Рубежный контроль по дисциплине “Технологии машинного обучения”

Алексеев Андрей ИУ5-62Б

Вариант№1

Задание

Для заданного набора данных (по варианту №1) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (Метод опорных векторов и Случайный лес). 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.

```
{
  "authors": [
    {
      "name": "Алексеев Андрей Сергеевич"
    }
  ],
  "group": "ИУ5-62Б",
  "kernelspec": {
    "name": "python3",
    "display_name": "Python 3 (ipykernel)",
    "language": "python"
  },
  "language_info": {
    "name": "python",
    "version": "3.9.7",
    "mimetype": "text/x-python",
    "codemirror_mode": {
      "name": "ipython",
      "version": 3
    },
    "pygments_lexer": "ipython3",
    "nbconvert_exporter": "python",
    "file_extension": ".py"
  },
  "title": "РК№2",
  "Вариант": "1"
}
```



```
{'authors': [{'name': 'Алексеев Андрей Сергеевич'}],
 'group': 'ИУ5-62Б',
 'kernelspec': {'name': 'python3',
  'display_name': 'Python 3 (ipykernel)',
  'language': 'python'},
 'language_info': {'name': 'python',
  'version': '3.9.7',
  'mimetype': 'text/x-python',
  'codemirror_mode': {'name': 'ipython', 'version': 3},
  'pygments_lexer': 'ipython3',
  'nbconvert_exporter': 'python',
  'file_extension': '.py'},
 'title': 'РК№2',
 'Вариант': '1'}
```


```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from sklearn.datasets import load_iris, load_boston
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsRegressor,
KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_squared_log_error, median_absolute_error, r2_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

```
# https://scikit-learn.org/stable/datasets/index.html#iris-dataset
boston = load_boston()
# Наименования признаков
boston.feature_names
```

```
C:\Users\Админ\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\deprecation.py:87: FutureWarning: Function
load_boston is deprecated; `load_boston` is deprecated in 1.0 and will
be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can
refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use
of this
    dataset unless the purpose of the code is to study and educate
about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22,
header=None)
        data = np.hstack([raw_df.values[::2, :],
raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]
```

```
    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames
housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)

array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

```
# Значения признаков
boston.data[:5]
```

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, 0.0000e+00, 5.3800e-01,
        6.5750e+00, 6.5200e+01, 4.0900e+00, 1.0000e+00, 2.9600e+02,
        1.5300e+01, 3.9690e+02, 4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
        6.4210e+00, 7.8900e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
        1.7800e+01, 3.9690e+02, 9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
        7.1850e+00, 6.1100e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
        1.7800e+01, 3.9283e+02, 4.0300e+00],
       [3.2370e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 4.5800e-01,
        6.9980e+00, 4.5800e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
        1.8700e+01, 3.9463e+02, 2.9400e+00],
       [6.9050e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 4.5800e-01,
        7.1470e+00, 5.4200e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
        1.8700e+01, 3.9690e+02, 5.3300e+00]])
```

```
# Значения целевого признака
np.unique(boston.target)
```

```
array([ 5. ,  5.6,  6.3,  7. ,  7.2,  7.4,  7.5,  8.1,  8.3,  8.4,
8.5,
        8.7,  8.8,  9.5,  9.6,  9.7, 10.2, 10.4, 10.5, 10.8, 10.9, 11.
,
       11.3, 11.5, 11.7, 11.8, 11.9, 12. , 12.1, 12.3, 12.5, 12.6,
12.7,
       12.8, 13. , 13.1, 13.2, 13.3, 13.4, 13.5, 13.6, 13.8, 13.9, 14.
,
```

```
       14.1, 14.2, 14.3, 14.4, 14.5, 14.6, 14.8, 14.9, 15. , 15.1,
15.2,
       15.3, 15.4, 15.6, 15.7, 16. , 16.1, 16.2, 16.3, 16.4, 16.5,
16.6,
       16.7, 16.8, 17. , 17.1, 17.2, 17.3, 17.4, 17.5, 17.6, 17.7,
17.8,
       17.9, 18. , 18.1, 18.2, 18.3, 18.4, 18.5, 18.6, 18.7, 18.8,
18.9,
       19. , 19.1, 19.2, 19.3, 19.4, 19.5, 19.6, 19.7, 19.8, 19.9, 20.
,
       20.1, 20.2, 20.3, 20.4, 20.5, 20.6, 20.7, 20.8, 20.9, 21. ,
21.1,
       21.2, 21.4, 21.5, 21.6, 21.7, 21.8, 21.9, 22. , 22.1, 22.2,
22.3,
       22.4, 22.5, 22.6, 22.7, 22.8, 22.9, 23. , 23.1, 23.2, 23.3,
23.4,
       23.5, 23.6, 23.7, 23.8, 23.9, 24. , 24.1, 24.2, 24.3, 24.4,
24.5,
       24.6, 24.7, 24.8, 25. , 25.1, 25.2, 25.3, 26.2, 26.4, 26.5,
26.6,
       26.7, 27. , 27.1, 27.5, 27.9, 28. , 28.1, 28.2, 28.4, 28.5,
28.6,
       28.7, 29. , 29.1, 29.4, 29.6, 29.8, 29.9, 30.1, 30.3, 30.5,
30.7,
       30.8, 31. , 31.1, 31.2, 31.5, 31.6, 31.7, 32. , 32.2, 32.4,
32.5,
       32.7, 32.9, 33. , 33.1, 33.2, 33.3, 33.4, 33.8, 34.6, 34.7,
34.9,
       35.1, 35.2, 35.4, 36. , 36.1, 36.2, 36.4, 36.5, 37. , 37.2,
37.3,
       37.6, 37.9, 38.7, 39.8, 41.3, 41.7, 42.3, 42.8, 43.1, 43.5,
43.8,
       44. , 44.8, 45.4, 46. , 46.7, 48.3, 48.5, 48.8, 50. ])
```

```
# Размер выборки
boston.data.shape, boston.target.shape
```

```
((506, 13), (506,))
```

```
# Сформируем DataFrame
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

raw_df.rename(columns={0: 'CRIM'}, inplace=True)
raw_df.rename(columns={1: 'ZN'}, inplace=True)
raw_df.rename(columns={2: 'INDUS'}, inplace=True)
raw_df.rename(columns={3: 'CHAS'}, inplace=True)
raw_df.rename(columns={4: 'NOX'}, inplace=True)
raw_df.rename(columns={5: 'RM'}, inplace=True)
```

```
raw_df.rename(columns={6: 'AGE'}, inplace=True)
raw_df.rename(columns={7: 'DIS'}, inplace=True)
raw_df.rename(columns={8: 'RAD'}, inplace=True)
raw_df.rename(columns={9: 'TAX'}, inplace=True)
raw_df.rename(columns={10: 'PTRATIO'}, inplace=True)
```

```
# Удаление строк, содержащих пустые значения
raw_df_2 = raw_df.dropna(axis=0, how='any')
(raw_df.shape, raw_df_2.shape)
```

```
((1012, 11), (506, 11))
```

```
# Проверим наличие пустых значений
# Цикл по колонкам датасета
for col in raw_df_2.columns:
    # Количество пустых значений - все значения заполнены
    temp_null_count = raw_df_2[raw_df_2[col].isnull()].shape[0]
    print('{} - {}'.format(col, temp_null_count))
```

```
CRIM - 0
ZN - 0
INDUS - 0
CHAS - 0
NOX - 0
RM - 0
AGE - 0
DIS - 0
RAD - 0
TAX - 0
PTRATIO - 0
```

```
raw_df_2['target'] = target
target1 = np.empty(len(raw_df_2['target']), dtype=np.int16)
j = 0
a = [0, 1, 2]
for i in raw_df_2['target']:
    if i <= 20 and i >= 5:
        target1[j] = 0
        j = j + 1
    if i <= 35 and i > 20:
        target1[j] = 1
        j = j + 1
    if i <= 50 and i > 35:
        target1[j] = 2
        j = j + 1
raw_df_2['target1'] = target1
raw_df_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 506 entries, 0 to 1010
Data columns (total 13 columns):
```

```
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  target   506 non-null    float64
 12  target1  506 non-null    int16
dtypes: float64(12), int16(1)
memory usage: 52.4 KB

C:\Users\7272~1\AppData\Local\Temp/ipykernel_2840/200341314.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  raw_df_2['target'] = target
C:\Users\7272~1\AppData\Local\Temp/ipykernel_2840/200341314.py:15:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  raw_df_2['target1'] = target1
```

```
boston_X_train, boston_X_test, boston_y_train, boston_y_test =
train_test_split(
    raw_df_2.drop(['target1'], axis=1), raw_df_2['target1'],
test_size=0.5, random_state=17)

def print_metrics(y_test, y_pred):
    rep = classification_report(y_test, y_pred, output_dict=True)
    print("weighted precision:", rep['weighted avg']['precision'])
    print("weighted recall:", rep['weighted avg']['recall'])
    print("weighted f1-score:", rep['weighted avg']['f1-score'])
    plt.figure(figsize=(4, 3))
    plt.title('Матрица ошибок')
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True,
cmap="Blues");
```

```
scaler = StandardScaler().fit(boston_X_train)
boston_X_train_scaled = pd.DataFrame(scaler.transform(boston_X_train),
columns=boston_X_train.columns)
boston_X_test_scaled = pd.DataFrame(scaler.transform(boston_X_test),
columns=boston_X_train.columns)
boston_X_train_scaled.describe()
```

```
               CRIM            ZN         INDUS        CHAS
NOX  \
count  2.530000e+02  2.530000e+02  2.530000e+02  253.000000
2.530000e+02
mean   5.616939e-17 -7.021173e-18 -5.967997e-17    0.000000
2.281881e-16
std    1.001982e+00  1.001982e+00  1.001982e+00    1.001982
1.001982e+00
min   -4.896445e-01 -4.765149e-01 -1.585390e+00   -0.276759 -
1.411455e+00
25%   -4.778570e-01 -4.765149e-01 -8.681584e-01   -0.276759 -
9.555808e-01
50%   -4.474570e-01 -4.765149e-01 -2.232402e-01   -0.276759 -
1.600349e-01
75%    1.132433e-01  1.022018e-01  1.017895e+00   -0.276759
6.712659e-01
max    8.782363e+00  4.153219e+00  2.440552e+00    3.613247
2.816558e+00

                 RM           AGE           DIS           RAD
TAX  \
count  2.530000e+02  2.530000e+02  2.530000e+02  2.530000e+02
2.530000e+02
mean  -1.011049e-15 -5.757362e-16 -2.106352e-16 -3.510587e-17
1.843058e-17
std    1.001982e+00  1.001982e+00  1.001982e+00  1.001982e+00
1.001982e+00
min   -3.742422e+00 -2.362153e+00 -1.258854e+00 -1.001348e+00 -
1.284044e+00
25%   -5.981299e-01 -7.768940e-01 -8.206675e-01 -6.646074e-01 -
7.683696e-01
50%   -1.456454e-01  2.894877e-01 -2.232697e-01 -5.523606e-01 -
4.670769e-01
75%    5.432726e-01  9.014046e-01  6.066735e-01  1.580328e+00
1.485531e+00
max    3.245951e+00  1.112534e+00  3.370167e+00  1.580328e+00
1.746265e+00

            PTRATIO        target
count  2.530000e+02  2.530000e+02
mean   4.844610e-16  3.484257e-16
std    1.001982e+00  1.001982e+00
min   -2.667693e+00 -1.814881e+00
```

```
25%   -4.805161e-01 -6.609675e-01
50%    2.029766e-01 -1.561302e-01
75%    7.953370e-01  4.105239e-01
max    1.615528e+00  2.821379e+00
```

```
svm_model = SVC()
svm_model.fit(boston_X_train_scaled, boston_y_train)
boston_y_pred_svm = svm_model.predict(boston_X_test_scaled)
print_metrics(boston_y_test, boston_y_pred_svm)
```

```
weighted precision: 0.8705314009661835
weighted recall: 0.8656126482213439
weighted f1-score: 0.864150968697932
```


```
params = {'C': np.concatenate([np.arange(0.1, 2, 0.03), np.arange(2,
20, 1)])}
grid_cv = GridSearchCV(estimator=svm_model, param_grid=params, cv=10,
n_jobs=-1, scoring='f1_macro')
grid_cv.fit(boston_X_train_scaled, boston_y_train)
print(grid_cv.best_params_)
```

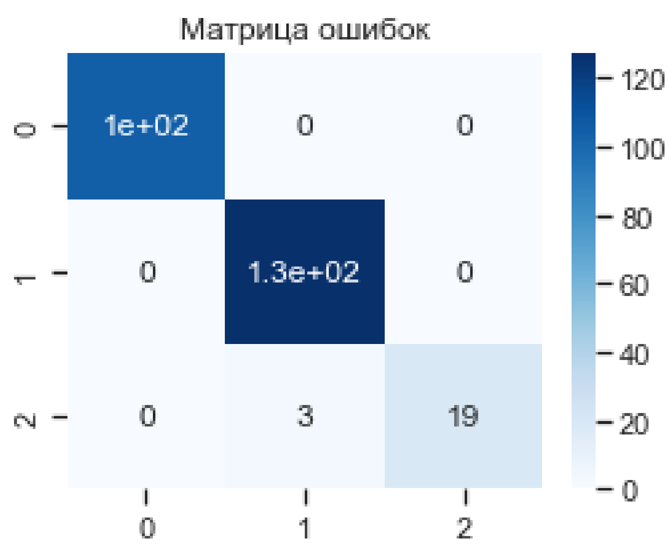
```
{'C': 11.0}
```

```
best_svm_model = grid_cv.best_estimator_
best_svm_model.fit(boston_X_train_scaled, boston_y_train)
boston_y_pred_svm = best_svm_model.predict(boston_X_test_scaled)
print_metrics(boston_y_test, boston_y_pred_svm)
```

```
weighted precision: 0.9129241554180029
weighted recall: 0.9130434782608695
weighted f1-score: 0.9127799736495389
```


```
rfc_model = RandomForestClassifier()
rfc_model.fit(boston_X_train, boston_y_train)
boston_y_pred_rfc = rfc_model.predict(boston_X_test)
print_metrics(boston_y_test, boston_y_pred_rfc)
```

```
weighted precision: 0.9884159318941929
weighted recall: 0.9881422924901185
weighted f1-score: 0.9877776823322747
```

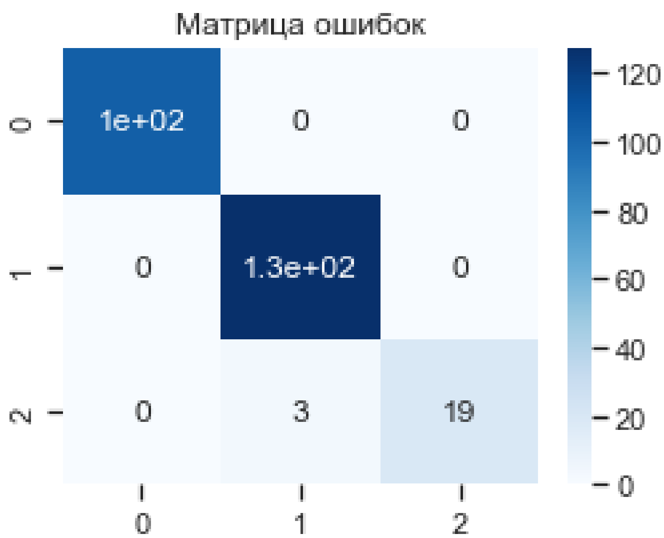



```
params = {'n_estimators': [5, 10, 50, 100], 'max_features': [2, 3, 4],
'criterion': ['gini', 'entropy'], 'min_samples_leaf': [1, 2, 3, 4, 5]}
grid_cv = GridSearchCV(estimator=rfc_model, param_grid=params, cv=10,
n_jobs=-1, scoring='f1_weighted')
grid_cv.fit(boston_X_train, boston_y_train)
print(grid_cv.best_params_)
```

```
{'criterion': 'gini', 'max_features': 4, 'min_samples_leaf': 1,
'n_estimators': 50}
```

```
best_rfc_model = grid_cv.best_estimator_
best_rfc_model.fit(boston_X_train, boston_y_train)
boston_y_pred_rfc = best_rfc_model.predict(boston_X_test)
print_metrics(boston_y_test, boston_y_pred_rfc)
```

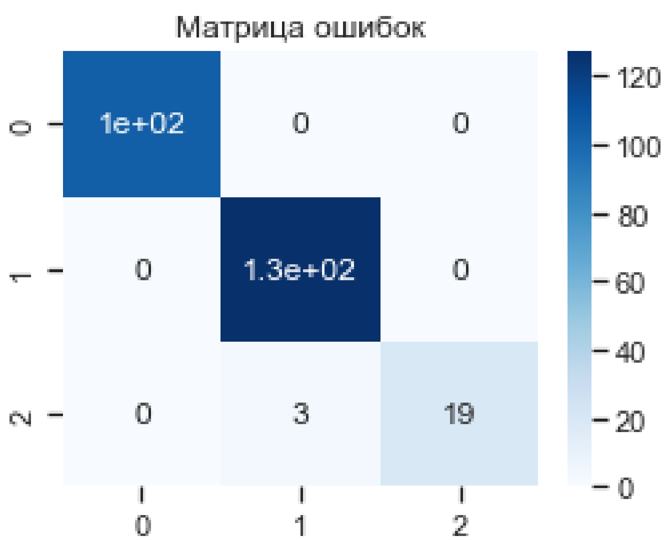
```
weighted precision: 0.9884159318941929
weighted recall: 0.9881422924901185
weighted f1-score: 0.9877776823322747
```




```
print_metrics(boston_y_test, boston_y_pred_svm)
print_metrics(boston_y_test, boston_y_pred_rfc)
```

```
weighted precision: 0.9129241554180029
weighted recall: 0.9130434782608695
weighted f1-score: 0.9127799736495389
weighted precision: 0.9884159318941929
weighted recall: 0.9881422924901185
weighted f1-score: 0.9877776823322747
```


Матрица ошибок
97
7
0
9
1.2e+02
2
0
4
18
0
1
2
0
1
2
100
80
60
40
20
0


Матрица ошибок
1e+02
0
0
0
1.3e+02
0
0
3
19
0
1
2
0
1
2
120
100
80
60
40
20
0

Вывод

Модели с подобранными гиперпараметрами оказались лучше базовых моделей. Обе конечные модели показали очень высокую точность прогноза, из-за того, что выбранный датасет является учебным. Из матриц ошибок видим, что модель метод опорных векторов совершила 22 неверных прогноза из 256, а модель случайный лес совершила 3 неверных прогноза из 256. Метрики показывают, что качества рассматриваемых моделей отличается.