

Лабораторная работа №5.

Разработка единого подхода к предварительной обработки данных

Цель лабораторной работы: изучение теоретических принципов и инструментальных средств для построения пайплайна для предварительной обработки данных.

Основные задачи:

- предварительная обработка данных;
- изучение библиотек для предварительной обработки данных;
- масштабирование признаков;
- представление категориальных данных;
- построение пайплайна для предварительной обработки данных.

Ход выполнения индивидуального задания:

1. Загрузка и визуализация набора данных

Первым делом импортируем все необходимые библиотеки для выполнения лабораторной работы, а также загрузим наш набор данных с помощью **pandas**:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


data_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data"

columns = [
    "Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type"
]

data = pd.read_csv(data_path, names=columns)
data.drop(columns=["Id"], inplace=True)
```

Признак **Id** удаляем из набора данных из-за ненужности.

При помощи строчки `print(data.head())` визуализируем первые **пять строк** из нашего набора данных. Как результат, получаем следующий вывод:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	

Разделим загруженные данные на **матрицу признаков** и **зависимую переменную**, после чего выведем их на экран:

```
X = data.iloc[:, :-1].values
y = data.iloc[:, 9].values

print("\nМатрица признаков")
print(X)
print("\nЗависимая переменная")
print(y)
```

Матрица признаков							
[1.52101	13.64	4.49	...	8.75	0.	0.
[1.51761	13.89	3.6	...	7.83	0.	0.
[1.51618	13.53	3.55	...	7.78	0.	0.
...							
[1.52065	14.36	0.	...	8.44	1.64	0.
[1.51651	14.38	0.	...	8.48	1.57	0.
[1.51711	14.23	0.	...	8.62	1.67	0.

[illegible]

2. Обработка пропущенных значений

Для обработки пропущенных значений в наборе данных воспользуемся функцией `SimpleImputer` из библиотеки **scikit**.

```
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
imputer = imputer.fit(X[:, 0:9])
X_without_nan = X.copy()
X_without_nan[:, 0:9] = imputer.transform(X[:, 0:9])

print("\nМатрица признаков после заполнения пропусков")
print(X_without_nan)
```

В качестве стратегии был выбран параметр `mean`, который заменяет пропущенные значения на **среднее значение по столбцу**.

Так как в наборе данных **Glass Identification** отсутствуют пропущенные значения, результатом работы должна быть изначальная матрица признаков.

Матрица признаков после заполнения пропусков

```
[ [ 1.52101 13.64 4.49 ... 8.75 0. 0. ]
  [ 1.51761 13.89 3.6 ... 7.83 0. 0. ]
  [ 1.51618 13.53 3.55 ... 7.78 0. 0. ]
  ...
  [ 1.52065 14.36 0. ... 8.44 1.64 0. ]
  [ 1.51651 14.38 0. ... 8.48 1.57 0. ]
  [ 1.51711 14.23 0. ... 8.62 1.67 0. ] ]]
```

3. Масштабирование признаков

В нашем наборе данных некоторые признаки имеют **разные масштабы**, что может повлиять на обучение модели. При помощи класса `StandardScaler` мы масштабируем признаки таким образом, чтобы их **среднее значение** стало **0**, а **стандартное отклонение** — **1**. Для этого воспользуемся кодом ниже:

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_without_nan[:, 0:9])

print("\nМатрица признаков после масштабирования")
print(X_scaled)
```

Матрица признаков после масштабирования

```
[ [ 0.87286765 0.28495326 1.25463857 ... -0.14576634 -0.352877
   -0.5864509 ]
  [-0.24933347 0.59181718 0.63616803 ... -0.79373376 -0.35287683
   -0.5864509 ]
  [-0.72131806 0.14993314 0.60142249 ... -0.82894938 -0.35287683
   -0.5864509 ]
  ...
  [ 0.75404635 1.16872135 -1.86551055 ... -0.36410319 2.95320036
   -0.5864509 ]
  [-0.61239854 1.19327046 -1.86551055 ... -0.33593069 2.81208731
   -0.5864509 ]
  [-0.41436305 1.00915211 -1.86551055 ... -0.23732695 3.01367739
   -0.5864509 ] ]]
```

Для **удобного просмотра** набора данных после масштабирования преобразуем его обратно в `DataFrame`:

```
X_data = pd.DataFrame(X_scaled, columns=columns[1:-1])
y_data = pd.DataFrame(y, columns=["Type"])
data_new = pd.concat([X_data, y_data], axis=1)
print(" ")
print(data_new.head())
```

	RI	Na	Mg	Al	...	Ca	Ba	Fe	Type
0	0.872868	0.284953	1.254639	-0.692442	...	-0.145766	-0.352877	-0.586451	1
1	-0.249333	0.591817	0.636168	-0.170460	...	-0.793734	-0.352877	-0.586451	1
2	-0.721318	0.149933	0.601422	0.190912	...	-0.828949	-0.352877	-0.586451	1
3	-0.232831	-0.242853	0.698710	-0.310994	...	-0.519052	-0.352877	-0.586451	1
4	-0.312045	-0.169205	0.650066	-0.411375	...	-0.624699	-0.352877	-0.586451	1

Как можно заметить, набор данных действительно был приведён к одному формату.

4. Разделение выборки на тестовую и тренировочную

Чтобы разделить выборку на тестовую и тренировочную воспользуемся функцией `train_test_split`:

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=24)
```

В качестве параметров передаём в функцию нашу **масштабированную матрицу признаков**, а также **массив зависимой переменной**. При помощи параметра `test_size` мы указываем процентную долю, которая уйдёт в тестовую выборку. В нашем случае она равняется 30%.

Приведём наши тестовые и тренировочные выборки в формате `DataFrame` для удобного просмотра:

```
X_train_df = pd.DataFrame(X_train, columns=columns[1:-1])
y_train_df = pd.DataFrame(y_train, columns=["Type"])

X_test_df = pd.DataFrame(X_test, columns=columns[1:-1])
y_test_df = pd.DataFrame(y_test, columns=["Type"])

print("\nОбучающая выборка (X_train):")
print(X_train_df.head())
print("\nЦелевая переменная обучающей выборки (y_train):")
print(y_train_df.head())

print("\nТестовая выборка (X_test):")
print(X_test_df.head())
print("\nЦелевая переменная тестовой выборки (y_test):")
print(y_test_df.head())
```

```
Обучающая выборка (X_train):
      RI      Na      Mg      ...      Ca      Ba
0 -0.691613 -0.500618  0.622270  ... -0.645828 -0.352877 -0.586451
1  0.183044  0.064011  0.795997  ... -0.526095 -0.352877  0.853719
2 -0.500178  1.856097 -1.865511  ... -0.392276  2.852405 -0.586451
3 -0.962261  0.898681 -1.865511  ...  0.079614  0.876823 -0.072105
4  0.037818  0.284953  0.823794  ... -0.448621 -0.352877  2.705365

[5 rows x 9 columns]
```

Целевая переменная обучающей выборки (y_train):

```

Type
0      2
1      2
2      7
3      7
4      2
```

```

Тестовая выборка (X_test):
      RI      Na      Mg      ...      Ca      Ba
0 -1.774207  1.217820 -0.656366  ... -0.962769 -0.352877 -0.586451
1 -0.335149 -0.476069  0.594473  ... -0.364103 -0.352877  0.133634
2 -0.189923  0.284953  0.670914  ... -0.018990 -0.352877 -0.586451
3 -0.493577 -0.218304  0.594473  ... -0.603569 -0.352877 -0.586451
4 -0.239432 -0.525167  0.594473  ... -0.300715 -0.352877 -0.586451

[5 rows x 9 columns]

Целевая переменная тестовой выборки (y_test):
Type
0      6
1      1
2      3
3      2
4      1

```

Контрольные вопросы

1. Для управления наборами данных чаще всего используется **pandas**.
2. Удаление строк с пропущенными значениями может привести к значительной потере информации, особенно если пропуски встречаются часто. Это может исказить выборку и повлиять на точность модели или дальнейший анализ данных.
3. Если y — это **целевой признак**, то **OneHotEncoder** не нужен. Вместо этого для классификационной задачи лучше использовать **Label Encoding**, который преобразует категориальные метки в числовые значения. **OneHotEncoding** следует применять к **категориальным признакам**, а не к целевой переменной.
4. Разбиение данных на **обучающую (train)** и **тестовую (test)** выборку необходимо для оценки качества модели. Вариант 20:80 наиболее оптимальный.
5. а) `dataset = read_csv("data.csv")`