```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
import torch
import torch.nn.functional as F
import numpy as np
from sklearn.metrics import precision_recall_fscore_support
from scipy import sparse



class BM25(object):
    def __init__(self, b=0.75, k1=1.6):
        self.vectorizer = TfidfVectorizer(norm=None, smooth_idf=False)
        self.b = b
        self.k1 = k1

    def fit(self, X):
        """ Fit IDF to documents X """
        self.vectorizer.fit(X)
        y = super(TfidfVectorizer, self.vectorizer).transform(X)
        self.avdl = y.sum(1).mean()

    def transform(self, q, X):
        """ Calculate BM25 between query q and documents X """
        b, k1, avdl = self.b, self.k1, self.avdl

        # apply CountVectorizer
        X = super(TfidfVectorizer, self.vectorizer).transform(X)
        len_X = X.sum(1).A1
        q, = super(TfidfVectorizer, self.vectorizer).transform([q])
        assert sparse.isspmatrix_csr(q)

        # convert to csc for better column slicing
        X = X.tocsc()[:, q.indices]
        denom = X + (k1 * (1 - b + b * len_X / avdl))[:, None]
        # idf(t) = log [ n / df(t) ] + 1 in sklearn, so it need to be coneverted
        # to idf(t) = log [ n / df(t) ] with minus 1
        idf = self.vectorizer._tfidf.idf_[None, q.indices] - 1.
        numer = X.multiply(np.broadcast_to(idf, X.shape)) * (k1 +
1)
        return (numer / denom).sum(1).A1


def tfidf_intersect_eval(n_intent=3, n_sample=20, top_k=20, dp=""):
    df = pd.read_csv(dp)
```

```
    # Step 1. Get embeddings
    samples = []
    rests = []
    intents = list(df.label.value_counts().index)
    intents = intents[:n_intent]
    for intent in intents:
        dfi = df[df['label']==intent].sample(frac=1)
        samples.append(dfi[:n_sample])
        rests.append(dfi[n_sample:])
    samples = pd.concat(samples)
    df_U = pd.concat(rests)

    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(samples.utterance)
    X = torch.tensor(X.toarray())
    U = vectorizer.transform(df_U.utterance)
    U = torch.tensor(U.toarray())

    # Step 2. Do intersection and evaluation
    preds = []
    for i in range(len(intents)):
        query = X[i*n_sample:(i+1)*n_sample]
        cossim_mat = F.normalize(U) @ F.normalize(query).t()

        n_intenti = df_U[df_U['label']==intents[i]].shape[0]

        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()
        idx_intenti = idx_to_set2[:n_intenti]
        df_Ui = df_U.iloc[idx_intenti]
        df_Ui['pred'] = intents[i]
        preds.append(df_Ui)

    df_Upred = pd.concat(preds)
    res = precision_recall_fscore_support(df_Upred['label'].to_list(),
df_Upred['pred'].to_list(), average='weighted')
    acc =
df_Upred[df_Upred['label']==df_Upred['pred']].shape[0]/df_Upred.shape[0]
```

```
    return (acc,) + res[:3]


def tfidf_diff_eval(n_intent=3, n_sample=20, top_k=20, dp=""):
    df = pd.read_csv(dp)

    # Step 1. Get embeddings
    samples = []
    rests = []
    intents = list(df.label.value_counts().index)
    intents = intents[:n_intent]
    for intent in intents:
        dfi = df[df['label']==intent].sample(frac=1)
        samples.append(dfi[:n_sample])
        rests.append(dfi[n_sample:])
    samples = pd.concat(samples)
    rests = pd.concat(rests)
    df = pd.concat([samples, rests])
    df_U = df.iloc[n_intent*n_sample:]

    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(samples.utterance)
    X = torch.tensor(X.toarray())
    U = vectorizer.transform(df_U.utterance)
    U = torch.tensor(U.toarray())

    # Step 2. Do diff and evaluation
    preds = []
    for i in range(len(intents)):
        query = X[i*n_sample:(i+1)*n_sample]
        cossim_mat = F.normalize(U) @ F.normalize(query).t()

        n_intenti = df_U[df_U['label']==intents[i]].shape[0]

        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()
        idx_intenti = idx_to_set2[n_intenti:]
        df_Ui = df_U.iloc[idx_intenti]
        intent_m1 = [x for x in intents if x!=intents[i]]
```

```
        df_Ui['label'] = df_Ui.label.str.replace('|'.join(intent_m1),
'not_'+intent)
        df_Ui['pred'] = 'not_'+intents[i]
        preds.append(df_Ui)

    df_Upred = pd.concat(preds)
    res = precision_recall_fscore_support(df_Upred['label'].to_list(),
df_Upred['pred'].to_list(), average='weighted')
    acc =
df_Upred[df_Upred['label']==df_Upred['pred']].shape[0]/df_Upred.shape[0]

    return (acc,) + res[:3]


def bm25_intersect_eval(n_intent=3, n_sample=20, top_k=20, dp=""):
    df = pd.read_csv(dp)

    # Step 1. Get embeddings
    samples = []
    rests = []
    intents = list(df.label.value_counts().index)
    intents = intents[:n_intent]
    for intent in intents:
        dfi = df[df['label']==intent].sample(frac=1)
        samples.append(dfi[:n_sample])
        rests.append(dfi[n_sample:])
    samples = pd.concat(samples)
    df_U = pd.concat(rests)

    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(samples.utterance)
    X = torch.tensor(X.toarray())
    U = vectorizer.transform(df_U.utterance)
    U = torch.tensor(U.toarray())

    # Step 2. Do intersection and evaluation
    preds = []
    for i in range(len(intents)):
        query = X[i*n_sample:(i+1)*n_sample]
        cossim_mat = F.normalize(U) @ F.normalize(query).t()

        n_intenti = df_U[df_U['label']==intents[i]].shape[0]

        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()
        idx_intenti = idx_to_set2[:n_intenti]
        df_Ui = df_U.iloc[idx_intenti]
        df_Ui['pred'] = intents[i]
        preds.append(df_Ui)

    df_Upred = pd.concat(preds)
    res = precision_recall_fscore_support(df_Upred['label'].to_list(),
df_Upred['pred'].to_list(), average='weighted')
    acc =
df_Upred[df_Upred['label']==df_Upred['pred']].shape[0]/df_Upred.shape[0]

    return (acc,) + res[:3]

n_rep = 5
dp = "data/ag_news/description_3000.csv"
n_intent = 3
n_sample = 20
top_k = n_sample

# inter_res_tfidf = np.zeros([n_rep, 4])
# for i in range(n_rep):
#     inter_res_tfidf[i] = tfidf_intersect_eval(n_intent=n_intent,
n_sample=n_sample, top_k=top_k, dp=dp)
#     print(f'Intersect finished: {i+1}!')

# inter_res_tfidf = pd.DataFrame(inter_res_tfidf, columns=['acc', 'precision',
'recall', 'F1'])
# inter_res_tfidf = inter_res_tfidf.mean().to_frame('Set_Intersect').T


# diff_res_tfidf = np.zeros([n_rep, 4])
# for i in range(n_rep):
#     diff_res_tfidf[i] = tfidf_diff_eval(n_intent=n_intent, n_sample=n_sample,
top_k=top_k, dp=dp)
#     print(f'Intersect finished: {i+1}!')

# diff_res_tfidf = pd.DataFrame(diff_res_tfidf, columns=['acc', 'precision',
'recall', 'F1'])
# diff_res_tfidf = diff_res_tfidf.mean().to_frame('Set_Intersect').T
```

```
# print(f"Inter_res_tfidf:\n {inter_res_tfidf}")
# print("\n")
# print(f"Diff_res_tfidf:\n {diff_res_tfidf}")


df = pd.read_csv(dp)

# Step 1. Get embeddings
samples = []
rests = []
intents = list(df.label.value_counts().index)
intents = intents[:n_intent]
for intent in intents:
    dfi = df[df['label']==intent].sample(frac=1)
    samples.append(dfi[:n_sample])
    rests.append(dfi[n_sample:])
samples = pd.concat(samples)
print(samples)
df_U = pd.concat(rests)

vectorizer = BM25()
X = vectorizer.fit(samples.utterance)
X = torch.tensor(X.toarray())
U = vectorizer.transform(df_U.utterance)
U = torch.tensor(U.toarray())

# Step 2. Do intersection and evaluation
preds = []
for i in range(len(intents)):
    query = X[i*n_sample:(i+1)*n_sample]
    cossim_mat = F.normalize(U) @ F.normalize(query).t()

    n_intenti = df_U[df_U['label']==intents[i]].shape[0]

    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()
    idx_intenti = idx_to_set2[:n_intenti]
    df_Ui = df_U.iloc[idx_intenti]
    df_Ui['pred'] = intents[i]
    preds.append(df_Ui)
```

```
df_Upred = pd.concat(preds)
res = precision_recall_fscore_support(df_Upred['label'].to_list(),
df_Upred['pred'].to_list(), average='weighted')
acc = df_Upred[df_Upred['label']==df_Upred['pred']].shape[0]/df_Upred.shape[0]

print(res)
print(acc)
```