Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: «Компьютерные науки и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Дискретный анализ»

Курсовая работа
Тема: «Автоматическая классификация документов»

Студенты:	Ирисов Т. А.
Группа:	М8О-309Б-22
Преподаватель:	Макаров Н.К.
Дата:	
Оценка:	

Москва 2024

Оглавление

Цель работы

Изучить наивный алгоритм классификации Байеса.

Постановка задачи

Необходимо реализовать наивный байесовский классификатор, который будет обучен на первой части входных данных и классифицировать им вторую часть.
Входные данные подаются в виде:
N M
class_type
training_data_1
.
.
class_type
training_data_N
test_data_1
.
.
test_data_M

где N – количество тренировочных наборов строк , M – количество тестовых строк.

Общие сведения о программе

Данная программа реализована с использованием байесовского наивного классификатора и лапласовского сглаживания. Для хранения количества посчитанных слов используется hash таблица, что обеспечивает доступ к данным за O(1). Слова (стоп слова), которые не несут значимой информации, игнорируются.

Общий алгоритм решения

Для обучающей выборки, мы проходим по словам и если этого слова нет, мы добавляем его в нашу таблицу. Если же слово уже есть, то увеличиваем его счетчик на единицу, то есть просто считаем сколько слов для каждого класса 1 и 0 есть в нашей обучающей выборке.
При тестировании, мы тоже проходимся по словам и ищем эти слова уже в нашей сформированной таблице. На этом этапе, у нас имеется информация о том: сколько раз это слово встречается в каждом классе, количество слов в каждом классе, общее количество слов в классах.
И считаем вероятность принадлежности слова к каждому классу по формуле:

$$P(\text{класс} \mid \text{слово}) = \frac{P(\text{слово} \mid \text{класс}) \cdot P(\text{класс})}{P(\text{слово})}$$

Где
P(слово | класс) - это количество текущих слов в классе, деленное на количество слов в самом классе, то есть вероятность появление слова, при условии его принадлежности к этому классу.
P(класс) – количество слов текущего класса на количество слов во всех классах, то есть вероятность попадания в текущий класс.
P(слово) – вероятность попадания слова в любой класс.

Вероятность строки высчитывается как произведение вероятностей для слов:

$$P(\text{класс}|\text{текст}) = P(\text{класс}) \cdot \prod_{i=1}^{N} P(\text{слово}_i|\text{класс})$$

Но тут возникают некие трудности.
Во первых, слово может не встречаться в классе, то есть P(слово | класс) может равняться нулю, что обнулит вероятность принадлежности ВСЕЙ строки данному классу, что интуитивно уже неправильно.

Во вторых, если слово не встречается во всех классах, то есть P(слово) равняется нулю, у нас возникает деление на 0.

В таком случае, мы используем лапласовское сглаживание:

$$P(\text{класс} \mid \text{слово}) = \frac{(P(\text{слово} \mid \text{класс}) + 1) \cdot P(\text{класс})}{P(\text{слово}) + V}$$

Где V - количество слов в классе.

Так как мы используем произведение вероятностей, то наш ответ может стать слишком маленьким и уже попасть в область потери точности, для этого существует логарифмическая формула Байеса:

$$\log P(C \mid w_1, w_2, \ldots, w_n) = \log P(C) + \sum_{i=1}^{n} \log P(w_i \mid C)$$

И тогда наша формула примет вид:

$$\log P(\text{класс} \mid \text{слово}) = \log P(\text{класс}) + \sum_{i=1}^{n} \log \frac{P(\text{слово} \mid \text{класс}) + 1}{P(\text{слово}) + V}$$

Реализация

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <cmath>
std::unordered_map<std::string, bool> stop_words_map = {
    {"the", true}, {"a", true}, {"an", true}, {"I", true},
    {"you", true}, {"he", true}, {"she", true}, {"it", true},
    {"and", true}, {"or", true}, {"but", true}, {"in", true},
    {"on", true}, {"at", true}, {"by", true},
    {"is", true}, {"are", true}, {"was", true}, {"were", true}
};
int is_stop_word(std::string& word) {
    if (stop_words_map.find(word) != stop_words_map.end()) {
        return 1;
    }
    return 0;
}
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int training_quantity, test_quantity;
    std::cin >> training_quantity >> test_quantity;
    std::unordered_map<std::string,int> class0;
    std::unordered_map<std::string,int> class1;
    size_t class_0_word_count = 0;
    size_t class_1_word_count = 0;
    for (int i = 0; i < training_quantity; i++) {
        size_t* word_count;
        std::string word;
        int class_type;
        std::unordered_map<std::string,int>* current_class;
        std::cin >> class_type;
        if (class_type == 0) {
            current_class = &class0;
            word_count  = &class_0_word_count;
        }
        else {
            current_class = &class1;
            word_count  = &class_1_word_count;
        }
        std::cin.ignore();
        char splitter = ' ';
        while (splitter != '\n') {
            std::cin >> word;
            splitter = std::cin.get();
            if (splitter == '\n') {
                if (word[word.length() - 1] == '.') {
                    word.pop_back();
                }
            }
            if (is_stop_word(word)) {
```

```
                continue;
            }
            (*word_count)++;
            std::unordered_map<std::string,int>::iterator it =
(*current_class).find(word);
            if (it == (*current_class).end()) {
                (*current_class)[word] = 1;
            }
            else {
                (*current_class)[word]++;
            }
        }
    }
    size_t common_count_all = class_0_word_count + class_1_word_count;
    double p0 = class_0_word_count * 1.0 / common_count_all;
    double p1 = class_1_word_count * 1.0 / common_count_all;
    std::string word;
    for (int i = 0; i < test_quantity; i++){
        char splitter = ' ';
        double p_in_0 = std::log(p0);
        double p_in_1 = std::log(p1);
        while (splitter  != '\n' && splitter != EOF) {
            std::cin >> word;
            splitter = std::cin.get();
            if (splitter == '\n') {
                if (word[word.length() - 1] == '.') {
                    word.pop_back();
                }
            }
            if (is_stop_word(word)) {
                continue;
            }
            auto w_in_0 = class0.find(word);
            auto w_in_1 = class1.find(word);
            size_t word_count_0 = 0;
            size_t word_count_1 = 0;
            int common_count =  0;
            if (w_in_0 != class0.end()) {
                    word_count_0 = w_in_0->second;
                    common_count += word_count_0;
            }
            if (w_in_1 != class1.end()) {
                word_count_1 = w_in_1->second;
                common_count += word_count_1;
            }
            double p_common = double(common_count) / common_count_all;
            p_in_0 += std::log((double)(word_count_0 + 1)/(p_common +
class0.size()));
            p_in_1 += std::log((double)(word_count_1 + 1)/(p_common +
class1.size()));
        }
        if (p_in_0 > p_in_1) {
            std::cout << 0 << '\n';
        }
        else {
            std::cout << 1 << '\n';
        }
        std::cout.flush();
    }
    return 0;
}
```

Пример работы

Input	output
В первом тесте Вы получите на ввод: 4 2 0 Cats and dogs are friends. 0 Mouse hiding from cat. 1 I play football with my friends. 1 Our football team is called the March cats. Mouse eats cheese next to cats I have friends on another football team.	0 1

Дневник отладки

Мы протестировали на сложной выборке наш алгоритм, то есть обучающая выборка не покрывала все случаи, которые могли встретиться в тестирующей, и была меньше на 22 строки. То есть мы протестировали наш код в сложных условиях.

Обучающей выборки 34 строки.

Тренировочной выборки 56 строк.

Для теста производительности и качества, нам пришлось немного модернизировать наш код, и теперь он принимает так же 2 числа(количество обучающих и тренировочных строк), но уже остальные строки однотипны:

N M
class_type
training_data_1
.
.
class_type
training_data_N
class_type
test_data_1
.
.
class_type
test_data_M

То есть, мы знаем ответ в тренировочной выборке, для того чтобы сравнить его с нашим предсказанным.

Посчитав время работы программы и долю правильных ответов, то есть accuracy, мы получили такой результат:

```
Study time: 16 miliseconds
Classification time: 26 miliseconds
Accuracy 0.839286
```

Наша программа показала 83% точности, что в наших придуманных условиях является хорошим результатом.

Тестовые данные будут приложены в репозитории курсовой работы.

Вывод

В ходе выполнения курсовой работы была реализована система для автоматической классификации текстовых документов с использованием наивного байесовского классификатора. Основной задачей было обучение модели на основе тренировочных данных и последующая классификация новых тестовых документов. Реализация программы показала хорошие результаты на тестовых данных, где была достигнута точность классификации на уровне 83%. Это подтверждает эффективность выбранного подхода для данной задачи, несмотря на то, что обучающие данные не всегда покрывают все возможные случаи, которые могут встретиться в тестовых данных. Работа позволяет сделать вывод о том, что наивный байесовский классификатор является мощным инструментом для классификации текстов.