РК 1

По дисциплине «Технология машинного обучения»

Вариант 6

Выполнил:

ФИО Павлов Сергей Алексеевич

Рецензент:

ФИО Гапанюк Юрий Евгеньевич

2024

Задание: для заданного набора данных построить модели классификации. Для построения моделей использовать метод опорных векторов и метод случайного леса. Оценить качество моделей на основе подходящих метрик качества (не менее двух метрик).

Выполнение работы

Загрузим необходимые библиотеки. Также пропишем в тексте программы команду, позволяющую отображать графики в ячейках блокнота, и установим стиль графиков для отображения делений на осях графиков:

```
pip install scikit-learn
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
```

```
%matplotlib inline
sns.set(style="ticks")
```

Загрузим обучающую выборку:

```
original_train = pd.read_csv('data.csv', sep=",")
```

Сделаем дупликат выборки:

```
train = original_train.drop_duplicates()
```

Отобразим первые 5 строк датасета:

```
train.head()
```

Результат работы команды представлен на рисунке 1.

Рисунок 1 — Результат выполнения команды

Отобразим размер обучающего датасета:

```
train.shape
```

Результат работы команды представлен на рисунке 2.

```
(6819, 96)
```

Рисунок 2 — Результат выполнения команды

Представим список колонок с помощью команды:

```
train_subset = train.head(500)
train_subset.columns
```

Результат работы команды представлен на рисунке 3.

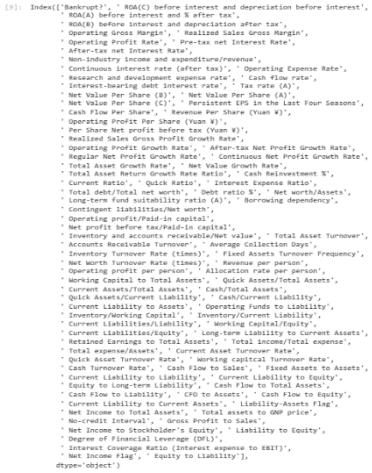

Рисунок 3 — Результат выполнения команды

Отобразим список колонок с типами данных:

train_subset.dtypes

Результат работы команды представлен на рисунке 4.

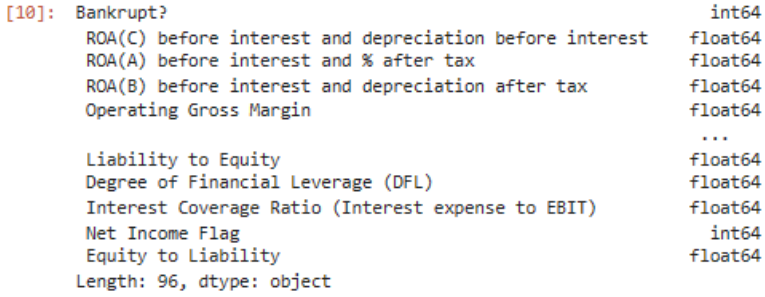

Рисунок 4 — Результат выполнения команды

Проверим, есть ли пропущенные значения:

train_subset.isnull().sum()

Результат работы команды представлен на рисунке 5.

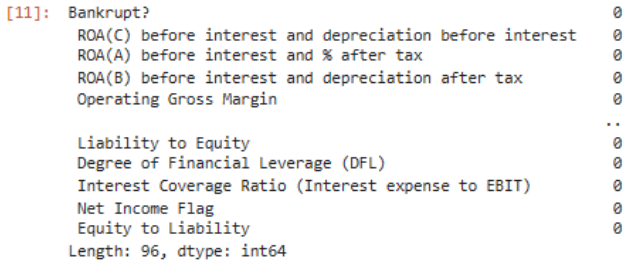

Рисунок 5 — Результат выполнения команды

Оценим дисбаланс классов для 'Bankrupt?':

fig, ax = plt.subplots(figsize=(2,2))

plt.hist(train['Bankrupt?'])

plt.show()

Результат работы команды представлен на рисунке 6.

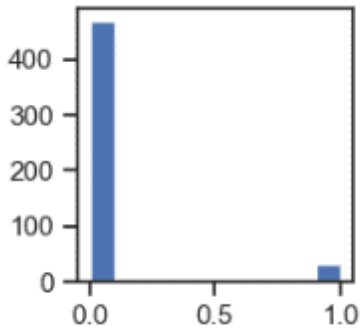

Рисунок 6 — Результат выполнения команды

Подсчитаем количество уникальных значений в столбце 'Bankrupt?:

train_subset['Bankrupt?'].value_counts()

Результат работы команды представлен на рисунке 7.

```
[14]: 0    469
      1     31
      Name: Bankrupt?, dtype: int64
```

Рисунок 7 — Результат выполнения команды

Подсчитаем дисбаланс классов:

```
total = train_subset.shape[0]
class_0, class_1 = train_subset['Bankrupt?'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
    .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

Результат работы команды представлен на рисунке 8.

```
Класс 0 составляет 93.8%, а класс 1 составляет 6.2%.
```

Рисунок 8 — Результат выполнения команды

Создадим вспомогательные колонки, чтобы наборы данных можно было разделить:

```
train_subset = train_subset.copy()
train_subset['dataset'] = 'TRAIN'
```

Колонки для объединения:

```
join_cols = ['Bankrupt?', ' ROA(C) before interest and depreciation before interest',
    ' ROA(A) before interest and % after tax',
```

' ROA(B) before interest and depreciation after tax', ' Liability to Equity',

' Degree of Financial Leverage (DFL)',

' Interest Coverage Ratio (Interest expense to EBIT)',

' Net Income Flag', ' Equity to Liability', 'dataset']

Проверим корректность объединения:

```
data_all = pd.concat([train_subset[join_cols]])
assert data_all.shape[0] == train_subset.shape[0]
```

Отобразим первые 5 строк:

```
data_all.head()
```

Результат работы команды представлен на рисунке 9.

[33]:

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Liability to Equity	Degree of Financial Leverage (DFL)	Interest Coverage Ratio (Interest expense to EBIT)	Net Income Flag	Equity to Liability	dataset
0	1	0.370594	0.424389	0.405750	0.290202	0.026601	0.564050	1	0.016469	TRAIN
1	1	0.464291	0.538214	0.516730	0.283846	0.264577	0.570175	1	0.020794	TRAIN
2	1	0.426071	0.499019	0.472295	0.290189	0.026555	0.563706	1	0.016474	TRAIN
3	1	0.399844	0.451265	0.457733	0.281721	0.026697	0.564663	1	0.023982	TRAIN
4	1	0.465022	0.538432	0.522298	0.278514	0.024752	0.575617	1	0.035490	TRAIN

Рисунок 9 — Результат выполнения команды

Числовые колонки для масштабирования:

```
scale_cols = ['Bankrupt?', ' ROA(C) before interest and depreciation before interest',
    ' ROA(A) before interest and % after tax',
    ' ROA(B) before interest and depreciation after tax',
    ' Interest Coverage Ratio (Interest expense to EBIT)', ' Equity to Liability']
```

Добавим масштабированные данные в набор данных:

```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data_all[scale_cols])
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data_all[new_col_name] = sc1_data[:,i]
```

Отобразим первые 5 строк:

```
data_all.head()
```

Результат работы команды представлен на рисунке 10.

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Liability to Equity	Degree of Financial Leverage (DFL)	Interest Coverage Ratio (Interest expense to EBIT)	Net income Flag	Equity to Liability	dataset	Bankrupt? scaled	ROA(C) before interest and depreciation before interest_scaled	ROA(A) before interest and % after tax_scaled	ROA(B) before interest and depreciation after tax_scaled	Interest Coverage Ratio (Interest expense to EBIT)_scaled	Equity to Liability_scaled
0	1	0.370594	0.424389	0.405750	0.260202	0.026001	0.564050	1	0.016469	TRAIN	1.0	0.570578	0.623531	0.602537	0.564050	0.037652
1	1	0.464291	0.538214	0.516730	0.283846	0.264577	0.570175	1	0.020794	TRAIN	1.0	0.746634	0.816810	0.793088	0.570175	0.059991
2	1	0.426071	0.499019	0.472295	0.290189	0.026555	0.563706	1	0.016474	TRAIN	1.0	0.674819	0.750255	0.716794	0.563706	0.037680
3	1	0.399844	0.451265	0.457733	0.281721	0.026697	0.564663	1	0.023982	TRAIN	1.0	0.625538	0.689166	0.691792	0.564663	0.076456
4	1	0.465022	0.538432	0.522298	0.278514	0.024752	0.575617	1	0.035490	TRAIN	1.0	0.748068	0.817180	0.802647	0.575617	0.135888

Рисунок 10 — Результат выполнения команды

Проверим, что масштабирование не повлияло на распределение данных:

```
for col in scale_cols:
    col_scaled = col + '_scaled'
    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data_all[col], 50)
    ax[1].hist(data_all[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

Результат работы команды представлен на рисунке 11.

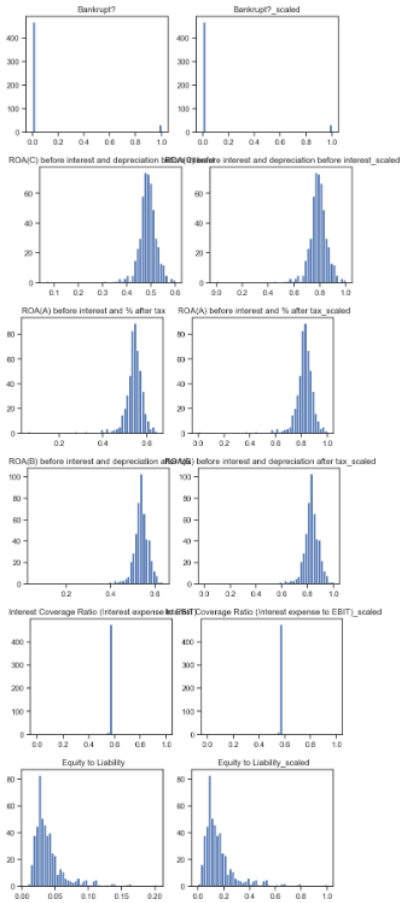

Рисунок 11— Результат выполнения команды

Воспользуемся наличием тестовых выборок, включив их в корреляционную матрицу:

```
corr_cols_1 = scale_cols + ['Bankrupt?']
corr_cols_1
```

Результат работы команды представлен на рисунке 12.

```
['Bankrupt?',
 ' ROA(C) before interest and depreciation before interest',
 ' ROA(A) before interest and % after tax',
 ' ROA(B) before interest and depreciation after tax',
 ' Interest Coverage Ratio (Interest expense to EBIT)',
 ' Equity to Liability',
 'Bankrupt?']
```

Рисунок 12 — Результат выполнения команды

Создадим новый список элементов:

```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['Bankrupt?']
corr_cols_2
```

Результат работы команды представлен на рисунке 13.

```
[46]: ['Bankrupt?_scaled',
       ' ROA(C) before interest and depreciation before interest_scaled',
       ' ROA(A) before interest and % after tax_scaled',
       ' ROA(B) before interest and depreciation after tax_scaled',
       ' Interest Coverage Ratio (Interest expense to EBIT)_scaled',
       ' Equity to Liability_scaled',
       'Bankrupt?']
```

Рисунок 13 — Результат выполнения команды

Исходные данные (до масштабирования):

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_all[corr_cols_1].corr(), annot=True, fmt='.2f')
ax.set_title('Исходные данные (до масштабирования)')
plt.show()
```

Результат работы команды представлен на рисунке 14.

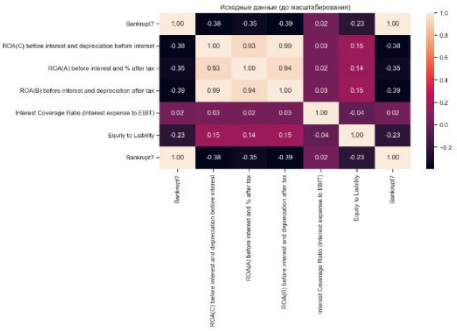

Рисунок 14— Результат выполнения команды

Масштабированные данные:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_all[corr_cols_2].corr(), annot=True, fmt='.2f')
ax.set_title('Масштабированные данные')
plt.show()
```

Результат работы команды представлен на рисунке 15.

Рисунок 15 — Результат выполнения команды

Выбор метрик для последующей оценки качества моделей. В качестве метрик для решения задачи классификации будем использовать: Метрики, формируемые на основе матрицы ошибок:

Метрика precision: Можно переводить как точность.

precision=TPTP+FP Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Используется функция precision_score.

Метрика recall (полнота): recall=TPTP+FN Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция recall_score.

Метрика F1 -мера Для того, чтобы объединить precision и recall в единую метрику используется Fβ -мера, которая вычисляется как среднее гармоническое от precision и recall:

Fβ=(1+β2)·precision·recallprecision+recall где β определяет вес точности в метрике.

На практике чаще всего используют вариант F1-меры (которую часто называют F-мерой) при β=1 :

F1=2·precision·recallprecision+recall Для вычисления используется функция f1_score.

Метрика ROC AUC Основана на вычислении следующих характеристик:

TPR=TPTP+FN

- True Positive Rate, откладывается по оси ординат. Совпадает с recall.

FPR=FPFP+TN

- False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

Идеальная ROC-кривая проходит через точки (0,0)-(0,1)-(1,1), то есть через верхний левый угол графика.

Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В качестве количественной метрики используется площадь под кривой - ROC AUC (Area Under the Receiver Operating Characteristic Curve). Чем ниже проходит кривая тем меньше ее площадь и тем хуже качество классификатора.

Для получения ROC AUC используется функция roc_auc_score.

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества:

```
class MetricLogger:
    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})
    def add(self, metric, alg, value):
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index,
inplace = True)
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
```

```
        self.df = self.df.append(temp, ignore_index=True)
    def get_data_for_metric(self, metric, ascending=True):
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

Для задачи классификации будем использовать следующие модели:

- Метод опорных векторов
- Случайный лес

Модели:

```
clas_models = {'SVC':SVC(probability=True),
               'RF':RandomForestClassifier()}
```

Сохранение метрик:

```
clasMetricLogger = MetricLogger()
```

Отрисовка ROC-кривой

```
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
```

```
                                pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    lw = 2
    ax.plot(fpr, tpr, color='darkorange',
            lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    ax.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_xlim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic')
    ax.legend(loc="lower right")
```

Создадим класс:

```
from sklearn.metrics import ConfusionMatrixDisplay
def clas_train_model(model_name, model, clas_X_train, clas_Y_train, clasMetricLogger):
    model.fit(clas_X_train, clas_Y_train)
    Y_pred = model.predict(clas_X_train)
    Y_pred_proba_temp = model.predict_proba(clas_X_train)
    Y_pred_proba = Y_pred_proba_temp[:,1]
    precision = precision_score(clas_Y_train, Y_pred)
    recall = recall_score(clas_Y_train, Y_pred)
    f1 = f1_score(clas_Y_train, Y_pred)
    roc_auc = roc_auc_score(clas_Y_train, Y_pred_proba)
    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)
    clasMetricLogger.add('f1', model_name, f1)
    clasMetricLogger.add('roc_auc', model_name, roc_auc)
    fig, ax = plt.subplots(ncols=2, figsize=(10,5))
    draw_roc_curve(clas_Y_train, Y_pred_proba, ax[0])
```

```
    cm = confusion_matrix(clas_Y_train, Y_pred)
    disp = ConfusionMatrixDisplay.from_estimator(model, clas_X_train, clas_Y_train, display_labels=['0','1'], cmap=plt.cm.Blues, normalize='true', ax=ax[1])
    fig.suptitle(model_name)
    plt.show()
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clas_X_train, clas_Y_train, clasMetricLogger)
```

Результат работы команды представлен на рисунке 16.

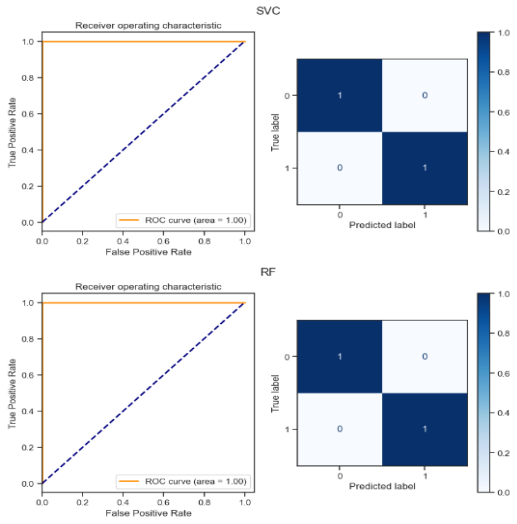

Рисунок 16— Результат выполнения команды

Метрики качества модели:

```
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

Результат работы команды представлен на рисунке 17.

```
[60]: array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

Рисунок 17 — Результат выполнения команды

Построим графики метрик качества модели:

```
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```

Результат работы команды представлен на рисунке 18.

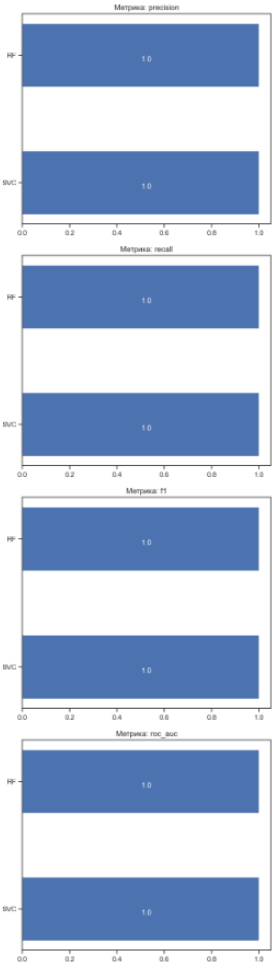

Рисунок 18— Результат выполнения команды

Вывод: на основании четырех метрик из четырех используемых, лучшими оказались модели случайного леса и дерево.