perceptron

February 10, 2023

0.1 3 Perceptron

[51]:
```
%config InlineBackend.figure_format = 'retina'
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
import random
```

0.1.1 Load the modified Iris dataset

[52]:
```
# Iris dataset.
iris = datasets.load_iris()     # Load Iris dataset.

X = iris.data                   # The shape of X is (150, 4), which means
                                # there are 150 data points, each data point
                                # has 4 features.

# Here for convenience, we divide the 3 kinds of flowers into 2 groups:
#     Y = 0 (or False):  Setosa (original value 0) / Versicolor (original value␣
↪1)
#     Y = 1 (or True):   Virginica (original value 2)

# Thus we use (iris.target > 1.5) to divide the targets into 2 groups.
# This line of code will assign:
#    Y[i] = True  (which is equivalent to 1) if iris.target[k]  > 1.5␣
↪(Virginica)
#    Y[i] = False (which is equivalent to 0) if iris.target[k] <= 1.5 (Setosa /␣
↪Versicolor)

Y = (iris.target > 1.5).reshape(-1,1).astype(np.float) # The shape of Y is␣
↪(150, 1), which means
                                # there are 150 data points, each data point
                                # has 1 target value.
Y[Y==0] = -1

X_and_Y = np.hstack((X, Y))     # Stack them together for shuffling.
np.random.seed(1)               # Set the random seed.
```

```
np.random.shuffle(X_and_Y)      # Shuffle the data points in X_and_Y array

print(X.shape)
print(Y.shape)
print(X_and_Y[0])               # The result should be always: [ 5.8  4.   1.2 
 ↪0.2  0. ]
```

```
(150, 4)
(150, 1)
[ 5.8  4.   1.2  0.2 -1. ]
```

```
/tmp/ipykernel_96/3314977914.py:17: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  Y = (iris.target > 1.5).reshape(-1,1).astype(np.float) # The shape of Y is
(150, 1), which means
```

[53]:
```
# Divide the data points into training set and test set.
X_shuffled = X_and_Y[:,:4]
Y_shuffled = X_and_Y[:,4]


X_train = X_shuffled[:100][:,[3,1]] # Shape: (100,2)
X_train = np.delete(X_train, 42, axis=0) # Remove a point for separability.
Y_train = Y_shuffled[:100]          # Shape: (100,)
Y_train = np.delete(Y_train, 42, axis=0) # Remove a point for separability.
X_test = X_shuffled[100:][:,[3,1]]  # Shape: (50,2)
Y_test = Y_shuffled[100:]           # Shape: (50,)
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

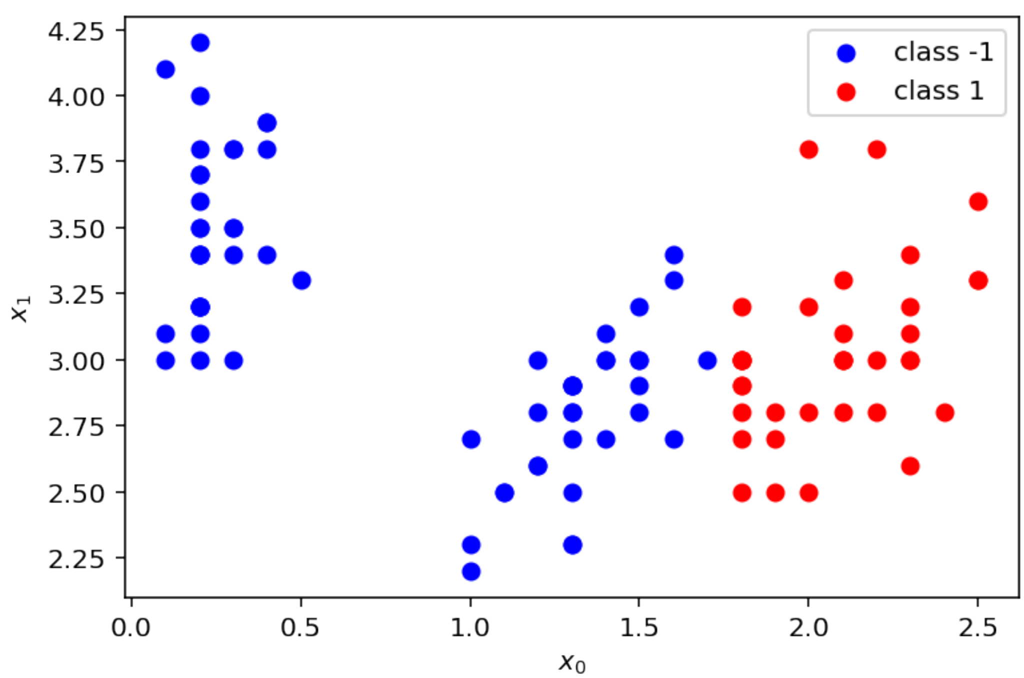
```
(99, 2)
(99,)
(50, 2)
(50,)
```

0.1.2 Visualization

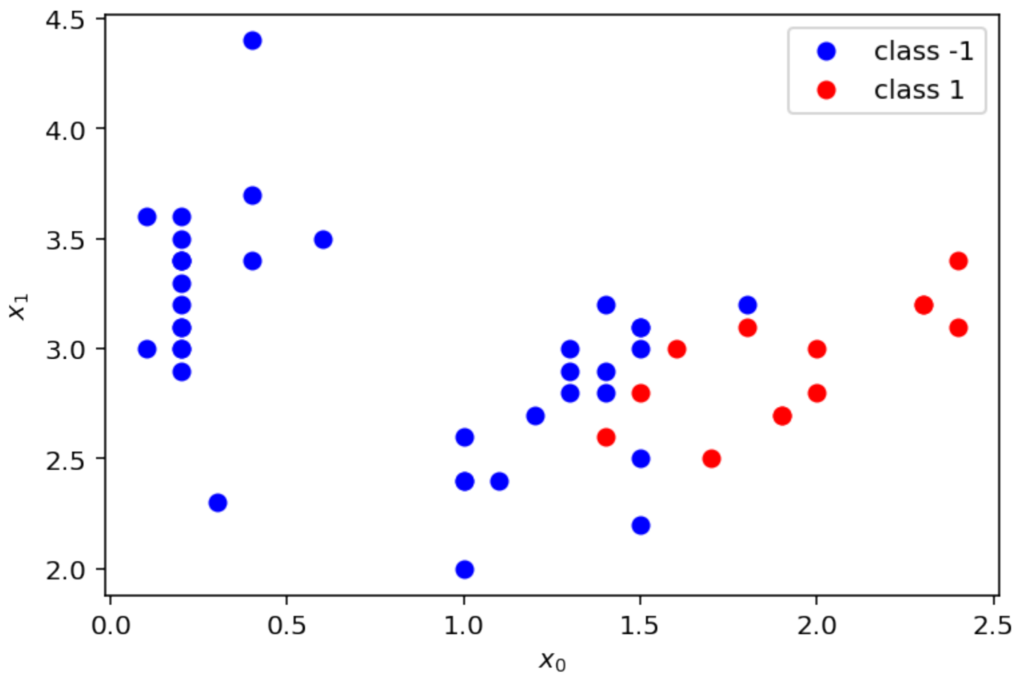
[54]:
```
def vis(X, Y, W=None, b=None):
    indices_neg1 = (Y == -1).nonzero()[0]
    indices_pos1 = (Y == 1).nonzero()[0]
    plt.scatter(X[:,0][indices_neg1], X[:,1][indices_neg1],
                c='blue', label='class -1')
```

```
    plt.scatter(X[:,0][indices_pos1], X[:,1][indices_pos1],
                c='red', label='class 1')
    plt.legend()
    plt.xlabel('$x_0$')
    plt.ylabel('$x_1$')

    if W is not None:
        # w0x0+w1x1+b=0 => x1=-w0x0/w1-b/w1
        w0 = W[0]
        w1 = W[1]
        temp = -w1*np.array([X[:,1].min(), X[:,1].max()])/w0-b/w0
        x0_min = max(temp.min(), X[:,0].min())
        x0_max = min(temp.max(), X[:,1].max())
        x0 = np.linspace(x0_min,x0_max,100)
        x1 = -w0*x0/w1-b/w1
        plt.plot(x0,x1,color='black')

    plt.show()
```

[55]:
```
# Visualize training set.
vis(X_train, Y_train)
```



[56]:
```
# Visualize test set.
vis(X_test, Y_test)
```



0.1.3 3.1 Perceptron Algorithm-deterministic

In this problem, we would like to train a perceptron model for the classification task on a modified Iris dataset. The training procedure of the perceptron model is shown in the algorithm below:

Note that in the code, we use `X_train` and `Y_train` to represent the feature vector X and labels Y in training set S_{training}. Besides, we use `W` and `b` to represent the weight vector $\mathbf{w}$ and bias scalar b.

Please fill the blanks of the skeleton code below to complete the perceptron training procedure.

[57]:
```
# Judge function: 1(a != b).
def judge(a, b):
    if a != b:
        return 1
    else:
        return 0

# Perceptron classifier.
def f_perceptron(x, W, b):
    # x should be a 2-dimensional vector,
    # W should be a 2-dimensional vector,
    # b should be a scalar.
    # you should return a scalar which is -1 or 1.
    if (W.T.dot(x) + b) < 0:
        return -1;
    else:
```

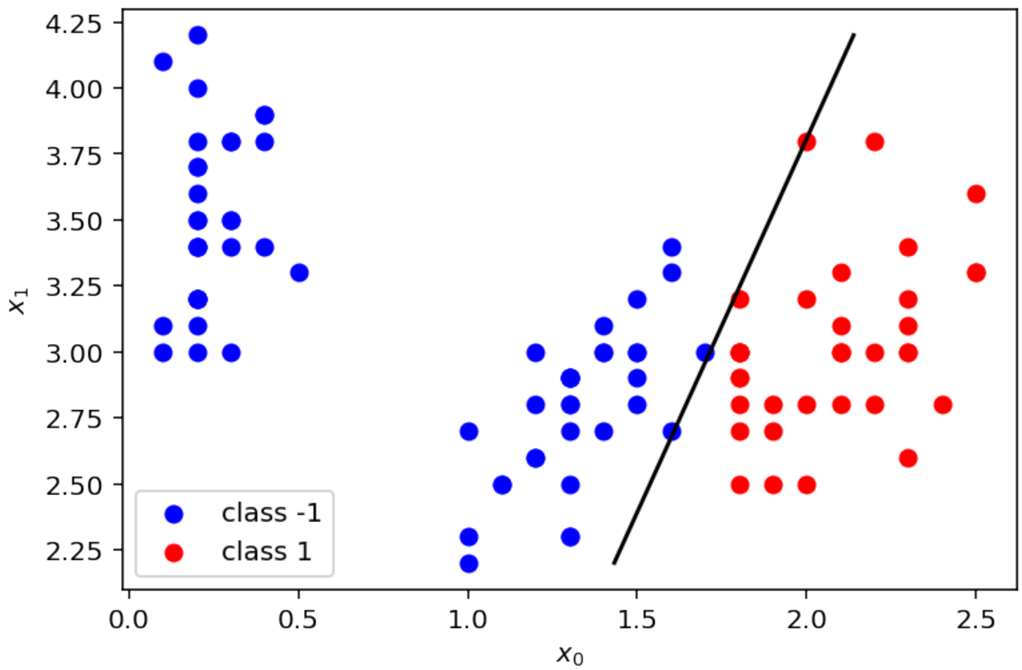
```
        return 1;

# Calculate error given feature vectors X and labels Y.
def calc_error(X, Y, W, b):
    ############ To be filled. ############
    err = 0
    l = len(X)
    for (xi, yi) in zip(X, Y):
        ############ To be filled. ############
        # Hint: Use judge() and f_perceptron()
        err = err + judge(yi, f_perceptron(xi, W, b))

    ############ To be filled. ############
    err = 1 * err/l
    return err ############ To be filled. ############
```

[58]:
```
# Some settings.
errors = []          # Error history.
lam    = 1           # Lambda which controls the step size.

# Initialization.
W      = np.zeros(2) # Weight.
b      = 0.0         # Bias.

# Perceptron learning algorithm.
while calc_error(X_train, Y_train, W, b) > 0:
    for xi, yi in zip(X_train, Y_train):   # Iterate over all data points.
        ####### To be filled #######       # Compute the model prediction.
        p = f_perceptron(xi, W, b)
        if p == yi:   # Compare prediction and label.
            continue                       # - If correct, continue.
        else:
            W = W + lam*(yi-p)*xi # - If not, update weight and bias.
            b = b + lam*(yi-p)

    # Track training errors.
    errors.append(calc_error(X_train, Y_train, W, b))
```

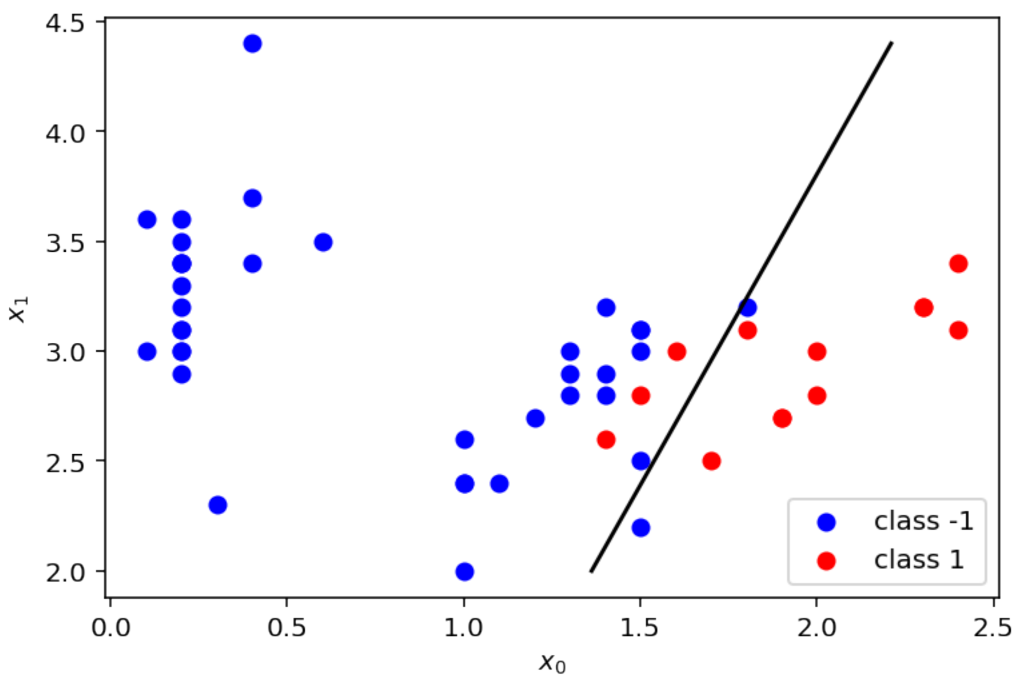
Visualize the results

[59]:
```
# Show decision boundary, training error and test error.
print('Decision boundary: {:.3f}x0+{:.3f}x1+{:.3f}=0'.format(W[0],W[1],b))
vis(X_train, Y_train, W, b)
print('Training error: {}'.format(calc_error(X_train, Y_train, W, b)))
vis(X_test, Y_test, W, b)
print('Test error: {}'.format(calc_error(X_test, Y_test, W, b)))
```

```
Decision boundary: 70.200x0+-24.800x1+-46.000=0
```



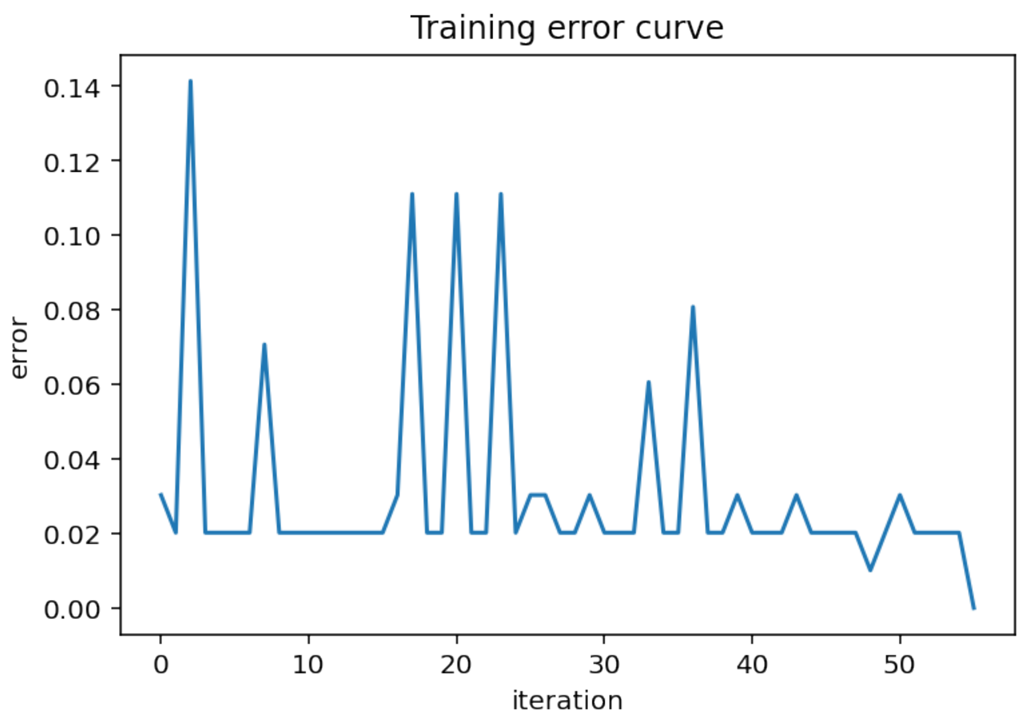
`Training error: 0.0`



`Test error: 0.1`

[60]:

```
# Plot training error curve.
plt.title('Training error curve')
plt.plot(errors)
plt.xlabel('iteration')
plt.ylabel('error')
plt.show()
```



0.1.4 3.2 Perceptron Algorithm-random

Please fill the blanks of the skeleton code below to complete the perceptron training procedure.

Most of the code is similar to 3.1 except a random sampling stategy with replacement is applied. You might have multiple ways to accomplish it.

Hint: You can randomly sample the indices of training data from 0 to k-1 and then use the index to get corresponding sample each time.

`random.choices(pop,n)` would return a n sized list of elements chosen from the input list pop with replacement.

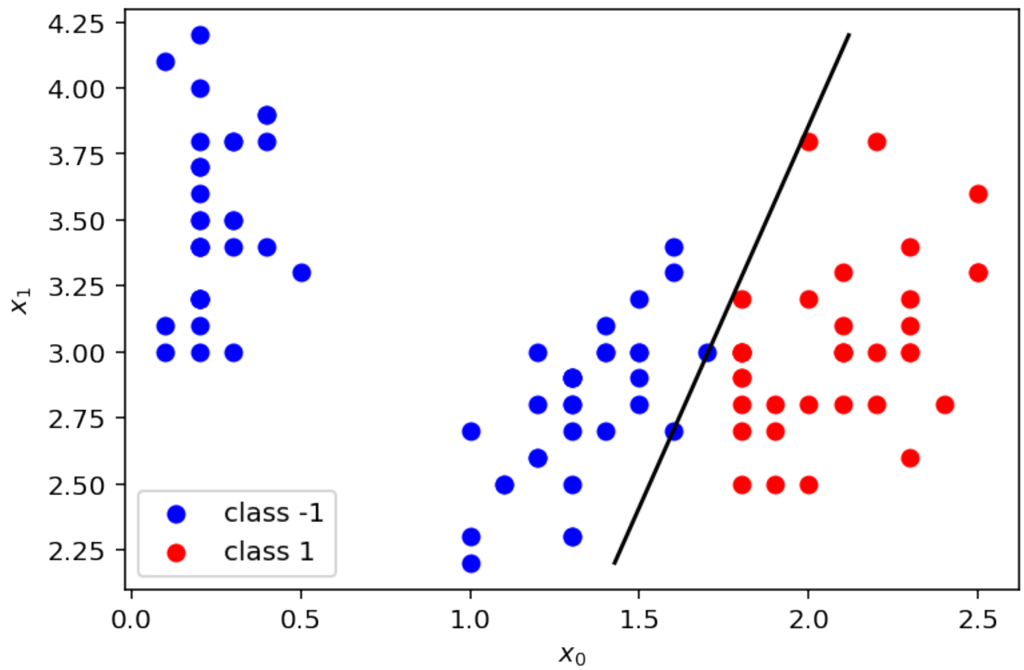
[76]:

```
# Some settings.
errors_random = []          # Error history.
lam    = 1              # Lambda which controls the step size.
for i in range(3):
    # Initialization.
```

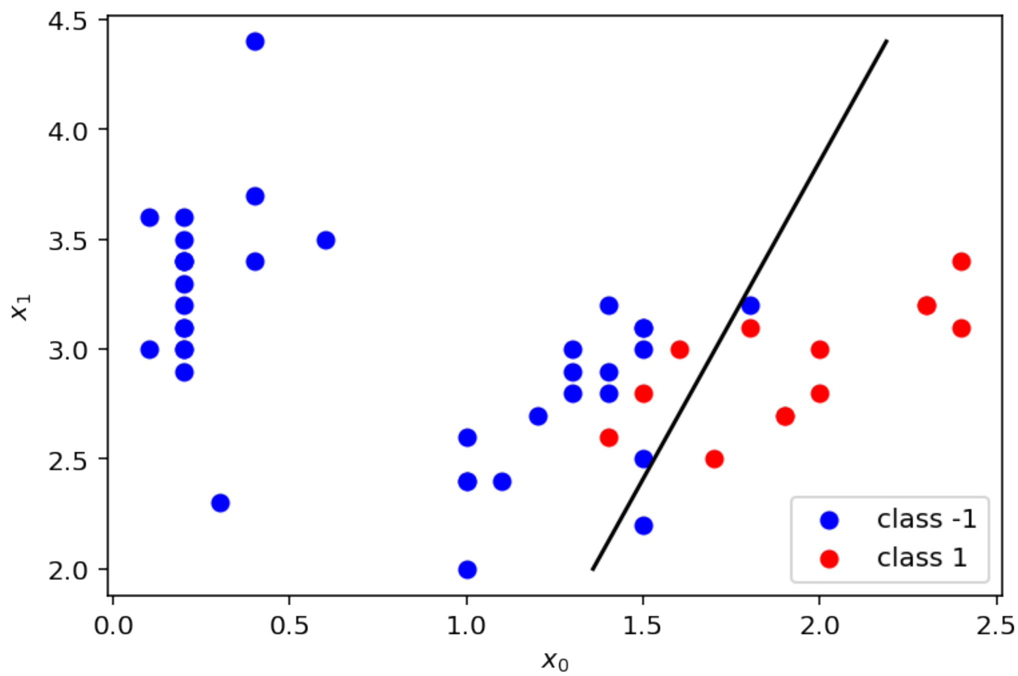
```
    W      = np.zeros(2)  # Weight.
    b      = 0.0          # Bias.
    k      = len(X_train) ############ To be filled. ############
    # Perceptron learning algorithm.
    errors = []
    while calc_error(X_train, Y_train, W, b) > 0:
        ############ To be filled. ############
        ind = random.choices(range(len(X_train)), k=len(X_train))
        for i in range(k):
            xi, yi = X_train[ind[i]], Y_train[ind[i]]############ To be filled.␣
↪############ Select batches randomly
            prediction = f_perceptron(xi, W, b)####### To be filled #######    ␣
↪  # Compute the model prediction.
            if (prediction == yi):####### To be filled #######    # Compare␣
↪prediction and label.
                continue                          # - If correct, continue.
            else:
                W = W+lam*(yi-prediction)*xi # - If not, update weight and bias.
                b = b+lam*(yi-prediction)
        # Track training errors.
        errors.append(calc_error(X_train, Y_train, W, b))
    errors_random.append(errors)
```

[77]:
```
# Show decision boundary, training error and test error.
print('Decision boundary: {:.3f}x0+{:.3f}x1+{:.3f}=0'.format(W[0],W[1],b))
vis(X_train, Y_train, W, b)
print('Training error: {}'.format(calc_error(X_train, Y_train, W, b)))
vis(X_test, Y_test, W, b)
print('Test error: {}'.format(calc_error(X_test, Y_test, W, b)))
```

```
Decision boundary: 81.000x0+-28.000x1+-54.000=0
```

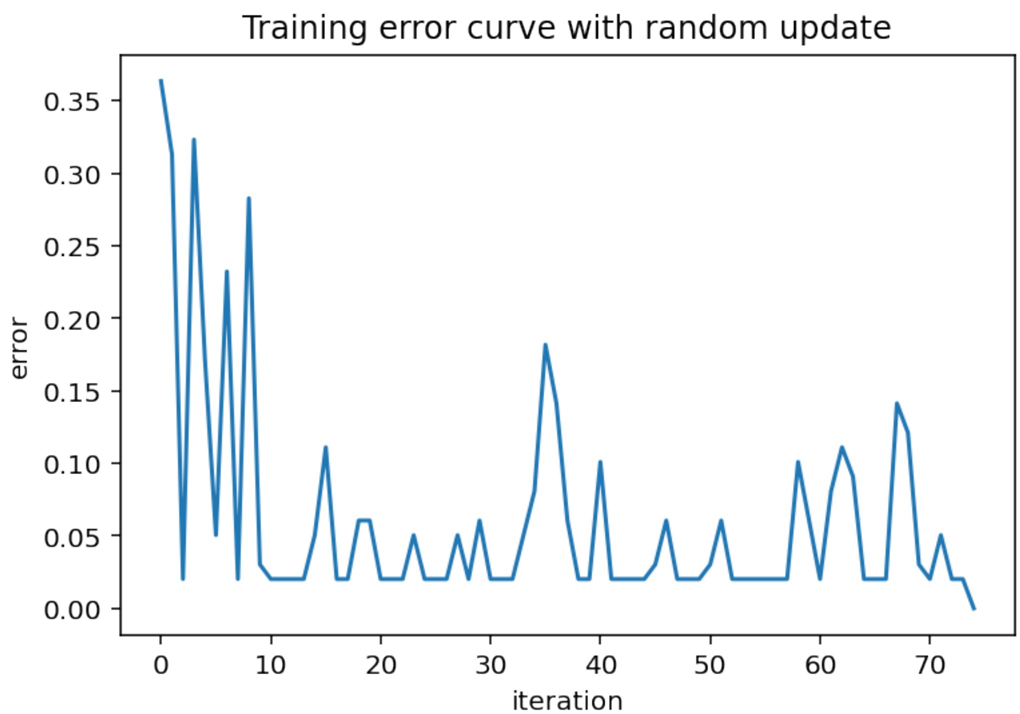


Training error: 0.0

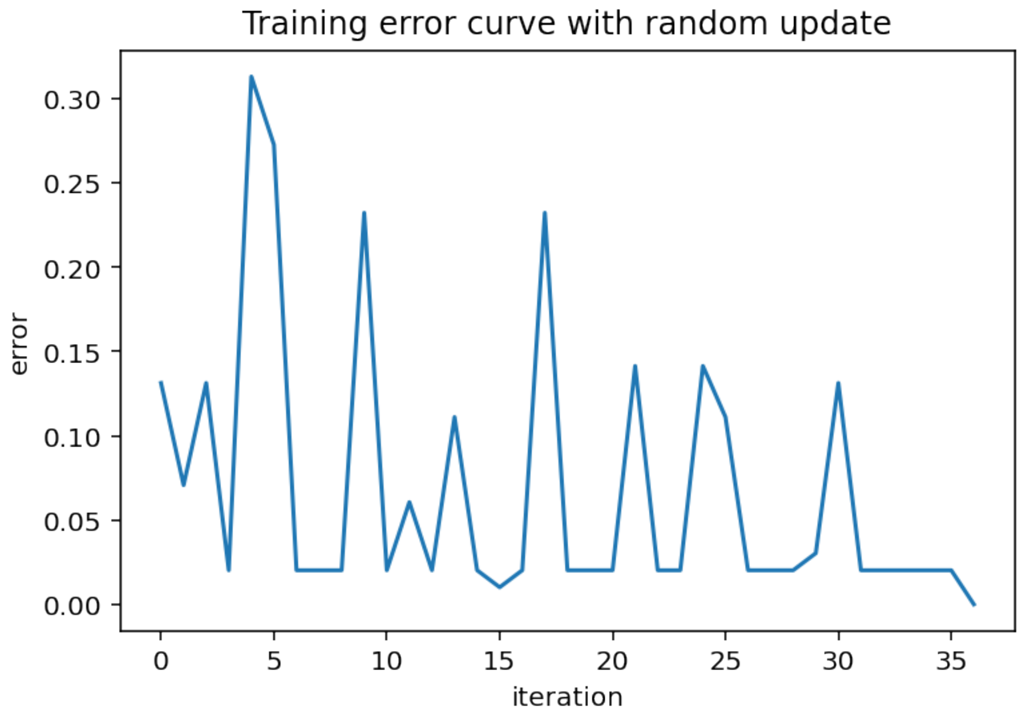


Test error: 0.1

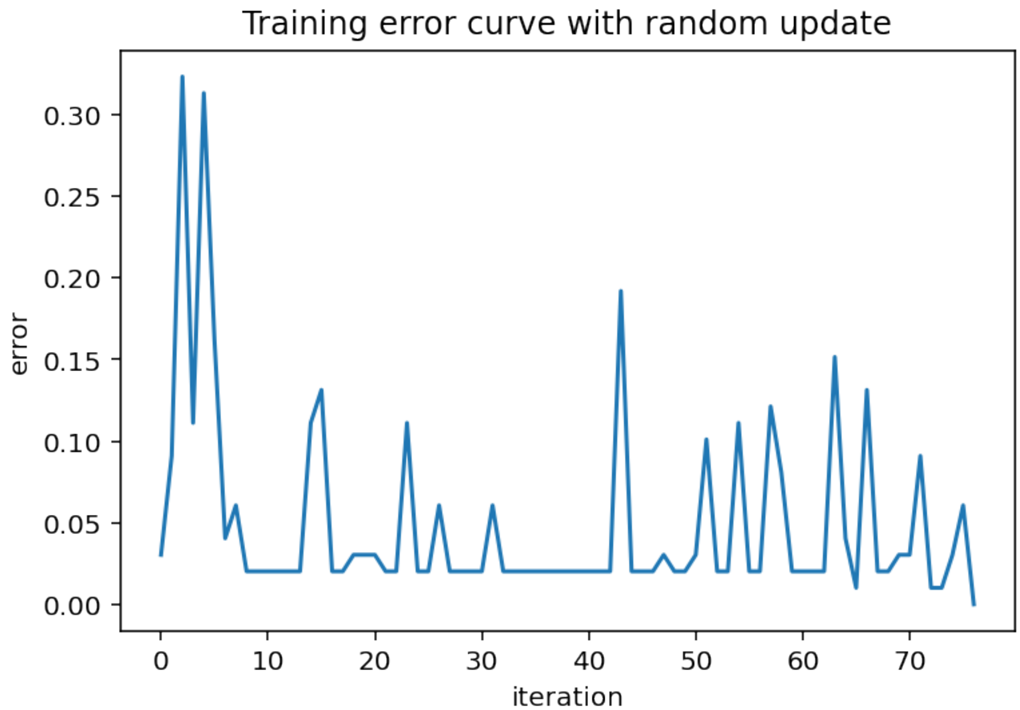
[78]:
```
# Plot training error curve.
plt.title('Training error curve with random update')
plt.plot(errors_random[0])
plt.xlabel('iteration')
plt.ylabel('error')
plt.show()
```



[79]:
```
# Plot training error curve.
plt.title('Training error curve with random update')
plt.plot(errors_random[1])
plt.xlabel('iteration')
plt.ylabel('error')
plt.show()
```



[80]:
```
# Plot training error curve.
plt.title('Training error curve with random update')
plt.plot(errors_random[2])
plt.xlabel('iteration')
plt.ylabel('error')
plt.show()
```


Training error curve with random update
error
iteration
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
60
70

```
[ ]:
```