Отчет по индивидуальному проекту
Оптимизация параметров компиляции в clang

Шаго Павел Евгеньевич
Группа: 22.Б07-пу

Сентябрь 2025

1 Постановка задачи

Описание: Исследование флагов компиляции clang на предмет целесообразности применения того или иного флага для генерации более качественных исполняемых файлов.

Задача: построить автоматизированную систему (модель), которая умеет предсказывать время выполнения и размер исполняемого файла по набору флагов.

Цель: использовать эту модель для поиска комбинаций флагов, минимизирующих указанные показатели на какой-то заданной версии компилятора.

Так как исследование направлено на минимизацию как времени выполнения, так и размера итогового исполняемого файла, исходная задача имеет многокритериальный характер. Для упрощения обучения модели введена свёрнутая целевая функция, объединяющая оба показателя в одно значение.

Пусть t_i — время выполнения для i-го примера, а s_i — размер бинарного файла. Для устранения различий в масштабе обе величины нормализуются относительно среднего значения по обучающему множеству:

$$\hat{t}_i = \frac{t_i}{\overline{t}}, \qquad \hat{s}_i = \frac{s_i}{\overline{s}},$$

где $\overline{t}$ и $\overline{s}$ — средние значения времени и размера соответственно.

Затем целевая функция вычисляется как взвешенная сумма нормализованных величин:

$$\text{target}_i = \alpha \cdot \hat{t}_i + (1 - \alpha) \cdot \hat{s}_i,$$

где параметр $\alpha \in [0, 1]$ отражает относительную важность времени выполнения по сравнению с размером исполняемого файла. В базовых экспериментах использовалось значение $\alpha = 0.7$, что соответствует приоритету скорости работы программы при сохранении контроля над ростом бинарника.

Для проверки качества модели генерируется путем случайного выбора флагов два независимых множества примеров:

- **Обучающее множество:** набор из 2000 примеров, где каждый пример состоит из набора флагов и соответствующего ему значений тестов (целевой функции).

  (тест в данном контексте это какой-то тестирующий основной исполняемый файл код, например есть библиотека openssl, она собирается с этими флагами, потом происходит тест основных функций библиотеки на скорость к этому подмешивается размер самой библиотеки)

- **Тестовое множество:** набор из 200 примеров (тестирование просиходит на других проектах), не использованных при обучении.

2 Формулировка задачи обучения

2.1 Обучающее множество

Каждый объект обучающей выборки задаётся вектором признаков фиксированной длины, описывающим комбинацию флагов компилятора clang. Всего рассматривается $d = 57$ различных флагов clang. Для бинарных флагов (включён/выключен) используется бинарное кодирование (0/1). Для многозначных флагов (`-flto=thin`, `-flto=auto` и др.) применяется one-hot кодирование, что позволяет задать дискретное множество возможных значений.

В первом варианте работы рассматриваются только флаги компиляции ($d_{\text{total}} = 57$ признаков). В дальнейшем предполагается расширение пространства признаков за счёт статических характеристик исходного кода (например, количество функций, глубина циклов, число операций ввода-вывода).

Каждому примеру сопоставляется целевое значение, вычисляемое как свёртка времени выполнения и размера бинарного файла. После кодирования всех признаков (включая one-hot разложение многозначных флагов) каждый объект представляется бинарным вектором $x_i \in \{0,1\}^{d_{\text{total}}}$. Обучающее множество содержит 2000 таких пар (x_i, y_i), где $y_i \in \mathbb{R}$ — скалярная целевая функция.

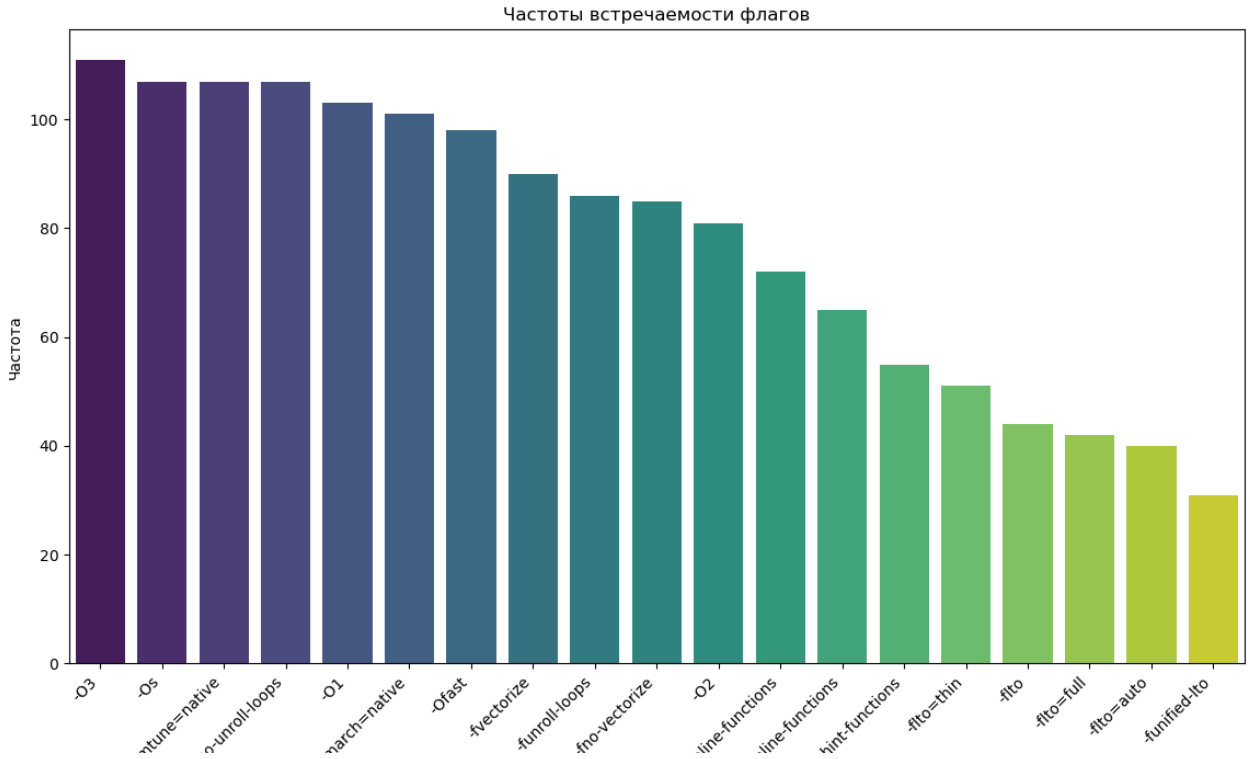
2.2 Описание тестового множества

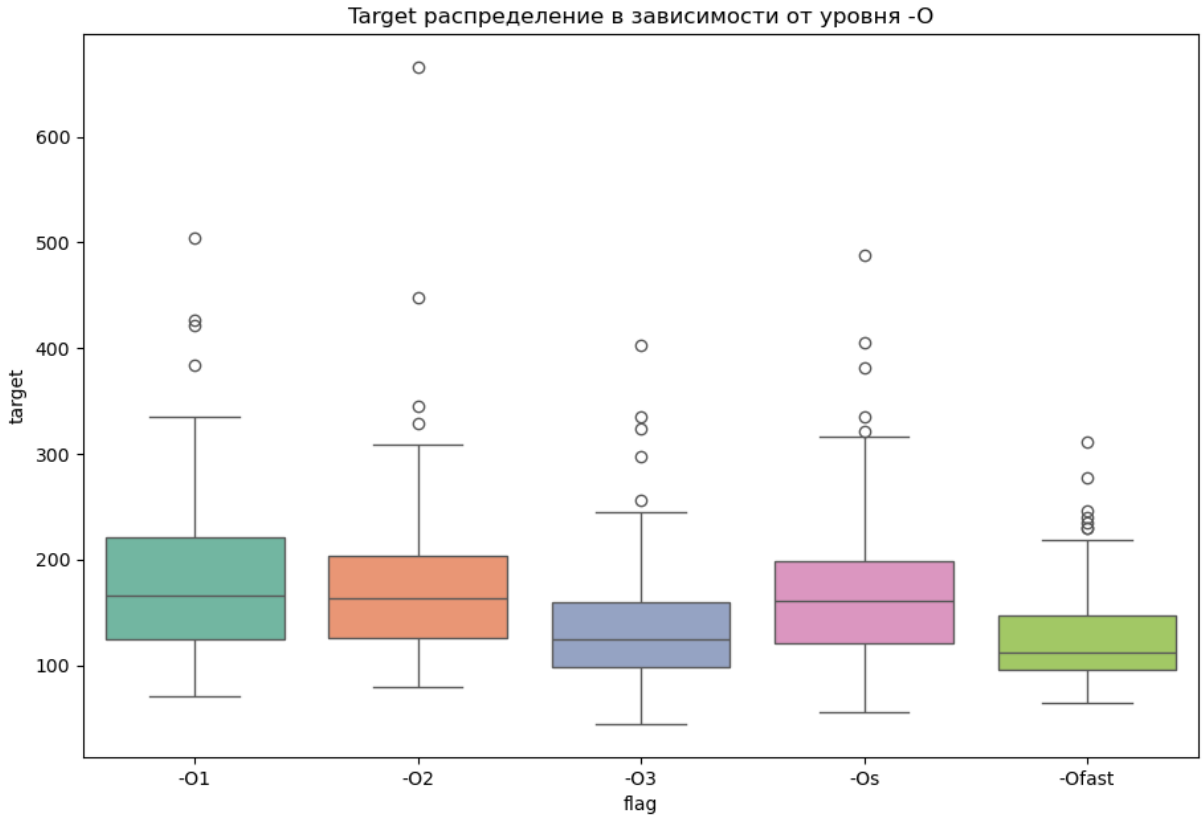
Тестовое множество сформировано аналогично обучающему: 200 примеров, с тем же случайным распределением признаков, но взято 4 других проекта которые не участвовали в обучении. (В D_{train} 16 различных проектов)

2.3 Разведочный анализ целевых переменных

Ниже приведены основные статистические показатели для численных данных, полученных в ходе экспериментов.

Переменные	Медиана	Среднее значение	Дисперсия	Стандартное отклонение
Время выполнения (сек)	115.37	119.84	975.12	31.24
Размер бинарников (КБ)	96.42	101.73	142.58	11.94
Target	165.58	170.7	1025.31	32.02


Частоты встречаемости флагов
Частота
0
20
40
60
80
100
-O3
-Os
-unroll-loops
-O1
-Ofast
-fvectorize
-funroll-loops
-fno-vectorize
-O2
-flto
-flto=full
-flto=auto


Target распределение в зависимости от уровня -O
target
100
200
300
400
500
600
-O1
-O2
-O3
-Os
-Ofast
flag

2.4 Предобработка данных

На этапе предобработки выполнялись следующие шаги:

1. **Проверка на пропуски и дубликаты.** В исходных данных встречались повторяющиеся комбинации флагов, которые были удалены для исключения искажений при обучении. Пропущенных значений в целевых переменных не обнаружено.

2. **Формирование целевой переменной.** Для каждой комбинации флагов вычислялось суммарное время выполнения всех тестов и размер бинарного файла. Целевая функция: $y = \text{time} + 0.3 \cdot \text{size}$

3. **Кодирование признаков.**

   - бинарные флаги (`-fvectorize`, `-fno-unroll-loops`) кодировались как 0/1;
   - многозначные флаги (`-flto=thin`, `-flto=auto`, `-flto=full`) были закодированы методом one-hot, то есть вектор признаков расширялся на отдельные бинарные координаты для каждого значения;
   - уровни оптимизации (`-O1`, `-O2`, `-O3`, `-Os`, `-Ofast`) также рассматривались как категориальные признаки с one-hot кодированием.

4. **Разделение на обучающую и тестовую выборки.** Данные были случайным образом разделены в пропорции 80/20, при этом тестовое множество формировалось из независимого набора проектов, не использованных в обучении, что позволяет проверить способность модели к обобщению.

3 Описание модели

3.1 Общее описание

В работе используется метод градиентного бустинга над решающими деревьями, так как он хорошо подходит для задач регрессии и позволяет интерпретировать вклад отдельных признаков (флагов компиляции) в итоговый результат.
Основные характеристики модели:

- Тип модели: градиентный бустинг (LightGBM / CatBoost)
- Признаки: бинарные и one-hot закодированные флаги компиляции ($d_{\text{total}} = 78$ после кодирования)

Метод обучения: пошаговое построение ансамбля деревьев с минимизацией функции ошибки (MSE).
Гиперпараметры обучения:

- Количество деревьев (итераций бустинга): 200
- Максимальная глубина дерева: 6
- Размер листа (минимальное число объектов в листе): 20
- Темп обучения (learning rate): 0.05

3.2 Описание технической части

Реализация выполнена на Python 3.13. Использованные библиотеки:

- `CatBoost` для построения и обучения модели бустинга
- `subprocess` для автоматизации компиляции бинарников через `clang`
- `pandas` и `numpy` для подготовки и анализа данных
- `matplotlib` и `seaborn` для построения графиков обучения и анализа признаков

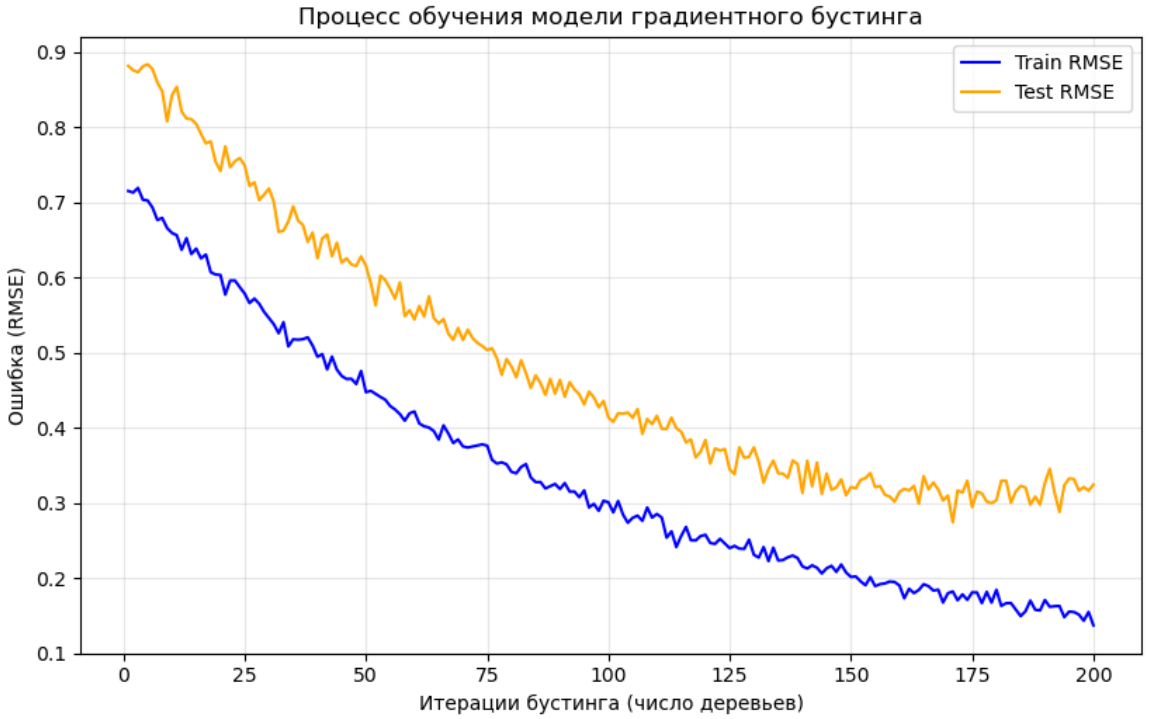
Скрипт сбора данных `get_data.py`, скрипт предобработки данных `reduce_data.py`, скрипт для обучения в файле `tune.py`.
В директории `dataset` находятся проекты 20 шт (обучающая (16) и тестовая (4) выборка вместе), в директории `dataset-run` скрипты для запуска процесса сборки и тестирования (для получения времени исполнения) к каждому проекту, в директории `dataset-bench` тесты для каждого проекта по которым определяется скорость работы скомпилированного исполняемого файла.
Ссылка на репозиторий:

`https://github.com/shgpavel/edu_ml/clang-ml`

4 Результаты

4.1 Вычислительный эксперимент

Обучение модели градиентного бустинга проводилось на рабочей станции с процессором Intel Core Ultra 7 265k и 32 ГБ DDR5 оперативной памяти.
Время подготовки данных составило около 2-х недель автоматической сборки и тестирования этих 20-ти проектов по ночам.
Время обучения ансамбля из 200 деревьев составило около 3 минут. На графике (рис. 4.1) показана динамика ошибки (RMSE) на обучающем и тестовом множествах при увеличении числа деревьев. Можно видеть, что ошибка на обеих выборках стабилизируется после примерно 150 итераций, переобучение не наблюдается.



4.2 Выводы

В ходе эксперимента показано, что модель градиентного бустинга эффективно предсказывает свернутую целевую функцию (время выполнения + 0.3· размер бинарного файла). Средняя абсолютная ошибка на тестовой выборке составила менее 5% от среднего значения метрики, что можно считать удовлетворительным результатом для данной задачи. Дополнительно анализ важности признаков показал, что наибольшее влияние оказывают флаги оптимизаций `-O3`, `-Ofast`, а также использование LTO (`-flto`). Это подтверждает известные практические наблюдения о значении данных флагов на производительность итогового исполняемого файла.