Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»
«Автоматическая классификация документов»

Студент: И. П. Моисеенков
Преподаватель: С. А. Сорокин
Группа: М8О-208Б-19

Дата: 29.06.2021
Оценка: хорошо
Подпись:

Москва, 2021

1 Задание

Требуется реализовать систему, которая на основе базы вопросов и тегов к ним будет предлагать варианты тегов, которые подходят к новым вопросам.
Формат запуска программы в режиме обучения:

./prog learn –input <input file> –output <stats file>

- –input - входной файл с вопросами
- –output - выходной файл с расчитанной статистикой

Формат запуска программы в режиме классификации:

./prog classify –stats <stats file> –input <input file> –output <output file>

- –stats - файл со статистикой, полученной на предыдущем этапе
- –input - входной файл с вопросами
- –output - выходной файл с тегами к вопросам

Формат входных файлов при обучении:

<Количество строк в вопросе [n]>
<Тег 1>,<Тег 2>,...,<Тег m>
<Заголовок вопроса>
<Текст вопроса [n строк]>

Формат входных файлов при запросах:

<Количество строк в вопросе [n]>
<Заголовок вопроса>
<Текст вопроса [n строк]>

Формат выходного файла: для каждого запроса в отдельной строке выводится предполагаемый набор тегов, через запятую.

2 Описание

Для решения этой задачи я использовал наивный байесовский классификатор. Это простой вероятностный классификатор, который работает на основе формулы Байеса:

$$P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$$

Классификатор является «наивным», т.к. он работает со строгим предположением о независимости классов. Таким образом, для вычисления вероятности принадлежности текста к определенному классу используется формула:

$$P(class|word_1, \ldots, word_n) = \frac{P(word_1|class)*\cdots*P(word_n|class)*P(class)}{P(word_1)*\cdots*P(word_n)}$$

Необходимо рассчитать вероятность принадлежности к каждому классу, применить функцию Softmax и выбрать классы с наибольшими вероятностями (в моей реализации это все классы с вероятностями больше $1/|classes|$. Именно к этим классам и будет относиться рассматриваемый вопрос.

Можно заметить, что знаменатель остается неизменным при рассмотрении одного вопроса, поэтому он не будет влиять на итоговое распределение вероятностей. Это значит, что его можно не вычислять.

Во избежание переполнения типа double в классификаторе вычисляется не произведение вероятностей, а сумма их логарифмов.

Отдельно стоит упомянуть следующую проблему. Если в тексте встретилось слово, которого не было в обучающей выборке, то по формуле получается нулевая вероятность принадлежности любому классу. Поэтому при вычислении вероятностей применяется сглаживание Лапласа:

$$P(word|class) = \frac{P(word)+\alpha}{P(class)+\alpha*|classes|}$$

В моей реализации по умолчанию $\alpha = 1$.

При запуске программы в режиме обучения классификатор собирает статистику - сколько всего слов в выборке, сколько слов встретилось при определенном классе, сколько раз встретилось каждое слово при определенном классе и сколько раз встрелилось каждое слово в общем. Эта же информация и сохраняется в файл со статистикой.

3 Исходный код

Класс NaiveBayesClassifier

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>

using TText = std::vector<std::string>;

struct data {
    TText tag;
    TText doc;
};

class NaiveBayesClassifier {
public:
    void fit(std::vector<data> &dataset) {
        for (const auto &data : dataset) {
            TText tags = data.tag;
            TText doc = data.doc;
            for (auto &tag: tags) {
                for (auto &word : doc) {
                    add(tag, word);
                }
            }
            dataset_size += doc.size();
        }
    }

    std::unordered_map<std::string, double> predict_proba(TText &doc) {
        std::unordered_map<std::string, double> probabilities;
        for (auto &data : tag_summary) {
            std::string cur_tag = data.first;
            probabilities[cur_tag] = probability(cur_tag, doc);
        }

        return probabilities;
    }

    void save_stats(std::ofstream &out) {
        out << dataset_size << "\n";
        for (auto &tag_data : tag_summary) {
            std::string tag = tag_data.first;
            auto data = tag_data.second;
            out << tag << " " << data.size() << " ";
            for (auto &word_count : data) {
                out << word_count.first << " " << word_count.second << " ";
            }
```

```
46              out << "\n";
47          }
48      }
49
50      void load_stats(std::ifstream &in) {
51          in >> dataset_size;
52          std::string tag;
53          while (in >> tag) {
54              int amount_of_words;
55              in >> amount_of_words;
56              for (int i = 0; i < amount_of_words; ++i) {
57                  std::string word;
58                  int count;
59                  in >> word >> count;
60                  add(tag, word, count);
61              }
62          }
63      }
64
65      int get_number_of_tags() {
66          return tag_sizes.size();
67      }
68
69  private:
70      std::unordered_map<std::string, int> tag_sizes; // amounts of words with each tag
71      std::unordered_map<std::string, std::unordered_map<std::string, int>> tag_summary;
            // amounts of each word with each tag
72      std::unordered_map<std::string, int> words_summary; // amounts of each word
73      int dataset_size = 0; // total amount of words in dataset
74
75      double probability(std::string &tag, TText &doc, double alpha = 1) { // P(tag | doc
            )
76          // the Bayes rule
77
78          double prob = 0;
79          for (auto word : doc) {
80              prob += log(probability(word, tag, alpha));
81          }
82
83          prob += log(probability(tag, alpha));
84
85          return prob;
86      }
87
88      double probability(std::string &tag, double alpha = 1) { // P(tag)
89          // applying Laplace smoothing
90          return (1. * tag_sizes[tag] + alpha) / (dataset_size + alpha * dataset_size);
91      }
92
```

```
93      double probability(std::string &word, std::string &tag, double alpha = 1) { // P(
            word | tag)
94          // applying Laplace smoothing
95          return (1. * tag_summary[tag][word] + alpha) / (words_summary[word] + alpha *
                tag_summary.size());
96      }
97
98      void add(std::string &tag, std::string &word) {
99          if (tag_summary[tag][word]) {
100             ++tag_summary[tag][word];
101         } else {
102             tag_summary[tag][word] = 1;
103         }
104
105         if (words_summary[word]) {
106             ++words_summary[word];
107         } else {
108             words_summary[word] = 1;
109         }
110
111         if (tag_sizes[tag]) {
112             tag_sizes[tag] = 1;
113         } else {
114             ++tag_sizes[tag];
115         }
116     }
117
118     void add(std::string &tag, std::string &word, int count) {
119         tag_summary[tag][word] = count;
120
121         if (words_summary[word]) {
122             words_summary[word] += count;
123         } else {
124             words_summary[word] = count;
125         }
126
127         if (tag_sizes[tag]) {
128             tag_sizes[tag] += count;
129         } else {
130             tag_sizes[tag] = count;
131         }
132     }
133 };
```

main.cpp

```
1  #include <cstring>
2  #include <fstream>
```

```cpp
#include <unordered_set>
#include <cmath>

#include "NaiveBayesClassifier.h"

TText convert(std::string &text); // convert text to vector of words
void print_error();
std::unordered_map<std::string, double> softmax(std::unordered_map<std::string, double
    >& preds);

int main(int argc, char *argv[]) {
    std::string input_file_name;
    std::string stats_file_name;
    std::string output_file_name;

    if (argc < 6) {
        print_error();
        return 1;
    }

    if (!strcmp(argv[1], "learn")) {
        if (argc != 6) {
            print_error();
            return 1;
        }

        for (int i = 2; i < argc; ++i) {
            if (!strcmp(argv[i], "--input") && input_file_name.empty()) {
                input_file_name = argv[++i];
            } else if (!strcmp(argv[i], "--output") && stats_file_name.empty()) {
                stats_file_name = argv[++i];
            } else {
                print_error();
                return 1;
            }
        }

        if (input_file_name.empty() || stats_file_name.empty()) {
            print_error();
            return 1;
        }

        std::ifstream input_file(input_file_name);
        std::ofstream stats_file(stats_file_name);
        std::vector<data> dataset;

        int lines;
        while (input_file >> lines) {
            std::string tags;
```

```
51              std::string text;
52
53              input_file.ignore();
54              getline(input_file, tags);
55              for (int i = 0; i < lines + 1; ++i) {
56                  std::string cur_line;
57                  getline(input_file, cur_line);
58                  text += " " + cur_line;
59              }
60              dataset.push_back({convert(tags), convert(text)});
61          }
62
63          NaiveBayesClassifier NB;
64          NB.fit(dataset);
65          NB.save_stats(stats_file);
66      } else if (!strcmp(argv[1], "classify")) {
67          if (argc != 8) {
68              print_error();
69              return 1;
70          }
71
72          for (int i = 2; i < argc; ++i) {
73              if (!strcmp(argv[i], "--input") && input_file_name.empty()) {
74                  input_file_name = argv[++i];
75              } else if (!strcmp(argv[i], "--output") && output_file_name.empty()) {
76                  output_file_name = argv[++i];
77              } else if (!strcmp(argv[i], "--stats") && stats_file_name.empty()) {
78                  stats_file_name = argv[++i];
79              } else {
80                  print_error();
81                  return 1;
82              }
83          }
84
85          if (input_file_name.empty() || stats_file_name.empty() || output_file_name.
                empty()) {
86              print_error();
87              return 1;
88          }
89
90          std::ifstream input_file(input_file_name);
91          std::ifstream stats_file(stats_file_name);
92          std::ofstream output_file(output_file_name);
93
94          NaiveBayesClassifier NB;
95          NB.load_stats(stats_file);
96
97          int lines;
98          while (input_file >> lines) {
```

```
            std::string text;

            input_file.ignore();
            for (int i = 0; i < lines + 1; ++i) {
                std::string cur_line;
                getline(input_file, cur_line);
                text += " " + cur_line;
            }
            auto doc = convert(text);
            auto probas = NB.predict_proba(doc);
            auto preds = softmax(probas);

            bool need_comma = false;
            double threshold = 1. / NB.get_number_of_tags();
            for (const auto& pred : preds) {
                if (pred.second > threshold) {
                    if (need_comma) {
                        output_file << ", ";
                    }
                    output_file << pred.first;
                    need_comma = true;
                }
            }
            output_file << "\n";
        }

    } else {
        print_error();
        return 1;
    }
}

TText convert(std::string &text) { // convert text to vector of words
    std::vector<std::string> words;
    int i = 0;
    std::string cur_word;
    while (i < text.size()) {
        if (isalpha(text[i])) {
            cur_word += tolower(text[i]);
        } else if (cur_word.length()) {
            words.push_back(cur_word);
            cur_word = "";
        }
        if (i + 1 == text.size() && cur_word.length()) {
            words.push_back(cur_word);
        }
        ++i;
    }
    return words;
```

```
148 }
149
150 void print_error() {
151     std::cerr << "Incorrect syntax!" << std::endl <<
152               "./classifier learn --input <input file> --output <output file>" << std::
                      endl <<
153               "./classifier classify --stats <stats file> --input <input file> --output
                      <output file>" << std::endl;
154 }
155
156 std::unordered_map<std::string, double> softmax(std::unordered_map<std::string, double
        >& probs) {
157     double max = -1e10;
158     for (const auto& prob : probs) {
159         if (prob.second > max) {
160             max = prob.second;
161         }
162     }
163
164     double sum = 0;
165     for (auto& prob : probs) {
166         sum += exp(prob.second - max);
167     }
168
169     double constant = max + log(sum);
170     std::unordered_map<std::string, double> res;
171     for (auto prob : probs) {
172         res[prob.first] = exp(prob.second - constant);
173     }
174     return res;
175 }
```

4 Демонстрация работы

Продемонстирую работоспособность классификатора на простом датасете. Будем классифицировать вопросы на три группы - про кошек, собак или хомяков.

Обучающие данные

1
Dogs
Buying a dog
Where can I buy a dog?
1
Dogs
Puppies
How many puppies are there in his house?
1
Dogs
Cute animals
What are the most beautiful breeds of dogs?
2
Dogs
Lost my dog
I've lost my dog.
How can I find it now?
1
Cats
A name for a cat
What are the best names for cats?
1
Cats
Mother cat with her kittens
My cat has many kittens. How should I take care of them?
1
Cats
Different cats
How many different kinds of cats exist?
1
Cats
Black cat
Is it true that a black cat causes problems?
1

Hamsters
Hamster's age
How many years do hamsters live?
1
Hamsters
Very small animals
What is the smallest hamster in the world?
1
Hamsters
Feeding hamsters
I have two hamsters. How should I feed them?
1
Hamsters
Hamsters living together
Can two small hamsters live together in one cage?
1
Cats, dogs
Friendship between cats and dogs
Can cats and dogs live together?
1
Dogs, hamsters
Animals
Can my puppy make friends with a new little hamster?

Тестовые данные

1
Feeding
How to feed my little friend who lives in a cage?
1
Puppies and kittens
Who is better - a puppy or a kitten?
1
Many animals!
Is it ok, that a little hamster, two black cats and a puppy live together with me?

Результат

hamsters
dogs, cats
dogs, cats

5 Выводы

Выполнив данное задание, я познакомился с наивным байесовским классификатором. Это очень простая, но в то же время эффективная модель для классификации текстов. Байесовский классификатор используется, например, для фильтра спама и определения тегов к текстам.

Я реализовал свой байесовский классификатор с нуля, при этом я познакомился с различными техниками для улучшения его работоспособности (например, сглаживание Лапласа).

При написании кода я использовал знания, полученные на курсах дискретного анализа и объектно-ориентированного программирования.

Для тестирования классификатора я создал несколько датасетов. Первый - с вопросами о животных, второй - с электронными письмами (классификация на спам и нормальные письма). На втором датасете я измерил точность работы классификатора. Полученный результат: $accuracy \approx 75\%$

Список литературы

[1] *Наивный классификатор Байеса*
URL: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c.

[2] *Наивный классификатор Байеса*
URL: https://vk.cc/c3sewd.

[3] *Сглаживание Лапласа*
URL: https://vk.cc/c3sexP.

[4] *Функция Softmax*
URL: https://slaystudy.com/implementation-of-softmax-activation-function-in-c-c/.