

Отчёт по лабораторной работе №10 "Градиентный бустинг"

In [3]:

```
import re
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_boston

np.seterr(divide='ignore', invalid='ignore', over='ignore')
```

Out[3]:

```
{'divide': 'ignore', 'over': 'ignore', 'under': 'ignore', 'invalid': 'ignore'}
```

1. Загрузите данные с помощью библиотеки sklearn.

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

In [9]:

```
df_data = load_boston()
X = df_data['data']
Y = df_data['target']
X.shape, Y.shape
```

Out[9]:

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

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

In [10]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.75)
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[10]:

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

Далее пункты 4-7 будут реализованы с помощью класса GradientBoostingRegressor

In [11]:

```
class GradientBoostingRegressor:
    models = []

    def __init__(self, etha=0.05, trees_count=50, **tree_params):
        self.ethas = np.array([etha(i) if callable(etha) else etha for i in range(trees_count)])
        self.tree_params = tree_params
        self.trees_count = trees_count
        self.trees = []

    def _negative_gradient(self, predictions, y):
        return y - predictions
```

```

def predict(self, X, trees_limit=None):
    trees = self.trees[:trees_limit] if trees_limit else self.trees
    predictions = np.array([tree.predict(X) for tree in trees]).T
    return np.dot(predictions, self.ethas[:len(trees)])

def fit(self, X, Y):
    for i in range(self.trees_count):
        predictions = self.predict(X)
        gradients = self._negative_gradient(predictions, Y)

        decision_tree = DecisionTreeRegressor(**self.tree_params)
        decision_tree.fit(X, gradients)
        self.trees.append(decision_tree)

    def score(self, X, Y, error_function=mean_squared_error, **predict_params):
        return error_function(Y, self.predict(X, **predict_params))

```

4. Заведите массив для объектов `DecisionTreeRegressor` (они будут использоваться в качестве базовых алгоритмов) и для вещественных чисел (коэффициенты перед базовыми алгоритмами).

Такие массивы заводятся при инициализации класса. Массив для объектов деревьев соответствует `self.trees`, а массив коэффициентов - `self.ethas`. Массив коэффициентов может быть задан функцией или числом.

5. 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 учатся прогнозировать дерево, будут меняться в соответствие с отклонением истинных значений от предсказанных.

Выше перечисленное реализовано в методе `fit`. Некоторые параметры необходимо использовать при инициализации объекта.

In [18]:

```

trees_count = 50
etha = 0.9 # composition element coef
regressor = GradientBoostingRegressor(etha, trees_count=trees_count, max_depth=5, random_state=42)
regressor.fit(X_train, Y_train)

```

6. 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.

При инициализации объекта использовалось переменная `etha` со значением 0.9

7. 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`.

Проверка качества обучения реализована через функцию `score`.

In [19]:

```

print('Prediction params:\nEthas: 0.9\n')
print('Prediction error on train set: ', regressor.score(X_train, Y_train))
print('Prediction error on test set: ', regressor.score(X_test, Y_test))

```

```

Prediction params:
Ethas: 0.9

```

```

Prediction error on train set:  9.915032444905594e-06
Prediction error on test set:  22.417951240321106

```

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 [20]:

```
print('\n\nPrediction params:\nEtha: 0.9 / (1.0 + i)\n')

etha = lambda i: 0.9 / (1.0 + i)
regressor = GradientBoostingRegressor(etha, max_depth=5, random_state=42)
regressor.fit(X_train, Y_train)

print('Prediction error on train set: ', regressor.score(X_train, Y_train))
print('Prediction error on test set: ', regressor.score(X_test, Y_test))
```

```
Prediction params:
Etha: 0.9 / (1.0 + i)

Prediction error on train set:  1.1782691583620633
Prediction error on test set:  19.264391438549474
```

Ошибка на контрольной выборке всё еще сохраняется, но по сравнению с предыдущим типом коэффициентов ее уменьшение на лицо.

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

Исследуем зависимость ошибки градиентного бустинга от числа итераций (количества деревьев):

In [21]:

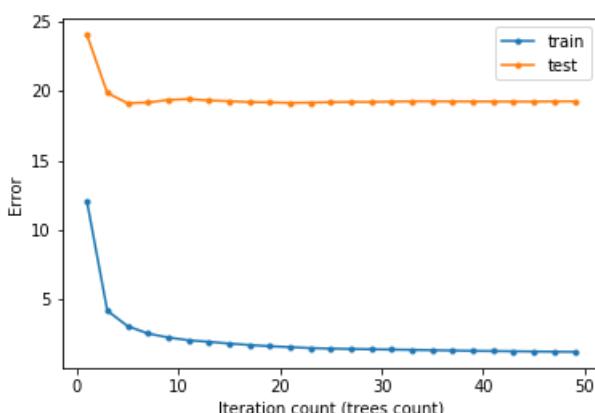
```
trees_count = np.arange(1, 50, 2)
max_count = trees_count.max()

regressor = GradientBoostingRegressor(etha, trees_count=max_count, max_depth=5, random_state=42)
regressor.fit(X_train, Y_train)

y_train = np.array([regressor.score(X_train, Y_train, trees_limit=count) for count in trees_count])
y_test = np.array([regressor.score(X_test, Y_test, trees_limit=count) for count in trees_count])

best_trees_count = trees_count[np.argmin(y_test)]

plt.plot(trees_count, y_train, marker='.', label="train")
plt.plot(trees_count, y_test, marker='.', label="test")
plt.xlabel('Iteration count (trees count)')
plt.ylabel('Error')
plt.legend()
plt.show()
```



На графике представлены кривые ошибок градиентного бустинга от количества деревьев на тестовой и обучающей выборках.

Исследуем зависимость ошибки градиентного бустинга от глубины деревьев:

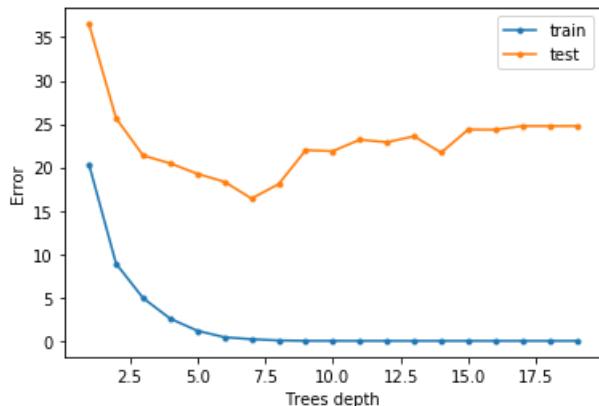
In [23]:

```
trees_depth = np.arange(1, 20)
y_train, y_test = [], []

for depth in trees_depth:
    regressor = GradientBoostingRegressor(etha, max_depth=depth, random_state=42)
    regressor.fit(X_train, Y_train)
    y_train.append(regressor.score(X_train, Y_train))
    y_test.append(regressor.score(X_test, Y_test))

best_trees_depth = trees_depth[np.argmin(np.array(y_test))]

plt.plot(trees_depth, y_train, marker='.', label="train")
plt.plot(trees_depth, y_test, marker='.', label="test")
plt.xlabel('Trees depth')
plt.ylabel('Error')
plt.legend()
plt.show()
```



На графике представлены кривые ошибок градиентного бустинга от глубины деревьев на тестовой и обучающей выборках.

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.

Обучим градиентный бустинг на тренировочных данных.

In [26]:

```
regressor = GradientBoostingRegressor(etha, trees_count=best_trees_count,
                                      max_depth=best_trees_depth, random_state=42)
regressor.fit(X_train, Y_train)
```

Вычислим RMSE данного регрессора:

In [27]:

```
rmse = np.sqrt(regressor.score(X_test, Y_test))
print('\n\nRMSE on Gradient Boosting regression: ', rmse)

regressor = LinearRegression().fit(X_train, Y_train)
predictions = regressor.predict(X_test)
rmse = np.sqrt(mean_squared_error(Y_test, predictions))
print('RMSE on Linear regression: ', rmse)
```

RMSE on Gradient Boosting regression: 4.077017725953908

RMSE on Linear regression: 5.339690800438038

Как видно из результатов, градиентный бустинг имеет преимущество над линейной регрессией.