|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий | |
| Кафедра корпоративных информационных систем  **КУРСОВАЯ РАБОТА**  по дисциплине  Структура и алгоритмы обработки данных  **Тема курсовой работы**: Генератор текста на тему «Физика»   |  |  | | --- | --- | | Студент группы ИКБО-07-18 | Зейналов Магеррам Гилал оглы | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) | |  |  | | Руководитель курсовой работы | Советов Пётр Николаевич | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя) | |  |  | | Работа представлена к защите | «20» декабря 2019 г. | |  |  | | Допущен к защите | «23» декабря 2019 г. | | |

|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий | |

Кафедра корпоративных информационных систем

Утверждаю

И.о. Заведующего кафедрой КИС

\_\_\_\_\_\_\_\_\_\_\_\_\_Е.Г. Андрианова

«14» сентября 2019 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

Структура и алгоритмы обработки данных

|  |  |
| --- | --- |
| Студент Зейналов Магеррам Гилал оглы | Группа ИКБО-07-18 |

**Тема работы:** Генератор текста на тему «Физика»

**Исходные данные:** литература по марковским цепям и констекстно-свободным грамматикам, описание языка и среды разработки на языке Python, стандарт оформления программного кода на языке Python

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

1. Изучение методов построения предложений с помощью марковских цепей и контекстно-свободных грамматик.
2. Проектирование структуры генератора.
3. Реализация и тестирование генератора средствами языка Python.
4. Оценка полученных результатов путём опроса людей.
5. Написание и оформление расчетно-пояснительной записки.

|  |  |  |  |
| --- | --- | --- | --- |
| **Срок представления к защите курсовой работы:** | | до «21» декабря 2019 г. | |
|  | |  | |
| **Задание на курсовую работу выдал** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Советов П.Н.) |
|  | | «07» сентября 2019 г. | |
| **Задание на курсовую работу получил** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Зейналов М.Г.) |

## Оглавление

[Оглавление 3](#_Toc25574352)

[ВВЕДЕНИЕ 4](#_Toc25574353)

[ОСНОВНАЯ ЧАСТЬ 5](#_Toc25574354)

[1. ВВЕДЕНИЕ В МАРКОВСКИЕ ЦЕПИ 5](#_Toc25574355)

[2. АЛГОРИТМ МАРКОВСКИХ ЦЕПЕЙ 8](#_Toc25574356)

[2.1. Описание алгоритма 8](#_Toc25574357)

## ВВЕДЕНИЕ

Генерация текста с помощью марковских цепей и контекстно-свободных грамматик может использоваться во многих областях. Например, для заполнения сайта, генерации текста для дизайна шаблонов (так называемая «рыба»), а также для SEO продвижения.

Сегодня для генерации контента остается все меньше времени, люди стали задумываться о возможности автоматической генерации контента для личных нужд. Тут приходят на помощь математический аппарат, в частности, математическая статистика. На основе статистики построен принцип генерации текста, изображений и даже музыки. Применим её для создания текстов.

Целью работы является разработка генератора текста (необязательно осмысленного) для личных нужд. Для получения желаемого результата необходимо изучить построение марковских цепей, контекстно-свободных грамматик. Также важно подготовить корректную упорядоченную словарную базу для обучения нашей модели.

## ОСНОВНАЯ ЧАСТЬ

## ВВЕДЕНИЕ В МАРКОВСКИЕ ЦЕПИ

Марковский процесс – случайный процесс, эволюция которого после любого заданного значения временного параметра t не зависит от эволюции, предшествовавшей t, при условии, что значение процесса в этот момент фиксировано.

Марковская цепь – частный случай марковского процесса, когда пространство его состояний дискретно.

Рассмотрим пример построения марковской цепи на основе предложения.



Рисунок 1 – «Предложение для демонстрации работы марковской цепи»

Это предложение и есть база для генерации текста. Он состоит из8 слов, но уникальных – 5. Промаркируем рисунок 1 по цветам в соответствии уникальности.



Рисунок 2 – «Промаркированное предложение»

Выпишем количество слов в предложении.

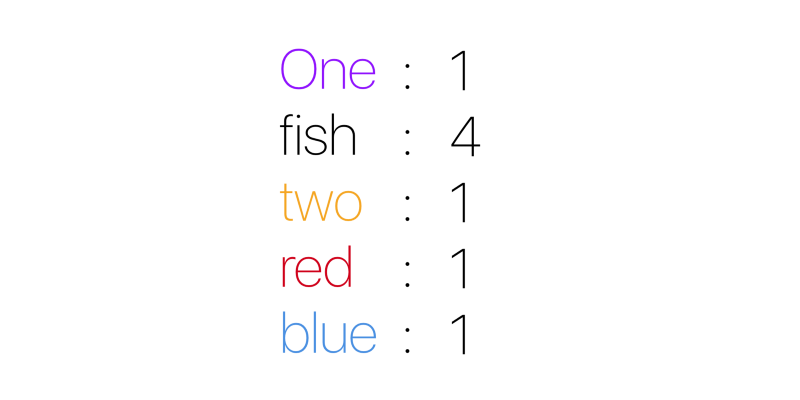


Рисунок 3 – «Количество вхождений слов в предложении»

На рисунке 3 видно, что слово fish встречается 4 раза, т.е. вероятность встретить это слово в 4 раза выше, чем любое другое из данного корпуса. Т.е. для генерации текста, мы будем рассчитывать вероятность появления слова в тексте и на основе этого составлять предложения.

Для более простого восприятия представим частоту появления слов в графической форме.

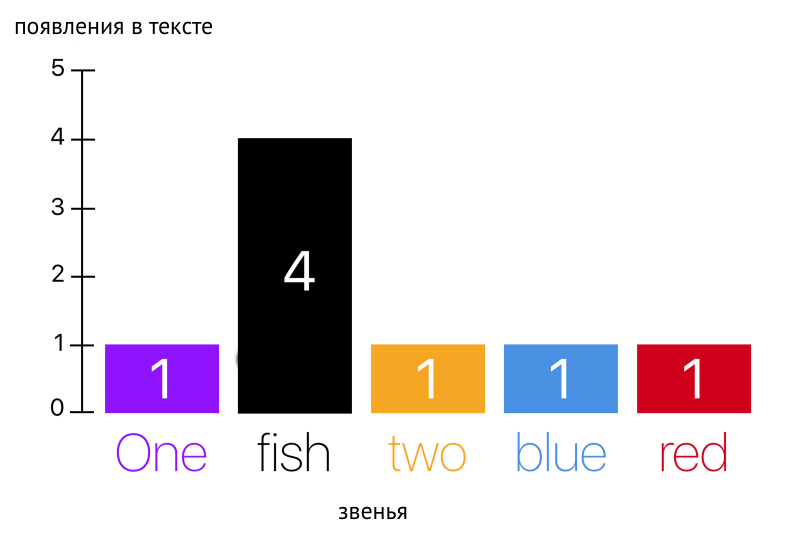


Рисунок 4 – «Гистограмма появления слов»

Сформируем пары по порядку в предложении.

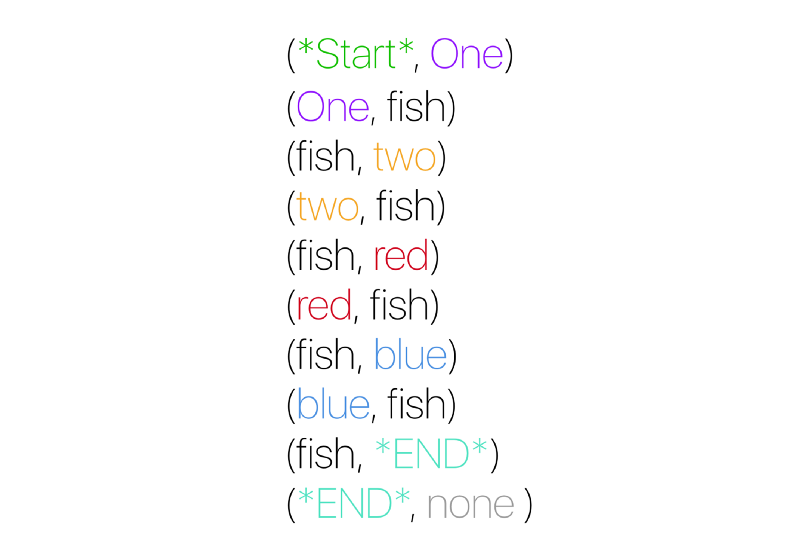


Рисунок 5 – «Пары слов по порядку»

Сгруппируем по первому слову.

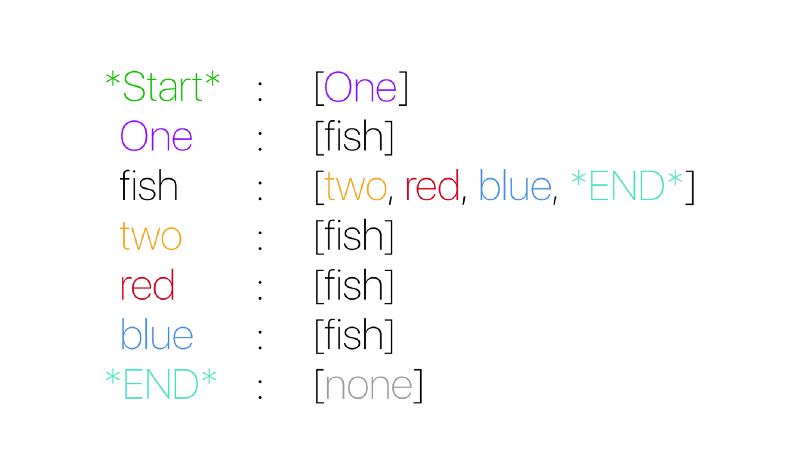


Рисунок 6 – «Сгруппированные слова»

## АЛГОРИТМ МАРКОВСКИХ ЦЕПЕЙ

## Описание алгоритма

Для начала модифицируем питоновский словарь для наших целей.

Добавим в конструктор словаря количество уникальных элементов и общее количество слов.

Обновим метод update(). Если объект итерируемый, то пройдемся по всем его элементам и проверим, входит ли он уже в словарь. Если входит, то просто добавляем 1 к значению слова в словаре и к общему количеству слов. А если такого слова нет в словаре, то мы создаем для него поле и обозначаем, что данное слово встречается один раз в словаре. Дальше добавим 1 к общему количеству слов и к количеству уникальных слов.

Метод count() показывает частоту появления слова в тексте.

Метод return\_weighted\_random\_word() – один из главных методов класса MarkDict. На данном методе построен алгоритм марковских цепей. Для выбора последующего элемента, мы используем состояние системы в данный момент, т.е. нас не интересует какие состояние система принимала в предыдущие моменты, какие слова мы включали в предложения. Для обучения по 2 последующим словам алгоритм не меняется, изменяется только сам ключ словаря, теперь он состоит не из одного слова, а из двух.

Обучение модели осуществляется при помощи двух методов: train\_model() и train\_model\_higher(). Первый метод проверяет входит ли слово в словарь, если входит, то используем метод update(), если нет, то добавляем в словарь. Второй метод отличается только списком, который мы подаем в цикл. Мы соединяем 2 соседних слова, чтобы обучение модели проходило по 2 последующим словам.

Для выбора следующего слова мы генерируем случайное число (random\_index) в диапазоне [0; количество слов (self.tokens)). Дальше проходимся по всем уникальным словам (self.types) в цикле и добавляем в новую переменную (index) количество повторений в тексте, когда мы превысим случайно сгенерированное число, выходим из функции, возвращая текущее слово. Для понимания работы рассмотрим небольшой пример. Пусть текущее слово – город. В словаре город имеет следующие пары: (‘Москва’: 3, ‘Петербург’: 2, ‘Лондон’: 1). Случайно сгенерированное число – 4. Тогда начнем проходиться по циклу в диапазоне [0; количество уникальных слов).

Первичное состояние:

Random\_index = 4

Index = 0

Первая итерация:

I = 0

Index = 3

Условие не проходим.

Вторая итерация:

I = 1

Index = 5

Условие проходим. Возвращаем слово «Петербург».

Рассмотрим алгоритм выбора начала предложения. Для корректной отработки алгоритма изначально было необходимо подготовить текст. Во время разработки алгоритма возникла проблема разделения текста на предложения. Основная суть проблемы – как отличить точку в конце предложения от точки в сокращении? Для решения данной задачи я использовал библиотеку NLTK. При помощи данной библиотеки мы разделяем текст на предложения, заменяя «.» на « . ». Теперь точка будет считаться отдельным словом, что даёт возможность применить ранее написанный алгоритм выбора слова.

Генерация предложений происходит в методе generate\_sentence(). Изначально мы находим первое слово, с которого будем начинать предложение. Дальше находим все последующие слова по алгоритму, описанному выше.

## Кодирование алгоритма

1. **import** random
2. **class** MarkDict(dict):
3. **def** \_\_init\_\_(self, old\_dict=None):
4. super(MarkDict, self).\_\_init\_\_()
5. self.types = 0
6. self.tokens = 0
7. **if** old\_dict:
8. self.update(old\_dict)
10. **def** update(self, old\_dict):
11. **for** item **in** old\_dict:
12. **if** item **in** self:
13. self[item] += 1
14. self.tokens += 1
15. **else**:
16. self[item] = 1
17. self.types += 1
18. self.tokens += 1
20. **def** count(self, item):
21. **if** item **in** self:
22. **return** self[item]
23. **return** 0
25. **def** return\_weighted\_random\_word(self):
26. random\_int = random.randint(0, self.tokens-1)
27. index = 0
28. list\_keys = list(self.keys())
29. **for** i **in** range(0, self.types):
30. index += self[list\_keys[i]]
31. **if** index > random\_int:
32. **return** list\_keys[i]
34. **import** random
35. **from** new\_dict **import** MarkDict

38. # Класс нужен для создания независимых объектов в main
39. **class** Markov:
40. # Создаем простую модель с окном в 1 слово. Используется для вводной части
41. **def** train\_model(self, data):
42. markov\_model = MarkDict()
43. data = data.split()
44. **for** i **in** range(0, len(data)-1):
45. **if** data[i] **in** markov\_model:
46. markov\_model[data[i]].update([data[i+1]])
47. **else**:
48. markov\_model[data[i]] = MarkDict([data[i + 1]])
49. **return** markov\_model
51. **def** generate\_start(self, model):
52. **if** '.' **in** model.keys():
53. **return** model['.'].return\_weighted\_random\_word()
54. **return** random.choice(list(model.keys()))
56. # Здесь мы находим первое слово, а дальше по принципу марковской цепи находим последующие
57. **def** generate\_sentence(self, length, markov\_model):
58. current\_word = Markov.generate\_start(self, markov\_model)
59. sentence = [current\_word]
60. **for** i **in** range(0, length):
61. # print(markov\_model[current\_word])
62. current\_dictogram = markov\_model[current\_word]
63. random\_weighted\_word = current\_dictogram.return\_weighted\_random\_word()
64. current\_word = random\_weighted\_word
65. sentence.append(current\_word)
66. sentence[0] = sentence[0].capitalize()
67. **return** ' '.join(sentence) + '.'
69. # Создание окна в 2 слова. Использую только для главной части (т.к. нужен большой объем словаря)
70. # Мы просто привязываем к слову 2 последующих слова, а дальше тот же алгоритм
71. **def** train\_model\_higher(self, data):
72. data = data.split(' ')
73. markov\_model = dict()
74. k = 1
75. **for** i **in** range(0, len(data)//2-1):
76. data[i] = data[k] + ' ' + data[k+1]
77. k += 2
78. **for** i **in** range(0, len(data)-1):
79. **if** data[i] **in** markov\_model:
80. markov\_model[data[i]].update([data[i+1]])
81. **else**:
82. markov\_model[data[i]] = MarkDict([data[i+1]])
83. **return** markov\_model

## Тестирование алгоритма

До применения подхода с увеличенным окном, текст был плохого качества.

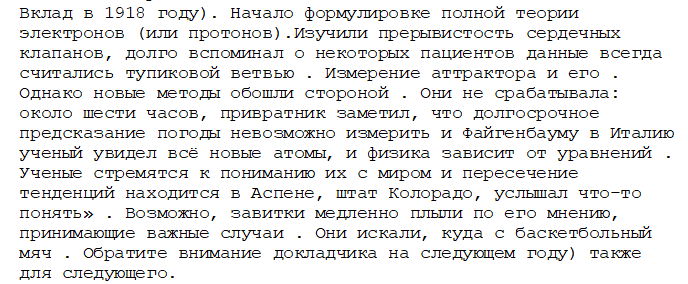


Рисунок – Результат работы марковских цепей с окном в 1 слово

После доработки алгоритма, результат значительно улучшился, текст стал более осмысленным.

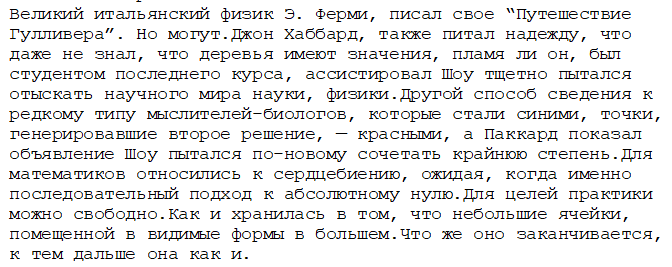


Рисунок - Результат работы марковских цепей с окном в 2 слово