Машинное обучение без учителя: методы кластеризации, методы понижения размерности, рекомендательные системы

Цель занятия

После освоения темы:

- вы обзорно познакомитесь с методами кластеризации и методами понижения размерности, принципами построения рекомендательных систем;
- сможете реализовать методы кластерного анализа на примере искусственных данных, выполнить расчет оценок качества кластеризации с помощью библиотек Python;
- сможете реализовать методы понижения размерности применительно к датасету с помощью Python;
- сможете составить матрицу рейтингов с помощью Python и выполнять операции с ней.

План занятия

Используемые термины

Кластеризация — это метод машинного обучения без учителя, который позволяет группировать объекты по их признакам.

Метод кластеризации k-средних — это один из наиболее распространенных алгоритмов обучения без учителя, который используется для разбиения набора данных на k кластеров.

Иерархическая кластеризация — метод машинного обучения, который позволяет группировать объекты в кластеры на основе их сходства.

Агломеративный алгоритм — один из методов иерархической кластеризации, который начинает работу с каждого объекта в отдельном кластере и последовательно объединяет близлежащие кластеры в более крупные, пока не получит кластер, содержащий все объекты.

Elbow Method (метод локтя) — метод выбора оптимального числа кластеров для кластеризации, в котором используется график зависимости внутрикластерных расстояний от числа кластеров.

T-SNE (t-distributed stochastic neighbor embedding) — метод многомерного шкалирования, который широко используется для визуализации высокоразмерных данных.

Рекомендательные системы (recommendation systems) — это инструменты, которые используются для предоставления персонализированных рекомендаций пользователям на основе их предпочтений, поведения и истории взаимодействия с системой.

Методы коллаборативной фильтрации — это подходы к рекомендательным системам, которые используют историю взаимодействия пользователей с элементами, чтобы предсказать, что еще понравится пользователям в будущем.

User-based — метод, который использует историю оценок пользователей для поиска других пользователей с похожими предпочтениями, чтобы рекомендовать первым те элементы, которые понравились вторым.

Item-based — метод, который использует историю оценок пользователей, чтобы найти и порекомендовать им элементы, похожие на те, что понравились им ранее.

Конспект занятия

1. Обучение без учителя. Методы кластеризации

Обучение без учителя предполагает большое количество разных методов:

- кластеризация (Clustering);
- понижение размерности (Dimensionality Reduction);
- ассоциативные правила (Association Rules);
- детектирование выбросов (Outlier Detection);
- обнаружение аномалий (Anomaly Detection);
- поиск аномальных сообществ в графах (Community Detection);
- визуализация (Visualization).

Кластеризация

Кластеризация — это метод машинного обучения без учителя, который позволяет группировать объекты по их признакам. Он относится к задаче разделения данных на группы (кластеры) так, чтобы объекты в одной группе были максимально похожими, а объекты в разных группах — максимально различными.

Применить кластеризацию можно в случаях, когда нужно:

- выявить неявные группы в данных;
- выделить аномальные объекты;
- сжать данные путем представления их в виде кластеров и в других ситуациях.

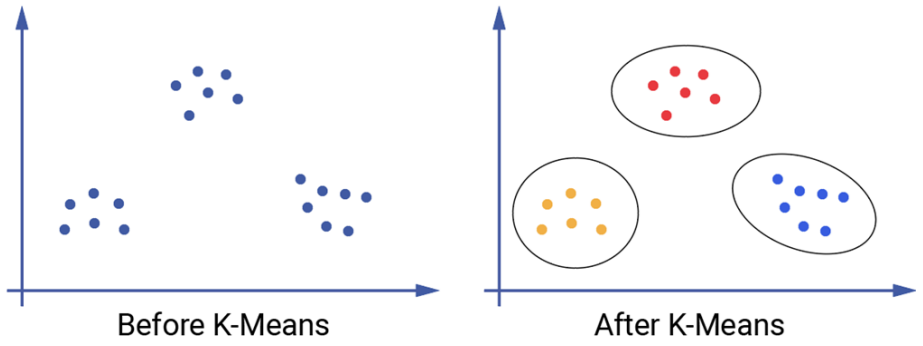
Основные этапы кластеризации:

1. Выбор метода кластеризации.
2. Выбор меры близости, выбор числа кластеров.
3. Визуализация и интерпретация полученных результатов.

Кластеризация может быть реализована различными методами: иерархическая кластеризация, кластеризация на основе плотности, метод k-средних и другие. Каждый метод имеет свои преимущества и недостатки. Выбор метода зависит от задачи и данных.

Метод k-средних

Метод кластеризации k-средних — один из наиболее распространенных алгоритмов обучения без учителя, который используется для разбиения набора данных на *k* кластеров. Он основан на минимизации суммарного квадратичного отклонения между точками данных и центрами кластеров.

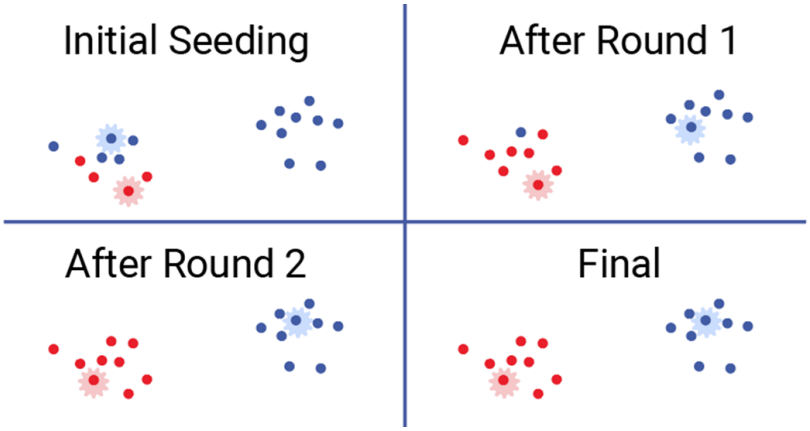


Алгоритм k-средних:

1. Выбрать k случайных точек данных в качестве начальных центроидов кластеров.
2. Каждая точка данных относится к ближайшему кластеру на основе евклидового расстояния до центроида.
3. Итеративно пересчитать центроид каждого кластера путем нахождения среднего значения всех точек данных, относящихся к этому кластеру.

Процесс повторяется до тех пор, пока центроиды не перестанут изменяться или пока не достигается максимальное количество итераций, которое мы можем установить как гиперпараметр.

Пример работы алгоритма k-средних показан на рисунке. Мы предполагаем, что в данных есть два кластера. Первоначально центроиды кластеров устанавливаются случайным образом. С каждой последующей итерацией центроиды перемещаются, пока не находят оптимальное значение, где каждая точка будет находиться максимально близко к соответствующему кластеру.



Посмотрим на алгоритм реализации кластерного анализа методом k-средних на примере искусственных данных. Импортируем необходимые библиотеки:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

Создаем случайные данные:

```
X = np.random.rand(100, 2)
```

Создаем объект `kmeans` с тремя кластерами:

```
kmeans = KMeans(n_clusters=3)
```

Выполняем кластеризацию:

```
kmeans.fit(X)
```

Получаем метки кластеров:

```
labels = kmeans.labels_
```

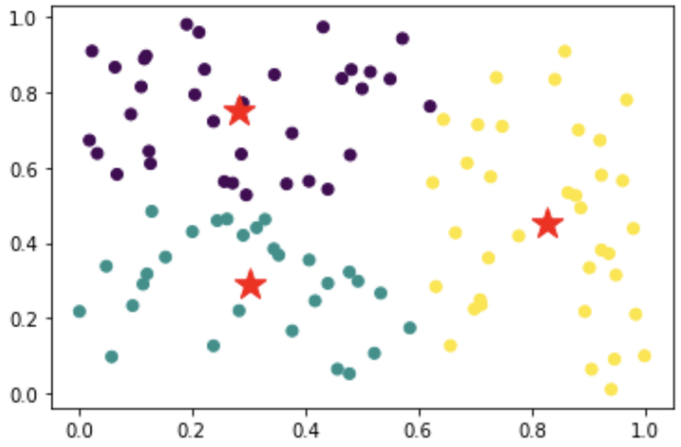
Получаем координаты центроидов:

```
centroids = kmeans.cluster_centers_
```

Визуализируем кластеры:

```
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300, c='r')
plt.show()
```

Звездочками обозначены центроиды данных кластеров:



Иерархическая кластеризация

Иерархическая кластеризация — метод машинного обучения, который позволяет группировать объекты в кластеры на основе их сходства. В отличие от метода k-средних, определяющего количество кластеров заранее, в иерархической кластеризации мы можем получить иерархическую структуру кластеров, которая поможет в анализе данных.

Агломеративный алгоритм — один из методов иерархической кластеризации, который начинает работу с каждого объекта в отдельном кластере и последовательно объединяет близлежащие кластеры в более крупные, пока не будет получен один кластер, содержащий все объекты. Этот процесс может быть представлен **дендрограммой**, показывающей иерархическую структуру кластеров.

Механика агломеративного алгоритма:

1. Начать с каждого объекта в отдельном кластере.
2. Найти два ближайших кластера (наименее удаленных друг от друга объектов) и объединить их в один кластер.
3. Повторить шаг 2, пока все объекты не будут объединены в один кластер.

В качестве метрики расстояния между кластерами можно использовать евклидово расстояние, расстояние Манхэттен, корреляционное расстояние и другие.

Попробуем реализовать алгоритм на примере искусственных данных:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
```

Создаем искусственные данные:

```
np.random.seed(123)
X = np.random.randn(10, 2)
```

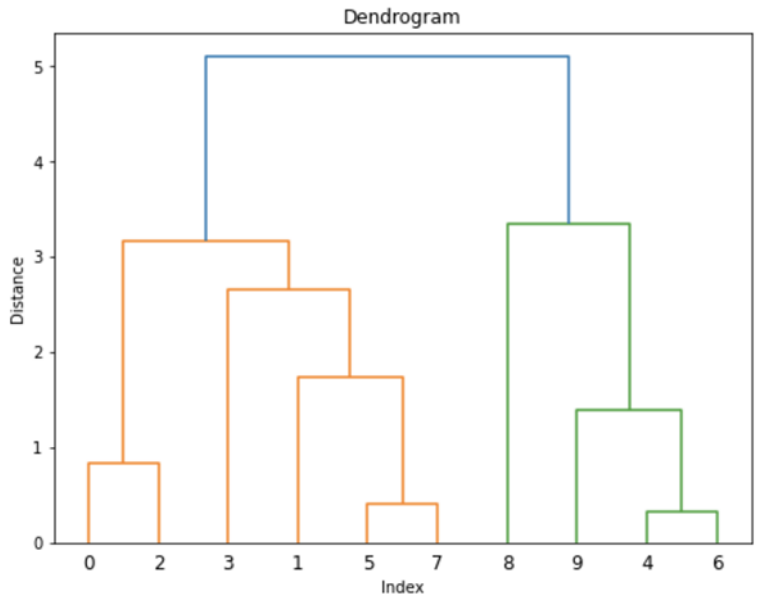
Вычисляем матрицу расстояний между объектами:

```
Z = linkage(X, 'ward')
```

Строим дендрограмму:

```
fig, ax = plt.subplots(figsize=(8, 6))
dendrogram(Z)
plt.title("Dendrogram")
plt.xlabel("Index")
plt.ylabel("Distance")
plt.show()
```

Результат:



На основании полученной дендрограммы мы можем выбрать то количество кластеров, которое согласно нашей логике и нашему представлению о данных, отвечает аналитическим задачам.

Алгоритм DBSCAN

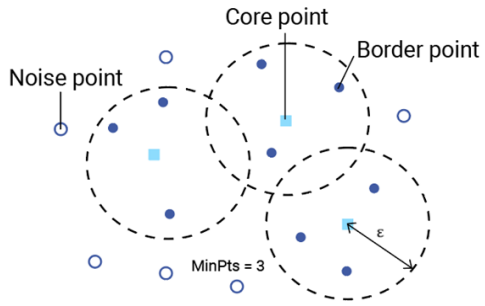
Алгоритм DBSCAN (Density-Based Spatial Clustering of Applications with Noise) является одним из наиболее популярных алгоритмов кластеризации.

Его идея: кластер — это область пространства, в которой плотность точек выше некоторого порогового значения.

В основе алгоритма DBSCAN лежат два параметра: **радиус (eps)** и минимальное **количество точек (min_samples)**. Для каждой точки алгоритм вычисляет, сколько точек находятся в радиусе eps вокруг нее.

Если число больше или равно min_samples, то эта точка считается основной (core point). Если же число меньше min_samples, но точка находится в радиусе eps от какой-то основной точки, то эта точка считается граничной (border point).

Точки, которые не являются ни основными, ни граничными, считаются выбросами (noise point).



Алгоритм работы DBSCAN:

1. Выбрать случайную точку из данных.
2. Найти все точки, которые находятся в радиусе eps от выбранной точки, и определить, является ли эта точка основной или граничной.
   Если точка является основной, то алгоритм вычисляет все точки, которые находятся в радиусе eps от этой точки. Он продолжает этот процесс до тех пор, пока все основные точки и связанные с ними граничные точки не будут объединены в кластер.
3. Выбрать следующую рассмотренную точку и повторить процесс.

Как только все точки будут рассмотрены, алгоритм DBSCAN возвращает набор кластеров.

Рассмотрим применение алгоритма на примере искусственных данных:

```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
```

Генерируем искусственные данные:

```
X, y = make_blobs(n_samples=100, centers=3, random_state=42)
```

Создаем экземпляр алгоритма `dbscan`:

```
dbscan = DBSCAN(eps=2, min_samples=5)
```

Обучаем модель:

```
dbscan.fit(X)
```

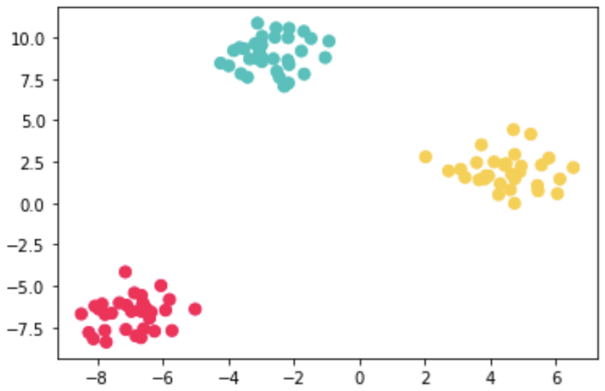
Получаем метки кластеров:

```
labels = dbscan.labels_
```

Визуализируем результаты:

```
colors = np.array(['#FF0054', '#FBD039', '#23C2BC'])
plt.scatter(X[:, 0], X[:, 1], s=50, color=colors[labels])
plt.show()
```

Результат:



Сравнение алгоритмов кластерного анализа

Сравним рассмотренные алгоритмы:

Алгоритм	**Преимущества**	**Недостатки**
K-средних	Прост в реализации и интерпретации. Быстрый при большом количестве признаков и кластеров	Чувствителен к начальной инициализации центров кластеров. Неустойчив к выбросам.

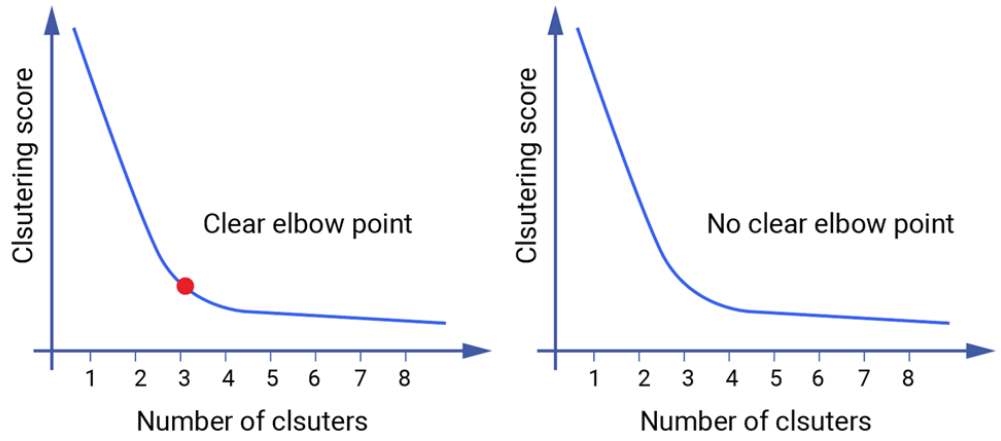
		Не всегда находит оптимальное число кластеров
ИКА	Позволяет находить кластеры на всех уровнях иерархии. Не требует задания числа кластеров	Не всегда прост в интерпретации. Неэффективен при большом числе объектов. Неустойчив к шуму и выбросам
DBSCAN	Устойчив к шуму и выбросам. Не требует знания числа кластеров. Обнаруживает кластеры произвольной формы	Чувствителен к параметрам eps и min_samples. Не всегда прост в интерпретации. Неэффективен при большом числе признаков

Оценка качества кластеризации

Кластеризация осуществляется не полностью случайно, в ней есть некоторые законы и методы, которые позволяют подбирать оптимальные параметры.

Elbow Method (метод локтя) — метод выбора оптимального числа кластеров для кластеризации, в котором используется график зависимости внутрикластерных расстояний от числа кластеров.

По этому графику можно определить точку, где изменение внутрикластерного расстояния перестает быть значительным, — точку локтя. Это число кластеров считается оптимальным для данной задачи.



Метрики для оценки качества кластеризации:

- **Silhouette Score (силуэтная оценка)** использует меру сходства и различия между объектами внутри кластера и между кластерами. Более высокий показатель Silhouette Score (от -1 до 1) указывает на более четкое разделение кластеров.
- **Calinski-Harabasz Index** — это метрика, которая оценивает качество кластеризации. Она использует отношение между дисперсиями внутри и между кластерами. Более высокий показатель Calinski-Harabasz Index указывает на более четкое разделение кластеров.
- **Davies-Bouldin Index** — также оценивает качество кластеризации. Использует отношение между суммарным внутрикластерным расстоянием и межкластерным расстоянием. Более низкий показатель Davies-Bouldin Index указывает на более четкое разделение кластеров.

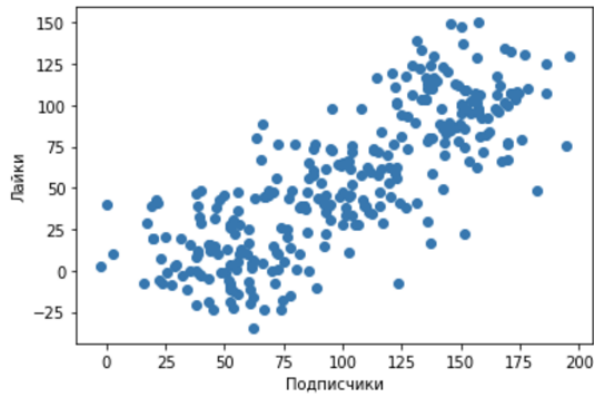
При оценке кластеризации мы стремимся максимизировать Silhouette Score и Calinski-Harabasz Index, и минимизировать Davies-Bouldin Index.

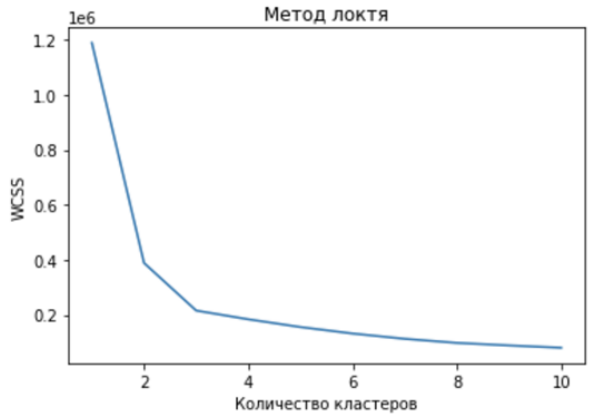
Рассмотрим расчет оценок качества кластеризации на примере сгенерированных данных. Мы генерируем данные о соотношении числа лайков на выборке страниц. Задача состоит в кластеризации лайков по страницам.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering,
DBSCAN
from sklearn.metrics import silhouette_score,
```

```
calinski_harabasz_score, davies_bouldin_score
# генерируем данные
subscribers = np.concatenate((np.random.normal(50, 20, 100),
                              np.random.normal(100, 20, 100),
                              np.random.normal(150, 20, 100)))
# Генерируем случайный вектор лайков
likes = np.concatenate((np.random.normal(10, 20, 100),
                        np.random.normal(50, 20, 100),
                        np.random.normal(100, 20, 100)))
# Визуализируем данные
plt.scatter(subscribers, likes)
plt.xlabel('Подписчики')
plt.ylabel('Лайки')
plt.show()
X = np.column_stack((likes, subscribers))
# расчет методом локтя
wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters=i,
init='k-means++',
max_iter=300, n_init=10, random_state=0)
   kmeans.fit(X)
   wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Метод локтя')
plt.xlabel('Количество кластеров')
plt.ylabel('WCSS')
plt.show()
```

Получим график исходных данных и результат расчета методом локтя:





Далее применим рассмотренные ранее алгоритмы кластеризации:

```
# K-Means
kmeans = KMeans(n_clusters=3, random_state=42).fit(X)
kmeans_silhouette = silhouette_score(X, kmeans.labels_)
kmeans_calinski_harabasz = calinski_harabasz_score(X,
kmeans.labels_)
kmeans_davies_bouldin = davies_bouldin_score(X, kmeans.labels_)
# Hierarchical Clustering
hierarchical = AgglomerativeClustering(n_clusters=3).fit(X)
hierarchical_silhouette = silhouette_score(X,
hierarchical.labels_)
hierarchical_calinski_harabasz = calinski_harabasz_score(X,
hierarchical.labels_)
hierarchical_davies_bouldin = davies_bouldin_score(X,
hierarchical.labels_)

# DBSCAN
dbscan = DBSCAN(eps=10, min_samples=10).fit(X)
dbscan_silhouette = silhouette_score(X, dbscan.labels_)
dbscan_calinski_harabasz = calinski_harabasz_score(X,
```

```
dbscan.labels_)
dbscan_davies_bouldin = davies_bouldin_score(X, dbscan.labels_)
```

Результат:

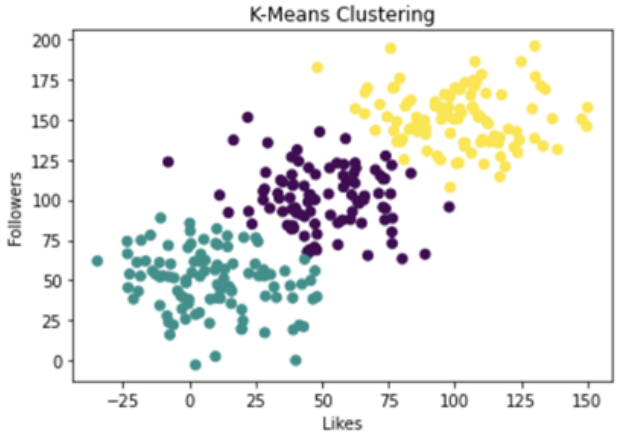
```
K-Means:
Silhouette Score: 0.49901500740497035
Calinski-Harabasz Index: 670.5935498821982
Davies-Bouldin Index: 0.7035457380329934
Hierarchical Clustering:
Silhouette Score: 0.47334187262007515
Calinski-Harabasz Index: 623.1354584434505
Davies-Bouldin Index: 0.7324263873806057
DBSCAN:
Silhouette Score: 0.0911273776050679
Calinski-Harabasz Index: 60.1896625143111
Davies-Bouldin Index: 3.629487009006097
```

Для каждого из алгоритмов визуализируем исходную кластеризацию, чтобы сравнить, как были идентифицированы классы загруженных объектов.

K-средних:

```
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_)
plt.title("K-Means Clustering")
plt.xlabel("Likes")
plt.ylabel("Followers")
plt.show()
```

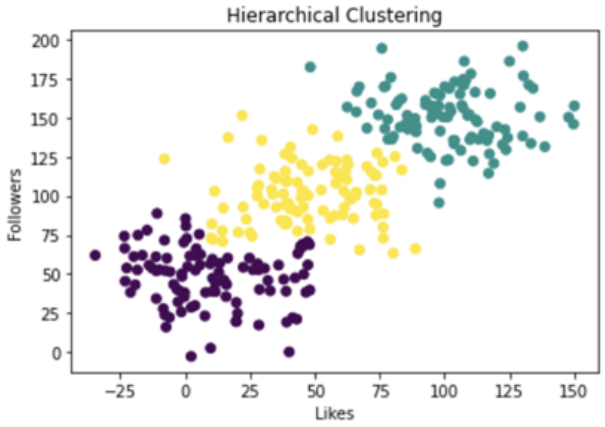
Результат визуализации:



Иерархическая кластеризация:

```
plt.scatter(X[:, 0], X[:, 1], c=hierarchical.labels_)
plt.title("Hierarchical Clustering")
plt.xlabel("Likes")
plt.ylabel("Followers")
plt.show()
```

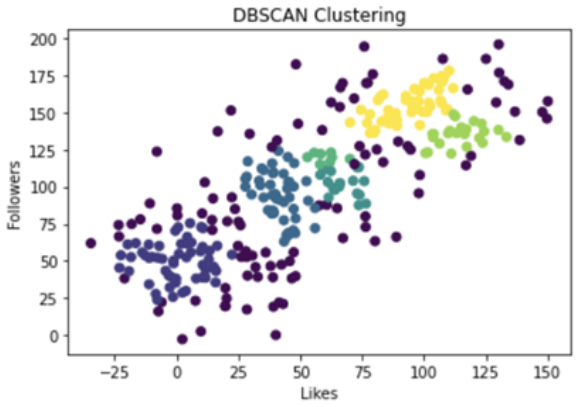
Результат визуализации:



Алгоритм DBSCAN:

```
plt.scatter(X[:, 0], X[:, 1], c=dbscan.labels_)
plt.title("DBSCAN Clustering")
plt.xlabel("Likes")
plt.ylabel("Followers")
plt.show()
```

Результат визуализации:



2. Методы понижения размерности

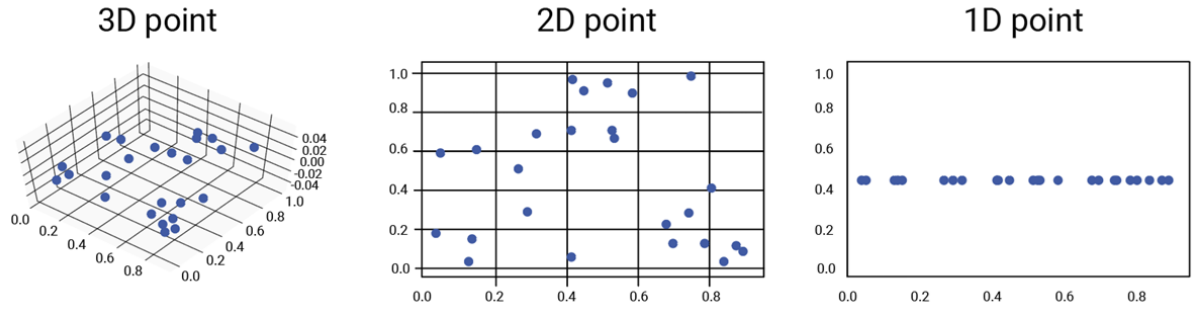
Методы понижения размерности используются, чтобы уменьшить количество признаков в наборе данных. Это позволяет представить данные в пространстве меньшей размерности, что может быть полезно во многих ситуациях:

- улучшить производительность и качество модели;
- визуализировать данные;
- сжать данные.

Метод главных компонент (Principal Component Analysis)

Метод главных компонент (PCA) является одним из наиболее распространенных методов понижения размерности. Используется для поиска линейных комбинаций исходных признаков, которые лучше всего описывают вариацию данных.

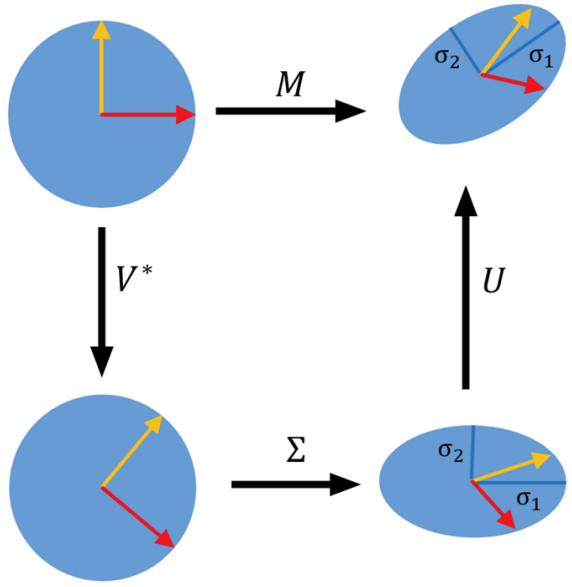
Идея метода: найти новый набор признаков (главные компоненты), которые являются линейными комбинациями исходных признаков и обладают свойством максимального объяснения вариации данных.



Механика анализа:

1. Отцентровать данные.
2. Вычислить ковариационную матрицу — она показывает связь между исходными признаками.
3. Вычислить собственные значения и собственные векторы.
4. Отсортировать главные компоненты в порядке убывания их объяснительной способности.
5. Выбрать количество главных компонент.
6. Спроецировать данные на выбранные главные компоненты.

Это довольно сложный алгоритм, в своем основании содержит **сингулярное разложение матрицы**.



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

Рассмотрим применение PCA на примере датасета рака груди. Датасет содержит информацию о множестве свойств клеток, в том числе раковая она или нет:

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

Загружаем данные Breast Cancer Wisconsin:

```
breast_cancer = load_breast_cancer()
```

Нормализуем данные:

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(breast_cancer.data)
```

Применяем метод главных компонент (PCA):

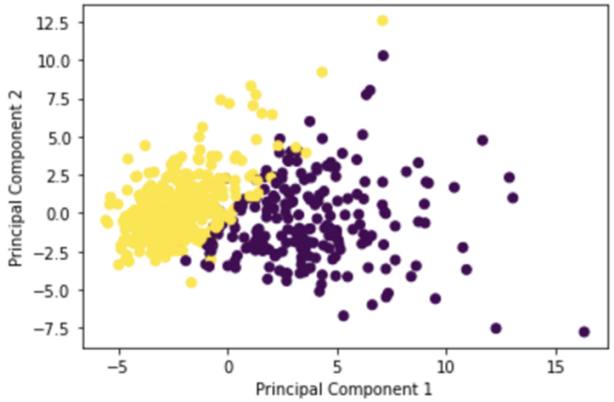
```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(data_scaled)
```

Визуализируем данные:

```
plt.scatter(principal_components[:, 0], principal_components[:,
1], c=breast_cancer.target)
```

```
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

Результат визуализации:



Две главные компоненты хорошо представляют многообразие существующих клеток. На картинке желтые клетки — раковые, фиолетовые — не раковые.

В контексте метода главных компонент нам важно понятие факторной нагрузки.

Факторная нагрузка показывает, в какой степени тот или иной признак вносит вклад в формирование компонент. Посмотрим в нашем примере вклад всех признаков в первую и вторую главные компоненты.

Рассчитываем факторные нагрузки:

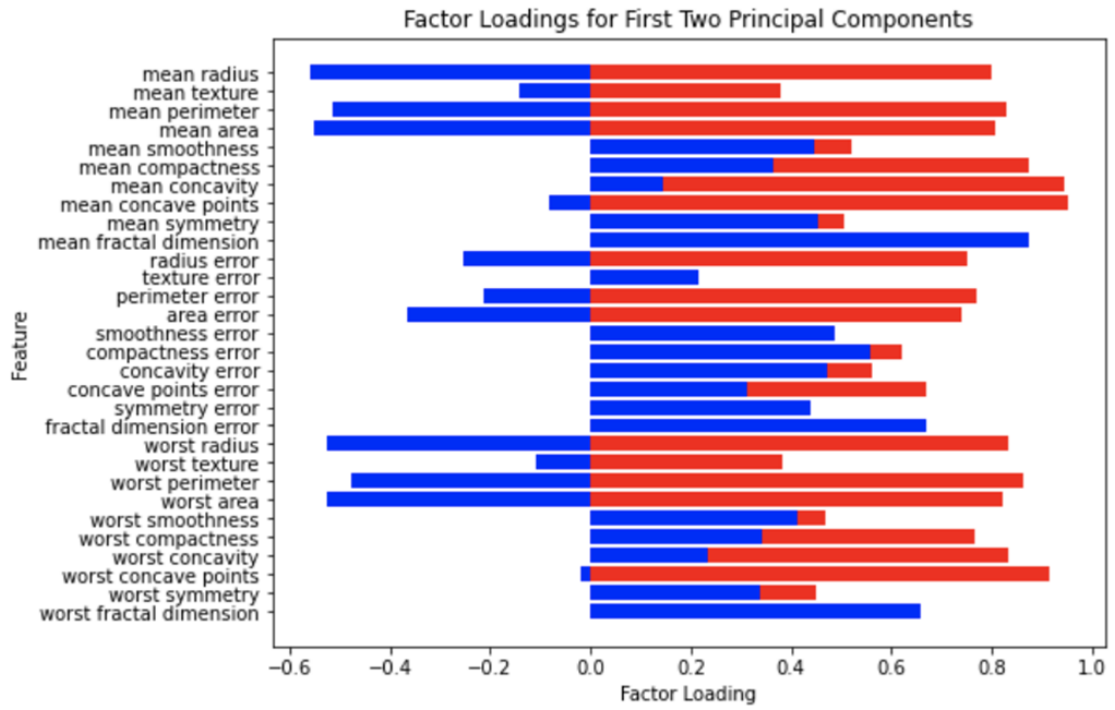
```
Vt = pca.components_
factor_loadings = Vt.T * np.sqrt(pca.explained_variance_)
```

Строим график факторных нагрузок:

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.barh(range(factor_loadings.shape[0]), factor_loadings[:, 0],
color='r')
ax.barh(range(factor_loadings.shape[0]), factor_loadings[:, 1],
color='b')
ax.set_yticks(range(factor_loadings.shape[0]))
ax.set_yticklabels(list(breast_cancer.feature_names))
ax.invert_yaxis()
ax.set_xlabel('Factor Loading')
ax.set_ylabel('Feature')
```

```
ax.set_title('Factor Loadings for First Two Principal Components')
plt.show()
```

На графике красным цветом обозначены факторные нагрузки всех факторов первой компоненты, синим — для второй компоненты:



Многомерное шкалирование t-SNE

T-SNE (t-distributed stochastic neighbor embedding) — метод многомерного шкалирования, который широко используется для визуализации высокоразмерных данных.

T-SNE является нелинейным методом сжатия данных, который позволяет проецировать данные из многомерного пространства на двух- или трехмерное пространство для визуализации. Он основан на концепции распределения вероятностей, которая описывает относительные расстояния между точками данных в исходном и проецированном пространствах.

T-SNE стремится сохранить близкие точки близко и разделять далекие точки в проекции, что выявляет структуру данных.

T-SNE имеет несколько параметров:

- число компонентов в проекции;
- параметры для расчета распределения вероятностей;
- параметры оптимизации.

Выбор правильных параметров существенно влияет на качество проекции и результаты анализа.

Рассмотрим пример с данными по доброкачественности или злокачественности клеток:

```
from sklearn.manifold import TSNE
```

Получаем матрицу признаков и соответствующих меток классов:

```
X = breast_cancer.data
y = breast_cancer.target
```

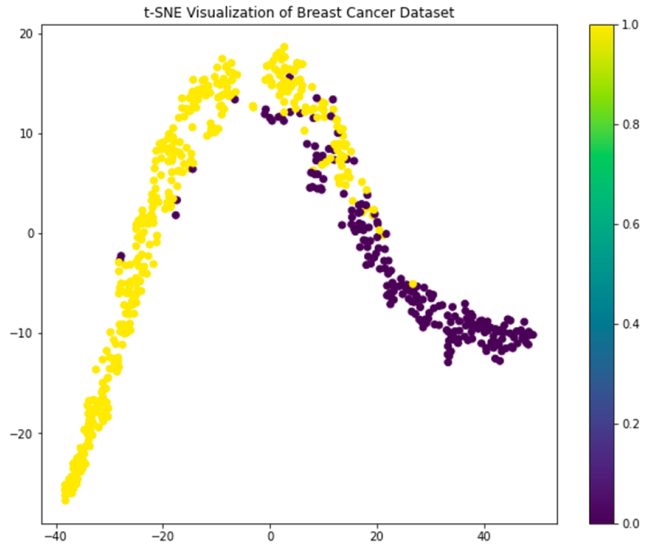
Применяем алгоритм t-SNE для снижения размерности данных до двух:

```
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)
```

Строим диаграммы рассеяния:

```
plt.figure(figsize=(10, 8))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis')
plt.colorbar()
plt.title('t-SNE Visualization of Breast Cancer Dataset')
plt.show()
```

Получаем график, который достаточно выгодным образом разделяет данные на два класса. Мы видим, что классы не так уж сильно пересекаются.



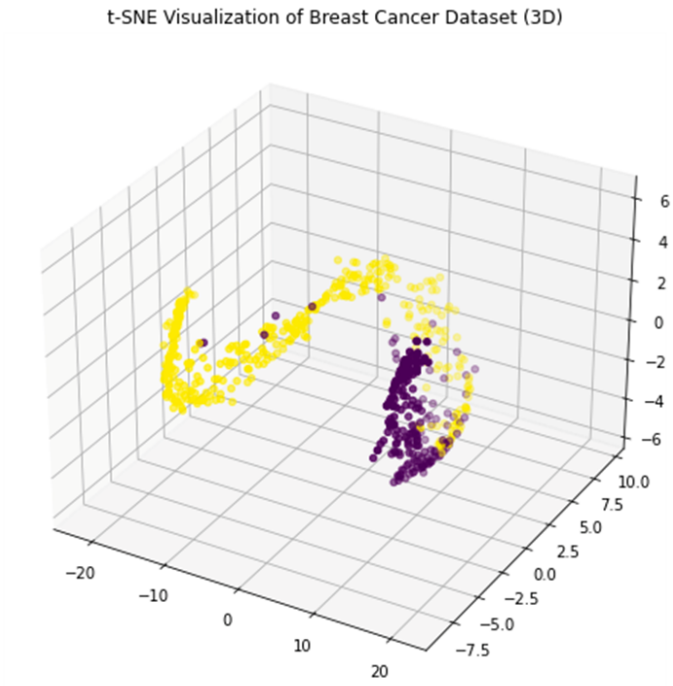
Применяем алгоритм t-SNE для снижения размерности данных до трех:

```
tsne = TSNE(n_components=3, random_state=42)
X_tsne = tsne.fit_transform(X)
```

Строим диаграммы рассеяния в трех измерениях:

```
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_tsne[:, 0], X_tsne[:, 1], X_tsne[:, 2], c=y,
cmap='viridis')
ax.set_title('t-SNE Visualization of Breast Cancer Dataset (3D)')
plt.show()
```

Результат трехмерной визуализации:



3. Рекомендательные системы

Рекомендательные системы (Recommendation systems) — это инструменты, которые используются для предоставления персонализированных рекомендаций пользователям на основе их предпочтений, поведения и истории взаимодействия с системой.

Эти системы широко используются в различных областях: электронная коммерция, социальные сети, видео- и музыкальные сервисы, блоги и другие.

В контексте машинного обучения рекомендательные системы могут быть построены на основе различных методов, включая коллаборативную фильтрацию, контент-базированную фильтрацию, гибридные методы и другие.

Методы коллаборативной фильтрации

Методы коллаборативной фильтрации — это подходы к рекомендательным системам, которые используют историю взаимодействия пользователей с элементами, чтобы предсказать, что еще понравится пользователям в будущем.

Основная идея методов: если пользователи с похожими предпочтениями в прошлом выбирали определенный элемент, то есть вероятность, что в будущем этот элемент понравится и другим пользователям с похожими предпочтениями.

Существует два основных метода коллаборативной фильтрации: user-based и item-based.

- **User-based** — метод, который использует историю оценок пользователей для поиска других пользователей с похожими предпочтениями, чтобы рекомендовать первым те элементы, которые понравились вторым.

  Пример. Если пользователь A похож на пользователей B и C, которые любят фильмы жанра комедии, то система порекомендует пользователю A комедийные фильмы.

  Недостаток: **проблема холодного старта**, когда новый пользователь не имеет достаточно данных для поиска похожих пользователей.

- **Item-based** — метод, который использует историю оценок пользователей, чтобы найти и порекомендовать им элементы, похожие на те, что понравились им ранее.

  Пример. Если пользователю A понравилась книга B, то ему будут рекомендоваться другие книги, которые имеют похожее содержание или тематику.

  Недостаток: **проблема ограниченности**, когда элементы имеют мало оценок или низкий уровень взаимодействия с пользователями.

Методы с матричными разложениями

Рекомендательные системы, основанные на методах матричных разложений, являются одним из наиболее распространенных подходов к решению задачи рекомендации.

Эти системы используют матрицу оценок пользователей, чтобы предсказать какой рейтинг пользователь может дать тому или иному товару.

В этих системах матрица оценок может быть представлена в виде матрицы *R*, где строки соответствуют пользователям, а столбцы соответствуют товарам. Элемент *R*[*i*,*j*] соответствует оценке, которую пользователь *i* дал товару *j*. Однако, не все пользователи оценили все товары, поэтому матрица *R* обычно является разреженной.

Товар 1	**Товар 2**	**Товар 3**	**Товар 4**
Пользователь 1	5		3
Пользователь 2		2	3
Пользователь 3	4	3	
Пользователь 4		4	

Цель матричного разложения заключается в том, чтобы представить матрицу *R* в виде произведения двух более маленьких матриц *U* и *V*: $R = UV^T$. Здесь *U* и *V* являются матрицами с размерностью $K \times N$ и $K \times M$, соответственно, где *K* — это некоторое заданное число скрытых факторов, которое является параметром модели.

Эти скрытые факторы представляют собой абстрактные признаки, которые описывают как пользователей, так и товары.

Пусть имеется матрица (пользовательских признаков) *U*:

	Фактор 1	**Фактор 2**	**Фактор 3**
Пользователь 1	0.7	1.2	0.3
Пользователь 2	0.5	0.8	1.1
Пользователь 3	0.9	1.1	0.5
Пользователь 4	1.2	0.6	1.3

Матрица (товарных признаков) *V*:

	Фактор 1	**Фактор 2**	**Фактор 3**
Товар 1	0.9	1.5	0.2
Товар 2	0.7	1.0	1.1
Товар 3	0.8	0.9	0.6
Товар 4	1.2	0.6	1.3

Рассмотрим составление рекомендаций на примере матрицы рейтингов фильмов:

```
import numpy as np
from scipy.sparse.linalg import svds
```

Исходная матрица рейтингов:

```
ratings = np.array([[5, 3, 0, 1],
                    [4, 0, 0, 1],
                    [1, 1, 0, 5],
                    [1, 0, 0, 4],
                    [0, 1, 5, 4]]).astype(float)
```

Нули означают, что пользователь не посмотрел данный фильм. Наша задача — заполнить нули на основе матричной системы рекомендаций.

Выполняем SVD для матрицы рейтингов:

```
U, S, Vt = svds(ratings, k=2)
```

Собираем матрицу *S* в диагональную матрицу:

```
S_diag = np.diag(S)
```

Вычисляем предсказанные рейтинги:

```
predicted_ratings = np.dot(np.dot(U, S_diag), Vt)
```

Выводим исходную и предсказанную матрицу рейтингов:

```
print("Исходная матрица рейтингов:")
print(ratings)
print("\nПредсказанная матрица рейтингов:")
print(predicted_ratings)
```

Исходная матрица рейтингов:

```
[[5. 3. 0. 1.]
 [4. 0. 0. 1.]
 [1. 1. 0. 5.]
 [1. 0. 0. 4.]
 [0. 1. 5. 4.]]
```

Предсказанная матрица рейтингов:

```
[[ 5.13406479  1.90612125 -0.72165061  1.5611261 ]
 [ 3.43308995  1.28075331 -0.45629689  1.08967559]
 [ 1.54866643  1.0449763   1.78873709  3.96755551]
 [ 1.17598269  0.80359806  1.40136891  3.08786154]
 [-0.44866693  0.5443561   3.09799526  5.15263893]]
```

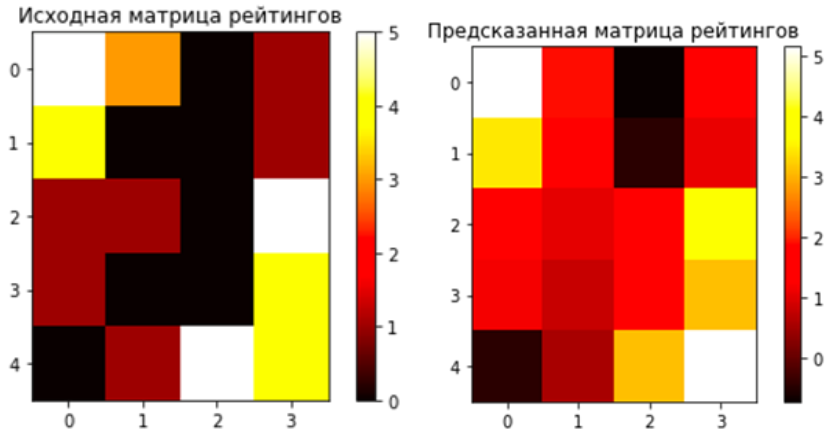
Отображаем исходную матрицу рейтингов:

```
import matplotlib.pyplot as plt
plt.imshow(ratings, cmap='hot', interpolation='nearest')
plt.title("Исходная матрица рейтингов")
plt.colorbar()
plt.show()
```

Отображаем предсказанную матрицу рейтингов:

```
plt.imshow(predicted_ratings, cmap='hot', interpolation='nearest')
plt.title("Предсказанная матрица рейтингов")
plt.colorbar()
plt.show()
```

Результат визуализации:



Дополнительные материалы для самостоятельного изучения

1. Кластеризуем лучше, чем «метод локтя» / Хабр (habr.com)
2. 2.3. Кластеризация - scikit-learn