«Машинное обучение и анализ данных»

Обучение без учителя

Александр Дьяконов

21 апреля 2020 года

План

Задачи обучения без учителя

Понижение (сокращение) размерности / Вложение в поверхности (Manifold Learning)

SVD, PCA, kernel PCA

LLE (Locally Linear Embedding)

SNE (Stochastic Neighbor Embedding)

t-SNE (t-distributed Stochastic Neighbor Embedding)

IsoMap (Isometric Mapping)

MDS (MultiDimensional Scaling)

Maximum Variance Unfolding

Spectral Embedding / Laplacian Eigenmap

ICA

Обучение без учителя (Unsupervised Learning)

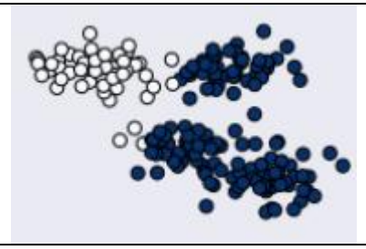
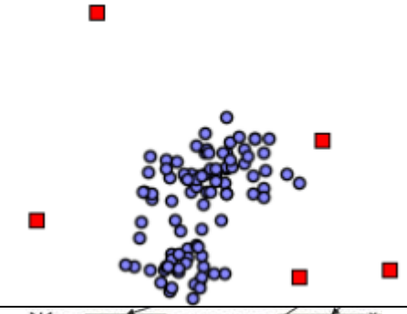
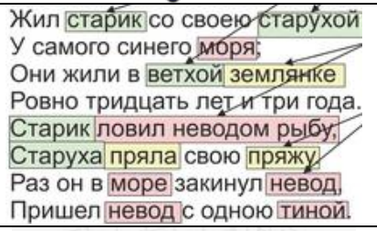
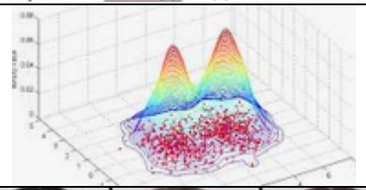
Главный вопрос исследователя – «как всё устроено?»

вход: неразмеченные данные

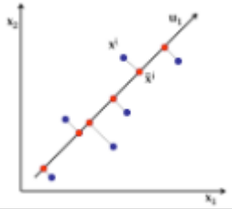
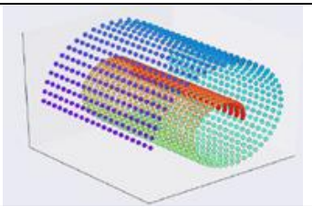
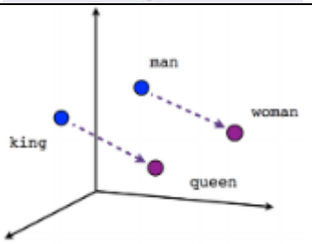
выход: описание структуры данных / упрощение данных / объяснение данных

Неформально: понимание, как данные устроены

Задачи обучения без учителя (Unsupervised Learning)

Кластеризация (Clustering) **отдельная лекция** нахождение таргетированных групп для акций, планирование эксперимента, визуализация, определение мутаций вируса	
Поиск аномалий **отдельная лекция** обнаружение фрода, поломок, инсайдеров и т.п.	
Тематические модели (Topic discovery) аналог кластеризации в тестах	
Оценка плотностей (Density Estimation) понимание вероятностной природы данных	
Генерация данных (Data Generation) создание объектов генеральной совокупности	

Задачи обучения без учителя (Unsupervised Learning)

Снижение размерности (Dimensionality Reduction) предобработка данных, сжатие информации, устранение избыточности	
Вложение в поверхности (Manifold Learning) понимание структурной природы данных, визуализация	
Получение представлений (Representation Learning) представление сложных объектов в простом пространстве	
Устранение шума (noise reduction) повышение качества данных	
Заполнение пропусков / дополнение матриц (matrix completion) матричная факторизация	
Поиск ассоциативных правил (association rule learning) отдельная лекция	**A → B**

Обучение без учителя – причины

- **неразмеченные данные проще получить**

- **методы USL можно применять до SL**

(в том числе, для получения новых признаков)

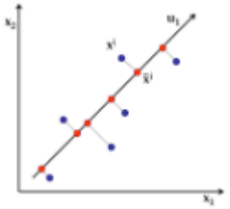
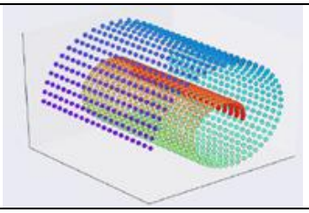
при этом нет риска переобучения, т.к. не видим метки,

но можем подглядывать в будущее

- $\Rightarrow$ **повышение качества / экономия памяти / интерпретация**

(в том числе, для последующей визуализации)

Дальше в этой лекции

Снижение размерности (Dimensionality Reduction) предобработка данных, сжатие информации, устранение избыточности	
Вложение в поверхности (Manifold Learning) понимание структурной природы данных, визуализация	

всё это полезно для визуализации и генерации новых признаков

Понижение (сокращение) размерности

$$X \in \mathbb{R}^{m \times n} \to Z \in \mathbb{R}^{m \times k}, k < n$$

подходы:

- выразимость X через Z (м.б. и наоборот)

ex: возможность восстановления (в DL автокодировщики)

- сохранение расстояний (или порядка расстояний)

меньше признаковое пространство =>

- **борьба с переобучением**
- **интерпретация**
- **визуализация**
- **скорость работы алгоритмов**
- **автоматическое удаление шума**
- **ниже стоимость признакового пространства**

Понижение (сокращение) размерности

$$X \in \mathbb{R}^{m \times n} \to Z \in \mathbb{R}^{m \times k}, k < n$$

Но отличается от отбора признаков!

получаем вообще говоря новую матрицу…

если признаков слишком много,

то найдётся случайно коррелирующий с целевым…

ex: случайная матрица размера n×n п.н. невырождена

Понижение (сокращение) размерности с помощью SVD

у нас было сингулярное разложение

$$X_{m\times n} = U\Lambda V^{\mathrm{T}}$$

и усечённое сингулярное разложение

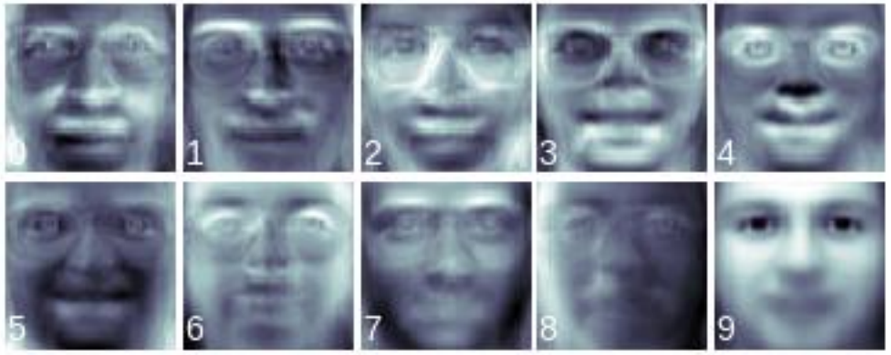
$$X_{m\times n} \approx X'_{m\times n} = U[:,1:k]\cdot \underbrace{\mathrm{diag}(\lambda_1,\ldots,\lambda_k)}_{\Lambda[1:k,1:k]}\cdot V[1:k,:]^{\mathrm{T}}$$

X' **лучшее (в каком смысле?) приближение матрицы** X
логично переходить к признаковому пространству $U[:,1:k]$

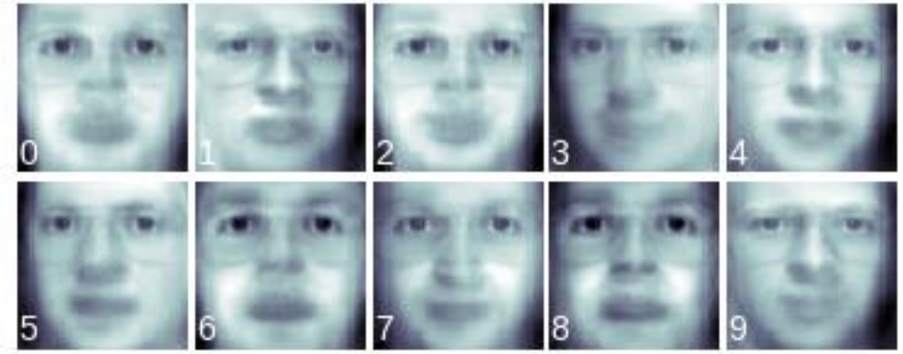
Эксперименты с лицами «Olivetti faces dataset»

датасет – 400 картинок 64×64

главные направления



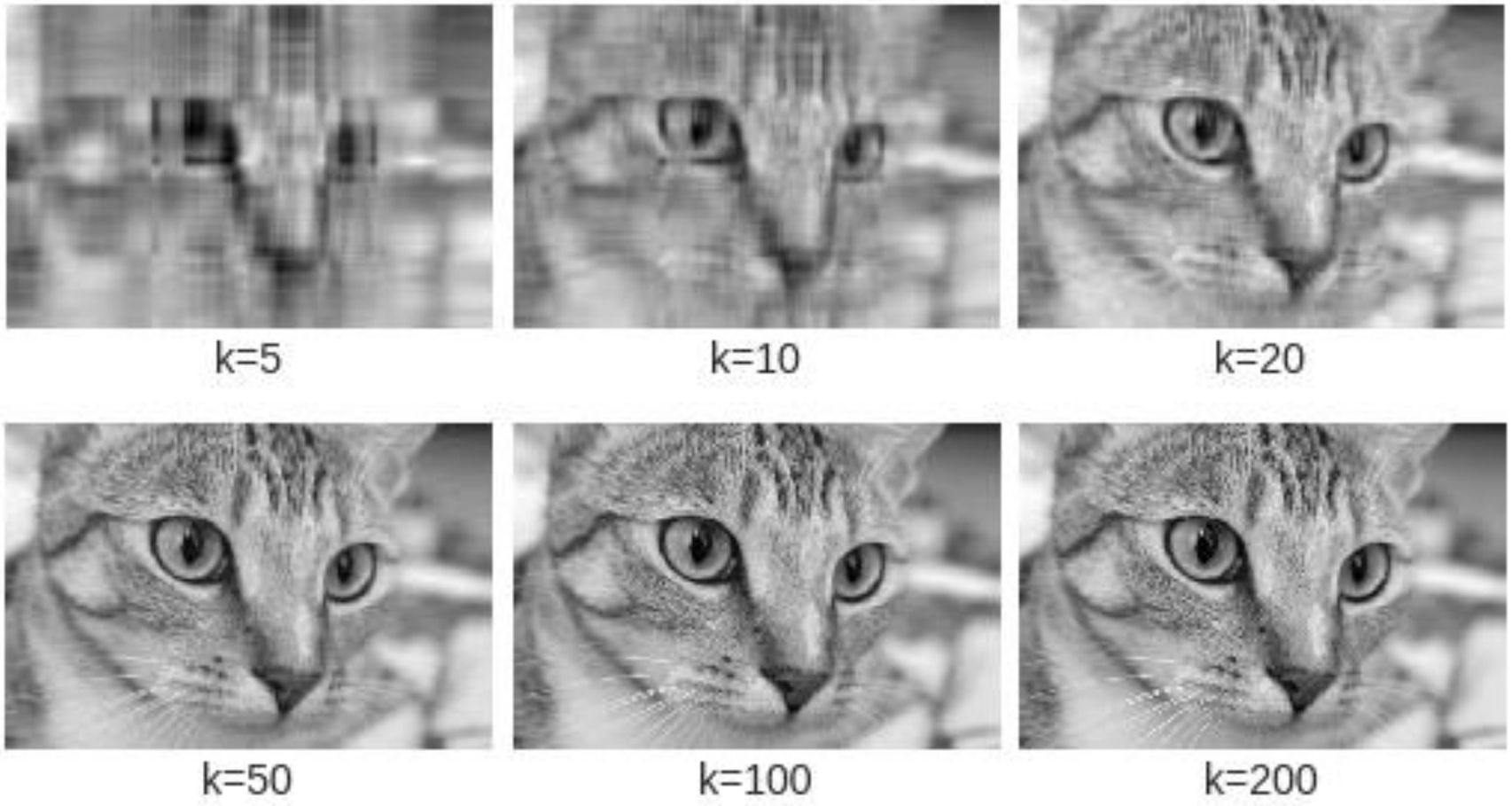
изображения, восстановленные по 10 компонентам



изображения, восстановленные по 2 компонентам

Вспомним – реконструкция изображений с помощью SVD

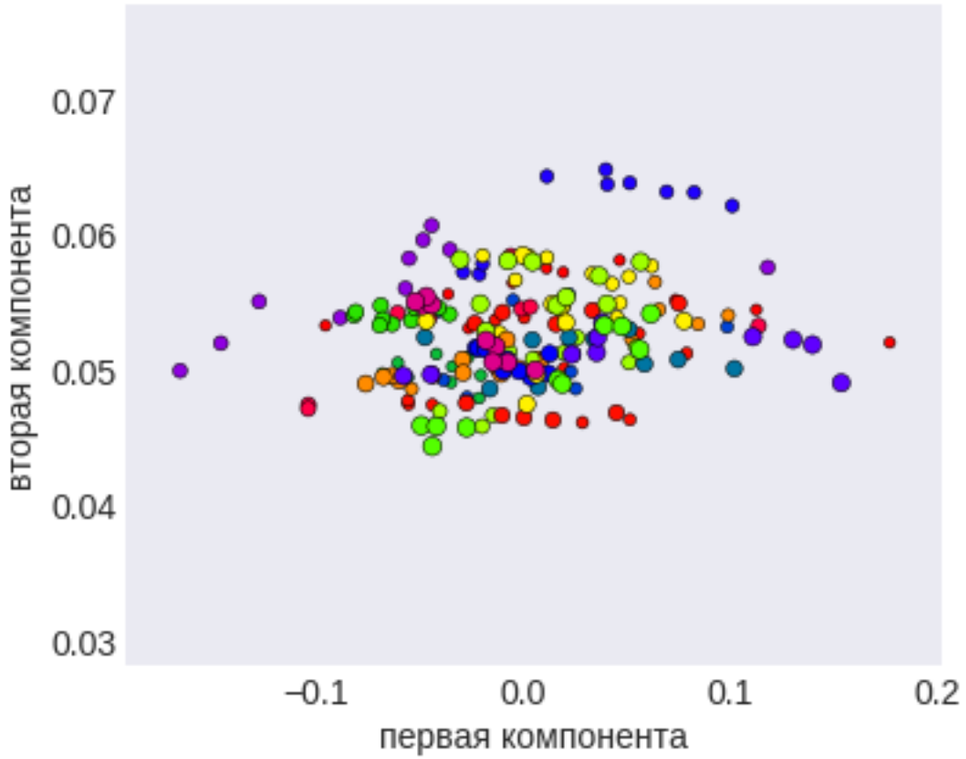
это отличается от применения SVD к изображениям, которое было ранее



Изначальный размер изображения 300×451 = 135 300

300×50 + 50×451 + 50 = 37 600

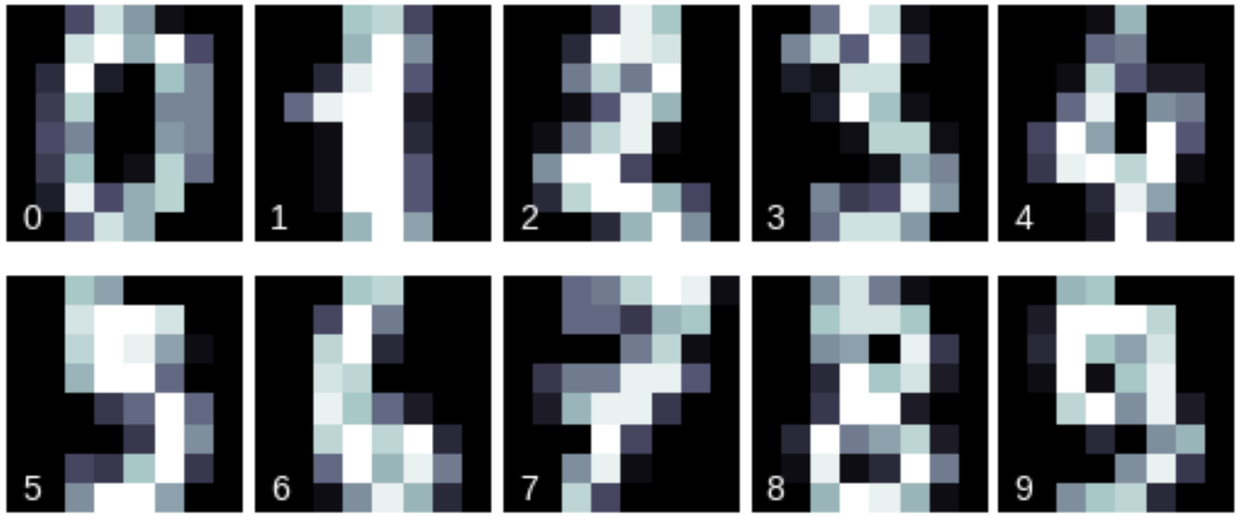
Эксперименты с лицами «Olivetti faces dataset»



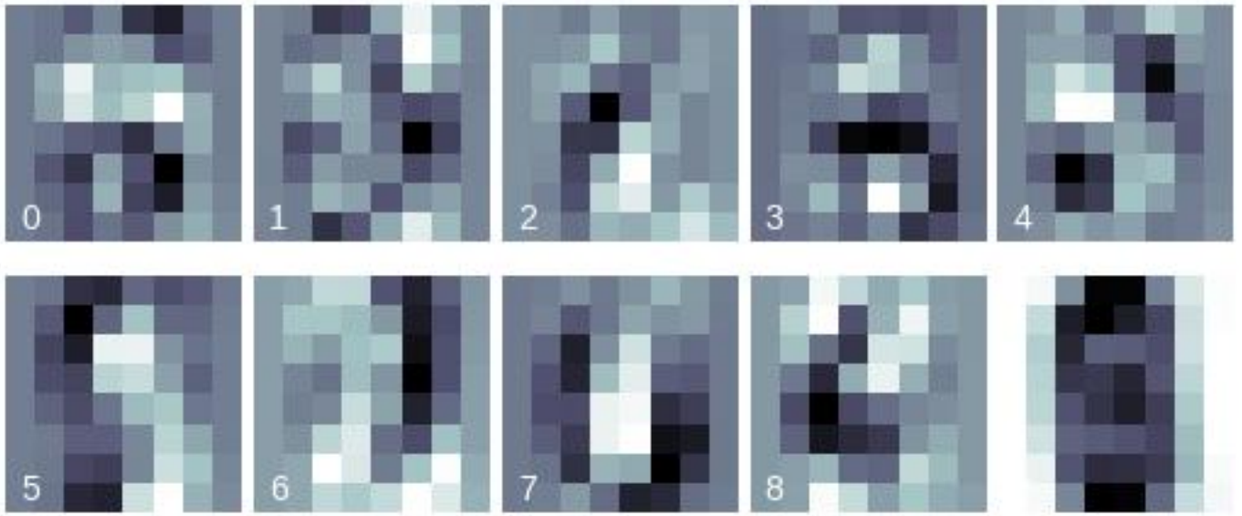

```
k = 2 # сколько компонент
from scipy.sparse.linalg import svds
U,L,V = svds(faces.data, k=k)

X2 = U @ np.diag(L) @ V
```

Эксперименты с лицами «digits»

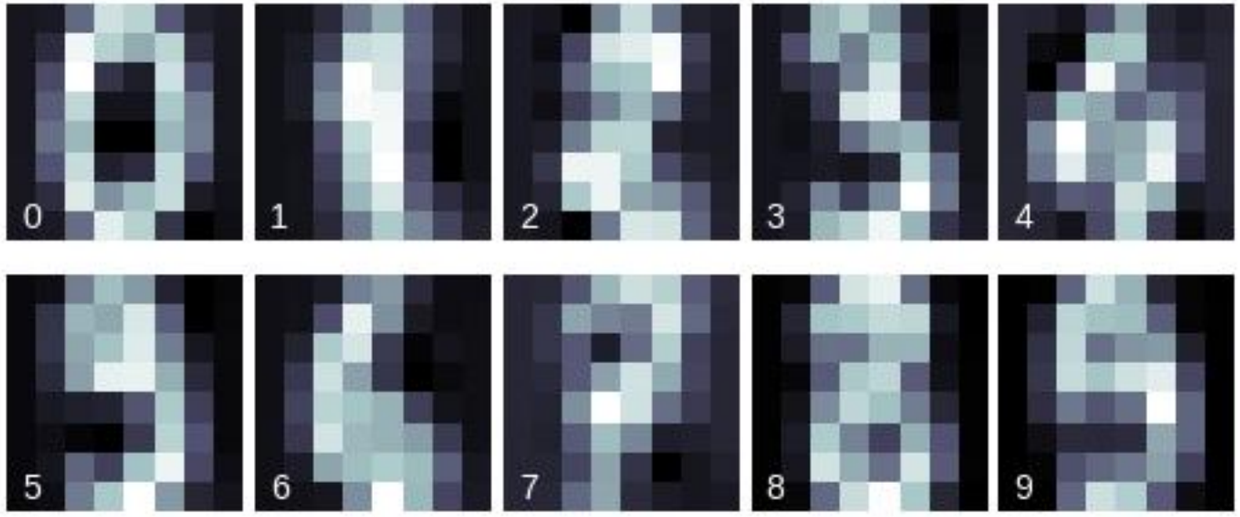
Датасет: 1797 картинок 8×8



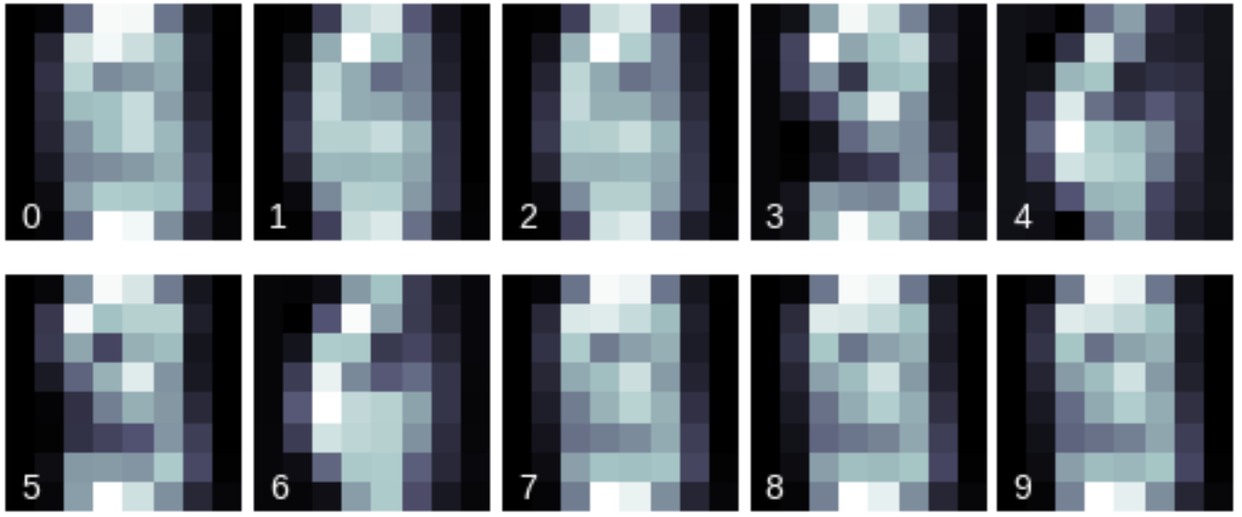
главные направления



восстановленные по 10 компонентам

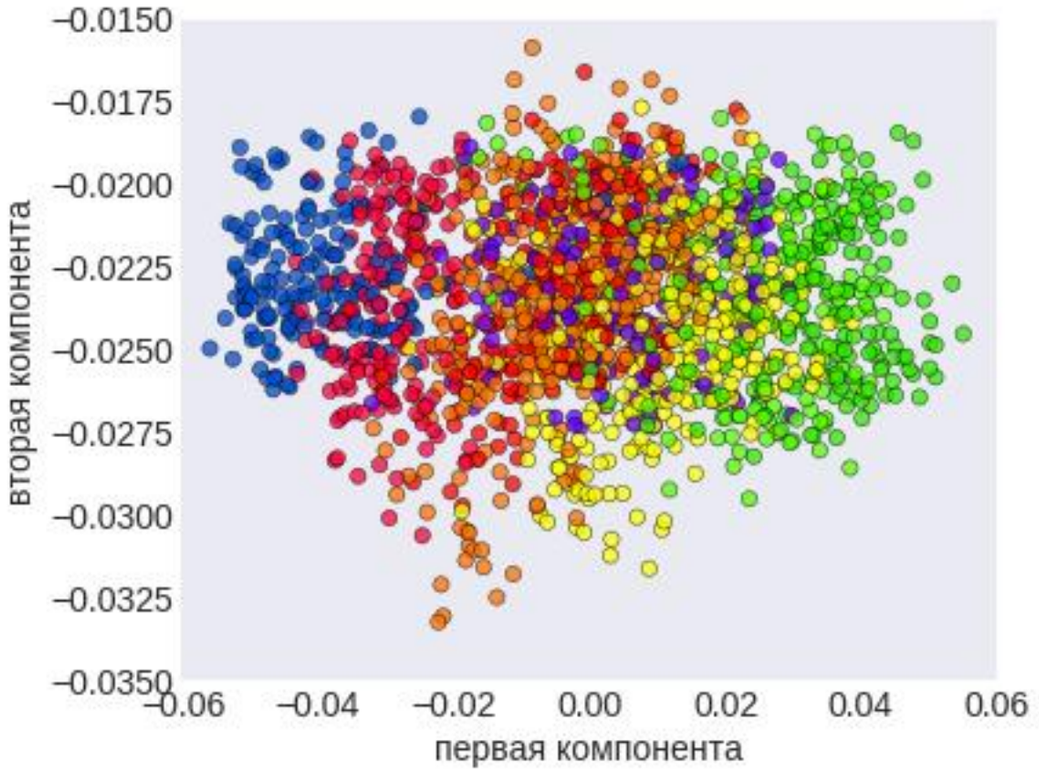


восстановленные по 2 компонентам

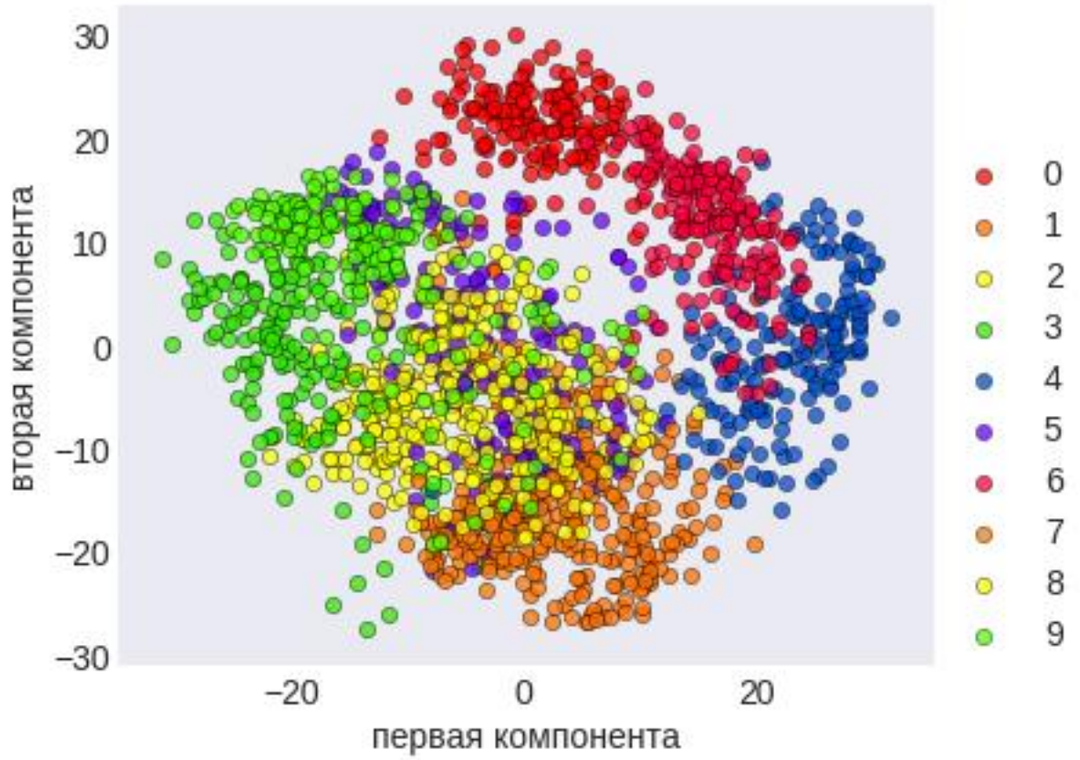


Эксперименты с лицами «digits»

SVD



PCA



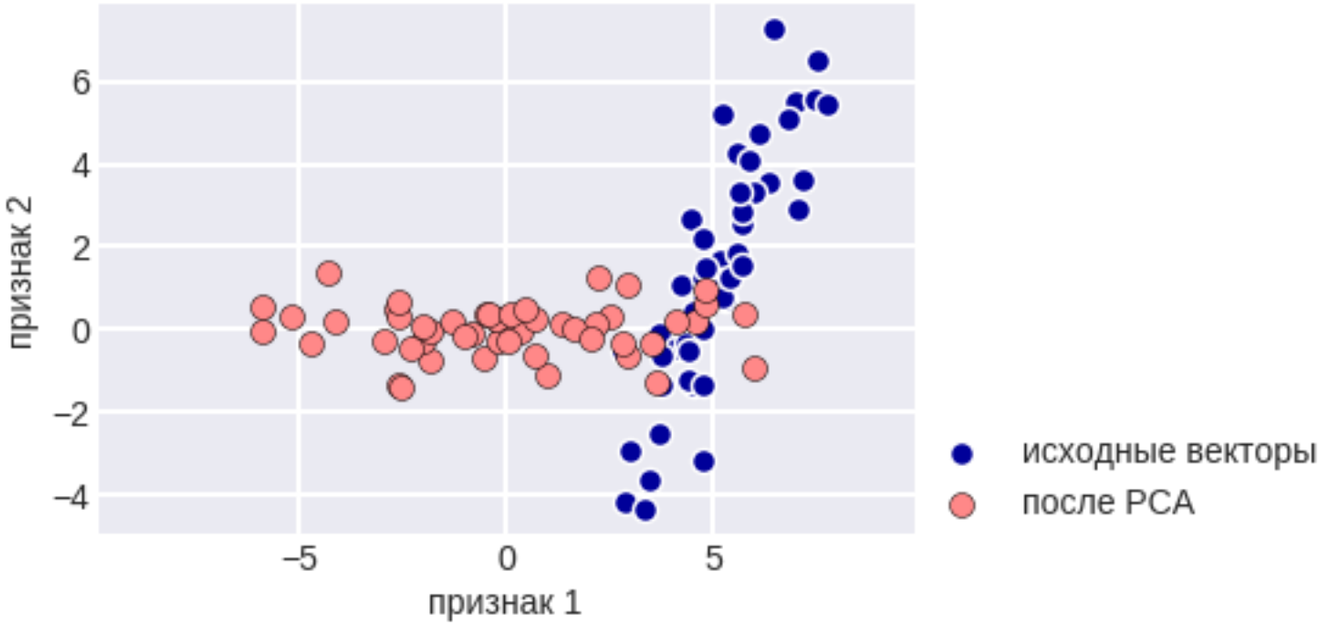
сейчас разберёмся в чём разница

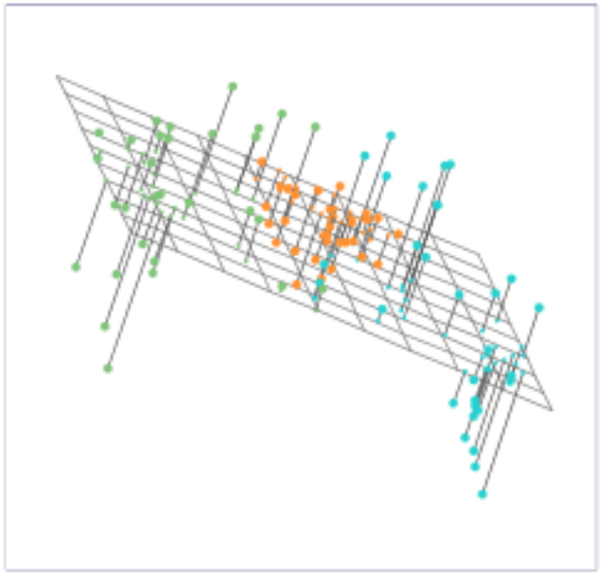
```
from sklearn.decomposition import PCA
estimator = PCA(n_components=10)
X_pca = estimator.fit_transform(X_digits)
```

Понижение размерности: PCA

Анализ главных компонент = Principal Component Analysis (PCA)

Представление данных в линейно преобразованном пространстве, если надо – меньшей размерности





Каждый признак нового пространства ищется в виде линейной комбинации исходных признаков так, чтобы максимизировать разброс при условии ортогональности (независимости) с уже найденными новыми признаками.

Понижение размерности: PCA – две интерпретации

ортогональная проекция данных в низкоразмерное пространство, которое

1) Maximum Variance Subspace – максимизирует разброс

Находим направление, проекции на которое имеют максимальный разброс

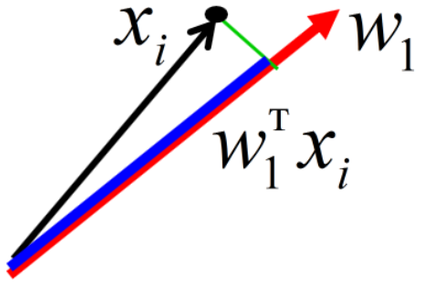
$$z_1^{\mathrm{T}} z_1 = w_1^{\mathrm{T}} X^{\mathrm{T}} X w_1 \to \max$$

формулу сейчас поясним

2) Minimum Reconstruction Error – минимизирует MSE (между точками и их проекциями)

Находим направление с минимальной ошибкой восстановления

$$\sum_{t=1}^{m} \| x_i - (w_1^{\mathrm{T}} x_i) w_1 \|^2 \to \min$$



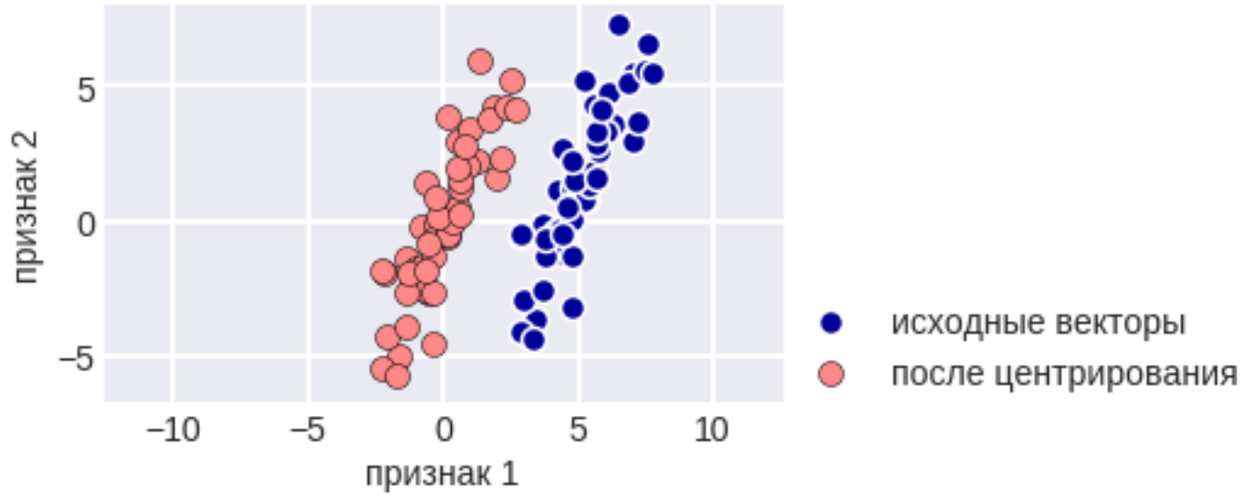
~ гиперплоскость, до которой минимальна сумма квадратов расстояний

картинка при условии $\| w_1 \| = 1$

Понижение размерности: PCA

Предполагаем, что все признаки центрированы (главное отличие от SVD):

$$\text{mean}(X_i) = 0$$



ищем первый признак в виде (он тоже будет центрированным)

$$Z_1 = w_1 X_1 + \ldots + w_n X_n$$

решая задачу (это разброс с учётом центрированности)

$$\frac{1}{m}\sum_{i=1}^{m}(w_1 x_{i1} + \ldots + w_n x_{in})^2 \to \max,\ w_1^2 + \ldots + w_n^2 = 1$$

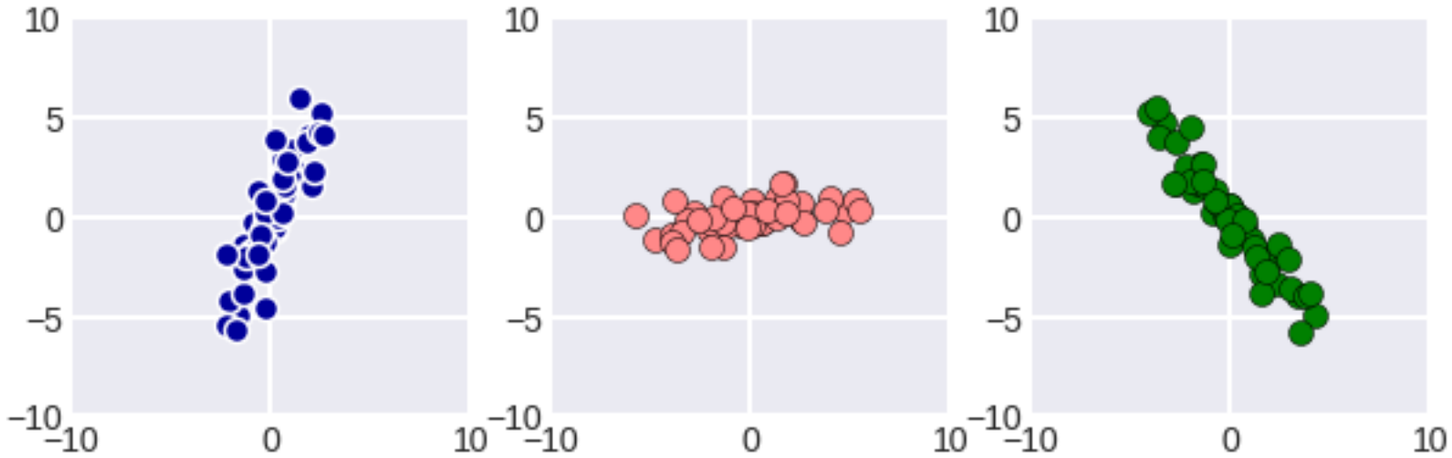
Понижение размерности: PCA

$$x \to x^{\mathrm{T}} w_1 = z_1$$

Матрично… хотим

$$\| z_1 \|^2 \to \max,\ z_1 = \frac{1}{\sqrt{m}} X w_1,\ \| w_1 \| = 1$$

$$\begin{cases} z_1^{\mathrm{T}} z_1 = \dfrac{1}{m} w_1^{\mathrm{T}} X^{\mathrm{T}} X w_1 \to \max \\ w_1^{\mathrm{T}} w_1 = 1 \end{cases}$$



ищем удачный поворот… точнее проекцию на ось с max разбросом

«теряется мало информации при переходе к проекции»

Понижение размерности: PCA

тут временно избавились от нормирующего множителя

$$J(w) = w^{\mathrm{T}} X^{\mathrm{T}} X w - \lambda(w^{\mathrm{T}} w - 1) \to \max$$

$$\frac{\partial J}{\partial w} = 2X^{\mathrm{T}} X w - 2\lambda w = 0$$

$$X^{\mathrm{T}} X w = \lambda w$$

если подставить…

$$J(w) = \lambda w^{\mathrm{T}} w - \lambda(w^{\mathrm{T}} w - 1) = \lambda$$

решение – с.в. ~ максимальное с.з. (= разброс в оптимальном решении)

понятна связь с SVD (**см. дальше**)

Понижение размерности: PCA

Если искать не один признак, а гиперплоскость, на которую проецируем

подробно не доказываем

$$x^{\mathrm{T}} \to z^{\mathrm{T}} = x^{\mathrm{T}}V = \begin{pmatrix} v_1^{\mathrm{T}}x \\ \vdots \\ v_k^{\mathrm{T}}x \end{pmatrix}$$

векторы $v_1,\cdots,v_k$ **– с.в., соотв. наибольшим с.з. матрицы ковариаций** $S = \frac{1}{m}X^{\mathrm{T}}X$**:**

$$\lambda_1 \geq \cdots \geq \lambda_k \geq \ldots$$

переменные в новом пространстве (координаты вектора z **)**
называются главными компонентами (principal components)

а «проекторы» $v_1,\cdots,v_k$ **называются главными направлениями / осями**
(principal directions/axes)

PCA

- **Вычислить средние**

$$\overline{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

- **Вычислить матрицу ковариаций**

$$S = \frac{1}{m}\sum_{i=1}^{m}(x_i - \overline{x})(x_i - \overline{x})^{\mathrm{T}}$$

- **Вычислить k с.в. соответствующих максимальным k с.з. матрицы S:**

$$v_1, \cdots, v_k : \lambda_1 \geq \cdots \geq \lambda_k$$

- **матрица проекций:** $V = [v_1, \cdots, v_k]$

$$z^{\mathrm{T}} = x^{\mathrm{T}}V = \begin{pmatrix} v_1^{\mathrm{T}}x \\ \vdots \\ v_k^{\mathrm{T}}x \end{pmatrix}$$

Чаще если

$$U, L, V = \mathrm{svd}([x_i - \overline{x}]_{i=1}^{m})$$

$$X \to [\lambda_1 u_1, \cdots, \lambda_k u_k]$$

эти лямбда корни тех ←

(с точностью до константы $1/m$)

Почему:

Если $X = ULV^{\mathrm{T}}$ для центрированных данных, то

$$S = X^{\mathrm{T}}X = VL^{\mathrm{T}}U^{\mathrm{T}}ULV^{\mathrm{T}} = VL^2V^{\mathrm{T}}$$

видим задачу на с.в.:

$$SV = VL^2$$

столбцы V – главные направления

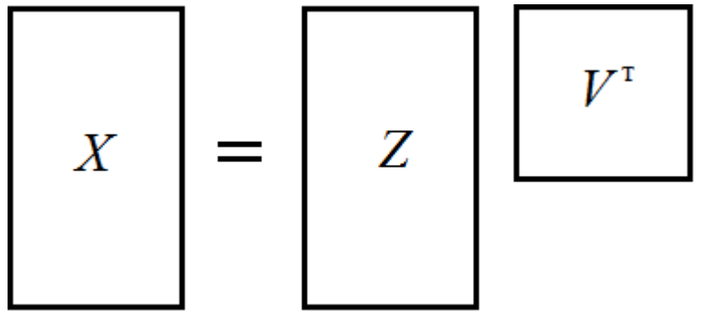
столбцы UL – главные компоненты

PCA / SVD – другой взгляд: факторизация

пусть мы не сокращаем размерность,

а просто проводим линейное преобразование $\mathbb{R}^n \to \mathbb{R}^n$**, тогда**

$$Z_{m\times n} = X_{m\times n} V_{n\times n}$$



если $V^{\mathrm{T}}V = VV^{\mathrm{T}} = I$**, то**

$$X = ZV^{\mathrm{T}}$$

т.е. это факторизация матрицы X

Кстати, что означает $VV^{\mathrm{T}} = I$

для вектора $z^{\mathrm{T}} = x^{\mathrm{T}}V$

$$\| \underset{z=Z[i,:]}{z} \|_2^2 = \| V^{\mathrm{T}} \underset{x=X[i,:]}{x} \|_2^2 = x^{\mathrm{T}}VV^{\mathrm{T}}x = \| x \|_2^2$$

(тут вектор-строки)

т.е. не меняются расстояния $\Rightarrow$ **поворот**

PCA – другой взгляд: декоррелированность

Рассмотрим центрированные данные, тогда

$$S = \frac{1}{m}\sum_{i=1}^{m}(x_i - \overline{x})(x_i - \overline{x})^{\mathrm{T}} = X^{\mathrm{T}}X$$

$$X = ULV^{\mathrm{T}}$$

Z

без сокращения размерности:

$$Z = XV$$

тогда матрица разброса после преобразования (данные тоже будут центрированные)

$$Z^{\mathrm{T}}Z = V^{\mathrm{T}}X^{\mathrm{T}}XV = V^{\mathrm{T}}VL^{\mathrm{T}}U^{\mathrm{T}}ULV^{\mathrm{T}}V = L^{\mathrm{T}}L = L^2$$

диагональная матрица, т.е. компоненты полученного вектора не коррелированны

а разброс ~ диагональные элементы (больше всего у первой координаты и т.д.)

PCA – другой взгляд: «Minimum Reconstruction Error»

в наших введённых обозначениях, когда мы сначала повернули пространство (с помощью V), а потом оставляем k компонент, ошибка реконструкции

$$\varepsilon = \| XV - \underset{Z}{XV\,|_{\text{zero}}} \|_F^2 = \| XVV^{\text{T}} - \underset{Z}{XV\,|_{\text{zero}}}\, V^{\text{T}} \|_F^2$$

zero **– зануление всех компонент (координат, тут столбцов), начиная с k+1**

V^{T} **– от домножения на ортогональную матрицу норма Фробениуса не зависит**

тогда

$$\varepsilon = \| ULV^{\text{T}} - UL\,|_{\text{zero}}\, V^{\text{T}} \|_F^2 = \| ULV^{\text{T}} - U\,\text{diag}(\lambda_1,\ldots,\lambda_k,0,\ldots,0)V^{\text{T}} \|_F^2$$

$$\varepsilon = \| \underbrace{U \underbrace{\text{diag}(0,\ldots,0,\lambda_{k+1},\ldots,\lambda_n)}_{D} V^{\text{T}}}_{H} \|_F^2 = \text{trace}(HH^{\text{T}}) = \text{trace}(UD^2U^{\text{T}}) = \text{trace}(D^2)$$

$$\varepsilon = \lambda_{k+1}^2 + \ldots + \lambda_n^2$$

Понижение размерности: PCA

Proportion Variance Explained (PVE) / Explained Variance Ratio (EVR) k**-й компоненты –** λ_k^2

Часто смотрят на

$$\frac{\lambda_1^2 + \ldots + \lambda_k^2}{\lambda_1^2 + \ldots + \lambda_n^2}$$

PCA: сколько компонент использовать?

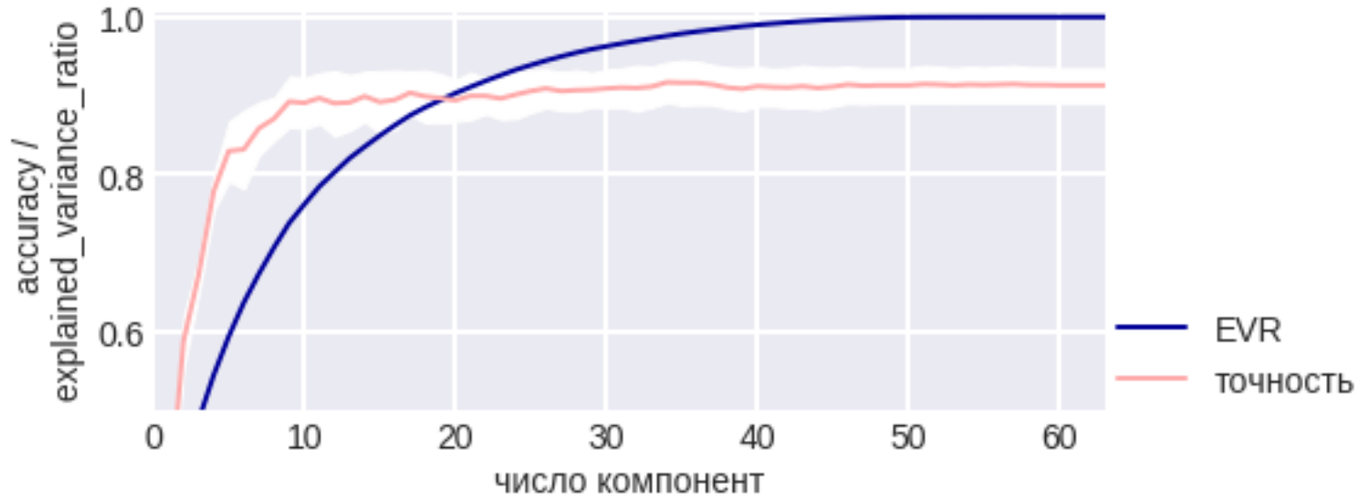
- **можно определить скользящим контролем,**

если PCA используется для обучения с учителем

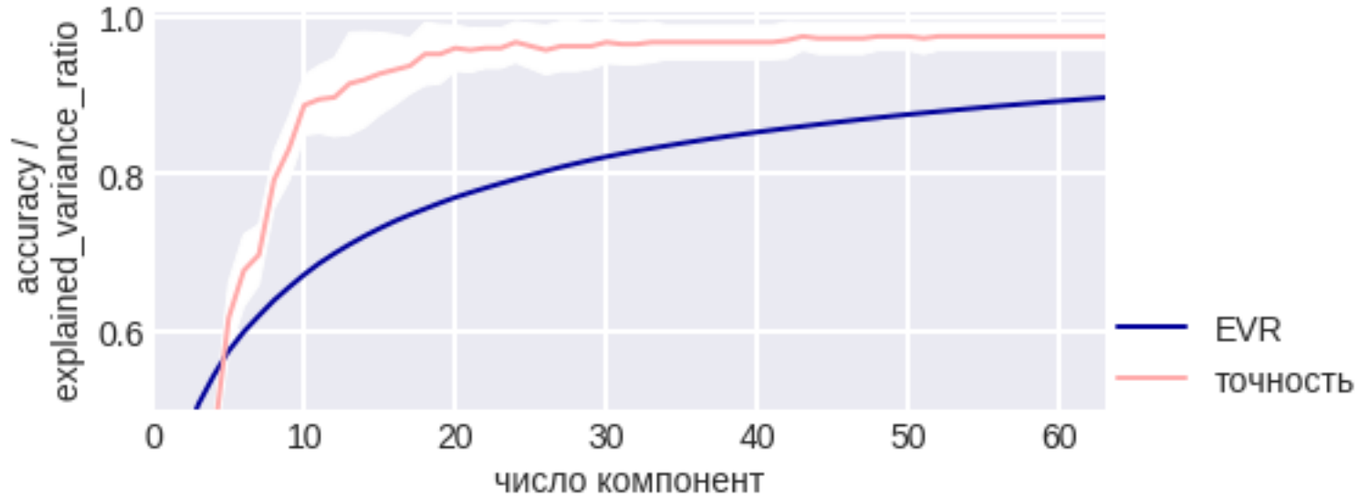
- **по графику кумулятивного PVE**

PCA: сколько компонент использовать?

«digits»



«faces»



показана точность на первых k компонентах методом логистической регрессии

PCA: поиск с.в.

1. «По определению»

2. Итерационный метод для нахождения с.в.

$$w^{(t+1)} = X^{\mathrm{T}} X w^{(t)}$$

$$w^{(t+1)} = w^{(t+1)} / \| w^{(t+1)} \|$$

3. EM-алгоритм

см. в [Бишопе] вероятностную трактовку PCA

PCA: поиск с.в.

$$S = \frac{1}{m} X^{\mathrm{T}} X$$

собственные векторы: $S = \frac{1}{m} X^{\mathrm{T}} X v_i = \lambda_i v_i$

умножим слева на X

$$\frac{1}{m} \underbrace{X X^{\mathrm{T}}}_{u_i} \underbrace{X v_i}_{u_i} = \lambda_i \underbrace{X v_i}_{u_i}$$

$$\frac{1}{m} X X^{\mathrm{T}} u_i = \lambda_i u_i$$

Если $n >> m$ **можно вычислить с.в.** u **матрицы** XX^{T}

$$\frac{1}{m} X X^{\mathrm{T}} \sim (\lambda_i, u_i) \quad \leftrightarrow \quad \frac{1}{m} X^{\mathrm{T}} X \sim (\lambda_i, v_i)$$

$$v_i = \frac{1}{(m\lambda_i)^{1/2}} X^{\mathrm{T}} u_i$$

константа из соображения нормировки
(подробно не рассказываем)

Особенности PCA

– PCA зависит от масштаба (стандартизация)

– PCA чувствителен к выбросам

+ PCA можно кернализовать…

+ PCA эквивалентен SVD после централизации данных и λ-масшабированием

$\Rightarrow$ оптимальное линейное преобразование

– но только линейное

+ сокращение размерности, можно для больших размерностей

– 2D может не годиться для интерпретации

+ новые признаки генерируются с помощью обучения без учителя

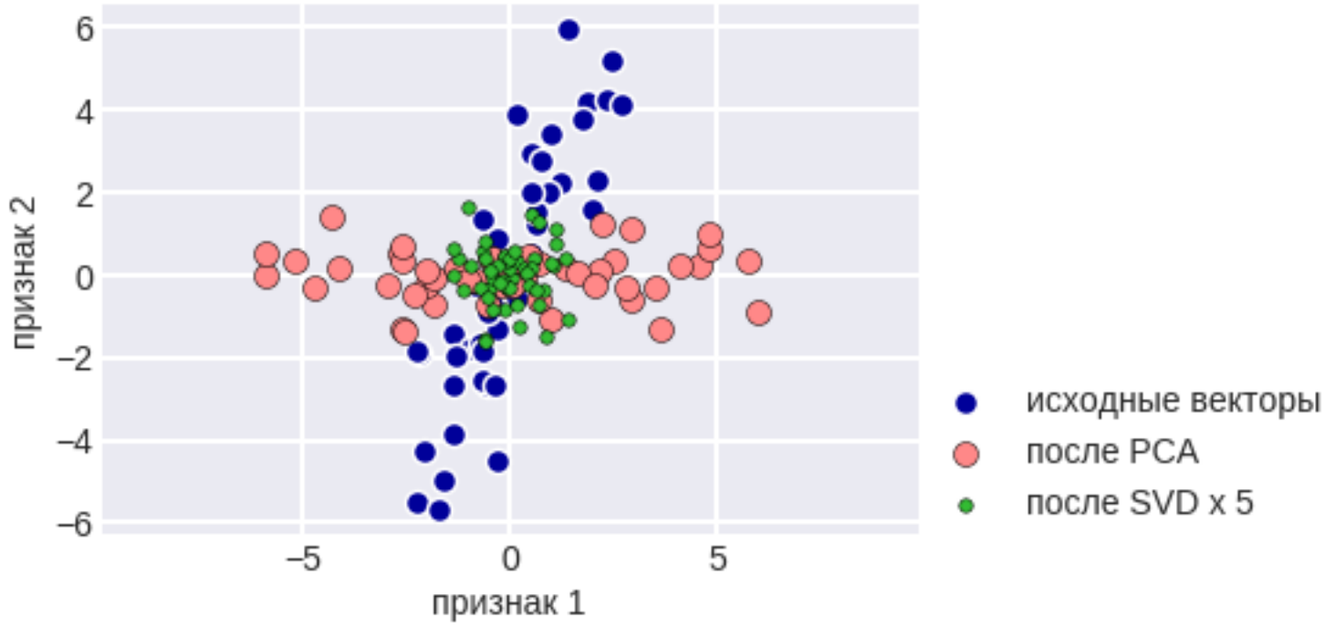
не подглядываем в целевые значения

– нет гарантии, что в получаемых признаковых пространствах задача хорошо решается

! столбцы в матрицы U могут быть неоднозначны (с точностью до знака)!

могут ли быть другие причины неоднозначности?

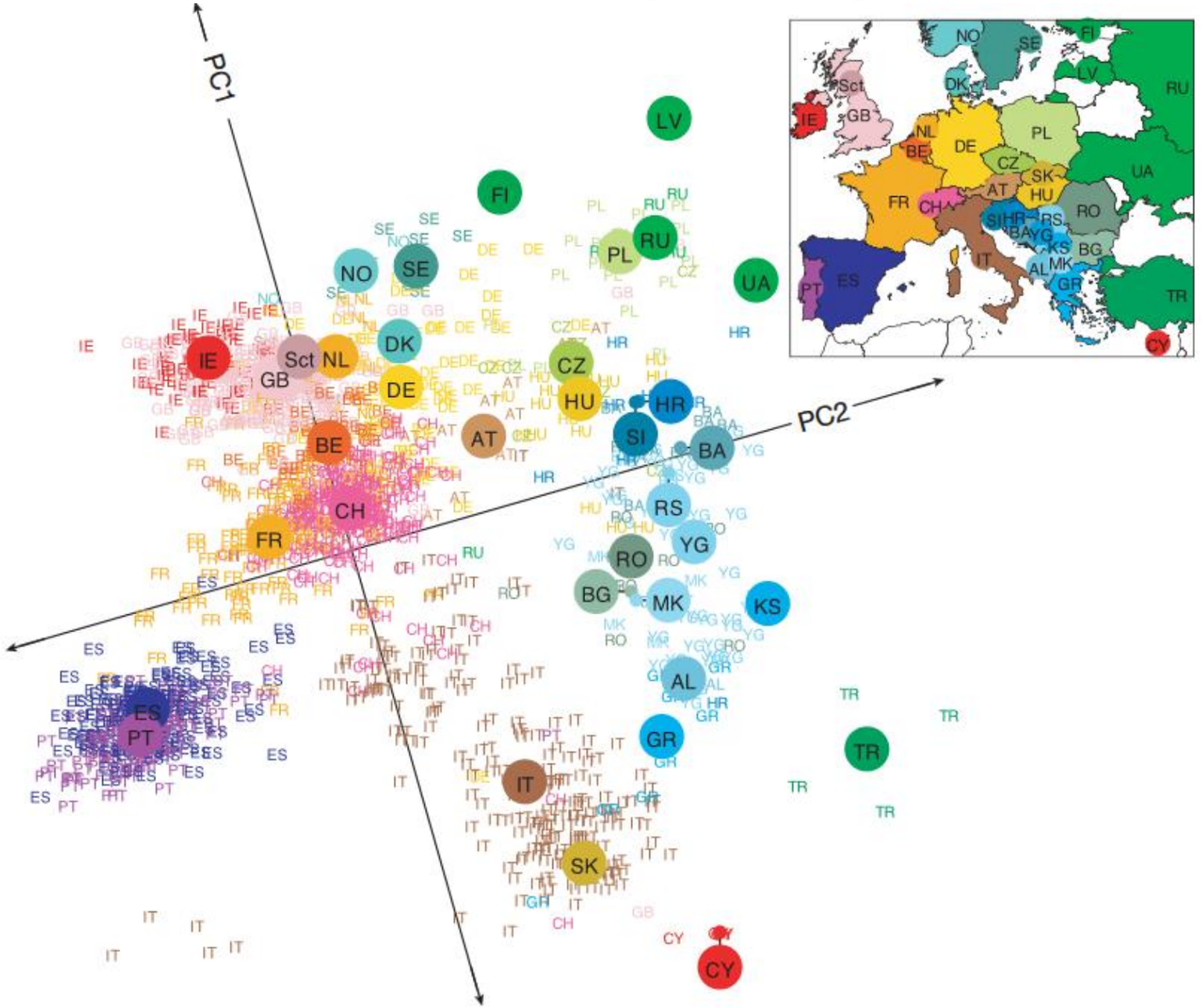
Минутка кода




```
X = X - X.mean(axis=0)
U, L, V = svd(X)
from sklearn.decomposition import PCA
pca_transformer = PCA()
X2 = pca_transformer.fit_transform(X)
```

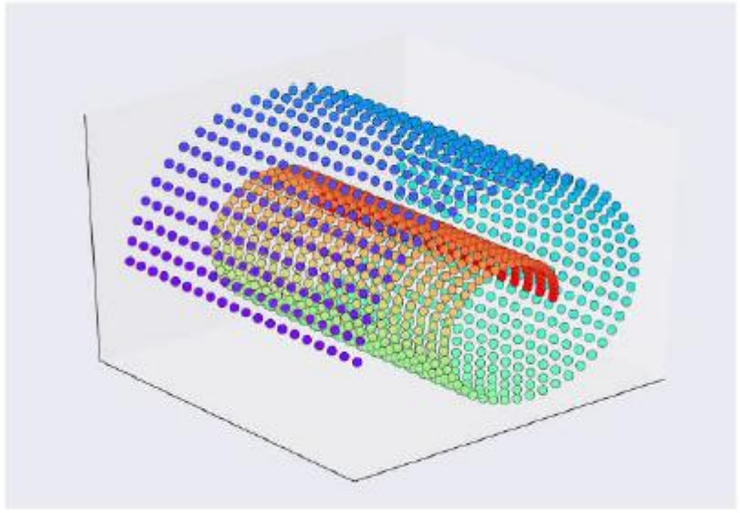
```
plt.scatter(X[:, 0], X[:, 1])
plt.scatter(X2[:, 0], X2[:, 1])
# ~ L[0]*U[:, 0], L[1]*U[:, 1]
plt.scatter(5*U[:, 0], 5*U[:, 1])
```

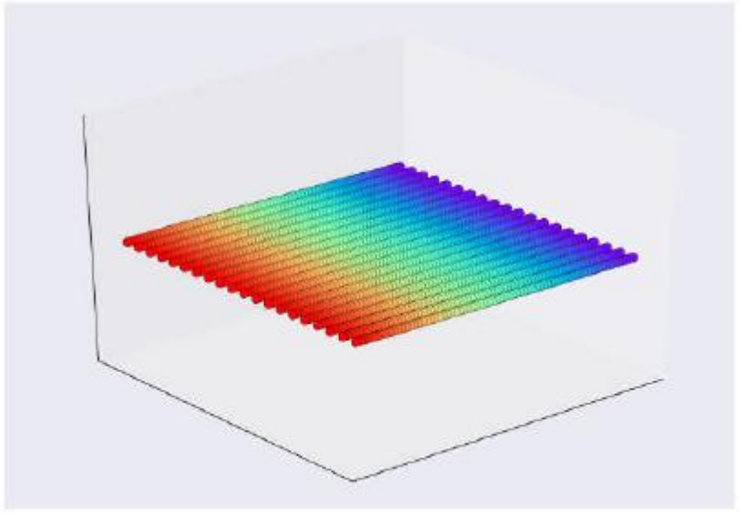
PCA: генотипы народов Европы



https://www.nature.com/articles/nature07331

Нелинейное сокращение размерности





тема связанная с вложением в поверхности (Manifold Learning)

ясно, что метрики здесь не описывают адекватно близость

мешаются «средние расстояния»

Нелинейное сокращение размерности: способы

- kernel PCA

Manifold based methods

- LLE (Locally Linear Embedding)
- SNE (Stochastic Neighbor Embedding)
- t-SNE (t-distributed Stochastic Neighbor Embedding)
- IsoMap (Isometric Mapping)
- MDS (MultiDimensional Scaling)
- Maximum Variance Unfolding
- Spectral Embedding / Laplacian Eigenmap

нейросетевые

- autoencoder

Kernel PCA

Обычный PCA: после центрирования векторов ищем с.в. матрицы ковариаций (covariance matrix):

$$S = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^{\mathrm{T}}$$

Теперь по аналогии ищем с.в. матрицы:

$$S = \frac{1}{m}\sum_{i=1}^{m} \varphi(x_i)\varphi(x_i)^{\mathrm{T}}$$

под суммой стоит не $K(x_i, x_i)$, т.к. там не скалярное произведение, а внешнее

$$Su_t = \lambda u_t \qquad \frac{1}{m}\sum_{i=1}^{m} \varphi(x_i)\varphi(x_i)^{\mathrm{T}} u_t = \lambda u_t$$

тогда с.в. представимы в виде

$$u_t = \sum_{j=1}^{m} a_{jt}\varphi(x_j)$$

Kernel PCA

Подставим с.в. в выражение его определения:

$$\frac{1}{m}\sum_{i=1}^{m}\varphi(x_i)\varphi(x_i)^{\mathrm{T}}\sum_{j=1}^{m}a_{jt}\varphi(x_j)=\lambda\sum_{j=1}^{m}a_{jt}\varphi(x_j)$$

умножим на $\varphi(x_s)^{\mathrm{T}}$**, учтём обозначение** $K(x_i,x_j)=\varphi(x_i)^{\mathrm{T}}\varphi(x_j)$

$$\frac{1}{m}\sum_{i=1}^{m}\varphi(x_s)^{\mathrm{T}}\varphi(x_i)\varphi(x_i)^{\mathrm{T}}\sum_{j=1}^{m}a_{jt}\varphi(x_j)=\lambda\sum_{j=1}^{m}a_{jt}\varphi(x_s)^{\mathrm{T}}\varphi(x_j)$$

$$\frac{1}{m}\sum_{i=1}^{m}\varphi(x_s)^{\mathrm{T}}\varphi(x_i)\sum_{j=1}^{m}a_{jt}\varphi(x_i)^{\mathrm{T}}\varphi(x_j)=\lambda\sum_{j=1}^{m}a_{jt}\varphi(x_s)^{\mathrm{T}}\varphi(x_j)$$

$$\frac{1}{m}\sum_{i=1}^{m}K(x_s,x_i)\sum_{j=1}^{m}a_{jt}K(x_i,x_j)=\lambda\sum_{j=1}^{m}a_{jt}K(x_s,x_j)$$

Kernel PCA

Пусть есть матрица $K = \| K(x_i, x_j) \|_{m \times m}$**, тогда полученное уравнение эквивалентно:**

$$K^2 a_t = \lambda m K a_t$$

$$K a_t = \lambda m a_t$$

т.е. получили аналогичную задачу на с.в. ~ kernel PCA

тут не надо вычислять $\varphi(x_i)$

Kernel PCA

Как и в PCA эту матрицу надо нормировать…

можно показать, что это делается так:

$$\tilde{K} = (I - E/m)K(I - E/m) =$$

$$= K - \left\|\frac{1}{m}\right\| K - K\left\|\frac{1}{m}\right\| + \left\|\frac{1}{m}\right\| K \left\|\frac{1}{m}\right\|$$

– центрированная матрица ядра (centered kernel matrix)

Пусть $a_1,\ldots,a_k$ **– с.в., соответствущие максимальным с.з. матрицы** $\tilde{K}$ **тогда**

$$x \to (z_1,\ldots,z_k)$$

$$z_t = \varphi(x)^{\mathrm{T}} u_t$$

$$z_t = \varphi(x)^{\mathrm{T}} u_t = \sum_{j=1}^{m} a_{tj} \varphi(x)^{\mathrm{T}} \varphi(x_j) = \sum_{j=1}^{m} a_{tj} K(x, x_j)$$

kernel PCA

1. Построить $m \times m$**-матрицу** K**.**

2. Центрировать, получить матрицу $\tilde{K}$

3. Найти наибольших с.з. и соответствующие с.в.

$$a_1, \ldots, a_k$$

4. Перенормировать их:

$$a'_s = \frac{a_s}{\sqrt{\lambda_s m}}$$

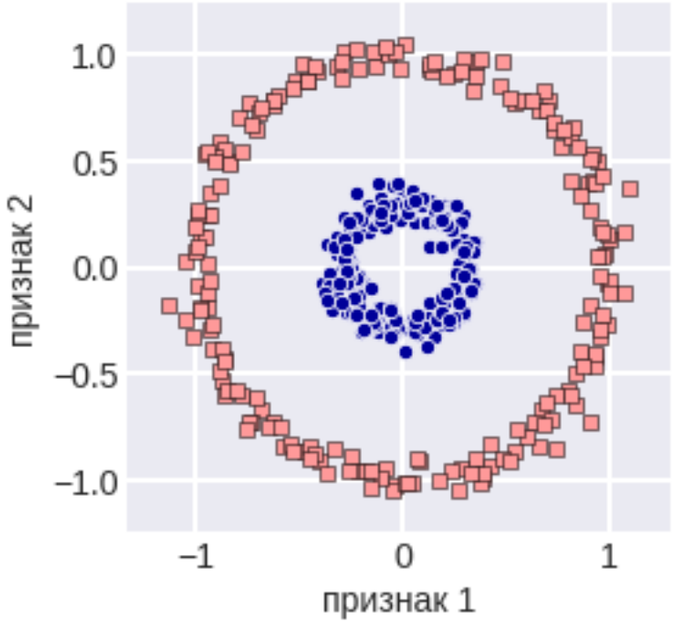
5. Выполнить вложение:

$$x \to (z_1, \ldots, z_k)$$

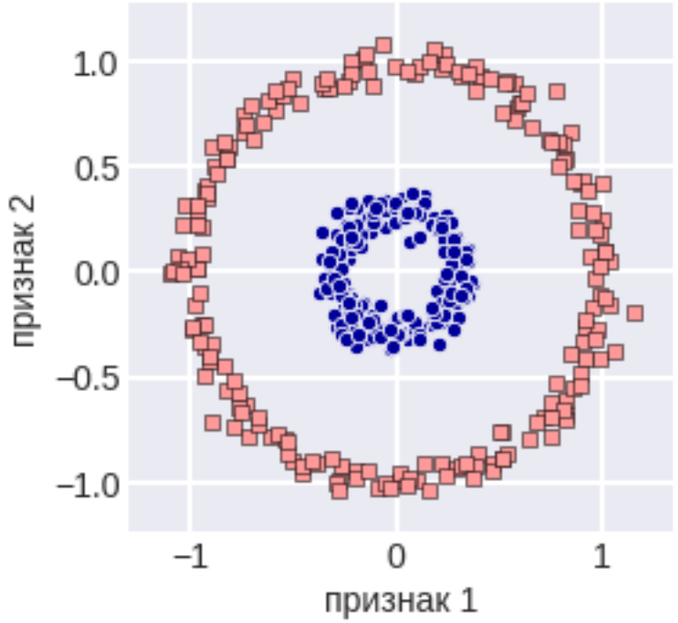
$$z_t = \sum_{j=1}^{m} a_{tj} K(x, x_j)$$

kernel PCA: примеры

данные



PCA



kernel PCA

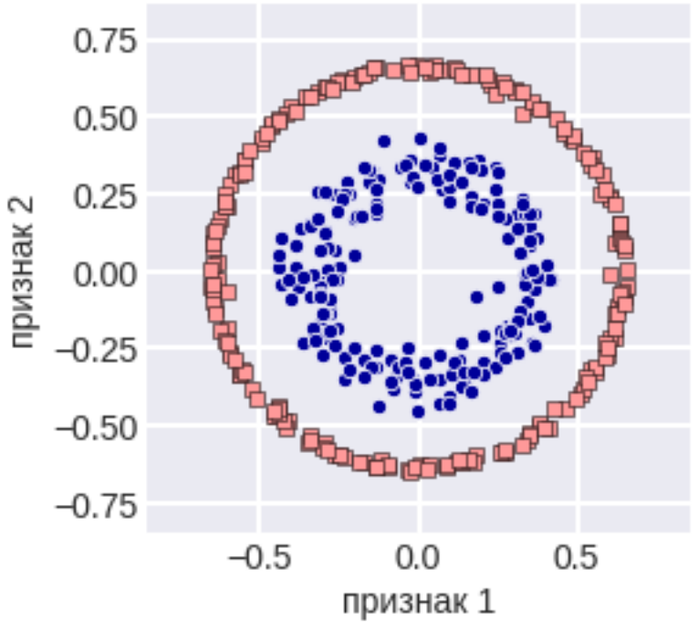

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel="rbf", gamma=1, random_state=1)
X_kpca = kpca.fit_transform(X)
```

есть возможность учить и обратное преобразование `fit_inverse_transform=True`

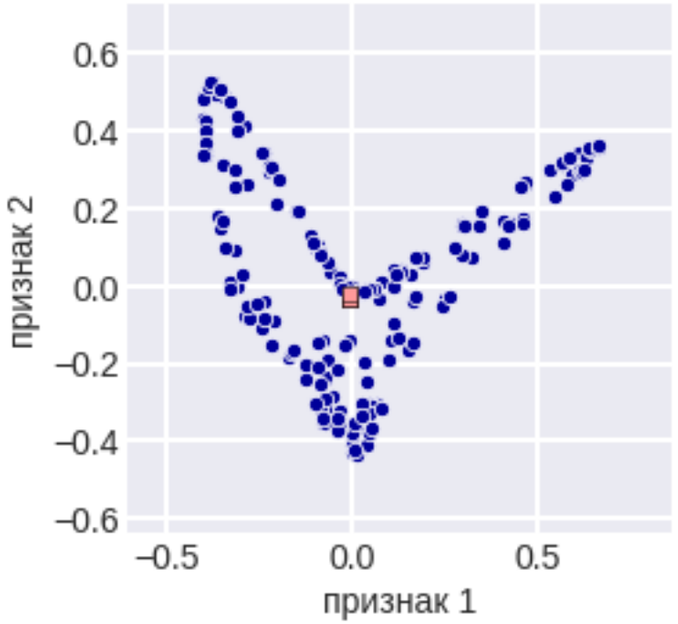
пример кода: https://scikit-learn.org/stable/auto_examples/decomposition/plot_kernel_pca.html

kernel PCA: примеры

gamma=1

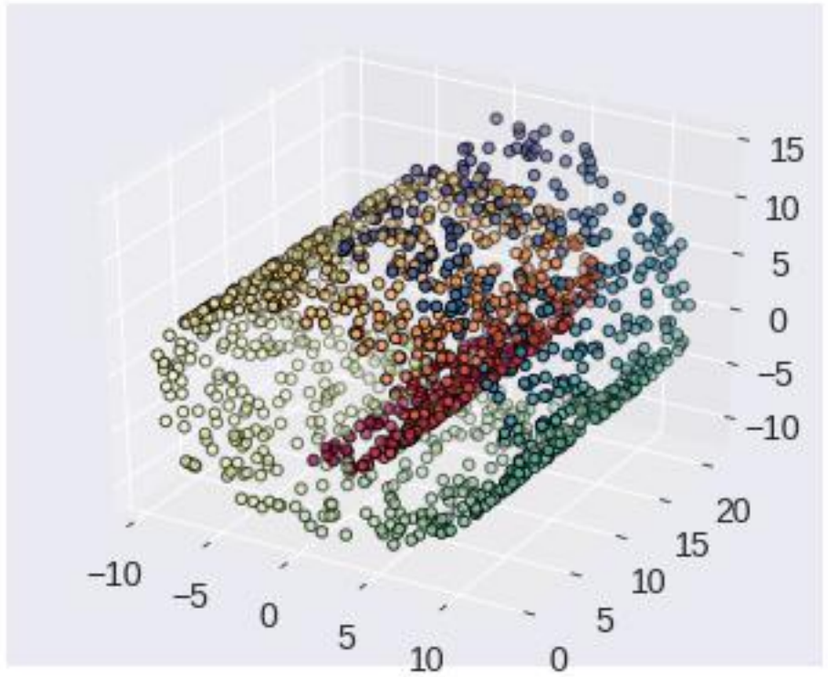


gamma=10

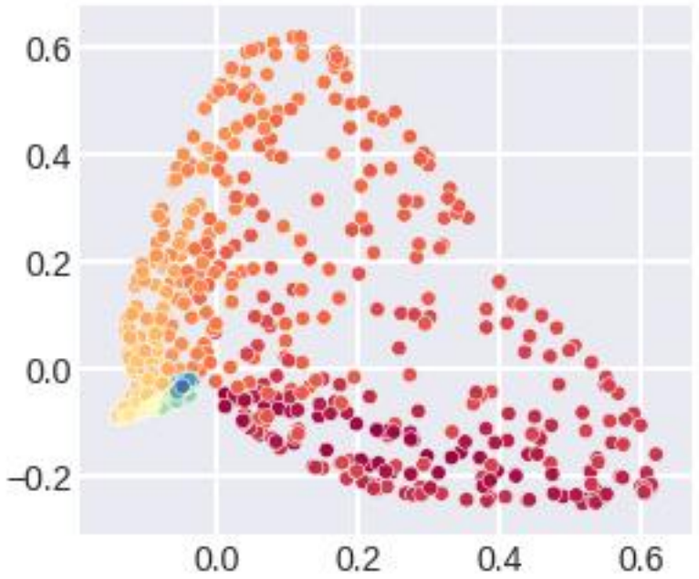


gamma=100

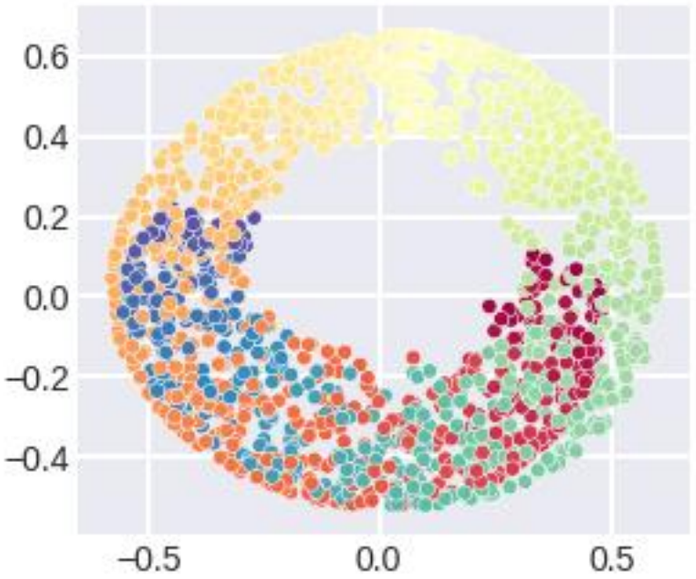
kernel PCA: примеры



gamma=0.1



gamma=0.01



kernel PCA: устранение шума

данные:

данные + гауссовский шум

линейный PCA

RBF kernel PCA

http://www.cs.haifa.ac.il/~rita/uml_course/lectures/KPCA.pdf

kernel PCA: свойства

– не всегда удобен для визуализации

– не всегда получается желаемое…

+ нелинейное сокращение размерности (и преобразование пространства)

+ может быть полезен для генерации признаков

Локально линейные преобразования – Locally Linear Embedding (LLE)

Гипотеза – локальная линейность

любая поверхность в малой окрестности линейная

близкие точки в исходном пространстве остаются близкими в итоговом

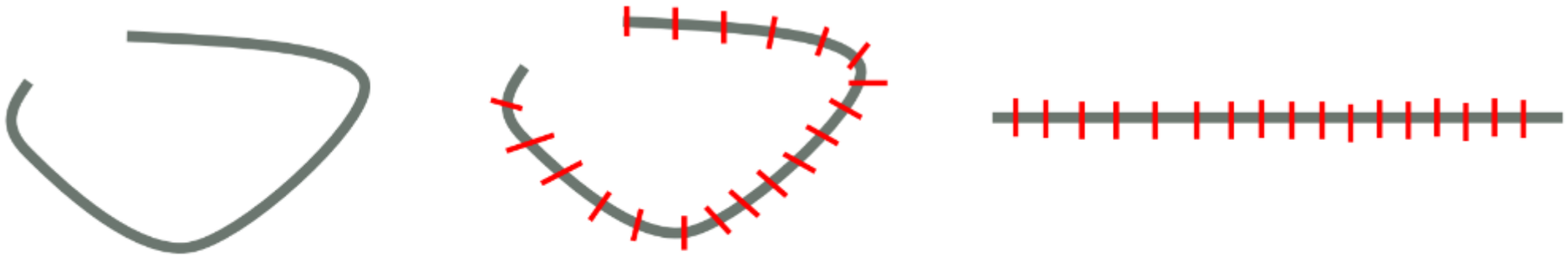


Figure 1. Piece-wise local unfolding of manifold by LLE (in this example from two dimensions to one intrinsic dimension). This local unfolding is expected to totally unfold the manifold properly.

«Locally Linear Embedding and its Variants: Tutorial and Survey»
https://arxiv.org/pdf/2011.10925.pdf

Локально линейные преобразования – Locally Linear Embedding (LLE)

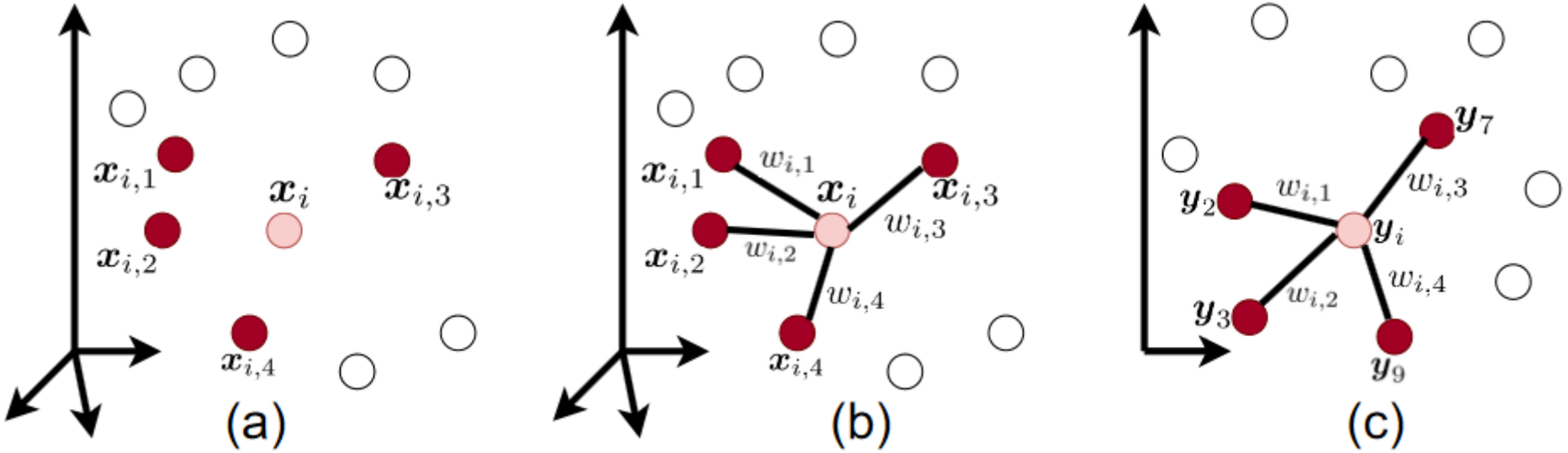


Figure 2. Steps in LLE for embedding high dimensional data in a lower dimensional embedding space: (a) finding k-nearest neighbors, (b) linear reconstruction by the neighbors, and (c) linear embedding using the calculated weights. In this figure, it is assumed that $k = 4$, $\boldsymbol{x}_{i,1} = \boldsymbol{x}_2$, $\boldsymbol{x}_{i,2} = \boldsymbol{x}_3$, $\boldsymbol{x}_{i,3} = \boldsymbol{x}_7$, and $\boldsymbol{x}_{i,4} = \boldsymbol{x}_9$.

+ понятная геометрия (локальный PCA)

+ есть кернализованные варианты

Локально линейные преобразования – Locally Linear Embedding (LLE)

1. Для каждой точки находим её k ближайших соседей

$$x_i \to x_{i1}, \ldots, x_{ik}$$

2. Вычисляем матрицу реконструкции W

$$\| w_{ij} \| = \arg\min_{w_{ij}: \forall i \sum_j w_{ij} = 1} \sum_{i=1}^{m} \left\| x_i - \sum_{j=1}^{k} w_{ij} x_{ij} \right\|_2^2$$

3. Вложение

$$\sum_{i=1}^{m} \left\| z_i - \sum_{j=1}^{k} w_{ij} z_{ij} \right\|_2^2 \to \min_{\{z_i\}}$$ **при условии** $$\frac{1}{m} \sum_{i=1}^{m} z_i z_i^{\mathrm{T}} = I, \sum_{i=1}^{m} z_i = 0$$

те же веса, но в пространстве меньшей размерности

sklearn.manifold.LocallyLinearEmbedding

n_neighbors=5 – **число соседей**

n_component=2 – **размерность итогового пространства**

reg=1e-3 – **регуляризация (multiplies the trace of the local covariance distance matrix)**

eigen_solver – **метод поиска с.в. {‘auto’, ‘arpack’, ‘dense’}**

tol – **tolerance для сходимости при вычислении с.в.**

max_iter=100 – **ограничение на число итераций**

method – **метод**

- standard – **обычный LLE**
- hessian – **Hessian eigenmap method**
- modified – **+регуляризация**
- ltsa –**local tangent space alignment algorithm**

hessian_tol=1e-12 – **Tolerance для Hessian eigenmapping method**

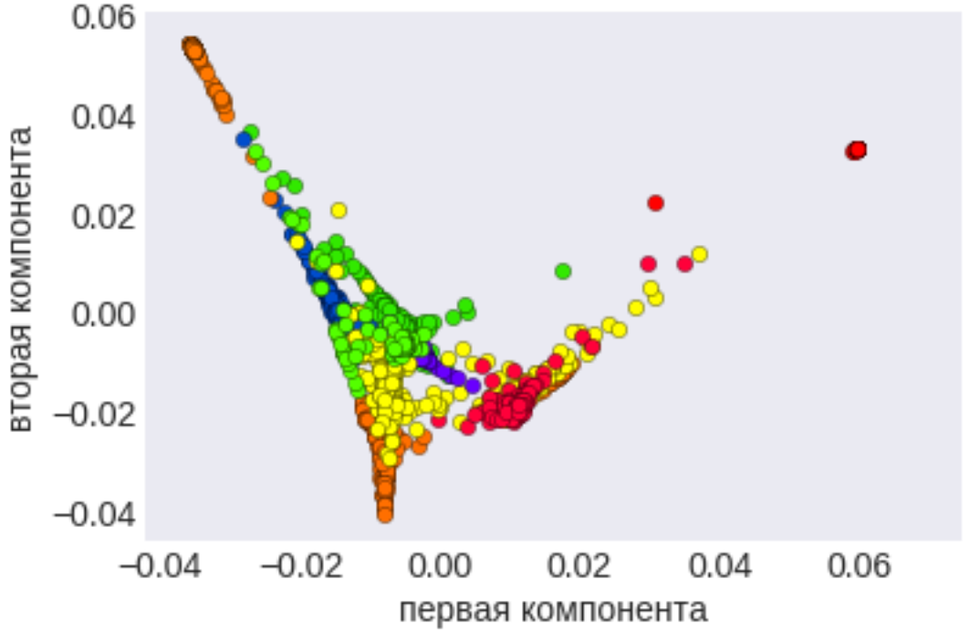
modified_tol – **Tolerance для modified LLE method**

neighbors_algorithm – **для поиска БС {‘auto’, ‘brute’, ‘kd_tree’, ‘ball_tree’}**

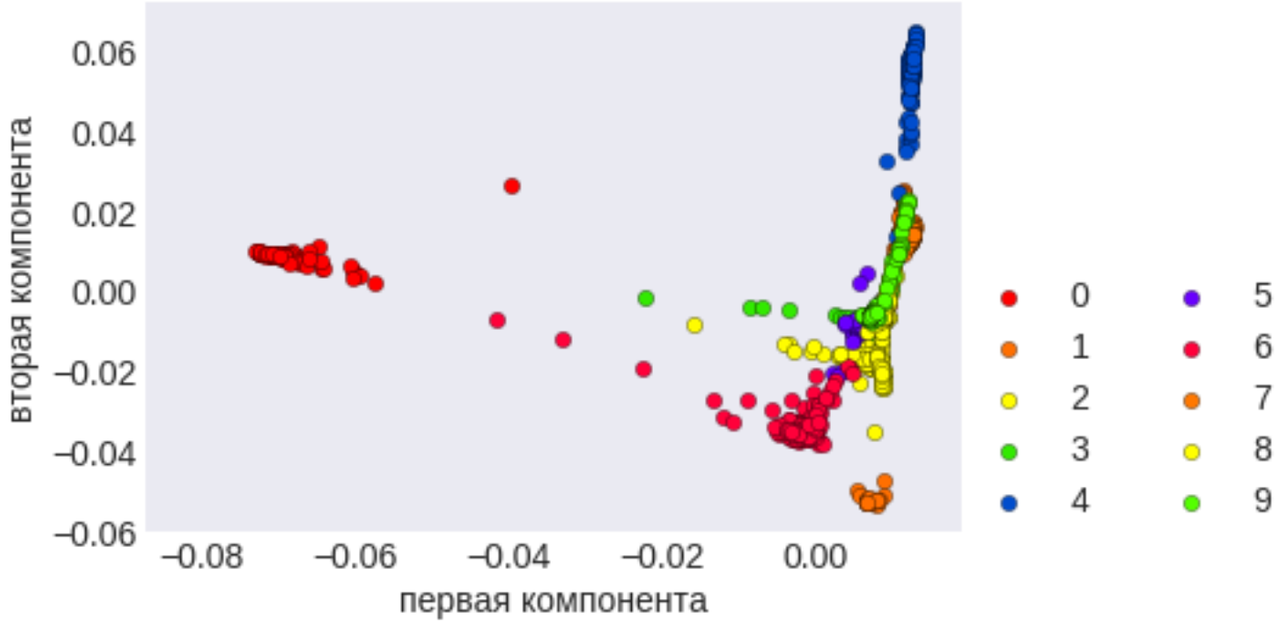
random_state

n_jobs

Locally Linear Embedding (LLE) на датасете «Digits»



`method='modified'`



`n_neighbors=10`

остальные модификации – хуже

SNE (Stochastic Neighbor Embedding)

1. Превращаем евклидово расстояние в

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

2. Откуда взять σ_i^2 – своя для каждой точки, идея – она будет зависеть от плотности точек

будем задаваться параметром «перплексия»

$$\text{perplexity} = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}}$$

отсюда подбором решая равенство определяем σ_i^2

3. Отображаем $\{x_i\} \to \{z_i\}$ в пространство, в котором

$$q_{j|i} = \frac{\exp(-\| z_i - z_j \|^2 / (2\sigma_i^2))}{\sum_{t \neq i} \exp(-\| z_i - z_t \|^2 / (2\sigma_i^2))}$$

будем минимизировать $\text{KL}(p_{\circ|\circ}, q_{\circ|\circ})$

t-SNE (t-distributed Stochastic Neighbor Embedding)

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

«сходство при фиксации соседней точки»

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

Считаем, что $p_{ii} = 0$

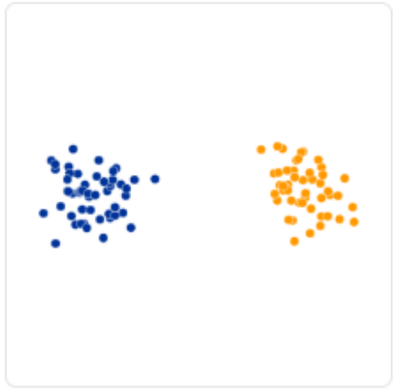
Используем распределение Стьюдента
(у него тяжелее хвосты)

$$q_{ij} = \frac{\frac{1}{1 + \| z_i - z_j \|^2}}{\sum_{t \neq i} \frac{1}{1 + \| z_i - z_t \|^2}}$$

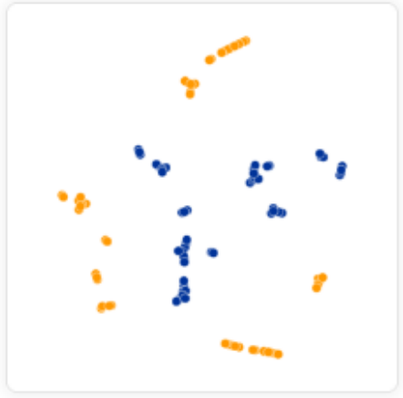
$$\mathrm{KL}(P \| Q) \to \min$$

градиент аналитически вычисляется

Примеры использования t-SNE

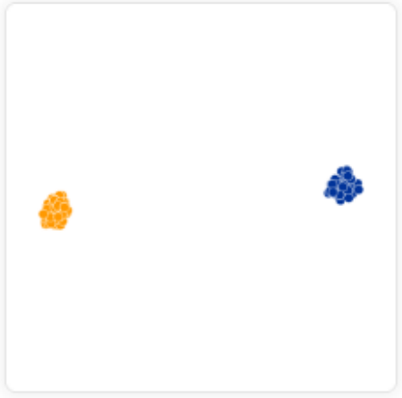


Original

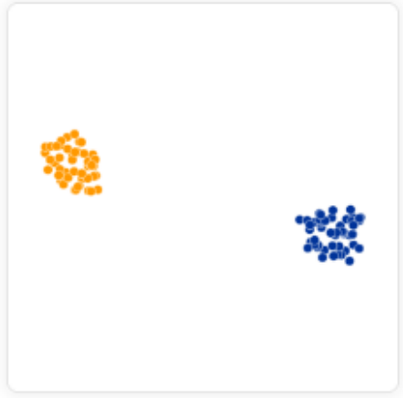


Perplexity: 2
Step: 5,000

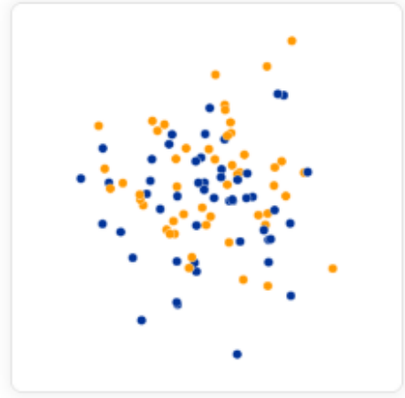
Perplexity: 5
Step: 5,000



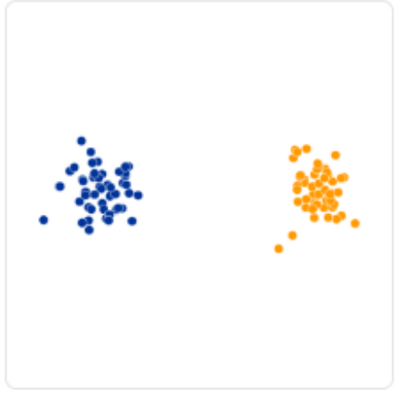
Perplexity: 30
Step: 5,000



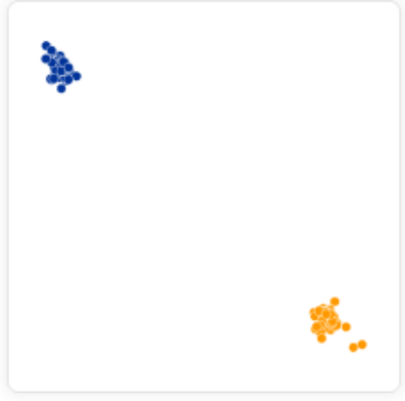
Perplexity: 50
Step: 5,000



Perplexity: 100
Step: 5,000



Original

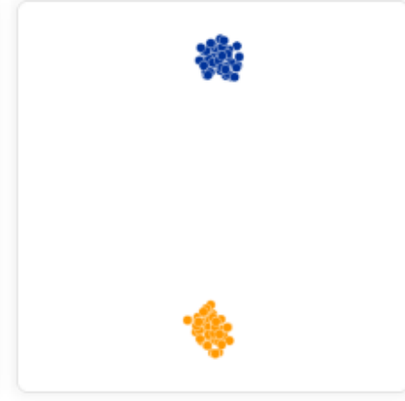


Perplexity: 30
Step: 10

Perplexity: 30
Step: 20

Perplexity: 30
Step: 60

Perplexity: 30
Step: 120

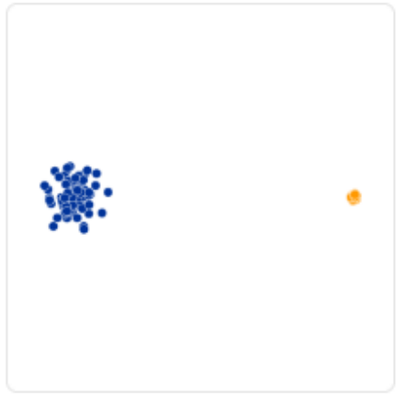


Perplexity: 30
Step: 1,000

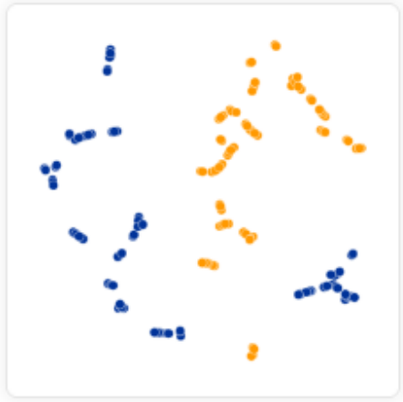
запуски с разными (!) начальными инициализациями

https://distill.pub/2016/misread-tsne/

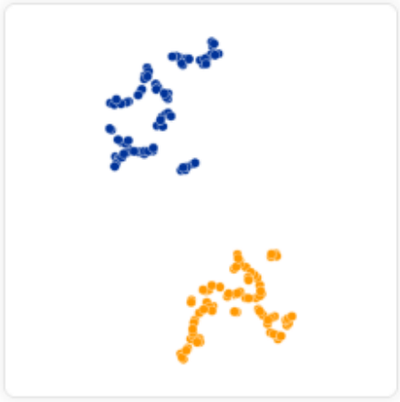
Примеры использования t-SNE



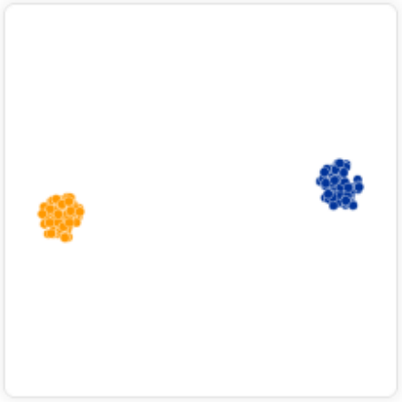
Original



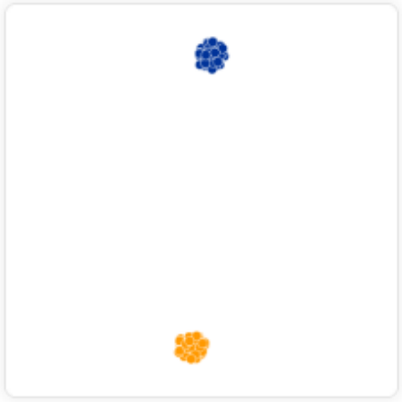
Perplexity: 2
Step: 5,000



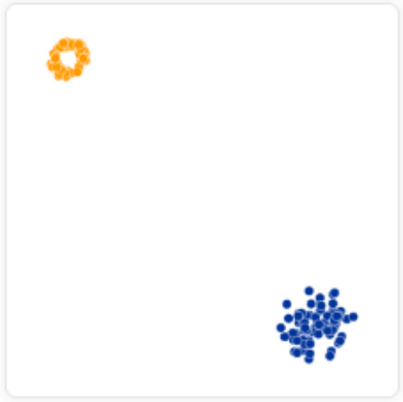
Perplexity: 5
Step: 5,000



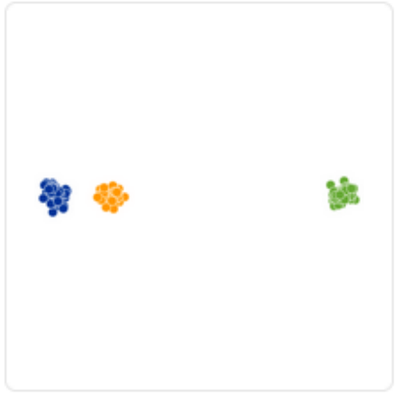
Perplexity: 30
Step: 5,000



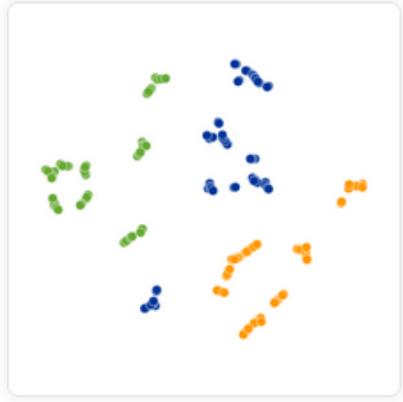
Perplexity: 50
Step: 5,000



Perplexity: 100
Step: 5,000

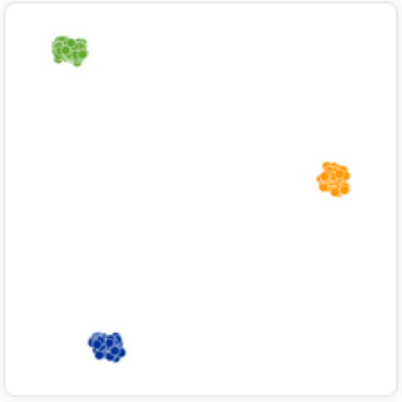


Original



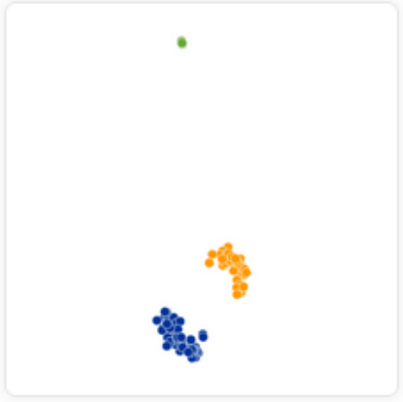
Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000



Perplexity: 30
Step: 5,000

Perplexity: 50
Step: 5,000



Perplexity: 100
Step: 5,000

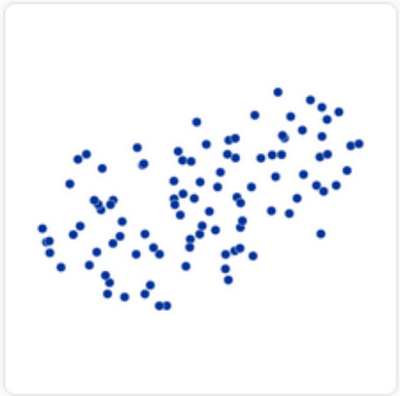
Примеры использования t-SNE

Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000



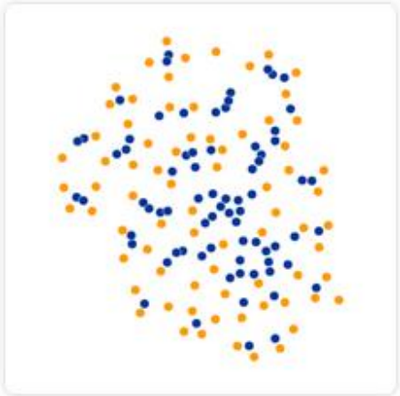
Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

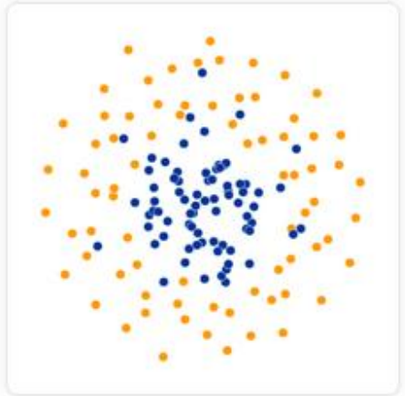
можно видеть закономерности в шуме

Original

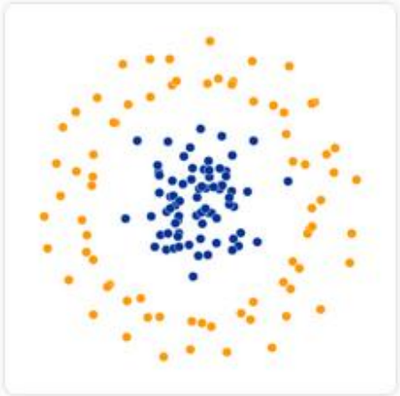
Perplexity: 2
Step: 5,000



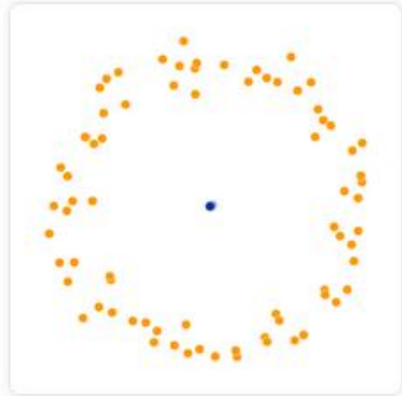
Perplexity: 5
Step: 5,000



Perplexity: 30
Step: 5,000



Perplexity: 50
Step: 5,000



Perplexity: 100
Step: 5,000

t-SNE (t-distributed Stochastic Neighbor Embedding)

перплексия определяет кластеры какого масштаба доминируют

не всегда сохраняет топологию

лучше делать несколько визуализаций

– нет глобальной структуры

хорошо инициализировать с помощью PCA

– скорость

– стохастический (результат не определён однозначно)

– не совсем ясная интерпретация

– нет понятия оптимальной размерности пространства

– сложности с новыми данными

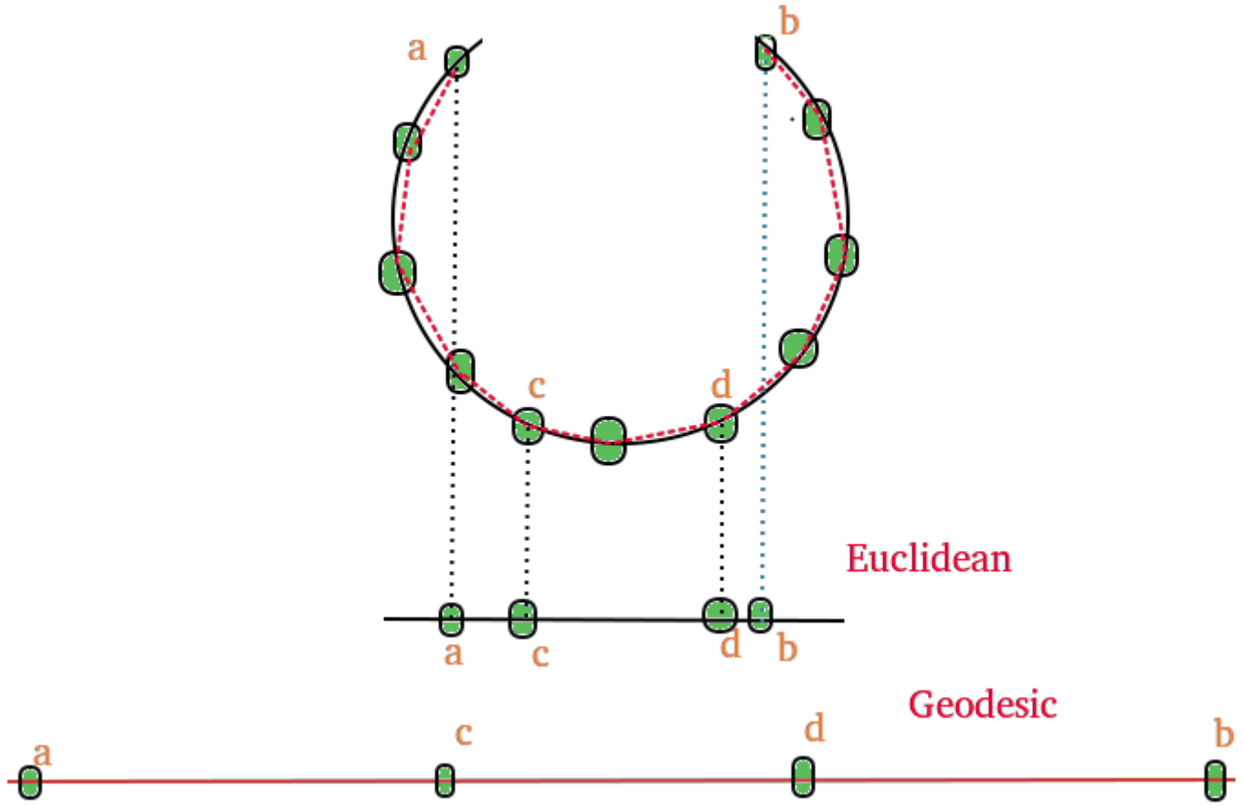
IsoMap (Isometric Mapping)

нелинейное сокращение размерности на спектральной теории,
сохраняя геодезические расстояния

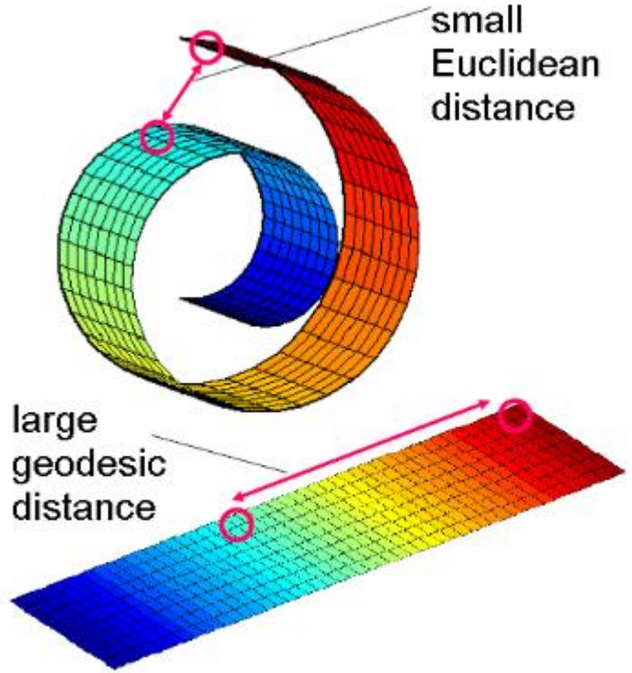
Вход: матрица данных
Строим граф *k*-соседства или ε-соседства
Вычисляем геодезическое расстояние между парами всех точек (кратчайший путь)
используем локальную информацию для восстановления глобальной (отличие от LLE)
Выполняем MDS (медленно!) – подробнее дальше

хорошо, если в данных «нет дырок» (т.е. более-менее плотные окрестности)

IsoMap (Isometric Mapping)



https://blog.paperspace.com/dimension-reduction-with-isomap/



Борьба с неадекватными «средними расстояниями»

http://www.cs.cmu.edu/~bapoczos/Classes/ML10715_2015Fall/slides/ManifoldLearning.pdf

IsoMap (Isometric Mapping)

`sklearn.manifold.Isomap`

`n_neighborsint=5` – **число соседей**

`n_components=2` – **размерность итогового пространства**

`eigen_solver` – **соловер** `{'auto', 'arpack', 'dense'}`

`tol` – **tolerance (контроль сходимости при вычислении с.в.)**

`max_iterint` – **ограничение на число итераций при вычислении с.в.**

`path_method` – **метод для поиска кратчайшего пути**

`neighbors_algorithm` – **метод поиска БС** `{'auto', 'brute', 'kd_tree', 'ball_tree'}`

`n_jobs`

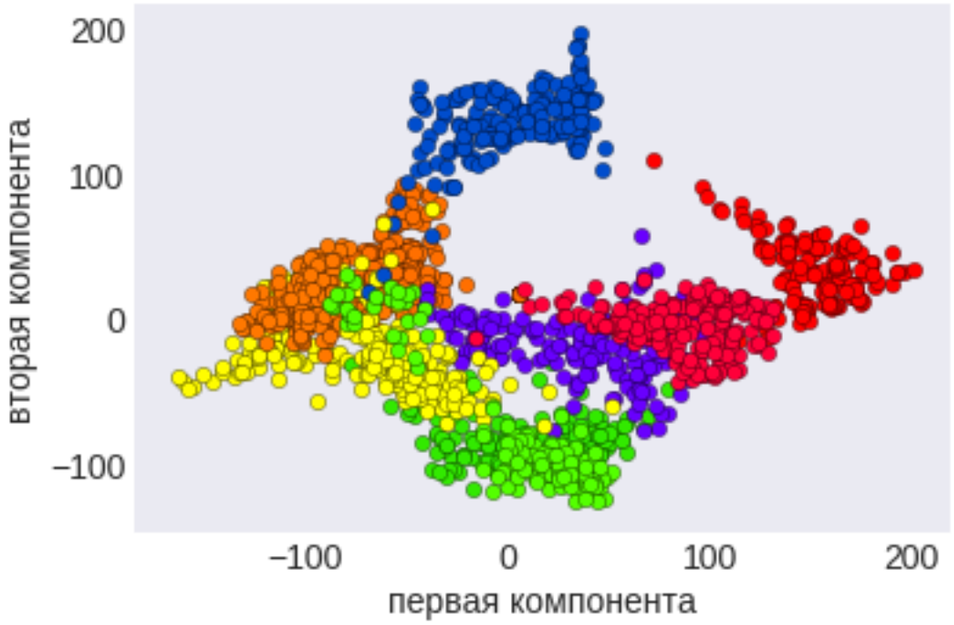
`metric="minkowski"` – **метрика**

`p` – **степень в расстоянии Минковского**

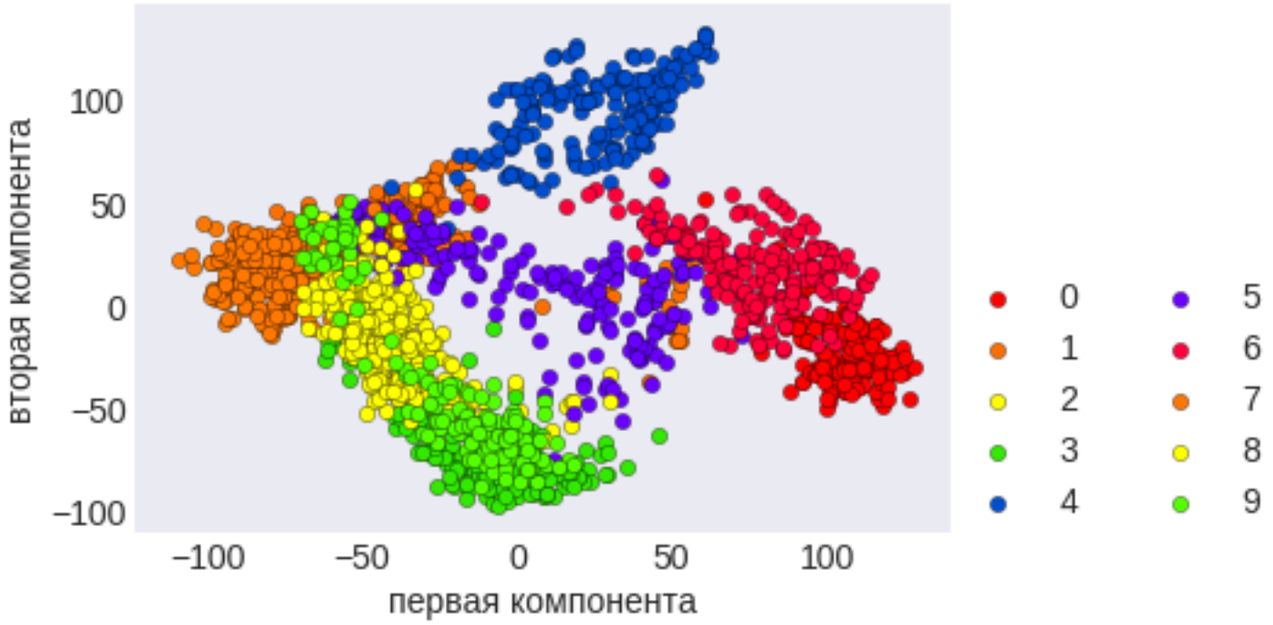
`metric_params` – **параметры ф-ии расстояния**

IsoMap (Isometric Mapping): датасет «Digits»

`n_neighborsint=5`



`n_neighborsint=10`



MDS (MultiDimensional Scaling)

классический алгоритм

ищем представление, в котором сохраняются расстояния

1. Пусть $D^{(2)} = \| d_{ij}^2 \|_{m\times m}$ **матрица квадратов эвклидовых расстояний**

2. Двойное центрирование

$$B = \frac{1}{2} C D^{(2)} C, \quad C = I - \frac{1}{n} E$$

есть вариант $B = XX^{\mathrm{T}}$ **(предполагая центрированность данных)**

3. Для матрицы B **находим наибольших** k **с.з.** $\lambda_1, \ldots, \lambda_k$ **и их с.в. векторов** $v_1, \ldots, v_k$

4. Новая признаковая матрица

$$VL^{1/2}, \quad L = \mathrm{diag}(\lambda_1, \ldots, \lambda_k), \quad V = [v_1, \ldots, v_k]_{m\times k}$$

MDS (MultiDimensional Scaling)

в общем случае, если $d_{ij} = \| x_i - x_j \|_2$, $\delta_{ij} = \| z_i - z_j \|_2$

минимизируем

strain	$\frac{1}{N} \sum_{1 \le i < j \le m} w_{ij} (\delta_{ij}^2 - d_{ij}^2)^2$
stress	$\frac{1}{N} \sum_{1 \le i < j \le m} w_{ij} (\delta_{ij} - d_{ij})^2$
Sammon's stress	$\sum_{1 \le i < j \le m} \frac{(\delta_{ij} - d_{ij})^2}{\delta_{ij}} \Big/ \sum_{1 \le i < j \le m} \delta_{ij}$

есть алгоритм SMACOF для минимизации взвешенного напряжения

sklearn.manifold.MDS

`n_components=2` – **размерность итогового пространства**

`metric=True` – **сохранять ли значения метрик или порядок (большие значения в большие)**

`n_init=4` – **число запусков SMACOF-алгоритма с разными инициализациями (выбирается лучший ответ)**

`max_iter=300` – **число итераций SMACOF**

`verbose=0` –**verbosity**

`eps=1e-3` – **tolerance**

`n_jobs`

`random_state`

`dissimilarity` – **режим**

- `euclidean` – **эвлидова метрика**
- `precomputed` – **передаём в fit**

Maximum Variance Unfolding

Строим граф (V,E) соседства (kNN или ε) нужно по нему построить отображение

$$x_i \to z_i$$

которое сохраняет расстояния соседей $(i,j) \in E$

$$\| x_i - x_j \|^2 = \| z_i - z_j \|^2$$

и при этом максимизируем разброс

$$\frac{1}{m}\sum_i \| z_i - \overline{z} \|^2 \to \max$$

нет в sklearn

Maximum Variance Unfolding

Пусть (по другому ориентируем матрицы):

$$X = [x_1, \ldots, x_m] \in \mathbb{R}^{n \times m},\ P = X^{\mathrm{T}} X$$

$$Z = [z_1, \ldots, z_m] \in \mathbb{R}^{k \times m},\ Q = Z^{\mathrm{T}} Z$$

идея – найти Q и над ней PCA (поэтому + ограничение неотрицательной определённости)

из $\| x_i - x_j \|^2 = \| z_i - z_j \|^2$ **получаем** $Q_{ii} - 2Q_{ij} + Q_{jj} = P_{ii} - 2P_{ij} + P_{jj}$

это ограничения в задаче

что максимизируем – разброс – можно записать как след матрицы $\frac{1}{m} ZZ^{\mathrm{T}} - \frac{1}{m^2} Z\tilde{1}\tilde{1}^{\mathrm{T}} Z^{\mathrm{T}}$

или $\frac{1}{m} \mathrm{tr}(ZZ^{\mathrm{T}}) - \frac{1}{m^2} \underbrace{\mathrm{tr}(Z\tilde{1}\tilde{1}^{\mathrm{T}} Z^{\mathrm{T}})}_{\mathrm{tr}(Z^{\mathrm{T}} Z\tilde{1}\tilde{1}^{\mathrm{T}})} = \frac{1}{m} \mathrm{tr}(Q) - \frac{1}{m^2} \mathrm{tr}(Q\tilde{1}\tilde{1}^{\mathrm{T}})$

Maximum Variance Unfolding

Итоговая задача

$$\frac{1}{m}\text{tr}(Q) - \frac{1}{m^2}\text{tr}(Q\tilde{1}\tilde{1}^{\text{T}}) \to \max$$

$$Q_{ii} - 2Q_{ij} + Q_{jj} = P_{ii} - 2P_{ij} + P_{jj}$$

$$Q \succeq 0$$

Spectral Embedding / Laplacian Eigenmap

1. Строим граф k-соседства

2. Назначаем веса

$$w_{ij} = \begin{cases} \exp\left(-\frac{1}{t}\| x_i - x_j \|^2\right), & (i,j) \in E, \\ 0, & (i,j) \notin E, \end{cases}$$

$$t \to \infty \quad \Rightarrow \quad w_{ij} \to 1 \text{ при } (i,j) \in E$$

Для каждой связной компоненты графа

строим матрицу Лапласа

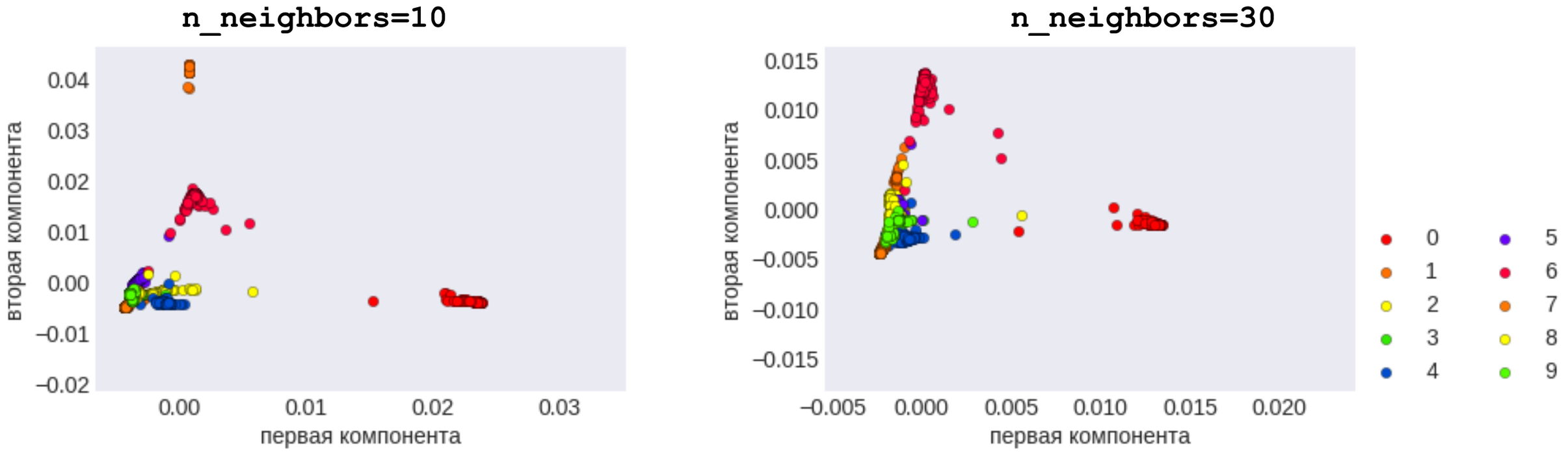
$$L = D - W$$

или для нормализованного случая $L = D^{1/2}(D-W)D^{1/2}$

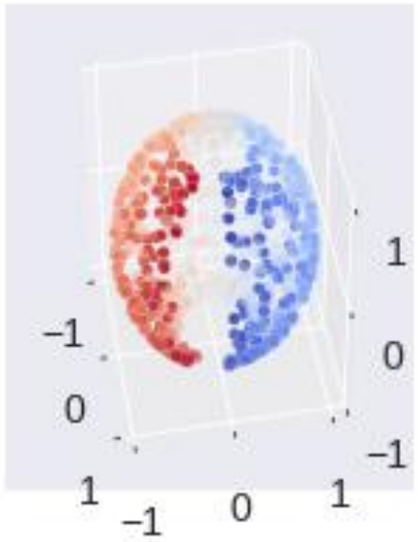
находим r+1 с.в. (соотв. наименьшим с.з.), первая компонента константна

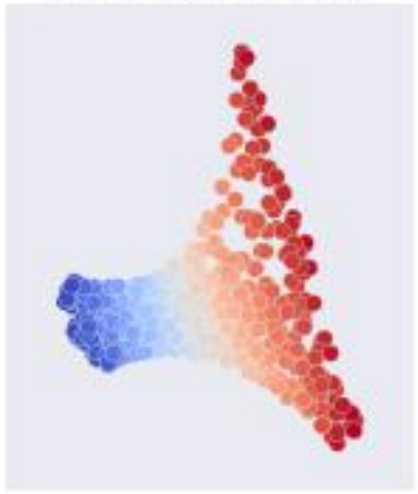
точки отображаем в строки соотв. матрицы $U \in \mathbb{R}^{m \times r}$

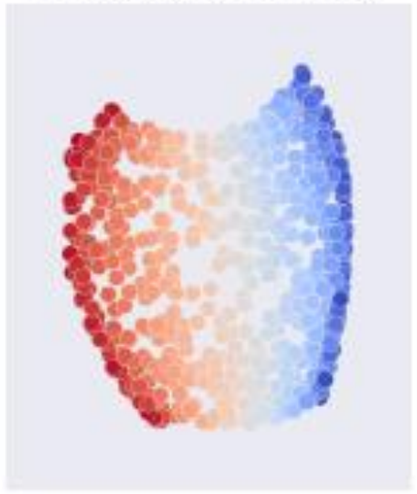
Spectral Embedding / Laplacian Eigenmap

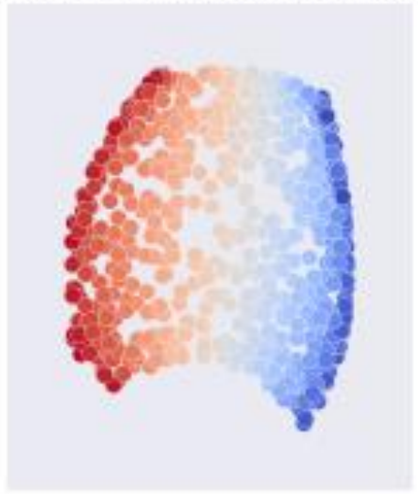


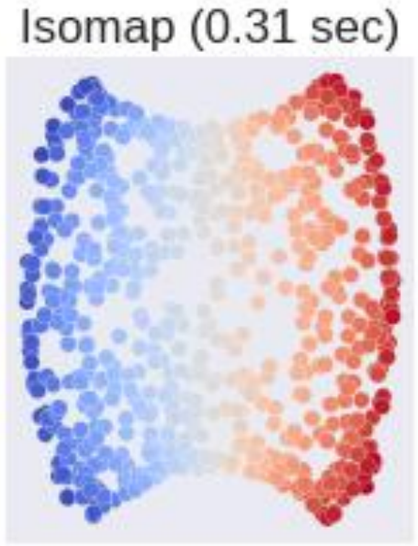
Manifold Learning

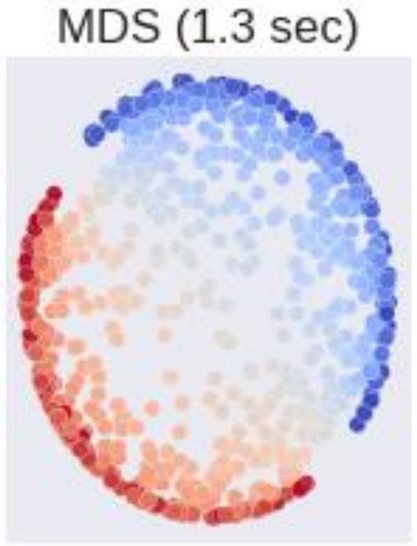


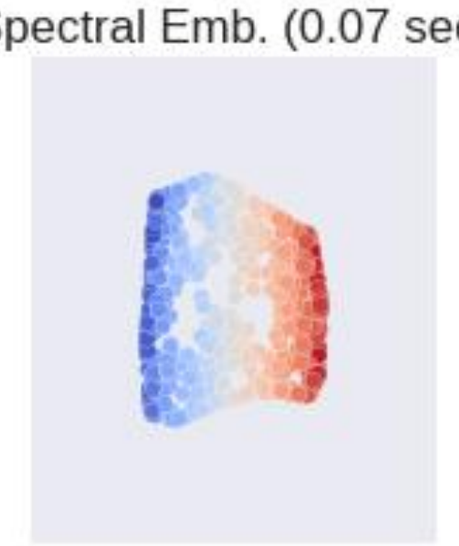




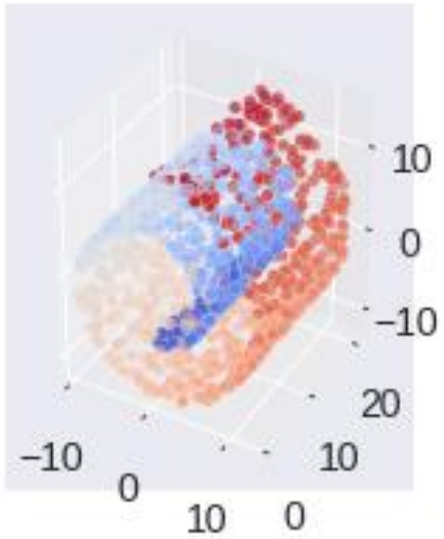


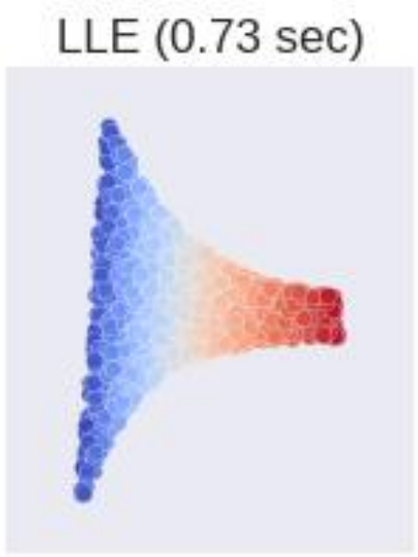


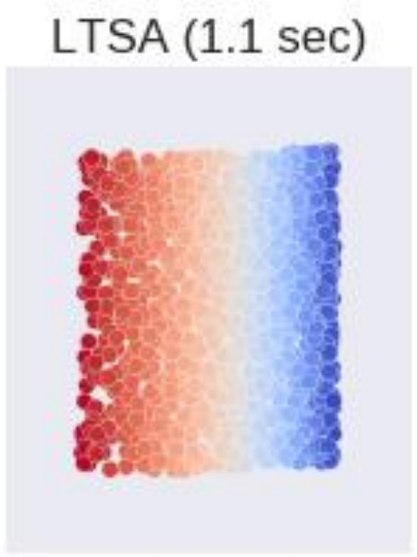


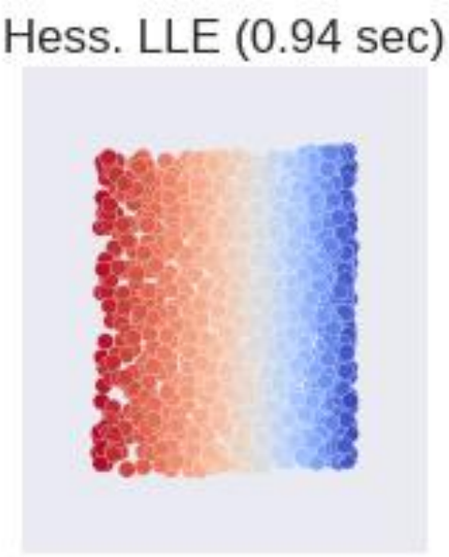


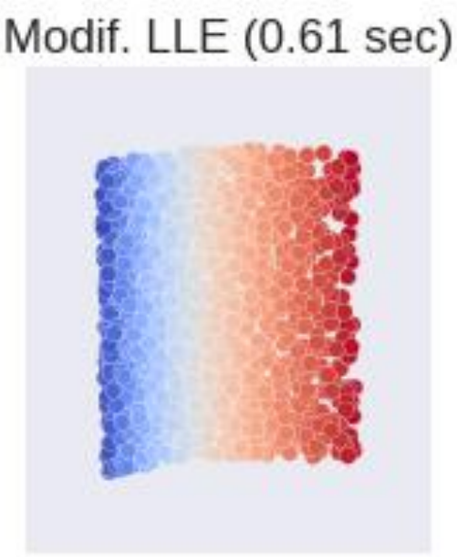
Manifold Learning

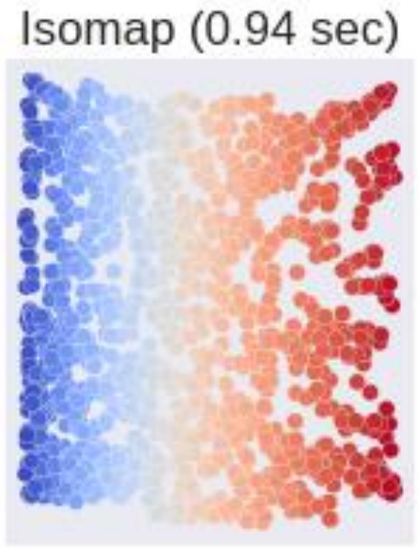


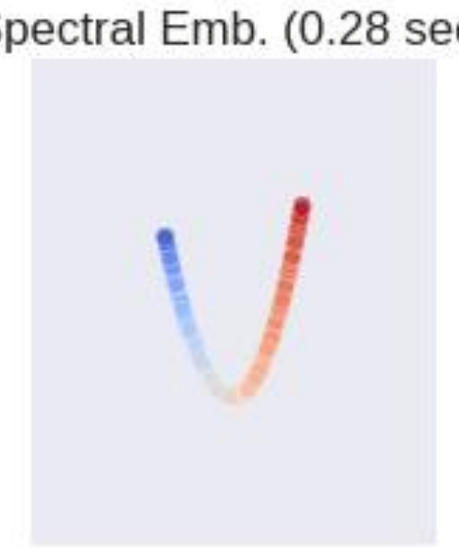






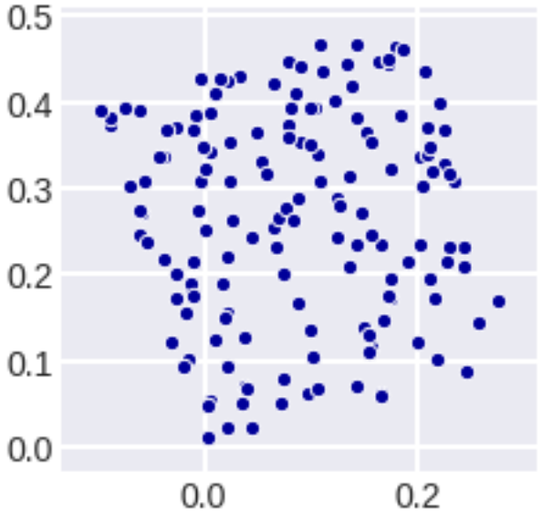


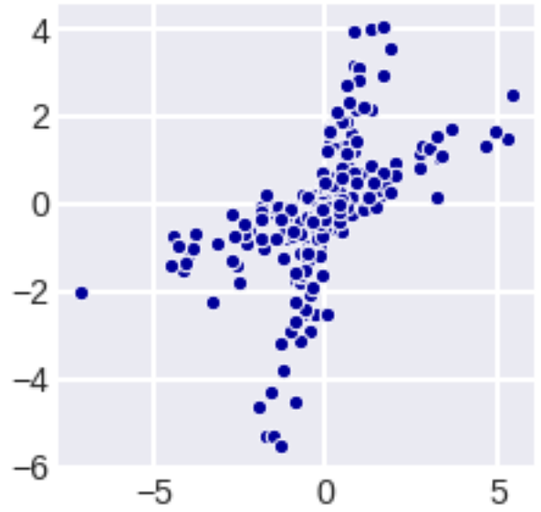




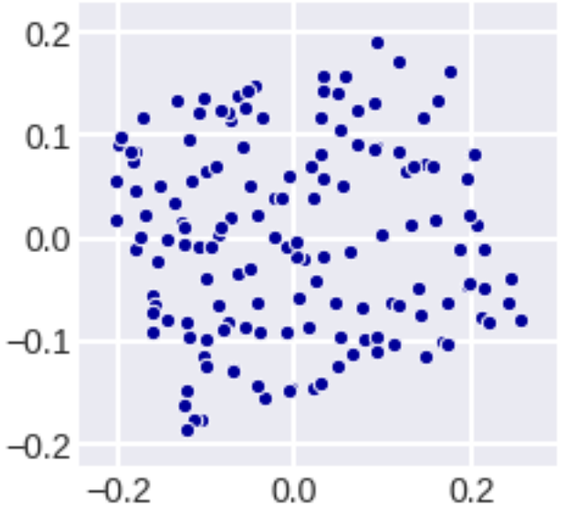
Independent component analysis (ICA)

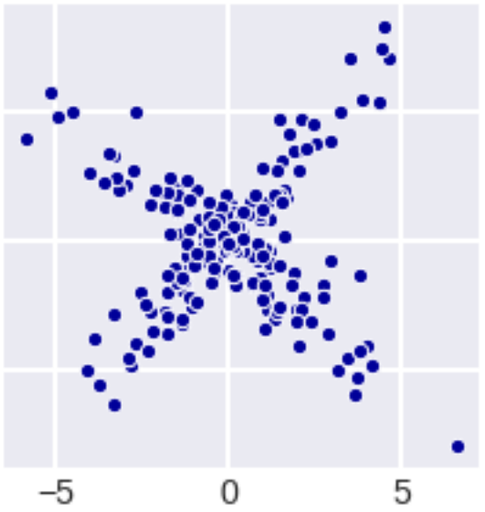
данные



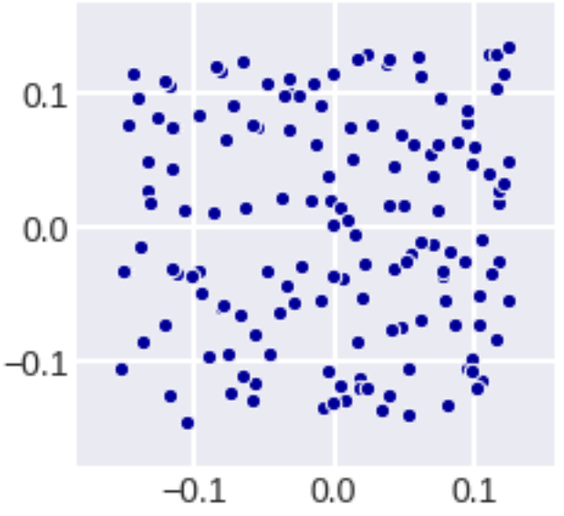


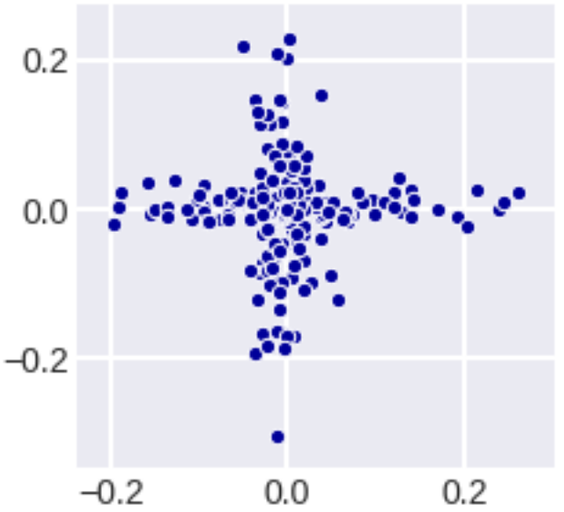
PCA





ICA





Independent component analysis (ICA)

ищем такие проекции, на которых данные максимально «негауссовские»
они не являются ортогональными в исходном пространстве
но они ортогональны в «whitened feature space»
(по всем направлениям одинаковая дисперсия)

```
from sklearn.decomposition import FastICA
ica = FastICA(random_state=1)
X_ica = ica.fit(X).transform(X)
```

ICA: метод FastICA

1. Центрируем признаки

$$\text{mean}(X_i) = 0$$

2. «Whitening» – делаем некоррелированные компоненты с дисперсией 1

если $X = ULV^{\text{T}}$ **то** $Z = XVL^{-1}$ **подходящее пребразование и** X **«превращается» в** U

3. Извлечение одной компоненты

ищем Xw

Вводится система функций (есть разные варианты):

$$f = -\exp(-u^2/2),\ f' = u\exp(-u^2/2),\ f'' = (1-u^2)\exp(-u^2/2)$$

Повторяем до сходимости

3.1. Случайная инициализация w

3.2. Пересчёт $w \leftarrow \mathbf{E}[X^{\text{T}} f'(Xw)] - \mathbf{E}[f''(Xw)]w$

матожидание – усреднение (на след слайде понятнее)

3.3. Нормировка $w \leftarrow w/\|w\|$

ICA: метод FastICA

Для извлечения нескольких компонент добавляем ортогонализацию:

1. Цикл по компонентам t=1:n

2.1 Инициализация w_t

2.2 Повторять до сходимости

$$w_t \leftarrow \frac{1}{m} X^{\mathrm{T}} f'(Xw_t) - \frac{1}{m}(\tilde{1}_m f''(Xw_t))w_t$$

$$w_t \leftarrow w_t - \sum_{j<t} (w_t^{\mathrm{T}} w_j) w_j$$

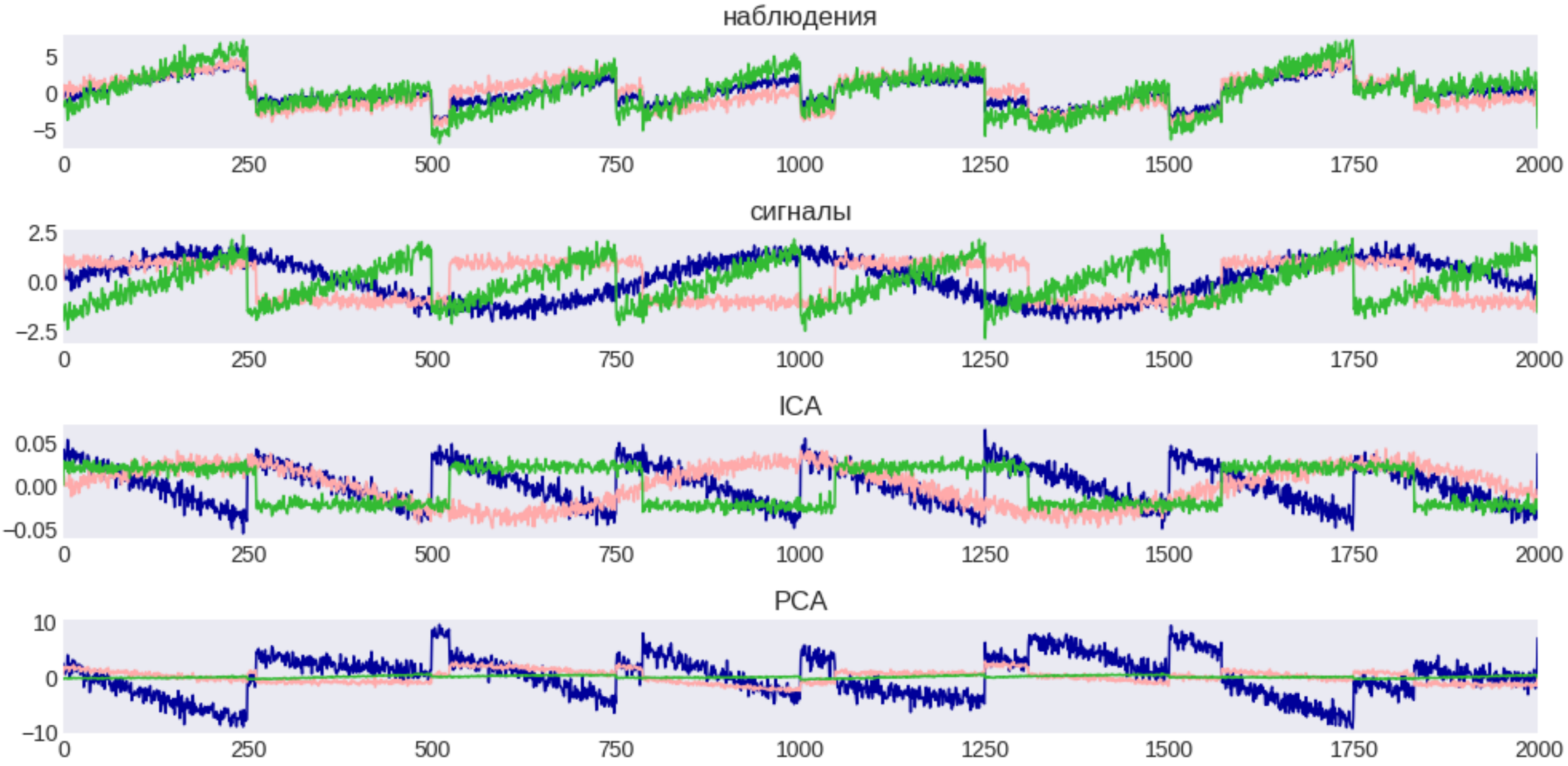
$$w_t \leftarrow w_t / \| w_t \|$$

Выход:

$$W = [w_1, \ldots, w_n]$$

$$Z = XW$$

ICA: прикладная задача «Blind Signal Separation»



Устранение шума (Noise Reduction)

[Glassner] !

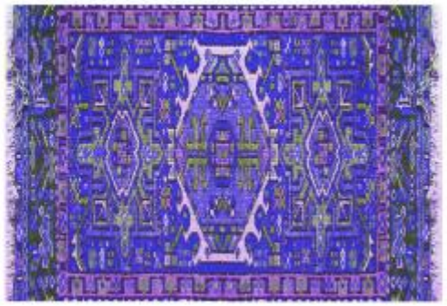
Генерация Данных (Data Generation)

Будет подробно в DL

Идея: есть данные – надо нагенерировать «таких же»

Дано:

Новые данные:



[Glassner]

Получение представлений (Representation Learning)

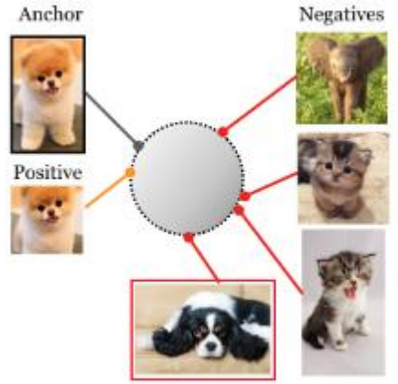
Будет подробно в DL

Сжатие и восстановление

автокодировщики

Сходства и представления

Consistency & Contrastive Learning



также полезно для генерации

часто связано с самообучением

Итоги

USL – определение «природы» (структуры) неразмеченных данных

Часто удаётся найти «хорошее» маломерное пространство

Также нет идеальных методов

не забывать про нормировку признаков

однородность пространства

интерактивная демка

http://colah.github.io/posts/2014-10-Visualizing-MNIST/

хорошая презентация по теме

https://sites.uclouvain.be/inma/reddot/slides/lee09.pdf