Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

Практическая работа № 3

по дисциплине "Математические основы анализа данных"

ТЕМА РАБОТЫ:

Построение модели «черный ящик»

Выполнил:

студент гр. ПРИм-124

Парахин К.В.

Принял:

Доцентр кафедры ИСПИ

Курочкин С.В.

Владимир 2024 г.

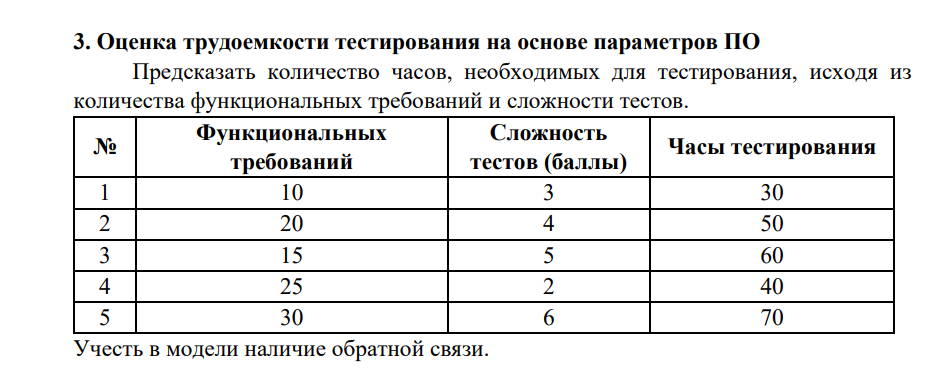
Цель работы:

Получить навыки применения аналитических методов построения модели системы «черный ящик»

Выполнение работы

Установим в Visual Studio Code расширение Jupiter, а также python сервер для работы с Jupiter Notebook. Также был установлен python и pip.

В соответствии с заданием в Jupiter Notebook разработать модель «черного ящика» и рассчитать прогнозные значения минимум одним способом из описанных в теоретическом обосновании.



Необходимо установить через pip пакеты pandas и scikit-learn

В работе проведем исследование, задав показатели, на основе которых будем изучать зависимость: а именно зависимость часов тестирования от функциональных требований и зависимость часов тестирования от сложности тестов.

import pandas as pd

import numpy as np

from typing import List, Tuple

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.base import RegressorMixin

import pickle

LEARN\_DATA\_FILE = 'learn\_data.csv'

MODEL\_GRADIENT\_FILE = 'model\_gradient.pkl'

MODEL\_REGRESSION\_FILE = 'model\_linear.pkl'

FEATURE = ['requirements', 'tests\_hardness']

Y = 'testing\_hours'

name\_mapping = {

    'requirements': 'Функциональных требований ',

    'tests\_hardness': 'Сложность тестов (баллы)',

    'testing\_hours': 'Часы тестирования',

    f'{Y}\_pred': 'Предсказание',

}

def with\_read\_cols(data: pd.DataFrame):

    data = data.copy().rename(columns=name\_mapping)

    return data

Зададим данные из таблицы в массив и сохраним в .csv файл. Далее будем использовать эти данные для использования в скрипте подсчета и визуализации сказанных зависимостей

data = [

    {'index': 1, 'requirements': 10, 'tests\_hardness': 3, 'testing\_hours': 30},

    {'index': 2, 'requirements': 20, 'tests\_hardness': 4, 'testing\_hours': 50},

    {'index': 3, 'requirements': 15, 'tests\_hardness': 5, 'testing\_hours': 60},

    {'index': 4, 'requirements': 25, 'tests\_hardness': 2, 'testing\_hours': 40},

    {'index': 5, 'requirements': 30, 'tests\_hardness': 6, 'testing\_hours': 70},

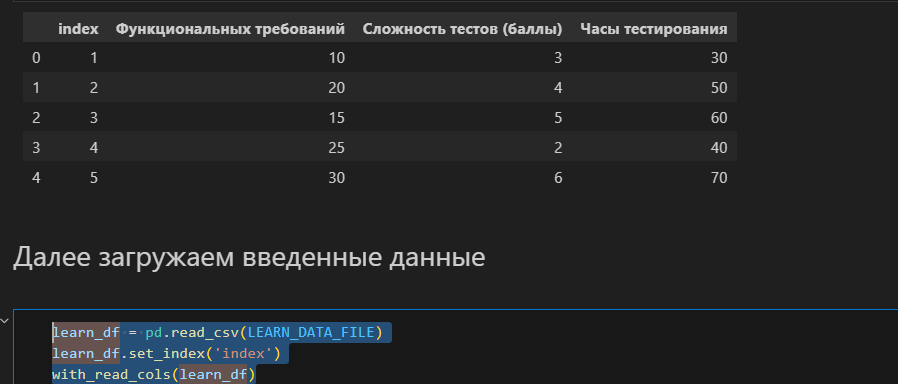
]

df = pd.DataFrame.from\_records(data)

df.set\_index('index')

df.to\_csv(LEARN\_DATA\_FILE, index=False)

with\_read\_cols(df)



Пробуем визуализировать заявленные зависимости часов тестирования от сепарированных друг от друга значений кол-ва функциональных требований и показателей сложности тестов

from matplotlib import pyplot as plt

from matplotlib.pyplot import subplots\_adjust

def get\_values\_xy\_for(feature: str) -> List[Tuple[int, int]]:

    result = []

    for \_, row in learn\_df.sort\_values(Y, axis=0).iterrows():

        x = row[feature]

        y = row[Y]

        result.append((x, y))

    return result

fig = plt.figure(num=1, facecolor='w', edgecolor='k')

fig.set\_figwidth(10)

fig.set\_figheight(10)

subplots\_adjust(hspace=0.3, wspace=0.3)

for index, feature in enumerate(FEATURE):

    data = get\_values\_xy\_for(feature=feature)

    yy = list(map(lambda x: x[0], data))

    xx = list(map(lambda x: x[1], data))

    ax = fig.add\_subplot(2, 2, index + 1)

    ax.plot(xx, yy, color='red', label=f'a1', linestyle='solid', marker='o')

    ax.set\_title('Зависимость от ' + '"' + f'{name\_mapping[feature]}' + '"')

    ax.set\_ylabel(name\_mapping[feature])

    ax.set\_xlabel(name\_mapping[Y])

    ax.grid()

    ax.legend()

plt.show()

Далее попробуем провести предсказания часов тестирования для тех же значения функциональных требований и сложности тестов с помощью метода градиентного спуска. Получаются достаточно точные результаты – квадрат ошибки около 1.

from sklearn.metrics import mean\_squared\_error

test\_df = learn\_df.copy()

X = learn\_df[FEATURE]

y = learn\_df[Y]

model = GradientBoostingRegressor(

    loss='squared\_error',

    learning\_rate=0.02,

    n\_estimators=200,

    criterion='friedman\_mse',

    max\_depth=5,

    min\_samples\_split=5,

)

model.fit(X, y)

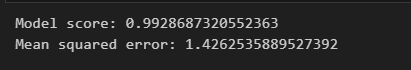
print(f"Model score: {model.score(X, y)}")

model\_predict = model.predict(test\_df[FEATURE])

print(f"Mean squared error: {mean\_squared\_error(test\_df[Y], model\_predict)}")

with open(MODEL\_GRADIENT\_FILE,'wb') as file:

    pickle.dump(model, file)



Далее сделаем анализ с использованием метода линейной регрессии. С ней не получилось добиться точных результатов – квадрат ошибки составляет целых 14 единиц.

from sklearn.metrics import mean\_squared\_error

test\_df = learn\_df.copy()

X = learn\_df[FEATURE]

y = learn\_df[Y]

model = LinearRegression(

    copy\_X=True,

    fit\_intercept=True,

    n\_jobs=-1,

    positive=False

)

model.fit(X, y)

print(f"Model score: {model.score(X, y)}")

print(f"Model coef: {model.coef\_}")

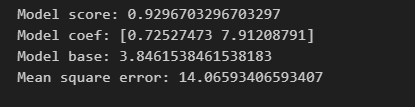
print(f"Model base: {model.intercept\_}")

model\_predict = model.predict(test\_df[FEATURE])

print(f"Mean square error: {mean\_squared\_error(test\_df[Y], model\_predict)}")

with open(MODEL\_REGRESSION\_FILE,'wb') as file:

    pickle.dump(model, file)



Далее на основе имеющихся реальных значений часов тестирования и предсказанных (с помощью градиентного спуска и линейной регрессии) - построим сравнительные линейчатые графики

from matplotlib import pyplot as plt

from matplotlib.pyplot import subplots\_adjust

Model1 = LinearRegression

Model2 = GradientBoostingRegressor

work\_df = learn\_df.copy()

with open(MODEL\_GRADIENT\_FILE, 'rb') as f:

    model1: Model1 = pickle.load(f)

with open(MODEL\_REGRESSION\_FILE, 'rb') as f:

    model2: Model2 = pickle.load(f)

predictions1 = model1.predict(work\_df[FEATURE])

predictions2 = model2.predict(work\_df[FEATURE])

show\_df = work\_df.copy()

show\_df[f"{Y}\_pred1"] = predictions1

show\_df[f"{Y}\_pred2"] = predictions2

fig = plt.figure(num=1, facecolor='w', edgecolor='k')

fig.set\_figwidth(10)

fig.set\_figheight(10)

x = list(show\_df.index)

y1 = show\_df[Y]

y2 = show\_df[f"{Y}\_pred1"]

y3 = show\_df[f"{Y}\_pred2"]

ax = fig.add\_subplot(2, 2, 1)

ax.plot(x, y1, color='red', label=f'реальные', marker='o', linestyle='-')

ax.plot(x, y2, color='green', label=f'предсказанные градиентом', marker='o', linestyle='-')

ax.plot(x, y3, color='blue', label=f'предсказанные регрессией', marker='o', linestyle='-')

ax.set\_title(f'Сравнение результатов: ')

ax.set\_xlabel('index')

ax.set\_ylabel(f'{Y}')

ax.grid()

ax.legend()

with\_read\_cols(show\_df)

Как видно на графиках, предсказанные показатели с помощью градиентного спуска реально очень близки и практически равны реальным.  
Показатели, предсказанные с помощью линейной регрессии имеют достаточно сильную погрешность – но в целом в некоторых точках все равно дискретно дает хорошую точность.

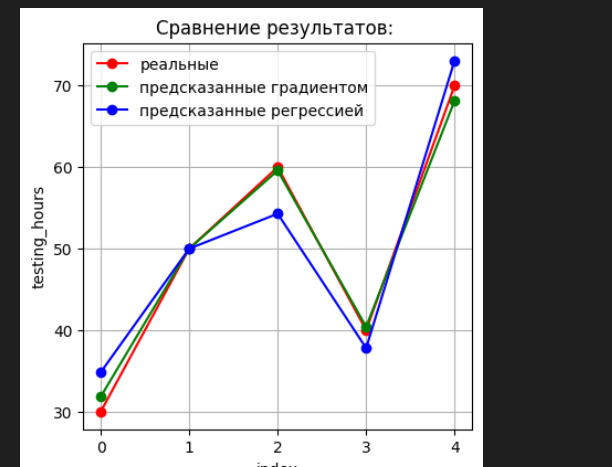


Рисунок 1. Сравнение реальных результатов и предсказанных разными способами

Вывод

В результате выполнения работы, я получил некоторые практические навыки применения аналитических методов построения модели системы «черный ящик»