Методы классификации в ML

Содержание

1. Логистическая регрессия
2. Деревья решений
3. Ансамблевые методы
4. Метод опорных векторов
5. Вероятностные методы
6. Методы на основе расстояний
7. Практическая реализация
8. Сравнительный анализ

Логистическая регрессия: Основы

Принцип работы: Моделирует вероятность принадлежности к классу через сигмоидную функцию.
Сигмоидная функция:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad z = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$$

Интерпретация коэффициентов:

- $\beta_i > 0$ — признак увеличивает вероятность класса 1
- $\beta_i < 0$ — признак уменьшает вероятность класса 1
- $|\beta_i|$ — сила влияния признака

Применение:

- Медицинская диагностика (вероятность заболевания)
- Кредитный скоринг (риск дефолта)
- Маркетинг (отклик на рекламу)

Логистическая регрессия: Математика и оптимизация

Функция потерь (бинарная кроссэнтропия):

$$J(\beta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\beta(x^{(i)})) + (1-y^{(i)})\log(1-h_\beta(x^{(i)}))]$$

Градиентный спуск:

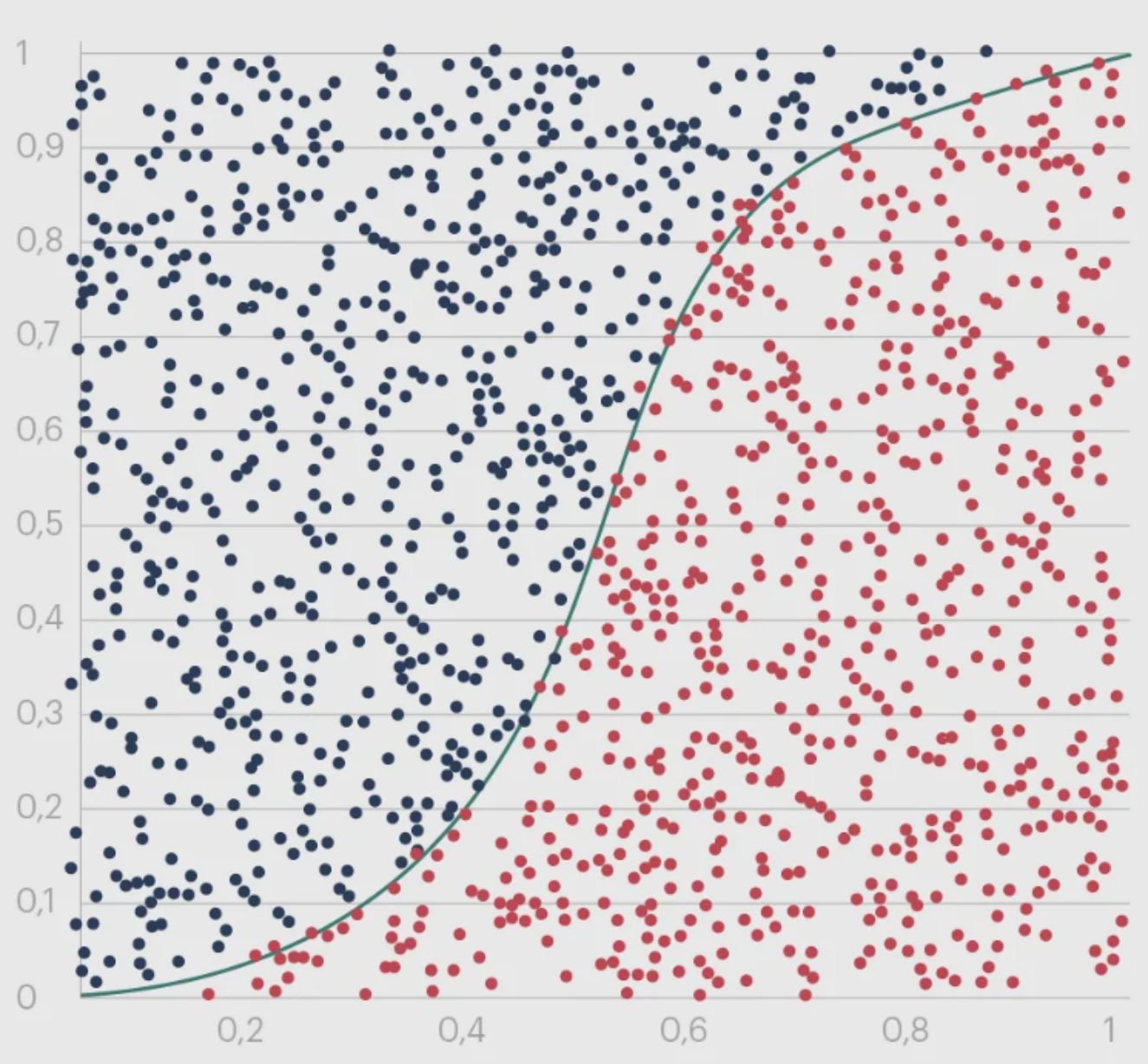
$$\beta_j := \beta_j - \alpha\frac{\partial J}{\partial \beta_j}$$

Регуляризация:

- L1 (Lasso): $J(\beta) + \lambda\sum_{j=1}^{n}|\beta_j|$ — отбор признаков
- L2 (Ridge): $J(\beta) + \lambda\sum_{j=1}^{n}\beta_j^2$ — сглаживание весов

Преимущества: Быстрое обучение, вероятностные предсказания, хорошая интерпретируемость
Недостатки: Только линейные границы решений, чувствительность к выбросам

Логистическая регрессия: Визуализация



Деревья решений: Принцип построения

Структура дерева:

- Корневой узел: содержит все данные
- Внутренние узлы: условия разделения (например, $x_i <$ порог)
- Листья: предсказываемые классы

Алгоритм построения (жадный подход):

1. Выбрать лучший признак и порог для разделения
2. Разделить данные на две части
3. Рекурсивно повторить для каждой части
4. Остановиться при достижении критерия останова

Критерии останова:

- Максимальная глубина дерева
- Минимальное количество образцов в узле
- Минимальное улучшение качества

Деревья решений: Критерии разделения

Индекс Джини (Gini Impurity):

$$\text{Gini} = 1 - \sum_{i=1}^{C} (p_i)^2$$

Где p_i — доля класса i в узле. Значения: 0 (чистый узел) до 0.5 (максимальная нечистота для 2 классов)
Энтропия:

$$\text{Entropy}(S) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

Прирост информации:

$$\text{IG}(S, A) = \text{Entropy}(S) - \sum_{v} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Практические рекомендации:

- Gini — быстрее вычисляется, хорошо для большинства задач
- Энтропия — более строгая мера, лучше для сбалансированных данных

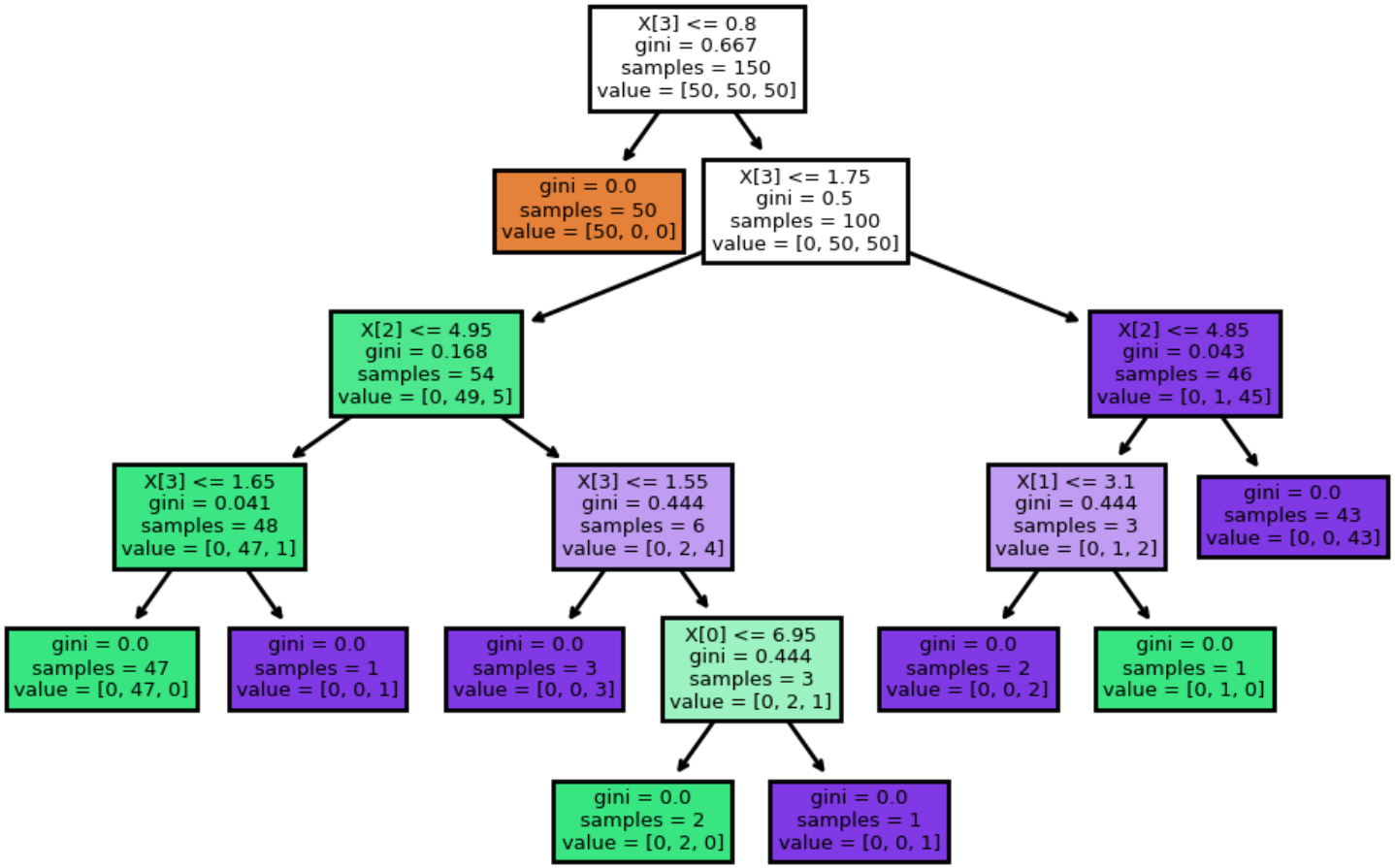
Деревья решений: Переобучение и методы борьбы

Проблема переобучения: Дерево может запомнить обучающие данные, создав очень сложные правила
Методы предотвращения:

- Предварительная обрезка (Pre-pruning):
  - max_depth: ограничение глубины
  - min_samples_split: минимум образцов для разделения
  - min_samples_leaf: минимум образцов в листе
- Последующая обрезка (Post-pruning):
  - Построить полное дерево, затем удалить ненужные ветви
  - Использовать валидационную выборку для оценки

Преимущества: Интерпретируемость, обработка категориальных признаков, не требует масштабирования
Недостатки: Переобучение, нестабильность к изменениям данных, биас к признакам с большим количеством значений

Деревья решений: Визуализация



Случайный лес: Принципы ансамблирования

Теоретическое обоснование: Если отдельные модели независимы и имеют точность > 0.5, то ансамбль всегда лучше
Снижение дисперсии: $\mathrm{Var}(\bar{X}) = \frac{\sigma^2}{k}$ (при независимых оценках)
Алгоритм Random Forest:

1. Bootstrap: Создать k подвыборок размера N с возвращением
2. Random Subspace: В каждом узле выбирать $m = \sqrt{p}$ случайных признаков
3. Обучение: Построить k деревьев на подвыборках
4. Агрегация: Мажоритарное голосование (классификация) или усреднение (регрессия)

Out-of-Bag (OOB) оценка: Для каждого образца 37% деревьев его не видели — можно использовать как валидацию

Случайный лес: Гиперпараметры и важность признаков

Ключевые гиперпараметры:

- n_estimators: количество деревьев (100-1000)
- max_features: количество признаков для разделения ($\sqrt{p}$, $\log_2(p)$)
- max_depth: глубина деревьев (обычно не ограничивают)
- min_samples_split, min_samples_leaf: контроль переобучения

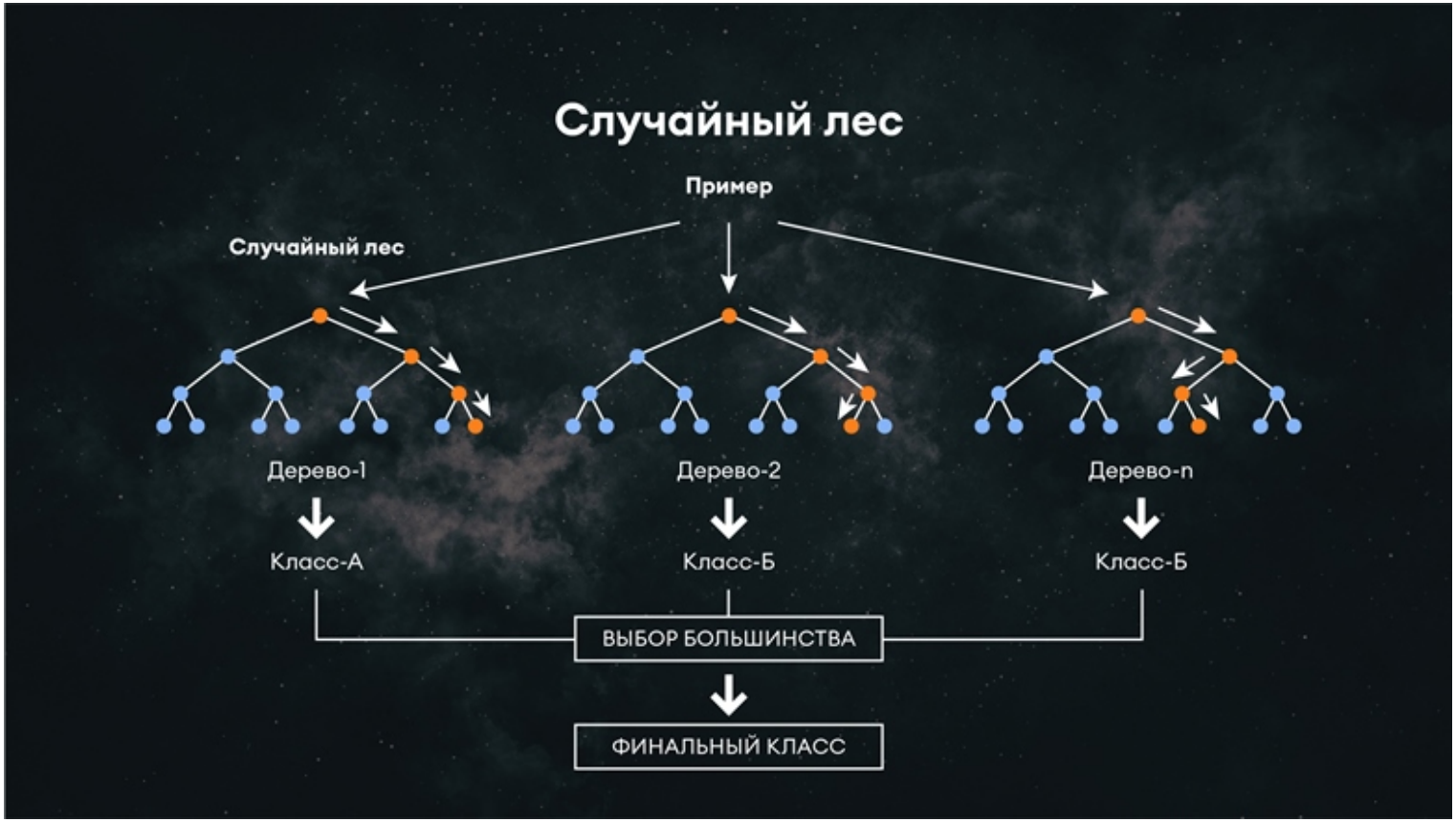
Важность признаков (Feature Importance):

$$FI_j = \frac{1}{k} \sum_{t=1}^{k} \sum_{i \in \text{узлы дерева } t} w_i \cdot \Delta_i \cdot I(v_i = j)$$

Где w_i — доля образцов в узле, Δ_i — уменьшение нечистоты
Применение:

- Задачи общего назначения (tabular data)
- Когда нужна робастность к выбросам
- Анализ важности признаков

Случайный лес: Визуализация



SVM: Геометрическая интуиция

Основная идея: Найти гиперплоскость, которая максимально разделяет классы

Оптимизационная задача (линейный случай):

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w \cdot x_i + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

Геометрический смысл:

- Отступ (margin): $\gamma = \frac{2}{\|w\|}$
- Опорные векторы: точки на границе отступа
- Решение зависит только от опорных векторов

Soft Margin (мягкий отступ):

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned}$$

SVM: Ядра и нелинейная классификация

Ядерный трюк (Kernel Trick): Отображение в высокоразмерное пространство без явного вычисления координат: $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$
Популярные ядра:

- Линейное: $K(x_i, x_j) = x_i \cdot x_j$
- RBF (Гауссово): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
- Полиномиальное: $K(x_i, x_j) = (\gamma x_i \cdot x_j + r)^d$
- Сигмоидное: $K(x_i, x_j) = \tanh(\gamma x_i \cdot x_j + r)$

Выбор ядра:

- Линейное — для высокоразмерных данных (текст)
- RBF — универсальный выбор для большинства задач
- Полиномиальное — для задач с явными полиномиальными зависимостями

Гиперпараметры: C (регуляризация), γ (ширина RBF), степень d (для полинома)

SVM: Практические аспекты

Масштабирование признаков — критично! SVM чувствителен к масштабу, так как использует расстояния. Обязательно применять StandardScaler или MinMaxScaler.

Выбор гиперпараметров:

- C (регуляризация): малое C $\rightarrow$ простая модель, большое C $\rightarrow$ сложная модель
- γ (для RBF): малое $\gamma \rightarrow$ гладкие границы, большое $\gamma \rightarrow$ сложные границы

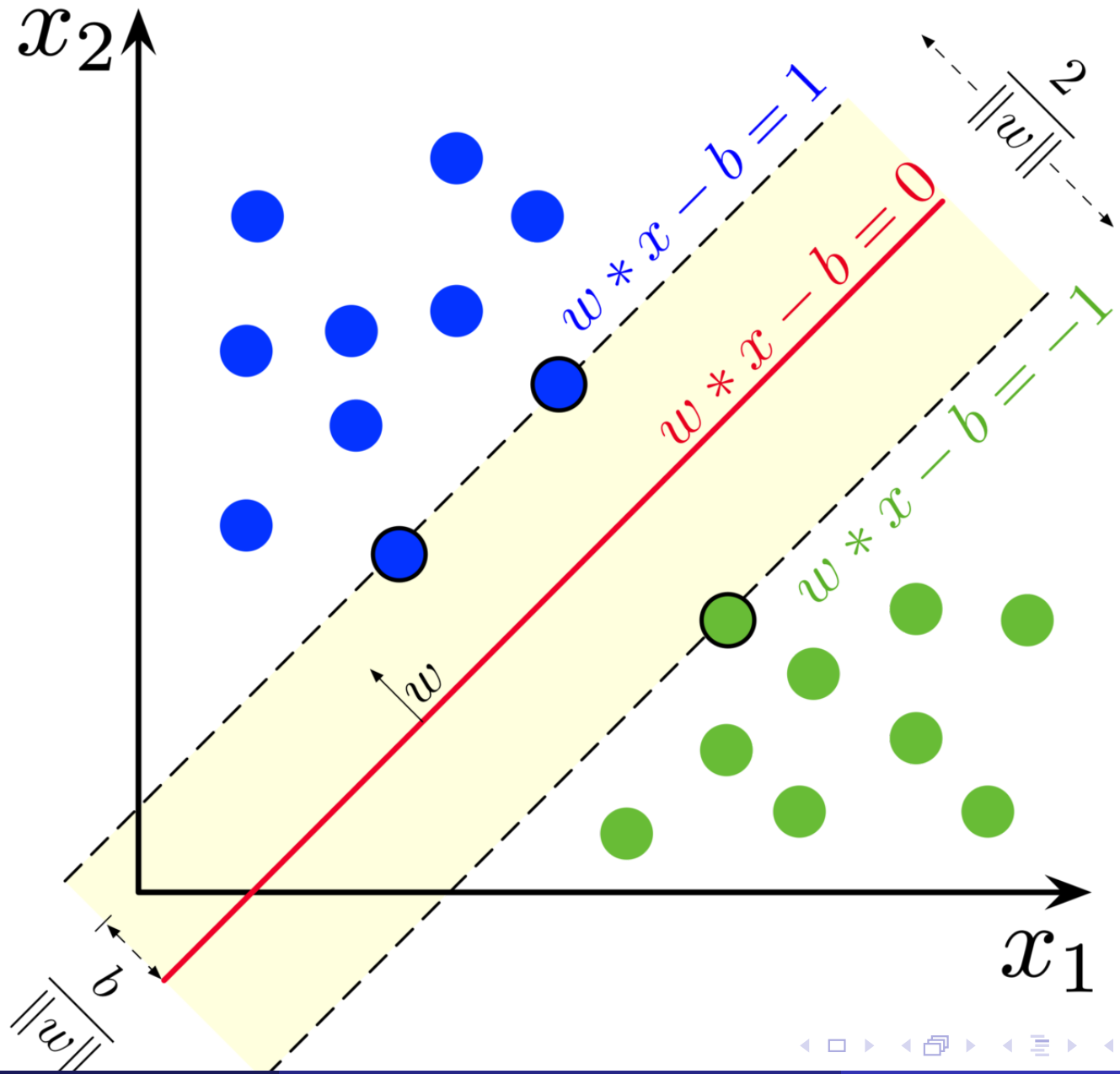
Временная сложность:

- Обучение: $O(n^2)$ до $O(n^3)$ в зависимости от алгоритма
- Предсказание: $O(k \cdot d)$, где k — количество опорных векторов

Применение:

- Классификация текстов (высокоразмерные данные)
- Биоинформатика (малые выборки)
- Компьютерное зрение (с соответствующими ядрами)

SVM: Визуализация



Наивный Байес: Теорема Байеса

Теорема Байеса — основа классификатора:

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)}$$

Компоненты:

- $P(C_k|x)$ — апостериорная вероятность (что мы хотим найти)
- $P(C_k)$ — априорная вероятность класса (доля класса в данных)
- $P(x|C_k)$ — правдоподобие (вероятность признаков при данном классе)
- $P(x)$ — маргинальная вероятность (нормализующая константа)

Наивное предположение о независимости:

$$P(x|C_k) = \prod_{i=1}^{n} P(x_i|C_k)$$

Хотя предположение часто нарушается, метод работает удивительно хорошо!

Наивный Байес: Варианты и применения

Решающее правило (MAP — Maximum A Posteriori):

$$\hat{y} = \arg\max_{k} P(C_k) \prod_{i=1}^{n} P(x_i|C_k)$$

Варианты классификатора:

- GaussianNB: для непрерывных признаков (предполагает нормальное распределение)

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}} \exp\left(-\frac{(x_i - \mu_{ik})^2}{2\sigma_{ik}^2}\right)$$

- MultinomialNB: для счетчиков (например, частота слов в тексте)
- BernoulliNB: для бинарных признаков (присутствие/отсутствие)

Сглаживание Лапласа: Добавляет псевдосчетчики для избежания нулевых вероятностей: $P(x_i|C_k) = \frac{\text{count}(x_i, C_k) + \alpha}{\text{count}(C_k) + \alpha \cdot n}$

Наивный Байес: Преимущества и применения

Преимущества:

- Очень быстрое обучение и предсказание $O(n \cdot d)$
- Хорошо работает с малыми данными
- Естественно обрабатывает многоклассовую классификацию
- Дает вероятностные оценки
- Не чувствителен к нерелевантным признакам
- Хорошо работает с высокоразмерными данными

Недостатки:

- Предположение о независимости часто нарушается
- Может давать плохие вероятностные оценки (хотя классификация правильная)
- Категориальные признаки должны быть представлены как счетчики

Классические применения:

- Фильтрация спама
- Классификация документов по темам
- Анализ тональности текста
- Медицинская диагностика

k-NN: Ленивое обучение

Принцип "ленивого" обучения:

- Не строит модель во время обучения
- Сохраняет все обучающие данные
- Все вычисления происходят во время предсказания

Алгоритм классификации:

1. Вычислить расстояния от новой точки до всех обучающих
2. Найти k ближайших соседей
3. Классифицировать по мажоритарному голосованию
4. (Опционально) взвесить голоса по обратному расстоянию

Взвешенное голосование:

$$\text{вес} = \frac{1}{\text{расстояние} + \epsilon}$$

Ближайшие соседи имеют больший вес в решении.

k-NN: Метрики расстояния

Метрики расстояния:

- Евклидово: $d(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$
- Манхэттен: $d(p, q) = \sum_{i=1}^{n} |q_i - p_i|$
- Минковского: $d(p, q) = \left(\sum_{i=1}^{n} |q_i - p_i|^r\right)^{1/r}$
- Хэмминга: для категориальных данных
- Косинусное: $1 - \frac{p \cdot q}{\|p\|\|q\|}$

Выбор метрики:

- Евклидово — для непрерывных признаков одного типа
- Манхэттен — более устойчиво к выбросам
- Косинусное — для текстовых данных и высокоразмерных векторов

Проклятие размерности: В высоких размерностях все точки становятся примерно одинаково далекими

k-NN: Оптимизация и практические аспекты

Выбор k:

- Малое k $\rightarrow$ высокая чувствительность к шуму
- Большое k $\rightarrow$ размытие границ между классами
- Нечетное k для избежания ничьих
- Типичные значения: $k = \sqrt{n}$ или подбор через кросс-валидацию

Оптимизация поиска соседей:

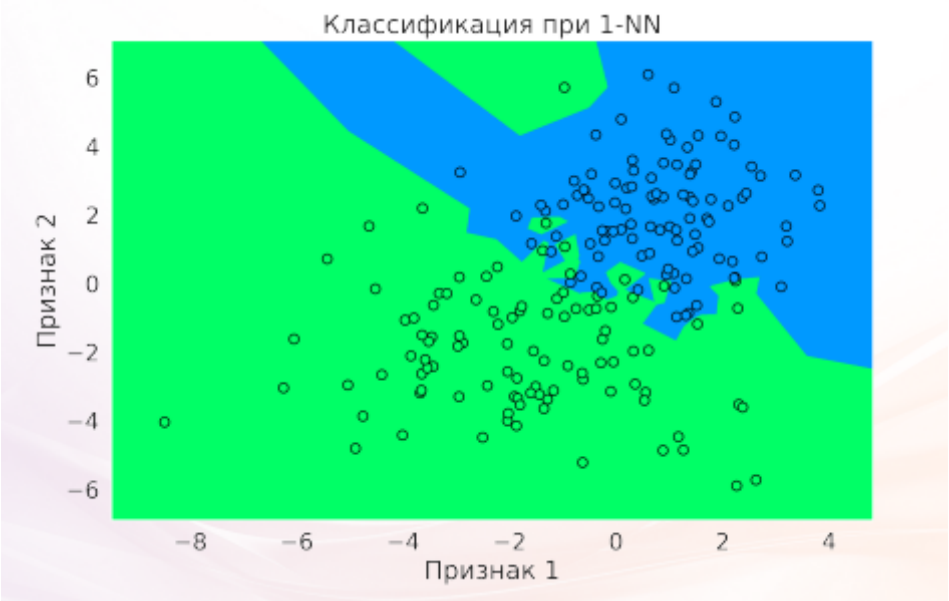
- KD-Tree: эффективно для низких размерностей (< 20)
- Ball Tree: работает в высоких размерностях
- LSH (Locality Sensitive Hashing): для приближенного поиска

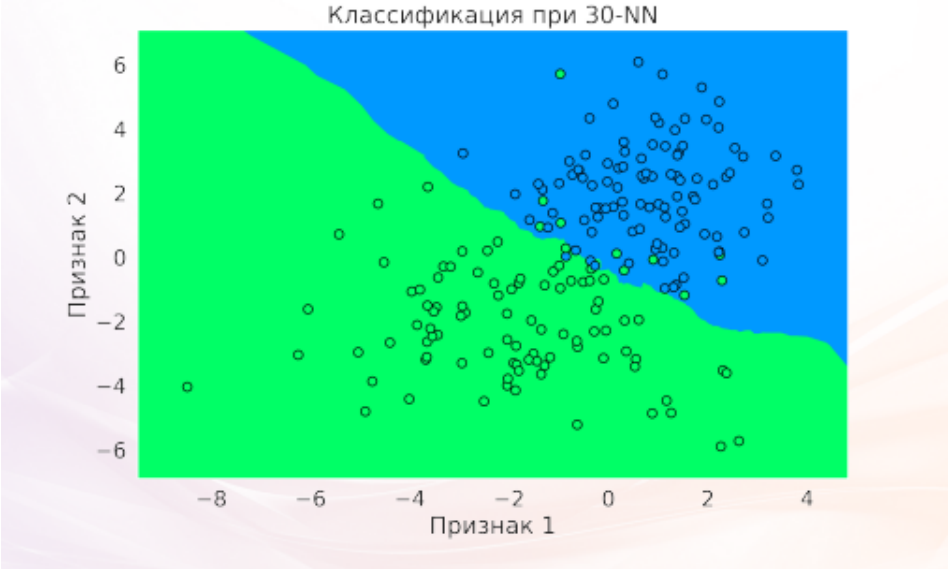
Временная сложность:

- Обучение: $O(1)$ (просто сохранение данных)
- Предсказание: $O(n \cdot d)$ (наивно) или $O(\log n \cdot d)$ (с деревьями)

Важно: Обязательное масштабирование признаков!

k-NN: Визуализация





Общая настройка среды

Установка библиотек:

pip install scikit-learn pandas numpy matplotlib seaborn

Стандартные импорты:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Типичный пайплайн предобработки:

```
# Разделение данных
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)


# Масштабирование (для SVM, k-NN, логистической регрессии)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Сравнение всех методов

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

# Словарь моделей
models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(random_state=42, max_depth=10),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'SVM (RBF)': SVC(kernel='rbf', random_state=42),
    'SVM (Linear)': SVC(kernel='linear', random_state=42),
    'Naive Bayes': GaussianNB(),
    'k-NN': KNeighborsClassifier(n_neighbors=5)
}

# Обучение и оценка с кросс-валидацией
results = {}
for name, model in models.items():
    if name in ['SVM (RBF)', 'SVM (Linear)', 'k-NN', 'Logistic Regression']:
        scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='accuracy')
    else:
        scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    results[name] = {'mean': scores.mean(), 'std': scores.std()}
    print(f"{name}: {scores.mean():.3f} (+/- {scores.std()*2:.3f})")
```

Сравнительная таблица методов

Метод	Интерпретируемость	Нелинейность	Скорость	Лучшее применение
Логистическая регрессия	Очень высокая	Нет	Очень быстро	Базовая классификация, медицина
Деревья решений	Очень высокая	Да	Быстро	Бизнес-правила, объяснимый AI
Случайный лес	Средняя	Да	Средне	Табличные данные, feature importance
SVM (линейный)	Средняя	Нет	Быстро	Текстовая классификация
SVM (RBF)	Низкая	Да	Медленно	Малые выборки, сложные границы
Наивный Байес	Высокая	Нет	Очень быстро	Текст, категориальные данные
k-NN	Средняя	Да	Медленно	Рекомендации, поиск похожих

Размер данных:

- Малые (< 1K): Наивный Байес, SVM, k-NN
- Средние (1K-100K): Все методы
- Большие (> 100K): Логистическая регрессия, Наивный Байес, Линейный SVM

Объяснение ключевых характеристик

Интерпретируемость:

- Очень высокая: можно легко объяснить каждое решение (логистическая регрессия, деревья)
- Высокая: общий принцип понятен (наивный Байес)
- Средняя: можно получить важность признаков (случайный лес, k-NN)
- Низкая: модель работает как "черный ящик" (SVM с нелинейными ядрами)

Нелинейность: Способность модели изучать сложные, нелинейные зависимости между признаками

Скорость:

- Очень быстро: $O(n \cdot d)$ или близко к этому
- Быстро: $O(n \cdot d \cdot \log n)$
- Средне: $O(n^{1.5} \cdot d)$
- Медленно: $O(n^2 \cdot d)$ или хуже

Практические рекомендации по выбору

Выбор по типу задачи:

- Нужна интерпретируемость: Логистическая регрессия, Деревья решений
- Табличные данные: Случайный лес (универсальный выбор)
- Текстовые данные: Наивный Байес, Линейный SVM
- Изображения: SVM с RBF, k-NN с подходящей метрикой
- Малая выборка: Наивный Байес, SVM
- Большая выборка: Логистическая регрессия, Линейный SVM
- Много шума: Случайный лес (устойчив к выбросам)
- Реальное время: Наивный Байес, Логистическая регрессия

Общий совет:

1. Начните с простого (логистическая регрессия)
2. Попробуйте случайный лес как baseline
3. Используйте кросс-валидацию для сравнения
4. Учитывайте требования к скорости и интерпретируемости