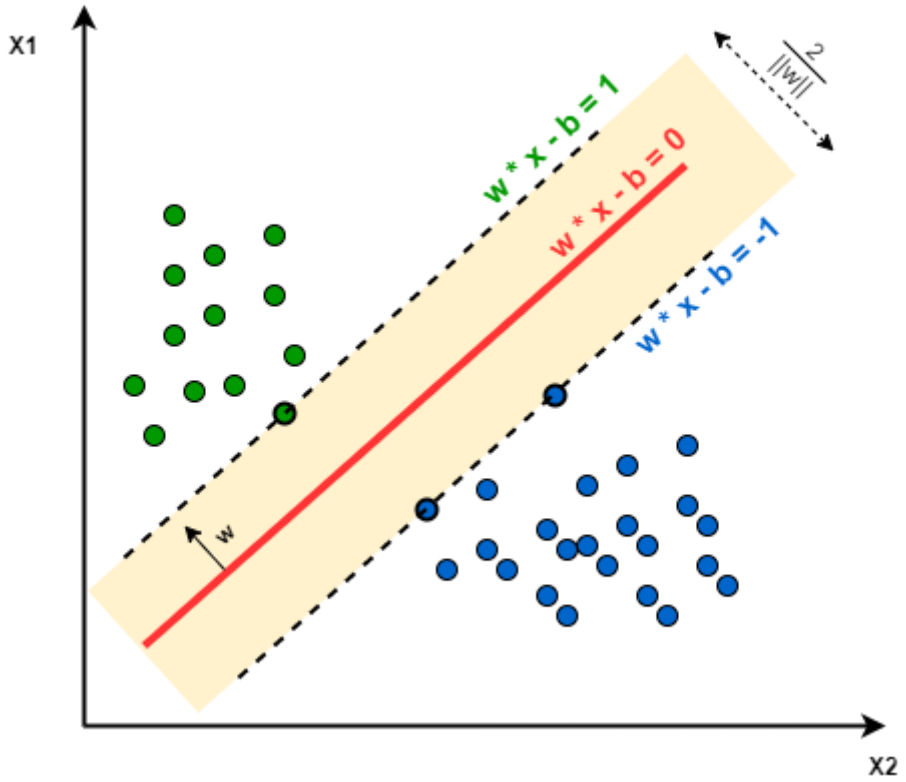
Εργασία 2 στην Αναγνώριση προτύπων



Χατζησάββας Χρήστος

Ακαδημαϊκό έτος 2023-2024

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΤΜΗΜΑ ΗΜ & ΜΥ

Άσκηση 1:

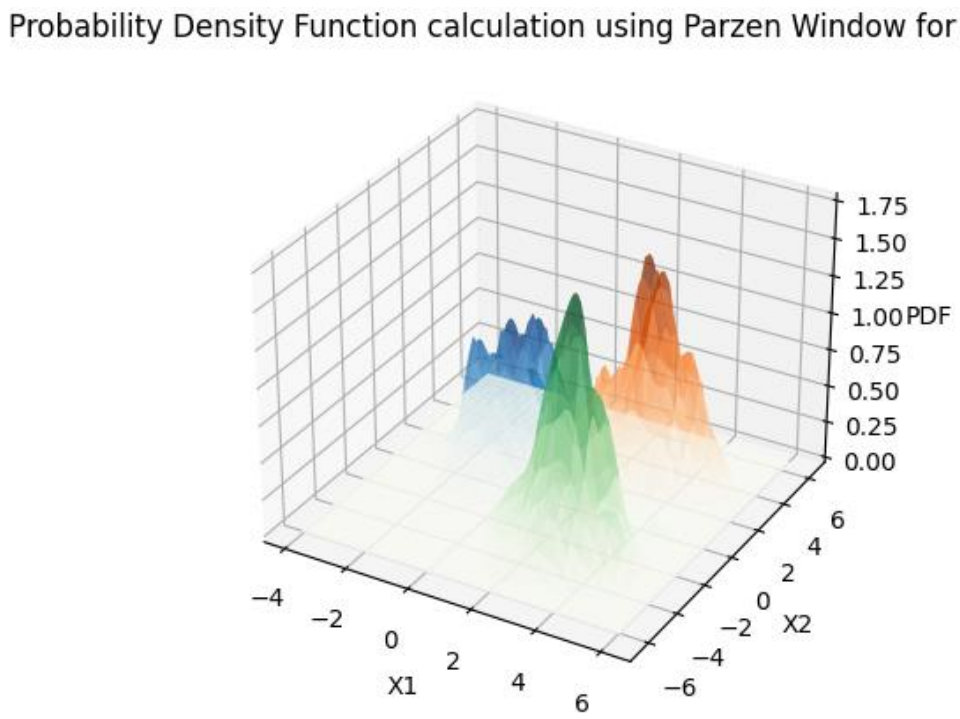
Α) Για την υλοποίηση του παραθύρου Parzen, η συνάρτηση που μας επιστρέφει την συνάρτηση πυκνότητας πιθανότητας σύμφωνα με τις διαφάνειες είναι:

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{Vn}\varphi(\frac{x - xi}{h_n})$$

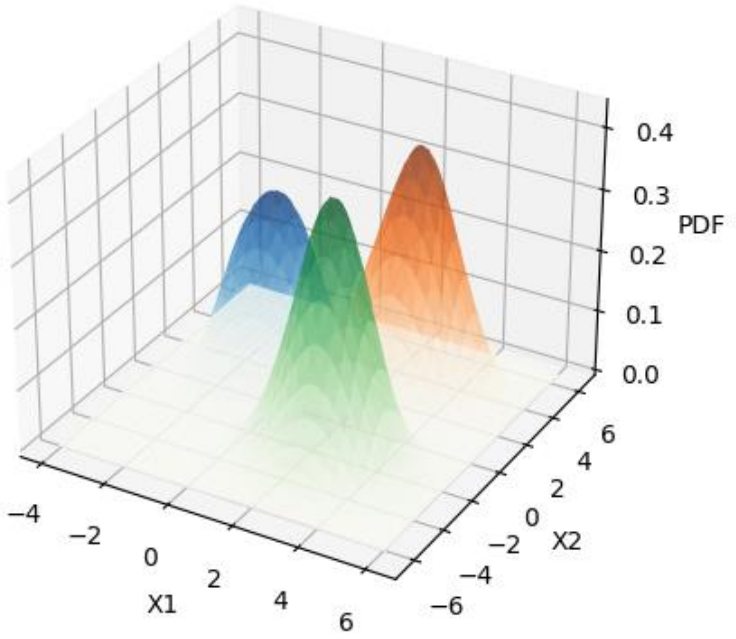
Χρησιμοποιώντας την συνάρτηση παραθύρου που μας δίνει η εκφώνηση ο κώδικας για το παράθυρο Parzen είναι ο ακόλουθος:

```
def parzen_window(data,h,x):
  a = (- np.sum((data - x)**2,axis = 1)/(2*h**2))
  b = np.zeros([data.shape[0]])
  for i in range(data.shape[0]):
    b[i] = (np.exp(a[i]) /(h*np.sqrt(2*np.pi)))/h**2

  px = np.sum(b)/data.shape[0]
  return px
```

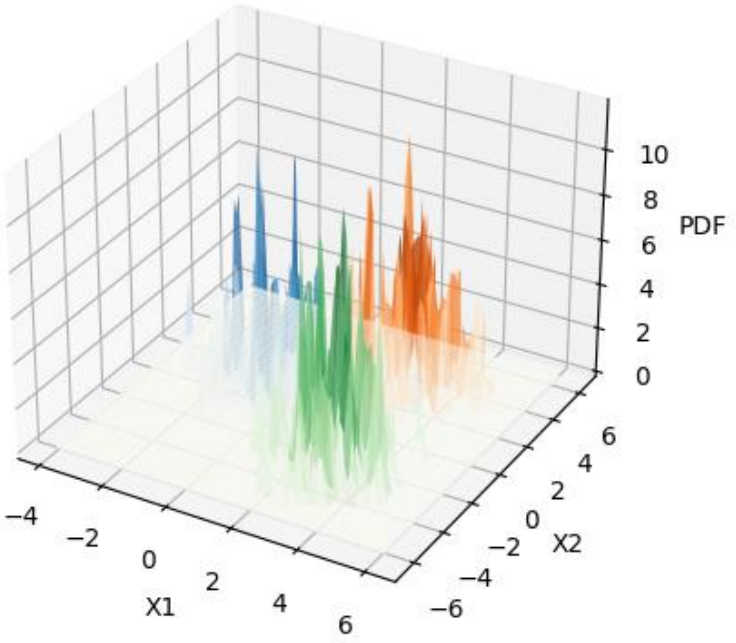
Για τις ζητούμενες τιμές του h (0.1,0.3,0.7) τα διαγράμματα των συναρτήσεων πυκνότητας πιθανότητας είναι:



Probability Density Function calculation using Parzen Window for h = 0.7



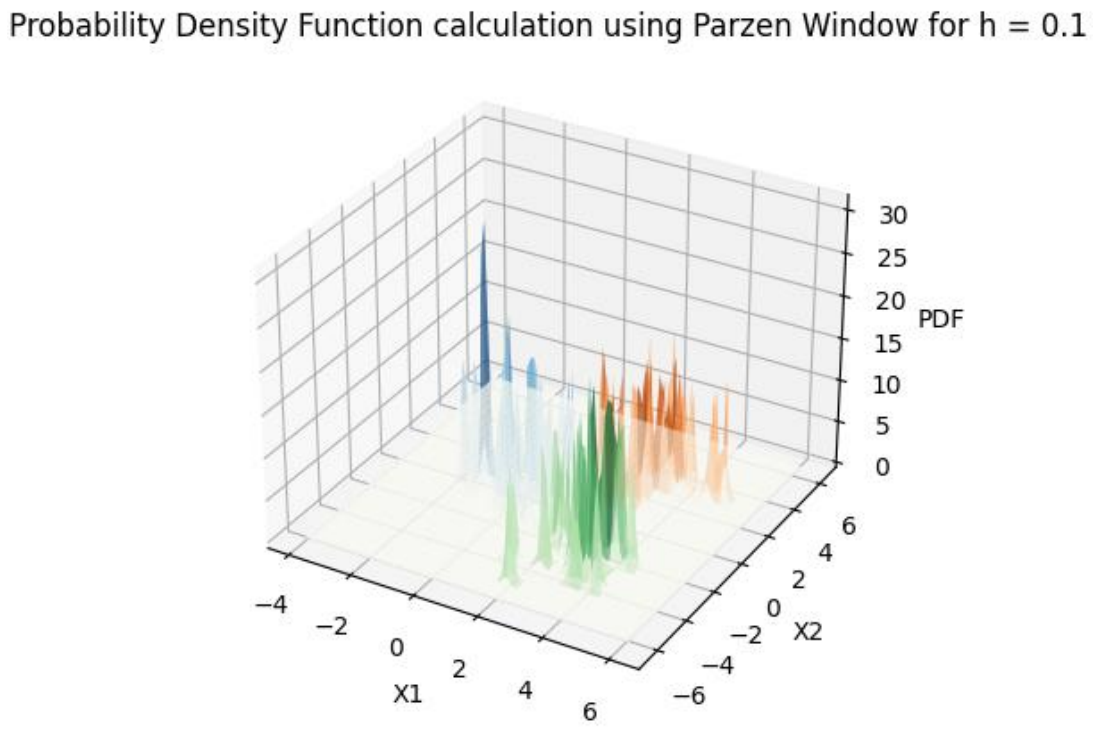
Probability Density Function calculation using Parzen Window for h = 0.1



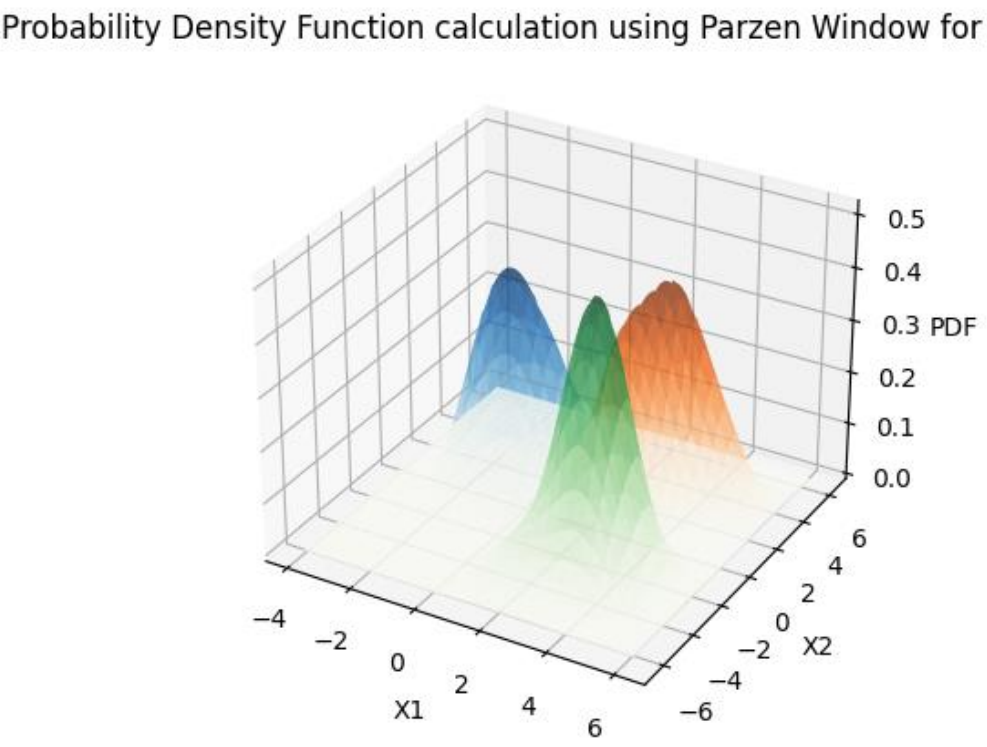
Η συνάρτηση που δημιουργεί τα παραπάνω plots υπάρχει στο jupyter notebook της εργασίας. Πιο συγκεκριμένα η συνάρτηση αυτή λαμβάνει ως ορίσματα τα δεδομένα των 3 κλάσεων και την παράμετρο h και παράγει ως έξοδο τις πυκνότητες πιθανότητας στον χώρο.

Γνωρίζουμε από την θεωρία πως όταν το h είναι μεγάλο η εκτίμηση του p(x) είναι ομαλή και χωρίς μεγάλη ανάλυση. Αντίθετα, όταν το h είναι μικρό η εκτίμηση του p(x) είναι θορυβώδης. Πιο συγκεκριμένα, όσο μικρότερο είναι το h, παρατηρούμε πως η συνάρτηση πυκνότητας πιθανότητας εμφανίζει περισσότερα spikes διότι γίνεται σημαντική η ύπαρξη θορύβου. Το φαινόμενο αυτό μοιάζει με overfitting.

Μειώνοντας το dataset στο 25% του αρχικού διατηρώντας τις αναλογίες των σημείων από κάθε κλάση παρατηρούμε πως για να επιτευχθεί λεπτομέρεια ίδια με αυτή που έχουμε όταν αξιοποιούμε όλα τα δεδομένα πρέπει να έχουμε μεγάλο h. Αυτό γίνεται ιδιαίτερα αντιληπτό από τα παρακάτω διαγράμματα. Παρατηρούμε πως για h = 0.1 έχουμε πολλά spikes που δείχνουν πως υπάρχει θόρυβος.



Αντιθέτως, για h = 0.7 παρατηρούμε μεγάλη ομοιότητα και παρόμοια λεπτομέρεια με την αρχική.



B) Για την υλοποίηση του αλγορίθμου KNN προκειμένου να υπολογισθεί η συνάρτηση πυκνότητας πιθανότητας αξιοποιήθηκε η παρακάτω διαδικασία από το υλικό του μαθήματος.

1. Επιλέγουμε μία τιμή για το k καθώς και την συνάρτηση μέτρου που θα χρησιμοποιήσουμε για την απόσταση (Ευκλείδεια, Mahalanobis, Manhattan, ..)
2. Βρίσκουμε την απόσταση του **x** από όλα τα δεδομένα
3. Βρίσκουμε τα πλησιέστερα k δεδομένα στο **x**
4. Υπολογίζουμε τον όγκο V(x) που περικλείει τα k σημεία
5. Προσεγγίζουμε την pdf στο χ με $p(\mathbf{x}) \approx \frac{k}{NV(\mathbf{x})}$

Εάν έχουμε επιλέξει την Ευκλείδεια απόσταση στο d-διάστατο χώρο και η απόσταση από το μακρινότερο δείγμα είναι ρ τότε

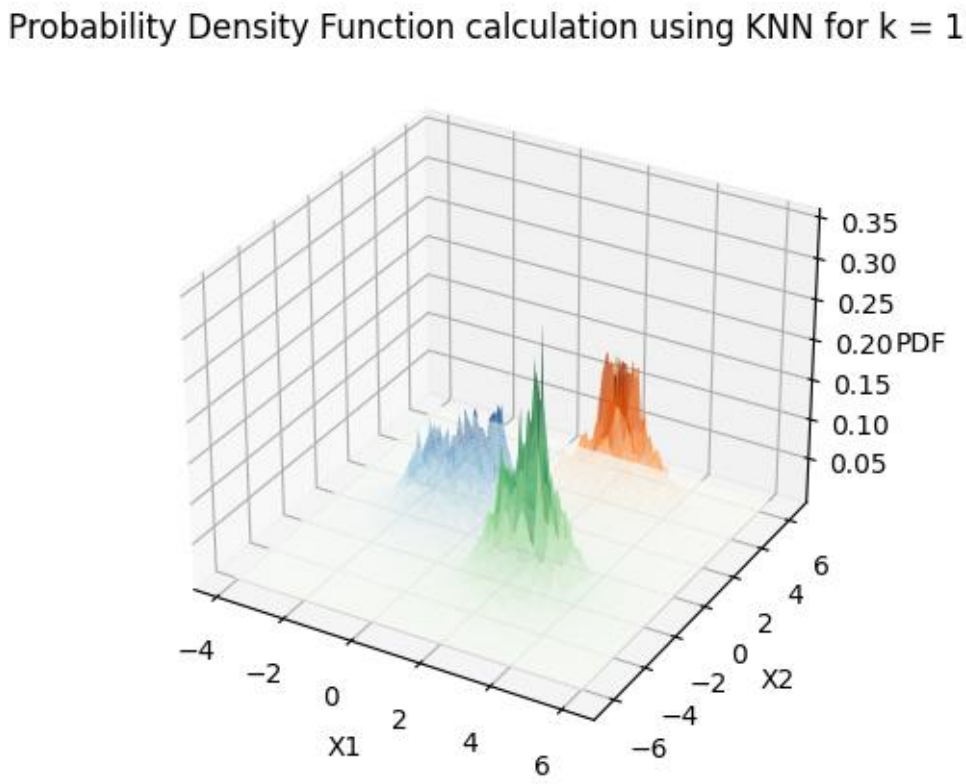
$V(x) = 2\rho$ για d=1, $V(x) = \pi\rho^2$ για d=2, $V(x) = (4/3)\pi\rho^2$ για d=3

Για την υλοποίηση στο πλαίσιο της εργασίας επιλέχθηκε η ευκλείδεια απόσταση. Δεδομένου ότι έχουμε 2 χαρακτηριστικά βρισκόμαστε στον 2-διάστατο χώρο. Ακολουθεί ο κώδικας που περιγράφει την παραπάνω διαδικασία.

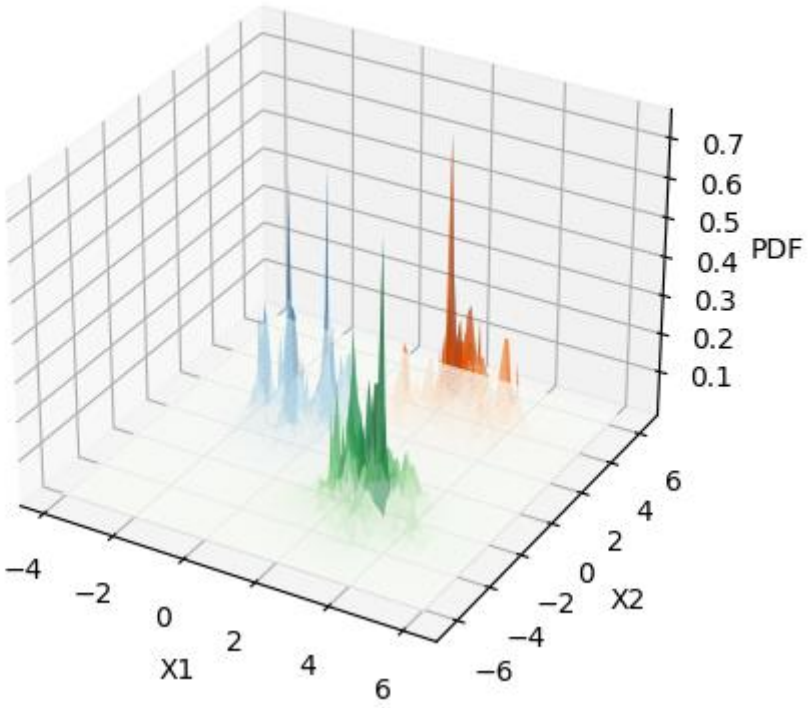
```
def KNN_algorithm(data,x,k):
  a = np.zeros([data.shape[0]])
  for i in range (data.shape[0]):
    a[i] = np.sqrt((data[i,0]-x[0])**2+(data[i,1]-x[1])**2)
```

```
  nearest_distances = []
  for i in range(k+1):
    nearest = np.argmin(a)
    nearest_dist = np.min(a)
    nearest_distances.append(nearest_dist)
    a[nearest] = np.inf

  nearest_distances = nearest_distances[1:]
  V = (np.pi)*(nearest_distances[-1]**2)
  pdf = k/(data.shape[0]*V)
  return pdf
```

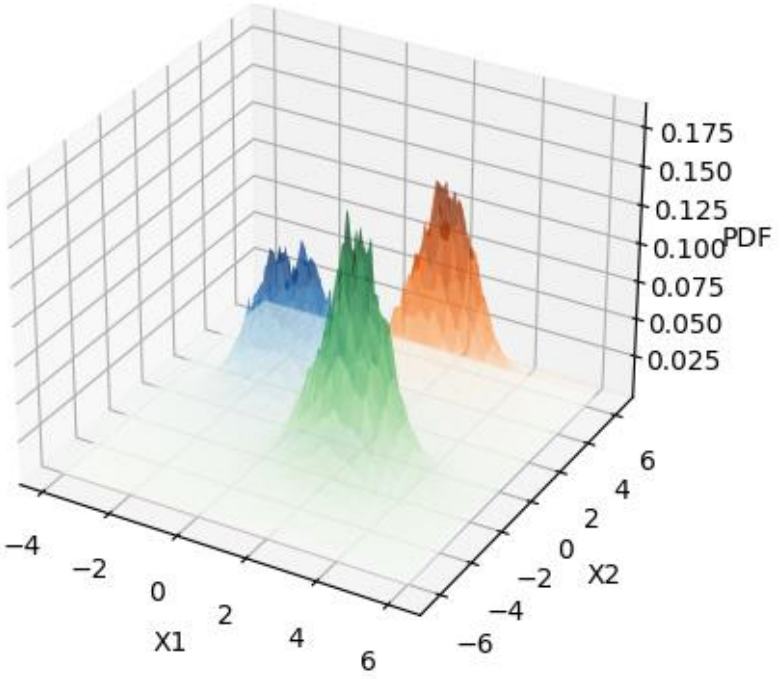
Οι ζητούμενες εκτιμήσεις για k = 10,3,30 ακολουθούν. Να σημειωθεί ότι η συνάρτηση που δημιουργεί τις γραφικές παραστάσεις αυτές βρίσκεται στο .ipynb αρχείο.



Probability Density Function calculation using KNN for k = 3



Probability Density Function calculation using KNN for k = 30



Η αύξηση του αριθμού των εγγύτερων γειτόνων (k) στον αλγόριθμο KNN όπως παρατηρούμε οδηγεί σε ένα μοντέλο λιγότερο ευαίσθητο στον θόρυβο. Καθώς δηλαδή το k αυξάνεται, αυξάνει και την ακρίβεια της εκτίμησης.

Γ) Σύμφωνα με τον κανόνα απόφασης του Bayes:

$$P(\omega_j|x) = \frac{p(x|\omega_j) * P(\omega_j)}{p(x)}$$

Προκειμένου να επιλέξουμε την κλάση ω1 αντί της ω2 ή της ω3 πρέπει:

$$P(\omega_1|x) \geq P(\omega_2|x) \; και \; P(\omega_1|x) \geq P(\omega_3|x)$$

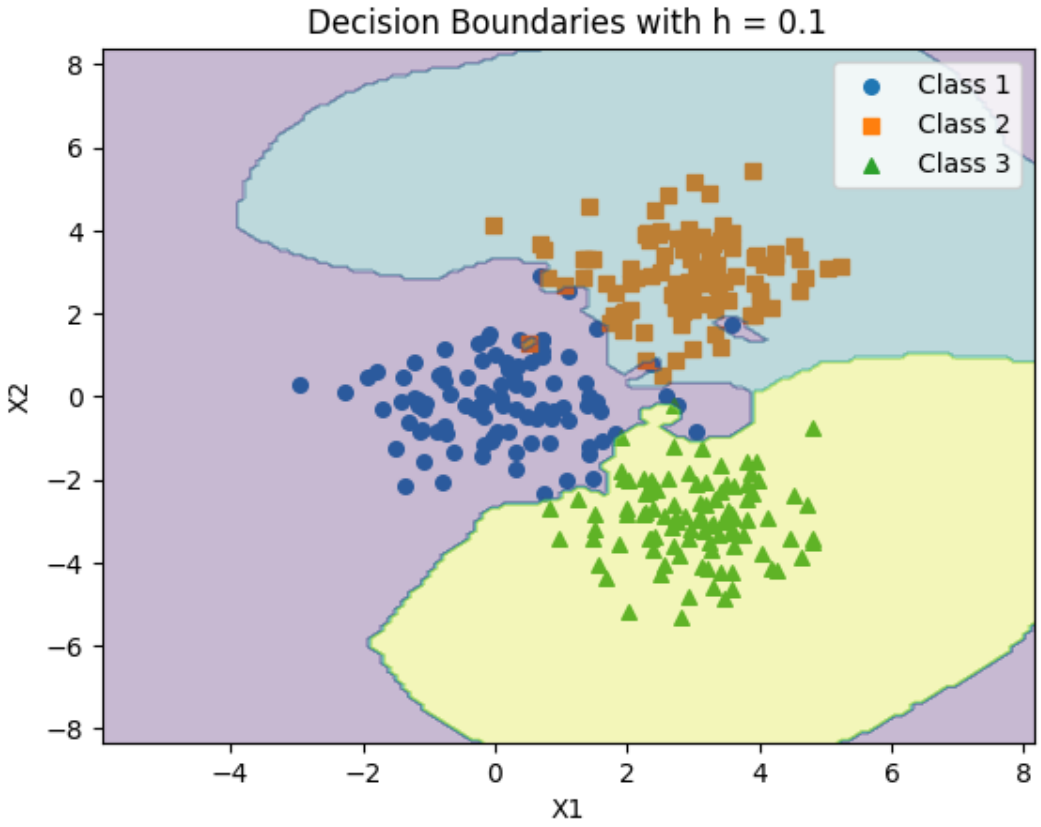
Επειδή έχουμε ότι οι a priori πιθανότητες είναι ίσες για όλες τις κλάσεις, το αν ανήκει στην κλάση ω1 εξαρτάται αποκλειστικά από την συνάρτηση πυκνότητας πιθανότητας $p(x|\omega_j)$.

Προκειμένου να βρεθούν οι περιοχές απόφασης, δημιουργούμε ένα grid που εξαρτάται από το min και το max των χαρακτηριστικών χ1 και χ2 και υπολογίζουμε την τιμή της συνάρτησης πυκνότητας πιθανότητας για όλα τα σημεία του grid με κάθε κλάση ξεχωριστά. Έτσι, με βάση τον παραπάνω συλλογισμό κάθε σημείο κατατάσσεται στην εκάστοτε κλάση ανάλογα με το ποια κλάση δίνει την μεγαλύτερη τιμή για την PDF. Όλα αυτά υλοποιούνται με την χρήση του παρακάτω κώδικα.

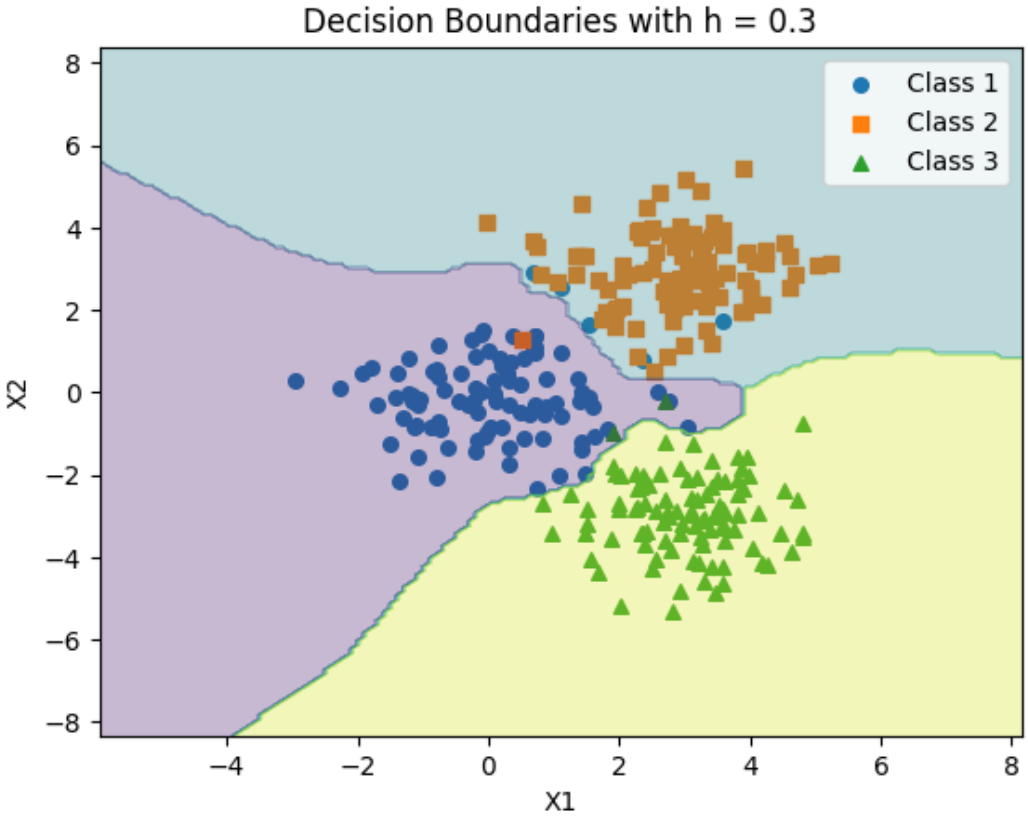
```
def parzen_plot_decision(data,h):
  #create a grid
  x_min, x_max = data[:, 0].min() - 3, data[:, 0].max() + 3
  y_min, y_max = data[:, 1].min() - 3, data[:, 1].max() + 3
  xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min,
y_max, 0.1))

  #find pdf for every point on the grid to separate regions
  Z = np.zeros_like(xx)
  for i in range(Z.shape[0]):
    for j in range(Z.shape[1]):
        point = np.array([xx[i, j], yy[i, j]])
        predictions = np.array([parzen_window(data[data[:, 2] == k,
0:2], h, point) for k in range(1, 4)])
        Z[i, j] = np.argmax(predictions)

  markers = ['o', 's', '^']

  #plot data from every class
  for class_idx in range(3):
    plt.scatter(data[data[:, 2] == class_idx + 1, 0],
                data[data[:, 2] == class_idx + 1, 1],
                label=f'Class {class_idx + 1}',
                marker=markers[class_idx], s=30)
```

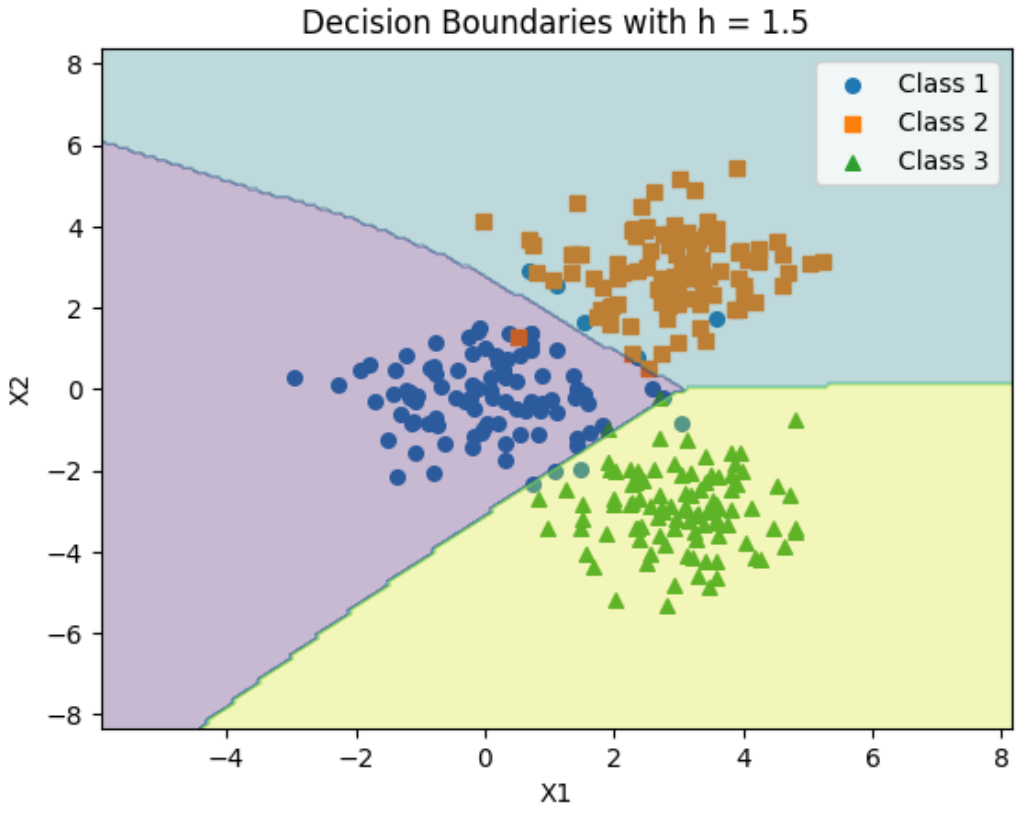
```
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title(f'Decision Boundaries with h = {h}')
plt.legend()
plt.show()
```

Έτσι, οι περιοχές απόφασης για τις διάφορες τιμές του h φαίνονται παρακάτω.

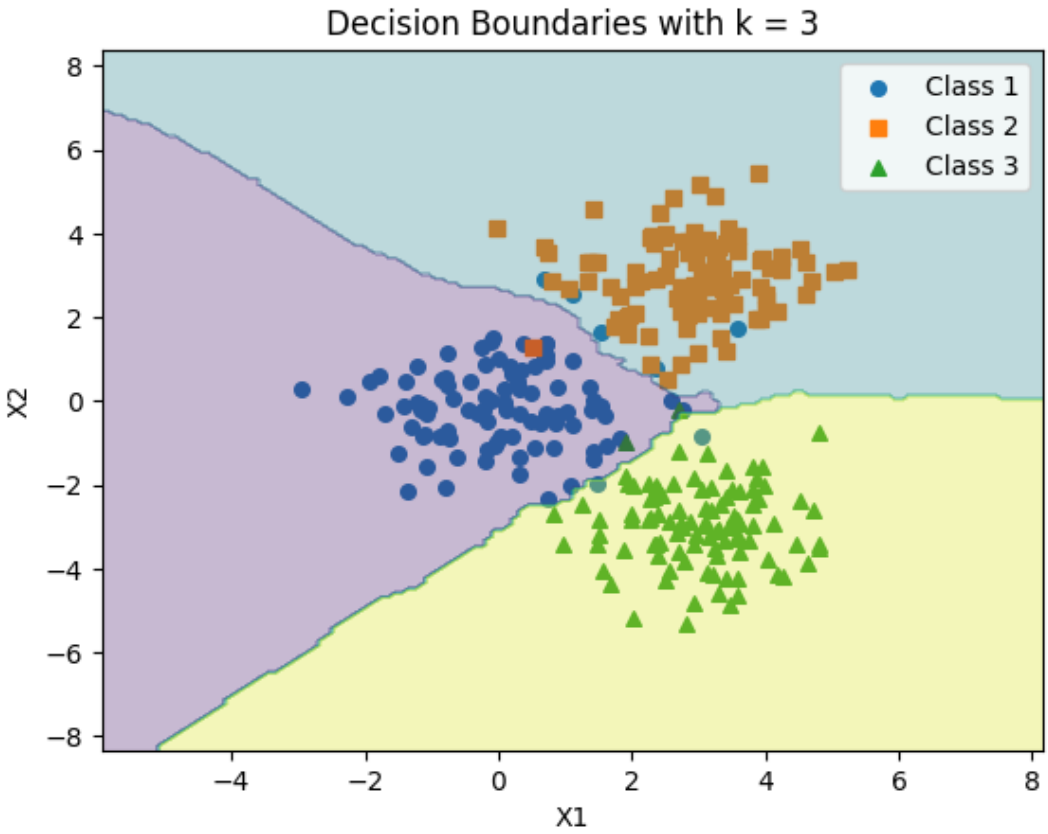


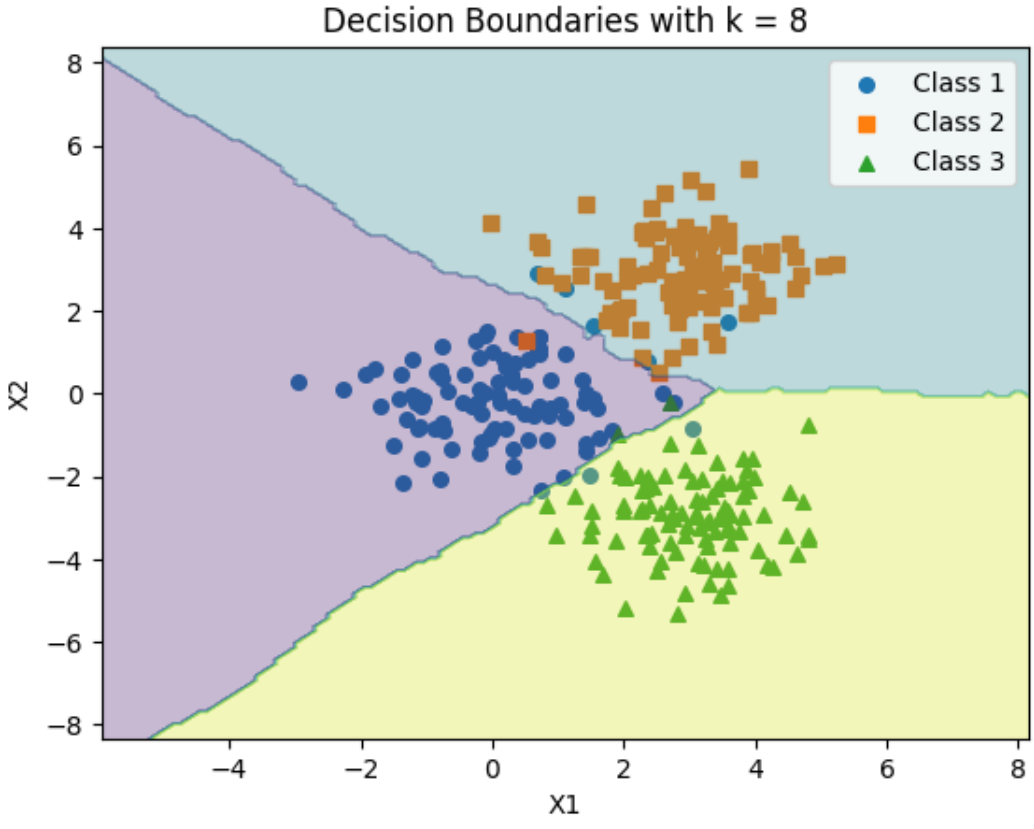
Όσο μικρότερο είναι το h παρατηρούμε φαινόμενο overfitting καθώς η περιοχή απόφασης φαίνεται να κάνει fit στα δεδομένα. Μία μεγάλη τιμή για το h κάνει πιο συμπαγή την περιοχή απόφασης με αποτέλεσμα να είναι πιο γενικευμένο το μοντέλο.

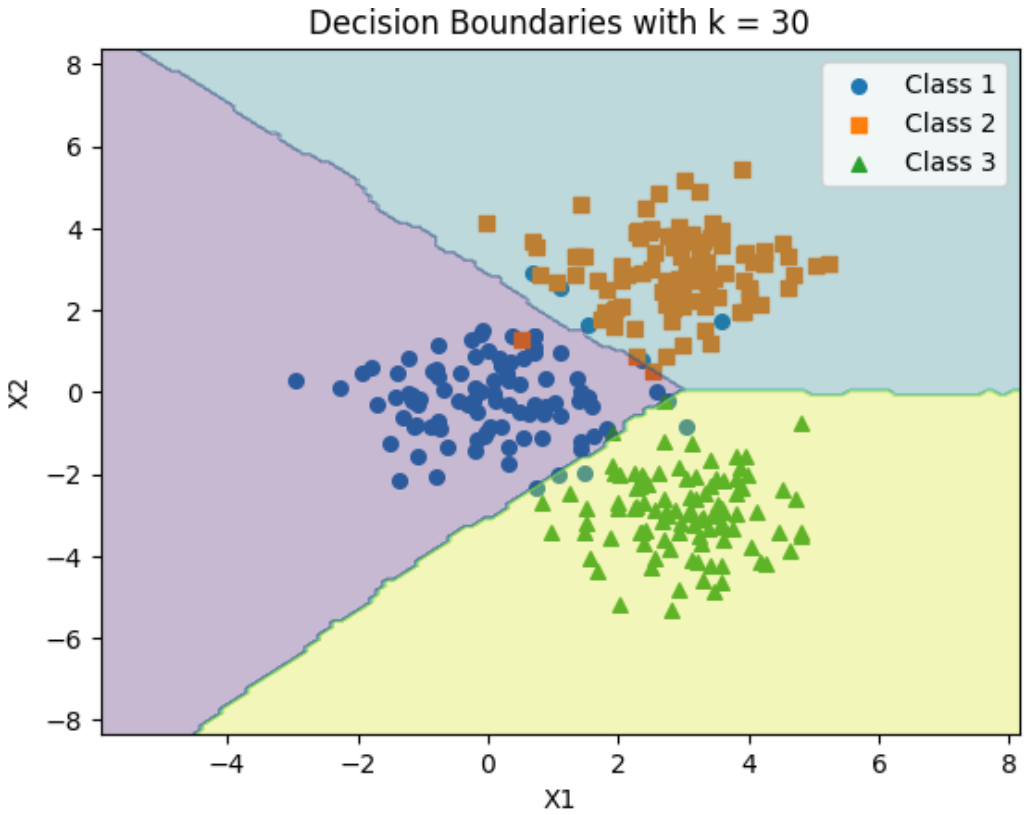

Decision Boundaries with h = 0.3
Class 1
Class 2
Class 3
X1
X2
8
6
4
2
0
−2
−4
−6
−8
−4
−2
0
2
4
6
8


Decision Boundaries with h = 1.5
Class 1
Class 2
Class 3
X1
X2
8
6
4
2
0
−2
−4
−6
−8
−4
−2
0
2
4
6
8

Δ) Αντίστοιχα με το ερώτημα γ) για να βρούμε τις περιοχές απόφασης δημιουργούμε ένα grid. Έπειτα υπολογίζουμε τις τιμές της συνάρτησης πυκνότητας πιθανότητας για κάθε σημείο και για τις 3 κλάσεις και κρατάμε την μεγαλύτερη. Η συνάρτηση για το plot υπάρχει στο notebook.





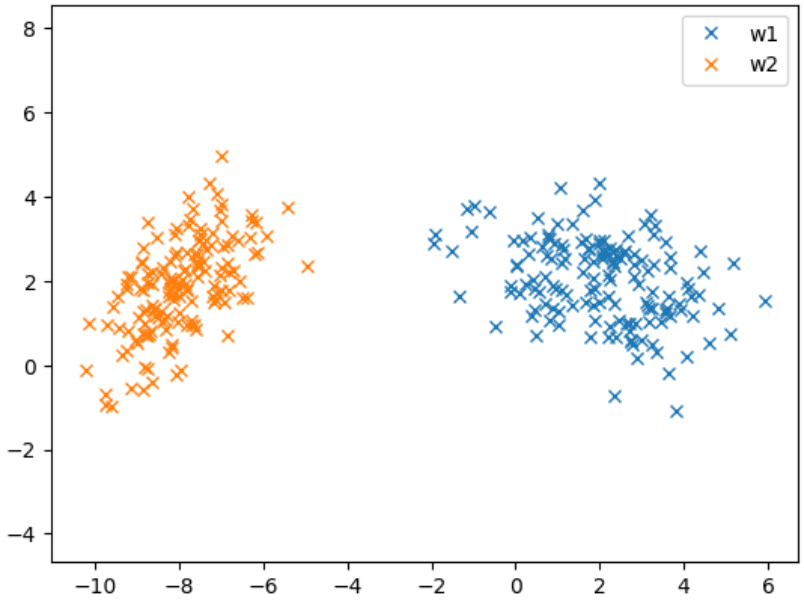


Παρατηρούμε πως όσο αυξάνουμε το k εξομαλύνονται τα όρια απόφασης. Αυτό έχει ως αποτέλεσμα την μείωση της επίδρασης του θορύβου. Επίσης, με την αύξηση του k έχουμε πιο σταθερές προβλέψεις αφού η απόφαση βασίζεται σε μεγαλύτερο αριθμό γειτόνων. Γενικά, όσο μικρότερη τιμή έχουμε στο k παρατηρούμε φαινόμενο overfitting.

E) Οι παραπάνω αλγόριθμοι έχουν βασικό πλεονέκτημα πως μπορούν να εφαρμοστούν σε δεδομένα από οποιαδήποτε κατανομή. Ο αλγόριθμος KNN είναι ένας απλός αλγόριθμος που μπορεί να πετύχει καλή ταξινόμηση εάν ο αριθμός των δειγμάτων είναι αρκετά μεγάλος. Από την άλλη, η χρήση του παραθύρου Parzen μπορεί να συγκλίνει (δηλαδή η εκτίμηση να είναι ακριβής) όσο ο αριθμός των δειγμάτων τείνει στο άπειρο. Στα μειονεκτήματα, οι δύο αυτές μέθοδοι είναι ακριβές υπολογιστικά γιατί για την εκτίμηση πιθανότητας κάθε σημείου πρέπει να χρησιμοποιηθούν πιθανώς πολλά δείγματα. Ένας ακόμη παράγοντας που καθιστά πρόβλημα είναι πως η επιλογή του h και του k αντίστοιχα δεν είναι πάντα εύκολη. Από την άλλη πλευρά, όσον αφορά τις γεωμετρικές τεχνικές ταξινόμησης (linear discriminants, SVMs) αυτές αποδίδουν καλύτερα και γρηγορότερα σε μεγάλα σύνολα δεδομένων. Επίσης είναι κατάλληλες για προβλήματα όπου οι κατανομές είναι γραμμικά διαχωρίσιμες. Στα μειονεκτήματα, οι δύο αυτές τεχνικές δεν αποδίδουν σε προβλήματα μη γραμμικής διαχωρισιμότητας.

Άσκηση 2:

Τα δεδομένα που δημιουργήθηκαν σύμφωνα με τα δεδομένα της εκφώνησης φαίνονται στο παρακάτω plot.



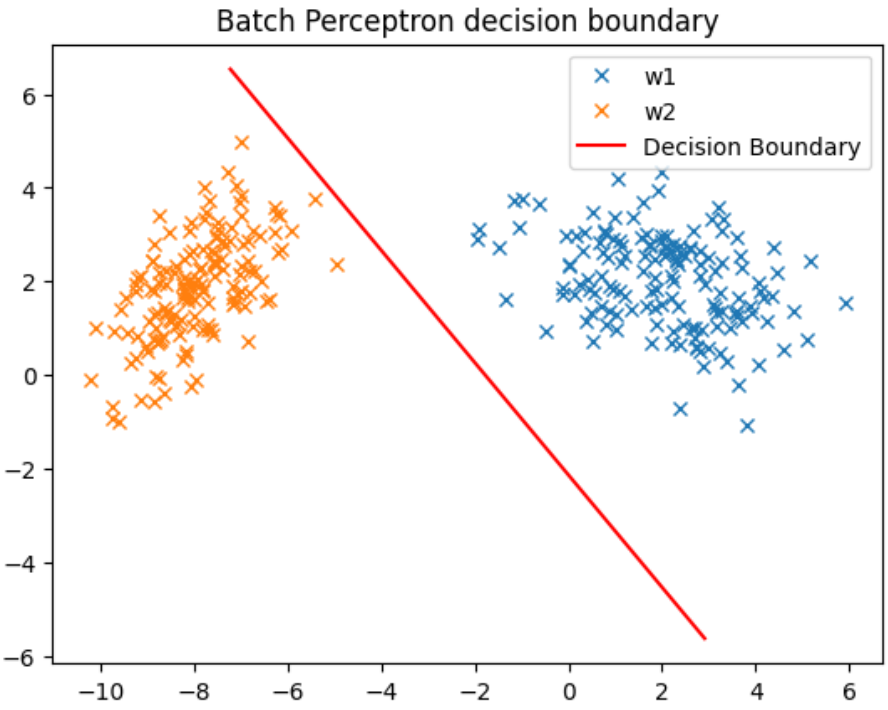
Α) Για τον αλγόριθμο Batch Perceptron, ο κώδικας βασίζεται σε αυτόν από τις διαφάνειες του μαθήματος που είναι σε MATLAB. Παρόλα αυτά με τις κατάλληλες αλλαγές ο κώδικας μετατρέπεται εύκολα σε Python. Ο συγκεκριμένος αλγόριθμος παίρνει ως ορίσματα τα δεδομένα των κλάσεων, τις ετικέτες (labels) με την μορφή 1 και -1 για την μια και για την άλλη κλάση αντίστοιχα ώστε να απλοποιηθεί η υλοποίηση. Επίσης λαμβάνει ως όρισμα τα αρχικά βάρη και τον αριθμό των εποχών που θα τρέξει. Κύριος στόχος του αλγορίθμου είναι να τροποποιήσει το διάνυσμα των βαρών προκειμένου να ελαχιστοποιηθεί το σφάλμα. Για τον λόγο αυτό χρησιμοποιείται η τεχνική του gradient descent. Η διαδικασία αυτή συμβαίνει έως ότου το κριτήριο τερματισμού ικανοποιηθεί δηλαδή ο αλγόριθμος να συγκλίνει ή φτάσουμε στον μέγιστο αριθμό εποχών.

Ακολουθεί η υλοποίηση του αλγορίθμου:

```
def batch_perceptron(data,labels,initial_weights,learning_rate,epochs):
  x,y = data.shape
  w = np.hstack((initial_weights,1)) #weight augmentation to add bias
  iter = 0
  mis_clas = 1
  data = np.hstack((data, np.ones((data.shape[0],1)))) #data
augmentation

  while iter < epochs and mis_clas > 0:
    iter = iter + 1
    mis_clas = 0
    gradi = np.zeros(y+1)
```

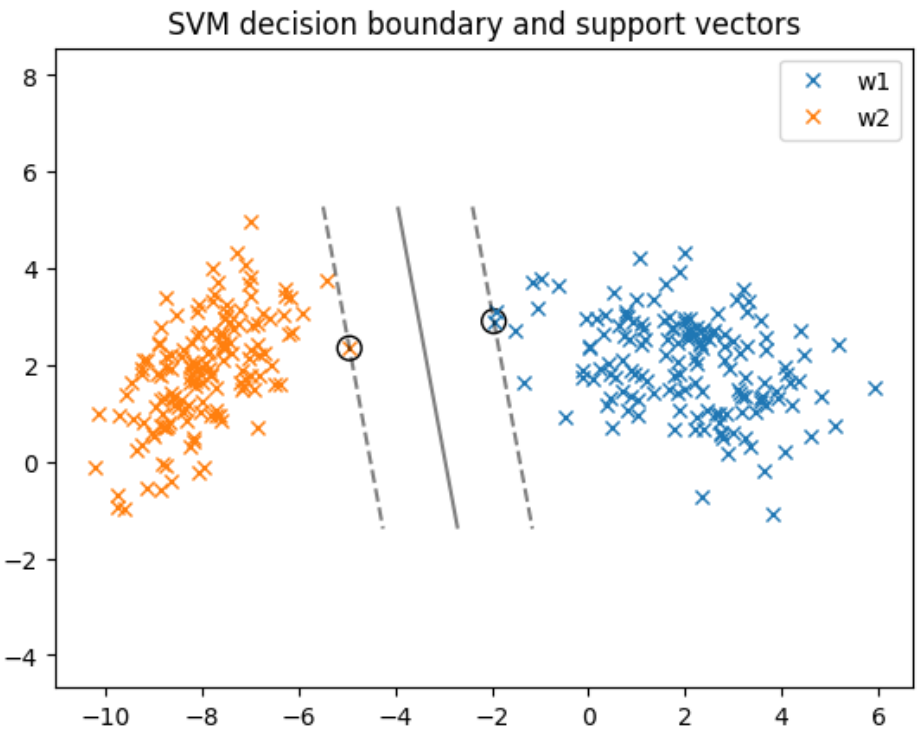
```
    for i in range(x):
      #check if the current sample is misclassified
      if(np.dot(data[i,:],w) * labels[i] < 0):
        mis_clas = mis_clas + 1 #increase misclassified counter
        gradi = gradi + learning_rate * (-labels[i]*data[i,:]) #update
the gradient vector
    w = w - learning_rate * gradi #change the weight vector based on
the gradient changes
  return w
```

Εκτελώντας τον αλγόριθμο για ένα τυχαίο seed και τα δεδομένα που παράχθηκαν στο προηγούμενο ερώτημα προκύπτει το παρακάτω διάγραμμα όπου φαίνεται η γραμμή απόφασης.



B) Τα Support Vectors Machines χρησιμοποιούνται ευρέως σε προβλήματα ταξινόμησης. Στόχος τους είναι να βρεθεί ένα υπερεπίπεδο στον χώρο που ορίζουν τα χαρακτηριστικά για να διαχωρίσει τις κλάσεις μεταξύ τους. Πιο συγκεκριμένα, τα SVMs επιδιώκουν να βρουν ένα υπερεπίπεδο που να έχει το μέγιστο δυνατό περιθώριο μεταξύ των κατηγοριών ενώ ταυτόχρονα προσπαθεί να ελαχιστοποιήσει το σφάλμα ταξινόμησης. Τα δεδομένα που βρίσκονται στα όρια των κατηγοριών ονομάζονται support vectors και είναι σημαντικά για τον προσδιορισμό του ζητούμενου υπερεπιπέδου.

Για την υλοποίηση αυτού του ερωτήματος χρησιμοποιήθηκε η βιβλιοθήκη sklearn. Ο κώδικας και το plot με την επιφάνεια απόφασης και τα support vectors ακολουθεί.




```
from sklearn import svm

#create and train a linear SVM
svm = svm.SVC(kernel='linear')
svm.fit(data, labels)
sup_vec = svm.support_vectors_

#plot data for class w1 and w2
plt.plot(w1[:, 0], w1[:, 1], 'x', label='w1')
plt.plot(w2[:, 0], w2[:, 1], 'x', label='w2')
plt.legend()

#get axis limits
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

#create a grid
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50),
np.linspace(ylim[0], ylim[1], 50))

#compute decision boundary values
Z = svm.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

#plot decision boundary
```

```
plt.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
linestyles=['--', '-', '--'])
plt.title('SVM decision boundary and support vectors')

#plot support vectors
plt.scatter(sup_vec[:, 0], sup_vec[:, 1], s=100, facecolors='none',
edgecolors='k')
plt.axis('equal')
plt.show()

#print support vectors
print(f"Support vectors are: {sup_vec}")
```

```
Support vectors are: [[-4.95100295  2.34975656]
 [-1.96116702  2.90695364]]
```

Γ) Αρχικά παρατηρούμε πως με την χρήση γραμμικού SVM γίνεται καλύτερος διαχωρισμός των κλάσεων. Πιο συγκεκριμένα, το επίπεδο απόφασης είναι μοιρασμένο μεταξύ των ακραίων σημείων (support vectors) της κάθε κλάσης έτσι ώστε να αφήνει ένα περιθώριο (το μέγιστο δυνατό). Εν αντιθέσει, η χρήση του batch perceptron πετυχαίνει και αυτή σωστό διαχωρισμό όμως το επίπεδο απόφασης συγκλίνει πολύ κοντά σε κάποια κλάση. Γενικά, ένας καλός ταξινομητής πρέπει να δημιουργεί σύνορο που απέχει πολύ από τα δεδομένα. Η διαφορά αυτή οφείλεται στο γεγονός πως τα SVMs είναι λιγότερο ευαίσθητα όταν γίνεται σφάλμα ταξινόμησης καθώς επιτρέπουν κάποιο περιθώριο σφάλματος με κάποια ανοχή αφού έχουν ως πρωταρχικό σκοπό την μεγιστοποίηση του margin μεταξύ των δύο κλάσεων. Ο αλγόριθμος batch perceptron προσπαθεί να ελαχιστοποιήσει τα σφάλματα ταξινόμησης αλλάζοντας απευθείας τα βάρη βασιζόμενος στα ίδια τα σφάλματα.

Άσκηση 3:

Α) Η ζητούμενη επεξεργασία των δεδομένων έχει γίνει με την χρήση του κώδικα που ακολουθεί:

```
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Specify the file path
file_path1 = '/content/wine.data'

#store data on an array
data = np.genfromtxt(file_path1, delimiter=',')

#data seperation
class1_data = data[0:59,:]
class2_data = data[59:130,:]
class3_data = data[130:,:]
```

```
sub_class1 = class1_data[:,1:6]
sub_class2 = class2_data[:,1:6]
sub_class3 = class3_data[:,1:6]

#labels correspond to the data we will use
sub_labels = np.concatenate((class2_data[:,0],class3_data[:,0]),axis =
0)

#array with total data
tot_sub_class = np.concatenate((sub_class2,sub_class3), axis = 0)

#dataset split into train,validation and test set
#stratify argument ensures that every set contains same number of
samples from each class
X_train, X_temp, y_train, y_temp = train_test_split(tot_sub_class,
sub_labels, test_size=0.5, stratify=sub_labels,random_state=25)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, stratify=y_temp,random_state=25)

print(f"Train set shape: Data: {X_train.shape}, Labels:
{y_train.shape}")
print(f"Validation set shape: Data: {X_val.shape}, Labels:
{y_val.shape}")
print(f"Test set shape: Data: {X_test.shape}, Labels: {y_test.shape}")
```

Αξίζει να σημειωθεί πως έχει δοθεί random seed μέσω της παραμέτρου random_state.

B) Αρχικά δημιουργούμε έναν SVM με linear kernel. Έπειτα εφαρμόζουμε τις διαφορετικές τιμές του C τις οποίες ορίσαμε προκειμένου να βρούμε χρησιμοποιώντας το validation set ποια είναι η καλύτερη τιμή. Αυτό υλοποιείται με την συνάρτηση της βιβλιοθήκης sklearn, GridSearchCV(). Στην συνέχεια για το καλύτερο C που βρέθηκε, εκπαιδεύουμε ένα SVM με τα training data και κάνουμε predict χρησιμοποιώντας το test set. Η ακρίβεια μπορεί εύκολα να υπολογιστεί από την συνάρτηση accuracy_score() και το σφάλμα γνωρίζουμε πως είναι error = 1 – accuracy. Ακολουθεί ο κώδικας που υλοποιεί τα όσα περιεγράφηκαν καθώς και η έξοδος του.

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

#create an linear SVM
svm_1 = svm.SVC(kernel='linear')

#define different values of c to find the best
```

```
c_val = {'C': [0.001, 0.01, 0.1, 1, 10, 100,1000]}

#search for the best c value in validation set
grid_search = GridSearchCV(svm_1, c_val)
grid_search.fit(X_val, y_val)

#define the best c found
best_C = grid_search.best_params_['C']

#use the best c to train the model in training set
svm_2 = svm.SVC(kernel='linear', C=best_C)
svm_2.fit(X_train,y_train)

#use the model to predict in test set
y_pred = svm_2.predict(X_test)

#model accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set is: {accuracy} so error is {1-
accuracy}")
```

```
Accuracy on the test set is: 0.9333333333333333 so error is 0.06666666666666665
```

Γ) Επαναλαμβάνοντας για 5 τυχαίους διαμερισμούς δεδομένων λαμβάνουμε τα ακόλουθα αποτελέσματα (ο κώδικας για λόγους έκτασης βρίσκεται στο αρχείο .ipynb)

```
Classification error in each case is: [0.167, 0.233, 0.133, 0.167, 0.267]
Mean classification error: 0.19340000000000002
Standard deviation of classification errors: 0.04904528519643861
```

Δ) Για μη γραμμικούς SVM επαναλαμβάνοντας το ερώτημα γ) έχουμε:

Τα αποτελέσματα από τις 5 εκτελέσεις ακολουθούν:

```
Using poly kernel:

Classification error in each case is: [0.1        0.13333333 0.13333333 0.13333333 0.1        ]
Mean classification error: 0.11999999999999997
Standard deviation of classification errors: 0.016329931618554516

Using rbf kernel:

Classification error in each case is: [0.06666667 0.13333333 0.13333333 0.23333333 0.33333333]
Mean classification error: 0.18
Standard deviation of classification errors: 0.09333333333333335

Using sigmoid kernel:

Classification error in each case is: [0.4        0.4        0.4        0.63333333 0.4        ]
Mean classification error: 0.44666666666666666
Standard deviation of classification errors: 0.0933333333333333
```

```
Using linear kernel:

Classification error in each case is: [0.2        0.16666667 0.13333333 0.13333333 0.1        ]
Mean classification error: 0.14666666666666664
Standard deviation of classification errors: 0.03399346342395189
```

Γενικά, από εκτέλεση σε εκτέλεση το μέσο σφάλμα ταξινόμησης αλλάζει. Όμως παρατηρούμε πως οι τιμές που δίνει ο polyonimal πυρήνας είναι καλύτερες από τους άλλους πυρήνες. Για τον sigmoid kernel το μέσο σφάλμα είναι αρκετά μεγάλο με αποτέλεσμα να μην είναι ιδανικός για αυτήν την ταξινόμηση. Έτσι, βάση της μέσης τιμής του σφάλματος και της τυπικής απόκλισης ο καλύτερος ταξινομητής για το πρόβλημα είναι non linear SVM με polyonimal kernel. Το γεγονός ότι ο πολυωνυμικός πυρήνας έδωσε καλύτερα αποτελέσματα ταξινόμησης πιθανώς δηλώνει πως τα δεδομένα έχουν πολυωνυμική δομή δηλαδή η σχέση μεταξύ των χαρακτηριστικών είναι πολυωνυμική.

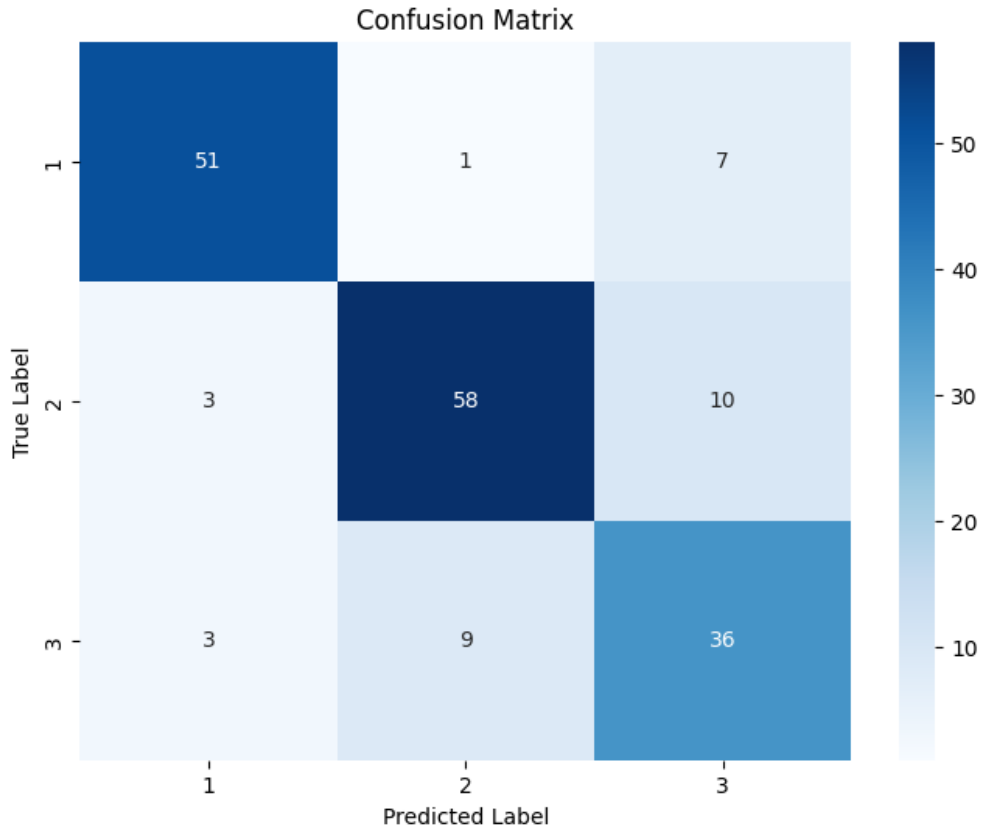
Ε) Για όλες τις κλάσεις με τα πέντε πρώτα χαρακτηριστικά ο κώδικας και ο πίνακας σύγχυσης ακολουθούν:

```
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import confusion_matrix
import seaborn as sns
#for the first 5 characteristics for all classes

data = np.genfromtxt(file_path1, delimiter=',')

#change the dataset to include only the required characteristics
class1_data = data[0:59, :]
class2_data = data[59:130, :]
class3_data = data[130:, :]

sub_class1 = class1_data[:, 1:6]
sub_class2 = class2_data[:, 1:6]
sub_class3 = class3_data[:, 1:6]

sub_labels = np.concatenate((class1_data[:, 0], class2_data[:, 0],
class3_data[:, 0]), axis=0)

tot_sub_class = np.concatenate((sub_class1, sub_class2, sub_class3),
axis=0)

#create a linear svm classifier with C = 1 and choose one vs one
decision function
#with argument value 'ovo'
svm = svm.SVC(kernel='linear', C=1, decision_function_shape='ovo')

#5 fold cross validation to find accuracy
cv_scores = cross_val_score(svm, tot_sub_class, sub_labels, cv = 5,
scoring='accuracy')
```

```
mean_error = 1 - np.mean(cv_scores)
print("Mean error:", mean_error)

#using 5 fold cross validation protocol to predict the labels
predicted_labels = cross_val_predict(svm, tot_sub_class, sub_labels,
cv=5)

#calculate confusion matrix using labels and predicted labels
conf_matrix = confusion_matrix(sub_labels, predicted_labels)

#plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=[1, 2, 3], yticklabels=[1, 2, 3])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

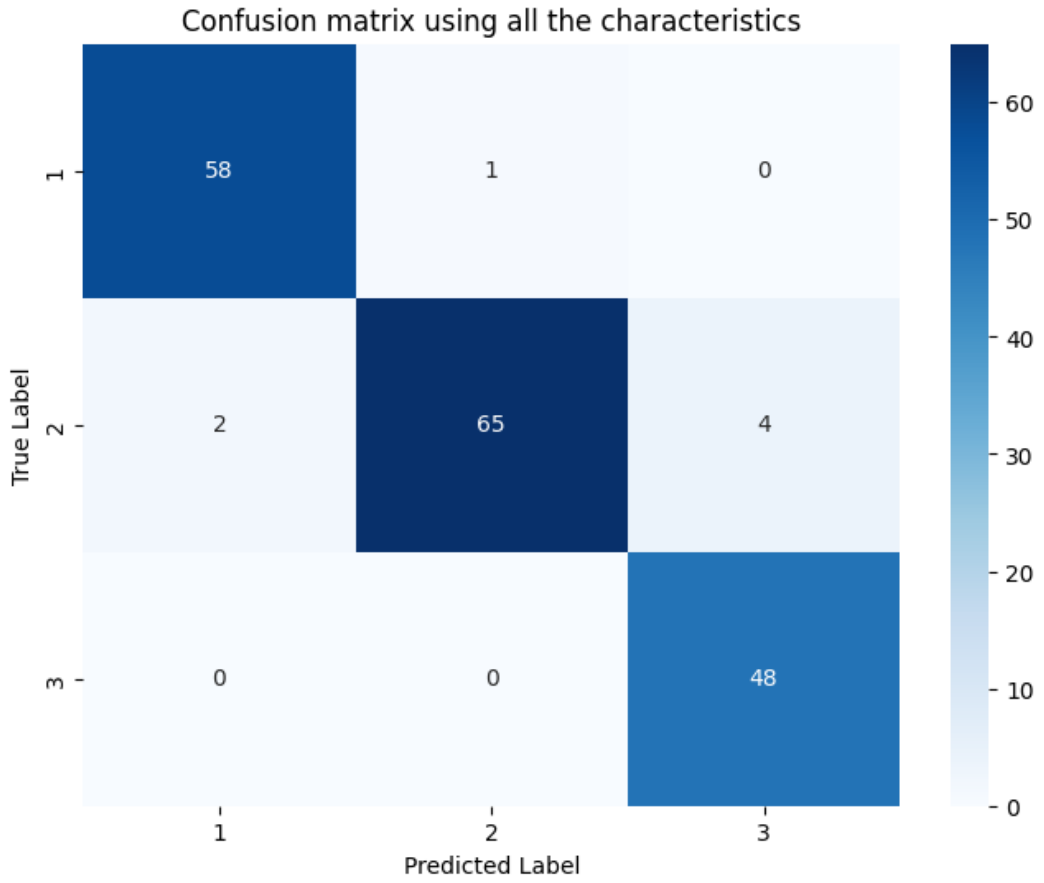
Το μέσο σφάλμα είναι:

```
Mean error: 0.1852380952380953
```



Για όλα τα χαρακτηριστικά και όλες τις κλάσεις το μέσο σφάλμα είναι:

```
Mean error: 0.03888888888888897
```



Παρατηρούμε πως όταν τρέχουμε τον SVM για 5 χαρακτηριστικά οι κλάσεις 2 και 3 ομοιάζουν περισσότερο. Αυτό το συμπεραίνουμε από τον πίνακα σύγχυσης αφού πλήθος δειγμάτων της κλάσης 2 ταξινομούνται στην κλάση 3 και αντίστροφα. Από την άλλη πλευρά, όταν έχουμε εισάγει στον SVM όλα τα χαρακτηριστικά των κλάσεων παρατηρούμε πολύ μικρή μέση τιμή σφάλματος ταξινόμησης και από τον πίνακα δεν παρατηρούμε κάποια υψηλή σύγχυση μεταξύ κλάσεων. Είναι προφανές πως η αξιοποίηση περισσότερων χαρακτηριστικών μείωσε το σφάλμα ταξινόμησης. Αξίζει επίσης να σημειωθεί πως η τεχνική 5-fold cross validation είναι πιο αξιόπιστη από τον απλό διαμοιρασμό σε train,validation και test. Εξασφαλίζει ότι το μοντέλο εκπαιδεύεται και δοκιμάζετε σε διαφορετικά δεδομένα του dataset πράγμα το οποίο επηρεάζει το πόσο καλή απόδοση έχει σε νέα άγνωστα δεδομένα.