Machine Learning – Assignment 1

Report by Mohamed Amine DASSOULI

Table of contents:

1. Introduction

In this report, we are going to approach Supervised machine learning models, and apply them in order to solve two different problematics.

◆ The first subject is about Bank Customer Churn prediction, it is a binary classification problem where we aim to predict whether a customer is going to churn or not.

◆ The second topic is about evaluating the quality of a car offer based on its physical qualifications, it is a multiclass classification problem with four labels: unacceptable, acceptable, good and very good.

We are going to use five models for our analysis: Decision Trees, Artifical Neural Network, Boosting, SVM and KNN. We are going to detail each of them in further parts.

2. Models

2.1. Decision Trees

A decision tree consists of the root which further splits into decision nodes. Depending on the outcome of the branches, the next branch or leaf nodes are formed. There are a lot of parameters to take into consideration while creating a decision tree, like:

- **criterion:** the function to measure the quality of a split, It could be 'gini' or 'entropy'.
- **max_depth:** the maximum depth of a tree.
- **min_samples_split:** the minimum number of samples required to split an internal node.

2.2. Neural Networks

Neural networks are series of algorithms that endeavors to recognize underlying relationships in a set of data. The input goes through a serie of layers of different sizes, finishing by the output layer. Each layer has an activation function that helps capture interactions between the features, the most common ones are:

- **ReLU,** which activates the node only if its input value is positive, introducing some non-linearity.
- **sigmoid,** which transforms the input into a value between 0 and 1.

2.3. Boosting

The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor. Boosting algorithms combine multiple low accuracy(or weak) models to create a high accuracy(or strong) models. In our study, we are going to use Adaboost Classifier. The hyperparameters of the model are:

- **n_estimators:** the maximum number of estimators at which boosting is terminated.
- **learning_rate:** float defining the contribution of each classifier.

2.4. SVM

It is a discriminative classifier formally defined by a separating hyperplane. His main parameters are:

- **kernel:** mathematical transformation to apply on input, it can be {linear, polynomial, sigmoid, radial}.
- **gamma:** float defining how far the influence of a single training example reaches.
- **cost (C):** regularization parameter.
- degree and coef0 for polynomial kernel.

2.5. k-Nearest neighbors

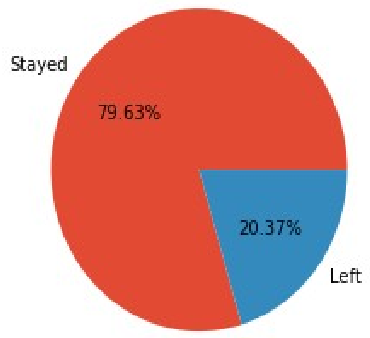
It is a classifier that studies feature similarity to predict the cluster a new point will fall into. The most important hyperparameter is **k**, which is the number of neighbors to take into account. Then, there is the **distance** used, which is by default the euclidean_distance.

3. First dataset: Bank Customer Churn

3.1. Approaching the dataset

The first dataset has a size of (10000, 13), there are 12 features and 10000 samples.

The label column is named **'Exited'**, it contains two values: '1' if the customer left the bank, and 0 if he stays. The figure below shows the distribution of the two classes:



We can see that **the positive class has a ratio of 20.37%** (2037 customers exited out of 10000), which means that the data is unbalanced.

This might lead to some domination of the majority class. The challenge of dealing with this imbalance is what makes this dataset interesting to study.

3.2. Preprocessing

Before creating our prediction models, we need to do some preprocessing on the data:

- Dropping the features that are not useful for the predictions, such as **"CustomerId"**, **"Surname"**. (We can always get back to the customer using the row number of the dataframe).
- Checking if there is any missing data. Hopefully, there isn't any in our dataset.
- Encoding the categorical features (**"Geography"** and **"Gender"**). We are going to do that as following:
  - If the feature has only two unique values, we are going to use a Label Encoder, and thus get a single column with two possible values '0' and '1'. This is applicable for "Gender" which can take only "Male" or "Female" as values.
  - Otherwise, if the feature takes multiple values (like "Geography" which has three values: 'France', 'Spain' and 'Germany'), we are going to encode it using a One Hot Encoder, and then we will get a number of columns equal to the number of values, with '1' in the column corresponding to the row's value, and '0' in the other columns.
- Scaling the the continuous features, such as **"CreditScore"**, **"EstimatedSalary"**, **"Balance"**.

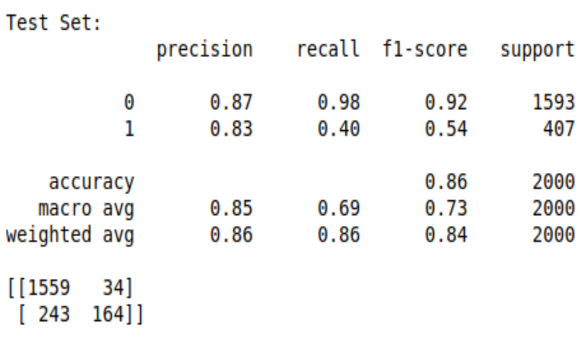
3.3. Creating training and testing sets

The data is now ready to use, and we can divide it into a training set and a testing set. We chose a ratio of **80% for the training set and 20% for the testing set**, thus we get 8000 rows for our training set and 2000 rows for our testing set. The choice of this ratio was motivated by the big size of the data: if we had a data with few examples, we might have chosen to take a higher ratio for the testing set in order to get more samples to test on.

Since the data is unbalanced, it is important to do it in a **stratified way**. The goal is to keep the same positive class ratio, in both training set and testing set, similar as it was in the original dataset.

3.4. Choosing the right metrics

The figure shows the prediction result of a Decision Tree on the Test Set:

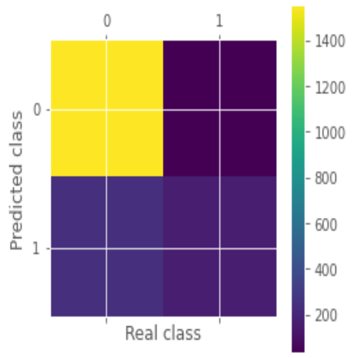


```
Test Set:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      1593
           1       0.83      0.40      0.54       407

    accuracy                           0.86      2000
   macro avg       0.85      0.69      0.73      2000
weighted avg       0.86      0.86      0.84      2000

[[1559   34]
 [ 243  164]]
```



- We got an **accuracy of 86%** ! This can be very satisfying for a lot of classification problems, but not in our case since the data is unbalanced. We can get a better idea of how the model worked in our testing set by looking at the precision and recall of the positive class.
- The **recall is equal to 40%**, meaning that the model has detected (predicted correctly) only 40% of the available 407 positive samples in the test set, and predicted all the other 60% of 407 as '0'.
- We have a **high precision (83%)** for the positive class. This means that most of the rows predicted as '1' by the model are correct.
- Having a high precision and a low recall means that the classifier mostly predicts '0', which means that **the positive class is dominated by the majority class** (negative class).
- In order to take the imbalance into account, we are going to use **balanced_accuracy**, which is defined as the average recall obtained on each class. In this example, its value is **69%**.

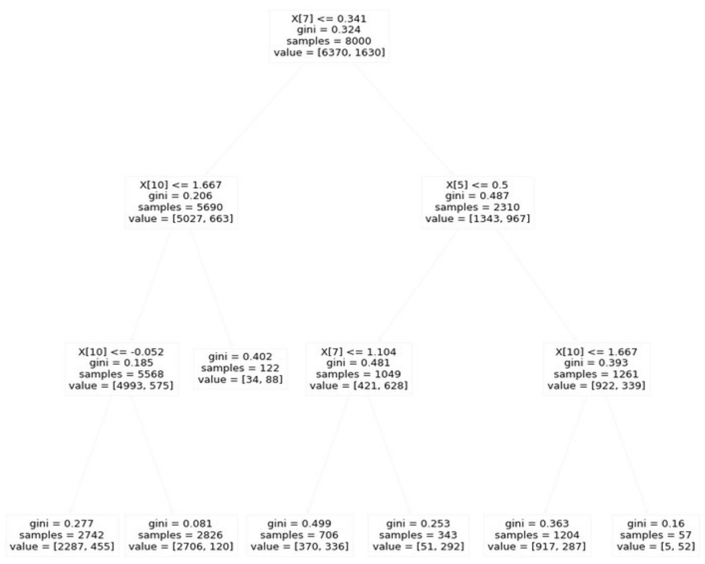
3.5. Experiments using supervised models

3.5.1. Decision Trees

a- Overview

Here is an overview of a decision tree, with depth=3:

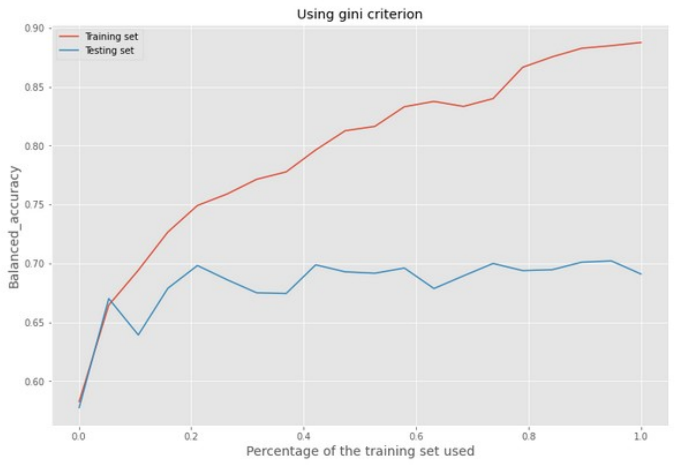
- ➢ There is a root, which is a starting point of all samples.
- ➢ On each leaf, there is a logical expression (condition) that decides if the sample would be directed into the left branch or the right branch, of the next level. For example the expression in the root is “X[7] <= 0.341”. All the samples satisfying this condition are redirected to one direction, and the others to the opposite direction (we can see that 5690 samples got redirected to the left and 2310 to the right).

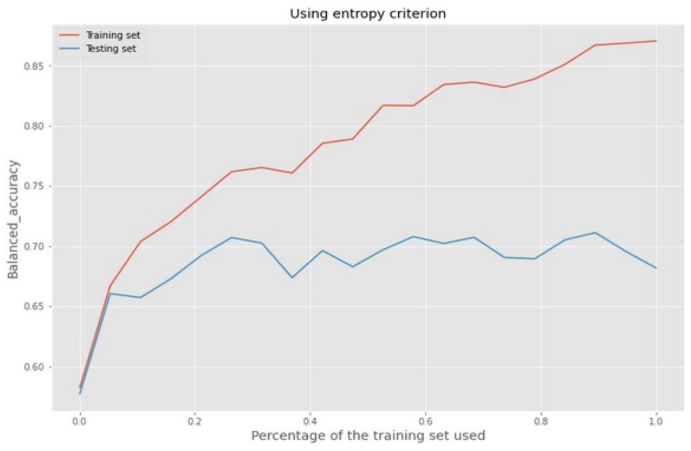


X[7] <= 0.341 gini = 0.324 samples = 8000 value = [6370, 1630]
X[10] <= 1.667 gini = 0.206 samples = 5690 value = [5027, 663]
X[5] <= 0.5 gini = 0.487 samples = 2310 value = [1343, 967]
X[10] <= -0.052 gini = 0.185 samples = 5568 value = [4993, 575]
gini = 0.402 samples = 122 value = [34, 88]
X[7] <= 1.104 gini = 0.481 samples = 1049 value = [421, 628]
X[10] <= 1.667 gini = 0.393 samples = 1261 value = [922, 339]
gini = 0.277 samples = 2742 value = [2287, 455]
gini = 0.081 samples = 2826 value = [2706, 120]
gini = 0.499 samples = 706 value = [370, 336]
gini = 0.253 samples = 343 value = [51, 292]
gini = 0.363 samples = 1204 value = [917, 287]
gini = 0.16 samples = 57 value = [5, 52]

b- Comparing ‘gini’ and ‘entropy’ criterions

Using default values for max_depth and min_samples_split, let’s see which criterion works best:





- ➢ The learning curves of the two criterions are quite similar, with **‘gini’ a bit better than ‘entropy’**.

c- Tuning hyperparameters

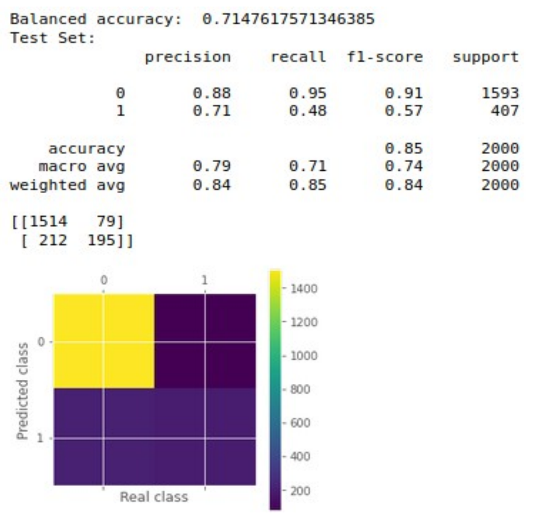
In order to find the best values for our hyperparameters (max_depth, min_samples_split), we are going to do a **GridSearch**:

- First, we are going to choose a range of values for every hyperparameter.
- Next, for each couple (max_depth_value, min_samples_split_value), we are going to compute the value of our metric (balanced_accuracy), and we are going to choose the couple corresponding to the highest value.
- Since we don’t have a validation set to use for tuning, we are going to use **Cross-validation**. This means that we are going to split the training set in k folds in a stratified way. Then, each time we are going to create the model, fit it on (k-1) folds and use the kth fold to compute our metric. We do this k times and then we consider the mean of the k values, as our metric value for the couple of hyperparameters used. Here are the results:

min_samples_split / max_depth	**2**	**10**	**20**	**50**	**100**	**200**
2	0.68	0.68	0.68	0.68	0.68	0.68
4	0.66	0.66	0.66	0.66	0.67	0.67
6	0.69	0.70	0.69	0.70	0.70	0.69
8	0.71	0.71	0.71	0.71	0.72	0.69
10	0.70	0.70	0.70	0.71	0.73	0.69
12	0.70	0.70	0.70	0.71	0.72	0.69
14	0.69	0.69	0.70	0.71	0.72	0.69

The highest value is 0.73, it corresponds to **max_depth=10 & min_samples_split=100,** so that's the couple we are going to use to create our model.

Here is the confusion matrix we get by predicting on the testing set:



```
Balanced accuracy:  0.7147617571346385
Test Set:
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      1593
           1       0.71      0.48      0.57       407

    accuracy                           0.85      2000
   macro avg       0.79      0.71      0.74      2000
weighted avg       0.84      0.85      0.84      2000

[[1514   79]
 [ 212  195]]
```

- ➢ We reached a **balanced accuracy of 71.48 %** !
- ➢ The recall for the positive class is equal to 48%, which means that 48% of the clients that churned would have been detected using the model.

d- Prevent overfitting

Decision Trees are prone to over-fitting, they tend to stick to the training set and thus perform bad on 'unseen' data.

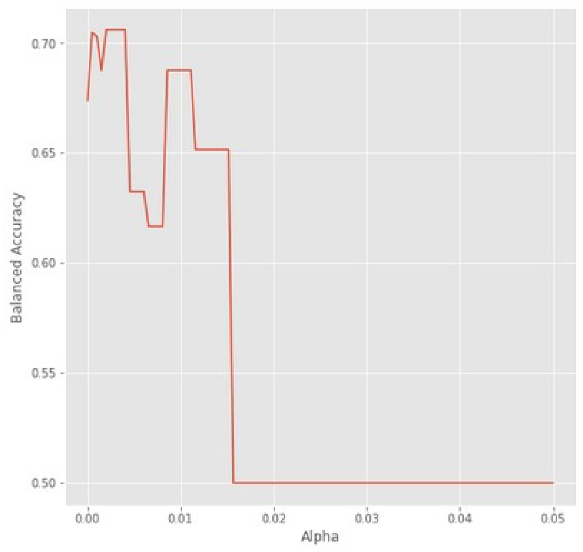
There are techniques to prevent that and make decision trees more generalized:

- ◆ **Setting constraints on the tree size** like we did before, by choosing a smaller value of max_depth, or a higher value for the minimum number of samples to split. This should be done carefully because if max_depth becomes very small, the decision tree is going to suffer from underfitting. That's why it's safer to do grid_search in order to choose appropriate values.
- ◆ **Pruning**, which is a technique to reduce the size of a decision tree after it is already created by removing sections of it.

One of the most famous technique to do pruning is **Minimal cost complexity,** which recursively finds the node with the "weakest link", until all the remaining nodes have an effective alpha below the threshold.

The curve below shows the value of Balanced Accuracy on the testing set depending on the ccp_alpha value.

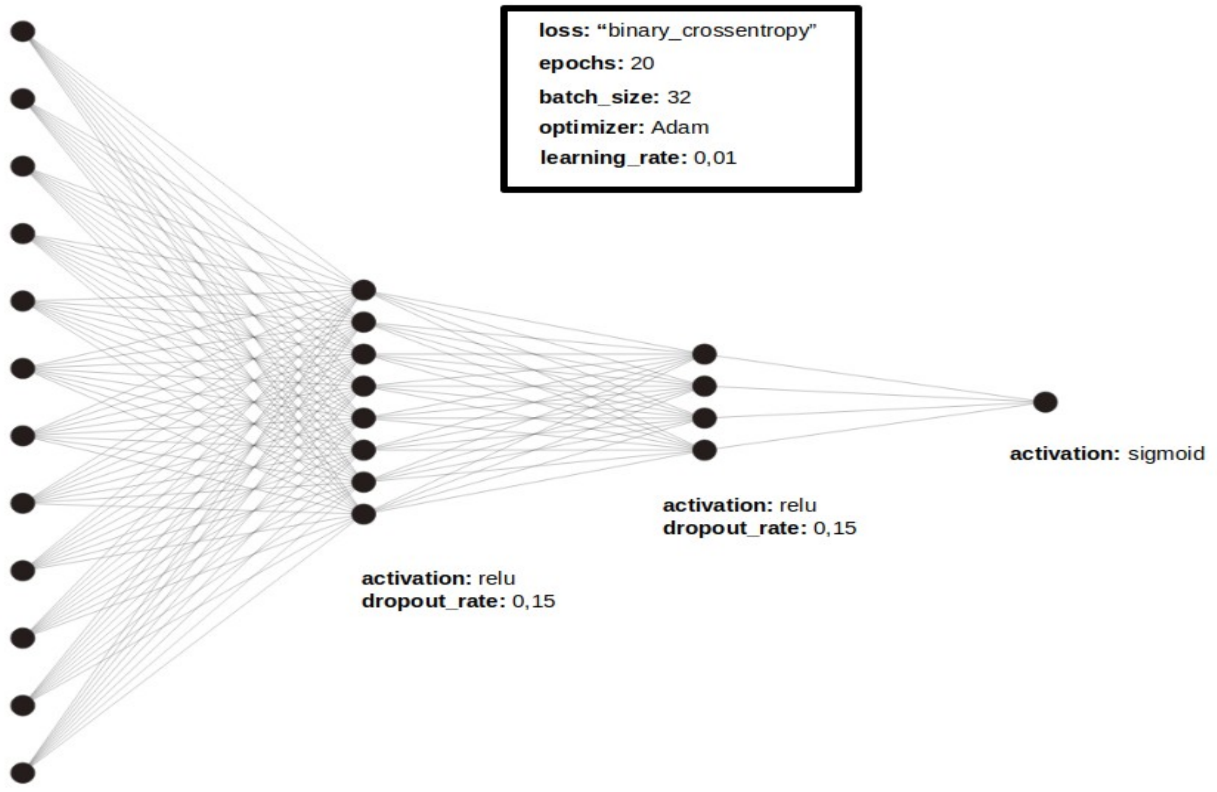
- ➢ The curve shows that we get better results by choosing a ccp_alpha higher than 0.
- ➢ However, if the ccp_alpha chosen is relatively big, the classifier tends to do random predictions. It happens because as alpha increases, more of the tree is pruned, which results on the total impurity of its leaves.



3.5.2. Neural Networks

a- Architecture model

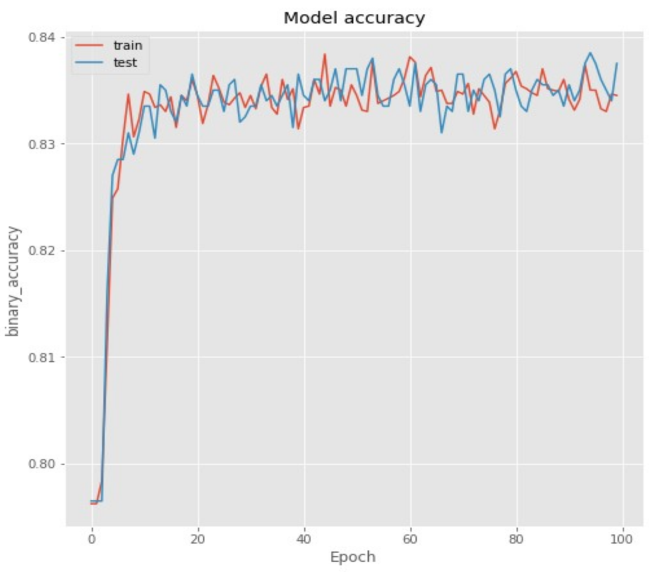
Here is the architecture of the model we used:

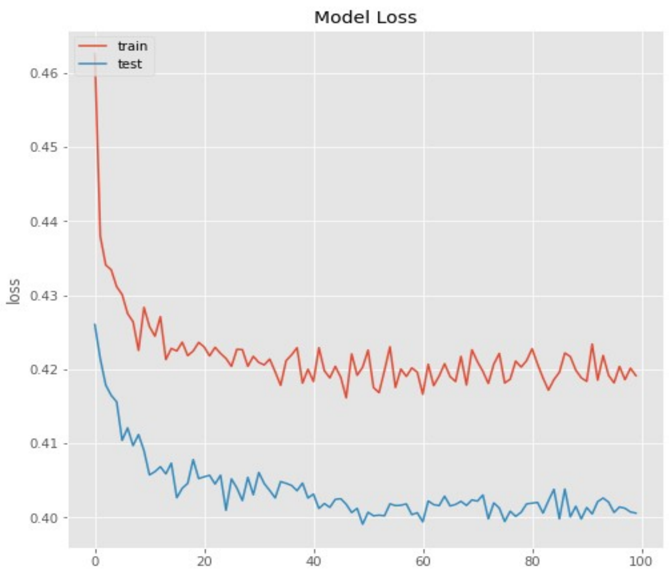


There are four layers: the input layer (12 units), two hidden layers (8 units | 4 units) and the output layer (single unit). The number of units for the hidden layers was chosen such that each layer has less nodes than the previous layer and more nodes than the following.

- ➢ We chose the relu activation function for the intermediate layers and a sigmoid activation function for the last layer since we are doing binary classification.
- ➢ A dropout has been introduced in the hidden layer in order to prevent the model from overfitting.

b- Learning curves





- ➢ In each epoch, the loss decreases and the accuracy increases, which means that the model is getting trained correctly
- ➢ The overall evolution of the curves stopped from a certain epoch, this means that the training was complete. Hopefully, the model did not overfit (the training loss stopped getting reduces).

3.5.3. Boosting

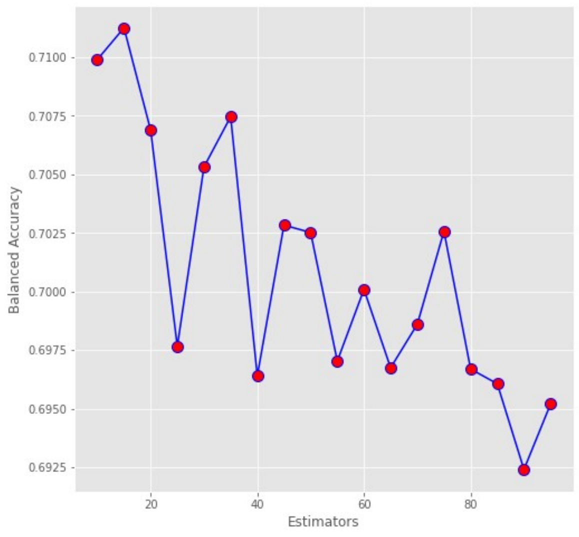
In this part, we are gonna do boosting to our previous Decision Tree using Adaboost.

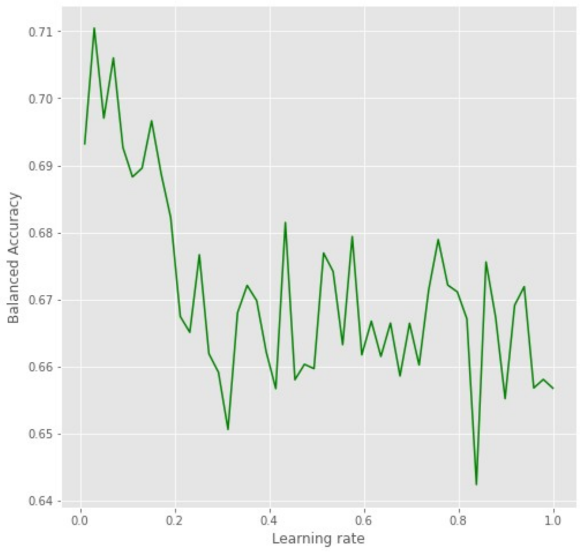
a- Choosing number of base estimators

We do a Grid Search in order to determine the number of base_estimators that would give us the best results on the testing set. The figure below on the left shows the results.

b- Varying the learning rate

Let’s now plot a similar curve for the evolution of the balanced in function of the learning rate.





➢ For our boosting, the best number of estimators to choose is **15**, since it corresponds to the highest value of balanced accuracy.

➢ Concerning the learning rate, the most appropriate value is close to **0.3**

➢ In our case, the boosting doesn’t improve considerably the performance of our decision tree because it was well designed thanks to the gridsearch done on the hyperparameters related to the construction.

3.5.4. SVM

a- Choosing the kernel

First, we need to determine the kernel to use for our transformation. In order to do that, we are going to use a 5-fold stratified cross validation on the training set and computer the mean balanced_accuracy. We can keep the default hyperparameters of each kernel, the idea is just to see how they behave.

Here are results that we found:

Kernel	linear	polynomial	radial basis function	sigmoid
Balanced_accuracy	0.5	0.67	0.68	0.54

➢ We got a metric value of 0.5 for the linear which corresponds to the balanced_accuracy of a random classifier.

➢ The linear and sigmoid kernels have such bad metrics because they are not suitable for our dataset. This was expected because the relation between the label and the features is quite complicated and it cannot be determined using a linear approach.

➢ We got a balanced_accuracy value of 0.67 for the polynomial kernel and 0.68 for the rbf, which are acceptable values. We can’t decide which of the two kernels is more adapted to our data because no tuning has been done yet.

Next, we keep both the polynomial kernel and the rbf kernel and we are going to tune their respective hyperparameters using Grid Search via 3-fold cross validation.

b- rbf kernel: Tuning hyperparameters

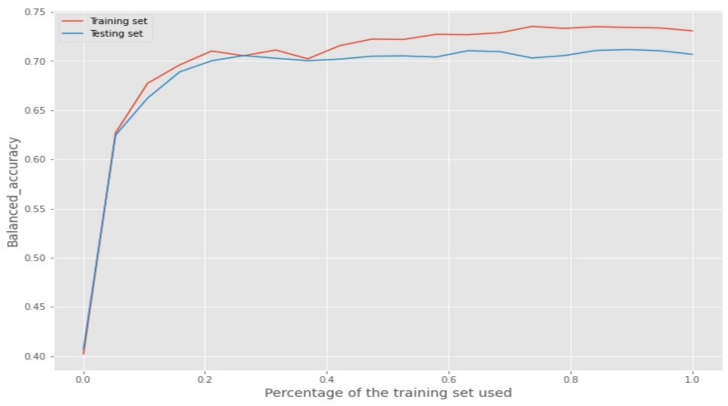
Here are the results we found for the rbf kernel using the Grid Search on gamma (line) and C (column):

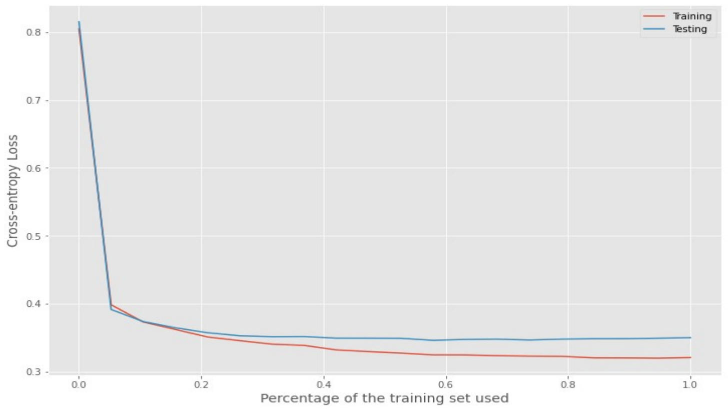
C / **Gamma**	0.1	1	5	10
0.001	0.50	0.50	0.50	0.50
0.01	0.50	0.51	0.60	0.65
0.1	0.56	0.68	0.70	0.70
1	0.50	0.61	0.63	0.62

The highest value corresponds to **gamma=0.1 & C=5.** Let's use those values to create our SVM model.

c- rbf kernel: Learning curves

Here are the learning curve for the SVM model:





➢ ➢ The more data we used, the better is our metric and the lower is the loss.

➢ The model performs very well starting from 40% of the total size of the training set. It means that the SVM does not need a lot of data for training.

d- Polynomial kernel

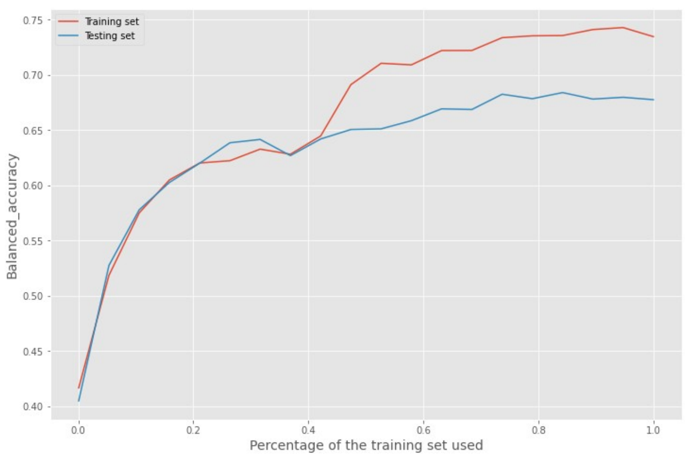
The polynomial kernel has one more parameter to tune than the rbf (degree). It is quite challenging in term of computational time, so we did it gradually by taking few values at each time. The values tested are:

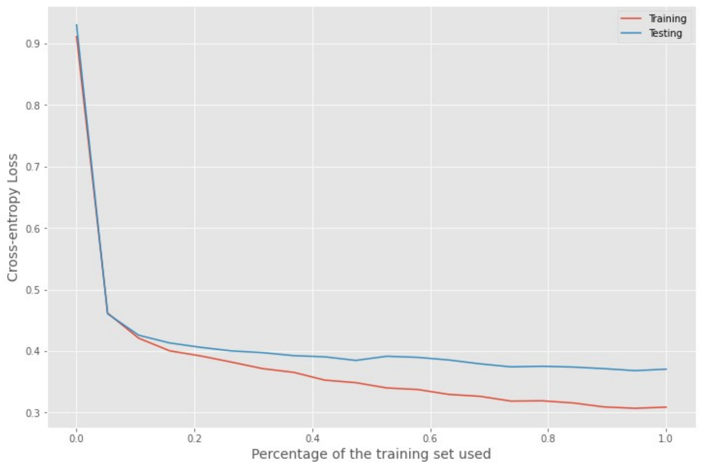
- range of values for gamma: [0.001, 0.01, 0.1, 1]
- range of values for C: [0.1, 1, 5]
- range of values for degree: [3, 5, 7]
- range of values for coef0: [0, 1, 2]

This takes almost 28 min to compute.

Finally, we found the highest metric value for **gamma=0.1 & C=1 & degree=5 & coef0=2**.

Next, we created the model using the hyperparameters above and here are the learning curves we got:





e- Comparing kernels

- Based on the curves, the polynomial kernel seems to need more data in order to be fitted than the rgb kernel (0.63 reached only after using 40% of training set to fit the polynomial, while the rgb reached it with just 10% of the training set).
- In the end, the rgb kernel reaches 70% of balanced accuracy, while the polynomial reaches only 68%.

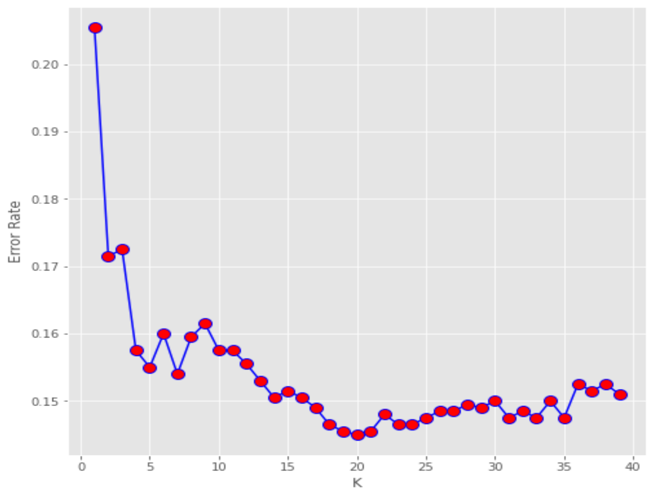
In conclusion, the rgb kernel is better suited for our dataset.

3.5.5. k-nearest neighbors

a- Determining best parameter k

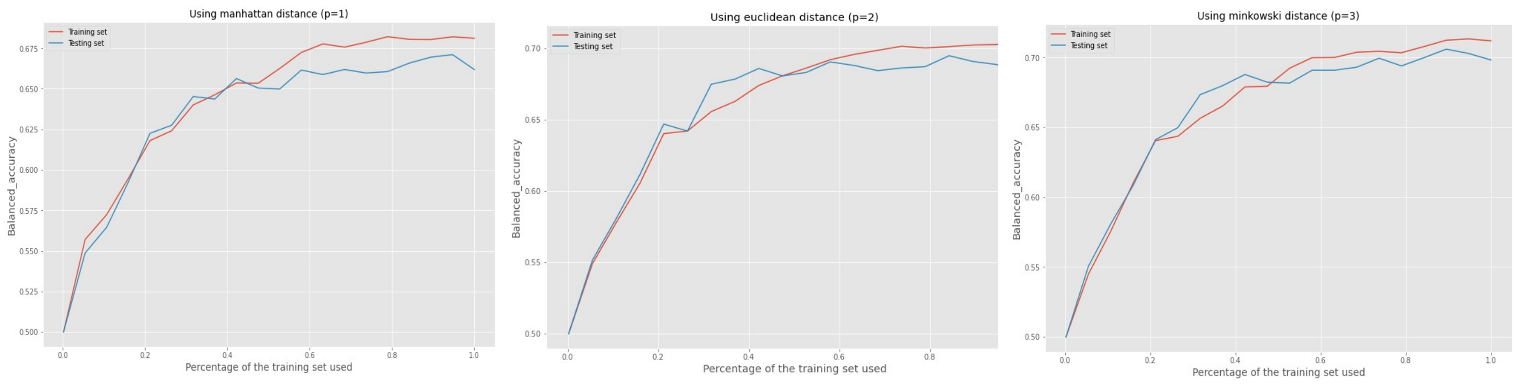
In order to determine the best value for k, we can use the elbow method:

- The error rate is the percentage of wrongly classified samples.
- The **best value for k is 20** since it corresponds to the minimum value of the error rate in the graph.



b- Comparing distances

Let's try to compare the manhattan_distance, the euclidean distance and the minkowski_distance with p=3.
Here are the learning curves using the three metrics:

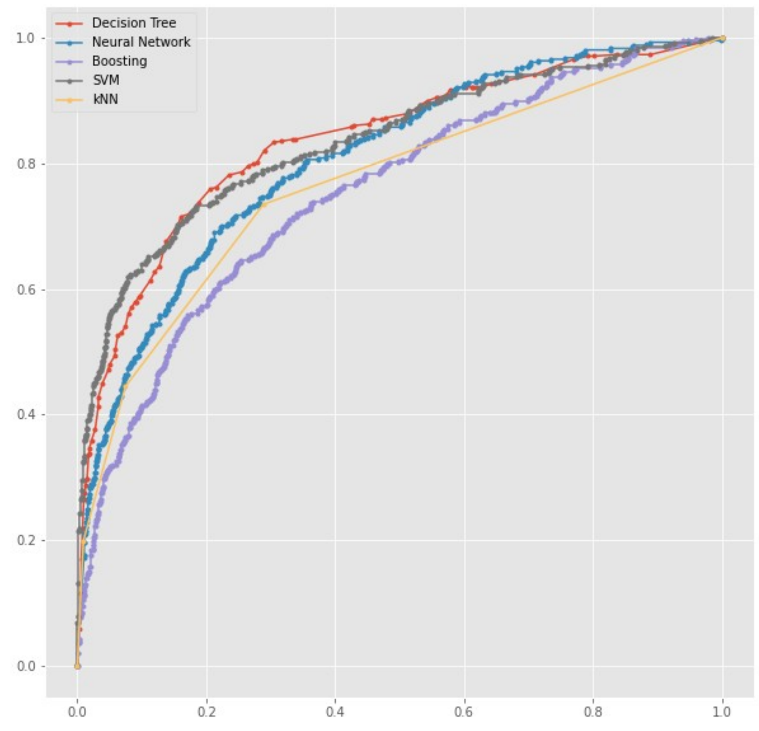


- From the three distances, the minkowski_distance power 3 gives the best result (we reach 70% balanced accuracy on the testing set).

3.6. Comparing Results

Now, let's compare the performance of the different models introduced in the previous parts, by looking at the balanced accuracy on the test set, the auc value and the ROC curves:

Model	**Balanced Accuracy**	**AUC**
Decision Tree	71 %	0.83
Neural Networks	63 %	0.81
Boosting	66 %	0.75
SVM	71 %	0.84
K-Nearest neighbors	70 %	0.77



- ➢ The Decision Tree and the SVM have the best balanced accuracy
- ➢ SVM has the highest AUC value
- ➢ From the ROC curves, we can clearly deduce that the SVM and the Decision Tree are the models that perform best in our problematic (they have a high true positive percentage for a low false positive percentage (it is clearer in the part where fpp is lower than 0.4).
- ➢ The Neural Network has the worst performance in term of balanced accuracy, it was expected because the data imbalance is very bad for the model, especially when we keep the prediction threshold at 0.5 (value by default).
- ➢ Overall, the **SVM is the model who performs the best for the Bank Customer Churn !**

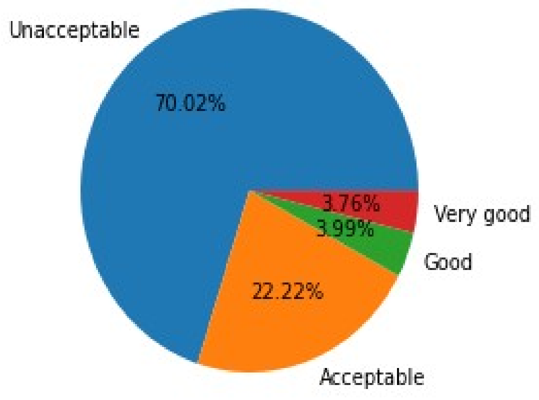
4. Second dataset: Car Evaluation

4.1. Introducing the dataset

This second dataset has a size of (1728, 7): there are 1728 samples and 6 features.

The label column refers to the car offer situation, there are four possible values: {Unacceptable, Acceptable, Good, Very Good}.

Once again, we chose an imbalanced dataset, but we have a multiclass classification problem now. The main motivation for choosing such dataset is to see how the imbalance can impact the performance of the models when there are more than two classes. In addition to that, the dataset is made according to a structural decision model, and we would like to see if the models would be able to discover this structure.



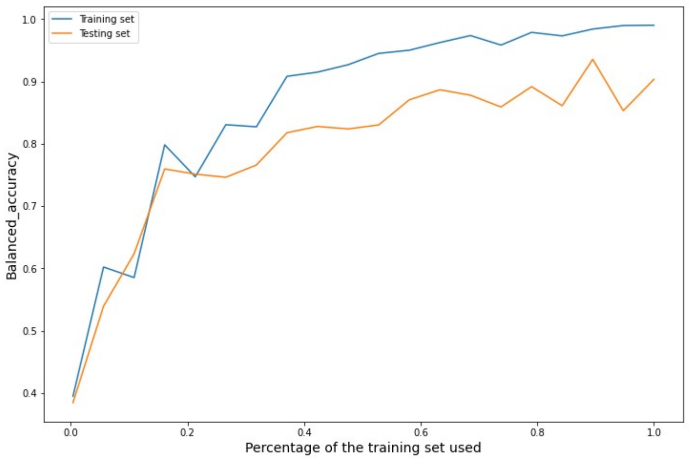
4.2. Preparing the data

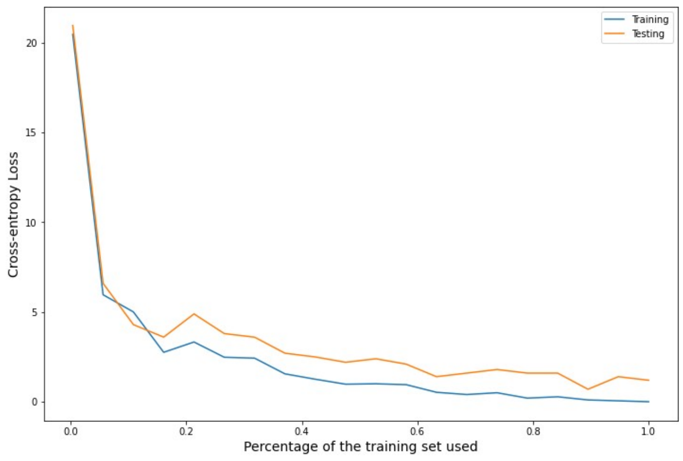
- ◆ The dataset does not contain any useless features, so we will keep the six of them.
- ◆ There are no missing data.
- ◆ All the features are categorical, so they need to be encoded.
- ◆ The label class should also be encoded since it has 4 string values.
- ◆ We divide once again the data into a training set (80%) and a testing set (20%) in a stratified way.
- ◆ Once again, we are going use the balanced accuracy metric and the log_loss to test the performance of our prediction models.

4.3. Experiments

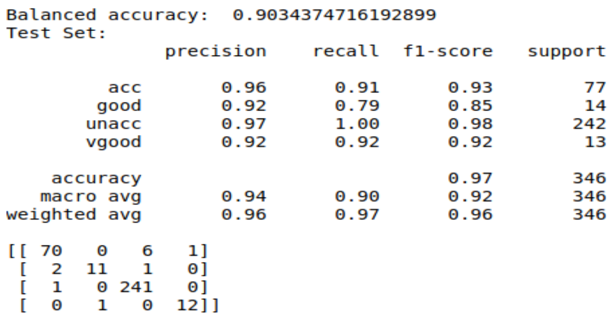
4.3.1. Decision Trees

The ‘gini’ criterion turned out to be better than ‘entropy’ again, here are its curves:





- ➢ The loss is very low and it reaches even 0 for the training set !
- ➢ Accordingly, the balanced accuracy reaches 100%.
- ➢ The first thought we can have is that the model has overfitted, however if that was the case, it would not have such good performance on the testing set.
- ➢ Therefore, the only possible explanation is that the model has figured out the relation between the features, that dominates the label choice !

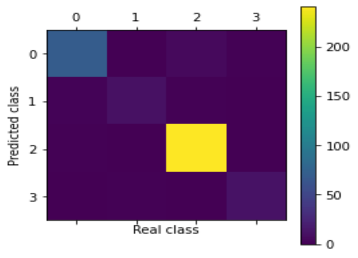


```
Balanced accuracy:  0.9034374716192899
Test Set:
                precision    recall  f1-score   support

         acc       0.96      0.91      0.93        77
        good       0.92      0.79      0.85        14
       unacc       0.97      1.00      0.98       242
       vgood       0.92      0.92      0.92        13

    accuracy                           0.97       346
   macro avg       0.94      0.90      0.92       346
weighted avg       0.96      0.97      0.96       346

[[ 70   0   6   1]
 [  2  11   1   0]
 [  1   0 241   0]
 [  0   1   0  12]]
```



We do once again the Grid Search in order to choose values for max_depth and min_samples_split.

We find **max_depth=14 & min_samples_split=2.**

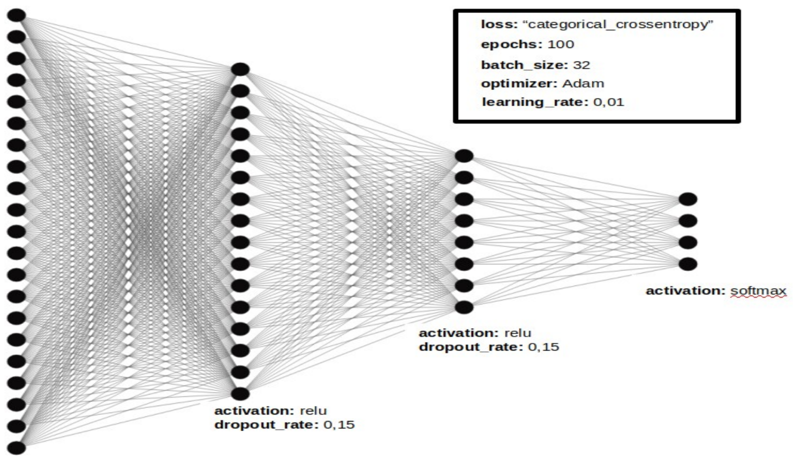
Now, let’s take a look at the confusion matrix:

- ➢ The accuracy is equal to 97%
- ➢ The balanced accuracy is equal to 90%
- ➢ We have a very high recall for all four classes (‘good’ has 79% recall but it is a high value because it corresponds to only 3 miss classified true ‘good’ samples, since we only have 14 ‘good’ samples in our testing set.

4.3.2. Neural Networks

The figure shows the architecture used for the Artifical Neural Network:

- ➢ For the multiclass classification, we use the **softmax** activation function in the output layer.
- ➢ Concerning the loss, we use the **‘categorical_crossentropy’**



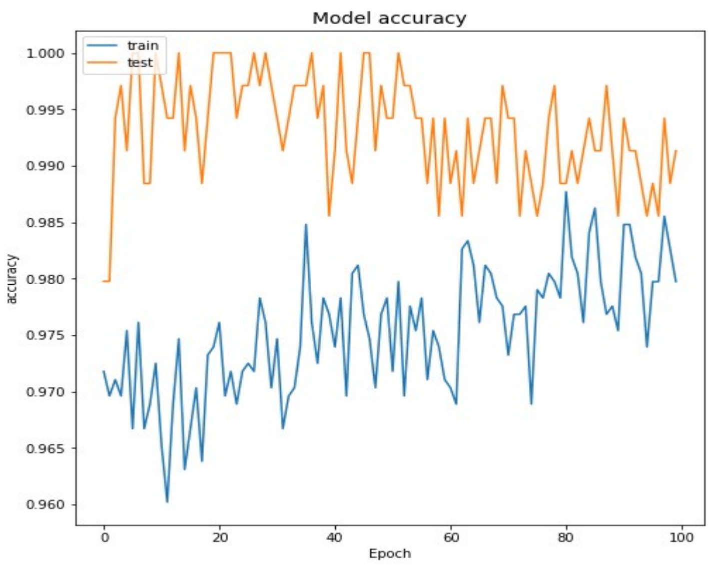
loss: “categorical_crossentropy”
epochs: 100
batch_size: 32
optimizer: Adam
learning_rate: 0,01

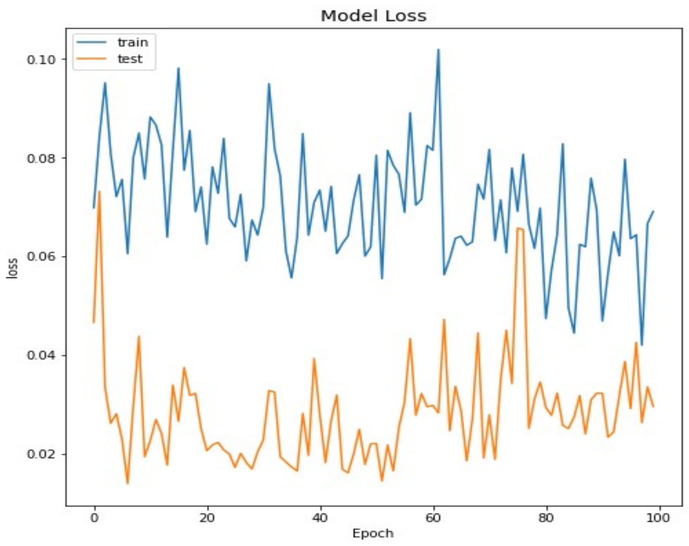
activation: relu
dropout_rate: 0,15

activation: relu
dropout_rate: 0,15

activation: softmax

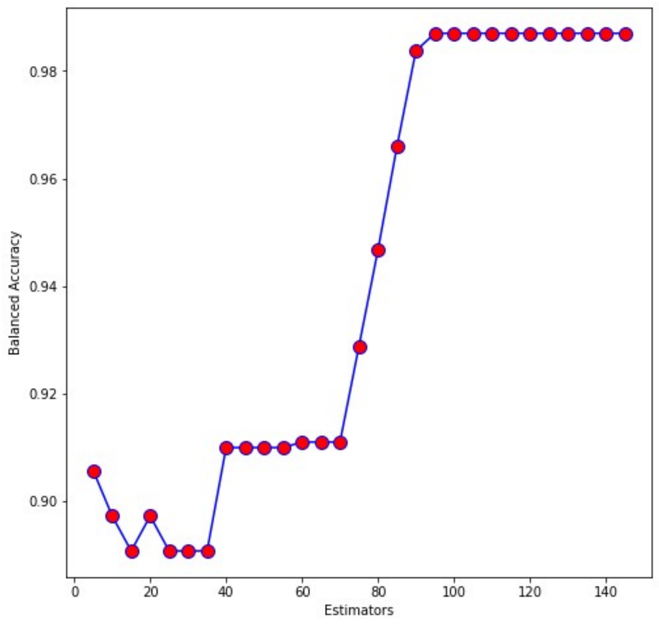
Here are the learning curves:

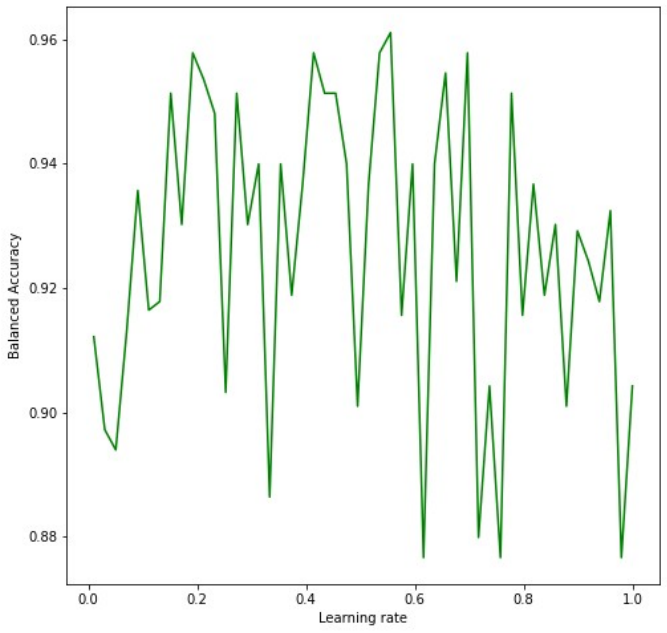




➢ The accuracy is very high (almost 100%) even for the testing set, is was expected since the data is structural.

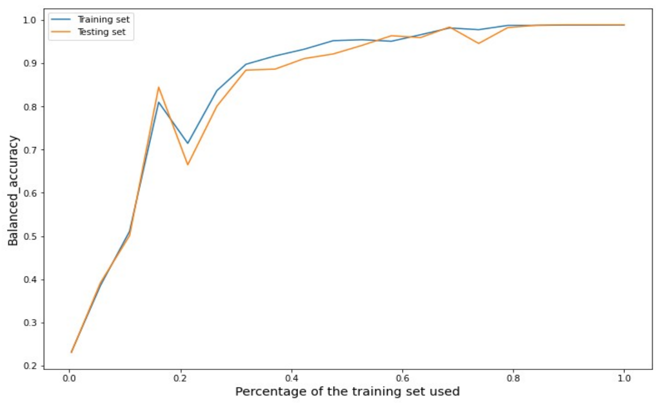
4.3.3. AdaBoost

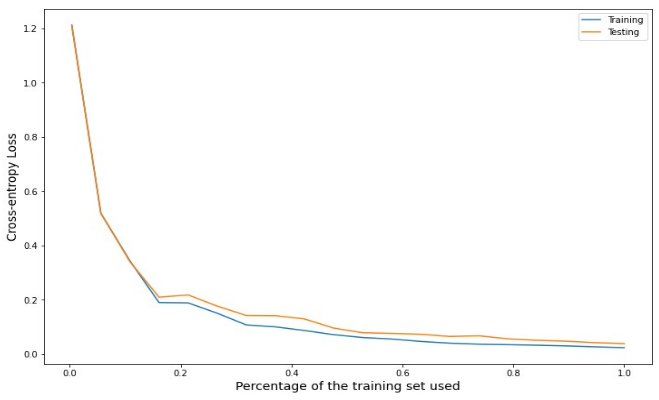




➢ The two figures show that we can improve the performance of decision tree, we should take a high number of estimators (**100**), and a learning rate of **0.5** seems to be the most efficient.

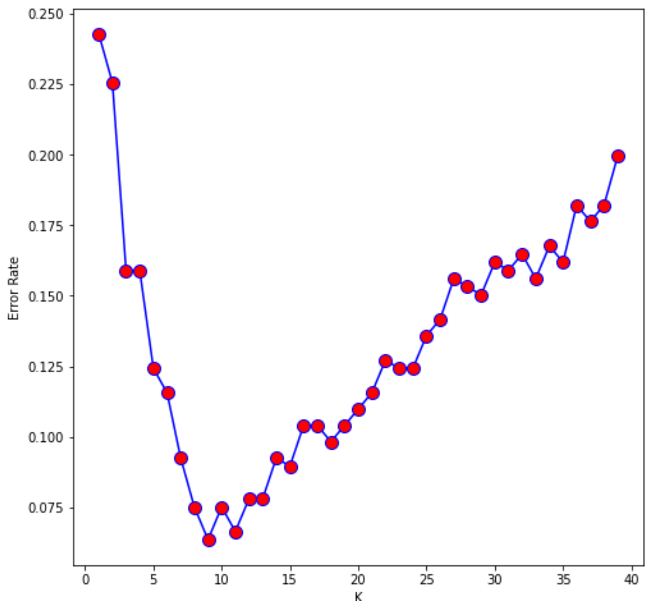
4.3.4. SVM

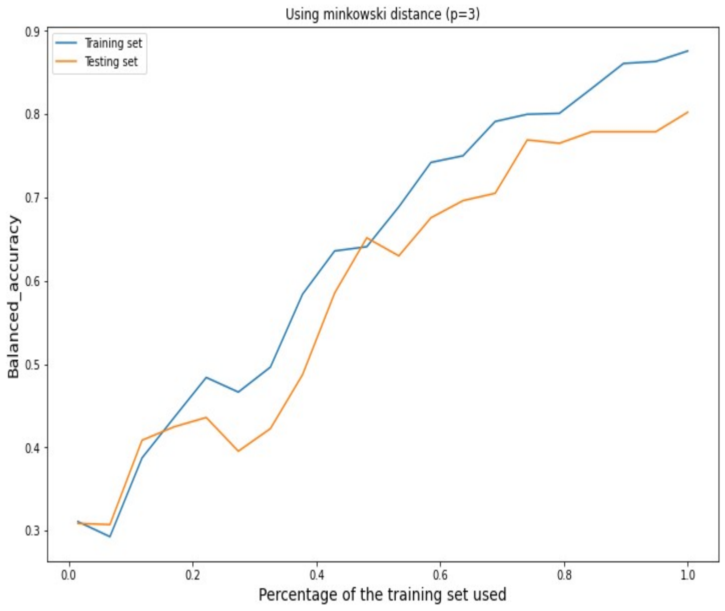




➢Both the rbf and the polynomial kernel reach a 100% accuracy on the testing set, which proves once again that the SVM is a very efficient supervised learning model (the curves above are for the rbf).

4.3.5. k-Nearest Neighbors





➢ The error rate is minimal for **k=9**, making it the best value to choose for the number of neighbors.

➢ The distance curves look similar, we can see that the accuracy reaches 90% for the training set and 80% for the testing set.

4.4. Comparing Results

Model	**Balanced Accuracy**
Decision Tree	90 %
Neural Networks	99 %
Boosting	90 %
SVM	100 %
K-Nearest neighbors	80 %

- ➢ The SVM is once again in the top, followed closely by the Neural Networks. Those models got perfectly the structure behind the data.
- ➢ Even though the classes are imbalanced, the results were very satisfying overall. This happened mostly because of the existing relation between the features.

5. Ways of improvements

In order to improve the performance of supervised learning models on such imbalanced data, we can think of some clues:

✔ The most obvious is to balance the data, either by undersampling if there are enough samples of each class on the training set, or by oversampling the minority class, or by some smooth sampling making a trade-off between the two.

✔ Another way to deal with binary imbalanced data might be to choose a different prediction threshold. The default threshold is equal to 0.5 (if the probability is lowest to the threshold, the sample is classified negative, otherwise it is classified positive). Having a prediction threshold closer to the minority class can reduce its dominance by the majority class. The most suitable threshold could have been chosen using a validation set or a cross-validation, for example.

✔ A third solution that can help reduce the complexity of the model and help get better results is to check the importance of the features and remove the features that don't have any influence on the predictions.