Unsupervised

Кластеризация
Понижение размерности

Кластеризация

Кла́стер (c*luster* — скопление, кисть, рой) — объединение нескольких однородных элементов, которое может рассматриваться как самостоятельная единица, обладающая определёнными свойствами.

Кластеризация - задача группировки элементов на подмножества, так чтобы они сформировали кластеры.

Более формально

Пусть X — множество объектов, Y— множество идентификаторов (меток) кластеров. На множестве X задана функция расстояния между объектами $\rho(x, x')$. Дана конечная обучающая выборка объектов $X_m = \{x_1, \ldots, x_m\} \subset X$. Необходимо разбить выборку на подмножества (кластеры), то есть каждому объекту $x_i \in X_m$ сопоставить метку $y_i \in Y$, таким образом чтобы объекты внутри каждого кластера были близки относительно метрики ρ, а объекты из разных кластеров значительно различались.

Алгоритм кластеризации — функция $a: X \to Y$, которая любому объекту $x \in X$ ставит в соответствие идентификатор кластера $y \in Y$

Зачем нужна Кластеризация

1. **Кластеризация клиентов** (товаров) компании
   для настройки рекламы, понимания платежеспособности клиентов

2. **Сжатие данных**
   устранение похожестей в данных, сжатие информации

3. **Тематическое моделирование**
   автоматическое создание групп новостей - тем

4. **Поиск выбросов в данных**
   непохожие картинки, записи, нестандартное поведение клиента и тд.

5. Поиск какой-либо **структуры в данных**
   в зависимости от применяемого алгоритма

6. Многие другие применения в составе более сложных алгоритмов

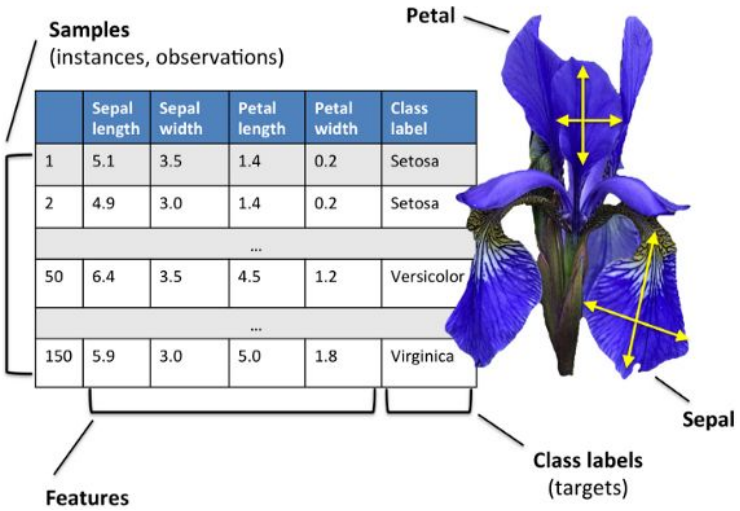
Как она выглядит

На входе имеем:

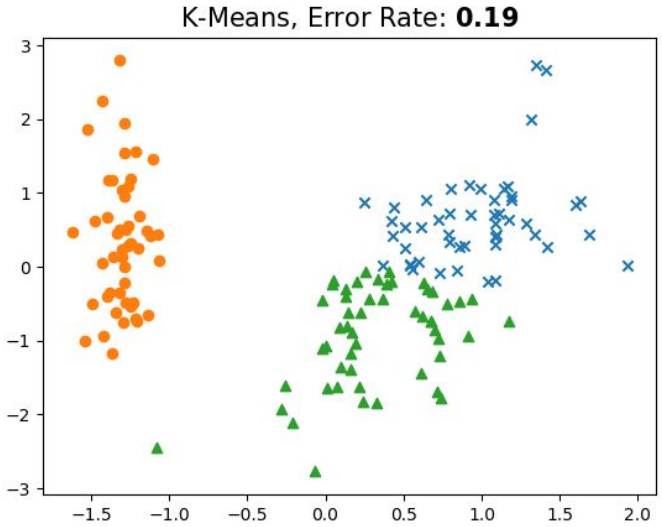
1. Признаковые описания объектов
2. Некоторую априорную инф. о кластерах

На выходе:

Сопоставление каждой строчки некоторому лейблу

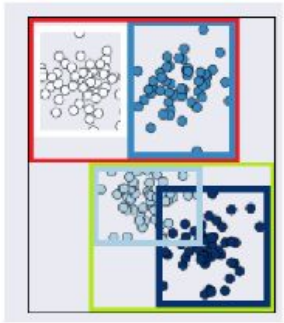


	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



Типы алгоритмов кластеризации

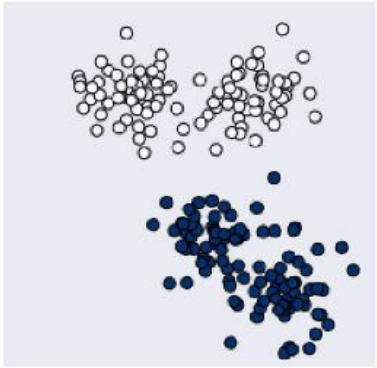
Плоские / разделяющие (Flat / Partitional) – кластеризация на k непересекающихся кластеров



Иерархические (Hierarchical) – данные один большой кластер, далее рекурсивно «кластер = объединение подкластеров»

~ система каталогов

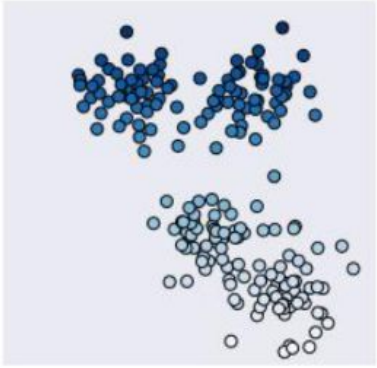
Типы алгоритмов кластеризации



чёткая (hard)

разбиение на непересекающиеся кластеры

$$\aleph = C_1 \cup \ldots \cup C_k$$
$$i \neq j \Rightarrow C_i \cap C_j = \varnothing$$



нечёткая (мягкая, fuzzy)

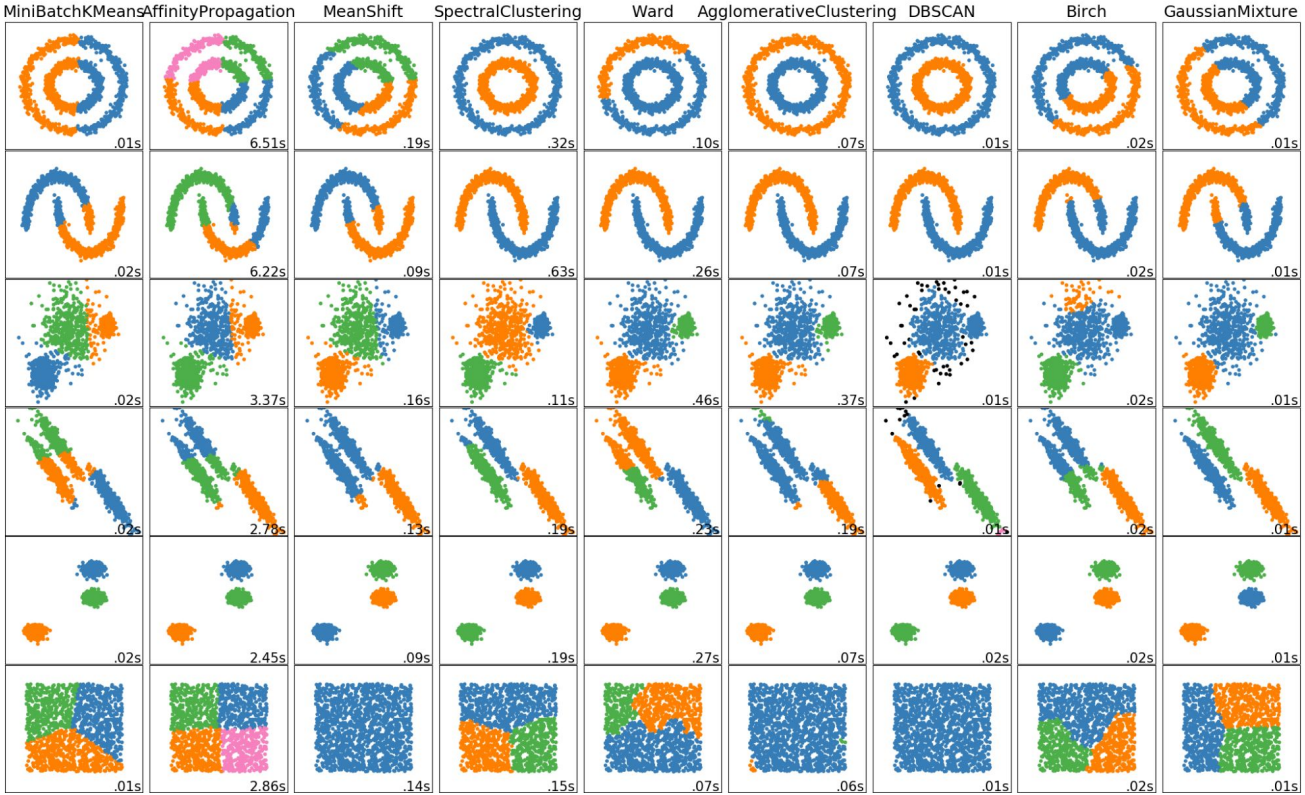
определение степени принадлежности кластерам

$$x_i \to r_{it} \in [0,1]$$
$$\forall i \in \{1,2,\ldots,m\} \sum_{t=1}^{k} r_{it} = 1$$

Кластеризация. Реализация в sklearn

`cluster.AffinityPropagation`(*[, damping, ...])	Perform Affinity Propagation Clustering of data.
`cluster.AgglomerativeClustering`([...])	Agglomerative Clustering.
`cluster.Birch`(*[, threshold, ...])	Implements the BIRCH clustering algorithm.
`cluster.DBSCAN`([eps, min_samples, metric, ...])	Perform DBSCAN clustering from vector array or distance matrix.
`cluster.FeatureAgglomeration`([n_clusters, ...])	Agglomerate features.
`cluster.KMeans`([n_clusters, init, n_init, ...])	K-Means clustering.
`cluster.BisectingKMeans`([n_clusters, init, ...])	Bisecting K-Means clustering.
`cluster.MiniBatchKMeans`([n_clusters, init, ...])	Mini-Batch K-Means clustering.
`cluster.MeanShift`(*[, bandwidth, seeds, ...])	Mean shift clustering using a flat kernel.
`cluster.OPTICS`(*[, min_samples, max_eps, ...])	Estimate clustering structure from vector array.
`cluster.SpectralClustering`([n_clusters, ...])	Apply clustering to a projection of the normalized Laplacian.
`cluster.SpectralBiclustering`([n_clusters, ...])	Spectral biclustering (Kluger, 2003).
`cluster.SpectralCoclustering`([n_clusters, ...])	Spectral Co-Clustering algorithm (Dhillon, 2001).

Сравнение алгоритмов



Все реализации доступны в пакете sklearn.cluster

K-Means (кластеризация k-средних, алг. Лойда)

Основная идея заключается в том, что на каждой итерации перевычисляется **центр масс** для каждого кластера, полученного на предыдущем шаге, затем объекты снова разбиваются на кластеры в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике. Алгоритм завершается, когда на какой-то итерации не происходит изменения внутрикластерного расстояния.

На вход подается выборка и количество кластеров K.

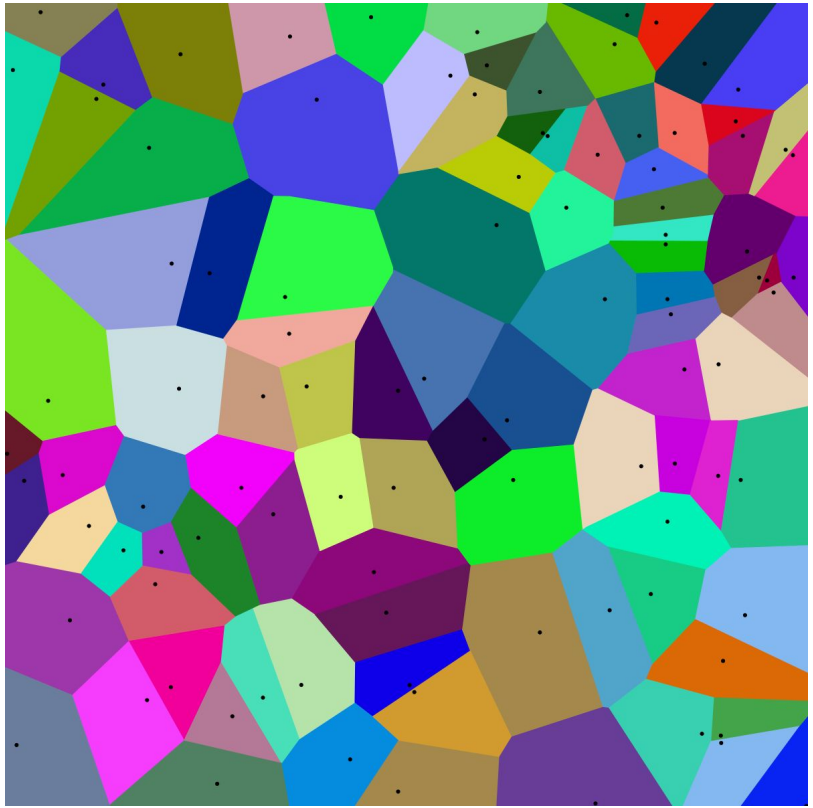
Алгоритм минимизирует сумму квадратов внутрикластерных расстояний:

$$\sum_{i=1}^{m} ||x_i - \mu_{a_i}||^2 \rightarrow \min_{\{a_i\},\{\mu_a\}}, \quad ||x_i - \mu_a||^2 = \sum_{j=1}^{n} (f_j(x_i) - \mu_{a_j})^2$$

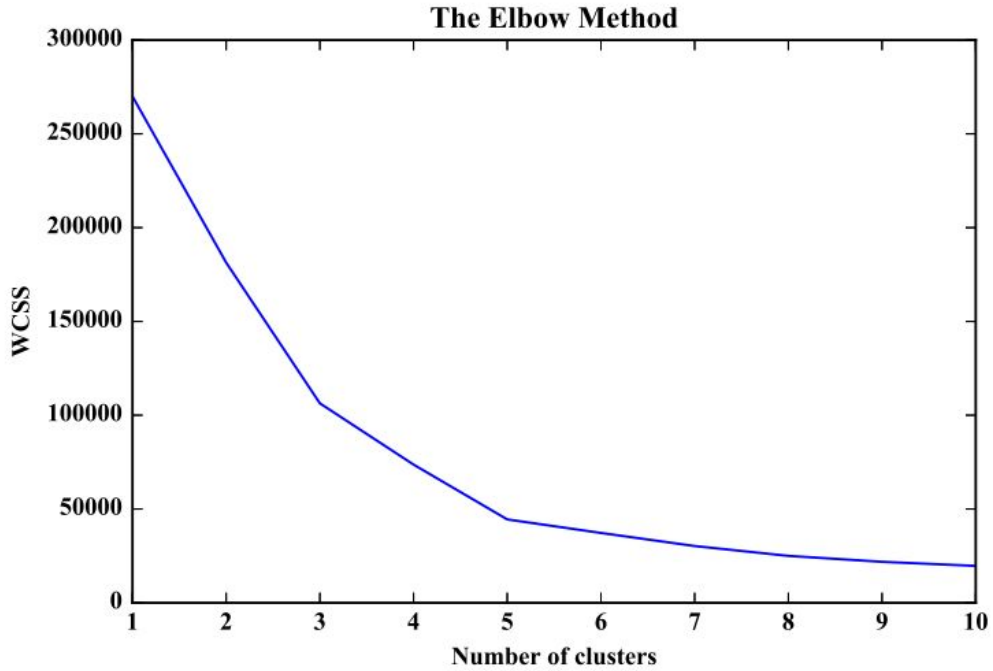
Недостатки:
- требуется заранее знать количество кластеров
- требуется конкретный вид кластера (есть центройд)
- не может работать с “вложенными” кластерами

K-Means - это диаграммы Вороного

Диаграмма Вороного конечного множества точек S на плоскости представляет такое разбиение плоскости, при котором каждая область этого разбиения образует множество точек, более близких к одному из элементов множества S, чем к любому другому элементу множества.



Метод плеча - поиск параметра K



Есть python библиотека которая делает поиск автоматически:

from kneed import KneeLocator

```
from kneed import KneeLocator

i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex',
direction='increasing', interp_method='polynomial')

fig = plt.figure(figsize=(5, 5))

knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")

print(distances[knee.knee])
```

KMeans. Пример использования

```
def simplify_image(image, k=10):
    image2 = image.copy()
    image2.resize([image.shape[0]*image.shape[1], 3])
    model = KMeans(n_clusters=k, random_state=11)
    model.fit(image2)
    result = model.cluster_centers_.round().astype(int)[model.labels_,:]
    result.resize([image.shape[0], image.shape[1], 3])
    mask = model.labels_
    mask.resize([image.shape[0], image.shape[1]])
    return result, mask
```

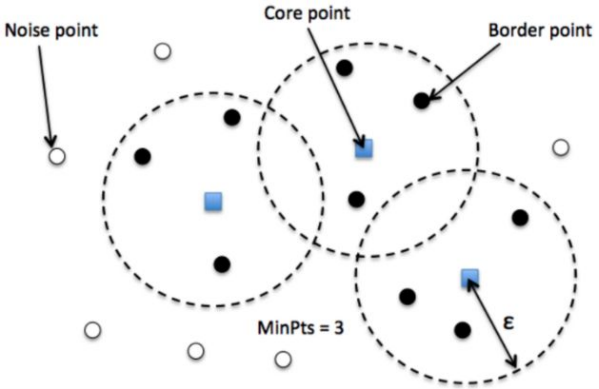
DBSCAN. Другой взгляд на данные

Основная идея **Density-based spatial clustering of applications with noise** (**DBSCAN**) - неконтролируемый (почти) алгоритм кластеризации, который используется для поиска базовых выборок с высокой плотностью для расширения кластеров.

На вход требует:

1. **minPts:** Минимальное количество точек (порог), сгруппированных вместе, чтобы область считалась плотной.

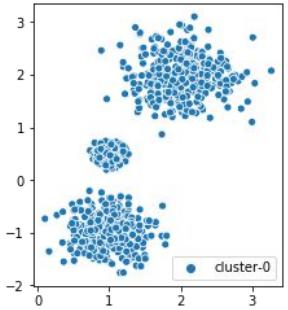
2. **eps (ε):** Мера расстояния, которая будет использоваться для определения местоположения точек в окрестностях любой точки.



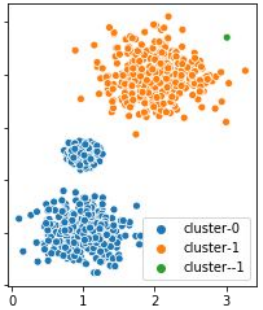
Недостатки:
- неоднозначные параметры (сложно угадывать)
- плотности могут быть сильно неоднородны (разрывы)
- тяжело работает с высокоразмерными данными (требуется понижение)

DBSCAN. Пример.

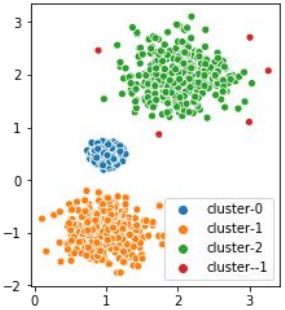
eps = 1.0



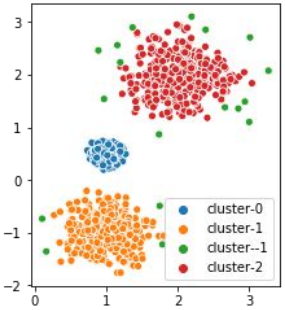
eps = 0.5



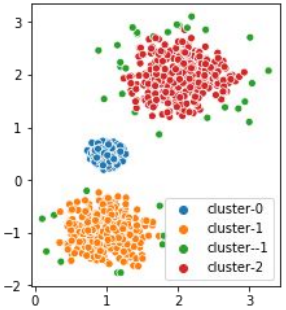
eps = 0.33



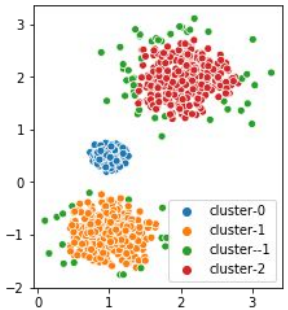
eps = 0.25



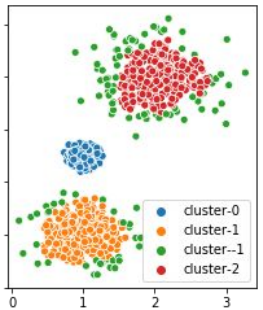
eps = 0.2



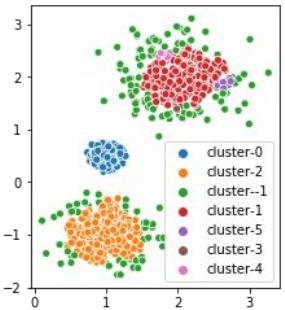
eps = 0.17



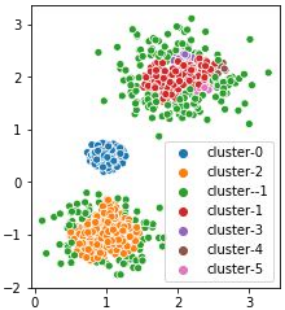
eps = 0.14



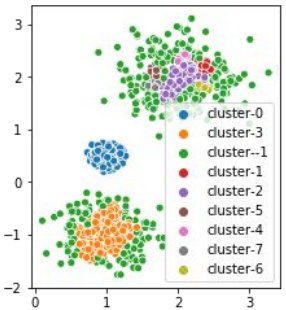
eps = 0.12



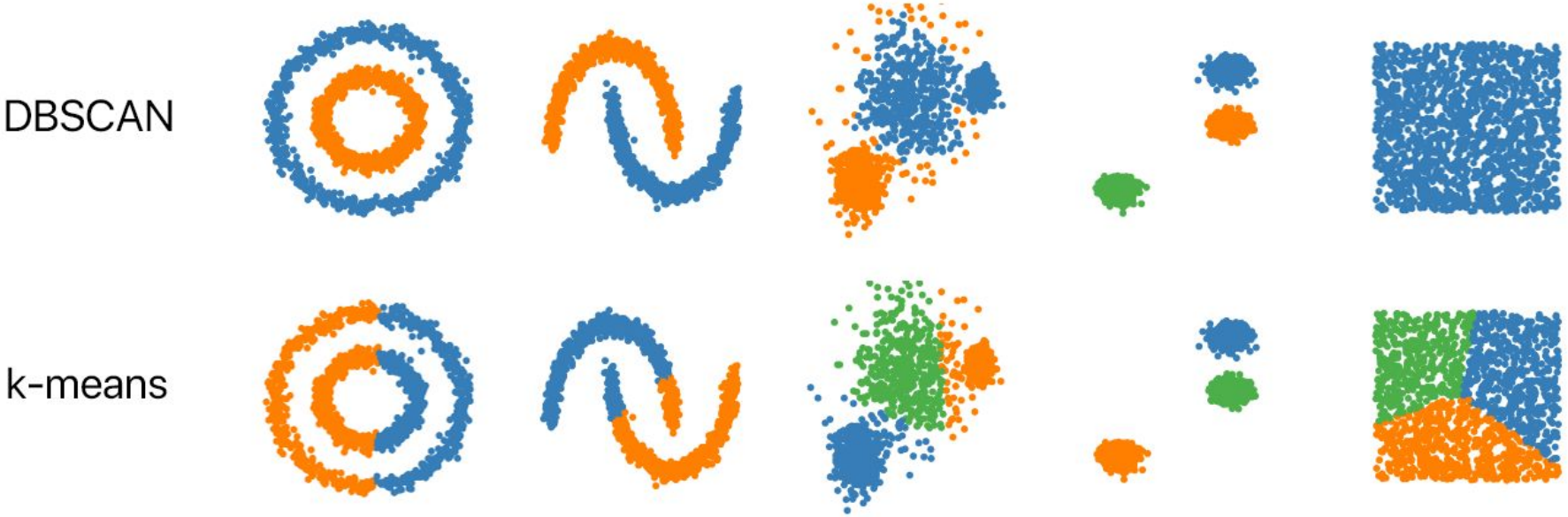
eps = 0.11



eps = 0.1



KMeans vs DBSCAN



Понижение размерности

В статистике, машинном обучении и теории информации **снижение размерности** — это преобразование данных, состоящее в уменьшении числа переменных путём **получения главных переменных**.

Такие методы можно разделить на **линейные** и **нелинейные**. Оба хороши, но первые требуют лучшего понимания данных, а вторые обычно более медленные.

В основе первых лежат линейные преобразование - разложения матриц признаков (SVD, PCA и прочие).

В основе нелинейных - модели нейронных сетей (автокодировщики), методы на основе многообразий. К ним относятся UMAP, tSNE и др.

Самое частое **применение** таких алгоритмов - для визуализации (2-d или 3-d) или как промежуточный шаг (например понижение размерности, а после кластеризация)

Реализации в sklearn (их много)

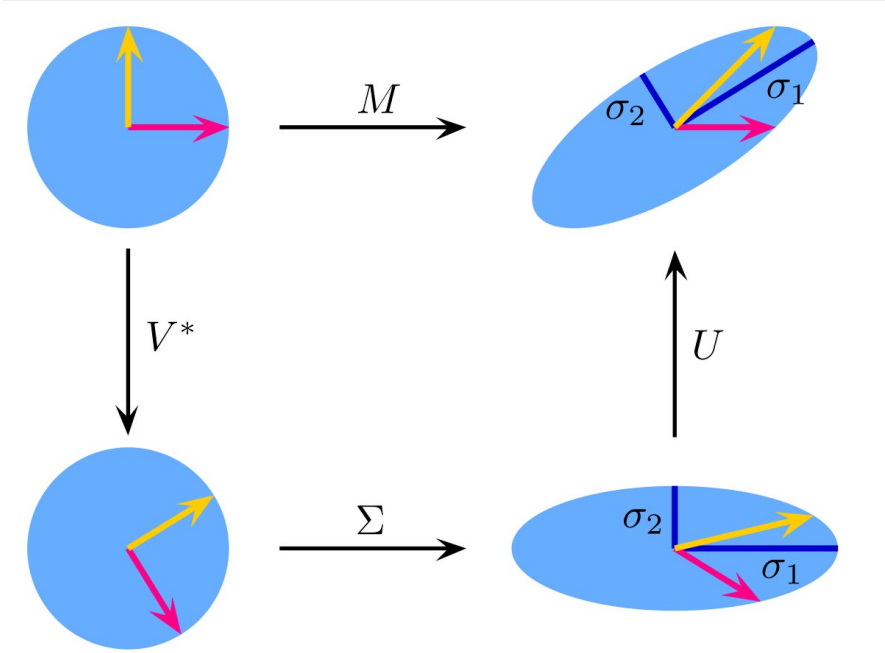
Разложения матриц

`decomposition.DictionaryLearning`([...])	Dictionary learning.
`decomposition.FactorAnalysis`([n_components, ...])	Factor Analysis (FA).
`decomposition.FastICA`([n_components, ...])	FastICA: a fast algorithm for Independent Component Analysis.
`decomposition.IncrementalPCA`([n_components, ...])	Incremental principal components analysis (IPCA).
`decomposition.KernelPCA`([n_components, ...])	Kernel Principal component analysis (KPCA) [R396fc7d924b8-1].
`decomposition.LatentDirichletAllocation`([...])	Latent Dirichlet Allocation with online variational Bayes algorithm.
`decomposition.MiniBatchDictionaryLearning`([...])	Mini-batch dictionary learning.
`decomposition.MiniBatchSparsePCA`([...])	Mini-batch Sparse Principal Components Analysis.
`decomposition.NMF`([n_components, init, ...])	Non-Negative Matrix Factorization (NMF).
`decomposition.MiniBatchNMF`([n_components, ...])	Mini-Batch Non-Negative Matrix Factorization (NMF).
`decomposition.PCA`([n_components, copy, ...])	Principal component analysis (PCA).
`decomposition.SparsePCA`([n_components, ...])	Sparse Principal Components Analysis (SparsePCA).
`decomposition.SparseCoder`(dictionary, *[, ...])	Sparse coding.
`decomposition.TruncatedSVD`([n_components, ...])	Dimensionality reduction using truncated SVD (aka LSA).

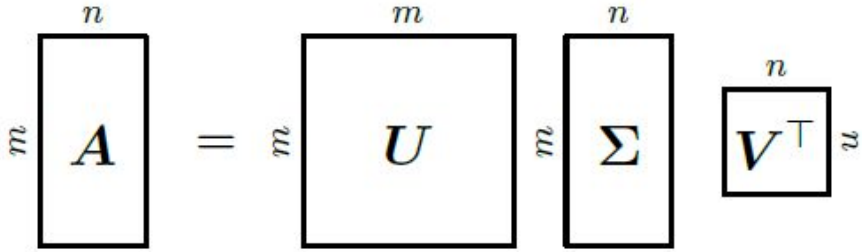
Многообразия

`manifold.Isomap`(*[, n_neighbors, radius, ...])	Isomap Embedding.
`manifold.LocallyLinearEmbedding`(*[, ...])	Locally Linear Embedding.
`manifold.MDS`([n_components, metric, n_init, ...])	Multidimensional scaling.
`manifold.SpectralEmbedding`([n_components, ...])	Spectral embedding for non-linear dimensionality reduction.
`manifold.TSNE`([n_components, perplexity, ...])	T-distributed Stochastic Neighbor Embedding.

Singular Value Decomposition (SVD)



$$M = U \cdot \Sigma \cdot V^*$$

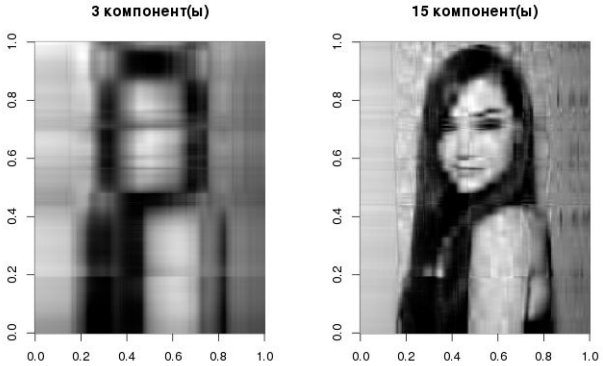


U, V - ортогональные матрицы, сигма - матрица собственных чисел.

Имеет геом. смысл (слева)

Как считать руками:

Зачем нужно SVD



Решение СЛАУ

- Приближаем матрицу *A* матрицей меньшего ранга *r*, учитывая то, что за основную часть информации матрицы *A* отвечают первые сингулярные числа.
- Таким образом получается сохранить наибольшую часть информации, задействовав меньше памяти. При этом отброшенные сингулярные числа считаем «шумовыми».
- Это является аналогом низкочастотной фильтрации данных.

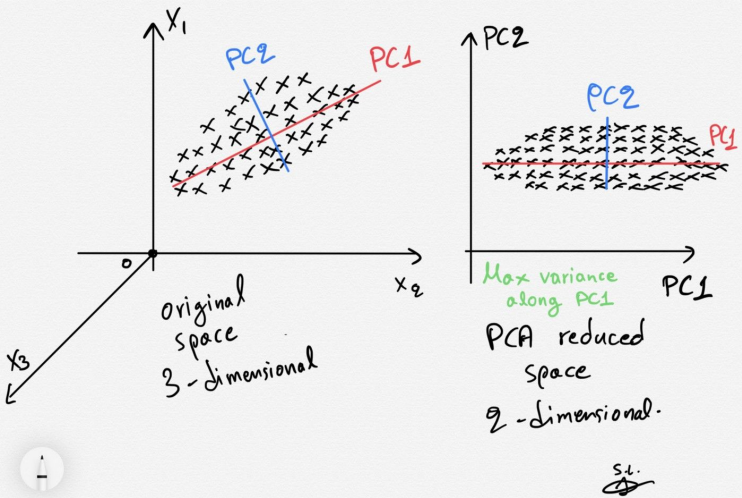
$$\underset{m \times n}{X} = \underset{m \times r}{U} \underset{r \times r}{\Sigma} \underset{r \times n}{V^{\mathsf{T}}}$$

Principal component analysis (PCA)

Человеческое объяснение от ВШЭ:

Убираем самые “изменчивые” из переменных, при этом получаяющиеся компоненты будут некоррелированны. Собственные вектора матрицы признаков образуют новые оси.

Самые первые главные компоненты будут иметь максимальную дисперсию. Следовательно выбрав N главных компонент вы максимально хорошо “объясняете” дисперсию в вашем датасете.

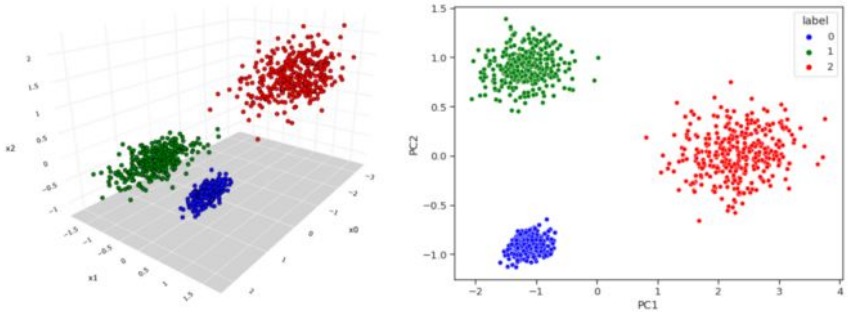


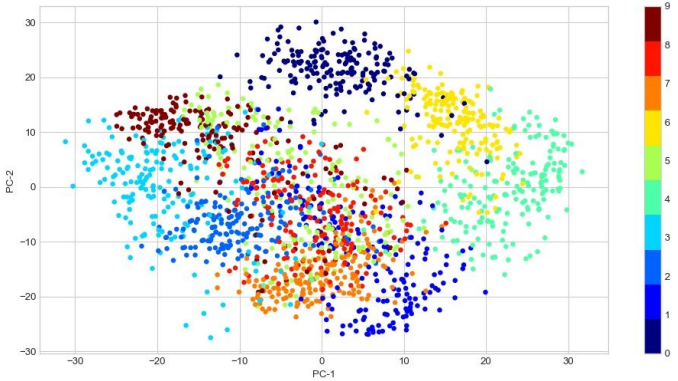
Хотим перейти от $X_1, \ldots, X_r$ к новым переменным $Y_1, \ldots, Y_r$ так, чтобы

- $M[Y_j] = 0$
- $\text{cov}[Y_j, Y_k] = 0$
- $D[Y_1] > D[Y_2] > \ldots > D[Y_r]$

$Y_1, \ldots, Y_r$ называют ***главными компонентами***.

PCA. Примеры





tSNE и UMAP. Примеры

Математика за ними сложна, но суть в том что они используют информацию о соседях точек и являются исключительно итеративными методами. Также они учитывают выбросы и подходят для многих задач, в том числе NLP.