A black and white logo

Description automatically generated with low confidence

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΕΧΝΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ

ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

1Η ΣΕΙΡΑ ΓΡΑΠΤΩΝ ΑΣΚΗΣΕΩΝ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: Καμπουγέρης Χαράλαμπος ΑΜ: 03120098

ΜΑΘΗΜΑ: ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

ΑΚ. ΕΤΟΣ: 2023-2024 ΕΞΑΜΗΝΟ: 7Ο

Email: xarrisk2002@gmail.com / el20098@mail.ntua.gr

ΑΣΚΗΣΗ 1.1

Θεωρούμε το ακόλουθο μοντέλο παλινδρόμησης πάνω σε 11 ανεξάρτητες μεταβλητές xi, i=1, 2, . . . , 11 και

Για την επίλυση θα χρησιμοποιηθεί η γλώσσα python και το google collab.

Α)

Πρώτα θα φορτώσουμε τα δεδομένα του csv αρχείου.

Ορίζουμε το ερωτηματικό ως οριοθέτη και τη πρώτη σειρά του αρχείου ως τα ονόματα των στηλών. Επίσης μετατρέπουμε όλες τις στήλες στο Data Frame σε αριθμητικές τιμές.

file\_path = '/content/drive/My Drive/machine\_learning/ML2023-24-hwk1.csv'

# Load the data

data = pd.read\_csv(file\_path, delimiter=';', header=None, names=['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality'])

# Convert numeric columns to float

data = data.apply(pd.to\_numeric, errors='coerce')

Έπειτα κανουμε κανονικοποίηση αφαιρώντας τη μέση τιμή των δεδομένων και διαιρώντας με την απόκλιση για κάθε στήλη. Έτσι διασφαλίζουμε ότι κάθε δεδομένο έχει μέσο όρο 0 και τυπική απόκλιση 1.

# Normalize the Data

normalized\_data = (data - data.mean()) / data.std()

Μετά υπολογίζουμε τη συσχέτιση μεταξύ των “pH” και “sulphates” (r(9,10)).

# Calculate the Normalized Correlation Coefficient (r(9,10))

correlation\_9\_10 = normalized\_data['pH'].corr(normalized\_data['sulphates'])

print(f"Normalized Correlation Coefficient between pH and sulphates: {correlation\_9\_10}")

το αποτέλεσμα είναι:

Normalized Correlation Coefficient between pH and sulphates: -0.19664759122485098

O συντελεστής συσχέτισης -0,197 υποδηλώνει μια ασθενή αρνητική γραμμική σχέση μεταξύ του pH και του sulphates στο dataset μας.

Τέλος, φτιάχνουμε τον πίνακα συσχέτισης με scatterplots.

# Create a Correlation Matrix with Scatterplots

scatter\_matrix = pd.plotting.scatter\_matrix(normalized\_data, figsize=(12, 12), diagonal='hist')

plt.show()

A graph of blue and white dots

Description automatically generated with medium confidence

Χωρίζουμε επίσης τα δεδομένα σε training set και test set όπως ζητήθηκε.

# Define the number of rows for training and test sets

num\_training\_rows = 100

num\_test\_rows = 50

# Split the data into training and test sets

training\_data = normalized\_data.iloc[:num\_training\_rows, :]

test\_data = normalized\_data.iloc[num\_training\_rows:num\_training\_rows + num\_test\_rows, :]

Β)

Επειτα, αφαιρουμε τις σειρές με NaN τιμές, χωριζουμε τα features από τα target variables(quality).

import numpy as np

# Extract features (X) and target variable (y) for training data

X\_train = training\_data.drop('quality', axis=1)  # All columns except the last one

y\_train = training\_data['quality']   # Last column (quality)

X\_test = test\_data.drop('quality', axis=1)

y\_test = test\_data['quality']

Η γραμμική παλινδρόμηση είναι μια στατιστική μέθοδος που χρησιμοποιείται για τη μοντελοποίηση της σχέσης μεταξύ μιας εξαρτημένης μεταβλητής (στόχος) και μιας ή περισσότερων ανεξάρτητων μεταβλητών (χαρακτηριστικά ή προγνωστικοί παράγοντες). Ο στόχος είναι να βρεθεί η πιο κατάλληλη γραμμική σχέση που ελαχιστοποιεί τη διαφορά μεταξύ των προβλεπόμενων και των πραγματικών τιμών της μεταβλητής στόχου.

Ο τύπος για τον υπολογισμό των βαρών στη γραμμική παλινδρόμηση μπορεί να αναπαρασταθεί ως:

όπου:

* W το διάνυσμα των βαρών
* Χ ο πίνακας τιμών των features
* y είναι το διάνυσμα των τιμών μεταβλητής στόχου.

Το αποτέλεσμα w περιέχει τα βέλτιστα βάρη για την γραμμική παλινδρόμηση.

Οπότε, εφαρμόζουμε την φόρμουλα στα στοιχεία μας με τον παρακάτω κώδικα και μας δίνει αυτό το αποτέλεσμα.

#Calculate the weights using the linear regression formula

weights\_linear\_regression = np.linalg.pinv(X\_train.T @ X\_train) @ X\_train.T @ y\_train

# Display the weights

print("Weights for Linear Regression:")

for i, weight in enumerate(weights\_linear\_regression):

    print(f"w{i}: {weight}")

Weights for Linear Regression:

w0: 0.20262448489665985

w1: -0.42745766043663025

w2: -0.3318268358707428

w3: -0.006668483838438988

w4: -0.008222042582929134

w5: 0.2698926329612732

w6: -0.3115069270133972

w7: -0.0863097608089447

w8: -0.19337978959083557

w9: 0.019866107031702995

w10: 0.4241105020046234

Γ)

Στη συνέχεια θα βρούμε τα βάρη χρησιμοποιώντας Ridge Regression στα δεδομένα μας.

Ορίζουμε τις τιμές της υπερπαραμετρου λ σε 10, 100 και 200.

Η Ridge Regression είναι μια επέκταση της γραμμικής παλινδρόμησης που εισάγει έναν όρο τακτοποίησης στη συνάρτηση κόστους. Αυτή η τεχνική είναι ιδιαίτερα χρήσιμη όταν δύο ή περισσότερες ανεξάρτητες μεταβλητές συσχετίζονται σε μεγάλο βαθμό.

Ο τύπος για τον υπολογισμό των βαρών δίνεται από:

όπου:

* W το διάνυσμα των βαρών
* Χ ο πίνακας τιμών των features
* y είναι το διάνυσμα των τιμών μεταβλητής στόχου.
* Ι είναι το identity matrix

#Define the regularization parameters

lambda\_values = [10, 100, 200]

#Initialize an empty dictionary to store weights for each lambda

weights\_ridge\_regression = []

ridge\_weights = {}

#Iterate over lambda values

for lambda\_val in lambda\_values:

    #Ridge Regression formula

    w\_ridge = np.linalg.inv(X\_train.T @ X\_train + lambda\_val \* np.identity(X\_train.shape[1])) @ (X\_train.T @ y\_train)

    weights\_ridge\_regression.append(w\_ridge)

    #Store the weights in the dictionary

    ridge\_weights[lambda\_val] = w\_ridge

#Display the weights

for lambda\_val, weights in ridge\_weights.items():

    print(f"Weights for λ = {lambda\_val}:")

    for i, weight in enumerate(weights):

        print(f"w{i}: {weight}")

    print()

Weights for λ = 10:

w0: 0.18100236572219375

w1: -0.32868156787091013

w2: -0.21136125974714082

w3: 0.003151488951650794

w4: -0.0201447706106405

w5: 0.2002102939657162

w6: -0.2654465581847494

w7: -0.07115676847634034

w8: -0.16396430991367222

w9: 0.014855298800767164

w10: 0.35758755850870483

Weights for λ = 100:

w0: 0.10664283271953415

w1: -0.13515933651423595

w2: -0.012598286218773732

w3: 0.009675205793405153

w4: -0.0310310669960041

w5: 0.04474869803682916

w6: -0.12276156908053337

w7: -0.017653716800999995

w8: -0.07544580310819109

w9: -0.00011754689701604093

w10: 0.16824376006969455

Weights for λ = 200:

w0: 0.07466473696029755

w1: -0.08964547279936119

w2: 0.008288733594097175

w3: 0.007402421147498225

w4: -0.02662987012203488

w5: 0.019048169412965672

w6: -0.08078192560860473

w7: -0.007586363543530622

w8: -0.04922597370338143

w9: -0.0027821410863150837

w10: 0.10948081147742252Δ)

Στη συνέχεια, το κοινό γράφημα που παράγεται από τα προηγούμενα δεδομένα είναι:

A graph of different colored rectangles

Description automatically generated

Ε)

Root Mean Squared Error (RMSE) είναι μια τιμή που χρησιμοποιείται συνήθως για τη μέτρηση του μέσου μεγέθους των σφαλμάτων μεταξύ προβλεπόμενων και πραγματικών τιμών σε ένα πρόβλημα παλινδρόμησης. Παρέχει έναν τρόπο ποσοτικοποίησης της ακρίβειας ενός μοντέλου.

Για να το υπολογίσουμε θα κανουμε τα επόμενα βήματα:

1. Υπολογισμός της υπόλοιπης τιμής (error) :   
   υπόλοιπο = actual – predicted
2. Τετραγωνισμός υπολοίπου
3. Μέση τιμή τετραγωνισμένου υπολοίπου  
   μέση τιμή =

Αυτό θα το κανουμε και για τα training και για τα validation sets.

rom sklearn.metrics import mean\_squared\_error

from math import sqrt

# Linear Regression

# Calculate predictions for the training set

y\_train\_pred\_linear = X\_train @ weights\_linear\_regression

# Calculate predictions for the test set

y\_test\_pred\_linear = X\_test @ weights\_linear\_regression

# Calculate RMSE for training set

rmse\_train\_linear = sqrt(mean\_squared\_error(y\_train, y\_train\_pred\_linear))

print(f"RMSE for Linear Regression (Training): {rmse\_train\_linear}")

# Calculate RMSE for test set

rmse\_test\_linear = sqrt(mean\_squared\_error(y\_test, y\_test\_pred\_linear))

print(f"RMSE for Linear Regression (Test): {rmse\_test\_linear}")

print()

# Ridge Regression

for lambda\_val, weights in ridge\_weights.items():

    # Calculate predictions for the training set

    y\_train\_pred\_ridge = X\_train @ weights

    # Calculate predictions for the test set

    y\_test\_pred\_ridge = X\_test @ weights

    # Calculate RMSE for training set

    rmse\_train\_ridge = sqrt(mean\_squared\_error(y\_train, y\_train\_pred\_ridge))

    print(f"RMSE for Ridge Regression (Training) with λ = {lambda\_val}: {rmse\_train\_ridge}")

    # Calculate RMSE for test set

    rmse\_test\_ridge = sqrt(mean\_squared\_error(y\_test, y\_test\_pred\_ridge))

    print(f"RMSE for Ridge Regression (Test) with λ = {lambda\_val}: {rmse\_test\_ridge}")

    print()

και το αποτέλεσμα που παράγεται είναι:  
  
RMSE for Linear Regression (Training): 0.7156631541673862

RMSE for Linear Regression (Test): 0.6819783710800131

RMSE for Ridge Regression (Training) with λ = 10: 0.7168834205261595

RMSE for Ridge Regression (Test) with λ = 10: 0.608263983248362

RMSE for Ridge Regression (Training) with λ = 100: 0.7978857359579214

RMSE for Ridge Regression (Test) with λ = 100: 0.5455839303648995

RMSE for Ridge Regression (Training) with λ = 200: 0.8363143974201055

RMSE for Ridge Regression (Test) with λ = 200: 0.584786817771439

Η επιλογή της κατάλληλης τιμής του λ χρειάζεται έναν συμβιβασμό μεταξύ της καλής προσαρμογής των δεδομένων εκπαίδευσης και της αποτροπής της υπερπροσαρμογής.

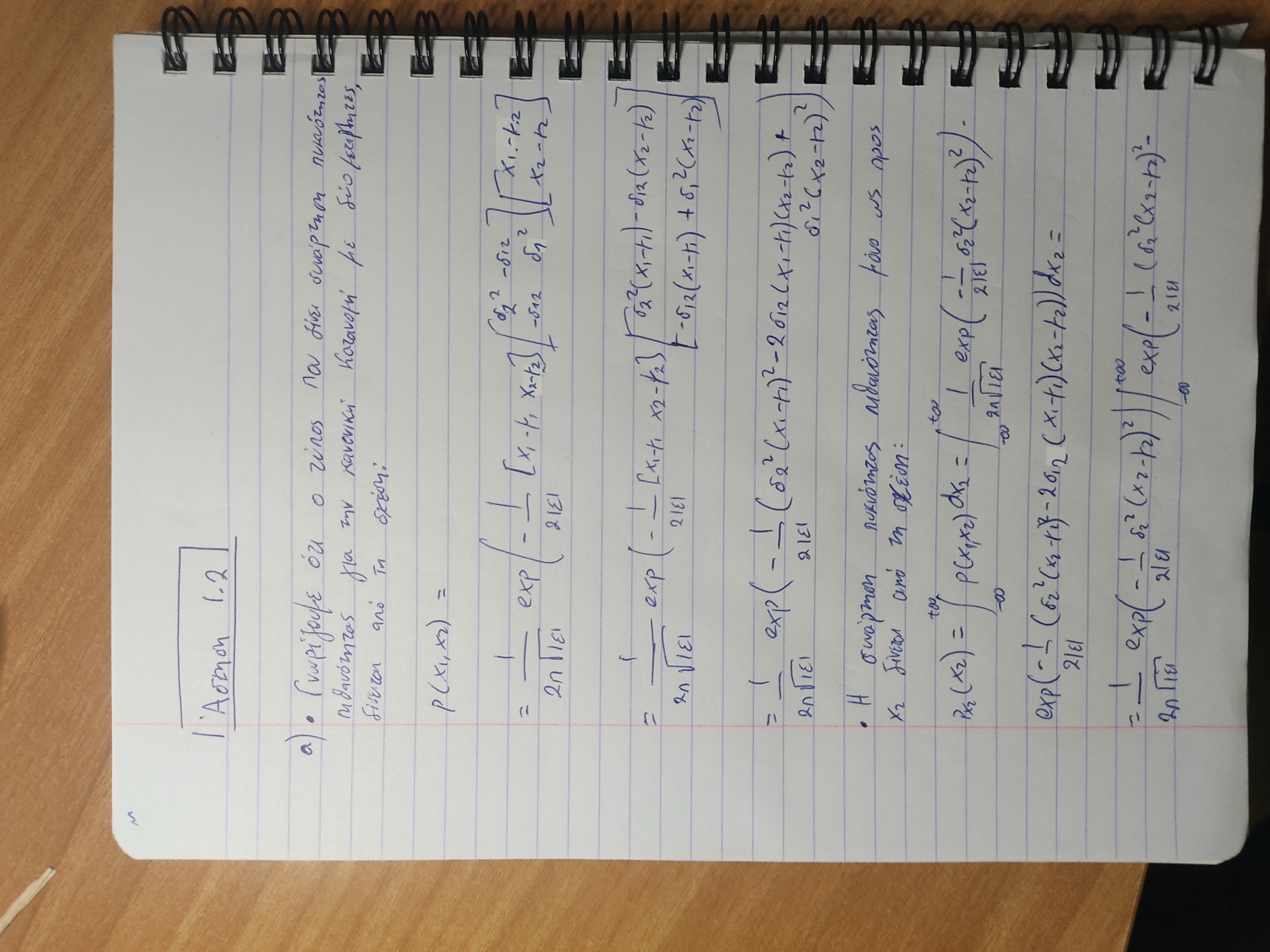
* Linear Regression (λ = N/A):  
  Training RMSE: 0.7156631541673862 Validation RMSE: 0.6819783710800131  
  Το μοντέλο γραμμικής παλινδρόμησης χωρίς κανονικοποίηση (λ = N/A) επιτυγχάνει ένα σχετικά χαμηλό RMSE τόσο στα σύνολα εκπαίδευσης όσο και στα σύνολα validation. Αυτό υποδηλώνει ότι το μοντέλο μπορεί να μην ταιριάζει υπερβολικά με τα δεδομένα εκπαίδευσης.
* Ridge Regression:  
  λ = 10: Training RMSE: 0.7168834205261595Validation RMSE: 0.608263983248362

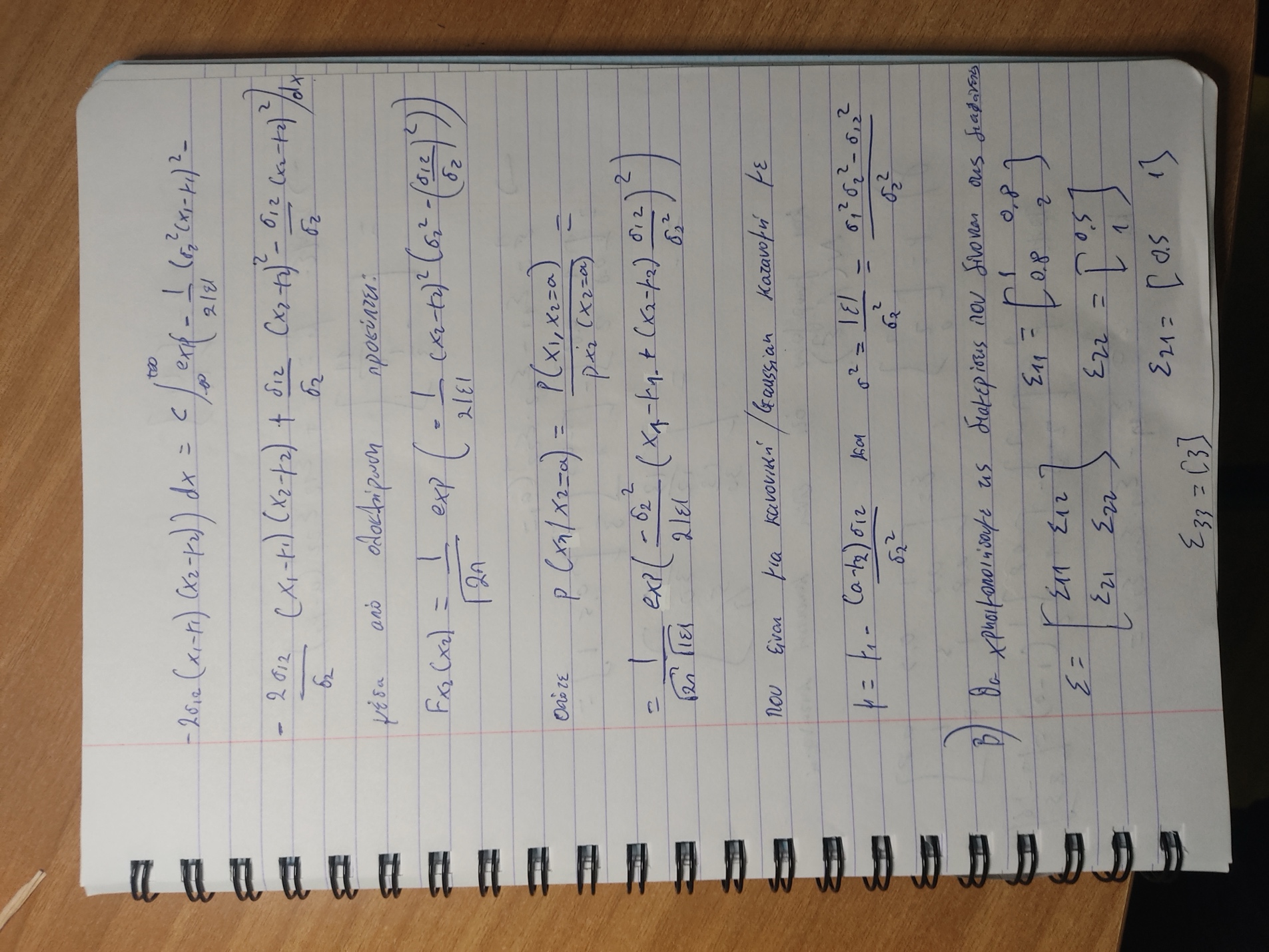
λ = 100: Training RMSE: 0.7978857359579214 Validation RMSE: 0.545583930364899  
λ = 200: Training RMSE: 0.8363143974201055 Validation RMSE: 0.584786817771439

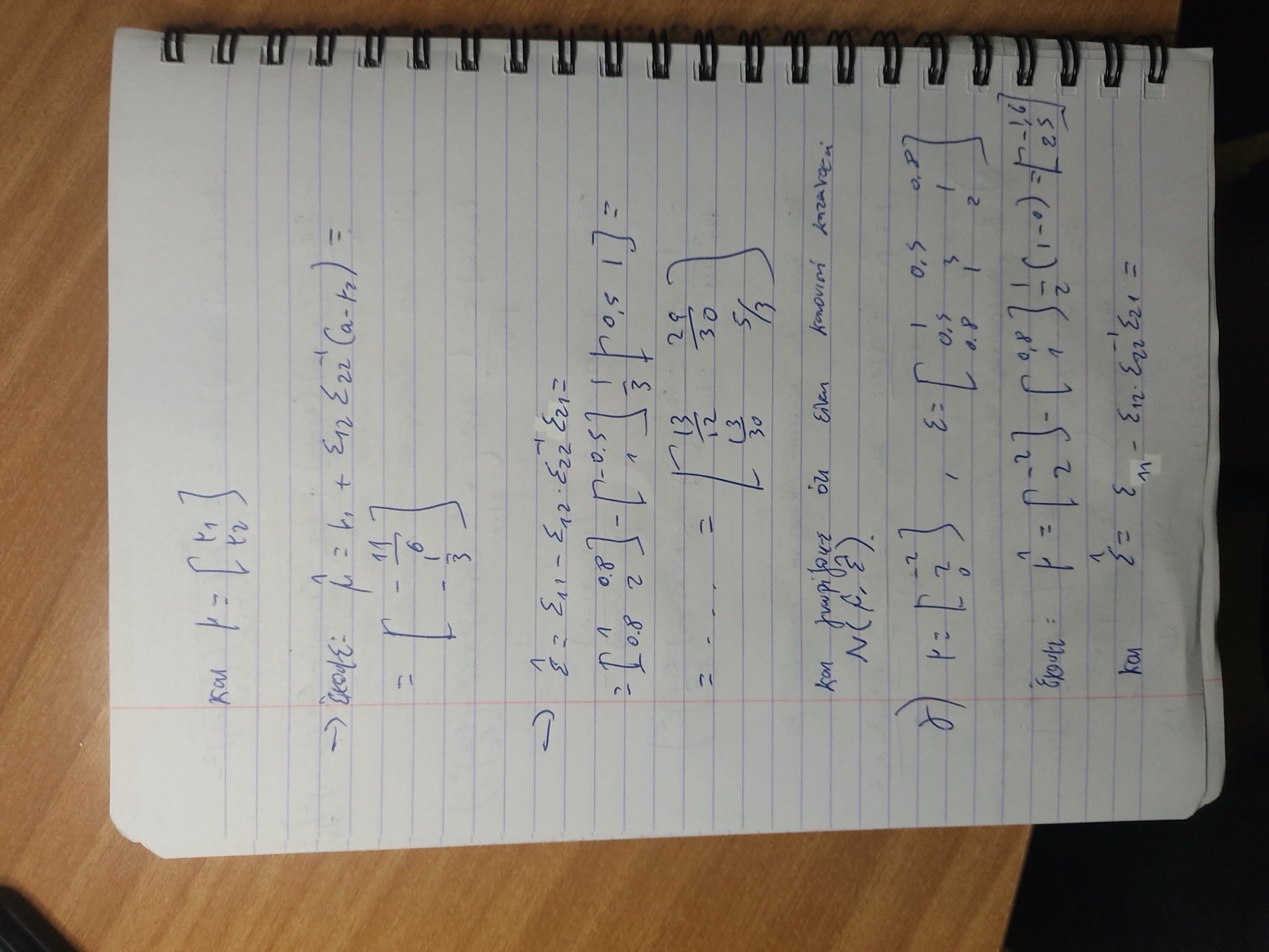
Tο λ = 10 παρέχει κανονικοποίηση διατηρώντας τις σχετικά χαμηλές τιμές RMSE. Επιτυγχάνει ισορροπία μεταξύ της προσαρμογής των δεδομένων προπόνησης και της αποτροπής της υπερπροσαρμογής.

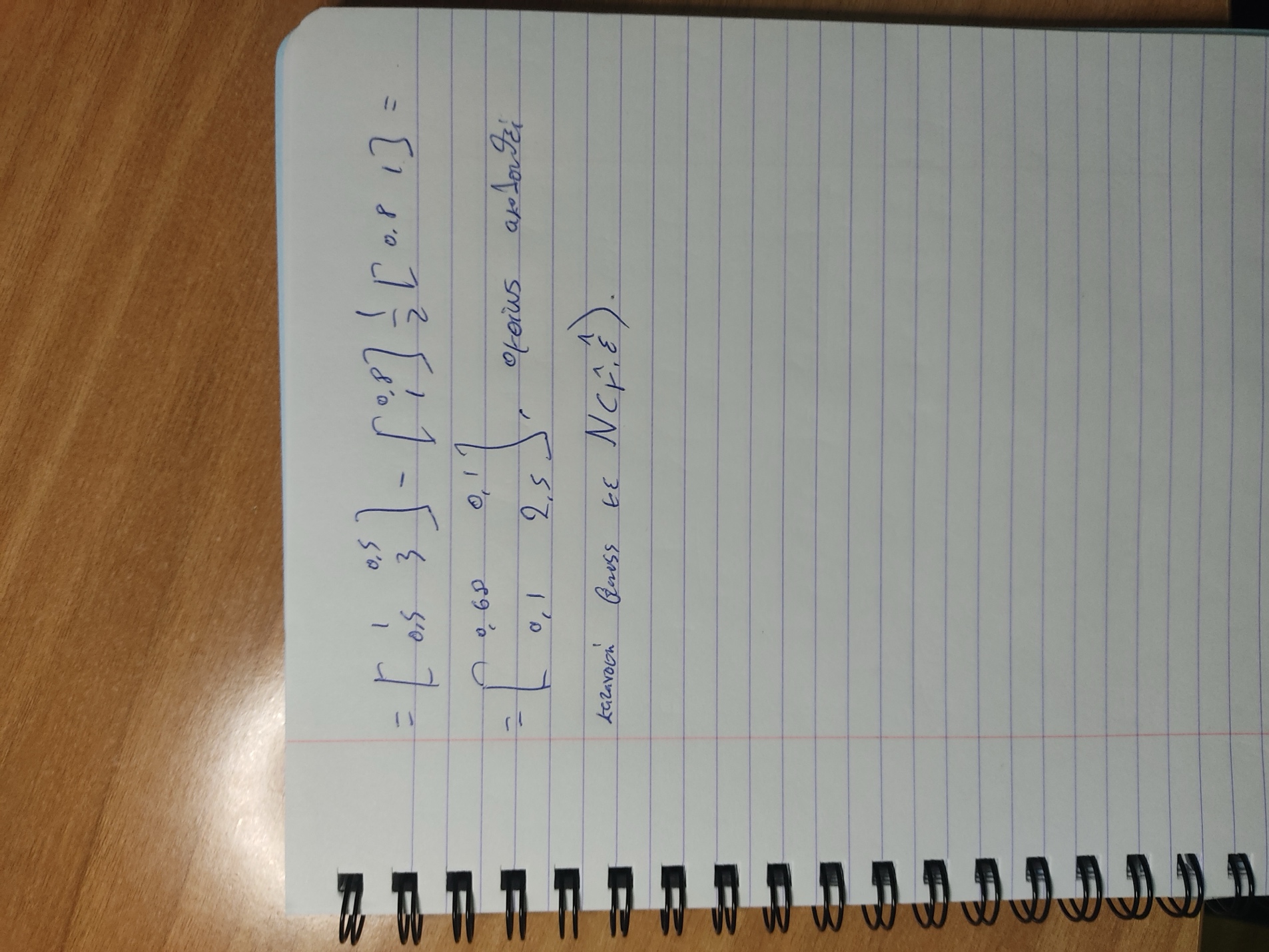
ΑΣΚΗΣΗ 1.2

Ακολουθεί χειρόγραφη απάντηση στα ερωτήματα:









ΑΣΚΗΣΗ 1.3

(α)

Για τα σημεία δύο κλάσεων που ακολοθούν κανονική κατανομή με κοινούς πίνακες συμμεταβλητότητας για τις δύο κλάσεις ισχύει σύμφωνα με τις σημειώσεις της αντίστοιχης ενότητας :

Επειδή P(ω1) = P(ω2) = 0.5 προκύπτει τελικά,

x0 = (μ1+μ2)/2 ,

θ = Σ-1 (μ1-μ2),

g(x) = θΤ (x-x0) = 0.

Επίσης Σ = Ι.

Με βάση τα παραπάνω και τα δεδομένα της άσκησης ακολουθεί ο παρακάτω κώδικας για υπολογισμό του Decision Boundary.

import numpy as np

# Given parameters

mu1 = np.array([-2, 0])

mu2 = np.array([2, 1])

cov\_matrix = np.eye(2) # Common covariance matrix I

P\_1 = 0.5

P\_2 = 0.5

# Calculate the weight vector (w) and bias term (b)

cov\_matrix\_inv = np.linalg.inv(cov\_matrix)

bT = 2\*np.dot((mu2 - mu1).T,cov\_matrix\_inv)

c = np.dot(mu1.T, np.dot(cov\_matrix\_inv, mu1)) - np.dot(mu2.T, np.dot(cov\_matrix\_inv, mu2)) - np.log(P\_1/P\_2)

# Print the decision boundary formula

print(f"Decision Boundary Formula: {bT[0]} \* x1 + {bT[1]} \* x2 + ({c}) = 0")

Decision Boundary Formula: 8.0 \* x1 + 2.0 \*x2 + -1.0 = 0

Δηλαδή, 8x1 + 2x2 - 1 = 0.

(β)

import numpy as np

# Given parameters

mu1 = np.array([-2, 0])

mu2 = np.array([2, 1])

cov\_matrix = np.array([[1, -0.6],[-0.6, 1]]) # Common covariance matrix

P\_1 = 0.5

P\_2 = 0.5

# Calculate the weight vector (w) and bias term (b)

cov\_matrix\_inv = np.linalg.inv(cov\_matrix)

bT = 2\*np.dot((mu2 - mu1).T,cov\_matrix\_inv)

c = np.dot(mu1.T, np.dot(cov\_matrix\_inv, mu1)) - np.dot(mu2.T, np.dot(cov\_matrix\_inv, mu2)) - np.log(P\_1 / P\_2)

# Print the decision boundary formula

print(f"Decision Boundary Formula: {bT[0]} \* x1 + {bT[1]} \* x2 + ({c}) = 0")

Decision Boundary Formula: 14.375 \* x1 + 10.625 \* x2 + (-5.3125) = 0

Δηλαδή, 14.375 x1 + 10.625 x2 – 5.3125 = 0

(γ)

Λόγω των συντελεστών βαρύτητας των σφαλμάτων ταξινόμησης ισχύει για ταξινόμηση στην ω1 κλάση χωρίς βλάβη της γενικότητας:

𝜆12 p(𝜔1) p (𝒙 |𝜔1) > 𝜆21 p(𝜔2) p(𝒙|𝜔2) ==>

==> 2 p (𝒙 |𝜔1) – p (𝒙 |𝜔2) > 0

Οπότε η εξίσωση g(x) = p (𝜔1 | 𝒙) – p(𝜔2 | 𝒙) = 0 για την υπερεπιφάνεια απόφασης θα γίνει

g(x) = ln[p(x|ω1)/p(x|ω2)] – ln[p(ω1)/p(ω2)] + ln2 = 0

Τελικά, προκύπτει :

import numpy as np

# Given parameters

mu1 = np.array([-2, 0])

mu2 = np.array([2, 1])

cov\_matrix = np.array([[1, -0.6],[-0.6, 1]]) # Common covariance matrix

P\_1 = 0.5

P\_2 = 0.5

# Calculate the weight vector (w) and bias term (b)

cov\_matrix\_inv = np.linalg.inv(cov\_matrix)

bT = 2\*np.dot((mu2 - mu1).T,cov\_matrix\_inv)

c = np.dot(mu1.T, np.dot(cov\_matrix\_inv, mu1)) - np.dot(mu2.T, np.dot(cov\_matrix\_inv, mu2)) - np.log(P\_1 / P\_2)+ np.log(2)

# Print the decision boundary formula

print(f"Decision Boundary Formula: {bT[0]} \* x1 + {bT[1]} \* x2 + ({c}) = 0")

Output :

Decision Boundary Formula: 14.375 \* x1 + 10.625 \* x2 + (-4.619352819440055) = 0

Δηλαδή, 14.375 x1 + 10.625 x2 – 4.619352819440055 = 0

(δ)

Ο κώδικας για τα ζητούμενα είναι ο ακόλουθος :

# Generate 200 points for each class

def generate\_points(mu, sigma, num\_points):

return np.random.multivariate\_normal(mu, sigma, num\_points)

mu1 = np.array([-2, 0])

mu2 = np.array([2, 1])

#case 1

cov\_matrix = np.eye(2) # Common covariance matrix I

P\_1 = 0.5

P\_2 = 0.5

# Calculate the weight vector (w) and bias term (b)

cov\_matrix\_inv = np.linalg.inv(cov\_matrix)

bT\_a = 2\*np.dot((mu2 - mu1).T,cov\_matrix\_inv)

c\_a = np.dot(mu1.T, np.dot(cov\_matrix\_inv, mu1)) - np.dot(mu2.T, np.dot(cov\_matrix\_inv, mu2)) - np.log(P\_1 / P\_2)

x\_class1\_a = generate\_points(mu1, cov\_matrix, 200)

x\_class2\_a = generate\_points(mu2, cov\_matrix, 200)

#case 2

cov\_matrix = np.array([[1, -0.6],[-0.6, 1]]) # Common covariance matrix

P\_1 = 0.5

P\_2 = 0.5

cov\_matrix\_inv = np.linalg.inv(cov\_matrix)

bT\_b = 2\*np.dot((mu2 - mu1).T,cov\_matrix\_inv)

c\_b = np.dot(mu1.T, np.dot(cov\_matrix\_inv, mu1)) - np.dot(mu2.T, np.dot(cov\_matrix\_inv, mu2)) - np.log(P\_1 / P\_2)

x\_class1\_b = generate\_points(mu1, cov\_matrix, 200)

x\_class2\_b = generate\_points(mu2, cov\_matrix, 200)

#case 3

cov\_matrix = np.array([[1, -0.6],[-0.6, 1]]) # Common covariance matrix

P\_1 = 0.5

P\_2 = 0.5

cov\_matrix\_inv = np.linalg.inv(cov\_matrix)

bT\_c = 2\*np.dot((mu2 - mu1).T,cov\_matrix\_inv)

c\_c = np.dot(mu1.T, np.dot(cov\_matrix\_inv, mu1)) - np.dot(mu2.T, np.dot(cov\_matrix\_inv, mu2)) - np.log(P\_1 / P\_2) + np.log(2)

x\_class1\_c = generate\_points(mu1, cov\_matrix, 200)

x\_class2\_c = generate\_points(mu2, cov\_matrix, 200)

import matplotlib.pyplot as plt

# Function to plot the points, class centers, and decision boundaries

def plot\_points\_decision\_boundary(x\_class1, x\_class2, mu1, mu2, bT, c):

# Plot the points

plt.scatter(x\_class1[:, 0], x\_class1[:, 1], color='blue', label='Class 1')

plt.scatter(x\_class2[:, 0], x\_class2[:, 1], color='red', label='Class 2')

# Plot the class centers

plt.scatter(mu1[0], mu1[1], color='green', marker='x', label='Class 1 Center')

plt.scatter(mu2[0], mu2[1], color='orange', marker='x', label='Class 2 Center')

# Plot the decision boundarycxcxx cfv

x = np.linspace(-6, 6, 100)

#y = np.dot(bT,x) + c

y = (-bT[0] \* x - c) / bT[1]

plt.plot(x, y, color='black', label='Decision Boundary')

# Set plot labels and legend

plt.xlabel('x1')

plt.ylabel('x2')

plt.legend()

# Show the plot

plt.show()

# Plot for case 1

plot\_points\_decision\_boundary(x\_class1\_a, x\_class2\_a, mu1, mu2, bT\_a, c\_a)

# Plot for case 2

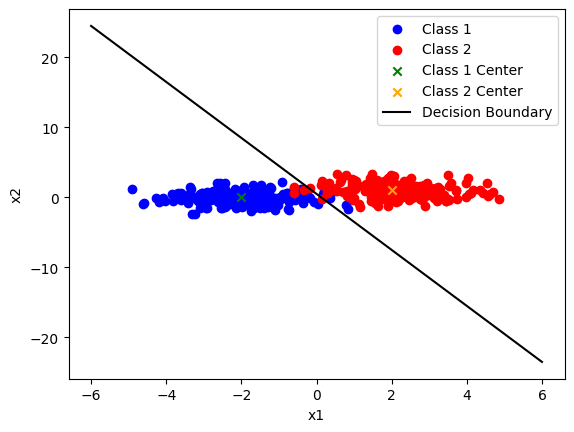
plot\_points\_decision\_boundary(x\_class1\_b, x\_class2\_b, mu1, mu2, bT\_b, c\_b)

# Plot for case 3

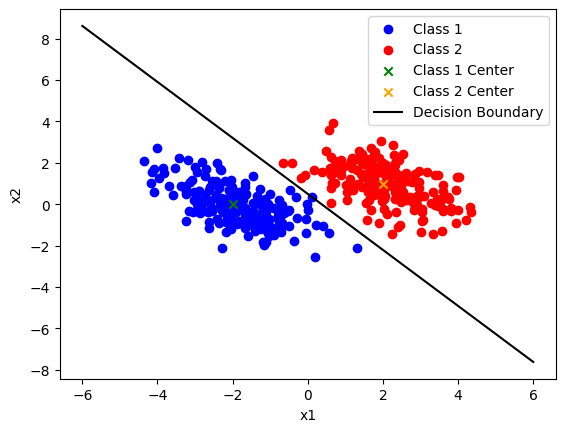
plot\_points\_decision\_boundary(x\_class1\_b, x\_class2\_b, mu1, mu2, bT\_c, c\_c)

Τα διαγράμματα που ακολουθούν είναι τα εξής :

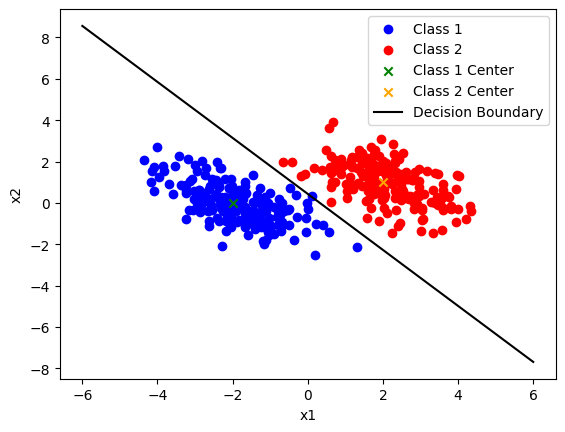
Α)



Β)



Γ)



ΑΣΚΗΣΗ 1.4

Α)

Θα χρησιμοποιήσουμε τον αλγόριθμο perceptron με (ω1, ω2, ω3, ω4) = (1,0,0,0)Τ = ΤΤ

**Δείγμα : χ(1)  =(1,4,3,6)Τ**

Έξοδος του perceptron: z = ΤΤ ⋅ χ(1) = (1,0,0,0)Τ ⋅ (1,4,3,6)Τ = 1 , step(2)=1 για z≥0, 0 για z<0

y = step (1) = 1 άρα y = 1

Άρα από τα αποτελέσματα της κλάσης έχουμε false positive (FP)

Διάνυσμα μεταβολής βαρών: error = 0 – y = 0 – 1 = -1

ΔΤ = n ⋅ e ⋅ x(1) = (-1,-4,-3,-6)T

Και το ανανεωμένο διάνυσμα βαρών είναι T=T+ΔΤ=(1,0,0,0)Τ + (-1,-4,-3,-6) Τ = (0,-4,-3,-6) Τ

**Δείγμα: x(2) = (1,2,-2,3)T**

Έξοδος του perceptron: z = ΤΤ ⋅ χ(2) = (0,-4,-3,-6)Τ ⋅ (1,2,-2,3)Τ = -20

Άρα y = step (-20) = 0 άρα y =0

Το δείγμα είναι false negative (FN)

Διάνυσμα μεταβολής βαρών: error = 1 – y = 1 – 0 = 1

ΔΤ = n ⋅ e ⋅ x(2) = (1,2,-2,3)T

Και το ανανεωμένο διάνυσμα βαρών είναι T= T+ΔΤ=(0,-4,-3,-6)Τ + ( 1,2,-2,3)Τ = (1,-2,-5,-3)Τ

**Δείγμα: x(3) = (1,1,0,-3)T**

Έξοδος του perceptron: z = ΤΤ ⋅ χ(3) = (1,-2,-5,-3)Τ ⋅ (1,1,0,-3)Τ = 8

Άρα y = step (8) = 1 άρα y =1

Το δείγμα είναι true positive (TP)

Διάνυσμα μεταβολής βαρών: error = 1 - y = 1 – 1 = 0

ΔΤ = n ⋅ e ⋅ x(3) = (0,0,0,0)T

Και το ανανεωμένο διάνυσμα βαρών είναι T= T+ΔΤ=(1,-2,-5,-3)Τ + (0,0,0,0)Τ = (1,-2,-5,-3)Τ

**Δείγμα: x(4) = (1,4,2,3)T**

Έξοδος του perceptron: z = ΤΤ ⋅ χ(4) = (1,-2,-5,-3)Τ ⋅ (1,4,2,3)Τ = -26

Άρα y = step (-26) = 0 άρα y =0

Το δείγμα είναι true negative (TN)

Διάνυσμα μεταβολής βαρών: error = 0 - y = 0 – 0 = 0

ΔΤ = n ⋅ e ⋅ x(4) = (0,0,0,0)T

Και το ανανεωμένο διάνυσμα βαρών είναι T= T+ΔΤ=(1,-2,-5,-3)Τ + (0,0,0,0)Τ = (1,-2,-5,-3)Τ

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Εποχή | Δείγμα | Ετικέτα | Σwixi | Έξοδος y | Χαρακτηρισμός | ΔΤ | Τ |
| 1 | (1,4,3,6) | 0 | 1 | 1 | FP | (-1,-4,-3,-6) | (0,-4,-3,-6) |
| 1 | (1,2,-2,3) | 1 | -20 | 0 | FN | (1,2,-2,3) | (1,-2,-5,-3) |
| 1 | (1,1,0,-3) | 1 | 8 | 1 | TP | (0,0,0,0) | (1,-2,-5,-3) |
| 1 | (1,4,2,3) | 0 | -26 | 0 | TN | (0,0,0,0) | (1,-2,-5,-3) |
| 2 | (1,4,3,6) | 0 | -40 | 0 | TN | (0,0,0,0) | (1,-2,-5,-3) |
| 2 | (1,2,-2,3) | 1 | -2 | 0 | FN | (1,2,-2,3) | (2,0,-7,0) |
| 2 | (1,1,0,-3) | 1 | 2 | 1 | TP | (0,0,0,0) | (2,0,-7,0) |
| 2 | (1,4,2,3) | 0 | -12 | 0 | TN | (0,0,0,0) | (2,0,-7,0) |
| 3 | (1,4,3,6) | 0 | -19 | 0 | TN | (0,0,0,0) | (2,0,-7,0) |
| 3 | (1,2,-2,3) | 1 | 16 | 1 | TP | (0,0,0,0) | (2,0,-7,0) |
| 3 | (1,1,0,-3) | 1 | 2 | 1 | TP | (0,0,0,0) | (2,0,-7,0) |
| 3 | (1,4,2,3) | 0 | -12 | 0 | TN | (0,0,0,0) | (2,0,-7,0) |

Β)

Η περιοχή `x1 ≥ 2` θα ταξινομηθεί από τον πρώτο νευρώνα, με την οριακή εξίσωση του νευρώνα 1 να είναι `x1 - 2 = 0`. Γενικά, ισχύει η εξίσωση `w11x1 + w12x2 + b1 = x1 - 2`.

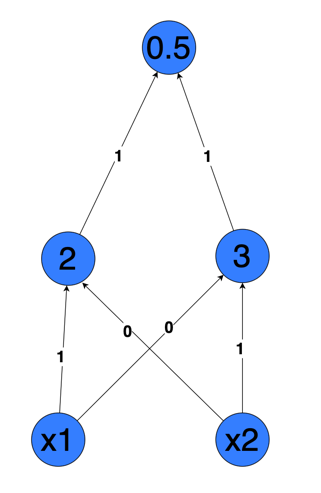
Για την επιλογή των βαρών, θα θέσουμε `w11 = 1` και `w12 = 0`, καθώς το `x1` συμβάλει στη λήψη απόφασης, σε αντίθεση με το `x2` το οποίο δεν συμβάλει. Επομένως, η εξίσωση μειώνεται σε `w11x1 + b1 = x1 + b1`. Δεδομένου ότι `w11x1 + b1 = x1 - 2`, προκύπτει ότι `b1 = -2`.

Η περιοχή `x2 ≥ 3` θα ταξινομηθεί από τον δεύτερο νευρώνα, με την οριακή εξίσωση του νευρώνα 2 να είναι `x2 - 3 = 0`. Γενικά, ισχύει η εξίσωση `w21x1 + w22x2 + b2 = x2 - 3`.

Για την επιλογή των βαρών, θα θέσουμε `w21 = 0` και `w22 = 1`, καθώς το `x1` δεν συμβάλει στη λήψη απόφασης, σε αντίθεση με το `x2` το οποίο συμβάλει. Έτσι, η εξίσωση μειώνεται σε `w21x1 + w22x2 + b2 = x2 + b2`. Δεδομένου ότι `w21x1 + w22x2 + b2 = x2 - 3`, προκύπτει ότι `b2 = -3`.

Ο τρίτος νευρώνας θα συνδυάσει τους προηγούμενους δύο για να εξάγει το τελικό αποτέλεσμα. Έστω `a1` το αποτέλεσμα του πρώτου νευρώνα και `a2` το αποτέλεσμα του δεύτερου νευρώνα. Τα αποτελέσματα είναι 0 ή 1, με 0 αν ταξινομείται στην αρνητική κλάση και 1 αν ταξινομείται στη θετική.

Άρα, `w1a1 + w2a2 + b3 = 0`. Θέλω `w1 = w2 = 1`, αφού εξίσου τα `a1` και `a2` συμβάλλουν στη λήψη απόφασης, και `b3 = -0.5`, αφού έστω και ένα γινόμενο `(w1a1, w2a2)` να είναι '1', ταξινομείται στη θετική κλάση. Άρα συνολικά:



ΑΣΚΗΣΗ 1.5

Γραμμικό SVM:

* Χρησιμοποιείται για γραμμικά διαχωρίσιμα δεδομένα, όπου μπορούν να ταξινομηθούν σε δύο κλάσεις με μια ευθεία γραμμή.
* Παρατηρείται στα (3) και (4).

Μη Γραμμικό SVM:

* Χρησιμοποιείται για μη γραμμικά διαχωρίσιμα δεδομένα.
* Εφαρμόζεται στα (1), (2), (5) και (6).

Παράμετρος C:

* Είναι παράμετρος τακτοποίησης που επηρεάζει την αντιστάθμιση μεταξύ του χαμηλού σφάλματος εκπαίδευσης και του χαμηλού σφάλματος δοκιμής.
* Όσο μικρότερο είναι το C, τόσο ενθαρρύνει ένα ευρύτερο περιθώριο αλλά αποδέχεται περισσότερα λάθη εκπαίδευσης.
* Όσο μεγαλύτερο είναι το C, οδηγεί σε μικρότερο περιθώριο αλλά λιγότερα λάθη εκπαίδευσης.
* Άρα, αν `a -> 4` και `b -> 3`.

Μη Γραμμικοί Πυρήνες SVM:

* Οι μη γραμμικοί πυρήνες SVM (γ, δ, ε) διαφέρουν στις μαθηματικές τους μορφές και εφαρμόζονται μεταξύ τους ανάλογα με την πολυπλοκότητα του προβλήματος.

Radial-Basis Function (RBF):

* Η συνάρτηση RBF ορίζεται ως: exp(1/(2σ2)||x-x1||2)
* Αντιστοιχεί στα (δ) και (ε),
* Αντιστοιχεί στα σχήματα (1) και (6).

Παράμετρος σ:

* Ελέγχει το πλάτος της καμπύλης σε ένα σχήμα καμπάνας Gauss. Επηρεάζει το όριο απόφασης.
* Όταν το σ είναι μικρό, η καμπύλη Gauss στενεύει, και το όριο απόφασης τείνει να ακολουθεί τις μικρές λεπτομέρειες των δεδομένων εκπαίδευσης, δημιουργώντας ένα πιο περίπλοκο όριο απόφασης.
* Αντιθέτως, όσο μεγαλύτερο είναι σ, καθιστά τον πυρήνα πιο σφαιρικό, καθώς το όριο είναι πιο ομαλό. Άρα δ->1 ε->6
* Όταν το σ αυξάνεται, το 1/2σ2 μειώνεται.

Μη Γραμμικό SVM:

* Ο πυρήνας k(u,v) = uTv + (uTv)2 καταγράφει τετραγωνικές σχέσεις.
* Το όριο απόφασης αντιπροσωπεύει μια καμπύλη διαχωρισμού στον χώρο εισόδου, άρα γ->2.

Οπότε ισχύει

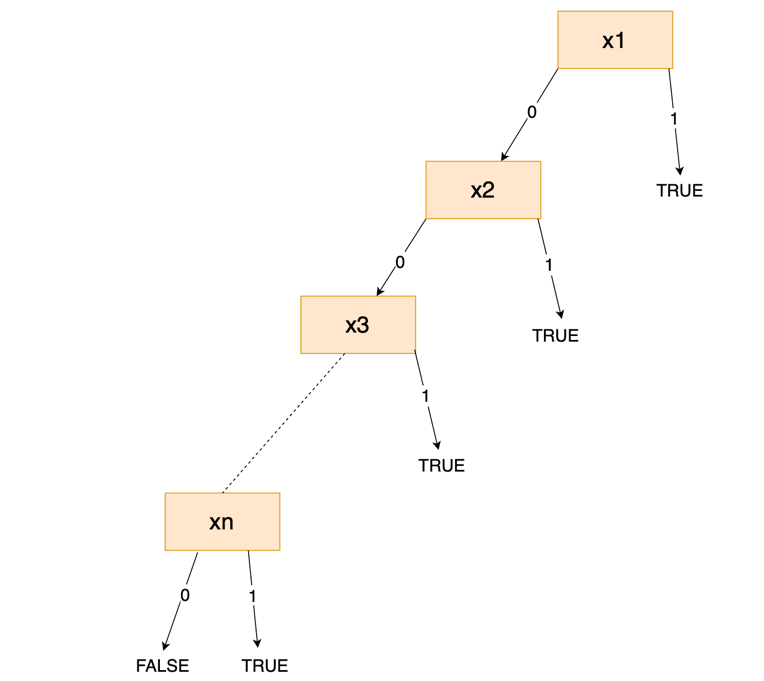
* α->4
* β->3
* γ->2
* δ->1
* ε->6
* στ->5

ΑΣΚΗΣΗ 1.6

Μέρος 1:

Χωρίς βλάβη της γενικότητας θεωρώ: *fn*​=(*x*1​∨*x*2​)∨(*x*3​∨*x*4​)∨…∨(*xn*−1​∨*xn*​)

Για το δέντρο απόφασης έχουμε:

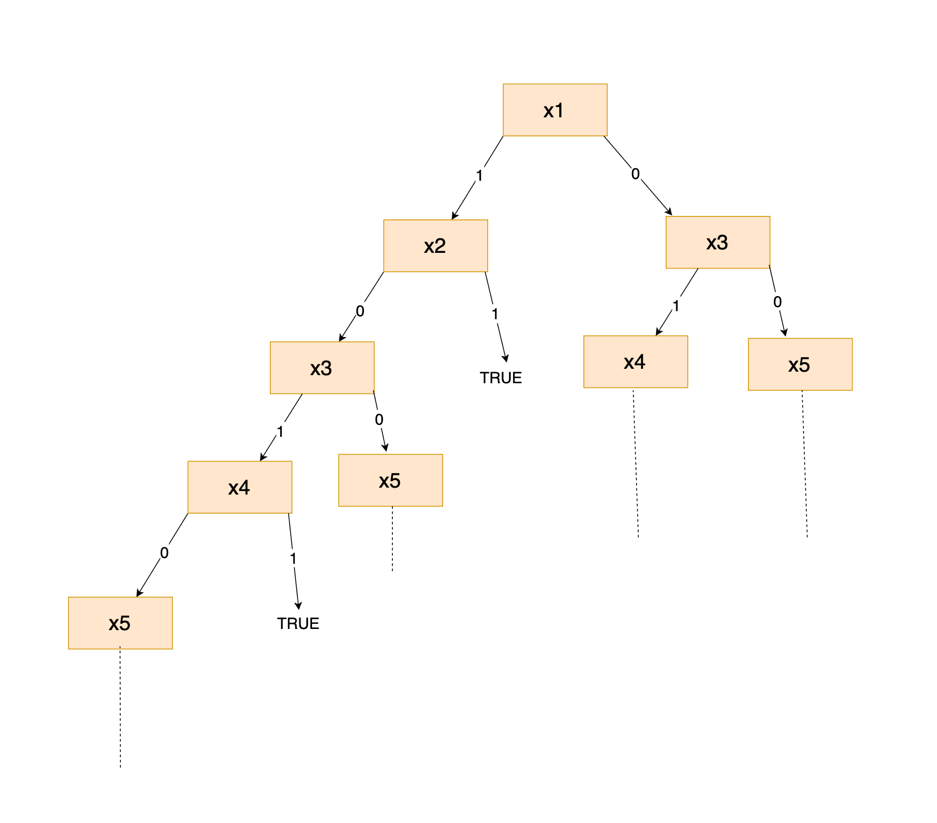


Παρατηρούμε ότι το δέντρο απόφασης έχει μέγιστο μήκος n και προσεγγίζει τον fn με ακρίβεια 1.

Μέρος 2:

Χωρίς βλάβη της γενικότητας θεωρώ: *fn*​=(*x*1​∧*x*2​)∨(*x*3​∧*x*4​)∨…∨(*xn*−1​∧*xn*​)

Για το δέντρο απόφασης έχουμε:



Β)

1.

Υπολογισμός με gini:

gini(Root) = 1 - (5/10)^2 - (5/10)^2 = 0.5

Για το x1:

gini(x1 = 1) = 1 – (3/5)2 – (2/5)2 = 12/25

gini(x1 = 0) = 1 – (2/5)2 – (3/5)2 = 12/25

gini(x1) = (5/10) gini(x1 = 1) + (5/10) gini (x1 = 0) = 12/25

𝗂𝗀(x1) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x1) = 0.02

Για το x2:

gini(x2 = 1) = 1 – (2/3)2 – (1/3)2 = 4/9

gini(x2 = 0) = 1 – (3/7)2 – (4/7)2 = 24/49

gini(x2) = (3/10) gini(x2 = 1) + (7/10) gini(x2 = 0) = 10/21

𝗂𝗀(x2) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x2) = 0.0238

Για το x3:

gini(x3 = 1) = 1 – (3/6)2 – (3/6)2 = 0.5

gini(x3 = 0) = 1 – (2/4)2 – (2/4)2 = 0.5

gini(x3) = (6/10) gini(x3 = 1) + (4/10) gini (x3 = 0) = 𝗂𝗀(x3) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x3) = 0

Για το x4:

gini(x4 = 1) = 1 – (2/3)2 – (1/3)2 = 4/9

gini(x4 = 0) = 1 – (3/7)2 – (4/7)2 = 24/49

gini(x4) = (3/10) gini(x4 = 1) + (7/10) gini (x4 = 0) = 10/21

𝗂𝗀(x4) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x4) = 0.0238

ig(x2) = ig(x4) > ig(x1) > ig(x3)

To x2 έχει το υψηλότερο ig οπότε το χρησιμοποιούμε ως root.

Συνεχίζουμε την ίδια διαδικασία για το αριστερό παιδί της ρίζας (για x2=0):

gini(Root) = 1 - (3/7)^2 - (4/7)^2 = 0.489

Για το x1:

gini(x1 = 1) = 1 – (3/5)^2 – (2/5)^2 = 12/25

gini(x1 = 0) = 1 – 0 – (2/2)^2 = 0

gini(x1) = (5/7) gini(x1 = 1) + (2/7) gini (x1 = 0) = (12/25 )\* (5/7)

𝗂𝗀(x1) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x1) = 0.1461

Για το x3:

gini(x3 = 1) = 1 – (3/5)^2 – (2/5)^2 = 0.48

gini(x3 = 0) = 1 – (2/2)^2 – (0/2)^2 = 0

gini(x3) = (5/7) gini(x3 = 1) + (2/7) gini (x3 = 0) = (12/25 )\* (5/7)

𝗂𝗀(x3) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x3) = 0.1461

Για το x4:

gini(x4 = 1) = 1 – (1/2)^2 – (1/2)^2 = 0.5

gini(x4 = 0) = 1 – (2/5)^2 – (3/5)^2 = 0.48

gini(x4) = (2/7) gini(x4 = 1) + (5/7) gini (x4 = 0) = 0.4857

𝗂𝗀(x4) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x4) = 0.003

ig(x1) = ig(x3) > ig(x4)

To x1 έχει το υψηλότερο ig οπότε το χρησιμοποιούμε ως root (κάτω από τον x2 για x2=0).

Συνεχίζουμε την ίδια διαδικασία για το αριστερό παιδί της ρίζας (για x2=1):

gini(Root) = 1 - (2/3)^2 - (1/3)^2 = 0.444

Για το x1:

gini(x1 = 1) = 1 - 0 = 1

gini(x1 = 0) = 1 -(2/3)^2 - (1/3)^2 = 0.444

gini(x1) = 0\* gini(x1 = 1) + (3/3) gini (x1 = 0) = 0.444

𝗂𝗀(x1) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x1) = 0

Για το x3:

gini(x3 = 1) = 1 – (1/1)^2 = 0

gini(x3 = 0) = 1 – (2/2)^2 = 0

gini(x3) = (1/3) gini(x3 = 1) + (2/3) gini (x3 = 0) = 0

𝗂𝗀(x3) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x3) = gini(Root)

Δεν χρειάζεται να προχωρήσουμε άλλο καθώς ig(x3) = gini(Root).

Συνεχίζουμε την ίδια διαδικασία για το αριστερό παιδί του (για x1=1):

gini(Root) = 1 - (3/5)^2 - (2/5)^2 = 0.48

Για το x3:

gini(x3 = 1) = 1 – (3/4)^2 – (1/4)^2 = 0.375

gini(x3 = 0) = 1 – (1/1)^2 = 0

gini(x3) = (4/5) gini(x3 = 1) + (1/5) gini (x3 = 0) = 0.3

𝗂𝗀(x3) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x3) = 0.16

Για το x4:

gini(x4 = 1) = 1 – (1/2)^2 – (1/2)^2 = 0.5

gini(x4 = 0) = 1 – (2/3)^2 – (1/3)^2 = 0.4444

gini(x4) = (2/5) gini(x4 = 1) + (3/5) gini (x4 = 0) = 0.4664

𝗂𝗀(x4) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x4) = 0.0136

ig(x3) > ig(x4)

To x3 έχει το υψηλότερο ig οπότε το χρησιμοποιούμε ως root (κάτω από τον x1 για x2=1).

Συνεχίζουμε την ίδια διαδικασία για το δεξί παιδί του x1(για x1=0):

gini(Root) = 1 - (2/2)^2 = 0

Αφού gini(root) = 0 δε χρειάζεται να συνεχίσουμε τους υπολογισμούς. Για x1=0 η απάντηση είναι -1.

Συνεχίζουμε την ίδια διαδικασία για το δεξί παιδί του x3 (για x3=0):

gini(Root) = 1 – (1/1)^2 = 0

Αφού gini(root) = 0 δε χρειάζεται να συνεχίσουμε τους υπολογισμούς. Για x3=0 η απάντηση είναι -1.

Συνεχίζουμε την ίδια διαδικασία για το αριστερό παιδί του x3 (για x3=1):

gini(Root) = 1 – (3/4)^2 – (1/4)^2 = 0.375

Για το x4:

gini(x4 = 1) = 1 – (1/2)^2 – (1/2)^2 = 0.5

gini(x4 = 0) = 1 – (2/2)^2 = 0

gini(x4) = (2/4) gini(x4 = 1) + (2/4) gini (x4 = 0) = 0.25

𝗂𝗀(x4) = 𝗀𝗂𝗇𝗂(𝖱𝗈𝗈𝗍) − 𝗀𝗂𝗇𝗂(x4) = 0.125

Θεωρούμε σαν root το x4.

Συνεχίζουμε την ίδια διαδικασία για το δεξί παιδί του x4 (για x4=0):

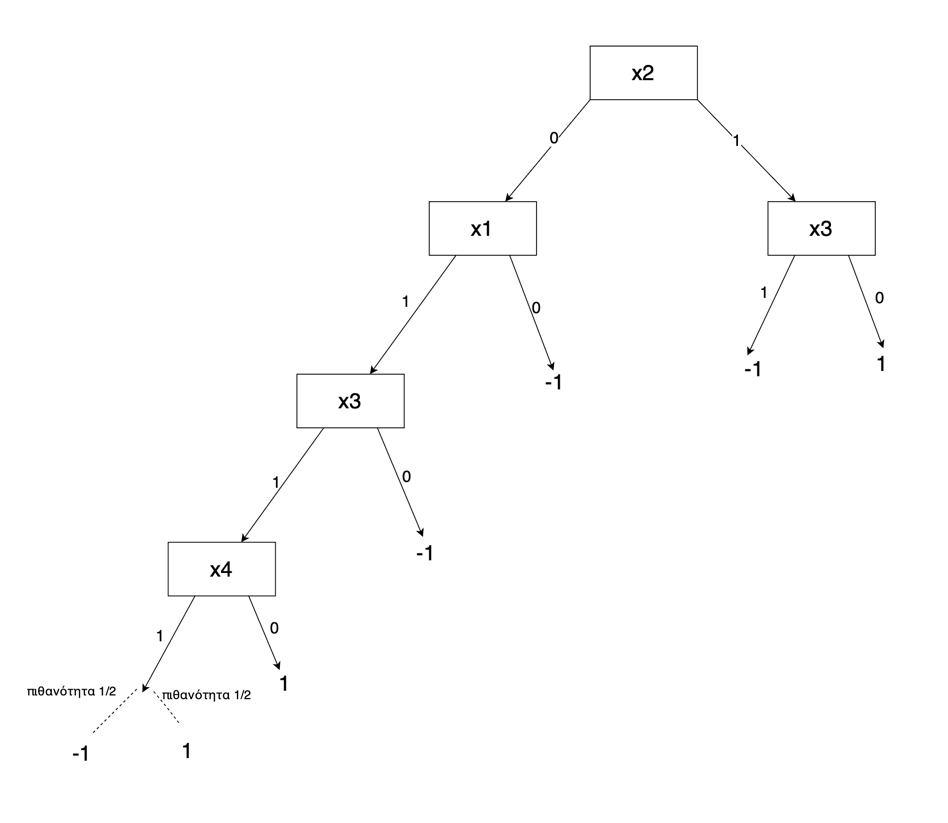
gini(Root) = 1 – (1/1)^2 = 0

Αφού gini(root) = 0 δε χρειάζεται να συνεχίσουμε τους υπολογισμούς. Για x4=0 η απάντηση είναι 1.

Συνεχίζουμε την ίδια διαδικασία για το δεξί παιδί του x4 (για x4=1):

Σε αυτή την περίπτωση, δηλαδή για x1=1, x2=0, x3=1, x4=1 έχουμε δύο πιθανά output(είτε 1 είτε -1) με ίδια πιθανότητα.

Οπότε έχουμε τελικό σχήμα:



2.

Υπολογισμός με εντροπία:

Ε(Root) = -(5/10) log2(5/10) - (5/10) log2(5/10) = 1

x1

E(x1 = 1) = -(3/5) log2(3/5) – (2/5) log2(2/5) = 0.971

E(x1 = 0) = -(2/5) log2(2/5) – (3/5) log2(3/5) = 0.971

E(x1) = (5/10) E(x1 = 1) + (5/10) E(x1 = 0) = 0.971

𝗂𝗀(x1) = E(𝖱𝗈𝗈𝗍) − E(x1) = 0.029

x2

E(x2 = 1) = -(2/3) log2(2/3) – (1/3) log2(1/3) = 0.918

E(x2 = 0) = -(3/7) log2(3/7) – (4/7) log2(4/7) = 0.985

E(x2) = (3/10) E(x1 = 1) + (7/10) E(x1 = 0) = 0.9649

𝗂𝗀(x2) = E(𝖱𝗈𝗈𝗍) − E(x2) = 0.0351

x3

E(x3 = 1) = -(3/6) log2(3/6) – (3/6) log2(3/6) = 1

E(x3 = 0) = -(2/4) log2(2/4) – (2/4) log2(2/4) = 1

E(x3) = (6/10) E(x3 = 1) + (4/10) E(x3 = 0) = 1

𝗂𝗀(x3) = E(𝖱𝗈𝗈𝗍) − E(x3) = 0

x4

E(x4 = 1) = -(2/3) log2(2/3) – (1/3) log2(1/3) = 0.918

E(x4 = 0) = -(3/7) log2(3/7) – (4/7) log2(4/7) = 0.985

E(x4) = (3/10) E(x4 = 1) + (7/10) E(x4 = 0) = 0.9649

𝗂𝗀(x4) = E(𝖱𝗈𝗈𝗍) − E(x4) = 0.0351

x2 = 0

Ε(Root) = -(3/7) log2(3/7) - (4/7) log2(4/7) = 0.985

x1

E(x1 = 1) = -(3/5) log2(3/5) – (2/5) log2(2/5) = 0.971

E(x1 = 0) = – (2/2) log2(2/2) = 0

E(x1) = (5/7) E(x1 = 1) + (2/7) E(x1 = 0) = 0.694

𝗂𝗀(x1) = E(𝖱𝗈𝗈𝗍) − E(x1) = 0.277

x3

E(x3 = 1) = -(3/5) log2(3/5) – (2/5) log2(2/5) = 0.971

E(x3 = 0) = – (2/2) log2(2/2) = 0

E(x3) = (5/7) E(x3 = 1) + (2/7) E(x3 = 0) = 0.694

𝗂𝗀(x3) = E(𝖱𝗈𝗈𝗍) − E(x3) = 0.277

x4

E(x4 = 1) = -(1/2) log2(1/2) – (1/2) log2(1/2) = 1

E(x4 = 0) = -(2/5) log2(2/5) – (3/5) log2(3/5) = 0.971

E(x4) = (2/7) E(x4 = 1) + (5/7) E(x4 = 0) = 0.979

𝗂𝗀(x4) = E(𝖱𝗈𝗈𝗍) − E(x4) = 0.02

x2 = 0 x1 = 0

Ε(Root) = - (2/2) log2(2/2) = 0

Άρα η απάντηση είναι γνωστή πλεόν και ίση με -1.

x2 = 0 x1 = 1

Ε(Root) = -(3/5) log2(3/5) - (2/5) log2(2/5) = 0.971

x3

E(x3 = 1) = -(3/4) log2(3/4) – (1/4) log2(1/4) = 0.811

E(x3 = 0) = 0

E(x3) = (4/5) E(x3 = 1) = 0.6488

𝗂𝗀(x3) = E(𝖱𝗈𝗈𝗍) − E(x3) = 0.162

x4

E(x4 = 1) = -(1/2) log2(1/2) – (1/2) log2(1/2) = 1

E(x4 = 0) = -(2/3) log2(2/3) – (1/3) log2(1/3) = 0.918

E(x4) = (2/5) E(x4 = 1) + (3/5) E(x4 = 0) = 0.22

𝗂𝗀(x4) = E(𝖱𝗈𝗈𝗍) − E(x4) = 0.751

Άρα επιλέγω x4 και τερματίζουμε αφού μετά απέμεινε το x3

x2 = 1

Ε(Root) = -(2/3) log2(2/3) - (1/3) log2(1/3) = 0.918

x1

E(x1 = 1) = 0

E(x1 = 0) = – (2/3) log2(2/3) – (1/3) log2(1/3) = 0.918

E(x1) = (3/3) E(x1 = 0) = 0.918

𝗂𝗀(x1) = E(𝖱𝗈𝗈𝗍) − E(x1) = 0

x3

E(x3 = 1) = -(1/1) log2(1/1) = 0

E(x3 = 0) = – (2/2) log2(2/2) = 0

E(x3) = 0

𝗂𝗀(x3) = E(𝖱𝗈𝗈𝗍) − E(x3) = E(Root)

Άρα επόμενο χαρακτηριστικό το x3 το οποίο μας οδηγεί σε απάντηση και σταματάμε.

Οπότε έχουμε τελικό σχήμα:

