Appendix A: API

ADS 509: Final Project Code

Team 4

Zachariah Freitas and Brianne Bell

In [1]:

```
# Loading necessary libraries
import pandas as pd
import numpy as np
import os
import re
import random
import time
import datetime

import nltk
from collections import Counter, defaultdict
from nltk.corpus import stopwords
from string import punctuation
from tqdm import tqdm
import sqlite3

# from https://github.com/soumik12345/multi-label-text-classification/blob/master/arxi
import arxiv
```

In [2]:

```
# doing similar set up with setting up keywords to focus on
## Alternative keywords can be used to attempt better model performance or for differe
query_keywords = [
    "\"representation learning\"",
    "\"image generation\"",
    "\"object detection\"",
    "\"transformers\"",
    "\"image segmentation\"",
    "\"natural language\"",
    "\"graph\"",
    "\"colorization\"",
    "\"depth estimation\"",
    "\"point cloud\"",
    "\"structured data\"",
    "\"reinforcement learning\"",
    "\"attention\"",
    "\"tabular\"",
    "\"unsupervised learning\"",
    "\"semi-supervised learning\"",
    "\"explainable\"",
    "\"time series\"",
    "\"molecule\"",
    "\"physics\"",
    "\"graphics\""
]
```

In [3]:

```python
# https://github.com/soumik12345/multi-label-text-classification/blob/master/arxiv_scr

# We are pulling just the terms (topic), titles, and abstracts of articles
    ## with a limit of 20k for each to save time and space
# other queries can be pulled if needed/wanted but we are focusing on titles and abstr

client = arxiv.Client(num_retries=20, page_size=500)

def query_with_keywords(query):
    search = arxiv.Search(
        query=query,
        max_results=20000,
        sort_by=arxiv.SortCriterion.LastUpdatedDate
    )
    terms = []
    titles = []
    abstracts = []
    for res in tqdm(client.results(search), desc=query):
        if res.primary_category in ["cs.CV", "stat.ML", "cs.LG"]:
            terms.append(res.categories)
            titles.append(res.title)
            abstracts.append(res.summary)
    return terms, titles, abstracts
```

In [4]:

```python
# setting up save file
# if not os.path.isdir("arxiv_data") :
#     os.mkdir("arxiv_data")
```

In [5]:

```python
# setting up for pull
all_titles = []
all_summaries = []
all_terms = []

# timing:
start_time = datetime.datetime.now()

# pulling
for query in query_keywords:
    terms, titles, abstracts = query_with_keywords(query)
    all_titles.extend(titles)
    all_summaries.extend(abstracts)
    all_terms.extend(terms)

# seeing how long ^that took:
end_time = datetime.datetime.now()
print(end_time - start_time)
```

```
"representation learning": 6118it [01:41, 60.09it/s]
"image generation": 1978it [00:30, 64.01it/s]
"object detection": 6536it [02:01, 53.69it/s]
"transformers": 20000it [06:55, 48.10it/s]
"image segmentation": 2890it [00:43, 66.93it/s]
"natural language": 13021it [03:36, 60.18it/s]
"graph": 20000it [05:39, 58.89it/s]
"colorization": 20000it [05:37, 59.20it/s]
"depth estimation": 1218it [00:20, 60.71it/s]
"point cloud": 4308it [01:30, 47.41it/s]
"structured data": 1915it [00:35, 54.30it/s]
"reinforcement learning": 16211it [04:36, 58.58it/s]
"attention": 20000it [05:48, 57.44it/s]
"tabular": 1382it [00:21, 63.47it/s]
"unsupervised learning": 2763it [00:41, 66.14it/s]
"semi-supervised learning": 0it [00:03, ?it/s]
"explainable": 20000it [06:17, 52.97it/s]
"time series": 15302it [04:17, 59.39it/s]
"molecule": 20000it [05:29, 60.67it/s]
"physics": 20000it [08:39, 38.50it/s]
"graphics": 15861it [04:34, 57.71it/s]
1:10:04.621341
```

Interpretting the scraping results:

- "representation learning": went through 6118 iterations at 60.09it/s.
- "image generation": went through 1978 iterations at 64.01it/s]
- "object detection": went through 6536 iterations at 53.69it/s.
- "transformers": went through 20000 iterations at 48.10it/s.
- "image segmentation": went through 2890 iterations at 66.93it/s.
- "natural language": went through 13021 iterations at 60.18it/s.
- "graph": went through 20000 iterations at 58.89it/s.
- "colorization": went through 20000 iterations at 59.20it/s.
- "depth estimation": went through 1218 iterations at 60.71it/s.
- "point cloud": went through 4308 iterations at 47.41it/s.
- "structured data": went through 1915 iterations at 54.30it/s.
- "reinforcement learning": went through 16211 iterations at 58.58it/s.
- "attention": went through 20000 iterations at 57.44it/s.
- "tabular": went through 1382 iterations at 63.47it/s.
- "unsupervised learning": went through 2763 iterations at 66.14it/s.
- "semi-supervised learning": went through 0 iterations at ?it/s.
- "explainable": went through 20000 iterations at 52.97it/s.
- "time series": went through 15302 iterations at 59.39it/s.
- "molecule": went through 20000 iterations at 60.67it/s.
- "physics": went through 20000 iterations at 38.50it/s.
- "graphics": went through 15861 iterations at 57.71it/s.

Of particular note, seven of the 21 keywords maxed out the number of iterations. They are: "transformers", "graph", "colorization", "attention", "explainable", and "physics". At the other end of the spectrum is "semi-supervised learning" which went through zero iterations either because it is not in the archive or it is but in a different format without the hypen.

In [6]:

```
raw_data = pd.DataFrame({
    'titles': all_titles,
    'abstracts': all_summaries,
    'terms': all_terms
})

raw_data.head()
```

Out[6]:

	titles	abstracts	terms
0	Reinforcement Learning from Multiple Sensors v...	In many scenarios, observations from more than...	[cs.LG]
1	Interventional Causal Representation Learning	Causal representation learning seeks to extrac...	[stat.ML, cs.LG]
2	Self-Supervised Node Representation Learning v...	Self-supervised node representation learning a...	[cs.LG]
3	Out-of-Distribution Representation Learning fo...	Time series classification is an important pro...	[cs.LG, cs.AI]
4	Trading Information between Latents in Hierarc...	Variational Autoencoders (VAEs) were originall...	[stat.ML, cs.CV, cs.IT, cs.LG, math.IT]

In [7]:

```
raw_data.shape #(64573, 3)
```

Out[7]:

```
(64573, 3)
```

In [8]:

```
# saving to csv file because pulling data takes a long while

    # well I didn't get it to the file folder but it did write.


# This is how you save a pandas dataframe and all the details associated with it.  You
raw_data.to_pickle('G:\\My Drive\\ADS-509_Final_Team_Project\\arxiv_data_2023_02_13.pk

# Nextime you want to use it.  All you have to do is read it as a pickle file. The fil
# raw_data = pd.read_pickle('G:\\My Drive\\ADS-509_Final_Team_Project\\arxiv_data_2023
```