Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчет по лабораторной работе №3

«Кластеризация»

по дисциплине «Машинное обучение»

Выполнил
студент гр. 3530904/90102 Афанасьев Е.Д.

Руководитель
старший преподаватель ВШПИ Селин И. А.

Санкт-Петербург 2022

Оглавление

Задачи

1. Разбейте множество объектов из набора данных pluton.csv на 3 кластера с помощью k-means. Сравните качество разбиения в зависимости от максимального числа итераций алгоритма и использования стандартизации.
2. Разбейте на кластеры множество объектов из наборов данных clustering_1.csv, clustering_2.csv и clustering_3.csv с помощью k-means, DBSCAN и иерархической кластеризации. Определите оптимальное количество кластеров (где это применимо). Какой из методов сработал лучше и почему?
3. Осуществите сжатие цветовой палитры изображения (любого, на ваш выбор). Для этого выделите n кластеров из цветов всех пикселей изображения и зафиксируйте центра этих кластеров. Создайте изображение с цветами из сокращенной палитры (цвета пикселей только из центров выделенных кластеров). Покажите исходное и сжатое изображения.
4. Постройте дендрограмму для набора данных votes.csv (число голосов, поданных за республиканцев на выборах с 1856 по 1976 год). Строки представляют 50 штатов, а столбцы - годы выборов (31). Проинтерпретируйте полученный результат.

Исследование качества разбиения в зависимости от максимального числа итераций алгоритма и использования стандартизации для k-means

Я обучил модели на стандартизированных и нестандартизированных данных для различного числа итераций от 1 до 10. Видно, что при любом количестве итераций происходит одинаковое разбиение на кластеры и их центры совпадают.

```
No standartization
Max iter = 1
[24, 15, 6]
 [[ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 2
[15, 24, 6]
 [[ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 3
[15, 24, 6]
 [[ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 4
[24, 15, 6]
 [[ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 5
[24, 15, 6]
 [[ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 6
[24, 15, 6]
 [[ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 7
[15, 24, 6]
 [[ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 8
[24, 15, 6]
 [[ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 9
[24, 15, 6]
 [[ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.38306667 60.63393333 24.38753333  8.66646667]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
Max iter = 10
[15, 24, 6]
 [[ 1.38306667 60.63393333 24.38753333  8.66646667]
```

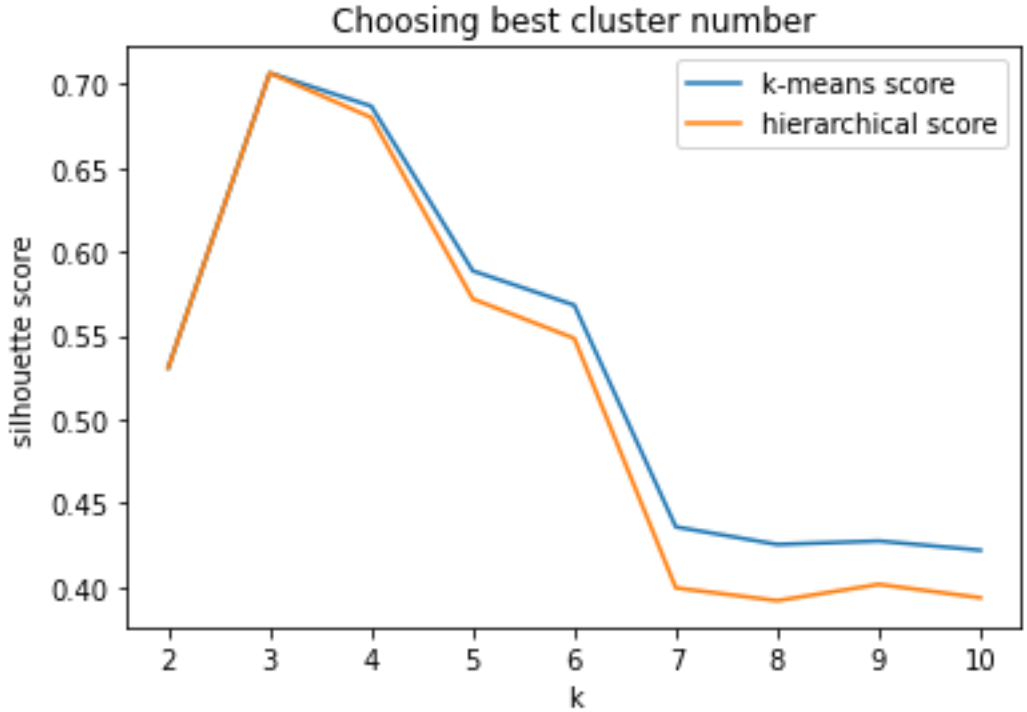
```
 [ 0.21195833 75.68516667 20.503       2.65925   ]
 [ 1.10666667 70.18466667 18.52033333  7.6705    ]]
With standartization
Max iter = 1
[6, 24, 15]
 [[ 0.67798387  0.03543992 -1.2636083   0.78475027]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 1.16466728 -1.31858916  1.19688286  1.11866428]]
Max iter = 2
[24, 15, 6]
 [[-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 1.16466728 -1.31858916  1.19688286  1.11866428]
 [ 0.67798387  0.03543992 -1.2636083   0.78475027]]
Max iter = 3
[15, 24, 6]
 [[ 1.16466728 -1.31858916  1.19688286  1.11866428]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 0.67798387  0.03543992 -1.2636083   0.78475027]]
Max iter = 4
[15, 24, 6]
 [[ 1.16466728 -1.31858916  1.19688286  1.11866428]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 0.67798387  0.03543992 -1.2636083   0.78475027]]
Max iter = 5
[15, 24, 6]
 [[ 1.16466728 -1.31858916  1.19688286  1.11866428]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 0.67798387  0.03543992 -1.2636083   0.78475027]]
Max iter = 6
[24, 15, 6]
 [[-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 1.16466728 -1.31858916  1.19688286  1.11866428]
 [ 0.67798387  0.03543992 -1.2636083   0.78475027]]
Max iter = 7
[6, 24, 15]
 [[ 0.67798387  0.03543992 -1.2636083   0.78475027]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 1.16466728 -1.31858916  1.19688286  1.11866428]]
Max iter = 8
[15, 24, 6]
 [[ 1.16466728 -1.31858916  1.19688286  1.11866428]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 0.67798387  0.03543992 -1.2636083   0.78475027]]
Max iter = 9
[15, 24, 6]
 [[ 1.16466728 -1.31858916  1.19688286  1.11866428]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 0.67798387  0.03543992 -1.2636083   0.78475027]]
Max iter = 10
[6, 24, 15]
 [[ 0.67798387  0.03543992 -1.2636083   0.78475027]
 [-0.89741301  0.81525825 -0.43214971 -0.89535274]
 [ 1.16466728 -1.31858916  1.19688286  1.11866428]]
```

Разные методы кластеризации и определение оптимального числа кластеров

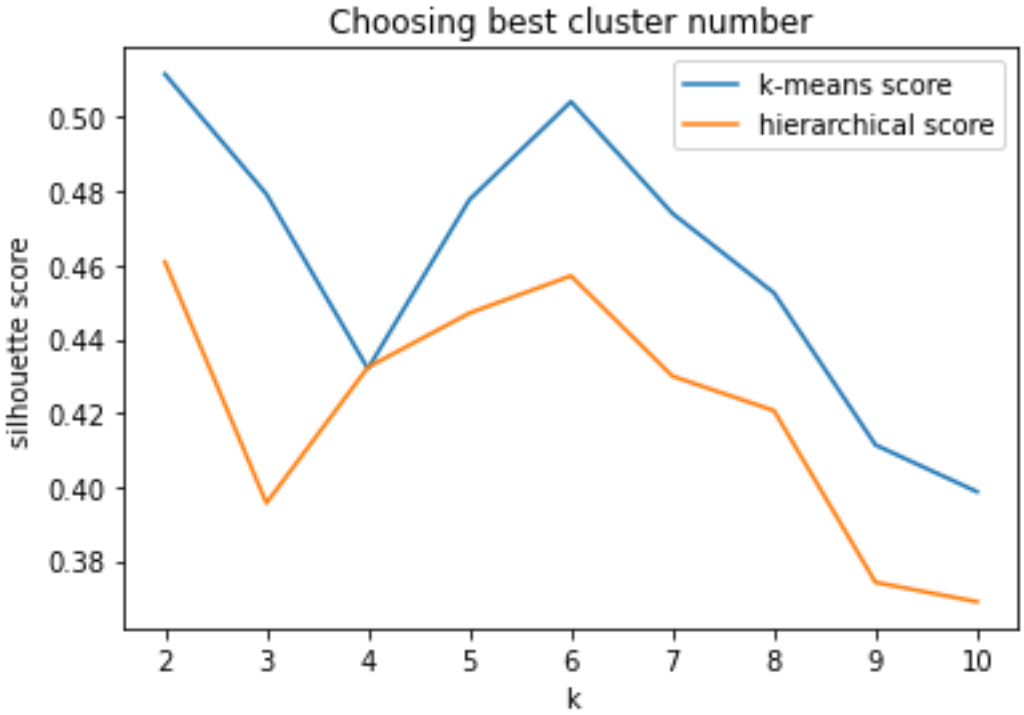
В первую очередь найдем оптимальное число кластеров для каждого набора данных с помощью метода ширины силуэта (silhouette_score). Их мы передадим в метод к средних и иерархическую кластеризацию.

`clustering_1.csv`

`clustering_2.csv`



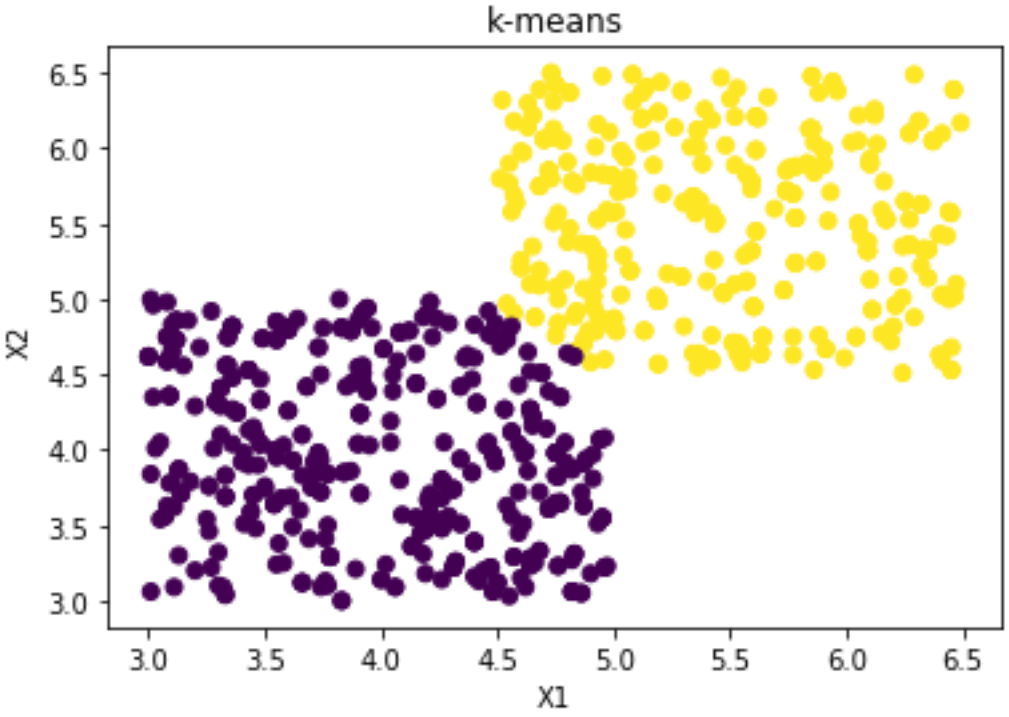
`clustering_3.csv`

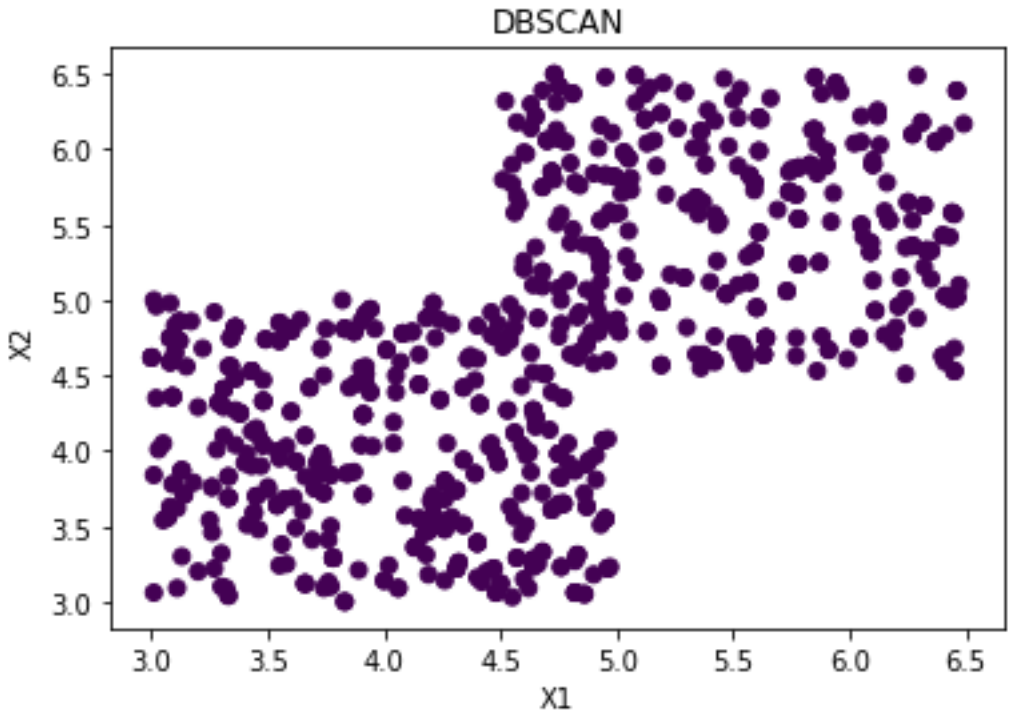


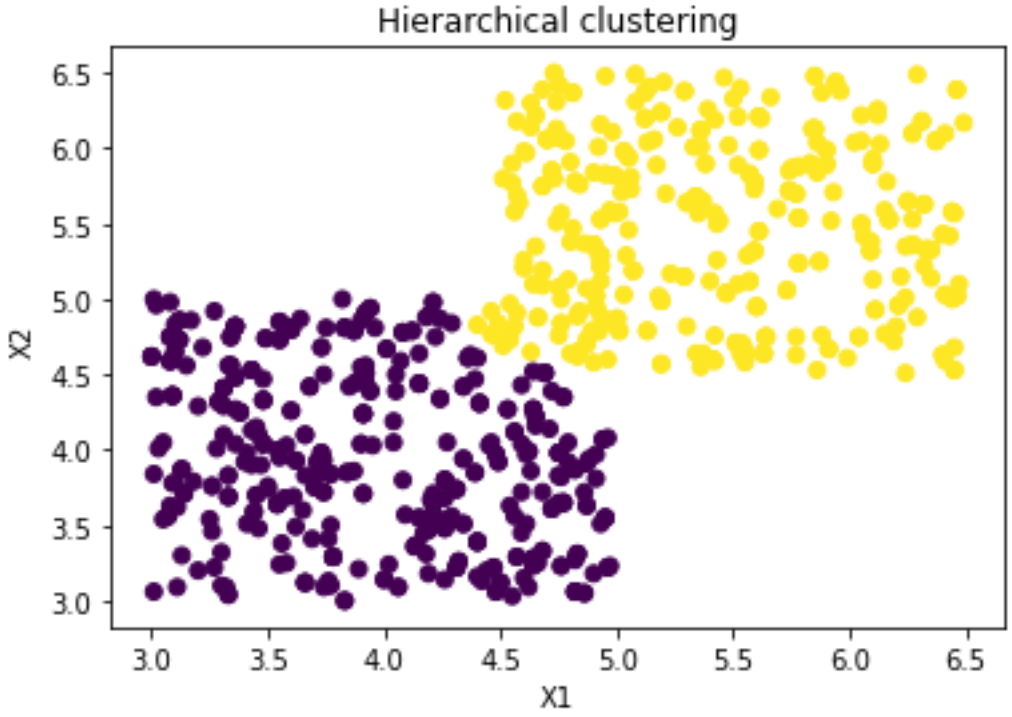
Видим, что оба метода ведут себя достаточно похоже, поэтому выберем для них одинаковое число кластеров, такое, в котором наша метрика максимальна. То есть 2 кластера в первом случае, 3 во втором и 2 в третьем.

Теперь проведем кластеризацию и проанализируем результаты.

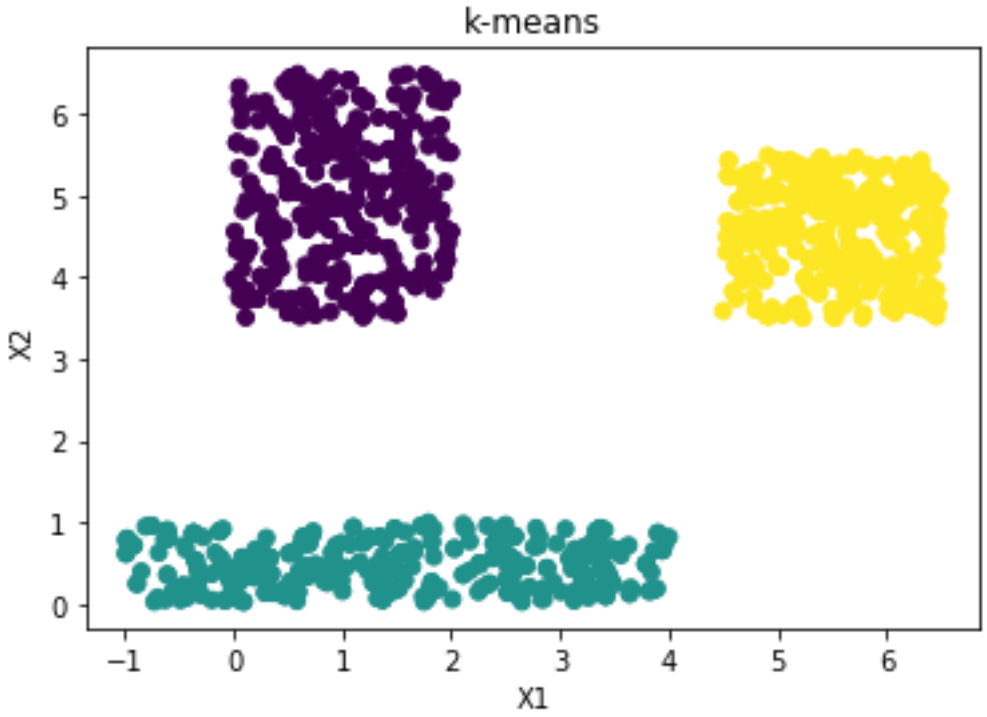
`clustering_1.csv`

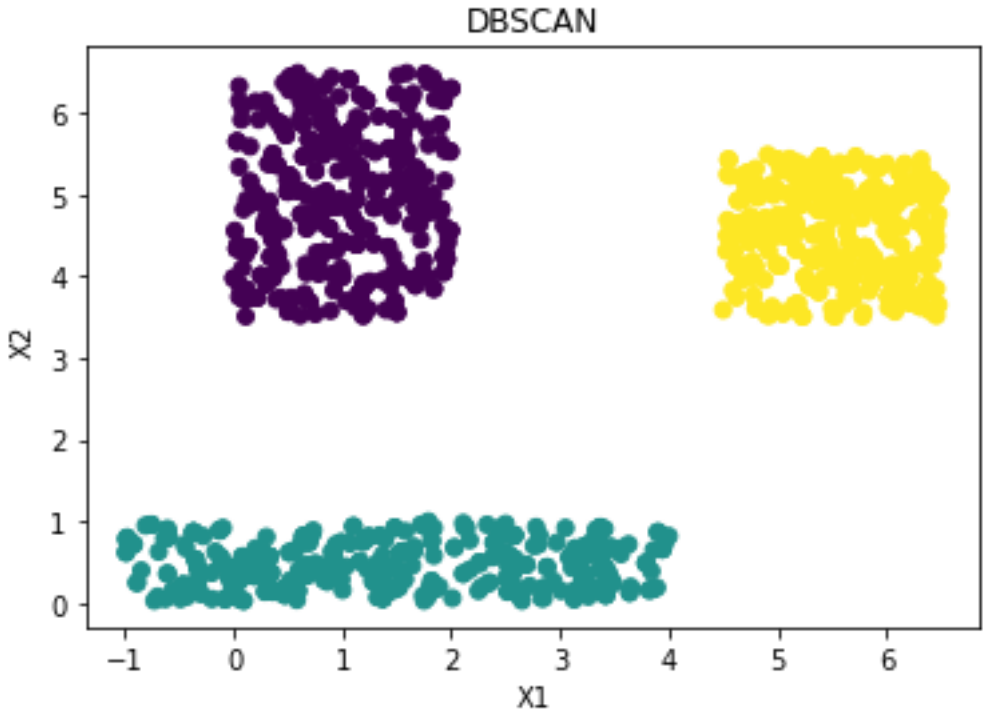


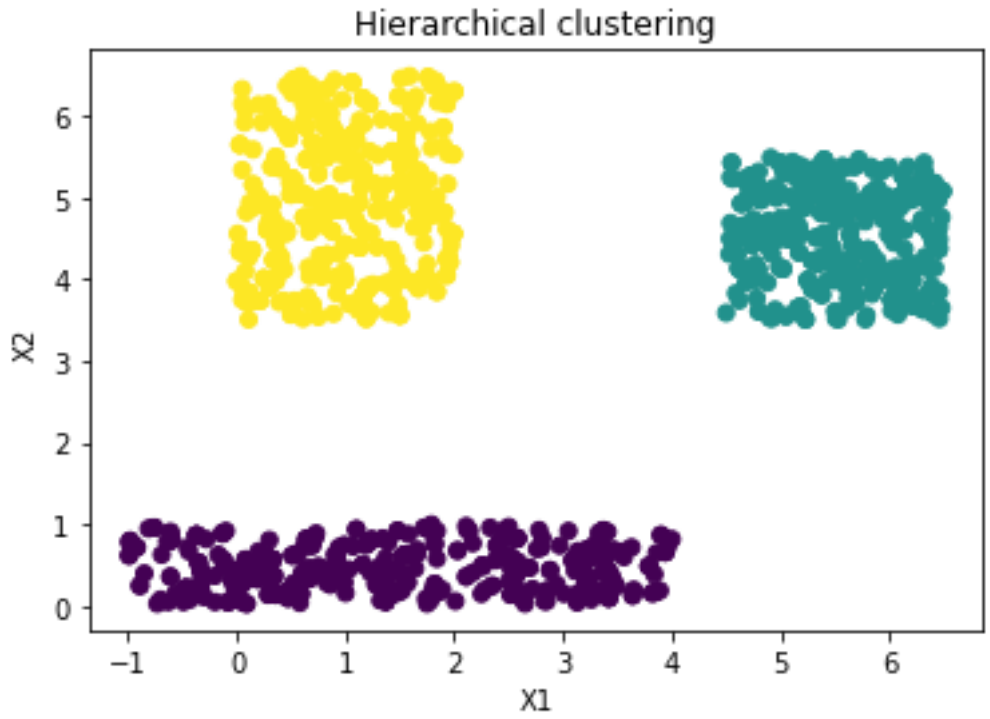




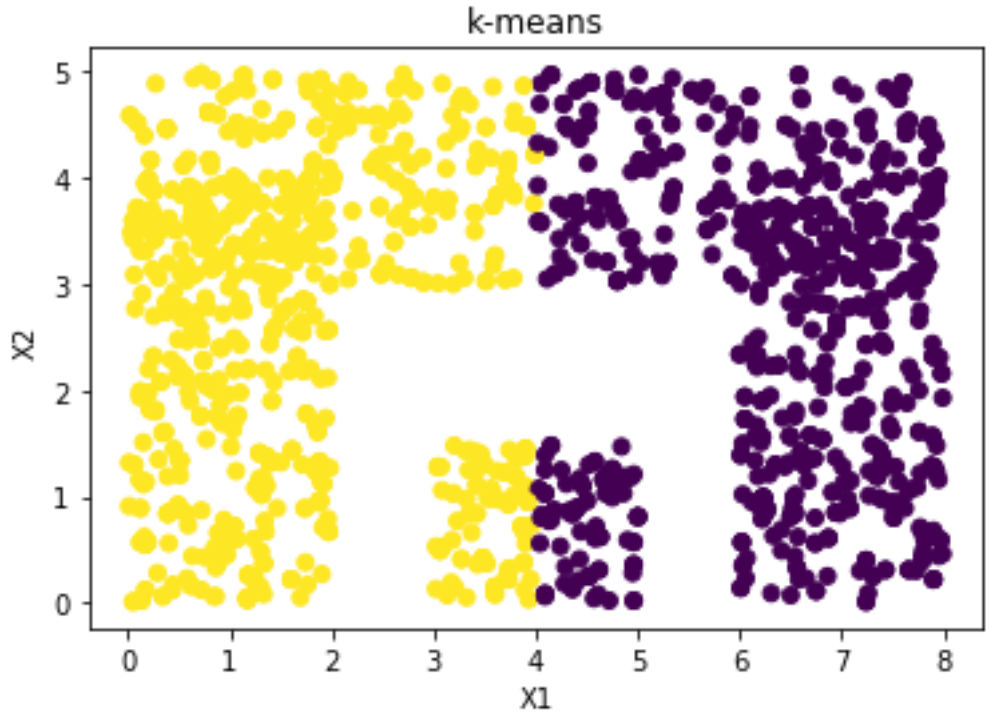
clustering_2.csv

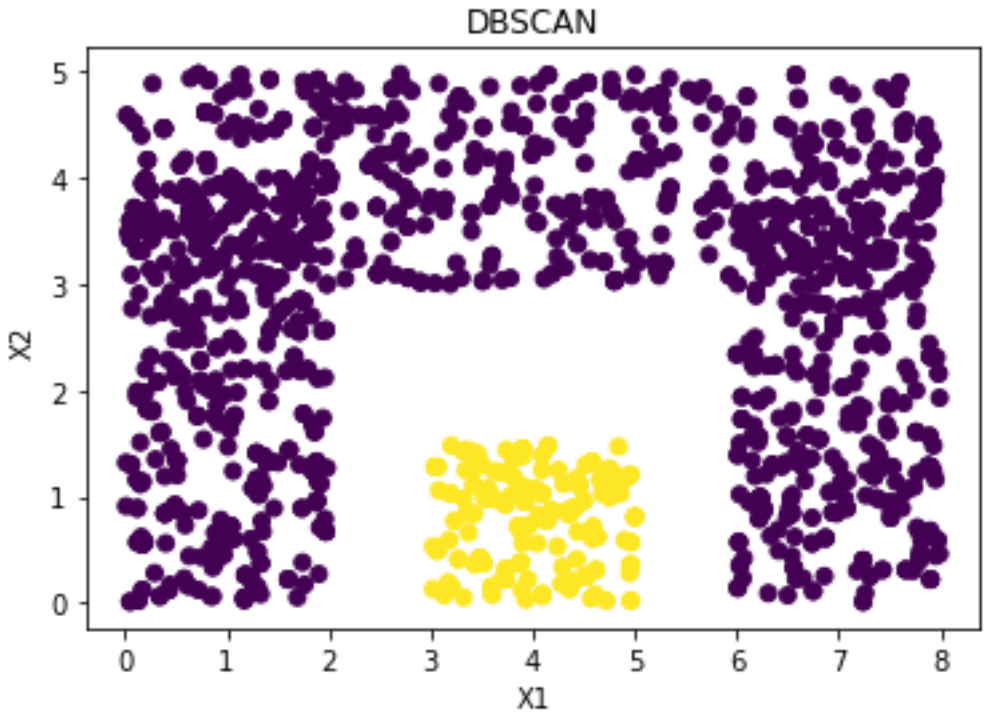

k-means
X2
X1
0
1
2
3
4
5
6
−1
0
1
2
3
4
5
6

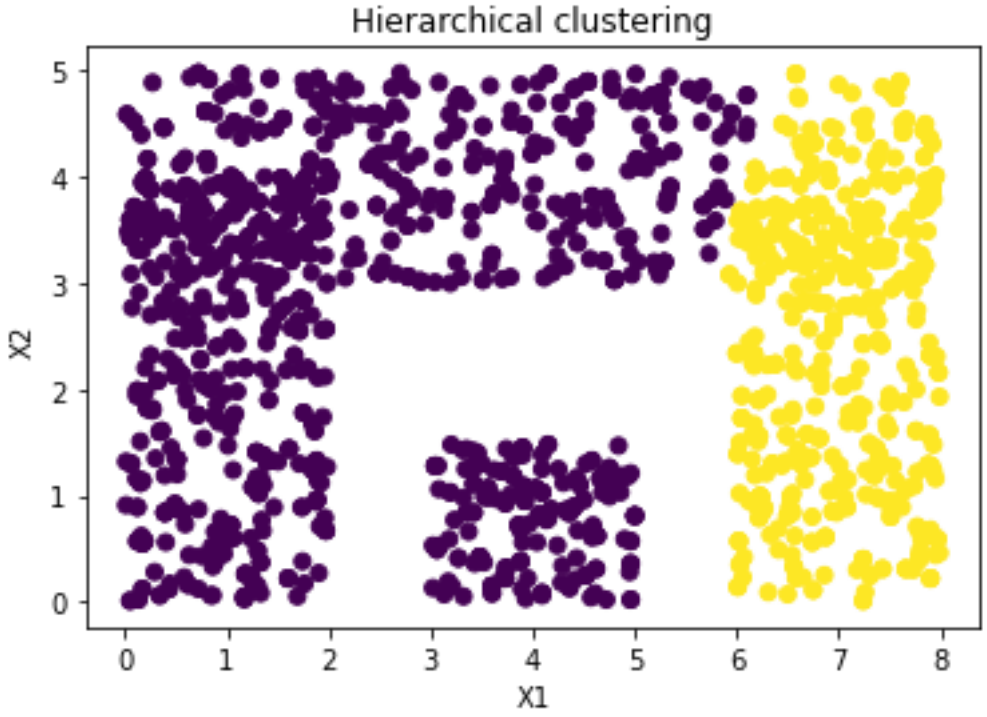

DBSCAN
X2
X1
0
1
2
3
4
5
6
−1
0
1
2
3
4
5
6



clustering_3.csv







В первом случае лучше всего справился метод к средних, чуть хуже иерархическая кластеризация (граница не совсем четкая) и DBSCAN вовсе не получил адекватного результата, поскольку он основан на различных плотностях в кластерах, а в этом наборе данных она одинакова везде.

Во втором случае все методы сработали хорошо, поскольку данные сильно отделены друг от друга.

В третьем случае отлично справился алгоритм DBSCAN, поскольку данные отделены друг от друга, между ними есть промежуток с маленькой плотностью. Метод к средних разделил данные пополам, что не совсем

ожидаемое разделение. Иерархическая кластеризация также плохо справилась с этим датасетом, поделила данные, не так как ожидалось.

Сжатие цветовой палитры

Для работы была выбран пейзаж с горами и озером.

Количество кластеров: 1

Количество кластеров: 2

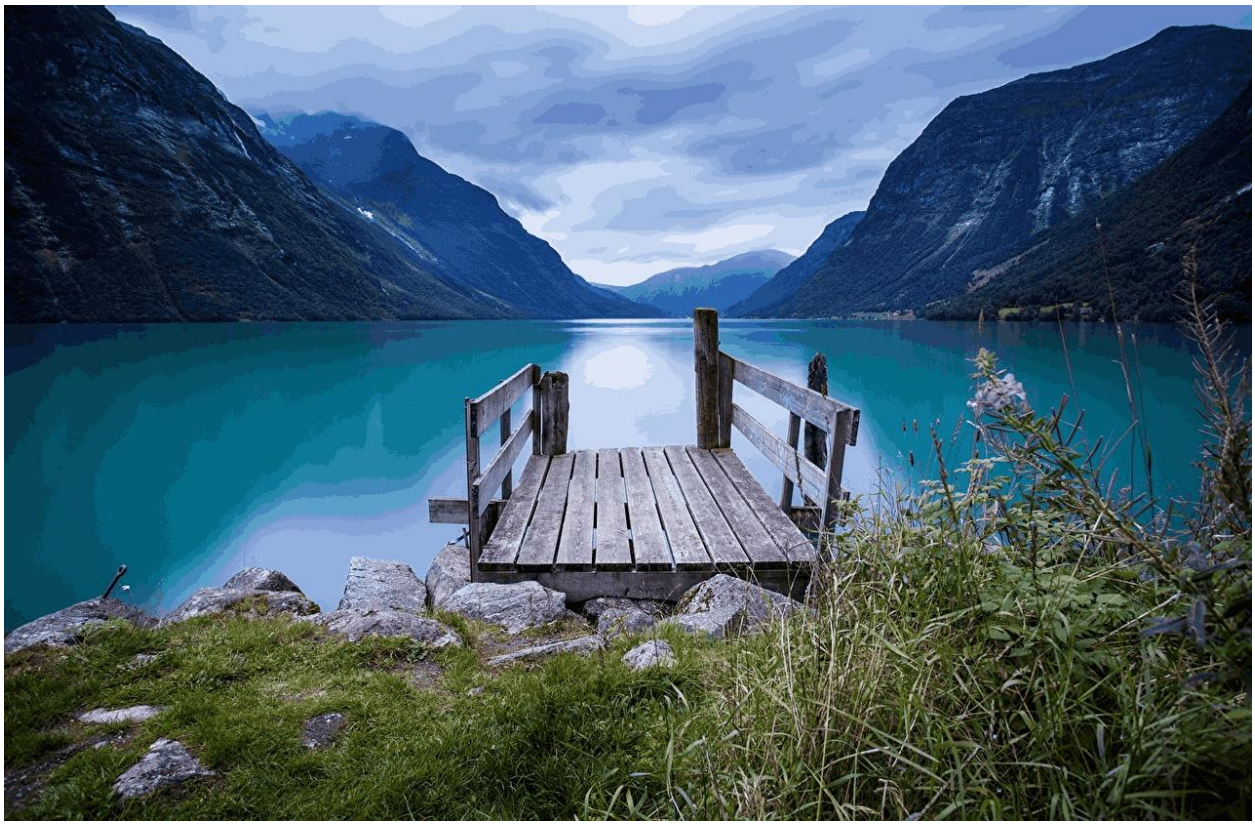
Количество кластеров: 4

Количество кластеров: 8

Количество кластеров: 16

Количество кластеров: 32

Количество кластеров: 64

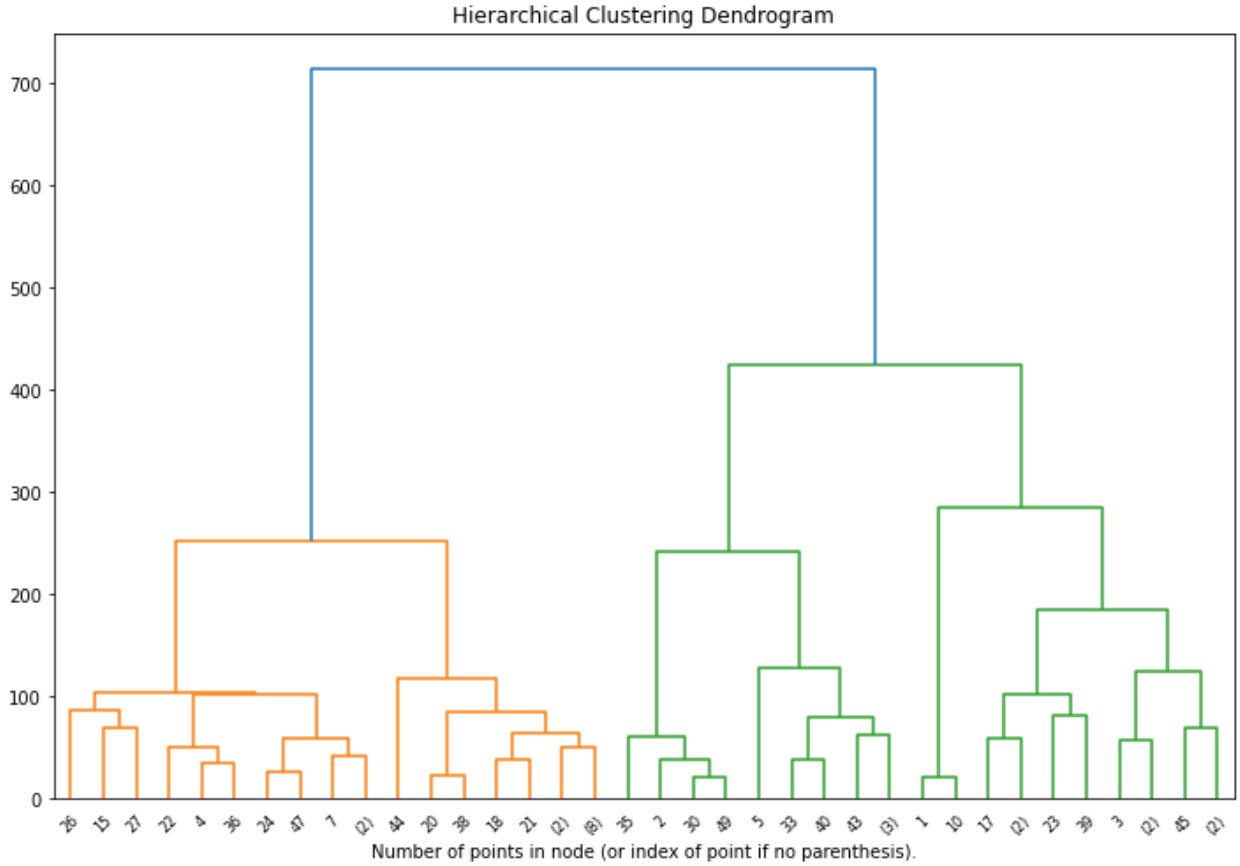


Количество кластеров: 128

Оригинал:

При разбиении на 128 кластеров видно, что не хватает оттенков для воды и неба. При каждых последующих изменениях вода и небо становятся более резкими, с большим количеством границ, более явно видны переходы между цветами. На 4 цветах картинка похожа на пикесль арт. При разбиении на 2 кластера изображение становится похоже на гравюры, при 1 сливается в один цвет.

Построение дендрограммы



Дендрограмма показывает на сколько близки между собой кластеры, на графике по горизонтали показаны количество штатов в кластерах, а по вертикали мера сходства. Мы видим разделение на два больших кластера слева (оранжевый) кластер где республиканцы проигрывали (выигрывали демократы), а справа там где республиканцы одерживали победу. Можно судить о том в каких штатах в среднем больше поддерживали республиканцев. Данные предоставлены за тот период, когда в США были серьезные проблемы с отношением белокожих американцев к темнокожим. Можно предположить, что в штатах, где доминировали республиканцы, люди больше хотели равенства прав, так как политика республиканцев была направлена на это.

Приложения

1_k_means.py

```
#!/usr/bin/env python
# coding: utf-8

# In[9]:


import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler


# In[11]:


data = pd.read_csv('pluton.csv')
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)


# In[25]:


for j, d in enumerate([data, scaled_data]):
    if j == 0:
        print('No standartization')
    else:
```

```
    print('With standartization')


for i in range(1, 11):
    est = KMeans(n_clusters=3, max_iter=i).fit(d)

    clusters = [0] * len(est.cluster_centers_)
    for l in range(len(est.cluster_centers_)):
        clusters[l] = np.count_nonzero(est.labels_ == l)

    print('Max iter =', i)
    print(clusters, '\n', est.cluster_centers_)
```

2_clustering.py

```
#!/usr/bin/env python
# coding: utf-8

# In[16]:


import numpy as np
import pandas as pd
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
```

```
# In[9]:


data_1 = pd.read_csv('clustering_1.csv', delimiter='\t')
data_2 = pd.read_csv('clustering_2.csv')
data_3 = pd.read_csv('clustering_3.csv')
data = ['clustering_1.csv', 'clustering_2.csv', 'clustering_3.csv']


# In[26]:


files = ['clustering_1.csv', 'clustering_2.csv', 'clustering_3.csv']

for f in files:
    data = pd.read_csv(f, delimiter='\t')
    print(f)

    kmeans_score = []
    hierarchical_score = []

    for k in range(2, 11):
        kmeans = KMeans(n_clusters=k).fit(data)
        hier = AgglomerativeClustering(n_clusters=k).fit(data)

        kmeans_score.append(silhouette_score(data, kmeans.labels_,
metric='euclidean'))
```

```
        hierarchical_score.append(silhouette_score(data, hier.labels_,
metric='euclidean'))


    plt.plot(range(2, 11), kmeans_score, label='k-means score')
    plt.plot(range(2, 11), hierarchical_score, label='hierarchical score')
    plt.legend(loc='best')
    plt.title('Choosing best cluster number')
    plt.xlabel('k')
    plt.ylabel('silhouette score')
    plt.show()



# In[15]:



clusters = [2, 3, 2]

for f, c in zip(files, clusters):
    data = pd.read_csv(f, delimiter='\t')
    print(f)
    estimators = (
        ('k-means', KMeans(n_clusters=c)),
        ('DBSCAN', DBSCAN()),
        ('Hierarchical clustering', AgglomerativeClustering(n_clusters=c))
    )

    for title, est in estimators:
        est.fit(data)
```

```
    plt.scatter(data.to_numpy()[:, 0], data.to_numpy()[:, 1], c=est.labels_)
    plt.title(title)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.show()
```

3_picture.py

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:



import cv2
import numpy as np
from sklearn.cluster import KMeans



# In[3]:



clusters = [2**i for i in range(8)]
file = 'img.jpg'

for c in clusters:
    img3d = cv2.imread(file)
    h, w, ch = img3d.shape
    img2d = img3d.reshape(h * w, ch)
```

```
    kmeans = KMeans(n_clusters=c).fit(img2d)

    labels = kmeans.labels_
    centers = kmeans.cluster_centers_
    for i in range(h*w):
        img2d[i] = centers[labels[i]].astype(np.int32)

    img3d = img2d.reshape(h, w, ch)
    #cv2.imshow(str(c) + ' clusters', img3d)
    cv2.imwrite(f"{c}_clusters.jpg", img3d)
```

4_dendrogramm.py

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:



import numpy as np
import pandas as pd
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering
from matplotlib import pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')



# In[13]:
```

```
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)



# In[21]:



data = pd.read_csv('votes.csv', na_values=["NA"])
```

```
data = data.fillna(0)


model = AgglomerativeClustering(distance_threshold=0,
n_clusters=None).fit(data)

plt.figure(figsize=(12, 8))

plt.title("Hierarchical Clustering Dendrogram")

plot_dendrogram(model, truncate_mode="level", p=5)

plt.xlabel("Number of points in node (or index of point if no parenthesis).")

plt.show()
```