

Проблема переобучения и недообучения модели. Кросс-валидация и регуляризация.

Урок 6

На этой лекции вы найдете ответы на такие вопросы как:

- Какие задачи решает классификация
- Что такое логистическая регрессия
- Метрики качества классификации



Булгакова Татьяна

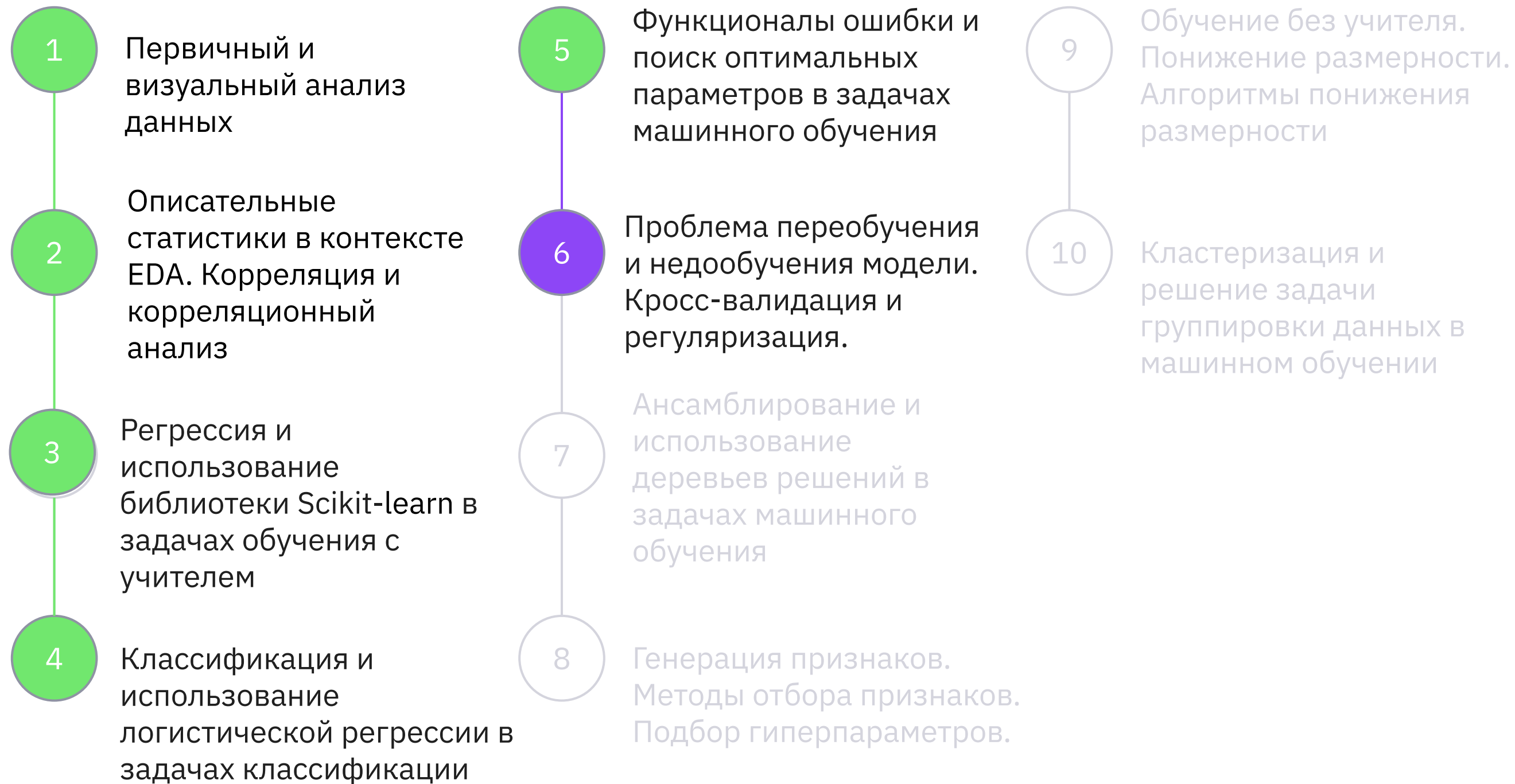
Преподаватель в GeekBrains, Нетология, Skillfactory

С 2010 года занимаюсь DataScience и NN. Фрилансер

- Участвовала в разработке программы по настройке оборудования для исследования пространственного слуха китообразных НИИ ИПЭЭ РАН
- Участвую в разработке рекомендательных систем по настройке нейростимуляторов для медицинских центров
- Работаю над курсом по нейронным сетям






План курса





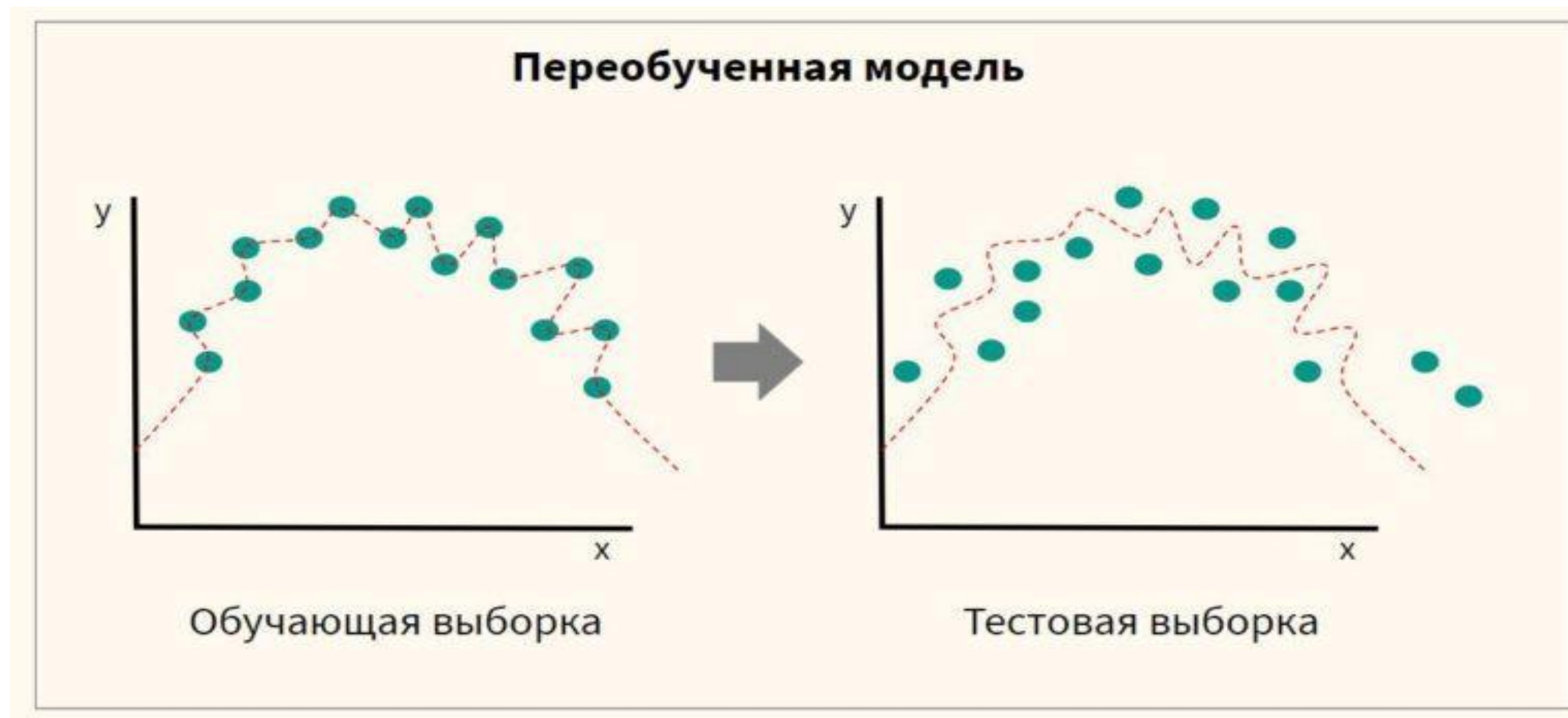
Что будет на уроке сегодня

-  Проблемах переобучения и недообучения модели
-  Способах борьбы с проблемой переобучения и недообучения
-  Кросс-валидации и регуляризации



Переобучение (overfitting)

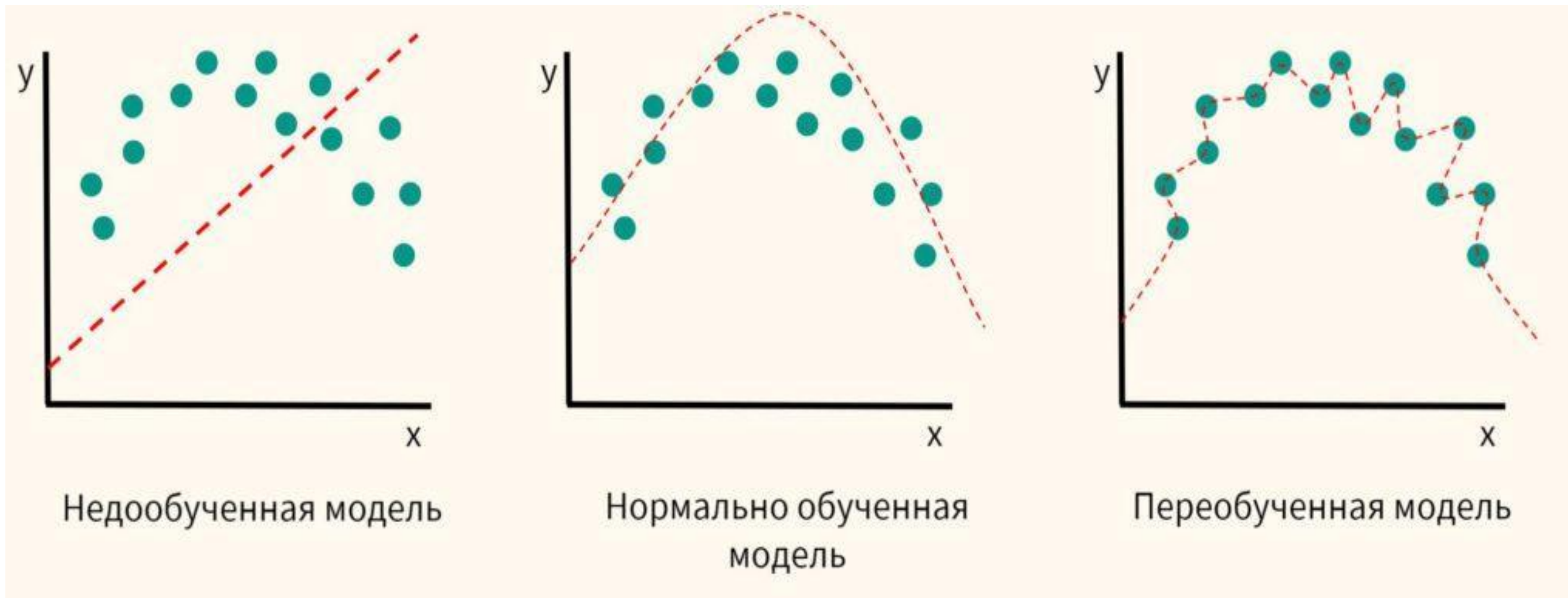
Переобучение (overfitting) происходит, когда модель демонстрирует высокую точность на обучающей выборке, но показывает низкую производительность на тестовых данных.





Недообучение (underfitting)

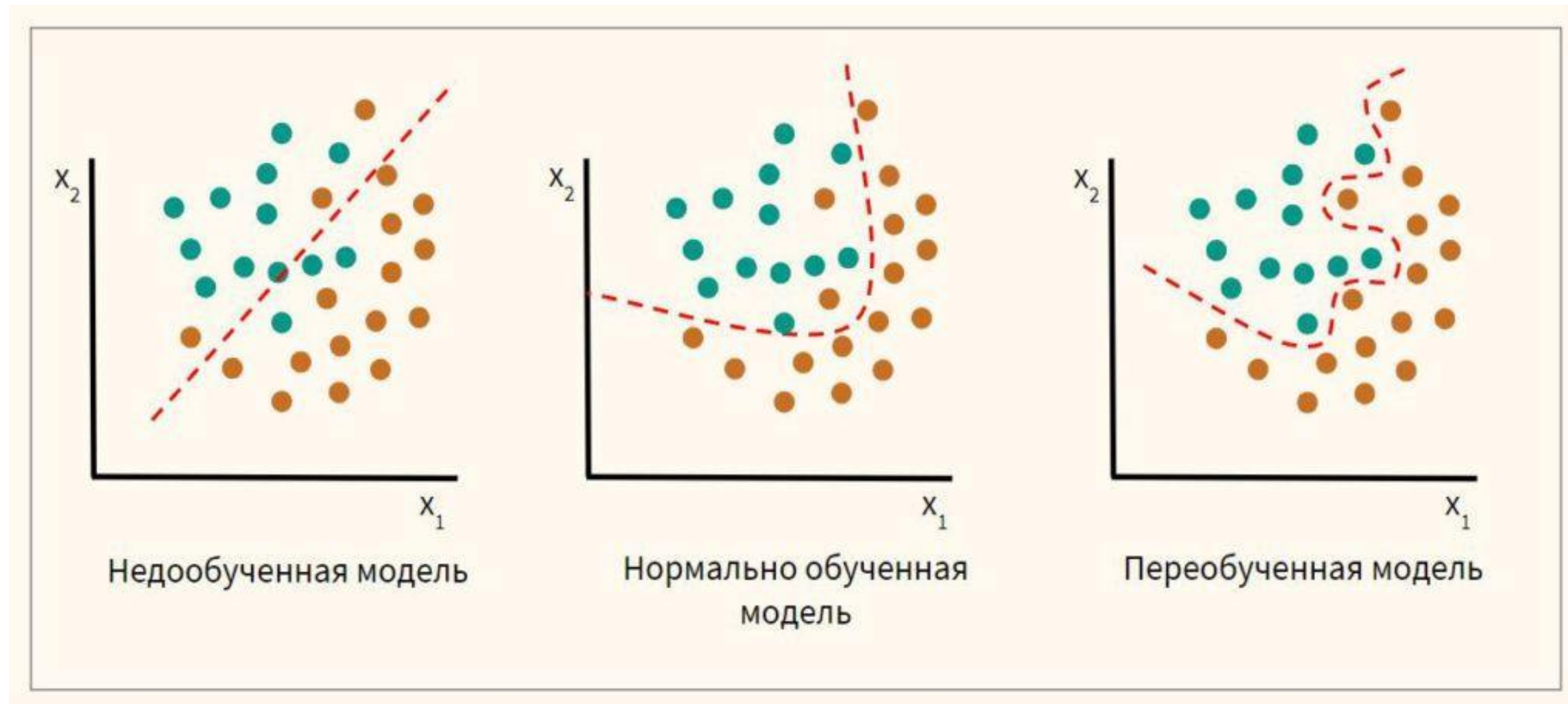
В случае недообучения (underfitting), наоборот, алгоритм не способен корректно определить зависимости на обучающей выборке.





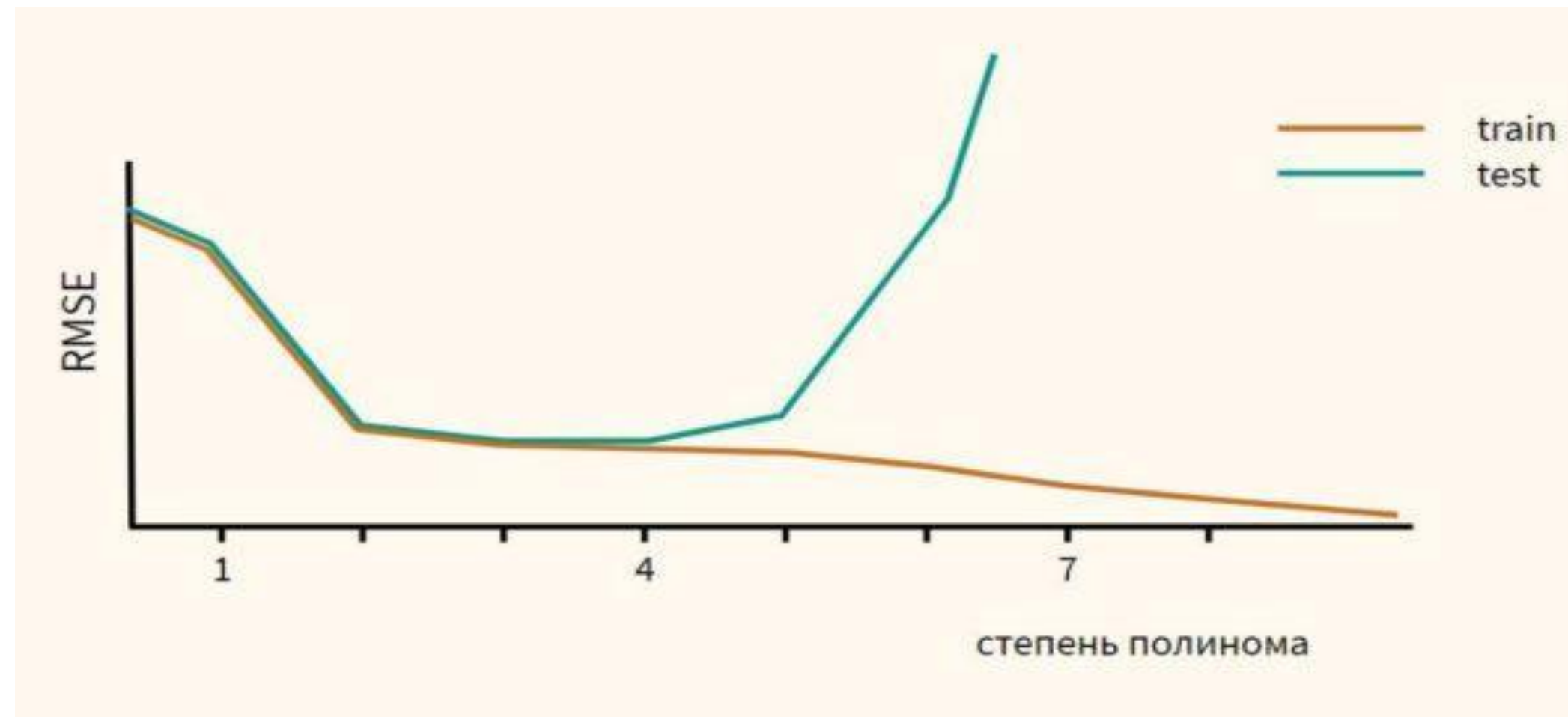
Переобучение (overfitting) и недообучение (underfitting)

Переобучение (overfitting) и недообучение (underfitting) - это два проблемных состояния в машинном обучении, которые могут возникать при построении моделей.





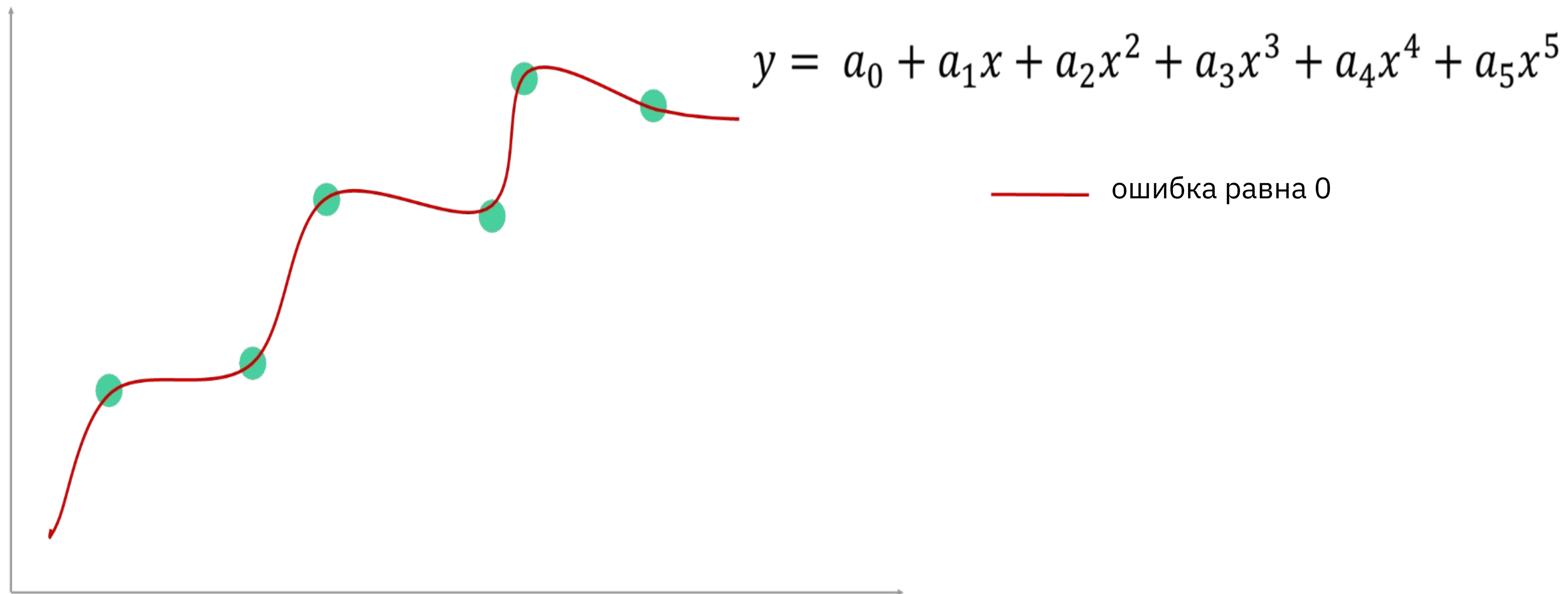
Переобучение (overfitting) и недообучение (underfitting)



В общем, можно сказать, что простая модель склонна к недообучению, а сложная модель - к переобучению.



Переобучение (overfitting) и недообучение (underfitting)

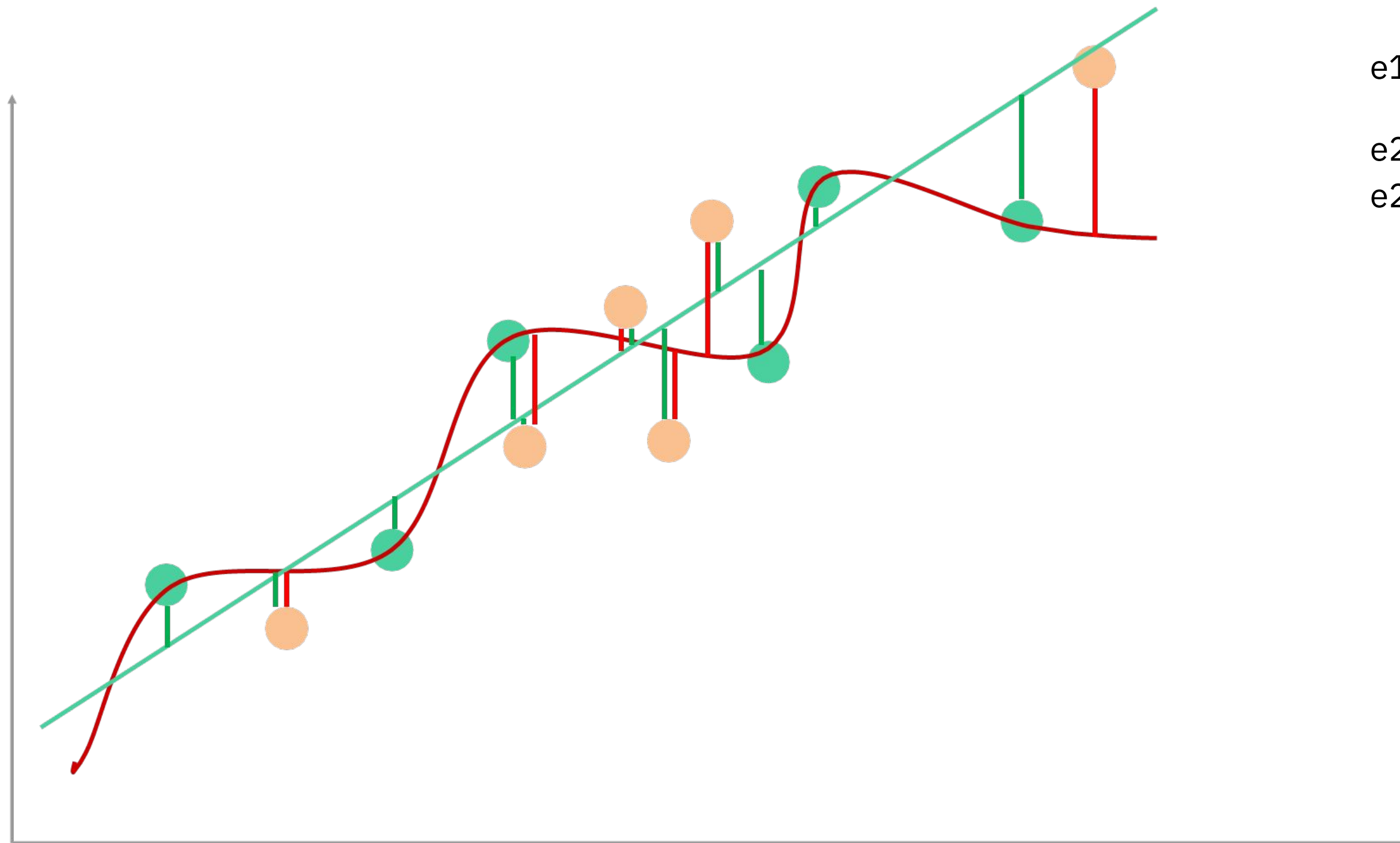


В общем, можно сказать, что простая модель склонна к недообучению, а сложная модель - к переобучению.



Переобучение (overfitting) и недообучение (underfitting)

На тестовых данных получаем большую ошибку



e_1 - ошибка на новых данных > 0

e_2 - ошибка на новых данных > 0

$e_2 > e_1$



Снова про смещение и разброс

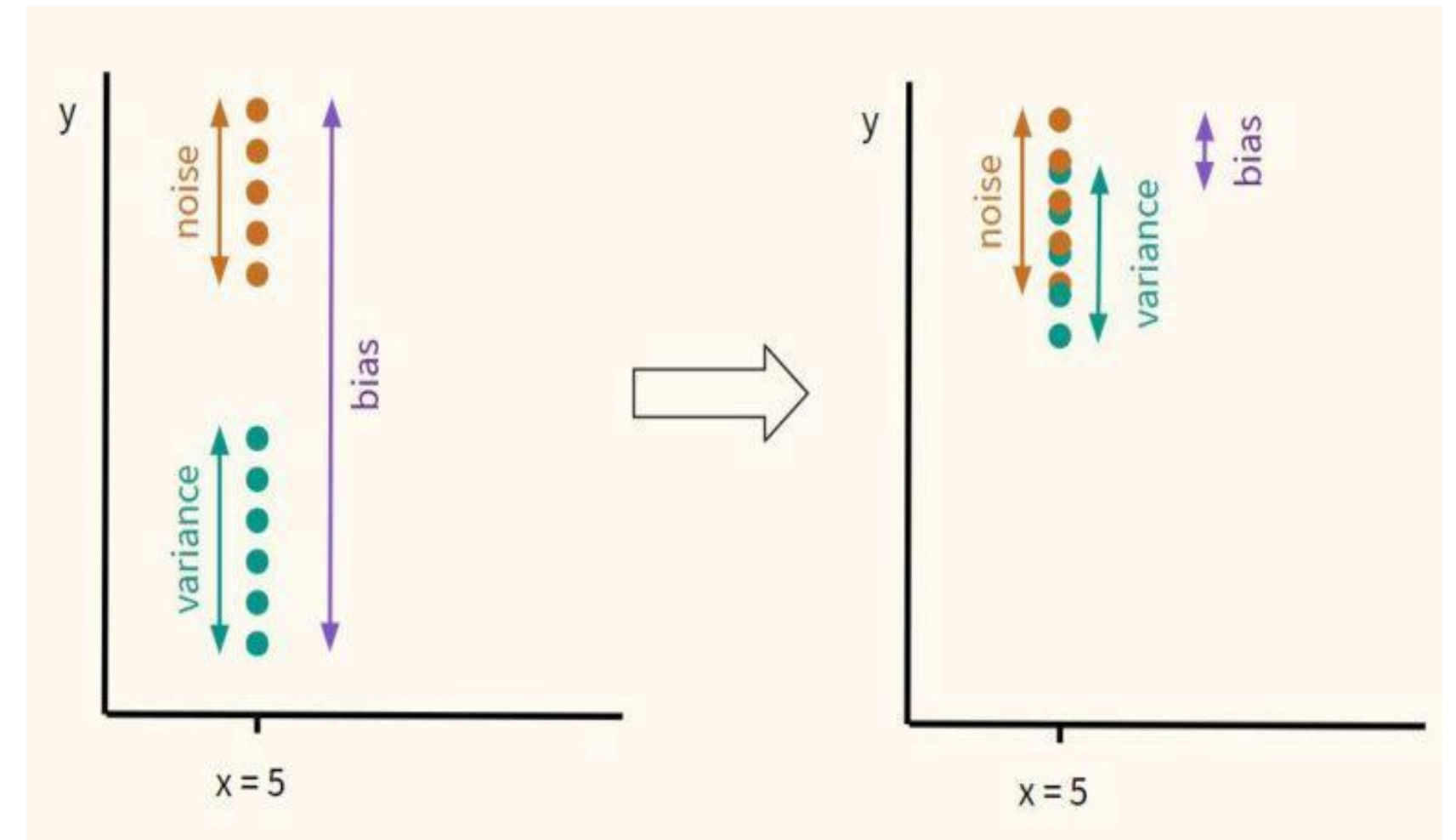
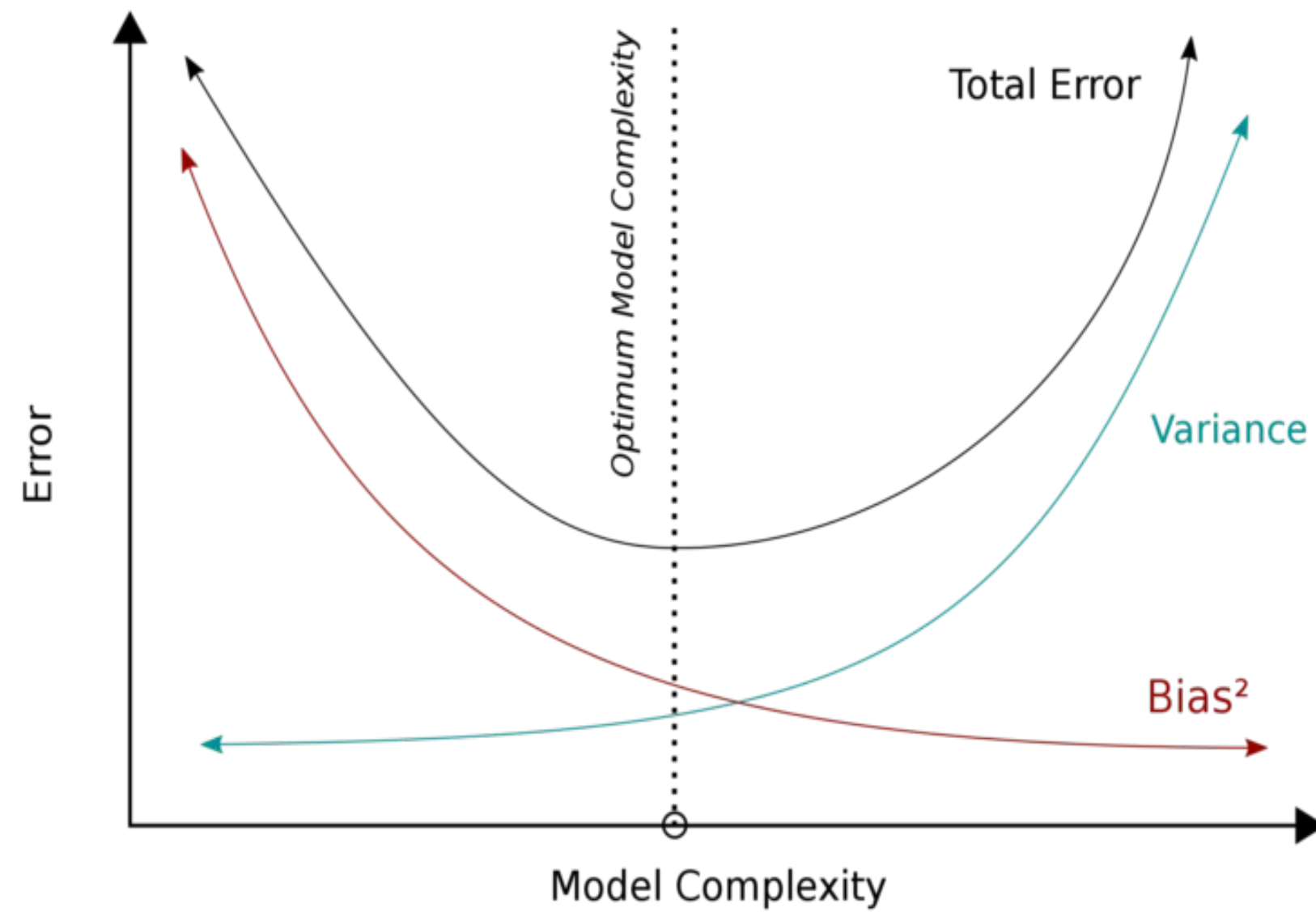
Смещение (bias) - это ожидаемая разница между истинным и прогнозируемым значением.

Разброс (variance) показывает, насколько результаты модели на обучающей выборке отличаются от результатов модели на тестовой выборке.

Практическое стремление к уменьшению смещения может привести к увеличению разброса, что означает переобучение. В то же время, высокое смещение модели может приводить к низкому разбросу, но в этом случае модель будет недообучена и не будет иметь смысла. Эта проблема называется компромиссом или дилеммой смещения и разброса (bias-variance tradeoff).



Снова про смещение и разброс





Как бороться с переобучением?

Существует три способа борьбы с этим:

1. Увеличение размера обучающей выборки. Маленькая выборка снижает способность модели к обобщению, что ведет к увеличению разброса.
2. Уменьшение количества признаков (вручную или с использованием алгоритма), а также введение наказания за новые признаки, например, скорректированный коэффициент детерминации (adjusted R-square). Однако здесь есть риск удалить важные признаки.
3. Использование регуляризации, которая позволяет уменьшить параметр (вес, коэффициент) признака и, следовательно, его влияние.



Регуляризация

Это форма регрессии, которая ограничивает , упорядочивает или сводит оценки коэффициентов к нулю. Другими словами, этот метод препятствует изучению более сложной или гибкой модели, чтобы избежать риска переобучения.

$$\text{Regularization} = \text{Loss Function} + \text{Penalty}$$

Regularization - регуляризация

Loss Function - функция потерь

Penalty - штраф



Регуляризация

Смысл регуляризации заключается в минимизации функционала ошибки с ограничением весов

Чем больше λ , тем меньшая сложность модели будет получаться в процессе обучения:




- Если увеличивать его, в какой-то момент оптимальным для модели окажется зануление всех весов
 - При слишком низких его значениях появляется вероятность чрезмерного усложнения модели и переобучения
- λ подбираем по метрикам и кросс-валидации

$$\begin{cases} Q(a, X) \rightarrow \min \\ ||a||^2 \leq C \end{cases}$$



Регуляризация

Существует три широко используемых метода регуляризации для управления сложностью моделей машинного обучения, а именно:

-  Регуляризация L2
-  Регуляризация L1
-  Эластичная сеть



Регуляризация L2

L-2 регуляризация, также известная как регуляризация риджа, представляет собой метод регуляризации, при котором к сумме квадратов весов модели добавляется сумма квадратов весов, умноженная на параметр регуляризации. L-2 регуляризация способствует уменьшению величины всех весов модели, но они остаются ненулевыми.

$$\text{Ridge Regression Cost Function} = \text{Loss Function} + \frac{1}{2} \lambda \sum_{j=1}^m w_j^2$$

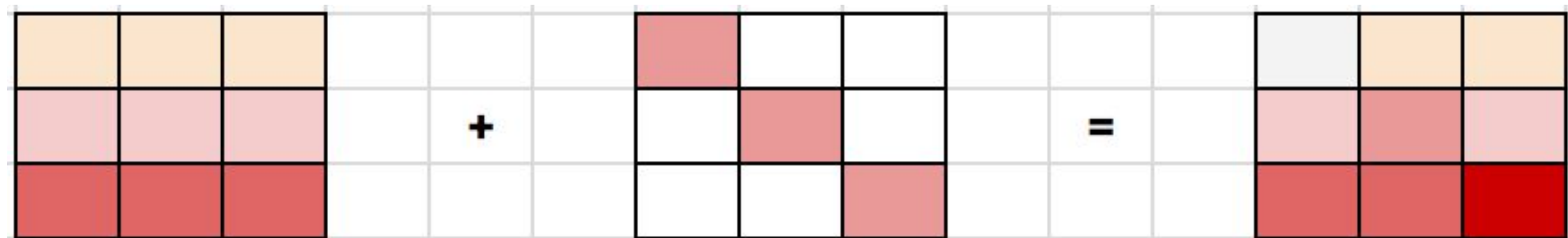
Ridge Regression Cost Function - функция стоимости гребневой регрессии

Loss Function - функция потерь



Регуляризация L2

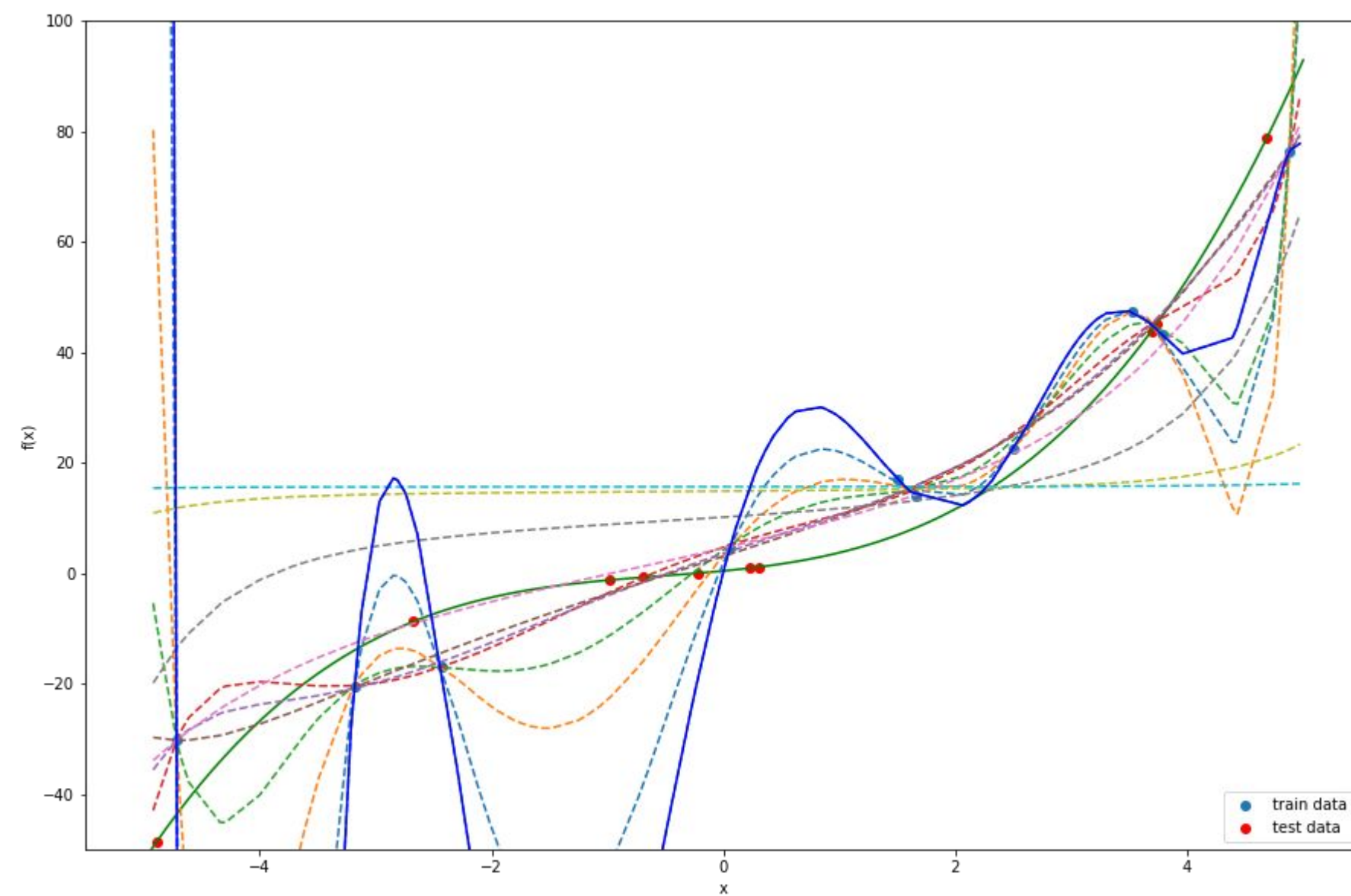
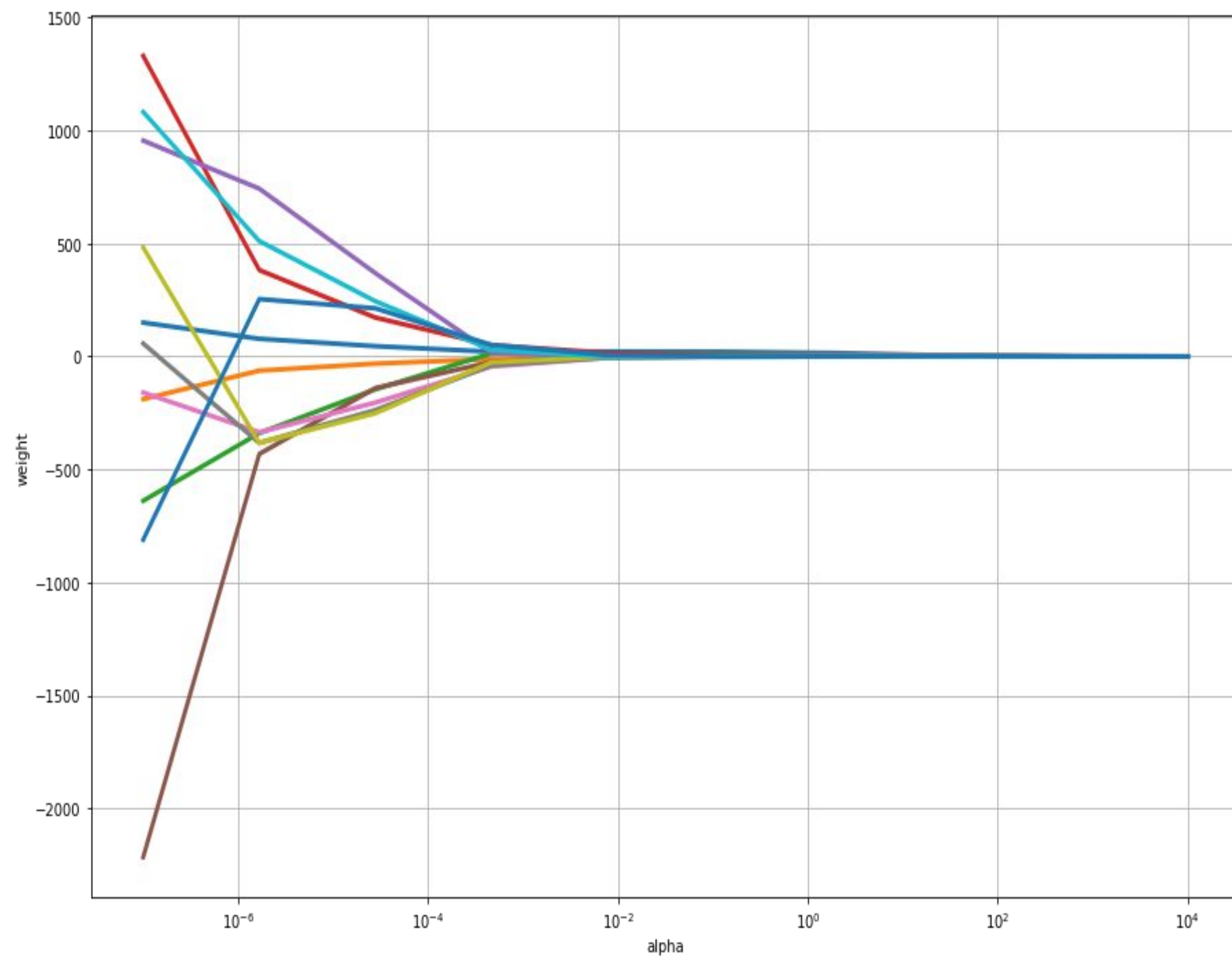
Такая регрессия называется гребневой регрессией (ridge regression). А гребнем является как раз диагональная матрица которую мы прибавляем к матрице с линейнозависимыми колонками, в результате получаемая матрица не сингулярна.



Таким образом, увеличивая параметр регуляризации мы уменьшаем число обусловленности, а обусловленность задачи улучшается.



Регуляризация L2





Регуляризация L2

`sklearn.linear_model.Ridge`

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None,  
tol=0.0001, solver='auto', positive=False, random_state=None) \[source\]
```

Эта модель решает регрессионную модель, где функция потерь является линейной функцией наименьших квадратов, а регуляризация задается l2-нормой. Также известна как регрессия хребта или регуляризация Тихонова.



Регуляризация L1

L-1 регуляризация, также известная как регуляризация Лассо, представляет собой метод регуляризации, при котором к сумме квадратов весов модели добавляется сумма абсолютных значений весов, умноженная на параметр регуляризации. L-1 регуляризация способствует сжатию некоторых весов до нулевых значений, что может быть использовано для отбора признаков

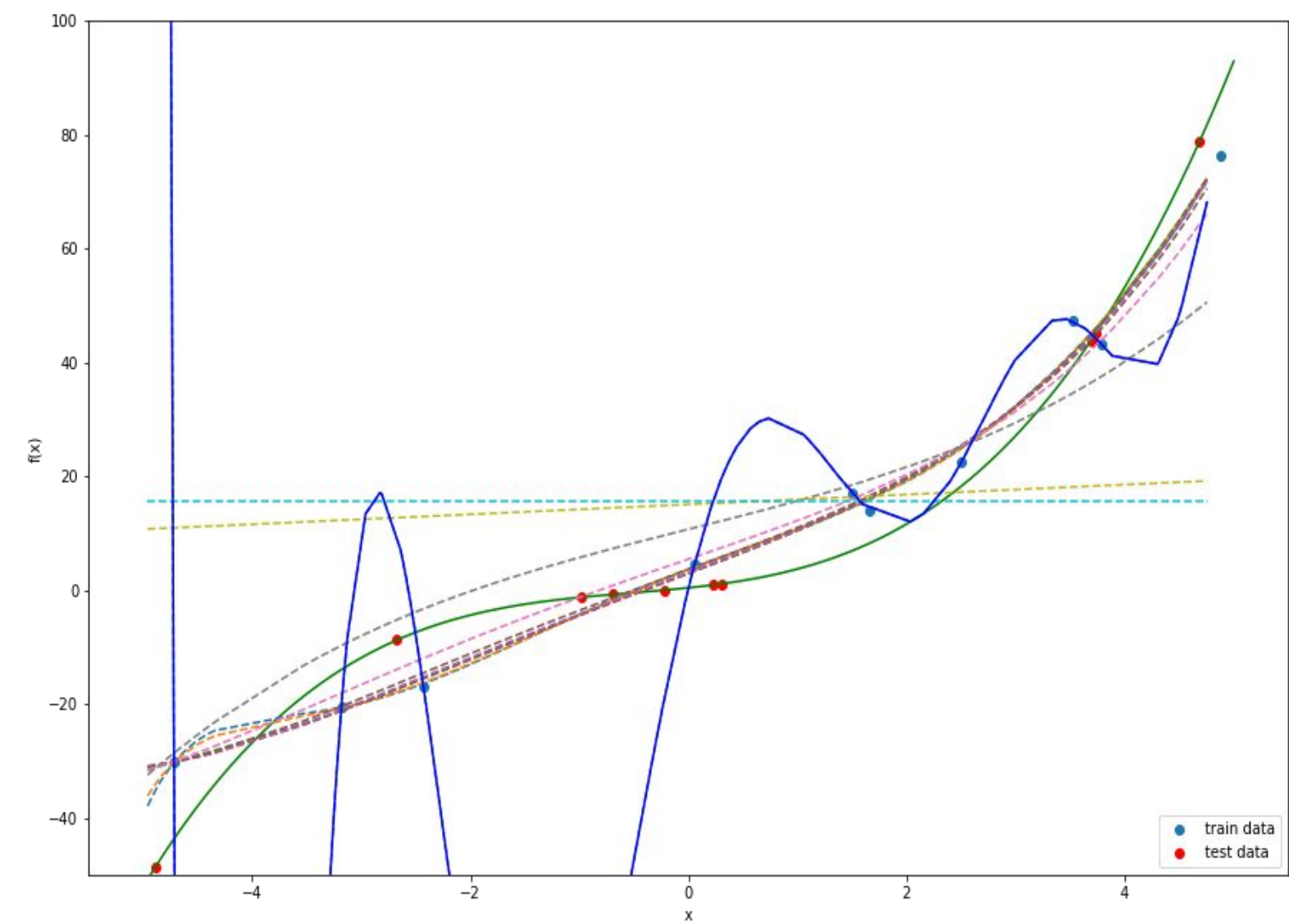
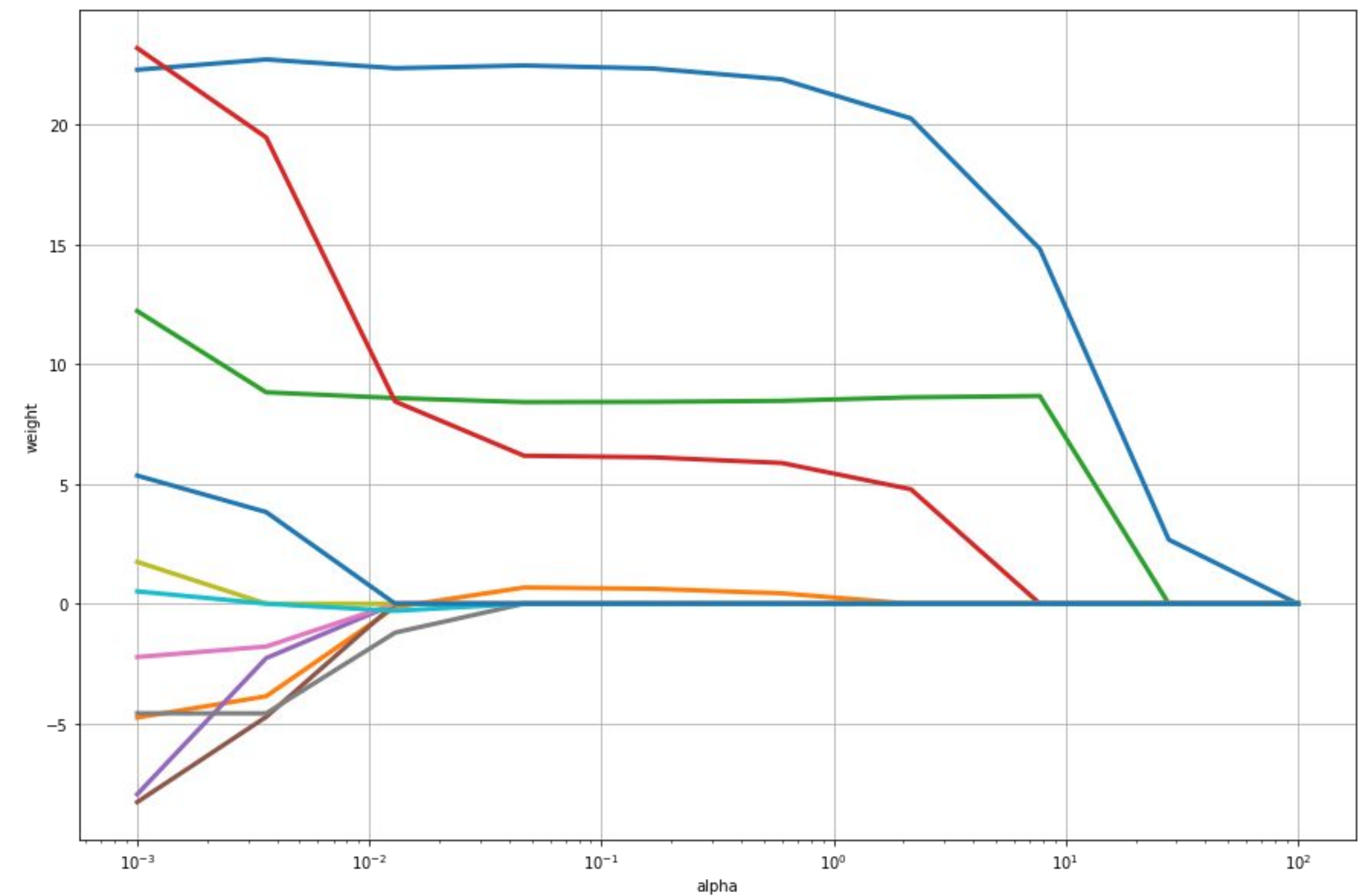
$$\text{Lasso Regression Cost Function} = \text{Loss Function} + \lambda \sum_{j=1}^m |w_j|$$

Lasso Regression Cost Function - функция стоимости регрессии Лассо

Loss Function - функция потерь



Регуляризация L1





Регуляризация L1

`sklearn.linear_model.Lasso`

```
class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, precompute=False,  
copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None,  
selection='cyclic')
```

[\[source\]](#)

Метод регрессии лассо (LASSO, Least Absolute Shrinkage and Selection Operator) — это вариация линейной регрессии, специально адаптированная для данных, которые демонстрируют сильную мультиколлинеарность (то есть сильную корреляцию признаков друг с другом).

Такая регуляризация часто приводит к более разреженным моделям с меньшим количеством коэффициентов, так как некоторые коэффициенты могут стать нулевыми и, следовательно, будут исключены из модели. Это позволяет ее интерпретировать.



L1-регуляризация может быть полезна для устранения неинформативных признаков и автоматического отбора признаков



L2-регуляризация больше подходит для уменьшения влияния признаков и предотвращения переобучения моделей.



Elastic Net

Это метод регуляризации, который комбинирует L1 (регуляризация Лассо) и L2 (регуляризация риджа) регуляризации. Это делается путем добавления к сумме квадратов весов модели суммы абсолютных значений весов, умноженных на параметр регуляризации L1, и суммы квадратов весов, умноженных на параметр регуляризации L2.

Elastic Net предоставляет баланс между отбором признаков (L1 регуляризация) и уменьшением весов (L2 регуляризация). Он позволяет моделировать зависимости между переменными, что может быть полезно в случае, когда есть мультиколлинеарность (высокая корреляция между признаками) или когда количество признаков велико.



Elastic Net

Лассо и гребневая регрессия представляют собой два различных метода регуляризации. В обоих случаях λ — это ключевой фактор, который контролирует размер штрафа:

1. Если $\lambda = 0$, то задача становится аналогичной простой линейной регрессии, достигая тех же коэффициентов.
2. Если $\lambda = \infty$, то коэффициенты будут равны нулю из-за бесконечного веса на квадрате коэффициентов. Всё, что меньше нуля, делает цель бесконечной.
3. Если $0 < \lambda < \infty$, то величина λ определяет вес, придаваемый различным частям объекта.

К параметру λ регрессия ElasticNet добавляет дополнительный параметр α , который измеряет, насколько «смешанными» должны быть регуляризации L1 и L2. Когда параметр α равен 0, модель является чисто гребневой регрессией, а когда он равен 1 — это чистая регрессия лассо.

«Коэффициент смешивания» α просто определяет, сколько регуляризации L1 и L2 следует учитывать в функции потерь. Все три популярные регрессионные модели — гребневая, лассо и ElasticNet — нацелены на уменьшение размера своих коэффициентов, но каждая действует по-своему.



Elastic Net

`sklearn.linear_model.ElasticNetCV`

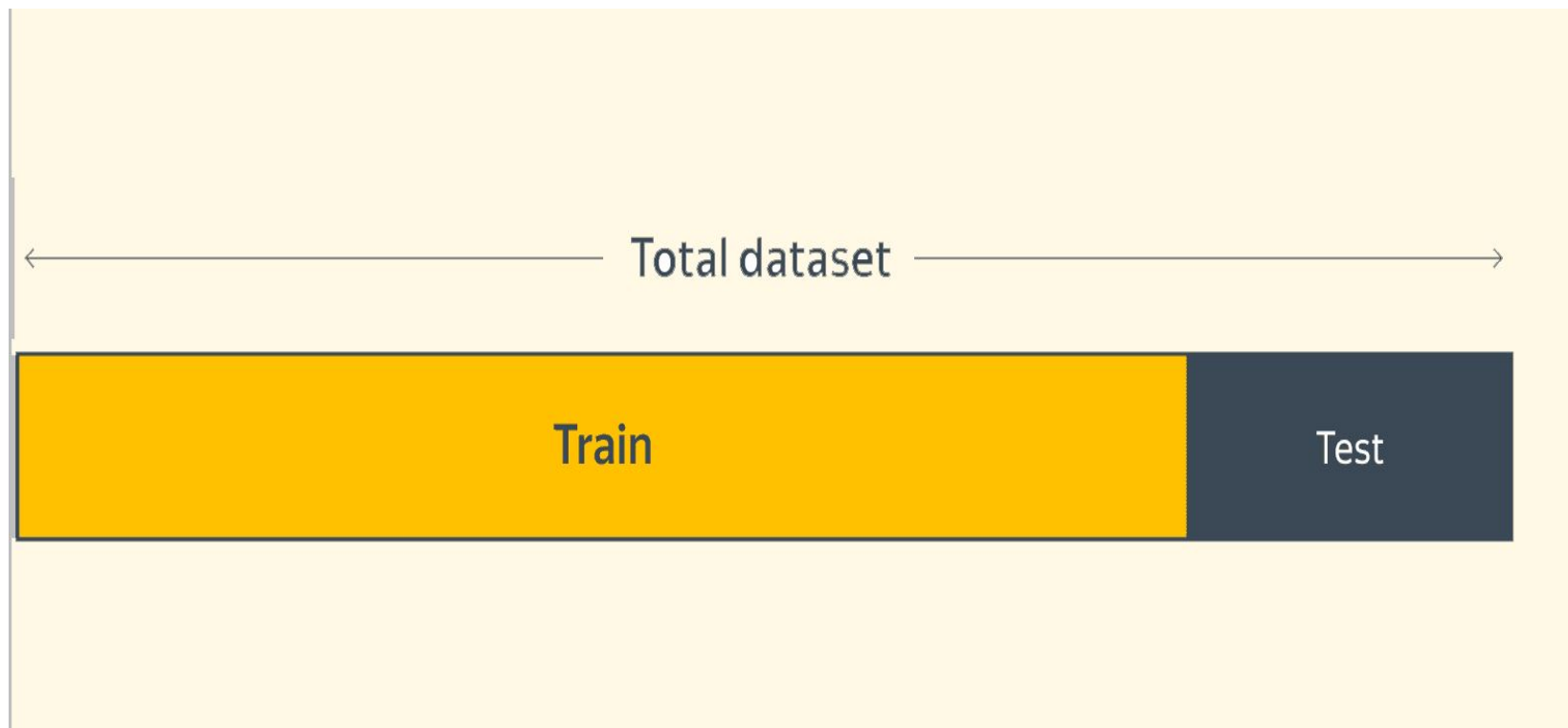
```
class sklearn.linear_model.ElasticNetCV(*, l1_ratio=0.5, eps=0.001, n_alphas=100,  
alphas=None, fit_intercept=True, precompute='auto', max_iter=1000, tol=0.0001, cv=None,  
copy_X=True, verbose=0, n_jobs=None, positive=False, random_state=None, selection='cyclic') \[source\]
```

ElasticNet стремится объединить лучшее из гребневой регрессии и регрессии лассо, комбинируя регуляризацию L1 и L2



Hold-out

Метод hold-out (метод отложенной выборки) представляет из себя простое разделение на train (обучающая) и test (обучающая)



```
import numpy as np
from sklearn.model_selection import train_test_split

X, y = np.arange(1000).reshape((500, 2)), np.arange(500)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train,
    test_size=0.1,
    random_state=42
)
```



Стратификация (stratification)

При случайном разделении на тренировочное и тестовое множества может возникнуть ситуация, когда распределения классов в тренировочном и тестовом множествах будут отличаться от распределения в исходном множестве.

```
import numpy as np
from sklearn.model_selection import train_test_split

X, y = np.arange(1000).reshape((500, 2)), np.random.choice(4, size=500, p=[0.1, 0.2, 0.3, 0.4])
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```



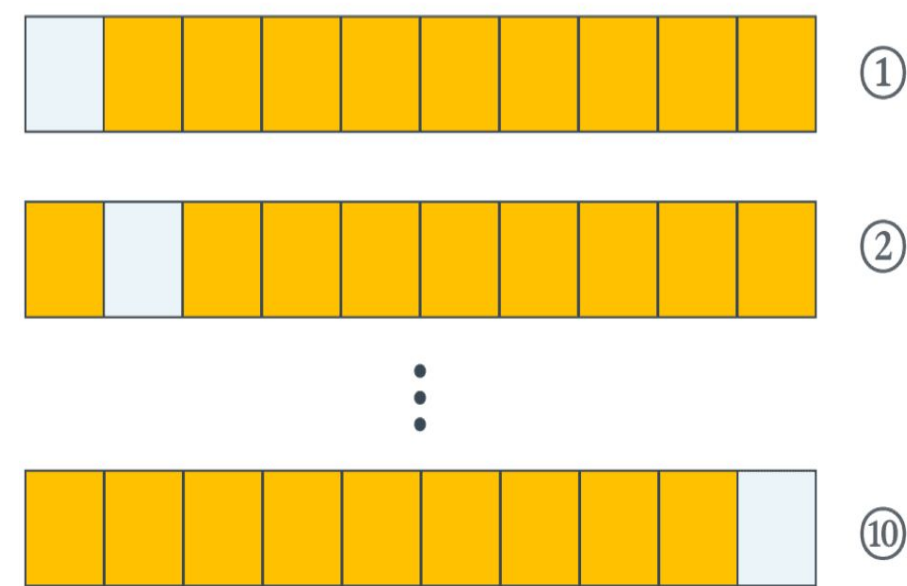

k-Fold

Этот метод является обобщением метода hold-out и представляет следующий алгоритм:

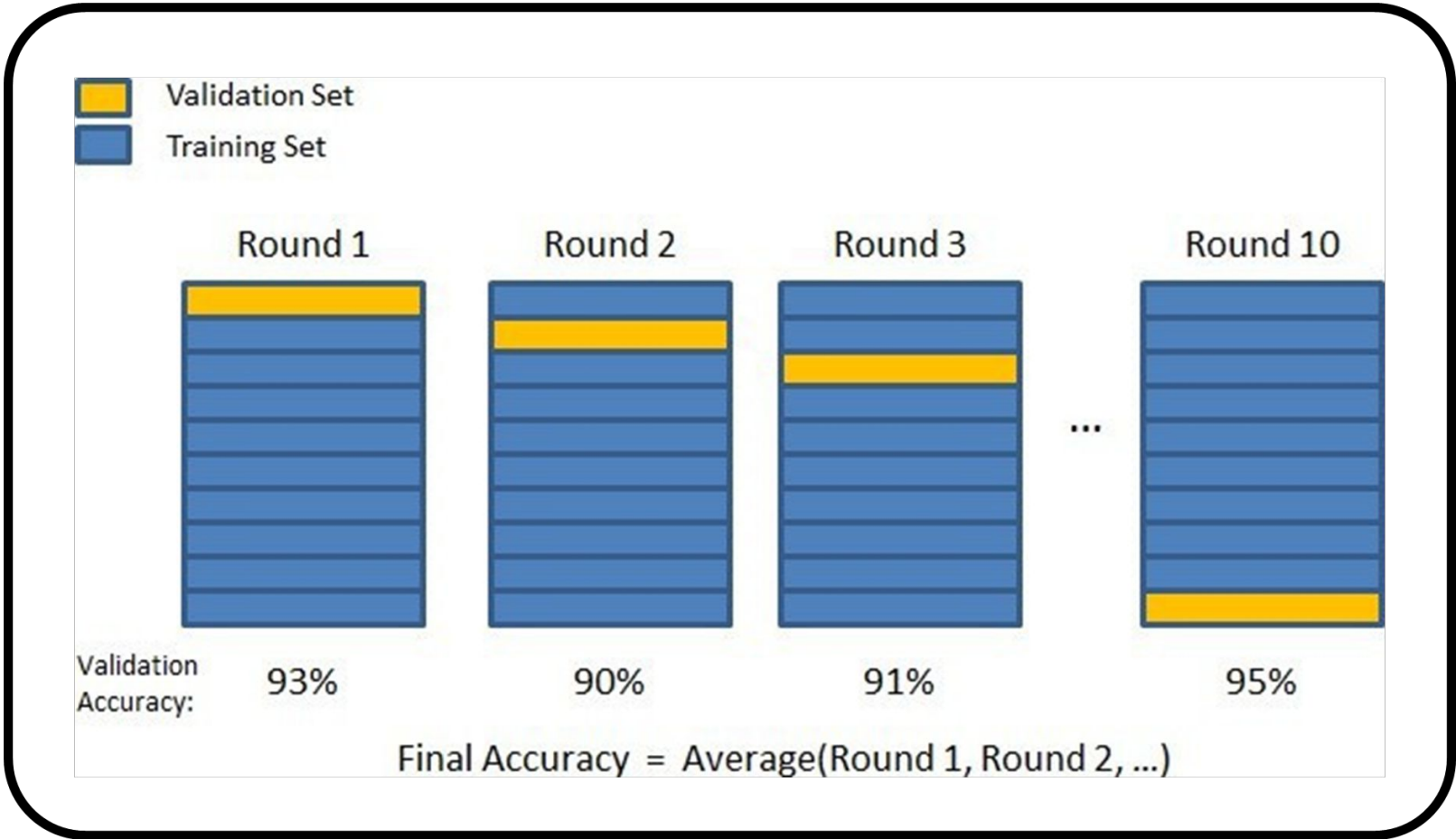
1. Фиксируется некоторое целое число k (обычно от 5 до 10), которое меньше числа семплов в датасете.
2. Датасет разбивается на k одинаковых частей (в последней части может быть меньше семплов, чем в остальных). Эти части называются фолдами.
3. Далее происходит k итераций, во время каждой из которых один фолд выступает в роли тестового множества, а объединение остальных фолдов - в роли тренировочного множества. Модель учится на тренировочном множестве и тестируется на тестовом.
4. Финальный скор модели получается либо усреднением результатов тестирования на всех фолдах, либо измеряется на отложенном тестовом множестве, которое не участвовало в кросс-валидации.



k-Fold



Training
Test
K = 10





k-Fold

```
import numpy as np
from sklearn.model_selection import KFold

X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
y = np.array([1, 2, 3, 4])
kf = KFold(n_splits=2)

for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
...
result:
TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1] TEST: [2 3]
...
```



k-Fold

1. Сделать предсказание, усреднив предсказания всех этих k инстансов.
2. Из этих k инстансов выбрать тот, который показал лучший результат на своем тестовом фолде, и использовать его дальше.
3. Повторно обучить модель уже на k фолдах и делать предсказания уже с помощью этой модели.

```
import numpy as np
from sklearn.model_selection import cross_val_score

clf = svm.SVC(kernel='linear', C=1, random_state=42)
scores = cross_val_score(clf, X, y, cv=5)
print(scores)
'''
result:
array([0.96..., 1. , 0.96..., 0.96..., 1. ])
```



Leave-one-out (LOO)

Метод leave-one-out (LOO) является частным случаем метода k-Fold: в нём каждый фолд состоит ровно из одного семпла.

```
import numpy as np
from sklearn.model_selection import LeaveOneOut

X = np.array([[1, 2], [3, 4], [5, 6]])
y = np.array([1, 2, 3])
loo = LeaveOneOut()

for train_index, test_index in loo.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
...
result:
TRAIN: [1 2] TEST: [0]
TRAIN: [0 2] TEST: [1]
TRAIN: [0 1] TEST: [2]
...
```




Stratified k-Fold

Метод stratified k-Fold представляет собой вариацию метода k-Fold, где используется стратификация при разделении на фолды. Каждый фолд содержит примерно такое же соотношение классов, как и в исходном наборе данных.

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

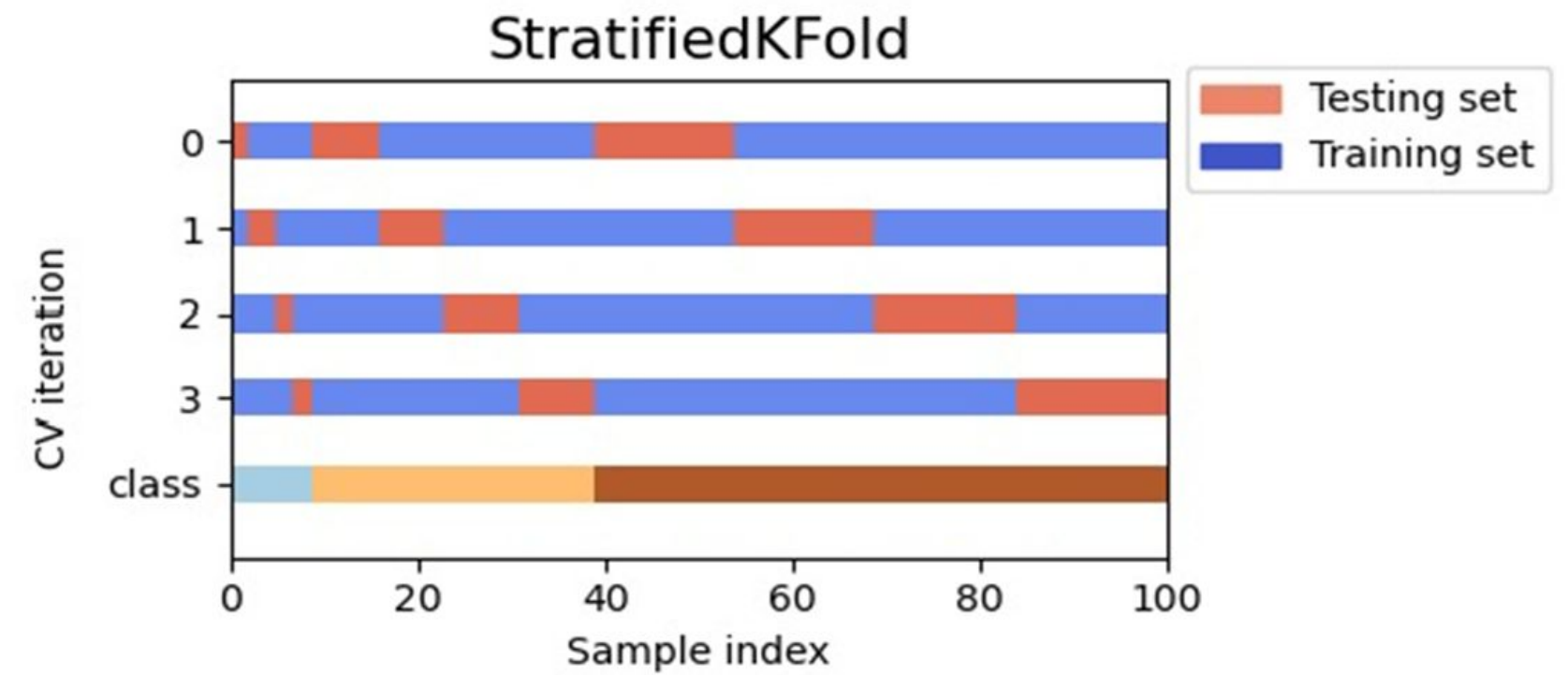
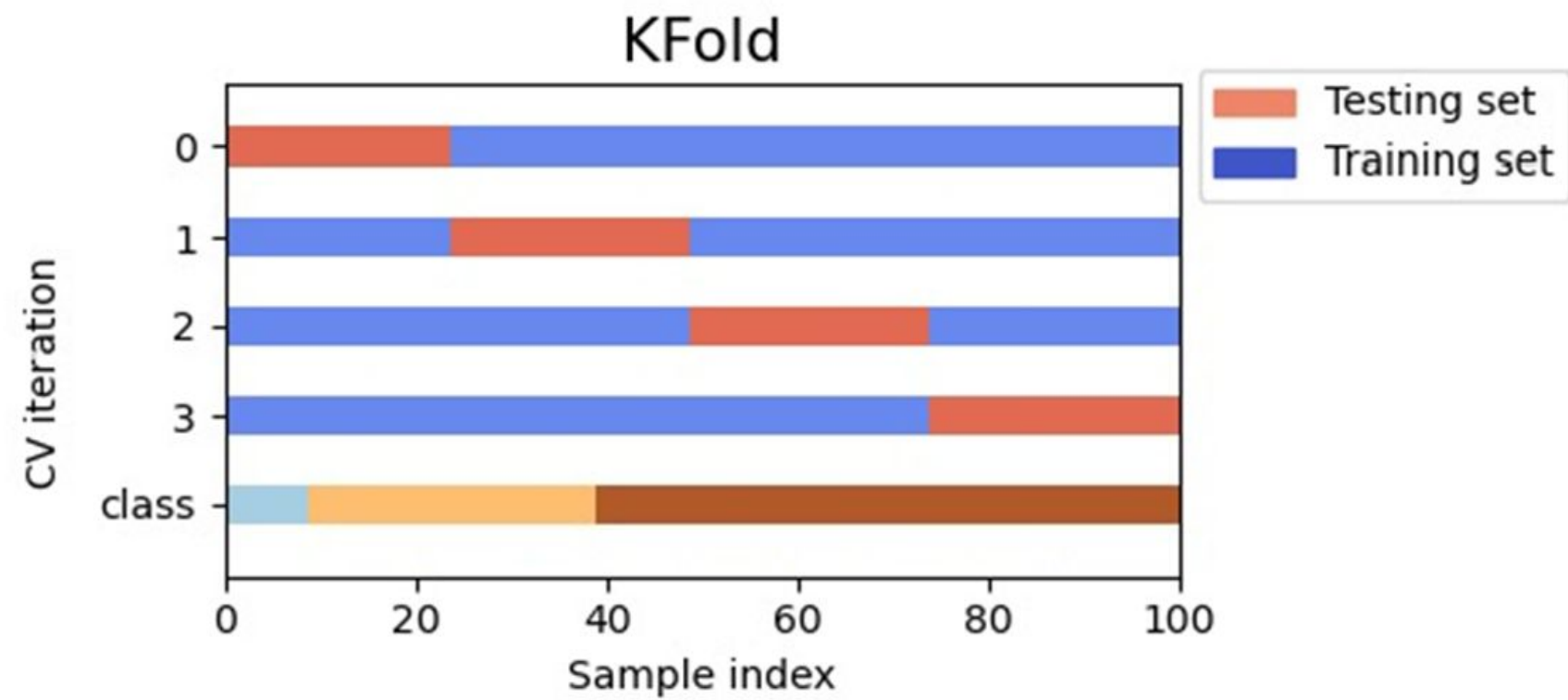
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
skf = StratifiedKFold(n_splits=2)

for train_index, test_index in skf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    ...

result:
TRAIN: [1 3] TEST: [0 2]
TRAIN: [0 2] TEST: [1 3]
...
```




Stratified k-Fold





Итоги



В ходе данной лекции мы рассмотрели две проблемы, которые возникают при обучении моделей: переобучение и недообучение.



Переобучение возникает, когда модель слишком хорошо подстроилась под тренировочные данные, однако не способна обобщить полученные знания на новые данные. Это происходит, когда модель слишком сложная и излишне запоминает шумы в данных. Переобучение можно предотвратить с помощью кросс-валидации и регуляризации.



Недообучение, в свою очередь, возникает, когда модель недостаточно обучена и не способна выявить скрытые закономерности в данных. Это происходит, когда модель слишком простая и не может запомнить достаточное количество информации из тренировочных данных. Проблему недообучения можно решить путем увеличения сложности модели, добавления новых признаков или использования другого алгоритма машинного обучения.



Итоги



Для предотвращения переобучения и недообучения моделей можно использовать кросс-валидацию. Кросс-валидация позволяет оценить качество работы модели на независимом наборе данных и предотвратить переобучение. Это достигается путем разделения исходных данных на несколько подмножеств и обучения модели на одной части данных, а оценке ее работы на другой. Повторение этого процесса несколько раз с разными разбиениями данных позволяет получить более надежную оценку качества модели.



Регуляризация - это методика, которая используется для управления сложностью моделей и предотвращения переобучения. Регуляризация добавляет дополнительный штраф к функции потерь и позволяет модели находить более обобщающие закономерности в данных.



Практическое применение данных тем заключается в том, что переобучение и недообучение являются распространенными проблемами в машинном обучении. Использование кросс-валидации и регуляризации позволяет более эффективно и точно обучать модели, улучшать их качество и достигать лучших результатов. Кросс-валидация позволяет выбирать наилучшие гиперпараметры модели и оценивать ее работу на новых данных, а регуляризация позволяет контролировать сложность модели и предотвращать переобучение. Эти методы являются важным инструментом для применения в задачах обучения моделей на реальных данных.



Спасибо за внимание

