Using Machine Learning techniques with Python

Undergraduate Thesis

Christina Konstantinidou (dai19042)

Department of Applied Informatics

University of Macedonia

Supervising Professor: Nikolaos Samaras

This dissertation is submitted for a Bachelor's degree.

University of Macedonia, January 2023

Περίληψη

Η μηχανική μάθηση είναι ένας συνεχώς αναπτυσσόμενος τομέας της τεχνητής νοημοσύνης. Πολλοί αλγόριθμοι μηχανικής μάθησης βρίσκονται σε απλές εφαρμογές της καθημερινότητας μας, οπως για παράδειγμα στον βέλτιστο τρόπο φόρτισης μιας μπαταρίας κινήτου ή αυτοκινήτου. Σκοπός αυτής της πτυχιακής εργασίας είναι η μελέτη και σύγκριση τεχνικών μηχανικής μάθησης με την χρήση της γλώσσας προγραμματισμού Python. Αρχικά πραγματοποιήθηκε βιβλιογραφική έρευνα με στόχο την διερεύνηση των μεθόδων και την καταγραφή των αλγορίθμων μηχανικής μάθησης και των στοιχειωδών εννοιών της. Στην συνέχεια, παρουσιάσαμε όλες τις έννοιες που συνδέονται με την μηχανική μάθηση, καθώς και τι είναι η Python και γιατί την προτιμάνε οι επιστήμονες. Τέλος, υλοποιήσαμε όλα τα παραπάνω σε τρεις διαφορετικές συλλογές πληροφοριών, εξηγήσαμε την διαδικάσια που ακολουθήσαμε για την κάθε μια και συγκρίναμε τα αποτελέσματά τους και τις διαφορές τους. Όσον αφορά στα εργαλεία που προσφέρει η Python, διαπιστώθηκε η αποτελεσματικότητά τους και το αντίκτυπο που έχουν για τους μηχανικούς μηχανικής μάθησης.

Λέξεις Κλειδιά: Μηχανική Μάθηση, Επιτηρούμενη μαθηση, Αυτοκίνητα, Python, Βιβλιοθήκες, Αλγόριθμος

Abstract

Machine learning is an ever-growing field of artificial intelligence. Many machine learning algorithms are found in simple applications in our daily lives, such as the optimal way to charge a cell phone or a car battery. The purpose of this dissertation is to study and compare machine learning techniques using the Python programming language. Initially, a bibliographic research was carried out with the aim of investigating the methods and the recording of machine learning algorithms and its basic concepts. Next, we presented all the concepts related to machine learning, as well as what Python is and why scientists prefer it. Finally, we implemented all of the above using three different datasets, explained the process that was followed for each on of them and compared their results and differences. As for the tools offered by Python, their effectiveness and impact for machine learning engineers has been established.

Keywords: Machine Learning, Supervised Learning, Cars, Python, Libraries, Algorithm

Table of Contents

Code Index

Illustrations Index

1. Introduction

People are often prone to making mistakes during analyses or, possibly, when trying to establish relationships between multiple features, which renders finding solutions to particular problems quite challenging. This is where Machine learning offers a useful and even essential advantage. Specifically, machine learning is a rapidly growing field of artificial intelligence that enables computers to learn from data by drawing inferences from patterns without being explicitly programmed. It involves developing algorithms and statistical models that allow a computer to make predictions and decisions or take actions based on input data. It can often be successfully applied to problems, improving the efficiency of systems and the design of machines. Python is one of the most popular programming languages for machine learning due to its simplicity and ease of use, and therefore it is establishing itself as one of the most popular languages for scientific computing.

1.1 Goals

The aim of this dissertation is the exploration of different tools that are used in machine learning to facilitate coding and make the results more efficient. Following plenty of research, I will present various machine-learning techniques and how they can be implemented using the Python programming language. Precisely, I will cover topics such as supervised and unsupervised learning as well as popular machine learning libraries such as sci-kit-learn and Pandas. I will explain the methods applied in the preprocessing part, how they were applied to several datasets and why, and I will present the reached conclusions. By the end of this dissertation, a solid understanding of how to apply machine learning techniques to real-world problems using Python will be built.

1.2 Structure

This dissertation consists of six (6) chapters. First and foremost, in the first chapter, there will be a brief introduction to machine learning, the aim of this thesis, and its structure. In the second chapter, there will be a thorough presentation of machine learning, including the most basic terms and parts of the whole process. In addition, the third chapter will introduce the most popular machine learning algorithms and their unique characteristics. The fourth chapter explains the significance of the use of Python in correlation with a few unique and easy-to-use

libraries. Last but not least, the fifth chapter describes the methodology followed for the manipulation of the individual datasets and the creation of the machine learning models after the research had been conducted.

2. Machine learning

In the mid-twentieth century, machine learning emerged as a subset of Artificial Intelligence. It was inspired by a conceptual understanding of how the human brain works. Machine Learning focuses on creating computer programs and machines that are capable of performing tasks that humans are naturally good at, including natural language understanding, speech comprehension, and image recognition (Raschka and al, 2020). Today, machine learning remains deeply intertwined with AI research. However, it is often more broadly regarded as a scientific field that focuses on the design of computer models and algorithms that can perform specific tasks, often involving pattern recognition, without the need to be explicitly programmed.

Being a subdomain of Artificial Intelligence, Machine Learning is likewise intelligent. Specifically, machine learning finds solutions by adapting to its changing environment and learning from it (Alpaydin, 2020). It is capable of recognizing patterns, learning from experience, abstracting new information from data, or optimizing the accuracy and efficiency of its processing and output. In this way, the designer of the algorithm does not need to foresee all the possible situations and find solutions for them since the machine learning algorithm does just that. Ultimately, machine learning is evolving from teaching computers to mimicking the human brain.

Machine learning's aim is to create algorithms that allow the computer to handle the data more efficiently. Its learning is a process of finding statistical regularities or other data patterns since sometimes one fails to interpret the extracted information from the data. The applied machine learning model is built upon specific parameters and is constantly learning. There are actually two types of models. A predictive model, which makes predictions for the future, and a descriptive one, which gains knowledge, based on past experiences. A model can combine both of those types.

The learning process in a simple machine-learning model consists of two steps: training and testing. As a result, data are also divided into training and testing data. In the training process, training data are used as input, with the help of which the learning algorithm builds the learning model, and, in the testing process, the learning model that is already built makes the predictions.

Machine learning also relies on multiple algorithms to solve data problems since no algorithm can fit perfectly into every problem. Precisely, the type of algorithm used is chosen by the designer based on different parameters, such as the number and type of variables needed and the kind of problem that needs to be solved. The purpose of this process is to find the most efficient learning algorithm to complete the task in a more optimized form.

Based on the papers of Nasteski and Mahesh, machine learning algorithms are divided into six categories, depending on the desired outcome. These categories are : a) Supervised learning, b) Unsupervised learning, c) Semi-supervised learning, d) Reinforcement learning, where the training information provided is a scalar reinforcement signal, and the learner is not advised which actions to take but instead must discover which actions yield the best reward. e) Transduction, f) Self-Training g) Multitask Learning and h) Neural Networks. Nevertheless, machine learning algorithms are categorized into two general groups, supervised and unsupervised learning.

2.1 Supervised learning

The main characteristic of supervised learning is that any dataset used by machine learning algorithms is portrayed using the same set of features. In supervised algorithms, the classes are predetermined. This means that the input is given with known labels (the corresponding correct outputs), which are defined by humans and are created as a result of the whole set. The so-called labels are defined as class labels in the classification process. The various algorithms generate a function that maps inputs into desired outputs. One standard formulation of the supervised learning task is the classification problem: the learner is required to learn (to approximate the behavior of) a function that maps a vector into one of several classes by looking at several input-output examples of the function (Nasteski, 2017).

Supervised machine learning is considered one of the most significant categories of computer programming since it drives multiple business applications and significantly affects our day-to-day lives. Supervised learning is also the most common technique for training neural networks and decision trees, both of which are dependent on the information given by the pre-determinate classification.

Supervised learning is the most common technique in classification problems since the aim is often to teach the machine a classification system that the programmer has created. Most commonly, supervised learning can leave the possibility of the input being undefined, only if the expected output is given. As a result, this type of learning provides a structured dataset which consists mainly of features and classes. The main purpose of supervised learning is to build an estimator, which can predict the class of an object based on a specific combination of the given features. Consequently, the learning algorithm receives a set of features as inputs along with the correct outputs and it learns by comparing its actual output with corrected outputs to calculate the error and modify the model accordingly.

There are many practical examples of supervised learning. An application that predicts the species of iris given a set of measurements of its flower is a case in

point. The supervised learning tasks are divided into two categories: classification and regression.

The task of the machine learning algorithm is to find patterns and construct mathematical models. These models are then evaluated based on the predictive capacity concerning measures of variance in the data itself.

2.2 Unsupervised learning

There are many differences between supervised and unsupervised learning. First and foremost, contrary to supervised learning, where instances are labeled, unsupervised learning uses unlabeled instances. This means that the learning algorithm is not provided with classifications. Additionally, the main task of unsupervised learning is to automatically develop classification labels. The algorithms use unstructured and raw data in the hopes of discovering unknown but useful classes of items. These algorithms are also searching similarities between pieces of data to determine if they can be categorized and create a group.

These groups are called clusters and represent a whole family of clustering machine-learning techniques. In this unsupervised classification (cluster analysis), the machine doesn't know how the clusters are being grouped, and therefore when using cluster analysis, there is a greater potential for surprising ourselves, which ultimately renders cluster analysis a promising tool for the exploration of relationships between any instances.

Recently, there has been a rising trend of employing unsupervised machine learning in order to improve network performance and provide services, such as quality of service optimization. In addition, unsupervised learning can unconstrain the users from the need for labeled data thereby facilitating flexible, general, and automated methods of machine learning (Usama et al, 2019).

2.3 Training data and Test data

Additionally, as previously stated, the learning process is divided into two categories: training and testing. In this case, in the first part, the training, the algorithm learns and develops based on the training set of information. What was not stated before is that the observations of the model in the training set can compromise the experience that the algorithm uses to learn. In supervised learning problems,

each of these observations relies on a single response variable and is built on one or more explanatory variables.

The test set is a similar collection of observations that are used to evaluate the performance of the model using some performance metric. No observations from the training set must be included in the test set since were that to be the case it would be a challenge to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

A program that generalizes well will be able to effectively perform the same task with new data. In contrast, a program that memorizes the training data by learning an overly complex model could predict the values of the response variable for the training set accurately but will fail to predict the value of the response variable for new examples. For example, if the input values are far from each other and the algorithm cannot find a steady tendency, the algorithm may learn the exact results of the training set and try to apply it to the test set.

Memorizing the training set is called over-fitting. A program that commits over-fitting may not perform its task well, as it could memorize relations and structures that are noise or coincidence. Balancing memorization and generalization, or overfitting and underfitting is a common problem to many machine learning algorithms.

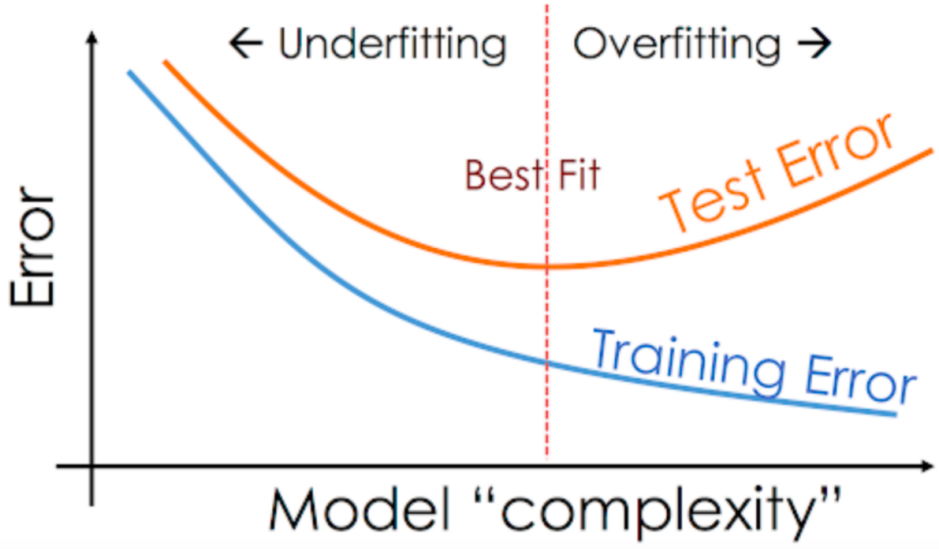

Illustration 2.1 Overfitting - Underfitting graph

In addition to the training and test data, a third set of observations called a validation or hold-out set is sometimes required. The validation set is used to tune variables called hyperparameters, which control how the model is taught. The program is still evaluated on the test set to provide an estimate of its performance in the real world. The performance of the learning algorithm on the validation set should

not be used as an estimate of the model's real-world performance since the program has been tuned specifically to the validation data.

It is common to partition a single set of supervised observations into training, validation, and test sets. There are no requirements for the sizes of the partitions, and they may vary according to the amount of data available. It is usual to split 50 percent or more of the data for the training set, 25 percent for the test set, and the remainder for the validation set. In this paper, there is no validation set and that is why the partition of the input data is 80 percent training set and 20 percent test set.

2.4 Performance measures

After the test process, we have to evaluate the efficiency of the algorithm. Thus, we use metrics in order to find out how efficiently the algorithm works and measure its performance. For supervised learning problems, many performance metrics measure the number of prediction errors. There are two fundamental roots of prediction error: a model's bias and its variance. Supposing that we have many training sets that are all unique but equally representative of the population, a model with a high bias would produce similar errors for an input regardless of its training set.

The model makes its own assumptions about the connection between the training data. A model with high variance will produce different errors for an input depending on its traning set. This type of data set may be flexible, but it creates noise in the training set. In contrast, a model with high bias is inflexible, but many of its inputs can lead to the same result. In consequence, a model with high variance overfits the training data, while a model with high bias underfits the training data.

There are some extra metrics we use to measure the effectiveness depending on the type of algorithm. For example, in regressional algorithms, we calculate the mean square error, whose value can vary from zero to infinity. For this reason, the value of the mean square error of each problem is relevant to the variety of inputs. The combination of these measures the performance of regression algorithms and they will be discussed later in this paper.

In classification algorithms, one can find the confusion matrix as a performance metric. A confusion matrix is a table that displays the efficiency of a supervised learning algorithm. Every column of this table represents the possible outcomes of a prediction, and every row represents the actual outcome.

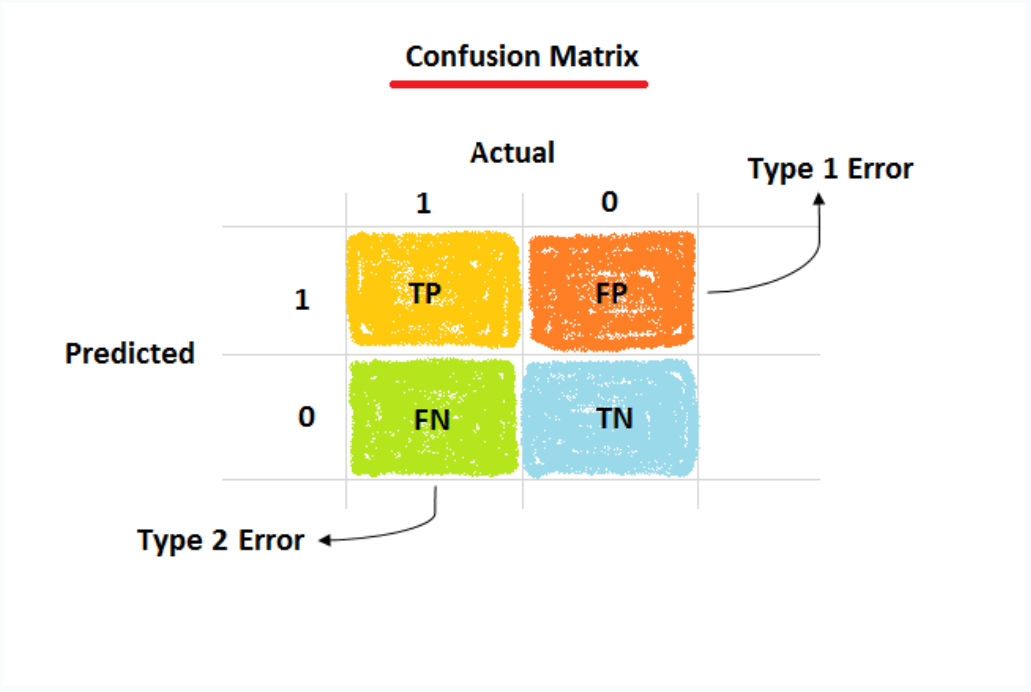

Illustration 2.2 Confusion matrix

- True Positive (TP) is the number of true positive cases.
- False Positive (FP) is the number of cases that are predicted as true but in reality, they are false.
- True Negative (TN) is the number of cases that are predicted as false but in reality, they are true.
- False Negative (FN) is the number of true negative cases.

Based on the confusion matrix, an engineer can calculate the following metrics, which are also considered performance metrics for supervised learning:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Illustration 2.3 Accuracy, precision, recall, and F1 score

3. Machine learning algorithms

As previously mentioned in this paper, the algorithm that the programmer uses depends on many parameters, such as the desired outcome, the structure of the input, and the number of data, to name a few. Therefore, based on those parameters, the programmers can choose between many algorithms to use, or they can even try to use more than one algorithm in order to decide the one that works most efficiently for a specific case. Different algorithms give the machine different learning experiences and focus on other relations from the input, after which the machine makes a decision and performs specialized tasks. In this paper, we are going to analyze the most popular of those algorithms, the experience they offer, and their differences.

3.1 Linear Regression

Simple linear regression can be used to model a linear relationship between one dependent variable and one independent variable. As is evident in the graph below, the machine creates a line in correspondence between the results and the variables as an output.

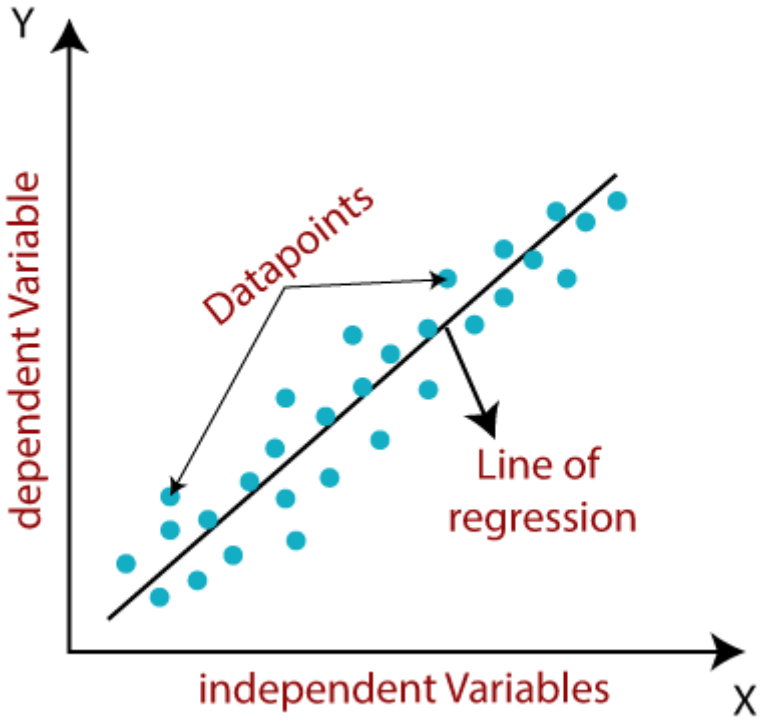

Illustration 4.1 Linear regression graph

3.2 Decision Tree

Decision trees are tree-like graphs that model a decision. They are used to predict results and draw conclusions about a set of observations. Additionally, they are commonly learned by recursively splitting the set of training instances into subsets based on the instances' values for the explanatory variables. The nodes in the graph represent an event or choice and its edges represent the decision rules or conditions. Each tree consists of nodes and branches. Each node represents attributes in a group that are to be classified and each branch represents a value that the node can take (Mahesh, 2018).

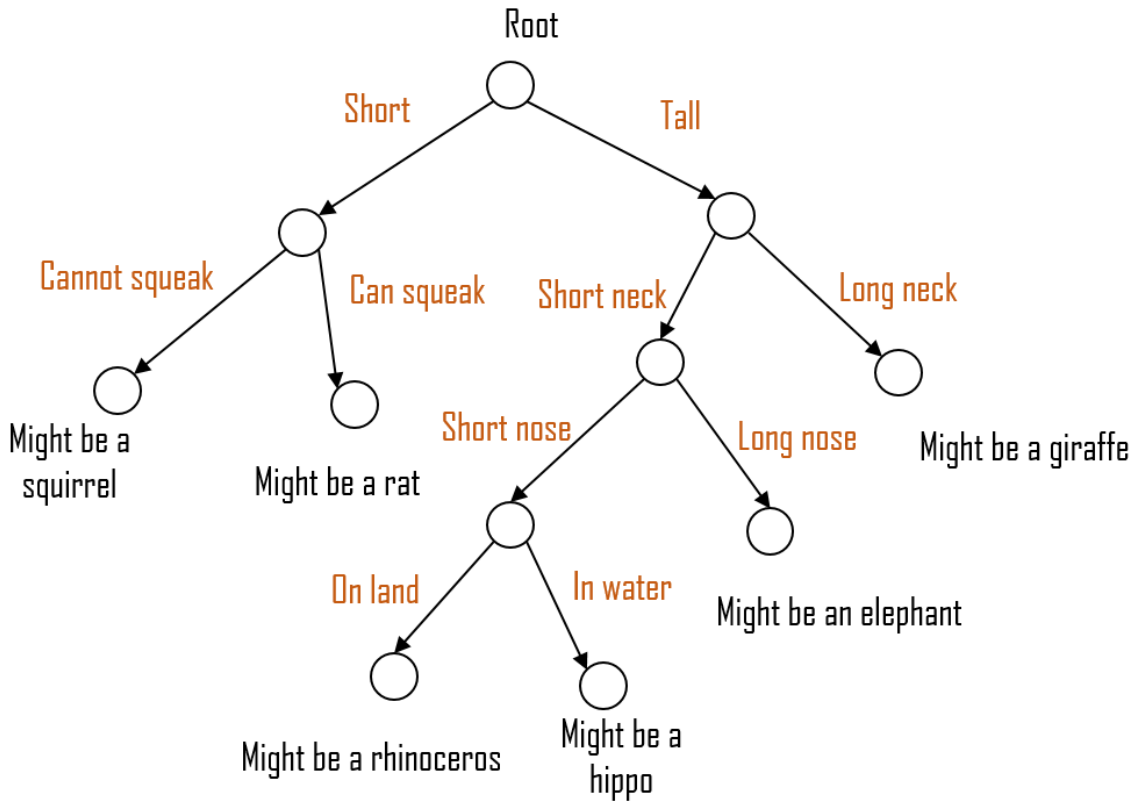

Illustration 4.2 Decision tree graph

3.3 Random Forest

Random Forest is a collection of decision trees that have been trained on randomly selected subsets of the training instances and explanatory variables. Random forests usually make predictions by returning the mean of the predictions of their composing trees. Scikit-learn's implementations also return the mean of the trees' predictions. Random forests are less prone to overfitting than decision trees because no single tree can learn from all instances and explanatory variables and memorize all the noise in the representation. But, their accuracy is lower than decision trees.

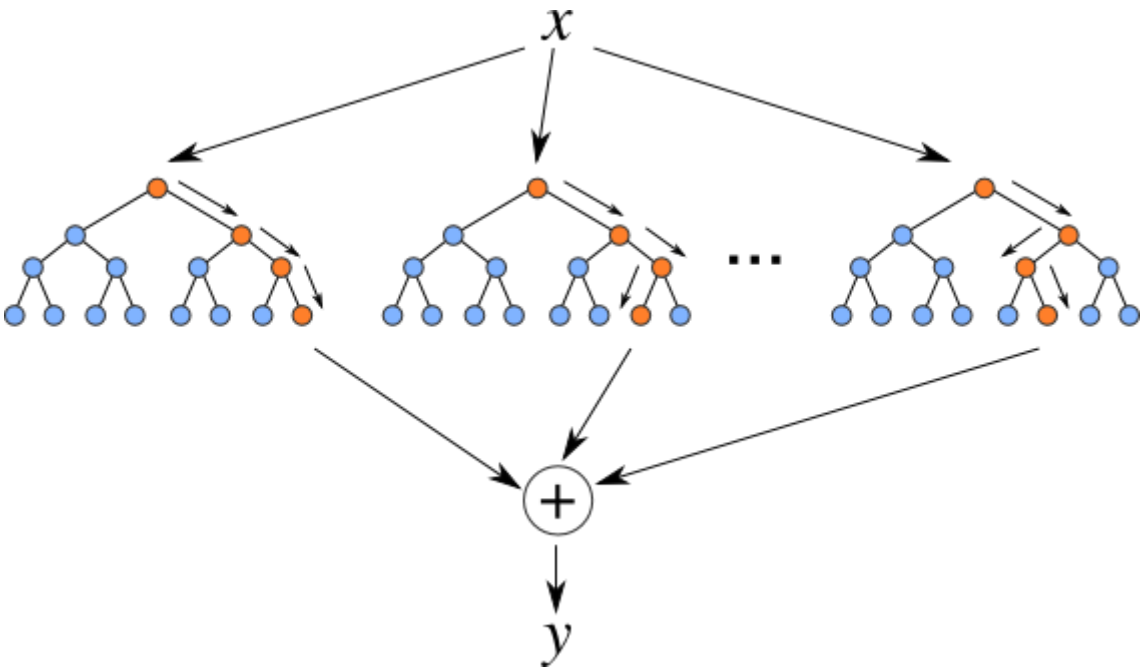

Illustration 4.3 Random Forest graph

3.4 K-means

K-means is an algorithm that aims to divide n observations into k clusters. The cluster’s center, the mean of n elements, is called the centroid, and it serves as a prototype for the cluster. So, each observation is categorized into a cluster based on the nearest mean or else centroid. This results in a partitioning of the data space into the Voronoi cells.

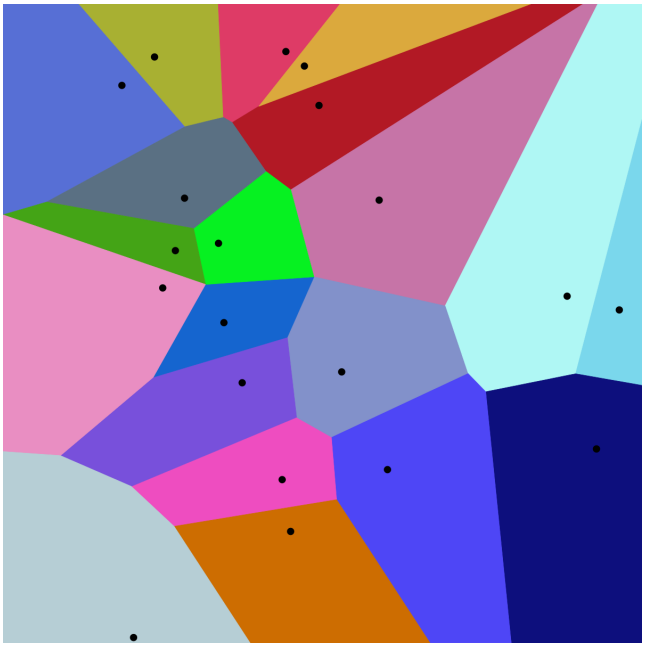

Illustration 4.4 Voronoi cells

The unsupervised k-means algorithm often and mistakenly passes as the k-nearest neighbor classifier, a popular supervised machine learning technique for classification, due to its name. K-means clustering is rather easy to apply to even large data sets. It has been successfully used in market segmentation, computer vision, and astronomy among many other domains and can be used as a preprocessing step for other algorithms.

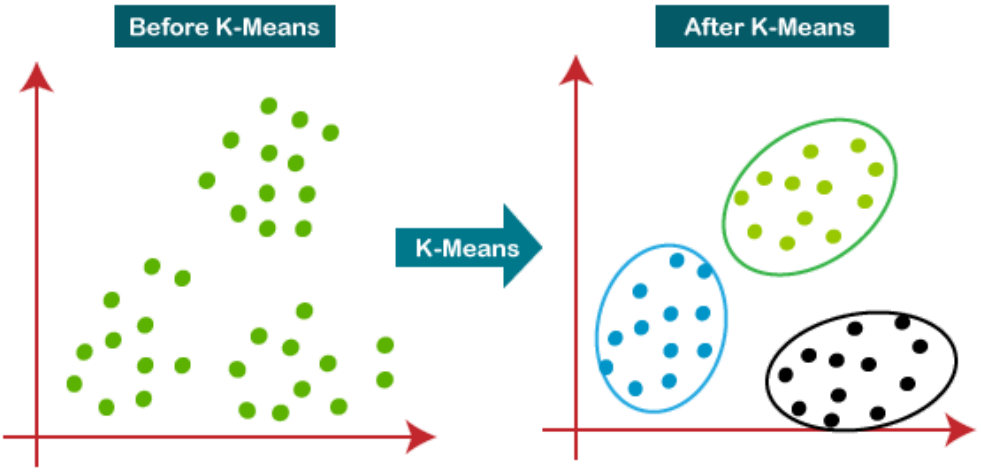

Illustration 4.5 K-means graph

3.5 Apriori

Apriori is an algorithm that uses association rule learning over relational databases. To be more thorough, it proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori (the circles outside the dotted line, illustrated below) can be used to determine association rules. The infrequent items are excluded. This algorithm uses unsupervised learning and has applications in domains such as market basket analysis.

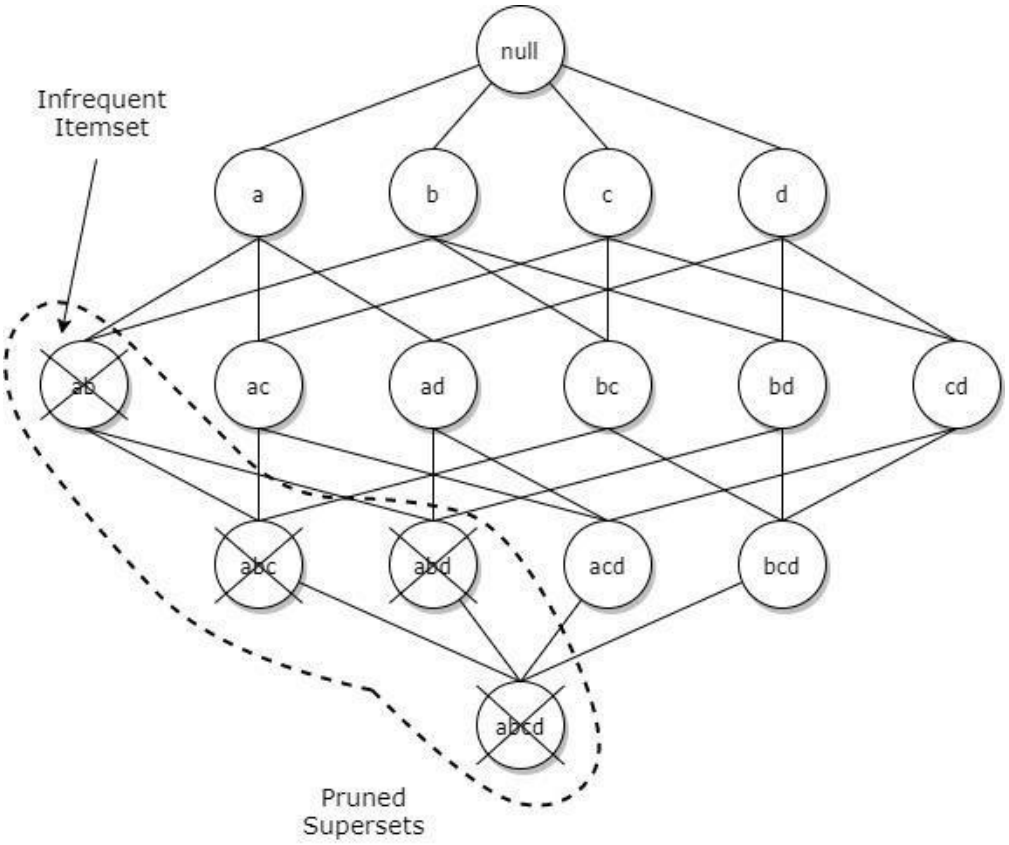

Illustration 4.6 Apriori graph

4. Basic libraries in Python for Machine learning

Historically, a wide range of different programming languages and environments have been used to enable machine learning research and application development. However, as the Python language has seen tremendous growth in popularity within the scientific computing community in the last decade, most recent machine learning and deep learning libraries are now Python-based. With its core focus on readability, Python is a high-level interpreted programming language, which is widely recognized for being easy to learn yet still able to harness the power of systems-level programming languages when necessary. Aside from the benefits of the language itself, the community around the available tools and libraries makes Python particularly attractive for workloads in data science, machine learning, and scientific computing. According to a recent KDnuggets poll that surveyed more than 1800 participants for preferences in analytics, data science, and machine learning, Python maintained its position at the top of the most widely used language in 2019 (Piatetsky, 2019).

The Python programming language provides a rich set of high-level data structures: lists for enumerating a collection of objects, dictionaries to build hash tables, and so forth. However, these structures are not ideally suited to high-performance numerical computation (Stéfan van der Walt et al., 2011). This is why machine learning with Python does not work independently. The following are the most useful and well-known libraries in Python that every single programmer uses for machine learning. At the end of this paper, the reader will be able to see the tremendous difference these libraries can make, in the context of time and difficulty.

4.1 Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms basic linear algebra, basic statistical operations, random simulation and much more. But, the most important thing to remember about NumPy is that it supports efficient operations on large arrays and multidimensional matrices and when skillfully applied, a loss of computational time is achieved rather than spending time in Python for-loops.

4.2 Pandas

Pandas is a Python library and the main tool in Python for data analysis and data handling. The library provides integrated, intuitive processes for performing common data manipulations and analysis on data sets. It aims to be the foundational layer for the future of statistical computing in Python and serves as a strong complement to the existing scientific Python stack while implementing and improving upon the kinds of data manipulation tools found in other statistical programming languages such as R.

The Pandas library, under development since 2008, is intended to close the gap in the richness of available data analysis tools between Python, a general purpose systems and scientific computing language, and the numerous domain-specific statistical computing platforms and database languages. It not only aims to provide equivalent functionality but also implement many features, such as automatic data alignment and hierarchical indexing, which are not readily available in such a tightly integrated way in any other libraries or computing environments. The library's name derives from panel data, a common term for multidimensional data sets encountered in statistics and econometrics.

Pandas makes it simple to do many of the time-consuming, repetitive tasks associated with working with data, including:

- Data cleansing
- Data fill
- Data normalization
- Merges and joins
- Data visualization
- Statistical analysis
- Data inspection
- Loading and saving data
- And much more

4.3 Scikit

Since its release in 2007, scikit-learn has become one of the most popular machine-learning libraries for Python. Scikit-learn exposes a wide variety of machine learning algorithms, both supervised and unsupervised, using a consistent, task-oriented interface and thus enabling easy comparison of methods for a given

application (Pedregosa et al, 2011). It also provides modules for extracting features, processing data, and evaluating models.

Scikit-learn is a library built on other popular libraries: Numpy and matplotlib. Scikit-learn is prominent for academic research because it is well-documented and easy to use. Developers can use scikit-learn to experiment with different algorithms by changing only a few lines of the code. Scikit-learn also includes a variety of datasets, allowing developers to focus on algorithms rather than obtaining and cleaning data. Finally, scikit-learn is noted for its reliability because the biggest part of the library is covered by automated tests.

To find the right balance between providing useful features and the ability to maintain high-quality code, the Scikit-learn development team only includes well-established algorithms in the library. However, the explosion in machine learning and artificial intelligence research over the past decade has created a great number of algorithms that are best left as extensions rather than being integrated into the core. Newer and often lesser-known algorithms are contributed as Scikit-learn compatible libraries or so-called “Scikit-contrib” packages, where the latter are maintained by the Scikit-learn community under a shared GitHub organization, “Scikit-learn-contrib” (https: //github.com/scikit-learn-contrib). When these separate packages follow the Scikit-learn API, they can benefit from the Scikit-learn ecosystem, providing users the ability to inherit some of Scikit-learn’s advanced features, such as pipelining and cross-validation, for free.

4.4 Matplotlib

The last library to mention in this dissertation is the matplotlib library. Matplotlib is a Python library for 2D plotting that generates production-quality graphs based on the information of the dataset. It supports interactive and non-interactive plotting, can save images in several output formats, and provides a wide variety of plot types (lines, bars, pie charts, histograms, and many more). In addition to this, it is highly customizable, flexible, and easy to use.

5. Implementation

In this chapter, I am going to present my implementation of all of the above. First and foremost, I used three different datasets related to cars, and I wrote my own code. Each of these datasets required different steps and methods in order to reach the intended result. I will follow through step by step with the actual code I wrote and the related results. Some steps were identical every time, and others were different, especially in the last dataset, which I had researched a lot, and therefore it was easier to handle the problems in smarter and quicker ways. Actually, this is something that everyone can notice in my code, showcasing my progress while doing this process over and over again.

All of the datasets contain labeled data, which necessarily means that I am using supervised learning and providing the machine with only supervised algorithms. However, each one of them contains different information, and the needed output is different. An important thing to note is that almost every job is implemented with the use of functions that the libraries offer.

For this reason, the first steps for anyone involved in machine learning are the following:

- Import the basic libraries. These libraries are the ones we analyzed above, pandas, numpy, matplotlib, and sklearn.
- Import the data set. This is easily implemented with the use of pandas.
- Print the first few lines of the dataset.
- Print the type of each column.

```
import pandas as pd
import numpy as np
import matplotlib
from sklearn import datasets

df = pd.read_csv (r'/content/data.csv')
print (df)
print(df.head()) // prints the first 5 rows of every column
print(df.columns) // prints the name of each column
```

Snippet of code 6.1 First steps of machine learning

5.1 First dataset (Car - Price)

The first dataset was also the most difficult one. Its contents are a series of characteristics of cars and the needed output, which is the price of the car based on these characteristics. So, our goal is to build a model that can predict the price of the car based on the given characteristics. The dataset contains 11,914 rows and 16 columns, which makes it the biggest of the three datasets. As you can see below (Illustration 6.1), the types of data were different in each column, meaning there were both numeric and categorical values involved. The input of this algorithm contains 15 characteristics of cars, and the needed output is the price of the car (the 16th column).

Make	Model	Year	Engine Fu...	Engine HP	Engine Cyl...	Transmissi...	Driven_Wh...
BMW	1 Series M	2011	premium unleaded (required)	335	6	MANUAL	rear wheel drive
BMW	1 Series	2011	premium unleaded (required)	300	6	MANUAL	rear wheel drive
BMW	1 Series	2011	premium unleaded (required)	300	6	MANUAL	rear wheel drive
BMW	1 Series	2011	premium unleaded (required)	230	6	MANUAL	rear wheel drive
BMW	1 Series	2011	premium unleaded (required)	230	6	MANUAL	rear wheel drive
BMW	1 Series	2012	premium unleaded (required)	230	6	MANUAL	rear wheel drive
BMW	1 Series	2012	premium unleaded (required)	300	6	MANUAL	rear wheel drive
BMW	1 Series	2012	premium unleaded (required)	300	6	MANUAL	rear wheel drive

Illustration 6.1 First 8 columns of car-price dataset

5.1.1 Statistics

The second step of our process is to find the statistics of this dataset. As mentioned before, this is a large dataset, and the statistics will help us read through it in order to be able to manipulate it later. This snippet of code is pure python coupled with functions from numpy. I found the statistics only for the numeric columns.

```
for index in range(16):
    if index == 2 or index == 4 or index == 5 or index == 8 or index == 12
or index == 13 or index == 14 or index == 15:
        sum_col = df.iloc[:,index]
        print ("Mean ", index, " : % s" % (np.mean(sum_col)))
        print ("Max ", index, " : % s" % (max(sum_col)))
        print ("Min ", index, " : % s" % (min(sum_col)))
        print ("Median ", index, " : % s" % (median(sum_col)))
        print ("Standard Deviation ", index, " : % s" % (np.std(sum_col))
```

Snippet of code 6.2 Statistics of car-price dataset

# Number of...	A Market Ca...	A Vehicle Size	A Vehicle St...	# highway M...	# city mpg	# Popularity	# MSRP
2	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916	46135
2	Luxury,Performance	Compact	Convertible	28	19	3916	40650
2	Luxury,High-Performance	Compact	Coupe	28	20	3916	36350
2	Luxury,Performance	Compact	Coupe	28	18	3916	29450
2	Luxury	Compact	Convertible	28	18	3916	34500
2	Luxury,Performance	Compact	Coupe	28	18	3916	31200
2	Luxury,Performance	Compact	Convertible	26	17	3916	44100
2	Luxury,High-Performance	Compact	Coupe	28	20	3916	39300

Illustration 6.2 Last 8 columns of car-price dataset

5.1.2 Data cleaning

Data are often extracted from multiple data sources. As a result, they are many times duplicated, mislabeled, incorrectly formatted, and incomplete, which brings us to the third step. This process is called data cleaning and is a rather significant part of the whole routine of machine learning. Data cleaning must be implemented before we insert the data in our algorithm should we want to get the most efficient result. In our case, I removed the column ‘Market Category’ from the dataset. It included data like ‘Factory Tuner, Luxury, High-Performance, Hatchback’, to name a few that were unimportant for the needed results. I also removed the duplicate rows and the null values. In order to do this data manipulation, I basically

used some “ready to go” functions that instantly remove the desired data. Almost every data manipulation that I do in the following datasets is given by the pandas library.

```
df = df.drop(['Market Category'], axis=1) // drop Market Category column
duplicate_rows_df = df[df.duplicated()] // find the duplicated rows
print("number of duplicate rows:", duplicate_rows_df.shape)
df = df.drop_duplicates() // remove duplicated rows
df = df.dropna() // remove the null values
```

Snippet of code 6.3 Data cleaning of the car-price dataset

5.1.3 Mapping

Almost half of the columns of this dataset were of categorical type. Algorithms cannot learn based on categorical variables but only based on numbers. So, my next step was to create a numeric map between every single string and a number. I then replaced every value of this dataset with the corresponding number to construct a dataset filled with numeric data.

```
df = df.replace({'Vehicle Size': {'Compact': 0, 'Midsize': 1, 'Large': 2 }}
```

Snippet of code 6.4 An example of a mapping between values and numbers of car-price dataset

5.1.4 Divide dataset

Now we have to split the dataset into a training and testing set. In my research, I tried to split the whole dataset, and the performance metrics of any algorithm I used were medium to low. This is why initially, I split the dataset into smaller datasets based on the statistics I found in the second step of my implementation. I relied upon column ‘Price’, which is the needed output, that has a mean of approximately 40594, and I split the dataset into four with the price ranges of:

- <25001
- <50001
- <100001

- >100000

```
# Prediction Model Building
X = df[['Year', 'HP', 'Cylinders', 'MPG-H', 'MPG-C', 'Doors', 'Make',
'Model', 'Transmission', 'Wheels', 'Engine Fuel Type', 'Vehicle Size',
'Popularity', 'Vehicle Style']]
y = df.Price;

# Create a better train-test split
df1 = df.query("Price < 25001");
percentage1 = (df1.shape[0] / df.shape[0]) * 100
X1 = df1[['Year', 'HP', 'Cylinders', 'MPG-H', 'MPG-C', 'Doors', 'Make',
'Transmission', 'Wheels', 'Model', ‘Vehicle Size', 'Engine Fuel Type',
'Popularity', 'Vehicle Style' ]]
y1 = df1.Price;

df2 = df.query("Price > 25000 & Price < 50001");
percentage2 = (df2.shape[0] / df.shape[0]) * 100
X2 = df2[['Year', 'HP', 'Cylinders', 'MPG-H', 'MPG-C', 'Doors', 'Make',
'Transmission', 'Wheels', 'Model', 'Vehicle Size', 'Engine Fuel Type',
'Popularity', 'Vehicle Style' ]]
y2 = df2.Price

df3 = df.query("Price > 50000 & Price < 100001");
percentage3 = (df3.shape[0] / df.shape[0]) * 100
X3 = df3[['Year', 'HP', 'Cylinders', 'MPG-H', 'MPG-C', 'Doors', 'Make',
'Transmission', 'Wheels', 'Model', 'Vehicle Size', 'Engine Fuel Type',
'Popularity', 'Vehicle Style' ]]
y3 = df3.Price;

df4 = df.query("Price > 100000");
percentage4 = (df4.shape[0] / df.shape[0]) * 100
X4 = df4[['Year', 'HP', 'Cylinders', 'MPG-H', 'MPG-C', 'Doors', 'Make',
'Transmission', 'Wheels', 'Model', 'Vehicle Size', 'Engine Fuel Type',
'Popularity', 'Vehicle Style' ]]
y4 = df4.Price
```

Snippet of code 6.5 Split car-price dataset

5.1.5 Training and testing

Now that the dataset or datasets are clean, readable, and generally comprehensive for the model to build a conclusion upon it, it is time to split the dataset into training and testing sets. Sci-kit learn provides us with a function that can randomly split a dataset however we ask for it. In my code, I want to split each one of

my new datasets into four new datasets, one training set and one testing set with the columns I give as input and one training set and one testing set with the column 'Price' that I want as an output. I also ask the 'train_test_split' function to randomly split them with a size of 80% for the training and 20% for the testing. In the previous step, I divided the initial dataset because of the randomness of this function. For example, there is a chance that a great number of cars is given to the testing set with a price larger than 1 million. In that case, the training set that the model is built upon and learning upon, is not actually training with cars that have a price over a million, or at least their number is small. Threfore, the results will be poor without any mistakes being made. So, after following this process, I know exactly how my dataset is split. Then, I rejoin all these smaller datasets into one in order to be ready for our model.

```
from sklearn.model_selection import train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size =
0.2, random_state = 8)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size =
0.2, random_state = 8)
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size =
0.2, random_state = 8)
X4_train, X4_test, y4_train, y4_test = train_test_split(X4, y4, test_size =
0.2, random_state = 8)
X_train = pd.concat([X1_train, X2_train, X3_train, X4_train], axis = 0)
X_test = pd.concat([X1_test, X2_test, X3_test, X4_test], axis = 0)
y_train = pd.concat([y1_train, y2_train, y3_train, y4_train], axis = 0)
y_test = pd.concat([y1_test, y2_test, y3_test, y4_test], axis = 0)
```

Snippet of code 6.6 Split into train and test

5.1.6 Algorithm

After many tries of multiple supervised learning algorithms, I ended up using Random Forest Regressor. This particular algorithm can take as parameters the estimators and the randomness. The estimators establish the maximum number of trees the algorithm can create. The more trees, the slower the code, however. Randomness refers to the random observations the algorithm can make. The following code presents how to import the algorithmic library, train the model, and apply the testing set. I also created a graph of the predictions and the true values using the matplotlib library.

```
from sklearn.ensemble import RandomForestRegressor
# create regressor object
```

```
reg = RandomForestRegressor(n_estimators = 200, random_state = 0)

# fitting the training data
model = reg.fit(X_train,y_train) // building the model
# make predictions
y_pred = reg.predict(X_test) // making a prediction based on test set
print(y_pred)

## The line / model - Create the graph using matplotlib
plt.scatter(y_test, y_pred)
plt.title("Cars Data Set")
plt.xlabel("True Values")
plt.ylabel("Predictions")
plt.show()
```

Snippet of code 6.7 Fit algorithm of car-price algorithm

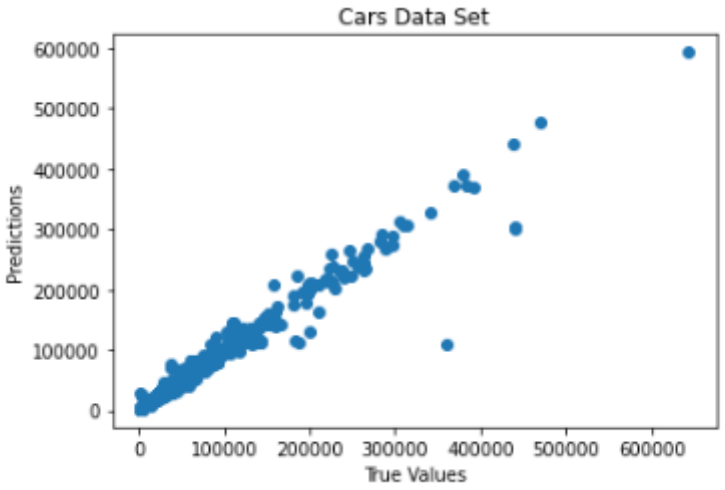

Illustration 6.3 Car-price graph

5.1.7 Results

The algorithm has a score of 0.9646938690195846, which means that it can predict the price of a car with a nearly perfect match. Using a regression analysis makes it almost impossible to have a score of one because regressor algorithms predict an actual numeric value, contrary to classification that predicts the category. In this algorithm, the 'MAE', which is the mean absolute error, has a score of 3,392.154, which may seem pretty big, but based on our inputs, it is an exceptional score. Criticizing the data, the prices vary between 2,000 and 2,065,902. So, the algorithm can predict a price with an error of 3,392.154, which based on our values, is not far from the truth.

```
## Score
print ("Score : ", model.score(X_test, y_test))
mae = mean_absolute_error(y_test, y_pred)
print('MAE: %.3f' % mae)
```

Snippet of code 6.8 Results of car-price algorithm

5.2 Second dataset (Car - Condition)

The second dataset also contains car characteristics, but the needed output here is the condition of the car, meaning good, very good, accounted, to name a few. This is a small dataset of 1,728 rows and 7 columns, including the output column. It was the easiest dataset because it did not need any data cleaning. It had no duplicates or null values. The only problem with this dataset was that every type of input is a string. In the beginning, I followed the same steps as in the first dataset except for when I imported the dataset. For some reason, the data did not have any headers, so I gave headers to the dataset when I imported it with the help of pandas. When I reached the point of data types, I tried to find an easier way to manipulate the data than the mapping.

```
headers = ["buying", "maint", "doors", "persons", "lug_boot", "safety",
"values"]
df = pd.read_csv (r'/content/car-data.csv', names=headers)
```

Snippet of code 6.9 Import the car-condition dataset

	buying	maint	doors	persons	lug_boot	safety	values
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
...	...	...	...	...	...	...	...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

Illustration 6.4 Car-condition data

5.2.1 Manipulate data types

In the first dataset, I created a map between the values of type string and the numeric values I gave. It was an exhausting and dull procedure. Thankfully, sci-kit learn provides programmers with a library that does exactly that. It takes a column that is consisted of non-numeric values and returns a vector representation of them, where all the elements of a vector are zeros, except for one. For example, if the column consisted of three different values, such as apple, banana, and strawberry, the return values for the banana would be [‘0’, ‘1’, ‘0’], which translates to ‘apple: false’, ‘banana: true’, ‘strawberry: false’. But when a column has many more values it creates greater values with only zeros and ones. This function is called One Hot Encoder, and it can relieve you of many hours of dull work, especially in large datasets. The following code is from the manipulation of the first column of this dataset, but the exact same code was used on repeat for the other columns, too.

```
from sklearn.preprocessing import OneHotEncoder
buy = df['buying'].values.reshape(-1, 1)
df['buying'] = OneHotEncoder(sparse=False).fit_transform(buy)
```

Snippet of code 6.10 One Hot Encoder

5.2.2 Results

After the One Hot Encoder, I used the same steps as in the first dataset. This time I split the whole dataset into training and testing sets and used Decision Trees Regressor to build my model.

```
# Split DataSet in test and train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X.values, Y, test_size =
0.2, random_state = 8)

from sklearn.tree import DecisionTreeRegressor
reg = DecisionTreeRegressor(random_state = 0)

# fitting the training data
model = reg.fit(X_train,y_train)
# make predictions
```

```
y_pred = reg.predict(X_test)

## Score
print("Score : ", model.score(X_test, y_test))
mae = mean_absolute_error(y_test, y_pred)
print('MAE: %.3f' % mae)
```

Snippet of code 6.11 Results of car-condition dataset

The model's score is 0.8762235483456776, lower than the first dataset but generally good enough. This time the mean absolute error has a score of 0.038. This happens because our output value consists of zeros and ones, like the rest of our columns,and the mean absolute error is based on this.

5.3 Third dataset (Car - Fuel Consumption)

# distance	# consume	# speed	# temp_inside	# temp_outs...	▲ specials
28	5	26	21,5	12	
12	4,2	30	21,5	13	
11,2	5,5	38	21,5	15	
12,9	3,9	36	21,5	14	
18,5	4,5	46	21,5	15	
8,3	6,4	50	21,5	10	
7,8	4,4	43	21,5	11	
12,3	5	40	21,5	6	
4,9	6,4	26	21,5	4	
11,9	5,3	30	21,5	9	
12,4	5,6	42	21,5	4	
11,8	4,6	38	21,5	0	
12,3	5,9	59	21,5	10	
24,7	5,1	58	21,5	12	
12,4	4,7	46	21,5	11	
17,3	5,1	24	21,5	5	
33,4	5,6	36	21,5	3	
11,8	5,1	32	21,5	3	
25,9	4,9	39	21,5	8	

Illustration 6.5 First 6 columns of car-fuel consumption dataset

The third dataset is about a person who drives the same car every day and wants to predict the consumption of fuel based on many factors, such as the fuel type and the weather. It is a pretty small dataset, consisting of 338 rows and 12 columns. Being more mature with this dataset, because of my friction with the previous two datasets and tons of research and reading, I tried to use more functions from our libraries to make things easier. Specifically, I used functions to identify the statistics of the numeric functions and all the needed information for the data cleaning and manipulation.

```
print(df.describe().T)

// Printed part
              count       mean        std   min    25%   50%   75%   max
speed         388.0  41.927835  13.598524  14.0  32.75  40.5  50.0  90.0
temp_outside  388.0  11.358247   6.991542  -5.0   7.00  10.0  16.0  31.0
AC            388.0   0.077320   0.267443   0.0   0.00   0.0   0.0   1.0
rain          388.0   0.123711   0.329677   0.0   0.00   0.0   0.0   1.0
sun           388.0   0.082474   0.275441   0.0   0.00   0.0   0.0   1.0

print(df.info())

// Printed part
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 388 entries, 0 to 387
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   distance       388 non-null    float32
 1   consume        388 non-null    float32
 2   speed          388 non-null    int64
 3   temp_inside    376 non-null    float32
 4   temp_outside   388 non-null    int64
 5   specials       93 non-null     object
 6   gas_type       388 non-null    object
 7   AC             388 non-null    int64
 8   rain           388 non-null    int64
 9   sun            388 non-null    int64
 10  refill liters  13 non-null     float32
 11  refill gas     13 non-null     object
dtypes: float32(4), int64(5), object(3)
memory usage: 30.4+ KB
None

print(df.isna().sum())
```

Snippet of code 6.12 Information on car-fuel consumption dataset

gas_type	AC	rain	sun	refill liters	refill gas
E10	0	0	0	45	E10
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		
E10	0	0	0		

Illustration 6.6 Last 6 columns of car-fuel consumption dataset

5.3.1 Null input

This particular dataset, like most datasets in the real world, had a great number of null inputs, always regarding its size. When there are null values in the dataset, the ‘split’ function does not work, so I had to fill the null values in a way that the dataset is not ruined. There is a great chance of ruining a dataset and its purpose if you replace null values with incorrect data. For example, supposing someone has a dataset about COVID-19 information in many countries, and the needed output is to predict the number of victims in a single day in a specific country. It is quite common occasionally not to have the number of victims of all the countries. In that case, if we replace the missing values with zero, then the dataset is ruined. So, firstly, I used Multivariate imputer, a function provided by sci-kit learn, that completes the missing values using values from the statistics, such as mean or median. In other cases, it might have ruined the dataset, but in this case, the missing values were only 12, so it did not have a high impact. For the other columns, I filled the null values with an extra option that fitted the data perfectly. For example, in the column ‘refill liters’, the

dataset had values only when the driver refilled the car, so for the other rows, I replaced the null value with zero. The same thought is implemented for the other columns, too.

```
#temp_inside: apply imputer because low number of nulls
impute_it = IterativeImputer()
df['temp_inside'] =
impute_it.fit_transform(df['temp_inside'].values.reshape(-1,1)).reshape(-1)

#specials: replace nan to other
df['specials'] = df['specials'].fillna('others')

#refill liters replace nan to 0 because not refill
df['refill liters'] = df['refill liters'].fillna(0)

#refill gas replace nan to notrefill because not refill
df['refill gas'] = df['refill gas'].fillna('norefill')
```

Snippet of code 6.13 Fill null values in car-fuel consumption dataset

5.3.2 Manipulate dataset

In addition to the previous step, I also changed the form of the dataset. I chose three columns that were partially empty, and with the help of the pandas library, I created more columns based on their values. For example, column ‘specials’ consisted of values such as ‘sun’, ‘rain’, and so forth. So, I added to the dataset a column with the name ‘specials_sun’ and values of zero when there was no sun and one when there was, and so on. After adding the new columns, I removed the old columns, because now they had no value. The next two illustrations show the new form of the car-fuel consumption dataset (Illustrations 6.7 and 6.8).

```
features = ["specials", "gas_type", "refill gas"] // mark the almost empty
columns
df_dummies = pd.get_dummies(df[features]) // create new columns based on the
values of ‘features’
df = pd.concat([df,df_dummies], axis = 1) // add the new columns to the
dataset
df = df.drop(features + ["gas_type_SP98"], axis = 1) // remove old columns
```

Snippet of code 6.14 Manipulate the car-fuel consumption dataset

	distance	consume	speed	temp_inside	temp_outside	AC	rain	sun	refill liters	specials_ac	...	specials_half rain half sun
0	28.0	5.0	26	21.5	12	0	0	0	45.0	0	...	0
1	12.0	4.2	30	21.5	13	0	0	0	0.0	0	...	0
2	11.2	5.5	38	21.5	15	0	0	0	0.0	0	...	0
3	12.9	3.9	36	21.5	14	0	0	0	0.0	0	...	0
4	18.5	4.5	46	21.5	15	0	0	0	0.0	0	...	0

5 rows × 23 columns

Illustration 6.7 First columns of the new form of car-fuel consumption dataset

specials_others	specials_rain	specials_snow	specials_sun	specials_sun ac	gas_type_E10	refill gas_E10	refill gas_SP98	refill gas_norefill
1	0	0	0	0	1	1	0	0
1	0	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0	1

Illustration 6.8 Last columns of the new form of car-fuel consumption dataset

5.3.3 Train grid search

Then, I split the dataset into training and testing sets, exactly like before. So, the only thing remaining is to find the algorithm that I am going to use. In order not to try random algorithms and find the best that suits this situation, I used a function from sci-kit learn that takes as input the possible algorithm you are going to use and some possible parameters and as a result, it returns the best combination for your model. This function is called ‘Grid Search’. Though, I did not stop there. I moved all the possible algorithms that I was going to try and all their possible parameters to a variable. Then, I wrote my own function that takes as input the variables that I created and it does the following :

- keeps the time that every model took to build
- uses the grid search function for every algorithm I passed as an input, with the parameters I also gave as an input
- trains my model
- makes a prediction based on the best estimator that the grid search returned as a result
- prints some performance metrics after every prediction

```
models = {"Linear Regression":LinearRegression(), "SGD
Regressor":SGDRegressor(), "Elastic Net":ElasticNet(), "Ada Boost
Regressor":AdaBoostRegressor(), "Random Forest
Regressor":RandomForestRegressor(), "Decision Tree
Regressor":DecisionTreeRegressor()}
param_grids = [
    {'n_jobs': [2, 5, 7, 10], 'fit_intercept': [True, False]},
    {'alpha': [0.0001, 0.0002, 0.0005, 0.001], 'loss': ( "squared_error",
"huber")},
    {'alpha': [1.0, 0.75, 0.5, 0.25]},
    {'n_estimators':[50,100,200,500,1000], 'learning_rate':[1.0, 2.0, 5.0,
10.0],'loss': ('linear', 'square')},
    {'n_estimators':[50,100,200,500], 'criterion':('squared_error',
'absolute_error', 'poisson'), 'min_samples_split':[2,5,10],
'min_samples_leaf':[1,2,3]},
    {'criterion':('squared_error', 'absolute_error',
'poisson'),'min_samples_split':[2,5,10], 'min_samples_leaf':[1,2,3]}
]

from time import time
def train_grid_search(models, param_grids):
    best_params = []
    best_models = []
    best_mae = []
    best_mse = []
    best_r2 = []

    for idx, model_name in enumerate(models.keys()):
        t1 = time()
        print("Model Name: ", model_name)
        print("Model: ", models[model_name])
        cv = GridSearchCV(models[model_name], param_grids[idx], cv=5)
        cv.fit(X_train, y_train.values)

        model = cv.best_estimator_
        y_pred = model.predict(X_test)
        mae = mean_absolute_error(y_test.values, y_pred)
        mse = mean_squared_error(y_test.values, y_pred)
        r2 = r2_score(y_test.values, y_pred)

        best_params.append(cv.best_params_)
        best_models.append(model)
        best_mae.append(mae)
        best_mse.append(mse)
        best_r2.append(r2)
        print("time:", time()-t1)
```

```
        print("---------------------------------")

    return {"model_names":models.keys(), "best_params":best_params,
"best_models":best_models, "best_mae":best_mae, "best_mse":best_mse,
"best_r2":best_r2}
results = train_grid_search(models, param_grids)
```

Snippet of code 6.15 Grid Search

5.3.4 Results

The grid search tool is really advantageous and works perfectly, but you still need to write all the parameters and algorithms by hand. The snippet above (Snippet of code 6.15) provides exactly the automated search that someone might need. The result of this snippet was that the best algorithm for this dataset is the ‘Random Forest Regressor’ with a score of 0.6281995772126836, which is not that good of a score but is at least above average. In conclusion, this result proves that the type of fuel, temperature, or weather does not really affect the consumption of car fuel.

6. Conclusion

In this dissertation, the term machine learning is defined along with its importance, and after the detailed explanation of my implementation, the process needed to build one model.

The aim of this dissertation was the exploration of different tools that are used in machine learning so as to facilitate coding and make the results more efficient, especially with the use of some libraries that Python offers. It is clear now why Python is so popular in the machine learning community and how helpful and well-documented its libraries are.

It is worth noting that the different approaches while seeking to solve the same problem do not follow the same methodology. Some approaches focus on the structural characteristics of its code, others on its semantic features, others on code metrics, and others in a combination of the above. Additionally, it was proved that a great number of data are not in the perfect condition that is generally expected. Data cleaning and data manipulation are integral parts of the machine learning process, and a bad implementation of these could lead to fallacious conclusions.

The evaluation of the models is a must. All the performance metrics serve different purposes and can be applied to different problems. Every engineer analyzes the data and the problem that must be resolved and creates a plan. This plan consists of the type of learning, the types of algorithms, the training and testing percentages, and, of course, the performance metrics.

To sum up, there are learning algorithms in many applications that we make daily use of. Every time a web search engine like Google is used to search the internet, one of the reasons that it works so well is because of a learning algorithm that has learned how to rank web pages. These algorithms are used for various purposes like data mining, image processing, and predictive analytics, to name a few. Even the datasets that are used in this dissertation can easily be combined to predict the price of a used car so that its owner can sell it at a reasonable price for the market. The main advantage of using machine learning is that once an algorithm learns what to do with data, it can do its work automatically.

References

1. Barto, A. G. & Sutton, R. (1997). Introduction to Reinforcement Learning. MIT Press.
2. Batta Mahesh (2018). Machine Learning Algorithms - A Review. International Journal of Science and Research (IJSR) ISSN: 2319-7064 ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426
3. Ethem Alpaydin (2020). Introduction to Machine Learning. Published by Massachusetts Institute of Technology
4. Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot and Edouard Duchesnay, 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12 (2011) 2825-2830
5. Jain, A.K., Murty, M. N., and Flynn, P. (1999). Data clustering: A review. ACM Computing Surveys, 31(3): 264–323.
6. Muhammad Usama, Junaud Qadir, Aunn Raza, Hunain Arif, Kok-Lim Alvin Yau, Yehia Elkhatib, Amir Hussain, Ala Al-Fuqaha (May 14, 2019). Unsupervised Machine Learning for Networking: Techniques, Applications, and Research Challenges. In Digital Object Identifier 10.1109/ACCESS.2019.2916648
7. Piatetsky, G. (2019). Python Leads the 11 Top Data Science, Machine Learning Platforms: Trends and Analysis. Available in: https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html
8. S. B. Kotsiantis (July 16, 2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica 31 249-268 249
9. Sandro Tosi (2009): Matplotlib for Python developers. Available in : https://d1wqtxts1xzle7.cloudfront.net/32168807/Matplotlib_for_Python_Developers_%282009%29.pdf20131027-19008-5fvdfp-libre-libre.pdf?1382866688=&response-content-disposition=inline%3B+filename%3DMatplotlib_for_Python_Developers_2009.pdf&Expires=1675024829&Signature=U2OnBRvR5knUFWkcrG9rdPQRYO-HYD9~~O9bKit2h5kJR1k~mz9PEqnfm3DDfxRBemNxhtLLf7b7o4QAQWM6dkG-tIhb7fT9QXQP2gjJ~luFjM1BQI9RcGsejj1CJyefdtMcLFcr31gIX-hMoATumrJl2PY2HmxlGhYkwTbeqz97jMMSoTh5sX6nOYkh-nXKhFEad-TLNmCGmSHJsewcFPOs5v2ItfrSYcoIdTAttGiwExj8FIbsRShJicxBQs0D3tm4YAh-1yZ~sEpFKT8ofugeLp5q1QRO0ghkOMHrXWjEIa02pr9gu92dIZoImGvXf6br1OCiEQfA0OEG76jLjg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

10. Sebastian Raschka, Joshua Patterson, Corey Nolet (2020). Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. In Information, Vol 11, Iss 193, p 193.
11. Sofia Visa, Brian Ramsay, Anca Ralescu, Esther van der Knaap (2011). Proceedings of the Twentysecond Midwest Artificial Intelligence and Cognitive Science Conference April 16 – 17, 2011 University of Cincinnati Cincinnati, Ohio. Available in : https://www.researchgate.net/profile/Jennifer-Seitzer-2/publication/220833258_Using_a_Genetic_Algorithm_to_Evolve_a_D_Search_Heuristic/links/545a2bea0cf2bccc49132577/Using-a-Genetic-Algorithm-to-Evolve-a-D-Search-Heuristic.pdf#page=126
12. Stéfan van der Walt, S. Chris Colbert, Gael Varoquaux (2011). The NumPy array: a structure for efficient numerical computation. Published in IEEE Computing in Science and Engineering.
13. Travis E. Oliphant (2006). Guide to NumPy. Available in: https://ecs.wgtn.ac.nz/foswiki/pub/Support/ManualPagesAndDocumentation/numpybook.pdf
14. Vladimir Nasteski (2017). An overview of the supervised machine learning methods. DOI 10.20544/HORIZONS.B.04.1.17.P05 UDC 004.85.021:519.718
15. W McKinney (2011). **Python** for high performance and scientific computing. Available in: https://www.dlr.de/sc/portaldata/15/resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf
16. Wei-Meng Lee (2019). Python Machine Learning. Published by: John Wiley & Sons, Crosspoint Boulevard, Indianapolis.
17. Σωτηρίου Γεώργιου (2022). ΤΕΧΝΙΚΕΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ (MACHINE LEARNING) ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΠΡΟΒΛΗΜΑΤΩΝ ΣΧΕΔΙΑΣΗΣ Ή ΑΛΛΩΝ ΠΑΡΑΜΕΤΡΩΝ ΣΕ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ. Bachelor Thesis, University of Macedonia
18. pandas documentation 1.5.3, Retrieved January 2023, from https://pandas.pydata.org/docs/
19. numpy documentation 1.24, Retrieved January 2023, from https://numpy.org/doc/stable/
20. matplotlib documentation 3.6.3, Retrieved January 2023, from https://matplotlib.org/stable/index.html
21. scikit-learn documentation 0.21.3, Retrieved January 2023, from https://scikit-learn.org/0.21/documentation.html