

My Language Model

User Manual

Run MyLanguageModel class.

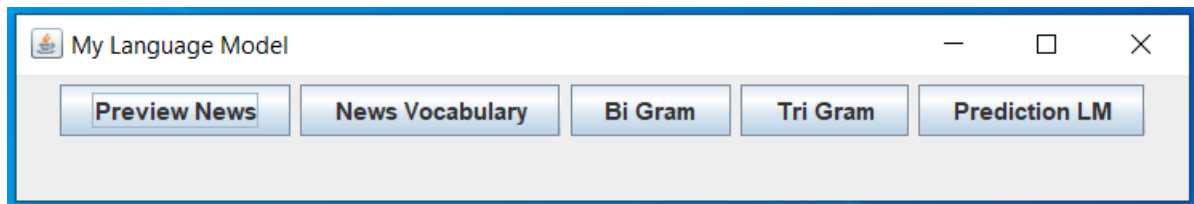


Figure 1.1: Main Screen

Click "Preview News" for a sample data view.

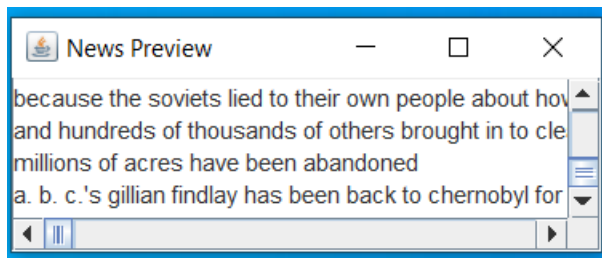
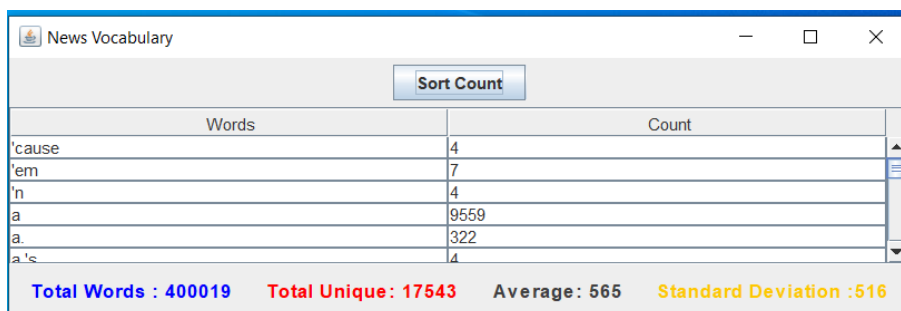


Figure 1.2: Preview News[5]

Load news vocabulary and stats with the "News Vocabulary" button, then sort by count with "Sort Count".



| Sort Count | |
|------------|-------|
| Words | Count |
| 'cause | 4 |
| 'em | 7 |
| 'n | 4 |
| a | 9559 |
| a. | 322 |
| a's | 4 |

Total Words : 400019 Total Unique : 17543 Average : 565 Standard Deviation : 516

Figure 1.3: News Vocabulary

The Main screen offers easy access to other data models. To switch to the Bi Gram or Tri Gram model, click their respective buttons. The Loader will typically disappear within 130 seconds.

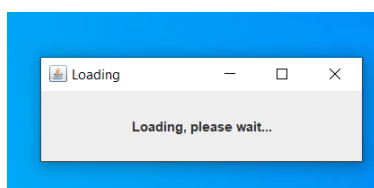


Figure 1.4: Loader

Open prediction screen and use n-grams to forecast most likely words: click "Prediction LM".

Button click selects model.

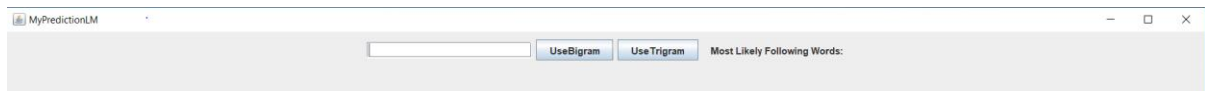


Figure 1.5: My Prediction LM

Java construct

My classes incorporate object-oriented programming paradigms such as data abstraction, encapsulation, inheritance, and polymorphism. Additionally, they show usage of the Collections Framework, GUI, and exception handling.

Data Abstraction: Data abstraction is shown using classes and interfaces. For instance, the `MyLinkedInterface` and `MyHashFunction` serve as interface and abstract class that define a contract for what a class can do, without specifying how it does.

Encapsulation: Encapsulation is achieved by declaring fields as private and providing public getter and setter methods. For example, the `VocabularyObject` class encapsulates the word, count, and probability fields, and provides getter and setter methods to access and modify these fields.

Inheritance: Inheritance is demonstrated using subclasses. The `MyHashImplementation` class extends the `MyHashFunction` abstract class, inheriting its abstract methods and providing specific implementations for them.

Polymorphism: Polymorphism is shown in the `MyHashTable` class, where the `calcHashDivision` and `calcHashFolding` methods in the `MyHashImplementation` class override the abstract methods in the `MyHashFunction`.

Collections Framework: The Collections Framework is used in the `MyHashTable` class and other classes for managing collections of objects. For example, an `ArrayList` is used to store the buckets in the hash table, and a `HashSet` is used to keep track of unique hash indices.

GUI: The Java Swing library is used for creating the GUI in the `MyLanguageModel`, `NewsFilePreview`, `BiGramLM`, `TriGramLM` and `NGramPrediction` classes. For example, `JFrame`, `DefaultTableModel`, `JButton`, `JTextField`, and `JLabel` are used to create windows, tables, buttons, text fields, and labels respectively[3] [4].

Exception Handling: Exception handling is used in multiple classes to catch and manage potential errors. Specific examples include:

Catching `NullPointerException` and `StackOverflow` in `processWord` method.

Catching `IOException` in `readFile` method. Catching `NullPointerException` and `ArrayIndexOutOfBoundsException` in `getMostLikelyNextWordsBigram` and `getMostLikelyNextWordsTriGram` methods[7].

Task 1

MyLinkedObject: This Inner class implements MyLinkedInterface and represents a node in a linked list of words. **Key properties:** word: The word stored in the node. - count: The number of times the word appears. - next: The next node in the list. **Key methods:** setWord(String w): Adds a word to the list, handling potential collisions in the hash table. processWord(MyLinkedObject current, String w): Recursively inserts a word into the linked list in alphabetical order[2]. hashTableReallocate(): Doubles the size of the hash table if it becomes too full(70%). fetchVocabularyList(): Creates and returns a VocabularyListResultWrapper containing the vocabulary list and its standard deviation. - setNGramWord(String w): Adds a word specifically for n-gram processing, adjusting hash table size if needed.

Task 2

MyHashImplementation: This inner class extends MyHashFunction, providing concrete implementations of the hash functions. It keeps track of the current hash table size. **Key methods:** calcHashDivision(String hashInput): Calculates a hash value by taking the first character's code point and dividing it by the hash table size. calcHashFolding(String hashInput): Divides the input string into segments (2 characters in this implementation). Sums up the character codes within each segment. Takes the modulo of the final sum with the hash table size to get the hash value[8].

$$h(K) = k \bmod M$$

Here,

k is the key value, and

M is the size of the hash table.

$$k = k_1, k_2, k_3, k_4, \dots, k_n$$

$$s = k_1 + k_2 + k_3 + k_4 + \dots + k_n$$

$$h(K) = s$$

Here,

s is obtained by adding the parts of the key k

Figure 1.6: Hash Division formula

Figure 1.7: Hash Folding formula

Task 3

MyHashTable: The MyHashTable class serves as the cornerstone of a vocabulary system designed to efficiently store, manage, and retrieve words, their frequencies, and n-gram occurrences. It employs a custom Hash table data structure using Linked objects to achieve this. The table is initialized with a size of 257, a prime number chosen to promote a more uniform distribution of words within the hash table. To foster modularity and maintainability, The MyHashTable class encapsulates its implementation details, shielding external tasks from the intricacies of the underlying data structures and hash calculations. This design offers a streamlined interface for interaction. The MyHashTable class adeptly employs linked lists to address collisions, ensuring that multiple words can be stored within the same hash bucket without compromising retrieval efficiency. To maintain optimal performance, the hash table dynamically doubles its size when occupancy surpasses 70%, averting potential performance degradation due to excessive collisions[1].

Task 4

VocabularyList

1. The code employs a comparator to arrange VocabularyObject instances based on their frequencies. This places words with higher frequencies at the top of the list.
2. Certain words naturally occur more frequently than others, reflecting their importance in the text. The list sorted by frequency reveals these patterns.
3. The most frequent words often highlight recurring topics within the text.
4. Comparing the total number of words to the number of unique words provides insight into vocabulary diversity.

Hash folding promotes a more balanced placement of keys within hash buckets. In contrast to hash division, which can sometimes lead to clumping of keys when patterns in the data cause collisions, hash folding offers a technique to achieve a more even spread.

| Iteration | Size(m) | Hash function | Average | Standard deviation |
|-----------|---------|---------------|---------|--------------------|
| 1 | 26 | Hash folding | 674 | 27 |
| 2 | 26 | Hash division | 674 | 495 |
| 3 | 257 | Hash folding | 68 | 10 |
| 4 | 257 | Hash division | 565 | 516 |

Table 1.1: Statistics – LinkedList Distribution

Sorting linked lists by decreasing word occurrences in NLP tasks elevates importance by prioritizing common words. This mimics the way humans comprehend language by emphasizing frequently used words to construct meaning. As a result, it holds the potential to develop language processing algorithms that are more intuitive and closely aligned with the understanding of natural language.

Task 5

N - Gram LM

Unigram probabilities are determined by dividing the count of each word by the total unigram count. In the case of bigrams, the probability is computed by dividing the count of each bigram by the count of its first word (retrieved from the unigram list). Likewise, for trigrams, the probability is calculated by dividing the count of each trigram by the count of the bigram formed by its initial two words (found in the bigram list). Following the probability calculations, candidate words are constrained to those present in either the bigrams (for the `getMostLikelyNextWordsBigram()` method) or trigrams (for the `getMostLikelyNextWordsTriGram()` method). Subsequently, the word with the highest probability among the remaining candidates is identified[6].

N Gram Prediction LM output-

1. **“you have”:** Unigram and Bigram: to a been the an no any you not some had this it that done in come their found already

2. **“you have”: Trigram:** to a any an been the for not one this two your and done engler it no now people some
3. **“this is one”: Unigram and Bigram:** of hundred thousand thing that in day and is point time half to way year on percent dollar for who
4. **“this is one”: Trigram:** of hundred thing that dollar aspect bureaucrats day eight in interview line more other part percent place reason sided two
5. **“today in sheffield”: Uni /Bi /Trigram:** not found / cannot predict as Wk-1/ Wk-2 is not present in the provided news data.
6. **“in sheffield today”: Uni and Bigram:** able to predict since Wk-1 is found whereas **Trigram** is not predicting since $p(wk|wk-2, wk-1)$ will always be 0.

Issues of implementing n grams where $n > 3$:

1. Storing larger n-gram models can consume more memory, which can be a limitation for systems with limited resources.
2. Language evolves over time, with new words and phrases emerging and old ones becoming obsolete. Larger n-grams might struggle to keep up with these changes, as they rely more on historical patterns in the data compared to unigrams, bigrams or trigrams[9].

Bibliography

- [1] GeeksforGeeks. "Implementing our own hash table with separate chaining in Java." [Online]. Available: <https://www.geeksforgeeks.org/implementing-our-own-hash-table-with-separate-chaining-in-java/>
- [2] Oracle. "Java Platform, Standard Edition (SE) 17 Documentation: java.lang.String." [Online]. Available: <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html>
- [3] Oracle. "Swing GUI Components." [Online]. Available: <https://docs.oracle.com/javase/tutorial/uiswing/components/index.html>
- [4] Oracle. "Event Handling in Swing." [Online]. Available: <https://docs.oracle.com/javase/tutorial/uiswing/events/index.html>
- [5] Baeldung. "Reading Files in Java." [Online]. Available: <https://www.baeldung.com/reading-file-in-java>
- [6] Oracle. "StringBuilder Class." [Online]. Available: <https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html>
- [7] Oracle. "Lesson: Exceptions." [Online]. Available: <https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>
- [8] GeeksforGeeks. "Hash Functions and List types of Hash Functions." [Online]. Available: <https://www.geeksforgeeks.org/hash-functions-and-list-types-of-hash-functions/>
- [9] Scaler. "N-grams in NLP." [Online]. Available: <https://www.scaler.com/topics/nlp/ngrams-in-nlp/>