```
import random
import numpy as np
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
import pandas as pd
from transformers import DPRQuestionEncoder, DPRQuestionEncoderTokenizer
import torch.nn.functional as F
from datasets import load_dataset
from torch.utils.data.dataloader import DataLoader
import torch
from tqdm import tqdm
import string
import os
import shutil
import pickle

def id_generator(size=6):
    return ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in
range(size))

def DPR_eval(df, tokenizer, model):
    temp_code = id_generator(4)
    data_tempdir = 'data/temp_' + temp_code

    if not os.path.exists(data_tempdir):
        os.makedirs(data_tempdir)

    n_sample = 20
    df.drop_duplicates()
    # step 1. get samples and pool
    sample_d = {}
    pool_txt = []
    txt = set(df.utterance)
    n_d = {}
    for i in set(df.label):
        sample_d[i] = random.sample( set(df[df.label==i].utterance), n_sample )
        sample_df = pd.DataFrame({"utterance":sample_d[i]})
        sample_df.to_csv(data_tempdir+"/sample_{i}_temp.csv", index=False)
        pool_txt_i = txt - set(sample_d[i])
        n_d[i] = len(pool_txt_i)
        pool_txt += list( pool_txt_i )
    pool_df = pd.DataFrame({"utterance":pool_txt})
    pool_df.to_csv(data_tempdir+"/pool_temp.csv", index=False)
```

```
    # get embeddings of pool and samples
    inter_dfi = {}
    diff_dfi = {}
    sample_emb = {}
    pool_emb = get_emb(data_tempdir+"/pool_temp.csv", tokenizer, model)
    for i in set(df.label):
        sample_emb[i] = get_emb(data_tempdir+"/sample_{i}_temp.csv", tokenizer,
model)
        cossim_mat = F.normalize(pool_emb) @ F.normalize(sample_emb[i]).t()
        cossim = cossim_mat.mean(dim=1)

        # intersection
        idx = (-cossim.argsort()[:n_d[i]])
        inter_dfi[i] =
df[df.utterance.isin( pool_df.iloc[idx].utterance.to_list() )]

        # difference
        top_n_diff = len(pool_txt) - n_d[i]
        idx = (cossim.argsort())[:top_n_diff]
        diff_dfi[i] =
df[df.utterance.isin( pool_df.iloc[idx].utterance.to_list() )]

    # evaluate
    pred_label_inter = []
    true_label_inter = []
    pred_label_diff = []
    true_label_diff = []
    for i in set(df.label):
        # inter
        pred_label_inter += [i] * inter_dfi[i].shape[0]
        true_label_inter += inter_dfi[i].label.to_list()
        # diff
        other_labels = set(df.label) -set([i])
        replace = {l: f"not_{i}" for l in other_labels}
        diff_dfi[i] = diff_dfi[i].replace({"label": replace})
        pred_label_diff += [f"not_{i}"] * diff_dfi[i].shape[0]
        true_label_diff += diff_dfi[i].label.to_list()
    res_inter = precision_recall_fscore_support(true_label_inter,
pred_label_inter, average='weighted', zero_division=0.0)
    acc_inter = accuracy_score(true_label_inter, pred_label_inter,
normalize=True)
    res_inter = (acc_inter,) + res_inter[:2] +
(2*res_inter[0]*res_inter[1]/(res_inter[0] + res_inter[1]),)
```

```
    res_diff = precision_recall_fscore_support(true_label_diff, pred_label_diff,
average='weighted', zero_division=0.0)
    acc_diff = accuracy_score(true_label_diff, pred_label_diff, normalize=True)
    res_diff = (acc_diff,) + res_diff[:2] +
(2*res_diff[0]*res_diff[1]/(res_diff[0] + res_diff[1]),)

    shutil.rmtree(data_tempdir)
    return res_inter, res_diff

def get_emb(data_path, tokenizer, model):
    model.to('cuda')
    dataset = load_dataset('csv', data_files=data_path)
    key = list(dataset.keys())[0]
    dataset = dataset[key]
    dataloader = DataLoader(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')
        embedding_ls.append( emb )
    embedding = torch.cat( embedding_ls )
    return embedding

def DPR_eval_n_rep(df, n_rep):
    tokenizer = DPRQuestionEncoderTokenizer.from_pretrained('facebook/dpr-
question_encoder-single-nq-base')
    model = DPRQuestionEncoder.from_pretrained('facebook/dpr-question_encoder-
single-nq-base', return_dict=True)
    res_inter = np.zeros([n_rep, 4])
    res_diff = np.zeros([n_rep, 4])
    for i in range(n_rep):
        res_inter[i], res_diff[i] = DPR_eval(df, tokenizer, model)
    res_inter = pd.DataFrame(res_inter, columns=['acc', 'precision', 'recall',
'F1'])
    res_inter = res_inter.mean().to_frame('Intersection').T

    res_diff = pd.DataFrame(res_diff, columns=['acc', 'precision', 'recall',
'F1'])
    res_diff = res_diff.mean().to_frame('Difference').T
    return res_inter, res_diff

df_dir = "data/ag_news/description_3000.csv"
```

```
df_dirs = ["data/ag_news/title_3000.csv", "data/ag_news/description_3000.csv",
"data/FPB/all.csv", "data/banking77/all.csv", "data/fb/en_all.csv"]
df_tags = ["AGT", "AGD", "FPB", "Banking77", "FMTOD"]
n_rep = 5
res_dict = {}
for i in range(len(df_dirs)):
    df = pd.read_csv(df_dirs[i])
    res_inter, res_diff = DPR_eval_n_rep(df, n_rep)
    res_dict[df_tags[i]] = pd.concat([res_inter, res_diff])

    with open(f'results/res_DPR_{df_tags[i]}.pickle', 'wb') as f:
        pickle.dump(res_dict, f)

print("Done!")
```