

Бокатуев М. С. ИУ5-62Б

Рубежный контроль №2 (вариант 1)

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.

Набор данных

https://scikit-learn.org/stable/modules/generated/ sklearn.datasets.load_iris.html#sklearn.datasets.load_iris

Методы

- метод опорных векторов
- случайный лес

Решение

```
In []: import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.datasets import load_iris
    from sklearn.preprocessing import StandardScaler, MinMaxScaler
    from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.svm import SVC
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
In []: iris = load iris()
```

```
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['target'] = iris.target
data.head()
```

Out[]:		sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
	0	5.1	3.5	1.4	0.2	0
	1	4.9	3.0	1.4	0.2	0
	2	4.7	3.2	1.3	0.2	0
	3	4.6	3.1	1.5	0.2	0
	4	5.0	3.6	1.4	0.2	0

```
In [ ]: data.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):

Column Non-Null Count Dtype _____ ---sepal length (cm) 150 non-null float64 0 1 sepal width (cm) 150 non-null float64 2 petal length (cm) 150 non-null float64 3 petal width (cm) float64 150 non-null 150 non-null int32 4 target

dtypes: float64(4), int32(1)

memory usage: 5.4 KB

In []: data.describe()

Out[

]:		sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
	count	150.000000	150.000000	150.000000	150.000000	150.000000
	mean	5.843333	3.057333	3.758000	1.199333	1.000000
	std	0.828066	0.435866	1.765298	0.762238	0.819232
	min	4.300000	2.000000	1.000000	0.100000	0.000000
	25%	5.100000	2.800000	1.600000	0.300000	0.000000
	50 %	5.800000	3.000000	4.350000	1.300000	1.000000
	75 %	6.400000	3.300000	5.100000	1.800000	2.000000
	max	7.900000	4.400000	6.900000	2.500000	2.000000

```
In [ ]: data['target'].value_counts(normalize=True)
```

```
Out[]: 0
             0.333333
             0.333333
             0.333333
        Name: target, dtype: float64
In []: print('Количество пропущенных значений')
        data.isnull().sum()
      Количество пропущенных значений
Out[]: sepal length (cm)
        sepal width (cm)
                            0
        petal length (cm)
                            0
        petal width (cm)
                            0
                            0
        target
        dtype: int64
        Пропуски в данных не обнаружены.
```

Выбор метрик и подготовка данных

Так как выполняется задача небинарной классификации и в тестовой выборке возможен дисбаланс классов, были выбраны следующие метрики:

- precision;
- recall;
- f1-score.

Всем метрикам был задан уровень детализации average='weighted'.

```
In []:
    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 fl-score:", rep['weighted avg']['fl-score'])
        plt.figure(figsize=(4, 3))
        plt.title('Matpuцa ошибок')
        sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues");

In []: x_train, x_test, y_train, y_test = train_test_split(data.drop(['target'], axis))
```

SVC

Базовая модель

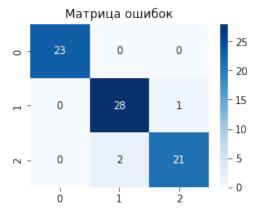
```
In [ ]: svm_model = SVC()
    svm_model.fit(x_train_scaled, y_train)
```

```
y_pred_svm = svm_model.predict(x_test_scaled)
print_metrics(y_test, y_pred_svm)
```

weighted precision: 0.9602828282828282

weighted recall: 0.96

weighted f1-score: 0.9598945386064031



Масштабирование данных

In []: scaler = StandardScaler().fit(x_train)
 x_train_scaled = pd.DataFrame(scaler.transform(x_train), columns=x_train.colum
 x_test_scaled = pd.DataFrame(scaler.transform(x_test), columns=x_train.columns
 x_train_scaled.describe()

_				
(1)	1.1	+	- 1	
\cup	ш		- 1	

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	7.500000e+01	7.500000e+01	7.500000e+01	7.500000e+01
mean	1.169435e-16	-7.460699e-16	-6.069219e-17	6.365279e-17
std	1.006734e+00	1.006734e+00	1.006734e+00	1.006734e+00
min	-1.710367e+00	-2.351670e+00	-1.469543e+00	-1.377354e+00
25%	-8.901782e-01	-5.655914e-01	-1.200725e+00	-1.188848e+00
50%	-6.998944e-02	-1.190719e-01	3.584252e-01	2.563688e-01
75 %	5.861615e-01	5.507074e-01	8.154174e-01	8.847238e-01
max	2.226539e+00	3.006565e+00	1.702520e+00	1.513079e+00

Подбор гиперпараметров

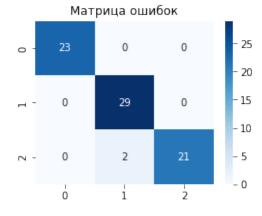
```
In []: params = {'C': np.concatenate([np.arange(0.1, 2, 0.03), np.arange(2, 20, 1)])}
   grid_cv = GridSearchCV(estimator=svm_model, param_grid=params, cv=10, n_jobs=-
   grid_cv.fit(x_train_scaled, y_train)
   print(grid_cv.best_params_)
```

{'C': 4.0}

Лучшая модель

```
In [ ]: best_svm_model = grid_cv.best_estimator_
    best_svm_model.fit(x_train_scaled, y_train)
    y_pred_svm = best_svm_model.predict(x_test_scaled)
    print_metrics(y_test, y_pred_svm)
```

weighted precision: 0.9750537634408601
weighted recall: 0.9733333333333334
weighted f1-score: 0.973171717171717



RandomForestClassifier

Базовая модель

```
In [ ]: rfc_model = RandomForestClassifier()
    rfc_model.fit(x_train, y_train)
    y_pred_rfc = rfc_model.predict(x_test)
    print_metrics(y_test, y_pred_rfc)
```



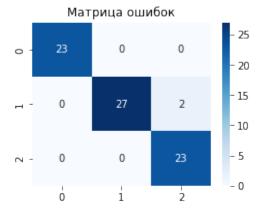
Подбор гиперпараметров

```
In [ ]: params = {'n_estimators': [5, 10, 50, 100], 'max_features': [2, 3, 4], 'criter
    grid_cv = GridSearchCV(estimator=rfc_model, param_grid=params, cv=10, n_jobs=-
    grid_cv.fit(x_train, y_train)
    print(grid_cv.best_params_)

{'criterion': 'gini', 'max_features': 2, 'min_samples_leaf': 2, 'n_estimators':
    5}
```

Лучшая модель

```
In [ ]: best_rfc_model = grid_cv.best_estimator_
    best_rfc_model.fit(x_train, y_train)
    y_pred_rfc = best_rfc_model.predict(x_test)
    print_metrics(y_test, y_pred_rfc)
```

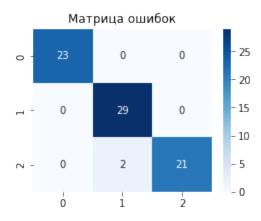


Сравнение результатов

```
In [ ]: print("SVC result\n")
    print_metrics(y_test, y_pred_svm)
```

SVC result

weighted precision: 0.9750537634408601
weighted recall: 0.9733333333333334
weighted f1-score: 0.973171717171717



```
In [ ]: print("RandomForestClassifier result\n")
    print_metrics(y_test, y_pred_rfc)
```

RandomForestClassifier result



Вывод

Оптимизация гиперпараметров позволила улучшить качество моделей по сравнению с базовыми версиями. Обе итоговые модели продемонстрировали исключительно высокую точность предсказаний, что связано **с простотой и небольшой размерностью** датасета. Согласно матрицам ошибок, каждая из моделей допустила всего 2 ошибки на 75 прогнозов, причем ошибочные предсказания у них не совпадали.

Результаты метрик оказались практически идентичными — различия наблюдаются лишь на уровне четвертого знака после запятой. Это свидетельствует о том, что в рамках данного датасета обе модели работают одинаково эффективно, и выбор между ними может основываться на других критериях, таких как скорость работы или интерпретируемость.