Ensemble Methods in Supervised Learning:

Review towards an application in a model for predictions about ecology

Nuria Gómez Vargas

Final Degree Project

Supervised by Rafael Blanquero Bravo

University of Seville

June 2020

Rafael Blanquero Bravo,

MAKES CERTAIN

that Nuria Gómez Vargas, has performed under my supervision the project entitled

Ensemble Methods in Supervised Learning:
review towards an application in a model for predictions about ecology

Once reviewed, I authorize the beginning of the procedures for its presentation as Final Degree Project to the court to judge.

Signed,

Rafael Blanquero Bravo
at University of Seville
16/06/2020

I, Nuria Gómez Vargas with ID 47564221Z,

DECLARE

my authorship of the work presented in the memory of this Final Degree Project which is entitled

Ensemble Methods in Supervised Learning:
review towards an application in a model for predictions about ecology

Signed,

Nuria Gómez Vargas
at University of Seville
16/06/2020

En primer lugar me gustaría agradecer a mi tutor, Rafa, por su voto de confianza y acompañamiento durante estos meses; y a mis padres, sin vuestra ayuda Marta y yo no podríamos haber llegado a donde estamos.

La carrera de Matemáticas es preciosa y más cuando tienes el placer de compartirla con tan buenas amigas - puede que esta etapa termine, pero tened por seguro que siempre vais a poder contar conmigo - y un profesorado que no sólo siente pasión por esta ciencia, sino también por transmitirla.

Por último, deseo a gradecer a Falk Huettmann que me proporcionara su libro, *Machine Learning for Ecology and Sustainable Natural Resource Management*, pues ha sido mi fuente principal de inspiración y la herramienta para comprender que el Aprendizaje Automático nos brinda una solución innovadora al problema ecológico global en el que estamos sumidos.

ABSTRACT

Advancements in Machine Learning techniques such as Ensemble Methods provide a powerful tool for quickly building accurate predictive models for sustainable decision making. These algorithms help to enhance predictions by combining the decisions of a set of models. Here, we use Stacking, Boosting and Bagging to asses countries footprint regarding ecological deficit or reserve using data collected by the Global Footprint Network. The comparison of those techniques in this context allows us to conclude that the first one provides the most accurate model (Accuracy> 0.97), although the others have the advantage of a lower cost and complexity, also providing remarkable predictions.

Keywords — Ensemble, Supervised Learning, Stacking, Boosting, Bagging, Classification, R, Ecology

RESUMEN

Los avances en las técnicas de Aprendizaje Automático, como los Métodos de *Ensemble*, proporcionan una herramienta poderosa para construir rápidamente modelos predictivos precisos para una toma de decisiones sostenible. Estos algoritmos ayudan a mejorar las predicciones al combinar las decisiones de un conjunto de modelos. Aquí, utilizamos las técnicas de *Stacking*, *Boosting* y *Bagging* para evaluar la huella de los países con respecto al déficit o reserva ecológica utilizando los datos recopilados por la *Global Footprint Network*. La comparación de esas técnicas en este contexto nos permite concluir que la primera proporciona el modelo más exacto (Precisión > 0.97), aunque las otras tienen la ventaja de un menor coste proporcionando también predicciones destacables.

Palabras clave — *Ensemble*, Aprendizaje Supervisado, *Stacking*, *Boosting*, *Bagging*, Clasificación, R, Ecología

Contents

Introduction

Un país sin investigación es un país sin desarrollo.

Margarita Salas

The increase in world population, technological development and the fever for rapid consumerism, are some of the factors that have influenced the deterioration of the environment. Humanity currently faces serious global environmental problems such as Climate Change, loss of biodiversity and ecosystems, depletion of annual natural resources at alarmingly early dates, etc. To solve these problems and try to remedy their consequences, it is necessary to develop efficient and scientifically backed actions that contribute to the sustainability and conservation of nature. Fortunately, the development of new technologies also provides us with tools that we can use to benefit the environment.

We are currently at the time of the fourth industrial revolution, marked by emerging technological advances in several fields such as Artificial Intelligence, the Internet of Things or Big Data. It is precisely this development, which has contributed so much to the generation of negative situations and environmental problems, that provides us with tools that help us to improve our understanding of environmental processes and increase the protection of our ecosystem. For example, researchers from the project "Conservation management of endangered wildlife species in Slovenia" assessed brown bear habitat using existing geocoded data on bear population spatial distribution as well as geographical information system (GIS) data layers covering several aspects of the study area [1]. They aimed to construct a model that supported spatial decisions, such as the extension of the current core bear protection area.
Among its broad possibilities, Artificial Intelligence also offers us powerful tools to predict scenarios of air quality, environmental changes, etc. The University of Alaska Fairbanks carried out a cluster analysis that used historical climate data to develop land-cover categories, to predict changes in the distribution of renewable resources [2]. The results of projecting climate data one century into the future showed that profound changes can be expected across the study area, with some regions having up to three dynamic landscapes, considering those that are least likely to change as potential refugia.

Each Machine Learning algorithm certainly has its assets and liabilities and, consequently, choosing the right algorithm to employ could be quite a hard task to do. While their use will always be essential in ecological data analysis, it is necessary to explore new techniques to model and treat the flood of environmental data from sources with different provenances and complexities such as drones, acoustic monitoring devices, tagged animals, and more. Ensemble Methods help with this, enhancing predictions by combining the decisions of a set of models. Moreover, the use of these techniques has proven [3, 4, 5] that it will change ecology management and decision making for the better.

With the intent to extend the applications of the Ensemble Methods to the Ecology and promote their use, this Final Degree Project presents a bibliographic review of these techniques towards a subsequent implementation thereof. First of all, we look over the foundations of Machine Learning, part of Artificial Intelligence oriented to "educate" the machinery to promote its autonomy. This field is mainly divided into Unsupervised and Supervised Learning. The latter allows predictions based on labelled data and, as we have a "supervisor", it is possible to assess model performance. So as not to fall in overfitting, the dataset is splitted into a training sample and a test sample for assessment. When there is not enough data, this division can be done by sample re-use, for instance, via cross-validation. Ensemble techniques belong to Supervised Learning, and their way to proceed is to give a final decision based on a vote or average of the predictions obtained from several base models. Some of the most prominent ensemble-based algorithms are Stacking, Boosting and Bagging.

The purpose of this project is to implement these three types of algorithms with an environmental dataset, for which the R statistical software [6] is used. The dataset has been chosen from the Global Footprint Network [7] database. In 2019, this organization informed that the annual natural resources of the entire planet were depleted in just seven months, which is the earliest date since there are records. The measurement and knowledge of this footprint help countries to improve sustainability and well-being. If the ecological deficit or reserve status were known in advance, countries could implement environmental policies to reverse their situation or continue to do so as well as so far.

Machine Learning

Reserve your right to think, for even to think wrongly is better than not to think at all.

Hypatia of Alexandria

We can formally define Machine Learning (ML) as the science and art by which computers learn things by themselves [8].

The birth of ML dates back to the end of World War II, when Alan Turing creates the "Turing Test" (1950) to determine if a machine was actually intelligent. To pass the test, the computer had to exhibit a human behaviour in natural language conversations. Only a couple of years after this, Arthur Samuel writes the first computer program capable of learning, designed to play checkers and improve game after game.
In the 1960s, we come across with a boom of interest in Machine Learning, due to Frank Rosenblatt's first neural network, the 'Perceptron', a technology that resembles the human brain by connecting a network of nodes, where simple decisions are made, to a larger program to solve complex problems.
However, in the decade of the 70s different agencies cut the funds for Artificial Intelligence (AI) research after numerous years of very little progress, resulting from the limited computational power. In 1967, the "Nearest Neighbor" algorithm was written. This milestone is considered as the birth to the field of pattern recognition.
In the early 80s, Gerald Dejong introduces the concept of 'Explanation-Based Learning' (EBL), where a computer analyzes training data and creates a general rule that can be followed to discard data.
We soon enter the era of computers and internet through the 1990s. It was during this time when Machine Learning gained popularity thanks to computer science and statistics, that led to probabilistic approaches in AI. People were bombarded with vast amounts of data, and this technology began to be used in commercial areas for data mining, web applications or text learning.
The arrival of the new millennium brought a great development as computers got more powerful. It was during this period when machine learning (primarily through a few select algorithms) was picked up in ecology [9]. Nowadays, everything points out that we will

continue having more and more data to feed our algorithms while the scientific community does not seem to run out of ideas to continue advancing in the field.

ML methods are deployed to solve challenging real-world problems that usually involve large amounts of data, especially when efficient decision-making is demanded, for instance, in the case of natural resource management. Let us consider the extraction of non-renewable resources such as petroleum, mainly used to generate energy. We have at our disposal a set of examples consisting of georeferenced occurrences data and environmental variables (topography, soil type or latitude) associated with the appearance of these resources or not. In order to extract the relevant features to characterize the extraction, we will have to make some questions at this point, e.g., is terrain relief an important variable in this study? We name these variables *input variables* because we will use them to learn some concept or predict over unobserved scenarios whether extractions might occur in space and time or not. The *output variable* can be an integer value, which assumes 1 for "yes", 0 for "no". ML algorithms solve the problems in an indirect way [10]; firstly, these associations are modeled and then the model is used to predict the label of a new input data point. That is to say, ML is the scientific discipline in charge of enabling machines to learn and predict using past experiences.

But Machine Learning is not just a database problem; it is also a part of Artificial Intelligence [11]. The breadth of AI field of study covers computational models focused on carrying out human activities based on two of our primary characteristics as thinking beings: reasoning and behaviour, whereas ML is oriented to "educate" the machinery to promote its autonomy. It can also provide insights into structures and patterns within large datasets [10].

In practice, it is always recommended to test and compare the behaviour of different algorithms to choose the most appropriate one in each specific case. This behaviour might be influenced, for example, by the available data or number of features. Therefore, selecting the right algorithm becomes an important factor. Building a ML model is not just selecting the right algorithm, although it is a crucially important factor, but it is a process that usually involves the following steps [12]:

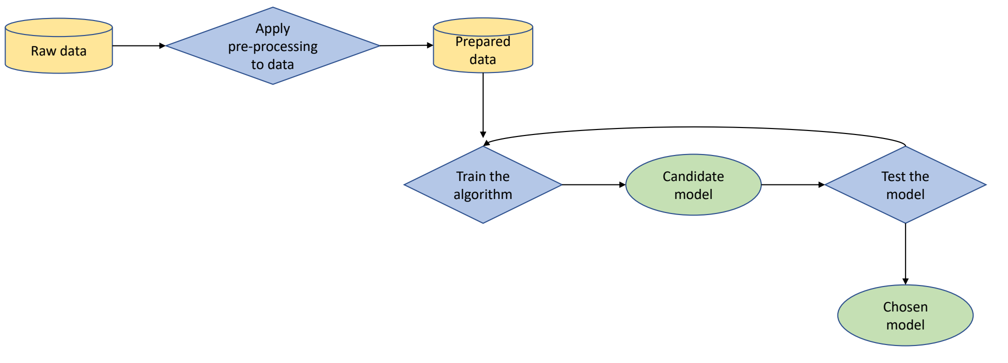


Figure 2.1: Machine Learning process

When we have collected the data, it is necessary to look over them in order to detect those that are incomplete, inconsistent or maybe with too much noise. That is why we start with the pre-processing step [13], to give the data the correct format to feed the learning algorithm for training. This step is divided into two tasks: data preparation, which is in charge of initializing the data properly (by integrating data from different sources into a single and homogeneous dataset, cleaning the errors, smoothing the noise, transforming data from one format to another, etc), and data reduction, which allows us to obtain a lessened representation of the data (it is carried out by reducing the number of attributes, which is helpful when interpreting, or when the original dataset has a huge number of instances, by replacing with a smaller representation of the data). After this stage, algorithms must be able to extract useful information from the prepared data and then make predictions efficiently. To test the algorithm is to evaluate how accurate the algorithm is in its knowledge extraction and predictions. If the expected performance is not obtained, we can go back to the previous step and continue training the algorithm by changing some parameters until achieving acceptable results.

As far as tools for data science are concerned, R [6] and Python [14, 15] are the tools of choice in most of the research groups and universities. A new programming language is also emerging, Julia [16], which is being welcomed by the statistical community.

The area of Machine Learning is typically organized in two main branches [17]: Supervised Learning and Unsupervised Learning. But there is also room for another distinction in the classification which falls somewhere between the previous two techniques: Semisupervised Learning. We will discuss with some examples their tasks and applications in the following sections.

2.1 Unsupervised Learning

Unsupervised Learning takes place when we only know the input data, but there is no output data that corresponds to a given input. That is to say, there is no such supervisor, meaning that we lack of labelled data available for training. Therefore, we can only describe the structure of the data, to try to identify trends of similarity that simplifies the analysis. In this way, it has an exploratory character and is considered a pre-processing step, as it is as well used for knowledge discovery before another Machine Learning technique is applied [18]. A marked method for Unsupervised Learning is the cluster analysis.

- ***Clustering***: The goal of this task is to put into different groups the input data points, in a manner which the points within a cluster share properties in common. We do not have training data to learn what the clusters are. Instead, they are inferred from the dataset by maximizing intra-cluster similarity and inter-cluster dissimilarity.
  An application is, e.g., cluster analysis as a means of defining 'cliomes' (climate envelope based biomes) [2], to show how regionalized climate data can be and to be later projected 100 years into the future. Each data point was assigned 24 variables, representing mean monthly temperature and precipitation. The model provided natural groupings, which also allowed identifying outliers, this is, rare events.
  We can highlight two families of clustering algorithms:

  – *Hierarchical clustering* looks for building a hierarchy in which elements are grouped. In this way, it shows the proximity relations that exist between the elements and therefore, the choice of distance measurement is important for clustering to make sense, as with different measures we will have different groupings. Among the proximity measures between clusters we find: single link (the proximity between two clusters is defined as the proximity between the two closest elements that belong to different clusters), complete link (it is defined as the proximity between the two farthest elements that belong to different clusters), group average (it is defined as the average proximity between all pairs of elements that belong to different clusters), distance between centroids (the proximity between two clusters is defined as the proximity between the centroids of each cluster, that is, the vectors whose coordinates are the means of the components of the variables in the cluster) and Ward's method (at each stage, the joined clusters are those two that lead to the cluster with the least sum of squared errors). When performing a hierarchical clustering it is not necessary to determine in advance the number of clusters that we are going to form. In a graphic representation, the elements are nested in a tree-shaped hierarchy called dendrogram. Two recursive methods can be then given for obtaining hierarchical clustering schemes [19], depending on the direction in which the algorithm executes the grouping: agglomerative procedures (we begin to group from each individual element and at each iteration the two closest clusters merge until there is only one cluster that brings together all the elements) and divisive procedures (we proceed in reverse, we start from a single cluster containing all the elements and at each stage we divide into smaller clusters).

– *Partitional clustering* has as objective to obtain a partition of the elements into groups in such a way that all the objects belong to one of the possible k (fixed in advance) disjointed clusters. Each cluster has a centroid associated (geometric centre of the cluster) and the data are assigned to the cluster whose centroid is closest, using any distance metric. Iteratively, the centroids are updated based on the assignments of points to clusters, until the centroids stop changing. One of the most widely used methods in practical implementations because of its simplicity is the k-Means method [20]. This method was proposed by McQueen in 1967 , but the most efficient algorithm is the one proposed by Hartigan and Wong, which is initialized by randomly choosing a centroid configuration (this is repeated several times with different configurations to get the best one). Secondly, we assign each point to the nearest cluster and the new centroids are recalculated. After this step, we reassign each object to the nearest centroid. We will repeat from step two until a certain stop criterion is reached. Nevertheless, the k-Means algorithm is sensitive to the presence of outliers. To limit their influence, the k-Medoids method looks for the k partitioning representative individuals (or medoids) among the set of observations. In each iteration, one of the representatives is replaced with a representative from the current data, in order to check if the quality of the clustering improves [20].

2.2 Semisupervised Learning

Approaching the field in which Ensemble Methods are registered, where we will consider labelled data, this section addresses the problem of learning from a small set of labelled examples and a large set of unlabelled examples [21], i.e., Semisupervised Learning [22]. Currently we deal with a large amount of data, however, not all of them are assigned a label with which to create a classifier. The clearest example of this is the availability of untagged text and images. The challenge is to see if unlabelled points can be combined with those labelled ones to build a better classifier due to labelling, which is often done manually, is associated with a high cost. The algorithm uses the labelled data points to build a classification function. It is necessary to point out that every class must have some labelled examples.
Assume that we want to create the animal inventory of a natural reserve. To exemplify a basic case, there are only five categories of interest: mammal, bird, fish, reptile and amphibian. For this task, we need thousands of images of animals in the five categories, and for some of these images we need to identify and manually tell the machine whether the image represents, for

example, a mammal or a fish. From this set of images (mostly unlabelled), the algorithm will find similarities images and all similar images will get the same tag. Thereby, the machine can use what has already taken in for its further learning as the animal population in the natural reserve grows. Clearly, Semisupervised Learning is quite useful and researchers in the field have discovered that its use can significantly improve learning accuracy.

2.3 Supervised Learning

Supervised Learning is compound of a set of techniques that allows future predictions based on labelled data. The *label* is just the response variable Y related to the explanatory variables $X = (x_1, ..., x_p)$, the analysed characteristics, which we already know and have registered in the dataset. The prediction obtained is represented by means of a function that takes the explanatory variables as input and returns the variable to be predicted as output, with the objective of understanding the relationship between them but also to predict Y for future observations (or test sample) of x_i, i = 1, ..., p. This output is continuous in regression problems and categorical in classification problems. There are two main tasks in Supervised Learning that should be discussed, function approximation and classification.

- ***Function approximation***: We consider an input, X, an output, Y, and the task is to learn the mapping from the input to the output [11]. Regression is a type of function approximation.
  Let us consider the analysis of the abundance of the soft coral species [23] from the Australian central Great Barrier Reef. We want to predict the ratings of abundances of *Asterospicularia laurae* (the numeric response variable): 0 (absent), 1 (few), 2 (uncommon), 3 (common), 4 (abundant), 5 (dominant). The explanatory variables used in the model are: shelf position, site location and depth. Let X be the spatial attributes and Y the abundance of the species. The mapping mentioned in the beginning is a model that depends on certain parameters:
  $$Y = f(X, \vartheta)$$
  here, f is the regression function with parameters ϑ. The ML techniques estimate ϑ so that the approximation error is minimized and thus have good predictions.

- ***Classification***: Classification problems are characterized by having a qualitative or categorical variable as response. The classifying methods use explanatory variables to assign the category of the observations.
  Identifying brown bear habitat [1] is an example of a classification problem where we consider two categories: habitat and non-habitat. The information about bear sightings includes data that are relevant in categorizing the study area: percentage of forest,

proximity to settlements, elevation above see, and so forth. The assessment is carried out making use of existing geocoded data on bear population spatial distribution as well as data containing ecological features of the study area. Once the algorithm has been trained with the input data, it learns a set of rules providing the association between the attributes which characterize an area and brown bear habitat. The function that distinguishes the different classes is called *discriminant*, e.g.,

```
IF elevation <= 541 THEN non-habitat
ELSE
IF percentage-of-rural-population <= 1 THEN habitat
...
```

That is, the algorithm creates a model that fits the past data with a goal, to be able to pick out a class for future observations.

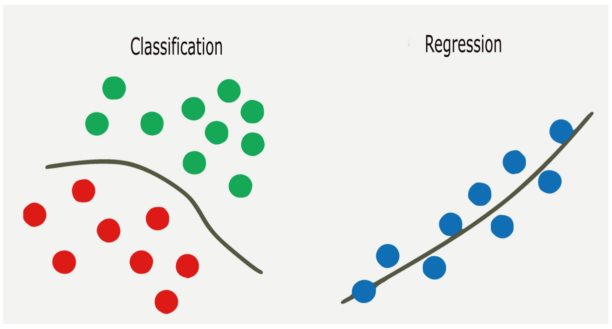


Figure 2.2: Supervised Learning tasks

Focusing on algorithms, we will briefly introduce the principal supervised learning methods hereunder:

- ***Decision Trees*** are tree-like representations of possible solutions to a decision based on certain conditions, and can perform both classification or regression tasks. To classify an observation the algorithm determines the main class in the terminal node in which this observation falls. On the other hand, with a new observation under consideration, regression is done by assigning as response the average of all the response variables of the training observations in that partition. The best known methodologies for building decision trees are CART (classification and regression trees), C4.5 or C5.0.
- The ***Naive Bayes classifiers*** are especially appropriate when the dimension of the input space is high, making density estimation unattractive [24]. They are based on the application of Bayes' theorem, with strong assumptions of independence between the features. They are a particular case of the so-called Bayesian networks [25].

- ***Ordinary Least Squares Regression*** is a method to perform linear regression [26]. The OLS approach adjusts the coefficients β of the linear model to minimize the residual sum of squares $RSS(\beta) = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2$.

- ***Logistic regression*** is a classification method that estimates the probabilities of the k classes, $P[Y = i|X = x]$ $(i = 1, \ldots, k)$, via linear functions in the independent variables using the cumulative distribution function of logistic distribution, $F(t) = \frac{1}{1+e^{-t}}$.

- ***Support Vector Machine*** is a powerful algorithm for binary classification [27]. Given a set of points of two types, SVM generates a hyperplane to separate the data into two groups. We say that the hyperplane is an optimal separating hyperplane if it maximizes the distance between the hyperplane and the closest observation. There is also a version for regression tasks know as SVR (Support Vector Regression) [28].

- ***k-Nearest Neighbor***, shortened $k - nn$, is an algorithm which classifies each new data in the corresponding group, according to k closer neighbors to one group or another. To proceed with the classification, it calculates the distance of the new element to each of the existing ones, and orders those distances from least to greatest to select the group to which they belong. This group will therefore be the one with the highest frequency considering the shortest distances.

- ***Neuronal Networks*** are a set of algorithms specially designed for pattern recognition. These networks are inspired by biology: resembling how neurons work in our brain, they try to simulate the way people make decisions. They are based on a basic structure, the "perceptron", and on the "backpropagation" mechanism, which allows the neuron to learn by itself and discover the hidden information in the input data with which we train the network. *Deep Learning* is a particular case of neural networks that is characterized by having multiple layers of neurons connected to each other. Although what defines deep learning is the hierarchical data processing, that is, the use of neural networks to obtain increasingly significant representations of the data through layered learning.

- ***Regularization techniques*** are used for model selection and to avoid overfitted models when predicting. In the context of linear models, we can give a generic formulation of the regularization techniques as follows:

  $$\hat{\beta} = arg\min_{\beta}\left\{\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \phi_\lambda(\beta)\right\}$$

  where $\beta = (\beta_1, \ldots, \beta_p)$, $\lambda \geq 0$ and $\phi_\lambda(\beta) = \lambda\sum_{j=1}^{p}\phi_j(|\beta_j|)$ is the penalty function on the size of β, which depends on λ.
  A family of penalty functions that is widely used is the one corresponding to the L_q norm, $q > 0$, given by

  $$\phi_\lambda(\beta) = \lambda\left(||\beta||_q\right)^q = \lambda\sum_{j=1}^{p}|\beta|^q$$

Among these techniques, we can highlight the *Ridge Regression*, which takes as penalty function the L_2 norm. Thus, the estimate will be given by the coefficients that minimize the corresponding function, this is

$$\hat{\beta}^{\text{ridge}} = arg \min_{\beta \in \mathbb{R}^{p+1}} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} = arg \min_{\beta \in \mathbb{R}^{p+1}} \left\{ RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

where $\lambda \geq 0$ is the fixed parameter that controls the shrinkage.
As an alternative to this type of regression, we find the LASSO (least absolute shrinkage and selection operator), with a slight difference in the penalty that has important consequences. Meanwhile with Ridge Regression we control the size of the coefficients β, the LASSO regression performs variable selection due to the L_1 norm, which looks more like $\# \{ j : \beta_j \neq 0 \}$. We can formulate the problem as

$$\hat{\beta}^{\text{lasso}} = arg \min_{\beta \in \mathbb{R}^{p+1}} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} = arg \min_{\beta \in \mathbb{R}^{p+1}} \left\{ RSS + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

Lastly, the regularization technique known as *Elastic Net* [29] combines the advantages of Ridge (highly correlated predictors have similar estimated coefficients) and Lasso (sparse solutions). We can define the Naive Elastic Net estimator as

$$\hat{\beta}^{\text{ene}} = arg \min_{\beta} L(\lambda_1, \lambda_2, \beta) = arg \min_{\beta} |y - X\beta|^2 + \lambda_2 |\beta|^2 + \lambda_1 |\beta|_1$$

where $|\beta|^2 = \sum_{j=1}^{p} \beta_j^2$, $|\beta|_1 = \sum_{j=1}^{p} |\beta_j|$ and $|y - X\beta|^2 = RSS$.

The described techniques can be combined through the so-called *Ensemble Methods* to build a final model. These learning algorithms are the case of study in this project and will be extensively introduced in the next chapter.

2.3.1 Model assessment and effectiveness measures

The objective of this subsection is to assess model performance, meaning the predictive ability on new and independent data. This assessment is a fundamental step in application to practice, as it provides the quality of the chosen model via different effectiveness measures.

Let us consider the case of a regression problem, where the target Y associated with the input variables in a vector X is numerical. Let also $\hat{f}(X) = \hat{Y}$ be the prediction of the target inferred in the model. To quantify the notion of "good", we consider a loss function $L(Y, \hat{f}(X)) = L(Y, \hat{Y})$ that takes both Y and its predictor and returns a real number which indicates the loss (or cost, or undesirability) of predicting $\hat{f}(X)$ when the correct label is Y [30]. Specifying this notion of goodness of fit, the commonly used functions to measure the loss are the following ones:

– *Mean Absolute Error (MAE)*: It is an average of the absolute errors $|e_i| = |y_i - \hat{f}(x_i)|$ and so, it is given by

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{f}(x_i)|}{n}$$

– *Root Mean Squared Error (RMSE)*: The calculation of the RMSE is carried out considering the square root of the average Euclidean distance between observed and estimated values.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2}{n}}$$

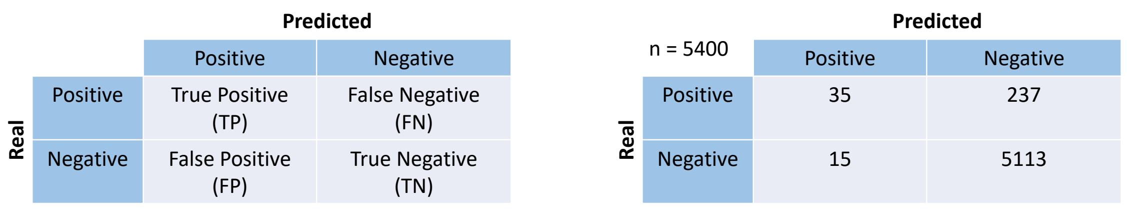
– *Relative Absolute Error (RAE)*: It takes the absolute errors and it is relative to a simple predictor, which is just the average of the actual values, $\overline{y}$. The RAE index ranges from 0 to infinity, with 0 corresponding to the ideal [31] as the numerator is equal to zero for a perfect fit.

$$RAE = \sum_{i=1}^{n} \frac{|y_i - \hat{f}(x_i)|}{|y_i - \overline{y}|}$$

– *Root Relative Squared Error (RRSE)*: This measure is similar to the previous one in the sense that it is relative to the simple average of the values but instead of using the absolute error, it is calculated making use of the Euclidean distance.

$$RRSE = \sqrt{\sum_{i=1}^{n} \frac{(y_i - \hat{f}(x_i))^2}{(y_i - \overline{y})^2}}$$

Similarly, if we consider now a categorical response G and the problem is to estimate the probability of G taking a category given the input X, or simply producing $\hat{G}(X)$, we would contemplate the loss function $L(G, \hat{G}(X))$. A typical choice would be $L(G, \hat{G}(X)) = I(G \neq \hat{G}(X))$ (0-1 loss). But there are other tools that allow the visualization of the performance of a classification algorithm. Focusing on a binary classification problem (e.g., positive and negative classes), we can define a *confusion matrix* as the 2x2 matrix whose rows are named according to the actual classes, while each column represents the classes provided by the model. Thus, each instance represents the number of predictions in each case. Let us introduce some notation related to the four possible cases when predicting:
· True Positive (TP) is the number of positives that were correctly classified as positive by the model.
· True Negative (TN) is the number of negatives that were correctly classified as negative by the model.
· False Negative (FN) is the amount of positives that were incorrectly classified as negative.
· False Positive (FP) is the amount of negatives that were incorrectly classified as positive.



		Predicted	
		Positive	Negative
Real	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

	n = 5400	Predicted	
		Positive	Negative
Real	Positive	35	237
	Negative	15	5113

Figure 2.3: Confusion matrix in a binary classification problem

These matrices summarize the class allocations made in each analysis from which accuracy may be assessed and may also provide valuable information for later users of the classification [32]. We can define a useful series of effectiveness measures based on the values of this confusion matrix:
– *Accuracy (AC)*: It is no more than the percentage of cases correctly allocated.

$$AC = \frac{TP + TN}{Total}$$

– *Misclassification Rate (MR)*: It indicates the percentage of the data that is classified incorrectly.

$$MR = \frac{FP + FN}{Total}$$

– *Sensitivity*: Also known as *recall* or *True Positive Rate*, it is the proportion of cases that were correctly identified by the algorithm in the positive class.

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

– *Specificity*: Also known as *True Negative Rate*. When the class is negative, it indicates what percentage is correctly classified.

$$Specificity = \frac{TN}{TN + FP}$$

– *Precision*: When positive is predicted by the model, it shows what percentage is correctly classified.

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

– *Negative Predictive Value*: When negative is predicted, it shows what percentage is correctly classified.

$$NPV = \frac{TN}{TN + FN}$$

– *F-Measure*: Also know as F1 score or F score, it is a measure of a test's accuracy in statistical analysis. It is a harmonic mean that combines the values of precision and recall, and reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = 2\frac{precision \cdot recall}{precision + recall}$$

– *Kohen's Kappa statistic*: It is an index that is used to measure concordance among a set of coders making category judgments [33] (interrater reliability), taking into account the possibility of the agreement occurring by chance. If the raters completely agree, then $\kappa = 1$. On the contrary, if there is no agreement other than what one would expect by chance, $\kappa = 0$. Let us denote by p_0 the proportion of observed agreement among raters, and by p_e the hypothetical probability that the raters would randomly both agree. The formula to calculate Cohen's kappa is

$$\kappa = \frac{p_0 - p_e}{1 - p_e} = 1 - \frac{1 - p_0}{1 - p_e}$$

The convenience of using one metric or another as a measure of the estimator goodness will depend on each particular case. We will never be able to guarantee that the percentage of correct will be 100% and, in fact, improving the proportion of TP will lower TN and vice versa. One way to observe the relationship between sensitivity and specificity is with ROC (Receiver Operator Characteristic) curves. In ROC space, one plots the False Positive Rate (which coincides with $1 - Specificity$) on the x-axis and the sensitivity (TPR) on the y-axis [34]. Ideally, the area under the curve (AUC) should be as large as possible, because this would mean that to guarantee a high sensitivity it is not necessary to inflate the amount of false positives.

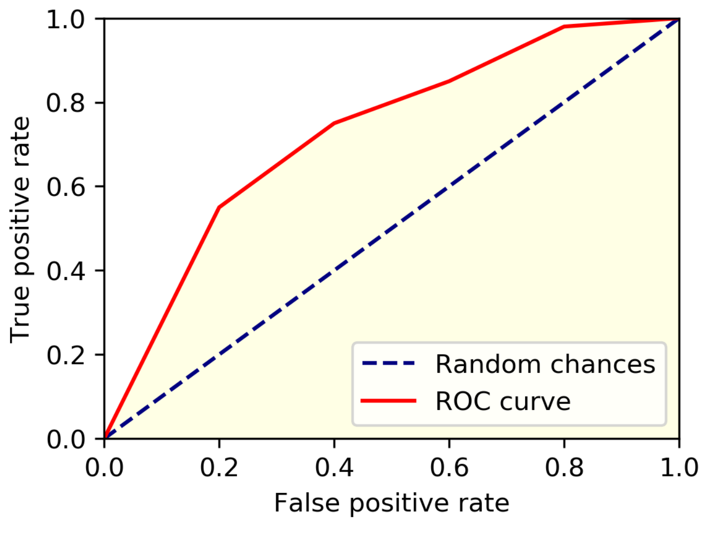


Figure 2.4: ROC curve example and its AUC

Thereupon, our predictors will be defined to minimize the *expected prediction error* $E[L]$. For convenience, we will talk in terms of Y and $\hat{f}(X)$, i.e., a regression problem (explanations in terms of a classification problem can be easily deduced). Our predictor will depend on a tuning parameter or vector of parameters α, which adjusts the complexity of the model. We express this by $\hat{f}_\alpha(x)$ and, being consistent with what was said above, we will choose the value of α that minimizes the error.

In order to develop the concept of prediction error correctly, first it is necessary to talk about the procedure to follow regarding the use of the sample. If we are rich in data, the suitable method is to randomly split the dataset into two parts: a *train sample* and a *test sample*. The train sample is used in the Supervised Learning part in charge of model construction. Whereas with the test sample we check if the model predictions fit well with their real values. Ideally, the test set should be kept in a "vault", and be brought out only at the end of the data analysis [35]. There is no general rule to divide the data, but we should leave a larger sample for training.

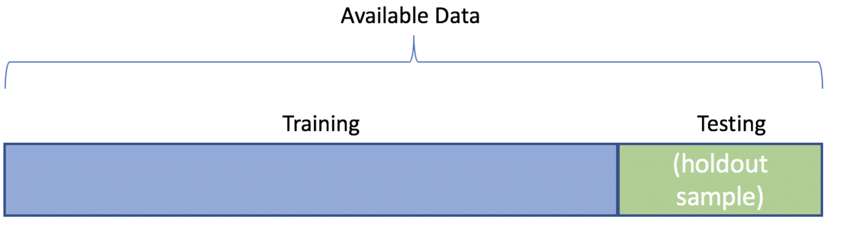


Figure 2.5: Train and test split

With these concepts in mind, we are prepared to go in depth in the effectiveness measures.

We are interested in knowing the expected prediction error (or expected test error) of our model over an independent test sample

$$Err = E[L(Y, \hat{f}(X))]$$

Considered only the information in the same training set, we can average the loss over this sample of length n_{train} with the *training error*

$$\overline{err} = \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} L(y_i, \hat{f}(x_i))$$

The following lines will give an explanation for why the training error is not suitable to estimate the test error. Training error consistently decreases with model complexity, typically dropping to zero if we increase the model complexity enough [35]. In this way, we would obtain an overfitted model that provides poor predictions on unseen data. Also, if the data points in the test sample are used to build and choose the final model, the prediction error would be considerably minimized and we would not obtain the true error in the test.

Nevertheless, it may be necessary to consider situations in which we do not have enough data to make the split, as in most real world problems. There are methods for these circumstances, which will approximate the error by sample re-use:

- ***Cross-validation (CV)***:
  K-fold cross-validation is a straightforward way for estimating prediction error. A positive integer K is taken, and the sample is randomly divided into K groups of the same size approximately. For each group, we take it as test sample and fit the model to the rest. In total, K models have been fitted with the $K-1$ parts defining the training sample, and we take as prediction error the average of the K errors obtained in each model.

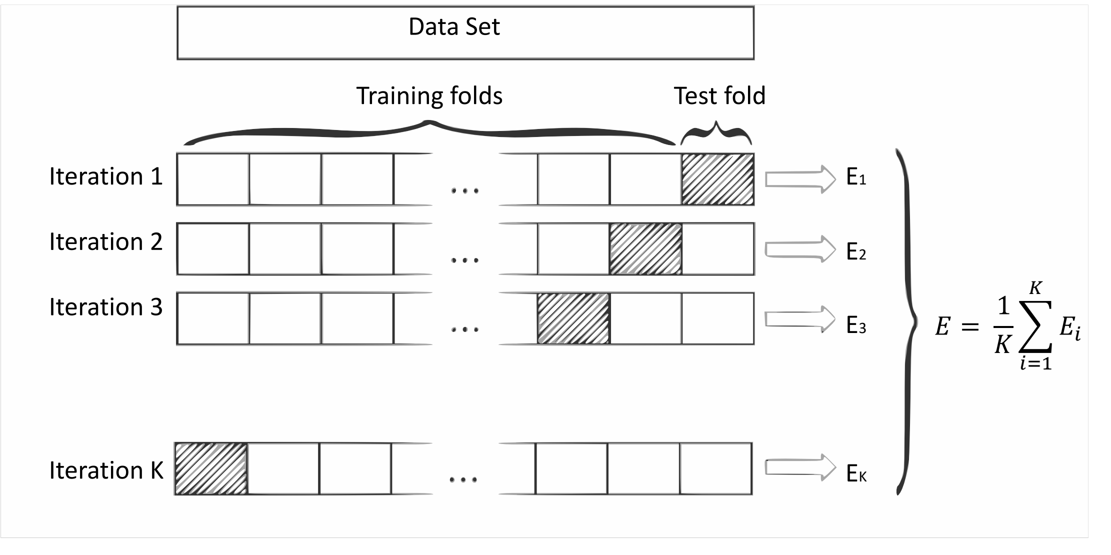


Figure 2.6: K-fold cross-validation

The extreme case, where we consider n folds (being n the length of the whole sample), is known as *leave-one-out*. In each step, the model is fitted using all the data except one. This single data is used as test sample, taking as the prediction error the average of the n errors obtained.

Going into more detail, let $\kappa : \{1,\dots,n\} \mapsto \{1,\dots,K\}$ be an indexing function that indicates the partition to which observation i is allocated by the randomization. Denote by $\hat{f}^{-k}(x,\alpha)$ the model indexed by the tuning parameter α and fitted with the kth part of the data removed [35]. We define

$$CV(\hat{f},\alpha) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}^{-\kappa(i)}(x_i,\alpha))$$

With this function we obtain an estimate of the test error curve, and finally we choose the tuning parameter $\hat{\alpha}$ that minimizes it.

- ***Monte Carlo cross-validation (MCCV)***:
  Monte Carlo validation is another simple and effective method, similar but subtly different from the previous cross-validation. The technique is based on repeating K times a procedure that consists of randomly splitting the samples into two parts $S_{train}(k)$ (of size n_{train}) and $S_{test}(k)$ (of size n_{test}), $k = 1,\dots,K$. The repeated MCCV criterion is defined [36]:

$$MCCV_{n_{test}}(k) = \frac{1}{n_{\text{test}}} \sum_{i \in S_{test}(k)} L(y_i, \hat{f}(x_i))$$

  Finally, we take as prediction error the average of the obtained errors, this is, $\frac{1}{K}\sum_{k=1}^{K} MCCV(k)$.

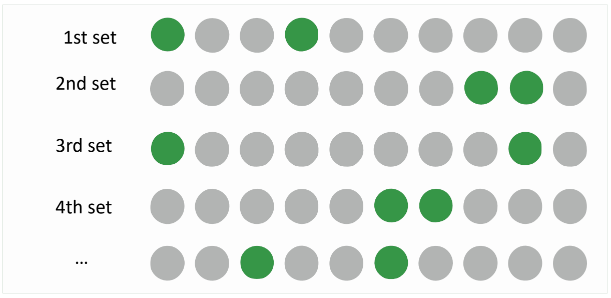


Figure 2.7: Monte Carlo validation (20%)

- ***The Bootstrap method***:
  The *bootstrap* method [**?**] was introduced in 1979 for estimating the accuracy of statistical models and it depends on the notion of a *bootstrap sample*, which is defined to be a random sample of size n drawn with replacement from the population of n objects $\mathbf{x} = (x_1, \ldots, x_n)$, say $x^* = (x_1^* , \ldots, x_n^*)$ [24]. The procedure is repeated B times (we call B the size of the bootstrap). In each rerun, we take the bootstrap sample for training, and the observations that have not been drawn (*out-of-bag*) as test sample. The out-of-bag estimates are remarkably accurate [37]. As in the previous techniques, we select as prediction error the average of the out-of-bag estimates of the loss function. This average is gotten as follows: if $\hat{f}^{*b}(x_i)$ is the predicted value at x_i, from the model fitted to the bth bootstrap data set (b in {1,. . . ,B}), our estimate is [24]

$$\widehat{Err}_{\text{boot}} = \frac{1}{B}\frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}^{*b}(x_i))$$

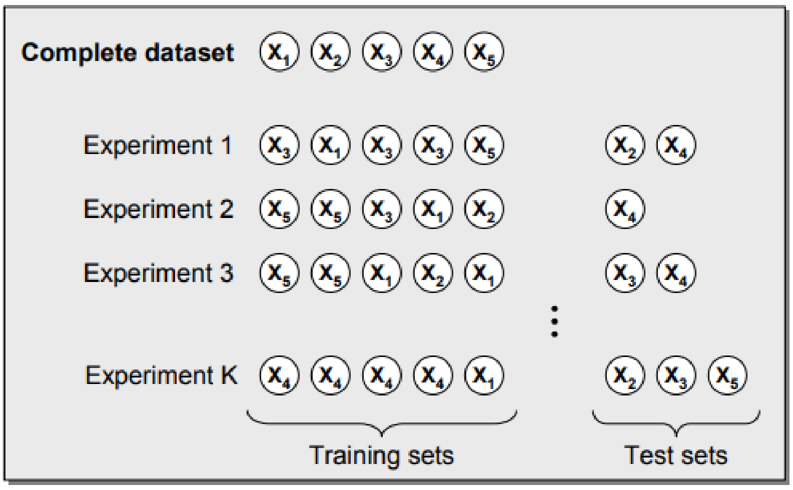


Figure 2.8: Bootstraping validation technique

Ensemble Methods

there are mountains growing
beneath our feet
that cannot be contained
all we've endured
has prepared us for this
bring your hammers and fists
we have a glass ceiling to shatter

rupi kaur

Within the uses of Machine Learning models for prediction, a set of techniques that stands out and has enjoyed growing attention over the last two decades is Ensemble Methods. So far, we have introduced many individual models for prediction regarding classification and regression problems in supervised learning. After building many models, we can wonder if it is possible to combine them to produce a better predictor. Going back to what was introduced in the previous chapter, the way to proceed in ensemble learning is to build a final model by combining the strengths of a collection of simpler base models [35]. As one can see, the idea behind this technique is nothing new to us; as humans, we use this method in in our day to day whenever we seek the opinions of different people or even experts when we have to make a decision. For example, reading user reviews before starting to watch a new series or consulting with several doctors before agreeing to take a detrimental medical treatment, are examples of ensemble-based decision making.

When using ensemble techniques, the final decision for each new test sample instance is made based on a (typically weighted but also unweighted) vote of the predictions obtained from the base models. Ensemble systems have proven themselves to be very effective and extremely versatile in a broad spectrum of problem domains and real-world applications [38]. The global discussion on climate change - probably one of the greatest concerns of our time and hence, the motivation of this project - is assessed at The Intergovernmental Panel on Climate Change (http://www.ipcc-data.org/), which has been using ensemble models for sustainable decision-making for over a decade already. Another application to mention when referring to ecology is forecasting of the species niche for endemic subspecies, such as Snow Crab (*Chionoecetes opilio*) [3].

The beginning of the development of ensemble systems dates back to 1979, with one of the first researches on ensemble learning, Dasarathy and Sheela's work on composite classifier system design [39], which addresses the partition of the input space using more than one classifier. Shortly after, in 1990, Hansen and Salamon concluded that an ensemble of similarly configured artificial neural networks can improve the generalization performance of the network [40]. Throughout the same year, Schapire proved that a strong classifier on a binary classification problem can be built by merging weak classifiers, in a procedure he named *Boosting* [41], the precursor of the algorithms which extended this concept to multiple class and regression problems, the so-called *AdaBoost* algorithms. Due to the undeniable success of these algorithms, many researches in this field have been carried out since then, resulting in new ensemble-based algorithms: stacked generalization and bagging (with random forests as a particular case), among others.

As mentioned above, ensemble methods usually build better predictors than simple algorithms, this is, we get to minimize the prediction error. To address the statistical motivation for ensembles, it is necessary to insert an aside about this error before continuing on the subject. In the previous section we assumed a model $Y = f(X) + \varepsilon$, where X and ε are independent random variables. The latter is the inherent noise in the problem that has zero mean and variance σ_ε^2. We can derive an expression for the expected prediction error of a regression fit $\hat{f}(X)$ at a *fixed* input point $X = x_0$, using squared-error loss [35]. This expectation is taken over all possible training sets $\mathcal{T}$ - because datasets are always of finite size, we may think of $\mathcal{T}$ as a random vector that follows the joint distribution $Pr(X,Y)$ - for when we change training sets we will change the value of $\hat{f}(x_0)$ since $\hat{f}$ is learned from the dataset $\mathcal{T}$.

$$\begin{aligned}
Err(x_0) &= E_{\mathcal{T}}\left[(Y - \hat{f}(x_0))^2 | X = x_0\right] = E_{\mathcal{T}}\left[(f(x_0) + \varepsilon - \hat{f}(x_0))^2\right] \\
&= E_{\mathcal{T}}\left[(f(x_0) - \hat{f}(x_0))^2 + \varepsilon^2 + 2 \cdot \varepsilon \cdot (f(x_0) - \hat{f}(x_0))\right] \\
&= E_{\mathcal{T}}\left[(f(x_0) - \hat{f}(x_0))^2\right] + \sigma_\varepsilon^2 \\
&= E_{\mathcal{T}}\left[(f(x_0) - E_{\mathcal{T}}\left[\hat{f}(x_0)\right] + E_{\mathcal{T}}\left[\hat{f}(x_0)\right] - \hat{f}(x_0))^2\right] + \sigma_\varepsilon^2 \\
&= E_{\mathcal{T}}\left[(f(x_0) - E_{\mathcal{T}}\left[\hat{f}(x_0)\right])^2 + (E_{\mathcal{T}}\left[\hat{f}(x_0)\right] - \hat{f}(x_0))^2 + \right. \\
&\left. +2 \cdot (f(x_0) - E_{\mathcal{T}}\left[\hat{f}(x_0)\right]) \cdot (E_{\mathcal{T}}\left[\hat{f}(x_0)\right] - \hat{f}(x_0))\right] + \sigma_\varepsilon^2 \\
&= \left(f(x_0) - E_{\mathcal{T}}\left[\hat{f}(x_0)\right]\right)^2 + E_{\mathcal{T}}\left[(E_{\mathcal{T}}\left[\hat{f}(x_0)\right] - \hat{f}(x_0))^2\right] + \sigma_\varepsilon^2 \\
&= \left(f(x_0) - E_{\mathcal{T}}\left[\hat{f}(x_0)\right]\right)^2 + Var_{\mathcal{T}}\left(\hat{f}(x_0)\right) + \sigma_\varepsilon^2 \\
&= ModelBias^2 + ModelVariance + IrreducibleError
\end{aligned}$$

In this way, the error has two components that we can control: bias, the accuracy of our estimate, and variance, the precision of the predictor when trained on different samples. The bias-variance tradeoff goes like this: models with low bias have a higher variance across samples, and vice versa. When our model suffers from high bias, the average response of the model is far from the true value and we call this *underfitting*. On the contrary, when the variance is high, this is usually a result of its inability to generalize well beyond the training data and we call this *overfitting*. In addition, another fact on which we base on is that averaging smooths (reduces variance) our model. Hence, the goal of ensemble systems is to create several classifiers with relatively fixed (or similar) bias and then combining their outputs, say by averaging, to reduce the variance [38].

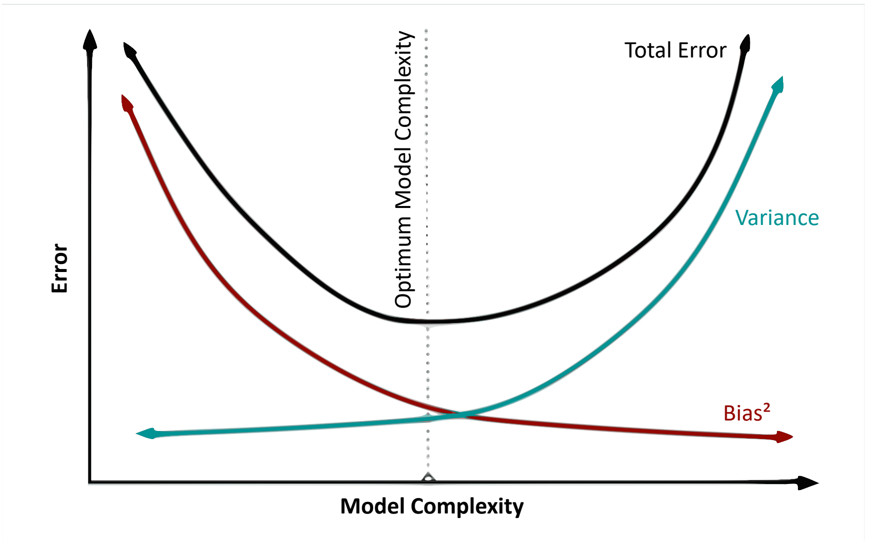


Figure 3.1: Bias - Variance tradeoff

Note that it is not always guaranteed to obtain a better prediction model when combining the classifier. Rather, the probability of choosing a poor estimate is reduced. Let us explain in depth the reason for the reduction of this probability. Hansen and Salamon pointed out in their work [40] that, in order to have an ensemble of classifiers which improves the generalization performance, this is, with the ensemble we provide more accurate classifications compared to its individual base models, we need these base members to be accurate and diverse. Respectively, these properties refers to a model that has an error rate of better than random guessing and, if we compare two of them, they make different errors on new data. To exemplify this improvement, let us consider an ensemble of three classifiers, $\{h_1, h_2, h_3\}$, and a new data point $\mathbf{x}$. If these classifiers are not diverse, then if one is wrong when predicting on $\mathbf{x}$, then the other two will also give an incorrect classification. Nevertheless, if the errors are uncorrelated, then when, for example, $h_1(\mathbf{x})$ is wrong, $h_2(\mathbf{x})$ and $h_3(\mathbf{x})$ may be correct, so that a majority vote will correctly classify $\mathbf{x}$. If we denote for p_1, p_2 and p_3 the error rates of these classifiers, the expression for the consensus error rate (we consider an error if and only if at least two of the participating classifiers make an error regardless of whether their erroneous classification coincide) is $p_1p_2p_3 + p_1p_2(1-p_3) + p_1(1-p_2)p_3 + (1-p_1)p_2p_3$. More precisely, if the error rates are all equal to $p < 0.5$, we can see that it does pay to use consensus classification. However, with error rates exceeding 0.5, the error rate of the ensemble will increase as a result of the voting. Hence, one key to successful ensemble methods is to construct individual classifiers with error rates below 0.5 whose errors are at least somewhat uncorrelated [42].

The guidelines from the previous paragraphs can be harnessed to produce a better model when addressing ensemble learning, which is done after developing the base models from the training data and consists mainly in one task: combining them in some of the many existing ways to form the composite predictor. The mechanism chosen to combine the individual models depends very much on the type of algorithm used in the ensemble members construction. In the case of classifiers, the output are discrete-valued labels and the most used combination rule for such outputs is (simple or weighted) majority voting. When continuous outputs, we have more rules at our disposal, such as arithmetic (sum, product, mean, ...) combiners. Even thought usually we can apply these combiners immediately after we obtain the base members, more complex combination algorithms may require an additional training step (as used in stacked generalization) [38]. We now describe these combination rules according to the type of output.

- **Combining class labels**: In the case of considering class labels, let us define T as the number of classifiers, C as the number of classes and so, $d_{t,c} \in \{0,1\}$ the decision of the t^{th} classifier on class ω_c ($t = 1,\dots,T$; $c = 1,\dots,C$). If the t^{th} classifier, h_t, chooses class ω_c, then $d_{t,c} = 1$, and 0 alternatively.

$$d_{t,c} = \begin{cases} 1, & \text{if } h_t \text{ chooses } \omega_c \\ 0, & \text{otherwise} \end{cases}$$

  Majority voting

  Among the most commonly used ensemble learning methods we find majority voting, in which each of the classifiers involved makes an independent prediction and it selects as consensus the one that has been chosen by the majority. We can highlight two families in the majority voting methods: *voting*, which uses models of different families (k-nn, SVM, etc.) with the same data set to obtain the metaclassifier, and *bagging*, which trains the same family of models with different data sets, expecting that each one specializes in different data. Majority voting has three strands subjected to how the ensemble decision is made: *unanimous voting*, if the decision is the class on which all classifiers agree, *simple majority*, if the decision is the class predicted by at least one more than half the number of classifiers, and *plurality voting*, when the decision is the class that receives the highest number of votes. When the strand is not specified, majority voting usually refers to plurality voting, which can be mathematically defined as follows: choose class ω_{c^*} where $c^* = \arg\max_c \sum_{t=1}^{T} d_{t,c}$.

Weighted Majority Voting

If we believe that some classifiers are more accurate than others, putting a heavy weight on the decision of those classifiers can improve the overall performance of our metaclassifier. But if we knew, a priori, which classifiers will predict better, we would only use those classifiers. As we lack of such information, we can use the performance of the classifiers on a separate validation dataset as an estimate of the generalization performance of each classifier. We then assign a weight w_t to classifier h_t proportional to its estimated generalization performance, commonly in a way that voting weights are normalized such that they add up to 1. Mathematically, we can define the procedure as follows: choose class ω_{c^*} where $c^* = \arg\max_c \sum_{t=1}^{T} w_t \cdot d_{t,c}$.

Borda Count

Borda Count is a decision method where the participants express their votes through an ordering of the options. It owes its name to the French mathematician Jean-Charles de Borda, who devised it in 1770. Bear in mind that there are several variants of the method. One of them is with the ordering done as follows: if there are C classes, the first class receives $C-1$ votes, the class with the second highest support receives $C-2$ votes, and so on until assigning 0 votes to the case ordered in the last position. Borda Count takes into account the consensus among the classifiers, in a way that if all the classifiers position a class near the winning class, then its Borda number (the corresponding vote) will be high so it will be among the first positions in the final ordering; on the other hand, if a single classifier is the one that puts a certain class in a privileged position, it doesn't mean necessarily that it will remain in a high position in the final rearrangement. This method treats all classifiers equally, which may not be desirable and one solution could be to add weights that express the quality of the classifier.

- **Combining continuous outputs**: In the case of combining continuous outputs, let T be the number of base members and so, $o_t(\mathbf{x})$ the prediction output of the t^{th} member ($t = 1, \ldots, T$) on the instance $\mathbf{x}$.

  Algebraic Combiners

  In algebraic combiners, the final output is obtained as a simple algebraic function of the predictions made by individual classifiers. There are several combination functions,

  Mean Rule: The final output is the average of all the individual ones, this is, $o_f(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} o_t(\mathbf{x})$.

  Trimmed mean: Sometimes predictors may erroneously give unusually low or high values such that the correct predictions of other members are not enough to undo the damage done by this unusual output. Trimmed mean avoids this problem by discarding the highest and lowest predictions before calculating the mean. For a R% trimmed mean, R% of the ordered values from each end is removed, with the mean calculated on the remaining ones.

Minimum/Maximum/Median Rule: These functions simply take the minimum, the maximum, or the median among the predictions from the base models.

Generalized Mean: All of the rules mentioned above are special cases of the generalized mean, defined as $o_f(\mathbf{x}) = \left(\frac{1}{T}\sum_{t=1}^{T} o_t^{\alpha}(\mathbf{x})\right)^{1/\alpha}$. Different choices of α lead to different combination rules. For instance, $\alpha \to -\infty$ leads to minimum rule, for $\alpha \to 1$ we get the mean rule, and $\alpha \to 0$ leads to $o_f(\mathbf{x}) = \left(\prod_{t=1}^{T} o_t(\mathbf{x})\right)^{1/T}$, the geometric mean.

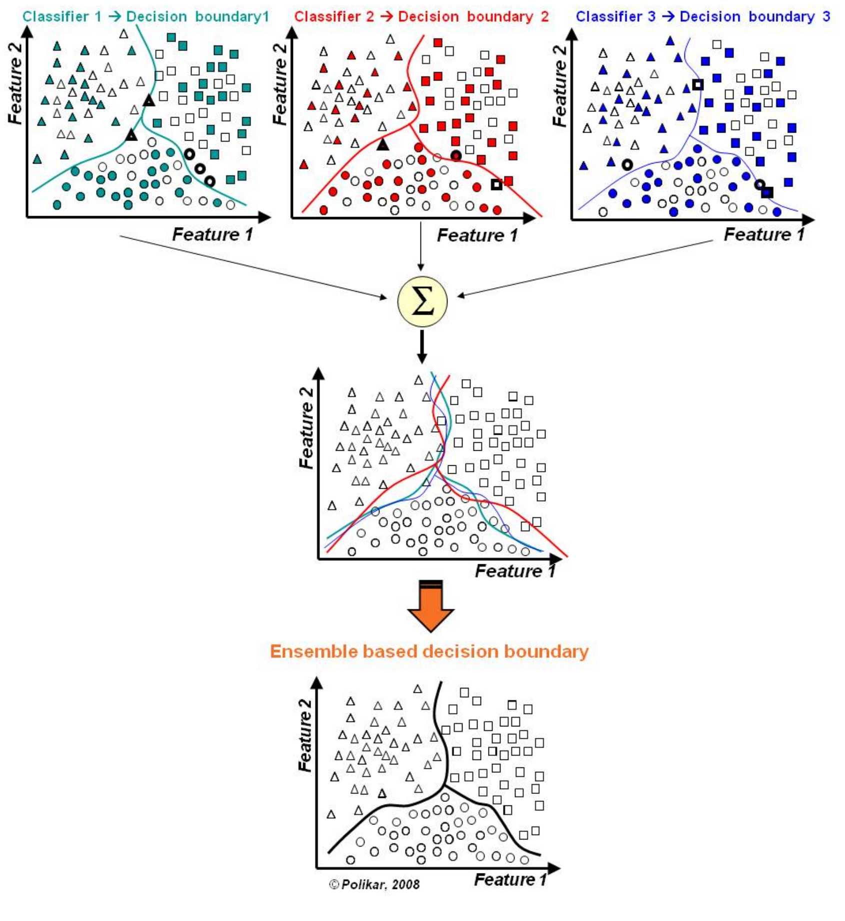

Classifier 1 → Decision boundary1
Classifier 2 → Decision boundary 2
Classifier 3 → Decision boundary 3
Feature 2
Feature 1
Σ
Ensemble based decision boundary
©Polikar, 2008

Figure 3.2: Ensemble Learning, example in classification

Once that we have provided a background on ensemble systems, in the following sections we present an overview of some of the most prominent ensemble-based algorithms.

3.1 Stacking

Also known as *stacked generalization*, it is a technique whose purpose is to achieve a generalization accuracy which is as high as possible [43]. We find two versions of stacking depending on the number of base models under consideration: when one has only a single predictor, stacking is used as a technique to improve it; but its main implementation (and the one considered in this section) is for combining many predictors. In this technique, proposed by Wolpert in 1992, we come across a different way of combining multiple base models through the concept of *meta learner*. This concept refers to a combiner system which, unlike voting, can be a nonlinear combination of the outputs of the base models.

This algorithm belongs to the heterogeneous aggregation methods. An important factor when creating the stacked ensembles is to choose a diverse variety of base models, due to each one should contribute somehow to the final meta learner. In order to add information not already in known and keep clear of redundancies, one should avoid models that are simply variations of one another. The algorithm goes as follows:

- Before starting stacked learning, we need to choose a quantity M of algorithms that we want to combine: $L_1, \ldots, L_M$. And we denote by L the learning algorithm for our meta model.

- Let $\mathscr{D} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ denote our dataset. In the first stage, to build the so-called *level-0* models or *generalizers*, we apply the level-0 learning algorithms to the original data set randomly splitted using cross-validation to train and test the upcoming models. This is, $h_m = L_m(\mathscr{D})$ for $m = 1, \ldots, M$.

- For each X_i of the level-0 test samples, we use each generalizer to predict their response variable. We denote then the output by $z_{mi} = h_m(X_i)$.

- These outputs, along with their true values Y_i, form a prediction vector with $M+1$ coordinates $(Y_i, z_{1,i}, \ldots, z_{M,i})$. For each instance on the test sample, these vectors are collected into a new dataset $\mathscr{D}'$, the *level-1* data.

- The latter allows training and testing L, the level-1 learning algorithm. We apply it to the new dataset in the same way as before. In notation, $h = L(\mathscr{D}')$.

- To predict on a new data, the obtained generalizers $h_1, \ldots, h_M$ and h are used. The final prediction on a new data x is:

$$f(x) = h(h_1(x), \ldots, h_M(x))$$

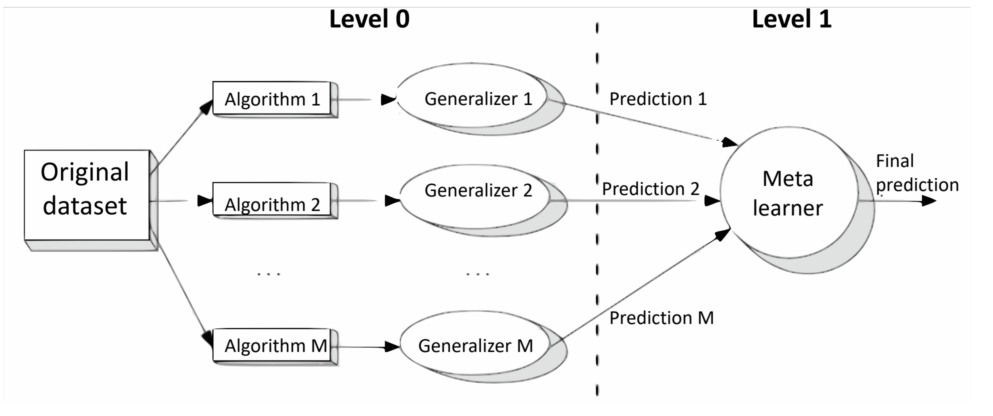


Figure 3.3: Stacked generalization: algorithm

The stacked generalization framework is quite flexible, allowing us to play around with some architectures. A settings that may help improve a stacked ensemble performance is *restacking*: we pass the data set unchanged from one layer to the other. This may improve the stacked ensemble performance in some cases, especially for more complicated ensembles with multiple layers. The intuition behind this model is derived from the fact that the higher level algorithm has extracted information from the input data, but rescanning the input space may yield new information that is not obvious from the first passes. The scheme for restacked generalization can be exemplified with the following graphic:

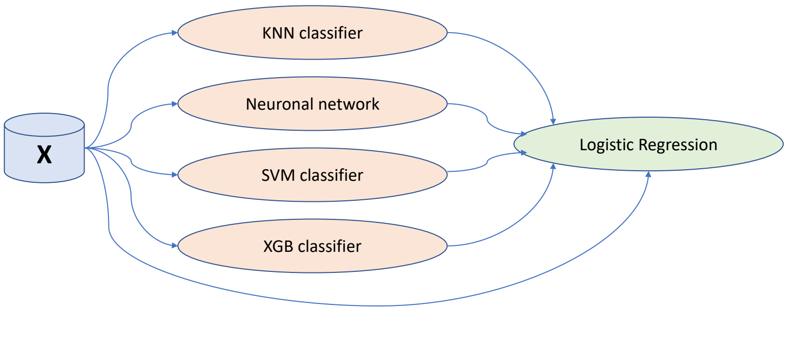


Figure 3.4: Restacking example

If we put in balance our trained stacked combiner with a fixed rule, such as voting or any of the others described at the beginning of this chapter, we find disadvantages and points in favor of both of them: the flexibility in stacked generalization framework that has just been discussed above may provide less bias, but adds extra parameters and risks introducing variance [11]; meanwhile with a fixed rule it is possible that we have lower accuracy, but it does not need extra time and data for training.

As application of this algorithm, we find the R package "SSDM" [44] to predict distribution of species richness and composition based on a stacked generalization of different species distribution models [45]. Stacked species distribution models (SSDMs) combine multiple individual species distribution models (SDMs) to produce a community-level model. An SDM (also referred as to "ecological niche model", "habitat suitability model" and "predictive habitat distribution models") refers to the process of using a statistical method to predict the distribution of a species in geographical space on the basis of a mathematical representation of its known distribution in environmental space [46]. A major strength to highlight of an SSDM is that it can predict species assemblages.

3.2 Boosting

The *Boosting* algorithm is based on creating each base model sequentially, instead of independently. The subsequent predictors "manipulate" the training instances by assigning a heavier weight to the ones that were previously predicted in a wrong way, while the weights of the instances that are correctly classified remain the same. This is, with boosting we actively try to generate complementary base learners.

The original boosting procedure by Schapire (1990) combined three weak learners to generate a strong learner. A weak learner is capable of producing predictors with probability of error strictly - but only slightly - less than that of random guessing (0.5 in the binary case); and a strong learner is able to yield models with arbitrarily small error probability. This initial version of boosting consists in, given a large training set $\mathscr{X}$, randomly dividing it into three. We use $\mathscr{X}_1$ to train the first model, h_1. We then use the second set $\mathscr{X}_2$ to feed it to h_1 and to train h_2. This is, we train the second model with all the instances misclassified by h_1 but also the ones on which h_1 is correct from $\mathscr{X}_2$. We then feed h_1 and h_2 with the last set, $\mathscr{X}_3$. Finally, the training set of h_3 are the instances on which h_1 and h_2 disagree (here remains the idea of building complementary models). The ensembled decision is as follows: given an instance, we give it to h_1 and h_2; if they agree, that is the response, otherwise the response of h_3 is taken as the output [11]. This system can also be used recursively, by executing a boosting algorithm of three models used as h_j in a higher system.

The main downside we can highlight when approaching this method is that it requires a very large training sample to be divided into three. In addition, the third predictor is only trained on a subset formed by the instances that were mispredicted by the other two.

Once again returning to the applications in ecology, ensemble algorithms such as boosting are of great help in the study of vulnerable species, especially when little is known regarding their habitats, which loss is the primary cause of their diminish. We can give as an example the use of boosting to infer stopover habitat selection and distribution of Hooded Cranes (*Grus monacha*) during Spring Migration in Lindian, Northeast China [4]. For instance, for roosting site selection, 249 optimal trees were generated from 1000 possible trees and the ROC curve was studied for the knowledge of the model discrepancy (accuracy).

3.2.1 AdaBoost

As a variant, Freund and Schapire (1996) proposed the most popular boosting algorithm known as *AdaBoost*, short for *adaptive boosting*. The key idea behind AdaBoost is to use weighted versions of the same training data instead of randomly subsamples thereof. The same training set is repeatedly used and, for this reason, it does not need to be very large, unlike earlier boosting methods [38]. Furthermore, AdaBoost can combine an arbitrary number of base learners, not only three, but they should be simple so that they do not overfit.

The idea behind the weighted training sample is to modify the probabilities of drawing the instances as function of the error. Let us denote by p_{ij} the probability of drawing the instance pair $(X_i, Y_i) \in \mathscr{X}$ to train the j^{th} base learner. At the beginning, $p_{i1} = \frac{1}{n}$ $(i = 1, \ldots, n)$, which means the algorithm starts by building the first base learner training on the dataset with equal weights.
Let also be ε_j the error rate of the base learner h_j not on the original problem but on the dataset used at step j. Since we require that learners are weak, the error rate is compared against a threshold ϵ symbolizing the probability of error in random guessing, that is, $\varepsilon_j < \epsilon \forall j$. If this constraint is not met, we stop adding new base-learners.
We then define $\beta_j = \frac{\varepsilon_j}{1-\varepsilon_j} < 1$. If h_j mispredicts, we set $p_{i(j+1)} = p_{ij}$; otherwise we set $p_{i(j+1)} = \beta_j \cdot p_{ij}$. The effect of this assignment is that the probability of drawing a mispredicted instance increases, while the probability for a correctly predicted one decreases. We should normalize p_{ij} as they are probabilities so that they sum up to 1.
Once these settings are made, at each step $j+1$ we train the learner h_{j+1} by drawing a new sample of the same size, with replacement, from the original dataset according to these modified probabilities. Once training is done, AdaBoost has a weighted voting scheme. Given a new data point x, all the h_j decide, with weights proportional to their previous accuracies on the training set: $w_j = \log \frac{1}{\beta_j}$.

Notice that as each new learner focuses more on mispredicted instances by the preceding model, the base-learners are chosen to be simple and not accurate. If this were otherwise,

Algorithm 1 AdaBoost algorithm

```
1:  $n$ = number of instances
2:  $p_{ij}$ = P[drawing instance i to train base learner j]
3:  $\epsilon$ = probability of error in random guessing
4:  for $i = 1, \dots, n$ do
5:      $p_{i1} = \frac{1}{n}$
6:  end for
7:  for $j = 1, \dots, J$ do
8:      Obtain base learner $h_j(X)$ that minimizes the error rate $\varepsilon_j = \sum_{i:y_i \neq h_j(X_i)} p_{ij}$
9:      if $\varepsilon_j \geq \epsilon$ then
10:         EXIT
11:     end if
12:     $\beta_j := \frac{\varepsilon_j}{1-\varepsilon_j}$
13:     $p_{i(j+1)} := \beta_j \cdot p_{ij}$
14:     if $h_j$ mispredicts then
15:         $p_{i(j+1)} := p_{ij}$
16:     end if
17:     for $j = 1, \dots, J$ do
18:         Normalize the weights so that they sum up to 1 $p_{i(j+1)} := \frac{p_{i(j+1)}}{\sum_{k=1}^{n} p_{k(j+1)}}$
19:     end for
20:     Classifier weight calculation for the final classifier $\omega_j := \log \frac{1}{\beta_j}$
21: end for
22: Final classificator $H(X) = arg\max_c \sum_{j=1}^{J} \omega_j \cdot d_{jc}$ where $d_{jc}$ is the decision of learner
    $h_j$ on class $c$.
```

the upcoming training samples would contain only a few outliers and many noisy instances. On the other hand, a consequence of weighting the instances is that, after all the runs, the mispredicted ones assigned with heavier weights are "hard" patterns to learn; these patterns are probably outliers. This is a kind of side effect of AdaBoost, which can be used for outlier detection on a given training set [38].

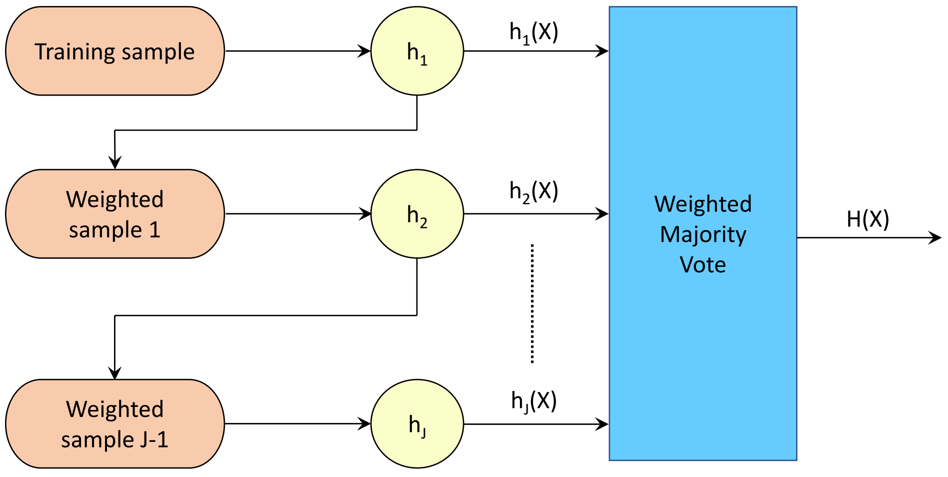


Figure 3.5: Graphical idea of the adaptative boosting algorithm

3.3 Bagging

Bagging, acronym for *Bootstrap Aggregating*, was proposed by Leo Breiman in 1994. It is a method for generating multiple versions of a predictor and using these to get an aggregated predictor [47]. These versions are trained over different resamples of the original dataset $\mathscr{X}$ with the same size generated via bootstrap, i.e., randomly drawing each instance with replacement. When this is done to generate B samples $\mathscr{X}_b$, $b = 1, \dots, B$, each of the independent observations follow the same underlying distribution as $\mathscr{X}$. After resampling is done, the base predictors h_b are trained with these B samples $\mathscr{X}_b$.

The key element is the instability of the learning algorithm. A learning algorithm is an unstable algorithm if small changes in the training set causes a large difference in the generated model, namely, the learning algorithm has high variance. Bagging trains the B base models using an unstable learning procedure [11], which can improve accuracy and push a significant step towards optimality. Some examples of unstable learning methods are decision trees or rule induction algorithms; on the contrary, k-nearest neighbor and naive Bayes classifiers are stable techniques, for which bagging may degrade the accuracy. Then, when combining the predictors during testing, it takes an average in the case of regression (or even the median to be more robust), or makes use of plurality vote when classifying the outcomes.

Note that because Bagging has no memory - each base model is fitted independently without any awareness of the other base-learners that have already been selected - it is easily parallelizable. This divisibility is a computational advantage of the algorithm [48].

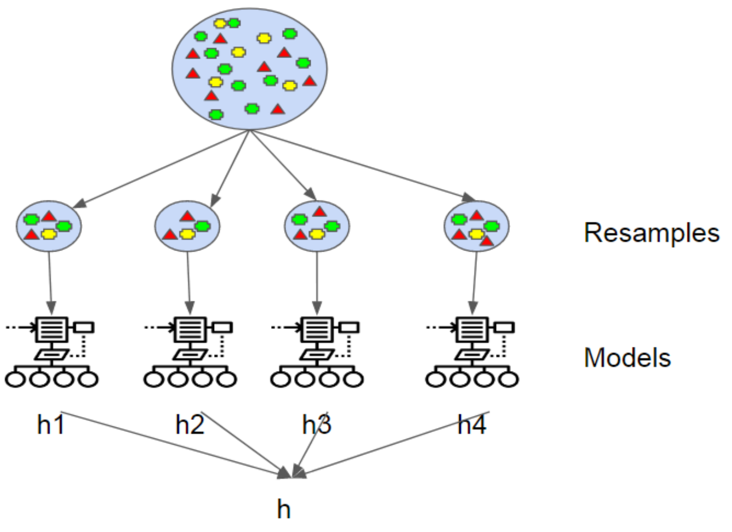


Figure 3.6: Bagging scheme

We now focus on the squared-error loss, $Err = (y - \hat{f}(x))^2$, to expound on the help of this method: bagging reduces the variance. Let the "idealized" bagging estimator be $\bar{h}(x) = E[h_b(x)]$. Using simple linear algebra and properties of the expectation (linearity and the inequality $E[X^2] \geq E^2[X]$), we can operate and obtain:

$$\begin{aligned} E[(y - h_b(x))^2] &= E[y^2 - 2 \cdot y \cdot h_b(x) + h_b(x)^2] \\ &= y^2 - 2 \cdot y \cdot E[h_b(x)] + E[h_b(x)^2] \\ &\geq y^2 - 2 \cdot y \cdot E[h_b(x)] + E^2[h_b(x)] \\ &\geq (y - E[h_b(x)])^2 \\ &\geq (y - \bar{h}(x))^2 \end{aligned}$$

We deduce then that the mean-squared error of $\bar{h}(x)$ is lower than the mean-squared error averaged of the predictors h_b trained over the bootstrap samples. This is, the aggregation never increases the squared error, and it often reduces it. How much lower depends on how unequal the two sides of $E^2[h_b(x)] \leq E[(h_b^2(x))]$ are. The effect of instability is clear. If the predictor does not change too much with the boostrap replicates, the two sides will be nearly equal, and aggregation will not help. So, the more highly variable the models are, the more improvement aggregation may produce [47]. Nevertheless, bagging a bad classifier can make the classification worse, as the above argument does not hold for classification because of the "non-additivity" of bias and variance.

Even though bagging has been applied frequently in fields such as biostatistics, its use is uncommon in the field of ecology. However, we can find papers in which it is used for environmental studies, such as the predictive vegetation mapping under current and future climate scenarios according to the Canadian Climate Centre global circulation model [5]. This technique was applied to four tree species common in the eastern United States: loblolly pine (*Pinus taeda*), sugar maple (*Acer saccharum*), American beech (*Fagus grandifolia*), and white oak (*Quercus alba*). Future estimates of suitable habitat after climate change were visually more reasonable with bagging trees and random forest (which will be described soon below), with slightly better performance by RF as assessed by Kappa statistics (described in section 2.3.1). Bagging trees and random forest modeling approaches are considered to be robust for predictive mapping and their inclusion in the ecological toolbox is recommended for better understanding.

3.3.1 Random Forests

Random Forests is a variant of bagging algorithm introduced by Leo Breiman [49] whose base models are decision trees. The reason for this choice is that bagging works well on unstable methods such as trees, which, being notoriously noisy procedures, benefit greatly from averaging. Moreover, they can capture complex interaction structures in the data and, if grown sufficiently deep, have relatively low bias. In addition, each tree in the random forest is fully grown and not pruned. Random forests tend to perform very well, especially for those datasets containing many features and, as decision trees, they can be used for either classification or regression problems.

Since each tree generated in bagging is identically distributed (i.d.), the expectation of an average of B such trees is the same as the expectation of any one of them. This means the bias of bagged trees is the same as that of the individual (bootstrap) trees, and the only hope of improvement is through variance reduction [35].
If the trees are independent and identically distributed (i.i.d.) - with σ^2 being the variance of these trees - the average of the B trees has variance

$$Var\left[\frac{1}{B}\sum_{i=1}^{B} T_i\right] = \frac{1}{B^2}\sum_{i=1}^{B} Var[T_i] = \frac{1}{B^2}B\sigma^2 = \frac{\sigma^2}{B}$$

But if they are simply i.d. - with variance σ^2, expectation μ and pairwise correlation ρ - the variance of the average is

$$\begin{aligned}
Var\left[\frac{1}{B}\sum_{i=1}^{B} T_i\right] &= \frac{1}{B^2} Var\left[\sum_{i=1}^{B} T_i\right] = \frac{1}{B^2}\left[E\left[\left(\sum_{i=1}^{B} T_i\right)^2\right] - E^2\left[\sum_{i=1}^{B} T_i\right]\right] \\
&= \frac{1}{B^2} E\left[\sum_{i=1}^{B}\sum_{j=1}^{B} T_i T_j\right] - \frac{1}{B^2}\left(\sum_{i=1}^{B} E[T_i]\right)^2 \\
&= \frac{1}{B^2} E\left[\sum_{i=1}^{B}\sum_{j=1}^{B} T_i T_j\right] - \frac{1}{B^2}(B\mu)^2 \\
&= \frac{1}{B^2} E\left[\sum_{i=1}^{B}\sum_{j=1}^{B} T_i T_j\right] - \mu^2
\end{aligned}$$

Using the expression of the correlation coefficient

$$\rho = \frac{E\left[(T_i - \mu)(T_j - \mu)\right]}{\sigma \cdot \sigma} = \frac{E[T_i T_j] - 2\mu^2 + \mu^2}{\sigma^2} = \frac{E[T_i T_j] - \mu^2}{\sigma^2}$$

we obtain that for $i \neq j$, $E[T_i T_j] = \sigma^2\rho + \mu^2$; and that $E[T_i^2] = \sigma^2 + \mu^2$.
Thus the variance of the average is now given by

$$\begin{aligned}
Var\left[\frac{1}{B}\sum_{i=1}^{B} T_i\right] &= \frac{1}{B^2} E\left[\sum_{i=1}^{B}\sum_{j=1}^{B} T_i T_j\right] - \mu^2 \\
&= \frac{1}{B^2}\left[B \cdot E[T_i^2] + (B^2 - B) \cdot E[T_i T_j]\right] - \mu^2 \\
&= \frac{1}{B^2}\left[B \cdot (\sigma^2 + \mu^2) + B(B-1) \cdot (\sigma^2\rho + \mu^2)\right] - \mu^2 \\
&= \frac{\sigma^2}{B} + \frac{\mu^2}{B} + (1 - \frac{1}{B})(\sigma^2\rho + \mu^2) \\
&= \rho \cdot \sigma^2 + \frac{1-\rho}{B} \cdot \sigma^2
\end{aligned}$$

Note that we need the correlation ρ to be positive. As we increase the size of the bootstrap B, the second term disappears; but the first one remains, and hence the amount of correlation limits the benefits of averaging.

The key in random forests is to upgrade the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance too much. This "perturbation" of the algorithm is carried out when constructing the base trees and is called *subset splitting*. The alteration consists in considering only a random subset of the input variables at each node when building a tree, instead of successively finding the best split at each node by considering every possible variable.

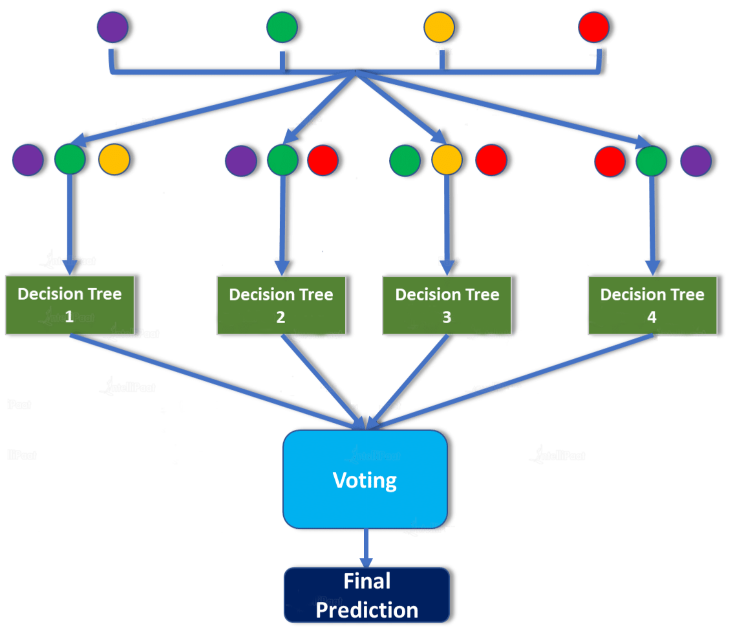


Figure 3.7: Random Forests scheme illustrating input variables selection

From a computational standpoint, Random Forests are appealing because they depend only on a few tuning parameters [38]. These are

- m, the number of randomly selected predictor variables chosen at each node.
- J, the number of trees in the forest.
- *tree size*, as measured by the smallest node size for splitting or the maximum number of terminal nodes.

The only one of these parameters to which the algorithm is somewhat sensitive appears to be m. In classification, the standard default is $m = \lfloor \sqrt{M} \rfloor$, where M is the total number of input variables. In regression, the default is $m = \lfloor \frac{n}{3} \rfloor$, where n is the sample size.
Regarding the number of trees, the out-of-bag estimate can be unstable and inaccurate for small values of J. Even so, the generalization error for forests converges almost surely to a limit as the number of trees in the forest becomes large. This result explains why random forests do not overfit as more trees are added, but produce a limiting value of the generalization error [49].
Finally, as for the size of the tree, originally it was recommended to grow very large trees. Nevertheless, in recent papers some examples have been given for which forests of large trees overfit. In those cases, out-of-bag error rates can be helpful to tune either the number of nodes or the smallest node size.

Application: prediction model about ecology

No one is too small to make a difference.

Greta Thunberg

4.1 Building Ensemble Algorithms with R

The R Project was initially developed by Robert Gentleman and Ross Ihaka, from the Department of Statistics, at the University of Auckland in 1993. R [6] is both a programming language and a free software environment for advanced statistical analysis with high quality graphics. It is available for most computer platforms and can be downloaded from CRAN at `cran.r-project.org`.

This programming language stands out as one of the most used in scientific research, being also very popular in the fields of machine learning, data mining, biomedicine and financial mathematics. This usefulness is greatly enhanced by the possibility of loading extension packages that enrinch and complement the base R software. Each of them has a specific purpose, for example, to enhace graphs or add calculations that extend its basic configuration. The download from CRAN of those which don't ship with the installation can be easily done within R. For example, `caret` [50] or `caretEnsemble` [51] will be useful to carry out the implementation of this Final Degree Project and their contents will be described later.

This section presents a review of how to create three of the most powerful types of ensembles in R: Stacking, Boosting and Bagging. We can summarize these techniques (already presented in sections 3.1, 3.2 and 3.3, respectively) in a few lines:

- Stacking: building multiple models (better if of differing types) and a meta-model that learns how to best combine the predictions of the base ones.
- Boosting: building multiple models from the same learner, each of which learns to fix the prediction errors of a prior model in the chain.

- Bagging: building multiple versions of a model from different bootstrap subsamples of the training dataset and using these to get an aggregated predictor.

We will look at each in turn. Examples will not be given in this section as their application will be shown in the software implementation one.

The first R package introduced is `caret` [50] (short for Classification And REgression Training), which consists of a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for data splitting, feature selection, model tuning using resampling, among other functionality. Related to this one, we also bring up the `caretEnsemble` [51] package for making ensembles of caret models.

Stacking Algorithms

For building stacking algorithms we use the helpful `caretList()` function for creating a list of standard caret models. Its argument `methodList` determines the caret models to ensemble, in a character vector form; also, we can pass to its argument `trControl` the function `trainControl()` specifying the train method for the base learners. Finally, the function `resamples()` applied to this list provides methods for analysing and visualizing (for example with the help of `summary()` and `dotplot()`) the set of results for each model applied to the common data set.
When we combine the predictions of different models using stacking, we already explained that it is desirable that the predictions made by the base models have low correlation, as this would suggest that they are skillful in different ways. Otherwise, if the predictions were highly corrected then they would be making very similar predictions, reducing the benefit of combining the predictions through the meta learner. This correlation can be checked with `modelCor()`, giving the correlation coefficients matrix, or `splom()`, which draws conditional scatter plots of these matrices.
Given the list of caret models, the `caretStack()` function can be used to specify the meta model to learn how to best combine the predictions of the base models together, which are given in its argument `all.models`. We can also type additional arguments to pass to the optimization function such as `metric`, choosing for example `"Accuracy"`, or `trControl`, as before. Via function `print()` we can analyse the results.

Boosting Algorithms

The most popular boosting algorithms when implementing in R are:

- AdaBoost Classification Trees (which uses decision trees as weak classifiers).
- Iterating boosting using the tree building algorithm C5.0 for the base models.

- Gradiente Boost algorithms, from which one can highlight Stochastic Gradient Boosting and eXtreme Gradient Boosting.

The chosen method is specified in the argument `method` (`"adaboost"`, `"C5.0"`, `"gbm"` and `"xgbDART"`, respectively) of the function `train()`, which fits predictive models over different tuning parameters. As in stacking algorithms construction, we can also pass the additional arguments to this function and results are displayed making use of `resamples()` via `summary()` and `dotplot()` in the same way.
Instead of using this function, whose functionality is actually based on loading the package in which the chosen method is defined and setting up a grid of its tuning parameters, we can directly use all the functions with which these packages complement R. For example, with the aid of package `gbm` [52] (short for Generalized Boosted Modeling), we can get efficient boosted trees by tuning the parameters of the function `gbm()`, such as `n.trees`, the total number of trees to fit; `shrinkage`, also known as the learning rate or step-size reduction, is the parameter applied to each tree in the expansion (smaller learning rates typically require more trees); `interaction.depth`, the maximum depth of each tree (i.e., the highest level of variable interactions allowed); or `n.minobsinnode`, the minimum number of observations in the terminal nodes of the trees. This package is also very useful when exploring the results: `gbm.perf()` estimates the optimal number of boosting iterations for a `gbm` object and plots various performance measures; `pretty.gbm.tree()` extracts the information from a single tree stored in the `gbm` object and displays it in a readable form. Also, by applying `summary()`, a plot of the relative influence of each variable is given.

Bagging Algorithms

When implementing bagging algorithms, two of the most outstanding methods are:

- Bagged CART
- Random Forest

As mentioned before, the function `train()` makes implementing ensemble methods very simple. For bagging it is the same as for boosting, we just have to change the specification of the training method (`"treebag"` and `"rf"`, respectively).
By model averaging, bagging helps to reduce variance and minimize overfitting. Although it can be used with any base learner, it is usually applied to decision trees [53]. Another way of building these bagging models is with the aid of packages `rpart` [54] (short for Recursive Partitioning And Regression Trees) to fit the decision trees, and `ipred` [55] (short for Improved PREDictors), to fit bagged decision trees. The function `bagging()` makes the prediction according to its atributes `formula`, which establishes the formula like `response ~`

`predictors`; `nbagg`, an integer giving the number of bootstrap replications; `coob`, a logical indicating whether an out-of-bag estimate of the error rate should be computed; and `control`, where options that control details of the `rpart` algorithm must be given. The value of the last attribute must be provided through the function `rpart.control`, indicating `minsplit`, the minimum number of observations that must exist in a node in order for a split to be attempted, among other parameters. Moreover, when using decision trees as base model, we can upgrade the variance reduction via subset splitting with a particular bagging algorithm, Random Forests. The corresponding R package is `randomForest` [56], whose main function `randomForest()` have several hyperparameters that can be tuned, although the default values tend to produce good results. Some of them are `ntree`, the number of trees in the forest; `mtry`, the number of features to consider at any given split; or `nodesize`, the minimum size of terminal nodes. Another useful function in this package is `importance()`, which gives measures of the importance of each variable in building the model.

4.2 Approaching the problem

The concern about the existence of physical limits to the consumption of natural resources by human societies is not new, it was already raised by the classical economists of the 19th century (e.g. Malthus). Nevertheless, sustainability has now become a universal policy goal - at least in the official rhetoric.

The ***Ecological Footprint*** [7] is the measure of the demand on and supply of nature. It is represented by the area that is necessary to produce the resources and cover the impacts of human activities.
On the demand side, the Ecological Footprint measures the ecological assets that a given population requires to produce the natural resources it consumes (including plant-based food and fiber products, livestock and fish products, timber and other forest products, space for urban infrastructure) and to absorb its waste, especially carbon emissions.
On the supply side, the *biocapacity* represents the productive land, which includes cropland, grazing land, forest land, fishing grounds and built-up land. These areas, especially if left unharvested, can also absorb much of the waste we generate, particularly additional carbon dioxide emissions that the oceans cannot lessen.
Both the biocapacity and the ecological footprint are expressed in the same unit: global hectares (gha) - globally comparable, standardized hectares with world average productivity. One global hectare is the world's annual amount of biological production for human use and human waste assimilation, per hectare of biologically productive land and fisheries.
If the biocapacity of a population exceeds its Ecological Footprint, it has an *ecological reserve*. Otherwise, that region runs an *ecological deficit*: its demand for the goods and services that its land and seas can provide (food, wood, cotton for clothing and carbon dioxide absorption)

exceeds what the ecosystems can renew and emits carbon dioxide into the atmosphere.

The measurement and knowledge of this footprint help countries improve sustainability and well-being, and also let individuals understand their impact on the planet. "Sustainable" implies that the development at service of this aforementioned well-being must come at no cost to future generations.

The Ecological Footprint trails the use of six different productive surface areas: cropland, grazing land, forest area, fishing grounds, built-up or urban land, and carbon demand on land (as said before, measured in global hectares). When the carbon footprint is reported within this context, it represents the amount of productive land area required to get rid of the tonnes of carbon dioxide emissions a country lets off.

When trying to predict a country's ecological reserve or deficit, there are some other variables that might be taken under consideration. These are the population of the country (in millions of habitants), its GDP (gross domestic product) per capita (expressed in US Dollars) and Human Development Index (HDI). The latter is a statistic composite index of life expectancy, education, and gross national income indicators, which are used to rank countries. The countries that score highest on the HDI also contribute most, in per capita terms, to climate change and other forms of ecological breakdown [57]. On the contrary, poor and underdeveloped countries produce much less per capita damage to nature, but as they develop - and in this situation are China or India - the index increases implying unsustainability. Reports by the environmental group World Wildlife Fund (WWF) include graphs where the ecological footprint is plotted against the United Nations HDI. However, in 2006 and again in 2016, Cuba was associated to a good level of development according to the UN thanks to its high level of literacy and a fairly high life expectancy, while its ecological footprint is not large as it is a country with low energy consumption.

In our time, it is quite common to find Artificial Intelligence and Statistics hybrid techniques so, we could wonder, why not considering other procedures better than Ensemble Methods? Because of to this plurality, we have the freedom of choice to use the ones that work best (this is, give us the best predictions). Each ML algorithm certainly has its assets and liabilities and, consequently, choosing the right algorithm to employ could be quite a hard task to do. Fortunately, ensembles can help. Based on the researches that have been exposed as examples throughout these pages, the use of Ensemble techniques will change ecology management and decision making for the better. At a minimum, these algorithms can analyse vast amounts of data, and they do so in less time and with greater efficiency than hitherto possible using traditional statistical approaches [9]. All the above reasons explain why these techniques have been considered a reasonable way to proceed and an appealing topic for this Final Degree Project.

4.3 Software implementation

The database to be used has been provided by the Global Footprint Network [7]. The different R files that gather the process of exploiting the raw data until the final extraction of knowledge are shown below, along with their executions.

Preprocessing

The library chosen for the treatment of .xlsx files is xlsx. We load the data set and ask for some information to check if the variables in the data frame are of the correct type.

```
library(xlsx)
data = read.xlsx("countries.xlsx", sheetName = "Sheet1", header = TRUE)
attach(data)
str(data)

#change the type of GDP.per.Capita to numeric
GDP.per.Capita = as.numeric(as.character(GDP.per.Capita))
```

```
## 'data.frame':    188 obs. of  20 variables:
##  $ Country                       : chr  "Afghanistan" "Albania" "Algeria" "Angola" ...
##  $ Region                        : chr  "Middle East/Central Asia" "Northern/Eastern Europe" "Africa
##  $ Population                    : num  29.82 3.16 38.48 20.82 0.09 ...
##  $ HDI                           : num  0.46 0.73 0.73 0.52 0.78 0.83 0.73 NA 0.93 0.88 ...
##  $ GDP.per.Capita                : chr  "614.66" "4534.37" "5430.57" "4665.91" ...
##  $ Cropland.Footprint            : num  0.3 0.78 0.6 0.33 NA 0.78 0.74 NA 2.68 0.82 ...
##  $ Grazing.Footprint             : num  0.2 0.22 0.16 0.15 NA 0.79 0.18 NA 0.63 0.27 ...
##  $ Forest.Footprint              : num  0.08 0.25 0.17 0.12 NA 0.29 0.34 NA 0.89 0.63 ...
##  $ Fish.Footprint                : num  0 0.02 0.01 0.09 NA 0.1 0.01 NA 0.11 0.06 ...
##  $ Urban.Land                    : num  0.04 0.06 0.03 0.04 NA 0.1 0.07 NA 0.14 0.15 ...
##  $ Carbon.Footprint              : num  0.18 0.87 1.14 0.2 NA 1.08 0.89 NA 4.85 4.14 ...
##  $ Total.Ecological.Footprint    : num  0.79 2.21 2.12 0.93 NA ...
##  $ Cropland                      : num  0.24 0.55 0.24 0.2 NA 2.64 0.44 NA 5.42 0.71 ...
##  $ Grazing.Land                  : num  0.2 0.21 0.27 1.42 NA 1.86 0.26 NA 5.81 0.16 ...
##  $ Forest.Land                   : num  0.02 0.29 0.03 0.64 NA 0.66 0.1 NA 2.01 2.04 ...
##  $ Fishing.Water                 : num  0 0.07 0.01 0.26 NA 1.67 0.02 NA 3.19 0 ...
##  $ Total.Biocapacity             : num  0.5 1.18 0.59 2.55 0.94 ...
##  $ Biocapacity.Deficit.or.Reserve: num  -0.3 -1.03 -1.53 1.61 -4.44 ...
##  $ Earths.Required               : num  0.46 1.27 1.22 0.54 3.11 1.82 1.29 6.86 5.37 3.5 ...
##  $ Countries.Required            : num  1.6 1.87 3.61 0.37 5.7 ...
```

We create the response variable “Classification” as defined when approaching the problem, and collect it along the neccessary input variables. All of them are written in a new .xlsx file.

```
Classification = factor(c(), levels = c("ecological.deficit",
                                        "ecological.reserve"))
for (i in 1:length(Total.Ecological.Footprint)) {
  a = (Total.Ecological.Footprint/Total.Biocapacity)[i]
  if (!is.na(a)) {
    if (a > 1)
      {Classification[i] = "ecological.deficit"}
    else
      {Classification[i] = "ecological.reserve"}
  }
}

useful_data = data.frame(Country, Region, Population, HDI, GDP.per.Capita,
                         Cropland.Footprint, Grazing.Footprint, Forest.Footprint,
                         Fish.Footprint, Urban.Land, Carbon.Footprint, Cropland,
                         Grazing.Land, Forest.Land, Fishing.Water, Classification)

write.xlsx(useful_data, file ="TFGdataset.xlsx", row.names=FALSE)
```

Data Cleansing

We import the file created in the Preprocessing section.

```
library(xlsx)
data = read.xlsx("TFGdataset.xlsx", sheetName = "Sheet1", header = TRUE)
attach(data)
```

Detection and Localization of errors

Missing values

First thing is to check for missing values (NAs).

- The complete.cases function detects rows in a data.frame that do not contain any missing value. The resulting logical can be used to remove incomplete records from the data.frame.
- Alternatively the na.omit function, does the same. The result of the na.omit function is a data.frame where incomplete rows have been deleted. The row.names of the removed records are stored in an attribute called na.action.

```
head(complete.cases(data))
```

```
## [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
```

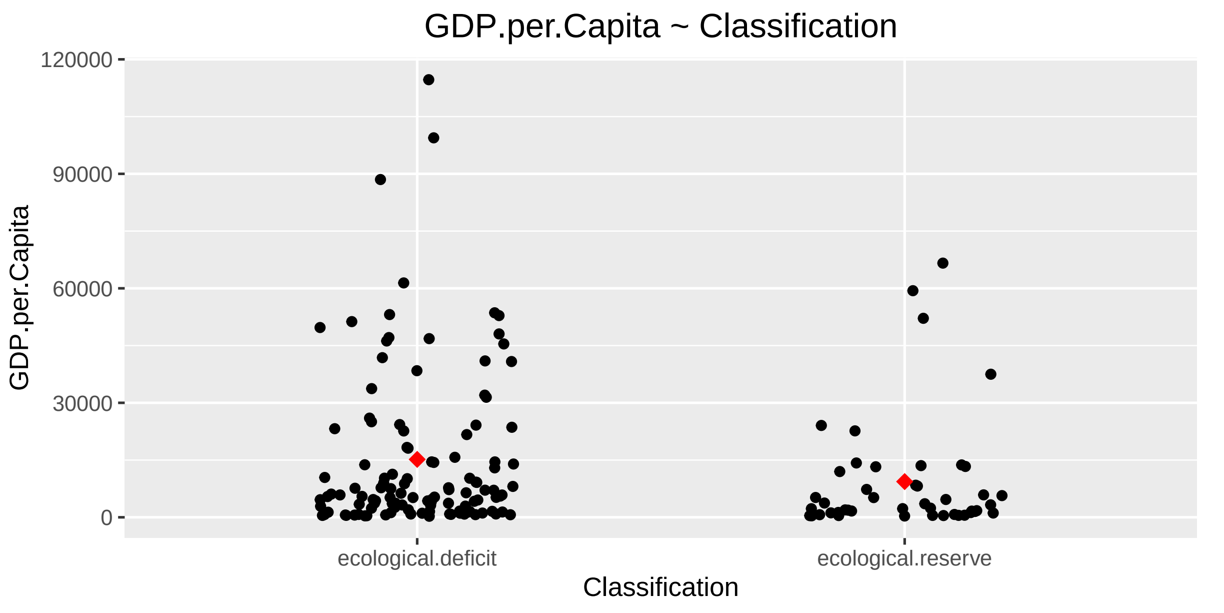
```
data_complete = na.omit(data)
head(na.action(data_complete))
```

```
##  5  8 19 25 30 31
##  5  8 19 25 30 31
```

Outliers

This subsection is in charge of detecting observations which appear to be inconsistent with the dataset.

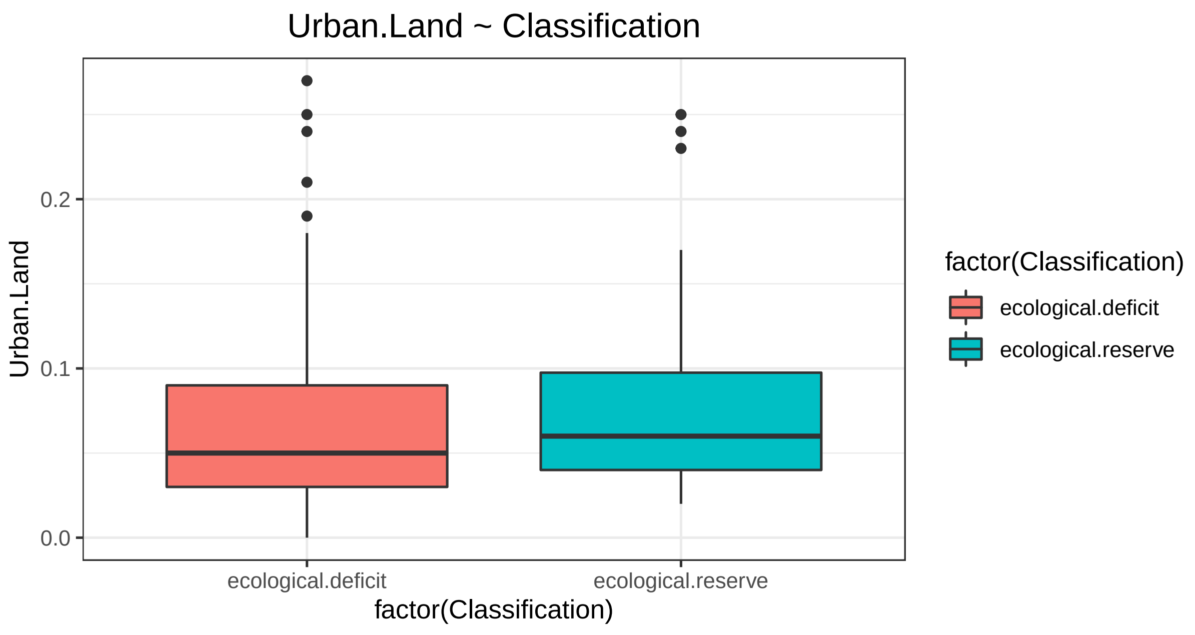
```
library(ggplot2)
ggplot(na.omit(data), aes(x = Classification,
                          y = GDP.per.Capita)) +
  geom_jitter(position=position_jitter(0.2)) +
  stat_summary(fun.y=mean, geom="point", shape=18,size=3, color="red") +
  labs(title = "GDP.per.Capita ~ Classification",
       caption = "Scatter plot of GDP.per.Capita depending on
\nthe category in Classification") +
  theme(
    plot.title = element_text(hjust = 0.5, size = 14),
    plot.caption = element_text(hjust = 0, face = "italic")
  )
```



Scatter plot of GDP.per.Capita depending on

the category in Classification

```r
ggplot(na.omit(data), aes(x = factor(Classification),
                          y = Urban.Land,
                          fill = factor(Classification))) +
  geom_boxplot() +
  theme_bw() +
  labs(title = "Urban.Land ~ Classification",
       caption = "Boxplot of Urban.Land for each \ncategory in Classification") +
  theme(
    plot.title = element_text(hjust = 0.5, size = 14),
    plot.caption = element_text(hjust = 0, face = "italic")
  )
```



Boxplot of Urban.Land for each category in Classification

Obvious inconsistencies

In this section, we detect records that contains a value or combination of values that cannot correspond to a real-world situation.

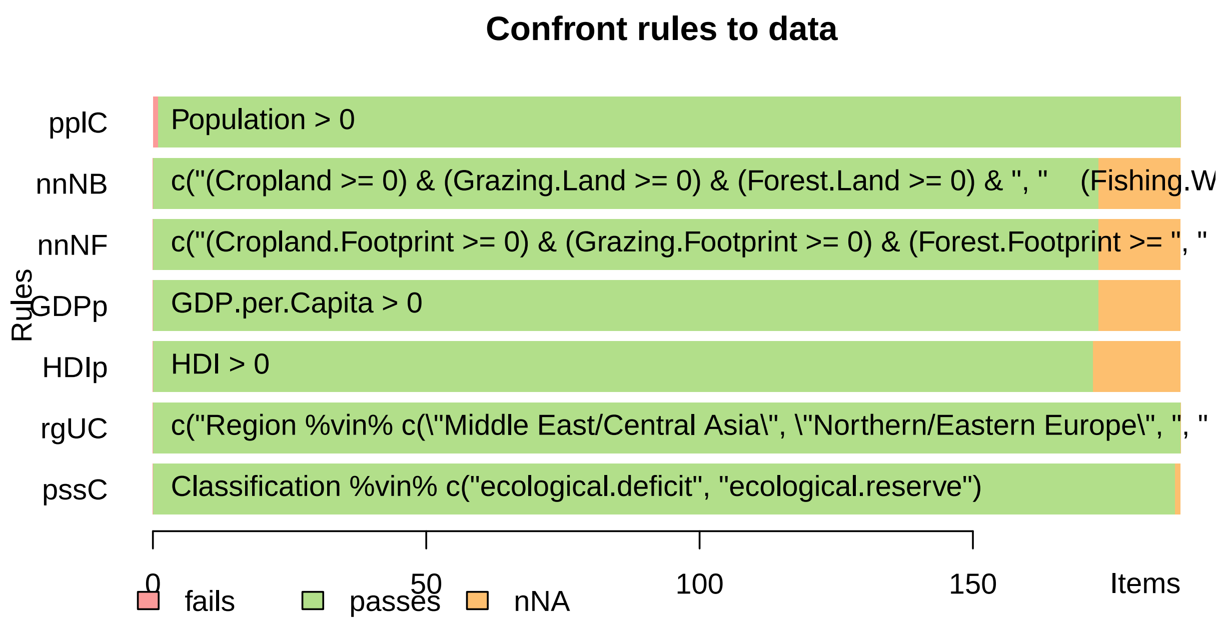
The edit rules are defined in a .yaml file which we can load and explore.

```r
library(data.table)
library(validate)

v = validator(.file='editrulesTFG.yaml')
for(rule in names(v)){
  print(v[[rule]])
}
```

```
##
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##  created    : 2020-05-25 16:34:05
##  meta       : language<chr>, severity<chr>
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##  created    : 2020-05-25 16:34:05
##  meta       : language<chr>, severity<chr>
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##  created    : 2020-05-25 16:34:05
##  meta       : language<chr>, severity<chr>
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##  created    : 2020-05-25 16:34:05
##  meta       : language<chr>, severity<chr>
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##  created    : 2020-05-25 16:34:05
##  meta       : language<chr>, severity<chr>
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##  created    : 2020-05-25 16:34:05
##  meta       : language<chr>, severity<chr>
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##  created    : 2020-05-25 16:34:05
##  meta       : language<chr>, severity<chr>
## Object of class rule.
##  expr       : Population > 0
##  name       : populatedCountries
##  label      : non deserted countries
##  description: It is obvious that every country has population.
##
##  origin     : editrulesTFG.yaml
##   created     : 2020-05-25 16:34:05
##   meta        : language<chr>, severity<chr>
## Object of class rule.
##   expr        : Population > 0
##   name        : populatedCountries
##   label       : non deserted countries
##   description: It is obvious that every country has population.
##
##   origin      : editrulesTFG.yaml
##   created     : 2020-05-25 16:34:05
##   meta        : language<chr>, severity<chr>
```

And we apply (confront) the rules to data. The results are easily seen with a barplot.

```r
cf = confront(data, v)

barplot(cf,
        main = "Confront rules to data",
        ylab = "Rules",
        xlab = "Items")
```



Imputation

In this process, we estimate or derive values for fields where data is missing. There is no one single best imputation method that works in all cases. The imputation model of choice depends on what auxiliary information is available and whether there are (multivariate) edit restrictions on the data to be imputed.

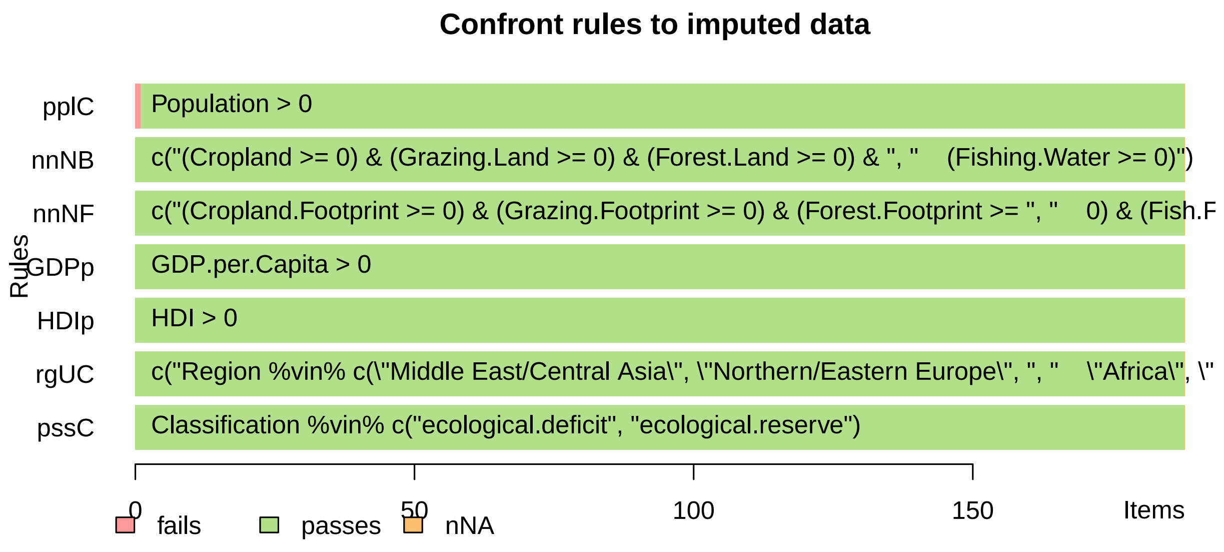
For our data, we use kNN-imputation. A distance function d(x_i, x_j) measures the dissimilarity between records. A missing value is then imputed by finding first the records nearest to the record with one or more missing values. Next, a value is chosen from or computed out of the nearest neighbors.

The VIM package contains a function called kNN that uses Gowers distance 12 (https://www.rdocumentation.org/packages/StatMatch/versions/1.2.5/topics/gower.dist) to determine the nearest neighbors.

```r
library(VIM)
data_knn = kNN(data)
```

We study again the inconsistency of the new complete dataset as before. We delete from our dataset the fails and then export it to a new file .xlsx that is finally ready to be trained.

```r
cf_new = confront(data_knn, v)
barplot(cf_new,
        main = "Confront rules to imputed data",
        ylab = "Rules",
        xlab = "Items")
```



```r
flags_new = as.data.table(values(cf_new))
data_knn = data_knn[flags_new$populatedCountries, 1:16]

write.xlsx(data_knn, file = "TFGdataset_complete.xlsx", row.names = FALSE)
```

Stacking

We applied preprocessing to raw data and tackled data cleansing. To carry out the modeling via stacking, we load the complete dataset we obtained from this last procedure and the libraries caret and caretEnsemble.

```
set.seed(15)

library(xlsx)
library(caret)
library(caretEnsemble)

dataset = read.xlsx(file = "TFGdataset_complete.xlsx", sheetName = "Sheet1")
attach(dataset)
nFeatures = dim(dataset)[2] - 3
```

First, we implement Monte Carlo Cross-Validation (MCCV, introduced in section 2.3.1) to obtain the test assessment by averaging the effectiveness measures in the sample re-uses. K is set to 10 and the train sample will contain the 80% of the dataset.

```
K = 10
splits = createDataPartition(y = Classification,
                             times = K, p = .80, list = FALSE)

sensitivities1 = numeric(K)
specificities1 = numeric(K)
accuracies1 = numeric(K)
kappas1 = numeric(K)

sensitivities2 = numeric(K)
specificities2 = numeric(K)
accuracies2 = numeric(K)
kappas2 = numeric(K)
```

Now, the base algorithms will be chosen and grids for those with tuning parameters will be defined too.

The base algorithms are:

- Linear Discriminant Analysis
- Random Forest
- Support Vector Machines with Radial Basis Function Kernel
- k-Nearest Neighbors
- Naïve Bayes
- Regularized Logistic Regression

```
base_algorithms = c("lda")


rf_grid = data.frame(mtry = c( floor(sqrt(nFeatures)), 2, 4))

svmRadial_grid = expand.grid(sigma = seq(from = 0.022, to = 0.032, by = 0.001),
                             C = c(1e-1, 1e0, 1e1))

kknn_grid = expand.grid(kmax = 5,
                        distance = c(1, 2),
                        kernel = c("epanechnikov", "biweight", "gaussian"))

naive_bayes_grid = expand.grid(laplace = c(0, 1, 2),
                               usekernel = c(TRUE, FALSE),
                               adjust = c(1,3,5))

regLogistic_grid = expand.grid(cost = c(0.1, 1, 10),
                               loss = c("L1", "L2_dual"),
                               epsilon = 0.001)
```

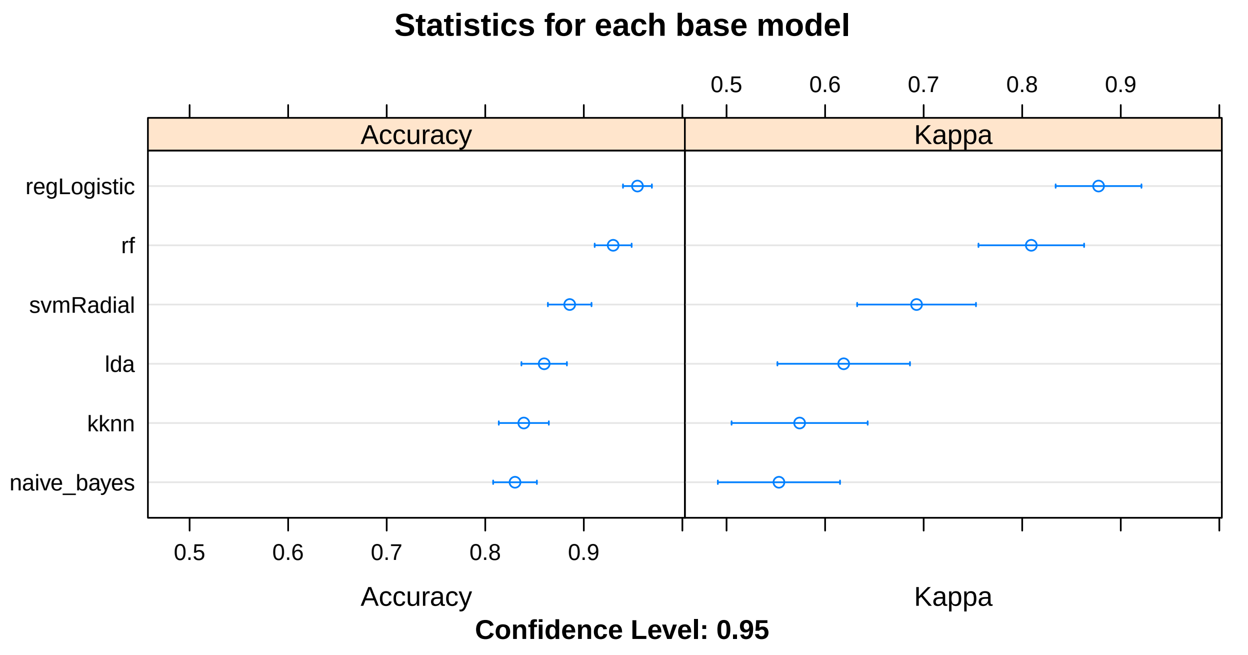
The loop is initialized.

- For the first run, some results will be shown just to see how the execution works. These are the best tuning parameters, a graphic with the cv-performance of each algorithm in the train sample, the correlation matrix (to check for uncorrelated predictions) and the confusion matrix in the test sample along with the corresponding statistics.
- Repeated Cross-Validation (5 times with 10 folds to reduce the variability of the performance estimator) is used for the division of the train sample into the train and validation samples, and so choose the best tune in each run.
- The stacking meta-model is built twice: the first time using a simple linear model via glm (there are no tuning parameters for this model) and the second time a more sophisticated algorithm such as random forest is used to combine the predictions.

```
for (i in 1:K){

  #train-test split
  split = splits[,i]
  trainset = dataset[split, ]
  testset = dataset[-split, ]


  #base models construction
  myControl = trainControl(method = "repeatedcv", number= 10, repeats = 5,
                           index = createMultiFolds(y =trainset$Classification,
                                                    k = 10, times = 5),
                           classProbs = TRUE, savePredictions = 'final')

  baseModels = caretList(Classification~.-Country-Region,
                         data = trainset,
                         trControl = myControl,
                         methodList = base_algorithms,
                         tuneList = list(
                           kknn = caretModelSpec(method='kknn',
                                                 tuneGrid=kknn_grid),
                           rf = caretModelSpec(method='rf',
                                               tuneGrid=rf_grid),
                           svmRadial = caretModelSpec(method='svmRadial',
                                                      tuneGrid=svmRadial_grid),
                           naive_bayes = caretModelSpec(method='naive_bayes',
                                                        tuneGrid=naive_bayes_grid),
                           regLogistic = caretModelSpec(method='regLogistic',
                                                        tuneGrid=regLogistic_grid)),
                         continue_on_fail = FALSE)
```

```r
#Meta-model:
stackControl =  trainControl(method = "repeatedcv", number = 10, repeats = 5)

stack_linear = caretStack(baseModels, method = "glm",
                          metric = "Accuracy", trControl = stackControl)

stack_rf = caretStack(baseModels, method = "rf",
                      metric = "Accuracy", trControl = stackControl)


#predictions
predictions1 = predict(stack_linear, newdata = testset)
cm1 = confusionMatrix(testset$Classification, predictions1)
sensitivities1[i] = cm1$byClass['Sensitivity']
specificities1[i] = cm1$byClass['Specificity']
accuracies1[i] = cm1$overall['Accuracy']
kappas1[i] = cm1$overall['Kappa']

predictions2 = predict(stack_rf, newdata = testset)
cm2 = confusionMatrix(testset$Classification, predictions2)
sensitivities2[i] = cm2$byClass['Sensitivity']
specificities2[i] = cm2$byClass['Specificity']
accuracies2[i] = cm2$overall['Accuracy']
kappas2[i] = cm2$overall['Kappa']


if(i==1){

  print("")
  print(paste("RUN: ",as.character(i)))
  print("")

  print("BEST TUNE PARAMETERS")
  for(alg in names(baseModels)){
    print(alg)
    print(baseModels[[alg]][["bestTune"]])
  }

  results = resamples(baseModels)
  print(dotplot(results,
                main = "Statistics for each base model"))

  print("")
  print("CORRELATION MATRIX")
  print(modelCor(results))

  print("")
  print("META-MODEL with glm and with rf")
  print("STACK_LINEAR:")
  print(stack_linear)

  print("")
  print("STACK_RF:")
```

```
    print(stack_rf)

    print("")
    print("CONFUSION MATRICES for each meta-model")
    print("CM FOR STACK_LINEAR:")
    print(cm1)

    print("")
    print("CM FOR STACK_RF:")
    print(cm2)
  }

}
```



```
## [1] ""
## [1] "RUN:  1"
## [1] ""
## [1] "BEST TUNE PARAMETERS"
## [1] "kknn"
##   kmax distance   kernel
## 2    5        1 biweight
## [1] "rf"
##   mtry
## 2    3
## [1] "svmRadial"
##    sigma  C
## 33 0.032 10
## [1] "naive_bayes"
##   laplace usekernel adjust
## 4       0      TRUE      1
## [1] "regLogistic"
##   cost loss epsilon
## 3    1   L1   0.001
## [1] "lda"
##   parameter
## 1      none
## [1] ""
## [1] "CORRELATION MATRIX"
##                    kknn        rf svmRadial naive_bayes regLogistic       lda
## kknn         1.00000000 0.2920834 0.4857871  0.04570179  -0.1123670 0.3657305
## rf           0.29208341 1.0000000 0.4517682  0.43428289   0.2862681 0.4172494
## svmRadial    0.48578705 0.4517682 1.0000000  0.40238610   0.2081158 0.5468543
## naive_bayes  0.04570179 0.4342829 0.4023861  1.00000000   0.2469142 0.1985397
## regLogistic -0.11236701 0.2862681 0.2081158  0.24691422   1.0000000 0.2335506
## lda          0.36573053 0.4172494 0.5468543  0.19853968   0.2335506 1.0000000
## [1] ""
## [1] "META-MODEL with glm and with rf"
## [1] "STACK_LINEAR:"
## A glm ensemble of 6 base models: kknn, rf, svmRadial, naive_bayes, regLogistic, lda
##
## Ensemble results:
## Generalized Linear Model
##
## 755 samples
##   6 predictor
##   2 classes: 'ecological.deficit', 'ecological.reserve'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 679, 679, 680, 680, 679, 680, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.954214  0.8872188
##
## [1] ""
## [1] "STACK_RF:"
## A rf ensemble of 6 base models: kknn, rf, svmRadial, naive_bayes, regLogistic, lda
##
## Ensemble results:
## Random Forest
##
## 755 samples
##   6 predictor
##   2 classes: 'ecological.deficit', 'ecological.reserve'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 680, 680, 680, 679, 680, 680, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9692947  0.9242139
##   4     0.9695439  0.9245476
##   6     0.9721895  0.9312534
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
## [1] ""
## [1] "CONFUSION MATRICES for each meta-model"
## [1] "CM FOR STACK_LINEAR:"
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           ecological.deficit ecological.reserve
##   ecological.deficit                 25                  1
##   ecological.reserve                  0                 10
##
##                Accuracy : 0.9722
##                  95% CI : (0.8547, 0.9993)
##     No Information Rate : 0.6944
##     P-Value [Acc > NIR] : 3.352e-05
##
##                   Kappa : 0.9328
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9091
##          Pos Pred Value : 0.9615
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6944
##          Detection Rate : 0.6944
##    Detection Prevalence : 0.7222
##       Balanced Accuracy : 0.9545
##
##        'Positive' Class : ecological.deficit
##
## [1] ""
## [1] "CM FOR STACK_RF:"
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           ecological.deficit ecological.reserve
##   ecological.deficit                 25                  1
##   ecological.reserve                  0                 10
##
##                Accuracy : 0.9722
##                  95% CI : (0.8547, 0.9993)
##     No Information Rate : 0.6944
##     P-Value [Acc > NIR] : 3.352e-05
##
##                   Kappa : 0.9328
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9091
##          Pos Pred Value : 0.9615
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6944
##          Detection Rate : 0.6944
##    Detection Prevalence : 0.7222
##       Balanced Accuracy : 0.9545
##
##        'Positive' Class : ecological.deficit
##
```

The averaged effectiveness measures for each stacked model with the same base learners are:

```
(sensitivity1 = mean(sensitivities1))
```

```
## [1] 0.9884615
```

```
(specificity1 = mean(specificities1))
```

```
## [1] 0.9427273
```

```
(accuracy1 = mean(accuracies1))
```

```
## [1] 0.975
```

```
(kappa1 = mean(kappas1))
```

```
## [1] 0.9383123
```

```
(sensitivity2 = mean(sensitivities2))
```

```
## [1] 0.9712496
```

```
(specificity2 = mean(specificities2))
```

```
## [1] 0.9186436
```

```
(accuracy2 = mean(accuracies2))
```

```
## [1] 0.9555556
```

```
(kappa2 = mean(kappas2))
```

```
## [1] 0.8858277
```

Boosting

To find the best model to predict based on our dataset, we are going to implement the three algorithms review in this work. Now, it is the turn of Boosting, for which we will used Generalized Boosted Models with decision trees as base models.

As usual, we load the complete dataset and we keep only the neccesary explanatory variables.

```r
set.seed(15)

library(xlsx)
library(caret)
library(gbm)
library(ggplot2)

dataset = read.xlsx(file = "TFGdataset_complete.xlsx", sheetName = "Sheet1")
dataset = dataset[,-c(1,2)]
attach(dataset)
```

As in Stacking, we implement Monte Carlo Cross-Validation with K = 10 folds and the train sample cointaining the 80% of the data. The statistics we are measuring are sensitivity, specificity, accuracy and the kappa statistic.

```r
K = 10
splits = createDataPartition(y = Classification, times = K, p = .80, list = FALSE)
senss = numeric(K)
specs = numeric(K)
accurs = numeric(K)
kaps = numeric(K)
```

We define the grid for the combination of the tuning parameters. The number of boosting iterations (`n.trees`) will be computed later by Cross-Validation as `gbm` includes it in its functionality. The shrinkage is set not to be higher than 0.01 so that the algorithm learns slowly and thus avoids overfitting, but also, much lower shrinkages carry an important computational cost; interaction depth is set to small number as boosting models look for simple base learners; finally, the number of minimum observations in the terminal nodes of the trees is chosen so as it is not excessively low, taking into account the total number of training observations.

```r
gbm_grid = expand.grid(shrinkage = c(0.01, 0.005, 0.001),
                       interaction.depth = c(3, 5, 7, 9),
                       n.minobsinnode = c(5, 10, 15))
```

The loop is initialized.

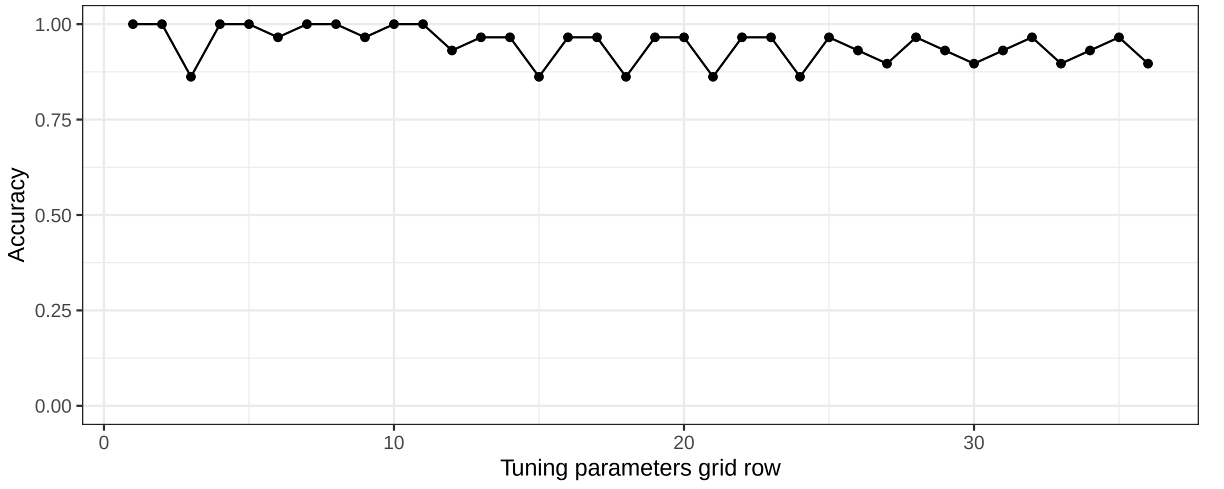
- First, the split of the dataset in train and test sets is done.
- For tuning the parameters, the train set is split into train and validation sets. For each combination of the tuning parameters (i.e., each row of the grid), a model is built. The best tuning is set as the mode of the grid rows that return the highest statistic, for each statistic.
- Once the tuning parameters are chosen, the final model is constructed. The number of trees built (i.e., the number of boosting iterations) is computed by a 10-fold Cross-Validation, where the 20% of the train set is used as validation set.
- In a final step, class predictions on the new data are done using the "best" number of trees.
- Also, for the first run, some results will be shown just to see how the execution works. These are the row of the grid with the best tuning parameters, four plots corresponding to each statistic's performance on the validation set, a print of the final model, a plot of the relative influence of each variable using the best estimated number of trees, a partial dependency plot of the two most influential variables (these graphs show the marginal effect of such variables on the modeled target variable through a heatmap), a

plot of train, validation and test errors estimated by the CV selection of the number of iterations, and finally the confussion matrix that evals the predictions over the test sample.

```r
for (k in 1:K){

  #################################

  # DIVISIÓN TRAIN TEST

  split = splits[,k]
  trainset = dataset[split, ]
  testset = dataset[-split, ]

  ##############################

  # TUNING OF PARAMETERS

  splitValidation = createDataPartition(y = trainset$Classification,
                                        times = 1, p = .80, list = FALSE)
  trainset2 = trainset[splitValidation, ]
  validationset = trainset[-splitValidation, ]

  sensitivities = numeric(dim(gbm_grid)[1])
  specificities = numeric(dim(gbm_grid)[1])
  accuracies = numeric(dim(gbm_grid)[1])
  kappas = numeric(dim(gbm_grid)[1])

  for (i in 1:dim(gbm_grid)[1]) {

    shrink = gbm_grid[i,1]
    depth = gbm_grid[i,2]
    minobs = gbm_grid[i,3]

    model_gbm = gbm(formula = Classification~.,
                    data = trainset2,
                    distribution = "multinomial",  # loss function
                    n.trees = 1000, # the number of iterations (to be tuned later)
                    shrinkage = shrink,  # learning rate or step-size reduction
                    interaction.depth = depth,  # the depth of each tree
                    n.minobsinnode = minobs)  # min nº of observations in the terminal nodes

    probPreds = predict(model_gbm,
                        newdata = validationset,
                        n.trees = model_gbm$n.trees,
                        type = "response")
    factPreds = apply(probPreds, 1, which.max) # for a matrix 1 indicates rows
    classPreds = factor(colnames(probPreds)[factPreds])

    cm = confusionMatrix(factor(validationset$Classification), classPreds)
    sensitivities[i] = cm$byClass['Sensitivity']
    specificities[i] = cm$byClass['Specificity']
    accuracies[i] = cm$overall['Accuracy']
    kappas[i] = cm$overall['Kappa']
  }
  bestTuning = as.numeric(names(
    which.max(table(c(which.max(accuracies),
                      which.max(kappas),
                      which.max(specificities),
                      which.max(sensitivities))))))

  if (k == 1) {
    print(paste("RUN:", as.character(k)))
    print("")

    print( gbm_grid[bestTuning,] )

    print( ggplot(data.frame(x = 1:dim(gbm_grid)[1], y = accuracies),
                  aes(x = x, y = y)) +
             geom_point() +
             geom_line() +
             scale_y_continuous(limits = c(0, NA)) +
             labs(x='Tuning parameters grid row', y = 'Accuracy') +
             theme_bw() +
             labs(
               title = "Validation Accuracy depending on the parameters",
               caption = "The validation accuracy obtained with the combination \n
of the tuning parameters for each row in the grid") +
             theme(
               plot.title = element_text(hjust = 0.5, size = 14),
               plot.caption = element_text(hjust = 0, face = "italic")
               ))

    print( ggplot(data.frame(x = 1:dim(gbm_grid)[1], y = sensitivities),
                 aes(x = x, y = y)) +
            geom_point() +
            geom_line() +
            scale_y_continuous(limits = c(0, NA)) +
            labs(x='Tuning parameters grid row', y = 'Sensitivity') +
             theme_bw() +
             labs(
               title = "Validation Sensitivity depending on the parameters",
               caption = "The validation sensitivity obtained with the combination \n
of the tuning parameters for each row in the grid") +
             theme(
               plot.title = element_text(hjust = 0.5, size = 14),
               plot.caption = element_text(hjust = 0, face = "italic")
               ))

    print( ggplot(data.frame(x = 1:dim(gbm_grid)[1], y = specificities),
                 aes(x = x, y = y)) +
            geom_point() +
            geom_line() +
            scale_y_continuous(limits = c(0, NA)) +
            labs(x='Tuning parameters grid row', y = 'Specificity') +
             theme_bw() +
             labs(
               title = "Validation Specificity depending on the parameters",
               caption = "The validation Specificity obtained with the combination \n
of the tuning parameters for each row in the grid") +
             theme(
               plot.title = element_text(hjust = 0.5, size = 14),
               plot.caption = element_text(hjust = 0, face = "italic")
               ))

    print( ggplot(data.frame(x = 1:dim(gbm_grid)[1], y = kappas),
                 aes(x = x, y = y)) +
            geom_point() +
            geom_line() +
            scale_y_continuous(limits = c(0, NA)) +
            labs(x='Tuning parameters grid row', y = 'Kappa statistic') +
             theme_bw() +
             labs(
               title = "Validation Kappa statistic depending on the parameters",
               caption = "The validation Kappa statistic obtained with the combination \n
of the tuning parameters for each row in the grid") +
             theme(
               plot.title = element_text(hjust = 0.5, size = 14),
               plot.caption = element_text(hjust = 0, face = "italic")
               ))
  }

  #####################################

  # Final model with the selected tuning parameters.
  # To compute n.trees by CV

  fit_gbm = gbm(formula = Classification~.,
                data = trainset,
                train.fraction = 0.8,
                distribution = "multinomial",
                n.trees = 1000,  # the number of iterations
                shrinkage = gbm_grid[bestTuning,1],
                interaction.depth = gbm_grid[bestTuning,2],
                n.minobsinnode = gbm_grid[bestTuning,3],
                cv.folds = 10) # Number of cross-validation folds to perform

  # number of trees with minimum CV error
  best_iter = gbm.perf(fit_gbm, method="cv", plot.it = FALSE)

  if ( k == 1 ) {
    #model
    print(fit_gbm)

    # Plot relative influence of each variable using the best estimated  number of trees
    print( summary(fit_gbm, n.trees = best_iter,
                   main = "Relative influence of each variable \n
using the estimated best number of trees") )

    print( plot(fit_gbm, i.var = c('Forest.Land', 'Grazing.Land'),
                n.trees = best_iter, type = "response",
                main = "partial dependency plot of \n
the two most influential variables",
                sub = "response > 0.5 ecological.deficit\n
response < 0.5 ecological.reserve") )

    # PLOT TRAIN VALIDATION TEST ERRORS
    nIter = seq(along = fit_gbm$train.error)
    print( ggplot(data.frame(nIter = rep(nIter, 3),
                             error = c(fit_gbm$train.error,
                                       fit_gbm$valid.error,
                                       fit_gbm$cv.error),
                             type = c(rep('train', length(nIter)),
                                      rep('validation', length(nIter)),
                                      rep('test', length(nIter)))),
                  aes(x = nIter, y = error, col = type)) +
            geom_line() +
            geom_point() +
            geom_vline(xintercept = best_iter,
                       linetype = "dashed", color = "black", size = 1.5) +
            annotate("text",
                     x = (best_iter+100),
                     y = 0.5,
                     label = as.character(best_iter)) +
            theme_bw() +
             labs(
               title = "Validation, Train and Test errors",
               caption = "The erros are plotted against the number of iterations, \n
showing also the best nIter") +
             theme(
               plot.title = element_text(hjust = 0.5, size = 14),
               plot.caption = element_text(hjust = 0, face = "italic")
               ))
  }

  # Predict on the new data using the "best" number of trees
  probPreds = predict(fit_gbm, newdata = testset,
                      n.trees = best_iter, type = "response")
  factPreds = apply(probPreds, 1, which.max)
  classPreds = factor(colnames(probPreds)[factPreds])

  cm = confusionMatrix(factor(testset$Classification), classPreds)
  senss[k] = cm$byClass['Sensitivity']
  specs[k] = cm$byClass['Specificity']
  accurs[k] = cm$overall['Accuracy']
  kaps[k] = cm$overall['Kappa']
  if ( k == 1 ) {
    print(cm)
  }
}
```

```
## [1] "RUN: 1"
## [1] ""
##   shrinkage interaction.depth n.minobsinnode
## 1      0.01                 3              5
```

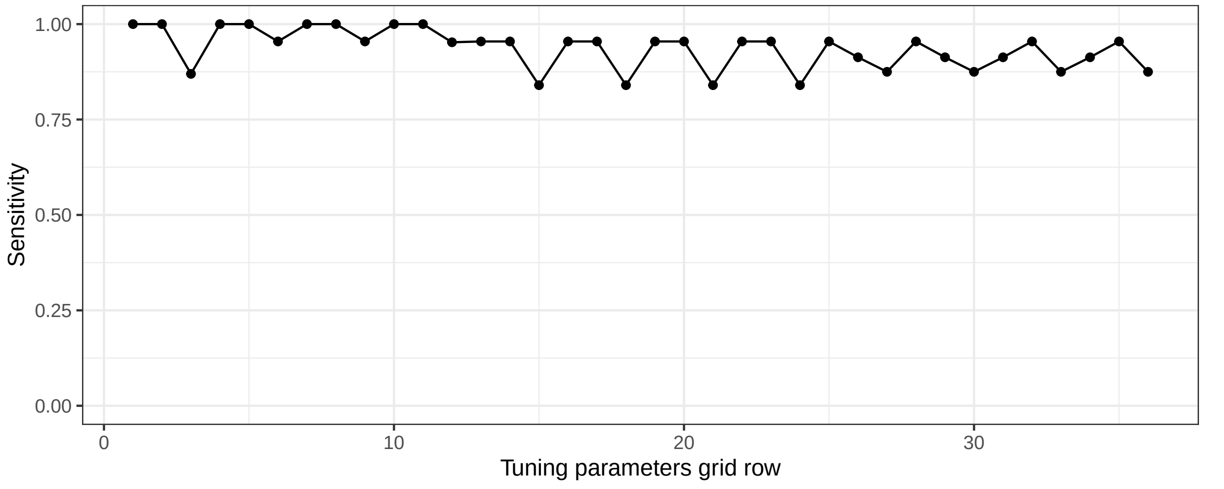
Validation Accuracy depending on the parameters



The validation accuracy obtained with the combination

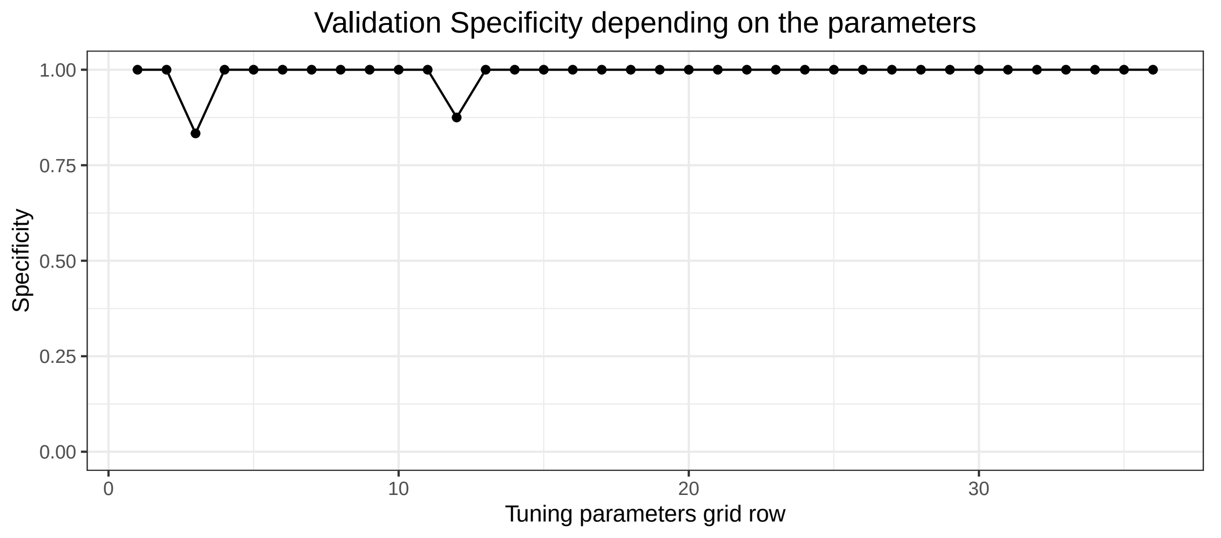
of the tuning parameters for each row in the grid

Validation Sensitivity depending on the parameters



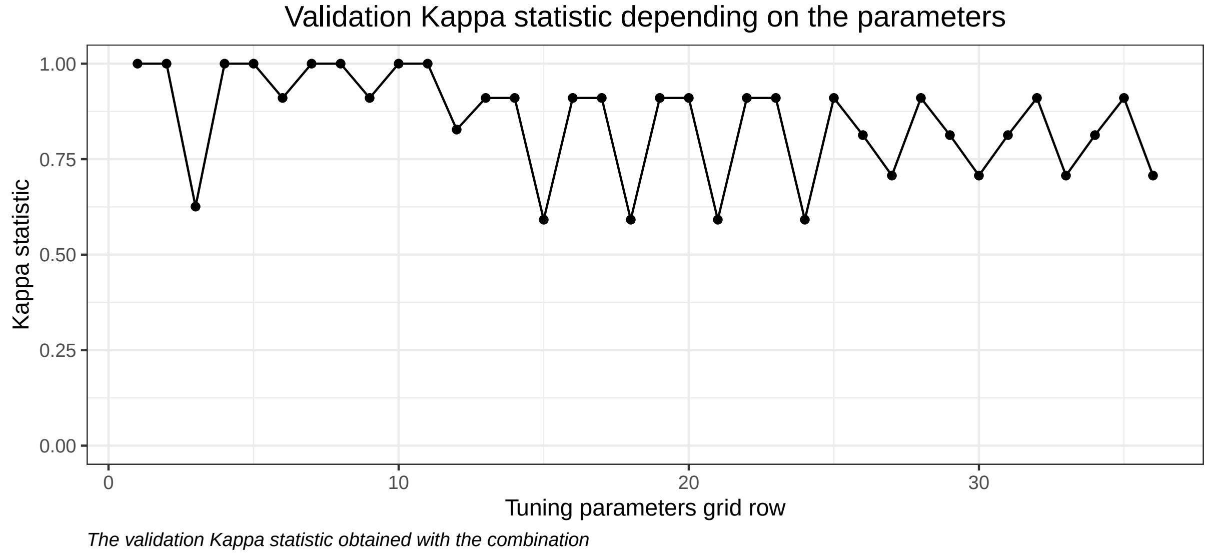
The validation sensitivity obtained with the combination

of the tuning parameters for each row in the grid



The validation Specificity obtained with the combination

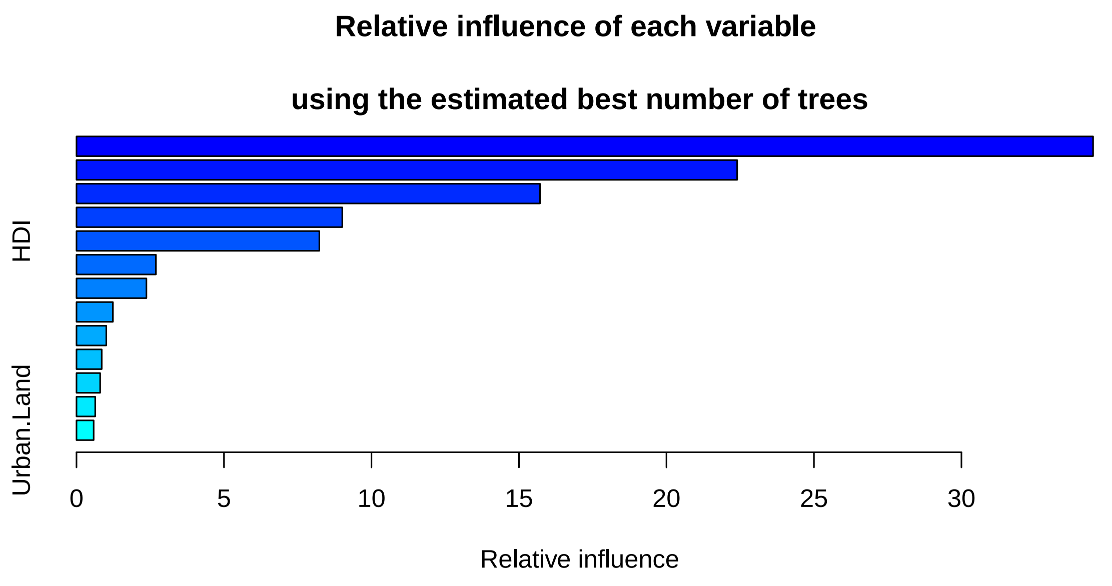
of the tuning parameters for each row in the grid



The validation Kappa statistic obtained with the combination

of the tuning parameters for each row in the grid

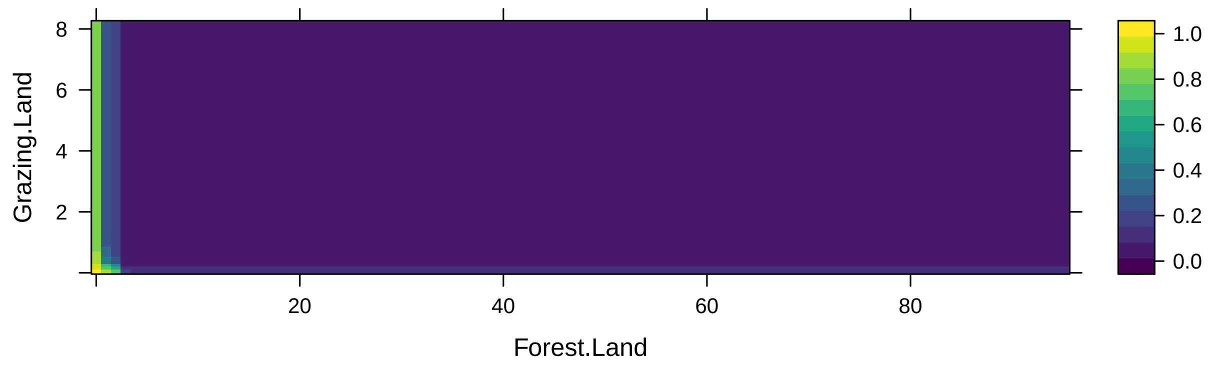
```
## gbm(formula = Classification ~ ., distribution = "multinomial",
##     data = trainset, n.trees = 1000, interaction.depth = gbm_grid[bestTuning,
##         2], n.minobsinnode = gbm_grid[bestTuning, 3], shrinkage = gbm_grid[bestTuning,
##         1], train.fraction = 0.8, cv.folds = 10)
## A gradient boosted model with multinomial loss function.
## 1000 iterations were performed.
## The best cross-validation iteration was 314.
## The best test-set iteration was 978.
## There were 13 predictors of which 13 had non-zero influence.
```



```
##                                    var    rel.inf
## Forest.Land               Forest.Land 34.4627436
## Grazing.Land             Grazing.Land 22.3972641
## Carbon.Footprint     Carbon.Footprint 15.7138305
## Fishing.Water           Fishing.Water  9.0108382
## HDI                               HDI  8.2326315
## Cropland.Footprint Cropland.Footprint  2.6924769
## Grazing.Footprint   Grazing.Footprint  2.3708171
## Fish.Footprint         Fish.Footprint  1.2346053
## GDP.per.Capita         GDP.per.Capita  1.0102299
## Cropland                     Cropland  0.8549664
## Forest.Footprint     Forest.Footprint  0.8024714
## Population                 Population  0.6352191
## Urban.Land                 Urban.Land  0.5819061
```

partial dependency plot of

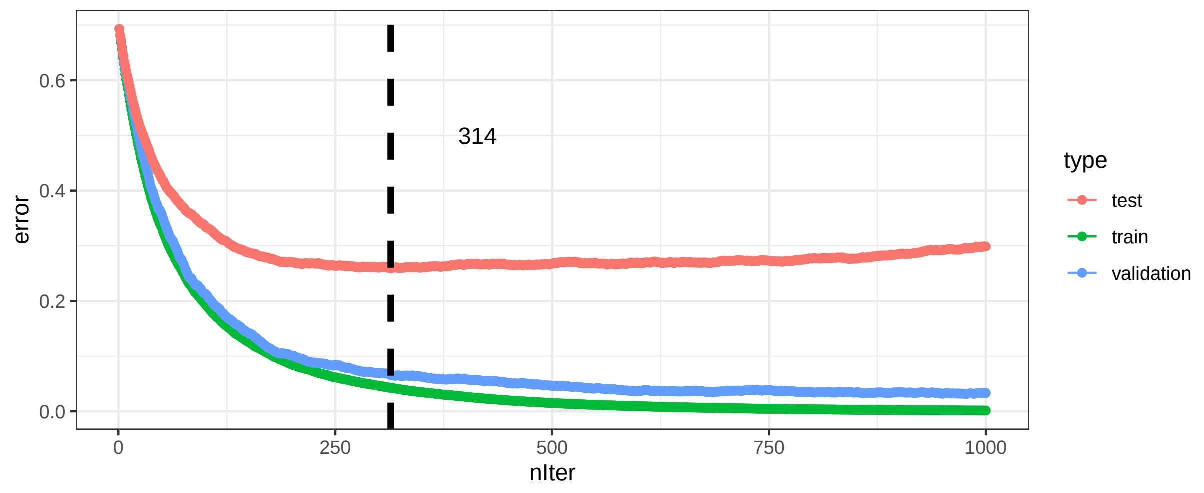
the two most influential variables



response > 0.5 ecological.deficit

response < 0.5 ecological.reserve

Validation, Train and Test errors



The erros are plotted against the number of iterations,

showing also the best nIter

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           ecological.deficit ecological.reserve
##   ecological.deficit                 26                  0
##   ecological.reserve                  1                  9
##
##                Accuracy : 0.9722
##                  95% CI : (0.8547, 0.9993)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 0.0004132
##
##                   Kappa : 0.9286
##
```

```
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.9630
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9000
##              Prevalence : 0.7500
##          Detection Rate : 0.7222
##    Detection Prevalence : 0.7222
##       Balanced Accuracy : 0.9815
##
##        'Positive' Class : ecological.deficit
##
```

The averaged effectiveness measures are:

```
(sens = mean(senss))
```

```
## [1] 0.9223591
```

```
(spec = mean(specs))
```

```
## [1] 0.8805916
```

```
(accur = mean(accurs))
```

```
## [1] 0.9111111
```

```
(kap = mean(kaps))
```

```
## [1] 0.7634279
```

Bagging

By last, Random Forest has been chosen as the algorithm with which to implement Bagging in this project.

As usual, we import the complete dataset from the .xlsx file.

```
set.seed(15)

library(xlsx)
library(randomForest)
library(ggplot2)
library(caret)

dataset = read.xlsx(file = "TFGdataset_complete.xlsx", sheetName = "Sheet1")
attach(dataset)
```

As in the previous analysis, we implement Monte Carlo Cross-Validation with $\mathsf{K} = 10$ folds and the train sample cointaining the 80% of the data. The statistics we are measuring are, again, sensitivity, specificity, accuracy and the kappa statistic.

```
K = 10
splits = createDataPartition(y = Classification, times = K, p = .80, list = FALSE)
senss = numeric(K)
specs = numeric(K)
accurs = numeric(K)
kaps = numeric(K)
```

We define the grid for the combination of the tuning parameters. The number of trees in the forest (ntrees), which needs to be sufficiently large to stabilize the error rate; the number of predictor variables chosen to split at each node (mtry), usually has the largest impact on predictive accuracy and so is set to every possible value; the minimum number of observations in terminal nodes (nodesize), is stablished as for boosted trees.

```
rf_grid = expand.grid(ntree = seq(20, 120, by = 10),
                      mtry = 1:(dim(dataset)[2]-3-1),
                      nodesize = c(5, 10, 15))
```

The loop is initialized.

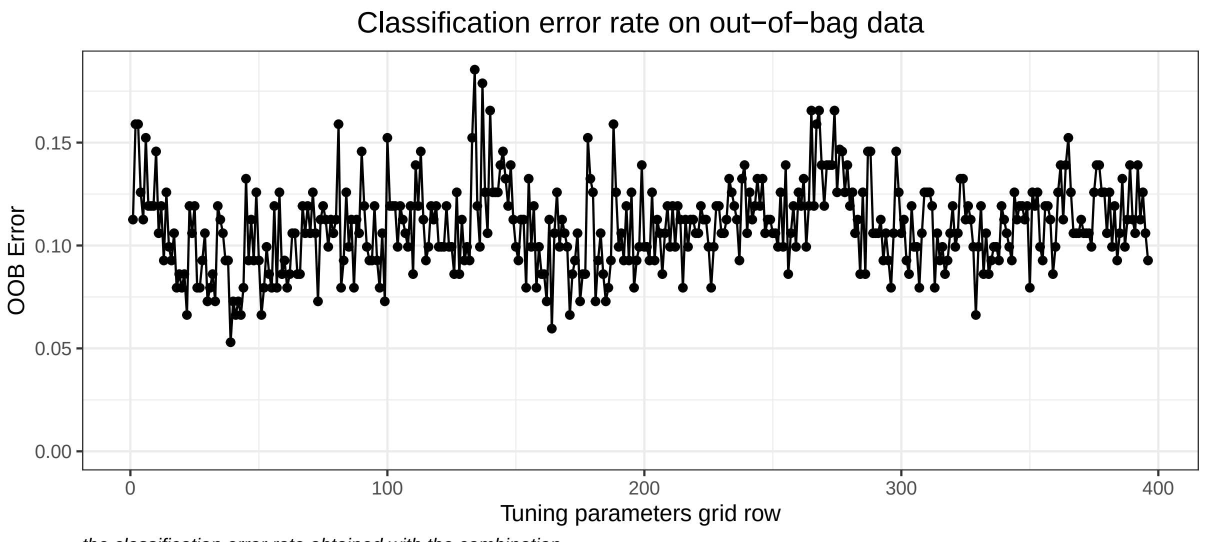
- First, the split of the dataset in train and test sets is done.
- For tuning the parameters, because of the nature of the bagging process, it is possible to directly estimate the validation error without the need of an independent validation set using the OOB-error, since it is estimated on the excluded observations when building the model. For each combination of the tuning parameters (i.e., each row of the grid), a model is built. The best tuning is set as the grid row that returns the lowest error.
- Once the tuning parameters are chosen, the final model is constructed and class predictions on the new data are given.
- Also, for the first run, some results will be shown just to see how the execution works. These are the row of the grid with the best tuning parameters, a plot corresponding to the OOB error for each row of the grid, a print of the final model, a print and a plot of importance measure referring to the prediction error on the out-of-bag portion and the node impurities, and finally the confussion matrix that evals the predictions over the test sample.

```
for (k in 1:K){

  ##################################
```

```r
# DIVISIÓN TRAIN-TEST

split = splits[,k]
trainset = dataset[split, ]
testset = dataset[-split, ]

##############################

# TUNING OF PARAMETERS

oobErrors = numeric(dim(rf_grid)[1])

for (i in 1:dim(rf_grid)[1]) {

  nt = rf_grid[i,1]
  mt = rf_grid[i,2]
  ns = rf_grid[i,3]

  model_rf = randomForest(formula = Classification~.-Country-Region,
                          data = trainset,
                          replace = TRUE,
                          ntree = nt,
                          mtry = mt,
                          nodesize = ns,
                          importance = FALSE)

  cm = model_rf$confusion #based on OOB data
  oobErrors[i] = (cm[2]+cm[3])/(cm[1]+cm[2]+cm[3]+cm[4])
}

bestTuning = which.min(oobErrors)
if (k==1){
  print(paste("RUN:", k))
  print("")

  print( rf_grid[bestTuning,] )

  print( ggplot(data.frame(x = 1:dim(rf_grid)[1], y = oobErrors),
                aes(x = x, y = y)) +
           geom_point() +
           geom_line() +
           scale_y_continuous(limits = c(0, NA)) +
           labs(x='Tuning parameters grid row', y = 'OOB Error') +
           theme_bw() +
           labs(
             title = "Classification error rate on out-of-bag data",
             caption = "the classification error rate obtained with the combination \n
of the tunning parameters for each row in the grid") +
           theme(
             plot.title = element_text(hjust = 0.5, size = 14),
             plot.caption = element_text(hjust = 0, face = "italic")
             ))
}
```

```
  #######################################

  # MODEL CONSTRUCTION WITH THE TUNING PARAMETERS

  fit_rf = randomForest(formula = Classification~.-Country-Region,
                        data = trainset,
                        replace = TRUE,
                        ntree = rf_grid[bestTuning,1],
                        mtry = rf_grid[bestTuning,2],
                        nodesize = rf_grid[bestTuning,3],
                        importance = TRUE)

  if(k == 1){
    print(fit_rf)

    importance(fit_rf)
    varImpPlot(fit_rf, main = "importance measures")
  }

  predicciones = predict(fit_rf, newdata = testset, type = "response")
  cm = confusionMatrix(testset$Classification, predicciones)
  senss[k] = cm$byClass['Sensitivity']
  specs[k] = cm$byClass['Specificity']
  accurs[k] = cm$overall['Accuracy']
  kaps[k] = cm$overall['Kappa']

  if(k==1){
    print(cm)
  }

}
```

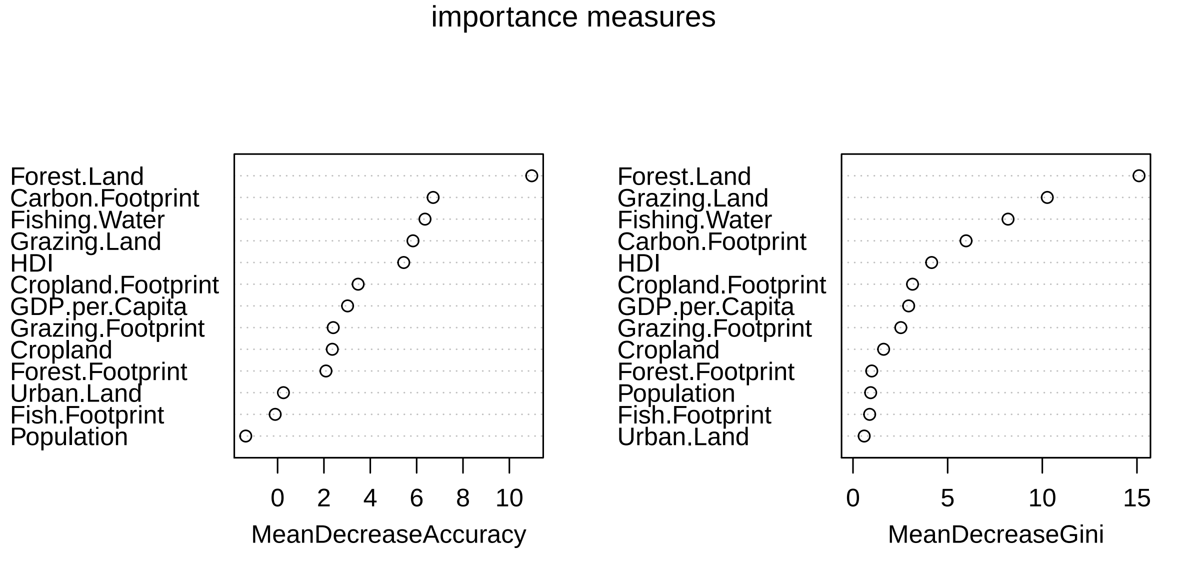
```
## [1] "RUN: 1"
## [1] ""
##    ntree mtry nodesize
## 39    70    4        5
```



the classification error rate obtained with the combination

of the tunning parameters for each row in the grid

```
##
## Call:
##  randomForest(formula = Classification ~ . - Country - Region,      data = trainset, replace = TRUE,
##                Type of random forest: classification
##                      Number of trees: 70
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 10.6%
## Confusion matrix:
##                     ecological.deficit ecological.reserve class.error
## ecological.deficit                 101                  7  0.06481481
## ecological.reserve                   9                 34  0.20930233
```



```
## Confusion Matrix and Statistics
```

```
##
##                       Reference
## Prediction           ecological.deficit ecological.reserve
##   ecological.deficit                 26                  0
##   ecological.reserve                  1                  9
##
##                Accuracy : 0.9722
##                  95% CI : (0.8547, 0.9993)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 0.0004132
##
##                   Kappa : 0.9286
##
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.9630
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9000
##              Prevalence : 0.7500
##          Detection Rate : 0.7222
##    Detection Prevalence : 0.7222
##       Balanced Accuracy : 0.9815
##
##        'Positive' Class : ecological.deficit
##
```

The averaged effectiveness measures are:

```
(sens = mean(senss))
```

```
## [1] 0.9215247
```

```
(spec = mean(specs))
```

```
## [1] 0.8607937
```

```
(accur = mean(accurs))
```

```
## [1] 0.9055556
```

```
(kap = mean(kaps))
```

```
## [1] 0.7509424
```

4.4 Results

This subsection analyses the results of the implementation of the three main algorithms which have been reviewed in this project. The following table summarizes the effectiveness measures on the test sample obtained with the respective algorithms.

Algorithm	Accuracy	Kappa statistic	Sensitivity	Specificity
Stacking$_{linear}$	0.975	0.938	0.988	0.943
Stacking$_{RF}$	0.956	0.886	0.971	0.919
Boosting with GBM	0.908	0.757	0.922	0.869
Bagging with Random Forest	0.906	0.751	0.923	0.861

When applying the Stacking algorithms, a wide variety of base algorithms have been chosen: Linear Discriminant Analysis, Random Forest, Support Vector Machines with Radial Basis Function Kernel, k-Nearest Neighbors, Naïve Bayes and Regularized Logistic Regression. Through tuning their parameters to achieve the best performance, we obtain six uncorrelated models, which is an important factor when creating the stacked ensembles. A significant generalization accuracy is achieved with a simple linear combination of the outputs, which shows that the algorithm has been fully exploited, accomplishing that each base learner contributes somehow to the final meta learner and avoiding redundancies. Even so, more sophisticated algorithms can used to combine the predictions too. Random Forest algorithm has been used to check whether the results could be improved. Although the statistics obtained can be considered practically equivalent, the slight worsening together with the higher computational and interpretive costs that this latter model entails, lead to that its use does not make up for.

A Generalized Boosted Model, which implements the gradient boosting method in decision trees, is computed in second place. In this case, we do not want to adjust very large trees to gain precision, we focus on generating small trees (with few terminal nodes) and sequentially readjust the predictions identifying the errors so that the algorithm learns to improve them at these points. In this sense, the proposed tuning parameters to apply the boosting algorithm to a data set with decision trees must be chosen taking into account that we want the model to be simple: the learning rate is not recommended to be greater than 0.01 for the algorithm to learn slowly and thus avoid overfitting; regarding the depth of the trees, very small values are selected (even one single division is often enough). As on previous occasions, the trained model is stored in an object, of class `gbm`, which overloads common functions to extract information. It can be seen in the provided graphics that the `Forest.Land` and `Grazing.Land` variables are those that most condition the class of the objective variable `Classification`. The partial dependency plot returns a heatmap which shows that countries with the least amount of hectares in forests and grazing lands have a greater probability of running an ecological deficit.

Finally, once the number of trees built is computed by cross validation, we manage to get nearly as good generalization measures as with Stacking with a much lower computational cost.

As Bagging works well on unstable methods such as trees, which, being notoriously noisy procedures, benefit greatly from averaging, Random Forests is the last algorithm implemented to try to model the ecological footprint. Regarding the tuning parameters, the only one to which the algorithm is somewhat sensitive appears to be the number of predictor variables chosen to split at each node, therefore all possible values are provided. When we take a look at the given importance measures - the first based on the average decrease in the prediction accuracy over the out-of-bag portion when that variable is excluded, and the second describing the total decrease of a variable (averaged by all the trees) in the node impurity measured by the Gini index -, we find that the variables with the highest measures are `Forest.land` and `Grazing.land`. In general, the higher these measurements, the more important the variable will be, hence, it matches with the information provided above by the Boosting model: the variable `Forest.Land` is by far the most important. This information is consistent with the logical, since it is the main productive land area required to get rid of the tonnes of carbon dioxide emissions. The results are also impressive with this algorithm.

As for which technique optimally addresses the problem, the choice is not unequivocal since it is a multiobjective problem: depending on the criterion to be taken into account - precision of the model, complexity, computation time - some models perform better than others. All the tested Ensemble Methods provide reliable predictions of the ecological footprint of the countries, nevertheless, none dominates all the criteria. Stacking provides the best performance, yet it is the most complex. On the other hand, the lower computational cost of Boosting because of its sequentially model generation, along with the easy parallelization of the base model construction of Bagging, makes them very appealing, although the loss in precision compared to the previous one is considerable.

Bibliography

[1] A. Kobler and M. Adamic, "Identifying brown bear habitat by a combined gis and machine learning method," *Ecological Modelling*, vol. 135, no. 2-3, 2000. (pages 1 and 8).

[2] K. Murphy, J. Reynolds, J. Jenkins, E. Whitten, N. Fresco, M. Lindgren, and F. Huettmann, "Predicting future potential climate-biomes for the yukon, northwest territories, and alaska," 2012. (pages 1 and 6).

[3] S. M. Hardy, M. Lindgren, H. Konakanchi, and F. Huettmann, "Predicting the distribution and ecological niche of unexploited snow crab (chionoecetes opilio) populations in alaskan waters: a first open-access ensemble model," 2011. (pages 2 and 20).

[4] T. Cai, F. Huettmann, and Y. Guo, "Using stochastic gradient boosting to infer stopover habitat selection and distribution of hooded cranes grus monacha during spring migration in lindian, northeast china," *PLoS One*, vol. 9, no. 2, 2014. (pages 2 and 28).

[5] A. M. Prasad, L. R. Iverson, and A. Liaw, "Newer classification and regression tree techniques: bagging and random forests for ecological prediction," *Ecosystems*, vol. 9, no. 2, 2006. (pages 2 and 32).

[6] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. (pages 2, 5, and 35).

[7] Global Footprint Network, "`https://www.footprintnetwork.org`." (pages 2, 38, and 40).

[8] A. K. Laha, *Advances in Analytics and Applications*. Springer, 2019. (page 3).

[9] G. R. Humphries, D. R. Magness, and F. Huettmann, *Machine learning for ecology and sustainable natural resource management*. Springer, 2018. (pages 3 and 39).

[10] G. Rebala, A. Ravi, and S. Churiwala, *An Introduction to Machine Learning*. Springer, 2019. (page 4).

[11] E. Alpaydin, *Introduction to machine learning*. MIT press, 2009. (pages 4, 8, 27, and 30).

[12] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011. (page 4).

[13] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, "Big data preprocessing: methods and prospects," *Big Data Analytics*, vol. 1, no. 1, 2016. (page 5).

[14] P. C. Team, "Python: A dynamic, open source programming language," *Python Software Foundation*, vol. 78, 2015. (page 5).

[15] G. Van Rossum and F. L. Drake, *The python language reference manual*. Network Theory Ltd., 2011. (page 5).

[16] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, 2017. (page 5).

[17] R. F. de Mello and M. A. Ponti, *Machine Learning: A Practical Approach on the Statistical Learning Theory*. Springer, 2018. (page 5).

[18] A. Thessen, "Adoption of machine learning techniques in ecology and earth science," *One Ecosystem*, vol. 1, 2016. (page 6).

[19] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, 1967. (page 6).

[20] C. A. Charu and K. R. Chandan, "Data clustering: algorithms and applications," 2013. (page 7).

[21] B. Liu, *Web data mining: exploring hyperlinks, contents, and usage data*. Springer Science & Business Media, 2007. (page 7).

[22] O. Chapelle, B. Schölkopf, and A. Zien, "Semi-supervised learning. adaptive computation and machine learning," *MIT Press, Cambridge, MA, USA. Cited in page (s)*, vol. 21, no. 1, 2010. (page 7).

[23] G. De'ath and K. E. Fabricius, "Classification and regression trees: a powerful yet simple technique for ecological data analysis," *Ecology*, vol. 81, no. 11, 2000. (page 8).

[24] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994. (pages 9 and 18).

[25] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, 1997. (page 9).

[26] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*, vol. 821. John Wiley & Sons, 2012. (page 10).

[27] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in neural information processing systems*, 1997. (page 10).

[28] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, 2004. (page 10).

[29] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, 2005. (page 11).

[30] M. Hardt, E. Price, N. Srebro, *et al.*, "Equality of opportunity in supervised learning," in *Advances in neural information processing systems*, 2016. (page 11).

[31] M. Cacciola, G. Megali, and F. C. Morabito, "An optimized support vector machine based approach for non-destructive bumps characterization in metallic plates," in *Intelligent Computer Techniques in Applied Electromagnetics*, Springer, 2008. (page 12).

[32] A. Mathur and G. M. Foody, "Multiclass and binary svm classification: Implications for training and classification users," *IEEE Geoscience and remote sensing letters*, vol. 5, no. 2, 2008. (page 13).

[33] J. Carletta, "Assessing agreement on classification tasks: the kappa statistic," *Computational linguistics*, vol. 22, no. 2, 1996. (page 14).

[34] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006. (page 14).

[35] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001. (pages 15, 16, 17, 19, 20, and 32).

[36] Q.-S. Xu and Y.-Z. Liang, "Monte Carlo cross validation," *Chemometrics and Intelligent Laboratory Systems*, vol. 56, no. 1, 2001. (page 17).

[37] L. Breiman, "Out-of-bag estimation," 1996. (page 18).

[38] C. Zhang and Y. Ma, *Ensemble machine learning: methods and applications*. Springer, 2012. (pages 19, 21, 22, 28, 30, and 34).

[39] B. V. Dasarathy and B. V. Sheela, "A composite classifier system design: concepts and methodology," *Proceedings of the IEEE*, vol. 67, no. 5, 1979. (page 20).

[40] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 10, 1990. (pages 20 and 21).

[41] R. E. Schapire, "The strength of weak learnability," *Machine learning*, vol. 5, no. 2, pp. 197–227, 1990. (page 20).

[42] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, Springer, 2000. (page 22).

[43] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, 1992. (page 25).

[44] S. Schmitt, R. Pouteau, D. Justeau, F. de Boissieu, and P. Birnbaum, "Ssdm: an r package to predict distribution of species richness and endemism based on stacked species distribution models," *Methods in Ecology and Evolution*, vol. 8, no. 12, pp. 1795–1803, 2017. (page 27).

[45] S. Schmitt, R. Pouteau, D. Justeau, F. de Boissieu, and P. Birnbaum, "ssdm: An r package to predict distribution of species richness and composition based on stacked species distribution models," *Methods in Ecology and Evolution*, vol. 8, no. 12, 2017. (page 27).

[46] A. Guisan and W. Thuiller, "Predicting species distribution: offering more than simple habitat models," *Ecology letters*, vol. 8, no. 9, 2005. (page 27).

[47] L. Breiman, "Bagging predictors," tech. rep., Technical Report, 1994. (pages 30 and 31).

[48] G. Seni and J. F. Elder, "Ensemble methods in data mining: improving accuracy through combining predictions," *Synthesis lectures on data mining and knowledge discovery*, vol. 2, no. 1, 2010. (page 31).

[49] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, 2001. (pages 32 and 34).

[50] M. Kuhn, *caret: Classification and Regression Training*, 2020. R package version 6.0-85. (pages 35 and 36).

[51] Z. A. Deane-Mayer and J. E. Knowles, *caretEnsemble: Ensembles of Caret Models*, 2019. R package version 2.0.1. (pages 35 and 36).

[52] B. Greenwell, B. Boehmke, J. Cunningham, and G. Developers, *gbm: Generalized Boosted Regression Models*, 2019. R package version 2.1.5. (page 37).

[53] B. Boehmke and B. M. Greenwell, *Hands-On Machine Learning with R*. CRC Press, 2019. (page 37).

[54] T. Therneau and B. Atkinson, *rpart: Recursive Partitioning and Regression Trees*, 2019. R package version 4.1-15. (page 37).

[55] A. Peters and T. Hothorn, *ipred: Improved Predictors*, 2019. R package version 0.9-9. (page 37).

[56] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. (page 38).

[57] J. Hickel, "The sustainable development index: Measuring the ecological efficiency of human development in the anthropocene," *Ecological Economics*, vol. 167, 2020. (page 39).