Московский авиационный институт (национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Дискретный анализ»

Студент: А. А. Каримов
Преподаватель: С. А. Сорокин
Группа: М8О-306Б-22

Дата:
Оценка:
Подпись:

Москва, 2024

Курсовой проект

Задача: Реализуйте 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>`

Ключ	Значение
--input	входной файл с вопросами
--output	выходной файл с рассчитанной статистикой

Формат запуска программы в режиме классификации:
`./prog classify --stats <stat file> --input <in file> --output <out file>`

Ключ	Значение
--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.

1 Описание

Наивный байесовский классификатор (Naive Bayes classifier) — вероятностный классификатор на основе формулы Байеса со строгим (наивным) предположением о независимости признаков между собой при заданном классе. Использоваться будет формула Байеса в виде:

$$P(class|question) = \frac{P(question|class)P(class)}{P(question)},$$

где $P(class)$ - вероятность встретить вопрос класса $class$ среди train-выборки
$P(question)$ - вероятность вопроса $question$ в выборке
$P(question|class)$ - вероятность встретить вопрос $question$ среди всех вопросов класса $class$
$P(class|question)$ - вероятность, что вопрос $question$ принадлежит классу $class$

Предположение независимости признаков: Если в обычной речи слова сильно зависят от контекста, то мы делаем предположение о том, что слова в вопросе друг от друга не зависят. Это даёт нам право представить формулу Байеса в следующем виде:

$$P(class|question) = \frac{P(w_1|class)P(w_2|class)...P(w_n|class)}{P(w_1, w_2, ..., w_n)} = \frac{\prod_{i=1}^{n} P(w_i|class)}{P(w_1, w_2, ..., w_n)},$$

где w_i - это i-ое слово в вопросе question.

Поскольку знаменатель данной дроби зависит только от признаков, а не от класса, его можно опустить и продолжать вычисления по следующей формуле:

$$P(class|question) \approx P(class) \prod_{i=1}^{n} P(w_i|class)$$

Логарифмирование функции: Мы будем по вышеописанной формуле перемножать множество вероятностей $P(w_i|class)$, из-за чего значение функции может быть настолько малым, что мы получим большую погрешность вычислений. Во избежание этого прологарифмируем функцию. Благодаря монотонности log параметры, при которых достигаются оптимальные значения, останутся теми же. Итого получим новую формулу расчета:

$$\log P(class|question) \approx \log P(class) + \sum_{i=1}^{n} \log P(w_i|class)$$

Расчет использующихся вероятностей: Расчет вероятностей будем проводить на основе их частотности:

$$P(class_i) = \frac{amountClass_i}{totalAmountClasses},$$

где $P(class_i)$ - вероятность класса $class_i$
$amountClass_i$ - количество классов $class_i$ в train-выборке
$totalAmountClasses$ - количество всего различных классов в train-выборке

$$P(w_i|class) = \frac{WordCountInClass_i}{UniqueWordCount},$$

где $WordCountInClass_i$ - количество вхождений слова w_i в вопросы класса $class$
$UniqueWordCount$ - количество уникальных слов среди всех классов.

Новые слова при классификации: С последней формулой есть большая проблема: если при классификации мы встретили слово, которое не встречалось в обучающей выборке, то мы получим $WordCountInClass_i = 0$, а значит и вероятность $P(w_i|class)$ будет ноль, а значит и вся формула расчета будет равнятся нулю, то есть мы не сможем классифицировать вопрос, содержащий неизвестное слово. Для решения этой проблемы будем использовать сглаживание Лапласа:

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad (i = 1, \ldots, d),$$

где $\mathbf{x} = \langle x_1, x_2, \ldots, x_d \rangle$ из d-мерного мультиномиальное распределение с N испытаниями. Коэффициент $\alpha = 1$

В нашем случае формула вероятности приобретет вид:

$$P(w_i|class) = \frac{WordCountInClass_i + \alpha}{\alpha * TotalWordCount + UniqueWordCount}$$

Идея заключается в том, чтобы добавить к каждому количеству вхождений слова единицу.

Вероятностное пространство: Поскольку мы считали логарифм от вероятности по основной формуле, мы не имеем чистых значений вероятности для каждого класса. Нам бы не помешало это, если бы мы, например, использовали бинарную классификацию. Тогда мы бы просто сравнили два числа и выдали в ответ наибольшее. Но в нашем случае классификация - многоклассовая, соответственно, такой способ нам не подойдет. Для того, чтобы создать некое вероятностное пространство, нам необходимо для всего вектора вычисленных значений выполнить следующие условия:
1) $value_i \in [0, 1]$

2) $value_1 + ... + value_n = 1$

Для этого будем использовать формулу softmax: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$

2 Описание алгоритма

Этап обучения:

На вход нам дается файл, на основе которого с помощью словаря мы считаем частоты вхождений для каждого встреченного слова. Для каждого тега добавляем расчитанную статистику.

```
void BayesClassificator::BayesTagClassificator::fit(const std::vector<std::string>&
    tags, const text_t& text) {
    std::unordered_map<std::string, size_t> freqDict = getFrequency(text);

    ++trials;

    for(const std::string& tag : tags) {
        fittedTags[tag].add(freqDict, text.size());
    }
}
```

Этап классификации:

Для каждого тега считаем вероятность по вышеописанной формуле, используя статистики рассчитанные на этапе обучения. Из полученного вектора отбираем классы.

```
std::vector<std::pair<std::string, double>> BayesClassificator::BayesTagClassificator
    ::predict(const text_t& text) {
    std::vector<std::pair<std::string, double>> tagsProbs;

    for(auto [tagName, tagInfo] : fittedTags) {
        double tagLogProb = log(tagInfo.tagEntry) - log(trials);
        // double tagLogProb = tagInfo.tagEntry / static_cast<double>(fittedTags.size()
            );
        // std::cout << tagName << ' ' << tagLogProb << std::endl;
        for(const std::string& word : text) {
            // tagLogProb *= (tagInfo.freq[word] + 1) ;
            // tagLogProb /= static_cast<double>(tagInfo.wordsUnderTag);
            tagLogProb += log(tagInfo.freq[word] + 1) - log(tagInfo.wordsUnderTag + 1 *
                uniqueWordsCount());
        }

        tagsProbs.push_back({tagName, tagLogProb});
    }

    tagsProbs = BayesClassificator::softmax(tagsProbs);
    std::vector<std::pair<std::string, double>> predicted;

    for(auto& [tagName, prob] : tagsProbs) {
        if(prob > 1.5 / fittedTags.size())
            predicted.emplace_back(tagName, prob);
    }

```

```
25 ||     return predicted;
26 || }
```

3 Консоль

```
karseny99@karseny99:/mnt/study/DA/lab4/src$ ./lc
Learnt on files/input.learn
Saved stats in files/out.stats
Running next step ...

(Classificator)


Stats imported from files/out.stats
Data to classify from files/in.classify
Classified data in files/out.result
Running next step ...

(Estimator)


===============
AveragePrecision: 0.818182
AverageRecall: 0.19697
===============
Class name: sort
Precision: 1 Recall: 0.166667
===============
Class name: theorem
Precision: 0 Recall: 0
===============
Class name: algorithm
Precision: 1 Recall: 0.166667
===============
Class name: counting_sort
Precision: 1 Recall: 0.166667
===============
Class name: proof
Precision: 0 Recall: 0
===============
Class name: tree
Precision: 1 Recall: 0.333333
===============
Class name: balanced_tree
```

```
Precision: 1 Recall: 0.5
===============
Class name: avl-tree
Precision: 1 Recall: 0.166667
===============
Class name: rbtree
Precision: 1 Recall: 0.333333
===============
Class name: string
Precision: 1 Recall: 0.166667
===============
Class name: suffix_tree
Precision: 1 Recall: 0.166667
karseny99@karseny99:/mnt/study/DA/lab4/src$ cat ./lc
make
./main learn --input files/input.learn --output files/out.stats

./main classify --stats files/out.stats --input files/in.classify --output
files/out.result

./estimator --answer files/in.ans --prediction files/out.result
```

Оценка качества: Для оценки качества используем precision и recall. Вычислим их обычным способом для каждого класса в отдельности, а затем усредним их. Такой метод учитывает дисбаланс классов. Соответственно, общая оценка модели:
$AveragePrecision$: 0.818182
$AverageRecall$: 0.19697
Первый параметр показывает, что модель правильно предсказывает конкретный класс. Второй параметр показывает, что модель какие-то классы не приводит в предсказании, хотя они есть. Если в первом случае precision близок к единице, что хорошо, то во втором качество модели не самое лучшее. Улучшить этот параметр можно с помощью расширения train-выборки.

4 Выводы

В ходе выполнения курсового проекта я смог реализовать наивный Байесовский классификатор. До этого я уже встречался с этим классификатором в статье Пола Грэма (2003). В те времена в электронную почту не были встроены умные системы фильтрации спама, поэтому Пол применил Байесовскую фильтрацию к своим приходящим письмам. На примерах он показывал, как определенные словесные конструкции на фоне огромного текста сильно повышают вероятность на то, что письмо спам. При первом прочтении я мало что понял, но сейчас, познакомившись с курсом машинного обучения, с некоторой теорией я смог лучше разобраться в применениях вероятностной формулы Байеса.

Список литературы

[1] Блог *разработчика программного обеспечения.*
URL: http://bazhenov.me/blog/2012/06/11/naive-bayes.html

[2] Блог *разработчика программного обеспечения.*
URL: https://habr.com/ru/articles/802435/

[3] Блог *разработчика программного обеспечения.*
URL: https://www.bazhenov.me/blog/2012/07/21/classification-performance-evaluation

[4] Блог *разработчика программного обеспечения.*
URL: https://habr.com/ru/articles/661119/

[5] *Лекции по Машинному обучнию*