HumanPrompt (v2022.10)

I. Introduction

Motivation

I believe most of you researchers and engineers have your own pipelines to prompt LM APIs. It is easy to implement a single script to call LMs, while *extremely difficult to unify and extend*, especially with the increasing emergence of "new LLM prompting method" papers.

This project provides a unified framework to prompt LM APIs, which is:

- **Modularized:** The prompting pipeline is split up into individual modules. Users can easily combine and integrate them as their wish!
- **Inclusive of 20+ LLM prompting methods:** More than 20 LLM prompting methods are already included in this modularized pipeline.
- **Fully customizable and extensible:** Set everything (prompt, method…) in a config file instead of hard coding.
- **Interactive and visualized:** An interactive UI for users to explore prompting LLMs everywhere at any time.

II. Implementation

Methods Components

1. [Optional] Sample selection -> *x*
2. [Optional] Sample annotation -> *x,y*
3. Select dataset -> *x,y* **(strong dependency)**
4. Retrieve in-context examples, *x_test* -> *x_test_prompt_i, (i = 1,2,3,..., k, k is the number of examples)*
5. Prompt wrap, *x_test, x_test_prompt_i* -> *prompt*.
    a. **Usage(Discussion needed)**
        i. `PromptFileFull()`: e.g., load a prompt .txt with few-shot and inference sample.
        ii. `PromptFileFewShot():` e.g., load a few-shot prompt .txt. Should combine with `PromptFormat()`.
            1. To be added
        iii. `PromptFormat(x_test, x_test_i)`: e.g., select the first three rows
        iv. `PromptCoT(x_test, x_test_i):` give a default zero-shot method to automatically annotate CoT, e.g., "Let's think step-by-step."
    b. **View, Annotate and Trade**
        i. Playground
            1. Free-form, OpenAI like
            2. Select dataset example, prompt and method to run (SKG annotation like)
        ii. Prompt management(from different publications, different users) for each datasets

        iii. Benchmark for methods on all datasets fitable
        iv. Trade prompt (TBD)
6. Function callings, *prompt -> response(s)*
    a. `Iterable calling`: Make it albe to start from (4) and again and again to get enough responses(When saving, save all the infos, add switch to control that)
    b. `Caching mechanism`: cache the response for the same (model, prompt), since the API calls can be expensive(ama, decomp)
    c. `Security`: Key protection of OpenAI for service machine
    d. `Multithread`:
7. Extraction
    a. Return value format due to different APIs(OpenAI format, huggingface format, AI21 format…)
    b. Exraction for answers
        i. `Matching`: e.g. CoT(regex…)
        ii. `Execution`: e.g. Basic Program, Binder Program
8. Aggregation
    a. No aggregation
    b. Majority vote(ThinkSum)
        i. Simple
        ii. Prob
        iii. Biased
            1. Hyper-parameter
            2. DiVerse-like
    c. Weakly supervised algorithm
        i. Dependency Graph
        ii. Reinforcement Learning
9. [Optional] Model Training
10. Method implementation under framework & run experiment
    a. Configuration
        i. jsonnet
    b. Logging
        i. wandb usage inside

Papers to Include (TO BE ADDED)

Modules each paper regards:

- CoT standard: 5
- ZeroGen: 5, 6, 7, 8, (9)
- Self-consistency: 5, 8
- STaR: 5, 6, 7 9
- ImplicitRelations: todo
- ZeroShotCoT: 5
- Least-to-Most: 5, 6
- RLPrompt: 5, 6, 7, 9?
- DiVerse: 5, 8
- RationaleEnsemble: 5(iteratively), 7, 8
- ReAct: 5, 6

- Decomp: 5, 6
- Self-Ask: 5, 6
- AutoCoT: todo
- Binder: 4, 5, 7, 8
- AMA Prompt: 5, 6, 8
- CocoGen: todo
- Self-Improve: 5, 8
- Generate rather than retrieve: 5, 9