Курсовой проект 23/24 по курсу дискретного анализа

Выполнил студент группы 08-307 МАИ *Путилин Дмитрий*.

Условие

Автоматическая классификация документов

Реализуйте систему, которая на основе базы вопросов и тегов к ним, буде 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 <stats file> \
                --input <input file> \
                --output <input 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.

Метод решения

Теория

В машинном обучении под классификацией понимают задачу определения категории, к которой принадлежит ранее не встречавшийся образец, на основании обучающего множества, для элементов которого эти категории известны. Это является примером обучения с учителем (supervised learning).

Дано:

Каждый пример x принимает значения из множества V и описывается атрибутами $\langle a_1, a_2, \ldots, a_n \rangle$.

Нужно найти наиболее вероятное значение данного атрибута, т.е.

$$v_{MAP} = argmax_{v \in V} p(x = v | a_1, a_2, \ldots, a_n)$$

По теореме Байеса

$$v_{MAP} = argmax_{v \in V} \frac{p(a_1, a_2, \ldots, a_n | x = v) p(x = v)}{p(a_1, a_2, \ldots, a_n)} = argmax_{v \in V} p(a_1, a_2, \ldots, a_n | x = v) p(x = v)$$

Предположим условную независимость атрибутов при условии данного значения целевой функции. Иначе говоря:

$$p(a_1, a_2, \ldots, a_n | x = v) = p(a_1 | x = v) p(a_2 | x = v) \ldots p(a_n | x = v).$$

Итак, наивный байесовский классификатор выбирает v :

$$v_{NB} = argmax_{v \in V} p(x = v) \prod_{i=1}^{n} p(a_i | x = v)$$

Так как на тренировочных данных может быть очень много данных, следовательно необходимо формулу для выбора класса прологарифмировать:

$$v_{NB} = argmax_{v \in V} [\log p(x = v) + \sum_{i=1}^{n} p(a_i | x = v)]$$

Также отметим, что если на тренировочных данных у нас встретилось слово, которого раньше не было, то это занулит вероятности. Во избежание данной проблемы используется метод сглаживания Лапласа.

$$p(a_i | x = v) = \frac{\#(X = a_i) + \alpha}{N + m * \alpha},$$

где

$\#(X = a_i)$ - количество слова a_i в статистике,

N - количество слов в статистике для класса v,

m - количество уникальных слов в тестовом запросе

Рассмотрим еще один прием для тюнинга нашей модели. Для этого после того, как каждая строчка была токенизирована, выполняется проверка на то, является ли слово "стоп-словом"(набор слов, которые не должны участвовать в классификации) : ("i", "me", "my", "$myself$", "we", "our", ...).

При токенизации самих слов внтури строки, также проверяется на то, чтобы не было различных символов из набора "стоп-символ": (" " , "." , "!" , "," , "," , " : ", ...).

Так как после вычисления score для запроса получаются очень большие значения, то данные значения отнормируем на длину входящего запроса и возведем в степень e.

Теперь разберемся с отбором тегов для входящего запроса. У нас после применения к каждому тегу вычисления score, образуются набор scores. Необходимо теперь из него выбрать подходящие и вывести нужный тег. Для этого воспользуемся функцией $softmax$

$$softmax(a_i) = \frac{a_i}{\sum_{k=1}^{n} a_k},$$

n = $len(scores)$

Дальше происходит отбор нужных значений. Для этого вводится константа treshold. Выбираем те, которые больше данного назначения и записываем тег в ответ.

Описание программы

Класс NaiveBayes представляет собой класс для наивного байесовского классификатора. В привате находятся map-ы для хранения статистики слов и классов на train выборке, . В публичном доступе находятся методы: fit и predict, сохранение в фай л ($save\_json$) и загрузка из файла ($load\_json$) статистики . В методе fit происходит обучение на тренировочных данных, в predict - соответственно происходит прогнозирование для тестовых данных.

Тест производительности

Для тренировочных данных используем такой набор данных: 20 тренивочных текстов с различным числом тегов и строк. Тегов 4 вида: c++, python, telegrambot, programming. На тестовой выборке 17 текстов с различным числом строк.

		Predicted				
		c++	python	telegrambot	programming	None
Actual	c++	2	2	1	0	0
	python	0	4	0	0	0
	telegrambot	0	2	4	0	1
	programming	0	1	1	2	2
	None	0	1	1	2	0

Теперь на тестовых данных посмотрим метрику. Выберем для этого f1-меру.

$$Precision_c = \frac{TP_c}{TP_c + FP_c}$$

$$Recall_c = \frac{TP_c}{TP_c + FN_c}$$

$$precision = \frac{\sum_{c=1}^{n} P_c}{n}$$

$$recall = \frac{\sum_{c=1}^{n} R_c}{n}$$

$$P_1 = 1$$

$$P_2 = 0.4$$

$$P_3 = 0.57$$

$$P_4 = 0.5$$

$$R_1 = 0.4$$

$$R_2 = 1$$

$$R_3 = 0.57$$

$$R_4 = 0.33$$

$$precision = 0.494$$

$$recall = 0.46$$

$$F1 = \frac{2 * precision * recall}{precison + recall} = 0.48$$

Выводы

Построенная модель неплохо классифицирует документы, добавляя им соответствующие теги, с F1-мерой ≈ 0.5.