BBM409 : Introduction to Machine Learning Lab.
Fall 2025
Instructor: Assoc. Prof. Dr. Hacer Yalım Keleş
TA: Sevginur İnce

Assignment 1: Understanding Perceptron

Due date: Tuesday, 4-11-2025, 11:59 PM

In this assignment, you will explore the **Perceptron Learning Algorithm** by implementing it for a binary classification task using the **Raisin Dataset** from the UCI Machine Learning Repository. The dataset contains 900 samples with 7 numerical features (*Area, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, Extent, Perimeter*) describing the physical characteristics of raisins. Your task is to classify raisin samples as either belonging to the **Kecimen** class (class 1) or the **Besni** class (class 0).

1 Assignment Overview

The steps you will follow include:

1. **Implementation:** Implement the Perceptron Learning Algorithm from scratch to train it on the Raisin Dataset.
2. **Evaluation and Metrics:** Evaluate your model using accuracy, F1 score, recall, and precision. You will also answer guiding questions to understand the importance of these metrics.
3. **Visualization:** Reduce the feature space to two dimensions using correlation analysis and visualize the Perceptron's decision boundary. You will also compare the 2D separation with a 1D Fisher's Linear Discriminant projection.

This assignment provides a comprehensive understanding of the Perceptron algorithm, emphasizing how its decision boundary and performance are affected by the dataset characteristics.

2 Perceptron Learning Algorithm

The **Perceptron Learning Algorithm** is one of the simplest and most fundamental binary classifiers, designed to find a linear separator between two classes. It was introduced by Frank Rosenblatt in 1958 and is based on a mathematical model of a neuron. In this assignment, we will use it for binary classification tasks, where the goal is to classify data into one of two categories. The Perceptron algorithm iteratively adjusts weights based on the classification errors encountered during training.

2.1 How the Algorithm Works

1. **Initialization:** The Perceptron starts by initializing a weight vector $\mathbf{w}$ and a bias b to zero (or small random values). Each input feature vector $\mathbf{x}$ has an associated label $y \in \{-1, +1\}$, representing two possible classes.

2. **Prediction Function:** For any input $\mathbf{x}$, the algorithm computes a weighted sum of the input features, adding the bias:

$$z = \mathbf{w}^T \mathbf{x} + b \tag{1}$$

   The activation function determines the predicted class label by applying a step function:

$$\hat{y} = \text{sign}(z) = \begin{cases} +1, & \text{if } z \geq 0 \\ -1, & \text{if } z < 0 \end{cases} \tag{2}$$

3. **Weight Update Rule:** The Perceptron updates its weights and bias each time it misclassifies a sample. For each misclassified example (x_i, y_i), where $\hat{y}_i \neq y_i$, the weights are adjusted as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i \tag{3}$$

$$b \leftarrow b + \eta y_i \tag{4}$$

   where η is the learning rate, which controls how much the weights are adjusted at each step.

4. **Convergence:** If the data is linearly separable, the Perceptron algorithm is guaranteed to converge, meaning it will find a hyperplane that separates the two classes with no classification errors. If the data is not linearly separable, the algorithm will not converge and will continue updating indefinitely or until a maximum number of iterations is reached.

2.2 Key Concepts

- **Linearly Separable Data:** Data that can be perfectly separated by a straight line (in 2D) or a hyperplane (in higher dimensions).
- **Margin:** The margin refers to the distance between the hyperplane and the closest data points on either side. Larger margins typically indicate better generalization performance.

3 Implementation

3.1 Step 1: Data Preparation

1. **Download and Load the Data:** Download the Raisin Dataset from the UCI Machine Learning Repository.

2. **Load the dataset:** Load the dataset using Pandas, ensuring that features and class labels are correctly formatted.

3. **Preprocess the Data:** Analyze the dataset and decide whether preprocessing (such as normalization or standardization) is needed. Explain your choices in a brief comment. The dataset consists of seven features: Area, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, Extent, and Perimeter. The goal is to classify the samples as either Kecimen (class 1) or Besni (class 0).

4. **Training and Validation Split:** Split the dataset into 80% training and 20% validation sets. Explain why this split is essential for model generalization. Discuss how evaluating the model on unseen data helps in assessing its generalization capability.

3.2 Step 2: Initialize the Perceptron

1. Initialize $\mathbf{w}$ and b to zeros or small random values.

2. Select appropriate values for the learning rate η and the number of epochs (iterations over the training set).

3. Discuss how these hyperparameters affect training speed and convergence. What happens with a very small or very large learning rate?

3.3 Step 3: Train the Perceptron

1. Implement the training loop where the Perceptron iteratively processes each training sample:

   (a) Compute the weighted sum of the input features plus the bias.

   (b) Apply the activation function (step function) to make a prediction based on the weighted sum.

   (c) If the prediction is incorrect, apply the weight update rule to adjust the weights and bias accordingly.

2. Continue this process for a set number of epochs or until the model converges (i.e., when no errors are made on the training set). Briefly discuss how the algorithm handles linearly separable versus non-linearly separable data.

3. After training, apply the trained Perceptron to both training and validation sets. For each sample: compute the prediction using the learned weights and bias, and store predictions for comparison with actual labels.

4 Evaluation

In this section, you will evaluate the performance of your Perceptron model on the validation set. Evaluation metrics help us understand how well the model performs beyond just accuracy. **You are not allowed to use ready-made libraries for calculating metrics and must explain each of them thoroughly by answering the guiding questions.**

4.1 Step 1: Accuracy

Accuracy measures the proportion of correctly classified samples out of the total number of samples:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{5}$$

Guiding Questions

- What does accuracy tell us about the performance of the model?
- Why is accuracy sometimes not enough, especially in cases where the data is imbalanced? Explain a scenario where high accuracy might be misleading.

4.2 Step 2: Precision, Recall, and F1 Score

Precision is the proportion of correctly predicted positive observations to the total predicted positives:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

Recall (or Sensitivity) is the proportion of correctly predicted positive observations to all actual positives:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

F1 Score is the harmonic mean of precision and recall, providing a single metric that balances the trade-off between them:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

Guiding Questions

- In what types of problems is precision more important than recall? Give an example scenario where high precision is desirable but low recall might be acceptable.
- In what cases is recall more critical? Discuss an example where missing true positives is more harmful than having false positives.
- When is the F1 score a better measure than accuracy? Explain why balancing precision and recall is important in some contexts.
- What are the limitations of using F1 score alone?
- Analyze each metric in the report — such as **accuracy, precision, recall, f1-score, and support— and interpret the performance of your model.

5 Visualization of the Decision Boundary

In this section, you will explore how the Perceptron Learning Algorithm forms the decision boundary (or hyperplane) that separates the two classes. To do this, you will reduce the feature space to two dimensions, train the Perceptron on the selected features, and visualize the hyperplane.

5.1 Step 1: Feature Selection Using Correlation Analysis

1. Compute the correlation matrix among all seven features to identify pairs of features with low correlation.
2. Select two features that are least correlated for visualization purposes. Highly correlated features may provide redundant information.

5.2 Step 2: Train on Selected Features and Visualize

1. Retrain the Perceptron using only the selected two features. Use the same training and evaluation process as before, but now the model will work in a two-dimensional feature space.
2. Visualize the decision boundary (hyperplane) to see how the Perceptron separates the two classes in 2D. (See: Decision Boundary Plot Tutorial)

5.3 Step 3: Experiment with Different Features

After visualizing the decision boundary for one pair of features, try selecting different combinations of features and retrain the Perceptron. Compare how the hyperplane changes with different features.

Guiding Questions

- How does the decision boundary change when you use different pairs of features?
- Can you find a pair of features that leads to better separation between the two classes?

5.4 Bonus Step 4: Incrementally Add Data (+5 points)

1. Start with a small portion of the dataset and progressively add more data points to train the Perceptron.
2. Observe how the decision boundary shifts as you add more data. This will help you understand how the Perceptron continuously adjusts the boundary to separate the classes.

Guiding Questions

- How does the decision boundary evolve as more data is added?
- Does the Perceptron find a stable boundary after a certain number of data points?
- How sensitive is the model to the order in which data points are introduced?

5.5 Step 5: Analyzing the Hyperplane

After running the experiments, analyze the hyperplane's behavior across different feature sets and data inputs.

Guiding Questions

- Why does the hyperplane change with different features or data points?
- How does the decision boundary relate to the linearly separable nature of the data?

5.6 Step 6: Fisher's Linear Discriminant Projection

In this step, you will implement Fisher's Linear Discriminant (LD) to project the dataset onto a 1D space and visualize the distribution of the two classes using a histogram. The goal of Fisher's LD is to find the direction in the feature space that maximizes the separation between two classes while minimizing the variance within each class.

Instructions

1. **Compute Fisher's Linear Discriminant:**
   (a) Calculate the mean vectors μ_0 and μ_1 of the two classes.
   (b) Compute the within-class scatter matrix S_{within} for both classes.
   (c) Compute the between-class scatter matrix to maximize class separation.
   (d) Find the projection direction using Fisher's Linear Discriminant formula:

$$\mathbf{w} = S_{\text{within}}^{-1}(\mu_1 - \mu_0) \tag{9}$$

   (e) Normalize the direction vector $\mathbf{w}$ to get a unit vector.
   (f) Visualize the projection direction computed by Fisher's LD in the original 2D feature space. This will help you understand how the data is being projected onto a 1D space and why this direction is optimal for class separation.
2. **Project the Data:** Use the projection direction $\mathbf{w}$ to project the data from the original feature space to 1D space:

$$\text{Projected Data} = \mathbf{x} \cdot \mathbf{w} \tag{10}$$

3. **Visualize the Projected Data:** Plot a histogram showing the distribution of the projected data for each class in the 1D space.

Guiding Questions

In addition to the coding tasks present in the Jupyter notebook file, there are also questions that you need to answer.

- How well does Fisher's LD separate the two classes in the 1D projected space?
- Compare the separation of the two classes in the original 2D space (used in the Perceptron) versus the 1D Fisher's LD projection. What do you observe about the class distributions in these spaces?
- What insights can you gain from using Fisher's LD for dimensionality reduction and class separation?

6 What to Hand In

Your submitted solution should include the following:

- The filled-in Jupyter Notebook as both source code and report, including: (1) markdown cells reporting written answers alongside relevant figures and images, and (2) well-commented code cells with reproducible results.
- A self-contained report encompassing:
  - A concise overview of the problem statement, detailing objectives and scope.
  - Description of the dataset, including features, target variables, and preprocessing steps.
  - Explanation of the approach adopted for each part, describing the algorithms implemented.
  - Explanation of evaluation metrics and their relevance to the problem.
  - Reflection on experimental outcomes, discussing strengths, limitations, and challenges.
  - Tables or figures to emphasize specific aspects and enhance clarity.
- **Important:** Utilizing ready-made libraries for the Perceptron learning algorithm, evaluation metrics, and Fisher LD implementation are **prohibited**. You must implement them from scratch.
- Numpy array functions are permissible for intermediate implementation steps. You may use Pandas for reading/writing CSV files and Matplotlib for visualization.
- A ZIP file named `<studentid>.zip` containing the Jupyter notebook (.ipynb) and its Python file (.py) version with all codes.

Submit via submit.cs.hacettepe.edu.tr.

7 Grading

- Implementation and Comments: 30 points
- Evaluation and Explanations: 25 points
- Visualization and Analysis: 45 points (+5 bonus points)

Note: Preparing a good report is as important as the correctness of your solutions. You should explain your choices and their effects on the results.

8 Academic Integrity

All work must be done individually. Discussing general ideas with classmates is allowed, but sharing or reusing code, either from peers or AI sources, is strictly prohibited. Violations will be considered breaches of academic integrity. Please note that this condition also holds for online/AI sources. Make use of them responsibly and refrain from generating the code you are asked to implement.