Домашнее задание 2 по ML

Нужно написать python класс градиентного бустинга и побить другую модель на предоставленном baseline

Критерии оценки

- Ваш ноутбук будет запущен через `run all` - он не должен упасть (допускается падение из-за отсутствия библиотеки, которую можно поставить через pip install)
- Вот этот код (внизу ноутбука) `assert imp_my_little_model > imp_baseline_model` не вызвал ошибок (успешно отработал)
- реализованы следующие гиперпараметры
  - вы реализовали гиперпараметр `learning_rate`
  - вы реализовали гиперпараметр `n_estimators`
  - вы реализовали гиперпараметр `max_depth`
  - вы реализовали гиперпараметр `bagging_fraction`
- Вы реализовали Huber loss function - она записана как отдельная def функция вне класса - и используется в вашем классе для расчета

Для успешной сдачи дз нужно выполнить полностью каждый пункт выше

- оценка 5 будет поставлена, если каждый пункт выполнен без недочетов
- оценка 4 будет поставлена, если будет найден один недочет
- незачет будет послявлен, если недочетов будет два или более
- незачет будет послявлен, если какой либо пункт не выполнен

In [ ]:
```
# pip install shap

import numpy as np
import shap

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.tree import DecisionTreeRegressor
```

```
c:\coding\python\MTS\ML\venv\lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not fou
nd. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user
_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [ ]:
```
np.random.seed(42)
```

In [ ]:
```
data, target = shap.datasets.california()
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state
```

Не меняйте название для предсказаний preds_my_little_model, иначе не получится сдать это ДЗ (сломается код)

Некоторые правила

- Нельзя использовать никакие другие алгоритмы моделей внутри вашего класса, кроме DecisionTreeRegressor.
- Код вашего бустинга должен быть написан в классе, у класса должно быть два ожидаемых метода : `fit` и `predict`.
- Нельзя менять датасет (и модифицировать тоже, например заполнять nan или применять scaler) или baseline модель
- Нельзя поднимать число n_estimators вашей модели выше 100 (чтобы результат был сравним с моделью-конкурентом `GradientBoostingRegressor`)

это место для вашего кода ↓↓↓

In [ ]:
```python
def huber(y_true, y_pred, reduction=None, delta=1):
    res = y_true - y_pred
    abs_res = np.abs(res)
    loss = np.where(abs_res < delta, 0.5 * res**2, delta * (abs_res - 0.5 * delta))
    if reduction == 'mean':
        return np.mean(loss)
    elif reduction == 'sum':
        return np.sum(loss)
    elif reduction == None:
        return loss

    return None

def grad_huber(y_true, y_pred, delta=1):
    res = y_true - y_pred
    grad = np.where(np.abs(res) < delta, -res, -delta * np.sign(res))
    return grad
```

In [ ]:
```python
import numpy as np
from sklearn.tree import DecisionTreeRegressor

class MyGradBoosting:
    def __init__(
        self,
        learning_rate=0.1,
        n_estimators=100,
        max_depth=5,
        random_state=4,
        bagging_fraction=0.75
    ):
        self.learning_rate = learning_rate
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.random_state = random_state
        self.bagging_fraction = bagging_fraction


    def fit(self, X, y):

        n_samples = X.shape[0]
        self.trees = []

        self.y_mean = np.mean(y)
        y_pred = np.ones_like(y) * self.y_mean

        for _ in range(self.n_estimators):
            grad = -grad_huber(y, y_pred)

            sample_indices = np.random.choice(range(n_samples), int(n_samples * self.bagging_
```

```
            X_sample = X.iloc[sample_indices]
            y_sample = grad[sample_indices]

            tree = DecisionTreeRegressor(max_depth=self.max_depth)
            tree.fit(X_sample, y_sample)

            y_pred += self.learning_rate * tree.predict(X)

            self.trees.append(tree)


    def predict(self, X):
        predictions = np.ones_like(X.shape[0]) * self.y_mean
        for tree in self.trees:
            predictions += self.learning_rate * tree.predict(X)
        return predictions
```

In [ ]:
```
my_little_model = MyGradBoosting()
my_little_model.fit(X_train, y_train)

preds_my_little_model = my_little_model.predict(X_test)
```

In [ ]:
```
# самопроверки
assert preds_my_little_model.shape == y_test.shape, 'что-то не так с выходным размером предик
```

это место для вашего кода ↑↑↑

ниже ничего менять не нужно

Это класс судья - он решит, какая модель оказалась лучше, ваша, или GradientBoostingRegressor из sklearn

Если ячейка ниже завершилась ошибкой, нужно поменять код вашей модели и попробовать еще раз, до тех пор, пока не получите сообщение "Ура, получилось!"

In [ ]:
```
baseline_model = GradientBoostingRegressor(random_state=4, verbose=0)
baseline_model = baseline_model.fit(X_train, y_train)
preds_baseline_model = baseline_model.predict(X_test)
print('mape - ваша модель', mean_absolute_percentage_error(y_test, preds_my_little_model))
print('mape - baseline', mean_absolute_percentage_error(y_test, preds_baseline_model))

final_estimator = RandomForestRegressor(random_state=16)
final_estimator = final_estimator.fit(
    np.hstack((preds_baseline_model.reshape(-1, 1), preds_my_little_model.reshape(-1, 1))),
    y_test
)

imp_baseline_model, imp_my_little_model = final_estimator.feature_importances_
result_message = f"baseline важность: {imp_baseline_model:0.3f}; важность вашей модели: {imp_

assert imp_my_little_model > imp_baseline_model,  f'попробуй еще раз: {result_message}'
print('Ура, получилось!',  result_message)
```

```
mape - ваша модель 0.19106502207812515
mape - baseline 0.2152446498010688
Ура, получилось! baseline важность: 0.097; важность вашей модели: 0.903
```

In [ ]: