Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет информатики и радиоэлектроники”

Факультет компьютерных систем и сетей

Специальность “Обработка больших объемов информации”

Лабораторная работа №10

**Градиентный бустинг**

Выполнил:

магистрант гр. 858641 Кальман В.А.

Проверил:

Стержанов М. В.

Минск 2019

**Данные.**

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.**

boston = datasets.load\_boston()

X = boston.data

y = boston.target

**2. Разделите выборку на обучающую (75%) и контрольную (25%)**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

**3-6. 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:**

tree = DecisionTreeRegressor()

tree\_cv = GridSearchCV(tree, {"max\_depth": [5, None], "random\_state": [42, None]})

abr = AdaBoostRegressor(tree\_cv, n\_estimators=50)

model: AdaBoostRegressor = abr.fit(X\_train, y\_train, sample\_weight=[0.9])

y\_pred\_4 = model.predict(X\_test)

*# Compute and print the metrics*

abr\_r2 = model.score(X\_test, y\_test)

abr\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_4))

**print**('score = ', abr\_r2)

**print**('RMSE = ', abr\_rmse)

Результат выполнения:

score = -0.6131443651273549

RMSE = 12.973577023604681

**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?**

sample\_weights = list(map(**lambda** i: [0.9 / (1 + i)], range(0, 50)))

model: AdaBoostRegressor = abr.fit(X\_train, y\_train)

y\_pred\_4 = model.predict(X\_test)

abr\_r2 = model.score(X\_test, y\_test)

abr\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_4))

**print**('score = ', abr\_r2)

**print**('RMSE = ', abr\_rmse)

Результат выполнения:

score = 0.8438016903430736

RMSE = 4.037023539083976

Ошибка немного уменьшилась.

**8. Исследуйте, переобучается ли градиентный бустинг с ростом числа итераций, а также с ростом глубины деревьев. Постройте графики. Какие выводы можно сделать?**

num\_iters = 50

J\_history = []

J\_test\_history = []

**for** i **in** range(num\_iters):

abr = AdaBoostRegressor(tree\_cv, n\_estimators=50, learning\_rate=1 / (1 + i))

model: AdaBoostRegressor = abr.fit(X\_train, y\_train)

y\_pred\_train = model.predict(X\_train)

abr\_rmse\_train = np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train))

y\_pred\_test = model.predict(X\_test)

abr\_rmse\_test = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test))

J\_history.append(abr\_rmse\_train)

J\_test\_history.append(abr\_rmse\_test)

plt.plot(range(num\_iters), J\_history, '-r', LineWidth=2)

plt.plot(range(num\_iters), J\_test\_history, '-b', LineWidth=2)

plt.ylabel('RMSE')

plt.xlabel('Iterations')

plt.legend(['Train', 'Test'])

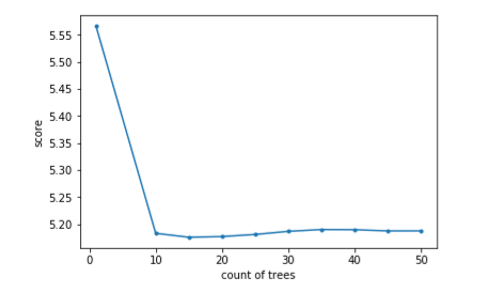


Рисунок 1 – график зависимости ошибки от количества деревьев (не библиотечная реализация)

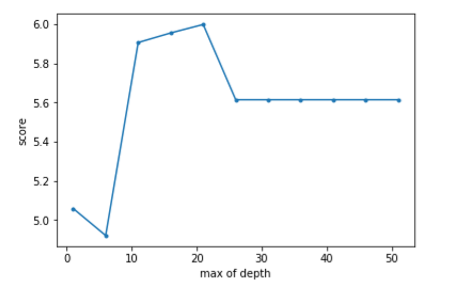


Рисунок 2 – Зависимость ошибки от глубины дерева (не библиотчечная реализация)

С ростом кол-ва деревьев cost function снижается. Увеличивается сложность вычислений. С ростом depth деревьев решений cost function падает до определённого момента, а потом начинается overfitting.

**11. 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:**

reg = LinearRegression()

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

y\_pred\_1 = reg.predict(X\_test)

r2 = reg.score(X\_test, y\_test)

OLS\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_1))

**print**(f'GB score = {abr\_r2}, LR score = {r2}')

**print**(f'GB RMSE = {abr\_rmse}, LR RMSE = {OLS\_rmse}')

Результат выполнения:

GB score = 0.8438016903430736, LR score = 0.7450343040728142

GB RMSE = 4.037023539083976, LR RMSE = 5.157794006761

Градиентый бустинг показал лучший результат. RMSE градиентный бустинг = 0.84 RMSE линейная регрессия = 0.74

**Вывод**

Градиентный бустинг — это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно Это техника использует идею о том, что следующая модель будет учится на ошибках предыдущей. Они имеют неравную вероятность появления в последующих моделях, и чаще появятся те, что дают наибольшую ошибку. Предсказатели могут быть выбраны из широкого ассортимента моделей, например, деревья решений, регрессия, классификаторы и т.д. Из-за того, что предсказатели обучаются на ошибках, совершенных предыдущими, требуется меньше времени для того, чтобы добраться до реального ответа. Но мы должны выбирать критерий остановки с осторожностью, иначе это может привести к переобучению.