|  |  |  |  |
| --- | --- | --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования  Белорусский Государственный Университет Информатики и Радиоэлектроники | | | |
| Факультет | Компьютерных сетей и систем | | |
| Кафедра | Информатики  Дисциплина: Конструирование те технологии электронных вычислительных средств | | |
|  |  | | |
| **лабораторная работа №10**  по курсу Машинное обучение  на тему  **Градиентный бустинг** | | | |
| Магистрант:  гр. 858341  Кудин Н.И. | |  | Проверил:  Стержанов М.В. |
| Минск, 2019 | | | |

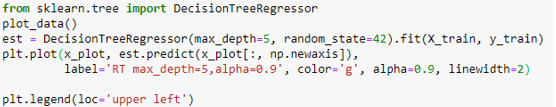
**Постановка задачи**

Для выполнения задания используйте набор данных boston из библиотеки sklearn

<https://scikit-learn.org/stable/datasets/index.html#boston-dataset>

**Ход выполнения работы**

1. Загрузим данные с помощью библиотеки sklearn.
2. Разделим выборку на обучающую (75%) и контрольную (25%).
3. Заведем массив для объектов DecisionTreeRegressor.
4. Будем использовать коэффициент равным 0.9 и обучим модель 50 решающих деревьев с параметрами max\_depth=5 и random\_state=42 (остальные параметры - по умолчанию). Функция обучения:



1. Реализуем функцию, которая позволит вычислять прогноз построенной на данный момент композиции деревьев на выборке.

def gbm\_predict(X):

return [sum([coeff \* algo.predict([x])[0] for algo, coeff in zip(basics, ws)]) for x in X]

5.1 Результат работы функции отобразим на рисунке 2. Полученное значение ошибки: 5.45.

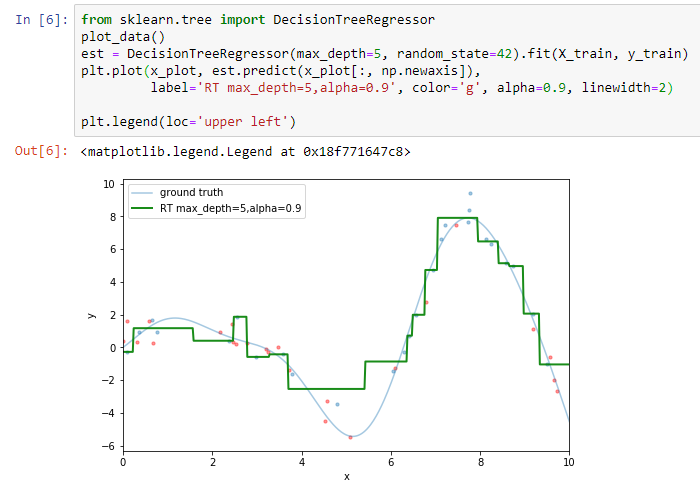


Рисунок 2 – Результат обучения и прогноза

1. Попробуем уменьшать вес перед каждым алгоритмом с каждой следующей итерацией по формуле 0.9 / (1.0 + i), где i - номер итерации (от 0 до 49).

basics = []

ws = []

y\_cur = np.array(y\_train)

for i in range(50):

regr = tree.DecisionTreeRegressor(max\_depth=5, random\_state=42).fit(X\_train, y\_cur)

basics.append(regr)

ws.append(0.9/(1.0+i%10))

y\_cur = y\_train - gbm\_predict(X\_train)

Полученное значение ошибки уменьшилось и стало равняться: 4.929.

1. Исследуем, переобучается ли градиентный бустинг с ростом числа итераций, а также с ростом глубины деревьев.

trees = np.array([5, 10, 15, 20, 30, 40, 50, 60, 70, 100])

depths = np.array([3, 5, 7, 10, 15, 22])

pyplot.figure(figsize=(16, 11))

i = 1

for d in depths:

scores\_train = []

scores\_test = []

for num in trees:

boost = GradientBoostingRegressor(n\_estimators=num, max\_depth=d, random\_state=42).fit(X\_train, y\_train)

scores\_train.append(np.sqrt(metrics.mean\_squared\_error(y\_train, boost.predict(X\_train))))

scores\_test.append(np.sqrt(metrics.mean\_squared\_error(y\_test, boost.predict(X\_test))))

print(scores\_test[:3])

ax = plt.subplot(5, 5, i)

ax.set\_title("forest {} depth".format(d))

pyplot.plot(trees, scores\_train, color="red", label="train")

pyplot.plot(trees, scores\_test, color="blue", label="test")

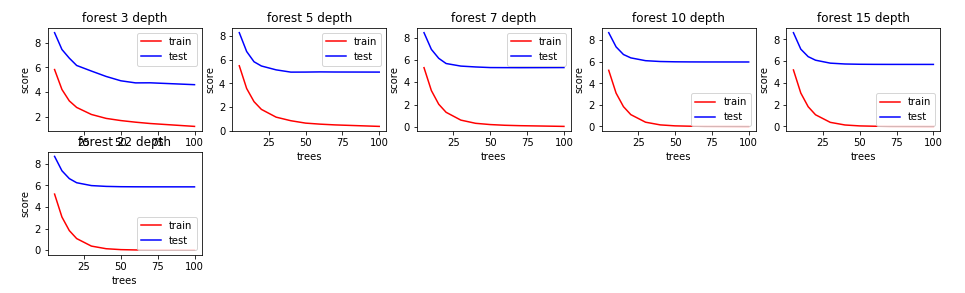
pyplot.xlabel("trees")

pyplot.ylabel("score")

pyplot.legend()

i += 1

Результат изменения параметров градиентного бустинга



1. Сравним качество, получаемое с помощью градиентного бустинга с качеством работы линейной регрессии. Будем использовать следующий код:

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

pred = lin.predict(X\_test)

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

Значение ошибки = 8.25, что означает худшее качество работы линейной регрессии в данном случае.

**Заключение**

Сравнив работу градиентного бустинга и линейной регрессии можно сделать следующие выводы:

1. С увеличением числа деревьев, начиная с некоторого момента, качество работы градиентного бустинга существенно не меняется.
2. С увеличением числа деревьев, начиная с некоторого момента, градиентный бустинг начинает переобучаться.
3. С ростом глубины деревьев, начиная с некоторого момента, качество работы градиентного бустинга на тестовой выборке начинает ухудшаться.
4. С ростом глубины деревьев, начиная с некоторого момента, качество работы градиентного бустинга перестает существенно изменяться.
5. Для данной выборки линейная регрессии работает хуже, однако на других выборках могут быть противоположные случаи.