Петрозаводский государственный университет
Институт математики и информационных технологий
Кафедра Информатики и математического обеспечения

Направление подготовки бакалавриата

09.03.04 Программная инженерия

Профиль направления подготовки бакалавриата

“Системное и прикладное программное обеспечение”

Отчет о прохождении производственной практики

Выполнил:
студент группы 22207

М.В.Базаров _____
подпись

Направление подготовки:
Программная инженерия

Место прохождения практики:
Кафедра информатики и математического обеспечения

Период прохождения практики:
29.05.23-11.06.23

Руководитель:
О.Ю. Богоявленская, к.т.н., доцент

_____
подпись

Итоговая оценка

_____
оценка

Петрозаводск — 2023

Содержание

Введение

Цель практики:

- закрепление и углубление теоретических знаний, полученных обучающимися в процессе обучения;
- приобретение навыков и опыта практической работы по реализации и поддержке жизненного цикла программных систем: управлению процессами разработки требований, оценки рисков, проектирования, конструирования, тестирования, сопровождения программных систем, контролю за ходом реализации программных проектов, стратегическому планированию развития программных систем, оценке эффективности профессиональных коммуникаций внутри предприятия или организации.

Задачи учебной практики: описание целей, задач, объекта и предмета практики.

План прохождения практики.
(описание мероприятий, в которых планируется участие, в том числе консультаций с научным руководителем, участие в научном семинаре и т.д.). Здесь также можно дать в виде перечисления: разработка n-ой главы с указание ее названия.

Методы, способы достижения поставленных целей и задач. (Например: изучение и анализ литературы, разработка математичских моделей, проведения экспериментов, анализ их результатов).

Матрица переходов (transition matrix) является основным компонентом модели цепи Маркова. В контексте генерации текста, она представляет собой структуру данных, которая отображает переходы между последовательностями слов в текстовом корпусе.

Матрица переходов имеет два уровня ключей: первый уровень ключей соответствует текущей последовательности слов (окну), а второй уровень ключей соответствует следующему слову после этой последовательности. Значения в матрице переходов обычно представляют собой частоты наблюдаемых переходов от одной последовательности к другой.

Во время обучения модели цепи Маркова на текстовом корпусе, алгоритм проходит по каждому слову и строит матрицу переходов, учитывая текущую последовательность

слов и следующее слово. Если текущая последовательность ещё не присутствует в матрице переходов, она добавляется в качестве ключа первого уровня. Затем проверяется, существует ли уже переход к следующему слову из данной последовательности. Если нет, то создаётся ключ второго уровня, и значение устанавливается на 1. Если такой переход уже существует, то значение в матрице переходов увеличивается на 1.

После обучения модели на текстовом корпусе, матрица переходов содержит информацию о частотах наблюдаемых переходов между последовательностями слов. Эта информация используется при генерации нового текста, чтобы выбирать следующее слово на основе вероятностей, определенных в матрице переходов.

1 Различие обычной программы и модифицированной

Основное отличие между подходами с использованием матрицы переходов и статистики префиксов заключается в способе представления и использования информации о переходах между словами.

1. Матрица переходов: В этом подходе используется двумерная матрица, где строки представляют текущие состояния (префиксы), а столбцы - следующие состояния (следующие слова). Значения в матрице переходов могут быть абсолютными частотами наблюдаемых переходов или вероятностями переходов.

Преимущества: - Хорошо подходит для моделирования случайных процессов с дискретными состояниями. - Позволяет явно выразить вероятности переходов между состояниями. - Удобно использовать для генерации текста, выбирая следующее слово на основе вероятностей в матрице переходов.

Недостатки: - Занимает больше памяти, поскольку требует хранения матрицы размером ‘N x N‘, где ‘N‘ - количество уникальных состояний. - Неэффективен, если количество уникальных состояний велико. - Требует предварительного обучения на корпусе текста для построения матрицы переходов.

2. Статистика префиксов: В этом подходе используется словарь, где ключами являются префиксы (например, кортежи слов), а значениями - суффиксы (следующие слова). Для каждого префикса хранится список возможных суффиксов.

Преимущества: - Требует меньше памяти, поскольку хранит только информацию о конкретных переходах между словами. - Более эффективен для больших корпусов текста, так как не требует создания и хранения большой матрицы переходов. - Не требует предварительного обучения на корпусе текста, так как информация о переходах форми-

руется непосредственно при обучении.

Недостатки: - Не является явным выражением вероятностей переходов между состояниями. - Необходимо использовать дополнительные методы (например, случайный выбор) для выбора следующего слова на основе статистики префиксов.

Оба подхода имеют свои преимущества и недостатки, и выбор подхода зависит от конкретной задачи и требований к модели. Матрица переходов обычно используется для моделирования более сложных процессов, требующих явного выражения вероятностей переходов. Статистика префиксов является более легковесным подходом, который может быть эффективным для генерации текста на основе больших корпусов.

Листинг 1 – Шаблон первой программы

```
import random
```

Импортирование модуля random для работы с генерацией случайных чисел.

Листинг 2 – Шаблон первой программы

```
class MarkovChain:
    def __init__(self, window_size):
        self.window_size = window_size
        self.transition_matrix = {}
```

Определение класса MarkovChain для работы с цепями Маркова. В методе init класса инициализируются атрибуты window size(размер окна) и transition matrix(матрица переходов).

Листинг 3 – Шаблон первой программы

```
    def train(self, corpus):
        words = corpus.split()

        for i in range(len(words) - self.window_size):
            window = tuple(words[i:i+self.window_size])
            next_word = words[i+self.window_size]

            if window not in self.transition_matrix:
                self.transition_matrix[window] = {}
```

```
            if next_word not in self.transition_matrix[window]:
                self.transition_matrix[window][next_word] = 0

            self.transition_matrix[window][next_word] += 1
```

Метод train используется для обучения модели на корпусе текста. Входной параметр corpus представляет собой текстовый корпус. Затем текст разбивается на отдельные слова. Далее происходит проход по всем словам и формирование матрицы переходов на основе наблюдаемых переходов между последовательностями слов.

Листинг 4 – Шаблон первой программы

```
    def generate_text(self, length):
        current_window = random.choice(list(self.transition_matrix.keys()))
        generated_text = list(current_window)

        while len(generated_text) < length:
            if current_window not in self.transition_matrix:
                break

            next_word = random.choices(
                list(self.transition_matrix[current_window].keys()),
                list(self.transition_matrix[current_window].values())
            )[0]

            generated_text.append(next_word)
            current_window = tuple(generated_text[-self.window_size:])

        if generated_text and self.is_preposition(generated_text[0]):
            return self.generate_text(length)

        return ' '.join(generated_text)
```

Метод generate text используется для генерации нового текста на основе обученной модели. Входной параметр length определяет желаемую длину сгенерированного текста. Начальное окно выбирается случайным образом из имеющихся ключей матрицы переходов.

Затем последовательно выбираются следующие слова на основе вероятностей в матрице переходов. Если сгенерированный текст начинается с предлога, вызывается рекурсивно метод generate text для генерации нового текста.

Листинг 5 – Шаблон первой программы

```
    def is_preposition(self, word):
        prepositions = [


        ]
        return word.lower() in prepositions
```

Метод is preposition используется для проверки, является ли слово предлогом. В данном примере представлен простой список русских предлогов. Метод возвращает True, если слово присутствует в списке предлогов (с учётом регистра), и False в противном случае.

Листинг 6 – Шаблон первой программы

```
corpus = "I like to eat bananas. Bananas are tasty. "
chain = MarkovChain(window_size=2)
chain.train(corpus)

generated_text = chain.generate_text(length=10)
print(generated_text)
```

Пример использования созданного класса. Создается экземпляр класса MarkovChain с указанием размера окна (в данном случае 2). Затем корпус текста corpus используется для обучения модели. В конечном итоге генерируется текст длиной 10 слов и выводится на экран.

Заключение

Была создана и модифицирована программа для генерации текста на основе Цепей Маркова

Список литературы

1. Котельников, И. А. LaTeX по-русски [Электронный ресурс]. / И. А. Котельников, П. З. Чеботаев. — 3-е изд., перераб. и доп. — Новосибирск : Сибирский хронограф, 2004. — 496 с. — URL: https://www.researchgate.net/publication/235255954_LaTeX_po-russki

Приложения (при наличии)