Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №10**

«Градиентный бустинг»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2019

ХОД РАБОТЫ

**Задание.**

Для выполнения задания используйте набор данных boston из библиотеки sklearn: https://scikit-learn.org/stable/datasets/index.html#boston-dataset

1. Загрузите данные с помощью библиотеки sklearn.
2. Разделите выборку на обучающую (75%) и контрольную (25%).
3. Заведите массив для объектов DecisionTreeRegressor (они будут использоваться в качестве базовых алгоритмов) и для вещественных чисел (коэффициенты перед базовыми алгоритмами).
4. В цикле обучите последовательно 50 решающих деревьев с параметрами max\_depth=5 и random\_state=42 (остальные параметры - по умолчанию). Каждое дерево должно обучаться на одном и том же множестве объектов, но ответы, которые учится прогнозировать дерево, будут меняться в соответствие с отклонением истинных значений от предсказанных.
5. Попробуйте всегда брать коэффициент равным 0.9. Обычно оправдано выбирать коэффициент значительно меньшим - порядка 0.05 или 0.1, но на стандартном наборе данных будет всего 50 деревьев, возьмите для начала шаг побольше.
6. В процессе реализации обучения вам потребуется функция, которая будет вычислять прогноз построенной на данный момент композиции деревьев на выборке X. Реализуйте ее. Эта же функция поможет вам получить прогноз на контрольной выборке и оценить качество работы вашего алгоритма с помощью mean\_squared\_error в sklearn.metrics.
7. Попробуйте уменьшать вес перед каждым алгоритмом с каждой следующей итерацией по формуле 0.9 / (1.0 + i), где i - номер итерации (от 0 до 49). Какое получилось качество на контрольной выборке?
8. Исследуйте, переобучается ли градиентный бустинг с ростом числа итераций, а также с ростом глубины деревьев. Постройте графики. Какие выводы можно сделать?
9. Сравните качество, получаемое с помощью градиентного бустинга с качеством работы линейной регрессии. Для этого обучите LinearRegression из sklearn.linear\_model (с параметрами по умолчанию) на обучающей выборке и оцените для прогнозов полученного алгоритма на тестовой выборке RMSE.

**Результат выполнения:**

1. Код выгрузки данных:

boston = datasets.load\_boston()

X, Y = boston.data, boston.target

1. Код реализации:

# ###################### QUITE IMPORTANT ##########################

# overall MSE really depends on factor how do we split our data

# so changing `random\_state` may lead to various results.

# In order to have the same results I set seed to value equal to 51

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=51)

3-7. Код реализации:

mocked\_coefficient\_quality = check\_quality(X\_train, y\_train, X\_test, y\_test, mock\_coefficient)()

print(mocked\_coefficient\_quality)

Результат выполнения:

5.569226762770745

8. Код реализации:

sequences\_coefficient\_quality = check\_quality(X\_train, y\_train, X\_test, y\_test, calculate\_coefficient)()

print(sequences\_coefficient\_quality)

Результат выполнения:

5.3607371501561625

Ошибка ументшилась, однако относительно несущественно. Посмотрим как будет меняться ошибка в зависимости от глубины и количества деревьев. Стоит сразу отметить, что реализация в следующем пункет была сделана в несколько этапов. Сначал я использовал градиентный бустинг, который был «самописный» (тот, что был реализован в первых пунктах с динамическим коэффициентом). Однако, после этого я решил попробовать использовать библиотечную реализацию, и результаты уже были совершенно другими и более ожидаемыми.

9. Код реализации:

# since Intel i7 6700HQ has 8 threads

trees\_number = [50, 100, 150, 200, 250, 300, 400, 500]

params = []

for i in trees\_number:

params.append(DecisionTreeParams(X\_train, y\_train, X\_test, y\_test, calculate\_coefficient, i, 5))

pool = Pool(len(trees\_number))

# change job\_gb to job for getting my own implementation of gradient boosting

errors = pool.map(job\_gb, params)

depth = [2, 4, 6, 8, 10, 13, 17, 20]

params = []

for i in depth:

params.append(DecisionTreeParams(X\_train, y\_train, X\_test, y\_test, calculate\_coefficient, 50, i))

# change job\_gb to job for getting my own implementation of gradient boosting

errors = pool.map(job\_gb, params)

Результат выполнения:

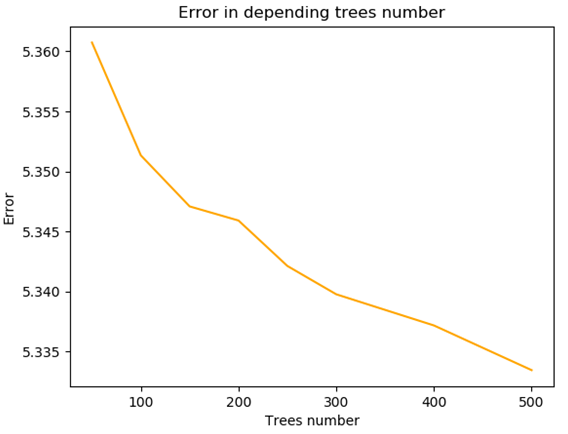


Рисунок 1 – график зависимости ошибки от количества деревьев (не библиотечная реализация)

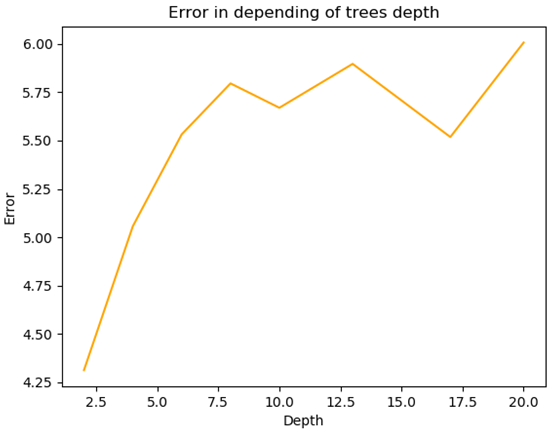


Рисунок 2 – Зависимость ошибки от глубины дерева (не библиотчечная реализация)

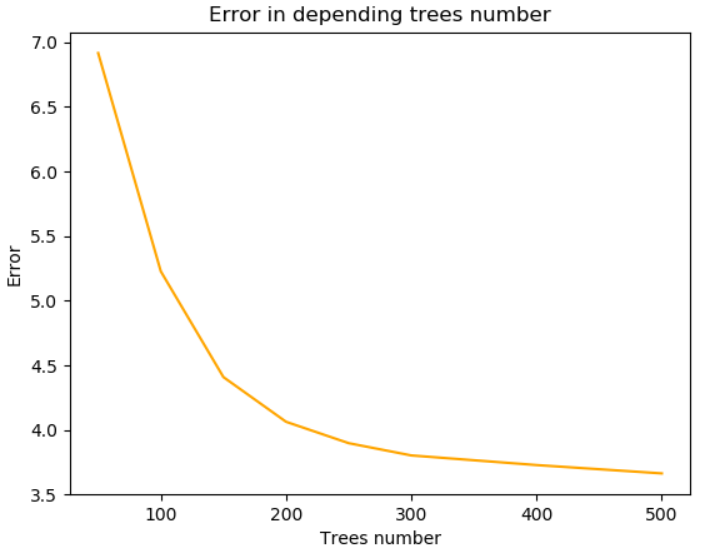


Рисунок 3 – зависимость ошибки от количества деревьев (библиотечная реализация)

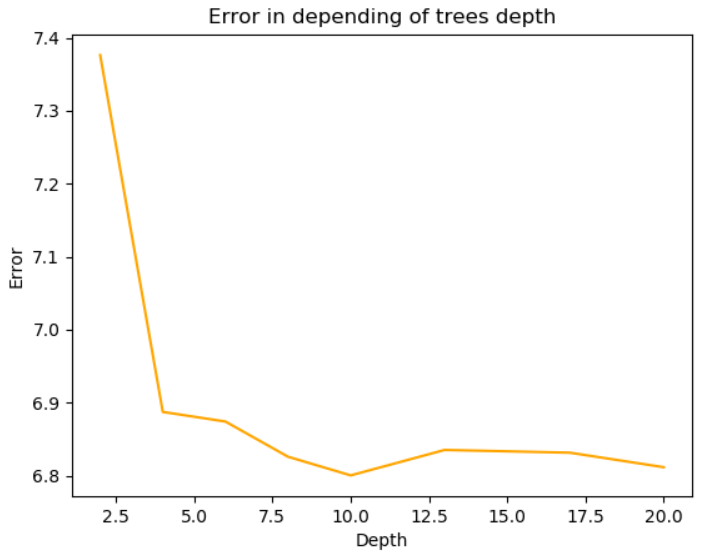


Рисунок 4 – зависимость ошибки од глубины дерева (библиотечная реализация)

На двух графиках с библиотечной реализацией видно, что ошибка уменьшается с ростом глубины дерева и увеличением количества деревьев, что свидетельствует о переобучении модели.

11. Код реализации:

reg = LinearRegression().fit(X\_train, y\_train)

pred = reg.predict(X\_test)

print('Linear regression MSE: ', np.sqrt(mean\_squared\_error(y\_test, pred)))

Результат выполнения:

('Linear regression MSE: ', 4.863502400380662)

Видно, что ошибка меньше, чем у начальной реализации градиентного бустинга. Однако очевидно, что, скорее всего, линейная регресси не сможет восстановить сложности всех данных, и я бы остановился на алгоритме градиентного бустинга. Хотя добавление полиномиальных фич в линейную регрессию, возможно, уменьшило бы итоговую ошибку.

**Программный код:**

"""

File -> Settings -> Tools -> Python Scientific -> uncheck mark

Run with python 3.7.5 (64 bit)

"""

from \_\_future\_\_ import division

from sklearn import datasets

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

import numpy as np

import matplotlib.pyplot as plt

from goto import with\_goto

from sklearn.linear\_model import LinearRegression

from multiprocessing import Pool

from sklearn import ensemble

def gradient(Y, X, base\_algorithms\_list, coefficients\_list):

return Y - predict(X, base\_algorithms\_list, coefficients\_list)

def predict(X, base\_algorithms\_list, coefficients\_list):

return [sum([coeff \* algo.predict([x])[0] for algo, coeff in zip(base\_algorithms\_list, coefficients\_list)])

for x in X]

def check\_quality(X\_train, y\_train, X\_test, y\_test, get\_coefficient):

def get\_model(trees\_number=50, depth=5):

print(trees\_number, depth)

# 4

base\_algorithms\_list = []

coefficients\_list = []

# 5

for i in range(0, trees\_number):

# create new algorithm

rg = DecisionTreeRegressor(random\_state=42, max\_depth=depth)

# fit algo in train dataset and new target

rg.fit(X\_train, gradient(y\_train, X\_train, base\_algorithms\_list, coefficients\_list))

# append results

base\_algorithms\_list.append(rg)

# ======= 6 =======

coefficients\_list.append(get\_coefficient(i))

# 7

pred = predict(X\_test, base\_algorithms\_list, coefficients\_list)

print(np.sqrt(mean\_squared\_error(y\_test, pred)))

return np.sqrt(mean\_squared\_error(y\_test, pred))

return get\_model

def mock\_coefficient(i):

return 0.9

def calculate\_coefficient(i):

return 0.9 / (1.0 + i)

class DecisionTreeParams():

def \_\_init\_\_(self, X\_train, y\_train, X\_test, y\_test, coefficients, number\_trees, depth):

"""Constructor"""

self.X\_train = X\_train

self.y\_train = y\_train

self.X\_test = X\_test

self.y\_test = y\_test

self.coefficients = coefficients

self.number\_trees = number\_trees

self.depth = depth

def worker(self):

return check\_quality(self.X\_train, self.y\_train, self.X\_test, self.y\_test, self.coefficients)(self.number\_trees, self.depth)

def run\_gb(self):

params = {'n\_estimators': self.number\_trees, 'max\_depth': self.depth, 'min\_samples\_split': 2,

'learning\_rate': 0.01, 'loss': 'ls'}

clf = ensemble.GradientBoostingRegressor(\*\*params)

clf.fit(self.X\_train, self.y\_train)

return np.sqrt(mean\_squared\_error(self.y\_test, clf.predict(self.X\_test)))

def job(A):

return A.worker()

def job\_gb(A):

return A.run\_gb()

@with\_goto

def main():

goto .task

label .task

# 1

boston = datasets.load\_boston()

X, Y = boston.data, boston.target

# 2

# ###################### QUITE IMPORTANT ##########################

# overall MSE really depends on factor how do we split our data

# so changing `random\_state` may lead to various results.

# In order to have the same results I set seed to value equal to 51

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=51)

# 3

mocked\_coefficient\_quality = check\_quality(X\_train, y\_train, X\_test, y\_test, mock\_coefficient)()

print(mocked\_coefficient\_quality)

# 8

sequences\_coefficient\_quality = check\_quality(X\_train, y\_train, X\_test, y\_test, calculate\_coefficient)()

print(sequences\_coefficient\_quality)

# 9

# since Intel i7 6700HQ has 8 threads

trees\_number = [50, 100, 150, 200, 250, 300, 400, 500]

params = []

for i in trees\_number:

params.append(DecisionTreeParams(X\_train, y\_train, X\_test, y\_test, calculate\_coefficient, i, 5))

pool = Pool(len(trees\_number))

# change job\_gb to job for getting my own implementation of gradient boosting

errors = pool.map(job\_gb, params)

plt.plot(trees\_number, errors, color='orange')

plt.xlabel('Trees number')

plt.ylabel('Error')

plt.title('Error in depending trees number')

plt.show()

depth = [2, 4, 6, 8, 10, 13, 17, 20]

params = []

for i in depth:

params.append(DecisionTreeParams(X\_train, y\_train, X\_test, y\_test, calculate\_coefficient, 50, i))

# change job\_gb to job for getting my own implementation of gradient boosting

errors = pool.map(job\_gb, params)

plt.plot(depth, errors, color='orange')

plt.xlabel('Depth')

plt.ylabel('Error')

plt.title('Error in depending of trees depth')

plt.show()

# 10

reg = LinearRegression().fit(X\_train, y\_train)

pred = reg.predict(X\_test)

print('Linear regression MSE: ', np.sqrt(mean\_squared\_error(y\_test, pred)))

if \_\_name\_\_ == '\_\_main\_\_':

main()