Data science

Лекция 4. Метрики классификации. Кластеризация

2025/2026 учебный год

Доцент кафедры МО&МО, Махно В.В.

Кодовое слово для способа МАГИСТРЫ 2026 = **12022026**

Data Science и инструменты Анализа Данных

ИНСТИТУТ
МАТЕМАТИКИ
МЕХАНИКИ
КОМПЬЮТЕРНЫХ
НАУК
имени И.И. Воровича

Переход от линейной к логистической регрессии

Линейная регрессия предсказывает **любое число** (от −∞ до +∞).
Пример: цена, рост, доход.

Логистическая регрессия используется, когда нужно предсказывать **класс (0 или 1)**.
Пример: купит / не купит, спам / не спам.

Переход:

1) вычисляется линейная комбинация признаков:

$$z = w \cdot x + b$$

2) Z пропускается через **сигмоиду**, чтобы получить вероятность:

3) Если $P = \frac{1}{1+e^{-z}}$ **> 0.5** → класс 1, иначе — 0.

Измерение качества классификации

Базовая метрика — Accuracy (точность)

$$\text{Accuracy} = \frac{\text{Количество правильных предсказаний}}{\text{Общее количество примеров}}$$

Пример:

Модель предсказала верно 90 из 100 → Accuracy = 90%

Хорошо работает, если классы сбалансированы.

Плохо, если один класс сильно преобладает.

Матрица ошибок (confusion matrix)

	Факт: 1	Факт: 0
Предсказал: 1	TP	FP
Предсказал: 0	FN	TN

- **TP (True Positive)** — правильно предсказал класс 1
- **FP (False Positive)** — ошибочно предсказал 1, но это был 0
- **FN (False Negative)** — ошибочно предсказал 0, но это был 1
- **TN (True Negative)** — правильно предсказал 0

Precision — точность положительного класса

Вопрос: из всех, кого модель посчитала положительными, **сколько были действительно положительными?**

- Важно, если **ошибки FP критичны**

(например, диагностика болезни — нельзя ошибочно поставить диагноз)

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall — полнота

Вопрос: из всех реальных положительных случаев, **сколько модель нашла?**

- Важно, если **нельзя пропустить важное**

(например, обнаружение мошенничества, болезнь)

$$\text{Recall} = \frac{TP}{TP+FN}$$

Мера F1

F1-score — баланс между precision и recall

Удобно использовать, если классы несбалансированы и важно учитывать обе метрики.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Свод метрик классификации

Метрика	Что измеряет	Когда использовать
Accuracy	Доля правильных ответов	Когда классы сбалансированы
Precision	Сколько предсказанных 1 были верны	Когда FP опасны (ошибки "ложного да")
Recall	Сколько реальных 1 было найдено	Когда FN опасны (пропуски)
F1-score	Баланс точности и полноты	При дисбалансе классов

Продвинутые алгоритмы машинного обучения

Почему базовых моделей недостаточно?

Базовые модели (линейная, логистическая регрессия, KNN) хороши для старта.

Но **в реальных задачах**:

- данные часто **нелинейные**, сложные;
- признаки взаимодействуют между собой;
- много **пропусков**, **шумов**, **категориальных переменных**.

Значит, нужны более гибкие, устойчивые и точные модели.

Decision Tree — дерево решений

Модель **разбивает данные по признакам**, задавая простые вопросы.

```
Если возраст < 30:
    если доход > 50k → класс = 1
    иначе → класс = 0
иначе:
    → класс = 1
```

Плюсы:

- Интуитивно понятно, легко визуализировать
- Умеет работать с числовыми и категориальными признаками
- Не требует масштабирования

Минусы:

- Переобучается (особенно без ограничения глубины)
- Нестабильна: небольшие изменения в данных → совсем другое дерево

Где используется:

- Бизнес-правила
- Кредитные скоринги
- Прогнозы продаж

Random Forest — случайный лес

Это **множество деревьев решений**, обученных на случайных подвыборках.
Каждое дерево голосует, и побеждает **большинство**.

Плюсы:

- Очень устойчив к переобучению
- Работает хорошо «из коробки»
- Умеет определять **важность признаков**

Минусы:

- Менее интерпретируемый, чем одно дерево
- Медленнее при больших объёмах

Где используется:

- Банковский скоринг
- Прогнозирование спроса
- Табличные данные в бизнесе

Gradient Boosting (XGBoost, LightGBM, CatBoost)

Алгоритм **обучает деревья по очереди**, каждое исправляет ошибки предыдущего: строится первая модель, затем обучается вторая — чтобы она "улучшила" первую и т д.

Популярные реализации:

- **XGBoost** — классика, гибкая и точная
- **LightGBM** — очень быстрая и эффективная
- **CatBoost** — хорошо работает с категориальными данными

Плюсы:

- Высокая точность
- Гибкие настройки
- Хорошо работает на соревнованиях (например, Kaggle)

Минусы:

- Требует настройки (бустинг ≠ магия)
- Может переобучиться без регуляризации
- Медленнее, чем случайный лес

Где используется:

- Финансовое моделирование
- Маркетинг и рекомендации
- Предсказание оттока, конверсий, лояльности

KNN — k ближайших соседей

Модель **не учится**, а при предсказании **сравнивает объект с "похожими"**.

Находит k ближайших точек и **голосует**.

Плюсы:

- Очень прост в реализации
- Нет этапа обучения
- Интуитивно понятен

Минусы:

- **Медленный при большом датасете**
- Не работает хорошо в задачах с большим числом признаков
- Чувствителен к масштабу признаков

Где используется:

- Рекомендательные системы
- Сегментация
- Простейшие классификаторы

Нейросети (Neural Networks)

Модели, вдохновлённые работой мозга.
Состоят из **слоёв нейронов**, которые преобразуют вход в результат.

Плюсы:

- Могут находить **очень сложные зависимости**
- Работают с **текстом, изображениями, звуком**
- Подходят для задач "глубокого обучения"

Минусы:

- Сложные и требовательные
- Трудны в интерпретации
- Требуют большого объёма данных и вычислений

Где используются:

- Компьютерное зрение
- Обработка естественного языка (NLP)
- ChatGPT, голосовые помощники, генерация изображений

AutoML — автоматическое машинное обучение

Сервисы и библиотеки, которые **автоматизируют весь pipeline**:

- выбор модели
- подбор гиперпараметров
- предобработка данных

Плюсы:

- Быстро, удобно
- Хорошо подходит для прототипов
- Может использоваться даже без глубоких знаний ML

Минусы:

- Не всегда даёт лучшие результаты
- Трудно контролировать
- Чёрный ящик

Примеры:

- Google AutoML
- H2O AutoML
- AutoSklearn
- AutoGluon

Кластеризация

Кластеризация — это задача, в которой модель **сама находит группы (кластеры)** внутри данных. Группировка объектов без меток

В отличие от классификации:

- **Нет целевой переменной (y)**
- Модель работает **без учителя** (unsupervised learning)

Примеры

Ситуация	Что группируем?	Кластеры могут быть
E-commerce	Поведение клиентов	Новички / Активные / Спящие
Бизнес	Товары по продажам	Хиты / Сезонные / Неактивные
География	Координаты точек	Города, кластеры по регионам

Цель алгоритма

Разделить объекты **на группы**, чтобы:

- Внутри каждой группы элементы были **похожи друг на друга**
- Между группами была **разница**

K-Means

Принцип:

- Задать количество кластеров k
- Алгоритм находит k центров и относит каждую точку к ближайшему

Плюсы:

- Простой и быстрый
- Подходит для числовых данных

Минусы:

- Нужно заранее выбрать k
- Плохо работает при сложной форме кластеров

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(X)
labels = model.labels_
```

DBSCAN

Принцип

- Объединяет точки, находящиеся **достаточно близко друг к другу**
- Не требует заранее k

Плюсы:

- Ищет кластеры **произвольной формы**
- Умеет находить **выбросы (noise)**

Минусы:

- Требует настройки радиуса и минимального числа точек

Иерархическая кластеризация

Принцип

- Создаёт древовидную структуру из объектов
- Можно строить **дендрограмму**

Плюсы:

- Удобно для визуального анализа
- Не требует k на старте (можно обрезать дерево на нужном уровне)

Применение кластеризации

Область	Применение
Маркетинг	Сегментация клиентов
Ритейл	Группировка товаров
Аналитика	Поиск аномалий, схожих пользователей
Биология	Группировка генов, белков

Пример визуализации кластеров

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.scatterplot(x=X[:,0], y=X[:,1], hue=labels, palette='viridis')
plt.title("Результаты кластеризации")
plt.show()
```

Метрики кластеризации

Метрика	Что показывает
Silhouette score	Насколько хорошо объекты "вписываются" в свой кластер
Davies–Bouldin index	Чем меньше, тем лучше разделение
Inertia (для KMeans)	Суммарное расстояние до центров кластеров

```
from sklearn.metrics import silhouette_score
score = silhouette_score(X, labels)
print("Silhouette:", score)
```