Implementation of Text Generation using Markov Chains in Python

Ahmad Farhan AlShammari
Department of Computer and Information Systems
College of Business Studies, PAAET
Kuwait

ABSTRACT
The goal of this research is to implement text generation using Markov chains in Python. Text generation is the process of creating a new text by analyzing the input text and then predicting the new words. It is used to make posts, reviews, reports, stories, poems, summaries, etc. Markov chains is a mathematical method used to predict the next state based on the current state. The text is generated by randomly selecting the new words based on their probabilities (or weights).

The basic steps of text generation using Markov chains are explained: reading file, cleaning text, creating words, creating chains, computing frequency, computing transition probability, generating text, and printing generated text.

The developed program was tested on an experimental text. The program has successfully performed the basic steps of text generation using Markov chains and provided the required results.

Keywords
Computer Science, Artificial Intelligence, Machine Learning, Natural Language Processing, NLP, Text Generation, Markov Chains, Python, Programming.

1. INTRODUCTION
In the recent years, machine learning has played a major role in the development of computer systems. Machine learning (ML) is a branch of Artificial Intelligence (AI) which is focused on developing algorithms and methods to improve the performance and efficiency of computer programs [1-10].

Text generation is a fundamental area in the field of machine learning. The knowledge body is shared with many other fields like: programming, numerical methods, mathematics, statistics, and natural language processing [11-14, 15-18].

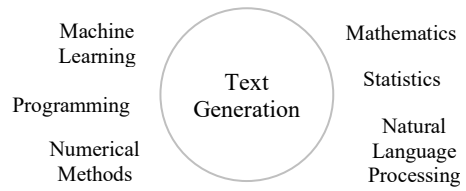

Fig 1: Area of Text Generation

Text generation is the process of producing a new text from a given text. It is performed using Markov chains to predict the new word based on the current word. Text generation is widely used in many applications, for example: text analysis, summarization, creative writing, chatbots, etc.

2. LITERATURE REVIEW
The literature was reviewed to explore the fundamental concepts, methods, and applications of text generation using Markov chains [19-28, 29-35].

The first practices of text generation programs started in the 1960s, for example: ELIZA at MIT [36]. Over time, researchers developed more models using different methods like: Markov chains and neural networks. Now, with the big advances in Natural Language Processing (NLP), machine learning, and deep learning, new models have emerged. A famous example is ChatGPT by OpenAI [37].

Text generation is used to produce a new text automatically from the input text. The given text is analyzed to train the text generation model, then the model predicts the new words.

Markov chains is a mathematical method developed by the Russian mathematician Andrey Markov [38]. It is used to predict the next state based on the current state. It has a wide range of applications in different fields like: mathematics, statistics, finance, economics, computing, networks, communications, etc.

The fundamental concepts of text generation using Markov chains are explained in the following section.

Text Generation:
Text generation is the process of generating a new text from the original text. First, the original text is cleaned from the unwanted characters (end of lines, punctuations, digits, and multiple spaces). Then, the text is split into words. After that, the frequency of words and their next words is computed. Next, the transition probability of words and their next words is computed. The text is generated by randomly selecting the words based on their probabilities (or weights).

The concept of text generation is illustrated by the following diagram:

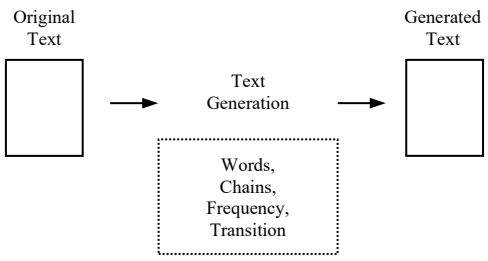

Fig 2: Concept of Text Generation

Markov Chains:

Markov chains is a mathematical method used to predict the next state based on the current state. The Markov chains model consists of states and transitions. It can be represented by a transition diagram.

Transition Diagram:

The transition diagram shows the states and transitions between them. The state is drawn as a circle and the transition is drawn as an arrow. For example, the following diagram shows the transitions between the states (*A*, *B*, and *C*):

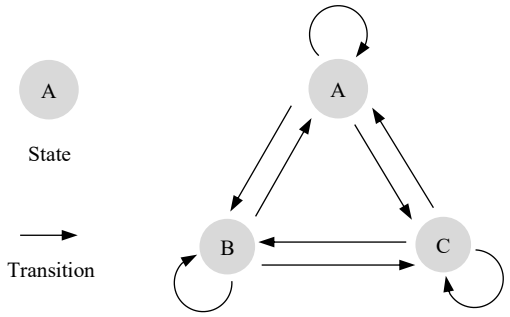

Fig 3: Representation of Transition Diagram

Transition Matrix:

Transition matrix is a matrix of size ($n \times n$) that shows the states and probabilities of transition between them. It can be represented by the following form:

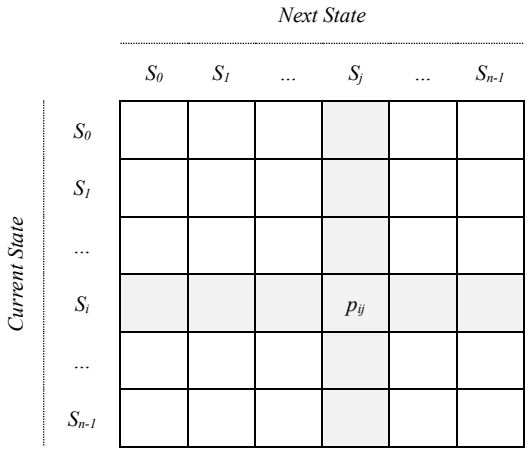

Fig 4: Representation of Transition Matrix

The rows represent the current states, and the columns represent the next states. The cells represent the intersections of rows and columns. For example, the cell (p_{ij}) represents the probability of transition from the current state (S_i) to the next state (S_j). The transition probability can be represented by the following formula:

$$p_{ij} = P(X_{n+1} = S_j \mid X_n = S_i) \qquad (1)$$

where: (X) is the random variable, and (*S*) is the state.

Example:

Assume a given state space = {*A*, *B*, *C*}, with probabilities of transition as shown in the following diagram:

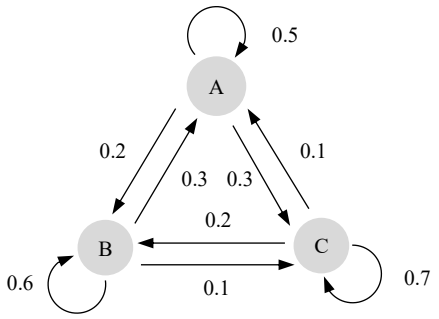

Fig 5: Example of Transition Diagram

Then, the transition matrix of the given example is shown by the following matrix:

Current State \ Next State	*A*	*B*	*C*
A	0.5	0.2	0.3
B	0.3	0.6	0.1
C	0.1	0.2	0.7

Fig 6: Example of Transition Matrix

Implementation Steps and Algorithm:

The implementation of text generation using Markov chains is explained as shown in the following steps:

Step (1): Start with an initial word:
current word = …

Step (2): Get the next words of the current word:
next words = Transition[current word]

Step (3): Randomly, select a word from the next words:
next word = random_choices(next words, weights)

Step (4): Make the next word as the current word:
current word = next word

Step (5): Repeat steps (2–4).

Here, the algorithm of text generation using Markov chains is illustrated as shown in the following view:

Algorithm 1: Text Generation using Markov Chains

```
# Transition matrix
Transition = {w0 : {…},
              w1 : {…},
              … ,
              wn-1 : {…}}
# Initialize current word
current_word = …
# Initialize generated text
text = current_word
for i = 1 to n do
    next_words = Transition[current_word]
    next_word = random_choices(next_words, weights)
```

```
        text = text + " " + next_word
        current_word = next_word
    end for
    text = text + "."
    print(text)
```

Model Order:

The model of text generation using Markov chains can be generalized to order (1, 2, 3, …). The model order is the number of words in the current state. For example: if (order=1) then the current state is one word, if (order=2) then the current state is two words, if (order=3) then the current state is three words, and so on.

Order=1	Order=2	Order=3
(A) → B	*(A, B) → C*	*(A, B, C) → D*
(B) → C	*(B, C) → D*	*(B, C, D) → E*
(C) → D	*(C, D) → E*	*(C, D, E) → F*

Fig 7: Order of Model

Text Generation System:

The text generation system is summarized by the following outline:

Input: Original Text.
Output: Generated Text.
Processing: First, the original text is read from the source file and cleaned from the unwanted characters. Then, the text is split into words. After that, the chains of words and their next words is created. Next, the frequency of words and their next words is computed. Also, the transition probability of words and their next words is computed. Finally, the text is generated by randomly selecting the new words based on their probabilities (or weights).

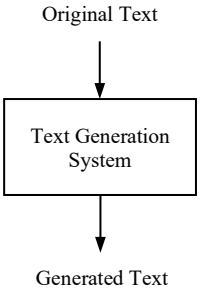

Fig 8: Text Generation System

Python:

Python [39] is an open source, object-oriented, and general-purpose programming language. It is simple, easy to learn, and powerful. It is the most popular programming language especially for the development of machine learning applications.

Python provides many additional libraries for different purposes. For example: Numpy [40], Pandas [41], Matplotlib [42], Seaborn [43], NLTK [44], SciPy [45], and SK Learn [46].

3. RESEARCH METHODOLOGY

The basic steps of text generation using Markov chains are: (1) reading file, (2) cleaning text, (3) creating words, (4) creating chains, (5) computing frequency, (6) computing transition probability, (7) generating text, and (8) printing generated text.

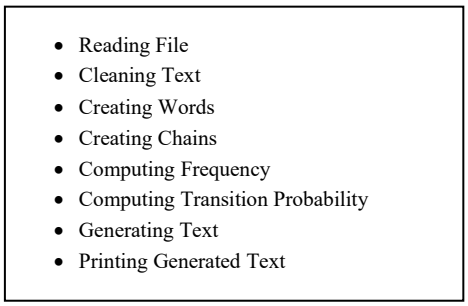

Fig 9: Basic Steps of Text Generation

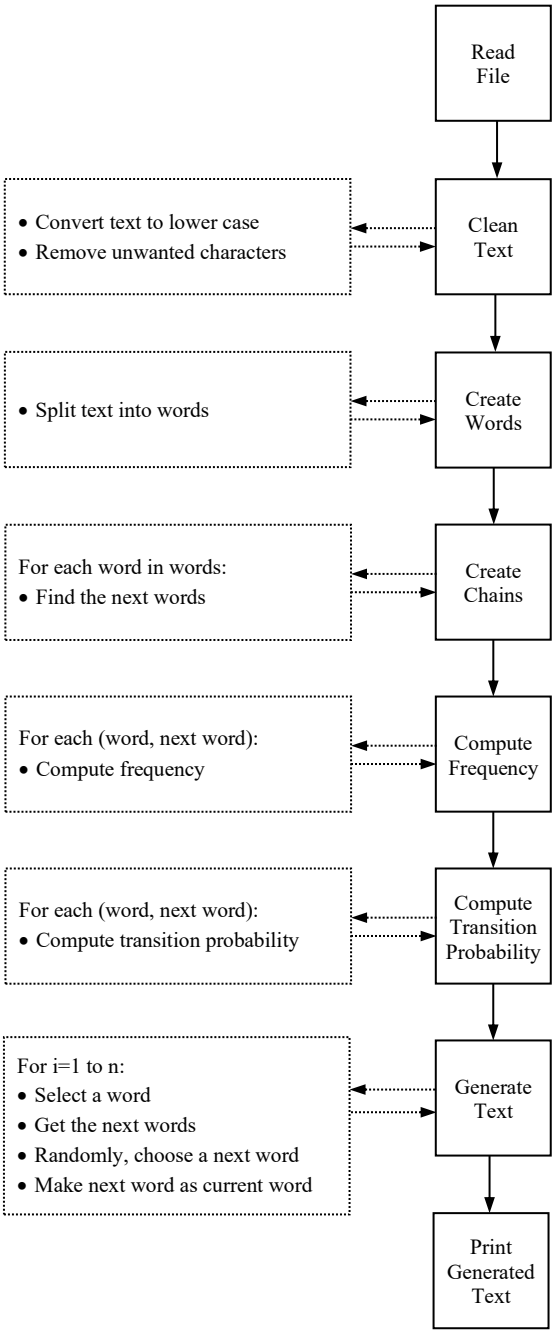

Fig 10: Flowchart of Text Generation

The basic steps of text generation using Markov chains are explained in the following section.

Note: The program is developed using the standard functions of Python without any additional library.

1. Reading File:

The source file (.txt) is read and the original text is loaded by the following code:

```
def read_file():
    file = open("...","r")
    text = file.read()
    file.close()
    return text
```

2. Cleaning Text:

The original text is converted to lower case and cleaned from the unwanted characters (end of lines, punctuations, digits, and multiple spaces) by the following code:

```
def clean_text(text):
    # convert to lower case
    text = text.lower()
    letters = "abcdefghijklmnopqrstuvwxyz"
    # remove unwanted characters
    for c in text:
        if (c not in letters):
            text = text.replace(c, " ")
    # remove multiple spaces
    text = " ".join(text.split())
    return text
```

3. Creating Words:

The list of all words in the text is created by the following code:

```
def create_words(text):
    words = text.split()
    return words
```

4. Creating Chains:

The chains of words and their next words is created by the following code:

```
def compute_chains(words, order):
    chains = {}
    for i in range(len(words)-order):
        key = tuple(words[i:i+order])
        next_word = words[i+order]
        if (key not in chains):
            chains[key] = []
        chains[key].append(next_word)
    # sort chains
    chains = dict(sorted(chains.items()))
    return chains
```

5. Computing Frequency:

The frequency of words and their next words is computed by the following code:

```
def compute_freq(chains):
    freq = {}
    for key, next_words in chains.items():
        freq[key] = {}
        for word in next_words:
            if (word not in freq[key]):
                freq[key][word] = 0
            freq[key][word] += 1
    return freq
```

6. Computing Transition Probability:

The transition probability of words and their next words is computed by the following code:

```
def compute_trans(freq):
    trans = {}
    for key, next_words in freq.items():
        trans[key] = {}
        total = sum(next_words.values())
        for word, count in next_words.items():
            trans[key][word] = count/total
    return trans
```

7. Generating Text:

The text is generated by randomly selecting the words and their next words based on their probabilities (weights). It is done by the following code:

```
import random as rm

def generate_text(trans, n):
    key = rm.choice(list(trans.keys()))
    words = list(key)
    for i in range(n):
        options = list(trans[key].keys())
        weights = list(trans[key].values())
        next_word = rm.choices(options,
                               weights)
        words.append(next_word[0])
        key = tuple(list(key)[1:] + next_word)
        if (key not in list(trans.keys())):
            break
    gen_text = " ".join(words)
    gen_text += "."
    return gen_text
```

8. Printing Generated Text:

The generated text is created and printed by the following code:

```
print("Generated Text:")
print("---------------")
for i in range(10):
    gen_text = generate_text(trans, 10)
    print(gen_text)
```

4. RESULTS AND DISCUSSION

The developed program was tested on an experimental text. The program has successfully performed the basic steps of text generation using Markov chains and provided the required results. The program output is explained in the following section.

Reading File:

The source file (.txt) is read and the original text is loaded to the program.

Cleaning Text:

The original text is converted to lower case, cleaned from the unwanted characters, and displayed as shown in the following view:

```
Cleaned Text:
-------------
once upon a time in a faraway land there lived
a clever crow one hot summer day the crow was
flying in search of water the sun was blazing
```

```
and the ground was dried the crow had been
searching for hours and was increasingly
thirsty and tired as the crow flew over a
village it noticed a pot near a house the crow
flew down and stood on the edge of the pot
hoping to find water inside the crow saw some
water inside the pot however the water level
...
```

Creating Words:

The list of words is created and displayed as shown in the following view:

```
List of Words:
--------------
['once', 'upon', 'a', 'time', 'in', 'a',
'faraway', 'land', 'there', 'lived', 'a',
'clever', 'crow', 'one', 'hot', 'summer',
'day', 'the', 'crow', 'was', 'flying', 'in',
'search', 'of', 'water', 'the', 'sun', 'was',
'blazing', 'and', 'the', 'ground', 'was',
'dried', 'the', 'crow', 'had', 'been',
'searching', 'for', 'hours', 'and', 'was',
'increasingly', 'thirsty', 'and', 'tired',
'as', 'the', 'crow', 'flew', 'over', 'a',
...
```

Creating Chains:

The chains of words and their next words (order=2) is created and displayed as shown in the following view:

```
Chains of Words:
----------------
('a', 'clever'),       crow
('a', 'faraway'),      land
('a', 'house'),        the
('a', 'pot'),          near
('a', 'stone'),        with
('a', 'time'),         in
('a', 'village'),      it
('about', 'how'),      to
('after', 'another'),  as
('an', 'idea'),        the
...
```

Computing Frequency:

The frequency of words and their next words is computed and displayed as shown in the following view:

```
Frequency:
----------
('a', 'clever'),       crow         :  1
('a', 'faraway'),      land         :  1
('a', 'house'),        the          :  1
('a', 'pot'),          near         :  1
('a', 'stone'),        with         :  1
('a', 'time'),         in           :  1
('a', 'village'),      it           :  1
('about', 'how'),      to           :  1
('after', 'another'),  as           :  1
('an', 'idea'),        the          :  1
...
```

Computing Transition Probability:

The transition probability of words and their next words is computed and displayed as shown in the following view:

```
Transition Probability:
-----------------------
('a', 'clever'),       crow         : 1.00
('a', 'faraway'),      land         : 1.00
('a', 'house'),        the          : 1.00
('a', 'pot'),          near         : 1.00
('a', 'stone'),        with         : 1.00
('a', 'time'),         in           : 1.00
('a', 'village'),      it           : 1.00
('about', 'how'),      to           : 1.00
('after', 'another'),  as           : 1.00
('an', 'idea'),        the          : 1.00
...
```

Generating Text:

The generated text is created and displayed as shown in the following view:

```
Generated Text:
---------------
crow sat on the edge of the pot the crow flew
over. been searching for hours and was
increasingly thirsty and tired as the. more
stones were dropped the water with its beak but
it still. a time in a faraway land there lived
a clever crow one. some water inside the pot
the crow tried to reach the water. thirsty and
tired as the crow sat on the edge of the. crow
began looking around for small stones it picked
up a stone. and tired as the crow continued to
drop one stone after another. upon a time in a
faraway land there lived a clever crow. crow
one hot summer day the crow flew away proud of
...
```

In summary, it is clear that the program has successfully performed the basic steps of text generation using Markov chains and provided the required results.

5. CONCLUSION

In this research, the goal was to implement text generation using Markov chains in Python. The literature was reviewed to understand the fundamental concepts of text generation using Markov chains: text generation, Markov chains, transition diagram, transition matrix, implementation steps, algorithm, and model order.

The author developed a program in Python to perform the basic steps of text generation using Markov chains: reading file, cleaning text, creating words, creating chains, computing frequency, computing transition probability, generating text, and printing generated text.

The developed program was tested on an experimental text. The program has successfully performed the basic steps of text generation using Markov chains and provided the required results.

In the future, more work is needed to improve the current methods of text generation using Markov chains. In addition, they should be more investigated on other texts from different fields and domains.

6. REFERENCES

[1] Sammut, C., & Webb, G. I. (2011). "Encyclopedia of Machine Learning". Springer.

[2] Jung, A. (2022). "Machine Learning: The Basics". Springer.

[3] Kubat, M. (2021). "An Introduction to Machine Learning". Springer.

[4] Li, H. (2023). "Machine Learning Methods". Springer.

[5] Zollanvari, A. (2023). " Machine Learning with Python". Springer.

[6] Chopra, D., & Khurana, R. (2023). "Introduction to Machine Learning with Python". Bentham Science Publishers.

[7] Müller, A. C., & Guido, S. (2016). "Introduction to Machine Learning with Python: A Guide for Data Scientists". O'Reilly Media.

[8] Raschka, S. (2015). "Python Machine Learning". Packt Publishing.

[9] Forsyth, D. (2019). "Applied Machine Learning". Springer.

[10] Sarkar, D., Bali, R., & Sharma, T. (2018). "Practical Machine Learning with Python". Apress.

[11] Igual, L., & Seguí, S. (2017). "Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications". Springer.

[12] VanderPlas, J. (2017). "Python Data Science Handbook: Essential Tools for Working with Data". O'Reilly Media.

[13] Muddana, A., & Vinayakam, S. (2024). "Python for Data Science". Springer.

[14] Unpingco, J. (2022). "Python for Probability, Statistics, and Machine Learning". Springer.

[15] Zelle, J. (2017). "Python Programming: An Introduction to Computer Science". Franklin, Beedle & Associates.

[16] Chun, W. (2001). "Core Python Programming". Prentice Hall Professional.

[17] Padmanabhan, T. (2016). "Programming with Python". Springer.

[18] Beazley, D., & Jones, B. K. (2013). "Python Cookbook: Recipes for Mastering Python 3". O'Reilly Media.

[19] Lee, R. (2023). "Natural Language Processing: A Textbook with Python Implementation". Springer.

[20] Bird, S., Klein, E., & Loper, E. (2009). "Natural Language Processing with Python". O'Reilly Media.

[21] Chopra, D., Joshi, N., & Mathur, I. (2016). "Mastering Natural Language Processing with Python". Packt Publishing.

[22] Thanaki, J. (2017). "Python Natural Language Processing". Packt Publishing.

[23] Arumugam. R., Shanmugamani, R. (2018). " Hands-On Natural Language Processing with Python". Packt Publishing.

[24] Ghosh, S., & Gunning, D. (2019). "Natural Language Processing Fundamentals". Packt Publishing.

[25] Kedia, A., & Rasu, M. (2020). " Hands-On Python Natural Language Processing". Packt Publishing.

[26] Reiter, E. (2025). "Natural Language Generation". Springer.

[27] Indurkhya, N., & Damerau, F. (2010). "Handbook of Natural Language Processing". CRC Press.

[28] Clark, A., Fox, C., & Lappin, S. (2013). "The Handbook of Computational Linguistics and Natural Language Processing". John Wiley & Sons.

[29] Norris, J. (2009). "Markov Chains". Cambridge University Press.

[30] Tolver, A. (2016). "An Introduction to Markov Chains". Department of Mathematical Sciences, University of Copenhagen.

[31] Weber, R. (2011). "Markov Chains". Department of Pure Mathematics and Mathematical Statistics. University of Cambridge.

[32] Gagniuc, P. (2017). "Markov Chains: From Theory to Implementation and Experimentation". John Wiley & Sons.

[33] Ching, W., Huang, S., Ng, M., & Siu, T. (2013). "Markov Chains: Models, Algorithms, and Applications". Springer.

[34] Privault, N. (2018). "Understanding Markov Chains: Examples and Applications". Springer.

[35] Ankan, A., & Panda, A. (2018). "Hands-On Markov Models with Python". Packt Publishing.

[36] ELIZA: https://elizagen.org

[37] ChatGPT: https://chatgpt.com

[38] Grinstead, C., & Snell, J. (1997). "Introduction to Probability". American Mathematical Society. pp. 464–466.

[39] Python: http://www.python.org

[40] Numpy: http://www.numpy.org

[41] Pandas: http://pandas.pydata.org

[42] Matplotlib: http://www. matplotlib.org

[43] Seaborn: http://seaborn.pydata.org

[44] NLTK: http://www.nltk.org

[45] SciPy: http://scipy.org

[46] SK Learn: http://scikit-learn.org