


Figure 4.3 Detailed Neural Network

In the “bias” named stage, as seen in figure 4.3, we can notice that the weights are summed and a bias is added. This sum goes then through an activation function. The activation function forms a gate that opens and closes. The activation function can give ones or zeros, judge based on a threshold or make probabilities. Regarding the probability part, it may utilize the sigmoid or the rectified linear function. More about activation function will be discussed later in this chapter.

The activation function output is the waited, predicted, output from the neural network. It is indicated as \hat{y} in the figure above. \hat{y} may be regression, binary or classification values, depending on the nature of the dataset and the required output.

4.4 activation function

In this section, we will make a quick review for common activation functions used for the ANN.

- **Threshold or step function:** If the sum of the weights and inputs is below a specific threshold it is zero and if it is above, it is one. Recall the gate analogy we talked about.
- **Sigmoid function:** It is the most widely used activation function. It looks like the threshold function but it is smoother. Sigmoid activation function is suitable for making probabilities. This means that the output range of the function will be always between zero and one. For example, if we are investigating a picture, the output will be the probability whether it is for a cat or a dog?!
- **Hyperbolic Tangent:** This function is a stretched version of the sigmoid function, as it ranges from -1 to 1. This function is chosen when we want deeper gradient and steeper derivative. Therefore, the choice between the sigmoid or the hyperbolic tangent functions depends on the gradient strength requirements.
- **Rectifier Linear Unit (RELU):** This function is normally applied in the reinforcement learning application. From its name, it is a linear function that has been rectified. This means that it has the value of zero in the negative domain and it increments linearly, i.e. it for a positive input x , it gives an output x and it is zero otherwise. One major advantage of RELU is its simplicity compared to the hyperbolic tangent and the sigmoid functions. This is a main point to consider in designing the deep neural networks.

According to the characteristics of the function we are trying to approximate, we can choose the activation function that achieves faster approximation, which means faster learning process. The sigmoid function, for example, shows a better performance in classification problems than the RELU. This means a faster

training process and also a faster convergence. One may use his own custom function. However, if the nature of the function we are trying to learn is not very clear, then it is preferable to start with RELU as a general approximator and then work backwards. Table 4.1 summarizes the mentioned activation functions along with there graphical representations and equations.

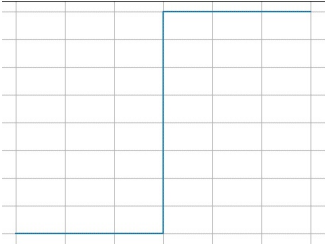
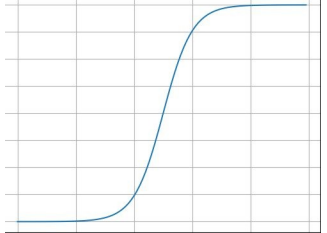
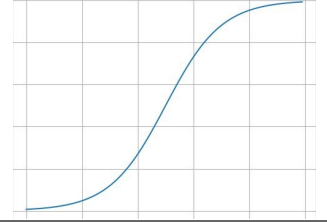
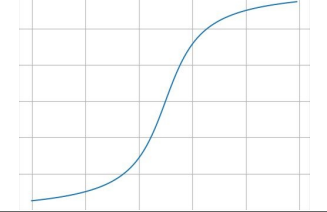
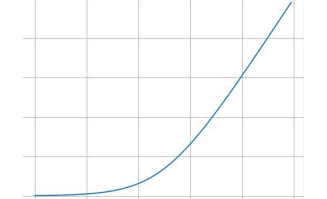
Activation Function	Equation	Graph
Unit Step	$\theta(x) = \begin{cases} 2, & x \geq 0 \\ -2, & x < 0 \end{cases}$	
Tanh	$\theta(x) = \tanh(x)$	
Sigmoid	$\theta(x) = \frac{1}{1 + e^{-x}}$	
Hyperbolic Tangent	$\theta(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
Smooth ReLU	$\theta(x) = \log(1 + e^x)$	

Table 4.1 Activation Functions

4.5 Working Example

In this section we are giving an example that translates some of the concepts that explained in the previous sections. Suppose that we have a trained neural network, figure 4.4. This means that we have the weights optimized as seen in the figure. The input layer contains a group of features: age, distance to the hospital, gender and income. The output on the other hand, is the variable that is dependent on the input features. It is the probability that a person will be hospitalized. If we take the age, for example, we can state that the older the person, the more likely he/she will be hospitalized. In terms of gender, statistically men are more likely to be hospitalized than women. The data collected can be utilized to make several predictions depending on the input features and this is the realization of the neural networks job. The hidden layer in this example was considered as a black box which contains the neurons and weights that result in the hospitality probability. The machine can be trained to predict the relation between the distance to the hospital and the probability of being hospitalized. Therefore, the neural network has to try first to find patterns in the data and figure out the relations between these different patterns. This is a kind of understanding data before making other processing or decisions. once trained, we are able to input different features extracted from the targeted dataset. For example, one record can be something like that: Age is 65 years old, gender is female, moderate distance to the hospital and high income. The neural network is a magical method that expect and predict the desired results after processing the features in a smart way that mimics the human neuron.

The neural network estimates the different combinations of the input features to identify patterns in the dataset based on the neural network architecture. It may be seen as a black box, but a deeper look allows us to see patterns amongst the weights, inputs and hidden layers.

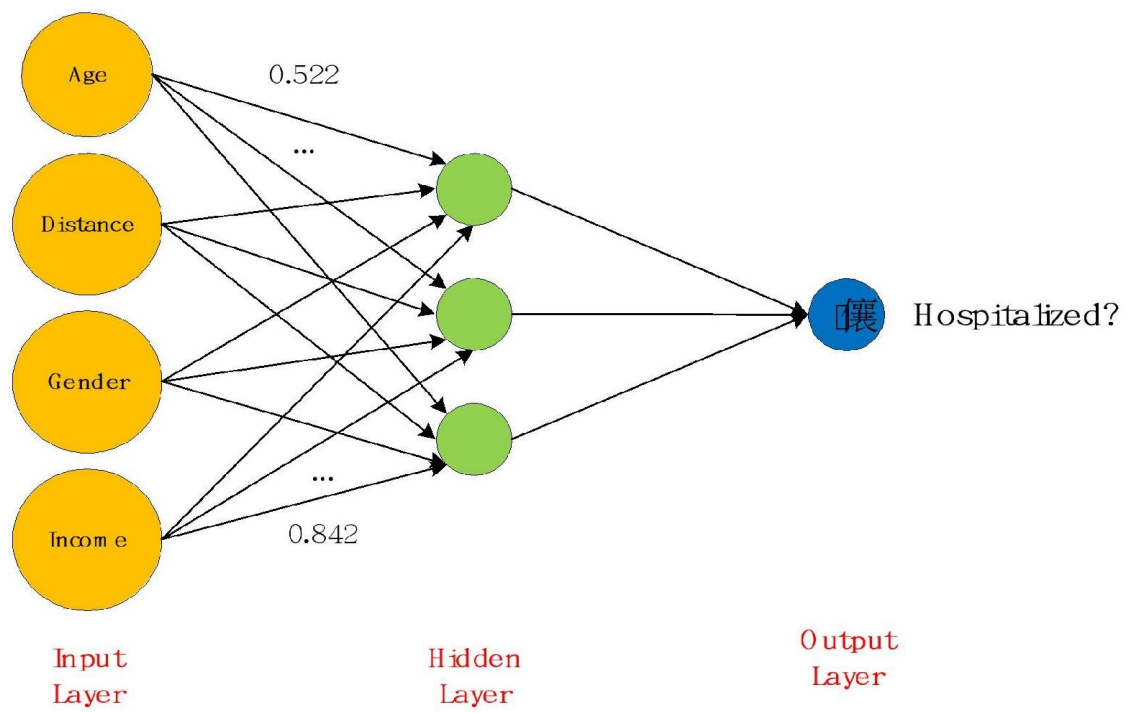


Figure 4.4 Neural Network, Hospitality Probability Example

4.6 The learning process (How the weights work)

We normally need machines to be learnt since we have large amounts of data to be processed and analyzed. Therefore, if we have a large dataset, we feed its inputs into the neural network to make predictions. The first step is to determine the neurons weights. They can be determined based on initial assumption or they can be predefined based on the application features and the required output.

The normal flow of the neural network is towards the output \hat{y} . That is, the input features are fed to the hidden layer where the calculations are performed. Then, the network flow continues until we have the desired output. This kind of flow is known as: **Forward Propagation**. At this point, the smart error handling appears. The resulted \hat{y} is compared to the actual value of y . The target is to make the difference between them as small as possible, i.e. minimizing the error. This can be clearer if we imagine the analogy of a child learning math. If he answered a specific question with 8 while the accurate answer is 5, then we have an error of 3. In this case, the calculations have to be re-performed until we converge as close as possible to 5.

The neural network utilizes the weights in its task to minimize the resulted error. It reduces the weights of the neurons that make significant contribution to the error. This process is known as: **Back Propagation**. This is because in this tuning process we travel back from the output to the neurons and input, in order to locate where exactly the error happens. Here is where the importance of the activation functions comes to the surface. The activation functions are differentiable and this helps in performing the backpropagation process through the neural network; by computing the gradients. Hence the weights are adjusted. The simultaneous adjusting process of the weights continues until we have a neural network output that is close to the actual output. The weights are slightly adjusted in each iteration to have a smaller error at each

run. The process is repeated for all the inputs and outputs possibilities of the training dataset until the error is significantly small and acceptable by the application.

4.7 How the back propagation is performed?!

The following equation is a representation of the simplified cost function in a neural network:

$$E(j) = \frac{1}{n} \sum_{i=1}^n (\hat{y}^i - y^i)^2 \quad (4.1)$$

E in equation (4.1) is the squared error between the predicted output \hat{y} and the actual output y . Conceptually, we can produce a plot between the error E and the predicted output \hat{y} . Then, all the possibilities of the weights in the network can be tried using methods like the Brute-Force technique. This is supposed to produce a result like a parabola of data. However, this easy approach requires imaginal computing power for large dataset to examine all the probabilities. To clarify it, for a moderate sized dataset we may need hundreds of years to tune the weights and get the results! Here where weights optimizing techniques comes to the surface.

4.7.1 Gradient Descent

In gradient decent, we focus on making accurate predictions in much less time than the case of estimating all of the possibilities, as mentioned before. The first step is to provide the neural network with initial values of weights. Then, we can pass in our data following the forward propagation way. Our mission now is to estimate the output \hat{y} and compare it to the actual output y . In most cases, the predictions resulted from the first run are not very accurate, i.e., we have a high error value.

Assume we have a cost function for a specific value of W . To make it simple we will assume numerical values. If the weight W is 1.6 and the resulted cost function value, E , is 3.2. We need to adapt the weight in the next run to reduce the value of the cost function. The question is: what if we could discover the way whether to make W larger or smaller, in order to decrease the cost function?!

What we are going to do is to test the cost function to the left and to the right at a specific test point. Then, we check which one of the two ways produces smaller cost function value. This method is called numerical gradient estimation. It is a good approach, but if we look back at the cost function equation (4.1), we can think in smarter way by utilizing derivatives, i.e., proceeding to the gradient decent concept. We want to know which way is downhill, leads the function to the minimum value. And in other words, we are going to check the rate of change of E with respect to W .

What we need to do at this stage is to derive the of $\frac{\partial E}{\partial W}$ which will give us the rate of change of E with respect to W . Then, at any value of W if we have $\frac{\partial E}{\partial W}$ positive then the cost function is directed uphill. Whereas if $\frac{\partial E}{\partial W}$ is negative, the cost function is directed downhill. This mean that we now know which direction decreases the cost function. So that, we are able to speed up the tuning process. This is because we saved all the time needed in searching for values in wrong directions. Moreover, additional computational time is saved by taking steps iteratively at the direction that minimize the cost function and then stopping when the cost function is not getting smaller any more. This is the conceptual and practical realization of the **gradient descent** scheme. we may think of the gradient descent optimization as a hiker. The weight in this case is to climb down the hill towards the valley, which is the cost minimum. Following this analogy, we can determine each step by the slope steepness, the gradient, and the step distance of pay growth which is the analogy to the learning rate at this case.

In the gradient decent, we have high and low learning rates. In our hiking analogy, the high learning rate is the case when we take big steps towards the valley before checking our position. This may end to the case where we never reach the minima of the cost function. On the hand, when small steps are taken, we boost up our opportunity in reaching the minima. However, this may take

very long time. So that, we need to tune the learning rate empirically until we reach the minima. A smart solution is to think about the adaptive learning rate utilizing the gradient concept. That is, the steeper the gradient, the higher the learning rate and the smaller the gradient the smaller steps it takes to reach the final value. The gradient decent may not seem very special in one dimensional problems. However, it significantly decreases the time required to tune the neural network's weights in higher dimensional datasets.

An important issue to take care about when solving using the gradient decent is the convexity of the data. Sometimes the dataset we are dealing with may be a non-convex. This means that the cost function doesn't always decrease while going in the same direction. Literally, the cost function in this case decreases and then increases again. This behavior is known mathematically as non-convex function. Example is shown in figure 4.5. In non-convex functions, the gradient decent method is not able to give accurate predictions any more since it will stuck in a local minima instead of spotting the global minima.

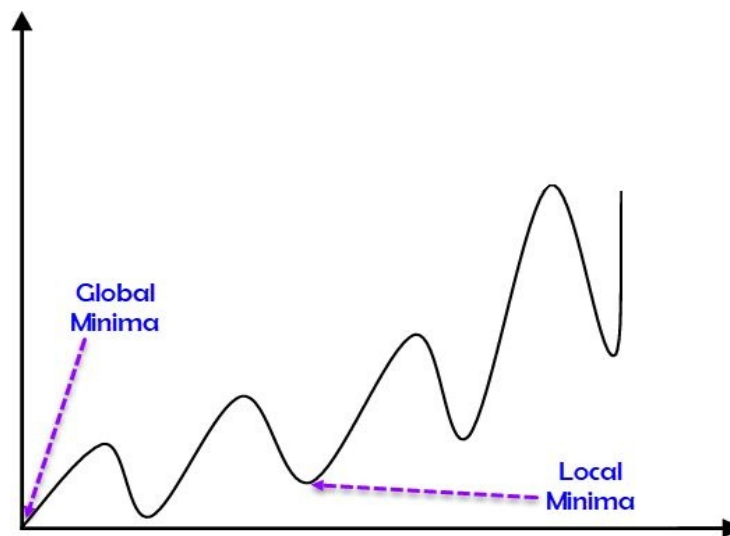


Figure 4.5 Non-Convex data Here the reason of having a squared cost function is revealed. The sum of the squared error values enables us to utilize the convexity nature of the quadratic functions.

Therefore, as the cost function E equals the squared values of x , then the plot of E is a convex parabola.

A main point to mention before finalizing our talk about the gradient

descent is that the practical convexity of a dataset depends on how we deal with the data. Sometimes we can follow the principal: “one at a time instead of all at once sometimes”. In this case, we are not very interested in the overall convexity issue of the dataset. This talk leads to our discussion in the next section about the stochastic gradient descent.

4.7.2 Stochastic Gradient Descent

The gradient descent calculated the gradient of the whole dataset. On the other hand, the stochastic gradient descent calculates the gradient using a single portion of the dataset. This makes the stochastic gradient descent shows a faster convergence than the gradient descent, since it performs updates much more frequently. Considering that the datasets often contain redundant information, we are very satisfied with the stochastic gradient descent which does not use the full dataset. Table 4.2 summarizes a comparison between the gradient and the stochastic gradient descent.

Gradient Descent	Stochastic Gradient Descent
Computes the gradient using the full dataset	Computes the gradient using a single sample of the dataset
May stuck in local minima	Converge faster and avoid sticking in local minima

Table 4.2. Comparison between Gradient Descent and Stochastic Gradient Descent

4.8 Understanding Machine Learning Categories in the Context of Neural Networks

First, let us make a quick review of the datasets types and how it is reacting with the neural networks. Mainly, we have three types of dataset:

- **A training dataset:** it is a group of data samples that are utilized for the learning process. Therefore, in the neural networks at this stage.
- **A validation dataset:** a set of samples which is used to adjust the network parameters. For example, in the neural network, it can be used to choose the number of hidden layers.
- **A test dataset:** This set is a group of examples that are used to evaluate the performance of a fully neural network or in predicting the output whose input is known. It is also used to check that we do not overfit our data.

We know from the **supervised learning** that we have a training data as the input to the network and the desired output is known. In neural networks, the weights are tuned till the output meets the desired value. In the **unsupervised learning** we have a data that we are trying to understand and extract its features. Therefore, in neural networks, the input data is utilized to train the network whose output is known. The network is then clusters the data and adjust the weights by feature extraction in the input data. In the **Reinforcement Learning**, we do not know the output. However, the neural network can give a feedback telling if the input is right or wrong. This type of learning is also called a semi-supervised learning. **Offline, or batch, learning** makes the required tuning to the weights and to the threshold only after employing the training set to the network. Finally, in the **Online learning**, which the opposite of the batch learning, the tuning process of the weights and the threshold is made after employing each string example to the network.

4.9 Neural Networks Applications

A lot of things we encounter daily use the recognition of patterns and exploit this results in making decisions. Therefore, neural networks have the ability to be adopted in daily life applications and missions. Examples of using neural networks are in stock market, weather predictions, radar systems to detect the enemies' aircrafts or ships. It can be also used to doctors in diagnosing complex diseases on the basis of their symptoms. Neural networks are in our computers or smart phones. They can be programmed to identify images or handwritings. certain neural network can monitor some parameters to spot the characters you are typing, such as: the lines you are making, your fingers' movements and the order of your movements. Voice recognition programs are another example that significantly utilize the neural networks technique. Some email programs or tools have the ability to separate the genuine emails out from the spam emails. These programs also use neural networks. Neural networks have shown a high efficiency in text translation, from language to another. Google online translator is one of the famous tools that employs neural networks over the last years to enhance the machine performance in converting/translating words from language to another.

[1] L.G. Alexander: Operation Mastermind.

[2] Paul Anderson: Superman from the South by Jim Murray

[3] Malley, Brian and Ramazzotti, Daniele and Wu, Joy Tzung-yu, Secondary Analysis of Electronic Health Records", year="2016".

[4] Son NH (2006) Data mining course—data cleaning and data preprocessing. Warsaw University. Available at URL <http://www.mimuw.edu.pl/~son/datamining/DM/4-preprocess.pdf>

[5] Previous references

[6] <http://www.sacbee.com/>

[7] https://github.com/ahmedfhd1/machine_learning/blob/master/real_estate_transactions.csv

[8] <http://www.statisticshowto.com/calculators/linear-regression-calculator/>

[9] http://en.wikipedia.org/wiki/ID3_algorithm

[10] <http://scott.fortmann-roe.com/docs/BiasVariance.html>