ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ФАКУЛЬТЕТ ТЕХНОЛОГИЙ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Отчёт по лабораторной работе № 2

“Hashcoding против спама“

по дисциплине “Алгоритмы и структуры данных“

Семестр 2

Выполнил студент:
Мавров Артём Николаевич
гр. J3113
ИСУ 466574

Отчет сдан: 24.04.2025

Санкт-Петербург
2025

Содержание

Введение

В данной лабороторной работе предстоит:

- Реализовать алгоритм хеширования “Murmur hash 3“
- Преобразовать строки в вектора с помощью “Feature hashing“
- Реализовать логическую регрессию для определения спама в датасете
- Обучить и протестировать модель.

Весь код реализован на C++. Репозиторий с кодом можно найти по ссылке.

Теоретическая часть

Murmur hash 3

MurmurHash3 — это некриптографическая хеш-функция, разработанная для быстрого хеширования с хорошими распределительными свойствами. Она основана на принципах перемешивания и эффекта лавины.

Определение: Пусть $x \in \{0,1\}^*$ — входная последовательность байтов. MurmurHash3 отображает x в фиксированную длину $h(x) \in \{0,1\}^n$ по формуле:

$$h(x) = \text{Mix}(\text{BlockMix}(x), \text{Seed})$$

Обоснование:

- **Перемешивание:** Каждый блок входных данных умножается на константу и циклически сдвигается:

  $$\text{BlockMix}(x_i) = \text{ROTL}(x_i \cdot c_1, r_1) \cdot c_2$$

  Это обеспечивает диффузию, аналогичную универсальному хешированию.

- **Эффект лавины:** Малое изменение входа приводит к значительному изменению выхода:

  $$\forall x, x' : d_H(x, x') = 1 \Rightarrow d_H(h(x), h(x')) \approx \frac{n}{2}$$

  где d_H — расстояние Хэмминга.

- **Снижение смещённости:** Финальные операции дополнительно перемешивают биты для устранения статистических зависимостей:

  $$h \leftarrow h \oplus (h \gg r) \cdot c_3$$

MurmurHash3 не является криптостойким, но обеспечивает высокую скорость и равномерное распределение для общих задач хеширования.

Feature hashing

Feature hashing — это метод отображения категориальных признаков в фиксированное пространство меньшей размерности с помощью хеш-функции.

Определение: Пусть входной вектор x имеет категориальные признаки, каждый из которых представлен парой (f_j, v_j), где f_j — имя признака, v_j — его значение. Тогда хеш-преобразование:

$$\phi_i(x) = \sum_{j:h(f_j)=i} \xi(f_j) \cdot v_j, \quad i = 1, \ldots, d$$

где:

- $h : \mathcal{F} \to \{1, \ldots, d\}$ — хеш-функция, определяющая индекс в выходном пространстве размерности d;
- $\xi : \mathcal{F} \to \{-1, +1\}$ — случайная знаковая хеш-функция, уменьшающая корреляцию;
- $\phi(x) \in \mathbb{R}^d$ — выходной вектор признаков фиксированной размерности.

Логическая регрессия

Логистическая регрессия моделирует вероятность бинарного события. Пусть:

- $y \in \{0, 1\}$ - целевая переменная
- $\mathbf{x} \in \mathbb{R}^n$ - вектор признаков
- $\mathbf{w} \in \mathbb{R}^n$ - веса модели
- $b \in \mathbb{R}$ - смещение

Линейная комбинация:

$$z = \mathbf{w}^T \mathbf{x} + b$$

Вероятность вычисляется через сигмоидную функцию:

$$P(y = 1|\mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Функция потерь Кросс-энтропия для m примеров:

$$J(\mathbf{w}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \sigma(z^{(i)}) + (1 - y^{(i)}) \log\big(1 - \sigma(z^{(i)})\big) \right]$$

Оптимизация Градиентный спуск для обновления весов:

$$\mathbf{w} := \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

где α - скорость обучения.

Практическая часть

Murmur hashing 3

Функция реализована согласно описанию из теоретической части.

```
uint32_t rotl32(uint32_t x, int r) {
    return (x << r) | (x >> (32 - r));
}

uint32_t fmix32(uint32_t h) {
    h ^= h >> 16;
    h *= 0x85ebca6b;
    h ^= h >> 13;
    h *= 0xc2b2ae35;
    h ^= h >> 16;
    return h;
}

void MurmurHash3 ( const void * key, int len, uint32_t seed, void *
   out ) {
    uint32_t h1 = seed;

    const uint32_t c1 = 0xcc9e2d51;
    const uint32_t c2 = 0x1b873593;

    const uint8_t* data = (const uint8_t*)key;
    const int nblocks = len / 4;

    const uint32_t* blocks = (const uint32_t*)(data);
    for (int i = 0; i < nblocks; i++) {
        uint32_t k1 = blocks[i];
        k1 *= c1;
        k1 = rotl32(k1, 15);
        k1 *= c2;
        h1 ^= k1;
        h1 = rotl32(h1, 13);
        h1 = h1 * 5 + 0xe6546b64;
    }

    const uint8_t* tail = (const uint8_t*)(data + nblocks * 4);
    uint32_t k1 = 0;
    switch (len & 3) {
        case 3: k1 ^= tail[2] << 16; break;
        case 2: k1 ^= tail[1] << 8; break;
        case 1: k1 ^= tail[0];
                k1 *= c1; k1 = rotl32(k1, 15); k1 *= c2; h1 ^= k1;
    }

    h1 ^= len;
    h1 = fmix32(h1);
    std::memcpy(out, &h1, sizeof(h1));
}
```

Feature hashing

В ходе тестирования было принято убрать L-2 нормаолизацию, так как она негативно сказывалась на метреках Recall и Accuracy. Причиной такого нетипичного поведения с большой долей вероятности является не корректность датасета.

Также для улучшения метрик была попытка добавить n-граммы, но это также оказало негативный эффект.

```
vector<double> text_to_features(const std::string& text) {
    vector<double> feats(HASH_DIM, 0);

    string low_text = text;
    transform(low_text.begin(), low_text.end(), low_text.begin(), ::
       tolower);

    istringstream iss(low_text);
    string word;
    while (iss >> word) {
        uint32_t hash = 0;
        MurmurHash3(&word, word.size(), 42, &hash);
        feats[hash % HASH_DIM] += 1;
    }

    return feats;
}
```

Логическая регрессия

Для логической регрессии была выбрана функция $\sigma(z) = \frac{1}{1+e^{-z}}$. Обуяение осуществляется функцией train, evalute используется для подсчёта метрик.

```
class LogisticRegression {
public:
    size_t dim = HASH_DIM;
    double lr = 0.05;
    double reg_lambda = 0.05;
    double class_weight_0 = 1.0;  // (ham)
    double class_weight_1 = 1.0;  // (spam)
    LogisticRegression()
    : weights(dim, 0.0),
      learning_rate(lr),
      lambda(reg_lambda),
      epochs(2000){}

    void train(const std::vector<Sample>& trainData, const std::
       vector<Sample>& validData) {
        for (int epoch = 0; epoch < epochs; ++epoch) {
            vector<double> gradients(dim, 0.0);
            double loss = 0.0;

            for (const auto& sample : trainData) {
                double pred = sigmoid(dot_product(weights, sample.
                   features));
                double error = pred - sample.label;
```

```
                if (pred == 0) {
                    pred += 1e-9;
                }else if (pred == 1) {
                    pred -= 1e-9;
                }
                loss += -sample.label * log(pred) - (1 - sample.label
                   ) * log(1 - pred);

                for (size_t i = 0; i < dim; ++i) {
                    double class_weight = (sample.label == 1) ?
                       class_weight_1 : class_weight_0;
                    gradients[i] += class_weight * error * sample.
                       features[i];
                }
            }

            loss /= trainData.size();
            for (size_t i = 0; i < dim; ++i) {
                weights[i] -= learning_rate * (gradients[i] /
                   trainData.size() + lambda * weights[i]);
            }

            if (epoch % 50 == 0) {
                learning_rate *= 1.0;
            }
        }
    }

    std::vector<double> evaluate(const std::vector<Sample>& data)
       const {
        double TP = 0.0, TN = 0.0, FP = 0.0, FN = 0.0;

        for (const auto& sample : data) {
            int prediction = predict(sample.features);

            if (sample.label == 1 && prediction == 1) {
                TP += 1.0;
            } else if (sample.label == 0 && prediction == 0) {
                TN += 1.0;
            } else if (sample.label == 0 && prediction == 1) {
                FP += 1.0;
            } else if (sample.label == 1 && prediction == 0) {
                FN += 1.0;
            }
        }

        return {TP, TN, FP, FN};
    }

    int predict(const FEATURE& feats) const {
        return (sigmoid(dot_product(weights, feats)) >= 0.5) ? 1 : 0;
    }

```

```
private:
    std::vector<double> weights;
    double learning_rate;
    double lambda;
    int epochs;

    static double dot_product(const FEATURE& w, const FEATURE& x) {
        double res = 0.0;
        for (size_t i = 0; i < w.size(); ++i) {
            res += w[i] * x[i];
        }
        return res;
    }

    static double sigmoid(double z) {
        return 1.0 / (1.0 + exp(-z));
    }
};
```

Тестирование

В ходе тестирования было выявлено, что L-2 нормализация негативно сказывается на метриках, n-граммы так-же не оказывают положительного влияния. Эпохи так-же не дают положительный результат. Также был понижен learning rate для градиентного спуска.

Учитывая эти доработки метрика $Accuracy \approx 0.65$, $Recall \approx 0.97$.

Заключение

По итогам лабороторной работы были получены и закреплены теоретические знания хэш-функций, устройства логической регрессии. Реализована хэш-функция Murmur hashing 3, преобразованы строки в вектора фия при помощи Feature hashing, реализован класс логической регрессии.

Все функции протестированы и граммотно работают. Из недочётов можно выявить невысокие показатели матрики Accuracy.

Для улучшения метрик, можно:

- Продолжать тестировать модель с разным набором констант, для подбора более лучших.
- Попробовать использовать такие методы как: метод максимального правдоподобия, метод Ньютона-Рафсона, кросс-валидация.