Artificial Intelligence - Assignment 1

VINDHYA JAIN (B22AI060)

Introduction

In this assignment, I tackle the problem of generating high-scoring sentences from a given vocabulary and transition matrix. IDDFS, UCS, Greedy, and A* search algorithms are implemented to maximize sentence scores, with an evaluation and analysis of each algorithm's performance across varying parameters and inputs.

Generating Vocabulary

First, a vocabulary of **L** words is created using the MLTK module in python. The ***generate_meaningful_vocabulary*** function picks words at random from NLTK's list of English words until the desired number of unique words is gathered.

Generating Transition Matrix

The transition matrix provides scores for the probability $P(w_a | w_b)$, which represents the likelihood of encountering word w_a given that the previous word was w_b. The first L rows of the transition matrix file correspond to the transition scores for words $\{w_1,...,w_L\}$ in the vocabulary given a particular word. The $(L+1)^{th}$ row contains scores $P(w_i|w_0)$ for i = 1,...,L. The $(L+2)^{th}$ row contains L scores $P(w_{L+1}|w_k)$ for k = 1,...,L.

Such a transition matrix has been generated for varying values of n = {3, 4, 5, 6}, keeping in mind that each element must be within the range of 0 to 1, as it denotes probability.

$$
\begin{array}{cccc}
P(w_1|w_0) & P(w_2|w_0) & \dots & P(w_L|w_0) \\
P(w_1|w_1) & P(w_2|w_1) & \dots & P(w_L|w_1) \\
P(w_1|w_2) & P(w_2|w_2) & \dots & P(w_L|w_2) \\
. & . & & . \\
. & . & & . \\
. & . & & . \\
P(w_{L+1}|w_1) & P(w_{L+1}|w_2) & \dots & P(w_{L+1}|w_L)
\end{array}
$$

For ease of indexing, I have shifted the $(L+1)^{th}$ row which contains $P(w_i|w_0)$ above.

Normalization of Transition Matrix

Since the sum of $P(w_1|w_1) + P(w_2|w_1) + \dots + P(w_L|w_1) + P(w_{L+1}|w_1)$ should be 1, the matrix must be normalized accordingly. This is because the sum of probabilities of all the words coming after a word a must be 1. The alignment of this matrix means that the sum of each row and also the $P(w_{L+1}|w_i)$, which is situated in the last row of the matrix, needs to be recorded, and consequently that row, and the $P(w_{L+1}|w_i)$ must be divided by this sum.

In the case of the first row, $P(w_{L+1}|w_0)$ will be equal to 0. This is because the probability of <eos> appearing after <sos> will be 0 as this would imply that there is no sentence.

Objective

The objective is to find a sentence that has n + 2 words (considering <sos> and <eos> tokens) that has the maximum score. The score of the sentence can be calculated as shown below.

$$S\left(w_0, w_1, \ldots, w_n, w_{L+1}\right) = P\left(w_{L+1}|w_n\right)\left(\prod_{j=1}^{n} P\left(w_j|w_{j-1}\right)\right) P\left(w_0\right)$$

To achieve this, 4 search algorithms have been employed, namely: IDDFS, Greedy Search, UCS, and A* Search. The following sections go through the intuitions, and validity of each of the algorithms.

IDDFS (Iterative Deepening Depth-First Search)

IDDFS is a combination of two search techniques: Depth-First Search (DFS) and Breadth-First Search (BFS). It gradually explores deeper levels of possible sentence constructions while avoiding the memory issues of a full BFS and the premature exploration cut-off of a traditional DFS.

Intuition

In this case, the algorithm starts with a start of sentence token <sos> and gradually increases the depth of exploration, looking for sentences that maximize a score based on the transition matrix. For each depth, it uses DFS to explore all possible sentence combinations up to that depth, keeping track of the sentence with the highest score. Since at each depth, one word is added to the sentence, the full sentence of size n (ignoring <sos> and <eos> tokens) is formed only at depth n+2.

For the sample transition matrix and sample vocabulary given to us, the results are:
Best Sentence: <sos> grass grass airport green <eos>
Best Score: 0.0013628499234583315

Validity of the IDDFS algorithm:

Let us try to solve a simple working example with the algorithm.
1. Verifying Optimality

vocab = {a, b}
tmatrix = [0.1, 0.9]
[0.1, 0.8]
[0.1, 0.8]
[0.1, 0.1]
n = 2

IDDFS Execution:

- Depth 1:
  Possible sequences:
  [0, 1] (score = 0.1)

[0, 2] (score = 0.9)
Sequences extended to depth 1 result in- Best sequence: [0, 2] with score 0.9

- Depth 2:
  Possible sequences:
  [0, 1, 1] (score = 0.1*0.1 = 0.01)
  [0, 1, 2] (score = 0.1*0.8 = 0.08)
  [0, 2, 1] (score = 0.9*0.1 = 0.09)
  [0, 2, 2] (score = 0.9*0.8 = 0.72)
- Depth 3:
  Possible sequences:
  [0, 1, 1, -1] (score = 0.1*0.1*0.1 = 0.001)
  [0, 1, 2, -1] (score = 0.1*0.8*0.1 = 0.008)
  [0, 2, 1, -1] (score = 0.9*0.1*0.1 = 0.009)
  [0, 2, 2, -1] (score = 0.9*0.8*0.1 = 0.072)
  Sequences extended to depth 2 result in- Best sequence: [0, 2, 2] with score 0.72

Result - The sentence should be **<sos>, b, b, <eos>** with a score of 0.072. Which is clearly the optimal solution.

2. Examining Scalability

vocab = {a, b}
tmatrix = [0.5, 0.5]
[0.3, 0.5]
[0.6, 0.3]
[0.2, 0.1]
n = 2

IDDFS Execution:

- Depth 1:
  Possible sequences:
  [0, 1] (score = 0.5)
  [0, 2] (score = 0.5)
  Sequences extended to depth 1 result in- Best sequence: [0, 2] with score 0.5

- Depth 2:
  Possible sequences:
  [0, 1, 1] (score = 0.5*0.3 = 0.15)
  [0, 1, 2] (score = 0.5*0.5 = 0.25)
  [0, 2, 1] (score = 0.5*0.6 = 0.30)
  [0, 2, 2] (score = 0.5*0.3= 0.15)
  Sequences extended to depth 2 result in- Best sequence: [0, 2, 2] with score 0.72

- Depth 3:
  Possible sequences:
  [0, 1, 1, -1] (score = 0.5*0.3*0.2 = 0.03)
  [0, 1, 2, -1] (score = 0.5*0.5*0.1 = 0.025)
  [0, 2, 1, -1] (score = 0.5*0.6*0.2 = 0.06)
  [0, 2, 2, -1] (score = 0.5*0.3*0.1 = 0.015)

Result - The sentence should be **<sos>, b, a, <eos>** with a score of 0.06

Optimizing IDDFS

Since at each depth, one word is added to the sentence, the full sentence of size n (ignoring <sos> and <eos> tokens) is formed only at depth n+2. So, exploring at depths above or below this is pointless. Hence to optimize the algorithm, the code should be run at a fixed depth 'n'. Since IDDFS is an optimal algorithm, it always returns the optimal result.

These steps will make the running time of the algorithm much smaller.

Greedy Search

In Greedy Search, at each step, the algorithm chooses the option that looks best at that moment, without considering future consequences.

Intuition

Here, the goal is to generate a sentence from a vocabulary (vocab) by selecting the word that maximizes the transition probability from the current word to the next one. The greedy nature of this algorithm means it makes the locally optimal choice at each step (the highest probability word), without considering the long-term consequences. Consequently, the greedy method almost always gave the lesser optimal solution.

- The sentence starts with the <SoS> token.
- At each step, it looks at the transition probabilities from the current word to all other possible words and picks the one with the highest probability (the "greedy" choice).
- This word is added to the sentence, and the process continues until n words are added.
- Once the sentence reaches the required length (n words), the algorithm appends <EoS>, and the sentence is complete.
- Throughout the process, the overall score is updated by multiplying the transition probabilities associated with the chosen words, providing the total score for the sentence.

For the sample transition matrix and sample vocabulary given to us, the results are:
Best Sentence: <sos> green airport at grass <eos>
Best Score: 0.00042083261917621664

Validity of the Greedy Search algorithm:

Let us try to solve a simple working example with the algorithm.

1. Verifying Optimality

vocab = {a, b}
tmatrix = [0.1, 0.9]
[0.4, 0.5]
[0.2, 0.1]
[0.1, 0.7]
n = 2

Greedy Search Execution:

1. Current sentence = **<sos>, b**; score = 0.9
2. Current sentence = **<sos>, b, a**; score = 0.9*0.2 = 0.18
3. Current sentence = **<sos>, b, a, <eos>**; score = 0.18*0.1 = 0.018

Result - The sentence found by greedy is **<sos>, b, a, <eos>** with a score of 0.018. However, **<sos>, b, b, <eos>** has a score of 0.63. Greedy search does not always give optimal solutions.

2. Examining Scalability

vocab = {a, b}
tmatrix = [0.5, 0.5]
[0.3, 0.5]
[0.6, 0.3]
[0.2, 0.1]
n = 2

Greedy Search Execution:

1. Current sentence = **<sos>, b**; score = 0.5
2. Current sentence = **<sos>, b, a**; score = 0.5*0.6 = 0.3
3. Current sentence = **<sos>, b, a, <eos>**; score = 0.3*0.2 = 0.06

Result - The sentence should be **<sos>, b, a, <eos>** with a score of 0.06

UCS (Uniform Cost Search)

The uniform_cost_search (UCS) algorithm is designed to generate a sentence by exploring possible word combinations in a way that maximizes the score, using a priority queue (heap) to always expand the most promising path first. This method ensures that the search finds the globally optimal sentence, as it examines all possible sentences but focuses on those with the highest cumulative score (g heuristic) so far.

Key intuitions behind the algorithm:

- Uses a min-heap to store negative scores, ensuring higher values are prioritized, starting with <SoS> and an initial score of -1.

- The highest-score path is popped from the queue and is extended by adding the next word, updating best_score to avoid redundant paths.
- A sentence of n words plus <sos> and <eos> is constructed, returning the score once the path reaches n+2 tokens.
- The score of each path is compuusing calculate_score based on cumulative transition probabilities.
- Optimal Sentence: Expands the highest-score path first, ensuring the most promising sentence paths are explored and updated.

Best Sentence: <sos> grass grass airport green <eos>
Best Score: 0.0013628499234583315

Validity of the UCS algorithm:

Let us try to solve a simple working example with the algorithm.

1. Verifying Optimality

vocab = {a, b}
tmatrix = [0.1, 0.9]
[0.1, 0.8]
[0.1, 0.8]
[0.1, 0.1]

UCS Execution:

1. Priority Queue = {[0, 2] score = 0.9, [0, 1] score = 0.1}
2. Priority Queue = {[0, 2, 2] score = 0.72, [0, 1] score = 0.1, [0, 2, 1] score = 0.09}
3. Priority Queue = {[0, 2, 2, -1] score = 0.072, [0, 1] score = 0.1, [0, 2, 1] score = 0.09}

Result - The sentence found by UCS is **<sos>, b, b, <eos>** with a score of 0.072. Which is clearly the optimal solution.

2. Examining Scalability

vocab = {a, b}
tmatrix = [0.5, 0.5]
[0.3, 0.5]
[0.6, 0.3]
[0.2, 0.1]
n = 2

UCS Execution:

1. Priority Queue = {[0, 2] score = 0.5, [0, 1] score = 0.5}
2. Priority Queue = {[0, 2, 1] score = 0.30, [0, 2, 2] score = 0.15, [0, 1] score = 0.1}
3. Priority Queue = {[0, 2, 2, -1] score = 0.06, [0, 1] score = 0.1, [0, 2, 1] score = 0.09}

Result - The sentence should be **<sos>, b, a, <eos>** with a score of 0.06

A* Search

A* search balances between actual scores and heuristic estimates to find the highest-scoring sentence efficiently. The heuristic (h) helps guide the search towards paths with better potential, ensuring optimal results and a more guided search.

Key intuitions behind the algorithm:

- Heuristic function (**h_heuristic**) estimates the best-case score by assuming maximum transition probabilities for remaining words.
- A min-heap is used to **prioritize paths** with the highest potential score by storing -f = g + h, where g is the actual score (calculate_score function) and h is the heuristic estimate.
- The path with the lowest f score is popped and explored further by adding possible words, and updates the score before pushing the new path back into the queue if it's better.
- **f = g + h** balances actual score and heuristic estimate to ensure the sentence with the highest potential score is explored first.

Best Sentence: <sos> grass grass airport green <eos>
Best Score: 0.0013628499234583315

***h* heuristic and it's admissibility**

The h_heuristic function estimates the best possible score for completing a sentence from a given path. It calculates the remaining words needed to complete the sentence, including the end-of-sequence token <eos>. If only <eos> is left, it returns the maximum transition probability from the current word to <eos>. For more than one remaining word, it assumes the highest transition probabilities at each step, multiplying them to estimate the future score. The heuristic optimistically assumes the best-case transitions for each remaining word and includes the final transition to <eos>, providing a forward-looking estimate of potential score.

$$h(path) = \left(\prod_{k=1}^{remaining\ words\ -\ 1} max(tmatrix[current\_word][j] \right) \times tmatrix[vocab\_size+1][last\_word]$$

A heuristic is admissible if it never overestimates the actual cost.

$$0 \leq h(x) \leq h^*(x)$$

Since we are trying to maximize the score rather than minimize the cost, the following inequality should be satisfied:

$$0 \leq s^*(x) \leq s(x)$$

For any sentence say, s = {<sos>, w_1, w_2, w_3, <eos>}, the actual score will be:

$$P(w1|w0) \cdot P(w2|w1) \cdot P(w3|w2) \cdot P(w4|w3) \text{ -(1)}$$

Now, for the estimation of best score at say s = {<sos>, w_1, w_2} will be:

$$P(w3|w2) \cdot P(w4|w3) \text{ -(2)}$$

Every term in both the equations will be less than 1 (<1) as they are probabilities. This implies the following:

$$P(w1|w0) \cdot P(w2|w1) \cdot P(w3|w2) \cdot P(w4|w3) \leq P(w3|w2) \cdot P(w4|w3)$$

Which satisfies the condition for an admissible heuristic. In fact, even in the case where the entire sentence needs to be estimated, the estimation will be, at best, equal to the actual score, and will not subceed it.

Validity of the A* algorithm:

Let us try to solve a simple working example with the algorithm.

1. Verifying Optimality

vocab = {a, b}
tmatrix = [0.1, 0.9]
[0.1, 0.8]
[0.1, 0.8]
[0.1, 0.1]

A* Execution:

1. Priority Queue = {[0, 2] f = 0.9 + 0.8*0.1 = 0.98, [0, 1] f = 0.1 + 0.8*0.1 = 0.18 }
2. Priority Queue = {[0, 2, 2] f = 0.72 + 0.1 = 0.82, [0, 2, 1] f = 0.09 + 0.1 = 0.19, [0, 1] f = 0.18}
3. Priority Queue = {[0, 2, 2, -1] f = 0.072, [0, 2, 1] f = 0.09 + 0.1 = 0.19, [0, 1] f = 0.18}

Result - The sentence found by UCS is **<sos>, b, b, <eos>** with a score of 0.072. Which is clearly the optimal solution.

2. Examining Scalability

vocab = {a, b}
tmatrix = [0.5, 0.5]
[0.3, 0.5]
[0.6, 0.3]
[0.2, 0.1]
n = 2

UCS Execution:

4. Priority Queue = {[0, 2] f = 0.5 + 0.6*0.2 = 0.62, [0, 1] f = 0.5 + 0.5*0.1 = 0.55}
5. Priority Queue = {[0, 2, 1] f = 0.3 + 0.2 = 0.5, [0, 2, 2] f = 0.15 + 0.1 = 0.25, [0, 1] f = 0.55}
6. Priority Queue = {[0, 2, 2, -1] f = 0.06, [0, 2, 2] f = 0.15 + 0.1 = 0.25, [0, 1] f = 0.55}

Result - The sentence should be **<sos>, b, a, <eos>** with a score of 0.06

Running the algorithms for n = {3, 4, 5, 6} and for L = {3, 5, 10, 15}

(showing results for 1 iteration only)

n, L	IDDFS	Greedy	UCS	A*
n = 3, L = 3	Best Sentence: <sos> episcotister brick episcotister <eos> Best Score: 0.026226487745486027	Best Sentence: <sos> episcotister brick blamelessly <eos> Best Score: 0.012576506189115037	Best Sentence: <sos> episcotister brick episcotister <eos> Best Score: 0.026226487745486027	Best Sentence: <sos> episcotister brick episcotister <eos> Best Score: 0.026226487745486027
n = 3, L = 5	Best Sentence: <sos> akmuddar trochospherical trochospherical <eos> Best Score: 0.006378472480457569	Best Sentence <sos> akmuddar akmuddar akmuddar <eos> Best Score 0.0013752359589764554	Best Sentence: <sos> akmuddar trochospherical trochospherical <eos> Best Score: 0.006378472480457569	Best Sentence: <sos> akmuddar trochospherical trochospherical <eos> Best Score: 0.006378472480457569
n = 3, L = 10	Best Sentence: <sos> camphoryl suprasensible spinster <eos> Best Score: 0.0007246235038835748	Best Sentence <sos> camphoryl camphoryl camphoryl <eos> Best Score 2.2596364220168506e-05	Best Sentence: <sos> camphoryl suprasensible spinster <eos> Best Score: 0.0007246235038835748	Best Sentence: <sos> camphoryl suprasensible spinster <eos> Best Score: 0.0007246235038835748
n = 3, L = 15	Best Sentence: <sos> prorsal fistulize urea <eos> Best Score: 0.0002997204515610171	Best Sentence <sos> predinner uncorruptness Tamerlanism <eos> Best Score 3.999789208266827e-05	Best Sentence: <sos> prorsal fistulize urea <eos> Best Score: 0.0002997204515610171	Best Sentence: <sos> prorsal fistulize urea <eos> Best Score: 0.0002997204515610171
n = 4, L = 3	Best Sentence: <sos> hypsographic disequalize hypsographic disequalize <eos> Best Score: 0.021302122351959888	Best Sentence <sos> hypsographic disequalize disequalize disequalize <eos> Best Score 0.015651259586463833	Best Sentence: <sos> hypsographic disequalize hypsographic disequalize <eos> Best Score: 0.021302122351959888	Best Sentence: <sos> hypsographic disequalize hypsographic disequalize <eos> Best Score: 0.021302122351959888
n = 4, L = 5	Best Sentence: <sos> plethorical plethorical plethorical Codiales <eos> Best Score: 0.0016205024990572777	Best Sentence <sos> plethorical Codiales stiltify Codiales <eos> Best Score 0.0011933055597579069	Best Sentence: <sos> plethorical plethorical plethorical Codiales <eos> Best Score: 0.0016205024990572777	Best Sentence: <sos> plethorical plethorical plethorical Codiales <eos> Best Score: 0.0016205024990572777
n = 4, L = 10	Best Sentence: <sos> escambron fumaric admonitionist escambron <eos> Best Score: 0.00010291793853042955	Best Sentence <sos> escambron fumaric covado grisaille <eos> Best Score 6.634200173892421e-05	Best Sentence: <sos> escambron fumaric admonitionist escambron <eos> Best Score: 0.00010291793853042955	Best Sentence: <sos> escambron fumaric admonitionist escambron <eos> Best Score: 0.00010291793853042955
n = 4, L = 15	<sos> seldomly deferrable deferrable seldomly <eos> 3.362092992762703e-05	Best Sentence <sos> spann hallmarker infallibility forebreast <eos> Best Score 4.623236003879526e-06	<sos> seldomly deferrable deferrable seldomly <eos> 3.362092992762703e-05	<sos> seldomly deferrable deferrable seldomly <eos> 3.362092992762703e-05
n = 5, L = 3	Best Sentence: <sos>	Best Sentence <sos>	Best Sentence: <sos>	Best Sentence: <sos>

	defeasanced defeasanced defeasanced defeasanced defeasanced <eos> Best Score: 0.004211681234409794	pungency Papilionoidea Papilionoidea Papilionoidea Papilionoidea <eos> Best Score 0.001678963912606569	defeasanced defeasanced defeasanced defeasanced defeasanced <eos> Best Score: 0.004211681234409794	defeasanced defeasanced defeasanced defeasanced defeasanced <eos> Best Score: 0.004211681234409794
n = 5, L = 5	Best Sentence: <sos> companionway unbiddable unbiddable unbiddable unbiddable <eos> Best Score: 0.0004791581524798238	Best Sentence <sos> drawbeam drawbeam drawbeam drawbeam drawbeam <eos> Best Score 0.00015427002610636674	Best Sentence: <sos> companionway unbiddable unbiddable unbiddable unbiddable <eos> Best Score: 0.0004791581524798238	Best Sentence: <sos> companionway unbiddable unbiddable unbiddable unbiddable <eos> Best Score: 0.0004791581524798238
n = 5, L = 10	Best Sentence: <sos> Hasidic stereochromy stereochromy stereochromy stereochromy <eos> Best Score: 5.2202642592441056e-05	Best Sentence <sos> Hasidic stereochromy catmint lacteous hesitant <eos> Best Score 3.474350756497131e-05	Best Sentence: <sos> Hasidic stereochromy stereochromy stereochromy stereochromy <eos> Best Score: 5.2202642592441056e-05	Best Sentence: <sos> Hasidic stereochromy stereochromy stereochromy stereochromy <eos> Best Score: 5.2202642592441056e-05
n = 5, L = 15	Best Sentence: <sos> Olax interaxial Jeremias proposable arjun <eos> Best Score: 4.103706828401125e-06	Best Sentence <sos> poststertorous ambition pseudocorneous proflavine virtually <eos> Best Score 2.067420664942701e-06	Best Sentence: <sos> Olax interaxial Jeremias proposable arjun <eos> Best Score: 4.103706828401125e-06	Best Sentence: <sos> Olax interaxial Jeremias proposable arjun <eos> Best Score: 4.103706828401125e-06
n = 6, L = 3	Best Sentence: <sos> Stradivari bulbilla bulbilla bulbilla bulbilla Stradivari <eos> Best Score: 0.002119129331662704	Best Sentence <sos> Stradivari bulbilla Stradivari bulbilla Stradivari bulbilla <eos> Best Score 0.00035908619149970677	Best Sentence: <sos> Stradivari bulbilla bulbilla bulbilla bulbilla Stradivari <eos> Best Score: 0.002119129331662704	Best Sentence: <sos> Stradivari bulbilla bulbilla bulbilla bulbilla Stradivari <eos> Best Score: 0.002119129331662704
n = 6, L = 5	Best Sentence: <sos> sane sane sane sane postpyretic befathered <eos> Best Score: 0.0002530944327184048	Best Sentence <sos> sane sane sane sane sane sane <eos> Best Score 8.632768242493892e-05	Best Sentence: <sos> sane sane sane sane postpyretic befathered <eos> Best Score: 0.0002530944327184048	Best Sentence: <sos> sane sane sane sane postpyretic befathered <eos> Best Score: 0.0002530944327184048
n = 6, L = 10	<sos> semipurulent visitorship Wanyakyusa unladled unredeemable abastardize <eos> 5.120782770002172e-06	Best Sentence <sos> predeficiency abastardize unladled unredeemable abastardize unladled <eos> Best Score 1.0459927288890906e-06	<sos> semipurulent visitorship Wanyakyusa unladled unredeemable abastardize <eos> 5.120782770002172e-06	<sos> semipurulent visitorship Wanyakyusa unladled unredeemable abastardize <eos> 5.120782770002172e-06
n = 6, L = 15	<sos> semipurulent visitorship Wanyakyusa unladled unredeemable abastardize <eos> 5.120782770002172e-06	Best Sentence <sos> hyphenated preferredly preferredly preferredly preferredly <eos> Best Score 1.1781705394182212e-07	<sos> semipurulent visitorship Wanyakyusa unladled unredeemable abastardize <eos> 5.120782770002172e-06	<sos> semipurulent visitorship Wanyakyusa unladled unredeemable abastardize <eos> 5.120782770002172e-06

Observations

From the above table, the following observations can be drawn:

1. Greedy Search does not always predict an optimal solution and typically has a lesser best score than the other algorithms.
2. IDDFS, UCS, A* search all are optimal search algorithms and therefore predict the same best sentence and best score always which is the optimal solution.

Bibliography

Sites used:
https://stanford-cs329s.github.io/syllabus.html
https://ai.berkeley.edu/home.html
https://ics.uci.edu/~kkask/Fall-2016%20CS271/slides/03-InformedHeuristicSearch.pdf

Discussion partners: Khushi Bhardwaj (B22AI026)