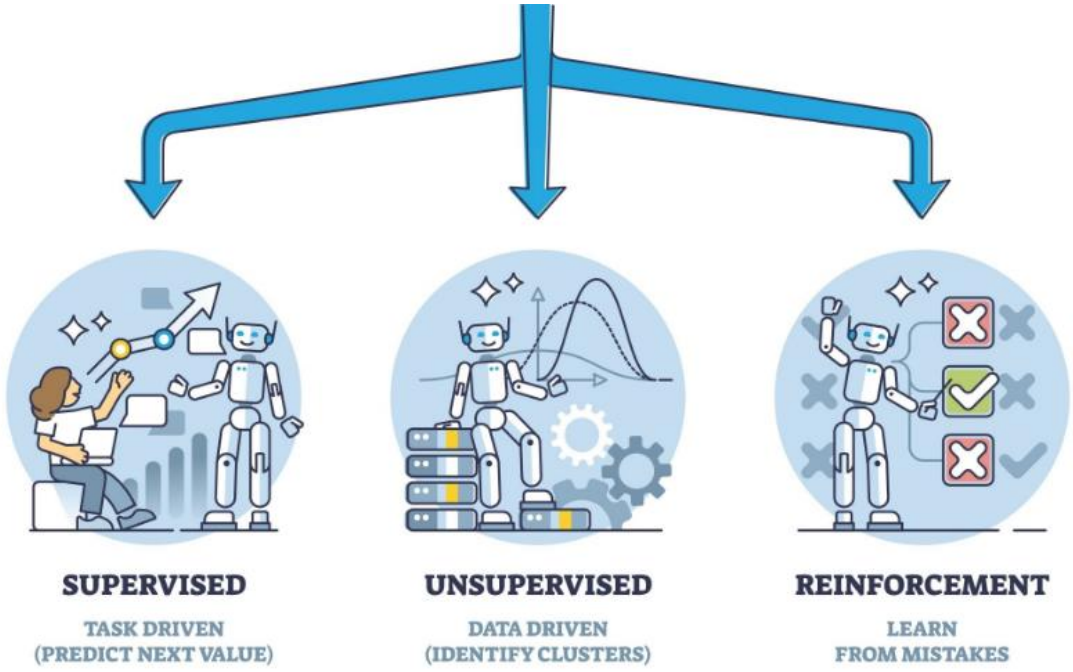
University of Bejaia

Department of Computer Sciences

Machine Learning For Big Data



Course Handout

By Dr. Houda EL BOUHISSI BRAHAMI

Academis Year 2024-2025

Table of Contents

Introduction

Machine Learning is a rapidly growing field that enables computers to learn from data and make intelligent decisions without being explicitly programmed. This course aims to provide students with both theoretical knowledge and practical skills necessary to understand and apply machine learning techniques effectively. It is structured into two main parts: lectures and practical sessions (labs).

The lecture sessions will cover fundamental concepts, mathematical foundations, and key machine learning algorithms, including supervised and unsupervised learning, model evaluation, and optimization techniques. The focus will be on developing a deep understanding of how these algorithms work and when to use them.

The practical sessions (labs) will complement the lectures by providing hands-on experience with implementing machine learning models using programming tools and real-world datasets. Students will work on exercises, case studies, and projects to strengthen their problem-solving skills and gain confidence in applying machine learning techniques to real-world problems.

By the end of this course, students will be able to:

- Grasp the core principles and methodologies of machine learning.
- Implement and evaluate different machine learning models.
- Utilize programming frameworks and libraries for data analysis and model training.
- Develop critical thinking and problem-solving skills through hands-on labs.

This course is designed for students who wish to build a strong foundation in machine learning, whether for academic research or industry applications. Through a combination of theoretical knowledge and practical exercises, students will gain valuable insights into one of the most exciting fields in artificial intelligence today.

Chapter One

Introduction to Machine Learning

Lecture Notes

I.1. Machine Learning – an overview

We are living in a world of Humans and Machines. Humans have been learning and evolving from experience for billions of years, on the other hand, the era of machines and robots has just begun.

In today's world, these machines or robots need to be instructed to perform, but what if machines started to learn from their own and this is where machine learning appears.

AI is a term being applied broadly in the technological world to describe solutions that can learn on their own. These algorithms can look at vast amounts of data and finding trends in it, trends that unveil insights, insights that would be extremely hard for a human to find. However, AI algorithms can't think like you and me. They are trained to perform very specialized tasks, whereas the human brain is a generic thinking system.

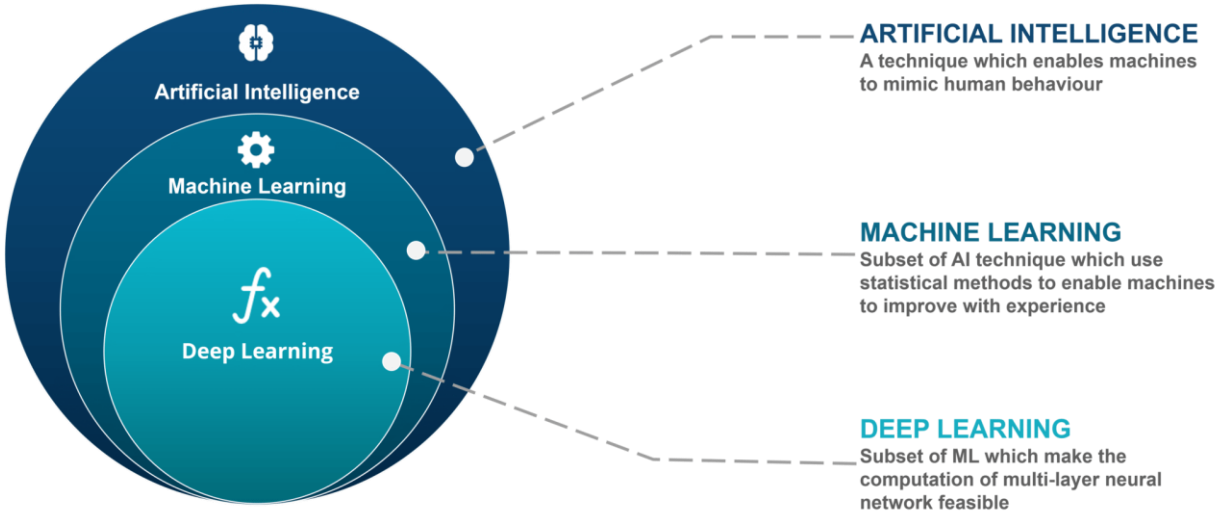


Figure 1: Artificial intelligence in pratice

Here are some definitions of machine learning:

Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed. Arthur L. Samuel, AI pioneer, 1959.

Now, before we introduce machine learning more formally, here is what some other people said about the field:

"*The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience*".

Tom Mitchell, Professor Machine Learning at Carnegie Mellon University and author of the popular \Machine Learning" textbook

"*Machine learning is the hot new thing*".

John L. Hennessy, President of Stanford (2000-2016)

"*A breakthrough in machine learning would be worth ten Microsofts*".
Bill Gates, Microsoft Co-Founder

In general, Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences, and make predictions. Machine learning contains a set of algorithms that work on a huge amount of data. Data is fed to these algorithms to train them, and based on training, they build the model & perform a specific task.

Machine Learning is different from traditional programming. In traditional programming, we would feed the input data and a well written and tested program into a machine to generate output. When it comes to machine learning, input data along with the output associated with the data is fed into the machine during the learning phase, and it works out a program for itself.

I.2. Machine Learning – How it works

While we go over some of these applications in class, it is a good exercise to think about how machine learning could be applied in these problems or tasks listed above:

- What is the desired outcome?
- What could the dataset look like?
- What machine learning types (we see them further) and algorithms would you use?
- How would you measure success?
- What are potential challenges or difficulties?

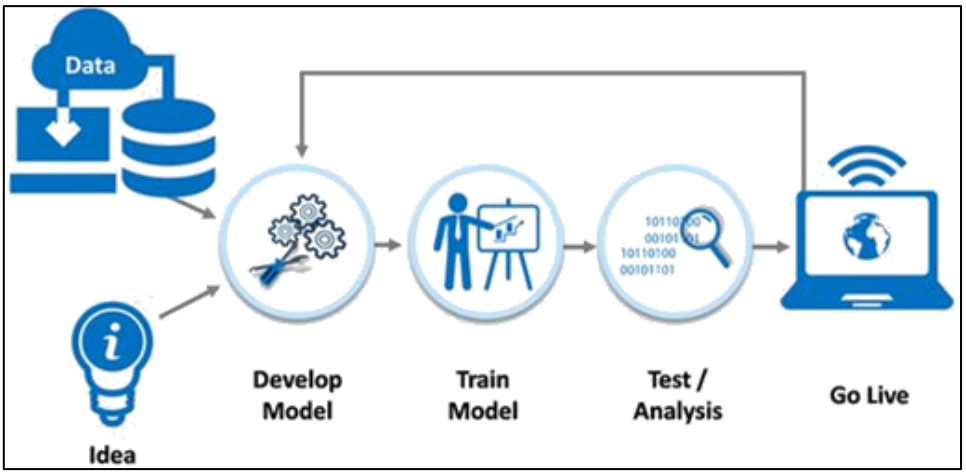


Figure 2 : Machine Learning Steps

Every machine learning project goes throw different steps :

- **Data collection**

This is maybe the most important and time-consuming process. In this step, we need to collect data that can help us to solve our problem.

For example, if you want to predict the prices of the houses, we need an appropriate dataset that contains all the information about past house sales and then form a tabular structure. We are going to solve a similar problem in the implementation part.

- **Data processing**

Once we have the data, we need to bring it in proper format and preprocess it. There are various steps involved in pre-processing such as data cleaning, for example, if your dataset has some empty values or abnormal values(e.g, a string instead of a number) how are you going to deal with it? There are various ways in which we can, but one simple way is to just drop the rows that have empty values.
Also sometimes in the dataset, we might have columns that have no impact on our results such as id's, we remove those columns as well.

- **Model building**:

Once data is ready is to be fed into a Machine Learning algorithm. A model is the output of a machine-learning algorithm run on data. For example, after

implementing Linear Regression on data, we get an equation of the best-fit line and this equation is termed as a model.

- **Model training**:

This step learning is the process of teaching a machine learning model to make predictions or decisions based on input data.
This process involves feeding the model with a labeled dataset (in supervised learning) or an unlabeled dataset (in unsupervised learning) and adjusting its internal parameters to minimize error and improve accuracy

- **Model evaluation:**

Model evaluation in machine learning is the process of assessing how well a trained model performs on unseen data. It helps determine whether the model generalizes well to new data and meets the desired performance criteria. Proper evaluation is essential to avoid issues like overfitting (performing well on training data but poorly on new data) or underfitting (failing to capture patterns in the data).

I.3. Applications for Machine Learning

Machine leaning is involved in various domains, here are some examples:

- Email spam detection
- Face detection and matching (e.g., iPhone X, Windows laptops, etc.)
- Web search (e.g. Bing, Google)
- Sports predictions
- Post office (e.g., sorting letters by zip codes)
- Credit card fraud
- Stock predictions
- Smart assistants (Apple Siri, Amazon Alexa, . . .)
- Product recommendations (Amazon)
- Self-driving cars (e.g., Uber, Tesla)
- Language translation (Google translation)
- Sentiment analysis
- Medical diagnoses

I.4. Machine Learning – Types

Based on the methods and way of learning, machine learning is divided into mainly three types, which are (figure 3):

- Supervised Machine Learning
- Unsupervised Machine Learning
- Reinforcement Learning

Supervised learning is the most developed branch of machine learning.

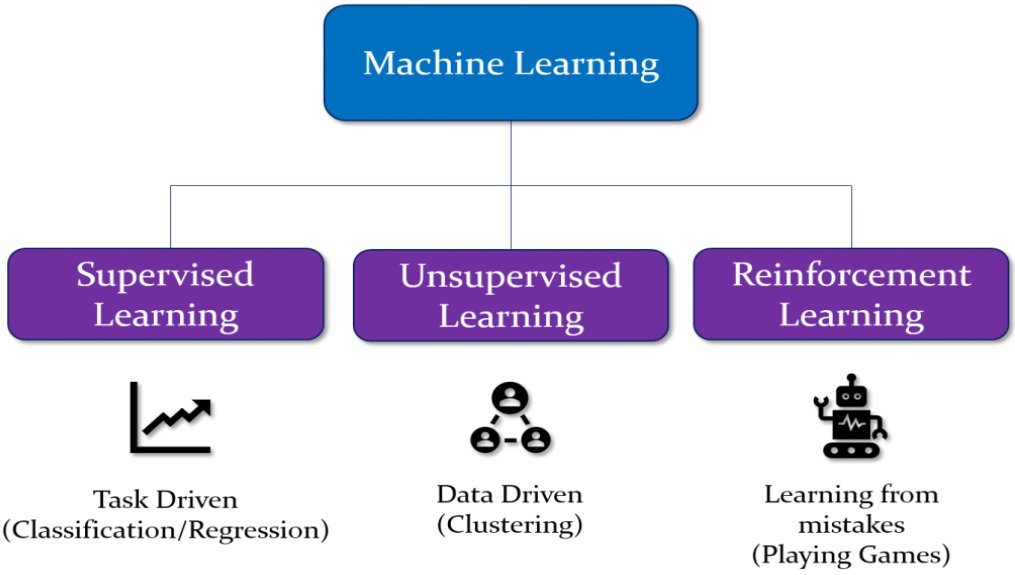


Figure 3 : Machine Learning Types

In this course, we will provide a detailed description of the types of Machines Learning along with their respective algorithms.

I.4.1. Supervised Machine Learning

Supervised learning is the subcategory of machine learning that focuses on learning a classification or regression model. As its name suggests, supervised machine learning is based on supervision. It means in the supervised learning technique, we train the machines using the "labelled" dataset, and based on the training, the machine predicts the output.

In supervised learning, you train your model on a labelled dataset that means we have both raw input data as well as its results. We split our data into a training dataset and test dataset where the training dataset is used to train our network whereas the test dataset acts as new data for predicting results or to see the accuracy of our model.

The main goal of the supervised learning technique is to map the input variable(x) with the output variable(y). Some real-world applications of supervised learning are Risk Assessment, Fraud Detection, Spam filtering, etc.

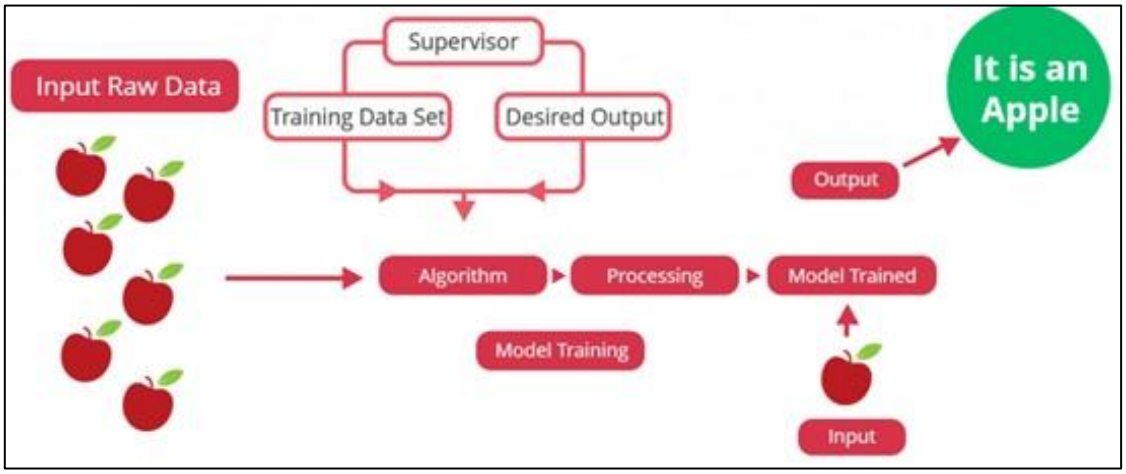


Figure 4 : Supervised learning Example

Real-life examples:

- Email Spam (Classification)– The algorithm takes historical spam and non-spam emails as input. Consequently, it draws patterns in data to classify spam from others.
- Stock Price Prediction (Regression)– Historical business market data is fed to the algorithm in this method. With proper regression analysis, the new price for the future is predicted.

Supervised machine learning can be classified into two types of problems, which are given below:

- Classification
- Regression

Since supervised learning work with the labelled dataset so we can have an exact idea about the classes of objects. These algorithms are helpful in predicting the

output based on prior experience. However, supervised learning algorithms present some drawbacks:

- They are not able to solve complex tasks.
- It may predict the wrong output if the test data is different from the training data.
- It requires lots of computational time to train the algorithm.

a. **Classification**

The prediction task is a classification when the target variable is discrete. An application is the identification of the underlying sentiment of a piece of text.
Some real-world examples of classification algorithms are Spam Detection, Email filtering, **etc.**
Some popular classification algorithms are given below:

- Random Forest Algorithm (RF)
- Decision Tree Algorithm (DT)
- Logistic Regression Algorithm (LR)
- Support Vector Machine Algorithm (SVM)

b. Regression

The prediction task is a regression when the target variable is continuous. An example can be the prediction of the salary of a person given their education degree, previous work experience, geographical location, and level of seniority.

Some popular Regression algorithms are given below:

- Simple Linear Regression Algorithm
- Multivariate Regression Algorithm
- Decision Tree Algorithm
- Lasso Regression

I.4.2. UnSupervised Machine Learning

In contrast to supervised learning, unsupervised learning is a branch of machine learning that is concerned with unlabeled data. The common task in unsupervised learning is clustering.

Unsupervised learning is different from the supervised learning technique; as its name suggests, there is no need for supervision. It means, in unsupervised machine learning, the machine is trained using the unlabeled dataset, and the machine predicts the output without any supervision.

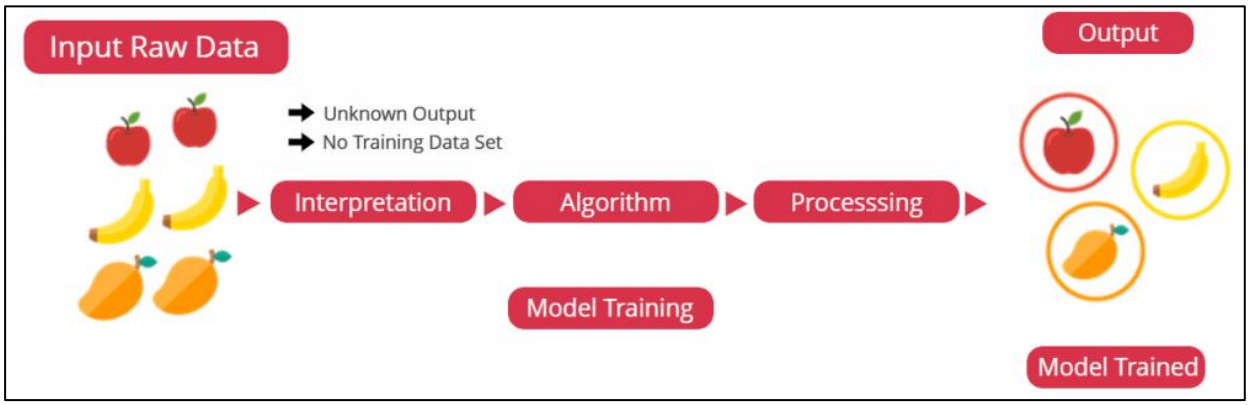


Figure 5 : Unsuppervised learning Example

In unsupervised learning, the models are trained with the data that is neither classified nor labelled, and the model acts on that data without any supervision.

The main aim of the unsupervised learning algorithm is to group or categories the unsorted dataset according to the similarities, patterns, and differences. Machines are instructed to find the hidden patterns from the input dataset.

Let's take an example to understand it more preciously; suppose there is a basket of fruit images, and we input it into the machine learning model. The images are totally unknown to the model, and the task of the machine is to find the patterns and categories of the objects.

So, now the machine will discover its patterns and differences, such as color difference, shape difference, and predict the output when it is tested with the test dataset.

For Example:

Data with similar traits are asked to group by the algorithm. These groups are called clusters, and the process is called clustering. In retail analytics, various customers are usually clustered based on their purchase and other behaviors.

Unsupervised Learning can be further classified into two types, which are given below:

- Clustering
- Association

a. Clustering

The clustering technique is used when we want to find the inherent groups from the data. It is a way to group the objects into a cluster such that the objects with the most similarities remain in one group and have fewer or no similarities with the objects of other groups. An example of the clustering algorithm is grouping the customers by their purchasing behavior.

Some of the popular clustering algorithms are given below:

- K-Means Clustering algorithm
- Mean-shift algorithm
- DBSCAN Algorithm
- Principal Component Analysis
- Independent Component Analysis

b. Association

Association rule learning is an unsupervised learning technique, which finds interesting relations among variables within a large dataset. The main aim of this learning algorithm is to find the dependency of one data item on another data item and map those variables accordingly so that it can generate maximum profit. This algorithm is mainly applied in Market Basket analysis, Web usage mining, continuous production, etc.

Some popular algorithms of Association rule learning are Apriori Algorithm, Eclat, FP-growth algorithm.

Unsupervised algorithms can be used for complicated tasks compared to the supervised ones because these algorithms work on the unlabeled dataset and Unsupervised algorithms are preferable for various tasks as getting the unlabeled dataset is easier compared to the labelled dataset.

However, these algorithms present some drawbacks:

- The output of an unsupervised algorithm can be less accurate as the dataset is not labelled, and algorithms are not trained with the exact output in prior.
- Working with Unsupervised learning is more difficult as it works with the unlabeled dataset that does not map with the output.

Some examples of unsupervised learning: Recommendation Systems, Anomaly Detection, Network Analysis:

I.4.3. Reinforcement Learning

Reinforcement learning (RL) is the process of learning from rewards while performing a series of actions. In reinforcement learning, we do not tell the learner or agent (for example, a robot), which action to take but merely assign a reward to each action and/or the overall outcome (figure 6). Instead of having "correct/false" labels for each step, the learner must discover or learn a behavior that maximizes the reward for a series of actions. In that sense, it is not a supervised setting.

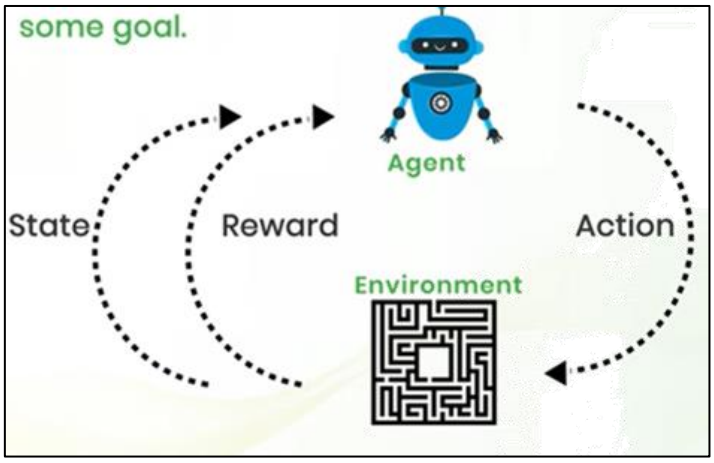


Figure 6 : Reinforcment learning

RL is somewhat related to unsupervised learning; however, reinforcement learning really is its own category of machine learning. Reinforcement learning will not be covered further in this class.

For an easier explanation, let’s take the example of a dog.

We can train our dog to perform certain actions, of course, it won't be an easy task. You would order the dog to do certain actions and for every proper execution, you would give a biscuit as a reward. The dog will remember that if it does a certain action, it will get biscuits. This way it will follow the instructions properly next time.

Typical applications of reinforcement learning involve playing games and some forms of robots, e.g., drones, warehouse robots, and more recently self-driving cars.

I.5. Machine Learning – Languages

Among the programming languages, Python is the most important to build machine learning models. This is due to the various benefits mentioned in the section below. Other programming languages that could use for Machine Learning Applications are R, C++, JavaScript, Java, C#.

Here are the main reasons:

- Today, Python aids to develop many data heavy.
- Highly readable, less complexity, fast prototyping
- Easy to offload number crunching to underlying C/Fortran/…
- Easy to install and import many rich libraries such as :
    - numpy: used for data structures and working with images
    - scipy: fast numerical recipes and Scientific Computing
    - Pandas for high-level data structures and analysis
    - Matplotlib, Seaborn, and Scikit for data representation
    - scikit-learn machine learning algorithms
    - TensorFlow and Pytorch for Deep Learning applications

I.6. Evaluation of a machine-learning model

Evaluation metrics are crucial in assessing the performance of machine learning models. They provide quantitative measures that guide the selection of models and the tuning of hyperparameters. Different tasks require different metrics, and understanding which metric to use is key to interpreting model results effectively.

There are different model evaluation criteria in machine learning.

We decide which metrics to choose depending on whether our problem is classification or a regression problem. Accuracy, precision, recall, F1 Score, ROC, AUC are used for classification tasks. Metrics such as MSE, RMSE, MAE, R2 Score can be used for regression tasks.

I.6.1 Performance Metrics for Classification Problems

The first thing to look for in model evaluation metrics is whether the class distribution in the dataset is unbalanced. Whether the dataset is balanced or unbalanced; we should look at the recal, precision, and their harmonic mean, F1 Score metrics. We also look at the AUC value and get an idea about the classification we have made.

In machine learning, model evaluation metrics such as accuracy, precision, recall, F1 Score, ROC, and AUC are used for classification tasks.

- **Accuracy**

Accuracy is used to describe the closeness of a measurement to the true value. It is a correct classification rate, in other words, the number of correct predictions of the model on all predictions made.

$$\frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Where :

— TP : True Positive which represents actual value Positive and predicted value Positive.

— TN : True Negative which reprsents actual value Negative and predicted value Negative.

— FP : False Positive which reprsents actual value Negative and predicted value Positive.

— FN : False Negative which reprsents actual value Positive and predicted value Negative.

Accuracy can be used if the classification problem we have has a balanced class distribution. If there is an unbalanced distribution in our dataset, we cannot use accurate value directly. We also need to look at the Recall and Precision values.

- **Precision**

Precision in machine learning is a metric used to evaluate the accuracy of positive predictions in a classification model. It measures how many of the instances predicted as positive are correct.

$$\frac{TP}{(TP + FP)}$$

Precision is particularly important in scenarios where false positives carry significant consequences, such as fraud detection or medical diagnosis. A high precision indicates that when the model predicts a positive case, it is likely correct.

However, precision alone does not provide a complete picture of performance, as it does not consider false negatives. It is often used alongside recall balancing predictive performance, especially in imbalanced datasets.

- **Recall**

Accuracy in machine learning is a performance metric that measures the proportion of correctly classified instances out of the total instances in the dataset.

$$\frac{TP}{(TP + FN)}$$

It provides a straightforward way to evaluate the overall effectiveness of a model by showing how often the model's predictions match the actual labels. While accuracy is a useful metric for balanced datasets, it may be misleading in cases where the classes are imbalanced (e.g., rare events or diseases).

In such cases, a model could achieve high accuracy by simply predicting the majority class most of the time, but it may fail to correctly identify the minority class.

- **F1 Score**

The F1 score is a performance metric in machine learning that combines both precision and recall into a single value. It is the harmonic means of precision and recall, meaning it balances the trade-off between the two.

$$2 * \frac{Precision * Recall}{Precision + Recall}$$

The F1 score is particularly useful when dealing with imbalanced datasets, as it provides a more balanced view of a model's performance than accuracy alone. A high F1 score indicates that the model has a good balance between identifying positive instances correctly (recall) and minimizing false positives (precision).

It's especially important in scenarios where both false positives and false negatives are costly, such as in medical diagnostics or fraud detection.

- **ROC Curve (Receiver Operating Characteristic Curve)**

  The **ROC curve** (Receiver Operating Characteristic curve) is a graphical representation used to evaluate the performance of a binary classification model across various threshold values. It plots the **True Positive Rate (Recall)** on the y-axis and the **False Positive Rate (FPR)** on the x-axis.

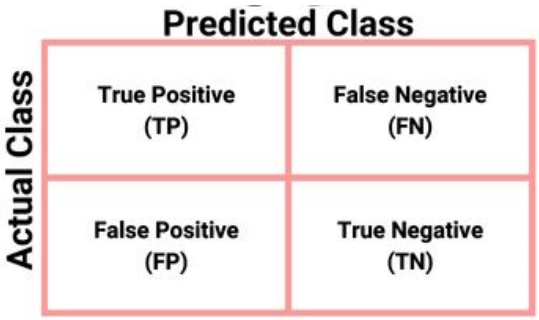
  The ROC curve shows how the model's predictions change as the threshold for classifying positive instances is adjusted.

  An ideal model will have a curve that hugs the top left corner, indicating high recall (detecting most positives) and low false positives. The **AUC** (Area Under the Curve) is often used to quantify the model's performance—higher AUC values indicate better overall performance in distinguishing between the two classes.

  The ROC curve is particularly useful when comparing multiple models and selecting the optimal threshold for a given application.

- **Confusion Matrix**

A confusion matrix is a probability table containing the actual and predicted dimensions of the number of correct and incorrect predictions made by a classifier.



A confusion matrix is a performance evaluation tool for classification models, commonly used in supervised learning tasks. It summarizes the outcomes of a model's predictions by comparing them to the actual labels in a dataset. The confusion matrix is especially useful in identifying how well a classification model is performing, especially in imbalanced datasets.

The confusion matrix is a powerful tool for understanding a classification model's performance, highlighting its strengths and weaknesses. It is especially important for problems where the costs of false positives and false negatives vary, such as in fraud detection, medical diagnostics, and spam classification.

I.6.2 Performance Metrics for Regression Problems

In regression problems, where the goal is to predict continuous values, the performance of the model is assessed using different metrics compared to classification tasks. These metrics evaluate how close the model's predictions are to the actual target values. Below are the key performance metrics commonly used in regression problems:Mean Squared Error (MSE).

Each of these metrics provides valuable insights depending on the nature of the regression problem and the specific requirements of the application. Understanding them is essential for accurately assessing model performance and making improvements where needed.

- **Mean Squared Error**

Mean Squared Error (MSE) is the mean of the squared error. It's more popular than Mean Absolute Error because the focus is geared more towards large errors. This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error (RMSE)**

Root Mean Squared Error (RMSE) is the square root of MSE. It is a standard way of measuring the error of a model. Generally, the lower the RMSE, the better.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Mean Absolute Error**

It is the mean of the absolute value of the errors. This is the easiest of the metrics to understand since it's just an average error.

$$\frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

- **R2 Score**

The R2 Score is the percentage of the dependent variable explanation of the independent variables in the dataset. It represents how close the data points are to the fitted regression line. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

How to choose the right metric ?

The choice of evaluation metric

depends on the specific application and the business or research goals. For instance, in medical diagnosis, recall might be more important than precision because missing a positive case could be life-threatening. In contrast, in email spam detection, precision might be more critical because false positives (non-spam emails marked as spam) are more inconvenient to users than false negatives (spam emails not marked as spam).

It is also common to use multiple metrics to get a more holistic view of the model's performance. For example, in a classification task, one might look at both the accuracy and the F1 score to understand both the overall correctness and the balance between precision and recall.

In conclusion, evaluation metrics are indispensable tools in machine learning that provide insights into the effectiveness of models. They guide the model development process and help stakeholders make informed decisions based on model performance. Understanding and selecting the appropriate metric is therefore fundamental to the success of machine learning projects.

I.7. Conclusion

Machine learning has revolutionized the way we process and analyze data, enabling intelligent systems to make predictions and automate decision-making. This chapter introduced the key concepts of machine learning, including supervised learning, where models learn from labeled data, unsupervised learning, which uncovers hidden patterns in data, and reinforcement learning, where agents learn through trial and error.

We also explored different types of machine learning models and their evaluation techniques, emphasizing the importance of performance metrics such as accuracy, precision, recall, and error rates. Proper model evaluation ensures that a trained model generalizes well to new data and avoids common pitfalls like overfitting and underfitting.

Understanding these foundational principles is crucial for developing robust and efficient machine learning models. As we delve deeper into more advanced topics, practical applications, and optimization techniques, mastering these fundamentals

will provide a strong foundation for real-world problem-solving in artificial intelligence and data science.

As we move forward, the next chapter will provide a deeper dive into supervised learning algorithms, exploring key models such as linear regression. Understanding these techniques will lay the groundwork for building more advanced and efficient machine learning solutions.

Chapter Two

Supervised Learning – Regression - Lecture Notes

II.1. Regression – an overview

Linear Regression is a machine-learning algorithm based on supervised learning**.** Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

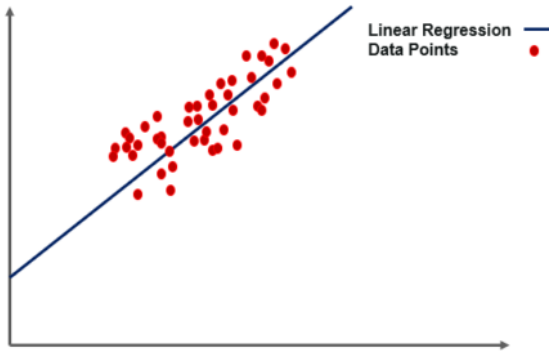


Figure 7: Linear regression line

Linear regression performs the task of predicting a dependent variable value (y) based on a given independent variable (x). Hence, the name is Linear Regression. For example, in the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best-fit line for the model.

The formula for simple linear regression is:

$$y = \beta_0 + \beta_1 X + \epsilon$$

- **Y** : is the predicted value of the dependent variable (y) for any given value of the independent variable (x).
- **B0** is the intercept, the predicted value of y when the x is 0.
- **B1** is the regression coefficient – how much we expect y to change as x increases.
- **X** is the independent variable (the variable we expect is influencing y).
- e is the error of the estimate, or how much variation there is in our estimate of the regression coefficient.

There are a range of different approaches used in machine learning to perform regression. Some of the most common regression techniques in machine learning can be grouped into the following types of regression analysis:

- **Simple Linear Regression**

A simple straight-line equation involving slope (dy/dx) and intercept (an integer/continuous value) is utilized in simple Linear Regression. Here a simple form is: y=mx+c where y denotes the output x is the independent variable, and c is the intercept when x=0. With this equation, the algorithm trains the model of machine learning and gives the most accurate output

- **Multiple linear regression**

When several independent variables more than one, the governing linear equation applicable to regression takes a different form like: y= c+m1x1+m2x2… mnxn where represents the coefficient responsible for impact of different independent variables x1, x2 etc. This machine-learning algorithm, when applied, finds the values of coefficients m1, m2, etc., and gives the best fitting line.

- **Logistic regression**

Logistic regression is used when the dependent variable can have one of two values, such as true or false, or success or failure. Logistic regression models can be used to predict the probability of a dependent variable occurring. Generally, the output values must be binary.

II.2. Linear Regression – How does it work

Linear regression uses the relationship between the data-points to draw a straight line through all them. This line can be used to predict future values.

Thus, linear regression is a supervised learning algorithm that simulates a mathematical relationship between variables and makes predictions for continuous or numeric variables such as sales, salary, age, product price, etc.

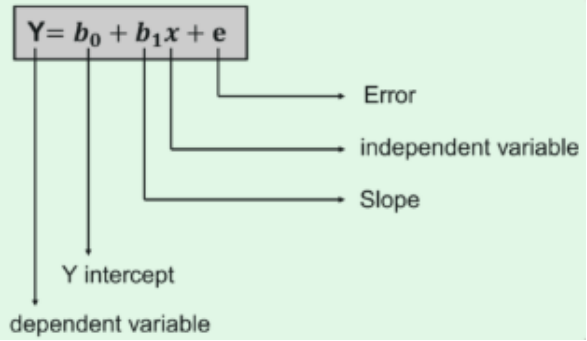
We will consider the process of applying Linear Regression in a Machine Learning project in the labs part.

II.3. Linear Regression Terminologies

The following terminologies are important to be familiar with before moving on to the linear regression algorithm.

- **Cost Function**

The best fit line can be based on the linear equation given below. The dependent variable that is to be predicted is denoted by Y.



— A line that touches the y-axis is denoted by the intercept b0.

— b1 is the slope of the line, x represents the independent variables that determine the prediction of Y.

— The error in the resultant prediction is denoted by e.

The cost function provides the best possible values for b0 and b1 to make the best-fit line for the data points. We do it by converting this problem into a minimization problem to get the best values for b0 and b1. The error is minimized in this problem between the actual value and the predicted value.

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

We choose the function above to minimize the error. We square the error difference and sum the error over all data points, the division between the total number of data points. Then, the produced value provides the averaged square error over all data points.

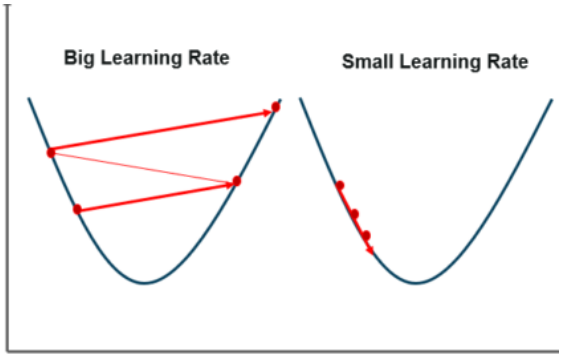
It is also known as MSE(Mean Squared Error), and we change the values of b0 and b1 so that the MSE value is settled at the minimum.

- **Gradient Descent**

The next important terminology to understand linear regression is gradient descent. It is a method of updating b0 and b1 values to reduce the MSE.

The idea behind this is to keep iterating the b0 and b1 values until we reduce the MSE to a minimum.

To update b0 and b1, we take gradients from the cost function. To find these gradients, we take partial derivatives with respect to b0 and b1. These partial derivatives are the gradients and are used to update the values of b0 and b1.



Let us look at a few advantages and disadvantages of linear regression for machine learning.

Advantages	Disadvantages
Linear regression performs exceptionally well for linearly separable data	The assumption of linearity between dependent and independent variables
Easier to implement, interpret and efficient to train	It is often quite prone to noise and overfitting
It handles overfitting well using dimensionally reduction techniques, regularization, and cross-validation	Linear regression is quite sensitive to outliers
One more advantage is the extrapolation beyond a specific data set	It is prone to multicollinearity

II.4. Implementing Linear Regression

The process takes place in the following steps:

1. Loading the Data
2. Exploring the Data
3. Slicing The Data

4. Train and Split Data
5. Generate The Model
6. Evaluate The accuracy

To conclude, in simple linear regression, the equation of the regression line is:

y=aX+by = aX + by=aX+b

The formula for calculating a (the slope) is:

$$a = \frac{n\sum(X_iY_i) - \sum X_i \sum Y_i}{n\sum X_i^2 - (\sum X_i)^2}$$

where:

- n is the number of data points,
- X_i and Y_i are individual values of experience and salary,
- $\sum(X_iY_i)$ is the sum of the product of X and Y,
- $\sum X_i$ and $\sum Y_i$ are the sum of X and Y,
- $\sum X_i^2$ is the sum of squared values of X.

Once we have the slope a, the intercept b is calculated using the formula:

$$b = \bar{Y} - a\bar{X}$$

where:

- $\bar{X}$ is the mean (average) of the X values,
- $\bar{Y}$ is the mean of the Y values.

Let's say we have the following data:

Years of Experience (X)	Salary (Y)
1	35000
3	50000
5	65000
7	80000
9	95000

- $\sum X = 1 + 3 + 5 + 7 + 9 = 25$
- $\sum Y = 35000 + 50000 + 65000 + 80000 + 95000 = 325000$
- $\sum X^2 = 1^2 + 3^2 + 5^2 + 7^2 + 9^2 = 165$
- $\sum XY = (1 \times 35000) + (3 \times 50000) + (5 \times 65000) + (7 \times 80000) + (9 \times 95000) = 1950000$
    - Number of data points n=5

Compute a:

$$a = \frac{(5 \times 1,950,000) - (25 \times 325,000)}{(5 \times 165) - (25^2)}$$

$$a = \frac{9,750,000 - 8,125,000}{825 - 625} = \frac{1,625,000}{200} = 8125$$

So, **a=8125 (the salary increases by $8,125 per year of experience).**

3. Compute b:

$$b = \frac{325,000}{5} - (8125 \times \frac{25}{5})$$

$$b = 65,000 - (8125 \times 5)$$

$$b = 65,000 - 40,625 = 24,375$$

So, **b=24,375 (the predicted salary for 0 years of experience).**

Final Regression Equation: **Salary=8125 x Years of Experience + 24,375**

This equation can now be used to predict salaries for different years of experience.

Quiz

1. Which of these is an example of a parameter that is calculated during training for a linear regression model?

a. Weight

b. Learning rate

c. Prediction

d. Label

2. Fill-in-the-blanks

Enter one or more words to complete the sentence.

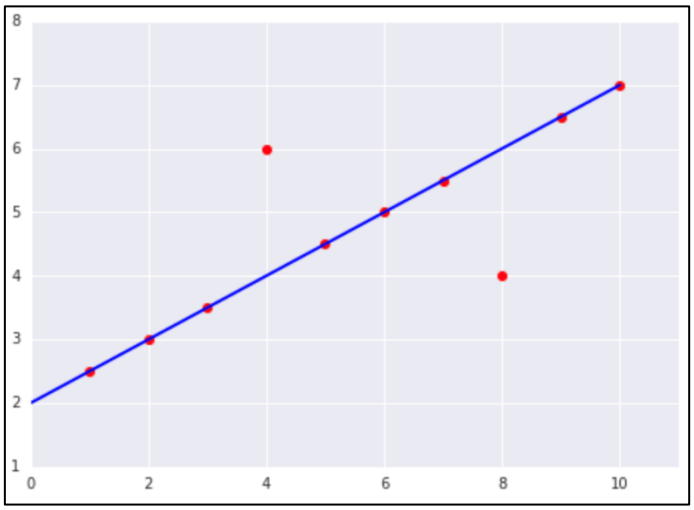
Suppose you are building a linear regression model to predict the sale price of a used car. The training dataset includes the following information: sale price (label),

model year (feature), MSRP (feature), odometer mileage (feature), gas mileage (feature). How many weights will there be for this model ? [ ]

3. Fill-in-the-blanks

Enter one or more words to complete the sentence.

Review the graph below. What is the mean squared error? [ ]



4. Which of these controls the size of the steps of the gradient descent algorithm?

a. Learning rate
b. Loss function
c. Batch size
d. Regularization rate

5. Suppose you are training a linear regression model and after about 100 iterations you notice that the loss is high and trending downward, but not by a significant amount. What is likely to be the problem?

a. The learning rate is too large.
b. The learning rate is too small.
c. Your dataset has too many examples.
d. Your dataset does not have enough examples.

II.5. Exercises

Exercise 1

Given a simple linear model: $Yt = \beta 0 + \beta 0Xt + ut$. The following information is given:

$$\sum YX=184500 \quad \sum Y^2=26350 \quad \sum X^2=1400000 \quad \bar{Y}=60 \quad \bar{X}=400 \quad n=7$$

Questions :

- Estimate the coefficients of the model.
- Evaluate the quality of the fit.
- Test the overall significance of the model

Solution :

Depending on the data available, the following formulae will be used to answer the three questions:

- $\hat{\beta}_1 = \dfrac{\sum X_t Y_t - n\bar{x}\bar{y}}{\sum X_t^2 - n\bar{x}^2}$ et $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$
- $R^2 = \dfrac{\hat{\beta}_1^2[\sum X_t^2 - n\bar{x}^2]}{[\sum Y_t^2 - n\bar{y}^2]}$
- $F = \dfrac{\frac{R^2}{1}}{\frac{(1-R^2)}{(n-2)}}$

After calculation, knowing that ΣYX =ΣXY, we have the following results:

$\hat{\beta}_1 = 0{,}0589$; $\hat{\beta}_0 = 36{,}44$; $R^2 = 0{,}8455$; $F = 27{,}3618$

Since the R^2 is relatively high, around 85%, the fitness is of good quality. And since $F > F\,[1\,;\,5]$ =6.61, we can conclude that the model is good overall.

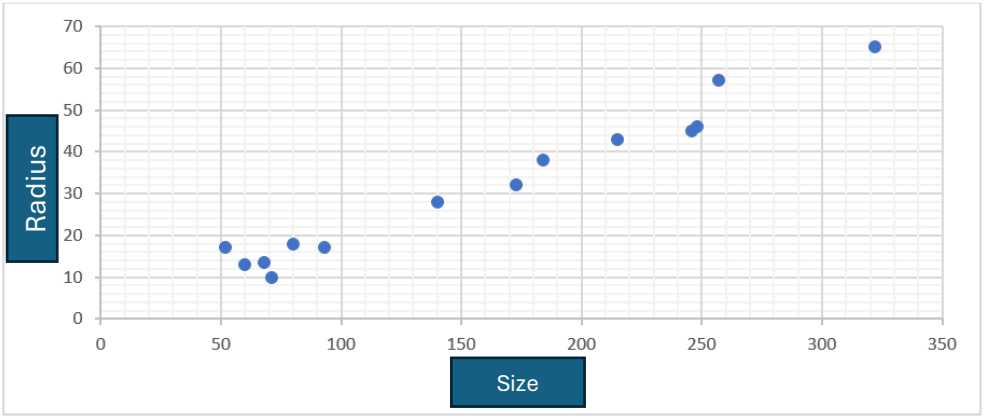
Exercise 2

We have 14 trees in this forest and have measured the height and radius (in cm) of each of them, then plotted these results in the table below.

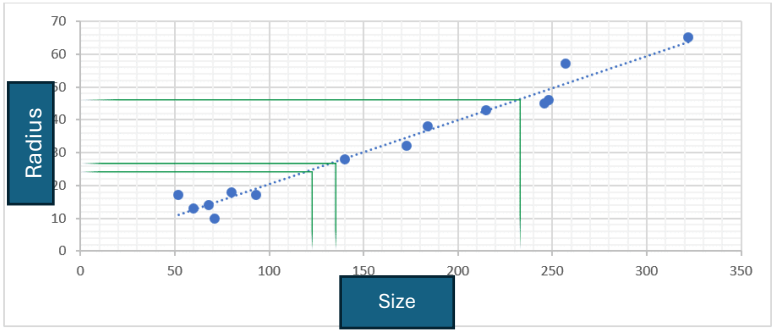
So, we want to know the radius of a 15th tree whose height we know (135 cm), and we also want to know the height of a 16th tree whose radius we know (25 cm)...

We plot these measurements on a graph and obtain the following scatter plot. These are the observed values.

	Size	Radius
A1	184	38
A2	246	45
A3	322	65
A4	257	57
A5	248	46
A6	215	43
A7	173	32
A8	93	17
A9	71	10
A10	52	17
A11	68	13,6
A12	60	13
A13	80	18
A14	140	28
A15	135	??
A16	??	25
A17	232	??



Human beings immediately see a relationship between size and radius. They could therefore easily draw a straight line that would best connect all these points.



We're now going to look at how the machine 'finds' this straight line that relates the size of a tree to its radius.

The first thing to understand is that there are an infinite number of possible straight lines, but which one is the best?

It's very simple: the best line is the one that minimizes the differences between reality (the sizes and radii observed) and predictions (the calculated sizes and radius).

To understand this, remember your math lessons: a straight line representing a relationship between 2 variables x and y has the function :

$$f(x) = a*x + b$$ / a represents the slope, b the intersection.

To do, Use random gradient descent. As a reminder, we want the sum of the squared errors to be as small as possible. To continue the explanation, we're not going to take the sum but the average. This average is called the cost function. It can be written as follows:

$$J(a,b) = \frac{1}{m}\sum_{i=1}^{m} erreur^i = \frac{1}{m}\sum_{i=1}^{m}((a * x^{(i)} + b) - y^{(i)})^2$$

m = represents the number of observations, in our example, the number of trees, i.e. 14 trees. (Remark: the literature divides the cost function by 2m and not by m. I've chosen to divide by m here so as not to complicate the explanation).

With our example data, which as a reminder is as follows, we would have the following cost function:

```
J(a, b) =
    (1/14) *
    [
     (a*184+b-38)^2
     + (a*246+b-45)^2
     + (a*322+b-65)^2
     + (a*257+b-57)^2
     + (a*248+b-46)^2
     + (a*215+b-43)^2
     + (a*173+b-32)^2
     + (a*93+b-17)^2
     + (a*71+b-10)^2
     + (a*52+b-17)^2
     + (a*68+b-13,60)^2
     + (a*60+b-13)^2
     + (a*80+b-18)^2
     + (a*140+b-28)^2
    ]
```

What we want to find out is the minimum of this cost function, i.e. the values of a and b that minimize the cost function. To begin with, we're going to simplify the function to better understand gradient descent.

The function is no longer ‘f(x) = a*x + b’ but ‘f(x) = a*x’ (this means that the straight line must pass through the point {y; x = 0; 0}).

The cost function then becomes :

$$J(a) = \frac{1}{m}\sum_{i=1}^{m} erreur^i = \frac{1}{m}\sum_{i=1}^{m}(a * x^{(i)} - y^{(i)})^2$$

With our example data, this cost function is as follows:

J(a) = (1/14)*((a*184-38)^2+(a*246-45)^2+(a*322-65)^2+(a*257-57)^2+ (a*248-46)^2+(a*215-43)^2+(a*173-32)^2+(a*93-17)^2+(a*71-10)^2+ (a*52-17)^2+(a*68-13,60)^2+(a*60-13)^2+(a*80-18)^2+(a*140-28)^2)

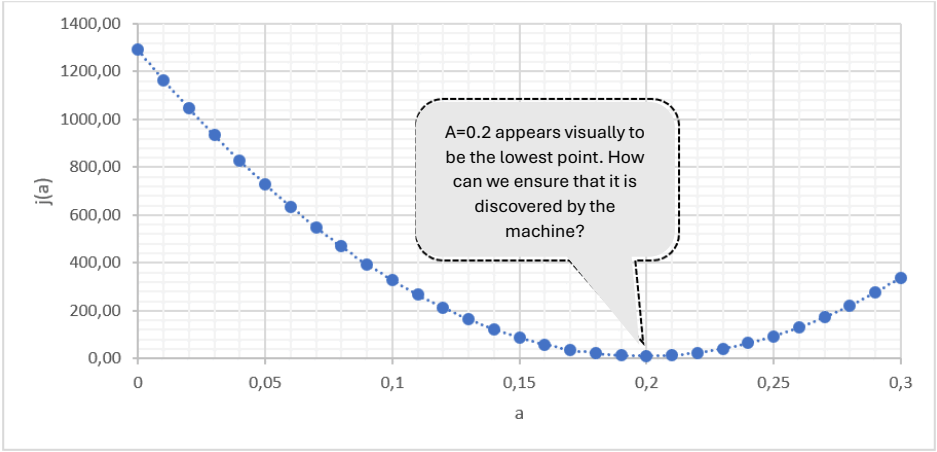
= (452 381/14)*a^2 – 12867,114*a + 1290,854

By replacing a with a few random values, we obtain the following table.

If we display these figures on a scatter plot and draw a curve, we obtain the following graph.

a	J(a)
0	1290,85
0,01	1165,41
0,02	1046,44
0,03	933,92
0,04	827,87
0,05	728,28
0,06	635,15
0,07	548,49
0,08	468,29
0,09	394,55
0,1	327,27
0,11	266,46
0,12	212,11
0,13	164,22
0,14	122,79
0,15	87,83

a	J(a)
0,16	59,33
0,17	37,29
0,18	21,71
0,19	12,6
0,2	9,95
0,21	13,76
0,22	24,03
0,23	40,77
0,24	63,97
0,25	93,63
0,26	129,76
0,27	172,35
0,28	221,4
0,29	276,91
0,3	338,88



Visually, a human being can find the value of a that minimizes the cost function J(a); in our example, it seems to be a = 0.2. But how can the machine find this value, and above all how can it find it more precisely than the human being (after all, visually it seems to be 0.20, but isn't it rather 0.199, 0.201, 0.20001... etc.?

To do this, we'll use the derivative. As a reminder, the derivative measures the slope of a point on a curve. The random gradient descent is a sequence of operations as follows.

Step 1 : The machine randomly chooses a number for a. Example : a = -758.

Step 2 : The machine calculates the derivative to determine whether it is to the left or right of the minimum point.

$$\frac{\partial J(a)}{a} = \frac{452\,381}{7} * a - 12\,867{,}114$$

a	derivative of a
-758	-48999267

The derivative of ‘a = -758’ is negative, which means that the slope is going down and we are to the left of the minimum. We can therefore continue and propose an ‘a’ greater than ‘-758’.

Step 3 : The machine randomly suggests a new number greater than ‘-758’. Example : a = -432.

Step 4 : The machine calculates the derivative to determine whether it is to the left or right of the minimum point.

a	derivative of a
-432	-27931237

The derivative of ‘a = -432’ is always negative, which means that the slope is decreasing, and we are to the left of the minimum. We can therefore continue and propose a greater than ‘-432’.

Step 5 : The machine randomly suggests a new number greater than ‘-432’. Example : a = 23.

Step 6 : The machine calculates the derivative to determine whether it is to the left or right of the minimum point.

a	derivative of a
23	1473528

The derivative of ‘a = 23’ is positive, which means that the slope is increasing, and we are to the right of the minimum. We can therefore continue and propose an a between ‘-432’ and ‘23’.

Step n : The machine stabilizes and finds the value of a that minimizes the cost function.

This method works, but it can be very time-consuming and requires a lot of calculations, so we're going to look at how to save time and minimize the number of calculations required.

Chapter Three

Supervised Learning – KNN algorithm -

III.1. Introduction

Supervised Machine Learning algorithms are used to solve classification or regression problems. The output of a regression problem is a real number (a decimal number with a decimal point).

For example, we could estimate a person's weight based on their height. A classification problem has a discrete value as its output. For example, the 2 headings, like horror films' and 'doesn't like horror films' are discrete data. There is no middle ground.

III.2. Nearest Neighbors (KNN)

The KNN algorithm assumes that similar objects exist nearby. In other words, similar elements are close to each other.

The k nearest neighbours algorithm is one of the algorithms used in the field of artificial intelligence. It is a supervised machine learning algorithm that assigns a category to an element based on the majority class of its nearest neighbours in the training sample. Its principle can be summed up in this sentence: Tell me who your friends are, and I'll tell you who you are.

III.3. Principles

- The K nearest neighbours algorithm is a supervised learning algorithm.
- The aim of the algorithm is to label data.
- Labelled data is available for training and for measuring the quality of predictions.
- Once the algorithm has been trained and tested, it can be used to predict the label of new data.

In general, KNN works as follows :

- Load the data.
- Initialise k to the chosen number of nearest neighbours.
- For each example in the data:

1. Calculate the distance between our query and the current iterative loop observation from the data.
2. Add the distance and index of the relevant observation to an ordered collection of data.

- Sort this ordered collection containing distances and indices from smallest distance to largest (in ascending order).
- Select the first k entries in the sorted data collection (equivalent to the k nearest neighbours).
- Obtain the labels of the selected entries.
- If regression, return the mean of the k labels.
- If classification, return the mode (most frequent/common value) of the k labels

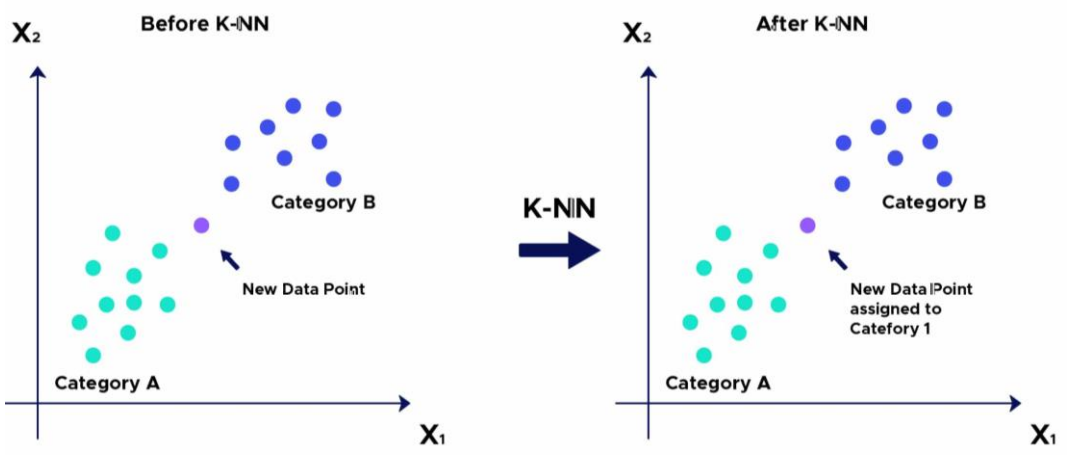


Figure 8: KNN Principles

The KNN algorithm uses various distance functions to compute the proximity amongst the data points.

There is no need to code these distances yourself, generally, Machine Learning libraries like Scikit Learn, perform these calculations internally. You just have to indicate the distance measure you want to use.

- **Euclidean distance:** calculates the square root of the sum of the square differences between the coordinates of two points. The formula says:

$$D_e(x, y) = \sqrt{\sum_{j=1}^{n}(x_j - y_j)^2}$$

Where: (xj, yj) are the coordinates of the points and De(x,y) is the distance between (x1, y1) and (x2, y2).

- **Manhattan distance**: calculates the sum of the absolute values of the differences between the coordinates of two points. The formula says:

$$D_m(x, y) = \sum_{i=1}^{k} | x_i - y_i |$$

- **Hamming distance**: the distance between two given points is the maximum difference between their coordinates on one dimension. The formula says:

$$D_h(x, y) = \sum_{i=1}^{k} | x_i - y_i |$$

If X=Y → D = 0
If X# Y → D = 1

Note that there are other distances depending on the use case of the algorithm, but the Euclidean distance remains the most used.

III.4. Choose the correct value for k

To select the right value of k for your data, we run the KNN algorithm several times with different values of k. Then we choose the k that reduces the number of errors encountered while maintaining the algorithm's ability to make accurate predictions when it receives new (previously unseen) data.

The choice of the K value to be used to make a prediction with KNN varies according to the dataset. In general, the fewer neighbours we use (a small K number), the more we are subject to underfitting. On the other hand, the more neighbours we use (a large number K), the more reliable our prediction will be. However, if we use a number K of neighbours with K=N and N being the number of observations, we run the risk of overfitting and therefore a model that generalises incorrectly on observations that it has not yet seen.

In KNN classification, a new data point is assigned to the most common class among its K nearest neighbors. If K is even, there is a possibility of a tie between

two or more classes, making it unclear which class to assign. Choosing an odd value for K reduces the likelihood of ties, ensuring a clear majority class in most cases.

The main disadvantage of KNN is that it slows down considerably as the volume of data increases, making it an impractical choice in a context where predictions need to be made quickly. In addition, some faster algorithms can produce more accurate classification and regression results.

Example

We consider two species: crocodiles and alligators. It is assumed that they can be distinguished by measuring the width of their mouths and the length of their bodies.

Mouth	Length	class
0.17	2.84	alligator
0.24	3.82	alligator
0.24	3.39	alligator
0.2	2.60	alligator
0.25	4.21	crocodile
0.47	4.64	crocodile
0.47	4.48	crocodile
0.49	4.9	crocodile
0.46	4.08	Crocodile
0.19	2.91	?

We add a new animal whose characteristics we know, but not its species: New : gueule = 0.19, longueur = 2.91 (animal)

What species does it belong to?

The distance separating this animal from the others is added to the previous data. We'll use the usual Euclidean distance here:

$$AB = \sqrt{(XB - XA)^2 + (YB - YA)^2}$$

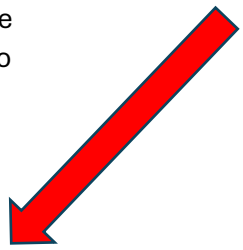
Mouth	Length	class
0.17	2.84	alligator
0.24	3.82	alligator
0.24	3.39	alligator
0.2	2.60	alligator
0.25	4.21	crocodile
0.47	4.64	crocodile
0.47	4.48	crocodile
0.49	4.9	crocodile

We calculate the distance of each observation from our animal (new)

Distance	Classe
0.072	alligator
0.911	alligator
0.482	alligator
0.310	alligator
1.301	crocodile
1.752	crocodile
1.594	crocodile
2.012	crocodile
1.200	crocodile

Distance	Class
0.072	alligator
0.310	alligator
0.482	alligator
0.911	alligator
1.200	crocodile
1.301	crocodile
1.594	crocodile
1.752	crocodile
2.012	crocodile

We then sort from the smallest to the largest value according to distance



New : Mouth = 0.19, Length = 2.91 (animal)

Choose **k=3** and keep only the three animals closest to our new animal:

Distance	Class
0.072	alligator
0.310	alligator
0.482	alligator

In this subset made up of three elements, the majority class is alligator, and this is the one we assign to our new animal. We chose a class because this is a classification problem.

For a regression problem: for example, predicting a person's weight as a function of height, we follow the same principle, only at the end we calculate the average of the values, let's look at the following example:

Height	Weight
160	80
150	60
10	20
12	25
140	?

We want to know the weight of a person whose height is 140.

So, we calculate the distance of each observation from the point (our point is size 140) and we sort these distances in ascending order, then we choose the lines according to the value of K, here k=2 (for example), so we'll have the lines in yellow.

As a result, the weight is the average of the weights of the two lines chosen (60+80)/2 =70

So, 140 has a weight of 70

Distance	Weight
10	60
20	80
128	25
130	20

III.5. KNN applications

- Comparison of people with similar financial characteristics for bank loans.

- Drawing up a profile to suggest appropriate films to subscribers.

- Classify a potential voter as ‘will vote’ or ‘will not vote’ for a particular candidate.

This list is far from exhausting.

In python, the KNN classification algorithm is implemented in the ***KNeighborsClassifier*** class in the neighbor's module. Before we can use the model, we need to instantiate the class into an object. This is when we will set any parameters of the model. The most important parameter of KNeighbor sClassifier is the number of neighbors.

In conclusion,

The KNN algorithm is a simple supervised machine learning algorithm that can be used to solve classification and regression problems. It is easy to implement and understand but has the major disadvantage of slowing down considerably as the size of the data used increases.

KNN searches for the distances between an ‘unknown’ and all the data in the training database, selects the specified number of examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or for the average of the labels (in the case of regression).

In the case of classification and regression, we saw that choosing the right K for our data was done by trying several Ks and choosing the one that worked best.

III.6. Exercises

Exercise 1

Suppose we have a classification problem which consists of determining the class to which new instances Xi. The value domain of the possible classes is [1,2,3].

Using the following knowledge base, determine by hand (or using a spreadsheet) the class of X6 whose values for the numerical attributes A1 to A5 are <3,12,4,7,8> using the k-nearest neighbour algorithm (K-NN) with K=1 then K=3.

Exercise 2

Let the points have the following coordinates :

A(1,6),B(2,6),C(3,1),D(4,2),E(6,0),F(7,5),G(7,3),H(10,3)

Using Euclidean distance, what are the two nearest neighbours of point P(5,5) ?

Chapter Four

Supervised Learning – Naïve Bayes-

Lecture Notes

IV.1 Introduction

When dealing with machine learning problems involving labeled training data, algorithms fall into two categories: classification and regression. Delving a little deeper, one of the most basic algorithms you will come up to is the Naive Bayes algorithm.

Naïve Bayes Classifier is a popular algorithm in Machine Learning. It is a Supervised Learning algorithm used for classification. It is particularly used for text classification (such as SPAM).

Naïve Bayes classifies a set of observations according to rules determined by the algorithm itself. This classification tool must first be trained on a training dataset that shows the expected class based on the inputs.

During the training phase, the algorithm develops its classification rules on this dataset and then applies them to the classification of a prediction dataset.

The naive Bayesian classifier implies that the classes in the training dataset are known and provided, hence the supervised nature of the tool.

IV.2 Bayes' theorem

The naive Bayes classifier is based on Bayes' theorem. This theorem is based on conditional probabilities (what is the probability of an event occurring knowing that another event has already occurred).

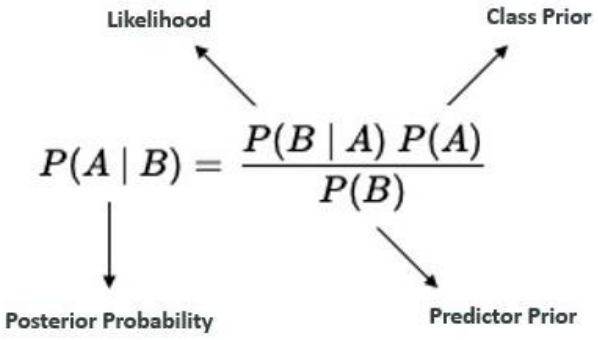
Let's take the following example: suppose we have a class of high school students. Let A and B be the following two events:

- Event A: The pupil is a girl.
- Event B: the pupil takes German.

What is the probability of randomly choosing a girl who practises German?
Bayes' theorem can be used to calculate this kind of probability.

Let P be the probability of an event.

The term P(A\B): the probability that event A will occur knowing that event B has already occurred (A: Evidence, B: Outcome).



— A, B : events

— P(A\B) : probability of A knowing that B is true

— P(B\A) **:** probability of B knowing that A is true

— P(A), P(B): independent probabilities of A and B

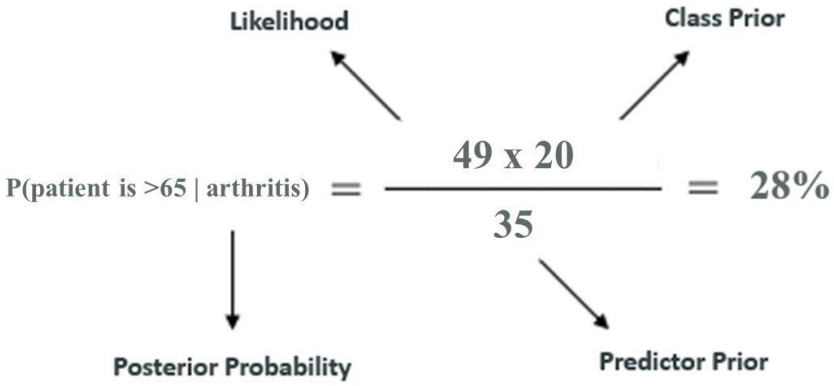
We consider now another example – Say the likelihood of a person having Arthritis if they are over 65 years of age is 49%.

Check the above stats at: Centre for Disease Control and Prevention

Now, let's assume the following:

- Class Prior: The probability of a person stepping in the clinic being >65-year-old is 20%
- Predictor Prior: The probability of a person stepping into the clinic having Arthritis is 35%

What is the probability that a person is >65 years given that he has Arthritis using the bayes theorem?



We will now take the example of the pupils at the lycée who speak German:

	Girls	Boys	Total
German	10	7	17
Other language	4	9	13
Total	14	16	30

Let's calculate the following probability: What is the probability of randomly drawing a student who speaks German, knowing that she is a girl?

According to Bayes' formula, we have the following:

Consider Ω the set of high school students in our example, so cardinal (Ω) = 30

Remember: Cardinal of a set = number of elements in the set

$$P(Allemand \mid Fille) = P(B \mid A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A|B) * P(B)}{P(A)}$$

- $P(A)$ est la probabilité de prendre au hasard une fille de la population des élèves de la classe. On appelle $P(A)$ **la probabilité antérieure (*prior probability*).**

$$P(A) = \frac{cardinal(A)}{cardinal(\Omega)} = \frac{14}{30} \approx 0.4666$$

$$\frac{cardinal(B \cap A)}{cardinal(\Omega)} = \frac{10}{30} \approx 0.3333$$

Ce qui donne :

$$P(B \mid A) = \frac{\frac{10}{30}}{\frac{14}{30}} \approx \frac{0.3333}{0.4666} \approx 0.7143$$

For this example, we applied Bayes' theorem with a single predictor variable (Evidence): Namely the gender of the student (Girl). In real applications of Naive Bayes, the result (Outcome) is calculated based on several variables rather than a single variable.

Applying Bayes' theorem to several variables makes the calculation more complex. To get round this, one approach is to consider these variables independently of each other.

This is a strong assumption. Generally, the predictive variables are linked together. The term ‘naive’ comes from the fact that this independence of variables is assumed.

If we take the example of classifying emails into spam and non-spam, Naïve Bayes will base its classification on the frequency of occurrence of words in the email.

When classifying, the algorithm will assume that the words in the email 'appear' independently of each other. Obviously, from a linguistic and semantic point of view, this assumption is false.

IV.3 Limitations of Naïve Bayes

- Assumes that all the features are independent, which is highly unlikely in practical scenarios.
- Unsuitable for numerical data.
- The number of features must be equal to the number of attributes in the data for the algorithm to make correct predictions.
- Encounters 'Zero Frequency' problem: If a categorical variable has a category in the test dataset that wasn't included in the training dataset, the model will assign it a 0 probability and will be unable to make a prediction. This problem can be resolved using smoothing techniques which are out of scope of this article.
- Computationally expensive when used to classify a large number of items.

Example

Let's be a set of individuals with the characteristics : cheveux, taille, poids, crème solaire, on désire classer ces individus selon la classe Classe.

N°	Cheveux	Taille	Poids	Crème Solaire	Classe
1	Blond	Moyenne	Léger	Non	Coup de soleil
2	Blond	Grande	Moyen	Oui	Bronzé
3	Brun	Petite	Moyen	Oui	Bronzé
4	Blond	Petite	Moyen	Non	Coup de soleil
5	Roux	Moyenne	Lourd	Non	Coup de soleil
6	Brun	Grande	Lourd	Non	Bronzé
7	Brun	Moyenne	Lourd	Non	Bronzé
8	Blond	Petite	Léger	Oui	Bronzé

1. Give the decision model deduced from this database using Bayesian classification.
2. Find the classes of the following examples:

N°	Cheveux	Taille	Poids	Crème Solaire
1	?	Petite	?	Oui
2	?	Grande	Moyen	?
3	Brun	?	?	Non
4	?	?	Lourd	?

Solution:

1. Let's start by filling in this table, which summarises the probabilities of the classes in relation to the attributes:

Attributs	**Valeurs**	**P(classe =Oui = Coup de soleil) = 3/8**	**P(classe =Non = Bronzé) = 5/8**
Cheveux	**Blond = 4**	**2/3**	**2/5**
	Brun = 3	**0/3**	**3/3**
	Roux = 1	**1/3**	**0/5**
Taille	**Petite = 3**	**1/3**	**2/5**
	Moyenne = 3	**2/3**	**1/5**
	Grande = 2	**0/3**	**2/5**
Poids	**Leger = 2**	**1/3**	**1/5**
	Moyen = 3	**1/3**	**2/5**
	Lourd = 3	**1/3**	**2/5**
Crème solaire	**Oui = 3**	**0/3**	**3/5**
	Non = 5	**3/3**	**2/5**

Note that our table contains zero values, in which case we will use the Laplace estimator.

Important: When the headcount is 0 (for a given class, and for a given attribute a): add a value (for example 1) to each count on the headcount table (for the class in question). Then consider that there is k more examples (k: number of possible values a).

The general idea is the following:

— Add a value μ to each denominator for the attribute under consideration a and the class under consideration.

— Add μ/k to the number of people associated with each value of the attribute under consideration and the class under consideration. This quantity, μ/k of the attribute considered, can be seen as an a priori probability of observing each of the values of the attribute.

Using Laplace's estimator, we obtain the following values:

		P(Classe = Oui = Coup de soleil) = 3/8	P(Classe= Non = Bronzé)=5/8
Cheveux	Blond = 4	2/3 = > (2+1)/(3+3)=3/6	2/5 => (2+1)/(5+3)=3/8
	Brun = 3	O/3 => (0+1)/(3+3) = 1/6	3/5 => (3+1)/(5+3) = 4/8
	Roux = 1	1/3 => (1+1)/(3+3) = 2/6	0/5 => (0+1)/(5+3) = 1/8
Taille	Petite = 3	1/3 => (1+1)/(3+3) = 2/6	2/5
	Moyenne =3	2/3 => (2+1)/(3+3) = 3/6	1/5
	Grande = 2	0/3 => (0+1)/(3+3) = 1/6	2/5
Poids	Léger = 2	1/3	1/5
	Moyen =3	1/3	2/5
	Lourd = 3	1/3	2/5
Crème Solaire	Oui = 3	0/3 =>(0+1)/(3+2) =1/5	3/5
	Non =5	3/3 => (3+1)/(3+2) = 4/5	2/5

The table above shows the Bayesian model we will use to make predictions.

Note that the step of calculating the Laplace estimator only comes into play if we have zero probabilities and concerns the range of the attribute in question.

2. Find the classes for the following examples (these data do not belong to the dataset):

N°	Cheveux	Taille	Poids	Crème Solaire
1	?	Petite	?	Oui
2	?	Grande	Moyen	?
3	Brun	?	?	Non
4	?	?	Lourd	?

Bayes' theorem : $P(X\backslash Y) = \frac{P(Y\backslash X)*P(X)}{P(Y)}$

X(? , petite, ?, Oui)

To classify X in one class or another, we will calculate We will calculate :

P(Classe = Oui\X) = $\frac{P(X\backslash Classe=Oui)*P(Classe=Oui)}{P(X)}$

P(Classe = Non\X) = $\frac{P(X\backslash Classe=Non)*P(Classe=Non)}{P(X)}$

Et on choisira la plus grande probabilité.

- **Class coup de soleil :**

(1) P(Classe = Oui\X) = $\frac{P(X\backslash Classe=Oui)*P(Classe=Oui)}{P(X)}$

(2) P(Classe = Non\X) = $\frac{P(X\backslash Classe=Non)*P(Classe=Non)}{P(X)}$

- **Class bronzé :**

(1) P(Classe = Oui\X) = $\frac{P(X\backslash Classe=Oui)*P(Classe=Oui)}{P(X)}$

(2) P(Classe = Non\X) = $\frac{P(X\backslash Classe=Non)*P(Classe=Non)}{P(X)}$

(1) P(Classe = Oui\X) = $\frac{P(X\backslash Classe=Oui)*P(Classe=Oui)}{P(X)}$ =

$$\frac{P(?\backslash classe=Oui)*P(petite\backslash classe=Oui)*P(?\backslash classe=Oui)*P(Oui\backslash classe=Oui)*P(Classe=Oui)}{P(?)*P(petite)*P(?)*P(Oui)}$$

We calculate all the probabilities and take the highest probability.

N°	Cheveux	Taille	Poids	Crème Solaire	
1	?	Petite	?	Oui	← Bronzé
2	?	Grande	Moyen	?	← Bronzé
3	Brun	?	?	Non	← Bronzé
4	?	?	Lourd	?	← Bronzé

Exercise :

A bank has the following information on a group of customers:

client	M	A	R	E	I
01	moyen	moyen	village	oui	oui
02	élevé	moyen	bourg	non	non
03	faible	âgé	bourg	non	non
04	faible	moyen	bourg	oui	oui
05	moyen	jeune	ville	oui	oui
06	élevé	âgé	ville	oui	non
07	moyen	âgé	ville	oui	non
08	faible	moyen	village	non	non

The customer attribute indicates the customer's number; the M attribute indicates the average credit on the customer's account; the A attribute gives the age group; the R attribute describes the customer's location; the E attribute has the value yes if the customer has a level of education higher than the baccalaureate; the I attribute (the class) indicates whether the customer carries out his account management operations via the Internet.

1. Give the decision model deduced from this database using naive Bayesian classification.
2. Find the classes for the following examples:

client	M	A	R	E
01	?	âgé	?	oui
02	élevé	?	ville	?
03	faible	?	?	?
04	?	moyen	bourg	?

IV.4 Conclusion

Today, Naïve Bayes is a renowned algorithm with applications in a wide range of fields. Naïve Bayesian classification achieves remarkable results in many everyday applications, making it the algorithm of choice among Machine Learning tools.

Among its strengths are its rapid learning curve, which does not require a large volume of data, the probability calculations are not very costly, and its extreme speed of execution compared with other more complex methods.

However, the Naïve Bayes Classifier algorithm assumes the independence of the variables: this is a strong assumption that is violated in most real cases.

Chapter Five

UnSupervised Learning – Kmeans –

Lecture Notes

V.1 Introduction

Supervised learning is a learning technique in which the machine is shown X, y examples of what it needs to learn. Unsupervised learning, on the other hand, involves providing the machine with X data only, and asking it to analyse the structure of this data to learn how to perform certain tasks on its own.

Clustering is one of the most popular applications of unsupervised learning. The principle is to let the machine learn to sort data according to their similarities (and therefore by analysing only features X). Well-known algorithms : K-Means.

V.2 Clustering

This unsupervised classification method brings together a set of learning algorithms whose aim is to group together unlabelled data with similar properties.

Clustering is used when it is costly to label data. It is, however, a mathematically ill-defined problem: different metrics and/or different representations of the data will result in different groupings without any of them necessarily being better than another.

So, the clustering method needs to be chosen carefully depending on the expected result and the intended use of the data.

V.3 KMeans

This is one of the most widely used clustering algorithms. It is used to analyse a dataset characterised by a set of descriptors, to group ‘similar’ data into groups (or clusters).

The similarity between two data sets can be inferred from the ‘distance’ between their descriptors; two very similar data sets are two data sets whose descriptors are very close. This definition allows us to formulate the data partitioning problem as the search for K ‘prototype data’, around which the other data can be grouped.

These prototype data are called centroids; in practice, the algorithm associates each data item with its closest centroid, to create clusters. On the other hand, the averages

of the descriptors of the data in a cluster define the position of their centroid in the descriptor space: this is the origin of the name of this algorithm (K-means).

After initialising its centroids by taking random data from the dataset, K-means alternates these two steps several times to optimise the centroids and their clusters:

1. Group each object around the nearest centroid.
2. Replace each centroid according to the average of the descriptors in its group.

After a few iterations, the algorithm finds a stable division of the dataset: we say that the algorithm has converged.

Like all algorithms, K-means has advantages and disadvantages: it is simple, fast and easy to understand; however, it cannot find groups with complex shapes.

Here the key Objectives of K-Means Clustering :

- Organizing Similar Data Points K-Means clustering focuses on grouping data points with shared characteristics into distinct clusters. Whether used for customer segmentation or image analysis, this method helps uncover underlying patterns in the dataset.

- Minimizing Variability Within Clusters A key goal is to ensure that data points within each cluster are closely positioned around the cluster's centroid. By reducing internal dispersion, the algorithm forms compact and well-defined clusters, improving result accuracy.

- Enhancing Separation Between Clusters K-Means also strives to maximize the distance between different clusters, ensuring clear distinctions between groups. This separation prevents overlapping and provides better insights into the structure of the data.

V.4 How K Means works

The K-means algorithm identifies a number of centroids in a dataset, a centroid being: the arithmetic mean of all the data points belonging to a particular cluster'.

The algorithm then assigns each data point to the nearest cluster, trying to keep the clusters as small as possible (the term ‘means’ in K-means refers to the average of the data or finding the centroid).

At the same time, K-means tries to keep the other clusters as different as possible.

K-means algorithm

Input *:*

K the number of clusters to form

The dataset

Begin

Randomly select K points (one row of the data matrix). These points are the cluster centres (called centroids).

Repeat

Assign each point (element of the data matrix) to the cluster whose centre it is closest to.

Recalculate the centre of each cluster and modify the centroid.

Until *Convergence OR (stabilisation of the total inertia of the population)*

End*.*

Note: The convergence of the K-Means algorithm can be one of the following conditions:

- Several iterations fixed in advance, in which case K-means will perform the iterations and stop regardless of the shape of the composed clusters.
- Stabilisation of cluster centres (centroids no longer move during iterations).

A point is assigned to a cluster according to its distance from the various centroids. In addition, the point will be assigned to a cluster if it is closer to its centroid (minimum distance). Finally, the distance between two points in the case of K-Means is calculated using the methods described in the ‘notion of similarity’ section.

Example

There are 8 points: from A1 to A8, with the following coordinates:

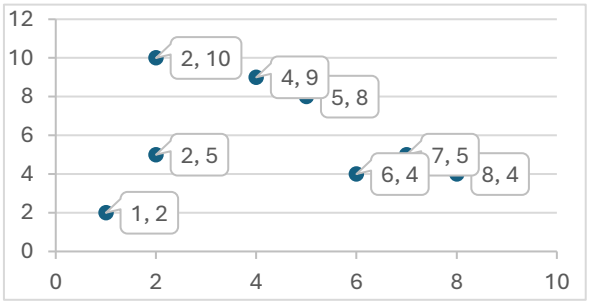
A1= (2.10), A2= (2.5), A3= (8.4), A4= (5.8), A5= (7.5), A6= (6.4), A7= (1.2), A8= (4.9)

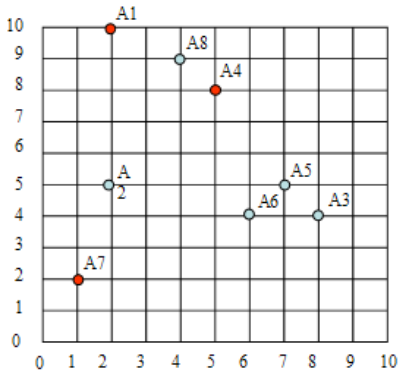
We want to apply the K-means algorithm to format 3 clusters, initially choosing A1, A4 and A7 as cluster centres.

Show all the calculation steps until you reach the result. Draw the intermediate states and the result.

Solution

Here's the point cloud





Let μ1, μ2 and μ3 be the centres of gravity of clusters (respectively) cluster1, cluster2 and cluster3.

μ1 = A1

μ2 = A4

μ3 = A7

- Iteration 1 :

Start by calculating the distance of points A1, ...A8 from the centres μi (Euclidean distance)

- For A1 (distance between A1 and the centres μ1, μ2 and μ3)

d(A1, μ1)=0

d(A1, μ2)= sqrt(13)

d(A1, μ3)= sqrt(65)

A1 ∈ cluster1 (we choose the smallest value, A1 must belong to cluster1)

- For A2 (distance between A2 and the centres μ1, μ2 and μ3)

d(A2, μ1)=sqrt(25)= 5

d(A2, μ2)= sqrt(18) = 4.24

d(A2, μ3)= sqrt(10) = 3.16

A2 ∈ cluster3 (choose the smallest value, A2 must belong to cluster3)

- For A3: (distance between A3 and the centres μ1, μ2 and μ3)

d(A3, μ1)= sqrt(36) = 6

d(A3, μ2)= sqrt(25) = 5

d(A3, μ3)= sqrt(53) = 7.28

A3 ∈ cluster2 (choose the smallest value, A3 must belong to cluster2)

- For A4: (distance between A4 and the centres μ1, μ2 and μ3)

d(A4, μ1)= sqrt(13)

d(A4, μ2)=0

d(A4, μ3)= sqrt(52)

A4 ∈ cluster2 (choose the smallest value, A4 must belong to cluster2)

- For A5: (distance between A5 and the centres μ1, μ2 and μ3)

d(A5, μ1)=sqrt(50)=7.07

d(A5, μ2)=sqrt(13)=3.60

d(A5, μ3)=sqrt(45)=6.70

A5 ∈ cluster2 (choose the smallest value, A5 must belong to cluster2)

- For A6: (distance between A6 and the centres μ1, μ2 and μ3)

d(A6, μ1)=sqrt(52)=7.21

d(A6, μ2)=sqrt(17)=4.12

d(A6, μ3)=sqrt(29)=5.38

A6 ∈ cluster2 (choose the smallest value, A6 must belong to cluster2)

- For A7: (distance between A7 and the centres μ1, μ2 and μ3)

d(A7, μ1)=sqrt(65)=7.21

d(A7, μ2)=sqrt(52)=4.12

d(A7, μ3)=0

A7 ∈ cluster3(choose the smallest value, A7 must belong to cluster3)

- For A8: (distance between A8 and the centres μ1, μ2 and μ3)

d(A8, μ1)=sqrt(5)

d(A8, μ2)=sqrt(2)

d(A8, μ3)=sqrt(58)

A8 ∈ cluster2(we choose the smallest value, A8 must belong to cluster2)

So the new clusters are : Cluster1: {A1}

Cluster2: {A3, A4, A5, A6, A8}

Cluster3: {A2, A7}

Now the cluster centres will be updated:

μ1= (2, 10) here we have a single point

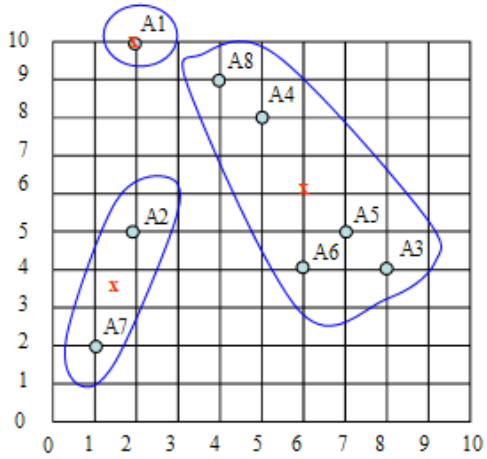
μ2= ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6, 6), here we calculate the average (hence the name means)

μ3= ((2+1)/2, (5+2)/2) = (1.5, 3.5) here we calculate the average (hence the name means)

μ1= (2, 10)

μ2= (6, 6)

μ3= (1.5, 3.5)



- **Iteration 2**: we will follow the same principle of calculating the distance from the points to the centres.

We found the new clusters :

cluster1: {A1, A8}

cluster2: {A3, A4, A5, A6}

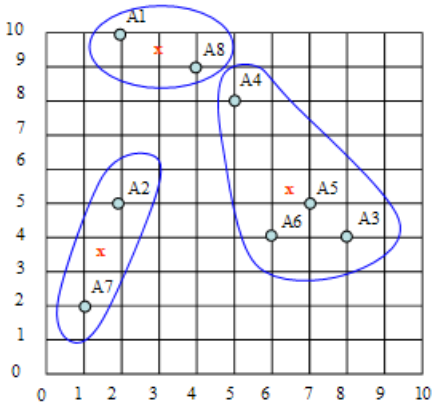
cluster3: {A2, A7}

And the new centres :

μ1=(3, 9.5),

μ2=(6.5, 5.25)

μ3=(1.5, 3.5).



- **Iteration 3**: we will follow the same principle of calculating the distance from points to centres.

We found the new clusters :

cluster1: {A1, A4, A8}

cluster2: {A3, A5, A6}

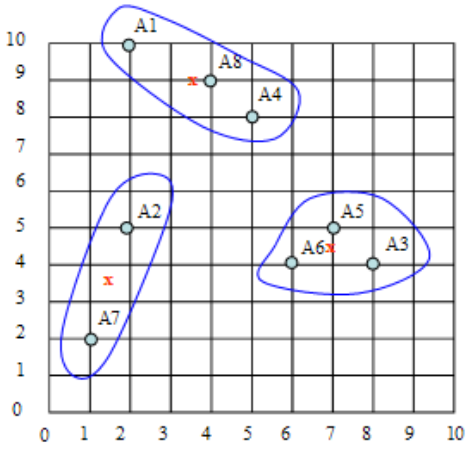
cluster3: {A2, A7}

As well as the new centres :

μ1=(3.66, 9),

μ2=(7, 4.33)

μ3=(1.5, 3.5)



- **Iteration 4**: we will follow the same principle of calculating the distance from point to centres.

The new clusters are :

cluster1: {A1, A4, A8}

cluster2: {A3, A5, A6}

cluster3: {A2, A7}

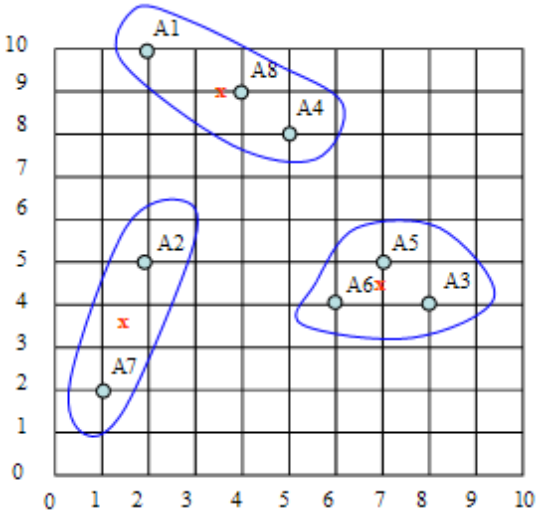
As well as the new centres :

$\mu1=(3.66, 9)$,

$\mu2=(7, 4.33)$

$\mu3=(1.5, 3.5)$

We stop at this stage because the centres $\mu1$, $\mu2$ and $\mu3$ do not change and we have obtained the same clusters.



V.5 Limits of KMeans

Despite its many advantages, K-Means has certain limitations:

- What problems do we have to deal with if we want to implement a clustering method?
- Nature observations: Binary, textual, numerical data, etc.?
- Notion of similarity (or dissimilarity) between observations
- Définition of a cluster
- Evaluation of the validity of a cluster
- Nombre of clusters that can be identified in the data
- Comparaison of different clustering results

Exercise 1

We want to perform clustering using the K-means method for the data set D={1,3,6,8,10,11,12,22,24,26,31,57}, assuming that the initial centres are: 5,10,25,30 for the 4 clusters.

Questions :

1. Knowing that the distance used is a Euclidean distance, apply the K-means algorithm for a single iteration.
2. How many iterations are needed to achieve stabilisation?
3. Give the result of K-means clustering.
4. What are the final cluster centres?

Exercise 2

A shopping mall wants to analyze customer purchasing behavior to create targeted marketing campaigns. The mall's management has collected data on 100 customers, including:

- Annual Income (in $1000s)
- Spending Score (a measure of customer engagement, from 1 to 100)

The goal is to group similar customers using K-Means clustering to identify different spending patterns.

Explain how to do this task

Chapter Six

Supervised Learning – Decision Tree –

Lecture Notes

VI.1 Introduction

A decision tree is a non-parametric supervised learning algorithm used for both classification and regression tasks. It has a hierarchical, tree-like structure, consisting of a root node, branches, internal nodes and leaf nodes.

A decision tree is a graphical representation (figure 9) of different options for solving a problem and shows how different factors are related. It has a hierarchical tree structure starts with one main question at the top called a node which further branches out into different possible outcomes where:

- **Root Node** is the starting point that represents the entire dataset.
- **Branches**: These are the lines that connect nodes. It shows the flow from one decision to another.
- **Internal Nodes** are Points where decisions are made based on the input features.
- **Leaf Nodes**: These are the terminal nodes at the end of branches that represent final outcomes or predictions

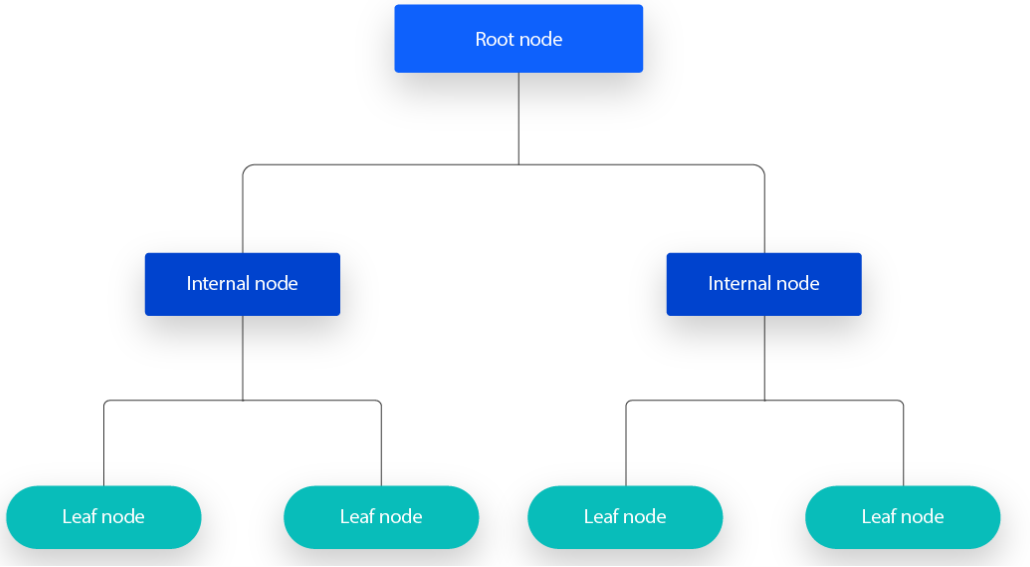
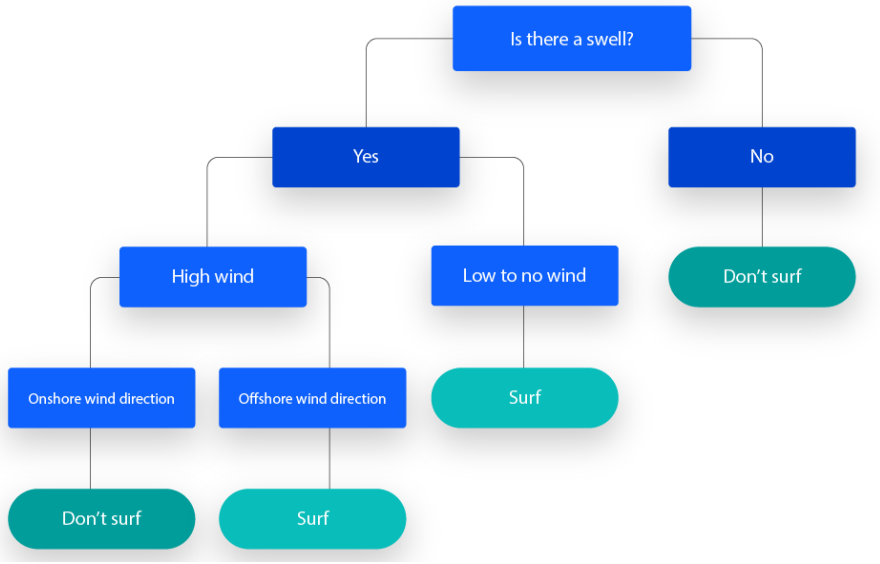


Figure 9 : Decisoon Tree structure

For example, let's say you're trying to decide whether to go surfing. You can use the following decision rules to make your choice:



Decision This type of flowchart structure also creates a representation of decision making, allowing different groups within an organisation to better understand why a decision has been made.

Decision tree learning employs a divide-and-conquer strategy by performing a greedy search to identify optimal splitting points within a tree. This splitting process is then repeated in a top-down, recursive manner until all or most of the data sets have been classified under specific class labels. Whether or not all data points are classified as homogeneous sets depends largely on the complexity of the decision tree.

For small trees, it is easier to achieve pure leaf nodes, i.e. data points in a single class. However, as a tree grows, it becomes increasingly difficult to maintain this purity, usually resulting in an insufficient number of data points from a given sub-tree. When this happens, it is called data fragmentation and can often lead to overfitting.

As a result, decision trees prefer small trees, in line with Occam's Razor's principle of parsimony: 'entities should not be multiplied more than necessary'. In other words, decision trees should only add complexity, when necessary, as the simplest explanation is often the best.

To reduce complexity and avoid overfitting, pruning is commonly used, a process that removes branches that split into less important functionalities. The fit of the

model can then be assessed using cross-validation. Decision trees can also retain their accuracy by forming an ensemble using a random forest algorithm; this classifier predicts more accurate results, particularly when individual trees are not correlated with each other.

VI.2 Decision Tree Types

We have mainly two types of decision tree based on the nature of the target variable: classification trees and regression trees.

- Classification trees: They are designed to predict categorical outcomes, which means they classify data into different classes. They can determine whether an email is "spam" or "not spam" based on various features of the email.
- Regression trees : These are used when the target variable is continuous. It predict numerical values rather than categories. For example, a regression tree can estimate the price of a house based on its size, location, and other features.

VI.3 How do you select the best attribute at each node?

Although there are several ways to select the best attribute at each node, the two methods, information gain and Gini impurity, are popular splitting criteria for decision tree models. They help to assess the quality of each test's condition and the extent to which it will be able to classify samples.

It is difficult to explain information gain without first mentioning entropy. Entropy is a concept from information theory that measures the impurity of sample values. It is defined by the following formula, where :

$$\text{Entropy}(S) = -\sum_{c \in C} p(c)\log_2 p(c)$$

- S represents the data set in which the entropy is calculated.
- c represents the classes in the set, S.
- p(c) is the proportion of data points belonging to class c relative to the total number of data points in the set, S.

Entropy values can be between 0 and 1. If all the samples in the dataset S belong to a single class, the entropy is zero. If half of the samples are classified in one class and half in another, the entropy will be less than or equal to 1. To select the best feature to divide and find the optimal decision tree, the attribute with the smallest amount of entropy should be used.

The information gain represents the difference in entropy before and after a split on a given attribute. The attribute with the highest information gain will produce the best split, as it is best able to classify the training data according to its target classification. The information gain is generally represented by the following formula, where :

- a represents a specific attribute or class label.
- Entropy(S) is the entropy of the dataset, S.
- |Sv| / |S| is the proportion of values in Sv relative to the number of values in the dataset, S.
- Entropy(Sv) is the entropy of the dataset, Sv.

Let's take an example to illustrate these concepts. Suppose we have the following arbitrary data set:

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

For this dataset, the entropy is 0.94. This can be calculated by finding the proportion of days when 'Play Tennis' is 'Yes', which is 9/14, and the proportion of days when 'Play Tennis' is 'No', which is 5/14. These values can then be incorporated into the entropy formula above.

$$\text{Entropy (tennis)} = -(9/14)\log2(9/14) - (5/14)\log2(5/14) = 0.94$$

We can then calculate the information gain for each of the attributes individually. For example, the information gain for the 'Humidity' attribute would be as follows:

$$\text{Gain (Tennis, Humidity)} = (0.94)-(7/14)*(0.985) - (7/14)*(0.592) = 0.151$$

As a reminder,

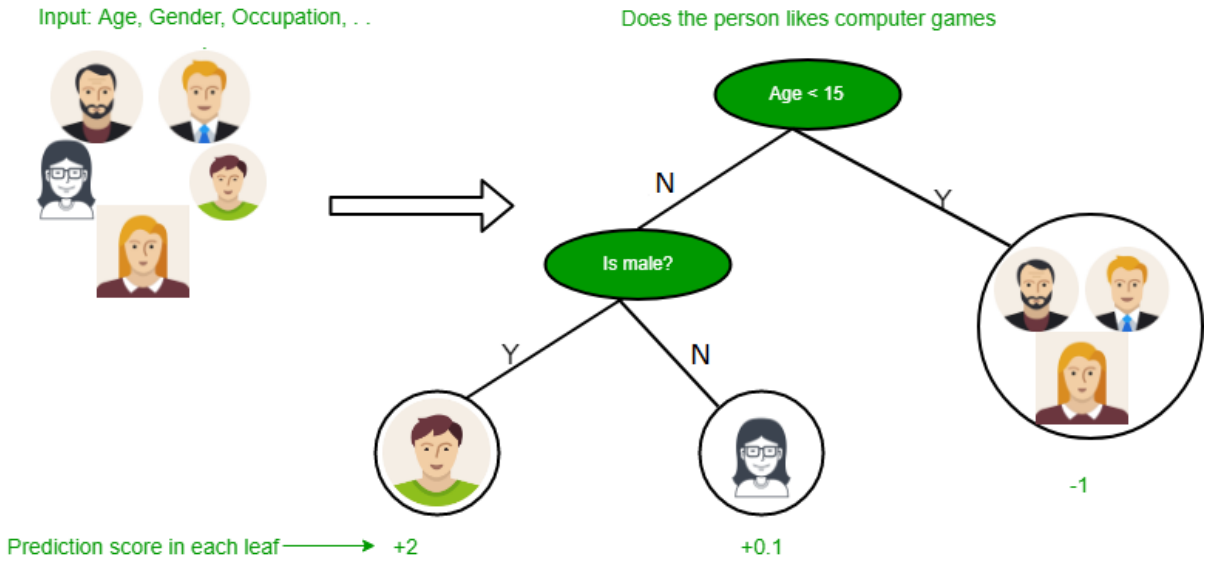
- 7/14 represents the proportion of values where the humidity is equal to 'high' in relation to the total number of humidity values. In this case, the number of values where the humidity is equal to 'high' is the same as the number of values where the humidity is equal to 'normal'.
- 0.985 is the entropy when the humidity is 'high'.
- 0.59 is the entropy when humidity = 'normal'.

Next, repeat the information gain calculation for each attribute in the table above and select the attribute with the highest information gain as the first split point in the decision tree. In this case, Outlook generates the highest information gain. The process is then repeated for each sub-tree.

The Gini impurity is the probability of misclassifying a random data point in the dataset if it were labelled according to the class distribution of that dataset. Similar to entropy, if it is defined, S is pure, i.e. belonging to a class), then its impurity is zero. This is reflected in the following formula:

$$\text{Gini Impurity} = 1 - \sum_i (p_i)^2$$

Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.



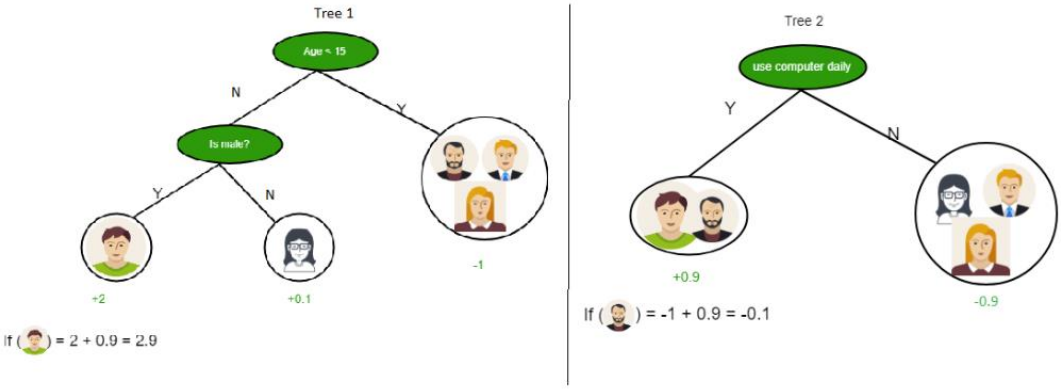
Example: Predicting Whether a Person Likes Computer Games

Imagine you want to predict if a person enjoys computer games based on their age and gender. Here's how the decision tree works:

1. **Start with the Root Question (Age):**
    - The first question is: *"Is the person's age less than 15?"*
    - If **Yes**, move to the left.
    - If **No**, move to the right.

2. **Branch Based on Age:**
    - If the person is **younger than 15**, they are likely to enjoy computer games (+2 prediction score).
    - If the person is **15 or older**, ask the next question: *"Is the person male?"*

3. **Branch Based on Gender (For Age 15+):**
    - If the person is **male**, they are somewhat likely to enjoy computer games (+0.1 prediction score).
    - If the person is **not male**, they are less likely to enjoy computer games (-1 prediction score)



Example: Predicting Whether a Person Likes Computer Games Using Two Decision Trees

Tree 1: Age and Gender

1. The first tree asks two questions:
    - **"Is the person's age less than 15?"**
        - If **Yes**, they get a score of **+2**.
        - If **No**, proceed to the next question.
    - **"Is the person male?"**
        - If **Yes**, they get a score of **+0.1**.
        - If **No**, they get a score of **-1**.

Tree 2: Computer Usage

1. The second tree focuses on daily computer usage:
    - **"Does the person use a computer daily?"**
        - If **Yes**, they get a score of **+0.9**.
        - If **No**, they get a score of **-0.9**.

Combining Trees: Final Prediction. The final prediction score is the sum of scores from both trees

VI.4 Decision Tree Limits

Although decision trees can be used for a wide range of applications, other algorithms generally perform better. That said, decision trees are particularly useful for data mining and knowledge discovery tasks.

- We cite here some limits : Overfitting: Overfitting occurs when a decision tree captures noise and details in the training data and it perform poorly on new data.
- Instability: instability means that the model can be unreliable slight variations in input can lead to significant differences in predictions.
- Bias towards Features with More Levels: Decision trees can become biased towards features with many categories focusing too much on them during decision-making. This can cause the model to miss out other important features, led to less accurate predictions.

VI.5 Exercises

Exercise 1

A bank wants to improve its loan approval process by predicting whether a customer is likely to repay a loan or default based on their financial profile. The bank has collected historical data on previous loan applicants, including:

- Age (years)
- Annual Income (in $1000s)
- Credit Score (rating from 300 to 850)
- Loan Amount Requested (in $1000s)
- Repayment Status (Target Variable: "Repaid" or "Defaulted")

Your task is to use a Decision Tree Classifier to build a model that helps the bank decide whether to approve or reject future loan applications.

Exercise 2

Consider the following table, which shows a training set drawn randomly from a database of a company's customer purchases.

Id	Âge	Salaire	Étudiant	A acheté
1	Petit	Haut	Non	Non
2	Petit	Haut	Non	Non
3	Moyen	Haut	Non	Oui
4	Sénior	Moyen	Non	Oui
5	Sénior	Bas	Oui	Oui
6	Sénior	Bas	Oui	Non
7	Moyen	Bas	Oui	Oui
8	Petit	Moyen	Non	Non
9	Petit	Bas	Oui	Oui
10	Sénior	Moyen	Oui	Oui
11	Petit	Moyen	Oui	Oui
12	Moyen	Moyen	Non	Oui
13	Moyen	Haut	Oui	Oui
14	Sénior	Moyen	Non	Non

We want to construct a decision tree to predict the class (A purchased) of a new customer, based on this table. Use the information gain technique to choose the attribute at the root of the of the tree.

Where: m is the number of classes present in D, v is the number of distinct values of an attribute A.

Q.1 - Using the maximum information gain, determine only the root attribute of the tree (with the maximum gain) and then deduce the final tree without making any further calculations.

Q.2 - Use the decision tree constructed to classify and calculate the accuracy on the following test set:

Id	Âge	Salaire	Étudiant	A acheté
1	Petit	Haut	Non	Non
2	Sénior	Bas	Oui	Non
3	Moyen	Bas	Oui	Oui
4	Petit	Bas	Oui	Oui
5	Moyen	Moyen	Non	Oui
6	Sénior	Moyen	Non	Non

Chapter Seven

Supervised Learning – Random Forest –

Lecture Notes

VII.1 Introduction

The random forest algorithm is a supervised classification algorithm. As the name suggests, this algorithm creates the forest with several trees.

In general, the more trees in the forest the more robust the forest looks . In the same way in the random forest classifier, the higher the number of trees in the forest gives the higher the accuracy results.

If you know the decision tree algorithm. You might be thinking are we creating more number of decision trees and how can we create more number of decision trees. As all the calculation of nodes selection will be the same for the same dataset.

To model a greater number of decision trees to create the forest you are not going to use the same apache of constructing the decision with information gain or Gini index approach.

Here are some random forest algorithm advantages :

- The same random forest algorithm or the random forest classifier can use for both classification and the regression task.
- Random forest classifier will handle the missing values.
- When we have more trees in the forest, a random forest classifier won't overfit the model.
- Can model the random forest classifier for categorical values also.

Let's see this example : Imagine you are trying to predict whether a customer will buy a product based on their past behavior. Instead of relying on just one decision tree, Random Forest creates multiple decision trees and combines their results to make a more accurate prediction.

How It works :

1. Multiple Trees: The algorithm builds several decision trees using different random subsets of the data.
2. Voting System: For classification tasks (e.g., "Buy" or "Not Buy"), each tree gives its own prediction, and the most common result wins.
3. Averaging for Accuracy: In regression tasks (predicting numbers like sales or prices), the final output is the average of all three predictions.

VII.2 Decision Tree basis

The decision tree concept is more to the rule-based system. Given the training dataset with targets and features, the decision tree algorithm will come up with some set of rules. The same rules can be used to perform the prediction on the test dataset.

Suppose you would like to predict that your daughter will like the newly released animation movie or not. To model the decision tree, you will use the training dataset like the animated cartoon characters your daughter liked in the past movies.

Therefore, once you pass the dataset with the target as your daughter will like the movie or not to the decision tree classifier. The decision tree will start building the rules with the characters your daughter likes as nodes and the targets like or not as the leaf nodes. By considering the path from the root node to the leaf node. You can get the rules.

The simple rule could be if some x character is playing the leading role, then your daughter will like the movie. You can think of a few more rules based on this example.

Then to predict whether your daughter will like the movie or not. You just need to check the rules which are created by the decision tree to predict whether your daughter will like the newly released movie or not.

In decision tree algorithm calculating these nodes and forming the rules will happen using the information gain and Gini index calculations.

In a random forest algorithm, instead of using information gain or Gini index for calculating the root node, the process of finding the root node and splitting the feature nodes will happen randomly.

Next, you are going to learn why random forest algorithms are used. When we have other classification algorithms to play with.

VII.3. How Random Firest Works

The random Forest algorithm works in several steps:

- Random Forest builds multiple decision trees using random samples of data. Each tree is trained on a different subset of data which makes each tree unique.
- When creating each tree, the algorithm randomly selects a subset of features or variables to split the data rather than using all available features at a time. This adds diversity to the trees.
- Each decision tree in the forest makes a prediction based on the data it was trained on. When making final predictions, random forest combines the results from all the trees.
    - For classification tasks the final prediction is decided by a majority vote. This means that the category predicted by most trees is the final prediction.
    - For regression tasks the final prediction is the average of the predictions from all the trees.
- The randomness in data samples and feature selection helps to prevent the model from overfitting by making the predictions more accurate and reliable.

Random Forest has the following a assumptions :

- Each tree makes its own decisions: Every tree in the forest makes its own predictions without relying on others.
- Random parts of the data are used: Each tree is built using random samples and features to reduce mistakes.
- Enough data is needed: Sufficient data ensures the trees are different and learn unique patterns and variety.
- Different predictions improve accuracy: Combining the predictions from different trees leads to a more accurate result.

VII.4. Random Forest in pratice

There are mainly four sectors where Random Forest mostly used:

- Banking: Banking sector mostly uses this algorithm for the identification of loan risk.

- Medicine: With the help of this algorithm, disease trends and risks of the disease can be identified.
- Land Use: We can identify the areas of similar land use by this algorithm.
- Marketing: Marketing trends can be identified using this algorithm.

Here are some reasons why we choose Random Forest :

- Random Forest can perform both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

However, random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Example :

A bank wants to automate its loan approval process using a Random Forest classifier. The goal is to predict whether a customer's loan application will be approved or rejected based on their financial profile.

The dataset includes the following features:

- Age (years)
- Annual Income (in $1000s)
- Credit Score (300-850)
- Loan Amount Requested (in $1000s)
- Employment Status (Employed/Unemployed)
- Repayment History (Number of past late payments)
- Loan Approval (Target Variable: Approved or Rejected)

Applicant	Age (Years)	Annual Income ($1000s)	Credit Score (300-850)	Loan Amount ($1000s)	Loan Approved?
A1	25	30	650	10	Yes
A2	40	50	700	20	Yes
A3	35	45	620	50	No
A4	50	80	750	30	Yes
A5	28	35	580	40	No
A6	45	70	720	25	Yes

Step 1: Create Multiple Decision Trees

Each Decision Tree is trained on a random subset of the data (this is called bootstrap sampling).

For simplicity, let's assume we have 3 trees:

- Tree 1 (Built on 4 random applicants A1, A2, A3, A5):Splitting based on Credit Score > 600 → Approves A1, A2 and rejects A3, A5
- Tree 2 (Built on 4 different applicants A2, A3, A4, A6) : Splitting based on Annual Income > 60 → Approves A2, A4, A6, rejects A3
- Tree 3 (Built on 4 different applicants A1, A3, A4, A5): Splitting based on Loan Amount > 35 → Approves A1, A4, rejects A3, A5

Step 2: Predict a New Applicant (A7)

New Applicant:

Age	Annual Income ($1000s)	Credit Score	Loan Amount ($1000s)
38	55	690	25

Each tree predicts whether A7's loan is approved or rejected:

- Tree 1: Approved (because Credit Score > 600)
- Tree 2: Approved (because Income > 60)
- Tree 3: Rejected (because Loan Amount > 35)

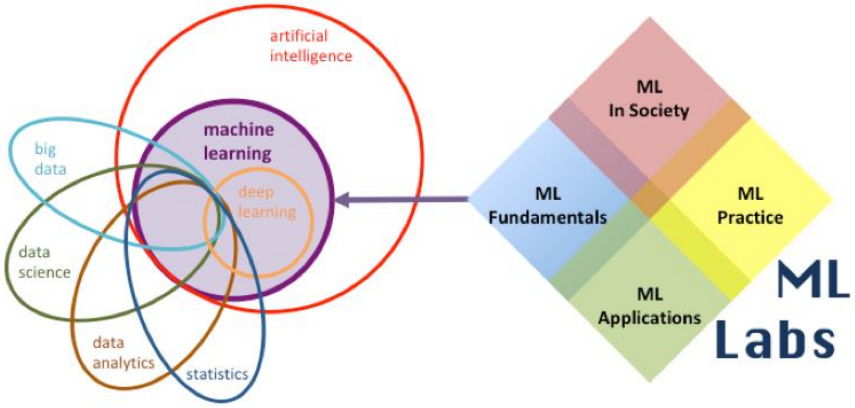
Step 3: Majority Voting for Final Prediction

The final prediction is based on majority voting among the trees:

Tree 1	Tree 2	Tree 3	Final Decision
Approved	Approved	Rejected	Approved

Since 2 out of 3 trees predicted "Approved", the final decision for A7 is "**Approved**".

Part Two : Labs



Introduction

The second part of the document delves into hands-on Machine Learning labs with Python, guiding readers through practical implementation of core ML concepts. Using libraries like NumPy, Pandas, Scikit-Learn, and TensorFlow, readers engage in data preprocessing, model training, and performance evaluation. Labs cover essential topics such as data visualization, feature engineering, model selection, and hyperparameter tuning, applying both supervised and unsupervised learning to real-world datasets.

Each lab emphasizes code optimization, experimentation, and insightful result interpretation, equipping readers with practical skills for effective ML development.

This section aims to build a comprehensive understanding of ML workflows and industry's best practices.

Lab 1 : Python review

Objective

The aim of this lab is to review some useful libraries (numpy, pandas matplotlib, etc.) to implement a machine learning model. The students must do all the activities to prepare the rest of the labs.

Instructions

Use Anaconda's jupyter notebook to implement the codes. Each pair must present its work in the form of a notebook file (Lab1.ipynb) and send the corrected version to the following address :**houda.elbouhissi@univ-bejaia.dz** with all necessary explanations.

The numpy library

The numpy library is a module for manipulating matrixes or multi-dimensional arrays. NumPy also includes functions for generating arrays**.** numpy is installed by default on Anaconda, but if you're using another editor, you can install it with:

pip install numpy

numpy functions begin with importing this library: *import numpy as np*

Using shortcuts (here np, which you can modify, rather than numpy) makes it easier to write library function calls.

Activity 1

$$\text{Si } A=\begin{pmatrix}-1 & -1 & -3\\ 1 & 2 & 0\\ 0 & 0 & 1\end{pmatrix} \text{ et } B=\begin{pmatrix}-2 & -1 & -6\\ 1 & 1 & 3\\ 0 & 0 & 1\end{pmatrix}. \text{ Calculer } A\times B$$

Is A invertible? If so, calculate A^{-1} (round the elements of A to 10^{-3}).

Activity 2

What does the following code give?

```
import numpy
v = numpy.array([1, 2, 3, 4]) * numpy.pi / 4 # numpy.pi = π
w = numpy.sin(v)
print(w)
```

What do the following NumPy functions do: zeros([num]), ones([num]), linspace(start, end, num), random.random([num])?

Activity 3

For this activity, the answer to each question must contain just one line of code.

- Create an array T1 containing only even numbers between 0 and 50.

- Create an array T2 that contains the cosines of the squares of the elements in T1.

- What is the minimum of array T2?

- What is the number of times that the maximum of the array T2 is present in T2.

The Matplotlib library

Matplotlib is a Python library that can be used to do all sorts of plots. Matplotlib is installed by default on Anaconda, but if you use another editor, you can install it with:

pip install Matplotlib

The first step in using Matplotlib functions is to import the library:

import Matplotlib.pyplot as plt

Using shortcuts (here plt, which you can modify) makes it easier to write calls to functions in the library.

x = [0, 1, 2] # abscissa list

y = [1, -1, 0] # ordinate list

plt.plot(x, y) # plots y as a function of x plt.show()

displays the plot window

Activity 1

Using the linspace and cos functions in NumPy, plot the function y = cos(x) on [0, 10π]. What influence does the number of points passed to the linspace function have? Using a second call to the plt.plot function before calling plt.show, superimpose the graph of the function y = exp(-x/10) cos(x). Still before calling plt.show, add a title with plt.title('The title of your choice') and names to the axes with plt.xlabel('x') and plt.ylabel('y=f(x)').

The Pandas library

The Pandas library is an important library for manipulating data in Python. Pandas implements the DataFrame class, which is a table structure (row, column). Each column has a name and contains a single type of data.

Pandas is installed by default on Anaconda, but if you use another editor you can install it with: pip install pandas

Using the Pandas functions starts with importing this library:

import pandas as pd

Using shortcuts (here pd , which you can modify, rather than pandas) makes it easier to write calls to functions in the library.

Activity 1

This first workshop involves manipulating covid data using the Pandas library. The data to be retrieved 'covid-hospit-incid-reg-2023-03-31-18h01.csv' is available at https://www.data.gouv.fr/fr/datasets/donnees-hospitalieres-relatives-a-lepidemie-de-covid-19/

1. Using the tail (or head) function, view the structure of the table. Identify the different data and their type (use dtypes).
2. Dates are considered to be character strings. It is easier to perform operations by converting the column into dates (use to_datetime).
3. Columns relating to departments and gender are deleted and then aggregated by day.
4. Now plot the data (use the logy option for a logarithmic scale).
5. Redo the same graph for your gender.

Activity 2

1. From the same link in Workshop 1, retrieve the temperature.csv file.
2. Use the describe function on the dataframe. What does this function do?
3. Create a new dataframe containing only the months March, June, September and December and deleting the cities in the 'East' region.
4. Retrieve the data using numpy and calculate the average temperature for each month. Also determine the correlation matrix between the 4 months of the year.

The Seaborn library

Seaborn allows you to produce a similar graph using Seaborn, a slightly more elaborate library for Python. Seaborn is based on Matplotlib and simplifies certain types of graphs related to

statistics. We can use the distplot function to plot a histogram with a kernel density estimate. A kernel density estimate is a curve - which is actually a smoothed version of the histogram that is easier to analyze.

Seaborn is installed by default on Anaconda, and if you use another editor, you can install it with: pip install seaborn

Using Seaborn's functions starts with importing this library:

import seaborn as sn

Using shortcuts (here sn , which you can modify, rather than seaborn) makes it easier to write calls to functions in the library.

For the next two workshops, we will use the dataset: https://www.data.gouv.fr/fr/datasets/impot-de-solidarite-sur-la-fortune-impot-sur-la-fortune-immobiliere-par-collectivite-territoriale/ , choose ISF 2017, then use the arrow to access the contents.

Activity 1

From the ISF table, draw a vertical bar chart representing the number of towns with more than 20,000 inhabitants that have more than 50 ISF taxpayers.

Activity 2

Using the ISF table, plot the average tax according to average wealth (for towns with more than 20,000 inhabitants and more than 50 ISF taxpayers).

Lab 2 : Regression (Supervised learning)

1/ Introduction

This second lab provides a general overview on how to build and evaluate a supervised learning algorithm such as regression. We focus on linear regression (simple) and logistic regression.

2/ Learning Objectives

- Train and test data
- Making predictions
- Evaluating predictions

3/ Some vocabulary

Machine Learning is a vast and complex field. There are 4 important concepts that you use during all your machine learning projects, so you must be aware of that.

- *Dataset*: In Machine Learning, everything starts with a Dataset containing our data. In supervised learning, the Dataset contains the questions (y) and answers (x) to the problem that the machine must solve.

- *The model and its parameters*: starting from the Dataset, we create a model (a mathematical function). The coefficients of this function are the model parameters.

- *Cost Function*: When we test our model on the Dataset, it returns some errors. The sum of these errors is called the *Cost* Function.

- *Learning algorithm*: The central idea of Machine Learning is to let the machine find the parameters of the model that *minimize* the Cost Function.

- *Overfitting:* Overfitting occurs when the model fits the training data too closely, capturing noise or random fluctuations in the data. This leads to a model that performs very well on the training data *but* doesn't generalize well to new data.

  Example: Suppose we're building a linear regression model to predict house prices based on their size. If we have a dataset with outliers or unrepresentative data points, a complex machine learning model might try to fit to these points even if they don't truly represent the overall trend. This would result in

overfitting, where the model is too complex relative to the size of the training data.

- *Underfitting:* Underfitting occurs when the model is too simple to capture the underlying structure of the data. This results in a model that fails to fit well even on the training data, leading to poor performance on both training and test data.
  Example: Let's revisit our house price prediction example. If we use a very simple linear regression model with just one feature (e.g., only the house size), it might underestimate the relationship between size and price, failing to account for other important factors like the number of bedrooms, location, etc. In this case, the model would be too simplistic to capture the complexity of the actual data.

  How to address these issues:

  - To avoid overfitting, techniques such as regularization (e.g., L1 or L2) can be used to penalize overly complex models, or cross-validation can be employed to select optimal hyperparameters.
  - To avoid underfitting, one can try more complex models or add relevant features to the data.

In summary, overfitting occurs when the model is too complex relative to the training data, while underfitting occurs when the model is too simple to capture the data's structure. Both issues need to be monitored and addressed to achieve high-performing and generalizable machine learning models (we'll see these concepts with more detail so far).

4/ Programming steps

To implement these 4 steps in Python, we need to take the following steps:

- Import all the necessary libraries.
- *Preparing the DataSet*. The machine receives data characterized by X variables (called features) and annotated with a y variable (called a label or target).
- *Select the model* (or estimator) the machine needs to learn, specifying the model's hyperparameters (for example, LinearRegression, …etc.).
- Train the model on data X and Y : **model.fit(X,Y)**
- Evaluate the model : **model.score(X,Y)**

- Use the model : **model.predict(x)**

5/ Software tools

You install anaconda (https://www.anaconda.com/), which is the best environment for machine learning codes and the sklearn library. If you use just Jupyter or spyder, you must install the appropriate libraries.

Duration: 2 hours (+ 2 hours homework)

---------- **Preliminaries** ----------

Regression is a supervised learning problem where there is an input x and an output y and the task are to learn the mapping from the input to the output. We have also seen that the approach in machine learning is that we assume a model, that is, a relation between x and y containing a set of parameters, say, _ in the following form:

$$y = g(x; \theta)$$

g(x; θ) is the regression function. The machine learning program optimizes the parameters θ such that the approximation error (called cost function) is minimized, that is, our estimates are as close as possible to the correct values given in the training set.

Several methods are proposed to optimize cost function such as variance, covariance, …etc. the best optimization method is called gradient descent.

Logistic regression is used when the dependent variable is binary (0/1, True/False, Yes/No) in nature. Even though the output is a binary variable, what is being sought is a probability function which may take any value from 0 to 1.

---------- **Examples** ----------

- Let see this simple example about linear regression with one variable as we called usually simple regression, we need to predict the house price using regression. The used dataset involves one features, one Target (price) and downloaded from Kaggle : https://www.kaggle.com/datasets/harlfoxem/housesalesprediction.

First, we import useful libraries.

```
import numpy as np
import pandas as pd
import seaborn as sb
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import classification_report
```

```
%matplotlib inline
```
Read the dataset, we use here the kc_house_data.csv downloaded from : https://www.kaggle.com/datasets/harlfoxem/housesalesprediction

```
df = pd.read_csv("kc_house_data.csv")
df.head(15)
```
Now, we explore our data set to find missed values. We Determine the features that affect (theoretically) house prices.
```
df.info()
df.isnull().sum()
df = df.drop(['id','date', 'lat', 'long','zipcode'], axis =1)
df.head()
plt.figure(figsize=(48, 6))
sb.stripplot(x="yr_built", y="bedrooms", data=df);
plt.figure(figsize=(20, 8))
sb.set_context("notebook", font_scale=1.5, rc={"lines.linewidth":
2.5})
sb.stripplot(x="bedrooms", y="price", data=df);
plt.figure(figsize=(48, 8))
sb.barplot(x="bedrooms", y="price", hue="grade", data=df);
sb.countplot(x='bedrooms',data=df, palette='hls')
```

Next, we try to build our model using linear regression, we split our data set to 70% for training and 30% for testing.

```
columns = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
'floors', 'waterfront', 'view', 'condition', 'grade','sqft_above',
'sqft_basement', 'yr_built','yr_renovated', 'sqft_living15',
'sqft_lot15']
labels = df['price'].values
features = df[list(columns)].values
X_train, X_test, y_train, y_test = train_test_split(features,
labels, test_size=0.30)
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```
We evaluate our model.
```
Accuracy = regr.score(X_train, y_train)
print ("Accuracy in the training data: ", Accuracy*100, "%")
accuracy = regr.score(X_test, y_test)
print ("Accuracy in the test data", accuracy*100, "%")
```

Now, let us see another example about logistic regression. We use a dataset provided by the teacher to predict id a client purchased an object or no, that means we study the influence of some features on the result.

Here is an excerpt about the dataset :

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000.0	0
1	15810944	Male	35.0	20000.0	0
2	15668575	Female	26.0	43000.0	0
3	15603246	Female	27.0	57000.0	0
4	15804002	Male	19.0	76000.0	0

```
# import libraries

%matplotlib
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

# Import the dataset
dataset = pd.read_csv('clients.csv')

# Data visualisation
dataset.head()

# Dataset visualisation
plt.scatter(dataset.EstimatedSalary, dataset.Purchased)

# Delete User ID
dataset.drop(['User ID'],axis='columns',inplace=True)

# Data visualisation
dataset.head()

# Transformation of gender variable
dataset.Gender = dataset.Gender.map({'Male': 1, 'Female': 2})
dataset.head()
ax = plt.axes(projection='3d')
ax.scatter(dataset.Gender,dataset.Age,dataset.EstimatedSalary,
c=dataset.Purchased)

# Achat percent
count_sub = len(dataset[dataset['Purchased']==1])
count_no_sub = len(dataset[dataset['Purchased']==0])
pct_of_no_sub = count_no_sub/(count_no_sub+count_sub)
print("Pourcentage absence d'achat", pct_of_no_sub*100)
```

```
# genre Influence on Purchases
table= pd.crosstab(dataset.Gender,dataset.Purchased)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar',
stacked=True)
plt.title('Gender / Purchases')
plt.xlabel('Gender')
plt.ylabel('Pourcentage de client')
dataset.drop(['Gender'],axis='columns',inplace=True)
dataset.head()
table= pd.crosstab(dataset.Age,dataset.Purchased)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar',
stacked=True)
plt.title('Age / Purchases')
plt.xlabel('Age')
plt.ylabel('Client percentage')
plt.savefig('Age-Purchases')
table= pd.crosstab(dataset.EstimatedSalary,dataset.Purchased)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar',
stacked=True)
plt.title('Salaire / Achat')
plt.xlabel('Salaire')
plt.ylabel('Pourcentage de client')

# define Y dependent variable and X independent variable
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, -1].values

# points Visualisation
plt.scatter(X[:,0],X[:,1], c=y)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.25, random_state = 0)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_test

# Build the model
classifier = LogisticRegression(random_state = 0,
solver='liblinear')
classifier.fit(X_train, y_train)
# Faire de nouvelles prédictions
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test)

# confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

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

# Results Visualisation
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.4, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Résultats du Training set')
plt.xlabel('Age')
plt.ylabel('Salaire Estimé')
plt.legend()
plt.show()
x_predict = sc.transform([[35,15000]])
classifier.predict(x_predict)
```

---------- **Activities** ----------

Activity 1

The aim of this activity is to predict the relationship between the price of a pizza and its size using linear regression. As previously specified, and as we have chosen a linear regression, this means that we are making the hypothesis that there is a linear relationship between the price of a pizza and its size.

- Dataset : We will first generate a set of points for the training set:

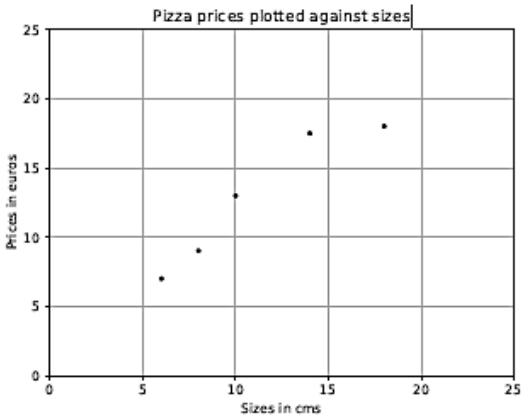
Size (Feature)	Price (Target)
6	7
8	9
10	13
14	17.5
18	18

We then need to declare a numpy array containing the data (pizza sizes) and declare an array containing the corresponding prices , for example :

X = np.array([[6], [8], [10], [14], [18]])

y = [7, 9, 13, 17.5, 18]

- Now display the data in a graph to see if there is a relationship between the size and price of a pizza. To do this, we'll use matplotlib. We can see that there is indeed a relationship: the price of a pizza increases with its size, which is consistent with our expertise on the subject.



We are going to use the sklearn library and in particular the linear regression model:

```
from sklearn.linear_model import LinearRegression
```

- Now use the functions fit for training and predict to predict the model's response to an example.
- We now need to evaluate our model. To do this we need to define a loss function (also called a cost function). The difference between the actual price of the pizzas and the price predicted by our model is called the residual error or learning error. When we evaluate our model on an independent test basis, the resulting error is called the prediction error or test error.

We can create the best possible model by minimizing the sum of the residual error. This error is called the residual sum of squares (RSS).

The cost function associated with this error is defined as follows : $RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2$,

With Yi the observed value (real value) and f(xi) the predicted value.

Calculate the error obtained in python (Residual sum of squares : 8.75)

Activity 2

The second activity involves applying simple linear regression on the dataset below using the least-squares method. The student would build his own model using Python without using the Python machine learning instructions.

Size (x)	Price (y)
2104	4.5
1416	3.5
1534	3.2
852	1.6

Our objective is to find the equation y= ax+b that fits with the cloud points.

- Calculate a and b?
- Calculate the correlation indicator?
- Indicate whether the equation found is accurate ?

Do the same work using gradient descent. Compare your prediction results with the python machine learning code.

Activity 3

In the first part of this exercise, we'll build a logistic regression model to predict whether a student gets admitted to a university.

Suppose that you are the administrator of a university department, and you want to determine each applicant's chance of admission based on their results on two exams. You have historical data from previous applicants that you can use as a training set for logistic regression. For each training example, you have the applicant's scores on two exams and the admissions decision.

To accomplish this, we're going to build a classification model that estimates the probability of admission based on the exam scores.

Let's start by examining the data :

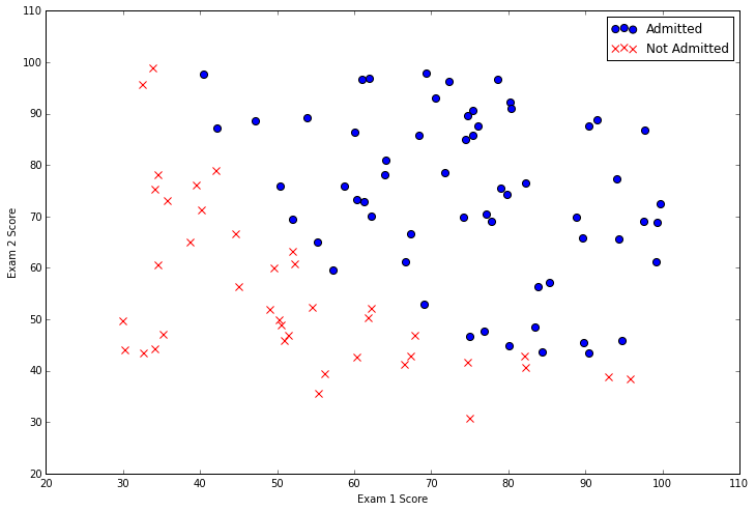
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import os
path = os.getcwd() + '\data\ex2data1.txt'
data = pd.read_csv(path, header=None, names=['Exam 1', 'Exam 2',
'Admitted'])
data.head()
```

	Exam 1	Exam 2	Admitted
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

Let's create a scatter plot of the two scores and use color coding to visualize if the example is positive (admitted) or negative (not admitted).

```
positive = data[data['Admitted'].isin([1])]
negative = data[data['Admitted'].isin([0])]

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b',
marker='o', label='Admitted')
ax.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r',
marker='x', label='Not Admitted')
ax.legend()
ax.set_xlabel('Exam 1 Score')
ax.set_ylabel('Exam 2 Score')
```



It looks like there is a clear decision boundary between the two classes. Now we need to implement logistic regression so we can train a model to predict the outcome.

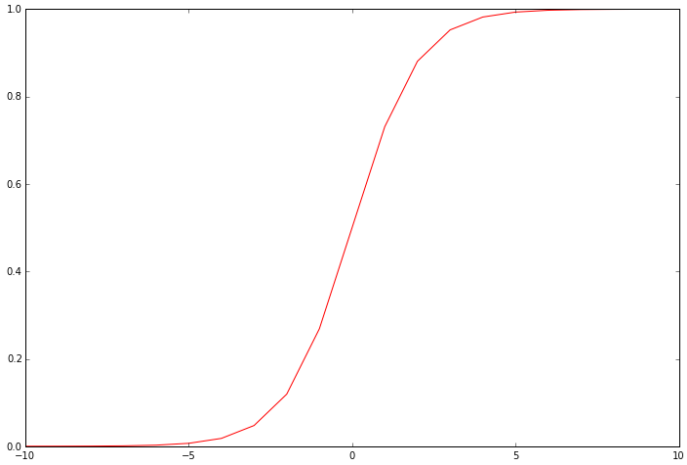
First, we need to create a sigmoid function. The code for this is simple.

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

Let's do a quick sanity check to make sure the function is working.

```
nums = np.arange(-10, 10, step=1)
```

```
fig, ax = plt.subplots(figsize=(12,8))
ax.plot(nums, sigmoid(nums), 'r')
```



Now we need to write down the cost function to evaluate a solution.

```
def cost(theta, X, y):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)
    first = np.multiply(-y, np.log(sigmoid(X * theta.T)))
    second = np.multiply((1 - y), np.log(1 - sigmoid(X * theta.T)))
    return np.sum(first - second) / (len(X))
```

Now we need to do some setup, like what we did in exercise 1 for linear regression.

```
# add a ones column - this makes the matrix multiplication work out
easier
data.insert(0, 'Ones', 1)

# set X (training data) and y (target variable)
cols = data.shape[1]
X = data.iloc[:,0:cols-1]
y = data.iloc[:,cols-1:cols]

# convert to numpy arrays and initalize the parameter array theta
X = np.array(X.values)
y = np.array(y.values)
theta = np.zeros(3)
```

Let's quickly check the shape of our arrays to make sure everything looks good.

```
X.shape, theta.shape, y.shape
```

Now let's compute the cost for our initial solution (0 value for theta).

```
cost(theta, X, y)
```

Looks good. Next, we need a function to compute the gradient (parameter updates) given our training data, labels, and some parameters theta.

```
def gradient(theta, X, y):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)
    parameters = int(theta.ravel().shape[1])
    grad = np.zeros(parameters)

    error = sigmoid(X * theta.T) - y

    for i in range(parameters):
        term = np.multiply(error, X[:,i])
        grad[i] = np.sum(term) / len(X)
    return grad
```

Note that we don't perform gradient descent in this function - we just compute a single gradient step. In the exercise, an Octave function called "fminunc" is used to optimize the parameters given functions to compute the cost and the gradients. Since we're using Python, we can use SciPy's "optimize" namespace to do the same thing.

Let's look at a single call to the gradient method using our data and initial parameter values of 0.

```
gradient(theta, X, y)
```

Now we can use SciPy's truncated newton (TNC) implementation to find the optimal parameters.

```
import scipy.optimize as opt
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient,
args=(X, y))
result
```

Let's see what our cost looks like with this solution.

```
cost(result[0], X, y)
```

```
cost(result[0], X, y)
```

Next, we need to write a function that will output predictions for a dataset X using our learned parameters theta. We can then use this function to score the training accuracy of our classifier.

```
def predict(theta, X):

    probability = sigmoid(X * theta.T)

    return [1 if x >= 0.5 else 0 for x in probability]
```

```
theta_min = np.matrix(result[0])

predictions = predict(theta_min, X)

correct = [1 if ((a == 1 and b == 1) or (a == 0 and b == 0)) else 0
for (a, b) in zip(predictions, y)]

accuracy = (sum(map(int, correct)) % len(correct))

print 'accuracy = {0}%'.format(accuracy)
```

Our logistic regression classifier correctly predicted if a student was admitted or not 89% of the time. Not bad! Keep in mind that this is training set accuracy though. We didn't keep a hold-out set or use cross-validation to get a true approximation of the accuracy so this number is likely higher than its true performance (this topic is covered in a later exercise).

Lab 3 : Random Forest (Supervised learning)

Activity 1

Loading the dataset into a pandas dataframe

```
import numpy as np

import pandas as pd

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

df.head() #Display the first 5 lines
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

We separate the properties/labels and extract only the values from the frame

```
columns = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction',
'Age']

labels = df['Outcome'].values

features = df[list(columns)].values
```

At this step , we have two arrays

Labels = labels (values only)

Features = properties only

The next step is to split the data into training and test samples

train_test_split = the function asked to split the data and takes the three-parameter matrices features and labels and then the size of the test sample

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features,
labels, test_size=0.30)
```

The results from the previous operation are four matrices

x_train = training matrix for properties

y_train = training matrix for attributes

X_test = the test matrix for properties

y_test = the test matrix for labels

Almost everything is ready for data and data engineering.

Now there are two important steps

1- Initialize the model (in this case we chose Random Forest)

2- Train the model by passing the two training matrices (properties and labels)

```
clf = RandomForestClassifier(n_estimators=1)
clf = clf.fit(X_train, y_train)
```

We evaluate the performance of the model by calculating the accuracy as follows :

```
accuracy = clf.score(X_train, y_train)
print(accuracy*100)
accuracy = clf.score(X_test, y_test)
print(accuracy*100)
```

confusion matrix and classification report for the testing sample

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
ypredict = clf.predict(X_train)
print('\nTraining classification report\n',
classification_report(y_train, ypredict))
print("\n Confusion matrix of training \n",
confusion_matrix(y_train, ypredict))
```

```
ypredict = clf.predict(X_test)
print '\nTraining classification report\n',
classification_report(y_test, ypredict)
print "\n Confusion matrix of training \n", confusion_matrix(y_test,
ypredict)
```

Activity 2

In the code below we use a Random Forest Classifier to analyze the Titanic dataset. The Random Forest Classifier learns from the training data and is tested on the test set and we evaluate the model's performance using a classification report to see how well it predicts the outcomes and used a random sample to check model prediction.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings('ignore')
# Corrected URL for the dataset
```

```
url                                                                    =
"https://raw.githubusercontent.com/datasciencedojo/datasets/master/t
itanic.csv"
titanic_data = pd.read_csv(url)
# Drop rows with missing 'Survived' values
titanic_data = titanic_data.dropna(subset=['Survived'])
# Features and target variable
X = titanic_data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
y = titanic_data['Survived']
# Encode 'Sex' column
X.loc[:, 'Sex'] = X['Sex'].map({'female': 0, 'male': 1})
# Fill missing 'Age' values with the median
X.loc[:, 'Age'].fillna(X['Age'].median(), inplace=True)
# Split data into training and testing sets
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,
test_size=0.2, random_state=42)
# Initialize RandomForestClassifier
rf_classifier       =       RandomForestClassifier(n_estimators=100,
random_state=42)
# Fit the classifier to the training data
rf_classifier.fit(X_train, y_train)
# Make predictions
y_pred = rf_classifier.predict(X_test)
# Calculate accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
# Print the results
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_rep)
# Sample prediction
sample = X_test.iloc[0:1]  # Keep as DataFrame to match model input
format
prediction = rf_classifier.predict(sample)
# Retrieve and display the sample
sample_dict = sample.iloc[0].to_dict()
print(f"\nSample Passenger: {sample_dict}")
print(f"Predicted Survival: {'Survived' if prediction[0] == 1 else
'Did Not Survive'}")
```

The output of the program :

```
Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support
           0       0.82      0.85      0.83       105
           1       0.77      0.73      0.75        74
  Sample Passenger: {'Pclass': 3, 'Sex': 1, 'Age': 28.0, 'SibSp': 1,
'Parch': 1, 'Fare': 15.2458}

Predicted Survival: Did Not Survive
```

Activity 3

Advantages of Random Forest3

- Random Forest provides very accurate predictions even with large datasets.
- Random Forest can handle missing data well without compromising with accuracy.
- It doesn't require normalization or standardization on datasets.
- When we combine multiple decision trees it reduces the risk of overfitting the model.

Limitations of Random Forest

- It can be computationally expensive especially with a large number of trees.
- It's harder to interpret the model compared to simpler models like decision trees.

Lab 4 : K-means(Unsupervised learning)

Activity 1

Here's a simple example of how to use the K-means algorithm to cluster students based on their study hours and exam scores.

Suppose you have data on students' study hours and their scores on an exam according to their hours of study. Let us group in cluster them to identify patterns in study habits and performance. The dataset contains a couple of study hours and corresponding exam scores. The aim is to identify the different clusters.

This example helps visualize how students with similar study habits perform similarly on exams.

Study hours	Score
2	50
3	60
5	80
8	90
1	40
4	70
6	85
7	88
5	75
2	55

Here is the python code:

First, we import useful libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

Load the data

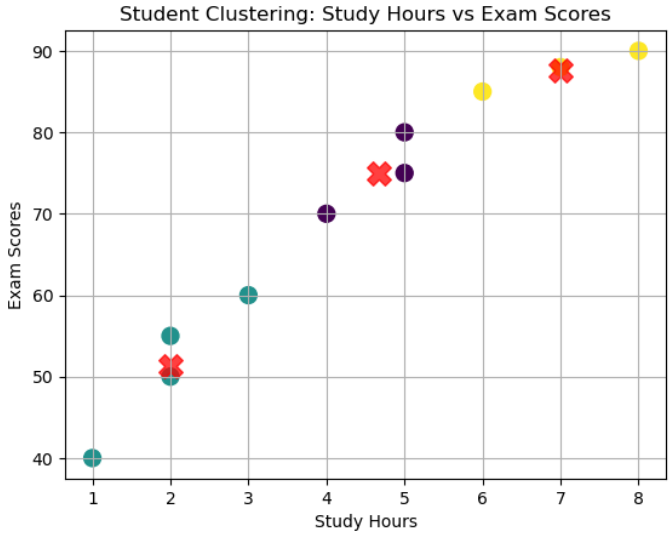
```
data = np.array([
    [2, 50],
    [3, 60],
    [5, 80],
    [8, 90],
    [1, 40],
    [4, 70],
    [6, 85],
    [7, 88],
    [5, 75],
    [2, 55],
])
```

Now, we apply the algorithm to find 3 clusters

```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(data)
y_kmeans = kmeans.predict(data)
```

we plot the results

```
plt.scatter(data[:, 0], data[:, 1], c=y_kmeans, s=100,
cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200,
alpha=0.75, marker='X')
plt.title('Student Clustering: Study Hours vs Exam Scores')
plt.xlabel('Study Hours')
plt.ylabel('Exam Scores')
plt.grid()
plt.show()
```



Activity 2

Lab 5 : KNN(Supervised learning)

1/ Introduction

This lab provides a general overview on how to build and evaluate a supervised learning algorithm such as classification. We focus on the KNN algorithm.

2/ Learning Objectives

- Train and test data
- Making predictions
- Evaluating predictions

3/ Programming steps

To implement these 3 steps in Python, we need to take the following steps:

- Import all the necessary libraries.
- Preparing the DataSet. The machine receives data characterized by X variables (called features) and annotated with a y variable (called a label or target).
- Select the model (or estimator) the machine needs to learn, specifying the model's hyperparameters ***(KNeighborsClassifier)***.
- Train the model on data X and Y : **model.fit(X,Y)**
- Evaluate the model : **model.score(X,Y)**
- Use the model : **model.predict(x)**

4/ Software tools

You install anaconda (https://www.anaconda.com/), which is the best environment for machine learning coding and the sklearn library. If you use just Jupyter or spyder, you have to install the appropriate libraries.

Duration: 2 hours (+ 2 hours homework)

---------- **Activities** ----------

Activity 1

Suppose we have the height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer based only on the height and weight information we have. The height, weight and T-shirt size data are shown below.

Height (in cm)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M

158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

1. Create the KNN model in Python with two different solutions : the first using the sklearn library, the second without using sklearn.
2. Predict the size of the T Shirt for values not belonging to the dataset.

Activity 2

Consider a regression problem that generates a real numerical value (1; 3.5; 7.2) as an output. For example, the table below estimates the age of a crab (decimal value label) as a function of its width and mass (predictors).

Width	Mass	Age
50	100	1
150	500	3.5
200	1200	7
225	1800	10
130	250	3

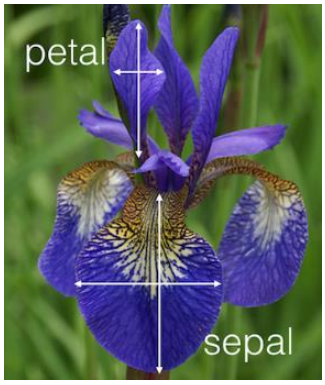
Create a jupyter notebook to predict new values (width and mass) using the KNN algorithm. (The first step is to run the algorithm manually).

Activity 3

This activity concerns the IRIS classification using KNN.

In the activity shown above the following steps are performed:

1. The k-nearest neighbor algorithm is imported from the scikit-learn package.
2. Create features and target variables.
3. Split data into training and test data.
4. Generate a k-NN model using neighbors value.
5. Train or fit the data into the model.
6. Predict the future.



First, we import the libraries to:

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
```

Now, we display dataset features :

```
iris.feature_names
```

```
Out[6]: ['sepal length (cm)',
         'sepal width (cm)',
         'petal length (cm)',
         'petal width (cm)']
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

The dataset in a dataframe :

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
df['target'] = iris.target
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df[df.target==1].head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

```
df[df.target==2].head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

```
df[45:55]
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
45	4.8	3.0	1.4	0.3	0	setosa
46	5.1	3.8	1.6	0.2	0	setosa
47	4.6	3.2	1.4	0.2	0	setosa
48	5.3	3.7	1.5	0.2	0	setosa
49	5.0	3.3	1.4	0.2	0	setosa
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

```
Train test split

from sklearn.model_selection import train_test_split
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)
len(X_train)
120

len(X_test)

30

Create KNN (K Neighrest Neighbour Classifier)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
knn.predict([[4.8,3.0,1.5,0.3]])

array([0])


Plot Confusion Matrix

from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]], dtype=int64)


Print classification report for precision, recall and f1-score for
each class

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Activity 4

Imagine we want to classify a new fruit based on its characteristics. We have a dataset of fruits with features such as weight, color, and sweetness, and we want to classify a new fruit as either "Apple," "Orange," or "Banana."

You have a table of fruits with the following characteristics:

Fruit	Weight (g)	Color	Sweetness (1-10)
Apple	150	Red	7
Apple	160	Red	8
Orange	130	Orange	6
Orange	140	Orange	5
Banana	120	Yellow	9
Banana	125	Yellow	8

We want to classify a new fruit (that does not belong to the dataset) with the KNN algorithm:

Fruit	Weight (g)	Color	Sweetness (1-10)
?	145	Orange	7

This example illustrates how the KNN algorithm classifies a new data point by examining its nearest neighbors in the feature space. By calculating distances and using majority voting, KNN can effectively categorize data based on similarity.

Let's start……

We use a distance metric (e.g., Euclidean distance) to measure how similar the new fruit is to each fruit in the dataset.

For each fruit, we calculate the distance to the new fruit based on the features:

- Distance to Apple (150, Red, 7):
  $d = \sqrt{(145-150)^2 + (0-1)^2 + (7-7)^2} = \sqrt{25 + 1 + 0} = \sqrt{26} \approx 5.1$
- Distance to Orange (130g, Orange, 6):
  $d = \sqrt{(145-130)^2 + (0-1)^2 + (7-6)^2} = \sqrt{225 + 1 + 1} = \sqrt{227} \approx 15.1$
- Distance to Orange (140g, Orange, 5):
  $d = \sqrt{(145-140)^2 + (0-1)^2 + (7-5)^2} = \sqrt{25 + 1 + 4} = \sqrt{30} \approx 5.5$
- Distance to Banana (120g, Yellow, 9):
  $d = \sqrt{(145-120)^2 + (0-1)^2 + (7-9)^2} = \sqrt{625 + 1 + 4} = \sqrt{630} \approx 25.1$
- Distance to Banana (125, Yellow, 8):
  $d = \sqrt{(145-125)^2 + (0-1)^2 + (7-8)^2} = \sqrt{400 + 1 + 1} = \sqrt{402} \approx 20.0$

Now, after calculating distances, we list the results in order from smallest to largest:

- Apple 1 (5.1)

- Orange 2 (5.5)
- Apple 2 (6.4)
- Orange 1 (15.1)
- Banana 1 (25.1)
- Banana 2 (20.0)

Choose the number of neighbors (k). For example, we choose k = 3.

The closest 3 neighbors are : Apple, Orange, Orange, so the majority class among these neighbors is Orange.

Finally, the new fruit is classified as Orange according to the obtained results (The nearest neighbors).

Here is the python code :

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
# Features: [Weight (grams), Color (1=Red, 2=Orange, 3=Yellow),
Sweetness (1-10)]
# Labels: 0=Apple, 1=Orange, 2=Banana
data = np.array([
    [150, 1, 7],  # Apple
    [160, 1, 8],  # Apple
    [130, 2, 6],  # Orange
    [140, 2, 5],  # Orange
    [120, 3, 9],  # Banana
    [125, 3, 8],  # Banana
])

labels = np.array([0, 0, 1, 1, 2, 2])

# Train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)  # k=3
knn.fit(data, labels)

# New fruit to classify: [Weight, Color, Sweetness]
new_fruit = np.array([[145, 2, 7]])

# Predict the class of the new fruit
prediction = classifier.predict(new_fruit)
class_names = ["Apple", "Orange", "Banana"]
print(f"The new fruit is classified as:
{class_names[prediction[0]]}")
```

```
The new fruit is classified as: Orange
```

Appendix A

Anaconda & Jupyter Notebook Installation Guide

Lecture Notes

A.1. Introduction

The Jupyter Notebook is an open-source web application for creating and sharing documents that contain code, equations, visualizations, and narrative text. Jupyter supports different programming languages (for over 40 with Python). This guide will focus on using Python, as it is the most common language used in data science and for our labs.

This tutorial will explain step-by-step how to install Jupyter Notebook locally and create a first file. We explain here to a beginner how to install jupyter and work with the main features.

Essentially, Jupyetr works with notebooks. A Notebook is a document that combines code and its output seamlessly. It allows to run code, display the results, and add explanations, formulas, and charts all in one place. This makes work more transparent, understandable, and reproducible and it is particularly beneficial for education and project presentations.

Jupyter Notebooks have become an essential part of the data science workflow in companies and organizations worldwide. They enable data scientists to explore data, test hypotheses, and share insights efficiently.

As an open-source project, Jupyter Notebooks are completely free. You can download the software directly from the Project Jupyter website or as part of the Anaconda data science toolkit.

If your goal is to work with data, using Jupyter Notebooks will streamline your workflow and make it easier to communicate and share your results.

A.2. Installation

The easiest way for a beginner to get started with Jupyter Notebooks is by installing Anaconda. Anaconda is the most widely used Python distribution for data science and comes pre-loaded with all the most popular libraries and tools.

Some of the biggest Python libraries included in Anaconda are : Numpy, pandas, and Matplotlib, and many other useful librairies.

Anaconda is available on https://www.anaconda.com/download/ , download the laster version. Installing Anaconda on our computer is very easy, follow the instructions on the download page and/or in the executable.

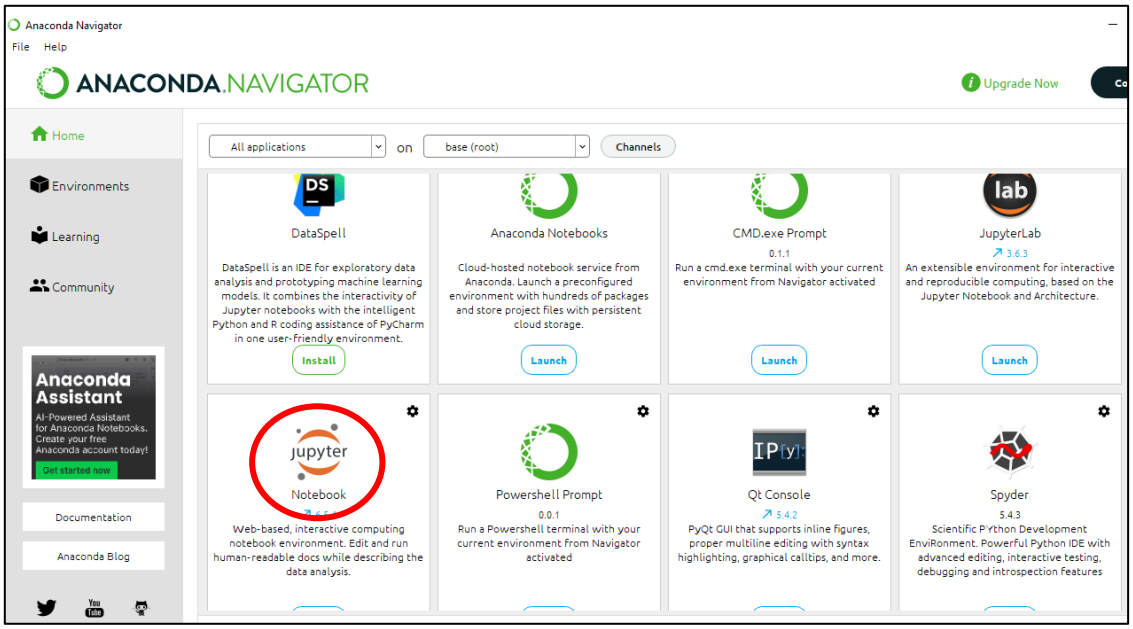


Figure : Anaconda Interface Screenshot

However, if you don't need Anaconda et you are an advanced python programmer, and you have already installed python on your system, and you prefer to manage the useful packages manually, you use from your terminal : *use pip3* to install any package.

For example , use this line to install jupyter : **`pip3 install jupyter`**

Note that Python is a requirement (Python 3.3 or greater, or Python 2.7) for installing the Jupyter Notebook itself (if Python is not installed before).

A.3. Creating a First Notebook

In this section, we're going to learn to run and save notebooks, familiarize ourselves with their structure, and understand the interface. We'll define some core terminology that will steer you towards a practical understanding of how to use Jupyter Notebooks by yourself and set us up for the next section, which walks through an example data analysis and brings everything we learn here to life.

On Windows, you can run Jupyter via the shortcut Anaconda adds to your start menu, which will open a new tab in your default web browser that should look something like the following screenshot:

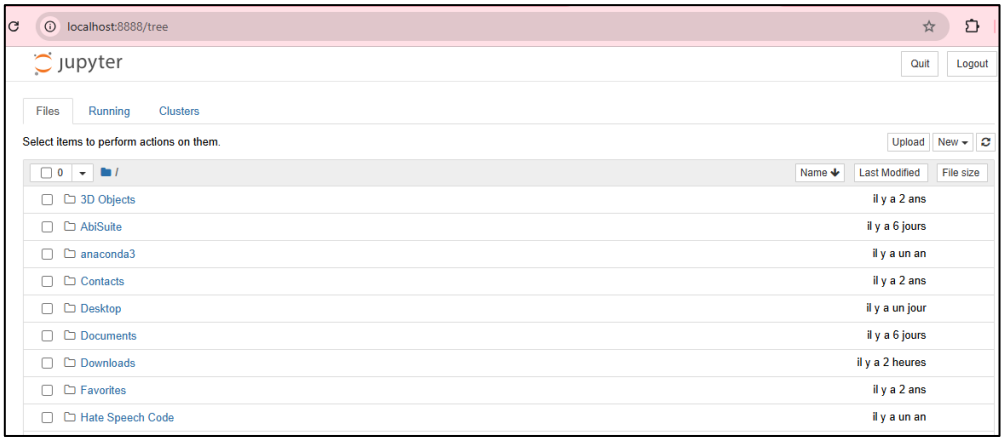


Figure : Jupyter Interface Screenshot

A.3.1. What is an ipynb File?

Each *.ipynb* file is a notebook, so each time you create a new notebook, a new .ipynb file will be created. Each .ipynb file is a text file that describes the contents of your notebook in a format called JSON. Each cell and its contents, including image heads, links.

To create a new notebook, click on the new button at the top right corner. Click it to open a drop-down list and then if you click on Python3, it will open a new notebook (see the figure below)

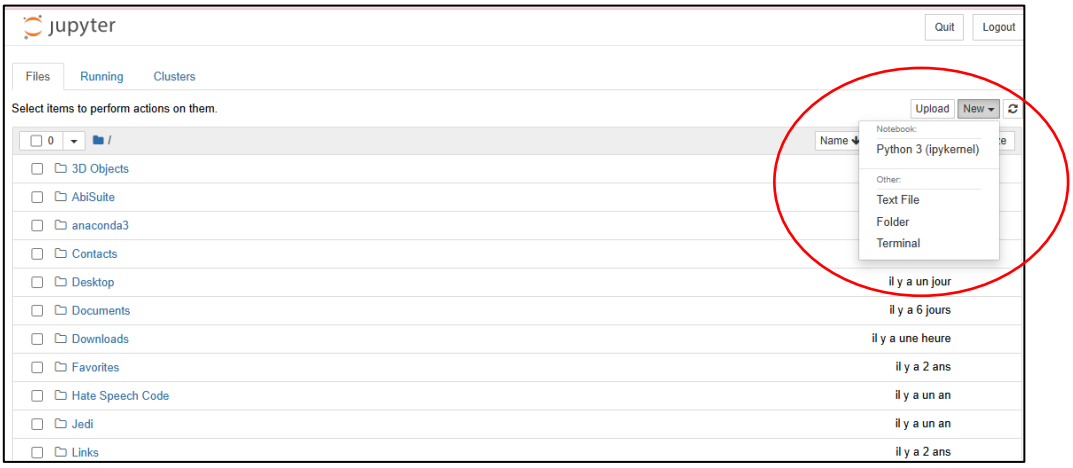


Figure : Creating a new Notebook

The web page should look like this:

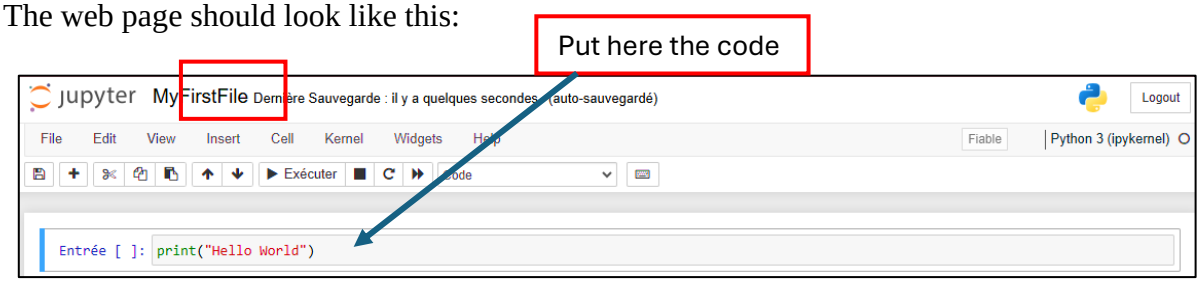


Figure : First Notebook (MyFirstFile.ipynb)

After successfully installing and creating a notebook in Jupyter Notebook, let's see how to write code in it. Jupyter notebook provides a cell for writing code in it. The type of code depends on the type of notebook you created.

For example, if you create a Python3 notebook then you can write Python3 code in the cell. Now, let's add the following code which displays the message Hello World as output ::

```
print("Hello World")
```

We save our file and name it MyFirstFile.ipynb (*File → Save as*).

To run a cell either click the run button or press shift ⇧ + enter ⏎ after selecting the cell you want to execute. After writing the above code in the jupyter notebook, the output was:**Note:** When a cell has executed the label on the left i.e. ln[] changes to ln[1]. If the cell is still under execution the label remains ln[*].

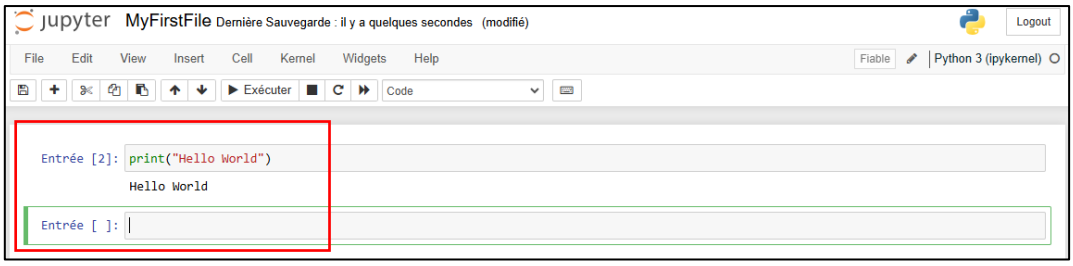


Figure : Running step

A.3.2. Cells in Jupyter Notebook

Cells can be considered as the body of the Jupyter. In the above screenshot, the box with the green outline is a cell.

There are 3 types of cells: code, Markup and Raw NBConverter.

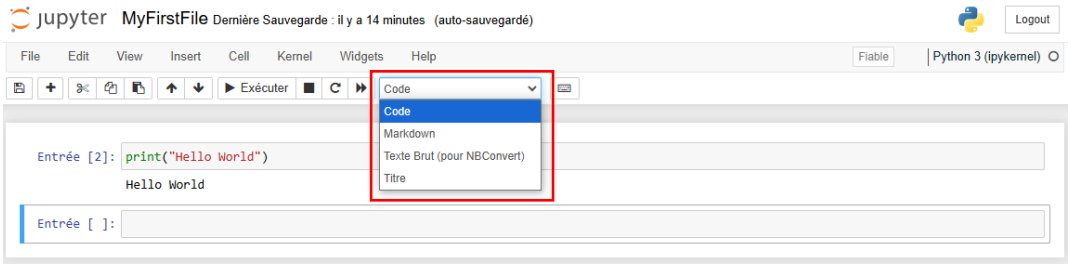


Figure : Cells in Jupyter Notebook

- **Code**

This is where the code is typed and when executed the code will display the output below the cell. The type of code depends on the type of notebook you have created.

For example, if the notebook of Python3 is created then the code of Python3 can be added.

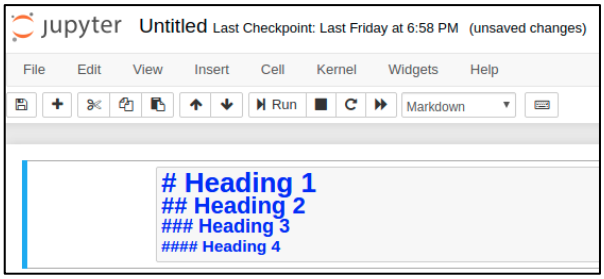
- **Markup**

Markdown is a popular markup language that is the superset of HTML. Jupyter Notebook also supports markdown. The cell type can be changed to markdown using the cell menu.

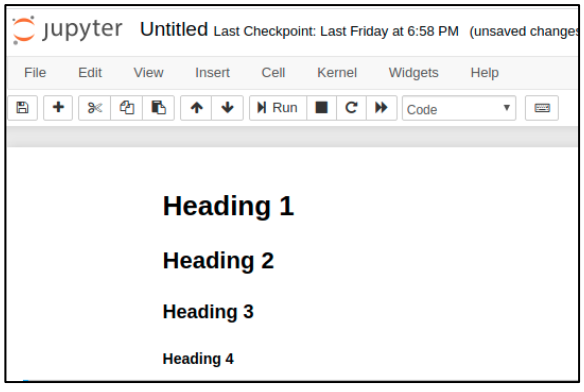
Heading can be added by prefixing any line by single or multiple '#' followed by space.

Example :

Input



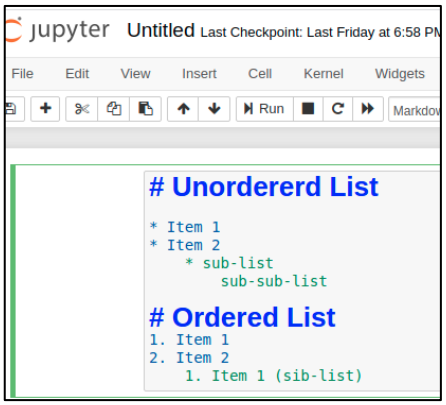
Output



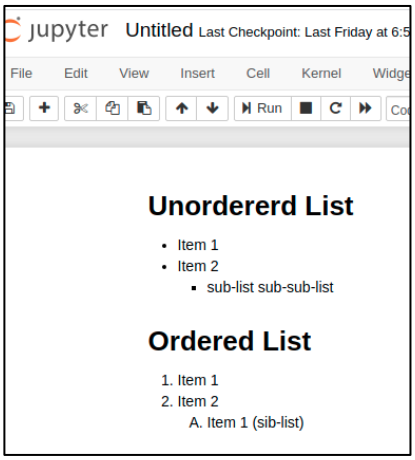
In addition, adding List is simple in Jupyter Notebook. The list can be added by using '*' sign. And the Nested list can be created by using indentation.

Example :

Input



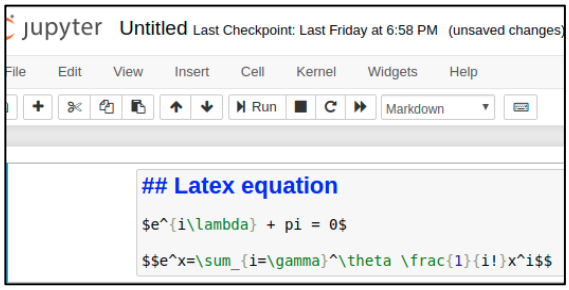
Output



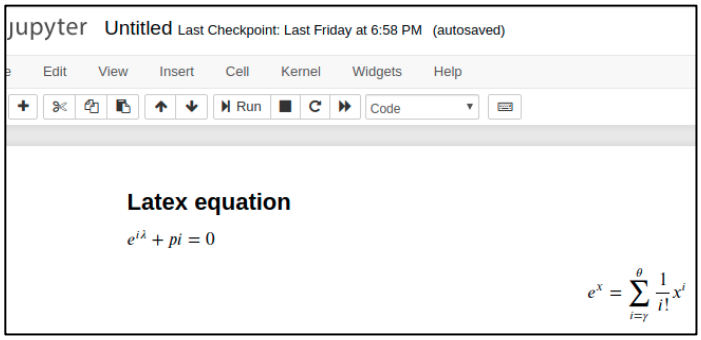
Latex expressions can be added by surrounding the latex code by '$' and for writing the expressions in the middle, surrounds the latex code by '$$'.

Example:

Input

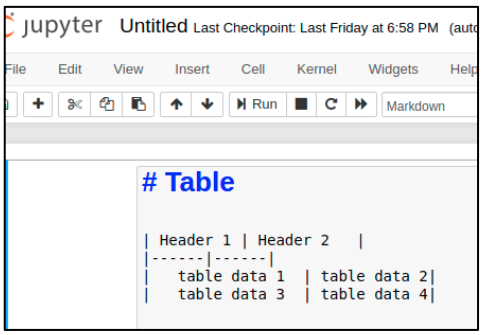


Output

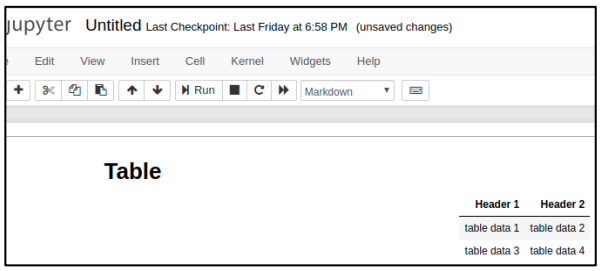


A table can be added by writing the content in the following format.

Input



Output



The text can be made bold or italic by enclosing the text in '**' and '*' respectively.

A.3.3. Kernel

A kernel runs behind every notebook. Whenever a cell is executed, the code inside the cell is executed within the kernel and the output is returned to the cell to be displayed. The kernel continues to exist to the document as a whole and not for individual cells.

For example, if a module is imported into one cell then, that module will be available for the whole document. See the below example for better understanding.

Example:

Figure : Cells in Jupyter Notebook

The order of execution of each cell is stated to the left of the cell. In the above example, the cell with In[1] is executed first then the cell with In[2] is executed.

Jupyter Notebook provides various options for kernels. This can be useful if you want to reset things:

- *Restart:* to restart the kernels and clear all the variables that were defined, clearing the modules that were imported, etc.
- *Restart and Clear Output:* This will do the same as above but will also clear all the output that was displayed below the cell.
- *Restart and Run All:* This is also the same as above but will also run all the cells in the top-down order.

- *Interrupt:* This option will interrupt the kernel execution. It can be useful in the case where the programs continue for execution, or the kernel is stuck over some computation.

A.3.3. Lunch a Notebook

To launch a Jupyter notebook, open your terminal and navigate to the directory where you would like to save your notebook. Then type the command jupyter notebook and the program will instantiate a local server at localhost:8888 (or another specified port).

A browser window should immediately pop up with the Jupyter Notebook interface, otherwise, you can use the address it gives you. The notebooks have a unique token since the software uses pre-built Docker containers to put notebooks on their own unique path. To stop the server and shut down the kernel from the terminal, hit control-C twice.

Jupyter Notebook files are very useful. You can navigate the interface using your mouse with menus and buttons or use keyboard shortcuts. They let you run small pieces of code, save your progress, or restart resetting everything.

Besides running code, you can use Markdown to organize your notebook and make it look neat and clear for others.

If you want to learn more, check out the Jupyter Notebook documentation. You can also try a notebook in your browser at https://try.jupyter.org/.

Appendix B

Useful Python librairies for Machine Leanring

Lecture Notes

B.1. Introduction

Python has become the leading programming language for Machine Learning and Data Science due to its simplicity, flexibility, and vast ecosystem of libraries. It provides powerful tools to handle large datasets, build predictive models, and automate decision-making processes.

One of Python's biggest advantages is its extensive collection of open-source libraries such as NumPy, Pandas, Scikit-Learn, TensorFlow, and PyTorch. These libraries allow developers to efficiently perform data preprocessing, model training, and evaluation with minimal effort.

Python's strong community support, combined with its integration with big data technologies and cloud platforms, makes it an essential tool for modern AI applications. Whether for image recognition, fraud detection, or medical diagnosis, Python empowers businesses and researchers to extract valuable insights from data and build intelligent systems.

We present here the main useful python librairies to start a machine leanring project.

B.2. NumPy

NumPy is a widely used Python library for handling multi-dimensional arrays and matrices, offering a broad range of mathematical operations. Its ability to perform linear algebra, Fourier transforms, and other numerical computations make it a valuable tool for machine learning and AI applications. With its efficient matrix manipulation capabilities,

NumPy helps enhance machine learning performance. Additionally, it is faster and more user-friendly compared to many other Python libraries.

B.3. Scikit-learn

Scikit-learn is a widely used machine learning library built on NumPy and SciPy. It provides support for a variety of supervised and unsupervised learning algorithms and is also useful for data mining, modeling, and analysis.

With its intuitive design and easy-to-use interface, Scikit-learn is an excellent choice for beginners in machine learning.

B.4. Pandas

Pandas is a powerful Python library built on NumPy, designed for handling and preparing high-level datasets for machine learning and training. It is based on two key data structures: Series (one-dimensional) and DataFrame (two-dimensional), making it highly versatile.

Pandas is widely used across various fields, including finance, engineering, and statistics. Despite sharing its name with a slow-moving animal, the Pandas library is fast, efficient, and highly flexible.

B.5. TensorFlow

TensorFlow is an open-source Python library designed for differentiable programming, allowing it to automatically compute function derivatives within a high-level language. It provides a flexible architecture and framework, making it easy to develop and evaluate both machine learning and deep learning models.

Additionally, TensorFlow supports model visualization and can be used across desktop and mobile platforms.

B.6. Seaborn

Seaborn is an open-source Python library built on top of Matplotlib (which focuses on plotting and data visualization) and integrates with Pandas data structures. It is frequently used in machine learning projects for generating visualizations of learning data.

Known for creating the most aesthetic and visually appealing plots among Python libraries, Seaborn is an excellent choice for not only data analysis but also marketing and presentation purposes.

B.7. Theano

Theano is a Python library designed for numerical computation, with a specific focus on machine learning. It specializes in optimizing and evaluating mathematical models and matrix operations that involve multi-dimensional arrays, which are essential for building machine learning models.

Theano is primarily used by machine learning and deep learning developers or programmers.

B.8. Keras

Keras is a Python library specifically built for developing neural networks in machine learning models. It operates on top of Theano and TensorFlow to train neural networks.

Known for its flexibility, portability, and ease of use, Keras can be seamlessly integrated with a variety of functions.

B.9. PyTorch

PyTorch is an open-source machine learning Python library built on the Torch framework, which is based on the C programming language. It is primarily used in machine learning applications that involve natural language processing or computer vision.

PyTorch is recognized for its exceptional speed in handling large, dense datasets and processing complex graphs.

B.10. Matplotlib

Matplotlib is a Python library focused on data visualization and primarily used for creating beautiful graphs, plots, histograms, and bar charts. It is compatible with plotting data from SciPy, NumPy, and Pandas. If you have experience using other types of graphing tools, Matplotlib might be the most intuitive choice for you.

Conclusion

In this course, we have explored the foundational concepts of Machine Learning and its practical applications using Python. We began with an introduction to key supervised and unsupervised learning algorithms like Linear Regression, Logistic Regression, Decision Trees, and K-Means Clustering, which are essential tools for building predictive models. We also delved into ensemble methods such as Random Forest and Boosting, which enhance model accuracy by combining multiple learners.

Through hands-on Python labs, we gained valuable experience using libraries like NumPy, Pandas, Scikit-learn, and TensorFlow, which provide powerful tools for data manipulation, model training, and evaluation. These labs reinforced our understanding of data preprocessing, feature selection, and model performance assessment techniques.

By the end of this course, you should feel confident applying machine learning techniques to real-world datasets, improving your ability to solve complex problems in fields such as finance, healthcare, and marketing. The next steps involve expanding your knowledge to advanced topics like deep learning and reinforcement learning, where Python continues to be a valuable tool for building intelligent systems.

In conclusion, Python's rich ecosystem, combined with the theoretical concepts and practical skills we've covered, equips you to begin your journey as a machine learning practitioner. Keep experimenting, learning, and applying these techniques to refine your skills further.

References

[1] Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly, 2022.

[2] John Paul Mueller, Luca Massaron. Machine Learning For Dummies, Kindle Edition, 2016.

[3] Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python: A Guide for Data Scientists, O'Reilly, 2016.

[4] Wei-Meng Lee . Python Machine Learning, Wiley Edition, 2019.

[5] https://www.datacamp.com/fr/, last access February 2025.

[6] https://ai.stanford.edu/~ang/originalHomepage.html, last access February 2025.

[7] https://www.geeksforgeeks.org/machine-learning/, last access February 2025.

[8] Nishant Shukla, Machine Learning with TensorFlow, Manning Publications, 2018.

[9] Laurence Moroney, AI and Machine Learning for Coders, 2020.

,