```
"""
This script is for evaluation
"""
import pandas as pd
import numpy as np
from sklearn.metrics import precision_recall_fscore_support
import logging
import set_ops as setops
import setops_sup
from transformers import AutoConfig, AutoModel, AutoTokenizer, PreTrainedModel
import torch
import random
import os
import argparse
import string

# 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)

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

def eval_setops_unsup_intersect(df, model_name, data_tempdir, n_sample=50,
n_intent=3, top_k=1, local_files_only=False):
    # do set operations
    intents = list(df.label.value_counts().index)
    intents = intents[:n_intent]
    sample_set = dict()
    pool = []
    for intent in intents:
        dfi = df[df['label']==intent].sample(frac=1)
        sample = dfi.iloc[:n_sample]
        sample.to_csv(data_tempdir+f'/{intent}.csv', index=False)
```

```
        sample_set[intent] = setops.Set(data_path=data_tempdir+f'/{intent}.csv',
embedding_hf_modelcard=model_name, local_files_only=local_files_only)
        pool.append(dfi.iloc[n_sample:])
    pool_df = pd.concat(pool).sample(frac=1)
    pool_df.to_csv(data_tempdir+'/pool.csv', index=False)
    pool = setops.Set(data_path=data_tempdir+'/pool.csv',
embedding_hf_modelcard=model_name, local_files_only=local_files_only)
    counts = pool_df.label.value_counts()

    pool_set_ls= []
    for intent in intents:
        idx, cossim = pool.intersect(sample_set[intent], top_k=top_k, debug=True)
        pool_df[f'cossim_{intent}'] = cossim
        pool_set = pool_df.iloc[idx]
        n = counts[intent]
        pool_set = pool_set.iloc[:n]
        pool_set['pred'] = intent
        pool_set_ls.append(pool_set)
    pool_set = pd.concat(pool_set_ls)
    res = precision_recall_fscore_support(pool_set['label'].to_list(),
pool_set['pred'].to_list(), average='weighted')
    acc =
pool_set[pool_set['label']==pool_set['pred']].shape[0]/pool_set.shape[0]
    return (acc,) + res[:3]


def eval_setops_unsup_difference(df, model_name, data_tempdir, n_sample=50,
n_intent=3, top_k=1, local_files_only=False):
    intents = list(df.label.value_counts().index)
    intents = intents[:n_intent]
    sample_set = dict()
    pool = []
    for intent in intents:
        dfi = df[df['label']==intent].sample(frac=1)
        sample = dfi.iloc[:n_sample]
        sample.to_csv(data_tempdir+f'/{intent}.csv', index=False)
        sample_set[intent] = setops.Set(data_path=data_tempdir+f'/{intent}.csv',
embedding_hf_modelcard=model_name, local_files_only=local_files_only)
        pool.append(dfi.iloc[n_sample:])
    pool_df = pd.concat(pool).sample(frac=1)
    pool_df.to_csv(data_tempdir+'/pool.csv', index=False)
    pool = setops.Set(data_path=data_tempdir+'/pool.csv',
embedding_hf_modelcard=model_name, local_files_only=local_files_only)
    counts = pool_df.label.value_counts()
```

```
    pool_set_ls = []
    for intent in intents:
        idx, cossim = pool.difference(sample_set[intent], top_k=top_k,
debug=True)
        pool_df[f'cossim_{intent}'] = cossim
        pool_set = pool_df.iloc[idx]
        n = sum(counts) - counts[intent]
        pool_set = pool_set.iloc[:n]
        intents_m1 = [x for x in intents if x!=intent]
        pool_set['label'] = pool_set.label.str.replace('|'.join(intents_m1),
'not_'+intent, regex=True)
        pool_set['pred'] = 'not_'+intent
        pool_set_ls.append(pool_set)
    pool_set = pd.concat(pool_set_ls)
    res = precision_recall_fscore_support(pool_set['label'].to_list(),
pool_set['pred'].to_list(), average='weighted')
    acc =
pool_set[pool_set['label']==pool_set['pred']].shape[0]/pool_set.shape[0]
    return (acc,) + res[:3]


def eval_SetCSE_intersect(df, model_name, data_tempdir, model_tempdir,
n_sample=50, n_intent=3, top_k=1, train_epoch=40, local_files_only=False):
    args = setops_sup.Training_args(model_name=model_name, bs=48,
train_epoch=train_epoch, output_dir=model_tempdir)
    trainer = setops_sup.SetCSETrainer(args, local_files_only=local_files_only)

    intents = list(df.label.value_counts().index)
    intents = intents[:n_intent]
    sampleset_dict = dict()
    sampleset_ls = []

    # prepare the sample sets and pool set
    pool = []
    for intent in intents:
        dfi = df[df['label']==intent].sample(frac=1)
        sample = dfi.iloc[:n_sample]
        sample.to_csv(data_tempdir+f'/{intent}.csv', index=False)
        sample_set = setops_sup.Set(data_path=data_tempdir+f'/{intent}.csv')
        sampleset_dict[intent] = sample_set
        sampleset_ls.append(sample_set)
        pool.append(dfi.iloc[n_sample:])
    pool_df = pd.concat(pool).sample(frac=1)
    pool_df.to_csv(data_tempdir+'/pool.csv', index=False)
    pool = setops_sup.Set(data_path=data_tempdir+'/pool.csv')
```

```
    counts = pool_df.label.value_counts()

    # do SetCSE training
    trainer.SetCSEtrain(sampleset_ls)

    # get tokenizer and model
    tokenizer = AutoTokenizer.from_pretrained(args.model_name)
    model = AutoModel.from_pretrained(args.output_dir)

    # do intersection
    pool_set_ls= []
    for intent in intents:
        idx, cossim = pool.intersect(sampleset_dict[intent], tokenizer, model,
top_k=top_k, debug=True)
        pool_df[f'cossim_{intent}'] = cossim
        pool_set = pool_df.iloc[idx]
        n = counts[intent]
        pool_set = pool_set.iloc[:n]
        pool_set['pred'] = intent
        pool_set_ls.append(pool_set)
    pool_set = pd.concat(pool_set_ls)
    res = precision_recall_fscore_support(pool_set['label'].to_list(),
pool_set['pred'].to_list(), average='weighted')
    acc =
pool_set[pool_set['label']==pool_set['pred']].shape[0]/pool_set.shape[0]
    return (acc,) + res[:3]

def eval_SetCSE_difference(df, model_name, data_tempdir, model_tempdir,
n_sample=50, n_intent=3, top_k=1, train_epoch=40, local_files_only=False):
    args = setops_sup.Training_args(model_name=model_name, bs=48,
train_epoch=train_epoch, output_dir=model_tempdir)
    trainer = setops_sup.SetCSETrainer(args, local_files_only=local_files_only)
    intents = list(df.label.value_counts().index)
    intents = intents[:n_intent]
    sampleset_dict = dict()
    sampleset_ls = []
    # prepare the sample sets and pool set
    pool = []
    for intent in intents:
        dfi = df[df['label']==intent].sample(frac=1)
        sample = dfi.iloc[:n_sample]
        sample.to_csv(data_tempdir+f'/{intent}.csv', index=False)
        sample_set = setops_sup.Set(data_path=data_tempdir+f'/{intent}.csv')
        sampleset_dict[intent] = sample_set
        sampleset_ls.append(sample_set)
```

```
        pool.append(dfi.iloc[n_sample:])
    pool_df = pd.concat(pool).sample(frac=1)
    pool_df.to_csv(data_tempdir+'/pool.csv', index=False)
    pool = setops_sup.Set(data_path=data_tempdir+'/pool.csv')
    counts = pool_df.label.value_counts()

    # do SetCSE training
    trainer.SetCSEtrain(sampleset_ls)

    # get tokenizer and model
    tokenizer = AutoTokenizer.from_pretrained(args.model_name)
    model = AutoModel.from_pretrained(args.output_dir)

    # do difference
    pool_set_ls= []
    for intent in intents:
        idx, cossim = pool.difference(sampleset_dict[intent], tokenizer, model,
top_k=top_k, debug=True)
        pool_df[f'cossim_{intent}'] = cossim
        pool_set = pool_df.iloc[idx]
        n = sum(counts) - counts[intent]
        pool_set = pool_set.iloc[:n]
        intent_m1 = [x for x in intents if x!=intent]
        pool_set['label'] = pool_set.label.str.replace('|'.join(intent_m1),
'not_'+intent, regex=True)
        pool_set['pred'] = 'not_'+intent
        pool_set_ls.append(pool_set)
    pool_set = pd.concat(pool_set_ls)
    res = precision_recall_fscore_support(pool_set['label'].to_list(),
pool_set['pred'].to_list(), average='weighted')
    acc =
pool_set[pool_set['label']==pool_set['pred']].shape[0]/pool_set.shape[0]
    return (acc,) + res[:3]



# Step 1. Read arguments:
parser = argparse.ArgumentParser()
parser.add_argument("-mn", "--model_name", default='princeton-nlp/sup-simcse-
bert-base-uncased', help="Name of the model for evaluation")
parser.add_argument("-dp", "--data_path", default='data/fb/en_all.csv',
help="Dataset for evaluation")
parser.add_argument("-nr", "--n_rep", default=5, help="Number to times to repeat
each experiment.")
```

```
parser.add_argument("-te", "--train_epoch", default=60, help="Number of training
epoch.")
parser.add_argument("-ni", "--n_intent", default=3, help="Number of sample set.")
parser.add_argument("-ns", "--n_sample", default=20, help="Sample set size.")
args = parser.parse_args()

data_path = args.data_path
df = pd.read_csv(data_path)
# ['data/fb/en_all.csv', 'data/agnews/title_3000.csv',
'data/agnews/description_3000.csv', 'data/FPB/all.csv']
# ["princeton-nlp/sup-simcse-bert-base-uncased", "princeton-nlp/sup-simcse-
roberta-base", "voidism/diffcse-bert-base-uncased-sts", "voidism/diffcse-roberta-
base-sts"]
# ["bert-base-uncased", "roberta-base", "facebook/contriever", "Muennighoff/SGPT-
125M-weightedmean-nli-bitfit"]

n_rep = int(args.n_rep)
n_sample = int(args.n_sample)
n_intent = int(args.n_intent)
top_k = n_sample
train_epoch = int(args.train_epoch)
model_name = args.model_name
if 'MCSE' in model_name:
    local_files_only = True
else:
    local_files_only = False


# Step 2. Make data and model temp directory:
temp_code = id_generator(4)
data_tempdir = 'data/temp_' + temp_code
model_tempdir = 'models/temp_' + temp_code

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

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



# Step 3. Do evaluations
set_seed(42)

# 3.1 intersect
```

```
inter_res_unsup = np.zeros([n_rep, 4])
for i in range(n_rep):
    inter_res_unsup[i] = eval_setops_unsup_intersect(df, model_name,
data_tempdir, n_sample=n_sample, n_intent=n_intent, top_k=top_k,
local_files_only=local_files_only)
    print(f'Unsup finished: {i+1}!')

inter_res_sup = np.zeros([n_rep, 4])
for i in range(n_rep):
    inter_res_sup[i] = eval_SetCSE_intersect(df, model_name, data_tempdir,
model_tempdir, n_sample=n_sample, n_intent=n_intent, top_k=top_k,
train_epoch=train_epoch, local_files_only=local_files_only)
    print(f'Sup finished: {i+1}!')


# 3.2 difference
diff_res_unsup = np.zeros([n_rep, 4])
for i in range(n_rep):
    diff_res_unsup[i] = eval_setops_unsup_difference(df, model_name,
data_tempdir, n_sample=n_sample, n_intent=n_intent, top_k=top_k,
local_files_only=local_files_only)
    print(f'Unsup finished: {i+1}!')

diff_res_sup = np.zeros([n_rep, 4])
for i in range(n_rep):
    diff_res_sup[i] = eval_SetCSE_difference(df, model_name, data_tempdir,
model_tempdir, n_sample=n_sample, n_intent=n_intent, top_k=top_k,
train_epoch=train_epoch, local_files_only=local_files_only)
    print(f'Sup finished: {i+1}!')


print(f'data_path={data_path}')
print(f'model_name={model_name}')

print('Unsupervised intersect:')
inter_res_unsup = pd.DataFrame(inter_res_unsup, columns=['acc', 'precision',
'recall', 'F1'])
print(inter_res_unsup)

print('Supervised intersect:')
inter_res_sup = pd.DataFrame(inter_res_sup, columns=['acc', 'precision',
'recall', 'F1'])
print(inter_res_sup)

print('Unsupervised difference:')
```

```
diff_res_unsup = pd.DataFrame(diff_res_unsup, columns=['acc', 'precision',
'recall', 'F1'])
print(diff_res_unsup)

print('Supervised difference:')
diff_res_sup = pd.DataFrame(diff_res_sup, columns=['acc', 'precision', 'recall',
'F1'])
print(diff_res_sup)


inter_res_unsup = inter_res_unsup.mean().to_frame('Set_Intersect').T
inter_res_sup = inter_res_sup.mean().to_frame('SetCSE_Intersect').T
diff_res_unsup = diff_res_unsup.mean().to_frame('Set_Difference').T
diff_res_sup = diff_res_sup.mean().to_frame('SetCSE_Difference').T

res = pd.concat([inter_res_unsup, inter_res_sup, diff_res_unsup, diff_res_sup])
print(f'data_path={data_path}')
print(f'model_name={model_name}')
print("Final result:")
print(res)
```