Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра информатики

Лабораторная работа №10
Градиентный бустинг

Выполнил: Полевой Александр Вадимович
магистрант кафедры информатики
группа № 858641

Проверил: Стержанов Максим Валерьевич

Минск 2019

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

1. Загрузите данные с помощью библиотеки sklearn.

In [1]:

```
import numpy as np
from sklearn import datasets, tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data = datasets.load_boston()
X = data['data']
y = data['target']

X.shape, y.shape
```

Out[1]:

```
((506, 13), (506,))
```

In [2]:

```
data.keys()
```

Out[2]:

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [3]:

```
data.feature_names
```

Out[3]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RA
D',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [4]:

```
print(data.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median V
alue (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over
25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds r
iver; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1
940
        - DIS      weighted distances to five Boston employment centre
s
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of bla
cks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (ht
tps://archive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at
Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedon
ic
prices and the demand for clean air', J. Environ. Economics & Manageme
nt,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diag
nostics
...', Wiley, 1980.   N.B. Various transformations are used in the tabl
e on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning pap
ers that address regression
```

```
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influ
ential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learn
ing. In Proceedings on the Tenth International Conference of Machine L
earning, 236-243, University of Massachusetts, Amherst. Morgan Kaufman
n.
```

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

In [24]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_sta
```

In [25]:

```
X_train.shape, X_test.shape
```

Out[25]:

```
((379, 13), (127, 13))
```

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.

In [26]:

```
def gb_predict(X, trees, coeffs):
    m,n = X.shape
    iterations = len(trees)
    result = np.zeros(m)

    for i in range(iterations):
        result += coeffs[i] * trees[i].predict(X)

    return result
```

In [27]:

```
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

In [28]:

```
trees_1 = []
coeffs_1 = []

iter = 50
last_y_pred_1 = y_train

for i in range(iter):
    model = tree.DecisionTreeRegressor(max_depth=5, random_state=42)
    model.fit(X_train, last_y_pred_1)
    coef = 0.9
    trees_1.append(model)
    coeffs_1.append(coef)
    last_y_pred_1 = y_train - gb_predict(X_train, trees_1, coeffs_1)
```

In [29]:

```
y_test_pred_1 = gb_predict(X_test, trees_1, coeffs_1)
print(f'RMSE: {rmse(y_test, y_test_pred_1)}')
```

```
RMSE: 4.043482624592399
```

8. 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?

In [30]:

```
trees_2 = []
coeffs_2 = []

iter_2 = 50
last_y_pred_2 = y_train

for i in range(iter_2):
    model = tree.DecisionTreeRegressor(max_depth=5, random_state=42)
    model.fit(X_train, last_y_pred_2)
    coef = 0.9 / (1.0 + i)
    trees_2.append(model)
    coeffs_2.append(coef)
    last_y_pred_2 = y_train - gb_predict(X_train, trees_2, coeffs_2)

y_test_pred_2 = gb_predict(X_test, trees_2, coeffs_2)
print(f'RMSE: {rmse(y_test, y_test_pred_2)}')
```

```
RMSE: 3.7670141013755623
```

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

In [33]:

```
def build_and_fit_model(X_train, y_train, iterations = 50, depth = 5):
    trees = []
    coeffs = []
    last_y_pred = y_train

    for i in range(iterations):
        model = tree.DecisionTreeRegressor(max_depth=depth, random_state=42)
        model.fit(X_train, last_y_pred)

        coef = 0.9 / (1.0 + i)

        trees.append(model)
        coeffs.append(coef)

        last_y_pred = y_train - gb_predict(X_train, trees, coeffs)

    def predictor(X):
        return gbm_predict(X, trees, coeffs)

    return predictor
```

In [37]:

```
rmse(y_test, train_and_predict(X_train, y_train, X_test))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
 last)
<ipython-input-37-3da0642b384f> in <module>
      1 from sklearn.ensemble import GradientBoostingRegressor
----> 2 rmse(y_test, train_and_predict(X_train, y_train, X_test))

NameError: name 'train_and_predict' is not defined
```

In [38]:

```
from sklearn.ensemble import GradientBoostingRegressor
```

In [39]:

```
iterations_count = np.linspace(1, 100, 20).astype(int)
iter_train_errors = []
iter_valid_errors = []

for it_count in iterations_count:
    predict = build_and_fit_model(X_train, y_train, iterations = it_count)

    iter_train_errors.append(rmse(y_train, predict(X_train)))
    iter_valid_errors.append(rmse(y_test, predict(X_test)))
```

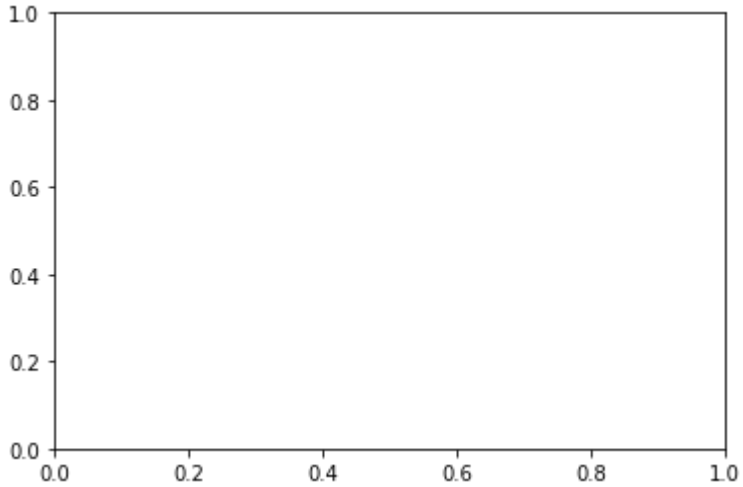
```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-39-9fafd319a4e0> in <module>
      6     predict = build_and_fit_model(X_train, y_train, iterations
= it_count)
      7
----> 8     iter_train_errors.append(rmse(y_train, predict(X_train)))
      9     iter_valid_errors.append(rmse(y_test, predict(X_test)))

<ipython-input-33-543b236f084a> in predictor(X)
     16
     17     def predictor(X):
---> 18         return gbm_predict(X, trees, coeffs)
     19
     20     return predictor

NameError: name 'gbm_predict' is not defined
```

In [40]:

```
plt.plot(iterations_count, iter_train_errors, marker='o', color='blue', label='Train
plt.plot(iterations_count, iter_valid_errors, marker='o', color='red', label='Test')
plt.xlabel('Iter')
plt.ylabel('RMSE')
plt.grid(True)
plt.legend(loc="upper right")
plt.show()
```

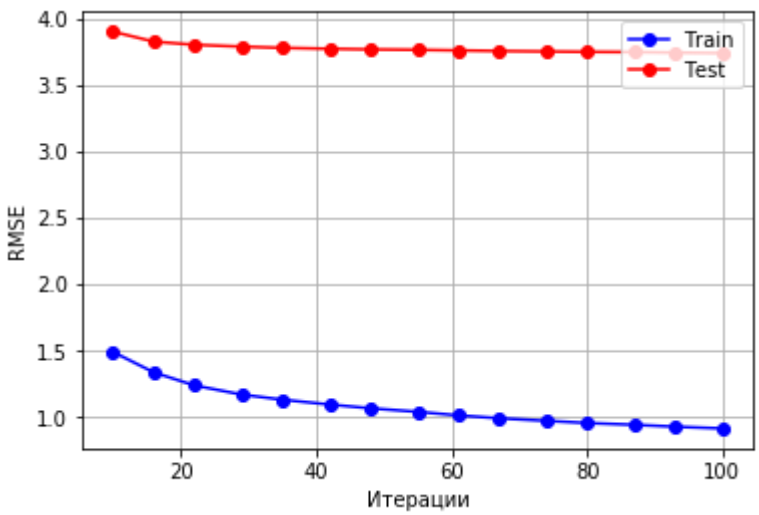
```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
 last)
<ipython-input-40-9ef67c73a50f> in <module>
----> 1 plt.plot(iterations_count, iter_train_errors, marker='o', colo
r='blue', label='Train')
      2 plt.plot(iterations_count, iter_valid_errors, marker='o', colo
r='red', label='Test')
      3 plt.xlabel('Iter')
      4 plt.ylabel('RMSE')
      5 plt.grid(True)

/usr/local/lib/python3.7/site-packages/matplotlib/pyplot.py in plot(sc
alex, scaley, data, *args, **kwargs)
   2787     return gca().plot(
   2788         *args, scalex=scalex, scaley=scaley, **({"data": data}
if data
-> 2789         is not None else {}), **kwargs)
   2790
   2791

/usr/local/lib/python3.7/site-packages/matplotlib/axes/_axes.py in plo
t(self, scalex, scaley, data, *args, **kwargs)
   1663         """
   1664         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D.
_alias_map)
-> 1665         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1666         for line in lines:
   1667             self.add_line(line)

/usr/local/lib/python3.7/site-packages/matplotlib/axes/_base.py in __c
all__(self, *args, **kwargs)
    223                 this += args[0],
    224                 args = args[1:]
--> 225             yield from self._plot_args(this, kwargs)
    226
    227     def get_next_color(self):

/usr/local/lib/python3.7/site-packages/matplotlib/axes/_base.py in _pl
ot_args(self, tup, kwargs)
    389             x, y = index_of(tup[-1])
    390
--> 391         x, y = self._xy_from_xy(x, y)
    392
    393         if self.command == 'plot':

/usr/local/lib/python3.7/site-packages/matplotlib/axes/_base.py in _xy
_from_xy(self, x, y)
    268         if x.shape[0] != y.shape[0]:
    269             raise ValueError("x and y must have same first dim
```

```
ension, but "
--> 270                                 "have shapes {} and {}".format(x.
shape, y.shape))
    271         if x.ndim > 2 or y.ndim > 2:
    272             raise ValueError("x and y can be no greater than 2
-D, but have "

ValueError: x and y must have same first dimension, but have shapes (2
0,) and (0,)
```



In [66]:

```
plt.plot(iterations_count, iter_train_errors, marker='o', color='blue', label='Train
plt.plot(iterations_count, iter_valid_errors, marker='o', color='red', label='Test')
plt.xlabel('Итерации')
plt.ylabel('RMSE')
plt.grid(True)
plt.legend(loc="upper right")
plt.show()
```

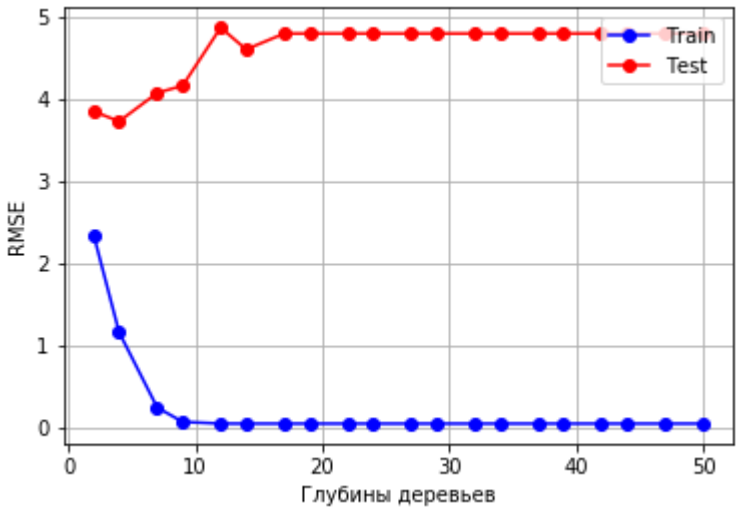


In [134]:

```
depths = np.linspace(2, 50, 20).astype(int)
depth_train_errors = []
depth_valid_errors = []

for depth_num in depths:
    boost = GradientBoostingRegressor(n_estimators=50, max_depth=depth_num, random_s
    boost.fit(X_train, y_train)
    depth_train_errors.append(rmse(y_train, boost.predict(X_train)))
    depth_valid_errors.append(rmse(y_test, boost.predict(X_test)))
```

In [135]:

```
plt.plot(depths, depth_train_errors, marker='o', color='blue', label='Train')
plt.plot(depths, depth_valid_errors, marker='o', color='red', label='Test')
plt.xlabel('Глубины деревьев')
plt.ylabel('RMSE')
plt.grid(True)
plt.legend(loc="upper right")
plt.show()
```



In [ ]:

```
plt.plot(depths, depth_train_errors, marker='o', color='blue', label='Train')
plt.plot(depths, depth_valid_errors, marker='o', color='red', label='Test')
plt.xlabel('Глубины деревьев')
plt.ylabel('RMSE')
plt.grid(True)
plt.legend(loc="upper right")
plt.show()
```

10. 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.

In [22]:

```
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)
rmse(y_test, y_pred_linear)
```

Out[22]:

```
5.6958350306172365
```

Вывод

Ансамбль — это набор предсказателей, которые вместе дают ответ (например, среднее по всем). Причина почему мы используем ансамбли — несколько предсказателей, которые пытаюсь получить одну и ту же переменную дадут более точный результат, нежели одиночный предсказатель. Техники ансамблирования впоследствии классифицируются в Бэггинг и Бустинг.

Бэггинг — простая техника, в которой мы строим независимые модели и комбинируем их, используя некоторую модель усреднения (например, взвешенное среднее, голосование большинства или нормальное среднее).

Бустинг — это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно

Градиентный бустинг — это техника машинного обучения для задач классификации и регрессии, которая строит модель предсказания в форме ансамбля слабых предсказывающих моделей, обычно деревьев решений.