POLITECNICO DI TORINO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MASTER THESIS

MODELING FISH SPECIES DISTRIBUTIONS WITH MACHINE LEARNING TECHNIQUES

Author:

Cristian Garcia Ventura

Supervisors:

Prof. Paolo Vezza

Eng. Giovanni Negro

Prof. Asuncion Santafe

A.A 2021-2022

ABSTRACT

Freshwater natural systems are widely affected by hydromorphological pressures, which result from human actions in the environment and the overconsumption of natural resources. In order to protect them, the European Union created the Water Framework Directive in 2000. The main aim of the directive is to improve the environmental status of the freshwater bodies by means of River Basin Management Plans.

In the case of the rivers, there is a direct relationship between their ecological status and the presence or absence of certain fish species. The machine learning techniques are valuable and powerful tools to study the relationship between the distribution of fish species and the habitat descriptors that characterize the mesohabitats, river areas of 10^{-1} - 10^{-3} meters that can be considered functional habitats.

Random Forests is one of the most widely used machine learning techniques. This technique, which is based on the use of decision trees, has been criticized in recent years because it gives greater importance to those variables with more possible values or classes. In this context, the Conditional Inference Forests technique was born, whose approach reduces the variable selection bias by using a two-step division criterion during the construction of the decision trees that make up the forest of predictors.

More specifically, the project's aim consists of comparing the performance of both techniques when they are used to predict the habitability of three fish species (young specimens of Lethenteron Zanandreai and adult specimens of Padogobius Bonelli and Phoxinus Lumaireul) along various rivers located in Piedmont and Emilia-Romagna, Italy. The project also studies the effects of balancing the input dataset before using them to build the models.

In order to achieve the project objectives, a code based on the programming language R has been written. The code was developed with RStudio, an integrated development environment for R, which has a considerable number of packages available (such as *Boruta*, *partykit,* and *cforest*), which have been used to implement the Random Forests and Conditional Inference Forests techniques; as well as to calibrate the different models developed and to compare their performances, among other tasks.

Finally, after representing and comparing the results, the project concluded that the models that achieved the best performances were those based on the Conditional Inference approach and whose input data was previously balanced with an oversampling method; those models reached the lowest errors and the highest sensitivities and specificities.

SOMMARIO

I sistemi naturali d'acqua dolce sono ampiamente influenzati dalle pressioni idromorfologiche, che derivano dalle azioni umane sull'ambiente e dal consumo eccessivo di risorse naturali. Per proteggerli, l'Unione Europea ha creato nel 2000 la Direttiva Quadro sulle Acque. L'obiettivo principale della direttiva è quello di migliorare lo stato ambientale dei corpi idrici d'acqua dolce attraverso i Piani di Gestione dei Bacini Idrografici.

Nel caso dei fiumi, esiste una relazione diretta tra il loro stato ecologico e la presenza o meno di determinate specie ittiche. Le tecniche di apprendimento automatico sono strumenti preziosi e potenti per studiare la relazione tra la distribuzione delle specie ittiche e i descrittori di habitat che caratterizzano i mesohabitat, aree fluviali di 10^{-1} - 10^{-3} metri che possono essere considerate habitat funzionali.

“Random Forests” è una delle tecniche di apprendimento automatico più utilizzate. Questa tecnica, che si basa sull'uso di alberi decisionali, è stata criticata negli ultimi anni perché dà maggiore importanza alle variabili con più valori o classi possibili. In questo contesto, è nata la tecnica delle “Conditional Inference Forests”, il cui approccio riduce il pregiudizio di selezione delle variabili utilizzando un criterio di divisione in due fasi durante la costruzione degli alberi decisionali che compongono la foresta di predittori.

In particolare, l'obiettivo del progetto consiste nel confrontare le prestazioni di entrambe le tecniche quando vengono utilizzate per prevedere l'abitabilità di tre specie ittiche (giovani esemplari di Lethenteron Zanandreai ed esemplari adulti di Padogobius Bonelli e Phoxinus Lumaireul) lungo diversi fiumi situati in Piemonte ed Emilia-Romagna, in Italia. Il progetto studia anche gli effetti del bilanciamento dei dati di input prima di utilizzarli per costruire i modelli.

Per raggiungere gli obiettivi del progetto, è stato scritto un codice basato sul linguaggio di programmazione R. Il codice è stato sviluppato con RStudio, un ambiente di sviluppo integrato per R, che dispone di un numero considerevole di pacchetti (come Boruta, partykit e cforest), che sono stati utilizzati per implementare le tecniche Random Forests e Conditional Inference Forests; così come per calibrare i diversi modelli sviluppati e confrontarne le prestazioni, tra gli altri compiti.

Infine, dopo aver rappresentato e confrontato i risultati, il progetto ha concluso che i modelli che hanno ottenuto le migliori prestazioni sono stati quelli basati sull'approccio dell'inferenza condizionale e i cui dati di input sono stati precedentemente bilanciati con un metodo di sovracampionamento; questi modelli hanno raggiunto gli errori più bassi e le sensibilità e specificità più elevate.

RESUMEN

Los sistemas naturales de agua dulce se ven ampliamente afectados por las presiones hidromorfológicas, las cuales derivan principalmente de la acción humana en el medio ambiente y de el consumo excesivo de los recursos naturales. Para proteger dichos sistemas, la Unión Europea creó en 2000 la Directiva Marco del Agua, cuyo principal objetivo consiste en mejorar el estado ambiental de las masas de agua dulce mediante Planes de Gestión de Recursos Hídricos de Cuenca.

En el caso de los ríos, existe una relación directa entre su estado ecológico y la presencia o ausencia de determinadas especies de peces. Las técnicas de aprendizaje automático son herramientas de gran valor que permiten estudiar la relación entre la distribución de las especies de peces y los descriptores que caracterizan los mesohábitats, áreas fluviales de 10^{-1} - 10^{-3} metros que pueden considerarse hábitats funcionales.

"Random Forests" es una de las técnicas de aprendizaje automático más utilizadas. Esta técnica, la cual se basa en el uso de árboles de decisión, ha sido criticada en los últimos años debido a que dota de mayor importancia a aquellas variables con más valores o clases posibles. En este contexto, nace la técnica de "Conditional Inference Forests", cuyo enfoque reduce el sesgo de selección de variables mediante la utilización de un criterio de división en dos pasos durante la construcción de los árboles de decisión que componen el bosque de predictores.

Más concretamente, el objetivo del proyecto consiste en comparar el rendimiento de ambas técnicas cuando se utilizan para predecir la habitabilidad de tres especies de peces (ejemplares jóvenes de Lethenteron Zanandreai y ejemplares adultos de Padogobius Bonelli y Phoxinus Lumaireul) a lo largo de varios ríos situados en Piamonte y Emilia-Romaña, Italia. El proyecto también estudia los efectos de equilibrar el conjunto de datos de entrada previamente a su uso para construir los modelos.

Para lograr los objetivos del proyecto, se ha escrito un código basado en el lenguaje de programación R. El código se ha desarrollado con RStudio, un entorno de desarrollo integrado para R y el cual dispone de un considerable número de paquetes disponibles (como *Boruta*, *partykit* y *cforest*), los cuales han sido utilizados para implementar las técnicas de "Random Forests" y "Conditional Inference Forests"; así como para calibrar los diferentes modelos desarrollados y comparar sus rendimientos, entre otras tareas.

Finalmente, tras representar y comparar los resultados, el proyecto concluyó que los modelos que alcanzaron los mejores rendimientos fueron aquellos basados en el enfoque de Inferencia Condicional y cuyos datos de entrada habían sido previamente equilibrados con un método de sobremuestreo; dichos modelos alcanzaron los menores errores y las mayores sensibilidades y especificidades.

RESUM

Els sistemes naturals d'aigua dolça es veuen àmpliament afectats per les pressions hidromorfològiques, les quals deriven principalment de l'acció humana en el medi ambient i del consum excessiu dels recursos naturals. Per a protegir aquests sistemes, la Unió Europea va crear en 2000 la Directiva Marc de l'Aigua, el principal objectiu de la qual consisteix a millorar l'estat ambiental de les masses d'aigua dolça mitjançant Plans de Gestió de Recursos Hídrics de Conca.

En el cas dels rius, existeix una relació directa entre el seu estat ecològic i la presència o absència de determinades espècies de peixos. Les tècniques d'aprenentatge automàtic són eines de gran valor que permeten estudiar la relació entre la distribució de les espècies de peixos i els descriptors que caracteritzen els mesohàbitats, àrees fluvials de 10^{-1} - 10^{-3} metres que poden considerar-se hàbitats funcionals.

"Random Forests" és una de les tècniques d'aprenentatge automàtic més utilitzades. Aquesta tècnica, la qual es basa en l'ús d'arbres de decisió, ha estat criticada en els últims anys pel fet que dota de major importància a aquelles variables amb més valors o classes possibles. En aquest context, neix la tècnica de "Conditional Inference Forests", l'enfocament de la qual redueix el biaix de selecció de variables mitjançant la utilització d'un criteri de divisió en dos passos durant la construcció dels arbres de decisió que componen el bosc de predictors.

Més concretament, l'objectiu del projecte consisteix a comparar el rendiment de totes dues tècniques quan s'utilitzen per a predir l'habitabilitat de tres espècies de peixos (exemplars joves de Lethenteron Zanandreai i exemplars adults de Padogobius Bonelli i Phoxinus Lumaireul) al llarg de diversos rius situats a Piemont i Emília-Romanya, Itàlia. El projecte també estudia els efectes d'equilibrar el conjunt de dades d'entrada prèviament al seu ús per a construir els models.

Per a aconseguir els objectius del projecte, s'ha escrit un codi basat en el llenguatge de programació R. El codi s'ha desenvolupat amb RStudio, un entorn de desenvolupament integrat per a R i el qual disposa d'un considerable nombre de paquets disponibles (com *Boruta*, *partykit* i *cforest*), els quals han estat utilitzats per a implementar les tècniques de "Random Forests" i "Conditional Inference Forests"; així com per a calibrar els diferents models desenvolupats i comparar els seus rendiments, entre altres tasques.

Finalment, després de representar i comparar els resultats, el projecte va concloure que els models que van aconseguir els millors rendiments van ser aquells basats en l'enfocament d'Inferència Condicional i les dades d'entrada de la qual havien estat prèviament equilibrades amb un mètode de sobremostrejo; aquests models van aconseguir els menors errors i les majors sensibilitats i especificitats.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1. INTRODUCTION

Freshwater is a crucial resource for humanity; it is not only used for consumption or to meet human physiological needs but consolidates one of the main pillars of economy and society. In addition, the importance of water in human systems has been highlighted in recent decades due to population and industrial activity growth. Moreover, the relevance of freshwater for natural systems is undeniable since more than fifty percent of word biodiversity is related to it. Therefore, protecting and maintaining water sources is a priority, as water conditions affect human life and its development as a community, as well as the natural ecosystems.

However, in the last decades, significant changes have affected water sources. On the one hand, humans have produced changes in the climate system and have damaged the environment by most of their activities: increasing emissions and discharges to water bodies, atmosphere, and soil; expansive urban, agriculture, and industrial development; use of pesticides and fertilizers, changes on the water catchments, etc. On the other hand, as population and human activities are increasing, water demand and dependence are growing too, leading to an overconsumption of water reservoirs. In order to protect water quality and availability and the ecosystems that depend on it, European Union actions are a must; isolated measures will never be successful: a coordinated response of all European Union countries is needed.

Under this premise, in 2000, the EU Water Framework Directive (WFD) was born [1]. The WFD's main aim is to protect inland surface waters, transitional waters, coastal waters, and groundwater [2], as well as to achieve a "good environmental status" for all artificial and heavily modified water bodies [3]. The directive defines the ecological status as "the quality of the structure and functioning of aquatic ecosystems associated with surface waters" [4], and it can be classified as high, good, moderate, poor, or bad depending on biological, hydromorphological, chemical, and physicochemical elements: the indications to type it are detailed in directive's Annex V [5]. With regards to surface water bodies, their environmental status and the functioning of their ecosystems are intimately related to their hydrological regime [6], and so to the ecological flows (Eflows), i.e. "the amount of water required for the aquatic ecosystem to continue to thrive and provide the services we rely upon" [7].

In order to accomplish its objectives, the WFD follows a river basin approach, which establishes that each of the member states will analyze the characteristics, human repercussions, and economic importance of each of the river basin districts (RBD) located in its territory by following the methodologies specified in the directive's Annexes II and III [8]. Each member state must also create River Basin Management Plans (RBMPs) for each RBD located in its territory, and they must review and update them every six years [9].

In this context, the habitat suitability models (HSMs) consolidate a valuable tool for the RBMPs to study rivers' ecological flow, quantify fish species' habitat requirements, and understand and predict temporal and spatial patterns of fish species distribution along the river. HSMs can consider abiotic and/or biotic variables to carry out their predictions and follow a micro-scale or a meso-scale approach [10].

More specifically, in 2007, Mouton proved that the importance of the biotic interactions should be considered when studying fish species distributions [11]. So, meso-scale resolution models play an important role in habitat suitability assessment, as they take into consideration the combined effect of abiotic and biotic interactions [12]; it would be essential to note that mesohabitats are river areas of 10^{-1} - 10^{-3} meters where fish species can be observed there during a noteworthy part of their diurnal routine [13]. Mesohabitats can also be referred to Hydromorphological units (HMUs).

This project uses two different machine learning techniques based on decision trees (Random Forest-Random Inputs and Conditional Inference Forest) to build different habitat suitability models and compare their performance. This process was implemented to study three fish species: Lethenteron zanandreai, Padogobius Bonelli, and Phoxinus lumaireul.

Every model was developed and compared using RStudio, an integrated development environment for the R programming language.

2. MATERIAL AND METHODS

2.1. Study area

The studied area consists of the mountainous regions of Piedmont and Emilia-Romagna, located in Italy. Piedmont is part of the Alps, the most extensive range system located entirely in Europe [14]; the French and Italian regions that are part of the alps are known as the Alps-Mediterranean Euroregion, whose territory covers 110,460 km^2 [15].

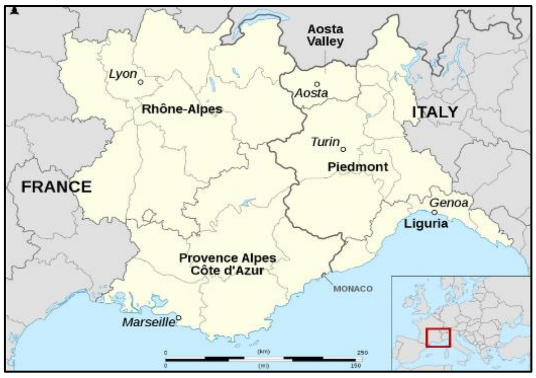


Figure 1: Alps-Mediterranean Euroregion

On the one hand, the Piedmont region stands out because of its mountainous relief, which covers 43.3% of its territory and has three sides surrounded by the Alps [16]. Monte Viso is one of the Alps mountains in Piedmont; at its peak, River Po is born, the most prominent Italian river and whose tributaries consolidate the project scope [17]. Piedmont's climate is humid subtropical at lower altitudes and alpine at higher ones; due to the mix of climates, Piedmont's waters beds are characterized by having little snowpack reservoirs, a high level of evapotranspiration, and low flows during summer [18].

On the other hand, Emilia-Romagna is mainly characterized by its plains, which cover 48% of the region's surface; its hills and mountains cover 27% and 25%, respectively [19]. The Apennine range completely delimits the western border of the region. Its climate is mainly humid subtropical in most regions; its rivers have large flows due to intense precipitations occurring all the year [20].

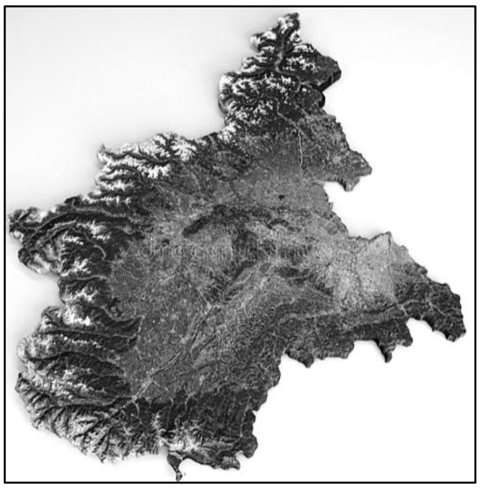


Figure 2: Piedmont region [21].

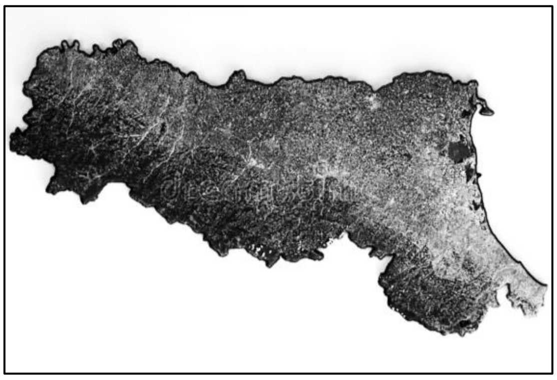


Figure 3: Emilia-Romagna region [22].

2.2. Habitat descriptors

The data provided corresponds to the following tributaries: Belbo, Chisola, Ghiandone, Orco and Sangone in Piedmont; and Trebbia in west Emilia-Romagna.

At each tributary, the available data cover at least ten hydromorphological units (HMUs), which are characterized by various habitat descriptors. The descriptors considered in the model development are the river slope, the river longitudinal connectivity, the kind of cover, the water depth, the flow velocity, and the substrate size. It would be essential to highlight some characteristics of the different variables:

- The river longitudinal connectivity is a binary variable that evaluates if there are obstacles to fish movements along the river [23].
- The substrate size range classifies as it is shown in the following table:

Substrate	**Size range**
Gigalithal	Bedrock
Megalithal	>40cm
Macrolithal	20 – 40 cm
Mesolithal	6 - 20 cm
Microlithal	2 – 6 cm
Akal	0.2 - 2 cm (gravel)
Psammal	0.06 - 0.2 cm (sand)
Pelal	<0.06 cm (clay)
Detritus	organic matter
Xylal	woody debris
Sapropel	anoxic mud
Phytal	submerged plants

Table 1: Substrate size ranges.

- To consider the variability of water depth, water velocity, and substrate size through each HMU, at least seven different points were measured. Those measurements were used to split each variable into frequency categories (see Table 2) [24].
- The kind of cover was divided into ten different binary variables (No/Yes) and determined visually (see Table 2).

In summarizing, the data input has the following structure:

Feature sampled	Units	Annotation	Variables	
Slope	Parts per unit		○ SLOPE	
River connectivity	Binary (yes/not)		○ CONNECTIV	
Cover type	Binary (yes/not)		○ BOULDER ○ CANOP_SHAD ○ OVERHA_VEG ○ ROOTS ○ SUBMER_VEG	○ EMERG_VEG ○ UNDERC_BAN ○ WOODY_DEBR ○ RIPRAP ○ SHALL_MARG
Water depth	Parts per unit of random samples	Each frequency category/variable measures a 15 cm depth increase	○ D_15 ○ D15_30 ○ D30_45 ○ D45_60 ○ D60_75	○ D75_90 ○ D90_105 ○ D105_120 ○ D_120
Flow velocity	Parts per unit of random samples	Each frequency category/variable measures a 15 cm/s depth increase	○ CV_15 ○ CV15_30 ○ CV30_45 ○ CV45_60 ○ CV60_75	○ CV75_90 ○ CV90_105 ○ CV105_120 ○ CV_120
Substrate size	Parts per unit of random samples		○ GIGALITHAL ○ MEGALITHAL ○ MACROLITHAL ○ MESOLITHAL ○ MICROLITHAL ○ AKAL	○ PSAMMAL ○ PELAL ○ DETRITUS ○ XYLAL ○ SAPROPEL ○ PHYTAL

Table 2: Input data variables.

2.3. Fish species

The fish species covered in this project are Lethenteron Zanandreai (young specimens), Padogobius Bonelli (adult specimens), and Phoxinus Lumaireul (adult specimens). To study their presence along the tributaries, each HMU was isolated with nets, and fishes were captured and classified [24].

2.3.1. Lethenteron Zanandreai

The common name of this specie is Lampreda Padana; its distribution area is limited to the river basins of the northern Adriatic [25].

The species spends most of its life in the larval state; in this state, the species is blind; during this period, it is found in rivers with moderate currents and soft substrate where it can bury itself. When metamorphosis occurs, the males cannot feed, and their only function is reproductive; in this state, specimens move to a river stream, where they reproduce. Once the eggs hatch, the current carries the larvae to areas with moderate currents [25].

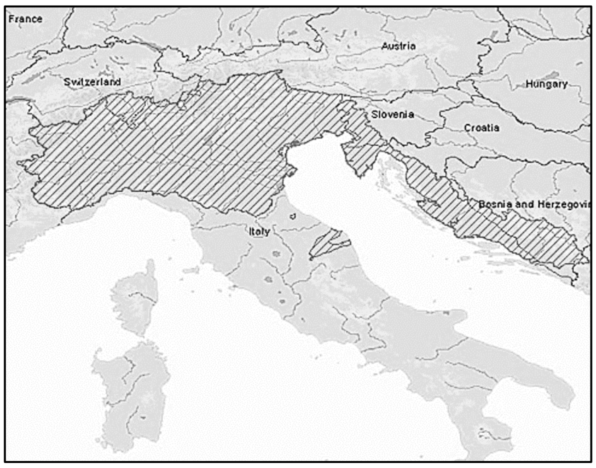


Figure 4: Lethenteron zanandrea habitat [26].

Specimens have an eel-shaped body that can reach 20 cm in length. The body tones are greyish, green, or bluish, and the belly color is lighter than the rest of its body. The Lampreda Padana has a large circular mouth, which is more prominent in males. Their gill slits stand out too, as they are arranged in a row; there are seven gill slits on each side of the body. Its dorsal fins are small, and the anal fin is only present in females [25].

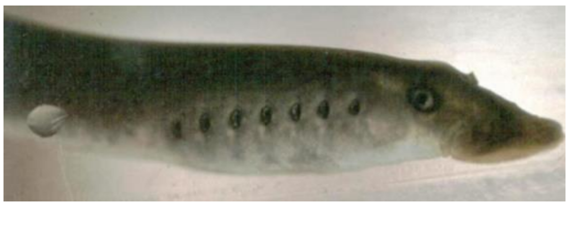


Figure 5: Lethenteron zanandrea [25].

2.3.2. Padogobius Bonelli

The Padogobius Bonelli, also known as Padanian goby, is a freshwater fish naturally located in the north of Italy rivers, as well as some rivers of Croatia, Slovenia, and south of Switzerland [27].

This specie can usually be found in shallow rivers with moderate current and areas with big gravel substrates. The specimens use the openings under the boulders as shelter and nest during the reproductive period, from May to July [27].

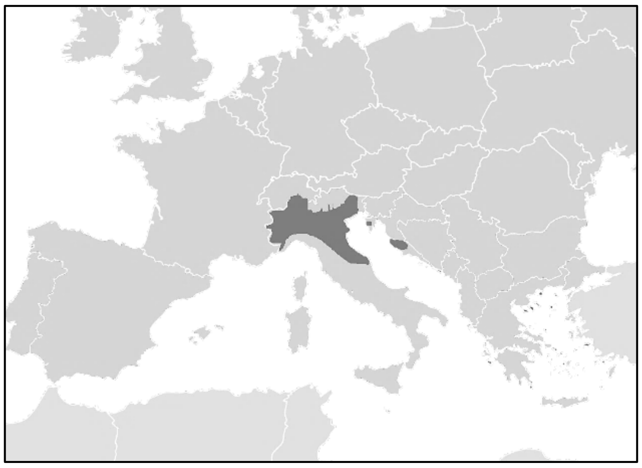


Figure 6: Padogobius bonelli habitat [27].

Regarding its aspect, the specimens have small brown scales and measure 6-7 cm, reaching 9 cm in rare situations. Like the typical goby, they are characterized by a large head with a slanted mouth and fleshy lips. It also has connected ventral fins and two separate dorsal fins. The first of the dorsal fins is spiny and has a dark gray horizontal line in the middle, while the second is softer and finer, as the rest of the fins [27].

Males are more prominent than females, and their body color turns almost black during the reproduction season [27].

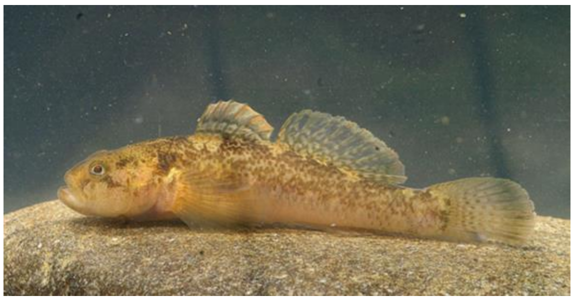


Figure 7: Padogobius bonelli [27].

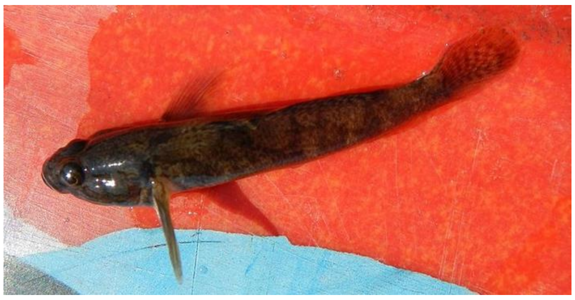


Figure 8: Padogobius bonelli male during reproduction season [28].

2.3.3. Phoxinus Lumaireul

The Phoxinus Lumaireul is located in the Adriatic zone, from the Po River basin to the Drin River basin [29].

It is found in cold-water rivers whose substrates can mix crushed stone, sand, and gravel, where the eggs are disposed of during reproduction. Specimens resist strong currents for long periods so that they can be found in river streams. In addition, the specie makes short rest periods in areas of the river with calmer currents and even pre-alpine lakes [29].

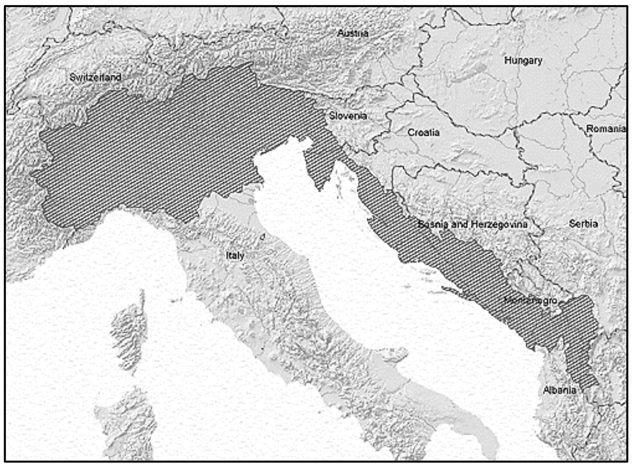


Figure 9: Phoxinus lumaireul habitat [30].

Regarding its appearance, the specimens have an elongated and cylindrical body, entirely covered with barely visible cycloid scales, except for the ventral area. Its head is rounded and proportionally large, and its large snout stands out [29].

Its body color is golden-bronze, and it is covered with dark spots that form longitudinal stripes on the highest parts of the body. The ventral area has lighter colors [29].

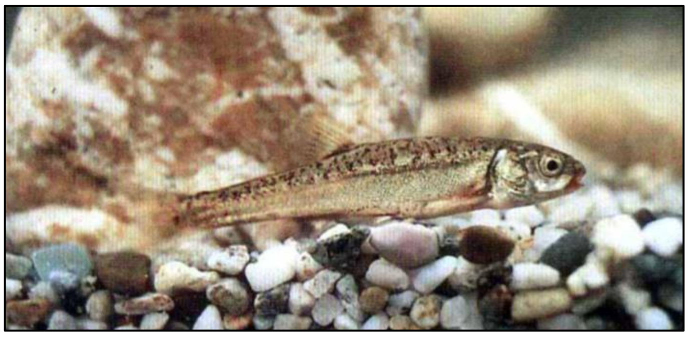


Figure 10: Phoxinus lumaireul [30].

2.4. Introduction to decision-tree-based methods

The project's main work involved developing an R code to study how habitat descriptors affect fish habitability along each HMU and to create reliable prediction models for fish presence using machine learning techniques.

The machine learning techniques proposed in this project are ensembled with decision trees, a primary modeling tool for regression and classification problems submitted by Leo Breiman in 1984 [31]. Decision trees are also well-known as CART (Classification And Regression Trees).

A decision tree model is composed of nodes, and its main structure looks like an upside-down tree. There are different kinds of nodes depending on their functions:

- Root node: The first node of the tree is called the root and is located on its top. The root proposes a binary decision rule that splits the input dataset into two groups: if the data verify the requirement submitted by the node, the data will continue to the left branch, otherwise, to the right.
- Splitting nodes: They are other binary decision nodes that may offer new and different data division conditions after the root.
- Leaf node or non-splitting nodes: If a node does not offer a decision criterion, it is called a terminal node or leaf. These nodes are labeled with the value that the response variable would have if it would accomplish all the criteria established by the tree in its previous branches.

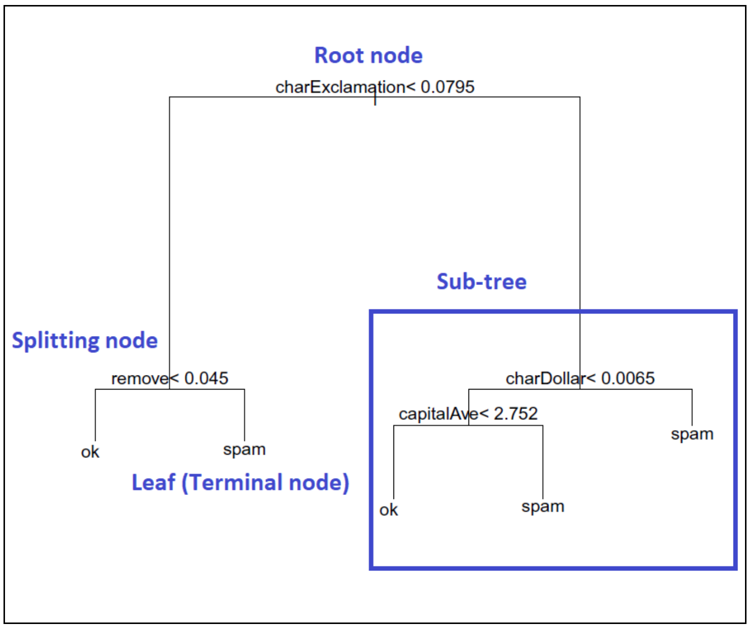


Figure 11: Decision tree structure [32].

The datasets available to build and test a prediction model are called samples. Samples are subsets of observations obtained from a population.

2.4.1. The bootstrap method and the Out-Of-Bag error

The bootstrap method is a widely used technique to evaluate a machine learning model's skill when estimating population statistics [33]. This method resamples the available dataset by randomly replacing some population observations. The observations not included in the bootstrap sample are called the out-of-bag samples (OOB) [34].

The bootstrap method builds the decision tree model by using the bootstrap sample as training data; then, the model prediction ability is estimated by applying the model to the OOB sample. Then, the Out-Of-Bag error (OOB error) is calculated as follows:

- Regression problem:

$$\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y_i}\right)^2$$

Equation 1: Definition of OOB error for regression problems [35].

- Classification problem:

$$\frac{1}{n}\sum_{i=1}^{n} 1_{Y_i != \widehat{Y_i}}$$

Equation 2: Definition of OOB error for classification problems [35].

Where *n* is the number of model variables, Y_i is the response variable real value, and $\widehat{Y_i}$ is the output value of the model.

The OOB error is expressed as a percentage.

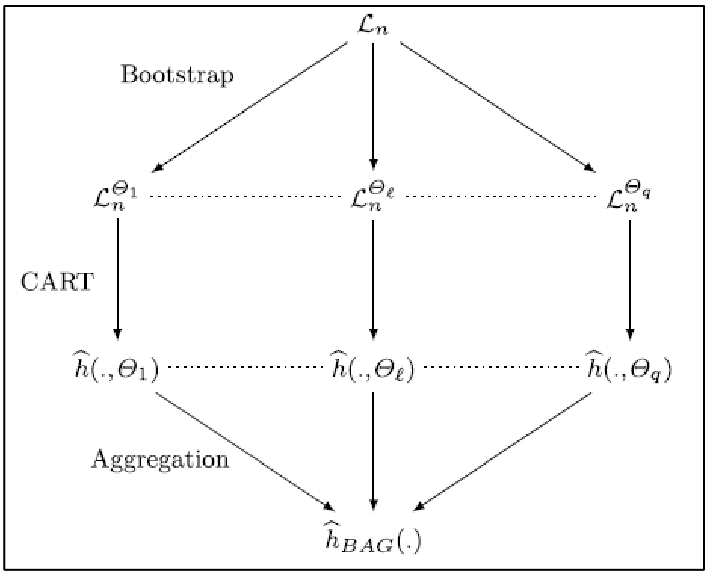
2.4.2. Trees instability

A machine learning technique is unstable when small changes in its training data will make the predictor generated substantially differ. Thus, an unstable model will provide significantly different classifiers when the data sample used changes [36].

CARTs are typically unstable, as the decision tree generated will widely change when different bootstrap samples are considered. This condition reduces decision tree prediction power [37].

However, there are different methods to reduce the instability problem. For example, using different randomly generated bootstrap samples stabilizes CART trees through aggregating the predictors generated; this technique is called Bagging or Bootstrap-aggregating method. Of course, the Bagging technique is less effective when the learning algorithm is already stable. [37]

The following scheme illustrates the primary performance of the Bootstrap-aggregating method:


$\mathcal{L}_n$
Bootstrap
$\mathcal{L}_n^{\Theta_1}$
$\mathcal{L}_n^{\Theta_\ell}$
$\mathcal{L}_n^{\Theta_q}$
CART
$\widehat{h}(.,\Theta_1)$
$\widehat{h}(.,\Theta_\ell)$
$\widehat{h}(.,\Theta_q)$
Aggregation
$\widehat{h}_{BAG}(\cdot)$

L_n: Learning sample

L_{n^Θ}: Bootstrap sample

$\hat{h}(.,\Theta)$: Tree predictor

$\hat{h}_{BAG}$: Aggregated predictor

Figure 12: Bootstrap aggregating method. [37]

Forest algorithms, discussed in the following paragraphs, can also reduce the instability problem by creating tree ensamples.

2.5. Random Forest

Both machine learning techniques used to develop this project are based on the random forests technique, which groups a set of decision trees randomly perturbed and aggregates them to create a new predictor. Leo Breiman defined it in 2001 as follows:

"A random forest is a classifier consisting of a collection of tree-structured classifiers {h(x,Θ_q), q=1, ...} where the {Θ_q} are independent identically distributed random vectors, and each tree casts a unit vote for the most popular class at input x "[38], being *q* the total number of tree predictors built.

It would also be essential to highlight that, in most random forest methods, Θ_q are a set of random variables independent of the learning sample used (Ln) [39].

The aggregation of the different tree predictors is made as shown:

- Regression problem:

$$\hat{h}_{rf}(x) = \frac{1}{q}\sum_{l=1}^{q} \hat{h}(x, \Theta_l)$$

Equation 3: Definition of random forest predictor for regression problems. [39]

- Classification problem:

$$\hat{h}_{rf}(x) = \arg max_{1\le c\le C} \sum_{l=1}^{q} 1_{\hat{h}(x,\Theta_l)=c}$$

Equation 4: Definition of random forest predictor for classification problems. [39]

2.5.1. Random Forest Random Inputs

There are several variants of the Random Forest method. One of the most widely used is Random Forests-Random Inputs, a method proposed by Leo Breiman in 2001 too [38]. From now on, this method will be shortened as RF.

It is called random inputs because it uses a small group of randomly selected input variables at each node while growing the decision trees that consolidate the forest; these trees are not pruned (the method does not select a subtree but the complete one), and they may not be optimal. So, not only is each tree perturbed when selecting the testing samples, as the bagging method does, but also at each tree level during the growth [40]. The following scheme shows how the algorithm mainly works:

$$\begin{tikzcd}
 & \mathcal{L}_n \arrow[dl, "\text{Bootstrap}"'] \arrow[d] \arrow[dr] & \\
\mathcal{L}_n^{\Theta_1} \arrow[r, dotted, dash] \arrow[d, "\text{RI tree}"'] & \mathcal{L}_n^{\Theta_\ell} \arrow[r, dotted, dash] \arrow[d] & \mathcal{L}_n^{\Theta_q} \arrow[d] \\
\widehat{h}(.,\Theta_1,\Theta_1') \arrow[r, dotted, dash] \arrow[dr, "\text{Aggregation}"'] & \widehat{h}(.,\Theta_\ell,\Theta_\ell') \arrow[r, dotted, dash] \arrow[d] & \widehat{h}(.,\Theta_q,\Theta_q') \arrow[dl] \\
 & \widehat{h}_{RF-RI}(\cdot) &
\end{tikzcd}$$

L_n: Learning sample

$L_{n^{\Theta}}$: Bootstrap sample

$\hat{h}(.,\Theta)$: RI tree predictor

$\hat{h}_{RF-RI}$: Final predictor

Figure 13: Random Forests-Random Inputs method. [41]

Thanks to this additional randomness, the collection of tree predictors is more diverse, leading to an improved prediction capacity compared to other methods such as Bootstrapping-aggregation. Moreover, this method is also faster and more robust to noise and outliers [40].

The two most important parameters of the method are the number of trees that make up the forest and the number of variables tried to split at each node.

- Number of trees in the forest: This number depends on the complexity of the population analyzed. When the quantity of trees grown is large enough, the model's predictive performance will stabilize and minimize. The number of trees is directly related to the computation time, as larger forests lead to higher computation times. If the number of trees is not large enough, the predictive performance of the method will be limited. [42]

- Number of variables tried to split at each node: At each tree node, the algorithm selects a random subset of input variables, then the node splits. This parameter is important because different values may lead to a different prediction capacity. It is also relevant to highlight that the number of variables tried to split at each node is constant within all node divisions and for all the tree predictors of the forest. [43]

From now on, these variables will be called “ntree” and “mtry”, as these are the names used by the RStudio packages implemented in the code and which will be introduced in the Procedure section.

2.5.2. The splitting rule and the conditional Inference approach

Another essential feature of the decision-trees-based algorithms is the splitting rule used to grow the tree predictors. In the case of RF classification problems, the rule consists of selecting the split that minimizes the Gini index within all the mtry variables. [44]

The Gini index, also known as the Gini coefficient or Gini impurity, is a parameter that measures the probability of a population feature or class being incorrectly classified when it is randomly selected in the nodal division during the predictor growth. This parameter was proposed by Leo Breiman in 1984 [45], and it is defined as:

$$G = \sum_{i=1}^{C} P_t^c \cdot (1 - P_t^c)$$

C: total number of features c.

P_t^c: proportion of observations of class c in the node t.

Equation 5: Gini impurity. [44]

The value of this parameter can vary from 0 to 1:

- G=0: Every element Is related to one class.
- G=1: Every element is randomly distributed between the classes.
- G=0.5: Elements are uniformly distributed between the classes.

However, this splitting rule has one inconvenience; it favors the choice of those variables that may have many possible different splits over those with few partitions. For example, those categorical variables with many other categories will be picked over binary ones. This phenomenon leads to a variable selection bias and harms the model prediction performance. [46]

In order to avoid the variable selection bias, in 2006, Hothorn created a new random forest variant called Conditional inference forests (CIF) [47].

The splitting criteria of this new method consist of two different steps [48]: firstly, the CIF method looks for the best possible split variable, and secondly, it looks for its best splitting point:

- The optimal split variable is determined by carrying out a linear rank association test.
- The best possible split is selected by comparing all possible partitions of the split variable. This second step of the algorithms allows the predictors to improve their performance but increases running time.

2.6. Implementation

The RF and CIF algorithms' performance has been compared using RStudio, an integrated development environment for the R programming language. Six different models were compared and optimized for each fish species considered.

The following subchapters summarize the code’s structure and the most crucial functions used.

2.6.1. RStudio packages

One of the main attractions of the R language is the large number of packages available on the Internet, which include a wide variety of functions, data, and code that the user can utilize on its scripts. It is essential to highlight that packages must not only be installed in RStudio but also be loaded into the workspace.

Thus, the first part of the code consists of initializing the libraries used. In this case, the most important libraries were randomForest, partykit, and Boruta. Next, the functions used, and their main arguments will be explained:

2.6.1.1. randomForest [49]

The package’s functions used in the code are *randomForest*, *importance,* and *partialPlot*.

The *randomForest* function implements the RF algorithm for classification and regression problems. Its most important arguments are:

- data: This argument must be a data frame or matrix containing the model variables.
- x, formula: It must contain or define the response variable of the model; in this case, it is the fish presence.
- ntree: This argument indicates the number of trees in the forest generated. The importance of this parameter has already been discussed in section 2.5.1.
- mtry: It must indicate the number of variables tried to split at each node of the trees. It is the most crucial parameter of the algorithm (section 2.5.1.).
- importance: It must be set as “TRUE” for the function to calculate each input variable’s importance.

The *importance* function allows RStudio to extract the predictors' importance measures from a *randomForest* model obtained previously. The relevant arguments of the function are:

- x: it indicates the *randomForest* model whose importance measures must be extracted.
- type: this argument must be set to 1 for the importance measurement to be based on the mean decrease in the model accuracy.

The *partialPlot* function can be used to plot a partial dependence plot that indicates how each variable of the *randomForest* model affects the response variable. Its essential arguments are:

- x: it indicates the *randomForest* model whose partial plot must be represented.
- x.var: it indicates the input variable of the model whose partial plot must be represented.

2.6.1.2. Partykit [50]

The package's functions used in the code are *cforest, ctree_control*, and *varimp.*

The *cforest* function implements the CIF algorithm to overcome the RF's variable selection bias. Function essential arguments are:

- data: Matrix or data frame with all model variables.
- formula: This argument must define the response variable of the model. It is essential to notice that in a classification problem this variable must be expressed as "factor(response variable) ~.", because the *cforest* function cannot automatically distinguish between regression and classification problems, in contradistinction to *randomForest*.
- ntree: Number of trees in the forest generated.
- mtry: Number of variables tried to split at each node of the trees.
- replace: If it is set as "TRUE", random observations will be selected more than once when creating the model. It may be helpful when input data availability is limited.
- control: It consists of a group of control parameters.

The *ctree_control* function saves a list with the control parameters for *cforest models* generation:

- teststat: It specifies the characteristics of the linear rank association test carried out to select the optimal split variable. When this parameter is set as "max", it considers all the variables per each split; this decision increases the computation time but improves the prediction capacity of the model.
- testtype: This parameter indicates the kind of test to be applied in the split variable selection. When set as "Teststatistic", it applies a raw data statistic test and follows the conditional inference approach.
- mincriterion: this variable controls the partitioning process of each split variable. In order to generate a group of maximal trees, it must be set to 0.

The *varimp* function calculates the variable importance of a *cforest* model by considering its mean decrease in the model accuracy, so it can be compared with the output data of the *importance* function. This function was used with its standard arguments, so it was only needed to indicate the *cforest* object.

- x: it indicates the *randomForest* model whose partial plot must be represented.
- x.var: it indicates the input variable of the model whose partial plot must be represented.

2.6.1.3. Boruta [51]

The input variables to consider in every model were previously selected by using the Boruta method, which was created in 2010 as a package for R. The preselection of the most relevant habitat descriptors (for each fish species) was made to homogenize the prediction models and to reject the unimportant variables in order to reduce the code running time needed.

The Boruta procedure is conformed of two main steps. The method's first step consists of creating randomly shuffling copies of the input variables, also known as "shadow variables", and generating a data frame with all the input and shadow variables. The second step consists of measuring the importance of all the data frame variables and evaluating if the input variable's importance is higher than its "shadow"; if this condition is fulfilled, the Boruta algorithm considers that the input variables make a "hit". These two steps are repeated for an "n" number of trials.

The method's output is a binary variable that indicates the number of hits of each variable per trial; the output vector follows a binomial distribution. Finally, depending on the position of the input variable along the binomial distribution, the variable's importance is classified:

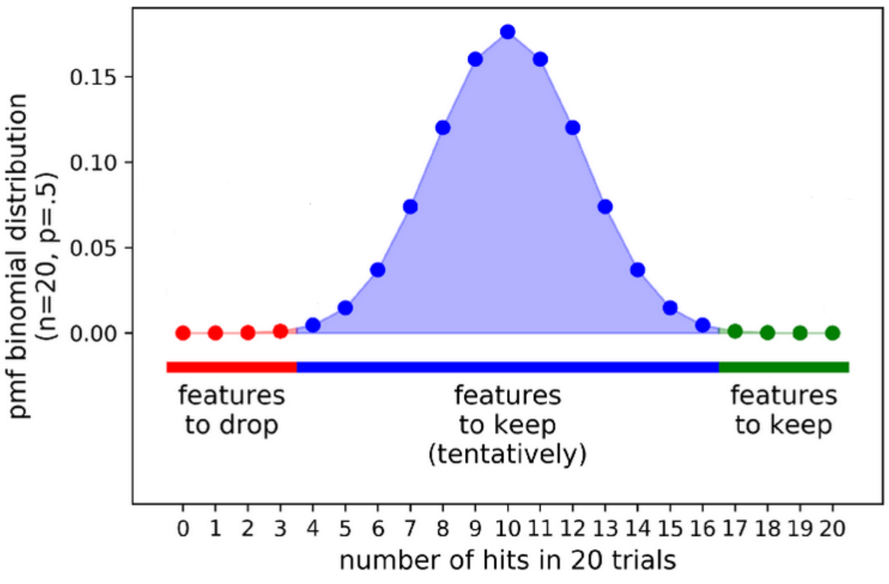


Figure 14: Binomial distribution output of the Boruta algorithm. [52]

The package function that applies the Boruta method is *Boruta*, and its main arguments are:

- x: It must be a matrix or a data frame with all the variables whose importance will be evaluated.
- y: It contains the response variable (the presence of fish).
- maxRuns: The number of times the two-steps process will be applied.

2.6.2. Structure of the code

This section shows the main structure of the code after initializing the packages; for this purpose, flow charts have been developed to show how the data flow through the different processes that occur in the code. These flow charts have been split up to make them clearer.

The complete code is included in Annex II.

2.6.2.1. The first steps

The first step of the code consisted of setting the directory and reading the data available from a .txt document. Then, the variables of interest were extracted and saved in different vectors and matrixes:

- *Iniz.data*: this matrix contains all the habitat descriptors (see Table 2) at each HMU, plus a last column indicating the presence of fish at each HMU.
- *fish.pres*: this vector indicates the presence of fish at each HMU.
- *before.boruta.data*: this matrix contains all the habitat descriptors at each HMU (see Table 2).

Once the data is saved, the *Boruta* function (contained in the *Boruta* package) carries out the Boruta test to study the relationship between the habitat descriptors and the fish presence at each HMU; test results are saved and plotted. Then, the program goes into a loop that selects the variables more related to the fish presence and saves them in the *after.boruta.data* matrix.

The *after.boruta.data* matrix is likely to contain unbalanced data, as the presence of fish will rarely be homogeneously distributed between 1 and 0 (presence may overcome absence and vice versa). To balance the data, two different methods were applied:

- *downSample* function: randomly selected data is deleted to balance the fish presence, so the result matrix (*undersample_model*) is smaller than *after.boruta.data*. This process is called undersampling.
- *upSample* function: randomly selected data is duplicated to balance the fish presence, so the result matrix (*oversample_model*) is bigger than *after.boruta.data*. This process is called oversampling.

The following flow chart shows schematically how all these tasks are done:

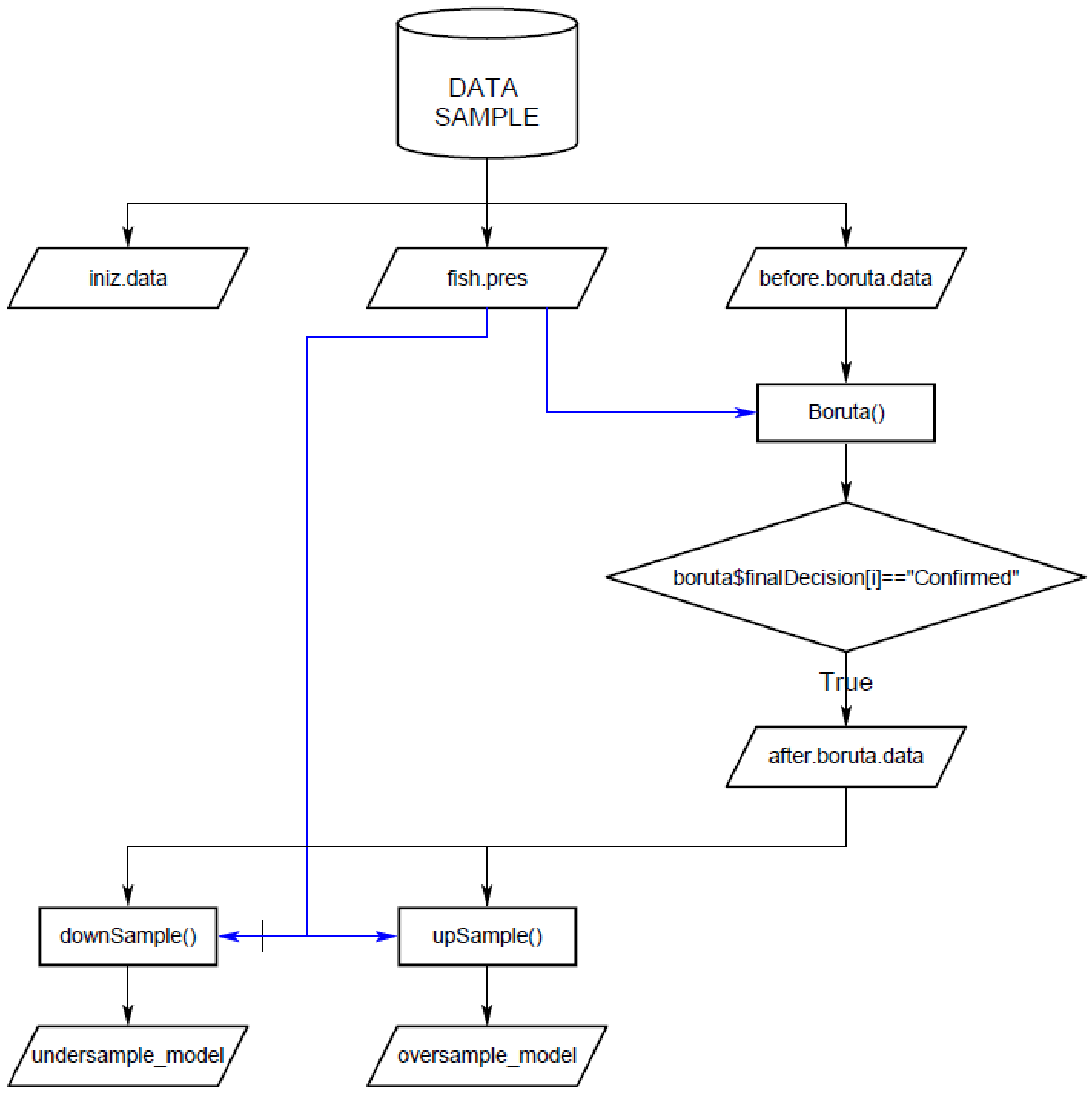


Figure 15: Flow chart of the first steps of the code.

The blue arrows indicate that the fish presence vector (*fish.pres*) is used as a decision factor by the functions *Boruta*, *downSample*, and *upSample*.

2.6.2.2. The creation of the models

The *after.boruta.data*, *undersample_model*, and *oversample_model* matrixes are used to create six different prediction models:

- *RF.normal*: This prediction model is based on RF and created using the randomForest package. The input data is unbalanced and consist of all the HMU features classified as important by the *Boruta* function.
- *RF.over*: This prediction model is also based on RF, but in this case, the input data is the *oversample_model*, which has been previously balanced.
- *RF.under*: This prediction model is based on RF, but in this case, the input data is the *undersample_model*, which has been previously balanced.
- *CIF.normal*: This prediction model is also based on the conditional inference forest approach, and it is created by using the partykit package. The input data is unbalanced and consist of all the HMU features that were classified as important by the *Boruta* function.
- *CIF.over*: This prediction model is also based on CIF, but in this case, the input data is the *oversample_model*, which has been previously balanced.
- *CIF.under*: This prediction model is also based on CIF, but in this case, the input data is the *undersample_model*, which has been previously balanced.

For each fish species, all six models were calibrated with the same parameters. On the one hand, the mtry parameter is set by calculating the root of the number of columns of the model input data and never being lower than two (otherwise, there would not be splitting criteria). On the other hand, the ntree is set to 5000 to ensure every model stabilizes.

In the case of the RF models (*RF.normal, RF.over,* and *RF.under*), the variable importance was calculated using the *importance* function explained in the previous section (*randomForest* package).

The three *randomForest* models were also used to predict the presence of fish at each hydro-morphological unit by using the *predict* function (*stats* package); the predictions obtained were compared with the real indicators of fish presence by using the *confusionMatrix* function (*caret* package), this function provides some important information as the accuracy of the model, its sensibility, the out-of-bag error, etc., which will be discussed in the results chapter.

It would also be important to note that the *randomForest* models also contain some interesting information (such as the OOB error) without being necessary to use functions from other packages. However, as *cforest* models do not include this information, and in order to unify the procedure for all the models, the code structure works as follows:

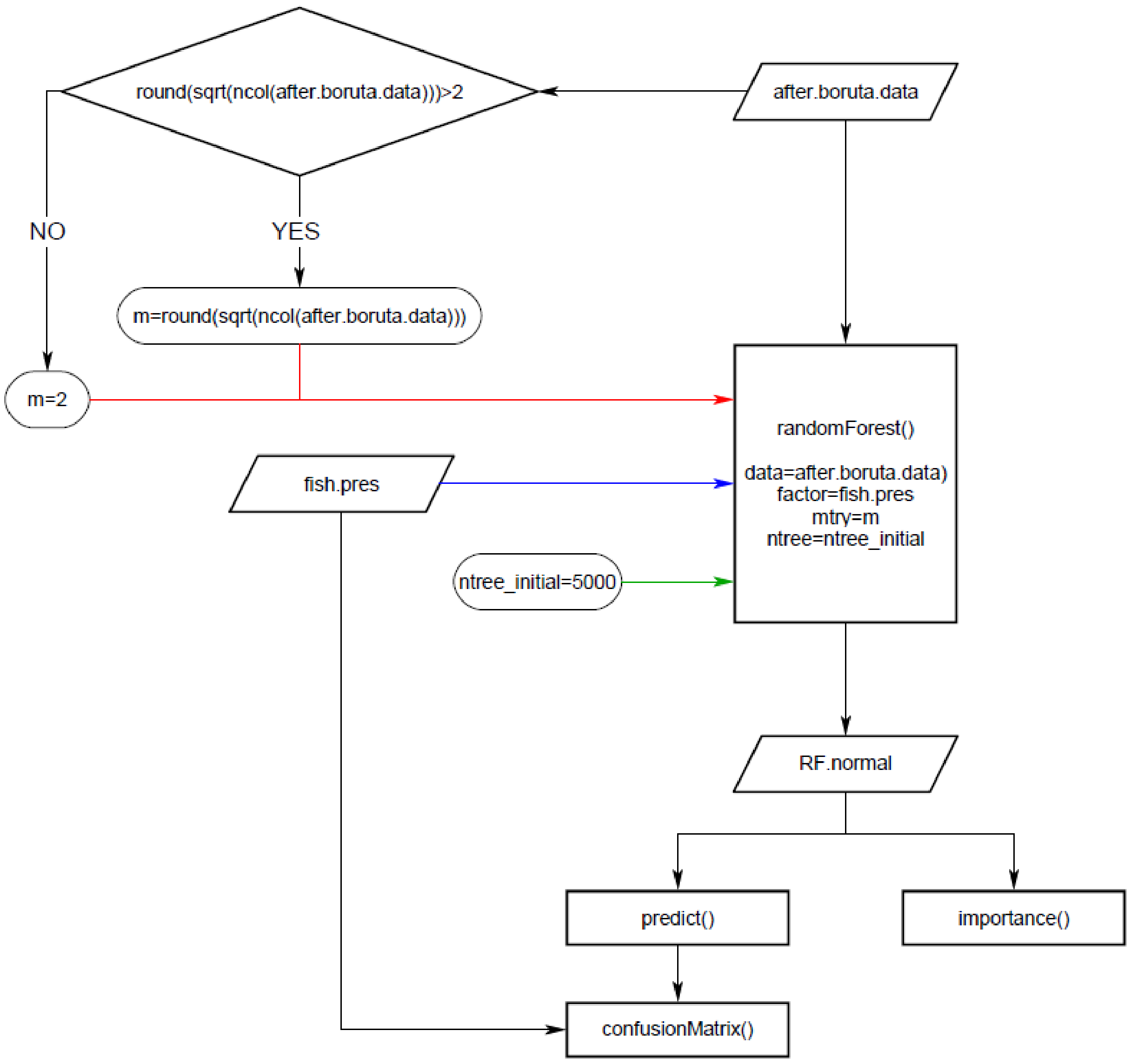


Figure 16: Flow chart of an RF model creation.

The red and green arrows indicate that the *m* and the *ntree_initial* variables were used to set the mtry and ntree parameters, respectively. The blue arrow indicates that *fish.pres* is the response variable of the *randomForest* model.

Regarding the CIF models (*CIF.normal, CIF.over* and *CIF.under*), the variable importance was calculated using the *varimp* function (*partykit* package). The rest of the procedure is identical to the one followed for the *randomForest* models:

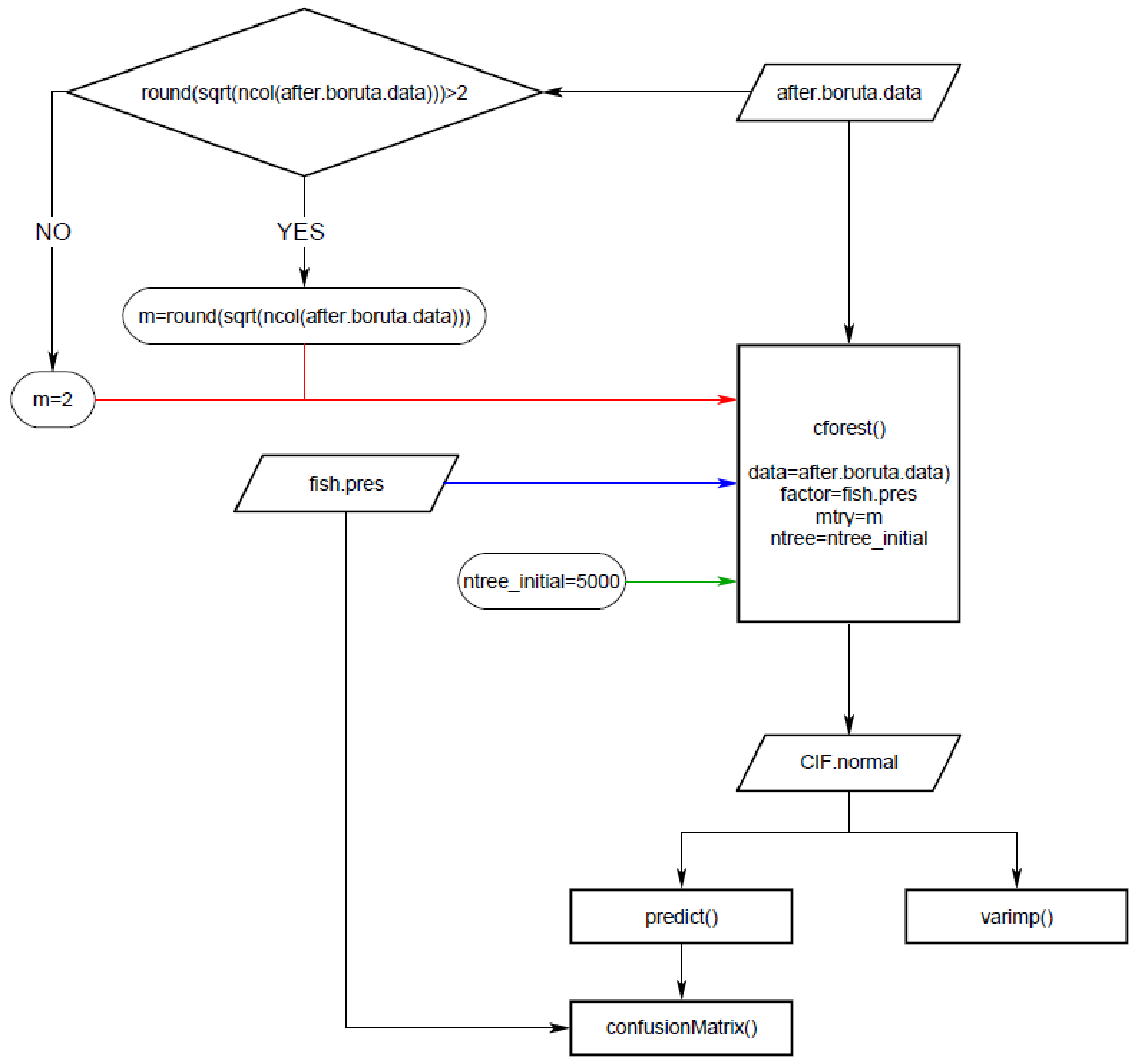


Figure 17: Flow chart of a CIF model creation

2.6.2.3. The optimization of the models

The next part of the code focuses on optimizing each model. First, the code selects the mtry value that minimizes the OOB error; second, it sets the minimum ntree value large enough for the models to stabilize.

On the one hand, in order to select the optimal mtry value, the code creates one model (*RF.normal.loopmtry*) per each possible mtry value employing a loop: the value of mtry can be at least 2 for the nodes to be split; and at most, as large as the number of variables that consolidate the model (*ncol(after.boruta.data)*). Once the code goes over all possible mtry values, it creates a data frame with all the information collected (the OOB error, the sensibility, and the specificity of the model), then the code automatically selects the mtry value that minimizes the OOB error. The schema of the process is the following one:

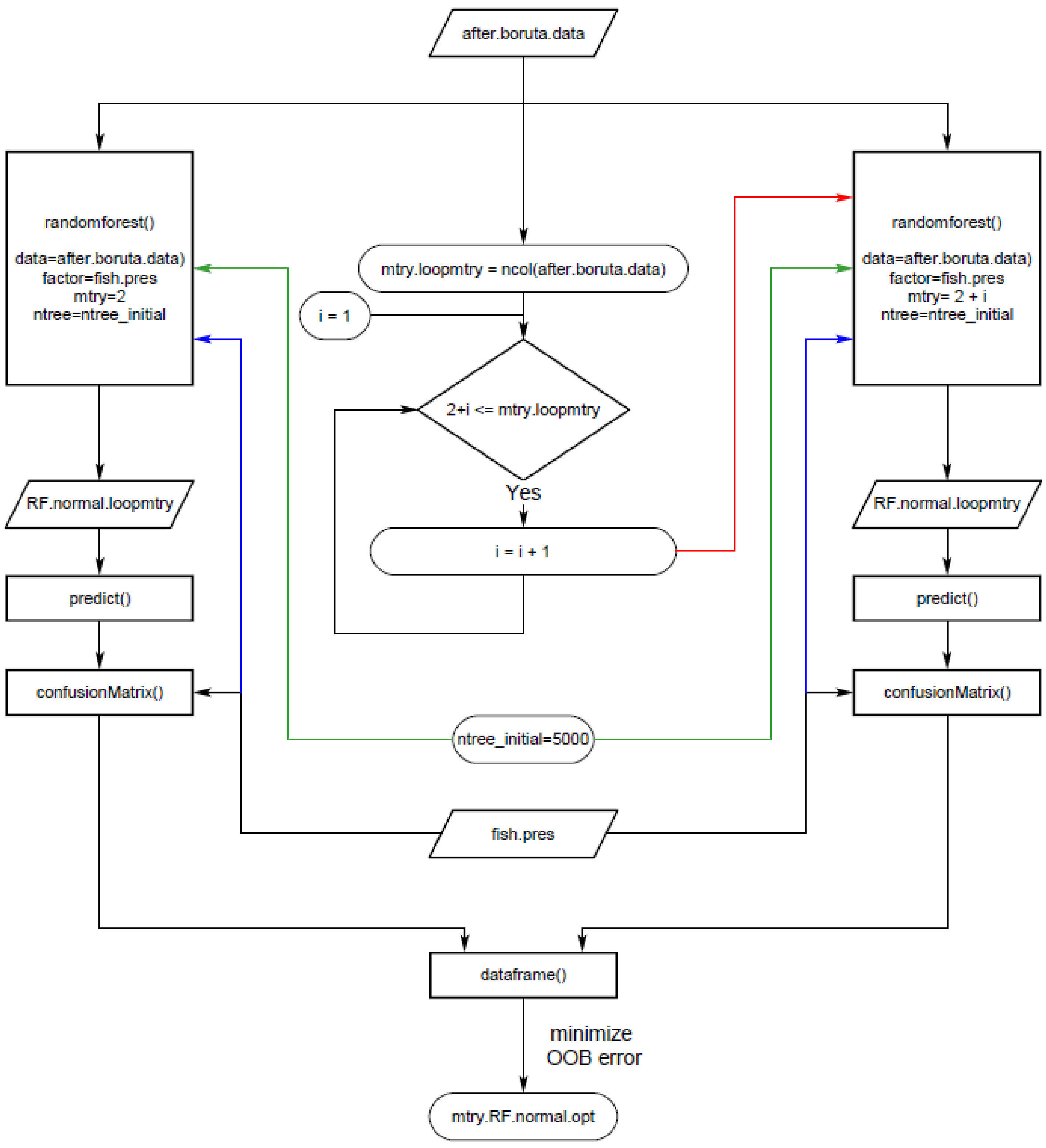


Figure 18: Flow chart of mtry selection procedure in an RF model.

On the other hand, in order to select the minimum feasible value of ntree, the code goes again into a loop and creates as many versions of the model (*RF.normal.loopntree*) as possible ntree values; all the information is saved in a data frame too. However, in the *cforest* models, the ntree loops programmed do not go through every possible ntree value in order to reduce the high computational time required; the loop considers three different counters:

- ntree < 100: the ntree parameter increases one by one (i = i + 1)
- 100 < ntree < 500: the ntree parameter increases by ten (i = i + 10).
- 500 < ntree < 5000: the ntree parameter increases by twenty (i = i + 20).

The process can scheme as follows:

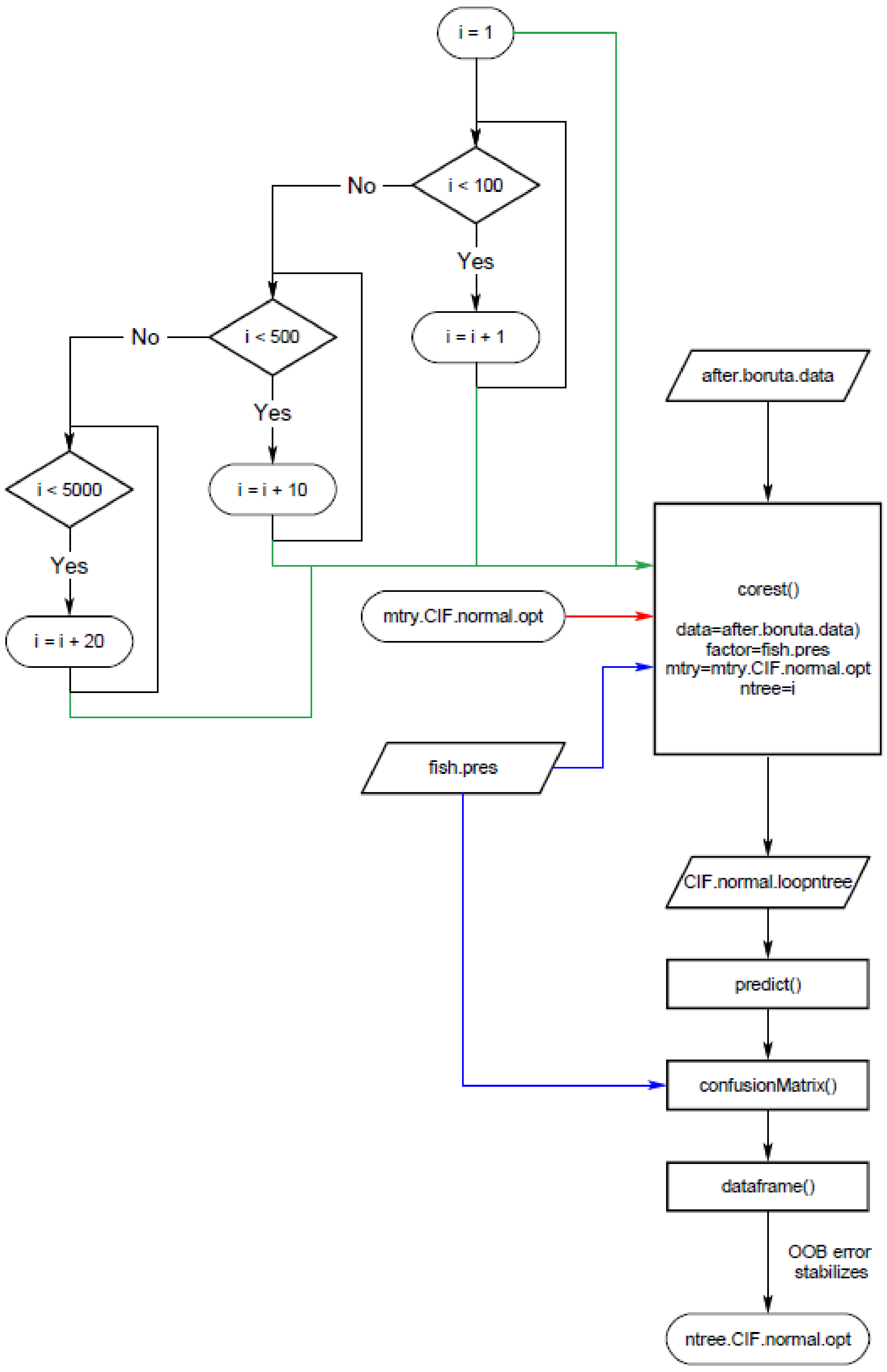


Figure 19: Flow chart of ntree selection procedure in a CIF model

It is important to note that the ntree value that the program saves (ntree.CIF.normal.opt) is not the minimum that stabilizes the OOB error but the double of it, to ensure no transitory is dismissed.

2.6.3. The output data

All the data and models calculated during the code procedure were saved in data frames and matrixes; the last step of the code consists of writing down all this information in an excel document by means of the *openxlsx* package.

Finally, the data was represented with the program "Grapher 10", and the models' results evaluated. A model was considered to have a good performance when:

- The OOB error of the model was inferior to 25%.
- The sensitivity was higher than 0,7 over one.
- The specificity was higher than 0,7 over one.
- The trust statistical skill (TSS) was higher than 0,8 over one, being the TSS calculated as the sum of the sensitivity and the specificity divided per two.

3. RESULTS

The methods previously explained were applied to the three different fish species in order to create their habitat suitability models. The input data used can be download in the links provided in Annex I.

3.1. Lethenteron zanandrea

3.1.1. Input data and variable selection

The Boruta method identified nine different variables as important, being PSAMMAL, which refers to sand substrates, the most important variable; this result was expected, as larvae of Lethenteron Zanandrea use soft substrate to bury themselves. The other important variables identified were AKAL, SHALL_MARG, CV60_75, MICROLITHAL, MESOLITHAL, SLOPE, MACROLITHAL, and D15_30. The rest of the variables were rejected.

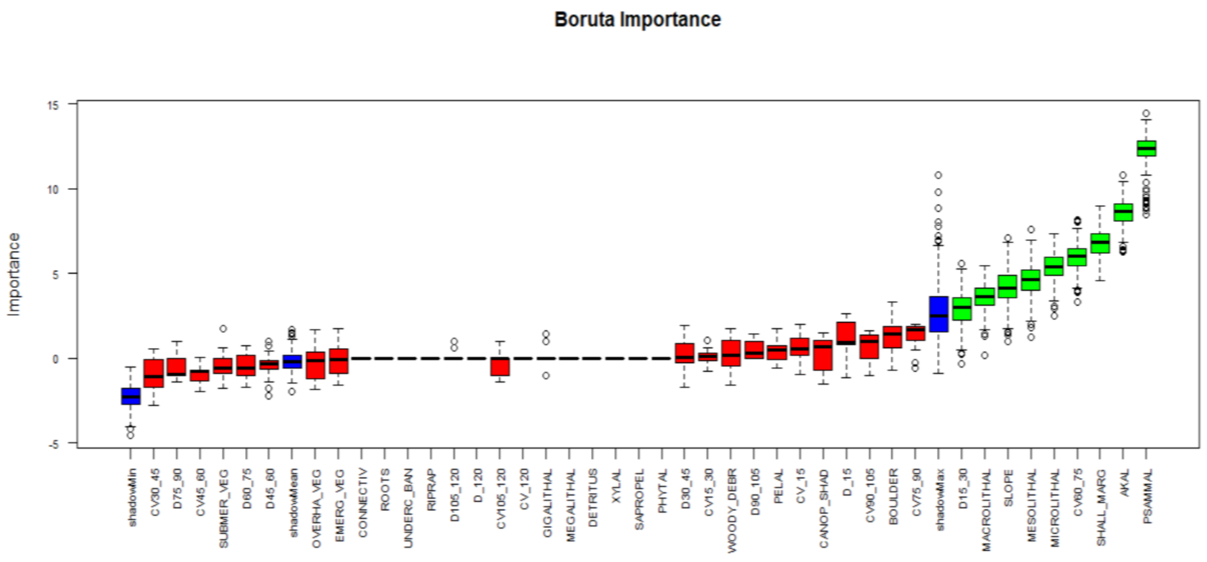


Figure 20: Boruta importance test results.

The remaining data was balanced by means of the oversampling and the undersampling methods.

3.1.2. Habitat suitability models

Then, the model parameters were set equally for all the models; on the one hand, the ntree value was set to 5000, to ensure the stabilization of every model; on the other hand, the mtry value was set to 3, as only 9 variables remained as important ($m = \sqrt{9} = 3$).

In this case only five different habitat suitability models were created (*RF.normal, RF.over, RF.under, CIF.normal, CIF.over)*; the *cforest* function was not able to generate a forest in the case of the undersampled data due to the lack of HMUs.

The importance of the variables was evaluated for every model:

- *RF.normal:*

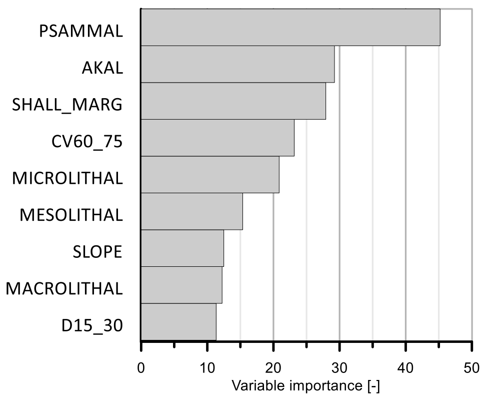


Figure 21: Variable importance measurements in the RF model with unbalanced data.

- *CIF.normal:*

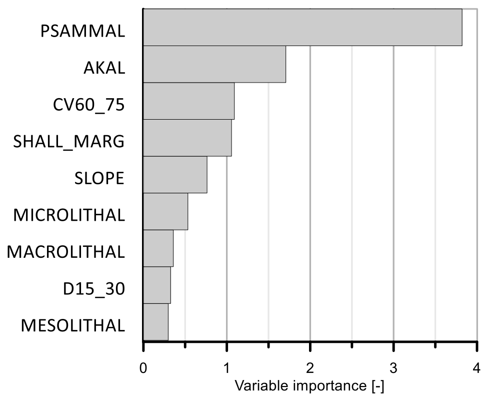


Figure 22: Variable importance measurements in the CIF model with unbalanced data.

- *RF.over:*

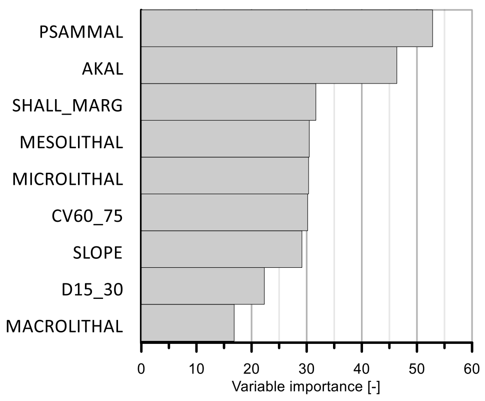


Figure 23: Variable importance measurements in the RF model with oversampled data.

- *CIF.over:*

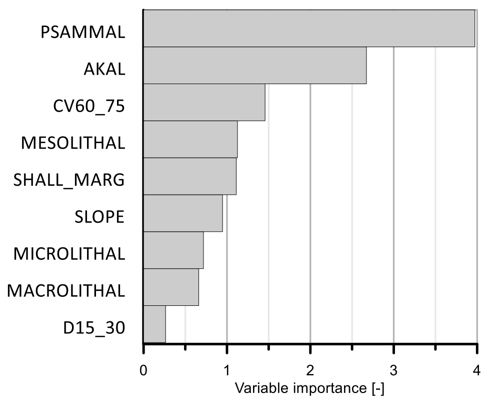


Figure 24: Variable importance measurements in the CIF model with oversampled data.

- *RF.under:*

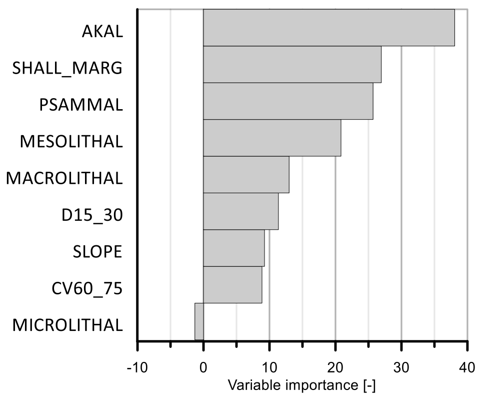


Figure 25: Variable importance measurements in the RF model with undersampled data.

The importance results showed similarities compared to the Boruta test; on the one hand, almost every model identified PSAMMAL and AKAL as the first and the second most important variables, as Boruta did; on the other hand, all the models related the water depth (D15_30) and the biggest substrates (MACROLITHAL), those variables have a lower level of importance. The RF model with undersampled data was the one with the biggest differences compared to the Boruta results.

Next, the five models were used to predict the presence of young specimens of Lethenteron Zanandrea at each HMUs and the out-of-bag error was calculated:

	RF models			CIF models		
	Unbalanced data	**Oversampled data**	**Undersampled data**	**Unbalanced data**	**Oversampled data**	**Undersampled data**
OOB error (%)	6,25	4,35	11,11	3,13	2,17	-

Table 3: OOB error before optimizing the models.

The CIF models achieved the best performances; being the CIF model with oversampled data the one that committed the lowest OOB error. Regarding RF models, the model with oversampled data also had the best performance.

The RF model with undersampled data obtained the highest OOB error.

3.1.3. Optimization of the models

3.1.3.1. Random forest model with unbalanced data

– Calibration of the mtry parameter:

The OOB error, the specificity, the sensitivity, and the TSS varied depending on the mtry value, as it is shown:

- OOB error:

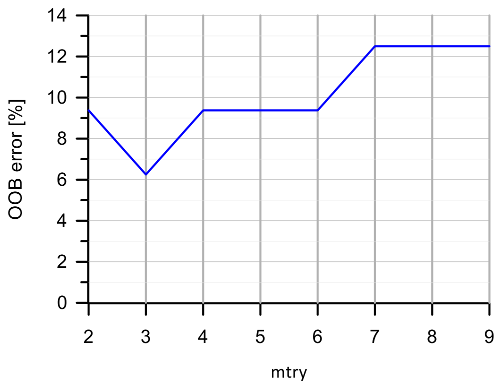


Figure 26: Variation of the OOB error depending on the mtry value during the calibration of the RF model with unbalanced data.

- Specificity, sensitivity and TSS:

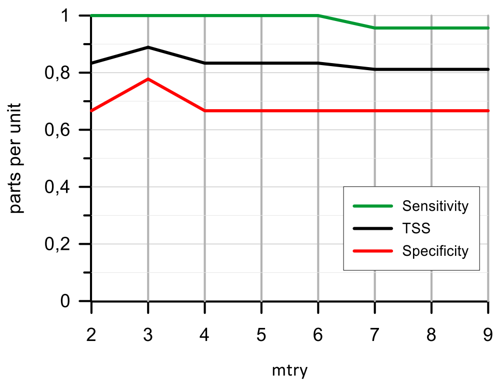


Figure 27: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with unbalanced data.

The best performance of the model occurs when the mtry value is 3, under this premise the final OOB value is 6,25%.

– Selection of the minimum ntree value:

During the mtry selection process, the ntree value was set to 5000; however, this parameter can be reduced as much as the model indicators (OOB error, the specificity, the sensitivity, and TSS) stabilize:

- OOB error:

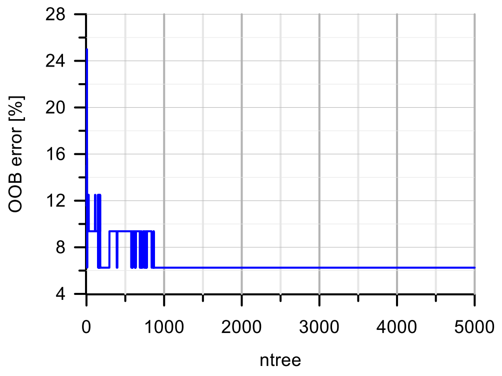


Figure 28: Variation of the OOB error depending on the ntree value during the calibration of the RF model with unbalanced data.

- Specificity, sensitivity and TSS:

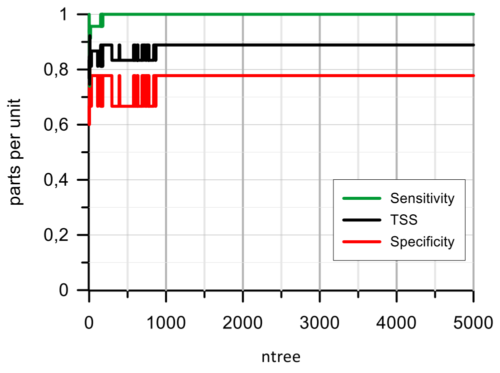


Figure 29: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with unbalanced data.

The model performance stabilizes when the number of trees reaches 867, so the ntree value was set to 1800.

3.1.3.2. Random forest model with oversampled data

– Calibration of the mtry parameter:

- OOB error:

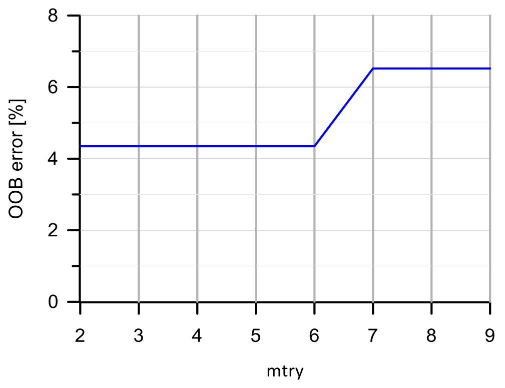


Figure 30: Variation of the OOB error depending on the mtry value during the calibration of the RF model with oversampled data.

- Specificity, sensitivity and TSS:

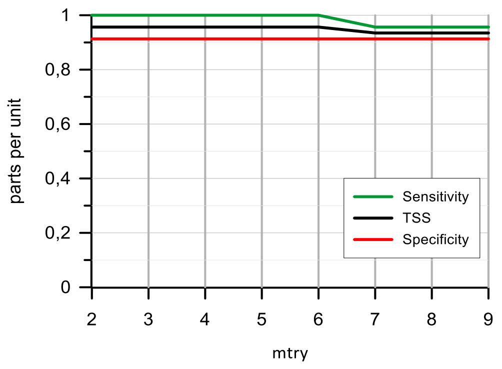


Figure 31: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with oversampled data.

The mtry value was set to 2, as the model performance is optimal when it varies from 2 to 6, and a lower mtry value leads to smaller software requirements. The final OOB error value is 4,35%.

– Selection of the minimum ntree value:

The models' indicators varied depending on the ntree value as it is shown:

- OOB error:

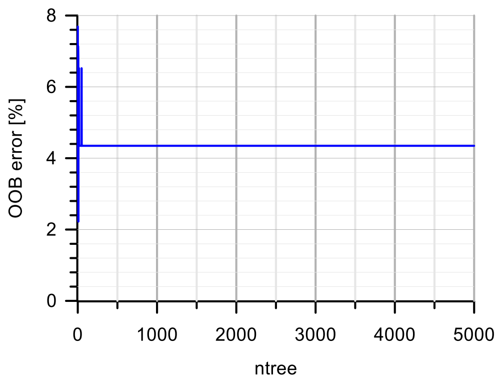


Figure 32: Variation of the OOB error depending on the ntree value during the calibration of the RF model with oversampled data.

- Specificity, sensitivity and TSS:

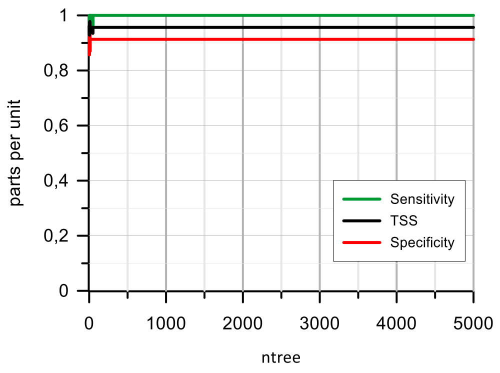


Figure 33: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with oversampled data.

The model performance stabilizes when the number of trees reaches 50, so the ntree value was set to 100.

3.1.3.3. Random forest model with undersampled data

– Calibration of the mtry parameter:

The model indicators varied as follows:

- OOB error:

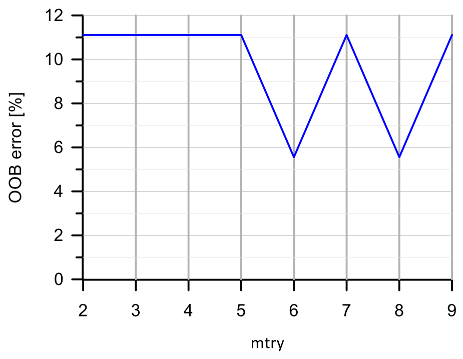


Figure 34: Variation of the OOB error depending on the mtry value during the calibration of the RF model with undersampled data.

- Specificity, sensitivity and TSS:

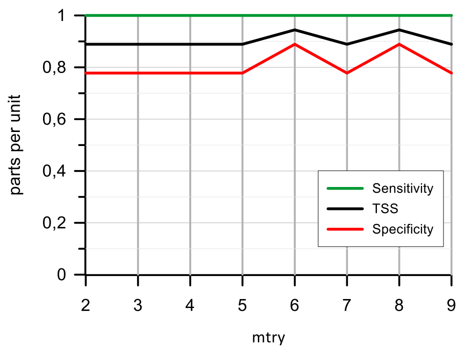


Figure 35: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with undersampled data.

The mtry value was set as 6, as the model performance was optimal for the values 6 and 8.

The OOB error stabilizes at 5,56%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS varied depending on the number of trees as follows:

- OOB error:

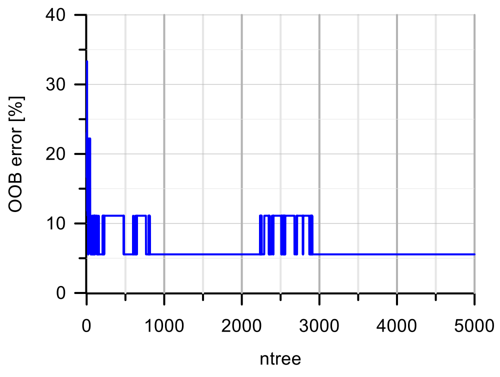


Figure 36: Variation of the OOB error depending on the ntree value during the calibration of the RF model with undersampled data.

- Specificity, sensitivity and TSS:

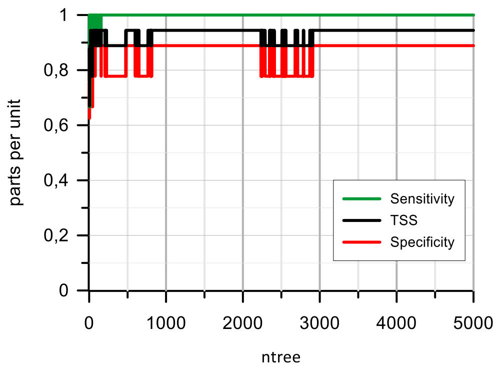


Figure 37: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with undersampled data.

The minimum number of trees for the model to stabilize is 2884, so the ntree value was set to 5900.

3.1.3.4. Conditional inference forest model with unbalanced data

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

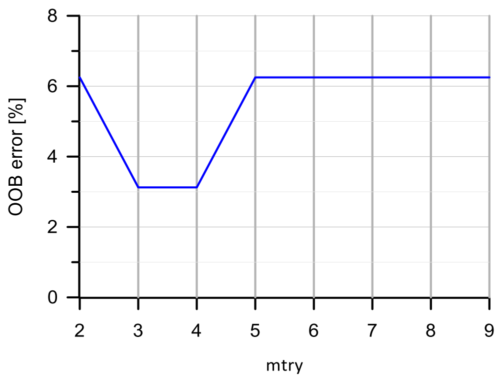


Figure 38: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with unbalanced data.

- Specificity, sensitivity and TSS:

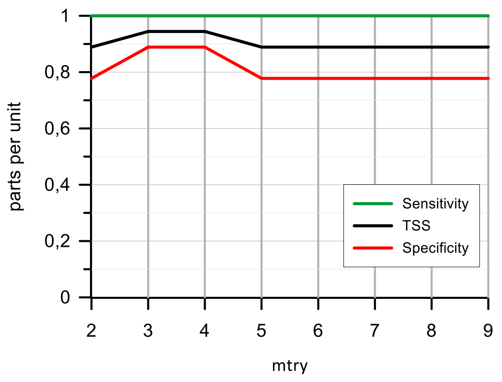


Figure 39: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with unbalanced data.

The mtry value selected was 3, the minimum between both possible options (3 and 4). The final OOB error value is 3,13%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees, as can be seen below:

- OOB error:

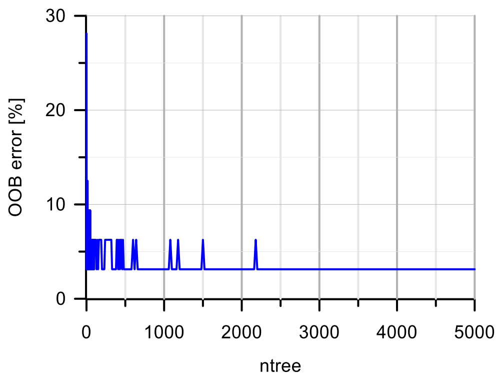


Figure 40: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with unbalanced data.

- Specificity, sensitivity and TSS:

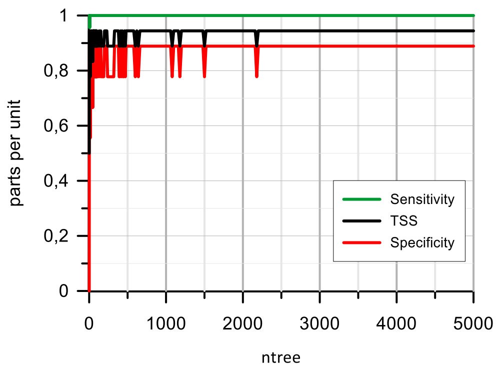


Figure 41: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with unbalanced data.

The minimum ntree value for the model to stabilize is 2180, so this parameter was set to 4400.

3.1.3.5. Conditional inference forest model with oversampled data

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

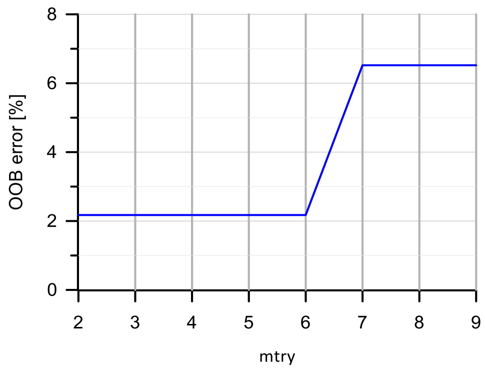


Figure 42: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

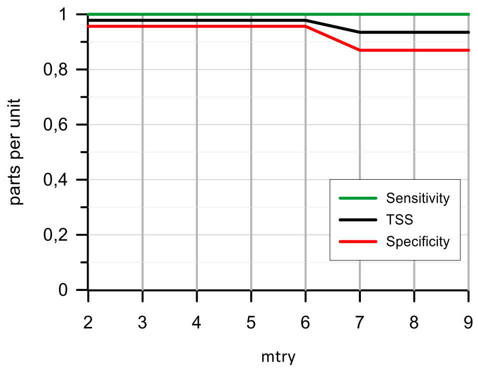


Figure 43: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with oversampled data.

The mtry was set to 2, the minimum between 2 and 6. The final OOB error value is 2,17%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees as can be seen below:

- OOB error:

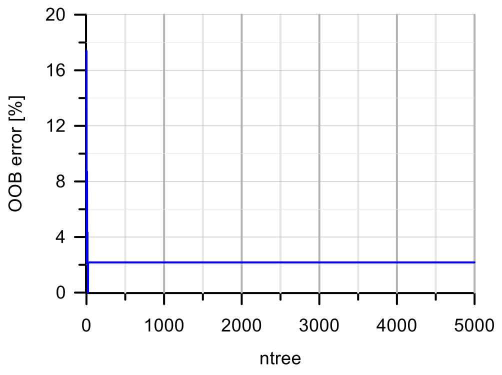


Figure 44: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

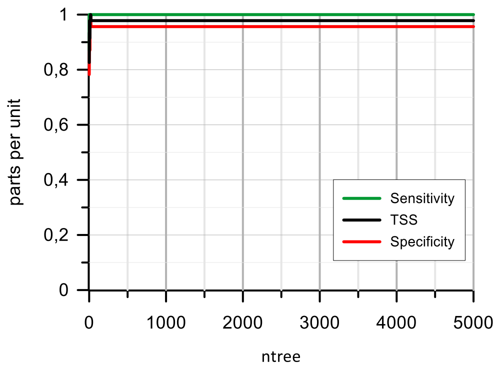


Figure 45: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with oversampled data.

The minimum ntree value for the model to stabilize is 21, so this parameter was set to 100.

3.1.4. Summary and discussion

The following table pools the mtry and ntree that optimize every model of Lethenteron Zanandrea presence, as well as the final OOB error, specificity, sensitivity, and TSS:

	RF models			**CIF models**		
	Unbalanced data	**Oversampled data**	**Undersampled data**	**Unbalanced data**	**Oversampled data**	**Undersampled data**
mtry	3	2	6	3	2	-
ntree	1800	100	5900	4400	100	-
OOB error	6,25	4,35	5,55	3,125	2,17	-
Specificity	1	1	1	1	1	-
Sensitivity	0,78	0,91	0,89	0,89	0,96	-
TSS	0,89	0,96	0,94	0,94	0,98	-

Table 4: Models' indicators after optimization.

Firstly, it would be important to note that every model had a good performance, as every final specificity, sensitivity and TSS is higher than 0,7; 0,7 and 0,85; respectively.

Regarding the prediction capacity of the models, on the one hand, the CIF models had a better performance, reaching lower OOB error values and the higher sensitivities compared to the RF models. On the other hand, the models that used the oversampled data had a better performance than those that used the unbalanced data or the undersampled data.

The model with the worst performance was the RF mode that used unbalanced data, with a 6,25% of error and the most unsatisfying TSS.

3.2. Padogobius bonelli

3.2.1. Input data and variable selection

The Boruta method identified six important variables: MICROLITHAL, PSAMMAL, CV_15, SHALL_MARG, CV60_75 and CV15_30. The most important variable identified was the MICROLITHAL, which refers to the presence of big gravel substrates (2-6 cm), which Padogobius Bonelli uses during its reproductive season to create the nests. The flow velocity was also greatly important, as three different moderate flow velocities frequencies were selected (CV_15, CV60_75 and CV15_30).

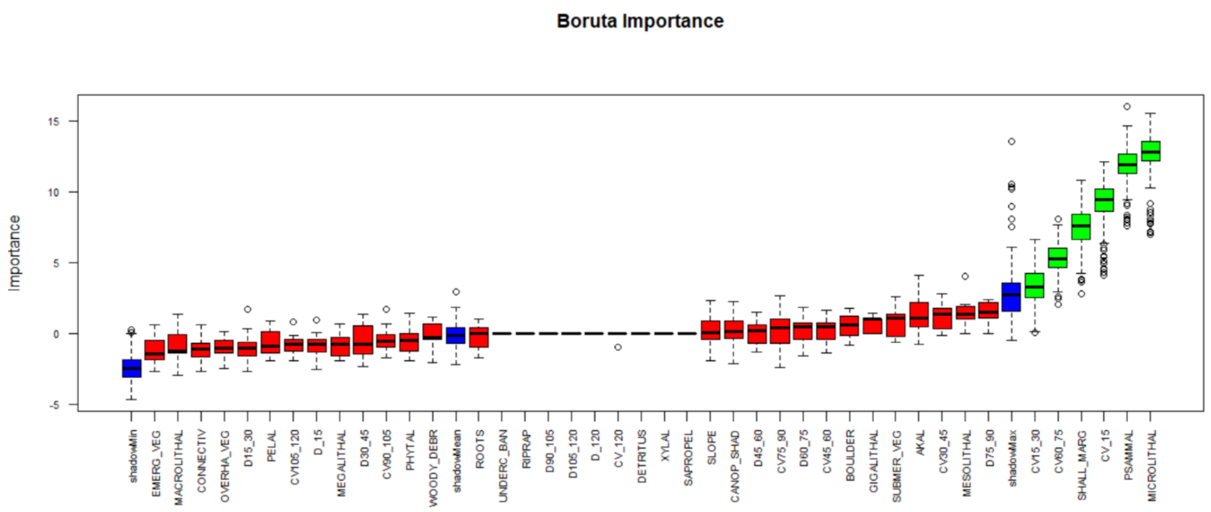


Figure 46: Boruta importance test results.

These variables were used to create the habitat suitability model. In addition, the variable D75_90 was also selected, in order to consider at least one water depth frequency variable in the creation of the models.

3.2.2. Habitat suitability models

As before, in the first instance the parameters were set identically for all the models: the ntree value was set to 5000, and the mtry value was set to 3 ($m = \sqrt{7} = 2{,}65 \approx 3$).

The importance of the variables was evaluated for every model:

- *RF.normal:*

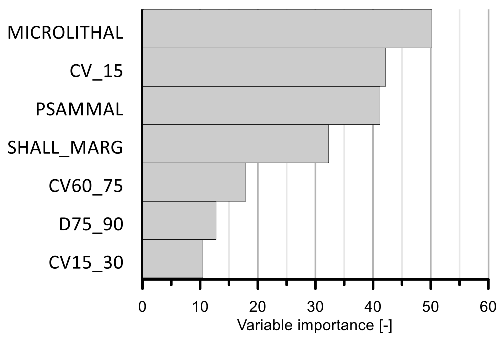


Figure 47: Variable importance measurements in the RF model with unbalanced data.

- *CIF.normal:*

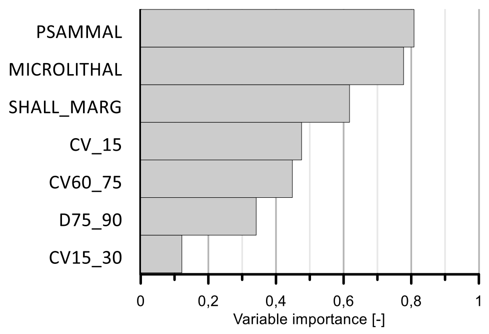


Figure 48: Variable importance measurements in the CIF model with unbalanced data.

- *RF.over:*

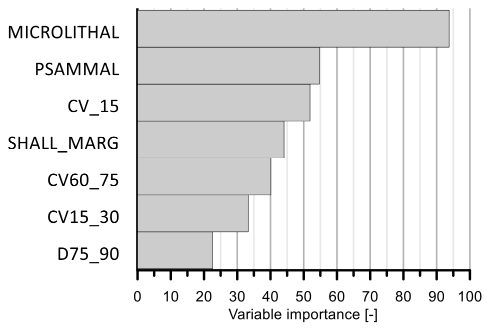


Figure 49: Variable importance measurements in the RF model with oversampled data.

- *CIF.over:*

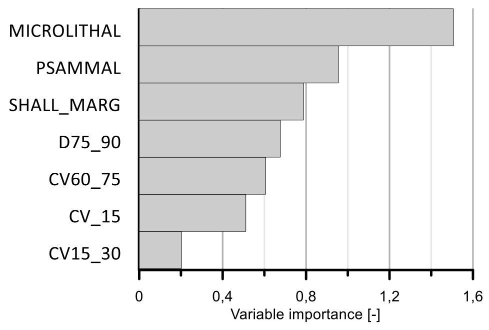


Figure 50: Variable importance measurements in the CIF model with oversampled data.

- *RF.under:*

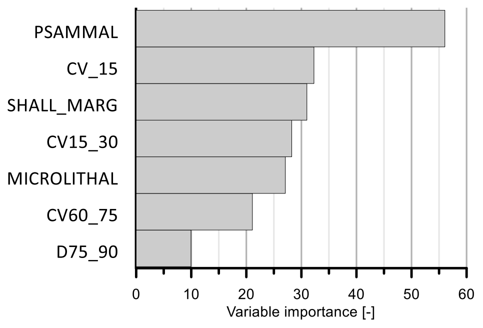


Figure 51: Variable importance measurements in the RF model with undersampled data.

- *CIF.under:*

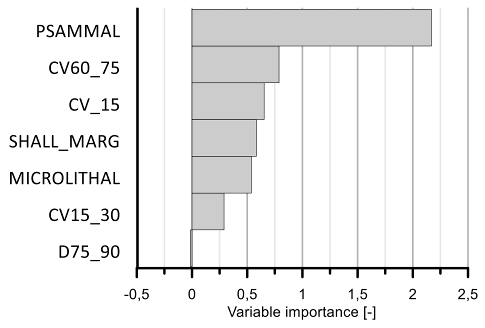


Figure 52: Variable importance measurements in the CIF model with undersampled data.

Even though the importance results were more divergent in this case, almost every model identified MICROLITHAL or PSAMMAL (the most important variables according to the Boruta test) as one of the most important variables.

The OOB error of the six models Is shown in the following table:

	RF models			**CIF models**		
	Unbalanced data	**Oversampled data**	**Undersampled data**	**Unbalanced data**	**Oversampled data**	**Undersampled data**
OOB error (%)	15,09	10	16,67	9,43	4,29	11,11

Table 5: OOB error before optimizing the models.

On the one hand, the best performances were achieved by the CIF models, whose OOB errors were lower than the RF ones.

On the other hand, the models that used the oversampled data also had a better performance than those that used the unbalanced data or the undersampled data, being the undersampling models slightly worse.

3.2.3. Optimization of the models

3.2.3.1. Random forest model with unbalanced data

– Calibration of the mtry parameter:

The OOB error, the specificity, the sensitivity, and the TSS varied depending on the mtry value as it is shown:

- OOB error:

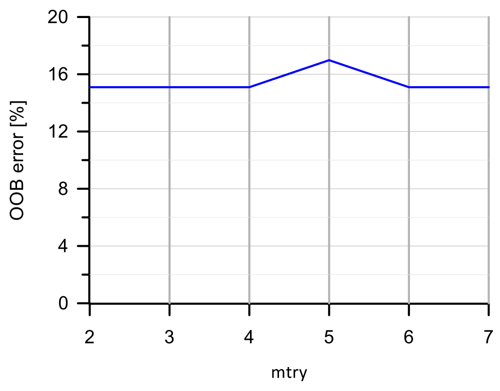


Figure 53: Variation of the OOB error depending on the mtry value during the calibration of the RF model with unbalanced data.

- Specificity, sensitivity and TSS:

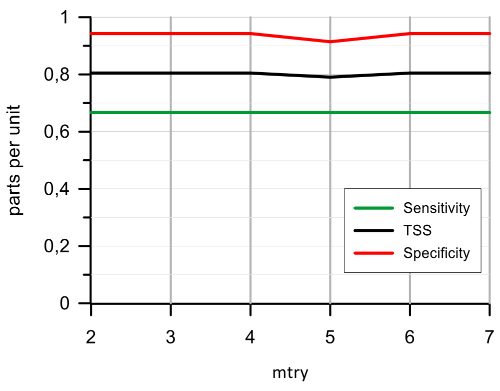


Figure 54: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with unbalanced data.

The best performance of the model occurs when the mtry value varies between 2 and 4, or 6 and 7. So, the mtry parameter was set to 2, the lowest value, which led to a 15,09% of OOB error.

– Selection of the minimum ntree value:

The following figures show how the model indicators (OOB error, the specificity, the sensitivity, and TSS) vary depending on the number of trees of the forest:

- OOB error:

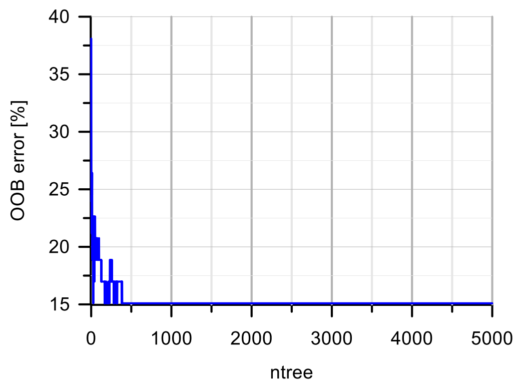


Figure 55: Variation of the OOB error depending on the ntree value during the calibration of the RF model with unbalanced data.

- Specificity, sensitivity and TSS:

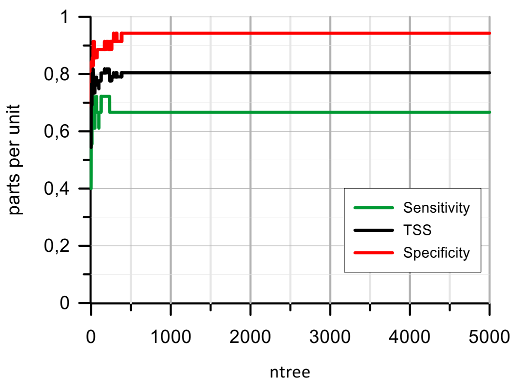


Figure 56: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with unbalanced data.

The model stabilizes when the number of trees is 387, so the ntree value was set to 800.

3.2.3.2. Random forest model with oversampled data

– Calibration of the mtry parameter:

- OOB error:

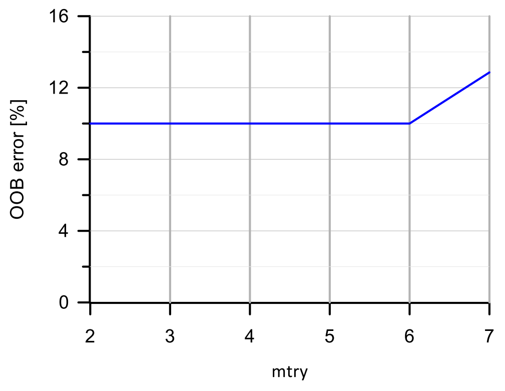


Figure 57: Variation of the OOB error depending on the mtry value during the calibration of the RF model with oversampled data.

- Specificity, sensitivity and TSS:

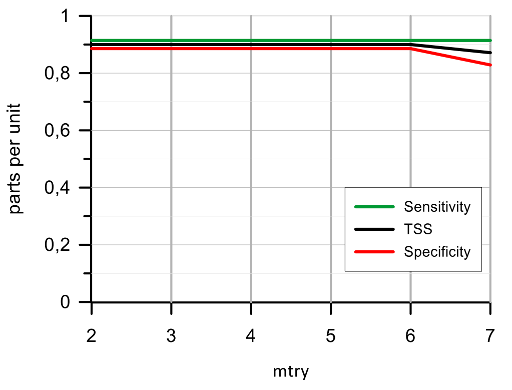


Figure 58: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with oversampled data.

The mtry value was set to 2 and the final OOB error value is 10%.

– Selection of the minimum ntree value:

The models' indicators varied depending on the ntree value as it is shown:

- OOB error:
- Specificity, sensitivity and TSS:

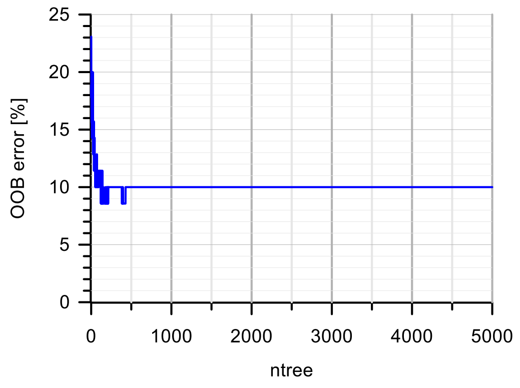


Figure 59: Variation of the OOB error depending on the ntree value during the calibration of the RF model with oversampled data.

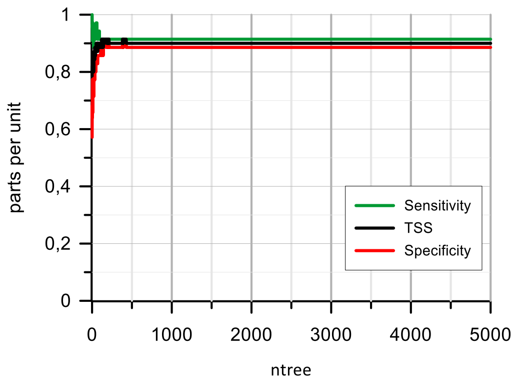


Figure 60: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with oversampled data.

The model performance stabilizes when the number of trees reach 213, so the ntree value was set to 500.

3.2.3.3. Random forest model with undersampled data

– Calibration of the mtry parameter:

The model indicators varied as follows:

- OOB error:
- Specificity, sensitivity and TSS:

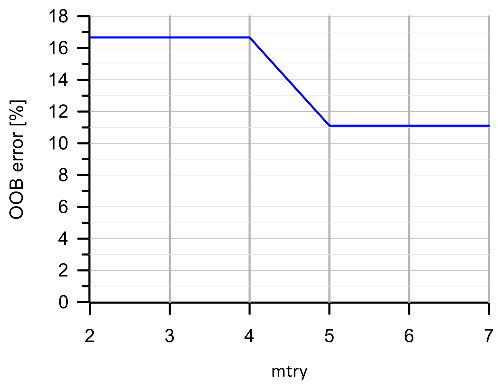


Figure 61: Variation of the OOB error depending on the mtry value during the calibration of the RF model with undersampled data.

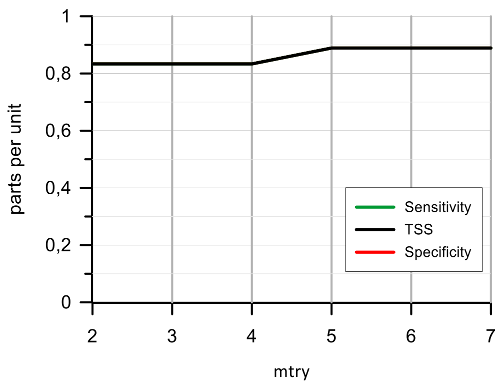


Figure 62: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with undersampled data.

The mtry value was set as 2, as the model performance was optimal between 2 and 4.

The OOB error stabilizes at 11,11%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS varied depending on the number of trees as follows:

- OOB error:

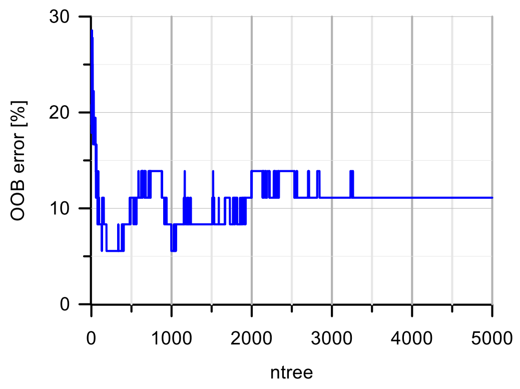


Figure 63: Variation of the OOB error depending on the ntree value during the calibration of the RF model with undersampled data.

- Specificity, sensitivity and TSS:

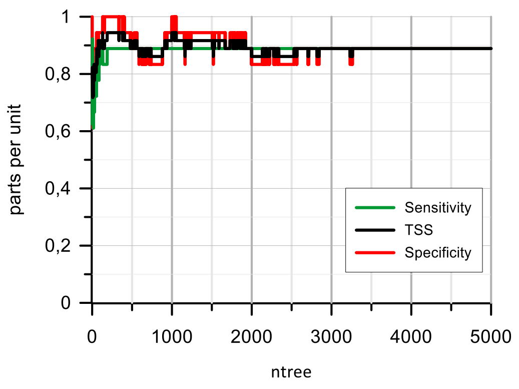


Figure 64: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with undersampled data.

The minimum number of trees for the model to stabilize is 3264, so the ntree value was set to 6600.

3.2.3.4. Conditional inference forest model with unbalanced data

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

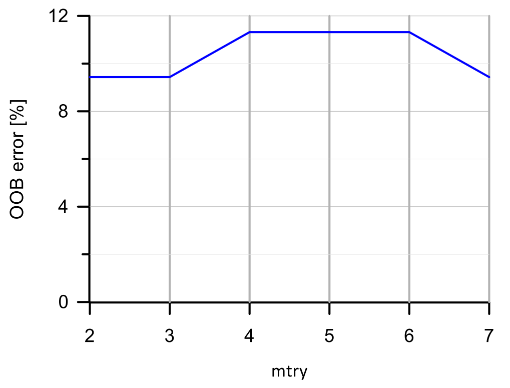


Figure 65: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with unbalanced data.

- Specificity, sensitivity and TSS:

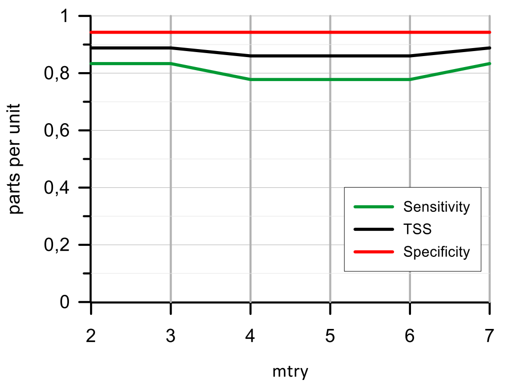


Figure 66: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with unbalanced data.

The mtry value selected was 3 and the final OOB error value is 9,43%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees as can be seen below:

- OOB error:

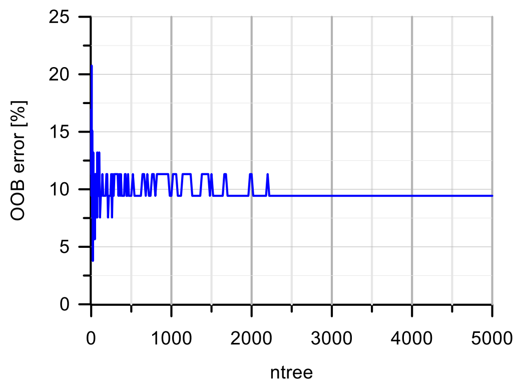


Figure 67: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with unbalanced data.

- Specificity, sensitivity and TSS:

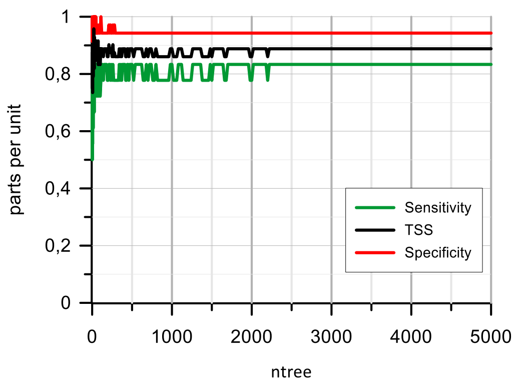


Figure 68: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with unbalanced data.

The minimum ntree value for the model to stabilize is 2200, so this parameter was set to 4400.

3.2.3.5. Conditional inference forest model with oversampled data

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

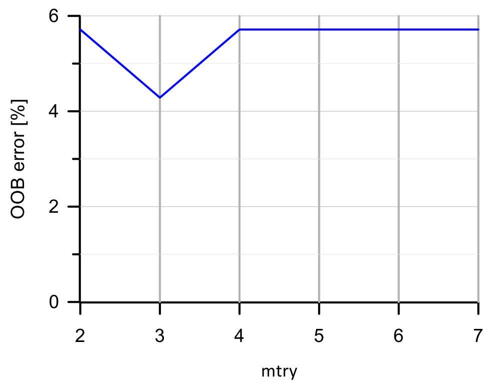


Figure 69: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

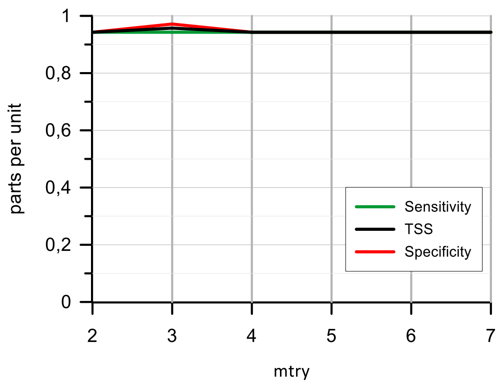


Figure 70: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with oversampled data.

The mtry was set to 3 and the final OOB error is 4,29%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees as can be seen below:

- OOB error:

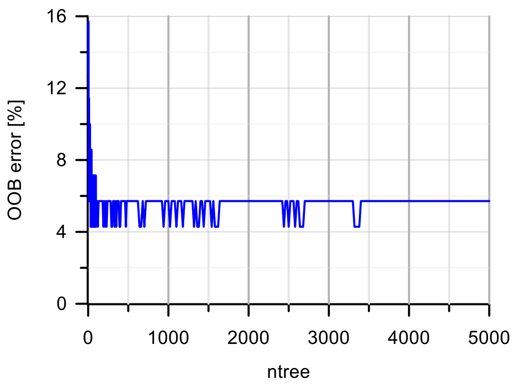


Figure 71: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

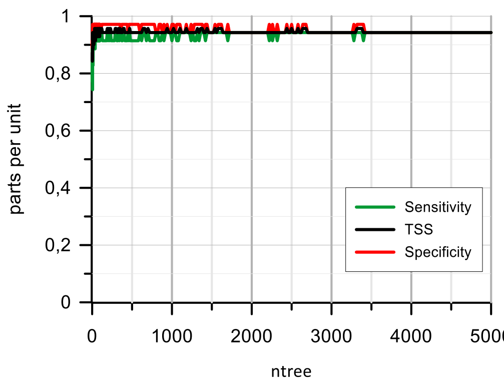


Figure 72: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with oversampled data.

The minimum ntree value for the model to stabilize is 3400, so this parameter was set to 6800.

3.2.3.6. Conditional inference forest model with undersampled data

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

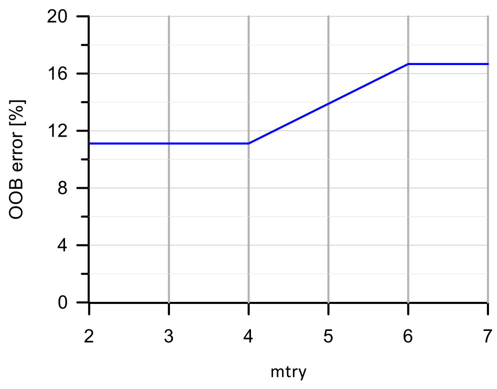


Figure 73: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

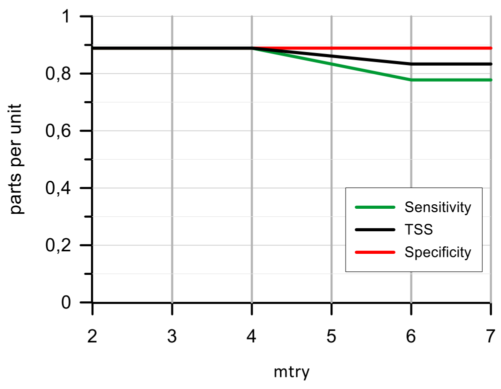


Figure 74: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with oversampled data.

The mtry was set to 2 and the final OOB error value is 11,11%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees as can be seen below:

- OOB error:

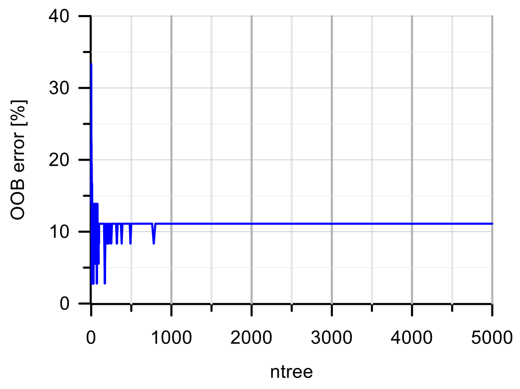


Figure 75: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

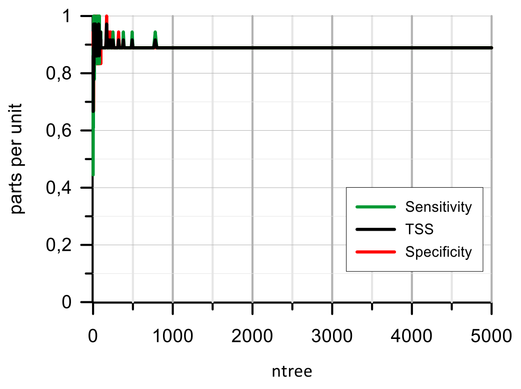


Figure 76: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with oversampled data.

The minimum ntree value for the model to stabilize is 780, so this parameter was set to 1600.

3.2.4. Summary and discussion

The following table pools the mtry and ntree that optimize every model of Padogobius Bonelli presence, as well as the final OOB error, specificity, sensitivity, and TSS:

	RF models			CIF models		
	Unbalanced data	**Oversampled data**	**Undersampled data**	**Unbalanced data**	**Oversampled data**	**Undersampled data**
mtry	2	2	5	2	3	2
ntree	800	900	6600	4400	6800	1600
OOB error	15,09	10	11,11	9,43	4,29	11,11
Specificity	0,94	0,89	0,89	0,94	0,94	0,89
Sensitivity	0,67	0,9	0,89	0,83	0,94	0,89
TSS	0,80	0,9	0,89	0,89	0,94	0,89

Table 6: Models' indicators after optimization.

Firstly, it would be relevant to highlight that the RF model that used the unbalanced data, was the only model that could not overcome the minimums established for the model performance to be considered good: its final sensitivity is not over 70%.

Regarding to the prediction capacity of the models: On the one hand, the CIF models had a better or equal performance, reaching lower OOB error values and higher sensitivities compared to the RF models; the CIF model that used the undersampled data was the only one whose performance was not better but equal to the RF model. On the other hand, the models that used the oversampled data had a better performance than those that used the unbalanced data or the undersampled data.

In the case of the RF models, the one with the worst performance was the one that used unbalanced data, with a 15,09% of error and the most unsatisfying TSS. In the case of the CIF models, the one with the worst performance was the undersampling model.

3.3. Phoxinus lumaireul

3.3.1. Input data and variable selection

In this case only three variables were identified as important by the Boruta test: SLOPE, D30_45, and AKAL. However, in order to consider at least one current velocity variable (CV60_75) and one cover type variable (EMERG_VED), four more variables were selected to create the models: CV60_75, D45_60, D60_75 and EMERG_VEG.

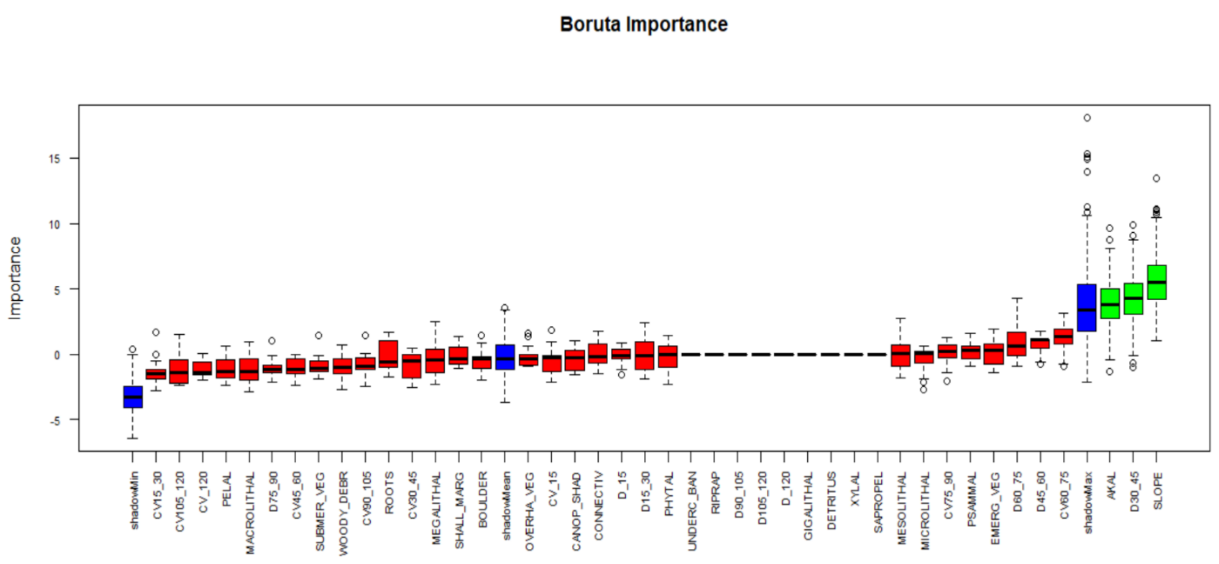


Figure 77: Boruta importance test results.

The most important variable identified was the SLOPE; it would be important to highlight that this variable was identified as the only important one in other tests carried out with different randomizer seeds.

3.3.2. Habitat suitability models

Newly, the models' parameters were set equally: the ntree value was set to 5000, and the mtry value was set to 3 ($m = \sqrt{7} = 2{,}65 \approx 3$).

The importance of the variables was evaluated for every model:

- *RF.normal:*

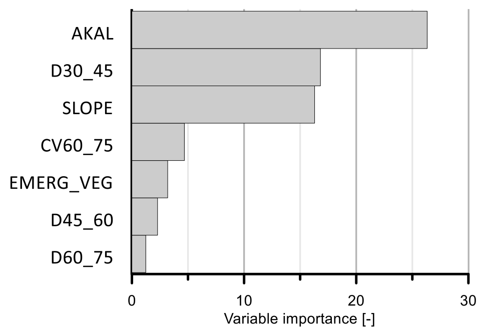


Figure 78: Variable importance measurements in the RF model with unbalanced data.

- *CIF.normal:*

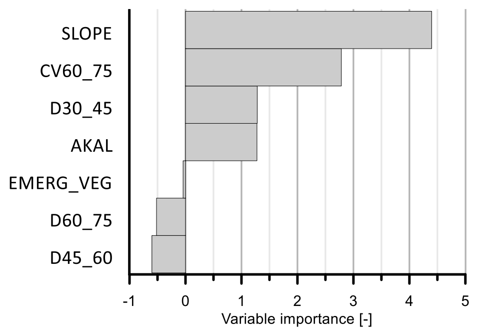


Figure 79: Variable importance measurements in the CIF model with unbalanced data.

- *RF.over:*

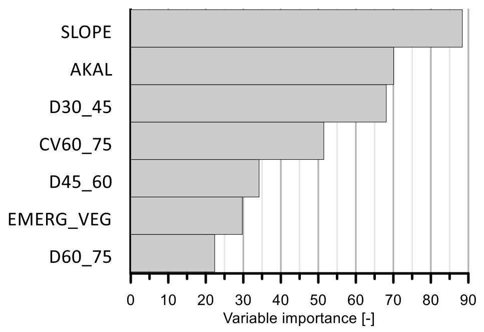


Figure 80: Variable importance measurements in the RF model with oversampled data.

- *CIF.over:*

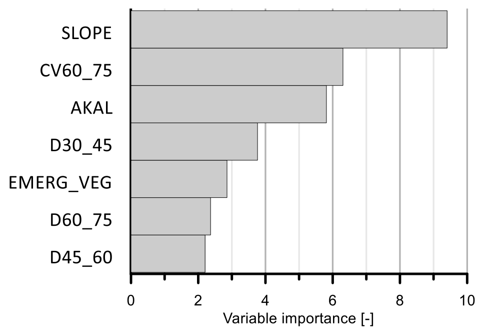


Figure 81: Variable importance measurements in the CIF model with oversampled data.

- *RF.under:*

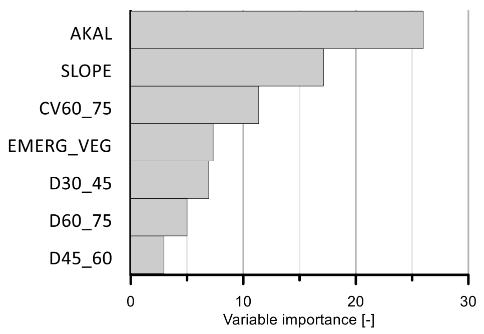


Figure 82: Variable importance measurements in the RF model with undersampled data.

- *CIF.under:*

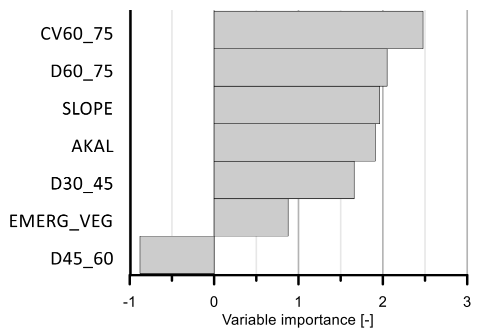


Figure 83: Variable importance measurements in the CIF model with undersampled data.

In all cases the variable SLOPE was identified among the three most important ones, always with a high relative importance.

Then, the models' OOB error was calculated:

	RF models			CIF models		
	Unbalanced data	**Oversampled data**	**Undersampled data**	**Unbalanced data**	**Oversampled data**	**Undersampled data**
OOB error (%)	32,08	14,47	33,33	18,87	7,89	20

Table 7: OOB error before optimizing the models.

These results followed the tendency set by the previous fish species' HSMs. On the one hand, the CIF models' performances were better than the RF's ones. On the other hand, the models with oversampled data were the best ones, with an OOB error lower than the other models.

3.3.3. Optimization of the models

3.3.3.1. Random forest model with unbalanced data

– Calibration of the mtry parameter:

The OOB error, the specificity, the sensitivity, and the TSS varied depending on the mtry value as it is shown:

- OOB error:

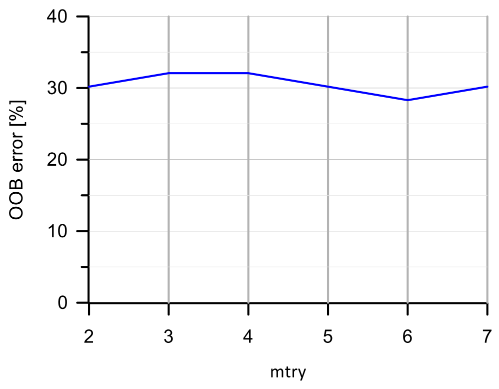


Figure 84: Variation of the OOB error depending on the mtry value during the calibration of the RF model with unbalanced data.

- Specificity, sensitivity and TSS:

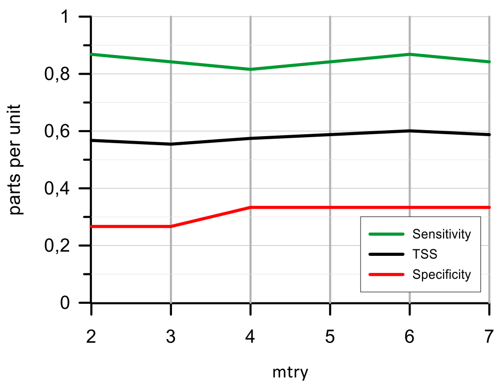


Figure 85: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with unbalanced data.

The mtry was set to 6, when the best model performance was achieved. In this case, the OOB error is 28,3%.

– Selection of the minimum ntree value:

The following figures show how the model indicators (OOB error, the specificity, the sensitivity, and TSS) vary depending on the number of trees of the forest:

- OOB error:

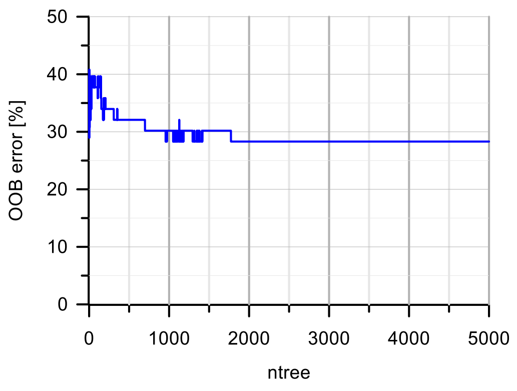


Figure 86: Variation of the OOB error depending on the ntree value during the calibration of the RF model with unbalanced data.

- Specificity, sensitivity and TSS:

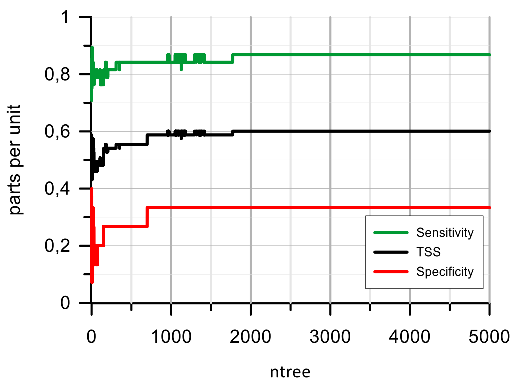


Figure 87: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with unbalanced data.

The model stabilizes when the number of trees is 1774, so the ntree value was set to 3600.

3.3.3.2. Random forest model with oversampled data

– Calibration of the mtry parameter:

- OOB error:

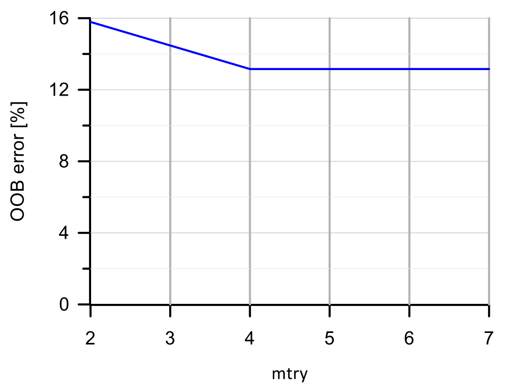


Figure 88: Variation of the OOB error depending on the mtry value during the calibration of the RF model with oversampled data.

- Specificity, sensitivity and TSS:

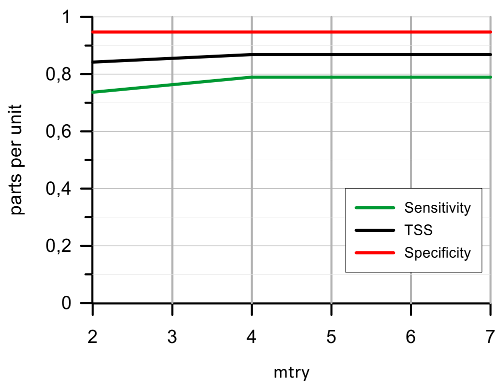


Figure 89: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with oversampled data.

The mtry value was set to 4 and the final OOB error value is 13,16%.

– Selection of the minimum ntree value:

The models' indicators varied depending on the ntree value as it is shown:

- OOB error:
- Specificity, sensitivity and TSS:

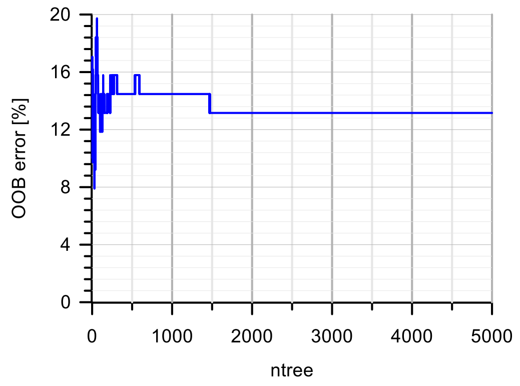


Figure 90: Variation of the OOB error depending on the ntree value during the calibration of the RF model with oversampled data.

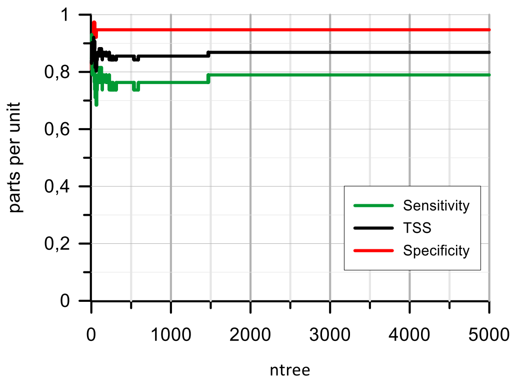


Figure 91: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with oversampled data.

The model performance stabilizes when the number of trees reach 1474, so the ntree value was set to 3000.

3.3.3.3. Random forest model with undersampled data

– Calibration of the mtry parameter:

The model indicators varied as follows:

- OOB error:
- Specificity, sensitivity and TSS:

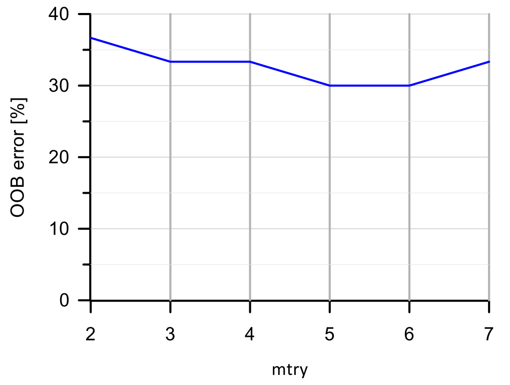


Figure 92: Variation of the OOB error depending on the mtry value during the calibration of the RF model with undersampled data.

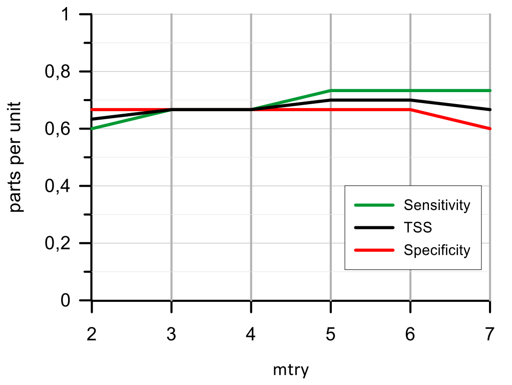


Figure 93: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the RF model with undersampled data.

The mtry value was set to 5, as the model performance was optimal between 5 and 6. The OOB error stabilizes at 30%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS varied depending on the number of trees as it follows:

- OOB error:

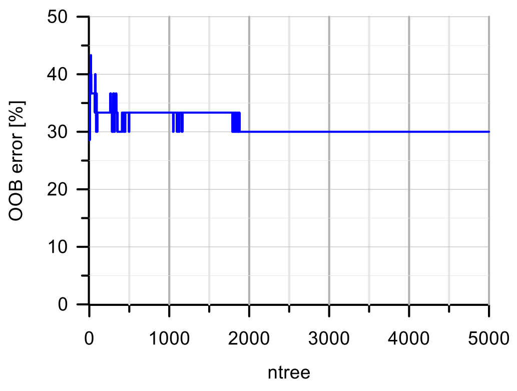


Figure 94: Variation of the OOB error depending on the ntree value during the calibration of the RF model with undersampled data.

- Specificity, sensitivity and TSS:

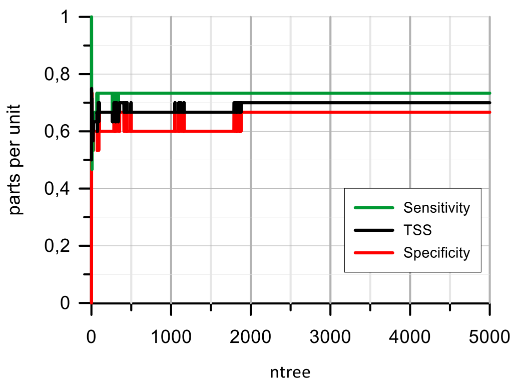


Figure 95: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the RF model with undersampled data.

The minimum number of trees for the model to stabilize is 1889, so the ntree value was set to 3800.

3.3.3.4. Conditional inference forest model with unbalanced data

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

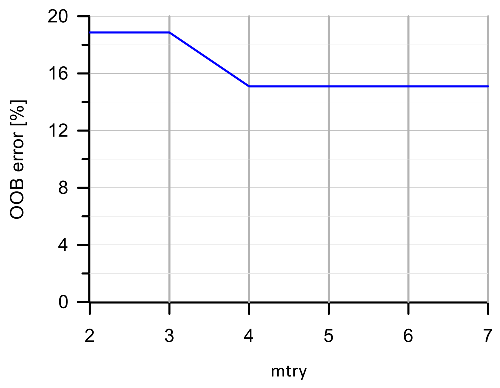


Figure 96: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with unbalanced data.

- Specificity, sensitivity and TSS:

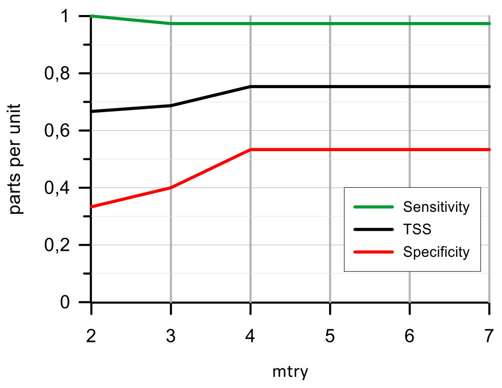


Figure 97: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with unbalanced data.

The mtry value selected was 4 and the final OOB error value is 15,09%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees as it can be seen below:

- OOB error:

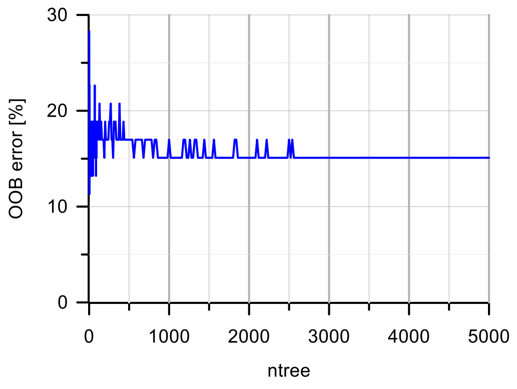


Figure 98: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with unbalanced data.

- Specificity, sensitivity and TSS:

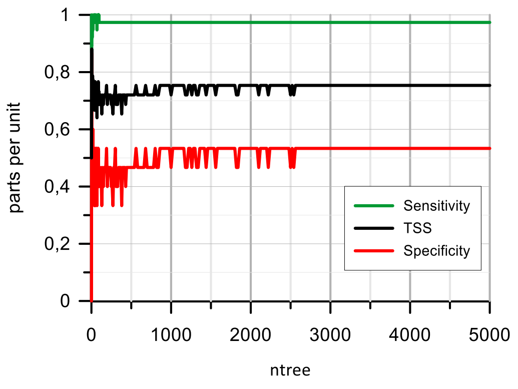


Figure 99: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with unbalanced data.

The minimum ntree value for the model to stabilize is 2560, so this parameter was set to 5200.

3.3.3.5. <u>Conditional inference forest model with oversampled data</u>

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

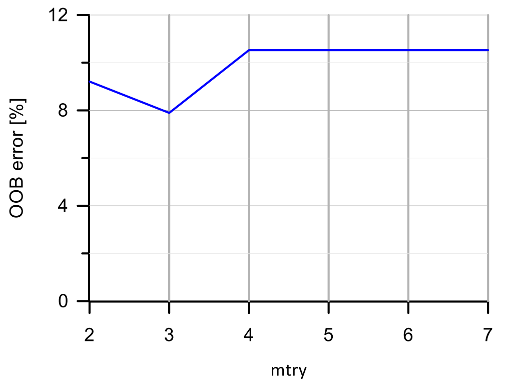


Figure 100: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

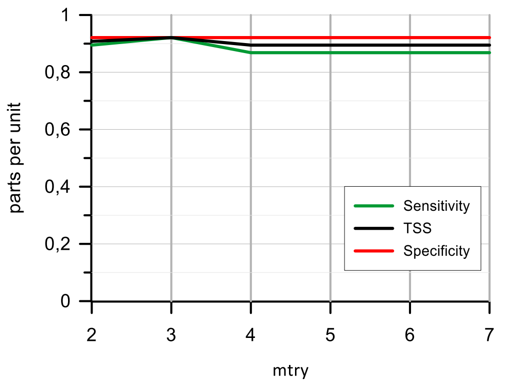


Figure 101: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with oversampled data.

The mtry was set to 3 and the final OOB error is 7,89%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees as it can be seen below:

- OOB error:

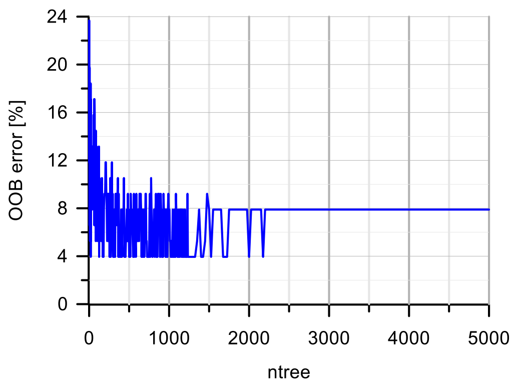


Figure 102: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

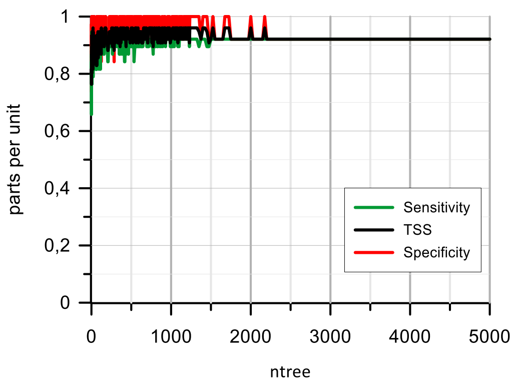


Figure 103: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with oversampled data.

The minimum ntree value for the model to stabilize is 2200, so this parameter was set to 4400.

3.3.3.6. Conditional inference forest model with undersampled data

– Calibration of the mtry parameter:

The model indicators change with the mtry value as it is shown:

- OOB error:

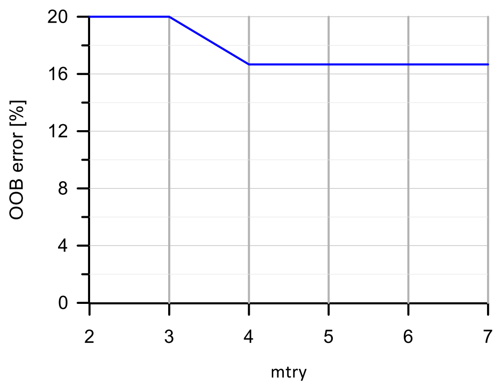


Figure 104: Variation of the OOB error depending on the mtry value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

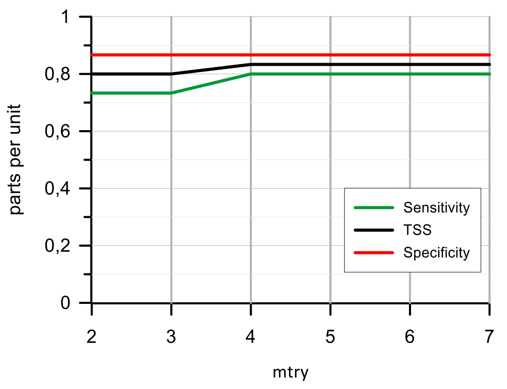


Figure 105: Variation of the specificity, sensitivity, and TSS depending on the mtry value during the calibration of the CIF model with oversampled data.

The mtry was set to 4 and the final OOB error value is 16,67%.

– Selection of the minimum ntree value:

The OOB error, the specificity, the sensitivity, and the TSS changed depending on the number of trees as it can be seen below:

- OOB error:

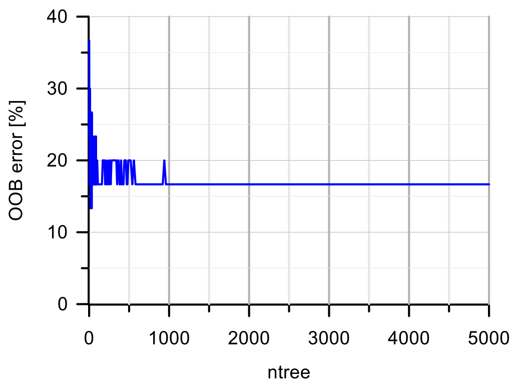


Figure 106: Variation of the OOB error depending on the ntree value during the calibration of the CIF model with oversampled data.

- Specificity, sensitivity and TSS:

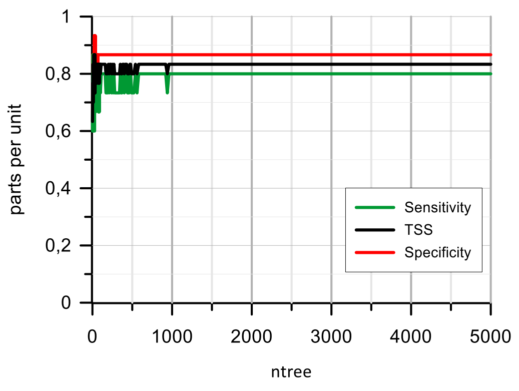


Figure 107: Variation of the specificity, sensitivity, and TSS depending on the ntree value during the calibration of the CIF model with oversampled data.

The minimum ntree value for the model to stabilize is 960, so this parameter was set to 2000.

3.3.4. Summary and discussion

The following table pools the mtry and ntree that optimize every model of Padogobius Bonelli presence, as well as the final OOB error, specificity, sensitivity, and TSS:

	RF models			**CIF models**		
	Unbalanced data	**Oversampled data**	**Undersampled data**	**Unbalanced data**	**Oversampled data**	**Undersampled data**
mtry	6	4	5	4	3	4
ntree	3600	3000	3800	5200	4400	2000
OOB error	28,3	13,16	30	15,09	7,89	16,67
Specificity	0,33	0,95	0,67	0,53	0,92	0,87
Sensitivity	0,87	0,79	0,73	0,97	0,92	0,8
TSS	0,6	0,87	0,7	0,75	0,92	0,83

Table 8: Models' indicators after optimization.

In this case three models could not overcome the minimum performance requirements established: the RF models that used the unbalanced data and the undersampled data, and the CIF model that used the unbalanced data.

Regarding the comparison between the CIF and the RF models, the CIF models had a better performance than the RF models again, with lower OOB error and higher TSS values.

Moreover, the models that used the oversampled data performed better than the others.

4. CONCLUSIONS AND DISCUSSION

Firstly, it would be essential to highlight a fact that could be inherent to the size of the input datasets: the input dataset used to develop the Lethenteron Zanandreai's models was 40% smaller than the ones used for the other species; it consisted of 32 mesohabitats. This fact led to a highly reduced undersampled dataset, composed of 18 mesohabitats. So, the lack of data could explain why the CIF model with undersampled data could not be developed for this fish species.

Secondly, and to compare the models, the following table summarizes the performance of each model developed:

		RF models			**CIF models**		
		Unbalanced data	**Oversampled data**	**Undersampled data**	**Unbalanced data**	**Oversampled data**	**Undersampled data**
OOB error [%]	**Lethenteron zanandreai**	6,25	4,35	5,55	3,125	2,17	-
	Padogobius bonelli	15,09	10	11,11	9,43	4,29	11,11
	Phoxinus lumaireul	28,3	13,16	30	15,09	7,89	16,67
Specificity [parts per unit]	**Lethenteron zanandreai**	1	1	1	1	1	-
	Padogobius bonelli	0,94	0,89	0,89	0,94	0,94	0,89
	Phoxinus lumaireul	0,33	0,95	0,67	0,53	0,92	0,87
Sensitivity [parts per unit]	**Lethenteron zanandreai**	0,78	0,91	0,89	0,89	0,96	-
	Padogobius bonelli	0,67	0,9	0,89	0,83	0,94	0,89
	Phoxinus lumaireul	0,87	0,79	0,73	0,97	0,92	0,8
TSS [parts per unit]	**Lethenteron zanandreai**	0,89	0,96	0,94	0,94	0,98	-
	Padogobius bonelli	0,80	0,9	0,89	0,89	0,94	0,89
	Phoxinus lumaireul	0,6	0,87	0,7	0,75	0,92	0,83

Table 9: Models' indicators summary.

– Effects of balancing the dataset:

On the one hand, thanks to balancing the input datasets with the *upSample* and the *downSample* functions, the number of false negatives and false positives predictions of the fish presence (These measures are related to the specificity and the sensitivity respectively) is reduced, leading to highest TSS values:

TSS [parts per unit]

	Lethenteron zanandreai		Padogobius bonelli		Phoxinus lumaireul	
	RF	CIF	RF	CIF	RF	CIF
Unbalanced data	0,89	0,94	0,8	0,89	0,6	0,75
Oversampled data	0,96	0,98	0,9	0,94	0,87	0,92
Undersampled data	0,94	-	0,89	0,89	0,7	0,83

Table 10: TSS comparison.

On the other hand, the models that used the oversampled data had the lowest OOB errors in all cases. This fact could be attributed to the fact that the datasets used to create these models were not only balanced, but also enlarged, leading to better performances:

OOB error [%]

	Lethenteron zanandreai		Padogobius bonelli		Phoxinus lumaireul	
	RF	CIF	RF	CIF	RF	CIF
Unbalanced data	6,25	3,125	15,09	9,43	28,3	15,09
Oversampled data	4,35	2,17	10	4,29	13,16	7,89

Table 11: OOB error comparison between the models that used the unbalanced and the oversampled data.

However, in the case of the models that used the undersampled data, the OOB error tendency is not clear: the error is reduced in some cases and increased in others. It would be necessary to compare more cases with bigger datasets in order to study the opportunities that the models balanced with the *downSample* function can provide.

OOB error [%]

	Lethenteron zanandreai		Padogobius bonelli		Phoxinus lumaireul	
	RF	CIF	RF	CIF	RF	CIF
Unbalanced data	6,25	3,125	15,09	9,43	28,3	15,09
Undersampled data	5,55	-	11,11	11,11	30	16,67

Table 12: OOB error comparison between the models that used the unbalanced and the undersampled data.

– Effects of using a two-step splitting criterion when creating the models:

The models that used the CIF approach had a better prediction capacity in all cases, leading to the lowest OOB errors and the highest TSS values:

OOB [%]

	Lethenteron zanandreai			Padogobius bonelli			Phoxinus lumaireul		
	Un-balanced data	Over-sampled data	Under-sampled data	Un-balanced data	Over-sampled data	Under-sampled data	Un-balanced data	Over-sampled data	Under-sampled data
RF	6,25	4,35	5,55	15,09	10	11,11	28,3	13,16	30
CIF	3,125	2,17	-	9,43	4,29	11,11	15,09	7,89	16,67

Table 13: OOB error comparison between RF and CIF.

TSS [parts per unit]

	Lethenteron zanandreai			Padogobius bonelli			Phoxinus lumaireul		
	Un-balanced data	Over-sampled data	Under-sampled data	Un-balanced data	Over-sampled data	Under-sampled data	Un-balanced data	Over-sampled data	Under-sampled data
RF	0,89	0,96	0,94	0,80	0,9	0,89	0,6	0,87	0,7
CIF	0,94	0,98	-	0,89	0,94	0,89	0,75	0,92	0,83

Table 14: TSS comparison between RF and CIF.

As has been already explained in the material and methods chapter, the CIF approach proposes a two-step splitting criterion when creating the decision trees, in order to reduce the variable selection bias:

The first step consists of selecting the best split variable by means of a linear rank association test, this step aims to reduce the RF bias when determining the importance of the variables. Some articles show that this bias is particularly high in those cases that mix categorical and continuous variables [53]; this is the case of this project, so the performance improvement could be due to this fact. However, as the input variables were narrowed down by using the Boruta test before applying the methods, the differences between the RF and the CIF performances should not be so pronounced. Because of these reasons, the CIF's enhancement could also be attributed to the second step of the splitting criteria, which consists of comparing all possible partitions of the split variable.

– Combined effects of balancing the dataset with the *upSample* function and considering the CIF approach:

The CIF models that used the oversampled data were the ones with the best performances, they had the lowest OOB errors and the highest TSS values. The project results show evidence to claim that these models were the bests thanks to the use of high enough balanced datasets and thanks to compare all possible partitions of the split variable when building the decision trees that compose the prediction models.

The benefits of these models could help to detailly predict how the habitat descriptors affect the mesohabitat and the fish species for the EU countries to develop RBMPs and progress toward achieving the WFD aims.

5. BIBLIOGRAPHY

[1] European Union. Directive (EU) 2000/60/EC of the European Parliament and of the Council, of October 23, 2000, establishing a framework for Community action in the field of water policy. Official Journal of the European Union L 327, December 22, 2000, article 25.

[2] European Union. Directive (EU) 2000/60/EC of the European Parliament and of the Council, of October 23, 2000, establishing a framework for Community action in the field of water policy. Official Journal of the European Union L 327, December 22, 2000, article 1.

[3] European Union. Directive (EU) 2000/60/EC of the European Parliament and of the Council, of October 23, 2000, establishing a framework for Community action in the field of water policy. Official Journal of the European Union L 327, December 22, 2000, article 4.

[4] European Union. Directive (EU) 2000/60/EC of the European Parliament and of the Council, of October 23, 2000, establishing a framework for Community action in the field of water policy. Official Journal of the European Union L 327, December 22, 2000, article 2.

[5] European Union. Directive (EU) 2000/60/EC of the European Parliament and of the Council, of October 23, 2000, establishing a framework for Community action in the field of water policy. Official Journal of the European Union L 327, December 22, 2000, Annex V.

[6] European Union. Technical Report-2015-086 of the European Commission, of 2015. Guidance Document No. 31 of the Water Framework Directive: Ecological flows in the implementation of the Water Framework Directive. Page 1.

[7] European Union. Technical Report-2015-086 of the European Commission, of 2015. Guidance Document No. 31 of the Water Framework Directive: Ecological flows in the implementation of the Water Framework Directive. Page 2.

[8] European Union. Directive (EU) 2000/60/EC of the European Parliament and of the Council, of October 23, 2000, establishing a framework for Community action in the field of water policy. Official Journal of the European Union L 327, December 22, 2000, Annex II and Annex III.

[9] European Union. Directive (EU) 2000/60/EC of the European Parliament and of the Council, of October 23, 2000, establishing a framework for Community action in the field of water policy. Official Journal of the European Union L 327, December 22, 2000, article 4 and 13.

[10] Vezza, P., C. Paranimf, G. De Gandia, V. España, F. Martinez-capel, R. Muñoz-mas and J.D. Alcaraz-hernandez. Habitat suitability modeling with random forest as a tool for fish conservation in Mediterranean rivers. Polito [Internet]. Available in: http://hdl.handle.net/11583/2518927.

[11]Mouton AM, Schneider M, Depestele J, Goethals PLM, De Pauw N. Fish habitat modelling as a tool for river management. Ecological Engineering, Vol. 29, No. 3, (2007), pp 305-315

[12]A Guisan A, Thuiller W. Predicting species distribution: offering more than simple habitat models. Ecology Letters, Vol. 8, No. 9, (2005), pp 993-1009.

[13]J.L. Kemp, D.M. Harper, and G.A. Crosa. Use of ‘functional habitats’ to link ecology with morphology and hydrology in river rehabilitation. Aquatic conservation: Marine and Freshwater Ecosystem (1999) [Internet]. Available in: https://doi.org/10.1002/(SICI)1099-0755(199901/02)9:1<159::AID-AQC319>3.0.CO.

[14]Wikipedia contributors. Alps [Internet]. Wikipedia, The Free Encyclopedia. 2022. Available in: https://en.wikipedia.org/w/index.php?title=Alps&oldid=1091323570

[15]espaces-transfrontaliers.org: Territory factsheets [Internet]. Espaces-transfrontaliers.org. [viewed in 13th june 2022]. Available in: http://www.espaces-transfrontaliers.org/en/resources/territories/territory-factsheets/territories/territory/show/euroregion-alpes-mediterranee/

[16]Wikipedia contributors. Piedmont [Internet]. Wikipedia, The Free Encyclopedia. 2022. Available in: https://en.wikipedia.org/w/index.php?title=Piedmont&oldid=1085701635

[17]Wikipedia contributors. Monte Viso [Internet]. Wikipedia, The Free Encyclopedia. 2021. Available in: https://en.wikipedia.org/w/index.php?title=Monte_Viso&oldid=1061073078

[18]Vezza P, Parasiewicz P, Rosso M, Comoglio C. Defining minimum environmental flows at regional scale: Application of mesoscale habitat models and catchments classification: Regional-scale e-flows assessment. River Res Appl [Internet]. 2012;28(6):717–30. Available in: http://dx.doi.org/10.1002/rra.1571

[19]Wikipedia contributors. Emilia-Romagna [Internet]. Wikipedia, The Free Encyclopedia. Available in: https://it.wikipedia.org/w/index.php?title=Emilia-Romagna&oldid=127614294

[20]Clima Emilia-Romaña: Climograma, Temperatura y Tabla climática para Emilia-Romaña - Climate-Data.org [Internet]. Climate-data.org. [viewed on 13th June 2022]. Available in: https://es.climate-data.org/europe/italia/emilia-romana-394/

[21]Dreamstime.com. [viewed on 13th June 2022]. Available in: https://thumbs.dreamstime.com/b/satellite-view-piedmont-region-italy-satellite-view-piedmont-region-italy-d-render-physical-map-piemonte-plains-172813483.jpg

[22]Dreamstime.com. [viewed on 13th June 2022]. Available in: https://thumbs.dreamstime.com/b/satellite-view-emilia-romagna-region-satellite-view-emilia-romagna-region-italy-d-render-physical-map-emilia-172813582.jpg

[23]Solà C, Ordeix M, Pou-Rovira Q, Sellarès N, Queralt A, Bardina M, et al. Longitudinal connectivity in hydromorphological quality assessments of rivers. The ICF index: A river connectivity index and its application to Catalan rivers. Limnetica [Internet]. 2011;30(2):273–92. Available in: http://dx.doi.org/10.23818/limn.30.21

[24]Vezza P, Parasiewicz P, Calles O, Spairani M, Comoglio C. Modelling habitat requirements of bullhead (Cottus gobio) in Alpine streams. Aquat Sci [Internet]. 2014;76(1):1–15. Available in: http://dx.doi.org/10.1007/s00027-013-0306-7

[25]La Lampreda Padana, Lethenteron zanandreai pg. 2 [Internet]. Ittiofauna.org. [viewed on 13th June 2022]. Available in: http://www.ittiofauna.org/webmuseum/agnati/petromyzontiformes/petromyzontidae/lethenteron/lethenteron_zanandreai/lethenteron_zanandreai.htm

[26]La lampreda padana, Lethenteron zanandreai (Vladykov, 1955) [Internet]. Ittiofauna.org. [viewed in 13th june 2022]. Available in: http://www.ittiofauna.org/webmuseum/agnati/petromyzontiformes/petromyzontidae/lethenteron/lethenteron_zanandreai/index.htm

[27]Porcellotti S. Ghiozzo padano, Padogobius bonelli (Bonaparte, 1846) [Internet]. Ittiofauna.org. [viewed in 13th june 2022]. Available in: http://www.ittiofauna.org/webmuseum/pesciossei/perciformes/gobidae/padogobius/padogobius_bonelli/padogobius_bonelli.html

[28]Wikipedia contributors. Padogobius bonelli [Internet]. Wikipedia, The Free Encyclopedia. Disponible en: https://es.wikipedia.org/w/index.php?title=Padogobius_bonelli&oldid=123473447

[29]Sanguinerola, Phoxinus lumaireul (Schinz, 1840) [Internet]. Ittiofauna.org. [viewed on 13th June 2022]. Available in: http://www.ittiofauna.org/webmuseum/pesciossei/cypriniformes/cyprinidae/phoxinus/phoxinus_lumaireul/p_lumaireul.htm

[30]Sanguinerola, Phoxinus lumaireul (Schinz, 1840) [Internet]. Ittiofauna.org. [viewed on 13th June 2022]. Available in: http://www.ittiofauna.org/webmuseum/pesciossei/cypriniformes/cyprinidae/phoxinus/phoxinus_lumaireul/index.htm

[31]Breiman L, Friedman J, Stone CJ, Olshen RA. Classification and Regression Trees. Filadelfia, PA, Estados Unidos de América: Chapman & Hall/CRC; 1984.

[32]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 10.

[33]Forrest P. An introduction to statistical learning. North Charleston, SC, Estados Unidos de América: Createspace Independent Publishing Platform; 2017. p. 187.

[34]Kuhn M, Johnson K. Applied predictive modeling. Nueva York, NY, Estados Unidos de América: Springer; 2019. p. 213.

[35]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 43.

[36]Dwyer K, Holte R. Decision tree instability and active learning. En: Machine Learning: ECML 2007. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007. p. 128

[37]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 37-39.

[38]Breiman, L. Random Forests. Machine Learning 45; 2001. p. 5-32.

[39]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 34.

[40]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 39-40.

[41]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 40.

[42]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 43-44.

[43]Genuer R, Poggi J-M. Random forests with R. 2020a ed. Nueva York, NY, Estados Unidos de América: Springer; 2020. p 44-46.

[44]What is the meaning of the Gini Index? [Internet]. Data Science. 2021. Available in: https://datascience.stackexchange.com/questions/102393/what-is-the-meaning-of-the-gini-index

[45]Breiman L, Friedman J, Stone CJ, Olshen RA. Classification and Regression Trees. Filadelfia, PA, Estados Unidos de América: Chapman & Hall/CRC; 1984.

[46]Probst P, Wright MN, Boulesteix A-L. Hyperparameters and tuning strategies for random forest. Wiley Interdiscip Rev Data Min Knowl Discov [Internet]. 2019;9(3):e1301. p.4. Available in: http://dx.doi.org/10.1002/widm.1301

[47]Hothorn T, Everitt BS. A handbook of statistical analyses using R. Filadelfia, PA, Estados Unidos de América: Chapman & Hall/CRC; 2006.

[48]Liu Y, Zhou S, Wei H, An S. A comparative study of forest methods for time-to-event data: variable selection and predictive performance. BMC Med Res Methodol [Internet]. 2021;21(1):193. Available in: http://dx.doi.org/10.1186/s12874-021-01386-8

[49]Liaw A, Maintainer A. Package “randomForest” [Internet]. 2022. Available in: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[50]Package “partykit” [Internet]. R-project.org. 2022. Available in: https://cran.r-project.org/web/packages/partykit/partykit.pdf

[51]Rudnicki WR, Miron M, Kursa B. Package “Boruta” [Internet]. R-project.org. 2020. Available in: https://cran.r-project.org/web/packages/Boruta/Boruta.pdf

[52]Mazzanti S. Boruta explained exactly how you wished someone explained to you [Internet]. Towards Data Science. 2020. Available in: https://towardsdatascience.com/boruta-explained-the-way-i-wish-someone-explained-it-to-me-4489d70e154a

[53]Mazzanti Strobl C, Boulesteix A-L, Zeileis A, Hothorn T. Bias in random forest variable importance measures: illustrations, sources and a solution. BMC Bioinformatics [Internet]. 2007;8(1):25. Available in: http://dx.doi.org/10.1186/1471-2105-8-25

6. ANNEXES

6.1. Input and output data

The excels documents containing the input and output data can be downloaded from the following links:

- Lethenteron zanandreai:
https://docs.google.com/spreadsheets/d/17CW_bY0sW-7ZmPc7lzWVnRpUk3n2DMba/edit?usp=sharing&ouid=100749357155596833726&rtpof=true&sd=true

- Padogobius Bonelli:
https://docs.google.com/spreadsheets/d/1XFBrSdYEHDRoRfNm7Ez_Q_0OK6tyzAkl/edit?usp=sharing&ouid=100749357155596833726&rtpof=true&sd=true

- Phoxinus lumaireul:
https://docs.google.com/spreadsheets/d/1AFYY3qJGSUO3Vp6O7il0pv8C_WpYKSf6/edit?usp=sharing&ouid=100749357155596833726&rtpof=true&sd=true

6.2. The code

```
####################################################
##INITIAL SETTINGS##

par(mar=c(1,1,1,1))
ntree_initial = 5000

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

##PACKAGES##
library(languageR)
library(party)
library(AUCRF)
library(vcd)
library(e1071)
library(caret)
library(vegan)
library(tree)
library(cluster)
library(randomForest)
library(ggplot2)
library(lubridate)
library(dplyr)
library(openxlsx)
library(rio)
library(polycor)
library(Boruta)
library(mlbench)
library(ipred)
library(VSURF)
library(partykit)
library(stablelearner)


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

## READ TXT AND SET THE DIRECTORY##
setwd("C:\\Users\\chris\\Desktop\\The Climate Change\\TFM\\Data 1")
T.meso<-read.table("Species_name_Base_RF_model.txt", sep="\t", header=T)


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

## SET THE DATA ##
T.meso_abu<-T.meso[(T.meso[,75]>0),]
dim(T.meso)
names(T.meso)
class(T.meso)
str(T.meso)

PRES_SPEC_Juv <- T.meso[,75]
fish.pres <- T.meso[,75]
dim(PRES_SPEC_Juv)
names(PRES_SPEC_Juv)

iniz.data<-cbind(T.meso[,13:24],T.meso[,43:72], PRES_SPEC_Juv)
dim(iniz.data)
names(iniz.data)
summary(iniz.data)

before.boruta.data<-cbind(T.meso[,13:24],T.meso[,43:72])
```

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

## BORUTA TEST: VARIABLE SELECTION ##
set.seed(8159780)
boruta <- Boruta(before.boruta.data,as.factor(PRES_SPEC_Juv),doTrace =
2,maxRuns = 500)
windows()
par(xpd=T, mar=par()$mar+c(3,0,4,0))
plot(boruta, las=2, cex.axis=0.7,xlab="",main="Boruta Importance")

# Loop to select the important variables from the Boruta Test
i=1
j=0
y=0
z=as.numeric(dim(before.boruta.data)[2])

while(i<=z)
{
  if(boruta$finalDecision[i]=="Confirmed" )
  {
    j=j+1

        if(j==1)
        {
          x=i;
        }

        if(j==2)
        {
          after.boruta.data <- cbind(before.boruta.data[x],
before.boruta.data[i])
          y=1
        }

        if(j>2)
        {
          after.boruta.data <-
cbind(after.boruta.data,before.boruta.data[i])
        }
  }

  i=i+1
}
if(y==0)
{
  after.boruta.data <- cbind(before.boruta.data[x])
}


dim(after.boruta.data)
names(after.boruta.data)
summary(after.boruta.data)
num.var=length(names(after.boruta.data))

# Number of important variables
num.variable.imp=j

####################################################
```
<

```
## DEFINE THE OVERSAMPLING AND THE UNDERSAMPLING DATA ##

# Data oversampling
set.seed(8159780)
data_oversample<-cbind(after.boruta.data,fish.pres)
data_oversample$fish.pres<-as.factor(data_oversample$fish.pres)
oversample_model<-upSample(data_oversample[,-(ncol(after.boruta.data)
+1)],data_oversample$fish.pres,yname = "fish.pres")
summary(oversample_model$fish.pres)

# Data undersampling
set.seed(8159780)
data_undersample<-cbind(after.boruta.data,fish.pres)
data_undersample$fish.pres<-as.factor(data_undersample$fish.pres)
undersample_model<-downSample(data_undersample[,-(ncol(after.boruta.data)
+1)],data_undersample$fish.pres,yname = "fish.pres")
summary(undersample_model$fish.pres)


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

## DEFINE HSMs (RF AND CIF) ##

# Set the mtry parameter
m<-round(sqrt(ncol(after.boruta.data))) #problema de clasificacion
if (m<2)
  m<-2
m

## MODEL: RF UNBALANCED ##
set.seed(8159780)
RF.normal <- randomForest(after.boruta.data, as.factor(fish.pres),
data=after.boruta.data, nodesize = 1, importance=T, ntree=ntree_initial,
mtry=m, proximity=TRUE,outscale=TRUE)

## MODEL: RF OVERSAMPLING ##
set.seed(8159780)
RF.over <- randomForest(oversample_model[1:num.var],
as.factor(oversample_model$fish.pres), data=oversample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree_initial, mtry=m,
proximity=TRUE,outscale=TRUE)

## MODEL: RF UNDERSAMPLING ##
set.seed(8159780)
RF.under <- randomForest(undersample_model[1:num.var],
as.factor(undersample_model$fish.pres), data=undersample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree_initial, mtry=m,
proximity=TRUE,outscale=TRUE)

## MODEL: CIF UNBALANCED ##
set.seed(8159780)
CIF.normal <- cforest(factor(fish.pres) ~., data=after.boruta.data,
               control = ctree_control(teststat = "max", testtype =
"Teststatistic",
                                       mincriterion = 0, saveinfo = TRUE),
               ytrafo = NULL, scores = NULL, ntree = ntree_initial,
               perturb = list(replace = TRUE), #fraction = 1#
               applyfun = NULL,
               cores = NULL, mtry = m, trace = TRUE)
```

```
# stable tree
set.seed(8159780)
CIF.normal.st <- stablelearner::as.stabletree(CIF.normal)

## MODEL: CIF OVERSAMPLING ##
set.seed(8159780)
CIF.over <- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model[1:num.var],
                 control = ctree_control(teststat = "max", testtype =
"Teststatistic",
                                         mincriterion = 0, saveinfo =
TRUE),
                 ytrafo = NULL, scores = NULL, ntree = ntree_initial,
                 perturb = list(replace = TRUE), #fraction = 1#
                 applyfun = NULL,
                 cores = NULL, mtry = m, trace = TRUE)
# stable tree
set.seed(8159780)
CIF.over.st <- stablelearner::as.stabletree(CIF.over)

## MODEL: CIF UNDERSAMPLING ##
set.seed(8159780)
CIF.under <- cforest(factor(undersample_model$fish.pres) ~.,
data=undersample_model[1:num.var],
                    control = ctree_control(teststat = "max", testtype =
"Teststatistic",
                                            mincriterion = 0, saveinfo =
TRUE),
                    ytrafo = NULL, scores = NULL, ntree = ntree_initial,
                    perturb = list(replace = TRUE), #fraction = 1#
                    applyfun = NULL,
                    cores = NULL, mtry = m, trace = TRUE)
# stable tree
set.seed(8159780)
CIF.under.st <- stablelearner::as.stabletree(CIF.under)


#####################################################
## VARIABLE IMPORTANCE TEST ##

## RF UNBALANCED ##
windows()
bar.RF.normal<-
cbind(names(after.boruta.data),as.numeric(importance(RF.normal,type=1)))
bar.RF.normal<-bar.RF.normal[order(as.numeric(bar.RF.normal[,
2])),decreasing=T]
par(xpd=T, mar=par()$mar+c(1,0,4,0))
barplot(sort(importance(RF.normal,type=1)), names.arg=bar.RF.normal[,
1],cex.axis=0.7,las = 2, cex.lab=0.8,main = "Variable importance: RF
unbalanzed", horiz=TRUE ,xlab="",cex.names=0.5, xlim=c(-10, 100))
abline(v=abs(min(importance(RF.normal,type=1))), col='red', lty='longdash',
lwd=2)
par(mar=c(5, 4, 4, 2) + 0.1)


## RF OVERSAMPLING ##
windows()
bar.RF.over<-
cbind(names(oversample_model[1:num.var]),as.numeric(importance(RF.over,type=1)))
bar.RF.over<-bar.RF.over[order(as.numeric(bar.RF.over[,2])),decreasing=T]
```

```
par(xpd=T, mar=par()$mar+c(1,0,4,0))
barplot(sort(importance(RF.over,type=1)), names.arg=bar.RF.over[,
1],cex.axis=0.7,las = 2, cex.lab=0.8,main = "Variable importance: RF
oversampling", horiz=TRUE ,xlab="",cex.names=0.5, xlim=c(-10, 100))
abline(v=abs(min(importance(RF.over,type=1))), col='red', lty='longdash',
lwd=2)
par(mar=c(5, 4, 4, 2) + 0.1)


## RF UNDERSAMPLING ##
windows()
bar.RF.under<-
cbind(names(undersample_model[1:num.var]),as.numeric(importance(RF.under,type=1)))
bar.RF.under<-bar.RF.under[order(as.numeric(bar.RF.under[,
2])),decreasing=T]
par(xpd=T, mar=par()$mar+c(1,0,4,0))
barplot(sort(importance(RF.under,type=1)), names.arg=bar.RF.under[,
1],cex.axis=0.7,las = 2, cex.lab=0.8,main = "Variable importance: RF
undersampling", horiz=TRUE ,xlab="",cex.names=0.5, xlim=c(-10, 100))
abline(v=abs(min(importance(RF.under,type=1))), col='red', lty='longdash',
lwd=2)
par(mar=c(5, 4, 4, 2) + 0.1)


## CIF UNBALANCED ##
windows()
varimp.CIF.normal<-partykit::varimp(CIF.normal)
bar.CIF.normal<-
cbind(names(sort(varimp.CIF.normal)),as.numeric(sort(varimp.CIF.normal)))
par(xpd=T, mar=par()$mar+c(1,0,4,0))
barframe.CIF.normal <- data.frame(bar.CIF.normal)
barplot(as.numeric(barframe.CIF.normal[,2]),
names.arg=barframe.CIF.normal[,1],main = "Variable importance: CIF
unbalanzed", horiz=T, las = 2, cex.axis=0.7, cex.lab=0.8, xlim=c(-10, 100),
cex.names=0.5)
abline(v=abs(min(varimp.CIF.normal)), col='red', lty='longdash', lwd=2)
par(mar=c(5, 4, 4, 2) + 0.1)


## CIF OVERSAMPLING ##
windows()
varimp.CIF.over<-partykit::varimp(CIF.over)
bar.CIF.over<-
cbind(names(sort(varimp.CIF.over)),as.numeric(sort(varimp.CIF.over)))
par(xpd=T, mar=par()$mar+c(1,0,4,0))
barframe.CIF.over <- data.frame(bar.CIF.over)
barplot(as.numeric(barframe.CIF.over[,2]), names.arg=barframe.CIF.over[,
1],main = "Variable importance: CIF oversampling", horiz=T, las = 2,
cex.axis=0.7, cex.lab=0.8, xlim=c(-10, 100), cex.names=0.5)
abline(v=abs(min(varimp.CIF.over)), col='red', lty='longdash', lwd=2)
par(mar=c(5, 4, 4, 2) + 0.1)


## CIF UNDERSAMPLING ##
windows()
varimp.CIF.under<-partykit::varimp(CIF.under)
bar.CIF.under<-
cbind(names(sort(varimp.CIF.under)),as.numeric(sort(varimp.CIF.under)))
par(xpd=T, mar=par()$mar+c(1,0,4,0))
barframe.CIF.under <- data.frame(bar.CIF.under)
```

```
barplot(as.numeric(barframe.CIF.under[,2]), names.arg=barframe.CIF.under[,
1],main = "Variable importance: CIF undersampling", horiz=T, las = 2,
cex.axis=0.7, cex.lab=0.8, xlim=c(-10, 100), cex.names=0.5)
abline(v=abs(min(varimp.CIF.under)), col='red', lty='longdash', lwd=2)
par(mar=c(5, 4, 4, 2) + 0.1)


####################################################
## PREDICTIONS ##

## RF AND CIF UNBALANCED##
Predition.normal<- data.frame(
  "Data unbalanzed"=fish.pres,
  "RF unbalanzed"=as.numeric(predict(RF.normal)) - 1,
  "CIF unbalanzed" = as.numeric(predict(CIF.normal)) - 1
  )
Predition.normal

## RF AND CIF OVERSAMPLED##
Predition.over<- data.frame(
  "Data oversampling"=oversample_model$fish.pres,
  "RF oversampling"=as.numeric(predict(RF.over)) - 1,
  "CIF oversampling" = as.numeric(predict(CIF.over)) - 1
)
Predition.over

## RF AND CIF UNDERSAMPLED##
Predition.under<- data.frame(
  "Data undersampling"=undersample_model$fish.pres,
  "RF undersampling"=as.numeric(predict(RF.under)) - 1,
  "CIF undersampling" = as.numeric(predict(CIF.under)) - 1
)
Predition.under


####################################################
## OOB ERROR ##

## RF UNBALANCED ##
RF.normal.oob = 100 * sum(abs(abs(as.numeric(predict(RF.normal)) - 1) -
abs(iniz.data$PRES_SPEC_Juv)))/length(iniz.data$PRES_SPEC_Juv)

## RF OVERSAMPLING ##
RF.over.oob = 100 * sum(abs(abs(as.numeric(predict(RF.over)) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)

## RF UNDERSAMPLING ##
RF.under.oob = 100 * sum(abs(abs(as.numeric(predict(RF.under)) - 1) -
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)

## CIF UNBALANCED ##
CIF.normal.oob = 100 * sum(abs(abs(as.numeric(predict(CIF.normal)) - 1) -
abs(iniz.data$PRES_SPEC_Juv)))/length(iniz.data$PRES_SPEC_Juv)

## CIF OVERSAMPLING ##
CIF.over.oob = 100 * sum(abs(abs(as.numeric(predict(CIF.over)) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)
```

```
## CIF UNDERSAMPLING ##
CIF.under.oob = 100 * sum(abs(abs(as.numeric(predict(CIF.under)) - 1) -
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)


OOBerror=cbind(RF.normal.oob, RF.over.oob, RF.under.oob, CIF.normal.oob,
CIF.over.oob, CIF.under.oob)
windows()
barplot(

        OOBerror,
        main = "OOB ERROR",
        xlab = "Model",
        ylab = "OOB error (%)",
        border = "black",
        names.arg = c("RF", "RF(over)", "RF(under)", "CIF", "CIF(over)",
"CIF(under)"),
        col = "darkblue",
        ylim = c(0, 50),
        cex.names=0.8,
        )

grid(nx = NA, ny = NULL, lwd = 1, lty = 1, col = "gray")
barplot(

        OOBerror,
        main = "OOB ERROR",
        xlab = "Model",
        ylab = "OOB error (%)",
        border = "black",
        names.arg = c("RF", "RF(over)", "RF(under)", "CIF", "CIF(over)",
"CIF(under)"),
        col = "darkblue",
        ylim = c(0, 50),
        cex.names=0.8,

        add = TRUE)


####################################################
## CONFUSION MATRIX ##

## RF UNBALANCED ##
pred.RF.normal<-predict(RF.normal)
truth.normal<-as.factor(fish.pres)
confusionMatrix(pred.RF.normal, truth.normal, positive='1')
windows()
plot(pred.RF.normal, truth.normal, main = "Confusion matrix: RF
unbalanzed")

## RF OVERSAMPLING ##
pred.RF.over<-predict(RF.over)
truth.over<-as.factor(oversample_model$fish.pres)
confusionMatrix(pred.RF.over, truth.over, positive='1')
windows()
plot(pred.RF.over,truth.over, main = "Confusion matrix: RF oversampling")

## RF UNDERSAMPLING ##
```

```
pred.RF.under<-predict(RF.under)
truth.under<-as.factor(undersample_model$fish.pres)
confusionMatrix(pred.RF.under, truth.under, positive='1')
windows()
plot(pred.RF.under,truth.under, main = "Confusion matrix: RF
undersampling")

## CIF UNBALANCED ##
pred.CIF.normal<-predict(CIF.normal)
confusionMatrix(pred.CIF.normal, truth.normal, positive='1')
windows()
plot(pred.CIF.normal,truth.normal,main = "Confusion matrix: CIF
unbalanzed")

## CIF OVERSAMPLING ##
pred.CIF.over<-predict(CIF.over)
confusionMatrix(pred.CIF.over, truth.over, positive='1')
windows()
plot(pred.CIF.over,truth.over, main = "Confusion matrix: CIF oversampling")

## CIF UNDERSAMPLING ##
pred.CIF.under<-predict(CIF.under)
confusionMatrix(pred.CIF.underr, truth.under, positive='1')
windows()
plot(pred.CIF.under,truth.under, main = "Confusion matrix: CIF
undersampling")


###################################################
## OTHER GRAPHS NOT INCLUDED IN THE PROJECT - CIF MODELS##

#Stable tree CIF unbalanCed
set.seed(8159780)
summary(CIF.normal, original = FALSE)
windows()
image(CIF.normal.st, main = "Variable selections: CIF unbalanzed")
windows()
barplot(CIF.normal.st, cex.names = 0.6, main = "Variable selection
frequencies: CIF unbalanzed")

#Stable tree CIF oversampling
set.seed(8159780)
summary(CIF.over, original = FALSE)
windows()
image(CIF.over.st, main = "Variable selections: CIF oversampling")
windows()
barplot(CIF.over.st, cex.names = 0.6, main = "Variable selection
frequencies: CIF oversampling")

#Stable tree CIF undersampling
set.seed(8159780)
summary(CIF.under, original = FALSE)
windows()
image(CIF.under.st, main = "Variable selections: CIF undersampling")
windows()
barplot(CIF.under.st, cex.names = 0.6, main = "Variable selection
frequencies: CIF undersampling")


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

```
## OPTIMIZATION ##

####################################################
##RF UNBALANCED##

#evaluate mtry
i=1
j=0
mtry.loopmtry=ncol(after.boruta.data)
#
set.seed(8159780)
RF.normal.loopmtry <- randomForest(after.boruta.data, as.factor(fish.pres),
data=after.boruta.data, nodesize = 1, importance=T, ntree=ntree_initial,
mtry=2, proximity=TRUE,outscale=TRUE)

pred.RF.normal.loopmtry<-predict(RF.normal.loopmtry)

OOB.RF.normal.loopmtry <-  100*RF.normal.loopmtry$err.rate[ntree_initial]
Accuracy.RF.normal.loopmtry <-confusionMatrix(pred.RF.normal.loopmtry,
truth.normal, positive='1')$overall[1]
Sensitivity.RF.normal.loopmtry <- confusionMatrix(pred.RF.normal.loopmtry,
truth.normal, positive='1')$byClass[1]
Specificity.RF.normal.loopmtry <- confusionMatrix(pred.RF.normal.loopmtry,
truth.normal, positive='1')$byClass[2]
num.matrix.RF.normal.loopmtry <- 2
#

while ((2+i)<=mtry.loopmtry) {
  set.seed(8159780)
  RF.normal.loopmtry <- randomForest(after.boruta.data,
as.factor(fish.pres), data=after.boruta.data, nodesize = 1, importance=T,
ntree=ntree_initial, mtry=2+i, proximity=TRUE,outscale=TRUE)


  pred.RF.normal.loopmtry<-predict(RF.normal.loopmtry)
  Oobtemp=100*RF.normal.loopmtry$err.rate[ntree_initial]
  Acctemp=confusionMatrix(pred.RF.normal.loopmtry, truth.normal,
positive='1')$overall[1]
  Sstemp=confusionMatrix(pred.RF.normal.loopmtry, truth.normal,
positive='1')$byClass[1]
  Sptemp=confusionMatrix(pred.RF.normal.loopmtry, truth.normal,
positive='1')$byClass[2]

  OOB.RF.normal.loopmtry <- cbind(OOB.RF.normal.loopmtry, Oobtemp)
  Accuracy.RF.normal.loopmtry <- cbind(Accuracy.RF.normal.loopmtry,
Acctemp)
  Sensitivity.RF.normal.loopmtry <- cbind(Sensitivity.RF.normal.loopmtry,
Sstemp)
  Specificity.RF.normal.loopmtry <- cbind(Specificity.RF.normal.loopmtry,
Sptemp)
  num.matrix.RF.normal.loopmtry <- cbind(num.matrix.RF.normal.loopmtry,2+i)
  i=i+1
  }

MATRIX.RF.normal.loopmtry<- data.frame(
  "mtry"=as.numeric(num.matrix.RF.normal.loopmtry),
  "Accuracy"=as.numeric(Accuracy.RF.normal.loopmtry),
  "Sensitivity" = as.numeric(Sensitivity.RF.normal.loopmtry),
  "Specificity" = as.numeric(Specificity.RF.normal.loopmtry),
  "OOB error" = as.numeric(OOB.RF.normal.loopmtry)
```

```
)

MATRIX.RF.normal.loopmtry

windows(title="RF unbalanzed")
par(mfrow = c(2, 2))
plot(x=num.matrix.RF.normal.loopmtry, y=Accuracy.RF.normal.loopmtry,
type="l", col="blue", ylab="Accuracy", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.normal.loopmtry, y=Sensitivity.RF.normal.loopmtry,
type="l", col="blue", ylab="Sensitivity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.normal.loopmtry, y=Specificity.RF.normal.loopmtry,
type="l", col="blue", ylab="Specificity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.normal.loopmtry, y=OOB.RF.normal.loopmtry, type="l",
col="blue", ylab="OOB", xlab="mtry", ylim=c(0, 30))




#set mtry
mtry.RF.normal.opt <-
MATRIX.RF.normal.loopmtry$mtry[min(which(MATRIX.RF.normal.loopmtry$OOB.error
== min(MATRIX.RF.normal.loopmtry$OOB.error)))]




#evaluate ntree
i=1
j=0
#
set.seed(8159780)
RF.normal.loopntree <- randomForest(after.boruta.data,
as.factor(fish.pres), data=after.boruta.data, nodesize = 1, importance=T,
ntree=i, mtry=mtry.RF.normal.opt, proximity=TRUE,outscale=TRUE)

pred.RF.normal.loopntree<-predict(RF.normal.loopntree)

OOB.RF.normal.loopntree <-  100*RF.normal.loopntree$err.rate[i]
Accuracy.RF.normal.loopntree <-confusionMatrix(pred.RF.normal.loopntree,
truth.normal, positive='1')$overall[1]
Sensitivity.RF.normal.loopntree <-
confusionMatrix(pred.RF.normal.loopntree, truth.normal, positive='1')
$byClass[1]
Specificity.RF.normal.loopntree <-
confusionMatrix(pred.RF.normal.loopntree, truth.normal, positive='1')
$byClass[2]
num.matrix.RF.normal.loopntree <- i
#

while (i<ntree_initial) {
  i=i+1
  set.seed(8159780)
  RF.normal.loopntree <- randomForest(after.boruta.data,
as.factor(fish.pres), data=after.boruta.data, nodesize = 1, importance=T,
ntree=i, mtry=mtry.RF.normal.opt, proximity=TRUE,outscale=TRUE)


  pred.RF.normal.loopntree<-predict(RF.normal.loopntree)
  Oobtemp=100*RF.normal.loopntree$err.rate[i]
  Acctemp=confusionMatrix(pred.RF.normal.loopntree, truth.normal,
positive='1')$overall[1]
  Sstemp=confusionMatrix(pred.RF.normal.loopntree, truth.normal,
positive='1')$byClass[1]
  Sptemp=confusionMatrix(pred.RF.normal.loopntree, truth.normal,
positive='1')$byClass[2]

  OOB.RF.normal.loopntree <- cbind(OOB.RF.normal.loopntree, Oobtemp)
  Accuracy.RF.normal.loopntree <- cbind(Accuracy.RF.normal.loopntree,
Acctemp)
  Sensitivity.RF.normal.loopntree <- cbind(Sensitivity.RF.normal.loopntree,
Sstemp)
  Specificity.RF.normal.loopntree <- cbind(Specificity.RF.normal.loopntree,
Sptemp)
  num.matrix.RF.normal.loopntree <- cbind(num.matrix.RF.normal.loopntree,i)
  print(i)
}


MATRIX.RF.normal.loopntree<- data.frame(
  "ntree"=as.numeric(num.matrix.RF.normal.loopntree),
  "Accuracy"=as.numeric(Accuracy.RF.normal.loopntree),
  "Sensitivity" = as.numeric(Sensitivity.RF.normal.loopntree),
  "Specificity" = as.numeric(Specificity.RF.normal.loopntree),
  "OOB error" = as.numeric(OOB.RF.normal.loopntree)
)

MATRIX.RF.normal.loopntree

windows(title="RF unbalanzed")
par(mfrow = c(2, 2))
plot(x=num.matrix.RF.normal.loopntree, y=Accuracy.RF.normal.loopntree,
type="l", col="blue", ylab="Accuracy", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.normal.loopntree, y=Sensitivity.RF.normal.loopntree,
type="l", col="blue", ylab="Sensitivity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.normal.loopntree, y=Specificity.RF.normal.loopntree,
type="l", col="blue", ylab="Specificity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.normal.loopntree, y=OOB.RF.normal.loopntree, type="l",
col="blue", ylab="OOB", xlab="ntree", ylim=c(0, 30))




#set ntree
ntree.RF.normal.opt <-
ceiling((2*(max(which(MATRIX.RF.normal.loopntree$OOB.error !=
MATRIX.RF.normal.loopntree$OOB.error[ntree_initial]))))/100)*100



#model calibrated
set.seed(8159780)
RF.normal.opt <- randomForest(after.boruta.data, as.factor(fish.pres),
data=after.boruta.data, nodesize = 1, importance=T,
ntree=ntree.RF.normal.opt, mtry=mtry.RF.normal.opt,
proximity=TRUE,outscale=TRUE)

CM.RF.normal.opt <- confusionMatrix(predict(RF.normal.opt), truth.normal,
positive='1')
CM.RF.normal.opt
```

```
windows()
plot(predict(RF.normal.opt),truth.normal, main = "Confusion matrix:
Optimized RF unbalanzed")



####################################################
##RF OVERSAMPLING##

#Evaluate mtry
i=1
j=0
mtry.loopmtry=ncol(oversample_model[1:num.var])
#
set.seed(8159780)
RF.over.loopmtry <- randomForest(oversample_model[1:num.var],
as.factor(oversample_model$fish.pres), data=oversample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree_initial, mtry=2,
proximity=TRUE,outscale=TRUE)

pred.RF.over.loopmtry<-predict(RF.over.loopmtry)

OOB.RF.over.loopmtry <-  100*RF.over.loopmtry$err.rate[ntree_initial]
Accuracy.RF.over.loopmtry <-confusionMatrix(pred.RF.over.loopmtry,
truth.over, positive='1')$overall[1]
Sensitivity.RF.over.loopmtry <- confusionMatrix(pred.RF.over.loopmtry,
truth.over, positive='1')$byClass[1]
Specificity.RF.over.loopmtry <- confusionMatrix(pred.RF.over.loopmtry,
truth.over, positive='1')$byClass[2]
num.matrix.RF.over.loopmtry <- 2
#

while ((2+i)<=mtry.loopmtry) {
  set.seed(8159780)
  RF.over.loopmtry <- randomForest(oversample_model[1:num.var],
as.factor(oversample_model$fish.pres), data=oversample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree_initial, mtry=2+i,
proximity=TRUE,outscale=TRUE)


  pred.RF.over.loopmtry<-predict(RF.over.loopmtry)
  Oobtemp=100*RF.over.loopmtry$err.rate[ntree_initial]
  Acctemp=confusionMatrix(pred.RF.over.loopmtry, truth.over, positive='1')
$overall[1]
  Sstemp=confusionMatrix(pred.RF.over.loopmtry, truth.over, positive='1')
$byClass[1]
  Sptemp=confusionMatrix(pred.RF.over.loopmtry, truth.over, positive='1')
$byClass[2]

  OOB.RF.over.loopmtry <- cbind(OOB.RF.over.loopmtry, Oobtemp)
  Accuracy.RF.over.loopmtry <- cbind(Accuracy.RF.over.loopmtry, Acctemp)
  Sensitivity.RF.over.loopmtry <- cbind(Sensitivity.RF.over.loopmtry,
Sstemp)
  Specificity.RF.over.loopmtry <- cbind(Specificity.RF.over.loopmtry,
Sptemp)
  num.matrix.RF.over.loopmtry <- cbind(num.matrix.RF.over.loopmtry,2+i)
  i=i+1
}

MATRIX.RF.over.loopmtry<- data.frame(
  "mtry"=as.numeric(num.matrix.RF.over.loopmtry),
  "Accuracy"=as.numeric(Accuracy.RF.over.loopmtry),
  "Sensitivity" = as.numeric(Sensitivity.RF.over.loopmtry),
  "Specificity" = as.numeric(Specificity.RF.over.loopmtry),
  "OOB error" = as.numeric(OOB.RF.over.loopmtry)
)

MATRIX.RF.over.loopmtry

windows(title="RF oversampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.RF.over.loopmtry, y=Accuracy.RF.over.loopmtry, type="l",
col="blue", ylab="Accuracy", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.over.loopmtry, y=Sensitivity.RF.over.loopmtry,
type="l", col="blue", ylab="Sensitivity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.over.loopmtry, y=Specificity.RF.over.loopmtry,
type="l", col="blue", ylab="Specificity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.over.loopmtry, y=OOB.RF.over.loopmtry, type="l",
col="blue", ylab="OOB", xlab="mtry", ylim=c(0, 30))




#set mtry
mtry.RF.over.opt <-
MATRIX.RF.over.loopmtry$mtry[min(which(MATRIX.RF.over.loopmtry$OOB.error ==
min(MATRIX.RF.over.loopmtry$OOB.error)))]




#evaluate ntree
i=1
j=0
#
set.seed(8159780)
RF.over.loopntree <- randomForest(oversample_model[1:num.var],
as.factor(oversample_model$fish.pres), data=oversample_model[1:num.var],
nodesize = 1, importance=T, ntree=i, mtry=mtry.RF.over.opt,
proximity=TRUE,outscale=TRUE)

pred.RF.over.loopntree<-predict(RF.over.loopntree)

OOB.RF.over.loopntree <-  100*RF.over.loopntree$err.rate[i]
Accuracy.RF.over.loopntree <-confusionMatrix(pred.RF.over.loopntree,
truth.over, positive='1')$overall[1]
Sensitivity.RF.over.loopntree <- confusionMatrix(pred.RF.over.loopntree,
truth.over, positive='1')$byClass[1]
Specificity.RF.over.loopntree <- confusionMatrix(pred.RF.over.loopntree,
truth.over, positive='1')$byClass[2]
num.matrix.RF.over.loopntree <- i
#

while (i<ntree_initial) {
  i=i+1
  set.seed(8159780)
  RF.over.loopntree <- randomForest(oversample_model[1:num.var],
as.factor(oversample_model$fish.pres), data=oversample_model[1:num.var],
nodesize = 1, importance=T, ntree=i, mtry=mtry.RF.over.opt,
proximity=TRUE,outscale=TRUE)
```

```
  pred.RF.over.loopntree<-predict(RF.over.loopntree)
  Oobtemp=100*RF.over.loopntree$err.rate[i]
  Acctemp=confusionMatrix(pred.RF.over.loopntree, truth.over, positive='1')
$overall[1]
  Sstemp=confusionMatrix(pred.RF.over.loopntree, truth.over, positive='1')
$byClass[1]
  Sptemp=confusionMatrix(pred.RF.over.loopntree, truth.over, positive='1')
$byClass[2]

  OOB.RF.over.loopntree <- cbind(OOB.RF.over.loopntree, Oobtemp)
  Accuracy.RF.over.loopntree <- cbind(Accuracy.RF.over.loopntree, Acctemp)
  Sensitivity.RF.over.loopntree <- cbind(Sensitivity.RF.over.loopntree,
Sstemp)
  Specificity.RF.over.loopntree <- cbind(Specificity.RF.over.loopntree,
Sptemp)
  num.matrix.RF.over.loopntree <- cbind(num.matrix.RF.over.loopntree,i)
  print(i)

}


MATRIX.RF.over.loopntree<- data.frame(
  "ntree"=as.numeric(num.matrix.RF.over.loopntree),
  "Accuracy"=as.numeric(Accuracy.RF.over.loopntree),
  "Sensitivity" = as.numeric(Sensitivity.RF.over.loopntree),
  "Specificity" = as.numeric(Specificity.RF.over.loopntree),
  "OOB error" = as.numeric(OOB.RF.over.loopntree)
)

MATRIX.RF.over.loopntree

windows(title="RF oversampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.RF.over.loopntree, y=Accuracy.RF.over.loopntree,
type="l", col="blue", ylab="Accuracy", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.over.loopntree, y=Sensitivity.RF.over.loopntree,
type="l", col="blue", ylab="Sensitivity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.over.loopntree, y=Specificity.RF.over.loopntree,
type="l", col="blue", ylab="Specificity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.over.loopntree, y=OOB.RF.over.loopntree, type="l",
col="blue", ylab="OOB", xlab="ntree", ylim=c(0, 30))




#set ntree
ntree.RF.over.opt <-
ceiling((2*(max(which(MATRIX.RF.over.loopntree$OOB.error !=
MATRIX.RF.over.loopntree$OOB.error[ntree_initial]))))/100)*100



#final model
set.seed(8159780)
RF.over.opt <- randomForest(oversample_model[1:num.var],
as.factor(oversample_model$fish.pres), data=oversample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree.RF.over.opt, mtry=mtry.RF.over.opt,
proximity=TRUE,outscale=TRUE)
```

```
CM.RF.over.opt <- confusionMatrix(predict(RF.over.opt), truth.over,
positive='1')
CM.RF.over.opt
windows()
plot(predict(RF.over.opt),truth.over, main = "Confusion matrix: Optimized
RF oversampling")



####################################################
##RF UNDERSAMPLIMG##

#Evaluate mtry
i=1
j=0
mtry.loopmtry=ncol(undersample_model[1:num.var])
#
set.seed(8159780)
RF.under.loopmtry <- randomForest(undersample_model[1:num.var],
as.factor(undersample_model$fish.pres), data=undersample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree_initial, mtry=2,
proximity=TRUE,outscale=TRUE)

pred.RF.under.loopmtry<-predict(RF.under.loopmtry)

OOB.RF.under.loopmtry <-  100*RF.under.loopmtry$err.rate[ntree_initial]
Accuracy.RF.under.loopmtry <-confusionMatrix(pred.RF.under.loopmtry,
truth.under, positive='1')$overall[1]
Sensitivity.RF.under.loopmtry <- confusionMatrix(pred.RF.under.loopmtry,
truth.under, positive='1')$byClass[1]
Specificity.RF.under.loopmtry <- confusionMatrix(pred.RF.under.loopmtry,
truth.under, positive='1')$byClass[2]
num.matrix.RF.under.loopmtry <- 2
#

while ((2+i)<=mtry.loopmtry) {
  set.seed(8159780)
  RF.under.loopmtry <- randomForest(undersample_model[1:num.var],
as.factor(undersample_model$fish.pres), data=undersample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree_initial, mtry=2+i,
proximity=TRUE,outscale=TRUE)


  pred.RF.under.loopmtry<-predict(RF.under.loopmtry)
  Oobtemp=100*RF.under.loopmtry$err.rate[ntree_initial]
  Acctemp=confusionMatrix(pred.RF.under.loopmtry, truth.under,
positive='1')$overall[1]
  Sstemp=confusionMatrix(pred.RF.under.loopmtry, truth.under, positive='1')
$byClass[1]
  Sptemp=confusionMatrix(pred.RF.under.loopmtry, truth.under, positive='1')
$byClass[2]

  OOB.RF.under.loopmtry <- cbind(OOB.RF.under.loopmtry, Oobtemp)
  Accuracy.RF.under.loopmtry <- cbind(Accuracy.RF.under.loopmtry, Acctemp)
  Sensitivity.RF.under.loopmtry <- cbind(Sensitivity.RF.under.loopmtry,
Sstemp)
  Specificity.RF.under.loopmtry <- cbind(Specificity.RF.under.loopmtry,
Sptemp)
  num.matrix.RF.under.loopmtry <- cbind(num.matrix.RF.under.loopmtry,2+i)
  i=i+1
}

MATRIX.RF.under.loopmtry<- data.frame(
  "mtry"=as.numeric(num.matrix.RF.under.loopmtry),
  "Accuracy"=as.numeric(Accuracy.RF.under.loopmtry),
  "Sensitivity" = as.numeric(Sensitivity.RF.under.loopmtry),
  "Specificity" = as.numeric(Specificity.RF.under.loopmtry),
  "OOB error" = as.numeric(OOB.RF.under.loopmtry)
)

MATRIX.RF.under.loopmtry

windows(title="RF undersampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.RF.under.loopmtry, y=Accuracy.RF.under.loopmtry,
type="l", col="blue", ylab="Accuracy", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.under.loopmtry, y=Sensitivity.RF.under.loopmtry,
type="l", col="blue", ylab="Sensitivity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.under.loopmtry, y=Specificity.RF.under.loopmtry,
type="l", col="blue", ylab="Specificity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.RF.under.loopmtry, y=OOB.RF.under.loopmtry, type="l",
col="blue", ylab="OOB", xlab="mtry", ylim=c(0, 30))




#set mtry
mtry.RF.under.opt <-
MATRIX.RF.under.loopmtry$mtry[min(which(MATRIX.RF.under.loopmtry$OOB.error
== min(MATRIX.RF.under.loopmtry$OOB.error)))]




#Evaluate ntree
i=1
j=0
#
set.seed(8159780)
RF.under.loopntree <- randomForest(undersample_model[1:num.var],
as.factor(undersample_model$fish.pres), data=undersample_model[1:num.var],
nodesize = 1, importance=T, ntree=i, mtry=mtry.RF.under.opt,
proximity=TRUE,outscale=TRUE)

pred.RF.under.loopntree<-predict(RF.under.loopntree)

OOB.RF.under.loopntree <-  100*RF.under.loopntree$err.rate[i]
Accuracy.RF.under.loopntree <-confusionMatrix(pred.RF.under.loopntree,
truth.under, positive='1')$overall[1]
Sensitivity.RF.under.loopntree <- confusionMatrix(pred.RF.under.loopntree,
truth.under, positive='1')$byClass[1]
Specificity.RF.under.loopntree <- confusionMatrix(pred.RF.under.loopntree,
truth.under, positive='1')$byClass[2]
num.matrix.RF.under.loopntree <- i
#

while (i<ntree_initial) {
  i=i+1
  set.seed(8159780)
  RF.under.loopntree <- randomForest(undersample_model[1:num.var],
as.factor(undersample_model$fish.pres), data=undersample_model[1:num.var],
nodesize = 1, importance=T, ntree=i, mtry=mtry.RF.under.opt,
proximity=TRUE,outscale=TRUE)


  pred.RF.under.loopntree<-predict(RF.under.loopntree)
  Oobtemp=100*RF.under.loopntree$err.rate[i]
  Acctemp=confusionMatrix(pred.RF.under.loopntree, truth.under,
positive='1')$overall[1]
  Sstemp=confusionMatrix(pred.RF.under.loopntree, truth.under,
positive='1')$byClass[1]
  Sptemp=confusionMatrix(pred.RF.under.loopntree, truth.under,
positive='1')$byClass[2]

  OOB.RF.under.loopntree <- cbind(OOB.RF.under.loopntree, Oobtemp)
  Accuracy.RF.under.loopntree <- cbind(Accuracy.RF.under.loopntree,
Acctemp)
  Sensitivity.RF.under.loopntree <- cbind(Sensitivity.RF.under.loopntree,
Sstemp)
  Specificity.RF.under.loopntree <- cbind(Specificity.RF.under.loopntree,
Sptemp)
  num.matrix.RF.under.loopntree <- cbind(num.matrix.RF.under.loopntree,i)
  print(i)

}


MATRIX.RF.under.loopntree<- data.frame(
  "ntree"=as.numeric(num.matrix.RF.under.loopntree),
  "Accuracy"=as.numeric(Accuracy.RF.under.loopntree),
  "Sensitivity" = as.numeric(Sensitivity.RF.under.loopntree),
  "Specificity" = as.numeric(Specificity.RF.under.loopntree),
  "OOB error" = as.numeric(OOB.RF.under.loopntree)
)

MATRIX.RF.under.loopntree

windows(title="RF undersampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.RF.under.loopntree, y=Accuracy.RF.under.loopntree,
type="l", col="blue", ylab="Accuracy", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.under.loopntree, y=Sensitivity.RF.under.loopntree,
type="l", col="blue", ylab="Sensitivity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.under.loopntree, y=Specificity.RF.under.loopntree,
type="l", col="blue", ylab="Specificity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.RF.under.loopntree, y=OOB.RF.under.loopntree, type="l",
col="blue", ylab="OOB", xlab="ntree", ylim=c(0, 30))




#Set ntree
ntree.RF.under.opt <-
ceiling((2*(max(which(MATRIX.RF.under.loopntree$OOB.error !=
MATRIX.RF.under.loopntree$OOB.error[ntree_initial]))))/100)*100



#Final models
```

```
set.seed(8159780)
RF.under.opt <- randomForest(undersample_model[1:num.var],
as.factor(undersample_model$fish.pres), data=undersample_model[1:num.var],
nodesize = 1, importance=T, ntree=ntree.RF.under.opt,
mtry=mtry.RF.under.opt, proximity=TRUE,outscale=TRUE)

CM.RF.under.opt <- confusionMatrix(predict(RF.under.opt), truth.under,
positive='1')
CM.RF.under.opt
windows()
plot(predict(RF.under.opt),truth.under, main = "Confusion matrix: Optimized
RF undersampling")



####################################################
##CIF UNBALANZED##

#Evaluate mtry
i=1
j=0
mtry.loopmtry=ncol(after.boruta.data)
#
set.seed(8159780)
CIF.normal.loopmtry<- cforest(factor(fish.pres) ~., data=after.boruta.data,
                      control = ctree_control(teststat = "max", testtype =
"Teststatistic",
                                              mincriterion = 0, saveinfo =
TRUE),
                      ytrafo = NULL, scores = NULL, ntree = ntree_initial,
                      perturb = list(replace = TRUE), #fraction = 1#
                      applyfun = NULL,
                      cores = NULL, mtry = 2, trace = TRUE)

pred.CIF.normal.loopmtry<-predict(CIF.normal.loopmtry)

OOB.CIF.normal.loopmtry <-  100 *
sum(abs(abs(as.numeric(pred.CIF.normal.loopmtry) - 1) -
abs(as.numeric(fish.pres))))/length(fish.pres)
Accuracy.CIF.normal.loopmtry <- confusionMatrix(pred.CIF.normal.loopmtry,
truth.normal, positive='1')$overall[1]
Sensitivity.CIF.normal.loopmtry <-
confusionMatrix(pred.CIF.normal.loopmtry, truth.normal, positive='1')
$byClass[1]
Specificity.CIF.normal.loopmtry <-
confusionMatrix(pred.CIF.normal.loopmtry, truth.normal, positive='1')
$byClass[2]
num.matrix.CIF.normal.loopmtry <- 2
#

while ((2+i)<=mtry.loopmtry) {
  set.seed(8159780)
  CIF.normal.loopmtry<- cforest(factor(fish.pres) ~.,
data=after.boruta.data,
                                control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                        mincriterion = 0,
saveinfo = TRUE),
                                ytrafo = NULL, scores = NULL, ntree =
ntree_initial,
                                perturb = list(replace = TRUE), #fraction =
1#
                                applyfun = NULL,
                                cores = NULL, mtry = 2+i, trace = TRUE)

  pred.CIF.normal.loopmtry<-predict(CIF.normal.loopmtry)
  Oobtemp=100 * sum(abs(abs(as.numeric(pred.CIF.normal.loopmtry) - 1) -
abs(as.numeric(fish.pres))))/length(fish.pres)
  Acctemp=confusionMatrix(pred.CIF.normal.loopmtry, truth.normal,
positive='1')$overall[1]
  Sstemp=confusionMatrix(pred.CIF.normal.loopmtry, truth.normal,
positive='1')$byClass[1]
  Sptemp=confusionMatrix(pred.CIF.normal.loopmtry, truth.normal,
positive='1')$byClass[2]

  OOB.CIF.normal.loopmtry <- cbind(OOB.CIF.normal.loopmtry, Oobtemp)
  Accuracy.CIF.normal.loopmtry <- cbind(Accuracy.CIF.normal.loopmtry,
Acctemp)
  Sensitivity.CIF.normal.loopmtry <- cbind(Sensitivity.CIF.normal.loopmtry,
Sstemp)
  Specificity.CIF.normal.loopmtry <- cbind(Specificity.CIF.normal.loopmtry,
Sptemp)
  num.matrix.CIF.normal.loopmtry <- cbind(num.matrix.CIF.normal.loopmtry,
2+i)
  i=i+1
}

MATRIX.CIF.normal.loopmtry<- data.frame(
  "mtry"=as.numeric(num.matrix.CIF.normal.loopmtry),
  "Accuracy"=as.numeric(Accuracy.CIF.normal.loopmtry),
  "Sensitivity" = as.numeric(Sensitivity.CIF.normal.loopmtry),
  "Specificity" = as.numeric(Specificity.CIF.normal.loopmtry),
  "OOB error" = as.numeric(OOB.CIF.normal.loopmtry)
)

MATRIX.CIF.normal.loopmtry

windows(title="CIF unbalanzed")
par(mfrow = c(2, 2))
plot(x=num.matrix.CIF.normal.loopmtry, y=Accuracy.CIF.normal.loopmtry,
type="l", col="blue", ylab="Accuracy", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.normal.loopmtry, y=Sensitivity.CIF.normal.loopmtry,
type="l", col="blue", ylab="Sensitivity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.normal.loopmtry, y=Specificity.CIF.normal.loopmtry,
type="l", col="blue", ylab="Specificity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.normal.loopmtry, y=OOB.CIF.normal.loopmtry, type="l",
col="blue", ylab="OOB", xlab="mtry", ylim=c(0, 30))




#set mtry
mtry.CIF.normal.opt <-
MATRIX.CIF.normal.loopmtry$mtry[min(which(MATRIX.CIF.normal.loopmtry$OOB.error
== min(MATRIX.CIF.normal.loopmtry$OOB.error)))]




#Evaluate ntree
```

```
i=1
j=1
#
set.seed(8159780)
CIF.normal.loopntree<- cforest(factor(fish.pres) ~.,
data=after.boruta.data,
                              control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                      mincriterion = 0,
saveinfo = TRUE),
                              ytrafo = NULL, scores = NULL, ntree = i,
                              perturb = list(replace = TRUE), #fraction =
1#
                              applyfun = NULL,
                              cores = NULL, mtry = mtry.CIF.normal.opt,
trace = TRUE)

pred.CIF.normal.loopntree<-predict(CIF.normal.loopntree)

OOB.CIF.normal.loopntree <-  100 *
sum(abs(abs(as.numeric(pred.CIF.normal.loopntree) - 1) -
abs(as.numeric(fish.pres))))/length(fish.pres)
Accuracy.CIF.normal.loopntree <-confusionMatrix(pred.CIF.normal.loopntree,
truth.normal, positive='1')$overall[1]
Sensitivity.CIF.normal.loopntree <-
confusionMatrix(pred.CIF.normal.loopntree, truth.normal, positive='1')
$byClass[1]
Specificity.CIF.normal.loopntree <-
confusionMatrix(pred.CIF.normal.loopntree, truth.normal, positive='1')
$byClass[2]
num.matrix.CIF.normal.loopntree <- i
#

while (i<ntree_initial) {
  if(i<100)
  {
    j=1+j
    i=i+1
    set.seed(8159780)
    CIF.normal.loopntree<- cforest(factor(fish.pres) ~.,
data=after.boruta.data,
                                  control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                          mincriterion = 0,
saveinfo = TRUE),
                                  ytrafo = NULL, scores = NULL, ntree = i,
                                  perturb = list(replace = TRUE), #fraction
= 1#
                                  applyfun = NULL,
                                  cores = NULL, mtry = mtry.CIF.normal.opt,
trace = TRUE)


    pred.CIF.normal.loopntree<-predict(CIF.normal.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.normal.loopntree) - 1) -
abs(as.numeric(fish.pres))))/length(fish.pres)
    Acctemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$byClass[1]
```

```
    Sptemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$byClass[2]

    OOB.CIF.normal.loopntree <- cbind(OOB.CIF.normal.loopntree, Oobtemp)
    Accuracy.CIF.normal.loopntree <- cbind(Accuracy.CIF.normal.loopntree,
Acctemp)
    Sensitivity.CIF.normal.loopntree <-
cbind(Sensitivity.CIF.normal.loopntree, Sstemp)
    Specificity.CIF.normal.loopntree <-
cbind(Specificity.CIF.normal.loopntree, Sptemp)
    num.matrix.CIF.normal.loopntree <-
cbind(num.matrix.CIF.normal.loopntree,i)

    print(i)
  }

  if(i>=100 && i<500)
  {
    j=1+j
    i=i+10
    set.seed(8159780)
    CIF.normal.loopntree<- cforest(factor(fish.pres) ~.,
data=after.boruta.data,
                                   control = ctree_control(teststat =
"max", testtype = "Teststatistic",
                                                           mincriterion =
0, saveinfo = TRUE),
                                   ytrafo = NULL, scores = NULL, ntree = i,
                                   perturb = list(replace = TRUE),
#fraction = 1#
                                   applyfun = NULL,
                                   cores = NULL, mtry =
mtry.CIF.normal.opt, trace = TRUE)


    pred.CIF.normal.loopntree<-predict(CIF.normal.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.normal.loopntree) - 1) -
abs(as.numeric(fish.pres))))/length(fish.pres)
    Acctemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$byClass[2]

    OOB.CIF.normal.loopntree <- cbind(OOB.CIF.normal.loopntree, Oobtemp)
    Accuracy.CIF.normal.loopntree <- cbind(Accuracy.CIF.normal.loopntree,
Acctemp)
    Sensitivity.CIF.normal.loopntree <-
cbind(Sensitivity.CIF.normal.loopntree, Sstemp)
    Specificity.CIF.normal.loopntree <-
cbind(Specificity.CIF.normal.loopntree, Sptemp)
    num.matrix.CIF.normal.loopntree <-
cbind(num.matrix.CIF.normal.loopntree,i)

    print(i)
  }

  if(i>=500 && i<ntree_initial)
  {
    j=1+j
    i=i+20
    set.seed(8159780)
    CIF.normal.loopntree<- cforest(factor(fish.pres) ~.,
data=after.boruta.data,
                                   control = ctree_control(teststat =
"max", testtype = "Teststatistic",
                                                           mincriterion =
0, saveinfo = TRUE),
                                   ytrafo = NULL, scores = NULL, ntree = i,
                                   perturb = list(replace = TRUE),
#fraction = 1#
                                   applyfun = NULL,
                                   cores = NULL, mtry =
mtry.CIF.normal.opt, trace = TRUE)


    pred.CIF.normal.loopntree<-predict(CIF.normal.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.normal.loopntree) - 1) -
abs(as.numeric(fish.pres))))/length(fish.pres)
    Acctemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.normal.loopntree, truth.normal,
positive='1')$byClass[2]

    OOB.CIF.normal.loopntree <- cbind(OOB.CIF.normal.loopntree, Oobtemp)
    Accuracy.CIF.normal.loopntree <- cbind(Accuracy.CIF.normal.loopntree,
Acctemp)
    Sensitivity.CIF.normal.loopntree <-
cbind(Sensitivity.CIF.normal.loopntree, Sstemp)
    Specificity.CIF.normal.loopntree <-
cbind(Specificity.CIF.normal.loopntree, Sptemp)
    num.matrix.CIF.normal.loopntree <-
cbind(num.matrix.CIF.normal.loopntree,i)

    print(i)
  }
}


MATRIX.CIF.normal.loopntree<- data.frame(
  "ntree"=as.numeric(num.matrix.CIF.normal.loopntree),
  "Accuracy"=as.numeric(Accuracy.CIF.normal.loopntree),
  "Sensitivity" = as.numeric(Sensitivity.CIF.normal.loopntree),
  "Specificity" = as.numeric(Specificity.CIF.normal.loopntree),
  "OOB error" = as.numeric(OOB.CIF.normal.loopntree)
)

MATRIX.CIF.normal.loopntree

windows(title="CIF unbalanzed")
par(mfrow = c(2, 2))
plot(x=num.matrix.CIF.normal.loopntree, y=Accuracy.CIF.normal.loopntree,
type="l", col="blue", ylab="Accuracy", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.normal.loopntree, y=Sensitivity.CIF.normal.loopntree,
type="l", col="blue", ylab="Sensitivity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.normal.loopntree, y=Specificity.CIF.normal.loopntree,
type="l", col="blue", ylab="Specificity", xlab="ntree", ylim=c(0.6, 1))
```

```
plot(x=num.matrix.CIF.normal.loopntree, y=OOB.CIF.normal.loopntree,
type="l", col="blue", ylab="OOB", xlab="ntree", ylim=c(0, 30))




#set ntree
ntree.CIF.normal <-
MATRIX.CIF.normal.loopntree$ntree[max(which(MATRIX.CIF.normal.loopntree$OOB.error
!= MATRIX.CIF.normal.loopntree$OOB.error[j]))]
ntree.CIF.normal.opt <-
ceiling((2*(MATRIX.CIF.normal.loopntree$ntree[max(which(MATRIX.CIF.normal.loopntree$
!= MATRIX.CIF.normal.loopntree$OOB.error[j]))]))/100)*100



#final model
set.seed(8159780)
CIF.normal.opt <- cforest(factor(fish.pres) ~., data=after.boruta.data,
                          control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                  mincriterion = 0,
saveinfo = TRUE),
                          ytrafo = NULL, scores = NULL, ntree =
ntree.CIF.normal.opt,
                          perturb = list(replace = TRUE), #fraction = 1#
                          applyfun = NULL,
                          cores = NULL, mtry = mtry.CIF.normal.opt, trace =
TRUE)


CM.CIF.normal.opt <- confusionMatrix(predict(CIF.normal.opt), truth.normal,
positive='1')
CM.CIF.normal.opt
windows()
plot(predict(CIF.normal.opt),truth.normal, main = "Confusion matrix:
Optimized CIF unbalanzed")



####################################################
##CIF OVERSAMPLING##

#Evaluate mtry
i=1
j=0
mtry.loopmtry=ncol(oversample_model[1:num.var])
#
set.seed(8159780)
CIF.over.loopmtry<- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model[1:num.var],
                            control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                    mincriterion = 0,
saveinfo = TRUE),
                            ytrafo = NULL, scores = NULL, ntree =
ntree_initial,
                            perturb = list(replace = TRUE), #fraction = 1#
                            applyfun = NULL,
                            cores = NULL, mtry = 2, trace = TRUE)
```

```
pred.CIF.over.loopmtry<-predict(CIF.over.loopmtry)

OOB.CIF.over.loopmtry <-  100 *
sum(abs(abs(as.numeric(pred.CIF.over.loopmtry) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)
Accuracy.CIF.over.loopmtry <- confusionMatrix(pred.CIF.over.loopmtry,
truth.over, positive='1')$overall[1]
Sensitivity.CIF.over.loopmtry <- confusionMatrix(pred.CIF.over.loopmtry,
truth.over, positive='1')$byClass[1]
Specificity.CIF.over.loopmtry <- confusionMatrix(pred.CIF.over.loopmtry,
truth.over, positive='1')$byClass[2]
num.matrix.CIF.over.loopmtry <- 2
#

while ((2+i)<=mtry.loopmtry) {
  set.seed(8159780)
  CIF.over.loopmtry<- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model,
                              control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                       mincriterion = 0,
saveinfo = TRUE),
                              ytrafo = NULL, scores = NULL, ntree =
ntree_initial,
                              perturb = list(replace = TRUE), #fraction =
1#
                              applyfun = NULL,
                              cores = NULL, mtry = 2+i, trace = TRUE)

  pred.CIF.over.loopmtry<-predict(CIF.over.loopmtry)
  Oobtemp=100 * sum(abs(abs(as.numeric(pred.CIF.over.loopmtry) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)
  Acctemp=confusionMatrix(pred.CIF.over.loopmtry, truth.over, positive='1')
$overall[1]
  Sstemp=confusionMatrix(pred.CIF.over.loopmtry, truth.over, positive='1')
$byClass[1]
  Sptemp=confusionMatrix(pred.CIF.over.loopmtry, truth.over, positive='1')
$byClass[2]

  OOB.CIF.over.loopmtry <- cbind(OOB.CIF.over.loopmtry, Oobtemp)
  Accuracy.CIF.over.loopmtry <- cbind(Accuracy.CIF.over.loopmtry, Acctemp)
  Sensitivity.CIF.over.loopmtry <- cbind(Sensitivity.CIF.over.loopmtry,
Sstemp)
  Specificity.CIF.over.loopmtry <- cbind(Specificity.CIF.over.loopmtry,
Sptemp)
  num.matrix.CIF.over.loopmtry <- cbind(num.matrix.CIF.over.loopmtry,2+i)
  i=i+1
}

MATRIX.CIF.over.loopmtry<- data.frame(
  "mtry"=as.numeric(num.matrix.CIF.over.loopmtry),
  "Accuracy"=as.numeric(Accuracy.CIF.over.loopmtry),
  "Sensitivity" = as.numeric(Sensitivity.CIF.over.loopmtry),
  "Specificity" = as.numeric(Specificity.CIF.over.loopmtry),
  "OOB error" = as.numeric(OOB.CIF.over.loopmtry)
)
```

```
MATRIX.CIF.over.loopmtry

windows(title="CIF oversampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.CIF.over.loopmtry, y=Accuracy.CIF.over.loopmtry,
type="l", col="blue", ylab="Accuracy", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.over.loopmtry, y=Sensitivity.CIF.over.loopmtry,
type="l", col="blue", ylab="Sensitivity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.over.loopmtry, y=Specificity.CIF.over.loopmtry,
type="l", col="blue", ylab="Specificity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.over.loopmtry, y=OOB.CIF.over.loopmtry, type="l",
col="blue", ylab="OOB", xlab="mtry", ylim=c(0, 30))




#set mtry
mtry.CIF.over.opt <-
MATRIX.CIF.over.loopmtry$mtry[min(which(MATRIX.CIF.over.loopmtry$OOB.error
== min(MATRIX.CIF.over.loopmtry$OOB.error)))]




#Evaluate ntree
i=1
j=1
#
set.seed(8159780)
CIF.over.loopntree<- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model,
                             control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                     mincriterion = 0,
saveinfo = TRUE),
                             ytrafo = NULL, scores = NULL, ntree = i,
                             perturb = list(replace = TRUE), #fraction = 1#
                             applyfun = NULL,
                             cores = NULL, mtry = mtry.CIF.over.opt, trace
= TRUE)

pred.CIF.over.loopntree<-predict(CIF.over.loopntree)

OOB.CIF.over.loopntree <-  100 *
sum(abs(abs(as.numeric(pred.CIF.over.loopntree) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)
Accuracy.CIF.over.loopntree <-confusionMatrix(pred.CIF.over.loopntree,
truth.over, positive='1')$overall[1]
Sensitivity.CIF.over.loopntree <- confusionMatrix(pred.CIF.over.loopntree,
truth.over, positive='1')$byClass[1]
Specificity.CIF.over.loopntree <- confusionMatrix(pred.CIF.over.loopntree,
truth.over, positive='1')$byClass[2]
num.matrix.CIF.over.loopntree <- i
#

while (i<ntree_initial) {
  if(i<100)
  {
    j=1+j
    i=i+1
    set.seed(8159780)
    CIF.over.loopntree<- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model,
                                 control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                         mincriterion = 0,
saveinfo = TRUE),
                                 ytrafo = NULL, scores = NULL, ntree = i,
                                 perturb = list(replace = TRUE), #fraction
= 1#
                                 applyfun = NULL,
                                 cores = NULL, mtry = mtry.CIF.over.opt,
trace = TRUE)


    pred.CIF.over.loopntree<-predict(CIF.over.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.over.loopntree) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)
    Acctemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$byClass[2]

    OOB.CIF.over.loopntree <- cbind(OOB.CIF.over.loopntree, Oobtemp)
    Accuracy.CIF.over.loopntree <- cbind(Accuracy.CIF.over.loopntree,
Acctemp)
    Sensitivity.CIF.over.loopntree <- cbind(Sensitivity.CIF.over.loopntree,
Sstemp)
    Specificity.CIF.over.loopntree <- cbind(Specificity.CIF.over.loopntree,
Sptemp)
    num.matrix.CIF.over.loopntree <- cbind(num.matrix.CIF.over.loopntree,i)

    print(i)
  }

  if(i>=100 && i<500)
  {
    j=1+j
    i=i+10
    set.seed(8159780)
    CIF.over.loopntree<- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model,
                                 control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                         mincriterion = 0,
saveinfo = TRUE),
                                 ytrafo = NULL, scores = NULL, ntree = i,
                                 perturb = list(replace = TRUE), #fraction
= 1#
                                 applyfun = NULL,
                                 cores = NULL, mtry = mtry.CIF.over.opt,
trace = TRUE)


    pred.CIF.over.loopntree<-predict(CIF.over.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.over.loopntree) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)
    Acctemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$byClass[2]

    OOB.CIF.over.loopntree <- cbind(OOB.CIF.over.loopntree, Oobtemp)
    Accuracy.CIF.over.loopntree <- cbind(Accuracy.CIF.over.loopntree,
Acctemp)
    Sensitivity.CIF.over.loopntree <- cbind(Sensitivity.CIF.over.loopntree,
Sstemp)
    Specificity.CIF.over.loopntree <- cbind(Specificity.CIF.over.loopntree,
Sptemp)
    num.matrix.CIF.over.loopntree <- cbind(num.matrix.CIF.over.loopntree,i)

    print(i)
  }

  if(i>=500 && i<ntree_initial)
  {
    j=1+j
    i=i+20
    set.seed(8159780)
    CIF.over.loopntree<- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model,
                             control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                     mincriterion = 0,
saveinfo = TRUE),
                             ytrafo = NULL, scores = NULL, ntree = i,
                             perturb = list(replace = TRUE), #fraction
= 1#
                             applyfun = NULL,
                             cores = NULL, mtry = mtry.CIF.over.opt,
trace = TRUE)


    pred.CIF.over.loopntree<-predict(CIF.over.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.over.loopntree) - 1) -
abs(as.numeric(oversample_model$fish.pres)-1)))/
length(oversample_model$fish.pres)
    Acctemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.over.loopntree, truth.over,
positive='1')$byClass[2]

    OOB.CIF.over.loopntree <- cbind(OOB.CIF.over.loopntree, Oobtemp)
    Accuracy.CIF.over.loopntree <- cbind(Accuracy.CIF.over.loopntree,
Acctemp)
    Sensitivity.CIF.over.loopntree <- cbind(Sensitivity.CIF.over.loopntree,
Sstemp)
    Specificity.CIF.over.loopntree <- cbind(Specificity.CIF.over.loopntree,
Sptemp)
    num.matrix.CIF.over.loopntree <- cbind(num.matrix.CIF.over.loopntree,i)
    print(i)
  }
}


MATRIX.CIF.over.loopntree<- data.frame(
  "ntree"=as.numeric(num.matrix.CIF.over.loopntree),
  "Accuracy"=as.numeric(Accuracy.CIF.over.loopntree),
  "Sensitivity" = as.numeric(Sensitivity.CIF.over.loopntree),
  "Specificity" = as.numeric(Specificity.CIF.over.loopntree),
  "OOB error" = as.numeric(OOB.CIF.over.loopntree)
)

MATRIX.CIF.over.loopntree

windows(title="CIF oversampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.CIF.over.loopntree, y=Accuracy.CIF.over.loopntree,
type="l", col="blue", ylab="Accuracy", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.over.loopntree, y=Sensitivity.CIF.over.loopntree,
type="l", col="blue", ylab="Sensitivity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.over.loopntree, y=Specificity.CIF.over.loopntree,
type="l", col="blue", ylab="Specificity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.over.loopntree, y=OOB.CIF.over.loopntree, type="l",
col="blue", ylab="OOB", xlab="ntree", ylim=c(0, 30))




#set ntree
ntree.CIF.over <-
MATRIX.CIF.over.loopntree$ntree[max(which(MATRIX.CIF.over.loopntree$OOB.error
!= MATRIX.CIF.over.loopntree$OOB.error[j]))]
ntree.CIF.over.opt <-
ceiling((2*(MATRIX.CIF.over.loopntree$ntree[max(which(MATRIX.CIF.over.loopntree$OOB.
!= MATRIX.CIF.over.loopntree$OOB.error[j]))]))/100)*100



#final model
set.seed(8159780)
CIF.over.opt <- cforest(factor(oversample_model$fish.pres) ~.,
data=oversample_model,
                        control = ctree_control(teststat = "max", testtype
= "Teststatistic",
                                                mincriterion = 0, saveinfo
= TRUE),
                        ytrafo = NULL, scores = NULL, ntree =
ntree.CIF.over.opt,
                        perturb = list(replace = TRUE), #fraction = 1#
                        applyfun = NULL,
                        cores = NULL, mtry = mtry.CIF.over.opt, trace =
TRUE)


CM.CIF.over.opt <- confusionMatrix(predict(CIF.over.opt), truth.over,
positive='1')
CM.CIF.over.opt
windows()
```

```
plot(predict(CIF.over.opt),truth.over, main = "Confusion matrix: Optimized
CIF oversampling")



####################################################
##CIF UNDERSAMPLING##

#Evaluate mtry
i=1
j=0
mtry.loopmtry=ncol(undersample_model[1:num.var])
#
set.seed(8159780)
CIF.under.loopmtry<- cforest(factor(undersample_model$fish.pres) ~.,
data=undersample_model[1:num.var],
                             control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                     mincriterion = 0,
saveinfo = TRUE),
                             ytrafo = NULL, scores = NULL, ntree =
ntree_initial,
                             perturb = list(replace = TRUE), #fraction = 1#
                             applyfun = NULL,
                             cores = NULL, mtry = 2, trace = TRUE)

pred.CIF.under.loopmtry<-predict(CIF.under.loopmtry)

OOB.CIF.under.loopmtry <-  100 *
sum(abs(abs(as.numeric(pred.CIF.under.loopmtry) - 1) -
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)
Accuracy.CIF.under.loopmtry <- confusionMatrix(pred.CIF.under.loopmtry,
truth.under, positive='1')$overall[1]
Sensitivity.CIF.under.loopmtry <- confusionMatrix(pred.CIF.under.loopmtry,
truth.under, positive='1')$byClass[1]
Specificity.CIF.under.loopmtry <- confusionMatrix(pred.CIF.under.loopmtry,
truth.under, positive='1')$byClass[2]
num.matrix.CIF.under.loopmtry <- 2
#

while ((2+i)<=mtry.loopmtry) {
  set.seed(8159780)
  CIF.under.loopmtry<- cforest(factor(undersample_model$fish.pres) ~.,
data=undersample_model,
                               control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                       mincriterion = 0,
saveinfo = TRUE),
                               ytrafo = NULL, scores = NULL, ntree =
ntree_initial,
                               perturb = list(replace = TRUE), #fraction =
1#
                               applyfun = NULL,
                               cores = NULL, mtry = 2+i, trace = TRUE)

  pred.CIF.under.loopmtry<-predict(CIF.under.loopmtry)
  Oobtemp=100 * sum(abs(abs(as.numeric(pred.CIF.under.loopmtry) - 1) -
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)
```

```
  Acctemp=confusionMatrix(pred.CIF.under.loopmtry, truth.under,
positive='1')$overall[1]
  Sstemp=confusionMatrix(pred.CIF.under.loopmtry, truth.under,
positive='1')$byClass[1]
  Sptemp=confusionMatrix(pred.CIF.under.loopmtry, truth.under,
positive='1')$byClass[2]

  OOB.CIF.under.loopmtry <- cbind(OOB.CIF.under.loopmtry, Oobtemp)
  Accuracy.CIF.under.loopmtry <- cbind(Accuracy.CIF.under.loopmtry,
Acctemp)
  Sensitivity.CIF.under.loopmtry <- cbind(Sensitivity.CIF.under.loopmtry,
Sstemp)
  Specificity.CIF.under.loopmtry <- cbind(Specificity.CIF.under.loopmtry,
Sptemp)
  num.matrix.CIF.under.loopmtry <- cbind(num.matrix.CIF.under.loopmtry,2+i)
  i=i+1
}

MATRIX.CIF.under.loopmtry<- data.frame(
  "mtry"=as.numeric(num.matrix.CIF.under.loopmtry),
  "Accuracy"=as.numeric(Accuracy.CIF.under.loopmtry),
  "Sensitivity" = as.numeric(Sensitivity.CIF.under.loopmtry),
  "Specificity" = as.numeric(Specificity.CIF.under.loopmtry),
  "OOB error" = as.numeric(OOB.CIF.under.loopmtry)
)

MATRIX.CIF.under.loopmtry

windows(title="CIF undersampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.CIF.under.loopmtry, y=Accuracy.CIF.under.loopmtry,
type="l", col="blue", ylab="Accuracy", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.under.loopmtry, y=Sensitivity.CIF.under.loopmtry,
type="l", col="blue", ylab="Sensitivity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.under.loopmtry, y=Specificity.CIF.under.loopmtry,
type="l", col="blue", ylab="Specificity", xlab="mtry", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.under.loopmtry, y=OOB.CIF.under.loopmtry, type="l",
col="blue", ylab="OOB", xlab="mtry", ylim=c(0, 30))




#Set mtry
mtry.CIF.under.opt <-
MATRIX.CIF.under.loopmtry$mtry[min(which(MATRIX.CIF.under.loopmtry$OOB.error
== min(MATRIX.CIF.under.loopmtry$OOB.error)))]




#Evaluate ntree
i=1
j=1
#
set.seed(8159780)
CIF.under.loopntree<- cforest(factor(undersample_model$fish.pres) ~.,
data=undersample_model,
                              control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                     mincriterion = 0,
saveinfo = TRUE),
                              ytrafo = NULL, scores = NULL, ntree = i,
                              perturb = list(replace = TRUE), #fraction =
1#
                              applyfun = NULL,
                              cores = NULL, mtry = mtry.CIF.under.opt,
trace = TRUE)

pred.CIF.under.loopntree<-predict(CIF.under.loopntree)

OOB.CIF.under.loopntree <-  100 *
sum(abs(abs(as.numeric(pred.CIF.under.loopntree) - 1) -
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)
Accuracy.CIF.under.loopntree <-confusionMatrix(pred.CIF.under.loopntree,
truth.under, positive='1')$overall[1]
Sensitivity.CIF.under.loopntree <-
confusionMatrix(pred.CIF.under.loopntree, truth.under, positive='1')
$byClass[1]
Specificity.CIF.under.loopntree <-
confusionMatrix(pred.CIF.under.loopntree, truth.under, positive='1')
$byClass[2]
num.matrix.CIF.under.loopntree <- i
#

while (i<ntree_initial) {
  if(i<100)
  {
    j=1+j
    i=i+1
    set.seed(8159780)
    CIF.under.loopntree<- cforest(factor(undersample_model$fish.pres) ~.,
data=undersample_model,
                                  control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                         mincriterion = 0,
saveinfo = TRUE),
                                  ytrafo = NULL, scores = NULL, ntree = i,
                                  perturb = list(replace = TRUE), #fraction
= 1#
                                  applyfun = NULL,
                                  cores = NULL, mtry = mtry.CIF.under.opt,
trace = TRUE)


    pred.CIF.under.loopntree<-predict(CIF.under.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.under.loopntree) - 1) -
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)
    Acctemp=confusionMatrix(pred.CIF.under.loopntree, truth.under,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.under.loopntree, truth.under,
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.under.loopntree, truth.under,
positive='1')$byClass[2]

    OOB.CIF.under.loopntree <- cbind(OOB.CIF.under.loopntree, Oobtemp)
    Accuracy.CIF.under.loopntree <- cbind(Accuracy.CIF.under.loopntree,
Acctemp)
```

```
    Sensitivity.CIF.under.loopntree <- 
cbind(Sensitivity.CIF.under.loopntree, Sstemp)
    Specificity.CIF.under.loopntree <- 
cbind(Specificity.CIF.under.loopntree, Sptemp)
    num.matrix.CIF.under.loopntree <- 
cbind(num.matrix.CIF.under.loopntree,i)

    print(i)
  }

  if(i>=100 && i<500)
  {
    j=1+j
    i=i+10
    set.seed(8159780)
    CIF.under.loopntree<- cforest(factor(undersample_model$fish.pres) ~., 
data=undersample_model,
                                  control = ctree_control(teststat = "max", 
testtype = "Teststatistic",
                                                        mincriterion = 0, 
saveinfo = TRUE),
                                  ytrafo = NULL, scores = NULL, ntree = i,
                                  perturb = list(replace = TRUE), #fraction 
= 1#
                                  applyfun = NULL,
                                  cores = NULL, mtry = mtry.CIF.under.opt, 
trace = TRUE)


    pred.CIF.under.loopntree<-predict(CIF.under.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.under.loopntree) - 1) - 
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)
    Acctemp=confusionMatrix(pred.CIF.under.loopntree, truth.under, 
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.under.loopntree, truth.under, 
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.under.loopntree, truth.under, 
positive='1')$byClass[2]

    OOB.CIF.under.loopntree <- cbind(OOB.CIF.under.loopntree, Oobtemp)
    Accuracy.CIF.under.loopntree <- cbind(Accuracy.CIF.under.loopntree, 
Acctemp)
    Sensitivity.CIF.under.loopntree <- 
cbind(Sensitivity.CIF.under.loopntree, Sstemp)
    Specificity.CIF.under.loopntree <- 
cbind(Specificity.CIF.under.loopntree, Sptemp)
    num.matrix.CIF.under.loopntree <- 
cbind(num.matrix.CIF.under.loopntree,i)

    print(i)
  }

  if(i>=500 && i<ntree_initial)
  {
    j=1+j
    i=i+20
    set.seed(8159780)
    CIF.under.loopntree<- cforest(factor(undersample_model$fish.pres) ~., 
data=undersample_model,
                                  control = ctree_control(teststat = "max",
testtype = "Teststatistic",
                                                        mincriterion = 0,
saveinfo = TRUE),
                                  ytrafo = NULL, scores = NULL, ntree = i,
                                  perturb = list(replace = TRUE), #fraction
= 1#
                                  applyfun = NULL,
                                  cores = NULL, mtry = mtry.CIF.under.opt,
trace = TRUE)


    pred.CIF.under.loopntree<-predict(CIF.under.loopntree)
    Oobtemp= 100 * sum(abs(abs(as.numeric(pred.CIF.under.loopntree) - 1) -
abs(as.numeric(undersample_model$fish.pres)-1)))/
length(undersample_model$fish.pres)
    Acctemp=confusionMatrix(pred.CIF.under.loopntree, truth.under,
positive='1')$overall[1]
    Sstemp=confusionMatrix(pred.CIF.under.loopntree, truth.under,
positive='1')$byClass[1]
    Sptemp=confusionMatrix(pred.CIF.under.loopntree, truth.under,
positive='1')$byClass[2]

    OOB.CIF.under.loopntree <- cbind(OOB.CIF.under.loopntree, Oobtemp)
    Accuracy.CIF.under.loopntree <- cbind(Accuracy.CIF.under.loopntree,
Acctemp)
    Sensitivity.CIF.under.loopntree <-
cbind(Sensitivity.CIF.under.loopntree, Sstemp)
    Specificity.CIF.under.loopntree <-
cbind(Specificity.CIF.under.loopntree, Sptemp)
    num.matrix.CIF.under.loopntree <-
cbind(num.matrix.CIF.under.loopntree,i)

    print(i)
  }
}


MATRIX.CIF.under.loopntree<- data.frame(
  "ntree"=as.numeric(num.matrix.CIF.under.loopntree),
  "Accuracy"=as.numeric(Accuracy.CIF.under.loopntree),
  "Sensitivity" = as.numeric(Sensitivity.CIF.under.loopntree),
  "Specificity" = as.numeric(Specificity.CIF.under.loopntree),
  "OOB error" = as.numeric(OOB.CIF.under.loopntree)
)

MATRIX.CIF.under.loopntree

windows(title="CIF undersampling")
par(mfrow = c(2, 2))
plot(x=num.matrix.CIF.under.loopntree, y=Accuracy.CIF.under.loopntree,
type="l", col="blue", ylab="Accuracy", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.under.loopntree, y=Sensitivity.CIF.under.loopntree,
type="l", col="blue", ylab="Sensitivity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.under.loopntree, y=Specificity.CIF.under.loopntree,
type="l", col="blue", ylab="Specificity", xlab="ntree", ylim=c(0.6, 1))
plot(x=num.matrix.CIF.under.loopntree, y=OOB.CIF.under.loopntree, type="l",
col="blue", ylab="OOB", xlab="ntree", ylim=c(0, 30))
```

```
#set ntree
ntree.CIF.under <-
MATRIX.CIF.under.loopntree$ntree[max(which(MATRIX.CIF.under.loopntree$OOB.error
!= MATRIX.CIF.under.loopntree$OOB.error[j]))]
ntree.CIF.under.opt <-
ceiling((2*(MATRIX.CIF.under.loopntree$ntree[max(which(MATRIX.CIF.under.loopntree$OO
!= MATRIX.CIF.under.loopntree$OOB.error[j]))]))/100)*100



#final model
set.seed(8159780)
CIF.under.opt <- cforest(factor(undersample_model$fish.pres) ~.,
data=undersample_model,
                         control = ctree_control(teststat = "max", testtype
= "Teststatistic",
                                                 mincriterion = 0, saveinfo
= TRUE),
                         ytrafo = NULL, scores = NULL, ntree =
ntree.CIF.under.opt,
                         perturb = list(replace = TRUE), #fraction = 1#
                         applyfun = NULL,
                         cores = NULL, mtry = mtry.CIF.under.opt, trace =
TRUE)


CM.CIF.under.opt <- confusionMatrix(predict(CIF.under.opt), truth.under,
positive='1')
CM.CIF.under.opt
windows()
plot(predict(CIF.under.opt),truth.under, main = "Confusion matrix:
Optimized CIF undersampling")



####################################################
## SAVE RESULTS: EXCEL ##

EXCEL.wb <- createWorkbook()

model.names=cbind("RF unbalanzed", "RF oversampling", "RF undersampling",
"CIF unbalanzed", "CIF oversampling", "CIF undersampling")

#data inicial
addWorksheet(EXCEL.wb, "Initial_data")
writeData(EXCEL.wb, "Initial_data", iniz.data)

addWorksheet(EXCEL.wb, "Unbalanzed_data")
writeData(EXCEL.wb, "Unbalanzed_data", after.boruta.data)

addWorksheet(EXCEL.wb, "Oversampling_data")
writeData(EXCEL.wb, "Oversampling_data", oversample_model)

addWorksheet(EXCEL.wb, "Undersampling_data")
writeData(EXCEL.wb, "Undersampling_data", undersample_model)


#importancia variables
```

```
addWorksheet(EXCEL.wb, "Importance_RF_unbalanzed")
frame.Imp.RF.normal <- data.frame(
  "names"=bar.RF.normal[,1],
  "importance"=as.numeric(bar.RF.normal[,2]))
writeData(EXCEL.wb, "Importance_RF_unbalanzed", frame.Imp.RF.normal)

addWorksheet(EXCEL.wb, "Importance_RF_oversampling")
frame.Imp.RF.over <- data.frame(
  "names"=bar.RF.over[,1],
  "importance"=as.numeric(bar.RF.over[,2]))
writeData(EXCEL.wb, "Importance_RF_oversampling", frame.Imp.RF.over)

addWorksheet(EXCEL.wb, "Importance_RF_undersampling")
frame.Imp.RF.under <- data.frame(
  "names"=bar.RF.under[,1],
  "importance"=as.numeric(bar.RF.under[,2]))
writeData(EXCEL.wb, "Importance_RF_undersampling", frame.Imp.RF.under)

addWorksheet(EXCEL.wb, "Importance_CIF_unbalanzed")
frame.Imp.CIF.normal <- data.frame(
  "names"=bar.CIF.normal[,1],
  "importance"=as.numeric(bar.CIF.normal[,2]))
writeData(EXCEL.wb, "Importance_CIF_unbalanzed", frame.Imp.CIF.normal)

addWorksheet(EXCEL.wb, "Importance_CIF_oversampling")
frame.Imp.CIF.over <- data.frame(
  "names"=bar.CIF.over[,1],
  "importance"=as.numeric(bar.CIF.over[,2]))
writeData(EXCEL.wb, "Importance_CIF_oversampling", frame.Imp.CIF.over)

addWorksheet(EXCEL.wb, "Importance_CIF_undersampling")
frame.Imp.CIF.under <- data.frame(
  "names"=bar.CIF.under[,1],
  "importance"=as.numeric(bar.CIF.under[,2]))
writeData(EXCEL.wb, "Importance_CIF_undersampling", frame.Imp.CIF.under)


#OOB error antes de la optimización
addWorksheet(EXCEL.wb, "OOB_before_optimizing")
frame.oob.initial <- data.frame(
  "names"=t(model.names),
  "OOB_initial"=t(t(as.numeric(OOBerror)))
)
writeData(EXCEL.wb, "OOB_before_optimizing", frame.oob.initial)


#Optimización de modelos
addWorksheet(EXCEL.wb, "RF_unbalanzed_mtry")
writeData(EXCEL.wb, "RF_unbalanzed_mtry", MATRIX.RF.normal.loopmtry)
addWorksheet(EXCEL.wb, "RF_unbalanzed_ntree")
writeData(EXCEL.wb, "RF_unbalanzed_ntree", MATRIX.RF.normal.loopntree)

addWorksheet(EXCEL.wb, "RF_oversampling_mtry")
writeData(EXCEL.wb, "RF_oversampling_mtry", MATRIX.RF.over.loopmtry)
addWorksheet(EXCEL.wb, "RF_oversampling_ntree")
writeData(EXCEL.wb, "RF_oversampling_ntree", MATRIX.RF.over.loopntree)

addWorksheet(EXCEL.wb, "RF_undersampling_mtry")
writeData(EXCEL.wb, "RF_undersampling_mtry", MATRIX.RF.under.loopmtry)
addWorksheet(EXCEL.wb, "RF_undersampling_ntree")
```

```
writeData(EXCEL.wb, "RF_undersampling_ntree", MATRIX.RF.under.loopntree)

addWorksheet(EXCEL.wb, "CIF_unbalanzed_mtry")
writeData(EXCEL.wb, "CIF_unbalanzed_mtry", MATRIX.CIF.normal.loopmtry)
addWorksheet(EXCEL.wb, "CIF_unbalanzed_ntree")
writeData(EXCEL.wb, "CIF_unbalanzed_ntree", MATRIX.CIF.normal.loopntree)

addWorksheet(EXCEL.wb, "CIF_oversampling_mtry")
writeData(EXCEL.wb, "CIF_oversampling_mtry", MATRIX.CIF.over.loopmtry)
addWorksheet(EXCEL.wb, "CIF_oversampling_ntree")
writeData(EXCEL.wb, "CIF_oversampling_ntree", MATRIX.CIF.over.loopntree)

addWorksheet(EXCEL.wb, "CIF_undersampling_mtry")
writeData(EXCEL.wb, "CIF_undersampling_mtry", MATRIX.CIF.under.loopmtry)
addWorksheet(EXCEL.wb, "CIF_undersampling_ntree")
writeData(EXCEL.wb, "CIF_undersampling_ntree", MATRIX.CIF.under.loopntree)


#mtry de cada modelo
addWorksheet(EXCEL.wb, "mtry_optimized_models")
frame.mtry.opt <- data.frame(
  "names"=t(model.names),
  "mtry"=t(cbind(mtry.RF.normal.opt, mtry.RF.over.opt, mtry.RF.under.opt,
mtry.CIF.normal.opt, mtry.CIF.over.opt, mtry.CIF.under.opt))
  )
writeData(EXCEL.wb, "mtry_optimized_models", frame.mtry.opt)


#ntree de cada modelo
addWorksheet(EXCEL.wb, "ntree_optimized_models")
frame.ntree.opt <- data.frame(
  "names"=t(model.names),
  "mtry"=t(cbind(ntree.RF.normal.opt, ntree.RF.over.opt,
ntree.RF.under.opt, ntree.CIF.normal.opt, ntree.CIF.over.opt,
ntree.CIF.under.opt))
)
writeData(EXCEL.wb, "ntree_optimized_models", frame.ntree.opt)


#OOB final de cada modelo
addWorksheet(EXCEL.wb, "OOB_final")
frame.oob.final <- data.frame(
  "names"=t(model.names),
  "OOB_final"=t(cbind(tail(MATRIX.RF.normal.loopntree$OOB.error, n = 1),
tail(MATRIX.RF.over.loopntree$OOB.error, n = 1),
tail(MATRIX.RF.under.loopntree$OOB.error, n = 1),
tail(MATRIX.CIF.normal.loopntree$OOB.error, n = 1),
tail(MATRIX.CIF.over.loopntree$OOB.error, n = 1),
tail(MATRIX.CIF.under.loopntree$OOB.error, n = 1)))
)
writeData(EXCEL.wb, "OOB_final", frame.oob.final)


saveWorkbook(EXCEL.wb, file = "Data_Results.xlsx")
```