COMS W4701 Artificial Intelligence, Summer 2025

Homework 5A

Yahao (Ivan) Chen

Uni: yc4516

07/01/2025

Problem 1 Decision Tree

1.1 Gini Index and Information gain

Binary classification, Gini Index $G = 2\hat{p}(1-\hat{p})$

Gini Index before splitting (root)

$$G(X) = 2 * \frac{1}{2} * \left(1 - \frac{1}{2}\right) = 0.5$$

Information gains for feature 1 and 2 respectively,

$$IG(F_1) = G(X) - \left[\frac{2}{6} * G(x_1, x_2) + \frac{3}{6} * G(x_3, x_4, x_5) + \frac{1}{6} * G(x_6)\right] =$$

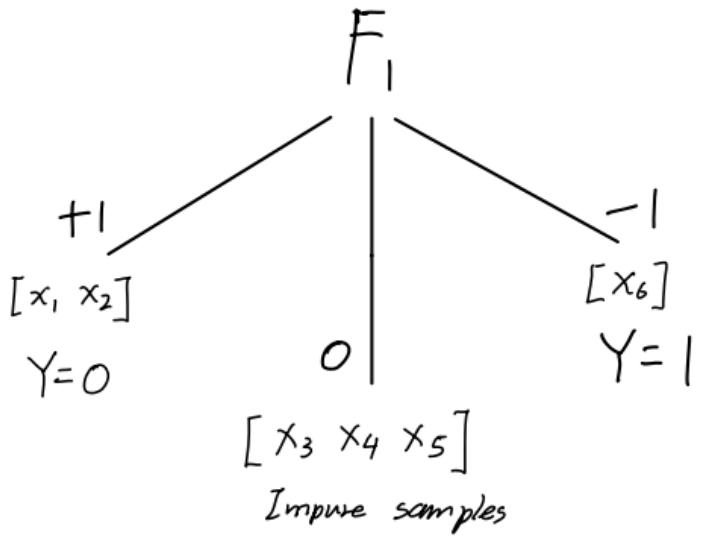
$$\frac{1}{2} - \left[\frac{2}{6} * (2 * 0) + \frac{3}{6} * \left(2 * \frac{2}{3} * \frac{1}{3}\right) + \frac{1}{6} * (2 * 0)\right] = \frac{5}{18} = 0.2778$$

$$IG(F_2) = G(X) - \left[\frac{3}{6} * G(x_2, x_5, x_6) + \frac{1}{6} * G(x_3) + \frac{2}{6} * G(x_1, x_4)\right] =$$

$$\frac{1}{2} - \left[\frac{3}{6} * \left(2 * \frac{2}{3} * \frac{1}{3}\right) + \frac{1}{6} * (2 * 0) + \frac{2}{6} * \left(2 * \frac{1}{2} * \frac{1}{2}\right)\right] = 1/9 = 0.1111$$

Since F1 gives us greater information gain, we choose F1 as the feature to split the decision tree.

x1 and x2 go to F1=+1 branch; x3, x4, x5 go to F1 = 0 branch; x6 go to F1 = -1 branch



1.2 Splitting on the remaining feature

Gini Index before splitting

$$G(x_3, x_4, x_5) = 2 * \frac{2}{3} * \frac{1}{3} = 4/9$$

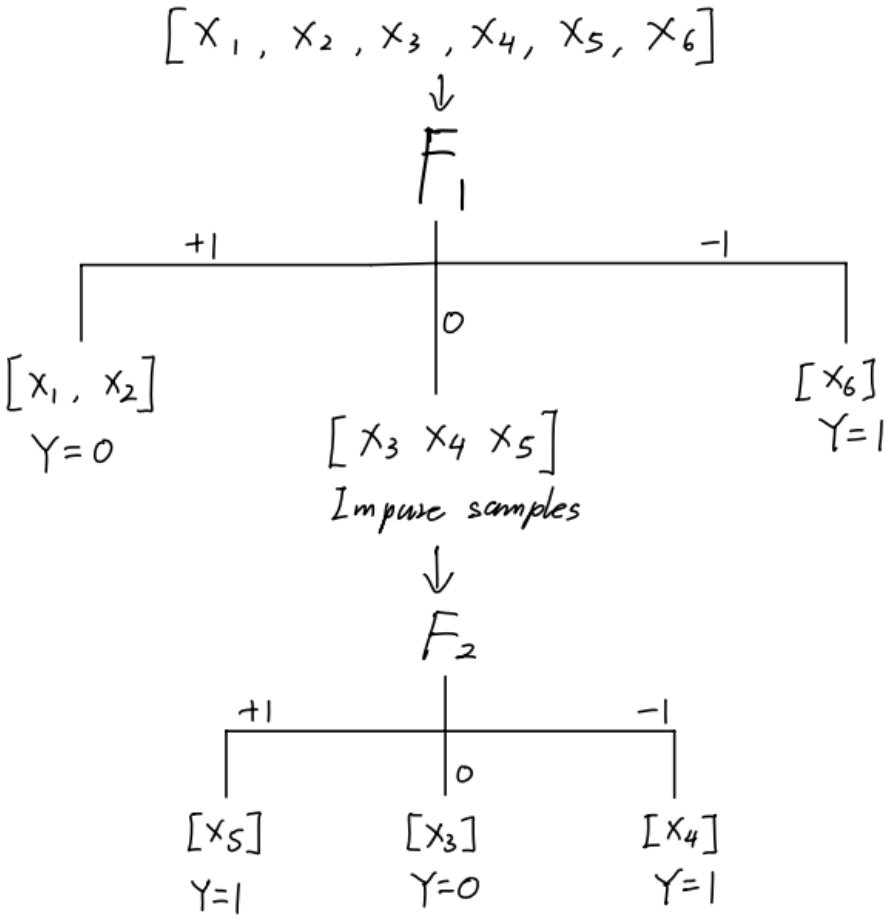
Information gain for F2

$$IG(F_2) = G(x_3, x_4, x_5) - \left[\frac{1}{3} * G(x_3) + \frac{1}{3} * G(x_4) + \frac{1}{3} * G(x_5)\right] =$$

$$\frac{4}{9} - 0 = 4/9 = 0.444$$

X3 split to F2=0; x4 split to F2 = -1; x5 split to F2 = +1

1.3 Full Learned Decision Tree



Problem 2 Naïve Bayes

2.1 Feature CPTs

For reference

Sample	F_1	F_2	Y
$\mathbf{x}_1$	+1	−1	0
$\mathbf{x}_2$	+1	+1	0
$\mathbf{x}_3$	0	0	0
$\mathbf{x}_4$	0	−1	1
$\mathbf{x}_5$	0	+1	1
$\mathbf{x}_6$	−1	+1	1

Given Max likelihood class CPT Pr(Y) = (0.5, 0.5)

Estimated feature CPTs of a Naïve Bayes model without Laplace smoothing (Excel tables):

Feature CPTs calculation: $\Pr(f|y) = \frac{count(f|y)}{count(y)}$

Pr(F_1|Y):

F1|Y	'+1'	'0'	'-1'
Y=1	0.0000	0.6667	0.3333
Y=0	0.6667	0.3333	0.0000

Pr(F_2|Y):

F2|Y	'+1'	'0'	'-1'
Y=1	0.6667	0.0000	0.3333
Y=0	0.3333	0.3333	0.3333

2.2 Predicted Y for new X data

Let's call the three new X combinations that do not appear in the given data set x_7, x_8, x_9.

And their feature combinations are:

New Sample	F1	F2
x7	'+1'	'0'
x8	'-1'	'0'
x9	'-1'	'-1'

Predict class label: $y = {}^{argmax}_{y} P(y|f1, \ldots . fn) = {}^{argmax}_{y} P(y) \prod_{i=1}^{n} P(fi|y)$

For x_7, F1=+1, F2=0:

$$P(y=1) * P(F1=1|y=1) * P(F2=0|y=1) = 0.5 * 0 * 0 = 0$$

$$P(y=0) * P(F1=1|y=0) * P(F2=0|y=0) = 0.5 * 2/3 * 1/3 = 0.1111$$

Therefore, Y=0 for sample x_7

For x_8, F1=-1, F2=0:

$$P(y=1) * P(F1=-1|y=1) * P(F2=0|y=1) = 0.5 * 1/3 * 0 = 0$$

$$P(y=0) * P(F1=-1|y=0) * P(F2=0|y=0) = 0.5 * 0 * 1/3 = 0$$

No clear prediction for sample x_8. Because probability for both Y classes is zero. Since no Laplace smoothing, even 1 zero probability for certain feature | class combinations lead to the whole joint probability vanishing to 0.

For x_9, F1=-1, F2=-1:

$$P(y=1) * P(F1=-1|y=1) * P(F2=-1|y=1) = 0.5 * 1/3 * 1/3 = 1/18$$

$$P(y=0) * P(F1=-1|y=0) * P(F2=-1|y=0) = 0.5 * 0 * 1/3 = 0$$

Therefore, Y=1 for sample x_9

2.3 Expectation-Maximization E-step

Estimated probabilities from part 1:

Pr(F_1|Y):

F1|Y	'+1'	'0'	'-1'
Y=1	0.0000	0.6667	0.3333
Y=0	0.6667	0.3333	0.0000

Pr(F_2|Y):

F2|Y	'+1'	'0'	'-1'
Y=1	0.6667	0.0000	0.3333
Y=0	0.3333	0.3333	0.3333

Calculating the joint probabilities Pr(Y, F1, F2) then normalize to get the expected counts

$$\Pr(Y|x1) \propto \Pr(Y)\Pr(+1|Y)\Pr(-1|Y) = (0.5, 0.5) \times \left(0, \frac{2}{3}\right) \times \left(\frac{1}{3}, \frac{1}{3}\right) = (0, \frac{1}{9}) \propto (0, 1)$$

$$\Pr(Y|x2) \propto \Pr(Y)\Pr(+1|Y)\Pr(+1|Y) = (0.5, 0.5) \times \left(0, \frac{2}{3}\right) \times \left(\frac{2}{3}, \frac{1}{3}\right) = (0, \frac{1}{9}) \propto (0, 1)$$

$$\Pr(Y|x3) \propto \Pr(Y)\Pr(0|Y)\Pr(0|Y) = (0.5, 0.5) \times \left(\frac{2}{3}, \frac{1}{3}\right) \times \left(0, \frac{1}{3}\right) = (0, \frac{1}{18}) \propto (0, 1)$$

$$\Pr(Y|x4) \propto \Pr(Y)\Pr(0|Y)\Pr(-1|Y) = (0.5, 0.5) \times \left(\frac{2}{3}, \frac{1}{3}\right) \times \left(\frac{1}{3}, \frac{1}{3}\right) = (\frac{1}{9}, \frac{1}{18}) \propto (\frac{2}{3}, \frac{1}{3})$$

$$\Pr(Y|x5) \propto \Pr(Y)\Pr(0|Y)\Pr(+1|Y) = (0.5, 0.5) \times \left(\frac{2}{3}, \frac{1}{3}\right) \times \left(\frac{2}{3}, \frac{1}{3}\right) = (\frac{2}{9}, \frac{1}{18}) \propto (\frac{4}{5}, \frac{1}{5})$$

$$\Pr(Y|x6) \propto \Pr(Y)\Pr(-1|Y)\Pr(+1|Y) = (0.5, 0.5) \times \left(\frac{1}{3}, 0\right) \times \left(\frac{2}{3}, \frac{1}{3}\right) = (\frac{2}{9}, 0) \propto (1, 0)$$

2.4 New class CPT

Based on the sum of the expected counts in 2.3, in terms of frequencies, we should see 2.4667 samples that belong to Y=1, and 3.5333 samples that belong to Y = 0. Same number of total samples as the original data set, but class counts skewed more to Y = 0 based on out updated EM.

Samples x4 and x5 simultaneously contribute Y = 0 and Y = 1 to the total.

New CPT of Pr(Y) (new prior):

$$\Pr(Y = 1, Y = 0) = \left(\frac{2.4667}{6}, \frac{3.5333}{6}\right) = (0.4111, 0.5889)$$

2.5 Maximization Step – New Feature CPTs

Instead of the original evenly split data, we now have the new fractional count data from 2.3:

$$\Pr(Y|x1) = (0, 1); \quad \Pr(Y|x2) = (0, 1); \quad \Pr(Y|x3) = (0, 1);$$

$$\Pr(Y|x4) = (\tfrac{2}{3}, \tfrac{1}{3}); \quad \Pr(Y|x5) = (\tfrac{4}{5}, \tfrac{1}{5}); \quad \Pr(Y|x6) = (1, 0);$$

For reference

Sample	F_1	F_2	Y
$\mathbf{x}_1$	$+1$	-1	0
$\mathbf{x}_2$	$+1$	$+1$	0
$\mathbf{x}_3$	0	0	0
$\mathbf{x}_4$	0	-1	1
$\mathbf{x}_5$	0	$+1$	1
$\mathbf{x}_6$	-1	$+1$	1

Feature CPTs calculation: $\Pr(f|y) = \frac{count(f|y)}{count(y)}$, computed in Excel:

F1|Y	'+1'	'0'	'-1'
Y=1	0.0000	0.5946	0.4054
Y=0	0.5660	0.4340	0.0000

F2|Y	'+1'	'0'	'-1'
Y=1	0.7297	0.0000	0.2703
Y=0	0.3396	0.2830	0.3774

Problem 3 Linear Classifiers

3.1 Perceptron Learning, learning rate = 1

Linear classifiers for this problem (2 features):

$$\boldsymbol{f_w = w_0 + w_1 F_1 + w_2 F_2}$$

$$\boldsymbol{if\ f_w \leq 0 \rightarrow Y = 0;\ \ Otherwise \rightarrow Y = 1}$$

Starting weight: $W = (w_0, w_1, w_2) = (1, 1, 1)$

Perception learning to process the six data in order once:

x1: $f = 1 + 1 * 1 + 1 * (-1) = 1 \rightarrow Y = 1 \neq 0;$ Incorrect, update weights

New weights: $W' = (1, 1, 1) + 1 * (0 - 1) * (1, 1, -1) = (0, 0, 2) \rightarrow W$

x2: $f = 0 + 0 * 1 + 2 * 1 = 2 \rightarrow Y = 1 \neq 0;$ Incorrect, update weights

New weights: $W' = (0, 0, 2) + 1 * (0 - 1) * (1, 1, 1) = (-1, -1, 1) \rightarrow W$

x3: $f = -1 + (-1) * 0 + 1 * 0 = -1 \rightarrow Y = 0;$ Correct, weights remain

x4: $f = -1 + (-1) * 0 + 1 * (-1) = -2 \rightarrow Y = 0 \neq 1;$ Incorrect, update weights

New weights: $W' = (-1, -1, 1) + 1 * (1 - 0) * (1, 0, -1) = (0, -1, 0) \rightarrow W$

x5: $f = 0 + (-1) * 0 + 0 * 1 = 0 \rightarrow Y = 0 \neq 1;$ Incorrect, update weights

New weights: $W' = (0, -1, 0) + 1 * (1 - 0) * (1, 0, 1) = (1, -1, 1) \rightarrow W$

x6: $f = 1 + (-1) * (-1) + 1 * 1 = 3 \rightarrow Y = 1;$ Correct, weights remain

Final weights after 1 round of learning: $W' = (1, -1, 1)$

3.2 Plotting in F1-F2 space with decision boundary from W'

Determining decision boundary:

$$f_w = w_0 + w_1 F_1 + w_2 F_2$$

Threshold happens when $f_w = 0$

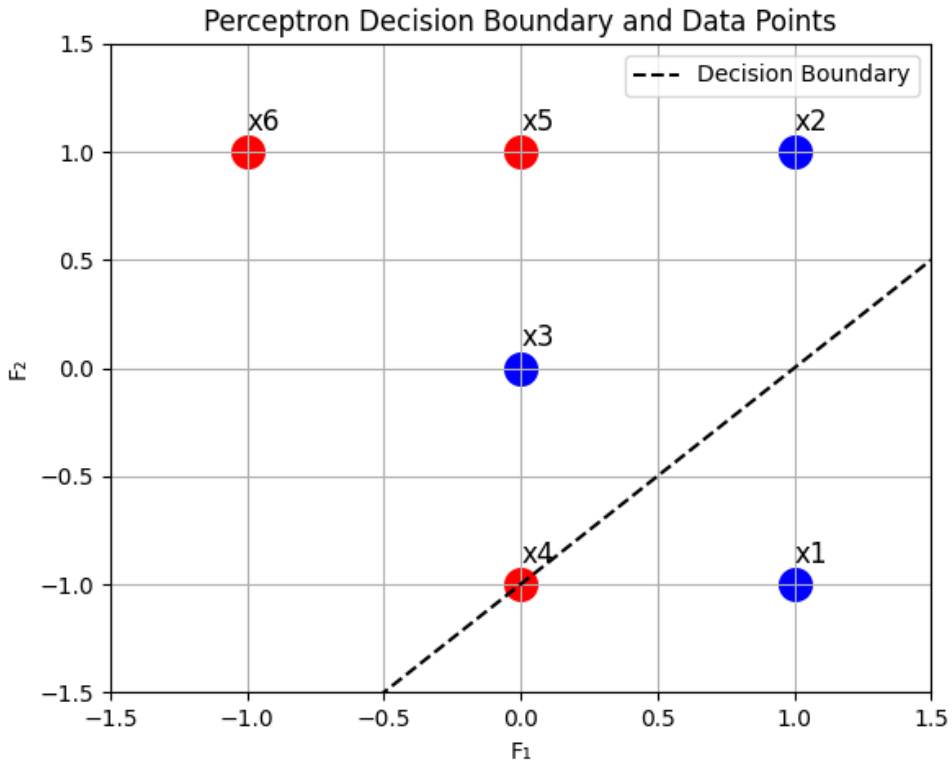
Plug in the new weights:

$$W' = (1, -1, 1)$$

$$1 - F_1 + F_2 = 0$$

The above is the equation of a line (decision boundary) in F1-F2 space

Data points plotted with Python (see code in Appendix)



As seen above in the plot, blue dots belong to class Y=0, red dots belong to class Y=1.

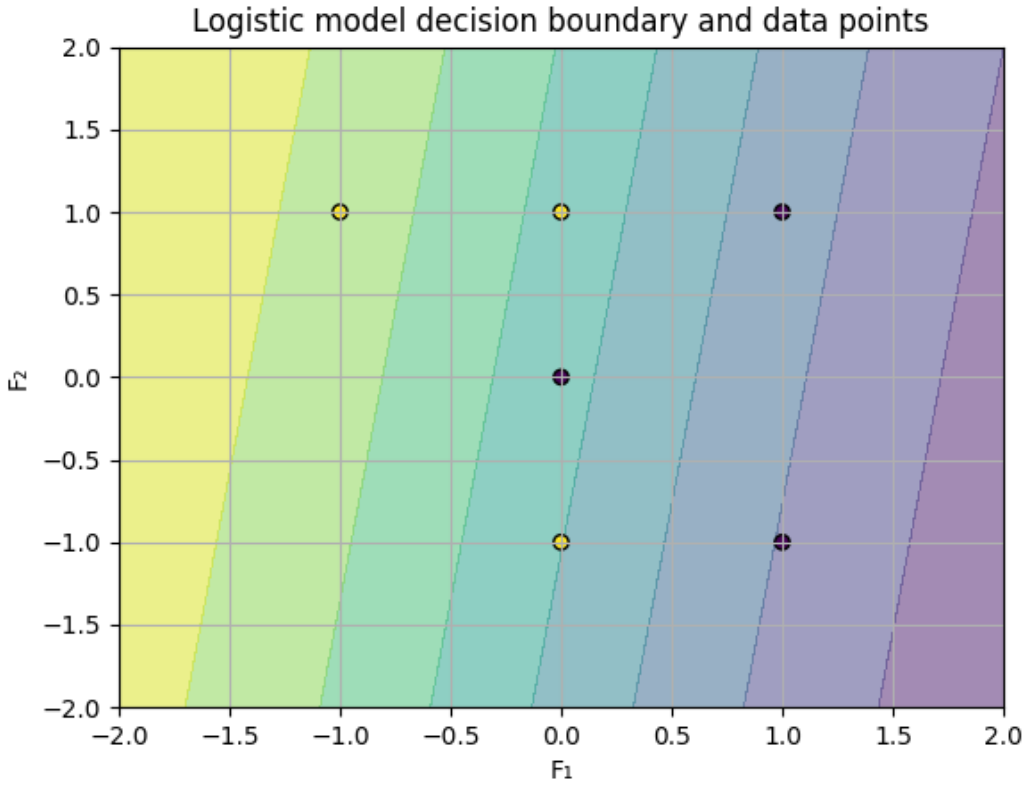
Clearly, the data points are **not** linearly separable.

Since $f_w \leq 0 \rightarrow Y = 0$, so according to the decision boundary, points that are below or at the boundary belong to class Y=0, points above the boundary belong to class Y=1. x1, x5, x6 are correctly classified, while x2, x3, x4, are not. The accuracy of the current linear classifier is 50%.

3.3 Logistic model

Logistic model fitted in Python based on the code provided in Lecture 10.

Generated scatter plot with DecisionBoundary-Display() (see code in Appendix)



The outputs of Logistic model are between [0, 1] thus can be interpreted as probabilities.

Learned weights are: w0 = [0.13315703]; [w1, w2] = [[-0.88423259 0.12516216]]

$$\boldsymbol{W'} = (\mathbf{0.1332}, -\mathbf{0.8842}, \mathbf{0.1252})$$

Classification accuracy: 0.8333

3.4 Mean negative log likelihood

$$h_w(x) = \sigma\big(f_w(x)\big) = \frac{1}{1 + \exp\left(-f_w(x)\right)}$$

Using predict_log_proba() to compute the log probability of each sample with the already learned weights based on logistic model. (see code in appendix)

log_probs: (For each sample and both classes)

[[-0.34808059 -1.2243183]

[-0.42838232 -1.05429572]

[-0.76194041 -0.62878338]

[-0.69715261 -0.68915773]

[-0.83062479 -0.5723056]

[-1.41942865 -0.27687687]]

Negative log-likelihoods: (for the right class label)

[0.34808059 0.42838232 0.76194041 0.68915773 0.5723056 0.27687687]

Mean negative log-likelihood: L = 0.5128

Current weight vector:

$$W' = (0.1332, -0.8842, 0.1252)$$

The gradients are calculated as such:

$$\frac{\partial L}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \qquad \frac{\partial L}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)\, x_{ij}$$

Gradient of the negative log likelihood: [1.30000e-05 1.47379e-01 -2.08500e-02]

Therefore:

$$\frac{\partial L}{w_0} = 0.000013; \qquad \frac{\partial L}{w_1} = 0.14379; \qquad \frac{\partial L}{w_2} = -0.02085$$

Appendix

3.2

```
import matplotlib.pyplot as plt

import numpy as np


print('\n')

# Feature vectors

x1 = np.array([+1, -1]) # [F1, F2]

x2 = np.array([+1, +1])

x3 = np.array([0, 0])

x4 = np.array([0, -1])

x5 = np.array([0, +1])

x6 = np.array([-1, +1])


X = [x1, x2, x3, x4, x5, x6]

Y = [0, 0, 0, 1, 1, 1] # Class labels

data = list(zip(X, Y)) # Compile into a list of tuples


plt.figure()

for i, (F, y) in enumerate(data, start=1):

  if y == 0:

    plt.scatter(F[0], F[1], c='blue', marker='o', s=200)

  else:

    plt.scatter(F[0], F[1], c='red', marker='o', s=200)

  plt.text(F[0], F[1]+0.1, f'x{i}', fontsize=12)
```

```
# Decision boundary with new Weights
f1_vals = np.linspace(-1.5, 1.5, 100) # Generate 100 points for F1
f2_vals = f1_vals - 1 # 1-F1+F2=0, so F2 = F1 - 1
plt.plot(f1_vals, f2_vals, 'k--', label='Decision Boundary')
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.xlabel('F1')
plt.ylabel('F2')
plt.grid(True)
plt.legend()
plt.title('Perceptron Decision Boundary and Data Points')
plt.show()
```

3.3 & 3.4

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.model_selection import train_test_split

print('\n')
# Feature vectors
x1 = np.array([1, -1]) # [F1, F2]
x2 = np.array([1, 1])
x3 = np.array([0, 0])
```

```
x4 = np.array([0, -1])
x5 = np.array([0, 1])
x6 = np.array([-1, 1])

X = np.array([x1, x2, x3, x4, x5, x6])
Y = np.array([0, 0, 0, 1, 1, 1]) # Class labels

# 3.3
classifier = LogisticRegression().fit(X, Y)
print("w0 =", classifier.intercept_)
print("[w1, w2] =", classifier.coef_)
print("Classification accuracy: ", classifier.score(X, Y))

disp = DecisionBoundaryDisplay.from_estimator(
   classifier, X, response_method="predict_proba",
   xlabel='F_1', ylabel='F_2',
   alpha=0.5)
disp.ax_.scatter(X[:, 0], X[:, 1], c=Y, edgecolor="k")

plt.grid(True)
plt.title('Logistic model decision boundary and data points')
plt.show()

# 3.4
print('\n')
# classifier is the already trained logistic model
```

```
log_probs = classifier.predict_log_proba(X) # Log probabilities of each samples with the learned weights

neg_log_L = -log_probs[np.arange(len(Y)), Y] # get the nll of being the right class for each sample

# the wrong class will have a log probability of 0, so it will not contribute to the negative log likelihood

mean_neg_log_L = np.mean(neg_log_L) # Taking the mean

print('log_probs:\n', log_probs)

print(f'Negative log-likelihoods: {neg_log_L}')

print(f'Mean negative log likelihood: {mean_neg_log_L:.4f}')

print('\n')


X_aug = np.hstack((np.ones((X.shape[0], 1)), X))  # Augment X with a X0 term

print("X_aug:\n", X_aug)

print("Y:", Y)

w = np.array([0.1332, -0.8842, 0.1252]) # Learned weights

f = X_aug @ w  # Linear combination

h = 1/(1+np.exp(-f))  # activation: Sigmoid function

gradient = (h - Y) @ X_aug / X.shape[0]  # Gradient calculation

print('Gradient of the negative log likelihood: ', gradient.round(6))

print('\n')
```