# Homework 1, CS685 Spring 2022

This is due on March 25th, 2022. 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:***

---

*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.

---

## Part 1: Annotation

# 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.

---

I've chosen tweets dataset. I found from Kaglle. It was easy to find, to download and it served the purpose so I chose it. I selected 120 sentences by first making sure that they can be surely

classified as "positive" and "negative" and they don't have very similar structure. With this classificaiton task I'll classify them as "negative" or "positive".

## 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!

---

categories -> "positive", "negative". For the given "text" column one of these categories must be chosen and inserted into the "sentiment" column.

positive:

- if a sentence conveys a positive tone
- talks about a positive event
- shares happiness

negative:

- if a sentence conveys a negative tone
- talks about a negative event
- shares a negative emotion or an event

Note: All sentences selected so that they're either positive or negative.

## 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.*

## 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.

- it was relatively ease task since it was a binary classification
- some examples such as "On the way to Malaysia...no internet access to Twit" doesn't contain any emotion. However as guideline explains with common sense it can be labeled as "negative"
- I've chosen the data carefuly so that there's room for minimal confusion. It's successful. I would repeat the process.

## 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.*

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

```
Mounted at /content/drive
```

```
### WRITE CODE TO LOAD ANNOTATIONS AND
import pandas as pd
from sklearn.metrics import cohen_kappa_score

df1 = pd.read_csv("/content/drive/MyDrive/Datas/courses/twitter-sentiment-user1.csv
df2 = pd.read_csv("/content/drive/MyDrive/Datas/courses/twitter-sentiment-user2.csv

df1_sent = df1["sentiment"].tolist()
df2_sent = df2["sentiment"].tolist()

df1_len = len(df1_sent)
df2_len = len(df2_sent)
```

```
print(f"Length of user1 annotation: {df1_len}")
print(f"Length of user2 annotation: {df2_len}")

raw_agreement = 0
for u1, u2 in zip(df1_sent, df2_sent):
  if u1 == u2:
    raw_agreement += 1

print(f"Raw agreement: {raw_agreement/df1_len}")
print(f"Cohen's Kappa: {cohen_kappa_score(df1_sent, df2_sent)}")
### COMPUTE AGREEMENT + COHEN'S KAPPA HERE!
```

```
Length of user1 annotation: 119
Length of user2 annotation: 119
Raw agreement: 0.9495798319327731
Cohen's Kappa: 0.8986083499005965
```

*RAW AGREEMENT*: 0.94

*COHEN'S KAPPA*: 0.89

### 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!

---

I chose the one that I agree the most.

## 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

# 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")
#device="cpu"
```

```
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-w
Collecting transformers
  Downloading transformers-4.26.1-py3-none-any.whl (6.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.3/6.3 MB 87.5 MB/s eta 0:00:0(
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dis
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/d
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 95.1 MB/s eta 0:00:0(
Collecting huggingface-hub<1.0,>=0.11.0
  Downloading huggingface_hub-0.12.1-py3-none-any.whl (190 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.3/190.3 KB 27.1 MB/s eta 0:00:
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/d
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/
Installing collected packages: tokenizers, huggingface-hub, transformers
Successfully installed huggingface-hub-0.12.1 tokenizers-0.13.2 transformers-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
import pandas as pd

#df = pd.read_csv('final_data.csv')
#df = pd.read_csv('tweets.csv')
df = pd.read_csv("/content/drive/MyDrive/Datas/courses/twitter-sentiment-final.csv"
df["label"] = df.label.map({'positive': 1, 'negative': 0})

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])
```

## 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,
test_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_val +

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

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.to(device)
```

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

```
                    )
epochs = 3
```

# 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 / 3 ========
Training...
Total loss: 0.025970637914724648
Validation accuracy: 0.7272727272727273

======== Epoch 2 / 3 ========
Training...
Total loss: 0.0018802960112225264
Validation accuracy: 0.7272727272727273

======== Epoch 3 / 3 ========
Training...
Total loss: 0.0006980888283578679
Validation accuracy: 0.9090909090909091

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)
```

```
0.7692307692307693
```

## 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?

---

Batch size has to be something that fits into the memory. We have very small dataset anyways so I chiose eight. For AdamW parametest I tried to go up and down however current setting is the best.

There's a gap for accuracy between train and validation. There may be couple of reasons for thes. We didn't do stratify split so model may not have learned a specific label well. Dataset is very small and has risk of overfitting. Validation set is small to so it may not convey variety of information that model has learned.

## 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
import numpy as np
# function to get validation accuracy
def get_validation_performance(val_set):
    counter = 0
```

```
# 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

    for idx, value in enumerate(pred_flat):
      if value != labels_flat[idx]:
```

```
            print(test_text[counter])
          counter += 1

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

get_validation_performance(test_set)
## print out up to 5 test set examples (or adversarial examples) that your model ge
     4am. And Im on the beach. Pretty
      Not a prob hun
     What better way to spoil mum than to let her kick back and relax over a nice m
     0.7692307692307693
```

First thing that I can observe is that there's no %100 "positive" or "negative" keywords here. Like I mentioned at the hard points of annotation they're a bit subjective. It may be positive to some, negative to others.

---

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**.