Lab Course Machine Learning

Exercise Sheet 11

Prof. Dr. Dr. Lars Schmidt-Thieme
Jung Min Choi

HiWi: Harish Malik
Submission deadline : January 18, 2025

1 Perceptron Algorithm (15 points)

A perceptron is a simple one node of a deeper neural network which processes weighted inputs and performs binary classification. We use a toy dataset for the problem. Set aside 50 examples in each for testing. Using an OOP code up the perceptron algorithm. The task also involves to generate an animation showing how the decision boundary varies in each iteration for both training and testing. Load the following datasets in your notebook:

1. **[5 Points]** `Xlin_sep.npy` and `ylin_sep.npy`. This dataset is linearly separable. Run your algorithm for this data and you should achieve 100% train and test accuracies!
2. **[5 Points]** `Xlinnoise_sep.npy` and `ylinnoise_sep.npy`. This dataset is not linearly separable and contains noise. Run your algorithm for this data and observe what happens to the decision boundary in the animation. You should get a test accuracy close to 88%.
3. **[5 Points]** `circles_x.npy` and `circles_y.npy`. This dataset is non-linear. Devise a strategy to make the dataset separable linearly. *Hint: Polynomial Features.* Plot the decision boundary showing how the two classes are separated.

Hints:

1. **Polynomial Feature Expansion:** It takes the original feature matrix `X` and appends higher-order polynomial terms (e.g., X^2) if `order` is greater than 1. This helps the perceptron handle non-linear boundaries.
2. **Decision Boundary:** For `order = 1`, the code solves for a **linear** boundary $y = (-a - bx)/(c + \epsilon)$. For `order = 2`, it handles a **quadratic** boundary by computing a radicand (under the square root) and solving for y. This requires careful consideration of domains where the radicand is non-negative.
3. **Perceptron Training (`fit`):**
   - **Single Pass:** The loop runs once through the entire training set (from $i = 1$ to N). To fully train, you might need multiple passes (epochs), but here it demonstrates the core update mechanism in a single pass.
   - **Update Rule:** The weights are only updated when a misclassification is found, i.e., when $a_i \neq y_i$. The update is
     $$\mathbf{w} \leftarrow \mathbf{w} + \eta(y_i - a_i)\mathbf{x}_i,$$
     where η (`self.r`) is the learning rate.
   - **Tracking the Boundary:** After each update, the method records a new boundary. This is how you can later visualize the evolution of the decision boundary over iterations.

2 MNIST Classification via Neural Networks (5 points)

The task for this exercise is to develop a Neural Network model that can classify human-written digits (0 through 9). We will reuse concepts from previous exercises such as hyperparameter optimization and k-fold cross-validation. Please use the `sklearn` library in your solution. You can only earn the points if you complete all steps and report plausible results.

- **Load the MNIST Digits Dataset:** Use the built-in utility function(s) in `sklearn` to load the MNIST digit images and labels.
- **Set up k-fold Cross-Validation:** Import the necessary classes from `sklearn` to perform k-fold validation. You are free to choose k based on your computational resources, but $k = 5$ is a common choice. Additionally, set aside 20% of the images for *testing*.
- **Define a Hyperparameter Grid:** Use `MLPClassifier` from `sklearn` as your Neural Network model. Consult its documentation to identify relevant hyperparameters (e.g., hidden layer sizes, activation functions, learning rate, etc.). Create a suitable hyperparameter grid or dictionary to explore.
- **Randomized Search:** Implement a *random search* over your chosen hyperparameter ranges. Train the model by calling the `.fit` method on your search object.
- **Report Results:** Provide the final test accuracy on the hold-out test set and print the best hyperparameters found by the search.