# Шпаргалка по обучению без учителя / Концепции

# Cheatsheet (XeLaTeX)

Краткий справочник

April 5, 2025

## Contents



## Обучение без учителя (Unsupervised Learning): Введение

В отличие от обучения с учителем, здесь у нас **нет правильных ответов** (меток) для данных. Цель — найти **скрытые структуры**, закономерности или группы в самих данных. Представь, что тебе дали кучу разных носков, и ты должен рассортировать их по парам, не зная изначально, какие носки составляют пару — ты ищешь схожесть сам.

- **Основные задачи:** Понижение размерности, кластеризация, поиск аномалий, изучение ассоциативных правил.
- **Ключевые методы для старта:** PCA (понижение размерности) и K-Means/DBSCAN (кластеризация).

# 1 Метод Главных Компонент (PCA - Principal Component Analysis)

### Идея: Понижение размерности

Представь, что у тебя есть сложный многомерный объект (данные с большим числом признаков), и ты хочешь сделать его "плоскую", но информативную фотографию (понизить размерность до меньшего числа признаков). PCA именно это и делает — находит новые, самые информативные "ракурсы" или "проекции" данных. **Цель:** Уменьшить количество признаков (столбцов), сохранив при этом максимальное количество **вариативности (дисперсии)** исходных данных. **Почему дисперсия = информация?** В контексте PCA предполагается, что направление, вдоль которого данные сильнее всего "разбросаны" (имеют большую дисперсию), несет больше всего информации об их различиях. Если вдоль какого-то направления все точки почти одинаковы (малая дисперсия), это направление мало что добавляет к пониманию структуры данных.

## Как работает PCA (Механика и Интуиция)

PCA находит новые оси (называемые **главными компонентами**, PC) в пространстве исходных признаков. Вот как это происходит по шагам:

1. **Стандартизация данных:** *Критически важно!* Вычитаем среднее и делим на стандартное отклонение для **каждого** признака. **Зачем?** PCA чувствителен к масштабу. Без стандартизации признаки с большими значениями (например, доход в рублях) будут доминировать над признаками с малыми значениями (например, стаж в годах), даже если последние не менее важны по смыслу, так как дисперсия первых будет искусственно завышена.
2. **Расчет ковариационной матрицы:** Строится матрица, показывающая, как стандартизированные признаки изменяются *совместно*. На диагонали — дисперсия каждого признака (после стандартизации равна 1), вне диагонали — ковариация между парами признаков (показывает их линейную связь).
3. **Нахождение собственных векторов и значений:** Для ковариационной матрицы вычисляются её **собственные векторы** (*eigenvectors*) и **собственные значения** (*eigenvalues*).
   - **Собственный вектор:** Это **направление** в пространстве признаков.
   - **Собственное значение:** Это число, показывающее, **сколько дисперсии** исходных данных объясняется вдоль направления, заданного соответствующим собственным вектором.
4. **Сортировка и выбор компонент:** Собственные векторы сортируются по убыванию их собственных значений. Первый собственный вектор (с самым большим $\lambda$) задает направление **первой главной компоненты (PC1)** — направление максимальной дисперсии. Второй (с вторым по величине $\lambda$) задает **PC2**, ортогональную PC1 и объясняющую максимум *оставшейся* дисперсии, и т.д. Все главные компоненты ортогональны друг другу (не коррелируют).
5. **Проецирование:** Исходные (стандартизированные) данные проецируются на выбранные $k$ главных компонент (те, что с наибольшими $\lambda$). Результат проекции — это новый набор данных с $k$ признаками, где каждый новый признак является линейной комбинацией исходных.

**Аналогия "Эллипс и Тени":** Представь облако точек данных как эллипс. Главные компоненты — это оси этого эллипса (самая длинная — PC1, следующая перпендикулярная ей — PC2 и т.д.). PCA находит эти оси и "поворачивает" данные так, чтобы оси совпали с осями координат. Затем он "отбрасывает" оси с наименьшей длиной (наименьшей дисперсией), оставляя проекцию на самые важные (длинные) оси.

## Когда применять PCA и Как выбрать компоненты?

- **Применения:** Визуализация (до 2D/3D), уменьшение шума, борьба с мультиколлинеарностью, ускорение обучения других моделей, сжатие данных.
- **Чувствительность к масштабу: Всегда стандартизируйте данные перед PCA!** Иначе признаки с большим масштабом "перетянут" всю дисперсию на себя.
- **Выбор количества компонент (k):**
  - **Доля объясненной дисперсии:** Смотрят на кумулятивную (накопленную) долю $\sum_{i=1}^{k} \lambda_i / \sum_{j=1}^{N} \lambda_j$. Выбирают $k$ так, чтобы объяснить достаточный процент (например, 90-99%) общей дисперсии.
  - **Метод Локтя:** Ищут "изгиб" на графике кумулятивной объясненной дисперсии или на графике самих собственных значений ($\lambda_i$ от $i$, т.н. *scree plot*). Точка изгиба показывает, где добавление новой компоненты перестает давать существенный прирост информации.
  - **Исходя из задачи:** Для визуализации $k = 2$ или $k = 3$. Для других задач можно подбирать $k$ по кросс-валидации для *конечной* ML-модели.

# 2 Кластеризация K-Means

### Идея: Кластеризация

Разделить все объекты на заранее заданное число ($K$) групп (кластеров) так, чтобы объекты внутри кластера были максимально похожи (близки) друг на друга, а объекты из разных кластеров — максимально различны. Цель — минимизировать суммарное квадратичное расстояние от точек до центров их кластеров (WCSS). **Аналогия "Почтовые отделения":** Открыть $K$ почтовых отделений (центроидов) в городе так, чтобы суммарное *квадратичное* расстояние от всех жителей (точек данных) до ближайшего к ним отделения было минимальным.

## Как работает K-Means (Алгоритм)

Это итеративный алгоритм:

1. **Инициализация:** Выбрать $K$. Разместить $K$ **центроидов** (начальных центров кластеров). Лучше использовать "умную" инициализацию **K-Means++** (размещает центроиды подальше друг от друга), чем чисто случайную.
2. **Шаг присваивания (Assignment):** *Цикл по каждой точке данных:* Вычислить расстояние от точки до **каждого** из $K$ центроидов. Присвоить точку тому кластеру, чей центроид **ближе всего** (обычно по евклидову расстоянию).
3. **Шаг обновления (Update):** *Цикл по каждому кластеру:* Найти новый центр масс (среднее арифметическое координат) **всех** точек, которые были присвоены этому кластеру на шаге 2. Переместить центроид кластера в эту новую точку.
4. **Повторение:** Повторять шаги 2 и 3 до тех пор, пока центроиды почти не перестанут смещаться или точки не перестанут менять кластеры (или достигнуто макс. число итераций).

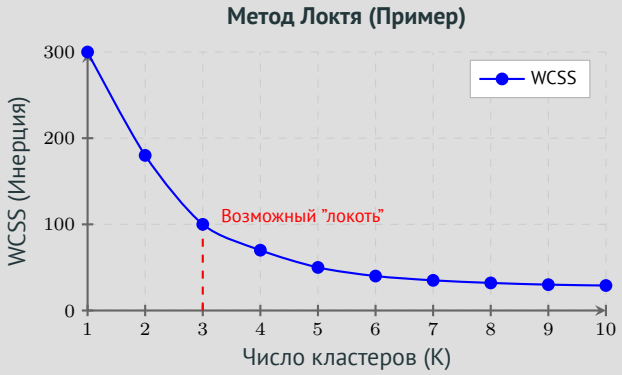
**Метрики расстояния:** Чаще всего используется **Евклидово** расстояние ($\sqrt{\sum(x_i - y_i)^2}$), которое хорошо работает для сферических кластеров. Иногда могут быть полезны: **Манхэттенское** ($\sum |x_i - y_i|$, "расстояние городских кварталов") или **Косинусное** ($1 - \cos(\theta)$, измеряет угол, полезно для текстов).

## 2.1 Выбор количества кластеров (K)

### Как подобрать K?

Выбор оптимального K — нетривиальная задача. Популярные методы:

- **Метод Локтя (Elbow Method):** Строится график **WCSS** (Within-Cluster Sum of Squares - сумма квадратов расстояний от точек до центроидов их кластеров) от числа кластеров K. Ищется точка "изгиба" (локтя), после которой WCSS убывает значительно медленнее.
- **Метод Силуэта (Silhouette Method):** Запускают K-Means для разных K. Для каждого K вычисляют **средний Силуэт** по всем точкам. Выбирают то K, для которого средний Силуэт максимален (ближе к 1). См. ниже подробнее.
- **Знание предметной области:** Иногда K можно определить из контекста задачи.




## 2.2 Оценка качества: Силуэт (Silhouette Score)

### Интуиция и Расчет Силуэта

Метрика "Силуэт" оценивает, насколько хорошо точка "сидит" в своем кластере по сравнению с соседними. Рассчитывается для каждой точки $i$:

- $a(i)$: Среднее расстояние от точки $i$ до **всех других точек в её собственном** кластере. *(Насколько точка близка к "своим"? Чем меньше, тем лучше).*
- $b(i)$: **Минимальное** из средних расстояний от точки $i$ до **всех точек в каждом из других** ("соседних") кластеров. *(Насколько точка далека от "ближайших чужих"? Чем больше, тем лучше).*

**Силуэт точки** $i$:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

**Интерпретация значений** $s(i)$**:** От -1 до +1.

- $\approx +1$: Точка плотно сидит в своем кластере, далеко от других (хорошо).
- $\approx 0$: Точка на границе между кластерами.
- $\approx -1$: Точка, вероятно, попала не в тот кластер (плохо).

**Общий Силуэт:** Усредняют $s(i)$ по всем точкам. Чем ближе средний силуэт к 1, тем лучше разделение на кластеры для данного K. **Аналогия "Районы города":** Высокий силуэт жителя ($s(i) \approx 1$) — он живет в центре четко очерченного района, далеко от других районов. Низкий ($s(i) \approx 0$) — живет на границе. Отрицательный ($s(i) \approx -1$) — живет ближе к центру соседнего района, чем к своему.

## 2.3 Плюсы и Минусы K-Means

### Преимущества

- **Простота и скорость:** Легко понять, реализовать, относительно быстро работает.
- **Интерпретируемость:** Концепция центроидов как "представителей" кластера понятна.

### Ограничения

- **Чувствительность к инициализации:** Результат зависит от начального положения центроидов. **Решение:** Использовать K-Means++ и многократные запуски с разной инициализацией (`n_init` в scikit-learn).
- **Необходимость задавать K:** Нужно знать K заранее или подбирать эвристиками.
- **Предположение о форме кластеров:** K-Means ищет **выпуклые, сферические** кластеры примерно одинакового размера. Плохо работает с вытянутыми, вогнутыми кластерами или кластерами разной плотности. (Для таких случаев лучше DBSCAN или спектральная кластеризация).
- **Чувствительность к выбросам:** Выбросы могут сильно смещать центроиды.

# 3 Кластеризация DBSCAN (Density-Based)

### Идея: Кластеризация на основе плотности

В отличие от K-Means, DBSCAN не ищет центры, а находит **плотные регионы** точек, разделенные областями с низкой плотностью. Позволяет находить кластеры **произвольной формы** и автоматически определяет **выбросы (шум)**. **Аналогия "Поиск островов":** Представь карту с точками-домами. DBSCAN ищет "острова" (кластеры), где дома стоят близко друг к другу, отделенные "водой" (разреженные области). Дома, стоящие совсем одиноко в "воде", считаются шумом.

### Как работает DBSCAN (Концепция)

Требует два параметра:

- `eps` ($\epsilon$): Радиус окрестности. Максимальное расстояние между двумя точками, чтобы считать их соседями.
- `min_samples`: Минимальное число соседей (включая саму точку) в $\epsilon$-окрестности, чтобы точка считалась "основной".

Основные понятия:

- **Основная точка (Core Point):** Точка, у которой в $\epsilon$-окрестности $\geq$ `min_samples` точек. Находится внутри плотного региона.
- **Граничная точка (Border Point):** Точка, у которой < `min_samples` соседей, но она сама является соседом какой-либо *основной* точки. Лежит на краю кластера.
- **Шум (Noise Point):** Точка, которая не является ни основной, ни граничной. Выброс.

**Алгоритм (кратко):** Начинает с произвольной точки. Если она основная, "выращивает" кластер, рекурсивно добавляя всех достижимых по плотности соседей (основных и граничных). Если точка не основная, переходит к следующей. Точки, не попавшие ни в один кластер, помечаются как шум.

### Плюсы и Минусы DBSCAN

- **Плюсы:** Не нужно задавать K, находит кластеры сложной формы, устойчив к выбросам (явно их определяет).
- **Минусы:** Чувствителен к выбору `eps` и `min_samples` (требует подбора), плохо работает с кластерами сильно разной плотности, может быть медленным на очень больших данных (хотя есть оптимизации).