OULU UNIVERSITY OF
APPLIED SCIENCES

Nibesh Khadka

GENERAL MACHINE LEARNING PRACTICES USING PYTHON

GENERAL MACHINE LEARNING PRACTICES USING PYTHON

Nibesh Khadka
Bachelor's Thesis
Spring 2019
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Nibesh Khadka
Title of the bachelor's thesis: General Machine Learning Practices Using Python
Supervisor: Kari Jyrkkä
Term and year of completion: Spring 2019 Number of pages: 73

Machine Learning (ML), is a process of teaching an algorithm to learn, feeding data. Algorithms try to find patterns from data to generalise a rule or relation to predict future unseen instances. Some of the popular results of machine learning systems are Google Translate, YouTube's Video Recommendation System, Movie Recommendation System.

The aim of the thesis was to introduce the readers the concept of ML and the phases of the ML model development process as well as their implementation in the Python programming language.

Different categories of ML with examples, typical phases in an ML modelling have been explained in theory in the thesis. In addition, pre-processing data, training and testing models, optimization of a model in the supervised machine learning have been further elaborated using Linear Regression and K-Nearest Neighbor (KNN) as examples in Python programming language.

The paper, as a result, can act as a starting tool and conventional guide to a beginner ML practitioner for any ML algorithms other than presented in the paper. People with knowledge of Python can further benefit from the conventions and alternatives presented in the coding section of the paper.

Keywords: Machine Learning, ML Phases, Supervised Machine Learning, Python Programming, Linear Regression, KNN

PREFACE

I would like to thank my supervisor, Kari Jyrkkä, a senior lecturer at Oulu University of Applied Sciences, for his continuous support and feedbacks which helped me improve my paper. I would like to thank my language teacher, Kaija Posio, a senior lecturer at Oulu University of Applied Sciences, for her feedbacks which helped me improve my writing.

Nibesh Khadka

31.05.2019, Oulu

CONTENTS

VOCABULARY

ABBREVIATIONS

CPU = Central Processing Unit

FP = False Positive

FN = False Negative

IDE = Integrated Development Environment

KNN = K-Nearest Neighbour

MAE = Mean Absolute Error

ML = Machine Learning

MSE = Mean Squared Error

RMSE = Root Mean Squared Error

SVC = Support Vector Classification

SVM = Support Vector Machine

TN = True Negative

TP= True Positive

TERMS

Bias = ML model performance is not good, i.e. very low accuracy, at both training and testing data. This situation is referred to as Highly Biased or underfitting. It indicates the model is not a good fit and/or data is not meaningful enough. During such a situation, it is recommended to use a complex ML algorithm and play through data to make it more meaningful.

Categorical Data / Qualitative Data = Data type that are non-numerical.

Classes = Prediction Values in Classification Algorithms.

Continuous Data / Quantitative Data = Data type that are numerical.

Data-frame = Database represented in columns and rows with labelled columns and indexed rows. It is a Pandas object.

Dependent Variable = Features column or target column to be predicted.

Domain Knowledge = Expertise in a field. For instance, ML engineer working for Nokia 5g network projects can benefit from knowledge about internet networking processes.

Dummy-Variable Trap = It is a scenario created due to multicollinearity between column. In such a situation on the column can help in the prediction of another one so, there is no need for an algorithm to learn anything. To avoid such a situation, one of the multiple dummy variables created during conversion from categorical to dummy numerical variables is dropped. After dropping one column, the algorithm is forced to keep on guessing.

False Negative = When the prediction result is False (or negative or 0) but the actual result is True (or positive or 1), it is called False Negative. It is also referred to as Type II error.

False Positive = When the prediction result is True (or positive or 1) but the actual result is False (or negative or 0), it is called False Positive. It is also referred to as Type I error.

Features / Variables = A column in data frame. For instance, age column, sex column, price column.

Feature Engineering = Processing raw data to make it ready to be used by algorithms.

Independent Variable = All the feature columns except the to be predicted columns, that affect the result of prediction, are the independent column.

Multivariate = More than one variable.

Outliers = Data in a column that is exceptional from its siblings, in ML outliers can affect results negatively. For instance, A number 1000000 in the list of numbers less than 1000. (1, 200, 300, 999, 100, 25, 56, 465, 789, 1000000).

Parameters = Input of the Estimators or algorithms, some of which can be determined by the practitioner and some cannot. They determine the result.

True Negative = When the prediction result is false (or negative or 0) and the actual value is also false (or negative or 0), it is called True Negative.

True Positive = When the prediction result is true (or positive or 1) and the actual value is also true (or positive or 1), it is called True Positive.

Univariate = Single features or variable.

Variance = It is the situation when the model's performance is exceptionally good for training data, but poor results are achieved for unseen or test data. The situation indicates the use of complex ML model unnecessarily. It is also referred to as High Variance or Overfitting. The model tries to fit each data point rather than generalizing underlying pattern between points.

1 INTRODUCTION

Human beings, can easily distinguish the tree from rock, can learn languages, learn to read and write, learn to drive. Everyday performance gets better and better. People learn, from their surroundings, from trails and success, from failures, from companions. Every time a task is performed people's brain receives feedback, learns from feedbacks and saves them for future references.

Similarly, Machine Learning (ML) is a process of teaching an artificial system to learn through enormous amounts of data. It is crucial to remember that the aim of the machine learning exercise is to produce a self-sufficient model for a specific task.

With the development of technologies, several programming languages have been adopted by machine learning engineers and data scientists to solve machine learning problems. Some of the popular programming languages used are Python, R and Java. Python is one of the popular programming languages for machine learning practices.

Python is a high level, general-purpose programming language developed in 1991. Python has been used in several platforms such as desktop applications, server-side scripting, machine learning, data analysis, web development. [1]

The thesis will review several phases in Machine Learning. It explains in brief the ML and ML phases carried out to obtain an optimal ML model. The thesis introduces several packages available in the Python programming language used for different ML phases through examples of two supervised ML algorithms and their development in code. However, the codes and packages are more conventions than rules. In coding, there can always be different methods and steps to perform the same operation.

2 MACHINE LEARNING

Machine Learning is a field of computer science which uses statistical algorithms and techniques to teach an artificial system an ability to learn feeding a huge amount of data. [2]

The field of machine learning is huge and widespread, but there are some popular use cases successfully touching the lives of billions of people.

1. Recommendation System: YouTube suggests your videos based on your personal history, likes and dislikes. Netflix users get movie recommendation based on the user profile and histories.
2. Spam Detection: Gmail has a separate folder that says Spam, which holds all spam emails. This is a brilliant application of machine learning saving people from frauds.
3. Google Translate: It detects language automatically and translates to the language of choosing instantly.
4. Speech Recognition: Siri in Apple products and Alexa from Amazon are products of deep learning which is the field of machine learning.

As shown in figure 1, the field of machine learning can be classified into three categories. [2]

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

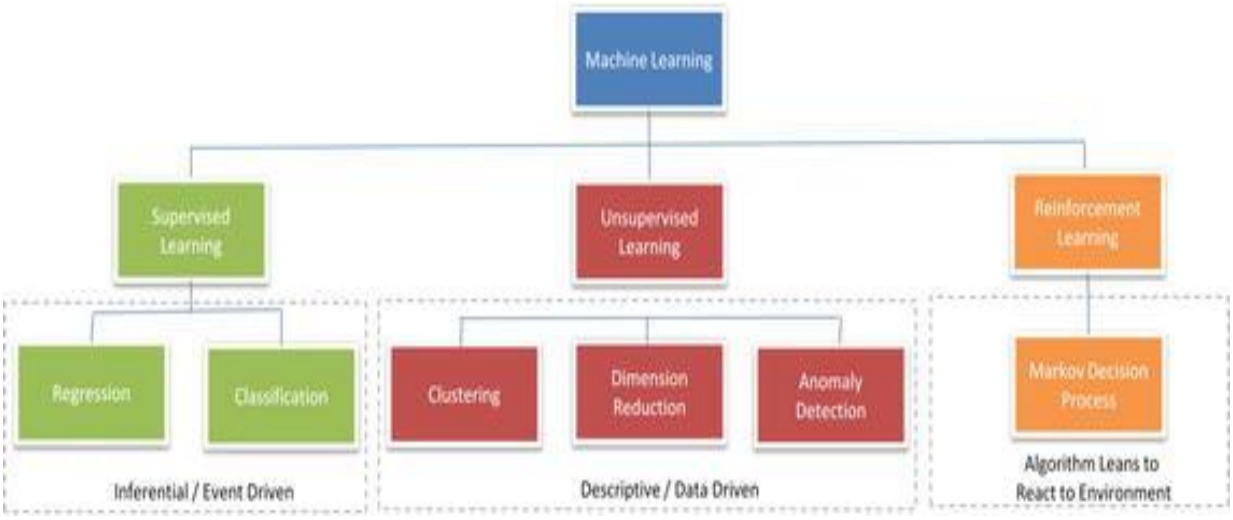


FIGURE 1. Types of machine learning [3]

2.1 Supervised Learning

Supervised learning in ML is the process in which prediction classes are already known. In supervised learning, a learning algorithm analyses rules and patterns using label input data, a group of dependent and independent variables during training and generalising rules to deduce future unseen instances of data [4]. For instance, if the sky is cloudy and the clouds are dark, the weather is more likely to be rainy than sunny.

Supervised Learning has been further divided into two sub-types depending upon the desired data type of prediction, Regression and Classification

2.1.1 Regression

ML algorithms that are used to predict continuous data-type are categorized as regression learning algorithms, for instance, prediction of stock prices or real state housing prices. [5]

Some examples of regression algorithms are Linear Regression, Polynomial Regression and Support Vector Machine (SVM).

Linear Regression

Linear Regression is the type of learning algorithms where the dependent and independent variable has linear relations. Figure 2 demonstrates a plot between training data and the prediction of the model. A linear slope is created between prediction and training data, where the difference is the error.

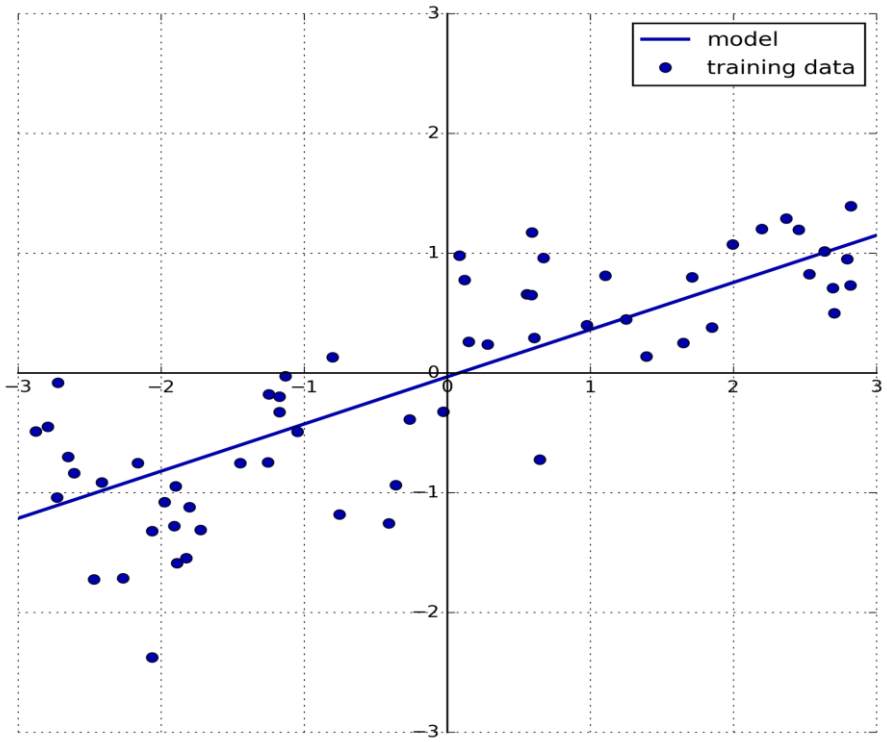


FIGURE 2. An example of linear regression [6]

A mathematical formula for a simple linear regression has been provided in formula 1 below.

*y= m*x + c* *FORMULA 1*

y = predicted value,

m = coefficient,

x = independent variable

c = constant or intercept.

A multivariate linear regression equation can be expanded as shown in formula 2 below.

$y = m_1*x_1 + m_2*x_2+\ldots+ m_n *x_n + c$ *FORMULA 2*

n = the number of independent columns.

Hence, in terms of simple linear regression, it can be interpreted as:

*y = sum of m*x of n number of columns + c*

2.1.2 Classification

Classification in ML is used to predict categorical data types (Vocabulary - Terms – Categorical Features), for instance, prediction of weather or directions. Some examples of classification algorithms are Random Forest Classification, Decision Tree, Support Vector Classification (SVC) and Logistic Regression. Logistic Regression, unlike the name suggests, is a classification algorithm. [5]

K-Nearest Neighbor

K-Nearest Neighbor divides a data point among given categories based on the distance between a new data point and previously assigned data points. Figure 3 is displaying the selection process in KNN through visual aid.

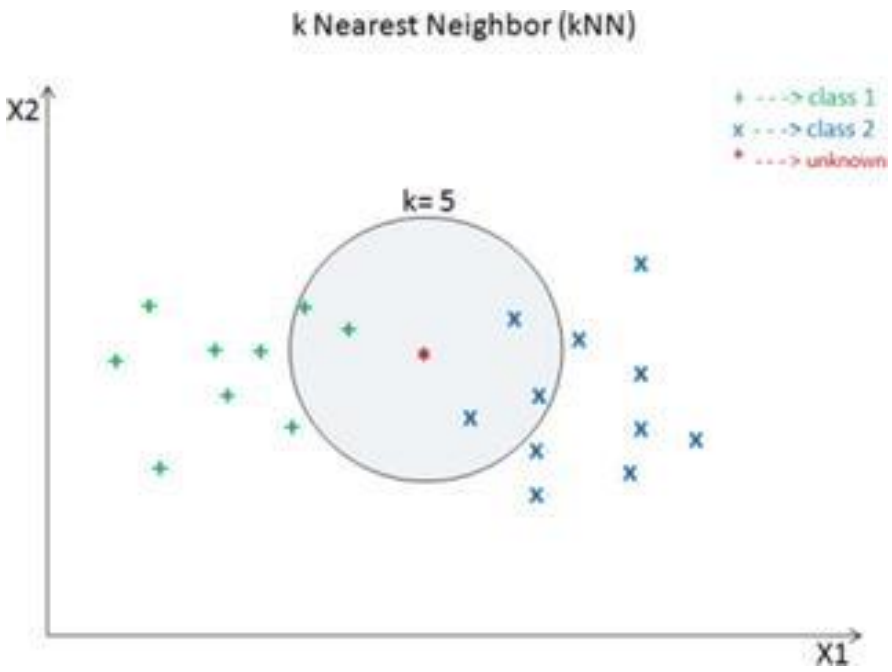


FIGURE 3. K-nearest neighbor [7]

Typically, a K-Nearest Neighbors selection process can be explained in the following steps:

1. The algorithm takes a new input data point (red point in figure 3), calculates the distance between the new data point and its neighbor data points (blue and green points in figure 3). Distance calculation metrics can be chosen, the default one in sklearn's KNN algorithm is "Euclidean Distance" metric.
2. Then, the KNN algorithm chooses a K number of data points among all previously assigned data points, fitting the distance metric. For instance, if k is 5, and distance metric is "Euclidean Distance", it will pick 5 data points (among blue and green points in figure 3), closest to new data points in a straight line.
3. Among the 5 chosen points, the category (dependent classes 1 and 2 in figure 3), consisting of the highest number of chosen data points or neighbors, is assigned a new data point. For instance, in figure 3 the

number of blue points is 3 and that of green is 2 which are the K = 5 closest points represented inside the circle. Thus, the redpoint falls into Class 2 in figure 3.

2.2 Unsupervised Learning

Unsupervised Learning is a category in ML where the prediction classes are unknown, and prediction is done based on an analysed pattern from features available. There are situations in real life where features available are unclear or meaningless, in such scenario, unsupervised learning algorithms are used. In simple words, if it is not known, what is predicted, then use unsupervised learning techniques should be used [8]. Some examples of unsupervised algorithms are Clustering, Anomaly Detection, Dimension Reduction.

Clustering

As the name suggests, in clustering, predictions are clustered together in several groups based on the patterns recognized from the relationship between independent variables. K-Means Clustering, Agglomerative Hierarchical Clustering are few examples of clustering algorithms. [8]

2.3 Reinforcement Learning

Reinforcement Learning is feedback-based learning, where algorithms receive feedbacks from the environment and adjust the parameters that affect the result. Some examples of the Reinforcement learning techniques are the Markov decision process, Q-learning, Temporal Difference methods, Monte-Carlo methods. [9]

Hence, Machine learning is a process of teaching a machine how to learn by feeding a large amount of relevant data. ML can be divided into three categories, Supervised Learning, Unsupervised Learning and Reinforcement learning. The method is Supervised learning when dependent variables are labelled. The method is Unsupervised learning when dependent variables are unlabelled,

thus, it is the responsibility of algorithms to group prediction in several classes. In the coming chapters, details of Supervised ML phases are explained and implemented using the Python programming language.

3 PYTHON FOR MACHINE LEARNING

Python is a high level, general-purpose programming language developed in 1991. Python has been used in several platforms such as desktop applications, server-side scripting, machine learning, data analysis, web development. [1]

Python Modules and Packages

Python modules are files in Python with “.py" extensions. Packages are collections of modules and packages are often referred to as namespaces [10]. Modules and Packages are a prewritten set of codes aimed for reuse to reduce the workload of developers.

3.1 Python Packages for Machine Learning

Python has been used widely in the field of ML. There are several packages in Python used for data analysis and machine learning. Some of them are as follows:

3.1.1 NumPy

A NumPy abbreviation for a Numerical Python is a package for scientific computation. It is used for creation and manipulation of multi-dimensional arrays, mathematical operations, such as mean, medians, sum, maximum and minimum values. [11]

3.1.2 Pandas

In machine learning, the data fetched from sources is converted into dataframes and further processed by desire. For Python, Pandas is an open source library used for creation and manipulation of dataframes. [12]

3.1.3 Scikit Learn

Scikit-Learn, also known as sklearn, is an open source package for Python, being used for data mining, data analysis, machine learning and data science. The sklearn package offers various machine learning algorithms and tools for various purposes, such as estimation, pre-processing and splitting. [13]

3.1.4 Matplotlib

Matplotlib is a Python library for data visualization. Many statistical diagrams, for instance, histogram, graph and scatterplot can be drawn using Matplotlib. Data Exploration is an essential part of machine learning, where data visualization libraries, such as Matplotlib, is used. [14]

3.1.5 Seaborn

Seaborn is a data visualization library based on Matplotlib. Statistical plots, such as heatmap, violin plot and time series can be constructed easily with seaborn. [15]

This chapter was a very short introduction on Python Programming language and the packages available for ML. Further in chapter 5, several modules inside these packages were utilized to elaborate ML phases.

4 PHASES IN MACHINE LEARNING

ML is a complicated process. Procedures mostly depend on the data, desired output, client or employer. Typical ML phases can be categorized as the following:

1. Fetching Data
2. Data Pre-processing
3. Data Splitting
4. Modelling
5. Model Optimization / Tuning

Feature Engineering

Sometimes, steps 1 to 3 (Data Fetching, Pre-processing and Splitting) in the list are also referred to combinedly as feature engineering. In short, Feature engineering is a process where a raw data is processed and remodelled as per need of the project after the feature engineering data is ready to be fed to an algorithm. Figure 4 demonstrates a lifecycle of data from a starting point to a trained model in ML. Feature engineering comprises a very important section in the flow of data.

FIGURE 4. Logical flow of data in machine learning model building [16]

In a real-world application, the size of data is huge. Data is generally referred to as big data. Not only the size but also the data type can be different, data can be in the form of video, audio, text or image. Feature engineering is applied to cut a meaningful portion of data randomly from a huge lump, make data more informative, the whole process faster and less memory intensive for the CPU

(Central Processing Unit). Hence, feature engineering is an iterative process. Domain knowledge can be a benefit when it comes to data pre-processing. [17]

Basically, after being pre-processed data should be meaningful, free from outliers and ready to be fed to algorithms. Table 2 is a table achieved from replacing, removing and adding values from table 1 for further processing.

TABLE 1. Raw data before pre-processing

ID	Age	Salary per Year (In US $)
1	NaN	150,000
2	20	60,000
3	30	175,000
4	NaN	180,000
5	35	120,000
6	110	120,000

TABLE 2. Data after pre-processing

ID	Age	Salary per Year (In US $)
1	28	150,000
2	20	60,000
3	30	175,000

4	28	180,000
5	35	120,000

However, some flaws of table 1 and table 2 are that in real-world data is not this simple and small. Also, if a table has a categorical feature column, it should be converted into numerical values since algorithms cannot take categorical values as an input.

4.1 Fetching Data

Data is often stored in clouds or hardware in companies. As mentioned earlier data can be in several forms such as video, audio, image and texts. In this thesis, all the data being used for experimentation is either freely available in Python sklearn libraries or has been created by the author. In Python, Pandas can be used to read the files in several formats. This will be handled in detail in Chapter 5.

4.2 Pre-processing

4.2.1 Missing Values

As mentioned in section Feature Engineering, pre-processing is the phase where raw data is processed and cooked to be fed to the algorithm. Often in practice, many values in data are missing and its responsibility of ML engineer to find a solution to that issue. [18]

Often the mean or the median of a feature column is used as replacement value if the feature is continuous. However, extreme values can affect the quality of the mean or median [18]. For instance,

A set of numbers X= [12,5,6,7,100, 90000, missing_value]

The formula for calculating mean is provided in formula 3 below.

Mean = Sum of total numbers / Total numbers *FORMULA 3*

missing_value_1 (Mean of X) = (12+5+6+7+100+90000)/6 =15021.67

If the outlier is removed from the set. Then,

missing_value_2 (Mean of X) = (12+5+6+7+100)/5 = 26

The difference is huge and can affect the predictive quality of the model negatively. Similar can be the case for the median. Hence, outliers should always be removed before performing operations, such as mean and median [18]. The mode operation can be used when the feature is a categorical value. The mode is a statistical method which returns the most occurring value in a list.

An ML prediction method is also used by advanced practitioners. In this method, the feature columns with missing values are treated as dependent variables and values are predicted to fill in the missing value with the use of appropriate ML algorithm. [18]

4.2.2 Normalizing Data

Data as raw obtained from various sources is often of various units and unbalanced. Using the data without scaling down or neutralizing can affect the algorithm. For instance, in table 2, the values in the "**Age**" column range from 20-30 but values in the "**Salaries**" column range from 60,000 – 80,000 and the scale is quite unbalanced. Since algorithms are some form of mathematical operation of variables (formula 1 and formula 2) the unscaled variable will not produce a concise result. Hence, it is a common practice to normalize continuous features. Normalizing in an ML process is done before or after splitting data into training and testing dataset [19]. Two common methods used for normalizing data are as follows:

Normalization / Min-Max Scaling

Normalization is the process of scaling numeric values within the range of 0 to 1. It is done by subtracting the minimum value in the list by the given value and dividing the difference by the difference between the minimum and maximum value in that list (or feature column) as shown below in formula 4. [19]

Normalization = (Given Number- Minimum Value) / (Maximum Value – Minimum value) *FORMULA 4*

Outliers can affect the normalized values since the calculation involves an upper and lower margin in the list. Hence, outliers should be removed before normalizing a numeric column.

Standardization

In standardization, numbers are transformed in a manner where their mean is zero and the standard deviation. The formula for calculating standardization is shown below in formula 5. [19]

Standardization = (Given Number In column – Mean of Column) / Standard Deviation of Column *FORMULA 5*

4.2.3 Feature Generation

Some features in a data-frames can be highly correlated and some can be unnecessary from the perspective of the algorithm.

TABLE 3. Salary of employees in company x

ID	Age	Salary per Year (In US $)
1	25	150,000

2	20	60,000
3	30	175,000
4	58	180,000
5	35	120,000

In table 3, there are three columns of which the ID column is unnecessary from the perspective of the algorithm. It is only helpful for human eyes. Hence, it should be removed before using it for training the algorithm. There are often multiple features in data. Merging or removing the features often requires domain knowledge and experience.

4.3 Data Splitting

ML algorithms analyse the correlation between features and learn from them to generalize a rule from those patterns. If this model is trained and tested in the same data, the model will fail to generalize a relation which is the sole purpose of ML. Since future instances of test data in the real world after deployment are unknown, data that is already present is divided into train and test set. Hence, it is a common practice to divide data into a train set around 70 to 80 per cent and test set around 30 to 20 per cent of the original data. The key is to divide data randomly.

4.4 Modelling

Data can be normalized before or after splitting data to a train and test portion. However, the inevitable question is which algorithm should be used.

Figure 5, provided by Sklearn, can be used as a guide to solving the query about estimator or algorithm.

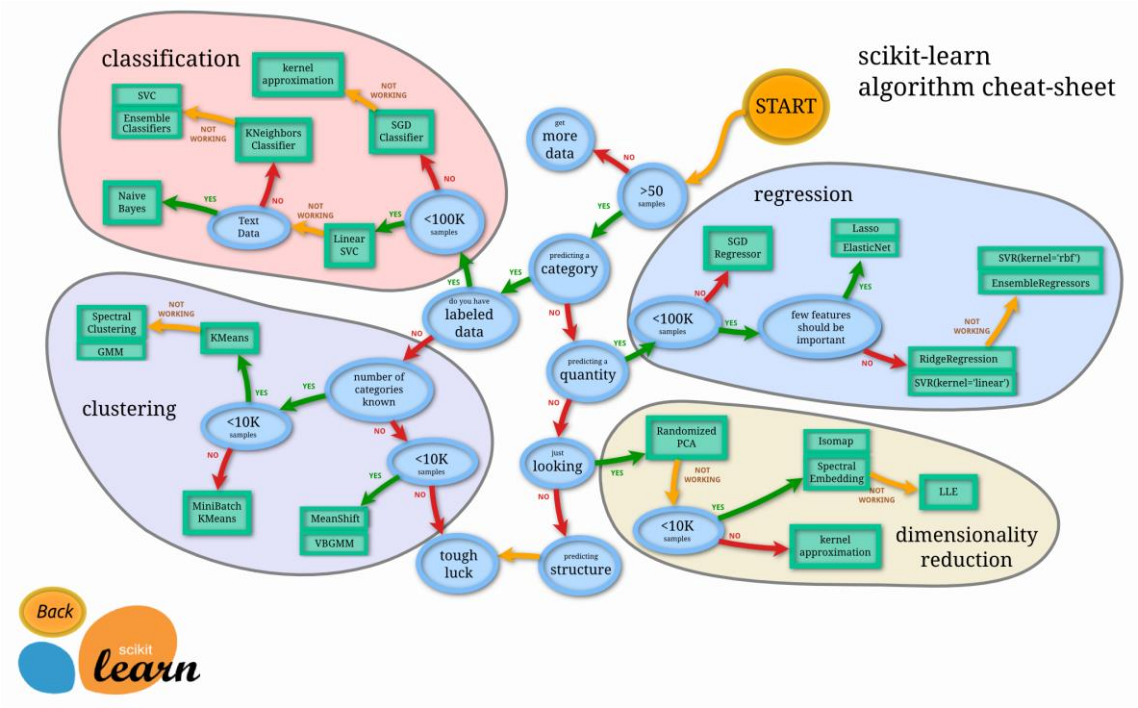


FIGURE 5. Scikit-learn algorithm cheat-sheet [20]

Hence, some key points that need to be considered while choosing an algorithm are:

1. If the dependent feature is of a continuous data type, regression algorithms must be used.
2. If the dependent feature is of a categorical data type, classification algorithms must be used.
3. If feature columns are unknown (unlabelled) and dependent classes cannot be identified, clustering algorithms must be used.
4. In the beginning, a simple algorithm should be used and then climb up to more complex algorithms. For instance, if the result is excellent using simple linear or multi-linear regression algorithms then there is not any need to use polynomial or Support Vector Machine (SVM).
5. Google search and forums can be used for the answers from experts.

After, the decision on an algorithm, in Python, sklearn package can be used to fetch algorithm. The algorithm comes with built-in methods to train and predict data.

4.5 Model Optimization/ Tuning

After the train and test of the model, several reports can be gathered depending upon the type of algorithms.

In regression, the common error evaluation metrics are:

1. Mean Absolute Error (MAE): Mean of the absolute value of errors
2. Mean Squared Error (MSE): Mean of the squared value of errors.
3. Root Mean Square Error (RMSE): Square root of mean squared errors.

Here, the error is the difference between the real value and the predicted value. RMSE is a widely popular one because the unit of measurement of RMSE is the same as the unit of dependent value.

For classification algorithms, a confusion matrix and a classification report can provide an evaluation report.

1. Confusion Matrix: Figure 6 demonstrates an example of a confusion matrix. It is a table consisting of True Positives (TP), True Negative (TN), False Positive (FP) and False Negative (FN). [21]

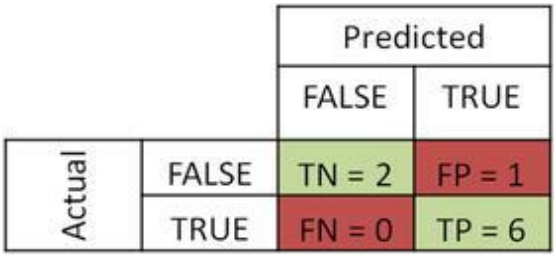


FIGURE 6. Confusion matrix in binary classifier [22]

Accuracy can be calculated as below in formula 6.

(TP+TN) / (Total Number of Values) *FORMULA 6*

2. Classification Report: A Classification Report is a table which provides several metrics of classification. A Classification Report table can be imported from the metrics module of sklearn [23] [24]. From a Classification Report some metrics and their formula are shown in formula 7 and formula 8 below.

   Precision can be calculated as:

   (True Positives / Predicted Positives) = TP / (TP + FP) *FORMULA 7*

   Recall Can be calculated as:

   (True Positives / Actual Positives) = TP / (TP + TN) *FORMULA 8*

The optimization of the model is done in various ways. Sometimes, the process restarts from point zero i.e. pre-processing, where, a column that seems unnecessary is removed. Other times, internal parameters of an algorithm, which can be altered, are altered. For instance, for KNN, the number of K-neighbors parameter or attribute can be altered to achieve better results.

Elbow Method

An elbow method can be used to determine the optimum number of K in K-Nearest Neighbor or in K-Means Clustering (an unsupervised ML algorithm). In this method, an average error (mean of incorrect prediction) from a range of K, for instance, 1 to 10 number of K, is plotted in the graph. Theoretically, the line plot obtained is supposed to be in the shape of an elbow. From the plot, the K which gave the lowest error is chosen as the K for KNN. The K, in theory, is usually the point (sometimes its further below or above) where the elbow bents.

Figure 7 displays an elbow method in theory created for demonstration. According to figure 7, the most suitable value for K is 6 since after that an error rate is not decreasing. Using K higher than 6 is not a good idea, it may force the algorithm to fit every outlier that can result in a high variance.

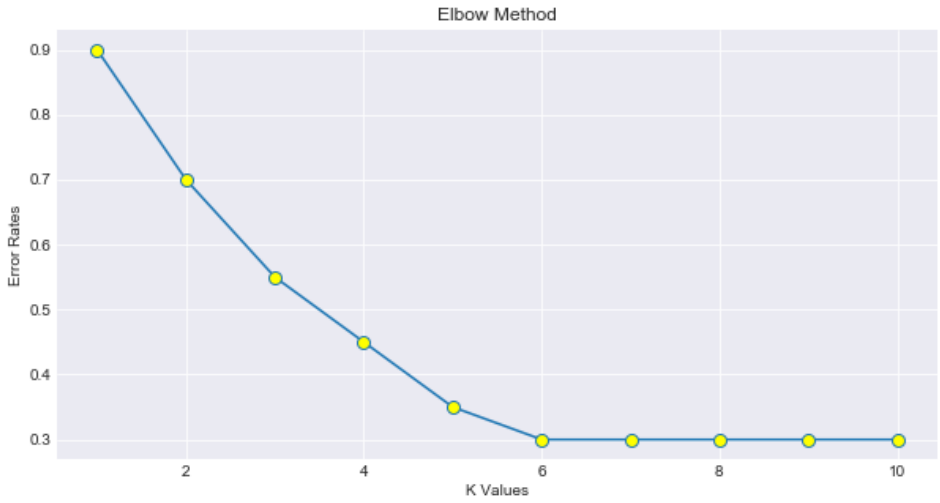


FIGURE 7. Elbow method

Hence, any output is as good as the input. If data is polished properly then only a good model can be achieved. Data is the basic need in the life of the ML model. An ML engineer is a chef that cleans, cuts and cooks the data. Only after that, the ML algorithm is fed with the data.

5 MACHINE LEARNING IN PYTHON

The chapter demonstrates the Supervised ML phases in the Python programming language. Linear Regression and K-Nearest Neighbors are the two algorithms of type regression and classification algorithms respectively.

For the pre-processing phase, a fake dataset was created. For the Regression algorithm, the Load Boston Dataset was used and the Iris Dataset for the KNN algorithm. Load Boston is a dataset about real state housing in Boston where the housing prices are predicted as a regression problem [25]. Iris Dataset is a collection of data about three Species of Iris flowers where three species of Iris flowers are predicted as a classification problem [26]. In sklearn, Datasets are of Bunch types (dictionary-like objects). Hence, it is required to convert to a data frame for further operations. (figure 21 and figure 22)

GitHub

All the source code used in the thesis can be found at the link https://github.com/nibukdk/Thesis.

Importing Libraries and Packages

Packages and Modules need to be installed and imported separately in each file since they are not part of the Python download package. Modules and Packages are sometimes imported whole or partial. The main reason behind partial importing is to save memory.

A way to import is shown below:

1. import package_name as conventional_short_form

for instance,

```
import numpy as np
```

2. import package_name.family_name as conventional_short_form
   for instance,

```
import matplotlib.pyplot as plt

#It imports pyplot submodule from
matplotlib library as plt.
```

3. Instead of importing the whole package small modules and files are imported whenever required. When working with large size files it helps save memory. Scikit-Learn which is known as sklearn in Python coding, importing package can like this.

   from sklearn.family_name import module_name
   for instance,

```
from sklearn.model_selection import train_test_split

#It imports the library to split dataframe.
```

Short forms are used to save time, they have been adopted worldwide as a rule though it is just a convention.

Importing Datasets

The Pandas package is used for importing databases. Pandas support several formats, such as HTML, CSV (Comma Separated Values), TSV (Tab Separated Values). Data is imported and converted into a dataframe object in one step using Pandas. [27]

For instance, to import a CSV file,

Dataframe_name = pd.read_csv(“Name of file”)

5.1 Pre-processing in Python

5.1.1 Information About Data

The dataset used for pre-processing in this section is fake data. The dataset is supposedly data about the salary of anonymous IT employees in five different cities of Finland. The dataset has four columns. They are as follows:

1. Years_of_Experience = Number of years of experience from employees with values ranging from 0 to 10.
2. Cities= Five cities in Finland. They are 'Oulu', 'Tampere', 'Helsinki', 'Turku' and 'Rovaniemi'.
3. Salaries = Fake Salaries of IT employees ranging between 2000 to 5000.
4. Gender = Gender category with values “M” or “F”. However, this column is nearly empty.

Since the dataset is fake and created randomly the correlation between columns is not meaningful. For instance, the salary of an employee of 2 years of experience can be greater than that of 8 years of experience.

The intention was to clean data so several values from the columns were replaced by “**NaN**” (not a number) values. This data was only created for preprocessing, so dependent or independent values are not required to be distinguished. If the reader wants to train an algorithm with this data, “Salaries” column can be used as the dependent column for regression problem or “Cities” column can be used as the dependent column for the classification problem.

The tutorial on creating dataframe was not the actual motive of the thesis. But the reader can visit the source code at GitHub and follow through the code and comments for extra knowledge.

5.1.2 Import the dataset

The full version of the code to preprocess data can be found at GitHub resource provided earlier.

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import seaborn as sns
        %matplotlib inline

In [2]: df = pd.read_csv('Raw Data To Preprocess')
```

FIGURE 8. Import pre-processing data

In figure 8 In [1] which can also be referred to as cell number 1 or input line 1, imports related packages. The last line "**%matplotlib inline"** is used to display plots in a browser in **Jupyter Notebook** IDE (Integrated Development Environment). Line 2 imports the dataset and creates a new dataframe "df".

5.1.3 Info Method

Since data is fake, the describe method (figure 23) and its statistical information are not useful. The result info method is shown below figure 9.

```
In [4]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 4 columns):
Years_of_Experience    42 non-null float64
Cities                 44 non-null object
Salaries               49 non-null float64
Gender                 2 non-null object
dtypes: float64(2), object(2)
memory usage: 1.6+ KB
```

FIGURE 9. Result of info method on pre-processing data

The Info method is very useful to check datatype of feature columns and their null value number. From figure 9, important key points are as follows:

1. RangeIndex: It gives a total length which is 50 ranging from 0 to 49.
2. Other Four Columns: Each column in the df has different lengths. For instance, in Cities, it says 44 non-null objects. It means among 50 values, 44 are non-null, 6 are null values and data-type is float64.

In Pandas, the sum of null values can be calculated by combining methods as displayed below in figure 10.

EDA

```
In [5]: df[['Years_of_Experience','Salaries', 'Cities','Gender']].isna().sum()

Out[5]: Years_of_Experience     8
        Salaries                1
        Cities                  6
        Gender                 48
        dtype: int64
```

FIGURE 10. Pandas null value count

1. **df[['Years_of_Experience','Salaries', 'Cities','Gender']]**, selects the provided list of columns.

2. **df[['Years_of_Experience', 'Salaries', 'Cities','Gender']].isna()**, checks null values in these columns and provides an answer either True for null and False for non-null values and return a dataframe of given lists.
3. **df[['Years_of_Experience','Salaries', 'Cities','Gender']].isna().sum()**, now calculates the sum of true values and outputs the result as shown in the image.

Similarly, the visualization of the null value proportion in the heatmap is shown below in figure 11.

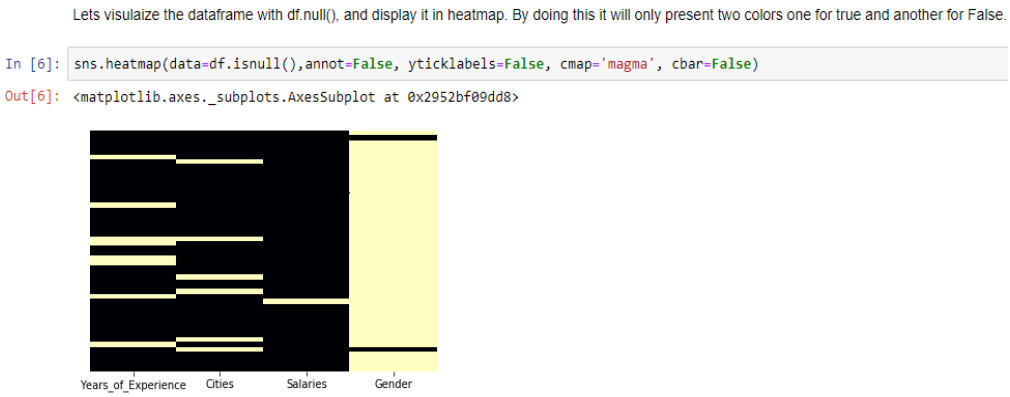


FIGURE 11. Null value count visualization in seaborn

The figure 11, "**sns.heatmap(data=df.isnull(),annot=False, yticklabels=False, cmap='magma', cbar=False)**" can be understood as:

1. **df.isnull()**, returns a dataframe like df.isna(), with values True for null and False for non-null values.
2. **annot=False, yticklabels=False, cbar=False,** just for not displaying data values because correlation is not aim of the operation like in figure 24.
3. In the figure, four columns can be observed in black and yellow colour. The black part represents non-null values and the yellow part represents null-values.

Hence, in conclusion, from figure 10 and 11, the Gender column nearly empty. Since, the Gender column is almost empty, dropping the whole column is a viable option. Gender column was dropped using the drop method provided by Pandas as displayed in figure 12.

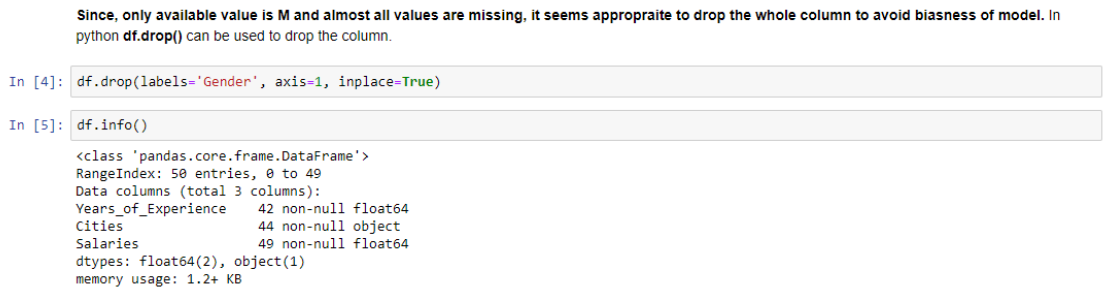


FIGURE 12. Info on data after dropping gender column

In figure 12, the line 4 dropped the column and the new df, as a result, is displayed in line 5 without the Gender column.

5.1.4 Dealing with Missing Continuous Values

Imputing Using Sklearn

Python provides the "**Imputer**" method in the Pre-processing module. The column "**Years_of_Experience**" from the data was used for imputation as shown in figure 13.

```
In [15]: from sklearn.preprocessing import Imputer
```

Imputer takes numpy array with 2 dimensions as input. So, line 18 is the array we need, but we have to reshape it from 1D to 2D as in line 20. Reshape can also be simply done by **your_arrary.reshape(-1,1) for 1D array**.

```
In [16]: type(df.loc[:,'Years_of_Experience'])
Out[16]: pandas.core.series.Series

In [17]: type(df.loc[:,'Years_of_Experience'].values)
Out[17]: numpy.ndarray

In [18]: yrs_experience = df.loc[:,'Years_of_Experience'].values

In [19]: np.shape(yrs_experience)
Out[19]: (50,)

In [20]: yrs_experience = yrs_experience.reshape(50,1)

In [21]: np.shape(yrs_experience)
Out[21]: (50, 1)
```

FIGURE 13. Setting up for imputing

In figure 13, line 15 imported the Imputer function. The Imputer function takes a 2D (2 Dimensions) array as an input. Thus, line 16 checks the type of the column type, which is a series, a Pandas object. However, the line 17 demonstrates the values of a column can be extracted as an array. "**Numpy.ndarray**" means a NumPy array with n dimension where n is the number of dimensions. As mentioned earlier, an array must be 2D. Hence, a variable named "**yrs_experience"** was created and then assigned with an array of values of the column "**Years_of_Experience**" in line 18. In line 19, "**np.shape()**" provides the shape of the array which is 1D (1 Dimension). After that, in line 20 "**name_of_array.reshape()**", a standard method provided by NumPy was used to reshape the 1D array to 2D [28]. The result of the operation is being displayed in line 21.

The figure 14 begins with the imputing initialization.

```
In [22]: imputer = Imputer(missing_values='NaN', strategy='mean')

In [23]: imputer= imputer.fit_transform(yrs_experience)

In [24]: yrs_experience= imputer

In [25]: df['Years_of_Experience']=yrs_experience

In [26]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 4 columns):
Years_of_Experience    50 non-null float64
Cities                 44 non-null object
Salaries               49 non-null float64
Gender                 2 non-null object
dtypes: float64(2), object(2)
memory usage: 1.6+ KB
```

FIGURE 14. Imputing column using imputer

The figure 14 can be described as the following:

1. The Imputer method is initialized and assigned to the variable name imputer. The attribute "**strategy**" is most important to note in line 22. The strategy basically implies the statistical operation to be used for replacing the missing values in the column. As mentioned in section "Missing Values" of chapter 4, statistical operations such as mean, median, mode can be used to replace missing values. So, it is important to provide such information to the imputer method while initializing the method. For this continuous feature column, the mean strategy was used.
2. Line 23 fits and transforms the variable "**yrs_experience**" into imputer at once. This means all the missing values were replaced by the mean of the same columns. It is important to note while calculating mean that only non-null values are included.
3. In line 24, "**yrs_experience**" variable gets reassigned a new list of imputed values. Similarly, in line 25, "**Years_of_Experience**" in the dataframe df gets new imputed values without missing values. Line 26 is the proof of a successful operation because the length of "Years of Experience" column is 50 with non-null values.

Alternative To Impute Using Pandas

It is also possible to use Pandas for imputing [29]. The whole imputing process from the sklearn can be replaced by a single line of code in figure 15, where the missing values were replaced using mean.

Alternative

```
In [22]: df['Years_of_Experience'] = df['Years_of_Experience'].fillna(value=df['Years_of_Experience'].mean())
```

FIGURE 15. Imputing using pandas

Figure 15 can be described as the following:

1. **df['Years_of_Experience'].fillna()**, checks whether the column has null-values.
2. **df['Years_of_Experience'].mean()**, is a function that calculates the mean of a given numerical column only using non-null values.
3. **df['Years_of_Experience'].fillna(value=df['Years_of_Experience'].mean())**, fillna(), if finds null values and replaces them with the mean obtained from the procedure mentioned in step 2. However, not only mean, the value attribute can take any strings or integers as an input.

The last continuous column in the dataframe df is "**Salaries"**. From figure 9, it can be agreed that the Salaries column has only one value missing. There are two things that can be done here.

1. The value can either be replaced with mean or median.
2. The row with null values can be dropped.

For versatility, the row with null value was dropped as shown below in figure 16.

```
In [23]: df.dropna(axis=0, inplace=True)

In [24]: df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 49 entries, 0 to 49
         Data columns (total 3 columns):
         Years_of_Experience    49 non-null float64
         Cities                 49 non-null object
         Salaries               49 non-null float64
         dtypes: float64(2), object(1)
         memory usage: 1.5+ KB
```

FIGURE 16. Drop the row with a null value in salaries column

In figure 16 Pandas "**dropna()**" method drops the null values if found [30]. The most important attribute is the axis. It determines whether to drop a row (axis = 0) or a column (axis = 1). The whole method iterates through the dataframe searching for a null value. In the dataframe df, only null value now is in the Salaries column. Hence, it drops the row with a null value in the Salaries column. Line 24 displays the new information about the dataframe with 49 rows with 0 null values.

5.1.5 Dealing with Missing Categorical Values

Using Mode is a viable option for replacing categorical variables. The dataset df has a categorical column named "Cities". The mode is the most repeated value in a list. For strings, there is no built-in function provided in Python. However, the Pandas "**value_counts()**" can be used to find the most repeated value. [31]

In figure 17, value_counts() was used for counting the total value counts of each value in the City feature column. From line 20 the most repeated city in the "Cities" column is Rovaniemi. In line 21, the fillna() method was used to replace null values with Rovaniemi.

```
In [20]: df['Cities'].value_counts().head(3)

Out[20]: Rovaniemi    10
         Helsinki      9
         Tampere       9
         Name: Cities, dtype: int64

In [21]: df['Cities']=df['Cities'].fillna(value='Rovaniemi')
```

FIGURE 17. Use of value_count() for mode

OneHotEncoder and LabelEncoder

Preprocessing is not yet complete. The feature column, “Cities”, is still a categorical column. ML algorithm can only process numerical values. The Preprocessing module in Python has two methods named LableEncoder and OneHotEncoder. [32] [33]

Figure 18 demonstrates the Label Encoding procedure.

```
In [51]: from sklearn.preprocessing import OneHotEncoder, LabelEncoder

         Import values of all the df

In [52]: X = df.iloc[:,:].values

         Declare label Encoder

In [53]: cities_label_encoder = LabelEncoder()

In [55]: X[0:10,1]

Out[55]: array(['Oulu', 'Tampere', 'Helsinki', 'Turku', 'Rovaniemi', 'Oulu',
                'Rovaniemi', 'Helsinki', 'Turku', 'Rovaniemi'], dtype=object)

In [56]: X[:,1] = cities_label_encoder.fit_transform(X[:,1])

In [57]: X[0:10,1]

Out[57]: array([1, 3, 0, 4, 2, 1, 2, 0, 4, 2], dtype=object)
```

FIGURE 18. Label encoding

In figure 18, line 51 imports both OneHotEncoder and LabelEncoder. The Label Encoder requires the array data type as an input, not the dataframe. Hence, the values of the dataframe were extracted in the next line 52. After that, LabelEncoder is initialized with the name "**cities_label_encoder**" in line 53. Line 54 checks the values of the city before encoding which is a column at position 1 in df. In line 56, the column Cities was fitted and transformed into a numeric column with "**fit_trainsfrom()**" and then reassigned. Line 57 proves the success of the operation.

The LabelEncoder changes categorical values to numerical values. Since an ML algorithm does not know that encoded labels in df are categorical values changed into numerical, a dummy variable, the algorithm can treat values as an actual number. If that happens they might rank cities by order according to their value. For instance, Turku is ranked the highest with its value 4 and Helsinki the lowest with its value 0 (line 55, and 57 of figure 18).

Hence, to prevent that situation OneHotEncoder is used. The OneHotEncoder creates one column for each encoded label, i.e. one column for each label encoded value for the city column. Since the categorical dummies are true and false rather than points, the onehotencoder takes care of that by providing values to newly create columns of either 0 or 1. For instance, after one-hot-encoding,

1. 'Oulu' will be a separate column with the values of either 1 or 0.
2. All the rows before encoding, which have cities with value 'Oulu', will now have the value of 1 as it emphasizes true.
3. All the rows before encoding, which did not have cities with value 'Oulu' will now have a value of 0 as it emphasizes false.
4. Same is the case for each city. For the 5 cities in df, it will create 5 columns.

Figure 19 demonstrates the process of one-hot encoding [33]. Line 59 is for initialization. The attribute "**categorical_features**" is for the position of the column that is to encode. For the dataframe df, the position of the "**Cities**" column is 1.

```
In [59]: oneHotEncoder = OneHotEncoder(categorical_features=[1])

In [60]: X=oneHotEncoder.fit_transform(X).toarray()

In [61]: np.shape(X)
Out[61]: (49, 7)
```

FIGURE 19. One hot encode procedure

In Line 60 transformed the label encoded X. The important thing to notice is that this time the input is whole array, not a single column unlike in LabelEncoder. "**oneHotEncoder.fit_transform(X)**" returns matrix object which is then again changed to an array by the "**toarray()**" function of Scipy [34]. After this operation, the categorical column "Cities" was converted to a dummy continuous variable.

Alternative To Label Encoder And One Hot Encoder Using Pandas

Pandas "**get_dummies()**" method can also create dummy variables from categorical variables. Get Dummies returns a newly created dataframe with the number of columns equal to the number of values in the category column. [35]

Alternative

```
dummy_var = pd.get_dummies(df['Cities'], prefix='Dummy_Var_For_',drop_first=True)
dummy_var.head()
```

	Dummy_Var_For__Oulu	Dummy_Var_For__Rovaniemi	Dummy_Var_For__Tampere	Dummy_Var_For__Turku
0	1	0	0	0
1	0	0	1	0
2	0	0	0	0
3	0	0	0	1
4	0	1	0	0

FIGURE 20. Pandas get dummies

Figure 20 demonstrates implementation of "**get_dummies()**". A new dataframe "**dummy_var**" was created.

The first attribute is the column to be converted. The second attribute "**prefix**" is to prefix the newly created column with the given value. The third attribute "**drop_first**" is set to true. This will drop the first column created after the conversion of cities. The newly created dataframe has four cities and Helsinki is missing. It is done to prevent the dummy variable trap (Vocabulary – Terms - Dummy Variable Trap). For OneHotEncoder a column must be dropped manually after an encode operation.

Now the dataframe df is clean and free of missing values and categorical variables. The other procedures, such as normalizing, more exploration and splitting of data will be demonstrated further in the Linear Regression and KNN section with different datasets. The reader can practice those methods in this data for the learning purpose.

5.2 Linear Regression

Scikit Learn provides several datasets to practice. All the datasets available in the sklearn package is available under module name datasets. For the linear regression, load_boston which is a dataset about housing price, was used to demonstrate ML phases. [25] [36]

5.2.1 Loading and Creating Dataframe

Figure 21 displays a screenshot of codes to load the Boston data however, the process is similar for any dataset available in the datasets module of the sklearn package.

```
In [3]: from sklearn.datasets import load_boston

In [4]: boston_data = load_boston()

In [5]: boston_data.keys()

Out[5]: dict_keys(['data', 'target', 'feature_names', 'DESCR'])

In [6]: #print(boston_data.DESCR)

In [7]: df =pd.DataFrame(data=boston_data.data,columns=boston_data.feature_names)
```

FIGURE 21. Importing load boston dataset

In figure 21, line 3 imports dataset named **load_boston** from the "**datasets**" module of the sklearn library. The line 4 instantiates the imported library, the "**boston_data**" name is an arbitrary name and can be anything. However, it is recommended to provide a meaningful name. The input line 5 displays the list of keys available in the boston_data since boston_data is a dictionary object type in Python. The line 6 prints the description of Boston housing data. It was commented in code to reduce the frame of the screenshot. The difference to notice here is between the codes, **boston_data.DESCR** and **print(boston_data.DESCR).** Both print the same result, however, the latter one takes care of indents, line space, alignments and other things to make it more readable.

Figure 22 demonstrates the process for creating a data frame from the Boston data from figure 21.

```
In [7]: df =pd.DataFrame(data=boston_data.data,columns=boston_data.feature_names)

In [8]: df.head(3)
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03

```
In [9]: df['Price'] = boston_data.target

In [10]: df.columns
```

Out[10]:

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'Price'],
      dtype='object')
```

```
In [11]: df.head(3)
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7

FIGURE 22. Creating dataframe from boston dataset

In the dataset module, it is imperative to understand that the key data has values only for independent features, names of the columns are in the key columns. The line 7 provides relevant values to attributes for the function "**pd.Dataframe()**" and creates dataframe object name **df**.

In line 8, uses the header method in the Pandas and displays 3 rows of the dataframe **df**. Still, the dependent part which is the price column is missing. Since, as mentioned in the previous paragraph, only the independent features were provided earlier to create the df. In the datasets module, the key target is consists of the dependent variable (Vocabulary - Terms – Dependent Variable and Independent Variable). The line 9 added the "**Price**" column to the dataframe "**df**" with the relevant target value. The "**Price**" column is the target column to predict for this regression problem. Hence, from the formula 1 and formula 2 in

the case of Load Boston Data, y is the “**Price**", x's are all other independent variables. The Line 10 and 11, confirms the addition of the “**PRICE**” column to the df dataframe.

5.2.2 Understanding the Dataset

Exploring and getting proper knowledge of dataset (feature engineering) is one of the most important parts of ML.

Pandas Describe and Info

The Pandas library provides the “**describe()**” and the “**info()**” methods that help in an analysis of dataframe. The describe function performs statistical operations and displays the values for each numerical column. Pandas describe method has been demonstrated in figure 23.

```
In [13]: df.describe()
Out[13]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.

```
In [14]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
Price      506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

FIGURE 23. Pandas describe and info

In figure 23, percentage values are first quartile (25%), median (50%) and third quartile (75%) in statistics. The describe method can be helpful to see the dispersion of data and to detect the outliers. [37]

The “**info()**” method prints information about a data frame, such as index, the memory usage, null-values, range and data-type of the columns (figure 7). Boston data, however, is already free from any null values. [38]

Using Pyplot and Seaborn

The graphical representation of data can elaborate on the relation of the database. The figure 24 demonstrates a heatmap on the correlation between features.

```
In [14]: plt.figure(figsize=(10,5))
         sns.heatmap(df.corr(), annot=True,cmap='viridis')
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2050fee14e0>
```

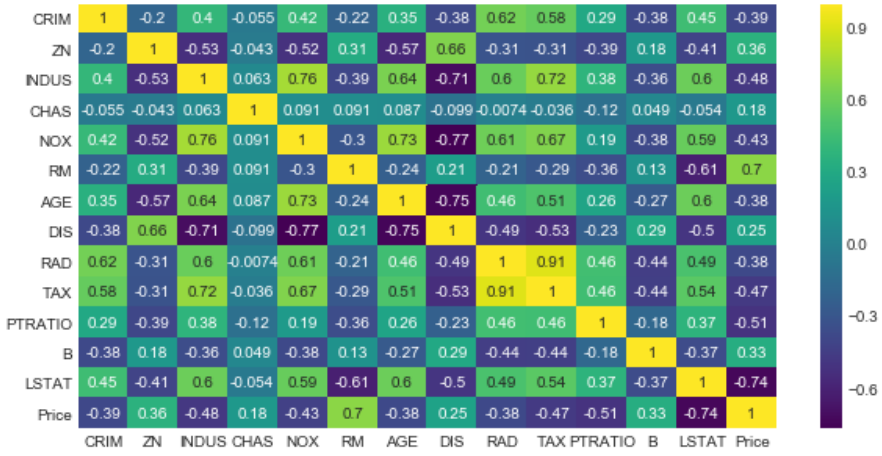


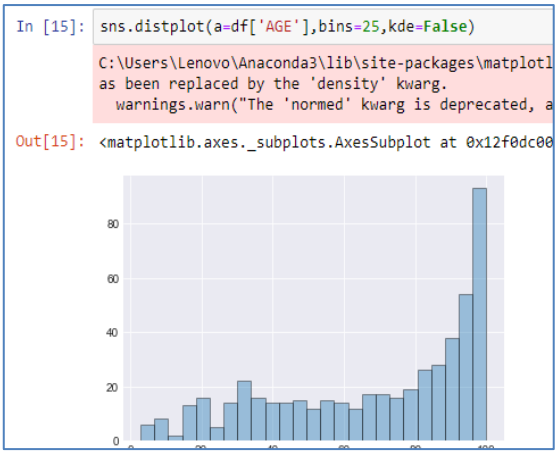
FIGURE 24. Heatmap of features correlation

Correlation values represent the interrelation between features with values ranging from -1 to 1. The higher the value, higher the correlation and vice-versa, a positive one interprets a positive relation (increment in one increases the other) and negative one indicates a negative relation (decrement in one decreases the other).

Based on the heat map, it seems that the feature “**RM**” has a high correlation with the "**Price**” column in a positive way, whereas **B** has a very low correlation

with all other features. Also, “**CRIM**”, which is related to the crime rate, affects the price in a negative way as it should.

The figures 25 and 26, both display the same information, the former in a graphical form and the latter in a numerical form. It can be concluded that the age column (**df[‘AGE’]** from figures 25 and 26) has unbalanced value of 100, which is highly repeated in comparison to other values.



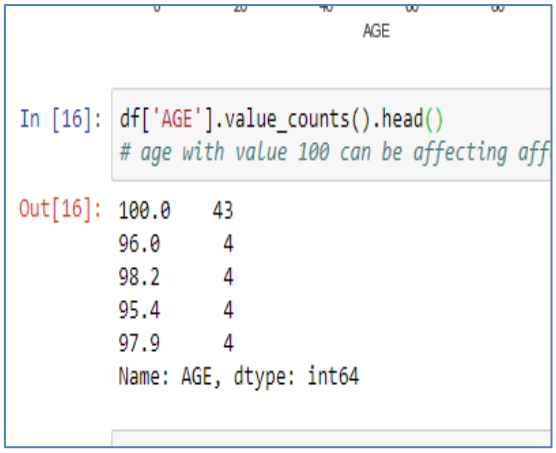


FIGURE 25. Age count plot *FIGURE 26. Age value count*

Other graphical representations that can be used to explore and understand the relations between data are Joint plot, pair plot, KDE plot, Dist plot (a histogram) from Seaborn. Seaborn is built on top of Matplotlib, hence, it is always possible to use the “**PyPlot”** only. If the reader is more interested in learning exploration of data in Python, the Kaggle.com provides useful resources that can be helpful [39].

5.2.3 Splitting Data

The sklearn package provides a module name "**model_selection**” with a submodule “**tain_test_split”** that can be used to split the data. [40]

Figure 27 represents a portion of code used to split the Boston data. The train_test_split submodule divides the data into independent features for training (X_train), independent features for testing (X_test), dependent features for

training (y_train) and dependent data for testing (y_test). The important thing to note is the order of initiating these four variables. The naming of datasets are arbitrary, however, it is a convention of prefix the independent dataset with a capital “X”, followed by a suffix “_train” and “_test” for training and testing data respectively. Similarly, dependent or target features are named as the prefix of small “y” followed by suffixes “_train” and “_test”.

```
In [16]: from sklearn.model_selection import train_test_split

In [17]: X_train,X_test, y_train, y_test = train_test_split(df.iloc[:,:13], df.iloc[:,13], test_size=0.35, random_state=101)
```

FIGURE 27. Data splitting

Pandas offer several methods for subsetting or slicing data frames.

1. Iloc: It is an integer based locating method. [41]
2. Loc: It is the Label (name of column and index) based locating method. [42]
3. Cross Section (xs): It takes the key argument in the multilevel index. [43]

Boston data was divided into train and test data with test data consisting of 35 per cent of the original data extracted randomly. The **df.iloc[: , :13]** returns all rows and independent feature columns i.e. 0 to 12 columns from the beginning since in Python indexing starts from 0 and not from 1 and **df.iloc[: ,13]** returns the dependent feature column “Price”. The attribute “**random_state**” provides the same value through random every time the code is run in a session. It is done for consistency. [40]

5.2.4 Scaling Data

As mentioned in Chapter 4, data must be normalized before training. Figure 28 displays how X_train and X_test were scaled using "StandardScaler" provided by the module "**preprocessing**" of the package sklearn. [44]

```
In [18]: from sklearn.preprocessing import StandardScaler

In [19]: scaler = StandardScaler()

In [20]: X_train = scaler.fit_transform(X_train)
         X_test = scaler.fit_transform(X_test)
```

FIGURE 28. Data scaling

Sometimes the "**fit_transform()"** method is also performed in two steps using "**fit()"** first and then "**transform()**" in the next line. Both steps provide the same result. [44]

5.2.5 Train Model

Sklearn has the module name "**linear_model**" which consists of "**LinearRegression**" estimator which is being imported as shown in figure 29. [45]

```
In [21]: from sklearn.linear_model import LinearRegression

In [22]: lm = LinearRegression()

In [23]: lm.fit(X_train,y_train)
Out[23]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

FIGURE 29. Import and train linear model

After importing any model, it is important to initialize the model. The Input cell 22 initializes linear regression model with the name "**lm**", which is an arbitrary name. The Model is trained by feeding training data in the input cell 23, with the method "**fit()**", which takes two inputs, the independent feature "X_train" and the

dependent feature “y_train” of training data. The output 23 is the linear regression model that was trained. Other parameters, such as **copy_x** and **fit_intercept** are default parameters that were not changed during the initialization. [45]

```
In [25]: lm.fit(X_train,y_train)
Out[25]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [26]: lm.fit(y=y_train, X=X_train)
Out[26]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

FIGURE 30. Default vs custom attribute assignment order

Figure 30 is for a demonstration that the order of the input matters. When the input is not ordered, it is important to relate them to their proper attributes or else an error will occur.

Exploring Results Of Trained Linear Model

The result of the training model can be explored using several methods provided by the linear model package. [45]

In figure 31, the coefficient of linear models was converted to a dataframe for better visualization.

```
In [27]: lm.coef_
Out[27]: array([-0.7492471 ,  0.84808429,  0.38207953,  0.81726119, -1.95369222,
                 2.1292077 ,  0.28364298, -2.60073948,  2.25547258, -1.84486884,
                -1.99800595,  0.6136503 , -4.08474212])

In [28]: cdf = pd.DataFrame(data=lm.coef_, index=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
                'PTRATIO','B', 'LSTAT'], columns=['Coef'])
         cdf
```

Out[28]:

	Coef
CRIM	-0.749247
ZN	0.848084
INDUS	0.382080
CHAS	0.817261
NOX	-1.953692
RM	2.129208
AGE	0.283643
DIS	-2.600739
RAD	2.255473
TAX	-1.844869
PTRATIO	-1.998006
B	0.613650
LSTAT	-4.084742

FIGURE 31. Linear model coefficient in dataframe

It is imperative to realize that algorithm takes one whole row as an input or multiple rows as one input if specified, for instance, in Neural Networks which are out of the scope of this thesis. But algorithms do not take one column at once as an input. From the formula of linear regression (formula 1 and formula 2), it implies that the coefficient plays a detrimental role in the value of y. The negative coefficient causes a decrease in value of prediction value and vice versa. The sign of coefficient and their effects are the same for all algorithms. For instance, in the Boston dataset, the column **CRIM** has a coefficient, " **-0.749247**" which is 1 unit, hence, every decrease in 1 unit of CRIM causes a decrease in the housing price.

Now, for the intercept, i.e. "**c**" in the regression algorithm (formula 1 and formula 2), "**lm.intercept**_", though not in the figure 31, however, was calculated in the code section. When the intercept is zero the line is passing through the origin or vice-versa.

5.2.6 Predictions and Errors

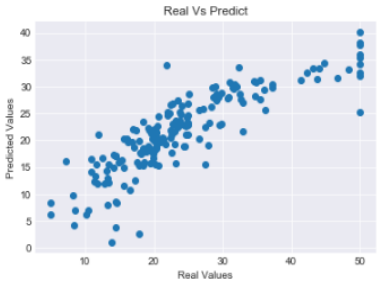
The "**predict()**" method provided by linear regression can be used for the value prediction which takes the X_test as an input. In figure 16, Plot "**Real Vs Predict**" displays the scatter plot of real and predicted values for Boston data. The almost straight slope observed in the scatter plot can be portrayed as the correct choice of using a regression model. The second part of figure 32, errors, which is the difference between real and predicted values, is displayed as a histogram. The bell-shaped graph, also known as the "Bell curve" or the "Gaussian Distribution", implies that outliers are not present in the data.

Predictions

```
In [31]: y_pred=lm.predict(X_test)

In [32]: plt.scatter(y_test,y_pred)
         plt.xlabel("Real Values")
         plt.ylabel("Predicted Values")
         plt.title('Real Vs Predict')

Out[32]: Text(0.5,1,'Real Vs Predict')
```



Errors

```
In [33]: errors = y_test-y_pred

In [34]: errors.hist(bins=20)

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1994598e0b8>
```

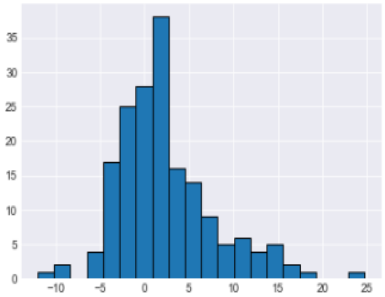


FIGURE 32. Predictions, errors and distribution

5.2.7 Evaluation

Prediction of the linear model for the Boston data was evaluated in figure 33.

```
In [52]: from sklearn import metrics

In [53]: mean_squared_error = metrics.mean_squared_error(y_test, y_pred)

In [54]: mean_squared_error

Out[54]: 35.23773983968171

In [55]: root_mean_squared_error = np.sqrt(mean_squared_error)

In [67]: root_mean_squared_error

Out[67]: 5.936138461970181
```

Root mean square error can be interpreted in y units or dependent variable unit in our dataframe. So, price is our dependent column, hence, error rate is **5.936138461970181** units

```
In [40]: variance_score = metrics.explained_variance_score(y_test,y_pred)

In [41]: variance_score

Out[41]: 0.7117284157045871
```

FIGURE 33. Evaluation of regression model

The “**Metrics**" sub-module provides a method “**mean_squared_error()**” to calculate the mean square error [46]. In the cell 55, the RMSE was calculated using NumPy’s built-in function called “sqrt()” [47].

The “Explained Variance Score” can be interpreted as quality grading of the regression model where the maximum possible value is 1. The higher the score, the better the model and vice-versa. [48]

5.2.8 Optimization

It is time to get back to the preprocessing phase again and observe what are the changes possible for data to improve the model if the decision was made to use the same model. Sometimes, if the evaluation implies that the model is not a good fit for this data, replacing the model with a more complex model is a good choice, too.

In the case of Boston Data, as concluded in figure 24, the feature column "**B**” was dropped and the whole model development process was repeated as demonstrated in figure 34.

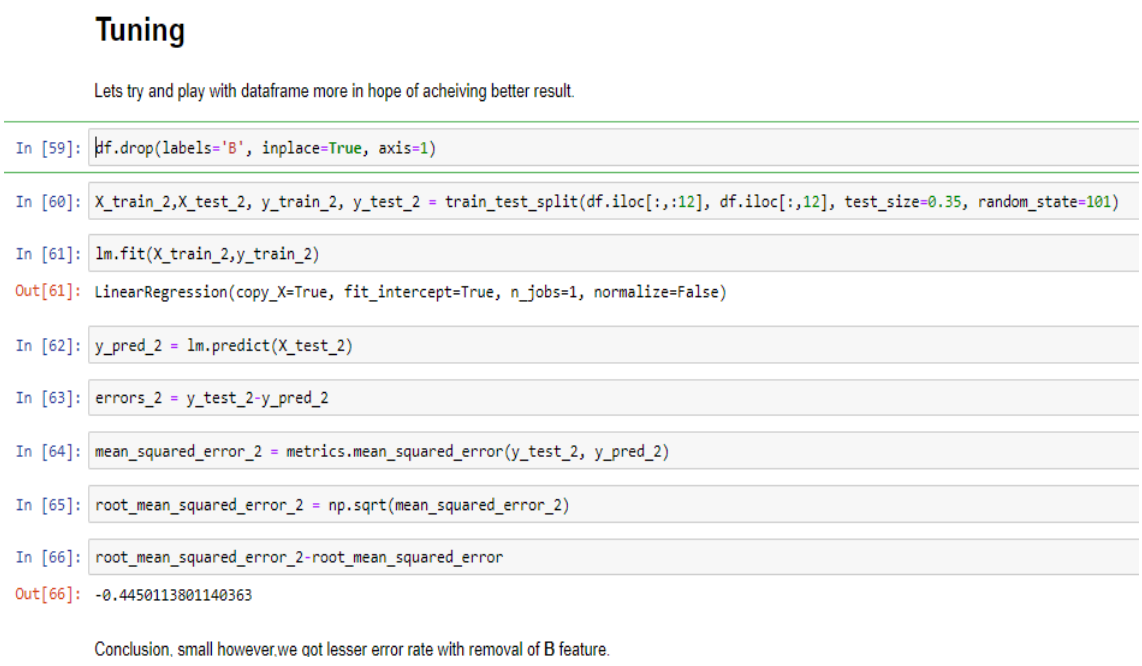


FIGURE 34. Optimizing linear model for boston data

Pandas “**drop()**” can be used to drop columns. It is imperative to note that the “**axis**” attribute is used for targeting either the row or the column with values 0 and 1 respectively.

Dropping the column “B” had very little but still an improvement in the error rate since the difference in RMSE in phases 1 and 2 is positive. This still shows that there are lots of ways to improve the quality of the model, the domain knowledge plays a significant role in tuning.

Figure 35 provides a tip on the different ways of using the drop function with its attribute “**inplace**” when its value is either True or False.

```
In [59]: df.drop(labels='B', inplace=True, axis=1)

In [69]: df = df.drop(labels='B', inplace=False, axis=1)
```

FIGURE 35. Inplace parameter use case

5.3 K-Nearest Neighbor

The steps for Importing the Iris Dataset and Preparing the dataframe are similar to that of Load Boston Datasets that were demonstrated for Regression Algorithm. The full code version provided at the beginning of this chapter demonstrates the pre-processing phase for the Iris Dataset.

5.3.1 Understanding Iris Dataset

Figure 36 displays the plot between the petal length and petal width of all three classes. Apart from the lmplot**,** the pairplot can be used to plot all the numerical columns of data-frame at once to see the relations, but the bigger the database, more time is consumed for plotting. (see the full version of code for pairplot). [49]

```
In [76]: sns.lmplot(x='petal length (cm)', y='petal width (cm)', data=df, hue='Target Class',
                    fit_reg=False, palette='Dark2', markers=['o','x','+'], aspect=2)
         plt.title('Petal Length Vs. Petal Width')
         plt.xlabel('Petal Length (cm)')
         plt.ylabel('Petal Width (cm)')

Out[76]: Text(23.5234,0.5,'Petal Width (cm)')
```

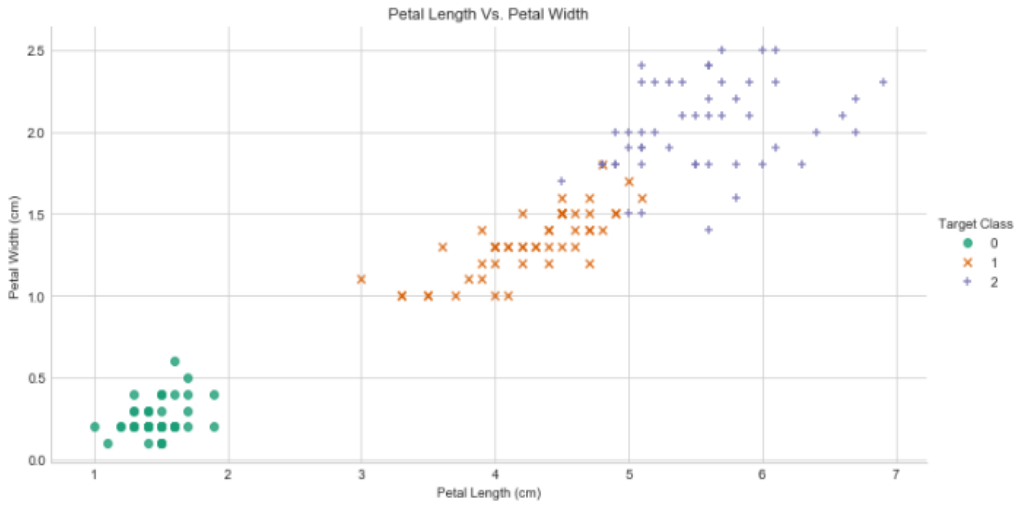


FIGURE 36. Petal length vs. petal width

The figure 36 demonstrates that the target Class 1 and Class 2, which are "**Versicolor**", "**Virginica**" respectively, are more similar since they are gathered together. The target Class 0 "**Setosa**", on the contrary, is quite far and distinguished from other classes. By replacing the values of x and y with the column sepal length and sepal width and running the code from figure 36, the graph will provide a similar result of correlation.

The splitting and scaling of the data were done in a similar fashion as Linear Regression. Hence, the following section starts from Training and Testing Data.

5.3.2 Train Model

```
In [24]: from sklearn.neighbors import KNeighborsClassifier

In [25]: knn = KNeighborsClassifier(n_neighbors=5)

In [26]: knn.fit(X_train, y_train)

Out[26]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                              weights='uniform')
```

FIGURE 37. Train knn model

From the figure 37, "**KNeighorsClassifier**" is the name of the algorithm in the "**neighbors**" family of the sklearn package which is a KNN algorithm being imported in line 24. The initialization of the model is done in the line 25, where the number of neighbors during the first training session is 5, which is also the default value of the "**n_neighbors**" parameter. In the line 26, the model was fitted with train data and the output 26 displays the KNN trained model with all its parameters and their values used by the model.

5.3.3 Prediction and Errors

In the figure 38, the line 27, the variable "**y_pred**" was declared in a similar fashion as the regression algorithm. The line 28, imports Classification Report and Confusion Matrix which are being executed in the line 29. The line 30 prints the tables.

```
In [27]: y_pred = knn.predict(X_test)

In [28]: from sklearn.metrics import classification_report, confusion_matrix

In [29]: cls_report = classification_report(y_test,y_pred)
         confusion_matrix = confusion_matrix(y_test,y_pred)

In [30]: print("Classification Report \n\n",cls_report)
         print('\n')
         print("Confusion matrix \n\n",confusion_matrix)

Classification Report

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.91      0.95        23
           2       0.88      1.00      0.93        14

avg / total       0.97      0.96      0.96        53


Confusion matrix

 [[16  0  0]
 [ 0 21  2]
 [ 0  0 14]]
```

FIGURE 38. Prediction and error knn model

From the index column of the Classification Report, values 0,1 and 2, are the classes in the dependent variable or three species of Iris Flowers. The Classification Report provides a classification report for prediction in each class. And at the bottom row, it gives an average of each metrics for all classes altogether. By looking at the score, the model is already very well trained. In the Confusion Matrix, the correct predictions are the main diagonal values, i.e. in matrix index values at position [0,0], [1,1], [2,2] and the misclassifications are the values other than that of main diagonals.

Hence, the accuracy of KNN model is, Accuracy (formula 6) = (16+21+14) / (16+21+14 + 2) = 0.96 = 96%.

5.3.4 Elbow Method and Grid Search CV

In figure 39, in the line 34 “**avg_error_rates**” has been initialized and calculated. The line 35 outputs the error the result. In classification algorithms, errors are misclassification or cases where the prediction values are incorrect from actual values.

Lets find the optimum value of k using elbow method.

```
In [34]: avg_error_rates = []
         for i in range(1,11):
             knn= KNeighborsClassifier(i)
             knn.fit(X_train, y_train)
             y_pred = knn.predict(X_test)
             avg_error_rates.append(np.mean(y_test != y_pred))

In [35]: avg_error_rates

Out[35]: [0.018867924528301886,
          0.03773584905660377,
          0.018867924528301886,
          0.018867924528301886,
          0.03773584905660377,
          0.018867924528301886,
          0.05660377358490566,
          0.03773584905660377,
          0.05660377358490566,
          0.07547169811320754]
```

FIGURE 39. Error calculation for a range of k

The operation calculates errors for values of K from the range(1,11) i.e. values 1 to 10. In each iteration of the for loop, the KNN algorithm is initialized using the value of “**i**” which goes from 1 to 10. The train and prediction processes are the same as the model train and prediction section. What differs is the last code of the line 31, i.e. **avg_error_rates.append(np.mean(y_test != y_pred))** explained as the following:

1. **y_test != y_pred** checks the error, i.e. if y_test is not equal to y_pred and if not it returns the value Ture.

2. **np.mean(y_test != y_pred))** calculates the mean of all errors, i.e. the mean of all cases where the value True was returned [50].
3. **avg_error_rates.append(np.mean(y_test != y_pred))** adds each average values to avg_error_rates list which is initially empty.

At the end of iteration of the for loop, the average error rates for algorithms, with k values ranging from 1 to 10 respectively are obtained.

The elbow graph plot in figure 40 does not look like an elbow due to irregular error rates, however, it is indeed informative (Chapter 4, Elbow Method). In figure 40, error rates in the y-axis are in an increasing order from bottom to top and k-values in the x-axis are in an increasing order from left to right. As demonstrated in the figure, the error rates from different values of K are not that huge and there is a very small difference. The error rates are very small and almost similar when k values are 1,3,4 and 6. When the value of K is 10, the error rate is the maximum among the given values of K. So, 3,4,6 are the values that can be tested to find optimum results. Using 1 as the value of K not a good idea since 1 number of neighbor can never compare the odds of the data point in question with more than one prediction class.

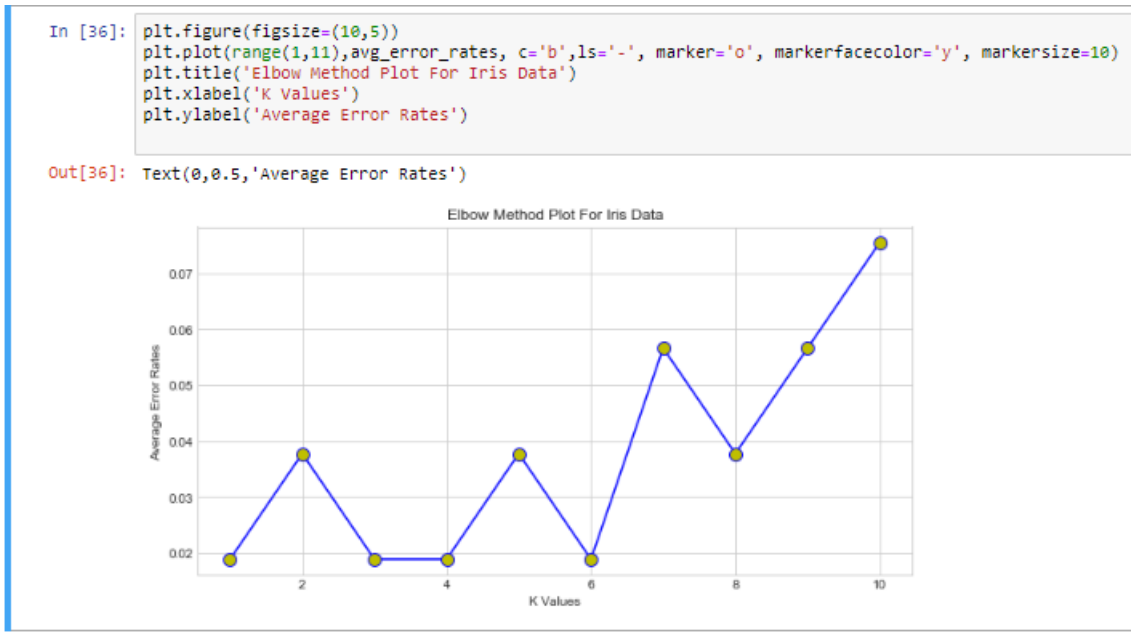


FIGURE 40. Elbow method plot for iris dataset

Grid Search CV is a method of searching the best fit for an estimator. In the grid search, a list of values for different parameters of an algorithm is provided and then the grid iterates through each of them. After the end of execution, GridSearchCV provides the best fit based on the parameters and their value provided for the search.

It is imperative to remember that

1. GridSearchCV is for classification algorithms.
2. It can be used for several Classification algorithms.
3. Parameters to the input are different for different algorithms. The detail information on the parameters is provided on the documentation page of the algorithm.
4. Grid Search uses default values of parameters if options are not provided.
5. In Sklearn the "**model_selection**" module provides the "**GridSearchCV**" method.

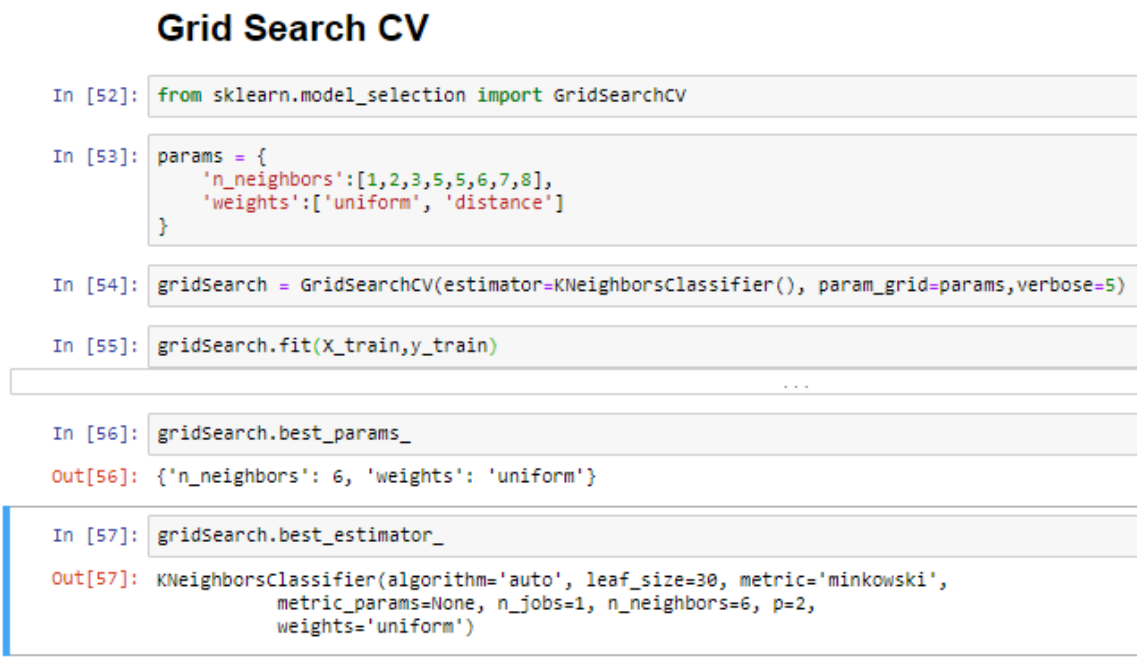


FIGURE 41. Grid search cv

In the figure 41 the cell 52 imports “**GridSearchCV**” from the module “**model_selection**” of the sklearn package. In the line 53, a dictionary Python object named “**params**” was created with it's key and value pairs. The params object has two keys “**n_neighbors**” and “**weights**”. These are the parameters of KNN algorithms. For this experiment, values of the attribute “**k_neighbors**” range from 1 to 8 and the value for the attribute “**weights**” are the two metrics “**uniform**” and “**distance**”. In the Cell 54, the GridSearchCV was initialized, where the attribute “**estimator**” takes the value of “**KNearestClassifier()**” which is the KNN algorithm. It is important to note that the "**estimator**" parameter can take a list of algorithms, not just one if there are other algorithms to search from. The “**parma_grid**” attribute takes the custom object “**params”**. The verbose displays the whole process in the console.

The line 55, fits the GridSearchCV with the training data. After the training, in the line 56, outputs the “**best_params_**”, a built-in property of GridSearchCv, which gives the best parameters from the list of parameters and values provided for search. From the Grid Search, it can be said that the optimal number of K is 6 for the KNN model. In the line 57, the Grid provides the best estimator, a KNN estimator, since the only estimator provided for the “**estimator**” parameter during initializing GridSearchCV was KNN. As mentioned earlier, Grid uses default values for the attributes if their custom value is not provided. If the reader wants to try the result of Grid Search, it can be done in a similar fashion as the train and test processes described in the thesis.

6 CONCLUSION

The thesis aimed to introduce novice readers to the world of Machine Learning and the process of development of Machine Learning model using the Python programming language.

Both the objectives of the introduction to Machine Learning and the phases of Machine Learning in Python were achieved. After reading the paper, the reader will have a better understanding of ML fundamentals. However, unfortunately, the thesis sets a requirement that the reader should be fluent in Python programming to get the best out of the thesis. The thesis used the Linear Regression and the K-Nearest Neighbor algorithms as examples. However, the intuition and mathematical background of these formulae are still not explained in paper detail. Moreover, the algorithms used as examples, are supervised learning algorithms which is only one of the three categories of the ML.

The truth about the “K” in the KNN algorithm was my take away from the paper. Before starting the thesis, I misunderstood the “K” as the number of prediction classes in the dependent column. However, the actual K turned out to be the number of data points to be used in the comparison.

In conclusion, now, the readers can use the Python programming to build the ML models from not only two algorithms explained in the thesis, but a similar procedure can be carried out for any other supervised learning algorithms. The preprocessing phase both the theory and the code can be used to clean any data regardless of the type of estimator being used for the model development. Moreover, the Elbow method can also be used in K-Means Clustering algorithm, a popular unsupervised learning algorithm. Lastly, for the readers who practice Machine Learning using other programming languages, such as Java and R, still can use the concepts explained in the thesis as a basis.

7 FUTURE POSSIBILITIES

Machine Learning is a huge field where lots of fields overlap, for instance, statistics, business, programming. Many years of relentless approach might not be enough to call yourself an expert in the field.

The reader can start practising ML algorithms other than those provided in this thesis, such as Logistic Regression, Decision Trees and SVM (Support Vector Machine) based on the instructions in the paper. Moreover, Kaggle.Com is a website for Data Scientist and Machine Learning Engineers where many datasets are available for practice, many experts in the field can provide tips and reviews about your codes.

Readers who are interested in the mathematics and intuition of several algorithms can take freely available tutorial of Andrew Ng on YouTube. In his tutorial, he explains intuition about various ML algorithms. [51]

REFERENCES

1. Python Tutorial. Python Programming Language. Date of Retrieval 07.03.2019,

https://docs.Python.org/3/tutorial/index.html

2. Swamynathan, M. 2017. Introduction To Machine Learning. Date of Retrieval 07.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

3. Swamynathan, M. 2017. Figure 1. Mastering Machine Learning with Python in Six Steps. Figure 2-9 Types of Machine Learning. Date of Retrieval 09.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

4. Swamynathan, M. 2017. Supervised Learning. Mastering Machine Learning with Python in Six Steps. Date of Retrieval 07.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

5. Guido Sarah, Müller C. Andreas . 2016. Introduction to Machine Learning with Python, 2.1 Classification and Regression. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch02.html#classification-and-regression

6. Guido et al. 2016. Introduction to Machine Learning with Python, Figure 2-11 Predictions of a linear model on the wave dataset. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch02.html#classification-and-regression

7. Swamynathan, M. 2017. Figure 3. Mastering Machine Learning with Python in Six Steps. Figure 3-26 k Nearest Neighbors with k = 5. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

8. Guido et al. 2016. Introduction to Machine Learning with Python, Chapter 3 Unsupervised Learning and Preprocessing. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch03.html#unsupervised-learning-and-preprocessing

9. Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps. Reinforcement Learning. Date of Retrieval 08.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

10. LearnPython. Python Modules and Packages. Date of Retrieval 10.03.2019,

https://www.learnPython.org/en/Modules_and_Packages

11. Numpy Documentation. Date of Retrieval 10.03.2019,

http://www.numpy.org/

12. Pandas Documentation. Date of Retrieval 10.03.2019,

https://pandas.pydata.org/

13. Scikit Learn Documentation. Date of Retrieval 10.03.2019,

https://scikit-learn.org/stable/index.html

14. Matplotlib Documentation. Date of Retrieval 20.03.2019,

https://matplotlib.org/

15. Seaborn Documentation. Date of Retrieval 20.03.2019,

https://seaborn.pydata.org/

16. Swamynathan, M. 2017. Figure 4. Mastering Machine Learning with Python in Six Steps, Logical Flow of data in Machine Learning model building. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

17. Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Feature Engineering. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

18. Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Dealing With Missing Data. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

19. Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Normalizing Data. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

20. Sklearn. Figure 4. Scikit Learn algorithm cheat-sheet. Date of Retrieval 15.04.2019,

https://scikit-learn.org/stable/_static/ml_map.png

21. Sklearn, Confusion Matrix. Confusion Matrix Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

22. Swamynathan, M. 2017. Figure 6. Mastering Machine Learning with Python in Six Steps, Figure 3-11 Confusion Matrix. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

23. Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Table 3-11 Classification performance matrices. Date of Retrieval 10.05.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

24. Sklearn, Precision, Recall, and F-score. Precision_recall_fscore_supprot Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support

25. Sklearn, Load Boston Dataset. Details on Load Boston Dataset. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html

26. Sklearn, Iris Dataset. Details on Load Iris Dataset. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris

27. Pandas, Read CSV, Read_CSV Documentation. Date of Retrieval 10.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

28. NumPy, Reshape. Reshape Documentation. Date of Retrieval 11.05.2019,

https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html

29. Pandas, FillNa. Pandas.fillna Documentation. Date of Retrieval 10.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html

30. Pandas, DropNa. Pandas.DropNa Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html

31. Pandas, Value Counts. Pandas.Series.value_counts Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html

32. Sklearn, Label Encoder. Sklearn.Preprocessing.LabelEncoder Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

33. Sklearn, One Hot Encoder. Sklearn.Preprocessing.OneHotEncoder Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

34. Scipy, ToArray. Scipy.Sparse.csr_matrix.toarray Documentation. Date of Retrieval 10.05.2019,

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.toarray.html

35. Pandas, Get Dummies. Pandas.get_dummies Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

36. Sklearn, Load_Boston User Guide. Boston Housing Price Dataset User Guide. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/datasets/index.html#boston-dataset

37. Pandas, Describe. Pandas.Describe Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html

38. Pandas, Info. Pandas.Info Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.info.html

39. Pedro Marcelino, Kaggle. Python Data Exploration Tutorials. Date of Retrieval 15.05.2019,

https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-Python

40. Sklearn, Train_Test_Split. Sklearn.Model_Selction.Train_Test_Split. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

41. Pandas, Iloc. Pandas.Iloc Documentation. Date of Retrieval 15.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.iloc.html

42. Pandas, Loc. Pandas.loc Documentation. Date of Retrieval 15.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html

43. Pandas, XS. Pandas.XS Documentation. Date of Retrieval 15.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.xs.html

44. Sklearn, Standard-Scaler. Sklearn.Preprocessing.StandardScaler Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

45. Sklearn, Linear Regression. Sklearn.linear_model.LinearRegression Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

46. Sklearn, Mean Squared Error. Sklearn.Metrics.Mean_Squared_Error Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

47. NumPy, Square Root. Numpy.Sqrt Documentation. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Errors.png

48. Sklearn, Explained Variance Score. Sklearn.Metrics.Explained_Variance_Score Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.explained_variance_score.html

49. Seaborn, LmPlot. Seaborn.LmPlot Documentation. Date of Retrieval 15.05.2019,

https://seaborn.pydata.org/generated/seaborn.lmplot.html

50. NumPy, Mean. NumPy.Mean Documentation. Date of Retrieval 15.05.2019,

https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html

51. Andrew Ng. Machine Learning – Andrew Ng, Standford University [Full Course]. Date of Retrieval 20.05.2019,

https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN