

# Марковский генератор текстов $n$ -ого порядка

Поликарпов Андрей

21 октября 2015 г.

# 1 Постановка задачи

Требуется реализовать марковский генератор текста  $n$ -ого порядка. Логически состоит из двух исполняемых файлов: обучающего и генерирующего на основе обученной модели.

1. Обучающий - получает на вход текстовый файл в ASCII кодировке, содержащий текст на естественном языке. Пунктуация не важна, но желательно привести текст к единому регистру, чтобы увеличить заполняемость цепи. Также задается параметр  $N$  - порядок цепи. По входному тексту строится марковская цепь, результат сериализуется в файл на жестком диске.
2. Генерирующий - получает на вход аналогичный предыдущему текстовый файл, содержащий начальный отрывок из  $n$  слов и число  $K$  - количество слов, которые надо достроить к отрывку на основании созданной предыдущим исполняемым файлом марковской цепи. Полученный текст нужно выводить на поток стандартного вывода.

# 2 Как решена задача

## 2.1 Построение цепи

Марковская цепь хранится как ассоциативный массив (`std::map`), в котором ключом является список слов `std::list<std::string>`, а значением еще один ассоциативный массив (`std::map<std::string, int>`). Размер ключа - порядок марковской цепи  $n$ .

Построение происходит таким образом:

Исходный список слов пустой. Для него ассоциируется следующий массив: `<первое слово> → 1`. Далее, список слов снова пустой, кроме последнего элемента. Этот элемент равен первому слову. Такому ключу сопоставляется массив: `<второе слово> → 1`. Таким образом заполняются первые  $n$  элементов ассоциативного массива. Далее, когда в списке слов уже нет пустых элементов, ключом являются первые  $n$  слов исходного текста. Идет поиск такого списка слов в тексте, в качестве значения в ассоциативном массиве будут слова, которые идут следующими после вхождения такого списка в исходном тексте, вместе с количеством вхождений. Например:

Это утверждение. Это следующее утверждение. Это еще одно утверждение.

Тогда этот текст будет представлен в таком виде:

`[ " ", " " ] → { "это" = 1 }`

`[ " ", "это" ] → { "утверждение" = 1 }`

`[ "это" , "утверждение" ] → { "это" = 1 }`

`[ "утверждение" , "это" ] → { "следующее" = 1, "еще" = 1 }`

... и так далее.

Исходный текст приводится к одному регистру, все знаки препинания и цифры удаляются.

## 2.2 Генерация текста

Есть список слов  $T$  который надо продолжить. Также есть построенная марковская цепь  $n$ -ого порядка. Для генерации следующего слова происходит поиск в такой таблице по следующему ключу:

Ключом является список из последних  $n$  слов из списка  $T$  в порядке следования. Следующее слово будет выбрано как значение по этому ключу в марковской цепи с учетом частоты. Например:

Пусть для ключа `["this" , "is"]` значением в цепи является `{ "ololo" = 1, "the" = 2 }`. При обращении по такому ключу в двух случаях из трех вернется "the". Пусть возвращено слово "the". Далее, ключ обновится и станет `["is" , "the"]`. Процесс продолжается.

В данной реализации, если возвращаемое слово пустое, то процесс будет продолжен, но уже с полностью пустым ключом, для которого точно есть значение в цепи в виде первого слова с частотой 1. Процесс закончится пока не будет построено  $K$  слов.

# 3 Сериализация

Для сериализации/десериализации используется библиотека Protocol Buffers.

.proto-файл приложен к проекту.

# 4 Присутствующие недостатки

1. Способ удаления знаков препинания подразумевает наличие после знака препинания пробела. В реальности это не всегда так.

2. Данные из файла в итоге хранятся в `std::vector`, возможно, это не очень хорошее решение(с точки зрения производительности).
3. Алгоритм поиска совпадений двух списков (`std::list`) в функции `Fit()` не оптимален.<sup>1</sup>

## 5 Что можно улучшить

1. Добавить тесты для генератора + добавить Doxygen-документацию.
2. Протестировать производительность + исправить (возможно, хранить все слова не в `std::list<std::string>`, а просто в `std::string` с разделителем).
3. Возможно, стоит использовать FlatBuffers, а не Protocol Buffers.
4. Сделать `configure` или сделать на `make`.
5. Сделать полноценный класс для исключений.
6. Добавить возможность генерировать отдельные буквы, а не слова.
7. Добавить markdown Readme в репозиторий.

## 6 Дополнительная информация

Для работы программы необходимо установить Google Protocol Buffers.

Используется C++11.

В файле `run` примеры запуска программы.

В файле `input.txt` содержатся субтитры к фильму "Брат-2" на английском языке.

---

<sup>1</sup> в приложенном файле `input.txt` 6207 слов, порядок цепи равен 3, обучение занимает примерно 16 секунд. Если порядок цепи 2 - примерно 11 секунд. Debian 8, Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz, RAM Size: 4096 MB, SSD 120 GB

## Список литературы

- [1] Markov Chain
- [2] Coursera Text Mining
- [3] ProtoBuf GitHub