### МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАНА

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

***Практическая работа***

***по предмету «Машинное обучение»***

Выполнил: студент гр.721-19 Хакимбеков Д Приняла: Маннапова Мафтуна

## Ташкент 2021

Необслуживаемые алгоритмы машинного обучения не имеют какого-либо руководителя, обеспечивающего какое-либо руководство. Вот почему они тесно связаны с тем, что некоторые называют истинным искусственным интеллектом.

При неконтролируемом обучении не будет правильного ответа и учителя для руководства. Алгоритмы должны обнаружить интересную закономерность в данных для обучения.

**Что такое кластеризация?**

По сути, это тип неконтролируемого метода обучения и распространенный метод статистического анализа данных, используемый во многих областях. Кластеризация главным образом является задачей разделения набора наблюдений на подмножества, называемые кластерами, таким образом, чтобы наблюдения в одном и том же кластере были похожи в одном смысле и не похожи на наблюдения в других кластерах. Проще говоря, мы можем сказать, что главная цель кластеризации — группировать данные на основе сходства и различий.

Например, следующая диаграмма показывает аналогичные данные в разных кластерах —

**Алгоритмы кластеризации данных**

Ниже приведены несколько распространенных алгоритмов кластеризации данных.

**Алгоритм K-средних**

Алгоритм кластеризации K-средних является одним из известных алгоритмов кластеризации данных. Нужно предположить, что номера кластеров уже известны. Это также называется плоской кластеризацией. Это алгоритм итеративной кластеризации. Для этого алгоритма необходимо выполнить следующие шаги:

**Шаг 1** — Нам нужно указать желаемое количество K подгрупп.

**Шаг 2** — Зафиксируйте количество кластеров и случайным образом назначьте каждую точку данных кластеру. Или, другими словами, нам нужно классифицировать наши данные на основе количества кластеров.

На этом этапе кластерные центроиды должны быть вычислены.

Поскольку это итеративный алгоритм, нам нужно обновлять местоположения K центроидов с каждой итерацией, пока мы не найдем глобальные оптимумы или, другими словами, центроиды достигают в своих оптимальных местоположениях.

Следующий код поможет в реализации алгоритма кластеризации K-средних в Python. Мы собираемся использовать модуль Scikit-learn.

Давайте импортируем необходимые пакеты —

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import numpy as np

from sklearn.cluster import KMeans

Следующая строка кода поможет в создании двумерного набора данных, содержащего четыре **больших объекта** , с помощью **make\_blob** из пакета **sklearn.dataset** .

from sklearn.datasets.samples\_generator import make\_blobs

X, y\_true = make\_blobs(n\_samples = 500, centers = 4,

cluster\_std = 0.40, random\_state = 0)

Мы можем визуализировать набор данных, используя следующий код —

plt.scatter(X[:, 0], X[:, 1], s = 50);

plt.show()

Здесь мы инициализируем kmeans в качестве алгоритма KMeans с обязательным параметром количества кластеров (n\_clusters).

kmeans = KMeans(n\_clusters = 4)

Нам нужно обучить модель K-средних с входными данными.

kmeans.fit(X)

y\_kmeans = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c = y\_kmeans, s = 50, cmap = 'viridis')

centers = kmeans.cluster\_centers\_

Код, приведенный ниже, поможет нам построить и визуализировать результаты работы машины на основе наших данных, а также в соответствии с количеством найденных кластеров.

plt.scatter(centers[:, 0], centers[:, 1], c = 'black', s = 200, alpha = 0.5);

plt.show()

**Алгоритм среднего смещения**

Это еще один популярный и мощный алгоритм кластеризации, используемый в обучении без учителя. Он не делает никаких предположений, следовательно, это непараметрический алгоритм. Это также называется иерархической кластеризацией или кластерным анализом среднего сдвига. Ниже приведены основные шаги этого алгоритма —

* Прежде всего, нам нужно начать с точек данных, назначенных на собственный кластер.
* Теперь он вычисляет центроиды и обновляет местоположение новых центроидов.
* Повторяя этот процесс, мы приближаемся к вершине кластера, т.е. к области более высокой плотности.
* Этот алгоритм останавливается на стадии, когда центроиды больше не двигаются.

Прежде всего, нам нужно начать с точек данных, назначенных на собственный кластер.

Теперь он вычисляет центроиды и обновляет местоположение новых центроидов.

Повторяя этот процесс, мы приближаемся к вершине кластера, т.е. к области более высокой плотности.

Этот алгоритм останавливается на стадии, когда центроиды больше не двигаются.

С помощью следующего кода мы реализуем алгоритм кластеризации Mean Shift в Python. Мы собираемся использовать модуль Scikit-learn.

Давайте импортируем необходимые пакеты —

import numpy as np

from sklearn.cluster import MeanShift

import matplotlib.pyplot as plt

from matplotlib import style

style.use("ggplot")

Следующий код поможет в создании двумерного набора данных, содержащего четыре **больших объекта** , с помощью **make\_blob** из пакета **sklearn.dataset** .

from sklearn.datasets.samples\_generator import make\_blobs

Мы можем визуализировать набор данных с помощью следующего кода

centers = [[2,2],[4,5],[3,10]]

X, \_ = make\_blobs(n\_samples = 500, centers = centers, cluster\_std = 1)

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

plt.show()

Теперь нам нужно обучить кластерную модель Mean Shift с использованием входных данных.

ms = MeanShift()

ms.fit(X)

labels = ms.labels\_

cluster\_centers = ms.cluster\_centers\_

Следующий код напечатает центры кластеров и ожидаемое количество кластеров согласно входным данным —

print(cluster\_centers)

n\_clusters\_ = len(np.unique(labels))

print("Estimated clusters:", n\_clusters\_)

[[ 3.23005036 3.84771893]

[ 3.02057451 9.88928991]]

Estimated clusters: 2

Код, приведенный ниже, поможет построить и визуализировать результаты работы машины на основе наших данных, а также соответствия в соответствии с количеством найденных кластеров.

colors = 10\*['r.','g.','b.','c.','k.','y.','m.']

for i in range(len(X)):

plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize = 10)

plt.scatter(cluster\_centers[:,0],cluster\_centers[:,1],

marker = "x",color = 'k', s = 150, linewidths = 5, zorder = 10)

plt.show()

**Измерение производительности кластеризации**

Данные реального мира не организованы естественным образом в несколько отличительных кластеров. По этой причине нелегко визуализировать и делать выводы. Вот почему нам необходимо измерять производительность кластеризации, а также ее качество. Это можно сделать с помощью анализа силуэта.

**Анализ силуэта**

Этот метод может использоваться для проверки качества кластеризации путем измерения расстояния между кластерами. По сути, это дает возможность оценить такие параметры, как количество кластеров, путем оценки силуэта. Эта оценка является метрикой, которая измеряет, насколько близко каждая точка в одном кластере находится к точкам в соседних кластерах.

**Анализ силуэта баллов**

Счет имеет диапазон [-1, 1]. Ниже приводится анализ этой оценки —

* **Оценка +1** — Оценка около +1 означает, что выборка находится далеко от соседнего кластера.
* **Оценка 0** — Оценка 0 указывает на то, что выборка находится или очень близка к границе решения между двумя соседними кластерами.
* **Оценка -1** — Отрицательная оценка означает, что выборки были назначены в неправильные кластеры.

**Оценка +1** — Оценка около +1 означает, что выборка находится далеко от соседнего кластера.

**Оценка 0** — Оценка 0 указывает на то, что выборка находится или очень близка к границе решения между двумя соседними кластерами.

**Оценка -1** — Отрицательная оценка означает, что выборки были назначены в неправильные кластеры.

**Расчет силуэта**

В этом разделе мы узнаем, как рассчитать оценку силуэта.

Оценка силуэта может быть рассчитана по следующей формуле:

силуэтсчет= frac left(pq right)max left(p,q right)

Здесь ? — среднее расстояние до точек в ближайшем кластере, частью которых точка данных не является. И, ? — среднее расстояние внутри кластера до всех точек его собственного кластера.

Чтобы найти оптимальное количество кластеров, нам нужно снова запустить алгоритм кластеризации, импортировав модуль **метрик** из пакета **sklearn** . В следующем примере мы запустим алгоритм кластеризации K-средних, чтобы найти оптимальное количество кластеров:

Импортируйте необходимые пакеты, как показано на рисунке —

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import numpy as np

from sklearn.cluster import KMeans

С помощью следующего кода мы сгенерируем двумерный набор данных, содержащий четыре **больших объекта,** используя **make\_blob** из пакета **sklearn.dataset** .

from sklearn.datasets.samples\_generator import make\_blobs

X, y\_true = make\_blobs(n\_samples = 500, centers = 4, cluster\_std = 0.40, random\_state = 0)

Инициализируйте переменные как показано —

scores = []

values = np.arange(2, 10)

Нам нужно пройти через модель K-средних по всем значениям, а также обучить ее входным данным.

for num\_clusters in values:

kmeans = KMeans(init = 'k-means++', n\_clusters = num\_clusters, n\_init = 10)

kmeans.fit(X)

Теперь оцените оценку силуэта для текущей модели кластеризации, используя евклидову метрику расстояния —

score = metrics.silhouette\_score(X, kmeans.labels\_,

metric = 'euclidean', sample\_size = len(X))

Следующая строка кода поможет в отображении количества кластеров, а также показателя Силуэт.

print("\nNumber of clusters =", num\_clusters)

print("Silhouette score =", score)

scores.append(score)

Вы получите следующий вывод —

Number of clusters = 9

Silhouette score = 0.340391138371

num\_clusters = np.argmax(scores) + values[0]

print('\nOptimal number of clusters =', num\_clusters)

Теперь вывод для оптимального количества кластеров будет следующим:

Optimal number of clusters = 2

**Нахождение ближайших соседей**

Если мы хотим создать рекомендательные системы, такие как система рекомендования фильмов, то нам нужно понять концепцию поиска ближайших соседей. Это потому, что система рекомендации использует концепцию ближайших соседей.

**Концепция нахождения ближайших соседей** может быть определена как процесс нахождения ближайшей точки к точке входа из данного набора данных. Основное использование этого алгоритма KNN) K-ближайших соседей) заключается в создании систем классификации, которые классифицируют точку данных о близости точки входных данных к различным классам.

Код Python, приведенный ниже, помогает найти K-ближайших соседей заданного набора данных —

Импортируйте необходимые пакеты, как показано ниже. Здесь мы используем модуль **NearestNeighbors** из пакета **sklearn**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import NearestNeighbors

Давайте теперь определим входные данные —

A = np.array([[3.1, 2.3], [2.3, 4.2], [3.9, 3.5], [3.7, 6.4], [4.8, 1.9],

[8.3, 3.1], [5.2, 7.5], [4.8, 4.7], [3.5, 5.1], [4.4, 2.9],])

Теперь нам нужно определить ближайших соседей —

k = 3

Нам также нужно предоставить тестовые данные, из которых можно найти ближайших соседей —

test\_data = [3.3, 2.9]

Следующий код может визуализировать и построить входные данные, определенные нами —

plt.figure()

plt.title('Input data')

plt.scatter(A[:,0], A[:,1], marker = 'o', s = 100, color = 'black')

Теперь нам нужно построить ближайший сосед. Объект также должен быть обучен

knn\_model = NearestNeighbors(n\_neighbors = k, algorithm = 'auto').fit(X)

distances, indices = knn\_model.kneighbors([test\_data])

Теперь мы можем напечатать K ближайших соседей следующим образом

print("\nK Nearest Neighbors:")

for rank, index in enumerate(indices[0][:k], start = 1):

print(str(rank) + " is", A[index])

Мы можем визуализировать ближайших соседей вместе с тестовой точкой данных

plt.figure()

plt.title('Nearest neighbors')

plt.scatter(A[:, 0], X[:, 1], marker = 'o', s = 100, color = 'k')

plt.scatter(A[indices][0][:][:, 0], A[indices][0][:][:, 1],

marker = 'o', s = 250, color = 'k', facecolors = 'none')

plt.scatter(test\_data[0], test\_data[1],

marker = 'x', s = 100, color = 'k')

plt.show()

**Выход**

**K Ближайшие соседи**

1 is [ 3.1 2.3]

2 is [ 3.9 3.5]

3 is [ 4.4 2.9]

**Классификатор ближайших соседей**

Классификатор K-ближайших соседей (KNN) — это классификационная модель, которая использует алгоритм ближайших соседей для классификации данной точки данных. Мы реализовали алгоритм KNN в последнем разделе, а теперь мы собираемся построить классификатор KNN, используя этот алгоритм.

**Концепция классификатора КНН**

Основная концепция классификации K-ближайших соседей состоит в том, чтобы найти заранее определенное число, т. Е. «K» — обучающих выборок, ближайших по расстоянию к новой выборке, которую необходимо классифицировать. Новые образцы получат свою этикетку от самих соседей. Классификаторы KNN имеют фиксированную пользовательскую константу для числа соседей, которые должны быть определены. Для расстояния стандартное евклидово расстояние является наиболее распространенным выбором. Классификатор KNN работает непосредственно с изученными образцами, а не создает правила обучения. Алгоритм KNN является одним из самых простых алгоритмов машинного обучения. Это было довольно успешно в большом количестве проблем классификации и регрессии, например, распознавания символов или анализа изображений.

**пример**

Мы строим классификатор KNN для распознавания цифр. Для этого мы будем использовать набор данных MNIST. Мы напишем этот код в блокноте Jupyter.

Импортируйте необходимые пакеты, как показано ниже.

Здесь мы используем модуль **KNeighborsClassifier** из пакета **sklearn.neighbors** —

from sklearn.datasets import \*

import pandas as pd

%matplotlib inline

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

import numpy as np

Следующий код будет отображать изображение цифры, чтобы проверить, какое изображение мы должны проверить —

def Image\_display(i):

plt.imshow(digit['images'][i],cmap = 'Greys\_r')

plt.show()

Теперь нам нужно загрузить набор данных MNIST. На самом деле всего 1797 изображений, но мы используем первые 1600 изображений в качестве обучающего образца, а оставшиеся 197 будут сохранены для целей тестирования.

digit = load\_digits()

digit\_d = pd.DataFrame(digit['data'][0:1600])

Теперь при отображении изображений мы можем видеть вывод следующим образом:

Image\_display(0)

**Image\_display (0)**

Изображение 0 отображается следующим образом —

**Image\_display (9)**

Изображение 9 отображается следующим образом —

**digit.keys ()**

Теперь нам нужно создать набор данных обучения и тестирования и предоставить набор данных тестирования для классификаторов KNN.

train\_x = digit['data'][:1600]

train\_y = digit['target'][:1600]

KNN = KNeighborsClassifier(20)

KNN.fit(train\_x,train\_y)

Следующий вывод создаст конструктор классификатора ближайшего соседа —

KNeighborsClassifier(algorithm = 'auto', leaf\_size = 30, metric = 'minkowski',

metric\_params = None, n\_jobs = 1, n\_neighbors = 20, p = 2,

weights = 'uniform')

Нам нужно создать тестовый образец, указав любое произвольное число больше 1600, которое было обучающим образцом.

test = np.array(digit['data'][1725])

test1 = test.reshape(1,-1)

Image\_display(1725)

**Image\_display (6)**

Изображение 6 отображается следующим образом —

Теперь мы будем прогнозировать данные теста следующим образом —

KNN.predict(test1)

Приведенный выше код сгенерирует следующий вывод:

array([6])

Теперь рассмотрим следующее —

digit['target\_names']

### Машинное обучение с Python : кластерный анализ

 В этой статьей разберем несколько самых популярных методов кластеризации и методов оценки их эффективности. Для начала сгенерируем тестовый набор данных и рассмотрим способы его  визуализации.  Для создания тестового набора данных используем функцию make\_blobs. Задаем количество элементов 100 и количество кластеров 4. Каждый элемент имеет два показателя для того, чтобы могли реализовать двумерный график. Кластеры имеют разные стандартное отклонение, то есть некоторые кластеры более рассеиваются, чем другие. Функция также возвращает метки кластеров, они понадобятся, чтобы проверить качество работы алгоритма.

***import mglearn  
import matplotlib  
import matplotlib.pyplot as plt  
import pandas as pd  
%matplotlib inline***

***from sklearn.datasets import make\_blobs***

***blobs = make\_blobs(n\_samples=100,***

***n\_features=2,***

***centers=4,***

***cluster\_std=[1,1.5, 2,2],***

***shuffle=False,***

***random\_state=7)***

***print("{},{},{},\n{}".format(type(blobs),len(blobs),blobs[0][:5],blobs[1]))***

**<class 'tuple'>,2,[[ 0.46546494 3.12315514]**

**[-3.83396959 8.36956175]**

**[ 1.7373078 4.42546234]**

**[ 1.1312175 4.68194985]**

**[ 2.20656076 5.50616718]],**

**[0 3 0 0 0 0 2 3 0 3 3 3 3 3 3 1 1 2 2 1 0 3 2 1 0 2 2 0 1 1 1 3 1 1 2 0 3**

**1 3 2 0 2 3 2 2 3 1 2 0 0 0 1 2 2 2 3 3 1 1 3 3 1 1 0 1 3 2 2 1 0 3 1 0 3**

**0 0 2 2 1 1 1 3 2 0 1 2 1 1 0 0 0 2 0 2 2 3 3 2 3 0]**

В качестве объекта мы получили тьюпл, состоящий из двух элементов. Первый элемент — это непосредственно набор данных, список описаний наших объектов. И второй элемент — это метки классов. Мы видим, что каждый объект представляется списком из двух элементов. Первый элемент — это x координата, второй элемент — это y координата. И метки классов, проставляются как цифры 0,1,2,3.

***print("features: {}".format(blobs[0][:10]))***

***print("target: {}".format(blobs[1][:10]))***

**features: [[ 0.46546494 3.12315514]**

**[-3.83396959 8.36956175]**

**[ 1.7373078 4.42546234]**

**[ 1.1312175 4.68194985]**

**[ 2.20656076 5.50616718]**

**[ 0.58894326 4.00148458]**

**[-3.24935538 6.73801217]**

**[ 0.65058584 8.0105625 ]**

**[-0.73000011 6.25456272]**

**[-1.98661945 7.35670166]]**

**target: [0 3 0 0 0 0 2 3 0 3]**

Перенесем эти элементы в два объекта, X и y

***X,y=blobs***

***print("{},{},{},{}".format(type(X),X.shape,type(y),y.shape))***

**class 'numpy.ndarray'>,(100, 2),<class 'numpy.ndarray'>,(100,)**

И сделаем из них DataFrame

***df\_blobs = pd.DataFrame(***

***{***

***'x1': X[:,0],***

***'x2': X[:,1],***

***'y': y***

***}***

***)***

***df\_blobs.head()***

**x1            x2        y**

**0 -8.474725 3.843652 0  
1 -7.456176 6.198874 0  
2 -9.099263 5.426828 0  
3 -7.968535 5.337019 0  
4 -8.716583 4.145134 0**

При визуализации будем также использовать библиотеку mglearn

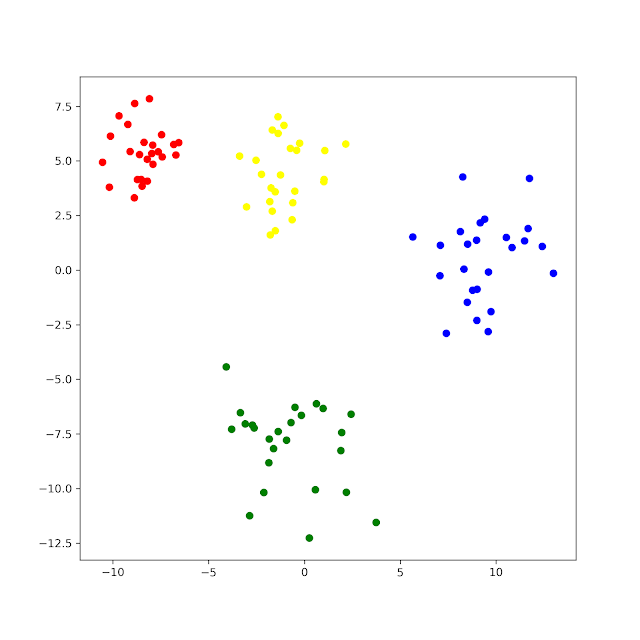
Определим следующую функцию для графического представления кластеров в двумерной плоскости. Удобная функция, которая выдает название графика и количество кластеров. Каждая точка данных будет отмечена в соответствии с ее меткой.

***def plot\_2d\_clusters(x, y, ax):  
    y\_uniques = pd.Series(y).unique()  
    for y\_unique\_item in y\_uniques:  
        x[y == y\_unique\_item].plot(  
        title=f'{len(y\_uniques)} Clusters',  
        kind='scatter',  
        x='x1', y='x2',  
        marker=f'***yuniqueitem***',  
        ax=ax,  
)***

Рассмотрим несколько вариантов визуализации, которые будем использовать в дальнейшем

Вариант 1

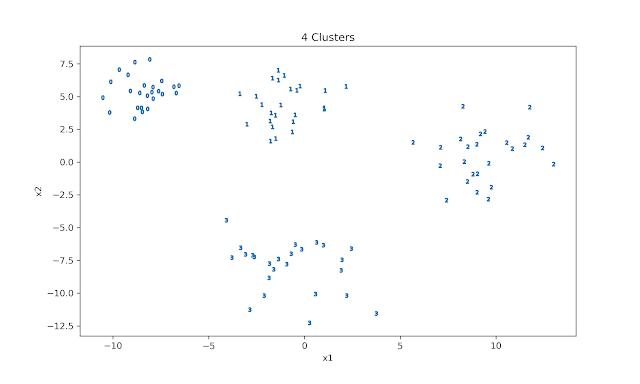
***from matplotlib.colors import ListedColormap  
plt.figure (figsize = (8, 8))   
colors = ListedColormap(['red', 'yellow','blue','green'])  
plt.scatter(X[:, 0], X[:, 1], c = y, cmap = colors)***

[](https://1.bp.blogspot.com/-A4Z9xHONu0U/X79LVtN6LZI/AAAAAAAAAb8/6cRTqqhXa7UO74hRcD_t0qpmvSd09dtlwCLcBGAsYHQ/s2048/Ris1.png)

В этом варианте используем функцию **scatter**. Эта функция позволяет нам вывести точки, зная их x и y координаты. На входе нужно подать два списка: первый — список x координат, второй — список y координат.  Дальше нам нужно указать, какими цветами рисовать объекты. Для этого используем объект colormap (цветовые карта), который создается функцией **ListedColormap**.   Создаем и говорим, что у нас есть всего 4 цвета, в данном случае красный, желтый, синий и зеленый ,при этом объекты с меткой 0 будут иметь красный цвет, с меткой 1 — желтый, 2 - синий, 3 - зеленый.

Вариант 2 с помощью определенной выше функции

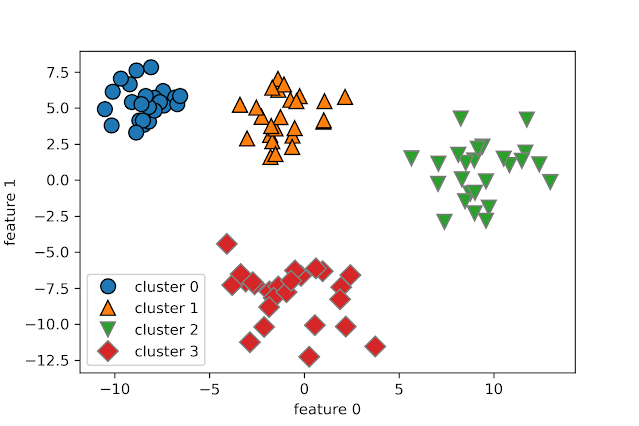
***fig, ax = plt.subplots(1, 1, figsize=(10, 6))  
x, y = df\_blobs[['x1', 'x2']], df\_blobs['y']  
plot\_2d\_clusters(x, y, ax)***

[](https://1.bp.blogspot.com/-Nqs8BFKrCGI/X79NVKOP6DI/AAAAAAAAAcI/3KqHnPnnzS4utPkiGBypvRU3j35zk0eUQCLcBGAsYHQ/s2048/Ris2.png)

Вариант 3 с помощью пакета mglearn

***mglearn.discrete\_scatter(X[:, 0], X[:, 1], y)  
plt.legend(["cluster 0", "cluster 1", "cluster 2","cluster 3"], loc='best')***

***plt.xlabel("feature 0")  
plt.ylabel("feature 1")***

[](https://1.bp.blogspot.com/-SZbMPwZeH0s/X7--uC5e2pI/AAAAAAAAAeo/3tjBaaRHWGojm90KyKEJXOpAVHIaAPJowCLcBGAsYHQ/s1800/Ris3.png)

Кластеризация k-средних – один из самых простых и наиболее часто используемых алгоритмов кластеризации Применить алгоритм k-средних, воспользовавшись библиотекой scikit-learn, довольно просто. Здесь мы применяем его к синтетическим данным, которые использовали для построения предыдущих графиков. Мы создаем экземпляр класса KMeans и задаем количество выделяемых кластеров.

Для кластеризации используем метод K-means, задаем количество кластеров = 5

***from sklearn.cluster import KMeans  
kmeans = KMeans(n\_clusters=5, random\_state=7)  
x, y = df\_blobs[['x1', 'x2']], df\_blobs['y']  
y\_pred = kmeans.fit\_predict(x)***

Cравнениваем наши предсказания c оригинальным метками :

***fig, axs = plt.subplots(1, 2, figsize=(14, 6))***

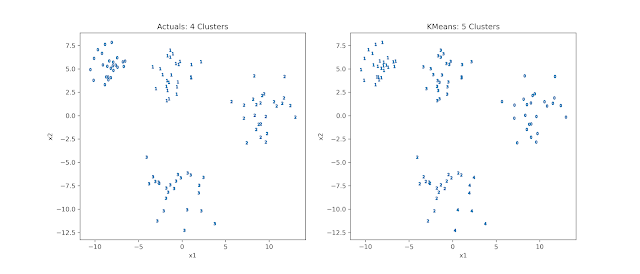
***x, y = df\_blobs[['x1', 'x2']], df\_blobs['y']***

***plot\_2d\_clusters(x, y, axs[0])***

***plot\_2d\_clusters(x, y\_pred, axs[1])***

***axs[0].set\_title(f'Actuals: {axs[0].get\_title()}')***

***axs[1].set\_title(f'KMeans: {axs[1].get\_title()}')***

[](https://1.bp.blogspot.com/-sNC0mTjFABA/X79RAraZt4I/AAAAAAAAAcg/OvRkhIgLFqkjhmjmMT02WGKtsdza3WMcgCLcBGAsYHQ/s2709/Ris4.png)

Один из первоначальных четырех кластеров был разделен на два, так как мы установили K на пять. И все же, как мы определяем значение K? У нас нет другого выбора, кроме как запустить алгоритм несколько раз с разным количеством кластеров и выбрать лучший. В следующем фрагменте кода мы перебираем три различных значения для n\_clusters. У нас также есть доступ к конечным центроидам, которые вычисляются для каждого кластера после того, как алгоритм сходится. Просмотр этих центроидов проясняет, как алгоритм назначил каждую точку данных своему собственному кластеру. В последней строке нашего фрагмента кода используется треугольный маркер для построения центроидов на каждом из трех графиков:

# Алгоритмы K-ближайших соседей и K-средних на Python

Одно из самых популярных приложений машинного обучения — решение задач классификации. Задачи классификации — это ситуации, когда у вас есть набор данных, и вы хотите классифицировать наблюдения из этого набора в определенную категорию.

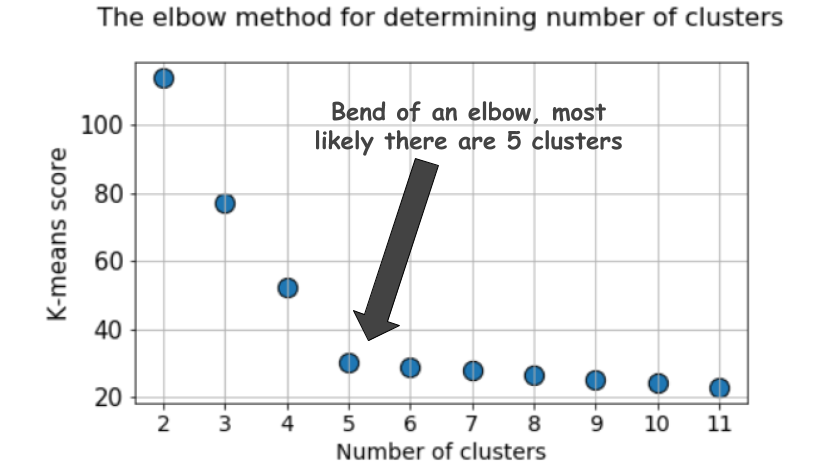
Известный пример — спам-фильтр для электронной почты. Gmail использует методы машинного обучения с учителем, чтобы автоматически помещать электронные письма в папку для спама в зависимости от их содержания, темы и других характеристик.

Две модели машинного обучения выполняют большую часть работы, когда дело доходит до задач классификации:

* Метод K-ближайших соседей
* Метод К-средних

Из этого руководства вы узнаете, как применять алгоритмы K-ближайших соседей и K-средних в коде на Python.

# **Метод локтя**

При кластеризации методом k-средних количество кластеров чаще всего оценивают с помощью «метода логтя». Он подразумевает многократное циклическое исполнение алгоритма с увеличением количества выбираемых кластеров, а также последующим откладыванием на графике балла кластеризации, вычисленного как функция от количества кластеров.  
  
Что это за балл, или метрика, которая откладывается на графике? Почему называют методом *локтя*?  
  
Характерный график выглядит так:  
  
  
  
Балл, как правило, является мерой входных данных по целевой функции k-средних, то есть некой формой отношения внутрикластерного расстояния к межкластерному расстоянию.

# **Силуэт — более подходящая метрика**

Коэффициент «силуэт» вычисляется с помощью среднего внутрикластерного расстояния (a) и среднего расстояния до ближайшего кластера (b) по каждому образцу. Силуэт вычисляется как (b - a) / max(a, b). Поясню: b — это расстояние между a и ближайшим кластером, в который a не входит. Можно вычислить среднее значение силуэта по всем образцам и использовать его как метрику для оценки количества кластеров.

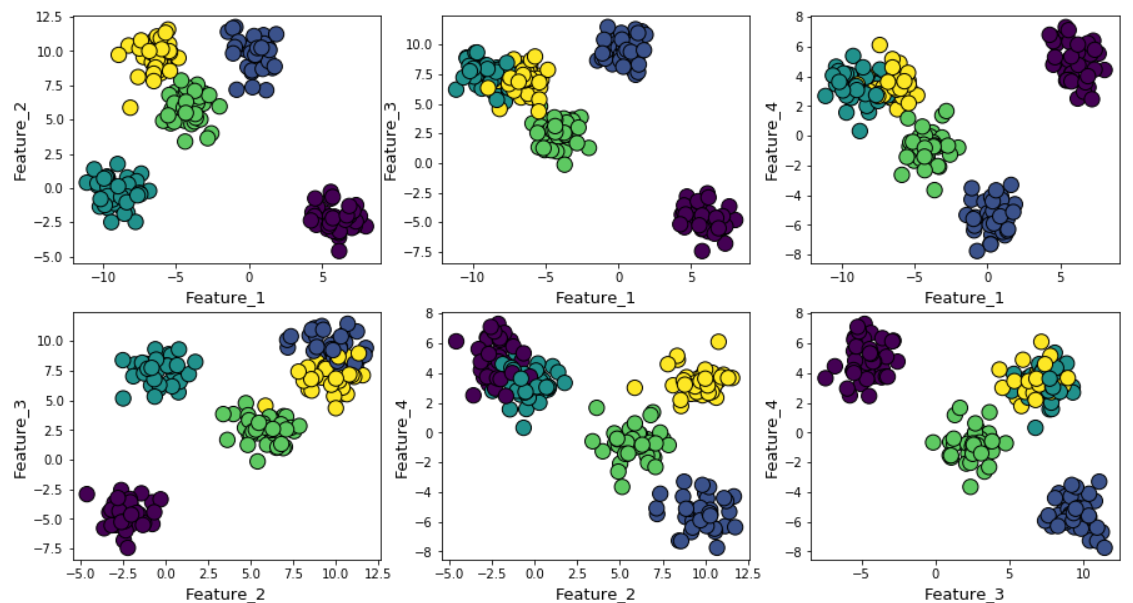
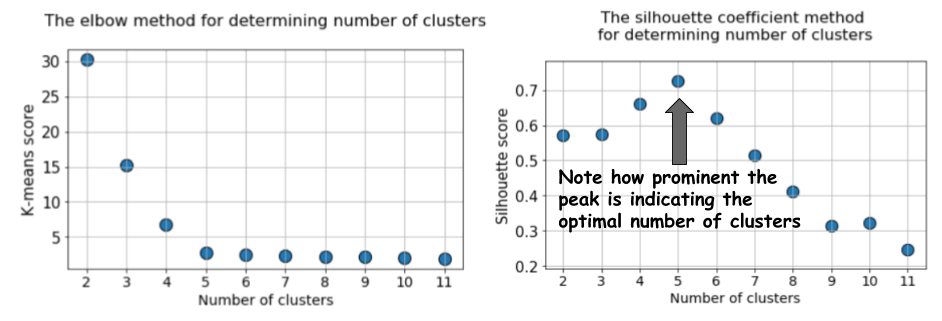
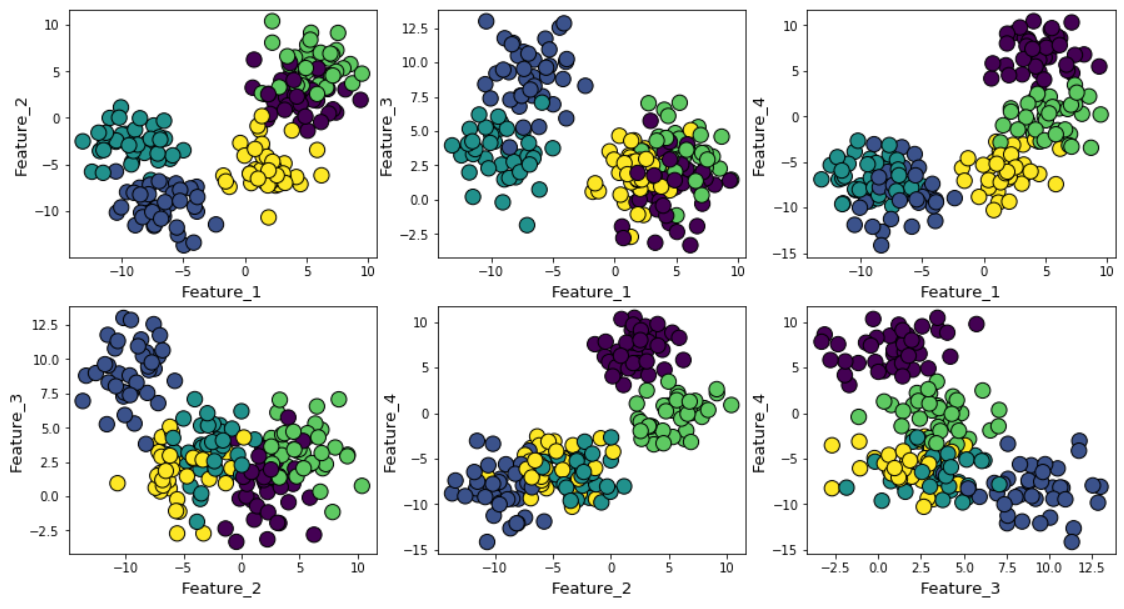
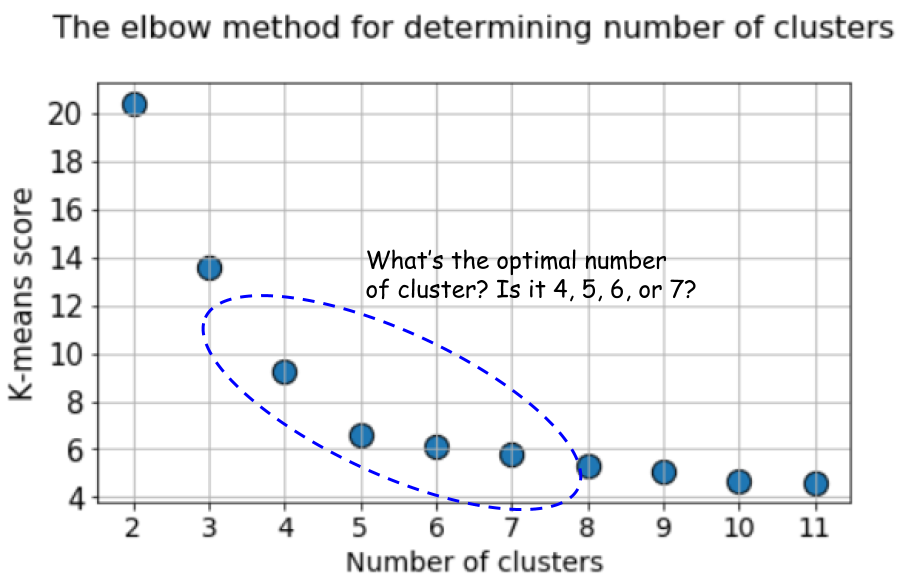
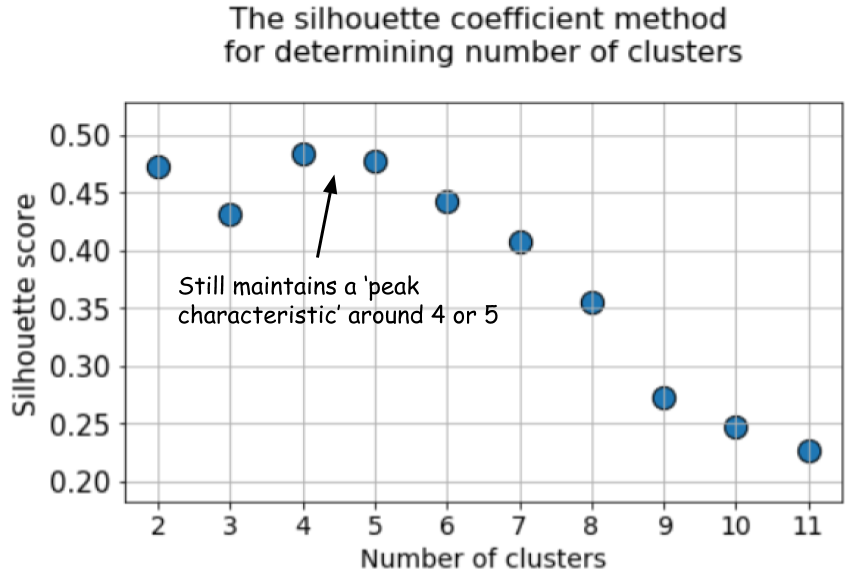
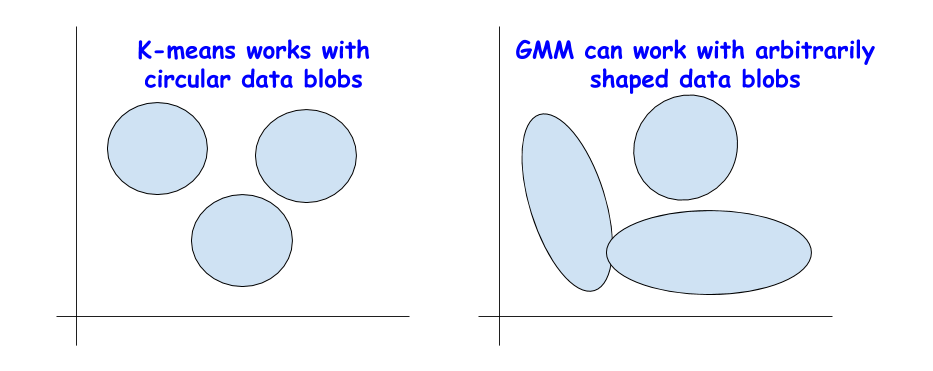
Допустим, мы сгенерировали случайные данные с помощью функции make\_blob из Scikit-learn. Данные расположены в четырёх измерениях и вокруг пяти кластерных центров. Суть проблемы в том, что данные сгенерированы вокруг пяти кластерных центров. Однако алгоритм k-средних об этом не знает.  
  
Кластеры можно отобразить на графике следующим образом (попарные признаки):  
  
  
  
Затем прогоним алгоритм k-средних со значениями от *k*=2 до *k*=12, а затем вычислим метрику по умолчанию к k-средних и среднее значение силуэта для каждого прогона, с выводом результатов в двух соседних графиках.  
  
  
  
Разница очевидна. Среднее значение силуэта возрастает до *k*=5, а затем резко снижается для более высоких значений *k*. То есть мы получаем выраженный пик при *k*=5, это количество кластеров, сгенерированных в исходном датасете.

График силуэта имеет пиковый характер, в отличие от мягко изогнутого графика при использовании метода локтя. Его проще визуализировать и обосновать.

Если увеличить гауссов шум при генерировании данных, то кластеры будут сильнее накладываться друг на друга.  
  
  
  
В этом случае вычисление k-средних по умолчанию с применением метода локтя даёт ещё более неопределённый результат. Ниже показан график метода локтя, на котором трудно выбрать подходящую точку, в которой линия на самом деле изгибается. Это 4, 5, 6 или 7?  
  
  
  
При этом график силуэта всё ещё демонстрирует пик в районе 4 или 5 кластерных центров, что существенно облегчает нам жизнь.  
  
Если вы посмотрите на накладывающиеся друг на друга кластеры, то увидите, что, несмотря на то, что мы сгенерировали данные вокруг 5 центров, из-за высокой дисперсии структурно можно выделить только 4 кластера. Силуэт легко выявляет это поведение и показывает оптимальное количество кластеров между 4 и 5.  
  


# **Оценка BIC с моделью смеси нормальных распределений**

Есть и другие замечательные метрики для определения истинного количества кластеров, например, байесовский информационный критерий (BIC). Но их можно применять лишь в том случае, если нам нужно перейти от метода k-средних к более обобщённой версии — смеси нормальных распределений (Gaussian Mixture Model (GMM)).  
  
GMM рассматривает облако данных как суперпозицию многочисленных датасетов с нормальным распределением, с отдельными средними значениями и дисперсиями. А затем GMM применяет алгоритм максимизации алгоритмов, чтобы определить эти средние и дисперсии.  
  


## BIC для регуляризации

Вы уже могли сталкиваться с BIC в статистическом анализе или при использовании линейной регрессии. BIC и AIC (Akaike Information Criterion, информационный критерий Акаике) используются в линейной регрессии в качестве методик регуляризации для процесса отбора переменных.  
  
Аналогичная идея применяется и в случае с BIC. Теоретически, крайне сложные кластеры можно смоделировать как суперпозиции большого количества датасетов с нормальным распределением. Для решения этой задачи можно применять неограниченное количество таких распределений.  
Но это аналогично увеличению сложности модели в линейной регрессии, когда для соответствия данным любой сложности может использоваться большое количество свойств, лишь для того, чтобы потерять возможность обобщения, поскольку излишне сложная модель соответствует шуму, а не настоящему паттерну.  
  
Метод BIC штрафует многочисленные нормальные распределения и пытается сохранить модель достаточно простой, чтобы она описывала заданный паттерн.  
  
Следовательно, можно прогнать алгоритм GMM для большого количества кластерных центров, и значение BIC вырастет до какой-то точки, а затем начнёт снижаться по мере роста штрафа.

