Project 1 - Classification algorithms

Valentin Absalon - S214256 Yassine Maziane - S184384

October 17, 2021

1 Decision Tree

1.1 Observation of the decision boundary based on tree complexity

1.1.1 The decision boundary for each hyper-parameter

A decision graph has been produced using **plot_boundary** for each hyper-parameter. The hyper-parameter that we use is **min_samples_split**. This parameter represents the minimum number of samples required to split an internal node in a decision tree.
We observe that the boundaries of graphs with the lowest **min_samples_split** values are the most complex ones. These graphs contain a lot of boundaries and the shapes of these get more and more intricate as the hyper-parameter decreases. Indeed one can see on figure 1 that the boundaries are far more sophisticated than on figure 6.This could have been expected.

When the hyper-parameter value is low, e.g fixed to 2, a node with a non-zero impurity may split into two leaves or nodes if it contains at least min_samples_split objects. That way, the internal node is split into 2, testing an attribute, and doing so amounts to dividing the original space again. Notice that when the hyper-parameter is fixed to 2, the training set is perfectly classified. Oppositely, high values of the hyper-parameter lead to a lesser number of internal nodes and leaves. That is, the original space will be less divided.

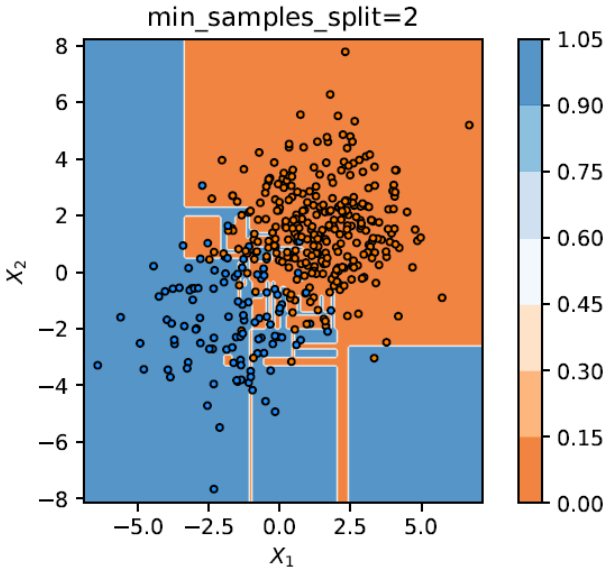

Figure 1: Decision boundary plot of decision tree models with min_sample_size=2 for an unbalanced dataset

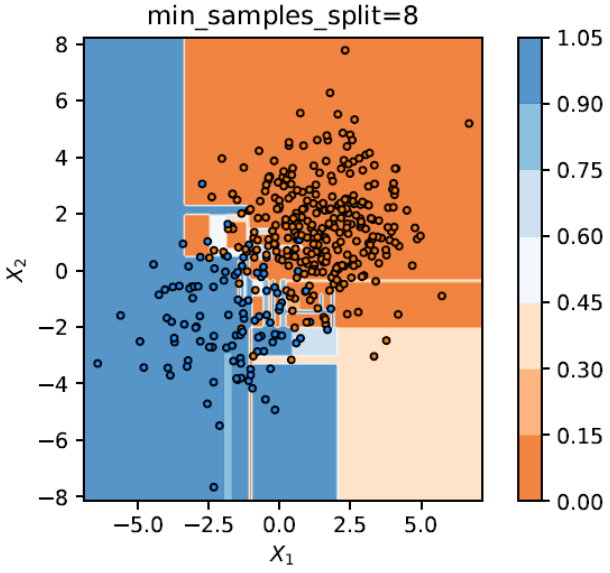

Figure 2: Decision boundary plot of decision tree models with min_sample_size=8 for an unbalanced dataset

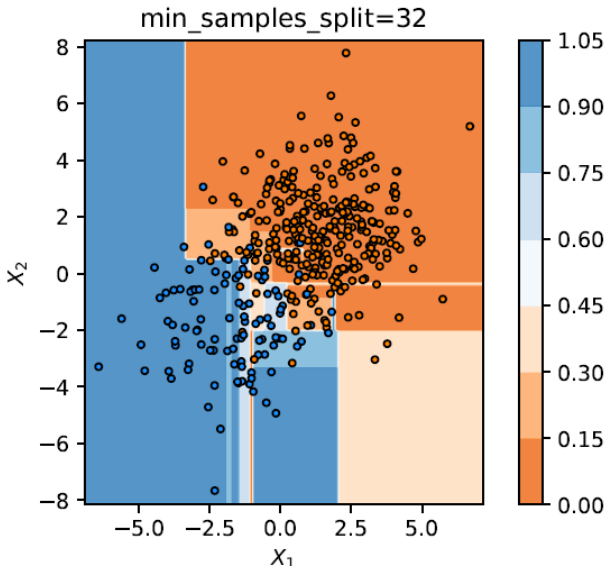

Figure 3: Decision boundary plot of decision tree models with min_sample_size=32 for an unbalanced data set

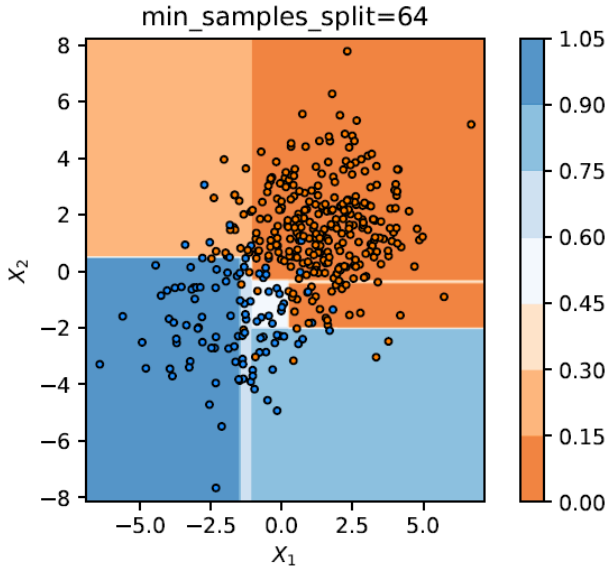

Figure 4: Decision boundary plot of decision tree models with min_sample_size=64 for an unbalanced dataset

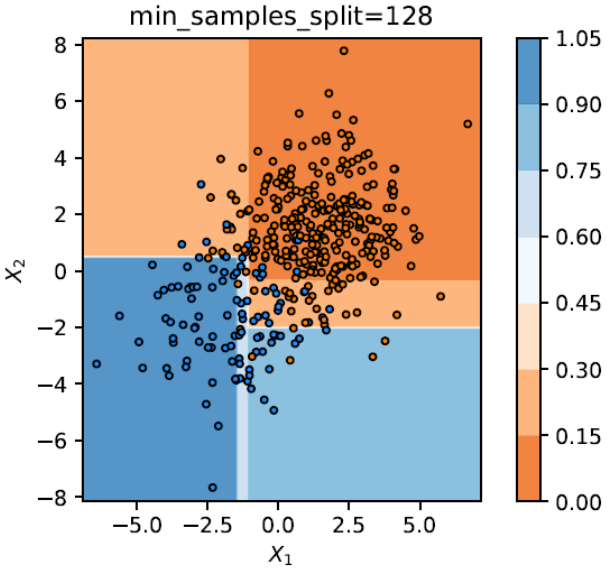

Figure 5: Decision boundary plot of decision tree models with min_sample_size=128 for an unbalanced dataset

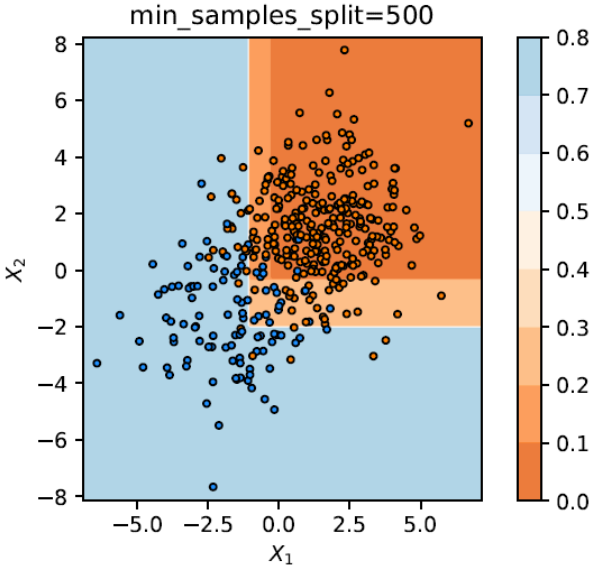

Figure 6: Decision boundary plot of decision tree models with min_sample_size=500 for an unbalanced dataset

1.1.2 Underfitting and overfitting

One can clearly observe underfitting on figure 6 and figure 5 where the space has been split in a very few number of zones, the boundaries are far too simple to classify such a set of data correctly. We can observe a lot of blue points in the orange region of the space and vice versa.
In the opposite way, we can clearly observe overfitting on figure 1 or 2. The space division is far too complex to be generalized, particularly in figure 1 where the decision tree created as many internal nodes as necessary to perfectly classify the training data. The tree does not allow an orange point

from the training set to be in what could be expected as the blue region of space. This leads to the creation of small orange regions in the blue space and vice versa.

1.1.3 Prediction confidence

Prediction confidence is a measure of the confidence of the model in classifying a given object, depending on its confidence, we may or may not trust the class he gave to the object. In the case of decision trees, the prediction confidence is the probability returned by the $predict - proba$ function.

The function returns an array where the i-th object is an array containing the probabilities of belonging the each classes. The probability that the i-th object belongs to the j-th class is simply given by the ratio of the number of objects of class j in the current leaf divided by the number of objects on the whole leaf. This can be better seen when leaves have a non-zero impurity, if a test sample ends up on that leaf while there are x samples of class 1 and y samples of class 2, both x and y coming from the training set, the probability that the test sample belongs to class 1 is $\frac{x}{x+y}$ and the probability that it belongs to class 2 is $\frac{y}{x+y}$. Notice we supposed that the leaf was only made of 2 classes.

We thus understand that prediction confidence of a sample is directly linked to the impurity of the leaf the sample is going to end up on. As the smaller values of the hyper-parameter allow leaves to be generally more pure, the prediction confidence is higher, it is even perfectly confident for the smallest value.

1.2 Visual Guess

1.2.1 Boundaries

Decision trees' internal nodes test an attribute and partition the initial space of dimension #attributes into sub spaces whose boundaries are aligned with the different axes. The more internal nodes we have, the more borders we have on decision boundaries plot.

1.2.2 Unbalanced Problem

We observe that the orange area tends to be larger than the blue's.

1.3 Average test set accuracy

In this section, we decided to show our result in a table 1.3

Hyper parameter	2	8	32	64	128	500
Accuracy average	0.8834	0.8842	0.9015	0.9103	0.9029	0.8637
Standard deviation	0.01582	0.0177	0.0098	0.0108	0.0127	0.0140

Table 1: Average testing set accuracy over 5 data sets with standard deviation

First of all, we can see that the average accuracy increases with the hyper- parameter. Nevertheless, we have a stabilization of the average when min_samples_split is between 32 and 128. We obtain the maximum at min_samples_split=64 then we have a decrease after this hyper parameter value. Notice the worst hyper parameter values in terms of mean are the largest and smallest value, this is due to their respective underfitting and overfitting nature.

2 K-nearest neighbors

2.1 Observation of the decision boundary

2.1.1 The decision boundary for each hyper parameter

A decision graph has been produced using **plot_boundary** for each hyper-parameter (figure 7 to 11). The hyper-parameter that we use is n_neighbors. This parameter represents the number of neighbors

we are considering for a test sample to be classified.

2.1.2 Evolution of the decision boundary

We observe that as the hyper-parameter value increases, the space is partitioned into more regions and the prediction confidence gets lower, especially in the intersection zones of the distributions where the confidence seems to be minimal. This could be expected since this is the region where opposite points are the most concentrated.

For example, for k=1, a test sample will get the class of the closest point from the training set. The model is always confident regardless of the area of data in which it is located. For example, the data can be located in intersection zones but still the model will be perfectly confident. This is a clear case of overfitting. Notice that by closest point, we mean the point that has the smallest Minkowski distance to our sample. Oppositely, for value of k = 50 and k = 100, the sample will line up with the majority even if the prominent class exceeds the other one by a simple unity, this will lead to very small confidence prediction in intersection zones. For k=500, the model is clearly underfitting. Indeed, as you consider 500 neighbors out of 1000 in the training set, the orange class dominates in the upper left and in the intersection zone where they are on average 3 times more. The bottom left region is pretty impure since even being in the old "blue zone",there are at most 250 blue points in its radius and 250 orange points in order to get to 500, even if these orange points happened to be quite far. This leads to a large impurity and therefore a very low prediction confidence. Notice we considered that the 3 to 1 proportion was perfectly respected for the latter explanation.

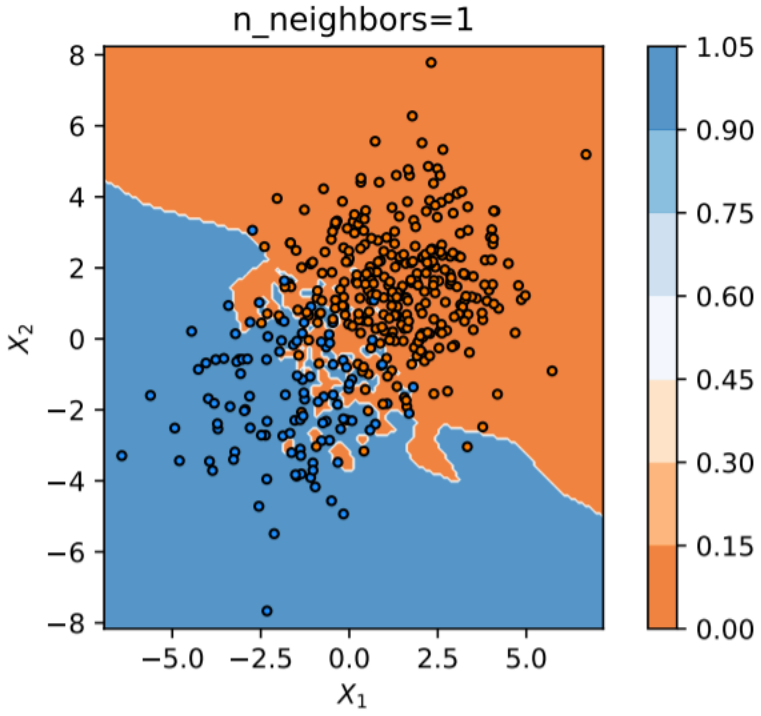

Figure 7: Decision boundary plot of k-nearest neighbours with k=1 for an unbalanced data set

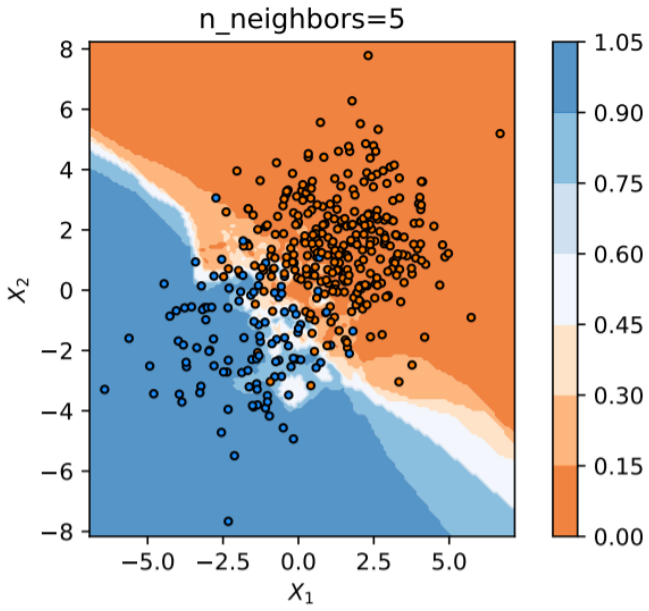

Figure 8: Decision boundary plot of k-nearest neighbours with k=5 for an unbalanced data set

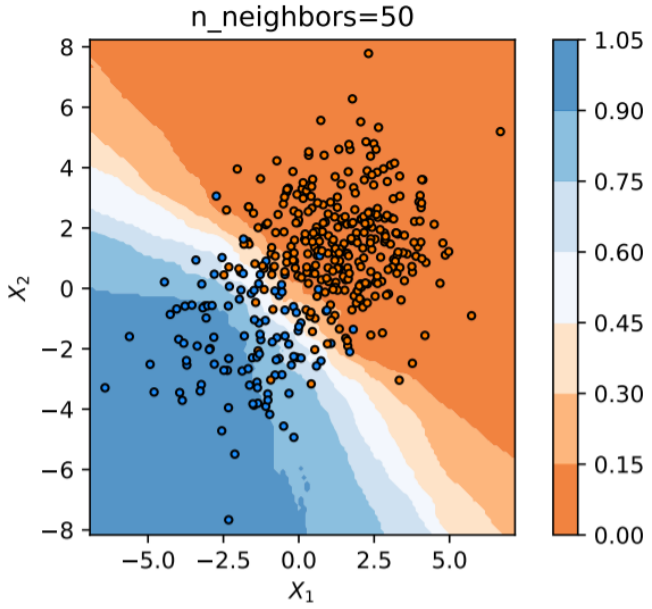

Figure 9: Decision boundary plot of k-nearest neighbours with k=50 for an unbalanced data set

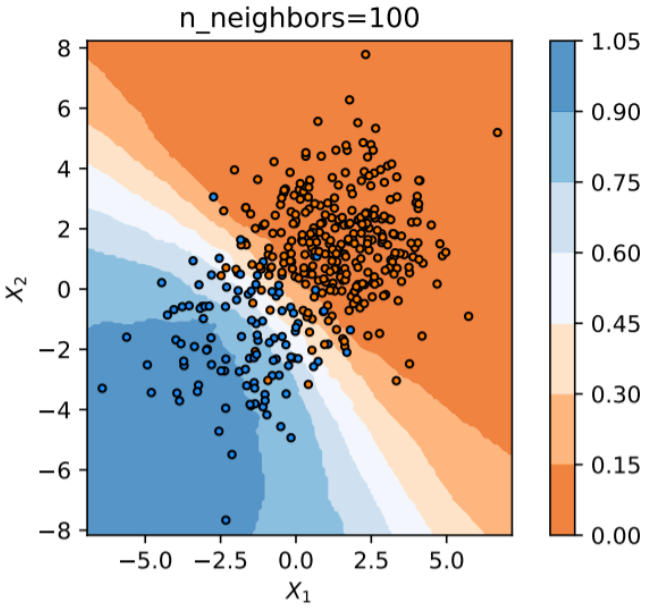

Figure 10: Decision boundary plot of k-nearest neighbours with k=100 for an unbalanced data set

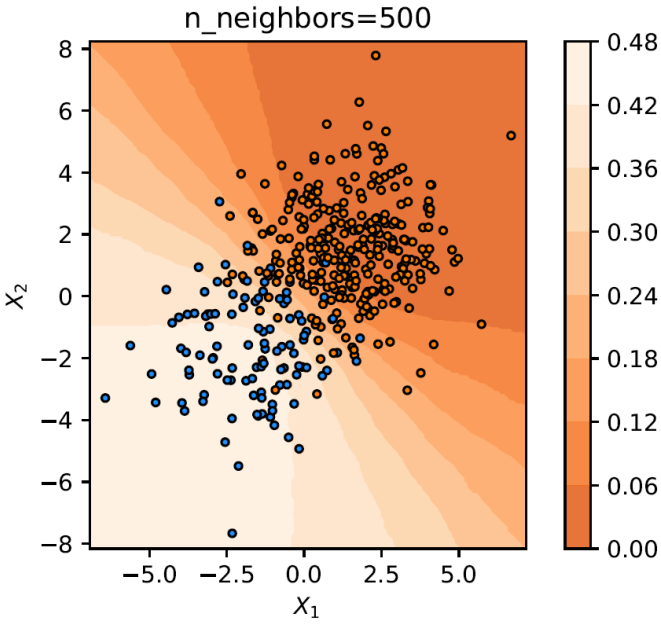

Figure 11: Decision boundary plot of k-nearest neighbours with k=500 for an unbalanced data set

2.2 Ten-Fold Cross Validation

2.2.1 Methodology

1. Partition the total set into 10 random subsets of equal length
2. Learn the model on the 9 other subsets

3. Test the model on the objects that are in the subset
4. Save the previously computed error and change the current subset into the next one
5. Repeat the 4 previous steps until you have tested each subset, one time each only.
6. Compute the average error

Following this methodology for every value of **n_neighbors** we will be able to find the value that leads to the least average error, it will be the optimal value.

2.2.2 Optimal value

In this section, we decided to show our result in a table 2

n_neighbors	1	5	50	100	500
mean accuracy	0.8976	0.9199	0.9269	0.9246	0.922

Table 2: Mean accuracy over 5 different values of n_neighbors

We notice that the optimal value is **n_neighbors** = 50. Its mean accuracy is equal to 0.9269.

These different values that we obtain at table corroborate our decision boundary based intuition. Indeed, a model with a good accuracy is a model which is not overfitting or underfitting. The model has to be less confident in the intersection zones and quite confident in pure zones, where we have a large majority of points of the same class.

Out of the different plots, figure 9 seems to be the best candidate It's the best model that we can obtain out of the set of different values the hyper-parameter can take.

2.3 Observation of the optimal value

2.3.1 The evolution of mean test accuracies

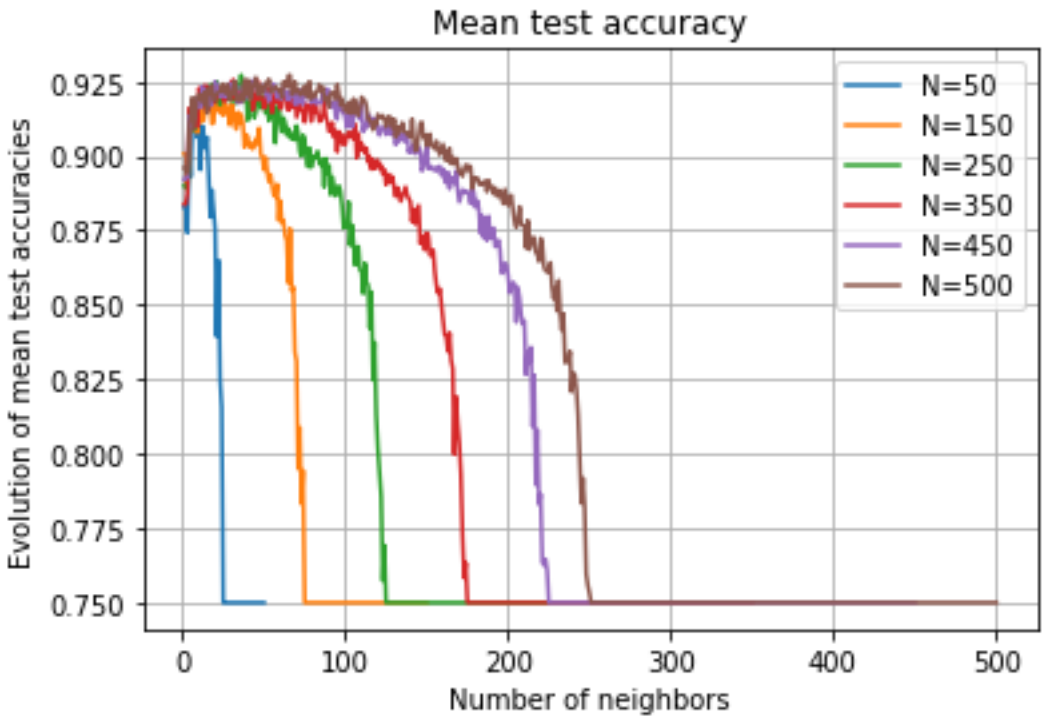

Figure 12: Evolution of the mean accuracy in terms of n_neighbors of different training set sizes of N

We decided to consider that when the number of neighbors is equal to 0, this value doesn't exist and so the number of neighbors begins at 1.

2.3.2 The optimal value

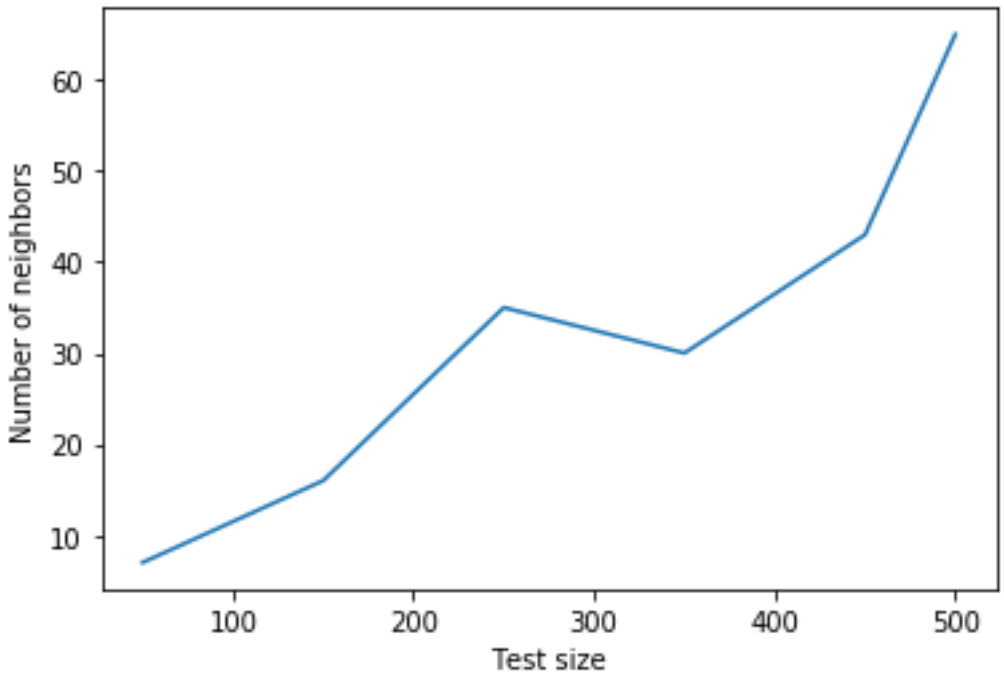

Figure 13: Evolution of the optimal n_neighbors over 6 different training set sizes

We notice that we obtain an almost linear evolution of the optimal value of **n_neighbors** over the all sizes that we studied before. When N increases, k increases but we notice a decrease between two point (**n_neighbors**=250 and **n_neighbors=**350), k should grow slower than N.
We also notice on figure 12 that the range of values for number of neighbor which lead to +0.9 accuracy gets larger as N increases.

3 Logistic Regression

3.1 Linear Decision Boundary

Our data set features are of the form $\underline{x} = (X[0], X[1]) = (x_0, x_1)$ while the label $Y \in \{-1, 1\}$ indicates the membership of the object x to the class positive or negative respectively denoted by Y=+1 or Y=-1.
The conditional probability is given by

$$P(Y = +1|\underline{x_i}; \omega_0, \underline{w}) = \frac{1}{1 + exp(-\omega_0 - \underline{w}^T \underline{x_i})} \tag{1}$$

where $\underline{w}$ is a vector of two parameters w_1 and w_2

To simplify the notations, $\underline{x} = (X[0], X[1]) = (x_0, x_1)$ is transformed into $\underline{x} = (1, X[0], X[1]) = (1, x_0, x_1)$ so that 1 becomes

$$P(Y = +1|\underline{x_i}, \underline{\theta}) = \frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i})} \tag{2}$$

where $\underline{\theta} = (\theta_0, \theta_1, \theta_2) = (\omega_0, w_1, w_2)$ are the trainable parameters.
We know that the decision boundary of a logistic regression is the set of all points x_i that satisfy

$$P(Y = +1|\underline{x_i}, \underline{\theta}) = \frac{1}{2} \tag{3}$$

This leads to the following developments,

$P(Y = +1|\underline{x_i}, \underline{\theta}) = \frac{1}{1+exp(-\underline{\theta}^T \underline{x_i})} = \frac{1}{2}$
$\Leftrightarrow 1 = exp(-\underline{\theta}^T \underline{x_i})$
$\Leftrightarrow 0 = -\underline{\theta}^T \underline{x_i}$
$\Leftrightarrow \theta_0 + \theta_1 * x_0 + \theta_2 * x_1 = 0$
$\Leftrightarrow x_1 = -\frac{\theta_1}{\theta_2} * x_0 - \frac{\theta_0}{\theta_2}$
where x_1 is a linear function of x_0, thus the decision boundary is linear.

3.2 Equivalent ways

We are asked to show that maximizing the likelihood of the observed data is equivalent to minimizing the negative log-likelihood loss function. To do so we will start by working on the first part.

Let D be the set of input-output pairs of size N. The assumption made by the logistic regression is that

$$P(Y = +1|\underline{x_i}, \underline{\theta}) = \frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i})} \tag{4}$$

therefore

$$P(Y = -1|\underline{x_i}, \underline{\theta}) = 1 - \frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i})} \tag{5}$$

by law of total probability.

One can notice that using $Y \in \{-1, 1\}$, there is a condensed way of writing P(Y=y_i) that is

$$P(Y = y_i|\underline{x_i}, \underline{\theta}) = \frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i} * y_i)} \tag{6}$$

It is easy to check that by plugging the different values of Y, we get either to 4 or 5.
Coming back to our main problem, we wish to find the parameters θ that will lead to the best classification possible.

Since Y is a binary label to which a probability is associated, we may see it as a Bernoulli random variable Y $\sim$ Bern($\sigma(z)$). We can then compute its mass function

$$P(Y = y_i|\underline{x_i}, \underline{\theta}) = \frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i} * y_i)} \tag{7}$$

Given a training set D of size N, the likelihood is

$$L(\underline{\theta}) = \prod_{i=1}^{N} P(Y = y_i|\underline{x_i}, \theta) \tag{8}$$

Since we are only interested in finding the optimal parameters, we may look for the values of θ that maximize 8

$$\theta* = argmax(\prod_{i=1}^{N} P(Y = y_i|\underline{x_i}, \theta)) \tag{9}$$

Since the log is a monotonic function, that is, argmax(f(x))=argmax(log(f(x))) over x, we may take the log of 9

$$\theta* = argmax(log(\prod_{i=1}^{N} P(Y = y_i|\underline{x_i}, \theta))) \tag{10}$$

over all possible values of θ.
Using the product property of log we get

$$\theta* = argmax(\sum_{i=1}^{N} log(P(Y = y_i|\underline{x_i}, \theta))) \tag{11}$$

We now plug in the last equation the expression of P given by 7

$$\theta* = argmax(\sum_{i=1}^{N} log(\frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i} * y_i)})) \tag{12}$$

Now using the property log(1/x)=-log(x) we get

$$\theta* = argmax - (\sum_{i=1}^{N} log(1 + exp(-\underline{\theta}^T \underline{x_i} * y_i))) \tag{13}$$

Since argmax(-f)=argmin(f) we get

$$\theta* = argmin(\sum_{i=1}^{N} log(1 + exp(-\underline{\theta}^T \underline{x_i} * y_i))) \tag{14}$$

Now re using the inverse property of logs but in the other way we get

$$\theta* = argmin - (\sum_{i=1}^{N} log(\frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i} * y_i)}) \tag{15}$$

and we plug in 7 to get

$$\theta* = argmin - (\sum_{i=1}^{N} log(P(Y = y_i|\underline{x_i}, \theta)) \tag{16}$$

Which is the expected result, the argmin of the negative log-likelihood to a factor 1/N. The absence of this factor does not matter since argmin(f(x))=argmin(k*f(x)) under some basic hypothesis.

PS : In the next question it is assumed that $Y \in \{0, 1\}$ and not $Y \in \{-1, 1\}$, we could answer question 3.2 in the same way using a different mass function, it was actually our first attempt but the proof is a bit longer. We will send it by mail if it was the one you expected us to perform.

3.3 Loss function gradient

The expression of the negative log-likelihood is given by

$$L(\theta) = \frac{-1}{N} \sum_{i=1}^{N} log(P(Y = y_i|\underline{x_i}, \underline{\theta})) \tag{17}$$

Let's first develop 17, its form comes from the fact that the likelihood has been log transformed, its initial form was derived from

$$L(\theta) = \prod_{i=1}^{N} (\frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i})})_i^y * (1 - \frac{1}{1 + exp(-\underline{\theta}^T \underline{x_i})})^{(1-y_i)} \tag{18}$$

Notice that 1-sigm(z)=sigm(-z).
Let us work with the form

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} y_i * log(1 + exp(\underline{-\theta}^T * \underline{x_i})) + (1 - y_i) * log(1 + exp(\underline{\theta}^T * \underline{x_i})) \tag{19}$$

In order to form the gradient of 19, let's take its derivative with respect to θ_j.
Since $(log(f))' = \frac{f'}{f}$ we get

$$\frac{\partial L(\theta)}{\theta_j} = \frac{1}{N}\sum_{i=1}^{N}(-y_i) * \frac{exp(\underline{-\theta}^T * \underline{x_i})}{1 + exp(\underline{-\theta}^T * \underline{x_i})} * x[j] + (1 - y_i)\frac{exp(\underline{\theta}^T * \underline{x_i})}{1 + exp(\underline{\theta}^T * \underline{x_i})} \tag{20}$$

We now factor x[j] where x=[1,x_0,x_1] and the $-y_i$. We get, setting $z = \underline{\theta}^T * \underline{x_i}$ to simplify notations.

$$\frac{\partial L(\theta)}{\theta_j} = \frac{1}{N}\sum_{i=1}^{N}[(\frac{exp(z)}{1 + exp(z)}) - y_i(\frac{exp(-z)}{1 + exp(-z)} + \frac{exp(z)}{1 + exp(z)})] * x[j] \tag{21}$$

Simplifying, we now get

$$\frac{\partial L(\theta)}{\theta_j} = \frac{1}{N}\sum_{i=1}^{N}[\frac{1}{1 + exp(-z)} - y_i] * x[j] \tag{22}$$

Which is nothing else than

$$\frac{\partial L(\theta)}{\theta_j} = \frac{1}{N}\sum_{i=1}^{N}(P(Y = 1|\underline{x_i}, \underline{\theta}) - y_i) * x[j] \tag{23}$$

Since the gradient is a vector of derivatives, we get

$$\nabla_\theta L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(P(Y = 1|\underline{x_i}, \underline{\theta}) - y_i)\underline{x_i} \tag{24}$$

Which is a vector and the result expected.

3.4 Initial value of θ

We proposed to set all the parameters equal to zero and observed that they tend to converge to what appears to be the quasi optimal parameters. We say quasi because we may never reach the true optimal values of the finite number of iterations and the finite precision of our machines. According to the literature, there is no way to predict pretty good initial values of the parameters and any random values will tend to converge to the optimal ones given a sufficiently large number of iterations and a small enough learning rate since the problem is convex.

3.5 Decision boundary

Throughout this section, we will use a learning rate = 0.01 based on scientific resources on the internet [1].

The learning rate is an important hyper-parameter that influences the convergence and the convergence speed of our model.
The plot is on figure 16.

We observe that the decision boundary is a straight line as it was expected and that the distinction between the two regions gets more and more clear as the number of iterations grows.

One more thing we may notice is that the decision boundary seems to be parallel to the median of the two distributions centers. Furthermore, we expected the boundary to be the median of the two distributions centers but slightly closer to the blue distribution, $\frac{1}{4}$ closer to respect the proportions.

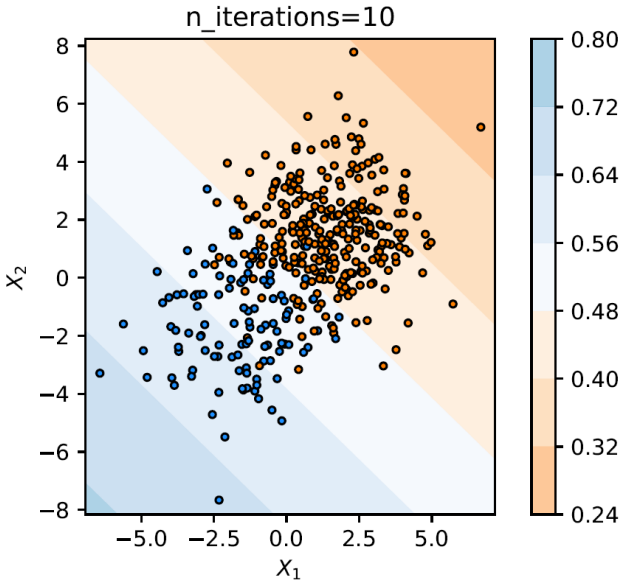

Figure 14: Decision boundary plot of logistic regression models with n_iterations=10 for an unbalanced data set

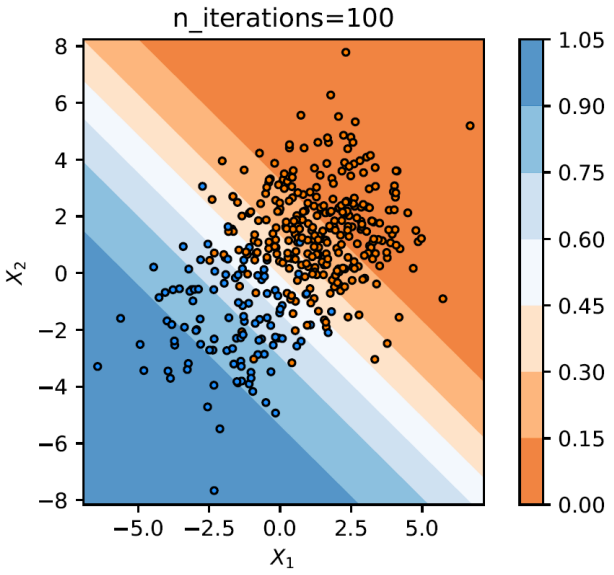

Figure 15: Decision boundary plot of logistic regression models with n_iterations=100 for an unbalanced data set

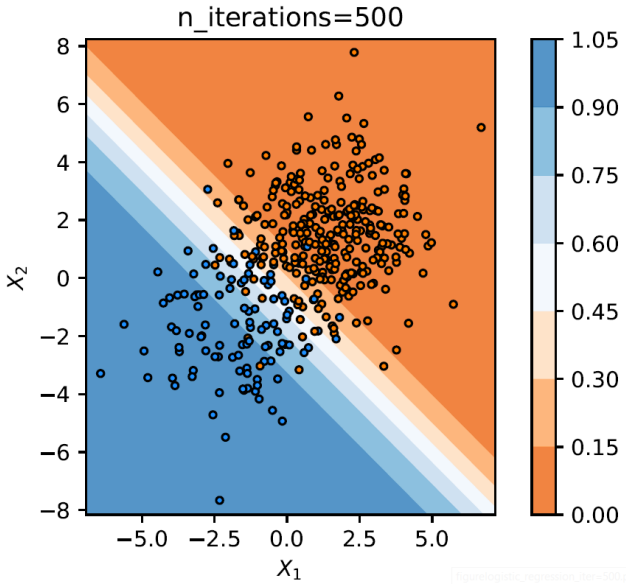

Figure 16: Decision boundary plot of logistic regression models with n_iterations=500 for an unbalanced data set

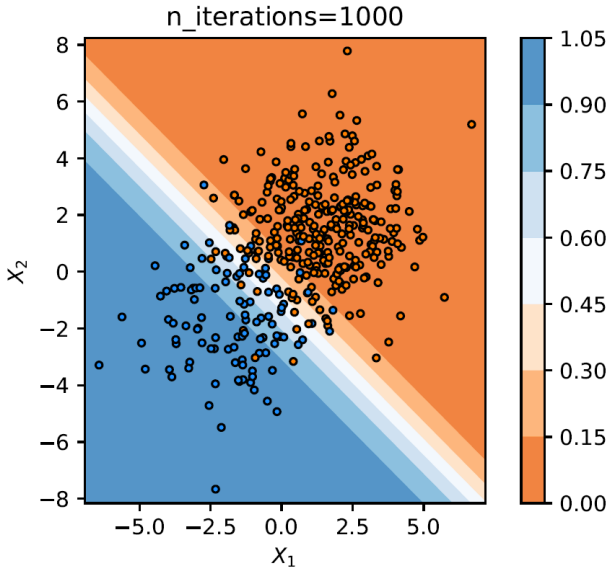

Figure 17: Decision boundary plot of logistic regression models with n_iterations=1000 for an unbalanced data set

3.6 Average test set accuracy

Throughout this section, we will use a learning rate = 0.01 based on scientific resources on the internet [1].
We decided to show our results in a table 3.6 :

Hyper-parameter (n_iterations)	10	100	500	1000
Accuracy average	0.9172	0.9179	0.9204	0.9206
Standard deviation	0.00691	0.0054	0.0049	0.0052

Table 3: Average testing set accuracy over 5 data sets with standard deviation

3.7 The effect of the number of iterations

Throughout this section, we will use a learning rate = 0.01 based on scientific resources on the internet [1].

We observe that when the hyper-parameter increases, the confidence increases and the separation of the two pure regions becomes more and more clear. As the number of iterations increases, we may get closer and closer to the true set of optimal parameters, therefore we get a better and better accuracy. Notice that after a certain number of iterations, the parameters we obtained may start to oscillate around the optimal solution because of the learning rate size. The optimal parameters are indeed located at the minimum of a convex problem.
Ideally, with an infinite power of computation, we would assign the lowest possible value to the learning rate so that we get the most accurate parameters however this is not the case. Therefore, we need to find a learning rate that allows to get to the optimum fast enough but not too large to avoid oscillations between the minimum.
Concerning the error, it decreases when the number of iterations increases until a certain point then stops decreasing.

Looking at our results table and figures, we observe that when the number of iterations is low, we clearly have a case of underfitting. The figure 14 clearly shows that the confidence is low. The table confirms this with a low accuracy average when the hyper parameter is low. It also seems like logistic regression does not allow any overfitting to take place, this is due to the fact that as the number of iterations increases, we get generally closer to the optimal parameters and therefore classify even better.
The best model observed seems the one whose hyper-parameter $n\_iterations = 1000$.

3.8 The best model

As a first answer, one could look at the maximum accuracy for each model and choose the best one considering the average accuracy and the standard deviation. Standard deviation needs to be considered if quite large with respect to the others.
The main problem with that approach is that we have not tested all possible combinations of parameters for each classifier and the optimal values of parameters might belong to a classifier we have not recognized as the best one. Here are the results of the different classifiers for their "optimal" parameters:

	DT(64)	KNN(50)	Log_Reg(0.001,1000)
average accuracy	0.9103	0.9269	0.9206
standard deviation	0.018	x	0.0052

Table 4: Comparisons of different classifier models

We observe that results are pretty similar however we believe that logistic regression is the best model. First, the decision trees splits several times the space along the x or y axis while the logistic regression splits it once but in a linear way which seems to be more appropriate in our case. Second, as we discussed above, we expect the optimal classifier to be a line parallel to the median of the two distributions centers as seen on the figure 18 where the green line would be the optimal classifier and R the distance between the centers. It seems that logistic regression is able to find that line. Indeed, by plotting the the decision boundary for growing number of iterations, we see that the decision boundary gets closer and closer to the green classifier on 18.

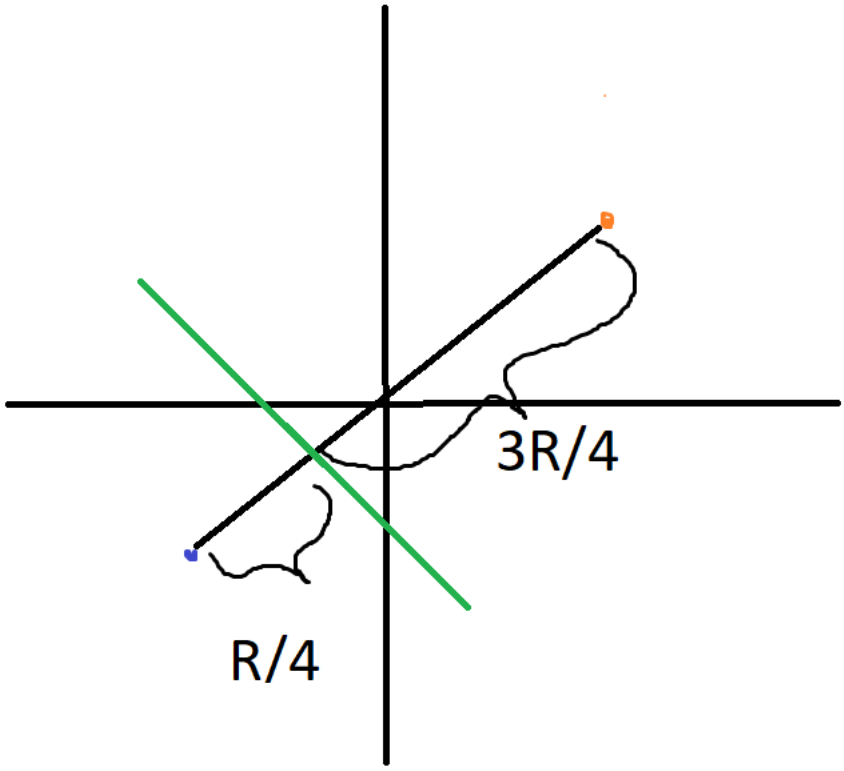

Figure 18: Green optimal classifier

References

[1] Logistic Regression. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.