Project 1 - Classification algorithms

Camille Trinh
S192024

Simon Gardier
S192580

October 27, 2024

1 Decision tree

1.1 Effects of the max depth value on the decision boundary

1.1.1 Decision boundary for every max depth value

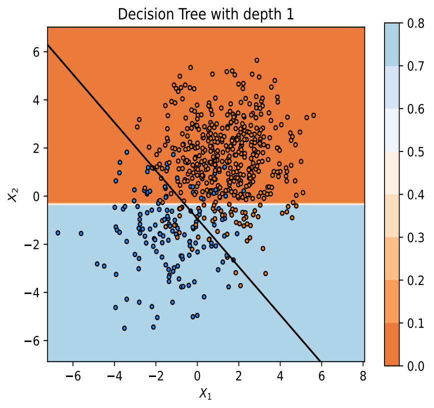

Figure 1: max depth = 1

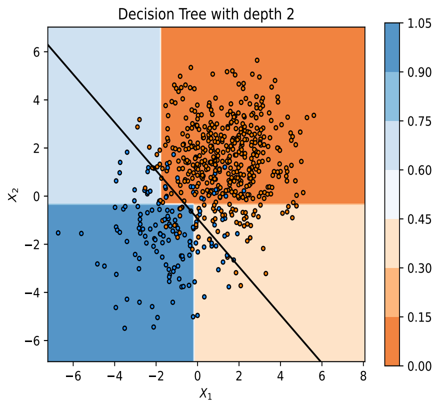

Figure 2: max depth = 2

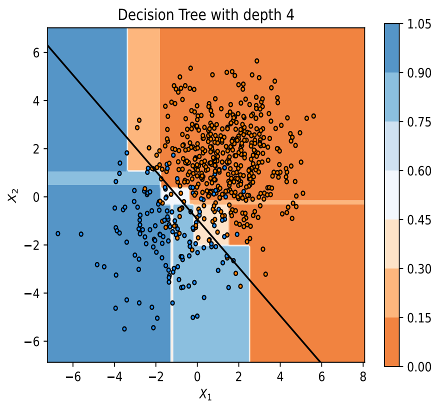

Figure 3: max depth = 4

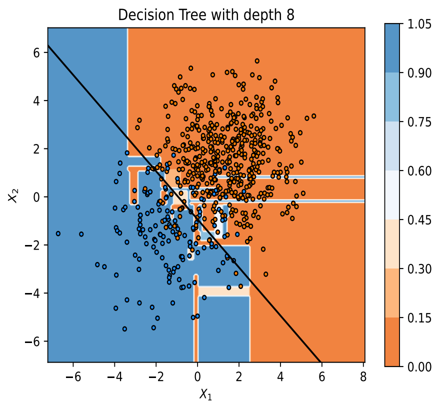

Figure 4: max depth = 8

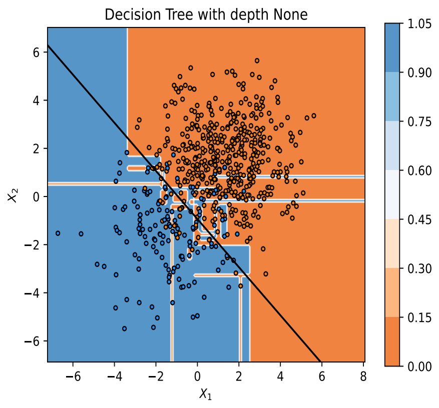

Figure 5: max depth = None

- For max depth = 1, there is only one decision made. So, the split created two regions based on the threshold of, more or less, -0.25 for X_2 and we can conclude that because the split is perpendicular to the X_2 axis.
- For max depth = 2, there is one more split made based on X_2. The model is now more confident around the centers of each of the circular Gaussian distribution.
- For max depth = 4, the model is overall more confident as the nodes starts to become purer with each split, some terminals nodes are reached.
- For max depth = 8, the model starts to fit the exceptions and noise from the training set, all terminal nodes are reached.
- For max depth = None, the model perfectly fits the training set, accentuating overfitting. It is now fully confident in its decisions as the nodes are now pure.

1.1.2 Model underfitting and overfitting

The model is **underfitting** when max_depth is set to 1 and 2 because there are too few splits for the decision boundary to be detailed enough about the data which leads to over simplification. This can be observed in the figures, in the middle of the graph, where there is a lot of blue dots that are misclassified in the red area, where the model is confident that the class is negative.

The model is **overfitting** when max_depth is set to 4, 8 and None. The large number of splits in the training set leads to very specific decision boundaries that perfectly fit the training data, even the noise and small variations. This can be observed in the figures with the isolated lines that go across the other color's region.

1.1.3 Confidence of the model when max depth is the largest

The model is the most confident when max depth is the largest due to the fact that the more the tree splits the data, the more nodes will be pure and with that, the model can closely fit the training set which explains why it is more confident in its accuracy.

1.2 Model evaluation

max_depth	**Accuracy average**	**Accuracy σ**
1	0.8619	0.006778
2	0.9021	0.005544
4	0.9184	0.013105
8	0.8964	0.011664
None	0.8882	0.011998

Table 1: Accuracy and its standard deviation over five generations

From the table, we can observe that as the max_depth increases, the average accuracy also increases and reaches its peak when max_depth = 4. Then, the accuracy average starts to decrease, which is due to the overfitting of the training set, the model becoming too specific. Regarding the standard deviation, it is the smallest when max_depth is set to 2 and 1 which can be explained by the underfitting, since the decision boundaries are still simplistic, the average accuracy won't vary too much.

2 K-nearest neighbors

2.1 Effects of the number of neighbors on the decision boundary

2.1.1 Decision boundary illustration for each number of neighbor

See figures on the next page.

2.1.2 Evolution of the decision boundary with respect to the number of neighbors

- For k = 1, the model overfits as it closely follows the training set data because it can rely only on one neighbor. Also, exceptions and noise have a lot of impact, creating, as consequences, small, isolated regions that does not represent the testing set.
- For k = 5/50/100, the decision boundary starts to smooth out the shape. The high number of neighbors helps to generalize, making the model less affected by exceptions and noise.
- For k = 500, the model underfits as it becomes overly simplified. Especially with our given dataset where the negative class is three times more present than the positive. The decision boundary favors the negative class, which is in majority, and over represents negative predictions.

2.2 Model evaluation

k	**Accuracy average**	**Accuracy σ**
1	0.898	0.008695
5	0.9228	0.00623
50	0.9249	0.00757
100	0.9249	0.006996
500	0.8092	0.04233

Table 2: Accuracy and its standard deviation over five generations

From the table, we can observe that, as mentioned in the previous point, the accuracy average is the lowest for the cases of 1 and 500 due to overfitting and underfitting, respectively. The standard deviation is quite similar for each model except for k = 500 where it is considerably higher. This is because the model is biased towards negative class so the accuracy when encountering positive class is poor, making the accuracy average inconsistent.

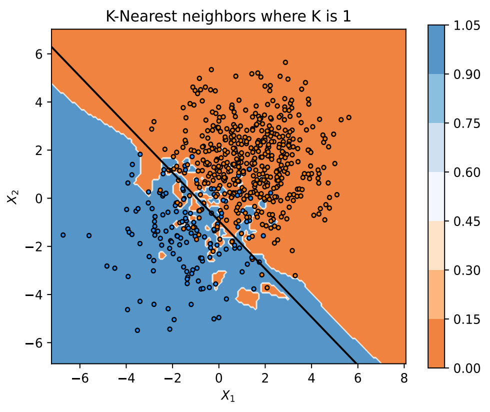

Figure 6: k = 1

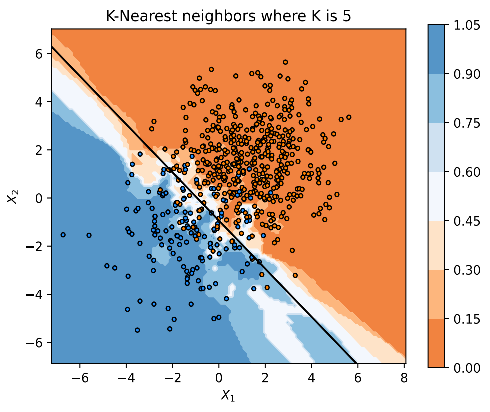

Figure 7: k = 5

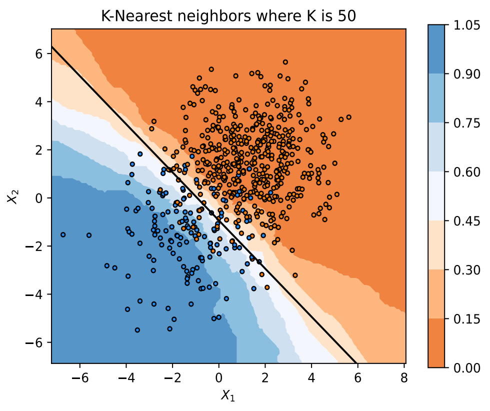

Figure 8: k = 50

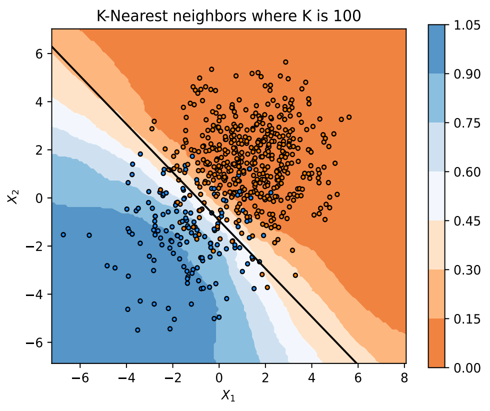

Figure 9: k = 100

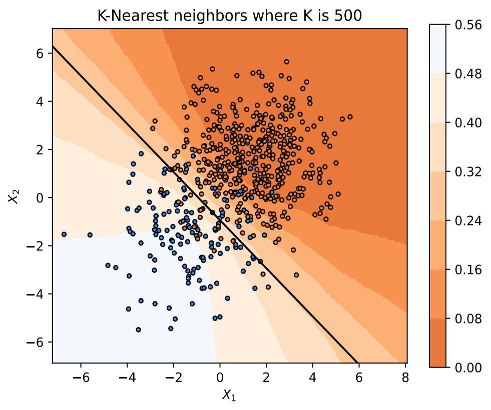

Figure 10: k = 500

3 Perceptron

3.1 Computation of $\nabla_w \mathcal{L}(x, y, w)$

To find the expression of $\nabla_w \mathcal{L}(x, y, w)$, we need to compute the partial derivatives of $\mathcal{L}(x, y, w)$ with respect to each component of the vector of trainable parameters $w = [w_0, w_1, \ldots, w_p]$.
The cross-entropy loss for a single data point (x, y) is given by:

$$\mathcal{L}(x, y, w) = -y \log \hat{f}(x; w) - (1-y) \log(1 - \hat{f}(x; w)) \tag{1}$$

Where $\hat{f}(x; w)$ (the prediction of the perceptron) equal :

$$\sigma(z) = \frac{1}{1 + e^{-z}} \text{ with } z = w_0 + \sum_{j=1}^{p} w_j x_j \tag{2}$$

We want to compute:

$$\nabla_w \mathcal{L}(x, y, w) = \left[\frac{\partial \mathcal{L}(x, y, w)}{\partial w_0}, \ldots, \frac{\partial \mathcal{L}(x, y, w)}{\partial w_p} \right] \tag{3}$$

By applying the chain rule, we first compute $\frac{\partial \mathcal{L}(x,y,w)}{\partial \hat{f}(x;w)}$:

- For the first term $-y \log \hat{f}(x; w)$ we have :

$$\frac{d}{d\hat{f}(x; w)} - y \log \hat{f}(x; w) = -\frac{y}{\hat{f}(x; w)} \tag{4}$$

- For the second term $-(1-y) \log(1 - \hat{f}(x; w)))$ we have:

$$\frac{d}{d\hat{f}(x; w)} \log(1 - \hat{f}(x; w)) = \frac{1}{1 - \hat{f}(x; w)} \times (-1) = \frac{-1}{1 - \hat{f}(x; w)} \tag{5}$$

- Multiplying by $-(1-y)$ gives :

$$-(1-y) \times \frac{-1}{1 - \hat{f}(x; w)} = \frac{1-y}{1 - \hat{f}(x; w)} \tag{6}$$

- We combine the two terms :

$$\frac{\partial \mathcal{L}(x, y, w)}{\partial \hat{f}(x; w)} = -\frac{y}{\hat{f}(x; w)} + \frac{1-y}{1 - \hat{f}(x; w)} \tag{7}$$

Now we compute $\frac{\partial \hat{f}(x;w)}{\partial z}$:

- We have :

$$\begin{aligned} \frac{\partial \hat{f}(x; w)}{\partial z} &= \frac{d}{dz}\sigma(z) = \frac{d}{dz}(1 + e^{-z})^{-1} \\ &= -(1 + e^{-z})^{-2} \times \frac{\partial}{\partial z}(1 + e^{-z}) \\ &= -(1 + e^{-z})^{-2} \times -e^{-z} \\ &= \frac{e^{-z}}{(1 + e^{-z})^2} \end{aligned} \tag{8}$$

From the following equality : $1 - \sigma(z) = \frac{e^{-z}}{(1+e^{-z})}$:
- We found :

$$\begin{aligned}
\frac{d\hat{f}(x;w)}{dz} &= \frac{d\sigma(z)}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \sigma(z)(1-\sigma(z)) \\
&= \hat{f}(x;w)(1-\hat{f}(x;w))
\end{aligned} \tag{9}$$

Now we compute $\frac{\partial z}{\partial w_j}$:

$$\begin{aligned}
&z = w_0 + \sum_{j=1}^{n} w_j x_j \\
&\frac{\partial z}{\partial w_0} = 1 \\
&\frac{\partial z}{\partial w_j} = x_j \text{ for } j \geq 1
\end{aligned} \tag{10}$$

Finally, we compute the partial derivative of $\mathcal{L}$ for w$_j$:

$$\begin{aligned}
\frac{\partial \mathcal{L}(x,y,w)}{\partial w_j} &= \frac{\partial \mathcal{L}(x,y,w)}{\hat{f}(x;w)} \times \frac{\partial \hat{f}(x;w)}{\partial z} \times \frac{\partial z}{\partial w_j} \\
&= (-\frac{y}{\hat{f}(x;w)} + \frac{1-y}{1-\hat{f}(x;w)}) \times \hat{f}(x;w)(1-\hat{f}(x;w)) \times x_j \\
&\text{Distributing } \hat{f}(x;w)(1-\hat{f}(x;w)) \text{ to the first term gives :} \\
&= (-\frac{y}{\hat{f}(x;w)} + \frac{1-y}{1-\hat{f}(x;w)}) \times x_j \\
&\text{Simplifying the first term gives :} \\
&= (\hat{f}(x;w) - y)x_j
\end{aligned} \tag{11}$$

That gives the expression for the gradient of $\mathcal{L}$:

$$\nabla_w \mathcal{L}(x,y,w) = (\hat{f}(x;w) - y)\,[1, x_1, ..., x_p] \tag{12}$$

3.2 Implementation

The implementation follows the provided template, one function was added to the template : `sigmoid(z)` that returns the value of $\sigma(z)$. The value of z is clipped in [-200; 200] to avoid overflow.

The weights are initialized with 0's.

The samples are traveled in random order at each epoch. This approach prevents the model from overfitting to a particular sequence of data from the training set.

3.3 Effects of the learning rate value on the decision boundary

3.3.1 Decision boundary for every learning rate value

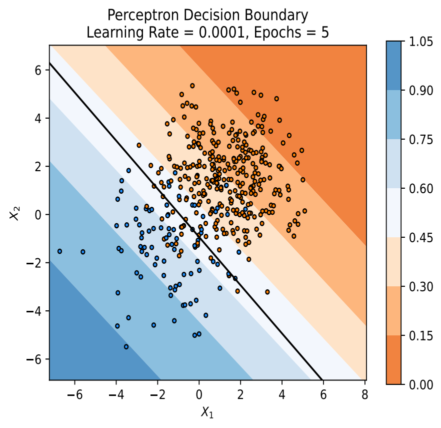

Figure 11: $\eta = 0.0001$

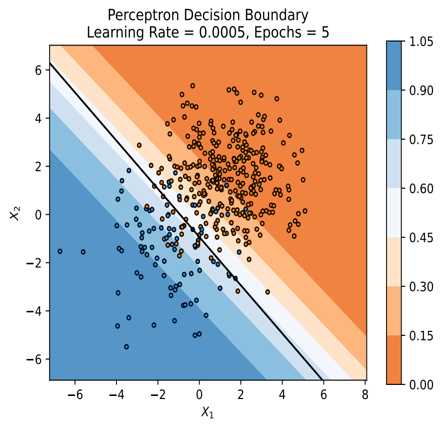

Figure 12: $\eta = 0.0005$

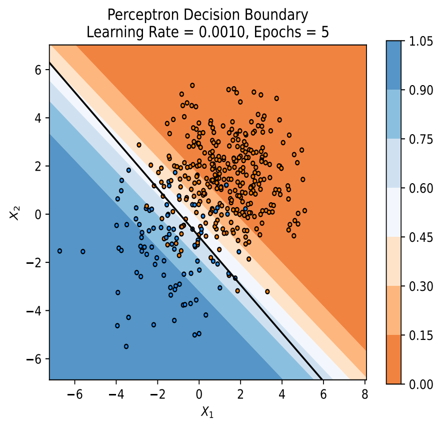

Figure 13: $\eta = 0.0010$

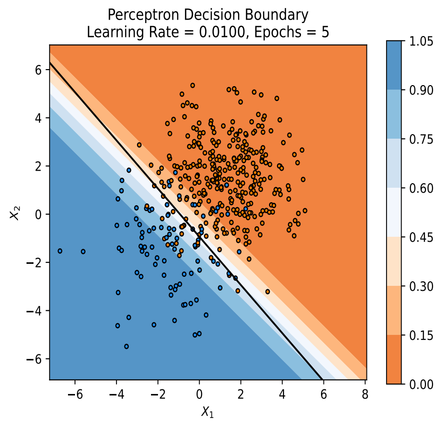

Figure 14: $\eta = 0.0100$

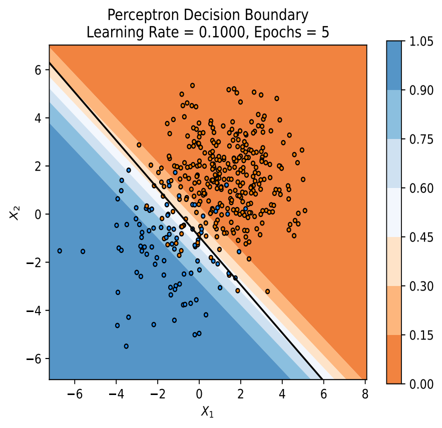

Figure 15: $\eta = 0.1000$

There is a clear improvement in classification between the model trained using $\eta = 10^{-4}$ compared to the model using $\eta = 10^{-1}$. Five epochs is enough to converge to a solution with a high accuracy.

3.4 Model evaluation

η	**Accuracy average**	**Accuracy σ**
$\eta = 10^{-4}$	0.9193	0.0036
$\eta = 5 \times 10^{-4}$	0.9201	0.0045
$\eta = 10^{-3}$	0.9225	0.0047
$\eta = 10^{-2}$	0.9232	0.0066
$\eta = 10^{-1}$	0.9206	0.0053

Table 3: Accuracy and its standard deviation over five generations

The results show an accuracy augmenting with respect to η up to 10^{-2} after which it decreases. The standard deviation follows the same trend. This is expected as a small learning rate (e.g. 10^{-4}) means a slow learning thus more stability. On the other side a high learning rate like 10^{-1} sees its standard deviation being reduce from the one of 10^{-1}. This is due to an overfit of the training data, driving the accuracy down when the model classifies the test sets.

4 Method comparison

4.1 Tuning of max_depth, n_neighbors and η

A simple way to tune the values of the hyperparameters is to do K-fold cross-validation. In our experiments we used `GridSearchCV` to produce the results, which uses 5 folds by defaults.

In the procedure, we define a discrete set of possible values for the parameters and we split the **training** set in k (in our case 5) subsets. In our implementation a Stratified KFold is used, this folding technique unsure that each fold keeps the same classes (true/false) distribution.

Finally, for each parameter value, the classifier is trained on k-1 folds and evaluated on the last fold. A final score is given to the model. The model with the best score is kept. This model can then be retrained on the entire training set.

During the tuning, we used the hyper-parameters given in the statement as well as a few other similar values.

Classifier	Parameter range	"Standard" dataset	Noisy dataset
Decision tree	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, None	max_depth = 3	max_depth = 3
K-nn	1, 5, 50, 100, 500	k = 50	k = 50
Perceptron	10^{-4}, 5×10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 5×10^{-1}	$\eta = 10^{-1}$	$\eta = 10^{-4}$

Table 4: Classifiers, the hyper-parameters values used in cross validation and tuning results.

Classifier	$\text{mean}_{\text{accuracy}}$	σ_{accuracy}	$\text{mean}_{\text{accuracy noisy}}$	$\sigma_{\text{accuracy noisy}}$
Decision tree	0.9133	0.00714	0.9019	0.01134
K-nn	0.9235	0.00713	0.7581	0.00835
Perceptron	0.9210	0.00592	0.9053	0.00615

Table 5: Accuracy and its σ for selected parameter over 10 generations of "standard" and "noisy" datasets.

4.2 Tuning results

4.3 Performance discussion and ranking

4.3.1 Non-noisy setting analysis

In our results, knn gave the best accuracy but the perceptron was more consistent in its results. The Decision tree is falling behind both in terms of accuracy and standard deviation.

4.3.2 Noisy setting analysis

In our results, the perceptron seems to beat the decision tree and k-nn on the accuracy and its standard deviation σ shows that it is consistent on its good results.

4.3.3 Overall analysis

We come to the realization that there is no perfect model, the distribution of the data and its quality (noisy or not) should help to determine the model to be used and its relative parameters.
However, for our dataset and settings, the best classifier would be the perceptron because of its accuracy and consistency overall both in the non-noisy and noisy settings. A second choice would be the decision tree since it also keeps a good accuracy and consistency in the noisy setting.

4.4 Discussions

The differences in these two settings rely on the way each classifier handles noises and how the model fits the data :

- For the **Decision tree**, we used max_depth = 3 where the model still underfits the data but since the decision boundary is partitioned by straight lines which creates segments, it is not that affected by noisy data, which explains the consistency.
- For **K-nn**, we used k = 50 where the model is more prone to underfit than overfit however we saw that K-nn is heavily influenced by the majority class and individual points, which is why its accuracy is so low when there are noises.
- For **Perceptron**, we used $\eta = 10^{-1}$ and $\eta = 10^{-4}$. The perceptron is designed to find a linear decision boundary between the two classes. The proportion of negative data being 3 times the one of positive data creates an imbalance helping the model to stay consistent even in presence of noise. Furthermore this noise is perfectly centered and distributed between the two classes in (0; 0) where the best linear separation would be located at an angle of 45 degree.

However, no matter the choice of the classifier, no model would be able to perfectly separate the dataset due to the overlap of the two Gaussian distributions. One of the best model would be a linear one with a 45 degree angle, separating the two Gaussians between their centers, at equal distance from them.