Question 1.4 (a-e)

a) Generate a linearly separa ble data set of size 20 as indicated. Plot the examples { (xn, Yn)} as well as the target function f on a plane. Be sure to mark the examples from different classes differently, and add labels to the axes of the plot.

b) Run the perceptron learning algorithm on the data set above. Report the number of updates that the algorithm takes before converging. Plot the examples { (xn, Yn)}, the target function f, and the final hypothesis g in the same figure. Comment on whether f is close to g.

c) Repeat everything in (b) with another randomly generated data set of size 20. Compare your results with (b).

d) Repeat everything in (b) with another randomly generated data set of size 100. Compare your results with (b).

e) Repeat everything in (b) with another randomly generated data set of size 1, 000. Compare your results with (b)

Solution

```python
import matplotlib.pyplot as plt
import numpy as np


def plot(W_orig, x, target, W=None, title=''):
    """
    Plot data and decision boundary.

    :param W_orig: randomly generated weights
    :param x: data
    :param target: labels
    :param W: perceptron weights
    :param title: title for plot
    :return: NoneType
    """
    # compute slope for randomly generated weights
    slope_f = -(W_orig[1] / W_orig[2])
    # compute intercept randomly generated weights
    intercept_f = -W_orig[0] / W_orig[2]
    # create points to pass in equation
    xx = np.linspace(-1, 1)
    # count y with slope, intercept and created points
    yy = (slope_f * xx) + intercept_f
    # create figure
    plt.figure(figsize=(8, 8))
    # create scatterplot with data with different colors set by target value
    plt.scatter(x[:, 1], x[:, 2], c=target)
    # plot decision line for randomly generated weights
    plt.plot(xx, yy, 'r-', label='Random w')
    # if perceptron weights were given
    # compute slope and intercept
    # count y with slope, intercept and created points
    # plot decision line for perceptron weights
    if W is not None:
```

```python
        slope_g = -(W[1] / W[2])
        intercept_g = -W[0] / W[2]
        yy_g = (slope_g * xx) + intercept_g
        plt.plot(xx, yy_g, 'b-', label='Perceptron w')
    # add labels to axis, legends, title and grid
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title(title)
    plt.legend()
    plt.grid()
    plt.show()


def perceptron(X, target):
    """
    Perceptron.

    :param X: data
    :param target: target
    :return: weights and number of iterations
    """
    # set weights to all 0
    w = np.zeros(X.shape[1])
    n_iters = 0
    # run perceptron
    while True:
        # make predictions
        yhat = np.sign(X.dot(w))
        # check stop criteria
        if np.sum(target != yhat) == 0:
            break
        # update weights
        for i in range(len(target)):
            if target[i] * yhat[i] <= 0:
                w += target[i] * x[i]
        n_iters += 1
    return w, n_iters


# 1.4. a
# generate random data
x1, x2 = [np.random.uniform(-1, 1, 20) for _ in range(2)]
# add 1 for bias
x = np.array([[1] * 20, x1, x2]).T
# randomly generate weights
w1 = np.random.uniform(-100, 100)
w2 = np.random.uniform(-100, 100)
w3 = np.random.uniform(-100, 100)
weights = np.array([w1, w2, w3])
# compute target
```

```
target = np.sign(x.dot(weights))
# plot results
plot(weights, x, target, W=None, title='Target function')

# 1.4. b
# two lines are different, to converge algorithm has made 8 iterations

# run perceptron and compute weights
w, n_iters = perceptron(x, target)
# plot results
plot(weights, x, target, w, 'Target function and hypothesys after {} Perceptron
iterations'.format(n_iters))

# 1.4. c
# two lines still different, but new data was generated
# to converge algorithm has made 4 iterations

# generate new data with 20 points
x1, x2 = [np.random.uniform(-1, 1, 20) for _ in range(2)]
x = np.array([[1] * 20, x1, x2]).T
target = np.sign(x.dot(weights))
# run perceptron
w, n_iters = perceptron(x, target)
# plot results
plot(weights, x, target, w, 'Target function and hypothesys after {} Perceptron
iterations'.format(n_iters))

# 1.4. d
# two lines become much closer
# amount of points was increased to 100
# to converge algorithm has made 21 iterations

# generate new data with 100 points
x1, x2 = [np.random.uniform(-1, 1, 100) for _ in range(2)]
x = np.array([[1] * 100, x1, x2]).T
target = np.sign(x.dot(weights))
# run perceptron
w, n_iters = perceptron(x, target)
# plot results
plot(weights, x, target, w, 'Target function and hypothesys after {} Perceptron
iterations'.format(n_iters))

# 1.4. e
# two lines become almost identical
# and amount of points was increased to 1000
# to converge algorithm has made 6 iterations

# generate new data with 1000 points
x1, x2 = [np.random.uniform(-1, 1, 1000) for _ in range(2)]
x = np.array([[1] * 1000, x1, x2]).T
```

```
target = np.sign(x.dot(weights))
# run perceptron
w, n_iters = perceptron(x, target)
# plot results
plot(weights, x, target, w, 'Target function and hypothesys after {} Perceptron iterations'.format(n_iters))
```

Answer plots to every question in folder “plots”