Масштабирование данных

Масштабирование данных необходимо для того, чтобы привести числовые признаки (фичи) к единому, сопоставимому диапазону значений. Это важно для многих алгоритмов машинного обучения и анализа данных, поскольку признаки с разным масштабом могут влиять на модель неравномерно и ухудшать её качество.

Основные причины и цели масштабирования:

– уравнивание влияния признаков: если в данных есть признаки с разными порядками величин (например, возраст от 0 до 100 и доходы от 0 до 1,000,000), то без масштабирования алгоритмы могут отдавать слишком большой вес признакам с большими значениями, что исказит результаты. Масштабирование выравнивает значения и позволяет каждой переменной вносить сопоставимый вклад в обучение модели;

– повышение скорости и стабильности обучения: Многие алгоритмы оптимизации, например, градиентный спуск, работают быстрее и стабильнее, когда входные признаки находятся в одном масштабе (пример — стандартизация с нулевым средним и единичным стандартным отклонением);

– адекватные вычисления расстояний: Алгоритмы, основанные на вычислении расстояний между точками (например, k-ближайших соседей или метод опорных векторов), сильно зависят от масштаба признаков. Без масштабирования признаки с большими значениями будут доминировать при вычислении расстояний, что исказит результаты;

– снижение влияния выбросов: Некоторые методы масштабирования (например, стандартизация) помогают уменьшить влияние выбросов и нормализуют данные по смыслу.

Часто используемые методы масштабирования:

– Стандартизация (StandardScaler): приводит данные к распределению с нулевым средним и стандартным отклонением 1.

– Нормализация (Min-Max Scaling): сжимает данные в интервал, например, от 0 до 1.

– Другие способы масштабирования: логарифмирование, масштабирование с учётом квантилей и прочее.

Масштабирование — это важный этап предобработки данных, который обеспечивает корректное и эффективное обучение моделей, помогает избежать смещения и улучшает качество предсказаний.

Это общепризнанный и необходимый шаг в задачах машинного обучения и анализа данных. Разберем следующий пример. Обучим модель для регрессии из sklearn на датасете про недвижимость.

Исходные данные показаны на Рисунок 1. Рисунок 3

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0000	6.9841	1.0238	322.0000	2.5556	37.8800	-122.2300
1	8.3014	21.0000	6.2381	0.9719	2401.0000	2.1098	37.8600	-122.2200
2	7.2574	52.0000	8.2881	1.0734	496.0000	2.8023	37.8500	-122.2400
3	5.6431	52.0000	5.8174	1.0731	558.0000	2.5479	37.8500	-122.2500
4	3.8462	52.0000	6.2819	1.0811	565.0000	2.1815	37.8500	-122.2500

Рисунок 1 – Исходные данные для задачи

Обучим 3 модели без масштабирования данных: kNN, линейную регрессию и дерево решений. Сравним метрики качества моделей (рис.2-4).

```
knn = KNeighborsRegressor()
knn.fit(X_train, y_train)
pred_train = knn.predict(X_train)
pred_test = knn.predict(X_test)

print(f'Train R2 {r2_score(y_train, pred_train):.2f}')
print(f'Test R2 {r2_score(y_test, pred_test):.2f}')
```

```
Train R2 0.45
Test R2 0.18
```

Рисунок 2 – Модель kNN

```
lr = LinearRegression()
lr.fit(X_train, y_train)
pred_train = lr.predict(X_train)
pred_test = lr.predict(X_test)

print(f'Train R2 {r2_score(y_train, pred_train):.2f}')
print(f'Test R2 {r2_score(y_test, pred_test):.2f}')
```

```
Train R2 0.61
Test R2 0.60
```

Рисунок 3 – Модель линейной регрессии

```python
# Обучим дерево решений
tree_1 = DecisionTreeRegressor(random_state=1, max_depth=9)
tree_1.fit(X_train, y_train)
pred_train = tree_1.predict(X_train)
pred_test = tree_1.predict(X_test)


print(f'Train R2 {r2_score(y_train, pred_train):.2f}')
print(f'Test R2 {r2_score(y_test, pred_test):.2f}')
```

```
Train R2 0.79
Test R2 0.72
```

Рисунок 4 – Модель дерева решений

По результатам моделирования видно, что модель дерева решений на данных обучилась, соответственно, можно сделать вывод, что проблема того, что метрики у двух других моделей низкие состоит не в данных, а в моделях (рис. 5). Изучив данные, можно увидеть, что присутствуют признаки с маленькими значениями (`MedInc`, `AveBedrms`), и присутствуют признаки со значениями побольше (`Latitude`, `HouseAge`), и есть признаки с очень большими значениями (`Population` и `AveOccup`). Значения в признаках отличаются на порядки и это является огромной проблемой для чувствительных моделей.

Имея такие данные, модель может ошибочно считать, что если признак большой (`Population`), то он является полезным, но признаки с небольшими значениями (`MedInc`) могут оказаться в разы полезней больших. А вот такие большие будут перетягивать на себя всё внимание модели.

```python
X.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000

Рисунок 5 – Статистики числовых признаков датасета

Нормализация данных. Для того, чтобы сделать нормализацию данных нужно найти в каждом признаке его минимум (min) и максимум (max), а затем сделать следующее вычисление:

$$x = \frac{x - min}{max - min}$$

После такого преобразования min=0,max=1.
Обучим sklearn версию (рис.6)

```
mms = MinMaxScaler()
mms.fit(X_train)
#  номализуем обучающую выборку и обернем ее в DataFrame
X_train_norm = pd.DataFrame(mms.transform(X_train), columns=X_train.columns)
X_train_norm.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
count	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000
mean	0.2320	0.5414	0.0325	0.0227	0.0399	0.0039	0.3294	0.4754
std	0.1304	0.2476	0.0184	0.0149	0.0318	0.0107	0.2273	0.1998
min	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
25%	0.1420	0.3333	0.0254	0.0199	0.0219	0.0029	0.1477	0.2540
50%	0.2091	0.5490	0.0310	0.0212	0.0326	0.0035	0.1828	0.5827
75%	0.2924	0.7059	0.0369	0.0227	0.0484	0.0043	0.5505	0.6315
max	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Рисунок 6 – Нормализованные данные для обучения

Все минимальные значения везде равны 0, а максимальные - 1. Нормализуем тестовую выборку. При этом пользуемся min и max значениями с обучающей выборки, а не тестовой. Если использовать min и max тестовой выборки, то масштаб данных для обучения и для валидации будет разным, а нам необходимо все данные привести к масштабу выборки для обучения (рис.7).

```
X_test_norm = pd.DataFrame(mms.transform(X_test), columns=X_train.columns)
X_test_norm.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
count	4128.000000	4128.000000	4128.000000	4128.000000	4128.000000	4128.000000	4128.000000	4128.000000
mean	0.234496	0.544198	0.032414	0.022440	0.039816	0.004262	0.325244	0.478847
std	0.133330	0.243659	0.013532	0.010172	0.031321	0.032298	0.225655	0.198421
min	0.000000	0.000000	0.002015	0.003294	0.000056	0.000465	0.002125	0.004980
25%	0.143938	0.333333	0.025715	0.019910	0.022142	0.002904	0.148778	0.258715
50%	0.210390	0.549020	0.031240	0.021195	0.032680	0.003552	0.180659	0.584661
75%	0.294279	0.705882	0.036861	0.022743	0.047619	0.004325	0.548353	0.631474
max	1.000000	1.000000	0.436515	0.408432	0.451778	2.074450	0.988310	0.982072

Рисунок 7 – Тестовые данные после нормализации

Именно потому, что используем min и max из обучения, то чистый 0 в min и 1 в max не получили. Обучаем модели на нормализованных данных (рис.8-9).

```
# обучение модели knn на нормализованных данных
knn = KNeighborsRegressor()
knn.fit(X_train_norm, y_train)
pred_train = knn.predict(X_train_norm)
pred_test = knn.predict(X_test_norm)


print(f'Train R2 {r2_score(y_train, pred_train):.2f}')
print(f'Test R2 {r2_score(y_test, pred_test):.2f}')
```

```
Train R2 0.81
Test R2 0.70
```

Рисунок 8 – Модель kNN, обученная на нормализованных данных

```
lr = LinearRegression()
lr.fit(X_train_norm, y_train)
pred_train = lr.predict(X_train_norm)
pred_test = lr.predict(X_test_norm)


print(f'Train R2 {r2_score(y_train, pred_train):.2f}')
print(f'Test R2 {r2_score(y_test, pred_test):.2f}')
```

```
Train R2 0.61
Test R2 0.60
```

Рисунок 9 - Модель линейной регрессии, обученной на нормализованных данных

По рисунку 8 видно, что метрики значительно выросли, а метрики модели линейной регрессии остались неизменными. Это произошло по той причине, что для линейной регрессии больше подходит другой способ масштабирования данных – стандартизация, или, возможно, это все, на что способна выбранная модель.

Стандартизация. Для того, чтобы сделать стандартизацию данных нужно посчитать в каждом признаке его среднее значение (mean) и стандартное отклонение (std), а затем сделать следующее вычисление:

$$x = \frac{x - mean}{std}$$

После такого преобразования mean=0,std=1. Обучаем при помощи sklearn (рис.10).

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_std = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)
X_train_std.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
count	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000
mean	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000
std	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
min	-1.7784	-2.1870	-1.7662	-1.5260	-1.2524	-0.3634	-1.4491	-2.3792
25%	-0.6895	-0.8405	-0.3840	-0.1835	-0.5641	-0.0931	-0.7993	-1.1082
50%	-0.1749	0.0308	-0.0807	-0.0986	-0.2292	-0.0326	-0.6450	0.5366
75%	0.4634	0.6645	0.2402	0.0019	0.2676	0.0396	0.9726	0.7808
max	5.8885	1.8526	52.5662	65.7683	30.1505	92.8858	2.9501	2.6250

Рисунок 10 – Стандартизированные данные для обучения

Также применим стандартизацию признаков, но со статистиками данных для обучения, так как нам необходимо привести все данные к единому масштабу, к масштабу данных для обучения (рис.11).

```
X_train_std = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)
pd.set_option('display.float_format', lambda x: '%0.4f' % x)
X_train_std.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
count	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000	16512.0000
mean	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000
std	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
min	-1.7784	-2.1870	-1.7662	-1.5260	-1.2524	-0.3634	-1.4491	-2.3792
25%	-0.6895	-0.8405	-0.3840	-0.1835	-0.5641	-0.0931	-0.7993	-1.1082
50%	-0.1749	0.0308	-0.0807	-0.0986	-0.2292	-0.0326	-0.6450	0.5366
75%	0.4634	0.6645	0.2402	0.0019	0.2676	0.0396	0.9726	0.7808
max	5.8885	1.8526	52.5662	65.7683	30.1505	92.8858	2.9501	2.6250

Рисунок 11 – Стандартизированные данные для валидационной выборки

Обучим выбранные в самом начале модели и сравним полученные метрики (рис.12-14).

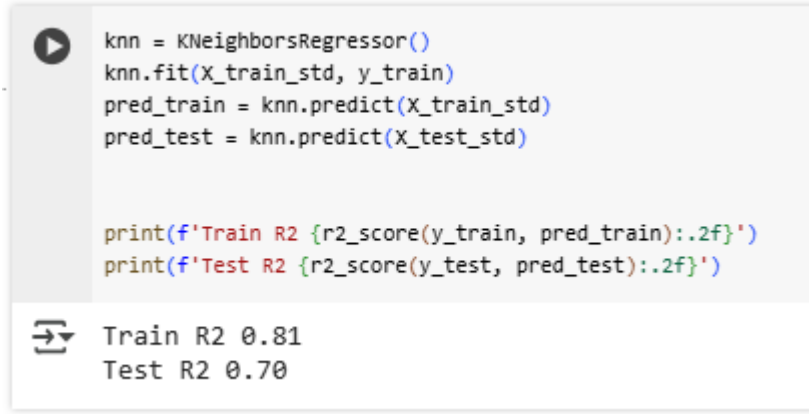

Рисунок 12 – Модель kNN, обученная на стандартизированных данных

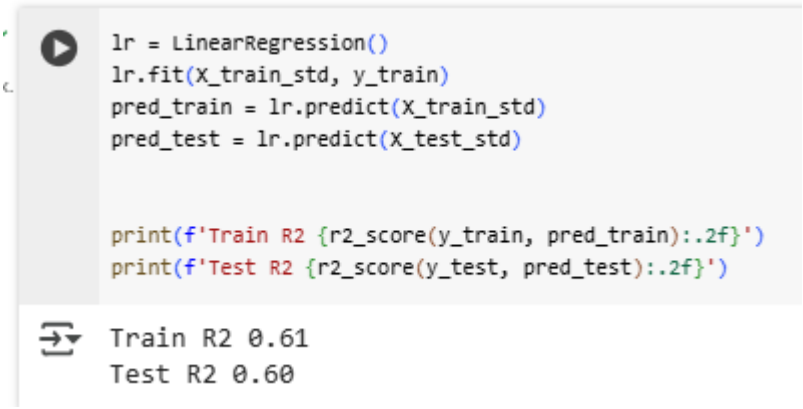

Рисунок 13 – Модель линейной регрессии, обученная на стандартизированных данных

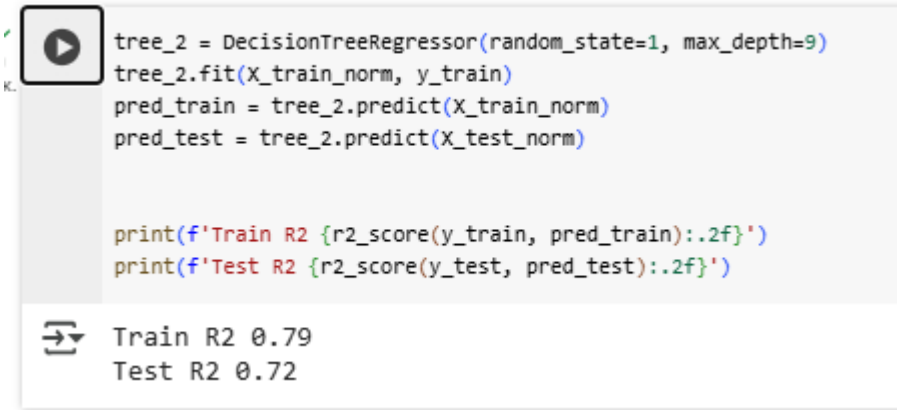

Рисунок 14 – Модель дерева решений, обученная на стандартизированных данных

Видно, что у дерева решений метрики не изменились, т.к. дереву решений не важно, какой масштаб у признака, дерево решений задает вопросы к данным. Если мы применяем масштабирование, то вопросы к признакам не изменятся, изменится только пороговое значение, с которым сравнивается вопрос.

Модно сделать такой вывод: масштабирование данных нужно для более стабильного обучения модели. Нормализация лучше зарекомендовала себя в подходах машинного обучения, которые работают с расстояниями (kNN). Стандартизация зарекомендовал себя в линейных подходах машинного обучения (LinearRegression, SVM). Для моделей, основанных не деревьях решений (DecisionTree, Bagging, RandomForest, Boosting) масштабирование данных необязательно.