Конспект по теме "Улучшение модели"

Валидационная выборка

Чтобы оценка качества была более надёжной, нужно подготовить новую выборку — **валидационную** (англ. *validation*, «проверка»), или **проверочную**. Валидационная выборка отбирается из исходного датасета ещё до обучения модели. Иначе, обучившись на тренировочном наборе, модель будет знать все ответы. Именно валидация подсказывает, как ведёт себя модель в полевых условиях и нет ли переобучения.

Какую часть данных отвести под валидационную выборку, решают в зависимости от количества объектов, признаков и вариативности данных. Вот два самых распространённых сценария:

1. Доступен исходный датасет, а тестовая выборка спрятана. Тогда рекомендуется 75% данных отвести под обучающую, а 25% — под валидационную. Соотношение 3:1.
2. Спрятанной тестовой выборки нет. Значит, данные нужно разбить на три части: обучающую, валидационную и тестовую. Размеры тестового и валидационного наборов обычно равны. Исходные данные разбивают в соотношении 3:1:1.

Деление на две выборки

В *sklearn* для извлечения валидационной выборки используется функция **train_test_split**. Она разбивает любой датасет на обучающую и тестовую выборки (в нашем случае, для разделения на обучающую и валидационную):

```
from sklearn.model_selection import train_test_split
```

Прежде чем разделить набор данных, нужно указать два параметра:

- **Название набора**, данные которого делим;

- **Размер валидационной выборки** (*test_size*). Выражается в долях — от 0 до 1.

Функция *train_test_split()* возвращает два новых набора данных — обучающий и валидационный.

```
df_train, df_valid = train_test_split(df, test_size=0.25, random_state=12345)
```

В *random_state* можно указать любое число, главное не *None*.

Гиперпараметры

Модель машинного обучения работает с **параметрами**, которые она определяет во время обучения на основе тренировочных данных.

Помимо обычных параметров, есть ещё **гиперпараметры** — настройки алгоритмов обучения. Гиперпараметры помогают улучшить модель. Изменить их можно до начала обучения.

У алгоритма решающего дерева есть следующие гиперпараметры:

- **max_depth** — максимальная глубина дерева;
- **criterion** — критерий разделения;
- **min_samples_split** — минимальное число объектов в узле дерева, достаточное для дальнейшего разделения;
- **min_samples_leaf** — минимальное число объектов в **листьях** дерева — нижних узлах с ответами.

Новые модели: случайный лес

Один из алгоритмов классификации — **случайный лес** (*random forest*). Алгоритм обучает большое количество независимых друг от друга деревьев, а потом принимает решение на основе голосования. Случайный лес помогает улучшить результат предсказания и избежать переобучения. В библиотеке *sklearn* алгоритм случайного леса **RandomForestClassifier** находится в модуле *sklearn.ensemble*:

```
from sklearn.ensemble import RandomForestClassifier
```

Чтобы управлять количеством деревьев в лесу, пропишем гиперпараметр *n_estimators*. Чем больше деревьев, тем дольше модель будет учиться, но результат станет лучше (и наоборот):

```
model = RandomForestClassifier(random_state=12345, n_estimators=3)
```

Логистическая регрессия

Если сделать гиперпараметр *n_estimators* больше, модель начнёт разрастаться и медленно обучаться. Это плохо. Мало деревьев и результаты не лучше. Тоже плохо.

Ещё один алгоритм машинного обучения — **логистическую регрессию.** Если название и «мимикрирует» под задачу регрессии, всё-таки это алгоритм классификации. Логистическая регрессия:

- Сначала считает, к какому классу близок объект
- В зависимости от ответа выбирает нужный класс: если результат вычисления положительный, то — «1»; отрицательный — «0».

В логистической регрессии параметров мало. Что-либо вызубрить по признакам в формуле не выйдет, поэтому и вероятность переобучения невелика.

Модель **LogisticRegression** лежит в модуле *sklearn.linear_model* библиотеки *sklear*:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=12345)
```