BERT Sentiment Analysis to Detect Twitter Sarcasm
(Naive Approach)
(Draft)

At the final stage of your project, you need to deliver the following:

- *Your documented source code and test set predictions.*
- *Explain your model, and how you perform the training. Describe your experiments with other methods that you may have tried and any hyperparameter tuning.*
- *A demo that shows your code can actually run on the test set and generate your submitted predictions. You don't need to run the training process during the demo. If your code takes too long to run, try to optimize it, or write some intermediate results (e.g. inverted index, trained model parameters, etc.) to disk beforehand.*

Abstract

I present a ***Naive*** approach to detect Twitter tweet sarcasm using a transformers-based pre-trained model that consider only the target utterance (***response***). This approach completely ignores the ***context*** of the response. My best model gives ***F1***-score of ***75.79%****, beating the baseline score after 4 epoch iterations.*

Introduction

Sarcasm is a form of figurative language that implies a negative sentiment while displaying a positive sentiment on the surface (Joshi et al., 2017)

This project presents a transformer-based sarcasm detection model that takes only target utterance and its context and predicts if the target utterance involves sarcasm. Our model uses a transformer encoder to coherently generate the embedding representation for the target utterance. This approach is evaluated on two Twitter. Depicts significant improvement over the baseline using only the target utterance as input.

Dataset Description

Each line contains a JSON object with the following fields:

- ***response***: the Tweet to be classified
- ***context***: the conversation context of the response
    - Note, the context is an ordered list of dialogue, i.e., if the context contains three elements, c1, c2, c3, in that order, then c2 is a reply to c1 and c3 is a reply to c2. Further, the Tweet to be classified is a reply to c3.
- ***label***: SARCASM or NOT_SARCASM
- ***id***: String identifier for sample. This id will be required when making submissions. (ONLY in test data)

Dataset size statistics

Train	Test
5000	1800

BERT Model used for training:

- BERT Large uncased
- BERT Base uncased

Training parameters:

- Learning rate: 2e-5
- Epsilon: 1e-8
- Random seed value: 17
- Epochs: 4 iterations
- Environment: Google Colab PRO

Naive and Context

I hypothesize the context does not ***always*** support the sentiment of a response. Context can have opposing affect to the sentiment of a response.

I hypothesize there are 2 types of contexts:

1. ***Positive context*** is the context that support the sentiment of a response.
2. ***Negative context*** is the context that does not support the sentiment of a response.

Hypothesis-1: Context reduce sentiment quality

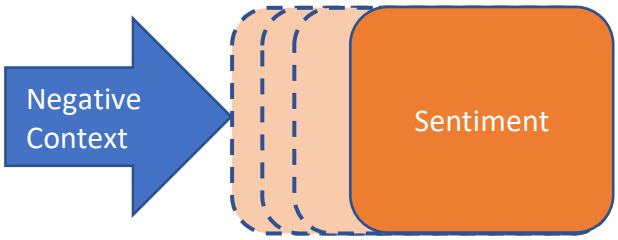


Fig.1: Illustration of context reduce sentiment quality

Hypothesis-2: Context increase sentiment quality

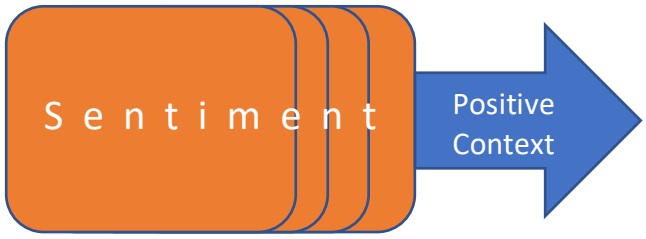


Fig.2: Illustration of context increase sentiment quality

To train the model it is important to select the context that support the sentiment of a response. In this project I chose to completely ignore the context. This approach I call it a Naive approach. I am using only the sentiment-labelled response to train the model.

In the future project, I can use advance machine learning techniques to utilize both response and context to train the model.

BERT model maximum character is 512. The training response maximum character length is 315. BERT model can consider all characters in the response.

```
import pandas as pd
import json

with open('sample_data/train.jsonl') as f:
    lines_train = f.read().splitlines()
with open('sample_data/test.jsonl') as f:
    lines_test = f.read().splitlines()

df_train = pd.json_normalize(pd.DataFrame(lines_train)[0].apply(json.loads))
df_test = pd.json_normalize(pd.DataFrame(lines_test)[0].apply(json.loads))

print(f'Maximum train-response characters length: {df_train.response.str.len().max()}')
print(f'Maximum test-response characters length: {df_test.response.str.len().max()}')
```

```
Maximum train-response characters length: 315
Maximum test-response characters length: 310
```

Maximum Training and Evaluation Response characters length

Software Code

I implemented software code into 2 Google Colab Notebooks:

1. Training and evaluation notebook: *NAIVE_BERT_sentiment_analysis.ipynb* (link)
2. Evaluate selected trained-model notebook (for DEMO purpose): *DEMO_Model_Evaluation.ipynb* (link)

Environment Google Colab PRO.

A. NAIVE_BERT_sentiment_analysis.ipynb

1. Colab Configuration
   a. Python module installation

```
!pip install transformers
!pip install PyDrive
```

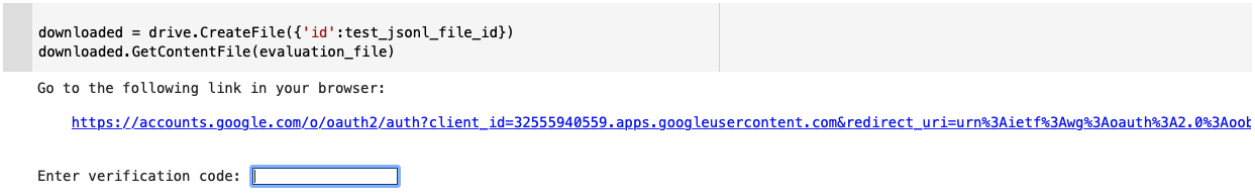
   b. Copy train.jsonl and test.jsonl files from Google Drive to Colab session

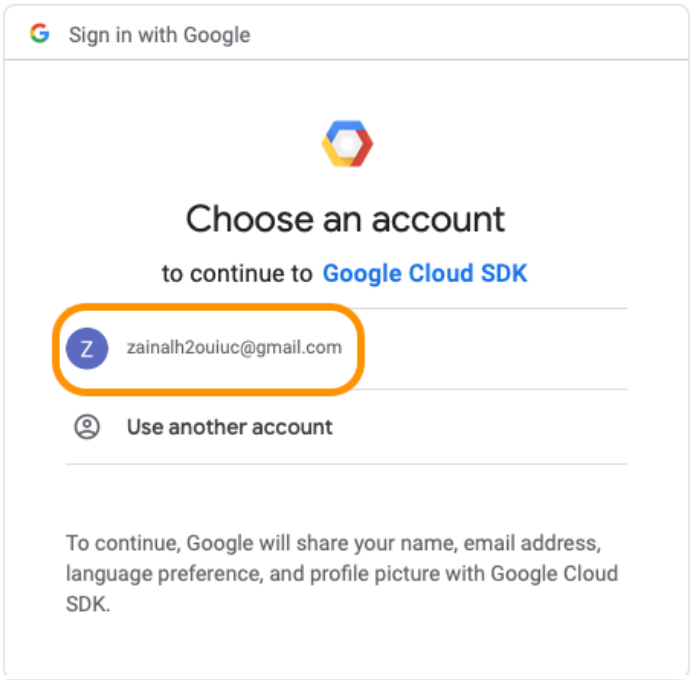
I have already copied train.json and test.jsonl files to my Google Drive account created for this project. I have shared the files to public. The following code will copy the files to the current running Colab session:

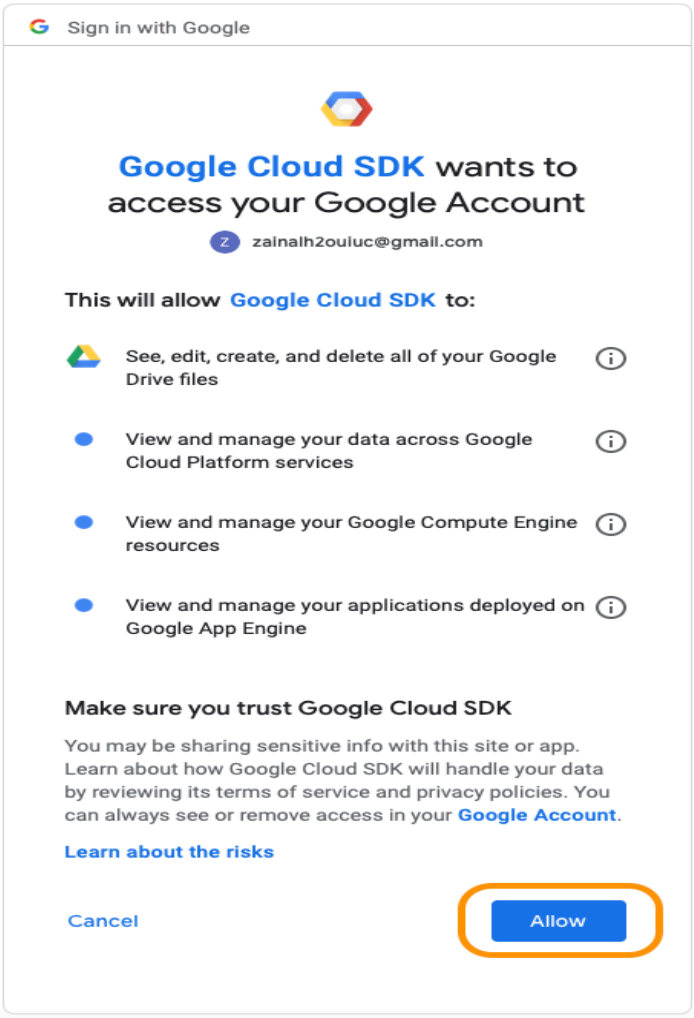
```
# train.json file location: https://drive.google.com/file/d/1d5lwaHPOUBAz7c-cNXXQeFn75ZV2HkUh/view?usp=sharing
# test.jsonl file location: https://drive.google.com/file/d/1vA3uyqy1TZmahgZ0PeNRFx67LuYeAkoW/view?usp=sharing

########################
# The training dataset #
########################
# Google Drive file name
training_file = 'train.jsonl'
# Google Drive unique file ID
training_file_id = '1d5lwaHPOUBAz7c-cNXXQeFn75ZV2HkUh'

##################################
# The evaluation/testing dataset #
##################################
# Google Drive file name
evaluation_file = 'test.jsonl'
# Google Drive unique file ID
test_jsonl_file_id = "1vA3uyqy1TZmahgZ0PeNRFx67LuYeAkoW"
```

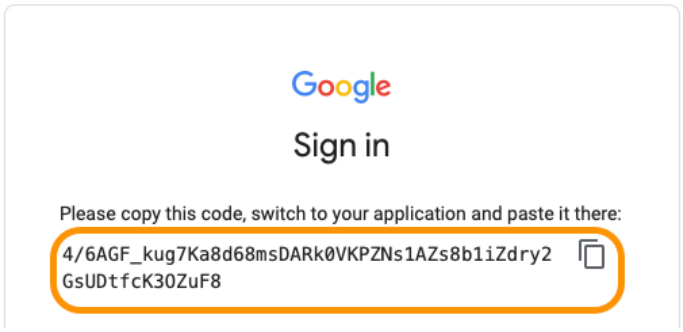
```
# The files are available to public in my Google Drive
# created for this project.
#
# Login using Google Account to proceed.
#
# Follow the link, copy-paste the code.

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

downloaded = drive.CreateFile({'id':training_file_id})
downloaded.GetContentFile(training_file)

downloaded = drive.CreateFile({'id':test_jsonl_file_id})
downloaded.GetContentFile(evaluation_file)
```

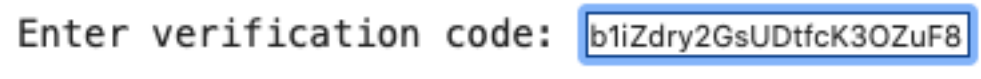
This source code will prompt you to a URL. Login to your Google account, click the link to get a code and paste the code to the “Enter verification code:” text box, then press “Enter” key.

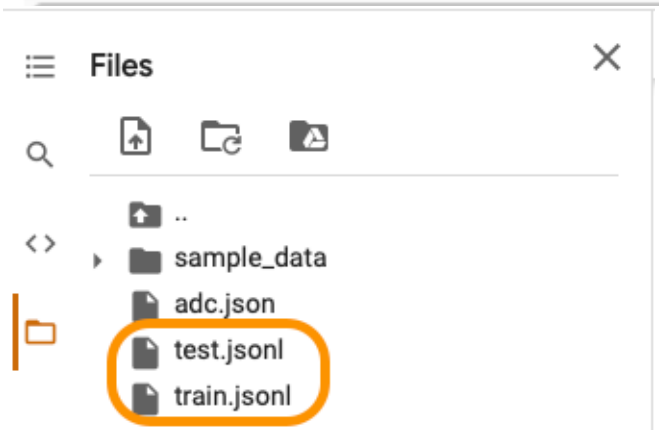



Sign in with Google
Choose an account
to continue to Google Cloud SDK
zainalh2ouiuc@gmail.com
Use another account
To continue, Google will share your name, email address, language preference, and profile picture with Google Cloud SDK.


Sign in with Google
Google Cloud SDK wants to access your Google Account
zainalh2ouiuc@gmail.com
This will allow Google Cloud SDK to:
See, edit, create, and delete all of your Google Drive files
View and manage your data across Google Cloud Platform services
View and manage your Google Compute Engine resources
View and manage your applications deployed on Google App Engine
Make sure you trust Google Cloud SDK
You may be sharing sensitive info with this site or app. Learn about how Google Cloud SDK will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.
Learn about the risks
Cancel
Allow







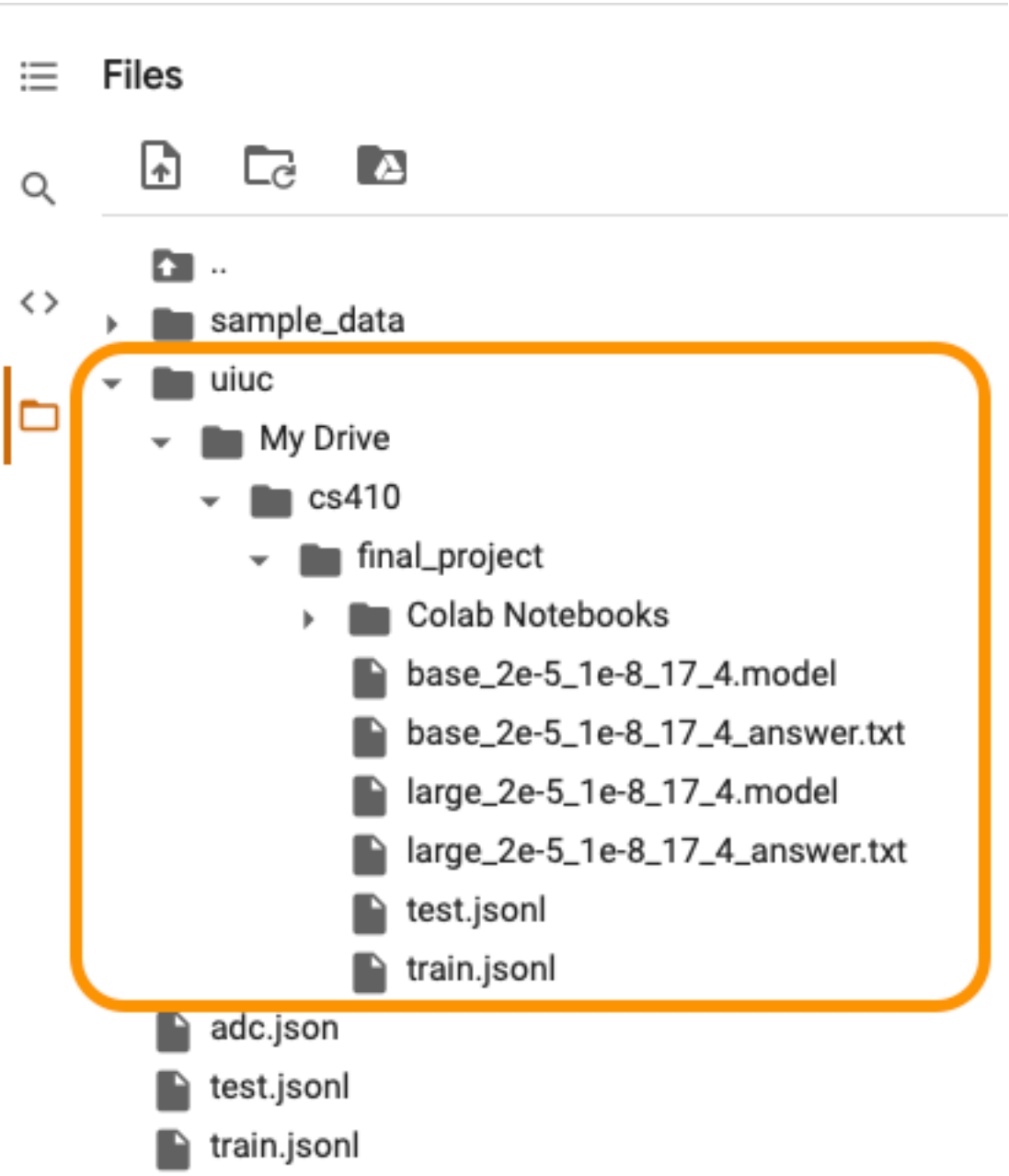
At this point have our training and testing datasets in the current running Colab session!

2. Mounting Google Drive to Colab session (To save result files)

```
# Mount Google Drive to the current Colab session
# The trained model and answer file are copied to
# the Google Drive.
from google.colab import drive
drive.mount('/content/uiuc')
```

```
Mounted at /content/uiuc
```

Follow the authentication process as described in the step above, when prompted, to mount Google Drive to the current Colab session. This allow us to have access to Google Drive directory where we will store our result files. In this project, I mounted my Google Drive directory to '***./content/uiuc***' folder.



Mounted Google Drive in Colab Session where to keep result files permanently

3. The main Python Class
   In this project I implemented a Python class that handles the following tasks:
   - Read the dataset from jsonl files into a list of json
   - Convert list of json to Pandas DataFrame
   - Create the BERT Model
   - Run the training and save the model for each epoch
   - Evaluate the model and store the result into file

   Below is the class signatures (for details source code please check it here link)

```
class BERT_Model:
  # Class Initialization (set the hyperparameters here)
  def __init__(self):
  # Read train.jsonl or test.jsonl
  # and store it to Pandas DataFrame
  def read_dataset_file(self, the_file):
  # We create BERT model here
  def create_bert_model(self, df, lr, eps):
  # Training the model using train.jsonl
  def run_training(self, file_save, seed_val=17):
  # Evaluate the trained-model using test.jsonl
  def evaluate_model(self, file_name, df):
  # Write the evaluation result to answer.txt
  # We will publish answer.txt to LiveDataLab
  # for F1, Precision and Recall score evaluation!
  def write_answers(self, file_name, preds_flat):
```

4. Training and Evaluation experiments
   To test my hypothesis, I run only 2 experiments with the same hyperparameters. I used 2 BERT models:
   - BERT base uncased
   - BERT LARGE uncased

   Hyperparameters:
   - Learning rate: 2e-5
   - Epsilon: 1e-8
   - Seed value for random: 17
   - EPOCH: 4 iterations

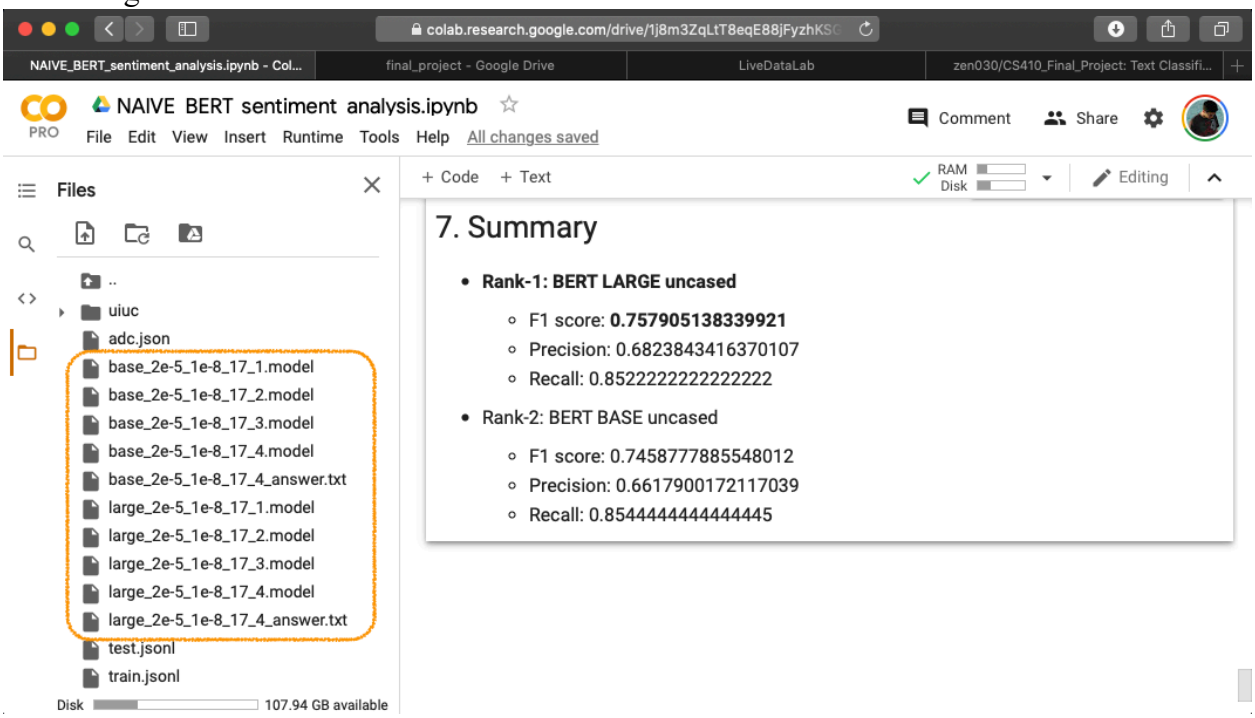
   1. Experiment-1: BERT base uncased

```
the_model = BERT_Model()
the_model.model_name = 'bert-base-uncased'
epoch_to_evaluate = 4

the_model.df_train = the_model.read_dataset_file(training_file)
the_model.df_eval = the_model.read_dataset_file(evaluation_file)

the_model.create_bert_model(the_model.df_train, 2e-5, 1e-8)
the_model.run_training('base_2e-5_1e-8_17', 17)
preds_flat = the_model.evaluate_model(f'base_2e-5_1e-8_17_{epoch_to_evaluate}.model', the_model.df_eval)
the_model.write_answers(f'base_2e-5_1e-8_17_{epoch_to_evaluate}_answer.txt', preds_flat)
```

   2. Experiment-2: BERT LARGE uncased

```
the_model = BERT_Model()
the_model.model_name = 'bert-large-uncased'
epoch_to_evaluate = 4

the_model.df_train = the_model.read_dataset_file(training_file)
the_model.df_eval = the_model.read_dataset_file(evaluation_file)

the_model.create_bert_model(the_model.df_train, 2e-5, 1e-8)
the_model.run_training('large_2e-5_1e-8_17', 17)
preds_flat = the_model.evaluate_model(f'large_2e-5_1e-8_17_{epoch_to_evaluate}.model', the_model.df_eval)
the_model.write_answers(f'large_2e-5_1e-8_17_{epoch_to_evaluate}_answer.txt', preds_flat)
```

This will generates the files in the current Colab session folder as shown below:



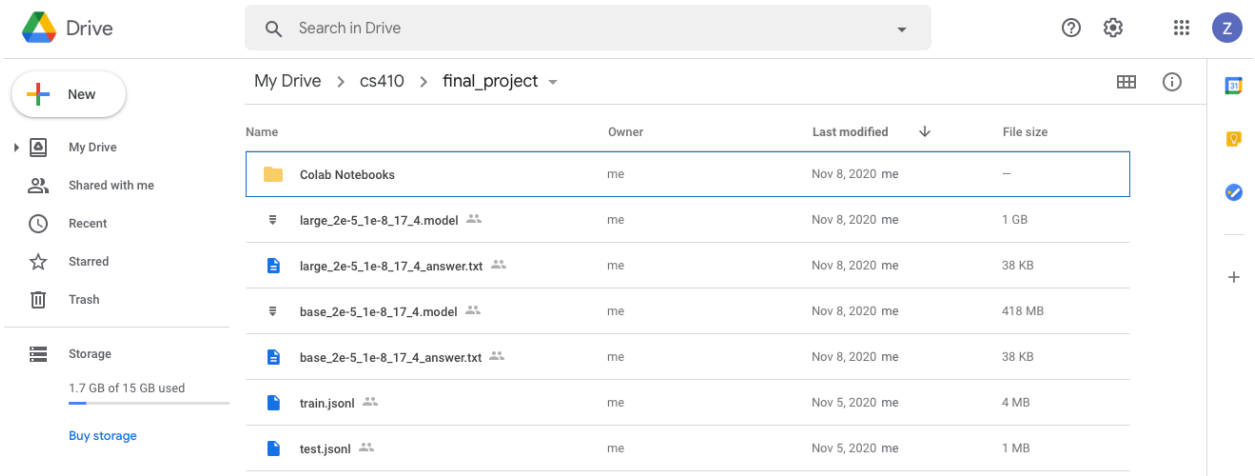
5. Save the result files to Google Drive
   The files in Colab session will be deleted when Colab session is ended. We need to store the files permanently in other location, in my case, in the project I use Google Drive.

   The code below will store the result files to my Google Drive folder using the folder mounted in the earlier step. In this project, I only store EPOCH # 4 model and evaluation result.

```
!cp 'base_2e-5_1e-8_17_4_answer.txt' 'uiuc/My Drive/cs410/final_project/base_2e-5_1e-8_17_4_answer.txt'
!cp 'base_2e-5_1e-8_17_4.model' 'uiuc/My Drive/cs410/final_project/base_2e-5_1e-8_17_4.model'
```

```
!cp 'large_2e-5_1e-8_17_4_answer.txt' 'uiuc/My Drive/cs410/final_project/large_2e-5_1e-8_17_4_answer.txt'
!cp 'large_2e-5_1e-8_17_4.model' 'uiuc/My Drive/cs410/final_project/large_2e-5_1e-8_17_4.model'
```

   The files are copied to the Google Drive folder:



We will use the files from Google Drive to run a DEMO which I will illustrate in the following section.

B. *DEMO_Model_Evaluation.ipynb*

In the demo, I will demonstrate how to generate 'answer.txt' from the BERT LARGE uncased trained model. The trained model is available to the public here link. With this model, we will create the reproduce evaluation result which is available to the public here link.

The first 3-steps are already explained in the previous section:

1. Colab configuration
2. Copy result files from Google Drive to Colab session, files copied are:
    a. large_2e-5_1e-8_17_4.model
    b. test.jsonl
3. Prepare Panda DataFrame for the testing dataset
4. Evaluate the model using testing dataset
    a. Load the model from the trained-model file
    b. Create the tokenizer to encode testing dataset
    c. Create the data loader to run the evaluation batch iterations

```
# 1. Encode the data
# 2. Create Tensor Dataset
# 3. Create Dataloader for the evaluation

bert_model = 'bert-large-uncased'
batch_size = 5

tokenizer = BertTokenizer.from_pretrained(bert_model, do_lower_case=True)

encoded_data_evaluation = tokenizer.batch_encode_plus(
    df.response.values,
    add_special_tokens=True,
    return_attention_mask=True,
    max_length=max_length,
    padding='max_length',
    return_tensors='pt'
)

input_ids_evaluation = encoded_data_evaluation['input_ids']
attention_masks_evaluation = encoded_data_evaluation['attention_mask']

dataset_evaluation = TensorDataset(input_ids_evaluation, attention_masks_evaluation)

dataloader_eval = DataLoader(dataset_evaluation, sampler=SequentialSampler(dataset_evaluation), batch_size=batch_size)
```

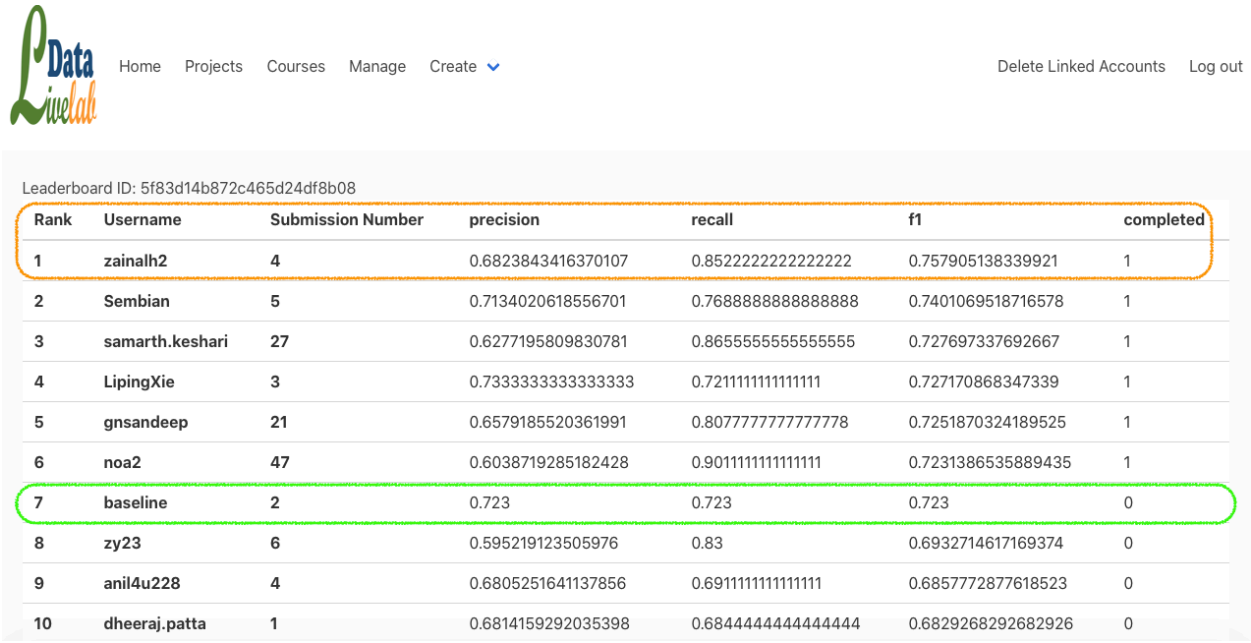
5. Run the evaluation batch iteration

```
# If GPU is available.
if torch.cuda.is_available():
    # PyTorch to use the GPU
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('We will use the GPU:', torch.cuda.get_device_name(0))
# If GPU is not available. Use the CPU.
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")

# To set the model into a training mode
label_dict = {'SARCASM': 0, 'NOT_SARCASM': 1}

# Load the pre-trained model
model = BertForSequenceClassification.from_pretrained(bert_model,
                                                      num_labels=len(label_dict),
                                                      output_attentions=False,
                                                      output_hidden_states=False)
model.to(device)
model.load_state_dict(torch.load(model_file, map_location=torch.device(device)))

# Set the model to evaluation/testing mode
model.eval()
loss_val_total = 0
predictions = []

# Iterate the evaluation/testing data loader
progress_bar = tqdm(dataloader_eval, leave=False, disable=False)
for batch in progress_bar:
  batch = tuple(b.to(device) for b in batch)
  inputs = {'input_ids': batch[0], 'attention_mask': batch[1]}

  with torch.no_grad():
    # evaluate the validation dataset
    output = model(**inputs)
    logits = output[0]
    logits = logits.detach().cpu().numpy()
    predictions.append(logits)

predictions = np.concatenate(predictions, axis=0)
preds_flat = np.argmax(predictions, axis=1).flatten()


print('######################')
print('# Evaluation is done #')
print('######################')
```

6. Generate the ‘anwer.txt’ file

```
# Print the answer.txt file using the evaluation/testing result.
#
# The same file I have generated using the code in this demo:
# https://drive.google.com/file/d/14tLEIr07SK4uq5cx7lrUOj_IzJZuUGYO/view?usp=sharing
#
# This is the same file I submitted to LiveDataLab Leaderboard for evaluation.

f = open('answer.txt',"w")
i = 1
for pred in enumerate(preds_flat):
  if pred[1] == 0:
    text = 'SARCASM'
  else:
    text = 'NOT_SARCASM'
  print('twitter_{0},{1}'.format(i, text))
  f.write('twitter_{0},{1}\n'.format(i, text))
  i = i + 1
f.close()
```

7. Post ‘answer.txt’ to LiveDataLab for scoring



Rank	Username	Submission Number	precision	recall	f1	completed
1	zainalh2	4	0.6823843416370107	0.8522222222222222	0.757905138339921	1
2	Sembian	5	0.7134020618556701	0.7688888888888888	0.7401069518716578	1
3	samarth.keshari	27	0.6277195809830781	0.8655555555555555	0.727697337692667	1
4	LipingXie	3	0.7333333333333333	0.7211111111111111	0.727170868347339	1
5	gnsandeep	21	0.6579185520361991	0.8077777777777778	0.7251870324189525	1
6	noa2	47	0.6038719285182428	0.9011111111111111	0.7231386535889435	1
7	baseline	2	0.723	0.723	0.723	0
8	zy23	6	0.595219123505976	0.83	0.6932714617169374	0
9	anil4u228	4	0.6805251641137856	0.6911111111111111	0.6857772877618523	0
10	dheeraj.patta	1	0.6814159292035398	0.6844444444444444	0.6829268292682926	0

Leaderboard snapshot on 03-Nov-2020

Result and Conclusion

Model	F1-Score	Recall	Precision
BERT Large uncased	**0.757905138339921**	0.8522222222222222	0.6823843416370107
BERT Base uncased	0.7458777885548012	0.8544444444444445	0.6617900172117039

Surprisingly, Base model performs almost as good as Large model.

In this project, I did try to use different trained model such as RoBERTa and XLNet (and different hyperparameters), but for a reason I could not produce result higher than BERT Large uncased score. In this project as proposed in the project proposal, I am reporting the result for BERT model only.

In the future, I would like to explore more on the following topics:

- Using advance machine learning technique to evaluate hyperparameters. I use the original BERT paper (link) to choose hyperparameters for my experiments.
- Utilizing the context instead of using response only. I hypothesize that context if handles properly can be used as an additional dataset to train the model.
- Explore another model such as RoBERTa and XLNet.

Reference

Aditya Joshi, Pushpak Bhattacharyya, and Mark J. Car- man. 2017. Automatic Sarcasm Detection: A Survey. *ACM Computing Surveys*, 50(5):1–22.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.