

РК №2

Прошкин Георгий Павлович ИУ5-61Б (Вариант 13)

Задание. Для заданного набора данных построить модели логистической регрессии и случайного леса. Оценить качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

Метод №1: Линейная/логистическая регрессия

Метод №2: Случайный лес

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
df = pd.read_csv('marvel-wikia-data.csv')
```

```
df.head()
```

| | page_id | name \ |
|---|---------|-------------------------------------|
| 0 | 1678 | Spider-Man (Peter Parker) |
| 1 | 7139 | Captain America (Steven Rogers) |
| 2 | 64786 | Wolverine (James \"Logan\" Howlett) |
| 3 | 1868 | Iron Man (Anthony \"Tony\" Stark) |
| 4 | 2460 | Thor (Thor Odinson) |

| | urlslug | ID \ |
|---|--|------------------|
| 0 | \\/Spider-Man_(Peter_Parker) | Secret Identity |
| 1 | \\/Captain_America_(Steven_Rogers) | Public Identity |
| 2 | \\/Wolverine_(James_%22Logan%22_Howlett) | Public Identity |
| 3 | \\/Iron_Man_(Anthony_%22Tony%22_Stark) | Public Identity |
| 4 | \\/Thor_(Thor_Odinson) | No Dual Identity |

| | ALIGN | EYE | HAIR | SEX | GSM \ |
|---|--------------------|------------|------------|-----------------|-------|
| 0 | Good Characters | Hazel Eyes | Brown Hair | Male Characters | NaN |
| 1 | Good Characters | Blue Eyes | White Hair | Male Characters | NaN |
| 2 | Neutral Characters | Blue Eyes | Black Hair | Male Characters | NaN |
| 3 | Good Characters | Blue Eyes | Black Hair | Male Characters | NaN |
| 4 | Good Characters | Blue Eyes | Blond Hair | Male Characters | NaN |

| | ALIVE | APPEARANCES | FIRST APPEARANCE | Year |
|---|-------------------|-------------|------------------|--------|
| 0 | Living Characters | 4043.0 | Aug-62 | 1962.0 |
| 1 | Living Characters | 3360.0 | Mar-41 | 1941.0 |
| 2 | Living Characters | 3061.0 | Oct-74 | 1974.0 |
| 3 | Living Characters | 2961.0 | Mar-63 | 1963.0 |
| 4 | Living Characters | 2258.0 | Nov-50 | 1950.0 |

```
df = df.drop(columns = ['page_id', 'name', 'urlslug', 'FIRST APPEARANCE', 'Year'], axis = 1)
```

```
df.columns = df.columns.str.lower()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16376 entries, 0 to 16375
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   id               12606 non-null  object
1   align            13564 non-null  object
2   eye              6609 non-null   object
3   hair             12112 non-null  object
4   sex              15522 non-null  object
5   gsm              90 non-null     object
6   alive            16373 non-null  object
7   appearances      15280 non-null  float64
dtypes: float64(1), object(7)
memory usage: 1023.6+ KB
```

```
sum(df.duplicated(df.columns))
```

```
7838
```

```
df = df.drop_duplicates(df.columns, keep = 'last')
sum(df.duplicated(df.columns))
```

```
0
```

```
df.shape
```

```
(8538, 8)
```

```
df.isnull().sum()
```

| | |
|-------|------|
| id | 1454 |
| align | 1346 |
| eye | 3138 |
| hair | 1012 |
| sex | 384 |
| gsm | 8448 |
| alive | 1 |

```
appearances      459
dtype: int64
```

Пропусков в столбце alive мало, можем их удалить.

```
df.dropna(subset=['alive'], inplace=True)
```

Узнаем количество уникальных значений в каждом столбце.

```
df.nunique()
```

```
id              4
align           3
eye            24
hair           25
sex            4
gsm            6
alive           2
appearances    358
dtype: int64
```

```
df.gsm.value_counts(dropna=False)
```

```
NaN              8447
Homosexual Characters    66
Bisexual Characters    19
Transgender Characters    2
Transvestites           1
Pansexual Characters     1
Genderfluid Characters    1
Name: gsm, dtype: int64
```

В данной колонке много пустых значений, поэтому можем ее удалить.

```
df = df.drop(columns = ['gsm'], axis = 1)
```

Рассмотрим подробнее столбцы id, align, sex, alive.

```
df['id'].value_counts(dropna=False)
```

```
Secret Identity      3491
Public Identity      2363
NaN                  1453
No Dual Identity     1215
Known to Authorities Identity    15
Name: id, dtype: int64
```

Заменим пропуски значением Identity Unknown.

```
df['id'].fillna(value = "Identity Unknown", inplace = True)
```

```
df['align'].value_counts(dropna=False)
```

```
Bad Characters      2873
Good Characters     2834
Neutral Characters  1485
NaN                1345
Name: align, dtype: int64
```

Заменим пропуски значением Neutral Characters.

```
df['align'].fillna(value = "Neutral Characters", inplace = True)
df['sex'].value_counts(dropna=False)
```

```
Male Characters      5468
Female Characters    2643
NaN                 383
Agender Characters   41
Genderfluid Characters 2
Name: sex, dtype: int64
```

Заполним пропуски значением Genderless Characters.

```
df['sex'].fillna(value = "Genderless Characters", inplace = True)
```

Рассмотрим столбец eye.

```
df['eye'].value_counts(dropna=False)
```

```
NaN                3137
Blue Eyes          1633
Brown Eyes         1413
Green Eyes          518
Black Eyes          461
Red Eyes            418
White Eyes          333
Yellow Eyes         234
Grey Eyes           94
Hazel Eyes          76
Variable Eyes       49
Purple Eyes         31
Orange Eyes         25
Pink Eyes           21
One Eye             21
Gold Eyes           14
Silver Eyes         12
Violet Eyes         11
Amber Eyes          10
Multiple Eyes        7
No Eyes             7
Yellow Eyeballs     6
Black Eyeballs      3
Magenta Eyes        2
```

```
Compound Eyes          1
Name: eye, dtype: int64
```

Сгруппируем значения и избавимся от пропусков

```
eyes = ['Blue Eyes', 'Brown Eyes', 'Black Eyes', 'Green Eyes', 'Red
Eyes']
eyes_new = []
for i in df.eyes.values:
    if i not in eyes:
        eyes_new.append('Other color')
    else:
        eyes_new.append(i)
df['eye'] = eyes_new

df['eye'].value_counts(dropna=False)
```

```
Other color    4094
Blue Eyes     1633
Brown Eyes    1413
Green Eyes     518
Black Eyes    461
Red Eyes      418
Name: eye, dtype: int64
```

Аналогично поступим со столбцом hair.

```
df['hair'].value_counts(dropna=False)
```

```
Black Hair      1890
Brown Hair      1370
Blond Hair      1033
NaN             1011
No Hair         815
Bald            535
White Hair      502
Red Hair        494
Grey Hair       386
Green Hair      107
Auburn Hair     74
Blue Hair       55
Purple Hair     47
Strawberry Blond Hair 47
Orange Hair     43
Variable Hair   32
Pink Hair       31
Yellow Hair     20
Silver Hair     16
Gold Hair       8
Reddish Blond Hair 6
Light Brown Hair 6
Magenta Hair    5
```

```
Orange-brown Hair      2
Bronze Hair            1
Dyed Hair              1
Name: hair, dtype: int64
```

```
hair = ['Black Hair', 'Brown Hair', 'Blond Hair', 'Red Hair', 'White Hair']
hair_new = []
for i in df.hair.values:
    if i not in hair:
        hair_new.append('Other color')
    else:
        hair_new.append(i)
df['hair'] = hair_new

df['hair'].value_counts(dropna=False)
```

```
Other color    3248
Black Hair     1890
Brown Hair     1370
Blond Hair     1033
White Hair      502
Red Hair       494
Name: hair, dtype: int64
```

Пропуски в столбце appearances заполним медианным значением.

```
df['appearances'] =
df['appearances'].fillna(df['appearances'].median())

df.isnull().sum()

id          0
align       0
eye         0
hair        0
sex         0
alive       0
appearances 0
dtype: int64
```

Кодирование категориальных признаков

Теперь закодируем категориальные признаки с помощью Label Encoder.

```
le = LabelEncoder()
df['id'] = le.fit_transform(df['id'])
df['align'] = le.fit_transform(df['align'])
df['eye'] = le.fit_transform(df['eye'])
df['hair'] = le.fit_transform(df['hair'])
df['sex'] = le.fit_transform(df['sex'])
df['alive'] = le.fit_transform(df['alive'])
```

```
df.head()
```

| | id | align | eye | hair | sex | alive | appearances |
|---|----|-------|-----|------|-----|-------|-------------|
| 0 | 4 | 1 | 4 | 2 | 4 | 1 | 4043.0 |
| 1 | 3 | 1 | 1 | 5 | 4 | 1 | 3360.0 |
| 2 | 3 | 2 | 1 | 0 | 4 | 1 | 3061.0 |
| 3 | 3 | 1 | 1 | 0 | 4 | 1 | 2961.0 |
| 4 | 2 | 1 | 1 | 1 | 4 | 1 | 2258.0 |

Разделение выборки

Разделим выборку на обучающую и тестовую.

Целевым признаком выберем столбец alive (жив герой или нет).

```
y = df['alive']
x = df.drop(['alive'], axis = 1)

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.3,
                                                    random_state = 42)
```

Метрики

```
def print_metrics(test, prediction):
    print("Accuracy:", accuracy_score(test, prediction))
    print("Precision:", precision_score(test, prediction))
```

Логистическая регрессия

```
lr = LogisticRegression()
lr_prediction = lr.fit(x_train, y_train).predict(x_test)
print_metrics(y_test, lr_prediction)
```

```
Accuracy: 0.7295081967213115
Precision: 0.7295081967213115
```

По значению метрик можно сказать, что модель приблизительно на 73% идентифицирует как сам объект, так и его класс.

Случайный лес

```
rf = RandomForestClassifier()
rf_prediction = rf.fit(x_train, y_train).predict(x_test)
print_metrics(y_test, rf_prediction)
```

```
Accuracy: 0.6221701795472288
Precision: 0.7164824603555983
```

В данном случае можно сделать вывод о том, что модель правильно классифицирует 62% объектов и при этом в 72% случаев верно определяет класс объекта.