علی آرمان 4023624002

در پیاده سازی ابتدا یک بار کد ها نوشته شد و در نهایت مرتب شد و به صورت بهینه کنار هم قرار گرفته.

در ابتدا نگارش کد ها گزارش شده و بعد از آن اجرای كد.

در ابتدا کتابخانه های لازم را ایمپورت میکنیم.

1. import warnings

2. import numpy as np

3. import pandas as pd

4. import seaborn as sns

5. import matplotlib.pyplot as plt

6. from sklearn.preprocessing import MinMaxScaler

7. from sklearn.model\_selection import train\_test\_split

8. from sklearn.metrics import classification\_report, confusion\_matrix

مرحله 1

در مرحله اول فایل مورد را باز کرده و ستون های مورد نظر را انتخاب میکنیم:

1. # Step 1: Read the dataset

2. def read\_data(file\_path):

3. data = pd.read\_csv(file\_path)

4. selected\_data = data[['tenure', 'income', 'employ']]

5. return selected\_data

مرحله 2

در این مرحله پیش پردازش متن را انجام میدهیم که در اینجا ابتدا مقادیر null با میانگین پر شده اند و داده های عددی با minmax scale نرمال سازی شده اند:

1. # Step 2: Preprocess the data

2. def preprocess\_data(data):

3. # Fill missing data with means

4. data.fillna(data.mean(), inplace=True)

5. # Normalize 'tenure', 'income', and 'employ' using Min-Max scaling

6. scaler = MinMaxScaler()

7. data[['tenure', 'income', 'employ']] = scaler.fit\_transform(data[['tenure', 'income', 'employ']])

8. return data

مرحله 3

در این مرحله مطابق توضیحات 80 درصد داده های برای آموزش جدا شده اند:

1. # Step 3: Split the data into training and testing sets

2. def split\_data(data):

3. X\_train, X\_test = train\_test\_split(data, test\_size=0.2, shuffle=True, random\_state=17)

4. return X\_train, X\_test

مرحله 4

در این مرحله به دلیل آزمایش های بیشتر، 6 روش برای تعیین مرکز دسته ها انتخاب شد که این 7 روش عبارت اند از:

1. Random:  
   انتخاب مرکز دسته های به صورت رندم
2. Kmeans++:  
   انتخاب مرکز دسته ها با استفاده از الگوریتم kmeans++ (که بهترین نتیجه را حاصل کرد)
3. First-row-point:  
   انتخاب k مقدار اول به عنوان مرکز دسته
4. Random-partition:  
   انتخاب به صورت پارتیشین بندی
5. Random-subset:  
   انتخاب به صورت تصادفی از بین ساب ست ها
6. Kmeans2:

استفاده از الگوریتم kmeans|| که شباهت زیادی به kmeans++ دارد.

در آموزش از تعداد دور های مختلفی جهت پیدا کردن بهترین نتیجه استفاده شده که تعداد دور های آزمایش شده برابر است با: 100 200 500 و 1000

در نهایت به تعداد دور های آموزش فاصله هر داده از هر مرکز داده (که با یکی از 6 روش بدست آمده) محاسبه می شود و کمترین فاصله از بین فاصله بین هر داده با مراکز داده ها به دست می آید سپس لیبل آن دسته به داده اختصاص داده میشود.

1. # Step 4: Implement the K-means algorithm

2. def k\_means(data, k, initialization, num\_iterations=100):

3. # Convert the data array back to a pandas DataFrame

4. data = pd.DataFrame(data)

5.

6. if initialization == 'random':

7. # Initialize centroids randomly

8. centroids = data.sample(n=k).values

9.

10. elif initialization == 'kmeans++':

11. # Initialize centroids using KMeans++ algorithm

12. centroids = [data.sample().values[0]]

13. for \_ in range(1, k):

14. distances = np.array([min([np.inner(c-x,c-x) for c in centroids]) for x in data.values])

15. probs = distances/distances.sum()

16. cumprobs = probs.cumsum()

17. r = np.random.rand()

18. for j,p in enumerate(cumprobs):

19. if r < p:

20. i = j

21. break

22. centroids.append(data.values[i])

23.

24. elif initialization == 'first\_few\_points':

25. # Initialize centroids using the first few data points

26. centroids = data.head(k).values

27.

28. elif initialization == 'random\_partition':

29. # Initialize centroids using random partition method

30. shuffled\_indices = np.random.permutation(len(data))

31. centroids = [data.values[i] for i in range(k)]

32.

33. elif initialization == 'random\_subset':

34. # Initialize centroids using a random subset of data points

35. subset\_indices = np.random.choice(len(data), k, replace=False)

36. centroids = data.iloc[subset\_indices].values

37.

38. elif initialization == 'kmeans2':

39. # Initialize centroids using k-means|| algorithm

40. centroids = [data.sample().values[0]]

41. for \_ in range(1, k):

42. distances = np.array([min([np.inner(c-x,c-x) for c in centroids]) for x in data.values])

43. probs = k \* distances / sum(distances)

44. cumprobs = probs.cumsum()

45. r = np.random.rand()

46. for j, p in enumerate(cumprobs):

47. if r < p:

48. i = j

49. break

50. centroids.append(data.values[i])

51.

52. for \_ in range(num\_iterations):

53. distances = np.sqrt(((data.values[:, np.newaxis, :] - centroids)\*\*2).sum(axis=2))

54. # Assign each data point to the nearest centroid

55. labels = np.argmin(distances, axis=1)

56. # Update centroids based on the mean of data points in each cluster

57. for i in range(k):

58. centroids[i] = np.mean(data.values[labels == i], axis=0)

59. return labels, centroids

مرحله 5

در این مرحله جهت مشاهده و مقایسه بهتر داده ها از 5 پلات استفاده شده که به شرح زیر می باشد:

در عنوان شماره تعداد دسته، تعداد دور آموزش، شماره آزمون و نوع انتخاب مرکز دسته ذکر شده

به ترتیب از سمت چپ

1-نموار دسته ها به همراه مراکز

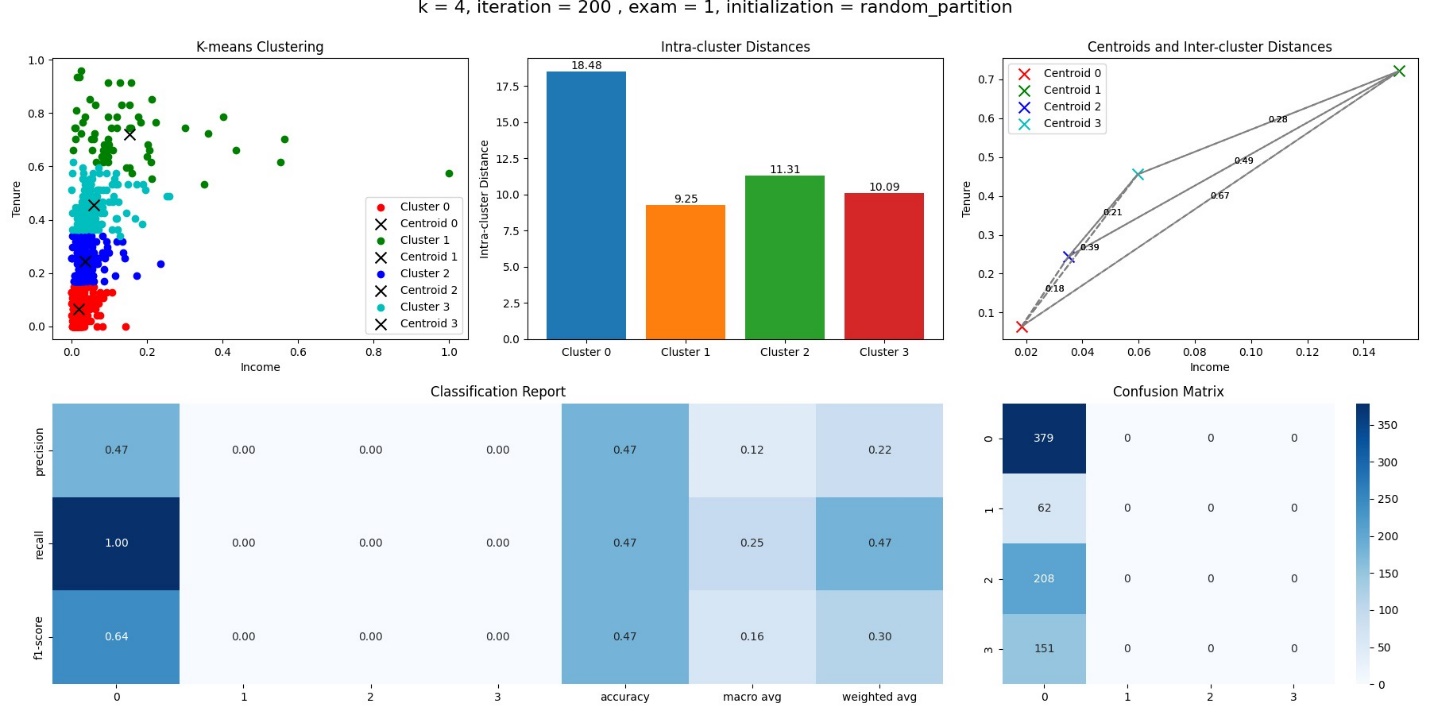
2-نمودار فاصله درون خوشه ای برای هر خوشه

3-فاصله مرکز دسته ها

4- classification report

5- confusion matrix

به عنوان مثال:



جهت ایجاد یک پلات 2 در 3 ایجاد شده و در هر ساب پلات داده مربوطه قرار داده شده برای محاسبه فاصله از intra\_cluster\_distance استفاده شده و برای فاصله بین مرکز دسته ها از در همان پلات هر مرکز دسته فاصله اش با سایرین حساب شده با خط در پلات قرار داده شده و مقادیر class\_report\_df و confusion\_matrix از تابع calculate\_measures استفاده شده که در اجرا شرح داده خواهد شد. در نهایت هر پلات ایجاد شده برای بررسی ذخیره می شود.

کد مربوط به این مرحله:

1. # Step 5: Visualize the clusters

2. def visualize\_clusters(data, labels, centroids, class\_report\_df, cm, k, it, exam, initialization):

3. plt.figure(figsize=(18, 9))

4. colors = ['r', 'g', 'b', 'c', 'm']

5.

6. plt.subplot(2, 3, 1)

7. for i in range(len(centroids)):

8. plt.scatter(data[labels == i][:, 0], data[labels == i][:, 1], color=colors[i], label=f'Cluster {i}')

9. plt.scatter(centroids[i][0], centroids[i][1], color='k', marker='x', s=100, label=f'Centroid {i}')

10. plt.xlabel('Income')

11. plt.ylabel('Tenure')

12. plt.title('K-means Clustering')

13. plt.legend()

14.

15. plt.subplot(2, 3, 2)

16. for i, intra\_distance in enumerate(intra\_cluster\_distance(data, labels, centroids)):

17. plt.bar(f'Cluster {i}', intra\_distance)

18. plt.text(i, intra\_distance, f'{intra\_distance:.2f}', ha='center', va='bottom')

19. plt.ylabel('Intra-cluster Distance')

20. plt.title('Intra-cluster Distances')

21.

22. plt.subplot(2, 3, 3)

23. for i, centroid in enumerate(centroids):

24. plt.scatter(centroid[0], centroid[1], color=colors[i], marker='x', s=100, label=f'Centroid {i}')

25. for j, other\_centroid in enumerate(centroids):

26. if i != j:

27. distance = np.sqrt(((centroid - other\_centroid)\*\*2).sum())

28. plt.plot([centroid[0], other\_centroid[0]], [centroid[1], other\_centroid[1]], color='gray', linestyle='--')

29. plt.text((centroid[0] + other\_centroid[0]) / 2, (centroid[1] + other\_centroid[1]) / 2, f'{distance:.2f}', fontsize=8)

30. plt.xlabel('Income')

31. plt.ylabel('Tenure')

32. plt.title('Centroids and Inter-cluster Distances')

33. plt.legend()

34.

35. plt.subplot(2, 3, (4, 5))

36. sns.heatmap(class\_report\_df.iloc[:, :-1].T, annot=True, cmap='Blues', cbar=False, fmt=".2f")

37. plt.title('Classification Report')

38.

39. plt.subplot(2, 3, 6)

40. sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(labels), yticklabels=np.unique(labels))

41. plt.title('Confusion Matrix')

42.

43. plt.suptitle(f'k = {k}, iteration = {it} , exam = {exam}, initialization = {initialization}', fontsize=16, y=1)

44. plt.tight\_layout()

45.

46. filename = f'img/k{k}\_it{it}\_exam{exam}\_{initialization}.png'

47. plt.savefig(filename)

مرحله 6

تابع calculate\_measures، در این تابع لیبل ها و دسته بندی که توسط الگوریتم ایجاد شده به عنوان ورودی داده میشود و در نهایت خروجی confusion\_matrix و classification\_report را ایجاد میکند و بازگشت می دهد:

1. # Step 6: Calculate evaluation measures

2. def calculate\_measures(labels, predictions):

3. cm = confusion\_matrix(labels, predictions)

4. class\_report\_df = pd.DataFrame(classification\_report(labels, predictions, output\_dict=True)).T

5. return cm, class\_report\_df

مرحله 7

محاسبه فاصله درون خوشه ای:

برای محاسبه تابع intra\_cluster\_distance نوشته شده که در هر دسته فاصله ی هر داده ها را حساب کرده و در نهایت تمام آن ها را با هم جمع میکند و داخل لیست فاصله ها قرار میدهد.

کد این مرحله:

1. # Step 7: Calculate intra-cluster distance

2. def intra\_cluster\_distance(data, labels, centroids):

3. distances = []

4. for i, centroid in enumerate(centroids):

5. distance = np.sqrt(((data[labels == i] - centroid)\*\*2).sum(axis=1)).sum()

6. distances.append(distance)

7. return distances

مرحله 8

این قسمت برای محاسبه فاصله بین مراکز دسته ها می باشد که به صورت مستقیم داخل مرحله 5 استفاده شده

1. # Step 8: Calculate inter-cluster distance

2. def inter\_cluster\_distance(centroids):

3. distances = []

4. k = len(centroids)

5. for i in range(k):

6. for j in range(i+1, k):

7. distance = np.sqrt(((centroids[i] - centroids[j])\*\*2).sum())

8. distances.append(distance)

9. return distances

مرحله اجرا

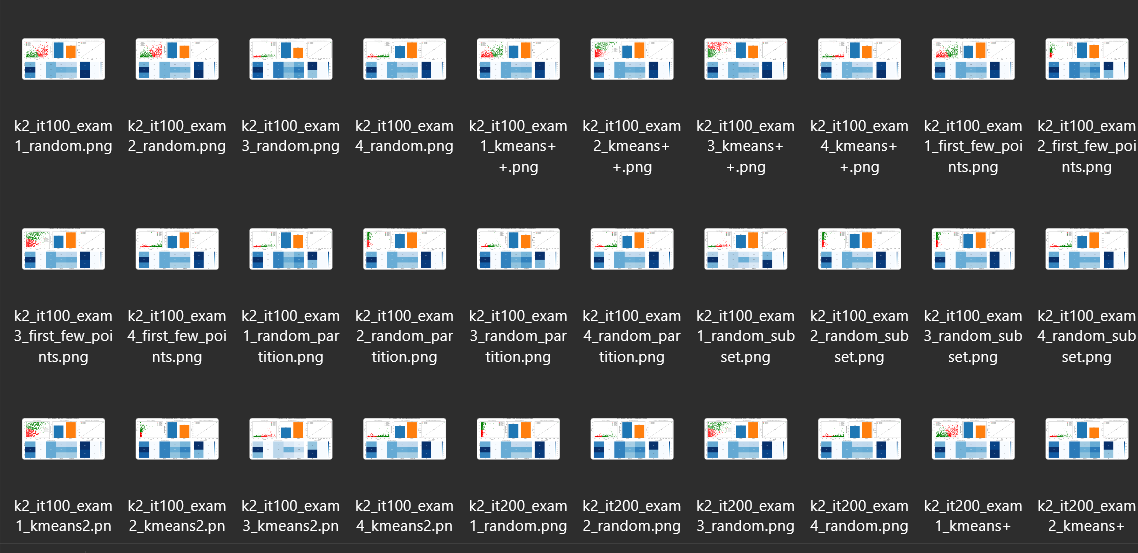
برای اجرا ابتدا داده فراخونی شده و مرحله پیش پردازش بر روی آن انجام شده، سپس داده تقسیم شده و وارد مرحله آموزش می شود.

جهت پیدا کردن بهترین نتیجه از 4 حلقه تو در تو استفاده شده

1. حلقه اول برای حرکت بین مقادیر k = 1 2 3 4 5
2. حلقه دوم برای حرکت بین مقادیر تعداد دور آموزش iterations\_values = 100 200 500 1000
3. حلقه سوم برای حرکت بین روش های انتخاب مراکز دسته (همان 6 روش ذکر شده در مرحله 4)
4. حلقه چهارم برای 4 آزمایش که در 3 آزمایش اول 2 ستون به صورت تصادفی می باش

در داخلی ترین حلقه هر بار تابع k\_means با مقادیر نسبت داده شده فراخوانی می شود و لیبل ها و مراکز دسته بازگشت داده میشود و با توجه به این مقادیر تابع calculate\_measures فراخونی میشود و جهت نمایش تابع نوشته شده در مرحله 5 (visualize\_clusters) را فراخوانی میکنیم.

تعداد دور های اجرا برابر است با 4\*4\*6\*4 = 386 در نتیجه 386 عکس ذخیره میشود



کد این مرحله:

1. def main():

2. # Step 1:

3. data = read\_data('Telecust1.csv')

4.

5. # Step 2:

6. data = preprocess\_data(data)

7.

8. # Step 3

9. X\_train, X\_test = split\_data(data)

10.

11. k\_values = [2, 3, 4, 5]

12. iterations\_values = [100, 200, 500, 1000]

13. initialization = ['random', 'kmeans++', 'first\_few\_points', 'random\_partition', 'random\_subset', 'kmeans2']

14.

15. warnings.filterwarnings("ignore", category=UserWarning)

16. counter = 1

17. for k in k\_values:

18. for it in iterations\_values:

19. for init in initialization:

20. for i in range(4):

21. if i < 3:

22. selected\_columns = np.random.choice(range(len(data.columns)), size=2, replace=False)

23. selected\_data = X\_train.iloc[:, selected\_columns].values

24. else:

25. selected\_data = X\_train.values

26.

27. labels, centroids = k\_means(selected\_data, k, initialization=init, num\_iterations = it)

28.

29. cm, class\_report\_df = calculate\_measures(labels, np.zeros\_like(labels))

30.

31. visualize\_clusters(selected\_data, labels, centroids, class\_report\_df, cm, k, it, i+1, init)

32.

33. print(f'{counter} of 386')

34. counter += 1

35.

36. if \_\_name\_\_ == "\_\_main\_\_":

37. main()

با بررسی تصاویر ذخیره شده، بهترین امتیاز به شرح زیر می باشد که در چندین حالت قابل مشاهده بوده:

Precision=0.72

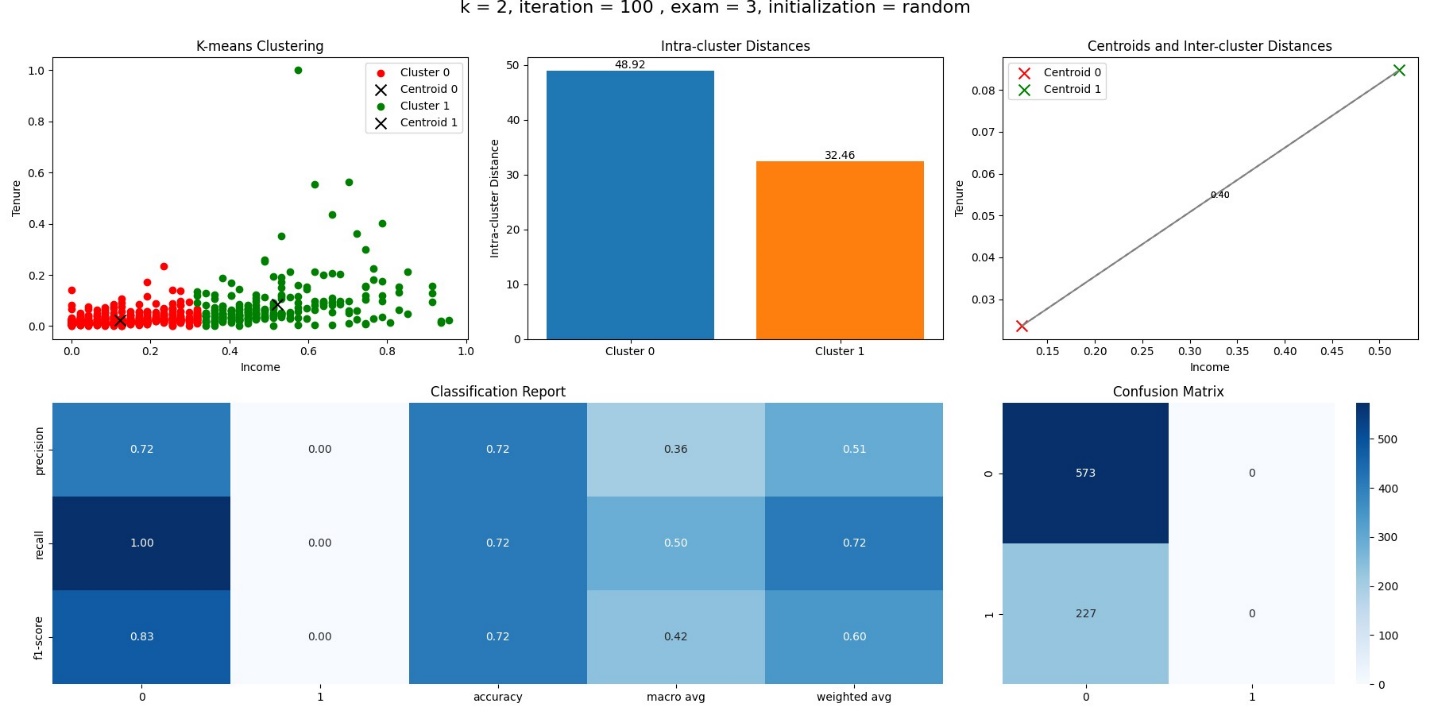
Recall=1.00

F1-score=0.83

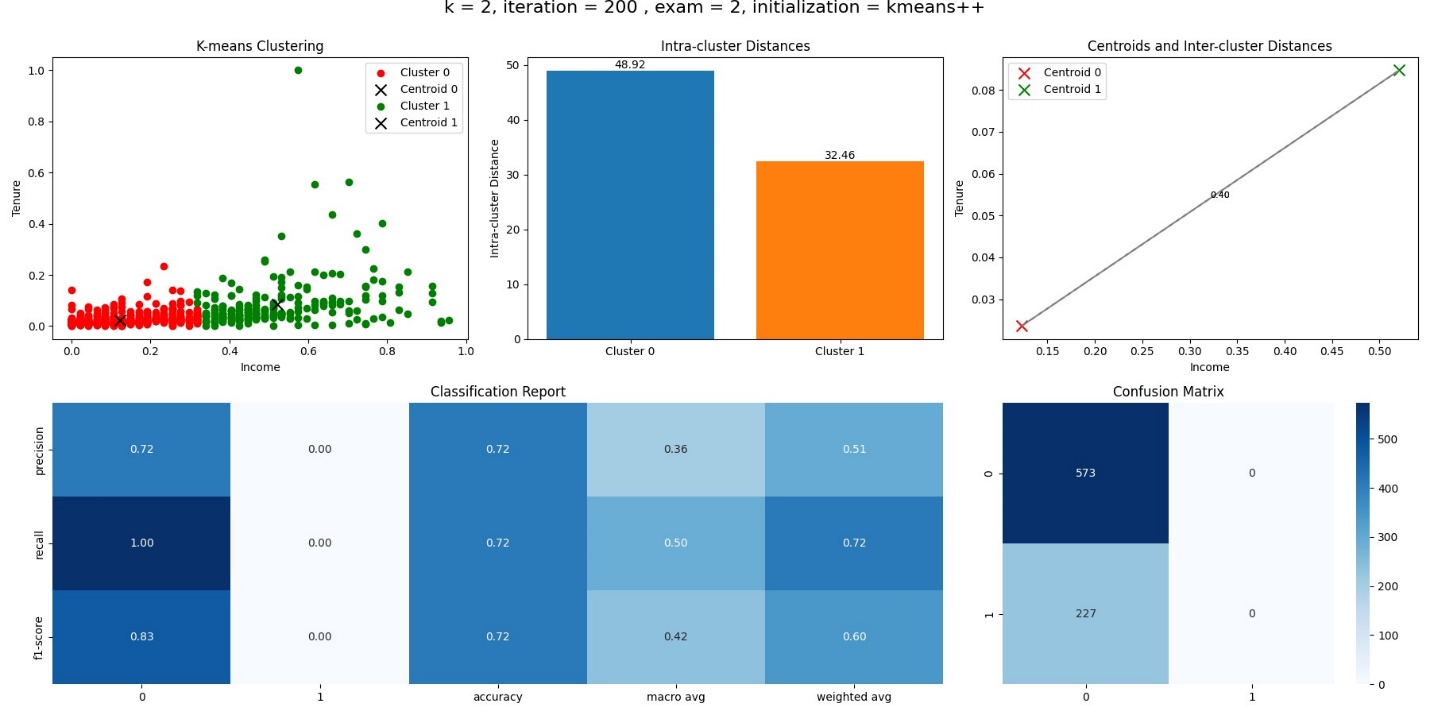
Accuracy=0.72

برخی از خروجی این داده ها:

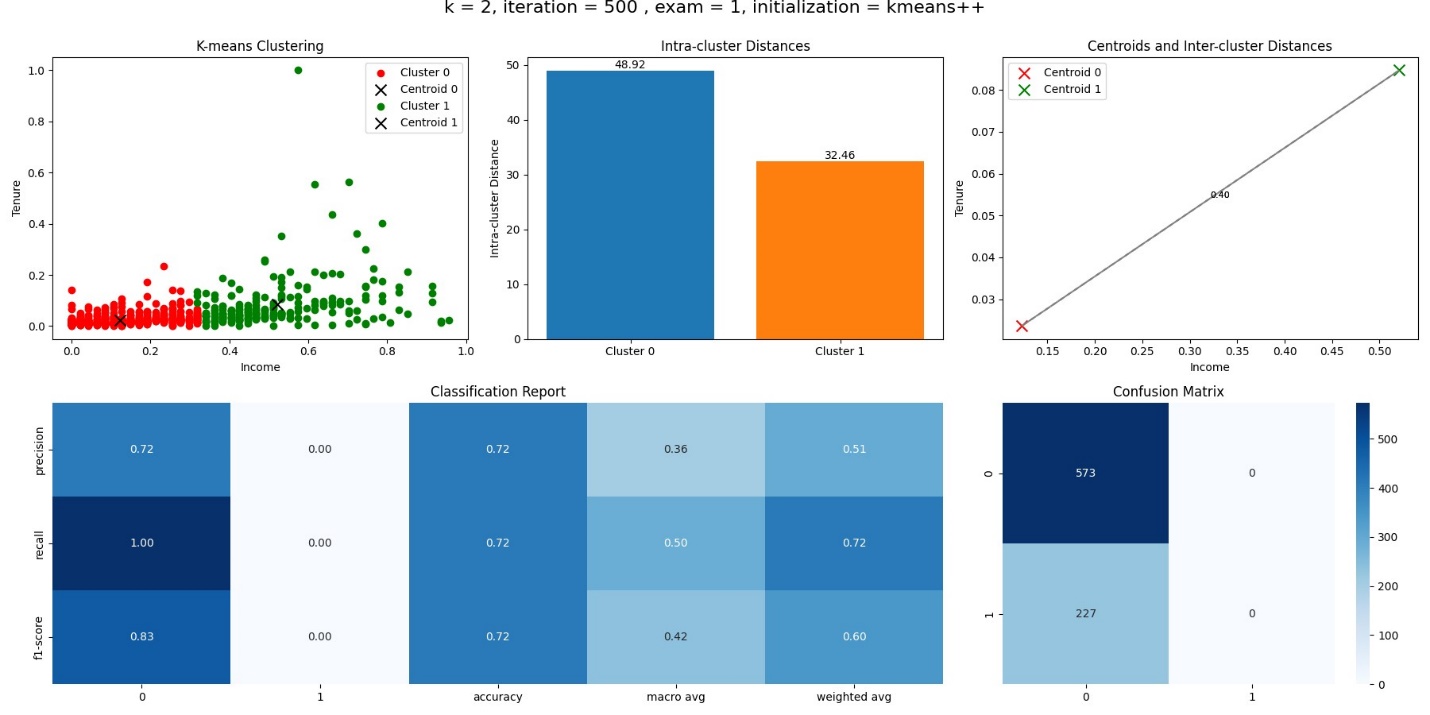
k2\_it100\_exam1\_random:



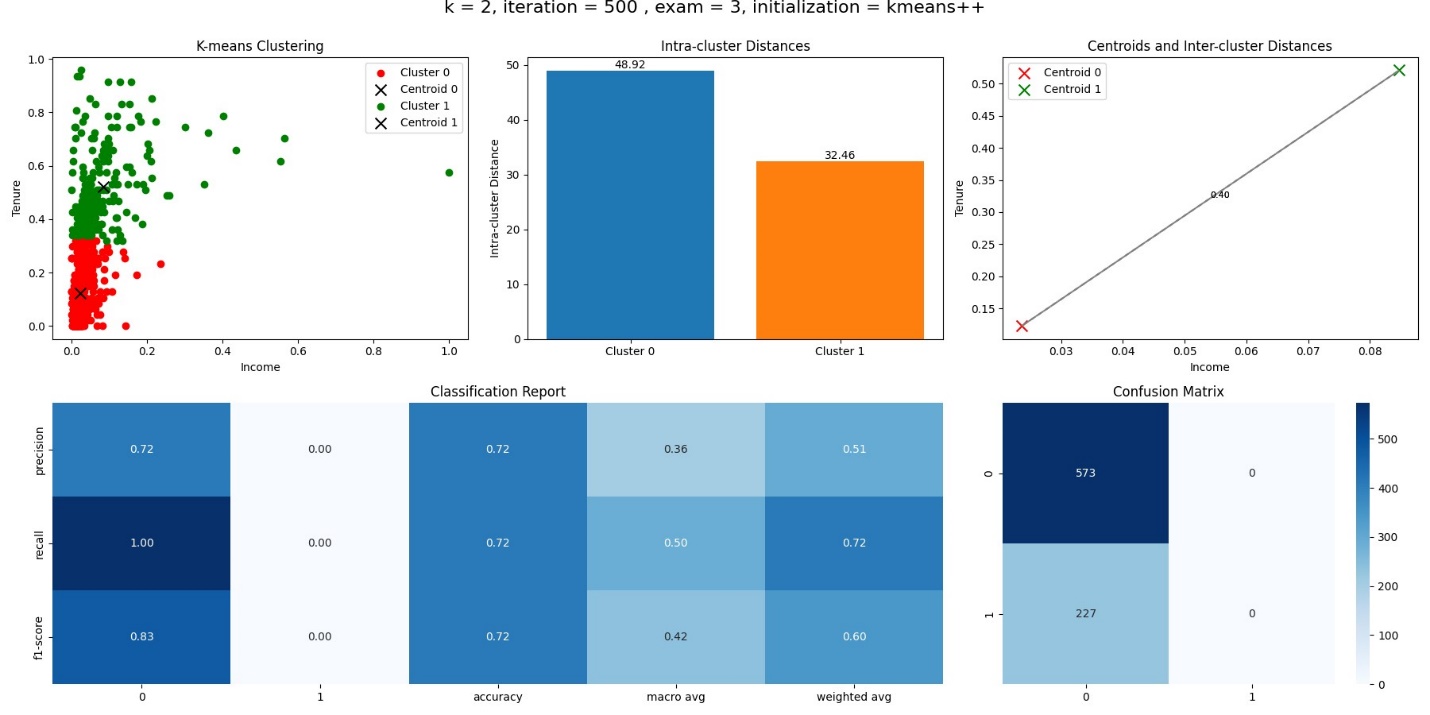
k2\_it200\_exam2\_kmeans++:



k2\_it500\_exam1\_kmeans++:



k2\_it500\_exam3\_kmeans++:



در کل بهترین نتیجه ها در تعداد خوشه برابر 2 و الگوریتم kmeans++ با تعداد دور کمتر از 1000 نتیجه میدهد.