HW4

113507489
Jiye "Violet" Oh

#1.

Exercise 1 – Nearest Neighbors (20 points). We are given the following training examples in 2D for binary classification:

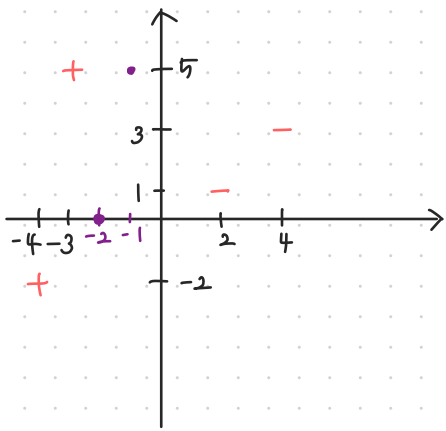
$$((-3,5),+),\ \ ((-4,-2),+),\ \ ((2,1),-),\ \ ((4,3),-)$$

Assume that we want to classify the points $(-2,0)$ and $(-1,5)$ using

(a) **(10 pts)** a 1-nearest neighbor rule, and

(b) **(10 pts)** a 3-nearest neighbor rule.

Use the Euclidean distance (L_2-norm) for calculating the distance between the points, in order to determine the k nearest neighbors to the query points given. What is the classification of these two points in each case?



a) $K=1$

- Euclidean distance : From $(-1,5)$ to

$$\begin{cases} (-3,5) : \sqrt{4} = 2 \\ (-4,-2) . \sqrt{9+49} = \sqrt{58} \\ (2,1) . \sqrt{9+16} = 5 \\ (4,3) : \sqrt{25+4} = \sqrt{29} \end{cases}$$

The closest training example from $(-1,5)$ is $((-3,5),+)$, and therefore point $(-1,5)$ is assigned positive $(+)$.

- Euclidean distance: From $(-2,0)$ to
$$\begin{cases} (-3,5) : \sqrt{1+25} = \sqrt{26} \\ (-4,-2) : \sqrt{4+4} = \sqrt{8} \\ (2,1) : \sqrt{16+1} = \sqrt{17} \\ (4,3) : \sqrt{36+9} = \sqrt{45} \end{cases}$$

The closest training example from $(-2,0)$ is $((-4,-2),+)$, and therefore point $(-2,0)$ is assigned positive $(+)$.

b) $K=3$

- For $(-1,5)$: 3 nearest neighbors are $((-3,5),+)$, $((2,1),-)$, $((4,3),-)$.

  2 out of 3 belong to class $(-)$, therefore $(-1,5)$ is assigned negative

- For $(-2,0)$: 3 nearest neighbors are $((-3,5),+)$, $((2,1),-)$, $((-4,-2),+)$.

  2 out of 3 belong to class $(+)$, therefore $(-2,0)$ is assigned positive.

#2.

Exercise 2 – Perceptron (30 points). We are given the following training examples in 2D (same as above):

$$((-3,5),+),\ \ ((-4,-2),+),\ \ ((2,1),-),\ \ ((4,3),-)$$

Use +1 to map positive (+) examples and -1 to map negative (-) examples.

We want to apply the learning algorithm for training a perceptron using the above data with starting weights $w_0 = w_1 = w_2 = 0$ and learning rate $\eta = 0.1$. In each iteration process the training examples in the order given above. Complete at most 3 iterations over the above training examples.

(a) **(24 pts)** What are the weights at the end of each iteration?

(b) **(6 pts)** Are these weights final?

(a) $$w_i \leftarrow w_i + \underbrace{\eta\,(t-o)\,x_i}_{\Delta w_i}$$

```
import numpy as np

# Define the training data
X = np.array([[1, -3, 5], [1, -4, -2], [1, 2, 1], [1, 4, 3]])
y = np.array([1, 1, -1, -1])

# Define the initial weight and learning rate
w = np.zeros(3)
lr = 0.1

# Train the perceptron for a maximum of 3 iterations
for iter in range(3):
    i = 0  # for keeping track of y
    for x in X:
        if np.dot(w, x) > 0:
            a = 1
        else:
            a = -1
        for j in range(3):
            # Update the weights
            w[j] = w[j] + lr * (y[i] - a) * x[j]
        i += 1

    print(f"Weights after iter {iter+1}: {w}")
```

```
(base) C:\Users\jiyeo\Desktop\workspace_python\ML>
ace_python/ML/SL/hw4_p2.py
Weights after iter 1: [ 0.  -1.4  0.4]
Weights after iter 2: [ 0.  -1.4  0.4]
Weights after iter 3: [ 0.  -1.4  0.4]
```

⇒ weights at the end of each iteration

(b) The weights are final.

Running more iterations result in the same weights values.

#3.

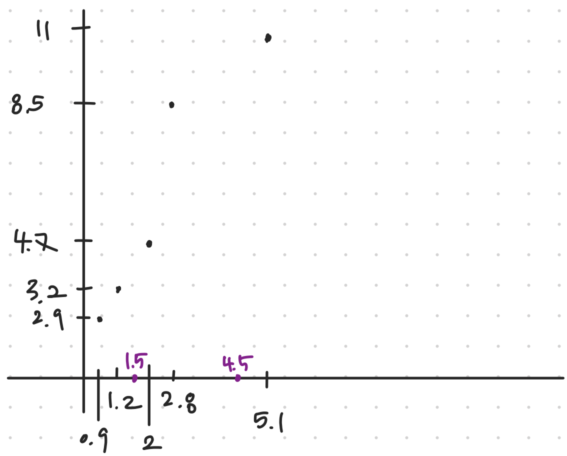
Exercise 3 – Nearest Neighbors (20 points). We are given the following training examples:

$$(1.2, 3.2),\ \ (2.8, 8.5),\ \ (2, 4.7),\ \ (0.9, 2.9),\ \ (5.1, 11)$$

We want to apply a 3-nearest neighbor rule in order to perform regression.

(a) **(8 pts)** Predict the label (real value) at each of the following two points: $x_1 = 1.5$ and $x_2 = 4.5$.

(b) **(8 pts)** Instead of weighing the contribution of each of the 3 nearest neighbors equally, this time we want to perform distance-weighted nearest neighbor regression. What values do we predict now for $x_1 = 1.5$ and $x_2 = 4.5$?



a). $x_1 = 1.5$ has $(1.2, 3.2)$, $(2, 4.7)$, and $(0.9, 2.9)$ as 3-nearest neighbors.

$$\frac{(3.2 + 4.7 + 2.9)}{3} = \boxed{3.6}$$

· $x_2 = 4.5$ has $(5.1, 11)$, $(2.8, 8.5)$, and $(2, 4.7)$ as 3-nearest neighbors.

$$\frac{(11 + 8.5 + 4.7)}{3} = \boxed{8.067}$$

b)

$$\frac{\sum_{i=1}^{K} W_i \cdot C(X_i)}{\sum_{i=1}^{K} W_i} = \frac{11.11 \cdot 3.2 + 4 \cdot 4.7 + 2.78 \cdot 2.9}{17.89} = \boxed{3.49}$$

for $x_1 = 1.5$

$$W_1 = \frac{1}{0.3^2} = 11.11$$

$$W_2 = \frac{1}{0.5^2} = 4$$

$$W_3 = \frac{1}{0.6^2} = 2.78$$

$$\frac{\sum_{i=1}^{K} W_i \cdot C(X_i)}{\sum_{i=1}^{K} W_i} = \frac{11 \cdot 2.78 + 8.5 \cdot .35 + 4.7 \cdot .16}{3.29} = \boxed{10.43}$$

for $x_2 = 4.5$

$$W_1 = \frac{1}{0.6^2} = 2.78$$

$$W_2 = \frac{1}{1.7^2} = .35$$

$$W_3 = \frac{1}{2.5^2} = .16$$

(c) **(4 pts)** Set tics on the horizontal axis in distance 0.1 in between them, so that you have 61 tics in the interval $[0.0, 6.0]$ (the tics include the endpoints of the interval). For each one of these values on the x-axis, predict the label (real number) using the previous two methods that you have implemented (3-NN with equal weight, as well as 3-NN distance-weighted). Store your predictions of each method in a different file and plot them together with the five points that were given to you for training.

As an example, you can use gnuplot (http://www.gnuplot.info) for plotting. Say, you have the files named 'kNN-equal.txt' and 'kNN-weighted.txt', then we can plot these points and connect them with a line using the command

```
gnuplot> plot "kNN-weighted.txt" w lp, "kNN-equal.txt" w lp
```

You can also plot individual points with a command like the following one:

```
gnuplot> set object circle at first 0.9, 2.9 radius char 0.5 \
fillstyle empty border lc rgb '#aa1100' lw 2
```

```
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt

points = np.array([[1.2, 3.2], [2.8, 8.5], [2, 4.7], [0.9, 2.9], [5.1, 11]])
X = np.array([[1, 1.2], [1, 2.8], [1, 2], [1, 0.9], [1, 5.1]])
#y = np.array([3.2, 8.5, 4.7, 2.9, 11])

# X = np.array([[1.2, 3.2], [2.8, 8.5], [2, 4.7], [0.9, 2.9], [5.1, 11]])
y = np.arange(0, 6.1, 0.1)

distances = cdist(X, y.reshape(-1, 1))
nearest = np.argpartition(distances, 3, axis=0)[:3]
labels_equal = np.mean(X[nearest], axis=1)
weights = 1 / distances[nearest]
labels_weighted = np.sum(
    X[nearest] * weights, axis=1) / np.sum(weights, axis=1)
labels_weighted = np.array([[1.5, 3.49], [4.5, 10.43]])
plt.scatter(points[:, 0], points[:, 1], color='red', label='points')
# plt.plot(points)
plt.plot(y, labels_equal, label='3-NN Equal Weight')
plt.plot(y, labels_weighted, label='3-NN Distance Weighted')
plt.legend()
plt.show()
```

#4.

Exercise 4 – Gradient Descent (30 points). We are given the following training examples (same as above):

$$(1.2, 3.2), \quad (2.8, 8.5), \quad (2, 4.7), \quad (0.9, 2.9), \quad (5.1, 11)$$

Suppose the weights are $w_0 = w_1 = 1$ initially. We want to minimize the cumulative loss $\mathcal{L}_S(h, c)$ that corresponds to the half of the residual sum of squares; i.e., we have that

$$\mathcal{L}_S(h, c) = \frac{1}{2} RSS_S(h, c) = \frac{1}{2} \sum_{i=1}^{m} (y_i - h(x_i))^2$$

and we will be using full gradient descent as discussed in our slides (slide 26 from Module 7).

(a) **(24 pts)** Using $\eta = 0.01$, perform three iterations of full gradient descent, listing w_0 and w_1 at each iteration.

(b) **(6 pts)** What is the value that we predict at the following points $x_1 = 1.5$ and $x_2 = 4.5$?

```
import numpy as np

# Define the training data
X = np.array([[1, 1.2], [1, 2.8], [1, 2], [1, 0.9], [1, 5.1]])
y = np.array([3.2, 8.5, 4.7, 2.9, 11])
P = np.array([[1, 1.5], [1, 4.5]])

# Define the initial weight and learning rate
w = np.ones(2)
lr = 0.01

print("#4 (a)")
for iter in range(3):
    i = 0  # for keeping track of y
    delta_w = np.zeros(2)
    for x in X:
        o = np.dot(w, x)
        for j in range(2):
            delta_w[j] = delta_w[j] + lr * (y[i] - o) * x[j]
        i += 1
    for k in range(2):
        w[k] = w[k] + delta_w[k]

    print(f"Weights after iter {iter+1}: {w}")

print("#4 (b)")
print(np.dot(P, w))
```

```
#4 (a)                        Wo        W1
Weights after iter 1: [1.133   1.4365]
Weights after iter 2: [1.20697    1.6820035]
Weights after iter 3: [1.24778108 1.8201837 ]
#4 (b)
[3.97805662 9.43860771]
```

$x_1 = 1.5$ $\qquad$ $x_2 = 4.5$