CmpE 462 : Machine Learning

Assignment #1 Report

Student Name & Surname & ID: **Emre Girgin & 2016400099**

Index

1. Assignment Description

In this assignment students are asked to implement **Perceptron Learning Algorithm (PLA)** for Part 1 and **Linear Regression** for Part 2. For the rest of the report, the algorithm name and the part number will be used interchangeably.

- ## Perceptron Learning Algorithm:

In this part the students are asked to create random points in the 2D space. Those points are separated by the function y = -3x + 1 into two classes. For those of the points satisfy $y < -3x + 1$ are considered in class 0 and the ones satisfy $y > -3x + 1$ are labeled as class 1. The goal is to find a classifier using PLA that separates all the data points regarding their class id. This part consists of three steps. The only difference between the steps is the datasize, which are 50, 100, and 5000, respectively.

- ## Linear Regression:

In this part students are expected to implement a Linear Regression Algorithm that predicts a target value for each sample in the dataset provided. Dataset consists of a thousand samples and each sample in the first dataset has a hundred features and five hundred features in the second dataset. The goal is to predict the target value, which is given in the last column, using these 100 or 500 features (depending on the dataset) for each sample. While optimizing the classifier, the students are expected to implement either the closed form solution of the algorithm or an iterative approach to find the optimal solution (weights). This part also consists of three steps. For the first step, the first dataset is used. For the second step, the second dataset is used. For the third step, the second dataset is used and L2 Regularization is added.

2. Methods Used

- Part 1
    - Data Generation

In order to create a dataset with size N, I've sampled from uniform distribution N samples for each feature, x and y. (In total we made 2N sampling) Then matched those lists of samples element-wise to have N 2D points. The points satisfying $y < -3x + 1$ are labeled as class -1 rather than 0 in order to have a clean implementation of PLA while updating weights. However, in the report I'll call them as class 0 for convenience. For those of the points satisfying $y > -3x + 1$ are labeled as class 1 as usual.

In addition, I've added a third feature to each datapoint (in addition to x and y). This new feature is called bias and initialized with 1 as default. Since each datapoint becomes 3-vectors (excluding the label), the weight vector is a 3-vector. This bias term allows the classifier to become affine rather than linear. If a classifier is linear, it has to pass through the origin. However, in our case our separator ($y = -3x + 1$) is not linear. Thus, we need this bias term to correctly classify the dataset.

    - Training

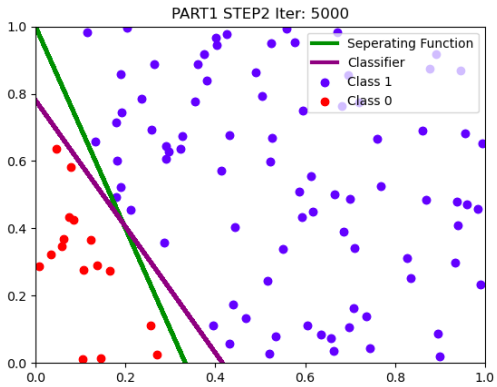
In the training phase, the algorithm provided in the lecture slides is directly used. Since the PLA is error intolerant and our separator is affine, weight initialization does not affect the performance much. Thus zero initialization is adopted.

I've run the algorithm for 5000 steps as default. At each iteration the whole dataset is iterated. For each data point the dot product of current weights and the data vector is calculated and considered as the prediction. If the prediction is below the zero, this means the algorithm predicted that sample as class 0. If the prediction is above the zero, otherwise (class 1). If there is a misprediction, the multiplication of the data sample and its label is added to the current weights and moved to the next iteration. (Note that I represented the class 0 with -1 in the code so that data point-label multiplication is not 0 for those cases.) If it is correctly classified, the PLA moves to the next data sample.

At the beginning of the training, the PLA misclassifies the first sample of the dataset consistently. To create variety while updating weights, the dataset is shuffled so that the algorithm updated its weights for a different datapoint each iteration.

- Delta Margin

We can provide a delta margin to the PLA so that its predictions are considered to be true if predictions absolute value exceeds a threshold (delta). This allows the classifier to have a more centered position between closest data points.



For example take the classifier on the left. It classifies all the points correctly but see that it is possible due to lack of some data points. Adding a delta margin to the PLA fixes this problem even if the data size is small.

Delta margin is introduced to the PLA but it is given 0.1 as default. See the code to change it.

Part 2

- Data Preparation

The provided datasets have 1000 samples and 100 or 500 features for each sample. The reason is to prevent overflow during computing. Also the bias vector is added to each sample like I do in the PLA part. Initial bias value is one by default. Also train-test split is applied. 20% of the dataset is splitted for the test data.

- Loss Function

The loss function is adopted from the lecture slides, which is a least-square problem.

$$\ell(\mathbf{w}) = \frac{1}{2}\|X\mathbf{w} - \mathbf{t}\|_2^2 = \frac{1}{2}(\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{w}^T X^T \mathbf{t} + \mathbf{t}^T \mathbf{t})$$

W is the weights, X is the data point, and t is the target value.

- Closed Form Solution

Since our loss function is a convex function, we can find the optimal **w*** by taking its derivative with respect to weights (w) and plugging the values in it.

- $\nabla \ell(\mathbf{w}) = X^T X \mathbf{w} - X^T \mathbf{t}$
- Set gradient to zero, we obtain

$$\mathbf{w}^* = (X^T X)^{-1} X^T \mathbf{t}$$

- Regularization

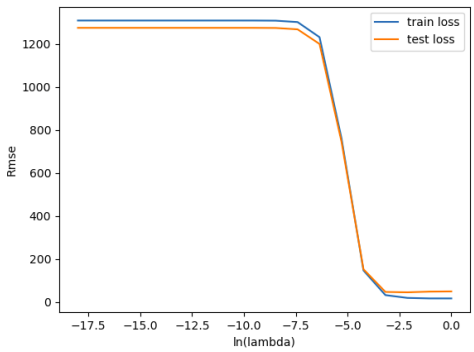
When regularization term is introduced it is added to the loss function so that loss function becomes like this.

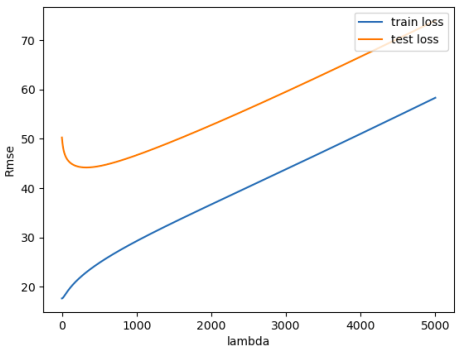
$$\frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

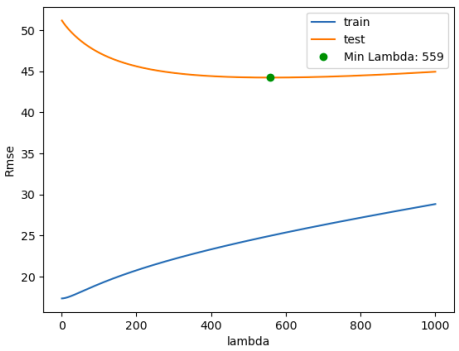
This results in a change in the gradient and the optimal solution as well.

$$\mathbf{w}^* = (X^T X + \lambda I)^{-1} X^T \mathbf{t}$$

Those formulas are directly implemented.







The regularization term is chosen by a linear search, shown in the figures above. The minimum RMSE score for the test set is obtained when lambda is 559

- Iterative Solution

I've implemented the iterative solution as well. However, its experiments are not included. Uncomment the necessary parts from the code to see the results. For the iterative solution, the gradient of the loss function with respect to weights is calculated and subtracted from the current weights with some scaling (learning rate).

Weights := Weights - (learning rate) * gradient(L,w)

- Loss Function

For this implementation, I have divided the loss value to the data size N, so that I obtained per sample loss.

- Gradients

The gradient of the loss function with respect to the weights are taken from the lecture slides and implemented directly. Note that X has the shape (N, (d+1)) where d is the number of features and plus one represents the bias. The output of the computation below has the shape of ((d+1), 1) which represents the gradients for each feature and the bias.

- $\nabla \ell(\mathbf{w}) = X^T X \mathbf{w} - X^T \mathbf{t}$

- Regularization

When regularization term is introduced, both the loss function and the gradient is changed. Loss becomes like this.

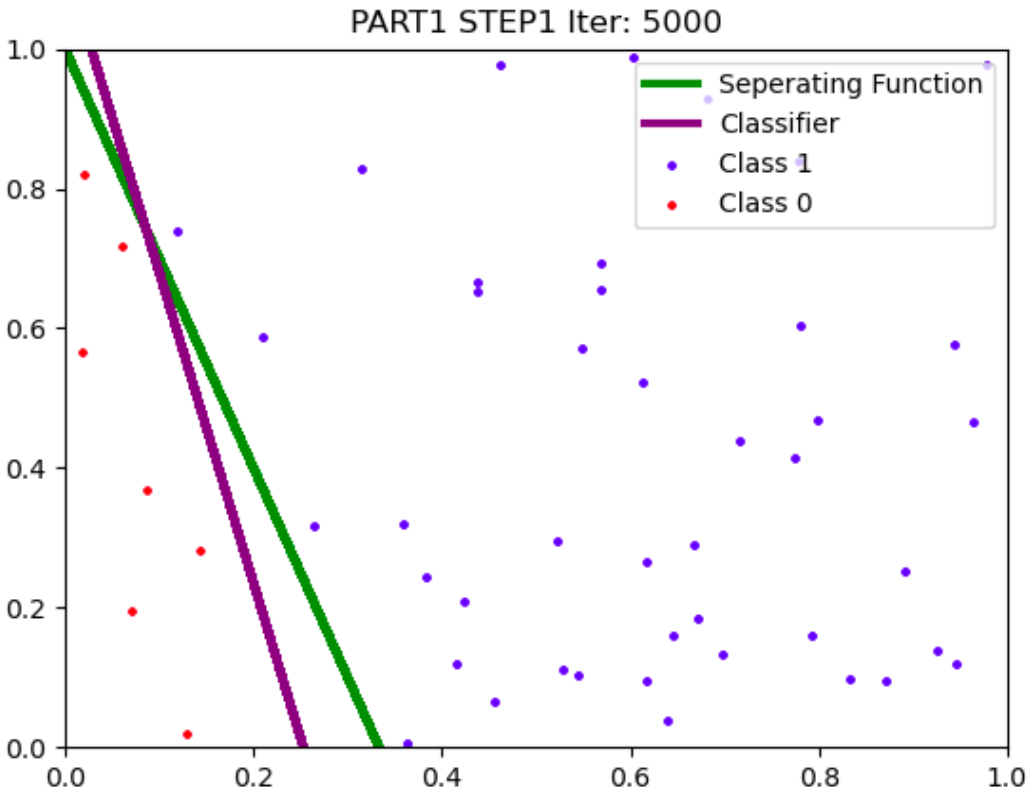
$$\frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

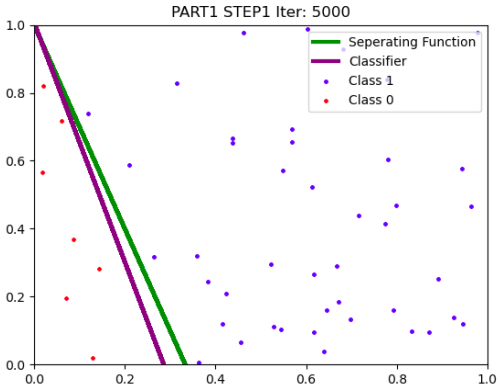
Also the gradient is summed with lambda (regularization coefficient) times weights.

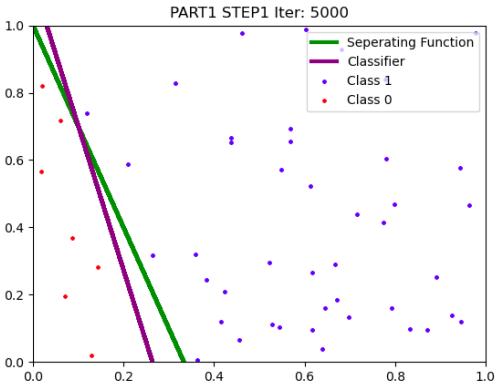
3. Experiments

- Part 1
  - Step 1

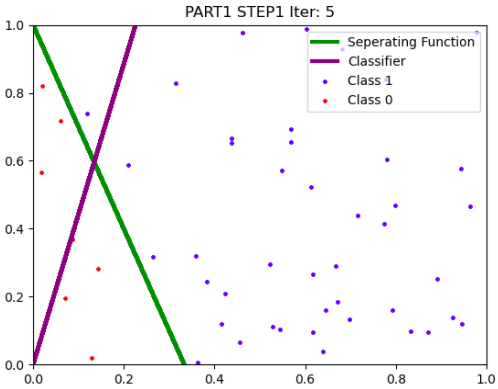
After 5000 iterations the result is on the left. Classifies all the points correctly.

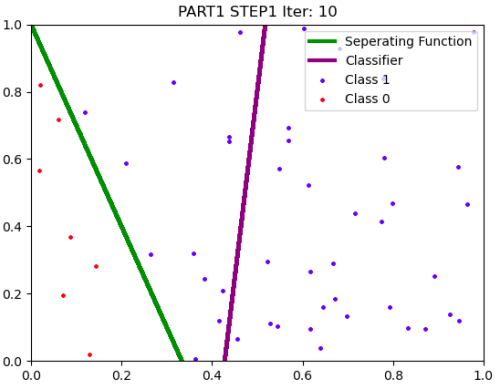


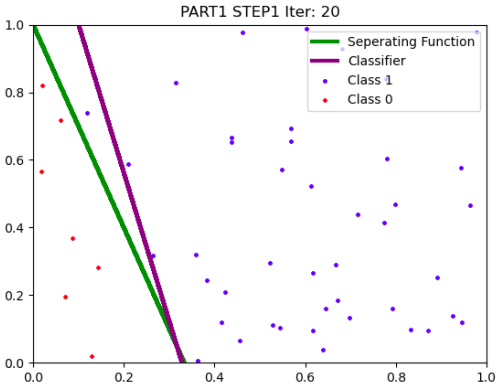


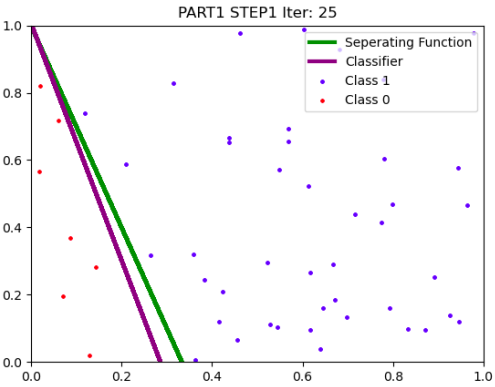


The difference between classifiers when delta margin is used (right). **delta=10** I did not tune the delta parameter. The value 10 is chosen for demonstrative purposes.





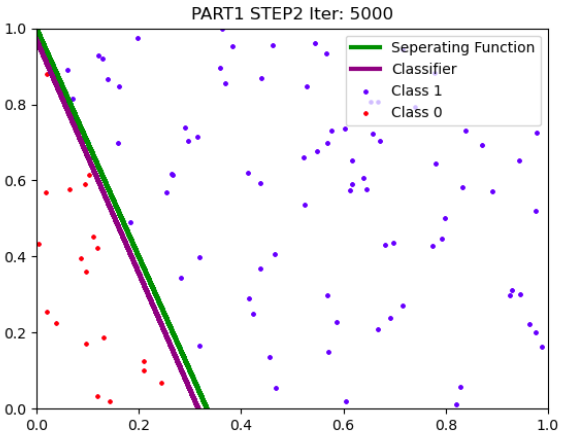


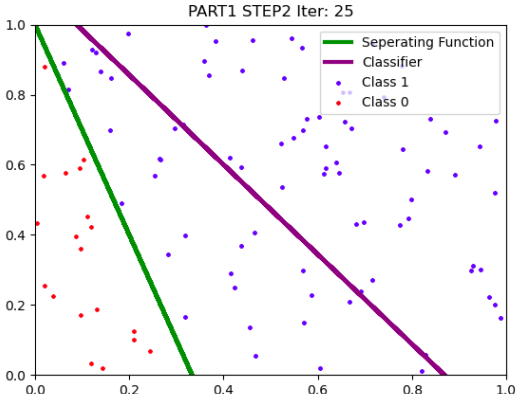


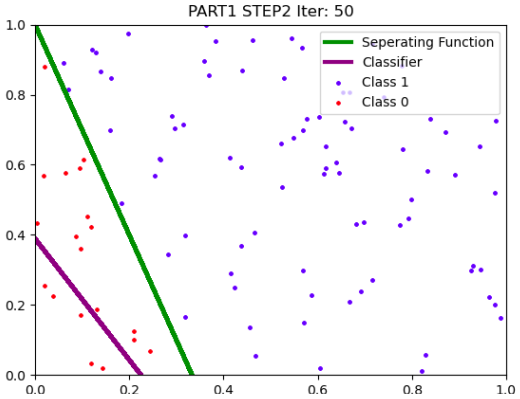
Position of the classifier during the iterations. After 25 iterations, it fitted almost perfectly and classifies all the points correctly.

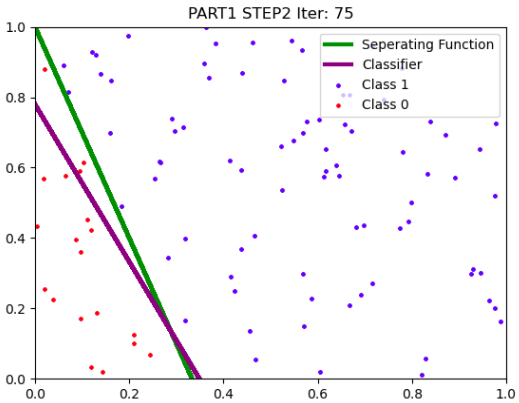
- Step 2

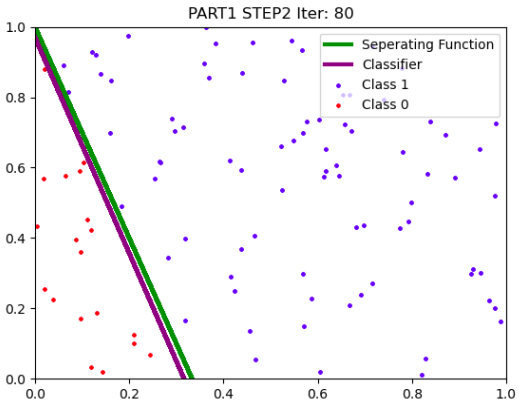
After 5000 iterations the classifier predicted all the data points correctly.





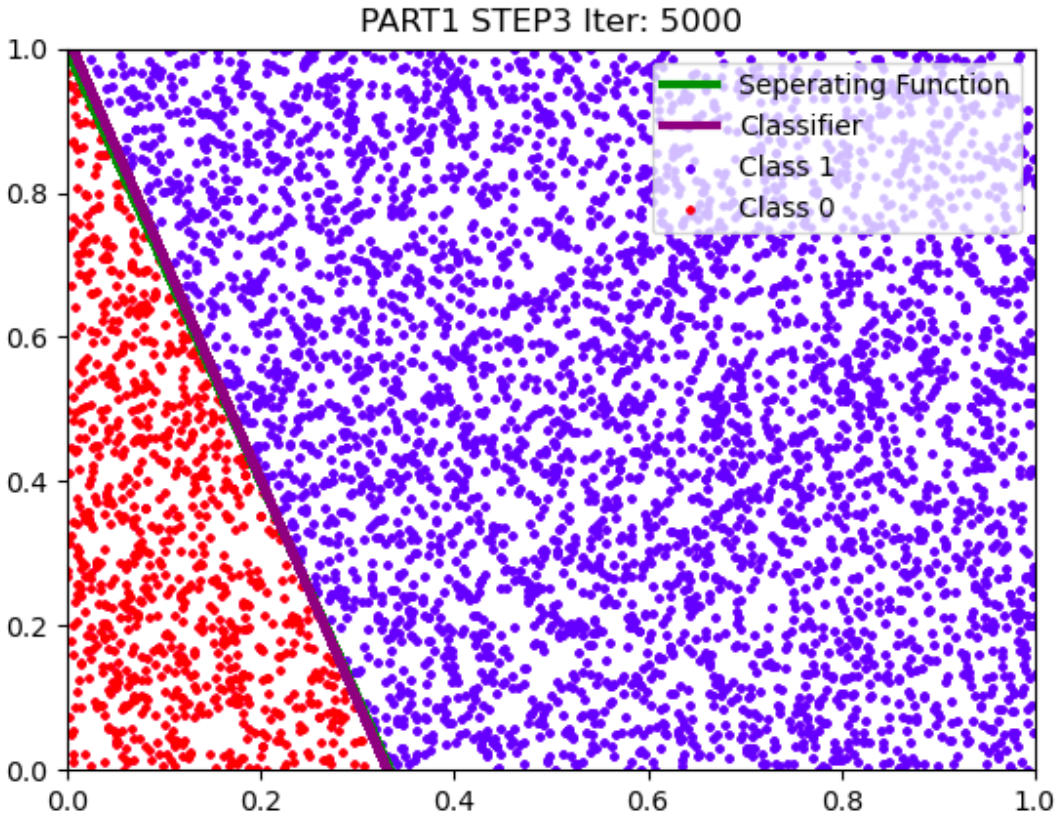




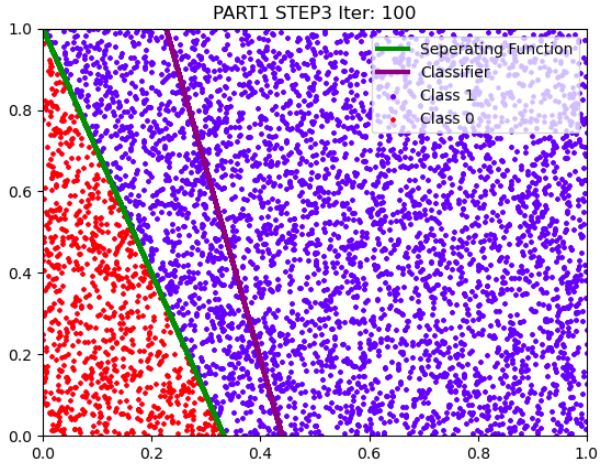


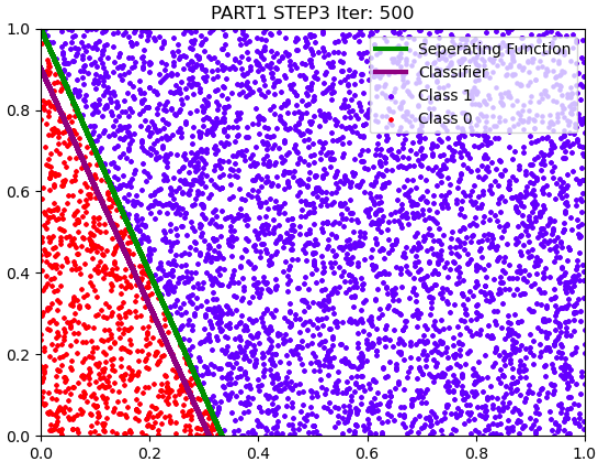
After 80 iterations PLA fitted 100 data points. Note that it fitted 50 points in 25 iterations.

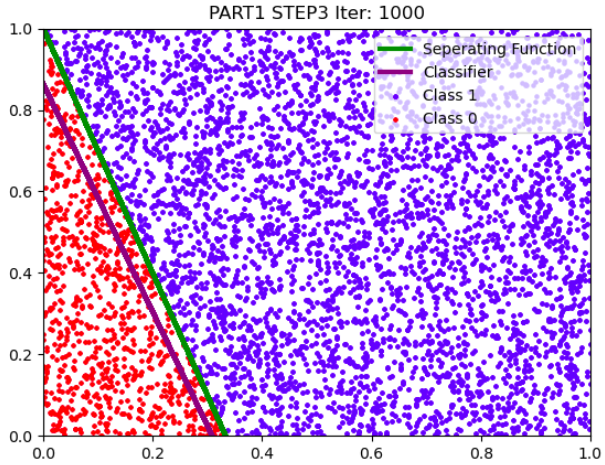
- Step 3

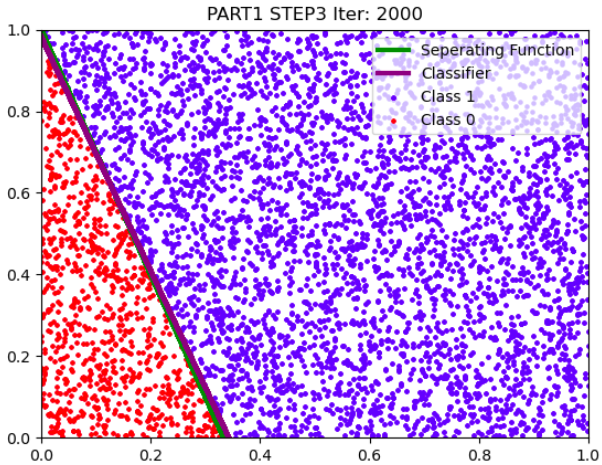


When the data size is 5000, the classifier fits almost perfectly to the separating function.

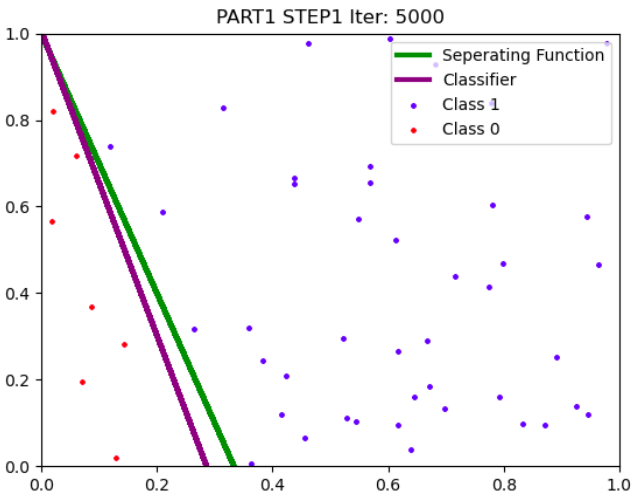


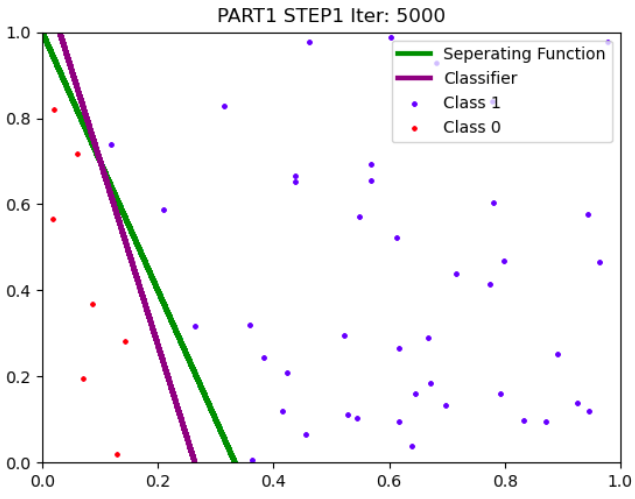






The PLA fitted to 5000 data points after around 2000 iterations. Note that it fitted to 100 points in 80 iterations.





The difference between classifiers when delta margin is used (right). **delta=10**
I did not tune the delta parameter. The value 10 is chosen for demonstrative purposes.

- Part 2

I've also implemented the iterative approach but it takes so much time to converge. Thus its experiments are not included.

  - Step 1
    - Time to complete part2_step1: 1msec
    - RMSE for Train Set: 26.32912
    - RMSE for Test Set: 30.16589

  - Step 2
    - Time to complete part2_step2: 23msec
    - RMSE for Train Set: 17.34493
    - RMSE for Test Set: 51.24355

  Step 3
    - Time to complete part2_step3: 25msec
    - RMSE for Train Set: 24.97703
    - RMSE for Test Set: 44.22877

4. Conclusion

For the PLA algorithm, it is obvious that the iterations needed to converge increases as the data size increases.

For the linear regression, the time difference between step 1 and step 2 is caused by the difference between the number of features. While dataset 2 has 500 features and the dataset 1 has 100 features, step 2 is 23 times slower than step 1. The difference between step 2 and step 3 is both time duration and the Root Mean Squared Errors. While regularization adds 2 ms overhead to the model, it makes test error smaller. We see that the training error increases but we are looking for a more generalized model, which has a lower test set error. (Test set should have been called the validation set since we tuned our hyperparameter lamba according to it. However, this is neglectable for the sake of the assignment.)