TO DO: 1 - create a custom loading class that generates training examples (anchor, positive examples, negatives examples) from 20 News groups (SentenceLabelDataset) https://github.com/UKPLab/sentence-(https://github.com/UKPLab/sentence-)
transformers/blob/6fcfdfb30f9dfcc5fb978c97ce02941a7aa6ba63/sentence_transf
ormers/datasets/SentenceLabelDataset.py.

2 - Build a training pipeline and finetune a "distilbert-base-nli-mean-token" model with the custom TripletLoss class (triplet generation strategy is what matters)

3 - fine an Approximate Nearest Neighbors library and explain my choose in few words

Build a basic classification pipeline:

- vectorization of the training set with finetune sBert model
- index all this vector with the Approximate Nearest Neighbors library (ANN)
- Build a knn classifier where the new text input get the same labed as that closest index from the index
- Benchmark the pipeline with the test set
- Compare the model with the pretrained sBert

5 - Create a simple REST API that serves this prediction via a "/predict" route (Given a input text it will predict one of the 20 News labels)

6 - Create a dockerfile to wrap the code in a docker container

Entrée [ ]:

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

Entrée [ ]:

```
import os
os.chdir('/content/drive/My Drive/Ubisoft_takehome_challenge_MLE')
```

Entrée [ ]:

```
!pip install -U sentence-transformers
```

Fine tuning SentenceTranformer

We first create a custom loading class that generates training examples (anchor, positive examples, negatives examples) from 20 News groups.

Given a input example(anchor), a postive example will be an example from the same label as input example. Negative example will be an an example from an other label

Entrée [9]:

```
from sklearn.datasets import fetch_20newsgroups
from torch.utils.data import Dataset
from typing import List
import bisect
import torch
import logging
import numpy as np
from tqdm import tqdm
from sentence_transformers import SentenceTransformer
from sentence_transformers.readers import InputExample
from multiprocessing import Pool, cpu_count
import multiprocessing


class Fetch20newsLabelDataset(Dataset):
    """
    Dataset for training with triplet loss.
    This dataset takes a list of sentences grouped by their label and uses this grou
    positive example from the same group and a negative example from the other sente

    This dataset should be used in combination with dataset_reader.LabelSentenceRead

    One iteration over this dataset selects every sentence as anchor once.

    This also uses smart batching like SentenceDataset.
    """

    def __init__(self,
                 model: SentenceTransformer,
                 provide_positive: bool = True,
                 provide_negative: bool = True,
                 parallel_tokenization: bool = True,
                 max_processes: int = 4,
                 chunk_size: int = 5000):
        """
        Converts 20news datasets to a SentenceLabelDataset usable to train the model
        SentenceTransformer.smart_batching_collate as the collate_fn for the DataLoa

        Assumes only one sentence per InputExample and labels as integers from 0 to
        and should be used in combination with dataset_reader.LabelSentenceReader.

        Labels with only one example are ignored.

        smart_batching_collate as collate_fn is required because it transforms the t

        """
        self.model = model
        self.groups_right_border = []
        self.grouped_inputs = []
        self.grouped_labels = []
        self.num_labels = 0
        self.max_processes = min(max_processes, cpu_count())
        self.chunk_size = chunk_size
        self.parallel_tokenization = parallel_tokenization

        if self.parallel_tokenization:
            if multiprocessing.get_start_method() != 'fork':
                logging.info("Parallel tokenization is only available on Unix system
                self.parallel_tokenization = False
```

```
        self.dataset = self.get_dataset()
        self.convert_input_examples(self.dataset[0], model)

        self.idxs = np.arange(len(self.grouped_inputs))

        self.provide_positive = provide_positive
        self.provide_negative = provide_negative

    def get_dataset(self, trainset: str="train", testset: str="test", validation_rat
        """
        Convert 20news dataset in Train, dev, and Test set

        Each instance of train_set is an InputExample with all the class attributes
        """
        ret = []
        for name in [trainset, testset]:
            file = fetch_20newsgroups(subset=name, remove=('headers', 'footers','quote

            examples = []
            guid=1
            for text, target in zip(file.data, file.target):
                guid += 1
                examples.append(InputExample(guid=guid, texts=[text], label=target))
            ret.append(examples)

        train_set, test_set = ret
        dev_set = None

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

        return train_set, dev_set, test_set

    def convert_input_examples(self, examples: List[InputExample], model: SentenceTr
        """
        Converts input examples to a SentenceLabelDataset.

        Assumes only one sentence per InputExample and labels as integers from 0 to
        and should be used in combination with dataset_reader.LabelSentenceReader.

        Labels with only one example are ignored.

        :param examples:
            the input examples for the training
        :param model
            the Sentence Transformer model for the conversion
        :param is_pretokenized
            If set to true, no tokenization will be applied. It is expected that the
        """

        inputs = []
        labels = []

        label_sent_mapping = {}
        too_long = 0
        label_type = None

        logging.info("Start tokenization")
```

```
        if not self.parallel_tokenization or self.max_processes == 1 or len(examples
            tokenized_texts = [self.tokenize_example(example) for example in example
        else:
            logging.info("Use multi-process tokenization with {} processes".format(s
            self.model.to('cpu')
            with Pool(self.max_processes) as p:
                tokenized_texts = list(p.imap(self.tokenize_example, examples, chunk

        # Group examples and labels
        # Add examples with the same label to the same dict
        for ex_index, example in enumerate(tqdm(examples, desc="Convert dataset")):
            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 hasattr(model, 'max_seq_length') and model.max_seq_length is not None
                too_long += 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)

        # Group sentences, such that sentences with the same label
        # are besides each other. Only take labels with at least 2 examples
        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[la
                self.grouped_labels.extend([labels[j] for j in label_sent_mapping[la
                self.groups_right_border.append(len(self.grouped_inputs)) #At which
                self.num_labels += 1

        self.grouped_labels = torch.tensor(self.grouped_labels, dtype=label_type)
        logging.info("Num sentences: %d" % (len(self.grouped_inputs)))
        logging.info("Sentences longer than max_seqence_length: {}".format(too_long)
        logging.info("Number of labels with >1 examples: {}".format(len(distinct_lab


    def tokenize_example(self, example):
        if example.texts_tokenized is not None:
            return example.texts_tokenized

        return [self.model.tokenize(text) for text in example.texts]

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

        # Anchor element
        anchor = self.grouped_inputs[item]

        # Check start and end position for this label in our list of grouped sentenc
        group_idx = bisect.bisect_right(self.groups_right_border, item)
```

```
        left_border = 0 if group_idx == 0 else self.groups_right_border[group_idx -
        right_border = self.groups_right_border[group_idx]

        if self.provide_positive:
            positive_item_idx = np.random.choice(np.concatenate([self.idxs[left_bord
            positive = self.grouped_inputs[positive_item_idx]
        else:
            positive = []

        if self.provide_negative:
            negative_item_idx = np.random.choice(np.concatenate([self.idxs[0:left_bo
            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)
```

We then create a function that generate a triplet from set of input_examples for model evaluation

Entrée [11]:

```
import random
from collections import defaultdict

def triplets_from_labeled_dataset(input_examples):
    # Creates triplets for a [(label, sentence), (label, sentence)...] dataset
    # by using each example as anchor and selecting randomly a
    # positive instance with the same label and a negative instance with different
    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], nega

    return triplets
```

Let's fine tuning a distilbert-base-nli-mean-tokens model with our custom loading class using the TripletLoss loss.

Entrée [12]:

```
from sentence_transformers import LoggingHandler, losses
from sentence_transformers.evaluation import TripletEvaluator
from torch.utils.data import DataLoader
from datetime import datetime

import logging
import os
import urllib.request

logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO,
                    handlers=[LoggingHandler()])

# Continue training distilbert-base-nli-mean-tokens on 20news_groups data
model_name = 'distilbert-base-nli-mean-tokens'

### Create a torch.DataLoader that passes training batch to our model
train_batch_size = 16

if not os.path.exists('/content/drive/My Drive/Ubisoft_takehome_challenge_MLE/Output
    os.makedirs('/content/drive/My Drive/Ubisoft_takehome_challenge_MLE/Output')

output_path = ("/content/drive/My Drive/Ubisoft_takehome_challenge_MLE/Output/fine-T
num_epochs = 2

# Load pretrained model
model = SentenceTransformer(model_name)

logging.info("Read 20 News groups datasets")
train_dataset = Fetch20newsLabelDataset(model=model,
                                        provide_positive=True,   # True for triplet
                                        provide_negative=True)
train_dataloader = DataLoader(train_dataset, shuffle=True, batch_size=train_batch_si
train_loss = losses.TripletLoss(model)
```

```
100%|██████████| 245M/245M [00:15<00:00, 15.3MB/s]
Downloading 20news dataset. This may take a few minutes.
INFO:sklearn.datasets._twenty_newsgroups:Downloading 20news dataset. T
his may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/597596
7 (https://ndownloader.figshare.com/files/5975967) (14 MB)
INFO:sklearn.datasets._twenty_newsgroups:Downloading dataset from http
s://ndownloader.figshare.com/files/5975967 (https://ndownloader.figsha
re.com/files/5975967) (14 MB)
Convert dataset: 100%|██████████| 11201/11201 [00:00<00:00, 310210.17i
t/s]
```

Entrée [13]:

```
### Evaluating model performance before model fine tuning
logging.info("Read 20 News dev set")
dev_set = train_dataset.dataset[1]
dev_evaluator = TripletEvaluator.from_input_examples(triplets_from_labeled_dataset(d

logging.info("Performance before fine-tuning:")
dev_evaluator(model)
```

Out[13]:

```
0.6517857142857143
```

Entrée [ ]:

```
### Model Fune tuning
warmup_steps = int(len(train_dataset) * num_epochs / train_batch_size * 0.1)  # 10%

model.fit(
    train_objectives=[(train_dataloader, train_loss)],
    evaluator=dev_evaluator,
    epochs=num_epochs,
    evaluation_steps=1000,
    warmup_steps=warmup_steps,
    output_path=output_path,
)
```

Entrée [16]:

```
### Evaluate model performance on test set
logging.info("Read 20 News test set")
test_set = train_dataset.dataset[2]
test_evaluator = TripletEvaluator.from_input_examples(triplets_from_labeled_dataset(

logging.info("Evaluating model on test set (after fine tune)")
output_path = "/content/drive/My Drive/Ubisoft_takehome_challenge_MLE/Output/fine-Tr
model = SentenceTransformer(output_path)
test_evaluator(model)
```

Out[16]:

```
0.8698884758364313
```