PYTHON

MACHINE LEARNING

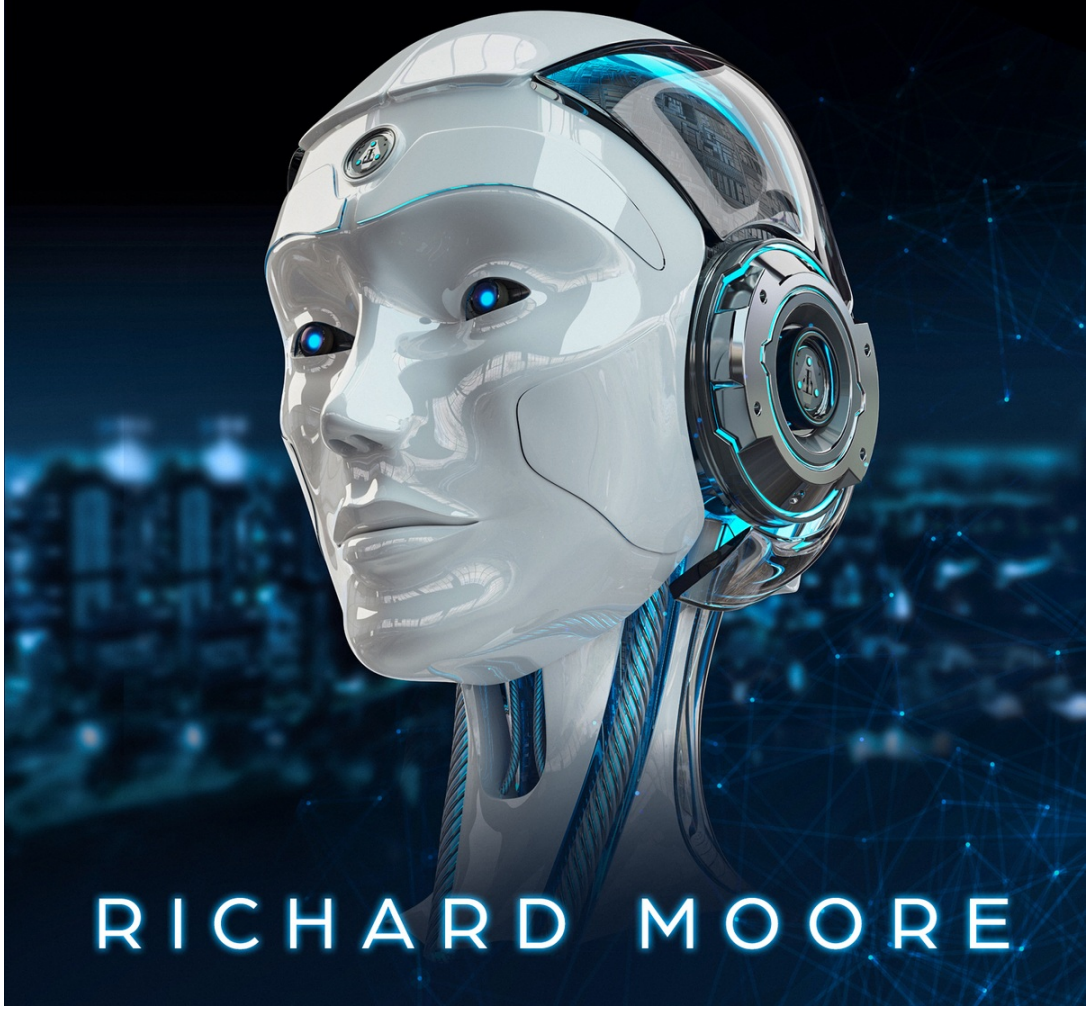
THE ULTIMATE GUIDE FOR BEGINNERS TO MACHINE LEARNING WITH PYTHON, PROGRAMMING AND DEEP LEARNING, ARTIFICIAL INTELLIGENCE, NEURAL NETWORKS, AND DATA SCIENCE

RICHARD MOORE

PYTHON

MACHINE LEARNING

THE ULTIMATE GUIDE FOR BEGINNERS TO MACHINE LEARNING WITH PYTHON, PROGRAMMING AND DEEP LEARNING, ARTIFICIAL INTELLIGENCE, NEURAL NETWORKS, AND DATA SCIENCE



RICHARD MOORE

Python Machine Learning

The Ultimate Guide for Beginners to Machine Learning with Python, Programming and Deep Learning, Artificial Intelligence, Neural Networks, and Data Science.

By Richard Moore

Table of Contents

Introduction

Machine learning is a fairly new domain, however it quickly became the latest trend on which every IT specialist is trying to jump, and rightly so. This technical field can be used to solve a wide variety of today's problems. The purpose of *Python Machine Learning* is to guide you step by step through all the basic concepts of machine learning.

Throughout the course of this book you will learn about all the specialised tools you will need on your journey. In addition, you will learn a variety of machine learning algorithms and techniques by combining theory with practice. You will be applying everything you learn on real datasets and learn the practical use of every concept you study.

Keep in mind that in order to benefit the most from using this ultimate guide for beginners, you will have to go through every practical application and dissect it until you understand it. In addition, you will have to start practicing on your own in order to build a proper foundation. You don't have to be a computer science major to understand machine learning, however you do need a great deal of determination and most importantly, practice!

Chapter 1: Why Machine Learning?

So you're interested in IT, programming, or all that is tech and recently you keep hearing about machine learning. Wherever you turn your head to explore something new, this concept keeps coming up. At this point you probably expect some shady guy wearing a trench coat to approach you in a dark alley and ask you, "Hey kid, interested in some machine learning?" You probably thought that there must be something worth studying here if everyone who knows how to write a few lines in Python keeps talking about it. So what is machine learning and why is everyone so crazy about it?

In this chapter we will address these questions and more. It's important to understand why machine learning is such a hot topic and why many see it as the future in all that is technology. You already took the first step in exploring this field, but if you get a better idea about the application of machine learning, you will definitely become more motivated to study it and learn it.

The Importance of Machine Learning

Machine learning started out as a component of Artificial Intelligence, and therefore its definition is somewhat tied to it. In essence, machine learning software learns from experience, almost like humans. Keep in mind that in this case experience refers to data and the ability to learn does not involve direct programming. When a machine learning application is exposed to data it will absorb it, learn from it, adapt itself based on it and evolve. In other words, systems that are powered by machine learning doesn't need to be instructed by someone how to search for something and where to find it. Instead, it relies on its own algorithms to learn from the data and then apply what it learned in order to perform better and repeat the cycle. See? It's almost human, but no this is not from the realm of science fiction. Machine learning has been here for quite some time and it's becoming more and more of an industry standard in research as well as product design.

In fact, the idea of machine learning is at least as old as the Enigma Machine, however its ability to perform a massive number of fully automated mathematical operations became possible only in the past couple of decades due to the massive leaps in computer processing power. Furthermore, a number of

new technologies were developed in recent years and they are the ones that take advantage of machine learning the most.

To get a better idea why machine learning is so important, you need to know where it's actually applied. Are you aware of Google's fully automated, self-driving car? You have machine learning to thank for that. Do you use Facebook, Amazon, or Netflix? Check, check and check, they all rely on machine learning to power their ability to recommend you things you may be interested in. That's how they know you love funny cat videos. Computer systems can learn all of this by running all the data you generate through a filter, keep only what is relevant to you and what fits your pattern, and then give you the accurate results you don't even know you're looking for.

Machine learning seems to have evolved so suddenly, even though it had been theorized by the great mathematicians who lived centuries ago (yes, even before the invention of the computer), only because of market demand and its real world application for mass consumption. For instance, you may have also heard about machine learning in the same sentence as Big Data. That's because it is highly sophisticated nowadays and it allows us to process and analyse massive amounts of data that no human could handle otherwise. Furthermore, machine learning has led to a revolution in the way we extract and analyze data because it allows us to automate the algorithms and therefore replace the slow, cumbersome statistical methods that used to be the norm.

Applied Machine Learning

Machine learning has been lately integrated in pretty much everything that is tech and you will often not even notice it or think of it. Earlier we mentioned the more obvious sectors where machine learning algorithms take over. However, you will also encounter them in anything that has to do with web search results, real time online ads, email spam protection, network protection, image recognition software, and much more. All of these products or services have one thing in common. They all take advantage of machine learning because of the massive amounts of data that need to be processed and analyzed.

Data analysis used to be quite tricky before the implementation of machine learning algorithms. It involved a trial and error approach, and it might've worked two decades ago. However, today data is produced at such a staggering speed that it is impossible to keep up with its exponential growth. The more the computer processing power advances, the more tech we devise, and the more

data and information we generate. For instance, even your smart home devices generate huge quantities of data that is collected for analysis in order to improve those devices further. Machine learning is the only alternative to the traditional techniques. It provides us with fast and powerful algorithms that produce accurate results. Machine learning will not go anywhere anytime soon, quite the contrary.

Popular Machine Learning Methods

Now that you understand why machine learning is part of your life, you need to understand how it is capable of learning.

Generally, there are two major machine learning techniques, namely supervised learning and unsupervised. Most of machine learning, around seventy percent, is in fact supervised learning. Another ten to twenty percent is unsupervised learning. There are other techniques as well, such as semi-supervised learning and reinforcement learning, but they aren't used as often. Here's a brief introduction to what these techniques achieve:

1. Supervised learning: This machine learning method can be implemented only when we can identify the inputs and outputs and we have labeled data to work with. The algorithm will obtain its inputs together with the matching output in order to detect the errors. Based on the inputs, the model will be adapted. This entire process is in fact pattern recognition and these algorithms always rely on techniques such as regression, classification, prediction, or gradient boosting. Keep in mind that you will also have to deal with unlabeled data, and this is where pattern recognition fills in these blanks. Once a pattern is established, the algorithm can predict a label's values. Supervised machine learning is normally used in applications that have to predict the future. One such example is finding credit card transaction frauds.
2. Unsupervised learning: Supervised learning involves historical data in order to predict future events, however, that's not the case with unsupervised learning. This type of algorithm analyses the collected data in order to determine its structure. This is why unsupervised learning is used to understand customer categories without any attributes that stand out, or to give you online recommendations. Some of the most popular unsupervised machine learning techniques

include nearest-neighbour and k-means clustering.

Machine learning is here to stay and that is why everyone is talking about it. It is one of the biggest trends in tech today because it serves a day to day purpose. It is used in so many areas that without a doubt include your daily life as well. This makes machine learning one of the most lucrative tech sectors with a wide array of career opportunities at your disposal. So if you want to get a piece of the action, you have all the resources you need.

Chapter 2: The Basics of Working with Python

Before we start working with machine algorithms, you should first understand the basics of working with Python. However, if you are already familiar with Python or you have experience programming in other languages such as C++ or C# you can probably skip this chapter or simply use it to refresh your memory.

In this chapter we are going to briefly discuss the basic concepts of working with Python. Machine learning and Python go hand in hand due to the simple fact that Python is a simple, but powerful and versatile language. Furthermore, there are many modules, packages and tools designed to expand Python's functionality to specifically work with machine learning algorithms, as well as data science.

Keep in mind that this is a brief introduction to Python, and therefore we will not be using any IDE's or fancy tools. All you need is the Python shell, in order to test and experiment with your code as you learn. You don't even need to install anything on your computer because you can simply head to Python's official website and use their online shell. You can find it here: *https://www.python.org/shell/*.

Data Types

Knowing the basic data types and how they work is a must. Python has several data types and in this section we will go through a brief description of each one and then see them in practice. Don't forget to also practice on your own, especially if you know nothing or very little about Python.

With that in mind, let's explore strings, numbers, dictionaries, lists and more!

Numbers

In Python, just like in math in general, you have several categories of numbers to work with and when you work them into code, you have to specify which one you're referring to. For instance, there are integers, floats, longs and others. However, the most commonly used ones are integers and floats.

Integers, written int for short, are whole numbers that can either be positive or negative. So make sure that when you declare a number as an integer you don't

type a float instead. Floats are decimal or fractional numbers.

Now let's discuss the mathematical operators. Just like in elementary school, you will often work using basic mathematical operators such as adding, subtracting, multiplication and so on. Keep in mind that these are different from the comparison operators, such as greater than or less than or equal to. Now let's see some examples in code:

```
x = 99

y = 26

print (x + y)
```

This basic operation simply prints the sum of x and y. You can use this syntax for all the other mathematical operators, no matter how complex your calculation is. Now let's type a command using a comparison operator instead:

```
x = 99

y = 26

print (x > 100)
```

As you can see, the syntax is the same, however, we aren't performing a calculation. Instead we are verifying whether the value of x is greater than 100. The result you will get is "false" because 99 is not greater than 100.

Next, you will learn what strings are and how you can work with them.

Strings

Strings have everything to do with text, whether it's a letter, number or punctuation mark. However, take note that numbers written as strings are not the same as the numbers data type. Anything can be defined as a string, but to do so you need to place quotation marks before and after your declaration. Let's take a look at the syntax:

```
n = "20"

x = 10
```

Notice that our n variable is a string data type and not a number, while x is defined as an integer because it lacks the quotation marks. There are many operations you can do on strings. For instance you can verify how long a string is, or you can concatenate several strings. Let's see how many characters there

are in the word “hello” by using the following function:

len (“Hello”)

The “len” function is used to determine the number of characters, which in this case is five. Here’s an example of string concatenation. You’ll notice that it looks similar to a mathematical operation, but with text:

‘42 ’ + ‘is ’ + ‘the ’ + ‘answer’

The result will be “42 is the answer”. Pay attention to the syntax, because you will notice we left a space after each string, minus the last one. Spaces are taken into consideration when writing strings. If we didn’t add them, all of our strings would be concatenated into one word.

Another popular operation is the string iteration. Here’s an example:

bookTittle = “Lord of the Rings”

for x in book: print c

The result will be an iteration of every single character found in the string. Python contains many more string operations, however these are the ones you will use most often.

Now let’s progress to lists.

Lists

This is a data type that you will be using often. Lists are needed to store data and they can be manipulated as needed. Furthermore, you can store objects of different types in them. Here’s what a Python list looks like:

n = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

The list is defined by the square brackets and every object separated by a comma is a list element. Here's an example of a list containing different data types:

myBook = [“title”, “somePages”, 1, 2.1, 5, 22, 42]

This is a list that holds string objects as well as integers and floats. You can also perform a number of operations on lists and most of them follow the same syntax as for the strings. Try them out!

Dictionaries

This data type is nearly identical to a list, however, you cannot access the elements the same way. What you need is to know the key which is linked to a dictionary object. Take a look at the following example:

dict = {‘weapon’ : ‘sword’, ‘soldier’ : ‘archer’}

dict [‘weapon’]

The first line contains the dictionary's definition, and as you can see the objects and their keys have to be stored between curly braces. You can identify the keys as “weapon” and “soldier” because after them you need to place a colon, followed by the attribute. Keep in mind that while in this example our keys are in fact strings, they can be other data types as well.

Tuples

This data type is similar to a list, except its elements cannot be changed once defined. Here’s an example of a tuple:

n = (1, 43, ‘someText’, 99, [1, 2, 3])

A tuple is defined between parentheses and in this case, we have three different data types, namely a few integers, a string, and a list. You can perform a number of operations on a tuple, and most of them are the same as for the lists and strings. They are similar data types, except that once you declare the tuple, you cannot change it later.

Conditional Statements

Now that you know the basic data types, it’s time to take a crash course on more complex operations that involve conditional statements. A conditional statement is used to give an application a limited ability to think for itself and make a decision based on their assessment of the situation. In other words, it analyzes the condition required by a variable in order to tell the program to react based on the outcome of that analysis.

Python statements are simple to understand because they are logical and the syntax reflect human thinking. For instance, the syntax written in English looks something like this “If I don’t feel well, I won’t go anywhere, else I will have to go to work”. In this example, we instruct the program to check whether you feel

well. If the statement is valued as false, it means you feel well and therefore it will progress to the next line which is an “else” statement. Both “if” and “if else” conditionals are frequently used when programming in general. Here’s an example of the syntax:

```
x = 100

if (x < 100):

    print(“x is small”)
```

This is the most basic form of the statement. It checks whether it’s true, and if it is then something will happen and if it’s not, then nothing will happen. Here’s an example using the else statement as well:

```
x = 100

if (x < 100):

    print(“x is small”)

else:

    print(“x is large”)

print (“Print this no matter what”)
```

With the added “else” keyword, we instruct the application to perform a different task if a false value is returned. Furthermore, we have a separate declaration that lies outside of the conditional statement. This will be executed no matter the outcome.

Another type of conditional involves the use of “elif” which allows the application to analyse a number of statements before it makes a decision. Here’s an example:

```
if (condition1):

    add a statement here

elif (condition2):

    add another statement for this condition

elif (condition3):

    add another statement for this condition

else:
```

if none of the conditions apply, do this

Take note that this time we did not use code. You already know enough about Python syntax and conditionals to turn all of this into code. What we have here is pseudo code, which is very handy whether you are writing simple Python exercises or working with machine learning algorithms. Pseudo code allows you to place your thoughts on “paper” by following the Python programming structure. This makes it a lot easier for you to organize your ideas and your application, by writing the code after you’ve outlined it. With that being said, here’s the actual code:

```
x = 10
if (x > 10):
    print (“x is larger than ten”)
elif x < 4:
    print (“x is smaller”)
else:
    print (“x is pretty small”)
```

Now you have everything you need to know about conditionals. Use them in combination with what you learned about data types in order to practice. Keep in mind that you always need to practice these basic Python concepts in order to later understand how machine learning algorithms work.

Loops

Code sometimes needs to be executed repeatedly until a specific condition is met. This is what loops are for. There are two types, the for loop and the while loop. Let’s begin with the first example:

```
for x  in range(1, 10):
print(x)
```

This code will be executed several times, printing the value of X each time, until it reaches ten.

The while loop, on the other hand, is used to repeat the execution of a code block only if the condition we set is still true. Therefore, when the condition is no

longer met, the loop will break and the application will continue with the next lines of code. Here's a while loop in action:

```
x = 1
while x < 10:
    print(x)
    x += 1
```

The x variable is declared as an integer and then we instruct the program that as long as x is less than ten, the result should be printed. Take note that if you do not continue with any other statement at this point you will create an infinite loop and that is not something you want. The final statement makes sure that the application will print the new value with one added to it with every execution. When the variable stops being less than ten, the condition will no longer be met and the loop will break, allowing the application to continue executing any code that follows.

Keep in mind that infinite loops can easily happen due to mistakes and oversight. Luckily, Python has a solution, namely the "break" statement which should be placed at the end of the loop. Here's an example:

```
while True:
    answer = input ("Type command:")
    if answer == "Yes":
    break
```

Now the loop can be broken by typing a command.

Functions

As a beginner machine learner this is the final Python component you need to understand before learning the cool stuff. Functions allow you to make your programs a great deal more efficient, optimized, and easier to work with. They can significantly reduce the amount of code you have to type, and therefore make the application less demanding when it comes to system resources. Here's an example of the most basic function to get an idea about the syntax:

```
def myFunction():
```

```
    print("Hello, I am now a function!")
```

Functions are first declared by using the "def" statement, followed by its name. Whenever we want to call this block of code, we simply call the function instead of writing the whole code again. For instance, you simply type:

```
myFunction()
```

The parentheses after the function represent the section where you can store a number of parameters. They can alter the definition of the function like this:

```
def myName(firstname):
    print(firstname + " Smith")

myName("Andrew")

myName("Peter")

myName("Sam")
```

Here we have a first name parameter and whenever we call the function to print its parameter, it does so together with the addition of the word "Smith". Take note that this is a really basic example just so you get a feel for the syntax. More complex function are written the same way, however.

Here's another example where we have a default parameter, which will be called only if there is nothing else to be executed in its place.

```
def myHobby(hobby = "leatherworking"):
    print ("My hobby is " + hobby)

myHobby ("archery")

myHobby ("gaming")

myHobby ()

myHobby ("fishing")
```

Now let's call the function:

```
My hobby is archery

My hobby is gaming

My hobby is leatherworking

My hobby is fishing
```

You can see here how the default parameter is used when we lack a specification.

Here you can see that the function without a parameter will use the default value we set.

In addition, you can also have functions that return something. For now we only wrote functions that perform an action, but they don’t return any values or results. These functions are far more useful, because the result can then be placed into a variable which will later be used in another operation. Here’s how the syntax looks in this case:

```
def square(x):
    return x * x
print(square (5))
```

Now that you’ve gone through a brief Python crash course and you understand the basics, it’s time to learn how to use the right tools and how to set up your machine learning environment. Don’t forget that Python is only one component of machine learning, however it’s an important one because it's the foundation and without it everything falls apart.

Chapter 3: Environment Setup

In this chapter you will learn all about setting up the appropriate environment for your machine learning needs. Keep in mind that while all you truly need is Python, working with machine learning algorithms and techniques without any other tool would be extremely difficult. That is why in this chapter you will build your machine learning toolkit which includes Python libraries, modules, packages, scientific distributions, and more.

You will learn how to create an optimal environment with the help of modules such as NumPy, Pandas, Jupyter, and how to use tools such as Scikit-learn and TensorFlow to greatly improve your productivity.

Python Distributions

Before you jump in head first, you should prepare your work environment in order to have an easier time pre-processing and analyzing your data. Working with algorithms, large datasets, and even just programming can be extremely time consuming without the right tools.

Installing Python and various libraries and modules will certainly give you the flexibility and power you need, however, there's an even more efficient way of getting started. The answer is Python scientific distributions. These distributions are Python installations packaged together with a number of tools that machine learners and data scientists need to work with data sets and specific algorithms.

One of the most popular scientific distributions is Anaconda and we will be discussing it briefly in the next section. However, keep in mind that it is only one of the many distributions which contains the modules and tools needed for machine learning. You can always explore as you study in order to find what suits your needs.

Anaconda

Anaconda is an open source Python scientific distribution which has risen in popularity due to the many useful packages and libraries it contains, along with the fact that it is a free application that anyone can use. There are close to 200 Python packages that come with Anaconda, including NumPy, Scikit-learn, and

Pandas, to name a few of the most valuable ones. Furthermore, Anaconda is also a package management application which makes the import and installation of any package or tool a breeze. You can download, install, or remove anything you need and you can also keep everything up to date with the click of a button. You can even setup a virtual environment if you prefer them over working directly on your system.

Anaconda can be downloaded and installed as the basic version or the premium one. Keep in mind that for most projects you won't need anything more than the free version. It includes everything you need for data pre-processing, exploration, or analysis. In addition, it is available for all computer systems, so you shouldn't encounter any issues when moving a project from one system to another.

In order to install or update anything with Anaconda you need to know a set of commands. It would probably be best for you to check the distribution's online documentation in order to familiarize yourself with all of its features. Now, let's go through some basic commands to get you started. Here's how to install a Python package:

conda install < my_package>

The syntax for removing or deleting a package is the same, just replace "install" with the appropriate keyword. Furthermore, you can install, update, or remove multiple packages at the same time instead of going one by one. Here's how that works:

conda install < my_package1 > < my_package2 > < my_package3 > < my_package233>

As for updating packages, you might want to keep all of the ones you install up to date and that might be a tedious operation if you would have to type the name of every single package you have. Here's how you can update everything at once:

conda update --all

Most package managers included with various scientific distributions work the same. If you don't want to use Anaconda for some reason, you can apply everything you learned in this section to any other application. Even the syntax is usually the same.

Python Toolkit

No machine learning toolkit is complete without a number of modules and libraries. Python is the chosen language in this field because of how easy it is to extend its functionality and push it to the limit. Preparing your work environment with a number of tools is as important as the machine learning tasks themselves.

In this section we will focus on a number of modules and tools that are specifically used in the machine learning field. Some of them are included inside many scientific distributions, however, you need to be aware of them in order to know what to import. Some of the most important tools include Scikit-learn, Pandas, NumPy, and Tensorflow, among many others. Let's briefly discuss the modules and libraries you'll be using and then explore tools like Tensorflow in more detail:

1. Pandas: This is a library that is designed to be used specifically with Python for the purpose of analysing and manipulating data. What defines it is the fact that it gives you functionality to work with numerical tables, as well as time series.
2. NumPy: This is another library meant to be used with Python because it extends the programming language with the ability to manipulate large multi-dimensional matrices. Furthermore, it provides you with a number of operations and functions that you can use on those matrices.
3. Scikit-learn: This machine learning library is the core of your toolkit. It includes all the machine learning algorithms you will be working with, such as k-means clustering, support vector machines, random forests, gradient boosting, and many more. Furthermore, Scikit-learn is designed to be fully integrated with the NumPy library.
4. Matplotlib: This open source Python library is designed to add a plotting functionality to the programming language, as well as the NumPy module. This tool is used for visualisation purposes by allowing you to create a plot from certain data, such as an array. In addition, you can interact with the plot directly once created, in the same environment without switching to another application.

These libraries are considered a must have whether you are a beginner machine learner or a professional. They are all open source and readily available with large, supporting communities built around them. In addition, they are well-documented and have been extensively used to work with real world datasets. This allows you to research other machine learners and see how they applied these tools for their various projects. Never underestimate the power of a popular product. Even if there are better paid ones out there, but they are used only by small groups of people, you should stick with the ones considered as the standard. That way you will always have access to plenty of resources, because you will often need them without a doubt.

Another tool worth mentioning here is Jupyter Notebook, which is similar to an IDE that you may have used when programming. However, it is mostly designed to be used to analyze and process data sets with the help of machine learning algorithms. At first glance, it may look similar to other IDE's such as Visual Studio, however it is quite different. One of the most powerful features it includes is the ability to manipulate plots and visualize them in the same panel where you write all your code. Eliminating the need for other applications and more confusing windows is quite an advantage.

Now that you have some information on the most important tools and you know what they are for, let's explore the final piece of the puzzle, Tensorflow.

Tensorflow

This is another open source Python-focused library that is designed specifically for machine learning. Keep in mind that machine learning is a complex field and it can be difficult to implement algorithms and training models manually. Fortunately, we have tools like Tensorflow to make our job a whole lot easier. Tensorflow isn't just a simple machine learning library. It is in fact a framework that simplifies nearly every aspect of machine learning, such as data acquisition, model training, serving predictions, and refining results until they are as accurate as possible.

Tensorflow was designed by Google, with the purpose of creating an open source library for complex, large scale machine learning problems. This tool is actually packaged together with a number of machine learning models, as well as neural network models, and a number of algorithms. In other words, TensorFlow provides you with everything you need to train neural networks to classify handwritten digits, perform accurate image recognition, build recurrent

neural networks, handle natural language processing and more.

Tensorflow also allows you to create graphs that represent the flow of data. In essence, each graph is a visual structure that shows how the information goes through a number of data processing nodes. Every single node is represented by an operation, and the link between the notes is a multi-dimensional array, also known as a tensor.

All of this functionality is provided through Python. As you already know, this is one of the easiest and most flexible programming languages you can use, and it suits all of your machine learning tasks due its ability to express how to connect complex abstractions together. The tensors and nodes we mentioned earlier are in fact Python objects. Therefore we can determine that Tensorflow programs are Python programs. However, take note that the mathematical operations are not performed using Python. Why? Because Tensorflow uses C++ binaries to write the transformations libraries for optimal performance. Python is only used to connect the dots and allow us to use complex programming operations to link all the components together.

Another advantage of working with Tensorflow is that any application you develop with it can run on any computer system. It can run on a local system, the cloud, Android systems, and so on. For instance, you can use Google's cloud service to run the framework together with Google's Tensorflow processing unit. However, the resulting machine learning prediction models will have to be installed on the system they're supposed to run.

Keep in mind that the largest advantage by far is that Tensorflow provides abstraction for our machine learning development projects. This means that you don't have to manually handle the implementation of a machine learning algorithm. You don't even need to know how to set up one function's output to become the input of another function. While you should know how to do all of this because you want to become a machine learner, after all, Tensorflow provides you with the ability to simply place all of your attention on the application itself and not its inner workings. In other words, you deal with the big picture, while Tensorflow works behind the curtain.

In addition, Tensorflow is all about making the developer's life easier. If you develop any applications through this framework you will benefit from debugging them by evaluating every operation individually. You'll be able to analyze the graphs and then edit them as needed instead of reconstructing them from scratch and then performing the analysis again. This way graph

visualisation is incredibly interactive and user-friendly, especially when you have full access to the graph and the way it runs through an easy to understand user interface.

A final advantage to this industry standard machine learning library and framework is the fact that Google continues to fuel its development and therefore makes it easier to work with and more powerful with each new version. Furthermore, it becomes much less time consuming to launch it into your development pipeline.

As you can see, there are many advantages to using this tool, however, like any other tool it has its drawbacks. One of the biggest issues is caused by the way Tensorflow is implemented. Some training model results are difficult to extract and we can see this when we train a model on one system and then on another system. The results will vary to some degree. The problem is that the reason for this is a complicated matter and therefore we can’t find a fix-all solution. That is why, to achieve optimal performance and accuracy, you need to know which machine learning techniques to apply, even if you don’t do all the dirty work yourself.

Chapter 4: Data Preprocessing

Now that you have all your tools and you have an understanding of basic machine learning, it's time to take the first step and import some data for preprocessing. In this chapter we are going to mainly use Pandas because it allows us to load tabular data such as tables, spreadsheets, or even entire databases. It works by creating a readable data structure from the tabular data we have.

Importing Data

To understand the process, we are going to import a well-documented, open source dataset called Iris. Let's get to work! Type the following lines:

```
In: import pandas as pd

iris_filename = 'datasets-ucl-iris.csv'

iris = pd.read_csv(iris_filename, sep=',', decimal='.', header=None,

names= ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',    'target'])
```

After importing the dataset we need to create a separator (sep) and a decimal character (decimal). For now all we have is the "iris" object which is in fact a pandas data frame. You'll notice that is actually very similar to a Python list or dictionary. See? Learning those Python basics is already paying off. Now let's take a look at the content:

```
iris.head()
```

You should now see a table containing the first five rows. Why five? If you don't set any parameters like in this case, then that's the default. If you want a certain number of rows, simply add the desired number between the parentheses. Now let's see the columns of the table by typing:

```
iris.columns

Out: Index(['sepal_length', 'sepal_width', 'petal_length',

'petal_width', 'target'], dtype='object')
```

You should now have a list and you can use it in order to extract some

information out of it:

```
Y = iris['target']

Y

Out:

0       Iris-setosa

1       Iris -setosa

2       Iris -setosa

3       Iris -setosa

...

149     Iris-virginica

Name: target, dtype: object
```

The “Y” in our line of instruction represents a Pandas series, which is something like an array, however, in this case it only goes one way. Furthermore, you’ll notice that the index class is identical to a Python dictionary index. Let’s see what happens when we type:

```
X = iris[['sepal_length', 'sepal_width']]
```

We used the column index in order to call them and receive a matrix. Earlier we only received a pandas series with one dimension because that was all we asked for. However, this time we want multiple columns, which translates to a matrix. Don’t forget that a matrix is simply an array of objects that are placed in intersecting rows and columns.

The last thing we might be interested in before switching to the preprocessing phase itself is the dataset’s dimension. Let’s find it out:

```
print (X.shape)

Out: (150, 2)

In:  print (Y.shape)

Out: (150,)
```

As you can see, what we have is a tuble. Now we know the array’s size and we can start preprocessing the data.

Preprocessing Data with Pandas

Now that the data is loaded we can begin preprocessing it with Pandas. The first thing we might consider is applying a mask in order to use a function only on a specific part of a row. Keep in mind that a mask in this case represents a collection of Booleans that are expressed whenever a line is chosen. Take note that Booleans are true or false values. Let’s see this step in code and things will become clearer:

In: mask_feature = iris['sepal_length'] > 6.0

In: mask_feature

0 False

1 False

2 False

...

146 True

147 True

148 True

149 True

Our objective during this step is to select all the iris items with a sepal length greater than 6. You’ll notice that those that hold a smaller value return a false observation and therefore don’t fit with our request. Next, we are going to use a mask in order to relabel one of our targets. Type the following lines:

In: mask_target = iris['target'] == 'Iris-virginica'

In: iris.loc[mask_target, 'target'] = 'New label'

We have selected iris-virginica as our target and have renamed it as “new label”. We can verify this process by asking for the list of labels that are part of the column:

iris['target'].unique()

Out: array(['Iris-setosa', 'Iris-versicolor', 'New label'], dtype=object)

You’ll notice that we’re using the “unique” function. This is used to analyze our

list after the update. In addition, we can group all of our columns together in order to verify the stats. Let’s look at the example:

grouped_targets_mean = iris.groupby(['target']).mean()

grouped_targets_mean

Out:

grouped_targets_var = iris.groupby(['target']).var()

grouped_targets_var

Out:

First we group our table columns by using the groupby function. If by any chance you have experience working with SQL databases, this function is identical to “GROUP BY” in SQL. In the next step we have to calculate the mean value by using the mean function. Keep in mind that you can use this operation on any number of columns. Furthermore we apply certain functions that are unique to Pandas, such as summation and variance. For now, we are still working on a data frame and therefore we can connect a number of operations. So far we used the ”groupby” function in order to compile all of our observations based on their label and then verify whether there is a difference between certain values within each group.

When you work with real data instead of nice, clean examples such as the iris datasets, you will sometimes have to also deal with time series. In machine learning, or data science, time series refer to certain data points that are marked on a graph or list by following a chronological order. In simpler terms, it’s a series of points in time with an equal distance between each other. This sequence of time data has a wide application anywhere between statistical analysis and counting dark spots on the sun. Take note that datasets with time series are often quite messy. Therefore you have to clean up the data points. The easiest way to do this is to use a mean function, like this:

In: smooth_time_series = pd.rolling_mean(time_series, 5)

Keep in mind that you can also use the median operation instead. Furthermore, you can use a Pandas-specific function called “apply” in order to find out how many non-null values we have in every row:

In: iris.apply(np.count_nonzero, axis=1).head()

Out: 0 5

1 5

2 5

3 5

4 5

dtype: int64

Keep in mind that this function has a number of applications. Make sure to check the Pandas' documentation for more information. Now, the final step is to use the “applymap” function in order to go deeper at the basic element level. For the sake of this demonstration, we are going to assume that the length of each cell’s string representation is something we will later need to process:

In: iris.applymap(lambda el:len(str(el))).head()

That’s it, but not quite. Data preprocessing is a vital step that ensures we are dealing with high quality data. We require useful information because it will certainly affect the way our machine learning model will be trained. We have covered some of the basic operations, however, there are many more for you to explore.

Chapter 5: Machine Learning Algorithms

Now that you understand the concept of machine learning and you've assembled all of your tools, it's time to start exploring the most powerful algorithms. Keep in mind that while you will learn about each algorithm and technique individually, they are rarely used like this in a real world scenario. When dealing with complex datasets you will often combine supervised machine learning techniques with unsupervised machine learning techniques, as well as a number of other processes.

In this chapter we are going to focus on a number of algorithms that work well separately when working with ideal data sets, and even better when combined to achieve high accuracy and efficiency. Some of the algorithms we'll explore include linear regression, decision trees, and support vector machines.

Linear Regression

Many machine learning problems, such as prediction analysis, are handled using linear regression. This technique is powerful in predicting an outcome with high accuracy as long as the training data is valuable and doesn't contain too much noise. Keep in mind that this is why you need to perform data pre-processing, to clear out some of the noise and outliers that can affect the results.

One of the characteristics of linear regression is calculating the prediction value based on independent variables. In other words, you should opt for this algorithm whenever you have to calculate the link between a number of variables. However, take note that linear regression can only handle regression problems, and not classification problems. There are other algorithms for that.

Now let's see an example on how to apply linear regression using Scikit-learn and an open source dataset. We will use the Scikit library because it contains a number of machine learning specific algorithms, as well as certain public datasets. One such dataset is the Boston housing market dataset, which is well-documented and clean. In order to use linear regression efficiently, we will divide the whole set into a training set (80%) and a testing set (20%). Let's get to work:

```
from sklearn.datasets import load_boston
```

```
boston = load_boston()
from sklearn.cross_validation import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(boston.data,
boston.target, test_size=0.2, random_state=0)
```

As you can see, the Python code is easy to read and quite understandable without much explanation. We have imported our dataset and have performed a split in order to obtain a training set and a test set. The splitting function is a perfect example of the tools that the Scikit-learn library contains. You will notice that we have declared the percentages by stating the target test dataset should have a value of 0.2. This means that it will contain 20% of the data. Furthermore, we have a random state parameter which is needed to define the value which will handle the random number generating during the split. Basically, it randomly selects the data that will be categorized as training data or testing data.

In the next phase, we need to apply a regressor in order for our linear regression algorithm to make an accurate prediction. Once it's implemented all we need to do is measure the results to see how it performed. Keep in mind that there are many factors that can influence the accuracy, such as random errors, mislabeled data, outliers, noise, and so on. That is why the pre-processing you perform is rarely enough, and you will have to apply other algorithms to improve the result. This is something you need to keep in mind for the near future when you practice on your own. Now let's see our regressor and the results we obtain:

```
In: from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, Y_train)
Y_pred = regression.predict(X_test)
from sklearn.metrics import mean_absolute_error
print ("MAE", mean_absolute_error(Y_test, Y_pred))
Out: MAE 3.9318708224346
```

That's it! As you can see linear regression is easy to implement as long as you use Scikit-learn to lessen the burden. The result we obtained, however, while not bad, is not exactly good either. It shows that even when we have fairly clean data, there is always room for improvement with other algorithms and techniques. Furthermore, whenever you have to make a choice between machine

learning algorithms, you will actually be balancing prediction accuracy with speed. In essence, it comes down to those two factors. Whenever you opt for one, the other one is sacrificed. With time and experience you will learn how to balance them based on the type of data you are working with and what kind of results you need to obtain.

Now, let's see another type of algorithm, namely the support vector machine.

Support Vector Machines

If you need a versatile algorithm that can handle nearly everything for you, then a support vector machine is precisely what you're looking for. While the linear regression algorithm we discussed earlier is only used for regression problems, support vector machines can handle both regression, as well as classification, and even the detection of outliers. This technique is one of the foundational algorithms every machine learner needs to study and learn how to implement.

Support vector machines are popular because they are powerful without being resource intensive. Usually, the more you push for accuracy, the slower the data processing will go because it will require more from your computer system. However, support vector machines require a lot less power without affecting the prediction accuracy. Furthermore, you can use this algorithm to also clean most of the noise within the dataset, thus reducing the amount of steps you need to perform during the pre-processing phase (though it's still advisable to perform all of them).

So why are support vector machines so important? Where are they used? Because of their power and versatility they are implemented in a variety of fields. For instance, they are used for facial recognition software, text classification, handwriting recognition, and in the medical industry as well for classifying genes. There are many more uses for this algorithm because it separates the data points within the dataset with great efficiency and that allows for accurate enough results to make it usable in complex applications.

We will go through an example to see how a support vector machine is implemented and how it works. Keep in mind that you will again have to make use of the Scikit-learn library in order to reduce the time it takes to implement the algorithm. As you can see, this tool is irreplaceable whether you're interested in machine learning, data science, or even data mining. With that being said, we

will use a support vector machine in order to predict how valid a bancnote is. This implies image classification, and as mentioned earlier, this algorithm is ideal for this process. The algorithm implementation will be simple because the classification will be binary. All we need to establish is whether a bancnote is authentic or fake, therefore true or false.

Take note that we will have a number of attributes that describe the banknote. In addition, we need to establish the separation line between the two dimensions in order to avoid any miss-classifications. While other classification algorithms can perform the same process, support vector machines have a unique approach. This technique determines the decision limit by calculating the maximum distance between certain data points that are in close proximity to every single class. The decision limit needs to be optimal and therefore we cannot just randomly select it. These optimal data points are known as support vectors, hence the name of our algorithm. Keep in mind that they are a mathematical concept and we will not dig too deep into how they are calculated and so on. At this stage you don't need to know the core details that lie in fairly complex algebra.

Now, let's see a practical example on how to apply a support vector machine that does everything we discussed. For this demonstration you will need to use several libraries and packages. Here's the code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset = pd.read_csv ("bank_note.csv")

print (dataset.shape)
```

The shape function is used to extract some information about the dataset. Namely, it tells us how many rows and columns we have. For the purpose of this exercise we are interested only in the first 5 rows (out of over 1,300).

Now type the following line to see the result:

```
print (dataset.head())
```

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.454590	8.1674	-2.4586	-1.46210	0

2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

The columns represent the attributes we mentioned earlier, namely the variance, skewness and so on. Now we can start processing this information in order to start building our training and testing sets. However, first we need labels:

x = dataset.drop (‘Class’, axis = 1)

y = dataset [‘Class’]

The first command instructs the system to store every column into a variable. The only exception is the class column and that is why we use the drop function in order to drop it. The reason we did so is because we want to store it separately in another variable, and that is what we do with the second command. Now we have the dataset attributes inside “x” and the labels inside “y”. This is the first processing step, separating the attributes from the labels.

The next phase involves the data separation in order to create our train and test sets. Simply use the function you used earlier for linear regression to perform this step, namely train_test_split. The breakdown should respect the same ratio as before.

Next, we will train the support vector machine, however, first we need to import it from the Scikit-learn library, otherwise we do not have any access to the algorithm or any of its components. Let’s see the code:

from sklearn.svm import SVC

svc_classifier = SVC (kernel = ‘linear’)

svc_classifier.fit (x_train, y_train)

Now let’s see the prediction with the help of the test data:

pred_y = svc.classifier.predict(x_test)

We’re almost done. All we need to do now is to see how accurate the prediction is. In the case of the support vector machine we will have to apply what is known as a confusion matrix, which holds a number of metric, such as precision, recall, and a few others. While the name of the confusion matrix might sound confusing, you should know that it is a basic table that contains information about the performance of the data classification process. It will list the positive and negative cases, as well as the false positive and negative cases. These

metrics are then used to establish the final accuracy result. Here's how it all looks in code:

```
from sklearn.metric import confusion_matrix

print (confusion_matrix (y_test, pred_y)
```

This is how the results will look:

```
[[160          1]

[1     113]]

Accuracy Score: 0.99
```

Let's also see the detailed report:

```
from sklearn.metrics import classification_report

print (classification_report(y_test, y_pred))
```

These are the final results:

precision	recall	f1-score	support
0.0	0.99	0.99	0.99	161
1.0	0.99	0.99	0.99	114
avg / total	0.99	0.99	0.99	275

As you can see from these numbers, the support vector machine has performed magnificently well because we managed to obtained a 0.99 accuracy value without applying any other algorithms or data-preprocessing techniques. This is why support vector machines are used in so many technical fields that require pin-point accuracy.

Next up, you are going to take a look at another powerful algorithm called a decision tree.

Decision Trees

Decision trees, just like support vector machines, are incredibly versatile because they can perform both regression and classification operations. This means they are extremely useful when working with large datasets and you should get acquainted with them. Furthermore, decision trees form the basis of another type

of machine learning algorithm known as an ensemble method, particularly the one known as a random forest. It's all fairly logical. If you have a lot of decision trees they form a forest, right?

As the name suggests, decision trees are used to model decisions, including the determination of any kind of consequences and system resources needed for the entire process. In addition, everything is presented visually in a graph. In other words, as a result you will have a flowchart that contains every test you perform. For instance, let's say you flip a coin a certain number of times. Every time the results will be recorded and you will have a number of tree branches that represent them, together with the leaves that represent class labels.

This supervised machine learning algorithm is considered to be one of the most powerful ones because it offers accurate results without sacrificing too much efficiency or requiring too many resources from your system. Furthermore, the graph system allows you to easily interpret the data and therefore make adjustments as needed instead of going through another tedious process that would achieve the same thing but with a lot more work. With that being said, let's discuss all the other advantages provided by decision trees:

1. User-friendly: Unlike many other algorithms or techniques, decision trees are logical even to those without significant machine learning or mathematical knowledge and experiences. You don't need to be a data scientist to start working with them. They follow a simple, programming-like structure, similar to Python's conditional operations. A decision tree follows the "if this, then do that, else do this" logic. Furthermore, the results are clear and do not require even a fundamental knowledge of statistics in order to interpret them. That is the benefit of having an inbuilt graph that clearly explains the data you are analyzing.
2. Clean data: As you already know from the chapter about pre-processing your data, clean data yields accurate results. Therefore, no matter which algorithm you use you always have to eliminate as much noise and as many outliers as you can by using certain methods and techniques. However, the benefit of using decision trees is that they are barely affected by these factors. Outliers and noise will only be a minor inconvenience and you will often be able to ignore them completely and still achieve an accurate result.
3. Versatile: As you already know, certain machine learning algorithms

are designed to work only with certain data types. Some of them can only be applied to categorical data, while others are specifically used on numerical data. Often for this reason they need to be combined when working with complex datasets that contain both types of information. However, decision trees can handle the training process for both data type, therefore significantly reducing your workload.

4. Exploring data: Data exploration is a valuable step that can be time consuming because you need to identify the most valuable data points and then establish a connection between them. Decision trees, however, excell at exploratory analysis and you will spend a lot less time performing this step. In addition, you will be able to create new features a lot more efficiently in order to boost the accuracy of the predictions even further. Exploring data can be exhausting because in the real world you will often deal with thousands of variables and finding the most important ones can be difficult without the appropriate tool.

Now that you have an idea about decision trees and why they are a preferred algorithm amongst machine learners, you should also learn which disadvantages they bring. Keep in mind there is no such thing as the perfect algorithm, so let's see what kind of difficulties you will be facing when working with decision trees:

1. Overfitting: This is probably the biggest problem you will face. Decision trees tend to exaggerate and build overly complex decision trees that can't perform accurately on general data. However, the issue of overfitting isn't only found in this case. Many algorithms have to deal with it. The easiest solution you can deploy to limit the risk of overfitting is the application of parameter limitations.

2. Categorization: When your data set has a large number of continuous numerical variables, decision trees will encounter difficulty when it comes to categorizing them. Information is often lost during this process. Take note that continuous variables are considered to be anything with a value that is specifically defined between a certain minimum and maximum. For instance, let's say that we have some data which specifies that only people between 21 and 45 can join the military. The age variable is considered to be continuous because we have set limits.

3. Picky algorithms: The algorithm doesn't always choose the most accurate and efficient decision tree from which you draw your conclusion. This is simply the nature of the technique. However, the solution is simple. All you need to do is train multiple trees in order to sample a larger number of features randomly and find the most accurate solution.

You learned earlier in this section that decision trees are used to perform regression tasks, as well as classification tasks. However, this doesn't mean that you use the identical method for both tasks. You need to split the decision trees into two different categories, namely the regression trees and classification trees. The basics are still the same for both categories, however, there are a number of differences you should be aware of:

1. Regression trees are specifically applied for tasks that involve continuous dependent variables. At the same time classification trees are needed when dealing with categorical dependent variables.
2. The values that we extract as the result from the training data are determined differently for each category of decision trees. In the case of classification trees the value is calculated as the mode value of the total observation. This means that if we have an unknown data observation, we can predict with the help of the mode value. Keep in mind that mode values are the values that we find the most frequently in a given set of values. On the other hand, regression trees extract the value from the training data in the form of the mean of the total observations. This mean value is then used to calculate any missing data observations.
3. Regression and classification trees rely on binary splitting. This concept involves going from the top of the tree to the bottom. This means that when we have observations in one region of the tree, there are a number of branches below it that are divided. The division happens within the predictor space. This concept is also often referred to as greedy because the algorithm seeks the variable that is needed for the current split that is being analyzed. The operation does not take any future splits in consideration and that is why sometimes we don't have the most accurate results. This issue needs to be taken into consideration in order to counteract its effects with various methods and create a better decision tree.

Now let’s discuss more about the concept of splitting and see where it occurs. As already mentioned, splitting directly affects the accuracy of the decision trees. This means that if we are applying a number of algorithms on a decision tree, they will all affect the way a data node is divided into a number of subnodes. At this time you should keep in mind the differences between the two decision tree categories. The subnodes that are created as a result of splitting will maintain some form of uniformity. In addition, the node splitting is performed on every variable, however only the most uniform subnodes are selected. This process involves some mathematics and we will not dig deeper into the details. It’s enough to understand the theory behind the process in order to start building the decision trees. With that being said, let’s see how to work with them in an application.

Decision Trees in Practice

In order to create the decision trees you need to start from the root node. Once you access that data you can select a certain attribute and then perform a logical test on it. As a result you will be able to branch out from the outcome to any subsets of data that lead to a desirable result from a child node. You will need to perform recursive splitting on these child nodes in order to clean up the leaves and keep only the near-perfect ones. These leaves will represent the data samples that come from the same class. In addition, you might consider pruning the trees in order to eliminate the excess sub-sections of the trees which do not lead to any boosts in accuracy. This is one of the purposes of the splitting process. Once it is complete you can choose the most optimal split and the most useful feature.

Now let’s finally stop with the theory and start looking at the practical application. We are going to create a decision tree. For the purpose of this demonstration we will need Scikit-learn once again because we will be using another public dataset called Iris. Let’s start by importing everything you need.

```
%matplotlib inline

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv('Iris.csv')

df.isnull().any()
```

After importing everything we have also performed a null value check with the help of the last line. Once we determine that no values are absent we can start examining the information. As always, the first step is learning about the variables and values by reading a summary analysis of the dataset. Once you know what you're up against we can start looking at the connections between the columns. To do this check we have imported the seaborn module because it provides us with a tool called "pairplot". This tool is in fact a function that plots the connections and makes it much easier for us to visualise them. At this point you can use a certain color for each column. For instance, you can manipulate the hue for the species section in order to find any outliers in that category. Here's how you can do this with Python code:

```
sns.pairplot(df, hue = 'Species')
```

If you decided to use this particular dataset, you will notice that we do have a number of outliers, but they aren't dominating. They may be the result of unknown anomalies or some faulty data that snuck in. Take note that these datasets you are working with are extremely clean when compared to real world data. Most data contains a great deal of noise and outliers and you have to thoroughly clean them. For now, we can ignore the outliers because they aren't prevalent enough to disturb our results significantly.

The next step is to divide the dataset into a training and a test set with the same method we used before. However, now that we're working with decision trees we will change the split to a 70 to 30 ratio instead of the usual 80 to 20. Let's examine the code:

```
all_inputs = df [['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values

all_classes = df ['Species'].values

(train_inputs, test_inputs, train_classes, test_classes) =

train_test_split (all_inputs, all_classes, train_size = 0.7, random_state = 1)
```

You will notice that the random state parameter is no longer set to 0 in order to guarantee a random distribution. This time we want to make sure the the split is performed identically no matter how many times we perform it. This way we

will always be able to recreate the dataset if needed. Now let’s apply the decision tree classifier:

```
dtc = DecisionTreeClassifier()

dtc.fit (train_inputs, train_classes)

dtc.score (test_inputs, test_classes)
```

The result should look something like this:

```
0.95555555
```

We have managed to obtain 95% accuracy, which is quite an excellent result considering that we haven’t performed any cleaning and we decided to ignore the outliers. This is why decision trees are so powerful and therefore popular in training various models. You can achieve high accuracy without performing too many operations.

Chapter 6: Machine Learning & Neural Networks

Neural networks started appearing in the 50s as a new technique to create computer systems that seek to mimic the human learning process. Keep in mind that there is very little connection between a human neural network and an artificial one. Unfortunately, the human brain is still too complex to turn into an artificial version. However, the name of the machine learning technique is mostly used for inspiration.

Neural network neurons have nodes, which actually are a form of processing code that is meant to execute repeatedly until a solution to a problem is found. When the solution is known, then the result is sent to the next node within the same layer. When all the nodes within a layer have finished processing, the final one will communicate the result to the first node in the next layer. The human neural network does pose a number of similarities, however, it is also capable of establishing real time communication between all the nodes in every single layer at the same time. This is what makes the human brain truly special and powerful and why it cannot yet be replicated. The concept behind it is known as parallel processing. In artificial neural networks processing is linear, going from one node to the next.

A neural network isn’t a particular machine learning algorithm. Instead, it is a subfield within machine learning, which contains certain systems that attempt to emulate some aspects of human thought processing. Neural networks transmit information through a network of nodes. Once the information is analyzed and classified, it is communicated to the neighboring node. Once there, another round of analysis and classification will take place and the cycle repeats itself. Take note that neural networks consist of a number of layers. The usual structure contains an input layer and an output layer. The most basic form of a neural network can even have only one layer, however, that is a fairly primitive model that is never used in practice. In addition, between the two layers we have a number of hidden layers, anywhere from none to as many as you need. These layers are the ones that handle the data that is sent from the input layer. Keep in mind that a standard network contains three layers, while those that have hundreds of layers are referred to as Deep Learning systems.

In this chapter we will focus on the classic feedforward neural network, as well as the recurrent neural network.

Feedforward Neural Networks

This is the most basic type of neural network. Data is transmitted in a linear fashion and it can only go in one direction. It is passed from the input layer to the hidden layer and finally to the output layer. That's it. This type of network doesn't even iterate over the data like the more modern systems. Once the process is over, it's over. Data doesn't loop through a cycle. Once the operation is complete, we have the result. However, there are two types of feedforward networks.

The first is known as the single layer perceptron and this is as basic as it gets. There is only one layer and the data is passed in a linear fashion from the first node to the last. When the input passes through one node it is weighted and if the next node determines the weight passes a minimal limit, it will either be activated or deactivated.

The multi-layer perceptron, on the other hand, needs to have at least two layers, though in reality it consists of more than two. The data transmission is still linear, however the output of a previous layer becomes the input of the current layer. Take note that when there are more than two layers involved, this type of neural network will usually make use of the concept of backpropagation. This means that the actual output will be compared to the output that was expected. The difference between the two is then calculated and the degree of error is established. This error value is then sent throughout the neural net in order to adjust it as it passes through all of the nodes again. These adjustments will lead to a closer result, and the process is then repeated until the result matches as accurately as possible. Keep in mind that depending on how many layers there are, a large number of iterations may take place in order to establish a satisfactory state. Furthermore, multi-layer perceptrons that take advantage of backpropagation cannot be called true feedforward neural networks anymore. They become multi-layer networks that can be some of the most powerful and versatile algorithms in the entire field of Machine Learning.

Recurrent Neural Networks

In order to understand recurrent neural networks you first need to understand the concept of sequence modeling. Keep in mind that machine learning is often

applied to sequences and when we do so we often change the input sequence to the output sequence. For instance, you might want to convert the sequence of various sound pressures into a sequence of words. This means that when we don't have a specific target sequence, we can attempt to predict the next word in the input sequence by using a teaching signal. Then the output sequence becomes the input sequence with an additional step. This is a fairly natural process, and it makes a lot of sense in this case. However, it could be problematic when applying the same principle to an image. Imagine predicting a section of a picture by using another section which you already know. Predicting a simple term will make use of both supervised and unsupervised machine learning and therefore eliminate any barriers between them. What's special though, is the fact that both types of techniques can be used, but without requiring two different teaching signals.

Models that are considered without memory are best suited for this task. To be more precise, it's the autoregressive models we are referring to because they can predict the next word based on the previous known words by using what's known as delay taps. Feedforward neural networks that have at least one layer with hidden units are in general considered to be autoregressive. However, if we introduce some kind of hidden state to the generative model we will obtain a new model that can store information inside the hidden state.

With all that in mind, let's switch back to the topic of recurrent neural networks. In essence they are perceptrons. However, keep in mind that perceptrons have specific states. Recurrent neural networks create connections between the passes and they are more powerful due to the combination of two features. First, they hold distributed hidden states and therefore they can store a lot more information that can be later passed forward. Secondly, the hidden state can be updated due to non-linear dynamics. In other words, as long as a recurrent neural network has plenty of time and a large enough number of neurons, it can calculate anything, provided the computer can handle it. This type of network is capable of a lot of things because it has the ability to implement a number of components that are each designed to contain some data. These components can then run and interact simultaneously in order to perform complex operations.

However, there is one downside when it comes to recurrent neural networks, namely the disappearing gradient problem. Over a long enough period of time information can eventually be lost. Take note that in this case the information refers to the weights, meaning that only the past information is lost, not the current neuron states. For instance, if a weight reaches a value of zero or a

million, the previous state will be eliminated.

In conclusion, we can determine that recurrent neural networks are used in many fields because they essentially offer the ability to autocomplete missing information.

Convolutional Neural Networks

Finally, we have the convolutional neural networks that appeared in the late 90s when a method was devised to recognize handwritten digits. This new concept relies on feedforward networks with backpropagation and a large number of hidden layers. These layers contained a large number of maps that held replicated units, as well as their output pooling and a large network that can handle multiple characters simultaneously. This way a new training system was born.

Convolutional neural networks don’t have much in common with the other two types we discussed because they are mostly used for image and audio processing. For instance, if you use one you can classify information based on what images you feed the network. Essentially, it will scan the images a number of pixels at a time with a good measure of overlapping. For instance, if you have an image that is 1000 x 1000 pixels in dimension, you will only use a section of 10 x 10 to send it to the neural network. Once that section is analysed, you send another 10 x 10 section by moving only one pixel to the right, therefore creating a significant amount of overlap.

The next step implies sending the input data through a number of convolutional layers and not normal layers. The difference is that the neural nodes are not linked to each other. Every single node will only communicate with cells that are in their neighborhood. Furthermore, convolutional layers sometimes also involve pooling layers. The concept of pooling is about removing unnecessary details. For instance, if you have a 2 x 2 pixels section colored in red, only the pixel with the highest amount of red will go through while the other will be discarded.

Conclusion

You have successfully completed your journey through *Python Machine Learning* ! Take a deep breath and congratulate yourself, because you have managed to explore a great deal of theoretical concepts, specialized tools, and practical applications of machine learning algorithms. Don’t forget that you are dealing with a relatively new field and it requires a great deal of patience and determination on your part.

The purpose of this book is to guide you through the foundational concepts behind machine learning, such as linear regression, decision trees, support vector machines, and various types of neural networks. Furthermore, it guides you step by step through all the steps you need to take to choose the right tools for the job. You learned the basics behind using Pandas, Scikit-learn, and even Tensorflow. All of them form a valuable kit, as they are the standard in the industry.

Furthermore, all the theory you learned you have put to the test by working with real datasets and seeing the results firsthand. However, you should keep in mind that going through these demonstrations is not enough. You should create your own training models and work on other datasets as well in order to solidify everything you learned. Machine learning is not an easy topic, however, you should now be more than enough prepared to tackle a large number of challenges. You have all that is required to get a headstart in this field, so continue practicing until you become a master machine learner!

Bibliography

Albon, C. (2018). Machine learning with Python cookbook: practical solutions from preprocessing to deep learning. Sebastopol, CA: OReilly Media.

Garreta Raúl, & Moncecchi, G. (2013). Learning scikit-learn: machine learning in Python: experience the benefits of machine learning techniques by applying them to real-world problems using Python and the open source scikit-learn library. Birmingham, UK: Packt Publishing Ltd.

Géron Aurélien. (2019). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: OReilly Media.

Zaccone, G., & Karim, M. R. (2018). Deep Learning with TensorFlow: Explore neural networks and build intelligent systems with Python, 2nd Edition. Birmingham: Packt Publishing.