University of Liège

Introduction to machine learning

ELEN0062

Projet 1 - Classification algorithms

Julien Gustin, Joachim Houyon

October 15, 2021

1 Decision tree

1.1

(a) The *min_ samples_ split* parameter is a parameter that told the model the minimum number of samples required to split an internal node. As an example **Fig. 11** shows the decision tree built with *min_ samples_ split* set to 64 and **Fig. 12** with *min_ samples_ split* set to 500, as we see the lower is *min_ samples_ split* the more the tree will be likely to have more test nodes (and thus a deeper and wider tree). Concerning boundaries a DT. split one variable at a time, this create some rectangle aligned to axis X_1 and X_2, and the deeper (the more test node) is the tree the more section appear as we can see in **Fig. 5** to **10**.

(b) From *min_ samples_ split* set to 500 to 64 the model seems to generalise quite well and do not underfit much, but then it is clearly over-fitting, as an example let see **Fig. 7** we see a thin orange rectangle at $X_1 \approx 32$, at that point the model is too much training dataset-dependent (noise in the training data is picked up and learned), and do not generalise quite well (it is getting worse and worse the lower is the set parameter, we simply need to see at all the small rectangles at the center of the figure).

(c) Prediction confidence in a decision tree is computed by computing the ratio of training instances of class k in the leaf node it ends up. The model seems more confident when *min_ samples_ split* is the smallest because by setting it to 2, during training the decision tree will continue to split *impure* nodes until the impurity test is equal to 0 (i.e. when there are only samples of same class in the node). And this lead to build a tree for which each leaf as a class for which the ratio of samples belonging to this class is 1.

1.2

(a) As explained before a DT. split one variable at a time, and this create some rectangles which are aligned to axis $X1$ and $X2$. And from **Fig. 5** to **10** we can see that decision boundary are composed of rectangles.

(b) Let say that the positive class is blue: We can see that they are way more false negative than false positive samples. (Can also be shown with confusion matrix)

1.3

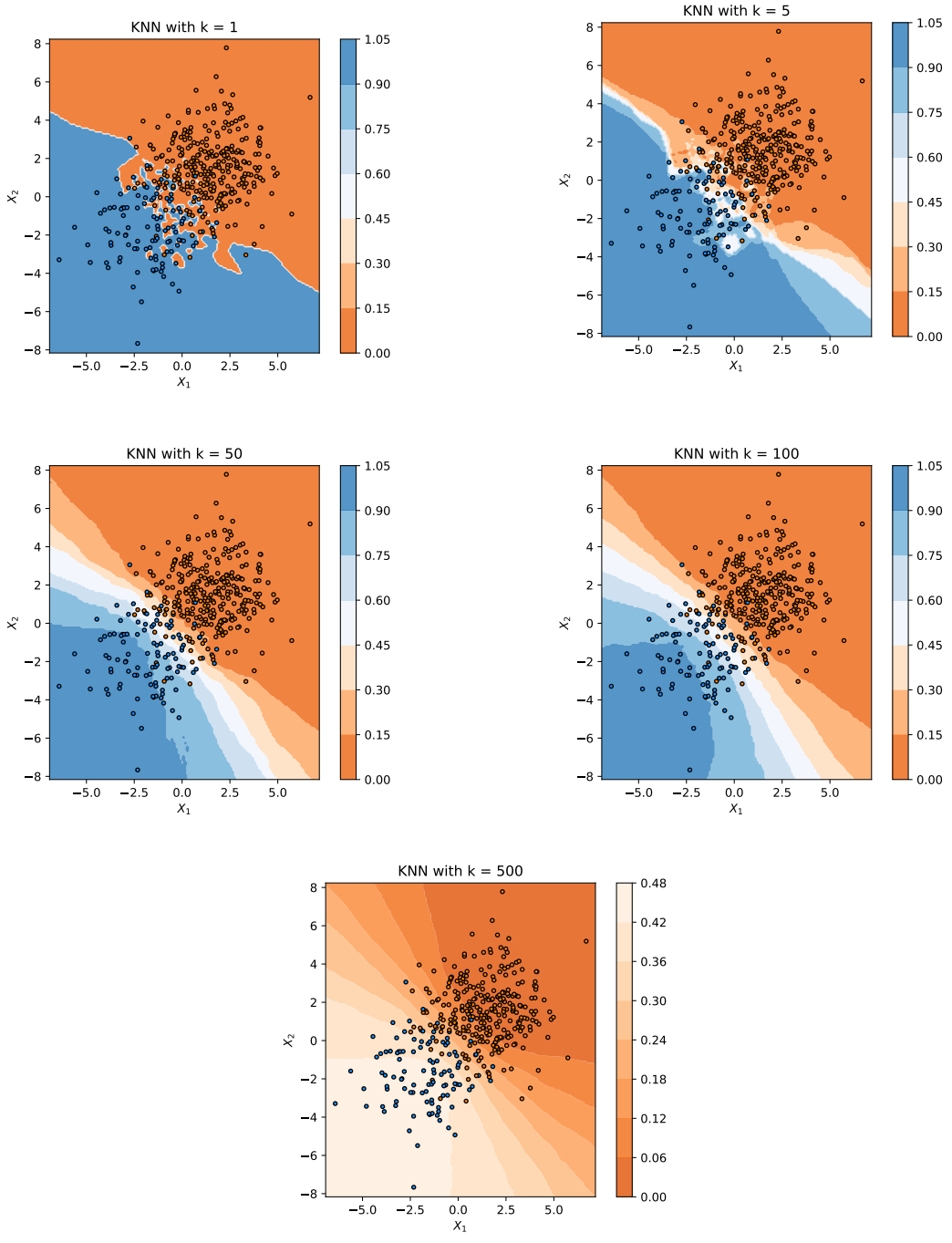
min_samples_split	**mean**	**std**
2	88.23	1.816
8	88.39	1.832
32	90.12	1.005
64	91.03	1.089
128	90.3	1.272
500	86.37	1.402

Table 1: Average test accuracy (in percent) over five generations of the dataset, along with the standard deviations for each values of *min_ samples_ split*

From top to bottom we see that as expected the accuracy keep increasing til *min_ samples_ split* is set to 64, this is due to overfitting as explained in **1.1.(b)**. When this maximum is reached it decrease because the model start underfitting

2 K-nearest neighbors

2.1



(a)

(b)

- **k=1:** The model is very confident. There is a bit of overfitting because the assignment is only done by one neighbor.
- **k=5:** There is a bit of uncertainty, the uncertainty area forms a sort of diagonal line between the left and right edges of the plan. The overall classification remains great.

- **k=50, 100:** The uncertainty area forms a curve and becomes larger as k grows, shrinking the blue area into the bottom left corner of the plan.
- **k=500**: The model uses half of the points in order to make its prediction and +-75% of them are from one class. Therefore, all predictions are very likely to output the dominant class. The overall classification is not good.

2.2

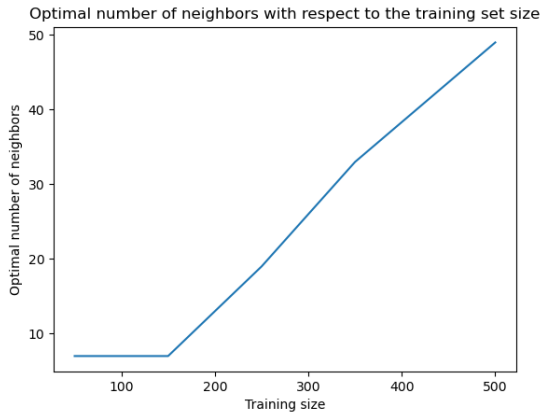
(a) For this, we use the `cross_val_score()` function of scikit-learn. We precise that we want to do a cross validation over 10 folds and we give the classifier to train. The function will do the following for each fold: The KNN is trained using 9 folds as training data and the score is calculated with the last fold, used as a test set. The function returns the score of each fold.

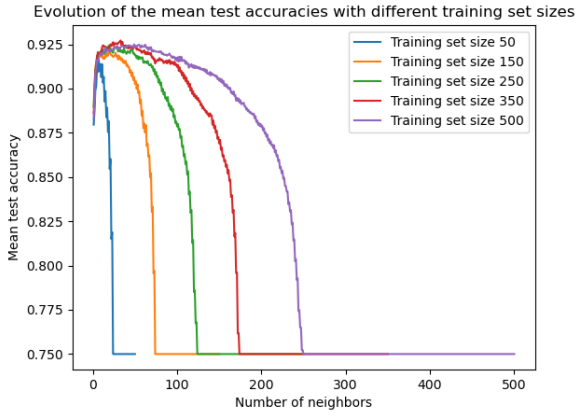
(b) The optimal value of k is 50, with a mean accuracy of 92.6%. However, the ten-fold cross validation shows that a value of k between 5 and 500 is also optimal. Reguarding k = 1, 5, 50 and 100, they corroborate our decision boundary-based intuitions, the classification is good. However, k = 500 looks surprising. The difference resides in the size of the training set: Indeed, the ten-fold cross validation is done with the whole dataset. Therefore, the training set size is 2700 instead of 1000. We do not take half of the points to make our prediction, so the overall classification should not always lead to the most dominant class. The optimal number of neighbors depends on the training set size.

k	**mean score**
1	89.8%
5	92%
50	92.7%
100	92.5%
500	92.2%

Table 2: CV mean score for different values of k

2.3





When we look at the evolution of mean test accuracies with different training set size, we see that when the training set size grows, the variability of an optimal value k grows. In general, for a training set size of N and an optimal value of k in the interval [1, M], when N grows, M grows. $1 \leq M < N$. We also see that as we have an unbalanced data set with 75% of orange class and 25% of blue after a certain number of neighbors given a training size we have a common agreement for which each new sample is classified as class orange therefore we reach a plateau of accuracy of 75% (rate of orange class).

3 Logistic regression

Lets define some notation $\boldsymbol{\theta} = (w_0, \boldsymbol{w^T})$ are the trainable parameters. And $\boldsymbol{x'_i} = (1, x_i^1, x_i^2)^T$ is the ith feature vector to which a constant 1 is used for the bias

3.1

We know that if $\frac{1}{1+exp(-\theta^T x'_i)} >= 0.5$ then the logistic classifier classifies sample 'i' as being in class **1**. And we want to prove that the decision boundary of a logistic regression model is linear.

$$\frac{1}{1+exp(-\theta^T x'_i)} \geq 0.5 \tag{1}$$

$$\Leftrightarrow 1 \geq 0.5 + 0.5 \times exp(-\theta^T x'_i) \tag{2}$$

$$\Leftrightarrow 1 \geq exp(-\theta^T x'_i) \tag{3}$$

$$\Leftrightarrow 0 \geq -\theta^T x'_i \tag{4}$$

$$\Leftrightarrow 0 \leq \theta^T x'_i \tag{5}$$

This prove that the classification depend of a linear combination of feature and parameters (plus a bias), because sample i is classified as 1 iff: $0 \leq \theta^T x'_i$.

3.2

We wish to maximize

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

Maximizing the likelihood is the same as maximizing the log-likelihood:

$$\underset{\theta}{\text{argmax}}\ L(\theta) = \underset{\theta}{\text{argmax}}\ log(L(\theta))$$

We get the following:

$$\underset{\theta}{\text{argmax}}\ log(\prod_{i=1}^{N} P(y_i|x_i, \theta)) \tag{6}$$

$$= \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} log(P(y_i|x_i, \theta)) \tag{7}$$

$$= \underset{\theta}{\text{argmax}} \frac{1}{N} \sum_{i=1}^{N} log(P(y_i|x_i, \theta)) \tag{8}$$

$$= \underset{\theta}{\text{argmin}} -\frac{1}{N} \sum_{i=1}^{N} log(P(y_i|x_i, \theta)) \tag{9}$$

We get that maximizing the likelihood is the same as minimizing the negative log-likelihood. (Note for (8) In an optimization problem multiplying the function by a constant does not change the optimal vector)

3.3

We know that a logistic regression estimates the conditional probability of the positive class with the following expression:

$$P(Y = +1|\boldsymbol{x_i'}, \boldsymbol{\theta}) = \frac{1}{1 + exp(-\theta^T x_i')} \tag{10}$$

This can be generalize for the two classes by:

$$P(Y = y_i|\boldsymbol{x_i'}, \boldsymbol{\theta}) = \left(\frac{1}{1 + exp(-\theta^T x_i')}\right)^{y_i} \times \left(1 - \frac{1}{1 + exp(-\theta^T x_i')}\right)^{1-y_i} \tag{11}$$

For which the loss function is defined as the negative log-likelihood as:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} logP(Y = y_i|\boldsymbol{x_i'}, \boldsymbol{\theta}) \tag{12}$$

Which can be re-write using (**11**) as

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} log\left(\left(\frac{1}{1 + exp(-\theta^T x_i')}\right)^{y_i} \times \left(1 - \frac{1}{1 + exp(-\theta^T x_i')}\right)^{1-y_i}\right) \tag{13}$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \times log\left(\underbrace{\frac{1}{1 + exp(-\underbrace{\theta^T x_i'}_{z})}}_{a}\right) + (1 - y_i) \times log\left(\underbrace{1 - \frac{1}{1 + exp(-\underbrace{\theta^T x_i'}_{z})}}_{a}\right)\right) \tag{14}$$

Using the chain rules we can now compute $\nabla_\theta \mathcal{L}(\theta)$ as

$$\nabla_\theta \mathcal{L}(\theta) = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial \theta} \tag{15}$$

Note that as the derivative of a sum of functions is the sum of their derivatives.

$$\frac{\partial \mathcal{L}}{\partial a} = -\frac{1}{N}\sum_{i=1}^{N} \frac{\partial (y_i \times log(a_i) + (1-y_i) \times log(1-a_i))}{\partial a} \tag{16}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} (\frac{y_i}{a_i} + \frac{1-y_i}{1-a_i}) \tag{17}$$

$$\tag{18}$$

$$\frac{\partial a}{\partial z} = \frac{\partial ((1+exp(-z))^{-1})}{\partial z} \tag{19}$$

$$= -(1+exp(-z))^{-2} \times -exp(-z) \tag{20}$$

$$= \frac{exp(-z)}{(1+exp(-z))^2} \tag{21}$$

$$= a \times \frac{exp(-z)+(1-1)}{1+exp(-z)} \tag{22}$$

$$= a \times \left(\frac{\cancel{exp(-z)+1}}{\cancel{exp(-z)+1}} - \frac{1}{1+exp(-z)} \right) \tag{23}$$

$$= a \times (1-a) \tag{24}$$

$$\tag{25}$$

$$\frac{\partial z}{\partial \theta} = \frac{\partial (\theta^T x')}{\partial \theta} \tag{26}$$

$$= x' \tag{27}$$

Now that we know all these partial derivative we can apply **(15)** and this give us:

$$\nabla_\theta \mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \left(\left(\frac{y_i}{a_i} - \frac{1-y_i}{1-a_i} \right) \times (a_i \times (1-a_i)) \times x'_i \right) \tag{28}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} (y_i \times (1-a_i) - (1-y_i) \times a_i) \times x'_i \tag{29}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} (y_i - \cancel{a_i y_i} - a_i + \cancel{a_i y_i}) \times x'_i \tag{30}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} (y_i - a_i) \times x'_i \tag{31}$$

$$= \frac{1}{N}\sum_{i=1}^{N} (a_i - y_i) \times x'_i \tag{32}$$

$$\tag{33}$$

And finally by substituting a_i by $P(Y{=}{+}1 \mid \boldsymbol{x'_i}, \boldsymbol{\theta})$ we have:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \left[P(Y=+1|x'_i, \theta) - y_i \right] \times x'_i \qquad \square \tag{34}$$

3.4

From a theoretical point of view, any values for theta are acceptable: the problem is convex and any initial theta will lead towards the global minimum. In the literature initializing to a vector of one is widely used, we have thus done the same.

3.5

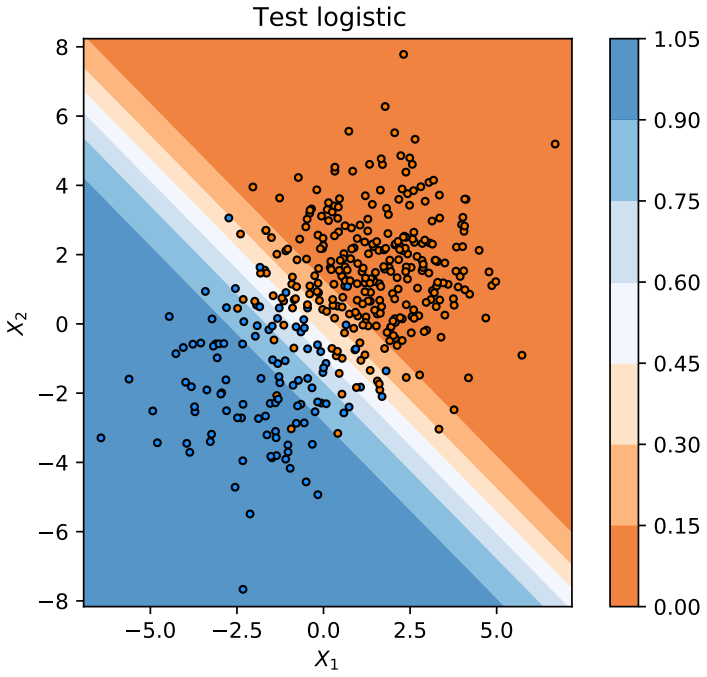


Figure 3: Boundary of a logistic classifier trained on 1000 samples, and plot using 2000 samples of the test set. With *n_iter = 10* and *learning_rate = 1*

We can see that first the boundary is as expected linear. In addition to that, the classification is quite good and the model is very confident, without overfitting: it generalises quite well.

3.6

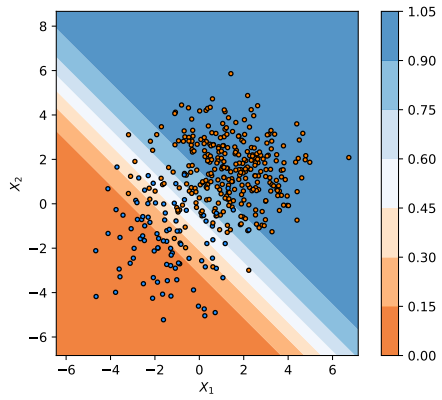
n_iter	LR=0.1	LR=0.5	LR=1	LR=2
5	11.1 (±0.41)	89.21 (±0.59)	91.02 (±0.7)	91.88 (±0.68)
10	60.58 (±3.67)	91.03 (±0.7)	92.1 (±0.59)	92.44 (±0.49)
50	91.02 (±0.69)	92.57 (±0.5)	92.68 (±0.7)	92.72 (±0.81)
150	92.5 (±0.41)	92.73 (±0.81)	92.72 (±0.81)	92.72 (±0.81)

Table 3: The average accuracy (in %) along with its standard deviation (in %). With different value of *learning_rate* (LR) and *n_iter*, over five generations of the dataset

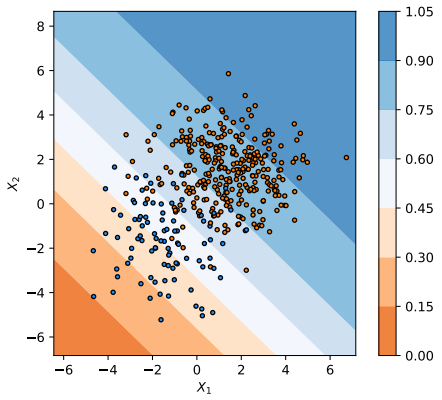
We see that when the learning rate is low, the model converge slowly to a *solution* but at the end it will be more likely to reach a better solution or even the optimal solution (see when *LR = 0.5*) than when the learning rate is high, but however when *LR* is high (*LR=2*) it converge

way more faster to a pretty good solution, it is then a trade-of between these two parameters . The average accuracy is quite good and reach quite fast an accuracy higher than 90%

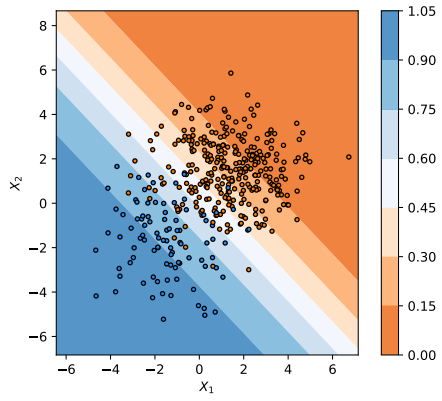
3.7



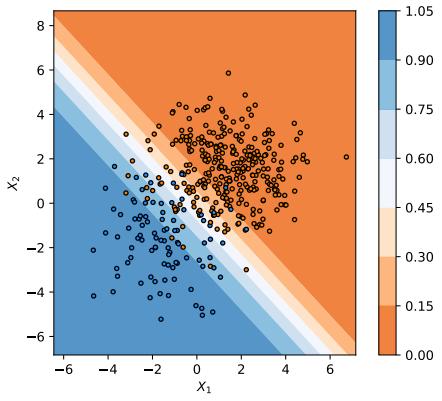
(a) *n_ iter=1*, accuracy=0.073



(b) *n_ iter=5*, accuracy=0.1125



(c) *n_ iter=20* accuracy=0.8785



(d) *n_ iter=100*, accuracy=0.923

Decision boundary of a logistic regression for several iterations with *learning_ rate* set to 0.1

As explained in **3.6** the choice of the number of iteration depend on the learning rate because if the learning rate is low and there are few iterations, the model may not converge to a good solution and depend too much on the initial value of θ (for instance **Fig. 4b** has a really poor result). Using a fixed LR of 0.1, we see that the gradient descent will take some iterations to reach a good solution but the higher is *n_ iter* the likelier the model will be close to the optimal solution. Thus as we can see in the figures, little by little it converge to a good solution, and get better and better accuracy. By looking in the boundary we can also see that it is getting more and more confident and from **Fig. 4b**, split quite well the two epicenter of these two classes ((-1.5, -1.5), (1.5, 1.5)).

3.8

Concerning accuracy, KNN and logistic regression have quite similar result the DT is a bit less precise.

In terms of boundaries, DT doesn't look very appropriate because, intuitively, we would divide our two set of classes by a diagonal line like in the Logistic Regression. Therefore, DT would not be a good candidate in order to fit this model. Indeed the decision tree can only split along X1 and X2.

Regarding KNN and Logistic regression, the Logistic regression looks more appropriate to our problem. Indeed, the problem is unbalanced and KNN is likelier to choose orange in the boundaries. When it comes to Logistic regression it looks like the more natural one to split these two classes

4 Appendix

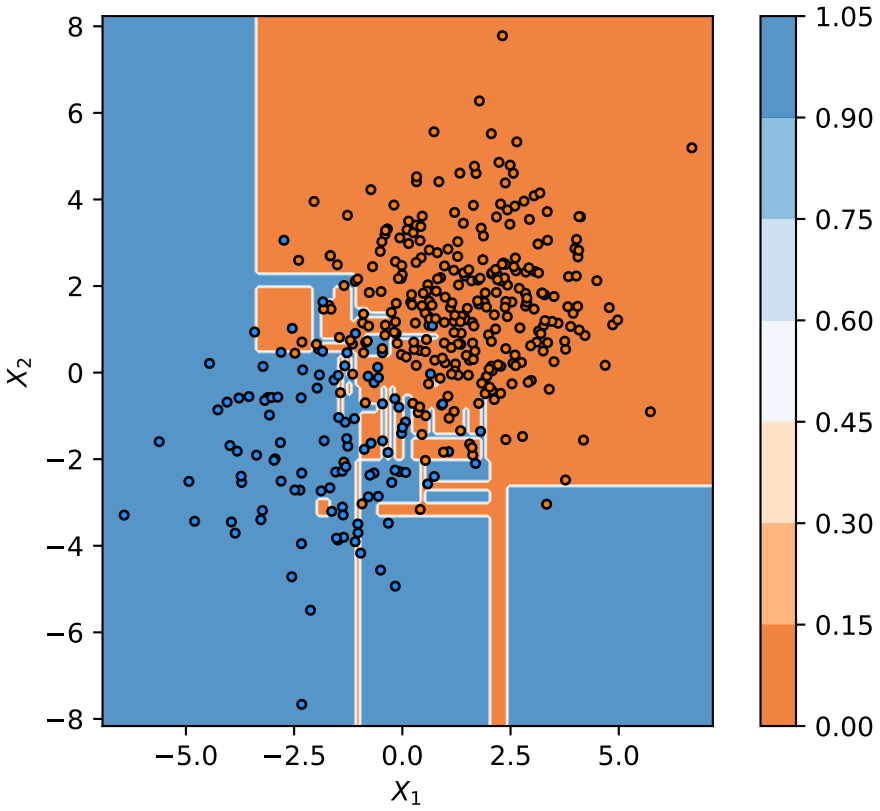


Figure 5: Decision Tree with samples_split set to 2

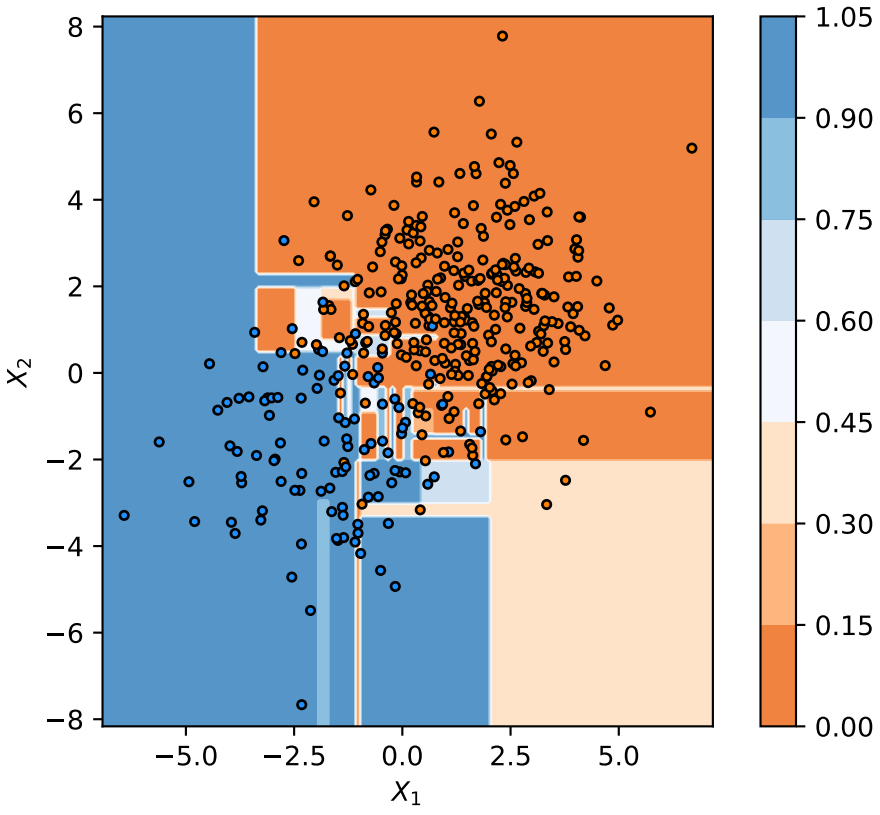


Figure 6: Decision Tree with samples_split set to 8

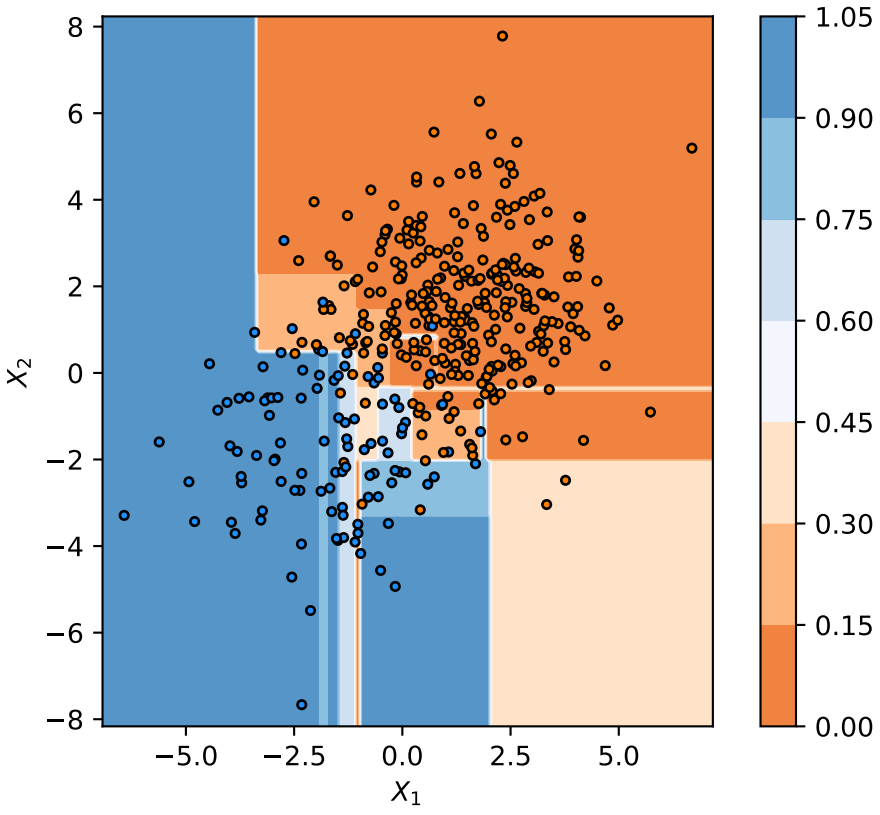


Figure 7: Decision Tree with samples_split set to 32

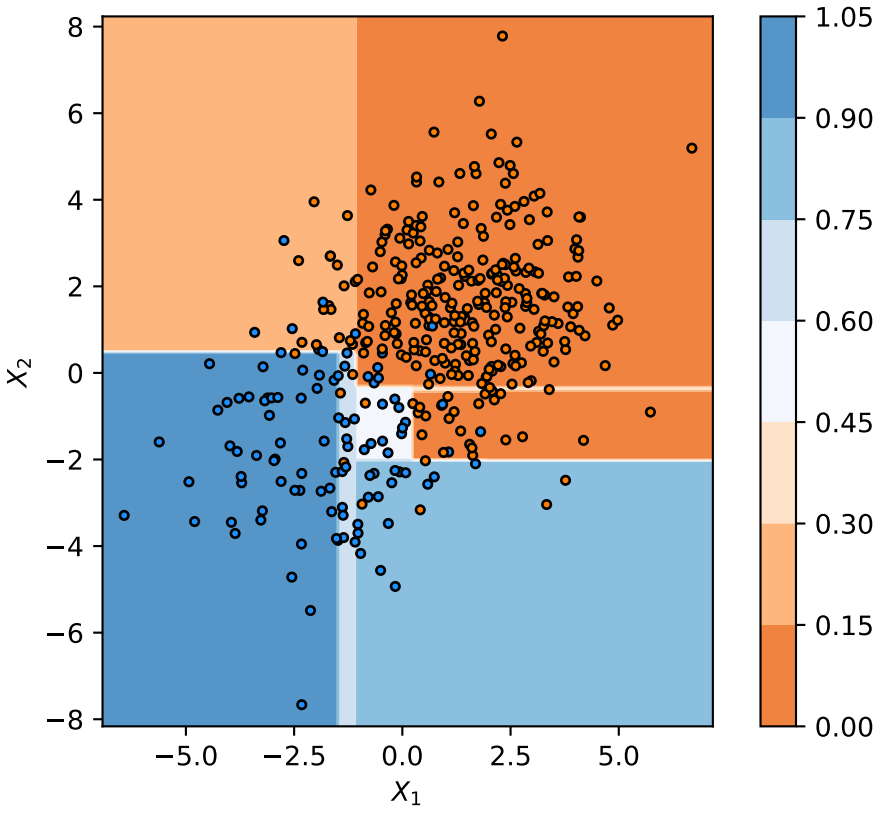


Figure 8: Decision Tree with samples_split set to 64

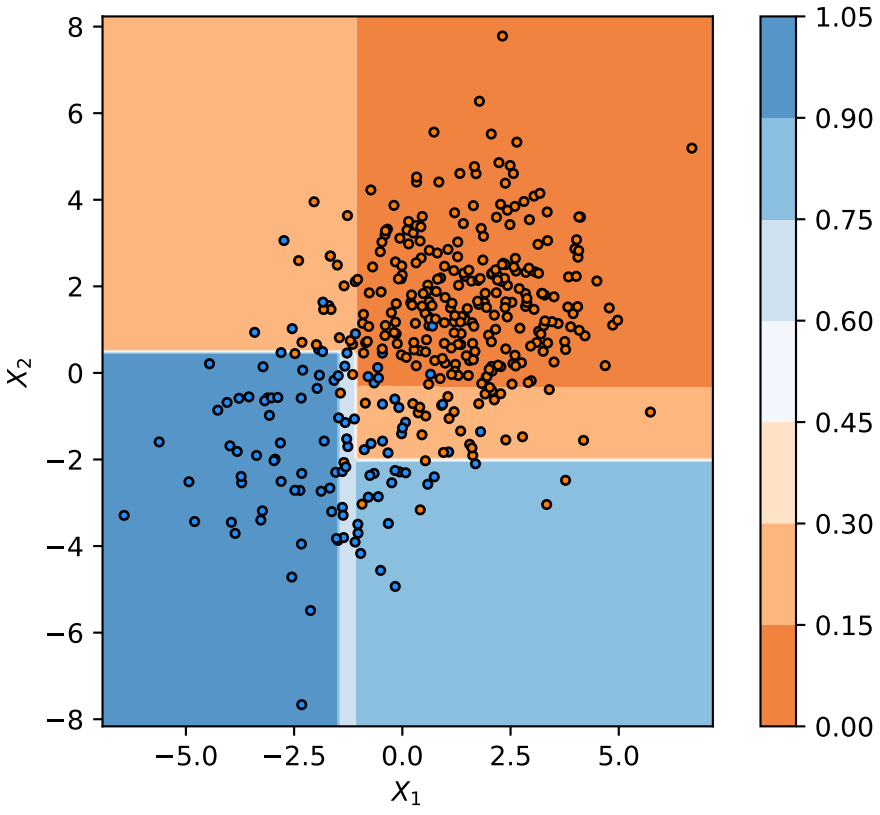


Figure 9: Decision Tree with samples_split set to 128

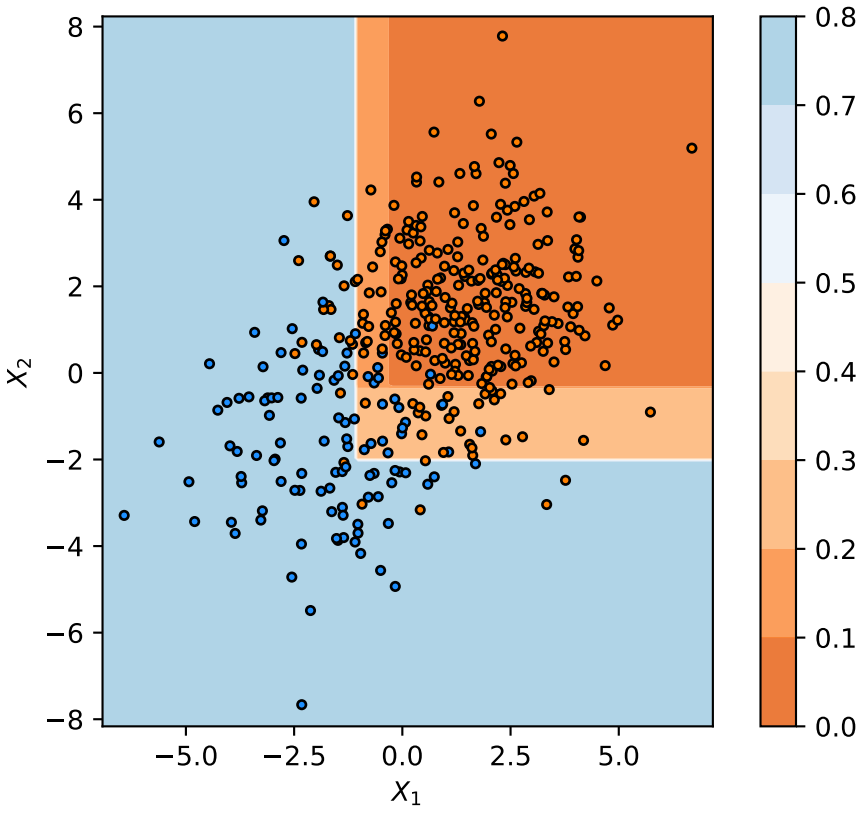


Figure 10: Decision Tree with samples_split set to 500

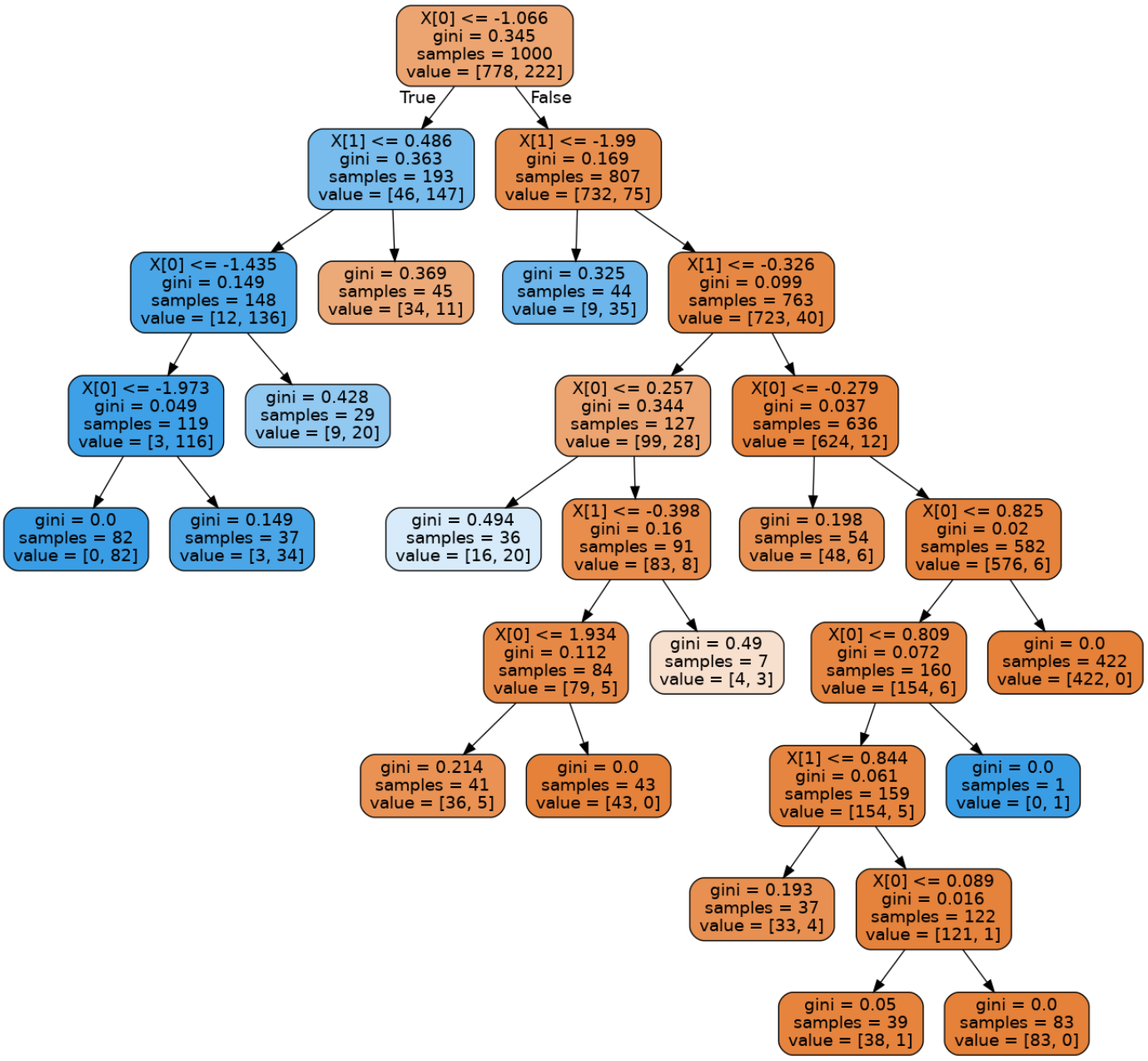


Figure 11: Decision Tree with samples_split set to 64

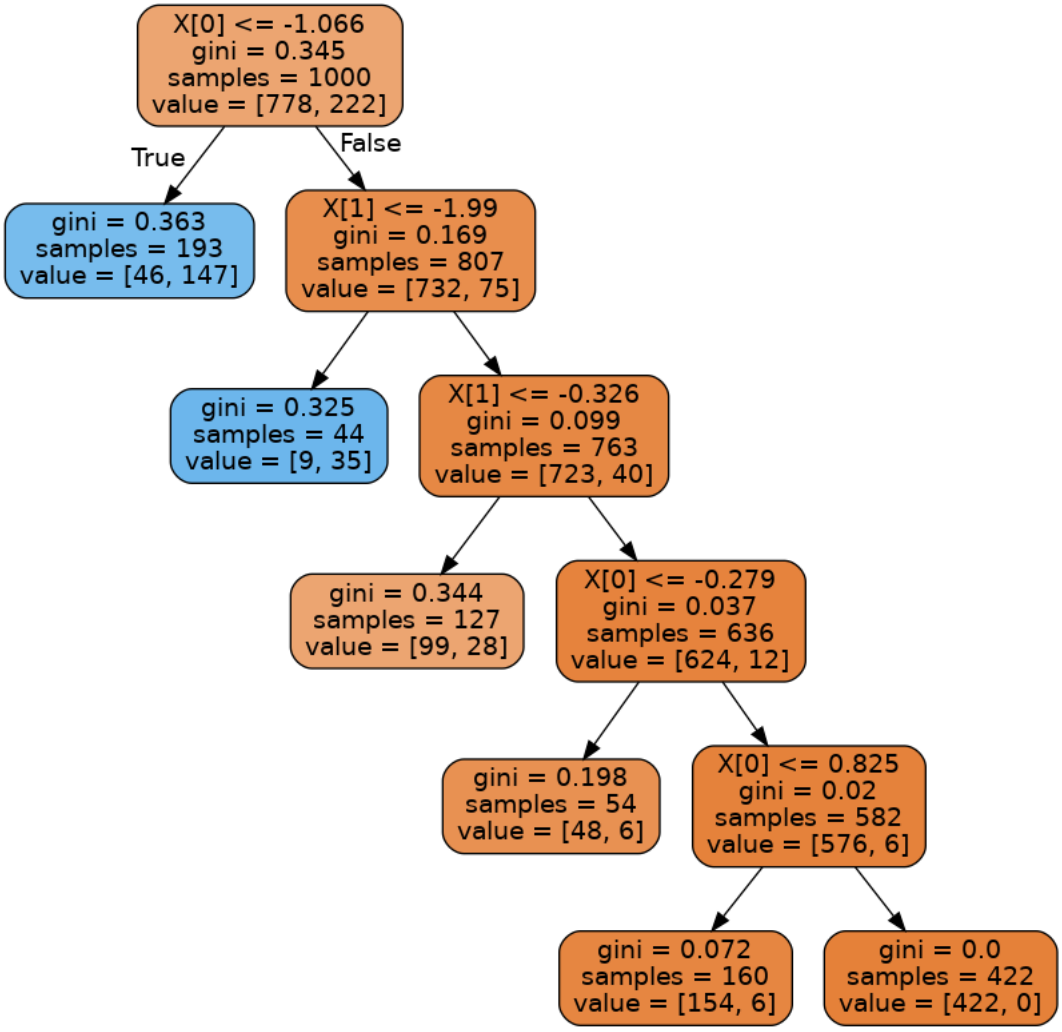


Figure 12: Decision Tree with samples_split set to 500