```
import torch.nn.functional as F
from torch.utils.data.dataloader import DataLoader
import pandas as pd
from tqdm import tqdm
import numpy as np
import random
import logging
from sklearn.metrics import precision_recall_fscore_support
import os
import sys
from transformers import AutoConfig, AutoModel, AutoTokenizer, PreTrainedModel
from torch import nn
import torch
from transformers import get_linear_schedule_with_warmup, Trainer,
TrainingArguments, HfArgumentParser
from datasets import load_dataset
from collections.abc import Mapping
from typing import TYPE_CHECKING, Any, Callable, Dict, List, Optional, Tuple,
Union

# set up logging info
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def set_seed(seed):
    """ Set all seeds to make results reproducible """
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    np.random.seed(seed)
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)


class Set:
    def __init__(self, data_path=None, data_df=None,
embedding_hf_modelcard='princeton-nlp/sup-simcse-bert-base-uncased',
local_files_only=False):
        self.data_path = data_path
        self.embedding_hf_modelcard = embedding_hf_modelcard
        self.embedding = None
        self.local_files_only = local_files_only
        if self.data_path:
            dataset = load_dataset('csv', data_files=self.data_path)
```

```
            key = list(dataset.keys())[0]
            self.dataset = dataset[key]

    def get_embedding(self, embedding_hf_modelcard=None):
        if embedding_hf_modelcard:
            logger.info(f"**** Running {embedding_hf_modelcard} model to get
embeddings ****")
            model = AutoModel.from_pretrained(embedding_hf_modelcard,
local_files_only=self.local_files_only)
            model.to('cuda')
            tokenizer =
AutoTokenizer.from_pretrained(embedding_hf_modelcard,local_files_only=self.local_
files_only)
            dataloader = DataLoader(self.dataset, batch_size=8, shuffle=False)
            embedding_ls = []
            for step, data in enumerate(tqdm(dataloader)):
                inputs = tokenizer(data['utterance'], truncation=True,
padding=True, return_tensors='pt')
                inputs.to('cuda')
                # emb = model(**inputs, output_hidden_states=True,
return_dict=True).pooler_output.clone().detach().to('cpu')
                emb = model(**inputs, output_hidden_states=True,
return_dict=True).last_hidden_state[:, 0, :].clone().detach().to('cpu')
                embedding_ls.append( emb )
            self.embedding = torch.cat(embedding_ls)

        elif self.embedding is None:
            logger.info(f"**** Running {self.embedding_hf_modelcard} model to get
embeddings ****")
            model = AutoModel.from_pretrained(embedding_hf_modelcard,
local_files_only=self.local_files_only)
            model.to('cuda')
            tokenizer = AutoTokenizer.from_pretrained(embedding_hf_modelcard,
local_files_only=self.local_files_only)
            dataloader = DataLoader(self.dataset, batch_size=8, shuffle=False)
            embedding_ls = []
            for step, data in enumerate(tqdm(dataloader)):
                inputs = tokenizer(data['utterance'], truncation=True,
padding=True, return_tensors='pt')
                inputs.to('cuda')
                # emb = model(**inputs, output_hidden_states=True,
return_dict=True).pooler_output.clone().detach().to('cpu')
                emb = model(**inputs, output_hidden_states=True,
return_dict=True).last_hidden_state[:, 0, :].clone().detach().to('cpu')
                embedding_ls.append( emb )
```

```
            self.embedding = torch.cat(embedding_ls)

    def refine_embedding(self):
        pass

    def intersect(self, set2, top_k=None, debug=False):
        if self.embedding is None or set2.embedding is None:
            if self.embedding is None:
                self.get_embedding(embedding_hf_modelcard=self.embedding_hf_model
card)
            if set2.embedding is None:
                set2.get_embedding(embedding_hf_modelcard=self.embedding_hf_model
card)
        # do intersection
        # the only method offered is to measure similarity using cosine
similarity
        cossim_mat = F.normalize(self.embedding) @
F.normalize(set2.embedding).t()
        if isinstance(top_k, int):
            cossim_k, idx_set2 = torch.topk(cossim_mat, top_k, dim=1)
            cossim = cossim_k.mean(dim=1)
        elif top_k == 'all':
            cossim = cossim_mat.mean(dim=1)
        idx_to_set2 = np.argsort(-cossim).tolist()
        if debug==True:
            return (idx_to_set2, cossim)
        return idx_to_set2

    def difference(self, set2, top_k=None, debug=False):
        if self.embedding is None or set2.embedding is None:
            if self.embedding is None:
                self.get_embedding(embedding_hf_modelcard=self.embedding_hf_model
card)
            if set2.embedding is None:
                set2.get_embedding(embedding_hf_modelcard=self.embedding_hf_model
card)
        # do difference
        # the only method offered hear is to measure dis-similarity using cosine
similarity
        cossim_mat = - ( F.normalize(self.embedding) @
F.normalize(set2.embedding).t() )
        if isinstance(top_k, int):
            cossim_k, idx_set2 = torch.topk(cossim_mat, top_k, dim=1)
            cossim = cossim_k.mean(dim=1)
        elif top_k == 'all':
```

```
    cossim = cossim_mat.mean(dim=1)
idx_to_set2 = np.argsort(-cossim).tolist()
if debug==True:
    return (idx_to_set2, cossim)
return idx_to_set2
```