Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: А.Д. Волков
Преподаватель: С. А. Сорокин
Группа: М8О-306Б

Дата: 06.11.2024
Оценка:
Подпись:

Москва, 2024

Курсовой проект

Задача: Необходимо реализовать наивный байесовский классификатор, который будет обучен на первой части входных данных и классифицировать им вторую часть.

1 Описание

Требуется реализовать наивный байесовский классификатор. Наивный байесовский классификатор — вероятностный классификатор на основе формулы Байеса со строгим (наивным) предположением о независимости признаков между собой при заданном классе, что сильно упрощает задачу классификации из-за оценки одномерных вероятностных плотностей вместо одной многомерной.

В данном случае, одномерная вероятностная плотность — это оценка вероятности каждого признака отдельно при условии их независимости, а многомерная — оценка вероятности комбинации всех признаков, что вытекает из случая их зависимости. Именно по этой причине данный классификатор называется наивным, поскольку позволяет сильно упростить вычисления и повысить эффективность алгоритма.

Сама формула Байеса выглядит следующим образом:

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

где:
$P(A|B)$ — апостериорная вероятность события A при условии выполнения события B;
$P(B|A)$ — условная вероятность события B при условии выполнения события A;
$P(A)$ и $P(B)$ — априорные вероятности событий A и B соответственно.
А в контексте машинного обучения формула Байеса приобретает следующий вид:

$$P(y_k|X) = \frac{P(y_k)P(X|y_k)}{P(X)}$$

где:
$P(y_k|X)$ — апостериорная вероятность принадлежности образца к классу y_k с учётом его признаков X;
$P(X|y_k)$ — правдоподобие, то есть вероятность признаков X при заданном классе y_k;
$P(y_k)$ — априорная вероятность принадлежности случайно выбранного наблюдения к классу y_k;
$P(X)$ — априорная вероятность признаков X.
Если объект описывается не одним, а несколькими признаками X_1, $X2$,...,Xn, то формула принимает вид:

$$P(y_k|X_1, X_2, ..., X_n) = \frac{P(y_k)\prod_{i=1}^{n} P(X_i|y_k)}{P(X_1, X_2, ..., X_n)}$$

На практике числитель данной формулы представляет наибольший интерес, поскольку знаменатель зависит только от признаков, а не от класса, и поэтому часто он опускается при сравнении вероятностей разных классов. В конечном счёте правило

классификации будет пропорционально выбору класса с максимальной апостериорной вероятностью:

$$y_k \propto \text{argmax}_{y_k} P(y_k) \prod_{i=1}^{n} P(X_i|y_K)$$

Для оценки параметров модели, то есть вероятностей $P(y_k)$ и $P(X_i|y_k)$, обычно применяется метод максимального правдоподобия, который в данном случае основан на частотах встречаемости классов и признаков в обучающей выборке.[1]

2 Исходный код

Сначала мы считываем данные тренировочной выборки. В вектор target мы заносим значения классов для каждого документа, а в вектор train_sample мы заносим сами документы. Далее мы создаем объект класса TNaiveBayes и тренируем его по считанным данным.
Разберемся как работает метод fit у объекта класса TNaiveBayes. В самом начале мы высчитываем априорную вероятность принадлежности случайно выбранного документа к классу 0, т.е. $P(y_0)$. Так как у нас всего два класса, априорную вероятность для класса 1 считать не надо, потому что она высчитывается как 1 - априорная вероятность класса 0. Затем мы начинаем цикл по каждому документу в выборке. Каждый документ мы разбиваем на отдельные слова, и получаем вектор слов. При этом, мы должны все слова привести к нижнему регистру, так как в нашей парадигме слова Cat и cat должны быть одинаковыми. Далее мы запускаем цикл по всем словам в документе и считаем частоту встречи этого слова в тренировочной выборке. Для этого у каждого объекта класса TNaiveBayes есть неупорядоченный словарь типа std::unordered_map<std::string, TWordStats>. То есть в данном словаре, мы каждому слову даем характеристику частоты встреч в тренировочной выборке для каждого класса. При этом в объекте класса TWordStats мы храним вектор чисел, в котором хранятся лишь два числа: а нулевом элементе вектора мы храним частоту встреч слова в документах нулевого класса, а в первом элементе аналогично для первого класса. Т.е. когда в нашем словаре уже есть слово которое мы встретили, то мы просто икрементируем частоту в необходимом классе. Если мы еще не встречали этого слова, то мы создаем новый объект класса TWordStats, инкрементируем частоту в необходимом классе и добавляем слово и его статистику в словарь. Затем мы запоминаем размер словаря после процесса обучения, а также нам необходимо просуммировать все частоты для каждого класса отдельно, так как нам это понадобится для подсчета вероятности каждого слова.
Далее мы запускаем цикл по тестовым документам. Мы считываем один тестовый документ, а затем для него вызываем метод predict для предсказания принадлежности документа к какому-либо классу.
Разберемся как работает метод predict. В методе predict мы высчитываем вероятность принадлежности документа к нулевому и первому классу, затем берем из них наибольшую вероятность и определяем по этой вероятности к какому классу принадлежит документ. Теперь разберемся как работает метод probability_calculator для подсчета вероятности. Перед этим необходимо сказать, что метод подсчета вероятностей обосновывается на методе максимального правдоподобия. Таким образом, вероятности принадлежности слова к какому-либо классу подсчитываем делением кол-ва этих слов в данном классе на суммарное кол-во частот всех слов в этом классе. При таком подходе может вохникнуть проблема нулевых вероятностей. Она возникает тогда, когда в тестовой выборке нам встречается слово, которого еще не было в тре-

нировочной выборке. Таким образом, частота этого слова в тестовой выборке - 0 и при делении 0 на сумму частот мы все равно получим 0. И в результате тоже получим нулевую вероятность. Такого допустить мы не можем, поэтому мы должны применить алгоритм сглаживания чтобы исправить эту ситуацию. Я воспользовался методом сглаживания Лапласа. Он заключается в том, чтобы к каждой частоте прибавить 1, чтобы избавиться от нулевых частот. Таким образом, для нулевых частот мы получим частоту 1. Теперь перейдем к методу. Заводим переменную для вероятности со значением 1. Далее запускаем цикл по каждому слову в документе. Если в тестовой выборке не было попавшегося слова, либо попавшегося слова не было в тренировочных документах с определенным классом, то вероятность высчитывается как $\frac{1}{\Sigma_{i=1}^{n} frequency(w_i, class)+n}$. Здесь мы к знаменателю должны прибавить объем тренировочной выборки, так как при выполнении сглаживания Лапласа мы прибавили к каждой частоте в тренировочной выборке 1. Далее, если слово есть уже в словаре, то просто берем его частоту, прибавляем 1 и делим все это на сумму частот и объема тренировочной выборки, т.е. $\frac{frequency(w_j, class)+1}{\Sigma_{i=1}^{n} frequency(w_i, class)+n}$. Эти вероятности мы должны посчитать для каждого слова в тестовом документе, а затем по формуле получившуюся вероятность умножить на априорную вероятность заданного класса. После всех этих вычислений мы получили вероятность принадлежности документа к классу, а затем сравнить эти вероятности для каждого класса и получить ответ.

```
1  #include <bits/stdc++.h>
2
3  // Foo for splitting documents on words by delimeters "!", "?", ".", "," and " "; "!",
       "?", "." theese delimeters needs, because sentence could ending with theese
       symbols
4  // references to: https://ru.stackoverflow.com/questions/1469554/%D0%A0%D0%B0%D0%B7%D0
       %B1%D0%B8%D1%82%D1%8C-%D1%81%D1%82%D1%80%D0%BE%D0%BA%D1%83-%D0%BD%D0%B0-%D1%81%D0%
       BB%D0%BE%D0%B2%D0%B0-%D0%A1
5  std::vector<std::string> WordsSplit(std::string sentence)
6  {
7      std::stringstream ss(sentence);
8      std::vector<std::string> words;
9      const char* const delimeters = "!?., ";
10     std::string line;
11     while (std::getline(ss, line))
12     {
13         char *token = std::strtok(line.data(), delimeters);
14         while (token != nullptr)
15         {
16             words.push_back(token);
17             token = std::strtok(nullptr, delimeters);
18         }
19     }
20     return words;
21 }
22
```

```cpp
// Foo for turning words in low registry, this needs becase in our paradigma Cat and cat are the same words
std::string ToLower(std::string input)
{
    for (size_t i = 0; i < input.size(); ++i)
    {
        input[i] = std::tolower(static_cast<unsigned char>(input[i]));
    }
    return input;
}

class TWordStats
{
public:
    // Default constructor
    TWordStats() = default;
    // stat fields
    std::vector<int> frequencies = {0, 0}; //[0] - zero class, [1] - first class
};

// argmax p(C = c) *  p(F_i = f_i | C = c)
// C - classes
// F_i - words
class TNaiveBayes
{
// 0 - zero class, 1 - first class
private:
    std::unordered_map<std::string, TWordStats> word_stats; //{word: [frequency in zero class, frequency in first class], ...}
    // prior probability of zero class, we don't need prior probability of first class, because this calculates as 1 - prior probability of zero class
    long double prior_probability_zero;
    // fixed dict size
    std::size_t dict_size;
    // summs of frequencies
    std::vector<int> freqs;
    // method for summaring frequencies of each class, this needs for calculating prior probability of classes
    std::vector<int> sum_freqs()
    {
        int zero_count = 0;
        int first_count = 0;
        for (const auto& pair : this->word_stats)
        {
            zero_count += pair.second.frequencies[0];
            first_count += pair.second.frequencies[1];
        }
        return {zero_count, first_count};
    }
```

```
68      // Method for calculating result probability of class for sentence
69      long double probability_calculator(std::vector<std::string>& words, int class_id)
70      {
71          long double probability = 1;
72          for (size_t i = 0; i < words.size(); ++i)
73          {
74              // Laplas smoothing
75              if (!this->word_stats[words[i]].frequencies[class_id])
76              {
77                  probability *= 1.0 / (freqs[class_id] + this->dict_size);
78              }
79              else
80              {
81                  probability *= 1.0 * (this->word_stats[words[i]].frequencies[class_id] +
                        1) / (this->freqs[class_id] + this->dict_size);
82              }
83
84          }
85          probability *= 1.0 * (class_id == 1 ? 1 - this->prior_probability_zero : this->
                prior_probability_zero);
86          return probability;
87      }
88  public:
89      // Default constructor
90      TNaiveBayes() = default;
91      // fit method
92      void fit(std::vector<std::string>& X, std::vector<int>& y)
93      {
94          this->prior_probability_zero = float(std::count(std::cbegin(y), std::cend(y),
                0)) / y.size();
95          for (size_t i = 0; i < X.size(); ++i)
96          {
97              std::vector<std::string> words = WordsSplit(ToLower(X[i]));
98              for (size_t j = 0; j < words.size(); ++j)
99              {
100                 // if we already have the word in dict, we just incrementing its
                        frequency for specify class
101                 if (this->word_stats.count(words[j]))
102                 {
103                     word_stats[words[j]].frequencies[y[i]]++;
104                 }
105                 // else, we add new word in our dict
106                 else
107                 {
108                     TWordStats stats;
109                     stats.frequencies[y[i]]++;
110                     word_stats[words[j]] = stats;
111                 }
112             }
```

```
113         }
114         this->dict_size = word_stats.size();
115         this->freqs = this->sum_freqs();
116     }
117 
118     // predict method for one sentence
119     int predict(std::string X)
120     {
121         std::vector<std::string> words = WordsSplit(ToLower(X));
122         // calculating probabities of each class for this sentence
123         long double probability_zero = this->probability_calculator(words, 0);
124         long double probability_first = this->probability_calculator(words, 1);
125         // take argmax
126         return probability_zero > probability_first ? 0 : 1;
127     }
128 };
129 
130 int main()
131 {
132     int train_files_count, test_files_count;
133     std::vector<std::string> train_sample;
134     std::vector<int> target;
135 
136     // reading data
137     std::cin >> train_files_count >> test_files_count;
138     for (int _ = 0; _ < train_files_count; _++)
139     {
140         int class_id;
141         std::string train_file_text;
142         std::cin >> class_id;
143         target.push_back(class_id);
144         std::cin.ignore();
145         getline(std::cin, train_file_text);
146         train_sample.push_back(train_file_text);
147     }
148     TNaiveBayes classificator;
149     classificator.fit(train_sample, target);
150     for (int _ = 0; _ < test_files_count; _++)
151     {
152         std::string test_file_text;
153         getline(std::cin, test_file_text);
154         std::cout << classificator.predict(test_file_text) << "\n";
155     }
156 }
```

main.cpp	
std::vector<std::string> WordsSplit()	Функция разбиения документа на слова

std::string ToLower()()	Функция перевода строки в нижний регистр
std::vector<int> TNaiveBayes::sum_freqs()	Метод для суммирования всех частот в каждом классе
long double TNaiveBayes::probability_calculator()	Метод для подсчета вероятности принадлежности тестовых документов к определенному классу
void TNaiveBayes::fit()	Метод для обучения модели
int TNaiveBayes::predict()	Метод для предсказания класса отдельно взятого документа
int main()	Точка входа программы

```
1  class TWordStats
2  {
3  public:
4      TWordStats() = default;
5      std::vector<int> frequencies = {0, 0};
6  };
7  class TNaiveBayes
8  {
9  private:
10     std::unordered_map<std::string, TWordStats> word_stats;
11     long double prior_probability_zero;
12     std::size_t dict_size;
13     std::vector<int> freqs;
14     std::vector<int> sum_freqs()
15     long double probability_calculator(std::vector<std::string>& words, int class_id);
16 public:
17     TNaiveBayes() = default;
18     void fit(std::vector<std::string>& X, std::vector<int>& y);
19     int predict(std::string X);
20 };
```

3 Консоль

```
lexasy@lexasy$ cat test.txt
4 1
0
Hi,how are you?
1
Congratulations,you won a prize!
1
```

```
Buy the product now and get a discount!
0
Let’s walk this evening
Hi,you won a discount and you can get the prize this evening.
lexasy@lexasy$ g++ main.cpp
lexasy@lexasy$ ./a.out <test.txt
1
lexasy@lexasy$ cat test1.txt
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.lexasy@lexasy:~/Prog/DA_labs/cw$
lexasy@lexasy$ ./a.out <test1.txt
0
1
```

4 Тест производительности

Тест производительности представляет из себя замер метрик presision и recall для предсказаний на синтетических данных. Вот сам датасет:
20 10
0
The dog is playing in the park with a ball.
0
The cat is sleeping on the windowsill, sunny day.
1
Today I bought a new computer.
0
The weather is beautiful, a perfect day for a walk.
1
I am studying machine learning and artificial intelligence.
0
Children are playing in the sandbox.
1
The new book on programming is very interesting.
0
The old house at the corner of the street.
1
The movie was thrilling and full of unexpected twists.
0
I love cooking and trying new recipes.
1
Scientific research opens new horizons.
0
Birds sing in the spring when the trees bloom.
1
I started exercising and feel better.
0
Snow is falling, and the streets are turning white.
1
Technology is advancing at an incredible pace.
0
Summer is the time for vacations and relaxation.
1
I write articles about health and wellness.
0
The sun is shining brightly, and the sky is clear.

1
Learning languages opens up new opportunities.
0
In the evening, I enjoy walking in the park.
The puppy is playing with a new toy.
I just finished reading a fascinating book.
The sun is setting, and the sky is beautiful.
I enjoy learning about new technologies.
The children are laughing and having fun.
This movie is one of the best I've ever seen!
Rainy days make me feel cozy and relaxed.
I am excited to start my new job next week.
The flowers are blooming in the garden.
I love exploring new places and cultures.
Программа выдала вот такой результат:
0 1 0 1 0 0 1 1 0 1
Но фактический результат вот такой:
0 1 0 1 0 1 0 1 0 1
По итогу метрики pressision и recall получились вот такими:

```
lexasy@lexasy$ python3 presision_recall.py
Precision: 0.80
Recall: 0.80
```

Это вполне нормальные значения метрик. Для данного датасета это означает, что модель допустила 2 ошибки на 10 тестах.

5 Выводы

Выполнив курсовую работу по курсу Дискретный Анализ я научился реализовывать модель наивного байесовского классификатора. Также я подтянул знания по классификаторам из курса машинного обучения и по методу максимального правдоподобия из курса математической статистики. Но реализованная мною модель точно не работает в практических задачах, так как она игнорирует множество признаков. Но скорее всего, данный курсовой проект позволит столкнуться с меньшим количеством трудностей при знакомстве с реальным текстовым классификатором.

Список литературы

[1] https://habr.com/ru/articles/802435/ - 2024