THE PERCEPTRON: A SINGLE-LAYER LINEAR CLASSIFICATION DEVICE

Sekenova Balym
23B031433

February 12, 2026

Contents

I. OBJECTIVE

The objective of this laboratory work was to study the structure and learning mechanism of a single-layer perceptron as a fundamental linear classification model.

During this experiment, I aimed to:

1. Understand the fundamental architecture of the perceptron and clearly identify its main components, including input features, synaptic weights, bias term, activation function, and binary output.
2. Manually compute the weighted sum for given input vectors and apply the step activation function to determine the predicted class.
3. Implement the perceptron learning algorithm in Python using the `scikit-learn` library and train the model on structured business-related data.
4. Apply the trained perceptron model to practical binary classification problems such as credit approval and quality control.
5. Evaluate the performance of the trained classifier by calculating accuracy, constructing a confusion matrix, and analyzing prediction results.
6. Analyze the limitations of the single-layer perceptron, particularly its inability to correctly classify linearly non-separable datasets.

II. APPARATUS AND MATERIALS

- Personal computer with Python 3.8+
- Libraries: `numpy`, `pandas`, `matplotlib`, `scikit-learn`
- Environment: Google Colab

III. THEORETICAL BACKGROUND

A. Historical Context

The perceptron was introduced in 1958 by Frank Rosenblatt at the Cornell Aeronautical Laboratory. It represents one of the earliest computational models inspired by biological neurons and is considered the simplest form of an artificial neural network.

Although modern deep learning models are significantly more complex, the perceptron remains a fundamental building block in neural network theory. It provides an essential understanding of linear classification and supervised learning principles. In practice, it is suitable for problems where the data are linearly separable.

B. Mathematical Formulation

A single-layer perceptron receives n input features, denoted as $x_1, x_2, \ldots, x_n$. Each input is multiplied by a corresponding weight $w_1, w_2, \ldots, w_n$. The model computes a weighted sum of the inputs and adds a bias term b.

The net input (weighted sum) is defined as:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

where:

- z — weighted sum (net input),
- w_i — weight associated with input x_i,
- x_i — value of the i-th input feature,
- b — bias term,
- n — number of input features.

C. Activation Function

After computing the weighted sum, the perceptron applies the Heaviside step function:

$$y = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

The output y represents the predicted class label. A value of 1 corresponds to the positive class (e.g., approval or acceptance), while 0 corresponds to the negative class (e.g., rejection).

D. Decision Boundary

For a perceptron with two input features (x_1 and x_2), the decision boundary is defined by:

$$w_1 x_1 + w_2 x_2 + b = 0$$

Solving for x_2 gives:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

This equation represents a straight line with slope:

$$m = -\frac{w_1}{w_2}$$

and intercept:

$$c = -\frac{b}{w_2}$$

Therefore, the perceptron can only separate classes using a linear decision boundary.

E. Perceptron Learning Rule

The perceptron learns by iteratively adjusting its weights when a misclassification occurs. The update rule is given by:

$$w_i^{new} = w_i^{old} + \eta(d - y)x_i$$

$$b^{new} = b^{old} + \eta(d - y)$$

where:

- η — learning rate,
- d — desired (target) output,
- y — predicted output,
- $(d - y)$ — error signal.

If the prediction is correct ($d = y$), no update is performed. If the model produces a false negative ($d = 1, y = 0$), the weights increase. If it produces a false positive ($d = 0, y = 1$), the weights decrease.

The perceptron converges only when the dataset is linearly separable. For non-linearly separable problems, convergence is not guaranteed.

F. Scikit-learn Implementation

In practical implementation, the perceptron algorithm can be applied using the `Perceptron` class from the `scikit-learn` library. Important parameters include:

- `eta0` — learning rate,
- `max_iter` — maximum number of training epochs,
- `tol` — stopping tolerance,
- `random_state` — random seed for reproducibility.

After training, the learned parameters are stored as:

- `coef_` — weight vector,
- `intercept_` — bias term.

IV. PROCEDURE

PART A: Environment Setup and Library Import

At the beginning of the laboratory work, the required Python libraries were imported and their versions were verified to ensure compatibility with the assignment requirements. The experiment was conducted using Google Colab.

The installed library versions are summarized in Table 1.

Table 1: Installed Python Library Versions

Library	**Version**
NumPy	2.0.2
Pandas	2.2.2
Matplotlib	3.10.0
Scikit-learn	1.6.1

All required libraries were successfully imported without errors, confirming that the environment was properly configured for the implementation of the perceptron algorithm.

PART B: Manual Computation Exercise

In this part, the perceptron output was calculated manually for a credit approval problem with two input features: income score (x_1) and credit history (x_2).

Table 2: Training Data for Credit Approval

Sample	x_1	x_2	d
1	0.8	0.6	1
2	0.3	0.2	0
3	0.6	0.8	1
4	0.2	0.4	0

The feature matrix has shape $(4, 2)$ and the target vector has shape $(4,)$.
The perceptron was initialized with the following parameters:

$$w_1 = 0.5, \quad w_2 = 0.5, \quad b = -0.5, \quad \eta = 1.0$$

Manual Computation for Sample 1

The weighted sum was computed as:

$$z = w_1 x_1 + w_2 x_2 + b$$

$$z = (0.5)(0.8) + (0.5)(0.6) - 0.5$$

$$z = 0.40 + 0.30 - 0.50 = 0.20$$

Since $z \geq 0$, the activation function produced:

$$y = 1$$

Given that the desired output was $d = 1$, the error is:

$$d - y = 0$$

Therefore, no weight update was required.

Epoch 1 Results

The same procedure was applied to all training samples. The results of Epoch 1 are summarized in Table 3.

Table 3: Manual Perceptron Computation – Epoch 1

Sample	x_1	x_2	z	y	d	$d-y$	Update Required
1	0.8	0.6	0.20	1	1	0	No
2	0.3	0.2	-0.25	0	0	0	No
3	0.6	0.8	0.20	1	1	0	No
4	0.2	0.4	-0.20	0	0	0	No

As shown in the table, all samples were correctly classified during the first epoch. The error term $(d - y)$ was equal to zero for every observation, meaning that no weight updates were performed.

Therefore, after completing Epoch 1, the final parameters remained unchanged:

$$w_1 = 0.5, \quad w_2 = 0.5, \quad b = -0.5$$

This result indicates that the dataset is linearly separable and that the initial weights already define a valid decision boundary.

PART C: Python Implementation with Scikit-learn

The perceptron model was implemented using the `Perceptron` class from the `scikit-learn` library. The same dataset from Part B was used for training.

The feature matrix has shape $(4, 2)$ and the target vector has shape $(4,)$.

The model was initialized with:

$$\eta_0 = 1.0, \quad \texttt{max\_iter} = 100, \quad \texttt{tol} = 10^{-3}, \quad \texttt{random\_state} = 42$$

Training Process

The training process converged after 9 epochs. The evolution of the training process is summarized in Table 4.

Table 4: Training Progress

Epoch	Norm	Bias	Avg. Loss
1	0.64	0.0	0.3400
2	1.84	0.0	0.4175
3	1.44	-1.0	0.1900
4	1.44	-1.0	0.0000
5	1.44	-1.0	0.0000
6	1.44	-1.0	0.0000
7	1.44	-1.0	0.0000
8	1.44	-1.0	0.0000
9	1.44	-1.0	0.0000

Convergence was achieved after 9 epochs.

Learned Parameters

The learned weights and bias are presented in Table 5.

Table 5: Learned Perceptron Parameters

Parameter	Value
w_1	1.2
w_2	0.8
b	-1.0
Number of iterations	9

Prediction Results

Predictions were generated for all training samples. The results are shown in Table 6.

Table 6: Sample-by-Sample Prediction Results

Sample	Actual	Predicted	Result
1	1	1	Correct
2	0	0	Correct
3	1	1	Correct
4	0	0	Correct

The training accuracy was calculated as:

$$Accuracy = \frac{4}{4} \times 100\% = 100\%$$

Confusion Matrix

The confusion matrix is shown in Table 7.

Table 7: Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	2	0
Actual 1	0	2

Classification Report

The detailed classification metrics are presented in Table 8.

Table 8: Classification Report

Class	Precision	Recall	F1-score	Support
Rejected	1.00	1.00	1.00	2
Approved	1.00	1.00	1.00	2
Accuracy		1.00 (100%)		
Macro Avg	1.00	1.00	1.00	4
Weighted Avg	1.00	1.00	1.00	4

PART D: Visualization

Using the learned parameters from Part C:

$$w_1 = 1.2, \quad w_2 = 0.8, \quad b = -1.0$$

the decision boundary of the perceptron is defined by:

$$1.2000\, x_1 + 0.8000\, x_2 - 1.0000 = 0$$

Solving for x_2 gives the slope–intercept form:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

Substituting the learned values:

$$x_2 = -1.5000\, x_1 + 1.2500$$

The slope and intercept are therefore:

$$m = -1.5, \quad c = 1.25$$

This confirms that the perceptron defines a linear decision boundary that separates the two classes in the feature space.

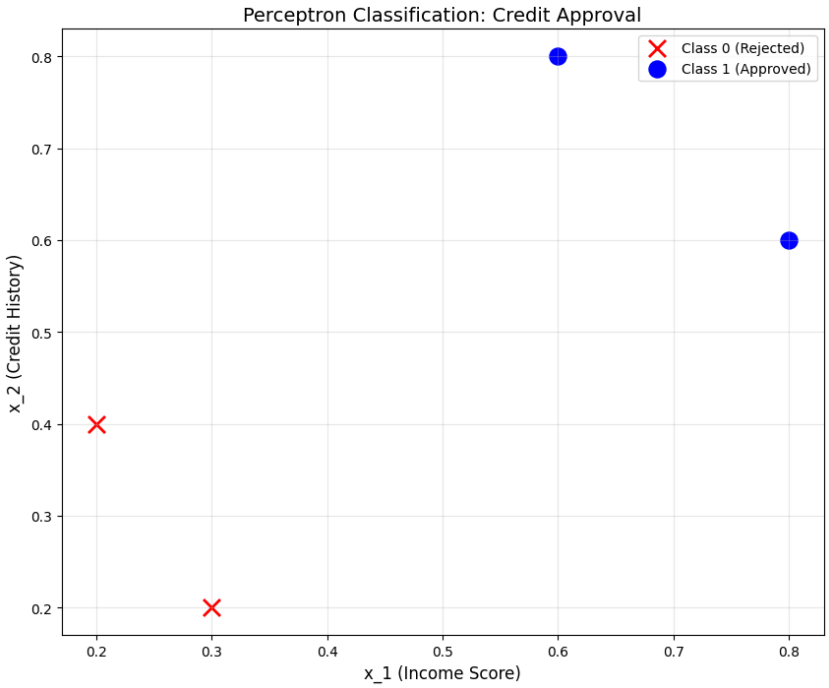

Figure 1: Training data visualization showing class distribution.

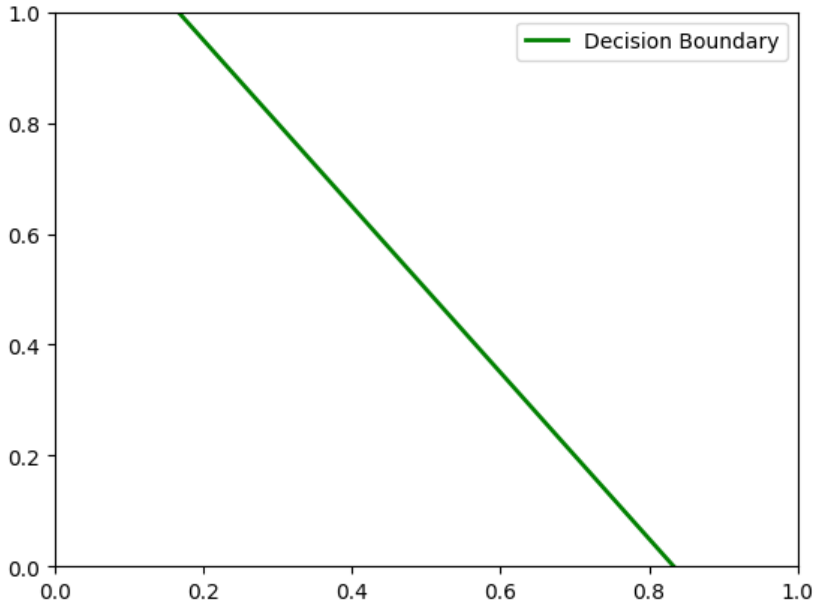

Figure 2: Linear decision boundary learned by the perceptron.

PART E: Testing with New Data

In this part, the trained perceptron model was applied to new unseen instances in order to evaluate its generalization capability.

Predictions for New Instances

The new input vectors are presented in Table 9.

Table 9: New Test Instances

Instance	x_1	x_2
1	0.7	0.5
2	0.4	0.3
3	0.5	0.7

The classification results are shown in Table 10.

Table 10: Predictions for New Instances

Instance	x_1	x_2	Predicted Class
1	0.7	0.5	Approved (1)
2	0.4	0.3	Rejected (0)
3	0.5	0.7	Approved (1)

Manual Verification for Instance 1

To verify the correctness of the model prediction, the weighted sum was computed manually using the learned parameters:

$$z = w_1 x_1 + w_2 x_2 + b$$

Substituting the values:

$$z = (1.2)(0.7) + (0.8)(0.5) - 1.0$$

$$z = 0.84 + 0.40 - 1.0 = 0.24$$

Since $z \geq 0$, the activation function gives:

$$y = 1$$

The manual prediction ($y = 1$) matches the model prediction, confirming the correctness of the classification.

The results demonstrate that the trained perceptron correctly classifies new linearly separable data points using the learned decision boundary.

V. INDIVIDUAL ASSIGNMENT — Variant 3

Problem 3.1 — Computational Analysis

For the given perceptron parameters:

$$w_1 = 0.5, \quad w_2 = 0.5, \quad w_3 = -0.4, \quad b = -0.3$$

and input vector:

$$(x_1, x_2, x_3) = (0.7, 0.4, 0.6), \quad d = 1$$

the weighted sum was computed as:

$$z = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$z = (0.5)(0.7) + (0.5)(0.4) + (-0.4)(0.6) - 0.3$$

$$z = 0.35 + 0.20 - 0.24 - 0.30 = 0.01$$

Since $z \geq 0$, the predicted output is:

$$y = 1$$

The error term:

$$d - y = 0$$

Because the classification was correct, no weight update was performed. The parameters remained unchanged.

Problem 3.2 — Applied Business Problem

The perceptron was trained on the provided dataset using:

$$\eta_0 = 1.0, \quad \texttt{max\_iter} = 100, \quad \texttt{random\_state} = 42$$

Training converged after 9 epochs.
The learned parameters are shown in Table 11.

Table 11: Variant 3 — Learned Parameters

Parameter	Value
w_1	1.0
w_2	1.0
b	-1.0
Iterations	9

The resulting decision boundary is:

$$1.0000\,x_1 + 1.0000\,x_2 - 1.0000 = 0$$

Solving for x_2:

$$x_2 = -1.0000\,x_1 + 1.0000$$

A new unit with:

$$x_1 = 0.4, \quad x_2 = 0.5$$

was classified using the trained model. The predicted class was:

$$y = 0$$

indicating rejection according to the learned decision boundary.

Problem 3.3 — Economic Impact Analysis

The business cost parameters were:

- Cost per false positive: \$200
- Cost per false negative: \$50
- Annual production volume: 50,000 units
- 15 false positives per 1000 units
- 25 false negatives per 1000 units

The total number of misclassifications annually is:

$$\text{False Positives} = \frac{50{,}000}{1000} \times 15 = 750$$

$$\text{False Negatives} = \frac{50{,}000}{1000} \times 25 = 1250$$

The total annual misclassification cost is therefore:

$$\text{Cost} = (750 \times 200) + (1250 \times 50)$$

$$\text{Cost} = 150{,}000 + 62{,}500 = 212{,}500$$

Thus, the annual expected loss due to misclassification equals:

$$\boxed{\$212{,}500}$$

This analysis demonstrates how even a simple linear classifier such as the perceptron can have significant financial implications in real-world business scenarios.

VII. ANALYSIS QUESTIONS

1. XOR Problem

The XOR problem consists of four points: $(0,0) \to 0$, $(0,1) \to 1$, $(1,0) \to 1$, $(1,1) \to 0$.

A single-layer perceptron cannot learn the XOR function because the data are not linearly separable. The perceptron constructs a decision boundary in the form of a straight line (or hyperplane in higher dimensions). However, in the XOR dataset, the two positive points lie diagonally opposite each other, and the two negative points also lie diagonally opposite each other.

No single straight line can separate the positive and negative classes without misclassifying at least one point. Therefore, convergence is impossible for a single-layer perceptron. Solving the XOR problem requires either a non-linear transformation of features or a multi-layer neural network.

2. Effect of Learning Rate (η_0)

The learning rate η_0 controls the magnitude of weight updates during training. If η_0 is too small, weight updates become very slow, resulting in slow convergence and increased training time. If η_0 is too large, updates may overshoot the optimal solution, causing oscillations or instability in training.

An excessively large learning rate may prevent convergence altogether, while a very small learning rate may require many iterations to reach a solution. In practice, a reasonable range for η_0 is typically between 0.1 and 1.0 for standard perceptron training, depending on the scale of the data.

3. Asymmetric Misclassification Costs

In many business problems, false positives and false negatives have different financial consequences. To account for asymmetric costs, the perceptron training procedure can be modified in several ways.

One approach is to adjust the decision threshold instead of using $z \geq 0$ strictly. Shifting the threshold allows prioritizing either sensitivity or specificity depending on business risk. Another approach is to introduce cost-sensitive learning by weighting training samples differently based on their class importance. This ensures that errors associated with higher financial loss receive larger weight updates during training.

4. Perceptron vs Logistic Regression

Both the perceptron and logistic regression are linear classifiers that construct a linear decision boundary. However, their outputs differ significantly.

The perceptron produces a hard binary output (0 or 1) using a step activation function. Logistic regression, on the other hand, outputs a probability between 0 and 1 using the sigmoid function. Logistic regression is trained using a differentiable loss function (log-loss), while the perceptron updates weights only when misclassification occurs.

In my opinion, logistic regression would be preferable in business settings because it provides probabilistic outputs. Probabilities allow better decision-making, threshold adjustment, and risk estimation. The perceptron is simpler and computationally efficient but less informative for decision analysis.

5. Failure to Converge

If a perceptron fails to converge after reaching `max_iter` epochs and produces a `ConvergenceWarning`, this usually indicates that the dataset is not linearly separable. Since the perceptron requires linear separability for guaranteed convergence, it cannot find a stable decision boundary in such cases.

Possible alternatives include using logistic regression, adding polynomial feature transformations to introduce non-linearity, or employing multi-layer neural networks. Another option is to apply regularization techniques or switch to algorithms that minimize differentiable loss functions instead of relying solely on misclassification updates.

VIII. CONCLUSION

In this laboratory work, the single-layer perceptron was studied both theoretically and practically. The experiment began with manual computation of weighted sums and activation outputs, which helped to clearly understand the internal mechanism of the model. The results confirmed that when the dataset is linearly separable, the perceptron can correctly classify all observations without further weight updates.

The perceptron was then implemented using the `scikit-learn` library. The model successfully converged after 9 epochs and achieved 100% training accuracy. The learned decision boundary was derived analytically and visualized, confirming the linear separation of the classes.

In the individual assignment (Variant 3), the perceptron was applied to a business-related scenario. Manual computation, model training, and economic analysis were performed. The financial evaluation demonstrated that classification performance directly affects annual business losses, emphasizing the practical importance of accurate modeling.

Overall, hrough this laboratory work, I gained a clear understanding of linear classification, perceptron learning dynamics, convergence behavior, and real-world implications. At the same time, it highlighted the key limitation of the perceptron model — its inability to solve non-linearly separable problems.