BBM409 : Introduction to Machine Learning Lab.
Fall 2024
Instructor: Assoc. Prof. Dr. Hacer Yalım Keleş
TA: Sümeyye Meryem Taşyürek

Assignement 1: Understanding Perceptron

Due date: Tuesday, 5-11-2024, 11:59 PM.

In this assignment, you will explore the Perceptron Learning Algorithm by implementing it for a binary classification task using the **Banknote Authentication Dataset**. The dataset contains 1,372 samples with 4 features (variance, skewness, kurtosis, and entropy of the banknote image), and your task will be to classify banknotes as either authentic (class 1) or fake (class 0).
The steps you will follow include:

1. **Implementation:** You will implement the Perceptron Learning Algorithm from scratch to train it on the Banknote Authentication Dataset.
2. **Evaluation and Metrics:** You will evaluate the performance of your model using metrics such as accuracy, F1 score, recall, and precision. A set of questions will guide you to understand the importance of using these metrics and validation data in assessing the model's performance.
3. **Visualization:** You will reduce the feature space to two dimensions using feature selection and visualize the hyperplane generated by the Perceptron. You will explore how the hyperplane's behavior depends on the specific data being used. You will also compare the separation of the two classes in the original 2D space (used in the Perceptron) versus the 1D Fisher's LD projection.

This assignment will provide a comprehensive understanding of the Perceptron algorithm, with a focus on how its performance is evaluated and how its decision boundary is influenced by the underlying data.

1 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.

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:

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

The *activation function* determines the predicted class label by applying a step function to the weighted sum:

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

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

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

$$b \leftarrow b + \eta y_i$$

Here, 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.

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.

In this assignment, you will implement this algorithm and evaluate its performance on the *Banknote Authentication Dataset*.

2 Implementation

2.1 Step 1: Data Preparation

1. **Download and Load the Data:** Start by downloading the Banknote Authentication Dataset from the UCI repository. Or you can import it in Python using the commands given in UCI page (Banknote Authentication Dataset from UCI).

2. Load the dataset using a data-handling library such as Pandas, ensuring that the data is formatted correctly with features and target labels.

3. **Preprocess the Data:** Analyze the dataset and decide whether preprocessing (such as normalization) is needed. Explain your choices in a brief comment. The dataset consists of four features: variance, skewness, kurtosis, and entropy. The goal is to classify the samples as either authentic (class 1) or fake (class 0).

4. **Training and Validation Split:** Split the dataset into a training set and a validation set (e.g., 80% training and 20% validation). The training set will be used to train the Perceptron, and the validation set will be used to evaluate the model's generalization performance. Briefly explain why this split is essential for the evaluation process.

5. Explain why it's important to use the validation set to evaluate your model instead of relying solely on the training data. Discuss how evaluating the model on unseen data helps in assessing its generalization capability.

2.2 Step 2: Initialize the Perceptron

1. **Initialization:** Initialize the weight vector $\mathbf{w}$ and bias b to zeros (or small random values). Also, select appropriate values for the learning rate η and the number of epochs(iterations over the training set).

2. Discuss how the choice of learning rate and the number of epochs might affect the training process. What happens with a very small or very large learning rate?

2.3 Step 3: Train the Perceptron

1. **Training Loop:** Implement the training loop, where the Perceptron iteratively processes each training sample.

2. Compute the **weighted sum** of the input features plus the bias.

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

4. If the prediction is incorrect, apply the **weight update rule** to adjust the weights and bias accordingly.

5. 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.

6. After training, apply the trained Perceptron to the training set. For each sample in the training set: Compute the prediction using the learned weights and bias. Store and compare the predictions with the actual labels.

3 Evaluation

In this section, you will evaluate the performance of your Perceptron model on the validation set by making predictions on validation data. 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 for your understanding.

3.1 Step 1: Evaluate with Accuracy

Accuracy measures the proportion of correctly classified samples out of the total number of samples. In other words, it is the ratio of true positives and true negatives to the total dataset.

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

Answer the Guiding Questions:

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

3.2 Step 2: Introduce Precision, Recall and F1 Score

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

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

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

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score: The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances the trade-off between precision and recall.

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

Answer the Guiding Questions:

1. 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.
2. In what cases is recall more critical? Discuss an example where missing true positives is more harmful than having false positives.
3. When is the F1 score a better measure than accuracy? Explain why balancing precision and recall is important in some contexts.
4. What are the limitations of using F1 score alone?

4 Visualization of the Hyperplane

In this section, you will explore how the Perceptron Learning Algorithm forms **the decision boundary** (or hyperplane) that separates the two classes. To be able to do that you will reduce the feature space to two dimensions, train the Perceptron on the selected features, and visualize the hyperplane. The goal is to understand how the Perceptron changes the decision boundary as the data changes, and how it adapts based on the features used.

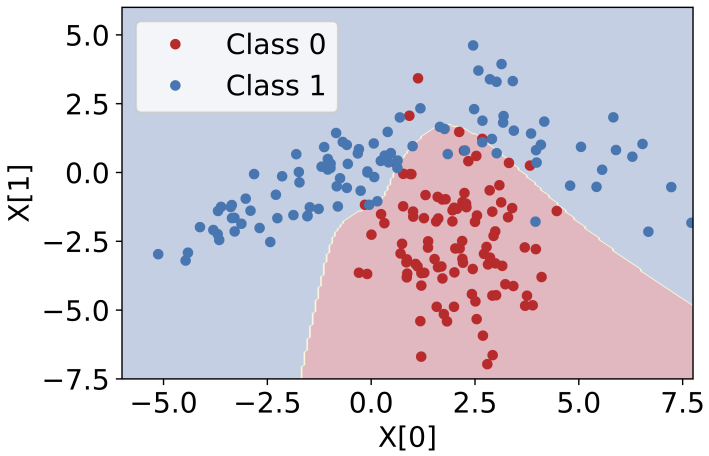

Figure 1: Example of a Decision Boundary Visualization

4.1 Step 1: Feature Selection Using Correlation Analysis

1. **Correlation Analysis:** One effective way to reduce the number of features is by analyzing the correlation between them. Highly correlated features may provide redundant information, so you can select two less correlated features for visualization.

2. **Compute the correlation matrix:** Analyze the relationship between the four features (variance, skewness, kurtosis, and entropy) to identify pairs of features with low correlation.

3. **Choose two features:** Based on the correlation analysis, select two features that are least correlated to visualize the Perceptron's decision boundary.

4.2 Step 2: Train the Perceptron on Selected Features

1. **Retrain the Perceptron:** After selecting two features, train the Perceptron using only these 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):** Once the Perceptron is trained, plot the decision boundary (hyperplane) to see how the Perceptron separates the two classes in 2D. (check "How To Plot A Decision Boundary For Machine Learning Algorithms in Python")

4.3 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. This helps in understanding how the Perceptron's decision-making process adapts to different feature sets.

Answer the guiding questions:

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

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

1. **Visualizing Incremental Learning:** Another experiment to help understand the behavior of the Perceptron is to add data incrementally. 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, observe how the hyperplane shifts and adapts to new points. This will help you understand how the Perceptron continuously adjusts the boundary to separate the classes.

Answer Guiding Questions:

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

4.5 Step 5: Analyzing the Hyperplane

After running the experiments, analyze the hyperplane's behavior across different feature sets and data inputs. Reflect on how the Perceptron algorithm adapts to the data and how different feature combinations lead to different decision boundaries.

Answer the guiding questions:

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

4.6 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 in the projected space 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. This technique projects the data onto a single dimension where the separation between the classes is optimized.

Instructions:

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

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

   (e) **Normalize** the direction vector $\mathbf{w}$ to get a unit vector.
   (f) While not mandatory, it can be highly instructive to 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 the 1D space. The projection for each data point $\mathbf{x}$ is computed as:

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

3. Visualize the Projected Data: Plot a **histogram** showing the distribution of the projected data for each class in the 1D space. The histogram will help you visualize how well the two classes are separated by the Fisher's LD projection.

Answer the guiding questions:

1. How well does Fisher's LD separate the two classes in the 1D projected space?
2. 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?
3. What insights can you gain from using Fisher's LD for dimensionality reduction and class separation?

5 What to Hand In

Your submitted solution should include the following:

- The filled-in Jupyter Notebook as both your source code and report. The notebook should include (1) markdown cells reporting your written answers alongside any relevant figures and images and (2) well-commented code cells with reproducible results.
- Additionally, your report should be self-contained, encompassing a concise summary of the problem statement and a detailed exposition of your implemented solution.
- Begin by providing a concise overview of the problem statement, detailing the objectives and scope of the assignment. Define the tasks involved in the assignment.

- Describe the dataset used for the assignment. Include information about the features, target variables, and any preprocessing steps applied to the data.
- Explain the approach adopted for each part of the assignment. Describe the algorithms implemented.
- Explain the evaluation metrics utilized to assess the performance of your models. Define metrics and justify their relevance to the problem at hand.
- Reflect on the outcomes of your experiments. Discuss the strengths and limitations of your models, as well as any challenges encountered during the implementation process.
- Feel free to include tables or figures to emphasize specific aspects of your solution and enhance clarity for the reader.
- Utilizing ready-made libraries for perceptron learning algorithm, evaluation metrics and Fisher LD implementation are prohibited. You are required to implement them from scratch in your assignment.
- Numpy array functions are permissible for intermediate implementation steps of the assignment. You may use Pandas library for reading and writing/creating csv files and Matplotlib for visualization.

You should prepare a ZIP file named <student id>.zip containing the Jupyter notebook in ipynb format as well as its .py (Python file) version containing all your codes. Submit it to submit.cs.hacettepe.edu.tr, where you will be assigned a submission. The file hierarchy is up to you as long as your Jupyter notebook and Python file works fine.

6 Grading

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

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

Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for online/AI sources. Make use of them responsibly, refrain from generating the code you are asked to implement. Remember that we also have have access to such tools, making it easier to detect such cases.