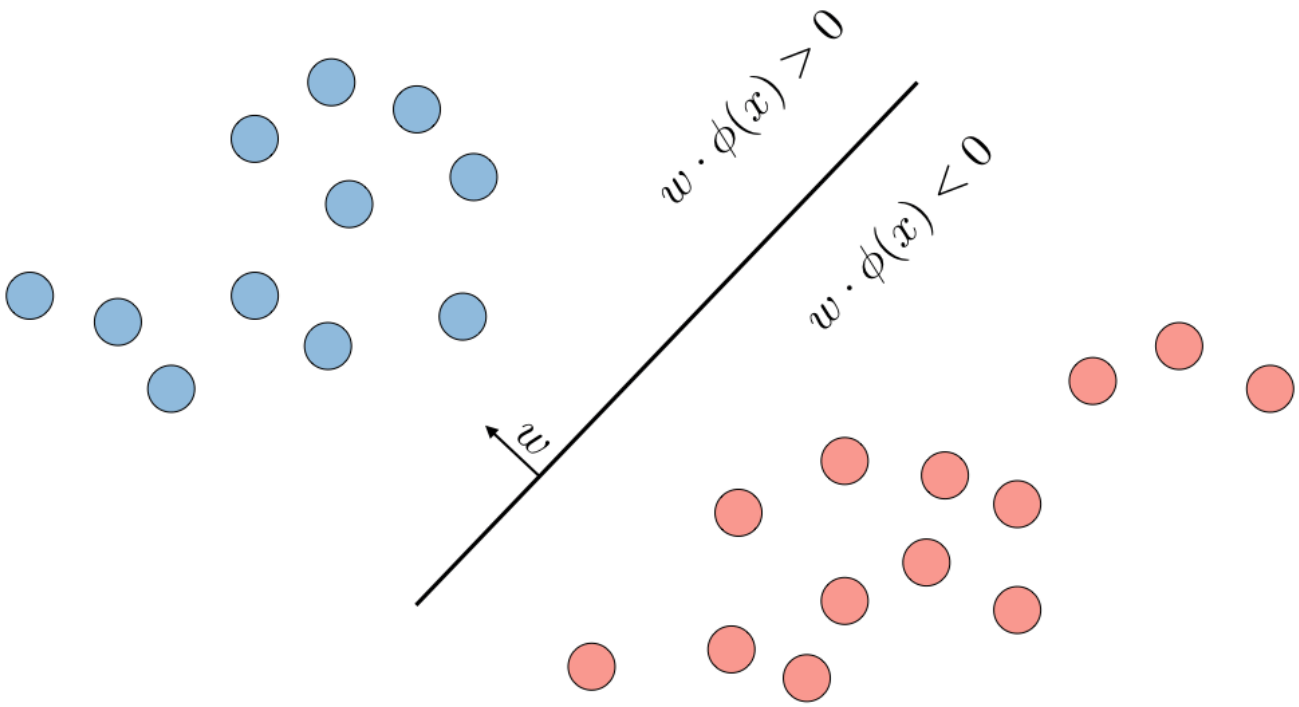
Αναγνώριση Προτύπων

2021-2022

Εργασία 5η

Γραμμικοί Ταξινομητές

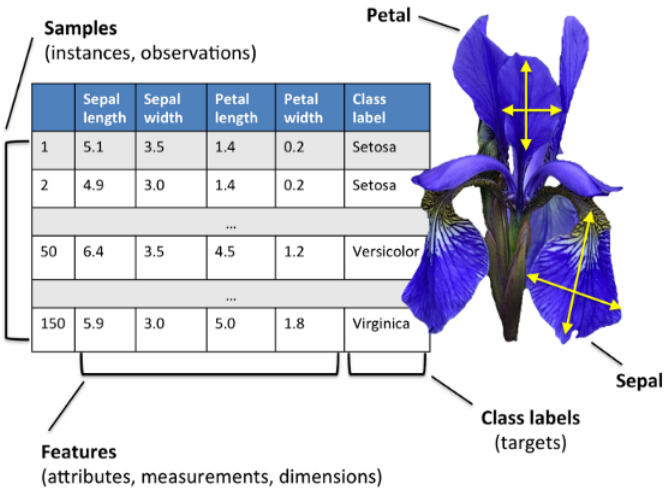


Καρυπίδης Ευστάθιος 57556

13/12/21, Ξάνθη

Για την υλοποίηση των ερωτημάτων της εργασίας έγινε χρήση γλώσσας python και των βιβλιοθηκών numpy,scipy και matplotlib.

Στην εργασία αυτή θα μελετηθούν γραμμικοί ταξινομητές στο IRIS data set) το οποίο περιέχει μετρήσεις της μορφής: (μήκος σέπαλου, πλάτος σέπαλου, μήκος πετάλου, πλάτος πετάλου) σε cm για 150 φυτά iris (είδος κρίνου, αγριόκρινο). Από αυτά τα 150 φυτά, τα 50 είναι Iris Setosa (ω1), τα 50 είναι Iris Versicolour (ω2) και τα υπόλοιπα 50 είναι Iris Virginica (ω3). Γνωρίζουμε ότι μόνο η μία (Iris Setosa) από τις άλλες δυο κλάσεις είναι γραμμικά διαχωρίσιμη.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

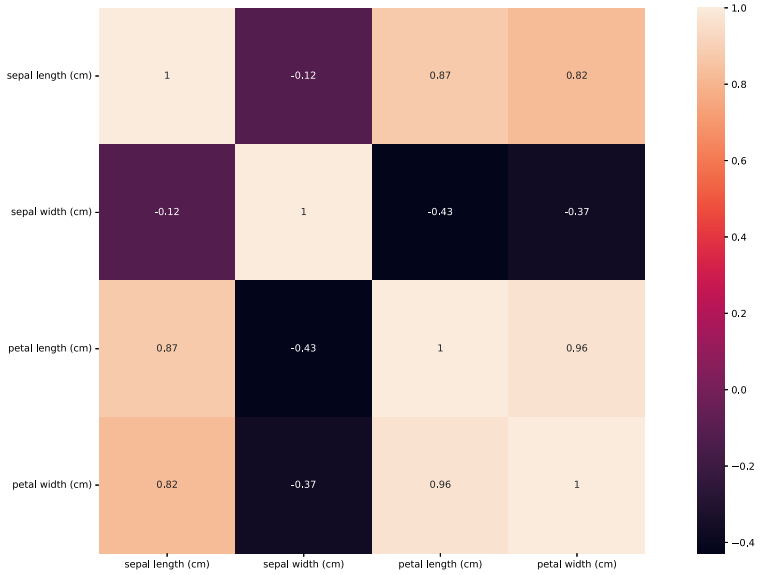
Πριν γίνει όμως η μελέτη των ταξινομητών κρίνεται χρήσιμη η μελέτη και η ανάλυση των δεδομένων του προβλήματος. Ειδικότερα το συγκεκριμένο dataset βρίσκεται έτοιμο στη βιβλιοθήκη sklearn. Συνεπώς στη python μπορούμε να το φορτώσουμε εισάγοντας τη βιβλιοθήκη `from sklearn.datasets import load_iris`. Στο αρχείο **HW5_DataAnalysis.py** μπορεί να βρεθεί ο κώδικας ο οποίος παράγει τα παρακάτω αποτέσματα. Η ανάλυση αυτή θα βοηθήσει και σε ερμηνεία αποτελεσμάτων των ταξινομητών. Αρχικά, μπορούμε να δούμε τη δομή των δεδομένων σε ένα pandas dataframe τυπώνοντας τις πρώτες 5 γραμμές, το σχήμα του dataset καθώς και πόσα δείγματα υπάρχουν από κάθε κλάση.

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) species
0                5.1               3.5                1.4               0.2  setosa
1                4.9               3.0                1.4               0.2  setosa
2                4.7               3.2                1.3               0.2  setosa
3                4.6               3.1                1.5               0.2  setosa
4                5.0               3.6                1.4               0.2  setosa
Dataset Shape:  (150, 5)

Number of species in each class:
setosa        50
versicolor    50
virginica     50
Name: species, dtype: Int64
```

Μερικές επιπλέον πληροφορίες θα ήταν κάποια στατιστικά για κάθε χαρακτηριστικό.

```
More information:
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
count         150.000000        150.000000         150.000000        150.000000
mean            5.843333          3.057333           3.758000          1.199333
std             0.828066          0.435866           1.765298          0.762238
min             4.300000          2.000000           1.000000          0.100000
25%             5.100000          2.800000           1.600000          0.300000
50%             5.800000          3.000000           4.350000          1.300000
75%             6.400000          3.300000           5.100000          1.800000
max             7.900000          4.400000           6.900000          2.500000
```

Στη συνέχεια μπορούμε να κάνουμε κάνουμε και οπτικοποίηση δεδομένων(data visualization) αλλά και διαφόρων χαρακτηριστικών τους. Για παράδειγμα μπορούμε να υπολογίσουμε και να παράγουμε το correlation matrix με μορφή heatmap για τις μεταβλητές.



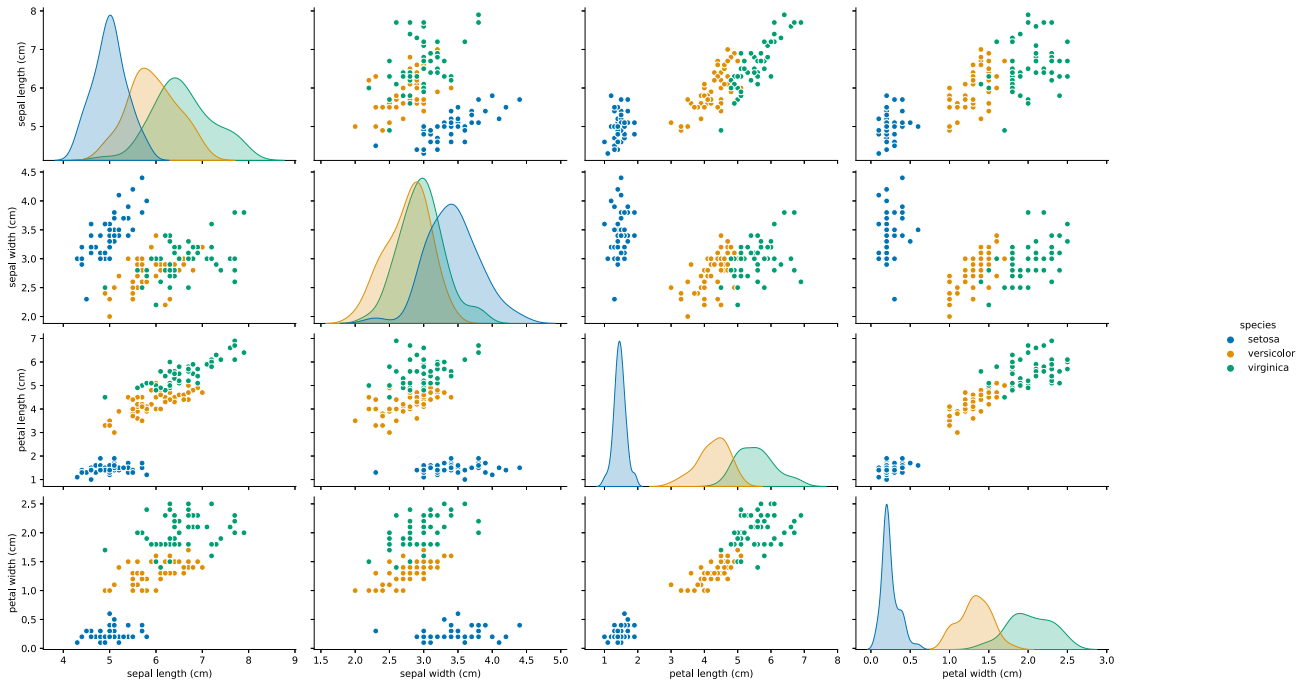
Από εδώ μπορούμε να παρατηρήσουμε ότι τα χαρακτηριστικά petal_length και petal_width έχουν πολύ υψηλή (γραμμική) συσχέτιση ίση με 0.96. Ωστόσο, αρκετά μεγάλη συσχέτιση έχουν και τα χαρακτηριστικά petal_length με το sepal_length ίση με 0.87 και τέλος το sepal_length με το petal_width.

Ακόμη μπορούμε να δούμε το correlation των χαρακτηριστικών με την επιθυμητή έξοδο:

```
Correlation with target:
 sepal length (cm)    0.782561
sepal width (cm)     -0.426658
petal length (cm)     0.949035
petal width (cm)      0.956547
dtype: float64
```

Αυτή η τεχνική είναι ιδιαίτερα χρήσιμη όταν έχουμε πάρα πολλές μεταβλητές καθώς εαν θα θέλαμε να μειώσουμε τη πολυπλοκότητα του προβλήματος. Ειδικότερα, από 2 μεταβλητές/χαρακτηριστικά τα οποία έχουν πολύ υψηλή συσχέτιση μεταξύ τους θα μπορούσαμε να κρατήσουμε μόνο το 1 ενώ ακόμη θα μπορούσαμε να θέσουμε ένα threshold και να κρατήσουμε στο πρόβλημα χαρακτηριστικά τα οποία έχουνε συσχέτιση με την έξοδο τουλάχιστον όσο η τιμή του threshold. By default οι εντολή για correlation χρησιμοποιεί τον τύπο του Pearson.

Τέλος μπορούμε να κάνουμε plot ανα δύο τις μεταβλητές και να μελετήσουμε τη συμπεριφορά τους. Αυτό μπορούμε να το πετύχουμε χρησιμοποιώντας την εντολή `sns.pairplot(df, hue="species", height=2, palette='colorblind')` της βιβλιοθήκης seaborn. Ειδικότερα το pairplot απεικονίζει τη σχέση μεταξύ των μεταβλητών αλλά και με την επιθυμητή έξοδο. Τα αποτελέσματα είναι τα εξής:



Μπορούμε να επιβεβαιώσουμε τις υψηλές τιμές του correlation μεταξύ μεταβλητών που είδαμε και παραπάνω. Επιπλέον μπορούμε να έχουμε μια αρχική εκτίμηση για τις κατανομές των χαρακτηριστικών για κάθε κατηγορία. Τέλος, ένα ακόμη πολύ χρήσιμο συμπέρασμα είναι οτι η κλάσση setosa είναι γραμμικά διαχωρίσιμη από τις άλλες 2 κλάσεις αλλά οι άλλες 2 κλάσεις δέν είναι μεταξύ τους. Ακόμη, από τα plot που βρίσκονται στη διαγώνιο δηλαδή τις κατανομές όσον αφορά ένα χαρακτηριστικό, μπορούμε να συμπεράνουμε ότι τα χαρακτηριστικά sepal_length, sepal_width είναι χαρακτηριστικά που δεν βοηθάν στη (γραμμική) ταξινόμηση καθώς η κατανομές έχουν αλληλοεπικαλύψεις ενώ τα χαρακτ petal_length και petal_width είναι αρκετά για χρήσιμα για ταξινόμηση ιδιαιτέρα για την κατηγορία setosa με τις άλλες δύο αλλά όπως βλέπουμε και στις κλάσεις virginica και versicolor το ποσοστό επικάλυψης είναι αρκετά μικρό. Τέλος, θα μπορούσαμε να εφαρμόσουμε και τεχνικές standardization και normalization στα δεδομένα αλλά στα πλαίσια αυτής της εργασίας δουλεύουμε με τα αρχικά.
Πλεόν εφόσον έχουμε βγάλει κάποια συμπεράσματα από το ίδιο το dataset προχωράμε στο κύριο μέρος της εργασίας.

Άσκηση Α

Batch Perceptron

Στο ερώτημα αυτό ζητείται να βρεθεί ένας διαδυκός γραμμικός ταξινομητής ο οποίος χωρίζει τη κλάση/κατηγορία Iris Setossa(έστω κατηγορία 1) από τις άλλες 2. Συνεπώς εφόσον έχουμε γραμμικό ταξινομητή θα το ανάγουμε σε πρόβλημα δυαδικής ταξινόμησης(binary classification) και οι άλλες 2 κλάσεις/κατηγορίες θα θεωρούνται ως μία δηλαδή ως η κλάση/κατηγορία 2. Ωστόσο στο κώδικα και για να δουλέψει ο αλγόριθμος Perceptron θα χρησιμοποιηθούν οι αριθμοί -1 και 1. Ειδικότερα με βάση το κριτήριο του perceptron γνωρίζουμε ότι εάν έχουμε ένα διάνυσμα βαρών w και ένα διάνυσμα με ένα δειγμα και τα χαρακτηριστικά του(ή πίνακα με πολλά δείγματα) έστω x τότε ο κανόνας για τη ταξινόμηση έχει ως εξής:

- Αν $w^T x > 0$ τότε το δείγμα ανήκει στη κλάση 1 ενώ εαν $w^T x < 0$ τότε το δείγμα ανήκει στη κλάση -1

Ένα τρίκ για να μην έχουμε δύο συνθήκες(που παρουσιάζεται στο βιβλίο του Bishop) είναι να πολλαπλασιάσουμε με τις τιμές στόχου t δηλαδή το 1 και το -1 σε κάθε περίπτωση αντίστοιχα. Έτσι, λοιπόν πετυχαίνουμε να έχουμε τη παρακάτω σχέση:

$$w^T x \cdot t > 0$$

Δηλαδή, γνωρίζουμε ότι το αποτέλεσμα της κατηγοριοποίησης του ταξινομητή είναι σωστό εφόσον πληρείται η παραπάνω σχέση. Έτσι μπορούμε να ορίσουμε τη παρακάτω συνάρτηση κόστους:

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^t \mathbf{y})$$

Όπως, βλέπουμε ορίζεται ως μείον το άθροισμα των γινομένων των βαρών με τα χαρακτηριστικά για τα δείγματα όπου υπάρχει λάθος ταξινόμηση. Στον αλγόριθμο αυτό λοιπόν υπάρχει ως στόχος η ελαχιστοποίηση της συνάρτησης κόστους και για το λόγο αυτό χρησιμοποιείται η τεχνική βελτιστοποίησης gradient descent. Με τη τεχνική αυτή βελτιστοποιούνται τα βάρη του ταξινομητή σε κάθε επανάληψη μέχρι να πετύχουμε την ελαχιστοποίηση του κέρδους, η πιο σωστά όταν το κριτήριο που θέσαμε για τον τερματισμό ικανοποιείται. Τα βάρη ενημερώνονται με την αφαίρεση από τη προηγούμενη τιμή τους ενός παράγοντα που είναι το γινόμενο του learning rate(δηλαδή του πόσο πολύ θέλουμε να μετακινηθούμε προς τη κατεύθυνση όπου η κλίση της συνάρτησης μειώνεται) με τη παράγγωγο της συνάρτησης κόστους. Ετσι, λοιπόν όταν το γινόμενο αυτό πέσει κάτω από μια τιμή θεωρούμε πως ο αλγόριθμος έχει συγκλινει σε ένα ελάχιστο της συνάρτησης. Ο κώδικας της συνάρτησης είναι ο εξής:

```
def batch_perceptron(data, bin_labels, learning_rate, max_epochs, init_weights, theta =
10**(-5)):
    """
    :param data: Matrix of all data, Size=(Num_of_features x Number of samples) -> Here
4x150
    :param bin_labels: Vector of labels of samples, Size=(1 x Number of samples) -> Here
1x150 == (150,)
    :param learning_rate: The step size at each iteration while moving toward a minimum of
a loss function
    :param max_epochs: The maximum number of epochs
    :param init_weights: Initial weights, Size=(Number of features x 1) -> Here 4x1
    :param theta: Stopping tolerance
    :return: weights(final_weights), iteration(last_iteration_number), errors(number of
errors at each iteration), criteria(value of criterion at each iteration)
    """
    iteration = 0
    errors, criteria = [], []
```

```
    pseudo_input = np.ones([1, data.shape[1]])
    data_aug = np.vstack([pseudo_input, data])  # Augmented data matrix
    weights = np.vstack([1, init_weights])  # Augmented weight vector
    while iteration < max_epochs:
        activation = np.multiply(np.dot(weights.transpose(), data_aug), bin_labels)
        wrong_class = np.where(activation < 0, 1, 0)
        errors.append(np.sum(wrong_class))
        error_grad = np.sum(np.where(wrong_class == 1, data_aug * bin_labels, 0),
axis=1).reshape(-1, 1)
        iteration += 1
        criterion = np.sum(np.abs(learning_rate*error_grad))/data_aug.shape[1]
        criteria.append(criterion)
        if criterion > theta:
            weights = weights + learning_rate*error_grad
        else:
            break
        # if errors[iteration] > 0:
        #     error_grad = np.sum(np.where(wrong_class == 1, data_aug*bin_labels, 0),
axis=1).reshape(-1, 1)
        #     weights = weights + learning_rate*error_grad
        #     print(np.sum(abs(learning_rate*error_grad)))
        #     iteration += 1
        # else:
        #     iteration += 1
        #     break
    return weights, iteration, errors, criteria
```

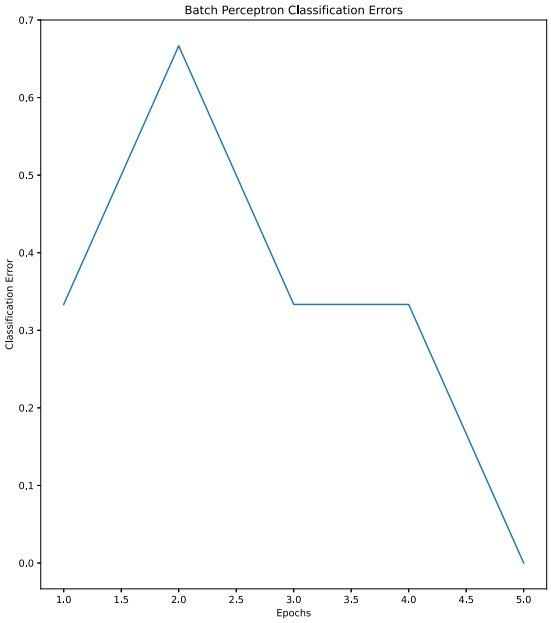
Στη συνέχεια με το παρακάτω κώδικα καλούμε τη συνάρτηση που φτιάξαμε αφού πρώτα φορτώσουμε τα δεδομένα. Αξίζει να σημειωθεί ότι η στη συνάρτηση έχω σε docstring σχόλια τις εισόδους της καθώς επίσης μέσα στη while βρίσκεται σε σχόλια και κριτήριο τερματισμού με βάση των αριθμό των λαθών. Ωστόσο, χρησιμοποιείται το κριτήριο που είδαμε και στη θεωρία. Ακόμη, στο αρχείο python υπάρχει αντίστοιχα και κώδικας για την οπτικοποίηση των αποτελεσμάτων που δεν παρουσιάζεται εδώ για εξοικονόμηση χώρου.

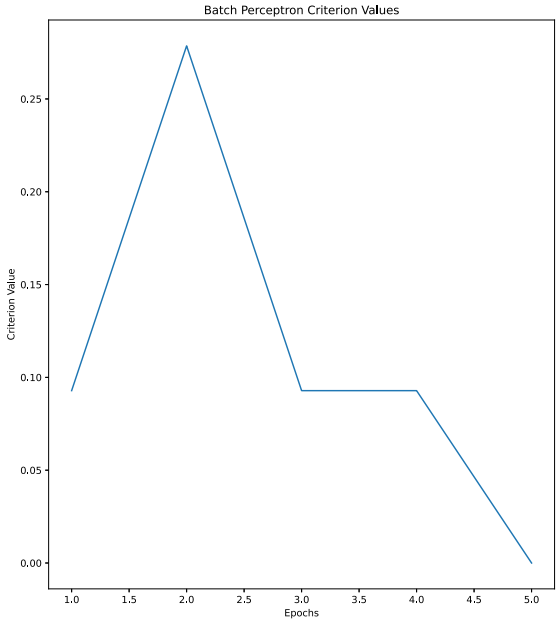
```
ris_dataset = load_iris()
iris_data = iris_dataset.data.transpose()
iris_labels = iris_dataset.target
binary_labels = np.where(iris_labels == 0, -1, 1)
initial_weights = np.ones([iris_data.shape[0], 1])
best_weights_per, iterations_per, error_list_per, criteria_per =
batch_perceptron(iris_data, binary_labels, 0.025, 1000, initial_weights, 10**(-5))
```

Παρακάτω παρουσιάζονται τα τελικά βάρη καθώς, ο αριθμώς των εποχών(επαναλήψεων) του αλγορίθμου καθώς, ο αριθμός των δειγμάτων που ταξινομούνται λάθος και η τιμή του κριτηρίου σε κάθε εποχή.

```
Final weights using Batch Perceptron are:
 [[-0.25  ]
 [-2.1175]
 [-4.675 ]
 [ 7.7825]
 [ 4.2675]]
Batch Perceptron finished terminated after  5  epochs with error  0.0
Values of Criterion are:  [0.09285, 0.2786, 0.09285, 0.09285, 0.0]
```

Στη συνέχεια μπορούμε να κάνουμε τα αντίστοιχα plot για τα errors ανα εποχή και τις τιμές του κριτηρίου.





Όπως παρατηρούμε ο αλγόριθμος συγκλίνει με μηδενικά σφάλματα και μηδέν τιμή κριτηρίου. Αξίζει να σημειωθεί ότι εφόσον χρησιμοποιούμε batch gradient descent και ο αριθμός των λαθών μπορεί να είναι μεγάλος η τιμή του error_grad στο κώδικα μπορεί να είναι πολύ μεγάλη. Συνεπώς, χρησιμοποίησα ένα σχετικά μικρό gradient descent. Διαφορετικά, θα μπορούσα να διαιρέσω με τον αριθμό των λαθών(δηλαδή ουσιαστικα να πάρω μέσες τιμές). Αυτό, το κάνουμε επειδή δε θέλουμε τεράστιες μεταβολές στα βάρη καθώς μπορεί να καταλήξει ο αλγόριθμος σε αστάθεια.

Batch Relaxation with Margin

Ο αλγόριθμος του batch relaxation with σε σχέση με το προηγούμενο διαφέρει στη συνάρτηση κόστους που χρησιμποιείται. Στη συγκεκριμένη περίπτωση χρησιμοποιείται η συνάρτηση κόσοτυς κανονικοποιείται με το μέτρο των διανυσμάτων των χαρακτηριστικών των οποίων έχουν ταξινομηθεί λάθος. Η συνάρτηση κόστους είναι:

$$J_r(\mathbf{a}) = \frac{1}{2} \sum_{\mathbf{y} \in \mathcal{Y}} \frac{(\mathbf{a}^t \mathbf{y} - b)^2}{\|\mathbf{y}\|^2}$$

Ακόμη, ως κριτήριο τερματισμού στο αλγόριθμου είναι η επίτευξη μηδενικού σφάλματος. Επίσης, τώρα η συνθήκη για να θεωρείται ένα δείγμα σωστά ταξινομημένο περιγράφεται ως $w^T x - b > 0 \Rightarrow w^T x > b$. Με τη προσθήκη αυτής της θετικής παραμέτρου b δημιουργείται ένα περιθώριο γύρω απο τη περιοχή της λύσης με αποτέλεσμα τα σημεία που βρίσκοται εντός αυτής της περιοχής να θεωρουνται αμφίβολαώς προς τη κατηγορία την οποία ανήκουν και ο αλγόριθμος δεν προσπαθεί να ταξινομήσει σωστά αυτά τα στοιχεία. Κάτι τέτοιο μπορεί να οδηγήσει εύκολα σε πολλές επαναλήψεις μέχρι να συγκλίνει ο αλγόριθμος ειδικά εν δεν επιλεχθούν σωστά οι παράμετροι. Τέλος, σε αυτό τον αλγόριθμο βολεύει και βοηθάει πολύ η χρήση ενός decaying learning rate. Ωστόσο μετά απο δοκιμές, βρήκα τιμή του learning rate που παρόλο που είναι σταθερή και επιτρέπει τον αλγόριθμο να συγκλίνει. Σε σχόλια υπάρχει όμως και υλοποίηση για decaying learning rate η οποία ακολουθεί την ακόλουθη σχέση:

$$\text{learing rate(t)} = \frac{\text{learing rate}(0)}{1 + a \cdot t}$$

όπου tο αριθμός επανάληψης και $a \in [0, 1]$ μία σταθερά όπου δηλώνει πόσο απότωμα θέλουμε να μειώνεται το learning

Ο κώδικας για τη συνάρτηση είναι:

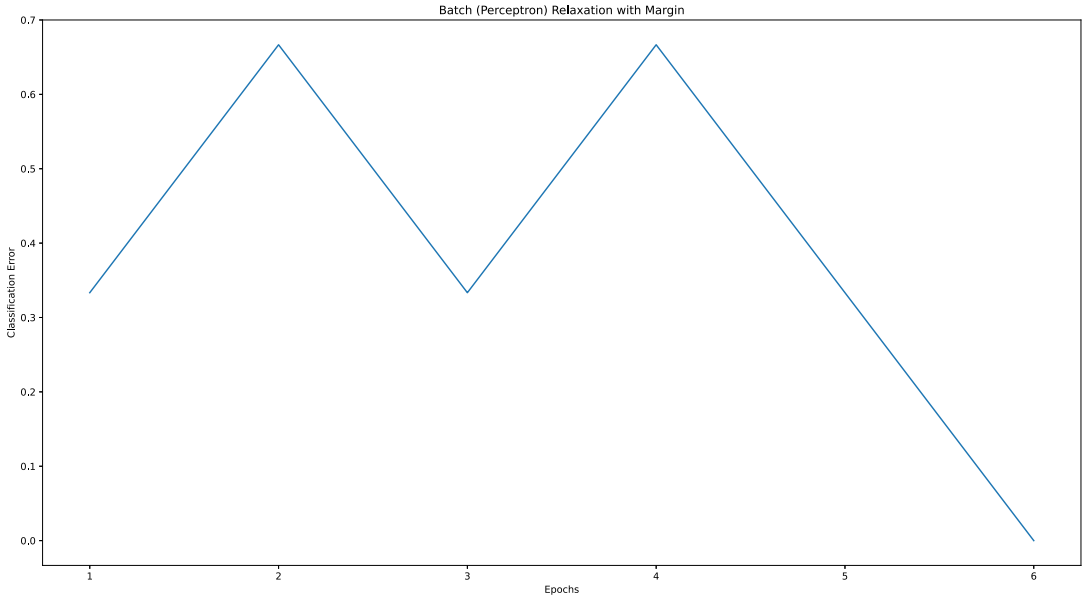
```
def batch_relaxation_with_margin(data, bin_labels, learning_rate, max_epochs, init_weights,
b):
    """
    :param data: Matrix of all data, Size=(Num_of_features x Number of samples) -> Here
4x150
    :param bin_labels: Vector of labels of samples, Size=(1 x Number of samples) -> Here
1x150 == (150,)
    :param learning_rate: The step size at each iteration while moving toward a minimum of
a loss function
    :param max_epochs: The maximum number of epochs
    :param init_weights: Initial weights, Size=(Number of features x 1) -> Here 4x1
    :param b: Margin
    :return: weights(final_weights), iteration(last_iteration_number), errors(number of
errors at each iteration)
    """
    iteration = 0
    errors = []
    pseudo_input = np.ones([1, data.shape[1]])
    data_aug = np.vstack([pseudo_input, data])
    weights = np.vstack([1, init_weights])
    while iteration < max_epochs:
        activation = np.multiply(np.dot(weights.transpose(), data_aug), bin_labels)
        wrong_class = np.where(activation < b, 1, 0)
        error_count = np.sum(wrong_class)
        errors.append(error_count)
        iteration += 1
        if error_count > 0:
            numerator = (b - np.dot(weights.transpose(), data_aug)).reshape(-1, 1)
            denominator = np.sum(np.where(wrong_class == 1, data_aug, 0)**2,
axis=0).reshape(-1, 1)
            wrong_class_idx = np.where(wrong_class == 1, True, False)
            grad =
(numerator[wrong_class_idx.transpose()]/denominator[wrong_class_idx.transpose()]).reshape(-
1, 1)
            error_grad = np.dot(np.ones([data_aug.shape[0], 1]), grad.transpose())
            temp = np.multiply(error_grad, data_aug[:, wrong_class_idx[0]])
            update = np.sum(temp, axis=1).reshape(-1, 1)
            # lr = learning_rate * (1 / (1 + 0.5 * iteration))
            # print(lr)
            weights = weights + learning_rate*update
            # weights = weights + lr*update
        else:
            break
    return weights, iteration, errors
```

Στη συνέχεια με τη παρακάτω εντολή καλούμε τη συνάρτηση και τυπώνουμε τα αποτελέσματα:

```
best_weights_per_relax, iterations_per_relax, error_list_per_relax =
batch_relaxation_with_margin(iris_data, binary_labels, 0.04, 1000, initial_weights, 1.5)
print("Final weights using Batch (Perceptron) Relaxation are : \n", best_weights_per_relax)
print("Batch (Perceptron) Relaxation with Margin finished terminated after ",
iterations_per_relax, " epochs with error ", error_list_per_relax[-1]/iris_data.shape[1])
```

Τα αποτελέσματα είναι τα εξής:

```
Final weights using Batch (Perceptron) Relaxation with Margin are :
 [[ 0.38353753]
 [-1.02787761]
 [-1.60438665]
 [ 3.00709264]
 [ 2.04543631]]
Batch (Perceptron) Relaxation with Margin finished terminated  with Margin after  6  epochs with error  0.0
```



Όπως παρατηρούμε θέσαμε τιμή b = 1.5 δηλαδή έχουμε τη περίπτωση του overelaxation. Μετά από δοκιμές που έγινε ήταν μια βολική τιμή η οποία οδηγούσε πιο εύκολα σε σύγκλιση ειδικά με σταθερή τιμή στο learninng rate. Όπως φαίνεται στη περίπτωση αυτή ο αριθμός των σφαλμάτων γίνεται 0 και ο αλγόριθμος τερματίζεται στην 6 αφού πρωτα όμως υπάρχουν κάποιες διακυμάνσεις στον αριθμό σφαλμάτων. Παρά το γεγονός ότι οι κλάσεις 1 με τις 2και4 είναι γραμμικά διαχωρίσιμες ο συγκεκριμένος αλγόριθμος απαιτεί ιδιαιτερη προσοχή. Η συνάρτηση κόστους εδώ είναι πιο ομαλή αλλά χρειάζεται ικανός αριθμός επαναλήψεων και κατάλληλη τιμή στο learning rate και το b δηλαδή την περιοχή όπου δεν υπάρχει σαφής απάντηση του αλγορίθμου ως προ τη κλάση, έτσι ώστε να έρθει η σύγκλιση. Όπως αναφέρθηκε προηγουμένος μια τεχνική όπως το decaying learning rate θα βοηθούσε στη διαδικασία και τη ταχύτητα της σύγκλισης.

Άσκηση Β

Ταξινομητής με μέθοδο MSE και χρήση Ψευδοαντιστρόφου

Στη περίπτωση του ταξινομητή με μέθοδο MSE και χρήση ψευδοαντιστρόφου, το πρόβλημα ανάγεται σε ένα συνολο ανισώσεων της μορφής $Ya = b$ όπου Y ο επαυξημένος πίνακας δειγμάτων, a ο επαυξημένο πινακας βαρών και b μία θετική σταθερά(συνήθως 1). Αξίζει να σημειωθεί ότι ο επαυξημένος πίνακας βαρών απαιτεί να υπάρχει ενα ενα έξτρα αντίθετο πρόσημο στα δείγματα της μία κλάσης. Αυτό, μπορούμε να το αποφύγουμε βάζοντας τα + και - στον πίνακα b όπου καταλήγει να εκφράζει λόγω τη κλάση του κάθε δείγματος εφόσον είναι +1 και -1. Αυτό το οδηγεί σε απλοποίηση πράξεων και η τελική μορφή είναι μαθηματικά ισοδύναμη με την αρχική. Η συνάρτηση κόστους στη περίπτωση αυτή είναι:

$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathbf{b}\|^2 = \sum_{i=1}^{n} (\mathbf{a}^t \mathbf{y}_i - b_i)^2$$

Βρίσκοντας τη παράγωγο της συνάρτησης κόστους προκύπτει ότι το διάνυσμα των βαρών ισούται μέ:

$$\begin{aligned} \mathbf{a} &= (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t\mathbf{b} \\ &= \mathbf{Y}^\dagger\mathbf{b}, \end{aligned}$$

Η συνάρτηση για υλοποίηση του αλγορίθμου είναι η εξής:

```
def mse_pseudoinverse(data, bin_labels):
    """
    :param data:  Matrix of all data, Size=(Num_of_features x Number of samples) -> Here
4x150
    :param bin_labels: Vector of labels of samples, Size=(1 x Number of samples) -> Here
1x150 == (150,)
    :return: weights(final_weights), errors(number of errors at each iteration)
    """
    pseudo_input = np.ones([1, data.shape[1]])
    data_aug = np.vstack([pseudo_input, data])
    pseudoinverse = np.linalg.pinv(data_aug.transpose())
    # e = 0
    # pseudoinverse_mine = np.dot(np.linalg.inv(np.dot(data_aug,
data_aug.transpose())+e*np.eye(data_aug.shape[0])), data_aug)
    weights = np.dot(pseudoinverse, bin_labels).reshape(-1, 1)
    disc_fun = np.multiply(np.dot(weights.transpose(), data_aug), bin_labels)
    errors = np.sum(disc_fun < 0)
    return weights, errors
```

Όσον αφορά την εύρεσης του ψευδοαντίστροφου υπάρχει έτοιμη συνάρτηση στη βιβλιοθήκη numpy ωστόσο για επιβεβαίωση των αποτελεσμάτων υλοποίησα και χειροκίνητα τον υπολογισμό τον οποίο τον άφησα σε σχόλια. Τέλος, ο αριθμός των λάθος ταξινομημένων δειγμάτων μπορεί να βρεθεί από την εντολή `np.sum(disc_fun < 0)` δηλαδή από το άθροισμα των σημείων όπου η τιμή της συνάρτησης διάκρισης που προκύπτει είναι μικρότερη του 0. Στη συνέχεια καλούμε τη συνάρτηση αφού έχουμε φορτώσει τα δεδομένα όπως στις προηγούμενες περιπτώσεις:

```
weights_pinv, errors_pinv = mse_pseudoinverse(iris_data, binary_labels)
```

Τα αποτελέσματα είναι τα εξής:

```
Weights using MSE with pseudoinverse method are:
 [[-0.76355422]
 [ 0.13205954]
 [ 0.48569574]
 [-0.44931423]
 [-0.11494546]]
0  out of 150  points misclassified using MSE with pseudoinverse method
Classification error sing MSE with pseudoinverse method is:  0.0
```

Ταξινομητής με αλγόριθμο Widrow-Hoff (LMS)

Στη περίπτωση αυτή, ο αλγόριθμος χρησιμοποιεί την ίδια συνάρτηση κόστους δηλαδή:

$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathbf{b}\|^2 = \sum_{i=1}^{n} (\mathbf{a}^t \mathbf{y}_i - b_i)^2$$

Ωστόσο, η διαφορά έγγυται στο γεγονός ότι ο αλγόριθμος προσπαθεί να βρεί τα βέλτιστα βάρη αναδρομικά μέσω βελτιστοποίησης δηλαδή ελαχιστοποίησης της συνάρτησης κόστους. Ειδικότερα εαν υπάρχουν ανάγκες για ελάττωση της μνήμης ο αλγόριθμος μπορεί να τρέξει και σε online έκδοση αλλά θεώρησα πως κάτι τέτοιο δε θα οδηγούσε απαραίτητα σε σύγκλιση, η πιο σωστά η σύγκλιση δε θα οδηγούσε απαραίτητα στα βέλτιστα βάρη. Έτσι λοιπόν υλοποίησα μία batch έκδοση του αλγορίθμου. Τέλος, υπάρχει και ένα rule of thumb για την αρχικοποίηση των βαρών που μπορεί να οδηγήσει σε πιο γρήγορη και εύκολη σύγκλιση. Ο κώδικας για την υλοποίηση του αλγορίθμου είναι:

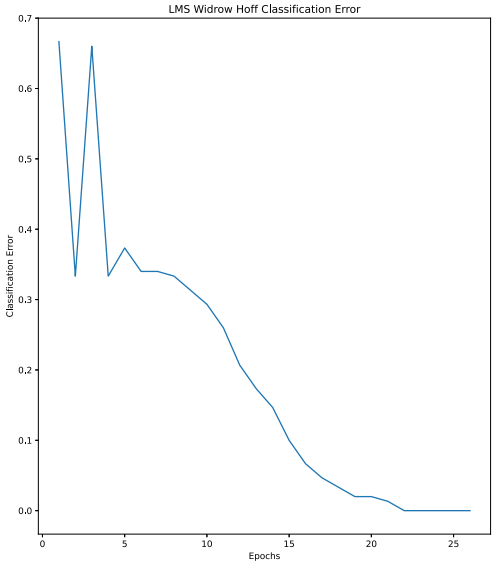
```
def lms_widrow_hoff(data, bin_labels, learning_rate, max_epochs, init_weights, theta):
    """
    :param data:  Matrix of all data, Size=(Num_of_features x Number of samples) -> Here
4x150
    :param bin_labels: Vector of labels of samples, Size=(1 x Number of samples) -> Here
1x150 == (150,)
    :param learning_rate: The step size at each iteration while moving toward a minimum of
a loss function
    :param max_epochs: The maximum number of epochs
    :param init_weights: Initial weights, Size=(Number of features x 1) -> Here 4x1
    :param theta: Stopping tolerance
    :return: weights(final_weights), iteration(last_iteration_number), errors(number of
errors at each iteration)
    """
    iteration, b  = 0, bin_labels
    pseudo_input = np.ones([1, data.shape[1]])
    data_aug = np.vstack([pseudo_input, data])
    weights = np.vstack([1, init_weights])
    # weights = np.sum(data_aug, axis=1).reshape(-1, 1)/data_aug.shape[1]
    criteria, errors = [], []
    while iteration < max_epochs:
        batch_update = learning_rate * (b - np.dot(weights.transpose(), data_aug))*data_aug
        criterion = np.sum(np.abs(np.sum(batch_update, axis=1)))/data_aug.shape[1]
        criteria.append(criterion)
        iteration += 1
        activation = np.multiply(np.dot(weights.transpose(), data_aug), bin_labels)
        errors.append(np.sum(activation < 0))
        if criterion > theta:
            weights = weights + (np.sum(batch_update, axis=1)/150).reshape(-1, 1)
        else:
            break
    return weights, errors, iteration, criteria
```

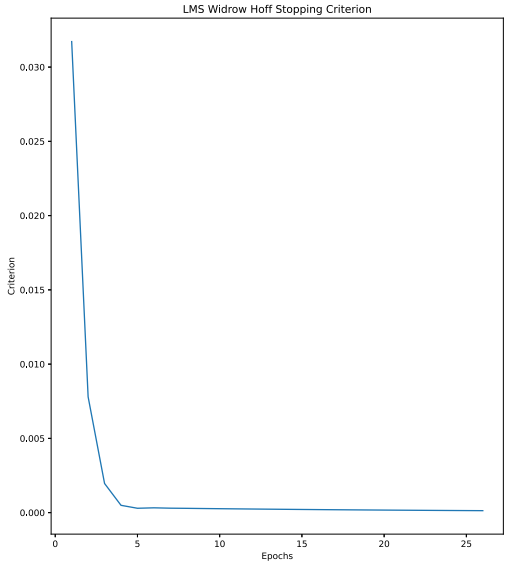
Στο αρχείο του κώδικα υπάρχει αντίστοιχα σε σχόλια και το online μέρος του αλγορίθμου δηλαδή όταν διατρέχει τα δείγματα 1-1. Στη συνέχεια καλούμε τη συνάρτηση έχοντας αρχικά βάρη 1 και στη συνέχεια με βάση το κανόνα στα σχόλια.

```
weights_lms, errors_lms, iterations_lms, criteria_lms = lms_widrow_hoff(iris_data,
binary_labels, 0.02, 100, initial_weights, 0.02)
```

Τα αποτελέσματα είναι τα εξής:

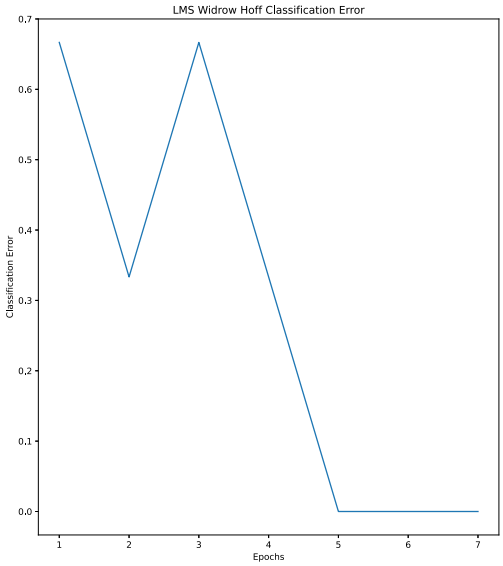
```
Final weights using LMS Widrow-Hoff are:
 [[ 0.80368783]
 [-0.34365865]
 [ 0.48589765]
 [-0.33574852]
 [ 0.498532  ]]
0  out of 150  points misclassified
LMS Widrow-Hoff finished terminated after  26  epochs with error  0.0
```

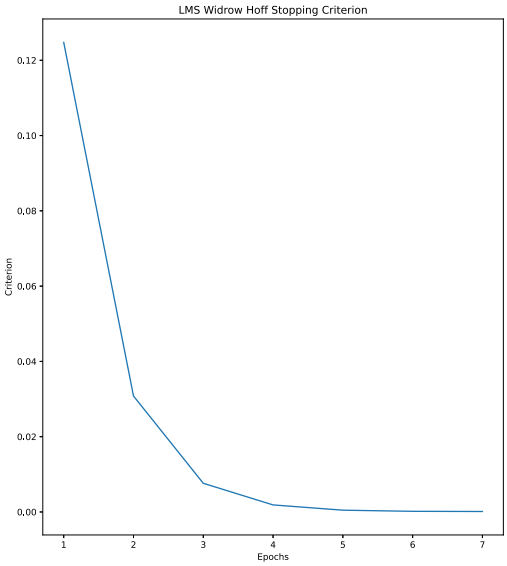




Εαν όμως κάναμε αρχικοποίηση βαρών με το τρόπου που υπάρχει στα σχόλια τότε:

```
Final weights using LMS Widrow-Hoff are:
 [[ 0.03825406]
 [ 0.04411931]
 [ 0.15143056]
 [-0.26217806]
 [-0.12276577]]
0  out of 150  points misclassified
LMS Widrow-Hoff finished terminated after  7  epochs with error  0.0
```





Όπως φαίνεται και στις δυο περιπτώσεις καταλήγει ο αλγόριθμος σε σύγκλιση, αλλά στη 2η περίπτωση όπου ο αλγόριθμος θα αρχικοποιηθεί σε βάση σύμφωνα με το κανόνα `np.sum(data_aug, axis=1).reshape(-1, 1)/data_aug.shape[1]` η σύγκλιση επέρχεται πιο γρήγορα. Ακόμη, αξίζει να σημειωθεί ότι παραόλο που ο αλγόριθμος καταλήγει να μη ταξινομεί λάθος κανένα δείγμα συνεχίζει λόγο της μικρής τιμής στο κριτήριο. Προτιμάω, μικρή τιμή έτσι ώστε στη περίπτωση σύγκλισης σίγουρα να μην γίνονται λάθος ταξινομήσεις. Ο αλγόριθμος αυτός έχει το πλεονέκτημα ότι αποφεύγεται ο υπολογισμός του ψευδοαντίστροφου ο οποίος πολλές φορές μάλιστα δεν μπορεί να υπολογιστεί. Η online έκδοση του αλγορίθμου δηλαδή αυτή όπου χρησιμοποιούσε 1 δείγμα τη φορά ήταν πολύ πιο δύσκολο να συγκλίνει και απαιτούσε το κριτήριο να γίνει της τάξης 10^-8 καθώς παρόλο που το κριτήριο μπορεί να ικανοποιούνταν για ένα δείγμα, τα βάρη αυτά μπορεί να μην ήταν γενικά τα βέλτιστα.

Άσκηση Γ

Ταξινομητής με μέθοδο MSE και χρήση Ψευδοαντιστρόφου

Στο συγκεκριμένο ερώτημα καλείται να υλοποιηθεί ο ταξινομητής με μέθοδο MSE και χρήση ψευδοαντιστρόφου αλλά για τις κλάσεις Iris Versicolour (ω2) από ηελ Iris Virginica (ω3). Όπως γνωρίζουμε αυτές οι κλάσεις δεν είναι γραμμικά διαχωρίσιμες οπότε αναμένουμε σφάλμα. Γίνεται χρήση της συνάρτησης που ορίστηκε πιο πάνω αλλά γίνεται αλλαγή στη χρήση των δεδομένων. Ειδικότερα χρησιμοποιούνται μόνο αυτά των 2 αυτών κλάσεων.

```
ris_dataset = load_iris()
iris_data = iris_dataset.data.transpose()
iris_labels = iris_dataset.target
iris_data_c12 = iris_data[:, 50:]
iris_labels_c12 = iris_labels[50:]
binary_labels_c12 = np.where(iris_labels_c12 == 1, 1, -1)

weights_pinv, errors_pinv = mse_pseudoinverse(iris_data_c12, binary_labels_c12)
```

Τα αποτελέσματα είναι τα εξής:

```
Weights using MSE pseudoinverse method are:
 [[ 1.83727773]
 [ 0.3921192 ]
 [ 0.6151007 ]
 [-0.76852876]
 [-1.3656893 ]]
3  out of 100  points misclassified using MSE pseudoinverse method
Classification error using MSE pseudoinverse method  is:  0.03
```

Όπως παρατηρούμε στη περίπτωση αυτή υπάρχουν 3 σημεία που ταξινομούνται λάθος.

Ταξινομητής με αλγόριθμο Ho-Kashyap

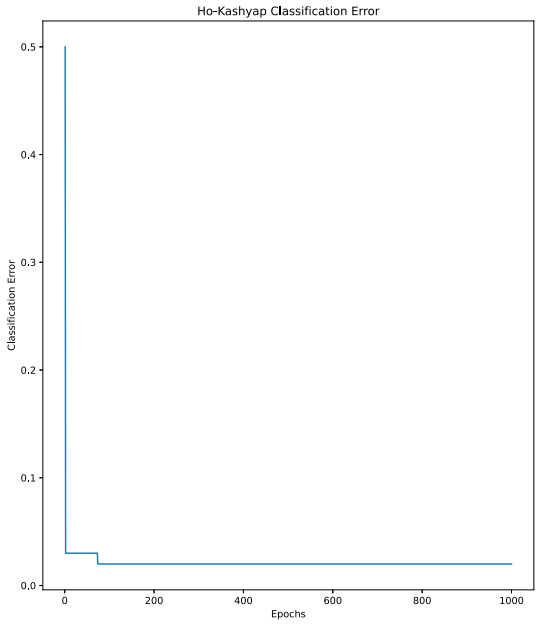
Το διάνυσμα b που χρησιμοποιείται στο προηγούμενο αλγόριθμο τίθεται αυθαίρετα ως είσοδος του αλγορίθμου, δηλαδή δίναμε στην είσοδο το διάνυσμα binary_labels με τα +1 και -1. Όμως αυτό δε το γνωρίζουμε. Αυτό το πρόβλημα καλείται να λύσει ο αλγόριθμος Ho-Kashyap ο οποίος επαναληπτικά προσπαθεί να βρεί το βέλτιστο b. Το βήμα ενημέρωσης του b εξαρτάται απο το σφάλμα για το σύστημα εξισώσεων($e = Ya - b$). Η επαναληπτική διαδικασία συνεχίζεται μέχρι αυτό το σφάλμα να γίνει πολύ μικρό και συγκεκριμένα μικρότερο από ενός threshold b_min. Ο περιορισμός του αλγορίθμου είναι το b να είναι θετικό πράγμα που επιτυγχάνεται με τη χρήση της απόλυτης τιμής κατα τον υπολογισμό του. Εγώ το επιλέγω αρχικά αυθαίρετα 1. Τέλος υπάρχει και σε αυτό τον αλγόριθμο ενα rule of thumb για αρχικοποίηση των βαρών που βοηθάει στη σύγκλιση και είναι `weights = np.dot(np.linalg.pinv(Y), b.transpose())` . Ωστόσο, θα γίνει χρήση τυχαίων αρχικών τιμών στα βάρη.

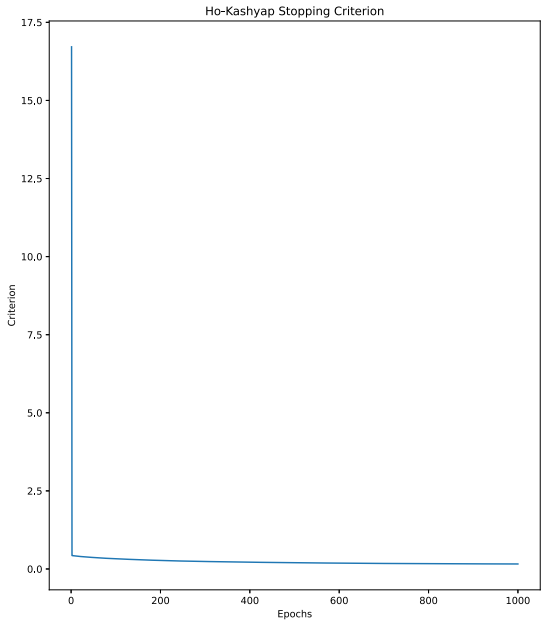
Ο κώδικας για τον αλγόριθμο είναι:

```
def ho_kashyap(data, bin_labels, learning_rate, max_epochs, b_min):
    """
    :param data: Matrix of all data, Size=(Num_of_features x Number of samples) -> Here
4x150
    :param bin_labels: Vector of labels of samples, Size=(1 x Number of samples) -> Here
1x150 == (150,)
    :param learning_rate: The step size at each iteration while moving toward a minimum of
a loss function
    :param max_epochs: The maximum number of epochs
    :param b_min: Stopping criterion
    :return:
    """
    iteration = 0
    pseudo_input = np.ones([1, data.shape[1]])
    data_aug = np.vstack([pseudo_input, data])
    b = np.ones([1, data_aug.shape[1]])
    Y = (data_aug*bin_labels).transpose()
    # weights = np.dot(np.linalg.pinv(Y), b.transpose())
    weights = np.random.rand(5).reshape(-1, 1)
    criteria = []
    errors = []
    while iteration < max_epochs:
        e = np.dot(Y, weights).transpose() - b
        iteration += 1
        activation = np.multiply(np.dot(weights.transpose(), data_aug), bin_labels)
        errors.append(np.sum(activation < 0))
        criterion = np.sum(np.abs(e))
        criteria.append(criterion)
        if criterion > b_min:
            e_plus = 0.5*(e+np.abs(e))
            b += 2 * learning_rate * e_plus
            weights = np.dot(np.linalg.pinv(Y), b.transpose())
        else:
            break
    return weights, errors, iteration, criteria
```

Τα αποτελέσματα είναι τα εξής:

```
Final weights using Ho-Kashyap are:
 [[ 5.17971655]
 [ 0.75525669]
 [ 1.10761597]
 [-1.74350453]
 [-2.63283528]]
2  out of 100  points misclassified
Ho-Kashyap terminated after  1000  epochs with error  0.02
```





Όπως παρατηρούμε τα αποτελέσματα είναι παρόμοια με το ταξινομητή με μέθοδο MSE και χρήση Ψευδοαντιστρόφου. Ειδικότερα παρατηρούμε ότι παραόλο που το κριτήριο συνεχώς πέφτει (μετά απο μερικές εποχές ελάχιστα αλλά συνεχίζει)το σφάλμα του αλγορίθμου δε βελτιώνεται αλλα μένει σταθερό μετά από ένα σημείο. Αυτό ειναι αναμενόμενο καθώς οι κλάσεις δεν είναι γραμμικά διαχωρίσιμες. Ώστόσο γνωρίζουμε ότι οι μέθοδοι ελαχίστων τετραγώων έχουν καλή απόδοση τόσο στις γραμμικά διαχωρίσιμες όσο και στις μη γραμμικά διαχωρίσιμες κατηγορίες κάτι που φαίνεται εδώ καθώς ελάττωσε το σφάλμα κατα μία λάθος ταξινόμηση.

Άσκηση D

Για να γίνει ταξινόμηση πολλών κλάσεων με τον ταξινομητή με μέθοδο MSE και χρήση ψευδοαντιστρόφου χρησιμοποιείται η μέθοδος 1 vs ALL. Ειδικότερα χρησιμοποιείται η συνάρτηση από τη άσκηση Β αλλά με τα κατάλληλα δεδομένα και τιμές στα labels. Ο κώδικας για την υλοποίηση αυτού του ερωτήματος είναι ο εξής:

```
iris_dataset = load_iris()
iris_data = iris_dataset.data.transpose()
iris_labels = iris_dataset.target
unique = np.unique(iris_labels)
weights_pinv = []
errors_pinv = []
binary_labels = []
for i in range(len(unique)):
    binary_labels.append(np.where(iris_labels == i, 1, -1))
    w_pinv, e_pinv = mse_pseudoinverse(iris_data, binary_labels[i])
    weights_pinv.append(w_pinv)
    errors_pinv.append(e_pinv)
    print("Classifying Class ", i, " from  other 2")
    print("Weights using MSE pseudoinverse method are: \n", w_pinv)
    print(e_pinv, " out of", iris_data.shape[1], " points misclassified ")
    print("Classification error is: ", e_pinv / iris_data.shape[1])
    print("-------------------------------------------------")
```

Στην επόμενη σελίδα παρουσιάζονται τα αποτελέσματα του αλγορίθμου.

Τα αποτελέσματα είναι τα εξής:

```
Classifying Class  0  from  other 2
Weights using MSE pseudoinverse method are:
 [[-0.76355422]
 [ 0.13205954]
 [ 0.48569574]
 [-0.44931423]
 [-0.11494546]]
0  out of 150  points misclassified
Classification error is:  0.0
--------------------------------------------------
Classifying Class  1  from  other 2
Weights using MSE pseudoinverse method are:
 [[ 2.15411795]
 [-0.04030737]
 [-0.89123252]
 [ 0.44133841]
 [-0.98861319]]
40  out of 150  points misclassified
Classification error is:  0.26666666666666666
--------------------------------------------------
Classifying Class  2  from  other 2
Weights using MSE pseudoinverse method are:
 [[-2.39056373]
 [-0.09175217]
 [ 0.40553677]
 [ 0.00797582]
 [ 1.10355865]]
11  out of 150  points misclassified
Classification error is:  0.07333333333333333
--------------------------------------------------
```

Όπως φαίνεται ο ταξινομητής που έχει τις περισσότερες λανθασμένες αποφάσεις είναι αυτός που διαχωρίζει τη κλάση 2 από τις υπόλοιπες. Ακόμη, όπως παρατηρούμε και στην κλάση 3 γίνεται λάθος ταξινόμηση 11 από τα 150 σημεία. Γενικά, λοιπόν η χρήση και των τεσσάρων χαρακτηριστικών δεν αρκεί για να είναι τα δεδομένα γραμμικά διαχωρίσιμα.

Άσκηση Ε

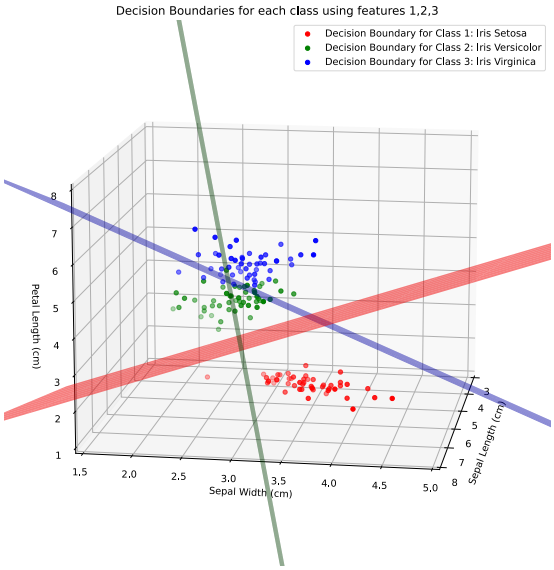
Στο ερώτημα αυτό επαναλαμβάνουμε τη διαδικασία για το ερώτημα D για 3 χαρακτηριστικά. Επαναλαμβάνουμε τον αλγόριθμο 2 φορές μια για τα χαρακτηριστικά 1,2,3 και μια για τα 2,3,4. Ο κώδικας είναι:

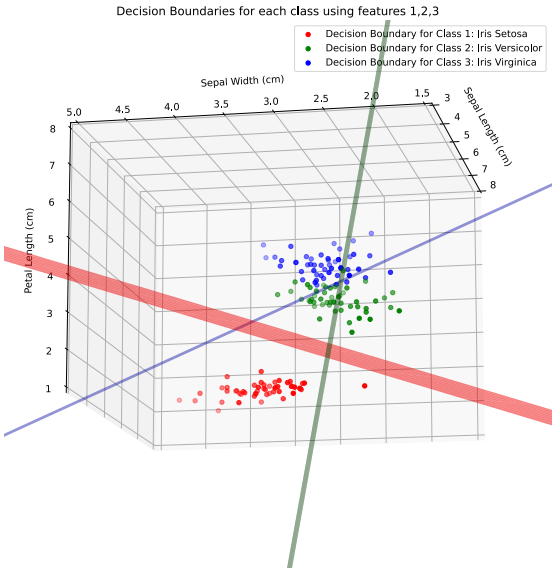
```
iris_dataset = load_iris()
iris_data = iris_dataset.data.transpose()
iris_labels = iris_dataset.target
iris_data_f_123 = iris_data[0:3]
iris_data_f_234 = iris_data[1:]
unique = np.unique(iris_labels)

print("Results using Features 1,2,3: Sepal length, Sepal width, Petal length")
weights_pinv123 = []
errors_pinv123 = []
binary_labels123 = []
for i in range(len(unique)):
    binary_labels123.append(np.where(iris_labels == i, 1, -1))
    w_pinv, e_pinv = mse_pseudoinverse(iris_data_f_123, binary_labels123[i])
    weights_pinv123.append(w_pinv)
    errors_pinv123.append(e_pinv)
    print("Classifying Class ", i, " from  other 2")
    print("Weights using MSE pseudoinverse method are: \n", w_pinv)
    print(e_pinv, " out of", iris_data.shape[1], " points misclassified ")
    print("Classification error is: ", e_pinv / iris_data.shape[1])
    print("-------------------------------------------------")
print("Total errors: ", sum(errors_pinv123), ". Error percentage :",
sum(errors_pinv123)/(3*iris_data.shape[1]))

print("+---"*20+"+")
print("Results using Features 2,3,4: Sepal width, Petal length, Petal width")
weights_pinv234 = []
errors_pinv234 = []
binary_labels234 = []
for i in range(len(unique)):
    binary_labels234.append(np.where(iris_labels == i, 1, -1))
    w_pinv, e_pinv = mse_pseudoinverse(iris_data_f_234, binary_labels234[i])
    weights_pinv234.append(w_pinv)
    errors_pinv234.append(e_pinv)
    print("Classifying Class ", i, " from  other 2")
    print("Weights using MSE pseudoinverse method are: \n", w_pinv)
    print(e_pinv, " out of", iris_data.shape[1], " points misclassified ")
    print("Classification error is: ", e_pinv / iris_data.shape[1])
    print("-------------------------------------------------")
print("Total errors: ", sum(errors_pinv234), ". Error percentage :",
sum(errors_pinv234)/(3*iris_data.shape[1]))
```

Στο αρχείο python στη συνέχεια περιλαμβάνεται και ο κώδικας για οπτικοποίηση των αποτελεσμάτων. Στην επόμενη σελιδα παρουσιάζονται τα αποτελέσματα.

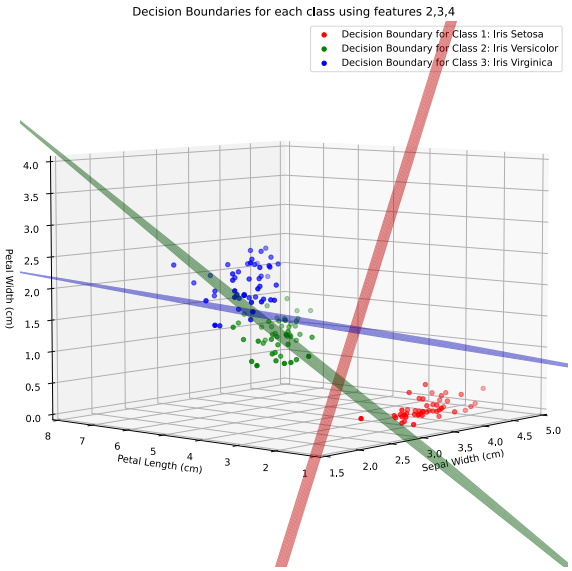
Χαρακτηριστικά 1,2,3	Χαρακτηριστικά 2,3,4
Results using Features 1,2,3: Sepal length, Sepal width, Petal length Classifying Class 0 from other 2 Weights using MSE pseudoinverse method are: [[-0.73593198] [0.15588383] [0.46008261] [-0.50955521]] 0 out of 150 points misclassified Classification error is: 0.0 --- Classifying Class 1 from other 2 Weights using MSE pseudoinverse method are: [[2.391689] [0.16459861] [-1.11152375] [-0.07677707]] 43 out of 150 points misclassified Classification error is: 0.2866666666666667 --- Classifying Class 2 from other 2 Weights using MSE pseudoinverse method are: [[-2.65575702] [-0.32048244] [0.65144114] [0.58633228]] 15 out of 150 points misclassified Classification error is: 0.1 --- Total errors: 58 . Error percentage : 0.1288888888888889	Results using Features 2,3,4: Sepal width, Petal length, Petal width Classifying Class 0 from other 2 Weights using MSE pseudoinverse method are: [[-0.51845205] [0.571645] [-0.35566659] [-0.1884343]] 0 out of 150 points misclassified Classification error is: 0.0 --- Classifying Class 1 from other 2 Weights using MSE pseudoinverse method are: [[2.07930757] [-0.91746605] [0.41275517] [-0.96618284]] 39 out of 150 points misclassified Classification error is: 0.26 --- Classifying Class 2 from other 2 Weights using MSE pseudoinverse method are: [[-2.56085552] [0.34582105] [-0.05708857] [1.15461714]] 9 out of 150 points misclassified Classification error is: 0.06 --- Total errors: 48 . Error percentage : 0.10666666666666667

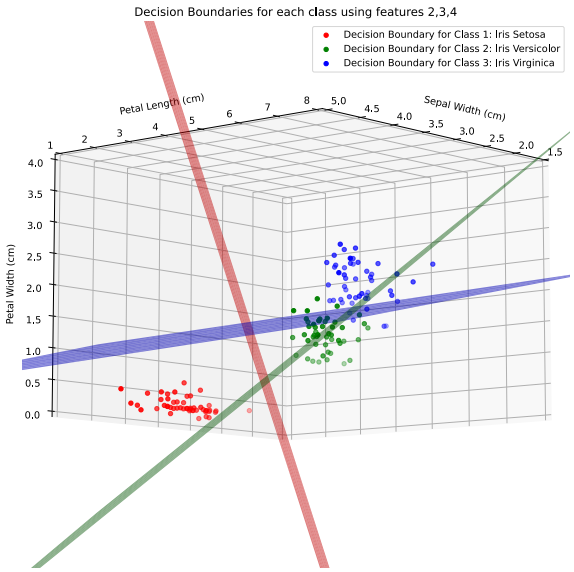
Για τα χαρακτηριστικά 1,2,3 τα υπερεπιπεδα είναι:





Για τα χαρακτηριστικά 2,3,4 τα υπερεπιπεδα είναι:





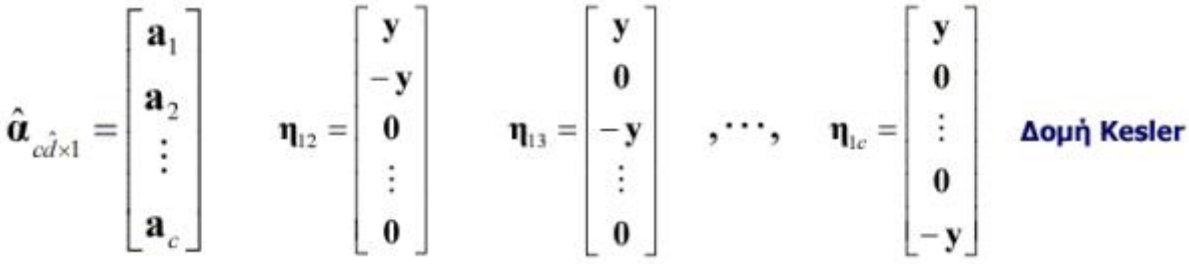
Όπως παρατηρούμε ούτε η χρήση των χαρακτηριστικών 1,2,3 ούτε των 2,3,4 επαρκούν ώστε είναι τα δεδομένα γραμμικά διαχωρίσιμα. Ειδικότερα έχουμε ότι:

- **Με χρήση χαρακτηριστικών 1,2,3:** Βλέπουμε πως η κλάση 1 είναι γραμμικώς διαχωρίσιμη σε σχέση με τις 2 και 3 οι οποίες δεν μπορούν να χωριστούν γραμμικά με αποτέλεσμα να υπάρχει μη μηδενικό σφάλμα ταξινόμησης. Ειδικότερα η κλάση 2 εμφανίζει το χειρότερο σφάλμα ταξινόμησης σχεδόν 30% (28,6%) ενώ η 3η κλάση έχει σφάλμα 10%. Ακόμη και έτσι βλέπουμε ότι έχουμε μια επαρκή ταξινόμηση στις κλάσεις 2 και 3.
- **Με χρήση χαρακτηριστικών 2,3,4:** Βλέπουμε και πάλι πως η κλάση 1 είναι γραμμικώς διαχωρίσιμη σε σχέση με τις 2 και 3 οι οποίες δεν μπορούν να χωριστούν γραμμικά με αποτέλεσμα να υπάρχει μη μηδενικό σφάλμα ταξινόμησης. Στη περίπτωση χρήσης των χαρακτηριστικών 2,3,4 παρατηρούμε ότι οι ταξινομητές καταφέρνουν να έχουν μικρότερο σφάλμα για τις κλάσεις 2 και 3 σε σχέση με τη περίπτωση που χρησιμοποιούσαμε τα χαρακτηριστικά 1,2,3. Αυτό θα μπορούσε να ερμηνευθεί και από το γεγονός οτι το χαρακτηριστικό 4 έχει μεγαλύτερο correlation με την επιθυμητή έξοδο(target) σε σχέση με το χαρακτηριστικό 1.

Λαμβάντας υπόψην τις παραπάνω παρατηρήσεις καθώς και τις επιφάνειες απόφασης συμπεραίνουμε πως καλύτερη ταξινόμηση επιτυγχάνεται με τη χρήση των χαρακτηριστικών 2,3,4.

Άσκηση F

Σε αυτό το ερώτημα ζητούνται οι γραμμικοί ταξινομητές και των 3 κατηγοριών που προκύπτουν με τη δομή Kesler. Με βάση τη θεωρία η δομή Kesler είναι:



Ωστόσο, η υλοποίηση με πίνακες είναι πολύ δύσκολη, και χρησιμοποιήθηκε επαναληπτική διαδικασία με ξεχωριστούς πίνακες που έκανε την ίδια ακριβός δουλειά. Ουσιαστικά η δομή Kesler μετατρέπει το πρόβλημα πολλών κλάσεων σε προβλημα μίας κλάσης. Με βάση τον αλγόριθμο για κάθε εσφαλμένη ταξινόμηση δείγματος γίνεται ενημέρωση 2 βαρων που αντιστοιχούν στο πνακα αυτης της ταξινόμησης ενώ τα υπόλοιπα μένουν ως έχουν. Ο κώδικας είναι:

```
def keslers_construction(data_augmented, class_labels, max_epochs):
    dims, num_of_samples = data_augmented.shape
    unique_classes = np.unique(class_labels)
    num_of_classes = len(unique_classes)
    weights = np.zeros([dims, num_of_classes])
    iteration = 0
    flag = False
    while iteration < max_epochs and not flag:
        iteration += 1
        flag = True
        for i in unique_classes:
            class_i = np.argwhere(class_labels == i)
            class_i = np.squeeze(class_i)
            activ_i = np.dot(weights[:, i].transpose(), data_augmented[:, class_i])
            for j in np.setdiff1d(unique_classes, i*np.ones(1)):
                activ_j = np.dot(weights[:, j], data_augmented[:, class_i])
                diff = activ_i - activ_j
                min_diff_idx = np.argmin(diff, axis=0)
                idx = np.atleast_1d(min_diff_idx)
                if diff[idx] <= 0:
                    idx = class_i[idx[0]]
                    weights[:, i] += data_augmented[:, idx]
                    weights[:, j] -= data_augmented[:, idx]
                    flag = False
                    break
            if not flag:
                break
    return weights, iteration
```

Αντίστοιχα καλούμε τη συνάρτηση:

```
iris_dataset = load_iris()
iris_data = iris_dataset.data.transpose()
iris_labels = iris_dataset.target
iris_data_aug = np.vstack([np.ones([1, iris_data.shape[1]]), iris_data])
weights_k, iteration_k = keslers_construction(iris_data_aug, iris_labels, 1000)
print(weights_k)
print(iteration_k)
activations = np.dot(weights_k.transpose(), iris_data_aug)
predictions = np.argmax(activations, axis=0)
errors = np.where(predictions == iris_labels, 0, 1)
print("Total errors are ", errors.sum(), " out of ", iris_data.shape[1])
print("Classification error is ", errors.sum()/iris_data.shape[1])
print("------------------------------------------------------------")
```

Τα αποτελέσματα είναι τα εξής: