A guidance for automatic annotation for biomodels

by Haoxue Wang

There are three stages of our automatic annotation procedure

- prepare the pmc link, download the full text in .txt if it is open source, otherwise obtain the abstract only
- use our algorithms to do the automatic annotation with matching ontology, output saved as json file
- transfer the output into readable .csv file, which can be compared with ground-true labels if available

Stage 1 prepare the text

For biomodels, we can get the models' corresponding pmc id through official API

```
import os
import requests
from bs4 import BeautifulSoup
import pandas as pd


# get all the curated biomodels api url
base_url = "https://www.ebi.ac.uk/biomodels/BIOMD"
start_index = 1
end_index = 1073


# Create the list of API URLs
urls = [f"{base_url}{str(i).zfill(10)}?format=json" for i in
range(start_index, end_index + 1)]
# store training data


df = pd.DataFrame(columns=["publicationId", "description", "abstract",
"link"])
for url in urls:
  # Perform the API request
  try:
      print("fetch the data for", url)
      response = requests.get(url)
      data = response.json()
      publicationId = data['publicationId']
      # Parse the HTML data using BeautifulSoup
      soup = BeautifulSoup(data['description'], 'html.parser')
      description = soup.get_text()
      abstract = data['publication']['synopsis']
```

```
      link=data['publication']['link']
      df = df.append({"publicationId":publicationId, "description":
description, "abstract" :abstract, "link": link}, ignore_index=True)
  except:
      pass

def pubmed_to_pmc(pubmed_link):
    return pubmed_link.replace("http://identifiers.org/pubmed/",
"pmc")

# there are other links from non pubmed, check them manually if needed
pubmed_rows = df[df['link'].str.contains('pubmed', case=False)]
# List of PubMed links
pubmed_links= pubmed_rows["link"]

# Loop through the PubMed links and convert to PMC format
pmc_links = [pubmed_to_pmc(link) for link in pubmed_links]
result_df = pd.concat([pubmed_rows, pd.DataFrame(pmc_links)], axis=1)
pmc_ids = list(set(pmc_links))  # the unique id

# from the above coding, we can get the abstract and the description
from the Biomodels directly
data = pd.DataFrame(df)
abstract = data['abstract']
description = data['description']
data.to_csv('data_ebi.csv')


# Using the result from csv
data = pd.read_csv('data_ebi.csv')

import csv
import os

def create_txt_files(csv_file, save_folder):
    if not os.path.exists(save_folder):
        os.makedirs(save_folder)

    with open(csv_file, 'r') as csvfile:
        csvreader = csv.reader(csvfile)

        # Skip the header row
        header = next(csvreader)

        for row in csvreader:
            abstract = row[2]  # Assuming 'abstract' column is at
index 2 (0-based index)
            name = row[0]  # Assuming '0' column is at index 4 (0-
based index)
```

```
            txt_filename = os.path.join(save_folder, f'{name}.txt')
            with open(txt_filename, 'w') as txtfile:
                txtfile.write(abstract)

# Usage example:
csv_file = 'data_ebi.csv'
save_folder = 'data'
create_txt_files(csv_file, save_folder)
# Thus we download each abstract as the content of .txt, with the
biomodels ID as the file name
```

we have the pmc id, we need to download the full text/abstract through official European PMC API/ PubMed Central® (PMC)

```
# download the full text through PubMed Central® (PMC)
def get_full_text(pmc_id):
    url =
f"https://www.ncbi.nlm.nih.gov/research/bionlp/RESTful/pmcoa.cgi/BioC_
json/{pmc_id[3:]}/unicode"
    response = requests.get(url)

    if response.status_code == 200:
        return response.json()
    else:
        print(f"Error: Unable to fetch the full-text for PMC ID:
{pmc_id}")
        return None

def extract_all_text(json_data):
    text_content = ''
    if isinstance(json_data, dict):
        for key, value in json_data.items():
            text_content += extract_all_text(value)
    elif isinstance(json_data, list):
        for item in json_data:
            text_content += extract_all_text(item)
    elif isinstance(json_data, str):
        text_content += json_data + ' '

    return text_content

save_path = '/Users/haoxuewang/haoxue_pytorch/data/texts'

for pmc_id in pmc_ids:
    full_text = get_full_text(pmc_id)
    if full_text:
        text_content = extract_all_text(full_text)
        file_name = f'{os.path.basename(pmc_id)}.txt'
        with open(os.path.join(save_path, file_name), 'w',
```

```
encoding='utf-8') as txt_file:
            txt_file.write(text_content)

        print(f"Converted to .txt file: {os.path.join(save_path,
file_name)}")

# download the full text through official European PMC API, it is
slightly different as we need to transfer the xml into .txt
import os
import requests
import json
import xml.etree.ElementTree as ET

# Replace 'YOUR_API_KEY' with your actual API key
API_KEY = '9d4ee27f78c8c5d685ab9251061512dfc708'
BASE_URL = 'https://www.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi'

def get_full_text(pmc_id):
    # url =
f"https://www.ncbi.nlm.nih.gov/research/bionlp/RESTful/pmcoa.cgi/BioC_
json/{pmc_id[3:]}/unicode"
    url =
f"https://www.ebi.ac.uk/europepmc/webservices/rest/{pmc_id}/fullTextXM
L"
    response = requests.get(url)

    if response.status_code == 200:
        return response.text
    else:
        print(f"Error: Unable to fetch the full-text for PMC ID:
{pmc_id}")
        return None

def extract_all_text_from_xml(xml_data):
    if xml_data is None:
        return None

    try:
        root = ET.fromstring(xml_data)
        all_text = ' '.join(element.text.strip() for element in
root.iter() if element.text)
        return all_text
    except ET.ParseError:
        print("Error parsing XML data.")
        return None



save_path = '/Users/your_own_path'
for pmc_id in pmc_ids:
```

```
    full_text = get_full_text(pmc_id)
    if full_text:
        text_content = extract_all_text_from_xml(full_text)
        file_name = f'{os.path.basename(pmc_id)}.txt'
        with open(os.path.join(save_path, file_name), 'w',
encoding='utf-8') as txt_file:
            txt_file.write(text_content)

        print(f"Converted to .txt file: {os.path.join(save_path,
file_name)}")
```

Stage 2 automatic annotation

```
# automatic annation for biomodels
# by Haoxue Wang

import spacy
import json
import requests
import os
from pathlib import Path

def get_spacy(file_path):
    nlp = spacy.load("en_core_sci_sm")
    # nlp = spacy.load("en_core_sci_scibert")
    with open(file_path, "r") as file:
        texts = file.readlines()
    processed_data = []
    unique_entities = set()
    for i, text in enumerate(texts):
        doc = nlp(text)
        # Extract entities from the document
        entities = [{"entity": ent.text} for ent in doc.ents]
        unique_entities.update(ent.text for ent in doc.ents)
        # Add the processed data for this text to the list
        processed_data.append({"entities": entities})
    return processed_data

def search_ols(query):
    base_url = "https://www.ebi.ac.uk/ols/api/select"
    params = {
        "q": query,
        "groupField": "iri",
        "start": 0
    }

    response = requests.get(base_url, params=params)
    if response.status_code == 200:
        return response.json()
```

```
    else:
        raise Exception(f"Error accessing OLS API. Status code:
{response.status_code}")

def get_first_result_info(query):
    try:
        data = search_ols(query)
        if data["response"]["numFound"] == 0:
            return
        first_result = data["response"]["docs"][0]
        label = first_result["label"]
        # bad_ontology = ['bao',
'cco','afo',"snomed","dicom","reproduceme","vo"]
        good_ontology =
["taxonomy","go","more","ncit","bto","reactome","hdo","chebi","fma","e
fo","po","hpo","cto","kc","kd","ofbi","eo","vario","uk","sbo","ko","kp
","en","teddy","eco"]
        bad_label = [',']
        if has_three_or_less_words(label):
            if label not in bad_label:
                ontology = first_result["ontology_name"]
                if ontology in good_ontology:
                    iri = first_result["iri"]

                    result_info = {
                        "label": label,
                        "ontology": ontology,
                        "iri": iri
                        }
                    return result_info

    except Exception as e:
        print("Error:", e)

def has_three_or_less_words(input_string):
    # Split the string into words
    words = input_string.split()

    # Count the number of words
    num_words = len(words)

    # Compare the count with the threshold
    if num_words < 4:
        return True
    else:
        return False

def isnot_common_word(word):
    common_words = {"functional","trends"} # replace the common words
in your own list
```

```
    return word.lower() not in common_words

# txt_dir='/Users/haoxuewang/Desktop/haoxue_ebi/abstract/'
txt_dir='/Users/haoxuewang/Desktop/haoxue_ebi/full_text/'
input_directory=Path(txt_dir,'texts')
output_directory=Path(txt_dir,'annotation')
file_names = os.listdir(input_directory)
models_name = [os.path.splitext(os.path.basename(file_name))[0] for
file_name in file_names if file_name.endswith('.txt')]

for model_name in models_name:
    input_path = os.path.join(input_directory, f"{model_name}.txt")
    words= get_spacy(input_path)
    result=[]
    for word in words:
        entities = word["entities"]
        unique_entities = set(item['entity'] for item in entities)
        unique_entities=list(unique_entities)
        for entity in unique_entities:
            search_query=entity
            if isnot_common_word(search_query):
                result.append(get_first_result_info(search_query))

    # Save the 'pure_list' as a JSON file with the same name as the
input file
    output_file_path = Path(output_directory, f"{model_name}.json")
    print(output_file_path)
    with open(output_file_path, "w") as output_file:
        json.dump(result, output_file, indent=4)
```

You will get the file with json file in the following strcture


```
[
    {
        "label": "INTACT",
        "ontology": "efo",
        "iri": "http://www.ebi.ac.uk/efo/EFO_0010037"
    },
    {
        "label": "tetrahydrofolate",
        "ontology": "chebi",
        "iri": "http://purl.obolibrary.org/obo/CHEBI_67016"
    },
    null,
    {
        "label": "Pharmacokinetic Parameters Domain",
        "ontology": "ncit",
        "iri": "http://purl.obolibrary.org/obo/NCIT_C49607"
    },
```

```
    {
        "label": "Enzyme",
        "ontology": "ncit",
        "iri": "http://purl.obolibrary.org/obo/NCIT_C16554"
    },
    null,
    {
        "label": "tetrahydrofolate interconversion",
        "ontology": "go",
        "iri": "http://purl.obolibrary.org/obo/GO_0035999"
    },
    {
        "label": "purine",
        "ontology": "chebi",
        "iri": "http://purl.obolibrary.org/obo/CHEBI_35584"
    }
]
```

Stage 3 transfer the output to .csv

```
# If you have the test label to compare the coverage
# test the data coverage
import pandas as pd

data = pd.read_csv('metadata.csv')
df = data.groupby('BIOMD')['comment'].apply(lambda x: '
'.join(x)).reset_index()
aggregated_df = df.groupby('BIOMD')['comment'].apply(lambda x:
x.str.split()).reset_index()

# Print the aggregated DataFrame
aggregated_df
```

we can calculate the coverage with the test labels if avilable

```
import os
from pathlib import Path
import json
from itertools import chain

def read_json_file(json_file_path):
    try:
        with open(json_file_path, 'r') as jsonfile:
                data = json.load(jsonfile)
                labels = []
                for i in range(len(data)):
                    if data[i] is not None:
                        label = data[i]['label'].split()
                        labels.append(label)
```

```
                label_list=list(chain(*labels))
            return label_list
    except FileNotFoundError:
    # Ignore the error and continue with the rest of the code
            pass


def find_overlapping_elements(list1, list2):
    set1 = set(list1)
    set2 = set(list2)
    overlapping_elements = set1.intersection(set2)
    return list(overlapping_elements)


txt_dir='/Users/haoxuewang/Desktop/haoxue_ebi/abstract/'
input_directory=Path(txt_dir,'annotation')
result=[]
for i in range(len(aggregated_df)):
    try:
        model_name = aggregated_df['BIOMD'][i]
        input_path = os.path.join(input_directory,
f"{model_name}.json")
        label_texts= read_json_file(input_path)
        lowercase_label_texts = [item.lower() for item in label_texts]
        true_list = [item for item in aggregated_df['comment'][i] if
item != '-']
        lowercase_true_list = [item.lower() for item in true_list]
        overlapping=
find_overlapping_elements(lowercase_label_texts,lowercase_true_list)
        coverage = len(overlapping)/len(aggregated_df['comment'][i])
        result.append((model_name, coverage))
    except (ValueError, TypeError):
        # Ignore the error and continue with the loop
        pass
```

We need the transfer the output in a csv format

For the below code, you change the .csv structure easily

```
# create the table
def check_json_file(json_file_path):
    try:
        with open(json_file_path, 'r') as jsonfile:
                data = json.load(jsonfile)
                labels = []
                iris=[]
                for i in range(len(data)):
                    if data[i] is not None:
                        label = data[i]['label']
                        iri =data[i]['iri']
```

```
                        labels.append(label)
                        iris.append(iri)
                label_list=list(labels)
                iri_list=list(iris)

        return label_list, iri_list
    except FileNotFoundError:
    # Ignore the error and continue with the rest of the code
        pass

# test the data coverage
import pandas as pd

data = pd.read_csv('metadata.csv')
df = data.groupby('BIOMD')['comment'].apply(lambda x: ',
'.join(x)).reset_index()
aggregated_df = df.groupby('BIOMD')['comment'].apply(lambda x:
x.str.split()).reset_index()
new_df=data.groupby('BIOMD')['value'].apply(lambda x: ',
'.join(x)).reset_index()
# Print the aggregated DataFrame

tests=[]
for i in range(len(result)):
    if result[i][1] >= 0:
        tests.append(result[i][0])
txt_dir='/Users/haoxuewang/Desktop/haoxue_ebi/abstract/'
table =[]
for test in tests:
    input_directory=Path(txt_dir,'annotation')
    input_path = os.path.join(input_directory, f"{test}.json")
    label_texts,iri_texts = check_json_file(input_path)
    which = df['BIOMD']== test
    label_true=list(df['comment'][which])
    iri_true=list(new_df['value'][which])
    table.append((test,label_texts,iri_texts,label_true,iri_true))

table=pd.DataFrame(table)
table_annotation = pd.merge(pd.DataFrame(result), table, on=0,
how='inner')
table_annotation
new_columns = {0: 'Biomodel name', '1_x': 'coverage', '1_y':'automatic
annotation', 2:'ontology for automatic annotation', 3:'manual
annotation', 4:'ontology for manual annotation'}
table_annotation.rename(columns=new_columns, inplace=True)
table_annotation
sorted_df = table_annotation.sort_values(by='coverage',
ascending=False)
sorted_df
```

```
sorted_df.to_csv('sorted_df.csv')
```

As a supplement material

This is to show the procedure to obtain useful ontology

```
import pandas as pd

data = pd.read_csv('metadata.csv')
df = data.groupby('BIOMD')['ontology'].apply(lambda x: '
'.join(x)).reset_index()
aggregated_df = df.groupby('BIOMD')['ontology'].apply(lambda x:
x.str.split()).reset_index()

def process_string(input_string):
    words = input_string.split()  # Split the input string into words
    result = []
    res=[]
    if len(words) == 1:
        words = ''.join([char.lower() for sublist in words for char in
sublist])
        result.append(words)
    else:
        for word in words:
            res.append(word[0])
        res = ''.join([char.lower() for sublist in res for char in
sublist])
        result.append(res)
    return result

abb=[]
for i in range(len(data)):
    ontology=data['ontology'][i]
    ont = process_string(ontology)
    abb.append(ont)

abbreviation= pd.DataFrame(abb)
data1 = pd.concat([data,abbreviation], axis=1)
data1

unique_df = abbreviation.drop_duplicates().to_string(index=False)
print(unique_df)
good_ono=["taxonomy","go","more","ncit","bto","reactome","hdo","chebi"
,"fma","efo","po","hpo","cto","kc","kd","ofbi","eo","vario","uk","sbo"
,"ko","kp","en","teddy","eco"]
```