8Η ΕΡΓΑΣΙΑ

Στο μάθημα:
«Αναγνώριση Προτύπων»
Καθηγητής: Νικόλαος Μητιανούδης

Στυλιανός Μούσλεχ
ΑΜ:57382
21/2/2021 ,Ξάνθη

ΕΙΣΑΓΩΓΗ

Η υλοποίηση του k-means έγινε με την χρήση της βιβλιοθήκης sklearn. Η υλοποίηση για τον fuzzy c-means έγινε μέσω της βιβλιοθήκης skfuzzy. Η υλοποίηση του ISODATA έγινε σε MATLAB με βάση τον κώδικα «isodata_ND.m» που μας δίνεται στο eclass.

Η πρώτη άσκηση υλοποιείται στο αρχείο “Ex8_1.py”

Η Δεύτερη άσκηση υλοποιείται στο αρχείο “Ex8_2.py”

Η υλοποίηση στο MATLAB του ISODATA γίνεται στα αρχεία που βρίσκονται στο φάκελο «matlab_isodata»

Επιπλέον ήθελα να δοκιμάσω να δω το elbow method στο SSE για την εύρεση του k στους αλγορίθμους k-means και fuzzy c-means καθώς πολλές φορές αυτό δεν είναι προ υπάρχουσα γνώση σε clustering προβλήματα. Αυτό περιγράφεται μετά τις 3 ασκήσεις και ο κώδικας υπάρχει στο αρχείο Ex8_extra.py

Άσκηση 1

Στην πρώτη άσκηση θα χρησιμοποιήσουμε τον αλγόριθμο k-means:

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

Φυσικά καθώς στο συγκεκριμένο dataset γνωρίζουμε από πριν ότι θέλουμε 3 clusters μπορούμε κατευθείαν να ορίσουμε αυτή την υπερπαράμετρο. Οπότε φορτώνουμε τα δεδομένα μας και καλούμε τον k-means.

```
iris=load_iris()
x=iris.data
y=iris.target

kmeans = KMeans(n_clusters = 3, init = 'random', max_iter = 500, n_init = 10, random_state =
0).fit(x)
y_kmeans_og = kmeans.labels_
print(y_kmeans_og)
```

Εδώ είναι σημαντικό να σχολιάσουμε και τις υπόλοιπες παραμέτρους και τις τιμές τους

Στην n_init=10 ουσιαστικά θα τρέξουμε τον αλγόριθμο 10 φορές με διαφορετικά κέντρα και θα κρατήσουμε το καλύτερο SSE (Sum of Squared errors).Όπου error ορίζουμε την απόσταση του σημείου από το κοντινότερο κέντρο

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)$$

Αυτή η επανάληψη με διαφορετικά κέντρα είναι πολύ σημαντική καθώς η ποιότητα το αποτελέσματος βασίζεται αρκετά στην επιλογή των κέντρων αυτών.

Φυσικά θα μπορούσαμε να κάνουμε μια πιο «έξυπνη» επιλογή κέντρων με την χρήση του αλγορίθμου kmeans++ (init = 'k-means++' αντί για init = 'random') αλλά στο συγκεκριμένο παράδειγμα καταλήγουμε εύκολα στα ίδια αποτελέσματα.

Αναφορικά με τα κριτήρια τερματισμού έχουμε max επαναλήψεις 500 και το default tolerance για την αλλαγή των κέντρων : 1e-4

Τέλος με το random_state δίνουμε ένα συγκεκριμένο seed στις τυχαίες επιλογές ώστε να έχουν επαναληψημότητα τα αποτελέσματα που βγάζουμε.

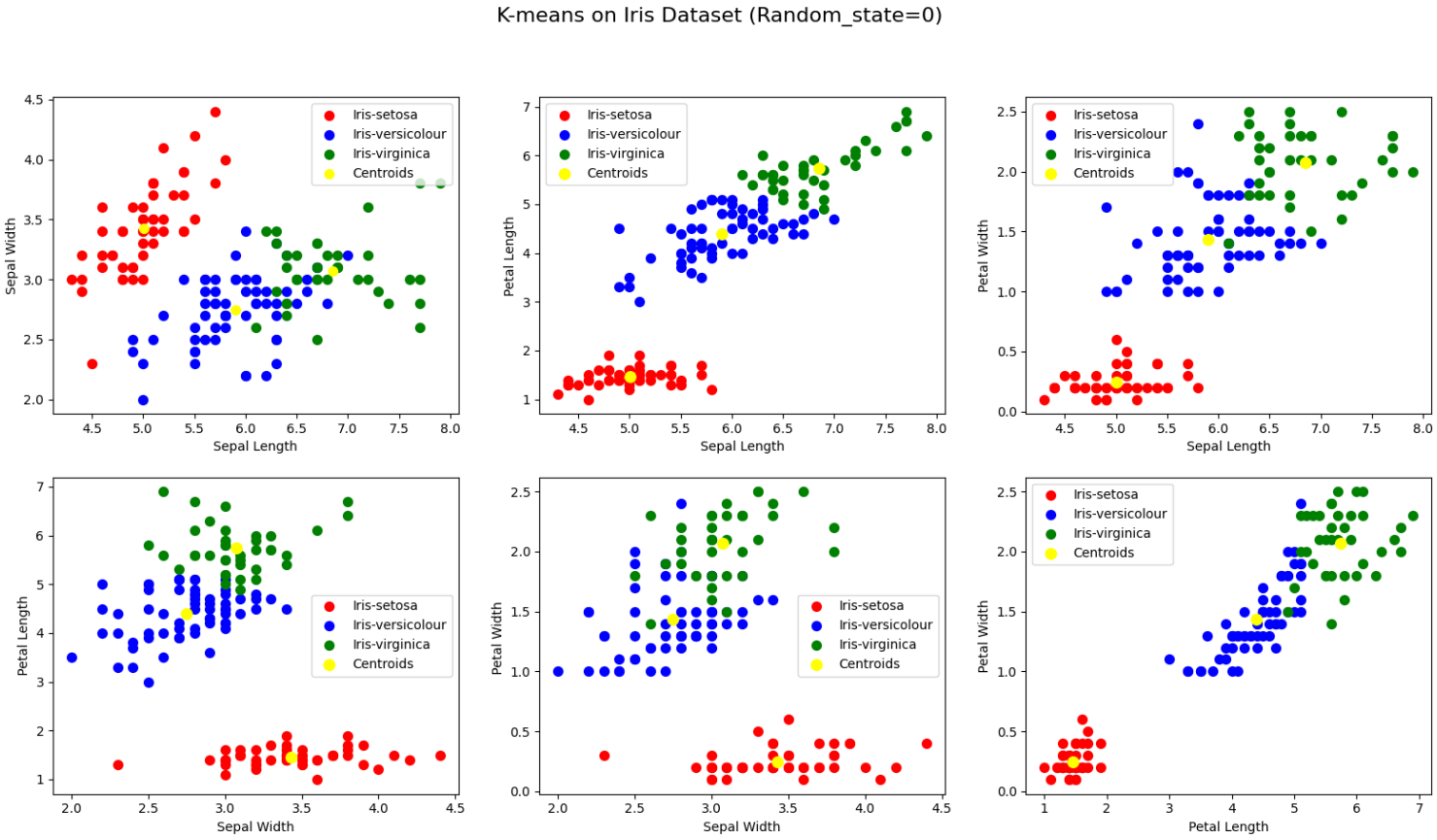
Για να βρούμε το error εδώ θα κάνουμε μια απλή αντικατάσταση των labels έτσι ώστε η κάθε ομάδα να έχει το ίδιο νούμερο με το πραγματικό . Φυσικά αυτό έχει νόημα μόνο για τον δικό μας υπολογισμό καθώς ο αλγόριθμος κάνει ομαδοποίηση με βάση τα χαρακτηριστικά χωρίς να διαλέγει με κάποιο συγκεκριμένο νούμερο την κάθε ομάδα.. Οπότε με βάση την αρχική ομαδοποίηση κάνουμε κάποιες αντικαταστάσεις και υπολογίζουμε το error καθώς και το SSE.

```
#K-means labels are different from ours we change it for visual reasons only and error calculation since we know classes
y_kmeans=np.zeros(y_kmeans_og.shape)
y_kmeans[y_kmeans_og==2]=0
y_kmeans[y_kmeans_og==0]=1
y_kmeans[y_kmeans_og==1]=2
```

Έχουμε τα παρακάτω αποτελέσματα:

```
Error rate of K-means on Iris Dataset is  0.10666666666666667
SSE is  78.851441426146
```

Τέλος κάνουμε plot τα clusters και τα κέντρα (με όλα τα πιθανά ζεύγη χαρακτηριστικών).



Άσκηση 2

Εδώ τώρα θα ακολουθήσουμε την ίδια διαδικασία αλλά χρησιμοποιώντας τον αλγόριθμο fuzzy c-means.

Ο αλγόριθμος περιγράφεται παρακάτω όπως τον είδαμε στο μάθημα

- Choose a number of clusters.
- Assign coefficients randomly to each data point for being in the clusters.
- Repeat until the algorithm has converged (that is, the coefficients' change between two iterations is no more than ε, the given sensitivity threshold) :
  - Compute the weighted centroid for each cluster (shown below).

$$c_k = \frac{\sum_x u_k(x)^m x}{\sum_x u_k(x)^m}$$

where m is the hyper- parameter that controls how fuzzy the cluster will be. The higher it is, the fuzzier the cluster will be in the end.

  - For each data point, compute its coefficients of being in the clusters.

Η συνάρτηση cmeans της βιβλιοθήκης skfuzzy ορίζεται ως εξής από το documentation:

skfuzzy.**cmeans**(*data*, *c*, *m*, *error*, *maxiter*, *init=None*, *seed=None*)[source]

Fuzzy c-means clustering algorithm [1].

Parameters:

data : 2d array, size (S, N)
Data to be clustered. N is the number of data sets; S is the number of features within each sample vector.

c : int
Desired number of clusters or classes.

m : float
Array exponentiation applied to the membership function u_old at each iteration, where U_new = u_old ** m.

error : float
Stopping criterion; stop early if the norm of (u[p] - u[p-1]) < error.

maxiter : int
Maximum number of iterations allowed.

init : 2d array, size (S, N)
Initial fuzzy c-partitioned matrix. If none provided, algorithm is randomly initialized.

seed : int
If provided, sets random seed of init. No effect if init is provided. Mainly for debug/testing purposes.

Returns:

cntr : 2d array, size (S, c)
Cluster centers. Data for each center along each feature provided for every cluster (of the c requested clusters).

u : 2d array, (S, N)
Final fuzzy c-partitioned matrix.

u0 : 2d array, (S, N)
Initial guess at fuzzy c-partitioned matrix (either provided init or random guess used if init was not provided).

d : 2d array, (S, N)
Final Euclidian distance matrix.

jm : 1d array, length P
Objective function history.

p : int
Number of iterations run.

fpc : float
Final fuzzy partition coefficient.

Οπότε:

```
iris=load_iris()
x=iris.data
y=iris.target

cntr, u, u0, distance_matrix, jm, p, fpc = fuzz.cluster.cmeans(x.T, 3, 2, error=0.005, maxiter=1000, seed=1998)
```

(Η τιμή του m=2 προέκυψε ως αυτή που μας έδινε το μικρότερο error)

Φυσικά εδώ έχουμε για κάθε δεδομένο μια τιμή που μας λέει κατά πόσο ανήκει στο κάθε cluster. Οπότε στο τέλος θα κρατήσουμε το max για να κάνουμε label τα δεδομένα μας:

```
cluster_membership = np.argmax(u,axis=0)
```

Όπως και πριν θα κάνουμε την αλλαγή των labels για τον υπολογισμό του error:

```
#change our labels for correct prediction testing

y_cmeans=np.zeros(cluster_membership.shape)
y_cmeans[cluster_membership == 2]=0
y_cmeans[cluster_membership == 0]=1
y_cmeans[cluster_membership == 1]=2
#calculate error
errors=(np.where(y_cmeans == y, 0, 1)).sum()
error_rate=errors/y.shape[0]
print("Error rate of Fuzzy c-means on Iris Dataset is ",error_rate)
```

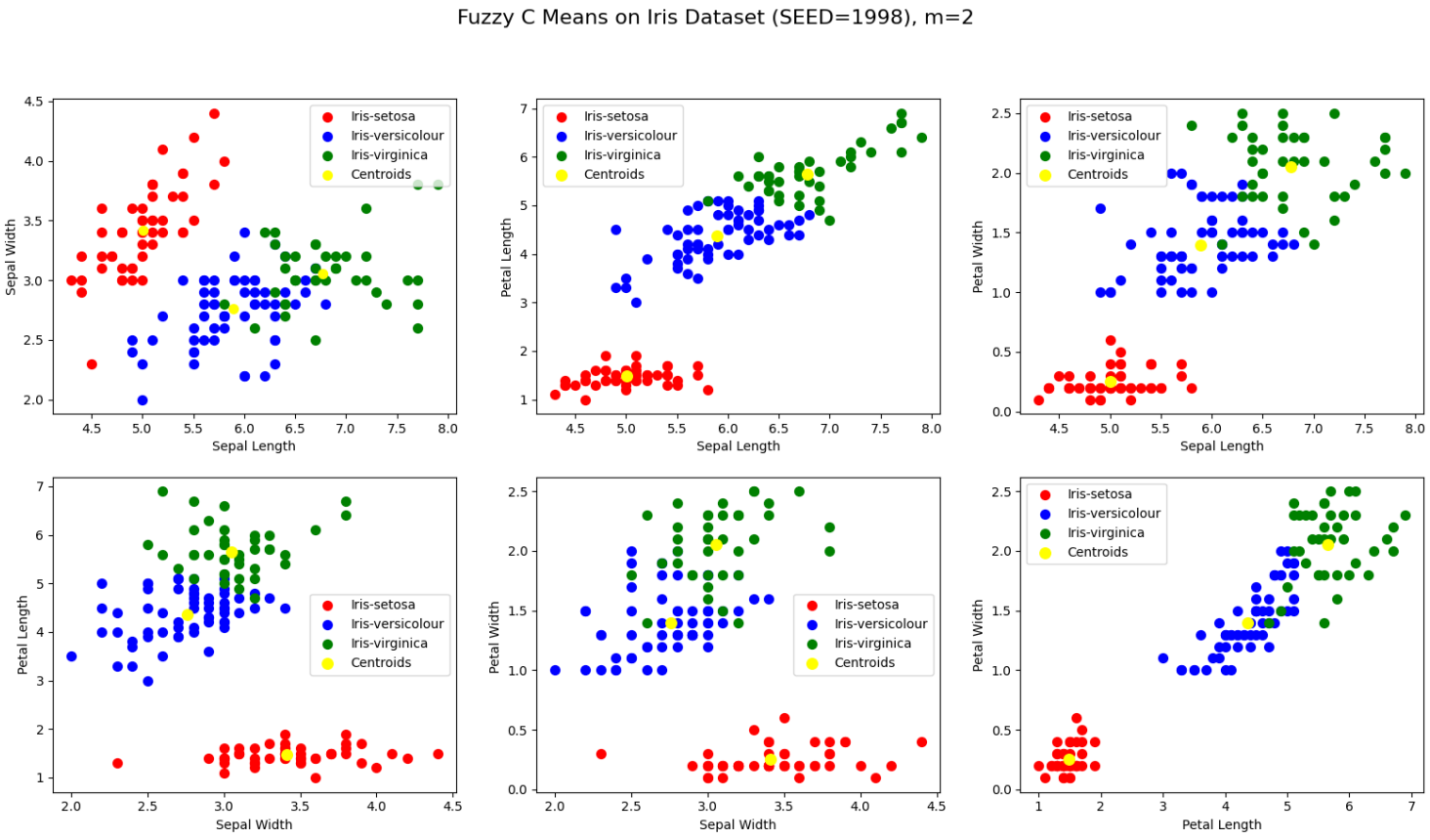
Χρησιμοποιώντας το distance_matrix θα υπολογίσουμε το sse αυτή το φορά μόνοι μας:

```
#sse calculation
sse=0
for i in range(x.shape[0]):
   sse=sse + distance_matrix[cluster_membership[i],i]**2
print("SSE is ",sse)
```

Και έχουμε τα παρακάτω αποτελέσματα:

```
Error rate of Fuzzy c-means on Iris Dataset is  0.10666666666666667
SSE is  79.36785904291425
```

Κάνουμε plot όπως και πριν:



Εδώ είχαμε σχεδόν τα ίδια αποτελέσματα, βέβαια χρειάστηκε περισσότερο tuning των παραμέτρων για να φτάσουμε σε αυτό το αποτέλεσμα.

Άσκηση 3

Στη συγκεκριμένη υλοποίηση χρησιμοποιήθηκε ο αλγόριθμος σε MATLAB που μας δόθηκε στο book software στο eclass. Εδώ θα χρησιμοποιήσουμε τον αλγόριθμο isodata :

1. Εκτέλεσε k-means clustering
2. Υπολόγισε την τυπική απόκλιση μέσα σε κάθε cluster, και τις αποστάσεις μεταξύ των κέντρων των clusters
3. Διάσπασε όσα clusters έχουν αρκετά ανόμοια δεδομένα (μεγάλες τυπικές αποκλίσεις)
4. Ένωσε όσα clusters έχουν αρκετά όμοια δεδομένα, εάν η απόσταση τους είναι μικρότερη από κάποιο προκαθορισμένο κατώφλι
5. Βήμα 1

```
[centro, Xcluster, A, clusters]=isodata_ND(patterns', k, L, I, ON, OC, OS,
NO, min_dist);

fprintf('Number of Clusters: %d\n',A);
counter = zeros(3,3);
k= zeros(1,3);
for i=0:2
    for j=1:50
        counter(i+1,clusters(i*50 + j))= counter(i+1,clusters(i*50 + j)) +
1;
    end
    %[~,k(i+1)] = max(counter);
end

[maxes,id] = max(counter);
maxid = sortrows([maxes; [1 2 3]; id]',-1);

clustersfixed=zeros(150,1);
for i=1:3
    clustersfixed(clusters==i) = id(i) - 1;
end

error_rate=nnz(clustersfixed'~=targets)/150;
fprintf('Error rate is: %d\n',error_rate);
```

Τρέχοντας τον αλγόριθμο είχαμε τα παρακάτω αποτελέσματα :

```
Number of Clusters: 3
Error rate is: 2.133333e-01
>>
```

Εδώ αξίζει να σημειώσουμε ότι χρειάστηκε αρκετή παραμετροποίηση για να φτάσουμε να έχουμε 3 clusters καθώς και το error είναι χειρότερο σε σχέση με τις προηγούμενες υλοποιήσεις. Αυτό συμβαίνει καθώς δυσκολεύεται αρκετά στις 2 μη γραμμικά διαχωρίσιμες κλάσεις.

Extra: Εύρεση Βέλτιστου k

Σε πολλά clustering προβλήματα δεν γνωρίζουμε εξαρχής το ιδανικό k που θέλουμε. Ένας τρόπος για να προσπαθήσουμε το βρούμε είναι κάνοντας χρήση του “elbow method”. Σε αυτή τη μέθοδο υπολογίζουμε το SSE για πολλά k και κάνοντας plot τις τιμές του SSE για κάθε k διαλέγουμε το k που μετά από αυτό ξεκινάει να πέφτει το SSE γραμμικά αυξάνοντας το k (elbow).

Δοκιμαστικά έκανα το ίδιο για το Iris dataset για να δούμε αν προκύπτει και από εδώ το k=3.

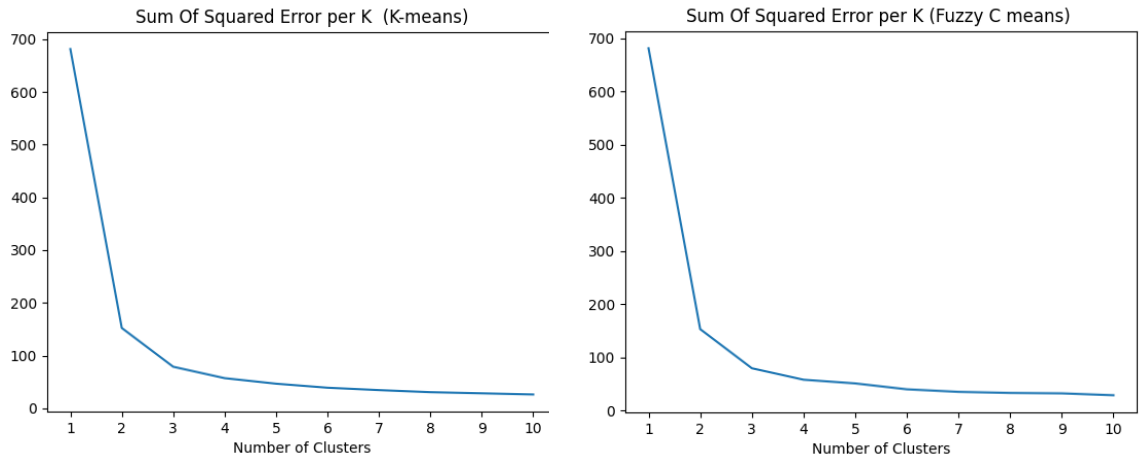
Δοκιμή στον k-means.

```
iris=load_iris()
x=iris.data
y=iris.target
testK=[1,2,3,4,5,6,7,8,9,10]
SSEs=[]
for k in testK:
   kmeans = KMeans(n_clusters = k, init = 'random', max_iter = 500, n_init = 10, random_state =
0).fit(x)
   SSEs.append(kmeans.inertia_)
```

Δοκιμή στον fuzzy c-means

```
SSEs_fuzz=[]
for k in testK:
   cntr, u, u0, distance_matrix, jm, p, fpc = fuzz.cluster.cmeans(x.T, k, 2, error=0.005,
maxiter=1000, seed=1998)
   cluster_membership = np.argmax(u,axis=0)
   sse=0
   for i in range(x.shape[0]):
      sse=sse + distance_matrix[cluster_membership[i],i]**2
   SSEs_fuzz.append(sse)
```

Τα αποτελέσματα :



Όπως βλέπουμε μπορούμε και έτσι να βρούμε ότι το βέλτιστο k είναι το 3.