Курсовая работа по курсу дискретного анализа: классификация документов

Выполнил студент группы М8О-307Б-21 МАИ *Кажекин Денис*.

Условие:

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 <output 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:

После анализа поставленной задачи, мне показалось, что разумным решением будет применение алгоритма Наивного Байеса несколько раз. А если быть точным, то ровно столько, сколько тэгов участвует в распределении на каждый документ. Таким образом, мы проводим бинарную классификацию несколько раз и, устанавливая пороговое значение, отсекаем те тэги, которые имеют недостаточную уверенность.

В работе использовались следующие эвристики: добавил сглаживание Лапласа, которое предотвращает обнуление вероятности класса из-за невстретившихся слов в обучающей выборке. В этой эвристике к вычислению условной вероятности слова

нужно добавить по одному «фиктивному» слову. То есть мы говорим, что пусть каждое слово встретилось на один раз больше. Таким образом, уходим от нулевой вероятности.

Также были применены аугментации над текстом. Я использовал лемматизацию для приведения всех слов в начальную форму, чтобы уменьшить количество коллизий надо словами, нормирование (приведение всех слов в нижний регистр), удаление знаков пунктуации, удаление стоп-слов, которые никак не влияют на вероятность тэга, а лишь уменьшают условную вероятность.

Так как датасет не нормирован относительно классов, мне пришлось использовать метрики micro-precision и micro-recall. Которые не учитывают доли тэгов относительно других.

Описание программы

Проект многофайловый. В директории materials находятся заранее заготовленный файлы с данными. В эту же директорию должен быть установлен датасет, если есть желание создать свою собственную выборку, которую необходимо прогнать через алгоритм. Также помимо materials можем видеть питоновский скрипт preprocessing.py. Он используется для аугментаций над текстами, по итогу его цель – подготовить входные файлы для модель. Пара nb_helper.cpp и nb_helper.h составляют вспомогательный заголовочный файл, который немного разгружает программу и улучшает ее логику в основном файле. В nb.cpp содержится логика всей программы. Помимо всего можем увидеть файл requirements.txt, который содержит все зависимости необходимые для питоновского скрипта, и мейкфайл для удобного взаимодействия с моделью.

Недочёты

Недочетов не выявлено

Выводы

Мне очень понравился курсовой проект. Так как я увлекаюсь машинным обучением, мне показалось очень полезным реализовать и разобраться в Байесовском алгоритме. Как оказалось, Байесовский алгоритм, несмотря на свою простоту, показывает впечатлительные результаты на задачах бинарной и мультиклассовой классификации.