РК №2

Задание. Для заданного набора данных (по Вашему варианту) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (по варианту для Вашей группы). Оцените 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.

ИУ5-65Б: Метод опорных векторов, Градиентный бустинг

https://www.kaggle.com/fivethirtyeight/fivethirtyeight-comic-characters-dataset (файл marvel-wikia-data.csv)

```
%pip install -q seaborn
%pip install -q xgboost

import numpy as np
import pandas as pd
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_squared_log_error, median_absolute_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR,
NuSVR, LinearSVR
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import AdaBoostClassifier,
GradientBoostingClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
%matplotlib inline
sns.set(style="ticks")

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

data
```

```
       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)
...        ...                                 ...
16371   657508                 Ru'ach (Earth-616)
16372   665474      Thane (Thanos' son) (Earth-616)
16373   695217        Tinkerer (Skrull) (Earth-616)
16374   708811       TK421 (Spiderling) (Earth-616)
16375   673702              Yologarch (Earth-616)

                                       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
...                                        ...               ...
16371                     \/Ru%27ach_(Earth-616)  No Dual Identity
16372      \/Thane_(Thanos%27_son)_(Earth-616)  No Dual Identity
16373          \/Tinkerer_(Skrull)_(Earth-616)   Secret Identity
16374         \/TK421_(Spiderling)_(Earth-616)   Secret Identity
16375                  \/Yologarch_(Earth-616)               NaN

                    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
...                   ...         ...         ...              ...  ..
.
16371      Bad Characters  Green Eyes     No Hair  Male Characters
NaN
16372     Good Characters   Blue Eyes        Bald  Male Characters
NaN
16373      Bad Characters  Black Eyes        Bald  Male Characters
NaN
16374  Neutral Characters         NaN         NaN  Male Characters
NaN
16375      Bad Characters         NaN         NaN              NaN
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
...                  ...         ...             ...     ...
16371  Living Characters         NaN             NaN     NaN
16372  Living Characters         NaN             NaN     NaN
16373  Living Characters         NaN             NaN     NaN
16374  Living Characters         NaN             NaN     NaN
16375  Living Characters         NaN             NaN     NaN

[16376 rows x 13 columns]
```

```
data.dtypes
```

```
page_id               int64
name                 object
urlslug              object
ID                   object
ALIGN                object
EYE                  object
HAIR                 object
SEX                  object
GSM                  object
ALIVE                object
APPEARANCES         float64
FIRST APPEARANCE     object
Year                float64
dtype: object
```

Обработка пустых значений

```
# Проверим наличие пустых значений
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    print('{} - {}'.format(col, temp_null_count))
```

```
page_id - 0
name - 0
urlslug - 0
ID - 3770
ALIGN - 2812
EYE - 9767
HAIR - 4264
SEX - 854
GSM - 16286
ALIVE - 3
APPEARANCES - 1096
```

```
FIRST APPEARANCE - 815
Year - 815
```

Удалим колонки в которых пропущено более проловины всех значений. Затем удалим строки с пропусками.

```
try:
    data = data.drop(['GSM', 'EYE'], axis=1)
    data = data.dropna(axis=0, how='any')
except:
    pass
data.shape
```

```
(8020, 11)
```

Кодирование категориальных признаков

```
#удалим признаки, не влияющие на целевой признак
try:
    data = data.drop(['name', 'urlslug','FIRST APPEARANCE'], axis=1)
except:
    pass
data.head()
```

	page_id	ID	ALIGN	HAIR	SEX	ALIVE	APPEARANCES	Year
0	1678	1.000000	0.5	0.250000	1.0	1.0	4043.0	1962.0
1	7139	0.666667	0.5	0.958333	1.0	1.0	3360.0	1941.0
2	64786	0.666667	1.0	0.083333	1.0	1.0	3061.0	1974.0
3	1868	0.666667	0.5	0.083333	1.0	1.0	2961.0	1963.0
4	2460	0.333333	0.5	0.125000	1.0	1.0	2258.0	1950.0

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
df_int = le.fit_transform(data['ID'])
data['ID'] = df_int
df_int = le.fit_transform(data['ALIGN'])
data['ALIGN'] = df_int
df_int = le.fit_transform(data['HAIR'])
data['HAIR'] = df_int
df_int = le.fit_transform(data['SEX'])
data['SEX'] = df_int
df_int = le.fit_transform(data['ALIVE'])
data['ALIVE'] = df_int
data.head()
```

	page_id	ID	ALIGN	HAIR	SEX	ALIVE	APPEARANCES	Year
0	1678	3	1	6	3	1	4043.0	1962.0
1	7139	2	1	23	3	1	3360.0	1941.0

```
2    64786   2       2      2    3      1       3061.0  1974.0
3     1868   2       1      2    3      1       2961.0  1963.0
4     2460   1       1      3    3      1       2258.0  1950.0
```

```
sc1 = MinMaxScaler()
data['ID'] = sc1.fit_transform(data[['ID']])
data['ALIGN'] = sc1.fit_transform(data[['ALIGN']])
data['HAIR'] = sc1.fit_transform(data[['HAIR']])
data['SEX'] = sc1.fit_transform(data[['SEX']])
data['ALIVE'] = sc1.fit_transform(data[['ALIVE']])
data.head()
```

	page_id	ID	ALIGN	HAIR	SEX	ALIVE	APPEARANCES	Year
0	1678	1.000000	0.5	0.250000	1.0	1.0	4043.0	1962.0
1	7139	0.666667	0.5	0.958333	1.0	1.0	3360.0	1941.0
2	64786	0.666667	1.0	0.083333	1.0	1.0	3061.0	1974.0
3	1868	0.666667	0.5	0.083333	1.0	1.0	2961.0	1963.0
4	2460	0.333333	0.5	0.125000	1.0	1.0	2258.0	1950.0

Разделение на обучающую и тестовую выборки.

```
target = data['ALIVE']
data_X_train, data_X_test, data_y_train, data_y_test =
train_test_split(
    data, target, test_size=0.2, random_state=1)
```

```
data_X_train.shape, data_X_test.shape, data_y_train.shape,
data_y_test.shape
```

```
((6416, 8), (1604, 8), (6416,), (1604,))
```

Метод опорных векторов

Стандартная модель без доп параметров

```
svr_1 = LinearSVC(dual=False)
svr_1.fit(data_X_train, data_y_train)
```

```
LinearSVC(dual=False)
```

```
data_y_pred_1 = svr_1.predict(data_X_test)
```

```
accuracy_score(data_y_test, data_y_pred_1)
```

```
0.7356608478802993
```

```
f1_score(data_y_test, data_y_pred_1, average='micro')
```

```
0.7356608478802993
```

```
f1_score(data_y_test, data_y_pred_1, average='macro')
```

```
0.4238505747126437
```

```
f1_score(data_y_test, data_y_pred_1, average='weighted')
```

```
0.6236205463353112
```

Добавим параметр регуляризации (C), который контролирует штраф за неправильную классификацию обучающих образцов

```
svr_2 = LinearSVC(C=1.0, max_iter=10000, dual=False)
svr_2.fit(data_X_train, data_y_train)
```

```
LinearSVC(dual=False, max_iter=10000)
```

```
data_y_pred_2 = svr_2.predict(data_X_test)
```

```
accuracy_score(data_y_test, data_y_pred_2)
```

```
0.7356608478802993
```

```
f1_score(data_y_test, data_y_pred_2, average='micro')
```

```
0.7356608478802993
```

```
f1_score(data_y_test, data_y_pred_2, average='macro')
```

```
0.4238505747126437
```

```
f1_score(data_y_test, data_y_pred_2, average='weighted')
```

```
0.6236205463353112
```

Градиентный бустинг

Модель градиентного бустинга с использованием библиотеки xgboost

```
ab1 = XGBClassifier()
ab1.fit(data_X_train, data_y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1,
colsample_bytree=1,
```

```
                early_stopping_rounds=None, enable_categorical=False,
                eval_metric=None, gamma=0, gpu_id=-1,
grow_policy='depthwise',
                importance_type=None, interaction_constraints='',
                learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
                max_delta_step=0, max_depth=6, max_leaves=0,
min_child_weight=1,
                missing=nan, monotone_constraints='()',
n_estimators=100,
                n_jobs=0, num_parallel_tree=1, predictor='auto',
random_state=0,
                reg_alpha=0, reg_lambda=1, ...)
```

```
data_y_pred_1 = ab1.predict(data_X_test)
```

```
data_y_pred_1_0 = ab1.predict(data_X_train)
```

```
accuracy_score(data_y_train, data_y_pred_1_0)
```

```
1.0
```

```
accuracy_score(data_y_test, data_y_pred_1)
```

```
1.0
```

```
f1_score(data_y_test, data_y_pred_1, average='micro')
```

```
1.0
```

```
f1_score(data_y_test, data_y_pred_1, average='macro')
```

```
1.0
```

```
f1_score(data_y_test, data_y_pred_1, average='weighted')
```

```
1.0
```

Модель градиентного бустинга показала себя лучше, чем модель, основанная на методе опорных векторов