```
import os
import torch
import random
import torch.nn as nn
import numpy as np
import bisect
from sklearn.datasets import fetch_20newsgroups
from torch.utils.data import DataLoader, Dataset
!pip install transformers==3.0.2
!pip install sentence-transformers==0.3.3
```

```
Requirement already satisfied: transformers==3.0.2 in /usr/local/lib/python3.7/dist-packages (3.0.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (4.62.3)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (2019.12.
Requirement already satisfied: tokenizers==0.8.1.rc1 in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (0.8.
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (2.23.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (3.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (1.19.5)
Requirement already satisfied: sentencepiece!=0.1.92 in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (0.1.
Requirement already satisfied: sacremoses in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (0.0.46)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from transformers==3.0.2) (21.0)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->transformers==3.0.2)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transformers==3.0.2)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers==3.0.2) (2.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers==3.0.2
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers==3.0.2) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers==3.0.2) (1.0.1)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers==3.0.2) (1.15.0)
Requirement already satisfied: sentence-transformers==0.3.3 in /usr/local/lib/python3.7/dist-packages (0.3.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from sentence-transformers==0.3.3) (4.62.3)
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (from sentence-transformers==0.3.3) (3.2.5)
Requirement already satisfied: torch>=1.2.0 in /usr/local/lib/python3.7/dist-packages (from sentence-transformers==0.3.3) (1.9.
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from sentence-transformers==0.3.3) (1.19.5)
Requirement already satisfied: transformers>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from sentence-transformers==0.3.3
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from sentence-transformers==0.3.3) (1.4.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from sentence-transformers==0.3.3) (0.22
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch>=1.2.0->sentence-transfo
Requirement already satisfied: sacremoses in /usr/local/lib/python3.7/dist-packages (from transformers>=3.0.2->sentence-transfo
Requirement already satisfied: tokenizers==0.8.1.rc1 in /usr/local/lib/python3.7/dist-packages (from transformers>=3.0.2->sente
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers>=3.0.2->sentence-transform
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers>=3.0.2->sentence-transform
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers>=3.0.2->sentence-
Requirement already satisfied: sentencepiece!=0.1.92 in /usr/local/lib/python3.7/dist-packages (from transformers>=3.0.2->sente
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from transformers>=3.0.2->sentence-transfor
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nltk->sentence-transformers==0.3.3) (1.15.0)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->transformers>=3.0.2-
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transformers>=3.0.2-
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers>=3.0.2->sent
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers>=3.0.2
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers>=3.0.2->sentence-
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers>=3.0.2->sentence
```

```
torch.manual_seed(2020)
np.random.seed(2020)
random.seed(2020)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)

if device.type == "cuda":
    print(torch.cuda.get_device_name())
```

```
cuda
Tesla K80
```

```
from sentence_transformers import SentenceTransformer
from sentence_transformers import evaluation
import random
from collections import defaultdict


model = SentenceTransformer('distilbert-base-nli-mean-tokens')


newsgroups_train = fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'))
newsgroups_test = fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'))


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
```

```
newsgroups_train.target.shape
newsgroups_train.target[:10]
```

```
array([ 7,  4,  4,  1, 14, 16, 13,  3,  2,  4])
```

```
from sentence_transformers.readers import InputExample
from tqdm import tqdm


class TripletDataLoader(Dataset):
  """
  custom loading class, compatible with Pytorch DataLoader, that generates training triplets (anchor, positive example negative examp

  """
    def __init__(self, model, chunk_size=1000, get_positive=True, get_negative=True):

        super(TripletDataLoader, self).__init__()
        self.grouped_inputs = []
        self.model = model
        self.num_labels = 0
        self.max_processes = min(max_processes, cpu_count())
        self.chunk_size = chunk_size
        self.groups_right_border = []
        self.grouped_labels = []


    # fetching dataset into train, dev, test
    group_items = []
    trainset = "train"
    testset = "test"
    validation_rate = 0.01
    for name in [trainset, testset]:
        file = fetch_20newsgroups(subset=name, remove=(
            'headers', 'footers', 'quotes'), shuffle=True)

        examples = []
        guid = 1
        # retriving text data and target sample
        for text, target in zip(file.data, file.target):
            guid += 1
            examples.append(InputExample(
                guid=guid, texts=[text], label=target))
        group_items.append(examples)

    train_set, test_set = group_items
    dev_set = None

    if validation_rate > 0:
        size = int(len(train_set) * validation_rate)
        dev_set = train_set[-size:]
        train_set = train_set[:-size]

    self.dataset = train_set, dev_set, test_set

    self.prepare_traindataset(self.dataset[0], model)
    self.idxs = np.arange(len(self.grouped_inputs))

    self.get_positive = get_positive
    self.get_negative = get_negative


def prepare_traindataset(self, examples, model):
  """
  main logic to prepare the training triplets from dataset by limiting the one sentance per class
  """
    inputs = []
    labels = []
    label_sent_mapping = {}
    s = 0
    label_type = None

    print("Start tokenization")
    tokenized_texts = [self.tokenize_example(example) for example in examples]

    for ex_index, example in enumerate(tqdm(examples, desc="Convert dataset")):
        if label_type is None:
```

```
    if label_type is None:
        if isinstance(example.label, int):
            label_type = torch.long
        elif isinstance(example.label, float):
            label_type = torch.float
    tokenized_text = tokenized_texts[ex_index][0]
    # if attribute max_seq_length is available and valid
    if hasattr(self.model, 'max_seq_length') and self.model.max_seq_length is not None and self.model.max_seq_length > 0 and
        s += 1

    if example.label in label_sent_mapping:
        label_sent_mapping[example.label].append(ex_index)
    else:
        label_sent_mapping[example.label] = [ex_index]

    inputs.append(tokenized_text)
    labels.append(example.label)

distinct_labels = list(label_sent_mapping.keys())
for i in range(len(distinct_labels)):
    label = distinct_labels[i]
    if len(label_sent_mapping[label]) >= 2:
        self.grouped_inputs.extend(
            [inputs[j] for j in label_sent_mapping[label]])
        self.grouped_labels.extend(
            [labels[j] for j in label_sent_mapping[label]])
        self.groups_right_border.append(len(self.grouped_inputs))
        self.num_labels += 1

    self.grouped_labels = torch.tensor(
        self.grouped_labels, dtype=label_type)

def tokenize_example(self, example):
    return [self.model.tokenize(text) for text in example.texts]

def __getitem__(self, item):
    if not self.get_positive and not self.get_negative:
        return [self.grouped_inputs[item]], self.grouped_labels[item]

    anchor = self.grouped_inputs[item]
    group_idx = bisect.bisect_right(self.groups_right_border, item)
    left_border = 0 if group_idx == 0 else self.groups_right_border[group_idx - 1]
    right_border = self.groups_right_border[group_idx]

    if self.get_positive:
        positive_item_idx = np.random.choice(np.concatenate(
            [self.idxs[left_border:item], self.idxs[item + 1:right_border]]))
        positive = self.grouped_inputs[positive_item_idx]
    else:
        positive = []

    if self.get_negative:
        negative_item_idx = np.random.choice(np.concatenate(
            [self.idxs[0:left_border], self.idxs[right_border:]]))
        negative = self.grouped_inputs[negative_item_idx]
    else:
        negative = []

    return [anchor, positive, negative], self.grouped_labels[item]

def __len__(self):
    return len(self.grouped_inputs)



def generate_triplets_from_dataset(input_examples):
    """
    generate triplets for each anchor input by taking positive sample from same class and negative sample from any of rest of the cla
    """
    triplets = []
    label2sentence = defaultdict(list)
    for example in input_examples:
        label2sentence[example.label].append(example)

    for example in input_examples:
        anchor = example

        if len(label2sentence[example.label]) < 2:
            continue
```

```
        positive = None
        while positive is None or positive.guid == anchor.guid:
            positive = random.choice(label2sentence[example.label])

        negative = None
        while negative is None or negative.label == anchor.label:
           negative = random.choice(input_examples)

        triplets.append(InputExample(texts=[anchor.texts[0], positive.texts[0], negative.texts[0]]))

    return triplets


train_ds = TripletDataLoader(model,get_positive=True, get_negative=True)
```

```
Start tokenization
Convert dataset: 100%|██████████| 11201/11201 [00:00<00:00, 305660.99it/s]
```

```
train_loader = DataLoader(train_ds, shuffle=True, batch_size=16)
```

```
train_loss = nn.TripletMarginLoss(margin=1.0)
```

```
from sentence_transformers.evaluation import TripletEvaluator


dev_set = train_ds.dataset[1]
dev_evaluator = TripletEvaluator.from_input_examples(generate_triplets_from_dataset(dev_set),  name='dev')


from sentence_transformers import LoggingHandler, losses
train_loss = losses.TripletLoss(model)


os.getcwd()
```

```
'/content'
```

```
import os
os.chdir("/content/drive/MyDrive/vocads_models")


warmup_steps = int(len(train_loader) * 1 / 8 * 0.1)

model.fit(train_objectives=[(train_loader, train_loss)],
          evaluator=dev_evaluator,
          epochs=1,
          evaluation_steps=1000,
          warmup_steps=warmup_steps,
          output_path="/content/drive/MyDrive/vocads_models/fine_tune_distilbert_nli_mean_token_")
```

```
Epoch: 100%                                    1/1 [18:01<00:00, 1081.74s/it]

Iteration: 100%                                701/701 [17:57<00:00, 1.26s/it]
```