Конспект по теме "Качество модели"

Случайность в алгоритмах обучения

Чтобы модели лучше замечали в данных зависимости, во многие алгоритмы машинного обучения добавляется случайность. По-настоящему случайные числа компьютер не создаёт. Он подключает **генераторы псевдослучайных чисел**, которые производят последовательности, похожие на случайные.

Со случайными числами не всё так просто: они непредсказуемы. Генераторы псевдослучайных чисел можно настроить так, чтобы результаты неизменно получались одинаковыми. Зафиксировать псевдослучайность для алгоритма обучения очень просто: при его создании нужно указать параметр *random_state*.

```
# указываем случайное состояние (число)
model = DecisionTreeClassifier(random_state=12345)

# обучаем модель как раньше
model.fit(features, target)
```

Если вы укажете `random_state=None` (по умолчанию), то псевдослучайность будет разной всегда.

Тестовый набор данных

Чтобы знать точно, что модель не вызубрила ответы, возьмём новый датасет — **тестовый набор данных**, или **тестовую выборку**.

Доля правильных ответов

Отношение числа правильных ответов к размеру тестовой выборки называется **accuracy**, в некоторых переводах — «доля правильных ответов». Формула выглядит так:

Метрики качества

Метрики качества оценивают качество работы и выражаются в числовой форме. Вы уже знакомы с одной метрикой качества — *accuracy*. Есть и другие, например:

- **точность** (*precision*) показывает, какая доля объектов, для которых модель предсказала ответ «1», действительно имеют ответ «1»;
- **полнота** (*recall*) выявляет, какую часть объектов, имеющих ответ «1», выделила модель.

Всегда сравнивайте модель со случайной, так вы сможете оценить её адекватность, или **проверите на вменяемость** (*sanity check*).

Метрики качества в библиотеке sklearn

В библиотеке *sklearn* метрики находятся в модуле *sklearn.metrics*. Вычисляется *accuracy* функцией *accuracy_score()*

```
from sklearn.metrics import accuracy_score
```

Функция принимает на вход два аргумента: 1) правильные ответы, 2) предсказания модели. Возвращает она значение *accuracy*.

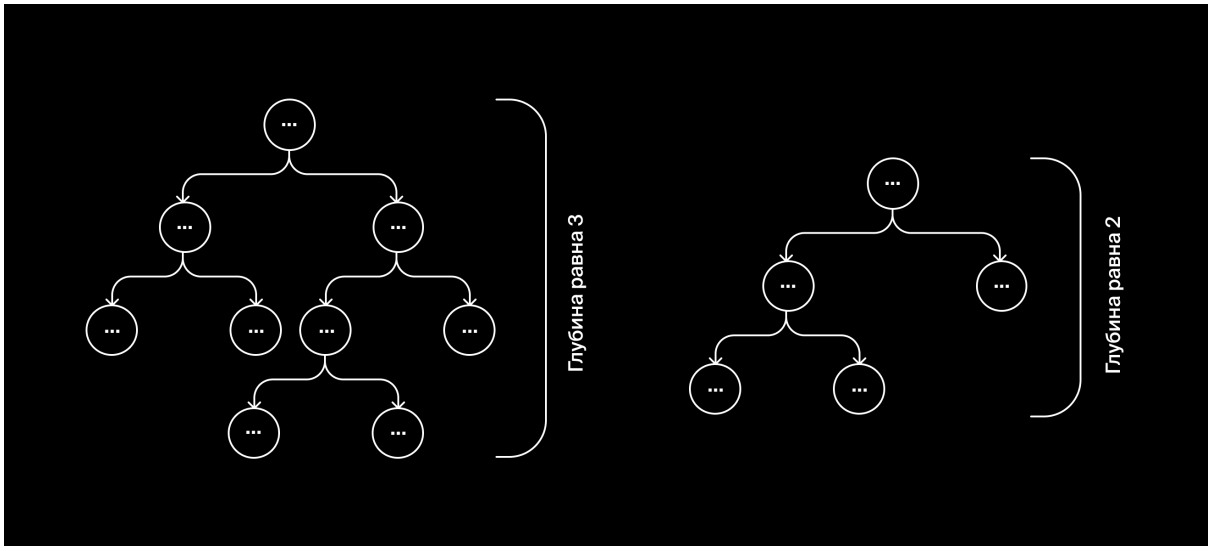
```
accuracy = accuracy_score(target, predictions)
```

Переобучение и недообучение

Модель хорошо объясняла примеры из обучающего набора данных, но запуталась в тестовой выборке и не смогла ответить правильно? Вы столкнулись с проблемой **переобучения.** Обратный эффект — **недообучение**. Оно возникает, когда качество на обучающей и тестовой выборках примерно одинаковое, причём низкое.

Не всегда удаётся избежать переобучения или недообучения. Когда избавитесь от первого, увеличивается риск появления второго эффекта, и наоборот.

Посмотрите на пример настройки алгоритма обучения. Как она влияет на баланс между переобучением и недообучением? **Глубина дерева** (**высота дерева**) — это максимальное количество условий от «вершины» до финального ответа. Считается по количеству переходов между узлами.



Глубина дерева в *sklearn* задаётся параметром *max_depth*:

```
# укажите глубину (по умолчанию не ограничена)
model = DecisionTreeClassifier(random_state=12345, max_depth=3)

model.fit(features, target)
```

Эксперименты с решающим деревом

Чтобы сохранить обученную модель в нужном формате, примените функцию библиотеки *joblib* — *dump* (англ. «сбрасывать, сваливать»).

```
# Сохранение обученной модели
# первый аргумент - модель
# второй аргумент - путь к файлу

from joblib import dump

joblib.dump(model, 'model.joblib')
```

Открыть и запустить модель можно функцией *load*.

```
import joblib

# аргумент - путь к файлу
# возращаемое значение - модель
model = joblib.load('model.joblib')
```