# Базовые методы обучения моделей

Паточенко Евгений

НИУ ВШЭ

# План занятия

- Регуляризация в линейной регрессии
- Кросс-валидация
- Кодирование категориальных признаков

# Напоминание

*Линейная регрессия*

Базовая модель машинного обучения и статистики, используемая для оценки зависимости между одной зависимой переменной (целевой) и одной или несколькими независимыми переменными (признаками), при этом целевая переменная — непрерывная величина

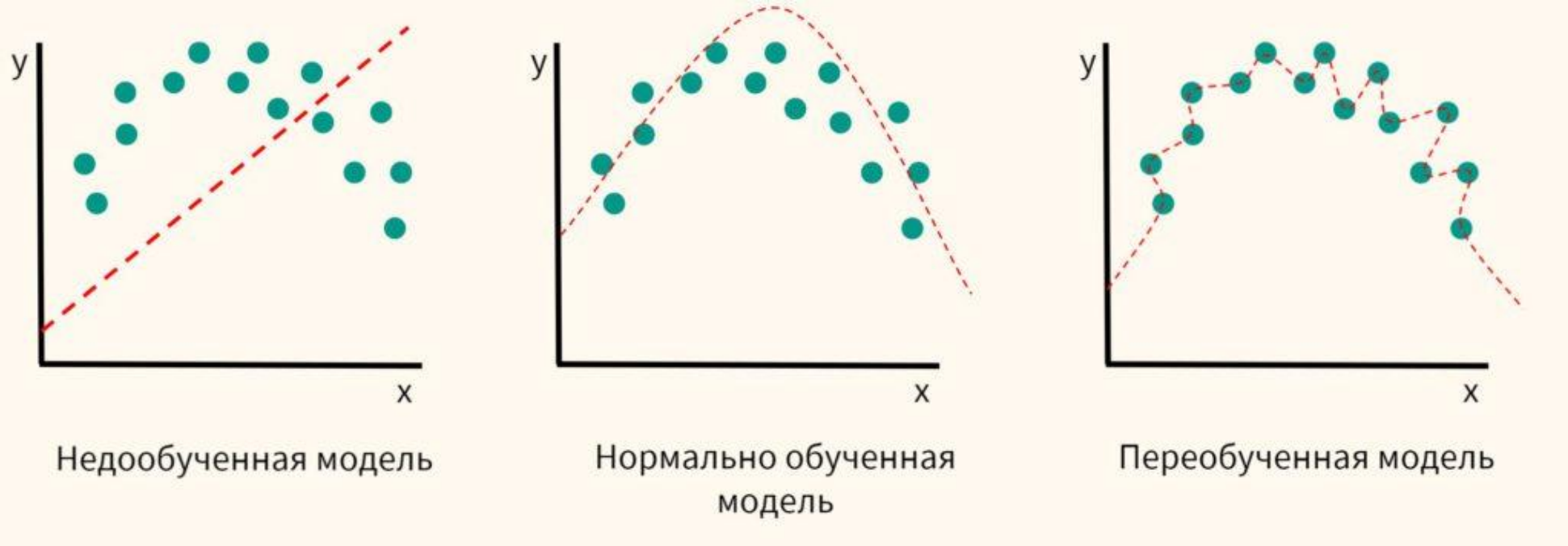
**Общий вид уравнения линейной регрессии**

$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \varepsilon$, где

- $y$ — зависимая (целевая) переменная
- $x_1, x_2, \ldots x_n$ — независимые переменные (признаки)
- $\beta_0$ — свободный член (intercept)
- $\beta_1, \beta_2, \ldots \beta_n$ — коэффициенты (веса) модели
- $\varepsilon$ — ошибка модели (ошибка прогноза)

# Напоминание

Переобучение (overfitting) — явление, при котором качество модели на новых данных сильно хуже, чем на обучающей выборке



# Напоминание

*Описание датасета*

Прежде всего необходимо понять как можно больше про данные, с которыми будем работать:

- Посмотреть объем и полноту
- Проверить характеристики данных
- Определить распределение
- Узнать есть ли зависимость между признаками и признаков с целевой переменной

# Напоминание

*Описание датасета*

Прежде всего необходимо понять как можно больше про данные, с которыми будем работать:

- Посмотреть объем и полноту
- Проверить характеристики данных
- Определить распределение
- Узнать есть ли зависимость между признаками и признаков с целевой переменной

# Регуляризация

*Зачем нужна?*

- При сильной корреляции признаков веса при них теряют физический смысл, а задача оптимизации $Q(\beta) \to min$ может иметь бесконечное число решений
- Возможна парадоксальная ситуация: признак, который должен повышать целевую переменную, получает отрицательный вес
- Это делает модель неточной и неинтерпретируемой
- Неадекватность знаков или величины весов — верный признак **мультиколлинеарности** (сильной линейной зависимости между признаками)

# Регуляризация

*Зачем нужна?*

При регуляризации мы модифицируем функцию потерь, добавляя штраф за слишком большие веса

# Регуляризация

*Зачем нужна?*

При регуляризации мы модифицируем функцию потерь, добавляя штраф за слишком большие веса

В общем виде это можно записать так:

$$Q_\lambda(\beta) = Q(\beta) + \lambda R(\beta),$$

где λ — коэффициент регуляризации (гиперпараметр, который отражает степень силы штрафа), а $R(\beta)$ — регуляризатор

# Регуляризация

*Зачем нужна?*

При регуляризации мы модифицируем функцию потерь, добавляя штраф за слишком большие веса

В общем виде это можно записать так:

$$Q_\lambda(\beta) = Q(\beta) + \lambda R(\beta),$$

где λ — коэффициент регуляризации (гиперпараметр, который отражает степень силы штрафа), а $R(\beta)$ — регуляризатор

λ подбирают на валидационной выборке по логарифмической шкале (например, от 1e-2 до 1e+2)

# Регуляризация

*L2-регуляризация (Ridge)*

$$R(\beta) = \|\beta\|_2 = \sum_{i=1}^{n} \beta_i^2$$

$$Loss = MSE + \lambda \sum \beta_i^2$$

*L1-регуляризация (Lasso)*

$$R(\beta) = \|\beta\|_1 = \sum_{i=1}^{n} |\beta_i|$$

$$Loss = MSE + \lambda \sum |\beta_i|$$

# Регуляризация

*L2-регуляризация (Ridge)*

Использует сумму квадратов весов, то есть L2-норму

Штрафует большие значения весов, заставляя их приближаться к нолю, но не зануляет их полностью. Уменьшает влияние шума в данных на модель

В библиотеке scikit-learn реализована в виде класса Ridge (ссылка на документацию)

# Регуляризация

*L1-регуляризация (Lasso)*

Использует сумму модулей весов, то есть L1-норму

Штрафует все веса одинаково, тем самым зануляя наименее важные. Успешно используется для отбора признаков

В библиотеке scikit-learn реализована в виде класса Lasso (ссылка на документацию)

# Регуляризация

*ElasticNet*

Комбинированный подход к регуляризации, сочетающий механизмы Lasso и Ridge

$$R(\beta) = \|\beta\|_1 + \|\beta\|_2 = \sum_{i=1}^{n} |\beta_i| + \sum_{i=1}^{n} \beta_i^2$$

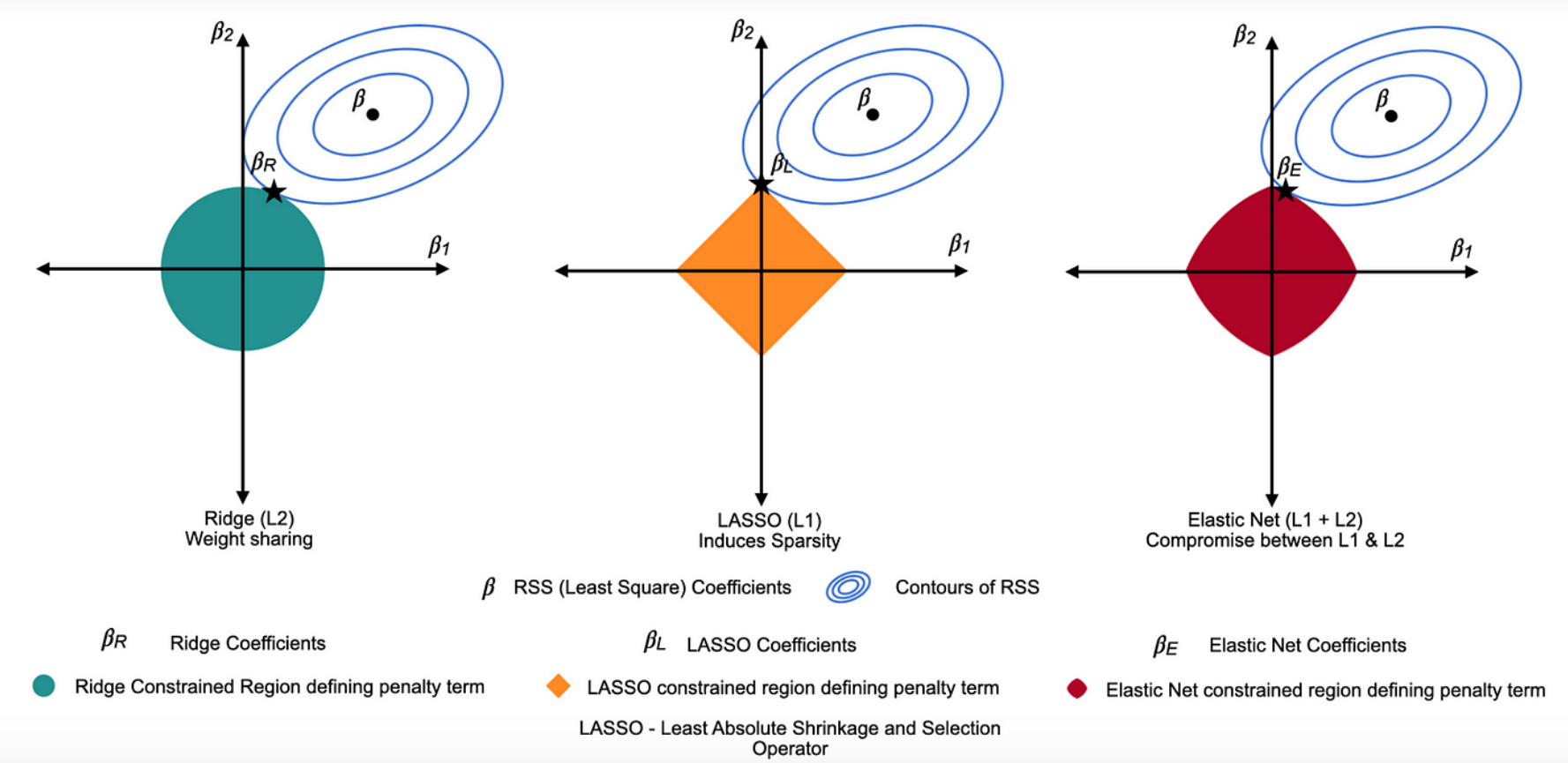
$$Loss = MSE + \lambda_1 \sum |\beta_i| + \lambda_2 \sum \beta_i^2$$

Совмещает плюсы L1 и L2: отбирает признаки и сглаживает веса

В библиотеке scikit-learn реализована в виде класса ElasticNet
(ссылка на документацию)

# Регуляризация

*Сравнение методов регуляризации*



# Регуляризация

*Плюсы и минусы регуляризации*

\+ Повышает устойчивость весов в случае мультиколлинеарности

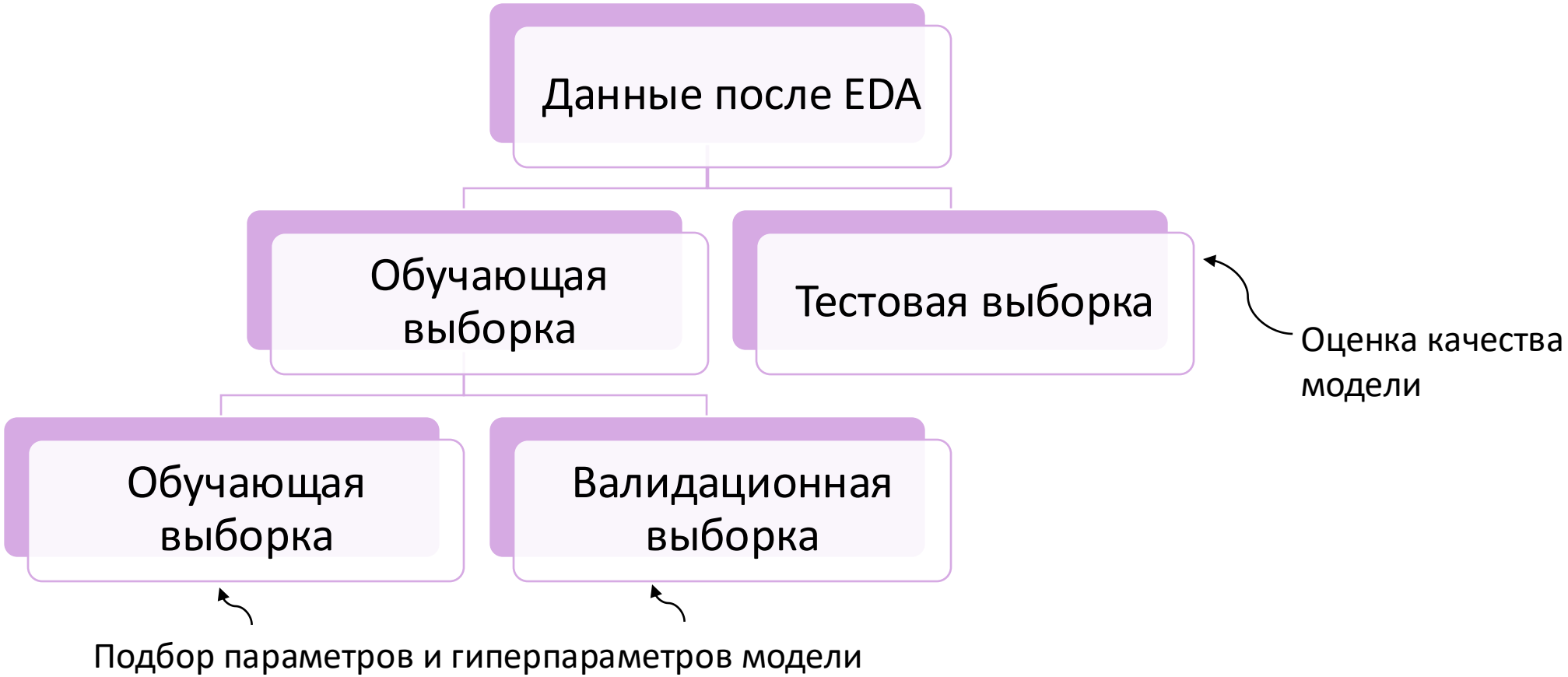
\+ Повышает обобщающую способность алгоритма

\+ Снижает риск переобучения

\- Высокие вычислительные затраты на кросс-валидацию для подбора силы регуляризации $\lambda$

# Напоминание

*Разбиение данных на обучающую и тестовую выборку*



# Кросс-валидация

*Определение*

**Кросс-валидация** — это процедура для оценки качества работы модели

Наиболее часто используемый механизм — **k-Fold CV**: фиксируется тестовая выборка, а обучающая — разделяется на $k$ равных частей (фолдов)

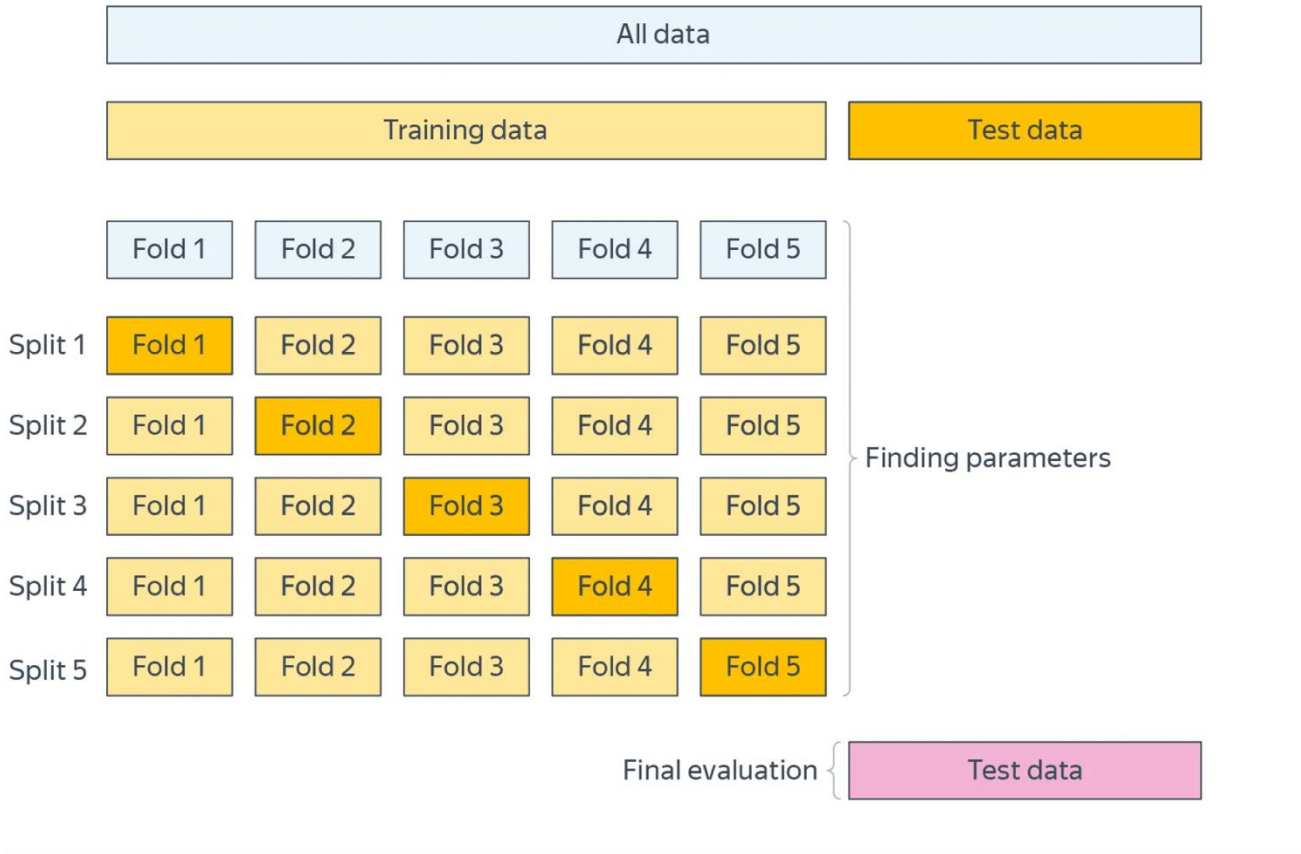
На каждом шаге одна часть используется как тестовая, остальные $k-1$ — для обучения

Процесс повторяется $k$ раз с последовательной сменой валидационной части

Итоговое качество модели — среднее значение метрики по всем фолдам, либо по отложенной выборке

# Кросс-валидация

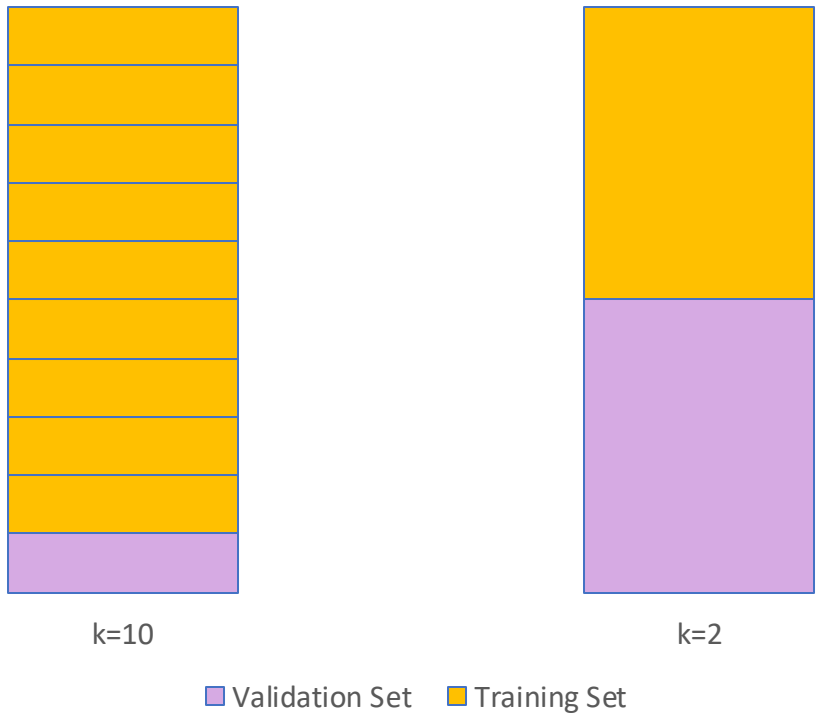
*k-Fold CV*




Рекомендуем ознакомиться с материалом о кросс-валидации и ее реализации в [scikit-learn]

# Кросс-валидация

*Выбор количества блоков в k-Fold CV*

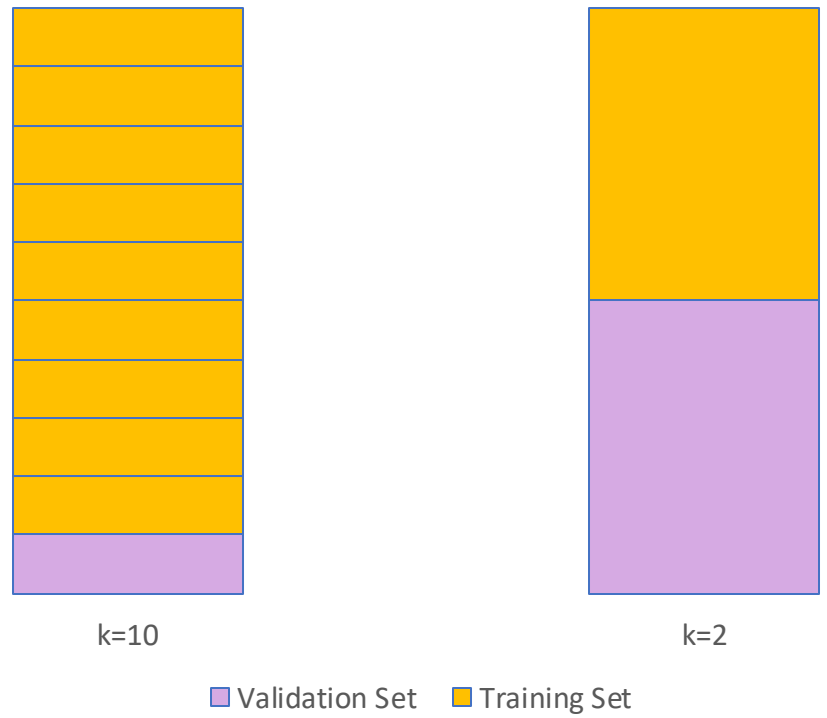



Проблемы при маленьком k?

Проблемы при большом k?

# Кросс-валидация

*Выбор количества блоков в k-Fold CV*




При маленьком k оценка может быть пессимистично занижена из-за маленького размера тренировочной части

При большом k оценка может иметь большую дисперсию из-за маленького размера валидационной части и долго обучать много моделей

# Кросс-валидация

*Stratified k-Fold*

Для разбиения на фолды используется стратификация, то есть каждый фолд содержит примерно такое же соотношение классов, как и все исходное множество.

Полезен для использования с датасетами с дисбалансом классов, когда при обычном случайном разбиении (random split) некоторые фолды могут содержать слишком мало экземпляров некоторых классов или не содержать их вовсе.

Реализация в scikit-learn

# Кросс-валидация

*Перебор гиперпараметров*

- **Grid Search** — перебор по сетке
    - Для каждого гиперпараметра фиксируется несколько значений
    - Последовательно перебираются все возможные комбинации значений гиперпараметров, на каждой из комбинаций обучается и тестируется модель
    - Выбирается модель с наилучшим качеством

Минус метода: если комбинаций значений слишком много, подбор будет длиться неразумно долго

Реализация в scikit-learn

# Кросс-валидация

*Перебор гиперпараметров*

- **Random Search**
  - Для каждого гиперпараметра задается распределение, из которого семплированием выбираются значения
  - Последовательно перебираются отобранные случайные комбинации значений, на каждой из комбинаций обучается и тестируется модель
  - Выбирается модель с наилучшим качеством

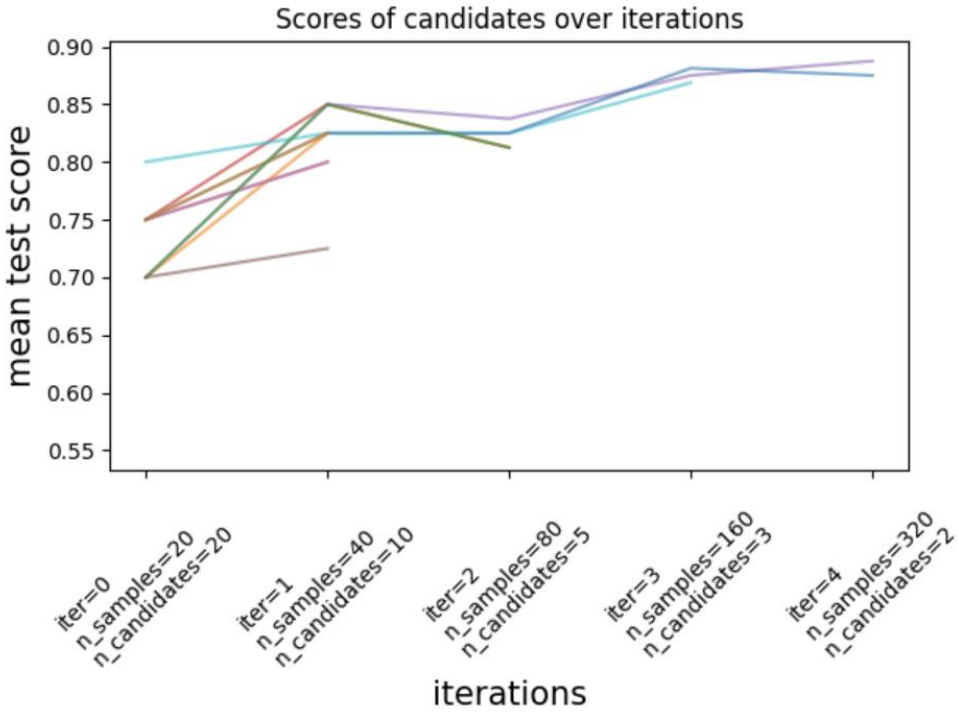
Преимущество: при достаточно большом количестве перебираемых значений RS может за то же число итераций, что GS, рассмотреть более разнообразные значения. Таким образом, мы оптимизируем поиск без риска пропустить хорошую комбинацию

Реализация в scikit-learn

# Кросс-валидация

*Перебор гиперпараметров*

- **Последовательный халвинг (Successive halving, SH) —** все комбинации параметров оцениваются с небольшим количеством ресурсов на первой итерации. На вторую итерацию с увеличенными ресурсами проходят только те комбинации, которые показали лучшие результаты. Процесс повторяется до нахождения наилучшей комбинации




Реализация в [scikit-learn по полной сетке]
и [с семплированием]

# Кросс-валидация

*Другие методы*

- **Байесовская оптимизация**
- **Tree-structured Parzen Estimator (TPE)**
- **Population Based Training (PBT)**
- **Hyperband**
- **и т. д.**

В простых случаях используются Grid Search и Random Search

# Кросс-валидация

*Основные библиотеки для подбора гиперпараметров*

- **Scikit-learn**
- **Hyperopt**
- **Optuna**

# Напоминание

*Типы признаков*

— Бинарные
— Категориальные
— Непрерывные
— Географические данные
— Дата и время
— и многие другие

# Напоминание

*Типы признаков*

— Бинарные
— Категориальные
— Непрерывные
— Географические данные
— Дата и время
— и многие другие

# Кодирование категориальных признаков

*Label Encoding*

Заменяет каждую категорию на уникальное целое число

| Color | Size | Price |
|---|---|---|
| Blue | L | 100 |
| Green | M | 150 |
| Red | S | 200 |
| Green | XL | 120 |
| Red | M | 180 |

Label Encoding

| Color | Size | Price |
|---|---|---|
| 0 | 0 | 100 |
| 1 | 1 | 150 |
| 2 | 2 | 200 |
| 1 | 3 | 120 |
| 2 | 1 | 180 |

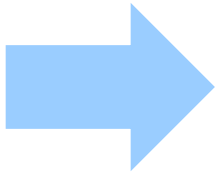
# Кодирование категориальных признаков

*One-Hot Encoding (OHE)*

Создает новую бинарную колонку для каждой категории

| id | color |
|---|---|
| 1 | red |
| 2 | blue |
| 3 | green |
| 4 | blue |

One-Hot Encoding

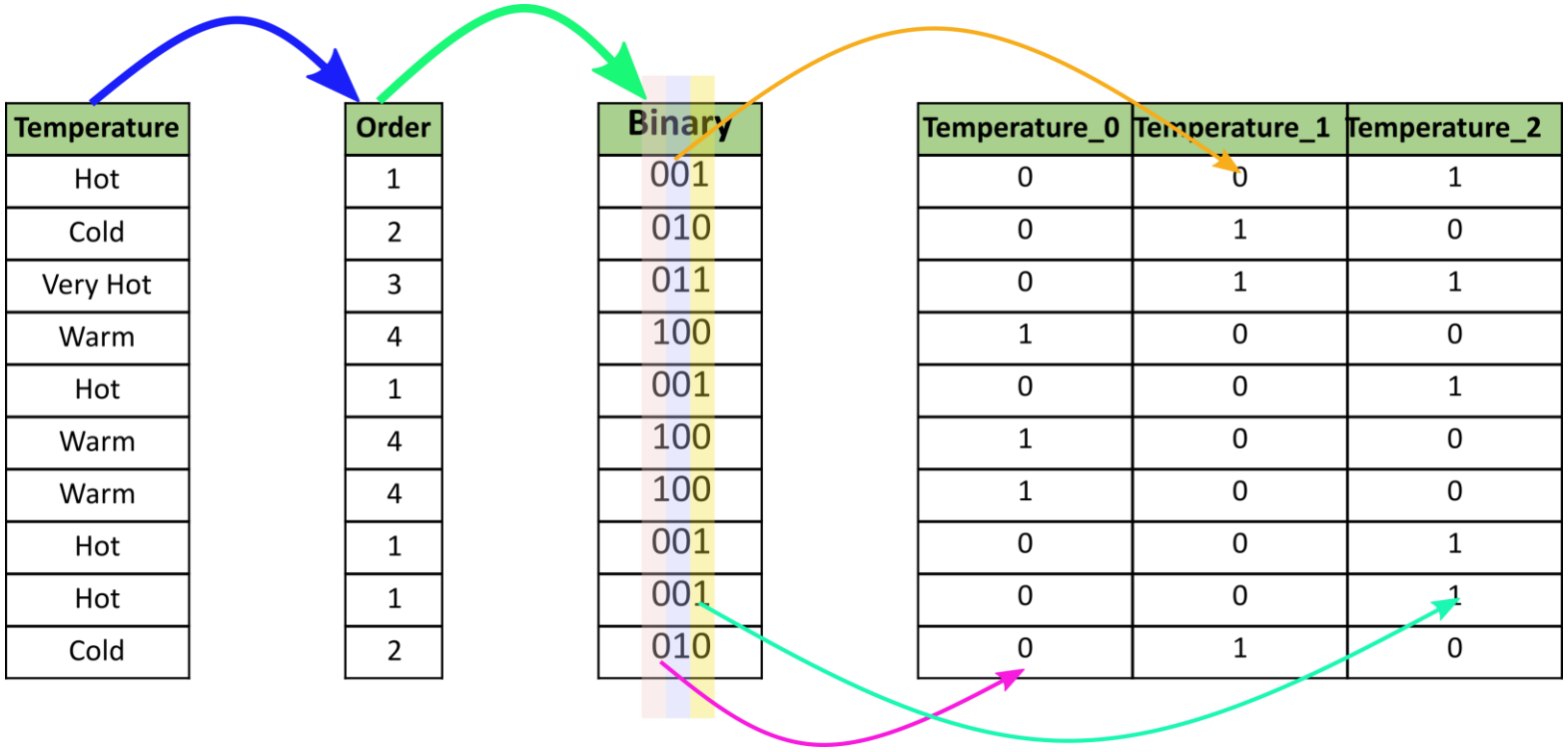


| id | color_red | color_blue | color_green |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 |

# Кодирование категориальных признаков

*Binary Encoding*

Сначала используется Label Encoding, затем лейблы переводятся в двоичный вид



| Temperature |
|---|
| Hot |
| Cold |
| Very Hot |
| Warm |
| Hot |
| Warm |
| Warm |
| Hot |
| Hot |
| Cold |

| Order |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 1 |
| 4 |
| 4 |
| 1 |
| 1 |
| 2 |

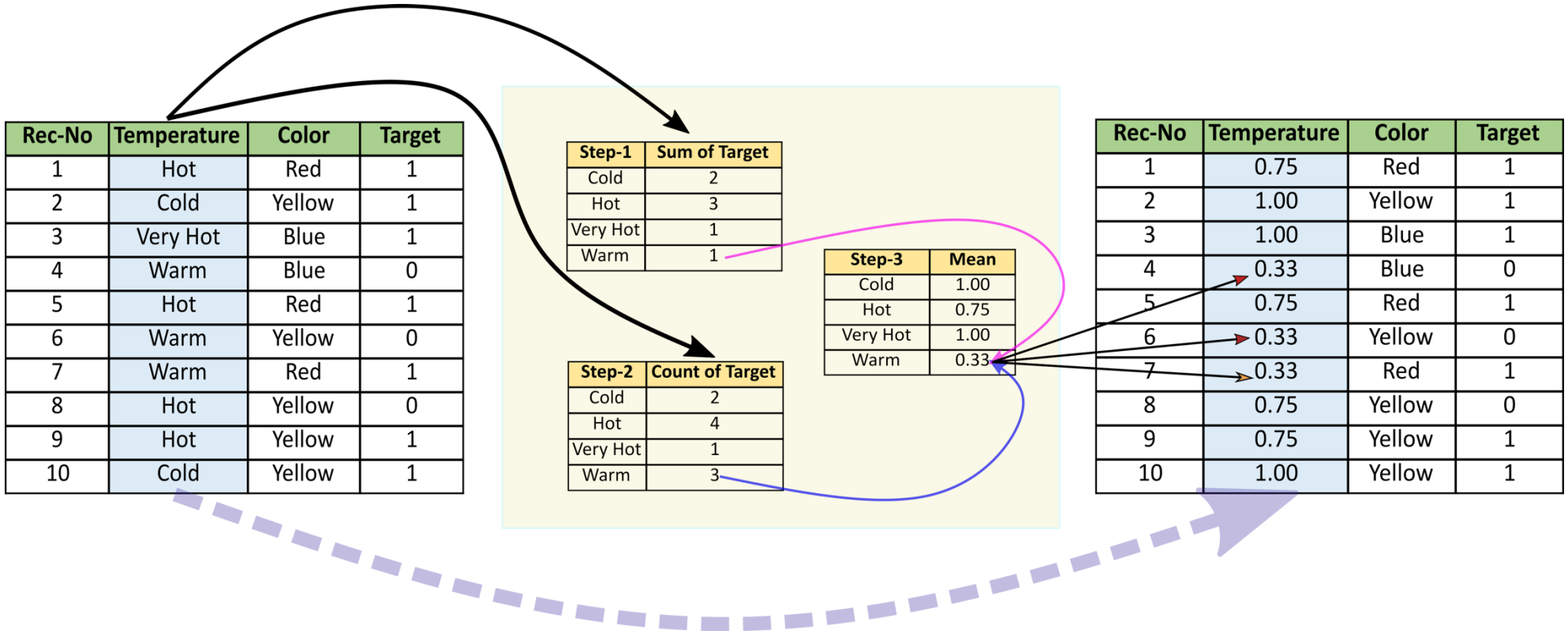
| Binary |
|---|
| 001 |
| 010 |
| 011 |
| 100 |
| 001 |
| 100 |
| 100 |
| 001 |
| 001 |
| 010 |

| Temperature_0 | Temperature_1 | Temperature_2 |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |

# Кодирование категориальных признаков

*Target (Mean) Encoding*

Заменяет каждую категорию на среднее значение целевой переменной



# Кодирование категориальных признаков

*Target (Mean) Encoding*

В чем минус метода?

# Кодирование категориальных признаков

*Target (Mean) Encoding*

В чем минус метода?

Мы заранее закладываем информацию о целевой переменной, тем самым переобучаемся

# Кодирование категориальных признаков

*Target (Mean) Encoding*

В чем минус метода?

Мы заранее закладываем информацию о целевой переменной, тем самым переобучаемся

*Leave-One-Out Encoding*

Модификация Target Encoding, в которой для каждого объекта исключается его собственное значение при расчете среднего, что позволяет уменьшить утечку информации (data leakage)

# Кодирование категориальных признаков

*Хеширование признаков*

Развивает идею One-Hot Encoding, позволяя получить любое заранее заданное число новых столбцов после кодировки.

# Кодирование категориальных признаков

*Хеширование признаков*

Развивает идею One-Hot Encoding, позволяя получить любое заранее заданное число новых столбцов после кодировки.

- Для каждого значения признака вычисляется значение некоторой функции — *hash-функции,* которая группирует значения признаков: часто встречающиеся формируют отдельную группу, редкие — попадают в одну.

# Кодирование категориальных признаков

*Хеширование признаков*

Развивает идею One-Hot Encoding, позволяя получить любое заранее заданное число новых столбцов после кодировки.

- Для каждого значения признака вычисляется значение некоторой функции — *hash-функции,* которая группирует значения признаков: часто встречающиеся формируют отдельную группу, редкие — попадают в одну.

- Задается итоговое количество различных значений признака — *hash_bucket_size*.

# Кодирование категориальных признаков

*Хеширование признаков*

Развивает идею One-Hot Encoding, позволяя получить любое заранее заданное число новых столбцов после кодировки.

- Для каждого значения признака вычисляется значение некоторой функции — *hash-функции,* которая группирует значения признаков: часто встречающиеся формируют отдельную группу, редкие — попадают в одну.
- Задается итоговое количество различных значений признака — *hash_bucket_size*.
- Берется остаток: *hash % hash_bucket_size*, каждое значение признака кодируется числом от 0 до *hash_bucket_size — 1.*

# Кодирование категориальных признаков

*Хеширование признаков*

Развивает идею One-Hot Encoding, позволяя получить любое заранее заданное число новых столбцов после кодировки.

- Для каждого значения признака вычисляется значение некоторой функции — *hash-функции,* которая группирует значения признаков: часто встречающиеся формируют отдельную группу, редкие — попадают в одну.
- Задается итоговое количество различных значений признака — *hash_bucket_size*.
- Берется остаток: *hash % hash_bucket_size*, каждое значение признака кодируется числом от 0 до *hash_bucket_size — 1.*
- К полученным числам применяется ОНЕ

# Кодирование категориальных признаков

*Хеширование признаков*