Part 1: Data Collection

In this assignment, you will first identify or annotate a text dataset, with at least **120 labeled sentences** (more is better), for a text classification task of your choice. Any text data is fine: e.g., news articles, reviews, legal docs, medical docs. The dataset should allow you to create a **binary text classification task (two labels only)**. Feel free to look at Hugging Face datasets or Kaggle, or other places.

You are welcome to annotate your own dataset, just make sure it is not a task that is trivally solvable.

CSE 256: Statistical NLP UCSD Assignment 2

Text Classification with Pretrained Language Models (40 points)

Due: Friday May 12, 2023 at 10pm

IMPORTANT: After copying this notebook to your Google Drive, paste a link to it below. To get a publicly-accessible link, click the *Share* button at the top right, then click "Get shareable link" and copy the link.

Link: paste your link here:

https://colab.research.google.com/drive/1W-zu-a6mPJbi3H9MPly8KgO5Awoh8P8i?usp=sharing

Notes:

Make sure to save the notebook as you go along.

Submission instructions are located at the bottom of the notebook.

Question 1.1 (10 points):

Give a brief summary of the dataset you picked. Biefly describe the text classification task, and why it is a non-trivial task. Provide basic statistics of the dataset (include: how many labeled examples, how many unique words)

Write your answer here (< 6 sentences)

I picked a dataset from hugging face called "poem_sentiment", it is a dataset with poem verses and their sentiment labels, 0 as a negative sentiment label and 1 as a positive sentiment label. There are mix/neutral sentiment labels which I stripped for binary text classification. After processing, there are in total 288 data points, which includes 155 negative data labels and 133 positive data labels. Base on the number this is a relatively balanced dataset. After stripping puncturation marks, there are 1010 unique words in these verses.

Part 2: Text classification

In this part, you will fine-tune pretrained language models on your dataset. This part is meant to be an introduction to the HuggingFace library, and it contains code that will potentially be useful for your final projecs. Since we're dealing with large models, the first step is to change to a GPU runtime.

Adding a hardware accelerator

Go to the menu and add a GPU as follows:

```
Edit > Notebook Settings > Hardware accelerator > (GPU)
```

Run the following cell to confirm that the GPU is detected.

```
import torch

# Confirm that the GPU is detected

assert torch.cuda.is_available()

# Get the GPU device name.
device_name = torch.cuda.get_device_name()
n_gpu = torch.cuda.device_count()
print(f"Found device: {device_name}, n_gpu: {n_gpu}")
device = torch.device("cuda")
```

```
Found device: Tesla T4, n_gpu: 1
```

Installing Hugging Face's Transformers library

We will use Hugging Face's Transformers (https://github.com/huggingface/transformers), an open-source library that provides general-purpose architectures for natural language understanding and generation with a collection of various pretrained models made by the NLP community. This library will allow us to easily use pretrained models like `BERT` and perform experiments on top of them. We

can use these models to solve downstream target tasks, such as text classification, question answering, and sequence labeling.

Run the following cell to install Hugging Face's Transformers library and download a sample data file called tweets.csv that contains tweets about airlines along with a negative, neutral, or positive sentiment rating. Note that you will be asked to link with your Google Drive account to download some of these files. If you're concerned about security risks (there have not been any issues in previous semesters), feel free to make a new Google account and use it for this assignment! alternatively, you can manually download the files from our Google drive, and read them directly in your notebook instead of using the PyDrive API.

```
!pip install datasets
from datasets import load_dataset
dataset = load_dataset('poem_sentiment', split='train')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: dill<0.3.7,>=0.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: multiprocess in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/python3
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: huggingface-hub<1.0.0,>=0.11.0 in /usr/local/lib/
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: responses<0.19 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/py
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/pyt
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pyth
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packag
WARNING:datasets.builder:Found cached dataset poem_sentiment (/root/.cache/huggi
```

```
import string
import collections
```

```
import random
binary_dataset = []
word_dic = collections.defaultdict(int)
negative_label_count = 0
positive_label_count = 0

for data in dataset:
  label = data['label']
  if label == 0 or label == 1:
    if label == 0:
      negative_label_count += 1
    else:
      positive_label_count += 1

    verse = data['verse_text']
    verse = verse.translate(str.maketrans('', '', string.punctuation))
    # maybe use verse without puncturation ?

    for word in verse.split(' '):
      word_dic[word] += 1
    binary_dataset.append(data)

random.Random(0).shuffle(binary_dataset)
print(len(word_dic)) #how many unique word
print(negative_label_count) # how many negative label
print(positive_label_count) # how many positive label
```

```
1010
155
133
```

Double-click (or enter) to edit

```
!pip install transformers
!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
print('success!')

import os
import zipfile
```

```
# Download helper functions file
helper_file = drive.CreateFile({'id': '1XHV97dCHMmsekJyXRduB9Q0sCWLylwrh'})
helper_file.GetContentFile('helpers.py')
print('helper file downloaded! (helpers.py)')

# Download sample file of tweets
data_file = drive.CreateFile({'id': '1pJephA7sBxMbshTtzLAhtzjQrwkfJSzu'})
data_file.GetContentFile('tweets.csv')
print('sample tweets downloaded! (tweets.csv)')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: huggingface-hub<1.0,>=0.11.0 in /usr/local/lib/py
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/di
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/l
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pyth
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/d
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pa
success!
helper file downloaded! (helpers.py)
sample tweets downloaded! (tweets.csv)
```

The cell below imports some helper functions to demonstrate the task on the sample tweet dataset.

```
from helpers import tokenize_and_format, flat_accuracy
```

Data Prep and Model Specification

Upload your data using the file explorer to the left. We have provided a function below to tokenize and format your data as BERT requires. Make sure that your csv file, titled **my_data.csv**, has one column "text" and another column "labels" containing integers. If your dataset comes pre-split, you may choose just the train split for the purpose of this assignment, or you may combine the splits into one single file. We will doing our own data split.

If you run the cell below without modifications, it will run on the tweets.csv example data we have provided. It imports some helper functions to demonstrate the task on the sample tweet dataset. You should first run all of the following cells with tweets.csv just to see how everything works. Then,

once you understand the whole preprocessing / fine-tuning process, change the csv in the below cell to your **my_data.csv** file, add any extra preprocessing code you wish, and then run the cells again on your own data.

```
from helpers import tokenize_and_format, flat_accuracy
import pandas as pd


df = pd.DataFrame.from_records(binary_dataset)
#df = pd.read_csv('tweets.csv')

df = df.sample(frac=1).reset_index(drop=True)

texts = df.verse_text.values
labels = df.label.values

### tokenize_and_format() is a helper function provided in helpers.py ###
input_ids, attention_masks = tokenize_and_format(texts)

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Print sentence 0, now as a list of IDs.
print('Original: ', texts[0])
print('Token IDs:', input_ids[0])
```

```
Original:  mine eyes were of their madness half beguiled,
Token IDs: tensor([  101,  3067,  2159,  2020,  1997,  2037, 12013,  2431, 11693
         2098,  1010,   102,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0])
```

Create train/test/validation splits

Here we split your dataset into 3 parts: a training set, a validation set, and a testing set. Each item in your dataset will be a 3-tuple containing an input_id tensor, an attention_mask tensor, and a label tensor.

```
total = len(df)

num_train = int(total * .7)
```

```
num_val = int(total * .1)
num_test = total - num_train - num_val

# make lists of 3-tuples (already shuffled the dataframe in cell above)

train_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_train)]
val_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_train, nur
test_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_val + nur

train_text = [texts[i] for i in range(num_train)]
val_text = [texts[i] for i in range(num_train, num_val+num_train)]
test_text = [texts[i] for i in range(num_val + num_train, total)]



import gc
gc.collect()
torch.cuda.empty_cache()
```

Here we choose the model we want to finetune from https://huggingface.co/transformers/pretrained_models.html. Because the task requires us to label sentences, we wil be using BertForSequenceClassification below. You may see a warning that states that `some weights of the model checkpoint at [model name] were not used when initializing. . .` This warning is expected and means that you should fine-tune your pre-trained model before using it on your downstream task. See here for more info.

```
from transformers import BertForSequenceClassification, AdamW, BertConfig

model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = 2, # The number of output labels.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# Tell pytorch to run this model on the GPU.
model.cuda()
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when ini
- This IS expected if you are initializing BertForSequenceClassification from th
- This IS NOT expected if you are initializing BertForSequenceClassification fro
Some weights of BertForSequenceClassification were not initialized from the mode
You should probably TRAIN this model on a down-stream task to be able to use it
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
```

```
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

ACTION REQUIRED

Define your approach to fine-tuning hyperparameters in the cell below (we have randomly picked some values to start with). We want you to experiment with different configurations to find the one that works best (i.e., highest accuracy) on your validation set. Feel free to also change pretrained models to others available in the HuggingFace library (you'll have to modify the cell above to do this). You might find papers on BERT fine-tuning stability (e.g., Mosbach et al., ICLR 2021) to be of interest.

```
batch_size = 99
optimizer = AdamW(model.parameters(),
                  lr = 5e-5, # args.learning_rate - default is 5e-5
                  eps = 1e-7 # args.adam_epsilon  - default is 1e-8
                )
epochs = 10
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:391: FutureW
  warnings.warn(
```

Fine-tune your model

Here we provide code for fine-tuning your model, monitoring the loss, and checking your validation accuracy. Rerun both of the below cells when you change your hyperparameters above.

```
import numpy as np
# function to get validation accuracy
def get_validation_performance(val_set):
    # Put the model in evaluation mode
    model.eval()

    # Tracking variables
    total_eval_accuracy = 0
    total_eval_loss = 0

    num_batches = int(len(val_set)/batch_size) + 1

    total_correct = 0

    for i in range(num_batches):

      end_index = min(batch_size * (i+1), len(val_set))

      batch = val_set[i*batch_size:end_index]

      if len(batch) == 0: continue

      input_id_tensors = torch.stack([data[0] for data in batch])
      input_mask_tensors = torch.stack([data[1] for data in batch])
      label_tensors = torch.stack([data[2] for data in batch])

      # Move tensors to the GPU
      b_input_ids = input_id_tensors.to(device)
      b_input_mask = input_mask_tensors.to(device)
      b_labels = label_tensors.to(device)

      # Tell pytorch not to bother with constructing the compute graph during
      # the forward pass, since this is only needed for backprop (training).
```

```
        with torch.no_grad():

          # Forward pass, calculate logit predictions.
          outputs = model(b_input_ids,
                                  token_type_ids=None,
                                  attention_mask=b_input_mask,
                                  labels=b_labels)
          loss = outputs.loss
          logits = outputs.logits

          # Accumulate the validation loss.
          total_eval_loss += loss.item()

          # Move logits and labels to CPU
          logits = logits.detach().cpu().numpy()
          label_ids = b_labels.to('cpu').numpy()

          # Calculate the number of correctly labeled examples in batch
          pred_flat = np.argmax(logits, axis=1).flatten()
          labels_flat = label_ids.flatten()
          num_correct = np.sum(pred_flat == labels_flat)
          total_correct += num_correct

    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_correct / len(val_set)
    return avg_val_accuracy
```

```
import random

# training loop

# For each epoch...
for epoch_i in range(0, epochs):
    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    # Reset the total loss for this epoch.
    total_train_loss = 0

    # Put the model into training mode.
    model.train()

    # For each batch of training data...
    num_batches = int(len(train_set)/batch_size) + 1
```

```
      for i in range(num_batches):
        end_index = min(batch_size * (i+1), len(train_set))

        batch = train_set[i*batch_size:end_index]

        if len(batch) == 0: continue

        input_id_tensors = torch.stack([data[0] for data in batch])
        input_mask_tensors = torch.stack([data[1] for data in batch])
        label_tensors = torch.stack([data[2] for data in batch])

        # Move tensors to the GPU
        b_input_ids = input_id_tensors.to(device)
        b_input_mask = input_mask_tensors.to(device)
        b_labels = label_tensors.to(device)

        # Clear the previously calculated gradient
        model.zero_grad()

        # Perform a forward pass (evaluate the model on this training batch).
        outputs = model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask,
                                labels=b_labels)
        loss = outputs.loss
        logits = outputs.logits

        total_train_loss += loss.item()

        # Perform a backward pass to calculate the gradients.
        loss.backward()

        # Update parameters and take a step using the computed gradient.
        optimizer.step()

      # ========================================
      #               Validation
      # ========================================
      # After the completion of each training epoch, measure our performance on
      # our validation set. Implement this function in the cell above.
      print(f"Total loss: {total_train_loss}")
      val_acc = get_validation_performance(val_set)
      print(f"Validation accuracy: {val_acc}")

  print("")
  print("Training complete!")
```

```
      ======== Epoch 1 / 10 ========
      Training...
      Total loss: 2.102378010749817
```

```
Validation accuracy: 0.5

======== Epoch 2 / 10 ========
Training...
Total loss: 1.7306684851646423
Validation accuracy: 0.5

======== Epoch 3 / 10 ========
Training...
Total loss: 1.644875094294548
Validation accuracy: 0.5

======== Epoch 4 / 10 ========
Training...
Total loss: 1.4121283888816833
Validation accuracy: 0.6071428571428571

======== Epoch 5 / 10 ========
Training...
Total loss: 1.1197766065597534
Validation accuracy: 0.6785714285714286

======== Epoch 6 / 10 ========
Training...
Total loss: 0.7651330605149269
Validation accuracy: 0.8928571428571429

======== Epoch 7 / 10 ========
Training...
Total loss: 0.3436094969511032
Validation accuracy: 0.9642857142857143

======== Epoch 8 / 10 ========
Training...
Total loss: 0.15294384770095348
Validation accuracy: 0.8214285714285714

======== Epoch 9 / 10 ========
Training...
Total loss: 0.08461884781718254
Validation accuracy: 0.9642857142857143

======== Epoch 10 / 10 ========
Training...
Total loss: 0.056054119020700455
Validation accuracy: 0.9642857142857143

Training complete!
```

Evaluate your model on the test set

After you're satisfied with your hyperparameters (i.e., you're unable to achieve higher validation accuracy by modifying them further), it's time to evaluate your model on the test set! Run the below

cell to compute test set accuracy.

```
get_validation_performance(test_set)
```

```
0.9152542372881356
```

Question 2.1 (10 points):

Congratulations! You've now gone through the entire fine-tuning process and created a model for your downstream task. Describe your hyperparameter selection process in words. If you based your process on any research papers or websites, please reference them. Why do you think the hyperparameters you ended up choosing worked better than others? Also, is there a significant discrepancy between your test and validation accuracy? Why do you think this is the case?

Write your answer here

I used a nested for loop to loop over a couple reasonable value of learning rates(from 5e-1 to 5e-19), and some adam_epsilon values (from 1e-1 to 1e-19). From this process, I found the default learning rate produces the biggest accuracy, and by increasing the epoch to 10 and decreasing the adam_epsilon values to 1e-7. Since the epochs means the one complete iteration over the entire dataset during training, I noticed that after 5 epochs, the loss hasn't stop decreasing, so it is not converged yet to optimal soltion, that is why I increase the epoch to 10. For adam_epsilon value, since the dataset I worked on is pretty small, so I tuned this parameter a bit to make sure the gradient 's fluctuation won't affect the result. The validation result is 0.96 accuracy and the actual test result is 0.91 accuracy, so there is no significant discrepancy between test and validation accuracy. I think this difference here is mostly due to a slight overfitting to a small dataset.

Question 2.2 (20 points):

(Involves both coding, and a written answer) Finally, perform an *error analysis* on your model. This is good practice for your final project. **Write some code** in the below code cell to print out the text five test set examples that your model gets **wrong**. If your model gets more than five test examples wrong, randomly choose five of them to analyze. If your model gets fewer than five examples wrong, please design five test examples that fool your model (i.e., *adversarial examples*). Then, in the following text cell, perform a qualitative analysis of these examples. See if you can figure out any reasons for errors that you observe, or if you have any informed guesses (e.g., common linguistic properties of these particular examples). Does this analysis suggest any possible future steps to improve your classifier?

```
def get_errors(val_set):
    # Put the model in evaluation mode
    model.eval()

    # Tracking variables

    input_id_tensors = torch.stack([data[0] for data in val_set])
    input_mask_tensors = torch.stack([data[1] for data in val_set])
    label_tensors = torch.stack([data[2] for data in val_set])

    # Move tensors to the GPU
    b_input_ids = input_id_tensors.to(device)
    b_input_mask = input_mask_tensors.to(device)
    b_labels = label_tensors.to(device)

    # Tell pytorch not to bother with constructing the compute graph during
    # the forward pass, since this is only needed for backprop (training).
    with torch.no_grad():

      # Forward pass, calculate logit predictions.
      outputs = model(b_input_ids,
                              token_type_ids=None,
                              attention_mask=b_input_mask,
                              labels=b_labels)


      logits = outputs.logits

      # Move logits and labels to CPU
      logits = logits.detach().cpu().numpy()
      label_ids = b_labels.to('cpu').numpy()

      # Calculate the number of correctly labeled examples in batch
      pred_flat = np.argmax(logits, axis=1).flatten()
      labels_flat = label_ids.flatten()


    return [i for i, item in enumerate(zip(pred_flat,labels_flat)) if item[0] != item[

err_lst = get_errors(test_set)
test_set_pre_embedding = binary_dataset[num_val + num_train:]
for ind in err_lst:
  print(test_set_pre_embedding[ind])
```

```
{'id': 493, 'verse_text': 'then thro’ his breast his fatal sword he sent,', 'lab
{'id': 569, 'verse_text': 'no word for a while spake regin; but he hung his head
{'id': 718, 'verse_text': 'make a fragrance of her fame.', 'label': 1}
{'id': 513, 'verse_text': 'in our embraces we again enfold her,', 'label': 1}
{'id': 234, 'verse_text': 'the visual nerve is withered to the root,', 'label': 
```

Provide a qualitative analysis of the above examples here.

From the five errors above, it seems the above verses have sentiments expressed with verb + noun instead of adjective. And the text are pretty abstract with these un-common word. I think the mistake is due to this small training dataset, it is hard to predict these vocabulary, especially short phrases meaning without seeing recurring example. So increase training dataset size will be a way to improve this. Another point I noticed is that a lot of verb with sentiments are with past tense. Maybe while processing training data, we can do some pre-processing to handle tense so a same verb in a training set can be learned more effectively.

Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Select Edit -> Clear All Outputs. This will clear all the outputs from all cells (but will keep the content of all cells).
3. Select Runtime -> Run All. This will run all the cells in order, and will take several minutes.
4. Once you've rerun everything, select File -> Download as -> PDF via LaTeX (If you have trouble using "PDF via LaTex", you can also save the webpage as pdf. Make sure all your solutions especially the coding parts are displayed in the pdf, it's okay if the provided codes get cut off because lines are not wrapped in code cells).
5. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing your graders will see!
6. Submit your PDF on Gradescope.

Acknowledgements

This assignment is based on an assignment developed by Mohit Iyyer

✓ 0s completed at 5:45 PM