ELEN0062 - Introduction to Machine Learning
Project 1 - Classification Algorithms

PARING MADANASSONO POUEZI (S228407) and DUY VU DINH (S2401627)

1 DECISION TREE

(1.1) Observe how the decision boundary is affected by tree complexity

(a) Illustrate and explain the decision boundary for each hyperparameter value

The required figures for this part are included in Figure 1 in the appendix.

The decision boundary relates to the splitting criteria which is based on which variable/threshold is the best in terms of reducing the impurity in the resulting subsets. When the *max_depth* is 1, there are only two subsets since the split is done only once. With a *max_depth* value of 2, there are four subsets since each of the subsets derived from the first split has been split again. And so on with higher values of *max_depth*, as long as the split stopping criteria are not met.

At $max\_depth = 1$, the graph shows two regions of classification due to a single decision, resulting in a simple model. At $max\_depth = 2$, the separation between the orange and blue regions becomes more defined but remains somewhat rough. With $max\_depth = 4$, the boundary distinction is finer, reflecting improved model performance. However, at $max\_depth = 8$ and $max\_depth = None$, the regions expand and interweave, indicating that the models may be overly tailored to the training data, which can affect their performance on unseen data.

(b) Discuss when the model is clearly underfitting/overfitting and detail our evidence for each claim

Figure 1a relating to a *max_depth* value of 1 shows a blue rectangle where any point will be predicted as a blue class one, and an orange rectangle where any point will be predicted as an orange one. It is obvious that such a model didn't capture enough of the variability of the learning data, since there is a significant amount of actual blue points in the orange rectangle and conversely, there is a significant amount of actual orange points in the blue rectangle. Hence the model is underfitting, which will lead to a high prediction error rate. On the other hand, Figure 1d relating to a *max_depth* value of 8 shows, straight in the middle of the orange points area, some small blue rectangles. New points in such small blue rectangles will be predicted as blue yet they shouldn't. In this case, the model went over the somehow right variability of the learning data and did capture noisy information from the peculiar inputs during the learning process. The model is overfitting, which will lead to an increase in the error rate.

(c) Explain why the model seems more confident when *max_depth* is the largest

When *max_depth* is the largest, all the leaves of the decision tree are pure. The learning process resulted in an orange rectangle dedicated to each peculiar orange point located in the blue points area, and in a blue rectangle dedicated to each peculiar blue point located in the orange points area. Thus, there is no learning error. Thus, there is no learning error due to excessive overfitting.

(1.2) Report the average test set accuracies over five generations of the dataset, along with the standard deviations, for each value of *max_depth*

A *max_depth* of 4 is the best degree of complexity since on average, we have the highest accuracy on unseen data along with a reasonable standard deviation (Table 1 and Figure 1): neither the model is underfitting nor

Table 1. Average accuracy and its standard deviation of decision tree classifiers with different maximum depths.

Max. Depth	Train Accuracy		Test Accuracy	
	Average	Std. Dev.	Average	Std. Dev.
1	0.8732	0.0079	0.8498	0.0078
2	0.9212	0.0050	0.8985	0.0065
4	0.9393	0.0063	0.9112	0.0046
8	0.9796	0.0044	0.8924	0.0072
None	1.000	0.0000	0.8868	0.0041

overfitting. The accuracy trend shows that as max_depth increases, train accuracy rises consistently, while test accuracy peaks at $max\_depth = 4$ before declining, suggesting underfitting at lower depths and overfitting at higher depths. The standard deviations are low across all max_depth values, indicating consistent model performance, but a drop in test accuracy and stability beyond $max\_depth = 4$ reinforces the concern of overfitting.

2 K-NEAREST NEIGHBORS

(1.1) Observe how the decision boundary is affected by the number of neighbors

(a) Illustrate the decision boundary for each value of *n_neighbors*
The required figures for this part are included in Figure 2 in the appendix.

(b) Comment on the evolution of the decision boundary with respect to the number of neighbors
In the learning process, when *n_neighbors* value is 1, the model seems very confident, as shown by Figure 2a. Each peculiar orange point located in the blue area is contained in an orange boundary, and each peculiar blue point located in the orange area is contained in a blue boundary. The model is clearly overfitting and there is no learning error. With increasing values of *n_neighbors*, the model is less and less confident, especially in the areas where there is a mixed presence of orange and blue points, as shown by Figure 2b relating to a *n_neighbors* value of 5. For a very high value for *n_neighbors*, the model favors the most represented class in the learning data. As shown by Figure 2e, looking for the 500 nearest neighbors of any input will in most cases encompass a lot of orange points. The model is clearly underfitting.

(1.2) Report the average test set accuracies over five generations of the dataset, along with the standard deviations, for each value of *n_neighbors*

Table 2. Average accuracy and its standard deviation of K-nearest neighbors classifiers with different numbers of neighbors.

Number of Neighbors	Train Accuracy		Test Accuracy	
	Average	Std. Dev.	Average	Std. Dev.
1	1.0000	0.0000	0.8841	0.0073
5	0.9380	0.0054	0.9126	0.0075
50	0.9282	0.0084	0.9202	0.0052
100	0.9250	0.0084	0.9205	0.0063
500	0.8130	0.0491	0.8121	0.0616

A *n_neighbors* of 50 is the best degree of complexity since on average, we have the runner-up highest accuracy on unseen data along with the smallest standard deviation (Table 2 and Figure 4b): neither the model is underfitting nor overfitting. As *n_neighbors* increases, train accuracy may initially decrease due to reduced sensitivity to noise, while test accuracy typically rises to a peak at *n_neighbors* = 50 before declining, illustrating the bias-variance tradeoff: lower values can lead to overfitting, while very high values may cause underfitting. The standard deviations are consistently low (below 0.01), suggesting reliable performance across different dataset generations, with *n_neighbors* = 1 showing complete overfitting and a standard deviation of 0.

3 PERCEPTRON

(3.1) Derive the mathematical expression of the gradient $\nabla_w L(x, y, w)$ for implementing the stochastic gradient descent algorithm

A perceptron predicts using:

$$\hat{f}(\mathbf{x}; \mathbf{w}) = \sigma\left(w_0 + \sum_{j=1}^{p} w_j x_j\right), \tag{1}$$

where $\mathbf{x} = (1, x_1, ..., x_p)$ denotes the p input variables, $\mathbf{w} = [w_0, w_1, ..., w_p] \in \mathbb{R}^{p+1}$ is the vector of trainable parameters, and $\sigma(z) = \frac{1}{1+e^{-z}}$. This can be expressed as:

$$\hat{f}(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}. \tag{2}$$

The gradient with respect to $\mathbf{w}$ is:

$$\frac{\partial \hat{f}(\mathbf{x}; \mathbf{w})}{\partial w} = \hat{f}(\mathbf{x}; \mathbf{w})(1 - \hat{f}(\mathbf{x}; \mathbf{w})) \cdot \mathbf{x}. \tag{3}$$

Using the cross-entropy loss:

$$L(\mathbf{x}, y, \mathbf{w}) = -y \log(\hat{f}(\mathbf{x}; \mathbf{w})) - (1 - y) \log(1 - \hat{f}(\mathbf{x}; \mathbf{w})), \tag{4}$$

the gradient is:

$$\nabla_w L(\mathbf{x}, y, \mathbf{w}) = -\left(y \cdot \frac{1}{\hat{f}(\mathbf{x}; \mathbf{w})} \cdot \frac{\partial \hat{f}(\mathbf{x}; \mathbf{w})}{\partial w} + (1 - y) \cdot \frac{1}{1 - \hat{f}(\mathbf{x}; \mathbf{w})} \cdot \frac{\partial (1 - \hat{f}(\mathbf{x}; \mathbf{w}))}{\partial w}\right) \tag{5}$$

$$= -y(1 - \hat{f}(\mathbf{x}; \mathbf{w}))\mathbf{x} + (1 - y)\hat{f}(\mathbf{x}; \mathbf{w})\mathbf{x}. \tag{6}$$

This simplifies to:

$$\nabla_w L(\mathbf{x}, y, \mathbf{w}) = (\hat{f}(\mathbf{x}; \mathbf{w}) - y) \cdot \mathbf{x}. \tag{7}$$

(3.2) Explain and motivate our implementation choices

The `PerceptronClassifier` is a binary classification model that adheres to scikit-learn conventions. It initializes weights to zero, including a bias term, enabling feature importance to be learned from the data. Using stochastic gradient descent, the model updates weights after each training sample, while predictions are made with a sigmoid activation function, allowing for classification decisions at a 0.5 threshold. This approach combines simplicity and effectiveness, making it suitable for binary classification tasks.

(3.3) The evolution of the decision boundary with respect to *learning_rate* η

(a) Decision Boundaries for Different Learning Rates

The required figures for this part are included in Figure 3 in the appendix.

(b) Comment on the evolution of the decision boundary with respect to learning rate η
Low Learning Rates ($\eta = 1 \times 10^{-4}$, 5×10^{-4}, and $\eta = 1 \times 10^{-3}$): Figure 3c-3e the decision boundary changes minimally after 5 epochs, leading to underfitting and poor class prediction due to slow convergence.
Moderate Learning Rate ($\eta = 1 \times 10^{-2}$): These rates yields a sharper, stable boundary that effectively captures class separation after just 5 epochs, balancing efficient convergence with stability.
Higher Learning Rates ($\eta = 1 \times 10^{-1}$): High rates cause significant fluctuations in the decision boundary, leading to instability and poor prediction due to excessive weight adjustments that overshoot optimal alignment.

(3.4) Report the average test set accuracies over five generations:

Table 3. Average accuracy and its standard deviation of perceptron classifiers with different learning rates.

Learning Rate, η	Train Accuracy		Test Accuracy	
	Average	Std. Dev.	Average	Std. Dev.
1×10^{-1}	0.9266	0.0042	0.9204	0.0067
1×10^{-2}	0.9278	0.0033	0.9233	0.0060
1×10^{-3}	0.9278	0.0024	0.9207	0.0044
5×10^{-4}	0.9268	0.0059	0.9191	0.0048
1×10^{-4}	0.9220	0.0076	0.9168	0.0060

Table 3 and Figure 4c show that accuracy improves with increasing learning rates, peaking at $\eta = 1 \times 10^{-2}$ for both training (0.9233) and test sets (0.9278). However, at the highest learning rate $\eta = 1 \times 10^{-1}$, accuracy declines, suggesting potential overfitting or instability. The standard deviations are relatively low, indicating consistent performance across runs. Notably, the standard deviation of test accuracies decreases with higher learning rates, reflecting greater stability. Given that the number of epochs was fixed at 5, lower learning rates may not have allowed sufficient training for optimal performance. Overall, the results suggest that moderate learning rates ($\eta = 1 \times 10^{-2}$) enhance model performance while maintaining stability.

4 METHOD COMPARISON

(4.1) Explain our way to tune the value of *max_depth*, *n_neighbors*, and `learning_rate` η
To tune the hyperparameters, K-fold cross-validation will be used with $k = 10$ due to our small training dataset (1000 samples). The models will be evaluated based on accuracy as the scoring metric. We will split the training dataset into 10 folds, using 9 folds for training and 1 for validation in each iteration. This process will allow us to assess the performance of each model across different subsets of the training data. Once the optimal hyperparameters are found, we will retrain the models on the entire training dataset using these values. Finally, the trained models will be used to make predictions on the test set to evaluate their performance on unseen data (test set).
For hyperparameter tuning in Python, we will utilize `GridSearchCV` from `sklearn.model_selection`. The parameter grids will include: *max_depth*, *n_neighbors*, and *learning_rate* corresponding to Decision Tree, K-nearest Neighbors, and Perceptron classifier. This systematic exploration will identify the best hyperparameters based on cross-validated accuracy.

(4.2) Implement the above method and use it to compute the average test set accuracies and its standard deviation over at least five generations of dataset

Table 4. Average accuracy and its standard deviation of different classifiers with tuned hyperparameters and $n\_inrrelevant = 0$.

Classifier	Tuned Hyperparameter[1]	Train Accuracy		Test Accuracy	
		Average	Std. Dev.	Average	Std. Dev.
Decision Tree	$\{4, 2, , 2, 4, 8\}$	0.9412	0.0255	0.9047	0.0115
K-nearest Neighbors	$\{50, 50, 5, 5, 50\}$	0.9344	0.0094	0.9158	0.0098
Perceptron	$\{10^{-3}, 10^{-2}, 10^{-3}, 10^{-2}, 10^{-1}\}$	0.9306	0.0044	0.9207	0.0052

(4.3) Implement the same method but add 200 new input features to the dataset that are pure Gaussian noise

Table 5. Average accuracy and its standard deviation of different classifiers with tuned hyperparameters and $n\_inrrelevant = 200$.

Classifier	Tuned Hyperparameter[1]	Train Accuracy		Test Accuracy	
		Average	Std. Dev.	Average	Std. Dev.
Decision Tree	$\{4, 2, 2, 2, 2\}$	0.9182	0.0170	0.9024	0.0054
K-nearest Neighbors	$\{5, 5, 5, 5, 5\}$	0.8550	0.0181	0.7870	0.0069
Perceptron	$\{10^{-4}, 10^{-4}, 5 \times 10^{-4}, 5 \times 10^{-4}, 10^{-4}\}$	0.9398	0.0158	0.9074	0.0074

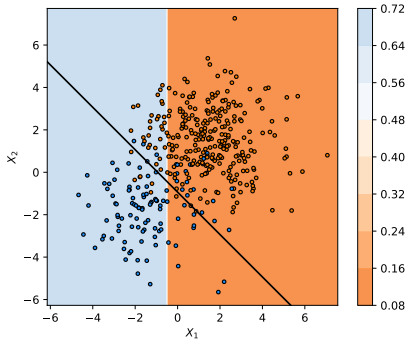
(4.4) Discuss the performance and the ranking of the different methods in two settings

According to Table 4, Table 5, and Figure 5, comparing classifier performance with and without noise reveals notable differences in accuracy and stability. Without noise, Decision Tree, K-Nearest Neighbors, and Perceptron all achieve similar training accuracies (around 0.93) and test accuracies (around 0.91). However, adding 200 Gaussian noise features significantly reduces KNN's test accuracy from 0.92 to under 0.80 and train accuracy to 0.85. In contrast, Decision Tree and Perceptron maintain test accuracies above 0.90, showing greater noise robustness.

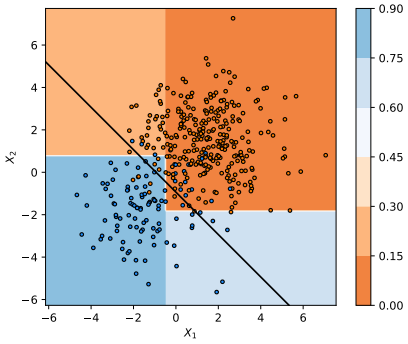
Tuned hyperparameters reflect adaptive strategies: Decision Tree reduces maximum depth for better generalization, Perceptron lowers the learning rate for smoother convergence, and KNN decreases neighbors, simplifying the model but leading to further accuracy loss. Overall, Perceptron ranks highest in stability and accuracy, followed by Decision Tree, with KNN most impacted by noise.

[1]Tuned hyperparameters for each of the 5 generations: max_depth for the Decision Tree, $n\_neighbors$ for K-nearest Neighbors, and $learning\_rate$ for the Perceptron.

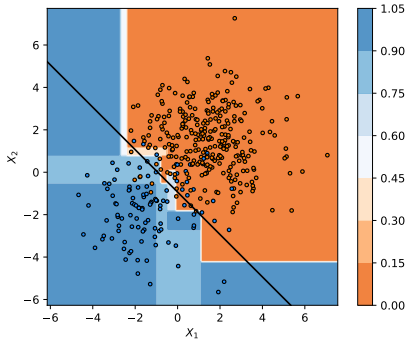
APPENDIX



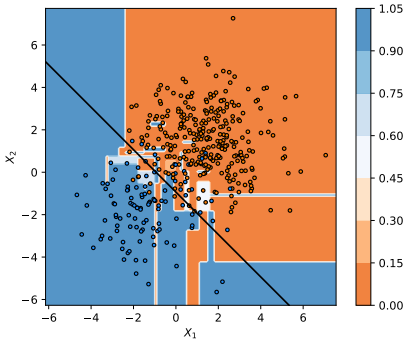
(a) $max\_depth = 1$



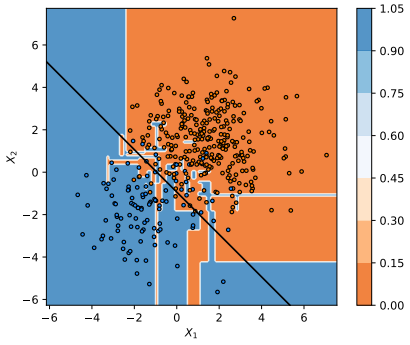
(b) $max\_depth = 2$



(c) $max\_depth = 4$

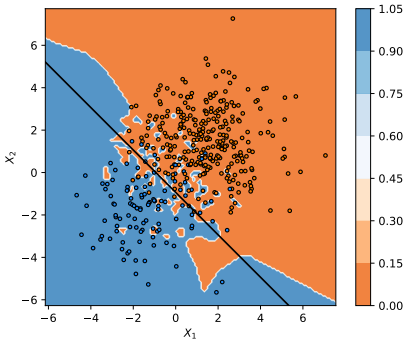


(d) $max\_depth = 8$

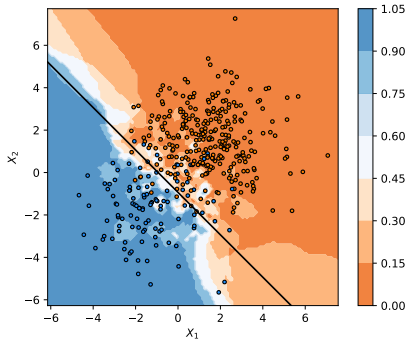


(e) $max\_depth = None$

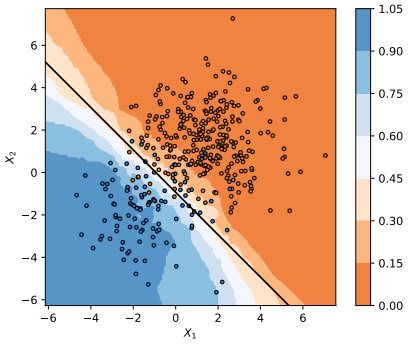
Fig. 1. Decision boundary - Decision tree classifier.



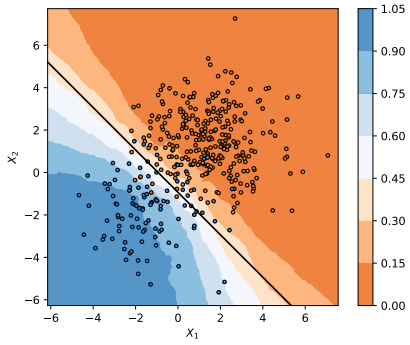
(a) $n\_neighbors = 1$



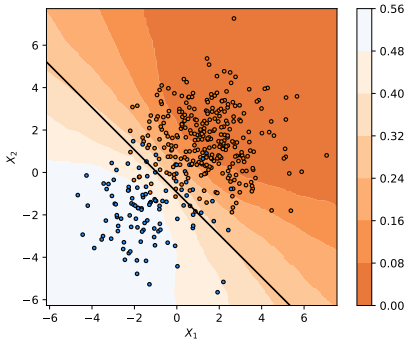
(b) $n\_neighbors = 5$



(c) $n\_neighbors = 50$

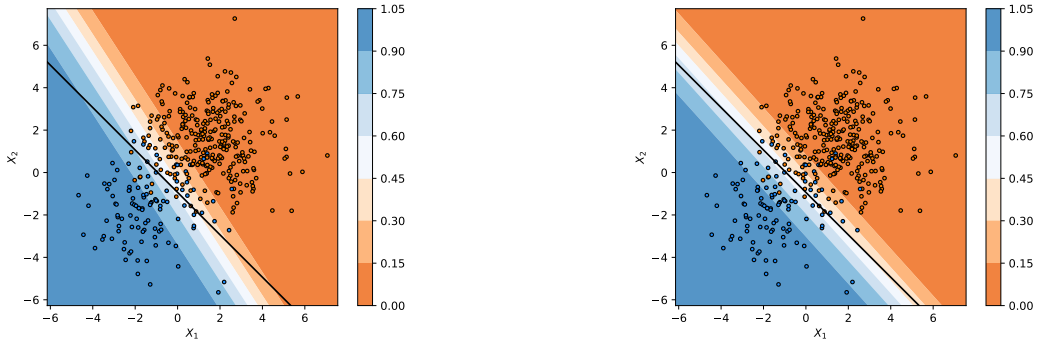


(d) $n\_neighbors = 100$



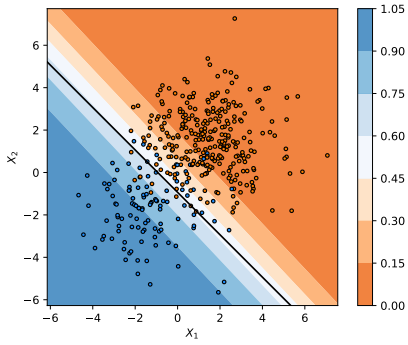
(e) $n\_neighbors = 500$

Fig. 2. Decision boundary - K-nearest neighbors classifier.

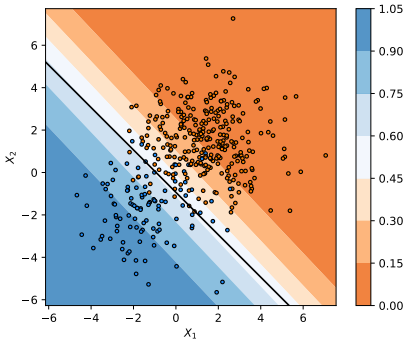


(a) $\eta = 1 \times 10^{-1}$

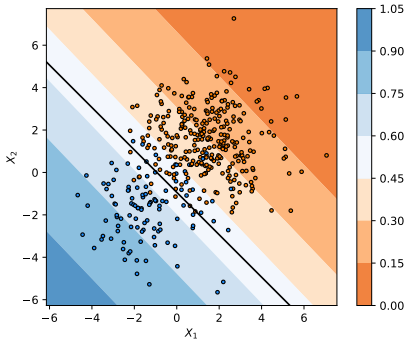
(b) $\eta = 1 \times 10^{-2}$



(c) $\eta = 1 \times 10^{-3}$

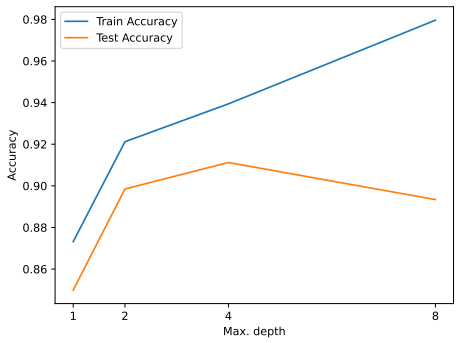


(d) $\eta = 5 \times 10^{-4}$

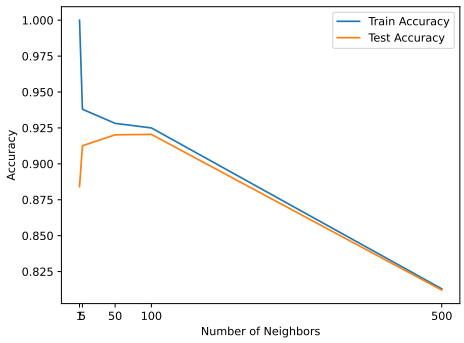


(e) $\eta = 1 \times 10^{-4}$

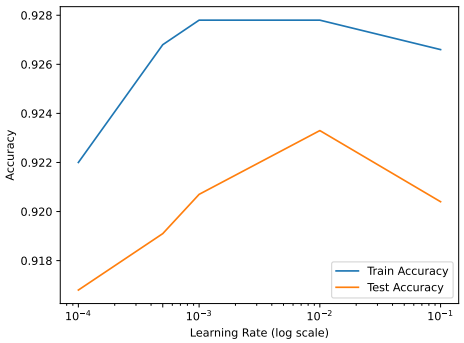
Fig. 3. Decision boundary - Perceptron classifier.



(a) Decision Tree

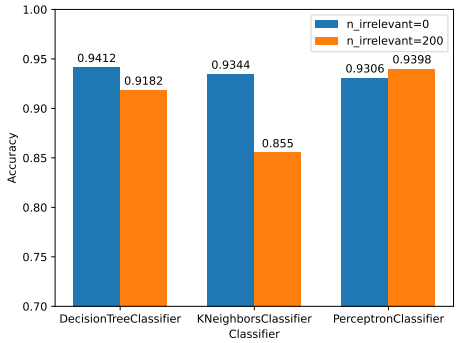


(b) K-nearest Neighbors

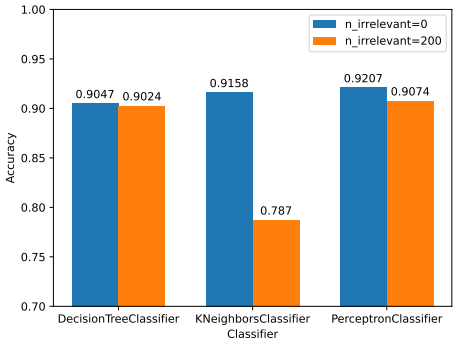


(c) Perceptron

Fig. 4. Complexity curves of average accuracy regarding 5 generations.



(a) Training Set



(b) Test Set

Fig. 5. Method comparison between data without noise ($n\_irrelevant = 0$) and with noise ($n\_irrelevant = 200$) regarding 5 generations.