Обучение без учителя

Терминология всё та же…

Строки – экземпляры (наблюдения, объекты)

$$\begin{pmatrix} x_1^1 & x_2^1 & x_3^1 & x_4^1 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 \\ & \cdots & & \\ x_1^n & x_2^n & x_3^n & x_4^n \end{pmatrix}$$

Столбцы - признаки (характеристики, измерения)

Понижение размерности

i-я строка матрицы – описание i-го объекта (конкретного экземпляра)

$$x^i = (x_1^i, x_2^i, \ldots, x_n^i)$$

Задача: перейти к новым переменным, понизив их количество, при этом пытаясь сохранить как можно больше информации о первоначальном распределении

$$z^i = (z_1^i, z_2^i, \ldots, z_d^i)$$

d < n

Метод главных компонент (PCA – Principal component analysis)

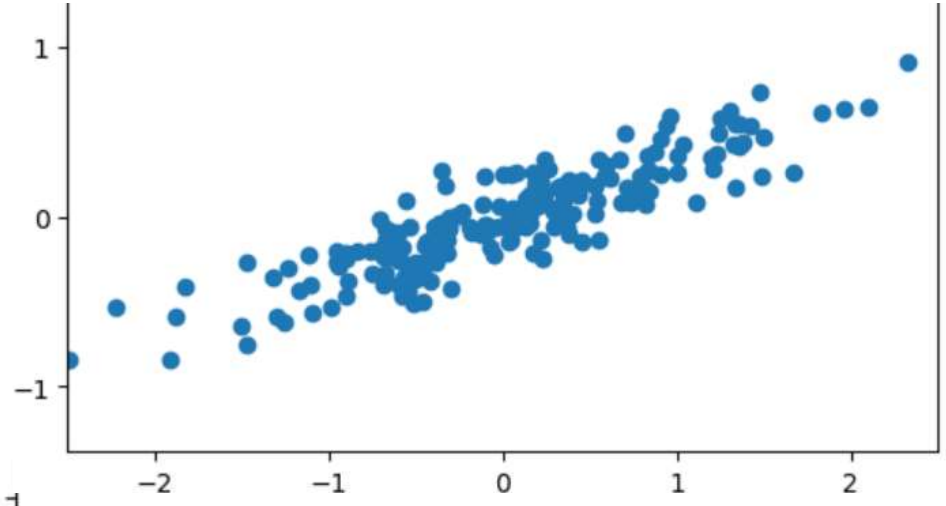
Отыскиваются главные оси координат (principal axes), которые используются для описания данных

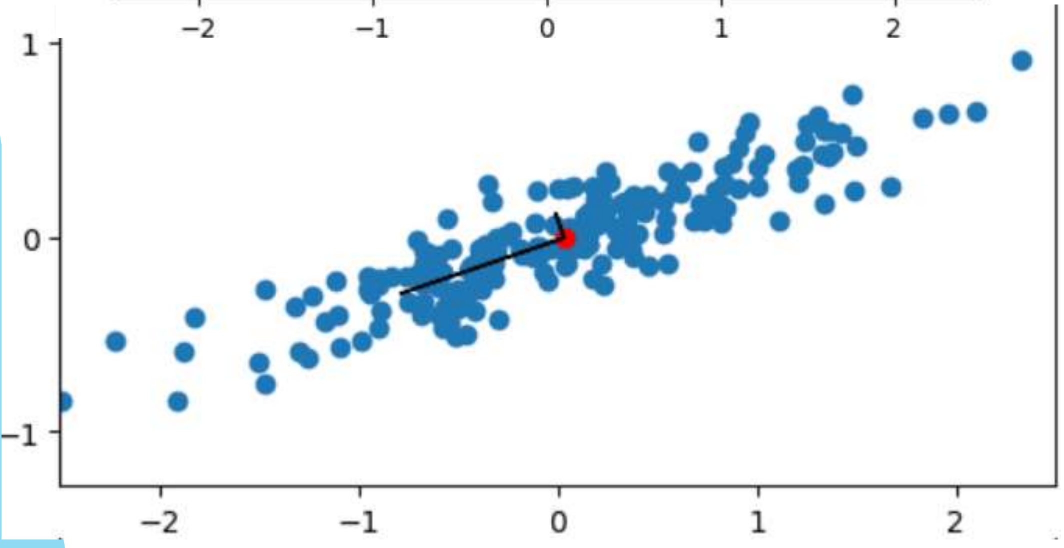
Дополнительно вычисляются относящиеся к данным величины так называемые объяснимые дисперсии – характеристика того, насколько тот или иной признак объясняет исходные данные

Метод состоит в том, что обнуляется одна или несколько главных компонент (остается базис для новой гиперплоскости меньшего размера, на которую проецируются исходные данные) с попыткой сохранить как можно больше дисперсии

Рассмотрим на примере (пока не понижая размерность)

Есть какой-то набор данных






```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

```
print(pca.components_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
```

```
print(pca.explained_variance_)
print(pca.explained_variance_ratio_)
```

```
[0.7625315 0.0184779]
[0.97634101 0.02365899]
```

Кол-во
%

```
pca.mean_
```

```
array([ 0.03351168, -0.00408072])
```

Визуализируем на данных: Компоненты - направления векторов, объяснимая дисперсия - квадраты длин

```
for exp_var, component in zip(pca.explained_variance_, pca.components_):
    dx, dy = math.sqrt(exp_var) * component
    plt.arrow(x0,y0,dx,dy,length_includes_head = True)
```

n_components = 1 и обратно

Преобразование с понижением размерности

```
pca1 = PCA(n_components=1)
pca1.fit(X)
```

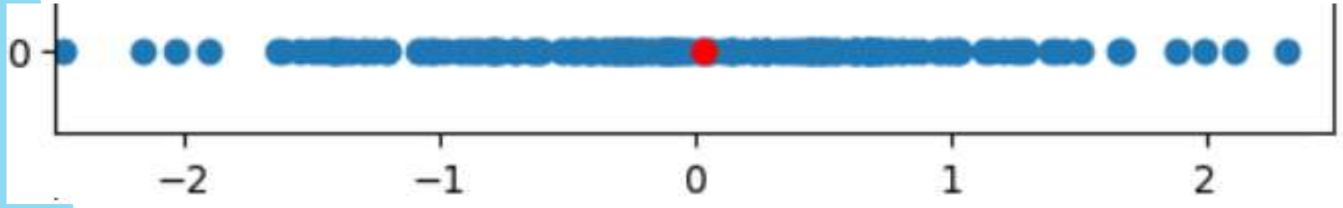
```
print(pca1.components_)
```

```
[[-0.94446029 -0.32862557]]
```

```
print(pca1.explained_variance_ratio_)
```

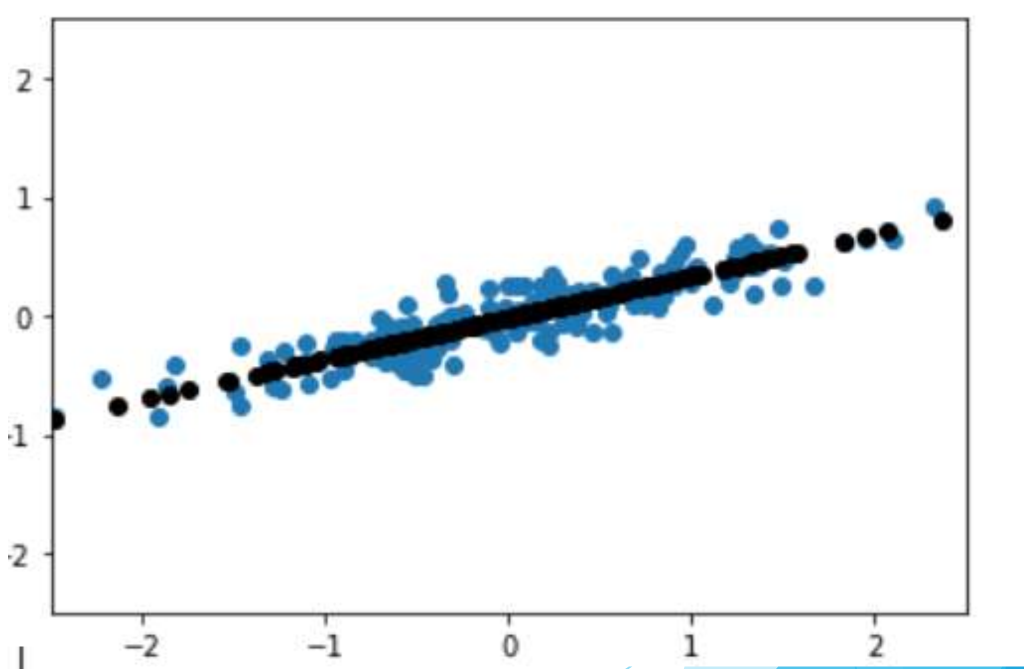
[0.97634101] [0.97634101 ~~0.02365899~~]

```
X_pca1 = pca1.transform(X)
```



Обратное преобразование

```
X_pca1inv = pca1.inverse_transform(X_pca1)
```



Сингулярное разложение (SVD – Singular Value Decomposition)

- Прямоугольную матрицу X (m*n) можно представить в виде произведения

$$X = U\Sigma V^*$$

- Σ – матрица m*n с неотрицательными элементами, лежащими на главной диагонали – т.н. сингулярные числа (расположены по убыванию)
- U (m*m),V(n*n) – две унитарные матрицы, состоящие из левых и правых сингулярных векторов

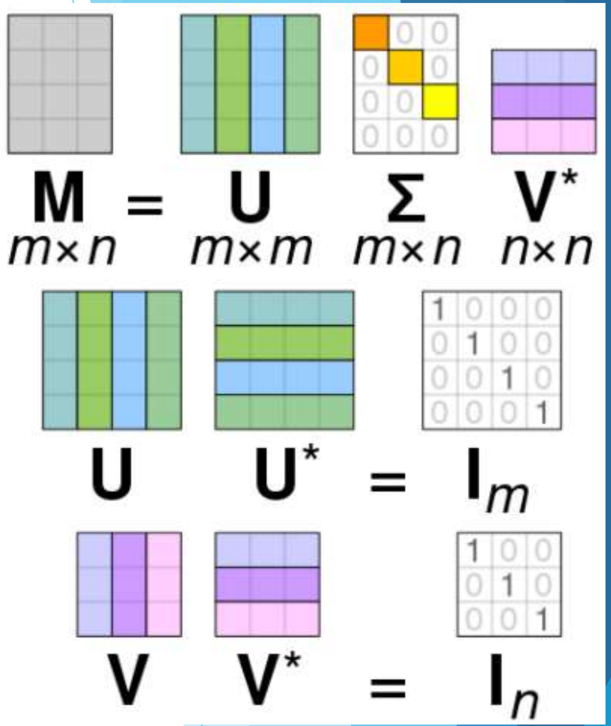
Матрица * - сопряженно-транспонированная (эрмитово-сопряженная) к исходной

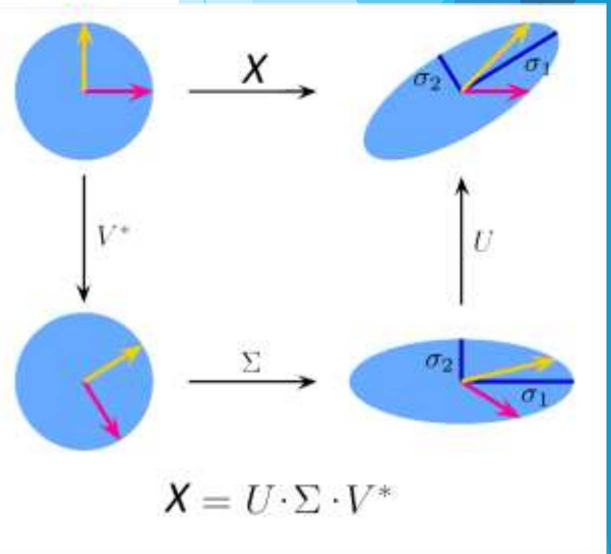
Унитарная матрица – квадратная матрица, результат умножения которой на сопряженно-транспонированную равен единичной матрице

Неотрицательное вещественное число σ называется **сингулярным числом** матрицы M тогда и только тогда, когда существуют два вектора единичной длины

$$Mv = \sigma u, M^*u = \sigma v$$

Векторы u и v называются, соответственно, **левым сингулярным вектором** и **правым сингулярным вектором**, соответствующим сингулярному числу σ.





Сингулярное разложение из примера n_components=2

```
X_m = X-X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_m)
```

```
Vt
```

```
array([[ 0.94465994,  0.3280512 ],
       [ 0.3280512 , -0.94465994]])
```

```
np.matrix(Vt).H * np.matrix(Vt)
```

```
matrix([[1., 0.],
        [0., 1.]])
```

```
sig = np.zeros([200,2])
sig[0,0]=s[0]
sig[1,1]=s[1]
```

```
s
```

```
array([12.33,  1.93])
```

```
mult = np.dot(U,sig)
mult = np.dot(mult,Vt)
```

```
mult-X+pca.mean_
```

```
array([[-0., -0.],
       [-0., -0.],
       [ 0.,  0.],
       ...,
       [-0.,  0.],
       [ 0., -0.],
       [-0.,  0.]])
```

property **matrix.H**

Returns the (complex) conjugate transpose of *self*.

Equivalent to `np.transpose(self)` if *self* is real-valued.

```
# Center data
self.mean_ = np.mean(X, axis=0)
X -= self.mean_

U, S, Vt = linalg.svd(X, full_matrices=False)
# flip eigenvectors' sign to enforce deterministic output
U, Vt = svd_flip(U, Vt)

components_ = Vt
```

https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/decomposition/_pca.py

Сингулярное разложение из примера n_components=1

```
X_m = X-X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_m)
```

```
Vt
```

```
array([[ 0.94465994,  0.3280512 ],
       [ 0.3280512 , -0.94465994]])
```

(V*)*=V

Главные компоненты 1 и 2

```
s
```

```
array([12.33,  1.93])
```

```
Vd = V[:,0]
Vd
```

```
matrix([[0.9445],
        [0.3286]])
```

```
X_new = np.dot(X_m, Vd)
```

```
pca1 = PCA(n_components=1)
```

```
X_pca1 = pca1.transform(X)
```

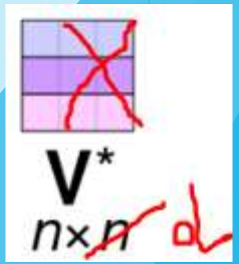
```
X_new-X_pca1
```

```
matrix([[0.],
        [0.],
        [0.],
        ...,
        [0.],
        [0.],
        [0.]])
```

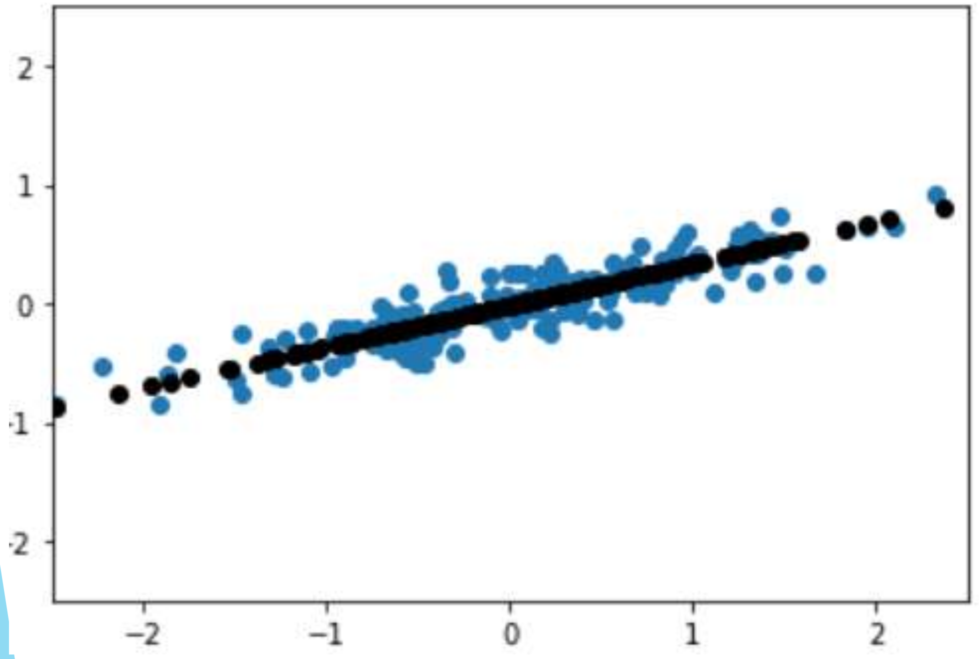
d < n

$$X_d = XV_d$$

mxd mxn nxd



Обратное преобразование




```
X_pca1inv = pca1.inverse_transform(X_pca1)
```

Прямое преобразование

$$X_d = XV_d$$

$$X_d V_d^T = XV_d V_d^T$$

т.к. V – унитарная матрица

$$V_d V_d^T = E$$

$$X = X_d V_d^T$$

Помним ещё про цветы? Классика…

Щетинистый

Iris setosa

Виргинский

Iris virginica

Разноцветный

Iris versicolor

4 характеристики:
Длина чашелистика
Ширина чашелистика
Длина лепестка
Ширина лепестка

Хотим отобразить на плоскости

Отображение цветов на плоскости

```
pca = PCA(n_components=2)
pca.fit(datasets.load_iris().data)
```

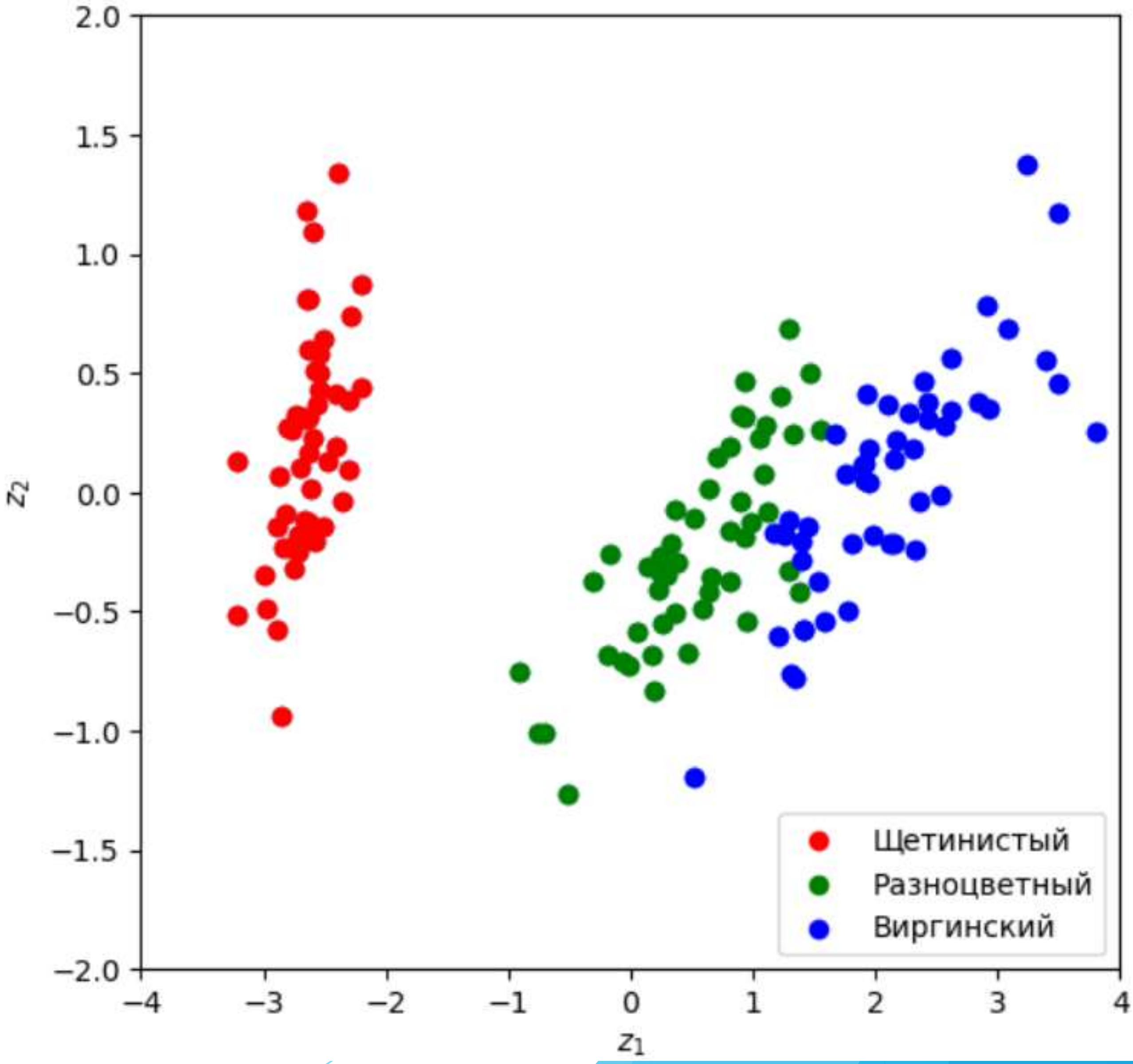
```
irisFlat = pca.transform(X)
```

```
pca.explained_variance_ratio_
```

```
array([0.9246, 0.0531])
```

```
sum(pca.explained_variance_ratio_)
```

```
0.977685206318795
```



Отображение цветов на прямую

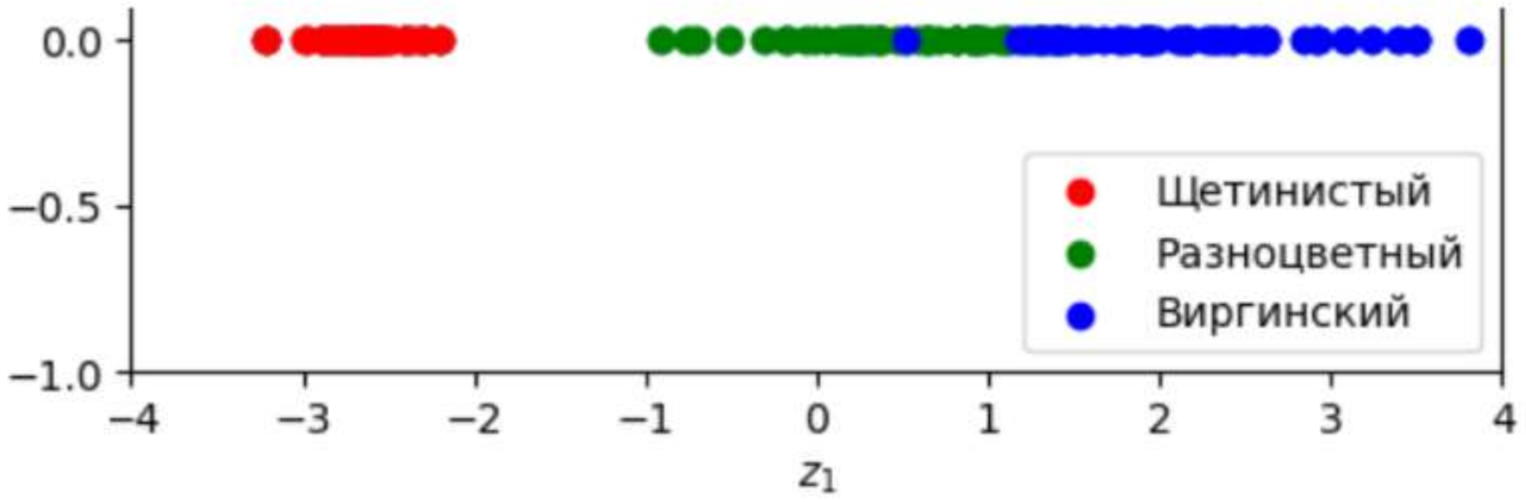
```
pca = PCA(n_components=1)
pca.fit(datasets.load_iris().data)
```

```
pca.explained_variance_ratio_
```

```
array([0.9246])
```

```
sum(pca.explained_variance_ratio_)
```

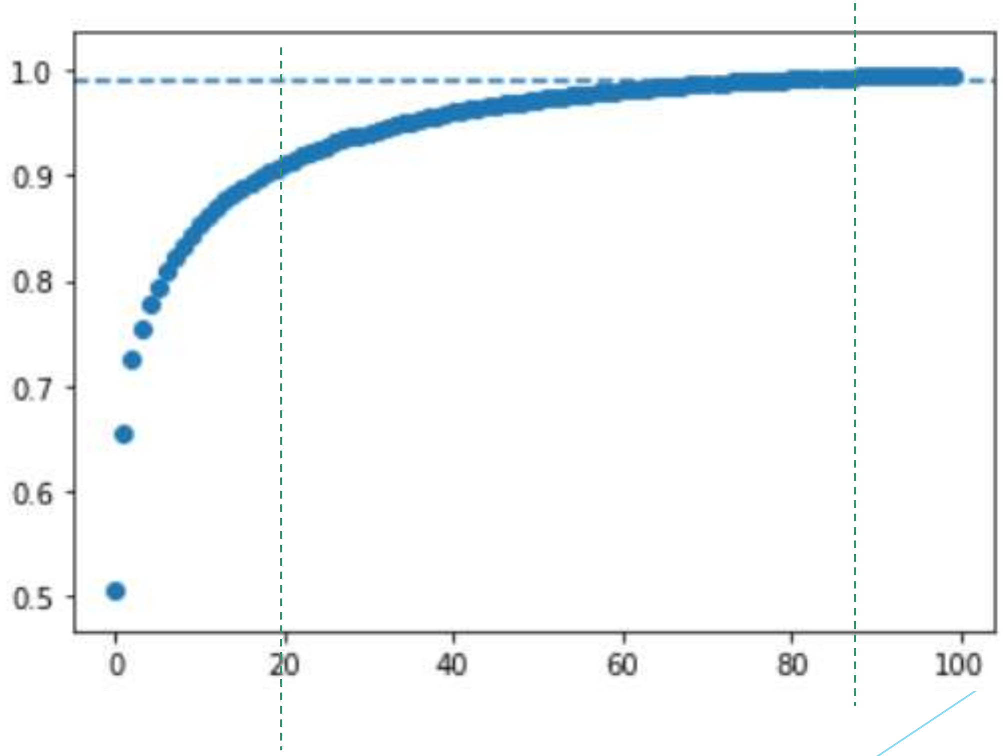
```
0.9246187232017271
```



Сколько компонент оставить?

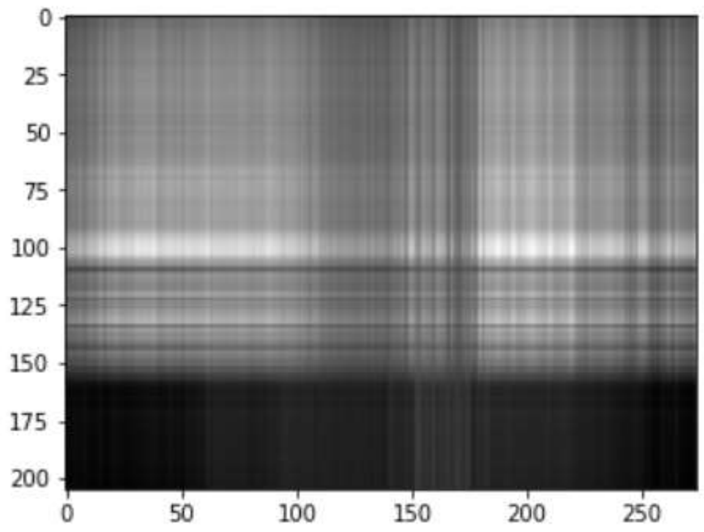
- Сделать вывод по графику. Отобразить визуально количество компонент и объясняемую ими дисперсию
- Сделаем PCA(100) и посмотрим объясняемую дисперсию накопительным итогом

Размер 205*274

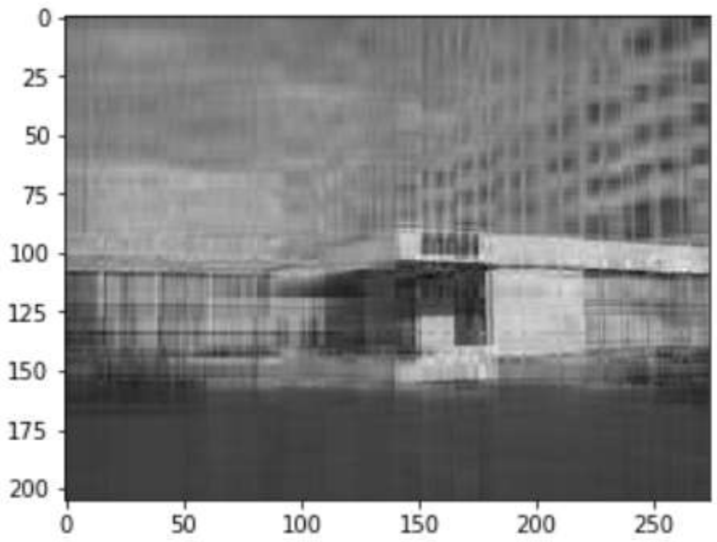


- Можно указать в конструкторе PCA(n_components=0.9)

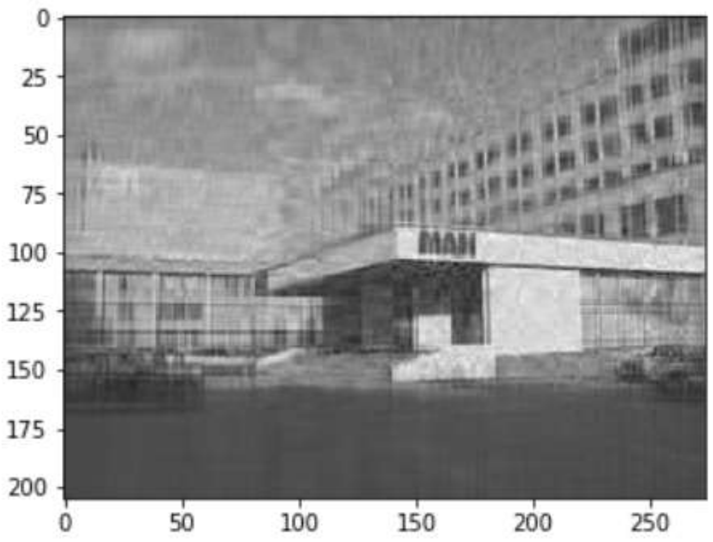
Разные n_components



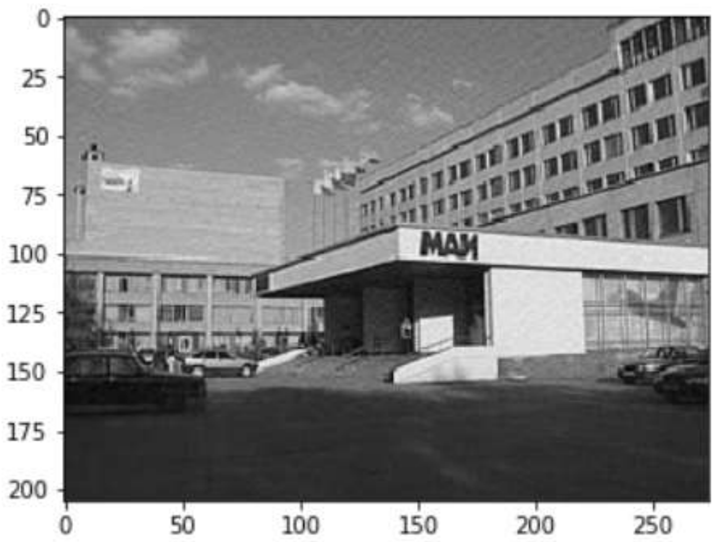
N=1 (50%)



N=10 (84.4%)



N=20 (90.7%)



N=79 (99%)

2/5 от начального количества

Добавление в конвейер для задач обучения с учителем

```
pca = PCA()
clfr = LogisticRegression(max_iter=10000, tol=0.1)
pipeline = Pipeline(steps=[('pca', pca), ('logistic', logistic)])
```

```
param_grid = {
    'pca__n_components': [1,2,3,4],
    'logistic__C': np.logspace(-4, 4, 9),
}
```

```
search = GridSearchCV(pipe, param_grid, n_jobs=-1)
search.fit(X_train, y_train)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

```
Best parameter (CV score=0.970):
{'logistic__C': 10.0, 'pca__n_components': 3}
```

Большие объемы данных

Если очень много признаков – svd_solver == ‘randomized’

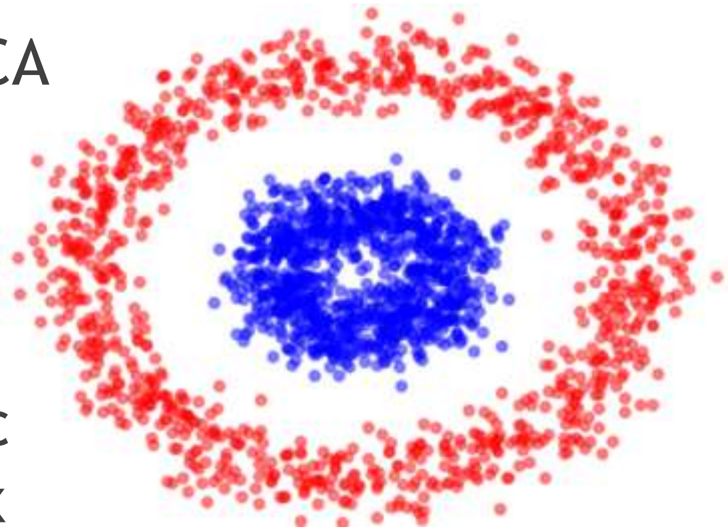
- Быстрая аппроксимация d главных компонент
- Вычислительная сложность $O(m \times d^2) + O(d^3)$ вместо $O(m \times n^2) + O(n^3)$

Если объем данных очень большой и не помещается в память - IncrementalPCA

- В алгоритм передаются данные минипакетами (параметр batch_size)
- Можно передавать новые данные по мере поступления
- Фиксированное ограничение по памяти batch_size * n_features

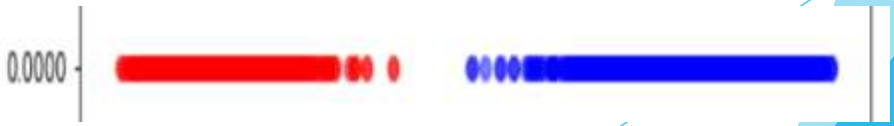
KernelPCA

- Применение ядерного трюка к PCA
- Пример: ядро RBF
- По умолчанию обратное преобразование недоступно (можно задать настройки чтобы с использованием дополнительных техник попытаться восстановить)

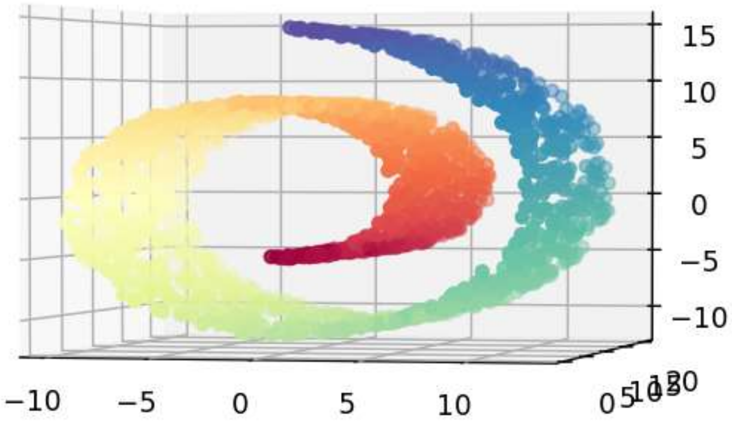


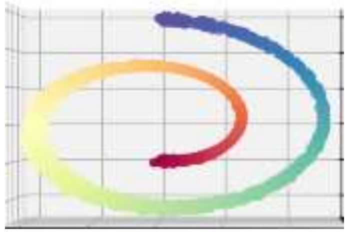
PCA(n_components=1)

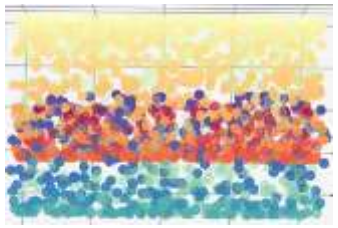
KernelPCA(n_components=1, kernel='rbf', gamma=1.7)

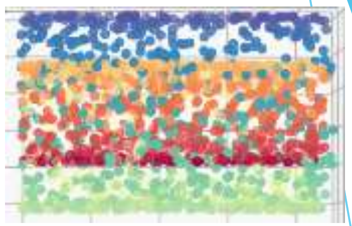


KernelPCA для швейцарского рулета

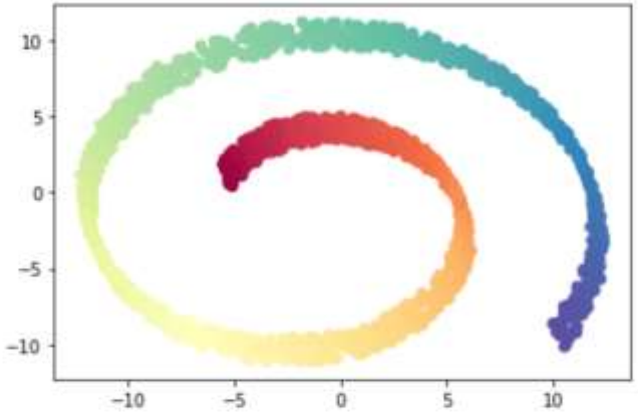




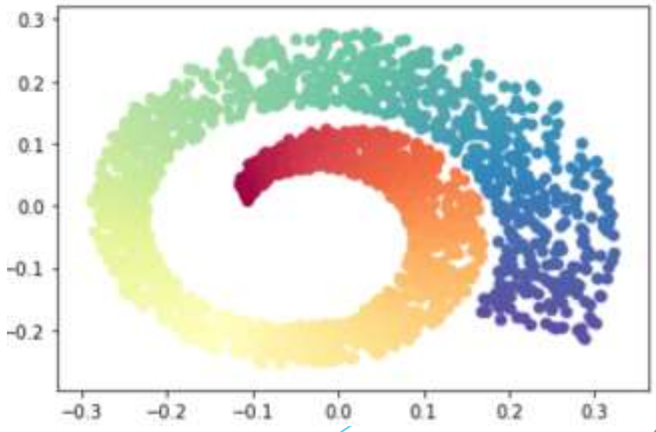




PCA(n_components=2)



KernelPCA(n_components=2, kernel='sigmoid', gamma=0.0013)



Как подобрать гиперпараметры если это не задача обучения с учителем?

- Гиперпараметры у KernelPCA degree, gamma, kernel

```
param_grid = [{ "degree": [1,2,3,4],
        "gamma": [0.1, 0.5, 1, 2, 3],
        "kernel": ["poly","rbf","sigmoid"]  }]
```

- GridSearchCV. Сигнатура метода fit

fit(*X*, *y=None*, *, *groups=None*, ***fit_params*)

y - Target relative to X for classification or regression; **None for unsupervised learning**.

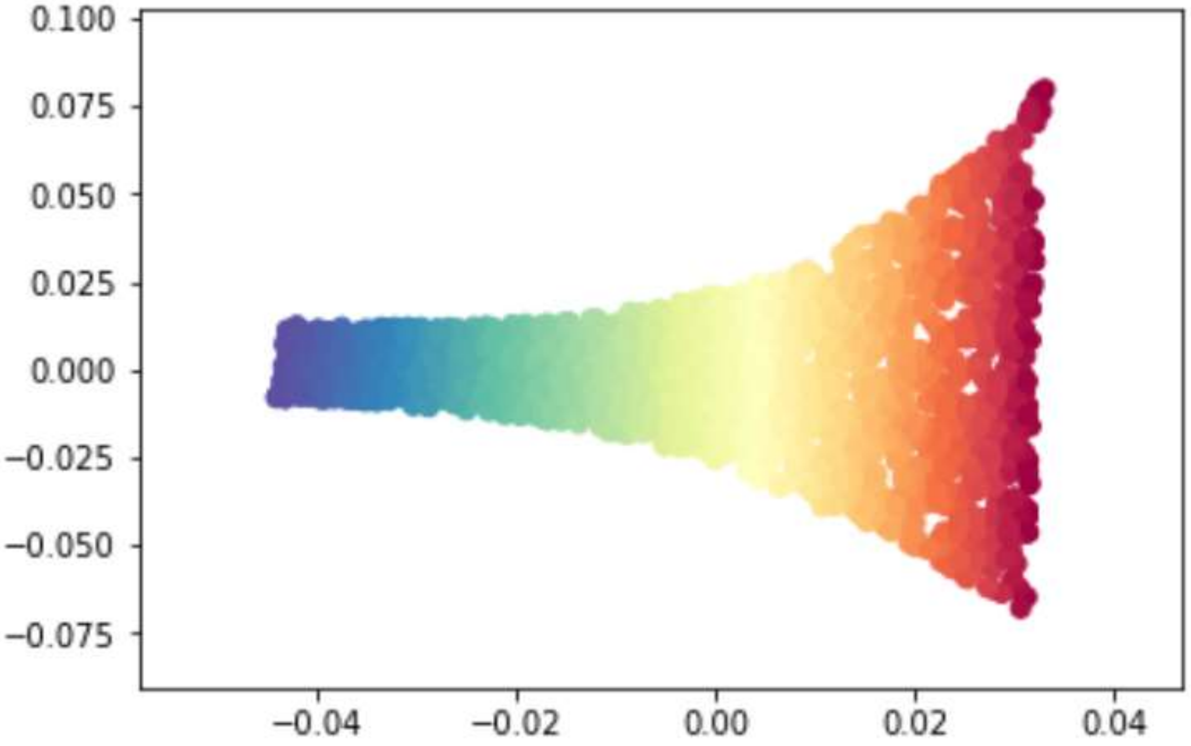
- GridSearchCV(kernelPca, param_grid, cv=4, scoring=??)

```
from sklearn.metrics import mean_squared_error
def my_scorer(estimator, X, y=None):
    X_reduced = estimator.transform(X)
    X_preimage = estimator.inverse_transform(X_reduced)
    return -1 * mean_squared_error(X, X_preimage)
```

Проверить, как хорошо смогут восстановиться признаки после обратного преобразования

Методики на основе многообразий

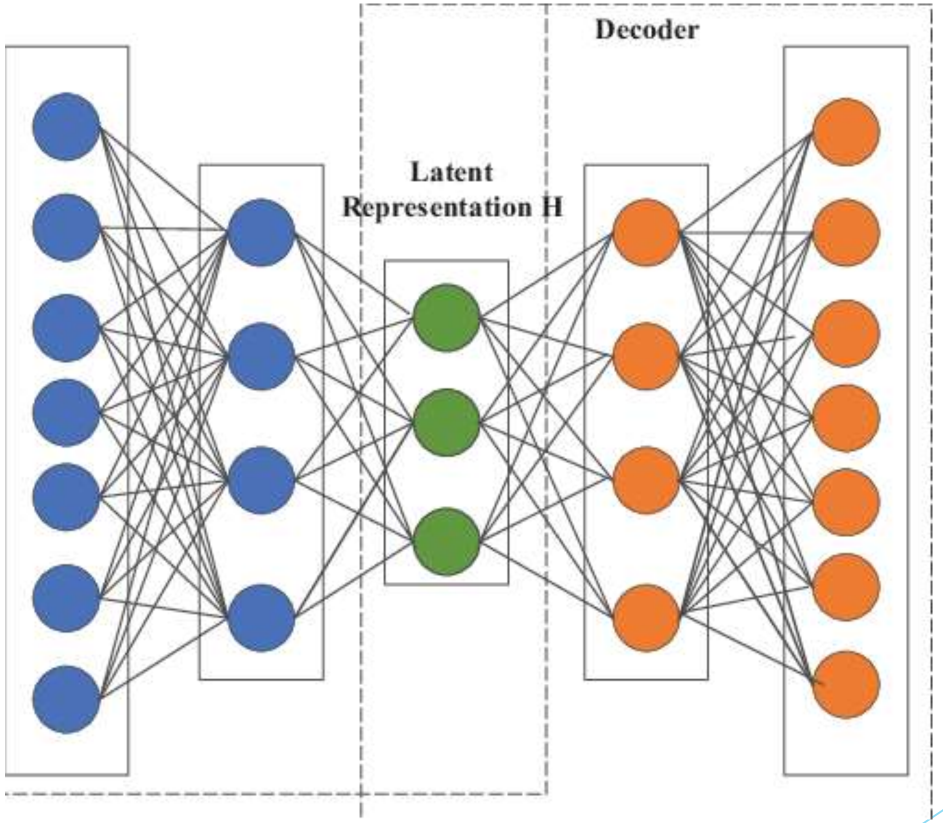
- Измеряет как каждый образец связан со своими соседями
- Ищет представление с меньшим набором измерений, где связи с соседями лучше сохраняются



Автокодировщики

- Искусственные нейронные сети, используемые для понижения размерности

- Если используется линейная функция активации и функция издержек MSE, можно показать, что выполняется анализ главных компонент (PCA)



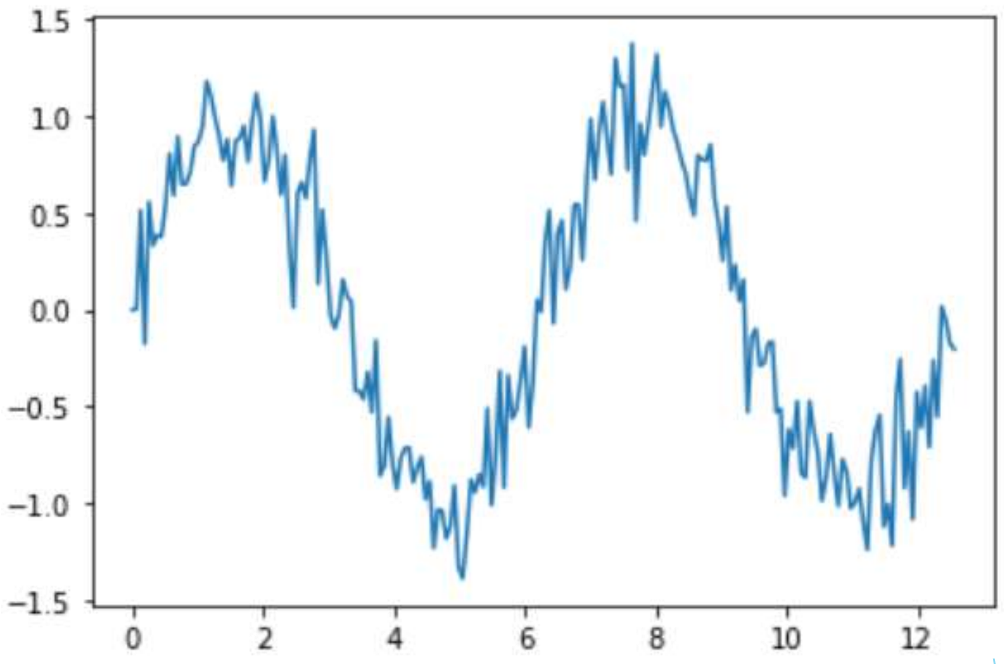
Фильтр шума

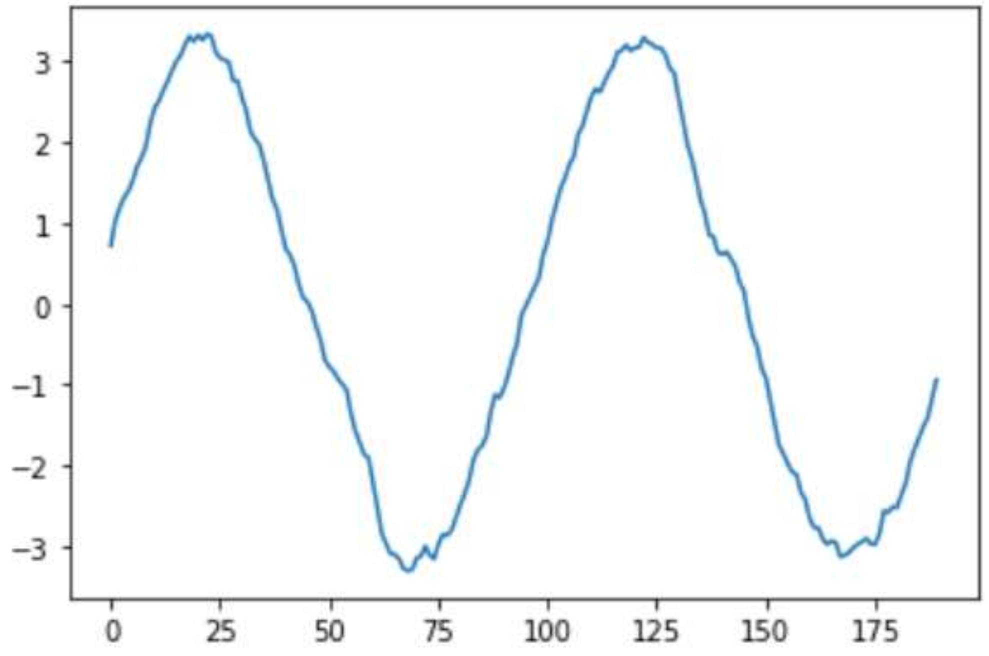
```
n = 200
points = np.linspace(0,4*3.14,n)
noise = np.random.normal(0,0.2,n)
sin = np.sin(points) + noise
```

Создадим 2D массив, где построчно будет элемент исходного массива и следующие 9 элементов

```
res = []
period = 10
for i in range(len(sin)-period):
    res.append(sin[i:i+period])
```

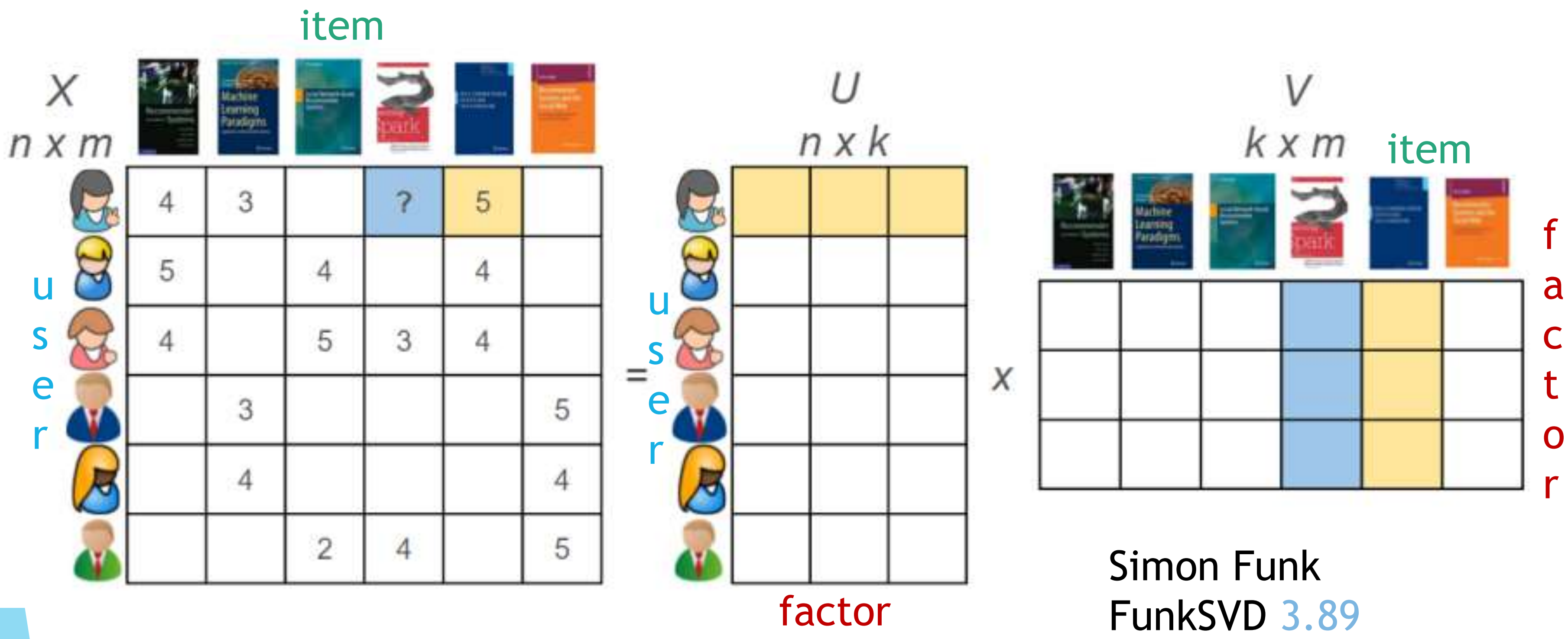
Выполним преобразование PCA с количеством компонент 1, отобразим полученный одномерный массив на графике



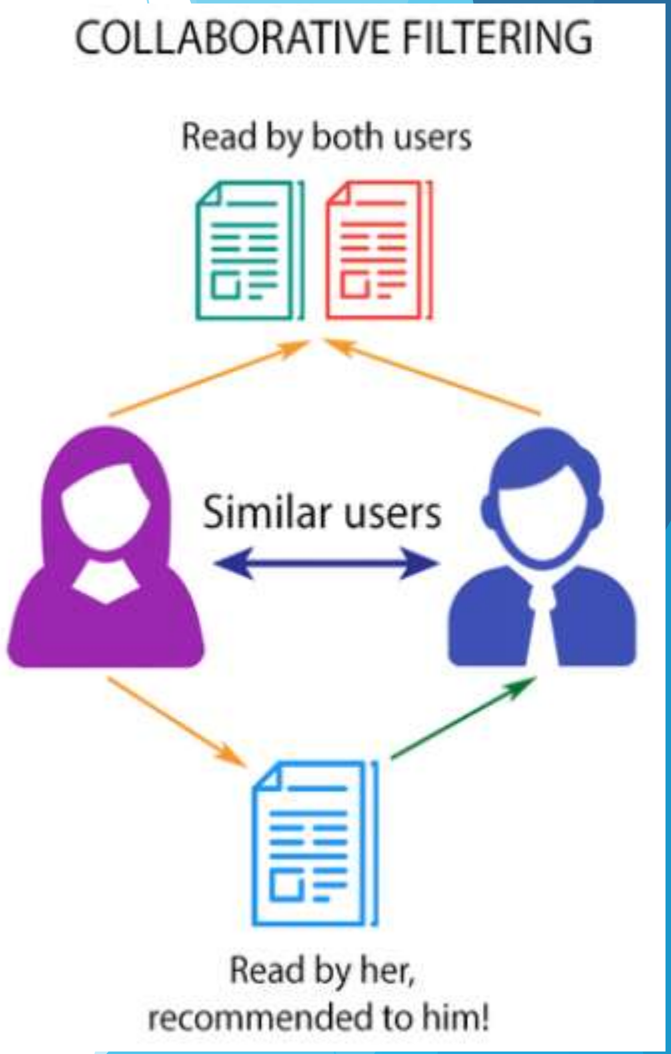


Рекомендательные системы

- Матрица: строки – пользователи, столбцы - продукты



Simon Funk
FunkSVD 3.89



Вопросы

- Какие главные мотивы для понижения размерности?
- В чем основной недостаток понижения размерности?