



# Лекция 3: Выявление скрытых структур в данных

# Статистическое обучение без учителя

- Иногда называют задачей «самоорганизации»
- Все признаки равнозначны, нет отклика  $X = (x_1, \dots, x_n)$ , поэтому:
  - Обучение без учителя более «субъективное» (нет единых интуитивно понятных мер качества типа «точности») и менее «автоматизируемо»
  - Сложнее подбирать метапараметры и сравнивать полученные модели
- Важность этого направления в Data mining велико:
  - «Истинный data mining» - ищет неизвестные заранее зависимости без «подсказок» эксперта
  - Много важных прикладных задач по сегментации, выявление скрытых характеристик, зависимостей между атрибутами и т.д.
  - Для больших данных получить качественно размеченный набор тяжело или невозможно, часто возникают задачи *semisupervised learning*, когда размечена лишь часть набора

# Основные задачи обучения без учителя

- Большинство задач сводится к поиску скрытых (латентных) признаков или скрытых структур (групп или зависимостей) в данных:

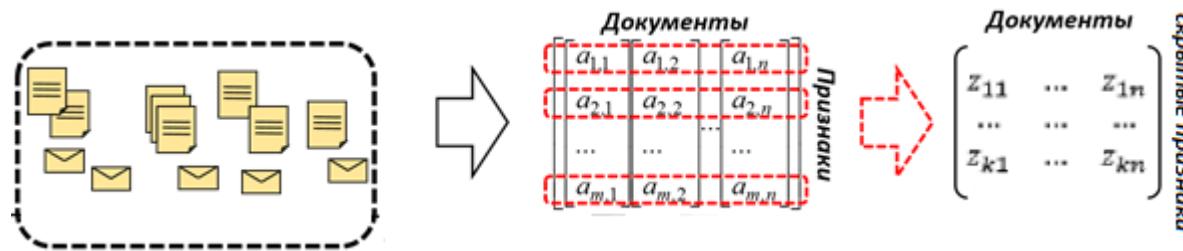
$$z_1 = F_1(x_1, \dots, x_n), \dots, z_k = F_k(x_1, \dots, x_n)$$

- Основные случаи:  $z$  или кластеры, или признаки, иногда как в SOM (сетях Кохонена) по сути и то, и другое.
- Основные прикладные задачи если  $z$  - признаки:
  - Предобработка данных (как правило перед применением регрессионных моделей или методов кластеризации) = сокращение пространства признаков  $k << n$ , удаление корреляций (или других зависимостей)
  - Тематической моделирование текстовых данных (переход из пространства термов в пространство тематик)
  - Свертка «транзакционной истории» в вектор признаков, особенно часто используется в рекомендательных системах.

# Тематическое моделирование текстов

## ■ Исходная задача:

- Каждый документ  $(a_1, \dots, a_m)$  – вектор признаков в пространстве словаря, например созданный по схеме «мешок слов»,  $a_i$  вес  $i$ -го слова в документе
- Корпус (множество документов) представим в виде матрицы



## ■ Тематическая модель:

- $z_1 = F_1(a_1, \dots, a_m), \dots, z_k = F_k(a_1, \dots, a_m)$ , где  $k < n$  - скрытые признаки документа или веса тематик

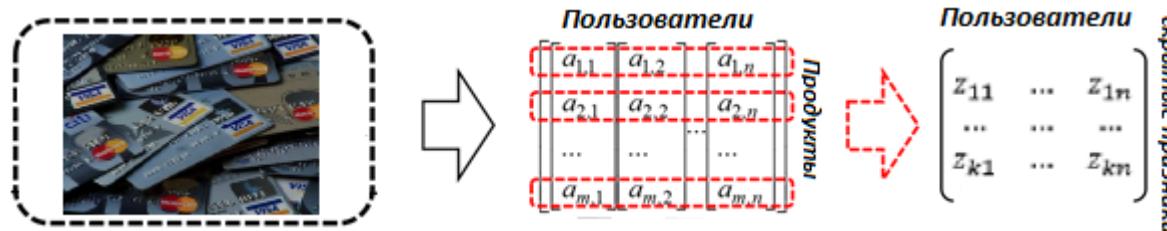
# Рекомендательные системы

- Дано:
  - Множество пользователей (клиентов)  $J$
  - Множество продуктов (услуг, ресурсов и т.д.)  $I$
  - Множество транзакций  $D$  <время, пользователь, услуга>
- Задача:
  - Построить модель, которая по истории пользователя будет способна предсказать какие продукту (услуги или ресурсы) его заинтересуют.
- Решение на основе ассоциативных правил:
  - реализовано в узле Link Analysis
  - подставить события из истории пользователя во все правила и найти наиболее достоверное следующее событие.
- Альтернатива – на основе тематического моделирования

# Рекомендательные системы

## ■ Исходная задача:

- Транзакционная история представляется в виде матрицы, где  $a_{ij}$ , например, равно числу транзакций пользователя  $j$  по использованию продукта  $i$



## ■ Тематическая модель:

- $z_1 = F_1(a_1, \dots, a_m), \dots, z_k = F_k(a_1, \dots, a_m)$ , где  $k < n$  - скрытые предпочтения пользователя или веса тематик

# Общая идея РСА

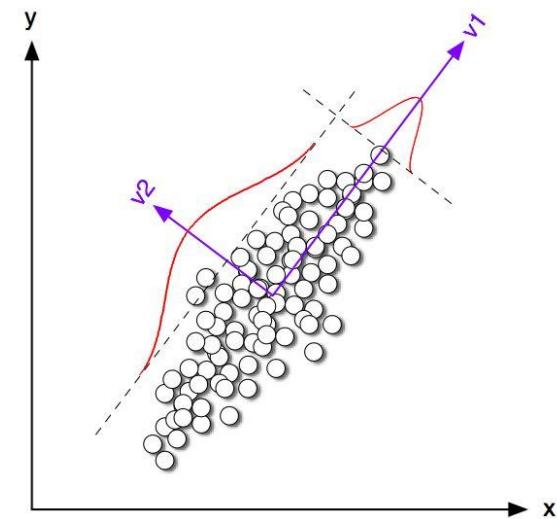
- Строится новый базис (линейное преобразование исходного пространства) такой, что:
  - Центр координат совпадает с мат. ожиданием наблюдений
  - Первый вектор направлен таким образом, что дисперсия вдоль него была максимальной
  - Каждый последующий вектор ортогонален предыдущим и направлен по направлению максимальной дисперсии
  - Последние компоненты – не важны!!!

- Формально:

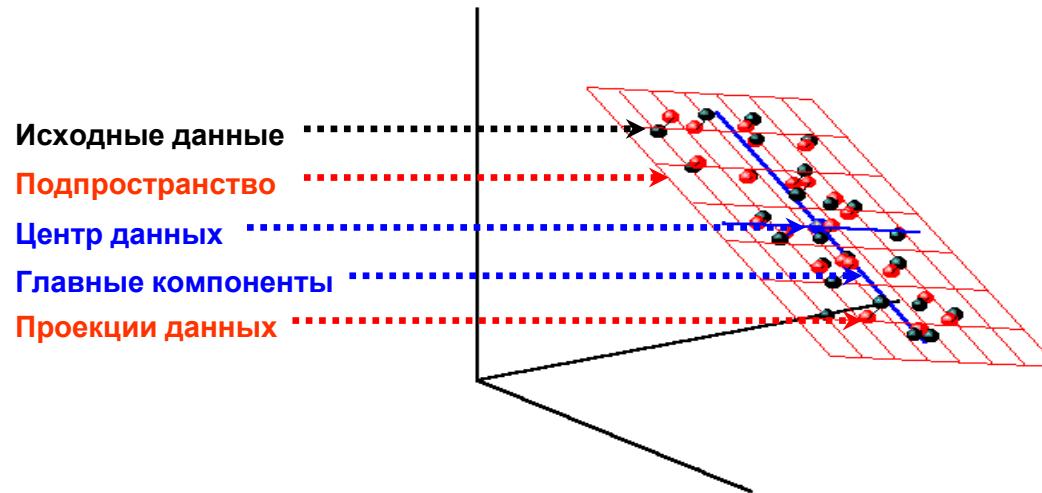
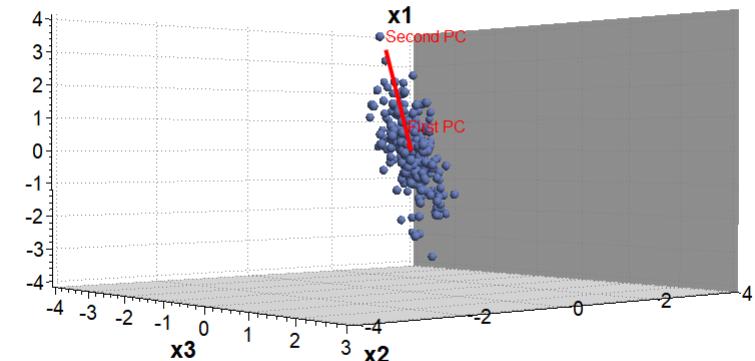
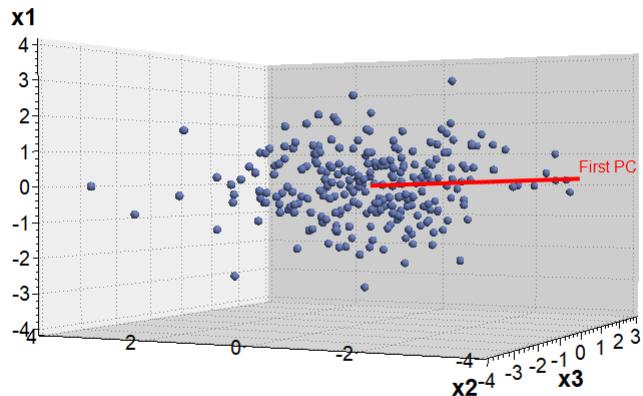
$$\mathbf{w}_1 = \arg \max_{\|\mathbf{w}\|=1} E\{(\mathbf{w}^T \mathbf{x})^2\}$$

$$\mathbf{w}_k = \arg \max_{\|\mathbf{w}\|=1} E\{[\mathbf{w}^T (\mathbf{x} - \sum_{i=1}^{k-1} \mathbf{w}_i \mathbf{w}_i^T \mathbf{x})]^2\}$$

- SVD разложение матрицы данных
- Собственные значения ковариационной матрицы



# Геометрическая интерпретация





# Поиск собственных значений и собственных векторов ковариационной

- Рассчитаем (или корреляционную) ковариационную матрицу:

- Ковариация = 0 – независимы
- Ковариация > 0 – вместе растут и убывают
- Ковариация < 0 – противофаза

$$C = \begin{bmatrix} cov(x_1, x_1) & \cdots & cov(x_1, x_m) \\ \vdots & \ddots & \vdots \\ cov(x_m, x_1) & \cdots & cov(x_m, x_m) \end{bmatrix}$$

- Проблема с.зн.:

$$\mathbf{C} * \mathbf{v} = \lambda * \mathbf{v}$$

решение: поиск корней

$$cov(x_i, x_j) = \mathbf{E}[(x_i - \mathbf{E}(x_i))(x_j - \mathbf{E}(x_j))]$$

$$|\mathbf{C} - \lambda \cdot \mathbf{I}| = 0$$

матрица положительно определенная – есть вещественные корни

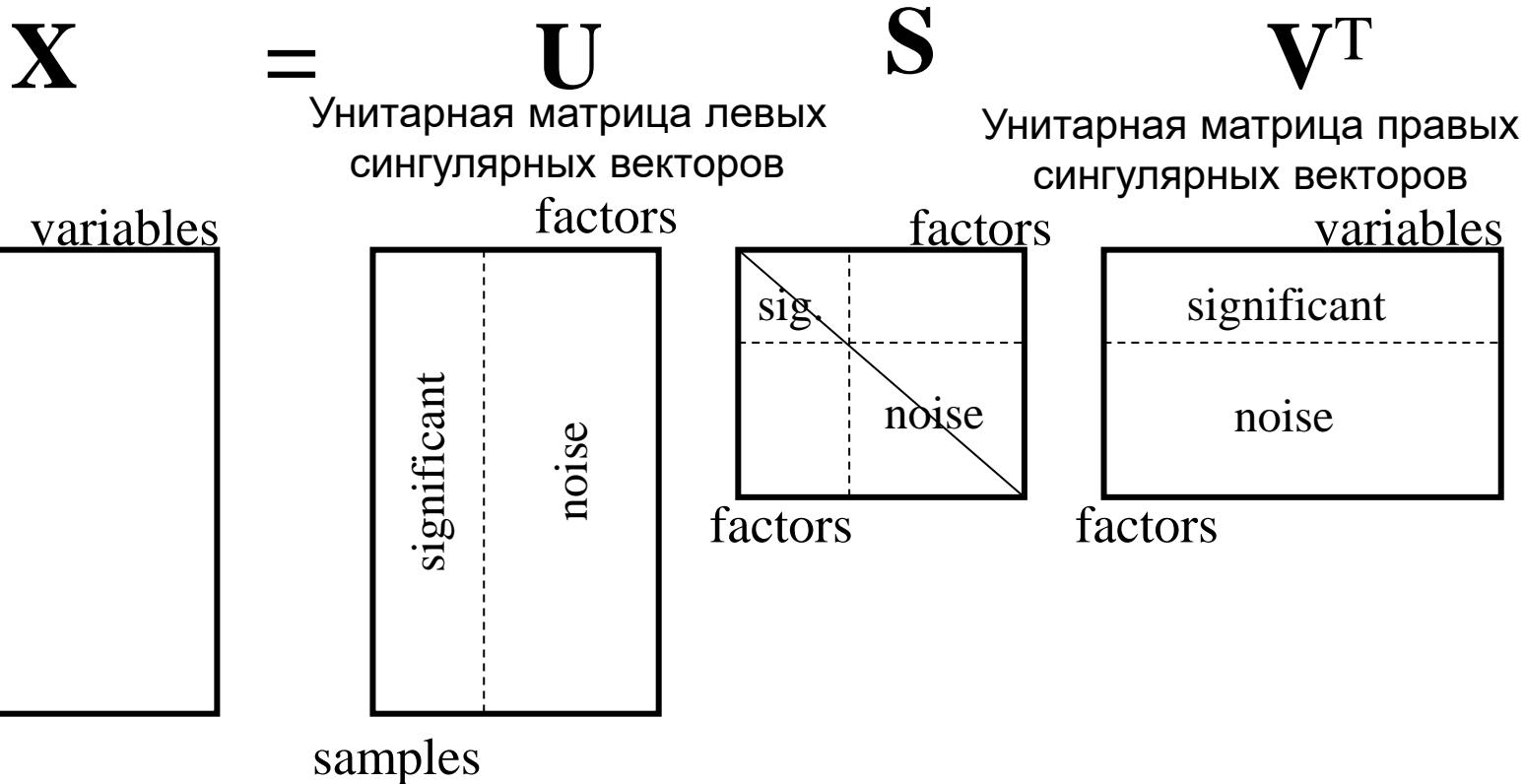
- Результат:

- $\lambda$  – дисперсии, с.в. – главные компоненты

$$z_1 = \alpha_{11}x_1 + \cdots + \alpha_{1m}x_m, \dots, z_k = \alpha_{k1}x_1 + \cdots + \alpha_{km}x_m,$$

$$\forall i \sum_{j=1}^m \alpha_{ij}^2 = 1$$

# Сингулярное разложение в PCA



## ■ Если тексты:

- X – матрица корпуса документов, samples – документы, variables – термы, factors – тематики, U – представление документов в пространстве тематик, V – представление тематик в пространстве термов, S – веса тематик

## ■ Если рекомендации:

- X – матрица частот покупок, samples – клиенты, variables – продукты, factors – предпочтения, U – представление клиентов в пространстве предпочтений, V – представление продуктов в пространстве предпочтений, S – веса предпочтений

# SVD разложение и обратная проекция

- SVD разложение матрицы X:  $X_{n \times m} = U_{n \times n} D_{n \times m} V_{m \times m}^T$
- SVD приближение (метод главных компонент):
  - отбрасываются с.в., соотв. наименьшим с.з.
  - остается p-я часть главных с.в., которые характеризуют основные зависимости в X
  - с их помощью приближается исходная матрица:

$$X^{(l+1)} = V_p V_p^T X^{(l)}$$

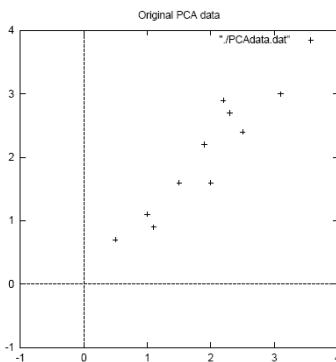


Figure 3.1: PCA example data, original data on the left, data with the means subtracted on the right, and a plot of the data

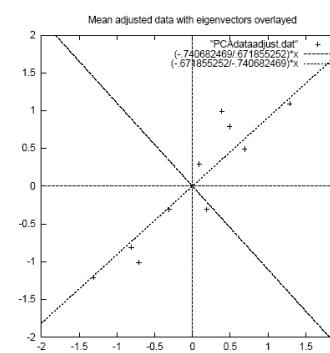


Figure 3.2: A plot of the normalised data (mean subtracted) with the eigenvectors of the covariance matrix overlaid on top.

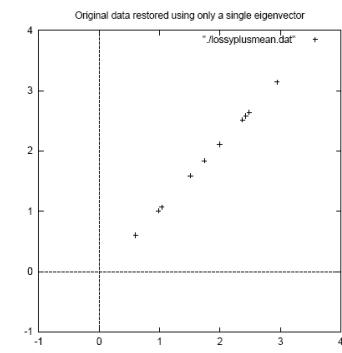
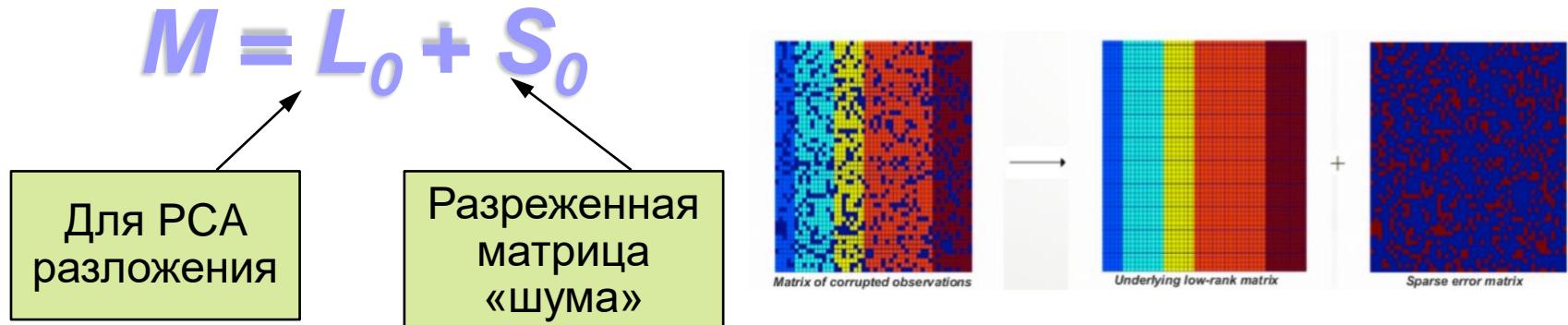


Figure 3.5: The reconstruction from the data that was derived using only a single eigenvector

# Робастный метод главных компонент

- RPCA раскладывает исходную матрицу как сумму двух:



- Алгоритм Alternating Optimization (блочный коорд. спуск):
  1. Находим  $L$ , раскладывая  $M$  с помощью SVD
  2. По заданному **порогу** определяем  $S$
  3. Находим новую  $M$ , вычитая  $S$  из старой  $M$
  4. Переходим на первый шаг
- Регуляризованная задача условной оптимизации:

$$\min \|L_0\| + \lambda \|S_0\|, M = L_0 + S_0,$$

$\lambda$  - **параметр** регуляризации

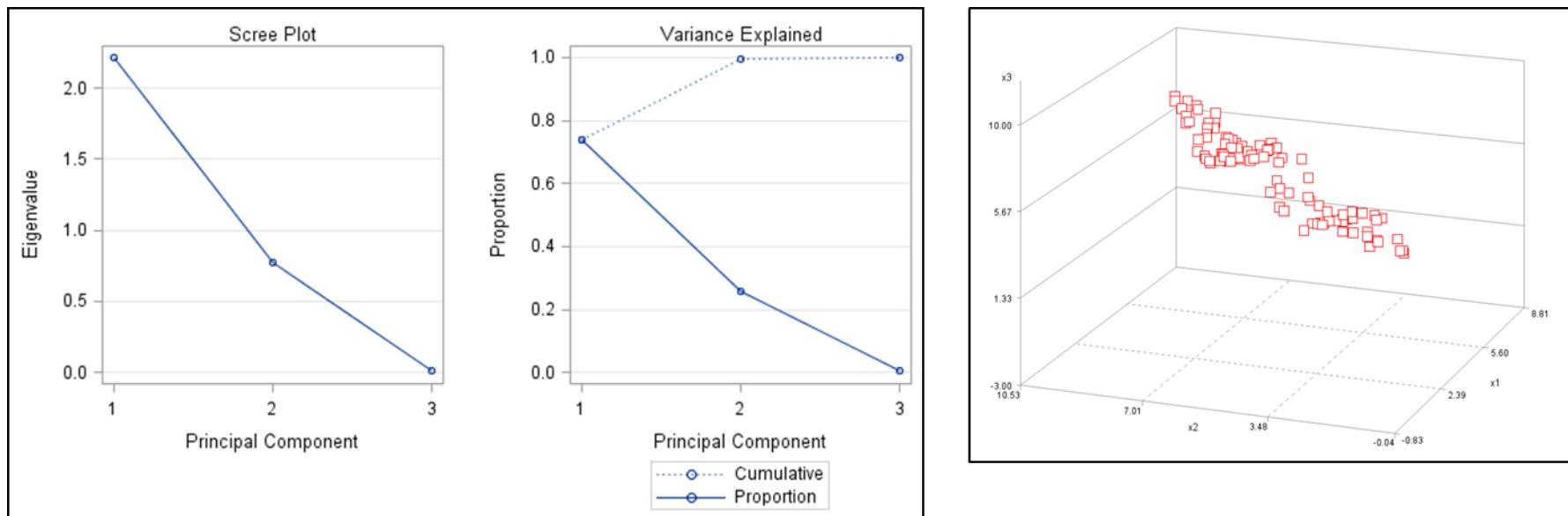
# Отбор числа главных компонент

- Пропорция описанной вариации пространства признаков
- По графику
- С.зн > 1 (для корреляционной матрицы)
- Теоретические методы:
  - Кайзера (остаются с.в. у которых с.зн. больше среднего с.зн)
  - Правило «сломанной стрости» - Набор нормированных на единичную сумму собственных чисел сравнивается с распределением длин обломков трости единичной длины, сломанной в n-1-й случайно выбранной точке (точки разлома выбираются независимо и равнораспределены по длине трости). Выбираются те компоненты, для которых:

$$\frac{\sum_{i=1}^n z_{im}^2}{\sum_{j=1}^p \sum_{i=1}^n x_{ij}^2}$$

$$\lambda_i > E(L_i) = \frac{1}{n} \sum_{j=i}^n \frac{1}{j}$$

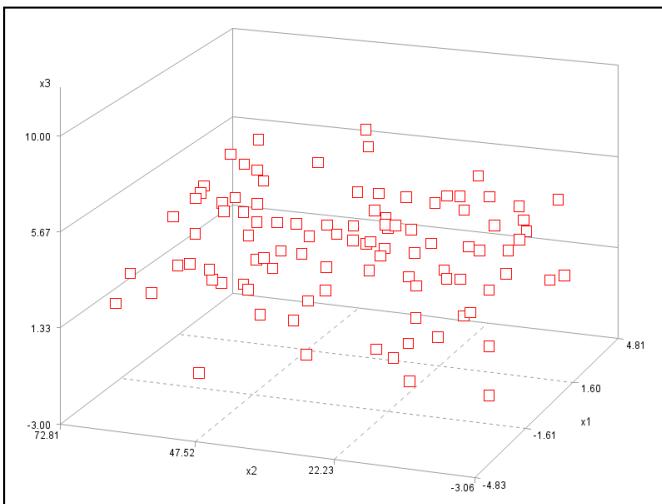
# Пример 1



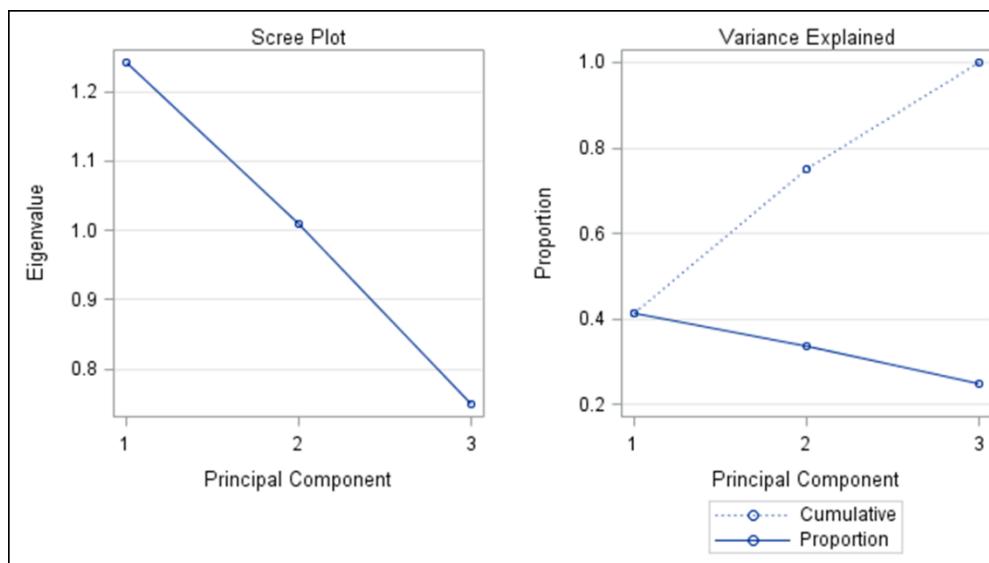
Eigenvalues of the Correlation Matrix				
	Eigenvalue	Difference	Proportion	Cumulative
<b>1</b>	2.21358154	1.44403082	0.7379	0.7379
<b>2</b>	0.76955072	0.75268297	0.2565	0.9944
<b>3</b>	0.01686775		0.0056	1.0000

Eigenvectors			
	Prin1	Prin2	Prin3
<b>x1</b>	0.650940	0.263685	0.711862
<b>x2</b>	0.645235	0.301851	-.701825
<b>x3</b>	-.399937	0.916164	0.026348

# Пример 2



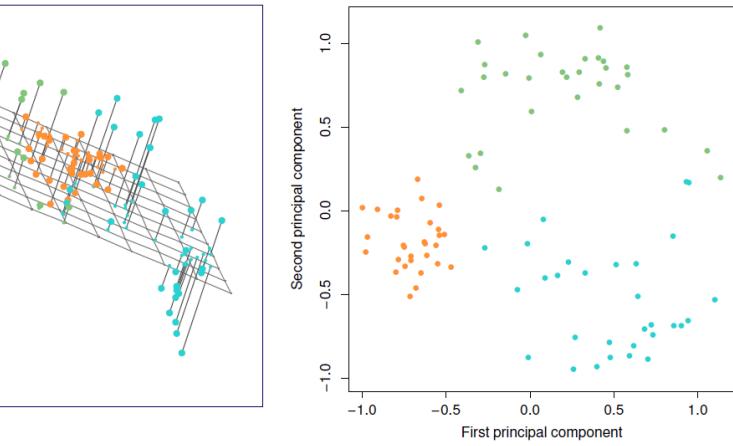
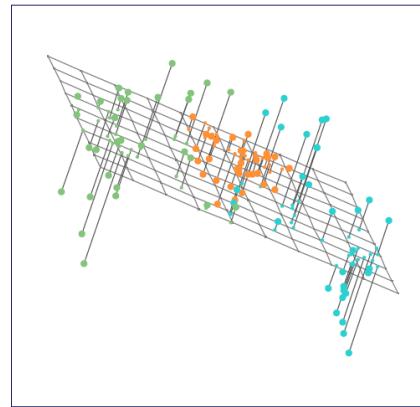
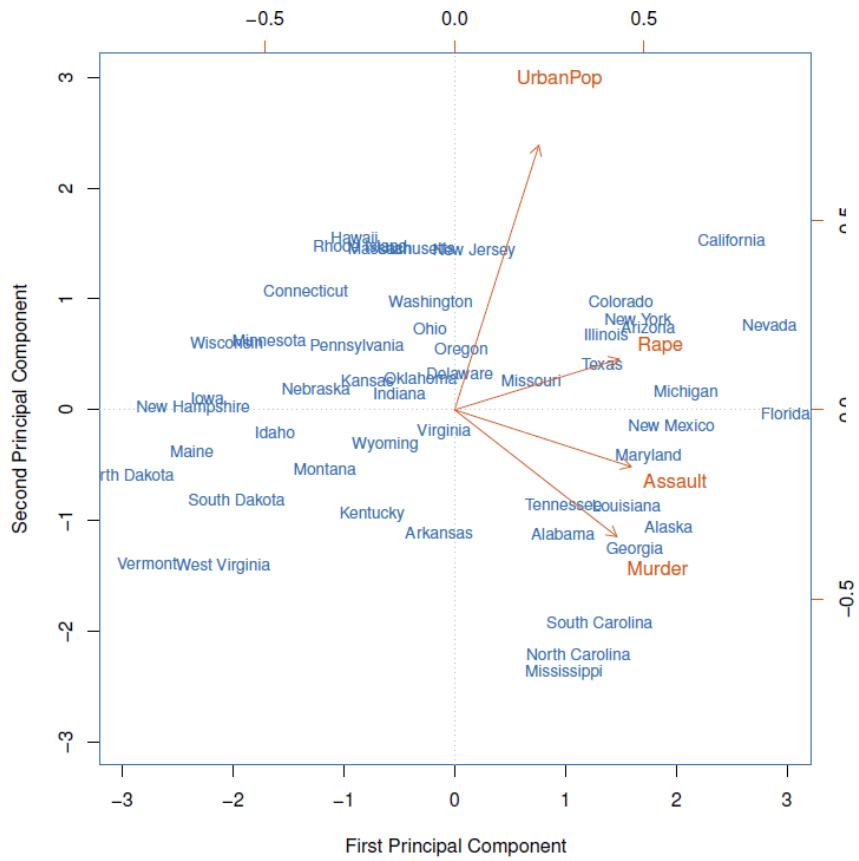
Eigenvalues of the Correlation Matrix				
	Eigenvalue	Difference	Proportion	Cumulative
1	1.24205697	0.23274599	0.4140	0.4140
2	1.00931098	0.26067894	0.3364	0.7505
3	0.74863205		0.2495	1.0000



Eigenvectors			
	Prin1	Prin2	Prin3
x1	0.704619	-0.156546	0.692102
x2	0.708942	0.113759	-0.696032
x3	0.030228	0.981097	0.191139

# Визуализация

- Одно из важнейших применений – проекция множества наблюдений на пары главных компонент



	PC1	PC2
Murder	0.5358995	-0.4181809
Assault	0.5831836	-0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

# Пример использования (Python)

## Пример:

- Преобразуем транзакционный набор ASSOC в матрицу клиент-продукт:

```
import pandas as pd
```

```
assoc = pd.read_csv("assoc.csv")
```

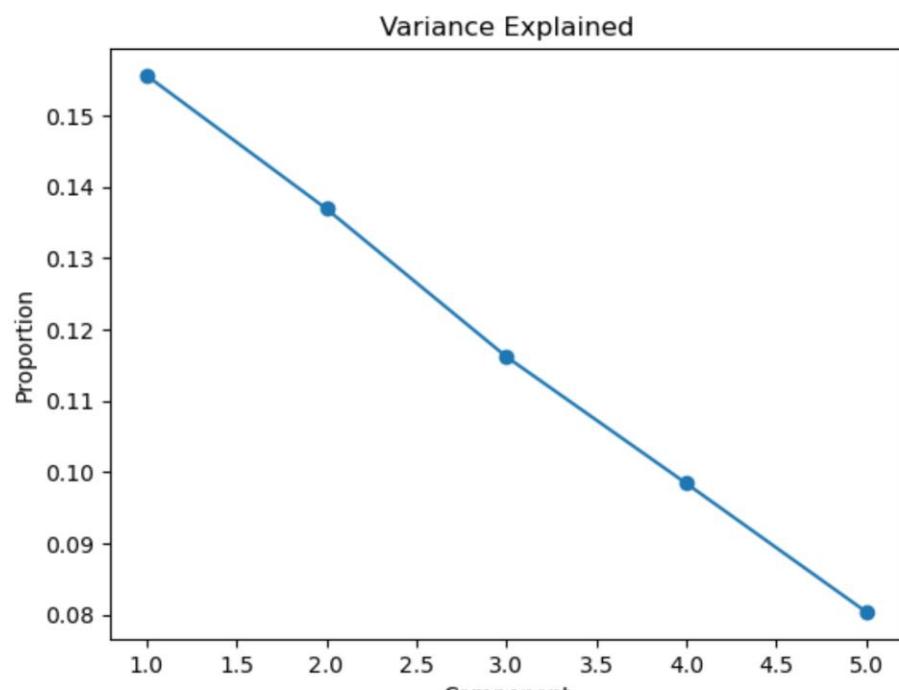
```
assoc = assoc.groupby("CUSTOMER")["PRODUCT"].value_counts().unstack(fill_value=0)
assoc
```

# Пример использования (Python)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

```
N = 5
pca = PCA(n_components=N)
features = pca.fit_transform(assoc)
```

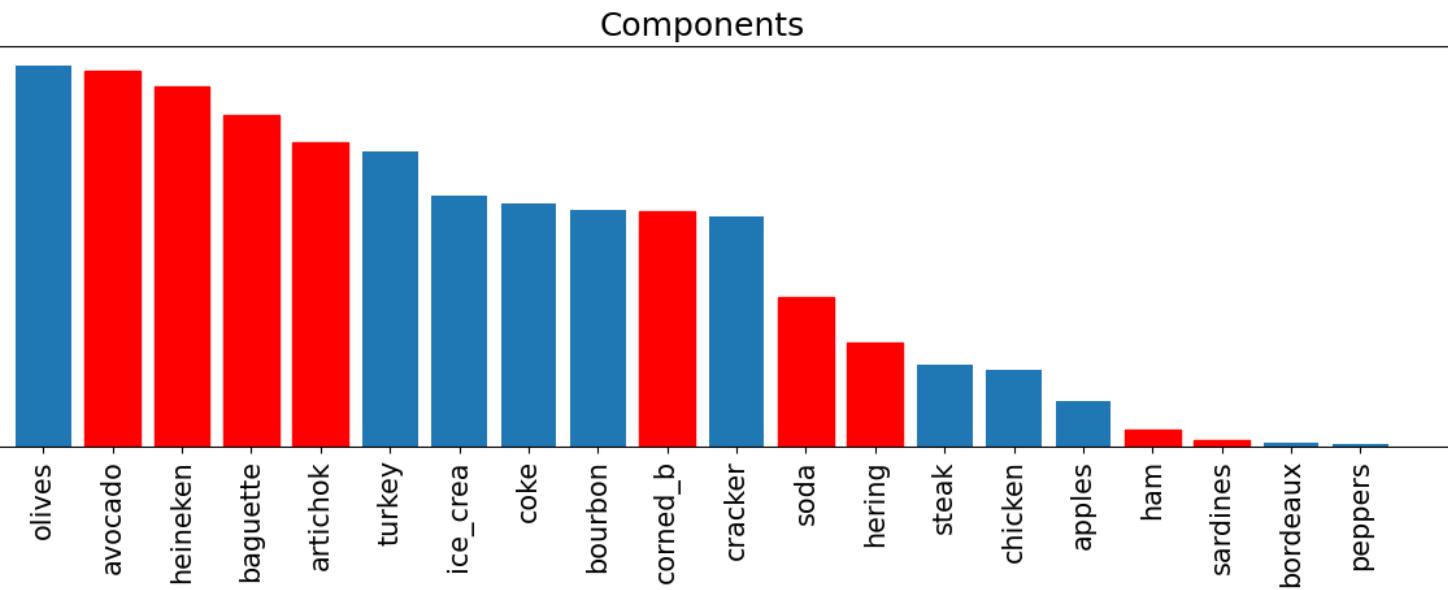
```
plt.plot(range(1, N+1), pca.explained_variance_ratio_)
plt.scatter(range(1, N+1), pca.explained_variance_ratio_)
plt.xlabel("Component")
plt.ylabel("Proportion")
plt.title("Variance Explained")
pass
```



# Пример использования (Python)

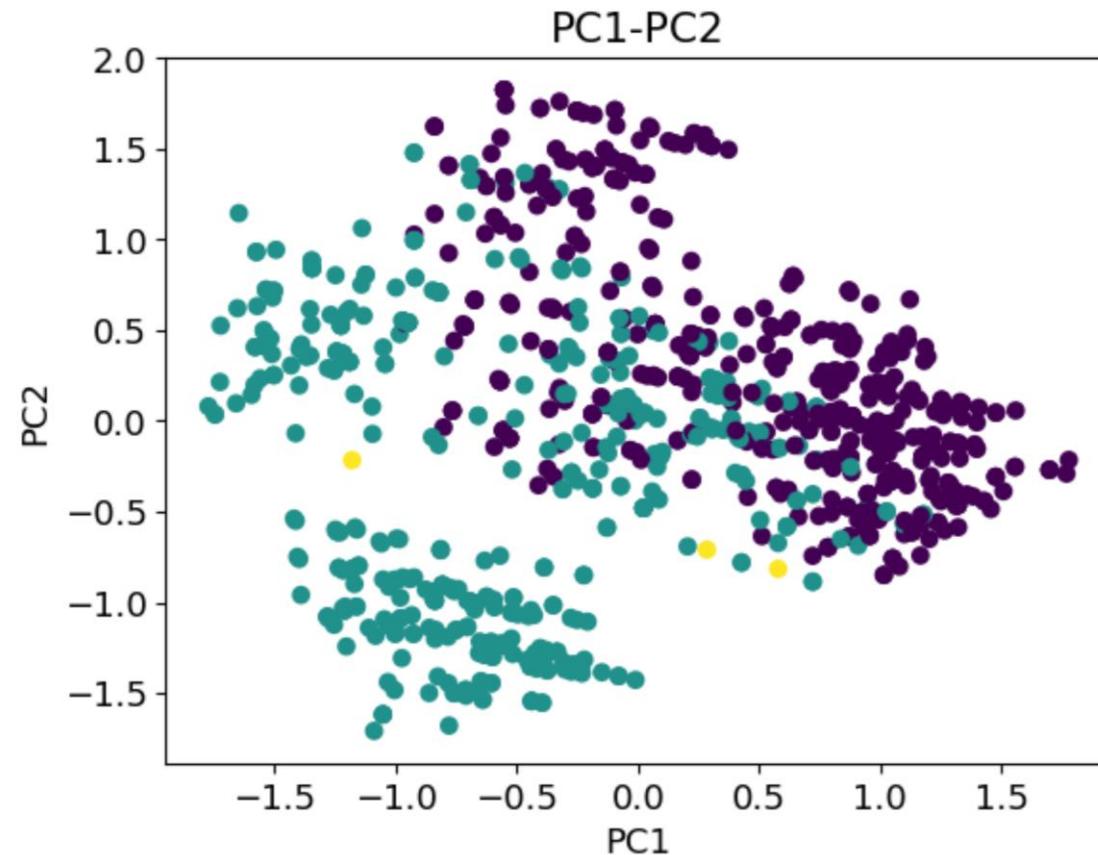
```
components = pca.components_ # [Число компонент, Число переменных]
sample = components[0] # Для примера возьмем первую
order = np.argsort(-abs(sample))
```

```
plt.xticks(rotation='vertical') # Поворачиваем текст на графике
# Нарисуем по модулю:
barplot = plt.bar(assoc.columns[order], abs(sample[order]))
# Отрицательные покрасим синими:
[x.set_color("red") for i, x in enumerate(barplot) if sample[i] > 0]
plt.title("Components")
pass
```



# Пример использования (Python)

```
# Расскраска - объем покупки оливок
plt.scatter(features[:, 0], features[:, 1], c=assoc["olives"])
plt.title("PC1-PC2")
plt.xlabel("PC1")
plt.ylabel("PC2")
pass
```



# Пример использования (Python)

## ■ Результаты:

Новые признаки клиентов - *features*  
(матрица U):

```
pd.DataFrame(data=features, index=assoc.index,  
             columns=[f"PC{i}" for i in range(1, N+1)])
```

CUSTOMER	PC1	PC2	PC3	PC4	PC5
0.0	-1.405232	-0.748403	-0.232557	-0.591675	0.319550
1.0	0.428760	-0.779644	-0.853852	0.144034	-0.758485
2.0	0.827363	0.242493	0.067664	-0.848863	1.020968
3.0	-1.516972	0.454502	-0.049412	-0.359889	0.325329
4.0	-0.667293	-1.491540	0.803896	-0.082407	-0.167066
...	...	...	...	...	...

Веса предпочтений  
(синг. числа lambda) =>

```
pca.singular_values_
```

```
array([26.34434604, 24.71066022, 22.76228395, 20.94970349, 18.93517467])
```

Новые признаки продуктов - *components*  
(матрица V):

```
pd.DataFrame(data=components, columns=assoc.columns,  
             index=[f"PC{i}" for i in range(1, N+1)])
```

PRODUCT	apples	artichok	avocado	baguette	bordeaux
PC1	-0.043935	0.298770	0.367965	0.324553	0.003791
PC2	-0.132510	-0.029870	-0.050568	-0.041311	0.001540
PC3	0.377463	0.040309	0.270928	0.188769	0.002230
PC4	0.247279	-0.259004	-0.072430	0.098635	-0.030733
PC5	-0.178475	0.313816	0.179802	-0.379810	0.009148

## ■ Как узнать купит ли клиент и продукт v?

- Посчитать скалярное произведение  $\langle u, v \rangle * \lambda$

# Неотрицательная матричная факторизация



	I1	I2	I3	I4	I5
U1	5	-	4	-	3
U2	2	-	-	4	5
U3	-	4	3	3	-
U4	1	2	1	-	-
U5	1	3	4	-	5
U6	4	3	-	5	4
U7	-	5	4	-	3
U8	3	-	-	4	5
U9	-	3	2	1	1
U10	5	4	-	3	-

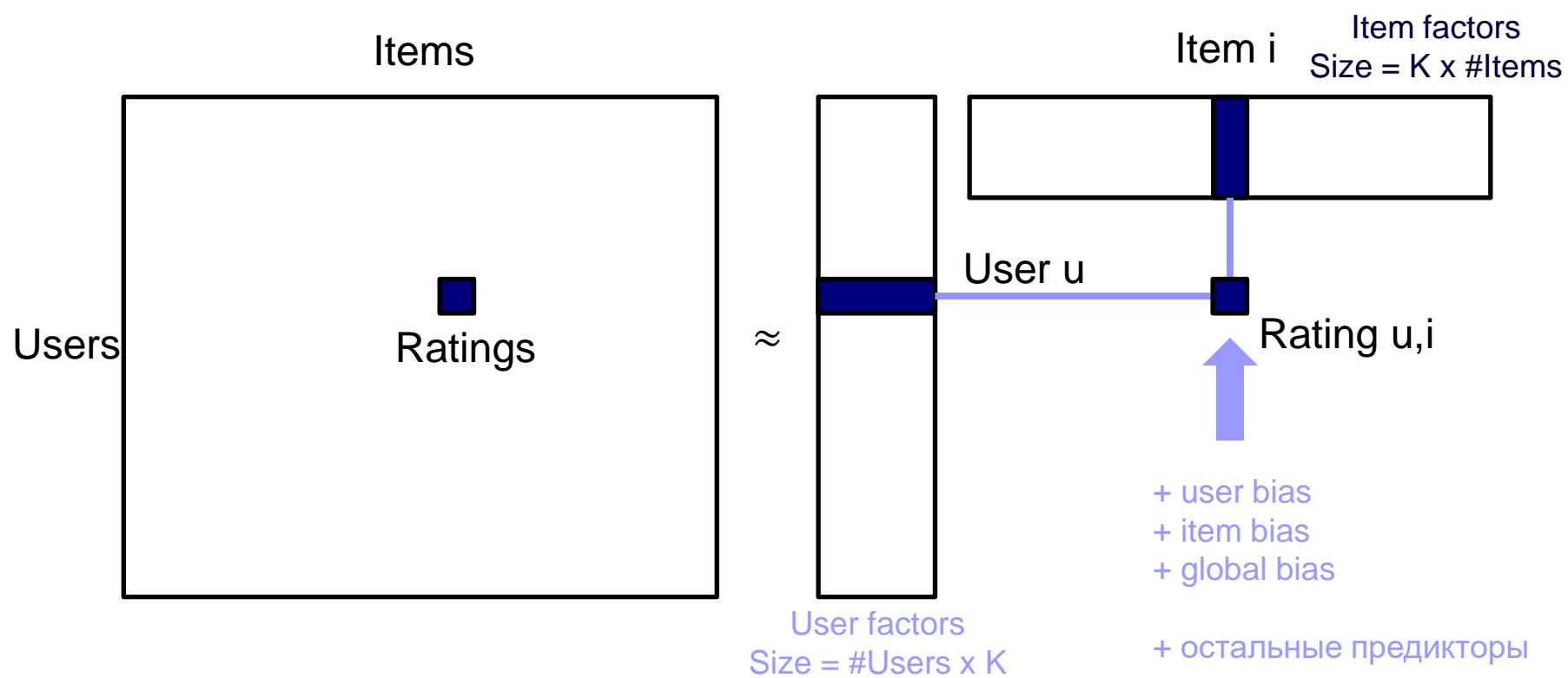


Разреженные данные (<1%)

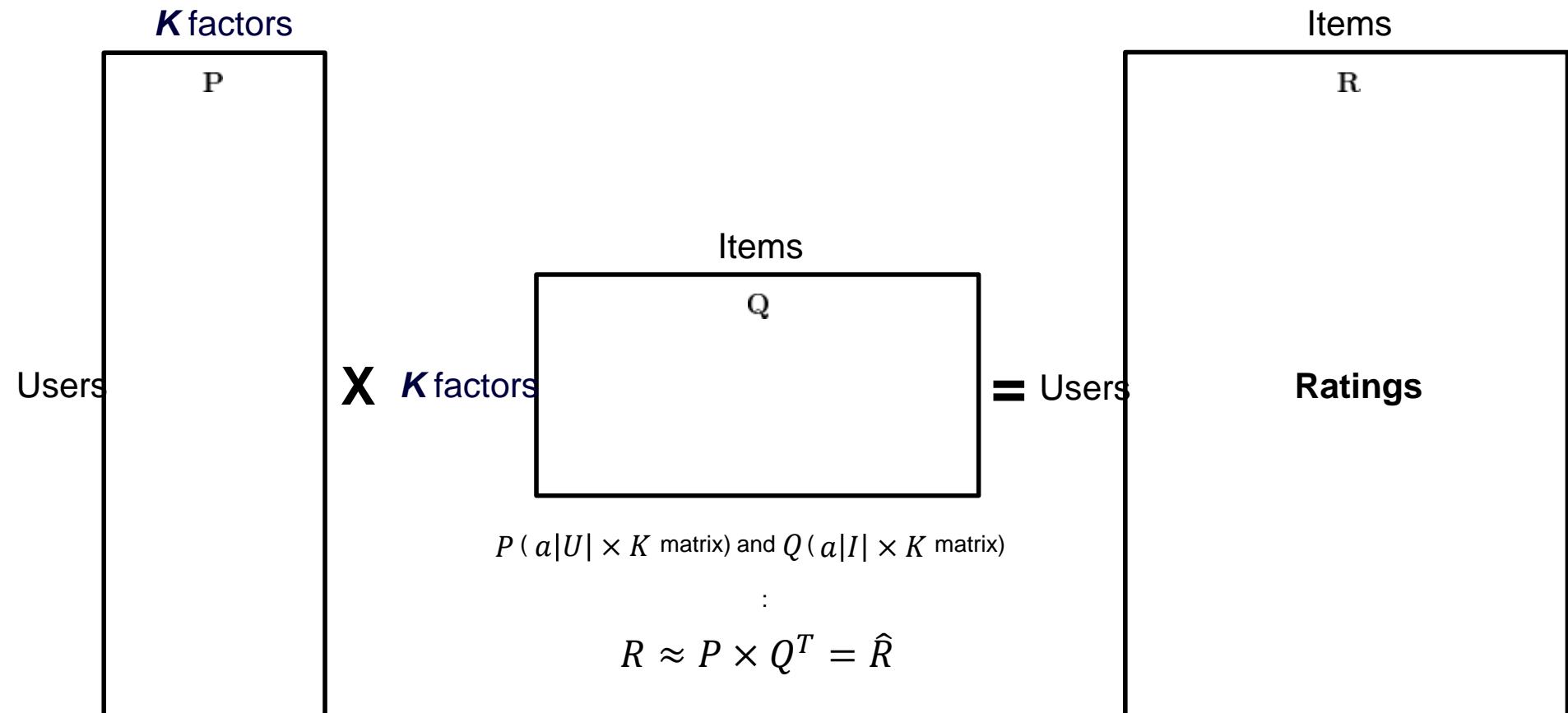
user	item	rating
1	14	5
1	21	3
2	8	1

Предсказать рейтинг  $i$  от  $u$ ?

# Разложение



# Факторизация

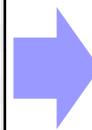


# Пример

user	item	rating
1 U1	I1	5
2 U1	I2	.
3 U1	I3	4
4 U1	I4	.
5 U1	I5	3
6 U2	I1	2
7 U2	I2	.
8 U2	I3	.
9 U2	I4	4
10 U2	I5	5
11 U3	I1	.
12 U3	I2	4
13 U3	I3	3
14 U3	I4	3
15 U3	I5	.
16 U4	I1	1
17 U4	I2	2
18 U4	I3	1
19 U4	I4	.
20 U4	I5	.
21 U5	I1	1
22 U5	I2	3
23 U5	I3	4
24 U5	I4	.
25 U5	I5	5
26 U6	I1	4
27 U6	I2	3
28 U6	I3	.
29 U6	I4	5
30 U6	I5	4
31 U7	I1	.
32 U7	I2	5
33 U7	I3	4
34 U7	I4	.
35 U7	I5	3



user	item	rating
1 U1	I1	5
2 U1	I3	4
3 U1	I5	3
4 U2	I1	2
5 U2	I4	4
6 U2	I5	5
7 U3	I2	4
8 U3	I3	3
9 U3	I4	3
10 U4	I1	1
11 U4	I2	2
12 U4	I3	1
13 U5	I1	1
14 U5	I2	3
15 U5	I3	4
16 U5	I5	5
17 U6	I1	4
18 U6	I2	3
19 U6	I4	5
20 U6	I5	4
21 U7	I2	5
22 U7	I3	4
23 U7	I5	3
24 U8	I1	3
25 U8	I4	4
26 U8	I5	5
27 U9	I2	3
28 U9	I3	2
29 U9	I4	1
30 U9	I5	1
31 U10	I1	5
32 U10	I2	4
33 U10	I4	3



user	item	rating	P_rating
1 U1	I1	5	0.7230614009
2 U1	I2	.	1.4543056794
3 U1	I3	4	2.9436770971
4 U1	I4	.	2.7634787906
5 U1	I5	3	2.1839976178
6 U2	I1	2	2.3006243215
7 U2	I2	.	4.7609997821
8 U2	I3	.	1.0879811832
9 U2	I4	4	3.4404221739
10 U2	I5	5	5.3208743099
11 U3	I1	.	4.166851895
12 U3	I2	4	4.88791012
13 U3	I3	3	5.0102871449
14 U3	I4	3	2.2439738265
15 U3	I5	.	4.298220204
16 U4	I1	1	1.9823621161
17 U4	I2	2	2.7822297809
18 U4	I3	1	0.8577964559
19 U4	I4	.	5.6414348019
20 U4	I5	.	3.0406873297
21 U5	I1	1	2.6027651282
22 U5	I2	3	4.1788161881
23 U5	I3	4	2.3633319756
24 U5	I4	.	5.1503110093
25 U5	I5	5	4.6818439748
26 U6	I1	4	4.4644140939
27 U6	I2	3	6.1247374954
28 U6	I3	.	2.3826838929
29 U6	I4	5	2.4068105187
30 U6	I5	4	5.3681943906
31 U7	I1	.	3.4547263077
32 U7	I2	5	3.0205538908
33 U7	I3	4	3.2398652209
34 U7	I4	.	3.7632093179
35 U7	I5	3	1.5005131987

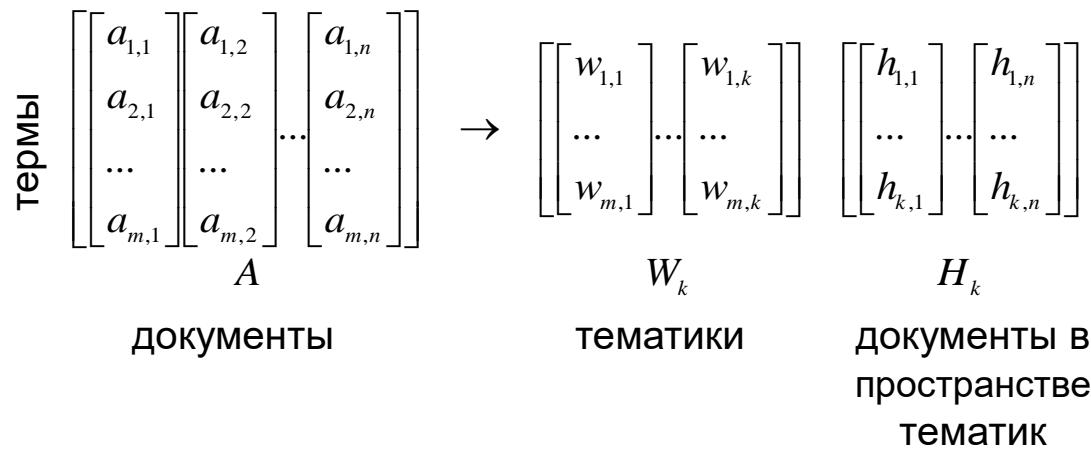
P\_rating

# Неотрицательная матричная факторизация

- $A \in \mathbb{R}^{m \times n}$ , ( $a_{i,j} \geq 0$ )
  - Цель *неотрицательной матричной факторизации* состоит в нахождении матриц  $W_k \in \mathbb{R}^{m \times k}$  и  $H_k \in \mathbb{R}^{k \times n}$  с неотрицательными элементами, которые минимизируют целевую функцию:

$$f(W_k, H_k) = \frac{1}{2} \|A - W_k H_k\|_F^2, k << \min(m, n)$$

- Часто добавляют L1 или L2 регуляризацию



# Мультипликативный алгоритм (НМФ)

1.  $W^1 \in \mathbb{R}^{m \times k}$ ,  $H^1 \in \mathbb{R}^{k \times n}$  инициализируются случайными неотрицательными числами
2. В цикле  $p$  раз выполняются итерационные формулы для вычисления матриц  $W$  и  $H$ :

$$H_{b,j}^{p+1} = H_{b,j}^p \frac{((W^p)^T A)_{b,j}}{((W^p)^T W^p H^p)_{b,j}}, \forall b, j : 1 \leq b \leq k, 1 \leq j \leq n$$

$$W_{i,a}^{p+1} = W_{i,a}^p \frac{(A(H^{p+1})^T)_{i,a}}{(W^p H^{p+1} (H^{p+1})^T)_{i,a}}, \forall i, a : 1 \leq i \leq m, 1 \leq a \leq k$$

# Ортонормированная неотрицательная матричная факторизация (ОНМФ)

Целевая функция:  $f(W_k, H_k) = \frac{1}{2} \|A - W_k H_k\|_F^2 + \frac{\alpha}{2} \|W_k^T W_k - I\|_F^2$

1.  $W^1 \in \mathbb{R}^{m \times k}$ ,  $H^1 \in \mathbb{R}^{k \times n}$  инициализируются случайными неотрицательными числами
2. В цикле  $p$  раз выполняются итерационные формулы для вычисления матриц  $W$  и  $H$ :

$$H_{b,j}^{p+1} = H_{b,j}^p \frac{((W^p)^T A)_{b,j}}{((W^p)^T W^p H^p)_{b,j}}, \forall b, j : 1 \leq b \leq k, 1 \leq j \leq n$$

$$W_{i,a}^{p+1} = W_{i,a}^p \frac{(A(H^{p+1})^T + \alpha W^p)_{i,a}}{(W^p H^{p+1} (H^{p+1})^T + \alpha W^p (W^p)^T W^p)_{i,a}}, \forall i, a : 1 \leq i \leq m, 1 \leq a \leq k$$

# Матричное разложение для рекомендательной системы

```
from sklearn.decomposition import NMF  
  
assoc = pd.read_csv("assoc.csv")  
assoc = assoc.groupby("CUSTOMER")["PRODUCT"].value_counts().unstack(fill_value=0)
```

```
model = NMF(n_components=3, init="random")  
W = model.fit_transform(assoc)  
H = model.components_
```

```
pd.DataFrame(data=model.inverse_transform(W), columns=assoc.columns, index=assoc.index)
```

PRODUCT	apples	artichok	avocado	baguette	bordeaux	bourbon	chicken	coke	corned_b	cracker
CUSTOMER										
0.0	0.519338	0.000000	0.000000	0.036794	0.063891	0.560310	0.277108	0.333505	1.000215	0.207534
1.0	0.427146	0.510485	0.599759	0.605546	0.075752	0.284905	0.074443	0.000000	0.608237	0.676996
2.0	0.201094	0.564403	0.663107	0.662720	0.067902	0.237976	0.210140	0.153851	0.081830	0.710042
3.0	0.320717	0.000000	0.000000	0.042381	0.067156	0.687009	0.602264	0.724839	0.509480	0.238749
4.0	0.554170	0.095198	0.111846	0.134998	0.056236	0.364830	0.013883	0.000000	1.090856	0.250928
...	...	...	...	...	...	...	...	...	...	...
996.0	0.176711	0.557283	0.654742	0.658958	0.073017	0.317996	0.330604	0.300082	0.004900	0.726924
997.0	0.553492	0.128531	0.151009	0.173210	0.058647	0.364667	0.018743	0.000000	1.071724	0.287628
998.0	0.185199	0.594551	0.698528	0.701715	0.075977	0.317037	0.327377	0.289657	0.004729	0.768163
999.0	0.148969	0.039002	0.045823	0.088607	0.068147	0.738745	0.815592	0.974739	0.082840	0.290031
1000.0	0.527035	0.135145	0.158779	0.184707	0.064032	0.436493	0.132426	0.135659	0.985418	0.316905

1001 rows × 20 columns

# «Кластеризация» переменных: сокращение пространства признаков

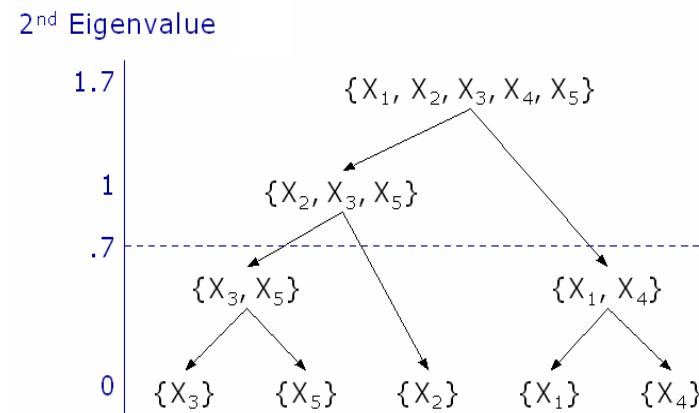
## ■ Задачи:

- группировка переменных в иерархические кластеры так, чтобы в одном кластере переменные были **максимально коррелированы**, а кластеры между собой нет
- Затем выбирается либо первая гл. компонента кластера либо лучший представитель кластера

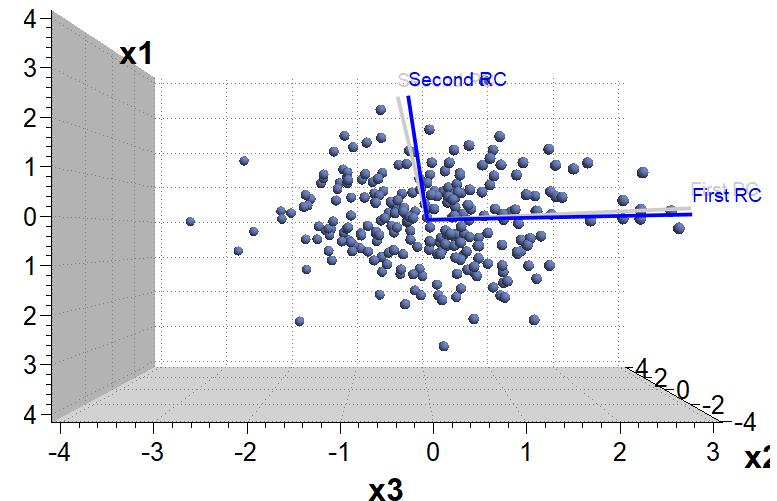
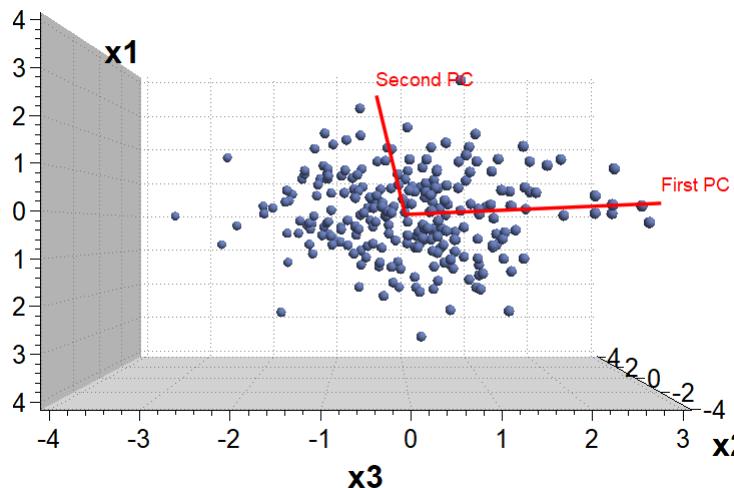
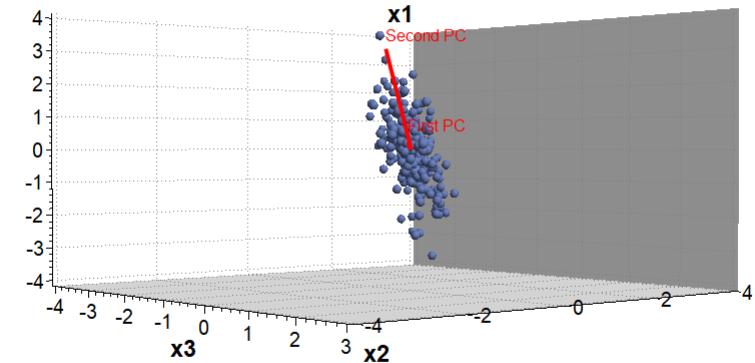
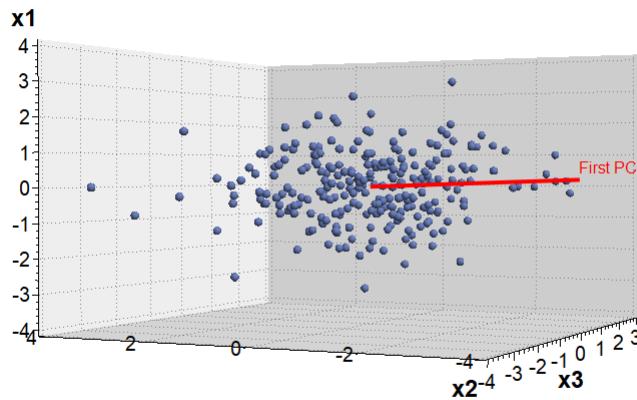


# Суть алгоритма группировки переменных

- Нисходящая иерархическая кластеризация, сначала все переменные в одном кластере
- Затем повторяется процесс:
  1. Выбор кластера (группы коррелирующих переменных) для разбиения
  2. Деление кластера на два с помощью метода гл. комп. с вращением
  3. Перераспределение переменных по кластерам



# Шаг разбиения

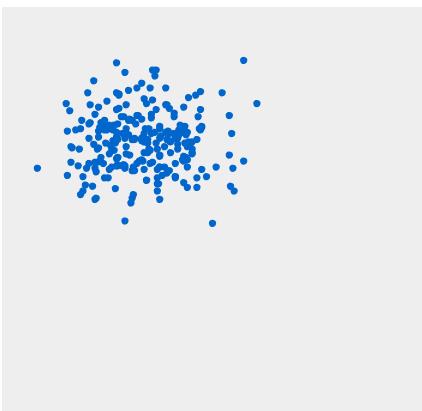


Поворот главных компонент

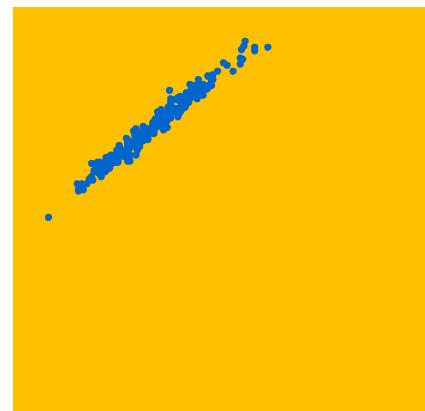
# Перераспределение переменных

First  
RC

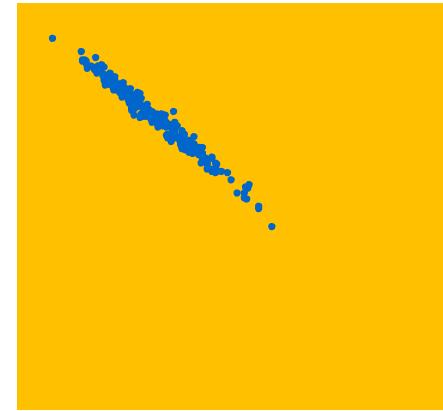
$X_1$



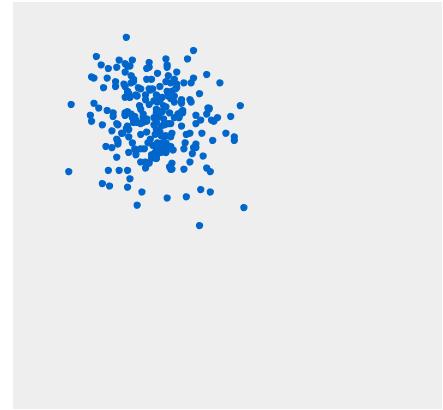
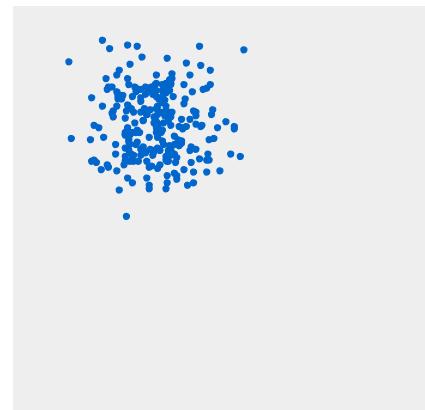
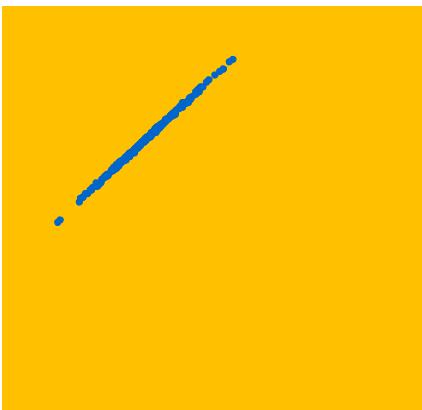
$X_2$



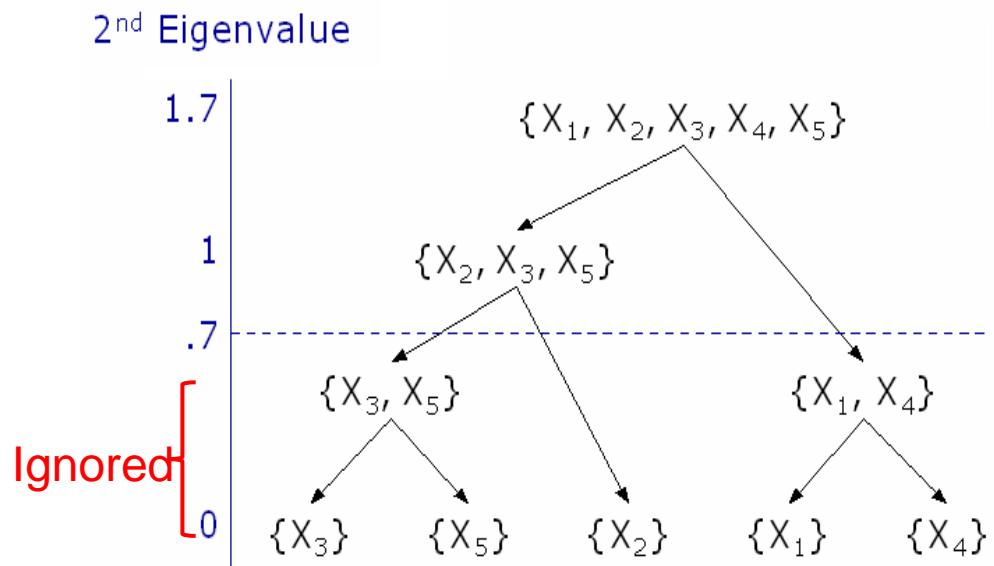
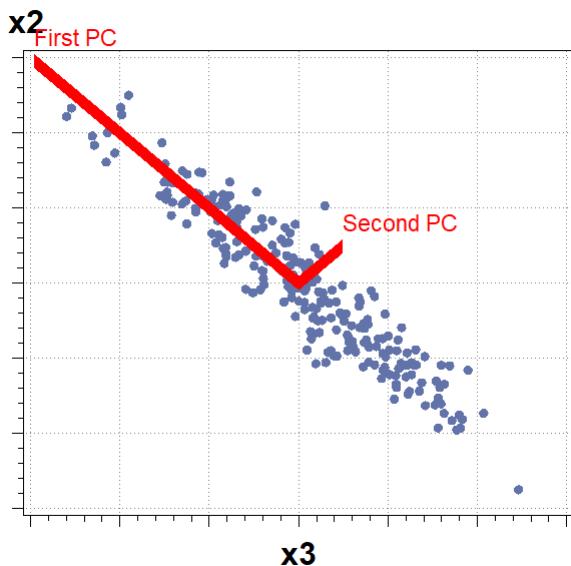
$X_3$



Second  
RC



# Разбивать ли дальше?

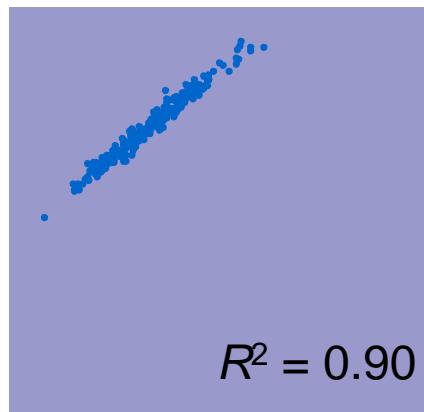


Варианты остановки (помимо порога на с.зн.):

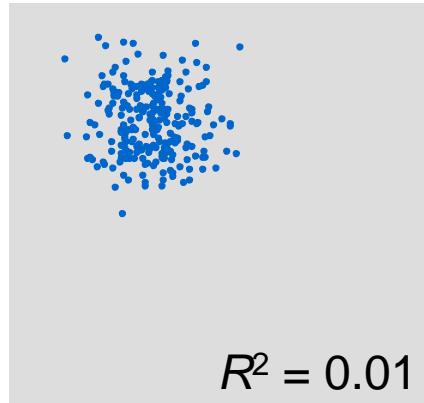
- Задать максимальное число кластеров
- Задать минимум описанной вариации (дисперсии)

# Выбор представителей кластеров

First  
Cluster  
PC



Second  
Cluster  
PC



$$\frac{1 - R^2_{\text{own cluster}}}{1 - R^2_{\text{next closest}}} = \frac{1 - 0.90}{1 - 0.01} = 0.101$$

Также можно выбирать не автоматически:

- Эксперт «вручную»
- По корреляции с откликом

# Пример использования (Python)

```
from varclushi import VarClusHi # varclushi package
```

```
clusters = VarClusHi(assoc, maxeigval2=1, maxclus=None) # assoc has products as columns
clusters.varclus()
```

clusters.rsquare

clusters.info

	Cluster	Variable	RS_Own	RS_NC	RS_Ratio
0	0	artichok	0.331882	0.073638	0.721228
1	0	avocado	0.542912	0.093912	0.504463
2	0	baguette	0.325976	0.059756	0.716861
3	0	olives	0.521717	0.110853	0.537912
4	0	turkey	0.269857	0.076499	0.790625
5	0	bourbon	0.277107	0.036053	0.749930
6	1	coke	0.784811	0.083987	0.234919
7	1	ice_crea	0.784811	0.080811	0.234108
8	2	apples	0.360347	0.076063	0.692313
9	2	corned_b	0.463746	0.075951	0.580331

	Cluster	N_Vars	Eigval1	Eigval2	VarProp
0	0	6	2.269452	0.993312	0.378242
1	1	2	1.569622	0.430378	0.784811
2	2	4	1.775050	0.903608	0.443763
3	3	5	1.747298	0.960767	0.349460
4	4	2	1.401720	0.598280	0.700860
5	5	1	1.000000	0.000000	1.000000

# Пример использования (Python)

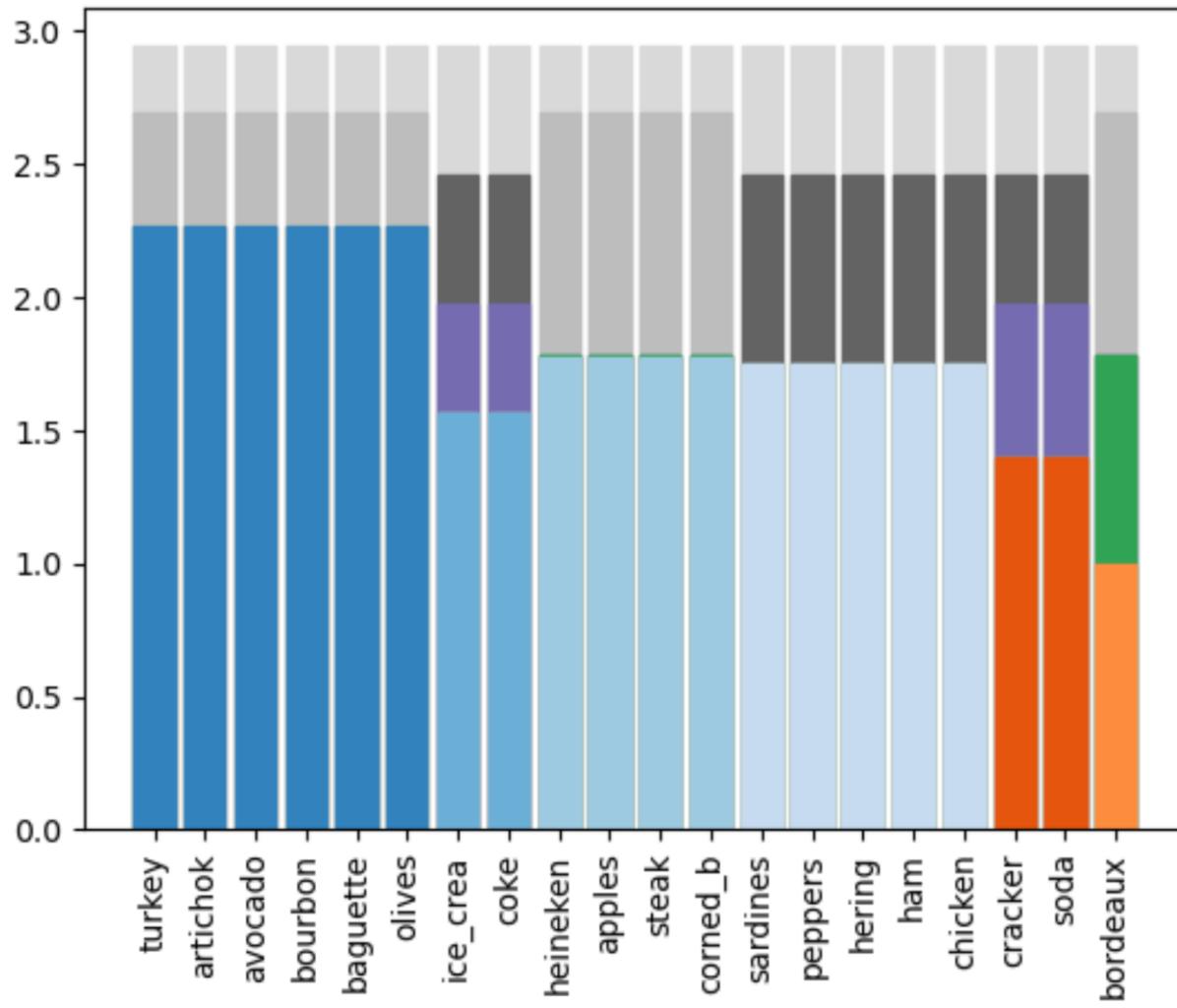
(1)

```
# The library doesn't provide the iterations
# So let's collect them manually
hierarchy = list()
variance_explained = dict()
for i in range(len(clusters.info), 0, -1):
    vch = VarClusHi(assoc, maxeigval2=1, maxclus=i)
    vch.varclus()
    for _, x in vch.clusters.items(): # number, cluster
        cluster = frozenset(x.clus)
        hierarchy.append(cluster)
        variance_explained[cluster] = x.eigval1
```

```
# Building linkage matrix for scipy.dendrogram is tedious
# Lets draw it manually
plt.xticks(rotation='vertical')
cmap = plt.cm.tab20c(np.linspace(0, 1, len(hierarchy)))
for i, cluster in enumerate(hierarchy):
    x = list(cluster)
    y = np.ones(len(x)) * variance_explained[cluster]
    for bar in plt.bar(x, y, zorder=-i):
        bar.set_color(cmap[i])
```

# Пример использования (Python)

(2)



# Graphical LASSO

$$\hat{\Theta} = \operatorname{argmin}_{\Theta \geq 0} \left( \operatorname{tr}(S\Theta) - \log \det(\Theta) + \lambda \sum_{j \neq k} |\Theta_{jk}| \right)$$

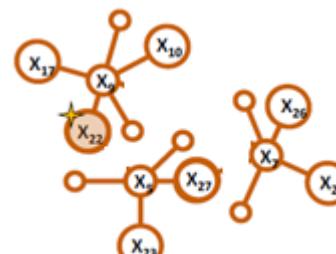
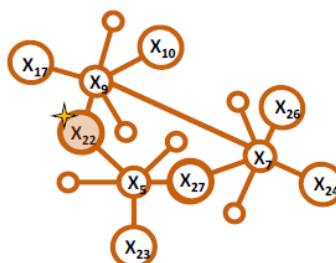
- Обратная ковариационная матрица  $\Theta$  описывает частичные попарные корреляции переменных,  $S$  наблюдаемая ковариационная матрицы,  $\lambda$  - параметр регуляризации, чем больше, тем матрица  $\Theta$  более разрежена:

$$\begin{pmatrix} \cdot & \cdot & \cdot & 0 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 0 & \cdot & \cdot & \cdot \end{pmatrix}$$

$$\begin{pmatrix} \cdot & \cdot & \cdot & 0 & \cdot \\ \cdot & \cdot & 0 & \cdot & 0 \\ \cdot & 0 & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 0 & 0 & \cdot & \cdot \end{pmatrix}$$

$$\begin{pmatrix} \cdot & 0 & \cdot & 0 & 0 \\ 0 & \cdot & 0 & 0 & 0 \\ \cdot & 0 & \cdot & \cdot & 0 \\ 0 & 0 & \cdot & \cdot & 0 \\ 0 & 0 & 0 & 0 & \cdot \end{pmatrix}$$

тем меньше дуг в графе связей переменных:



# Пример (Python)

```
from sklearn.covariance import GraphicalLasso
```

Регуляризация

```
cov = GraphicalLasso(alpha=0.01).fit(assoc)  
pd.DataFrame(columns=assoc.columns, index=assoc.columns, data=cov.covariance_)
```

PRODUCT	apples	artichok	avocado	baguette	bordeaux	bourbon	chicken	coke	corned_b
PRODUCT									
apples	0.227114	-0.021079	0.015019	0.018002	0.000000	-0.029258	-0.022309	-0.011946	0.025641
artichok	-0.021079	0.226149	0.087879	0.024196	0.000000	-0.036957	-0.014775	-0.028545	-0.032747
avocado	0.015019	0.087879	0.240205	0.062205	0.000000	-0.064977	-0.049531	-0.053192	-0.049661
baguette	0.018002	0.024196	0.062205	0.238251	0.000000	-0.050282	-0.035487	-0.038649	-0.052448
bordeaux	0.000000	0.000000	0.000000	0.000000	0.068461	0.000000	0.000000	0.000000	0.000000
bourbon	-0.029258	-0.036957	-0.064977	-0.050282	0.000000	0.260165	0.024992	0.027112	0.003663
chicken	-0.022309	-0.014775	-0.049531	-0.035487	0.000000	0.024992	0.222754	0.035286	0.009449
coke	-0.011946	-0.028545	-0.053192	-0.038649	0.000000	0.027112	0.035286	0.213072	-0.043645
corned_b	0.025641	-0.032747	-0.049661	-0.052448	0.000000	0.003663	0.009449	-0.043645	0.240249

# Пример (Python)

```
from scipy.sparse.csgraph import connected_components
```

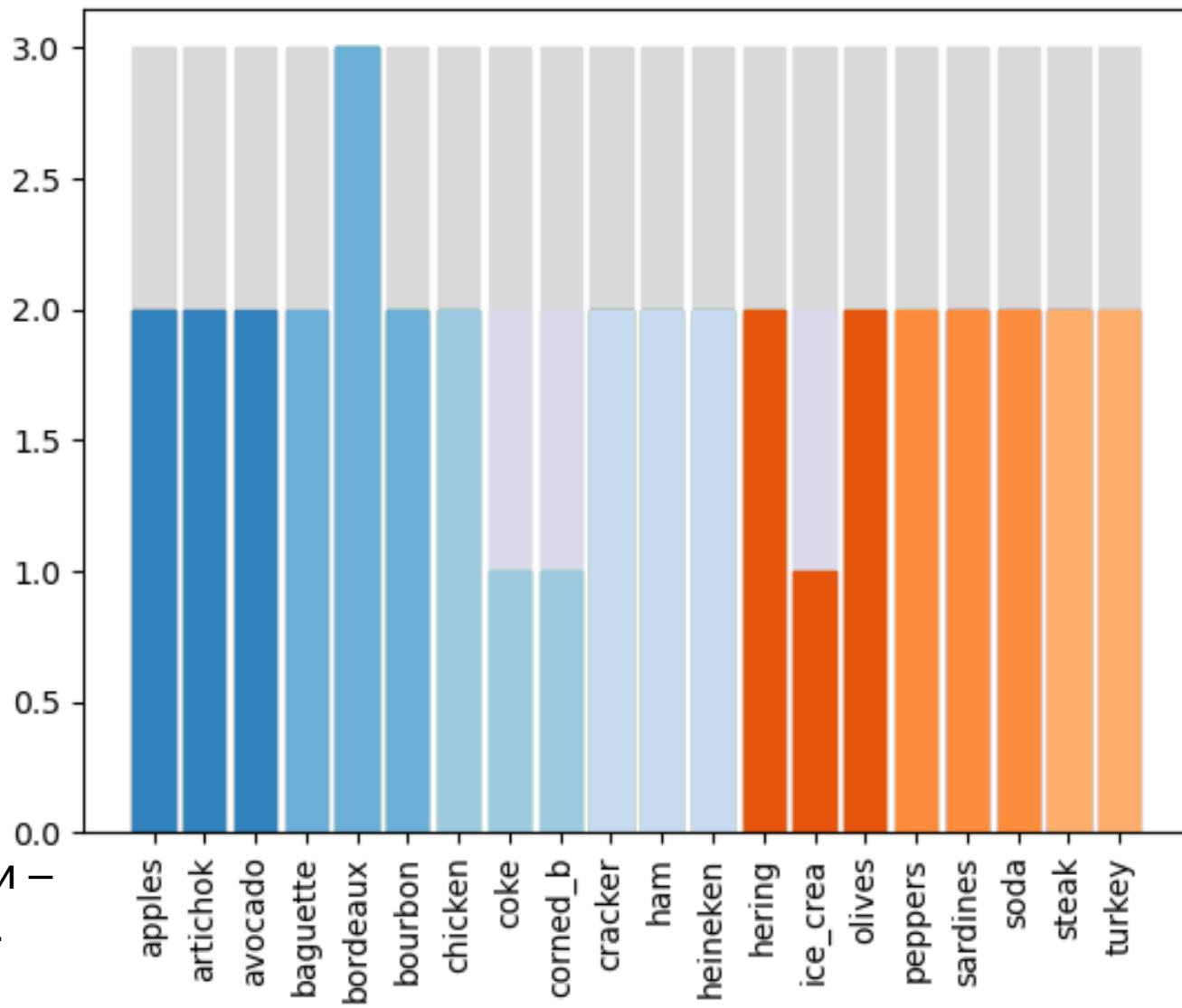
```
L = 0.001          Восходящая кластеризация переменных -  
RATE = 10         последовательный поиск обратной ковариационной  
STEPS = 4          матрицы с домножением параметра регуляризации при  
hierarchy = list()    каждом шаге  
step = dict()  
for i in range(STEPS, 0, -1):  
    cov = GraphicalLasso(alpha=L*STEP**i, max_iter=1000).fit(assoc)  
    distance = cov.covariance_ > 0  
    # Find clusters for current Lambda  
    n, clusters = connected_components(distance, directed=False)  
    # Repeat the dendrogram process  
    # but variance_explained -> step  
    for j in range(clusters.max() + 1):  
        cluster = frozenset(assoc.columns[clusters==j])  
        hierarchy.append(cluster)  
        step[cluster] = STEPS - i
```

(1)

# Пример (Python)

(1)

По вертикали –  
число шагов.



# Простые жадные алгоритмы отбора переменных

- $X_1$  - множество переменных, включенных в модель (изначально пустое)
- $X_2$  - множество переменных, НЕ включенных в модель (изначально все переменные)
- На каждом делается перенос одной переменной  $x$  из  $X_2$  в  $X_1$  , такое чтобы:

$$\min \text{Trace} \left( X_2^T \left( I - X_1 (X_1^T X_1)^{-1} X_1^T \right) X_2 \right)$$

- До тех пор, пока не отберется заданное число переменных, либо пока не будет описана заданная пропорция вариации
- Замечание: уравнение похоже на линейную регрессию, но  $X_1$  описывают не вариацию отклика  $Y$ , а вариацию  $X_2$

# Пример (Python)

Let  $\mathbf{C}_{11} = \mathbf{X}_1^\top \mathbf{X}_1$ ,  $\mathbf{C}_{12} = \mathbf{X}_1^\top \mathbf{X}_2$ , and  $\mathbf{C}_{21} = \mathbf{X}_2^\top \mathbf{X}_1$ . The following equations hold:

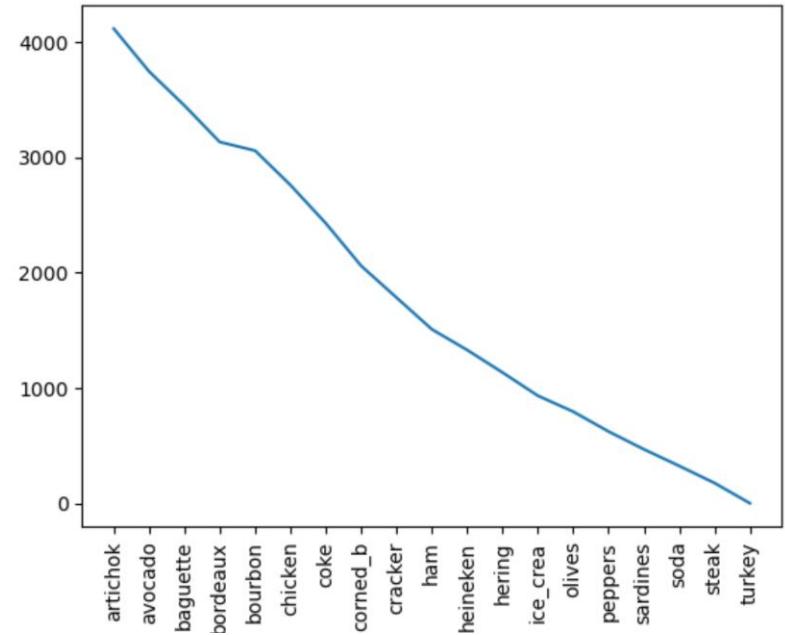
$$\mathbf{C} = \mathbf{X}^\top \mathbf{X} = \begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix}$$

$$\mathbf{X}_2^\top \left( \mathbf{I} - \mathbf{X}_1 (\mathbf{X}_1^\top \mathbf{X}_1)^{-1} \mathbf{X}_1^\top \right) \mathbf{X}_2 = \mathbf{C}_{22} - \mathbf{C}_{21} \mathbf{C}_{11}^{-1} \mathbf{C}_{12}$$

```
assoc = assoc - assoc.mean()  
matrix = assoc.values
```

```
history = []  
for i in range(1, len(assoc.columns)):  
    X1, X2 = matrix[:, :i], matrix[:, i:]  
    C11 = X1.T @ X1  
    C12 = X1.T @ X2  
    C21 = X2.T @ X1  
    C22 = X2.T @ X2  
    x = C22 - C21 @ np.linalg.inv(C11) @ C12  
    score = np.trace(x)  
    history.append(score)
```

```
plt.plot(assoc.columns[1:], history)  
plt.xticks(rotation='vertical')  
pass
```

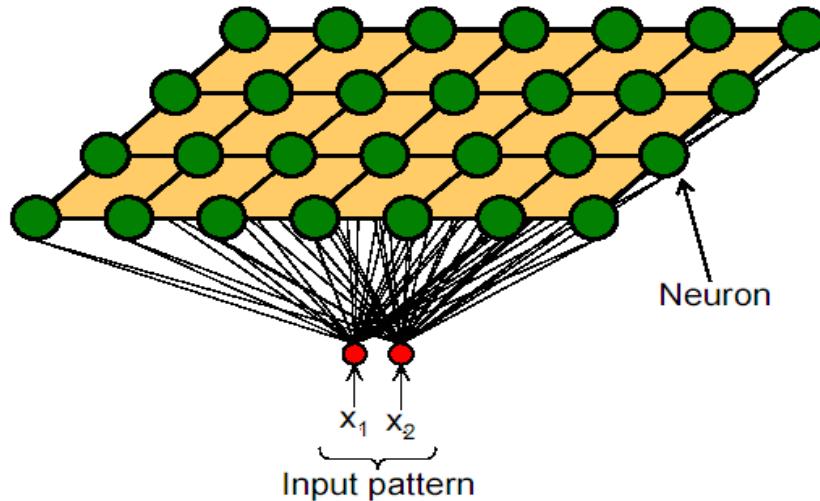


# Self-Organizing Maps (SOM)

## ■ Общая идея нейросетевого подхода (сети Кохонена):

- Базируется на моделировании процесса обучения/запоминания в мозге
- Каждый кластер (нейрон) определяется своим «прототипом» (число кластеров задается априори)
- Прототипы (нейроны) объединены в виде 2D (реже 3D) решетки (сети) с квадратными (или шестиугольными) ячейками
- Структура решетки определяет понятие «окрестности» каждого прототипа (дискретное расстояние по решетке)
- У прототипа кластера (нейрона) есть векторный «вес» – соответствует точке в исходном пространстве
- Процесс активации – реакция на образ входного пространства, определяется мерой сходства между «весом» нейрона и входным образом (или расстоянием между прототипом кластера и объектом)
- Конкурентное обучение: нейроны соревнуются за право активации (winner-takes-all, всегда один ближайший - победитель)

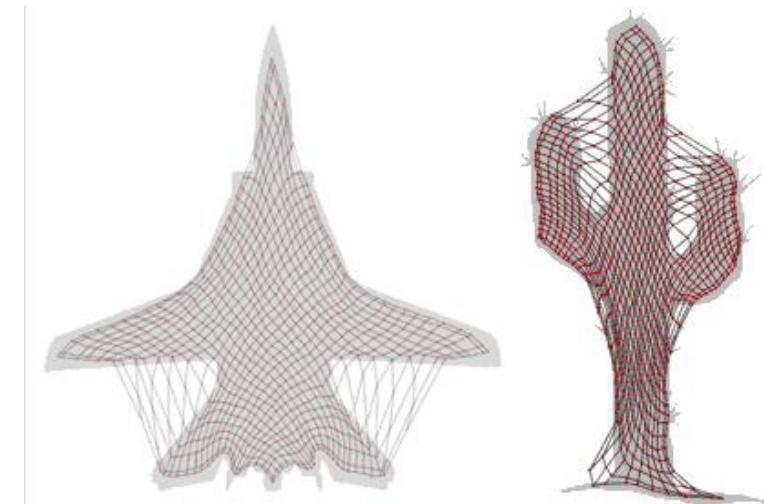
# Основная задача SOM



- Задача:
  - формирование топографической карты входных образов, в которой пространственное расположение нейронов решетки (прототипов кластеров) в некотором смысле отражает статистические закономерности во входных параметрах.
- Или:
  - построение отображения многомерного исходного пространства на 2x (или 3x) мерную решетку с сохранением топологических зависимостей (близкие объекты исходного пространства будут рядом и на решетке).

# Процедура работы SOM (неформально)

- Неформально:
  - Есть решетка нейронов  $r \times s$ , с каждым узлом которой связан центр кластера исходного пространства  $\mu_{1,1}, \dots, \mu_{r,s}$
  - Алгоритм SOM двигает центры кластеров в исходном многомерном пространстве, сохраняя топологию решетки
  - Точка исходного пространства относится к тому кластеру, чей вес ближе (расстояние до центра меньше)
  - При обработке новой точки центр кластера-победителя и всех его соседей по решетке сдвигается в сторону этой точки
- Упрощенный пример:
  - Добавляя точки из картинок, решетка «обтягивает» их контур
  - Небольшой обман, ибо тут размерность решетки и исходного пространства совпадают



# Процедура работы SOM

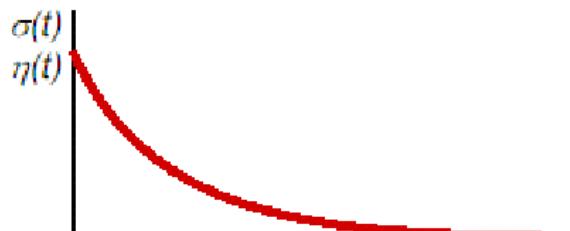
- Шаг 0. Инициализация:
  - структура решетки и число кластеров (нейронов)
  - инициализация «весов» прототипов  $w_j(0)$  (полностью случайно или случайной выборкой из данных)
  - начальные параметры (скорость обучения и размер окрестности)
- Шаг 1. Выборка (итерация  $t$ ):
  - Выбираем случайный  $x(t)$  из исходного пространства
- Шаг 2. Конкуренция:
  - Находим «лучший» нейрон для активации:  $i(x) = \arg \min_j \|x(t) - w_j(t)\|$
- Шаг 3. Коррекция весов с учетом кооперации:
  - Для победителя и соседей по решетке пересчитываем их «вес» – двигаем их центры к точке  $x$  в исходном пространстве
  - Уменьшаем скорость обучения и размер окрестности
- Шаг 4. Проверка условий остановки и переход на Шаг 1.
  - Стабилизация структуры либо превышение числа выполненных итераций установленного значения

# Коррекция весов с учетом кооперации

- Перерасчет весов победителя и соседей:
  - Стохастический градиентный спуск:

$$w_j(t+1) = w_j(t) + \eta(t) h_{ij(x)}(t)(x - w_j(t))$$

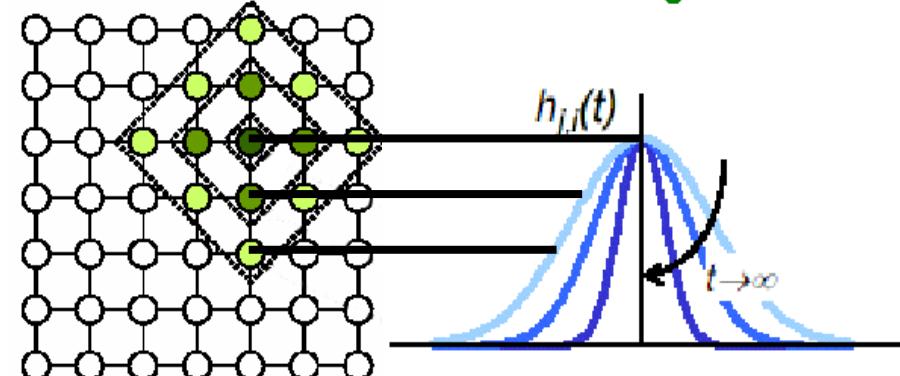
скорость обучения



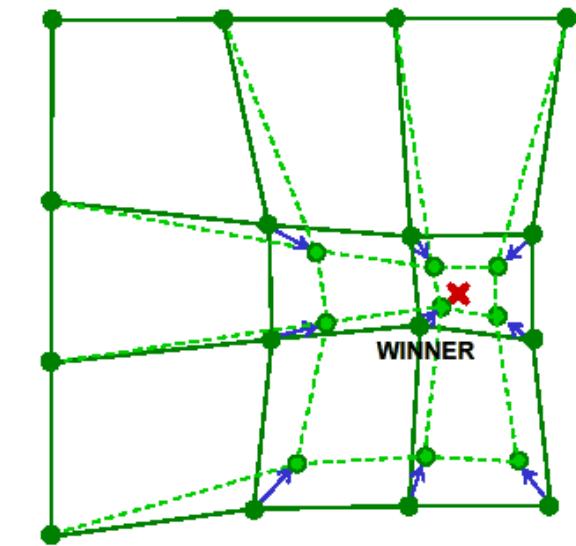
$$\sigma(t+1) = \sigma_0 \exp\left(-\frac{t}{\tau_\sigma}\right)$$

$$\eta(t+1) = \eta_0 \exp\left(-\frac{t}{\tau_\eta}\right)$$

размер топологической окрестности (на решетке!!!!)



$$h_{ij(x)}(t) = \exp\left(-\frac{d_{grid}^2(i, j)}{2\sigma^2(t)}\right)$$



# Приимер

## ■ Входные данные:

продукт	белки	углеводы	жиры
Apples	0.4	11.8	0.1
Avocado	1.9	1.9	19.5
Bananas	1.2	23.2	0.3
Beef Steak	20.9	0.0	7.9
Big Mac	13.0	19.0	11.0
Brazil Nuts	15.5	2.9	68.3
Bread	10.5	37.0	3.2
Butter	1.0	0.0	81.0
Cheese	25.0	0.1	34.4
Cheesecake	6.4	28.2	22.7
Cookies	5.7	58.7	29.3
Cornflakes	7.0	84.0	0.9
Eggs	12.5	0.0	10.8
Fried Chicken	17.0	7.0	20.0
Fries	3.0	36.0	13.0
Hot Chocolate	3.8	19.4	10.2
Pepperoni	20.9	5.1	38.3
Pizza	12.5	30.0	11.0
Pork Pie	10.1	27.3	24.2
Potatoes	1.7	16.1	0.3
Rice	6.9	74.0	2.8
Roast Chicken	26.1	0.3	5.8
Sugar	0.0	95.1	0.0
Tuna Steak	25.6	0.0	0.5

## ■ SOM(10X10):



# Приимер

```
wine = pd.read_csv("white_wine.csv")
wine
```

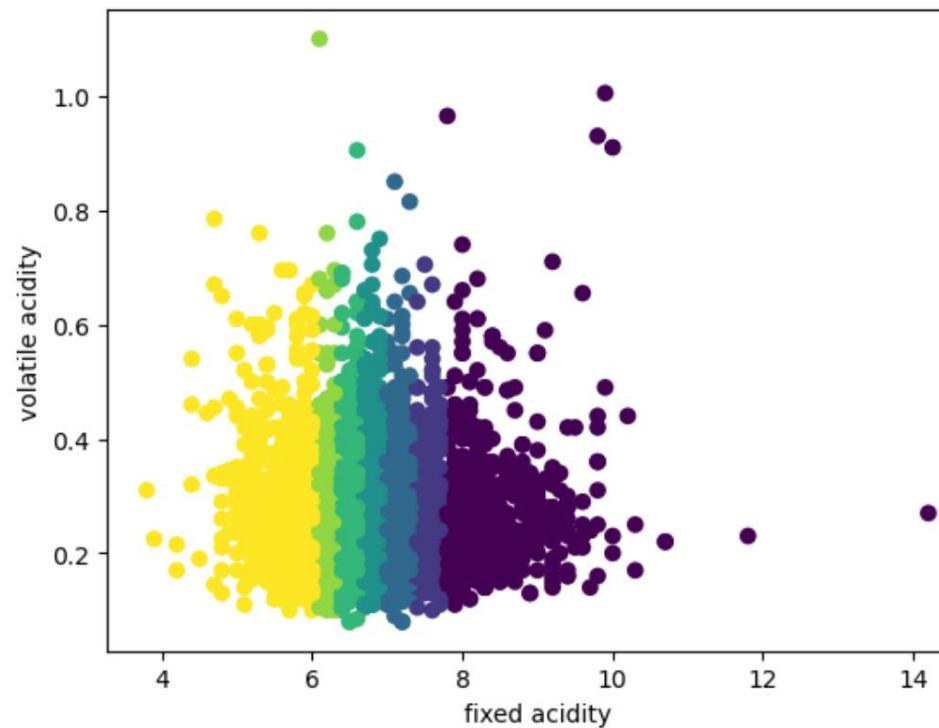
	fixed acidity	volatile acidity	target_quality
0	7.0	0.27	6
1	6.3	0.30	6
2	8.1	0.28	6
3	7.2	0.23	6
4	7.2	0.23	6
...	...	...	...
4893	6.2	0.21	6
4894	6.6	0.32	5
4895	6.5	0.24	6
4896	5.5	0.29	7
4897	6.0	0.21	6

4898 rows × 12 columns

```
X = wine.values[:, :2]
som = SOM(m=1, n=7, dim=2) # quality has 7 values
som.fit(X)
```

```
pred = som.predict(X)
```

```
plt.scatter(wine.iloc[:, 0], wine.iloc[:, 1], c=pred)
plt.xlabel("fixed acidity")
plt.ylabel("volatile acidity")
```



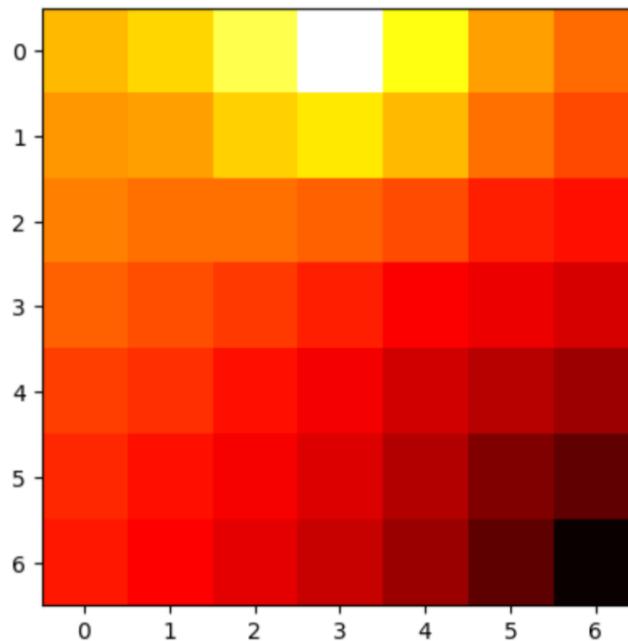
# Свойства SOM

- Апроксимация входного пространства признаков
  - «Сжатие» информации, связь с методом LVQ (кластеризация на основе теории информации, задача - выбрать кодовые слова-кластеры так, чтобы минимизировать возможное искажение)
- Топологический порядок
  - Рядом в исходном пространстве => рядом на решетке и наоборот
- Соответствие плотности
  - Области исходного пространства с высокой плотностью отображаются в большие области на решетке и наоборот
- Выбор признаков:
  - осуществляет нелинейную дискретную аппроксимацию главных компонент (точнее главных кривых и плоскостей)
- Недостатки:
  - Алгоритм простой, но мат. анализу поддается плохо, в общем случае не доказана ни сходимость, ни даже устойчивость
  - Много неочевидных, но важных параметров, задаваемых априори, включая структуру решетки

# Применение (Python)

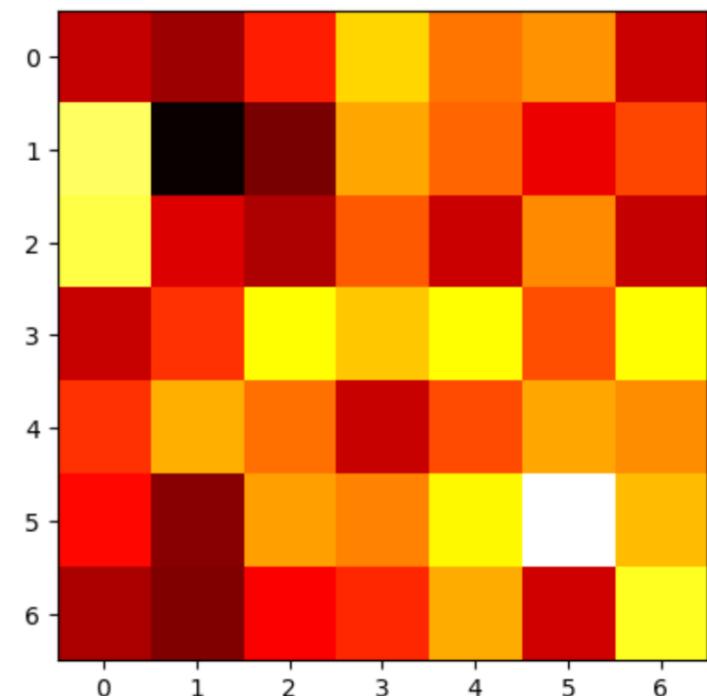
Распределение переменной  
(карта интенсивности)

```
# First variable - X[:, 0] (this example is 7x7)
pred = som.predict(X)
xy = np.array([pred // 7, pred % 7, X[:, 0]]).T
map_ = pd.DataFrame(xy)
map_ = map_.groupby([0, 1]).mean().unstack()
plt.imshow(map_, cmap='hot', interpolation='nearest')
```

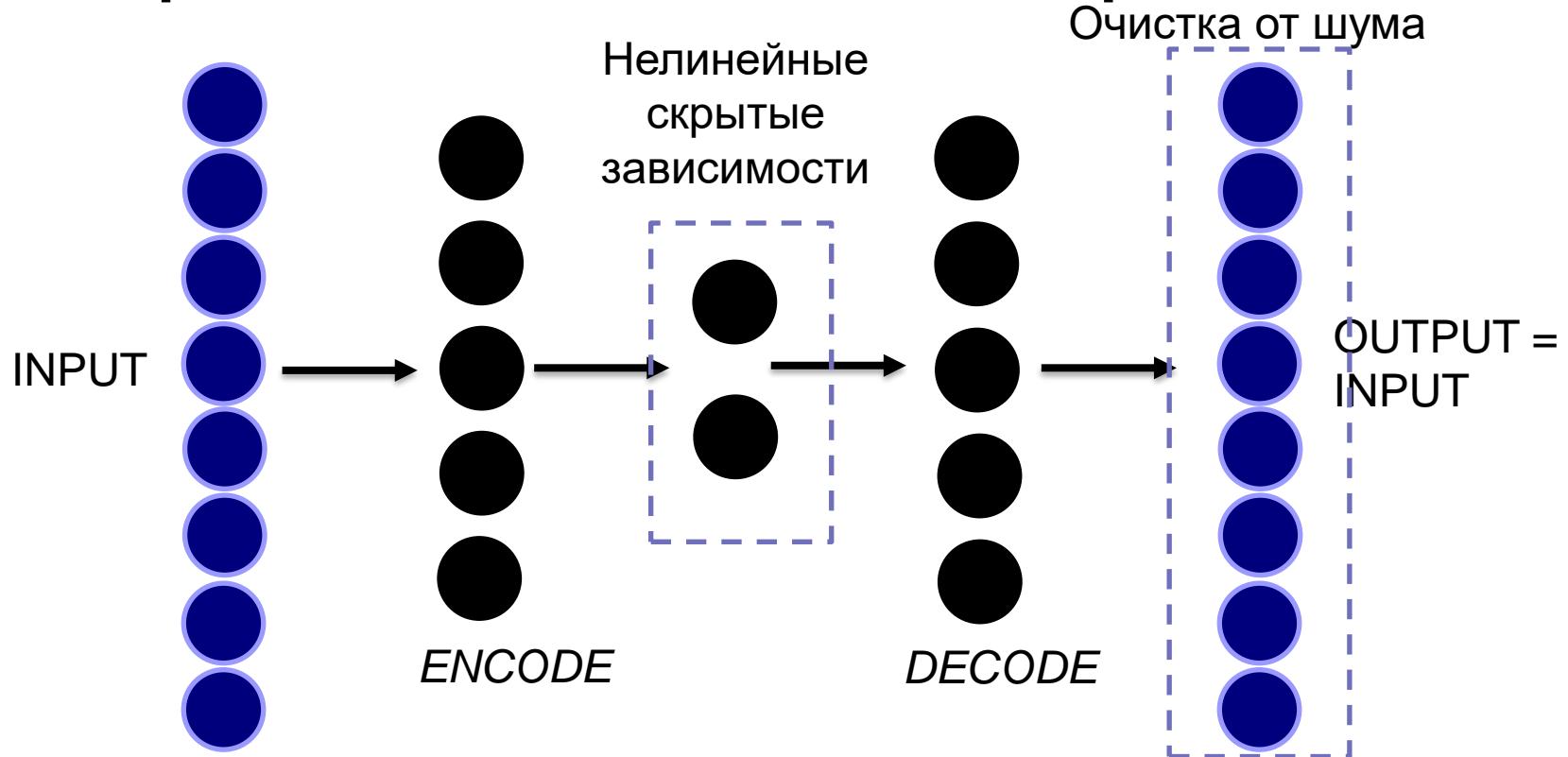


Размеры кластеров  
(карта интенсивности)

```
map_ = pd.DataFrame(xy).value_counts().unstack()
plt.imshow(map_, cmap='hot', interpolation='nearest')
pass
```



# Нейросетевой автокодировщик



- использует входной вектор как входной и выходной,
- через «бутилочное горло» среднего уровня выявляются нелинейные скрытые зависимости, описывающие вариацию
- выходном часто используется как очистка от шума
- обнаружение выбросов на основе ошибки реконструкции
- Очень важна регуляризация (L1,L2, dropout)

# Пример (Python)

```
from sklearn.neural_network import MLPRegressor  
  
wine = pd.read_csv("white_wine.csv")  
  
model = MLPRegressor(hidden_layer_sizes=(5,), # 1 hidden layer  
                      max_iter=200, # 100 epochs to fit  
                      alpha=0.0001) # L2 regularization  
  
model.fit(X=wine, y=wine)
```

# Stochastic Neighbor Embedding

Цель как и SOM – отобразить многомерное исходное пространство  $X$  в двух или трехмерное пространство  $Y$  *топологически корректно!*

Для этого оценка близости двух точек в исходном многомерном пространстве  $X$  и в пространстве отображения  $Y$  через условные вероятности (в предположении нормального распределения) :

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)},$$

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}.$$

# Stochastic Neighbor Embedding

SNE ищет такое отображение, чтобы сохранить близость распределений, оценивая их через дивергенцию Кульбака-Лейблера:

$$C = \sum_i KL(P_i || Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}},$$

При этом градиент:

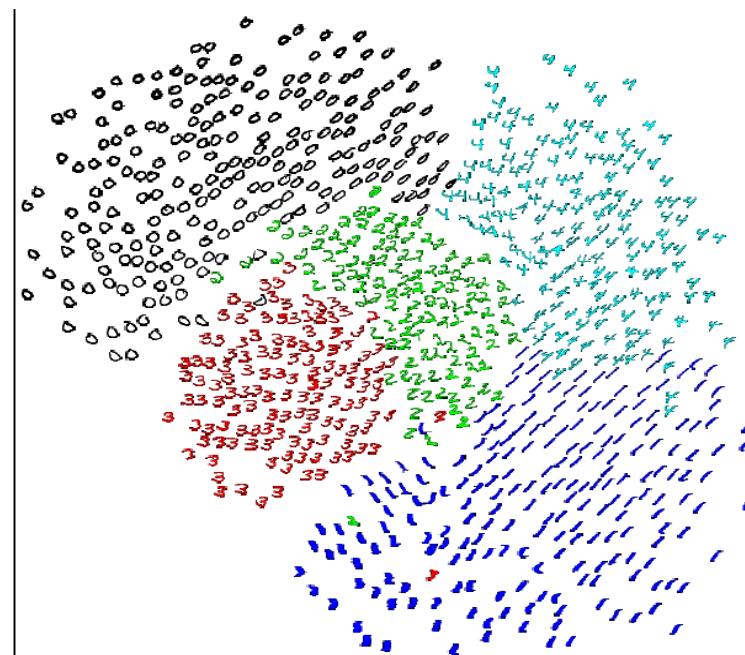
$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j).$$

и градиентный спуск:

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) (\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)})$$

# Stochastic Neighbor Embedding

- Достоинство: рядом в X => рядом в Y и далеко в X => относительно далеко в Y
- Недостаток: неэффективная оптимизация и проблема «скученных точек» (crowding problem):



# SSNE (симметричный SNE) – эффективнее оптимизация

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)},$$

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}.$$

$$C = \sum_i KL(P_i || Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}},$$

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j).$$

$$\left\{ \begin{array}{l} p_{ij} = 0.5(p_{j|i} + p_{i|j}) \\ p_{ij} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2)}{\sum_{k < l} \exp(-\|\mathbf{x}_k - \mathbf{x}_l\|^2 / 2\sigma^2)} \end{array} \right.$$

$$q_{ij} = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2)}{\sum_{k < l} \exp(-\|\mathbf{y}_k - \mathbf{y}_l\|^2)}$$

$$C_{sym} = KL(P || Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

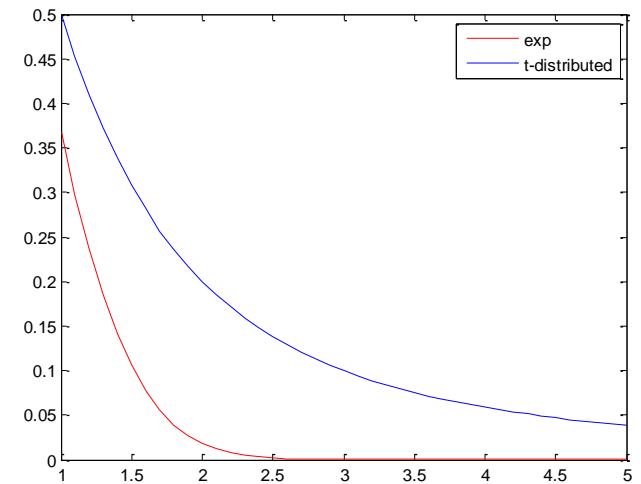
$$\frac{\partial \mathcal{J}}{\partial y_i} = 4 \sum_j (P_{ij} - Q_{ij})(y_i - y_j).$$

# t-SNE

Симметричный SNE с распределением стьюдента вместо нормального (более тяжелый хвост) в пространстве  $Y$ :

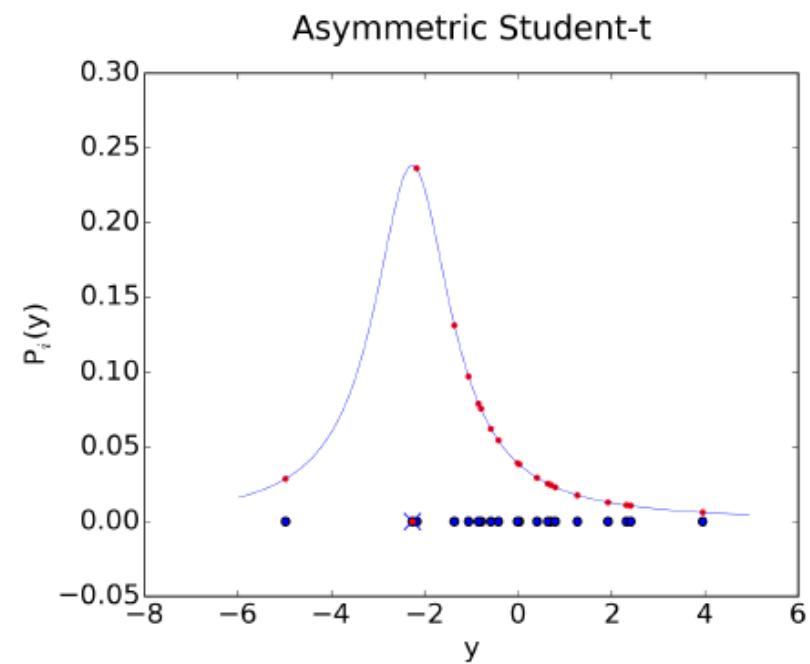
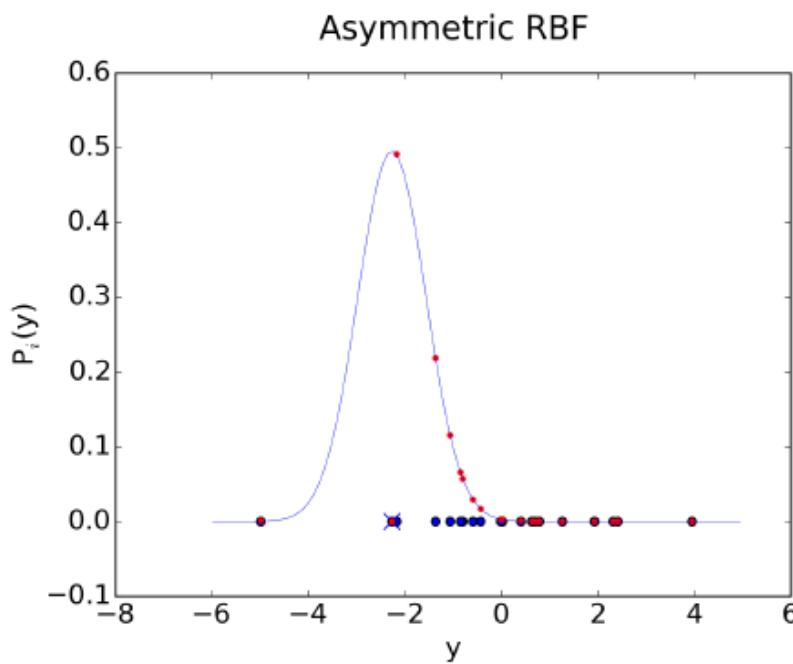
$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n},$$

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$



$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij}) (y_i - y_j) (1 + \|y_i - y_j\|^2)^{-1}.$$

# Почему в t-SNE распределение Стьюдента?



У t-распределения более тяжелый хвост – при  
отображении точки в хвосте не так «скучены»

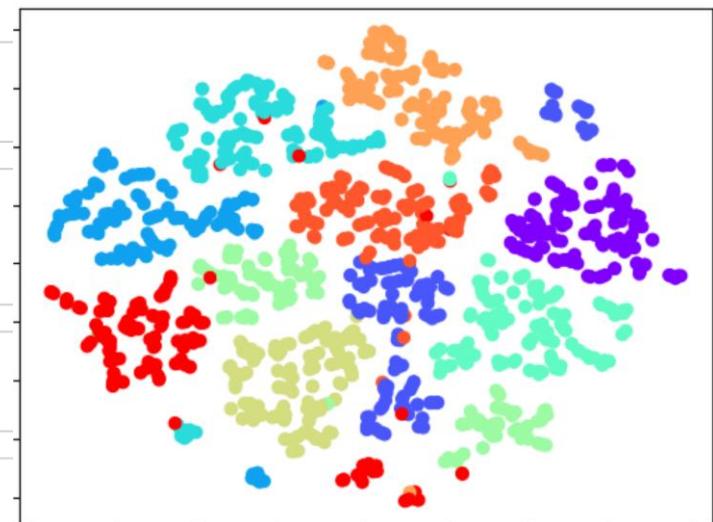
# Пример (набор MNIST)

```
from sklearn.datasets import load_digits  
from sklearn.manifold import TSNE
```

```
MNIST = load_digits()  
X = MNIST.data  
Y = MNIST.target
```

```
embedded = TSNE(n_components=2, learning_rate='auto',  
                 init='random', perplexity=3).fit_transform(X)
```

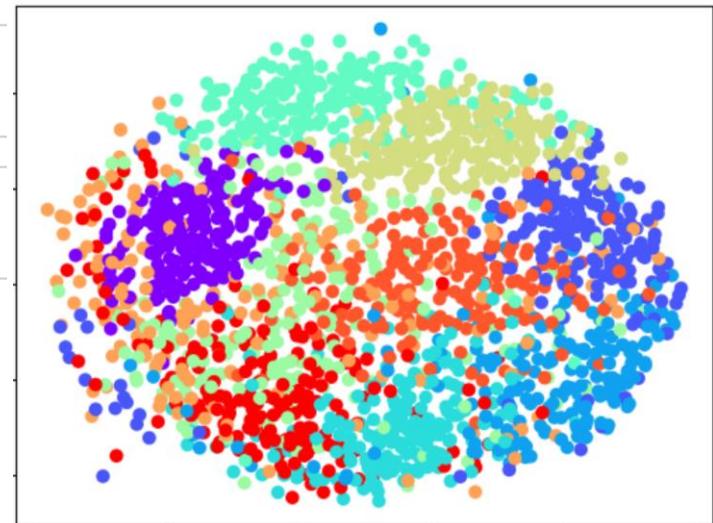
```
plt.scatter(*embedded.T, c=Y, cmap="rainbow")
```



```
# https://github.com/tompollard/sammon  
from sammon import sammon
```

```
# map_ is the output map, E is the final cost  
map_, E = sammon(X, n=2, maxiter=100) # n - output dimension
```

```
plt.scatter(*map_.T, c=Y, cmap="rainbow")
```

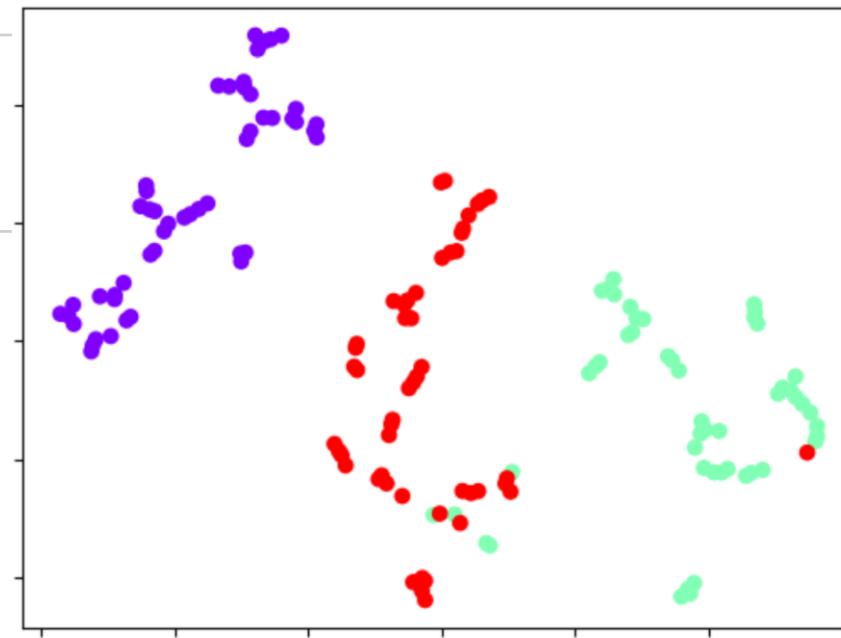


# Примеры разных схем визуализации для набора Iris

```
from sklearn.datasets import load_iris
```

```
iris = load_iris()  
X = iris.data  
Y = iris.target
```

tSNE

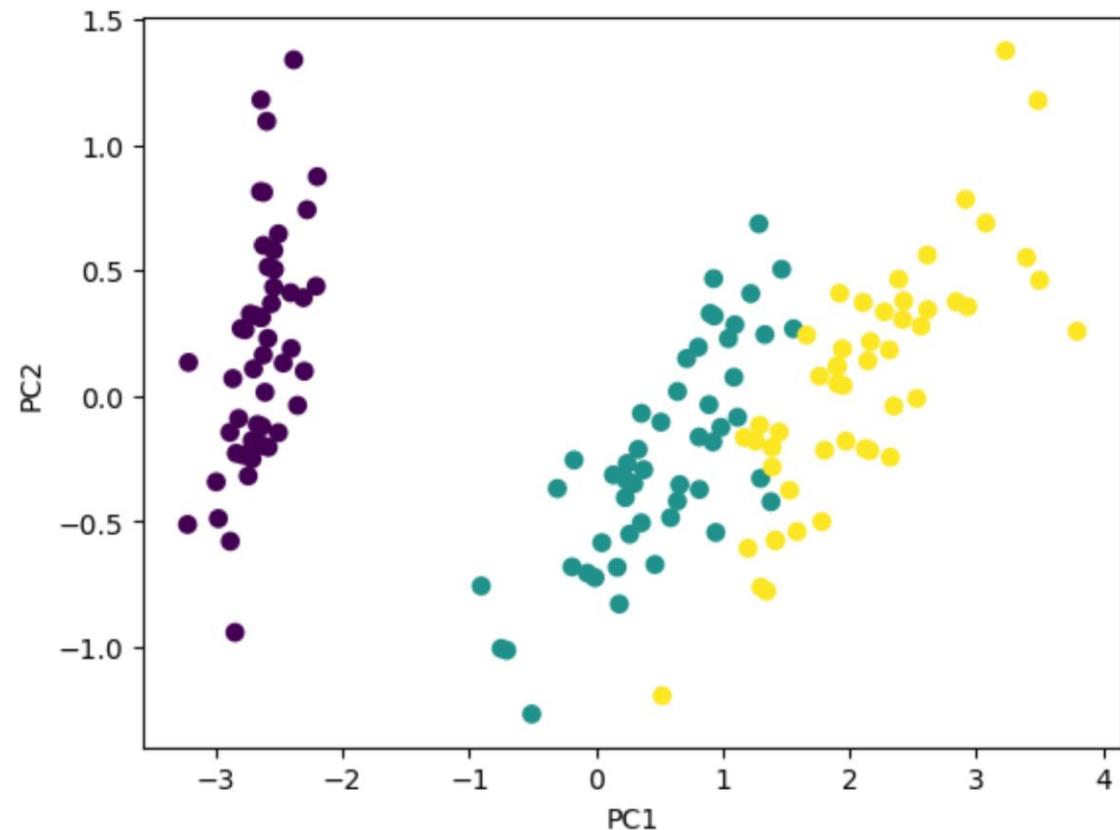


```
# t-SNE  
from sklearn.manifold import TSNE  
embedded = TSNE(n_components=2, learning_rate='auto',  
                 init='random', perplexity=3).fit_transform(X)  
plt.scatter(*embedded.T, c=Y, cmap="rainbow")
```

# Примеры разных схем визуализации для набора Iris

```
# PCA
from sklearn.decomposition import PCA
features = PCA(n_components=2).fit_transform(X)
plt.scatter(features[:, 0], features[:, 1], c=Y)
plt.xlabel("PC1")
plt.ylabel("PC2")
pass
```

Линейный PCA



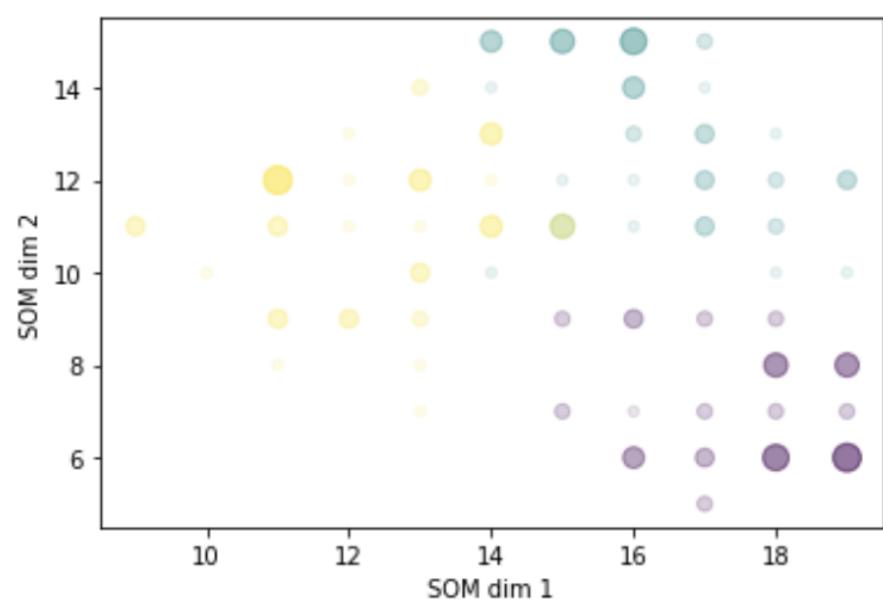
# Примеры разных схем визуализации для набора Iris

```
# SOM
from sklearn_som.som import SOM
N, M = 20, 20
pred = SOM(m=M, n=N, dim=X.shape[-1], lr=4).fit_predict(X)
cnt=np.unique(pred, return_counts=True)
cnts=[dict(map(lambda i,j : (i,j) , cnt[0],cnt[1]))[k]*20 for k in pred]

xy = np.array([pred // N, pred % M, Y]).T

plt.scatter(xy[:, 0], xy[:, 1], c=Y, alpha=0.1, s=cnts)
plt.xlabel("SOM dim 1")
plt.ylabel("SOM dim 2")
```

SOM с решеткой 20 на 20



# Примеры разных схем визуализации для набора Iris

```
# Neural
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(hidden_layer_sizes=(2,), max_iter=1000, alpha=0.0001)
model.fit(X, X)

dummy = np.zeros((1, 2)) # Only dimension matters
hidden = MLPRegressor(hidden_layer_sizes=[], max_iter=1).fit(X[:1], dummy)
# hidden.coefs_: [(2, 4)], model.coefs_: [(4, 2), (2, 4)]
hidden.coefs_[0] = model.coefs_[0] # setting weight matrixes
```

```
# Lets build a dummy model to assess the hidden layer
```

```
#
# * \ / *
# * - * - *
# * - * - * - model
# * / \ *
#
# * \
# * - *
# * - * - dummy
# * /
#
neurons = hidden.predict(X)
plt.scatter(*neurons.T, c=Y)
plt.xlabel("Neuron-1")
plt.ylabel("Neuron-2")
pass
```

АЕ с 2 нейронами в среднем слое

