Problem Set 02: Linear Classifiers I

Johannes C. B. Dietschreit, Sascha Mausenberger

Due: 21.10.2024

The problems below are meant to be solved using Python functions and libraries. You can do so by either writing short scripts or using Jupyter notebooks. **Report your results in the Quiz on Moodle** to obtain a grade.

1 Introduction

In this PSet you will program the first ML algorithm yourself. We will present you with some introductory code for some important packages, however, we expect you to look for functions you might need yourself (the documentations of all mayor Python packages we use in this course are very good, further Google and ChatGPT are of help).

1.1 NumPy

NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a wide collection of mathematical functions to operate on these arrays. NumPy is widely used for numerical computations, data analysis, and serves as a foundation for many other scientific computing libraries in Python.

```python
import numpy as np

array_1d = np.array([1, 2, 3, 4, 5])
array_2d = np.array([[1, 2, 3], [4, 5, 6]])

# Special arrays
zeros_array = np.zeros((2, 3))
ones_array = np.ones((2, 3))
range_array = np.arange(0, 10, 2) # [0, 2, 4, 6, 8]
random_array = np.random.rand(3, 3) # 3x3 random numbers

# Basic operations
sum_array = np.sum(array_1d)
mean_array = np.mean(array_1d)
max_value = np.max(array_1d)
min_value = np.min(array_1d)

# Reshape an array
reshaped_array = array_1d.reshape((1, 5)) # now 2D
```

```
# Matrix operations
matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])

matrix_add = matrix_a + matrix_b
matrix_sub = matrix_a - matrix_b
elementwise_mul = matrix_a * matrix_b # Element-wise Multiplication
dot_product = np.dot(matrix_a, matrix_b) # Dot Product (only for vectors)
# 2 ways to do Matrix Multiplication
matrix_product = np.matmul(matrix_a, matrix_b)
matrix_product =  matrix_a @ matrix_b

transpose_a = np.transpose(matrix_a)
det_a = np.linalg.det(matrix_a) # Determinant
inv_a = np.linalg.inv(matrix_a) #Inverse
eigvals, eigvecs = np.linalg.eig(matrix_a) # Eigenvalues and Eigenvectors
trace_a = np.trace(matrix_a) # Trace of a Matrix =  sum of diagonal
```

2 The Problems

2.1 Classify Samples

Given the following linear classifiers ($\mathbf{w}$, b), report the class labels for the test points (valid answers are only the integers -1 and 1)? (each 0.5 P)

1.
```
w = [0, 1]
b = 0
x = [1, -2]
```

2.
```
w = [0, 1]
b = 3
x = [1, -2]
```

3.
```
w = [1, 7]
b = -3
x = [3, 4]
```

4.
```
w = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = 1.23
x = [3, 4, -13, 7, 15, -6, -9, 17, 11, 15]
```

2.2 Perceptron without Offset

Implement the perceptron algorithm without offset and train on the given training sets. Set the maximum number of epochs (iterations over the entire data set) to 100. Report the values of learned weights $\mathbf{w}$ rounded to 2 digits after the decimal point. If the algorithm does not converge report `nan` for the elements of $\mathbf{w}$. (each 1.0 P)

1.
```
Xs = np.array([[ 0.87578245,  0.30375931,  1.691093  ],
       [-0.53813241, -0.45833158, -1.12207415],
       [ 0.99702363, -0.08317198, -0.222416  ],
       [ 0.72118904, -0.24845405,  0.1664802 ],
       [ 0.31284622, -0.40632737,  0.73926731],
       [ 0.20900079, -0.20243908, -0.67501829],
       [ 0.48727053, -0.71112876,  1.50976111],
       [ 0.15070634,  0.26012988, -0.08667005],
       [-0.65639742,  1.07434399,  2.09392423],
       [-0.73303502, -0.15598171, -0.14274047]])

ys = [1, -1, 1, 1, 1, -1, 1, -1, 1, -1]
```

2.
```
Xs = np.array([[ 0.58715982,  0.76053109],
       [-1.09492364,  2.07989924],
       [ 0.04097468, -0.09437459],
       [-1.15073364, -0.81630217],
       [ 0.57206772,  0.3813378 ],
       [ 0.93345859,  0.92408983],
       [ 0.58881134, -0.71864108],
       [-0.07350665,  0.51727535],
```

```
        [-1.32103648,  1.2211991 ],
        [-0.82438608,  0.35219818],
        [-0.89396187,  0.67834576],
        [-1.56266947,  1.01928997],
        [-0.06133912,  0.57360667],
        [ 0.23596025,  0.5329578 ],
        [ 1.21315196,  2.27594733],
        [-1.03981669, -0.89622489],
        [ 0.13371041, -0.39055231],
        [ 0.81622561,  0.42530499],
        [ 0.24819307,  0.86653034],
        [ 0.27427925,  0.78433078],
        [-0.17038365, -3.21936226],
        [-0.47376705,  0.21173016],
        [ 0.7697084 , -0.16046939],
        [ 1.55237121,  0.6258206 ],
        [ 1.93907678,  1.07390485],
        [-0.6018449 ,  0.07496144],
        [ 0.15338813,  1.10189419],
        [-0.96326495,  1.59790052],
        [-0.68733638, -1.42420832],
        [-1.26075745, -0.273788  ]])

ys = [1, 1, -1, -1, 1, 1, 1, 1, -1, -1, -1, -1, 1, 1, 1, -1, -1, 1,
    1, 1, -1, -1, 1, 1, 1, -1, 1, 1, -1,-1]
```

3.

```
Xs = np.array([[-0.31909535, -0.09867284],
        [-2.03820957, -0.63471519],
        [ 0.91455592, -0.24052323],
        [ 0.52258241, -0.73623614],
        [-0.73571052,  1.58181469],
        [ 0.02969905,  2.51855278],
        [-0.51895267, -1.45991397],
        [ 0.96851532, -0.36872916],
        [-0.67072931,  1.87263213],
        [ 1.38877095, -0.66055964]])

ys = [1, -1, 1, -1, 1, 1, -1, 1, 1, -1]
```

2.3 Averaged Perceptron

Implement the Averaged Perceptron algorithm **WITH** offset and train on the given training sets. Set the maximum number of epochs (iterations over the entire data set) to 100. Report the values of learned weights `w` and the bias `b`. (each 1.0 P)

1.

```
Xs = np.array([[-1.76762651,  0.25606913,  0.42479004],
        [-0.28857931,  0.62839084,  0.03800888],
        [-0.52516867,  2.27344459, -1.24928071],
        [-0.51704967, -1.92189839,  0.75714427],
        [ 0.47149895, -0.4314566 ,  0.02741449],
        [ 1.14736743,  0.73926771, -0.25313897],
        [-0.36569072,  0.06845024, -1.01979881],
        [ 0.33973728,  0.36858664,  1.33468387],
        [-0.93029838,  0.72336265, -1.54566681],
        [-0.41389771, -0.19616516,  0.32847688]])
```

```
ys = [1, 1, 1, -1, -1, 1, -1, 1, -1, -1]
```

2.
```
Xs = np.array([[-0.31909535, -0.09867284],
       [-2.03820957, -0.63471519],
       [ 0.91455592, -0.24052323],
       [ 0.52258241, -0.73623614],
       [-0.73571052,  1.58181469],
       [ 0.02969905,  2.51855278],
       [-0.51895267, -1.45991397],
       [ 0.96851532, -0.36872916],
       [-0.67072931,  1.87263213],
       [ 1.38877095, -0.66055964]])
ys = [1, -1, 1, -1, 1, 1, -1, 1, 1, -1]
```

2.4 Visualizing the Decision Boundary

Given the 2D linear classifier, plot the decision boundary as a line. (1.0 P)
`w = [2.13, 0.87], b = 0.0`

2.5 Checking a Classifier

Given the labeled data set (see below) as well as the linear classifier `w = [1.20, 2.18], b = 0.0`, plot the decision boundary as a line, plot the points as a scattter plot (color the points using the result of the classifier, positive label = orange, negative label = blue, and wrong label = red). (1.5 P)
Compute the mean classification error, round the result to 2 digits after the decimal point. (0.5 P)

```
Xs = np.array([[ 0.58715982,  0.76053109],
       [-1.09492364,  2.07989924],
       [ 0.04097468, -0.09437459],
       [-1.15073364, -0.81630217],
       [ 0.57206772,  0.3813378 ],
       [ 0.93345859,  0.92408983],
       [ 0.58881134, -0.71864108],
       [-0.07350665,  0.51727535],
       [-1.32103648,  1.2211991 ],
       [-0.82438608,  0.35219818],
       [-0.89396187,  0.67834576],
       [-1.56266947,  1.01928997],
       [-0.06133912,  0.57360667],
       [ 0.23596025,  0.5329578 ],
       [ 1.21315196,  2.27594733],
       [-1.03981669, -0.89622489],
       [ 0.13371041, -0.39055231],
       [ 0.81622561,  0.42530499],
       [ 0.24819307,  0.86653034],
       [ 0.27427925,  0.78433078],
       [-0.17038365, -3.21936226],
       [-0.47376705,  0.21173016],
       [ 0.7697084 , -0.16046939],
       [ 1.55237121,  0.6258206 ],
       [ 1.93907678,  1.07390485],
       [-0.6018449 ,  0.07496144],
       [ 0.15338813,  1.10189419],
```

```
        [-0.96326495,  1.59790052],
        [-0.68733638, -1.42420832],
        [-1.26075745, -0.273788  ]])

ys = np.array([1, 1, -1, -1, 1, 1, 1, 1, -1, -1, -1, -1, 1, 1, 1, -1, -1,
    1, 1, 1, -1, -1, 1, 1, 1, -1, 1, 1, -1,-1], dtype=np.int64)
```