EDS Assignment 6

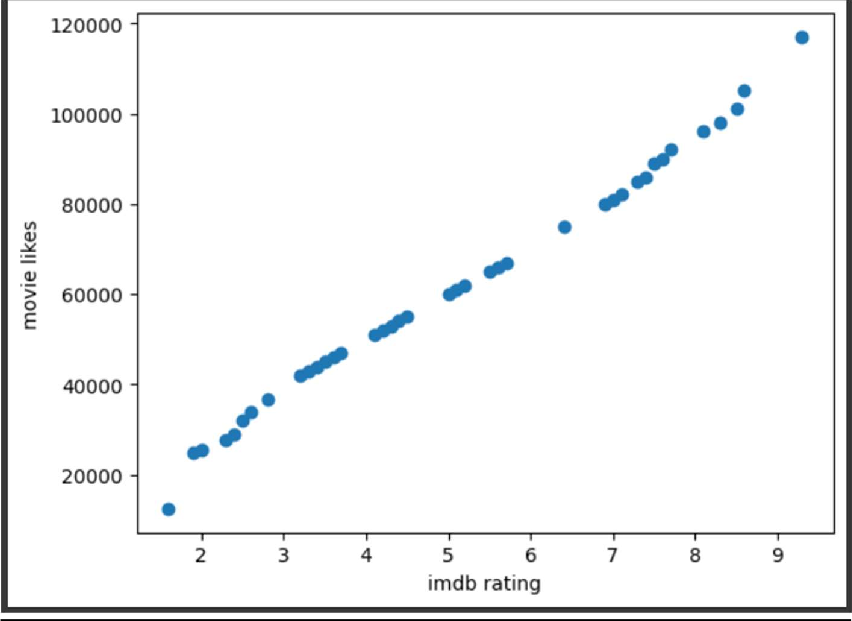
# Our Team Members:

Amrik Bhadra (203)

Tejashree Darade (213)

Prajwal Ganar (220)

# Linear Regression:



import numpy as np import pandas as pd

import matplotlib.pyplot as plt from sklearn import linear\_model

from sklearn.model\_selection import train\_test\_split df = pd.read\_csv("/content/movie\_data.csv")

#data cleaning df.dropna(inplace=True)

df.reset\_index(drop=True, inplace=True) df1 = df.head(40)

# print(df1)

plt.scatter(df1['imdb\_score'], df1['movie\_likes']) plt.xlabel('imdb rating')

plt.ylabel('movie likes')

X = np.array(df1[['imdb\_score']]).reshape(-1,1)

Y = np.array(df1[['movie\_likes']]).reshape(-1,1) X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size = 0.25)

# create linear regression object reg = linear\_model.LinearRegression()

reg.fit(X\_train, Y\_train) #training the model

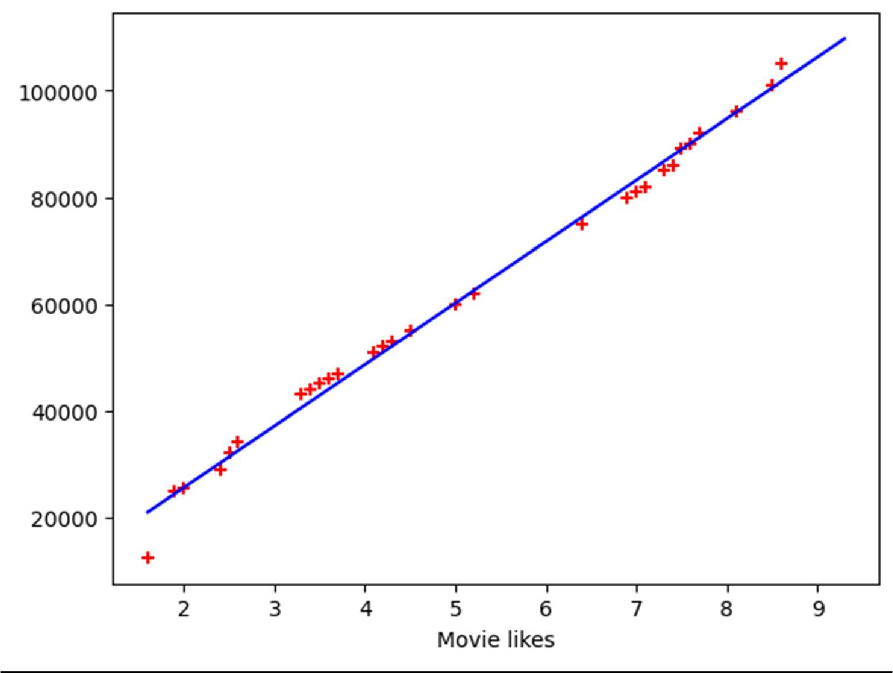
# predicting movie likes using the testing dataset on the trained model reg.predict(X\_test)

# ploting linear regression line

plt.scatter(X\_train, Y\_train, color='red', marker='+') plt.xlabel('IMDB')

plt.xlabel('Movie likes')

plt.plot(df1['imdb\_score'], reg.predict(df1[['imdb\_score']]), color='blue')



1. KNN
2. import pandas as pd
3. import seaborn as sns
4. import matplotlib.pyplot as plt
5. import numpy as np
6. df = pd.read\_csv("prostate.csv")
7. df.head()
8. from sklearn.preprocessing import StandardScaler 10.

11.

12.

13.

14.

15.

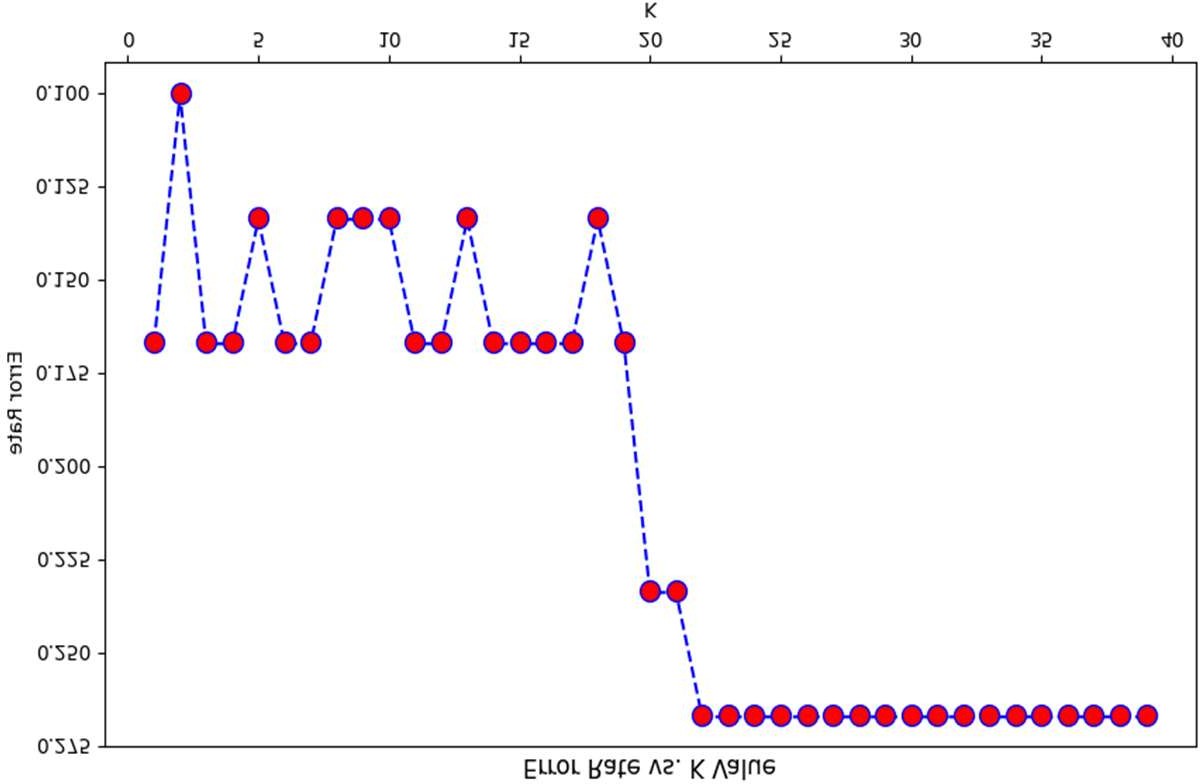
16.

scaler = StandardScaler()

scaler.fit(df.drop('Target', axis=1)) scaled\_features = scaler.transform(df.drop('Target',

axis=1))

1. df\_feat = pd.DataFrame(scaled\_features,
2. columns=df.columns[:-1])
3. df\_feat.head()
4. from sklearn.metrics import classification\_report,\
5. confusion\_matrix
6. from sklearn.neighbors import KNeighborsClassifier
7. from sklearn.model\_selection import train\_test\_split 24.
8. X\_train, X\_test,\
9. y\_train, y\_test = train\_test\_split(scaled\_features,
10. df['Taregt'],
11. test\_size=0.30)
12. # Remember that we are trying to come up
13. # with a model to predict whether
14. # someone will Target or not.
15. # We'll start with k = 1. 34.
16. knn = KNeighborsClassifier(n\_neighbors=1)
17. knn.fit(X\_train, y\_train)
18. pred = knn.predict(X\_test) 38.
19. # Predictions and Evaluations
20. # Let's evaluate our KNN model !
21. print(confusion\_matrix(y\_test, pred))
22. print(classification\_report(y\_test, pred))
23. error\_rate = [] 44.
24. # Will take some time
25. for i in range(1, 40):
26. knn = KNeighborsClassifier(n\_neighbors=i)
27. knn.fit(X\_train, y\_train)
28. pred\_i = knn.predict(X\_test)
29. error\_rate.append(np.mean(pred\_i != y\_test)) 52.
30. plt.figure(figsize=(10, 6))
31. plt.plot(range(1, 40), error\_rate, color='blue',
32. linestyle='dashed', marker='o',
33. markerfacecolor='red', markersize=10) 57.
34. plt.title('Error Rate vs. K Value')
35. plt.xlabel('K')
36. plt.ylabel('Error Rate')
37. plt.show()



# FIRST A QUICK COMPARISON TO OUR ORIGINAL K = 1

knn = KNeighborsClassifier(n\_neighbors = 1)

knn.fit(X\_train, y\_train) pred = knn.predict(X\_test)

print('WITH K = 1') print('Confusion Matrix')

print(confusion\_matrix(y\_test, pred)) print('Classification Report')

print(classification\_report(y\_test, pred))

WITH K = 1

Confusion Matrix [[19 3]

[ 2 6]]

Classification Report

precision recall f1-score support

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0.90 | 0.86 | 0.88 | 22 |
| 1 | 0.67 | 0.75 | 0.71 | 8 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| accuracy |  |  | 0.83 | 30 |  |
| macro avg | 0.79 | 0.81 | 0.79 | 30 |  |
| weighted avg | 0.84 | 0.83 | 0.84 |  | 30 |

# NOW WITH K = 10

knn = KNeighborsClassifier(n\_neighbors = 10)

knn.fit(X\_train, y\_train) pred = knn.predict(X\_test)

print('WITH K = 10')

print('Confusion Matrix') print(confusion\_matrix(y\_test, pred)) print('Classification Report') print(classification\_report(y\_test, pred))

WITH K = 10

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

X,y = make\_blobs(n\_samples = 500,n\_features = 2,centers = 5,random\_state = 23) fig = plt.figure(0)

plt.grid(True) plt.scatter(X[:,0],X[:,1]) plt.show()

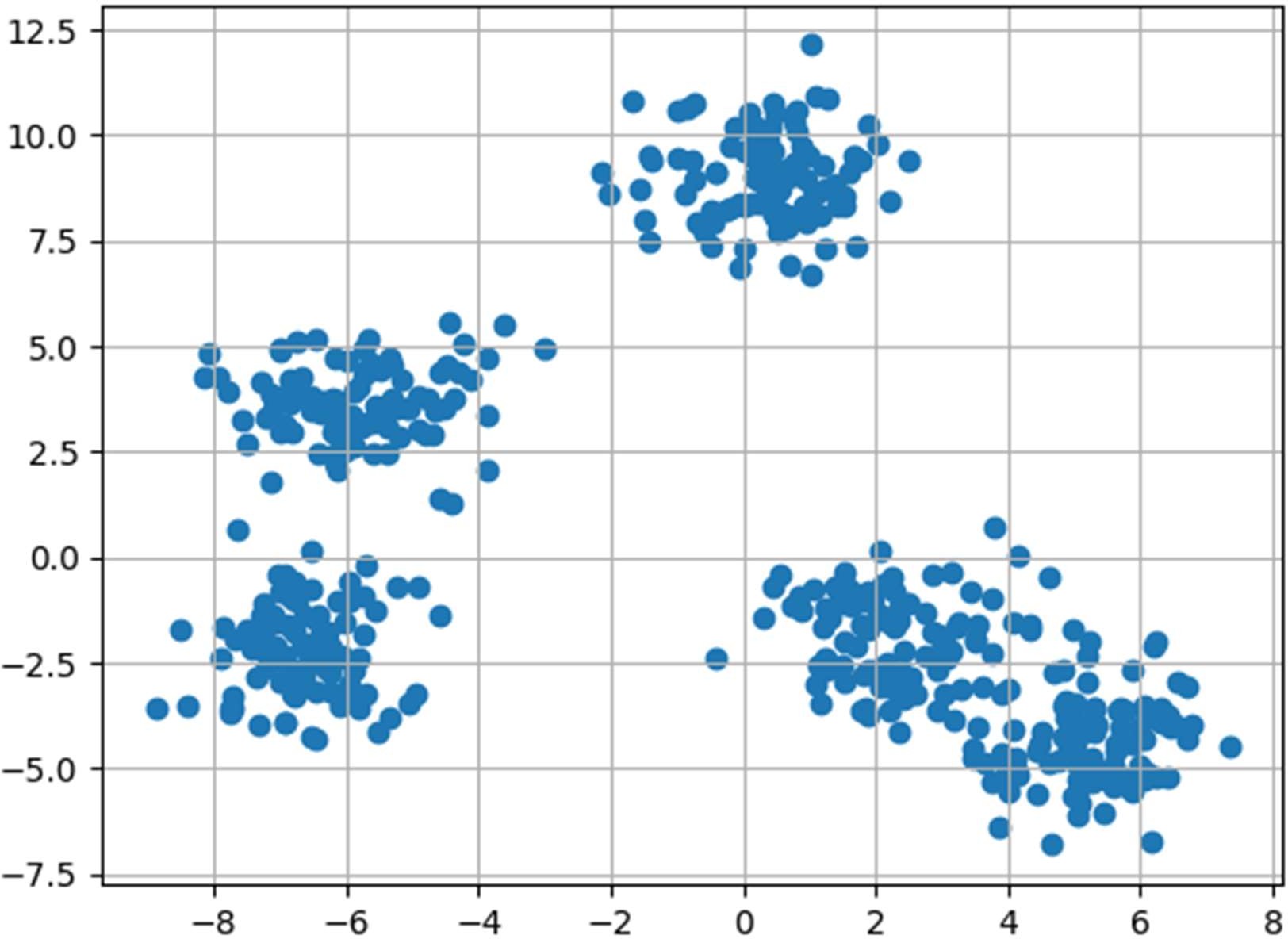
Confusion Matrix [[21 1]

[ 3 5]]

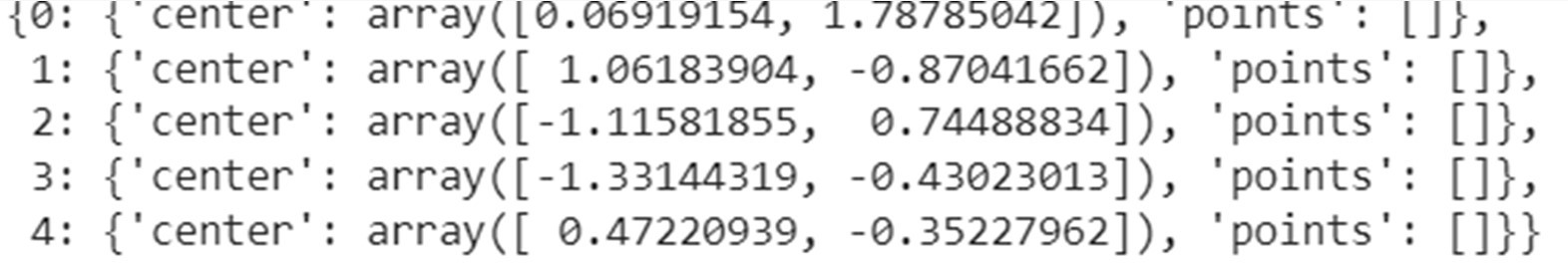
Classification Report

precision recall f1-score support

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0.88 | 0.95 | 0.91 | 22 |
| 1 | 0.83 | 0.62 | 0.71 | 8 |
| accuracy |  |  | 0.87 | 30 |
| macro avg | 0.85 | 0.79 | 0.81 | 30 |
| weighted avg | 0.86 | 0.87 | 0.86 | 30 |
| 3. K means |  |  |  |  |
| #k-Means  import numpy | as np |  |  |  |



k = 5



clusters = {} np.random.seed(23)

for idx in range(k):

center = 2\*(2\*np.random.random((X.shape[1],))-1) points = []

cluster = {

'center' : center, 'points' : []

}

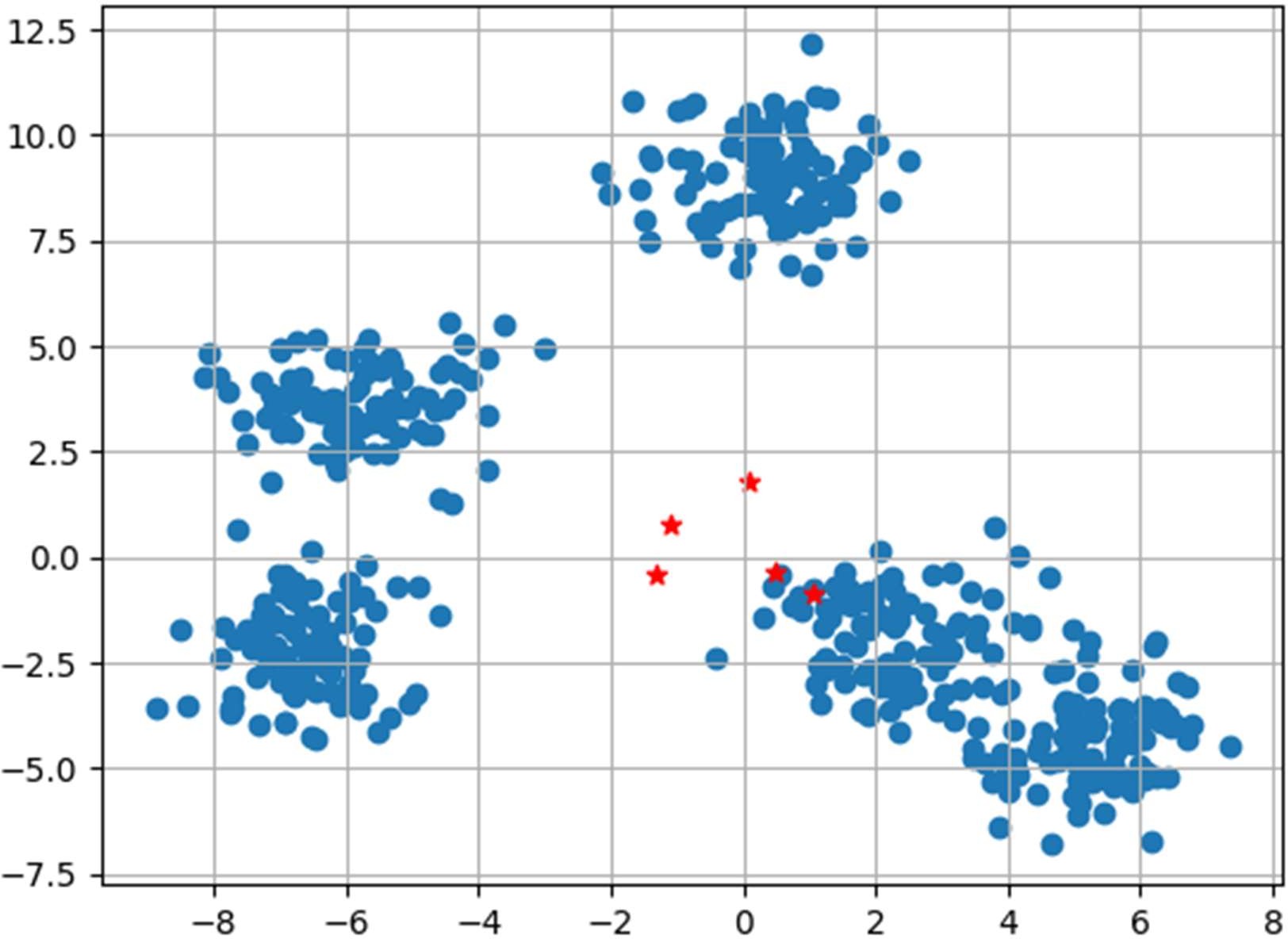
clusters[idx] = cluster clusters

plt.scatter(X[:,0],X[:,1]) plt.grid(True)

for i in clusters:

center = clusters[i]['center'] plt.scatter(center[0],center[1],marker = '\*',c = 'red')

plt.show()



def distance(p1,p2):

return np.sqrt(np.sum((p1-p2)\*\*2))

#Implementing E step

def assign\_clusters(X, clusters): for idx in range(X.shape[0]):

dist = []

curr\_x = X[idx]

for i in range(k):

dis = distance(curr\_x,clusters[i]['center']) dist.append(dis)

curr\_cluster = np.argmin(dist) clusters[curr\_cluster]['points'].append(curr\_x)

return clusters

#Implementing the M-Step

def update\_clusters(X, clusters): for i in range(k):

points = np.array(clusters[i]['points']) if points.shape[0] > 0:

new\_center = points.mean(axis =0) clusters[i]['center'] = new\_center

clusters[i]['points'] = [] return clusters

def pred\_cluster(X, clusters): pred = []

for i in range(X.shape[0]): dist = []

for j in range(k): dist.append(distance(X[i],clusters[j]['center']))

pred.append(np.argmin(dist)) return pred

clusters = assign\_clusters(X,clusters) clusters = update\_clusters(X,clusters) pred = pred\_cluster(X,clusters)

plt.scatter(X[:,0],X[:,1],c = pred) for i in clusters:

center = clusters[i]['center'] plt.scatter(center[0],center[1],marker = '^',c = 'red')

plt.show()

