Genetic Algorithm For Prompt Engineering

Anmol Goyal

December 2023

1 Introduction

Large Language Models have been gaining a large amount of popularity in recent years. They have further been integrated into different areas from personal use and customer service to driving key decisions in healthcare and finance. However, the outputs or the responses of the models are largely dependent on the inputs or 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 evolutionary algorithms and the discrete nature of the problem to automate the engineering of long prompts and 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 areas such as:

1. Enhancing user experience for customer service agents and similar chat services as consumers would get more accurate results while using natural language.
2. Provide deeper insights into the workings of LLMs and further enhance the field of prompt engineering.

The program provides an easy to use interface for users and programmers to test the usability of the algorithm for their needs.

3 Classes

The application has the following classes:

3.1 Template

Facilitates the creation of templates to be used in both the mutator and the original prompts.

3.2 Dataset

Functions to load and process datasets.

3.3 Mutator

Creates the mutator instance used for mutating the prompts for the LLM.

3.4 GA

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

4 Algorithm

The key aspects of the algorithm are:

4.1 Chromosomes

The chromosomes 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.

4.2 Genes

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

4.3 Mutation Function

The mutation is carried out by randomly paraphrasing a single sentence from the prompt. The sentence selected for mutation is 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,

$$h = \begin{matrix} h \\ \\ \end{matrix} \begin{matrix} s^{before} \\ {}_t, s^{after}{}_t, r_t \end{matrix} \begin{matrix} i \\ , t = 1 \cdots T \\ - 1 \end{matrix}$$

If H is a matrix of $T - 1$ rows with each row being an embedding of the sentence, s^{before} ${}_i$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,

$$A = H^T H + \lambda I$$

$$w_T = A^{-1}H^T r$$

$$e = \phi s_{(i)} \quad T_{id} = argmax_i e^T w_T + \alpha \sqrt{e^T A^{-1} e}$$

The sentence selected using this method is mutated with a probability of *p*.

4.4 Crossover

The crossover is done by selecting a point to cross over the two prompts and exchange the sentences from the point into 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.

4.5 Fitness function

Since the problem chosen is the causal judgment problem from the Big Bench dataset, the possible answers are ”yes” and ”no”. Thus the fitness function is the measure of accuracy.

5 Algorithm

The algorithm is as follows, 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 are found using the fitness function.
4. For i in 1 to *n* do:
    a. Create a new population using the crossover function.
    b. Randomly select 2 prompts to mutate.
    c. Evaluate the new population.
    d. Keep the top *k* prompts in the population.