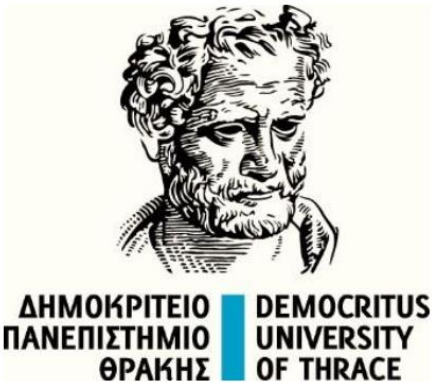


5Η ΕΡΓΑΣΙΑ

Στο μάθημα:
«Αναγνώριση Προτύπων»
Καθηγητής: Νικόλαος Μητιανούδης

Στυλιανός Μούσλεχ
ΑΜ:57382
6/12/2020 ,Ξάνθη

ΕΙΣΑΓΩΓΗ

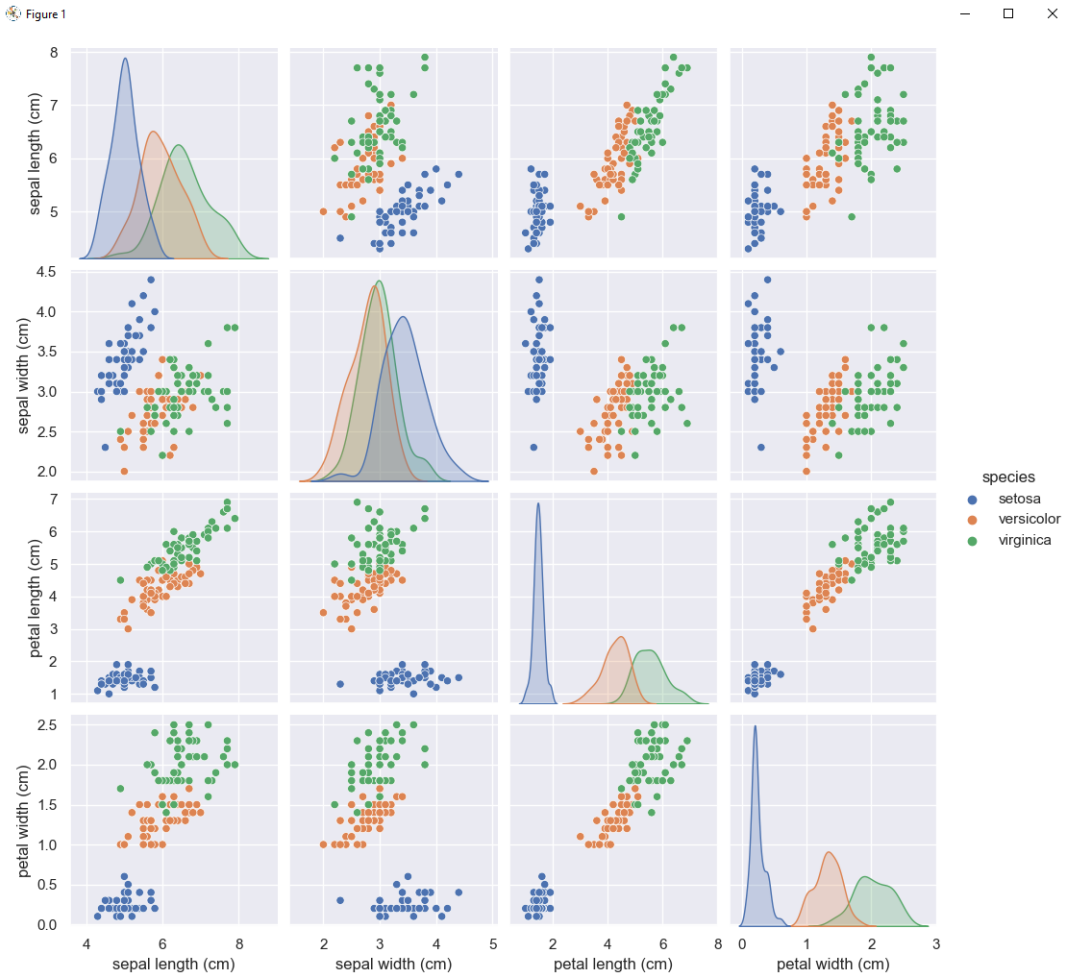
Στο αρχείο **“Ex5_myFunctions.py**” έχω γράψει όλες τις συναρτήσεις που χρησιμοποιώ για τα ερωτήματα. Για κάθε ερώτημα έχω ένα διαφορετικό αρχείο κώδικα που τρέχω. Στο ερώτημα C επειδή στην Python μάλλον κάτι γινόταν λάθος τον αλγόριθμο Ho Kashyapp τον έτρεξα και με MATLAB όπως θα δείτε παρακάτω.

Θα δουλέψω και με 4 χαρακτηριστικά στην εργασία.

Το iris dataset το φορτώνουμε από την βιβλιοθήκη sklearn, εκεί κάθε γραμμή είναι ένα δείγμα και κάθε στήλη είναι ένα χαρακτηριστικό .Τα χαρακτηριστικά έχουν την ίδια σειρά με την σειρά που τα δίνει η εκφώνηση (1=μήκος σέπαλου, 2=πλάτος σέπαλου,3=μήκος πετάλου,4= πλάτος πετάλου)

Αρχικά μπορούμε να κάνουμε μια πρόχειρη ανάλυση του dataset ώστε να ξέρουμε τι να περιμένουμε από τις παρακάτω μεθόδους.

Αυτήν την κάνω στο αρχείο **“Ex5_analysis.py”** όπου με τη χρήση της βιβλιοθήκης seaborn και pandas μπορούμε να κάνουμε ένα pair plot όλα τα ζευγάρια των χαρακτηριστικών και την κατανομή των τιμών του κάθε χαρακτηριστικού για κάθε κλάση:



Παρατηρώντας λοιπόν το παραπάνω plot βλέπουμε ότι η πρώτη κλάση είναι εύκολα γραμμικώς διαχωρίσιμη από τις άλλες 2 , αλλά ότι οι άλλες 2 μεταξύ τους δεν είναι.

Επίσης βλέπουμε ότι θα μπορούσαμε να κάνουμε standardization και normalization στα δεδομένα μας αλλά στα πλαίσια αυτής της εργασίας δουλεύουμε με τα αρχικά.

ΆΣΚΗΣΗ 5.Α

Ο κώδικας βρίσκεται στο αρχείο **"Ex5_a.py".**

Αρχικά έφτιαξα τις συναρτήσεις σε python για το batch perceptron και batch perceptron with margin βασιζόμενος στον αλγόριθμο της διάλεξης και τον κώδικα του βιβλίου του duda.

Αλγόριθμος batch perceptron:

Αλγόριθμος 3. Batch Perceptron

1 **begin initialize** a, κριτήριο θ, $\eta(0)>0$, k=0
2 **do** k ← k+1
3 $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\sum_{y \in Y_k} \mathbf{y}$
4 **until** $\left|\eta(k)\sum_{y \in Y_k} \mathbf{y}\right| < \theta$
5 **return** a
6 **end**

Ο κώδικας:

```
def duda_batch_Perceptron(data, dataLabels, learningRate, epochs, init_weights):
if init_weights.shape[0] != data.shape[1]:
   print("check dims of w or data")
   return

erros_per_epoch = []
N,l  = data.shape
data = np.hstack((data, np.ones((data.shape[0],1), dtype=data.dtype)))
weights = np.concatenate([init_weights,[1.0]])
iters = 0
errors = 1
while iters in range(epochs) and errors > 0:
   errors = 0

   gradi = np.zeros(l+1)
   for i in range(N):
      x = data[i,:]  # this makes X an 2d row!!!
      if (x.dot(weights))*dataLabels[i] < 0:
         errors += 1
         gradi=gradi+ learningRate*(-dataLabels[i]*x)

   weights=weights - learningRate*gradi
   erros_per_epoch.append(errors)
   iters+=1

return weights,erros_per_epoch,iters
```

Τα ορίσματα της συνάρτησης είναι ίδια με αυτά του κώδικα σε MATLAB με μόνη διαφορά ότι δίνουμε τα δεδομένα σε πίνακα ανεστραμμένο σε σχέση με το πως γίνεται στη συνάρτηση του βιβλίου.

Ο αλγόριθμος

Αλγόριθμος 6. Batch Relaxation with Margin

1 **begin initialize** $\mathbf{a}$, margin b, $\eta(0)$, $k=0$
2 **do** $k \leftarrow (k+1) \bmod n$
3 $Y_k=\{\}$
4 $j=0$
5 **do** $j \leftarrow j+1$
6 **if** $\mathbf{a}^t\mathbf{y}^k<b$, **then** append $\mathbf{y}^j$ to Y_k
7 **until** $j=n$
8 $\mathbf{a} \leftarrow \mathbf{a}+\eta(k)\sum_{y\in Y}\frac{b-\mathbf{a}^t\mathbf{y}}{\|\mathbf{y}\|^2}\mathbf{y}$
9 **until** $Y_k=\{\}$
10 **return** $\mathbf{a}$
11 **end**

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

Ο κώδικας:

```
def duda_batch_Perceptron_with_Margin(data, dataLabels,b, learningRate, epochs):
c,n= data.shape
data = np.vstack((data, np.ones((1,data.shape[1]), dtype=data.dtype)))
train_zero=np.argwhere(dataLabels==0)
#augment my daata
augmentData=np.copy(data)
for i in train_zero:
  augmentData[:,i]=-augmentData[:,i]

weights = np.sum(augmentData, axis=1)/n

yk=[0]
iter=0
errors_perepoch=[]
while yk and iter<epochs:
  iter += 1
  yk=[]
  errors=0

  for k in range(n):
    x=augmentData[:,k]
    if weights.dot(x) <= b:
      yk.append(k)
      errors=errors+1
  errors_perepoch.append(errors)
  if not yk:
    break

  y=augmentData[:,yk]
  var1 = b - weights.dot(y)
  var2= np.sum(y**2,axis=0)
  grad=var1/var2
  grad=np.atleast_2d(grad) # so matrix mul works correctly..
  test=np.ones((c+1,1)).dot(grad)
  test2=test*y
  update=np.sum(test2,axis=1)
```

```
    weights=weights + learningRate*update


return weights,iter,errors_perepoch
```

Εδώ τα ορίσματα είναι ίδια με του βιβλίου.

Έχοντας τις συναρτήσεις αυτές χωρίζουμε τα δεδομένα και τις κατηγορίες τους και κάνουμε τα labels από 0 1 και 2 σε -1 για την πρώτη κλάση και 1 για τις άλλες 2 (ομαδοποιημένες)

```
iris=load_iris()
X=iris.data
Y=iris.target
class_labels_perceptron=np.where(Y>0,-1,1)
```

και καλώντας την συνάρτηση βλέπουμε ότι έχουμε training error 0 μετά από 4 επαναλήψεις. Το αποτέλεσμα είναι πολύ λογικό καθώς η ω1 είναι γραμμικώς διαχωρίσιμη από τις ω2 και ω3.

Το αποτέλεσμα και τα βάρη:

```
Batch Perceptron ,withs weights:  [-0.01736  1.45036 -1.78964 -0.1583   1.     ]
Number of missclassifications:  0
After  4  Iterations
```

Καλώντας τώρα την συνάρτηση για Batch Perceptron with Margin και υπολογίζοντας το error.

```
class_labels_perceptron2=np.where(Y>0,1,0)
weights_margin,iter,errors_perepoch=ex5.duda_batch_Perceptron_with_Margin(X.T,class_labels_perceptron
2,0.2,0.02,40)

print("Batch perceptron with Margin with Weights: :",weights_margin)

#error calculation
data = np.hstack((X, np.ones((X.shape[0], 1), dtype=X.dtype)))
decision_values=data.dot(np.atleast_2d(weights_margin).T)
missclassified=np.sum((decision_values.T*class_labels_perceptron)>0) #>0 cause we make -data the data of
class 0 in margin batch perceptron
error_rate=missclassified/X.shape[0]
print("Using Batch perceptron with Margin,error is: ",error_rate)
```

βλέπουμε τα βάρη τώρα καθώς και το error:

```
Batch perceptron with Margin with Weights: : [-0.11050223 -1.00794183  2.01180532  0.90546662 -0.19287015]
Using Batch perceptron with Margin,error is:  0.0
```

Εδώ είναι πάλι 0 και θεωρητικά η απόσταση της επιφάνειας απόφασης απέχει περισσότερο από τα σημεία μας (δεν έκανα οπτικοποίηση λόγω των διαστάσεων που έχουμε) Βέβαια η σύγκλιση σε αυτόν τον αλγόριθμο φάνηκε να έγινε πιο δύσκολα από οτι στο πρώτο.

ΆΣΚΗΣΗ 5.Β

Ο κώδικας βρίσκεται στο αρχείο **"Ex5_b.py".**

Στην αρχή θα υπολογίσουμε την αναλυτική λύση των βαρών με την χρήση του ψευδοαντιστρόφου με τον παρακάτω τρόπο:

$$\mathbf{a} = \underbrace{\left(\mathbf{Y}^t\mathbf{Y}\right)^{-1}\mathbf{Y}^t}_{\text{ψευδοαντίστροφος}}\ \mathbf{b}$$

Στο MSE αντί να αλλάξουμε το πρόσημο στα Y για την άλλη κλάση (ω2 και ω3) Θα αλλάξω το πρόσημο στο b για τα δεδομένα που ανήκουν στις άλλες κλάσεις.

Αυτή την υλοποίηση την κάνουμε με το παρακάτω κώδικα:

```
iris=load_iris()
X=iris.data
Y=iris.target
data = np.hstack((X, np.ones((X.shape[0], 1), dtype=X.dtype)))
binary_classes_neg=np.where(Y>0,-1,1)
b=np.where(Y>0,-1,1)
weights=np.linalg.pinv(data).dot(b)
print("Using MSE with LS, weights are: ",weights)

#calculate error
decision_values=data.dot(np.atleast_2d(weights).T)
missclassified=np.sum((decision_values.T*binary_classes_neg)<0)
error_rate=missclassified/X.shape[0]
print("Using MSE error is: ",error_rate)
```

και έχουμε το παρακάτω βάρη και error rate:

```
Using MSE with LS, weights are:  [ 0.13205954  0.48569574 -0.44931423 -0.11494546 -0.76355422]
Using MSE error is:  0.0
```

Τα αποτελέσματα είναι συνεπή με αυτό που αναφέραμε και παραπάνω, δηλαδή ότι εφόσον τα δεδομένα μας είναι γραμμικώς διαχωρίσιμα μπορούμε να βρούμε μια διαχωριστική επιφάνεια που να χωρίζει τις κλάσεις με 0 error.

Στην συνέχεια κάνω και την επαναληπτική μέθοδο LMS:

Αλγόριθμος 8. Widrow-Hoff (LMS)

1 **begin initialize** a, b, κριτήριο θ, $\eta()$, k=0
2 **do** $k \leftarrow (k+1) \bmod n$
3 $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\left(b_k - \mathbf{a}^t\mathbf{y}^k\right)\mathbf{y}^k$
4 **until** $\left|\eta(k)\left(b_k - \mathbf{a}^t\mathbf{y}^k\right)\mathbf{y}^k\right| < \theta$
5 **return** a
6 **end**

Η υλοποίηση αυτής της συνάρτησης:

```
def duda_LMS(data,dataLabels,max_iter,theta,learningRate):
    c, n = data.shape
    data = np.vstack((data, np.ones((1, data.shape[1]), dtype=data.dtype)))


    # augment my daata
    augmentData = np.copy(data)
    train_zero=np.argwhere(dataLabels==0)
    b = 2*dataLabels -1
    for i in train_zero:
        augmentData[:, i] = -augmentData[:, i]
    weights=np.sum(augmentData,axis=1)/n
    update=100
    updates=[]
    k=0
    iter=0

    while np.sum(np.abs(update)) > theta and iter<max_iter :
        iter +=1
        #changing a bit cause python arrays index at 0
        y=data[:,k]
        var=learningRate*(b[k] - weights.dot(y))
        update = var*y
        weights=weights+update
        updates.append(np.sum(np.abs(update)))
        k+=1
        if k==n:
            k=0

    return weights,updates
```

Οπότε με βάση αυτή τη συνάρτηση ξανά αλλάζουμε τα labels με τον τρόπο που χρειάζεται η συνάρτηση και την καλούμε. Τέλος υπολογίζουμε το error rate:

```
binary_classes=np.where(Y>0,0,1)
weights_lms,updates=ex5.duda_LMS(X.T,binary_classes,200,0.0001,0.02)
print("LMS Weights: :",weights_lms)
#calculate error
decision_values=data.dot(np.atleast_2d(weights_lms).T)
missclassified=np.sum((decision_values.T*binary_classes_neg)<0)
error_rate=missclassified/X.shape[0]
print("Using LMS error is: ",error_rate)
```

Και έχουμε τα παρακάτω αποτελέσματα:

```
LMS Weights: : [ 0.12509663  0.70972623 -1.45426298 -0.61808835  0.16061937]
Using LMS error is:  0.0
```

Το error rate είναι αναμενόμενο, επίσης παρατηρούμε ότι τα weights είναι διαφορετικά από πριν.

ΆΣΚΗΣΗ 5.C

Ο κώδικας βρίσκεται στο αρχείο **“Ex5_c.py”.**

Εδώ πλέον καλούμαστε να διαχωρίσουμε το ω2 με το ω3. Όπως είδαμε με την αρχική ανάλυση τα δεδομένα φαίνεται να μην είναι γραμμικώς διαχωρίσιμα οπότε περιμένουμε να δούμε και κάποιο error rate.

Αρχικά χρησιμοποιώντας την μέθοδο του ψευδοαντιστρόφου όπως κάναμε και πριν:

```
iris=load_iris()
X=iris.data
Y=iris.target
data = np.hstack((X, np.ones((X.shape[0], 1), dtype=X.dtype)))
#get w2 and w3 data
newData=data[50:150,:]
newLabels=Y[50:150]
binary_classes_neg=np.where(newLabels>1,1,-1)
b=np.where(newLabels>1,1,-1)
weights=np.linalg.pinv(newData).dot(b)
print("Using MSE with LS, weights are: ",weights)
decision_values=newData.dot(np.atleast_2d(weights).T)
missclassified=np.sum((decision_values.T*binary_classes_neg)<0)
error_rate=missclassified/newData.shape[0]

print("Using MSE error is: ",error_rate)
```

και έχουμε τα παρακάτω αποτελέσματα:

```
Using MSE with LS, weights are:  [-0.3921192  -0.6151007   0.76852876  1.3656893  -1.83727773]
Using MSE error is:  0.03
```

Έχουμε αρκετά καλό error rate αλλά δεν είναι 0 κάτι λογικό με βάση αυτό που είπαμε και πριν.

Στην συνέχεια έκανα την υλοποίηση του αλγορίθμου Ho-Kashyap :

Αλγόριθμος 9. Ho-Kashyap

1 **begin initialize** **a**, **b**, η()<1, threshold b_{min}, k_{max}
2 **do** $k \leftarrow (k+1) \bmod n$
3 $\mathbf{e} \leftarrow \mathbf{Ya}-\mathbf{b}$
4 $\mathbf{e}^{+} \leftarrow (\mathbf{e}+|\mathbf{e}|)/2$
5 $\mathbf{b} \leftarrow \mathbf{b} + 2\eta(k)\mathbf{e}^{+}$
6 $\mathbf{a} \leftarrow (\mathbf{Y}^{t}\mathbf{Y})^{-1}\mathbf{Yb}$
7 **if** Abs(**e**)< b_{min} **then return** **a**, **b** and **exit**
8 **until** k= k_{max}
9 print “No solution found”
10 **end**

Με βάση τα τον κώδικα τον διαφανειών έκανα :

```
def duda_Ho_Kashyap(data,labels,type,max_iter,b_min,learningRate):

    c,n=data.shape
    train_class2=np.squeeze(np.argwhere(labels==-1))
    Y=data
    Y[:,train_class2]=-Y[:,train_class2]
    b=np.ones((1,n))
    weights=(np.linalg.pinv(Y.T)).dot(b.T)
    k=0
    e=1000
    found=0
    test=0
    criterion=np.sum(np.greater(np.abs(e),b_min))
    while criterion>0 and (k<max_iter) and (not found):
        k=k+1
        var=(Y.T).dot(weights)
        e=var.T-b
        e_plus=0.5*(e+np.abs(e))
        b=b+ (2*learningRate*e_plus)

        if type==0:
            weights=(np.linalg.pinv(Y.T)).dot(b.T)

        else:
            weights=weights+ learningRate*((np.linalg.pinv(Y.T)).dot(e_plus.T))
        criterion=np.sum(np.greater(np.abs(e),b_min))



    if k == max_iter:
        print("Algorithm did not find solution")
    else:
        print("found solution after", k, " iterations")
    return weights,b
```

και καλώντας την συνάρτηση:

```
weights_Ho,b=ex5.duda_Ho_Kashyap(newData.T,binary_classes_neg,0,100000,0.001,0.0001)

test=weights_Ho.T.dot(newData.T)
missclassified=np.where(test>0,1,0)
k=np.squeeze(2*missclassified-1)
error_rate=np.sum(np.where(k!=binary_classes_neg,1,0))/newData.shape[0]
print("Using Ho Kashyap , weights are: ",weights_Ho)
print("Using Ho Kashyap error is: ",error_rate)
```

Έχουμε τα παρακάτω (λανθασμένα μάλλον) αποτελέσματα:

```
Algorithm did not find solution
Using Ho Kashyap , weights are:  [[-0.64951741 -0.95674141  1.51318331  2.29267254 -4.42599239]]
Using Ho Kashyap error is:  0.49
```

Εδώ ότι και να προσπαθούσα σε υπερπαραμέτρους είχα πολύ υψηλό error rate, κάτι που δεν έβγαζε νόημα. Επειδή θεωρώ ότι ίσως συμβαίνει κάτι λάθος στην μεταφορά του αλγορίθμου σε Python δοκίμασα στη συνέχεια να τρέξω τον ίδιο αλγόριθμο σε MATLAB με βάση την συνάρτηση της διαφάνειας έγραψα τον παρακάτω κώδικα (το αρχείο είναι το **“testHoKashyapp.m”**):

```
%% Ho Kashyap
[X, ~] = iris_dataset;
X = X(:,51:150);
y = [ ones(1,50) -(ones(1,50))];
X1=[X;ones(1,100)];
[w,b]=Ho_Kashyap_cc(X1,y, 0, 1000, 0.1, 0.01);

KH_out=2*(w'*X1>0)-1;
err_KH=sum(KH_out.*y<0)/100
```

Και έχουμε το παρακάτω αποτέλεσμα:

```
>> testHoKashyapp
No solution found
weights are:

w =

    0.6496
    0.9569
   -1.5134
   -2.2932
    4.4272

error rate is

err_KH =

    0.0300
```

Αυτό το αποτέλεσμα είναι πιο συνεπές με τη προηγούμενη μέθοδο (μάλιστα ίδιο error rate).

ΆΣΚΗΣΗ 5.D

Ο κώδικας βρίσκεται στο αρχείο **“Ex5_d.py”.**

Εδώ ουσιαστικά είναι η ίδια μέθοδος με πριν απλώς κάνουμε μια κατηγοριοποίηση one vs all κάθε φορά (αλλάζοντας τον πίνακα b κάθε φορά) για κάθε κλάση εναντίων των άλλων και έτσι έχουμε 3 διανύσματα με βάρη.

Για να κατηγοριοποιήσουμε ένα δείγμα σε κάποια κλάση θα πάρουμε το μέγιστο γινόμενο δεδομένου με βάρη, ουσιαστικά σε ποια κλάση «ανήκει περισσότερο».

Η παραπάνω λογική υλοποιείται εδώ:

```
iris=load_iris()
X=iris.data
Y=iris.target
data = np.hstack((X, np.ones((X.shape[0], 1), dtype=X.dtype)))
weights=[]
for i in range(3):
   b=np.where(Y==i,1,-1)
   weights.append(np.linalg.pinv(data).dot(b))
   print("Using MSE with LS for class,",i,", weights are: ",weights[i])

decision1=data.dot(np.atleast_2d(weights[0]).T)
decision2=data.dot(np.atleast_2d(weights[1]).T)
decision3=data.dot(np.atleast_2d(weights[2]).T)
decisionsGrouped=np.hstack((decision1,decision2,decision3))
classifications=np.argmax(decisionsGrouped,axis=1)
errors=np.sum(np.where(Y!=classifications,1,0))
print("Error Rate for multiclass classification: ",errors/X.shape[0])
```

και έχουμε τα παρακάτω αποτελέσματα

```
C:\Users\Stelios\AppData\Local\Programs\Python\Python36\python.exe D:/PatternRec2/Ex5_d.py
Using MSE with LS for class, 0 , weights are:  [ 0.13205954  0.48569574 -0.44931423 -0.11494546 -0.76355422]
Using MSE with LS for class, 1 , weights are:  [-0.04030737 -0.89123252  0.44133841 -0.98861319  2.15411795]
Using MSE with LS for class, 2 , weights are:  [-0.09175217  0.40553677  0.00797582  1.10355865 -2.39056373]
Error Rate for multiclass classification:  0.15333333333333332
```

Παρατηρούμε ότι στο Multi Class πρόβλημα έχουμε μεγαλύτερη δυσκολία και το Error Rate μας είναι 15%.

ΆΣΚΗΣΗ 5.Ε

Ο κώδικας βρίσκεται στο αρχείο **“Ex5_e.py” για (1,2,3) και “Ex5_e2.py”. για (2,3,4)**

(Έχω ένα τυπογραφικό στα plots όπου m είναι cm)

Εδώ ουσιαστικά κάνουμε ακριβώς το ίδιο πράγμα με το ερώτημα D για απλά για άλλες διαστάσεις κάθε φορά Δηλαδή :

Για (1,2,3) τα δεδομένα:

```
data_dims123=X[:,:3]
```

Για (2,3,4) τα δεδομένα:

```
data_dims123=X[:,1:4]
```

Πρώτα για χαρακτηριστικά 1,2,3:

Έχουμε τα παρακάτω αποτελέσματα:

```
Using MSE with LS for class, 0 , weights are:  [ 0.15588383  0.46008261 -0.50955521 -0.73593198]
Using MSE with LS for class, 1 , weights are:  [ 0.16459861 -1.11152375 -0.07677707  2.391689  ]
Using MSE with LS for class, 2 , weights are:  [-0.32048244  0.65144114  0.58633228 -2.65575702]
Error Rate for multiclass classification:  0.18666666666666668
```

Και στη συνέχεια κάνουμε plot κάνοντας scatter plot τα δεδομένα και υπολογίζοντας την επιφάνεια απόφασης με βάση τον τύπο

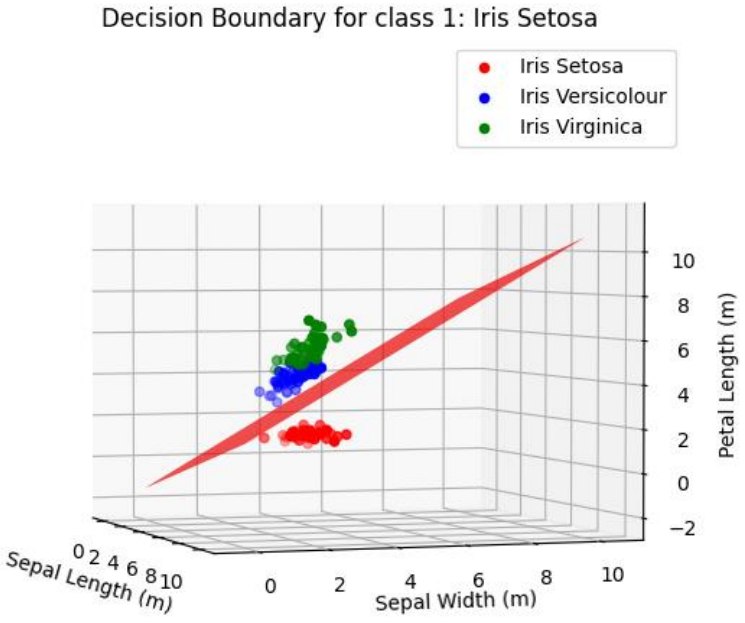
$$w^T X + w_0 = 0$$

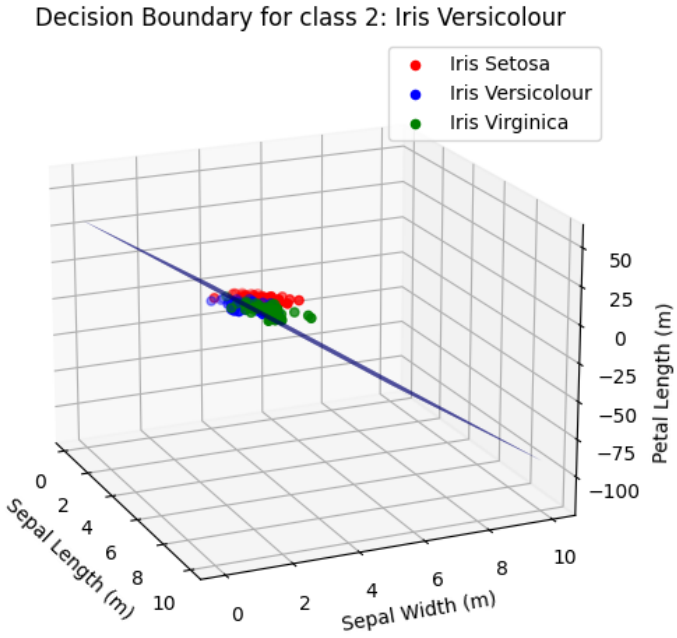
Κάνουμε ένα meshgrid για την κάθε επιφάνεια:

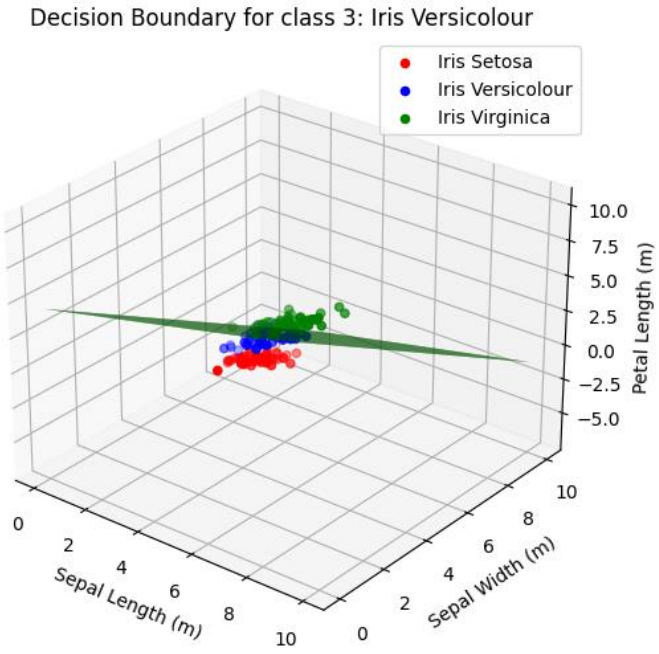
```
z1 = lambda x,y: (-weights[0][3]-weights[0][0]*x -weights[0][1]*y) / weights[0][2]
z2 = lambda x,y: (-weights[1][3]-weights[1][0]*x -weights[1][1]*y) / weights[1][2]
z3 = lambda x,y: (-weights[2][3]-weights[2][0]*x -weights[2][1]*y) / weights[2][2]

tmp = np.linspace(0,10,30)
x,y = np.meshgrid(tmp,tmp)
```

Τα γραφήματα:

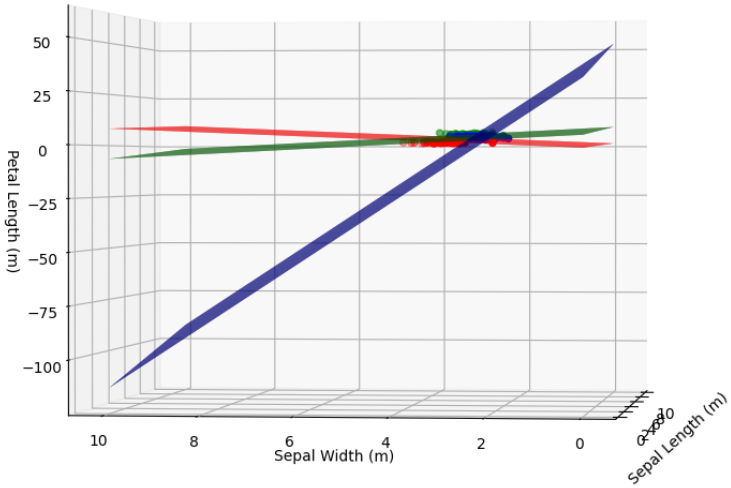






Decision Boundaries Grouped

Και όλα μαζί:

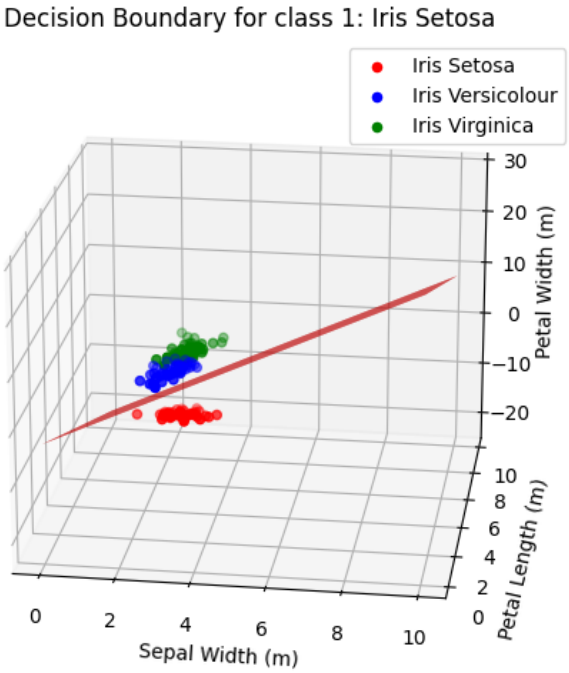


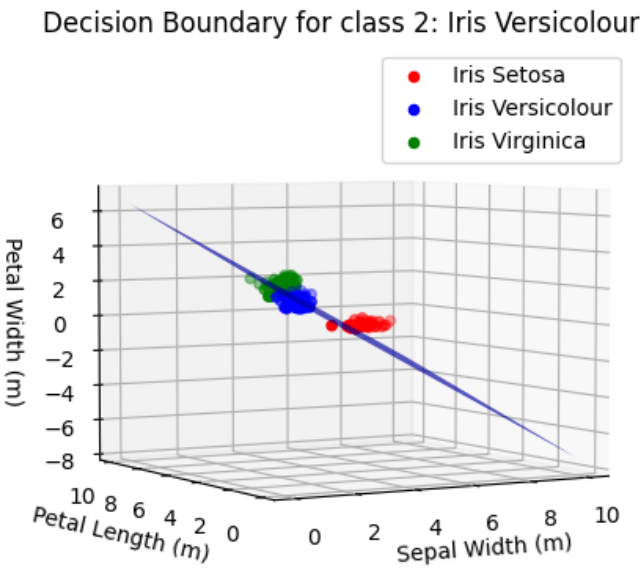
Επαναλαμβάνουμε για τον χώρο (2,3,4):

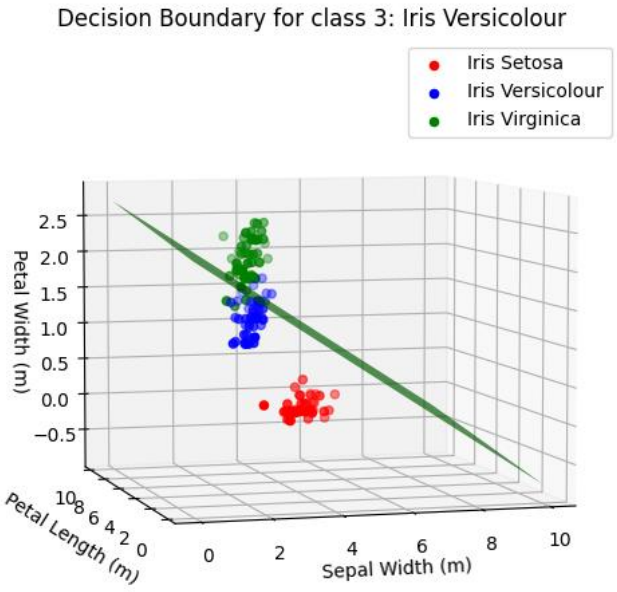
```
Using MSE with LS for class, 0 , weights are:  [ 0.571645   -0.35566659 -0.1884343  -0.51845205]
Using MSE with LS for class, 1 , weights are:  [-0.91746605  0.41275517 -0.96618284  2.07930757]
Using MSE with LS for class, 2 , weights are:  [ 0.34582105 -0.05708857  1.15461714 -2.56085552]
Error Rate for multiclass classification:  0.14666666666666667
```

Το error είναι λίγο μικρότερο από πριν. Αν παρατηρήσουμε το pair plot στην αρχή το αποτέλεσμα αυτό θα μπορούσε να σημαίνει ότι το χαρακτηριστικό 1 (petal length) μας μπερδεύει περισσότερο.

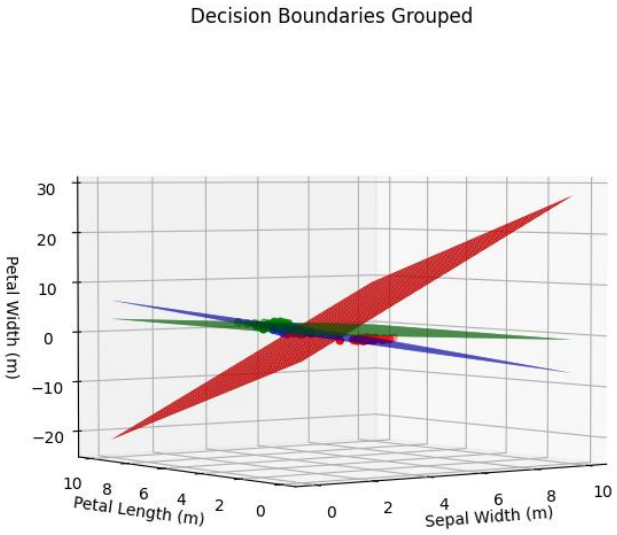
Έχουμε και τα αντίστοιχα plots:







Και όλα μαζί :



ΆΣΚΗΣΗ 5.F

Ο κώδικας βρίσκεται στο αρχείο **“Ex5_f.py”**

Υλοποιώ την συνάρτηση mperceptron με την χρήση της δομής Kesler σε python:

```
def mperceptron(data,labels,max_iter):

  dim,num_data=data.shape
  nclass=np.max(labels)+1
  W=np.zeros((dim,nclass))
  b=np.zeros((nclass,1))
  t=0
  flag=0
  while max_iter>t and flag==0:
    t+=1
    flag=1
    for i in range(nclass):
      class_i=np.argwhere(labels==i)
```

```
        class_i=np.squeeze(class_i)
        dfce_i= np.dot(W[:,i].T,data[:,class_i])+b[i,0]

        for j in np.setdiff1d(np.arange(start=0,stop=nclass),np.atleast_1d(i)):
            dfce_j=np.dot(W[:,j].T,data[:,class_i])+b[j,0]
            min_dif=np.min(dfce_i-dfce_j,axis=0)
            inx=np.argmin(dfce_i-dfce_j,axis=0)
            inx=np.atleast_1d(inx)
            if min_dif  <=0:

                inx=class_i[inx[0]]
                W[:,i]=W[:,i]+ data[:,inx]
                b[i,0]=b[i,0]+1

                W[:, j] = W[:, j] - data[:, inx]
                b[j,0] = b[j,0] - 1
                flag=0
                break
        if flag==0:
            break

    return W,b,t
```

Στη συνέχεια τρέχω τον αλγόριθμο για διαφορετικά iterations, καθώς εφόσον οι κλάσεις μας δεν είναι γραμμικώς διαχωρίσιμες ο αλγόριθμος δεν περιμένω να συγκλίνει με περισσότερες επαναλήψεις :

```
weights,bias,t=ex5.mperceptron(data,Y,1000)
fullWeights=np.vstack((weights,bias.T))
print("full Weights after 1000 iters for 3 classes\n",fullWeights)
dataAug = np.vstack((data, np.ones((1, data.shape[1]), dtype=data.dtype)))
test=np.dot(fullWeights.T,dataAug)
classifications=np.argmax(test.T,axis=1)
errors=np.sum(np.where(Y!=classifications,1,0))
print("Error Rate for multiclass classification Using Kesler for 1000 iters: ",errors/X.shape[0])
weights,bias,t=ex5.mperceptron(data,Y,10000)
fullWeights=np.vstack((weights,bias.T))
print("full Weights after 10000 iters for 3 classes\n",fullWeights)

dataAug = np.vstack((data, np.ones((1, data.shape[1]), dtype=data.dtype)))
test=np.dot(fullWeights.T,dataAug)
classifications=np.argmax(test.T,axis=1)
errors=np.sum(np.where(Y!=classifications,1,0))
print("Error Rate for multiclass classification Using Kesler for 10000 iters: ",errors/X.shape[0])
```

Τα αποτελέσματα είναι τα εξής:

```
full Weights after 1000 iters for 3 classes
 [[ 10.3  -0.6  -9.7]
 [ 28.7 -11.7 -17. ]
 [-46.3  14.3  32. ]
 [-24.2   6.1  18.1]
 [  6.   17.  -23. ]]
Error Rate for multiclass classification Using Kesler for 1000 iters:  0.05333333333333334
full Weights after 1000 iters for 3 classes [[ 10.3  -6.2  -4.1]
 [ 28.7 -14.  -14.7]
 [-46.3  10.   36.3]
 [-24.2   4.2  20. ]
 [  6.   17.  -23. ]]
Error Rate for multiclass classification Using Kesler for 10000 iters:  0.3333333333333333
```

Παρατηρώ μάλιστα ότι για λιγότερα iterations έχω πάρα πολύ καλύτερο error rate.