Лекция 4: кластеризация

Что есть кластер?

- Кластер: группа «похожих» объектов
  - «похожих» между собой в группе (внутриклассовое расстояние)
  - «не похожих» на объекты других групп
  - Определение неформальное, формализация зависит от метода
- Кластерный анализ
  - Разбиение множество объектов на группы (кластеры)
- Тип моделей:
  - «описательный» (descriptive) Data mining => одна из задач - наглядное представление кластеров
  - «прогнозный» (predictive) Data mining => разбиение на кластеры, а затем «классификация» новых объектов
- Тип обучения:
  - всегда «без учителя» (unsupervised) => тренировочный набор не размечен
- Этапы кластерного анализа:
  - Подготовка данных
  - Применение алгоритма
  - Визуализация и интерпретация результатов

Требования к методу кластеризации

- Масштабируемость
- Поддержка различных типов атрибутов и структур данных
- Возможность находить кластеры сложной формы
- Отсутствие обязательных требований к наличию априорных знаний о выборке (например, о распределениях)
- Устойчивость к «шуму» и выбросам
- Возможность работы с высокой размерностью и с большой выборкой
- Возможность включать пользовательские ограничения и зависимости
- Интерпретируемость и наглядность (прототипы, границы, правила, функции принадлежности и т.п.)
- Интуитивность параметров кластеризации

Качество кластеризации

- Хороший метод кластеризации находит кластеры
  - с высоким «внутриклассовым» сходством объектов
  - и низким «межклассовым» сходством объектов
- Оценка качества кластеризации (нет понятия «точность»)
  - необходима, так как влияет на выбор параметров метода
  - определяется либо экспертом – субъективная величина
  - либо «перекрестной» проверкой целевой функции кластеризации
- Качество кластеризации зависит:
  - от метода кластеризации
  - от меры сходства (или расстояния)

Подготовка данных для кластеризации

- Отбор наблюдений
  - Что я разбиваю на кластеры?
  - Решаемые задачи: исключить выбросы (Filter), уменьшить выборку (Sample)
- Отбор и трансформация переменных
  - Какие характеристики объектов важны? Выбирает эксперт.
  - Переменные коррелируют? (PCA и Variable Clustering)
  - Распределения переменных симметричны? (Transform)
- Стандартизация переменных
  - Сравнимы ли масштабы переменных?
  - Делается автоматически узлом кластеризации

Фильтрация данных

- Цель – удаление из выборки артефактов и выбросов
- Правила фильтрации задаются для отдельных переменных:
  - Ручные – задаются допустимые значения переменных (диапазоны для числовых, список для категориальных)

```
df[(df["ATMAMT"] > 10) & (df["ATMAMT"] < 1000)]
```

	ACCTAGE	AGE	ATM	ATMAMT	BRANCH	CASHBK	CC
0	0.3	34.0	1.0	166.17	B4	0.0	0.0
2	7.5	NaN	1.0	518.22	B17	0.0	0.0
6	10.8	63.0	1.0	138.78	B12	0.0	1.0
27	8.1	34.0	1.0	464.94	B17	0.0	0.0

```
df[df["RES"].isin(["U", "R"])]
```

	ACCTAGE	AGE	ATM	ATMAMT	BRANCH	CASHBK	CC
0	0.3	34.0	1.0	166.17	B4	0.0	0.0
1	4.7	43.0	0.0	0.00	B16	0.0	1.0
5	3.9	NaN	1.0	1672.45	B2	0.0	1.0
8	21.8	41.0	0.0	0.00	B17	0.0	1.0

```
df.query("(AGE * CC) < 30")
```

	ACCTAGE	AGE	ATM	ATMAMT	BRANCH	CASHBK	CC
0	0.3	34.0	1.0	166.17	B4	0.0	0.0
3	2.5	70.0	1.0	1459.89	B2	1.0	0.0
10	25.3	27.0	0.0	0.00	B3	0.0	1.0
12	4.7	29.0	1.0	1073.09	B17	0.0	0.0

Фильтрация данных

- Цель – удаление из выборки артефактов и выбросов
- Правила фильтрации задаются для отдельных переменных:
  - Редкие значения для категориальных
  - Нетипичные значения для числовых (задается допустимое отклонение от мат. ожидания или допустимое отклонение от медианы или экстремальные процентили и другое).

```
m = df["ATMAMT"].mean()
sigma = df["ATMAMT"].std()

rule = (m - 3*sigma < df["ATMAMT"]) &
       (df["ATMAMT"] < m + 3*sigma)

print(m - 3*sigma, m + 3*sigma)
df[~rule].head()  # аномальные
```

```
-10628.768563579122 13073.772466385677
```

	ACCTAGE	AGE	ATM	ATMAMT	BRANCH	CASHBK	CC
174	4.3	NaN	1.0	13819.93	B5	0.0	1.0
179	6.3	38.0	1.0	13795.32	B1	0.0	1.0
310	28.2	NaN	1.0	49962.41	B11	0.0	1.0
336	4.3	41.0	1.0	54831.16	B6	0.0	1.0
342	3.6	20.0	1.0	34505.55	B14	0.0	NaN

```
q1 = df["ATMAMT"].quantile(0.025)
q2 = df["ATMAMT"].quantile(0.975)

rule = (q1 <= df["ATMAMT"]) &
       (df["ATMAMT"] <= q2)
print(q1, q2)
df[~rule].head()  # аномальные
```

```
0.0 7608.918000000005
```

	ACCTAGE	AGE	ATM	ATMAMT	BRANCH	CASHBK	CC
11	4.3	64.0	1.0	9986.04	B5	0.0	1.0
64	5.7	51.0	1.0	7882.20	B4	0.0	1.0
79	1.9	67.0	1.0	7720.66	B16	0.0	1.0
81	8.7	41.0	1.0	11577.95	B13	0.0	0.0
129	4.0	63.0	1.0	7950.58	B1	0.0	1.0

Сокращение обучающей выборки – случайная выборка (Sampling)

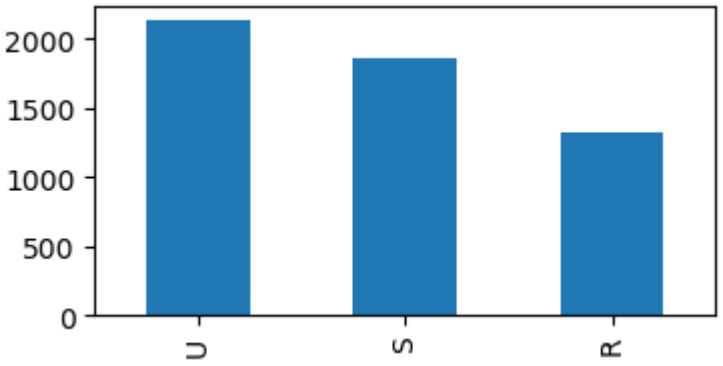
- Цель – выбрать «представительное» подмножество примеров:
  - В идеале с тем же распределением
  - Просто случайная выборка работает плохо – не удается сохранить характеристики всего набора
- Адаптивные методы случайной выборки:
  - В соответствии с «грубой» моделью, например кластерной
  - Случайная выборка в рамках экспертных «срезов» (условия на срезы формируются аналитиком)
  - Случайная выборка в рамках «срезов», построенных автоматически по какому-либо классу, высоко селективному атрибуту или их комбинации
  - Основная особенность – выборка в рамках среза или кластера пропорциональна размеру среза или кластера

Сокращение обучающей выборки – случайная выборка (Sampling)

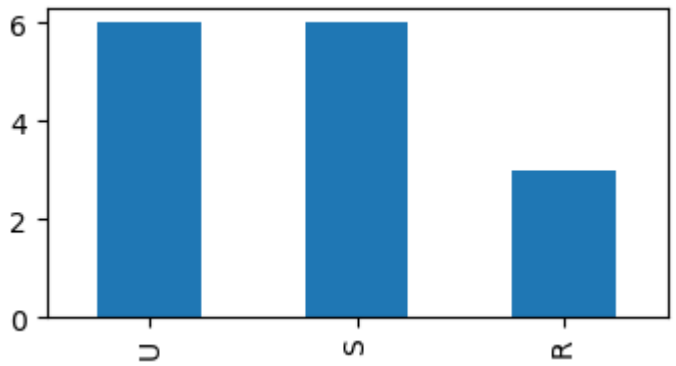
```
# Sample без стратификации
spl = df.sample(n=15,  # или frac
                #weights
                replace=True, random_state=1)
```

	ACCTAGE	AGE	ATM	ATMAMT	BRANCH	CASHBK	CC	CCBAL	CCPURC	CD	...	PHONE	POS	POSAMT	RES
5157	2.9	38.0	0.0	0.00	B1	0.0	0.0	0.00	0.0	0.0	...	1.0	0.0	0.00	U
235	4.3	NaN	0.0	0.00	B2	0.0	1.0	305.65	0.0	0.0	...	1.0	0.0	0.00	U
3980	1.0	44.0	1.0	184.69	B5	0.0	1.0	249.82	1.0	0.0	...	0.0	0.0	0.00	R
5192	4.3	45.0	1.0	2471.67	B7	0.0	1.0	0.00	0.0	0.0	...	0.0	8.0	145.41	S
905	0.4	53.0	1.0	1967.91	B4	0.0	0.0	0.00	0.0	0.0	...	0.0	0.0	0.00	U

```
df["RES"].value_counts().plot.bar()
```

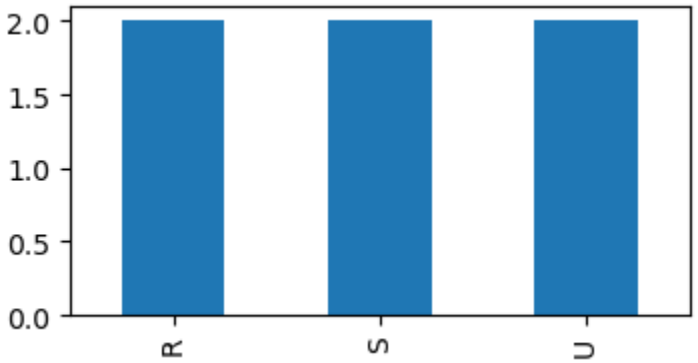



```
spl.value_counts().plot.bar()
```



Сокращение обучающей выборки – случайная выборка (Sampling)

```
# Фиксировано n для каждой страты
# исходное распределение => равномерное
spl = df.groupby('RES', group_keys=False).sample(n=2)

spl["RES"].value_counts().plot.bar(figsize = (4, 2))
```




```
# Проверим распределение RES
df["RES"].value_counts()
```

```
U    2131
S    1856
R    1322
Name: RES, dtype: int64
```

Сокращение обучающей выборки (SAMPLING) – метод стратификации

- Задается процент исходной выборки
- Для выбранной категориальной переменной (переменная стратификации) строится частотная диаграмма (для числовой необходима предварительная дискретизация)
- Наблюдения случайным образом выбрасываются так, чтобы сохранить распределение переменной стратификации

```
# Формируем словарь частотностей страт
freq = (df["RES"].value_counts()/df.shape[0])
freq = freq.round(4).to_dict()
```

```
{'U': 0.4014, 'S': 0.3496, 'R': 0.249}
```

```
# Задаем размер
size = 1000
# Для каждой страты семплим int(размер * доля страты)
sample_by_strata = lambda x: x.sample(n = int(size * freq[x["RES"].iloc[0]]))
smpl = df.groupby('RES', group_keys=False).apply(sample_by_strata)
```

```
freq_s = (smpl["RES"].value_counts()/smpl.shape[0])
freq_s = freq_s.round(4).to_dict()
freq_s
```

```
{'U': 0.4014, 'S': 0.3493, 'R': 0.2492}
```

Сокращение обучающей выборки (SAMPLING) – метод стратификации

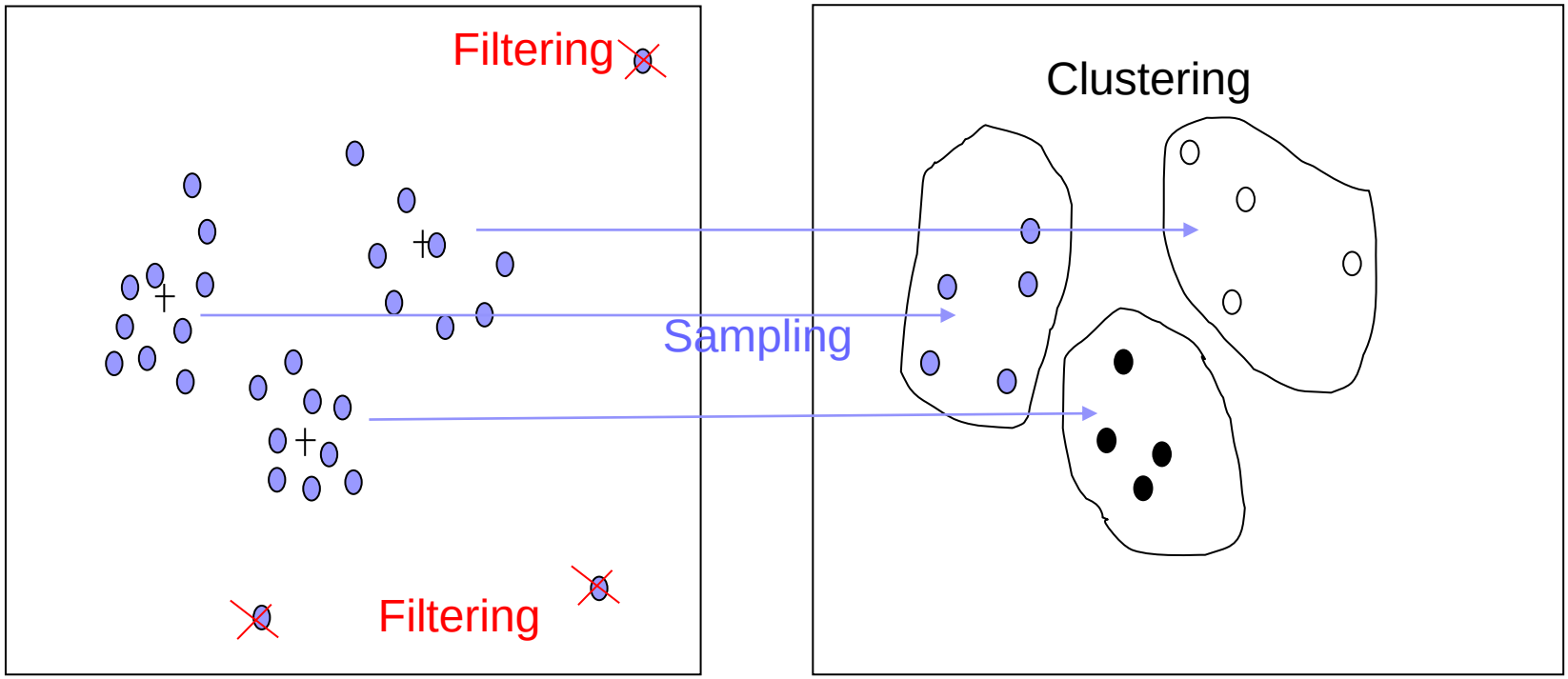
```
# Альтернативный вариант
# Для train_test_split для нужно минимум 2 наблюдения по страте
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(df, train_size=8, random_state=1,
                                     shuffle=True, stratify=df["RES"])
df_train
```

	ACCTAGE	AGE	ATM	ATMAMT	BRANCH	CASHBK	CC	CCBAL	CCPURC	CD	...	PHONE	POS	POSAMT	RES
1103	4.2	61.0	0.0	0.00	B16	0.0	1.0	125744.74	2.0	0.0	...	0.0	0.0	0.00	R
917	0.3	37.0	1.0	330.97	B14	0.0	NaN	NaN	NaN	0.0	...	NaN	NaN	NaN	U
1875	2.4	30.0	0.0	0.00	B5	0.0	0.0	0.00	0.0	0.0	...	0.0	0.0	0.00	U
4782	0.8	34.0	0.0	0.00	B4	0.0	1.0	55642.34	0.0	0.0	...	1.0	0.0	0.00	S
733	3.6	38.0	1.0	1738.95	B16	0.0	0.0	0.00	0.0	0.0	...	2.0	2.0	48.79	R
2502	1.4	44.0	0.0	0.00	B1	0.0	1.0	8.50	0.0	0.0	...	0.0	0.0	0.00	U
1420	2.3	41.0	1.0	1142.95	B12	0.0	0.0	0.00	0.0	0.0	...	0.0	1.0	73.16	S
1943	1.8	60.0	1.0	772.38	B1	1.0	0.0	0.00	0.0	0.0	...	0.0	0.0	0.00	S

Подготовка данных для кластеризации

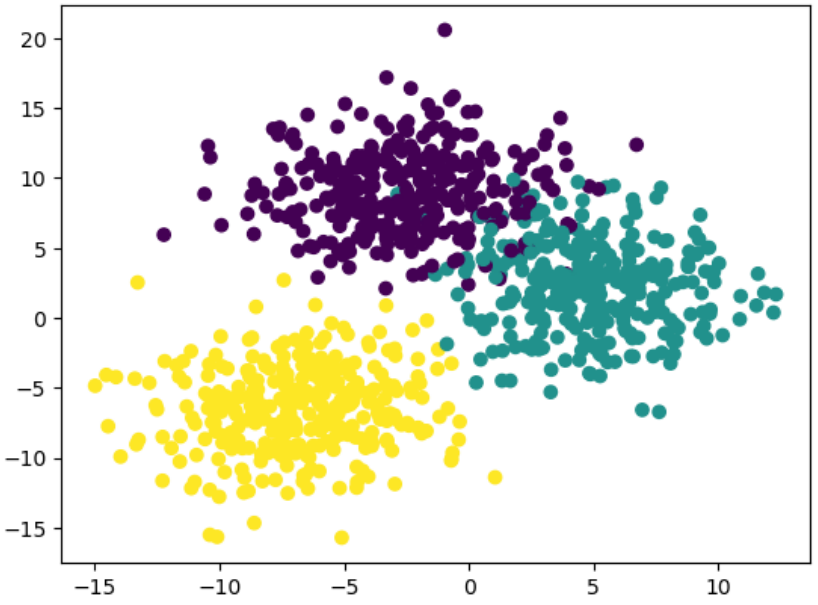
«Сырые» данные

Кластерная/стратифицированная случайная выборка

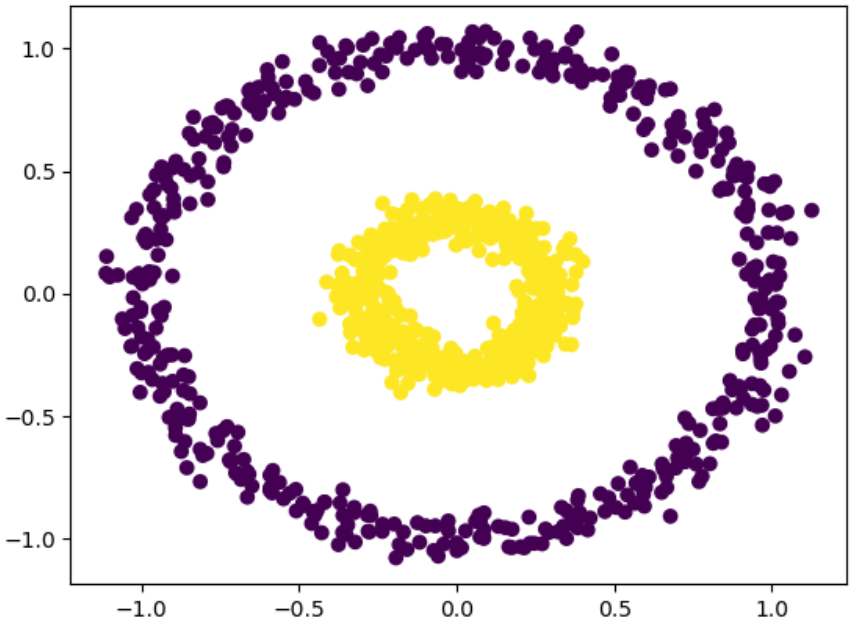


Пример данных

```python
# Генерация данных 3х облаков точек
from sklearn.datasets import make_blobs

n_samples = 1000
X_blob, y_blob = make_blobs(n_samples=n_samples,
                            centers=3, cluster_std=3,
                            random_state=42)
plt.scatter(X_blob[:, 0], X_blob[:, 1], c=y_blob)
```




```python
# Генерация данных концентрических кругов
from sklearn.datasets import make_circles

n_samples = 1000
X_cl, y_cl = make_circles(n_samples=n_samples, factor=0.3,
                          noise=0.05, random_state=0)
plt.scatter(X_cl[:, 0], X_cl[:, 1], c=y_cl)
```



Исходные данные

- Матрица признаков:
  - Числовые
  - Бинарные
  - Номинальные (категориальные)
  - Упорядоченные шкалы
  - Нелинейные шкалы

$$\begin{bmatrix} x_{11} & \dots & x_{1f} & \dots & x_{1p} \\ \dots & \dots & \dots & \dots & \dots \\ x_{i1} & \dots & x_{if} & \dots & x_{ip} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & \dots & x_{nf} & \dots & x_{np} \end{bmatrix}$$

- Матрица различия (или сходства):
  - «Естественные» расстояния предметной области
  - Экспертные оценки (противоречивы, нетранзитивны, недостоверны)

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \dots & \dots & 0 \end{bmatrix}$$

Числовые значения – приведение к близким шкалам

- Нормализация на абсолютное отклонение более робастно (устойчиво к ошибкам), чем нормализация на стандартное отклонение:
  - Среднее абсолютное отклонение

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + ... + |x_{nf} - m_f|)$$

где

$$m_f = \frac{1}{n}\left(x_{1f} + x_{2f} + ... + x_{nf}\right)$$

  - *z-score* $z_{if} = \frac{x_{if} - m_f}{s_f}$

- Обычная нормализация: $y_{if} = \frac{x_{if} - m_f}{std_f}$

$$std_f = \sqrt{\frac{1}{n-1}\left[(x_{1f} - m_f)^2 + (x_{2f} - m_f)^2 + ... + (x_{nf} - m_f)^2\right]}$$

Меры сходства и различия для исходных данных с числовыми атрибутами

- Обычно строится на основе расстояния:
  - $d(i,j) \geq 0$, $d(i,i) = 0$, $d(i,j) = d(j,i)$, $d(i,j) \leq d(i,k) + d(k,j)$
- Наиболее популярно расстояние Минковского:

$$d(i,j) = \sqrt[q]{(|x_{i_1} - x_{j_1}|^q + |x_{i_2} - x_{j_2}|^q + \ldots + |x_{i_p} - x_{j_p}|^q)}$$

где $i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ и $j = (x_{j1}, x_{j2}, \ldots, x_{jp})$ - два объекта с p числовыми атрибутами, q - положительное целое число

- $q = 2$ - Евклидово (не фамилия, но имя) расстояние:

$$d(i,j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \ldots + |x_{i_p} - x_{j_p}|^2)}$$

- $q = 1$, d – расстояние «Манхэтен»:

$$d(i,j) = |x_{i_1} - x_{j_1}| + |x_{i_2} - x_{j_2}| + \ldots + |x_{i_p} - x_{j_p}|$$

Бинарные атрибуты

- Расстояние Хэмминга = сумма единиц после XOR $(M_{10} + M_{01})$
- Таблица «сопряженных признаков»
  - В ячейках – число совпадающих и несовпадающих значений из p бинарных атрибутов для объектов *j* и *i*

		Object *j*		
		1	0	*sum*
Object *i*	1	M_{11}	M_{10}	$M_{11}+M_{10}$
	0	M_{01}	M_{00}	$M_{01}+M_{00}$
	sum	$M_{11}+M_{01}$	$M_{10}+M_{10}$	$M_{00}+M_{01}+M_{10}+M_{11}$

- На основе коэффициента совпадения $d(i, j) = \frac{M_{10} + M_{01}}{M_{11} + M_{01} + M_{10} + M_{00}}$
  - для симметричных атрибутов (значения равнозначны)
- На основе коэффициента Jaccard $d(i, j) = \frac{M_{10} + M_{01}}{M_{11} + M_{01} + M_{10}}$
  - для асимметричных атрибутов (единица важнее)

Пример

Имя	Пол	Жар	Кашель	Test-1	Test-2	Test-3	Test-4
Jack	M	Y	N	P	N	N	N
Mary	F	Y	N	P	N	P	N
Jim	M	Y	P	N	N	N	N

- пол - симметричный атрибут
- остальные ассиметричные
- пусть Y и P соответствует 1, а N соответствует 0

$$d(jack, mary) = \frac{0+1}{2+0+1} = 0.33$$

$$d(jack, jim) = \frac{1+1}{1+1+1} = 0.67$$

$$d(jim, mary) = \frac{1+2}{1+1+2} = 0.75$$

Категориальные атрибуты и шкалы

- Категориальные атрибуты:
  - много значений, например, цвета: red, yellow, blue, green
- Подход 1: простое совпадение
  - *M* - число совпадений, *p* - число переменных (аналог нормированного расстояния Хэмминга) $d(i, j) = \frac{p - m}{p}$
- Подход 2: кодирование бинарными векторами
  - Для каждого значения категориального атрибута создается отдельная бинарная переменная: один категориальный атрибут с M возможными значениями => бинарный вектор длины M
- Категориальные упорядоченные шкалы: $r_{if} \in \{1, \ldots, M_f\}$
  - Могут быть и дискретными и непрерывными
  - Порядок важен, «разница» - нет = ранги
  - сводятся к числовым: заменить x_{if} на его ранг, отобразить на [0, 1] с нормировкой: $z_{if} = \frac{r_{if} - 1}{M_f - 1}$
  - затем использовать стандартные расстояния

Смешанные типы атрибутов

- Нелинейные шкалы:
  - логарифмические, экспоненциальные и другие
  - «обратное» преобразование к линейной шкале
  - «экспертное» преобразование к ранговой шкале
- «Взвешенное» расстояние:

$$d(i,j) = \frac{\sum_{f=1}^{p} w_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} w_{ij}^{(f)}}$$

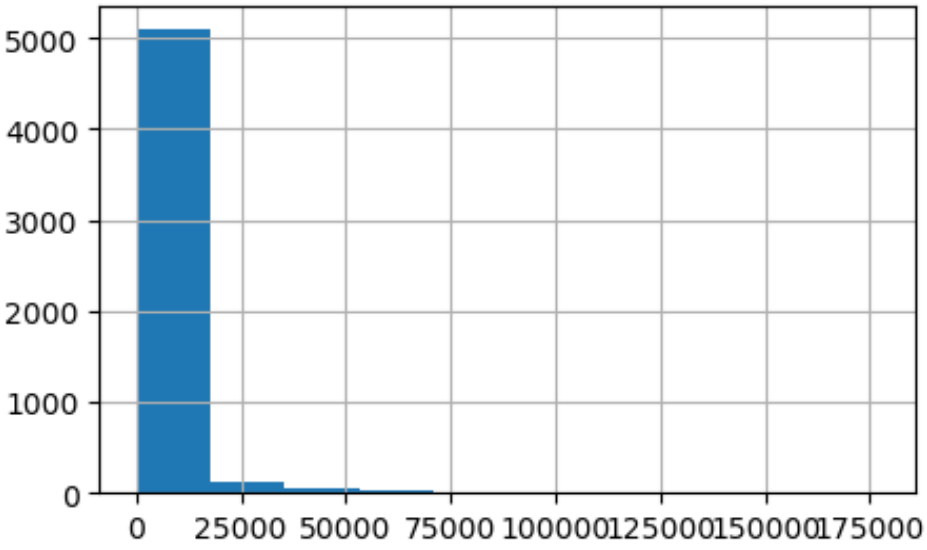
  - *f* - бинарный или номинальный: $d_{ij}^{(f)} = 0$ если $x_{if} = x_{jf}$, иначе $d_{ij}^{(f)} = 1$
  - *f* - числовой: использовать нормализованное расстояние
  - *f* - шкала: рассчитать ранги r_{if} , нормализовать и считать числовым
- Другие меры сходства:
  - Ядерные функции
  - На основе корреляции
  - «Тригонометрические»:

$$s(i,j) = \frac{\langle \vec{x}_i, \vec{x}_j \rangle}{\sqrt{\langle \vec{x}_i, \vec{x}_i \rangle \langle \vec{x}_j, \vec{x}_j \rangle}}$$

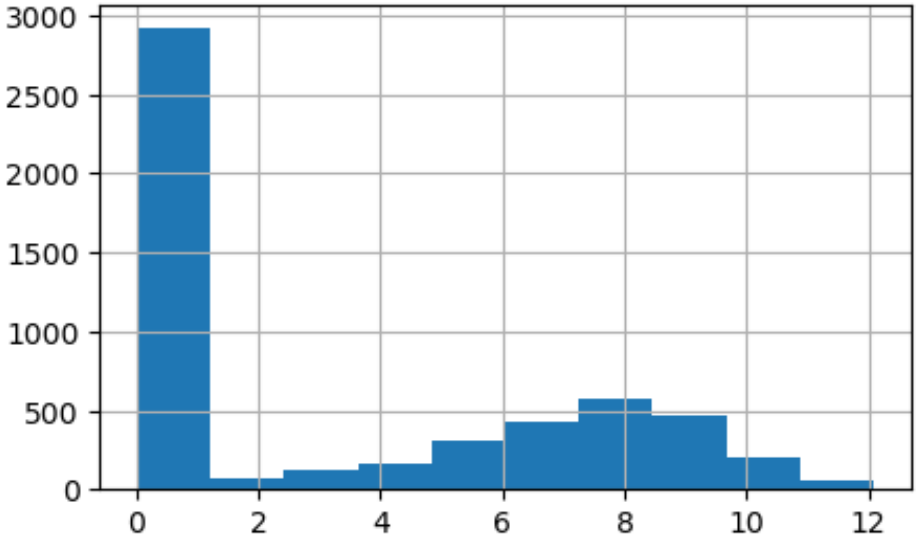
Преобразование переменных

- Простые преобразования:
  - Функции от исходной (log, exp, …), дискретизации (на бакеты и квантили), объединение редких категориальных значений и т.д.
- Адаптивные преобразования – перебор простых и выбор лучшего по некоторому криетрию:
  - Нормальность распределения результата, корреляция с откликом, Оптимальная дискретизация и т.д.

```
df["SAVBAL"].hist(figsize = (5, 3))
```




```
df["log_SAVBAL"] = np.log(df["SAVBAL"] + 1)
df["log_SAVBAL"].hist(figsize = (5, 3))
```



sklearn.preprocessing (предобработка данных)

- Обучение: fit, fit_transform
- Преобразование: transform, inverse_transform
- Стандартизация:
  - StandardScaler: (x - mean(x))/std(x)
  - RobustScaler: (x - median(x))/IQR(x)
- Нормализация:
  - MinMaxScaler: (x - min(x))/(max(x) - min(x))
  - MaxAbsScaler: x/max(|x|)
  - Normalizer(norm) # l1, l2, max
- Преобразование:
  - QuantileTransformer(n_quantiles)
  - PowerTransformer # yeo-johnson, box-cox
  - KBinsDiscretizer: x -> 0..k-1
  - Binarizer: x -> 0,1

```
X = np.array([[1, 10, 100, 200, 1000]]).T

StandardScaler().fit_transform(X)
RobustScaler().fit_transform(X)
MinMaxScaler().fit_transform(X)
MaxAbsScaler().fit_transform(X)
Normalizer().fit_transform(X)
QuantileTransformer().fit_transform(X)
PowerTransformer().fit_transform(X)
KBinsDiscretizer(encode="ordinal").fit_transform(X)
Binarizer(threshold=20).fit_transform(X)
```

```
StandardScaler:
 [[-0.69493808]
 [-0.67099305]
 [-0.43154271]
 [-0.16548679]
 [ 1.96296062]]
RobustScaler:
 [[-0.52105263]
 [-0.47368421]
 [ 0.        ]
 [ 0.52631579]
 [ 4.73684211]]
MinMaxScaler:
 [[0.        ]
 [0.00900901]
 [0.0990991 ]
 [0.1991992 ]
 [1.        ]]
MaxAbsScaler:
 [[0.001]
 [0.01 ]
 [0.1  ]
 [0.2  ]
 [1.   ]]
Normalizer:
 [[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
QuantileTransformer:
 [[0.  ]
 [0.25]
 [0.5 ]
 [0.75]
 [1.  ]]
PowerTransformer:
 [[-1.42849354]
 [-0.76770092]
 [ 0.21971925]
 [ 0.55897206]
 [ 1.41750316]]
KBinsDiscretizer:
 [[0.]
 [1.]
 [2.]
 [3.]
 [4.]]
Binarizer:
 [[0]
 [0]
 [1]
 [1]
 [1]]
```

Предобработка данных

- Кодирование категориальных признаков:
  - Модуль sklearn.preprocessing
  - LabelEncoder()
  - OrdinalEncoder(categories)
  - OneHotEncoder(categories, min_frequency)
  - Параметр handle_unknown (error, ignore, …)
- Обработка пропусков:
  - Модуль sklearn.impute
  - SimpleImputer(missing_values, strategy)
  - KNNImputer(n_neighbors, weights)
  - MissingIndicator(missing_values)

```
X = [['male', 'sin', 'fit'],
     ['female', 'cos', 'predict'],
     ['female', 'sin', 'transform']]

OneHotEncoder().fit_transform(X).toarray()
```

```
[[0., 1., 0., 1., 1., 0., 0.],
 [1., 0., 1., 0., 0., 1., 0.],
 [1., 0., 0., 1., 0., 0., 1.]]
```

```
X = np.array([[1, np.nan], [np.nan, 3], [7, 6]])
imp = SimpleImputer(missing_values=np.nan,
                    strategy='mean')
imp.fit_transform(X)
```

```
[[1. , 4.5],
 [4. , 3. ],
 [7. , 6. ]]
```

```
knn_imp = KNNImputer(n_neighbors=1,
                     weights="uniform")
knn_imp.fit_transform(X)
```

```
[[1., 6.],
 [7., 3.],
 [7., 6.]]
```

Предобработка признакового пространства

- Генерация признаков:
  - Модуль sklearn.preprocessing
  - PolynomialFeatures(degree, interaction_only)
  - SplineTransformer(degree, n_knots)
  - FunctionTransformer(func, inverse_func)

- Уменьшение размерности:
  - Модуль sklearn.decomposition
  - PCA(n_components) # число или mle
  - TruncatedSVD(n_components, algorithm)
  - Атрибуты: **explained_variance_, singular_values_**
  - NMF(n_components, init, solver)
  - LatentDirichletAllocation(n_components)
  - Общий атрибут: **components_**

```
X = np.arange(6).reshape(3, 2)
PolynomialFeatures(2).fit_transform(X)
```

```
[[ 1.,  0.,  1.,  0.,  0.,  1.],
 [ 1.,  2.,  3.,  4.,  6.,  9.],
 [ 1.,  4.,  5., 16., 20., 25.]]
```

```
SplineTransformer(2, 1).fit_transform(X)
```

```
[[1. , 0. , 1. , 0. ],
 [0.5, 0.5, 0.5, 0.5],
 [0. , 1. , 0. , 1. ]]
```

```
X = np.arange(100).reshape(10, 10)

PCA(n_components=2).fit_transform(X)
```

```
[[ 1.42302495e+02,  1.67908651e-14],
 [ 1.10679718e+02, -2.41419792e-15],
 [ 7.90569415e+01, -3.65037002e-15],
 [ 4.74341649e+01, -1.51614199e-15],
 [ 1.58113883e+01, -2.24513984e-16],
 [-1.58113883e+01,  2.24513984e-16],
 [-4.74341649e+01,  1.51614199e-15],
 [-7.90569415e+01,  3.65037002e-15],
 [-1.10679718e+02,  2.41419792e-15],
 [-1.42302495e+02,  7.91882609e-15]]
```

Предобработка признакового пространства

```
X = np.arange(6).reshape(3, 2)
print("X.shape:", X.shape)
print("X:\n", X)

pf = PolynomialFeatures(2)
pol_X = pf.fit_transform(X)
print("pol_X.shape:", pol_X.shape)
print("pol_X:\n", pol_X)

pca = PCA(n_components=2)
decom_X = pca.fit_transform(pol_X)
print("decom_X.shape:", decom_X.shape)
print("decom_X:\n", decom_X)

print("components_:\n",
      np.round(pca.components_, 3))
print("explained_variance_:\n",
      pca.explained_variance_)
print("singular_values_:\n",
      pca.singular_values_)
```

```
X.shape: (3, 2)
X:
 [[0 1]
 [2 3]
 [4 5]]
pol_X.shape: (3, 6)
pol_X:
 [[ 1.  0.  1.  0.  0.  1.]
 [ 1.  2.  3.  4.  6.  9.]
 [ 1.  4.  5. 16. 20. 25.]]
decom_X.shape: (3, 2)
decom_X:
 [[-15.51982135  -0.68445502]
 [ -4.51113148   0.99147675]
 [ 20.03095282  -0.30702173]]
components_:
 [[-0.     0.107  0.107  0.457  0.564  0.671]
 [ 0.     0.488  0.488 -0.612 -0.124  0.364]]
explained_variance_:
 [331.22711642   0.77288358]
singular_values_:
 [25.73818628  1.24328885]
```

Основные типы алгоритмов кластеризации

- Иерархические:
  - Создается иерархическая декомпозиция исходного множества объектов в соответствии с некоторой стратегией «объединения» (восходящая кластеризация) или «разбиения» (нисходящая)
- На основе группировки (partitioning):
  - Направленный перебор вариантов разбиения исходного множества объектов, выбор лучшего по некоторому критерию
  - k-means, k-medoids
- На основе связности:
  - Кластеры ищутся в виде связных областей с помощью локальной оценки числа ближайших соседей
- Модель-ориентированые (статистические):
  - Выбирается некоторая гипотеза (параметрическая модель) о структуре кластеров и находятся, параметры, наилучшим образом приближающие эту модель

Natural Grouping Criterion

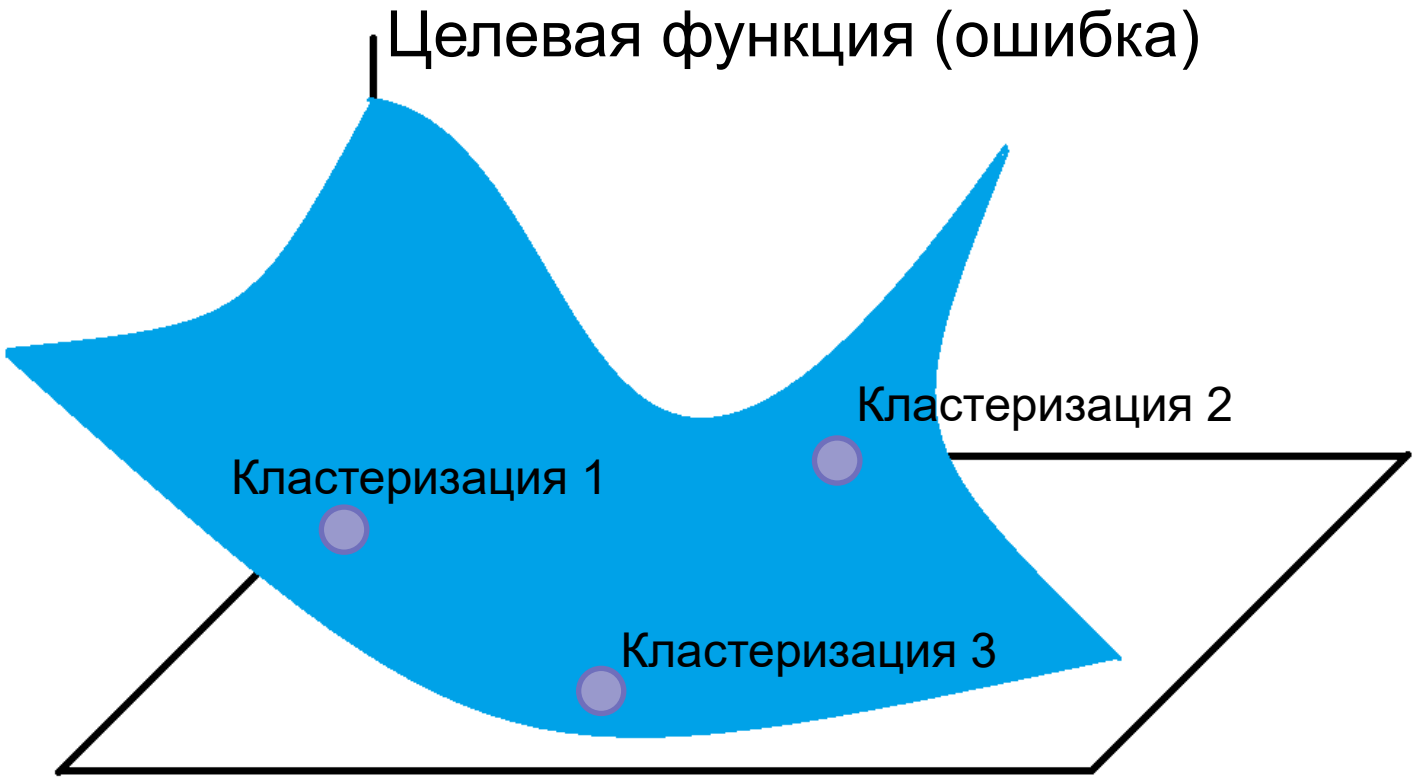
Мера похожести объектов

$$F = \frac{\sum_k \sum_{l,m} \Phi_1 \left(\begin{matrix} \text{расстояние между} \\ \text{наблюдениями } x_l \text{ и } x_m \text{ в кластере } k \end{matrix} \right)}{\sum_{i,j} \Phi_2(\text{расстояние между кластерами } i \text{ и } j)}$$

Мера близости кластеров

- Большое межкластерное расстояние → F уменьшается
- Малое внутрикластерное расстояние → F уменьшается

Оптимизация NGC



Рассматриваем значения целевой функции при различных кластеризациях и выбираем наименьшее

Типы алгоритмов

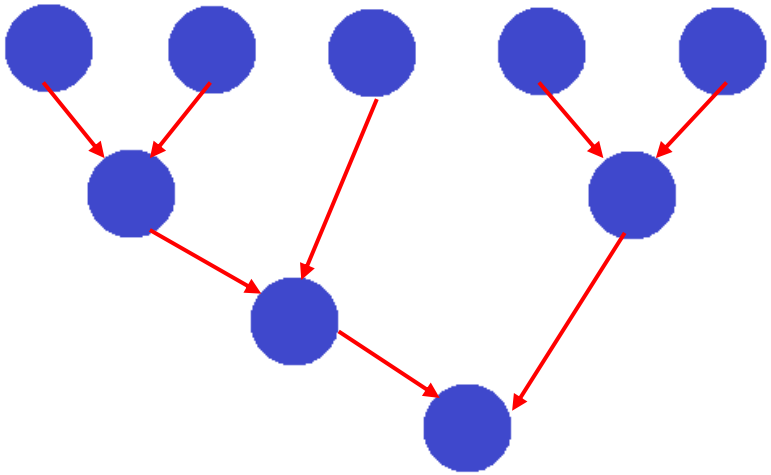
По оптимизируемой функции

- Оптимизирующие NGC
  - Группировка (разбиение) *(например K-means)*
  - Иерархические
- Оптимизирующие NGC в условиях ограничений
  - Параметрическое семейство алгоритмов (*Expectation-Maximization*)
  - Непараметрическое семейство алгоритмов *(Density / Kernel-based*)

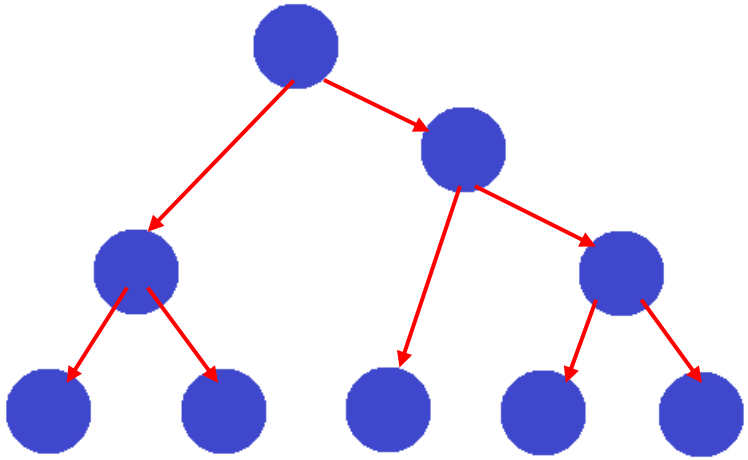
Иерархическая кластеризация

Способ добавления объектов в кластеры

Восходящая («склейка»)

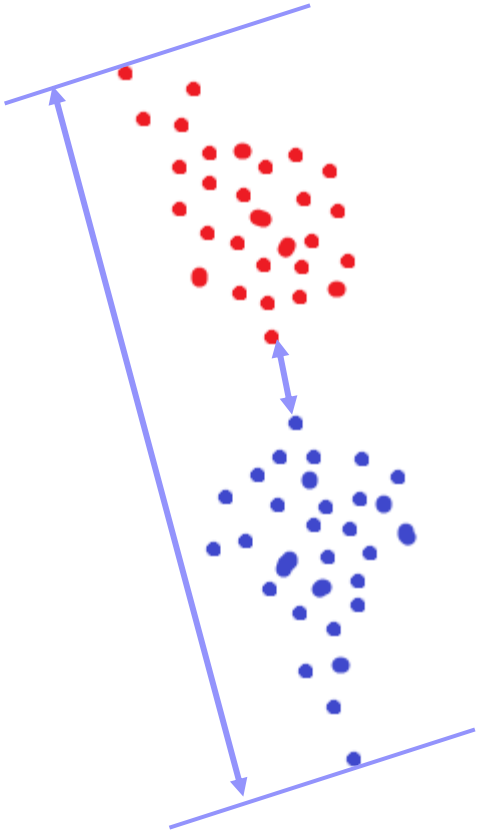


Нисходящая («разбиение»)



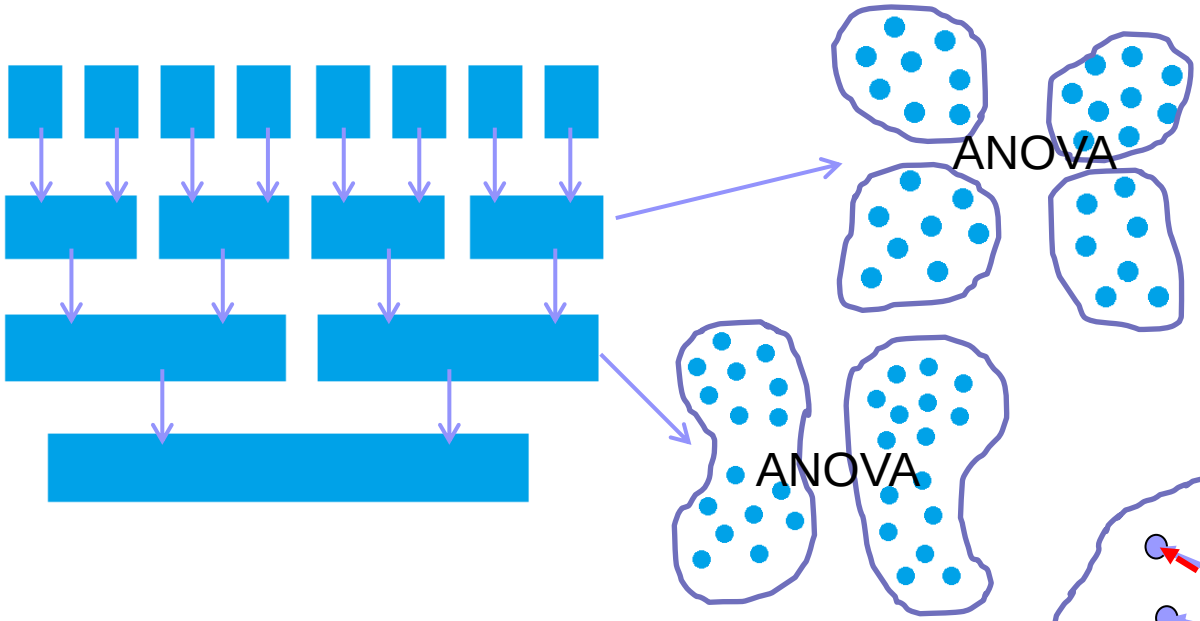
Оценка близости кластеров

- Расчет расстояния на основе попарных расстояний между элементами различных кластеров:
  - **Полное связывание**: наибольшее попарное расстояние. Дает компактные сферические кластеры.
  - **Среднее связывание**: усредненное попарное расстояние. Редко используется.
  - **Единственное связывание**: наименьшее попарное расстояние. Дает «растянутые» кластеры сложной формы.
  - **Центроидное связывание*:*** расстояние между центрами (мат. ожидание) кластеров.
  - Другие методы (например **метод Ward'a** – минимизирует внутрикластерные дисперсии или другую целевую функцию)



Стандартные меры связывания

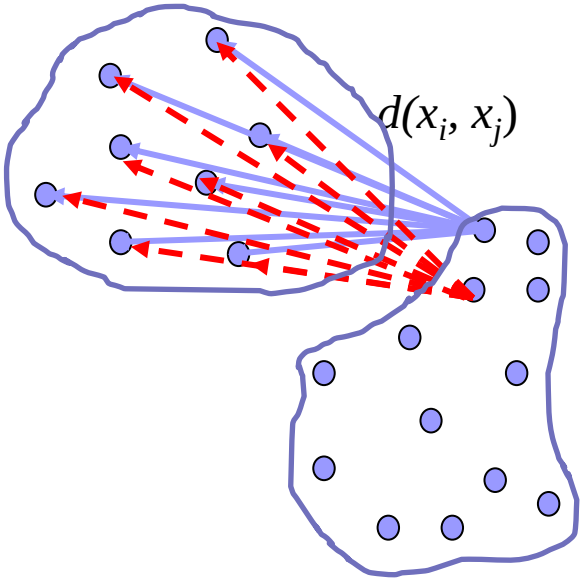
- Ward использует ANOVA



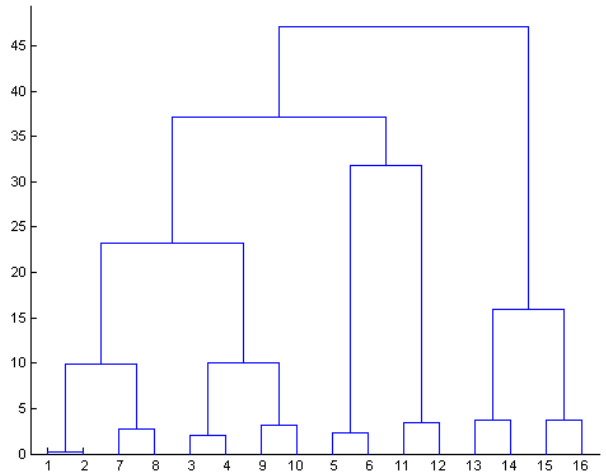
$$D_{KL} = \frac{\left\| \bar{x}_K - \bar{x}_L \right\|^2}{\left(\frac{1}{n_K} + \frac{1}{n_L} \right)}$$

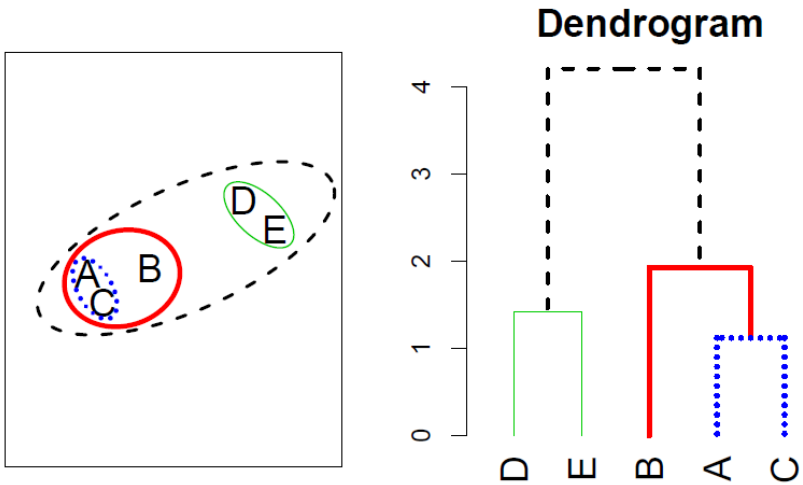
- Среднее связывание

$$D_{KL} = \frac{1}{n_K n_L} \sum_{i \in C_K} \sum_{j \in C_L} d\left(x_i, x_j\right)$$



Представление иерархических кластеров - Дендрограмма



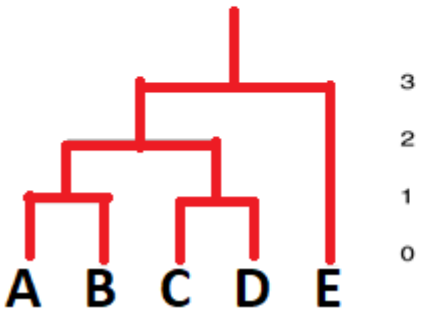


- бинарное дерево, описывающее все шаги разбиения
- Корень – общий кластер, листья - элементы
- «Высота» ветвей (до пересечения) – порог расстояния «склейки» («разделения»)
- Результат кластеризации – «срез» дендрограммы

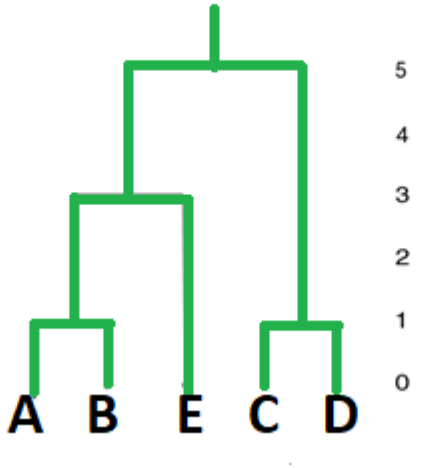
Многое зависит от расстояния

	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0

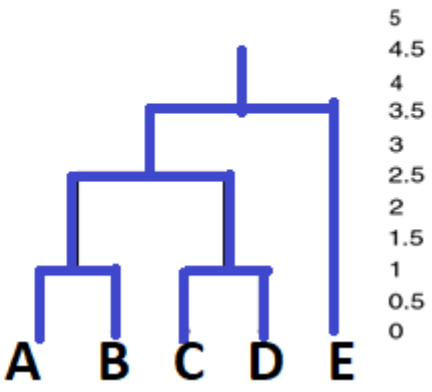
Single Link



Complete Link



Average Link

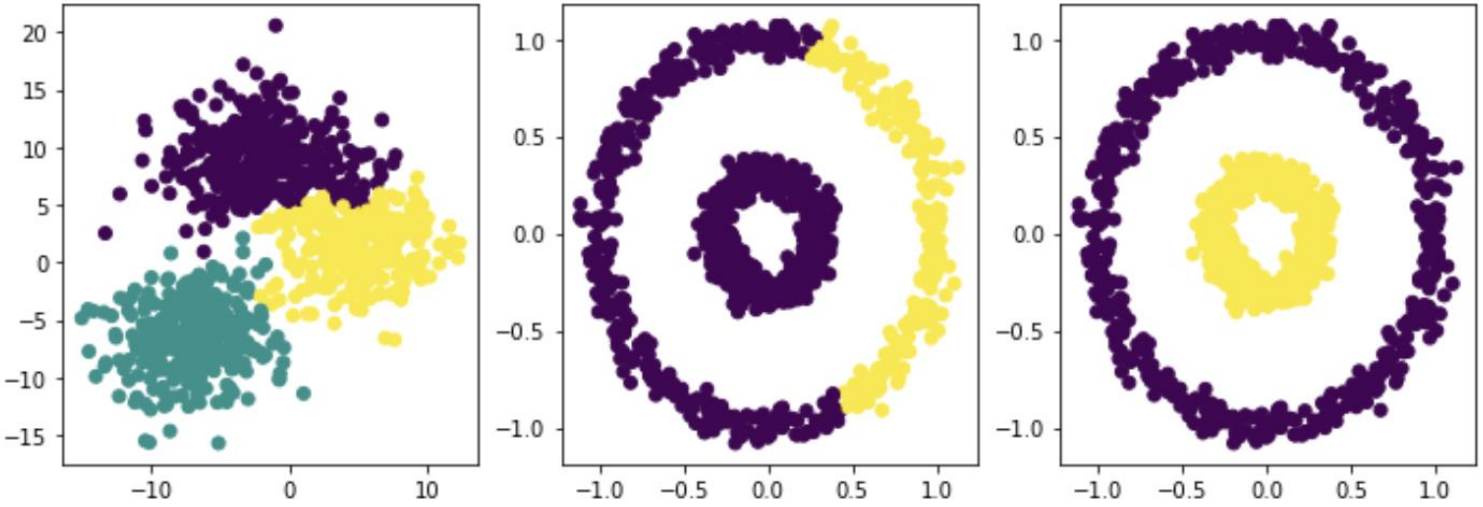


Обсуждение иерархической кластеризации

- Достоинства:
  - Очень просто и понятно, легко реализовать
  - Наглядные дендрограммы
  - Нет метапараметров
- Недостатки:
  - не масштабируется: временная сложность $O(n^2)$, где n - число кластеризуемых объектов
  - жадный алгоритм - локальная оптимальность с точки зрения минимизации внутриклассовых расстояний и максимизации межклассовых
  - относительно слабая интерпретируемость

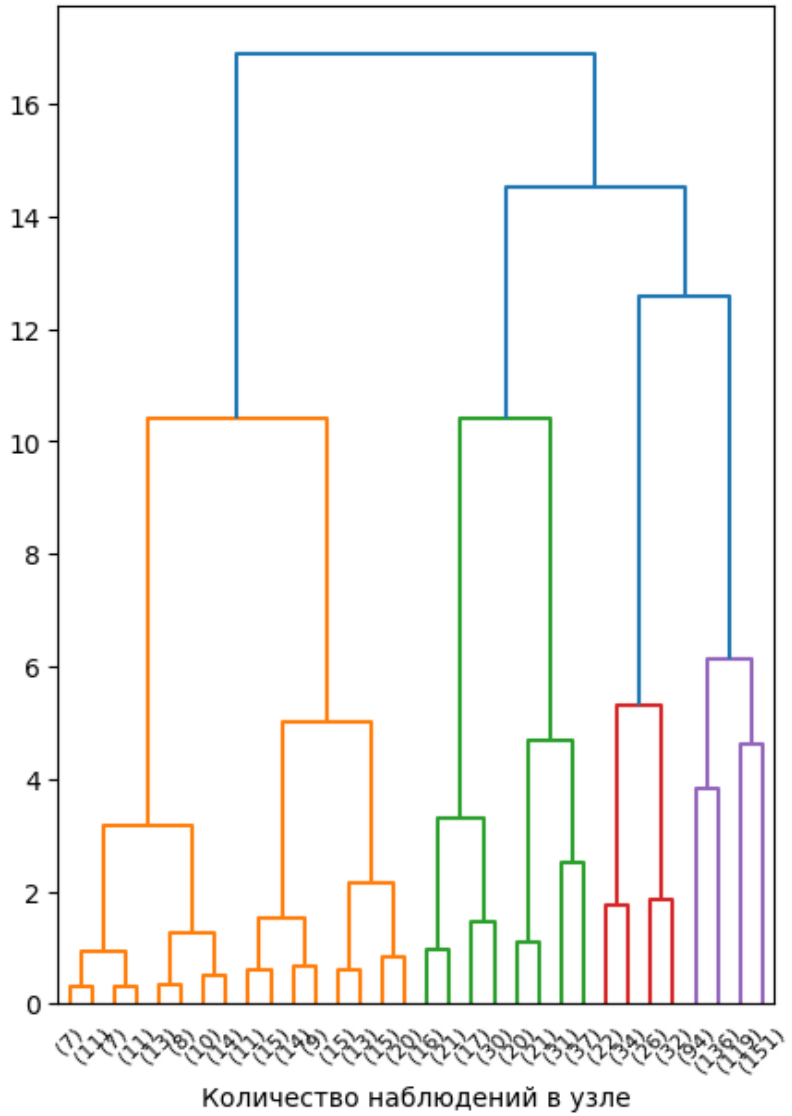
Пример использования

```
from sklearn.cluster import AgglomerativeClustering

fig, axes = plt.subplots(ncols=3, figsize=(12,4))
agg_bl = AgglomerativeClustering(n_clusters=3)
axes[0].scatter(X_blob[:, 0], X_blob[:, 1], c=agg_bl.fit_predict(X_blob))

agg_cl = AgglomerativeClustering(n_clusters=2)
axes[1].scatter(X_cl[:, 0], X_cl[:, 1], c=agg_cl.fit_predict(X_cl))

agg_cl1 = AgglomerativeClustering(n_clusters=2, linkage="single")
axes[2].scatter(X_cl[:, 0], X_cl[:, 1], c=agg_cl1.fit_predict(X_cl))
```



Пример использования дендрограммы

```python
from scipy.cluster.hierarchy import dendrogram

def plot_dendrogram(model, **kwargs):
    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
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)
    dendrogram(linkage_matrix, **kwargs)
    plt.xlabel("Количество наблюдений в узле")

model = AgglomerativeClustering(distance_threshold=0,
                                n_clusters=None)
model = model.fit(X_cl)
plot_dendrogram(model, truncate_mode="level", p=4)
```



Пример использования с матрицей данных и матрицей расстояний на основе Jaccard

```
df = pd.read_csv("C:\\SAS\\edu\\course\\DM 2017\\assc_TRANSACTION.csv")
df=df.groupby("CUSTOMER")["PRODUCT"].value_counts().unstack(fill_value=0)
df.head(5)
```

PRODUCT CUSTOMER	apples	artichok	avocado	baguette	bordeaux	bourbon	chicken	coke	corned_b	cracker	ham	heineken	hering	ice_crea	olives	peppers
0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	0
1	0	0	0	1	0	0	0	0	1	1	0	1	1	0	1	0
2	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0
3	0	0	0	0	0	1	0	1	0	0	1	0	0	1	1	1
4	1	0	1	0	0	0	0	0	1	0	0	0	1	0	1	0

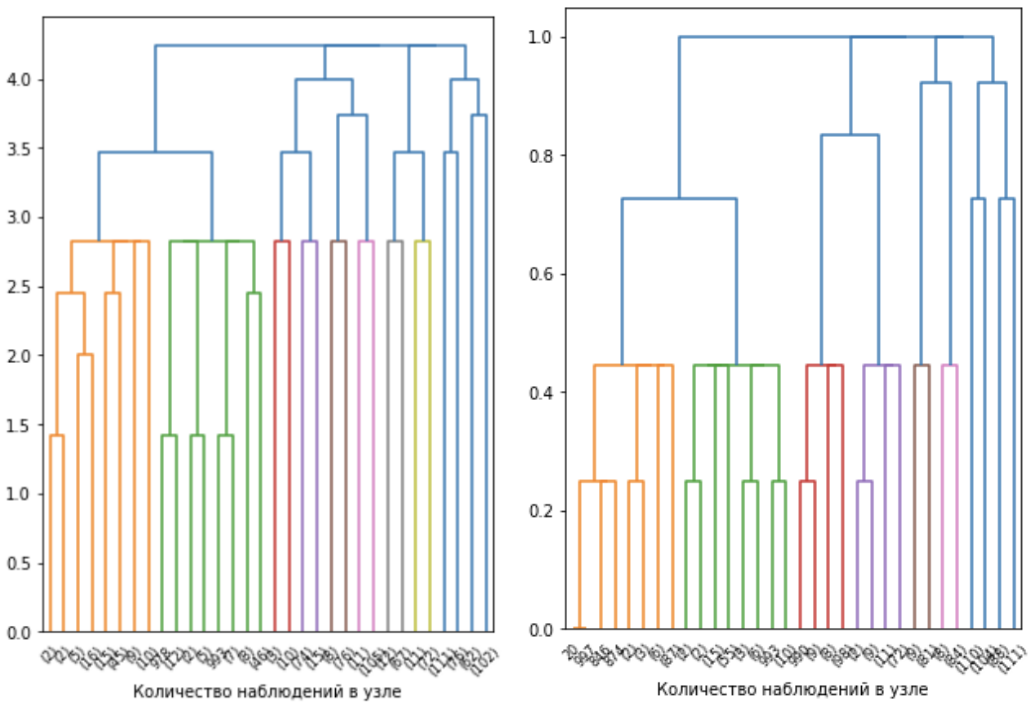
```
#рассчет матрицы сходства на основе коэф Jaccard'а
from sklearn.metrics import jaccard_score as js

N = df.axes[0].values.size
dist_matrix=np.zeros(N*N)
dist_matrix=dist_matrix.reshape(N,N)
for i in range(1,N):
    for j in range(0,i):
        dist_matrix[i,j]=dist_matrix[j,i]=1-js(df.iloc[j]>0,df.iloc[i]>0)
```

```
pd.DataFrame(dist_matrix).head(5)
```

	0	1	2	3	4	5	6	7	8	9	...	991	992	993	994	995
0	0.000000	0.727273	0.833333	0.444444	0.600000	0.833333	0.600000	0.727273	0.833333	0.727273	...	0.600000	0.916667	0.727273	0.923077	0.833333
1	0.727273	0.000000	0.833333	0.923077	0.727273	0.923077	0.833333	0.727273	0.444444	0.727273	...	0.833333	0.818182	0.600000	0.727273	0.444444
2	0.833333	0.833333	0.000000	0.833333	0.833333	0.727273	0.833333	0.923077	0.833333	0.923077	...	0.923077	0.375000	0.833333	0.727273	0.833333
3	0.444444	0.923077	0.833333	0.000000	0.833333	0.600000	0.444444	0.727273	0.727273	0.833333	...	0.444444	0.916667	0.923077	1.000000	0.923077
4	0.600000	0.727273	0.833333	0.833333	0.000000	1.000000	0.727273	0.923077	0.923077	0.833333	...	0.727273	0.916667	0.444444	0.727273	0.833333

Пример использования с матрицей данных и матрицей расстояний на основе Jaccard

```
model = AgglomerativeClustering(distance_threshold=0, linkage="complete",
                                n_clusters=None)
model = model.fit(df)
plot_dendrogram(model, truncate_mode="level", p=4)

model = AgglomerativeClustering(affinity="precomputed", linkage="complete",
                                distance_threshold=0, n_clusters=None)
model = model.fit(dist_matrix)
plot_dendrogram(model, truncate_mode="level", p=4)
```



Кластеризация на основе строгой группировки (partitioning):

- Основная задача:
  - Найти такое разбиение *C* исходного множества *X* из *N* объектов на *K* непересекающихся подмножеств C_k, покрывающих *X*, чтобы внутриклассовое расстояние было минимальным:

$$\min_{C_i \cap C_j = \varnothing, \bigcup C_i = X} \sum_{i=1}^{K} \sum_{x \in C_i} \sum_{x' \in C_i} d(x, x')$$

- Точное решение – перебор с отсечением
  - метод «ветвей и границ», но число комбинаций неприемлемо даже для 100 объектов:

$$S(N,K) = \frac{1}{K!} \sum_{i=1}^{K} (-1)^{K-i} \binom{K}{i} K^N$$

- Эвристические методы:
  - K-means (прототип кластера – мат. ожидание *m*), K-medoids (прототип кластера – средний элемент)

$$\min_{C_i \cap C_j = \varnothing, \bigcup C_i = X} \sum_{i=1}^{K} \sum_{x \in C_i} d(m_i, x)$$

  - ищется локальный минимум

Переборные алгоритмы разбиения

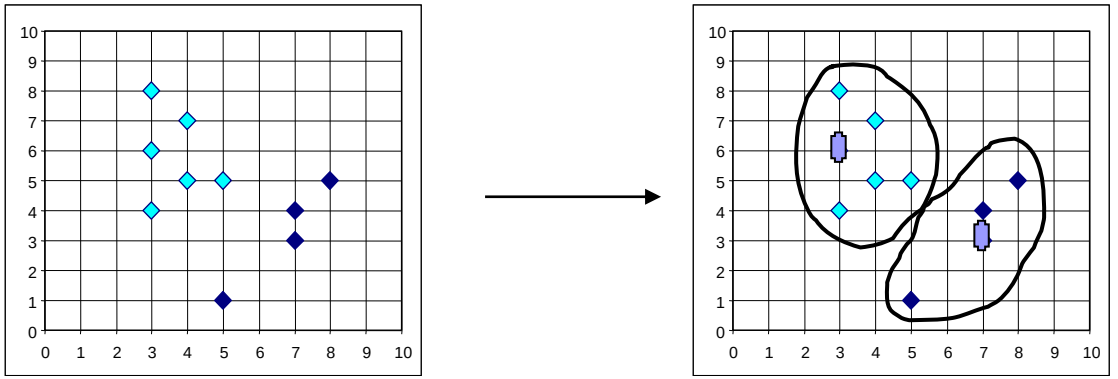
1. Начальные приближения центров кластеров и отнесение каждого наблюдения к одному (ближайшему) из центров
2. Расчет штрафной функции для всех вариантов переноса наблюдения из кластера в кластер
3. Выбор лучшего переноса
4. Повторение Шагов 2-3 до остановки

K-Medoids

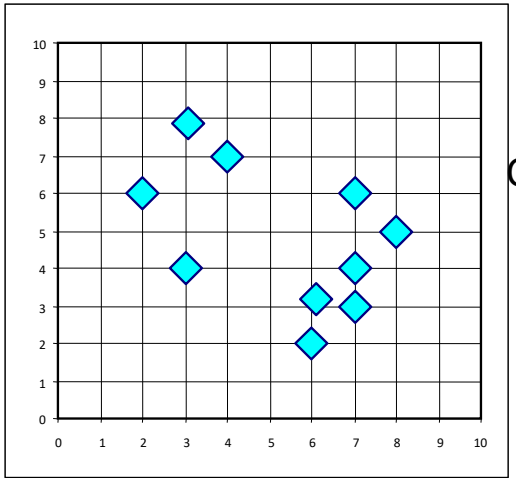
- K-Medoids:
  - Идея: вместо мат. ожидания кластера ищется представительный («наиболее центральный») объект – medoid
  - Процесс: случайная инициализация, переход к новому medoid, если это улучшает целевую функцию:

$$\min_{C_i \cap C_j = \varnothing, \bigcup C_i = X} \sum_{i=1}^{K} \sum_{x, m_i \in C_i} d(m_i, x)$$

- НО: не масштабируется и вычислительно неэффективный:
  - $O(K(N-K)^2)$ для каждой итерации

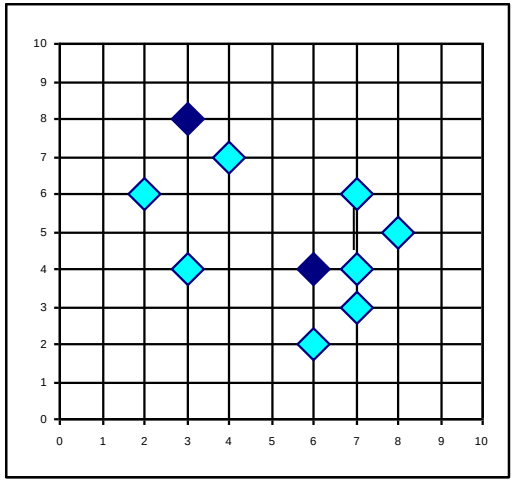


Алгоритм K-Medoids

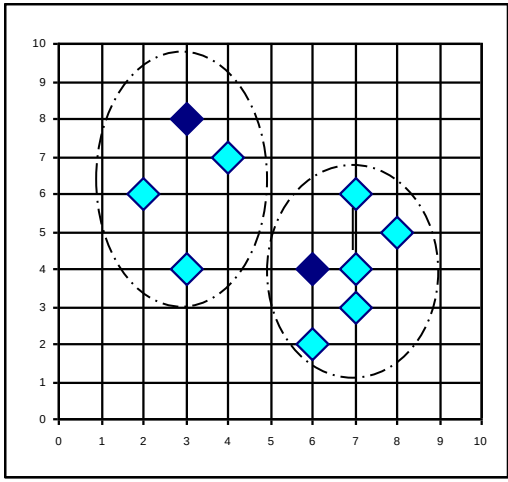


K=2

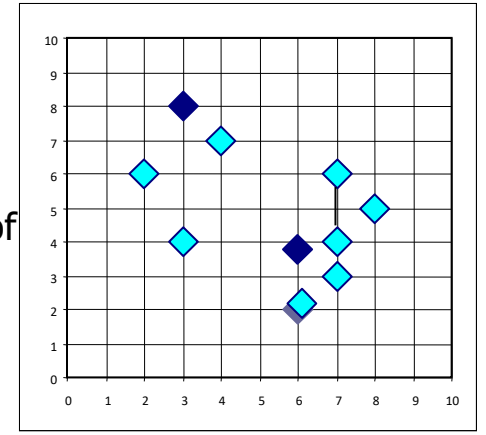
Случайн. K medoids



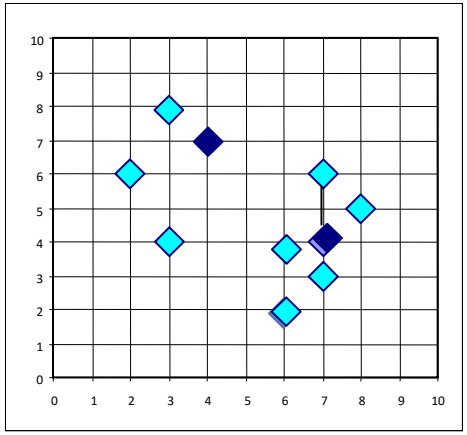
Каждый объект к ближ. medoid



Перебор всех кандидатов на замену medoid



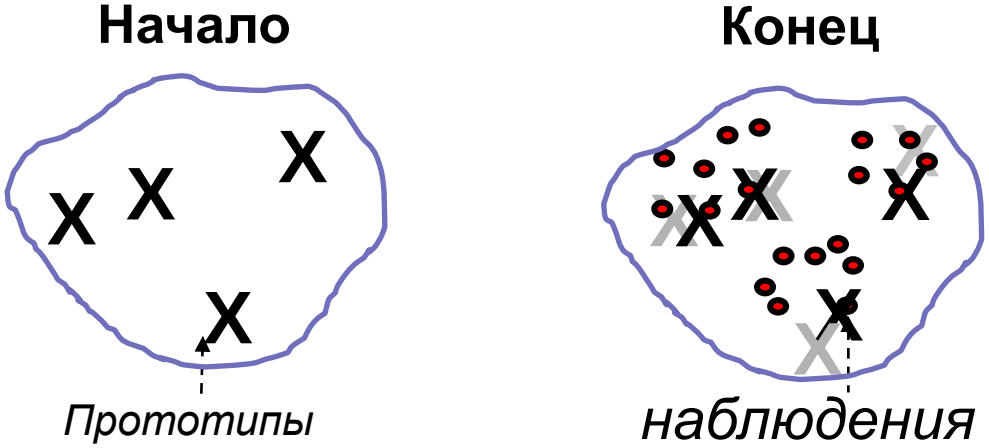
Расчет стоимости перехода (total cost of swapping)



Если качество кластеризац. улучш., то переход к нов. medoid

Пока есть изменения

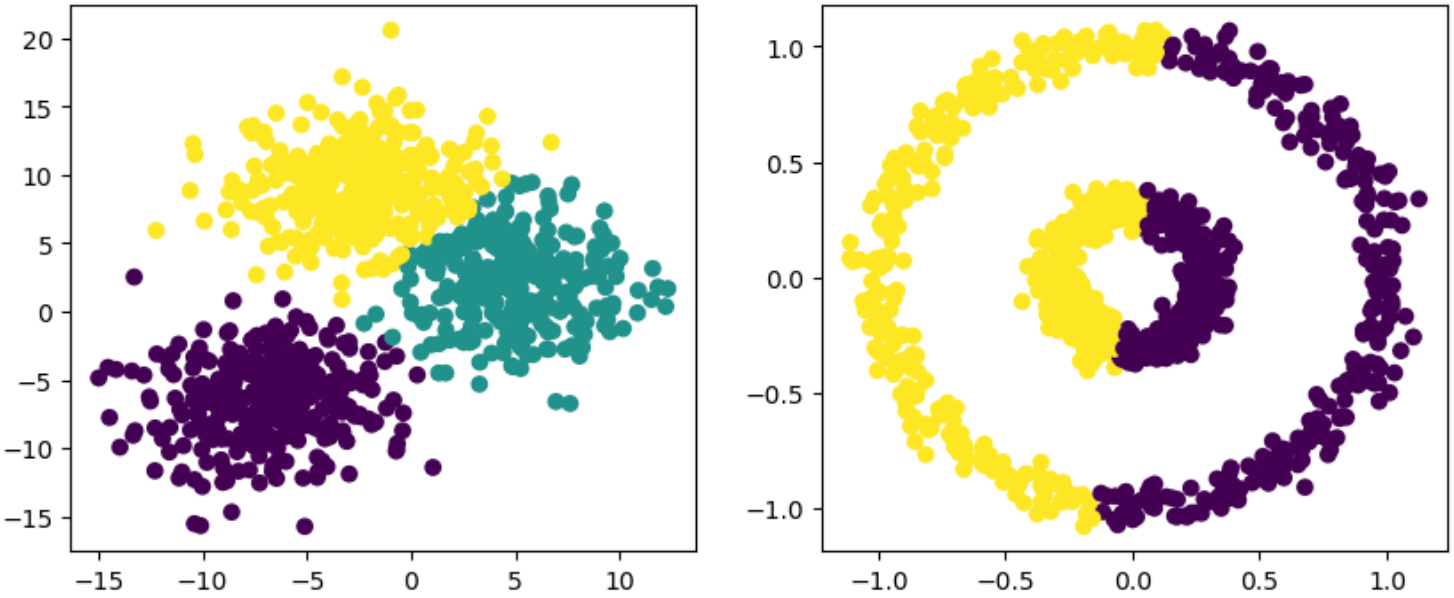
Основные проблемы строгой группировки



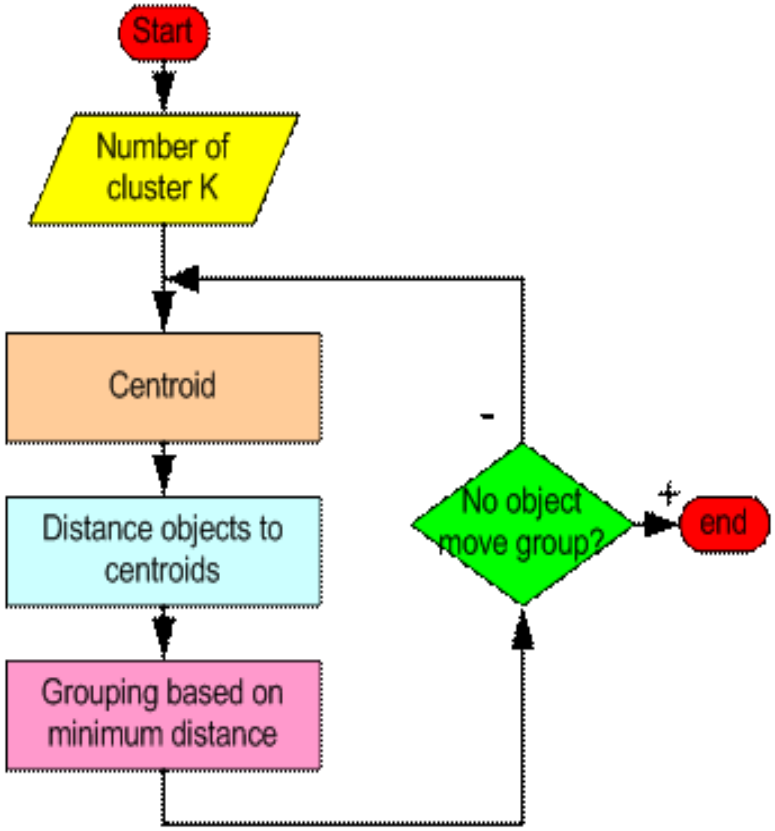
- Угадать число кластеров и хорошие начальные приближения
- Сферические кластеры только
- Влияют начальные приближения, выбросы, порядок обхода выборки
- Не находит глобально оптимального решения!!!!

Пример использования

```
# Расширенный функционал scikit-learn
# Установка: "pip install scikit-learn-extra"
from sklearn_extra.cluster import KMedoids

fig, axes = plt.subplots(ncols=2, figsize=(10,4))
km_bl = KMedoids(n_clusters=3)
axes[0].scatter(X_blob[:, 0], X_blob[:, 1],
                c=km_bl.fit_predict(X_blob))

km_cl = KMedoids(n_clusters=2)
axes[1].scatter(X_cl[:, 0], X_cl[:, 1],
                c=km_cl.fit_predict(X_cl))
```



Метод K-Means

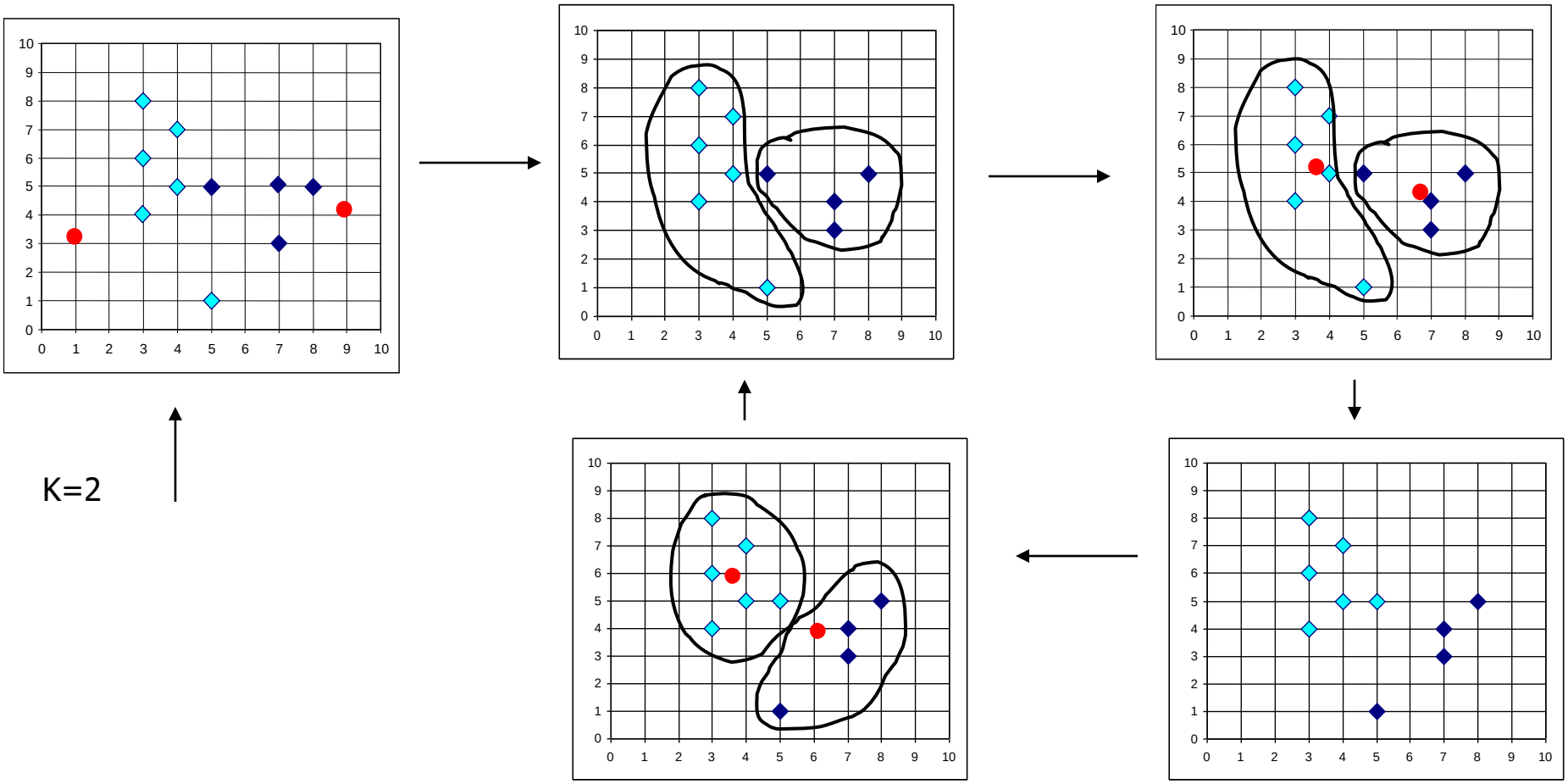


- Шаг 0. Инициализация:
  - произвольное разбиение на заданное число кластеров K (где значение K выбирается по CCC на основе иерархической кластеризации)
- Шаг 1. Поиск центров:
  - Для всех K кластеров $m_i = \sum_{x \in C_i} x \Big/ \|C_i\|$
- Шаг 2. Расчет расстояний до центров:
  - Для всех N объектов и K кластеров $d(m_i, x) = \sum_{x \in C_i} x \Big/ \|C_i\|$
- Шаг 3. Выбор ближайшего кластера:

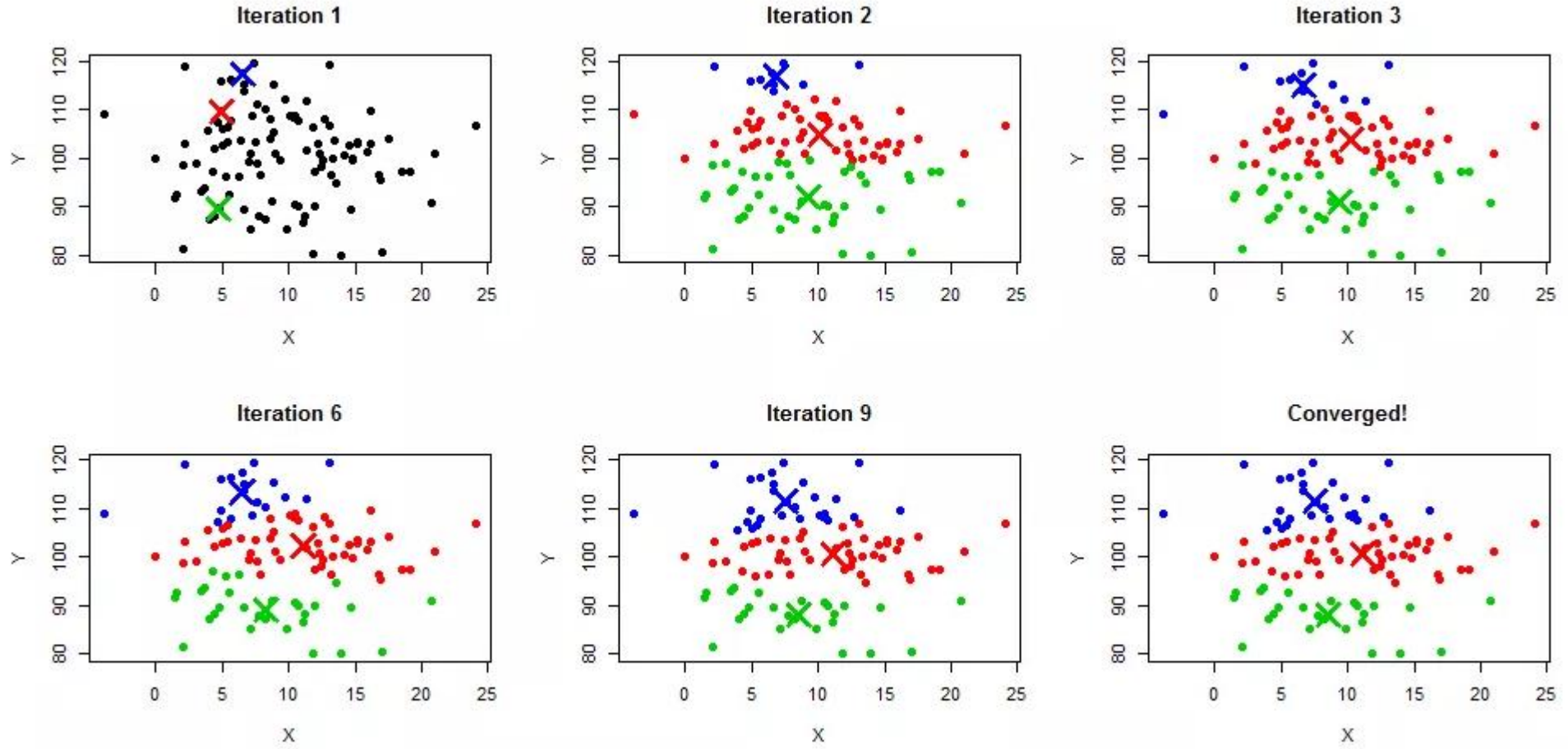
$$x \in C_i \Leftrightarrow i = \min_j d(m_j, x)$$

- Если были перестановки, то Шаг 1.

Пример

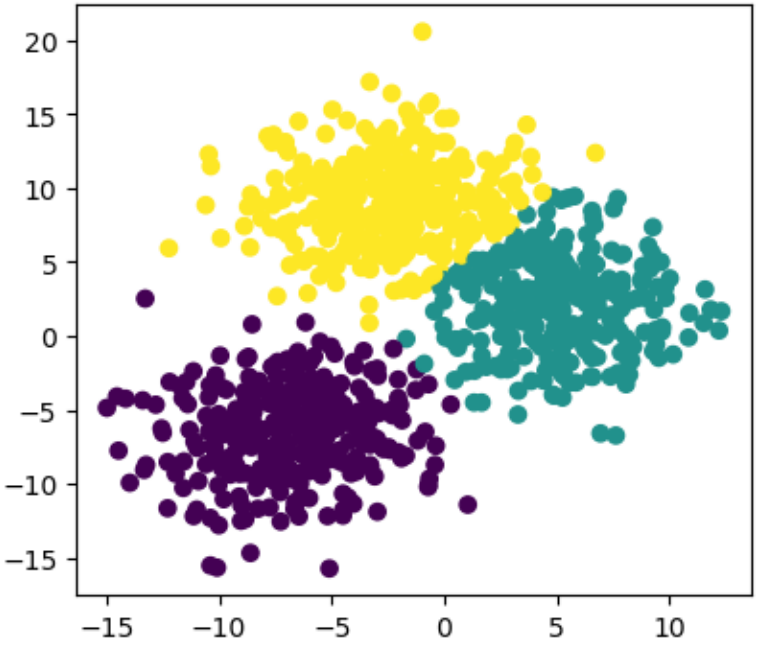


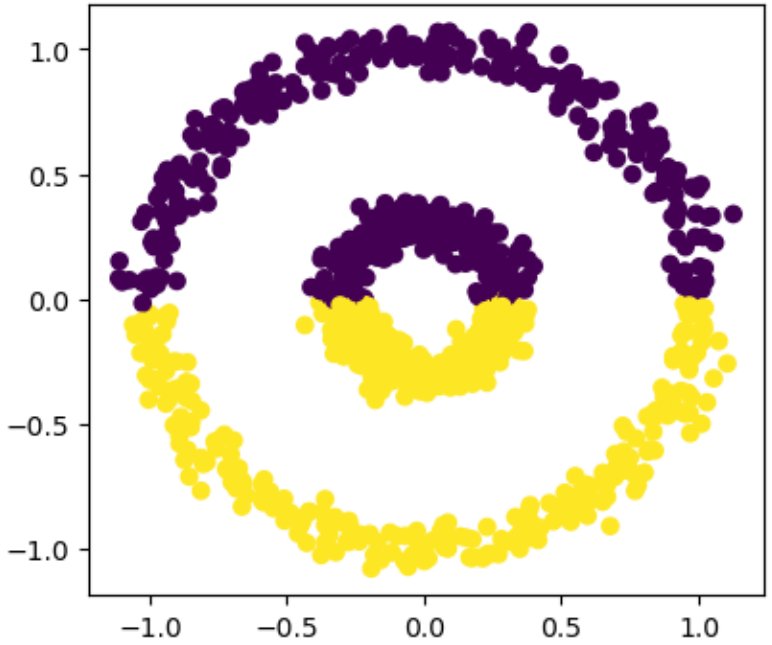
k-Means алгоритм



Пример использования

```
from sklearn.cluster import KMeans

fig, axes = plt.subplots(ncols=2, figsize=(10,4))
km_bl = KMeans(n_clusters=3)
axes[0].scatter(X_blob[:, 0], X_blob[:, 1],
                c=km_bl.fit_predict(X_blob))

km_cl = KMeans(n_clusters=2)
axes[1].scatter(X_cl[:, 0], X_cl[:, 1],
                c=km_cl.fit_predict(X_cl))
```





Особенности *K-Means*

- Достаточно быстрый
- Локальный экстремум:
  - Глобальный можно искать «разумным» перебором: имитация отжига, генетические алгоритмы и т.д.
  - В SAS на основе нескольких инициализаций
- Недостатки
  - Числовые данные (иначе как найти центр?) – моды, центр кластера – число (для числовых атрибутов)
  - Необходимо задавать K заранее (есть методы «отбора» K)
  - Чувствительность к шуму и выбросам, кластеры сферической формы

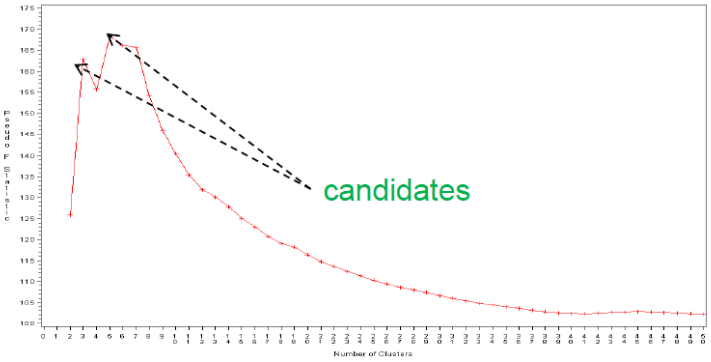
Определение числа кластеров

- По сути – перебор моделей с разным числом кластеров и выбор по некоторому критерию, например Pseudo-F (Calinski and Habarasz, 1974)

$$W_j = \sum_{i \in C_j} \| \mathbf{T}(v_i) - \overline{\mathbf{T}}_j \|^2 \qquad Q = \sum_{i=1}^{V} \| \mathbf{T}(v_i) - \overline{\mathbf{T}} \|^2$$

  - N- число объектов,
  - G – число кластеров,
  - W-сумма внутри-кластерных квадратов расстояний,
  - Q – сумма всех кв. расстояний до общего центра

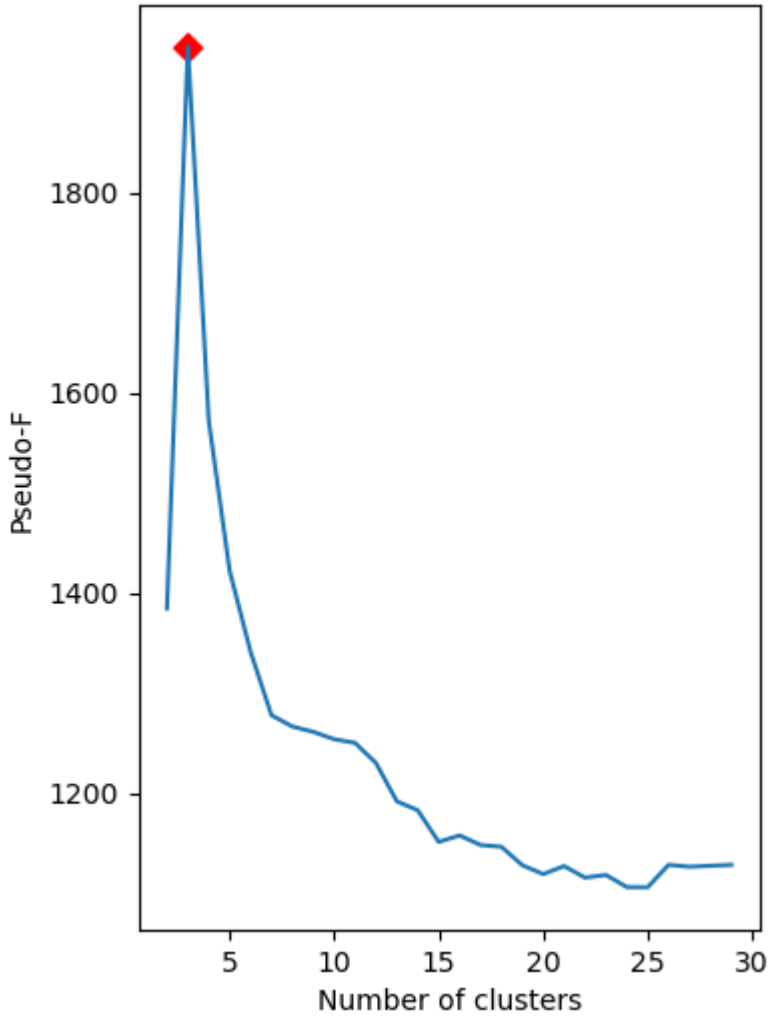
$$pseudoF = \frac{(Q - \sum_{g=1}^{G} W_g)/(G-1)}{\sum_{g=1}^{G} W_g /(N-G)}$$



  - Выбирается вариант с максимальным *pseudoF*

Пример использования

```python
def sum_dist_to_center(X):
    center = np.mean(X, axis = 0)
    return ((X - center)**2).sum()

def choose_num_clusters(X, max_clust = 30):
    N = X.shape[0]
    Q = sum_dist_to_center(X)
    pseudo_f = np.array([])
    for G in range(2, max_clust):
        clustering = KMeans(n_clusters = G).fit(X)
        W = 0
        for l in range(G):
            elems = X[clustering.labels_ == l]
            W += sum_dist_to_center(elems)
        fisher_stat = ((Q - W)/(G - 1))/(W/(N - G))
        pseudo_f = np.append(pseudo_f, fisher_stat)
    plt.plot(range(2, max_clust), pseudo_f)
    ind_best_clust = np.argmax(pseudo_f)
    plt.scatter(ind_best_clust + 2,
                pseudo_f[ind_best_clust],
                color="r", marker="D", s=50)
    plt.xlabel("Number of clusters")
    plt.ylabel("Pseudo-F")
    return ind_best_clust + 2

k = choose_num_clusters(X_blob)
```



Однородность, полнота и V-мера (если все-таки есть отклик)

- Пусть n – число наблюдений, C – число классов, K – число кластеров
- $n_{c,k}$ - число наблюдений класса c и кластера k
- $H(C)$ – энтропия классов
- $H(C|K)$ – условная энтропия классов
- Homogeneity – кластер содержит только членов одного класса
- Completeness – все члены класса относятся к одному и тому же кластеру
- V-measure – гармоническое среднее

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{n_{c,k}}{n}\cdot\log\left(\frac{n_{c,k}}{n_k}\right)$$

$$H(C) = -\sum_{c=1}^{|C|}\frac{n_c}{n}\cdot\log\left(\frac{n_c}{n}\right)$$

$$h = 1 - \frac{H(C|K)}{H(C)}$$

$$c = 1 - \frac{H(K|C)}{H(K)}$$

$$v = 2\cdot\frac{h\cdot c}{h+c}$$

```
score_funcs = [
    ("Homogen", metrics.homogeneity_score),
    ("Complete", metrics.completeness_score),
    ("V-measure", metrics.v_measure_score)
]
```

Другие метрики (если все-таки есть отклик)

- Пусть C – истинный класс,
  K – результат кластеризации
- Индекс Рэнда
  - a – кол-во пар элементов: C = K
  - b – кол-во пар элементов: C != K
  - $C_2^{n_{samples}}$ – кол-во всевозможных пар
  - Не гарантирует малое значение для случайного прогноза
- Скорректированный индекс Рэнда – снижение ценности случайных маркировок

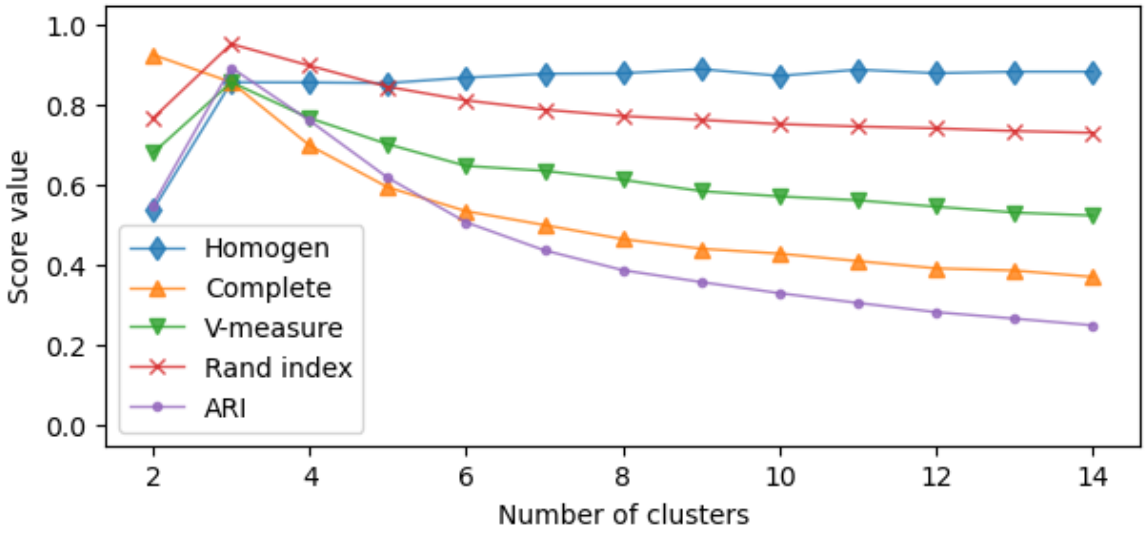
$$\mathrm{RI} = \frac{a+b}{C_2^{n_{samples}}}$$

$$\mathrm{ARI} = \frac{\mathrm{RI} - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]}$$

```
score_funcs = [
    ("Rand index", metrics.rand_score),
    ("ARI", metrics.adjusted_rand_score)
]
```

Пример использования

```
def apply_clustering(X, y, score_func, cl_range):
    scores = np.zeros(len(cl_range))
    for i, n_clust in enumerate(cl_range):
        cl = KMeans(n_clusters = n_clust).fit(X)
        scores[i] = score_func(y, cl.labels_)
    return scores

cl_range = range(2, 15)
plots, names = [], []

for marker, (name, func) in zip("d^vx.,", score_funcs):
    scores = apply_clustering(X_blob, y_blob, func, cl_range)
    plots.append(plt.errorbar(cl_range, scores, alpha=0.8,
                              linewidth=1, marker=marker)[0])
    names.append(name)
```



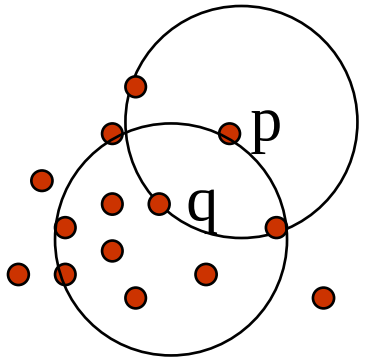
Непараметрическая кластеризация на основе связности

- Основные свойства:
  - Произвольная форма кластеров
  - Работа в условиях шума
  - Один проход базы
  - Недостаток: нужны параметры для «тонкой настройки»
- Популярные алгоритмы:
  - На основе расстояний DBSCAN: Ester, et al. (KDD’96)
  - На основе оценки плотности DENCLUE: Hinneburg & D. Keim (KDD’98)

DBSCAN: основные принципы

- Важные параметры:
  - *Eps*: радиус области поиска ближайших соседей
  - *MinPts*: минимальное число ближайших соседей в Eps-области
- Множество ближайших соседей:
  - $N_{Eps}(p)$:*{q belongs to D | dist(p,q) <= Eps}*
- Непосредственно достижимые точки:
  - Точка *p* напрямую непосредственно достижима из *q* с учетом. *Eps*, *MinPts,* если *p* принадлежит $N_{Eps}(q)$
- Ядровая точка:

$$|N_{Eps}(q)| >= MinPts$$

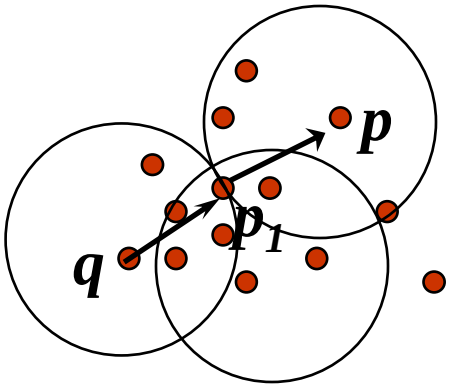


MinPts = 5

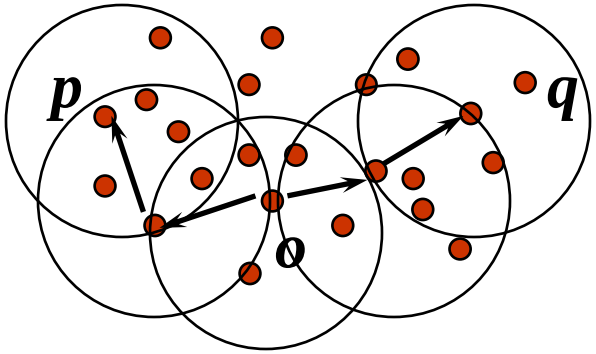
Eps = 1 cm

DBSCAN: основные принципы

- Достижимость:
  - Точка *p* достижима из *q* с учетом *Eps*, *MinPts,* если существует путь p_1, …, p_n, $p_1 = q$, $p_n = p$ такой, что p_{i+1} непосредственно достижима из p_i

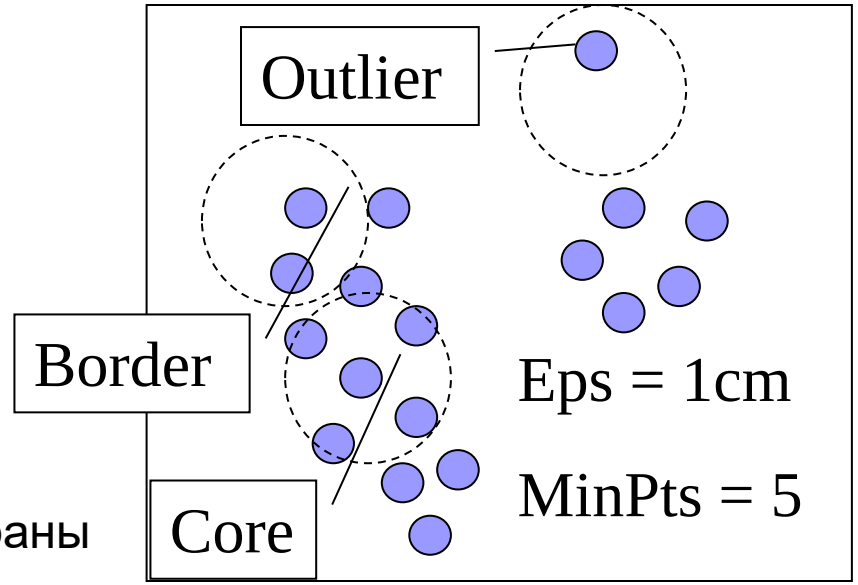


- Связность:
  - Точка *p* связана с *q* с учетом *Eps*, *MinPts,* если существует точка *o* *такая, что* обе точки *p* и *q* достижимы из *o* с учетом *Eps* и *MinPts*.



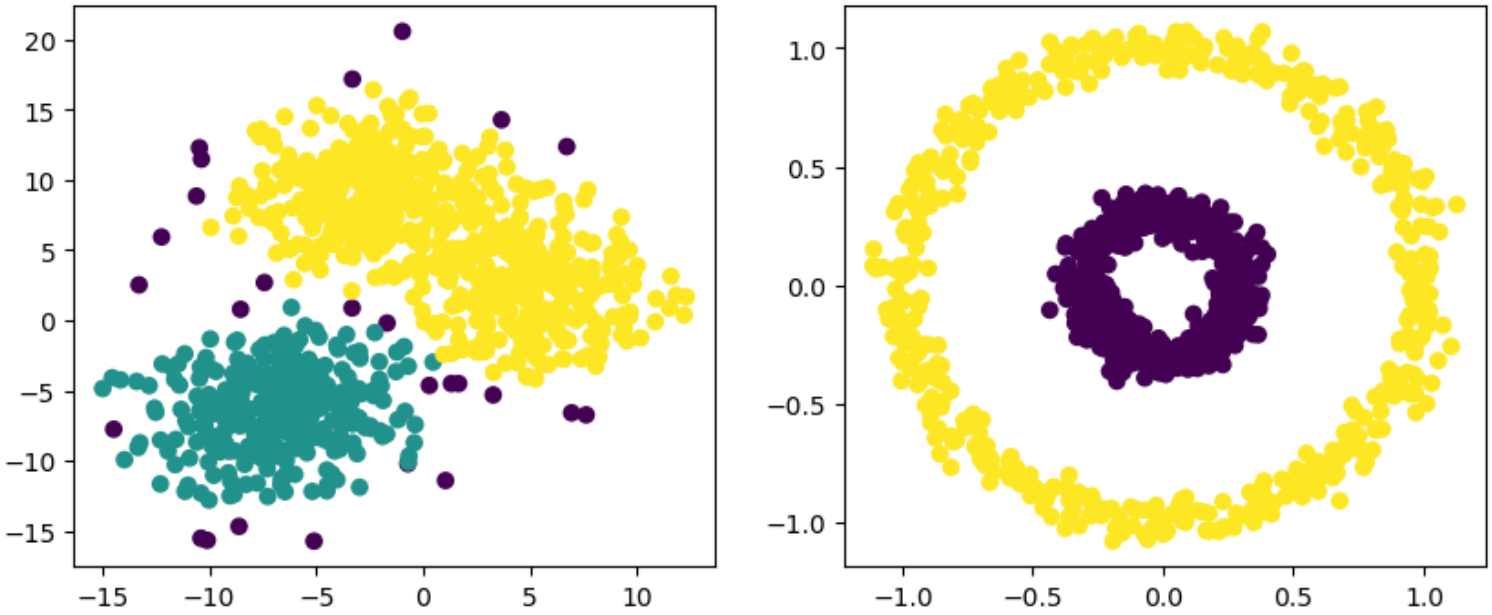
DBSCAN: Density Based Spatial Clustering of Applications with Noise

- Основан на понятии связного кластера:
  - Кластер определен как максимальное множество связных точек
- Позволяет находить кластеры произвольной формы в условиях шума
- Процедура:
  - Произвольный выбор точки *p*
  - Выбор всех достижимых из *p* точек с учетом *Eps* и *MinPts*.
  - Если *p* – ядровая, то кластер сформирован
  - Если *p* граничная или выброс, то обработка следующей точки
  - Продолжать пока не будут выбраны все точки



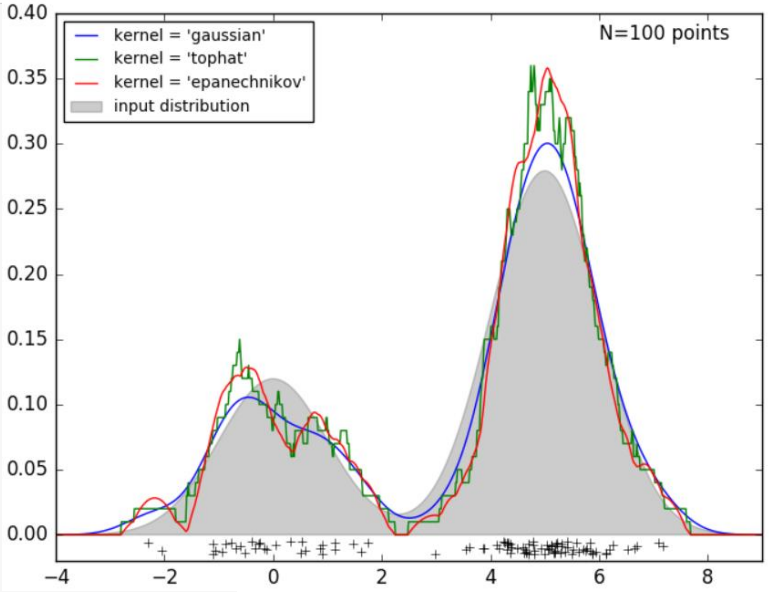
Пример использования

```
from sklearn.cluster import DBSCAN

fig, axes = plt.subplots(ncols=2, figsize=(10,4))
gbsc_bl = DBSCAN(eps=1.5, min_samples=5)
axes[0].scatter(X_blob[:, 0], X_blob[:, 1],
                c=gbsc_bl.fit_predict(X_blob))

gbsc_cl = DBSCAN(eps=0.1)
axes[1].scatter(X_cl[:, 0], X_cl[:, 1],
                c=gbsc_cl.fit_predict(X_cl))
```

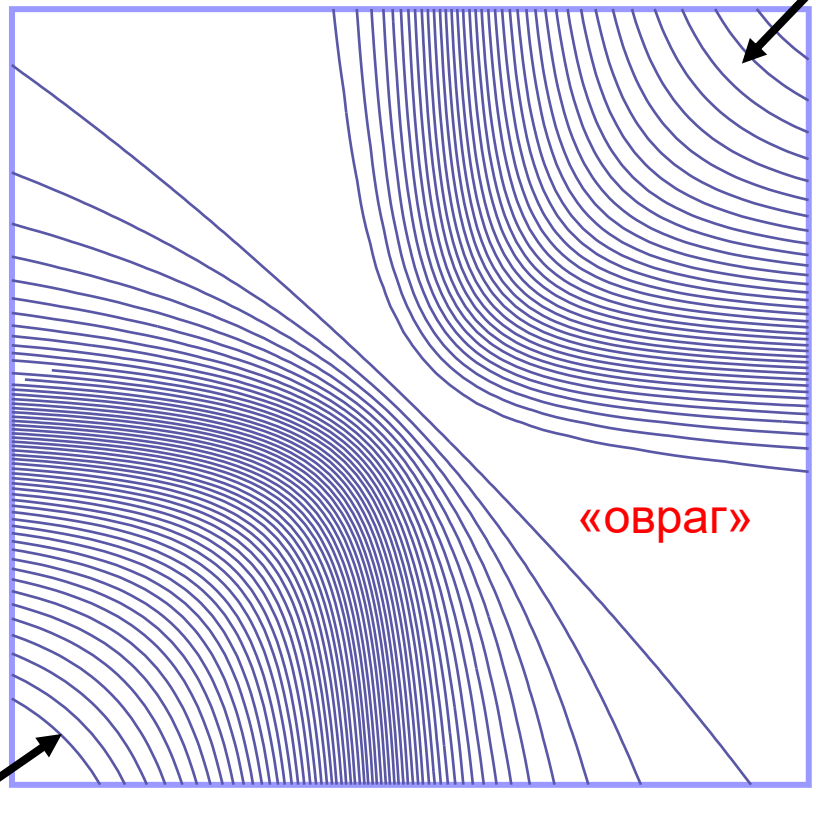


Kernel-приближение плотности распределения

```
from scipy.stats import norm
from sklearn.neighbors import KernelDensity

N = 100
X = np.concatenate(
    (np.random.normal(0, 1, int(0.3 * N)),
     np.random.normal(5, 1, int(0.7 * N)))
)[:, np.newaxis]

X_plot = np.linspace(-5, 10, 1000)[:, np.newaxis]

true_dens = (0.3 * norm(0, 1).pdf(X_plot[:, 0])
             + 0.7 * norm(5, 1).pdf(X_plot[:, 0]))

fig, ax = plt.subplots()
ax.fill(X_plot[:, 0], true_dens, fc='black', alpha=0.2,
        label='input distribution')
colors = ['navy', 'cornflowerblue', 'darkorange']
kernels = ['gaussian', 'tophat', 'epanechnikov']

for color, kernel in zip(colors, kernels):
    kde = KernelDensity(kernel=kernel, bandwidth=0.5).fit(X)
    log_dens = kde.score_samples(X_plot)
    ax.plot(X_plot[:, 0], np.exp(log_dens), color=color, lw=2,
            linestyle='-', label=f"kernel = '{kernel}'")

ax.legend(loc='upper left')
ax.plot(X[:, 0], -0.005 - 0.01 * np.random.random(X.shape[0]), '+k')
plt.show()
```

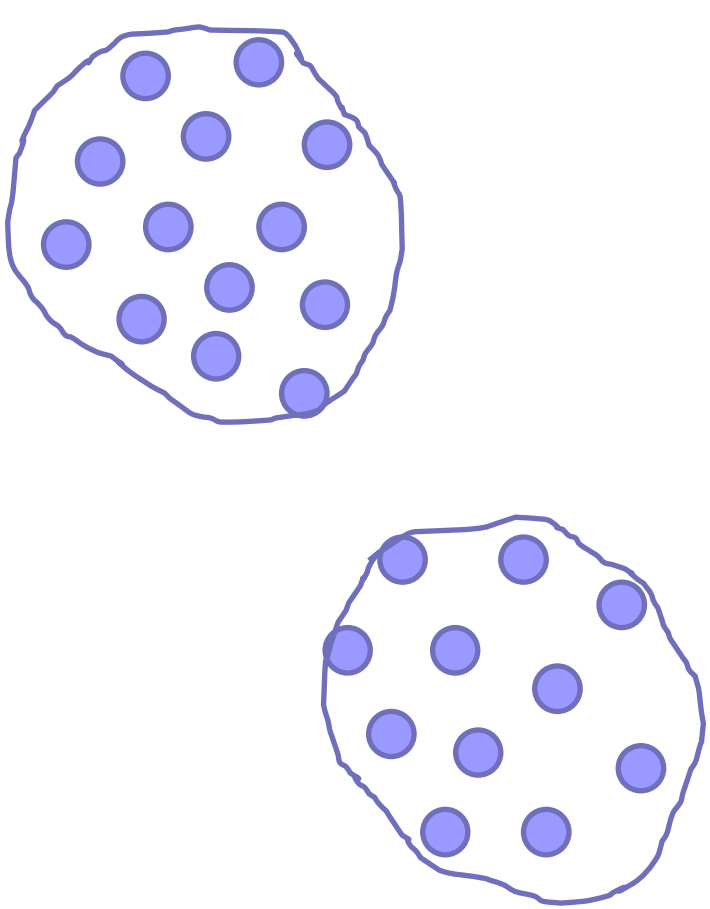


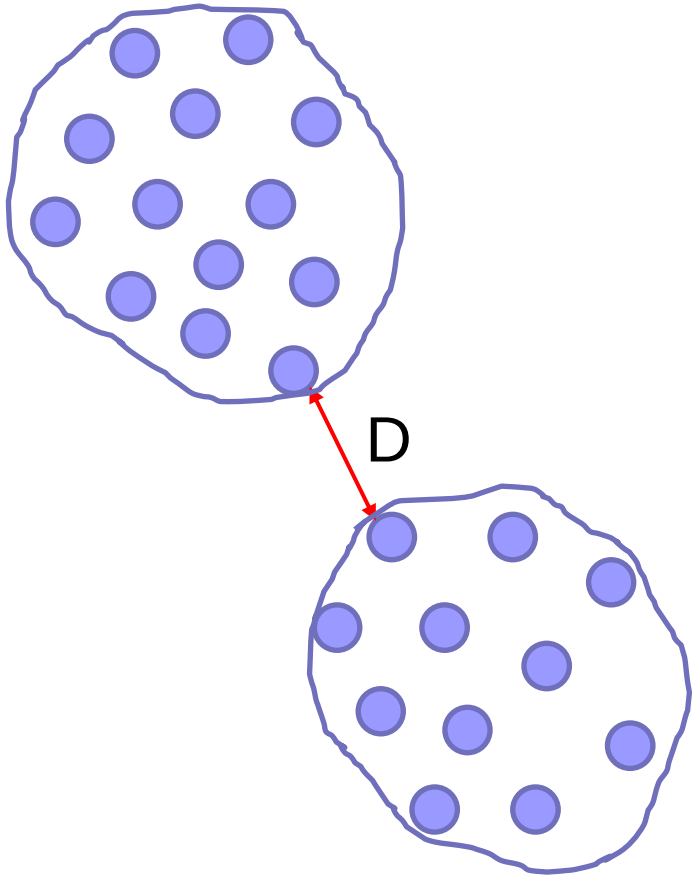
Поиск «оврагов» и «пиков» плотности распределения



Двух шаговое объединение кластеров

Поиск кластеров по модам



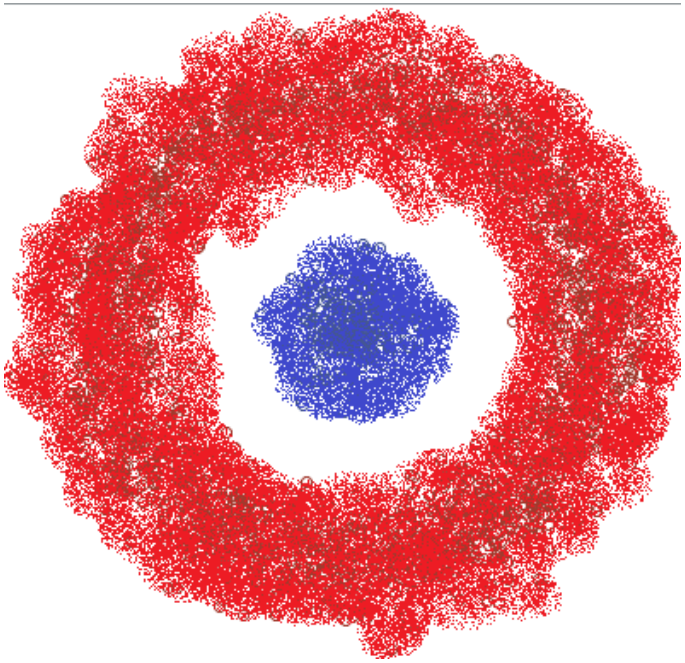


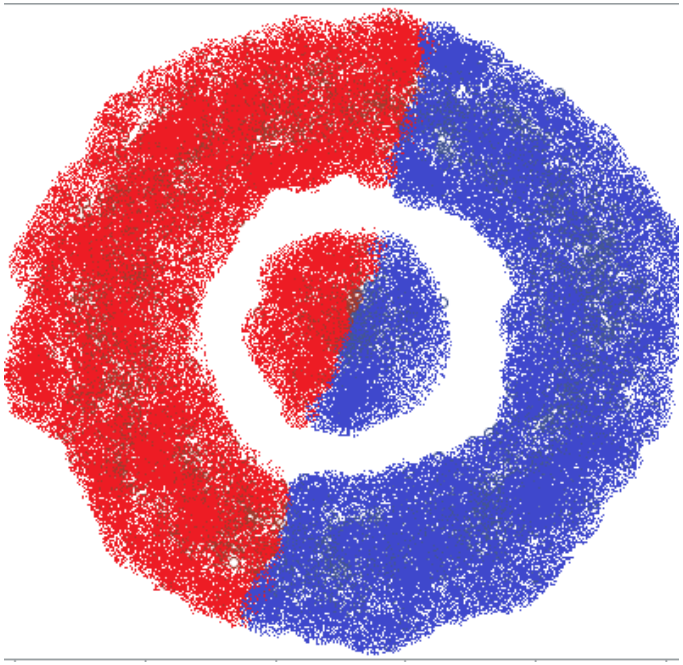
Применение Single Linkage

Суть двух-шагового объединения

- На первом шаге строятся изолированные кластеры по оценке плотности распределения.
- На втором шаге проверяется, что кластер имеет размер больше или равный “***n***”. Если кластер – кандидат на «объединение», то используется *single linkage* для склейки.

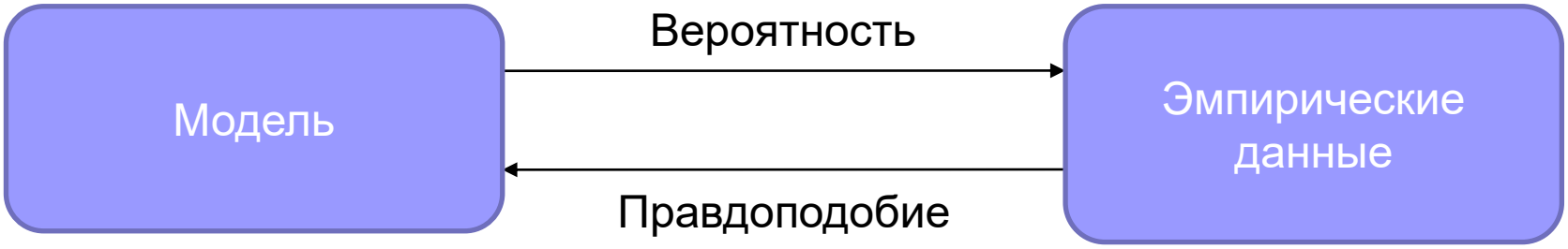
Достоинства непараметрической кластеризации





- Хорошие результаты для компактных кластеров.
- Находит кластеры сложной формы.
- Менее чувствительна к выбросам, шкалам и зависимостям в пространстве признаков.
- Не требует указывать число кластеров заранее.

Общая идея параметрического подхода в кластеризации (и не только)



- Модель описывает процесс порождения эмпирических данных в терминах теории вероятности (например, плотность распределения для непрерывных данных)
- Наиболее общий подход оценки параметров модели через функцию правдоподобия

Функция правдоподобия

Формула Байеса

$$P(Model \mid Data) = \frac{P(Data \mid Model)P(Model)}{P(Data)}$$

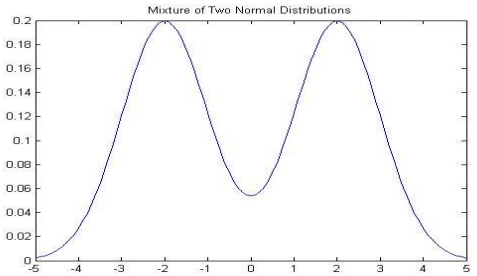
Функция правдоподобия – совместное распределение выборки из параметрического распределения как функция параметра модели.

- Задача – найти «лучшую» модель, ту, которая наиболее точно описывает эмпирические данные.
- Функция правдоподобия – функция модели (параметров модели) при фиксированных данных.
- Спецификация модели – «искусство» (не процедура), фиксируется тип модели (например, распределение), а ищутся «лучшие» параметры.
- Для сложных задач модель = «смесь» (линейная комбинация) распределений.

Примеры

- Испытания Бернулли:
  - Пусть дана последовательность {0,1}: 0,1,1,0,0,1,1,0
  - Ищем модель в классе распределений B(p): $P(X = x) = p^x(1-p)^{1-x}$
  - Задача - найти наилучший параметр p: $\text{argmax}_p P(Data|B(p))$
- Смесь нормальных распределений:
  - Пусть дана выборка, например, рост людей:185,140,134,150,170 …
  - Предполагаем нормальное распределение, но в среднем женщины ниже мужчин, а значит – «смесь» распределений:

$$\pi_1 N(\mu_1, \sigma_1) + \pi_2 N(\mu_2, \sigma_2)$$



  - где: $N(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2\sigma}(x-\mu)^2}$
  - Задача - найти параметры смеси:

$$\underset{\pi_1,\pi_2,\mu_1,\mu_2,\sigma_1,\sigma_2}{\arg\max} \; P(Data \mid \pi_1, \pi_2, \mu_1, \mu_2, \sigma_1, \sigma_2)$$

Оценка максимального правдоподобия

- Точечная оценка параметров, которая максимизирует функцию правдоподобия при фиксированной реализации выборки.
- Если $A_1,\ldots,A_n$ независимые одинаково распределенные сл. вел.:

$$P(A_1,\ldots,A_n) = \prod_{i=1}^{n} P(A_i)$$

- l(Data|Model) = log(P(Data|Model)) - логарифмическая функция правдоподобия:
  - log – монотонная функция, точки экстремумов совпадают
  - вместо произведения – сумма логарифмов
- Максимум функции правдоподобия в нуле производных:
  - Для всех параметров тета:

$$\frac{\partial l(X \mid \theta_1,\ldots,\theta_n)}{\partial \theta_i} = 0$$

  - Для нахождения максимума решаем полученную систему уравнений

Пример с распределением Бернулли

$$
\begin{aligned}
L(p) &= P(0,1,1,0,0,1,0,1|p) \\
&= P(0|p)P(1|p)\ldots P(1|p) \\
&= (1-p)p\ldots p \\
&= p^4(1-p)^4
\end{aligned}
$$

- Надо найти p, максимизирующий логарифмическое правдоподобие *l(p)= Log P(Data|B(p))*

$$
\begin{aligned}
\ell(p) &= logL(p) = 4log(p) + 4log(1-p) \\
\frac{d\ell(p)}{dp} &= \frac{4}{p} - \frac{4}{1-p} \equiv 0 \\
\rightarrow p &= \frac{1}{2}
\end{aligned}
$$

Скрытые (латентные) переменные и EM алгоритм

- Задача кластеризации как оценка скрытых (латентных) переменных – «меток» кластеров:
  - Пусть Data = {x(1),x(2),…x(n)} - набор сл. векторов «наблюдений»
  - Пусть H = {z(1),z(2),..z(n)} - множество соответствующих значений скрытой величины Z, где z(i) соответствует x(i)
  - Считаем, что Z – дискретна.
- Оценка скрытых (ненаблюдаемых) величин – цель EM
- Приложения EM:
  - кластеризация
  - заполнение пропущенных значений в выборке
  - поиск скрытых состояний марковской модели

EM Алгоритм

- Логарифмическое правдоподобие наблюдаемых данных:

$$l(\theta) = \log p(D \mid \theta) = \log \sum_H p(D, H \mid \theta)$$

- Нужно оценить не только параметры θ , но и H
- Пусть Q(H) есть распределение вероятности скрытых величин
- Неравенство Дженсона (для выпуклой функции):

$$\varphi\left(\frac{\sum a_i x_i}{\sum a_i}\right) \leq \frac{\sum a_i \varphi(x_i)}{\sum a_i};$$

$$\begin{aligned}
\ell(\theta) &= log \sum_H p(D, H|\theta) \\
&= log \sum_H Q(H) \frac{P(D, H|\theta)}{Q(H)} \\
&\geq \sum_H Q(H) log \frac{P(D, H|\theta)}{Q(H)} \\
&= \sum_H Q(H) log p(D, H|\theta) + \sum_H Q(H) log \frac{1}{Q(H)} \\
&= F(Q, \theta)
\end{aligned}$$

- F(Q,θ) - нижняя граница *l(θ)*

EM Алгоритм

- Процедура EM (в цикле):
  - максимизация F по Q (тета зафиксированы)
  - максимизация F по тета (Q фиксировано)

E-step $Q^{k+1} = argmax_Q F(Q^k, \theta^k)$

M-step $\theta^{k+1} = argmax_\theta F(Q^{k+1}, \theta^k)$

- E-step: $Q^{k+1} = P(H|D, \theta^k)$
- M-step: $\theta^{k+1} = argmax_\theta \sum_H p(H|D, \theta^k) logp(D, H|\theta^k)$

EM алгоритм для смеси нормальных распределений

Смесь нормальных распределений

$$f(x) = \sum_{k=1}^{K} \pi_k f_k(x, \mu_k, \sigma_k)$$

E Step

$$P(k|x) = \frac{\pi_k f_k(x, \mu_k, \sigma_k)}{f(x)}$$

M-Step

$$\pi_k = \frac{1}{n} \sum_{i=1}^{n} P(k|x(i))$$

$$\mu_k = \frac{1}{n\pi_k} \sum_{i=1}^{n} P(k|x(i))x(i)$$

$$\sigma_k = \frac{1}{n\pi_k} \sum_{i=1}^{n} P(k|x(i))(x(i) - \mu_k)^2$$

- Похоже по сути на k-means, но:
  - Оценка не только мат. ожидания, но и дисперсии (размер кластера)
  - «перекрывающиеся» кластеры, лучше с выбросами

Пример использования

```
from sklearn.mixture import GaussianMixture

fig, axes = plt.subplots(ncols=2, figsize=(10,4))
gmm_bl = GaussianMixture(n_components=3)
axes[0].scatter(X_blob[:, 0], X_blob[:, 1],
                c=gmm_bl.fit_predict(X_blob))

km_cl = GaussianMixture(n_components=2)
axes[1].scatter(X_cl[:, 0], X_cl[:, 1],
                c=km_cl.fit_predict(X_cl))
```