Chatgpt Approach for Classifying Repos

We want to use the most powerful tool "chatgpt" to help us classify repos.

Using ChatGPT API (*https://openai.com/blog/introducing-chatgpt-and-whisper-apis*) and GitHub API (*https://docs.github.com/en/rest*), we develop a Python program that retrieves the readme file content for each repository and feeds it into the algorithm below. There are 3 scenarios from supervised to unsupervised approach.

Scenario 1: First, we start with supervised approach by giving cluster topics based on Ethereum EIP categories (*https://eips.ethereum.org*): Core, Network, Interface, ERC, Meta, and Informational.

This method is the most predictable regarding our anticipated clustering of categories. Yet, its applicability to other open-source software projects is limited without predefined categories from external sources. To address this constraint, we have developed additional scenarios (scenario2 and scenario3).

* Note: we upload "ethereum_repos.json" for running this code block, which is a file consisting of repo name, description and readme.

```
!pip3 install -q openai
```

```
##########################################################################
# Senario 1
##########################################################################

import json
from openai import OpenAI
from random import shuffle
import time
import pickle

with open('ethereum_repos.json', 'r') as f:
    repos = json.load(f)

client = OpenAI(api_key= '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')
```

```
instruction = '''Classify each given repository Readme within one of the given categories:

Core: Improvements requiring a consensus fork, as well as changes that are not necessarily consensus critical but may be relevant to “core dev” discussions (for example, the PoA algorithm for testnets).
Networking: Includes improvements around devp2p and Light Ethereum Subprotocol, as well as proposed improvements to network protocol specifications of whisper and swarm.
Interface: Includes improvements around client API/RPC specifications and standards, and also certain language-level standards like method names and contract ABIs. The label “interface” aligns with the interfaces repo and discussion should primarily occur in that repository before an EIP is submitted to the EIPs repository.
ERC: Application-level standards and conventions, including contract standards such as token standards, name registries, URI schemes, library/package formats, and account abstraction.
Meta: Describes a process surrounding Ethereum or proposes a change to (or an event in) a process. Process EIPs are like Standards Track EIPs but apply to areas other than the Ethereum protocol itself. They may propose an implementation, but not to Ethereum's codebase; they often require community consensus; unlike Informational EIPs, they are more than recommendations, and users are typically not free to ignore them. Examples include procedures, guidelines, changes to the decision-making process, and changes to the tools or environment used in Ethereum development. Any meta-EIP is also considered a Process EIP.
Informational: Describes an Ethereum design issue or provides general guidelines or information to the Ethereum community but does not propose a new feature. Informational EIPs do not necessarily represent Ethereum community consensus or a recommendation, so users and implementers are free to ignore Informational EIPs or follow their advice.

Your output should be in a consistent format <repo name>: <class>.
'''

N = 6

completions = []

for i in range(N):
    partition = len(repos)//N
    for repo in repos[partition*i:min(partition*(i+1), len(repos))]:
        messages = [
            {"role": "system", "content": instruction},
        ]
        repo_name, repo_desc, readme = repo['name'], repo['desc'][:1024] if repo['desc'] else 'None',
repo['readme'][:3000] if repo['readme'] else 'None'
```

```
        messages.append({"role": "user", "content": f"Repo name: {repo_name}. Description: {repo_desc}. Readme:
{readme}"})
        completion = client.chat.completions.create(
          model="gpt-4",
          messages=messages
        )
        print(f'Done {repo_name}')
        completions.append(completion)
        with open(f'results/{repo_name}', 'wb') as f:
            pickle.dump(completion, f)
    # time.sleep(60)

print([completion.choices for completion in completions])
```

Scenario 2: We continue with a semi-supervised approach. In scenario 2, we ease the constraints on the number of categories and their respective topics. While we continue to recommend the initial six categories to the model, we now permit the model to generate additional clusters if there are repositories that support the formation of these new groups.

* Note: due to token limits, we only upload repo description to feed into chatgpt. Based on all these repo descriptions, chatgpt will come up with several cluster topics.

```
##############################################################################
# Senario 2
##############################################################################
```

```
# collect all repo descriptions

with open ('ethereum_repos.json', 'r') as f:
  repos=json.load(f)


information_feed = []
for repo in repos:
  desc = repo["desc"]
```

```
  information_feed.append(desc)
```

```
# Feed ChatGPT with all repo descriptions and instructions.

client = OpenAI(api_key='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')

instruction = '''Based on the content of the readme files provided, group all repositories into 6-10 clusters and provide a short definition of each cluster. Suggested categories for clusters but not limited to: Core: Improvements requiring a consensus fork, as well as changes that are not necessarily consensus critical but may be relevant to “core dev” discussions. Networking: Includes improvements around devp2p and Light Ethereum Subprotocol, as well as proposed improvements to network protocol specifications of whisper and swarm. Interface: Includes improvements around client API/RPC specifications and standards, and also certain language-level standards like method names and contract ABIs. ERC: Application-level standards and conventions, including contract standards such as token standards, name registries, URI schemes, library/package formats, and account abstraction. Meta: Describes a process surrounding Ethereum or proposes a change to (or an event in) a process. Process EIPs are like Standards Track EIPs but apply to areas other than the Ethereum protocol itself. They may propose an implementation, but not to Ethereum's codebase; they often require community consensus; unlike Informational EIPs, they are more than recommendations, and users are typically not free to ignore them. Examples include procedures, guidelines, changes to the decision-making process, and changes to the tools or environment used in Ethereum development. Any meta-EIP is also considered a Process EIP. Informational: Describes a Ethereum design issue, or provides general guidelines or information to the Ethereum community, but does not propose a new feature.'''

messages = [
    {"role": "system", "content": instruction},
 ]

# Build the prompt to integrate all repo descriptions into it
prompt = ''

for i, desc in enumerate(information_feed):
  prompt = prompt + f'{desc}\n'

# prompt: 'these are ... repo 1 desc is ...; repo 2 ...'
messages.append({"role": "user", 'content': prompt})
completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
```

```
  messages=messages
)

import pickle

with open('completion', 'wb') as f:
  pickle.dump(completion, f)

print(completion.choices[0].message.content)
```

Scenario 3: we explore an unsupervised approach, in other words, highly autonomous, minimally supervised approach. In this scenario, we permit the model to classify without predefined categories, maintaining an upper limit on the number of categories to ensure they remain interpretable by humans.

```
##############################################################################
# Senario 3
##############################################################################
# collect all repo descriptions

with open ('ethereum_repos.json', 'r') as f:
  repos=json.load(f)


information_feed = []
for repo in repos:
  desc = repo["desc"]
  information_feed.append(desc)
```

```
# Feed ChatGPT with all repo descriptions and instructions.

client = OpenAI(api_key='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')

instruction = '''Based on the content of the file provided, group all repositories into up to 15 clusters and provide a
short definition of each cluster. Note that each cluster should contain more than 1 repository. In addition, the topics of
```

```
repositories in the cluster should be as similar as possible.The repositories are primarily software development repositories"'

messages = [
      {"role": "system", "content": instruction},
  ]

# Build the prompt to integrate all repo descriptions into it
prompt = ''

for i, desc in enumerate(information_feed):
  prompt = prompt + f'{desc}\n'

# prompt: 'these are ... repo 1 desc is ...; repo 2 ...'
messages.append({"role": "user", 'content': prompt})
completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=messages
)

import pickle

with open('completion', 'wb') as f:
  pickle.dump(completion, f)

print(completion.choices[0].message.content)
```