Project Final Report

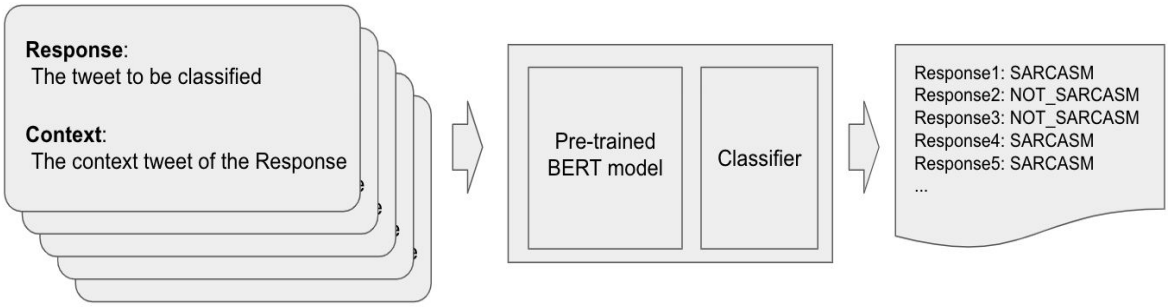
Text Classification Competition: Twitter Sarcasm Detection

Ryo Takaki

1. Overview of functions

1.1 Twitter Sarcasm Detection

The main function of this system is to predict a label(SARCASM or NOT_SARCASM) of input tweets. The input tweets include response tweets and its context tweets. We are given 5,000 training data sets and 1,800 test data sets. The prediction performance will be evaluated using F1 score and compared to the baseline score.



Input: Pairs of the tweets(response and context tweets)
Output: Predicted labels of the response tweets(SARCASM or NOT_SARCASM)
Process: Pre-trained BERT model + single layer classifier

1.2 Code outline

The code of the twitter sarcasm detection system mainly consists of the components below. Detailed instructions of how to implement those components are explained in the next section.

No	Component	Function outline
1	Library installation	Install the Hugging Face Library for BERT model
2	Context selection	Derive the latest tweet from context tweets
3	Removing @USER	Remove unnecessary words from the input tweets
4	Tokenization	Convert the input text to the tokenized IDs
5	Formatting	Reshape the input IDs size with fixed length
6	Data split	Split the data into training and validation data set
7	Model training	Build and train the BERT model with recommended parameters
8	Label prediction	Predict the labels using test set data

2. Implementation

2.1 Library Installation

BERT pre-trained model and its library can be used by just installing the library.
https://huggingface.co/transformers/

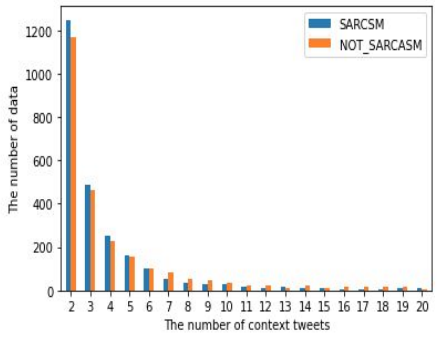
```
In [3]: # In order to use a BERT framework, install the Hugging Face library
        !pip install transformers
```

2.2 Context selection

As you can see that some contexts have several context tweets(e.g. 20), however the majority have only two to five context tweets. Although we can see that at least two tweets are included in the context information, I decided to use only one latest context tweet as a first step. In the actual code below, the derived latest context tweet is stored as a new column(pre_comment).

```
In [8]: # Check a number of comments for each tweets
        import re

        # This function it to devide the several context tweets by using specific patterns
        def split_context(a_string):
            stripped_string = str(a_string).strip('[]').strip('\'\"')
            pattern = '\", \"|\', \'|\', \"|\", \''
            splitted_string = re.split(pattern, stripped_string)
            splitted_string = [x for x in splitted_string if x is not None]
            return pd.Series([splitted_string[0], len(splitted_string)])

        # Create "comment_num" column from context information
        df[['pre_comment', 'comment_num']] = df['context'].apply(split_context)
        #df.sample(n=5, random_state=0)

        # Number of training data(SARCASM) corresponding to the number of comments
        num_sarcasm_data_by_comments = df[df['label']=='SARCASM']['comment_num'].value_counts().sort_index()

        # Number of training data(NON_SARCASM) corresponding to the number of comments
        num_non_sarcasm_data_by_comments = df[df['label']=='NOT_SARCASM']['comment_num'].value_counts().sort_index()

        # Graph
        val1 = num_non_sarcasm_data_by_comments
        val2 = num_sarcasm_data_by_comments
        index = num_non_sarcasm_data_by_comments.index
        df_plot = pd.DataFrame({'SARCSM': val1, 'NOT_SARCASM': val2}, index=index)
        ax = df_plot.plot.bar(rot=0)
        ax.set_xlabel('The number of context tweets')
        ax.set_ylabel("The number of data")
```

```
Out[8]: Text(0, 0.5, 'The number of data')
```



2.3 Removing @USER

As you can see, the original response and context include unnecessary words like “@USER”.

Out[6]:

	label	response	context
398	SARCASM	Countered with #climatechange activists stuck ...	[One good #climatechange meme deserves a story...
3833	NOT_SARCASM	@USER @USER @USER When we share love , we get ...	[" Heaven on Earth is a choice you must make ,...
4836	NOT_SARCASM	@USER @USER @USER Give me nothing . Just sayin...	[@USER @USER @USER A question noone is asking ...
4572	NOT_SARCASM	🍗Sinning happens nearly everywhere . As a res...	[🍗The Spiritual explanation for cancer appear...
636	SARCASM	@USER I was going for the . PV method gives MM...	[In PEI Electoral Reform Plebiscite , FPTP get...

The unnecessary word(@USER) is removed by this function.

In [9]:
```
# Remove unnecessary string(@USER)
def remove_strings(a_string):
    stripped_string = str(a_string).lstrip('(@USER )')
    return stripped_string

df['response'] = df['response'].apply(remove_strings)
df['pre_comment'] = df['pre_comment'].apply(remove_strings)
df.loc[:,['response', 'pre_comment']].sample(n=5, random_state=0)
```

Out[9]:

	response	pre_comment
398	Countered with #climatechange activists stuck ...	One good #climatechange meme deserves a story ...
3833	When we share love , we get love too . Our vib...	Heaven on Earth is a choice you must make , no...
4836	Give me nothing . Just saying no one really kn...	A question noone is asking is ' Parnas the ins...
4572	🍗Sinning happens nearly everywhere . As a res...	🍗The Spiritual explanation for cancer appeari...
636	I was going for the . PV method gives MMP the ...	In PEI Electoral Reform Plebiscite , FPTP gets...

2.4 Tokenization

In order for the BERT model to handle the input tweets, the original tweets sentences will be converted to tokenized words and its IDs.

In [12]:
```
# The code in this cell is inspired by https://mccormickml.com/2019/07/22/BERT-fine-tuning/

print(' Original: ', df['response'][1])
print('Tokenized: ', tokenizer.tokenize(df['response'][1]))
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(df['response'][1])))
```

```
 Original:  trying to protest about . Talking about him and his labels and they label themselves WTF does that make em ?
Tokenized:  ['trying', 'to', 'protest', 'about', '.', 'talking', 'about', 'him', 'and', 'his', 'labels', 'and', 'they', 'label', 'themselves', 'w', '##tf', 'does', 'that', 'mak
e', 'em', '?']
Token IDs:  [2667, 2000, 6186, 2055, 1012, 3331, 2055, 2032, 1998, 2010, 10873, 1998, 2027, 3830, 3209, 1059, 24475, 2515, 2008, 2191, 786
1, 1029]
```

As we can see, the original tweet was converted to the tokenized words and its token ID.

2.5 Formatting

In order to define the BERT model’s input dimensions, we need to identify the maximum length of the input tweets(response + context). Thanks to the library, what we need to do to create the fixed size input data is to set the maximum input length when we encode the data. Even if the actual input data length is shorter than the maximum input length, remaining parts will be automatically filled with zero.

In [14]:
```
# The code in this cell is inspired by https://huggingface.co/transformers/v2.4.0/glossary.html
# encoded_sequence = tokenizer.encode(sequence_a, sequence_b)

def get_input_len(df):
   return len(tokenizer.encode(df['input_pair'][0], df['input_pair'][1]))
df['len_input_pair'] = df.apply(get_input_len, axis=1)
print('The maximum length of the input is {}' .format(df['len_input_pair'].max()))
```

```
The maximum length of the input is 180
```

-> We will use "185" as the maximum length (actual max length(180) + margin(5))

3.2.2 Data encoding

In [15]:
```
# The code in this cell is inspired by the tech report made by my UIUC's class mate(zen030).
# https://github.com/zen030/tech_review/blob/master/techreview.pdf


# Encoding
encoded_data = tokenizer.batch_encode_plus(
                df['input_pair'],
                add_special_tokens=True,
                return_attention_mask=True,
                max_length=185,
                padding='max_length',
                return_tensors='pt'
            )

# Convert the lists into tensors
input_ids_all = encoded_data['input_ids']
attention_masks_all = encoded_data['attention_mask']
labels_all = torch.tensor(df['label_num'].values)
token_type_ids_all = encoded_data['token_type_ids']
```

In [16]:
```
print(encoded_data['input_ids'][0])
```

```
tensor([  101,  1045,  2123,  1005,  1056,  2131,  2023,  1012,  1012,  5525,
         2017,  2079,  2729,  2030,  2017,  2052,  1005,  2310,  2333,  2157,
         2247,  1012,  1012,  2612,  2017,  2787,  2000,  2729,  1998, 18792,
         2014,  1012,  1012,   102,  1037,  3576,  2775, 17210,  9394,  1998,
         2323,  2022,  2921,  2041,  1997,  4331,  1012, 17217,  6382,  2319,
         1010,  2017,  2323,  2022, 14984,  1997,  2115,  2200,  4854,  1998,
         5525, 25352,  2270,  6090,  4063,  2075,  1010,  1998,  2478,  1037,
         2775,  2000,  2079,  2009,  1012,   102,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     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])
```

We can see that input ids were padding with zero until the maxmum length(185)

2.6 Data split

5,000 Data sets are divided into 90% of training sets and 10% of validation sets randomly(although it is random selection, it is reproducible as the seed value is specified).

In [19]:
```
# The code in this cell is inspired by https://mccormickml.com/2019/07/22/BERT-fine-tuning/
from torch.utils.data import TensorDataset, random_split

# For reproducing the same result
seed_val = 34
torch.manual_seed(seed_val)

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids_all, attention_masks_all, labels_all, token_type_ids_all)

# Training: 90%, Validation: 10%
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size

# Select the data randomly
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

# Chcek the number of data just in case
print('Training data: {}'.format(train_size))
print('Validation data: {}'.format(val_size))
```

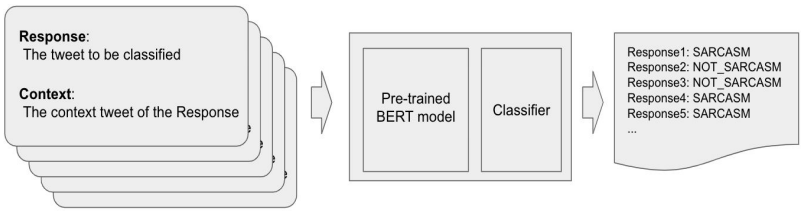
```
Training data: 4500
Validation data: 500
```

Data set were divided into the training data and validation data as expcted ratio.

2.7 Model training

2.7.1 Model definition

The model here consists of mainly two parts. One is the pre-trained BERT model and the other is single layer linear classification for this tweet classification application. In order to build the model, we only need to import “BertForSequenceClassification” from transformers like below, and specify some parameters. In our case, num_lebels is 2 as we want to classify the tweets into SARCASM or not.



In [21]:
```
# The code in this cell is inspired by https://huggingface.co/transformers/model_doc/bert.html
# and https://mccormickml.com/2019/07/22/BERT-fine-tuning/


from transformers import BertForSequenceClassification, AdamW, BertConfig
# "BertForSequenceClassification" consitsts of the pre-trained BERT model
# and single linear classification layer.

model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 2, # SARCASM or NOT_SARCASM
    output_attentions = False,
    output_hidden_states = False,
)

# The model will be on the GPU
model.cuda()
```

2.7.2 Trainable parameters setting

Finally, I decided not to change the trainable parameters of the BERT model as the performance with fixing the BERT model was not good enough. However, those parameters can be changed by enabling the cell below if needed for other purposes.

In [22]:
```
# Change trainable parameters
# https://github.com/huggingface/transformers/issues/400

# When I fixed the BERT pretrained paramters for the model training, final performance for our project was not better than that of the unfixed version.
# So, the code below are now commented out.
'''
for name, param in model.named_parameters():
    if 'classifier' not in name: # classifier layer
        #print(param.requires_grad)
        param.requires_grad = False

for name, param in model.named_parameters():
    print(param.requires_grad)
'''
```

2.7.3 Model training with recommended parameters

The batch size, the learning rate, the epsilon, and the number of epochs were determined based on the recommendation of the author of the BERT papar. Although I have not tried to change the parameters, It worked well at the first trial.

In [20]:
```
# The code in this cell is inspired by https://mccormickml.com/2019/07/22/BERT-fine-tuning/
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

# 16 or 32 are recommended by the BERT author
batch_size = 32

# DataLoaders for the training data set
train_dataloader = DataLoader(
            train_dataset,
            sampler=RandomSampler(train_dataset),
            batch_size=batch_size
        )

# DataLoaders for the validation data set
validation_dataloader = DataLoader(
            val_dataset,
            sampler=SequentialSampler(val_dataset),
            batch_size=batch_size
        )
```

In [23]:
```
# The code in this cell is inspired by the tech report made by my UIUC's class mate(zen030).
# https://github.com/zen030/tech_review/blob/master/techreview.pdf

# The learning rate and epsiron values are included in the values recommended by the BERT author.

optimizer = AdamW(model.parameters(),
                  lr = 2e-5,
                  eps = 1e-8
                )
```

In [24]:
```
from transformers import get_linear_schedule_with_warmup

# Number of training epochs
# From 2 to 4 is recommended by the BERT author.
epochs = 2

# Total number of training steps
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0,
                                            num_training_steps = total_steps)
```

```
============================================================
=== The model training will take about 5 minutes. Just a moment please. ===
============================================================

======== Epoch 1 / 2 ========
----------------------------------------
(1) Training phase
----------------------------------------
Batch   10  of   141.   Elapsed: 0:00:10.
Batch   20  of   141.   Elapsed: 0:00:20.
Batch   30  of   141.   Elapsed: 0:00:30.
Batch   40  of   141.   Elapsed: 0:00:41.
Batch   50  of   141.   Elapsed: 0:00:51.
Batch   60  of   141.   Elapsed: 0:01:01.
Batch   70  of   141.   Elapsed: 0:01:12.
Batch   80  of   141.   Elapsed: 0:01:22.
Batch   90  of   141.   Elapsed: 0:01:32.
Batch  100  of   141.   Elapsed: 0:01:42.
Batch  110  of   141.   Elapsed: 0:01:52.
Batch  120  of   141.   Elapsed: 0:02:02.
Batch  130  of   141.   Elapsed: 0:02:12.
Batch  140  of   141.   Elapsed: 0:02:23.

Average training loss: 0.55
Training epcoh took: 0:02:23

----------------------------------------
(2) Validation phase
----------------------------------------
Accuracy: 0.79
Validation Loss: 0.48
Validation took: 0:00:06

======== Epoch 2 / 2 ========
----------------------------------------
(1) Training phase
----------------------------------------
Batch   10  of   141.   Elapsed: 0:00:10.
Batch   20  of   141.   Elapsed: 0:00:20.
Batch   30  of   141.   Elapsed: 0:00:30.
Batch   40  of   141.   Elapsed: 0:00:41.
Batch   50  of   141.   Elapsed: 0:00:51.
Batch   60  of   141.   Elapsed: 0:01:01.
Batch   70  of   141.   Elapsed: 0:01:11.
Batch   80  of   141.   Elapsed: 0:01:21.
Batch   90  of   141.   Elapsed: 0:01:31.
Batch  100  of   141.   Elapsed: 0:01:42.
Batch  110  of   141.   Elapsed: 0:01:52.
Batch  120  of   141.   Elapsed: 0:02:02.
Batch  130  of   141.   Elapsed: 0:02:12.
Batch  140  of   141.   Elapsed: 0:02:22.

Average training loss: 0.41
Training epcoh took: 0:02:23

----------------------------------------
(2) Validation phase
----------------------------------------
Accuracy: 0.79
Validation Loss: 0.46
Validation took: 0:00:06

---------- Finish ----------
Total training time 0:04:58 (h:mm:ss)
```

2.8 Label prediction on the test data

By applying the same preprocessing as for the training/validation data set, the test data can also be inputted to the trained model. And finally, we can get the predicted labels by applying the softmax function to predicted values.

In [38]:
```
df['predicted_value'] = [item for l in predictions for item in l]
df['pred_flat'] = [np.argmax(item) for l in predictions for item in l]
df['pred_flat_label'] = df['pred_flat'].apply(lambda x: 'SARCASM' if x==1 else 'NOT_SARCASM')
df.loc[:,['id', 'response', 'pre_comment', 'pred_flat_label']]
```

Out[38]:

	id	response	pre_comment	pred_flat_label
0	twitter_1	My 3 year old , that just finished reading Nie...	Well now that ' s problematic AF <URL>	NOT_SARCASM
1	twitter_2	How many verifiable lies has he told now ? 15,...	Last week the Fake News said that a section of...	SARCASM
2	twitter_3	Maybe Docs just a scrub of a coach ... I mean ...	Let ' s Aplaud Brett When he deserves it he co...	SARCASM
3	twitter_4	is just a cover up for the real hate inside @U...	Women generally hate this president . What's u...	SARCASM
4	twitter_5	The irony being that he even has to ask why .	Dear media Remoaners , you excitedly sharing c...	SARCASM
...	...	...	...	...
1795	twitter_1796	is definitely the best out there . No question...	I have been a business customer of MWeb @USER ...	NOT_SARCASM
1796	twitter_1797	Ye let her out run wild and infect 10000 more ...	A woman refuses to have her temperature taken ...	SARCASM
1797	twitter_1798	Thanks for that , I would have never known .	The reason big government wants @USER out is b...	SARCASM
1798	twitter_1799	Yes also #found this on #new with loads of <UR...	Happy #musicmonday and #thanks for #all your 😎...	NOT_SARCASM
1799	twitter_1800	you still need to send the link to the fan you...	Not long wrapped on the amazing #January22nd W...	NOT_SARCASM

2.9 Competition result

The F1 score (0.737) beat the baseline score(0.723) as below.

Rank	Username	Submission Number	precision	recall	f1	completed
42	**ryotakaki**	**17**	0.6959526159921027	0.7833333333333333	0.7370622059592263	1
78	**baseline**	**2**	0.723	0.723	0.723	0

2.10 Reproducing the same prediction

After re-running all the code, a reproducibility test will be done automatically as below by downloading the actual submitted prediction file and comparing them to the prediction labels generated in your environment.

In [41]:
```
import pandas as pd
# Load the answer.txt and the reproduced_answer.txt as pandas dataframe
df = pd.read_csv('./answer.txt', header=None, names=["id","Pred_original"])
df_tmp = pd.read_csv('./reproduced_answer.txt', header=None, names=["id","Pred_reproduce"])
df = df.merge(df_tmp)
df['Same'] = (df['Pred_original'] == df['Pred_reproduce'])
df
```

Out[41]:

	id	Pred_original	Pred_reproduce	Same
0	twitter_1	NOT_SARCASM	NOT_SARCASM	True
1	twitter_2	SARCASM	SARCASM	True
2	twitter_3	SARCASM	SARCASM	True
3	twitter_4	SARCASM	SARCASM	True
4	twitter_5	SARCASM	SARCASM	True
...	...	...	...	...
1795	twitter_1796	NOT_SARCASM	NOT_SARCASM	True
1796	twitter_1797	SARCASM	SARCASM	True
1797	twitter_1798	SARCASM	SARCASM	True
1798	twitter_1799	NOT_SARCASM	NOT_SARCASM	True
1799	twitter_1800	NOT_SARCASM	NOT_SARCASM	True

1800 rows × 4 columns

In [42]:
```
same_data = (df['Same'] == True)
print('{} / {} data is same'.format(same_data.sum(), len(same_data)))
```

1800 / 1800 data is same

We can see that all the predition results are same as the submitted predictions.

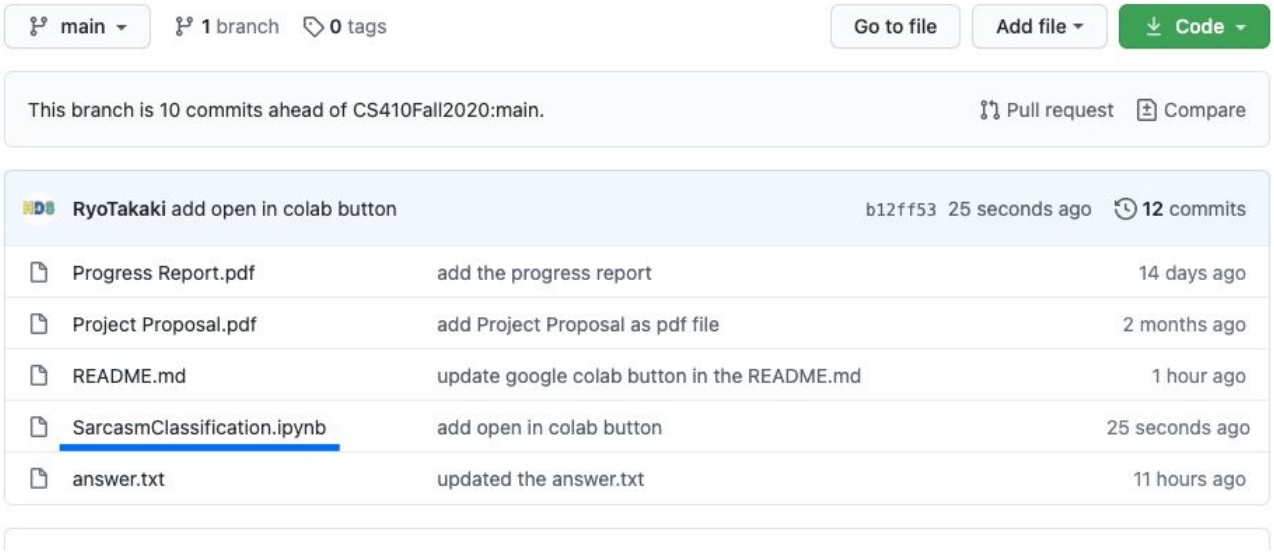
2.11 Reference

The project code was inspired by a UIUC's tech review and a public website below.

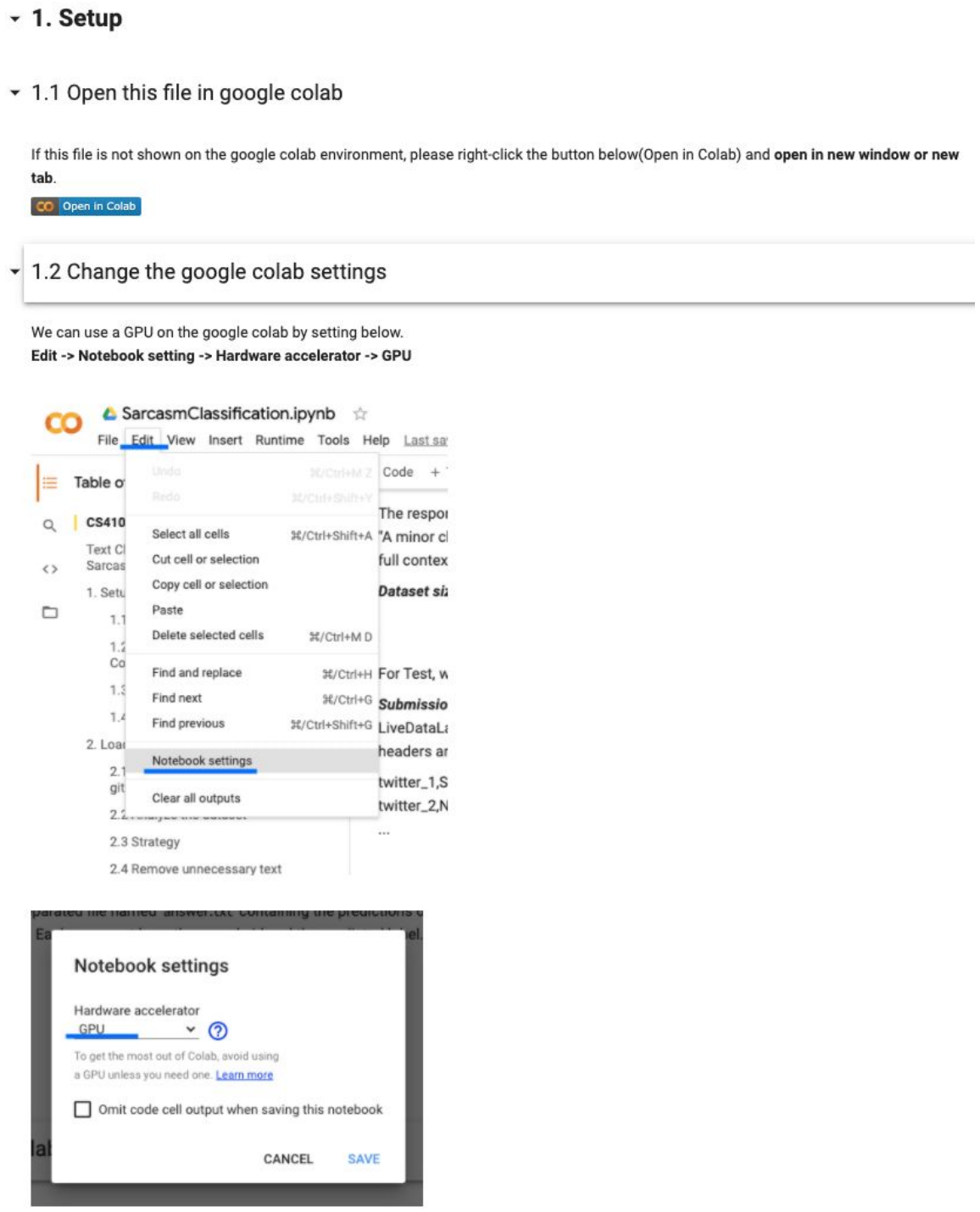
- https://github.com/zen030/tech_review/blob/master/techreview.pdf
- https://mccormickml.com/2019/07/22/BERT-fine-tuning/

3. Usage

3.1 Open "SarcasmClassification.ipynb"

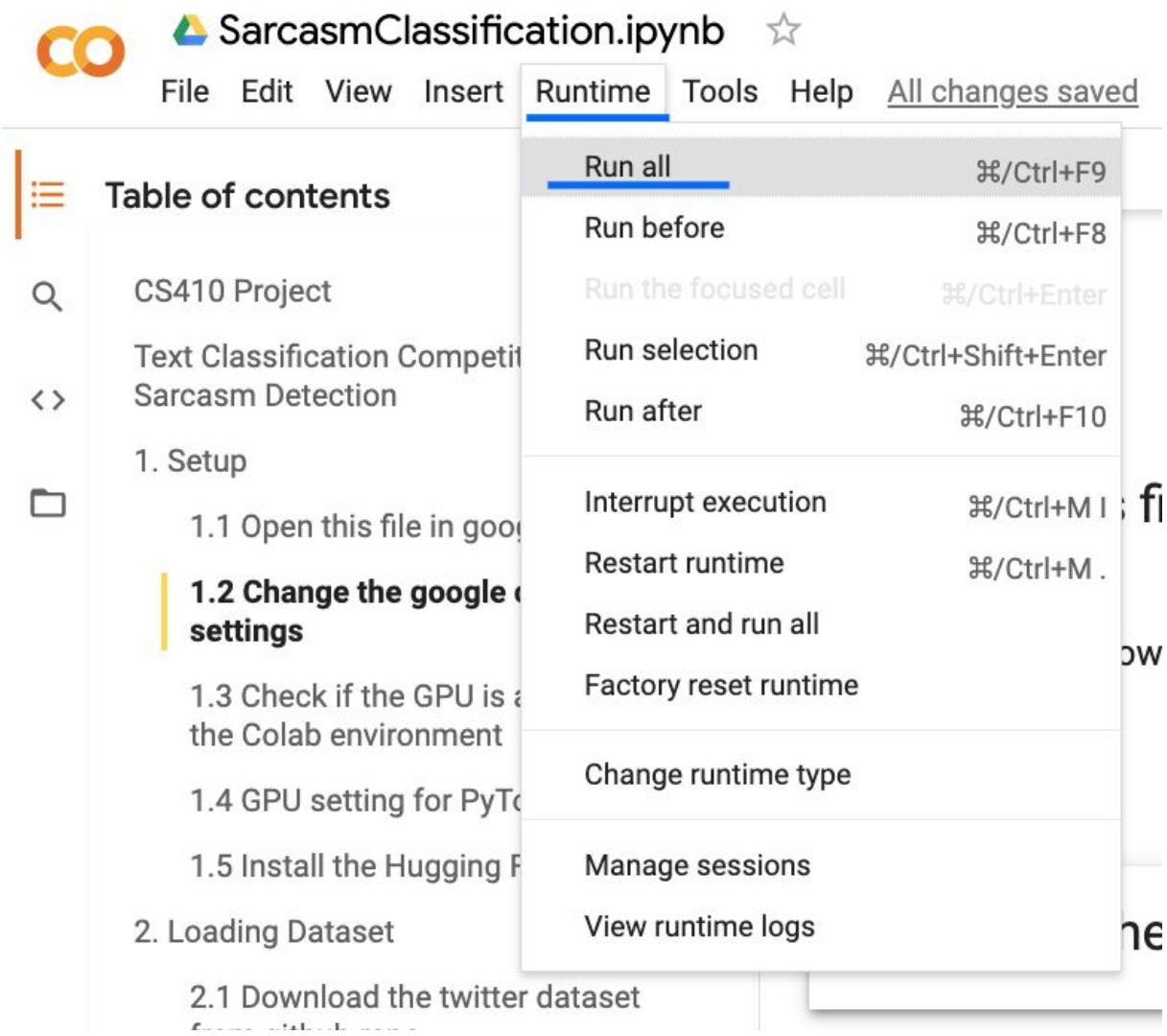


3.2 Follow the setup instruction 1.1 and 1.2 below.



3.3 Run the code

After setting up the google colab GPU setting above, you can run all the code by clicking the “Runtime -> Run all” below, or simply run all the cells one by one.



4. Contribution

As my team member is only me, everything is done by myself.