Homework 1, CS685 Spring 2023

This is due on April 5th, 2023. This notebook is to be submitted via Gradescope as a PDF file, while your three dataset files (annotator1.csv, annotator2.csv, and final_data.csv) should be emailed to cs685instructors@gmail.com with the subject line formatted as **Firstname_Lastname_HW1data**. 100 points total.

IMPORTANT: After copying this notebook to your Google Drive, please paste a link to it below. To get a publicly-accessible link, hit the *Share* button at the top right, then click "Get shareable link" and copy over the result. If you fail to do this, you will receive no credit for this homework!

LINK: https://colab.research.google.com/drive/1K-mD4XpcA5qrAeCLAXiVAn1mLCy_z6Si?usp=sharing

---

How to submit this problem set:

- Write all the answers in this Colab notebook. Once you are finished, generate a PDF via (File -> Print -> Save as PDF) and upload it to Gradescope.
- **Important:** check your PDF before you submit to Gradescope to make sure it exported correctly. If Colab gets confused about your syntax, it will sometimes terminate the PDF creation routine early.
- **Important:** on Gradescope, please make sure that you tag each page with the corresponding question(s). This makes it significantly easier for our graders to grade submissions, especially with the long outputs of many of these cells. We will take off points for submissions that are not tagged.
- When creating your final version of the PDF to hand in, please do a fresh restart and execute every cell in order. One handy way to do this is by clicking `Runtime -> Run All` in the notebook menu.

Academic honesty

- We will audit the Colab notebooks from a set number of students, chosen at random. The audits will check that the code you wrote actually generates the answers in your PDF. If you turn in correct answers on your PDF without code that actually generates those answers, we will consider this a serious case of cheating. See the course page for honesty policies.
- We will also run automatic checks of Colab notebooks for plagiarism. Copying code from others is also considered a serious case of cheating.
- There is no penalty for using AI assistance on this homework as long as you fully disclose it in the final cell of this notebook (this includes storing any prompts that you feed to large language models). That said, anyone caught using AI assistance without proper disclosure will receive a zero on the assignment (we have several automatic tools to detect such cases). We're literally allowing you to use it with no limitations, so there is no reason to lie!

Part 1: Annotation

In this homework, you will first collect a labeled dataset of **120** sentences for a text classification task of your choice. This process will include:

1. *Data collection*: Collect 120 sentences from any source you find interesting (e.g., literature, Tweets, news articles, reviews, etc.)

2. *Task design*: Come up with a binary (i.e., only two labels) sentence-level classification task that you would like to perform on your sentences. Be creative, and make sure your task isn't too easy (e.g., perhaps the labels have some degree of subjectivity to them, or the task is otherwise complex for humans). Write up annotator guidelines/instructions on how you would like people to label your data.
3. On your dataset, collect annotations from **two** classmates for your task. Everyone in this class will need to both create their own dataset and also serve as an annotator for two other classmates. In order to get everything done on time, you need to complete the following steps:

> - Find two classmates willing to label 120 sentences each (use the Piazza "search for teammates" thread if you're having issues finding labelers).
> - Send them your annotation guidelines and a way that they can easily annotate the data (e.g., a spreadsheet or Google form)
> - Collect the labeled data from each of the two annotators.
> - Sanity check the data for basic cleanliness (are all examples annotated? are all labels allowable ones?)

4. Collect feedback from annotators about the task including annotation time and obstacles encountered (e.g., maybe your guidelines were confusing! or maybe some sentences were particularly hard to annotate!)
5. Calculate and report inter-annotator agreement.
6. Aggregate output from both annotators to create final dataset.
7. Perform NLP experiments on your new dataset!

Question 1.1 (10 points):

Describe the source of your unlabeled data, why you chose it, and what kind of sentence selection process you used (if any) to choose 120 sentences for annotation. Also briefly describe the text classification task that you will be collecting labels for in the next section.

Description: of Dataset: Toxicity Classification Dataset

Source of Unlabeled Data: One hundred twenty user comments were hand-curated with great cautiousness and extensiveness from online platforms such as YouTube, Twitter, Reddit, Instagram, and Sports, Fashion, and Beauty websites.

Reason for choosing: Toxicity classification datasets are vital because they can not only help us understand and mitigate the impact of toxic behavior online but also help in understanding whether a user comment is really toxic or just an honest, unfiltered opinion that is harmless and not targeted to anyone in the user community(Sometimes it is falsely classified as toxic).

With the rise of social media, online harassment and toxic behavior have become increasingly prevalent, and it is crucial to develop effective tools to combat these issues.

A toxicity classification dataset is a collection of labeled text data classified as toxic or non-toxic. This dataset can be used to train machine learning models to detect and flag toxic content in real-time automatically. By doing so, social media platforms and other online communities can take appropriate action to remove or moderate toxic content and protect their users from harm.

Moreover, toxicity classification datasets can also be used for research purposes, such as analyzing trends in toxic behavior, identifying the factors that contribute to it, and clearly distinguishing between actual toxic user comments and an honest, unfiltered opinion that is harmless and not targeted to anyone in the user community. This can help us develop more effective interventions and strategies to prevent and reduce toxic behavior online.

Overall, toxicity classification datasets are essential tools for understanding and mitigating the impact of toxic behavior online, and they have the potential to make online communities safer and more inclusive for everyone.

Sentence Selection Process: The sentence selection process was hand-curated from various online platforms such as YouTube, Twitter, Reddit, Instagram, and Sports, Fashion, and Beauty websites.

For relevant sentences, I decided to choose ambigious user comments (which can be toxic for some and non-toxic for others) as well as positive and non toxic user comments in proportions of 30%, 35% and 35% respectively. Parameters such as tone of the sentence, slang language used in the context and individual perception as a receiver were taken into consideration. The proportions of such user comments was considered to give a balanced dataset and create a generalised model to mimick real life scenario

Text Classification Task: There is a fine line between non-toxic passive-aggressive user comments and an honest, unfiltered opinion that is harmless and not targeted to anyone in the user community. Confidant/genuine/uncensored/vulnerable/real/non-superficial/not out of social courtesy or obligations opinions should be encouraged. Being opinionated and not wrong hurting sentiments is like freedom from social taboo or stigma and can break generational/traditional thoughts conditioned from eons and design a free, open, and more accepting world. With this in mind, the Text Classification Task is to label the user comments as “toxic” or “non-toxic”, depending on how you feel and perceive the comment.

Question 1.2 (25 points):

Copy the annotation guidelines that you provided to your classmates below. We expect that these guidelines will be very detailed (and as such could be fairly long). You must include:

> - The two categories for your binary classification problem, including the exact strings you want annotators to use while labeling.
> - Descriptions of the categories and what they mean.
> - Representative examples of each category (i.e., sentences from outside your dataset that you have manually labeled to give annotators an idea of how to perform the task)
> - A discussion of of tricky corner cases, and criteria to help the annotator decide them. If you look at the data and think about how an annotator could do the task, you will likely find a bunch of these!

ANNOTATION GUIDELINES

Thank you for agreeing and taking time to annotate the dataset. Please click on the link to access the **Toxicity Classification Dataset** (https://docs.google.com/spreadsheets/d/1RFSl3VF4JrTTTHYGxUaHbw_WXGEV4D75QnM9zjKZG60/edit?).

Problem Statement: The shared dataset carefully curates 120 user comments from online platforms such as YouTube, Twitter, Reddit, Instagram, Sports, Fashion, and Beauty websites. Your task is to label the user comments as “toxic” or “non-toxic”, depending on how you feel and perceive the comment.

Kindly follow the below guidelines to take care while annotating:

a) Label the user comment as “toxic” or “non-toxic” in lowercase.

b) **Non-Toxic Cases:** Assign “non-toxic” label to user comments which typically encourages, inspires, validates, or provides valuable feedback to social media posts / videos / statuses / tweets/ blogs in a positive respectful manner. Examples: i. Great video, really enjoyed it! Keep up the

good work. ii. Just tried a new recipe and it turned out delicious! Cooking is so much fun. iii. Congratulations on your achievement! You worked hard for this and it paid off. iv. Congratulations to the winning team. They deserved the victory and played with great sportsmanship.

c) **Toxic Cases:** Assign “toxic” label to user comment which promotes hate speech, harassment, profanity and vulgarity on social media posts / videos / statuses / tweets / blogs in a negative disrespectful manner. Examples: i. You're just being dramatic. It's not that bad. Get over it. ii. Your music is trash. I can't believe people actually listen to this. iii. Your channel is trash, I can't believe you have any subscribers. iv. This is such a pointless post. Why are you even wasting our time with this?

d) **Tricky Corner Cases:** There may be few subjective users comments whose classification will depend on one’s perspective. You may follow the 2-Step Process described below to decide:

1. When in doubt, you may ask yourself whether the user comment is an honest / unfiltered / harmless talk and not an offensive remark? If yes label it as ‘non-toxic’.
2. If you still can’t decide, trust your instincts! (Think about the tone of the sentence, slang language used in the context, how you will perceive it as a receiver, whether it is passive aggressive aka ‘toxic’ or harmless aka ‘non-toxic’ etc.) Some ‘non-toxic’ examples where the users are voicing their unfiltered opinion and yet not being disrespectful to the online platform’s community are: i. Some people are inconsiderate. They don't care how their actions affect others. ii. People can be non-judgmental around things which they don't like. iii. This video is a joke! You clearly have an idea on what you're doing. iv. This is rigged. The referees clearly favored the other team.

e) Additionally, please ignore any grammatical/syntactical errors.

Feel free to share your experience about the task in this feedback form (https://forms.gle/gkicGYLTv3SYceS89). Hope you enjoy the task!

Question 1.3 (5 points):

Write down the names and emails of the two classmates who will be annotating your data below. Once they are finished annotating, create two .csv files (annotator1.csv and annotator2.csv) that contains each annotator's labels. The file should have two columns with headers **text** and **label**, respectively. You will include these files in an email to the instructors account when you're finished with this homework.

The tweets.csv file provided as an example in Part 2 below uses the same format.

WRITE CLASSMATE 1 NAME/EMAIL HERE:

Nandhinee Periyakaruppan nperiyakarup@umass.edu

WRITE CLASSMATE 2 NAME/EMAIL HERE:

Shebin Scaria sscaria@umass.edu

Question 1.4 (10 points):

After both annotators have finished labeling the 120 sentences you gave them, ask them for feedback about your task and the provided annotation guidelines. If you were to collect more labeled data for this task in the future, what would you change from your current setup? Why? Please include a summary of annotator feedback (with specific examples that they found challenging to label) in your answer.

Feedback Summary From Nandhinee Periyakaruppan:

Did you find the task easy/difficult to annotate ? Bit Easy(Level 2)

Was the dataset easy/difficult to understand? Bit Easy(Level 2)

Were the annotation guidelines easy to comprehend and useful in performing the task efficiently? Yes

How much time did it take for you to complete the annotation ? 35 min

Please list down few user comments which you may find tricky / ambiguous / hard to annotate. Also state how you approached the trickiness to classify the label.

This is an interesting task. I enjoyed working on this task. There are two things which I found tricky. If I had some details about the platform or the content for which comment was posted, it would have been helpful. There are a few sentences that were tricky to annotate like "I dont understand why people try to control everything! Live and let live.". This opinion makes sense but if it was posted for something just to show hatred, then this can be toxic. Else this statement is non-toxic.

Feedback Summary From Shebin Scaria:

Did you find the task easy/difficult to annotate ? Easy(Level 1)

Was the dataset easy/difficult to understand? Easy(Level 1)

Were the annotation guidelines easy to comprehend and useful in performing the task efficiently? Yes

How much time did it take for you to complete the annotation ? 30 min

Please list down few user comments which you may find tricky / ambiguous / hard to annotate. Also state how you approached the trickiness to classify the label. I dont understand why people try to control everything! Live and let live; I wouldn't even feed that to my dog. Looks disgusting; are amongst few examples which were tricky to classify. These could be interpreted as simply stated as neutral opinions which may/may not be toxic.

If you were to collect more labeled data for this task in the future, what would you change from your current setup? Why?

1. More research could be done to understand nontoxic-centric and toxic-centric sentence formations in the online activity.
2. Instead of providing just 2 different classes to label, maybe we could provide a set of attributes on which annotators could rate user comments on a scale (like 1-5, 1 for least non-toxic to 5 for most toxic).

Reason: These changes would be done to develop a more exhaustive generalised model.

Question 1.5 (10 points):

Now, compute the inter-annotator agreement between your two annotators. Upload both .csv files to your Colab session (click the folder icon in the sidebar to the left of the screen). In the code cell below, read the data from the two files and compute both the raw agreement (% of examples for which both annotators agreed on the label) and the Cohen's Kappa. Feel free to use implementations in existing libraries (e.g., sklearn). After you're done, paste the numbers in the text cell that follows your code.

If you're curious, Cohen suggested the Kappa result be interpreted as follows: values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41– 0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement.

```
### WRITE CODE TO LOAD ANNOTATIONS AND
### COMPUTE AGREEMENT + COHEN'S KAPPA HERE!
import pandas as pd
import numpy as np
```

```
from sklearn.metrics import cohen_kappa_score

#Setting Seed to Reproduce results
import random
np.random.seed(0)
random.seed(0)

# Load annotations
annotatordata1 = pd.read_csv("annotator1.csv")
annotatordata2 = pd.read_csv("annotator2.csv")
annotatorlabel1 = annotatordata1['label'].tolist()
annotatorlabel2 = annotatordata2['label'].tolist()

annotatordata1['label'] = annotatordata1['label'].to_numpy()
annotatordata2['label'] = annotatordata2['label'].to_numpy()

# Calculate Raw Agreement
matches = 0
for i in range(len(annotatorlabel1)):
  if annotatorlabel1[i] == annotatorlabel2[i]:
    matches+=1
rawAgreement = matches * 100/len(annotatorlabel1)
print("Raw Agreement: ", rawAgreement)

# Calculate Cohen's Kappa
cohenkappaScore = cohen_kappa_score(annotatordata1['label'], annotatordata2['label'])
print("Cohen Kappa Score: ", cohenkappaScore)
```

```
     Raw Agreement:  88.33333333333333
     Cohen Kappa Score:  0.7619047619047619
```

RAW AGREEMENT: 88.33333333333333

COHEN'S KAPPA: 0.7619047619047619

Question 1.6 (10 points):

To form your final dataset, you need to *aggregate* the annotations from both annotators (i.e., for cases where they disagree, you need to choose a single label). Use any method you like other than random label selection to perform this aggregation (e.g., have the two annotators discuss each disagreement and come to consensus, or choose the label you agree with the most). Upload your final dataset to the Colab session (in the same format as the other two files) as final_dataset.csv. Remember to include this file in your final email to us!

DESCRIBE YOUR AGGREGATION STRATEGY HERE

I used a third annotater's data for tie-breaks. This dataset was annotated by Vijayalakshmi Vasudevan (vijayalakshm@umass.edu) . The strategy used for tie-breaking is as follows:

- **If Annotator 1 and 2 are in agreement:**

```
Keep the annotated label/classification
```

- **Else:**

```
Use the annotated label/classification from the tie-breaker dataset
```

```
#AGGREGATION STRATEGY IMPLEMENTATION
annotatordata3 = pd.read_csv("annotator3.csv")
annotatorlabel3 = annotatordata3['label'].tolist()
annotatordata3['label'] = annotatordata3['label'].to_numpy()

final_data = annotatordata1
final_datalabel = final_data['label'].tolist()
final_data['label'] = final_data['label'].to_numpy()

for i in range(len(annotatorlabel1)):
  if annotatorlabel1[i] == annotatorlabel2[i]:
    final_datalabel[i] = annotatorlabel1[i]
  else:
    final_datalabel[i] = annotatorlabel3[i]

final_data['label'] =  final_datalabel
final_data.to_csv('final_data.csv', index=False)
```

Part 2: Text classification

Now we'll move onto fine-tuning pretrained language models specifically on your dataset. This part of the homework is meant to be an introduction to the HuggingFace library, and it contains code that will potentially be useful for your final projects. Since we're dealing with large models, the first step is to change to a GPU runtime.

Adding a hardware accelerator

Please 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

#Setting Seed to Reproduce results
torch.manual_seed(0)
torch.cuda.manual_seed(0)

# 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 homework!

```
!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': '16HW-z9Y1tM3gZ_vFpJAuwUDohz91Aac-'})
helper_file.GetContentFile('helpers.py')
print('helper file downloaded! (helpers.py)')

# Download sample file of tweets
data_file = drive.CreateFile({'id': '1QcoAmjOYRtsMX7njjQTYooIbJHPc6Ese'})
data_file.GetContentFile('tweets.csv')
print('sample tweets downloaded! (tweets.csv)')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting transformers
  Downloading transformers-4.27.4-py3-none-any.whl (6.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.8/6.8 MB 76.7 MB/s eta 0:00:00
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from transformers) (23.0)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from transformers) (2.27.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.9/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from transformers) (6.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from transformers) (3.10.7)
```

```
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.3-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 71.5 MB/s eta 0:00:00
Collecting huggingface-hub<1.0,>=0.11.0
  Downloading huggingface_hub-0.13.4-py3-none-any.whl (200 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.1/200.1 KB 27.4 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.9/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-packages (from huggingface-hub<1.0,>=0.11.0->transformers) (4.5.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (1.26.15)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->transformers) (3.4)
Installing collected packages: tokenizers, huggingface-hub, transformers
Successfully installed huggingface-hub-0.13.4 tokenizers-0.13.3 transformers-4.27.4
success!
helper file downloaded! (helpers.py)
sample tweets downloaded! (tweets.csv)
```

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

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

Part 1: Data Prep and Model Specifications

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 final_data.csv, has one column "text" and another column "labels" containing integers.

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 we wrote 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 final_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

df = pd.read_csv('final_data.csv')
#df = pd.read_csv('tweets.csv')

#Extra preprocessing: Conversting the labels format to binary numbers
df['label'].replace(['non-toxic', 'toxic'], [0, 1], inplace=True)

df2 = df.copy()
df = df.sample(frac=1).reset_index(drop=True)

texts = df.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])
```

```
Downloading (…)solve/main/vocab.txt: 100%  232k/232k [00:00<00:00, 1.96MB/s]
Downloading (…)okenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 962B/s]
Downloading (…)lve/main/config.json: 100%  570/570 [00:00<00:00, 11.3kB/s]
Original:   This dish looks delicious! Can you share the recipe with us?
Token IDs: tensor([  101,  2023,  9841,  3504, 12090,   999,  2064,  2017,  3745,  1996,
        17974,  2007,  2149,  1029,   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])
```

```
#Non-Matching Indices
nonMatchesID = []
for i in range(len(annotatorlabel1)):
  if annotatorlabel1[i] != annotatorlabel2[i]:
    nonMatchesID.append(i)
print("ID:", nonMatchesID)

#Non-Matching Labels
nonMatchesLabel = []
for i in range(len(annotatorlabel1)):
  if annotatorlabel1[i] != final_datalabel[i]:
    nonMatchesLabel.append(final_datalabel[i])
print("Label", nonMatchesLabel)

#Ambigious Sentences
for i in range(len(nonMatchesID)):
  idx = nonMatchesID[i]
  print(df2.text.values[idx])
```

```
ID: [3, 29, 31, 53, 75, 77, 79, 87, 89, 93, 95, 105, 111, 119]
Label ['toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic', 'toxic']
 You're so stupid, I can't believe you actually made this video.
 Some people are so fake. They act one way in front of you and another way behind your back.
 I dont understand why people try to control everything! Live and let live.
 That outfit is so outdated. You need to get with the times.
 You're so stupid for not knowing this already. It's common knowledge.
 You're such an idiot. How could you not know this? It's basic knowledge.
 I can't believe how naive you are. You clearly have no idea what you're talking about.
 This team is going nowhere. They'll be lucky if they even make it to the playoffs.
 This athlete is overrated. They're not as good as everyone makes them out to be.
 These fans are obnoxious. They need to calm down and learn some respect.
 This coach is clueless. They have no idea what they're doing and it shows on the court/field.
 This makeup tutorial was a waste of time. You don't know what you're doing.
 This makeup is boring. You need to add more color and sparkle to your look.
 This beauty product is a scam. Don't waste your money on it.
```

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 * .8)
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, num_val+num_train)]
test_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_val + num_train, total)]

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)]
```

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

###############################################################################################
#                                        Grid Search                                          #
###############################################################################################

def get_validation_performanceGS(modelGS, val_set, batch_size):
    # Put the model in evaluation mode
    modelGS.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 = modelGS(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

batch_sizeGSList = [8, 16, 32, 64]
lrGSList = [3e-5, 5e-5, 7e-5]
epsGSList = [1e-3, 1e-5, 1e-8]
epochsGSList = [5, 10]

best_acc = 0
best_model = None
best_batch_size = 0
best_epochs = 0
best_lr = 0
best_eps = 0

for epochs in epochsGSList:
  for batch_size in batch_sizeGSList:
    for lr in lrGSList:
      for eps in epsGSList:
        modelGS = 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.
    )
optimizerGS = AdamW(modelGS.parameters(), lr = lr, eps = eps)

for epoch_i in range(0, epochs):
  # Tell pytorch to run this model on the GPU.
  modelGS.cuda()

  # 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.
  modelGS.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
    modelGS.zero_grad()

    # Perform a forward pass (evaluate the model on this training batch).
    outputs = modelGS(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.
    optimizerGS.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_performanceGS(modelGS, val_set, batch_size)
            print(f"Validation accuracy: {val_acc}")

            if val_acc > best_acc:
              best_acc = val_acc
              best_model = modelGS
              best_batch_size = batch_size
              best_epochs = epochs
              best_lr = lr
              best_eps = eps

print("Best Params")
print("Batch Size: ", best_batch_size,"Number of Epochs: ", best_epochs,"Learning Rate: ", best_lr, "Epsilon: ",best_eps, "Accuracy: ", best_acc)
```

```
======== Epoch 8 / 10 ========
Training...
Total loss: 0.056147702038288116
Validation accuracy: 0.9166666666666666

======== Epoch 9 / 10 ========
Training...
Total loss: 0.03508703038096428
Validation accuracy: 0.9166666666666666

======== Epoch 10 / 10 ========
Training...
Total loss: 0.019753691740334034
Validation accuracy: 0.9166666666666666
Best Params
```

```
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 initializing BertForSequenceClassification: ['cls.predictions.transform.LayerNorm.weight', 'cls.pre
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSeque
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.weight', 'classifier.b
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
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 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 = best_batch_size
optimizer = AdamW(model.parameters(), lr = best_lr, eps = best_eps)
epochs = best_epochs
```

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.

```
# 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



# 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 / 5 ========
Training...
Total loss: 6.324203014373779
Validation accuracy: 0.9166666666666666

======== Epoch 2 / 5 ========
Training...
Total loss: 2.9033351354300976
Validation accuracy: 0.9166666666666666

======== Epoch 3 / 5 ========
Training...
Total loss: 3.0525616500526667
Validation accuracy: 0.9166666666666666

======== Epoch 4 / 5 ========
Training...
Total loss: 1.7131653502583504
Validation accuracy: 0.9166666666666666

======== Epoch 5 / 5 ========
Training...
Total loss: 1.2410315573215485
Validation accuracy: 0.9166666666666666

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)
```

```
0.9166666666666666
```

Question 2.1 (10 points):

Congratulations! You've now gone through the entire fine-tuning process and created a model for your downstream task. Two more questions left :) First, 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?

Hyperparameter selection process: Handwritten Grid search process was perfomed across the following values: batch_size = [8, 16, 32, 64] lr = [3e-5, 5e-5, 7e-5] eps = [1e-3, 1e-5, 1e-8] epochs = [5, 10]

Reference: https://medium.com/fintechexplained/what-is-grid-search-c01fe886ef0a

Best Params Results: Batch Size: 8

Number of Epochs: 5

Learning Rate: 7e-05

Epsilon: 1e-05

Grid search is a straightforward way of selecting hyperparameters, where we go through a list of parameters, exhaustively checking all combinations to find the best-performing hyperparameters. It is a brute-force approach to hyperparameter tuning.

I can't say why the chosen hyperparameters are better than the others I used during the grid search. Hyperparam tuning is not an exact science, and it could be possible that there are better hyper-params for this model. A better approach for hyper-param tuning would be random search or Bayesian optimization. There is a significant discrepancy between my test and validation accuracies. This is probably because the amount of training data needs to be increased to generalize well, i.e., the model cannot learn much from the limited amount of data provided. Instead, more training data would be needed to create a model with deep enough insight to generalize well on validation and test set.

Question 2.2 (20 points):

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 of up to 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?

```
## YOUR ERROR ANALYSIS CODE HERE
## print out up to 5 test set examples (or adversarial examples) that your model gets wrong

def findLabelsPreds(error_set):
    # Put the model in evaluation mode
    model.eval()

    input_id_tensors = torch.stack([test[0] for test in error_set])
    input_mask_tensors = torch.stack([test[1] for test in error_set])
    label_tensors = torch.stack([test[2] for test in error_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)

    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 labels_flat, pred_flat

labels_flat, pred_flat = findLabelsPreds(test_set)
print("Qualitative Analysis on Test Set \n Correct Labels: ",labels_flat,"\n Predict Labels: ", pred_flat)
for i in range(len(labels_flat)):
  #Find index of non-matching labels and corresponding sentences
  if pred_flat[i] != labels_flat[i]:
    print(f"User Comment {i+1}: {test_text[i]}")
print("Since only one example is wrong , designing 5 new Adversarial examples")

adversarialText  = []
adversarialLabel = []

#If your model gets fewer than five examples wrong, please design five test examples that fool your model
#'non-toxic' 0, 'toxic' 1
sentence1 = "Nobody cares about your vacation. Stop spamming us with your photos."
label1 = 0

sentence2 = "You're all idiots for even talking about this. You don't know what you're talking about."
label2 = 0

sentence3 = "Information easily available. Leave!"
label3 = 1

sentence4 = "I appreciate the thoughtful discussion in this forum. However its naive and dumb."
label4 = 1

sentence5 = "The author of this article is dumb!"
label5 = 1

adversarialText.extend([sentence1, sentence2, sentence3, sentence4, sentence5])
adversarialLabel.extend([label1, label2, label3, label4, label5])

adversarial_input_ids,  adversarial_attention_masks = tokenize_and_format(adversarialText)

# Convert the lists into tensors.
adversarial_input_ids = torch.cat(adversarial_input_ids, dim=0)
adversarial_attention_masks = torch.cat(adversarial_attention_masks, dim=0)
adversarialLabel = torch.tensor(adversarialLabel)

adversarial_set = [(adversarial_input_ids[i], adversarial_attention_masks[i], adversarialLabel[i]) for i in range(5)]

labels_flat, pred_flat = findLabelsPreds(adversarial_set)
print("Qualitative Analysis on Adversarial Examples\n Correct Labels: ",labels_flat,"\n Predict Labels: ", pred_flat)
for i in range(len(labels_flat)):
  #Find index of non-matching labels and corresponding sentences
  if pred_flat[i] != labels_flat[i]:
    print(f"User Comment {i+1}: {adversarialText[i]}")
```

```
Qualitative Analysis on Test Set
 Correct Labels:  [1 0 1 1 0 1 1 1 0 1 1 0]
 Predict Labels:  [1 0 1 1 0 0 1 1 0 1 1 0]
User Comment 6:  Your skincare routine is a joke. You need to do more research before recommending products.
Since only one example is wrong , designing 5 new Adversarial examples
Qualitative Analysis on Adversarial Examples
 Correct Labels:  [0 0 1 1 1]
 Predict Labels:  [1 1 0 0 0]
User Comment 1: Nobody cares about your vacation. Stop spamming us with your photos.
User Comment 2: You're all idiots for even talking about this. You don't know what you're talking about.
User Comment 3: Information easily available. Leave!
User Comment 4: I appreciate the thoughtful discussion in this forum. However its naive and dumb.
User Comment 5: The author of this article is dumb!
```

DESCRIBE YOUR QUALITATIVE ANALYSIS OF THE ABOVE EXAMPLES HERE

The task of classifying sentences based on the degree of toxicity is extremely vague. A deeper analysis needs to be done to differentiate between toxic and non-toxic writing styles. This may be even more difficult to determine in the case of social media user comments, where the number of characters is sometimes restricted, the tone of the sentence, slang language used in the context, and individual perception as a receiver may vary.

Sentences that are being mislabelled in the test/adversarial set to deal with the issue of ambiguity or degree of toxicity. Obviously, style of communication is an issue that has no particular way, equally affecting both toxic and non-toxic user comments, which might be why the model would have a hard time classifying these user comments as toxic or non-toxic.

Finished? Remember to upload the PDF file of this notebook to Gradescope **AND** email your three dataset files (annotator1.csv, annotator2.csv, and final_data.csv) to cs685instructors@gmail.com with the subject line formatted as **Firstname_Lastname_HW1data**.

AI Disclosure

- Did you use any AI assistance to complete this homework? If so, please also specify what AI you used.
  - *No*

(only complete the below questions if you answered yes above)

- If you used a large language model to assist you, please paste *all* of the prompts that you used below. Add a separate bullet for each prompt, and specify which problem is associated with which prompt.
  - *your response here*
- **Free response**: For each problem for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good answer, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to get the answer or check your own answer?
  - *your response here*

✓ 0s completed at 10:36 PM