РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра теории вероятностей и кибербезопасности

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № **

Дисциплина: Компьютерные науки и технологии программирования

Студент: Каримов Тимур Ринатович

Группа: 1132246817

Преподаватель: Бегишев В.О.

МОСКВА 2024 г.

Цель работы:

Изучить алгоритм работы градиентного бустинга.

Выполнение работы

**Задание 1:

Для реализованной модели градиентного бустинга построить графики зависимости ошибки от количества деревьев в ансамбле и от максимальной глубины деревьев. Сделать выводы о зависимости ошибки от этих параметров.

Решение:

Зависимость от количества деревьев:

При малом количестве деревьев (1-10) можно отметить следущее:модель недообучена, высокая ошибка на обеих выборках, с каждым новым деревом ошибка значительно уменьшается **Оптимальная зона это 10-50 деревьев** Такое кол-во имеет следующие преимущества: ошибка достигает минимума, разница между train и test ошибкой умеренная, лучший баланс bias-variance.

При большом количестве деревьев (50+) может наблюдаться возможное переобучение (разница ошибок растет), уменьшение возврата от добавления новых деревьев и увеличение вычислительных затрат

Зависимость от глубины деревьев:

При мелких деревьях (depth=1-2): сильное недообучение, требуется больше деревьев для достижения хорошего качества, стабильная работа, но ограниченная предсказательная сила

Оптимальная глубина это 3-5: хороший баланс сложности и обобщения, быстрое уменьшение ошибки, умеренное переобучение **Глубокие деревья (6+)**: сильное переобучение (большая разница train/test ошибок), быстрое достижение минимума ошибки, чувствительность к выбросам.

Я заметил взаимосвязь параметров: глубина влияет на оптимальное количество деревьев**: Более глубокие деревья → нужно меньше деревьев, мелкие деревья → нужно больше деревьев

График:

![[Pasted image 20251204214300.png]]

```
from sklearn import model_selection
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_diabetes
import matplotlib.pyplot as plt
import numpy as np


# Загрузка данных
X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    X, y, test_size=0.25, random_state=42
)


def gb_predict(X, trees_list, eta):
    """Предсказание градиентного бустинга"""
    predictions = np.zeros(X.shape[0])
    for tree in trees_list:
        predictions += eta * tree.predict(X)
    return predictions


def mean_squared_error(y_real, prediction):
    """Среднеквадратичная ошибка"""
```

```
    return np.mean((y_real - prediction) ** 2)


def gb_fit(n_trees, max_depth, X_train, X_test, y_train, y_test, eta):
    """Обучение градиентного бустинга"""
    trees = []
    train_errors = []
    test_errors = []
    current_prediction_train = np.zeros(len(y_train))
    current_prediction_test = np.zeros(len(y_test))
    for i in range(n_trees):
        tree = DecisionTreeRegressor(max_depth=max_depth, random_state=42)
        if i == 0:
            tree.fit(X_train, y_train)
        else:
            residuals = y_train - current_prediction_train
            tree.fit(X_train, residuals)
        trees.append(tree)
        current_prediction_train += eta * tree.predict(X_train)
        current_prediction_test += eta * tree.predict(X_test)
        train_errors.append(mean_squared_error(y_train, current_prediction_train))
        test_errors.append(mean_squared_error(y_test, current_prediction_test))
    return trees, train_errors, test_errors


# ============================================
# 1. Анализ зависимости от количества деревьев
# ============================================
print("=" * 50)
```

```
print("1. Зависимость от количества деревьев")
print("=" * 50)

max_depth = 3
eta = 0.1
tree_counts = [1, 5, 10, 20, 30, 50, 75, 100]

# Обучаем модель с максимальным количеством деревьев
trees_all, train_errors_all, test_errors_all = gb_fit(
    max(tree_counts), max_depth, X_train, X_test, y_train, y_test, eta
)

# Собираем результаты для каждого количества деревьев
results_trees_train = []
results_trees_test = []

for n in tree_counts:
    train_pred = gb_predict(X_train, trees_all[:n], eta)
    test_pred = gb_predict(X_test, trees_all[:n], eta)
    results_trees_train.append(mean_squared_error(y_train, train_pred))
    results_trees_test.append(mean_squared_error(y_test, test_pred))

# График 1: Ошибки vs количество деревьев
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(tree_counts, results_trees_train, 'b-', linewidth=2, label='Train MSE')
plt.plot(tree_counts, results_trees_test, 'r-', linewidth=2, label='Test MSE')
```

```
plt.xlabel('Количество деревьев')

plt.ylabel('MSE')

plt.title(f'Зависимость MSE от количества деревьев\n(max_depth={max_depth}, eta={eta})')

plt.grid(True, alpha=0.3)

plt.legend()



# График 2: Динамика ошибок во время обучения (для 100 деревьев)

plt.subplot(1, 2, 2)

plt.plot(range(1, len(train_errors_all) + 1), train_errors_all, 'b-', label='Train MSE', alpha=0.7)

plt.plot(range(1, len(test_errors_all) + 1), test_errors_all, 'r-', label='Test MSE', alpha=0.7)

plt.xlabel('Итерация обучения (номер дерева)')

plt.ylabel('MSE')

plt.title('Динамика MSE в процессе обучения\n(100 деревьев)')

plt.grid(True, alpha=0.3)

plt.legend()



plt.tight_layout()

plt.show()



# Выводы по количеству деревьев

print("\nАнализ зависимости от количества деревьев:")

print(f"Глубина деревьев: {max_depth}")

print("Кол-во деревьев | Train MSE  | Test MSE   | Разница")

print("-" * 50)

for i, n in enumerate(tree_counts):
```

```
    diff = results_trees_test[i] - results_trees_train[i]

    print(f"{n:14d} | {results_trees_train[i]:10.2f} |
{results_trees_test[i]:10.2f} | {diff:7.2f}")



# ============================================

# 2. Анализ зависимости от глубины деревьев

# ============================================

print("\n" + "=" * 50)

print("2. Зависимость от глубины деревьев")

print("=" * 50)



n_trees = 50

eta = 0.1

depths = [1, 2, 3, 4, 5, 6, 8, 10]



results_depth_train = []

results_depth_test = []



for depth in depths:

    trees, train_errors, test_errors = gb_fit(

        n_trees, depth, X_train, X_test, y_train, y_test, eta

    )

    results_depth_train.append(train_errors[-1])

    results_depth_test.append(test_errors[-1])

    print(f"Глубина {depth}: Train MSE = {train_errors[-1]:.2f}, Test MSE =
{test_errors[-1]:.2f}")



# График 3: Ошибки vs глубина деревьев
```

```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(depths, results_depth_train, 'b-', linewidth=2, marker='o', label='Train
MSE')

plt.plot(depths, results_depth_test, 'r-', linewidth=2, marker='s', label='Test
MSE')

plt.xlabel('Максимальная глубина деревьев')

plt.ylabel('MSE')

plt.title(f'Зависимость MSE от глубины деревьев\n(n_trees={n_trees}, eta={eta})')

plt.grid(True, alpha=0.3)

plt.legend()



# График 4: Разница ошибок vs глубина деревьев

plt.subplot(1, 2, 2)

diff_errors = np.array(results_depth_test) - np.array(results_depth_train)

plt.bar([str(d) for d in depths], diff_errors, color=['red' if d > 0 else 'green'
for d in diff_errors])

plt.xlabel('Максимальная глубина деревьев')

plt.ylabel('Test MSE - Train MSE')

plt.title('Степень переобучения от глубины')

plt.grid(True, alpha=0.3, axis='y')

plt.axhline(y=0, color='k', linestyle='--', alpha=0.5)



plt.tight_layout()

plt.show()
```

**Задание 2:

Модифицировать реализованный алгоритм градиентного бустинга, чтобы получился стохастический градиентный бустинг. Размер подвыборки принять равным 0.5. Сравнить на одном графике кривые изменения ошибки на тестовой выборке в зависимости от числа итераций.

Решение:

Добавил случайный выбор индексов

```
def gb_fit_stochastic(..., subsample=0.5):
```

Добавил расчет размера подвыборки:

```
n_samples = int(len(y_train) * subsample)
```

Добавил новый параметр `subsample=0.5`

```
indices = np.random.choice(len(y_train), size=n_samples, replace=False)
```

Вместо обучения на всей выборке `tree.fit(X_train, ...)` Теперь обучаем на подвыборке `tree.fit(X_batch, ...)`

Какие преимущества я выделил:

1. **Меньше переобучения** - деревья менее коррелированы
2. **Быстрее обучение** - каждое дерево обучается на меньшем объеме данных
3. **Лучшая обобщающая способность** - особенно на больших данных

**Задание 3:

Оптимизировать процесс обучения градиентного бустинга, чтобы он занимал меньше времени.

Решение:

Для того, чтобы оптимизировать код я добавил раннюю остановку и векторизацию

```
from sklearn import model_selection

from sklearn.tree import DecisionTreeRegressor

from sklearn.datasets import load_diabetes

import matplotlib.pyplot as plt

import numpy as np



# Загрузка данных

X, y = load_diabetes(return_X_y=True)
```

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    X, y, test_size=0.25, random_state=42
)


# 1. Базовая версия градиентного бустинга
def gb_fit_slow(n_trees, max_depth, X_train, X_test, y_train, y_test, eta):
    train_errors = []
    test_errors = []
    current_prediction_train = np.zeros(len(y_train))
    current_prediction_test = np.zeros(len(y_test))
    for i in range(n_trees):
        tree = DecisionTreeRegressor(max_depth=max_depth, random_state=42)
        if i == 0:
            tree.fit(X_train, y_train)
        else:
            residuals = y_train - current_prediction_train
            tree.fit(X_train, residuals)
        pred_train = tree.predict(X_train)
        pred_test = tree.predict(X_test)
        current_prediction_train += eta * pred_train
        current_prediction_test += eta * pred_test
        train_errors.append(np.mean((y_train - current_prediction_train) ** 2))
        test_errors.append(np.mean((y_test - current_prediction_test) ** 2))
    return train_errors, test_errors


def gb_fit_fast(n_trees, max_depth, X_train, X_test, y_train, y_test, eta):
    train_errors = []
```

```
    test_errors = []

    current_prediction_train = np.zeros(len(y_train), dtype=np.float32)

    current_prediction_test = np.zeros(len(y_test), dtype=np.float32)

    for i in range(n_trees):

        tree = DecisionTreeRegressor(max_depth=max_depth, random_state=42)

        if i == 0:

            tree.fit(X_train, y_train)

        else:

            residuals = y_train - current_prediction_train

            tree.fit(X_train, residuals)

        pred_train = tree.predict(X_train).astype(np.float32)

        pred_test = tree.predict(X_test).astype(np.float32)

        np.add(current_prediction_train, eta * pred_train,
out=current_prediction_train)

        np.add(current_prediction_test, eta * pred_test,
out=current_prediction_test)

        train_errors.append(np.mean((y_train - current_prediction_train) ** 2))

        test_errors.append(np.mean((y_test - current_prediction_test) ** 2))

    return train_errors, test_errors



# 3. Оптимизированная версия с ранней остановкой

def gb_fit_fast_early_stop(n_trees, max_depth, X_train, X_test, y_train, y_test,
eta, early_stop=10):

    train_errors = []

    test_errors = []

    current_prediction_train = np.zeros(len(y_train), dtype=np.float32)

    current_prediction_test = np.zeros(len(y_test), dtype=np.float32)

    best_test_error = float('inf')
```

```
    no_improvement = 0

    for i in range(n_trees):

        tree = DecisionTreeRegressor(max_depth=max_depth, random_state=42)

        if i == 0:

            tree.fit(X_train, y_train)

        else:

            residuals = y_train - current_prediction_train

            tree.fit(X_train, residuals)

        pred_train = tree.predict(X_train).astype(np.float32)

        pred_test = tree.predict(X_test).astype(np.float32)

        np.add(current_prediction_train, eta * pred_train,
out=current_prediction_train)

        np.add(current_prediction_test, eta * pred_test,
out=current_prediction_test)

        train_error = np.mean((y_train - current_prediction_train) ** 2)

        test_error = np.mean((y_test - current_prediction_test) ** 2)

        train_errors.append(train_error)

        test_errors.append(test_error)

        # Проверка ранней остановки

        if test_error < best_test_error:

            best_test_error = test_error

            no_improvement = 0

        else:

            no_improvement += 1

        if no_improvement >= early_stop:

            break

    return train_errors, test_errors
```

```
# Параметры для экспериментов

n_trees = 100

max_depth = 3

eta = 0.1



# Запуск всех версий

train_slow, test_slow = gb_fit_slow(n_trees, max_depth, X_train, X_test, y_train,
y_test, eta)

train_fast, test_fast = gb_fit_fast(n_trees, max_depth, X_train, X_test, y_train,
y_test, eta)

train_early, test_early = gb_fit_fast_early_stop(n_trees, max_depth, X_train,
X_test, y_train, y_test, eta, early_stop=10)



# Построение графиков

plt.figure(figsize=(12, 4))



# График 1: Сравнение ошибок на тестовой выборке

plt.subplot(1, 2, 1)

plt.plot(range(1, len(test_slow) + 1), test_slow, 'b-', label='Базовая',
linewidth=2)

plt.plot(range(1, len(test_fast) + 1), test_fast, 'r-', label='Оптимизированная',
linewidth=2)

plt.plot(range(1, len(test_early) + 1), test_early, 'g-', label='С ранней
остановкой', linewidth=2)

plt.xlabel('Количество деревьев')

plt.ylabel('MSE на тестовой выборке')

plt.title('Сравнение версий градиентного бустинга')

plt.grid(True, alpha=0.3)

plt.legend()
```

```
# График 2: Финальные ошибки

plt.subplot(1, 2, 2)

final_errors = [test_slow[-1], test_fast[-1], test_early[-1]]

labels = ['Базовая', 'Оптимизированная', 'Ранняя остановка']

colors = ['blue', 'red', 'green']


bars = plt.bar(labels, final_errors, color=colors, alpha=0.7)

plt.ylabel('Финальная MSE')

plt.title('Финальные ошибки на тестовой выборке')

plt.grid(True, alpha=0.3, axis='y')


# Добавляем значения на столбцы

for bar, error in zip(bars, final_errors):

    height = bar.get_height()

    plt.text(bar.get_x() + bar.get_width()/2., height + 50,

             f'{error:.0f}', ha='center', va='bottom')


plt.tight_layout()

plt.show()


# Минимальные ошибки для анализа

min_slow = min(test_slow)

min_fast = min(test_fast)

min_early = min(test_early)




n_trees_early = len(test_early)
```