Lab Two: Linear classifier based on perceptron algorithm

name: Huizhi Yu
number: ********
email: ********

1 Bi-variate Gaussian distribution

To create two bi-variate Gaussian distributions, different mean vectors and covariance matrices are needed. In this report, we choose:

Mean vector $\mu_1 = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$ and $\mu_2 = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$. A shared covariance matrix $\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

Using these parameters, the data points for each distribution are generated by sampling from the standard Gaussian distribution and transforming them using the Cholesky decomposition of the covariance matrix. Then, we plot the data points to obtain the distribution as shown in Figure 1.

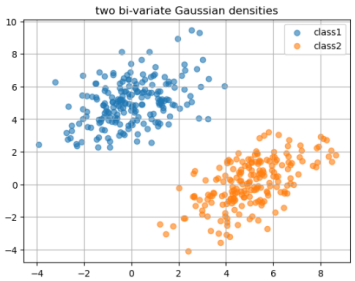


Figure 1: Distribution of the two bi-variate Gaussian densities.

2 Perceptron algorithm

2.1 Classify two classes of Gaussian-distributed data

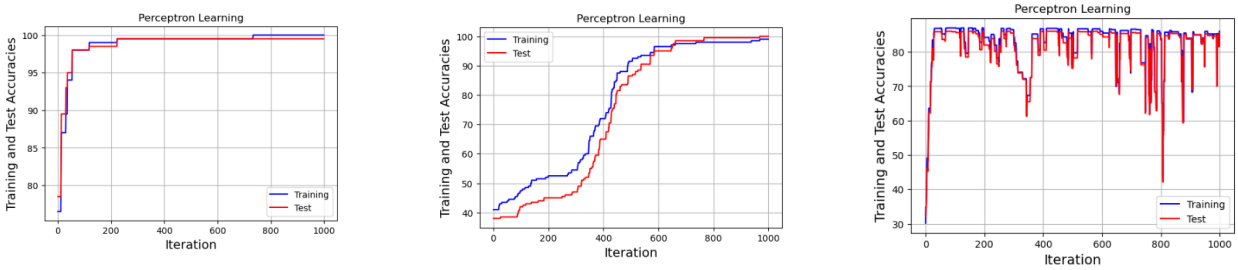
The objective of this section is to use the perceptron algorithm to classify two classes of Gaussian-distributed data. The aim is to compare the performance of a manually implemented perceptron algorithm with that of the perceptron model provided in the scikit-learn package.

- **Data Generation**: We generated two classes of 2D Gaussian-distributed data, each containing 200 points. The means were set at $[0, 5]$ and $[5, 0]$, respectively, with an equal covariance matrix of $\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.
- **Label Assignment**: The data points were labeled as +1 and -1 to distinguish between the two classes.
- **Data Partitioning**: The data was shuffled and split equally into training and test sets, with 200 points in each set.
- **Perceptron Implementation**:
  - The weight vector was initialized randomly.
  - A learning rate of 0.002 was used, and the maximum number of iterations was set to 1000.
  - During each iteration, a random data point was selected. If the point was misclassified, the weight vector was updated.
- **Evaluation**: The accuracy of the model was evaluated on both the training and test sets. The scikit-learn Perceptron model was also trained and tested for comparison.

2.1.1 Results

The initial accuracy on the training set was 76.50%. After training, the manually implemented perceptron achieved 100.00% accuracy on the training set and 99.50% accuracy on the test set, indicating successful convergence. The scikit-learn Perceptron model achieved 100.00% accuracy on both the training and test sets, demonstrating perfect classification on the linearly separable dataset. The plot Figure 2 (a) depicting

the training process of the perceptron algorithm shows both training and test accuracies increasing over the iterations. The two lines rise steadily, illustrating that the model is effectively learning and adjusting its decision boundary during the training process.



(a) (b) (c)

Figure 2: Training process of a perceptron algorithm.

2.2 Analysis of Modified Problem

The problem was modified by setting the means of the two classes at $m1 = \begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix}$ and $m2 = \begin{bmatrix} 10.0 \\ 10.0 \end{bmatrix}$, while keeping the covariance matrices equal and the same as before. The goal is to determine if the perceptron, as previously implemented, can correctly classify the data generated under these conditions.

In testing the perceptron with the new means, the model struggled to achieve high accuracy, especially on the test set. The initial accuracy was significantly lower compared to the original problem setup. This indicates that the perceptron was unable to linearly separate the two classes effectively. To address this issue, a bias term was added to the input data by appending a column of ones. This helps the perceptron learn a decision boundary that is not restricted to passing through the origin, enabling it to better separate the classes.

2.2.1 Results

The initial accuracy on the training set was 41.00%, indicating that the initial random weights were not effective in classifying the two classes correctly. After training, the manually implemented perceptron achieved 99.00% accuracy on the training set and 100.00% accuracy on the test set. This demonstrates that the algorithm effectively adjusted the weights to separate the two classes. The scikit-learn Perceptron model achieved 100.00% accuracy on both the training and test sets, confirming that the dataset is linearly separable and can be perfectly classified.

The plot Figure 2 (b) depicting the training process of the perceptron algorithm shows both training and test accuracies increasing steadily over the iterations. The two lines rise, demonstrating the model's capability to learn and adjust the decision boundary effectively as training progresses.

2.3 Perceptron Classification on Wine Quality Dataset

The objective of this section is to classify a two-class problem using the Wine dataset from the UCI Machine Learning Repository. The dataset consists of various chemical properties of wines, such as acidity, alcohol content, and pH levels, along with a quality rating. For this experiment, the quality ratings are binarized into two classes: high quality and low quality, to simplify the classification task. The aim is to use these features to build a perceptron model that effectively distinguishes between the two classes. We apply our own implementation of the perceptron algorithm and compare its performance with any quoted results on this dataset by other researchers.

The Wine dataset was downloaded and pre-processed using the Pandas library for efficient data manipulation. The perceptron algorithm was then trained using the dataset with a training-test split to evaluate its performance. The algorithm was implemented with a fixed number of iterations and a learning rate optimized for the dataset.

2.3.1 Results

From Figure 2 (c), it can be seen that the perceptron algorithm achieved 85.98% accuracy on the training set and 85.50% accuracy on the test set. The scikit-learn Perceptron model achieved an accuracy of 88% on the training set and 85% on the test set, which is consistent with our implementation.

The results show that while the perceptron can achieve reasonable performance on this dataset, it may not reach the highest reported results due to the simplicity of the algorithm and the linear separability limitations of the dataset. when compared to other researchers' results using more sophisticated models like decision trees or support vector machines (SVMs), which often achieve over 90% accuracy, the perceptron's performance is slightly lower. Further tuning of the hyperparameters or using a more complex model may be required for better performance.