MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des SCiences Exactes et de l'Informatique

Département de Mathématiques et d'Informatique

Filière: Informatique

PROJET DE FIN D'ÉTUDES

Option : **Ingénierie des Systèmes d'Information**

PRESENTE PAR:

Afoun Bouchra Yousra

THEME :

Heterogeneous Parallel Ensemble Learning

Présenté le 04/07/2022 devant la commission de jury:

Mme Kaid Slimane Bouchra	FSEI, université de Mostaganem	Présidente
Mr Zahmani Habib Mohamed	FSEI, université de Mostaganem	Examinateur
Mr Henni Fouad	FSEI, université de Mostaganem	Encadrant

2021-2022

Acknowledgments

I would like to start by thanking the good lord for helping me and guiding me through each point of my career that led me here today, allowing me to gain knowledge and practice in a field that I am very passionate about.

To my parents who were my number one supporters throughout this long journey, giving me all the support that kept me pushing in the hardest moments.

To all of my sisters, my nephew, friends and loved ones for showing me unconditional support and for believing in me since day one.

To my supervisor, who made me love this field five years ago, who poured so much time and effort, guiding me and sharing a life-long knowledge throughout the years and was so keen on helping.

I would also like to express the honor to the members of the jury that accorded me and took their time to evaluate our work.

At the end, I would love to express my gratitude towards anyone who believed in me.
Thanks.

Abstract

Machine learning is a continuously developing field that benefits humans in enormous areas, such as systems automation, security, and medical examinations.

Machine learning generally aims to extract knowledge from large masses of data and fit that data into models that can be understood and utilized. In other terms, this technology provides systems that can learn and enhance from experience automatically without being specifically programmed. In many cases, one model is not enough since models can suffer from overfitting or underfitting. Ensemble learning methods solve this issue by generating multiple models and combining the results which maintain a better prediction and lead to better performance .

The goal of this research is to study, conceive and then implement a system that based on Heterogeneous ensemble learning would allow us to bypass those limits.

Keywords:

Artificial intelligence, Machine learning, Ensemble methods, Bootstrapping, Bagging, Boosting, Stacking, Homogeneous parallel learning

List of Figures

List of Tables

List of Equations

List of abbreviations

Abbreviation	Complete Expression	Page
ML	Machine learning	7
RMSE	Root Mean squared error	14
MSE	Mean Squared Error	14
MAE	Mean absolute error	15
OOB	Out-of-bag	19
SVM	Support vector machines	20
Sklearn	Scikit-learn	23
EWM	Entropy weight method	28
DST	Dempster-Shafer Theory	28
BPA	Basic probability assignment	29
3NN	3-nearest neighbor	29
GNB	Gaussian naïve Bayes	29
CV	Cross-validation	32

Table of contents

Introduction

Artificial intelligence became so mainstream that it evolved into different subsets. One of these subsets is Machine learning. This discipline allows software applications to become more accurate at predicting outcomes without being explicitly told to do so. It uses previous data as input to predict new output values. It became essential to leading companies such as Facebook, Google, and Uber since it gives a view of trends in customer behavior and supports the development of new products.

Machine learning models are trained on datasets that regroup information of the same type regarding a specific topic, whether it is medical, sports, real estate, etc. The model is trained using algorithms for different purposes depending on the type of model we are aiming for. It can either be a classifier that outputs binary or ordinal values, or linear models for continuous values. The resulting model is then tested and evaluated before being deployed to real-world usage.

During the learning phase, a situation can occur where the model sticks too much to the data and its variation, or cannot perform well on the training data itself. This is due to the lack of **Data** and lack of **Good Data**. Two types of problems linked to that exist: they are known as **underfitting** and **overfitting**. There are techniques used to overcome these limits, among them we can find ensemble methods [6]. These techniques combine multiple learning algorithms to improve the accuracy and reduce the variance of the final model.

The ensemble methods are more commonly known for the homogeneous type, such as Bagging, where the same algorithm is trained on different samples of data by using a bootstrap mechanism. The other class of ensemble methods consists in Boosting where the same algorithm is trained sequentially (through numerous iterations) to get the best model. Both Bagging and Boosting principles are implemented in available modules such as Scikit-learn[1].

In this project, we are interested in heterogeneous parallel ensemble learning where, unlike Bagging, different machine learning algorithms are used to create an ensemble learner.

[1] https://scikit-learn.org/stable/

In traditional Bagging, the diversity is created through training the same algorithm on different samples of data, whereas in this project, we aim to create the diversity through the use of different algorithms. These algorithms are combined to form the heterogeneous ensemble learner. The mixture of base algorithms can be done either by using a combination function or through the use of a meta-model (stacking).

The objective of this study is to design and implement a heterogeneous parallel ensemble learner over many base algorithms. In this first attempt, five main machine learning algorithms are considered: decision tree, logistic regression, support vector machine, Gaussian naive bayes and K-nearest neighbors. These algorithms are first trained and tested on a heart disease dataset, then combined to produce the ensemble learner which in turn is trained and tested on the same dataset. Our model improves the results by up to 30%.

The document is divided into 5 chapters :

1. **Chapter 1**: A global introduction to machine learning, model evaluation and the known limits.
2. **Chapter 2**: An introduction to Homogeneous ensemble methods and bagging
3. **Chapter 3**: Heterogeneous ensemble methods.
4. **Chapter 4**: Heterogeneous Parallel Ensemble Learning : Design
5. **Chapter 5**: Heterogeneous Parallel Ensemble Learning : Implementation

Chapter 1

Machine Learning

1.1 Introduction

During the past few years, Artificial Intelligence (AI) started getting more and more visibility and has become a well-known term for many people thanks to its many strong points, such as its revolutionary solutions and the wide range of products used in most households. Sub-branches of AI were born for different needs, such as machine learning and deep learning. Machine learning (ML) is the part of AI that can learn from previously generated observations (data) without being explicitly told to do as such. It facilitates the process of extracting knowledge from data which, in turn, makes it excel in solving complex, data-rich problems where traditional approaches such as human judgment and software engineering sometimes fail.

Machine learning predicts future outcomes based on pre-existing data. In addition, it can lead to a variety of automated tasks; which makes this technology affect virtually every industry, from weather forecasting to stockbrokers looking for optimal trades. Machine learning requires complex math and a lot of coding to achieve the desired functions and results. It also incorporates classical algorithms for various kinds of tasks such as clustering, regression, or classification. We have to train these algorithms on large amounts of data. The more data you provide for your algorithm, the better your model gets. There are four types of machine learning algorithms: supervised, semi-supervised, unsupervised, and reinforcement.

To build efficient models and extract features, many started combining data mining with machine learning, which is currently considered a crucial part of its process.

1.2 Data mining

Data mining is a key part of Machine Learning and one of the core disciplines in data science, which uses advanced analytics techniques to find valuable patterns and trends hidden within vast volumes of data [14].

1.2.1 Process

The process of data mining can be divided into four steps, starting with data gathering, then data preparation moving to the mining, and finally data analysis and interpretation.

1.2.1.1 Data gathering

It is the procedure of collecting relevant data for an analytics application to be identified and assembled. The data can be located in different source systems. The more information collected, the better the analysis is, under the condition that the source is reliable [14].

1.2.1.2 Data preparation

It is a set of steps that aim to make the data ready to be mined by keeping only the necessary one and removing the unwanted so that it would not lead us to false conclusions. It starts with data exploration, profiling, and pre-processing, then cleansing work to fix errors and other data quality issues, it can be followed by data transformation to make datasets consistent [24].

1.2.1.3 Mining the data

It is the process that starts with selecting the appropriate data mining technique, then implementing one or more algorithms to do the mining. However, in machine learning applications, the algorithms typically must be trained on sample datasets to look for the information being sought before they are run against the full set of data [14].

1.2.1.4 Data analysis and interpretation

It is the process of assigning meaning to the collected information from the data mining, finding patterns, and determining the conclusions, significance, and implications of the findings; this can help drive decision-making and other business actions [14].

1.2.2 Models

Many data mining techniques can be used to turn raw data into actionable insights. These techniques are divided into two types, namely, Descriptive and Predictive models.

1.2.2.1 Descriptive modeling

A descriptive model distinguishes relationships or patterns in data. It detects the similarities between the collected data and the reasons behind them. It also serves as a way to explore the properties of the data being examined. Clustering, summarization, associating rules, and sequence discovery are descriptive model data mining tasks [1].

1.2.2.2 Predictive modeling

Predictive modeling is an approach based on the analysis of various historical data to create, process, and validate a model that can be used to predict future behaviors [1].

1.2.3 Applications of data mining

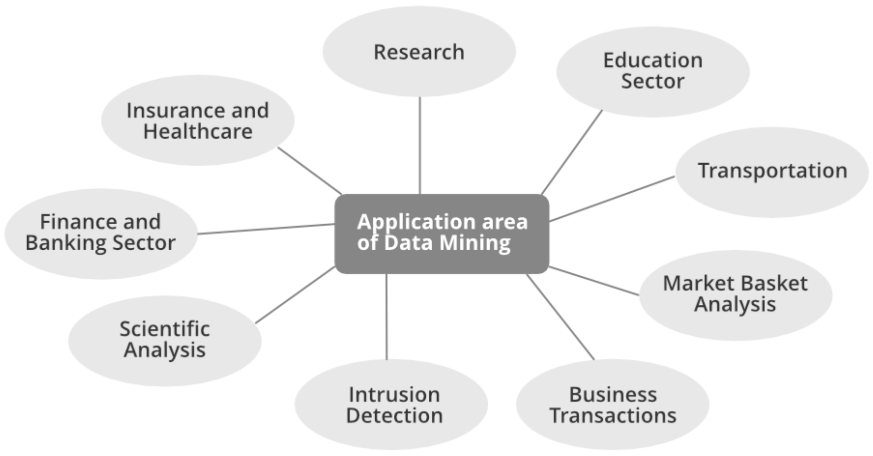


Figure I.1 : Data mining applications

As can be seen in Figure I.1, data mining can be used for different applications that can vary from simple things like marketing to very complex domains like making environmental disaster predictions. Some of these applications will be discussed in the following sections [1].

1.2.3.1 Marketing

Using data mining, we can analyze customers' behaviors for targeted advertising [1].

1.2.3.2 Customers' purchase behavior

Data Mining will help to identify trends of customers for goods in the market then allowing the retailer to understand the purchase behavior of a buyer [1].

1.2.3.3 Education

Learning institutions would be able to upgrade the proposed courses based on the behaviors of students extracted using data mining [1].

1.3 Learning methods

A machine learning algorithm, also called model, is a mathematical expression that represents data in the context of a problem. The aim is to go from data to insight. The process of learning can be divided into two major categories: unsupervised learning and supervised learning.

In unsupervised machine learning, the desired output is not given; the techniques that follow this approach extract conclusions from datasets that consist of the input data without the labeled response.

Supervised machine learning techniques attempt to find a relationship between input attributes (independent variables) and a target attribute (dependent variable). These techniques can further be classified into two main subcategories: classification and regression. In regression, the output variable takes continuous values while in classification the output variable takes class labels [10][27].

1.3.1 Regression

Regression is an approach that is used when the output of a problem is continuous. Different models exist, their usage varies according to the nature of the data, the most popular among these techniques is linear regression [29].

Linear regression is one of the most basic types of regression in machine learning that consists of a predictor variable and a dependent variable related linearly to each other. In case the data involves more than one independent variable, then linear regression is called multiple linear regression [13].

1.3.2 Classification

Classification is an approach that is used to forecast group membership for data instances. It represents the process of recognizing, understanding, and grouping ideas and objects into preset categories using pre-categorized training datasets. The classification type is widely used in Machine learning, including Binary classification and Multi-class classification [12].

1.3.2.1 Binary classification

Refers to classification techniques that divide instances into two classes. Generally, the results involve one class that is the normal state and another class that is the abnormal state. For example: email spam detection (spam or not) [3].

1.3.2.2 Multi-class classification

Multi-class classification is a classification task with more than two classes.This method makes the assumption that each sample is assigned to one and only one category. For example : classify a set of images of fruits which may be oranges, apples, or pears, where fruit can be either an apple or a pear but not both at the same time [20].

1.4 M odel evaluation

In order to determine if we are on the right track or should furthermore adjust our model, we should evaluate it. Evaluation is one of the most important steps of the machine learning process. This step allows us to detect errors at an early stage.

The most important aspect needed to properly evaluate a predictive model is to not train it on the entire dataset. A typical train/test would be to use a portion of the data (70% in most cases) for training and the remaining portion for testing. This would prove useful when trying to prevent overfitting [15].

1.4.1 E valuation methods

In order to evaluate model accuracy, metrics are made available so that analysts would test how robust their model is. The choice of metric completely depends on the type of model and the implementation plan of the model [1.8].

1.4.1.1 Classification metrics

When performing classification predictions, the prediction can be considered as one of the four types below [1.8]:

- **True Positives (TP):** When a prediction of observation is correctly classified.
- **True Negatives (TN):** When a prediction of an observation not belonging to a class is correct.
- **False Positives (FP):** When an observation is predicted to belong in a class but in reality, it does not.
- **False Negatives (FN):** When an observation is predicted to not belong in a class but in reality it does.

These four outcomes form what is called the confusion matrix, which can be used to compute different metrics to evaluate the model. The most used ones are described below.

a. Accuracy

It is defined as the percentage of correct predictions for the test data, it can be calculated using the following formula (Equation I.1):

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

Equation I.1: Accuracy equation

b. Precision

It is defined as the fraction of relevant observations among those predicted to belong in a certain class. It is calculated using the following formula (Equation I.2):

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

Equation I.2: Preci equation

c. Recall

It is defined as the fraction of relevant observations among those that belong to a certain class. It is calculated using the following formula (Equation I.3):

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

Equation I.3: Recall equation

1.4.1.2 Regression metrics

Evaluation metrics for regression models are different than the ones used in classification problems, that is because the prediction is in a continuous value rather than a defined number of outcomes;

The goal now is instead of checking which class the prediction falls into, we would rather have a metric that would determine if the prediction was good or not [4].

d. Mean Squared Error

MSE (Mean Squared Error) is an important loss function for algorithms, it uses the least squares framing of a regression problem. Here “least squares” refers to minimizing the mean squared error between predictions and expected values.

The MSE is calculated as the mean or average of the squared differences between predicted and expected target values in a dataset (Equation I.4) [5].

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (Expected_i - Predicted_i)^2$$

Equation I.4: MSE equation

e. Root Mean Squared Error

RMSE (Root Mean Squared Error) is an extension of the MSE, with the same units and not the square of the unit. It may be common to use MSE to train a regression model while using the RMSE to evaluate its performance (Equation I.5).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (Expected_i - Predicted_i)^2}$$

Equation I.5: RMSE equation

f. Mean absolute error

MAE is a metric where, like RMSE, error units match with the target units. The difference is that, unlike RMSE, the changes in MAE are linear and therefore intuitive.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} \quad |Expected_i - Predicted_i|$$

Equation I.6: MAE equation

1.4.2 L imits of classic models

During our learning, we may encounter a situation where our model sticks too much to the data and its variation, or can not perform well on the training set itself. This is due to the lack of **data** and lack of **good data**. Two types of problems linked to that exists, underfitting and overfitting:

1.4.2.1 U nderfitting

Underfitting is the scenario where a model is unable to capture the relationship between the input and output variables accurately, which leads to a high error rate on both the training set and predicted data [6].

To prevent underfitting, we can train our model using more data over more training time.

1.4.2.2 O verfitting

Overfitting refers to the situation where the model is useful for the training data set, and irrelevant for any other data sets (irrelevant in generalization). This happens when the model learns the detail and noise of the training data to the extent that it negatively affects the performance of the model on new data [6].

To prevent overfitting, several options are available, the most popular solutions are:

- **Cross-validation :** It is a standard way to find out-of-sample prediction errors. It is usual to use a 5-fold cross validation [7].
- **Early Stopping :** Its rules provide us with guidance as to how many iterations can be run before the learner begins to over-fit [7].

- **Training with more data :** Training with more data can help the algorithm detect association rules better. This would not work if the added data is noisy.

In addition to those techniques mentioned above, another type of methods to prevent from overfitting/underfitting, they are called **Ensemble methods** [37].

1.5 Ensemble methods

Ensemble methods are techniques that combine multiple learning algorithms to produce improved results and optimize better predictive performance. These methods reduce overfitting in models and make the model more robust.

1.6 Conclusion

Data mining, which is a subset of machine learning, provides us with many different types of classic learning methods such as classification and regression, but they both have a limit. In this first chapter of our scientific study, we tried to make a clear summary of machine learning and data mining. This chapter also cited learning methods and how to measure their performance and also their limits. We also mentioned methods that would help us prevent and overcome those limits. Now we can carry on in the next chapter to explore one of these solutions, Ensemble methods.

Chapter 2

Homogeneous Parallel Algorithms (Bagging)

2.1 Introduction

Ensemble methods rely on a notion known as the “wisdom of the crowd”. It is the idea that a combined answer of many diverse models is often better than any one individual answer. The ensemble learning approach embodies this process by generating multiple models and combining them, intending to improve model performance.

These methods fall into two broad categories: sequential ensemble techniques which generate base learners in a sequence, in a way that each model is dependent on the previous one (Figure II.1, B); and parallel ensemble techniques that utilize the parallel generation of base learners to encourage independence between the base learners then combines predictions from individual learners to get the final prediction of the ensemble learner (Figure II.1, A).

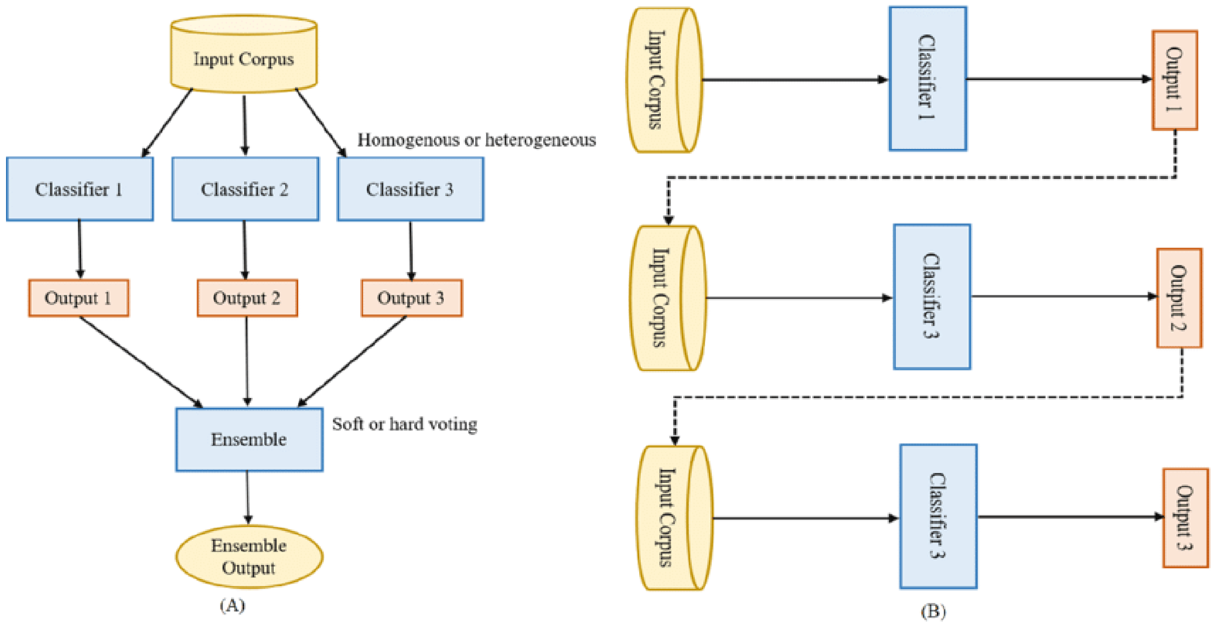


Figure II.1 : Ensemble Learning Techniques

Parallel ensemble methods can additionally be divided into homogeneous and heterogeneous parallel ensembles based on the learning algorithms used in the process.

Homogeneous ensemble methods use the same learning algorithm while heterogeneous ensemble methods train the models using different learning algorithms.

One of the simplest ensemble algorithms with high performance under the parallel homogeneous methods is known as Bagging [30][33], which will be defined in this chapter, as long as its principles, down to its implementation.

2.2 Bagging: Bootstrap Aggregating

Bagging is a short term for bootstrap aggregating; it was introduced by Leo Breiman in 1996 [30]. This ensemble technique is the most basic homogeneous parallel ensemble method that can be constructed to either improve ensemble diversity or overall computational efficiency.

This technique trains base estimators on replicates of the dataset that result from multiple base estimators from a single dataset and a single learning algorithm. This process is achieved through bootstrap sampling which guarantees ensemble diversity. The next step is performing ensemble prediction through model aggregating [30].

The following paragraphs introduce the 3 steps of bagging as illustrated in Figure II.2

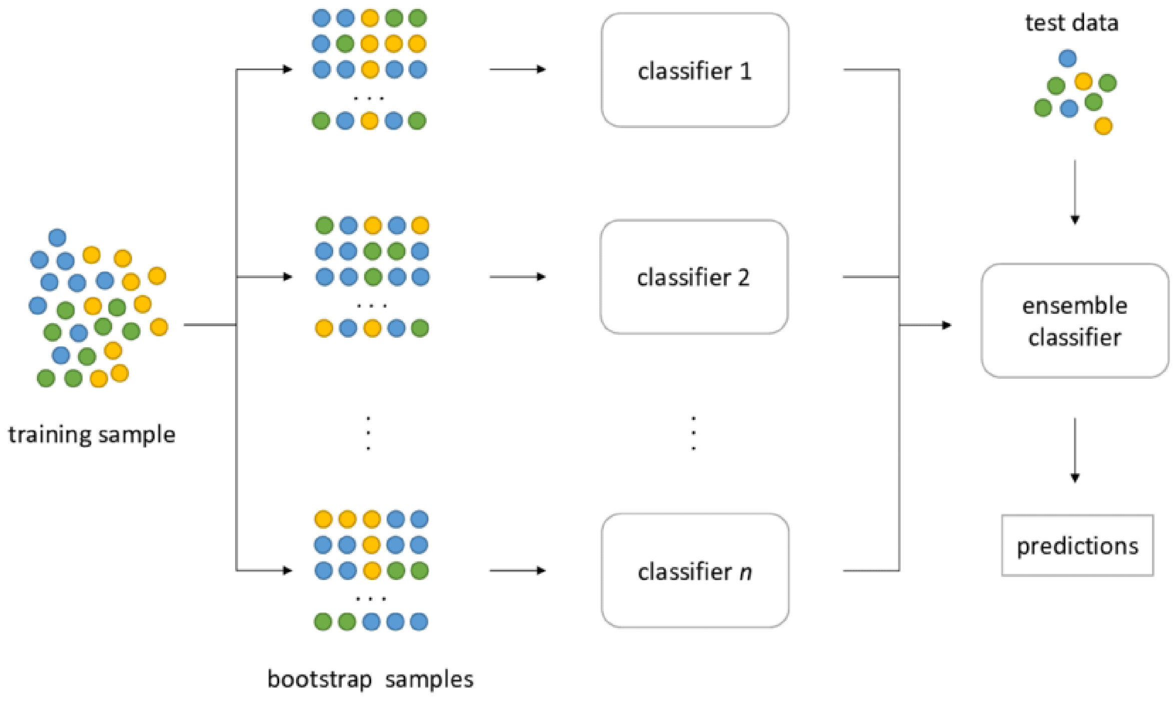


Figure II.2 : The process of bagging

2.2.1 Bootstrap Sampling

A bootstrap sample is a smaller sample that is a subset of the initial dataset. The bootstrap sample is created from the initial dataset by sampling with replacement. Suppose for a dataset having n rows and f features, we perform a bootstrap sampling that refers to sampling with replacement into k different smaller datasets each of size m with the same f features. Each smaller dataset D_i formed is used to train the chosen algorithm.

When sampling with replacement, some objects that were already sampled have a chance to be sampled a second time (or even a third, or fourth, and so on) because they were replaced. Some objects may be sampled many times, while some objects may never be sampled [35].

Thus, bootstrap sampling naturally partitions a dataset into two sets: a bootstrap sample (with training examples that were sampled at least once) and an out-of-bag (OOB) sample (with training examples that were never sampled even once).

We can use each bootstrap sample for training a different base estimator. Since different bootstrap samples will contain different entries (some entries are used in different samples), each base estimator will turn out to be somewhat different from the others.

In Figure II.3, the initial dataset D of shape (n, f) is sampled to k datasets each of shape (m, f), where $m<n$ [35].

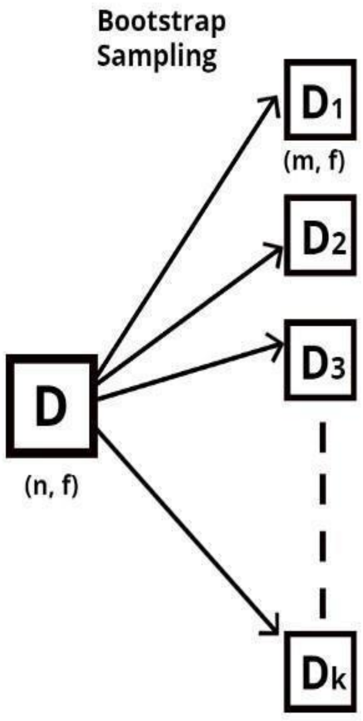


Figure II.3 : Bootstrap sampling

Figure II.4 is an illustration of the bootstrap procedure. The dataset D having 10 rows is sampled with replacement into k smaller datasets each having 5 rows. Here $n=10$ and $m=5$. It is observed that each of the datasets formed by bootstrapping sees only a part of the original dataset and all the datasets are independent of each other [35].

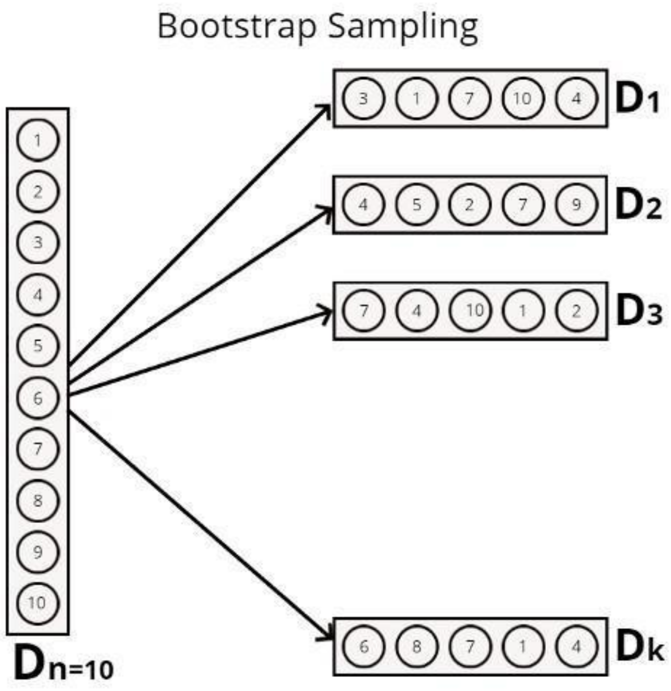


Figure II.4 : Detailed bootstrap sampling

This is the first step of the bagging ensemble technique in which k smaller independent datasets are created by bootstrapping [35].

2.2.2 Modeling

Modeling is the second step of bagging. After k smaller datasets are created by bootstrapping each of the k datasets is trained using the same ML algorithm. In the training phase, we can either use this algorithm with similar or different configurations for our models. For example, Decision Tree algorithms can be used as a base model while changing hyperparameters such as ‘depth’. A combination of different algorithms such as SVM, Naive Bayes, Logistic Regression can be used. The models which are trained on each bootstrap dataset are called base models or weak learners. Figure II.5 describes the training of each dataset of separate models [35]:

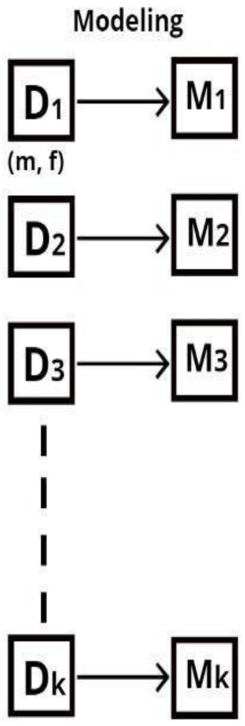


Figure II.5 : Modeling samples

2.2.3 Aggregation

A final powerful robust model is created by combining the k different base models (as can be seen in Figure II.6). Since the base models are trained on a bootstrap sample, each model may have different predictions. An aggregation technique is different depending on the problem statement [35][33].

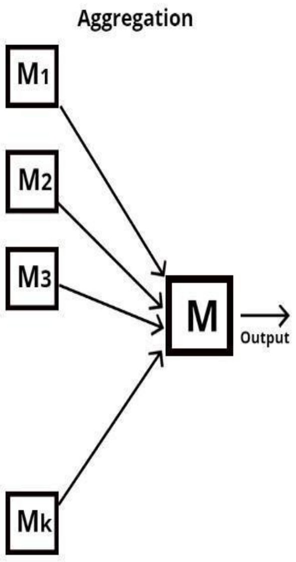


Figure II.6 : Aggregation

- For a regression problem: The aggregation can be taking the mean of prediction of each base model (Equation II.1).

$$prediction = \frac{1}{k} \times \sum_{i=1}^{k} \quad pred_i$$

Equation II.1: $calculating\ regression\ prediction$

- For a classification problem: The aggregation can be using majority voting, the class having the maximum vote can be declared as the final prediction (Equation II.2).

$$prediction = \ argmax_C(pred_i[1, 2, 3, \cdots, c])$$

Equation II.2: $calculating\ classification\ prediction$

2.2.4 Parallel Training

Bagging is a parallel ensemble algorithm as it trains each base learner independently of other base learners. This means that training bagging ensembles can be parallelized if one has access to computing resources such as multiple cores or clusters. The bootstrap samples would be trained independently and in parallel with each other using weak or base learners [35][33].

2.3 Random Forests

Random forest is a special case of bagging where the learning algorithm used is the decision tree algorithm; it increases diversification by adding more randomness. It is a widely utilized technique and a popular go-to method for many applications, especially bioinformatics, due to its computational efficiency in training. Moreover, it ranks data features by importance, which is very useful for high-dimensional data analysis [2].

2.3.1 Randomized Decision Trees

Random forest is a particular case of the bagging algorithm that is constructed using randomized decision trees as base estimators. Starting with performing bootstrap sampling to generating a training subset, then it trains the base estimators using a modified decision-tree learning algorithm [2].

The main difference between the decision tree and the random forest algorithm is the construction of their nodes. In standard decision tree construction, all the features are evaluated, and the one with the highest information gain is considered as the best feature, this feature is selected to split the data/dataset. However, if we use the bagging on the standard decision tree algorithm the results will be approximately similar since we are relying on one feature, in a way the similarity of the result can lead to low variance [2].

Random forest eliminates this problem by introducing randomness in the process of building trees. While selecting the features, instead of evaluating all the features to identify the split with the highest information gain. This technique prepares a random subset of features, then evaluates each subset to identify the best feature to split on, therefore, increasing ensemble diversity and improving the predictive performance [2].

2.3.2 Features Importance

In the modeling concept, feature selection is a crucial process. It affects the model accuracy and performance, especially when using high-dimensional data. Fortunately, the random forests technique enables us to rank the features by importance and identify the most pertinent ones and drop low impact features. This gives us the ability to ensure that the models are trained with the most relevant features.

Another important step is ejecting features. This is applied on the least relevant features which minimizes overfitting, especially since a large number of features can inhibit the model's ability to generalize effectively. This helps improve generalization and computational performance [30][33].

We can also encounter a case where multiple features are strongly correlated or dependent. then intuitively, dropping one of them wouldn't affect the model. However, the order in which features are used can prioritize ones and reduce the importance of the others. This problem can be mitigated somehow through random features selection.

2.4 Conclusion

In this chapter we discussed how we can improve accuracy and reduce variance of the prediction results by using homogeneous parallel algorithms, also called homogeneous ensembles. We then went on to present one of the most efficient, yet simplest homogeneous algorithms, known as Bagging.

The main difficulty regarding homogenous ensembles is to generate diversity, bagging solves it using bootstrap sampling, but other techniques such as boosting exist. In the next chapter, we will discuss another approach that solves it using different learning algorithms, it is known as Heterogeneous ensemble learning.

Chapter 3

Heterogeneous Parallel Algorithms

3.1 Introduction

The main goal of bagging algorithms is improving accuracy and reducing variance, which eliminates the problem of overfitting. It deals with homogeneous models that are considered as weak learners. It trains them independently and in parallel, and at the end of the process, it combines them following a deterministic process [29].

As discussed in the previous chapter, homogeneous ensembles are practically composed of classifiers of the same type. The ones based on classifiers of different types are called heterogeneous. Even when using the same learning algorithm, the main difficulty regarding homogenous ensembles is to generate diversity. Different strategies are used for both of the mentioned ensembles, bagging solves it through bootstrapping, but it can also be solved using boosting, etc... These techniques, which have mainly been used in bagging, can also be used to reach additional diversity in heterogeneous ensembles. Yet, since various learning algorithms are used to generate the base learners, heterogeneous ensembles are naturally various. The main difficulty in this case resides in defining the optimal way to combine the different predictions of the models in the ensemble [29].

3.2 Principle

Heterogeneous ensemble learning is divided into two main steps. This process starts with training a set of models using different base learning algorithms, then combining the base estimators results using the weighting approach or the meta-learning which is a model that predicts the final results.

The first step in building heterogeneous ensembles for any application is training a set of base estimators using different learning algorithms. Choosing the best base estimators is a crucial step. The key is to ensure that we choose algorithms that fit the problem and that are different enough to produce diversity. This difference in the learning process will explicitly visualize the decision behavior and boundaries of each base estimator along with the diversity of the estimators.

Unlike the homogeneous ensemble learning that uses different training sets for each base estimator, in the heterogeneous ensembles approach the base estimators are trained using the same training set. Then a validation set is used to evaluate each base estimator individually and, eventually, a test set is used to estimate the final model performance. The three subsets are mutually exclusive, as they do not have any overlapping examples.

In the test phase, which determines the performance of the overall ensemble, we collect the predictions of each test-set example by each trained base estimator, which represents each base estimator's confidence in its predictions. Then we combine the results using one of the two approaches.

3.3 Combining predictions

As mentioned in the previous section, one of the pillars of the heterogeneous ensembles approach is combining the results. In the following paragraphs, we will discuss the two main approaches used to obtain the final prediction of an heterogeneous ensemble model.

3.3.1 Classic approach

For combining predictions, we can use equations to achieve the result we are looking for. This approach is called the classical approach. In the following, we will discuss some of the main methods/equations used.

Most of these equations base their computations on a value called weight. This weight is generated from the given model results and it differs from equation to equation. It allows us to give more importance to the prediction of a model compared to the other models.

3.3.1.1 Majority Vote

The voting methods rely on a democratic process that combines the predictions provided by the classification models independently [25]. Among these methods, the majority vote is the most simple and intuitive approach, since it assigns for each base estimator an equal weight [31].

In the classification problems, when using the majority vote strategy, each classifier predicts a class, and then, the class in which the majority predicted would win. In other terms, we choose the most common (popular) prediction. Figure III.1 illustrates this approach.

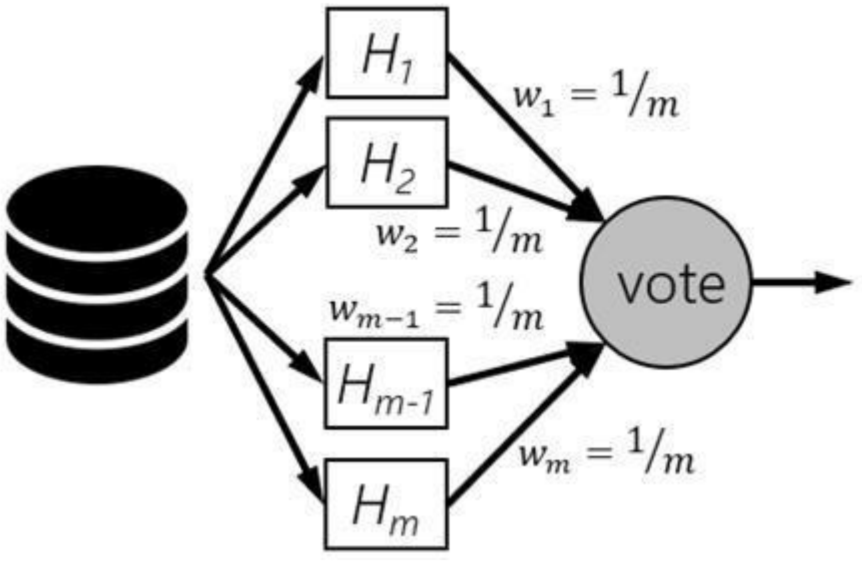


Figure III.1 : Combining predictions in majority vote

3.3.1.2 Accuracy weighting

When using multiple base estimators on the same training set, it is better to know which of these base estimators has the best performance, as it is mostly going to give the best predictions as well.

Likewise, knowing which of the base estimators is well trained helps to combine the final predictions, since we have a better view of which of these base estimators has the best impact. In this approach, each base estimator is assigned a weight that represents how well this base estimator performed which is also known as the accuracy

The accuracy of a machine learning classification algorithm represents the most intuitive measure that provides how often the algorithm classifies a data point correctly. In other terms, this measure is simply the percentage of its predictions that turn out to be correct.

After training the base estimators, each one of them would be evaluated and its accuracy computed using the validation set and the formula presented in section (1.4.1.1) [10,18]. The accuracy would then help us take into consideration the prediction of the best model since the higher the accuracy is, the better the estimator is. With this measure we can combine the final prediction (equation III.1)

$$\frac{1}{\sum_{j=1}^{n} A_i} \times \sum_{i=1}^{n} A_i . P_i$$

Equation III.1: $Accuracy\ combining\ equation$

i: represent the index of the base estimator

Ai: the accuracy of the **i**th base estimator

Pi: the prediction made by the **i**th base estimator

3.3.1.3 Entropy weighting

Another commonly used weighting method is the entropy weight method (EWM). It was introduced in the field of information theory by Claude Shannon to quantify the amount of information conveyed by a variable [25]. This method allows us to measure the value of dispersion in decision-making. The EWM measures the uncertainty in a set of events. It can be used as an evaluation metric to judge the value of the estimation. Thus, this notion can evaluate the classifiers.

The entropy of a base classifier is computed using only the predicted labels (equation III.2) which defines how uncertain a classifier is about its predictions.

$$E = -\sum_{i=1}^{n} P_i Log_2 P_i$$

Equation III.2: $Entropy\ \ equation$

The low entropy (uncertainty) signifies a better classifier. Thus, individual base classifier weights are inversely proportional to their corresponding entropies (equation III.3). Eventually, all the base estimator predictions are combined taking the weights into consideration.

$$W_i = \frac{\frac{1}{E_i}}{\sum_{i=1}^{n} \frac{1}{E_i}}$$

Equation III.3: $Entropy\ combining\ equation$

3.3.1.4 Dempster - Shafer combination

This approach is based on the Dempster-Shafer Theory (DST) which allows fusing beliefs and evidence from multiple sources, such as base estimators, into an overall final belief, or prediction probability [29].

DST is a mathematical theory of evidence [29]. This theory was introduced by Arthur P. Dempster in the context of statistical inference, and later it was developed by Glenn Shafer into a general framework for modeling epistemic uncertainty [28]. This theory allows one to combine evidence from different sources and arrive at a degree of belief (represented by a mathematical object called belief function) that takes into account all the available evidence [28].

DST uses the basic probability assignment (BPA) to combine the beliefs. This measure allows us to translate an estimator's confidence to a belief over the true label. The BPA is simply a number between 0 and 1 that indicates the belief in a proposition such as "a test example x belongs to Class 1”. Using the BPA value we can express the certainty that the test example x belongs to Class 1. The decisions made with more certainty are characterized with BPA values closer to 1 [29][22].

3.3.2 Meta Model approach

We saw in the prior section one of the many approaches to assemble heterogeneous ensembles classifiers: weighting. Each classifier was weighted by its performance and used a pre-established function that we had to carefully design so that it would combine predictions of each classifier reflecting our performance priorities.

Another methodology to construct heterogeneous ensembles is meta-learning. In this approach, instead of designing the function ourselves, we would train a second-level meta-classification algorithm that combines inputs that consist of predictions generated by the base estimators.

Meta-learning techniques are already widely and successfully applied to a variety of tasks in chemometrics analysis, recommendation systems, text classification, and spam filtering. Meta-learning methods, stacking, and blending for recommendation systems were brought to light after being used by top teams during the Netflix prize competition [32].

3.3.2.1 Stacking

One of the most common meta-learning methods is stacking. This method stacks a second classifier on top of its base estimators and it consists of two steps :

- **level 1**: This step is similar to bagging. It aims to create a diverse set of base classifiers by fitting base estimators on training data.
- **level 2**: Based on the outputs generated by the base classifiers, this step would see us construct a new data set. The inputs it receives would become meta-features that can either be predictions or the probability of predictions.

Let us construct a simple heterogeneous ensemble considering a 2d synthetic dataset and two classifiers (3-nearest neighbor and Gaussian naïve Bayes). We would train the above-mentioned classifiers then use their classifications to create new features (called meta-features) as illustrated in Figure III.2. Therefore, for each training example "Xi", we generate two meta-features, "Yi-3nn" and "Yi-gnb": the prediction probabilities of Xi according to 3nn and gnb respectively [29].

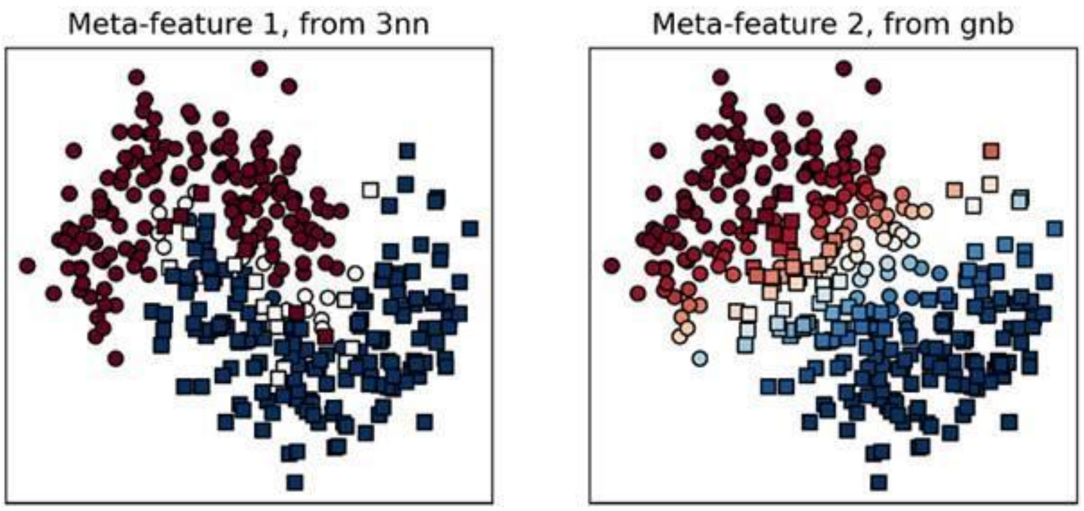


Figure III.2 : Meta features from 3nn and gnb.

These generated features would become meta-data for a second-level classifier. It would use them as a new training set to learn a combination function, differing from the combination by weighting that would use them directly in predetermined functions [29].

An infinite number of base estimators can be used in Stacking since our objective is to ensure sufficient diversity through the predictions of these base estimators. Figure III.3 below shows the schema of six popular algorithms [29].

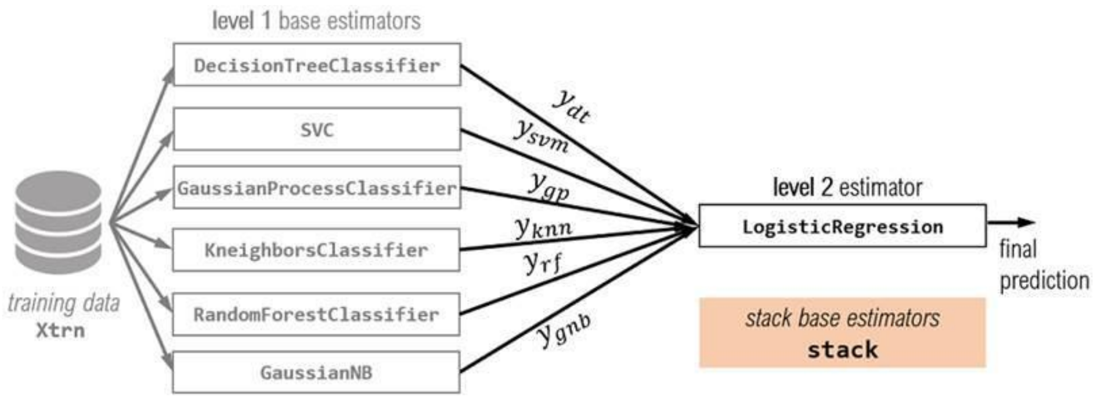


Figure III.3 : Stacking in details

For the level 2 estimator, any base learning algorithm can be used to train it. Linear models like linear and logistic regression proved to be effective when used. A model using these kinds of linear methods is called linear stacking. It is popular because of how fast and computationally efficient it is even for large datasets. It can often be an effective exploratory step in analyzing datasets [29].

Regardless, stacking also uses non-linear classifiers as meta-models such as artificial neural networks, etc... This lets the ensemble combine meta-features in complex ways but at the cost of interpretability in linear models.

Going back to our example, we can implement a linear stacking procedure passing by two steps. The first one consists of training the base estimators. The other would see us construct meta-features from training the base estimators and then train a linear regression model. It should be important to note that the meta-data can either be the predictions or the prediction probabilities [29]. The result would be slightly different as can be seen in Figure III.4.

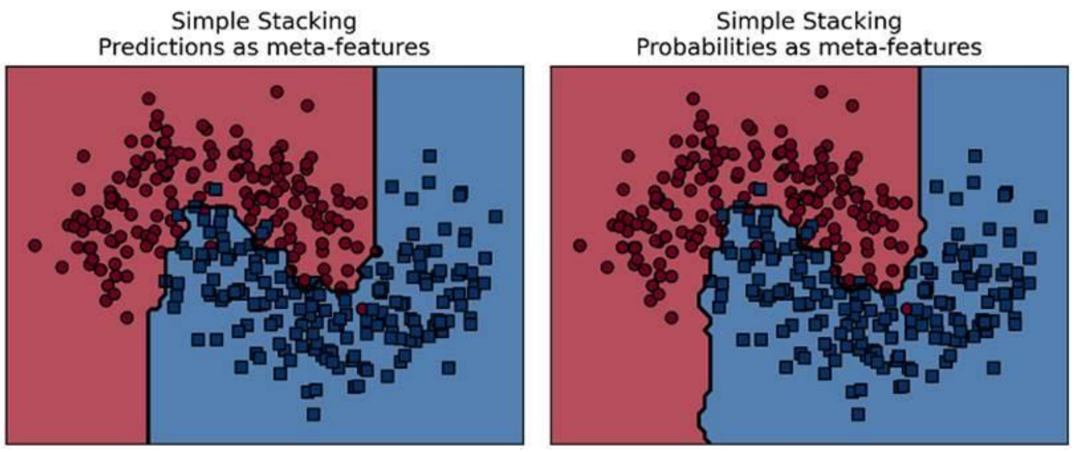


Figure III.4 : Stacking results using different types of meta-features

To generate predictions, we proceed by two steps:

1. Get the meta-features from the trained level 1 estimators for each test example, and create a corresponding test meta-example.
2. For each meta-example, get using the level 2 estimator a final prediction.

To prevent overfitting, we can implement k-fold cross-validation so that each base estimator is trained on a different data set, but instead of using it for parameter selection and model evaluation, here, we use it to partition the dataset into subsets so that the base estimators train on different subsets, this often leads to better diversity and more robustness, while reducing the odds of overfitting.

3.3.2.2 Stacking with cross validation

Cross-validation is a resampling technique for model validation and evaluation used to simulate out-of-sample testing, tune a model's hyperparameters and test its efficiency and accuracy [32][21].

The dataset would be partitioned into k subsets, thus the prefix “k-fold”. In a 5-fold cross-validation example, data is (often randomly) partitioned into five non-overlapping subsets. It then leads to 5 folds or combinations of these subsets for training and validation as shown in Figure III.5.

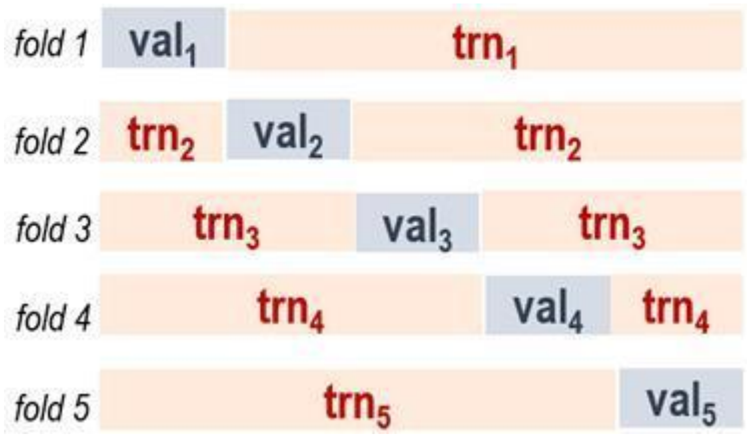


Figure III.5 : 5-fold cross-validation

For example, we split a dataset D into five subsets: D_1, D_2, D_3, D_4 and D_5.These subsets are disjoint, each fold would constitute a training set and a validation set, that includes the subset that was excluded from the training set. This fold would allow us to train and validate one model. Overall, a 5-fold CV allows us to train and validate five models.

The cross-validation procedure will be different in our case. The validation sets k would be used for generating meta-feature for the level 2 estimator instead of using them for evaluation.

Combining stacking with cross-validation goes through the following steps:

1. Generating k equal-sized subsets randomly from the data set;
2. For each base estimator, train k models using the training data "trn_k" from the corresponding k-th fold.
3. From each trained base estimator, we would use the validation data "val_k" of the corresponding k-th fold to generate k sets of meta-examples
4. Retrain each level 1 base estimator on the complete data set.

The three first steps of this process are shown in Figure III.6.

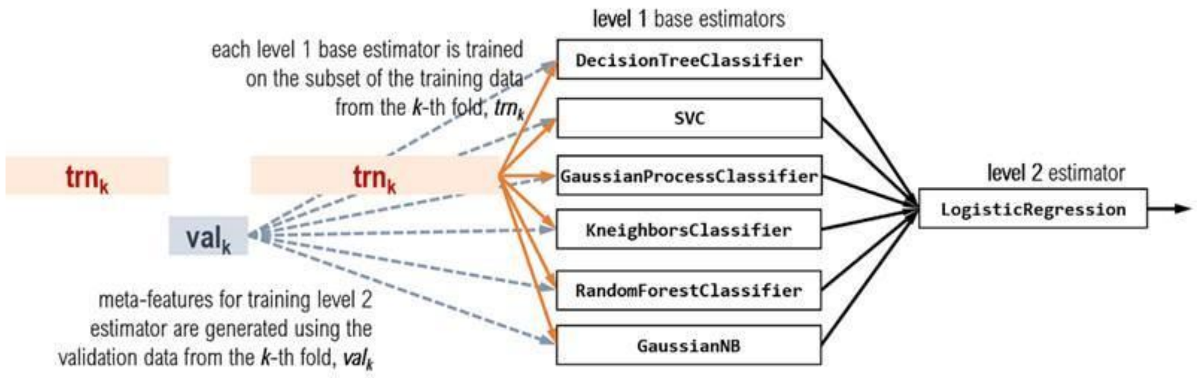

level 1 base estimators
each level 1 base estimator is trained on the subset of the training data from the k-th fold, trn_k
DecisionTreeClassifier
SVC
GaussianProcessClassifier
KneighborsClassifier
RandomForestClassifier
GaussianNB
trn_k
trn_k
val_k
meta-features for training level 2 estimator are generated using the validation data from the k-th fold, val_k
level 2 estimator
LogisticRegression

Figure III.6 : Stacking using cross-validation

Since this approach consists of adding cross-validation, hence additional training time, it is worth mentioning that for a small-sized data set this increase is well worth the cost. But for larger data sets, it can be significant.

It is usually acceptable to hold out a single validation set rather than several cross-validation subsets if the model is too expensive to train, this procedure is known as blending.

3.4 Conclusion

In this chapter, we discussed heterogeneous ensembles and its principle that consists of training models using different algorithms and then combining their results. We also saw that different approaches can be used to combine these results, classical ones based on arithmetic operations and another based on meta-models.

Chapter 4

Heterogeneous Parallel Ensemble Learning : Design

4.1 Introduction

As described in the previous chapters, ensemble learning can be done in either sequential or parallel mode. This chapter presents the design and implementation of a system that uses a set of heterogeneous base estimators to construct an ensemble parallel learner capable of achieving better predictions.

4.2 Process

The process of construction of a heterogeneous parallel learner follows mainly the same steps as when designing a homogeneous parallel estimator. These steps are described in the following sections.

4.2.1 Data splitting

This step consists in splitting the data to a training set and a testing set that will be used by all estimators (base + meta). Usually, the splitting ratio varies from 20% to 30% for the test dataset. Additionally, we can use bootstrapping for better results.

4.2.2 Training base estimators / and meta-model

This step of the process consists of training all the estimators (base + meta) using a heterogeneous set of estimators as explained in the previous chapter. During this phase, we would try to do the shared task (training) in parallel. This would prove to be less time consuming than if it has been done sequentially.

4.2.3 Fitting base estimators / and meta-model

This step of the process consists of testing all the trained estimators (base + meta) using the test data. During this phase, the shared task (training) can obviously be done in parallel.

4.2.4 Combining the predictions

Once the base models are trained and eventually tested, their results can be combined to get the ensemble parallel estimator. Many combining strategies can be applied.

4.3 Models used

As described in the previous sections, ensemble learning methods require base estimators / models to achieve their learning part. In our system, and since it is heterogeneous, we used different models to serve as base estimators and another model that will eventually serve as the meta-model.

In the following sections, we are going to present each one of these algorithms as well as their main features.

4.3.1 Support Vector Machine (SVM)

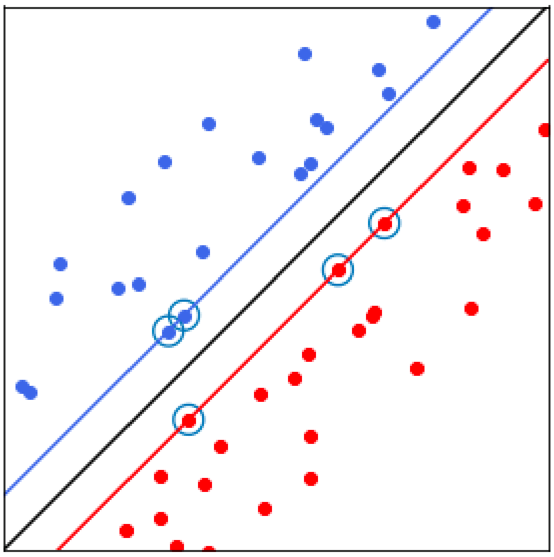


Figure IV.1: Support Vector Machine hyperplane

Support Vector Machine, abbreviated as SVM, can be used for regression and classification tasks. It is mainly and widely used in classification problems. Many research studies consider SVM as the off the shelf machine learning algorithm due to its significant accuracy with less computation power. In this study, SVM is experimented as a base estimator for the construction of a heterogeneous parallel learner [26][25].

The objective of the SVM algorithm is to find the best hyperplane in N-dimensional space (N-1 — the number of features) that best separates the data points. In a binary problem, to separate 2 categories, SVM moves the data into a high dimension space and finds a high dimensional SVC (Support Vector Classifier) that can effectively classify the observations [26][25].

SVC takes into consideration many different parameters, we will explore some of them in the following.

4.3.1.1 Hyperplane

A hyperplane is simply a function that helps differentiate between features, the hyperplane dimension depends on the number of input features in the dataset. If the number of features is 2 the hyperplane is a line. whereas, if we have 3 input features, it will become a two-dimensional plane [26]. In other words, a hyperplane is a function that classifies the point in a higher dimension, thus in an ‘M’ dimensional space, the hyperplane equation can be given by (Equation IV.1)

$$
\begin{aligned}
y &= w_O + w_1x_1 + w_2x_2 + w_3x_3 \ldots \\
&= w_O + \sum_{i=1}^{m} \blacksquare\, w_ix_i \\
&= w_O + w^TX \\
&= b + w^TX
\end{aligned}
$$

Equation IV.1: Hyperplane function

Where :

Wi = vectors (W0, W1, W2, W3……Wm)

b = biased term (W0)

X = variables.

4.3.1.2 Kernel

The kernel is the function that helps solve problems by transforming the data, in order to go to higher dimensions and perform smooth calculations. we can cite three types of kernel [26]:

- **Linear kernel :** This kernel separates the classes with a straight line.
- **Polynomial kernel :** The polynomial kernel simply works by increasing the power of the kernel and calculating the dot product, which can define the relationship between two points. In the polynomial kernel function (Equation

IV.2) the degree of the polynomial (b) is an important parameter that helps define the best results.

$$\mathrm{K}(X_1,X_2) = (a + X_1^T X_2)^b$$

Equation IV.2: Polynomial kernel function

- **Radial kernel basis (RBF) :** RBF kernel function (Equation IV.3) value depends on the distance from the origin or from some point, where the closest observations have more influence than the further ones on how to classify the new observations

$$\mathrm{K}(X_1,X_2) = exponent(-\gamma||X_1 - X_2||^2)$$

Equation IV.3: Radial kernel basis function

4.3.1.3 Gamma

The gamma parameter controls the influence distance of a single training point. It is mainly used in the RBF kernel. A low value of gamma refers to a large similarity radius that results in more data grouped, whereas, for high gamma, points must be very close to each other to be grouped together [26].

4.3.1.4 Regularization

The Regularization parameter (also known as the C parameter in Python's Sklearn library) is used in the SVM optimization to measure how much we avoid misclassifying each training example. It adds a penalty for each misclassified data point. Small C means that the penalty for misclassified points is low, thus a decision boundary with a large margin is chosen at the expense of a greater number of misclassifications. While if C is large, SVM tries to minimize the number of misclassified observations due to elevated penalty which results in a decision boundary with a smaller margin. The penalty is directly proportional to the distance to the decision boundary [26].

4.3.2 Logistic regression

Regression is a method of modeling a target value based on independent predictors. This method is mostly used for forecasting and finding out the cause and effect relationship between variables. Regression techniques mostly differ based on the number of independent variables and the type of relationship between the independent and dependent variables [11].

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are email spam or not spam, online transactions fraud or not fraud, tumor malignant, or benign. Logistic regression transforms its output using the logistic sigmoid function (Figure IV.2) to return a probability value [11]. In our study, this algorithm is used as a classifier for a base-estimator and also as a meta-model.

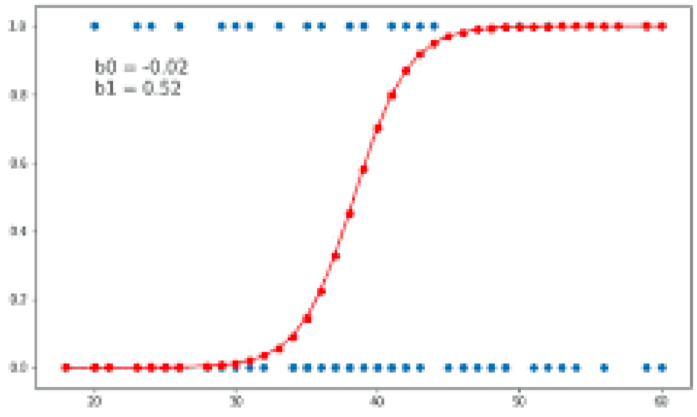


Figure IV.2 - Logistic function (Sigmoid-Function)

Logistic regression can be divided into three types depending on the outcome types which are :

- **binomial:** when a target variable can have only 2 possible types such as “0” or “1”.
- **multinomial:** target variable can have 3 or more possible types with no order such as “disease A” vs “disease B” vs “disease C”.
- **ordinal:** target variables can take multiple possible categories that are ordered, for example, the outcome can be “very low”, “low”, “high”, and “very high”. [11]

4.3.2.1 Sigmoid-Function

The Sigmoid-Function is an S-shaped (Equation IV.4) curve that can take a real-value number and maps it into a value in the range of 0 and 1, but never exactly at those limits.

$$\sigma(t) = \frac{e^{t}}{e^{t} + 1} = \frac{1}{1 + e^{-t}}$$

Equation IV.4: Sigmoid function

4.3.3 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm that helps solve both classification and regression problems. KNN is a non-parametric algorithm as it does not make any assumptions for underlying data distribution [17].

Following this algorithm, the training phase focuses on storing the data, while the learning process is done during the test phase. It is a distance-based algorithm.

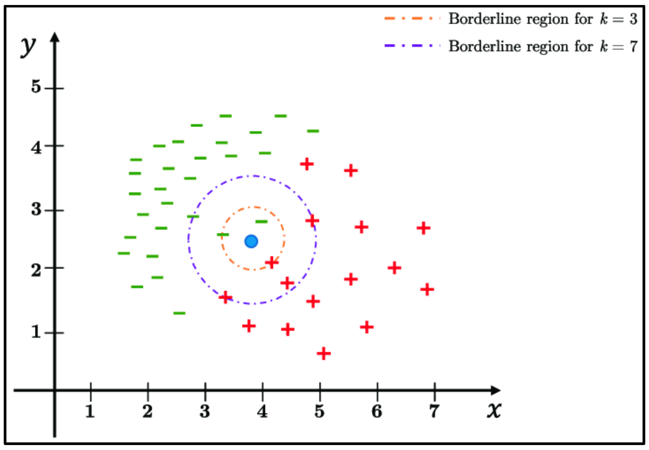


Figure IV.3: The KNN classification algorithm

4.3.3.1 Algorithm

In a binary problem, when we want to classify a new datapoint in category A or category B, we can use the KNN algorithm (Figure IV.3) that observes the behavior of the nearest points and classifies itself accordingly. In such a case, the behavior is which category it belongs to. The KNN algorithm principle can be explained by the following steps [8]:

a) Select the number K of the neighbors to be considered.

b) Calculate the distance between new data points and all the training points.

c) Sort the computed distance between training points and new data points in ascending order.

d) Choose the first K distances from the sorted list.

e) Take the mean of the classes associated with the distances.

f) Among these K neighbors, count the number of the data points in each category.

g) Assign the new data points to that category for which the number of the neighbor is maximum.

4.3.3.2 Distance Metrics

Several distance metrics (Figure IV.4) can be used, it's essential to choose the most appropriate one based on the studied dataset [16].

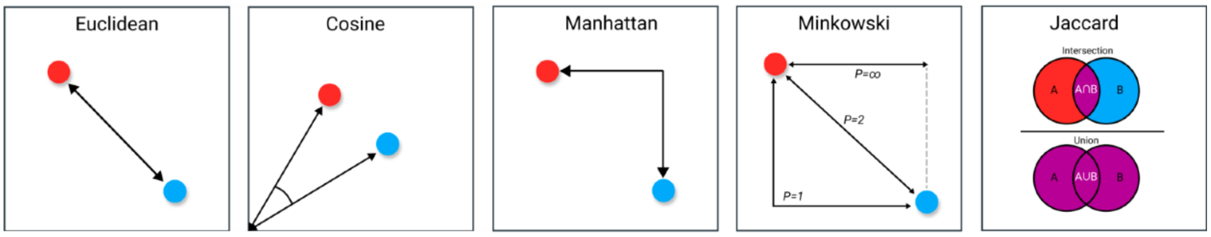


Figure IV.4: Distance Metrics

- **Minkowski Distance :** Is calculated where distances are in the form of vectors that have a length that cannot be negative.
- **Manhattan Distance :** Represents the distance between two points, calculated with the sum of the absolute differences of their Cartesian coordinates.
- **Euclidean Distance :** Represents the length of the straight line between two points in Euclidean space.
- **Cosine Distance :** It measures the direction by calculating the angle between two vectors using the cosine function.
- **Jaccard Distance :** It analyzes two data sets and tries to find the incident where both values are equal to 1.

4.3.3.3 The K value

A key process in the KNN algorithm is to find the best K value since a small k value can lead to overfitting, and a high k value can lead to underfitting. Likewise choosing an even number in a binary classification can fall in a tie between the two classes [16].

To find the best K, we can use Plot the elbow curve between different K values and select the K value when there is a sudden drop in the error rate (Figure IV.5).

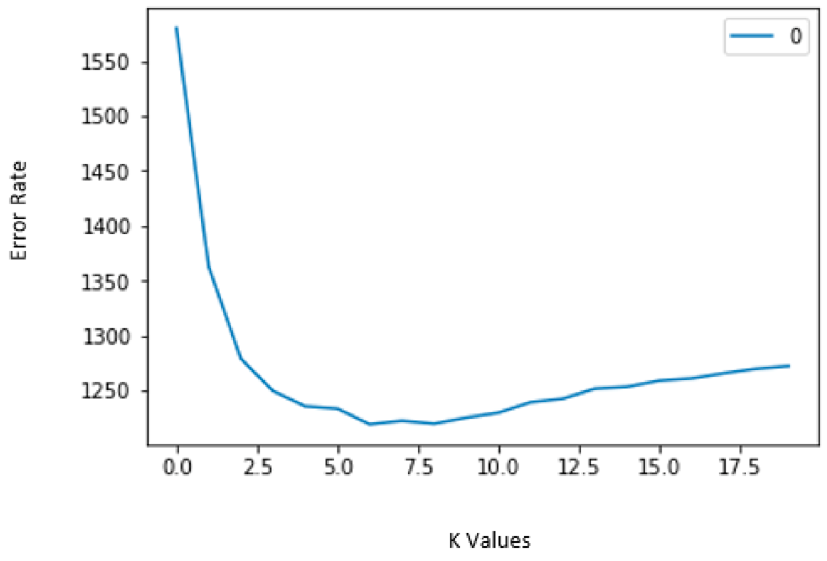


Figure IV.5: Plot the elbow curve

4.3.4 Decision tree

The decision tree Algorithm (Figure IV.6) belongs to the family of supervised machine learning algorithms. It can be used for both classification problems and regression problems.

The goal of this algorithm is to create a model that predicts the value of a target variable, for which the decision tree uses the tree representation to solve the problem in which the leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning.

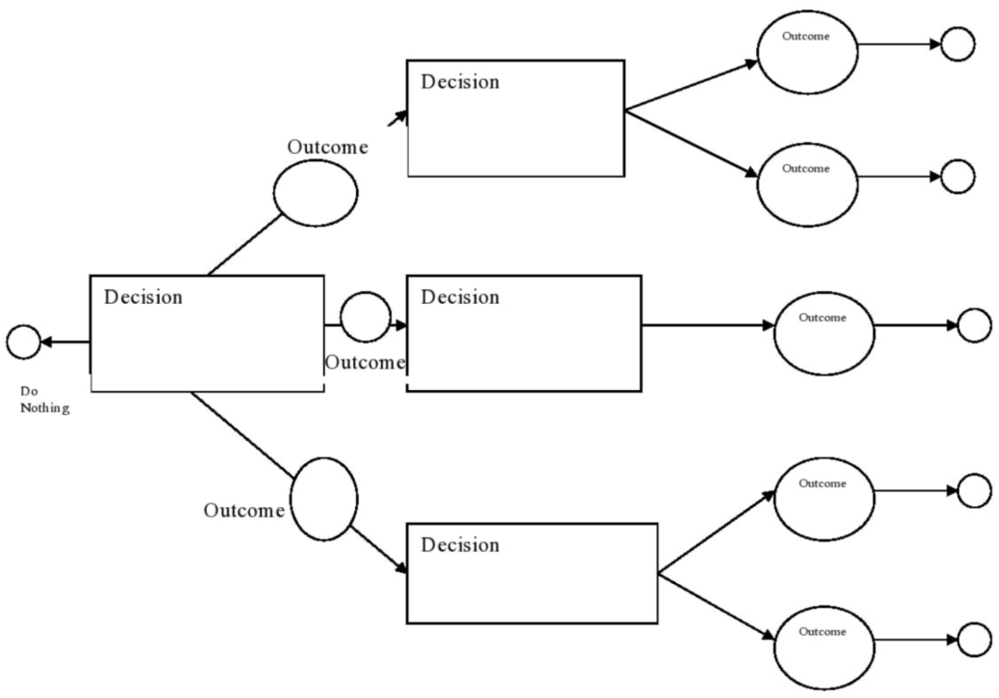


Figure IV.6: Decision Tree Algorithm

4.3.4.1 Steps

a) Get a list of rows (dataset) which are taken into consideration for making a decision tree recursively at each node.

b) Calculate uncertainty of our dataset or Gini impurity or how much our data is mixed up, etc.

c) Generate a list of all questions which need to be asked at that node.

d) Partition rows into True rows and False rows based on each question asked.

e) Calculate information gain based on gini impurity and partition the data from the previous step.

f) Update highest information gain based on each question asked.

g) Update best question based on information gain (higher information gain).

h) Divide the nodes on the best question. Repeat again from step 1 again until we get pure nodes (leaf nodes).

4.3.4.2 Criterion

It is the function that measures the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

Impurity is a measure of the homogeneity of the labels on a node. There are many ways to implement the impurity measure, two of which scikit-learn has implemented is the Information gain and Gini Impurity or Gini Index.

4.3.4.3 Splitter

It is the strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose a random split.

4.3.5 Gaussian Naive Bayes

Gaussian Naive Bayes is a variant of Naïve Bayes classifiers which are probabilistic machine learning algorithms used mostly in classification tasks based on the Bayes’ Theorem [9].

4.3.5.1 Bayes Theorem

Bayes’ Theorem is a mathematical formula (Equation IV.5) that calculates conditional probabilities, which is a measure of the probability of an event occurring given that another event has (by assumption, presumption, assertion, or evidence) occurred [9].

$$\mathrm{P(A|B)} = \frac{P(B|A)\,.P(A)}{P(B)}$$

Equation IV.5:Posterior probability

- P(A|B): represents how often A happens given that B happens, also called posterior probability.
- P(B|A): represents how often B happens given that A happens
- P(A): is how likely A is on its own
- P(B): how likely B is on its own.

4.3.5.2 Naïve Bayes Classifier

Naive Bayes is a probabilistic machine learning algorithm used in several classification tasks, such as classification of documents, filtering spam, prediction, and so on. This algorithm incorporates in its model features that are independent of each other, which means that any modification in the values of some features does not impact the other features [9].

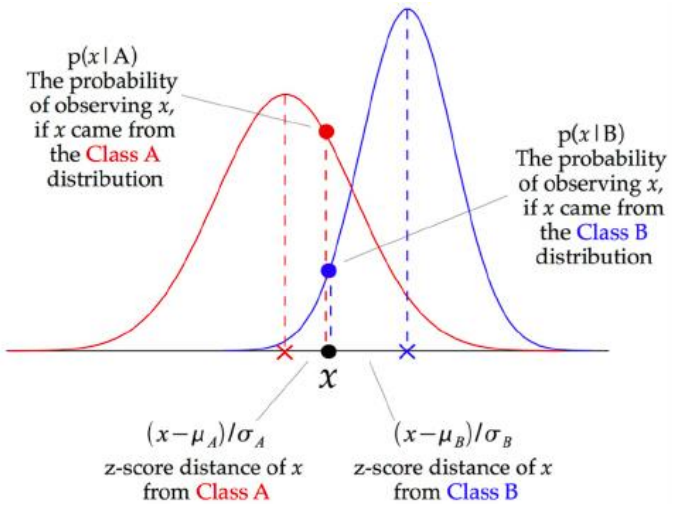

p(x | A)
The probability of observing x, if x came from the Class A distribution
p(x | B)
The probability of observing x, if x came from the Class B distribution
x
$(x-\mu_A)/\sigma_A$
z-score distance of x from Class A
$(x-\mu_B)/\sigma_B$
z-score distance of x from Class B

Figure IV.7: Gaussian Naive Bayes classifier

4.3.5.3 Gaussian Naïve Bayes Classifier

The Gaussian Naïve Bayes classifier (Figure IV.7) supports continuous-valued models and features each conforming to a Gaussian (normal) distribution. In another way, when the predictors take up a continuous value, we assume that these values are sampled from a gaussian distribution [27][19].

$$P(x_i \ |y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(\frac{(x-\mu y \ \ )}{2\sigma_y^2}\right)$$

Equation IV.6: Likelihood equation

This distribution gives a bell-shaped curve when plotted (Figure IV.7) that is symmetric to the mean of the feature values. In a classification process, in order to determine the class result, we first calculate the likelihood of each class using Equation IV.6 (σ and μ are the variance and mean of the continuous). After calculating each likelihood, the class with the higher likelihood is considered the resulting class [18][19].

4.4 Conclusion

In this chapter, we discussed each algorithm, from the concept to its parameters. The next step would be implementing them and fine-tuning the parameters to get the best models out of these algorithms.

Chapter 5

Heterogeneous Parallel Ensemble Learning : Implementation

5.1 Introduction

After describing the full process, the algorithms and the methods used in the design of our system, it is time to implement them and bring them into fruition. For that, we will be using a variety of tools and technologies to implement the steps described in the previous chapter. One the implementation is finished, experimentations will be held to test the effectiveness of the proposed system.

5.2 Problem at hand and dataset

Machine learning systems are designed to treat different problems, from medical to real estate ones. In this study, we will consider a medical problem, to detect whether a patient will present a heart disease or not.

For that, we used a relatively small dataset (270 records) related to our problem, with the features presented in the table V.1 splitted into a training and a validation set according to a 80% 20% ratio. The dataset concerns a special case of heart disease: Coronary Artery Disease (CAD).

CAD symptoms may differ from person to person. However, because many people have no symptoms, they do not know they have the CAD until they have chest pain, a heart attack, or sudden cardiac arrest. This led to the construction of heart disease datasets from previous patients' records. Most of CAD datasets are provided by the University of California Irvine (UCI) machine learning repository [36]. CAD prediction models can be trained on available datasets and used to diagnose the presence of this disease for new patients.

Attr.	Name	Type	Description
C_1	Age	Continuous	Age in years
C_2	Gender	Discrete	0 = female 1 = male
C_3	Cp	Discrete	Chest pain type:1=typical angina, 2=atypical angina, 3=non-anginal pain, 4=asymptomatic
C_4	Trestbps	Continuous	Resting blood pressure (in mm Hg)
C_5	Chol	Continuous	Serum cholesterol in md/dl
C_6	Fbs	Discrete	Fasting blood sugar > 120: 0=False, 1=True
C_7	Restecg	Discrete	Resting electrocardiographic results: 0=normal, 1=having ST-T wave abnormality 2=showing probable or define left ventricular hypertrophy by Estes'crietria
C_8	Thalach	Continuous	Maximum heart rate achieved
C_9	Exang	Discrete	Exercise induced angina: 0=yes, 1=no
C_{10}	Oldpeak	Continuous	ST depression induced by exercise relative to rest
C_{11}	Slope	Discrete	The slope of the peak exercise ST segment: 1=up sloping, 2=flat, 3=down sloping
C_{12}	Ca	Discrete	Number of major vessels (0-3) colored by fluoroscopy
C_{13}	Thal	Discrete	3=normal, 6=fixed defect, 7=reversible defect
C_{14}	Target class	Discrete	Diagnosis class: 1=absence of CAD, 2=presence of CAD

Table V.1: The Statlog heart disease dataset description [36]

5.3 Tools used

For the implementation of the project, and as mentioned earlier, we needed a set of tools to make it happen (Figure V.1). We describe these tools in this section.

Figure V.1: Tools used to build the proposed system.

5.3.1 Python

Python is an object oriented programming language considered as the most powerful language people can use these days. It can be used as a scripting language as well as a tool for data scientists to achieve their goals. One of its key points is the fact that it is an interpreted language, and since it has a highly readable syntax. It is commonly known to be a very simple language compared to others.

Also, Python offers many libraries which are open source and installed by a simple command line (pip install). It is also worth mentioning that many leading companies are using it as their main programming language [38].

Python is considered very versatile language, it is used in almost all fields [38], including :

- Data science
- Scientific and mathematical computing
- Web development
- Finance and trading
- System automation and administration
- Computer graphics
- Basic game development
- Security and penetration testing
- General and application-specific scripting
- Mapping and geography (GIS Software)

5.3.2 Github

Github is a hosting platform for software development and version control using Git. It allows programmers across the globe to share code, search for it, download it and even like it.

Through Github, developers can upload their projects as repositories, and then invite other people to collaborate with them, review their code and validate it.

Github also provides the ability to execute certain actions automatically after pushing to the remote repository [39].

To organize one's project, Github offers the ability to push under different branches constructing a tree, where a developer can develop a new feature in an isolated branch without messing with the stable version, also creating commits that would serve as points to locate the current state and old ones [40].

5.3.3 Jupyter notebook

Jupyter notebook is an interactive web-based open source computational environment for creating notebook books/documents from computational material. A notebook can make reference to many different entities, essentially the web application, the Python web server or the document depending on context. These documents contain code, equations markdown and the result of its execution.

This tool was used to hold the code of our algorithms and the result of the training and the predictions on our dataset.

With the rise of machine and deep learning, Jupyter notebooks became more popular than ever. It is now used by many top companies in their environment for this domain. An example of this is Google's project, known as Google Colaboratory.

5.3.4 Google colab

Colaboratory, also known as Colab, is a free Jupyter notebook environment provided by Google that runs on the cloud and uses google drive as a storage for its notebooks. It started initially as an internal project.

We used colab as our main environnement for the development of our system, and that is because google offers highly performing virtual machines that would allow us to get results in less execution time compared to our local machines (A 2017 model macbook pro).

5.4 Libraries used

To better implement the described algorithms, Python, thanks to its libraries, offers an out of the box solution that implements them and helps the developer save time and focus on implementing concrete steps of his system. Below, we are going to describe these libraries.

5.4.1 Pandas

Before starting the machine learning part, it is necessary to clean and manipulate the provided dataset. That is why Pandas is now considered as one of the most important python tools used by Data scientists.

Pandas, short for Python Data Analysis Library, is an open source Python library that lets users explore, manipulate and visualize data easily.

Pandas is popular among data scientists for many reasons, most importantly the fact that it is easy to learn, extremely fast and can take in a huge variety of data (csv, excel, sql, …etc).

5.4.2 Numpy

Machine learning implies the need to work with numerical data, multidimensional arrays and matrix data structures, that is what motivated the use of Numpy.

Numpy, short for Numerical Python, is an open-source python library that is used in almost every field of science and engineering. Its API is used in other popular libraries such as Pandas, Scipy, …etc.

In this project, it was used to create homogeneous n-dimensional arrays as required in our algorithms and then efficiently perform computations on them.

5.4.3 Matplotlib

Matplotlib is a 2D plotting library that is built on Numpy arrays and designed to work with the Scipy stack.

In this project, and like many other machine learning ones, this library is used to visualize the predicted values and the actual values on the same plane.

It is also worth mentioning that it is also used to present data in the form of graphs or charts to understand complex data and identify trends and patterns easily within it, which would eventually help decision makers take their decision.

5.4.4 Sklearn

Scikit-learn (also known as Sklearn) is a Python library used to build machine learning models, in addition to, data preparation and analysis, and evaluation.

The Sklearn library provides methods for data reading, preparation, and unsupervised clustering, aside from many unsupervised and supervised learning algorithms, and much more [23].

5.4.5 Scipy

SciPy (stands for Scientific Python) is a computation Python library used to solve many mathematical equations and algorithms, in addition to data processing, such as numerical integration, interpolation, optimization, linear algebra, and statistics.

This library uses Numpy under the hood, in addition to added and optimized functions that are frequently used in Data science. Scipy provides extensions for scientific mathematical formulae such as Matrix Rank, Inverse, polynomial equations, and LU Decomposition.

In our project, we used this library to compute the correlation using a Pearson correlation coefficient.

5.5 The implemented architecture

During this phase, we tried to implement the proposed architecture in chapter 4. But due to time constraints, we only implemented it partially.

Our system is construced using models that relies on the algorithms presented in chapter 4. When it comes to the combining methods, we only used 3 of the 4 combining equations presented (majority vote, accuracy weighing and entropy weighting). On the other hand, we also propose stacking ensemble learners where a meta-model is applied on the individual learners' results to get the final result.

The following (FigureV.2 and FigureV.3) illustrate the overall architecture of the proposed solution. In figure V.2, the results of base learners (Y1…Yn) are combined using a specific function and in figure V.3, a stacking concept is applied through the use of logistic regression as a meta-model.

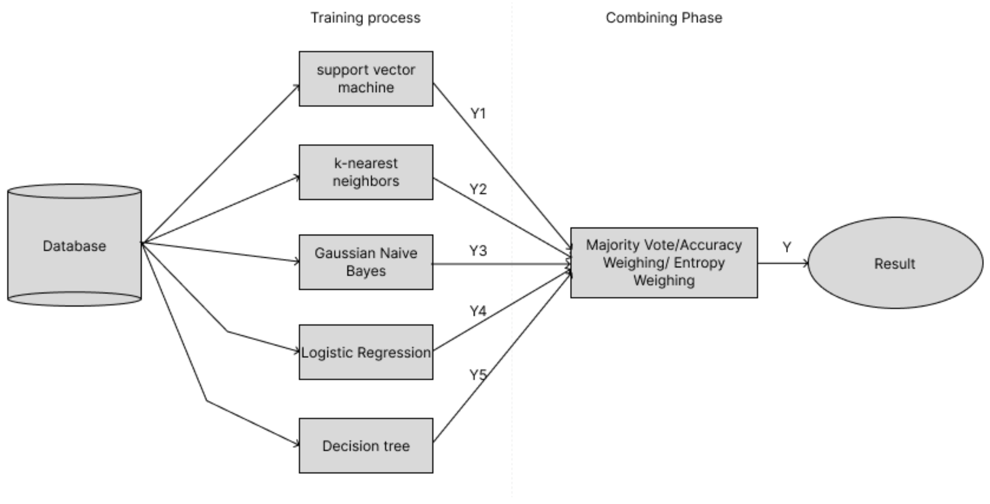


Figure V.2: Heterogeneous ensemble learning using functions.

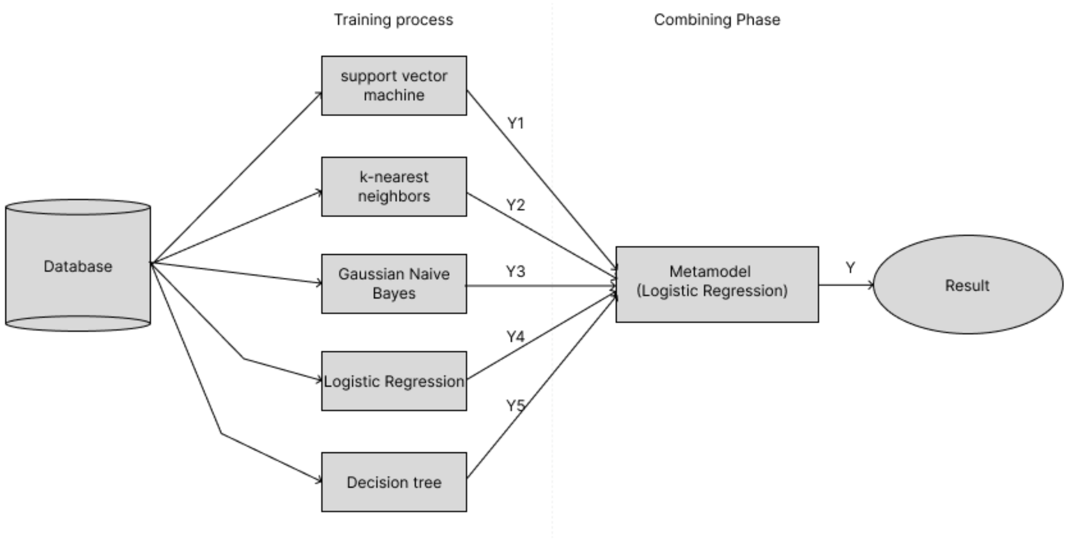


Figure V.3: Heterogeneous ensemble learning using logistic regression as meta-model.

5.6 Experimentations And results

In the following, we will discuss the objective of our experimentation, then present the results obtained by each implemented algorithm, how we fine tuned[2] each one of them to get better results, and then use them all together to build the heterogeneous parallel ensemble learner.

5.6.1 Objective of the experimentation

These experimentations aim to build our system by following the presented architecture as well as the process and concepts defined in the earlier chapters (3 & 4).

Through it, we were able to build the most accurate system by picking the most accurate base estimators and combinations of models possible.

5.6.2 Steps of this experimentation

This experimentation follows the process described in the previous chapter, but instead of building the algorithm from scratch, and as mentioned earlier, we are using the Sklearn module to build our system. It provides a function for each of the implemented base algorithms. We will then apply both learning and validation in order to get and compare the accuracy of each model using different parameters.

[2] https://www.merriam-webster.com/dictionary/fine-tune

After getting the most accurate base estimators, we will combine the results as described in chapters and 4.

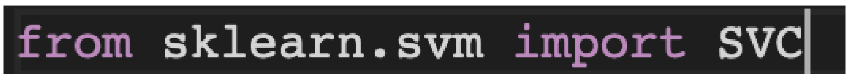
5.6.3 Experimenting on the base estimators

For each of the discussed algorithms, we will show how we imported it, used it, then present a table that contains the parameters and the precision of the most accurate models we got in addition to the default configuration.

To fine tune each base model and get the best configuration possible, we used a “GridSearch”. GridSearch is a method that takes the possible values for attributes as an input and tries all possible combinations in order to test the model’s accuracy and then indicate the best combination possible.

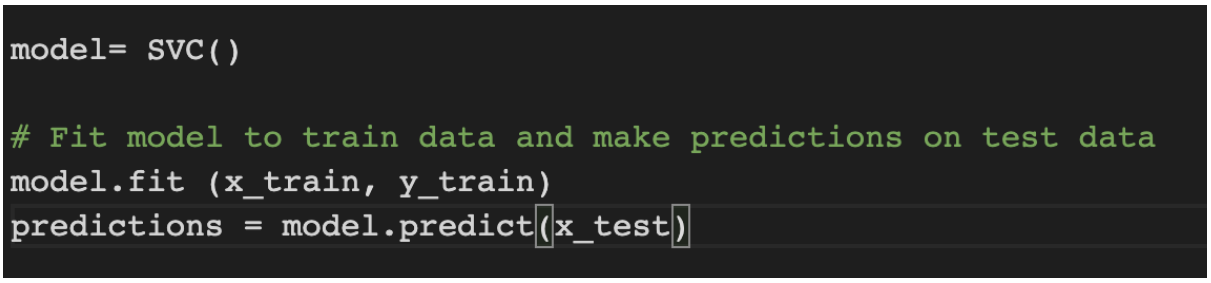
5.6.3.1 Support vector machine

Sklearn provides an SVC (Support Vector Classification) function that can be imported and then used as shown in figures V.4 and V.5.



```
from sklearn.svm import SVC
```

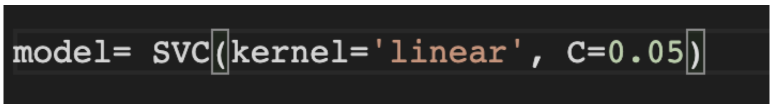
Figure V.4: SVC import



```
model= SVC()

# Fit model to train data and make predictions on test data
model.fit (x_train, y_train)
predictions = model.predict(x_test)
```

Figure V.5: Initiating the model, fitting and then using it for prediction.

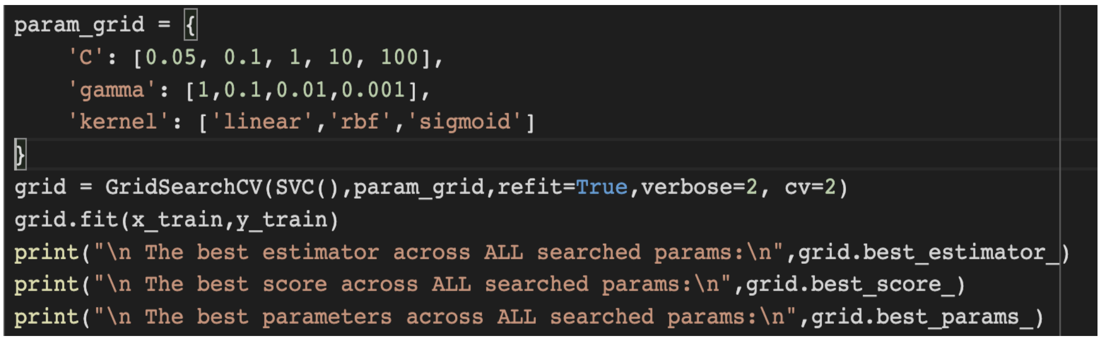
Figure V.5 uses SVM with default values for each of its parameters, but it is possible to change some parameters as it can be seen in figure V.6.



```
model= SVC(kernel='linear', C=0.05)
```

Figure V.6: SVC initiated with parameters.

The provided possibilities for our model can be seen in figure V.7:



```
param_grid = {
    'C': [0.05, 0.1, 1, 10, 100],
    'gamma': [1,0.1,0.01,0.001],
    'kernel': ['linear','rbf','sigmoid']
}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2, cv=2)
grid.fit(x_train,y_train)
print("\n The best estimator across ALL searched params:\n",grid.best_estimator_)
print("\n The best score across ALL searched params:\n",grid.best_score_)
print("\n The best parameters across ALL searched params:\n",grid.best_params_)
```

Figure V.7: Fine tuning SVC using GridSearch

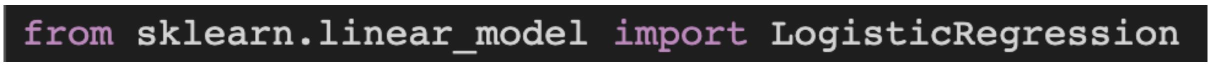
Parameters	Accuracy
`default`	0,69
`C=0.05, kernel='linear'`	0,88
`C=10, kernel='linear'`	0.88
`C=100, kernel='poly'`	0.85

Table V.2 The results obtained by fine tuning the SVC classifier

The results are shown in table V.2. The first row concerns the accuracy of SVM by using default parameters while the other rows show the accuracy when changing some parameters. As it can be noticed, by tuning some parameters, we can improve the accuracy of the SVC classifier.

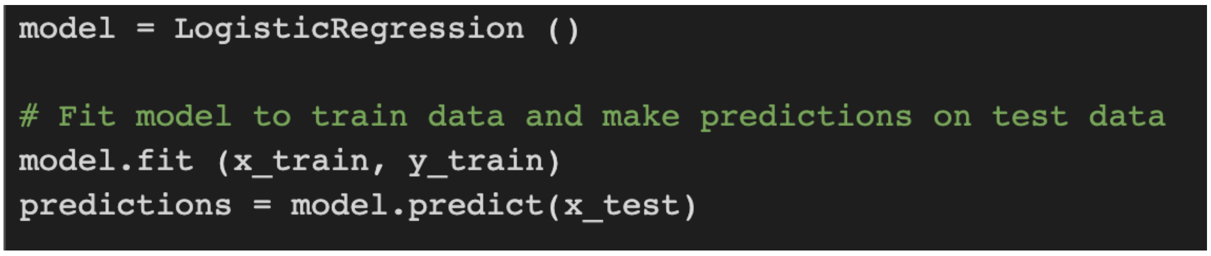
5.6.3.2 Logistic regression

Sklearn also provides a function for the logistic regression function that can be imported and then used as shown in figures V.8 and V.9.



```
from sklearn.linear_model import LogisticRegression
```

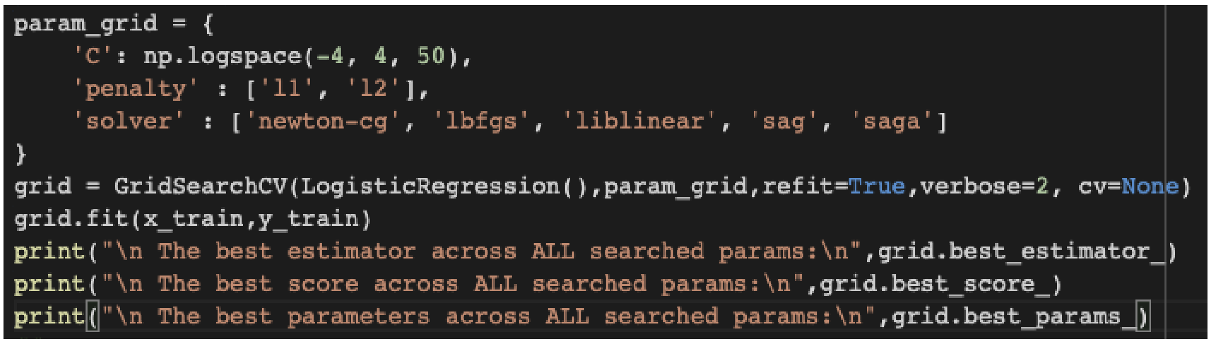
Figure V.8: Import of logistic regression



```
model = LogisticRegression ()

# Fit model to train data and make predictions on test data
model.fit (x_train, y_train)
predictions = model.predict(x_test)
```

Figure V.9: Initiating the model, fitting and then using it for prediction.

Figure V.9 uses Logistic regression with default values for each of its parameters, but as seen in the previous section, we can explicitly change their values. The figureV.10 shows some of the possible values.



```
param_grid = {
    'C': np.logspace(-4, 4, 50),
    'penalty' : ['l1', 'l2'],
    'solver' : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}
grid = GridSearchCV(LogisticRegression(),param_grid,refit=True,verbose=2, cv=None)
grid.fit(x_train,y_train)
print("\n The best estimator across ALL searched params:\n",grid.best_estimator_)
print("\n The best score across ALL searched params:\n",grid.best_score_)
print("\n The best parameters across ALL searched params:\n",grid.best_params_)
```

Figure V.10: Fine tuning logistic using GridSearch

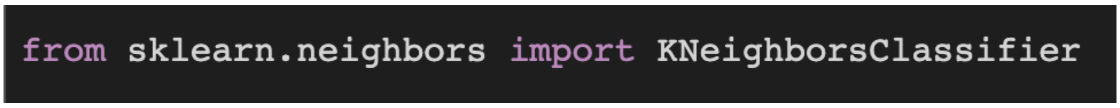
The results are shown in table V.3.

Parameters	Accuracy
`default`	0,89
`C=1000`	0,89
`C=11.513953993264458, penalty='l1', solver='liblinear'`	0.89
`C=16.768329368110066, solver='liblinear'`	0.89

Table V.3 The results obtained by fine tuning the Logistic regression classifier

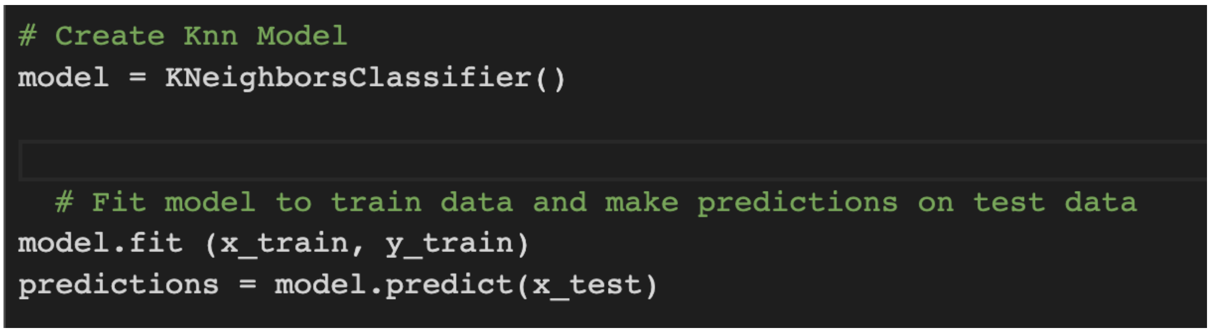
5.6.3.3 K-nearest neighbors

Sklearn also provides a function for the KNN algorithm that can be imported and then used as shown in figures V.11 and V.12.



```
from sklearn.neighbors import KNeighborsClassifier
```

Figure V.11: Importing KNN

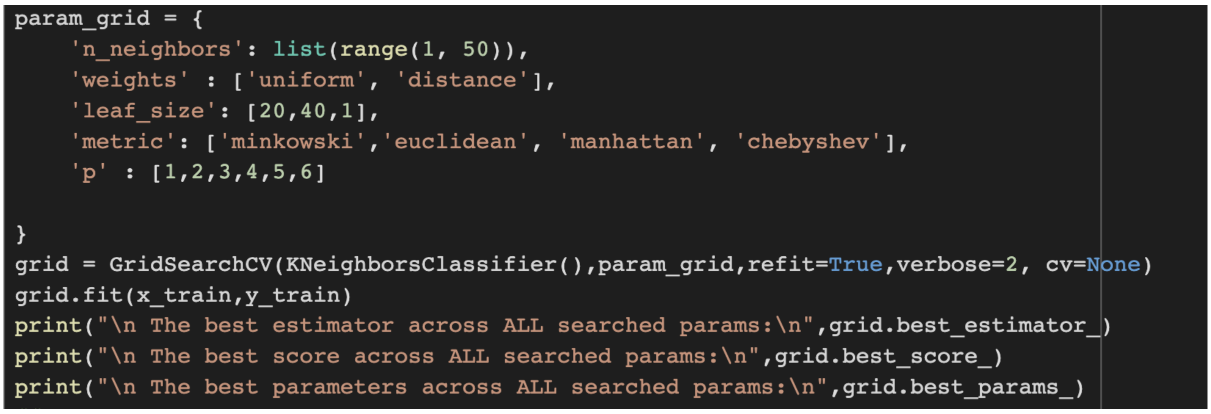


```
# Create Knn Model
model = KNeighborsClassifier()


  # Fit model to train data and make predictions on test data
model.fit (x_train, y_train)
predictions = model.predict(x_test)
```

Figure V.12: Initiating, fitting and then using knn to predict.

Figure V.12 uses KNN with default values for each of its parameters, but as seen in the previous sections, we can explicitly change their values.

Figure V.13 shows the possible values for some of the model's attributes.



```
param_grid = {
    'n_neighbors': list(range(1, 50)),
    'weights' : ['uniform', 'distance'],
    'leaf_size': [20,40,1],
    'metric': ['minkowski','euclidean', 'manhattan', 'chebyshev'],
    'p' : [1,2,3,4,5,6]

}
grid = GridSearchCV(KNeighborsClassifier(),param_grid,refit=True,verbose=2, cv=None)
grid.fit(x_train,y_train)
print("\n The best estimator across ALL searched params:\n",grid.best_estimator_)
print("\n The best score across ALL searched params:\n",grid.best_score_)
print("\n The best parameters across ALL searched params:\n",grid.best_params_)
```

Figure V.13: Fine tuning KNN

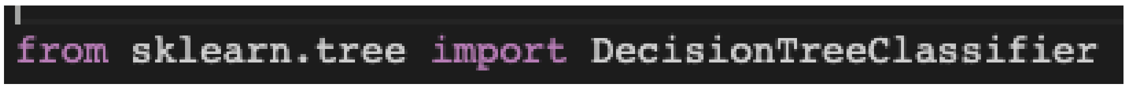
The results are shown in table V.4.

Parameters	Accuracy
`default`	0,70
`n_neighbors=29, weights="uniform",algorithm="auto",metric="minkowski",p=1, leaf_size=20`	0,83
`leaf_size=20, n_neighbors=9, p=1`	0,76

Table V.4 The results obtained by fine tuning the k-nearest neighbors classifier

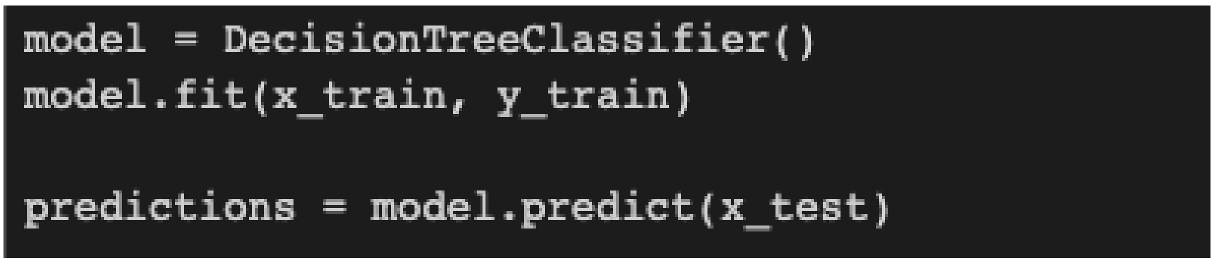
5.6.3.4 Decision tree

Sklearn also provides a function for the decision tree algorithm that can be imported and then used as shown in figures V.14 and V.15.



```python
from sklearn.tree import DecisionTreeClassifier
```

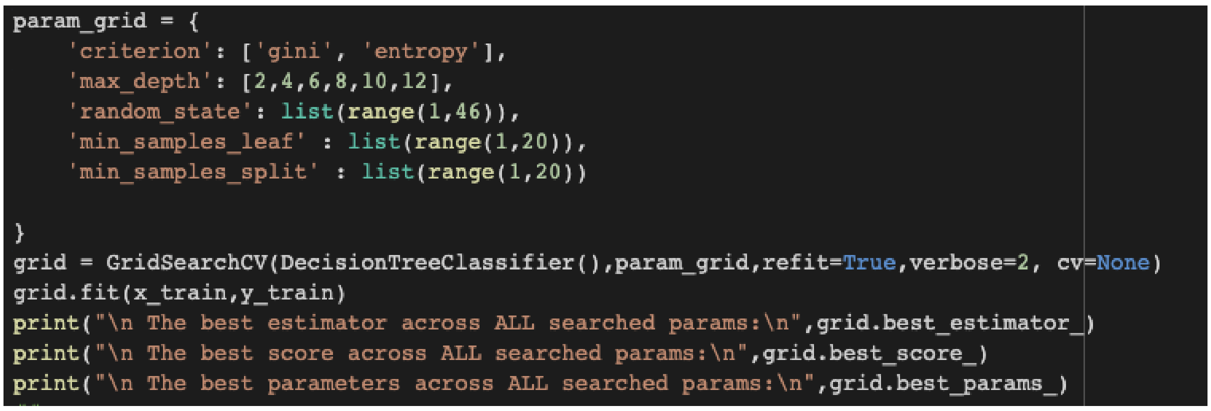
Figure V.14: Importing the Decision Tree



```python
model = DecisionTreeClassifier()
model.fit(x_train, y_train)

predictions = model.predict(x_test)
```

Figure V.15: Initiating, fitting and then using a Decision tree to predict.

Figure V.15 uses the Decision tree with default values for each of its parameters, but as seen in the previous sections, we can explicitly change their values. Some of these values are shown in the figure V.16.



```python
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2,4,6,8,10,12],
    'random_state': list(range(1,46)),
    'min_samples_leaf' : list(range(1,20)),
    'min_samples_split' : list(range(1,20))

}
grid = GridSearchCV(DecisionTreeClassifier(),param_grid,refit=True,verbose=2, cv=None)
grid.fit(x_train,y_train)
print("\n The best estimator across ALL searched params:\n",grid.best_estimator_)
print("\n The best score across ALL searched params:\n",grid.best_score_)
print("\n The best parameters across ALL searched params:\n",grid.best_params_)
```

Figure V.16: Fine tuning Decision tree using GridSearch

The results are shown in table V.5.

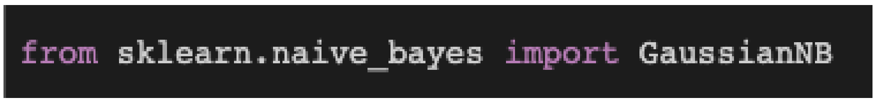
Parameters	Accuracy
`default`	0,81

`criterion='entropy', max_depth=3, min_samples_leaf=3, min_samples_split=5, random_state=42`	0,87
`max_depth=4,min_samples_leaf=9, random_state=1`	0,87

Table V.5 The results obtained by fine tuning the Decision tree classifier

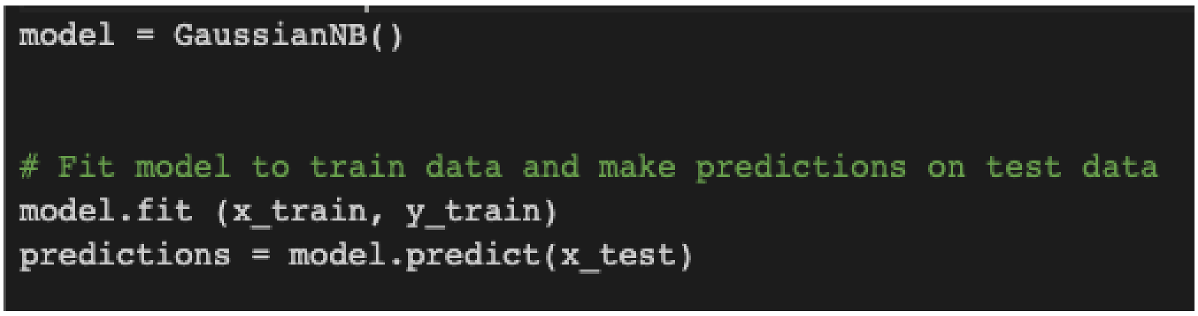
5.6.3.5 Gaussian Naive Bayes

Sklearn also provides a function for gaussianNB (Gaussian naive bayes) that can be imported and then used as shown in figures V.17 and V.18



```
from sklearn.naive_bayes import GaussianNB
```

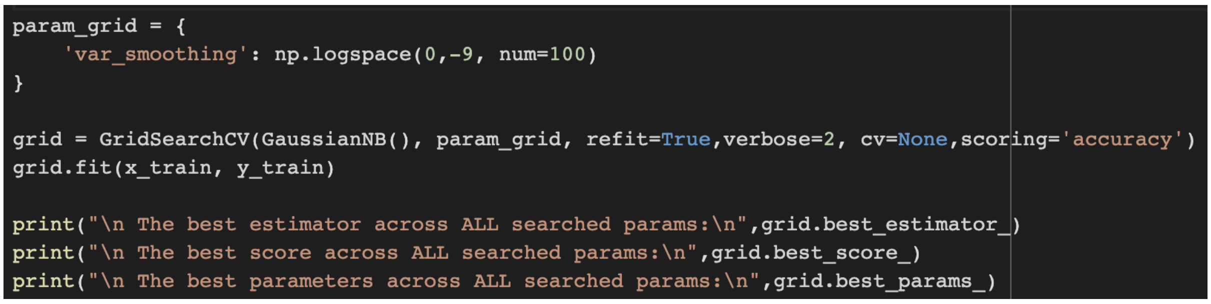
Figure V.17: Importing Gaussian Naive Bayes



```
model = GaussianNB()


# Fit model to train data and make predictions on test data
model.fit (x_train, y_train)
predictions = model.predict(x_test)
```

Figure V.18: Initiating Gaussian naive bayes, training it then using it to predict.

The provided possibilities for Gaussian Naive Bayes can be seen in figure V.19.



```
param_grid = {
    'var_smoothing': np.logspace(0,-9, num=100)
}

grid = GridSearchCV(GaussianNB(), param_grid, refit=True,verbose=2, cv=None,scoring='accuracy')
grid.fit(x_train, y_train)

print("\n The best estimator across ALL searched params:\n",grid.best_estimator_)
print("\n The best score across ALL searched params:\n",grid.best_score_)
print("\n The best parameters across ALL searched params:\n",grid.best_params_)
```

Figure V.19: Fine tuning GaussianNB using GridSearch

The results are shown in table V.6.

Parameters	Accuracy
`default`	0,88
`GaussianNB(var_smoothing=4.328761281083062e-05)`	0.89

Table V.6 The results obtained by fine tuning the GaussianNB classifier

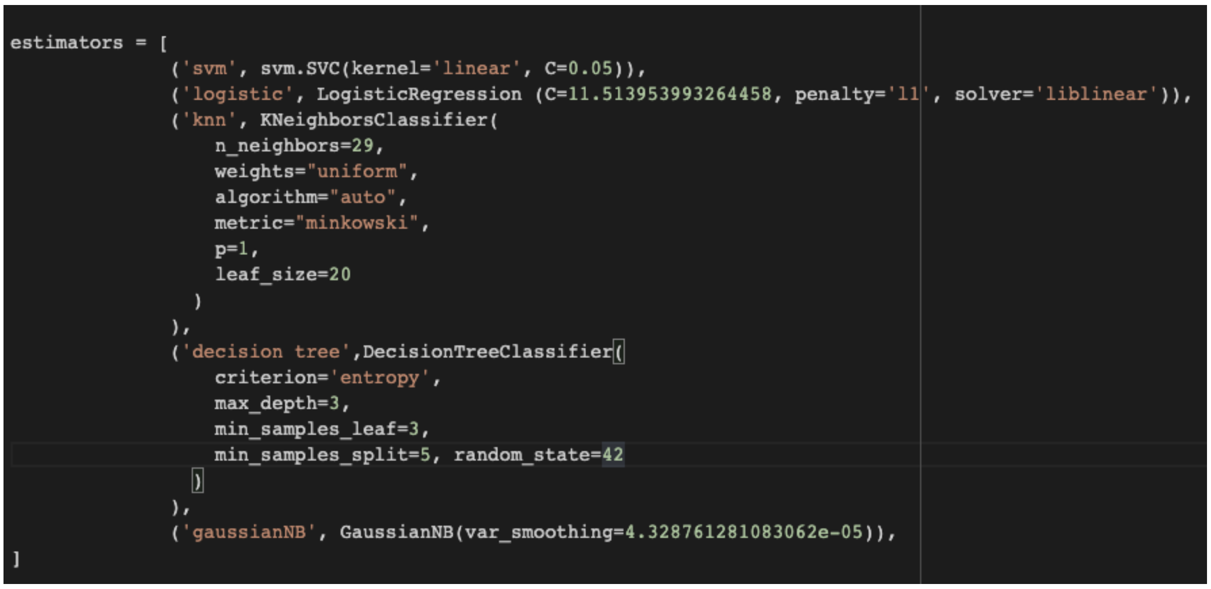
5.6.4 Experimenting on meta-model and combinators

In this section, we will discuss the technical implementation of each of the combining methods, including the meta-model technique (stacking).

The implementation of these techniques will follow what was stated back in the previous chapters. First we use the models we experimented on in the previous section and then run the learning and prediction phases as seen in the implemented architecture.

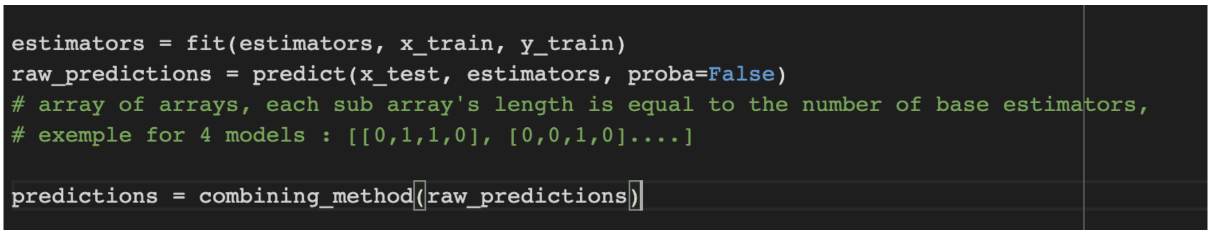
In the following sections, we will present and explain the code used to bring the methods to life, then present the result of its execution.

All the following methods share the same starting point, they all invoke the models the same way by storing them in an array of tuples, as can be seen in Figure V.20 and then execute the 2 phases mainly the same way (it will be different when using stacking) Figure V.21.



```
estimators = [
              ('svm', svm.SVC(kernel='linear', C=0.05)),
              ('logistic', LogisticRegression (C=11.513953993264458, penalty='l1', solver='liblinear')),
              ('knn', KNeighborsClassifier(
                  n_neighbors=29,
                  weights="uniform",
                  algorithm="auto",
                  metric="minkowski",
                  p=1,
                  leaf_size=20
                )
              ),
              ('decision tree',DecisionTreeClassifier(
                  criterion='entropy',
                  max_depth=3,
                  min_samples_leaf=3,
                  min_samples_split=5, random_state=42
                )
              ),
              ('gaussianNB', GaussianNB(var_smoothing=4.328761281083062e-05)),
]
```

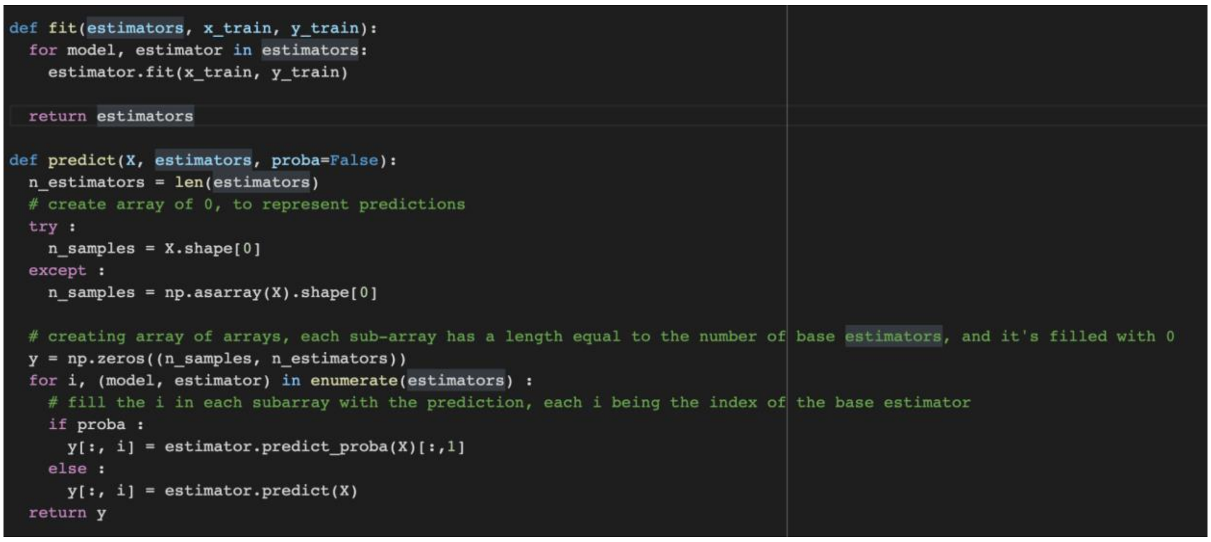
Figure V.20: Storing the base estimators



```
estimators = fit(estimators, x_train, y_train)
raw_predictions = predict(x_test, estimators, proba=False)
# array of arrays, each sub array's length is equal to the number of base estimators,
# exemple for 4 models : [[0,1,1,0], [0,0,1,0]....]

predictions = combining_method(raw_predictions)
```

Figure V.21: training, estimating and combining the results

The training phase is just a fitting of each of the models that are in the array of estimators (Figure V.22). The predicting phase will create an array of size X containing arrays of size Y, X is the number of instances, and Y is the number of estimators, each nested array containing the estimation for a given estimator.

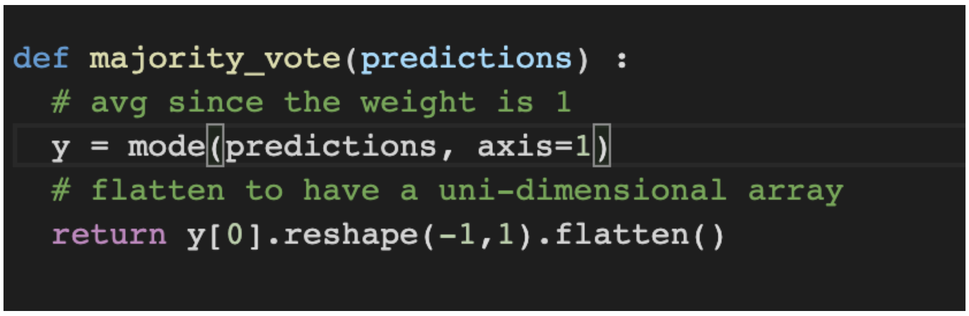


```
def fit(estimators, x_train, y_train):
  for model, estimator in estimators:
    estimator.fit(x_train, y_train)

  return estimators

def predict(X, estimators, proba=False):
  n_estimators = len(estimators)
  # create array of 0, to represent predictions
  try :
    n_samples = X.shape[0]
  except :
    n_samples = np.asarray(X).shape[0]

  # creating array of arrays, each sub-array has a length equal to the number of base estimators, and it's filled with 0
  y = np.zeros((n_samples, n_estimators))
  for i, (model, estimator) in enumerate(estimators) :
    # fill the i in each subarray with the prediction, each i being the index of the base estimator
    if proba :
      y[:, i] = estimator.predict_proba(X)[:,1]
    else :
      y[:, i] = estimator.predict(X)
  return y
```

Figure V.22: Fit and predict functions

5.6.4.1 Majority vote

The implementation of this method is the easiest one. We compute the average using the function mode (supplied by numpy), then reduce the array of arrays to an array of estimations. Figure V.23 shows this in action.



```
def majority_vote(predictions) :
  # avg since the weight is 1
  y = mode(predictions, axis=1)
  # flatten to have a uni-dimensional array
  return y[0].reshape(-1,1).flatten()
```

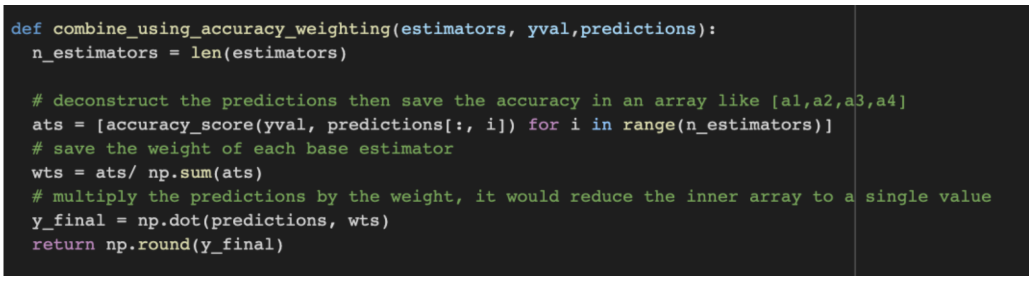
Figure V.23: Majority vote combinator

Using the estimators shown in Figure V.20, we got an accuracy of **89%**.

5.6.4.2 Accuracy weighing

The goal here is the same as in the previous method, which is flatting the array to a uni-dimensional one while using the equation (Equation III.1) presented in chapter 3.

We first start by saving the accuracy of each model in an array, compute the weight of it, then use this weight to compute the final outcome. Figure V.24 brings this algorithm to life.



```python
def combine_using_accuracy_weighting(estimators, yval,predictions):
  n_estimators = len(estimators)

  # deconstruct the predictions then save the accuracy in an array like [a1,a2,a3,a4]
  ats = [accuracy_score(yval, predictions[:, i]) for i in range(n_estimators)]
  # save the weight of each base estimator
  wts = ats/ np.sum(ats)
  # multiply the predictions by the weight, it would reduce the inner array to a single value
  y_final = np.dot(predictions, wts)
  return np.round(y_final)
```

Figure V.24: Accuracy weighing

The accuracy obtained was equal to **89%**.

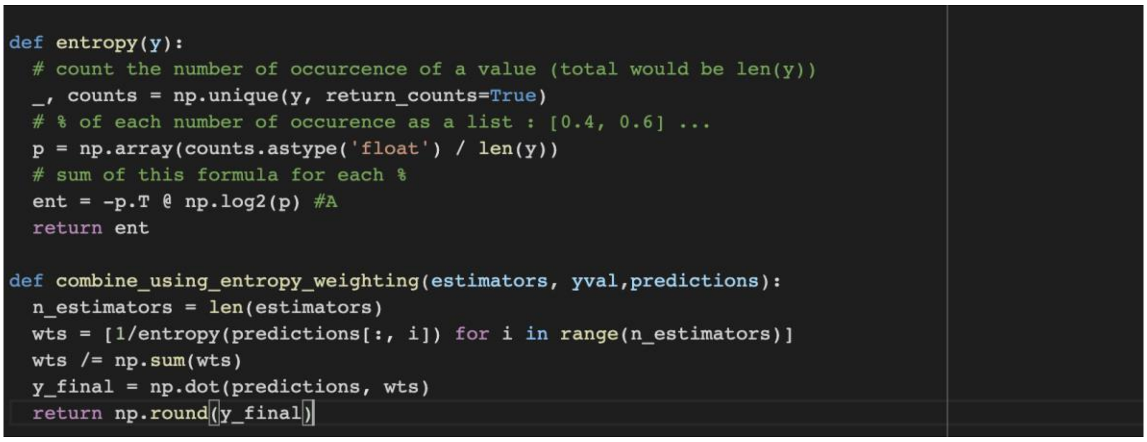
5.6.4.3 Entropy weighing

In the following, we will try to implement the equations (Equation III.2 and Equation III.3) presented in chapter 3, while being coherent to what was presented in the architecture and done in the previous methods.

As can be seen in Figure V.25, we started by implementing the entropy equation, counting the number of occurrences of a value and turning it into a percentage. Let us say the class we are counting for is 1, overall, we had 10 occurrences of it, we might have an array that resembles the following: [1,0,4,2,0,1,2]. We then turn them into percentages, giving the following: [0.1, 0, 0.4, 0.2, 0, 0.1, 0.2]. Then we compute the entropy as defined earlier.

We implemented the combining function, computing the weight using the entropy function, then using them in the weighing process.

The accuracy was around **89%**.



```
def entropy(y):
  # count the number of occurcence of a value (total would be len(y))
  _, counts = np.unique(y, return_counts=True)
  # % of each number of occurence as a list : [0.4, 0.6] ...
  p = np.array(counts.astype('float') / len(y))
  # sum of this formula for each %
  ent = -p.T @ np.log2(p) #A
  return ent

def combine_using_entropy_weighting(estimators, yval,predictions):
  n_estimators = len(estimators)
  wts = [1/entropy(predictions[:, i]) for i in range(n_estimators)]
  wts /= np.sum(wts)
  y_final = np.dot(predictions, wts)
  return np.round(y_final)
```

Figure V.25: entropy weighting

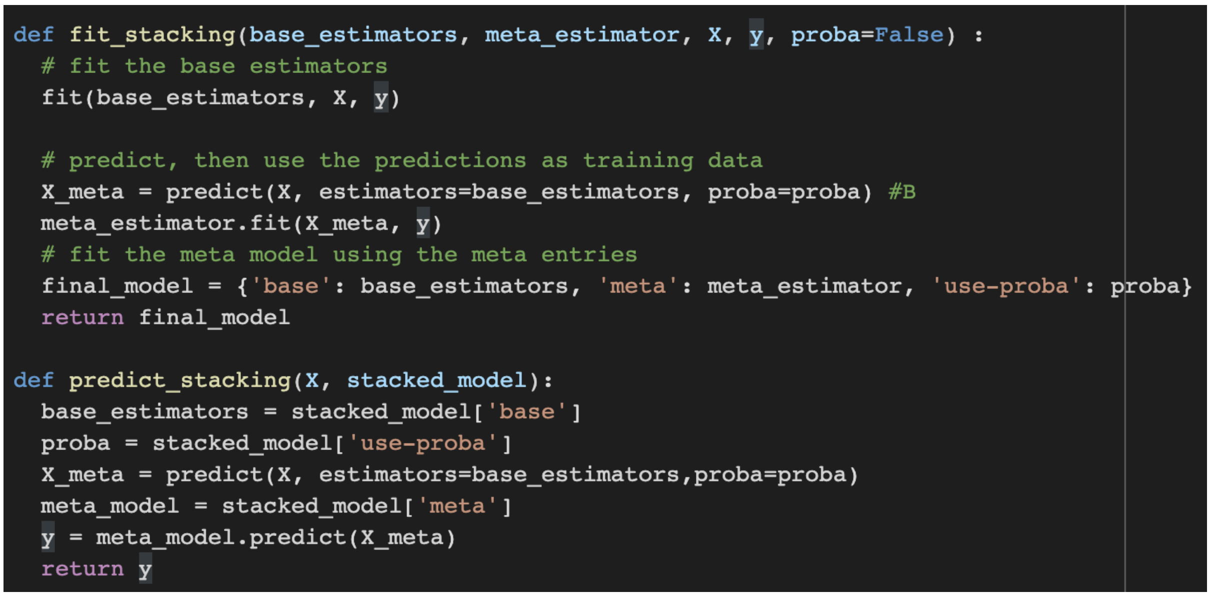
5.6.4.4 Stacking

When it comes to stacking, the phases differ a bit than what was seen in the previous methods. According to what we saw in the previous chapters, and as presented in the architecture we will be doing each step twice. This means training will be done twice. Once for the base estimators and once for the meta-estimator, and also the estimating phase for both base and meta-estimators.

The base estimators are treated the same way they were during the previous experimentations, using the same 2 functions (fit and predict). On the other hand, since the meta-estimator takes the result of each step as its input, we define a new function that uses the previous ones in order to forward the base-outputs as meta-inputs.

The following figure (Figure V.26) shows the implementation of the concept that was detailed in both chapters 3 and 4, and discussed in the architecture. The meta-model used is a LogisticRegression(C=16.768329368110066, solver='liblinear')

The accuracy was around **90%.**



```
def fit_stacking(base_estimators, meta_estimator, X, y, proba=False) :
  # fit the base estimators
  fit(base_estimators, X, y)

  # predict, then use the predictions as training data
  X_meta = predict(X, estimators=base_estimators, proba=proba) #B
  meta_estimator.fit(X_meta, y)
  # fit the meta model using the meta entries
  final_model = {'base': base_estimators, 'meta': meta_estimator, 'use-proba': proba}
  return final_model

def predict_stacking(X, stacked_model):
  base_estimators = stacked_model['base']
  proba = stacked_model['use-proba']
  X_meta = predict(X, estimators=base_estimators,proba=proba)
  meta_model = stacked_model['meta']
  y = meta_model.predict(X_meta)
  return y
```

Figure V.26: Fit and predict functions for stacking

5.6.5 Discussing our results

The aim of this project was to develop a system that would be as accurate as possible reducing the chances for the initial problems to occur (being under and overfitting). This was possible by introducing as much variance as possible while keeping the right amount of bias. Throughout this system, we tried to achieve our goal by combining the result of different highly effective (by extension accurate) models.

By fine tuning our models in the first phase of this experimentation, we made sure we would get the best of them (by accuracy terms) so that we would logically get the best accuracy after combining.

The base models had different accuracy while on different configurations, but it would mainly sit at around **87%**. It was enough to move on to the next step.

When it comes to combining the results, we would logically have a good accuracy since the base models are already cherry picked. This was confirmed by the result of what stacking provided, an accuracy of **90%.** A first reaction would be that our experimentation did lead somewhere, but that we could have used the best base estimator without further overhead, but this is considered wrong, since our system would not only provide consistent results, but also would still help us avoid the main issues by adding the variance we were looking for.

5.7 Conclusion

During this chapter, we discussed the technologies and tools used, the dataset, the architecture we implemented, as well as its implementation. We also discussed the results we got after running our experimentations.

Conclusion

Artificial Intelligence is a huge part of our daily life, it helps us achieve numerous tasks and simplifies the user's life. In this study, we discuss one of its subsets. Machine learning predicts outcomes without being told to do so.

The first objective of this study is to introduce methods that will allow us to overcome some limits of actual machine learning algorithms: overfitting and underfitting. For overfitting, the risk is that the model will be underperforming when used on new data (the model isn't generalizing well). To avoid it, we need to add some diversity and to variance. This can be achieved either by training a model on different versions of data (Bagging) or by training different base models and combining their results. In this project we are taking the second direction: constructing a heterogeneous parallel ensemble learner.

Since many base algorithms must be used, we included in this study five machine learning algorithms among the most used in classification. Also, we have proposed two ways to produce the ensemble learner: a combination function and the use of a meta-model. In this first study three combination functions are included. This project can be considered as the first brick in the construction of heterogeneous parallel ensemble learners. Recall that existing Python modules (like Sklearn) do not integrate such ensemble learners.

This project represented a huge opportunity to learn about machine learning algorithms and their limits. It was also a great chance to discover methods and techniques to deal with the main problems with machine learning algorithms: underfitting and overfitting. On the other hand, this project gave us the prospect to enlarge our technical knowledge. We tried to use modern programming paradigms and platforms.

The experimentations held showed that heterogeneous parallel ensemble learners can improve the effectiveness of machine learning algorithms. Even if the improvement is not huge, we know that the ensemble learner does not suffer from overfitting. It means that it is intended to generalize very well. Further experimentations must be done to integrate more base algorithms in the ensemble learner and also to explore more machine learning datasets.

The objective of this project goes beyond the obtained results. Heterogeneous parallel ensemble learners must be defined as functions (or as a module) that can be integrated in a programming language like Python. Before reaching that goal, many further experimentations and adjustments must be done through new M.Sc projects. The other direction to prospect is the definition of new combining functions to produce the ensemble learner. Once all this is done, the main objective of this study can be reached.

Bibliography

[1] S. Asiri. A Comprehensive Guide to Ensemble Learning (with Python codes). Available at: https://towardsdatascience.com/data-mining-in-brief-26483437f178 (Accessed on: 13 December 2021).

[2] J.Brownlee. Bagging and Random Forest Ensemble Algorithms for Machine Learning. Available at https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/ (Accessed on: 25 December 2021).

[3] J.Brownlee. 4 Types of Classification Tasks in Machine Learning. Available at: https://machinelearningmastery.com/types-of-classification-in-machine-learning/ (Accessed on: 26 December 2021).

[4] J. Brownlee. Difference Between Classification and Regression in Machine Learning Available at: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/ (Accessed on: 20 December 2021).

[5] J.Brownlee .Regression Metrics for Machine Learning. Available at: https://machinelearningmastery.com/regression-metrics-for-machine-learning/ (Accessed on: 04 february 2022).

[6] J.Brownlee. Overfitting and Underfitting With Machine Learning Algorithms. Available at:

https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/ (Accessed on: 22 December 2021).

[7] J.Brownlee. A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks. Available at: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/ (Accessed on: 13 December 2021).

[8] A.Christopher. K-Nearest Neighbor. Available at : https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4 (Accessed on : 11 May 2022).

[9]: N. S.Chauhan.Naïve Bayes Algorithm: Everything You Need to Know. Available at: https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html (Accessed on : 15 May 2022).

[10] D.Fumo. Types of Machine Learning Algorithms You Should Know, Available at:

https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

[11] A.Gupta, M.Dayanand,Understanding Logistic Regression. Available at: https://www.geeksforgeeks.org/understanding-logistic-regression/ (Accessed on: 02 Mars 2022).

[12] D.Gong. Top 6 Machine Learning Algorithms for Classification. Available at: https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501 (Accessed on: 05 December 2021).

[13] R.Gandhi. Introduction to Machine Learning Algorithms: Linear Regression. Available at:https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a (Accessed on: 03 December 2021).

[14] A.Hughes. Data mining, By Craig Stedman. Available at: https://www.techtarget.com/searchbusinessanalytics/definition/data-mining (Accessed on: 07 December 2021).

[15] J.Jordan. Evaluating a Machine Learning Model. Data mining. Available at:

https://www.jeremyjordan.me/evaluating-a-machine-learning-model/ (Accessed on: 20 December 2021).

[16] V.Jain. Introduction to KNN Algorithms. Available at: https://www.analyticsvidhya.com/blog/2022/01/introduction-to-knn-algorithms/ (Accessed on: 08 May 2022).

[17] R.Khandelwal K-Nearest Neighbors(KNN) Available at:

https://medium.datadriveninvestor.com/k-nearest-neighbors-knn-7b4bd0128da7 (Accessed on: 07 May 2022).

[18] P.Majumder. Gaussian Naive Bayes. Available at: https://iq.opengenus.org/gaussian-naive-bayes (Accessed on: 04 May 2022).

[19] S .ML grows its family of classifiers: Gaussian Naive Bayes on Arduino Available at: https://eloquentarduino.github.io/2020/08/eloquentml-grows-its-family-of-classifiers-gaussian-naive-bayes-on-arduino/ (Accessed on: 02 May 2022).

[20] J.Nabi. Machine Learning — Multiclass Classification with Imbalanced Dataset Available at: https://towardsdatascience.com/machine-learning-multiclass-classification-with-imbalAnced-data-set-29f6a177c1a (Accessed on: 03 December 2021).

[21] A.Oppermann.Artificial Intelligence vs. Machine Learning vs. Deep Learning: What's the Difference? Available at: https://builtin.com/artificial-intelligence/ai-vs-machine-learning (Accessed on: 02 December 2021).

[22] K.Sentz, S.Ferson. Combination of Evidence in Dempster-Shafer Theory. Sandia National Laboratories, Sand2002-0835. Available at: https://www.stat.berkeley.edu/~aldous/Real_World/dempster_shafer.pdf (Accessed on : 16 January 2022).

[23] S. Shukla. Introduction to scikit-learn. Available at: https://blog.rwth-aachen.de/itc-events/files/2021/02/01a_Introduction_to_scikit-learn.pdf (Accessed on: 04 february 2022).

[24] C.Stedman. What is data preparation? An in-depth guide to data prep. Available at: https://www.techtarget.com/searchbusinessanalytics/definition/data-preparation (Accessed on: 10 January 2022).

[25] S.Saxena. Beginner's Guide to Support Vector Machine(SVM) Available at: https://www.analyticsvidhya.com/blog/2021/03/beginners-guide-to-support-vector-machine-svm/ (Accessed on: 23 April 2022).

[26] A.Yadav. SUPPORT VECTOR MACHINES(SVM). Available at: https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589 (Accessed on: 22 April 2022).

[27] S.Shukla. Regression and Classification | Supervised Machine Learning. Available at: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/ (Accessed on: 22 January 2022).

[28] Dempster-Shafer Theory. Available at: https://en.wikipedia.org/wiki/Dempster%E2%80%93Shafer_theory. (Accessed on: 15 February 2022).

[29] Heterogeneous Parallel Ensembles: Combining Strong Learners. Available at: https://livebook.manning.com/book/ensemble-methods-for-machine-learning/chapter-3/v-4/1 (Accessed on: 01 November 2021).

[30] Homogeneous Parallel Ensembles: Bagging and Random Forests. Available at: https://livebook.manning.com/book/ensemble-methods-for-machine-learning/chapter-2/v-4/ (Accessed on: 01 November 2021).

[31] Majority Voting. Available at: https://www.sciencedirect.com/topics/computer-science/majority-voting (Accessed on: 22 January 2022).

[32] Manning Live Book. Ensemble Methods for Machine Learning. Available at: https://livebook.manning.com/book/ensemble-methods-for-machine-learning/chapter-2/v-4 (Accessed on: 20 December 2021).

[33] Bagging. Available at: https://www.ibm.com/cloud/learn/bagging (Accessed on: 12 December 2021).

[34] Coronary Artery Disease Available at: https://www.cdc.gov/heartdisease/coronary_ad.htm#:~:text=Coronary%20Artery%20Disease%20(CAD)&text=Coronary%20artery%20disease%20is%20caused,arteries%20to%20narrow%20over%20time (Accessed on: 2 June 2022).

[35] Improving the Performance of Machine Learning Model using Bagging Available at: https://towardsdatascience.com/improving-the-performance-of-machine-learning-model-using-bagging-534cf4a076a7 (Accessed on: 5 January 2022).

[36] Dua, D., & Graff. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science, 2019, http://archive.ics.uci.edu/ml (Accessed on: 2 February 2022).

[37] Soner Yıldırım. The Power of Ensemble Methods in Machine Learning https://towardsdatascience.com/the-power-of-ensemble-methods-in-machine-learning-7ddd28d7d8e6 (Accessed on: 1 june 2022).

[38] Sarah LillyWhite. What is python used for ?
https://www.futurelearn.com/info/blog/what-is-python-used-for (Accessed on: 1 june 2022).

[39] Github documentation. Understanding github actions.
https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
(Accessed on: 1 june 2022).

[40] Github documentation. Git commit https://github.com/git-guides/git-commit
(Accessed on: 1 june 2022).