Титульный лист

**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.



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.

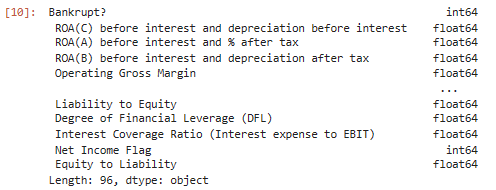


Рисунок 4 — Результат выполнения команды

Проверим, есть ли пропущенные значения:

train\_subset.isnull().sum()

Результат работы команды представлен на рисунке 5.

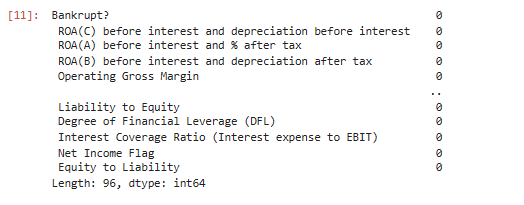


Рисунок 5 — Результат выполнения команды

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.

Подсчитаем количество уникальных значений в столбце 'Bankrupt?:

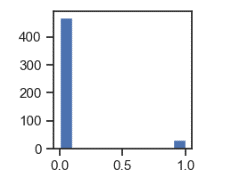


Рисунок 6 — Результат выполнения команды

train\_subset['Bankrupt?'].value\_counts()

Результат работы команды представлен на рисунке 7.

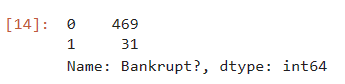


Рисунок 7 — Результат выполнения команды

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.



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.

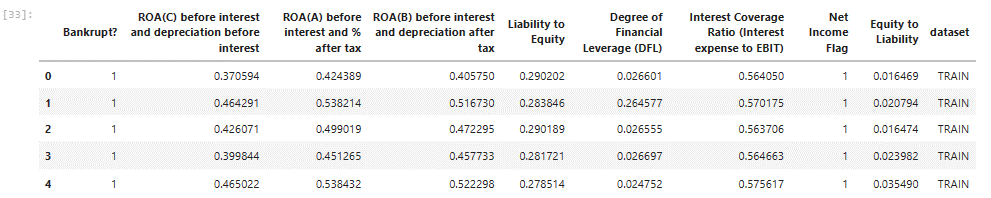


Рисунок 9 — Результат выполнения команды

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.



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)

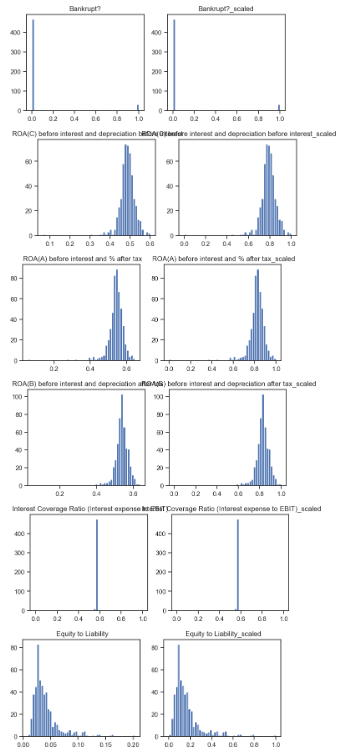
ax[0].title.set\_text(col)

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.

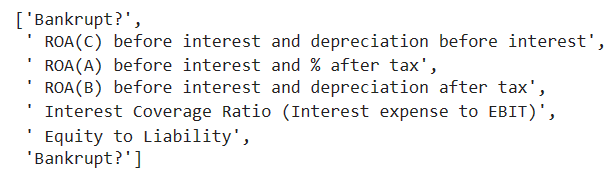


Рисунок 12 — Результат выполнения команды

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.

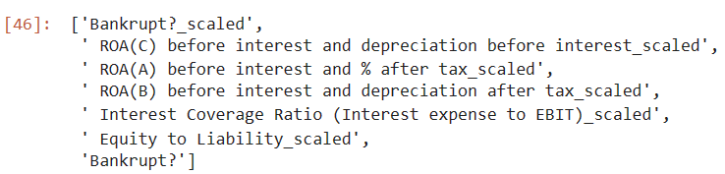


Рисунок 13 — Результат выполнения команды

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)

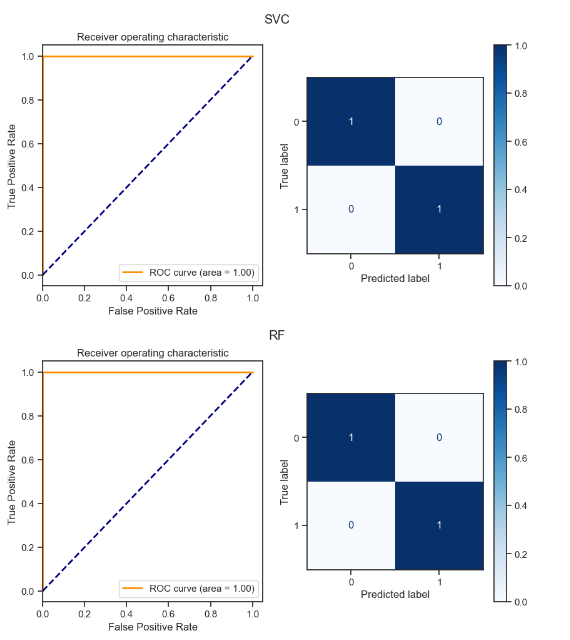
plt.show()

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.



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.

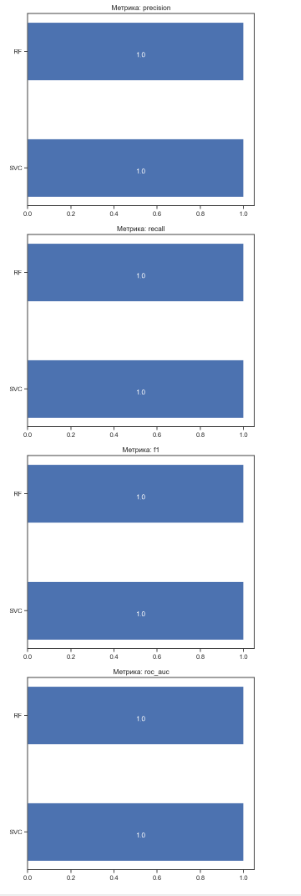


Рисунок 18— Результат выполнения команды

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