GeekBrains

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

Библиотеки Python для Data Science

Оглавление

Введение

Добро пожаловать на лекцию по проблеме переобучения и недообучения модели, а также о методах, которые помогают справиться с этими проблемами — кросс-валидации и регуляризации.

В машинном обучении одной из основных задач является создание моделей, которые могут обобщать данные и делать точные прогнозы для новых наблюдений. Однако существует две основные проблемы, связанные с обучением моделей: переобучение и недообучение.

Переобучение возникает, когда модель слишком точно подстраивается под тренировочные данные, несмотря на то, что они могут содержать случайные или нерепрезентативные шумы. Такая модель будет иметь плохую обобщающую способность и будет плохо предсказывать новые данные.

С другой стороны, недообучение возникает, когда модель слишком проста и не способна захватить сложности данных. Такие модели будут иметь высокую ошибку как на тренировочных данных, так и на новых данных.

Для решения этих проблем существуют методы, такие как кросс-валидация и регуляризация. Кросс-валидация позволяет оценить качество модели на независимых данных, используя разделение имеющихся данных на обучающую и валидационную выборки. Регуляризация, в свою очередь, добавляет некоторые ограничения на модель, чтобы сделать ее более устойчивой к переобучению.

В этой лекции мы погрузимся в детали проблемы переобучения и недообучения, рассмотрим, как работает кросс-валидация и регуляризация, и изучим, как правильно применять эти методы для создания моделей с лучшей обобщающей способностью.

На этой лекции вы найдете ответы на такие вопросы как / узнаете:

- Проблемах переобучения и недообучения модели
- Способах борьбы с проблемой переобучения и недообучения
- Кросс-валидация и регуляризация

Термины, используемые в лекции

Переобучение — это ситуация, когда модель слишком тесно прилегает к обучающим данным и плохо обобщает свои предсказания на новые, неизвестные данные.

Недообучение — это ситуация, когда модель недостаточно сложна или недостаточно обучена, чтобы хорошо обобщать данные.

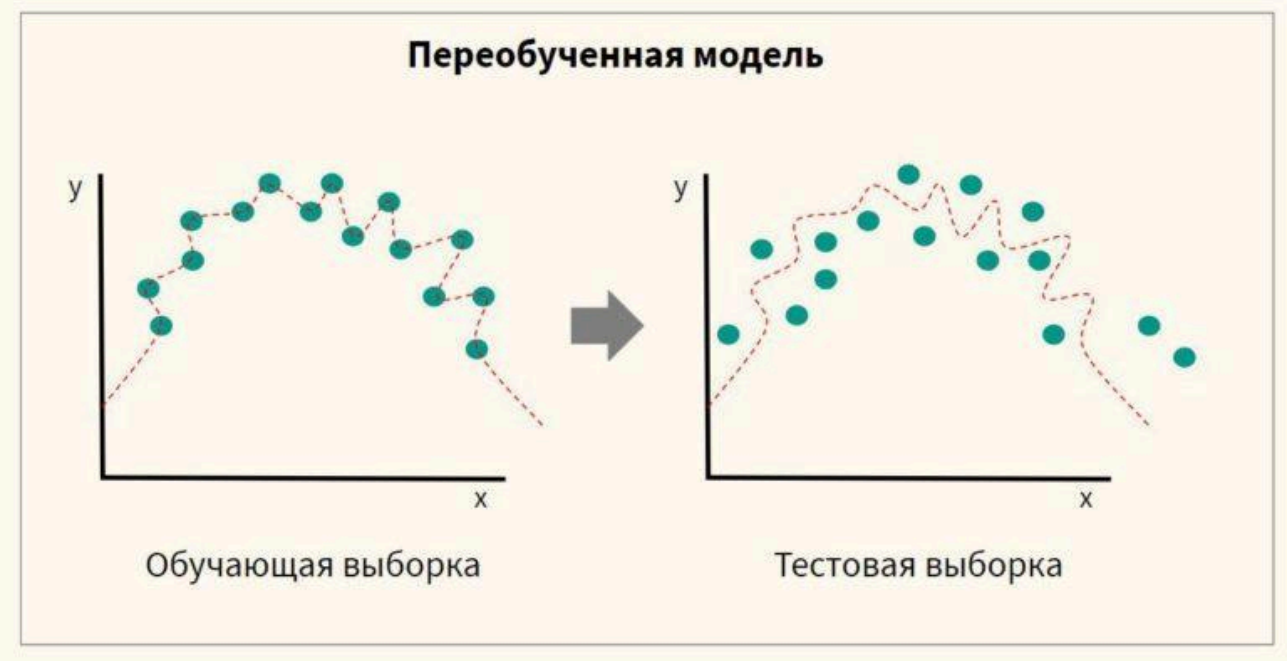
Кросс-валидация — это метод оценки производительности модели машинного обучения на основе доступных данных. Он заключается в разбиении исходного набора данных на обучающую выборку и тестовую выборку несколько раз.

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.

Переобучение и недообучение модели

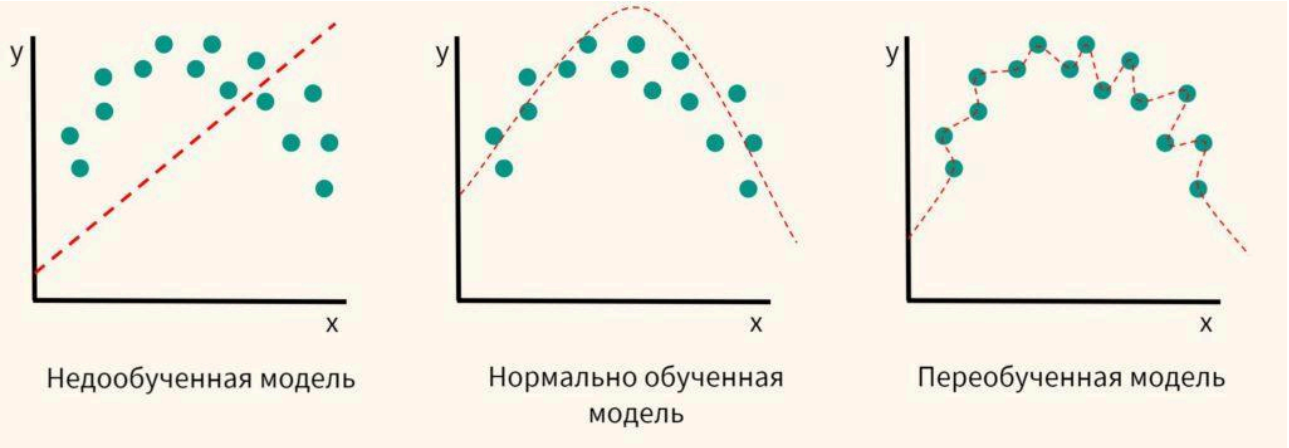
Переобучение (overfitting) и недообучение (underfitting) — это два проблемных состояния в машинном обучении, которые могут возникать при построении моделей.

Переобучение (overfitting) происходит, когда модель демонстрирует высокую точность на обучающей выборке, но показывает низкую производительность на тестовых данных. В этом случае модель "запоминает" или "подстраивается" под шумы и несущественные детали в обучающих данных, что приводит к плохой способности обобщения. Переобучение часто проявляется в чрезмерной сложности модели и недостатке регуляризации.

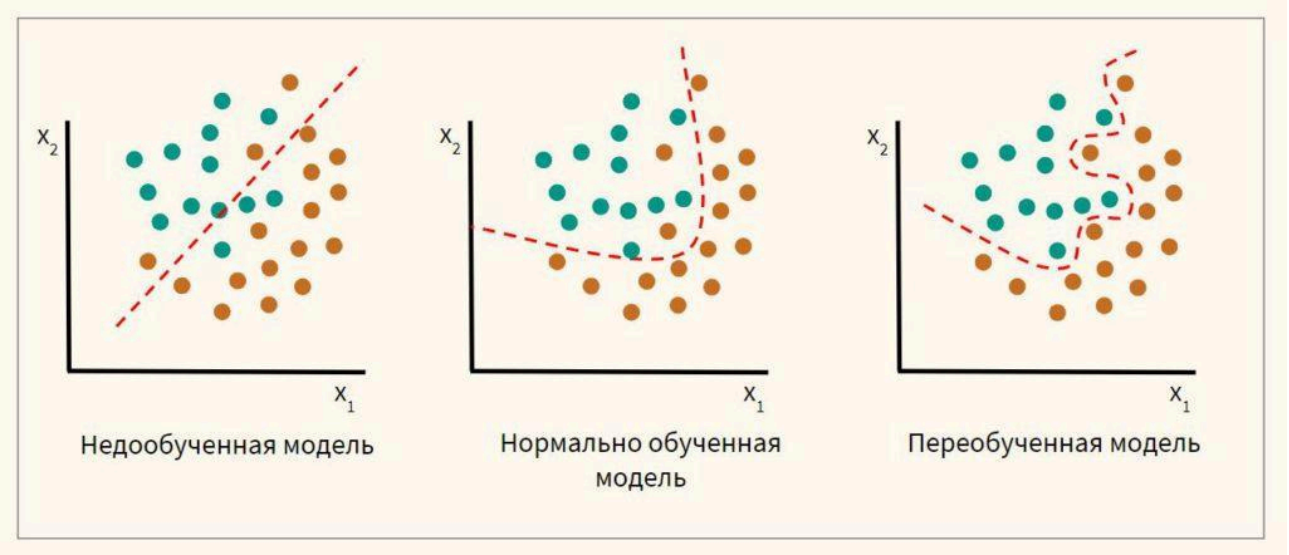


В случае недообучения (underfitting), наоборот, алгоритм не способен корректно определить зависимости на обучающей выборке. В этом случае модель не смогла извлечь основные закономерности или шаблоны в данных и дает недостаточно точные или неправильные предсказания. Недообучение проявляется в низкой точности модели на обучающей выборке и неспособности обобщать эти результаты на новые данные.

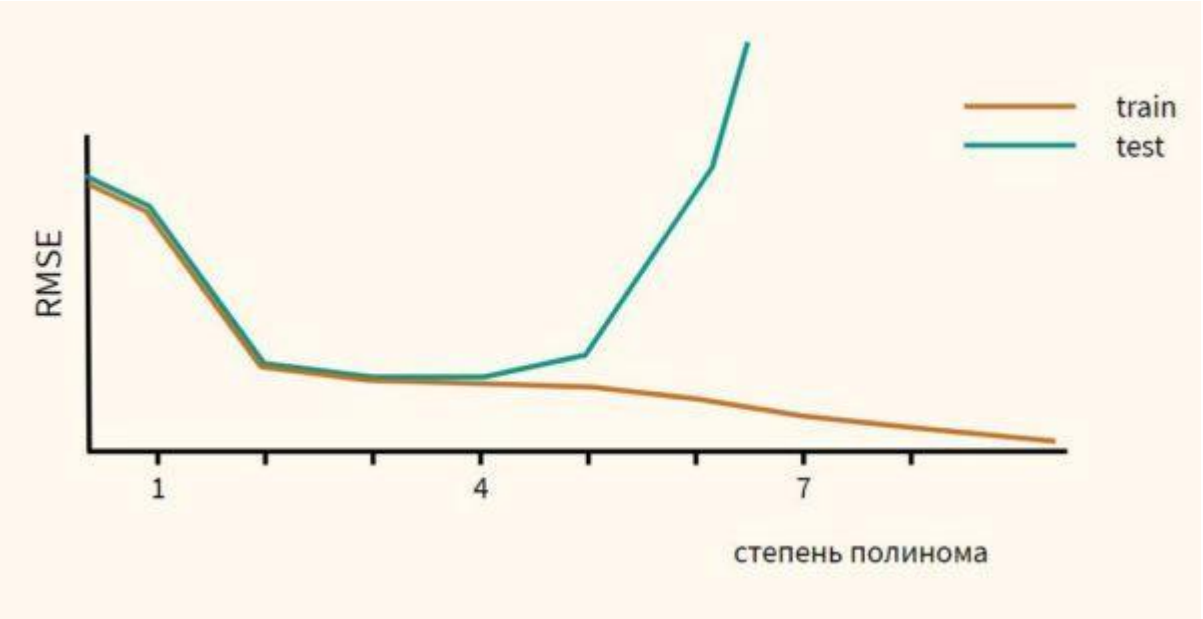
Идеальное состояние находится между переобучением и недообучением, когда модель способна хорошо обобщать данные и давать точные предсказания на невидимых ранее данных.



Также рассмотрим пример переобучения и недообучения в задаче классификации.



Если мы рассмотрим уровень ошибки на обучающих и тестовых данных, то можно представить эту информацию в виде графика. Предположим, что мы использовали полиномиальную регрессию и изучили значения функции потерь на обучающих и тестовых данных для разных степеней регрессии.



Заметим, что после определенного уровня сложности модели ошибка на тестовых данных начинает возрастать. Мы можем рассмотреть этот вопрос с разных точек зрения и ввести несколько новых терминов.
Сложность модели может быть причиной переобучения или недообучения. Сложность в данном контексте определяется типом используемых алгоритмов, их способностью выявлять линейные или нелинейные зависимости, а также количеством признаков и другими факторами.

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

Например, полиномиальная модель, особенно с высокой степенью полинома, часто может уловить более сложную зависимость, чем линейная регрессионная модель. Если модель хорошо описывает исходные данные, то говорят, что у нее низкое смещение.

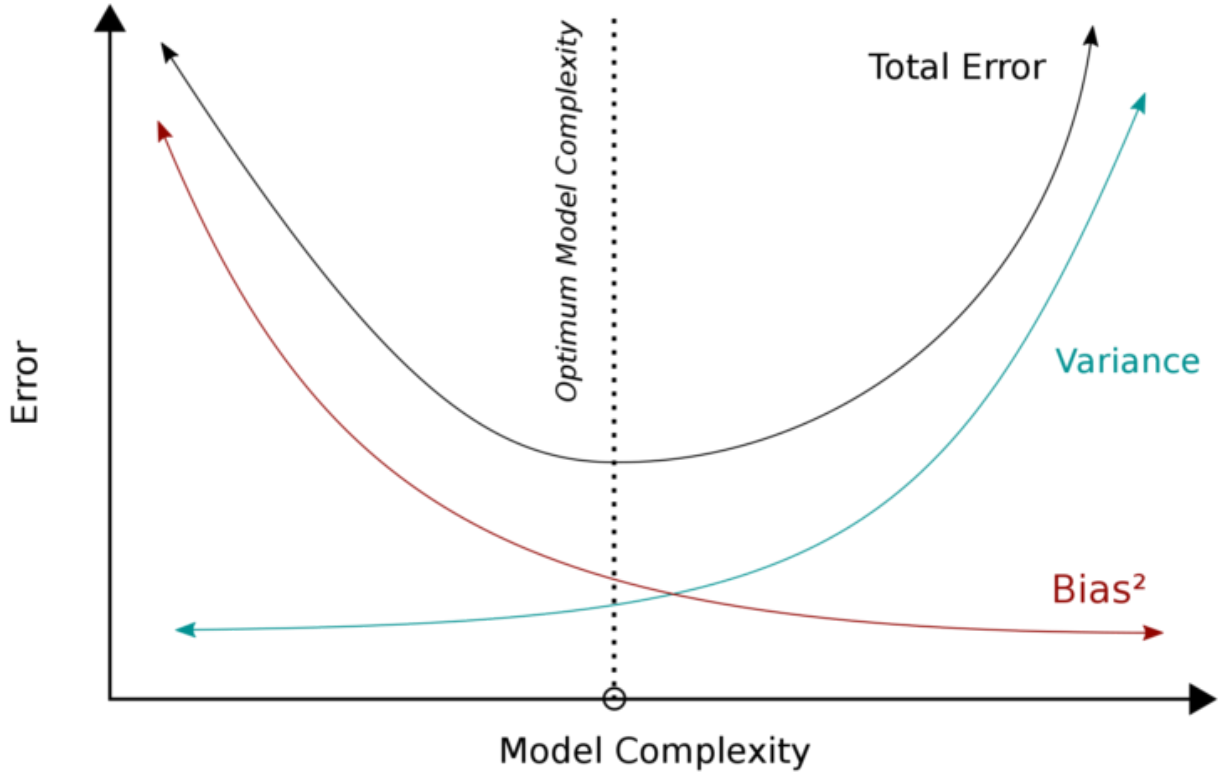
Снова про смещение и разброс

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

Другими словами, смещение показывает, насколько хорошо модель обучена на имеющихся данных (настроена на выборку). Однако как мы уже упомянули в самом начале, идеально обученная (переобученная) модель может иметь минимальное смещение и при этом показывать слабые результаты на тестовых данных. В таком случае говорят о большом разбросе.

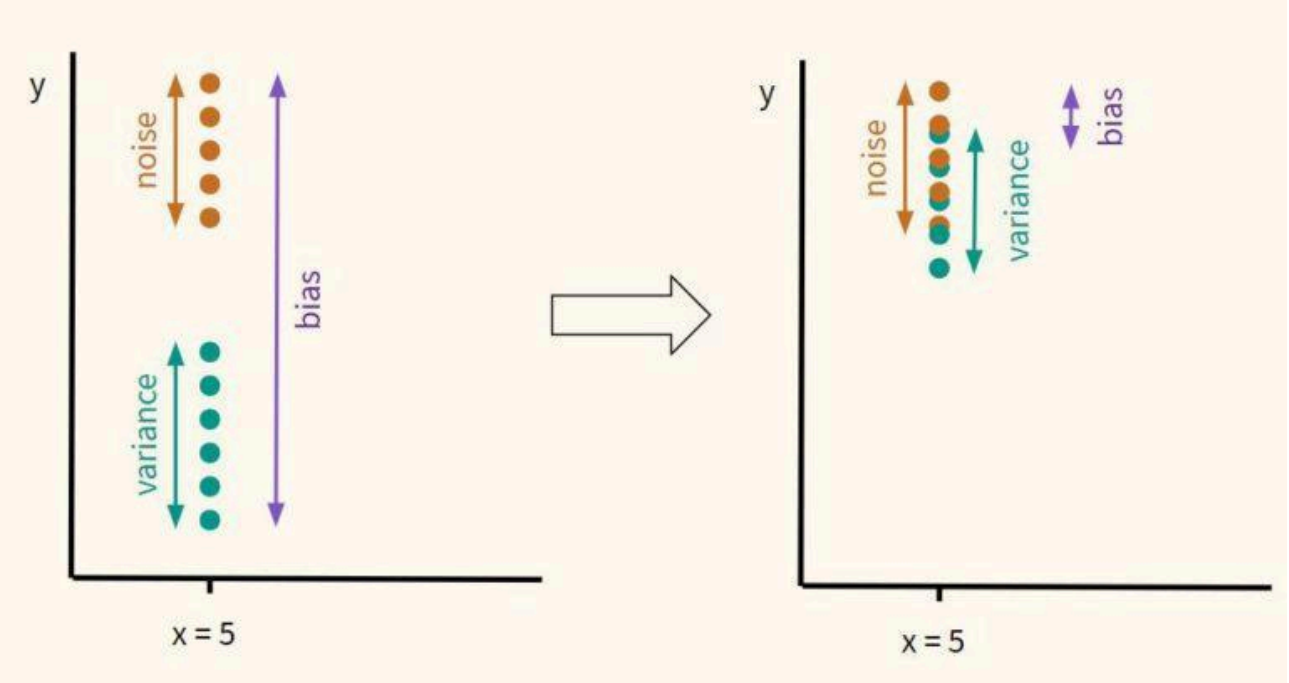
Разброс (variance) показывает, насколько результаты модели на обучающей выборке отличаются от результатов модели на тестовой выборке. То есть предсказания на тесте (например, на нескольких выборках) имеют разброс, и нам важно, чтобы эти результаты не слишком отличались (тогда можно сказать, что у модели хорошая способность к обобщению).

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). Ниже показан рисунок, иллюстрирующий эту проблему.



Давайте рассмотрим другой пример, чтобы лучше понять смещение и разброс. На левом графике мы изображаем модели (каждая точка на графике представляет прогноз одной модели), которые не очень хорошо описывают данные. Допустим, мы пытаемся описать зависимость с помощью полинома первой степени, хотя она лучше описывается квадратичной функцией.

Давайте оценим прогнозы этих моделей в точке x = 5.



Ошибка в прогнозировании может быть разложена на смещение, разброс/дисперсию и шум. "Правильные" ответы, которые мы хотели бы получить, обозначены коричневым цветом. Мы не можем свести их в одну точку, так как даже в идеальном случае мы получаем прогноз с точностью до шума (случайных колебаний). Смещение (фиолетовая стрелка) отражает ошибку, связанную с выбором неправильной модели (степень полинома). Разброс (зеленый цвет) - это различные прогнозы моделей на разных обучающих выборках.

На правом графике мы смогли значительно улучшить качество прогнозов благодаря правильно подобранным (например, полином второй степени, снижает смещение) и хорошо настроенным (снижает разброс) моделям.

Как бороться с переобучением?

У модели, которая сложная и переобучена, может быть большое количество признаков, особенно если речь идет о полиноме.

Очевидно, в большинстве случаев невозможно визуально подобрать степень полинома (как мы делали ранее для линейной и логистической полиномиальной

регрессии). Более того, модели с таким количеством признаков, из-за их высокой сложности, склонны к переобучению.

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

Теперь рассмотрим, пожалуй, основной и самый универсальный метод борьбы с переобучением – регуляризацию

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.

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

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

- Регуляризация L2
- Регуляризация L1
- Эластичная сеть

Давайте подробно обсудим эти стандартные методы.

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.

Линейная регрессия, использующая метод регуляризации L2, называется гребневой регрессией. Другими словами, в регрессионной регрессии к функции затрат линейной регрессии добавляется член регуляризации, который сохраняет величину весов (коэффициентов) модели как можно меньшей. Метод регуляризации L2 пытается поддерживать веса модели близкими к нулю, но не равными нулю, что означает, что каждая функция должна оказывать минимальное влияние на выходные данные, в то время как точность модели должна быть как можно выше.

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

Функция стоимости Гребневой регрессии Функция потерь

Где λ управляет степенью регуляризации и ω_j это веса (коэффициенты) модели. Путем увеличения λ, модель становится более совершенной и недостаточно приспособленной. С другой стороны, уменьшая λ модель становится более приспособленной, а с $\lambda = 0$, условие регуляризации будет исключено.

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

Регрессия с наименьшей абсолютной усадкой и оператором выбора (лассо) является альтернативой ridge для регуляризации линейной регрессии. Регрессия Лассо также добавляет штрафной член к функции затрат, но немного по-другому, называемый регуляризацией L1. Регуляризация L1 обнуляет некоторые коэффициенты, что означает, что модель будет игнорировать эти особенности. Игнорирование наименее важных функций помогает подчеркнуть существенные особенности модели.

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

Функция стоимости Регрессии Лассо = функция потерь

Где λ управляет степенью регуляризации и ω_j это веса (коэффициенты) модели.

Регрессия лассо автоматически выполняет выбор функций, устраняя наименее важные функции.

L1-регуляризация может быть полезна для устранения неинформативных признаков и автоматического отбора признаков, тогда как 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.

Параметр регуляризации Elastic Net позволяет управлять вкладом каждой регуляризации в модель. Если параметр регуляризации L1 равен нулю, Elastic Net сводится к L2 регуляризации, а если параметр регуляризации L2 равен нулю, Elastic Net сводится к L1 регуляризации.

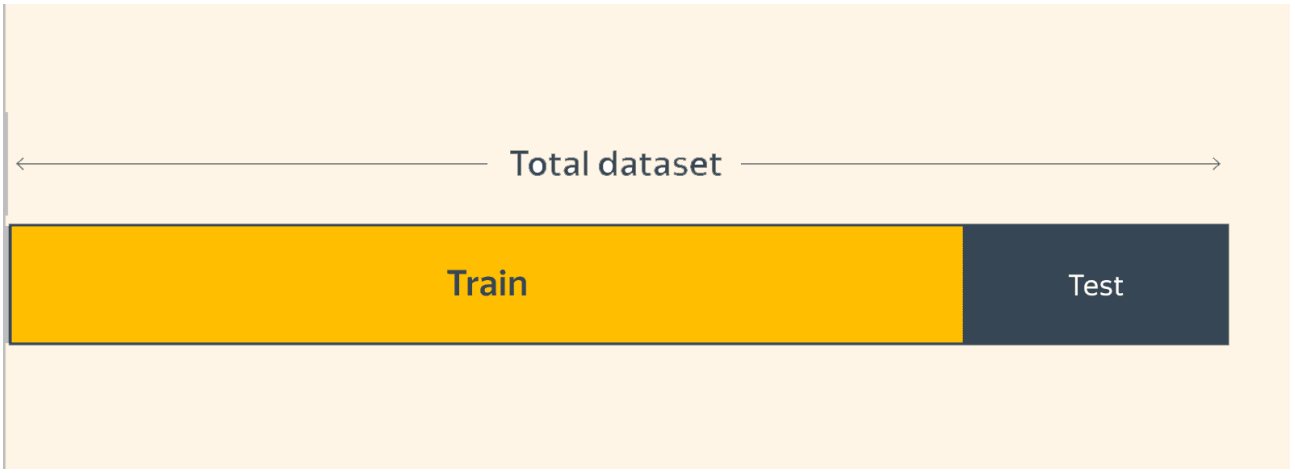
Elastic Net широко используется в задачах регрессии и отбора признаков для создания моделей, которые лучше справляются с мультиколлинеарностью и могут быть более стабильными при наличии шумовых или избыточных признаков.

Кросс-валидация

Кросс-валидация является методом, который используется в машинном обучении для оценки качества работы модели. Его широко применяют для сравнения разных моделей и выбора наилучшей для определенной задачи.

Hold-out

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



Для оценки модели требуется обучить ее на тренировочном наборе данных и затем измерить результаты на тестовом наборе. В библиотеке sklearn по умолчанию параметр shuffle (перемешать) установлен в значение True, что означает, что перед разделением на тренировочный и тестовый наборы происходит перемешивание образцов (для воспроизводимости такого разделения необходимо задать random_state (случайное состояние). Для чего нужно перемешивание данных?

```
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
)
```

Данный простой пример показывает, как отсутствие перемешивания данных может сильно обмануть вас. Предположим, у вас есть большой набор данных из миллиона изображений кошек и собак, и вы хотите обучить модель их различать. Допустим, исходный порядок тренировочных данных таков: первые полмиллиона изображений - кошки, а затем следуют изображения собак. В этом случае модель в первой половине обучения запомнит, что на изображении всегда кошка, а затем забудет то, что она учила на первом этапе и всегда будет предсказывать собаку. При этом она не будет использовать сами данные для предсказания.

Если вы рассматриваете различные модели для решения вашей задачи, то рекомендуется оптимизировать их качество на валидационном наборе данных, а окончательное сравнение моделей проводить на тестовом наборе данных. Оптимизация качества модели может включать подбор гиперпараметров, выбор архитектуры (в случае нейронных сетей), подбор оптимального порога для максимизации значения целевой метрики (например, при двух классовой классификации, когда модель выдает непрерывные значения от 0 до 1, которые нужно бинаризовать для достижения максимального значения F1-меры) и т.д. Если оптимизировать и сравнивать модели на одном и том же наборе данных, то можно неявно включить информацию о тестовом наборе данных в модель и получить результаты, которые будут хуже ожидаемых на новых данных.

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. Это можно проиллюстрировать на примере разделения датасета Iris на тренировочное и тестовое множества. В исходном датасете классы распределены равномерно.

- 33.3 Setosa
- 33.3 Versicolor
- 33.3 Virginica

Случайное разбиение, в котором две трети цветов (100) отправились в трейн, а оставшаяся треть (50) отправилась в тест, может выглядеть, например, так:

- трейн: 38 x Setosa, 28 x Versicolor, 34 x Virginica (распределение 38)
- тест: 12 x Setosa, 22 x Versicolor, 16 x Virginica (распределение 24)

Если цвета в исходном датасете распределены так же, как в природе, то мы только что получили два новых датасета, которые не соответствуют распределению цветов в природе. Оба датасета имеют несбалансированные и разные распределения: самый частый класс в тренировочном наборе соответствует наименее частому классу в тестовом наборе.

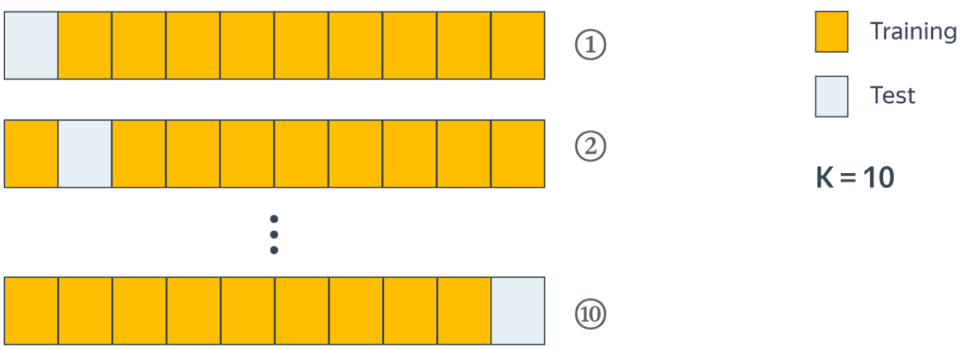
```
import numpy as np
from sklearn.model_selection import train_test_split

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
)
```

В такой ситуации стратификация может быть полезна: это разделение на тренировочный и тестовый наборы, которое сохраняет соотношение классов, представленное в исходном датасете. В библиотеке sklearn такое разделение можно получить с помощью параметра stratify.
В случае работы с большими датасетами (по крайней мере, порядка 10 тысяч семплов) и сбалансированными классами, можно не сильно беспокоиться о ранее описанной проблеме и использовать обычное случайное разделение данных. Однако, если у вас имеются сильно несбалансированные данные, в которых один класс встречается гораздо чаще другого (как, например, в задачах фильтрации спама или сегментации осадков на спутниковых снимках), стратификация может значительно помочь.

k-Fold

Когда говорят о кросс-валидации, чаще всего имеют в виду метод 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.



Интересный вопрос заключается в выборе модели для сравнения с другими на отложенном тестовом наборе данных (если таковой имеется) или для окончательного использования в задаче. После применения метода k-Fold для одной модели у вас будет несколько экземпляров (инстансов) этой модели, обученных на различных подмножествах тренировочных данных. Возможные варианты:

```
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]
'''
```

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. ])
'''
```

Выбор наилучшего метода зависит от конкретной задачи и доступных вычислительных возможностей. Метод k-Fold обеспечивает более надежную оценку качества модели по сравнению с hold-out, так как обучение и тестирование модели происходят на разных подмножествах исходного датасета. Однако проведение итераций обучения и тестирования может быть ресурсоемким, поэтому этот метод обычно используется, когда данных немного или имеется большое количество вычислительных ресурсов, позволяющих выполнять все итерации параллельно. В реальных задачах обычно имеется достаточно данных, чтобы hold-out давал хорошую оценку качества модели, поэтому метод k-Fold редко применяется в больших задачах.

Leave-one-out

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

Данный подход может быть полезен, если у вас имеется очень ограниченное количество данных (например, при сегментации клеток на изображениях с помощью оптического микроскопа) и вы стремитесь максимально использовать их для обучения модели. Для валидации на каждой итерации этому методу требуется всего один образец. Однако количество итераций будет равно количеству образцов в данных, поэтому данный метод не применим для средних и больших задач.

```python
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. Этот подход может быть полезен в случае, когда классы в исходном наборе данных сильно несбалансированы. При обычном случайном разделении некоторые фолды могут либо не содержать семплов определенных классов, либо содержать их в недостаточном количестве.

```python
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]
'''
```

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.

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.

Что можно почитать еще?

1. Как случайное разбиение данных может испортить ML-модель.
2. Кросс-валидация
3. Фундаментальные концепции переобучения и недообучения в машинном обучении

Используемая литература

1. https://www.helenkapatsa.ru/pierieobuchieniie/
2. https://habr.com/ru/companies/ods/articles/323890/
3. https://datascience.eu/ru/машинное-обучение/регуляризация-в-машинном-обучении/