



# Лекция 18: Выявление скрытых структур в данных

# Выявление скрытых структур в данных

- Выявление скрытых структур в данных с целью:
  - Сокращения признакового пространства (отбор или преобразование признаков) с учителем и без
  - Интерпретируемые (понятные человеку) проекции многомерных данных на простые структуры (решетки, плоскости, трехмерные фигуры)
  - Решение прикладных задач, например, тематического моделирования или рекомендательных систем
- Рассмотрим (некоторые вспомним):
  - Фильтрацию признаков в задачах с учителем и без
  - Пошаговый отбор признаков, L1 регуляризация, линейные методы сокращения признакового пространства, некоторые приложения
  - Нелинейные векторизации и проекции (SOM, t-SNE, UMAP) сложных структур данных

# Фильтрация признаков в задачах с ЧИСЛОВЫМ ОТКЛИКОМ

- Фильтрация – признаки отбираются независимо друг от друга:
  - Оценивается сила или значимость связи признак-отклик, все признаки упорядочиваются по этой оценке и отбираются по порогу или топ n штук
- Если числовой отклик то:
  - для числовых признаков для проверки линейных зависимостей можно использовать корреляцию Пирсона:  $r_{xy} = \frac{cov_{xy}}{\sigma_x \sigma_y}$
  - для нелинейных зависимостей и числового предиктора – ранговые корреляции, например, Спирмана:  $\rho_{xy} = 1 - \frac{6}{l(l-1)(l+1)} \sum_{i=1}^l (r(x_i) - r(y_i))^2$
  - если категориальный предиктор принимает B различных значений  $\{v_1, \dots, v_B\}$  или он числовой, но дискретезирован на B групп (бакеты, квантили или оптимальные отрезки), то как в деревьях решений можно использовать уменьшение вариации отклика или критерий Фишера:

$$F = \left( \frac{SS_{model}}{SS_{error}} \right) \left( \frac{l-B}{B-1} \right) \sim F_{B-1, l-B}, \text{ где } SS_{total} = \sum_{i=1}^l (y_i - \bar{y})^2,$$

$$SS_{error} = \sum_{b=1}^B \sum_{i:x_i=v_b} (y_i - \bar{y}_b)^2, SS_{model} = SS_{total} - SS_{error}$$

# Фильтрация признаков в задачах с категориальным откликом

- Если отклик принимает  $K$  значений то для категориальных предикторов с  $B$  различными значениями или для дискретизированных числовых признаков с  $B$  группами:
  - Обозначим:  $p_k = P(y = k)$  – априорная вероятность значения отклика  $k$ ,  
 $l_b = |\{x_i = v_b\}|$  - число наблюдений с  $b$ -м значением предиктора,  
 $p_{kb} = P(y = k | x_i = v_b)$  – условная вероятность значения отклика  $k$  для наблюдений с  $b$ -м значением предиктора
  - Статистические критерии для анализа таблиц сопряженности, например,  $\chi^2 = \sum_{k=1}^K \sum_{b=1}^B \frac{l_b(p_{kb} - p_k)^2}{p_k} \sim \chi^2_{(B-1)(K-1)}$
  - Критерии на основе индекса gini:

$$\Delta Gini(x) = 1 - \sum_{k=1}^K p_k^2 - \sum_{b=1}^B \frac{l_b}{l} \left(1 - \sum_{k=1}^K p_{kb}^2\right)$$

- Критерии на основе взаимной информации (разность между энтропией отклика и ожидаемой условной энтропией отклика):

$$IG = - \sum_{k=1}^K p_k \log_2 p_k + \sum_{b=1}^B \frac{l_b}{l} \sum_{k=1}^K p_{kb} \log_2 p_{kb}$$

# Фильтрация признаков в задачах с бинарным откликом

- Можно рассматривать бинарный отклик:
  - как частный случай числового со значениями 0 и 1 или как частный случай категориального с двумя значениями и пользоваться соответствующими инструментами, рассмотренными выше
- Более эффективно пользоваться для фильтрации переменных в задачах с бинарным откликом (если категориальный или дискретезированный предиктор принимает  $B$  значений  $\{v_1, \dots, v_B\}$ )
  - отношением шансов и WOE (чем ближе WOE к 0 тем менее значимая переменная) или сглаженным WOE:

$$WOE(x) = \sum_{b=1}^B WOE_b,$$
$$WOE_b = \log_2 \left( \frac{\text{count}(y = 1) \text{and} (x = v_b)) / \text{count}(y = 1)}{\text{count}(y = 0) \text{and} (x = v_b)) / \text{count}(y = 0)} \right)$$

- Информационной важностью предиктора:

$$IV = \sum_{b=1}^B (P(x = v_b | y = 1) - P(x = v_b | y = 0)) WOE_b$$

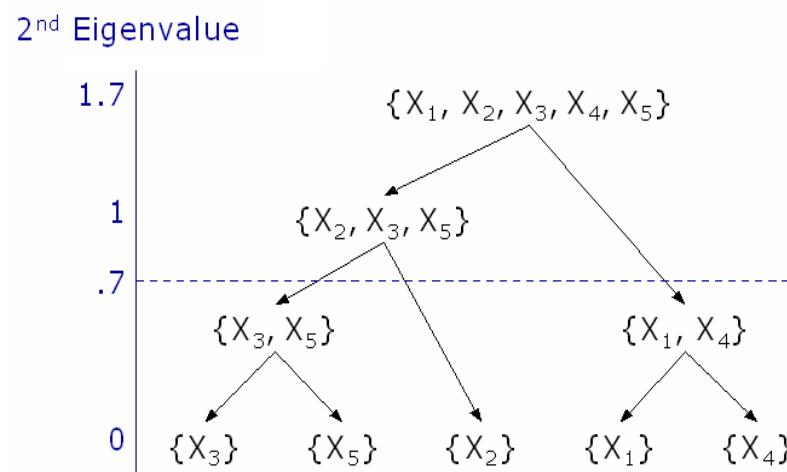
# Фильтрация признаков в задачах без учителя

- Можно посчитать попарные взаимные оценки силы связей предикторов, но как выбрать лучших представителей?
- Методы кластеризации переменных без учителя:
  - группировка переменных в иерархические кластеры так, чтобы в одном кластере переменные были **максимально коррелированы**, а кластеры между собой нет
  - затем выбирается лучший представитель кластера

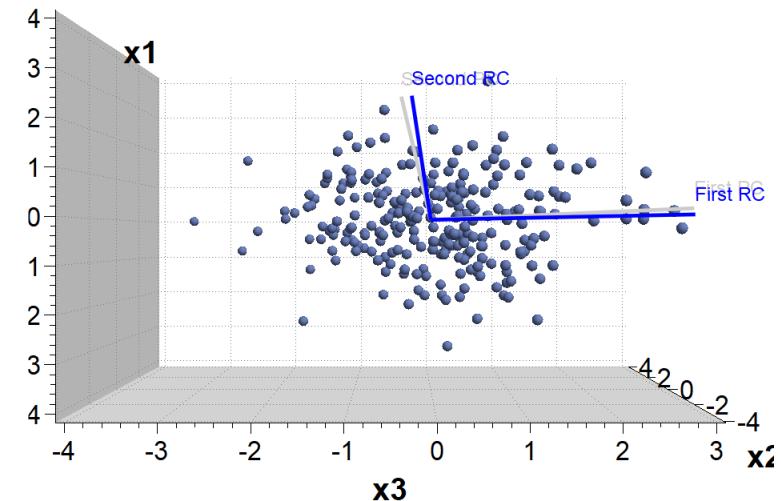
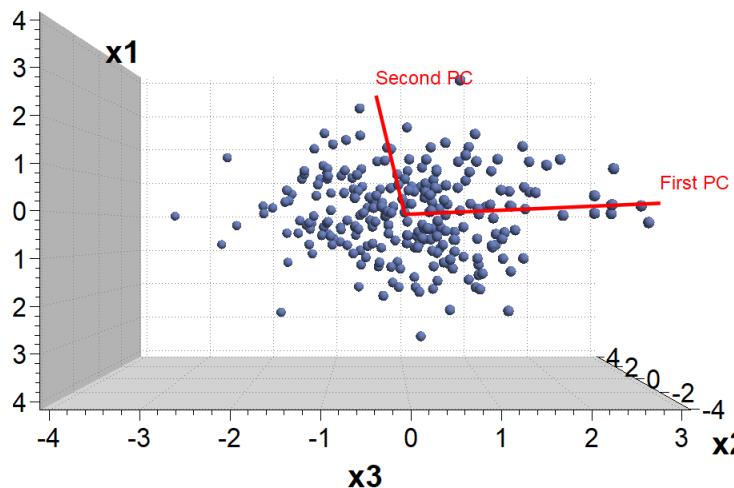
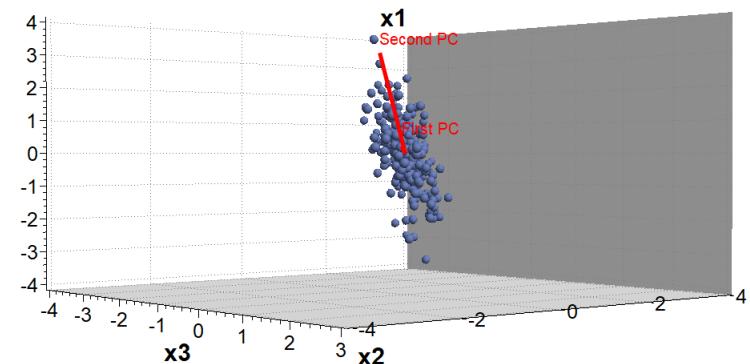
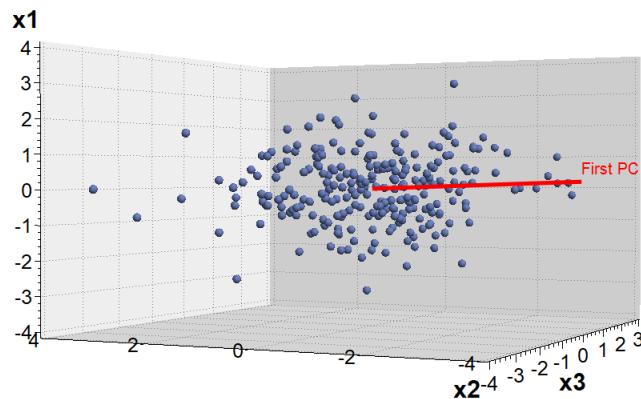


# Суть алгоритма группировки переменных

- Н исходящая иерархическая кластеризация, сначала все переменные в одном кластере
- Затем повторяется в цикле 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

# Поворот главных компонент

- Проблема РСА - много близких к 0 коэффициентов у переменных, плохая интерпретируемость

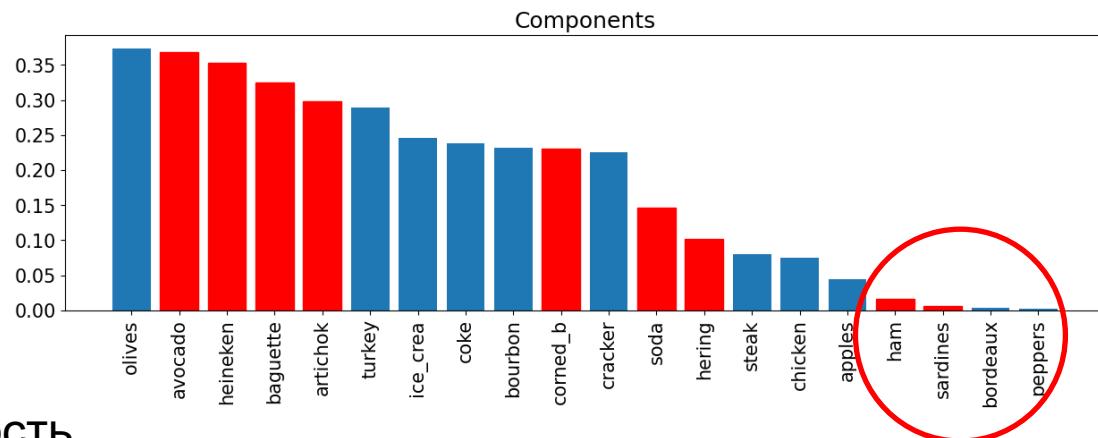
- Цель поворота:

- Перейти к новому максимально близкому базису с большим числом 0 весов (увеличить определенность) и улучшить интерпретируемость

- VARIMAX:

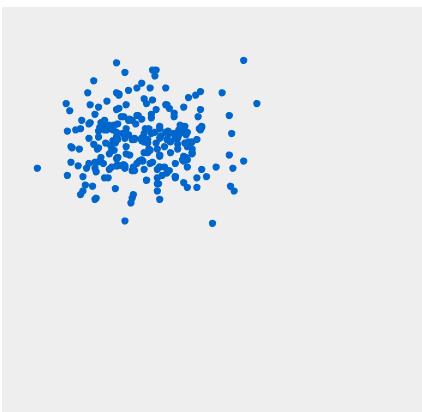
- максимизирует сумму квадратов корреляций между переменными и факторами (гл. компонентами), осуществляя поворот исходной матрицы факторов  $V_{p \times k}$  с помощью матрицы  $R_{VARIMAX}$ , такой что:

$$R_{VARIMAX} = \operatorname{argmax}_R \left( \frac{1}{p} \sum_{j=1}^k \sum_{i=1}^p (VR)^4_{ij} - \sum_{j=1}^k \left( \frac{1}{p} \sum_{i=1}^p (VR)^2_{ij} \right) \right)$$

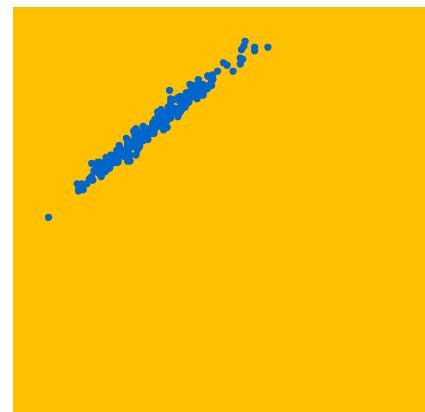


# Перераспределение переменных

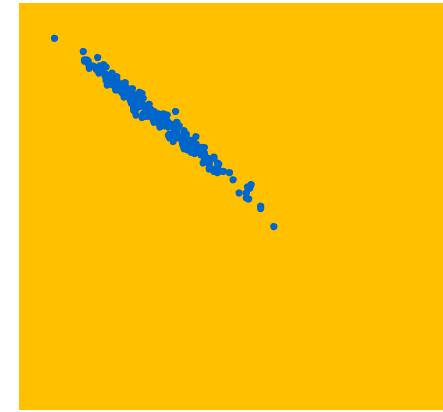
$X_1$



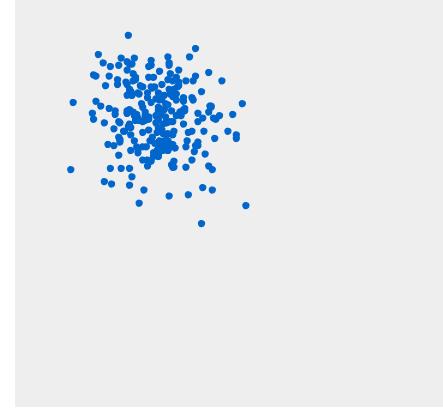
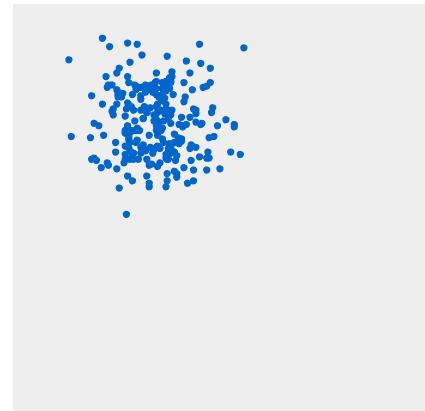
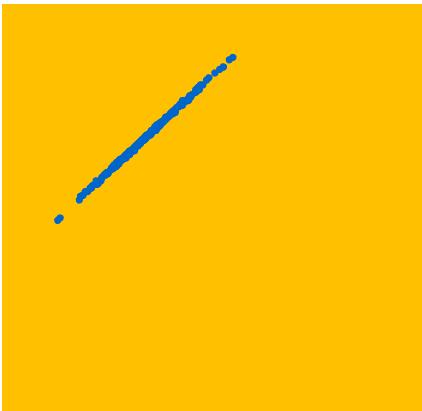
$X_2$



$X_3$

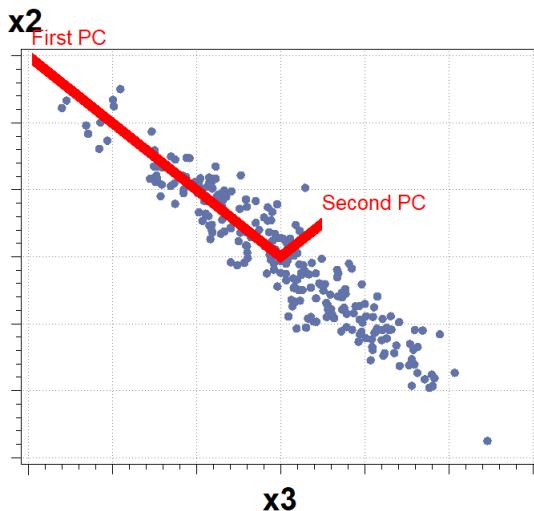


Вторая RC

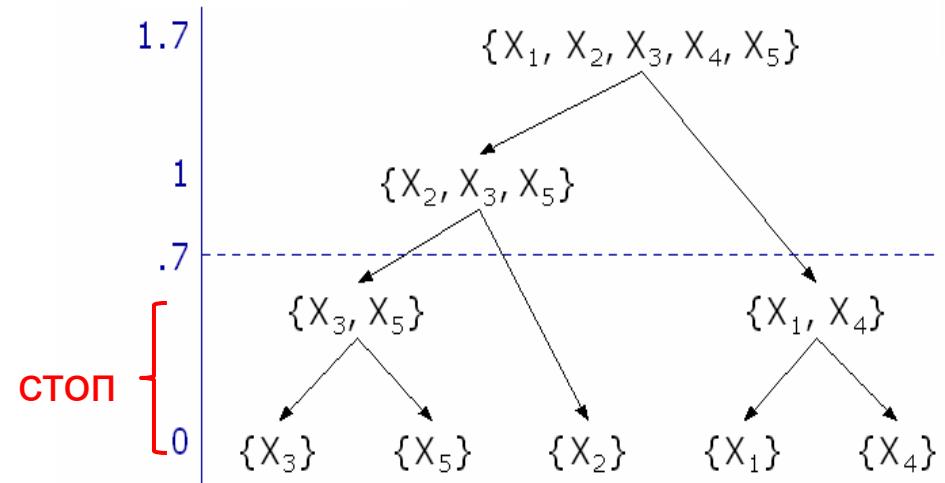


# Разбивать ли дальше и какой кластер выбрать?

Первое с  
Второе с



2<sup>nd</sup> Eigenvalue



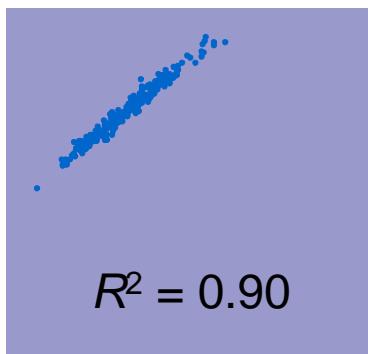
- Варианты критерия остановки:
  - Задать порог на допустимое второе с.зн.
  - Задать максимально допустимое число кластеров
  - Задать необходимый минимум описанной вариации – сумма первых с.зн по всем листьям, деленных на размерность пространства признаков

# Выбор представителей кластеров

$x_2$

РС своей  
группы

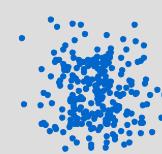
$$R^2 = 0.90$$



РС ближайшей  
чужой группы

$$Ratio_{1-R^2} = \frac{1 - 0.9}{1 - 0.01} = 0.101$$

$x_2$



$$R^2 = 0.01$$

- Лучший представитель это переменная, которая:
  - максимально хорошо объясняет разброс в своей группе – максимально коррелирующая с первым с.в. своего кластера
  - максимально плохо объясняет разброс в любой другой группе – минимально значение максимальной корреляции с первым с.в. по всем остальным кластерам
  - Отношение  $1 - R^2$ :

$$Ratio_{1-R^2} = \frac{1 - R_{\text{own cluster}}^2}{1 - R_{\text{next closest}}^2}$$

# Пример использования

## 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

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

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

clusters.rsquare

clusters.info

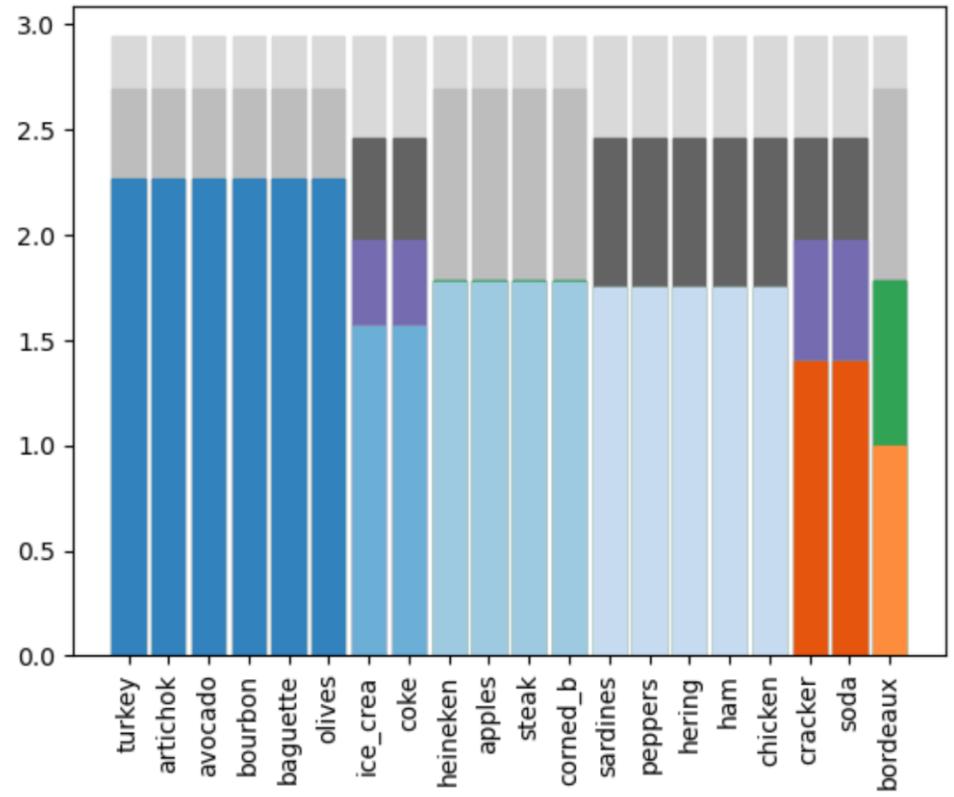
	Cluster	Variable	RS_Own	RS_NC	RS_Ratio
0	0	artichok	0.331882	0.073638	0.721228
1	0	avocado	0.542912	0.093912	0.504463
2	0	baguette	0.325976	0.059756	0.716861
3	0	olives	0.521717	0.110853	0.537912
4	0	turkey	0.269857	0.076499	0.790625
5	0	bourbon	0.277107	0.036053	0.749930
6	1	coke	0.784811	0.083987	0.234919
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

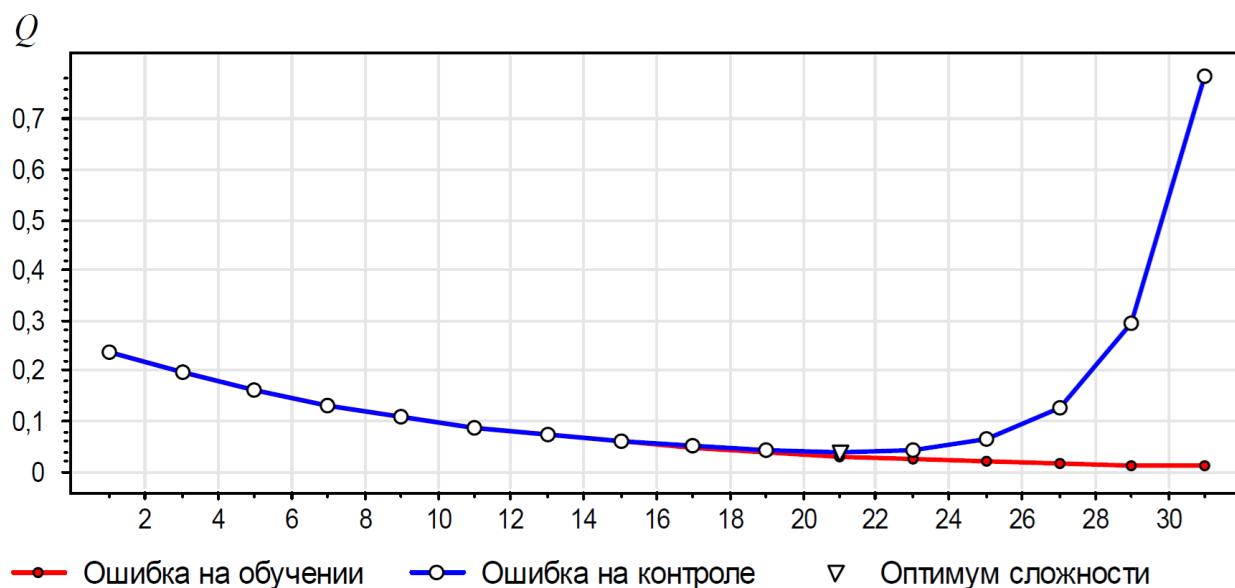
```
# The Library doesn't provide the iterations
# So let's collect them manually
hierarchy = list()
variance_explained = dict()
for i in range(len(clusters.info), 0, -1):
    vch = VarClusHi(assoc, maxeigval2=1, maxclus=i)
    vch.varclus()
    for _, x in vch.clusters.items(): # number, cluster
  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



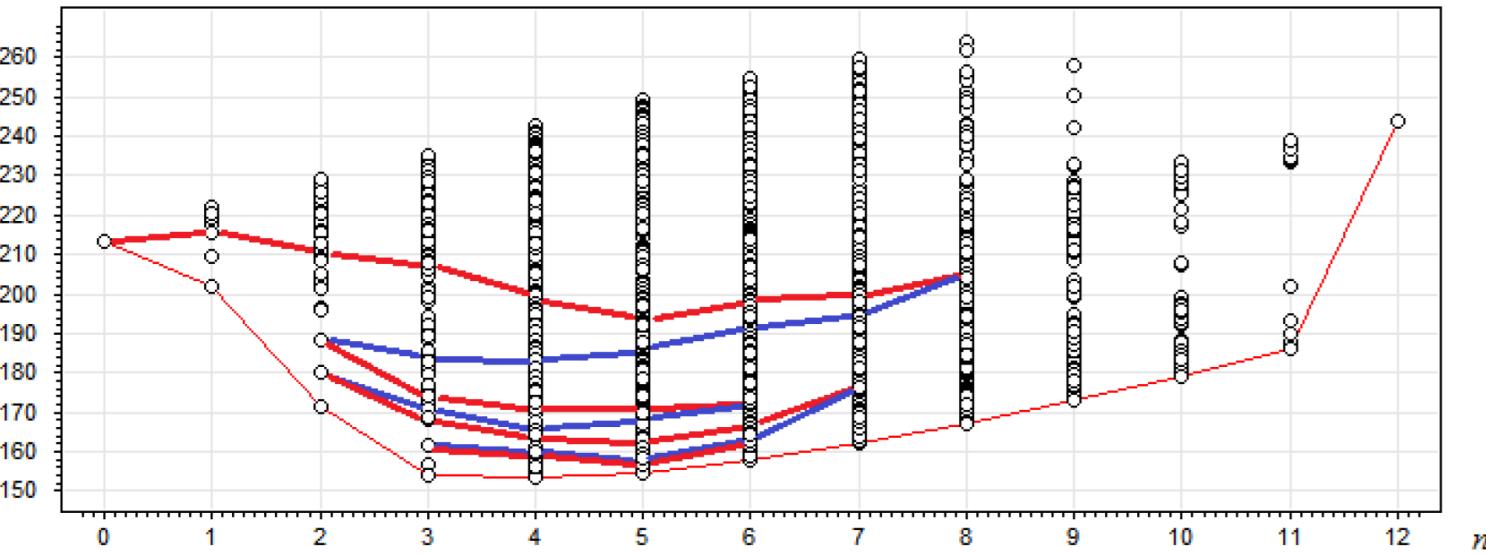
# Пошаговый отбор переменных с учителем - критерии

Внутренний критерий и внешний критерий:



- Примеры критериев:
  - Внешние: кросс-валидационная и валидационная ошибка, информационные критерии AIC и BIC, статистические критерии, учитывающие число степеней свободы модели
  - Внутренние: эмпирический риск, статистики отклонения, MSE, MAE на тренировочном наборе

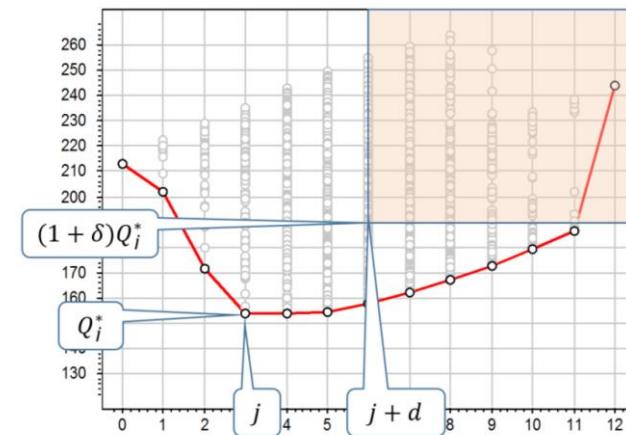
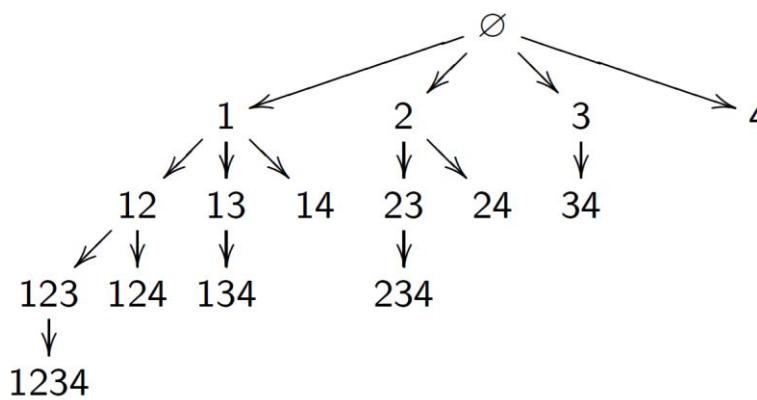
# Пошаговый отбор переменных с учителем - стратегии



- Ранее рассмотренные пошаговые методы:
  - прямые (от простого к сложному), обратные (от сложного к простому), комбинированные – шаг(и) вперед и шаг(и) назад, SWAP пары переменных
- Что еще бывает?
  - Поиск в ширину и глубину, эволюционные алгоритмы, случайный поиск

# Направленный перебор «в глубину»

- Дерево перебора признаков – от простых моделей к сложным:

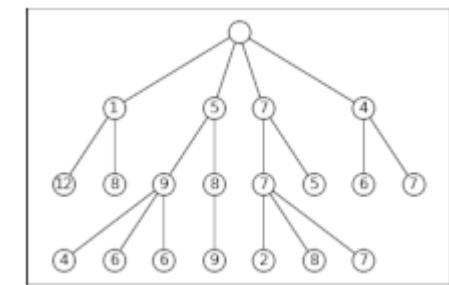


- Метод ветвей и границ – поиск «в глубину» с отсечением малоперспективных путей:
  - Пусть  $Q_j^*$ - лучшая уже найденная модель сложности  $j$ , тогда ветка с набором  $J$  отсекается, если найдется такой  $j$ , что  $Q(J) \geq (1 - \delta)Q_j^*$  и  $|J| \geq j + d$ , где  $\delta$  и  $d$  – параметры
  - При шаге в глубину фильтровать (сортировать) переменные по важности для ускорения поиска
  - Все равно как правило сохраняется экспоненциальная сложность

# Усеченный поиск в ширину (beam search)

## ■ Основной принцип «неокончательных решений»:

- На каждом шаге оставлять максимальную свободу выбора принятия решений
- На каждой  $j$ -ой итерации строить не один набор, а множество из  $B_j$  наборов  $R_j = \{J_j^1, \dots, J_j^{B_j}\}$ , где  $|J_j^b| = j$ ,  $b = 1, \dots, B_j$ ,  $B_j \leq B$  – параметр ширины поиска



## ■ Алгоритм (итерации по $j$ – сложность наборов)

- Инициализация:  $R_1$  - одноэлементные наборы,  $Q^* = Q(\emptyset)$
- Для всех  $j = 1, \dots, n$ 
  1. Отсортировать  $R_j = \{J_j^1, \dots, J_j^{B_j}\}$  по возрастанию критерия  $Q(J_j)$
  2. Если  $B_j > B$ , то оставить  $B$  лучших наборов в  $R_j$
  3. Если есть улучшение  $Q(J_j^1) < Q^*$ , то запомнить  $j^* = j$ ,  $Q^* = Q(J_j^1)$
  4. Если  $j - j^* \geq d$  то вернуть  $J_{j^*}^1$
  5. Породить следующий набор:  $R_j = \{J \cup \{X_i\} | J \in R_{j-1}, X_i \in X \setminus J\}$

# Пошаговый отбор без учителя

- Можно реализовать пошаговые стратегии отбора (как с учителем), но в качестве критерия брать не долю описанной вариации отклика, а долю описанной вариации не включенных в модель переменных:

- $X_{in}$  - множество переменных, включенных в модель
  - $X_{out}$  - множество переменных, не включенных в модель
  - критерий для оценки шага:

$$tr \left( X_{out}^T \left( I - X_{in} (X_{in}^T X_{in})^{-1} X_{in}^T \right) X_{out} \right) \rightarrow \min$$

- похоже на линейную регрессию с матрицей проекции  $H = X_{in} (X_{in}^T X_{in})^{-1} X_{in}^T$ , но проецируем не отклик, а не включенные переменные  $X_{out}$ , т.е. оценивается как  $X_{in}$  описывают не вариацию отклика, а вариацию оставшихся переменных
  - Критерий «внутренний» – не учитывает сложность модели, поэтому обычно делается пошаговый перебор с ограничением на число включенных переменных, а потом весь путь проверяется по внешнему критерию

# Пример

$\mathbf{C}_{11} = \mathbf{X}_1^\top \mathbf{X}_1$ ,  $\mathbf{C}_{12} = \mathbf{X}_1^\top \mathbf{X}_2$ , and  $\mathbf{C}_{21} = \mathbf{X}_2^\top \mathbf{X}_1$ .

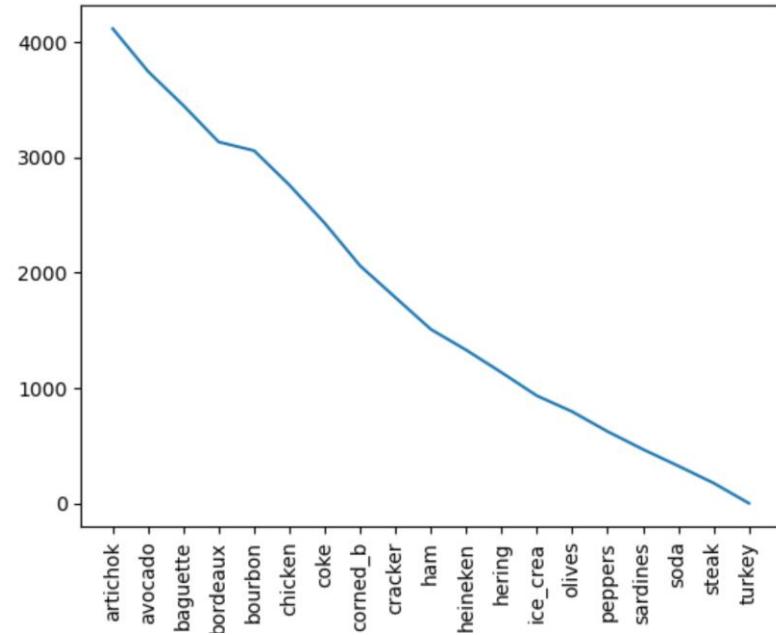
$$\mathbf{C} = \mathbf{X}^\top \mathbf{X} = \begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix}$$

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

```
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
```



# Отбор признаков в линейных моделях с LASSO

- При обучении линейных моделей с учителем:
  - регуляризация L1 и Elastic Net «отбирает» признаки
- А можно ли без учителя? Да – Graphical LASSO (пусть  $x_i \sim N(0, \Sigma)$ ):

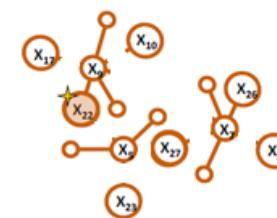
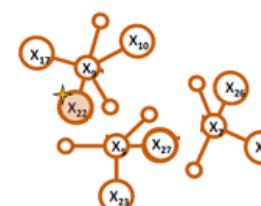
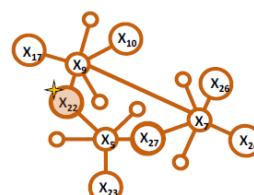
$$\widehat{\Theta} = \operatorname{argmin}_{\Theta \geq 0} \left( \text{tr}(S\Theta) - \log|\Theta| + \lambda \sum_{j \neq k} |\Theta_{jk}| \right)$$

- $\widehat{\Theta}$  - обратная ковариационная матрица, где  $S$  - наблюдаемая ковариационная матрицы,  $\lambda$  - параметр регуляризации, чем он больше, тем  $\widehat{\Theta}$  более разрежена и меньше дуг в графе связей переменных :

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

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

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



# Пример

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

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

Регуляризация

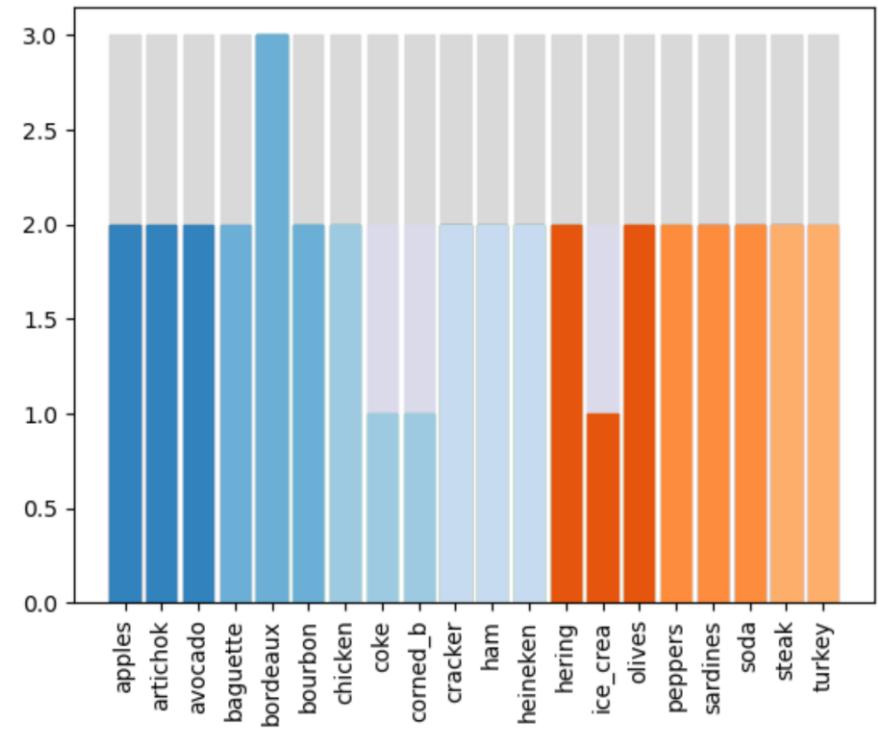
PRODUCT	apples	artichok	avocado	baguette	bordeaux	bourbon	chicken	coke	corned_b
PRODUCT									
apples	0.227114	-0.021079	0.015019	0.018002	0.000000	-0.029258	-0.022309	-0.011946	0.025641
artichok	-0.021079	0.226149	0.087879	0.024196	0.000000	-0.036957	-0.014775	-0.028545	-0.032747
avocado	0.015019	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

```
from scipy.sparse.csgraph import connected_components

L = 0.001
RATE = 10
STEPS = 4
hierarchy = list()
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
```

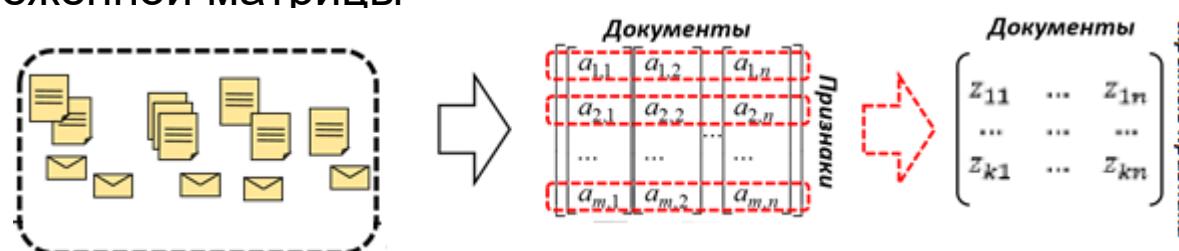
Восходящая кластеризация переменных -  
последовательный поиск обратной  
ковариационной матрицы с домножением  
параметра регуляризации при каждом шаге



По вертикали –  
число шагов.

# Линейные преобразования пространства признаков

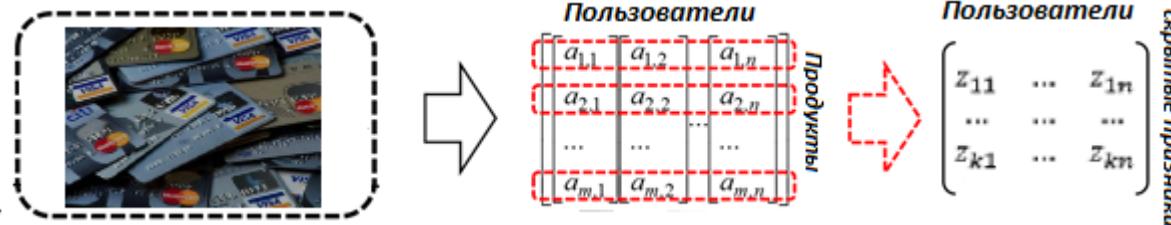
- Главные компоненты с учителем - PLS и LDA, без учителя – PCA и SVD.
- Чем плохи? Для многих прикладных задач не интерпретируемы из-за отрицательных весов, пропуски нужно подставлять заранее, ортогональность скрытых признаков (не всегда нужно)
- Примеры задач:
  - Тематическое моделирование, где  $z_1 = F_1(a_1, \dots, a_m), \dots, z_k = F_k(a_1, \dots, a_m)$ , где  $k < n$  - скрытые признаки документа или веса тематик. 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 представим в виде разреженной матрицы



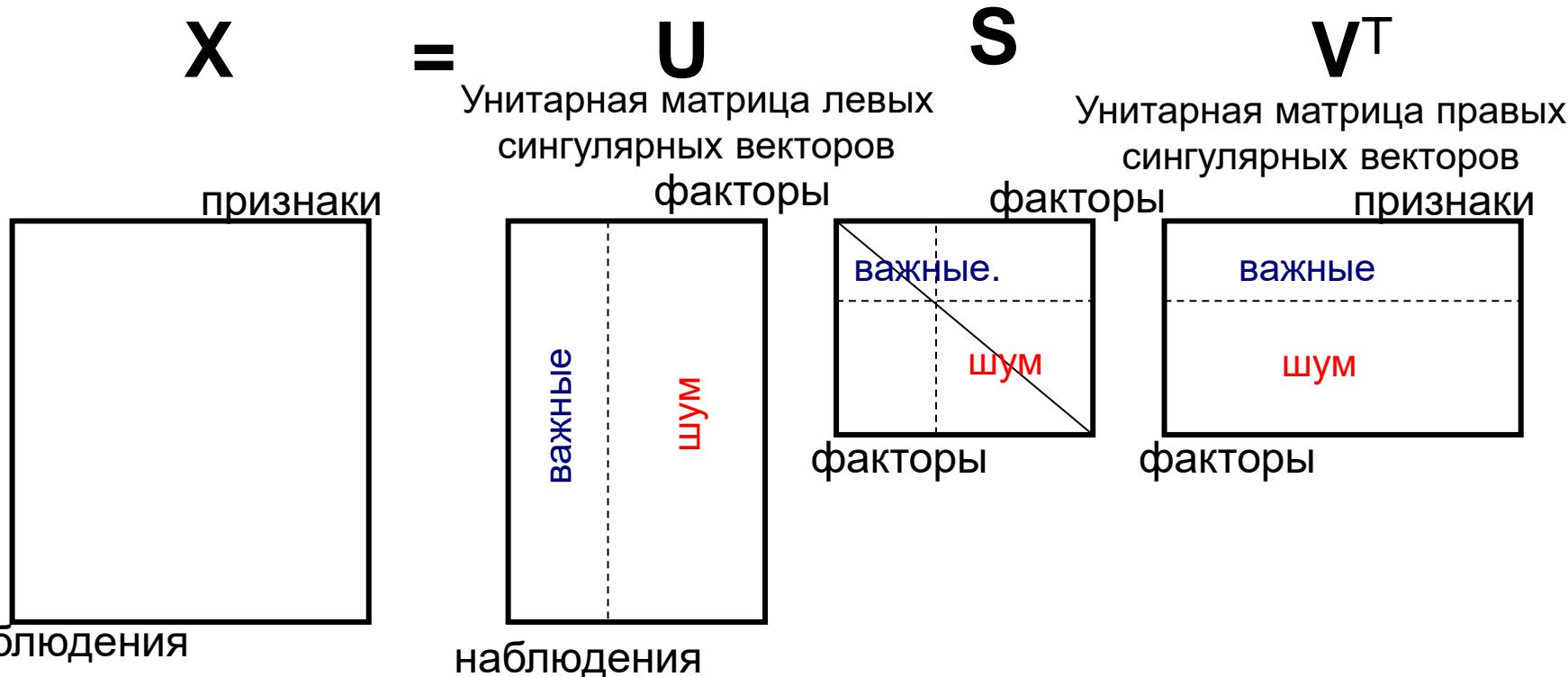
# Линейные преобразования пространства признаков

## ■ Рекомендательные системы.

- Дано: множество пользователей (клиентов)  $U$ , множество продуктов (услуг, ресурсов и т.д.)  $I$ , множество транзакций <время, пользователь, услуга, деньги, количество покупок и т.д.>
- Задача: построить модель, которая по истории пользователя будет предсказывать какие его заинтересуют.
- Транзакционная история представляется в виде матрицы, где  $a_{ij}$ , например, равно числу транзакций пользователя  $j$  по использованию продукта  $i$  (или сумме покупок, или среднему чеку и т.д.).  $z_1 = F_1(a_1, \dots, a_m), \dots, z_k = F_k(a_1, \dots, a_m)$ , где  $k << n$  - скрытые предпочтения пользователя или скрытые характеристики продуктов.



# Сингулярное разложение



## 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

# Пример использования РСА

## Пример:

- Преобразуем транзакционный набор ASSOC в матрицу клиент-продукт:

```
import pandas as pd
```

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

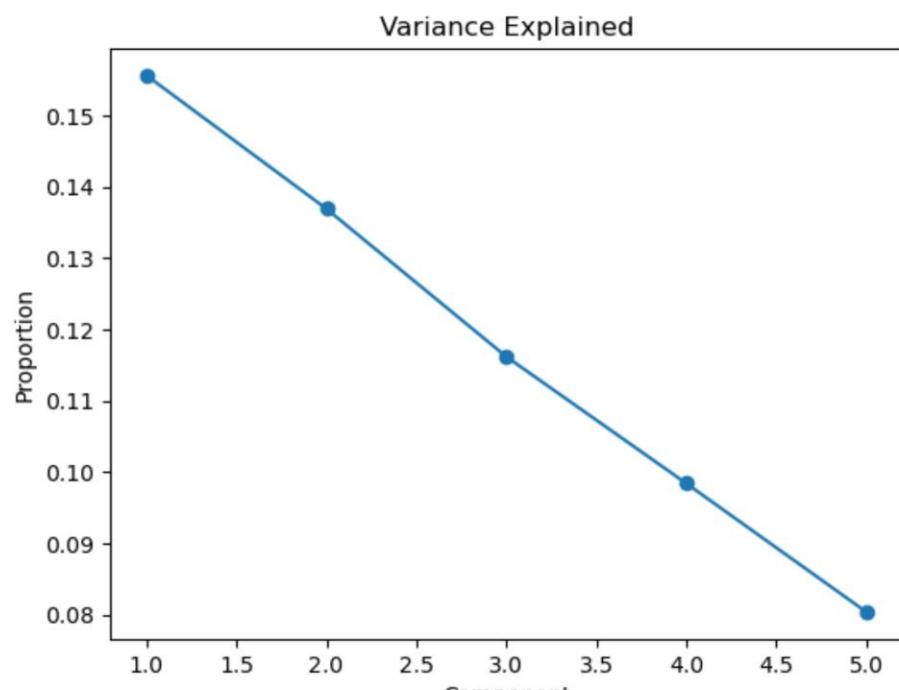
```
assoc = assoc.groupby("CUSTOMER")["PRODUCT"].value_counts().unstack(fill_value=0)
assoc
```

# Пример использования PCA

```
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)
```

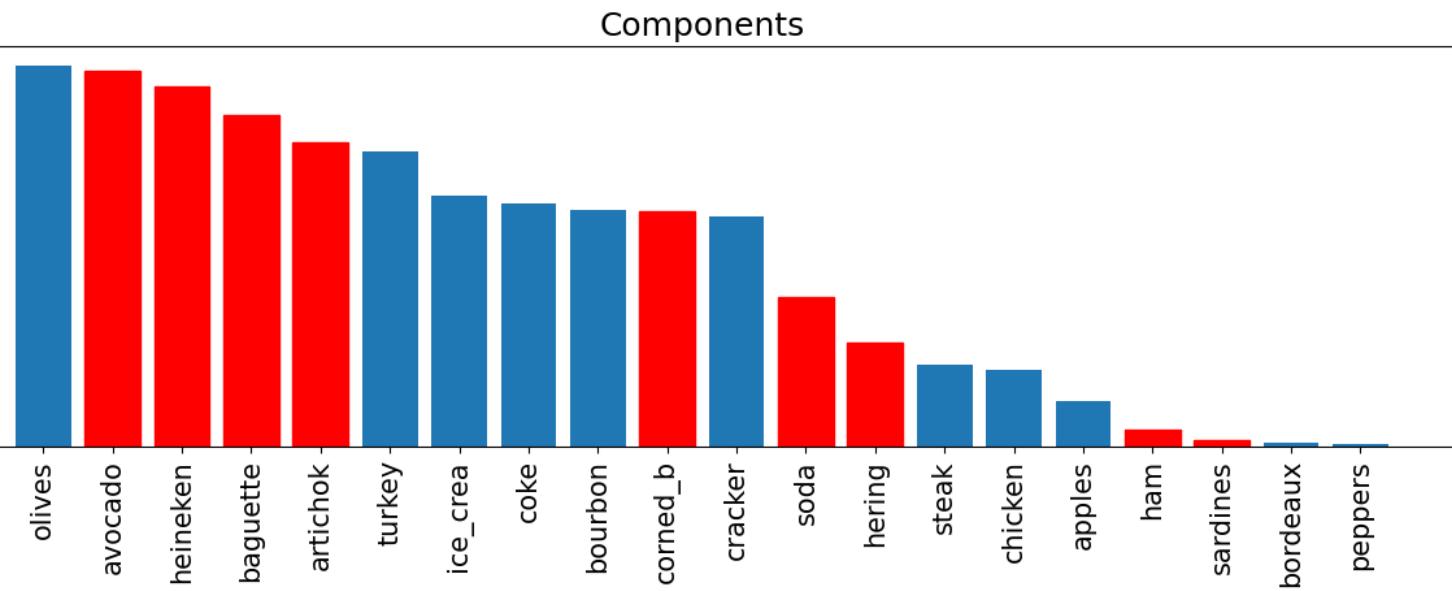
```
plt.plot(range(1, N+1), pca.explained_variance_ratio_)
plt.scatter(range(1, N+1), pca.explained_variance_ratio_)
plt.xlabel("Component")
plt.ylabel("Proportion")
plt.title("Variance Explained")
pass
```



# Пример использования РСА

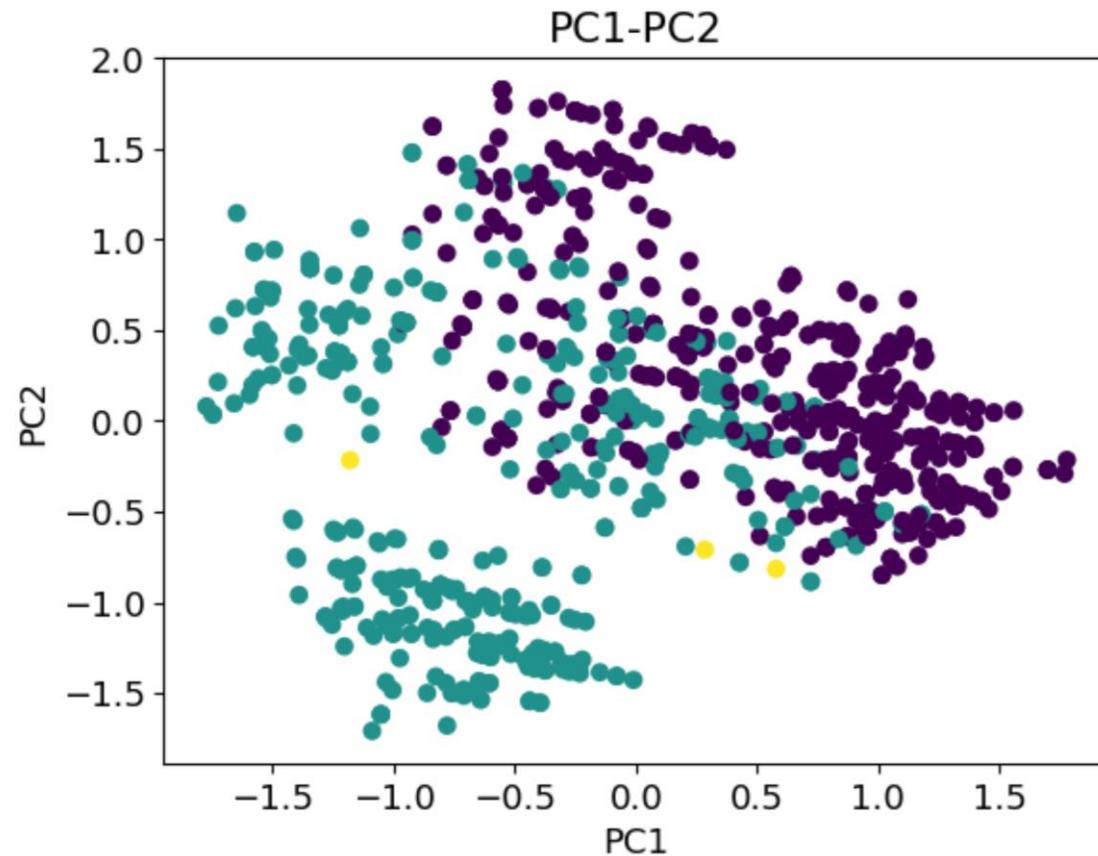
```
components = pca.components_ # [Число компонент, Число переменных]
sample = components[0] # Для примера возьмем первую
order = np.argsort(-abs(sample))
```

```
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 PCA

```
# 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 PCA

## ■ 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  $\langle u, v \rangle * \lambda$

# Неотрицательность и разреженность важны для многих задач



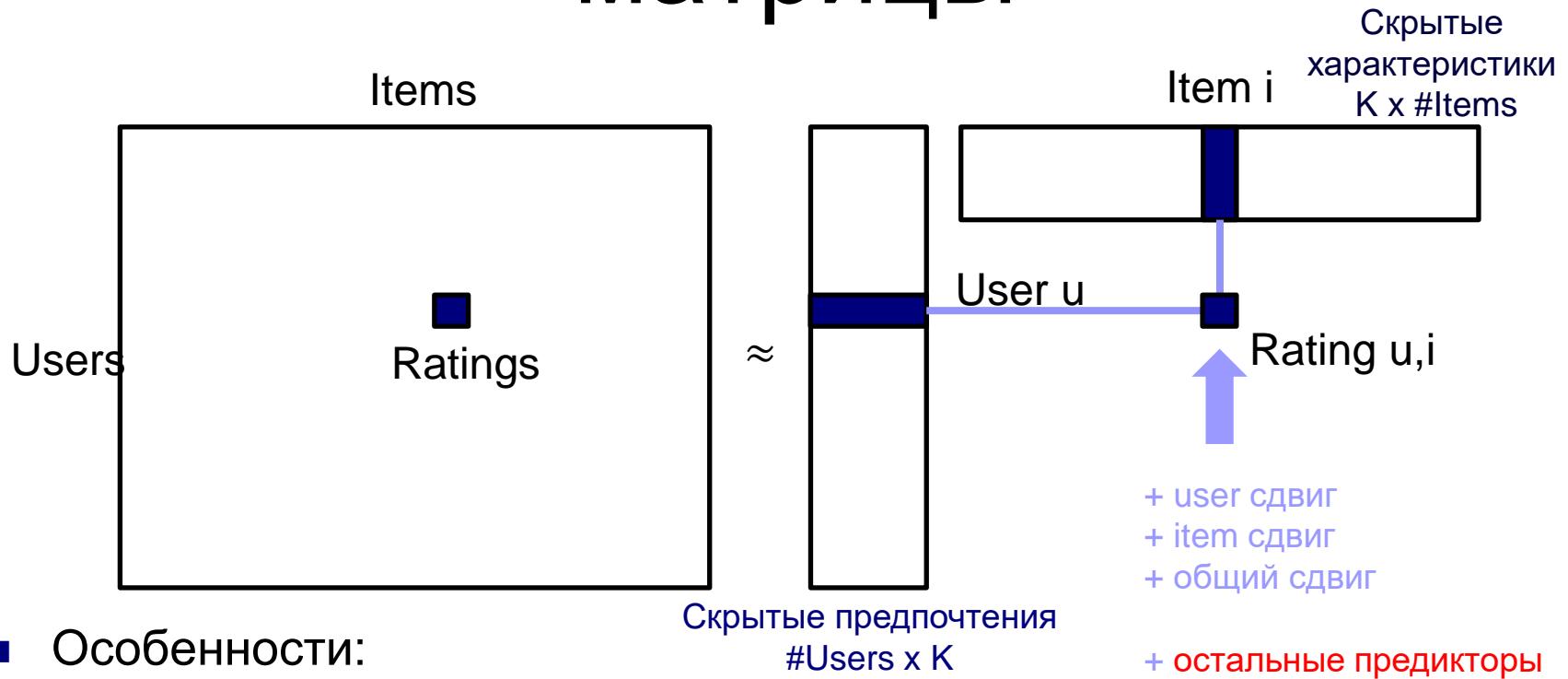
	I1	I2	I3	I4	I5
U1	5	-	4	-	3
U2	2	-	-	4	5
U3	-	4	3	3	-
U4	1	2	1	-	-
U5	1	3	4	-	5
U6	4	3	-	5	4
U7	-	5	4	-	3
U8	3	-	-	4	5
U9	-	3	2	1	1
U10	5	4	-	3	-

Разреженные данные (<1%)

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

Предсказать рейтинг  $i$  от  $u$ ?

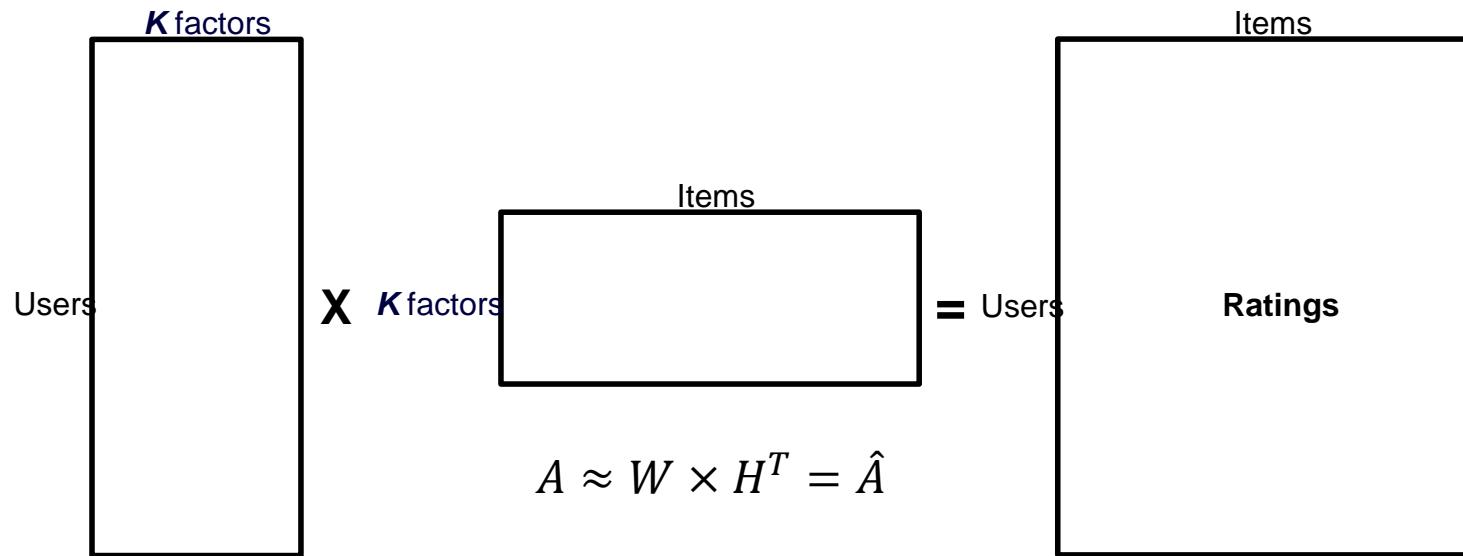
# Факторизация разреженной матрицы



## ■ Особенности:

- Пропуски могут быть не нули (не учитываем их при расчете ошибки)
- Можно регуляризовать (штрафовать по норме) матрицы, чтобы не было переобучения в пустых ячейках
- Есть маргинальные «сдвиги» пользователя и продукта, которые можно учесть вычитанием из ячейки, а также дополнительные предикторы

# Восстановление как прогноз



	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	.



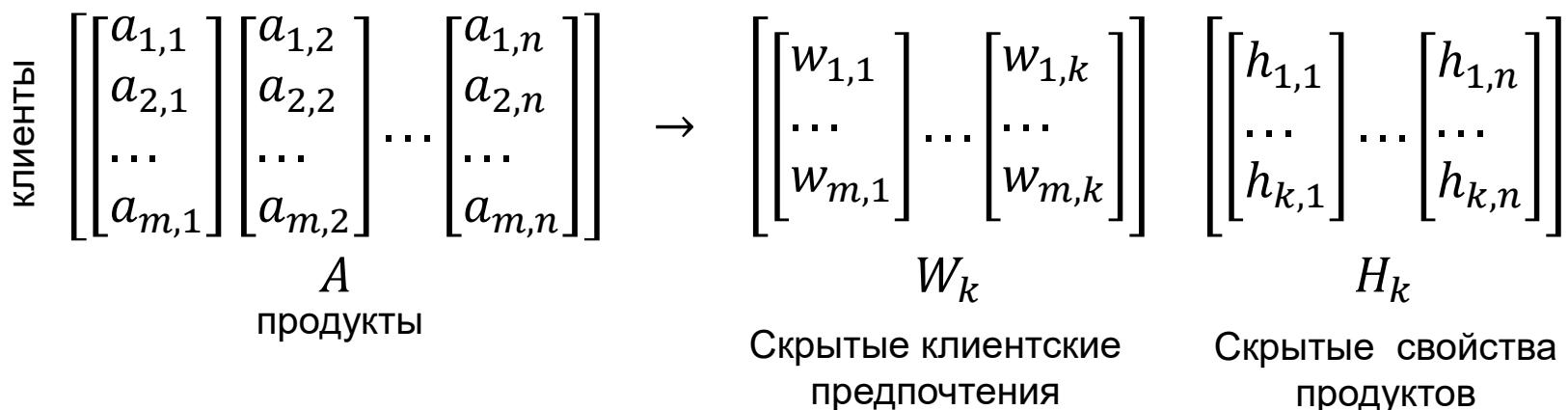
	user	item	rating	P_rating
1	U1	I1	5	0.7230614009
2	U1	I2	.	1.434936794
3	U1	I3	4	2.9436770371
4	U1	I4	.	2.7834737905
5	U1	I5	3	2.188975178
6	U2	I1	2	2.3086243215
7	U2	I2	.	4.7809997821
8	U2	I3	.	1.0679811332
9	U2	I4	4	3.4404221789
10	U2	I5	5	5.8208743099
11	U3	I1	.	4.166851086
12	U3	I2	4	4.88791012
13	U3	I3	3	3.0102871449
14	U3	I4	3	2.2489738265
15	U3	I5	.	4.29822004

# Неотрицательная матричная факторизация

- Дано: разреженная матрица  $A \in \mathbb{R}_+^{m \times n}$
- Цель неотрицательной матричной факторизации состоит в:
  - нахождении матриц  $W_k \in \mathbb{R}_+^{m \times k}$ ,  $H_k \in \mathbb{R}_+^{k \times n}$  с неотрицательными элементами, которые минимизируют отклонение по некоторой норме (например, Фробениуса) исходной матрицы от восстановленной:

$$\frac{1}{2} \|A - W_k H_k\|_F^2 \rightarrow \min_{W_k, H_k}$$

где задано число скрытых факторов  $k \ll \min(m, n)$



# Мультипликативный алгоритм (НМФ)

- Из необходимых условий локального минимума можно вывести:
  - инициализируются случайными неотрицательными числами

$$W_k^{(0)} \in \mathbb{R}_+^{m \times k}, H_k^{(0)} \in \mathbb{R}_+^{k \times n}$$

- В цикле  $t$  раз выполняются вычисления матриц:

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

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

- Можно добавить штраф за не ортогональность любой из матриц:

$$\frac{1}{2} \|A - W_k H_k\|_F^2 + \frac{\alpha}{2} \|W_k^T W_k - I\|_F^2 \rightarrow \min$$

- тогда изменится пересчет соответствующей матрицы:

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

# 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

PRODUCT	apples	artichok	avocado	baguette	bordeaux	bourbon	chicken	coke	corned_b	cracker	ham	heinel
CUSTOMER												
0.0	0	0	0	0	0	1	0	0	1	0	1	
1.0	0	0	0	1	0	0	0	0	1	1	0	
2.0	0	1	1	0	0	0	0	0	0	1	1	
3.0	0	0	0	0	0	1	0	1	0	0	1	
4.0	1	0	1	0	0	0	0	0	1	0	0	

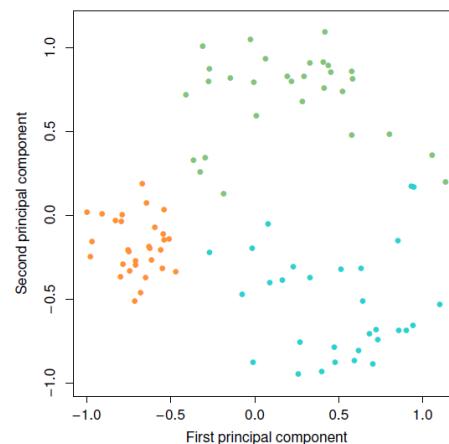
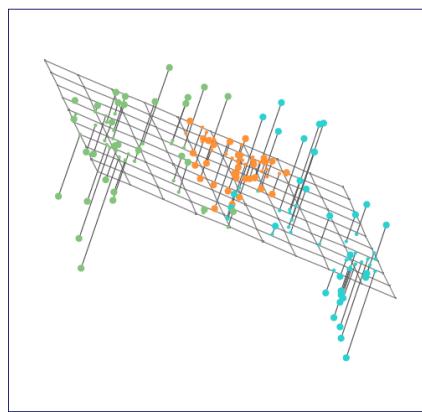
# Другие варианты разложений

- Если в NMF взять  $A \in \{0,1\}^{m \times n}$  и вместо  $\|\cdot\|_F^2$  использовать дивергенцию Кульбака-Лейблера:
  - то получим NMF как вероятностный семантический анализ (PLSA):
$$\sum_{(u,i)} a_{ui} \ln \sum_k w_{ik} h_{ku} \rightarrow \max_{W,H} , w_{ik} = P(i|k), h_{ku} = P(k|u)$$
  - EM алгоритм (результат Е-шага  $P(k|i, u)$  встроен сразу в М-шаг):
$$h_{ku} = \text{norm}_k \left( \sum_{i \in I} a_{ui} \left( \frac{w_{ik} h_{ku}}{(HW)_{ui}} \right) \right), w_{ik} = \text{norm}_i \left( \sum_{i \in I} a_{ui} \left( \frac{w_{ik} h_{ku}}{(HW)_{ui}} \right) \right)$$
- Если добавить условие ортогональности (взаимоисключения принадлежности)  $HH^T = I$ , то получим кластеризацию k-средних
- Можно учитывать смещения и доп. признаки (но уже не NMF, решаем с помощью SGD):

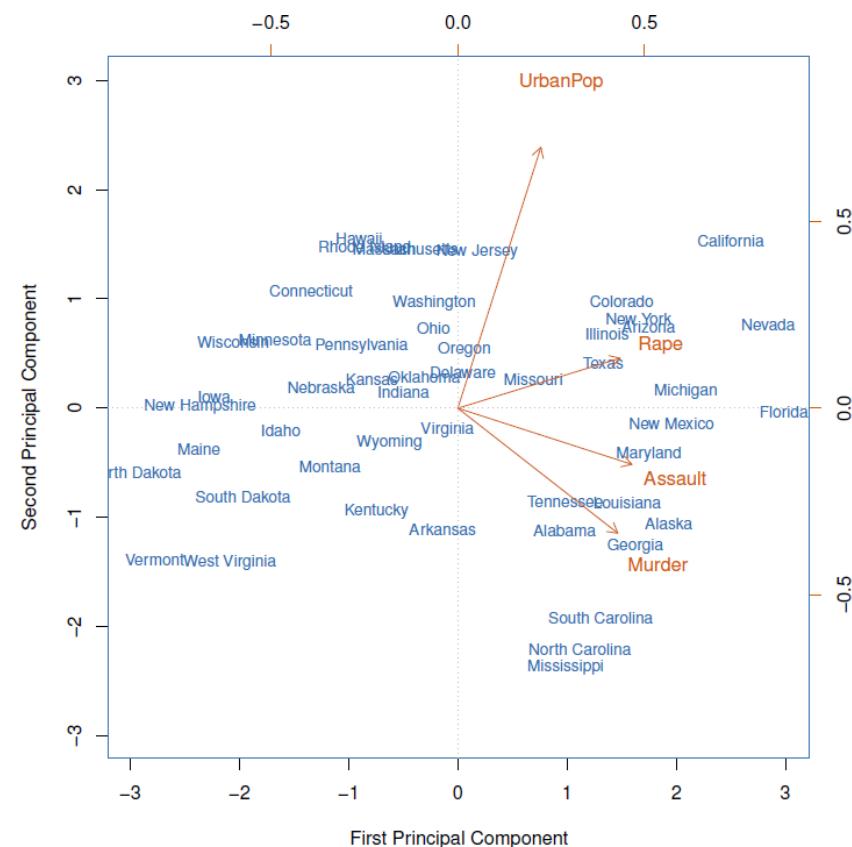
$$\sum_{(u,i)} \left( a_{ui} - s - s_u - s_i - \sum_k w_{ik} h_{ku} \right) \rightarrow \min_{W,H}$$

# Визуализация с РСА

- Одно из важнейших применений – проекция множества наблюдений на пары главных компонент.
- Основной недостаток – линейность.



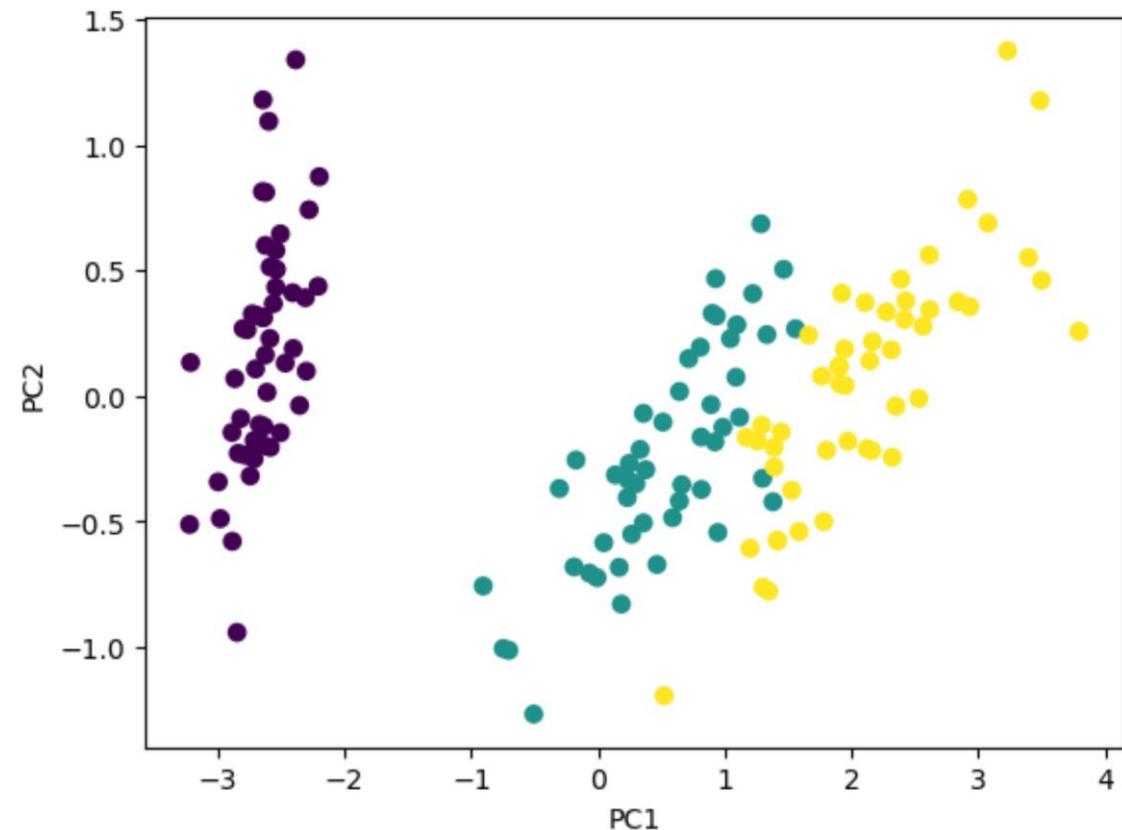
	PC1	PC2
Murder	0.5358995	-0.4181809
Assault	0.5831836	-0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186



# Пример 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 РСА



# Kernel Trick

- Преобразование исходного векторного пространства признаков с помощью отображения  $\varphi: \mathbb{R}^d \mapsto H, x \mapsto \varphi(x)$ :
  - где скалярное произведение  $K_H(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle$
  - НО в явном виде не нужно вычислять новые признаки и образы, достаточно заменить скалярное произведение на ядерную функцию
- Ядерный РСА:
  - ищем с.зн. и с.в. ковариационной матрицы не в  $X$ , а в  $H$ : вместо  $C_X = \frac{1}{l} \sum \langle x_i, x_j \rangle$ , где  $x$  - центрированы, а  $C_H = \frac{1}{l} \sum \langle \varphi(x_i), \varphi(x_j) \rangle$ , где  $\varphi(x)$  тоже центрированы
  - тогда  $\alpha$  собственные вектора центрированной матрицы ядер  $\tilde{K}$ :

$$\lambda \alpha = \tilde{K}, \text{ где } \tilde{K} = K - 1_l K - K 1_l + 1_l K 1_l,$$

$1_l$  - матрица  $l \times l$  со всеми значениями  $1/l$ ,

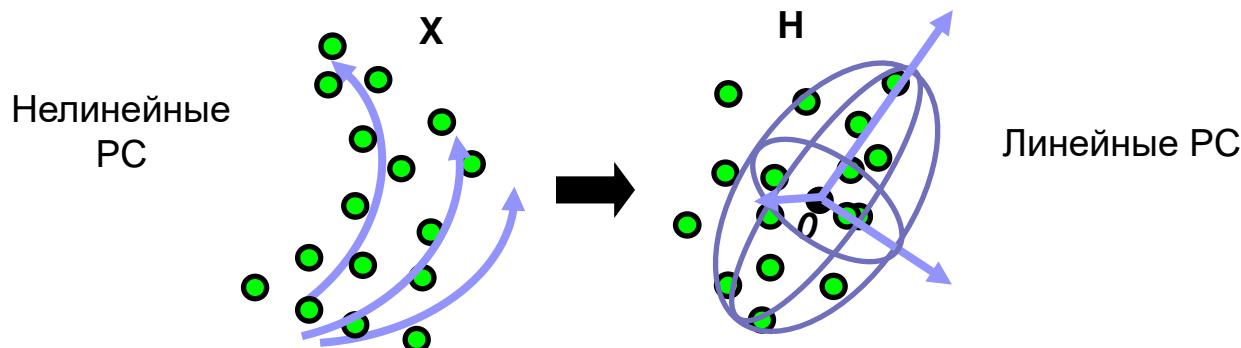
$\lambda$  - с.зн.,  $l$  – размер выборки.

# Ядерный РСА

- В результате:

- в  $H$  строится гиперэллипсоид с центром в начале координат, содержащий основную часть образов наблюдений  $\varphi(x)$  и с осями, ориентированными в направлении наибольшего разброса образов
- главные компоненты линейны в  $H$  и нелинейны в  $X$ , не выражаются в виде вектора, но выражаются как линейная комбинация ядер, где проекция образа  $\varphi(x)$  на  $k$ -ю компоненту  $V^k$ :

$$z_k(x) = V^{kT} \varphi(x) = \left( \sum_{i=1}^l a_i^k \varphi(x_i) \right)^T \varphi(x) = \sum_{i=1}^l a_i^k K(x_i, x)$$



# Свойства КРСА

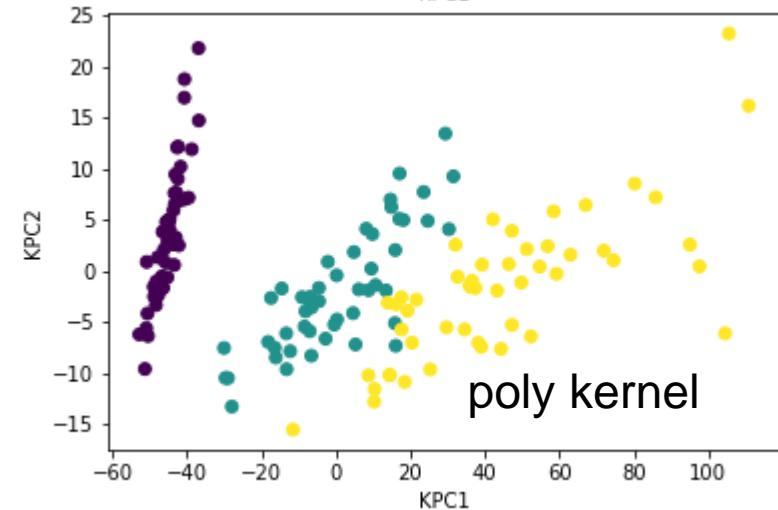
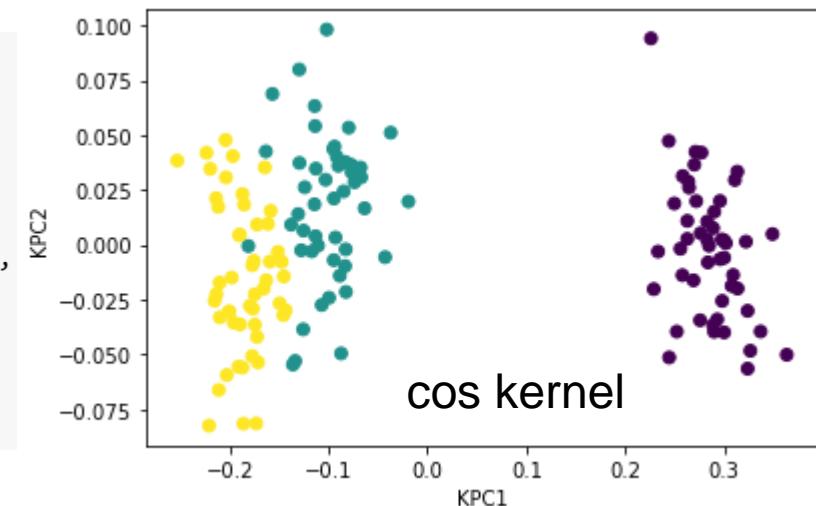
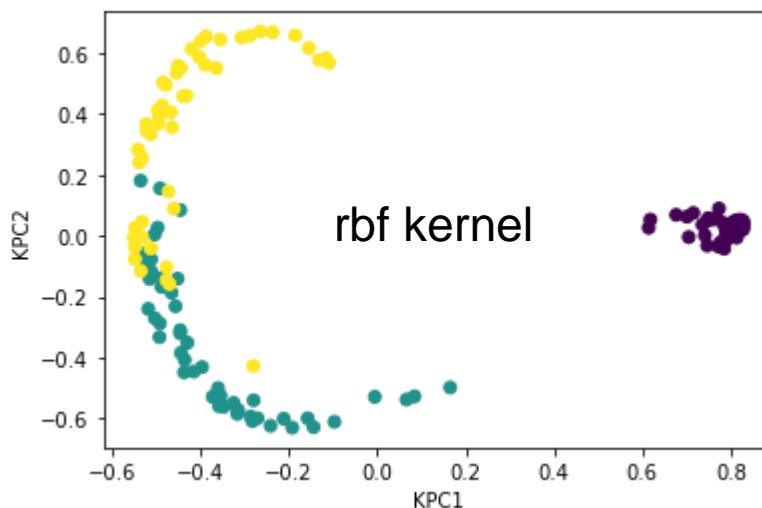
- Ошибка реконструкции (аномальность) наблюдения при проекции на  $q$  главных компонент:

$$\begin{aligned} Err(x) &= \\ &= K(x, x) - \frac{2}{l} \sum_{i=1}^l K(x, x_i) + \\ &+ \frac{1}{l^2} \sum_{i,j=1}^l K(x_i, x_j) - \sum_{k=1}^q \left( \sum_{i=1}^l a_i^k (K(x, x_i) - \frac{1}{l} \sum_{r=1}^l K(x_i, x_r) - \right. \\ &\quad \left. - \frac{1}{l} \sum_{r=1}^l K(x, x_r) + \frac{1}{l^2} \sum_{r,s=1}^l K(x_r, x_s)) \right)^2 \end{aligned}$$

- где  $l$  – размер выборки,  $a_i^k$  коэф.  $k$ -го КРС,  $q$  – число главных компонент
- Ключевые особенности:
  - Главные компоненты не коррелированы
  - Первые  $q$  главных компонент с гауссовским ядром имеют максимальную взаимную информацию с входными признаками

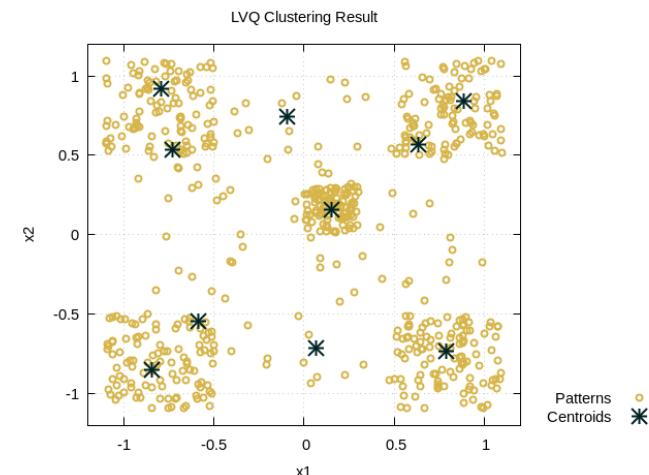
# Пример Kernel PCA с разными ядрами

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import KernelPCA
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
kernel_pca = KernelPCA(n_components=2, kernel="cosine",
                       fit_inverse_transform=True)
features=kernel_pca.fit_transform(X)
plt.scatter(features[:,0],features[:,1],c=y)
plt.xlabel("KPC1")
plt.ylabel("KPC2")
```

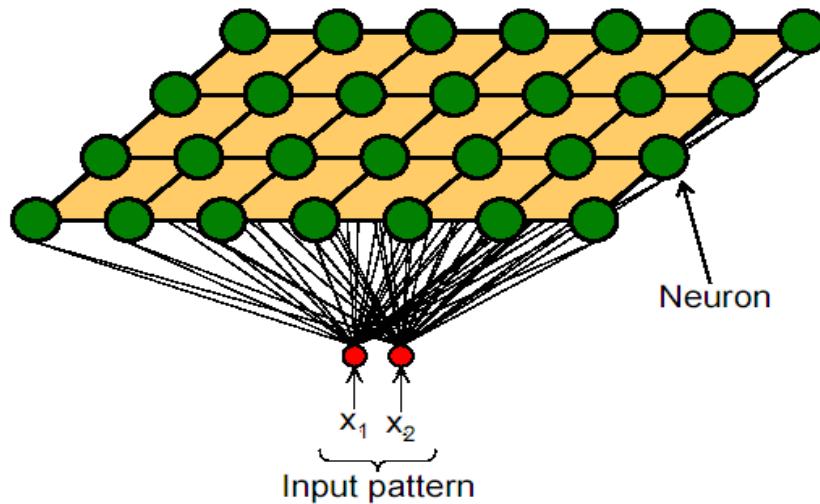


# Постановка задачи кластеризации и квантизации

- Дано:
  - $X^l = \{x_i\}_{i=1}^l$  – обучающая выборка объектов,  $x_i \in \mathbb{R}^n$
  - $\rho^2(x, w) = \|x - w\|^2$  – евклидова метрика в  $\mathbb{R}^n$
- Найти:
  - Центры кластеров  $w_y \in \mathbb{R}^n, y \in Y$ ; алгоритм кластеризации «правило жесткой конкуренции» (WTA, Winner Takes ALL):
$$a(x) = \arg \min_{y \in Y} \rho(x, w_y)$$
- Критерий:
  - среднее внутрикластерное расстояние
$$Q(w; X^l) = \sum_{i=1}^l \rho^2(x_i, w_{a(x_i)}) \rightarrow \min_{w_y: y \in Y}$$
- Квантизация данных:
  - замена  $x_i$  на ближайший центр  $w_{a(x_i)}$



# 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).

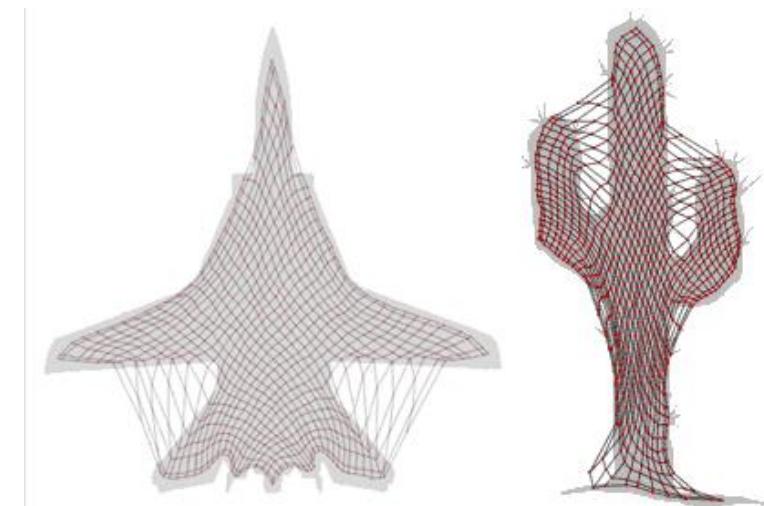
# Процедура работы SOM (неформально)

## ■ Суть метода:

- 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 набора
- Шаг 2. Конкуренция:
  - Находим «лучший» нейрон для активации
$$i(x) = \operatorname{argmin}_j \|x(t) - w_j(t)\|$$
- 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 достижение порога числа итерации

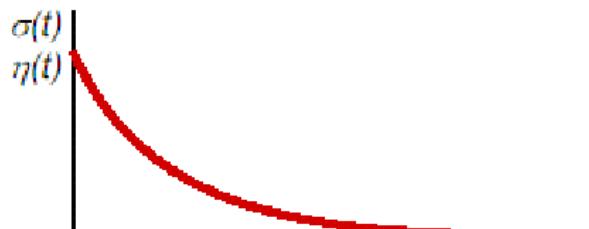
# Коррекция весов с учетом кооперации

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

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

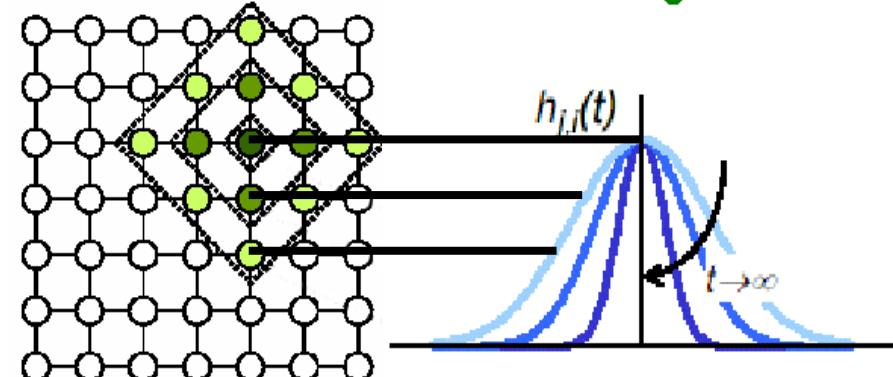
скорость обучения

размер топологической  
окрестности на решетке!

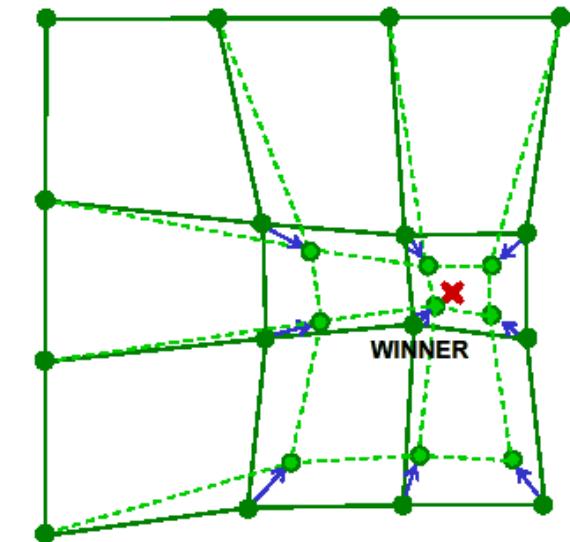


$$\sigma(t+1) = \sigma_0 \exp\left(-\frac{t}{\tau_\sigma}\right)$$

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



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



# 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

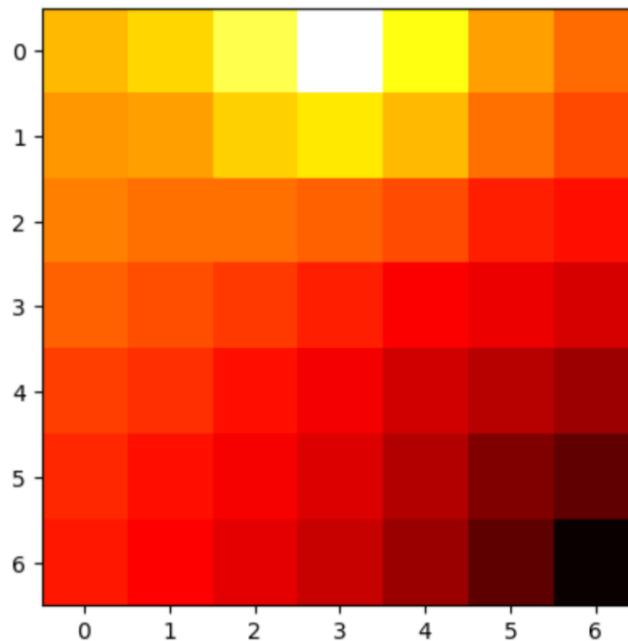
## ■ SOM(10X10):



# Визуализация в виде теплокарт

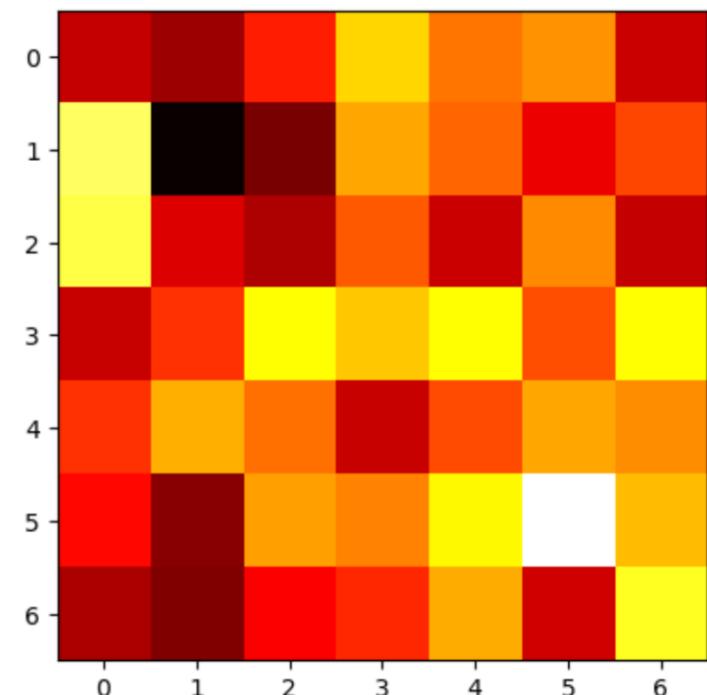
Распределение переменной  
(карта интенсивности)

```
# 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')
```



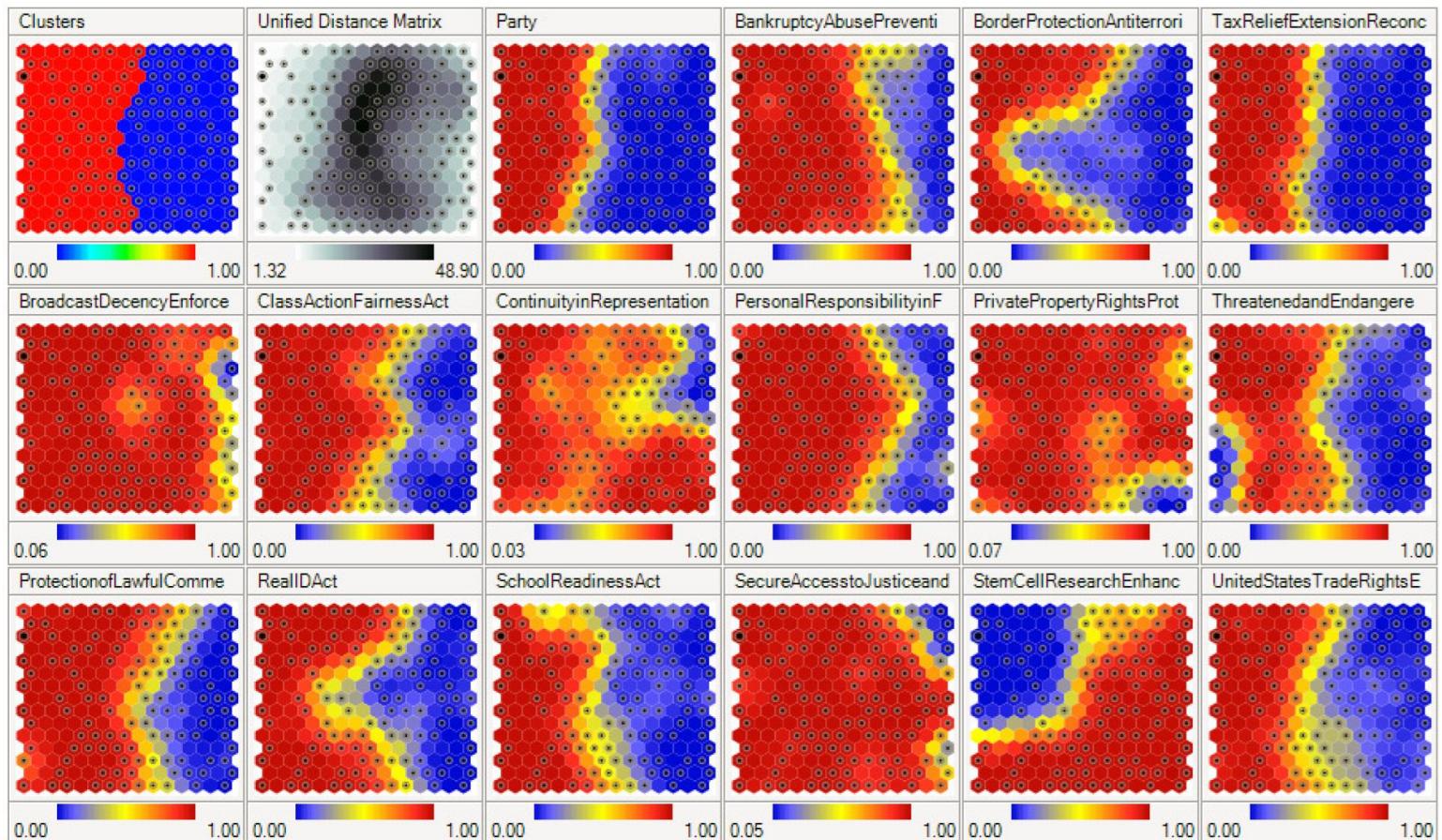
Размеры кластеров  
(карта интенсивности)

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



# Пример

- Наблюдения – конгрессмены, признаки партия и вопросы для голосования

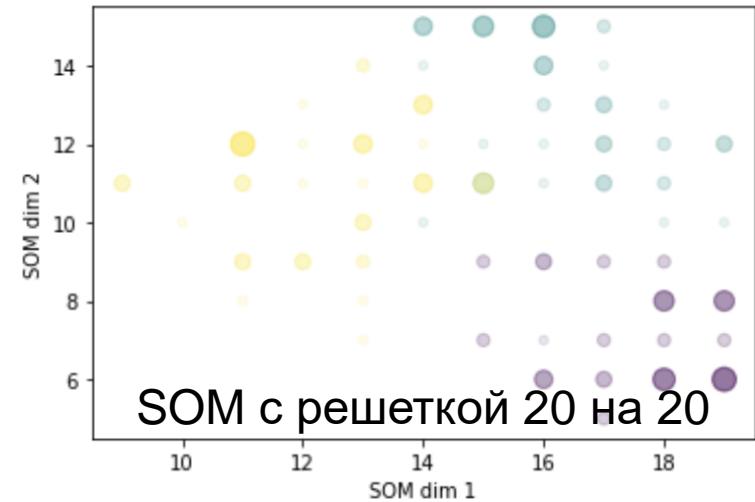
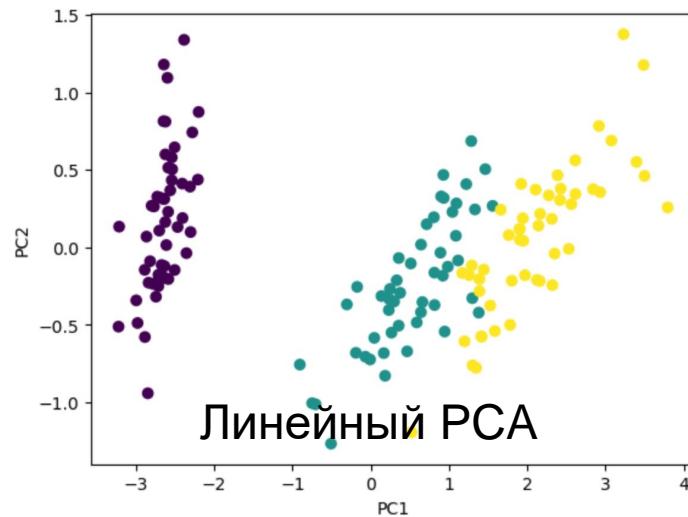


# Пример визуализации для набора Iris

```
# 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")
```



# Свойства SOM

- Апроксимация входного пространства признаков
  - «Сжатие» информации, связь с методом LVQ (кластеризация, 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 высокоплотные области на решетке
- 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

# Многомерное шкалирование (multidimensional scaling, MDS)

- Дано:  $(i, j) \in E$  – выборка ребер графа  $\langle V, E \rangle$ 
  - $R_{ij}$  – расстояния между вершинами ребра  $(i, j)$ , например, длина кратчайшего пути по графу (IsoMAP), при этом обычно учитывается только  $k$  ближайших соседей или максимальное расстояние

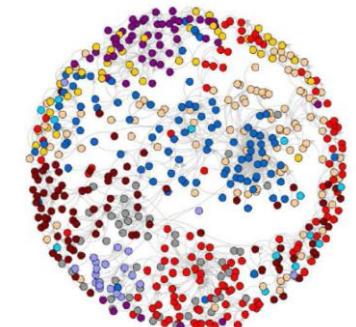
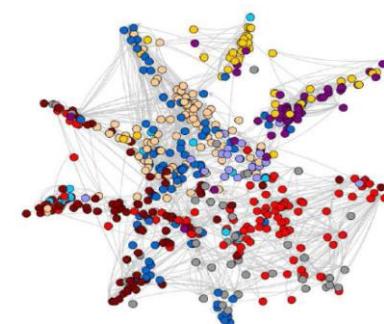
- Найти:
  - векторные представления вершин  $z_i \in \mathbb{R}^d$ , так, чтобы близкие (по графу) вершины имели близкие векторы.

- Свойства:

$$\sum_{(i,j) \in E} w_{ij} (\rho(z_i, z_j) - R_{ij})^2 \rightarrow \min_z, \quad Z \in \mathbb{R}^{V \times d}$$

где  $\rho(z_i, z_j) = \|z_i - z_j\|$  – расстояние Минковского,  $w_{ij}$  – веса (какие расстояния важнее).

- Обычно решается методом SGD, но **искажения**

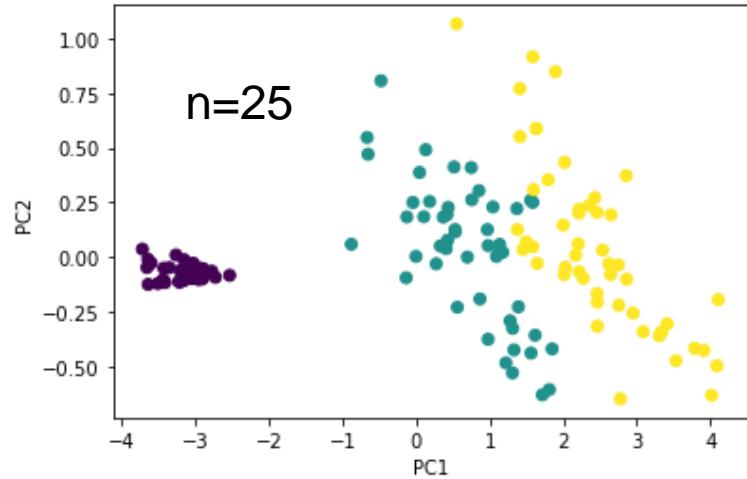
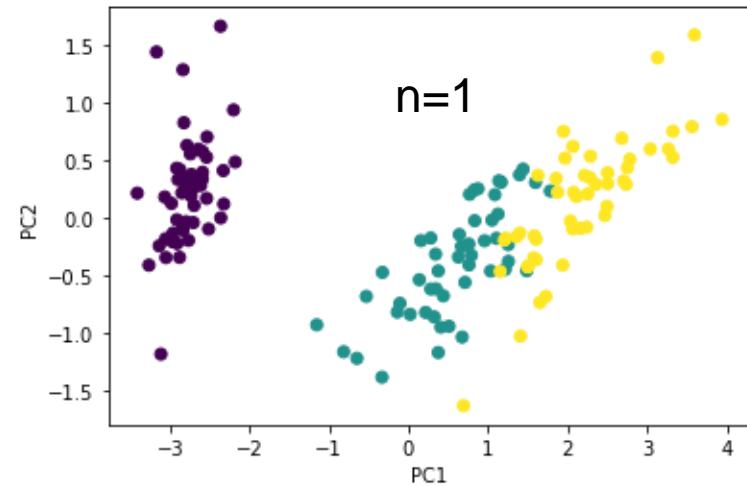
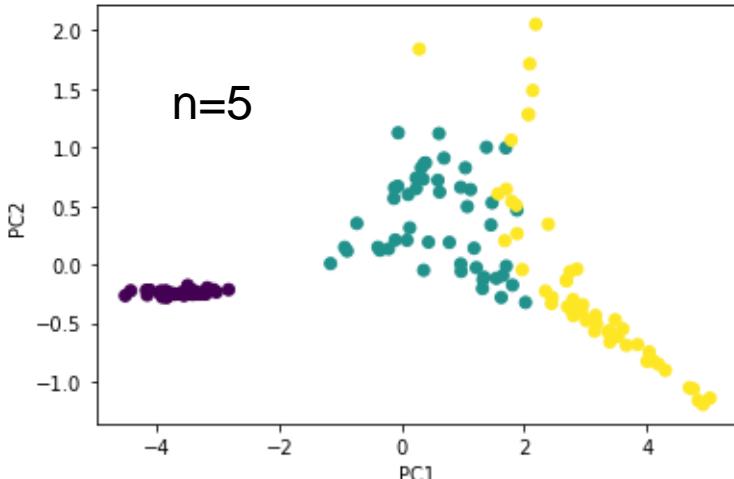


При  $d=2$  -  
проекции

# Пример для разных взвешенных расстояний

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

$n$  – параметр «число ближайших соседей»,  
похоже на параметр ширина RBF ядра  
(также может быть параметр – радиус)



# Stochastic Neighbor Embedding

- Цель (как и в SOM):
  - отобразить многомерное исходное пространство  $X$  в двух или трехмерное пространство  $Y$  топологически корректно

- Оценка близости двух точек через условные вероятности (в предположении нормального распределения) « $j$  рядом с  $i$ »:

- в **исходном** многомерном пространстве  $X$ :

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

- в пространстве **отображения**  $Y$ :

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

- Задача: ищется такое отображение, чтобы сохранить близость распределений  $p_{j|i}$  и  $q_{j|i}$  по дивергенции Кульбака-Лейблера

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

# Stochastic Neighbor Embedding

- Метод решения SGD с инерцией:

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

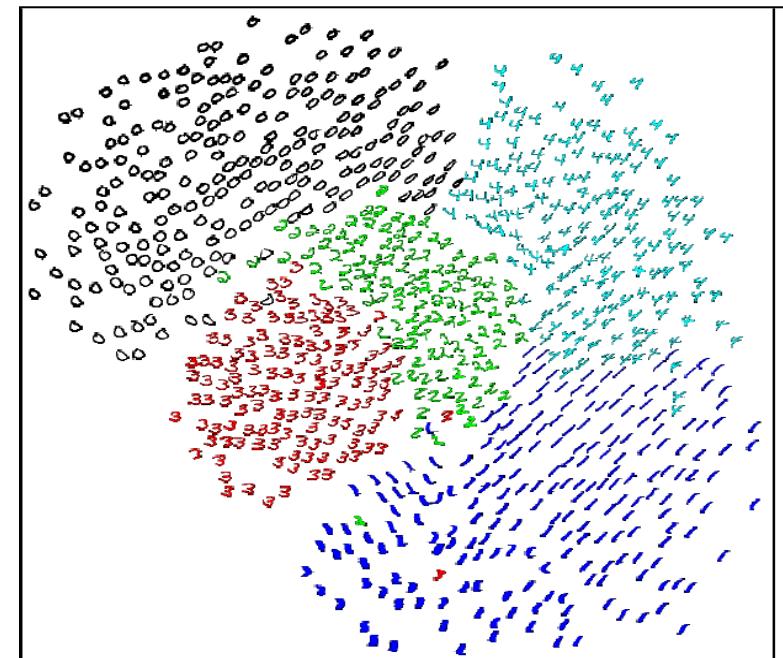
$$y^{(t)} = y^{(t-1)} + \eta \frac{\partial C}{\partial y} + \alpha(t)(y^{(t-1)} - y^{(t-2)})$$

- Свойства:

- рядом в X => рядом в Y
- далеко в X => не всегда далеко в Y

- Недостаток (crowding problem):

- неэффективная оптимизация
- проблема «скученных точек»
- нет границ между «облаками»



# SSNE (симметричный SNE) – эффективнее оптимизация

- Сходство:

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

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

- Целевая функция:

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

- Градиент:

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

- Симметричное сходство  
(совместное распределение вместо условного):

$$p_{ij} = 0.5(p_{j|i} + p_{i|j})$$
$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma^2)}{\sum_{k < n} \exp(-\|x_n - x_k\|^2 / 2\sigma^2)}$$
$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k < n} \exp(-\|y_n - y_k\|^2)}$$

- Симметричная целевая функция:

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

- Градиент:

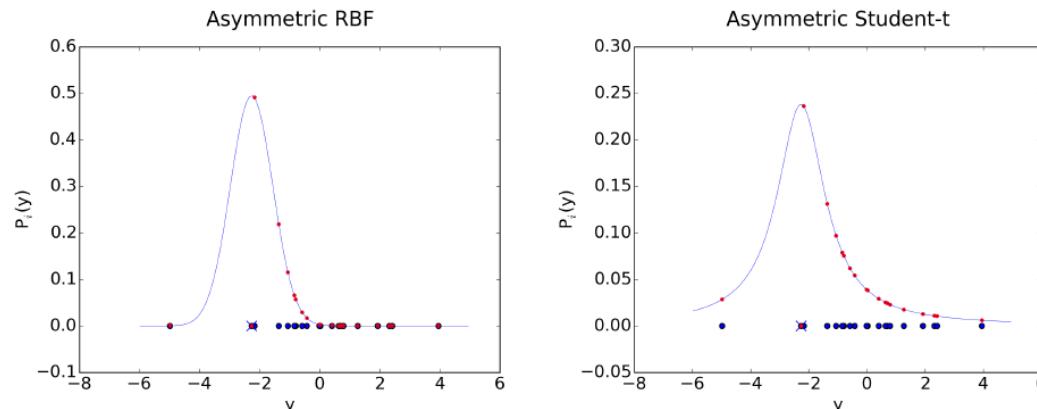
$$\frac{\partial C_{sym}}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

# t-SNE

- t-SNE: SSNE с распределением Стьюдента в пространстве Y:

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

у распределения Стьюдента более тяжелый хвост - при отображении точек в хвосте не так «скученны» при проекции:



- Градиент:

$$\frac{\partial C_{tSNE}}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}$$

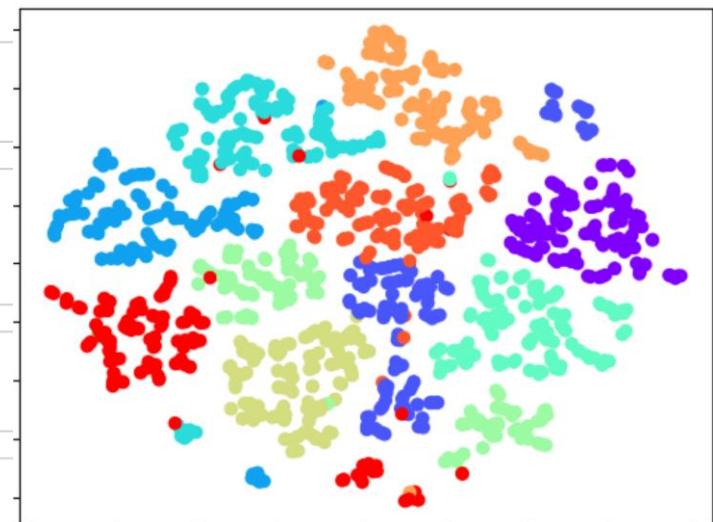
# Пример (набор MNIST)

```
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)
```

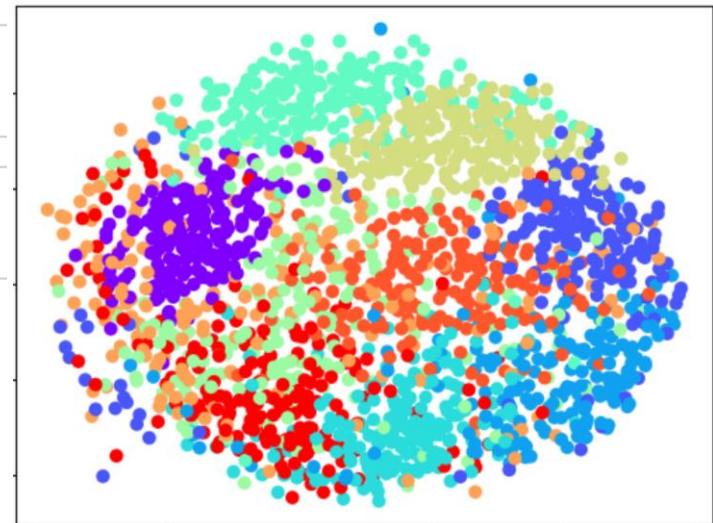
```
plt.scatter(*embedded.T, c=Y, cmap="rainbow")
```



```
# 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")
```

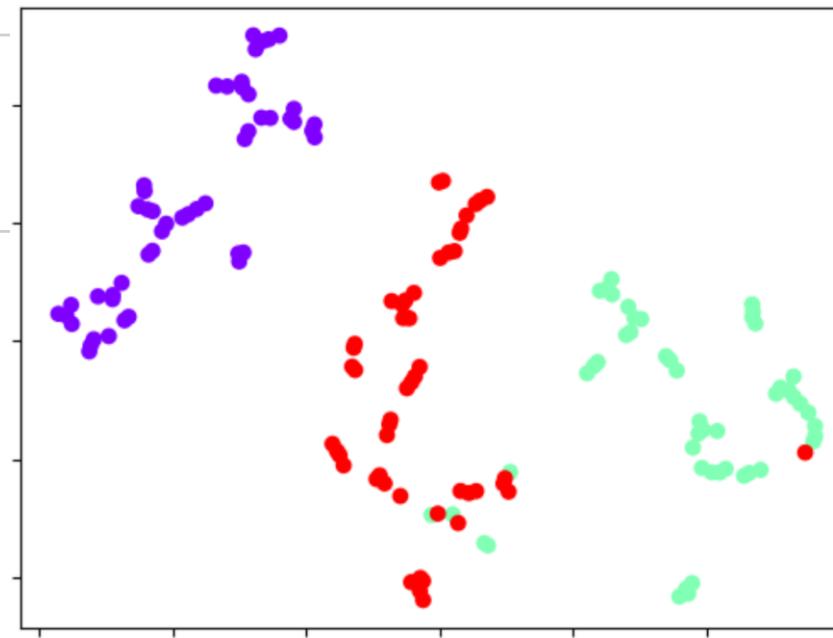


# Пример визуализации для набора Iris

```
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")
```

# UMAP (Uniform Manifold Approximation and Projection)

- Построение взвешенного графа, соединяя ребрами только те объекты, которые являются ближайшими соседями ( $k$  - параметр).

- Для всех  $x_i$  из исходного многомерного пространства:

- $N_k(x_i)$  – множество  $k$  ближайших соседей  $x_i$ ,  $\rho_i$  - расстояние до ближайшего соседа, находится нормирующий параметр  $\sigma_i$ , чтобы

$$\sum_{t \in N_k(x_i)} \exp\left(\frac{\rho(x_i, t) - \rho_i}{\sigma_i}\right) = \log_2 k$$

- Множество ребер графа — **нечёткое** с функцией принадлежности

- возможность существования ребра между соседями  $x_i$  и  $t_j \in N_k(x_i)$ :

$$w(x_i \rightarrow t_j) = \exp\left(\frac{\rho(x_i, t_j) - \rho_i}{\sigma_i}\right)$$

- и двумя произвольными вершинами (нечеткое «ИЛИ»):

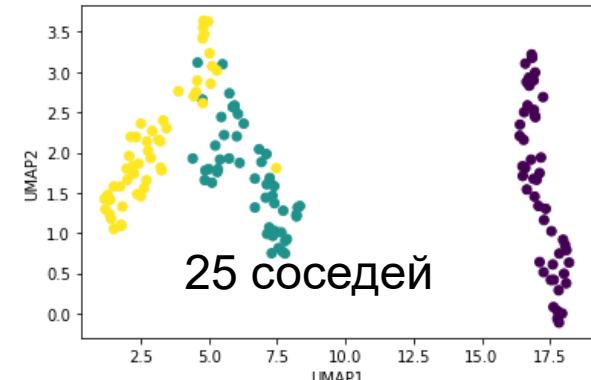
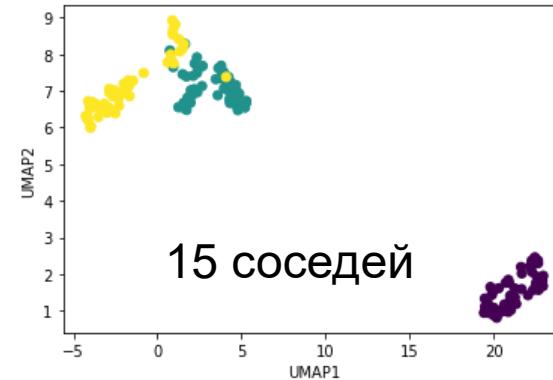
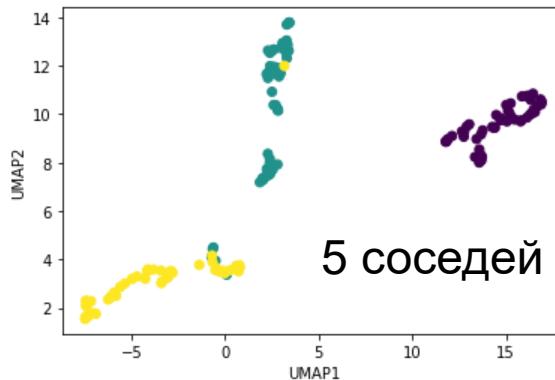
$$w(x_i, x_j) = w(x_i \rightarrow x_j) + w(x_j \rightarrow x_i) - w(x_i \rightarrow x_j)w(x_j \rightarrow x_i)$$

# Алгоритм UMAP

- Создается аналогичный ориентированный граф в низкоразмерном пространстве и приближается к исходному, минимизируя сумму дивергенций Кульбака-Лейблера для каждого ребра  $e$  в графе:

$$\sum_e \left[ w_{high}(e) \log \frac{w_{high}(e)}{w_{low}(e)} + (1 - w_{high}(e)) \log \frac{(1 - w_{high}(e))}{(1 - w_{low}(e))} \right] \rightarrow \min_{w_{low}}$$

- $w_{high}$  — функция принадлежности нечеткого множества из ребер в высокоразмерном пространстве,
  - $w_{low}$  — функция принадлежности нечеткого множества из ребер в низкоразмерном пространстве
- Пример Iris dataset:



# Автокодировщик (без учителя)

## ■ Постановка задачи:

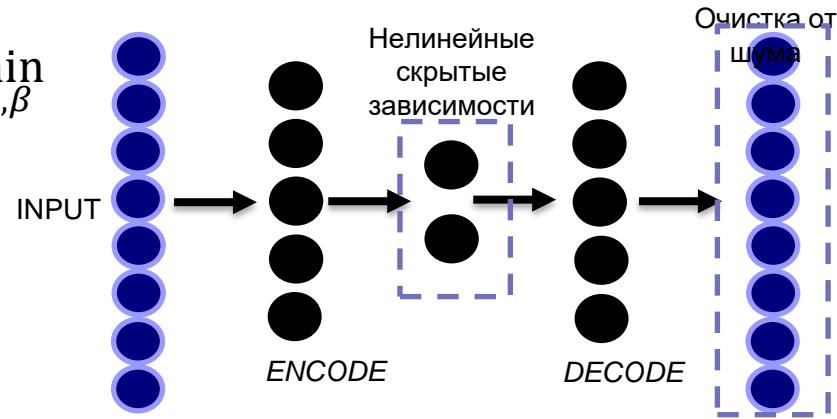
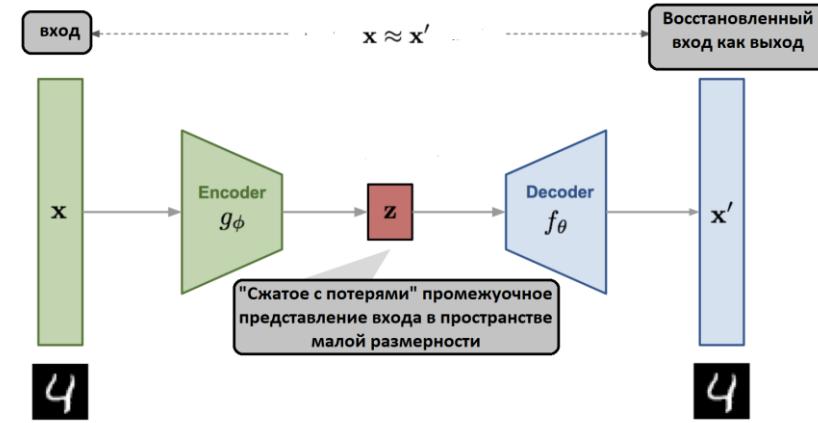
- $X^l = \{x_1, \dots, x_l\}$  – обучающая выборка
- $f: X \rightarrow Z$  – кодировщик (encoder),  
кодовый вектор  $z = f(x, \alpha)$
- $g: Z \rightarrow X$  – декодировщик (decoder),  
реконструкция  $\hat{x} = g(z, \beta)$
- Суперпозиция  $\hat{x} = g(f(x))$  должна  
восстанавливать исходные  $x_i$ :

$$\mathcal{L}_{AE}(\alpha, \beta) = \sum_{i=1}^l \mathcal{L}(g(f(x_i, \alpha), \beta), x_i) \rightarrow \min_{\alpha, \beta}$$

- Квадратичная функция потерь:  
$$\mathcal{L}(\hat{x}, x) = \|\hat{x} - x\|^2$$

## ■ Пример: простая сеть с функциями активации $\sigma_f, \sigma_g$ :

- $f(x, A) = \sigma_f(Ax + a), g(z, B) = \sigma_g(Bz + b)$



# Линейный автокодировщик и метод главных компонент

- Линейный автокодировщик:  $f(x, A) = Ax$ ,  $g(z, B) = Bz$

$$\mathcal{L}_{AE}(A, B) = \sum_{i=1}^l \|BAx_i - x_i\|^2 \rightarrow \min_{A, B}$$

- Метод главных компонент:  $f(x, U) = U^T x$ ,  $g(z, U) = Uz$
- В матричных обозначениях:  $F = (x_1, \dots, x_l)^T$ ,  $U^T U = I_m$ ,  $G = FU$

$$\|F - GU^T\|^2 = \sum_{i=1}^l \|UU^T x_i - x_i\|^2 \rightarrow \min_U$$

- Автокодировщик обобщает метод главных компонент:
  - Не обязательно  $B = A^T$  (хотя часто именно так и делают)
  - Произвольные  $A, B$  вместо ортогональных
  - Нелинейные модели  $f(x, \alpha), g(z, \beta)$  вместо  $Ax, Bz$
  - Произвольная функция потерь  $\mathcal{L}$  вместо квадратичной
  - SGD оптимизация вместо сингулярного разложения SVD

# Пример визуализации для набора Iris

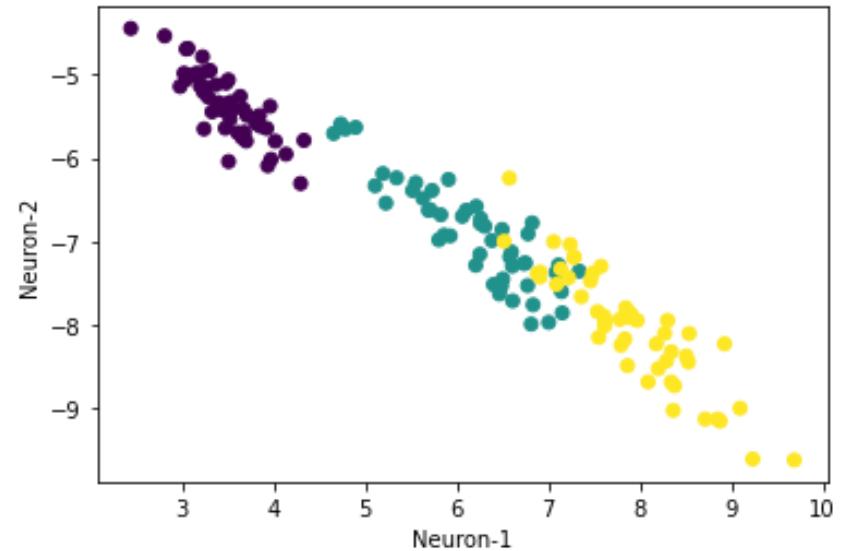
```
# Neural
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(hidden_layer_sizes=(2,), activation='tanh', max_iter=1000, alpha=0.001)
model.fit(X, X)

dummy = np.zeros((1, 2)) # Only dimension matters
hidden = MLPRegressor(hidden_layer_sizes=[], activation='tanh', 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
#hidden.coefs_[1] = model.coefs_[1] # 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
```

AE с 2 нейронами в среднем слое



# Разреживающие автокодировщики (Sparse AE)

- Применение  $L_1$  или  $L_2$ -регуляризации к векторам весов  $\alpha, \beta$ :

$$\mathcal{L}_{AE}(\alpha, \beta) + \lambda \|\alpha\| + \lambda \|\beta\| \rightarrow \min_{\alpha, \beta}$$

- Применение  $L_1$ -регуляризации к кодовым векторам  $z_i$ :

$$\mathcal{L}_{AE}(\alpha, \beta) + \lambda \sum_{i=1}^l \sum_{j=1}^m |f_j(x_i, \alpha)| \rightarrow \min_{\alpha, \beta}$$

- Энтропийная регуляризация для случая  $f_j \in [0,1]$ :

$$\mathcal{L}_{AE}(\alpha, \beta) + \lambda \sum_{j=1}^m KL(\varepsilon || \bar{f}_j) \rightarrow \min_{\alpha, \beta}$$

где  $\bar{f}_j = \frac{1}{l} \sum_{i=1}^l f_j(x_i, \alpha)$ ;  $\varepsilon \in (0, 1)$  – близкий к нулю параметр,

$KL(\varepsilon || \rho) = \varepsilon \log \frac{\varepsilon}{\rho} + (1 - \varepsilon) \log \frac{1 - \varepsilon}{1 - \rho}$  – KL-дивергенция.

# Шумоподавляющий автокодировщик (Denoising AE)

- Устойчивость кодовых векторов  $z_i$  относительно шума в  $x_i$ :

$$\mathcal{L}_{DAE}(\alpha, \beta) = \sum_{i=1}^l E_{\tilde{x} \sim q(\tilde{x}|x_i)} \mathcal{L}(g(f(\tilde{x}, \alpha), \beta), x_i) \rightarrow \min_{\alpha, \beta}$$

- Вместо вычисления  $E_{\tilde{x}}$  в методе SG объекты  $x_i$  сэмплируются и зашумляются по одному:  $\tilde{x} \sim q(\tilde{x}|x_i)$ .
- Варианты зашумления  $q(\tilde{x}|x_i)$ :

- $\tilde{x} \sim \mathcal{N}(x_i, \sigma^2 I)$  – добавление гауссовского шума
- Обнуление компонент вектора  $x_i$  с вероятностью  $p_0$
- Такие искажения  $x_i \rightarrow \tilde{x}$ , относительно которых реконструкция  $\hat{x}_i$  должна быть устойчивой

# Реляционный автокодировщик (Relational AE)

- Наряду с потерями реконструкции объектов минимизируем потери реконструкции отношений между объектами:

$$\mathcal{L}_{AE}(\alpha, \beta) + \lambda \sum_{i < j} \mathcal{L}(\sigma(\hat{x}_i^T \hat{x}_j), \sigma(x_i^T x_j)) \rightarrow \min_{\alpha, \beta}$$

- где  $\hat{x}_i = g(f(x_i, \alpha), \beta)$  – реконструкция объекта  $x_i$ ,
- $x_i^T x_j$  – скалярное произведение (близость) пары объектов,
- $\sigma(s) = (s - s_0)_+$  – пороговая функция с параметром  $s_0$   
(если векторы не близки, то неважно, насколько),
- $\mathcal{L}(\hat{s}, s)$  – функция потерь, например,  $(\hat{s} - s)^2$

- Эксперимент:
  - улучшается качество классификации изображений с помощью кодовых векторов на задачах MNIST, CIFAR-10

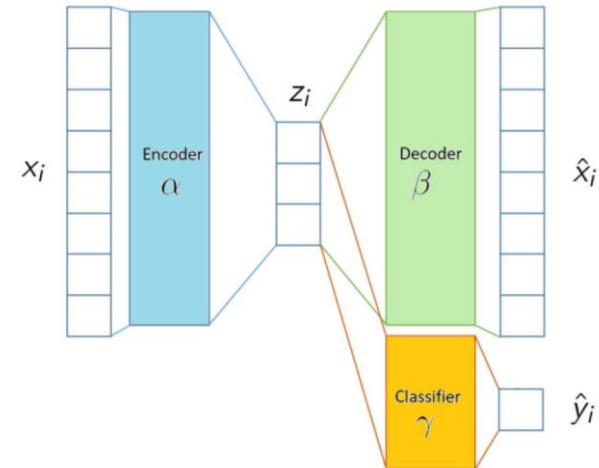
# Автокодировщики для обучения с учителем

- Данные:
  - размеченные  $(x_i, y_i)_{i=1}^k$ , неразмеченные  $(x_i)_{i=k+1}^l$
- Совместное обучение кодировщика, декодировщика и предсказательной модели (классификации, регрессии или др.):

$$\sum_{i=1}^l \mathcal{L}(g(f(x_i, \alpha), \beta), x_i) + \lambda \sum_{i=1}^k \tilde{\mathcal{L}}(\hat{y}(f(x_i, \alpha), \gamma), y_i) \rightarrow \min_{\alpha, \beta, \gamma}$$

- $z_i = f(x_i, \alpha)$  – кодировщик
- $\hat{x}_i = g(z_i, \beta)$  – декодировщик
- $\hat{y}_i = \hat{y}(z_i, \gamma)$  – предиктор

- Функции потерь:
  - $\mathcal{L}(\hat{x}_i, x_i)$  – реконструкция
  - $\tilde{\mathcal{L}}(\hat{y}_i, y_i)$  – предсказание



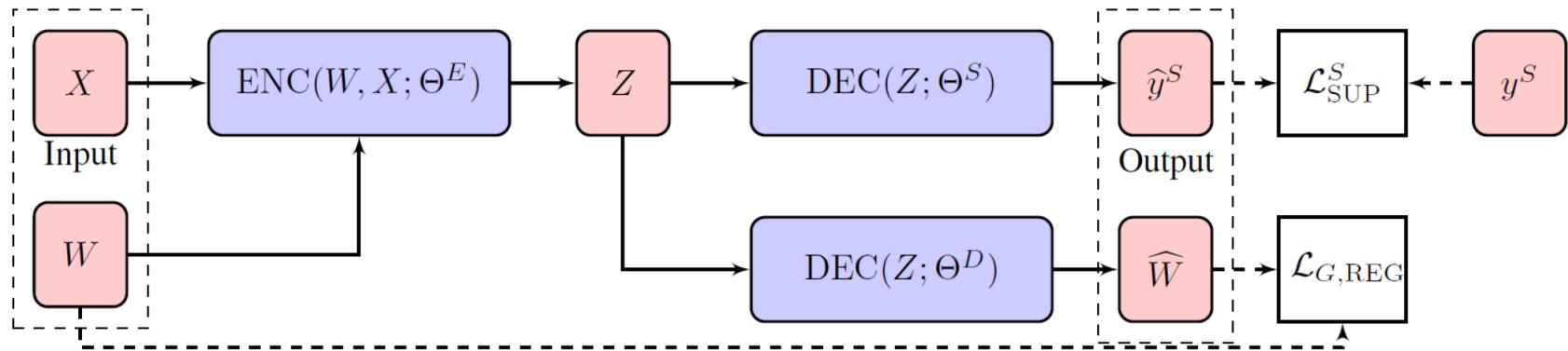
# Матричные разложения графа (graph factorization)

- Дано:  $(i, j) \in E$  – выборка ребер графа  $\langle V, E \rangle$ 
  - $S_{ij}$  – близость между вершинами ребра  $(i, j)$ .
  - Например,  $S_{ij} = [(i, j) \in E]$  – матрица смежности вершин.
- Найти: векторные представления вершин, так, чтобы близкие (по графу) вершины имели близкие векторы.
- Критерий для неориентированного графа ( $S$  симметрична):
$$\|S - ZZ^T\|_E = \sum_{(i,j) \in E} (\langle z_i, z_j \rangle - S_{ij})^2 \rightarrow \min_Z, Z \in \mathbb{R}^{V \times d}$$
- Критерий для ориентированного графа ( $S$  несимметрична):
$$\|S - \Phi\Theta^T\|_E = \sum_{(i,j) \in E} (\langle \varphi_i, \theta_j \rangle - S_{ij})^2 \rightarrow \min_{\Phi, \Theta}, \Phi, \Theta \in \mathbb{R}^{V, d}$$
- Обычно решается методом стохастического градиента (SGD).

# Векторные представления графов как автокодировщики

- Все рассмотренные выше методы векторных представлений графов суть автокодировщики данных о ребрах:
  - Многомерное шкалирование:  $R_{ij} \rightarrow \|z_i - z_j\|$
  - SNE и t-SNE:  $p(i, j) \rightarrow q(i, j) \propto K(\|z_i - z_j\|)$
  - Матричные разложения:  $S_{ij} \rightarrow \langle \varphi_i, \theta_j \rangle$
- Вход кодировщика:
  - $W_{ij}$  – данные о ребре графа  $(i, j)$
- Выход кодировщика:
  - Векторные представления вершин  $z_i$
- Выход декодировщика:
  - аппроксимация  $\hat{W}_{ij}$ , вычисляемая по  $(z_i, z_j)$

# GraphEDM: обобщенный автокодировщик на графах



- Graph Encoder Decoder Model – обобщает более 30 моделей.
  - $W \in \mathbb{R}^{V \times V}$  – входные данные о ребрах
  - $X \in \mathbb{R}^{V \times n}$  – входные данные о вершинах, признаковые описания
  - $Z \in \mathbb{R}^{V \times d}$  – векторные представления вершин графа
  - $\text{DEC}(Z; \Theta^D)$  – декодер, реконструирующий данные о ребрах
  - $\text{DEC}(Z; \Theta^S)$  – декодер, решающий задачу с учителем
  - $y^S$  – (semi-)supervised данные о вершинах или ребрах
  - $\mathcal{L}$  – функции потерь

# Векторизация слов

- Дано: текст  $(w_1, \dots, w_n)$ , состоящий из слов словаря  $W$
- Найти: векторные представления (embedding) слов  $v_w \in \mathbb{R}^d$ , так, чтобы близкие по смыслу слова имели близкие векторы, смысл определяется контекстом (окружением) слов.
- Модель Skip-gram:
  - учитывает порядок слов в контексте (скользящее окно) для предсказания по слову  $w_i$  вероятности слов контекста
$$C_i = (w_{i-k}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+k})$$
  - $p(w|w_i) = \text{Softmax}_{w \in W} \langle u_w, v_{w_i} \rangle \equiv \text{norm}_{w \in W} (\exp \langle u_w, v_{w_i} \rangle),$
  - $v_w$  – вектор предсказывающего слова,
  - $u_w$  – вектор предсказываемого слова, в общем случае  $u_w \neq v_w$
- Модель CBOW: на основе «непрерывного мешка слов», порядок не учитывает, предсказывает слово по контексту
- Критерий максимума log-правдоподобия,  $U, V \in \mathbb{R}^{|W| \times d}$ :
$$\sum_{i=1}^n \sum_{w \in C_i} \log p(w|w_i) \rightarrow \max_{U,V}$$

# Выводы по автокодировщикам

- Обучение – обычно SGD
- Достаточно универсальный инструмент
- Применение:
  - Понижение размерности
  - Нелинейные главные компоненты
  - Выявление аномалий (по ошибке реконструкции)
  - Векторизация сложно структурированных объектов
  - Генерация синтетических примеров
  - Удаление шума и выявление нелинейных признаков в задачах прогнозирования