Utilizing Vector Spaces for Optimizing Pattern Search in Wordle

Qodri Azkarayan – 13523010[1,2]
Bachelor's Program in Informatics Engineering
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Ganesha Street No. 10, Bandung
[1]13523010@mahasiswa.itb.ac.id, [2]azkarayan05@gmail.com

***Abstract*—This paper explores the application of vector space models and cosine similarity in optimizing the Wordle game. By leveraging character-level features, we represent five-letter words as multidimensional vectors and iteratively refine candidate guesses based on feedback-driven elimination. The methodology combines principles from natural language processing and computational game theory, showcasing an efficient strategy for solving Wordle puzzles by integrating vectorized word representations and systematic elimination processes. Experimental results demonstrate the algorithm's effectiveness, narrowing down the solution space with high accuracy and minimal iterations. The findings highlight the broader potential of these techniques for optimization tasks in other word-based games and natural language processing applications.**

***Keywords*—Wordle Optimization, Vector Space Models, Cosine Similarity, Natural Language Processing (NLP), Feedback-Driven Elimination**

I. Introduction

Wordle is a popular word puzzle game combining simplicity with strategic complexity. Players are tasked with guessing a hidden five-letter word within six attempts, receiving feedback after each guess. Letters in the right position are "green," letters in the word but in the wrong position are "yellow," and letters not in the word are "grey." This feedback mechanism transforms Wordle into a constrained optimization problem, where players iteratively refine their guesses based on logical deductions and probabilistic reasoning. Despite its simple mechanics, the game's underlying complexity has drawn significant interest from linguistics and game optimization researchers.

The fundamental problem of Wordle deals with exploitation and exploration. Players must decide whether to use new letters in their guesses or to use letters they know are not in the solution but redirect them to better understand the solution's form. The choice between these two strategies is not random. Players use a kind of combinatorial reasoning to decide when to switch between the two. This reasoning is much easier for human players because they can use all sorts of shortcuts, both mental and physical (like writing down candidate letters), to get to the solution.

In this study, we proposed a new use of vector space models to optimize the pattern search process in Wordle. Using character-level features, we represent words as multidimensional vectors and evaluate their closeness using cosine similarity. This metric serves well when the aim is to find good candidates among many options. Moreover, we integrate elimination mechanisms based on the game's feedback to convergently and efficiently narrow the options to the target word.

This work draws from the literature on natural language processing (NLP) and computational game theory. Vector space representations and similarity metrics are used to model linguistic and pattern-matching tasks. The proposed methodology builds on these ideas to solve a particular instance of a combinatorial problem—specifically, the problem of creating a linguistically constrained version of Wordle. Our approach demonstrates the power of mathematical modeling in combinatorics and the potential for these projects to illuminate the structure of games played with words.

This paper is structured as follows: Related research on the optimization of Wordle and vector space modeling is reviewed in Section 2. The methodology is addressed in Section 3, with the construction of vector representations, feedback-based elimination processes, and iterative optimization algorithms in the limelight. Section 4 presents the experimental setup and results, which evaluate the performance of the proposed approach against baseline strategies. Section 5 discusses the implications of the findings at length, with the conversation centering on the potential applications to other word games and natural language processing tasks. Finally, Section 6 serves as the conclusion, summarising the contributions made and looking forward to discussions of future research directions.

This study illustrates the practical utility of vector space models, especially cosine similarities, in addressing complex decision-making problems in a popular word game. Moreover, the findings extend beyond Wordle, offering insights into applying mathematical and computational techniques in fields requiring iterative optimization and constrained problem-solving.

II. Literature Review

A. Vector Space

A vector space is a mathematical structure consisting of objects called vectors, which can be added together and scaled by numbers known as scalars, typically real or complex numbers, depending on the context. Formally, a vector space V over a field F (e.g., the field of real numbers R) is defined by two operations: vector addition, which combines two vectors to produce a third vector.

$$V \times V \to V$$

Also, scalar multiplication which scales a vector by a scalar from the field.

$$F \times V \to V$$

These operations are governed by specific axioms that ensure the structural integrity of the vector space (Axler, 2015)[1].

To qualify as a vector space, a set V and a field F must follow certain rules known as axioms. These include closure under addition and scalar multiplication, meaning any two vectors added or a vector scaled by a scalar result in another vector in V. Addition must be commutative and associative, and a zero vector in V must act as an identity for addition. Each vector must have an additive inverse, ensuring the sum of a vector and its inverse equals zero. Scalar multiplication must also be distributed over vector addition and field addition, and it must be associative. Lastly, there must be an identity scalar in F such that multiplying a vector by this scalar leaves it unchanged. These axioms ensure the consistent behavior of vectors and scalars in a vector space.

B. *Wordle*

Wordle is a word-based puzzle game that has gained widespread popularity for its simplicity and intellectual challenge. Players aim to guess a hidden five-letter word within six attempts, receiving feedback after each guess. This feedback is provided through color-coded tiles: green indicates a correct letter in the correct position, yellow signals a correct letter in the wrong position, and gray signifies a letter not present in the target word. The game is played on a 5x6 grid where players input guesses row by row, refining their attempts based on the feedback provided. The objective is to deduce the target word as efficiently as possible using logical reasoning and vocabulary knowledge.

Wordle can be analyzed as a constraint satisfaction problem (CSP), where each guess introduces constraints that narrow the solution space by eliminating invalid candidates. Players face the challenge of balancing exploration, testing new letters, exploitation, and refining knowledge based on feedback (Shi & Chen, 2023)[2]. The game also lends itself to analysis through information theory principles. Optimal guessing strategies aim to maximize information gained from each guess, thereby reducing the entropy of the solution space. Early guesses, for instance, often target high-frequency letters and diverse patterns to eliminate large subsets of possibilities.

Regarding computational complexity, Wordle's design maintains a manageable search space. Although a typical English dictionary contains around 12,000 five-letter words, the game reduces the number of valid target words to around 2,300 commonly used ones. This limited vocabulary ensures the game is solvable within the six-guess constraint.

From a linguistic and cognitive perspective, Wordle engages players' vocabulary knowledge, pattern recognition, and phonetic reasoning. It also involves cognitive strategies such as hypothesis testing and deductive reasoning. These elements make Wordle valuable for studying human problem-solving and decision-making processes.

C. *CountVectorizer*

The CountVectorizer from the scikit-learn library is a tool in natural language processing (NLP) that converts textual data into a numerical format suitable for machine learning models (ScikitLearn)[3]. Depending on the specified configuration, it creates a "bag of words" representation by tokenizing the input text into smaller units, such as characters or words. For example, when configured with analyzer="char" and ngram_range=(1, 1), the vectorizer tokenizes the text at the character level, extracting individual characters as features. This configuration is particularly useful for applications like spelling correction, phonetic analysis, or games like Wordle, where character-level patterns play a crucial role. The `ngram_range` parameter further allows customization of token sizes, enabling the analysis of sequences of one or more characters. The output of `CountVectorizer` is a sparse matrix where rows correspond to text samples, columns represent tokens, and values denote the frequency of each token in the respective text. This numerical representation provides a foundation for applying machine learning or computational techniques to textual data, making `CountVectorizer` an essential preprocessing tool in NLP pipelines.

After processing the input text, the CountVectorizer produces a sparse matrix that encodes the frequency of tokens extracted from the text. Each row in the matrix corresponds to an individual text sample, such as a word from the vocabulary, while each column represents a unique token—in this case, individual characters. The matrix values indicate each token's frequency within the corresponding text sample. For instance, given a vocabulary of ["pace", "space", “peace”], the resulting matrix would be like this

	s	p	a	c	e
pace	0	1	1	1	1
space	1	1	1	1	1
peace	0	1	1	1	2

Table 1. CountVectorizer Representation of Vocabulary Words

D. *Cosine Simmilarity*

Cosine Similarity is a metric that measures the similarity between two vectors in a multidimensional space. It is often applied in text analysis and natural language processing tasks. In the context of a literature review, cosine similarity helps identify relationships or thematic overlaps between academic papers, books, or other textual resources. This is particularly useful when comparing abstracts, keywords, or full texts of documents. Mathematically, cosine similarity is defined as the cosine of the angle between two vectors. Given two vectors, A and B, the formula is:

$$\cos(\theta) = \frac{A \cdot B}{|A||B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

One of its key features is its range, which spans from −1 to 1. A value of 1 indicates perfect similarity, where the vectors are aligned in the same direction. A value of 0 signifies no similarity, as the vectors are orthogonal, and −1 indicates that the vectors are opposed. Another significant feature is that cosine similarity is scale-invariant, meaning it is unaffected by the magnitude of the vectors. This makes it particularly valuable in applications like text analysis, where the focus is on the direction and not the magnitude of document vectors.

In terms of applications, cosine similarity is widely used in various domains. Text analysis measures the similarity between documents or terms represented in high-dimensional spaces, such as TF-IDF vectors. It is also instrumental in recommender systems, where it helps identify items similar to a user's preferences. Furthermore, cosine similarity serves as a distance measure in clustering and classification algorithms, aiding in grouping or categorizing data based on similarity.

Turney (2001)[4] demonstrated the efficacy of cosine similarity in determining semantic relationships through Latent Semantic Analysis (LSA) for synonym recognition and analogy tasks. This work established cosine similarity as a cornerstone for relational analysis in NLP.

Levy and Goldberg[5] (2014) extended the use of cosine similarity to evaluate word embeddings produced by neural models. They compared cosine similarity with other metrics, reinforcing its robustness in tasks like analogy completion and clustering.

E. NLTK words

The Natural Language Toolkit (NLTK) is a widely recognized Python library for natural language processing (NLP) tasks. It provides a comprehensive suite of tools for tokenization, stemming, tagging, parsing, and more, simplifying complex text processing workflows for researchers, developers, and educators. Among its standout features is the words module, which grants access to extensive word lists and linguistic corpora. These resources are pivotal for numerous NLP applications, including stopword removal, stemming, spell-checking, and generating linguistic annotations. The words module integrates seamlessly with tools like WordNet, a lexical database of English, enabling sophisticated semantic analyses such as measuring word similarity and exploring hierarchical relationships like synonyms, antonyms, and hypernyms. For instance, WordNet organizes words into synsets (sets of synonyms) and provides their definitions and contextual examples, enhancing the analysis of semantic relationships in textual data (Perkins, 2014)[6].

NLTK's capabilities have been extensively utilized in both academic and industrial contexts. The words module and NLTK's tokenization and text parsing tools have been employed in spell-checking systems, sentiment analysis, and text classification. It is particularly valuable in tasks requiring preprocessing, such as removing stopwords, identifying parts of speech, and reducing words to their base forms via stemming or lemmatization. Additionally, its integration with WordNet enables advanced applications like question answering, keyword extraction, and building semantically aware models for chatbots and virtual assistants. As a result, NLTK has become a cornerstone in NLP education and practice, facilitating diverse applications in fields such as information retrieval, sentiment analysis, and computational linguistics.

F. Feedback-Based Optimization Strategies

Feedback-driven elimination algorithms play a crucial role in optimizing Wordle gameplay, as they systematically narrow down the set of possible solutions based on the feedback provided for each guess. By leveraging the feedback, such algorithms iteratively refine the set of valid candidate words, improving the efficiency of future guesses.

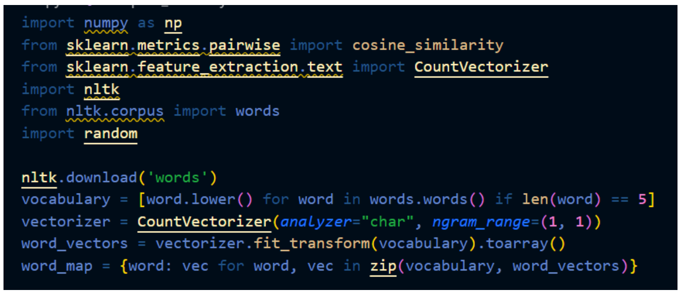
Wang et al.[7] (2019) contributed significantly to this domain by proposing integrating feedback-driven elimination with advanced similarity metrics and visual optimization techniques. Their approach involved tailoring Wordle-like puzzles using shape-aware designs, which account for word representations' visual and structural properties. This framework improved the computational efficiency of solution algorithms and introduced visual feedback mechanisms to aid in understanding the elimination process. The method effectively highlighted the interplay between algorithmic optimization and user interaction by aligning visual cues with computational decisions.

III. Implementation

A program was implemented using Python to develop a more efficient solution for the Wordle puzzle game. The program leverages natural language processing (NLP) libraries to create a robust method for iteratively narrowing down possible solutions based on feedback. It combines linguistic resources, such as the NLTK words corpus, with machine learning tools like cosine_similarity to evaluate and refine guesses systematically.

The program begins by importing the necessary libraries, including nltk, numpy, and Scikit-learn modules. The nltk.corpus.words dataset is used to create a list of valid five-letter words in English. After downloading the corpus, all words are converted to lowercase, and those with lengths other than five characters are filtered out. This ensures that the vocabulary aligns with the constraints of the Wordle game.

To prepare these words for numerical analysis, the program utilizes the CountVectorizer from Scikit-learn. Configured with analyzer="char" and ngram_range=(1, 1), the vectorizer tokenizes the words into individual characters, creating a matrix where each row corresponds to a word, and each column represents a character's frequency in that word. This transformation converts the vocabulary into a format suitable for similarity calculations. A dictionary, word_map, is then created to map each word to its corresponding vector representation, facilitating efficient access during subsequent computations.



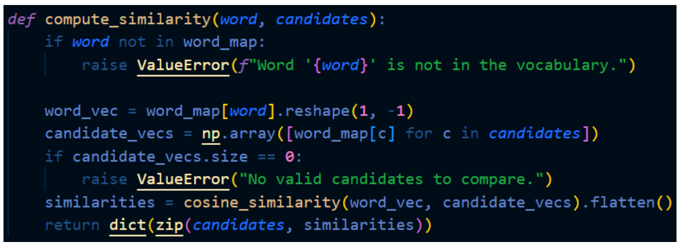
```python
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
import nltk
from nltk.corpus import words
import random

nltk.download('words')
vocabulary = [word.lower() for word in words.words() if len(word) == 5]
vectorizer = CountVectorizer(analyzer="char", ngram_range=(1, 1))
word_vectors = vectorizer.fit_transform(vocabulary).toarray()
word_map = {word: vec for word, vec in zip(vocabulary, word_vectors)}
```

Figure 1. Implementation of Vocabulary Preparation
Source: https://github.com/qodriazka/Vector-Space-in-Wordle

To guide the guessing process, the program incorporates a function called compute_similarity. This function calculates the cosine similarity between a given word and a list of candidate words. Cosine similarity measures the closeness of two vectors by computing the cosine of the angle between them. It is particularly useful in this context because it quantifies how similar the letter composition of two words is, regardless of their overall frequency or magnitude.

The function first checks if the given word exists in the vocabulary. It then extracts the vector representation of the word and compares it with the vectors of all candidate words. The resulting similarity scores are returned as a dictionary, where each candidate word is paired with its similarity score. These scores are later used to rank the candidates, ensuring the next guess is as informed as possible.

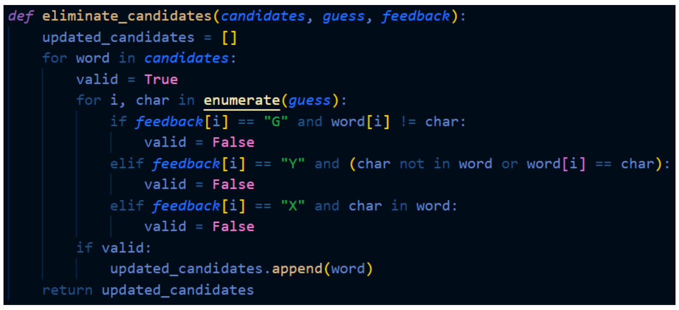


```python
def compute_similarity(word, candidates):
    if word not in word_map:
        raise ValueError(f"Word '{word}' is not in the vocabulary.")

    word_vec = word_map[word].reshape(1, -1)
    candidate_vecs = np.array([word_map[c] for c in candidates])
    if candidate_vecs.size == 0:
        raise ValueError("No valid candidates to compare.")
    similarities = cosine_similarity(word_vec, candidate_vecs).flatten()
    return dict(zip(candidates, similarities))
```

Figure 2. Implementation of Similarity Computation
Source: https://github.com/qodriazka/Vector-Space-in-Wordle

The program's decision-making process relies on its ability to interpret Wordle feedback, handled by the eliminate_candidates function. This function iterates through the list of candidate words, filtering out those that do not comply with feedback rules represented by a five-character string: "G" (green) for correct letters in the correct position, "Y" (yellow) for correct letters in the wrong position, and "X" (grey) for letters not present in the word. For each candidate word, the function ensures it meets these constraints, discarding those that violate the rules. This process progressively narrows the candidate pool, sharpening the program's focus for the next iteration.



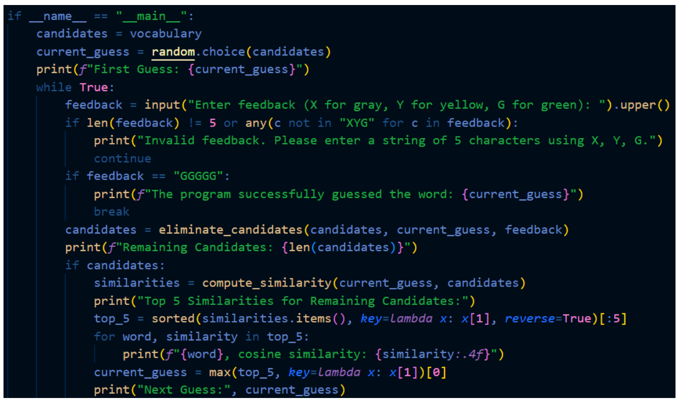
```python
def eliminate_candidates(candidates, guess, feedback):
    updated_candidates = []
    for word in candidates:
        valid = True
        for i, char in enumerate(guess):
            if feedback[i] == "G" and word[i] != char:
                valid = False
            elif feedback[i] == "Y" and (char not in word or word[i] == char):
                valid = False
            elif feedback[i] == "X" and char in word:
                valid = False
        if valid:
            updated_candidates.append(word)
    return updated_candidates
```

Figure 3. Implementation of Candidate Elimination Based on Feedback
Source: https://github.com/qodriazka/Vector-Space-in-Wordle

The main section of the program begins by initializing the candidate pool to include all words from the prepared vocabulary. The first guess is chosen randomly from this list, and the user is prompted to enter feedback based on the game's response to this guess. Feedback validation ensures that the input matches the expected format of five characters using only X, Y, and G.

If the feedback indicates that the guess is correct (all G), the program announces success and exits. Otherwise, it calls the eliminate_candidates function to update the candidate list based on the feedback. Next, the program uses the compute_similarity function to rank the remaining candidates by their similarity to the current guess. It selects the word with the highest similarity score as the next guess.

This iterative process continues, with each step refining the candidate pool and improving the guesses based on feedback and similarity scores. The program also provides diagnostic information, such as the number of remaining candidates and the top five most similar words, allowing users to understand the decision-making process.

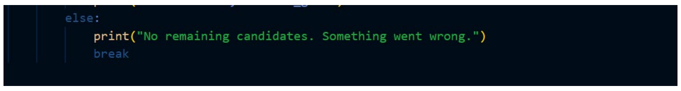


```python
if __name__ == "__main__":
    candidates = vocabulary
    current_guess = random.choice(candidates)
    print(f"First Guess: {current_guess}")
    while True:
        feedback = input("Enter feedback (X for gray, Y for yellow, G for green): ").upper()
        if len(feedback) != 5 or any(c not in "XYG" for c in feedback):
            print("Invalid feedback. Please enter a string of 5 characters using X, Y, G.")
            continue
        if feedback == "GGGGG":
            print(f"The program successfully guessed the word: {current_guess}")
            break
        candidates = eliminate_candidates(candidates, current_guess, feedback)
        print(f"Remaining Candidates: {len(candidates)}")
        if candidates:
            similarities = compute_similarity(current_guess, candidates)
            print("Top 5 Similarities for Remaining Candidates:")
            top_5 = sorted(similarities.items(), key=lambda x: x[1], reverse=True)[:5]
            for word, similarity in top_5:
                print(f"{word}, cosine similarity: {similarity:.4f}")
            current_guess = max(top_5, key=lambda x: x[1])[0]
            print("Next Guess:", current_guess)
```

Figure 4. Implementation of Main Algorithm (Iterative guess)
Source: https://github.com/qodriazka/Vector-Space-in-Wordle

The program includes safeguards to handle scenarios where no valid candidates remain, which may occur due to incorrect feedback or limitations in the vocabulary. In such cases, it terminates gracefully with an error message, prompting the user to review the feedback or input.

This implementation demonstrates how computational methods can be applied to solve Wordle systematically. By combining linguistic resources, vector-based similarity measures, and feedback-driven elimination, the program exemplifies the use of NLP and machine learning techniques to address real-world problems in an efficient and interpretable manner.



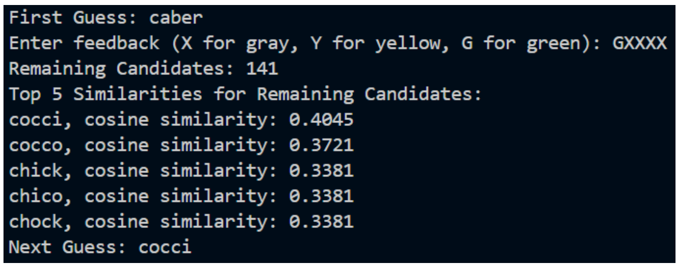
```
    else:
        print("No remaining candidates. Something went wrong.")
        break
```

Figure 5. Implementation of Error Handling
Source: https://github.com/qodriazka/Vector-Space-in-Wordle

IV. Experiment

A. *Test Case 1*

This test case illustrates the systematic and computationally guided application of the Wordle-solving algorithm. The algorithm utilizes feedback-driven elimination, cosine similarity measures, and an iterative guessing approach to identify the correct word efficiently. For this test, the correct answer is “CLOUD.” Below is a scientific breakdown of the process and its outcomes.



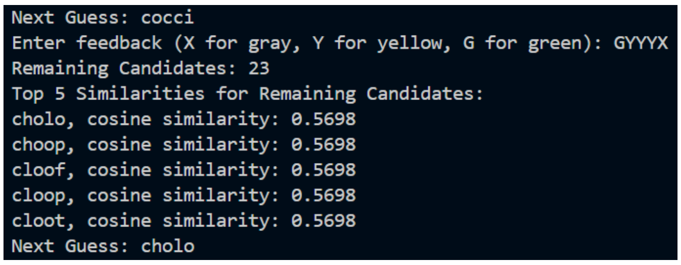
```
First Guess: caber
Enter feedback (X for gray, Y for yellow, G for green): GXXXX
Remaining Candidates: 141
Top 5 Similarities for Remaining Candidates:
cocci, cosine similarity: 0.4045
cocco, cosine similarity: 0.3721
chick, cosine similarity: 0.3381
chico, cosine similarity: 0.3381
chock, cosine similarity: 0.3381
Next Guess: cocci
```

Figure 6. First Guess in Test Case

The program begins with an initial guess of "CABER," chosen randomly from the vocabulary. The feedback entered is GXXXX, indicating that the first letter "c" is correct and positioned correctly, while the other letters are not part of the target word.

Following this, the candidate elimination process applies feedback rules to exclude all words that do not start with "c" or contain any of the letters "a," "b," "e," or "r" in any position, reducing the pool of candidates to 141 possibilities.

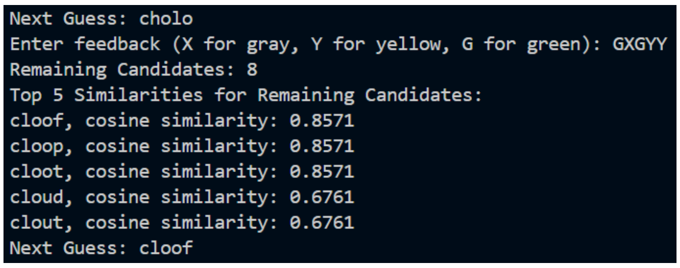
To refine the selection further, cosine similarity is calculated between the vector representation of "CABER" and the remaining candidates. The top five words are identified based on similarity scores, including "COCCI" with a score of 0.4045. Among these, "COCCI," the word with the highest similarity score, is selected as the next guess.



```
Next Guess: cocci
Enter feedback (X for gray, Y for yellow, G for green): GYYYX
Remaining Candidates: 23
Top 5 Similarities for Remaining Candidates:
cholo, cosine similarity: 0.5698
choop, cosine similarity: 0.5698
cloof, cosine similarity: 0.5698
cloop, cosine similarity: 0.5698
cloot, cosine similarity: 0.5698
Next Guess: cholo
```

Figure 7. Second Guess in Test Case

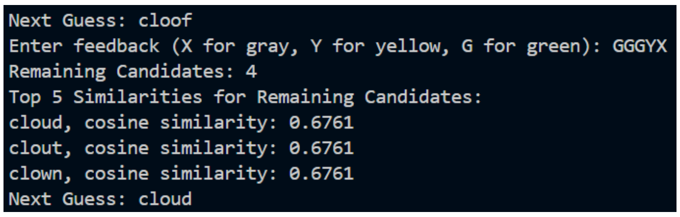
The second guess, "COCCI," receives feedback GYYYX, indicating that the first letter "c" is correct and in the correct position, the second and third letters "o" and "c" are correct but not in the correct positions, and the fifth letter "i" is not part of the target word. Following this feedback, candidate elimination excludes words starting with "c" that do not contain "o" and "c" in positions 2 and 3, as well as those containing "i," reducing the candidate pool to 23 words. Cosine similarity is then recalculated for the remaining candidates, and "cholo" is identified as the next guess with a similarity score of 0.5698, tied with several others. The selection of "CHOLO" reflects the program's systematic prioritization of equally high-scoring candidates.



```
Next Guess: cholo
Enter feedback (X for gray, Y for yellow, G for green): GXGYY
Remaining Candidates: 8
Top 5 Similarities for Remaining Candidates:
cloof, cosine similarity: 0.8571
cloop, cosine similarity: 0.8571
cloot, cosine similarity: 0.8571
cloud, cosine similarity: 0.6761
clout, cosine similarity: 0.6761
Next Guess: cloof
```

Figure 8. Third Guess in Test Case

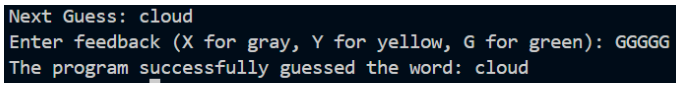
The third guess, "CHOLO," receives feedback GXGYY, indicating that the first letter "c" and the third letter "l" are correct and in their correct positions, while the fourth and fifth letters "o" are correct but mispositioned. Based on this feedback, the candidate elimination process excludes words that do not align with the updated rules, reducing the pool to 8 viable candidates. Cosine similarity is then computed for these remaining words, with "cloof," "cloop," and "cloot" emerging as the top candidates, each scoring 0.8571. Due to its high similarity score, the algorithm selects "CLOOF" as the next guess.



```
Next Guess: cloof
Enter feedback (X for gray, Y for yellow, G for green): GGGYX
Remaining Candidates: 4
Top 5 Similarities for Remaining Candidates:
cloud, cosine similarity: 0.6761
clout, cosine similarity: 0.6761
clown, cosine similarity: 0.6761
Next Guess: cloud
```

Figure 9. Third Guess in Test Case

The fourth guess, "CLOOF," yields feedback GGGYX, indicating that the first three letters "c," "l," and "o" are correct and in their correct positions, while the fifth letter "f" is not part of the target word. Following this feedback, the candidate elimination process removes words that do not meet the updated constraints, narrowing the pool to four remaining candidates: "CLOUD," "CLOUT," and "CLOWN." Cosine similarity is then recalculated, and "cloud" is selected as the next guess due to its highest similarity score of 0.6761.



```
Next Guess: cloud
Enter feedback (X for gray, Y for yellow, G for green): GGGGG
The program successfully guessed the word: cloud
```

Figure 10. Final Guess in Test Case

The fifth guess, "CLOUD," receives feedback GGGGG, indicating that all letters are correct and in position. The algorithm successfully identifies the target word in five guesses.

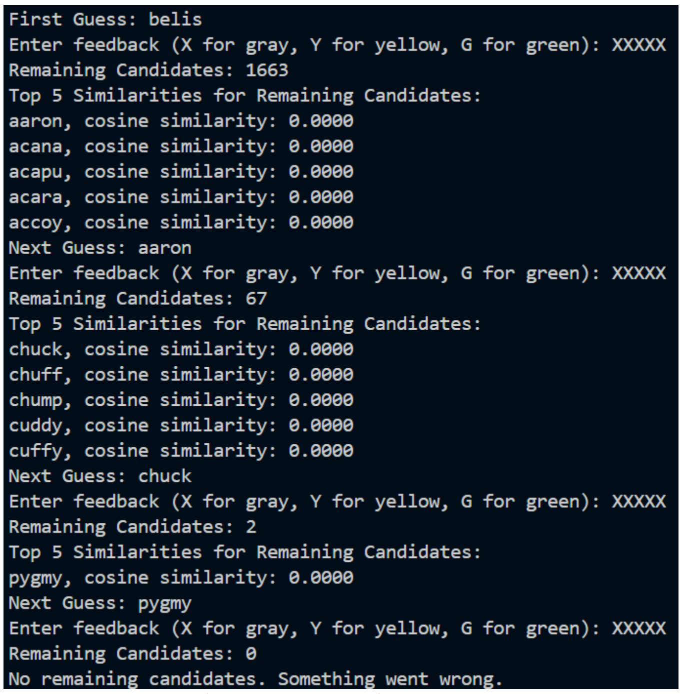
The algorithm demonstrates significant efficiency by systematically reducing the candidate pool with each iteration. Starting with 141 candidates after the first feedback, it efficiently narrows the possibilities, arriving at the correct word in just four subsequent guesses. This process combines linguistic constraints and similarity metrics to optimize the guessing strategy.

Using cosine similarity provides a quantitative approach to measure the alignment between vector representations of the current guess and the remaining candidates. This method allows the algorithm to prioritize words most likely to align with the target, effectively balancing exploration and exploitation during the guessing process.

Feedback-driven elimination translates the provided feedback into precise rules that refine the solution space dynamically. This ensures that all remaining candidates comply with the game's constraints, leading to a more focused and effective search.

Finally, the program exhibits robust performance by gracefully managing ties in similarity scores and adapting seamlessly to updated feedback. This resilience highlights its capability to handle the task's inherent uncertainty and complexity.

B. *Test Case 2*



```
First Guess: belis
Enter feedback (X for gray, Y for yellow, G for green): XXXXX
Remaining Candidates: 1663
Top 5 Similarities for Remaining Candidates:
aaron, cosine similarity: 0.0000
acana, cosine similarity: 0.0000
acapu, cosine similarity: 0.0000
acara, cosine similarity: 0.0000
accoy, cosine similarity: 0.0000
Next Guess: aaron
Enter feedback (X for gray, Y for yellow, G for green): XXXXX
Remaining Candidates: 67
Top 5 Similarities for Remaining Candidates:
chuck, cosine similarity: 0.0000
chuff, cosine similarity: 0.0000
chump, cosine similarity: 0.0000
cuddy, cosine similarity: 0.0000
cuffy, cosine similarity: 0.0000
Next Guess: chuck
Enter feedback (X for gray, Y for yellow, G for green): XXXXX
Remaining Candidates: 2
Top 5 Similarities for Remaining Candidates:
pygmy, cosine similarity: 0.0000
Next Guess: pygmy
Enter feedback (X for gray, Y for yellow, G for green): XXXXX
Remaining Candidates: 0
No remaining candidates. Something went wrong.
```

Figure 11. Second Test Case

This test case illustrates how a Wordle-solving program attempts to guess the target word based on feedback. It begins with the guess "BELIS," and feedback of "XXXXX" eliminates all words containing the letters b, e, l, i, or s, reducing the candidate pool to 1663 words. With no further information, it guesses "AARON," but feedback of "XXXXX" again excludes words with a, r, o, or n, leaving 67 candidates. The next guess, "CHUCK," also receives "XXXXX," further shrinking the pool to 2 words. The program then guesses "PYGMY," and feedback of "XXXXX" eliminates all remaining candidates. At this point, the program stops, reporting no valid candidates left, likely because the target word is not in the vocabulary or the feedback provided was inaccurate. This case proves that the Wordle-solving program uses feedback to eliminate candidates iteratively but fails to find the target word, likely due to incorrect feedback.

V. Implication

This code demonstrates a computational framework for solving Wordle, utilizing vector representations, similarity metrics, and feedback-driven elimination. Beyond Wordle, its methodology lays a foundation for optimizing strategies in other word-based games and natural language processing (NLP) tasks.

The approach is versatile and can be adapted to games like Scrabble and crosswords, where cosine similarity and elimination techniques help identify optimal moves by comparing word vectors to game constraints. Hangman's feedback-driven elimination mechanism closely aligns with Wordle’s iterative narrowing process, systematically reducing candidate words. For Boggle or word search puzzles, integrating vector representations could aid in identifying high-scoring or hidden words by evaluating character adjacency or predefined rules. Moreover, this method could inform adaptive word-based learning tools, tailoring puzzles to players' vocabulary levels for maximized engagement and educational value.

In NLP, the techniques have broad applications in pattern recognition, similarity analysis, and constrained search. Cosine similarity is valuable for tasks such as text similarity and query expansion in search engines, where algorithms suggest semantically related queries or refine results. The feedback-driven elimination algorithm could improve context-aware spelling correction by iteratively narrowing down suggestions based on valid dictionary matches. Similarly, constrained next-word prediction models, such as autocomplete systems or assistive writing tools, could benefit from this method, enhancing accuracy in text generation. Interactive systems like chatbots and virtual assistants can use feedback mechanisms to refine responses for improved conversational relevance.

Beyond games and NLP, this methodology has machine learning and data analysis implications. Cosine similarity is central to clustering algorithms, while iterative elimination can be adapted for semi-supervised clustering with user-imposed constraints. Recommender systems could employ feedback-driven refinement to personalize movie, product, or content suggestions effectively.

Expanding the implementation could involve integrating pre-trained embeddings like Word2Vec or BERT for nuanced semantic understanding or adapting the vocabulary for multilingual applications. Optimization techniques would also enhance scalability for larger datasets or complex rules.

From a scientific perspective, this framework mirrors human problem-solving, providing insights into cognitive psychology and human-computer interaction. It is also a robust teaching tool in computational linguistics and AI, illustrating fundamental concepts in vector spaces, similarity metrics, and constraint-based algorithms.

VI. Conclusion

The study presents a novel computational approach to solving Wordle by employing vector space models and cosine similarity to reduce the solution space systematically. The algorithm efficiently identifies target words through feedback-driven elimination and iterative refinement, demonstrating the robustness of mathematical and computational methods in constrained decision-making tasks. Beyond Wordle, this framework offers valuable insights for optimizing strategies in various word-based puzzles and natural language processing challenges. Future work could extend the methodology to incorporate pre-trained embeddings, multi-language support, and applications in interactive NLP systems, underscoring the versatility of this approach in solving complex linguistic and combinatorial problems.

Acknowledgment

I express my deepest gratitude to God Almighty for His endless guidance, blessings, and strength throughout the journey of completing this paper. I also thank Dr. Ir. Rinaldi Munir, M.T., for his invaluable role as the IF2123 Linear and Geometry Algebra lecturer. His guidance and comprehensive explanations have significantly contributed to my understanding of the subject matter, providing a strong foundation for this work.

References

[1] Sheldon Jay Axler, *Linear algebra done right*. Cham Etc.: Springer, Cop, 2015.

[2] L. Shi, Y. Chen, J. Lin, X. Chen, and G. Dai, "A black-box model for predicting difficulty of word puzzle games: a case study of Wordle," Knowledge and information systems, vol. 66, no. 3, pp. 1729–1750, Oct. 2023, doi: https://doi.org/10.1007/s10115-023-01992-6.

[3] "CountVectorizer," scikit-learn, 2024. https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html accessed on 31 December 2024

[4] Luc De Raedt, P. Turney, and Springerlink (Online Service, Machine Learning: ECML 2001: 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001. Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[5] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in Advances in Neural Information Processing Systems (NeurIPS), vol. 27, 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/feab05aa91085b7a8012516bc3533958-Paper.pdf

[6] J. Perkins, Python 3 Text Processing with NLTK 3 Cookbook. Packt Publishing Ltd, 2014.

[7] Y. Wang et al., "ShapeWordle: Tailoring Wordles using Shape-aware Archimedean Spirals," IEEE Transactions on Visualization and Computer Graphics, vol. 26, no. 1, pp. 991–1000, Jan. 2020, doi: https://doi.org/10.1109/TVCG.2019.2934783.

STATEMENT

Hereby, I declare that the paper I have written is my work, not an adaptation or translation of someone else's paper, and is not plagiarized.

Bandung, 1 January 2025

Qdi

Qodri Azkarayan
13523010