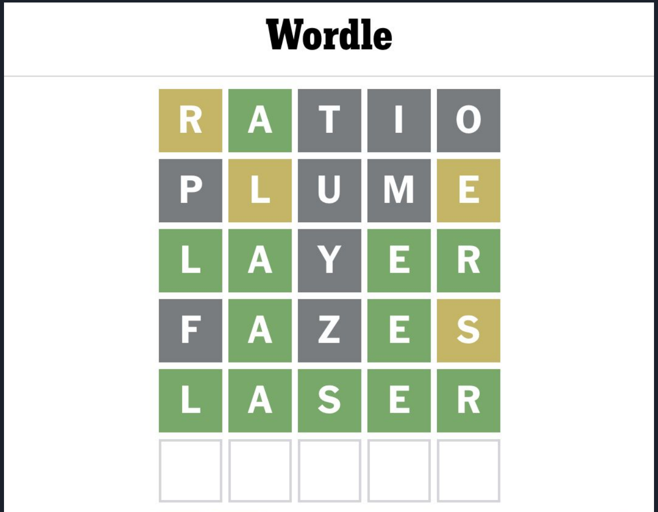
Exploring Algorithms for Optimal Play in Wordle

Max Van Fleet, Vasishta Tumuluri, Jun Ikeda

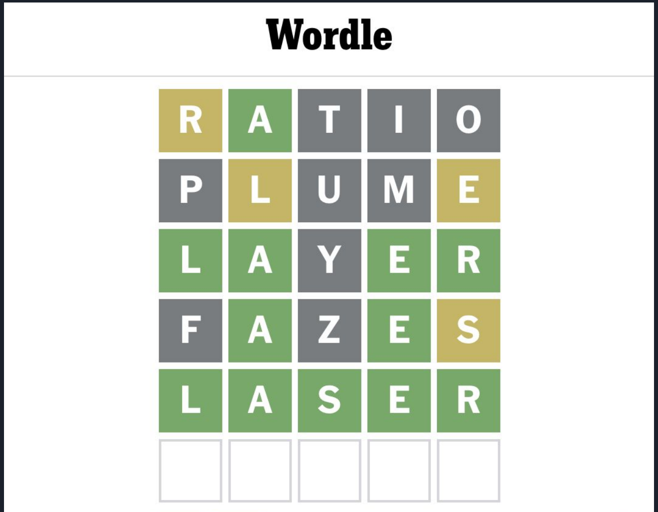
Introduction

What is Wordle?

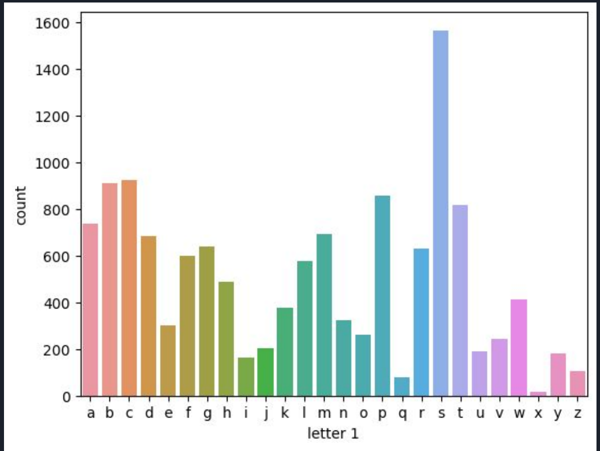


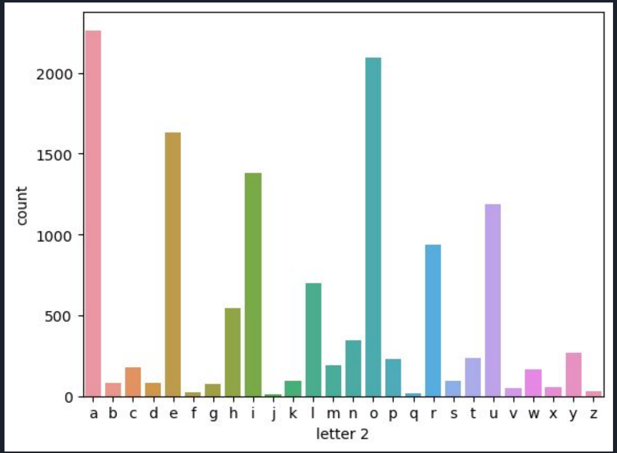
- 5 letter word guessing game with 6 attempts
- Green Tile
  - Letter is in the solution word and in the right position
- Yellow Tile
  - Letter is in the solution word but not in the right position
- Gray Tile
  - Letter is not in the solution word

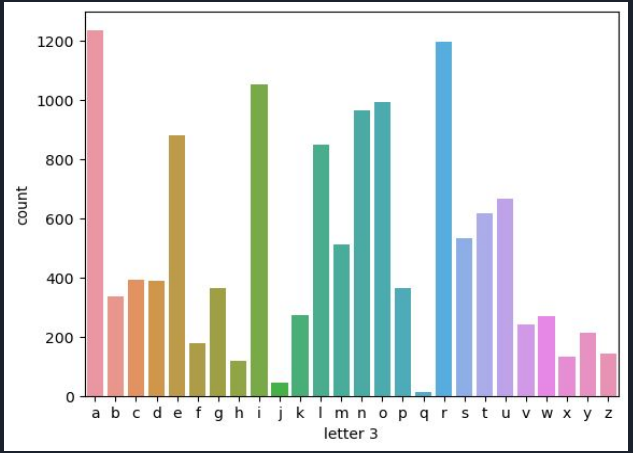
Wordle Dictionary

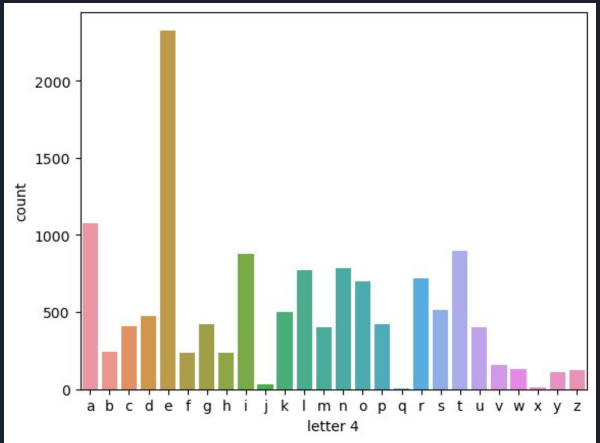


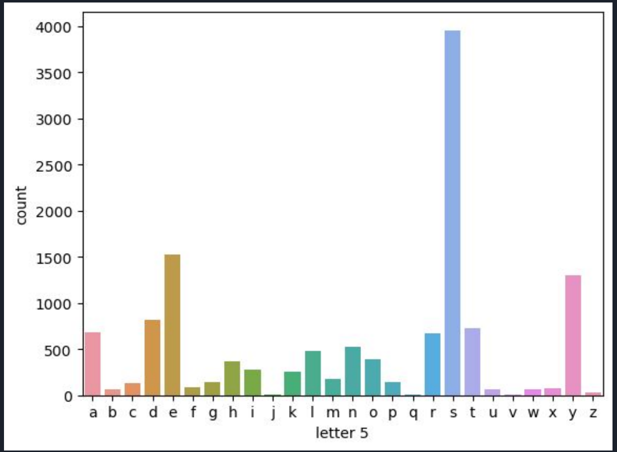
- Using Wordle Dictionary derived from the 2022 version of the game
    - 2315 possible solution words
        - S
    - 10357 exclusively guess words
    - 12672 total possible guesses
        - G


1600
1400
1200
1000
800
600
400
200
0
count
a b c d e f g h i j k l m n o p q r s t u v w x y z
letter 1


2000
1500
1000
500
0
count
a b c d e f g h i j k l m n o p q r s t u v w x y z
letter 2


1200
1000
800
600
400
200
0
count
a b c d e f g h i j k l m n o p q r s t u v w x y z
letter 3


2000
1500
1000
500
0
count
a b c d e f g h i j k l m n o p q r s t u v w x y z
letter 4


4000
3500
3000
2500
2000
1500
1000
500
0
count
a b c d e f g h i j k l m n o p q r s t u v w x y z
letter 5

Methods

Framing and Metrics

"Guess-indifferent"

- Guess-indifferent reward function:
  - Solution is guessed
    - r = 1
  - Solution not guessed
    - r = 0

"Guess-biased"

- Guess-biased reward function:
  - Solution is guessed
    - r = - (number of guesses used)
  - Solution is not guessed
    - r = -α where $\alpha \geq 7$

- We additionally consider execution time as a metric to compare different approaches

Exact Dynamic Programming Solution

- In 2022, Bertsimas and Paskov found an exact dynamic programming solution to Wordle
- This approach guarantees guessing the solution within the allotted guesses, guaranteeing a a guess-indifferent reward of 1
- On average, it takes 3.421 attempts to guess the solution, so the average guess-biased reward is -3.421
- Took several days to run their algorithm to solve the game, even with an efficient C++ implementation parallelized across a 64-core computer

Elimination Algorithm

- Starting with the set of all possible solutions S, when we make guesses, we learn from their colorings that some of the words in S can't be the solution anymore
- Need some way to determine which words are ruled out and which could still potentially be the solution
- We use elimination algorithm elim(Z, g, coloring_g); input is current set Z of all possible solutions, the guess g just made, and the coloring coloring_g of that guess
- Iterates through each word $z \in Z$ and uses the information learned from the g and coloring_g to eliminate/not eliminate z from Z
- Time complexity $O(|Z|)$ i.e. runtime is linear in the cardinality of Z

Random Algorithm

- “Gold standard of badness”
- At each stage (each time we have to make a guess), this algorithm picks a random guess from the current Z
  - i.e. at start of game, picks a random guess from S
- Then runs elimination algorithm elim(Z, g, coloring_g) to rule out words from Z before moving onto the next stage
- Linear time complexity O(|S|)

Minimax and Minex

- Cardinality of the set of remaining possible solutions could be a useful heuristic for optimality
- At each stage, minimax attempts to minimize maximum size of the next Z, and minex attempts to minimize expected size of next Z
- Must consider every possible guess in G, and for each of those, condition on every possible solution in S, to determine which guess g* in G has the least minimum/expected cardinality of next Z
- Very computationally heavy; instead we restrict guesses to solution set S, choose $\lceil |S|/200 \rceil$ random guesses from S, and condition on $\lceil |S|/200 \rceil$ random solutions in S
- Cubic time complexity $O(|S|^3)$

Representing Solutions as a Matrix

- We can convert any 5-letter word into a unique column vector of length 130 containing exactly 5 entries of 1 with remaining 125 entries 0
- For example, if we were dealing with two-letter words, the word AZ would be the following 52-component vector:

$$AZ := [1, 0, 0 \ldots, 0, 0, 1]^T$$

- We do this for the whole solution set S and concatenate them all into a (130 x |S|) matrix A
- We want to find vectors (words) in G that are “good representations” of the matrix A. How do we do this?

Application of Rank-One Approximation

- Consider the SVD $A = U\Sigma V^T$, where Σ is a diagonal matrix whose diagonal entries are the singular values of A in descending order
- Closest rank-one approximation to A is $u_1\sigma_1 v_1^T$ where σ_1 is the largest singular value of A, and u_1 and v_1 are the associated left and right singular vectors
- Therefore, u_1 can be considered the column vector that "best represents" A
- σ_1 is the largest singular value of A, so σ_1^2 must be the largest magnitude eigenvalue (called the dominant eigenvalue) of AA^T, with associated dominant eigenvector u_1 as $AA^T = U\Sigma^2U^T$ is an eigendecomposition.
- Thus, the dominant eigenvector u_1 of AA^T is the column vector that best represents A

Dominant Eigenvectors & Latent Semantic Indexing

1. Vectorize words in solution space into (130 x |Z|) matrix A
2. Find dominant eigenvector u associated with dominant eigenvalue of AA^T
3. Use cosine similarity to find the word in the action space closest to the eigenvector

$$\hat{g} = \arg\ \min\left(\theta = \ \arccos\left(\frac{u^t \cdot g}{||g|| \cdot ||u||}\right)\right)$$

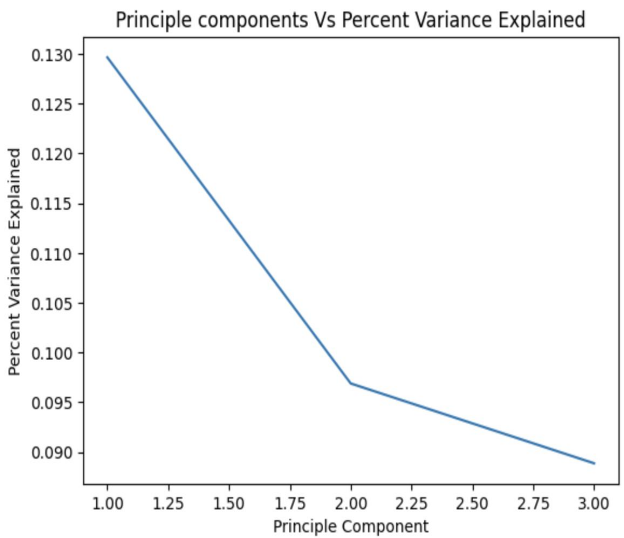
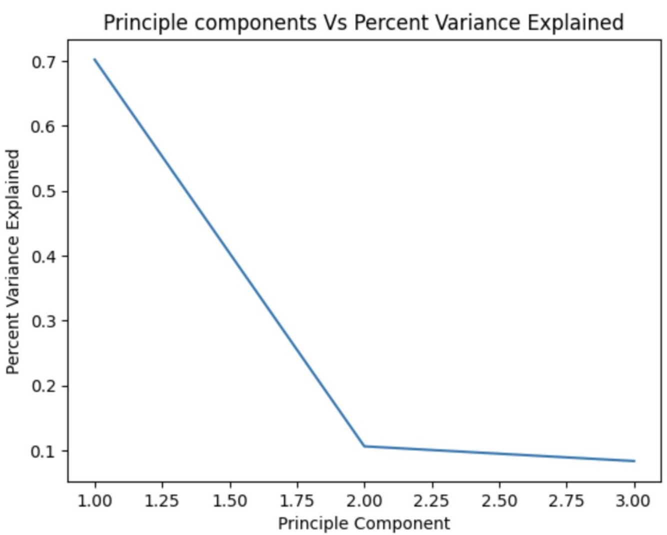
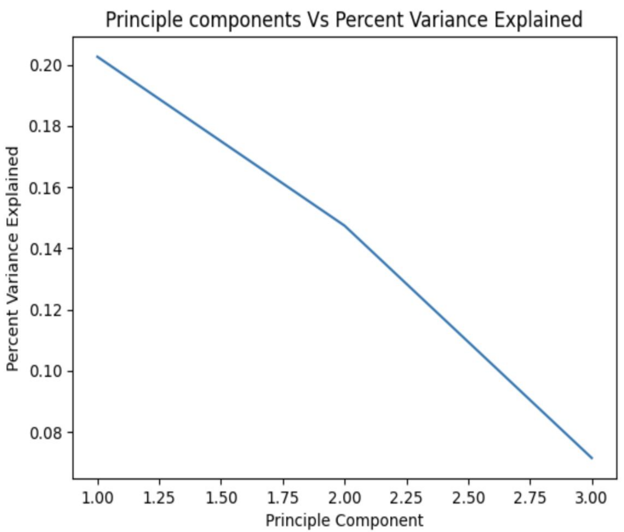
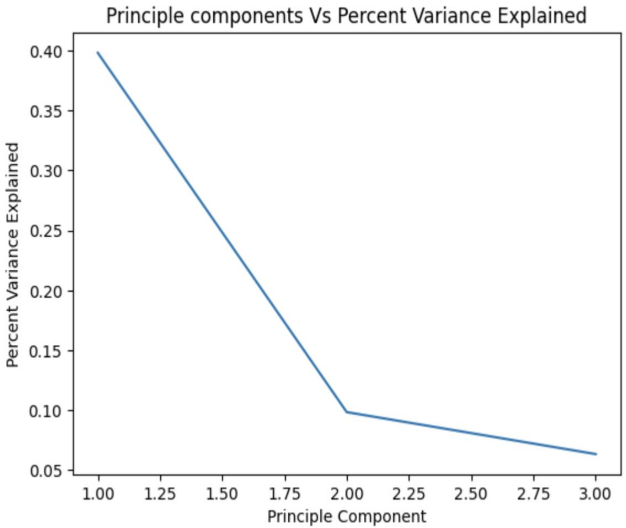
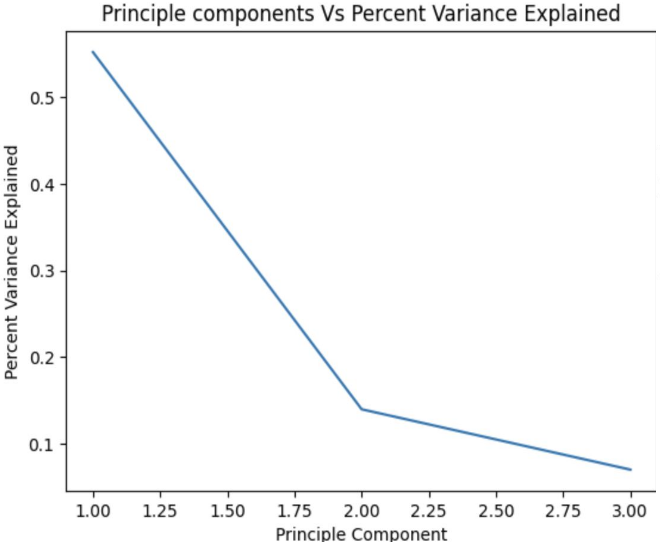
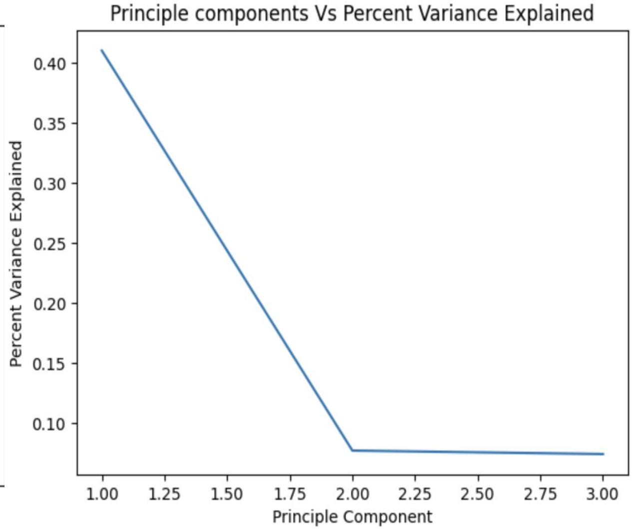
4. Guess ĝ and eliminate impossible solutions in solution space by running elim(Z, ĝ, coloring_ĝ)
5. Repeat steps 1-4

Cubic time complexity O($|S|^3$)

Adapted from Michael Bonthron’s 2022 approach

PCA & 90 - Cosine Similarity

1. Vectorize words in solution space into (130 x |Z|) matrix A
2. Find the SVD of A and select the 3 best rank 1 approximations
3. Find percent variance explained by rank 1 approximation matrix
4. Use cosine similarity to find smallest distance from rank 1 approximation to closest word
5. Find the best word that maximizes percent variance explained multiplied by (90 - cosine similarity)
6. Eliminate solution space based on best word
7. Repeat till solution is found


Principle components Vs Percent Variance Explained
0.130
0.125
0.120
0.115
0.110
0.105
0.100
0.095
0.090
Percent Variance Explained
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
Principle Component

Principle components Vs Percent Variance Explained
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Percent Variance Explained
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
Principle Component

Principle components Vs Percent Variance Explained
0.20
0.18
0.16
0.14
0.12
0.10
0.08
Percent Variance Explained
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
Principle Component

Principle components Vs Percent Variance Explained
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
Percent Variance Explained
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
Principle Component

Principle components Vs Percent Variance Explained
0.5
0.4
0.3
0.2
0.1
Percent Variance Explained
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
Principle Component

Principle components Vs Percent Variance Explained
0.40
0.35
0.30
0.25
0.20
0.15
0.10
Percent Variance Explained
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00
Principle Component

Results & Conclusion

Graphs & Tables

	Success Rate	Avg No. of Attempts	Avg Guess Biased Reward	Avg Game Runtime
Random	0.973	4.08	$-(4.08 + 0.027\alpha) \leq -4.269$	0.381 sec
Minimax	0.984	3.92	$-(3.92 + 0.016\alpha) \leq -4.032$	3.45 sec
Minex	0.981	3.93	$-(3.93 + 0.019\alpha) \leq -4.063$	3.45 sec
Dominant Eigenvector	0.771	3.72	$-(3.72 + 0.229\alpha) \leq -5.323$	0.996 sec
PCA & 90 - Cosine Similarity	0.989	3.79	$-(3.79 + .011\alpha) \leq -3.867$	2.63 sec

Conclusions

- Random algorithm performs surprisingly well given its simplicity
- Dominant Eigenvector is good at guessing quickly when it succeeds but also has a very high failure rate
- PCA & 90 - Cosine Similarity performs the best
- Improvements upon the Bonthron approach.
  - Alternating 1st and 2nd singular vectors
  - Removing guessed word from action space
  - Restricting guess space to solution space, at least for the last 1-2 guesses
- Very applicable at beating your friends in Wordle
  - All code will be public on GitHub!