Задача кластеризации

Кластеризация относится к алгоритмам обучения без учителя. Исходные данные не имеют меток.

Кластер – это подмножество объектов в выборке, которые похожи друг на друга в пространстве. Метрику сходства необходимо выбирать под конкретную задачу, но чаще всего используется евклидово расстояние. Когда мы работаем с новыми данными, которые не размечены, кластеризация может быть хорошим способом получить некоторое представление о данных. Алгоритмы кластеризации используются для различных задач, например, обнаружение аномалий, выбросов в данных, сжатие данных, обнаружение структуры данных, восстановление недостающих данных и тд. Некоторые реальные приложения кластеризации обнаруживают мошенничества в страховании, сегментируют клиентов, проводят анализ землетрясений или городского планирования. Алгоритмы кластеризации используются в биологии, например, для описания и проведения пространственных и временных сравнений комплексов организмов.

Существуют множество алгоритмов кластеризации, но самые популярные из них – это k-means (k-средних), иерархический кластерный анализ и алгоритм кластеризации, основанной на плотности (DBSCAN). Рассмотрим данные алгоритмы подробнее.

K-means – является наиболее часто используемым алгоритмом кластеризации. Именно с него начинается знакомство с данным разделом машинного обучения.

K-means работает на основе центроидов. Центроид – это центр кластера.

Алгоритм k-means:

1) Берутся случайные точки в пространстве, которые принимаются за центры кластеров (центроиды C_i).
2) Для каждого объекта в выборке находится ближайший к нему центроид.
3) Каждому центроиду соответствует множество ближайших объектов. Для каждого образованного кластера находится его центр по формуле:

$$c_i = \frac{1}{N_i} \sum_{j=1}^{N_i} x_j,$$

где N_i – количество объектов в кластере.

4) Центроиды переходят в найденный центр кластера. Далее алгоритм повторяется, пока центроиды не перестанут менять положение.

Функция стоимости алгоритма:

$$L(C_1, C_{2,\dots}, C_k) = \sum_{i=1}^{k} \sum_{j=1}^{N_i} \|x_j - C_i\| \to min,$$

где C_i – центры кластеров;

k – количество кластеров;

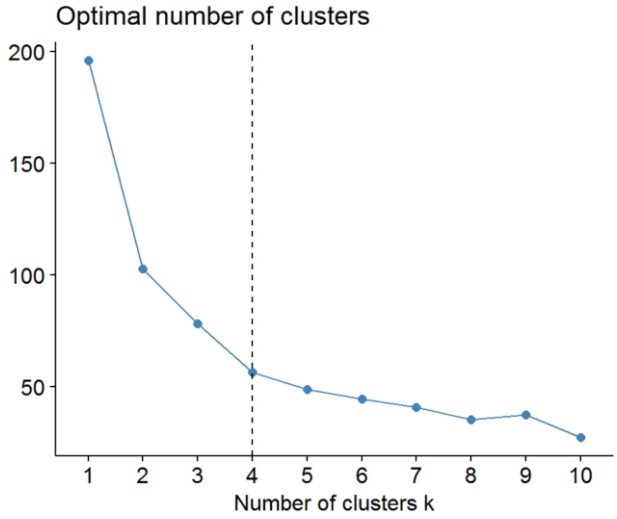
N_i – количество объектов в кластере;

x_j – объекты в кластере.

Целью алгоритма является минимизация дисперсии внутри кластера.

Одни и те же объекты можно по-разному разбить на кластеры, все зависит от начального положения центроидов. Чаще всего выбирается k случайных объектов из выборки, которые становятся центроидами. Можно запустить алгоритм несколько раз и посмотреть, какой результат будет лучше.

Выбор количества кластеров k обусловлен решаемой задачей. Если неизвестно, какое количество кластеров использовать, то используется так называемое «правило сломанной трости» или «правило локтя». Строится график зависимости функции стоимости от числа кластеров. С ростом числа кластеров значение функции стоимости уменьшается. Количество кластеров, где происходит «перелом» графика, берут в качестве k.



Но не всегда такой метод позволяет понять, какое же количество кластеров взять. Почему на предыдущем графике выбор пал на k=4? Почему не 5?

Существует еще способ определения количества кластеров – коэффициент силуэта. Он наиболее понятен и аргументирован, так как

коэффициент силуэта демонстрирует пиковое значение, нежели плавный изгиб в методе локтя. Коэффициент силуэта рассчитывается с использованием среднего внутрикластерного расстояния a и среднего расстояния до ближайшего кластера b для каждой выборки. Коэффициент силуэта для выборки равен

$$\frac{b-a}{max(a,b)}$$

Минусы алгоритма:

- Необходимо выбирать количество кластеров;
- Если объектов очень много, то требуется очень много расчетов, что занимает много времени. Именно поэтому лучше использовать k-means на небольших наборах.

Рассмотрим пример кластеризации для данных о клиентах магазина.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import plotly.graph_objects as go
```

```
data
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

Датафрейм содержит id клиента, пол, возраст, годовой доход и оценку расходов.

Предварительно удалим столбец id клиента и приведем столбец Gender к числовым значениям.

```
data.Gender.replace({'Male':1,'Female':0}, inplace=True)
data.drop('CustomerID',axis = 1,inplace=True)
```

Используя библиотеку sklearn проведем кластеризацию данных. Обучим 10 моделей с разным количеством кластеров и построим график зависимости

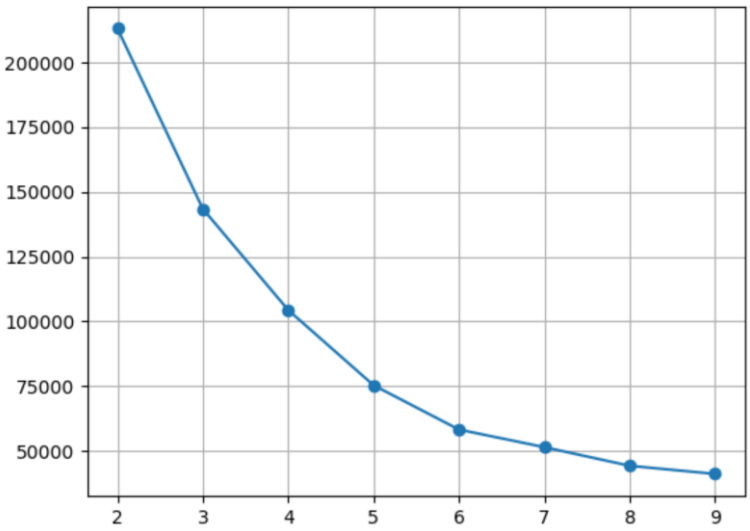
функции стоимости от числа кластеров, и коэффициента силуэта от числа кластеров.

```
from sklearn.metrics import silhouette_score
```

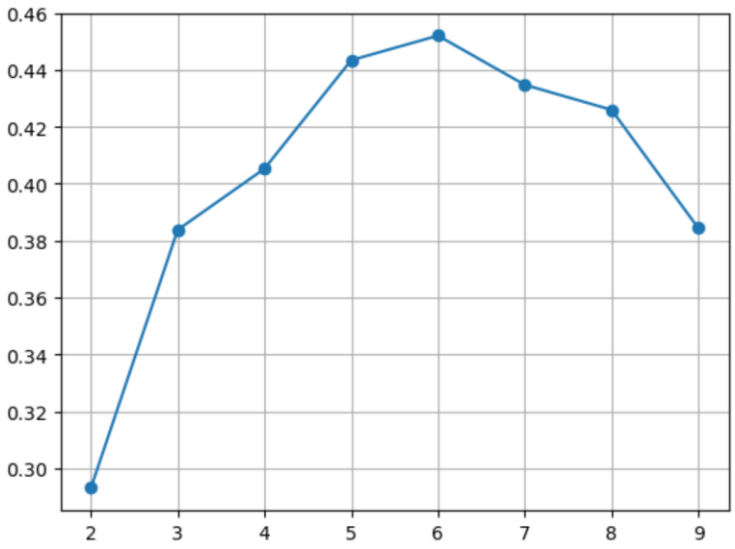
```
models = []
score1 = []
score2 = []
for i in range(2,10):
    model = KMeans(n_clusters=i,random_state=123, init='k-means++').fit(data)
    models.append(model)
    score1.append(model.inertia_)
    score2.append(silhouette_score(data,model.labels_))
```

score1 – это значения функции стоимости, score2 – это значения коэффициента силуэта. model.labels_ – это номера кластеров, которые присваиваются объектам в data.

```
plt.grid()
plt.plot(np.arange(2,10), score1, marker = 'o')
plt.show()
```




```
plt.grid()
plt.plot(np.arange(2,10), score2, marker = 'o')
plt.show()
```



Коэффициента силуэта достигает максимума при k = 6. Результат кластеризации данных для 6 кластеров.

```
model1 = KMeans(n_clusters=6,random_state=123, init='k-means++')
```

```
model1.fit(data)
```

```
model1.cluster_centers_
```

```
array([[ 0.44444444, 56.15555556, 53.37777778, 49.08888889],
       [ 0.46153846, 32.69230769, 86.53846154, 82.12820513],
       [ 0.34210526, 27.        , 56.65789474, 49.13157895],
       [ 0.57142857, 41.68571429, 88.22857143, 17.28571429],
       [ 0.40909091, 25.27272727, 25.72727273, 79.36363636],
       [ 0.38095238, 44.14285714, 25.14285714, 19.52380952]])
```

```
labels = model1.labels_
```

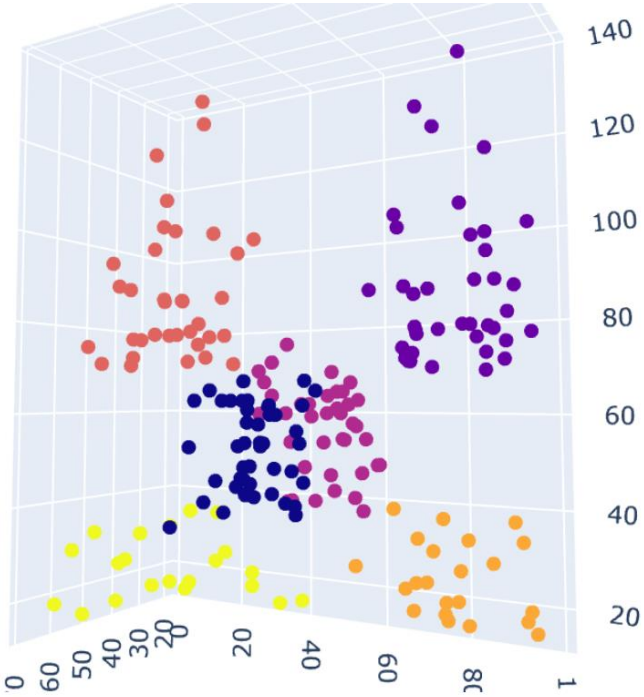
```
data['Claster'] = labels
```

model1.cluster_centers_ – это координаты центров кластеров в пространстве. Количество объектов в каждом кластере:

```
data['Claster'].value_counts()
```

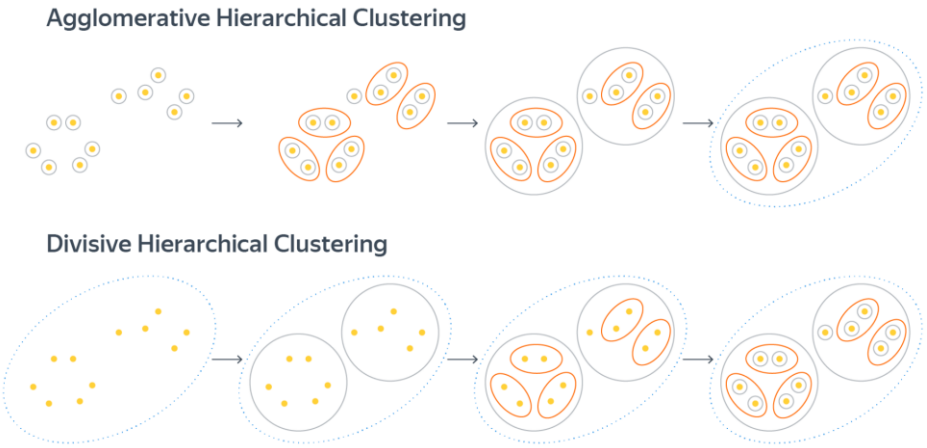
```
0    45
1    39
2    38
3    35
4    22
5    21
Name: Claster, dtype: int64
```

```
fig = go.Figure(data=[go.Scatter3d(x=data['Age'],y=data['Spending Score (1-100)'],z=data['Annual Income (k$)'],
                                   mode='markers', marker_color=data['Claster'], marker_size = 4)])
fig.show()
```



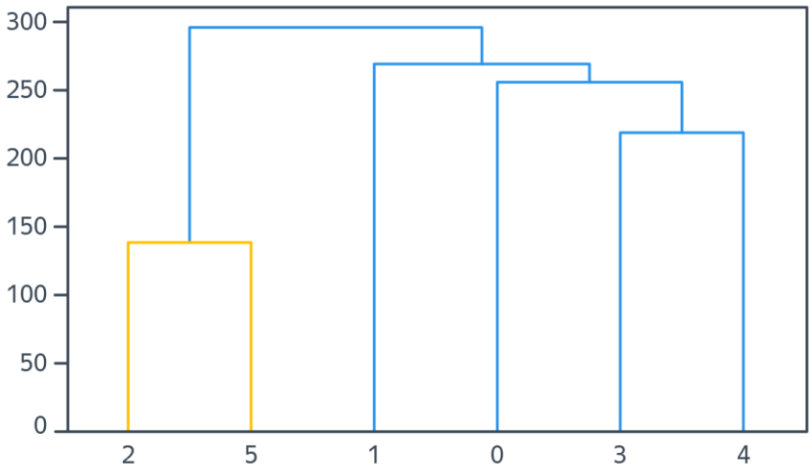
Следующий алгоритм кластеризации – это **иерархическая агломеративная кластеризация** (кластеры вложены друг в друга и образуют древовидную структуру).

Иерархическая кластеризация используются для того, чтобы определить взаимосвязи между объектами. Агломеративная означает, что алгоритм начинает работу с одного объекта и постепенно объединяет их в кластеры побольше. Второй вариант – это дивизионные алгоритмы. Такой алгоритм начинает с больших кластеров и постепенно их делит на кластеры поменьше.



Для начала вычисляется матрица расстояний между каждой парой объектов. Далее выбирается пара объектов с минимальным расстоянием и вычисляется среднее значение, после чего снова рассчитывается новая матрица расстояний, с учетом того, что пара объектов становится одним новым объектом. Таким образом алгоритм повторяется, пока не останется только одно значение.

В результате кластеризации строится дендрограмма, которая позволяется понять структуру исходного набора данных.



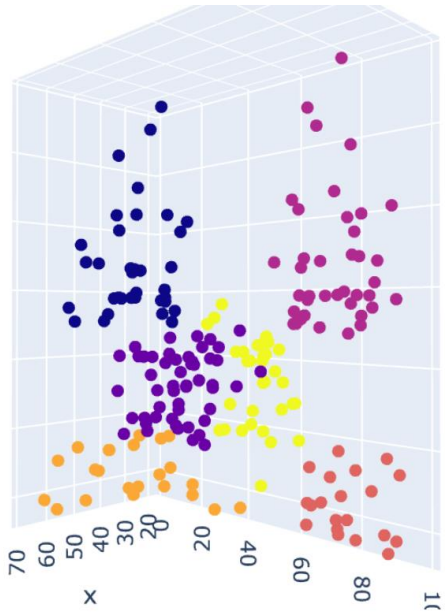
Ось абсцисс – номер объекта. Ось ординат – расстояние между объектами в момент слияния.

Пример:

```
from sklearn.cluster import AgglomerativeClustering
```

```
model2 = AgglomerativeClustering(6,compute_distances=True)
clastering = model2.fit(data)
data['Claster']=clastering.labels_
```

```
fig = go.Figure(data=[go.Scatter3d(x=data['Age'],y=data['Spending Score (1-100)'],z=data['Annual Income (k$)'],
                                   mode='markers', marker_color=data['Claster'], marker_size = 4)])
fig.show()
```



DBSCAN (Density-based spatial clustering of applications with noise) основан на плотности. Алгоритм группирует вместе точки, которые тесно расположены, а выбросами помечает точки, которые находятся в областях с малой плотностью. Плотность в DBSCAN определяется в окрестности каждого объекта выборки x_i как количество других точек выборки в шаре B(ε, x_i). В качестве гиперпараметра кроме радиуса ε (окрестности шара) задается порог N по количеству точек в окрестности.

Все объекты в пространстве делятся на 3 типа: основные точки, граничные точки и шумовые точки. Основные точки – это точки, в окрестности которых больше N точек. Граничные точки – это точки, в окрестности которых есть основные точки, но всего точек меньше N. И шумовые точки – это точки, в окрестности которых нет основных точек и всего точек меньше N.

Шумовые точки убираются из рассмотрения и не приписываются ни к какому кластеру.

Основные точки, у которых есть общая окрестность, соединяются ребром.

В полученном графе выделяются компоненты связности.

Каждая граничная точка относится к тому кластеру, в который попала ближайшая к ней основная точка.

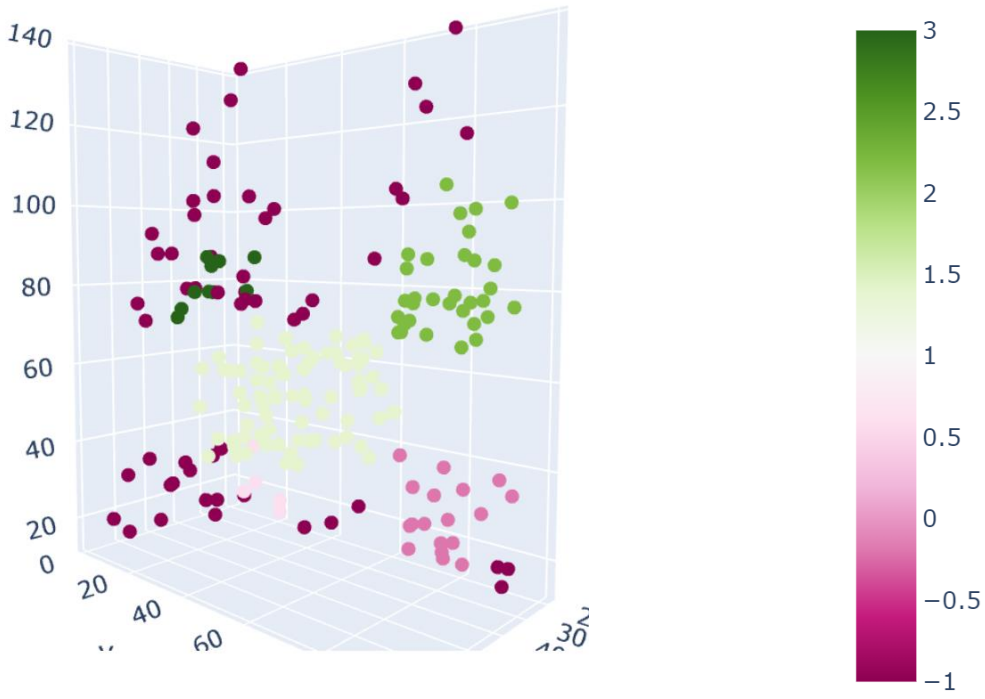
Большим плюсом DBSCAN является то, что он сам выбирает количество кластеров, а также очень хорошо справляется со сложными формами кластеров. Это один из самых эффективных алгоритмов кластеризации, но работает он достаточно долго. К минусам алгоритма относят: неспособность соединять кластеры через проёмы, и, наоборот, способность связывать явно различные кластеры через плотно населённые перемычки.
Пример:

```
from sklearn.cluster import DBSCAN
```

```
model3 = DBSCAN(eps=11, min_samples=5).fit(data)
```

```
data['Claster'] = model3.labels_
```

```
fig = go.Figure(data=[go.Scatter3d(x=data['Age'],y=data['Spending Score (1-100)'],z=data['Annual Income (k$)'],
                                   mode='markers', marker_color=data['Claster'], marker_size = 4)])
fig.show()
```



Номер кластера «-1» – это объекты, которые алгоритм выделил как шумовые.

Практическое задание

1. Найти данные для кластеризации. Данные в группе не должны повторяться! Внимание, если признаки в данных имеют очень сильно разные масштабы, то необходимо данные предварительно нормализовать.
2. Провести кластеризацию данных с помощью алгоритма k-means. Использовать «правило локтя» и коэффициент силуэта для поиска оптимального количества кластеров.
3. Провести кластеризацию данных с помощью алгоритма иерархической кластеризации.
4. Провести кластеризацию данных с помощью алгоритма DBSCAN.
5. Сравнить скорость работы алгоритмов. Результаты изобразить в виде таблицы.
6. Визуализировать кластеризованные данные с помощью t-SNE или UMAP если данные многомерные. Если данные трехмерные, то можно использовать трехмерный точечный график.
7. Оформить отчет о проделанной работе. Сделать выводы.