|  |  |  |
| --- | --- | --- |
|  | |  |
| МИНОБРНАУКИ РОССИИ | |  |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** | |  |
|  | Институт информационных технологий (ИТ) | |
|  | Кафедра прикладной математики | |

|  |  |
| --- | --- |
| **Отчет по седьмой практической работе.** | |
| **по дисциплине** | |
| **«Технологии и инструментарий анализа больших данных»** | |
|  | |
| Выполнил студент группы ИКБО- 22-20 | Никулин К.В. |
| Принял | Парамонов А.А. |

Москва 2023 г.

СОДЕРЖАНИЕ

[1. ДАННЫЕ ДЛЯ РАБОТЫ 3](#_Toc152244118)

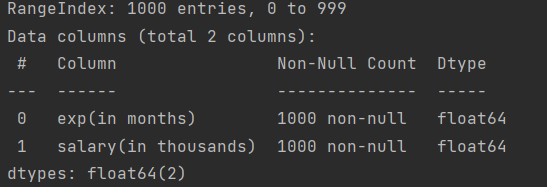
[2. BAGGING 4](#_Toc152244119)

[3. BOOSTING 6](#_Toc152244120)

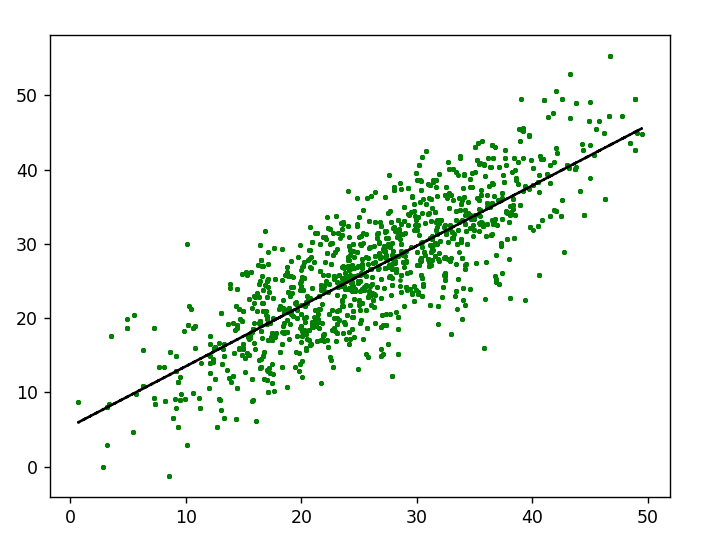
[ПРИЛОЖЕНИЕ 1 – КОД 7](#_Toc152244122)

1. ДАННЫЕ ДЛЯ РАБОТЫ

Набор данных содержит два параметра exp(опыт работы в месяцах) и salary(зарплата в тысячах). Возьмем параметр exp как целевой (зарплата работника зависит от опыта работы).



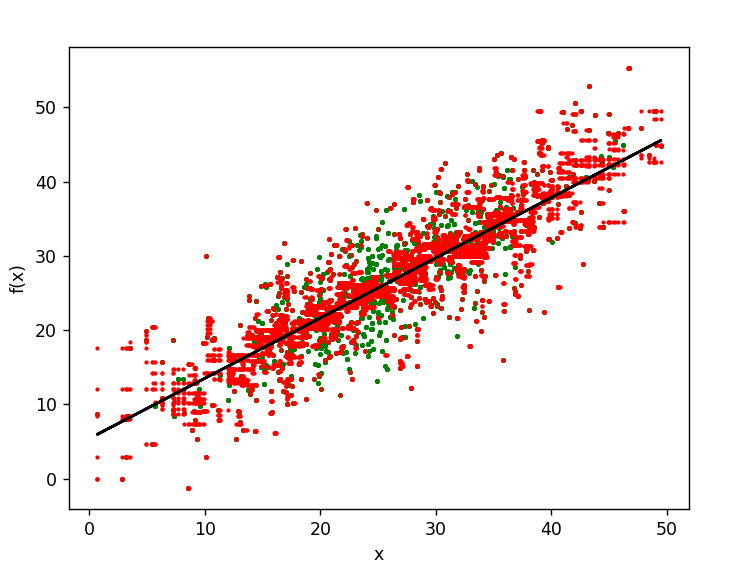
**Рисунок 1 – данные для работы**

****

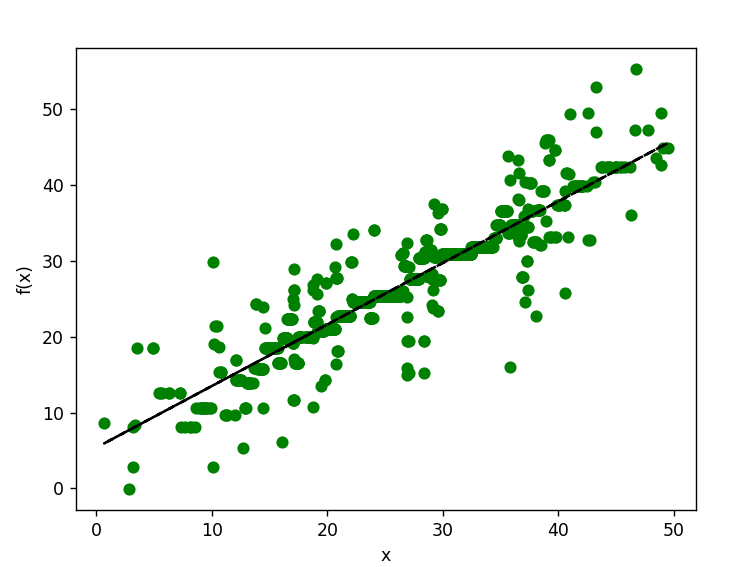
**Рисунок 2 – диаграмма рассеяния для данных**

1. BAGGING

Реализуем случайный лес:



**Рисунок 3 – результат случайного леса**



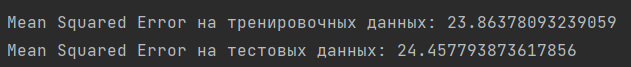
**Рисунок 4 – результат одного дерева**

Сравним результат работы случайного леса и одного дерева при помощи коэффициента детерминации.



**Рисунок 5 – оценка качества коэффициент детерминации**

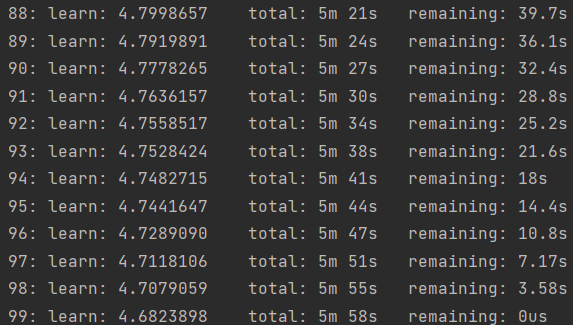
Видим, что случайный лес точнее, чем одно дерево. Также посчитаем MSE для дальнейшего сравнивания с Boosting.



**Рисунок 6 – оценка качества MSE**

1. BOOSTING

Для реализации Boosting построим последовательность из 100 моделей, обучая теми же данными что и в Bagging.



**Рисунок 7 – время работы Boosting**

Рассчитаем MSE для оценки качества:



Рисунок 8 – оценка качества MSE

ПРИЛОЖЕНИЕ 1 – КОД

Листинг А - Bagging

import pandas as pd  
import numpy as np  
import warnings  
import matplotlib.pyplot as plt  
from sklearn import tree  
from sklearn import ensemble  
from sklearn.datasets import fetch\_covtype  
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor  
from sklearn.metrics import r2\_score, f1\_score, mean\_squared\_error  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
import catboost as cb  
  
  
def f(x):  
 return 5.373 + 0.812 \* x  
  
  
def stacking(df, x):  
 x\_datasets = []  
 y\_datasets = []  
 y = f(x)  
 for i in range(10):  
 sampled\_data = df.sample(frac=0.8, replace=True, random\_state=i)  
 x\_datasets.append(np.array(sampled\_data["exp(in months)"]))  
 y\_datasets.append(np.array(sampled\_data["salary(in thousands)"]))  
 for i in range(10):  
 plt.scatter(x\_datasets[i], y\_datasets[i], c='green', s=3)  
 plt.plot(x, y, '--', color='black')  
 plt.show()  
 return x\_datasets, y\_datasets, y  
  
  
def bagging(x, y, x\_datasets, y\_datasets):  
 models = []  
 for i in range(10):  
 model\_tree = tree.DecisionTreeRegressor(max\_depth=8, random\_state=1)  
 model\_tree.fit(x\_datasets[i].reshape(-1, 1), y\_datasets[i])  
 models.append(model\_tree)  
 y\_pred = []  
 for i in range(len(models)):  
 y\_pred.append(models[i].predict(x.reshape(-1, 1)))  
 plt.xlabel('x')  
 plt.ylabel('f(x)')  
 for i in range(10):  
 plt.scatter(x, y\_pred[i], c='red', s=2)  
 plt.plot(x, y, color='black')  
 plt.show()  
  
 mean\_pred = np.array(y\_pred).mean(axis=0)  
 plt.xlabel('x')  
 plt.ylabel('f(x)')  
 plt.scatter(x, mean\_pred, c='green', zorder=2)  
 plt.plot(x, y, '--', color='black', lw=1)  
 plt.show()  
  
 model\_tree = tree.DecisionTreeRegressor(max\_depth=8, random\_state=1)  
 one\_model = model\_tree.fit(np.array(x\_datasets).reshape(-1, 1), np.array(y\_datasets).reshape(-1, 1))  
 one\_pred = one\_model.predict(x.reshape(-1, 1))  
 plt.xlabel('x')  
 plt.ylabel('f(x)')  
 plt.scatter(x, one\_pred, c='green', zorder=2)  
 plt.plot(x, y, '--', color='black', lw=1.5)  
 plt.show()  
 print('Коэффициент детерминации для случайного леса: ', r2\_score(mean\_pred, y))  
 print('Коэффициент детерминации для одного дерева решений: ', r2\_score(one\_pred, y))  
 return  
  
  
def boosting(df):  
 predictors = np.array(df['exp(in months)'])  
 target = np.array(df['salary(in thousands)'])  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(predictors.reshape(-1, 1), target, test\_size=0.2, random\_state=0)  
 print('Размер для признаков обучающей выборки', x\_train.shape, '\n',  
 'Размер для признаков тестовой выборки', x\_test.shape, '\n')  
 random\_forest = RandomForestRegressor()  
 params\_grid = {  
 "max\_depth": [12, 18],  
 "min\_samples\_leaf": [3, 10],  
 "min\_samples\_split": [6, 12]  
 }  
  
 grid\_search\_random\_forest = GridSearchCV(estimator=random\_forest, param\_grid=params\_grid,  
 scoring="neg\_mean\_squared\_error", cv=2)  
 grid\_search\_random\_forest.fit(x\_train, y\_train)  
 best\_model = grid\_search\_random\_forest.best\_estimator\_  
 y\_preds\_train = best\_model.predict(x\_train)  
 mse\_train = mean\_squared\_error(y\_train, y\_preds\_train)  
 print('Mean Squared Error на тренировочных данных:', mse\_train)  
 y\_preds\_test = best\_model.predict(x\_test)  
 mse\_test = mean\_squared\_error(y\_test, y\_preds\_test)  
 print('Mean Squared Error на тестовых данных:', mse\_test)  
  
 model\_catboost\_clf = cb.CatBoostClassifier(iterations=100,  
 task\_type="GPU",  
 devices='0')  
 model\_catboost\_clf.fit(x\_train, y\_train)  
 y\_preds\_t = model\_catboost\_clf.predict(x\_train, task\_type="CPU")  
 mse\_train = mean\_squared\_error(y\_train, y\_preds\_t)  
 print('Mean Squared Error на тренировочных данных:', mse\_train)  
 y\_preds\_tt = model\_catboost\_clf.predict(x\_test, task\_type="CPU")  
 mse\_test = mean\_squared\_error(y\_test, y\_preds\_tt)  
 print('Mean Squared Error на тестовых данных:', mse\_test)  
 return  
  
  
def main():  
 df = pd.read\_csv('Experience-Salary.csv')  
 x = np.array(df['exp(in months)'])  
 y = np.array(df['salary(in thousands)'])  
 print(df.info())  
 x\_datasets, y\_datasets, y = stacking(df, x)  
 bagging(x, y, x\_datasets, y\_datasets)  
 boosting(df)  
 return  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()