Конспект по теме "Подготовка признаков"

Прямое кодирование

Преобразовать категориальные признаки в численные поможет техника **прямого кодирования**, или **отображения** (One-Hot Encoding).

Техникой *OHE* категориальные признаки переводятся в численные в два этапа:

- 1. Для каждого значения признака создаётся новый столбец;
- 2. Если объекту категория подходит, присваивается 1, если нет 0.

Новые признаки называются **дамми-переменными**, или **дамми- признаками**.

Для прямого кодирования в библиотеке pandas есть функция pd.get_dummies().

```
pd.get_dummies(df['column'])
```

Дамми-ловушка

Когда данных в избытке, можно угодить в ловушку фиктивных признаков. Если в таблицу добавились три новых столбца, сильно связанных между собой, один из них можно не добавлять. Восстановить столбец можно по оставшимся двум. Так мы не попадём в **дамми-ловушку**.

Столбец уберём вызовом функции pd.get_dummies() с аргументом drop_first. Он удаляет первую колонку и передаётся как drop_first=True или drop_first=False (True — первый столбец сбрасывается, False — не сбрасывается).

```
pd.get_dummies(df['column'], drop_first=True)
```

Порядковое кодирование

Ещё одна техника, чтобы закодировать цифрами выраженные в тексте категории — **Ordinal Encoding**. Она работает так:

- 1. Фиксируется, какой цифрой кодируется класс;
- 2. Цифры размещаются в столбце.

Чтобы выполнить кодирование, в *sklearn* есть структура данных **OrdinalEncoder**. Она находится в модуле **sklearn.preprocessing**:

```
from sklearn.preprocessing import OrdinalEncoder
```

Преобразование выполняется в три этапа:

1. Создаём объект этой структуры данных.

```
encoder = OrdinalEncoder()
```

2. Чтобы получить список категориальных признаков, вызываем метод *fit()* — как и в обучении модели, передаём ему данные как аргумент.

```
encoder.fit(data)
```

3. Преобразуем данные функцией transform():

```
data_ordinal = encoder.transform(data)
```

Чтобы код добавил названия столбцов, оформим данные в структуру DataFrame():

Если преобразование признаков требуется лишь один раз, как в нашей задаче, код можно упростить вызовом функции **fit_transform()**. Она объединяет функции: *fit()* и *transform()*.

Масштабирование признаков

Если в данных присутствуют колличественные признаки с разными разбросами значений, то алгорим может решить, что признаки с большими значениями и разбросом важнее. Чтобы избежать этой ловушки, **признаки масштабируются** — приводятся к одному масштабу.

Один из методов масштабирования — **стандартизации данных**. Предположим, что все признаки распределены нормально, среднее (*M*) и дисперсия (*D*) определяются по выборке. Значения признака преобразуются по формуле:

```
egin{aligned} \mathsf{H}\mathsf{o}\mathsf{b}\mathsf{o}\mathsf{e}\;\mathsf{3}\mathsf{h}\mathsf{a}\mathsf{v}\mathsf{e}\mathsf{h}\mathsf{u}\mathsf{e} &= \dfrac{\mathsf{C}\mathsf{T}\mathsf{a}\mathsf{p}\mathsf{o}\mathsf{e}\;\mathsf{3}\mathsf{h}\mathsf{a}\mathsf{v}\mathsf{e}\mathsf{h}\mathsf{u}\mathsf{e} &= \dfrac{\mathsf{M}}{\sqrt{\mathsf{D}}} \end{aligned}
```

У нового признака устанавливается среднее, равное 0, и дисперсия, равная 1.

В sklearn есть отдельная структура для стандартизации данных — StandardScaler. Он находится в модуле sklearn.preprocessing:

```
from sklearn.preprocessing import StandardScaler
```

Создадим объект этой структуры и настроим его на обучающих данных. Настройка — это вычисление среднего и дисперсии:

```
scaler = StandardScaler()
scaler.fit(df)
```

Преобразуем обучающую и валидационную выборки функцией *transform()*.

df_scaled = scaler.transform(df)