DATA ANALYSIS AND MACHINE LEARNING
FYS-STK4155

Project 3

Marco POZZOLI
Lila CASSAN

December 17, 2023

Abstract

The aim of this study was to find an efficient way of modeling the Forest Cover Types dataset, often used in Machine Learning. Three main algorithms were used: a Feed-Forward Neural Network , a Random Forest and XGboost. The XGboosting gave us the best accuracy (1.00) and was the fastest algorithm. The Random Forest gave us an efficient model but it was computationally expensive to find the most adequate number of tree. The Feed-Forward Neural Network (FFNN) gave us a satisfying result (0.90), but needs a tuning of much parameters and performs worse than the others algorithms.

1 Introduction

Forests play an important role in climate modeling due to their impact on carbon sequestration, oxygen release and biodiversity, and the use of machine learning algorithms helps to determine their structure and composition from data that can be collected more or less automatically, from ground observations or remote sensing data.

The Forest Cover Type dataset consists of ground observations made on forest plots. Our aim was to find the model fitting the best the dataset, in order to make accurate predictions. Modeling it allows us to develop an approach for other datasets to see what works best. Selecting this dataset also allows for working with a commonly used training dataset, which has been the subject of competitions, facilitating result comparisons.

In this study, we used three algorithms: Random Forest, Feed-Forward Neural Network, and XGboost whose parameters we adjusted.

This report is organized into three sections: Methods, Results, and Discussion. In the Methods section, We describe the techniques employed in our modeling of the Forest Cover Type dataset. The Results section presents the findings of our analysis, using different metrics. Finally, the Discussion section provides a comparative analysis of the two algorithms employed, as well as with previous studies.

2 Methods

2.1 Dataset

For this project, we chose the "Forest Cover Type Dataset" [1]. It contains data from four areas of the Roosevelt National Forest in Colorado, tree observations in particular. All the observations refer to 30x30 meter sections of the forest. The dataset is quite big as it contains more than 500'000 measurements.
The original owners are Jock A. Blackard, Dr. Denis J. Dean, and Dr. Charles W. Anderson of the Remote Sensing and GIS Program at Colorado State University.
Our use for this dataset is to find which is the best model to predict what type of coverage a certain area has based on the other features.
The dataset contains features such as elevation, aspect, slope, horizontal distance to hydrology, vertical distance to hydrology, horizontal distance to roadways, horizontal distance to fire point, hillshade (at different times of the day), different types (4) of the wilderness area and of soil type (40).

2.2 Metrics

Here we introduce the metrics that we used to evaluate the quality and performance of the models.

2.2.1 Accuracy

The accuracy score is a simple metric given by the number of correct guesses (t_i) divided by the number of total guesses (n):

$$Accuracy = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \tag{1}$$

Where:

$$I = \begin{cases} 1 \text{ if the condition is met} \\ 0 \text{ if the condition is not met} \end{cases} . \tag{2}$$

2.2.2 ROC curve

The ROC curve (*Receiver Operating Characteristic Curve*) is a graph that represents the performance of a classification model at different thresholds.

This curve takes into account two parameters: TPR (True Positive Rate) on the y-axis and FPR (False Positive Rate) on the x-axis.
TPR is calculated on the amount of positive results among all positive samples available during the test. FPR is instead defined as how many incorrect positive results occur among all negative samples available during the test.

$$TPR = \frac{TP}{TP + FN} \tag{3}$$

$$FPR = \frac{FP}{FP + FN} \tag{4}$$

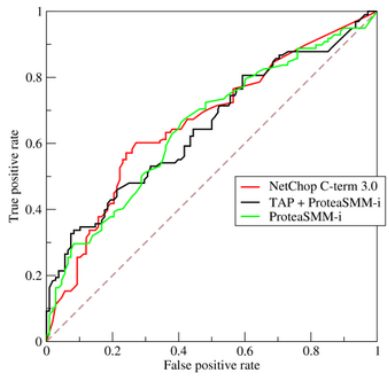

Figure 1: Example of a ROC curve. [5]

2.3 Feed-Forward Neural Network

Describe the methods and algorithms We use a Feed-Forward Neural Network, as described in [4]. This architecture of Neural Networks corresponds to hidden layers with information going forward, and no feedback. The Neural Network allows us to obtain a model by adjusting weights in biases with a backpropagation pass, in an iterative process (see [4] for further details).

In a first time, we tried to adjust our code for Project 2 for modelling the Forest Cover Type Dataset. The result diverged so we decided to use the library Tensorflow and the API Keras. After loading and reshaping our data, we create our neurons layers. We choose one hidden layer with 64 neurons, and an output layer consisting in 7 neurons. The number 7 correspond to the number of classes in the dataset, we have then a probability for each class. We choose an optimizer : ADAM with a learning rate $\gamma = 0.0001$. Then,

we train the model, using our data. Finally, we can predict classes for given features, and compute metrics.

```
model = keras.Sequential([
    layers.Dense(64, activation='relu',
    input_shape=[X_train.shape[1]]),
    layers.Dense(64, activation='relu'),
    layers.Dense(7, activation='softmax')
])

optimizer = Adam(learning_rate=10e-4)

model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'],
)
```

2.4 Decision Trees

Decision trees are the key component and basic algorithm of random forests. The main idea behind them is finding features that are the most important in identifying the correct target and splitting the dataset along the values of the aforementioned features.

As the name suggests, decision trees are divided into *root node*, *interior nodes*, and the *final leaf nodes/leaves*. They're connected by *branches*.

A decision tree is grown by performing binary splitting, via an algorithm that tries splitting the dataset at different feature split points and tested with a cost function. The splitting is then performed on the resulting subtrees. This algorithm is *greedy*, as the optimal split is chosen at the current step, instead of anticipating future steps and selecting a split that could result in a more effective tree later on.

In constructing a classification tree, the Gini index or entropy are commonly employed to assess the effectiveness of a specific split. This is because these methods are more responsive to the purity of a node compared to the classification error rate.

The Gini index gives us an idea of how mixed the classes are after the split.

It's is defined as:

$$g = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{5}$$

Entropy measures how much information we would get by knowing the value of a feature.
It's is defined as:

$$s = -\sum_{k=1}^{K} p_{mk} \log p_{mk} \tag{6}$$

Where k = 1,...,K is the number of observations.
p_{mk} represents the number of observations of a class k in a region R_m in a region N_m observations. It's defined as:

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R,} I(y_i = k) \tag{7}$$

We can also visualize a tree as:

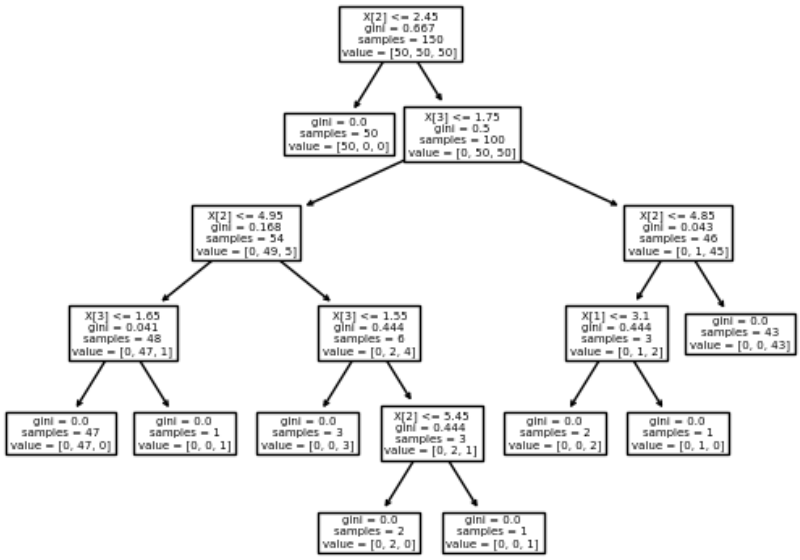

Figure 2: Example of a tree. [2]

2.5 Random Forest

With a single decision tree, we often end up overfitting our training data, which implies a high variance. To reduce variance we introduce the *ensemble methods*. One of these is $random\ forests$.
In random forests, similarly to bagging, we build several decision trees on training samples obtained with bootstrapping. In the process of building the trees every time that a split is considered, a random sample of m predictors is chosen as candidates for splitting from the full set of p predictors. This way trees are different from each other and have a low correlation.
In classification problems, every tree votes for the most likely class, and the class with the most votes is taken as the final prediction of the model.
Random forests are effective on both classification and regression problems. Another important strength of random forests is that they are effective on large datasets with many variables, on the downside they're usually computationally intensive and may take a long time to use.

The code is fairly easy to implement thanks to Sci-kit Learn, a library that provides very useful tools for machine learning coding. After importing the libraries we load the data from our chosen dataset and split it into features (X) and target (Y), we then split it into training and testing. We scale it and we then create the random forest using sci-kit Learn, fit the data, and evaluate the model with the metrics that we previously introduced. The best parameters are found by performing a grid search.

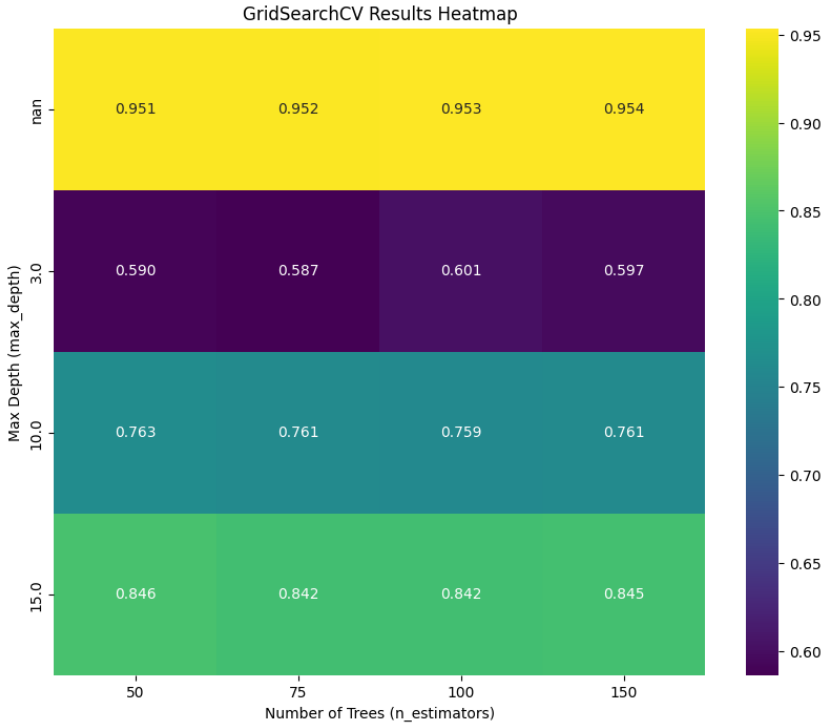

Figure 3: Results of the grid search for the optimal depth and number of trees.

The optimal parameters turn out to be depth = *none* and n_estimators = 150.

2.6 XGBoost

XGboost, which stands for *extreme gradient boosting*, is a gradient boosting library designed to make learning quicker and more precise. Since its invention and proposal, it has won several machine learning competitions. We code it by using its library *xgboost*. With XGboost we also get a metric that gives us the importance of each feature in classifying data.

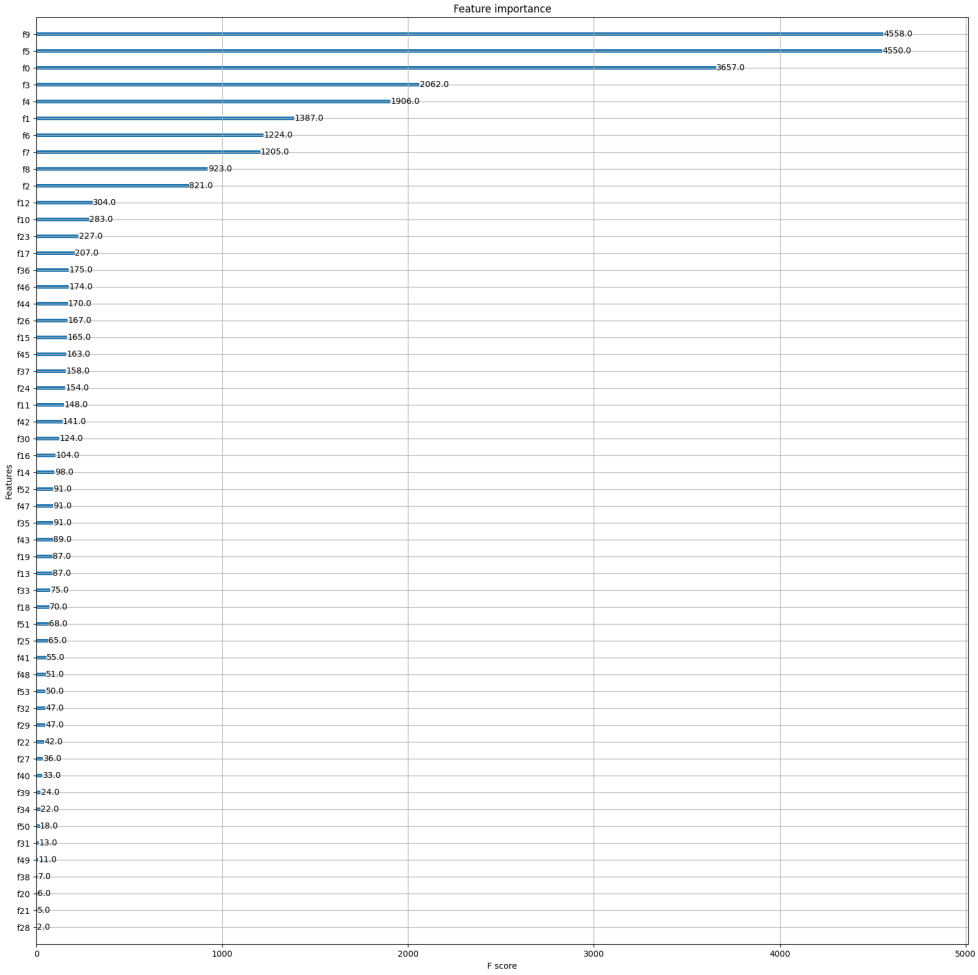

Figure 4: Importance of each feature given by XGboost.

The most important features are by far horizontal distance to fire points (f9), horizontal distance to roadways (f5) and elevation (f0). The least important features for the prediction are, in general, soil types.

3 Results

3.1 Neural Network

Using 30 epochs and one hidden layer, the ADAM optimizer with a learning rate $\gamma = 0.001$, the model accuracy increases really fast until approximately 10 epochs and then stabilizes. The model loss decreases significantly until 5 epochs and then stabilizes.

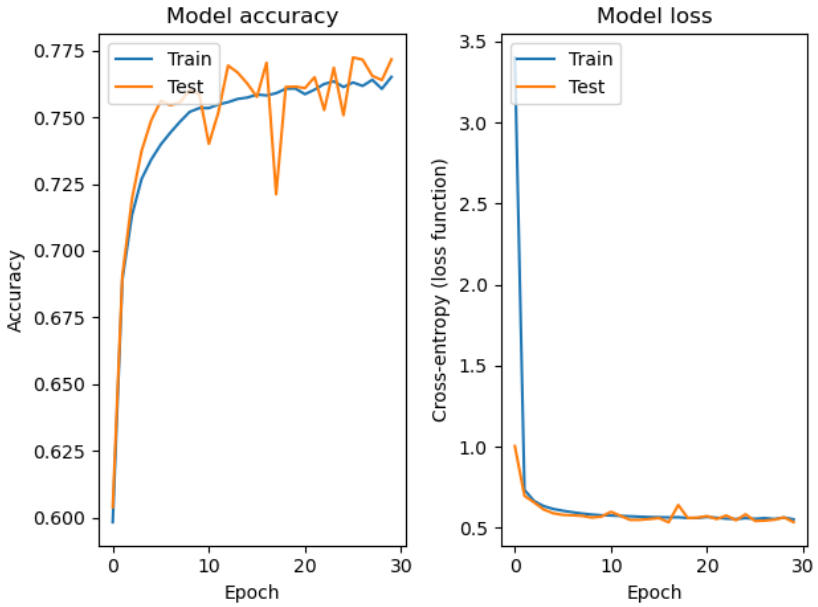

Figure 5: Accuracy and Lost for a Feed Forward Neural Network with 1 hidden layer.

Running the following command:

```
#fit the model
model_fitting = model.fit(
    X_train, y_train,
    validation_data = (X_test, y_test),
    verbose=1, epochs=30)
```

took approximately 5 minutes and 30 seconds, even with a limited number of epochs (30). This limit does not allow us to try as many parameters

combinations as we want.

After computing the ROC curves for our Feed-Forward Neural Network (Figure 6, we obtain AUC (Area Under the Curve) values between 0.89 (class 2) and 1.00 (class 4) for each class against the other, which is satisfying [3]. The AUC value for micro-average of the ROC curve is 0.97, which means that our model can discriminate well between classes. The AUC value for micro-average of the ROC curve is 0.96, which means an adequate global performance of the model.

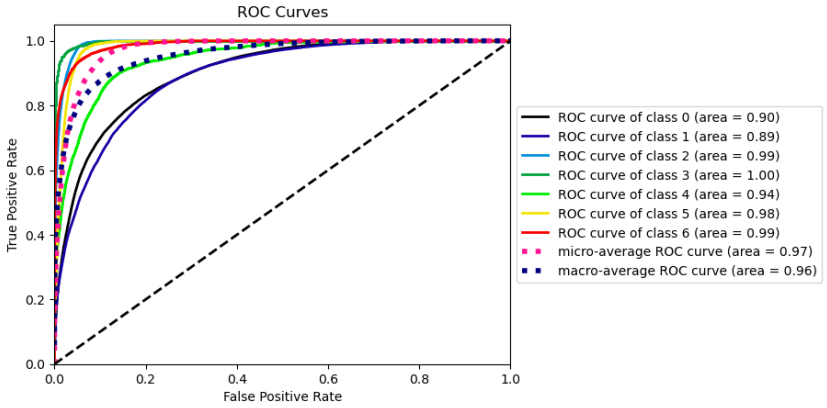

Figure 6: ROC Curve for a Feed Forward Neural Network with 2 hidden layers.

On the diagonal of the matrix, we want to have values near to 1. However, the label 1 was predicted with a probability of 0.74 where the true label was 4 and the correct label was only assigned with a probability 0.17. On the other side, the correct label was assigned with a probability of 0.89 for class 1. This means that our model can easily distinguish what can belong to class 1, but this class is predicted to too much inputs.

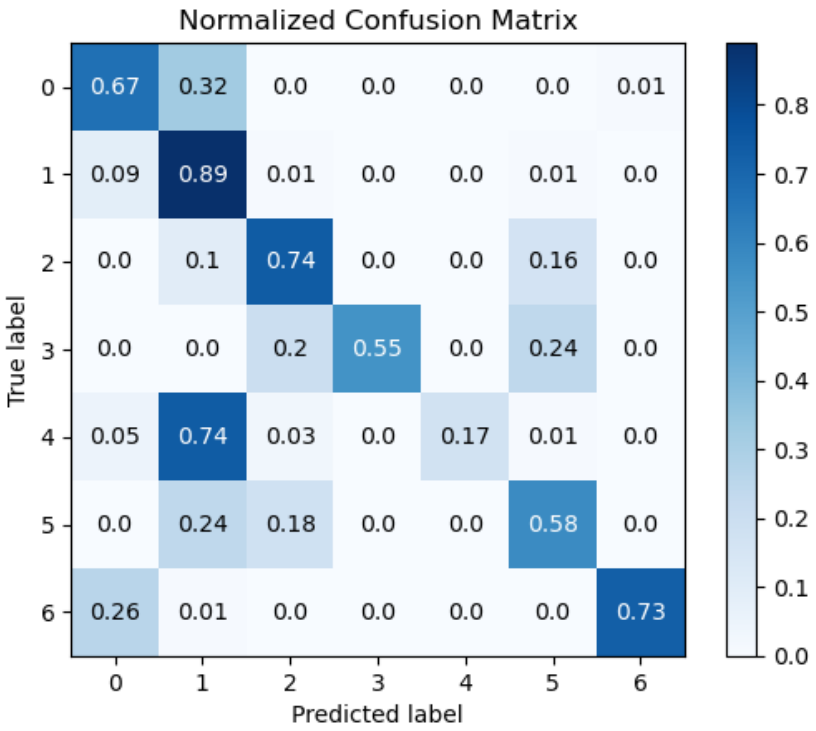

Figure 7: Confusion Matrix a Feed Forward Neural Network with 2 hidden layers.

We then ran the code multiple times to try to find the best parameters, and with 3 hidden layers of 120 neurons, the ADAM optimizer with a learning rate of $\gamma = 0.00001$ gave us a satisfying layer with an accuracy of 0.90 (Figure 8), in 15 minutes. The number of epochs appears satisfying because the accuracy was in average still improving but with a small step.

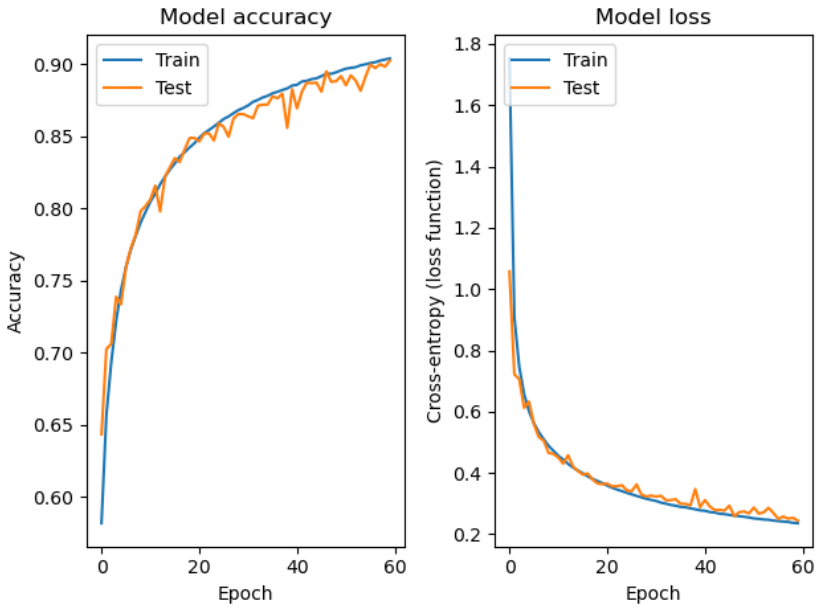

Figure 8: Accuracy and Lost for a Feed Forward Neural Network with 3 hidden layers of 120 neurons.

The area under the ROC curves was better also, and between 0.97 and 1 for all classes (Figure 9)

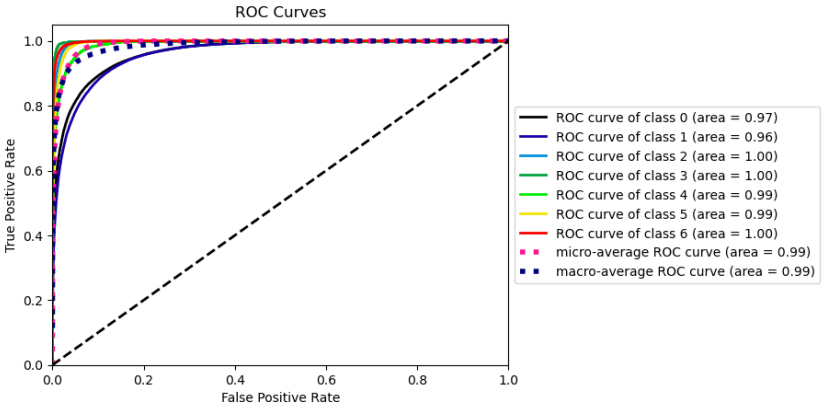

Figure 9: ROC curves for a Feed Forward Neural Network with 3 hidden layers of 120 neurons.

The problem of confusion between classes 1 and 4 with our previous model was solved, and the value on the diagonal for class 4 is now better than 0.5, which means that our model is better than a random choice. However, it can still be improved because this value is much lower than for the other classes (0.67).

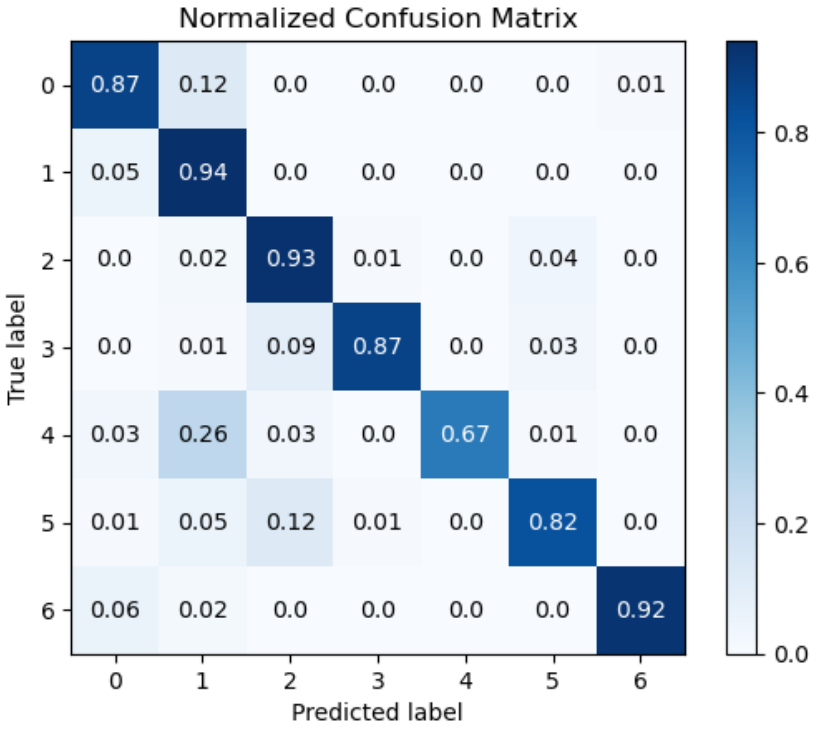

Figure 10: Confusion matrix for a Feed Forward Neural Network with 3 hidden layers of 120 neurons.

3.2 Random Forest

Running the code section where the random forest is trained:

```
#Random Forest fitting
#grid search's best params
best_params = grid_search.best_params_

optimized_rf = RandomForestClassifier(**best_params)
```

```
6
7 #we train the random forest model
8 optimized_rf.fit(X_train_scaled, y_train)
```

took around two hours. This is influenced by the size of the dataset and the computational power of the device on which the code is run.
After the data fitting, we take a look at the ROC curves.

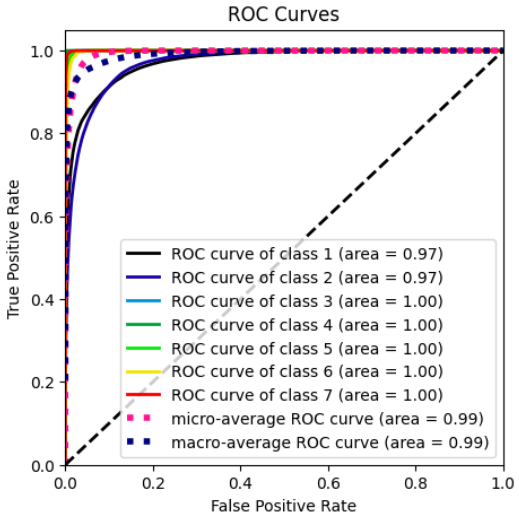

Figure 11: ROC curves of our random forest.

The curves are very good, as almost every single one of them has an area of 1. Classes one and two have a slightly inferior area but are still very close to 1. Thus the random forest is a very good model for our classification problem and it can differentiate effectively.
We also take a look at the confusion matrix.

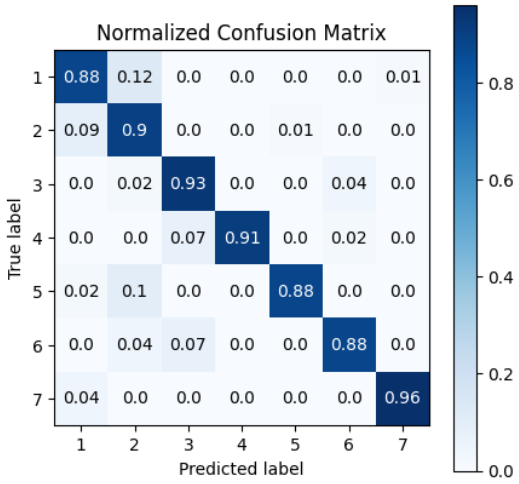

Figure 12: Confusion matrix of the random forest.

We can easily see how the Random forest sometimes fails to identify the right class. The predicted label is often mistaken for just one other true label, for example, the first class is sometimes mistaken almost only for the second.
The class that gets identified correctly more often is the seventh with an accuracy of 0.96, meanwhile, the hardest to predict correctly are the first, fifth, and sixth, with an equal accuracy of 0.88.

3.3 XGboost

The code for XGboost is:

```
1 #XGboost training
2 xg_clf = xgb.XGBClassifier()
3 xg_clf.fit(X_train_scaled,y_train)
```

and it takes xxxx minutes to train.
We record an accuracy of 1.00, which makes the model perfect for our problem, as it predicts correctly every single time.
We now take a look at the ROC curve.

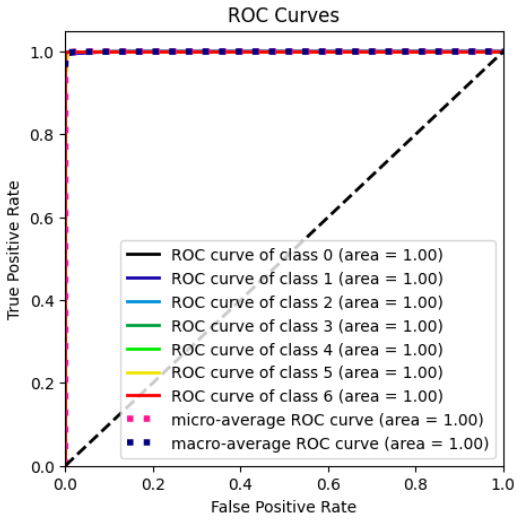

Figure 13: ROC curves of XGboost.

All the curves are perfect: they all have an area of 1. This was expected, as we saw with accuracy that the XGboost model can perfectly predict the right cover type from the given features.
The confusion matrix is just as predictable:

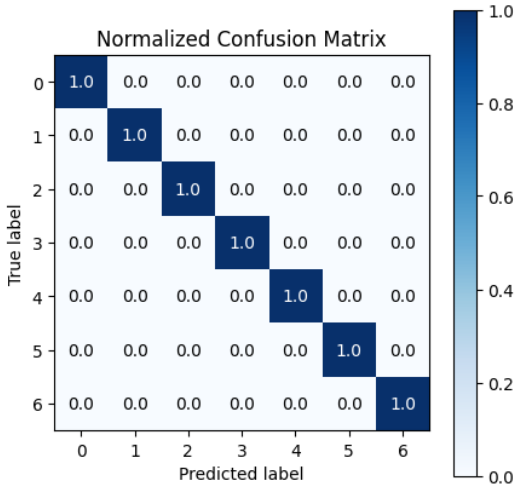

Figure 14: Confusion matrix of XGboost.

The matrix is a perfect diagonal.

4 Discussion

4.1 Comparison

We can now compare our results. We start by comparing the accuracy of the three methods.

Algorithm	Accuracy
Neural Network	0.904
Random Forest	0.955
XGboost	1.000

Table 1: Accuracy for Neural Network, Random Forest and XGboost classifying the Forest Cover Type Dataset

The best method is by far XGboosting (Table 4.1). It has perfect accuracy and never fails to identify the correct class. It's also very quick, taking only around 30 seconds to fit and predict.
The random forest is very accurate as well, even if it's not flawless, with an accuracy of 0.955. Results obtained with Feed-Forward Neural Networks are not as accurate at the latters. However, it can still be improved by tuning different parameters (learning rate, number of epochs...), or by trying different algorithms for the optimizer.

5 Conclusion

Our aim was to determine the best method for modelling the Forest Cover Type Dataset. Here, the XGBoosting give us the best results, and requires less time. It is possible to use different algorithms that can give us best results, but also to try different parameters to determine an efficient way to use Feed-Forward Neural Network.

6 Appendix

Code: see https://github.com/lilacandolle/Forest-Cover-Type

References

[1] Jock Blackard. Forest cover type dataset, 1998. https://www.kaggle.com/datasets/uciml/forest-cover-type-dataset.

[2] M. Hjorth-Jensen. Lecture notes, applied data analysis and machine learning, 2023. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week46.html.

[3] Jayawant N. Mandrekar. Receiver operating characteristic curve in diagnostic test assessment. *Journal of Thoracic Oncology*, 5(9):1315–1316, 2010.

[4] Marco Pozzoli and Lila Cassan. Project2, fys-stk4155, 2023.

[5] Wikipedia. Receiver operating characteristic, wikipedia., 2023. https://en.wikipedia.org/wiki/Receiver_operating_characteristic.