Indian Institute of Technology Bombay

Course: Foundations of Machine Learning CS 725

Experimental Verification of Perceptron Convergence Algorithm

Submitted By

Adithi Roy Chowdhury (25M2167)
Rishi Raj Vishwakarma (25M2161)
Abhishek Pal (25M2160)
Deepak Mewada (25M2158)
Samridh Agarwal (25M2156)

Academic Year: 2025–2026

Under the Guidance of:

Dr. Abir De

Abstract

The perceptron is one of the earliest linear classifiers and comes with a classical convergence theorem: if the training data are linearly separable, the perceptron learning algorithm converges in a finite number of updates, with a worst–case mistake bound of order $\left(\frac{R}{\gamma}\right)^2$, where R is the radius of the dataset and γ is the geometric margin. This project presents an experimental verification of this convergence behaviour.

We first construct two-dimensional synthetic datasets using `make_classification`, controlling class separation to obtain both linearly separable and non-separable scenarios. Labels are encoded in $\{-1, +1\}$ and a standard perceptron with an explicit bias term is implemented. During training, we record the sequence of weight and bias updates and visualize the evolution of the decision boundary over the training samples, including an animation that shows how misclassified points successively rotate and translate the separating hyperplane until a stable separator is reached in the separable case.

For separable datasets, we empirically estimate the radius R and the margin $\hat{\gamma}$ of the final classifier, compute the theoretical upper bound $\left(\frac{R}{\hat{\gamma}}\right)^2$ on the number of mistakes, and compare it with the actual number of updates observed. Our experiments confirm that the perceptron converges in finitely many steps and that the observed number of updates is significantly smaller than the conservative theoretical bound. Additional experiments on overlapping (non-separable) data and a hinge-loss-based variant further illustrate the limitations of the classical theorem and the behaviour of perceptron-like algorithms beyond the separable setting.

Introduction

The perceptron is a linear binary classifier that updates its parameters using misclassified training examples. The classical *Perceptron Convergence Theorem* states that if the training data are linearly separable, then the perceptron learning algorithm converges to a separating hyperplane in a finite number of updates. The maximum number of mistakes made during training can be bounded in terms of the radius of the dataset and the geometric margin between the two classes.

This project experimentally verifies this theorem by implementing the perceptron from scratch, training it on synthetically generated two–dimensional datasets with controlled separability, recording the number of updates until convergence, and comparing it with the theoretical mistake bound. We also study the perceptron on non–separable data, showing that the algorithm does not converge in this case, thereby underscoring the necessity of the separability assumption.

Theory

Perceptron Model

The perceptron is a linear classifier for binary labels $y \in \{-1, +1\}$. Given an input vector $x \in \mathbb{R}^d$, the perceptron predicts

$$\hat{y} = \operatorname{sign}(\langle w, x \rangle + b),$$

where $w \in \mathbb{R}^d$ is the weight vector and $b \in \mathbb{R}$ is the bias term.

Learning Rule

During training, whenever the perceptron misclassifies a sample (x_i, y_i), the parameters are updated as

$$w \leftarrow w + y_i x_i, \qquad b \leftarrow b + y_i.$$

No update is made on correctly classified samples.

Linear Separability

A dataset $\{(x_i, y_i)\}_{i=1}^n$ is called *linearly separable* if there exists some (w^*, b^*) such that

$$y_i(\langle w^*, x_i \rangle + b^*) > 0 \quad \forall i.$$

Define the *margin*

$$\gamma = \min_i \frac{y_i(\langle w^*, x_i \rangle + b^*)}{\|w^*\|}$$

and let the data radius be

$$R = \max_i \|x_i\|.$$

Perceptron Convergence Theorem

Theorem. If the dataset is linearly separable with margin $\gamma > 0$, then the perceptron learning algorithm makes only finitely many mistakes, and the total number of updates T satisfies

$$T \leq \left(\frac{R}{\gamma}\right)^2.$$

Hence, the algorithm converges to a separating hyperplane in a finite number of steps.

Non–Separable Case

If the dataset is not linearly separable, no finite update bound exists and the perceptron may continue updating indefinitely, oscillating without reaching convergence.

Proof of the Perceptron Convergence Theorem

Step 1: The Perceptron Vector Makes Progress Toward the Optimal Separator
Assume the perceptron makes an update on (x_i, y_i):

$$w^{(t+1)} = w^{(t)} + y_i x_i.$$

Take the dot product with the optimal weight vector w^* (where $\|w^*\| = 1$):

$$w^{(t+1)} \cdot w^* = w^{(t)} \cdot w^* + y_i(x_i \cdot w^*).$$

Since the margin is at least γ, we have

$$y_i(x_i \cdot w^*) \geq \gamma.$$

Thus,

$$w^{(t+1)} \cdot w^* \geq w^{(t)} \cdot w^* + \gamma.$$

By induction, after T updates:

$$w^{(T)} \cdot w^* \geq T\gamma.$$

Step 2: The Norm of the Perceptron Vector Cannot Grow Too Quickly
We compute:

$$\|w^{(t+1)}\|^2 = \|w^{(t)} + y_i x_i\|^2 = \|w^{(t)}\|^2 + \|x_i\|^2 + 2y_i\big(w^{(t)} \cdot x_i\big).$$

Since the point is misclassified:

$$y_i\big(w^{(t)} \cdot x_i\big) \leq 0.$$

Thus,

$$\|w^{(t+1)}\|^2 \leq \|w^{(t)}\|^2 + \|x_i\|^2 \leq \|w^{(t)}\|^2 + R^2.$$

By induction:

$$\|w^{(T)}\|^2 \leq TR^2.$$

Combining:

$$T^2\gamma^2 \leq TR^2.$$

Divide both sides by $T\gamma^2$:

$$T \leq \frac{R^2}{\gamma^2}.$$

Algorithm

Perceptron Learning Algorithm (with Bias Term)

We implement the standard perceptron learning algorithm with an explicit bias term. The training data are $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$.

Inputs

- Training set $\{(x_i, y_i)\}_{i=1}^n$
- Learning rate $\eta > 0$ (in the code: `lr = 1.0`)
- Maximum number of epochs $T_{\max}$ (in the code: `max_epochs = 1000`)

Outputs

- Final weight vector $w \in \mathbb{R}^d$
- Final bias $b \in \mathbb{R}$
- History of parameters after each update: $(w^{(t)}, b^{(t)})$, used for visualization

Pseudocode

```
Initialize:
  • $w \leftarrow 0 \in \mathbb{R}^d$
  • $b \leftarrow 0$
  • history $\leftarrow [\,]$   (empty list to store $(w, b)$ after each update)
For epoch $= 1$ to $T_{\max}$:
  1. Set errors $\leftarrow 0$.
  2. For $i = 1$ to $n$:
     (a) Compute the activation
             $a_i = w \cdot x_i + b$.
     (b) Predict
             $\hat{y}_i = \begin{cases} +1, & \text{if } a_i \geq 0, \\ -1, & \text{if } a_i < 0. \end{cases}$
     (c) If $\hat{y}_i \neq y_i$ (misclassification), then:
         • Update the weights:
             $w \leftarrow w + \eta\, y_i x_i$,
         • Update the bias:
             $b \leftarrow b + \eta\, y_i$,
         • Increment errors $\leftarrow$ errors $+1$,
         • Append current $(w, b)$ to history.
  3. If errors $= 0$, then:
       • Stop the training (the perceptron has converged).
Return final $(w, b)$ and the full history of updates.
```

Experimental Setup

Dataset Generation

We generate two–dimensional synthetic datasets using `make_classification` from `sklearn.datasets`. The data consist of two classes labeled $+1$ and -1, and the separability is controlled by the parameter `class_sep`.

- Each dataset contains $n = 200$ samples in $\mathbb{R}^2$.

- Labels are converted to $\{-1, +1\}$ after generation.
- We study two settings:
  1. **Linearly separable data:** high `class_sep` value.
  2. **Non–separable data:** overlapping classes (small `class_sep`).

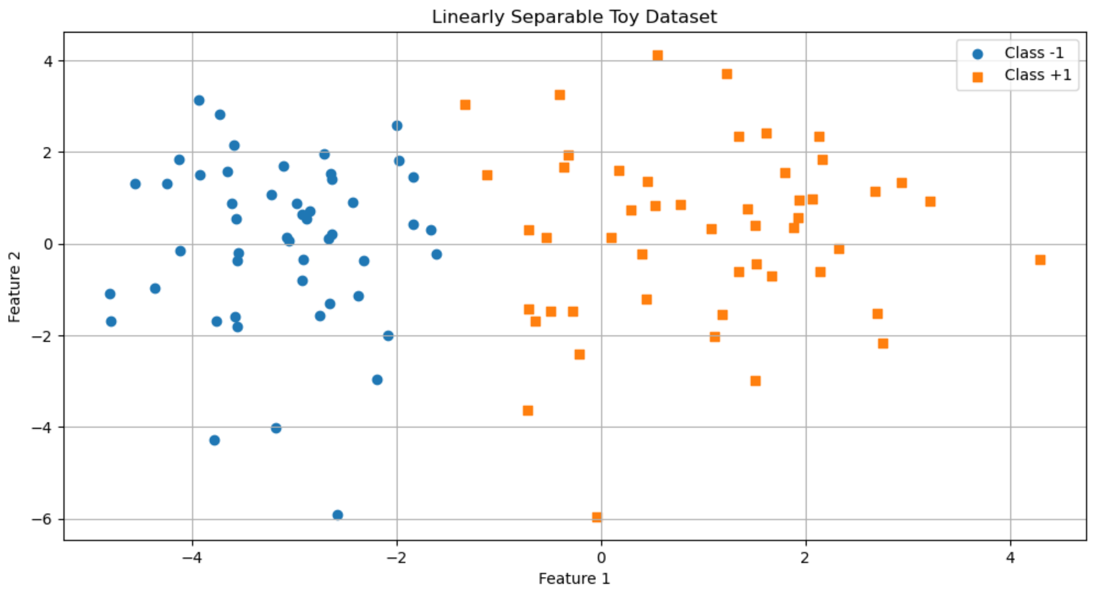


Figure 1: Generated synthetic Linearly separable dataset with two classes in $\mathbb{R}^2$.

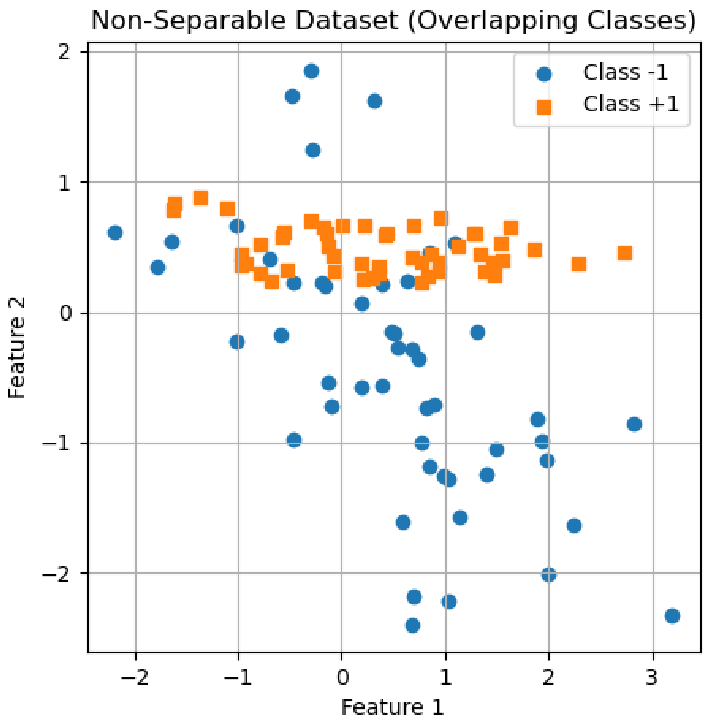


Figure 2: Generated synthetic Non-Linearly separable dataset with two classes in $\mathbb{R}^2$.

Data Generation code

Here is the code that we used to generate both linearly separable and non-linearly separable cases

Listing 1: Synthetic Linearly separable dataset generation

```
import numpy as np
import mathplotlib as plt

X, y = make_classification(
    n_samples=100,
    n_features=2,
    n_informative=1,
    n_redundant=0,
    n_classes=2,
    n_clusters_per_class=1,
    random_state=41,
    hypercube=False,
    class_sep=10
)
y_perc = np.where(y == 0, -1, 1)
variance_scale = 2.0
X = X * variance_scale
plt.figure(figsize=(12, 6))
plt.scatter(X[y_perc == -1, 0], X[y_perc == -1, 1], marker='o', label='Class
    -1')
plt.scatter(X[y_perc ==  1, 0], X[y_perc ==  1, 1], marker='s', label='Class
    +1')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.title('Linearly Separable Toy Dataset')
plt.grid(True)
plt.savefig("D:\Important Subjects\Machine Learning\Project\Report\Media")
plt.show()
```

In order to experimentally study the perceptron convergence behavior, a two–dimensional synthetic dataset is generated using the `make_classification` function from `sklearn.datasets`. The dataset consists of two linearly separable classes, where separability is controlled through a large `class_sep` parameter. The resulting labels from `make_classification` are initially encoded as $\{0, 1\}$, which are manually converted to $\{-1, +1\}$ to match the perceptron update rule. To further spread the samples in the feature space, a scaling factor is applied to the feature matrix. Finally, the data are visualized by plotting the two classes with distinct markers, providing a clear view of their separability before applying the perceptron learning algorithm.

Listing 2: Synthetic Non Separable dataset generation

```python
X_bad, y_bad = make_classification(
    n_samples=100,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_classes=2,
    n_clusters_per_class=1,
    class_sep=0.5,
    random_state=0
)

y_bad_perc = np.where(y_bad == 0, -1, 1)
plt.figure(figsize=(5, 5))
plt.scatter(X_bad[y_bad_perc == -1, 0], X_bad[y_bad_perc == -1, 1], marker='o'
    , label='Class -1')
plt.scatter(X_bad[y_bad_perc ==  1, 0], X_bad[y_bad_perc ==  1, 1], marker='s'
    , label='Class +1')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.title("Non-Separable Dataset (Overlapping Classes)")
plt.grid(True)
plt.savefig("D:\Important Subjects\Machine Learning\Project\Report\Media")
plt.show()
```

To illustrate the limitations of the perceptron convergence theorem, a non–separable dataset is generated using the `make_classification` function from `sklearn.datasets`. In this case, the data are constructed in two dimensions with both features being informative, and a small value of `class_sep` is chosen to intentionally produce overlapping classes. As before, the labels originally encoded as $\{0, 1\}$ are converted to $\{-1, +1\}$ to comply with the perceptron update rule. The resulting data are visualized using a scatter plot, where the two classes overlap significantly in the feature space, making it impossible for any linear separator to perfectly classify all points. This visualization highlights that under non–separable conditions, the perceptron fails to converge, which is consistent with the theoretical guarantees.

Training Procedure

We train the perceptron using the update rule described in the Algorithm section. The learning rate is set to $\eta = 1.0$, and training stops early when no misclassification occurs in a full pass over the data.

- Maximum epochs: `1000`
- Learning rate: `1.0`
- Bias term is updated together with weights
- After every update, the current (w, b) pair is stored to visualize the evolution of the decision boundary

Listing 3: Perceptron Trainning Function

```python
def perceptron_train(X, y, lr=1.0, max_epochs=1000):
    n_samples, n_features = X.shape
    w = np.zeros(n_features)
    b = 0.0
    history = []

    for epoch in range(max_epochs):
        errors = 0
        for i in range(n_samples):
            activation = np.dot(w, X[i]) + b
            y_pred = 1 if activation >= 0 else -1
            if y_pred != y[i]:
                w += lr * y[i] * X[i]
                b += lr * y[i]
                errors += 1
                history.append((w.copy(), b))
        if errors == 0:
            print(f"Perceptron converged in epoch {epoch + 1}")
            break

    return w, b, history

w, b, history = perceptron_train(X, y_perc, lr=1.0, max_epochs=1000)
print("Final weights:", w)
print("Final bias:", b)

plot_decision_boundary(X, y_perc, w, b,"Perceptron (Vanilla) on Separable Data
    ")
```

Once we run the code the Algorithm converged within 4 epoches and following was the final result. Using the function that plots graphs for each of the 4 eposches, we plotted the decision boundary on the input space. Here is the code snippet that we used

Listing 4: Generating plots of decision boundary after epoch

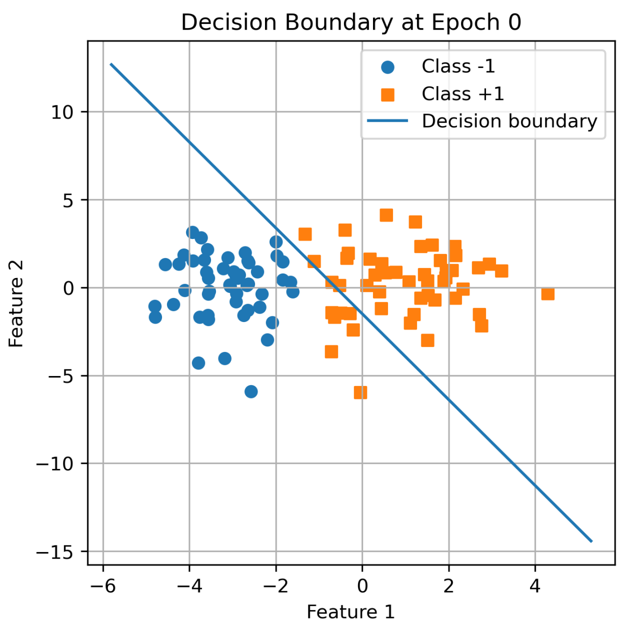
```python
import os
def plot_decision_boundary_save(X, y, w, b, title, filename):
    plt.figure(figsize=(5, 5))
    plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', label='Class -1')
    plt.scatter(X[y ==  1, 0], X[y ==  1, 1], marker='s', label='Class +1')

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    xs = np.linspace(x_min, x_max, 200)

    if abs(w[1]) > 1e-12:
        ys = -(w[0] * xs + b) / w[1]
        plt.plot(xs, ys, linestyle='-', label='Decision boundary')
    else:
        x_line = -b / (w[0] + 1e-12)
        plt.axvline(x_line, linestyle='-', label='Decision boundary')

    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(title)
    plt.legend()
    plt.grid(True)

    plt.savefig(os.path.join("D:\Important Subjects\Machine Learning\Project\
        Report\Media\Convergence plots", filename), dpi=300, bbox_inches='tight
        ')
    plt.close()
```

Results of Training

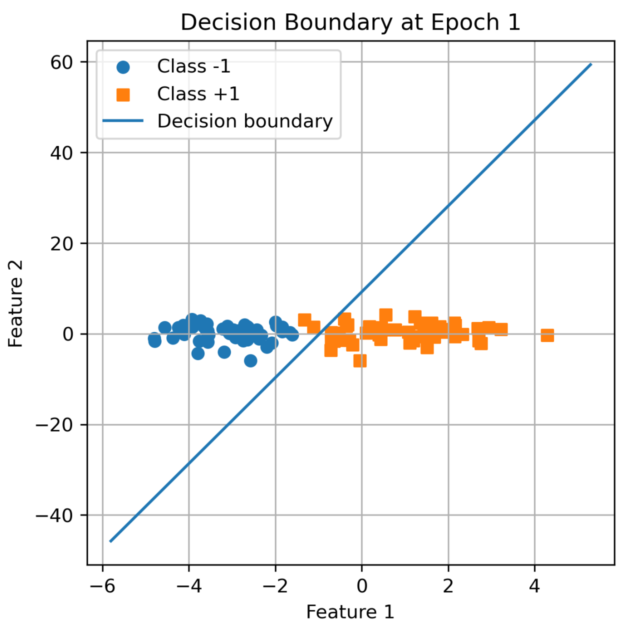
Once the trainning for loop was exited. The following results are obtained

```
Perceptron converged in epoch 4
Final weights: [7.09703775 0.62987918]
Final bias: 9.0
```

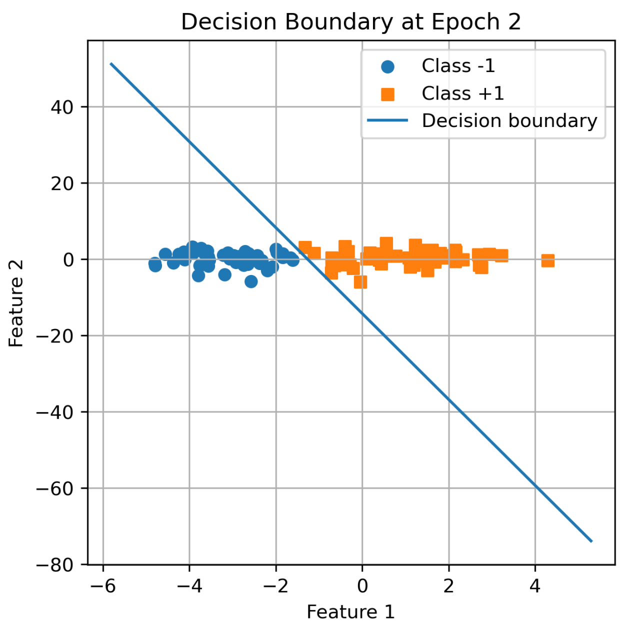
Here are the plots showing decision boundary after every epoch



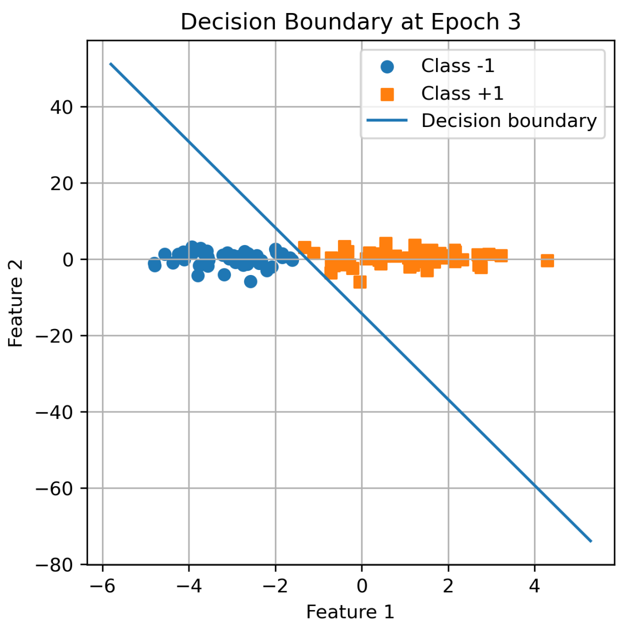
(a) Epoch 1



(b) Epoch 2



(c) Epoch 3



(d) Epoch 4

Figure 3: Evolution of the perceptron decision boundary over training epochs.

Updates in Weights and Bias terms

As we ran the algorithm it converged in 4 epoches but the total number of updates that is mentioned in Convergence theorem is 21. As Updates happens only when we have a mistake in classification, then only the update rule is called. Our code not only captures the training process but also the update history of the weights the final history array that was created once the algorithm converged is given as

Listing 5: Final History array after the algorithm

```
    history array at epoch  {2}  sample  {8}  :  [(array([3.18173012,
       4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
        2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]),
       2.0), (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857,
       1.31221853]), 2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array
       ([2.08122805, 1.95332431]), 4.0), (array([5.18645851, 0.264209  ]),
       3.0), (array([ 4.47104397, -3.36906979]), 4.0), (array([7.04722514,
       2.53839253]), 3.0), (array([6.51581261, 2.66839097]), 4.0), (array
       ([6.01817204, 1.20608633]), 5.0), (array([ 5.30919124, -0.21012546]),
       6.0), (array([4.19188526, 1.28706508]), 7.0), (array([ 4.14800223,
       -4.67478355]), 8.0), (array([ 7.90712105, -2.99617403]), 7.0), (array([
        7.54246282, -1.32163278]), 8.0), (array([6.21458314, 1.72774991]),
       9.0), (array([ 8.21434374, -0.86731136]), 8.0), (array([7.09703775,
       0.62987918]), 9.0)]
```

Why we captured the history of every update?

Capturing the history enabled us to cleary plot the decision boundary after each update. Length of the history array also gives us the total number of updates that the algorithm takes to converge that is 21. For full length updation in weights please refer to appendix of the report.

Decision Boundary after each update for all the 21 updates are shown here in appendix. Once we ploted the decision boundaries it is quite easy to see how actually the decision boundary changes with updation rule. As the algorithm converges we get the final plot beyond which the decision boundary will not change.

Margin and Theoretical Bound Estimation

For the separable dataset, we estimate the geometric margin of the learned separator and compute the theoretical mistake bound

$$\left(\frac{R}{\gamma}\right)^2,$$

where R is the maximum norm of the data points. This value is compared with the actual number of perceptron updates recorded during training.

Where R is given mathematically as

$$R = \max_i \|x_i\|$$

and

$$\gamma = \min_i \frac{y_i(\langle w, x_i\rangle + b)}{\|w\|}$$

to calculate the values of R and γ for our dataset we used the following code

Listing 6: Code to calculate Radius and γ

```
R = np.max(np.linalg.norm(X, axis=1))
print("R (max norm of samples):", R)

def estimated_margin(X, y, w, b):
    w_norm = np.linalg.norm(w)
    margins = y * (X @ w + b) / (w_norm + 1e-12)
    return np.min(margins)

gamma_hat = estimated_margin(X, y_perc, w, b)
print("Estimated margin gamma_hat:", gamma_hat)
```

To Calculate the bound suggested by Convergence theorem, we used -

```
bound = (R / (gamma_hat + 1e-12)) ** 2
```

Finally we get the results of above code blocks as

```
R (max norm of samples): 6.4447513916413
Estimated margin gamma_hat: 0.21006936522129688
Approx. theoretical upper bound on number of updates: 941.2107779492442
Actual number of updates until convergence: 21
```

Which is far lesser than the theoritical bound suggested by the Convergence Theorem.

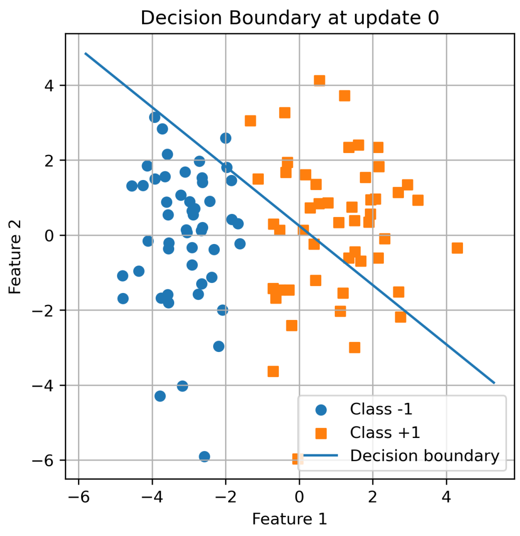
.1 Perceptron Weight and Bias updates

Here is the complete result that was obtained after each of 21 updates of the weights and bias stored in the history array as in the code.

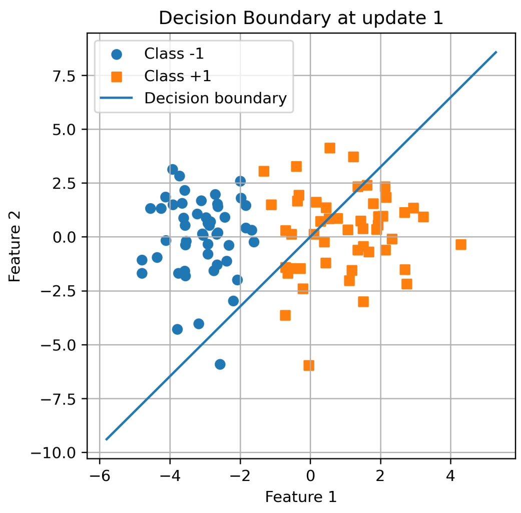
Listing 7: Full history of (w, b) updates used for animation

```
history array at epoch  {0}  sample  {3}  :  [(array([3.18173012, 4.02398429])
   , -1.0)]

history array at epoch  {0}  sample  {12}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0)]

history array at epoch  {0}  sample  {18}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0)]

history array at epoch  {0}  sample  {35}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0)]

history array at epoch  {0}  sample  {48}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0)]

history array at epoch  {0}  sample  {51}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0)]

history array at epoch  {0}  sample  {59}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0)]

history array at epoch  {0}  sample  {65}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0)]

history array at epoch  {0}  sample  {68}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0)]

history array at epoch  {0}  sample  {84}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0)]

history array at epoch  {0}  sample  {89}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0)]

history array at epoch  {0}  sample  {99}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (
   array([6.51581261, 2.66839097]), 4.0)]

history array at epoch  {1}  sample  {1}  :  [(array([3.18173012, 4.02398429])
   , -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([ 2.77318887,
   -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0), (array
   ([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]), 2.0), (
   array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805, 1.95332431]),
   4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([ 4.47104397,
   -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (array
   ([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]), 5.0)]

history array at epoch  {1}  sample  {2}  :  [(array([3.18173012, 4.02398429])
   , -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([ 2.77318887,
   -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0), (array
   ([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]), 2.0), (
   array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805, 1.95332431]),
   4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([ 4.47104397,
   -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (array
   ([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]), 5.0), (
   array([ 5.30919124, -0.21012546]), 6.0)]

history array at epoch  {1}  sample  {8}  :  [(array([3.18173012, 4.02398429])
   , -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([ 2.77318887,
   -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0), (array
   ([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]), 2.0), (
   array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805, 1.95332431]),
   4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([ 4.47104397,
   -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (array
   ([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]), 5.0), (
   array([ 5.30919124, -0.21012546]), 6.0), (array([4.19188526, 1.28706508]),
   7.0)]

history array at epoch  {1}  sample  {12}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (
   array([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]),
   5.0), (array([ 5.30919124, -0.21012546]), 6.0), (array([4.19188526,
   1.28706508]), 7.0), (array([ 4.14800223, -4.67478355]), 8.0)]

history array at epoch  {1}  sample  {17}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (
   array([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]),
   5.0), (array([ 5.30919124, -0.21012546]), 6.0), (array([4.19188526,
   1.28706508]), 7.0), (array([ 4.14800223, -4.67478355]), 8.0), (array([
   7.90712105, -2.99617403]), 7.0)]

history array at epoch  {1}  sample  {18}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (
   array([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]),
   5.0), (array([ 5.30919124, -0.21012546]), 6.0), (array([4.19188526,
   1.28706508]), 7.0), (array([ 4.14800223, -4.67478355]), 8.0), (array([
   7.90712105, -2.99617403]), 7.0), (array([ 7.54246282, -1.32163278]), 8.0)]

history array at epoch  {1}  sample  {65}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (
   array([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]),
   5.0), (array([ 5.30919124, -0.21012546]), 6.0), (array([4.19188526,
   1.28706508]), 7.0), (array([ 4.14800223, -4.67478355]), 8.0), (array([
   7.90712105, -2.99617403]), 7.0), (array([ 7.54246282, -1.32163278]), 8.0),
   (array([6.21458314, 1.72774991]), 9.0)]

history array at epoch  {1}  sample  {69}  :  [(array([3.18173012,
   4.02398429]), -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([
   2.77318887, -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0),
   (array([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]),
   2.0), (array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805,
   1.95332431]), 4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([
   4.47104397, -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (
   array([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]),
   5.0), (array([ 5.30919124, -0.21012546]), 6.0), (array([4.19188526,
   1.28706508]), 7.0), (array([ 4.14800223, -4.67478355]), 8.0), (array([
   7.90712105, -2.99617403]), 7.0), (array([ 7.54246282, -1.32163278]), 8.0),
   (array([6.21458314, 1.72774991]), 9.0), (array([ 8.21434374, -0.86731136]),
    8.0)]

history array at epoch  {2}  sample  {8}  :  [(array([3.18173012, 4.02398429])
   , -1.0), (array([ 3.1378471 , -1.93786434]), 0.0), (array([ 2.77318887,
   -0.2633231 ]), 1.0), (array([ 2.12962518, -1.94226489]), 2.0), (array
   ([4.32446697, 1.01781785]), 1.0), (array([3.62140857, 1.31221853]), 2.0), (
   array([ 3.40910773, -1.09605838]), 3.0), (array([2.08122805, 1.95332431]),
   4.0), (array([5.18645851, 0.264209  ]), 3.0), (array([ 4.47104397,
   -3.36906979]), 4.0), (array([7.04722514, 2.53839253]), 3.0), (array
   ([6.51581261, 2.66839097]), 4.0), (array([6.01817204, 1.20608633]), 5.0), (
   array([ 5.30919124, -0.21012546]), 6.0), (array([4.19188526, 1.28706508]),
   7.0), (array([ 4.14800223, -4.67478355]), 8.0), (array([ 7.90712105,
   -2.99617403]), 7.0), (array([ 7.54246282, -1.32163278]), 8.0), (array
   ([6.21458314, 1.72774991]), 9.0), (array([ 8.21434374, -0.86731136]), 8.0),
    (array([7.09703775, 0.62987918]), 9.0)]
```

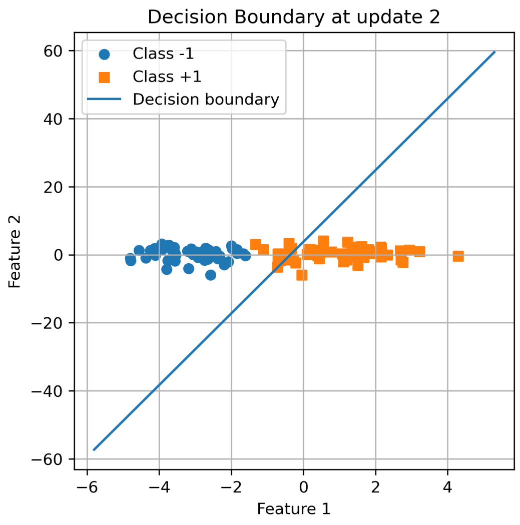
Plots of Decision Boundary



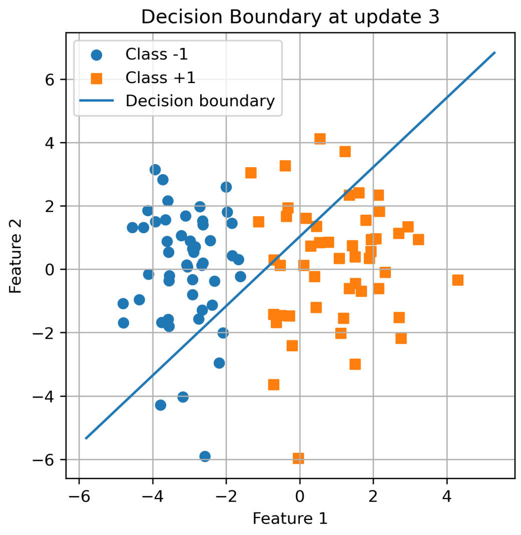
(a) Plot 1



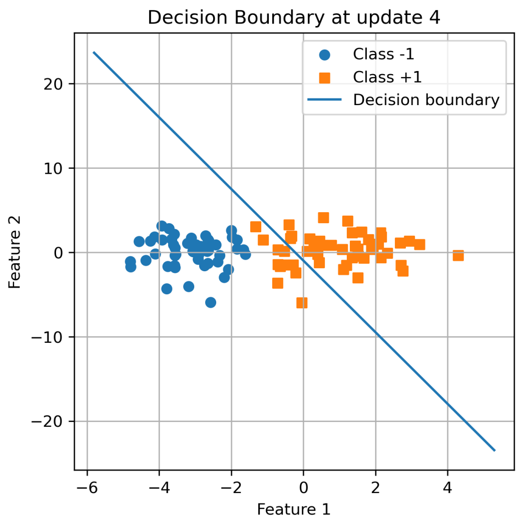
(b) Plot 2



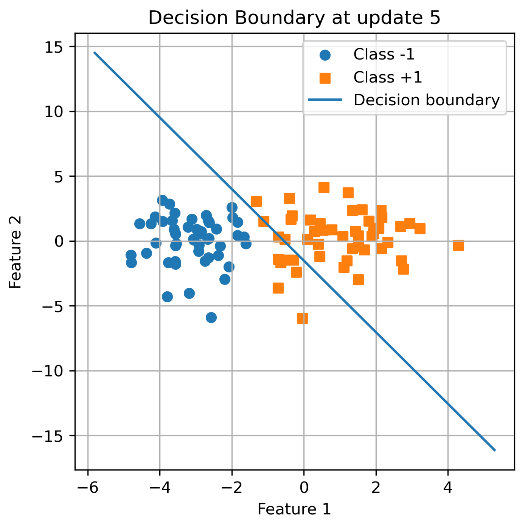
(c) Plot 3



(d) Plot 4

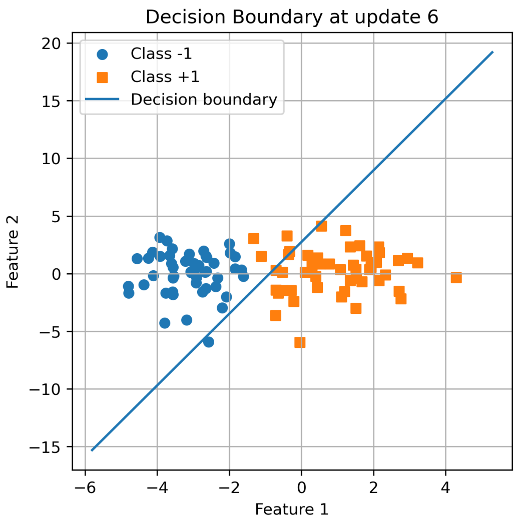


(e) Plot 5

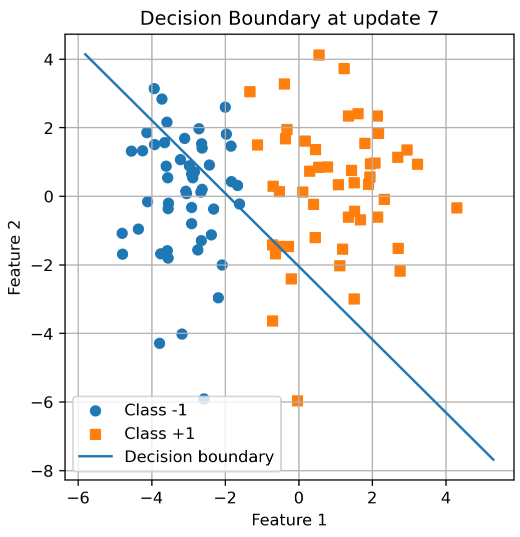


(f) Plot 6

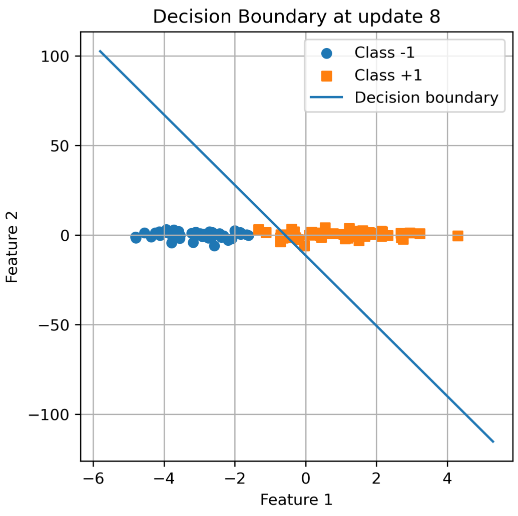
Figure 4: Evolution of Perceptron Decision Boundary



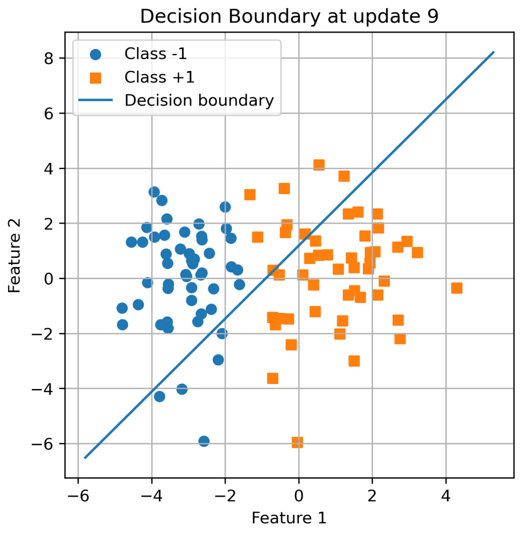
(a) Plot 7



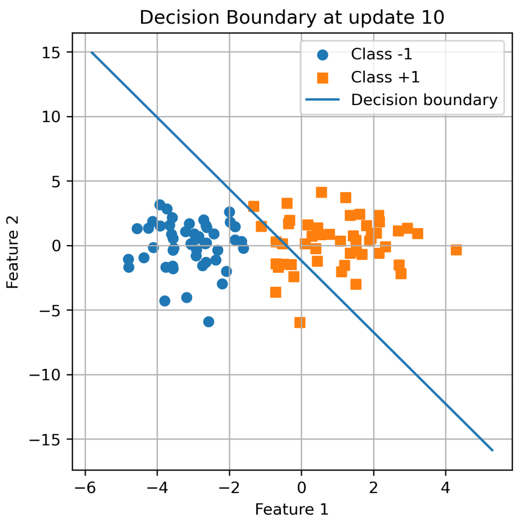
(b) Plot 8



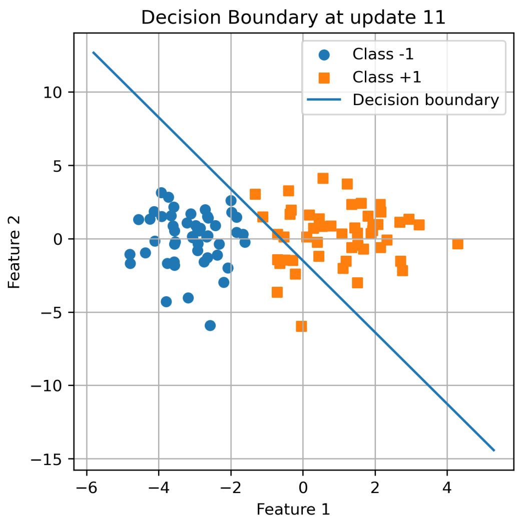
(c) Plot 9



(d) Plot 10

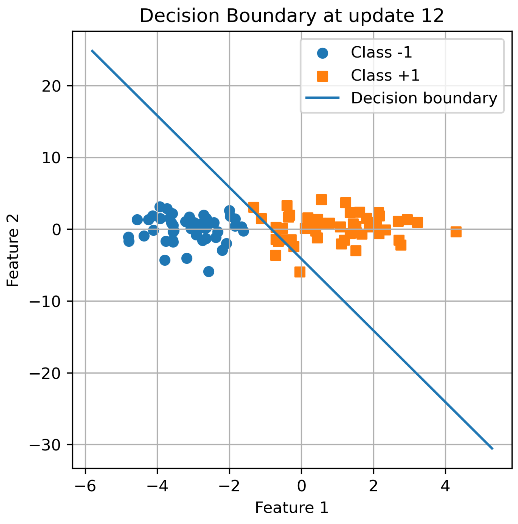


(e) Plot 11

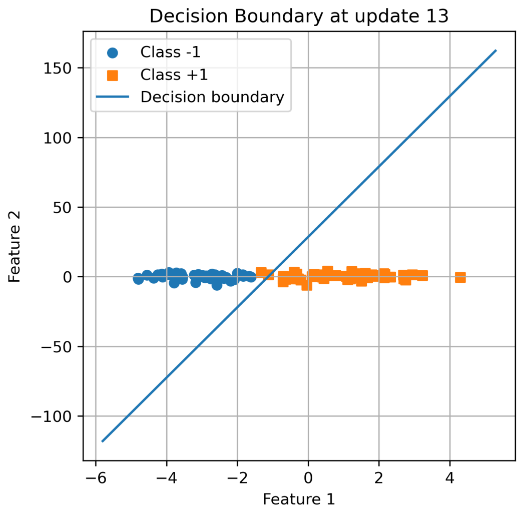


(f) Plot 12

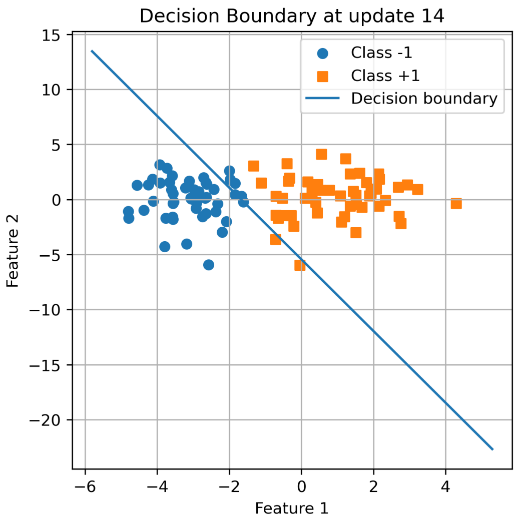
Figure 5: Evolution of Perceptron Decision Boundary



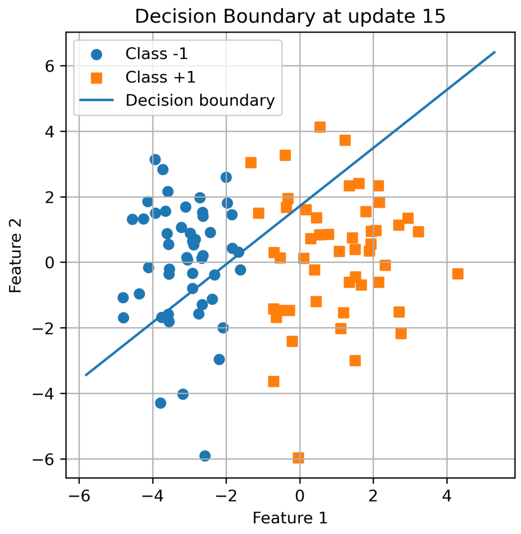
(a) Plot 13



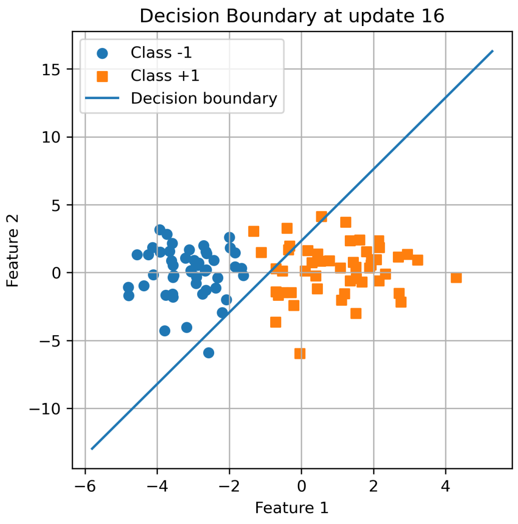
(b) Plot 14



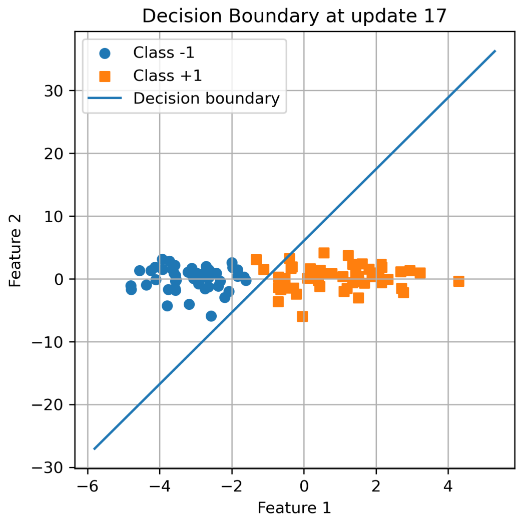
(c) Plot 15



(d) Plot 16

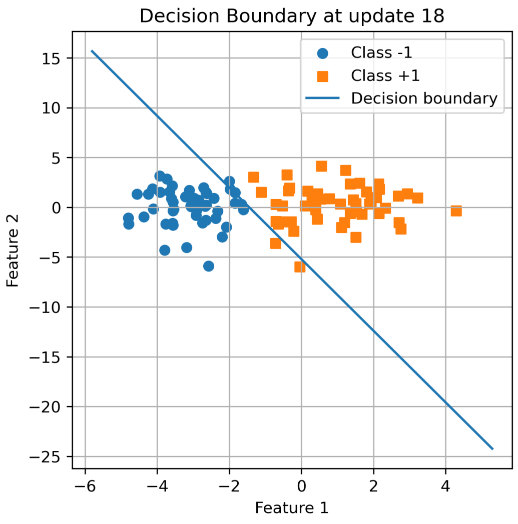


(e) Plot 17

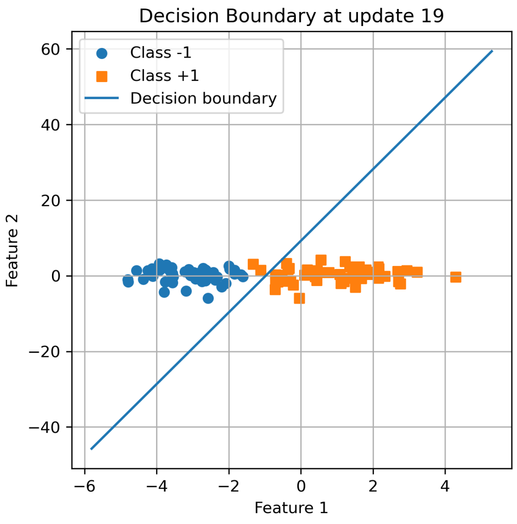


(f) Plot 18

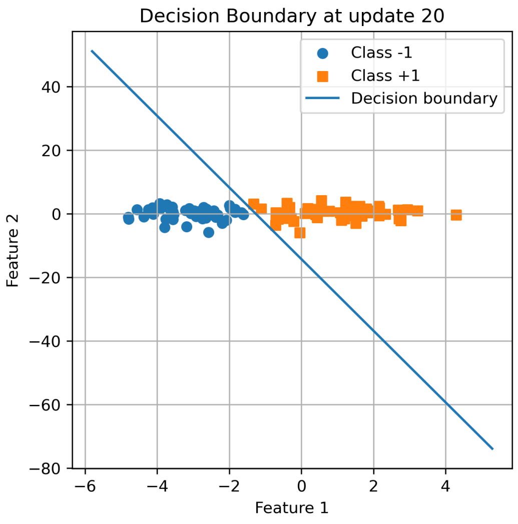
Figure 6: Evolution of Perceptron Decision Boundary



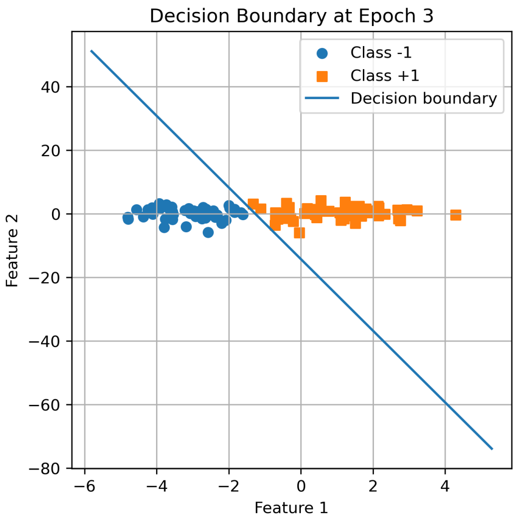
(a) Plot 19



(b) Plot 20



(c) Plot 21



(d) Plot 22

Figure 7: Evolution of Perceptron Decision Boundary

Project Repository and Source Files

The complete source code used for data generation, perceptron training, visualization, animation, and theoretical margin computation is available in the official Git repository linked below. The repository also contains:

- Python implementations of the perceptron algorithm (with bias),
- Synthetic dataset generation scripts,
- Plotting utilities for decision boundary evolution,
- Full report LaTeX source file,
- Experimental results and saved convergence plots.

GitHub Repository Link:

Repository Link