DTAM

DIGITAL TRANSFORMATION IN ADVANCED MANUFACTURING

Co-funded by the Erasmus+ Programme of the European Union

Basic Concepts in Machine Learning

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 3:57 PM
Book:	Basic Concepts in Machine Learning		

Description

In this section you will get introduced to what is Machine Learning.

Table of contents

1. Definition

2. Machine Learning Categories

3. How it’s made

4. Why insist on Machine Learning though?

5. Examples of machine learning use

6. Machine Learning advantages and disadvantages

6.1. Advantages

6.2. Disadvantages

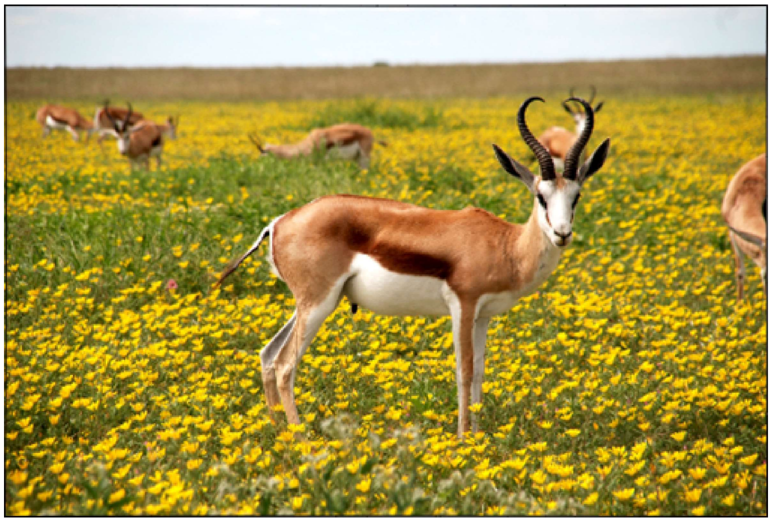
6.3. Advantages applications

7. Machine learning for manufacturing applications

8. Global market examples

1. Definition

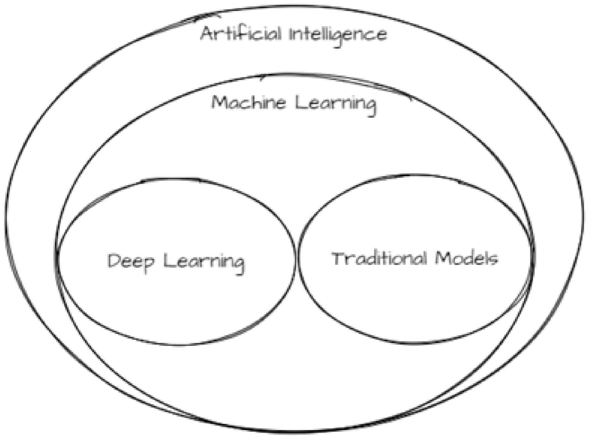
Humans try to understand their environment by observing it and creating a simplified (abstract) version of it called a mental model or simply **a model**. The method is known as induction, and the building of such a model is known as **inductive learning**. Furthermore, humans may organize and link their experiences and observations by forming new structures known as mental patterns or simple **patterns [1]**. For example, a human can easily identify all the animals in the following Figure 1.1. But can a computer do that as efficiently? And if yes, how?



Machine learning is the creation of models or templates based on a set of data, from a computer system.

A more "scientific" definition of Machine Learning is the one bellow:

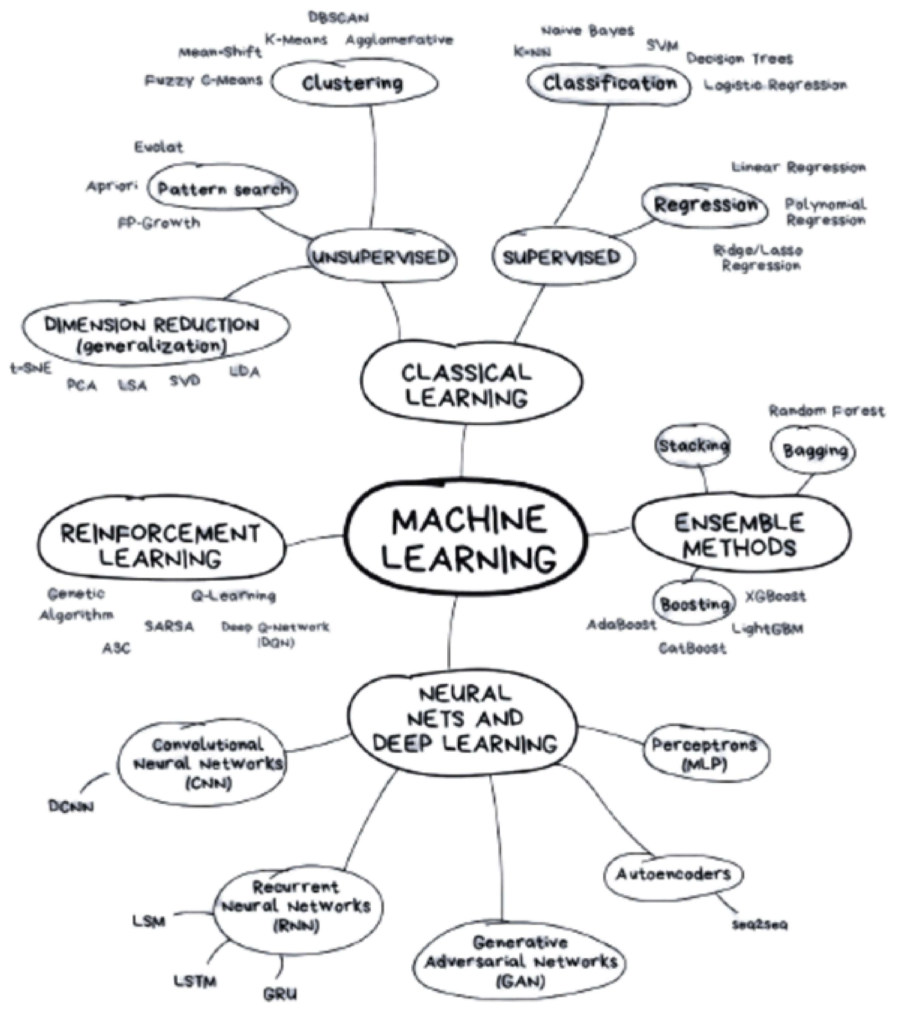
Machine learning (ML) is a branch of research devoted to understanding and developing 'learning' methods, or methods that use data to improve performance on a set of tasks. [2] Artificial intelligence is highly associated with ML and many scientists consider it to be a subsection of it as Figure 1.2 shows. Machine learning algorithms create a model based on training data to make predictions or judgments without having to be explicitly programmed to do so. [3]



Machine learning algorithms are utilized in a broad range of applications, including medicine, email filtering, speech recognition, and computer vision, where developing traditional algorithms to do the required tasks is difficult or impossible. [4] Machine learning is also closely connected other scientific fields such as **computational statistics**, which focuses on generating predictions with computers; nevertheless, statistical learning is not all machine learning. The discipline of machine learning benefits from the study of mathematical optimization since it provides tools, theory, and application fields [5]. **Data mining** is also, a closely connected topic of research that focuses on exploratory data analysis via unsupervised learning. Furthermore, some machine learning implementations employ data and neural networks in a way that replicates **the operation of a biological brain [6]**.

2. Machine Learning Categories

As we saw, ML is part of a bigger picture but also itself consist of many, many parts. Figure 1.3 depicts the various Machine Learning branches. In this course we are going to focus on classical learning methods.



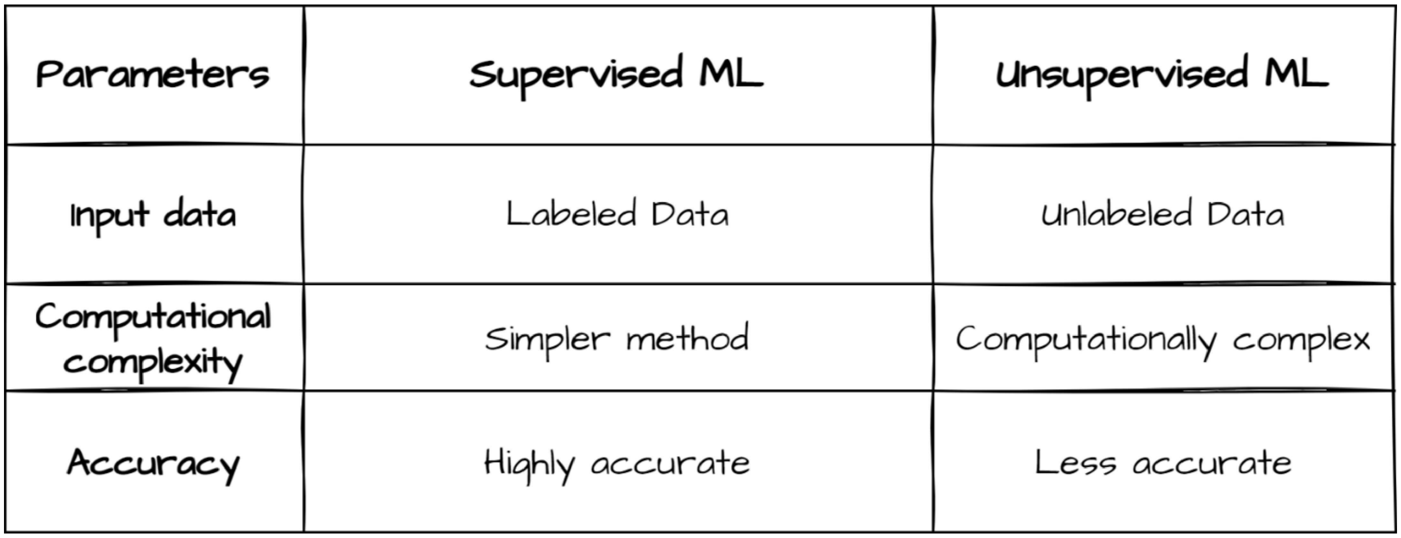
Depending on the type of the "signal" or "feedback" provided to the learning system, machine learning systems are generally categorized into three major categories [8]:

o **Supervised learning:** A "teacher" presents the computer with sample inputs and desired outputs, with the purpose of learning a general rule that maps inputs to outputs.

o **Unsupervised learning:** is when the learning algorithm is not given labels and is left to uncover structure in the data on its own. Unsupervised learning can be a goal in and of itself (finding hidden patterns in data) or a means to an end (finding hidden patterns in data) (feature learning).

o **Reinforcement learning:** occurs when a computer program interacts with a dynamic environment to achieve a certain objective (such as driving a vehicle or playing a game against an opponent). The software is given input in the form of incentives as it navigates its issue space, which it strives to optimize.

The most common ones are the first two and they will be further explained in the upcoming sections. The comparison between supervised and unsupervised learning techniques can be summarized in the following Table 1.



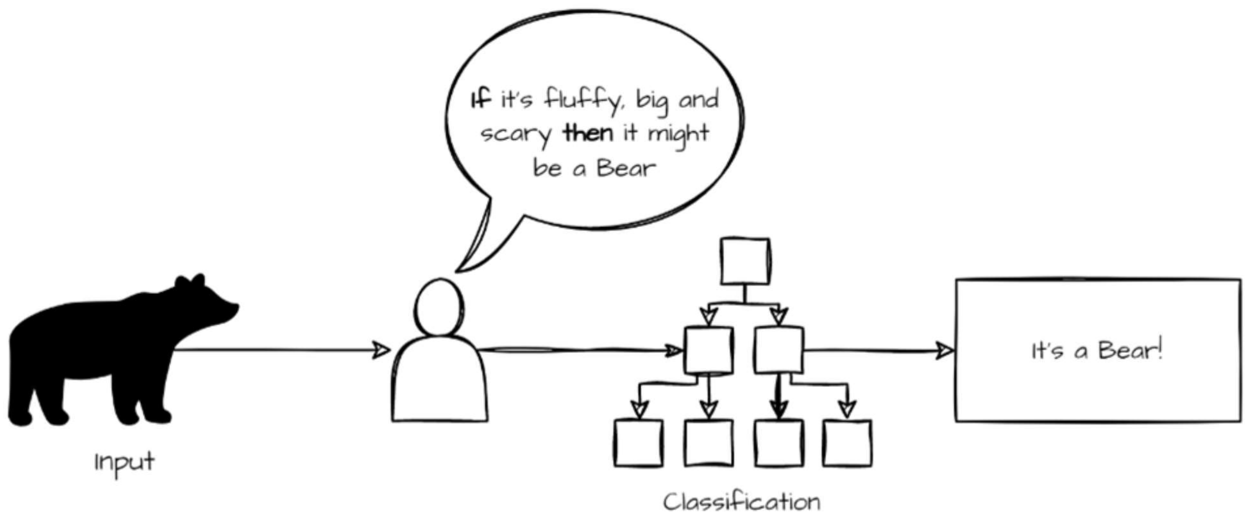
Parameters	Supervised ML	Unsupervised ML
Input data	Labeled Data	Unlabeled Data
Computational complexity	Simpler method	Computationally complex
Accuracy	Highly accurate	Less accurate

Other techniques that do not cleanly fit into this three-fold categorization have been created, and often more than one is employed by the same machine learning system. For instance, Boosting Methods and Meta Learning [9]. Moving forward to the present trends, **Deep learning** has been the primary technique for much continuing work in the field of machine learning as of 2020 [10].

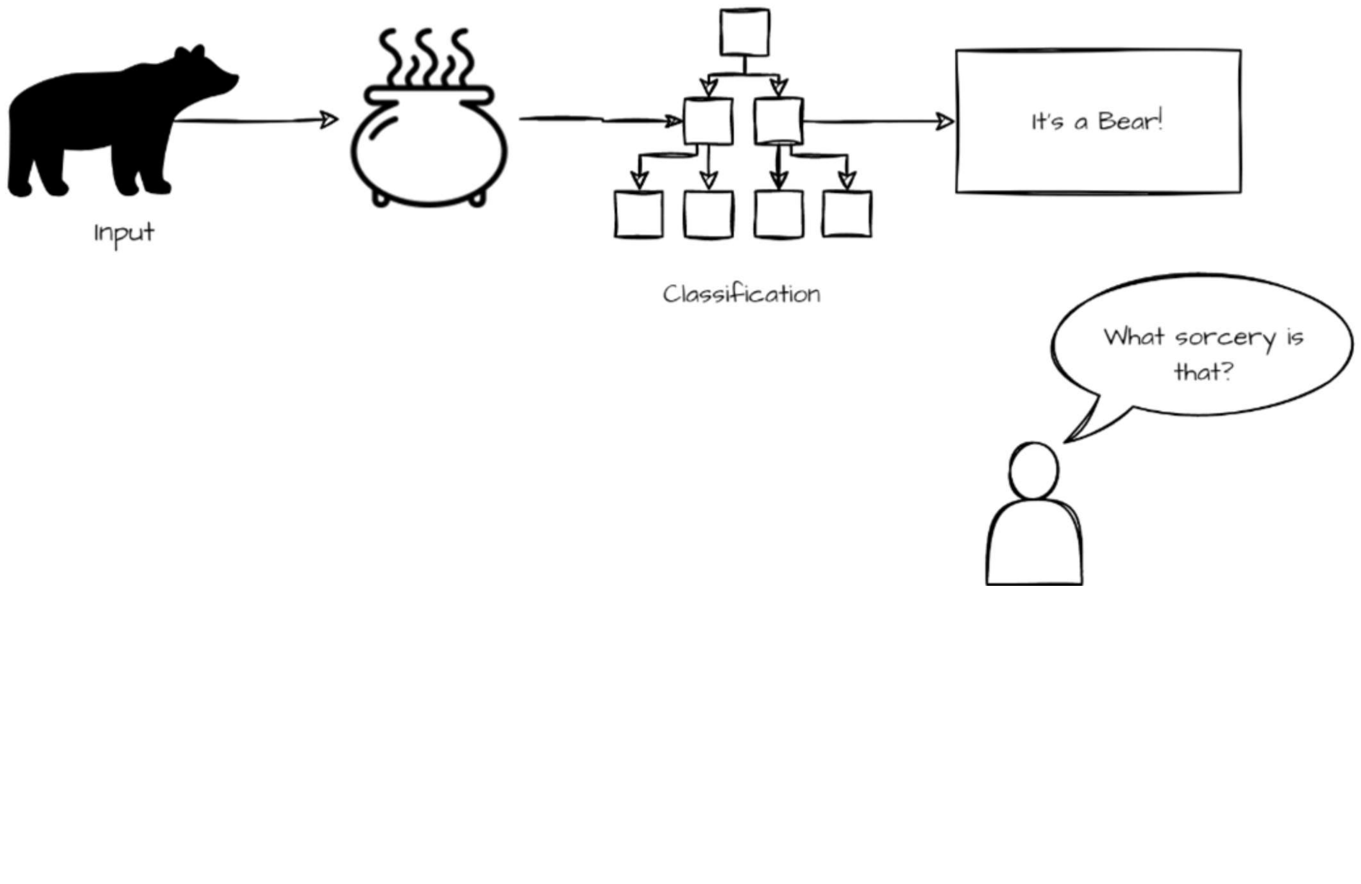
Deep learning is a combination of computing power and neural networks to learn complicated patterns in the data. The state-of-the-art deep learning techniques identify objects in images and words in sounds. The research in towards the creation of more general applications and the performance of highly optimized tasks. Task such as automatic language translation, medical diagnoses and social and business solutions are the most advanced for pattern recognition with deep learning.

In essence, the main difference for machine and deep learning is shown in Figure 1.4, where the feature extraction is done by the algorithm and not by humas in the Deep Learning making the result uninterpretable. Deep learning methods are frequently seen as a black box, with most empirical rather than theoretical confirmations [11].

MACHINE LEARNING

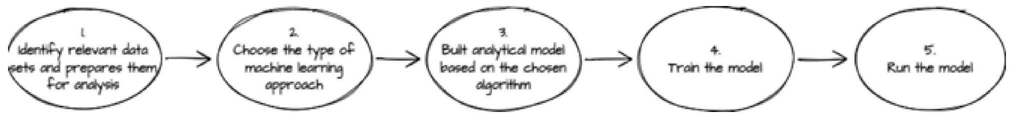

If it's fluffy, big and scary **then** it might be a Bear
Input
Classification
It's a Bear!

DEEP LEARNING



3. How it’s made

There are several approaches in Machine Learning field, but **how do we build them?** The techniques we explained earlier may be far apart, however the main building approach is the same. To create a machine learning system, the engineers should first prepare the system for the data acquisition and the data retrieval. Then, the most suitable algorithms, both basic and advanced are chosen and automated and iterative processes are implemented for the algorithms. The system is scaled for accepting more data and finally a complete model is created.


1. Identify relevant data sets and prepares them for analysis
2. Choose the type of machine learning approach
3. Built analytical model based on the chosen algorithm
4. Train the model
5. Run the model

The machine learning process follows the requirements described above and can be summarized in five steps as shown in Figure 1.5:

1. **Identification** of the relevant datasets and preparation for the analysis.
2. Selection of the type of the **machine learning approach** to use.
3. **Build of the analytical model** based on the chosen algorithm.
4. **Model training** on the data sets, revising as needed.
5. **Execution of the model** to generate scores, decisions, results.

4. Why insist on Machine Learning though?

As previously said, articulating the knowledge of the problem in a conventional language is extremely difficult to impossible to tackle big problems with many data and complex interactions between data. However, such issues have a **significant influence on society** and are of **tremendous scientific and commercial interest**. As a result, solving them with machine learning techniques seemed an obvious choice.

The most common examples of the problems are:

· Use of historical data for decision making processes, e.g., patience's' medical records are used for medical knowledge and reports.

· Complex computer programs, such as: robots that learn their environment and can wonder in it, self-learning games etc.

· Learning of human behavior and intelligence: speech patterns and face recognition.

· Development of programs adaptable to the user, e.g., speech recognition and written patterns recognition.

· Creation of real Artificial Intelligence: if a smart system, which is genius created cannot learn for its mistakes, it is not smarter than a worm or a cat.

5. Examples of machine learning use

One of the most common applications of machine learning in day-to-day use is Facebook's recommendation engine for the news feed. Facebook uses machine learning to personalize the delivered feed. When a member frequently reads a particular group's post, the recommendation engine shows more of the group's activity higher in the feed. The engine attempts to discover the member's pattern of online behavior and reinforce the algorithm. When a member stop reading the group and changes patterns the recommendation engine adapts and after a while there are new relevant suggestions in the feed.

Other applications that heavily based on machine learning these days are:

- **Customer Relationship Management**. CRM software use machine learning models and analyze emails. They also support the sales teams to respond to the most important messages first. The most advanced systems can recommend possible effective responses.
- **Business Intelligence.** BI and analytics companies use machine learning in the software to recognize important data, patterns and anomalies in the data.
- **Human Resource Information Systems.** HRIS use machine learning models to decide the best candidates for open positions.
- **Self – driving cars.** They are dependent on the machine learning algorithms to recognize the visible object and alert the driver. This leads to semi – autonomous cars.
- **Virtual assistants.** They combine supervised and unsupervised machine learning models to interpret the human natural speech and apply context to it and the correctly reply.
- **Financial services**. Banks use machine learning to identify knowledge of the data and to prevent fraud. The knowledge also leads to investment opportunities and personalized suggestions to clients.
- **Government**. Several government organizations use machine learning and analyze sensor data from traffic cameras, CCTV (Closed Circuit TV) and other sources for efficiency and accuracy. Governments also analyze census data for health and social security services and economic reliability.

6. Machine Learning advantages and disadvantages

The advantages and disadvantages of the machine learning application led us to the most suitable implementations and where machine learning techniques provide the best results, because machine learning is a very powerful tool that can optimize the solutions in data science.

6.1. Advantages

The **main advantages** of machine learning are:

1. **Easy identification of trends and patterns**

Machine learning looks at large amounts of data and finds trends and patterns that are not apparent to humans. The best example comes from the e-commerce websites that can detect and understand the browsing behaviors of the users and the purchase history and then suggest the right products for them or the relevant products with their behavior. The techniques are also used for advertising similar products to the users.

2. **Automation – no human intervention**

A machine learning application is independent from the developer. The code is minimal and gives the ability to the machines to learn. ML lets the machines make predictions and improve the algorithm on their own. Antivirus software is such an example. They learn and filter the new threats as they recognize them. These ML applications are also able to recognize spams.

3. **Continuous improvement**

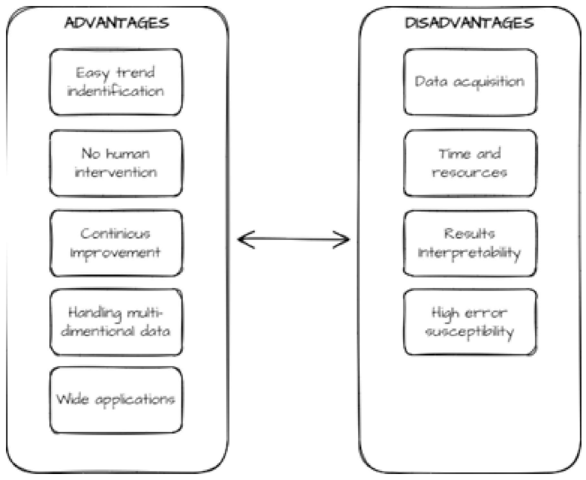
Machine learning algorithms become better as they gain experience. The longer they work, the more efficient and accurate they become, and this leads to better decisions. The weather forecast models work this way: the amount of data increases over time and the machine learning algorithms makes more accurate predictions faster.

4. **Handling of multi – dimensional and multi – variety of data**

The machine learning algorithms are good at handling data with multiple dimensions and variety in dynamic or uncertain environments.

5. **A wide variety of applications**

Machine learning can work either for e-commerce or healthcare or even manufacturing. It holds the capability to deliver personalized experience to the users for each application.



6.2. Disadvantages

The **main disadvantages** of the machine learning applications are:

1. **Data acquisition**

The machine learning algorithms require massive data sets to train on and the datasets should be unbiased, to include all kinds of the available data and be of high quality. Also, most of the applications need to wait for the datasets to be created.

2. **The time and the resources needed**

Machine learning needs enough time so the algorithms can learn from the datasets and be developed enough to provide accurate and relevant results. Furthermore, machine learning algorithms require large computational power from the systems.

3. **Interpretation of the results**

One of the major challenges of the machine learning applications are the ability to interpret the generated results with accuracy. For this, the responsive algorithms should be carefully chosen.

4. **Highly susceptible to errors**

The machine learning algorithms are highly susceptible to errors. When an algorithm is trained with small datasets that are not inclusive the predictions are biased coming from a biased training dataset. For example, the irrelevant advertisements displayed to customers. In the machine learning applications, these errors can be undetected for long periods. When such an error is recognized, it also takes some time to identify the sources of the issue and then correct it.

6.3. Advantages applications

The above advantages have been applied to many aspects of entrepreneurship. The following is a list of some of these applications.

1. **Simplification of Product Marketing and leads to Precise Sales Forecasts**

ML assists businesses in a variety of ways, including improved product promotion and more accurate sales forecasting. ML has several benefits for the sales and marketing industry, the most notable of which are:

- **Massive Data Consumption from a diverse range of sources.**

Due to large amount of data consumed in ML, based on the consumer behavioral patterns, these collected data may be utilized to regularly assess and adapt your sales and marketing efforts.

- **Rapid Evaluation Prevision and Processing**

Because of the speed with which machine learning consumes data and discovers important facts, you can take suitable decisions at the proper time. The customer will receive the best possible offer at any given time, as machine learning will help optimize this offer, without you having to spend time planning and making the best ad accessible to your customers.

- **Analyze Previous Customer Behaviors**

ML will allow you to study and evaluate data relating to previous behaviors or results. As a result, you will be able to create better predictions of client behavior based on the new and varied data.

2. **Allows for more accurate medical diagnoses and predictions**

Another important branch ML is used is healthcare industry, as helps identifying high-risk patients make near-perfect diagnoses, recommend the best possible medicines, and predict readmissions. These are largely based on publicly available datasets of anonymized patient records as well as the symptoms they present. Faster patient recovery will be facilitated by near-perfect diagnoses and improved pharmaceutical prescriptions, which will eliminate the need for unnecessary medications. In this approach, machine learning allows the medical industry to enhance patient health at a low cost.

3. **Time-Intensive Documentation in Data Entry is simplified**

It is well-known that all organizations prefer to automate their data entry process but data duplication and inaccuracy are obstacles and need be addressed. Predictive modeling and machine learning techniques, on the other hand, can greatly improve this condition. Machines can now do time-consuming data entry chores, freeing up your experienced personnel to focus on other value-added work.

4. **Enhances the accuracy of financial rules and models**

Machine Learning has a huge influence on the financial industry as well. Portfolio management, algorithmic trading, loan underwriting, and, most critically, fraud detection is some of the frequent machine learning benefits in finance. Furthermore, according to an Ernst & Young paper titled "The Future of Underwriting," machine learning allows for continuous data evaluations for detecting and assessing abnormalities and subtleties. This aids in the refinement of financial models and rules.

5. **Strong filter spam**

One of the first challenges that machine learning handled was spam detection. To filter out spam, email providers used rule-based systems a few years ago. However, with the advent of machine learning, spam filters are creating new rules to delete spam letters using brain-like neural networks. The neural networks evaluate the rules over a large network of computers to identify phishing mails and junk mail.

6. **Boosts the Manufacturing Industry's Predictive Maintenance efficiency**

Corrective and preventative maintenance procedures are in place in manufacturing companies. However, these are frequently ineffective and pricey. This is where machine learning may be quite useful. Machine learning assists in the development of highly effective predictive maintenance programs. Following such predictive maintenance programs reduces the likelihood of unexpected breakdowns, lowering the need for wasteful preventative maintenance.

7. **Improved Customer Segmentation and Prediction of Lifetime Value**

Marketers now confront enormous issues such as customer segmentation and predicting lifetime value. Sales and marketing teams will have access to massive volumes of relevant data gleaned from a variety of sources, including lead data, website visits, and email campaigns. With machine learning, however, accurate forecasts for incentives and specific marketing offers are simple to produce. Marketers that are well-versed in machine learning are increasingly using it to reduce the guesswork that comes with data-driven marketing. For example, analyzing data reflecting a specific group of users' behavioral patterns during a trial period might assist organizations in estimating the likelihood of conversion to a paid version. Customer interventions are triggered by such a model to better engage customers in the trial and encourage them to convert early.

8. **Recommendation of the Right Product**

Any sales and marketing plan, including upselling and cross-selling, should include product recommendations. ML models will look at a customer's purchasing history and, based on that, identify the things in your inventory that the consumer is interested in. The program will look for hidden patterns in the goods and group similar things together into clusters. Unsupervised learning is a form of machine learning algorithm that does this. Businesses will be able to provide better product suggestions to their clients as a result of this strategy, which will encourage product purchase. Unsupervised learning aids in the development of a superior product-based recommendation system in this way [14].

7. Machine learning for manufacturing applications

The manufacturing applications that are based on machine learning are the following:

1. **Predictive Maintenance**

ML plays a significant role in the field of maintenance as with predictive maintenance, equipment failures are predicted before they occur, so timely maintenance is scheduled, and downtime is minimized. It has been observed that it is preferable to allocate resources for planned maintenance than fixing breakdowns in order to save valuable time. Machine learning algorithms have a 92 percent accuracy rate in predicting equipment failure, helping organizations to better plan their maintenance plans and improve asset dependability and product quality. According to studies, applying machine learning and predictive analytics raised total equipment efficiency from 65 percent to 85 percent.

2. **Quality Assurance**

Machine learning is an important tool in product inspection and quality control. The historical data provide very useful information to ML-based computer vision algorithms to categorize products according to the quality (acceptable products or defective product). So, the procedure of inspection and monitoring is becoming automated. In that case, it is not required making a library of possible defects, as the only thing that it is necessary is good samples. However, an algorithm that compares samples to the most prevalent types of flaws may be created. Using ML offers important savings in visual quality control in manufacturing. An interesting Forbes's survey shows ML-based automated quality testing may improve detection rates by up to 90%.

3. **Logistics Management of inventory**

Logistics has now become a vital part in all sectors of industry. Especially, in the manufacturing industry to successfully execute all production processes. Once again, ML plays a crucial role as provides automating several logistics-related tasks, increasing efficiency and lowering expenses. Manual, time-consuming procedures such as logistics and production-related documentation are expected to cost the average US firm $171,340 per year. It is obvious that ML can automate these regular processes, saving thousands of man-hours each year.

4. **Product Development**

One of the most popular applications of machine learning is product development. To achieve the greatest outcomes, both new product design and existing product improvement require substantial data analysis. Machine learning technologies may assist in the collection and analysis of vast amounts of product data to:

- Recognize consumers demands and desires,
- find vulnerabilities that have been buried,
- and discover new business chances

This can lead to advance existing product designs and develop new items that will provide new income streams for the business. Moreover, enterprises can minimize risks involved with developing new goods by making better-informed decisions based on improved data.

5. **Cybersecurity**

To function properly, machine learning solutions rely on networks, data, and technological platforms — both on-premises and in the cloud. The security of these data and systems is crucial, and machine learning can help by restricting access to important digital platforms and data. Individual users' access to sensitive data, the apps they use, and how they connect to it may all be streamlined with machine learning. This can help businesses secure their digital assets by immediately recognizing irregularities and taking appropriate action.

6. **Robotics**

Human workers still play a significant role in modern production. However, factory automation is increasing since robots can already undertake many difficult activities, with the exception of a few areas that require extremely high accuracy, which can only be provided by human professionals. In the future, robots that are flexible enough to operate alongside humans might take over a significant portion of production. They will be able to function in a variety of challenging and dynamic conditions with little human intervention. Advanced machine learning techniques may be used to assist firms in developing complicated strategies and production processes thanks to robotics [15] .

8. Global market examples

Machine learning is having a game-changing influence in manufacturing. Danone Group, a French food business, employs machine learning to increase the accuracy of its demand forecasting. This has resulted in a:

- Forecasting mistakes are reduced by 20%.
- 30 percent reduction in missed revenues
- Demand planners' workload is reduced by 50%.

Fanuc, a Japanese automation business, employs robots to run its operations around the clock. The robots can manufacture critical components for CNCs and motors, run all production floor gear continuously, and provide continuous monitoring of all processes [16].

Meanwhile, the BMW Group employs computerized image recognition for quality control, inspections, and the elimination of false problems (deviations from target despite no actual faults). As a result, they've attained great levels of production accuracy [17].

Porsche is another firm that has profited from ML in production. They employ autonomous guided vehicles (AGVs) to automate large aspects of the car production process. The AGVs transport car body components from one processing station to the next, removing the need for human interaction and making the plant more resistant to disturbances such as pandemics [18].

These are just a few instances of organizations using artificial intelligence in manufacturing to boost overall production and operational efficiency.

What is supervised Learning?

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 3:59 PM
Book:	What is supervised Learning?		

Table of contents

1. Supervised Learning Definition

1. Supervised Learning Definition

Supervised learning is one of the main categories of Machine Learning.

> As the name indicates, there is a supervisor present as a tutor. In reality, supervised learning is when we use well-labeled data to instruct or train a machine. This indicates that the correct answer has already been assigned to some data. After that, the computer is given a fresh collection of examples (data) to analyze the training data (set of training instances) and provide an appropriate result from labeled data using the supervised learning approach [16].

Consider the following scenario: you are handed a basket containing several fruits. The first stage now is to teach the machine all of the different fruits one by one, as follows:

· The object will be labeled as –**Apple** if it has a spherical form with a dip at the top and is red in color.

· The item will be labeled as –**Banana** if its form is a long curving cylinder with a Green-Yellow tint.

Now imagine that once you've trained the data, you've given a new independent fruit, like a banana from the basket, and requested it to be identified.

Because the system has already learnt from previous data, it is necessary to apply it intelligently this time. It will categorize the fruit based on its shape and color, then validate its name as BANANA and place it in the Banana category. As a result, the machine learns from training data (a fruit basket) and then applies what it has learned to test data (new fruit).

The above task was a classification problem that can be solved via supervised learning techniques. There are two main types of problems for supervised learning:

- **Classification:** In classifications problems the output variable is a category. For instance, “Green” or “Yellow” or “spam” and “not spam”.
- **Regression:** In regression problems the output variable is a real value like “euros” or “length”.

"Labeled" data is dealt with or learned with in supervised learning. This signifies that some information has already been labeled with the right answer [17].

Mapping to real life Manufacturing Cases

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 3:59 PM
Book:	Mapping to real life Manufacturing Cases		

Table of contents

1. Industrial Examples

1. Industrial Examples

Now let's examine more industrial examples of Supervised Learning techniques.

Energy efficiency is a critical problem in developing sustainable society. Global primary energy consumption is predicted to rise by 1.6 percent each year due to rising populations, rising incomes, and the industrialisation of developing nations. This scenario highlights concerns about the rising scarcity of natural resources, environmental degradation, and the imminent threat of global climate change [18].

Understanding energy demands at reasonably high geographical and temporal precision is crucial for improving the efficiency of supply systems and thereby reducing energy usage. A **precise estimate of energy** demand might give helpful information for energy generation and buying decisions. Furthermore, a precise **prognosis** would help to minimize overloading and enable for effective energy storage.

Types of supervised learning

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:00 PM
Book:	Types of supervised learning		

Description

The supervised learning involves many categories. However, the most common and used techniques are the following:

a) Classification and

b) Regression

are described in detail further below.

Table of contents

1. Classification
1.1. Relation to other problems
1.2. Classification tree analysis
1.3. Types of classification
1.4. The training
1.5. The classification
1.6. How does classification work?
1.7. Generalized classification block diagram
1.8. Types of classifiers
1.9. Decision Trees
1.10. Pros and Cons of Decision Trees
1.11. Real World Binary Classification use case

2. Regression
2.1. What is the purpose of regression analysis?
2.2. Types of regression
2.3. Real world Regression use case

1. Classification

> The problem of determining which of a collection of categories (sub-populations) an observation (or observations) belongs to is known as classification. Assigning a diagnosis to a patient based on observable features and categorizing a specific email as "spam" or "non-spam" are two examples (sex, blood pressure, presence or absence of certain symptoms, etc.).

Individual observations are sometimes broken down into a collection of measurable attributes referred to as explanatory variables or features. These attributes can be categorical (e.g. "A," "B," "AB," or "O" for blood type), ordinal (e.g. "big," "medium," or "small"), integer-valued (e.g. the number of instances of a specific word in an email), or real-valued (e.g. the number of occurrences of a particular word in an email) (e.g. a measurement of blood pressure). Other classifiers use a similarity or distance function to compare data to prior observations.

The technique of guessing the class of given data points is known as classification. Classes are sometimes known as goals, labels, or categories. The job of estimating a mapping function (f) from discrete input variables to discrete output variables is known as classification predictive modeling .

A **classifier** is an algorithm that accomplishes classification, particularly in a concrete implementation. The word "classifier" can also refer to the mathematical function that translates input data to a category and is implemented by a classification algorithm.

The terminology used in various disciplines is fairly diverse. The properties of observations are known as explanatory variables (or independent variables, regressors, etc.) in statistics, where classification is often done with logistic regression or a similar procedure, and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, instances are the observations, features are the explanatory factors (grouped into a feature vector), and classes are the potential categories to be predicted. Other areas may use alternative terms; for example, in community ecology, "classification" usually refers to cluster analysis.

1.1. Relation to other problems

Pattern recognition is the assignment of some form of output value to a given input value. Classification and clustering[1] are two instances of the more general issue of pattern recognition. Other examples include regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence); parsing, which assigns a parse tree to an input sentence, describing its syntactic structure; and so on.

[1] Clustering is further explained in the following sections.

1.2. Classification tree analysis

A classification chart, often known as **a classification tree analysis (CTA)**, is a diagram that shows the structure of a categorization scheme.

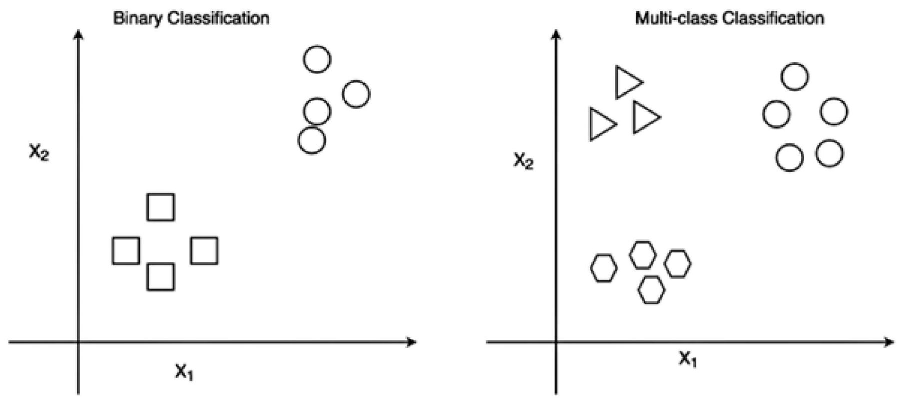
Classification is the act of recognizing, differentiating, and comprehending concepts and things, and classification charts are designed to aid in the creation and visualization of the end result. "In a classification chart, the facts, data, and so on are organized so that the place of each in relation to the others is plainly seen," Brinton says. Although a quantitative analysis adds to the usefulness of a classification chart, it is not required." [19]. "In any chart-making, the material to be displayed must be carefully collated before it can be plotted," Karsten noted. We must dig into the complexities of categorization and indexing in order to comprehend the classification chart.

1.3. Types of classification

Classification has to main types:

- **Binary Classification:** When we must categorize given data into 2 distinct classes. Example – On the basis of given health conditions of a person, we must determine whether the person has a certain disease or not.

- **Multiclass Classification:** The number of classes is more than 2. For Example – On the basis of data about different species of flowers, we must determine which specie does our observation belongs to.



1.4. The training

To create a classification tree, the user needs first utilize the training samples. This is referred to as the training phase. This tree is then used to classify the entire picture.

To begin, the root is allocated to all of the training pixels from all of the classes. Because the base of the tree contains all training pixels from all classes, an iterative procedure is started to expand the tree and divide the classes. CTA uses a binary tree structure in Terrset, which means that the root, as well as all following branches, may only develop two additional internodes before splitting or turning into a leaf. The binary splitting rule is defined as a threshold in one of the many input pictures that isolates the most homogeneous subset of training pixels from the rest of the training data.

The tree evolves by recursively dividing data into new internodes containing increasingly homogenous sets of training pixels at each internode. When a freshly developed internode includes exclusively training pixels from one class, or when pixels from one class dominate the population of pixels in that internode, and the dominance is at an acceptable level set by the user, the internode may become a leaf. The final classification tree rules are generated when there are no more internodes to divide [20].

1.5. The classification

Image classification is the second step in the CTA process. Using the decision rules of the previously trained classification tree, each pixel is labeled with a class in this stage. A pixel is initially supplied into the tree's root, where its value is compared to what is already in the tree, and the pixel is then transferred to an internode based on where it sits in respect to the splitting point. The procedure continues until the pixel reaches a leaf, at which point it is assigned a class.

The classification rules from the root to the leaf are straightforward to learn and comprehend, thanks to an intuitive graphical depiction in the interface. Numerical pictures, such as reflectance values from remotely sensed data, categorical images, such as a land use layer, or a combination of the two can be used as input images.

If you know that a data set follows a specific distribution pattern, you should use a good parametric classifier instead of the classification tree technique. If the picture data is known to follow a Gaussian distribution, a parametric classifier like MAXLIKE in TerrSet may be preferable [20].

1.6. How does classification work?

Assume we need to determine whether a patient has a specific ailment based on three characteristics known as features. There are two conceivable consequences as a result of this [21]:

1. The patient is suffering from the ailment in question. In other words, a "Yes" or "True" outcome.

2. The patient is free of sickness. "No" or "False" as a response.

This is a problem of binary classification. We have a training data set of observations, which consists of sample data with real classification results. On this data set, we train a model called Classifier, which we then use to predict whether a certain patient would get the disease or not. As a result, the outcome now depends on:

1. How effectively these characteristics "map" to the final result.

2. The accuracy of our data. We are referring to statistical and mathematical attributes when we say quality.

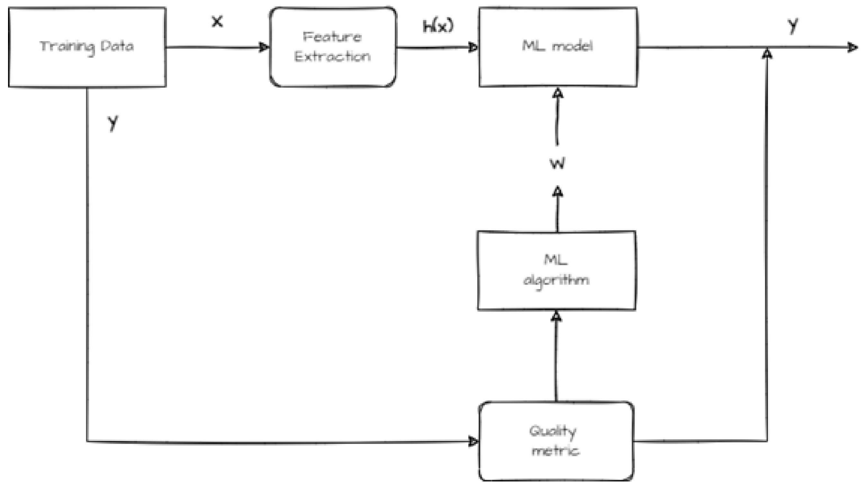
3. The degree to which our Classifier can generalize the link between the features and the output.

4. The values of the x1 and x2.

1.7. Generalized classification block diagram

The following elements are included in the generalized classification block diagram [22]:

1. X: pre-classified data, in the form of a N*M matrix. N is the number of observations and M is the number of features.
2. y: An N-d vector of anticipated classes for each of the N observations.
3. Feature Extraction: Using a sequence of transformations, extracting meaningful information from input X.
4. Machine Learning Model: We'll train the "Classifier".
5. y': The Classifier's predicted labels.
6. Quality Metric: A metric for assessing the model's performance.
7. Machine Learning Algorithm: The technique for updating weights w', which iteratively updates the model and "learns".



1.8. Types of classifiers

There are various types of classifiers. Some of them are:

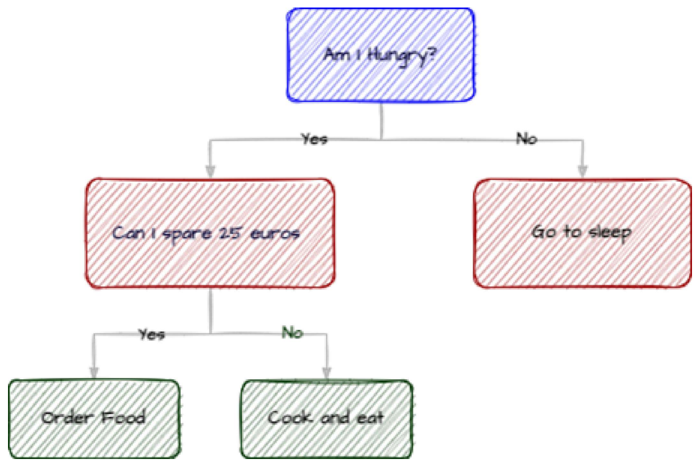
- Linear Classifiers: Logistic Regression.
- Tree-Based Classifiers: **Decision Tree Classifier**.
- Support Vector Machines.
- Artificial Neural Networks.
- Bayesian Regression.
- Gaussian Naive Bayes Classifiers.
- Stochastic Gradient Descent (SGD) Classifier.
- Ensemble Methods: Random Forests, AdaBoost, Bagging Classifier, Voting Classifier, ExtraTrees Classifier.

1.9. Decision Trees

A decision tree, in its most basic form, is a graphical depiction of all possible solutions to a problem. In today's supervised learning settings, tree-based algorithms are the most commonly used algorithms. They're easy to understand and envision, and they're quite adaptable. Tree-based algorithms may be utilized for both regression and classification issues, however they are more commonly employed for classification problems [23].

Let's look at an example of a decision tree: I didn't eat supper at my normal hour last evening since I was preoccupied with other matters. I had butterflies in my tummy later that night. I figured that if I wasn't hungry, I could have just gone to sleep, but since I wasn't, I opted to eat something. I had two choices: order anything from outside or prepare things yourself.

I thought that if I ordered, I'd have to set aside at least INR 250. I decided to order it anyhow because it was late and I was not in the mood to cook. As seen in the diagram below, the entire episode may be visually depicted.



The following are the terminology used to describe a decision tree:

- **Parent node:** A parent node is the higher hierarchically connected node of any two connected nodes.
- **Child node:** A kid node is the lower hierarchically connected node in any two connected nodes.
- **Root node:** The root node is the node from which the tree grows, and it has no child nodes. There is no parent node for the root node. (in the figure above, the dark blue node).
- **Leaf Node/leaf:** Leaf nodes, or simply leaf, are nodes at the end of the tree that do not have any children. (in the picture above, green nodes).
- **Internal nodes/nodes:** Internal nodes, or simply nodes, are found between the root node and the leaf nodes. Internal nodes have at least one child and a parent. (In the figure above, the red nodes).
- **Splitting:** Adding two or more children to a node or dividing a node into two or more sun-nodes.
- **Decision node:** When a parent node splits into two or more offspring nodes, the node is referred to as a decision node.
- **Pruning:** Pruning is the process of removing a decision node's sub-node. You might think of it as the total opposite of splitting.
- **Branch/Sub-**tree: A branch, also known as a sub-tree, is a portion of the whole tree.

1.10. Pros and Cons of Decision Trees

Pros of a decision tree:

- **Easy to see and interpret:** Its graphical depiction is highly straightforward to understand, and it can be interpreted without any prior understanding of statistics.

- **Helpful in data investigation:** With a decision tree, we may quickly find the most important variable and the relationship between variables. It can assist us in the creation of new variables or the grouping of certain features into a single bucket.

- **Data cleansing isn't as necessary:** Because it is somewhat resilient to outliers and missing data, it requires minimal data cleansing.

- **The data type is not a constraint**: It is capable of dealing with both category and numerical data [25].

Cons of decision tree:

- **Overfitting:** A single decision tree has a tendency to overfit the data, which may be mitigated by imposing limitations on model parameters such as tree height and pruning (which we will discuss in detail later in this article)

- **For continuous data, the fit isn't perfect:** When numerical variables are classified into distinct groups, part of the information associated with them is lost.

1.11. Real World Binary Classification use case

In the semiconductor processing business and more specifically in the silicon wafer manufacturing sector, dry pumps are utilized to seal the wafers in vacuum and be protected from dust. The light-weight dry pumps that are used in that production phase, pass some quality stress tests after their production. For each pump that fails there is a log record indicating the failure mode and the duration that the pump was working before it fails. The factory that produces the pumps, wants to know whether a newly produced pump is going to fail in less than a year of continuous operation. To handle this problem a binary classification problem can be formed, where the quality test results can be used as features and the failures log can be utilized to label each feature vector (fail in less than a year or not).

2. Regression

The second major type of supervised learning is the regression.

> Regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often referred to as the 'outcome' or'response' variable, or a 'label' in machine learning terminology) and one or more independent variables (often referred to as 'predictors,''covariates,'explanatory variables,'or'features').

This approach facilitates in the detection of variable correlations and allows the continuous output variable to be predicted using one or more predictor variables. Common uses include prediction, forecasting, time series modeling, and identifying the causal-effect link between variables. Regression analysis is a set of statistical methods for evaluating relationships between a dependent variable and one or more independent variables. It may be used to assess the strength of a relationship between variables as well as forecast how they will interact in the future.

Linear regression is the most frequent type of regression analysis, in which one finds the line (or a more sophisticated linear combination) that best fits the data according to a set of mathematical criteria. Ordinary least squares, for example, finds the single line (or hyperplane) that minimizes the sum of squared differences between the genuine data and that line (or hyperplane).

Regression analysis is generally applied for two reasons that are conceptually separate.

- For starters, regression analysis is extensively used for **prediction and forecasting**, and it shares a lot of ground with machine learning.

- Second, regression analysis may be used to identify **causal links between independent and dependent variables** in particular scenarios. Importantly, regressions reveal correlations between a dependent variable and a group of independent variables in a given dataset by themselves. A researcher must carefully demonstrate why existing correlations have predictive power for a new context or why a link between two variables has a causal interpretation before using regressions for prediction or inferring causal relationships. When utilizing observational data to determine causal links, the latter is very crucial.

2.1. What is the purpose of regression analysis?

As previously stated, the use of regression analysis in the prediction of a continuous variable is beneficial. In the real world, there are a variety of scenarios where we need to make future predictions, such as weather conditions, sales predictions, marketing trends, and so on.

The following are some of the reasons why regression analysis is used:

- Through regression, an estimate of the relationship between the target and the independent variable is provided.
- It is very effective in finding trends in data.
- It aids in the prediction of real and continuous variables.
- By using the regression, we can clearly determine the most essential and least significant aspects, as well as how each one affects the others.

2.2. Types of regression

Regression analysis comes in a variety of forms [26]. The following are most important:

1. **Linear regression**

Linear regression is one of the most fundamental forms of regression in machine learning. It consists of a predictor variable and a dependent variable that are linearly connected to each other. As previously said, linear regression uses the usage of a best fit line. When your variables are linearly connected, you should apply linear regression. If you're predicting the impact of increased advertising spend on sales, for example. However, because this method is prone to outliers, it should not be utilized to evaluate large data sets.
It's way to implement using Python to create linear regression. All you have to do now is use the appropriate packages, functions, and classes. NumPy is a basic Python scientific library that allows you to execute a variety of high-performance operations on single- and multi-dimensional arrays. It also includes a number of mathematical functions. It is, after all, open source.

If you're new to NumPy, start with the official NumPy User Guide and read Look Ma, No For-Loops: NumPy Array Programming. Furthermore, the performance benefits you may receive while using NumPy can be shown in the Pure Python vs NumPy vs TensorFlow Performance Comparison.

Scikit-learn is a commonly used Python machine learning toolkit that is built on top of NumPy and a few additional libraries. It includes tools for data preparation, dimensionality reduction, regression, classification, clustering, and more. Scikit-learn (https://scikit-learn.org/stable/), like NumPy (https://numpy.org/), is free and open source.

To understand more about linear models and how this package works, go to the scikit-learn website and look at the article Generalized Linear Models.

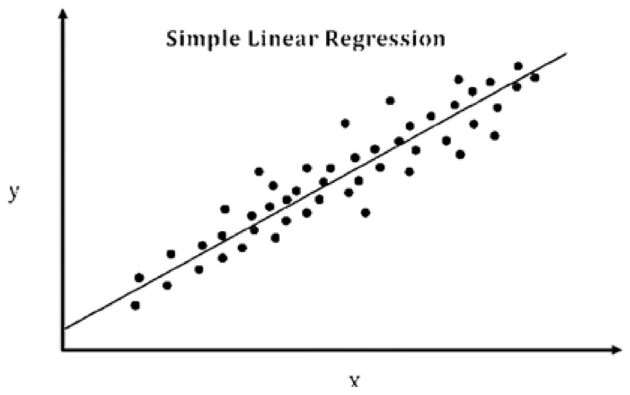
If you want to use linear regression but need something more than scikit-learn can provide, statsmodels is a good option. It's a robust Python module that can be used to estimate statistical models, run tests, and more. It's also free and open source.

A model that examines the connection between a dependent variable and an independent variable is known as simple linear regression. The following equation expresses the simple linear model:

$$Y = a + bX + e$$

Where:

- $Y: the\ dependent\ variable$
- $X: the\ independent\ (explanatory)\ variable$
- $a: the\ intercept$
- $b: the\ slope$
- $e: the\ residual\ (error)$



2. **Assumptions for the Linear Regression**

Linear regression is a widely used prediction approach for defining a linear connection between independent and dependent variables. When learning about predictive algorithms, regression analysis is typically one of the first things you learn. It is still a potent approach frequently used in statistics and data science, as easy as it appears (after you have used it enough). In this post, we'll go over the assumptions you'll need to examine in order to use linear regression appropriately [27].

There are five important assumptions in linear regression analysis. These are the following:

- Multicollinearity is little or non-existent.
- We're looking into a linear relationship.
- Autocorrelation is minimal to non-existent.
- Data is homoscedastic
- The distribution of all variables is normal.

A Linear Relationship is being investigated.

When conducting a linear regression model, we want to identify a linear connection between the independent and dependent variables, as the name suggests. Scatter plots are an easy visual technique to determine this.

The Distribution of Variables Is Normal

The variables have a normal distribution, which is the following assumption. That is to say, we want to make sure that y is a random variable with a normal distribution and that its mean is on the regression line for each x value.

The Q-Q-Plot is one method for visually testing this assumption. The Quantile-Quantile plot (Q-Q) is a tool for visually comparing two probability distributions.

To make this Q-Q plot, we'll use scipy's probplot function, which compares a given variable to a normal probability.

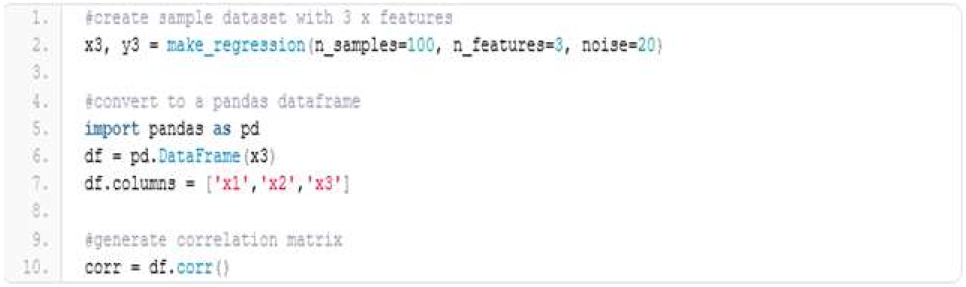
What is the best way to tell if your variable has a normal distribution? Is there a red line in the graph above? To infer that it follows a normal distribution, the points must lie on this line. Yes, it does in our situation! Because of our tiny sample size, a few points are outside the line. Because it is a visual test, you may determine how severe you want to be in practice.

There is Little or no Mutlicollinearity

The term "multicollinearity" refers to the fact that your independent variables are significantly associated with one another. Keep in mind what your Xs are named; they're called independent variables for a reason. If they have multicollinearity, they are no longer independent, which causes problems when modeling linear regressions.

We may leverage the power of Pandas and their styling options (in development) to visually test for multicollinearity. These options allow us to decorate data frames according to the data within them.

Let's start by creating a regression dataset in the same way we did in the last example, but this time with three X variables. We next turn this array into a Pandas data frame and compute the pairwise correlation of our columns using the Pandas corr function.



```
#create sample dataset with 3 x features
x3, y3 = make_regression(n_samples=100, n_features=3, noise=20)

#convert to a pandas dataframe
import pandas as pd
df = pd.DataFrame(x3)
df.columns = ['x1','x2','x3']

#generate correlation matrix
corr = df.corr()
```

There is Little or no Autocorrelation

This assumption is similar to the one before it, except it applies to the residuals of your linear regression model. We won't dig deeper into this assumption until our next post, when we delve into running a linear regression model, because establishing a linear regression model is outside the scope of this article.

Homoscedasticity

The homoscedasticity assumption is the last assumption of linear regression; this analysis is also applied to the residuals of your linear regression model and can be readily evaluated with a scatterplot of the residuals.

When the noise in your model can be represented as random and consistent across all independent variables, you have homoscedasticity. If you see a trend in the scatterplot of residuals from your linear regression study, this is a strong indication that this assumption is being broken.

The 5 basic assumptions of linear regression were tested using Python. The first three are used before starting a regression analysis, while the final two (AutoCorrelation and Homoscedasticity) are used on the residual values once the regression study is finished.

3. Multiple Linear Regression

With the exception that numerous independent variables are utilized in the model, multiple linear regression analysis is substantially the same as simple linear regression analysis. Multiple linear regression is expressed mathematically as follows:

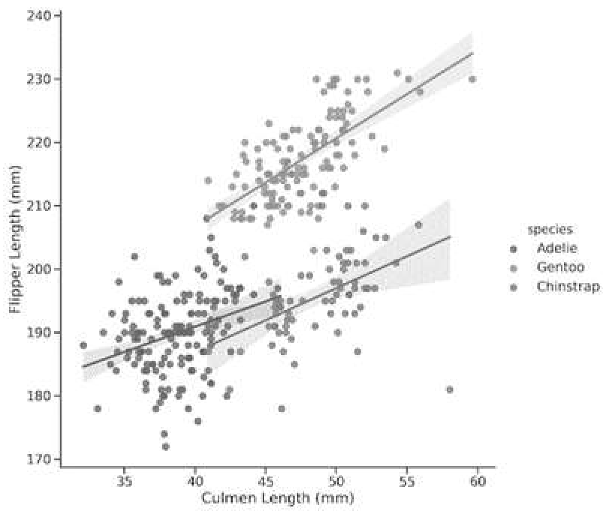
$$Y = a + bX_1 + cX_2 + dX_3 + e$$

Where:

- $Y: the\ dependent\ variable$
- $X_1, X_2, X_3 : the\ independent\ (explanatory)\ variable$
- $a: the\ intercept$
- $b, c, d: the\ slopes$
- $e: the\ residual\ (error)$

The requirements for multiple linear regression are the same as for the basic linear model. However, because multiple linear analysis involves numerous independent variables, there is additional need for the model:

● **Non-collinearity:** refers to the fact that independent variables should have a low correlation with one another. It will be difficult to analyze the genuine relationships between the dependent and independent variables if the independent variables are strongly linked.



Regression analysis is a very effective machine learning approach for data analysis. Here, we'll look at how it works, the many sorts, and what it can do for your company.

The use of regression analysis to predict future events between a dependent (goal) and one or more independent variables is known as predicting (also known as a predictor). It can, for example, be used to anticipate the link between reckless driving and the overall number of road accidents caused by a driver, or, in the case of a business, the influence on sales and the amount of money spent on advertising.

One of the most prevalent machine learning models is regression. It varies from classification models in that it estimates a numerical value rather than identifying the category to which an observation belongs. Forecasting, time series modeling, and determining the cause-and-effect connection between variables are some of the most common applications of regression analysis.

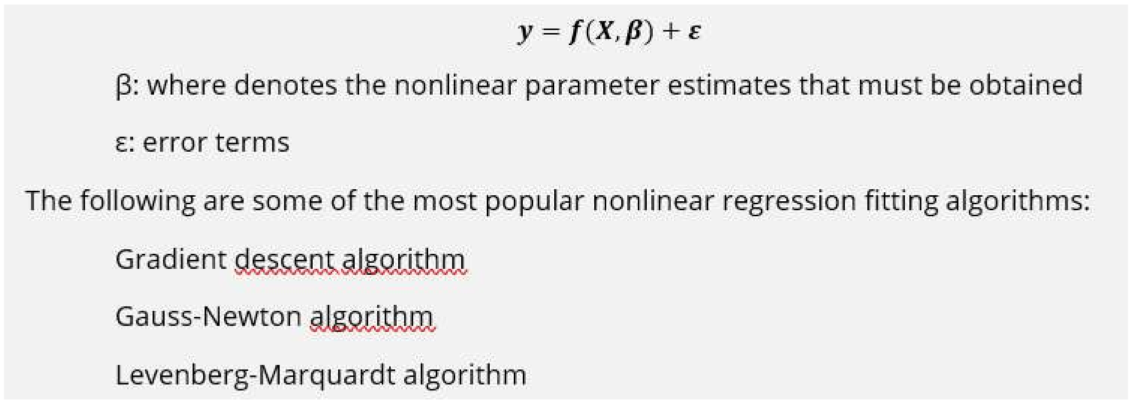
You may use the same methods for multiple linear regression as you would for simple regression.

4. Non – Linear Regression

The statistical approach of nonlinear regression is used to characterize nonlinear interactions in experimental data. When a nonlinear regression model is described as a nonlinear equation, it is assumed to be parametric. For non-parametric nonlinear regression, machine learning approaches are commonly utilized.

The dependent variable (also known as the response) is modeled as a function of a set of nonlinear parameters and one or more independent variables in parametric nonlinear regression (called predictors). The model might be univariate (with a single response variable) or multivariate (with several response variables) (multiple response variables).

An exponential, trigonometric, power, or any other nonlinear function can be used as a parameter. An iterative approach is commonly used to determine nonlinear parameter estimations.



$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\beta}) + \boldsymbol{\varepsilon}$$

β: where denotes the nonlinear parameter estimates that must be obtained

ε: error terms

The following are some of the most popular nonlinear regression fitting algorithms:

Gradient descent algorithm

Gauss-Newton algorithm

Levenberg-Marquardt algorithm

5. Polynomial regression

Polynomial regression uses a linear model to model a non-linear dataset. It's the same as trying to put a square peg into a round hole. It uses a non-linear curve and operates similarly to multiple linear regression (which is basically linear regression with several independent variables). It's used when there are a lot of non-linear data points.

The model converts these data points into polynomial features of a particular degree and uses a linear model to represent them. Instead of the straight line used in linear regression, a polynomial line, which is curved, is used to best suit them. However, because this model is prone to overfitting, you should scrutinize the curve near the end to avoid strange-looking outcomes.

Although there are more forms of regression analysis than those described here, these five are the most common. If you choose the appropriate one, your data will be able to reach its full potential, putting you on the road to greater insights.

2.3. Real world Regression use case

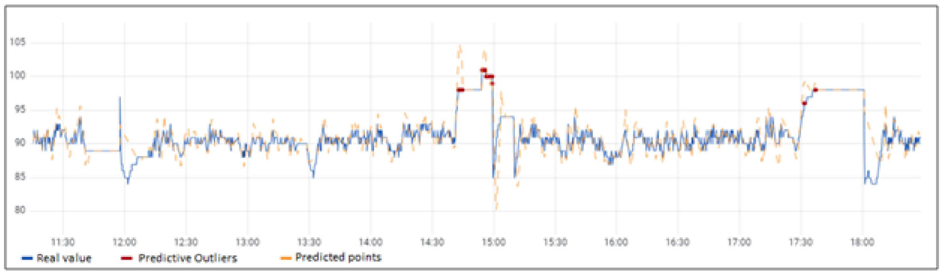
A company that has a fleet of vehicles the use of which is part of its smooth day-to-day operations would like to minimize the number of times its vehicles are out of service. Suppose are working as a Machine Learning Expert in this company and you have to build a model that predicts when a vehicle will experience malfunctions. You have numerous incoming measurements but you have discovered that the most valuable one is the temperature of the engine. For forecasting the future values, you can apply linear regression to the incoming measurements in order to calculate the trend of the line that the individual measurements create. If the line presents a highly ascending slope at a point, we can assume that there is an incident that is causing malfunction to the vehicle. Moving forward, we can assist the training of our model by feeding the real values that actually happen to the vehicle.

Use historical data to predict future values within a user-defined window.

Locating ectopic points in terms of:

- Slope
- Predicted Price/Current Price
- Estimated value

A point is considered an outlier when it is outside the limits set by the user.



K Nearest Neighbor (kNN)

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:00 PM
Book:	K Nearest Neighbor (kNN)		

Description

Now let's examine a famous algorithm that implements supervised learning techniques, the k-nearest neighbors algorithm or kNN.

Mathematicians have developed a variety of machine learning models that you may utilize. One of these is the k-Nearest Neighbors algorithm [28].

All of these models have their own characteristics. If you work in machine learning, you should be familiar with all of them so that you can apply the appropriate model to the appropriate circumstance. Next, you'll see how kNN stacks up against other machine learning models to see why and when to employ it.

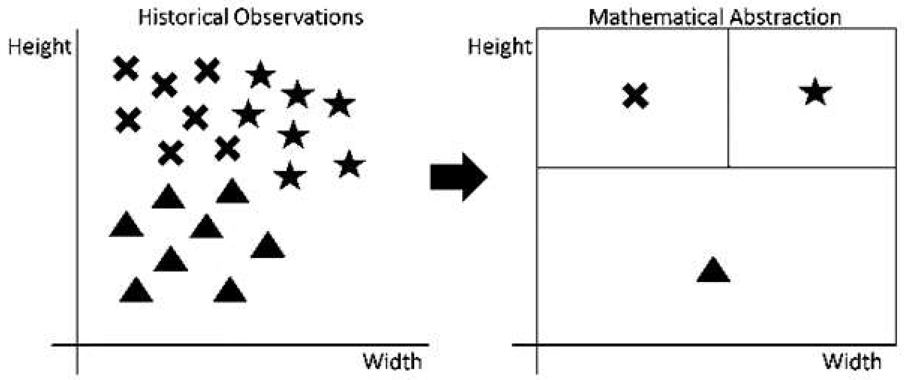
The kNN algorithm is a supervised machine learning method. You have two sorts of variables in supervised models at the same time:

A. The dependent variable y, commonly known as the y variable, is the target variable.

B. Independent variables, often known as x variables or explanatory variables

The goal is to predict the target variable. These variables are not known in advance and depend on the values of the independent variables. Obviously, the independent variables are defined. With solving an equation is easy to calculate or predict the target variable. In this sense, is similar to y = ax + b case.

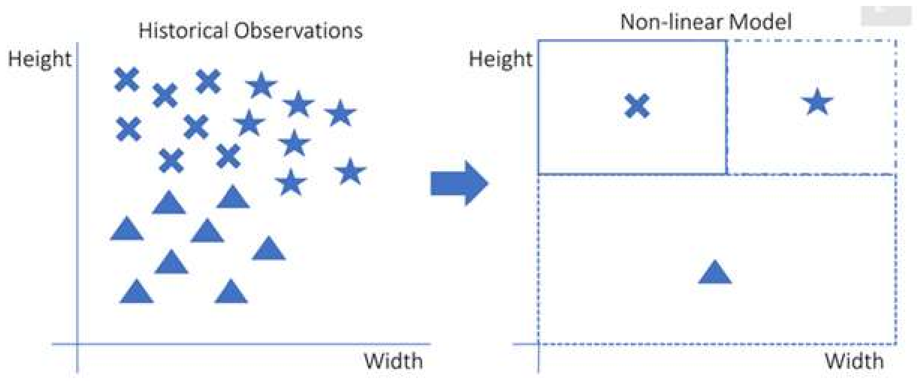
Looking the following graph, the target variable is the shape of the data point. The height and width are the independent variables.



So, what we see in this graph? Each data point has its own height, width, and shape. There are crosses, stars, and triangles the symbols. A decision rule learned by a machine learning model is shown on the right.

The observations highlighted with a cross in this example are tall but not wide. Stars are towering and broad at the same time. Triangles are not just short, but they may also be wide or narrow. Essentially, the model has learnt a decision rule that uses only the height and breadth of an observation to determine whether it is more likely to be a cross, a star, or a triangle.

The kNN technique is a supervised machine learning model. That is, it uses one or more independent variables to predict a target variable. A nonlinear learning algorithm is kNN. Nonlinear models are those that split their cases using a method other than a line. The decision tree, which is just a lengthy set of if... else statements, is a well-known example. If...else statements in the nonlinear graph would allow you to draw squares or any other shape you chose. A nonlinear model applied to the example data is seen in the graph below:



This graph demonstrates how a nonlinear decision may be made. Three squares make up the decision rule. A new data point's anticipated form is determined by the box in which it falls. It's worth noting that utilizing a line won't allow you to fit everything in at once: A total of two lines are required. This model might be recreated using the following if...else statements:

- It's a triangle if the data point's height is low.
- Otherwise, it's a cross if the data point's width is small.
- Otherwise, it's a star if none of the above is true.

As mentioned above, kNN is an example of a nonlinear model. For both classification and regression, kNN is a supervised learning.

Some models are only capable of regression, whereas others are only capable of classification. The kNN method adapts to both classification and regression in a smooth manner. In the following section of this lesson, you'll learn exactly how this works.

kNN can be characterized as interpretable and fast. Model complexity is a final factor to consider when describing machine learning models. Machine learning, particularly artificial intelligence, is booming right now, and it's being used for a variety of complex activities including analyzing text, photos, and voice, as well as self-driving automobiles.

Advanced and complex models, such as neural networks, are likely to be able to learn more than a k-Nearest Neighbors model. Those complex forms, after all, are excellent learners. However, take into account that this level of sophistication comes at a cost. You'll need to invest a lot more effort on development to make the models suit your predictions.

To fit a more complicated model, you'll also need a lot more data, which isn't always accessible. Finally, more complex models are more difficult for us humans to comprehend, which may be quite helpful at times.

It is easy to understand that the kNN model is particularly significant. As, it enables users to comprehend and analyze what is occurring within the model, and it is quite quick to construct. As a result, kNN is an excellent model for many machine learning applications that do not necessitate the employment of extremely complicated algorithms.

Table of contents

1. Pros and Cons of kNN

1. Pros and Cons of kNN

K-Nearest Neighbors, or K-NN for short, is a supervised machine learning technique that uses a labeled (Target Variable) dataset to predict the class of a new data point [30]. The K-NN technique is a reliable classifier that is frequently used as a benchmark for more complicated classifiers like Artificial Neural Networks (ANN) and Support Vector Machines (SVM) (SVM). A few of the reasons to use the K-NN machine learning algorithm are listed below:

o **K-NN is easy to use and understand:** The K-NN technique is straightforward to comprehend and implement. To categorize a new data point, the K-NN method searches the whole dataset for K nearest neighbors.

o **There are no assumptions in K-NN**: Because K-NN is a non-parametric method, some assumptions must be met in order to implement it. Data must meet several assumptions in parametric models like linear regression before they can be applied, however this is not the case with K-NN.

o **There is no Training Step:** K-NN does not create a model directly; instead, it tags new data entries based on past data. In the nearest neighbor, new data would be labeled with the majority class.

o **It always changing:** Because it is an instance-based learning method, k-NN is a memory-based method. As additional training data is collected, the classifier adjusts instantaneously. It enables the algorithm to react swiftly to changes in the input during real-time operation.

o **For a multi-class issue, it's quite simple to implement:** Most classifier algorithms are simple to implement for binary issues but require more work to implement for multi-class problems, but K-NN adapts to multi-class problems without any additional effort.

o **Both classification and regression may be done using it:** K-NN has the benefit of being able to be utilized for both classification and regression issues.

o **One Hyper Parameter:** K-NN may take a long to choose the initial hyper parameter, but once it is chosen, the rest of the parameters are aligned to it.

o **There are several distance criteria to choose from**: When generating a K-NN model, the K-NN method allows the user to pick the distance.

▪ Euclidean Distance

▪ Hamming Distance

▪ Manhattan Distance

▪ Minkowski Distance

K-NN provides a number of advantages, but it also has a number of significant disadvantages or constraints. A few disadvantages of K-NN are described below.

- **K-NN lazy algorithm**: K-NN is a simple method to build, but as the dataset expands, the algorithm's efficiency or speed decreases rapidly.

- **The Curse of Dimensionality:** While KNN works well with a small number of input variables, it fails to anticipate the output of additional data points as the number of variables grows.

- **K-NN requires homogenous characteristics:** If you elect to create k-NN using a common distance, such as the Euclidean or Manhattan distances, it is absolutely required that features have the same scale, since absolute differences in features must have the same weight, i.e., a given distance in feature 1 must mean the same for feature 2.

- **Optimal number of neighbors**: One of the most difficult aspects of K-NN is determining the best number of neighbors to consider when categorizing fresh data.

- **Data that is imbalanced causes issues:** It has been observed that on imbalanced data, k-NN doesn't perform well. If we assume two classes, A and B, and the majority of the training data is classified as A, the model will eventually favor A. This might lead to the classification of the less common class B being incorrect.

- **Outlier sensitivity**: Because it chooses neighbors based on distance criteria, the K-NN method is particularly sensitive to outliers.

- **Missing Value Treatment**: K-NN is incapable of dealing with missing values by default.

Advantages and disadvantages of Supervised Learning

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:01 PM
Book:	Advantages and disadvantages of Supervised Learning		

Table of contents

1. Supervised learning advantages

2. Supervised learning disadvantages

1. Supervised learning advantages

- Supervised learning allows collecting data and produces data output from previous experiences.
- Helps to optimize performance criteria with the help of experience.
- Supervised machine learning helps to solve various types of real-world computation problems.

2. Supervised learning disadvantages

- Classifying big data can be challenging.
- Training for supervised learning needs a lot of computation time. So, it requires a lot of time.

Exercise 1: Supervised Learning (Linear Regression)

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:01 PM
Book:	Exercise 1: Supervised Learning (Linear Regression)		

Table of contents

1. General description

2. Desired objectives

3. Required material

4. Other requirements

1. General description

A steel processing company, as part of its effort to apply predictive maintenance to its equipment, forecasts the next day's energy consumption in order to anticipate anomalies in production operations. You, as a lead machine learning expert, are asked to implement a simple yet effective prediction technique given a dataset containing historic measurements. The given data contains the following attributes

- **Date:** Continuous-time data taken on the first of the month
- **Usage_kWh:** Industry Energy Consumption Continuous kWh
- **Lagging Current:** reactive power Continuous kVarh
- **Leading Current:** reactive power Continuous kVarh
- **CO2:** Continuous ppm
- **NSM:** Number of Seconds from midnight Continuous S
- **Week:** status Categorical (Weekend (0) or a Weekday (1))
- **Day of the week:** Categorical Sunday, Monday: Saturday
- **Load Type:** Categorical Light Load, Medium Load, Maximum Load

Instructions:

The first task is to implement a Linear Regression algorithm, preferably in Python.

1. Download the csv file from the following link (password: dt@mml)

https://versions.aimms.gr/index.php/s/4fvrVA6oK1fd8kZ

2. Load the downloaded csv file (dataset1.csv) and apply the Linear Regression method it is recommended to use the scikit learn library for Python to predict the power usage of a steel processing factory in the data Calculate the values of the Metrics Correlation, Mean Squared Error, and R^2, create the graph that will capture the points of the data set on the plane as well as the regression line.

3. Copy and paste the following notebook to your python IDE and follow the instructions to accomplice the exercise requirements. You should fill the missing parts of the exercise as dictated in the provided comments.

```
#
=============================================================================
# Exercise 1 - Supervised learning
# LINEAR REGRESSION ALGORITHM TEMPLATE
# Complete the missing code by implementing the necessary commands.
#
=============================================================================

# From the 'sklearn' library, we need to import:
# 'datasets', for loading our data
# 'metrics', for measuring scores
# 'linear_model', which includes the LinearRegression() method
# From the 'scipy' library, we need to import:
# 'stats', which includes the spearmanr() and pearsonr() methods for computing correlation
# Additionally, we need to import
# 'pyplot' from package 'matplotlib' for our visualization purposes
# 'numpy', which implements a wide variety of operations
#
=============================================================================

# IMPORT NECESSARY LIBRARIES HERE
import pandas as pd

from sklearn import
#
=============================================================================
# Load the dataset
#
=============================================================================

# ADD COMMAND TO LOAD DATA HERE
steel_power = pd.read_csv('../input/dataset1.csv') # change the path according to the
location

# With this
#
=============================================================================

# Get samples from the data, and keep only the features that you wish.
#
=============================================================================

# Load the features and the target value which in our case is the Usage KhW
# X: features
# Y: target value (prediction target)
X = steel_power.drop('Usage_kWh', axis = 1)
y = df['Usage_kWh']

#
=============================================================================
# Create a linear regression model. All models behave differently, according to
# their own, model-specific parameter values. In our case, however, the linear
# regression model does not have any substantial parameters to tune. Refer
# to the documentation of this technique for more information.
#
=============================================================================


# ADD COMMAND TO CREATE A LINEAR REGRESSION MODEL HERE
```

```
linearRegressionModel =
#
==========================================================================

# Split the dataset that we have into two subsets. We will use
# the first subset for the training (fitting) phase, and the second for the evaluation phase.
# By default, the train set is 75% of the whole dataset, while the test set makes up for the rest
25%.
# This proportion can be changed using the 'test_size' or 'train_size' parameter.
# Also, passing an (arbitrary) value to the parameter 'random_state' "freezes" the splitting
procedure
# so that each run of the script always produces the same results (highly recommended).
# Apart from the train_test_function, this parameter is present in many routines and should
be
# used whenever possible.
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)




# Let's train our model.
#
==========================================================================

# ADD COMMAND TO TRAIN YOUR MODEL HERE
#
==========================================================================

# Ok, now let's predict the output for the test input set
#
==========================================================================

# ADD COMMAND TO MAKE A PREDICTION HERE
y_predicted =

#
==========================================================================


# Time to measure scores. We will compare predicted output (resulting from input x_test)
# with the true output (i.e. y_test).
# You can call 'pearsonr()' or 'spearmanr()' methods for computing correlation,
# 'mean_squared_error()' for computing MSE,
# 'r2_score()' for computing r^2 coefficient.
#
==========================================================================

# ADD COMMANDS TO EVALUATE YOUR MODEL HERE (AND PRINT ON CONSOLE)
print()
print()
print()
#
==========================================================================


# Plot results in a 2D plot (scatter() plot, line plot())
#
==========================================================================
# ADD COMMANDS FOR VISUALIZING DATA (SCATTER PLOT) AND REGRESSION MODEL


# Display 'ticks' on the x-axis and y-axis
plt.xticks()
plt.yticks()

# Show plot
plt.show()

#
======================================================================
```

2. Desired objectives

- Import basic machine learning libraries.
- Get familiar with dataset handling.
- Split dataset to training and testing sections.
- Train a Linear Regression Model.
- Calculate metrics such as R^2 score in order to check the sufficiency of the model.
- Get familiar with plotting the data.

3. Required material

For the execution of this task, you need to:

1. Install Python 3 Release according to your operation system.

- Windows
- macOS
- other

2. A python IDE is required or you can use Google Colab [1]

---

[1] Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

4. Other requirements

There are no other special requirements for the execution of this exercise.

What is unsupervised Learning?

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:01 PM
Book:	What is unsupervised Learning?		

Description

Table of contents

1. Unsupervised Learning Definition

1. Unsupervised Learning Definition

> Unsupervised learning is a sort of algorithm that uses untagged data to discover patterns. The goal is that the machine will be pushed to develop a compact internal picture of its surroundings through imitation, which is a key way of learning in humans, and then generate inventive material from it. Unlike supervised learning, in which data is labeled by an expert, such as "ball" or "fish," unsupervised approaches display self-organization, which captures patterns as probability densities or a mixture of neural feature preferences [31].

For example, assume it is shown a picture that has both **huskies** and **chihuahuas** that it has never seen before.

As a result, the machine has no understanding of the characteristics of huskies and chihuahuas, and we are unable to classify it as such. However, it can classify them based on their similarities, patterns, and differences, allowing us to simply divide the above image into two sections. The first section may have all photos with huskies, while the second half may contain all photos with chihuahuas.

Types of unsupervised learning

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:02 PM
Book:	Types of unsupervised learning		

Description

Unsupervised learning is classified into two categories of problems:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association:** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

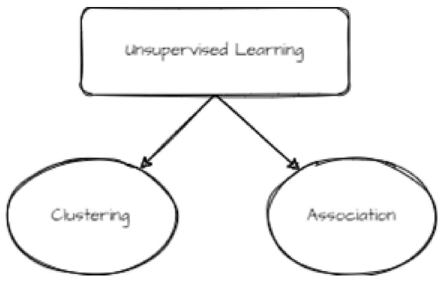


Table of contents

1. Clustering

1.1. Applications of clustering in advanced Manufacturing

1.2. Clustering methods

1.3. Real world manufacturing use case

2. Association

2.1. Association categories

2.2. The basics

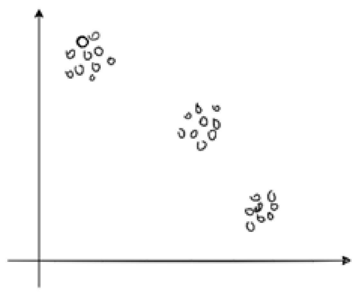
2.3. What is the mechanism of Association Rule Learning?

2.4. Real world Association Rules use case

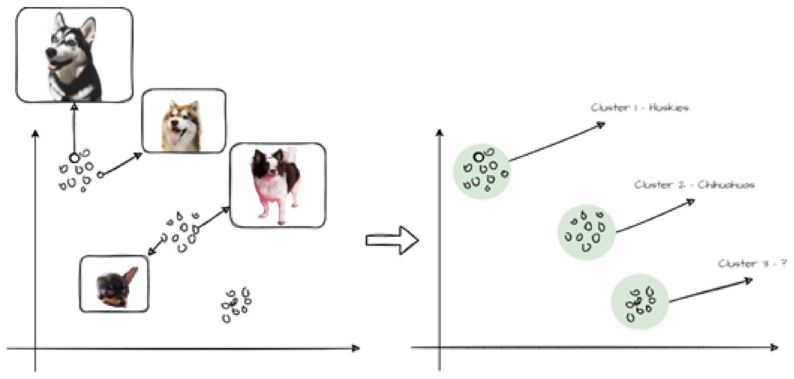
1. Clustering

It's essentially a form of unsupervised learning. Unsupervised learning is a technique for extracting references from datasets that contain input data but no labeled responses. It's a method for identifying significant structure, explaining underlying processes, generating traits, and groups in a set of samples.

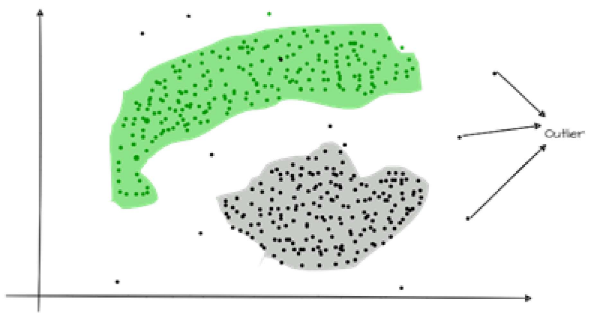
Clustering is the process of dividing a population or set of data points into numerous groups such that data points in the same group are more similar to each other and different from data points in other groups. It's essentially a grouping of items based on their resemblance and dissimilarity.



The data points grouped together in the graph below, for example, may be categorized into a single group. The clusters can be distinguished, and we can see that there are three clusters in the image below.



Clusters aren't all spherical. These data points are grouped based on the premise that each data point must fall within a certain distance from the cluster center. **Outliers** are calculated using a variety of distance measures and approaches.



An outlier is a data point that deviates considerably from the norm. An outlier can be caused by measurement variability or by experimental mistake; the latter is sometimes eliminated from the data set. In statistical analysis, an outlier can generate major consequences.

1.1. Applications of clustering in advanced Manufacturing

There are numerous applications of cluster analysis across the range of sciences. Indicative, we present some of them in the list below.

1. **Predictive Maintenance**

Predictive maintenance is highly depending on anomaly detection therefore in outlier discovery as they indicate non normal behavior. Clustering based outlier detectors have been used in the industry as they offer numerous benefits such as high accuracy. Moreover, such applications operate very quickly with the right configurations and can handle big amounts of data that is a crucial asset nowadays. Last but to least, the unsupervised characteristics allow us to detect early unprecedented failures of the machinery.

2. **Field robotics**

To follow objects and find outliers in sensor data, clustering techniques are utilized in robotic situational awareness. [32]

3. **Pharmacy Industry**

For example, to determine structural similarities, 3000 chemical compounds were grouped in the space of 90 topological indices. [33]

1.2. Clustering methods

There are four main clustering approaches that are based in different mathematical approaches.

- **Density-Based Methods:** These approaches interpret clusters as a dense area with some similarities and differences to the space's lower dense region. These algorithms are highly accurate and can combine two clusters. DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), and other algorithms are examples.

- **Hierarchical Based Methods:** The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. The two categories into which it is divided are the following:

  - Divisive (top-down approach)

  - Agglomerative (bottom-up approach)

CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and Hierarchies), and so on are some examples.

- **Methods of Partitioning:** These approaches divide the items into k clusters, with each division being a single cluster. This approach is used to optimize an objective criterion similarity function, such as K-means, CLARANS (Clustering Large Applications based upon Randomized Search), and others, when distance is a large parameter.

- **Grid-based Techniques:** The data space is divided into a finite number of cells that form a grid-like structure in this technique. STING (Statistical Information Grid), wave cluster, CLIQUE (Clustering in Quest), and other clustering processes on these grids are rapid and independent of the quantity of data items.

1.3. Real world manufacturing use case

In fruit processing industry the categorization of fruit according to its quality characteristics was until recently a laborious job performed by human resources.

For example, peaches are categorized based on their size, shape and other visual quality indicator. Smaller and less qualitive fruits (Group 1) are purposed for juice production, medium sized and average quality ones (Group 2) for canning and jam production and last but not least, the bigger and most undamaged ones (Group 3) for direct consumption. This work, however, can be an application of the machine learning clustering algorithms.

Installing cameras above the work line can provide us instantly the diameter, the shape, the color and other metrics of each fruit and feed a clustering algorithm that could sort out each fruit to one of the above groups. The clustering model could have been trained and categorize the peaches to each group as an experiences fruit selector would.

2. Association

Association rule learning is a machine learning approach that uses rules to uncover interesting relationships between variables in huge databases. Its goal is to detect strong rules identified in databases using various interestingness criteria. [34]. Association rules are intended to uncover the rules that define how or why particular objects are associated in any given transaction involving a range of goods.

Rakesh Agrawal, Tomasz Imieliski, and Arun Swami [35] proposed association rules based on the notion of strong rules for detecting regularities between items in large-scale transaction data captured by point-of-sale (POS) systems in supermarkets. For example, the rule found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. Such data may be utilized to make judgments about marketing operations such as special pricing or product positioning.

Association rules, like the one used in market basket analysis, are used in a variety of applications today, including Web usage mining, intrusion detection, continuous manufacturing, and bioinformatics. In contrast to sequence mining, association rule learning normally does not take into account the order of elements inside or across transactions.

The association rule algorithm itself is comprised of several parameters that might be difficult to perform for people lacking data mining experience, as well as numerous rules that are tough to comprehend [36]. Despite this, association rule learning is an excellent technique for anticipating data connectivity behavior. This distinguishes it as an important tool for categorization, or detecting patterns in data, when using machine learning methods.

2.1. Association categories

The two main types of association rules are the following [37]:

· **Single-dimensional:** refers to one dimension

· **Multidimensional**: refers to more than one dimension

Both forms of association rules can be classified as either Boolean or quantitative. The former is concerned with the existence or absence of an item, whereas the latter is concerned with numerical values that are divided into item intervals.

<u>Example of single-dimensional:</u>

An association rule that describes customers who buy honey and cheese

Buys \(\lbrace{x, honey}\rbrace \Rightarrow \lbrace{x, cheese}\rbrace \)

Customers who buy honey also buy cheese, according to the rule. Buying honey "triggers" the buying of cheese, as seen by the direction of the association, which runs from the left to right.

<u>Example of multi-dimensional:</u>

An association rule describing graduate students might read as follows: major

\(\lbrace{x, Computer Engineering}\rbrace AND \)

takes course \(\lbrace{x, Advanced Data Analysis and Decision Making}\rbrace \Rightarrow level \lbrace{x, PhD}\rbrace \)

Another categorization for the association rules:

· single-level

· multilevel

The former is based on objects that may be articulated at multiple levels in a hierarchy, whereas the later are based on a single degree of abstraction. A multilevel association rule is demonstrated in the example below (which can be contrasted with the single-level rule given in the first example)

A multilevel association rule that describes customers buying eucalyptus honey and large white cheese.

\(buys \lbrace{x, eucalyptus honey}\rbrace \Rightarrow buys \lbrace{x, large white cheese}\rbrace \)

The honey and cheese products are split into different types in the above rule, forming a hierarchy. Cheese, for example, is split into white and yellow varieties, with each of these two types subdivided into small, medium, and large sizes.

2.2. The basics

We use a short example from the grocery realm to demonstrate the ideas. Table 2 depicts a tiny database holding the items, where the value 1 denotes the existence of an item in the associated transaction and the value 0 represents the absence of an item in that transaction. The set of items is:

I={milk,bread,butter,beer,diapers,eggs,fruit}

A supermarket rule may be as follows: {butter,bread}⇒{milk}. Customers will buy milk if they buy butter and bread.

Transaction ID	milk	bread	butter	beer	diapers	eggs	fruit
1	1	1	0	0	0	0	1
2	0	0	1	0	0	1	1
3	0	0	0	1	1	0	0
4	1	1	1	0	0	1	1
5	0	1	0	0	0	0	0

Constraints on various metrics of relevance and interest are used to pick interesting rules from a collection of all feasible rules. The most well-known limitations are minimum support and confidence levels.

Let X,Y be itemsets , X⇒Y an association rule and T a set of transactions of a given database. Please keep in mind that this is a very modest sample. In practice, a rule requires several hundred transactions to be supported before it can be considered statistically significant, and datasets can contain hundreds or millions of interactions.

Support

The amount of support indicates how frequently the itemset appears in the dataset [38].

$$support = P(A \cap B) = \frac{(\text{number of transactions containing } A \text{ and } B)}{(\text{total number of transactions})}$$

where A and B are independent transactions that occurred inside the overall set of recorded transactions. Using Table 2 as an example, the itemset X={beer,diapers} has support of 1/5=0.2 since it happens in 20% of all transactions (1 out of 5 transactions). The argument for X's support is a set of preconditions, and as it expands, it gets more restricted (instead of more inclusive) [39].

Confidence

The percentage of all transactions that fulfill both X and Y is known as confidence [40].

The ratio of transactions having both X and Y to the entire quantity of X values present, where X is the antecedent and Y is the consequent, is the confidence value of an association rule in terms of T, typically indicated as X⇒Y.

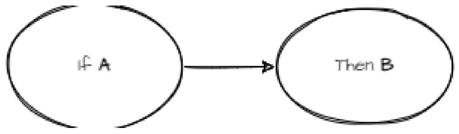
$$\mathrm{conf}(X \Rightarrow Y) = P(Y|X) = \frac{\mathrm{supp}(X \cap Y)}{\mathrm{supp}(X)} = \frac{\text{number of transactions containing } X \text{ and } Y}{\text{number of transactions containing } X}$$

The equation shows how to calculate confidence by comparing the co-occurrence of transactions X and Y in the dataset versus transactions containing only X. This indicates that the total number of transactions in X and Y is divided by the total number of transactions in X.

For example, Table 2 shows the rule {butter,bread}⇒{milk} which has a confidence of (1/5)/(1/5)=0.2/0.2=1 when a client buys butter and bread, they also buy milk, according to the dataset. For transactions comprising both butter and bread, this example shows that the rule is correct 100 percent of the time. The rule {fruit}⇒{eggs}, however, has a confidence of (2/5)/(3/5)=0.4/0.6=0.67. This means that eggs are purchased 67 percent of the time when fruit is delivered. Fruit is purchased a total of three times in this dataset, with two of those purchases consisting of egg purchases.

2.3. What is the mechanism of Association Rule Learning?

Association rule learning works on the concept of If and Else Statement, such as if A then B [41].



The If element is referred to as **antecedent,** and then statement is called as **Consequent.** Single cardinality refers to relationships in which we can find an association or link between two objects. It's all about making rules, and as the number of objects increases, so does cardinality. There are numerous metrics for measuring the relationships between thousands of data items. The best known are described in the above section (The basics).

2.4. Real world Association Rules use case

A type of association rule mining is the sequential pattern mining (SPM), which identifies associations between time ordered events/records, providing sets of sequential patterns. A real manufacturing case study is presented in [42], where the goal is to predict critical failures on aircrafts using post flight report data. When these failures occur, the aircraft must stay on the ground for control or repair and therefore, being able to predict these failures in advance and plan the corresponding maintenance increases the availability of the aircraft. The research work [43] applies and evaluates a set of machine learning algorithms on event logs, which consist of time-ordered events from the alphabet of events included in ATA numbering system [https://en.wikipedia.org/wiki/ATA_100]; Table I shows an example of an event log file. An event is composed by a timestamp that corresponds to the time of its occurrence and other attributes that contain information about the event. These attributes may include a system or subsystem identifier that generated the event, a task id (in information system logs), description about the activity (in server logs), failure description (in hardware monitoring) and a unique event identifier.

Applying SPM we can discover useful patterns in the data about associations between fault events as a sequence of minor faults or other events can potentially lead to a major failure.

Sys id	**Timestamp**	**Event id**	**Source**	**Description**
sys1	2013-11-10 13:30	6226	3412	failure 1
sys1	2013-11-10 14:33	3401	4902	failure 1
sys2	2013-12-10 10:12	3401	5552	failure 2
sys3	2013-12-10 11:40	408	4414	failure 2
. . .	. . .	. . .	. . .	. . .

Unsupervised learning advantages

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:02 PM
Book:	Unsupervised learning advantages		

Description

Table of contents

1. Advantages

1. Advantages

· It has the ability to see what human mind cannot visualize.

· It is very significant for the industry as well dig hidden patterns which hold utmost importance and has extensive real-time applications.

· An unsupervised job might result in the creation of a completely new company segment or enterprise.

· Compared to the supervised learning process, there is a reduction in complexity.

· There is no requirement to interpret the relevant labels thus reducing the complexity.

· Unlabeled data is a lot easier to get by.

Unsupervised learning disadvantages

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:03 PM
Book:	Unsupervised learning disadvantages		

Table of contents

1. Disadvantages

1. Disadvantages

· It is more expensive since human involvement may be required to comprehend the patterns and link them with domain expertise.

· Because there is no label or output metric to certify its usefulness, it is not always assured that the generated results will be beneficial.

· An unsupervised task's sorting and output cannot be precisely defined. It is highly reliant on the model and, as a result, on the machine.

· The precision of the results is frequently inadequate.

Classification vs clustering in machine learning

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:03 PM
Book:	Classification vs clustering in machine learning		

Table of contents

1. Classification vs clustering

1. Classification vs clustering

Although they are techniques that belong to one supervised learning and the other to unsupervised learning, both follow comparable process as they are used for the categorization of items into one or more classes based on the characteristics. However, these present a slight difference that is related about the labels. For example, clustering does not include labels, while in the classification, each input instance is given a preset label based on its attributes [44].

Type	Description	Fundamental	Need	Complexity	Example Algorithms
CLASSIFICATION	belongs to supervised learning category	the process of categorizing the input instances according to their class labels	It has label, hence a training and testing dataset is required to verify the model.	more complex as compared to clustering	Support vector machines, Logistic regression, Naive Bayes classifier etc.
CLUSTERING	belongs to unsupervised learning category	without the use of class labels, grouping the instances based on their similarity	There is no requirement for a dataset for training and testing.	less complex as compared to classification	Gaussian (EM) clustering algorithm, k-means clustering algorithm, Fuzzy c-means clustering algorithm

Differences between classification and clustering

1. In supervising learning category, classification is applied while clustering is used for unsupervised learning.

2. Classification is the process of categorizing input instances based on their associated class labels, whereas clustering is the process of grouping instances based on their similarity without the use of class labels.

3. Because classification has labels, a training and testing dataset is required for confirming the model, but clustering does not require a training and testing dataset.

4. Compared to clustering, it is certain that classification is more complex. This complexity is due to the existence of many levels in classification phase, while in clustering only grouping takes place.

5. Logistic regression, Naive Bayes classifiers, Support vector machines, and other classification methods are examples. The k-means clustering method, the Fuzzy c-means clustering algorithm, the Gaussian (EM) clustering algorithm, and others are examples of clustering algorithms.

Linear Regression with Python

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:04 PM
Book:	Linear Regression with Python		

Table of contents

1. Basics

2. Linear Regression with Python
2.1. Step 1: Import packages and classes
2.2. Step 2: Provide data
2.3. Step 3: Create a model and fit it
2.4. Step 4: Get results
2.5. Step 5: Predict response

1. Basics

Machine learning is an emerging technology in the computer science, including many different applications in healthcare, manufacturing, and e-commerce as we have already mentioned in the previous section.

NASA, Uber, Tesla, Google and Amazon all use machine learning in their algorithms. There is at least one more thing these companies have in common. They all use Python as their preferred language for their machine learning projects. There are several reasons why Python is so used in machine learning [48]. The main ones are:

1. **It's simple to use and enables for quick data validation**

Machine learning is used to identify patterns in data. To construct intelligent algorithms, a machine learning engineer is responsible for obtaining, processing, refining, cleaning, organising, and making sense of data. Python is a simple language to learn. While linear algebra and calculus topics might be quite difficult, they need the most effort. Python can be swiftly implemented, allowing machine learning developers to quickly evaluate a concept.

2. **Python is known for its immense libraries**

Python's access to many libraries is one of the key reasons it is the chosen language for machine learning. A library is a set of functions and procedures that may be used by a computer language. Having access to a variety of libraries helps developers to do complicated tasks without having to rewrite a large number of lines of code. Python libraries make it easier for data scientists to conduct numerous research since machine learning depends largely on mathematical optimization, probability, and statistics. Here are some of the Python libraries you may use:

- **Pandas:** is recommended for high-level data structures and analysis.
- **StatsModels:** it is used for statistical methods, data exploration, and a variety of other applications.
- **Keras:** known for its use in deep learning.
- **Matplotlib:** for developing 2D plots, histograms, charts, and other graphs.

3. **A low entry barrier**

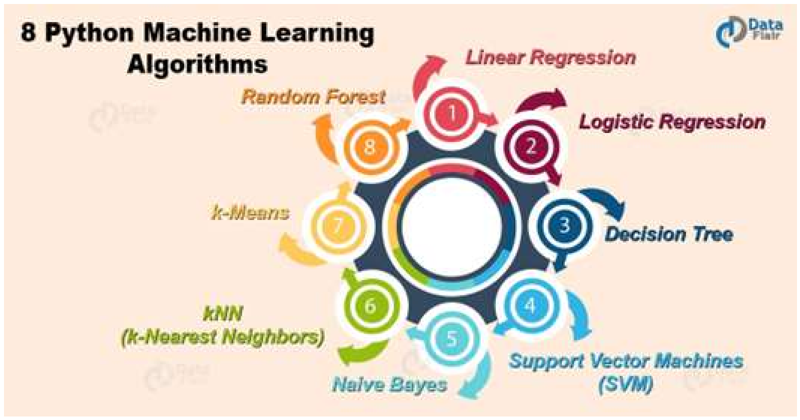
There is a global shortage of programmers. Python is a simple language to learn with a low entrance barrier. What does this imply? More data scientists will be able to master it fast, allowing them to participate in machine learning initiatives. Python is, believe it or not, remarkably close to the English language, making it easy to learn. You can comfortably work with complicated systems because to its simple phrase structure.

4. **Distinguished for its flexibility**

Why is Python the best language for machine learning? Because it allows us a lot of flexibility. Python may be used in conjunction with other programming languages by developers to achieve their objectives. The source code does not need to be recompiled. Any modifications may be made instantaneously, allowing for quick viewing of the outcomes. Because of Python's versatility, the chances of problems developing are quite low.

5. **It is versatile**

Testing is a crucial aspect of software development. Python for machine learning runs on almost any platform, including Windows, macOS, Linux, Unix, and a slew of others. What is the significance of this? Because you can run tests on any platform, it makes testing a breeze. To prepare their code for multiple platforms, your developers only need to utilize PyInstaller, for example. Python can help you save a lot of time and money when it comes to machine learning.



6. **It's legible**

Python is simple to read, so any Python developer may readily implement, duplicate, or distribute it anytime a code update is necessary. It removes ambiguity, errors, and competing paradigms, resulting in more efficient algorithm interchange, idea sharing, and tool sharing between AI and machine learning specialists. There are also applications like IPython that provide further functionality like testing, debugging, tab completion, and so forth. Parallel application development, execution, debugging, and interactive monitoring are all possible.

7. **Python is becoming even more popular**

Python is quickly becoming the most widely used programming language on the planet. Python is the programming language of choice for many well-known organizations, like Facebook, Google, Quora, Amazon, and Netflix, to mention a few, due to its simplicity, adaptability, and ease of maintenance. Machine learning, artificial intelligence, and robotics are just a few of the intriguing and new technologies that employ it.

Python is also in great demand at institutions, where it is quickly becoming the most popular introductory language. It's also something that a lot of seasoned developers learn to add to their skill set. The more companies and individuals who utilize Python, the better. More resources are being developed around it, assisting developers in completing complicated jobs without encountering code issues.

8. **Large support community**

Python has a large community of supporters, and it's comforting to know that if you have an issue, someone will be able to assist you. Python is an open-source language, which means that it has a large library of resources that can be utilized by both beginners and experts.

Many regularly occurring topics are discussed in Python forums and communities, and Python documentation is available online. If you run into an issue that you cannot handle on your own, you can always ask a question and get help from other developers

2. Linear Regression with Python

Simple Linear Regression with scikit – learn

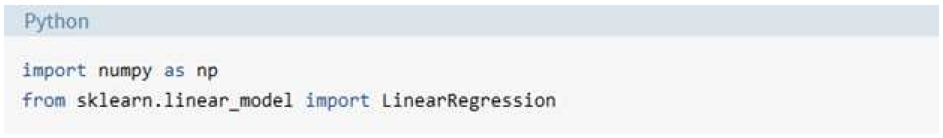
Let's start with the most fundamental case: basic linear regression. When using linear regression, there are five main procedures to follow:

1. Import the necessary packages and classes.
2. Provide data to work with, and then do the necessary transformations.
3. Develop a regression model and test it against historical data.
4. Check the model fitting results to see if the model is suitable.
5. Use the model to make predictions.

The majority of regression techniques and implementations follow these phases [49].

2.1. Step 1: Import packages and classes

The first step is to import the package **numpy** and the class **LinearRegression** from **sklearn.linear_model**:

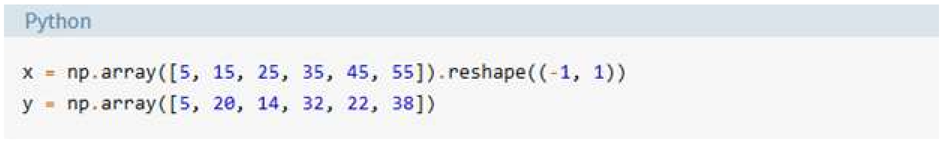

Python

```python
import numpy as np
from sklearn.linear_model import LinearRegression
```

You now have access to all of the features you'll need to run linear regression. NumPy's most basic data type is numpy.ndarray, which is an array type. The term array is used throughout this text to refer to instances of the numpy.ndarray type. The class sklearn.linear_model.LinearRegression will be used to perform linear and polynomial regression and make predictions accordingly.

2.2. Step 2: Provide data

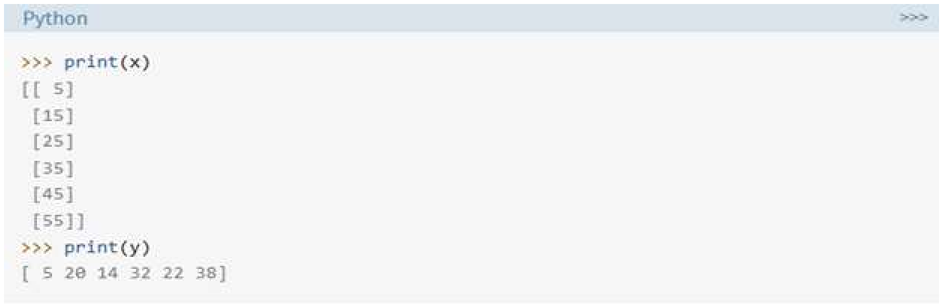
The second stage is to define the data that will be used. The inputs (regressors, x) and outputs (predictor, y) should both be arrays (ndarray instances) or comparable objects. This is the most straightforward method of supplying data for regression:



Python

```python
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
```

You now have two arrays: an input x array and an output y array. Because this array must be two-dimensional, or to be more specific, contain one column and as many rows as necessary, you should call.reshape() on x. That is precisely what the.reshape() parameter (-1, 1) specifies. This is how x and y now appear:



Python >>>

```python
>>> print(x)
[[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]]
>>> print(y)
[ 5 20 14 32 22 38]
```

As can be seen, x has two dimensions and x.shape is (6, 1), however y only has one dimension and y.shape is (0, 1). (6,).

2.3. Step 3: Create a model and fit it

The following goal is to construct a linear regression model and fit it to the data. To describe the regression model, let's build an instance of the class LinearRegression:

Python

```python
model = LinearRegression()
```

This statement constructs a variable model as a LinearRegression object. LinearRegression accepts a number of optional parameters:

- **fit_intercept** is a Boolean (True by default) that determines whether to compute or ignore the intercept b0 (True) (False).
- **normalize** is a Boolean value (False by default) that determines whether or not the input variables should be normalized (True) (False).
- **copy_X** is a Boolean (True by default) that determines whether the input variables should be copied (True) or overwritten (False) (False).
- **n_jobs** : The number of tasks utilized in parallel computing is represented by n jobs, which can be an integer or None (default). None normally denotes one job, whereas -1 denotes the utilization of all processors.

All parameters are set to their default settings in this example. It's time to put the model to work. To begin, you must first call.fit() on model:

Python

```python
model.fit(x, y)
```

Using the existing input and output (x and y) as arguments,.fit() calculates the best values of the weights b_0 and b_1,. to put it another way,.fit() fits the model. It returns self, which is the model variable. As a result, the final two sentences may be replaced with the following:

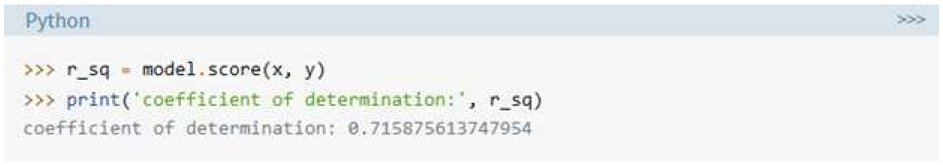
Python

```python
model = LinearRegression().fit(x, y)
```

The purpose of this statement is the same as the previous two. It is simply that it is shorter.

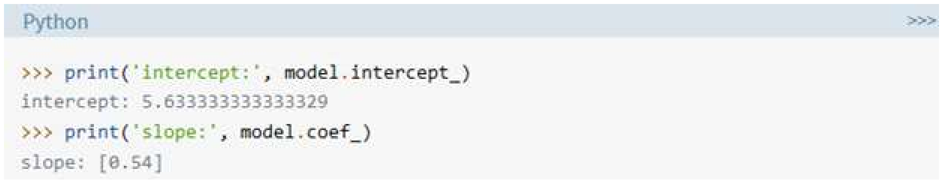
2.4. Step 4: Get results

After you've fitted your model, you may retrieve the data to see if it's working properly and interpret it. With .score() on model, you may get the coefficient of determination R^2


Python

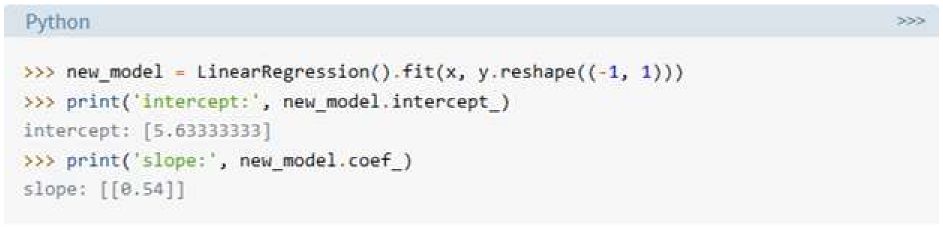
```
>>> r_sq = model.score(x, y)
>>> print('coefficient of determination:', r_sq)
coefficient of determination: 0.715875613747954
```

When you're applying .score(), the arguments are also the predictor x and regressor y, and the return value is R^2. The attributes of model are .intercept_, which represents the coefficient, b_0 and .coef_, which represents b_1:


Python

```
>>> print('intercept:', model.intercept_)
intercept: 5.633333333333329
>>> print('slope:', model.coef_)
slope: [0.54]
```

The code above shows how to obtain b_0 and b_1. It's worth noting that .intercept_ is a scalar and .coef_ is an array. When x is zero, your model predicts the answer 5.63, as shown by the value b_0 = 5.63 (roughly). The result b_1 = 0.54 indicates that when x is increased by one, the projected reaction increases by 0.54. It's worth noting that you may also supply y as a two-dimensional array. You'll obtain a similar result in this scenario. This is how it may appear:


Python

```
>>> new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
>>> print('intercept:', new_model.intercept_)
intercept: [5.63333333]
>>> print('slope:', new_model.coef_)
slope: [[0.54]]
```

As you can see, this example is fairly similar to the last one, except .intercept_ is a one-dimensional array with a single element b_0 and .coef_ is a two-dimensional array with a single element b_1 .

2.5. Step 5: Predict response

Once you've found a model that you like, you may use it to make predictions with existing or new data. Use.predict() to get the predicted response:

Python

```
>>> y_pred = model.predict(x)
>>> print('predicted response:', y_pred, sep='\n')
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

When applying .predict(), you pass the regressor as the argument and get the corresponding predicted response. This is a nearly identical way to predict the response:

Python

```
>>> y_pred = model.intercept_ + model.coef_ * x
>>> print('predicted response:', y_pred, sep='\n')
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
```

In this scenario, model.coef_is multiplied by each element of x, and model.intercept is added to the result. Only the dimensions of the output alter from the previous example. The expected response is now a two-dimensional array, rather than a one-dimensional array as before. These two procedures will provide the same result if the number of dimensions of x is reduced to one. When multiplying with model.coef_, you may achieve this by substituting x with x.reshape(-1), x.flatten(), or x.ravel(). Regression models are frequently used for forecasting in practice. This implies you may utilize fitted models to determine outputs depending on some new, additional inputs:

Python

```
>>> x_new = np.arange(5).reshape((-1, 1))
>>> print(x_new)
[[0]
 [1]
 [2]
 [3]
 [4]]
>>> y_new = model.predict(x_new)
>>> print(y_new)
[5.63333333 6.17333333 6.71333333 7.25333333 7.79333333]
```

Here .predict() returns the result y when applied to the new regressor x_new. This example makes use of numpy's arange() function to create an array containing members ranging from 0 (inclusive) to 5 (exclusive), i.e. 0, 1, 2, 3, and 4.

Multiple Linear Regression with scikit-learn

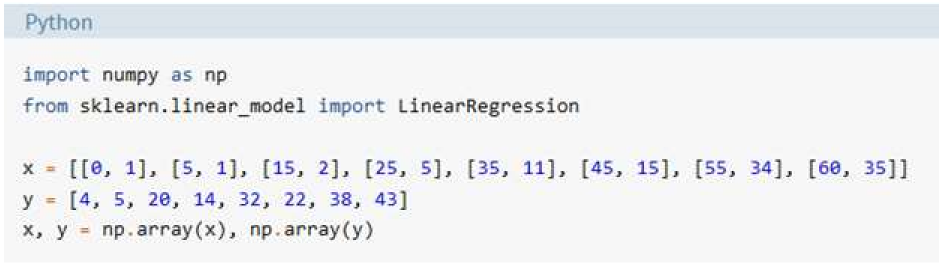
Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:04 PM
Book:	Multiple Linear Regression with scikit-learn		

Table of contents

1. Step 1 and 2: Import packages and classes and provide data
2. Step 3: Create a model and fit it
3. Step 4: Get results
4. Step 5: Predict response

1. Step 1 and 2: Import packages and classes and provide data

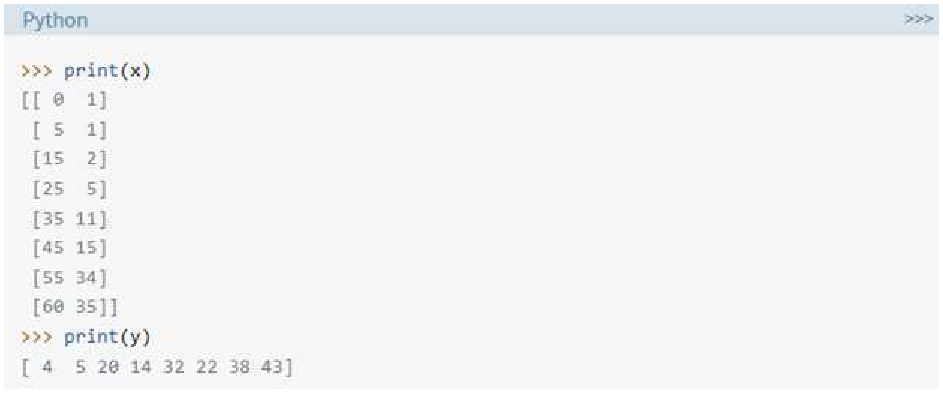
First, you import numpy and sklearn.linear_model.LinearRegression and provide known inputs and output:



Python

```python
import numpy as np
from sklearn.linear_model import LinearRegression

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)
```

That's a simple way to define the input x and output y. You can print x and y to see how they look now:



Python >>>

```
>>> print(x)
[[ 0  1]
 [ 5  1]
 [15  2]
 [25  5]
 [35 11]
 [45 15]
 [55 34]
 [60 35]]
>>> print(y)
[ 4  5 20 14 32 22 38 43]
```

x is a two-dimensional array with at least two columns in multiple linear regression, whereas y is normally a one-dimensional array. This is a simple multiple linear regression example, with x having precisely two columns.

2. Step 3: Create a model and fit it

The next step is to create the regression model as an instance of LinearRegression and fit it with .fit():

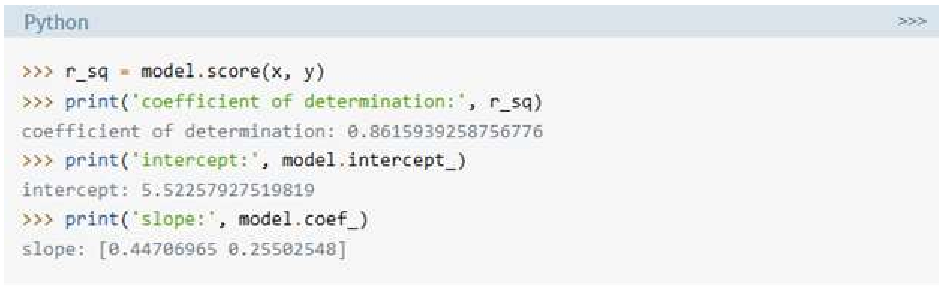
Python

```python
model = LinearRegression().fit(x, y)
```

The variable model refers to the object of type LinearRegression as a result of this statement. It indicates the regression model that has been fitted to the data.

3. Step 4: Get results

You can obtain the properties of the model the same way as in the case of simple linear regression:



```
Python                                                        >>>

>>> r_sq = model.score(x, y)
>>> print('coefficient of determination:', r_sq)
coefficient of determination: 0.8615939258756776
>>> print('intercept:', model.intercept_)
intercept: 5.52257927519819
>>> print('slope:', model.coef_)
slope: [0.44706965 0.25502548]
```

Using .score() you obtain the value of R^2 and the values of the estimators of regression coefficients with .intercept_ and .coef_. Again, .intercept_ holds the bias b_0, while now .coef_ is an array containing b_1 and b_2 respectively.

The intercept in this case is around 5.52, which is the value of the projected response when $x_1 = x_2 = 0$. The projected response rises by 0.45 when x1 is increased by one. When x2 increases by one, the response increases by 0.26.

4. Step 5: Predict response

Predictions also work the same way as in the case of simple linear regression:

Python

```
>>> y_pred = model.predict(x)
>>> print('predicted response:', y_pred, sep='\n')
predicted response:
[ 5.77760476  8.012953   12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
```

The predicted response is obtained with .predict(), which is very similar to the following:

Python

```
>>> y_pred = model.intercept_ + np.sum(model.coef_ * x, axis=1)
>>> print('predicted response:', y_pred, sep='\n')
predicted response:
[ 5.77760476  8.012953   12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
```

You can predict the output values by multiplying each column of the input with the appropriate weight, summing the results, and adding the intercept to the sum. You can apply this model to new data as well:

Python

```
>>> x_new = np.arange(10).reshape((-1, 2))
>>> print(x_new)
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
>>> y_new = model.predict(x_new)
>>> print(y_new)
[ 5.77760476  7.18179502  8.58598528  9.99017554 11.3943658 ]
```

That is the prediction using a linear regression model.

Polynomial Regression with Python

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:05 PM
Book:	Polynomial Regression with Python		

Description

Implementing polynomial regression with scikit-learn is very similar to linear regression [47]. There is only one extra step: you need to transform the array of inputs to include non-linear terms such as x^2 .

Table of contents

1. Step 1: Import packages and classes
2. Step 2a: Provide data
3. Step 2b: Transform data
4. Step 3: Create a model and fit it
5. Step 4: Get results
6. Step 5: Predict response

1. Step 1: Import packages and classes

In addition to numpy and sklearn.linear_model.LinearRegression, you should also import the class PolynomialFeatures from sklearn.preprocessing:

Python

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

The import is now done, and you have everything you need to work with.

2. Step 2a: Provide data

This step defines the input and output and is the same as in the case of linear regression:

```python
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([15, 11, 2, 8, 25, 32])
```

Now you have the input and output in a suitable format. Keep in mind that you need the input to be a **two-dimensional array**. That's why .reshape() is used.

3. Step 2b: Transform data

This is the new step you need to implement for polynomial regression. As you've seen earlier, you need to include x^2 (and perhaps other terms) as additional features when implementing polynomial regression. For that reason, you should transform the input array x to contain the additional column(s) with the values of x^2 (and eventually more features).

It's possible to transform the input array in several ways (like using insert() from numpy), but the class PolynomialFeatures is very convenient for this purpose. Let's create an instance of this class:

Python

```python
transformer = PolynomialFeatures(degree=2, include_bias=False)
```

The variable transformer refers to an instance of PolynomialFeatures which you can use to transform the input x. You can provide several optional parameters to PolynomialFeatures:

- **degree** is an integer (2 by default) that represents the degree of the polynomial regression function.
- **interaction_only** is a Boolean (False by default) that decides whether to include only interaction features (True) or all features (False).
- **include_bias** is a Boolean (True by default) that decides whether to include the bias (intercept) column of ones (True) or not (False).

This example uses the default values of all parameters, but you'll sometimes want to experiment with the degree of the function, and it can be beneficial to provide this argument anyway. Before applying transformer, you need to fit it with .fit():

Python

```python
transformer.fit(x)
```

Once transformer is fitted, it's ready to create a new, modified input. You apply .transform() to do that:

Python

```python
x_ = transformer.transform(x)
```

That's the transformation of the input array with .transform(). It takes the input array as the argument and returns the modified array. You can also use .fit_transform() to replace the three previous statements with only one:

Python

```python
x_ = PolynomialFeatures(degree=2, include_bias=False).fit_transform(x)
```

That's fitting and transforming the input array in one statement with .fit_transform(). It also takes the input array and effectively does the same thing as .fit() and .transform() called in that order. It also returns the modified array. This is how the new input array looks:

Python >>>

```python
>>> print(x_)
[[   5.   25.]
 [  15.  225.]
 [  25.  625.]
 [  35. 1225.]
 [  45. 2025.]
 [  55. 3025.]]
```

The modified input array contains two columns: one with the original inputs and the other with their squares.

4. Step 3: Create a model and fit it

This step is also the same as in the case of linear regression. You create and fit the model:

```python
model = LinearRegression().fit(x_, y)
```

The regression model is now created and fitted. It's ready for application. You should keep in mind that the first argument of .fit() is the modified input array x_ and not the original x.

5. Step 4: Get results

The properties of the model may be obtained in the same manner as linear regression properties can be obtained:

```
Python                                                        >>>

>>> r_sq = model.score(x_, y)
>>> print('coefficient of determination:', r_sq)
coefficient of determination: 0.8908516262498564
>>> print('intercept:', model.intercept_)
intercept: 21.372321428571425
>>> print('coefficients:', model.coef_)
coefficients: [-1.32357143  0.02839286]
```

Again, .score() returns R^2. Its first argument is also the modified input x_, not x. The values of the weights are associated to .intercept_ and .coef_: .intercept_ represents b_0, while .coef_ references the array that contains b_1 and b_2 respectively. You can obtain a very similar result with different transformation and regression arguments:

```
Python

x_ = PolynomialFeatures(degree=2, include_bias=True).fit_transform(x)
```

If you call PolynomialFeatures with the default parameter include_bias=True (or if you just omit it), you'll obtain the new input array x_ with the additional leftmost column containing only ones. This column corresponds to the intercept. This is how the modified input array looks in this case:

```
Python                                                        >>>

>>> print(x_)
[[1.000e+00 5.000e+00 2.500e+01]
 [1.000e+00 1.500e+01 2.250e+02]
 [1.000e+00 2.500e+01 6.250e+02]
 [1.000e+00 3.500e+01 1.225e+03]
 [1.000e+00 4.500e+01 2.025e+03]
 [1.000e+00 5.500e+01 3.025e+03]]
```

The first column of x_ contains ones, the second has the values of x, while the third holds the squares of x. The intercept is already included with the leftmost column of ones, and you don't need to include it again when creating the instance of LinearRegression. Thus, you can provide fit_intercept=False. This is how the next statement looks:

```
Python

model = LinearRegression(fit_intercept=False).fit(x_, y)
```

The variable model again corresponds to the new input array x_. Therefore x_ should be passed as the first argument instead of x. This approach yields the following results, which are like the previous case:

```
Python                                                        >>>

>>> r_sq = model.score(x_, y)
>>> print('coefficient of determination:', r_sq)
coefficient of determination: 0.8908516262498565
>>> print('intercept:', model.intercept_)
intercept: 0.0
>>> print('coefficients:', model.coef_)
coefficients: [21.37232143 -1.32357143  0.02839286]
```

You see that now .intercept_ is zero, but .coef_ actually contains b_0 as its first element. Everything else is the same.

6. Step 5: Predict response

If you want to get the predicted response, just use .predict(), but remember that the argument should be the modified input x_ instead of the old x:

```
Python                                                                        >>>

>>> y_pred = model.predict(x_)
>>> print('predicted response:', y_pred, sep='\n')
predicted response:
[15.46428571  7.90714286  6.02857143  9.82857143 19.30714286 34.46428571]
```

As you can see, the prediction is nearly identical to that of linear regression. It just needs the updated input rather than the original. If you have many input variables, you may use the same technique. You'll have a multi-column input array, but everything else remains the same. Let's see the next example:

```
Python

# Step 1: Import packages
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Step 2a: Provide data
x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)

# Step 2b: Transform input data
x_ = PolynomialFeatures(degree=2, include_bias=False).fit_transform(x)

# Step 3: Create a model and fit it
model = LinearRegression().fit(x_, y)

# Step 4: Get results
r_sq = model.score(x_, y)
intercept, coefficients = model.intercept_, model.coef_

# Step 5: Predict
y_pred = model.predict(x_)
```

This regression example yields the following results and predictions:

```
Python                                                                        >>>

>>> print('coefficient of determination:', r_sq)
coefficient of determination: 0.9453701449127822
>>> print('intercept:', intercept)
intercept: 0.8430556452395734
>>> print('coefficients:', coefficients, sep='\n')
coefficients:
[ 2.44828275  0.16160353 -0.15259677  0.47928683 -0.4641851 ]
>>> print('predicted response:', y_pred, sep='\n')
predicted response:
[ 0.54047408 11.36340283 16.07809622 15.79139    29.73858619 23.50834636
 39.05631386 41.92339046]
```

In this case, there are six regression coefficients (including the intercept), as shown in the estimated regression function $f(x_1, x_2) = b_0 + b_1x_1 + b_2x_2 + b_3x_1^2 + b_4x_1x_2 + b_5x_2^2$. You'll also observe that, for the identical issue, polynomial regression produced a greater coefficient of determination than multiple linear regression. At first glance, getting such a high R^2 may appear to be a fantastic achievement. It's possible. In real-world scenarios, however, a sophisticated model with an R^2 near to 1 might indicate overfitting. You should evaluate a model's performance with new data, that is, observations that were not utilized to fit (train) the model.

Advanced Linear Regression with statsmodel

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:05 PM
Book:	Advanced Linear Regression with statsmodel		

Description

You can use the statsmodels module to construct linear regression in Python rather quickly. This is usually preferred when more thorough results are required. The process is comparable to scikit- learn's [47].

Table of contents

1. Step 1: Import packages
2. Step 2: Provide data
3. Step 3: Create a model and fit it
4. Step 4: Get results
5. Step 5: Predict response

1. Step 1: Import packages

You must first do some imports. You must also import statsmodels.api in addition to numpy:

Python

```python
import numpy as np
import statsmodels.api as sm
```

You now have all of the packages you require.

2. Step 2: Provide data

You may provide inputs and outputs in the same way that you did with scikit-learn:

```python
x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)
```

Although the input and output arrays have been created, the work is not yet complete. If you want statsmodels to calculate the intercept b0, you must include the column of ones in the inputs. By default, b0 isn't taken into consideration. This is simply one call to a function:

```python
x = sm.add_constant(x)
```

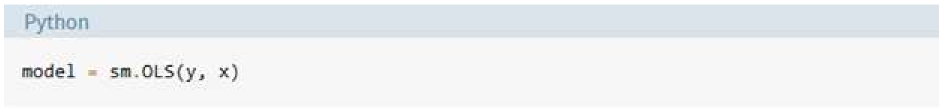
That's how you add the column of ones to x with add_constant(). It takes the input array x as an argument and returns a new array with the column of ones inserted at the beginning. This is how x and y look now:

```python
>>> print(x)
[[ 1.  0.  1.]
 [ 1.  5.  1.]
 [ 1. 15.  2.]
 [ 1. 25.  5.]
 [ 1. 35. 11.]
 [ 1. 45. 15.]
 [ 1. 55. 34.]
 [ 1. 60. 35.]]
>>> print(y)
[ 4  5 20 14 32 22 38 43]
```

You can see that the modified x has three columns: the first column of ones (corresponding to b_0 and replacing the intercept) as well as two columns of the original features.

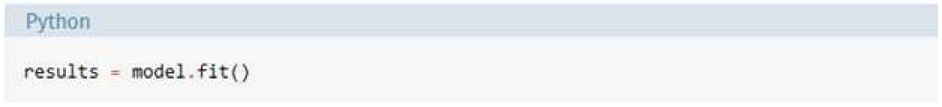
3. Step 3: Create a model and fit it

The class statsmodels.regression.linear model.OLS represents a regression model based on ordinary least squares. Here's how you can get one:



```python
model = sm.OLS(y, x)
```

You must use caution in this situation. Please note that the output is the first parameter, followed by the input. There are a few more options available. You may apply .fit() on your model once it's been created:

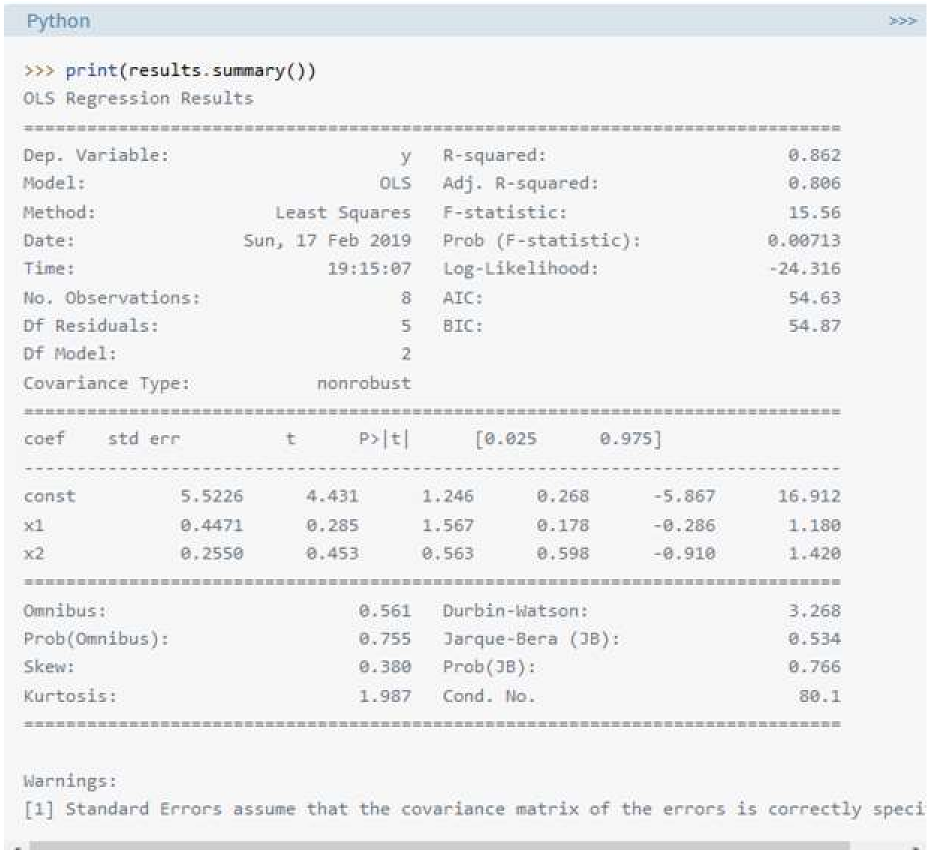


```python
results = model.fit()
```

By calling .fit(), you obtain the variable results, which is an instance of the class statsmodels.regression.linear_model.RegressionResultsWrapper. This object contains a wealth of information on the regression model.

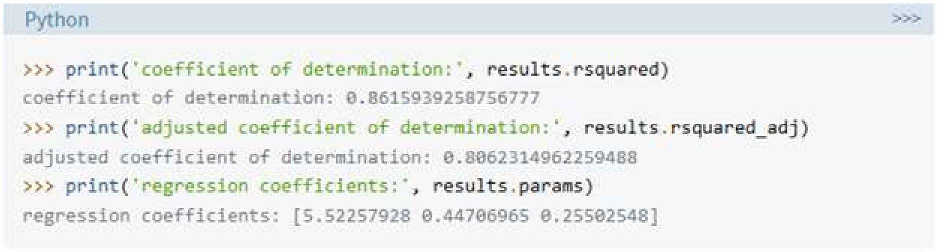
4. Step 4: Get results

The object containing detailed information about the outcomes of linear regression is referred to as the variable results. Although it is beyond the scope of this essay to explain them, you will discover how to extract them here. You may receive the table with the results of linear regression by calling .summary():



```
Python                                                                        >>>

>>> print(results.summary())
OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.862
Model:                            OLS   Adj. R-squared:                  0.806
Method:                 Least Squares   F-statistic:                     15.56
Date:                Sun, 17 Feb 2019   Prob (F-statistic):            0.00713
Time:                        19:15:07   Log-Likelihood:                -24.316
No. Observations:                   8   AIC:                             54.63
Df Residuals:                       5   BIC:                             54.87
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.5226      4.431      1.246      0.268      -5.867      16.912
x1             0.4471      0.285      1.567      0.178      -0.286       1.180
x2             0.2550      0.453      0.563      0.598      -0.910       1.420
==============================================================================
Omnibus:                        0.561   Durbin-Watson:                   3.268
Prob(Omnibus):                  0.755   Jarque-Bera (JB):                0.534
Skew:                           0.380   Prob(JB):                        0.766
Kurtosis:                       1.987   Cond. No.                         80.1
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
```

This table contains a lot of information. R^2, b_0, b_1, and b_2 are some of the statistical values related with linear regression. In this situation, you may receive a warning about kurtosistest. Due to the minimal number of observations supplied, this is the case. Any of the values in the table above can be extracted. Here's an illustration:



```
Python                                                                        >>>

>>> print('coefficient of determination:', results.rsquared)
coefficient of determination: 0.8615939258756777
>>> print('adjusted coefficient of determination:', results.rsquared_adj)
adjusted coefficient of determination: 0.8062314962259488
>>> print('regression coefficients:', results.params)
regression coefficients: [5.52257928 0.44706965 0.25502548]
```

This is how you get some of the linear regression results:

- **.rsquared** holds R^2.
- **.rsquared_adj** represents adjusted R^2 (R^2 corrected according to the number of input features).
- **.params** refers the array with b_0, b_1, and b_2 respectively.

You'll also observe that these results are similar to those produced for the same issue using scikit-learn.

5. Step 5: Predict response

Using .fittedvalues or .predict() with the input array as the parameter, you may get the predicted response on the input values used to create the model:

```
Python                                                        >>>

>>> print('predicted response:', results.fittedvalues, sep='\n')
predicted response:
[ 5.77760476  8.012953   12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
>>> print('predicted response:', results.predict(x), sep='\n')
predicted response:
[ 5.77760476  8.012953   12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
```

For known inputs, this is the predicted response. You may also use if you decide to make predictions using new regressors. new data as the parameter to predict():

```
Python                                                        >>>

>>> x_new = sm.add_constant(np.arange(10).reshape((-1, 2)))
>>> print(x_new)
[[1. 0. 1.]
 [1. 2. 3.]
 [1. 4. 5.]
 [1. 6. 7.]
 [1. 8. 9.]]
>>> y_new = results.predict(x_new)
>>> print(y_new)
[ 5.77760476  7.18179502  8.58598528  9.99017554 11.3943658 ]
```

You'll see that the predicted outcomes for the same issue are identical to those generated using scikit-learn.

Linear regression isn't always the best choice, especially for complicated nonlinear models. There are, fortunately, additional regression approaches that may be used in circumstances when linear regression fails. Support vector machines, decision trees, random forests, and neural networks are a few of them.

There are a slew of Python libraries that use these approaches to do regression. So, python is one of the most popular programming languages for machine learning for this reason.

Other regression approaches can be used in a similar way to what you've seen with the scikit-learn package. It includes support vector machines, decision trees, random forests, and other classes, as well as techniques. fit(),.predict(),.score(), and so on are only a few examples.

Cost Function in the Linear Regression

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:06 PM
Book:	Cost Function in the Linear Regression		

Table of contents

1. Cost Function in the Linear Regression

1. Cost Function in the Linear Regression

The **cost function**, also known as the loss function, is the function that may be reduced (or increased) by changing the decision variables [48]. Under the surface, many machine learning approaches handle optimization challenges. They usually alter the model parameters to reduce the difference between actual and expected outcomes (like weights and biases for neural networks, decision rules for random forest or gradient boosting, and so on).

In a regression problem, the vectors of input variables x = (x_1, ... x_r) and the actual outputs y are commonly used. You're looking for a model that relates x to a predicted response f(x) that is as close to y as feasible. For example, you could want to forecast a person's income based on inputs such as their number of years at the organization or their degree of schooling.

The objective is to keep the difference between the prediction *f*(x) and the actual data y as little as possible. The residual is the term for this difference.

You aim to minimize the sum of squared residuals (SSR) in this kind of problem, where SSR = $\Sigma_i(y_i - f(x_i))^2$ for all observations i = 1, ... , n where n is the total number of observations. Instead of SSR, you might calculate the mean squared error (MSE = SSR / n).

The square of the difference between the actual and predicted outputs is used by both SSR and MSE. The smaller the difference, the more precise the forecast. When the difference is zero, then the prediction is equal to the actual data.

The model parameters are adjusted to minimize SSR or MSE. For instance, if you want to find the function $f(x) = b_0 + b_1x_1 + ... + b_rx_r$, in linear regression, you must first define the weights b_0, b_1, ... , b_r that minimize SSR or MSE.

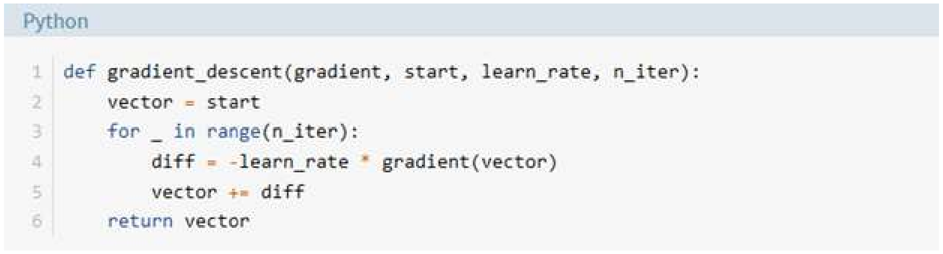
Consider a drop of water flowing down the edge of a bowl or a ball rolling down a hill to comprehend the gradient descent process. Until they reach the bottom, the drop and the ball tend to proceed in the direction of the quickest reduction. They will gain accelate and momentum.

Gradient descent works in a similar way: you start with an arbitrarily determined position of the point or vector v = (v_1, ..., v_r)and move it repeatedly in the direction of the cost function's quickest decline. This is the direction of the negative gradient vector, - \(\bigtriangledown \) C, as previously stated.

You update or relocate it to a new place in the direction of the negative gradient (v \(\rightarrow \) v – \(\eta \) \(\bigtriangledown \)C , where \(\eta \) (pronounced "ee-tah") is a small positive value called the learning rate, after you have a random starting point v = (v_1, ..., v_r).

The update or moving step size is determined by the learning rate. This is a crucial parameter. If \(\eta \) is too small, the method may take a long time to converge. Large \(\eta \) values can potentially cause problems with convergence or divergence in the method.

This is a simple version of the procedure that starts with an arbitrary position, pushes it toward the minimum repeatedly, and returns a location that is hopefully at or near the minimum:



Python

```python
def gradient_descent(gradient, start, learn_rate, n_iter):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        vector += diff
    return vector
```

gradient_descent() takes four arguments:

- **gradient** is the function or any Python callable object that takes a vector and returns the gradient of the function you're trying to minimize.
- **Start:** the point from which the algorithm starts its search is called the start
- **learn_rate:** is the learning rate that controls the magnitude of the vector update.
- **n_iter:** is the number of iterations.

This function does exactly what it says on the tin: it takes a beginning point (line 2), iteratively updates it based on the learning rate and gradient value (lines 3–5), and then returns the latest position obtained. You may add another termination condition before using gradient_descent():

Python

```python
 1  import numpy as np
 2
 3  def gradient_descent(
 4      gradient, start, learn_rate, n_iter=50, tolerance=1e-06
 5  ):
 6      vector = start
 7      for _ in range(n_iter):
 8          diff = -learn_rate * gradient(vector)
 9          if np.all(np.abs(diff) <= tolerance):
10              break
11          vector += diff
12      return vector
```

You now have the additional parameter tolerance (line 4), which specifies the minimal allowed movement in each iteration. You've also defined the default values for tolerance and n_iter, so you don't have to specify them each time you call gradient_descent().

Lines 9 and 10 enable gradient_descent() to stop iterating and return the result before n_iter is reached if the vector update in the current iteration is less than or equal to tolerance. This often happens near the minimum, where gradients are usually very small. Unfortunately, it can also happen near a local minimum or a saddle point.

Line 9 uses the convenient NumPy functions numpy.all() and numpy.abs() to compare the absolute values of diff and tolerance in a single statement. That's why you import numpy on line 1.

Now that you have the first version of gradient_descent(), it's time to test your function. You'll start with a small example and find the minimum of the function $C = v^2$.

The derivative 2v is the gradient of this function, which contains just one independent variable (v). It's a differentiable convex function, and finding its minimum analytically is simple. Analytic differentiation, on the other hand, may be difficult, if not impossible, in practice, and is frequently approximated with numerical approaches. To test your gradient descent implementation, you simply need one statement:

Python

```python
>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.2
... )
2.210739197207331e-06
```

You use the lambda function lambda v: 2 * v to provide the gradient of v^2. You start from the value 10.0 and set the learning rate to 0.2. You get a result that's very close to zero, which is the correct minimum.

The learning rate is a very important parameter of the algorithm. Different learning rate values can significantly affect the behavior of gradient descent. Consider the previous example, but with a learning rate of 0.8 instead of 0.2:

Python

```python
>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.8
... )
-4.77519666596786e-07
```

Small learning rates can result in very slow convergence. If the number of iterations is limited, then the algorithm may return before the minimum is found. Otherwise, the whole process might take an unacceptably large amount of time. To illustrate this, run gradient_descent() again, this time with a much smaller learning rate of 0.005:

```
Python                                                                  >>>

>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005
... )
6.050060671375367
```

The search process starts at v = 10 as before, but it can't reach zero in fifty iterations. However, with a hundred iterations, the error will be much smaller, and with a thousand iterations, you'll be very close to zero:

```
Python                                                                  >>>

>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005,
...     n_iter=100
... )
3.660323412732294
>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005,
...     n_iter=1000
... )
0.0004317124741065828
>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005,
...     n_iter=2000
... )
9.952518849647663e-05
```

Nonconvex functions might have local minima or saddle points where the algorithm can get trapped. In such situations, your choice of learning rate or starting point can make the difference between finding a local minimum and finding the global minimum.

Consider the function $v^4 - 5v^2 - 3v$. It has a global minimum in $v \approx 1.7$ and a local minimum in $v \approx -1.42$. The gradient of this function is $4v^3 - 10v - 3$. Let's see how gradient_descent() works here:

```
Python                                                                  >>>

>>> gradient_descent(
...     gradient=lambda v: 4 * v**3 - 10 * v - 3, start=0,
...     learn_rate=0.2
... )
-1.4207567437458342
```

During the first two iterations, your vector was moving toward the global minimum, but then it crossed to the opposite side and stayed trapped in the local minimum. You can prevent this with a smaller learning rate:

```
Python                                                                  >>>

>>> gradient_descent(
...     gradient=lambda v: 4 * v**3 - 10 * v - 3, start=0,
...     learn_rate=0.1
... )
1.285401330315467
```

A lower learning rate prevents the vector from making significant leaps, therefore it remains closer to the global optimum in this situation.

It's difficult to change the learning rate. You can't know the best value in advance. Many strategies and heuristics have been developed to assist with this. Furthermore, during model selection and assessment, machine learning practitioners frequently adjust the learning rate.

Aside from the learning rate, the starting point can have a substantial impact on the solution, especially for nonconvex functions.

Exercise 2: Outlier Detection

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:06 PM
Book:	Exercise 2: Outlier Detection		

Table of contents

1. General description

2. Desired objectives:

3. Required material

4. Other requirements

1. General description

Outlier detection is a critical tool in the use of machine learning approaches in advanced manufacturing. It enables the prompt discovery of irregularities, which can lead to cost savings for the particular firm. Furthermore, product quality control is an essential component of a product processing plant's manufacturing line. Machine learning technologies can improve quality control.

Assume you're a machine learning specialist who works for a huge winery. The management wants to improve the quality of its products by incorporating machine learning algorithms into the specific process. You are asked to put in place an acceptable model to accomplish this purpose using a dataset of old measurements of quality measurements of wine batches of the past years.

Instructions:

There are many outlier detection techniques in the literature that can be applied in different scenarios and different datasets. Finding the best one that is suitable for each case might be a challenging process that requires trying different techniques to find out which one is more fitting.

1. Download the csv file from the following link (password: dt@mml)

https://versions.aimms.gr/index.php/s/YLnsaDxw5YSEECz

2. Load the downloaded csv file (dataset3.csv) and apply different outlier detection techniques as the following scripts suggest.

3. Copy and paste the following notebook to your python IDE and follow the instructions to accomplice the exercise requirements. You should fill the missing parts of the exercise as dictated in the provided comments.

```
# ========================================================================
# Exercise 3 – OUTLIER DETECTION
# Z-SCORE, DBSCAN and VISUALIZATION TECHNIQUES
# Complete the missing code by implementing the necessary commands.
========================================================================
# For this project, you will need the NumPy library, the stats library, the pandas library, the
DBSCAN and the IsolationFprest from sklearn.cluster, the seaborn library, the pyplot from
matplotlib
# However, before importing it, you must first install the library into Python.
# Read the instructions on how to do that (it might be a bit trickier than usual!)
========================================================================
# IMPORT LIBRARIES HERE (trivial but necessary...)
import
import
import
from
from matplotlib import pyplot as plt
from statsmodels.graphics.gofplots import qqplot
# ========================================================================
# Load the 'wine' dataset (Dataset 3)
# ========================================================================
# ADD COMMAND TO LOAD TRAIN AND TEST DATA HERE TO DATAFRAME
wineData =
#======================================================================
# Now, let's try some outlier detection methods.
# Try all the methods that are presented below.
# ========================================================================
# Z-SCORE METHOD - Using Z score method,we can find out how many standard deviations
value away from the mean.
# If the z score of a data point is more than 3 (because it cover 99.7% of area), it indicates
that the data value is quite different from the other values. It is taken as outliers.

# ========================================================================
# ADD COMMANDS TO CONFIGURE THE LEARNER HERE
out=[]

def Zscore_outlier(df):
    m = np.mean(df)
    sd = np.std(df)
    for i in df:
        z = (i-m)/sd
        if np.abs(z) > 3:
            out.append(i)
    print("Outliers:",out)
Zscore_outlier(wineData['']) # YOU CAN TRY APPLY IT FOE DIFFERENT COLUMNS OF THE
DATASET OR THE WHOLE DATASET. WHAT DO YOU OBSERVE??

#
========================================================================
# DBSCAN METHOD (DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE)
# DBSCAN is a density-based clustering algorithm that divides a dataset into
# subgroups of high-density regions and identify high-density regions cluster as
# outliers. Here cluster -1 indicates that the cluster contains outliers and the
# rest of the clusters have no outliers. This approach is similar to the K-mean
# clustering. There are two parameters required for DBSCAN. DBSCAN gives the best
# result for multivariate outlier detection.

# 1. epsilon: a distance parameter that defines the radius to search for nearby neighbors.
# 2. The minimum amount of points required to form a cluster.

def DB_outliers(df):
    outlier_detection = DBSCAN(eps = 2, metric='euclidean', min_samples = 5)
    clusters = outlier_detection.fit_predict(df.values.reshape(-1,1))
    data['cluster'] = clusters
    print(data['cluster'].value_counts().sort_values(ascending=False))
DB_outliers(wineData['clorides']) # WHICH COLUMN OF THE DATASET WILL YOU CHOOSE?
#
============================================================================
# Visualization of the Data might be another outlier detection technique if you are using the
right tools.

# BEGIN WITH BOX PLOT
def Box_plots(df):
    plt.figure(figsize=(10, 4))
    plt.title("Box Plot")
    sns.boxplot(df)
    plt.show()
Box_plots(wineData['clorides'])
Box_plots(wineData['volatile acidity'])
# which other column would you choose to visualize with qqplot? What do you notice? Do
the outliers that you can see the same as the above ones?

def hist_plots(df):
    plt.figure(figsize=(10, 4))
    plt.hist(df)
    plt.title("Histogram Plot")
    plt.show()
hist_plots(wineData['clorides'])

def scatter_plots(df1,df2):
    fig, ax = plt.subplots(figsize=(10,4))
    ax.scatter(df1,df2)
    ax.set_xlabel('Age')
    ax.set_ylabel('Fare')
    plt.title("Scatter Plot")
    plt.show()
scatter_plots(wineData['free sulfur dioxide'], wineData['clorides'])
# This is for combining two different values

def qq_plots(df):
    plt.figure(figsize=(10, 4))
    qqplot(df,line='s')
    plt.title("Normal QQPlot")
    plt.show()
qq_plots(wineData['clorides'])
```

2. Desired objectives:

- Implement more complex machine learning models
- Deal with a real-life problem solution of machine learning
- Advanced plot making

3. Required material

For the execution of this task, you need to:

1. Install Python 3 Release according to your operation system.

- Windows
- macOS
- other

2. A python IDE is required or you can use Google Colab [1]

[1] Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

4. Other requirements

There are no other special requirements for the execution of this exercise.

Let’s get started

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:06 PM
Book:	Let’s get started		

Description

Python implements a number of complex machine learning methods. Simple k Nearest Neighbor (kNN), Decision Trees, Classification, Clustering, and Simple Vector Machine are the most prevalent (SVM). However the kNN and Decision trees algorithms are the most well-known and widespread algorithms with many applications in the developed industry and will be analyzed in detail.

Table of contents

1. Importing the abalone dataset

2. Descriptive statistics from the abalone dataset

3. Use a mathematical definition of distance to define "nearest"

1. Importing the abalone dataset

The Abalone Dataset will be used. You could download it and then use pandas to import it into Python, but it's far faster to let pandas handle it for you. You may use pandas to import the data as follows:

```python
>>> import pandas as pd
>>> url = (
...     "https://archive.ics.uci.edu/ml/machine-learning-databases"
...     "/abalone/abalone.data"
... )
>>> abalone = pd.read_csv(url, header=None)
```

You import pandas first, then use it to read the data in this code. If you make the path a URL, the file will be downloaded straight from the Internet. You may perform a fast check to ensure that you've imported the data correctly as follows:

```python
>>> abalone.head()
   0      1      2      3       4       5       6      7   8
0  M  0.455  0.365  0.095  0.5140  0.2245  0.1010  0.150  15
1  M  0.350  0.265  0.090  0.2255  0.0995  0.0485  0.070   7
2  F  0.530  0.420  0.135  0.6770  0.2565  0.1415  0.210   9
3  M  0.440  0.365  0.125  0.5160  0.2155  0.1140  0.155  10
4  I  0.330  0.255  0.080  0.2050  0.0895  0.0395  0.055   7
```

This should display the first five lines of the Abalone Dataset, which has been imported as a pandas DataFrame in Python. The column names are still lacking, as you can see. The abalone.names file in the UCI machine learning repository contains those names. You may include them in your DataFrame in the following way:

```python
>>> abalone.columns = [
...     "Sex",
...     "Length",
...     "Diameter",
...     "Height",
...     "Whole weight",
...     "Shucked weight",
...     "Viscera weight",
...     "Shell weight",
...     "Rings",
... ]
```

The imported data should now be easier to comprehend. However, there is one more thing you need do: eliminate the Sex column. The present experiment's purpose is to utilize physical measures to estimate the abalone's age. You should exclude sex from the dataset because it is not a strictly physical parameter. Using.drop:, you may remove the Sex column.

```python
>>> abalone = abalone.drop("Sex", axis=1)
```

2. Descriptive statistics from the abalone dataset

With this code, the Sex column is removed because it adds no value to the models.

The abalone dataset's descriptive statistics.

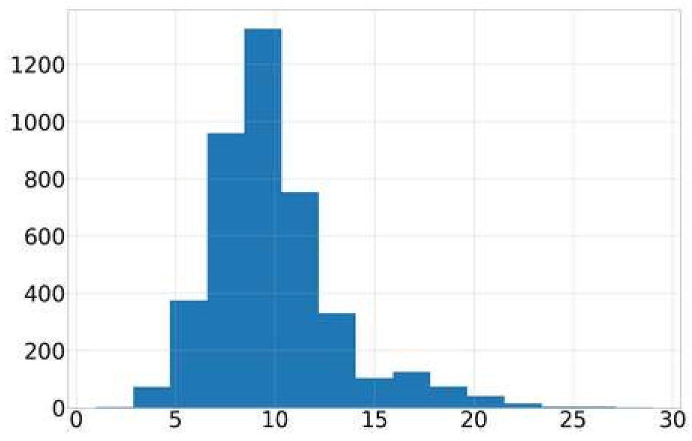
When it comes to machine learning, you need to know what kind of data you're dealing with. Here are some exploratory data and graphs, without getting into too much detail.

You can begin with Rings, which is the exercise's objective variable. A histogram will provide you with a fast and informative summary of the age ranges to expect:

Python

```python
>>> import matplotlib.pyplot as plt
>>> abalone["Rings"].hist(bins=15)
>>> plt.show()
```

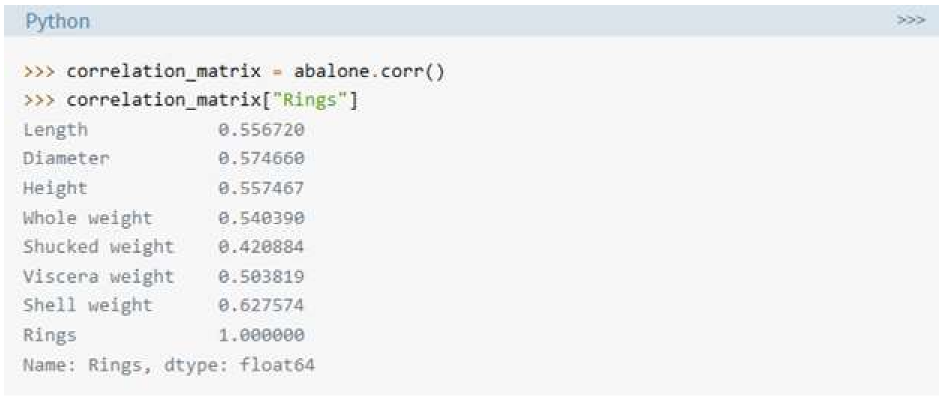
This code creates a fifteen-bin histogram using the pandas charting functionality. A few attempts led to the decision to employ fifteen bins. When deciding on the number of bins, you should aim for a balance of neither too many nor too few observations each bin. A histogram with too few bins may obscure certain patterns, whereas a histogram with too many bins may lack smoothness. The histogram may be seen in the graph below:



The histogram demonstrates that the majority of abalones in the sample had between five and fifteen rings, but that up to twenty-five rings are feasible. In this dataset, elder abalones are underrepresented. This makes sense, because age distributions are skewed in this way owing to natural processes.

A second worthwhile investigation is to see which factors, if any, have a strong relationship with age. A substantial connection between an independent variable and your aim variable is a positive indicator, since it confirms the relationship between physical measures and age.

In correlation_matrix, you may see the entire correlation matrix. The correlations with the target variable Rings are the most significant. You can get those correlations by doing something like this:



```python
>>> correlation_matrix = abalone.corr()
>>> correlation_matrix["Rings"]
Length            0.556720
Diameter          0.574660
Height            0.557467
Whole weight      0.540390
Shucked weight    0.420884
Viscera weight    0.503819
Shell weight      0.627574
Rings             1.000000
Name: Rings, dtype: float64
```

Examine the correlation coefficients between Rings and the other factors now. The closer they are to one, the stronger the link.

You may deduce that there is some relationship between physical measures of adult abalones and their age, although it isn't particularly strong. Because of the significant correlations, you may expect a simple modeling approach. In this scenario, you'll have to experiment with the kNN method to see what results you can get.

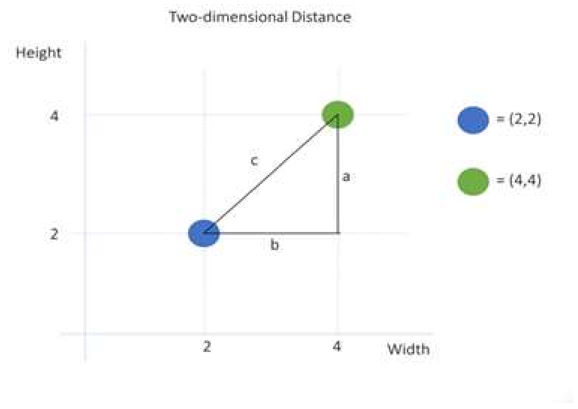
In comparison to other machine learning algorithms, the kNN algorithm is a little unusual. Each machine learning model has its own formula that must be calculated, as you saw before. The k-Nearest Neighbors approach is unique in that the formula is determined at the time of prediction rather than at the time of fitting. This is not the case with the majority of other models.

When a new data point is added, the kNN method begins by finding the new data point's nearest neighbors, as the name implies. After that, it utilizes the values of those neighbors as a forecast for the new data point.

Consider your neighbors as an intuitive illustration of why this works. Your neighbors are frequently similar to you. They're most likely from the same socioeconomic background as you. Perhaps they work in the same field as you, or their children attend the same school as yours, and so on. However, for particular activities, this method is ineffective. It wouldn't make sense, for example, to forecast your favorite color by looking at your neighbor's.

3. Use a mathematical definition of distance to define "nearest"

You can use a mathematical model of distance called Euclidean distance to locate the data points that are closest to the point you need to predict. To understand this concept, you must first understand what the term "difference of two vectors" means. See the following example:



Two data points are shown in this diagram: blue at (2,2) and green at (4,4). To find the distance between them, start by multiplying two vectors together. Vector a connects points (4,2) and (4,4), whereas vector b connects points (4,2) and (4,4) to point (2,2). The colorful spots on their heads denote their heads. They're at a 90-degree angle, as you can see.

This figure shows two data points: blue at (2,2) and green at (4,4). Start by multiplying two vectors together to calculate the distance between them.

The vector c, which runs from the head of vector a to the head of vector b, is the difference between these two vectors. The distance between your two data points is represented by the length of vector c.

The norm refers to the length of a vector. The vector's magnitude is indicated by the norm, which is a positive number. The Euclidean formula may be used to get the norm of a vector:

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2 + \cdots + (a_n - b_n)^2}$$

The distance is calculated using this formula by taking the squared differences in each dimension and then taking the square root of the sum of those values. To get the distance between the data points, you should compute the norm of the difference vector c.
To apply this to your data, you must first understand that the data points are vectors. The distance between them may then be calculated by determining the difference vector's norm. NumPy's linalg.norm() may be used to calculate this in Python. For an example:

```
Python                                                          >>>

>>> import numpy as np
>>> a = np.array([2, 2])
>>> b = np.array([4, 4])
>>> np.linalg.norm(a - b)
2.8284271247461903
```

In this code block, you define your data points as vectors. You then compute norm() on the difference between two data points. This way, you directly obtain the distance between two multidimensional points. Even though the points are multidimensional, the distance between them is still a scalar, or a single value.

Find the k nearest neighbors

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:07 PM
Book:	Find the k nearest neighbors		

Description

You may use this to identify the nearest neighbors of a point on which you want to make a prediction now that you know how to compute the distance from any point to any point.

You must determine the number of neighbors, which is provided by k. k has a minimum value of one. This signifies that only one neighbor will be used to make the prediction. The number of data points you have is the maximum. This entails utilizing all of your neighbors. The user is in charge of determining the value of k. As you'll see in the last section of this course, optimization tools can help you with this.

Return to the Abalone Dataset to identify the nearest neighbors in NumPy. As you can see, you'll need to set distances on the vectors of the independent variables, so use the .values attribute to convert your pandas DataFrame into a NumPy array:

```
Python                                                    >>>

>>> X = abalone.drop("Rings", axis=1)
>>> X = X.values
>>> y = abalone["Rings"]
>>> y = y.values
```

This code block creates two objects: X and y, each of which now has your data. Your model's independent variables are X and y, respectively. It's worth noting that X is written with a capital letter, whereas y is written with a lowercase letter. Because mathematical notation utilizes a capital letter for matrices and a lowercase letter for vectors, this is frequently done in machine learning programs. This is how you can make the NumPy array for this data point:

```
Python                                                    >>>

>>> new_data_point = np.array([
...     0.569552,
...     0.446407,
...     0.154437,
...     1.016849,
...     0.439051,
...     0.222526,
...     0.291208,
... ])
```

The next step is to use the following code to calculate the distances between this new data point and each of the data points in the Abalone Dataset:

```
Python                                                    >>>

>>> distances = np.linalg.norm(X - new_data_point, axis=1)
```

So now, you have a vector of distances and must determine which three neighbors are the closest. You'll need to find the IDs of the minimum distances to do this. You may sort the array from lowest to highest using the .argsort() function, and you can get the indices of the k closest neighbors by taking the first k elements:

```
Python                                                    >>>

>>> k = 3
>>> nearest_neighbor_ids = distances.argsort()[:k]
>>> nearest_neighbor_ids
array([4045, 1902, 1644], dtype=int32)
```

This informs you which of your new data point's three neighbors are the closest. You'll learn how to transform those neighbors into an estimate in the next paragraph.

Table of contents

1. Voting or averaging of multiple neighbors

1.1. Average for regression

1.2. Mode for classification

2. Fitting kNN in Python using scikit-learn

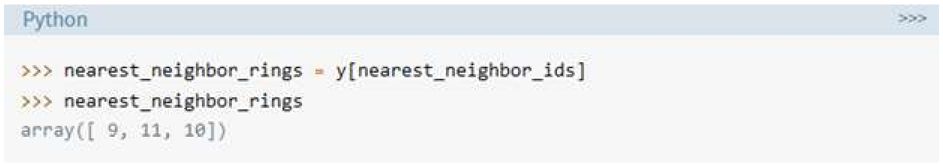
2.1. Using scikit-learn to fit a kNN regression on the abalone dataset

2.2. Examining the model fit using scikit-learn

3. kNN Algorithm in Python – The Model to Predict Abalone Age

1. Voting or averaging of multiple neighbors

After you've determined the indices of your abalone's three closest neighbors, you'll need to integrate them into a prediction for your new data point. As a starting step, you must gather the following facts about those three neighbors:

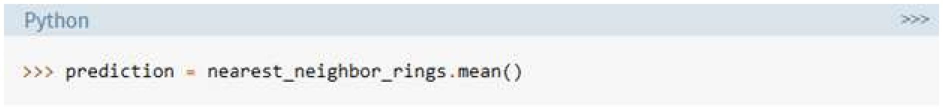


```
>>> nearest_neighbor_rings = y[nearest_neighbor_ids]
>>> nearest_neighbor_rings
array([ 9, 11, 10])
```

You'll integrate the values for those three neighbors into a prediction for your new data point now that you have them. For regression and classification, combining the neighbors into a prediction works differently.

1.1. Average for regression

The target variable in regression issues is a number. By averaging the target variable values of numerous neighbors, you may make a single forecast. You can do so by following these steps:


Python

```
>>> prediction = nearest_neighbor_rings.mean()
```

For prediction, you'll earn a score of ten. This suggests that your new data point's 3-Nearest Neighbor prediction is 10. You may repeat the process for as many new abalones as you wish.

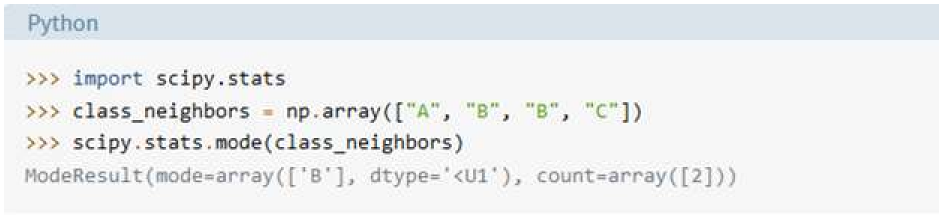
1.2. Mode for classification

The target variable in classification tasks is categorical. As previously stated, categorical variables cannot be averaged. What would the average of three anticipated automobile manufacturers be, for example? It'd be impossible to say that. On class predictions, you can't use an average.

In the case of categorization, you should instead use the mode. The mode is the value that appears the most frequently. This means that you count all of your neighbors' courses and keep the most common one. The value that appears the most frequently among the neighbors is the prediction.

There are various solutions if there are different modes. You might choose a final winner from among the winners at random. You might alternatively base your final selection on the distances between neighbors, in which case the nearest neighbors' mode would be preserved.

The SciPy mode() function may be used to calculate the mode. Because the abalone example isn't a classification case, the following code demonstrates how to compute the mode for a toy example:



Python

```
>>> import scipy.stats
>>> class_neighbors = np.array(["A", "B", "B", "C"])
>>> scipy.stats.mode(class_neighbors)
ModeResult(mode=array(['B'], dtype='<U1'), count=array([2]))
```

2. Fitting kNN in Python using scikit-learn

While learning to code an algorithm from scratch is beneficial, it is rarely practical while working on a machine learning problem. In this part, you'll learn how to utilize scikit-learn, one of Python's most extensive machine learning tools, to build the kNN method.

Your abalone kNN model's quality will be assessed. You had a technical focus in the previous parts, but now you'll take a more pragmatic and results-oriented approach. There are other methods for assessing models, but the train-test split is the most frequent. When evaluating a model with a train-test split, you divide the dataset into two parts:

1. **Training data** : The model is fitted using training data. This signifies that the training data will be utilized as neighbors in the kNN algorithm.

2. **Test data:** The model is evaluated using test data. It implies you'll estimate the number of rings on each abalone in the test data and compare those estimates to the known real number of rings.

The data may be divided into training and test sets in Python using scikit-learn's built-in train_test_split():

Python >>>

```python
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.2, random_state=12345
... )
```

The test_size refers to the amount of observations you intend to include in both the training and test data sets. When you choose a test_size of 0.2, your test_size will be 20% of the original data, leaving the remaining 80% as training data.

The random_state parameter allows you to get consistent results every time you execute the code. The train test split() function creates a random split in the data, which makes it difficult to reproduce the findings. As a result, random state is frequently used. In random state, the value is chosen at random.

The data is divided into training and test data in the code above. For objective model evaluation, this is required. You can now use scikit-learn to fit a kNN model to the training data.

2.1. Using scikit-learn to fit a kNN regression on the abalone dataset

To fit a scikit-learn model, you must first create a model of the right class. You must also select settings for your hyperparameters at this time. You must choose a value for k in the kNN algorithm, which is denoted n neighbors in the scikit-learn implementation. This is how you do it in Python:

Python

```
>>> from sklearn.neighbors import KNeighborsRegressor
>>> knn_model = KNeighborsRegressor(n_neighbors=3)
```

With knn model, you may make an unfitted model. To predict the value of a future data point, this model will use the three closest neighbors. You can next fit the model on the training dataset to get the data into the model:

Python

```
>>> knn_model.fit(X_train, y_train)
```

You may let the model learn from the data by using .fit(). kNN model now has everything necessary to generate predictions on new abalone data points. That's all the Python code you'll need to fit a kNN regression.

2.2. Examining the model fit using scikit-learn

However, simply fitting a model isn't enough. In this part, you'll learn about some of the functions that may be used to assess the fit. There are a variety of regression assessment metrics available, but you'll choose one of the most prevalent, the root-mean-square error (RMSE). The following formula is used to calculate the RMSE of a prediction:

1. Calculate the difference between the actual and predicted values for each data point.
2. Take the square of the difference for each difference.
3. Add all of the squared differences
4. Calculate the square root of the total sum.

To begin, assess the prediction error using the training data. This implies that you utilize the training data to make a prediction, so you know the outcome will be acceptable. The RMSE may be calculated using the following code:

```python
>>> from sklearn.metrics import mean_squared_error
>>> from math import sqrt
>>> train_preds = knn_model.predict(X_train)
>>> mse = mean_squared_error(y_train, train_preds)
>>> rmse = sqrt(mse)
>>> rmse
1.65
```

Using the knn_model that you fitted in the previous code block, you compute the RMSE in this code. For the time being, you'll compute the RMSE using the training data. You should assess the performances on data that isn't included in the model for a more realistic outcome. As a result, you separated the test set for the time being. With the same function as previously, you can assess the predicted performance on the test set:

```python
>>> test_preds = knn_model.predict(X_test)
>>> mse = mean_squared_error(y_test, test_preds)
>>> rmse = sqrt(mse)
>>> rmse
2.37
```

This code block evaluates the error on data that the model hasn't seen yet. This RMSE is somewhat greater than previously since it is more realistic. Because the RMSE represents the average inaccuracy of the predicted age, you may interpret this as an error of 1.65 years on average. It's difficult to say if an improvement from 2.37 to 1.65 years is significant. At the very least, you're coming closer to approximating the age appropriately.

You've only used the scikit-learn kNN algorithm out of the box up until now. You haven't done any hyperparameter adjustment and have chosen k at random. Between the RMSE on the training data and the RMSE on the test data, there is a significant difference. This indicates that the model is overfitted to the training data and so does not generalize well. At this stage, there's no need to be concerned.

3. kNN Algorithm in Python – The Model to Predict Abalone Age

Worthy of mention in the abalone dataset. Many abalones' ages are recorded in this dataset. Abalones are small sea snails that resemble mussels in appearance. Cutting an abalone's shell and counting the number of rings on the shell can reveal its age. Many abalones' ages, as well as a variety of other physical measurements, may be found in the Abalone Dataset.

The project's objective is to create a model that can estimate an abalone's age only based on other physical measures. Researchers would be able to measure the age of the abalone without having to break its shell and count the rings. To determine the most accurate prediction score, you'll use a kNN.

The same steps could be **applied to the manufacturing world** and for example predict the age of battery and therefore its remaining useful lifetime.

Decision Trees in Python

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:07 PM
Book:	Decision Trees in Python		

Table of contents

1. Reading the data and importing the libraries

2. Data processing

3. Modelling the tree and testing it

4. Visualizing the decision tree

1. Reading the data and importing the libraries

Libraries and classes that are necessary must be imported [59].

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline#for encoding
from sklearn.preprocessing import LabelEncoder#for train test splitting
from sklearn.model_selection import train_test_split#for decision tree object
from sklearn.tree import DecisionTreeClassifier#for checking testing results
from sklearn.metrics import classification_report, confusion_matrix#for visualizing tree
from sklearn.tree import plot_tree
```

Load the dataset now. The IRIS dataset may also be found in the seaborn library. You may use the following command to import it:

```
#reading the data
df = sns.load_dataset('iris')
df.head()
```

You should be able to obtain the information shown above. With one target column species, we have four feature columns: sepal_length, sepal_width, petal_length, and petal_width. Now go ahead and do some simple operations on it.

```
#getting information of dataset
df.info()
```

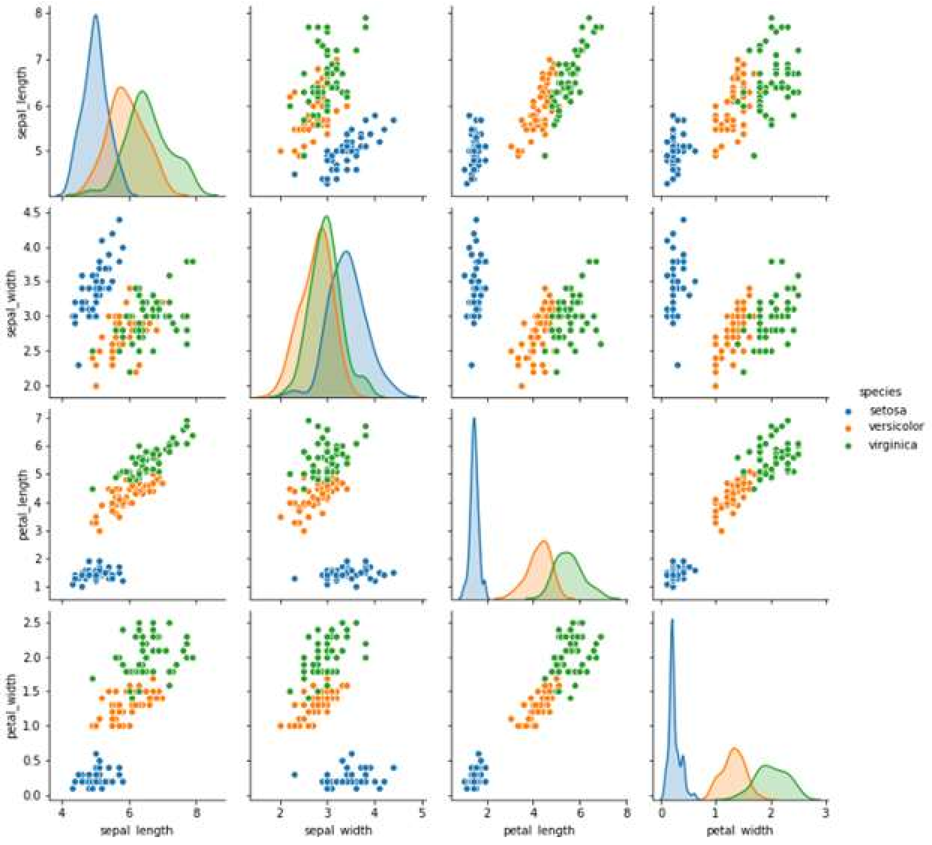
```
df.shape
```

According to the following command, this dataset comprises 150 records, 5 columns, the first four of which are of type float and the last of which is of type object str, and there are no NAN values:

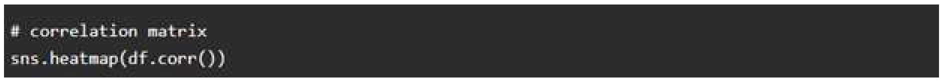
```
df.isnull().any()
```

On this dataset, we now run some basic EDA. Let's look at how all of the aspects relate to one another:

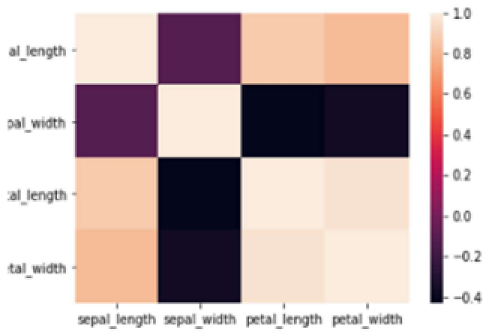
```
# let's plot pair plot to visualise the attributes all at once
sns.pairplot(data=df, hue = 'species')
```



Setosa, versicolor, and virginica are the three species we aim to predict. Setosa always creates a distinct cluster than the other two, as can be seen.



```
# correlation matrix
sns.heatmap(df.corr())
```



The following is what we can see from the two graphs above:

· Setosa generates a unique cluster every time.

· Petal length and width are inextricably linked.

· Sepal length is not related to sepal width.

2. Data processing

The target variable (y) and features(X) will now be separated as follows:

```
target = df['species']
df1 = df.copy()
df1 = df1.drop('species', axis =1)
```

It's best not to remove or add additional columns from the original dataset. Make a copy of it and then edit it so that if things don't go as planned, we have the original data with which to restart with a new strategy. We are storing df in X only to follow a well accepted norm.

```
# Defining the attributes
X = df1
```

Let's have a look at our target variable now. We will encode categorical variables in numeric values for working with the target.

```
#label encoding
le = LabelEncoder()
target = le.fit_transform(target)
target
```

Setosa:0, versicolor:1, virginica:2 are the encodings we obtain. Target is renamed to y in order to adhere to the normal naming practice.

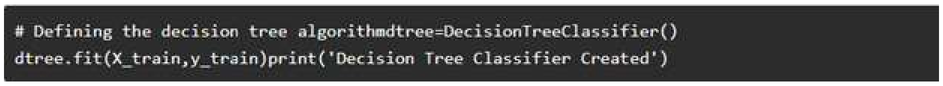
```
y = target
```

Creating training and testing sets from the dataset. 20 percent of the records were chosen at random for testing.

```
# Splitting the data - 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size = 0.2,
random_state = 42)print("Training split input- ", X_train.shape)
print("Testing split input- ", X_test.shape)
```

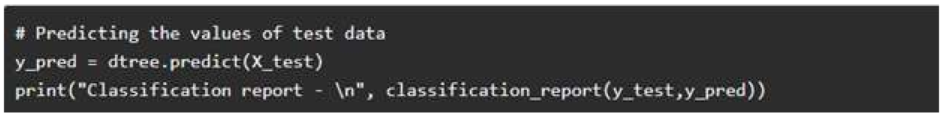
We have 120 records (rows) for training and 30 records (rows) for testing after splitting the dataset.

3. Modelling the tree and testing it



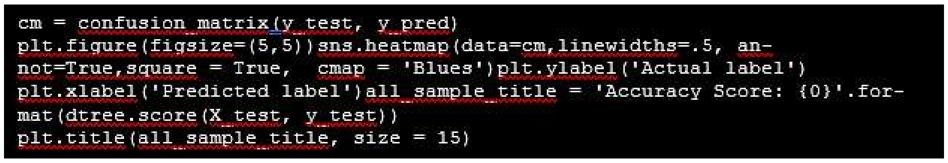
```
# Defining the decision tree algorithmdtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)print('Decision Tree Classifier Created')
```

We generated an object of the type DecisionTreeClassifier in the preceding code and stored its address in the variable dtree so that we could access it using dtree. Then we use our X train and y train to suit this tree. Finally, the statement Decision Tree Classifier is printed. After the decision tree has been constructed, this object is created.



```
# Predicting the values of test data
y_pred = dtree.predict(X_test)
print("Classification report - \n", classification_report(y_test,y_pred))
```

On a test dataset of 30 records, we achieved a 100% accuracy rate. Let's draw the confusion matrix like this.

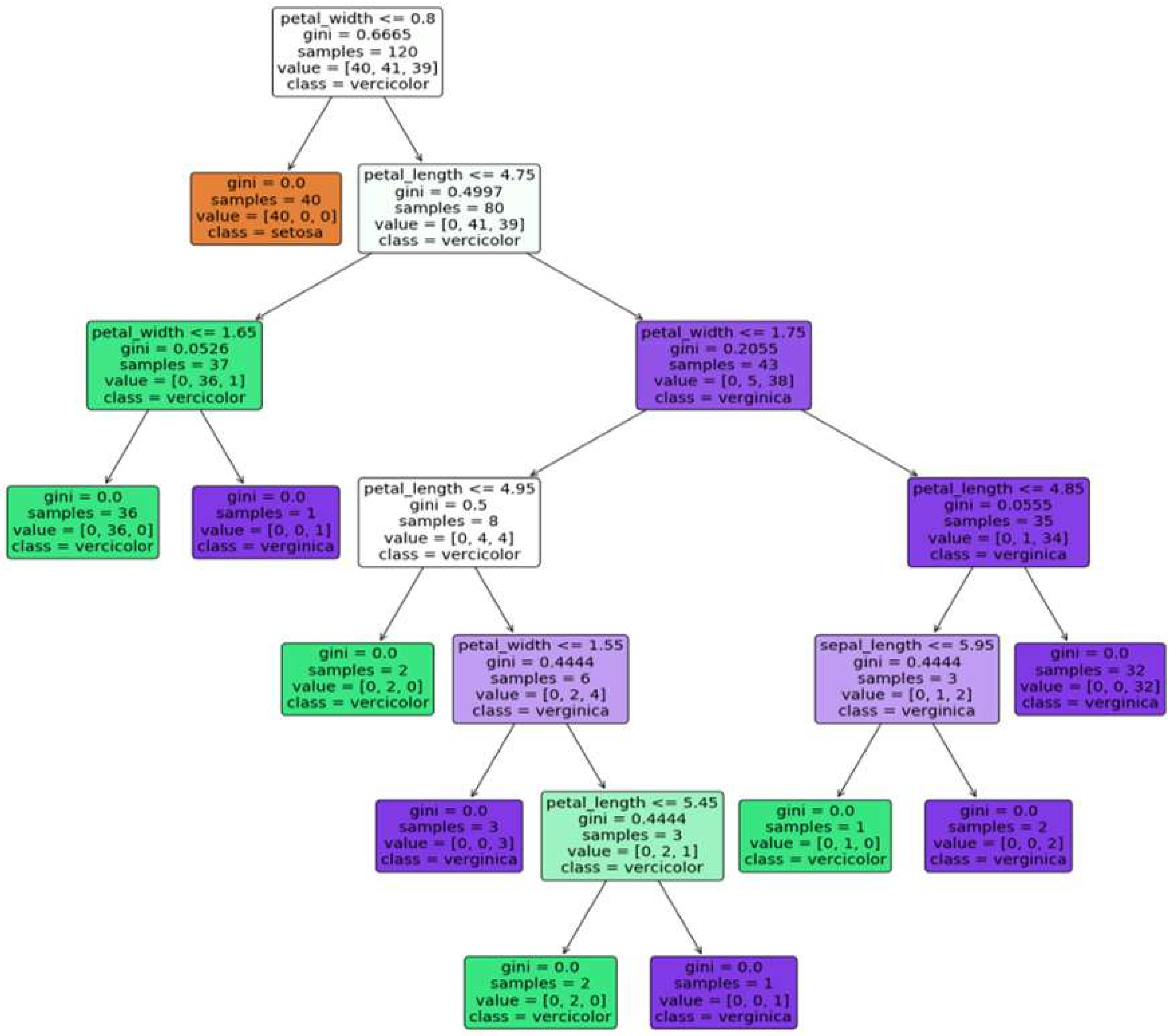


```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))sns.heatmap(data=cm,linewidths=.5, an-
not=True,square = True,  cmap = 'Blues')plt.ylabel('Actual label')
plt.xlabel('Predicted label')all_sample_title = 'Accuracy Score: {0}'.for-
mat(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

4. Visualizing the decision tree

Using the following instructions, we can plot the tree we created directly:

```
# Visualising the graph without the use of graphvizplt.figure(figsize =

dec_tree = plot_tree(decision_tree=dtree, feature_names = df1.columns,
                     class_names =["setosa", "vercicolor", "verginica"] ,
filled = True , precision = 4, rounded = True)
```



Comparison of Decision Trees and KNN

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:07 PM
Book:	Comparison of Decision Trees and KNN		

Description

Table of contents

1. Decision Trees vs KNN

1. Decision Trees vs KNN

In this section a comparison between the 2 algorithms is given [50].

A. They're both non-parametric. As a result, the data distribution cannot be determined using only a few parameters. In other words, decision trees and KNNs make no assumptions about the data distribution.

B. Both may be used to solve problems involving regression and classification.

C. KNN does not provide automated feature interaction, although decision trees do.

D. Although decision trees are faster, KNN is slower with large datasets since it scans the entire dataset to predict and does not generalize the data beforehand.

Exercise 3: Decision Trees and RF classifier

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Machine Learning	Date:	Friday, 8 December 2023, 4:08 PM
Book:	Exercise 3: Decision Trees and RF classifier		

Table of contents

1. General description

2. Desired objectives

3. Required material

4. Other requirements

1. General description

Fruitex Inc. is a fruit processing and packaging factory. In one of the production lines, which processes peaches, the sorting of the fruit into acceptable or not is done by humans. The company's management decided to automate the process and use the benefits of machine learning to sort the fruit, reducing both the chance of error and the production costs in the long run. You, as an expert in machine learning and an employee of Fruitex Inc., analyzed the data and concluded that the variables related to the quality of the fruit are directly related to its image, and for this reason, you proposed the installation of cameras in the production line that will return the necessary information. With the help of fruit quality experts, you created a dataset to use to train a machine learning model.

Instructions:

1. Download the csv file from the following link (password: dt@mml)

https://versions.aimms.gr/index.php/s/0BeKYYIj6MXScc8

2. Load the downloaded csv file (dataset2.csv) and apply a Decision Tree algorithm by testing different values of the split function (criterion) and max depth parameters. The code to output the model, as defined by the metric Accuracy, Precision, Recall, F1, and create a tree plot.

3. Apply the Random Forest algorithm by testing different parameter values split function (criterion) and the number of trees (n_estimators). The code to output the model, as defined by the metric Accuracy, Precision, Recall, F1. In addition, to create 4 graphs (one for each metric), which will show the performance of the model (y-axis) as it changes the number of trees in a Random Forest model (x-axis, from 1 to 200 trees).

Indicative results from the experiments are to be recorded in the table below.

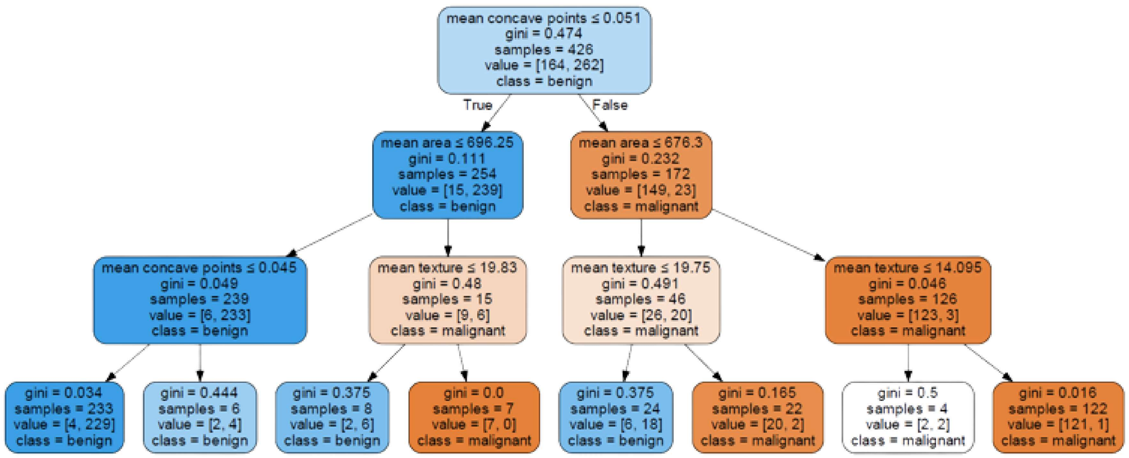
a/a	Algorithm	Criterion	Max depth	Accuracy	Precision	Recall	F1
1	Decision Tree						
2	Decision Tree						
3	Decision Tree						
4	Decision Tree						
5	Decision Tree						
6	Decision Tree						

a/a	Algorithm	Number of Estimators	Criterion	Accuracy	Precision	Recall	F1
1	Random Forest						
2	Random Forest						
3	Random Forest						
4	Random Forest						
5	Random Forest						
6	Random Forest						
a/a	Algorithm	Number of Estimators	Criterion	Accuracy	Precision	Recall	F1

1	Random Forest						
2	Random Forest						
3	Random Forest						
4	Random Forest						
5	Random Forest						
6	Random Forest						

4. Copy and paste the following notebook to your python IDE and follow the instructions to accomplice the exercise requirements. You should fill the missing parts of the exercise as dictated in the provided comments.

```
# ==============================================================================
# Exercise 2a - DECISION TREES
# DECISION TREE ALGORITHM TEMPLATE
# Complete the missing code by implementing the necessary commands.
# ==============================================================================

# From sklearn, we will import:
# 'metrics' package, for measuring scores
# 'tree' package, for creating the DecisionTreeClassifier and using graphviz
# 'model_selection' package, which will help test our model.
# =============================================================================


# IMPORT NECESSARY LIBRARIES HERE
import pandas as pd

from sklearn import


# Load the data
# ==============================================================================

# ADD COMMAND TO LOAD DATA HERE
fruits = pd.read_csv('../input/dataset2.csv') # change the path according to the location

# ==============================================================================
# Get samples from the data, and keep only the features that you wish.
# Decision trees overfit easily from with a large number of features! Don't be greedy.
numberOfFeatures = 10
X = fruits.data[:, :numberOfFeatures]
y = fruits.target


# DecisionTreeClassifier() is the core of this script. You can customize its functionality
# in various ways, but for now simply play with the 'criterion' and 'maxDepth' parameters.
# 'criterion': Can be either 'gini' (for the Gini impurity) and 'entropy' for the information gain.
# 'max_depth': The maximum depth of the tree. A large depth can lead to overfitting, so start with
a maxDepth of
#            e.g. 3, and increase it slowly by evaluating the results each time.
# ==============================================================================

# ADD COMMAND TO CREATE DECISION TREE CLASSIFIER MODEL HERE
 model =

# =================================================================================
# The function below will split the dataset that we have into two subsets. We will use
# the first subset for the training (fitting) phase, and the second for the evaluation phase.
# By default, the train set is 75% of the whole dataset, while the test set makes up for the rest 25%.
x_train, x_test, y_train, y_test = model_selection.train_test_split(X, y)


# Let's train our model.
#
==================================================================================
# ADD COMMAND TO TRAIN YOUR MODEL HERE

# =================================================================================
# Ok, now let's predict the output for the test input set
# ==============================================================================
# ADD COMMAND TO MAKE A PREDICTION HERE
y_predicted =

# ============================================================================


# Time to measure scores. We will compare predicted output (from input of x_test)
# with the true output (i.e. y_test).
# You can call 'recall_score()', 'precision_score()', 'accuracy_score()', 'f1_score()' or any other
available metric
# from the 'metrics' library.
# The 'average' parameter is used while measuring metric scores to perform a type of averaging
on the data.
# ============================================================================


# ADD COMMANDS TO EVALUATE YOUR MODEL HERE (AND PRINT ON CONSOLE)
print()
print()
print()
print()

# ============================================================================

# By using the 'plot_tree' function from the tree classifier we can visualize the trained model.
# There is a variety of parameters to configure, which can lead to a quite visually pleasant result.
# Make sure that you set the following parameters within the function:
# feature_names = fruits.feature_names[:numberOfFeatures]
# class_names = fruits.target_names
# filled = True
# ============================================================================

model.plot_tree
```



```
#
================================================================================

# Exercise 2b - DECISION TREES

# RANDOM FOREST ALGORITHM TEMPLATE

# Complete the missing code by implementing the necessary commands.

#
================================================================================



# From sklearn, we will import:

# 'datasets', for our data

# 'metrics' package, for measuring scores

# 'ensemble' package, for calling the Random Forest classifier

# 'model_selection', (instead of the 'cross_validation' package), which will help validate our
results.

#
================================================================================

# IMPORT NECESSARY LIBRARIES HERE

from sklearn import




#
================================================================================

# Load the data

#
================================================================================



# ADD COMMAND TO LOAD DATA HERE

fruits =

#
================================================================================

# Get samples from the data, and keep only the features that you wish.

# Decision trees overfit easily from with a large number of features! Don't be greedy.

numberOfFeatures = 10

X = fruits.data[:, :numberOfFeatures]

y =


# Split the dataset that we have into two subsets. We will use
# the first subset for the training (fitting) phase, and the second for the evaluation phase.

# By default, the train set is 75% of the whole dataset, while the test set makes up for the rest
25%.

# This proportion can be changed using the 'test_size' or 'train_size' parameter.

# Also, passing an (arbitrary) value to the parameter 'random_state' "freezes" the splitting
procedure

# so that each run of the script always produces the same results (highly recommended).

# Apart from the train_test_function, this parameter is present in many routines and should be

# used whenever possible.

x_train, x_test, y_train, y_test =



# RandomForestClassifier() is the core of this script. You can call it from the 'ensemble' class.

# You can customize its functionality in various ways, but for now simply play with the 'criterion'
and 'maxDepth' parameters.

# 'criterion': Can be either 'gini' (for the Gini impurity) and 'entropy' for the Information Gain.

# 'n_estimators': The number of trees in the forest. The larger the better, but it will take longer to
compute. Also,

#                 there is a critical number after which there is no significant improvement in the results

# 'max_depth': The maximum depth of the tree. A large depth can lead to overfitting, so start
with a maxDepth of

#               e.g. 3, and increase it slowly by evaluating the results each time.

#
================================================================================


# ADD COMMAND TO CREATE RANDOM FOREST CLASSIFIER MODEL HERE

 model =


#
================================================================================

# Let's train our model.

#
================================================================================

# ADD COMMAND TO TRAIN YOUR MODEL HERE



#
================================================================================
# Ok, now let's predict the output for the test set

#
==============================================================================

# ADD COMMAND TO MAKE A PREDICTION HERE

y_predicted =



#
==============================================================================

# Time to measure scores. We will compare predicted output (from input of the second subset,
i.e. x_test)

# with the real output (output of the second subset, i.e. y_test).

# You can call 'accuracy_score', 'recall_score', 'precision_score', 'f1_score' or any other
available metric

# from the 'sklearn.metrics' library.

# The 'average' parameter is used while measuring metric scores to perform a type of averaging
on the data.

# One of the following can be used for this example, but it is recommended that 'macro' is used
(for now):

# 'micro': Calculate metrics globally by counting the total true positives, false negatives, and
false positives.

# 'macro': Calculate metrics for each label, and find their unweighted mean. This does not take
label imbalance into account.

# 'weighted': Calculate metrics for each label, and find their average weighted by support (the
number of true instances for each label).

#            This alters 'macro' to account for label imbalance; it can result in an F-score that is not
between precision and recall.

#
==============================================================================

# ADD COMMANDS TO EVALUATE YOUR MODEL HERE (AND PRINT ON CONSOLE)

print()

print()

print()

print()

#
==============================================================================

# A Random Forest has been trained now, but let's train more models,

# with a different number of estimators each, and plot performance in terms of

# the different metrics. In other words, we need to make 'n'(e.g. 200) models,

# evaluate them on the aforementioned metrics, and plot 4 performance figures
# (one for each metric).

# In essence, the same pipeline as previously will be followed.

#
==========================================================================

# CREATE MODELS AND PLOTS HERE

#
==========================================================================
```

2. Desired objectives

- Implement one of the most famous machine learning models
- Deal with a real-life problem solution of machine learning
- Advanced plot making

3. Required material

For the execution of this task, you need to:

1. Install Python 3 Release according to your operation system.

- Windows
- macOS
- other

2. A python IDE is required or you can use Google Colab [1]

---

[1] Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

4. Other requirements

There are no other special requirements for the execution of this exercise.

Co-funded by the
Erasmus+ Programme
of the European Union

DTAM
DIGITAL TRANSFORMATION IN ADVANCED MANUFACTURING

DIGITAL TRANSFORMATION IN ADVANCED MANUFACTURING

Machine Learning Challenge

Deliverable factsheet

Project Number:	**621496-EPP-1-2020-1-ES-EPPKA2-SSA**
Project Acronym:	**DTAM**
Project Title:	**Digital Transformation in Advanced Manufacturing**
Work package:	**WP3: DTAM Training Course**
Title of Deliverable:	**DTAM Machine Learning via Big Data Techniques (1st version)**
Editor(s):	
Reviewer(s):	

Delivery Slip

Version	Date	Comments
1.0	09/09/2022	Outlines of the Deliverable
1.1	19/10/2022	Revision

Contents

1 Machine Learning Project 4

1.1 Presentation of the project 4

1.1.1 Introduction 4

1.1.2 The Data 4

1.1.3 Objective and methodology 4

1.1.4 Deliverables 6

1.2 Group challenge 7

1.2.1 Project Logistics 7

1.2.2 Roles 7

1.3 Goals to reach with the challenge 8

1.4 Rubric (evaluation form) for groups of students 9

2 Learning Units reference 11

3 Learning Outcomes 11

4 IoT Lab Resources 13

4.1 Requirements 13

5 The project step by step 14

6 Expected result example 15

Co-funded by the Erasmus+ Programme of the European Union

1 Machine Learning Project

1.1 Presentation of the project

This section outlines the project's structure, the data provided, the objective, and the technique suggested to be used.

1.1.1 Introduction

Prompted by the current energy crisis as well as wanting to follow new tactics on sustainability and green technologies, an industrial construction company wants to adopt techniques and materials that will reduce the energy footprint of the buildings it constructs. To achieve this the company collected data from already existing buildings in order to analyze them and draw conclusions about the heating and cooling load. **This information will let them estimate the required heating and cooling units for each industrial unit in order to avoid the installation of either unnecessary big units or an inadequate amount of them.**

1.1.2 The Data

The data that the leading architect suggested should be the most informative are the following eight variables:

Relative Compactness | Surface Area | Wall Area | Roof Area | Overall Height

Orientation | Glazing Area | Glazing Area distribution

And the target variables will be:

Heating Load | Cooling Load

! You can download the dataset here.

1.1.3 Objective and methodology

Describing the real-world problem

The company decided to hire you and your team as machine learning experts to perform analysis via suitable techniques and present the results to the board. Your analysis should be complete and the steps you choose should be justified.
Such an analysis is especially important because it actively contributes to an industry's ecological goal. Your main objective is to recommend a precise number of heating and cooling units per new building, so that a building maintains livable conditions while consuming no more energy than necessary.

Co-funded by the Erasmus+ Programme of the European Union

From real world to ML approach

To estimate the quantity of cooling and heating units needed by the firm, it is required to know the heating and cooling load that each building demands during its operation. To accomplish this purpose, you must first examine the data presented to you and how it links to the target data. In other words, we must find a way to predict the target data considering only the given data.

Because the situation described above is a real-world problem, it encompasses all of the obstacles that may arise in such a scenario. Some of these issues may include dealing with unfamiliar data, data that is not homogeneous, or data that displays strange values with no obvious explanation.
Inadequate data knowledge may be addressed by considering that the main thing is not necessary to comprehend the substance of a technical word, but to understand its behavior as data and how it might interact with other data. To do this, we first select to visualize the data before examining it with tools for discovering correlations between variables.
But first, we must address the issue of data inhomogeneity by doing the necessary data preprocessing operations. If we don't, we risk drawing incorrect conclusions or, worse, ones that we can't comprehend.
Then, based on our understanding, we will then be asked to select a suitable machine learning model. In our scenario, we want to forecast heating and cooling load values that are real continuous numbers. Will supervised or unsupervised learning be used?

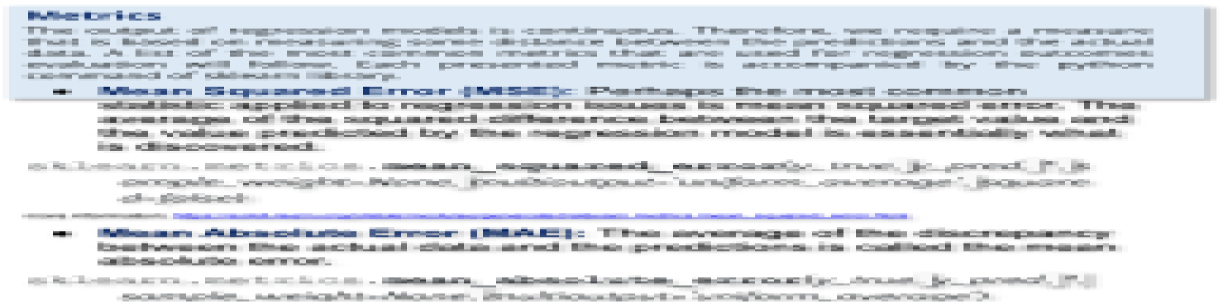
ML methodology steps

! Tip: A good ML specialist may use more than one models.

Afterwards, we must evaluate our findings. This manner, we can see if our projections are accurate and enhance our word and the company's faith in our exports. To do this, we must select an appropriate metric and apply it to our outcome. The following page contains a list of relevant metrics for forecasting real numbers.

! Tip: A good ML specialist never use one metric.

Finally, keep in mind that all of your labor should be documented and accompanied by proper explanations for each stage.



Metrics
The output of regression models is continuous. Therefore, we require a measure that is based on measuring some distance between the predictions and the actual data. A list of the most common metrics that are used for regression outcomes evaluation will follow. Each presented metric is accompanied by the python command of sklearn library.

- **Mean Squared Error (MSE):** Perhaps the most common statistic applied to regression issues is mean squared error. The average of the squared difference between the target value and the value predicted by the regression model is essentially what is discovered.

sklearn.metrics.mean_squared_error(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average', squared=True)

See more: [illegible]

- **Mean Absolute Error (MAE):** The average of the discrepancy between the actual data and the predictions is called the mean absolute error.

sklearn.metrics.mean_absolute_error(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average')

1.1.4 Deliverables

Summarizing all of the above, your deliverables should include the following:

1. Python Notepad with all the processes that you followed.
2. Brief report that documents all the steps that you followed including:
   a. Preprocessing of the data
   b. Data understanding and exploration
   c. Data visualization
   d. Data modeling
   e. Evaluation of the results

1.2 Group challenge

This section contains project logistics as well as a discussion of possible responsibilities for participants.

1.2.1 Project Logistics

To do this challenge, you will need to form a group with 2-4 other students.

You will be instructed by a teacher on how to perform the challenge.

When your group is finished, the product and the process will be evaluated by a teacher using the evaluation rubric.

1.2.2 Roles

It is proposed that you split the distinct jobs to make the greatest use of your resources. More specifically, we propose the following possible roles:

The Data Analyst
The data analyst is in charge of pre-processing the data, visualizing it, and inferring the relationship between the variables.

The Model Maker
The model maker is in charge of locating and applying one or more suitable machine learning models to the data.

The Evaluator
The evaluator is in charge of supervising the selection of relevant techniques, identifying appropriate metrics and implementing them, and preparing the final report.

These roles are indicative and may change depending on the composition of your group. There can for example be two Model Makers or a team member may have more than one role.

1.3 Goals to reach with the challenge

- ✔ Interact with a real-world case scenario and deal with the difficulties that may occur.
- ✔ Understand the physical meaning of the data and how they may interact with each other.
- ✔ Visualize the data using common python tools.

Co-funded by the Erasmus+ Programme of the European Union

- ✔ Preprocess the data in order to prepare them for further analysis.
- ✔ Decide which machine learning model is more suitable for the given data and the wanted outcome.
- ✔ Build the model from scratch using Python tools.
- ✔ Train and test the model.
- ✔ Evaluate the outcomes of the chosen model using suitable metrics.
- ✔ Document your work.

-

1.4 Rubric (evaluation form) for groups of students

After completing the challenge, the instructor(s) will evaluate the work of the students. This rubric will be used to evaluate competencies and learning outcomes on a 5-point scale of: very poor, poor, average, good, and excellent.

Criterium	Very poor	Needs Improvement	As expected	Good	Excellent
Students visualize the data before analysis.	There is no visualization attempt or The student is not able to operate without help from other students/ teachers.	Some visualization techniques are not suitable for the given data and have no or poor description.	The student can plot the data and can draw some simple conclusions about them.	The student uses different techniques to visualize different kinds of variables (numerical, categorical, etc.) and can understand the plots.	The student uses suitable techniques to visualize each variable depending on its kind. Also, the student manages to visualize combinations of the variables and can figure out the possible correlation status. The student can interpret the visualizations.
Data model selection.	The student cannot understand the concept of a machine learning model and cannot suggest one without the active help of other students or the teacher.	The student can understand the concept of a machine learning model but cannot suggest a suitable model for this case.	The student can suggest a suitable machine learning model for the data. The suggestion may be boosted by a constructive discussion with the teacher or other students.	The student can suggest a machine learning problem and demonstrate its benefits contrary to other models.	The student can suggest a suitable machine learning problem and adequately argue for his/her choice as well as suggest alternative approaches.
The student trains and tests the machine learning model.	The student is not able to train or test the ML problem.	The student can import the right Python libraries but the training and testing commands are not suitable for the use case.	The student can train and test the machine learning problem, probably referring to the previous exercises of this lesson.	The student can make micro-adjustments to the training and testing of the model such as the percentage of the split of the dataset to train and test the sample.	The student can train and test the ML model and fine-tune it to provide better results. The student is also able to perform advanced testing techniques.

Co-funded by the Erasmus+ Programme of the European Union

The student peak suitable metrics for the evaluation of the results.	The student does not comprehend the concept of evaluating metrics.	The student does understand the concept of evaluation metrics but is unable to provide a suitable suggestion.	the student can propose and use one or two metrics.	The student also understands the outcomes and interpret the numbers.	The student can understand and interpret the meaning of the metrics and apply fine-tuning to the original ML model to receive better results. Finally, the student can recognize phenomena like overfitting.
The student documents his/her work properly	The student didn't provide any documentation.	The student provided a little documentation that didn't have any reflection to the theory that he/she has been taught.	The student provided adequate documentation.	The student provided a full report with solid connection between the data with his/her knowledge.	documenting each step and suggesting alternatives.

2 Learning Units reference

These Learning units have to be completed before approaching this Project Work/

TRAINING MODULE	LEARNING UNITS
MACHINE LEARNING (TM3)	**Unit 1:** **§1.7** Machine learning for manufacturing applications **Unit 2:** **§2.1** What is supervised Learning? **§2.2** Mapping to real-life Manufacturing Cases **§2.3.2** Regression **Unit 4:** **§4** Regression with Python for ML

3 Learning Outcomes

These Skills and Knowledge will be improved upon the project work completion.

TRAINING MODULE	SKILLS AND KNOWLEDGE
MACHINE LEARNING (TM3)	**Skills** **S1:** Understand the basic concepts of an unknown dataset derived from an unfamiliar field. **S2:** Understand an advanced manufacturing problem and be able to verify if it can be solved via machine learning techniques. **S3:** Understand what field of machine learning is fitting a use case **S4:** Preprocess the given dataset to ensure homogeneity. **S5:** Find a fitting machine learning model for the given use case **S6:** Train a machine learning model using Python **S7:** Evaluate a machine learning model using suitable techniques.
	Knowledge **K1:** Machine Learning Theory **K2:** Basic machine learning types (supervised and unsupervised) **K3:** Basic machine learning techniques (regression, classification, clustering, association) **K4:** Regression algorithms in Python **K5:** Basic evaluation metrics of machine learning models.

4 IoT Lab Resources

4.1 Requirements

To be able to do this challenge, you will need the following things:

- A PC or a Laptop that meets the needs of ML model training Python Environment.
- Internet access.
- A python IDE is required or you can use Google Colab.
- Installed Python 3 Release according to the operating system.
    - Windows
    - macOS
    - other
- Download the dataset. The dataset can be found here in .csv form.

These IoT lab resources are suggested to complete the project work

- Installed the basic ML and visualization Python libraries in a pre-created project that the teacher can give to the students.
  indicatively and not exclusively:
    - Pandas
    - Numpy
    - Pyplot
    - Matplotlib
    - Seaborn
    - Sklearn

5 The project step by step

To complete successfully the project work you can follow this step sequence.

SEQUENCE	WORK	Estimated time
STEP 1: Definition of the problem	***Students will read the challenge and make sure they understand all the requisites asked***	***2h***
STEP 2: Visualize and understand the data	***Students will use the techniques they have learned to check the size and the kind of the given data, visualize it and understand how each variable affects the others.***	***6h***
STEP 3: Prepare the data	***Students will preprocess the data to ensure homogeneity.***	***2h***
STEP 4: Choose a suitable Machine Learning model	***Students will study the theory and choose a suitable model to implement***	***5h***
STEP 5: Train and test the machine learning model	***Students will use Python libraries to train, test, and fine-tune their model***	***8h***
STEP 6: Evaluate the Results	***Students will use suitable evaluation metrics and will share their thoughts about the performance of their model. They may suggest future improvements for the machine learning model and even to the data collection attributes.***	***8h***