Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Петрозаводский государственный университет»
Институт математики и информационных технологий

09.03.04 - Программная инженерия

Профиль направления подготовки бакалавриата

«Системное и прикладное программное обеспечение»

Отчёт о прохождении производственной практики

Выполнил:
студент 2 курса группы 22207

Павлов Максим Павлович

Место прохождения практики:
Кафедра информатики и математического обеспечения

Сроки прохождения практики:
с 29.05.2023 по 11.06.2023

Руководитель практики:
к.т.н., доцент
Богоявленская Ольга Юрьевна

Оценка _____

Дата _____

Петрозаводск — 2023

Содержание

Введение

В наши дни методы генерации текста по математическим моделям (Natural Language Generation) набирают всё большую популярность, появляются всё новые практические примеры их использования в повседневной жизни. Но современные решения, как правило, основываются на колоссальных нейронных сетях, включающих в себя триллионы нейронов. Но, если ограничиться задачей генерации текста по его статистическим моделям, то такая задача изящно решается с помощью алгоритма на основе Марковских цепей [1].

Цель практики - реализовать алгоритм построения статистики суффиксов и префиксов по заданным текстам.

Задачи производственной практики:

1. Ознакомление с теорией и литературой по генерации текстов с помощью Марковских цепей;
2. Создание собственной структуры данных для хранения префиксов текста на C++ и её последующая интеграция в Python;
3. Создание программного модуля по подсчёту и анализу префиксов и суффиксов в тексте;
4. Тестирование разработанного модуля и структуры данных.

Организация и кооперация с другими разработчиками данной задачи (Кирилловым Иваном и Афанасьевым Артёмом), а также контроль версий программного кода и распределение подзадач осуществлялось с помощью GitHub. Моя роль в команде заключалась в ведении GitHub, подключении структуры данных для хранения префиксов на C++ в Python, создании модуля по подсчёту статистики префиксов, а также его тестирование.

1 Создание модуля-обёртки структуры данных для хранения префиксов и суффиксов в тексте

Модули, написанные на C++ (или C), обычно используются для расширения возможностей интерпретатора Python. Они также обеспечивают доступ к низкоуровневым возможностям операционной системы. Их, как правило, можно разделить на три следующих основных типа [2]:

1. Модули ускорения. Так как Python является интерпретируемым языком, можно написать модули ускорителя на C++ для повышения производительности.

2. Модули-оболочки. Они открывают существующие интерфейсы C/C++ для кода Python или предоставляют адаптированный API, который удобно использовать в дальнейшем.

3. Модули низкоуровневого системного доступа. Они создаются для доступа к низкоуровневым функциям среды выполнения CPython, операционной системы или базового оборудования.

При реализации алгоритма построения статистики суффиксов и префиксов по заданным текстам была написана собственная структура данных на языке C++, в основе которой лежит бор указателей, чтобы максимально оптимизировать и ускорить алгоритм Маркова. Для её дальнейшего использования в Python необходим соответствующий модуль-оболочка. При решении данной подзадачи наш выбор остановился на библиотеке pybind11 [3]. Это облегчённая библиотека Boost.Python, разработанная как раз для создания модулей-оболочек C++ структур в Python и наоборот. Её основными преимуществами являются: кроссплатформенность, скорость выполнения, поддержка всех необходимых функций и инструментариев для создания обёрток структур данных.

Для создания модуля-оболочки [4, 5] был создан cmake файл для сборки библиотеки, программа-сборщик setup.py для создания обёртки под конкретную ОС и архитектуру процессора, внесены дополнительные описания методов структуры на C++ и её краткое описание в секцию PYBIND11_MODULE. В результате с помощью библиотеки pybind11 созданная структура была обёрнута в python .so и .pyd библиотеки (модули-оболочки) - "Статистикум"(StatistiCuM). Обёртка класса была создана для двух ОС Windows и Linux и архитектуры процессоров Intel (файлы .pyd и .so соответственно).

2 Модуль подсчёта статистик префиксов

Для подсчёта статистик и анализа префиксов в текстах был разработан Python-класс PrefixStat. При его инициализации достаточно указать строку с текстом (возможна передача и многострочного текста) в первом аргументе, и количество слов в префиксе во втором. Каждый из методов также оснащён Python типизацией. Разработанная структура PrefixStat поддерживает следующие методы для работы с префиксами в текстах:

1. Добавление текста в структуру данных для анализа;
2. Создание списка n по популярности префиксов в i-ом тексте;
3. Создание списка n по популярности суффиксов после заданного префикса;
4. Средняя частота встречаемости заданного префикса в текста;
5. Максимальная частота употребления заданного префикса в текстах.

Листинг 1: Добавление нового текста для последующей аналитики

```
1   def add(self, text, k) -> None:
2       """Добавление текста для анализа статистики префиксов"""
3       assert(type(text) == str and type(k) == int)
4       self.stat.append(StatistiCuM.statistic_counter())
5       self.text.append(list(filter(lambda word: word != '',
6                                    text.translate(str.maketrans('', '', string.punctuation)) \
7                                        .replace('\t', '') \
8                                        .replace('\n', '') \
9                                        .split(' '))))
10      index = len(self.stat)-1
11      for cur in range(len(self.text[index]) - k + 1):
12          prefix = ""
13          for word_id in range(k):
14              prefix += self.text[index][word_id + cur] + ' '
15          prefix = prefix[:len(prefix)-1]
16          self.stat[index].add(prefix.lower())
```

Добавление текста в структуру для анализа осуществляется с помощью метода add (Листинг 1). В его основе лежит разбиение текста по пробелам, удаление из него знаков

препинания и табуляции, разбиение текста на префиксы, а также заполнение ими разработанной структуры данных.

Листинг 2: Создание списка n по популярности префиксов в структуре данных

```
17    @private
18     def most_common_in_statistic(self, stat, n, with_number=False) -> List[List]:
19        """Создание списка из n по частоте элементов в структуре stat"""
20        stat.set_pointer(0)
21        arr = [[ ]]
22        count = 0
23        last_n = current_n = None
24        s = stat.get_next()
25        while s != ' ':
26            if s == '':
27                break
28            data = s.split(' ')
29            prefix = ' '.join(data[:-1])
30            last_n = current_n; current_n = data[len(data) - 1]
31            if current_n != last_n and last_n is not None:
32                arr.append([ ])
33                count += 1
34            if count == n:
35                break
36            arr[count].append(prefix+ ' ' + current_n if with_number else prefix)
37            s = stat.get_next()
38        stat.set_pointer(0)
39        return list(filter(lambda x: x != [ ], arr))
40    def most_common_in_text(self, index, n) -> List[List]:
41        """амыеС часто встречающиеся префиксы в данном текстов"""
42        if type(index) != int or type(n) != int or n < 1 or
43            index < 0 or index > len(self.stat)-1:
44            return [[ ]]
45        return self.most_common_in_statistic(self.stat[index], n)
```

Одной из важных характеристик текста является частота использования определённых

слов. Для составления списка n по популярности префиксов был разработан метод класса PrefixStat - most_common_in_text (Листинг 2). В его основе лежит private метод (его вызов за пределами класса невозможен) most_common_in_statistic, составляющий такой список по разработанной структуре данных. Данное разбиение на методы будет важно в дальнейшем, чтобы не повторять методы, схожие по функциональности.

Листинг 3: Создание списка n по популярности суффиксов, следующих после определённого префикса в i-ом тексте

```
46    def most_common_in_word(self, index, prefix, n, with_number=False) -> List[List]:
47        """Создание списка n по популярности суффиксов, следующих после
      определённого префикса"""
48        if type(prefix) != str or type(index) != int or type(n) != int or
49            n < 1 or index < 0 or index > len(self.stat)-1:
50            return [[ ]]
51        s = prefix.split(' ')
52        suffux = StatistiCuM.statistic_counter()
53        for cur in range(len(self.text[index]) - len(s) + 1):
54            current_prefix = ""
55            for word_id in range(len(s)):
56                current_prefix += self.text[index][word_id + cur] + ' '
57            current_prefix = current_prefix[:len(current_prefix) - 1]
58            if current_prefix == prefix:
59                suffux.add(self.text[index][word_id + cur + 1])
60        return self.most_common_in_statistic(suffux, n, with_number=with_number)
```

Одной из самых важных статистик префиксов для алгоритма Маркова является составление списка из n по популярности суффиксов, следующих после определённого префикса. Данную характеристику реализует метод most_common_in_word (Листинг 3). Для его реализации и потребовалась описанная ранее декомпозиция программных функций на методы. Помимо индекса анализируемого текста, ограничения списка на количество мест и заданного префикса в метод можно также передать флаг with_number, чтобы получить помимо суффиксов и частоту их употребления после заданного префикса. Например, результат работы метода для составления списка из трёх по популярности суффиксов может быть следующим - [['test 7', 'tea 7'], ['bread 5'], ['milk 2', 'bottom 2', 'coffee 2', 'ill 2']].

Листинг 4: Методы подсчёта средней и максимальной частоты употребления заданного префикса в текстах

```
61    def mean_frequency_of_occurrence(self, prefix) -> float:
62        """Средняя частота встречаемости заданного префикса в текстах"""
63        if type(prefix) != str:
64            return 0.0
65        arr = []
66        for text in self.stat:
67            arr.append(text.get_by_pref(prefix))
68        return sum(arr)/len(arr)
69
70    def max_frequency_of_prefix_occurrence(self, prefix) -> int:
71        """Максимальная частота употребления заданного префикса в текстах"""
72        if type(prefix) != str:
73            return 0
74        max = 0
75        for text in self.stat:
76            if text.get_by_pref(prefix) > max:
77                max = text.get_by_pref(prefix)
78        return max
```

Разработанная структура PrefixStat поддерживает также методы подсчёта средней и максимальной частоты употребления заданного префикса во всех анализируемых текстах (Листинг 4). Данная характеристика используется для анализа тематики и жанра текста, и может служить в дальнейшем для принятия решений о добавлении новых текстов для анализа и генерации.

3 Тестирование разработанного программного модуля и структуры данных

Тестирование является важной и неотъемлемой частью разработки любого программного кода. Для проверки работы разработанного модуля статистик префиксов и модуля-оболочки структуры данных были написаны 19 и 6 юнит-тестов соответственно. При их написании также использовалась методология TDD (Test-driven development). Её основная идея заключается в первоначальном создании методик проверки (тестовых модулей) программного кода и только потом написание исполняемых методов.

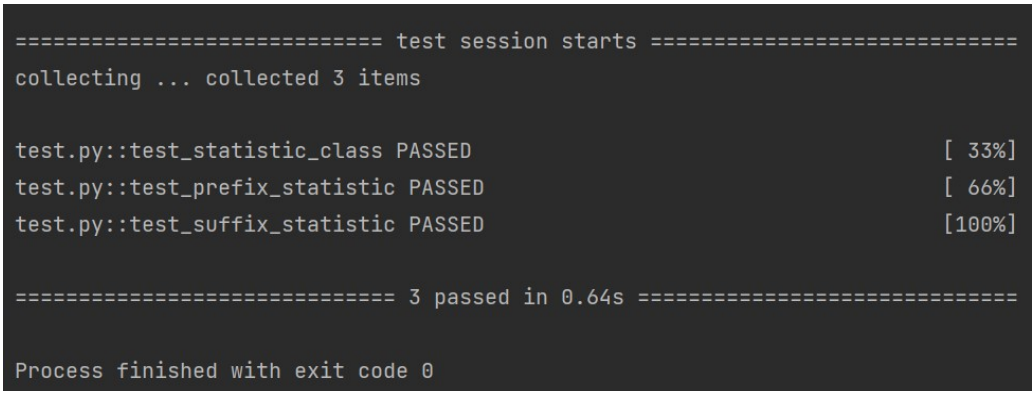

Рис. 1: Тестирование разработанного программного модуля и структуры данных

При тестировании разработанного модуля статистик префиксов и модуля-оболочки структуры данных были подготовлены:

1. Позитивные тесты (8 и 3 соответственно), проверяющие работу методов всех методов класса PrefixStat и модуля-оболочки для хранения префиксов тексте в сценариях, соответствующих их нормальной работе (по одному тесту на каждый метод);

2. Негативные тесты (11 и 3 соответственно), проверяющие корректность работы методов во всех возможных сценариях. В первую очередь проверялась работа методов в случае некорректной передачи аргументов. Например, задание строкового префикса типом None, int или float, или передача в качестве индекса анализируемого текста строки, None, переполненного int (10^{64}), вещественного или отрицательного числа.

Благодаря тестированию были найдены как ошибки в текущих реализация модулей, так и потенциальные. В соответствии с используемой методологией TDD найденные нюансы были учтены, внесены соответствующие правки в программный код (Рис. 1). Созданные тесты оказали положительный эффект на разработку, и будут полезны в дальнейшем при сопровождении и обновлении разработанных модулей.

4 Заключение

В результате производственной практики были достигнуты все поставленные цели и задачи. Разработанный модуль статистики префиксов предоставляет весь необходимый функционал для последующей генерации текстов на его основе. Благодаря созданию собственной C++ структуры данных на основе префиксного дерева (бора) из [1], разработанное решение является одним из оптимальных для алгоритма Маркова.

В ходе решении задач производственной практики были изучены и закреплены навыки подключения собственных структур данных на C/C++ в Python, создание модулей-обёрток; модульное тестирование методов классов; написание модулей в соответствии с методологией Test-driven development. Созданный в процессе производственной практики программный код, его тесты и документация доступны на GitHub для использования в дальнейших задачах.

Список литературы

1. Керниган, Брайан У., Пайк, Роб. Практика программирования. : Пер. с англ. - М. : ООО "И.Д. Вильямс -288 с.

2. Создание расширения C++ для Python // Microsoft [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/visualstudio/python/working-with-c-cpp-python-in-visual-studio?view=vs-2022 (дата обращения: 09.06.2023).

3. pybind11 — Seamless operability between C++11 and Python // GitHub [Электронный ресурс]. Режим доступа: https://github.com/pybind/pybind11 (дата обращения: 09.06.2023).

4. Use pybind11 for a detailed but simple example // Sorush Khajepor [Электронный ресурс]. Режим доступа: https://iamsorush.com/posts/pybind11-robot (дата обращения: 09.06.2023).

5. Создаём C++ Python расширения с помощью pybind11 // совершенство разработки программного обеспечения [Электронный ресурс]. Режим доступа: https://smyt.ru/blog/sozdaem-s-python-rasshireniya-s-pomshyu-pybind11/ (дата обращения: 09.06.2023).