from abc import ABC, abstractmethod

from datetime import date

from importlib import import\_module

import pickle

from toxicity\_ml\_pipeline.settings.default\_settings\_tox import (

CLIENT,

EXISTING\_TASK\_VERSIONS,

GCS\_ADDRESS,

TRAINING\_DATA\_LOCATION,

)

from toxicity\_ml\_pipeline.utils.helpers import execute\_command, execute\_query

from toxicity\_ml\_pipeline.utils.queries import (

FULL\_QUERY,

FULL\_QUERY\_W\_TWEET\_TYPES,

PARSER\_UDF,

QUERY\_SETTINGS,

)

import numpy as np

import pandas

class DataframeLoader(ABC):

def \_\_init\_\_(self, project):

self.project = project

@abstractmethod

def produce\_query(self):

pass

@abstractmethod

def load\_data(self, test=False):

pass

class ENLoader(DataframeLoader):

def \_\_init\_\_(self, project, setting\_file):

super(ENLoader, self).\_\_init\_\_(project=project)

self.date\_begin = setting\_file.DATE\_BEGIN

self.date\_end = setting\_file.DATE\_END

TASK\_VERSION = setting\_file.TASK\_VERSION

if TASK\_VERSION not in EXISTING\_TASK\_VERSIONS:

raise ValueError

self.task\_version = TASK\_VERSION

self.query\_settings = dict(QUERY\_SETTINGS)

self.full\_query = FULL\_QUERY

def produce\_query(self, date\_begin, date\_end, task\_version=None, \*\*keys):

task\_version = self.task\_version if task\_version is None else task\_version

if task\_version in keys["table"]:

table\_name = keys["table"][task\_version]

print(f"Loading {table\_name}")

main\_query = keys["main"].format(

table=table\_name,

parser\_udf=PARSER\_UDF[task\_version],

date\_begin=date\_begin,

date\_end=date\_end,

)

return self.full\_query.format(

main\_table\_query=main\_query, date\_begin=date\_begin, date\_end=date\_end

)

return ""

def \_reload(self, test, file\_keyword):

query = f"SELECT \* from `{TRAINING\_DATA\_LOCATION.format(project=self.project)}\_{file\_keyword}`"

if test:

query += " ORDER BY RAND() LIMIT 1000"

try:

df = execute\_query(client=CLIENT, query=query)

except Exception:

print(

"Loading from BQ failed, trying to load from GCS. "

"NB: use this option only for intermediate files, which will be deleted at the end of "

"the project."

)

copy\_cmd = f"gsutil cp {GCS\_ADDRESS.format(project=self.project)}/training\_data/{file\_keyword}.pkl ."

execute\_command(copy\_cmd)

try:

with open(f"{file\_keyword}.pkl", "rb") as file:

df = pickle.load(file)

except Exception:

return None

if test:

df = df.sample(frac=1)

return df.iloc[:1000]

return df

def load\_data(self, test=False, \*\*kwargs):

if "reload" in kwargs and kwargs["reload"]:

df = self.\_reload(test, kwargs["reload"])

if df is not None and df.shape[0] > 0:

return df

df = None

query\_settings = self.query\_settings

if test:

query\_settings = {"fairness": self.query\_settings["fairness"]}

query\_settings["fairness"]["main"] += " LIMIT 500"

for table, query\_info in query\_settings.items():

curr\_query = self.produce\_query(

date\_begin=self.date\_begin, date\_end=self.date\_end, \*\*query\_info

)

if curr\_query == "":

continue

curr\_df = execute\_query(client=CLIENT, query=curr\_query)

curr\_df["origin"] = table

df = curr\_df if df is None else pandas.concat((df, curr\_df))

df["loading\_date"] = date.today()

df["date"] = pandas.to\_datetime(df.date)

return df

def load\_precision\_set(

self, begin\_date="...", end\_date="...", with\_tweet\_types=False, task\_version=3.5

):

if with\_tweet\_types:

self.full\_query = FULL\_QUERY\_W\_TWEET\_TYPES

query\_settings = self.query\_settings

curr\_query = self.produce\_query(

date\_begin=begin\_date,

date\_end=end\_date,

task\_version=task\_version,

\*\*query\_settings["precision"],

)

curr\_df = execute\_query(client=CLIENT, query=curr\_query)

curr\_df.rename(columns={"media\_url": "media\_presence"}, inplace=True)

return curr\_df

class ENLoaderWithSampling(ENLoader):

keywords = {

"politics": [

...

],

"insults": [

...

],

"race": [

...

],

}

n = ...

N = ...

def \_\_init\_\_(self, project):

self.raw\_loader = ENLoader(project=project)

if project == ...:

self.project = project

else:

raise ValueError

def sample\_with\_weights(self, df, n):

w = df["label"].value\_counts(normalize=True)[1]

dist = np.full((df.shape[0],), w)

sampled\_df = df.sample(n=n, weights=dist, replace=False)

return sampled\_df

def sample\_keywords(self, df, N, group):

print("\nmatching", group, "keywords...")

keyword\_list = self.keywords[group]

match\_df = df.loc[df.text.str.lower().str.contains("|".join(keyword\_list), regex=True)]

print("sampling N/3 from", group)

if match\_df.shape[0] <= N / 3:

print(

"WARNING: Sampling only",

match\_df.shape[0],

"instead of",

N / 3,

"examples from race focused tweets due to insufficient data",

)

sample\_df = match\_df

else:

print(

"sampling",

group,

"at",

round(match\_df["label"].value\_counts(normalize=True)[1], 3),

"% action rate",

)

sample\_df = self.sample\_with\_weights(match\_df, int(N / 3))

print(sample\_df.shape)

print(sample\_df.label.value\_counts(normalize=True))

print("\nshape of df before dropping sampled rows after", group, "matching..", df.shape[0])

df = df.loc[

df.index.difference(sample\_df.index),

]

print("\nshape of df after dropping sampled rows after", group, "matching..", df.shape[0])

return df, sample\_df

def sample\_first\_set\_helper(self, train\_df, first\_set, new\_n):

if first\_set == "prev":

fset = train\_df.loc[train\_df["origin"].isin(["prevalence", "causal prevalence"])]

print(

"sampling prev at", round(fset["label"].value\_counts(normalize=True)[1], 3), "% action rate"

)

else:

fset = train\_df

n\_fset = self.sample\_with\_weights(fset, new\_n)

print("len of sampled first set", n\_fset.shape[0])

print(n\_fset.label.value\_counts(normalize=True))

return n\_fset

def sample(self, df, first\_set, second\_set, keyword\_sampling, n, N):

train\_df = df[df.origin != "precision"]

val\_test\_df = df[df.origin == "precision"]

print("\nsampling first set of data")

new\_n = n - N if second\_set is not None else n

n\_fset = self.sample\_first\_set\_helper(train\_df, first\_set, new\_n)

print("\nsampling second set of data")

train\_df = train\_df.loc[

train\_df.index.difference(n\_fset.index),

]

if second\_set is None:

print("no second set sampling being done")

df = n\_fset.append(val\_test\_df)

return df

if second\_set == "prev":

sset = train\_df.loc[train\_df["origin"].isin(["prevalence", "causal prevalence"])]

elif second\_set == "fdr":

sset = train\_df.loc[train\_df["origin"] == "fdr"]

else:

sset = train\_df

if keyword\_sampling == True:

print("sampling based off of keywords defined...")

print("second set is", second\_set, "with length", sset.shape[0])

sset, n\_politics = self.sample\_keywords(sset, N, "politics")

sset, n\_insults = self.sample\_keywords(sset, N, "insults")

sset, n\_race = self.sample\_keywords(sset, N, "race")

n\_sset = n\_politics.append([n\_insults, n\_race])

print("len of sampled second set", n\_sset.shape[0])

else:

print(

"No keyword sampling. Instead random sampling from",

second\_set,

"at",

round(sset["label"].value\_counts(normalize=True)[1], 3),

"% action rate",

)

n\_sset = self.sample\_with\_weights(sset, N)

print("len of sampled second set", n\_sset.shape[0])

print(n\_sset.label.value\_counts(normalize=True))

df = n\_fset.append([n\_sset, val\_test\_df])

df = df.sample(frac=1).reset\_index(drop=True)

return df

def load\_data(

self, first\_set="prev", second\_set=None, keyword\_sampling=False, test=False, \*\*kwargs

):

n = kwargs.get("n", self.n)

N = kwargs.get("N", self.N)

df = self.raw\_loader.load\_data(test=test, \*\*kwargs)

return self.sample(df, first\_set, second\_set, keyword\_sampling, n, N)

class I18nLoader(DataframeLoader):

def \_\_init\_\_(self):

super().\_\_init\_\_(project=...)

from archive.settings.... import ACCEPTED\_LANGUAGES, QUERY\_SETTINGS

self.accepted\_languages = ACCEPTED\_LANGUAGES

self.query\_settings = dict(QUERY\_SETTINGS)

def produce\_query(self, language, query, dataset, table, lang):

query = query.format(dataset=dataset, table=table)

add\_query = f"AND reviewed.{lang}='{language}'"

query += add\_query

return query

def query\_keys(self, language, task=2, size="50"):

if task == 2:

if language == "ar":

self.query\_settings["adhoc\_v2"]["table"] = "..."

elif language == "tr":

self.query\_settings["adhoc\_v2"]["table"] = "..."

elif language == "es":

self.query\_settings["adhoc\_v2"]["table"] = f"..."

else:

self.query\_settings["adhoc\_v2"]["table"] = "..."

return self.query\_settings["adhoc\_v2"]

if task == 3:

return self.query\_settings["adhoc\_v3"]

raise ValueError(f"There are no other tasks than 2 or 3. {task} does not exist.")

def load\_data(self, language, test=False, task=2):

if language not in self.accepted\_languages:

raise ValueError(

f"Language not in the data {language}. Accepted values are " f"{self.accepted\_languages}"

)

print(".... adhoc data")

key\_dict = self.query\_keys(language=language, task=task)

query\_adhoc = self.produce\_query(language=language, \*\*key\_dict)

if test:

query\_adhoc += " LIMIT 500"

adhoc\_df = execute\_query(CLIENT, query\_adhoc)

if not (test or language == "tr" or task == 3):

if language == "es":

print(".... additional adhoc data")

key\_dict = self.query\_keys(language=language, size="100")

query\_adhoc = self.produce\_query(language=language, \*\*key\_dict)

adhoc\_df = pandas.concat(

(adhoc\_df, execute\_query(CLIENT, query\_adhoc)), axis=0, ignore\_index=True

)

print(".... prevalence data")

query\_prev = self.produce\_query(language=language, \*\*self.query\_settings["prevalence\_v2"])

prev\_df = execute\_query(CLIENT, query\_prev)

prev\_df["description"] = "Prevalence"

adhoc\_df = pandas.concat((adhoc\_df, prev\_df), axis=0, ignore\_index=True)

return self.clean(adhoc\_df)