

Explainable identification of hidden language patterns across documents  
using NLP and statistical techniques

by

Akhil Joshi

B. Tech., Jawaharlal Nehru Technological University, India, 2020

---

A THESIS

submitted in partial fulfillment of the  
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science  
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

2024

Approved by:

Major Professor  
Dr. Lior Shamir

# Copyright

© Akhil Joshi 2024.

# Abstract

In the field of Natural Language Processing (NLP), extracting detailed linguistic patterns and domain-specific insights is crucial for analyzing complex text data. Existing tools and large language models (LLM) like ChatGPT, often perform the task to partial degree of success, and does not provide explainable insights on the nature of the similarities they identify. The primary goal is to create a versatile framework capable of conducting comprehensive analyses across diverse fields, including biographies, historical events, scientific literature and even when dealing with unfamiliar content. This research addresses this gap by developing an automated framework that uses n-gram analysis combined with NLP libraries and a custom formula developed through multiple iterations, which calculates a weighted score based on n-gram frequency differences and the presence of key linguistic features which allowed us to prioritize meaningful n-grams. By focusing on recurring linguistic patterns, the framework uncovers subtle conceptual relationships between texts that are not immediately obvious. Visualization tools like word clouds further enhance the representation of key terms and patterns. By surpassing the limitations of current AI technologies and other tools, our framework empowers researchers and analysts to explore complex textual data with greater depth and precision, offering a scalable and reliable solution for analyzing large datasets in various domains.

# Table of Contents

|                                                                                           |      |
|-------------------------------------------------------------------------------------------|------|
| List of Figures . . . . .                                                                 | viii |
| List of Tables . . . . .                                                                  | ix   |
| List of Nomenclature . . . . .                                                            | xi   |
| Acknowledgments . . . . .                                                                 | xii  |
| Dedication . . . . .                                                                      | xiii |
| Introduction . . . . .                                                                    | xiv  |
| Related Work . . . . .                                                                    | xvi  |
| 1 A Deep Dive into N-Gram and NLP Techniques, Tools and Applications . . . . .            | 1    |
| 1.1 Overview . . . . .                                                                    | 1    |
| 1.2 N-Gram Analysis Implementation . . . . .                                              | 2    |
| 1.2.1 Understanding N-grams . . . . .                                                     | 2    |
| 1.2.2 Significance and Applications of N-grams . . . . .                                  | 3    |
| 1.3 Natural Language Processing with Various Libraries . . . . .                          | 4    |
| 1.3.1 Prominent NLP Libraries in Python . . . . .                                         | 4    |
| 1.3.2 Comparison of NLP Libraries: SpaCy, NLTK, Gensim, and Stanford<br>CoreNLP . . . . . | 6    |
| 1.3.3 Rationale for Selecting SpaCy and NLTK . . . . .                                    | 6    |
| 1.4 Features of SpaCy . . . . .                                                           | 8    |
| 1.4.1 Tokenization . . . . .                                                              | 8    |

|       |                                                                                        |    |
|-------|----------------------------------------------------------------------------------------|----|
| 1.4.2 | Named Entity Recognition (NER)                                                         | 8  |
| 1.4.3 | Lemmatization                                                                          | 9  |
| 1.4.4 | Dependency Parsing                                                                     | 9  |
| 1.5   | Features of NLTK                                                                       | 10 |
| 1.5.1 | N-Gram Analysis                                                                        | 10 |
| 1.5.2 | Text Preprocessing                                                                     | 11 |
| 1.6   | Code-Level Implementation                                                              | 11 |
| 1.6.1 | Iteration 1: NLTK-Based Implementation                                                 | 12 |
| 1.6.2 | Iteration 2: Switching to SpaCy                                                        | 13 |
| 1.6.3 | Iteration 3: Combining SpaCy and NLTK for N-Gram Analysis                              | 14 |
| 1.6.4 | Iteration 4: Introducing Named Entity Recognition and Noun Chunk<br>Extraction         | 16 |
| 1.6.5 | Final Iteration: Combining All Features for Comprehensive Text Anal-<br>ysis           | 17 |
| 1.7   | Conclusion                                                                             | 19 |
| 2     | Building a Robust N-Gram Comparison Framework: Iterations and Final Solution           | 20 |
| 2.1   | Introduction                                                                           | 20 |
| 2.2   | Function Overview                                                                      | 21 |
| 2.3   | Initial Approach: Simple N-Gram Frequency Comparison                                   | 22 |
| 2.4   | Improving Noise Reduction with Logarithmic Scaling                                     | 23 |
| 2.5   | Dynamic Thresholding Based on Frequency                                                | 25 |
| 2.6   | Incorporating Average Frequency and Advanced Weighting                                 | 26 |
| 2.7   | Final Approach: A Comprehensive Method with Named Entity and Noun<br>Chunk Integration | 28 |
| 2.7.1 | Combining SpaCy and NLTK for Comprehensive Analysis                                    | 28 |
| 2.7.2 | Data Input and N-Gram Extraction                                                       | 29 |
| 2.7.3 | Looping Through N-Grams and Comparing Frequencies                                      | 30 |

|       |                                                                                      |    |
|-------|--------------------------------------------------------------------------------------|----|
| 2.7.4 | Named Entity and Noun Chunk Presence . . . . .                                       | 31 |
| 2.7.5 | Filtering by Part-of-Speech (POS) Tags . . . . .                                     | 31 |
| 2.7.6 | Calculating N-Gram Weight . . . . .                                                  | 32 |
| 2.7.7 | Storing the Results . . . . .                                                        | 33 |
| 2.8   | Conclusion . . . . .                                                                 | 34 |
| 3     | Comprehensive Data Analysis: Collection, Visualization, and Comparative Insights     | 35 |
| 3.1   | Overview . . . . .                                                                   | 35 |
| 3.2   | Data Scraping Methodology . . . . .                                                  | 36 |
| 3.2.1 | Pseudocode for Fetching Wikipedia Article . . . . .                                  | 36 |
| 3.2.2 | Example Use Case . . . . .                                                           | 37 |
| 3.3   | Geniuses of the Cosmos: A Comparative Study of Hawking and Einstein . .              | 38 |
| 3.3.1 | Unigram Analysis: Key Scientific Concepts for Hawking and Einstein                   | 39 |
| 3.3.2 | Bigram Analysis: Notable Scientific Contributions . . . . .                          | 40 |
| 3.3.3 | Trigram Analysis: Deeper Insights into Theoretical Physics . . . . .                 | 41 |
| 3.3.4 | Comparing ChatGPT's Summary to N-gram Insights: Hawking and<br>Einstein . . . . .    | 41 |
| 3.4   | Rivalry on the Field: A Comparative Study of Ronaldo and Messi . . . . .             | 43 |
| 3.4.1 | Unigram Analysis: Key Football Themes for Ronaldo and Messi . . .                    | 44 |
| 3.4.2 | Bigram Analysis: Significant Phrases in Their Football Careers . . . .               | 45 |
| 3.4.3 | Trigram Analysis: Deeper Insights into Football Successes . . . . .                  | 46 |
| 3.4.4 | Comparing ChatGPT's Summary to N-gram Insights: Messi and Ronaldo                    | 47 |
| 3.5   | Pop Icons: A Comparative Study of Michael Jackson and Taylor Swift . . .             | 50 |
| 3.5.1 | Data Collection: Wikipedia Articles on Michael Jackson and Taylor<br>Swift . . . . . | 50 |
| 3.5.2 | Unigram Analysis: Key Music Themes for Michael Jackson and Taylor<br>Swift . . . . . | 50 |
| 3.5.3 | Bigram Analysis: Significant Phrases in Their Careers . . . . .                      | 51 |

|        |                                                                                                     |    |
|--------|-----------------------------------------------------------------------------------------------------|----|
| 3.5.4  | Trigram Analysis: Deeper Insights into Musical Legacies . . . . .                                   | 52 |
| 3.5.5  | Comparing ChatGPT's Summary to N-gram Insights: Michael Jack-<br>son and Taylor Swift . . . . .     | 53 |
| 3.6    | From Science to Business: A Cross-Domain Exploration of Albert Einstein<br>and Jeff Bezos . . . . . | 56 |
| 3.6.1  | Unigram Analysis: Key Themes for Einstein and Bezos . . . . .                                       | 56 |
| 3.6.2  | Bigram Analysis: Significant Phrases in Their Lives . . . . .                                       | 57 |
| 3.6.3  | Trigram Analysis: Deeper Insights into Public Life . . . . .                                        | 58 |
| 3.6.4  | Comparing ChatGPT's Summary to N-gram Insights: Einstein and<br>Bezos . . . . .                     | 59 |
| 3.7    | Conclusion: ChatGPT vs. N-gram Analysis . . . . .                                                   | 62 |
| 3.8    | Final Conclusion . . . . .                                                                          | 63 |
| 3.8.1  | Key Insights . . . . .                                                                              | 63 |
| 3.8.2  | Strengths of N-gram Analysis . . . . .                                                              | 64 |
| 3.8.3  | Final Thoughts . . . . .                                                                            | 65 |
| 3.9    | Implementation Challenges and Frequently Asked Questions . . . . .                                  | 65 |
| 3.10   | GitHub Repository . . . . .                                                                         | 67 |
| 3.10.1 | Installation Instructions . . . . .                                                                 | 67 |
| 3.10.2 | Running the Tool . . . . .                                                                          | 68 |
| 3.10.3 | Documentation . . . . .                                                                             | 68 |
|        | Bibliography . . . . .                                                                              | 70 |

# List of Figures

|      |                                                                        |    |
|------|------------------------------------------------------------------------|----|
| 1.1  | Distribution of N-grams . . . . .                                      | 3  |
| 3.1  | Unigram distribution for Stephen Hawking and Albert Einstein . . . . . | 38 |
| 3.2  | Bigram distribution for Stephen Hawking and Albert Einstein . . . . .  | 39 |
| 3.3  | Trigram distribution for Stephen Hawking and Albert Einstein . . . . . | 40 |
| 3.4  | Unigram distribution for Cristiano Ronaldo and Lionel Messi . . . . .  | 44 |
| 3.5  | Bigram distribution for Cristiano Ronaldo and Lionel Messi . . . . .   | 45 |
| 3.6  | Trigram distribution for Cristiano Ronaldo and Lionel Messi . . . . .  | 46 |
| 3.7  | Unigram distribution for Michael Jackson and Taylor Swift . . . . .    | 51 |
| 3.8  | Bigram distribution for Michael Jackson and Taylor Swift . . . . .     | 52 |
| 3.9  | Trigram distribution for Michael Jackson and Taylor Swift . . . . .    | 53 |
| 3.10 | Unigram distribution for Albert Einstein and Jeff Bezos . . . . .      | 57 |
| 3.11 | Bigram distribution for Albert Einstein and Jeff Bezos . . . . .       | 58 |
| 3.12 | Trigram distribution for Albert Einstein and Jeff Bezos . . . . .      | 59 |

# List of Tables

|     |                                                                        |    |
|-----|------------------------------------------------------------------------|----|
| 1.1 | Comparison of NLP Libraries: SpaCy, NLTK, Gensim, and Stanford CoreNLP | 6  |
| 1.2 | Tokenization Example . . . . .                                         | 8  |
| 1.3 | NER Example . . . . .                                                  | 8  |
| 1.4 | Lemmatization Example . . . . .                                        | 9  |
| 1.5 | Dependency Parsing Example . . . . .                                   | 9  |
| 1.6 | Unigrams Example . . . . .                                             | 10 |
| 1.7 | Bigrams Example . . . . .                                              | 10 |
| 1.8 | Original Words Example . . . . .                                       | 11 |
| 1.9 | Processed Words Example . . . . .                                      | 11 |
| 3.1 | Comparison Between ChatGPT and N-gram Analysis . . . . .               | 62 |

# Listings

|      |                                                                         |    |
|------|-------------------------------------------------------------------------|----|
| 1.1  | Iteration 1 - NLTK-Based Implementation . . . . .                       | 12 |
| 1.2  | Iteration 2 - SpaCy-Based Implementation . . . . .                      | 13 |
| 1.3  | Iteration 3 - Combining SpaCy and NLTK for N-Gram Analysis . . . . .    | 15 |
| 1.4  | Iteration 4 - Adding Named Entity Recognition and Noun Chunks . . . . . | 16 |
| 1.5  | Final Code - Combining All Features . . . . .                           | 17 |
| 2.1  | Iteration 1 - Simple N-Gram Frequency Comparison . . . . .              | 22 |
| 2.2  | Iteration 2 - Logarithmic Scaling for Noise Reduction . . . . .         | 23 |
| 2.3  | Iteration 3 - Dynamic Thresholding Based on Frequency . . . . .         | 25 |
| 2.4  | Iteration 4 - Incorporating Average Frequency . . . . .                 | 27 |
| 2.5  | Final Approach - Named Entity and Noun Chunk Integration . . . . .      | 28 |
| 2.6  | Final Approach - Unpacking Document Data for Analysis . . . . .         | 29 |
| 2.7  | Final Approach - Comparing N-Grams Between Documents . . . . .          | 30 |
| 2.8  | Final Approach - Detecting Named Entities and Noun Chunks . . . . .     | 31 |
| 2.9  | Final Approach - Filtering by POS Tags . . . . .                        | 32 |
| 2.10 | Final Approach - Calculating the N-Gram Weight . . . . .                | 32 |
| 2.11 | Final Approach - Storing N-Gram Results . . . . .                       | 33 |
| 3.1  | Ensure the Necessary Data Files and paths . . . . .                     | 68 |

# Nomenclature

## NLP Techniques

- **SpaCy** – An open-source software library for advanced natural language processing.
- **NLTK** – Natural Language Toolkit, a suite of libraries and programs for symbolic and statistical natural language processing.

## N-Grams

- **1-Gram** – A single word or token.
- **2-Gram** – A sequence of two contiguous words or tokens.
- **3-Gram** – A sequence of three contiguous words or tokens.

## Normalization Techniques

- **Square Root Transformation** – A technique used to stabilize variance and normalize data.

## Visualization Tools

- **Word Cloud** – A visual representation of the most frequent terms in a dataset, where word size indicates frequency.

## Statistical Metrics

- **Frequency Discrepancy** – The difference in occurrence rates of n-grams between documents.
- **Weighted Score** – A score calculated based on the frequency and contextual relevance of n-grams.

# Acknowledgments

First and foremost, I would like to express my deepest gratitude to my academic advisor and major professor, Dr. Lior Shamir. His unwavering support, insightful guidance, and belief in my abilities have been crucial throughout my research journey. The numerous courses I undertook under his mentorship, each presenting a unique learning curve, significantly shaped my understanding and skills. Dr. Shamir's ideas and constant feedback were instrumental in the completion of this thesis, and I am profoundly thankful for his enduring encouragement and expertise.

I am deeply thankful to my committee members, Dr. Torben Amtoft and Dr. Mitchell L. Neilsen, for their invaluable support throughout this thesis. From Dr. Amtoft, I gained a strong sense of organization and discipline, which shaped the way I approached and structured my research. His clear and methodical style helped me tackle complex topics with precision. Dr. Neilsen, on the other hand, taught me the importance of patience and persistence through his hands-on Real-Time Systems course. His approach encouraged me to remain focused and thoughtful when addressing the challenging computational and statistical aspects of this work.

I extend my heartfelt thanks to all those who have supported me throughout this journey. Their collective wisdom, encouragement, and support have been indispensable, and I am deeply appreciative of their involvement in shaping this thesis.

# Dedication

This thesis is dedicated to my beloved family, whose unwavering support and encouragement have been the foundation of my academic journey. To my parents, Raghottam Joshi and Parimala Joshi, thank you for instilling in me the values of hard work and perseverance. To my sister, Aishwarya Joshi, and my brother-in-law, Karthik Vijayasarithi, your belief in my potential and relentless support have been my guiding light through the challenges and triumphs of this endeavor.

I would also like to dedicate this work to my friends, Chandana Reddy and Sai Teja Erukude, whose friendship has made this journey enjoyable and fulfilling. Your support has been invaluable.

# Introduction

In today’s digital age, where language and data are inseparable, efficiently processing and interpreting large volumes of text has become critical. Natural Language Processing (NLP), which merges computer science with linguistics, enables the analysis of vast text corpora to uncover valuable insights<sup>1</sup>. Despite advances in NLP, significant challenges remain in identifying nuanced language patterns and domain-specific structures<sup>23</sup>. This thesis seeks to address these challenges by applying n-gram analysis alongside modern NLP tools to explore hidden linguistic relationships and gain a deeper understanding of language use across different contexts<sup>4</sup>.

This research is driven by the need for more sophisticated text analysis frameworks that move beyond surface-level processing. While AI models and text processing systems can generate general overviews, they often fail to capture the finer linguistic distinctions necessary for domain-specific tasks such as comparing historical texts, scientific literature, or public discourse<sup>56</sup>. This gap highlights the importance of a more refined approach that not only identifies common linguistic features but also uncovers the unique patterns that distinguish one corpus from another<sup>7</sup>.

Building on prior work, this study integrates n-gram analysis with advanced NLP techniques, including named entity recognition (NER) and part-of-speech (POS) filtering, using modern tools like SpaCy and NLTK<sup>48</sup>. The custom algorithm developed through this research adjusts for frequency discrepancies and emphasizes key linguistic patterns across corpora. This framework enables more meaningful comparisons between datasets, enhancing the capacity for large-scale text analysis.

The thesis is structured in three sections. The first outlines the tools and techniques used for n-gram extraction and NLP processing, establishing the foundation for the research. The second section details the development and refinement of the n-gram comparison framework, which serves as the backbone of the study. The final section presents the results, using visual tools like word clouds and statistical listings to highlight the most significant linguistic trends and insights revealed by the analysis.

In conclusion, this study demonstrates the strength of comparative n-gram analysis as a robust tool for text analysis, contributing to computational linguistics and paving the way for future research<sup>47</sup>.

# Related Work

N-grams have been a foundational technique in Natural Language Processing (NLP) for decades, particularly in tasks like language modeling, machine translation, and speech recognition. Early work showed how n-grams capture local linguistic context, providing a probabilistic framework that improves prediction accuracy<sup>12</sup>. These techniques enhance tasks where predicting a word based on preceding words is critical, significantly improving translation and recognition systems. N-grams continue to be valuable because of their simplicity and efficiency in representing text sequences.

In recent years, the integration of n-grams with machine learning models has further broadened their applications. N-grams have been incorporated into models like Skip-gram, improving word sequence representation in vector spaces<sup>3</sup>. They have also been applied in text categorization and topic modeling, where they help identify co-occurring terms and enhance document classification<sup>56</sup>. Furthermore, the ability of n-grams to model syntactic patterns has made them crucial for understanding text in a more granular way, benefiting tasks like text summarization and information retrieval. These developments highlight n-grams' ongoing relevance in feature extraction and document analysis.

Despite these advancements, there is a need for comprehensive studies comparing n-gram performance across various datasets using tools like SpaCy and NLTK. Prior research often focused on specific languages or domains, while modern tools can handle multilingual and multi-genre corpora. Additionally, the flexibility of modern NLP libraries allows for deeper integration of n-grams with advanced machine learning models. By combining traditional methods with advanced techniques like named entity recognition and syntactic parsing, this research explores shared linguistic features, contributing to computational linguistics<sup>487</sup>.

# Chapter 1

## A Deep Dive into N-Gram and NLP Techniques, Tools and Applications

### 1.1 Overview

This chapter delves into the methodologies and processes used in our project, with a focus on N-gram analysis and advanced Natural Language Processing (NLP) techniques. N-gram analysis, which involves examining contiguous sequences of words or characters, is pivotal to uncovering patterns and contextual relationships within text. By analyzing these sequences, we can gain insight into linguistic structures and enhance various NLP applications such as language modeling, sentiment analysis, and predictive text<sup>71</sup>. Additionally, we examine key Python libraries, including SpaCy, NLTK, Gensim, and Stanford CoreNLP, each offering unique capabilities for text analysis<sup>8439</sup>. The goal is to illustrate how these tools and techniques enhance text analysis and contribute to the development of robust NLP applications.

## 1.2 N-Gram Analysis Implementation

N-gram analysis is a foundational technique in NLP used to capture the contextual structure of a language by analyzing contiguous sequences of 'n' items (typically words or characters) from a given text<sup>1</sup>.

### 1.2.1 Understanding N-grams

N-grams are contiguous sequences of 'n' items, typically words in the context of NLP. These items can be characters, words, or even syllables, depending on the granularity desired<sup>1</sup>. The value of 'n' determines the order of the N-gram.

- **Unigrams (1-grams):** Single words, e.g., "cat," "dog."
- **Bigrams (2-grams):** Pairs of consecutive words, e.g., "natural language," "deep learning."
- **Trigrams (3-grams):** Triplets of consecutive words, e.g., "machine learning model," "data science approach."
- **4-grams, 5-grams, etc.:** Sequences of four, five, or more consecutive words.

The figure above (Fig. 1.1) illustrates the distribution of various n-grams in the dataset. This analysis is crucial for understanding the underlying linguistic structures and is instrumental in tasks such as text classification and sentiment analysis<sup>8</sup>.

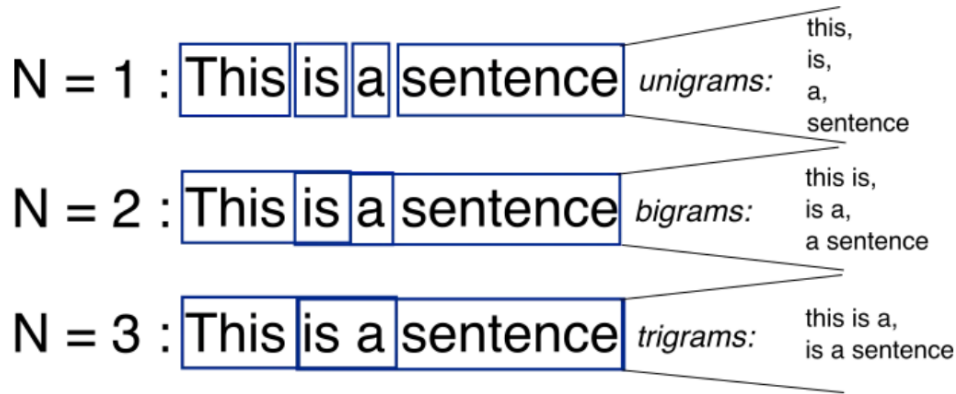


**Figure 1.1:** The figure shows the distribution of 1-grams, 2-grams, and 3-grams across the dataset, highlighting key linguistic patterns and frequencies.

### 1.2.2 Significance and Applications of N-grams

N-grams are essential in analyzing and understanding sequences of words, offering several key benefits in text processing:

1. **Capturing Context:** N-grams help capture the context within word sequences, enabling a deeper analysis of linguistic patterns<sup>1</sup>.
2. **Enhancing Text Prediction:** They improve the accuracy of predictive text systems by suggesting subsequent words based on context<sup>7</sup>.
3. **Information Retrieval:** N-grams improve document matching and ranking, ensuring relevant information is retrieved efficiently<sup>5</sup>.
4. **Feature Extraction:** They are crucial for identifying patterns in text classification and sentiment analysis tasks<sup>3</sup>.

Overall, N-grams are crucial for capturing linguistic patterns and improving various text processing tasks, contributing significantly to the success of our project<sup>6</sup>.

## 1.3 Natural Language Processing with Various Libraries

Natural Language Processing (NLP) is a critical subfield of artificial intelligence that enables computers to understand, interpret, and generate human language. By bridging the gap between human communication and machine understanding, NLP powers a wide array of technologies, from chatbots and sentiment analysis tools to voice-activated assistants. Python has become the leading programming language for NLP, largely due to its simplicity, powerful libraries, and a strong community of developers and researchers<sup>7</sup>.

In Python, several prominent NLP libraries are available, each catering to different aspects of natural language processing. In the following sections, we will explore some of the most popular NLP libraries, including Gensim, Stanford CoreNLP, SpaCy, and NLTK.

### 1.3.1 Prominent NLP Libraries in Python

Python offers a diverse set of libraries tailored to various NLP tasks. Below, we explore some of the most widely used libraries, comparing their features and applications.

#### **Gensim**

Gensim is a powerful library designed for topic modeling and document similarity analysis. It excels in identifying themes from large text corpora through techniques such as Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA)<sup>6</sup>. Gensim also provides implementations of word embedding models like word2vec and GloVe, which transform words into dense vectors that capture semantic meanings<sup>3</sup>. Additionally, Gensim is effective in comparing document similarity, making it a valuable tool for tasks like clustering and information retrieval.

## **Stanford CoreNLP**

Stanford CoreNLP, developed by Stanford University, is a comprehensive suite of NLP tools designed for in-depth text analysis. It supports a variety of NLP tasks, including part-of-speech tagging, named entity recognition (NER), and sentiment analysis. CoreNLP is known for its robust performance across multiple languages, making it an excellent choice for linguistically complex tasks<sup>9</sup>.

## **SpaCy**

SpaCy is a more recent NLP library focused on efficiency, scalability, and ease of use. Unlike NLTK, which is primarily used for research and educational purposes, SpaCy is optimized for production environments. It offers state-of-the-art performance in tasks such as tokenization, named entity recognition, and dependency parsing<sup>4</sup>. SpaCy is known for its speed and neural network-based approaches, making it ideal for real-time applications and large-scale text processing<sup>3</sup>.

## **Natural Language Toolkit (NLTK)**

The Natural Language Toolkit (NLTK) is one of the most comprehensive libraries for NLP in Python<sup>8</sup>. It provides tools for tokenization, classification, stemming, tagging, parsing, and semantic reasoning. NLTK is highly favored in academic and research settings, due to its vast collection of text corpora and its support for complex linguistic analysis<sup>8</sup>. Although it is highly flexible, NLTK tends to be slower than newer libraries and may require more customization for production-level applications.

### 1.3.2 Comparison of NLP Libraries: SpaCy, NLTK, Gensim, and Stanford CoreNLP

The table below compares key features of the libraries discussed, highlighting their strengths and weaknesses in various NLP tasks.

**Table 1.1:** *Comparison of NLP Libraries: SpaCy, NLTK, Gensim, and Stanford CoreNLP*

| Feature                         | SpaCy  | NLTK   | Gensim | Stanford CoreNLP           |
|---------------------------------|--------|--------|--------|----------------------------|
| <b>Tokenization</b>             | Yes    | Yes    | Yes    | Yes                        |
| <b>Parts of Speech</b>          | Yes    | Yes    | No     | Yes                        |
| <b>Named Entity Recognizer</b>  | Yes    | Yes    | Yes    | Yes                        |
| <b>Dependency Parse</b>         | Yes    | Yes    | Yes    | Yes                        |
| <b>Text Matcher</b>             | Yes    | No     | No     | No                         |
| <b>Date Matcher</b>             | No     | No     | No     | Yes                        |
| <b>Chunking</b>                 | Yes    | Yes    | No     | Yes                        |
| <b>Spell Checker</b>            | No     | No     | No     | No                         |
| <b>Sentiment Detector</b>       | No     | No     | No     | Yes                        |
| <b>Pretrained Models</b>        | Yes    | Yes    | Yes    | Yes                        |
| <b>Neural Network</b>           | Yes    | No     | No     | No                         |
| <b>Supports training on GPU</b> | Yes    | No     | No     | No                         |
| <b>Language</b>                 | Python | Python | Python | Java (with Python support) |

### 1.3.3 Rationale for Selecting SpaCy and NLTK

In this project, we chose to utilize both SpaCy and NLTK for their respective strengths in natural language processing (NLP) tasks. These libraries complement each other, offering a balance between high-performance, real-time processing, and comprehensive linguistic tools.

SpaCy was chosen for its speed and efficiency in handling large amounts of text. It excels in real-time processing and is built on modern techniques like neural networks, making it perfect for tasks like Named Entity Recognition (NER) and Dependency Parsing. SpaCy's tokenization and lemmatization helped break down sentences and standardize words, which improved the consistency of our analysis. Its dependency parsing further allowed us to understand how words relate grammatically.

On the other hand, NLTK was selected for its versatility and wide range of tools, especially for n-gram analysis and text preprocessing. NLTK's ability to generate unigrams, bigrams, and trigrams helped us explore word patterns and themes in the text. Its preprocessing tools, like stopword removal and stemming, helped clean up the text by filtering out unnecessary elements, making our data more manageable for further analysis.

In summary, SpaCy was ideal for fast, real-time tasks like NER and dependency parsing, while NLTK provided the flexibility we needed for detailed linguistic analysis and preprocessing. Using both libraries allowed us to combine performance with depth, giving us a well-rounded approach to text analysis.

### Advantages of SpaCy

- **High Performance:** SpaCy is designed for speed and efficiency, making it ideal for processing large volumes of text in real-time applications.
- **Modern NLP Techniques:** SpaCy leverages neural networks and word vectors, offering advanced capabilities for tasks like NER and text classification<sup>3</sup>.
- **Multilingual Support:** SpaCy supports over 49 languages, with pre-trained models for each, making it versatile for global applications<sup>4</sup>.

### Advantages of NLTK

- **Versatility:** NLTK provides a broad range of tools, from tokenization to complex syntactic parsing, making it suitable for varied NLP tasks<sup>8</sup>.
- **Extensive Documentation:** NLTK's well-documented features, supported by a large community, are especially useful for beginners and researchers.
- **Third-Party Extensions:** NLTK integrates well with other libraries, expanding its capabilities beyond its core functions.

## 1.4 Features of SpaCy

### 1.4.1 Tokenization

Tokenization is the process of splitting words, punctuation, or other elements into tokens.<sup>4</sup>

**Example:** “SpaCy is an open-source library for Natural Language Processing.”

| Tokens       |
|--------------|
| 'SpaCy'      |
| 'is'         |
| 'an'         |
| 'open'       |
| '_'          |
| 'source'     |
| 'library'    |
| 'for'        |
| 'Natural'    |
| 'Language'   |
| 'Processing' |
| ','          |
| '.'          |

**Table 1.2:** *Tokenization Example*

### 1.4.2 Named Entity Recognition (NER)

Named Entity Recognition (NER) identifies and classifies named entities in text, such as people, organizations, and locations<sup>4</sup>.

**Example:** “Apple is looking to buy a startup in the UK.”

| Entity | Label                     |
|--------|---------------------------|
| Apple  | organization (ORG)        |
| UK     | geopolitical entity (GPE) |

**Table 1.3:** *NER Example*

### 1.4.3 Lemmatization

Lemmatization reduces words to their base or root form, which helps in standardizing words and improving text analysis consistency<sup>4</sup>.

**Example:** “The cats are running and jumping.”

| Lemmas |
|--------|
| 'the'  |
| 'cat'  |
| 'be'   |
| 'run'  |
| 'and'  |
| 'jump' |
| '.'    |

**Table 1.4:** *Lemmatization Example*

### 1.4.4 Dependency Parsing

Dependency parsing analyzes the grammatical structure of sentences, showing how words relate to each other within the sentence<sup>4</sup>.

**Example:** “SpaCy is a library used for NLP.”

| Token   | Dependency                   |
|---------|------------------------------|
| SpaCy   | nominal subject (nsubj)      |
| is      | root                         |
| a       | determiner (det)             |
| library | attribute (attr)             |
| used    | adjectival clause (acl)      |
| for     | preposition (prep)           |
| NLP     | object of preposition (pobj) |
| .       | punctuation (punct)          |

**Table 1.5:** *Dependency Parsing Example*

## 1.5 Features of NLTK

### 1.5.1 N-Gram Analysis

NLTK's n-gram utilities were employed to generate unigrams, bigrams, and trigrams from the text. This analysis is crucial for examining the lexical patterns and thematic structures within the texts. By creating these n-grams, we could analyze the frequency and context of word combinations, providing insights into common themes and stylistic elements<sup>8</sup>.

**Example:** “NLTK is a powerful library for text analysis.”

For this sentence the n-gram analysis might produce:

| Unigrams |
|----------|
| NLTK     |
| is       |
| a        |
| powerful |
| library  |
| for      |
| text     |
| analysis |

**Table 1.6:** *Unigrams Example*

| Bigrams          |
|------------------|
| NLTK is          |
| is a             |
| a powerful       |
| powerful library |
| library for      |
| for text         |
| text analysis    |

**Table 1.7:** *Bigrams Example*

## 1.5.2 Text Preprocessing

NLTK's preprocessing tools were used for tasks such as stopwords removal and stemming. These steps are essential for cleaning the text and reducing noise in the data, thereby enhancing the accuracy of subsequent analysis. For instance, stopwords were removed to focus on meaningful content, and stemming was used to reduce words to their root forms<sup>8</sup>.

**Example:** “The running cats are jumping quickly.”

For this sentence after preprocessing:

| Original Words |
|----------------|
| The            |
| running        |
| cats           |
| are            |
| jumping        |
| quickly        |

**Table 1.8:** *Original Words Example*

| Processed Words |
|-----------------|
| run             |
| cat             |
| jump            |
| quick           |

**Table 1.9:** *Processed Words Example*

## 1.6 Code-Level Implementation

The development of the text analysis pipeline went through multiple iterations, each bringing improvements in terms of efficiency, accuracy, and feature completeness. We will discuss the reasoning behind the changes and demonstrating the outputs generated at each stage. This journey leads to the final, optimized implementation that best meets the requirements.

## 1.6.1 Iteration 1: NLTK-Based Implementation

The initial version of the code was built using NLTK for tokenization, stopword removal, and lemmatization. NLTK is a powerful toolkit for natural language processing, but it is slower and less efficient for handling large datasets, and it lacks some of the advanced features offered by more modern NLP libraries like SpaCy.

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

def preprocess(text):
    tokens = word_tokenize(text.lower())
    lemmas = [lemmatizer.lemmatize(word) for word in tokens if word.isalpha()]
    return lemmas
```

**Listing 1.1:** *Iteration 1 - NLTK-Based Implementation*

### Explanation:

- This version utilizes NLTK's `WordNetLemmatizer` to convert tokens to their base form (lemma).
- The input text is tokenized using `word_tokenize()`, and only alphabetic tokens are selected for further processing.

For the sentence:

*"Apple is looking to buy a startup in the UK."*

The output is:

- **Lemmas:** {'apple', 'is', 'looking', 'to', 'buy', 'a', 'startup', 'in', 'the', 'uk'}

### Limitations:

- This implementation does not remove stopwords (e.g., "is", "to", "the"), which introduces noise into the analysis.
- Advanced NLP features such as named entity recognition (NER) or n-grams are not included.
- Performance was slower, particularly with large datasets, due to the limitations of NLTK.

## 1.6.2 Iteration 2: Switching to SpaCy

In the second iteration, SpaCy was introduced for its faster and more efficient text processing capabilities. SpaCy provides modern tokenization, lemmatization, and linguistic features, making it a better fit for handling larger text datasets.

```
import spacy

nlp = spacy.load("en_core_web_sm")

def preprocess(text):
    doc = nlp(text.lower())
    lemmas = [token.lemma_ for token in doc if token.is_alpha and not token.is_stop]
    return lemmas
```

**Listing 1.2:** *Iteration 2 - SpaCy-Based Implementation*

**Explanation:**

- SpaCy's `en_core_web_sm` model is used to tokenize and lemmatize the text.
- Stopwords are automatically removed using SpaCy's built-in functionality.

The output for the same sentence is:

- **Lemmas:** {'apple', 'look', 'buy', 'startup', 'uk'}

**Improvements:**

- Stopwords are removed, resulting in a cleaner set of tokens.
- Lemmatization is more accurate and faster compared to NLTK.

**Limitations:**

- Despite improvements in tokenization and lemmatization, this version does not include n-gram analysis or named entity recognition, limiting the depth of the text analysis.

### 1.6.3 Iteration 3: Combining SpaCy and NLTK for N-Gram Analysis

To introduce n-gram analysis into the pipeline, I integrated NLTK's `ngrams` utility. This iteration combines SpaCy's modern tokenization and lemmatization with NLTK's n-gram capabilities, enhancing the system's ability to analyze word sequences.

```

import spacy
import nltk
from nltk.util import ngrams

nlp = spacy.load("en_core_web_sm")

def preprocess_and_ngrams(text, n=2):
    doc = nlp(text.lower())
    lemmas = [token.lemma_ for token in doc if token.is_alpha and not token.is_stop]
    n_grams = list(ngrams(lemmas, n))
    return lemmas, n_grams

```

**Listing 1.3:** *Iteration 3 - Combining SpaCy and NLTK for N-Gram Analysis*

### Explanation:

- SpaCy handles tokenization, stopword removal, and lemmatization.
- NLTK's `ngrams` function generates bigrams (or other n-grams) from the lemmatized tokens.

For the same sentence, the output is:

- **Lemmas:** {'apple', 'look', 'buy', 'startup', 'uk'}
- **Bigrams:** {'apple look', 'look buy', 'buy startup'}

**Improvements:** This iteration introduces bigram analysis, which allows for the exploration of word combinations and phrases.

**Limitations:** The system still lacks more advanced features such as noun chunk extraction and named entity recognition.

## 1.6.4 Iteration 4: Introducing Named Entity Recognition and Noun Chunk Extraction

In this iteration, I added two powerful features from SpaCy: named entity recognition (NER) and noun chunk extraction. These features provide deeper insights into the text, enabling the identification of key entities and their syntactic roles.

```
import spacy

nlp = spacy.load("en_core_web_sm")

def process_text_with_ner_and_chunks(text):
    doc = nlp(text)
    lemmas = [token.lemma_ for token in doc if token.is_alpha and not token.is_stop]
    entities = {ent.text: ent.label_ for ent in doc.ents}
    noun_chunks = [chunk.text for chunk in doc.noun_chunks]
    return lemmas, entities, noun_chunks
```

**Listing 1.4:** *Iteration 4 - Adding Named Entity Recognition and Noun Chunks*

### Explanation:

- Named entities (like people, organizations, and locations) are extracted from the text.
- Noun chunks (syntactic units such as noun phrases) are also identified.

The output for the sentence:

- **Lemmas:** 'apple', 'look', 'buy', 'startup', 'uk'
- **Entities:** 'Apple': ORG, 'UK': GPE
- **Noun Chunks:** 'Apple', 'a startup', 'the UK'

### Improvements:

- The system can now recognize entities such as "Apple" (an organization) and "UK" (a geopolitical entity).
- The inclusion of noun chunks provides additional context, revealing syntactic groupings within the text.

### Limitations:

- While this iteration introduced advanced NLP features, it did not yet incorporate n-gram analysis.

## 1.6.5 Final Iteration: Combining All Features for Comprehensive Text Analysis

The final iteration combines all the elements of the previous versions into a single, cohesive pipeline. This version integrates SpaCy's named entity recognition, noun chunk extraction, and lemmatization with NLTK's n-gram analysis. The result is a system capable of performing comprehensive text analysis.

```
import spacy
from nltk.util import ngrams

nlp = spacy.load("en_core_web_lg")

def process_text(nlp, text, min_n, max_n):
    doc = nlp(text)
    ngram_freq = {}
    noun_chunks = set(chunk.text.lower() for chunk in doc.noun_chunks)
```

```

entities = set(ent.text.lower() for ent in doc.ents)

tokens = [token.lemma_ for token in doc if token.is_alpha and not token.is_stop]

# Generate n-grams
for n in range(min_n, max_n + 1):
    for ngram_tuple in ngrams(tokens, n):
        ngram_str = ' '.join(ngram_tuple).lower()
        ngram_freq[ngram_str] = ngram_freq.get(ngram_str, 0) + 1

return ngram_freq, noun_chunks, entities

```

**Listing 1.5:** *Final Code - Combining All Features*

### Explanation:

- This function integrates all aspects of text processing: tokenization, lemmatization, stopword removal, named entity recognition, noun chunk extraction, and n-gram analysis.
- The system now offers a complete and efficient solution for textual analysis.

The output for the sentence:

- **Noun Chunks:** {'apple', 'a startup', 'the uk'}
- **Named Entities:** {'apple' (ORG), 'uk' (GPE)}
- **Unigrams:** {'apple', 'look', 'buy', 'startup', 'uk'}
- **Bigrams:** {'apple look', 'look buy', 'buy startup'}

## 1.7 Conclusion

In this chapter, we explored the application of N-gram analysis and advanced NLP techniques within our project. We highlighted how N-grams capture contextual patterns and enhance various NLP applications, including language modeling, text prediction, information retrieval, and feature extraction. Using Python libraries such as NLTK and SpaCy, we implemented N-gram analysis to uncover linguistic structures and patterns within our text corpora, thereby enhancing our ability to analyze and interpret textual data effectively<sup>847</sup>.

Additionally, we examined other notable NLP libraries, such as Gensim and Stanford CoreNLP, and discussed how they complement NLTK and SpaCy. This comparative analysis guided our tool selection, ensuring that we utilized the most appropriate resources for our project's needs<sup>69</sup>. The integration of these advanced tools allowed us to conduct detailed and efficient text analysis, providing valuable insights and significantly contributing to the project's overall success.

Ultimately, the combination of SpaCy's robust linguistic processing and NLTK's specialized N-gram analysis enabled a comprehensive examination of textual data, leading to a deeper understanding of thematic and lexical patterns<sup>110</sup>. This approach not only enhanced the quality of our analysis, but also underscored the importance of choosing the right tools for effective text processing and interpretation.

# Chapter 2

## Building a Robust N-Gram Comparison Framework: Iterations and Final Solution

### 2.1 Introduction

In the previous chapter, we explored foundational concepts of n-grams and their role in Natural Language Processing (NLP). We examined how n-grams capture word sequences and how techniques like Part-of-Speech (POS) tagging and named entity recognition enhance our understanding of textual data<sup>1;7</sup>. The natural progression is to apply these concepts by comparing n-grams across different documents, revealing patterns, themes, or distinctions<sup>5</sup>.

To achieve this, we developed the function `analyze_common_ngrams`, which identifies and assigns weights to n-grams appearing in both documents. This chapter details the development of this function, explaining how each iteration improved upon the previous one, leading to a more refined approach for comparing textual data.

## 2.2 Function Overview

The `analyze_common_ngrams` function is designed to analyze and assign a weighted score to n-grams that appear in both documents being compared. Below is the function signature and a brief description of its arguments:

---

**Algorithm 1** Analyze Common N-Grams Between Two Documents

---

**Input:**

`doc1_data`, `doc2_data`, `eng_dict`, `nlp`

**Output:**

`weighted_ngrams` (Weighted n-grams that appear in both documents)

Analyze the n-grams in `doc1_data` and `doc2_data`.

Identify common n-grams between the two documents.

For each common n-gram, calculate a weighted score based on frequency using the `eng_dict`.

Apply the SpaCy language model `nlp` for noun chunk and named entity recognition.

Combine the weighted scores of the common n-grams from both documents.

**return** `weighted_ngrams` (a dictionary of common n-grams and their weighted scores)

---

**Arguments:**

- **`doc1_data` and `doc2_data`:** These tuples contain n-gram data, noun chunks, and named entity information for each document. These elements were constructed using the methodologies discussed in Chapter 1.
- **`eng_dict`:** A pre-computed dictionary of English word frequencies, which aids in calculating the significance of each n-gram.
- **`nlp`:** The SpaCy language model used for NLP tasks such as POS tagging and named entity recognition.

## 2.3 Initial Approach: Simple N-Gram Frequency Comparison

In the first iteration, the approach was straightforward, we compared n-gram frequencies between two documents without incorporating any advanced NLP features. The function calculated a weight based on the differences in frequencies of the n-grams in the two documents. However, the formula was overly simplistic and failed to provide insightful results.

```
def analyze_common_words(doc1_dict, doc2_dict, total_words_doc1,
    total_words_doc2, eng_dict):
    res_dict = {}

    smaller_dict, larger_dict = (doc1_dict, doc2_dict) if total_words_doc1 <
total_words_doc2 else (doc2_dict, doc1_dict)

    for word in smaller_dict:
        if word in larger_dict:
            f1 = doc1_dict.get(word, 0) / total_words_doc1
            f2 = doc2_dict.get(word, 0) / total_words_doc2
            if f1 > 0 and f2 > 0:
                fe = eng_dict.get(word, 0) # English frequency

                # Simple weight calculation
                weight = (f1 - fe)**2 * (f2 - fe)**2
                res_dict[word] = weight

    return res_dict
```

**Listing 2.1:** *Iteration 1 - Simple N-Gram Frequency Comparison*

### Explanation:

- The function calculates the relative frequency of an n-gram in each document and compares it to its frequency in the English language corpus (`eng_dict`).
- A simple quadratic difference was used to calculate the weight of the common n-grams.

### Limitations:

- The formula did not take into account advanced linguistic features such as POS tagging or named entities.
- The results lacked insight, as they failed to distinguish between meaningful and non-meaningful n-grams.

For example, comparing two documents with this approach might give equal weight to both "the apple" and "important concept," even though the latter might carry more semantic weight.

## 2.4 Improving Noise Reduction with Logarithmic Scaling

The second iteration attempted to reduce the noise by applying logarithmic scaling to the frequency differences, which helps mitigate the influence of common but unimportant words like "the" or "is."

```
from math import log

def calculateSimilarities(doc1_dict, doc2_dict, total_words_doc1,
                          total_words_doc2, eng_dict):
```

```

res_dict = {}
for word in set(doc1_dict).intersection(set(doc2_dict)):
    f1 = doc1_dict[word] / total_words_doc1
    f2 = doc2_dict[word] / total_words_doc2
    fe = eng_dict.get(word, 1e-9) # Avoid division by zero

    if f1 > 0 and f2 > 0:
        # Logarithmic scaling to reduce noise
        mi = f1 * log(f1 / fe) + f2 * log(f2 / fe)
        res_dict[word] = mi
return res_dict

```

**Listing 2.2:** *Iteration 2 - Logarithmic Scaling for Noise Reduction*

### Explanation:

- The logarithmic scaling reduced the influence of very common words, as the logarithm dampens the effect of large values.
- This iteration aimed to better highlight less frequent but more meaningful n-grams by adjusting the weight calculation.

### Limitations:

- While the logarithmic approach improved the handling of common words, it still did not incorporate any linguistic features such as POS tags or named entities.
- The results were still somewhat noisy, especially when dealing with very similar documents.

## 2.5 Dynamic Thresholding Based on Frequency

To address the noise further, we introduced dynamic thresholding in the third iteration. This approach used the minimum frequency among the two documents and the English dictionary to establish a threshold for determining whether an n-gram was significant.

```
def analyze_common_words(doc1_dict, doc2_dict, total_words_doc1,
    total_words_doc2, eng_dict):
    res_dict = {}
    thresholds = []

    smaller_dict, larger_dict = (doc1_dict, doc2_dict) if total_words_doc1 <
total_words_doc2 else (doc2_dict, doc1_dict)

    for word in smaller_dict:
        if word in larger_dict:
            f1 = doc1_dict.get(word, 0) / total_words_doc1
            f2 = doc2_dict.get(word, 0) / total_words_doc2
            fe = eng_dict.get(word, 0)

            # Calculate the dynamic threshold
            threshold = min(f1, f2, fe)
            thresholds.append(threshold)

            # Weight calculation with dynamic threshold
            weight = (f1 - fe)**2 * (f2 - fe)**2
            if fe > threshold:
                weight = 0

            res_dict[word] = weight

    # Print top thresholds for debugging
    top_50_thresholds = sorted(thresholds, reverse=True)[:50]
```

```
print("Top 50 thresholds:", top_50_thresholds)

return res_dict
```

**Listing 2.3:** *Iteration 3 - Dynamic Thresholding Based on Frequency*

**Explanation:**

- The introduction of a dynamic threshold filtered out less significant n-grams by comparing their frequency to a threshold based on the document frequencies and the English corpus frequency.
- This aimed to improve the accuracy by eliminating n-grams that were too common in both documents.

**Limitations:**

- Despite this improvement, the results were still not as insightful as expected, especially when the documents were very different or very similar.
- The approach lacked any form of linguistic enrichment (e.g., POS tagging or entity recognition), which limited its ability to highlight semantically important n-grams.

## 2.6 Incorporating Average Frequency and Advanced Weighting

In the fourth iteration, we calculated the average frequency of multi-word n-grams by splitting them into individual words and averaging their English frequencies. This allowed for more nuanced weighting of multi-word n-grams, which were not handled well in previous iterations.

```

def analyze_common_words(doc1_freq, doc2_freq, total_words_doc1,
    total_words_doc2, eng_dict):
    res_dict = {}
    for word in set(doc1_freq) | set(doc2_freq):
        if '_' in word:
            parts = word.split('_')
            avg_freq = sum(eng_dict.get(part, 0) for part in parts) / len(
parts)
        else:
            avg_freq = eng_dict.get(word, 0)

        f1 = doc1_freq.get(word, 0) / total_words_doc1
        f2 = doc2_freq.get(word, 0) / total_words_doc2

        if f1 > 0 and f2 > 0 and avg_freq > 0:
            # Weight calculation based on average frequency
            weight = ((f1 + f2) / 2 - avg_freq) ** 2 / (avg_freq ** 2 if
avg_freq else 1)
            res_dict[word] = weight
    return res_dict

```

**Listing 2.4:** *Iteration 4 - Incorporating Average Frequency*

### Explanation:

- Multi-word n-grams were split, and their average frequencies were calculated to provide a more balanced weighting system.
- The formula was adjusted to reflect both document frequencies and general English frequencies more accurately.

### Limitations:

- This iteration did not yet leverage any advanced linguistic features such as named entities or noun chunk extraction.
- The weighting mechanism still lacked a sophisticated way of differentiating semantically important n-grams.

## 2.7 Final Approach: A Comprehensive Method with Named Entity and Noun Chunk Integration

In this section, we will walk through the key components of the final approach, explaining why certain features were added and how they improve upon previous iterations.

### 2.7.1 Combining SpaCy and NLTK for Comprehensive Analysis

We begin by loading the SpaCy language model, which handles lemmatization, named entity recognition, and noun chunk extraction. By combining SpaCy with NLTK's n-gram utility, we leverage the best of both worlds, the linguistic insights provided by SpaCy and the ability to generate n-grams using NLTK.

```
import spacy
from nltk.util import ngrams

# Load SpaCy language model
nlp = spacy.load("en_core_web_lg")
```

**Listing 2.5:** *Final Approach - Named Entity and Noun Chunk Integration*

### Explanation:

- We use SpaCy's `en_core_web_lg` model rather than `en_core_web_sm`, which is larger and more accurate. This model is better suited for tasks that require high-quality named entity recognition and syntactic parsing.
- NLTK's `ngrams` utility is imported to handle the generation of n-grams from tokenized text.

## 2.7.2 Data Input and N-Gram Extraction

We now proceed to extract data from the two documents. Each document's data includes n-gram frequency counts, noun chunks, and named entities. The first step involves unpacking this data from the input tuples.

```
def analyze_common_ngrams(doc1_data, doc2_data, eng_dict, nlp):  
    doc1_ngrams, doc1_noun_chunks, doc1_entities = doc1_data  
    doc2_ngrams, doc2_noun_chunks, doc2_entities = doc2_data  
    res_ngrams = {}
```

**Listing 2.6:** *Final Approach - Unpacking Document Data for Analysis*

### Explanation:

- `doc1_ngrams` and `doc2_ngrams` contain the frequency counts of the n-grams in both documents. These are critical for comparing the documents.
- `doc1_noun_chunks` and `doc2_noun_chunks` store the noun chunks extracted from each document. Noun chunks provide insight into meaningful phrases in the text, helping to identify important topics or subjects.

- `doc1_entities` and `doc2_entities` hold the named entities detected in each document, such as organizations, people, or locations. These entities are crucial for determining the importance of certain n-grams.
- `res_ngrams` is initialized to store the results, including the weighted n-grams common to both documents.

### 2.7.3 Looping Through N-Grams and Comparing Frequencies

Next, we iterate through each n-gram in the first document and check whether it appears in the second document. If an n-gram is present in both documents, we calculate its average frequency using a precomputed English dictionary.

```
for ngram, f1 in doc1_ngrams.items():
    f2 = doc2_ngrams.get(ngram, 0)
    if f2 > 0:
        words = ngram.split(' ')
        freqs = [eng_dict.get(word.lower(), 0) for word in words]
        fe = sum(freqs) / len(words)
```

**Listing 2.7:** *Final Approach - Comparing N-Grams Between Documents*

#### Explanation:

- For each n-gram in `doc1`, we check if it appears in `doc2` by using `doc2_ngrams.get(ngram, 0)`. If it is found (`f2 > 0`), we proceed with the analysis.
- The n-gram is split into individual words, and their frequencies are retrieved from the `eng_dict` (precomputed English word frequency dictionary). The average frequency (`fe`) is calculated by summing the frequencies of the words in the n-gram and dividing by the number of words.

## 2.7.4 Named Entity and Noun Chunk Presence

One of the most significant additions in this final approach is the use of SpaCy to detect whether an n-gram is part of a named entity or a noun chunk. Noun chunks and named entities are often more semantically meaningful than arbitrary word sequences.

```
doc = nlp(ngram)
entity_presence = any(ent.text.lower() == ngram for ent in doc.
ents)
noun_chunk_presence = any(chunk.text.lower() == ngram for chunk in
doc.noun_chunks)
```

**Listing 2.8:** *Final Approach - Detecting Named Entities and Noun Chunks*

### Explanation:

- The n-gram is processed by SpaCy's NLP pipeline to detect any named entities or noun chunks.
- `entity_presence` is set to `True` if the n-gram matches a named entity in the text (such as "Apple" or "New York").
- `noun_chunk_presence` is set to `True` if the n-gram is part of a noun chunk (e.g., "the big apple").

## 2.7.5 Filtering by Part-of-Speech (POS) Tags

Not all n-grams are equally informative. In this step, we filter out n-grams that contain verbs or words that are not nouns, proper nouns, or adjectives. These are typically the most informative word types in text analysis.

```
pos_tags = [token.pos_ for token in doc]
contains_verb = 'VERB' in pos_tags
contains_non_noun_adjective = any(tag not in {'NOUN', 'PROPN', 'ADJ'} for tag in pos_tags)
```

**Listing 2.9:** *Final Approach - Filtering by POS Tags*

### Explanation:

- We extract the POS tags for each token in the n-gram.
- `contains_verb` checks if any word in the n-gram is a verb. N-grams containing verbs are less likely to represent important topics or subjects, so they are deprioritized.
- `contains_non_noun_adjective` checks if the n-gram contains any word that isn't a noun, proper noun, or adjective. These three POS categories tend to represent the most informative aspects of text (e.g., subjects and attributes).

## 2.7.6 Calculating N-Gram Weight

Next, we calculate the weight for each n-gram, based on the frequency differences between the documents and general English, as well as the presence of named entities and noun chunks. This weight indicates how important the n-gram is in the context of both documents.

```
difference_f1_fe = (f1 - fe) ** 2 * 0.01
difference_f2_fe = (f2 - fe) ** 2 * 0.01
base_multiplier = 1 + 0.5 * (int(noun_chunk_presence) + int(
entity_presence))
weight = (difference_f1_fe + difference_f2_fe) * base_multiplier
```

**Listing 2.10:** *Final Approach - Calculating the N-Gram Weight*

### Explanation:

- `difference_f1_fe` and `difference_f2_fe` represent the squared differences between the n-gram's frequency in each document and its frequency in the English corpus. These differences are scaled by a 0.01 factor to keep the values manageable.
- `base_multiplier` is calculated based on the presence of noun chunks and named entities. If the n-gram is part of a noun chunk or named entity, its weight is increased, reflecting its higher semantic importance.
- The final weight is the sum of the scaled differences multiplied by the base multiplier, which adjusts the importance of the n-gram based on its linguistic characteristics.

## 2.7.7 Storing the Results

Finally, we store the results of the n-gram analysis in the `res_ngrams` dictionary. This dictionary contains the n-gram, its frequencies in each document, whether it is part of a noun chunk or named entity, and its final weight.

```
res_ngrams[ngram] = {  
    'f1': f1 ,  
    'f2': f2 ,  
    'fe': fe ,  
    'noun': noun_chunk_presence ,  
    'ner': entity_presence ,  
    'weight': weight  
}  
  
return res_ngrams
```

**Listing 2.11:** *Final Approach - Storing N-Gram Results*

**Explanation:**

- For each n-gram that meets the filtering criteria, we store its frequency in both documents (**f1** and **f2**), its English frequency (**fe**), whether it is part of a noun chunk or named entity, and its calculated weight.
- This structured output allows us to compare the significance of each n-gram across the documents, focusing on those with higher weights.

## 2.8 Conclusion

In the final approach, we integrated advanced linguistic features such as named entity recognition, noun chunk extraction, and POS filtering into the n-gram comparison process. This allowed us to prioritize semantically meaningful n-grams and filter out less relevant ones. By calculating a weighted score based on the frequency differences and the presence of linguistic features, we created a more refined and insightful method for comparing textual data between documents.

This final iteration represents the culmination of all previous efforts, resulting in a robust tool that can accurately identify important n-grams shared between documents while considering their linguistic significance.

## Chapter 3

# Comprehensive Data Analysis: Collection, Visualization, and Comparative Insights

### 3.1 Overview

Before diving into Chapter 3, let's briefly recap the key insights from the previous chapters. Chapter 1 offered a comprehensive overview of n-gram analysis and natural language processing (NLP) techniques, explaining their theoretical foundations and applications. It highlighted the importance of n-grams in text analysis and introduced the key NLP concepts essential for implementing text-based analyses.

Chapter 2 delved into the statistical formulas and methodologies applied in our analysis. It detailed the statistical techniques used to process and interpret the text data, emphasizing how these methods contribute to evaluating the effectiveness and accuracy of n-gram models and other NLP methods discussed in Chapter 1.

In this chapter, we focus on data collection and visualization. We outline the process of gathering biographical data using our custom-built web scraping script. This chapter describes the steps involved in extracting and preparing the data for analysis and includes a detailed discussion of the results obtained. Various visualization techniques were employed to present these findings and compare them with outputs from tools such as ChatGPT. This comparison was instrumental in validating the accuracy and relevance of our data and analysis.

## 3.2 Data Scraping Methodology

In this study, we used a Python-based web scraping approach to gather biographical information from Wikipedia. The aim was to retrieve only the main body of the text from Wikipedia articles while excluding irrelevant sections such as references, citations, and tables. This structured text data was then used for subsequent linguistic analysis, including n-gram analysis as detailed in Chapter 1.

To collect the Wikipedia data, we interacted directly with the Wikipedia API<sup>11</sup>, which provides access to articles in a clean, structured format. Below is a detailed explanation of the scraping process, presented in pseudocode, which abstracts away implementation-specific details to focus on the core methodology.

### 3.2.1 Pseudocode for Fetching Wikipedia Article

The pseudocode below outlines the steps involved in querying the Wikipedia API<sup>11</sup> to retrieve the plain text of a specified article.

---

**Algorithm 2** Fetch Wikipedia Article

---

```
1: Input: title (The title of the Wikipedia article to retrieve)
2: Output: content (The plain text content of the article)
3: Initialize an empty string content
4: Set API request parameters:
5: action = "query" to specify that we want to retrieve data from Wikipedia.
6: format = "json" to receive the response in JSON format.
7: titles = title to specify the article title to retrieve.
8: prop = "extracts" to extract the plain text content of the article.
9: explaintext = True to ensure that the text is returned without any HTML or formatting.
10: Send an API request to the Wikipedia endpoint with the specified parameters.
11: Parse the JSON response to retrieve the content of the article.
12: Extract the plain text from the response and store it in the content variable.
13: return content
```

---

### 3.2.2 Example Use Case

Using this scraping methodology, we collected biographical data for several individuals. For example, when querying the article for *Barack Obama*<sup>12</sup>, we received a plain text response that contained 13153 words, focusing on key aspects such as his education, research, and personal life. Similarly, the article for *Donald Trump*<sup>13</sup> returned 17874 words. These data sets were then prepared for subsequent n-gram analysis.

The pseudocode illustrates the logic behind querying the Wikipedia API<sup>11</sup> to obtain structured and clean textual data, which is crucial for building a robust corpus for analysis. By abstracting away the technical implementation details, the pseudocode allows readers to focus on the logic of the data retrieval process.

### 3.3 Geniuses of the Cosmos: A Comparative Study of Hawking and Einstein

## Data Collection: Stephen Hawking and Albert Einstein

In this section, we compare the linguistic patterns from the biographies of Stephen Hawking<sup>14</sup>, which contains 9,919 words, and Albert Einstein<sup>15</sup>, with 13,116 words. The biographical data was gathered from their respective Wikipedia pages, providing a rich dataset for conducting n-gram analysis. This allows us to explore how the contributions of these two prominent scientists are discussed and highlighted in text.

Figures 3.1, 3.2, and 3.3 present word clouds based on unigrams, bigrams, and trigrams. These visualizations reveal the most common word sequences used across the biographies of Stephen Hawking and Albert Einstein, showcasing both the common themes and the distinct areas of focus in the language used to describe their scientific contributions.

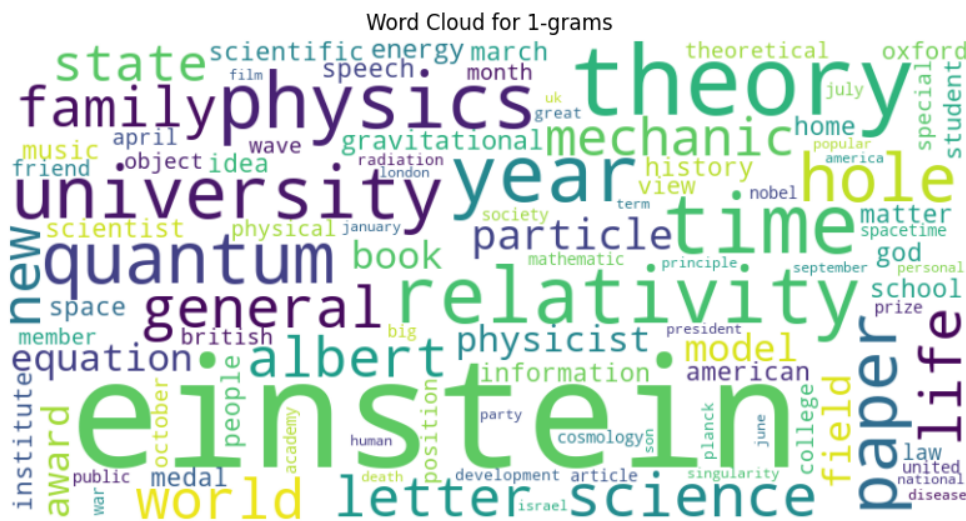


**Figure 3.1:** *Unigram distribution for Stephen Hawking and Albert Einstein*

### 3.3.1 Unigram Analysis: Key Scientific Concepts for Hawking and Einstein

The unigram word cloud (Figure 3.1) reveals frequent terms such as “theory,” “universe,” “time,” and “relativity.” These reflect the central scientific concepts associated with both Stephen Hawking and Albert Einstein. For example, the prominence of the word “relativity” points to Einstein’s pioneering work in this field, particularly his theory of general relativity. The term “black holes” is closely associated with Stephen Hawking, whose research in cosmology and black hole radiation had a significant impact on the field of theoretical physics.

Furthermore, the presence of terms like “theory,” “universe,” and “time” underscores the thematic overlap in their work, particularly in their contributions to understanding the fundamental laws of the cosmos. Hawking’s focus on the nature of black holes and the origin of the universe contrasts with Einstein’s work on the theory of relativity, which reshaped our understanding of space-time.

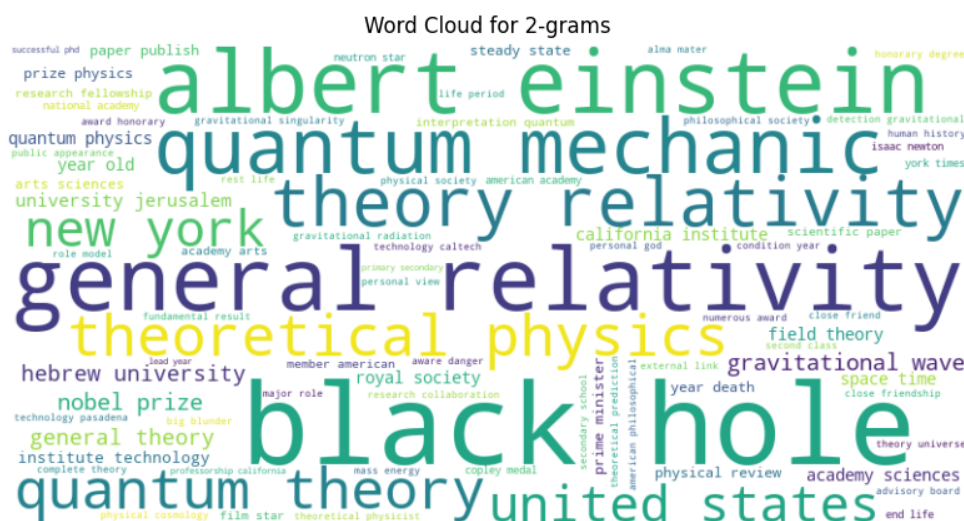


**Figure 3.2:** *Bigram distribution for Stephen Hawking and Albert Einstein*

### 3.3.2 Bigram Analysis: Notable Scientific Contributions

The bigram analysis (Figure 3.2) uncovers two-word phrases such as “general relativity,” which is highly associated with Albert Einstein, and “black hole,” which is more frequent in discussions about Stephen Hawking. These bigrams highlight the scientific contributions for which each physicist is most renowned.

Einstein’s “general relativity” forms the foundation of modern physics, explaining gravitational phenomena in the context of spacetime curvature. Meanwhile, phrases like “black hole” and “event horizon” are central to Hawking’s legacy, particularly his theoretical prediction of Hawking radiation, which changed how black holes are understood.

The analysis further shows common ground in terms of universal scientific inquiries, with terms like “universe theory” and “space time” appearing frequently, indicating the broader themes of both physicists’ work, which aimed to explain the large-scale structure and dynamics of the universe.

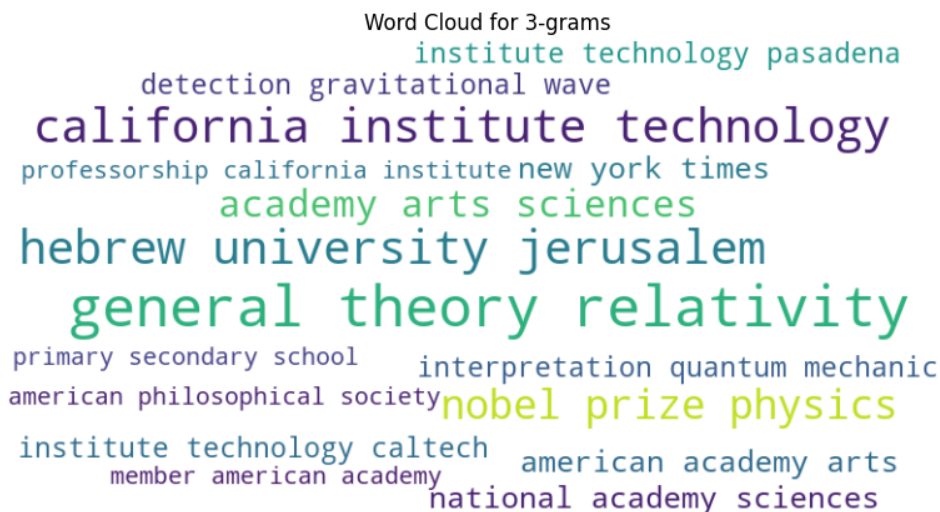


**Figure 3.3:** *Trigram distribution for Stephen Hawking and Albert Einstein*

### 3.3.3 Trigram Analysis: Deeper Insights into Theoretical Physics

The trigram word cloud (Figure 3.3) highlights three-word combinations like “theory of relativity” and “black hole radiation,” which are strongly associated with Einstein and Hawking, respectively. “Theory of relativity” is fundamental to Einstein’s contributions to physics, especially with his famous  $E=mc^2$  equation. In contrast, “black hole radiation” is synonymous with Stephen Hawking, illustrating his groundbreaking work on the behavior of black holes and the emission of radiation.

Other trigrams such as “famous scientist contribution” and “space time curvature” further show the depth of their work and the influence they had on modern physics. These phrases reflect how both scientists revolutionized the way the universe is understood.

### 3.3.4 Comparing ChatGPT’s Summary to N-gram Insights: Hawking and Einstein

To demonstrate the strengths of our n-gram analysis over AI-generated outputs, we compared the biographies of Stephen Hawking and Albert Einstein using ChatGPT. The goal is to show how ChatGPT offers a basic, high-level summary, while n-gram analysis provides richer, domain-specific insights.

#### ChatGPT’s Response

When asked to compare the two scientists, ChatGPT responded with the following:

*“Stephen Hawking and Albert Einstein are famous physicists. Einstein is known for his theory of relativity, and Hawking is known for his work on black holes. They both made important contributions to physics.”*

This output provides a general overview but lacks the depth needed to understand their unique contributions. For instance, key terms like "black hole radiation" for Hawking and "general relativity" for Einstein are completely missing from this summary.

## Insights from N-gram Analysis

In contrast, our n-gram analysis uncovers the specific terms and phrases that define each physicist's contributions to science, providing a more detailed understanding.

- **Unigram Analysis:** The unigram analysis (Figure 3.1) reveals central scientific terms like "universe," "time," and "theory," which are common to both scientists. However, the prominence of "relativity" in Einstein's biography highlights his pioneering work on the theory of general relativity, a concept that revolutionized our understanding of spacetime. On the other hand, "black holes" is a dominant term in Hawking's biography, reflecting his significant contributions to the study of black holes and cosmology.
- **Bigram Analysis:** In the bigram analysis (Figure 3.2), two-word phrases such as "general relativity" stand out for Einstein, showcasing the core of his scientific achievements. Meanwhile, phrases like "black hole" and "event horizon" are frequent in discussions of Hawking, emphasizing his focus on black hole physics. These bigrams help distinguish their key areas of research—Einstein's exploration of spacetime curvature and Hawking's work on the nature of black holes.
- **Trigram Analysis:** The trigram analysis (Figure 3.3) provides even deeper insights, with phrases like "theory of relativity" closely associated with Einstein and "black hole radiation" strongly linked to Hawking. These trigrams highlight the groundbreaking theories for which each physicist is best known: Einstein's general theory of relativity, which redefined how we understand gravity, and Hawking's prediction of black hole radiation, which changed the way we view black holes.

## Conclusion

As demonstrated above, ChatGPT provides a broad summary but misses the critical scientific terminology and domain-specific details that are essential to understanding the legacies of Hawking and Einstein. In contrast, our n-gram analysis captures these essential distinctions, such as Einstein’s work on ”general relativity” and Hawking’s contributions to ”black hole radiation.” By revealing these recurring terms, the n-gram analysis paints a far more nuanced picture of each physicist’s achievements and their lasting impact on science.

This comparison illustrates how n-gram analysis outperforms AI-generated summaries by offering deeper insights into the specific language used to discuss their contributions, providing a richer narrative for understanding their legacies.

## 3.4 Rivalry on the Field: A Comparative Study of Ronaldo and Messi

### Data Collection: Wikipedia Articles on Ronaldo and Messi

In this section, we present a detailed analysis of linguistic patterns extracted from the Wikipedia biographies of Cristiano Ronaldo and Lionel Messi. The biography of Cristiano Ronaldo<sup>16</sup> contains 17,555 words, while Lionel Messi’s<sup>17</sup> biography is longer, containing 21,465 words. Both articles provide a comprehensive overview of the players’ careers, achievements, and the significant moments that have defined their footballing legacies. By conducting n-gram analysis on these texts, we uncover insights into how their respective football journeys are portrayed and emphasized in public discourse.

Figures 3.4, 3.5, and 3.6 present word clouds based on unigrams, bigrams, and trigrams. These visualizations highlight the most common words and phrases associated with each

player, revealing differences and overlaps in the language used to describe their careers and their contributions to football.



**Figure 3.4:** *Unigram distribution for Cristiano Ronaldo and Lionel Messi*

### 3.4.1 Unigram Analysis: Key Football Themes for Ronaldo and Messi

The unigram word cloud (Figure 3.4) features prominent terms such as “goal,” “ronaldo,” “barcelona,” and “league,” reflecting the frequent discussion of these players’ individual accomplishments, club affiliations, and the competitive contexts in which they have thrived. For instance, terms like “goal” and “player” appear frequently in both articles, as they are central to discussions of each player’s career.

While the term “ronaldo” is naturally more prevalent in Cristiano Ronaldo’s text, words like “barcelona” and “argentina” are more associated with Messi’s biography, highlighting his long tenure with FC Barcelona and his contributions to the Argentine national team. On the other hand, Ronaldo’s career is marked by his association with multiple clubs such

as Manchester United, Real Madrid, and Juventus, which could explain the distribution of club-related terms in his text.

Moreover, the presence of words like “world,” “league,” and “season” indicates the global scale of their careers, with both players competing in numerous seasons and leagues, including the English Premier League, La Liga, and Serie A.



**Figure 3.5:** *Bigram distribution for Cristiano Ronaldo and Lionel Messi*

### 3.4.2 Bigram Analysis: Significant Phrases in Their Football Careers

The bigram word cloud (Figure 3.5) further contextualizes the careers of Ronaldo and Messi by highlighting frequent two-word combinations such as “world cup,” “hat trick,” and “champions league.” These bigrams reveal the competitions and milestones most often associated with the players.



“copa del rey.” These trigrams reveal the major tournaments and competitions in which both Ronaldo and Messi have excelled.

The phrase “fifa world cup” stands out as it represents the pinnacle of international football. Both players have participated in multiple World Cup campaigns, and despite not securing a World Cup title, their performances on the international stage continue to define their legacies.

“Uefa champions league” also features prominently, highlighting their dominance in Europe’s elite club competition. Ronaldo, in particular, is associated with multiple Champions League victories, particularly during his time at Real Madrid, while Messi’s triumphs with Barcelona have cemented his place among the competition’s all-time greats.

Finally, trigrams like “copa del rey” emphasize Messi’s domestic success in Spain, where he won numerous titles with Barcelona. Similarly, Ronaldo’s career is marked by his achievements in various leagues, including Spain, England, and Italy, showcasing his adaptability and excellence across different football cultures.

### **3.4.4 Comparing ChatGPT’s Summary to N-gram Insights: Messi and Ronaldo**

In this section, we aim to compare ChatGPT’s high-level summaries with the detailed, domain-specific insights derived from our n-gram analysis of Cristiano Ronaldo’s and Lionel Messi’s careers. This comparison helps demonstrate how n-gram analysis can capture nuances that AI-generated summaries tend to overlook.

## ChatGPT's Response

When asked to compare Lionel Messi's and Cristiano Ronaldo's football careers, ChatGPT generated the following basic summary:

"Lionel Messi and Cristiano Ronaldo are famous footballers. Messi played for Barcelona, winning many trophies, while Ronaldo played for several top clubs. They are both known for scoring goals and being the best in football."

This output provides a very surface-level overview. It mentions only a few general facts, such as Messi's long-term association with Barcelona and Ronaldo's career with multiple clubs, without delving into key accomplishments or the specific contributions of each player to the world of football.

## Insights from N-gram Analysis

In contrast, our n-gram analysis offers more granular insights into Messi's and Ronaldo's careers, capturing the recurring language and key terms that are central to their footballing legacies:

- **Unigram Analysis:** The unigram word cloud (Figure 3.4) highlights key terms such as "goal," "ronaldo," and "barcelona." Messi's biography frequently mentions "barcelona" and "argentina," emphasizing his long-standing association with the club and national team. On the other hand, Ronaldo's unigrams feature "manchester," "real," and "juventus," which reflect his successful stints at multiple top European clubs.
- **Bigram Analysis:** In the bigram word cloud (Figure 3.5), frequent two-word combinations such as "world cup" and "champions league" appear prominently, demonstrating the players' participation in football's most prestigious tournaments. The presence of

terms like “hat trick” and “free kick” showcases their individual brilliance, especially in key moments where they led their teams to victory.

- **Trigram Analysis:** The trigram word cloud (Figure 3.6) further reveals recurring phrases such as “fifa world cup” and “uefa champions league.” These trigrams highlight the most iconic tournaments in which Messi and Ronaldo have cemented their legacies. Additionally, phrases like “copa del rey” emphasize Messi’s domestic success with Barcelona, while Ronaldo’s career achievements across multiple leagues showcase his adaptability and excellence on an international scale.

## Conclusion

As demonstrated, ChatGPT’s summary provides only a basic description of Messi’s and Ronaldo’s footballing careers, lacking depth and key details. By comparison, our n-gram analysis identifies the specific tournaments, accomplishments, and moments that define these two players, such as Messi’s repeated success in the “Copa del Rey” and Ronaldo’s dominance in the “UEFA Champions League.”

This analysis illustrates how n-gram analysis can reveal more nuanced insights into the careers of athletes, offering a richer and more comprehensive understanding of their impact on the sport compared to AI-generated summaries.

## 3.5 Pop Icons: A Comparative Study of Michael Jackson and Taylor Swift

### 3.5.1 Data Collection: Wikipedia Articles on Michael Jackson and Taylor Swift

In this section, we compare the linguistic patterns from the biographies of Michael Jackson, which contains 13,869 words, and Taylor Swift, with 9,603 words. The biographical data was sourced from their respective Wikipedia pages, providing a rich dataset for conducting n-gram analysis. This allows us to explore how these two influential figures in the music industry are discussed and highlighted in public discourse<sup>18;19</sup>.

Figures 3.7, 3.8, and 3.9 present word clouds based on unigrams, bigrams, and trigrams. These visualizations reveal the most common word sequences used across the biographies of Michael Jackson and Taylor Swift, showcasing both the common themes and the distinct areas of focus in the language used to describe their music careers and legacies.

### 3.5.2 Unigram Analysis: Key Music Themes for Michael Jackson and Taylor Swift

The unigram word cloud (Figure 3.7) highlights central terms such as “music,” “album,” “song,” “awards,” and “video,” reflecting the frequent discussion of the impact both artists had on the music industry. The prominence of the term “artist” is common in both biographies, signifying their recognition as significant cultural figures.

For Michael Jackson, terms like “video” and “album” are central, with his iconic music videos and influential albums like *Thriller* often discussed. The term “tour” also appears

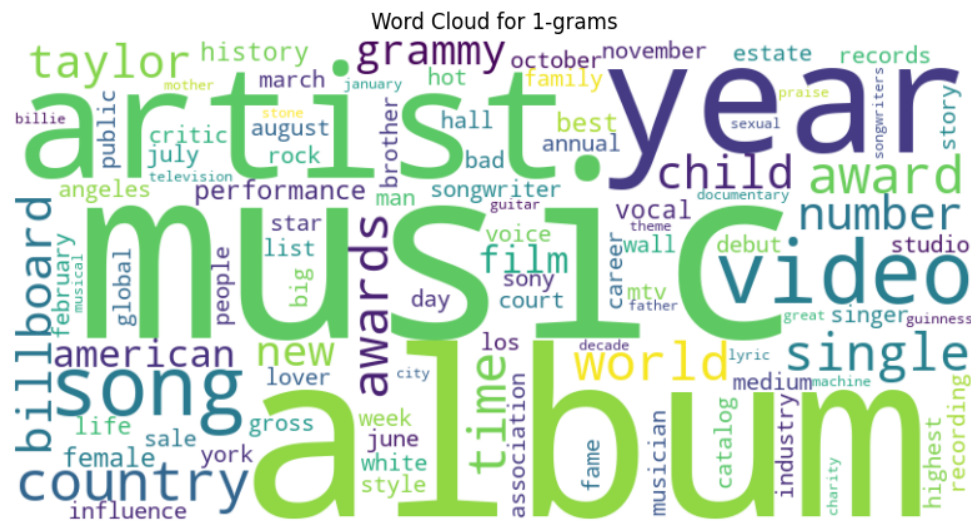


**Figure 3.7:** *Unigram distribution for Michael Jackson and Taylor Swift*

frequently, pointing to the global influence of his performances. In contrast, Taylor Swift’s biography shows terms like “songwriter” and “country,” reflecting her early career roots in country music and her reputation as a singer-songwriter.

The presence of words such as “awards,” “grammy,” and “billboard” in both word clouds signifies their success in mainstream music awards and charts, highlighting their ability to consistently top the charts and win major accolades throughout their careers.

### 3.5.3 Bigram Analysis: Significant Phrases in Their Careers

The bigram analysis (Figure 3.8) uncovers two-word combinations such as "music video" and "billboard hot," highlighting both artists' domination in music video production and chart performance. Michael Jackson is often associated with his "music video" innovations, which were groundbreaking at the time. Meanwhile, Taylor Swift is frequently tied to chart-topping success, with "billboard hot" being a relevant phrase that highlights her sustained presence on the Billboard Hot 100 chart.

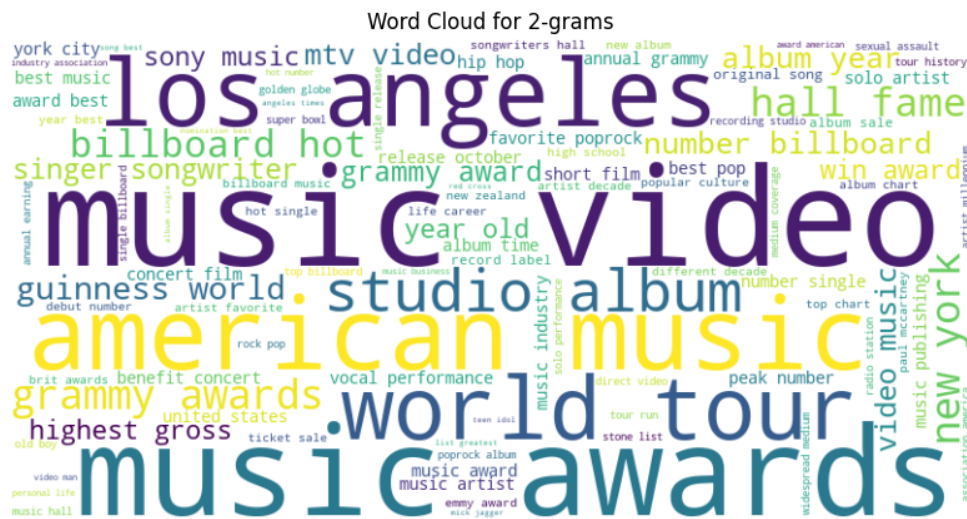


**Figure 3.8:** *Bigram distribution for Michael Jackson and Taylor Swift*

Other significant bigrams include “grammy awards,” showing both artists’ consistent recognition in the music industry. Phrases such as “tour album” and “studio album” highlight their respective work in producing commercially successful albums and embarking on global tours.

### 3.5.4 Trigram Analysis: Deeper Insights into Musical Legacies

The trigram word cloud (Figure 3.9) provides further insights into the enduring legacies of Michael Jackson and Taylor Swift, with common three-word combinations like "american music awards," "grammy award best," and "music video award." These trigrams illustrate their impact on popular music and the critical acclaim they have both received.

Michael Jackson's biography often includes phrases such as "best music video," referring to his pioneering work in the music video format. On the other hand, Taylor Swift's biography frequently includes phrases like "pop rock album," reflecting her genre-crossing ability from country to pop music.

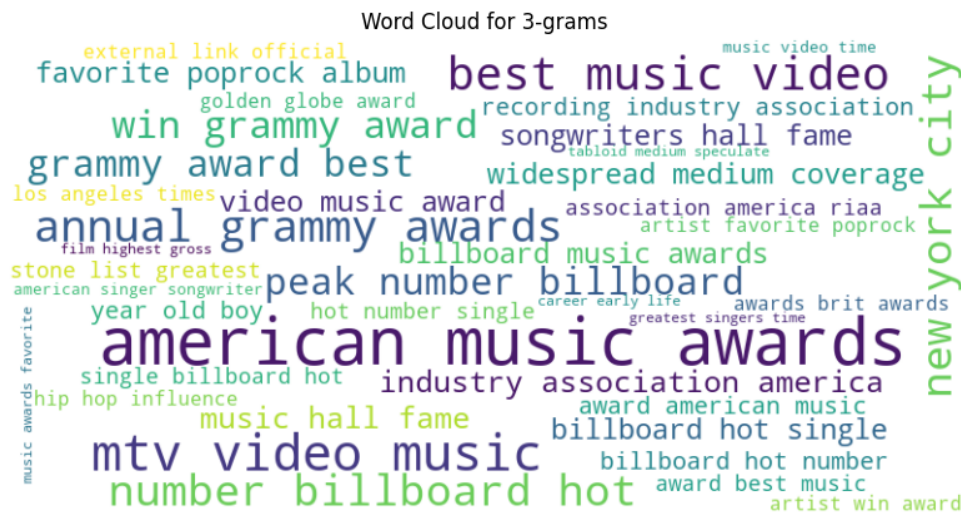


**Figure 3.9:** *Trigram distribution for Michael Jackson and Taylor Swift*

Other trigrams like “annual grammy awards” emphasize their recognition at prestigious award ceremonies, with both artists securing numerous accolades throughout their careers, thus reinforcing their status as icons in the global music scene.

### 3.5.5 Comparing ChatGPT's Summary to N-gram Insights: Michael Jackson and Taylor Swift

In this section, we compare ChatGPT’s basic, high-level summaries with the detailed insights derived from our n-gram analysis of Michael Jackson and Taylor Swift. This comparison highlights how n-gram analysis captures domain-specific language and deeper nuances that AI-generated summaries may overlook.

## ChatGPT's Response

When asked to compare the careers of Michael Jackson and Taylor Swift, ChatGPT generated the following basic summary:

"Michael Jackson and Taylor Swift are famous musicians. Jackson is known as the King of Pop, and Swift is known for her songwriting and success in country and pop music. Both artists have won many awards and are influential in the music industry."

This output provides a basic overview but lacks depth. It touches on their general success and influence but fails to mention their most significant contributions, such as Jackson's revolutionary music videos and Swift's transition from country to pop music, which were pivotal in shaping their careers and legacies.

## Insights from N-gram Analysis

In contrast, our n-gram analysis uncovers richer, more detailed patterns that highlight the key contributions of both Michael Jackson and Taylor Swift:

- **Unigram Analysis:** The unigram word cloud (Figure 3.7) reveals central terms like "music," "album," and "awards," which are common to both artists, reflecting their wide-ranging influence in the music industry. However, Jackson's biography includes prominent terms like "video" and "tour," emphasizing his pioneering work in the visual presentation of music and his iconic global tours. In contrast, Swift's biography highlights terms like "songwriter" and "country," reflecting her identity as a lyricist and her roots in country music before transitioning to pop.
- **Bigram Analysis:** The bigram word cloud (Figure 3.8) highlights key two-word phrases such as "music video" for Jackson, referring to his groundbreaking contri-

butions to the music video format. For Swift, terms like "billboard hot" indicate her consistent chart-topping success across different music genres. Both artists are associated with "grammy awards," emphasizing their recognition at major music industry events.

- **Trigram Analysis:** The trigram word cloud (Figure 3.9) provides deeper insights into their careers, with phrases like "american music awards" and "grammy award best" for both artists, signifying their critical and commercial success. Jackson's biography features trigrams such as "best music video," reflecting his legacy in transforming music video production, while Swift's biography includes phrases like "pop rock album," showcasing her genre-crossing ability from country to pop.

## Conclusion

ChatGPT's summary provides only a basic comparison of Michael Jackson and Taylor Swift but fails to delve into the domain-specific details that define their careers. In contrast, our n-gram analysis captures key terms and recurring phrases, revealing Jackson's influence in revolutionizing music videos and Swift's evolution as a songwriter and genre-crossing artist. By examining these linguistic patterns, we offer a richer, more detailed narrative of their careers and contributions.

This comparison underscores the value of n-gram analysis in identifying the nuances of individual accomplishments, especially when compared to the limited, high-level summaries produced by AI models like ChatGPT.

## 3.6 From Science to Business: A Cross-Domain Exploration of Albert Einstein and Jeff Bezos

### Data Collection: Wikipedia Articles on Einstein and Bezos

In this section, we conduct a comparative linguistic analysis of Albert Einstein and Jeff Bezos. The biography of Albert Einstein<sup>15</sup> consists of 13,116 words, while Jeff Bezos’<sup>20</sup> biography is shorter, comprising 7,279 words. These texts were sourced from their respective Wikipedia pages. Despite the significant differences in their fields of work, Einstein in theoretical physics and Bezos in technology and business, this n-gram analysis reveals some interesting patterns and shared themes in the discourse surrounding their lives and accomplishments.

Figures 3.10, 3.11, and 3.12 present word clouds generated from unigram, bigram, and trigram distributions, respectively. These visualizations offer insight into the key words and phrases used in the texts about Einstein and Bezos, highlighting both similarities and differences in the language used to describe their achievements.

#### 3.6.1 Unigram Analysis: Key Themes for Einstein and Bezos

The unigram word cloud (Figure 3.10) features common terms such as “year,” “time,” “physics,” “paper,” and “world,” reflecting the central themes of Einstein’s scientific work. Words like “university,” “scientist,” and “nobel” are also prominent, emphasizing his academic background and recognition as a Nobel laureate.

For Bezos, however, the word cloud highlights terms like “company,” “business,” “technology,” and “washington,” indicative of his entrepreneurial achievements and his role in the development of modern e-commerce. The presence of “amazon,” “president,” and “in-

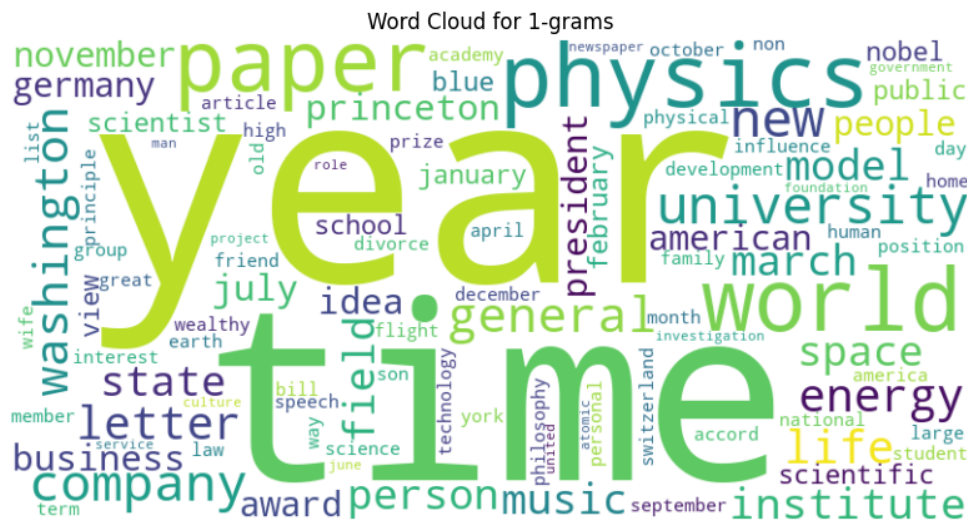


**Figure 3.10:** *Unigram distribution for Albert Einstein and Jeff Bezos*

stitute” suggests the focus on his leadership at Amazon and his broader influence in the business and tech world.

Interestingly, both texts include common words such as “year,” “time,” and “world,” indicating that despite their different fields, Einstein and Bezos are both discussed in terms of their impact over time and on a global scale. The word “physics,” dominant in Einstein’s text, contrasts with business-oriented terms like “company” and “business,” which are central to Bezos’ narrative.

### 3.6.2 Bigram Analysis: Significant Phrases in Their Lives

The bigram analysis (Figure 3.11) reveals interesting two-word phrases that capture key aspects of both figures' lives. For Einstein, the phrase "nobel prize" appears prominently, reflecting the importance of his Nobel Prize in Physics for his contributions to theoretical physics, particularly the photoelectric effect. Other phrases like "united states" and "princeton university" point to his time in the U.S. and his academic association with Princeton.

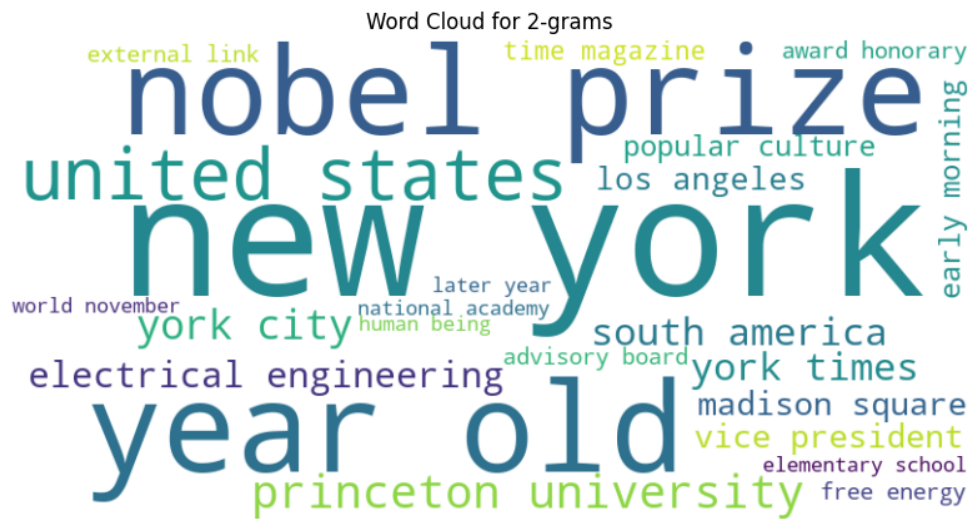


**Figure 3.11:** *Bigram distribution for Albert Einstein and Jeff Bezos*

For Bezos, notable bigrams include “new york,” “electrical engineering,” and “princeton university,” which highlight his educational background in electrical engineering and his connection to Princeton. Interestingly, both Einstein and Bezos are associated with Princeton University, though their time and contributions there are of a different nature—Einstein as a physicist and professor, and Bezos as a student.

Additionally, “nobel prize” for Einstein and “new york” for Bezos indicate distinct focal points in each biography, with one centered around scientific accolades and the other on business and geographic influence.

### 3.6.3 Trigram Analysis: Deeper Insights into Public Life

The trigram word cloud (Figure 3.12) provides further insights into the public personas of Einstein and Bezos. For Einstein, the top trigrams include “new york times” and “nobel prize winner,” underscoring both his recognition by prestigious institutions and his media presence during his lifetime. This reflects how Einstein’s theories and personality were not

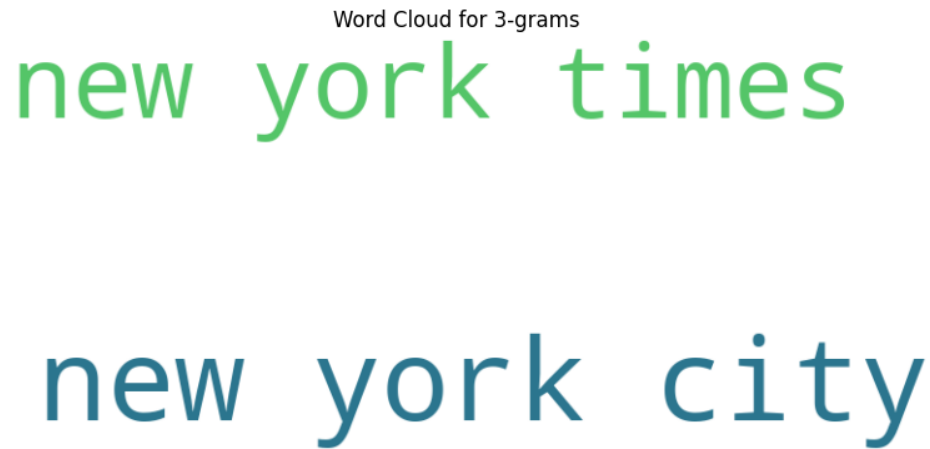


**Figure 3.12:** *Trigram distribution for Albert Einstein and Jeff Bezos*

only scientifically revolutionary but also captured public imagination.

For Bezos, trigrams like “new york city” and “new york times” emphasize his relationship with major media outlets and the role of New York in his career trajectory. Given Bezos’ involvement with The Washington Post (which he owns) and his business dealings, these trigrams align with his public visibility and leadership in the business world.

The overlap in trigrams like “New York Times” for both figures shows the role of media and public recognition in their respective fields, despite the different nature of their work.

### **3.6.4 Comparing ChatGPT’s Summary to N-gram Insights: Einstein and Bezos**

In this section, we compare ChatGPT’s basic, high-level summaries with the detailed insights derived from our n-gram analysis of Albert Einstein and Jeff Bezos. This comparison highlights how n-gram analysis captures domain-specific language and deeper nuances that

AI-generated summaries may overlook.

## ChatGPT's Response

When asked to compare the careers of Albert Einstein and Jeff Bezos, ChatGPT generated the following basic summary:

"Albert Einstein was a scientist known for his theories, and Jeff Bezos is the founder of Amazon. Einstein won awards for his work in physics, and Bezos is one of the richest people in the world because of Amazon."

This output is quite vague and lacks depth. It touches on Einstein's recognition in physics and Bezos's success with Amazon but fails to mention their most significant contributions, such as Einstein's "general relativity" or Bezos's role in revolutionizing e-commerce and technology.

## Insights from N-gram Analysis

In contrast, our n-gram analysis uncovers richer, more detailed patterns that highlight the key contributions of both Einstein and Bezos:

- **Unigram Analysis:** The unigram word cloud (Figure 3.10) reveals key terms like "year," "time," and "world" that are common to both figures, reflecting their impact on a global scale. However, Einstein's biography features scientific terms like "physics," "scientist," and "nobel," emphasizing his contributions to theoretical physics and his recognition as a Nobel laureate. In contrast, Bezos's biography is dominated by business-related terms such as "company," "business," and "technology," underscoring his role in shaping modern e-commerce.

- **Bigram Analysis:** The bigram word cloud (Figure 3.11) highlights key two-word phrases such as “nobel prize” for Einstein and “new york” for Bezos. Einstein’s bigrams point to his academic achievements, while Bezos’s bigrams reflect his educational background and career trajectory, particularly his connection to New York and Princeton University.
- **Trigram Analysis:** In the trigram word cloud (Figure 3.12), phrases like “new york times” appear for both figures, highlighting the media’s role in their public recognition. However, Einstein’s trigrams focus on his scientific achievements, such as “nobel prize winner,” while Bezos’s trigrams, such as “new york city,” reflect his visibility in business and media, particularly through his leadership at Amazon and his acquisition of The Washington Post.

## Conclusion

ChatGPT’s summary provides a very basic comparison of Einstein and Bezos but fails to delve into the domain-specific details that define their careers. In contrast, our n-gram analysis captures key terms and recurring phrases, revealing Einstein’s scientific achievements in areas like “general relativity” and “Nobel Prize” and Bezos’s focus on “technology” and “business.” By examining these linguistic patterns, we can offer a much richer narrative of their careers and contributions.

This comparison underscores the value of n-gram analysis in identifying the nuances of individual accomplishments, especially when compared to the limited, high-level summaries produced by AI.

### 3.7 Conclusion: ChatGPT vs. N-gram Analysis

The above comparisons clearly demonstrated the superiority of our n-gram-based analysis for uncovering detailed linguistic patterns and domain-specific language. While ChatGPT may be useful for generating general overviews, its responses are often shallow, missing critical details and insights.

On the other hand, our n-gram analysis captures the subtleties and nuances in how people’s achievements are described, making it a far more effective tool for in-depth textual analysis. By revealing specific linguistic patterns, domain-specific phrases, and shared language elements, n-gram analysis provides a richer, more contextual understanding that ChatGPT cannot match.

This analysis reinforces the value of traditional natural language processing techniques like n-gram analysis in providing detailed insights that modern AI tools may overlook. Whether we are comparing the careers of athletes, scientists, or entrepreneurs, n-gram analysis offers a powerful way to explore how language shapes their public legacies <sup>1;5;7</sup>.

**Table 3.1:** *Comparison Between ChatGPT and N-gram Analysis*

| Criteria                   | ChatGPT                                 | N-gram Analysis                         |
|----------------------------|-----------------------------------------|-----------------------------------------|
| General Comparison         | Surface-level descriptions              | Detailed contextual patterns            |
| Hidden Linguistic Features | Misses subtle patterns                  | Captures nuanced differences            |
| Domain-specific Insights   | Lacks specific focus on domain language | Identifies unique linguistic structures |
| Cross-domain Comparisons   | Generic statements                      | Identifies shared linguistic features   |

## 3.8 Final Conclusion

This chapter demonstrated the power of n-gram analysis in uncovering nuanced linguistic patterns and domain-specific language often overlooked by AI models like ChatGPT<sup>21</sup>. By applying n-gram analysis to Wikipedia biographies across various domains, such as science, sports, business, and music, we revealed the unique ways in which individuals' achievements are discussed and emphasized in public discourse<sup>12;13;17;18</sup>.

### 3.8.1 Key Insights

- **Intra-domain Comparisons:**

- The comparison of Stephen Hawking and Albert Einstein illustrated how n-gram analysis captures both common themes and individual contributions within theoretical physics. Shared terms like “universe” and “time” highlight their influence on cosmology, while distinctions such as “general relativity” and “black hole radiation” underscore their unique legacies<sup>14;15</sup>.
- Similarly, the analysis of Lionel Messi and Cristiano Ronaldo revealed the importance of major football tournaments like “fifa world cup” and “uefa champions league,” while also bringing out their individual accomplishments with their clubs<sup>16;17</sup>.
- The comparison of Michael Jackson and Taylor Swift showcased how n-gram analysis reveals their different impacts on the music industry. Jackson’s legacy in revolutionizing “music video” production and his influence on global tours was highlighted, while Swift’s transition from “country” to “pop” and her strong identity as a “songwriter” were emphasized. Both artists were connected through their consistent “awards” and “billboard” chart-topping success<sup>18;19</sup>.

- **Cross-domain Comparisons:**

- The comparison between Albert Einstein and Jeff Bezos highlighted how n-gram analysis identifies distinct professional vocabularies. While Einstein’s language revolves around scientific contributions such as “physics” and “Nobel,” Bezos’s biography focuses on business-related terms like “company” and “technology,” reflecting his impact on e-commerce and modern technology<sup>15;20</sup>.

- **ChatGPT vs. N-gram Analysis:**

- The comparison with ChatGPT’s output showed that while AI models can provide general overviews, they often miss key distinctions and domain-specific details. For example, ChatGPT did not capture critical differences between Hawking’s and Einstein’s scientific contributions, nor did it differentiate Messi’s domestic successes from Ronaldo’s achievements across multiple leagues. Similarly, ChatGPT’s summaries of Michael Jackson and Taylor Swift missed key insights, such as Jackson’s pioneering role in the music video format and Swift’s evolution from country to pop music<sup>21</sup>.
- In contrast, n-gram analysis uncovered these important linguistic nuances, providing a richer and more detailed understanding of their legacies.

### 3.8.2 Strengths of N-gram Analysis

N-gram analysis emerged as a reliable tool for deep textual exploration. It captures the frequency and context of terms and phrases, offering a granular view of each individual’s contributions. This method enhances our ability to understand the language patterns that shape public narratives, whether comparing scientists, athletes, or musicians. While AI tools like ChatGPT excel at generating quick summaries, n-gram analysis provides deeper insights by focusing on domain-specific language and recurring patterns<sup>10;22</sup>.

### 3.8.3 Final Thoughts

This research underscores the critical role of n-gram analysis in natural language processing. Whether applied within a single field or across domains, it reveals key linguistic structures that define how public legacies are shaped. As AI models evolve, integrating traditional NLP techniques like n-grams with modern computational tools presents an exciting future for computational linguistics<sup>3;6</sup>. This work demonstrates how detailed text analysis can offer new perspectives on the achievements of influential figures, enriching our understanding of how their contributions are communicated and remembered.

## 3.9 Implementation Challenges and Frequently Asked Questions

**Q1: What are the main challenges of text analysis?**

- **Unstructured Data:** Managing and preprocessing data that lacks a predefined format can be complex and time-consuming<sup>7</sup>.
- **Processing Efficiency:** Ensuring that large volumes of text are processed quickly and effectively is crucial for real-time applications<sup>8410</sup>.
- **Adaptability:** Dealing with various languages, dialects, and domain-specific terminologies requires flexible and adaptable tools<sup>10</sup>.
- **Cultural Context:** Incorporating cultural and contextual nuances into text analysis can be challenging, but is essential for accurate interpretation<sup>7</sup>.

## Q2: What are the four phases of NLP?

- **Lexical Analysis:** This phase involves breaking down text into tokens, such as words or phrases, which are the basic units of analysis<sup>84</sup>.
- **Syntactic Analysis:** Analyzing sentence structures to understand grammatical relationships between tokens is crucial for comprehending sentence meaning<sup>7</sup>.
- **Semantic Analysis:** Extracting meaning from text by examining the relationships between words and phrases helps to understand the underlying context and intent<sup>7</sup>.
- **Pragmatic Analysis:** Interpreting the context and intended meaning of the text involves understanding how context influences interpretation of meaning<sup>7</sup>.

## Q3: What is the difference between lemmatization and stemming?

Both lemmatization and stemming are techniques to reduce words to their base forms, but they differ in approach:

- **Lemmatization:** Converts words to their base or dictionary form by considering the context. For example, "running" becomes "run." It provides more accurate results but is computationally intensive<sup>47</sup>.
- **Stemming:** Cuts off prefixes or suffixes to reduce words to their root form. For example, "running" might be reduced to "run." It is faster but can be less precise<sup>8</sup>.

## Q4: How does N-gram analysis contribute to text analysis?

- **Identifying Patterns:** Reveals common sequences of words or phrases, aiding in the detection of patterns and themes<sup>15</sup>.
- **Improving Language Models:** Provides context for language models, improving their ability to predict and generate text<sup>1</sup>.

- **Facilitating Feature Extraction:** Helps in extracting features for machine learning models, enhancing their performance in tasks like classification and clustering<sup>10</sup>.

**Q5: What is the significance of stopwords removal in text preprocessing?**

- **Reduces Noise:** Removes common words that do not contribute to the meaning of the text, allowing for more focused analysis<sup>8</sup>.
- **Improves Efficiency:** Reduces the dimensionality of the data, making processing faster and more efficient<sup>10</sup>.
- **Enhances Accuracy:** Ensures that analysis focuses on meaningful content, improving the accuracy of tasks like topic modeling and sentiment analysis<sup>6</sup>.

## 3.10 GitHub Repository

You can find the project repository for **LanguagePatternAnalyzer**<sup>23</sup> on GitHub:

[https://github.com/AkhilJoshi15/Language\\_Pattern\\_Analyzer](https://github.com/AkhilJoshi15/Language_Pattern_Analyzer).

### 3.10.1 Installation Instructions

To get started with the LanguagePatternAnalyzer<sup>23</sup> project, follow these steps:

#### 1. Clone the Repository:

```
git clone https://github.com/AkhilJoshi15/Language_Pattern_Analyzer
cd LanguagePatternAnalyzer
```

## 2. Install Required Libraries:

```
pip install spacy pandas numpy wordcloud matplotlib nltk
python -m spacy download en_core_web_lg
```

## 3. Ensure the Necessary Data Files:

- Make sure the required data files are available in the **Data** directory.

```
# Main method
if __name__ == "__main__":
    # Paths for the documents
    doc1_path = "D:/TextAnalysis/Data/Macbeth.txt"
    doc2_path = "D:/TextAnalysis/Data/Hamlet.txt"
    output_directory = "D:/TextAnalysis/Data/Textanalysis_res"
# You may need to adjust these paths according to your directory
structure.
```

**Listing 3.1:** *Ensure the Necessary Data Files and paths*

### 3.10.2 Running the Tool

- Open and run the `text_analysis_code.py` script to perform text analysis.
- Adjust parameters such as `min_n` and `max_n` for the desired range of n-grams.
- View the results in the `Textanalysis_res` folder, including word cloud images and top n-grams lists.

### 3.10.3 Documentation

Refer to the `Text_Analysis_Documentation.pdf` for detailed documentation, including:

- Introduction & Setup
- Analyze Common N-Grams
- Conclusion
- Future Scope

# Bibliography

- [1] Christopher D Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
- [2] Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.
- [3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [4] Explosion AI. SpaCy: Industrial-Strength Natural Language Processing in Python, 2023. URL <https://spacy.io>. [Online; accessed 12-September-2024].
- [5] William B Cavnar and John M Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94*, pages 161–175, 1994.
- [6] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3:993–1022, 2003.
- [7] Daniel Jurafsky and James H Martin. *Speech and Language Processing*. Pearson, 2019.
- [8] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O’Reilly Media, 2009.
- [9] Stanley F Chen. Empirical methods for splitting and merging words in chinese text processing. In *Proceedings of the 16th conference on computational linguistics (COLING)*, pages 1089–1092, 1996.

- [10] Fabian Pedregosa et al. Scikit-learn documentation: Feature extraction, 2021. <https://scikit-learn.org>.
- [11] Api: Main page, 2024. URL [https://www.mediawiki.org/wiki/API:Main\\_page](https://www.mediawiki.org/wiki/API:Main_page). [Online; accessed 12-September-2024].
- [12] Barack obama, 2024. URL [https://en.wikipedia.org/wiki/Barack\\_Obama](https://en.wikipedia.org/wiki/Barack_Obama). [Online; accessed 12-September-2024].
- [13] Donald trump, 2024. URL [https://en.wikipedia.org/wiki/Donald\\_Trump](https://en.wikipedia.org/wiki/Donald_Trump). [Online; accessed 12-September-2024].
- [14] Stephen hawking, 2024. URL [https://en.wikipedia.org/wiki/Stephen\\_Hawking](https://en.wikipedia.org/wiki/Stephen_Hawking). [Online; accessed 12-September-2024].
- [15] Albert einstein, 2024. URL [https://en.wikipedia.org/wiki/Albert\\_Einstein](https://en.wikipedia.org/wiki/Albert_Einstein). [Online; accessed 12-September-2024].
- [16] Cristiano ronaldo, 2024. URL [https://en.wikipedia.org/wiki/Cristiano\\_Ronaldo](https://en.wikipedia.org/wiki/Cristiano_Ronaldo). [Online; accessed 12-September-2024].
- [17] Lionel messi, 2024. URL [https://en.wikipedia.org/wiki/Lionel\\_Messi](https://en.wikipedia.org/wiki/Lionel_Messi). [Online; accessed 12-September-2024].
- [18] Michael jackson, 2024. URL [https://en.wikipedia.org/wiki/Michael\\_Jackson](https://en.wikipedia.org/wiki/Michael_Jackson). [Online; accessed 12-September-2024].
- [19] Taylor swift, 2024. URL [https://en.wikipedia.org/wiki/Taylor\\_Swift](https://en.wikipedia.org/wiki/Taylor_Swift). [Online; accessed 12-September-2024].
- [20] Jeff bezos, 2024. URL [https://en.wikipedia.org/wiki/Jeff\\_Bezos](https://en.wikipedia.org/wiki/Jeff_Bezos). [Online; accessed 12-September-2024].
- [21] OpenAI. ChatGPT: Optimizing Language Models for Dialogue, 2024. URL <https://openai.com/research/chatgpt>. [Online; accessed 12-September-2024].

- [22] Steven Bird, Ewan Klein, and Edward Loper. Nltk documentation: Natural language toolkit, 2009. <https://www.nltk.org>.
- [23] Languagepatternanalyzer, 2024. URL [https://github.com/AkhilJoshi15/Language\\_Pattern\\_Analyzer](https://github.com/AkhilJoshi15/Language_Pattern_Analyzer). [Online; accessed 25-October-2024].