RL Genetic Algorithm For Prompt Engineering

Anmol Goyal, Connor Doug Smith

December 2023

1 Introduction

The LLMs have been gaining a large amount of popularity in the recent years. They have further been integrated into different areas from personal use, customer service to drive key decisions in healthcare and finance. However the outputs or the responses of the models are largely depended on the inputs or the way the questions are asked. The task of crafting the inputs for the modes, called prompts, is called prompt engineering. The project aims to leverage the evolutionary algorithms to the discrete nature of the problem to automate the engineering of long prompts to improve the responses of the systems.

2 Objective

The aim is to automate the process of engineering prompts for the large language models to enhance the performance in specific tasks. This would have significant impact on different areas

1. Enhancing user experience for customer service agents and similar chat services as they would be able to get better responses without putting lot of effort to think how to formulate the problem.
2. Provide deeper insights into working of LLMs and further enhance the field of prompt engineering.

3 Algorithm

The key aspects of the algorithm are,

3.1 Chromosomes

The chromosomes for for our algorithms are the prompts broken down into a list of sentences. Sophisticated sentence extraction algorithms are used to ensure the generated sentences are semantically correct.

3.2 Genes

Each sentence in a prompt is considered a gene. Except for the structural parts like "Answer: ", "Question: " etc.

3.3 Mutation Function

The mutation is carried out by randomly paraphrasing a single sentence from the prompt. The sentence selected for mutation are selected using a multi arm bandit approach, evaluating the Lin-UCB scores for the sentences selecting the one with the highest score. The scores are calculated using a history of sentences. If s^{before} is the sentence before modification and s^{after} is the sentence after modification and r is the difference of scores for the prompts for the two sentences then at iteration T history h_T is,

$$h = \left[\left(s_t^{before}, s_t^{after}, r_t\right)\right], t = 1 \cdots T-1$$

If H is a matrix of $T-1$ rows with each row being an embedding of the sentence, s_i^{before} in a vector space for all $i = 1 \cdots T-1$, r is the vector of score difference, then the index of the sentence to mutate id,

$$\begin{aligned} A &= H^T H + \lambda I \\ w_T &= A^{-1} H^T r \\ e &= \phi\left(s_{(i)}\right)^T \\ id &= argmax_i e^T w_T + \alpha\sqrt{e^T A^{-1} e} \end{aligned}$$

The sentence selected using this method is mutated with a probability of p. The constants α and λ are used to control the exploitation and exploration trade off in the contextual bandit algorithm. Higher values of α would result in the algorithm putting more emphasis on exploration than exploitation, selecting more diverse sentences to mutate.

3.4 Crossover

The crossover is done by selecting a point to cross over the two prompts and exchange the sentences from the point int the two prompts. The optimal point of selection is found by randomly selecting a cross over point and finding the similarity between the sentences at that point in the two prompts. If the similarity is above a certain threshold then the point is selected otherwise it is sampled again.

The top prompt is kept using the crossover function.

3.5 Fitness function

Since the problem chosen is the causal judgement problem from the Big Bench dataset, the possible answers are "yes" and "no". Thus the fitness functions is the accuracy.

3.6 Algorithm

The algorithm is, Fig 1, if k is the size of the population and n is the number of iterations,

1. The score for the initial prompt is calculated and stored by the algorithm.
2. The initial population is generated by mutating the initial prompt.
3. The scores for the initial population is found using the fitness function.
4. For i in 1 to n do
5. Create a new population using crossover function
6. Randomly select 2 prompts to mutate.
7. Evaluate the new population
8. Keep the top k prompts in the population

4 Classes

The application has the following classes,

4.1 Template

Facilitates the creation of templates to be used in both mutator and the original prompts. The key functions are,

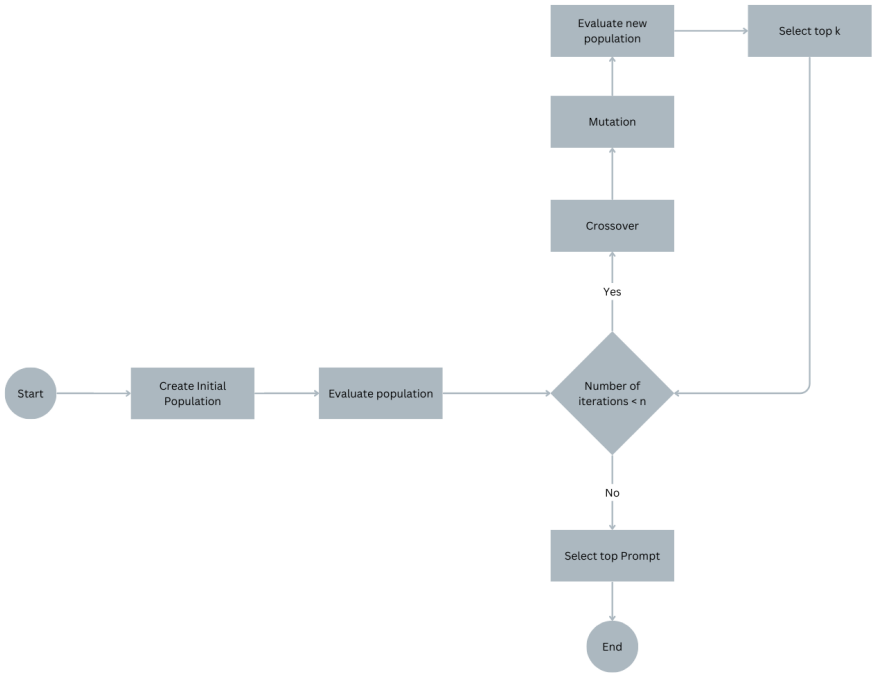

Figure 1: Flow chart of the algorithm

4.1.1 __init__

The constructor of the class. Takes in two arguments,

1. template, specifies either the path to the template or the template string
2. demo_template, specifies either the path or the template string to the template for the demos used in the few shot learning of the LLM

The function loads and saves the templates in the instance variables.

4.1.2 get_llm_input

Takes as input the information to populate the template from and returns the populated template as a string. Takes as arguments,

1. instruction, replaces it with the [instruction] placeholder
2. demos, an array of dictionary with keys, question, and answer and populates the demo template for each element in the list

4.2 Dataset

Functions to load and process the datasets. The key functions include,

4.2.1 __init__

The constructor of the class. Takes as input,

1. format, format of the data provided
2. filepath, path of the file containing the data to load

3. dict_data, a dictionary or lost of dictionaries to load the data from. Used only if format is dict
4. template, an instance of Template class. Specifies the template of the prompt to be used for creating the inputs from the samples in the dataset

4.3 Mutator

To configure and create the mutator instance used for mutating the prompts for the LLMs in the genetic algorithm. The key functions are,

4.3.1 __init__

The constructor of the class. Takes the parameters to set the configuration of the mutator. The arguments are,

1. model, the model to be used. See the readme document for more details.
2. lambda, lambda to be used in the CB algorithm
3. p, probability of selecting the sentence to be mutated using the CB algorithm
4. alpha, α in the CB algorithm
5. combine_prob, the probability of combining sentences while selecting sentences for mutation. Does not work for smaller models, thus not used in the current implementation

4.3.2 get_sentences

Splits the prompt into sentences preserving the semantic correctness. Takes two arguments,

1. prompt, prompt to be split into sentences
2. remove_tags, removes tags like "Answer: " or "Question: " from the prompt before splitting

4.3.3 get_sentence_distance

Finds the similarity between two sentences using a normalized dot product of encoded vectors. Takes two arguments,

1. a, b, sentences to find similarity for.

4.3.4 select_sentence

Selects the sentence to be mutated using the Contextual Bandit Algorithm. Takes a single argument, prompt, the prompt from to be mutated.

4.3.5 get_mutated_prompt

Mutates the prompt and returns the mutated prompt as a string. Takes a single argument, prompt, the prompt from to be mutated.

4.3.6 save_history

Saves the history generated during the training phase in a csv file. Takes a single argument filename, the location to store the history.

4.4 GA

The main genetic algorithm class. Provides functions to generate the best prompts as well as for diagnosing the working of the algorithm. The key functions are,

4.4.1 __init__

The constructor of the class. Takes parameters to configure the genetic algorithm.

1. model, The model to use for inference
2. dataset, an instance of Dataset class. Specifies the dataset to be used for training and testing purposes
3. mutator, an instance of Mutator class. Specifies the mutator to use for mutating the prompts.
4. fitness_func, a function or callable object. The custom fitness function.
5. num_iter, number of generations to run the genetic algorithm.
6. k, number of chromosomes in the population
7. sample_size, number of samples to use from the dataset during the training

4.4.2 evaluate_fitness

Evaluates a prompt on the dataset using the fitness function. The arguments,

1. prompt, the prompt to evaluate
2. df, the dataset to use while evaluating the fitness function. If None, the dataset stored in the instance is used.

4.4.3 generate_answers

Generates the responses of the model for a prompt for every sample in dataset. The arguments,

1. prompt, the prompt to evaluate
2. df, the dataset to use while evaluating the fitness function. If None, the dataset stored in the instance is used.

4.4.4 tune_prompt

Tunes the initial prompt and stores the top k prompts, top candidates in the instance.

4.4.5 best_prompt

Tunes the initial prompt, saves the history of the mutator and returns the best prompt from the top chromosomes.

4.4.6 save_best_prompt

Tunes the best prompt for the initial prompt. Finds the fitness of intial and the best prompt on the entire dataset and prints the corresponding scores. Saves the best prompt in a text file and returns the best prompt and score. The arguments,

1. filename, the location to store the best prompt in.

4.4.7 answer_question

Generates the response of the model for a question. If the algorithm has been run then uses the saved best prompt from the file best_prompt.txt, otherwise uses the initial prompt. For inference purposes. The arguments,

1. question, the question for which the answer needs to be found.

4.5 GeneticWrapper

The class for hypercycle integration.

The data flow between the classes, Fig 2.

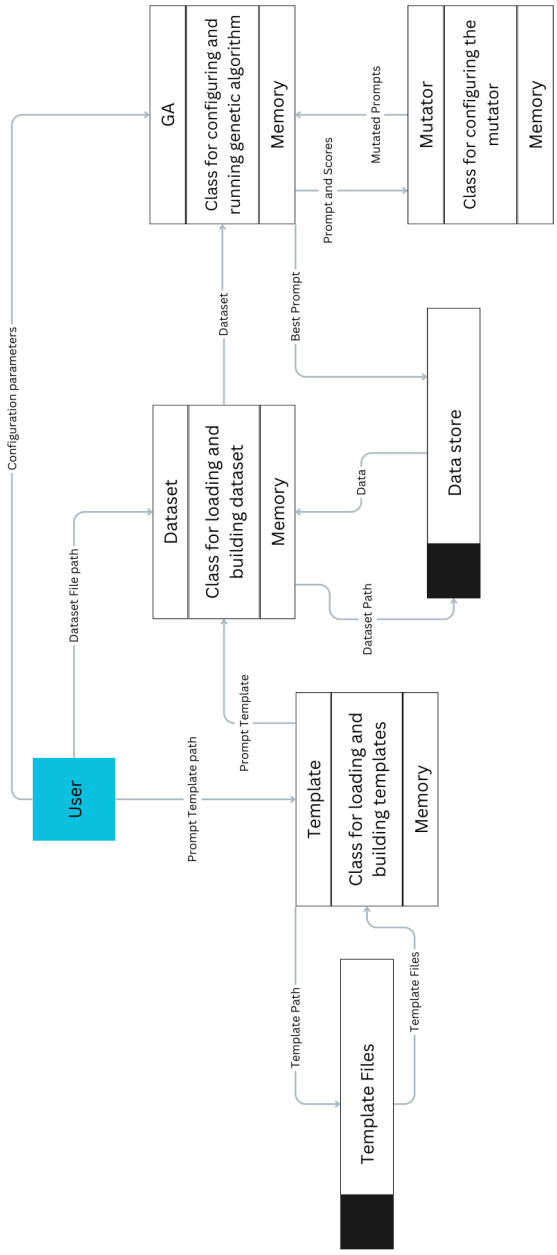

Figure 2: Level 1 Data Flow diagram for flow between the classes