CS 410: Text Information Systems

Final Project - Final Report
Project Type: Classification Competition
Team: The Classifiers
November 27, 2020

Praveen Pathri (ppathri2@illinois.edu (mailto:ppathri2@illinois.edu))
Steven Piquito (piquito2@illinois.edu (mailto:piquito2@illinois.edu))
Mattias Rumpf (mrumpf2@illinois.eu (mailto:mrumpf2@illinois.eu))

References and web-sites used in this code extensively:

https://towardsdatascience.com/sarcasm-detection-step-towards-sentiment-analysis-84cb013bb6db (https://towardsdatascience.com/sarcasm-detection-step-towards-sentiment-analysis-84cb013bb6db)
https://simpletransformers.ai/docs/usage/ (https://simpletransformers.ai/docs/usage/)

Key Libraries Used in this Project to Date:

As part of our project certain libraries are being used extensively.

For the core traditional ML models as well as feature vector transformations:

SKLearn Library

For the implementation of the RoBERTa pre-trained neural network architecture, the following libaries are used:

HuggingFace Transformer Library

SimpleTransformer.ai Library

Description of Work In This Notebook:

This code notebook explains the next steps we followed post the completion of the project progress report notebook and should be seen (or read) as a follow on to that piece of work. Our total project documentation is covered by both notebooks as well additional descriptions of further testing we performed/optimisation as part of the project.

This notebook focuses on the final implementation of the chosen RoBERTa transformer model implementation designated by the project team for investigation and use in the classification competition and leaderboard. Optimisation and experimentation is described in another document, however this code focuses on the following:

1. The creation and learning of an existing RoBERTa model widely used and available in the simpletransformers.ai library
2. Some feature engineering in the form of stemming, stop word removal and general cleaning of text input data
3. The inclusion of Context in the model to enhance the model performance

Additional commentary and description can be found in the document below code snippets as to what the project team found to be working or not working.

Import key python libraries for this project

The below cell installs all necessary key libraries used in this notebook first should they not already be installed:

In [ ]:

```
!pip install jsonlines
!pip install pandas
!pip install transformers
!pip install nltk
!pip install torch
```

The below cell imports all necessary libraries used in this notebook :

In [1]:

```
import os
import jsonlines
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import simpletransformers
import torch

#Libraries for RoBERTa
from simpletransformers.classification import ClassificationModel,ClassificationArgs
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

Importing of training and test data

The following code reads both the train and test JSON files and imports the data into a python dictionary format:

In [2]:

```
test_file = 'data/test.jsonl'
train_file = 'data/train.jsonl'

data_train = []
iter = 1
with jsonlines.open(train_file) as f:
    for line in f.iter():
        iter +=1
        data_train.append(line)

data_test = []
iter = 1
with jsonlines.open(test_file) as f:
    for line in f.iter():
        iter +=1
        data_test.append(line)

print("Count of training data entries:")
print(len(data_train))
print("Count of test data entries:")
print(len(data_test))
```

```
Count of training data entries:
5000
Count of test data entries:
1800
```

The following code converts the training and test data dictionaries into a Pandas DataFrame format for use later in the RoBERTa model

In [3]:

```
train_data_pd = pd.DataFrame.from_dict(data_train)
test_data_pd = pd.DataFrame.from_dict(data_test)
print("Training and Test Datasets converted to Pandas DataFrames...")
```

```
Training and Test Datasets converted to Pandas DataFrames...
```

Adding Context to training and test datasets as well as basic NLP processing

This code will append the context data to the response/text data linearly so as to add context information to the various model input feature vectors and capture the additional information therein.

Additionally, some basic processing such is performed on the raw data such as stemming, removal of stop words

```
In [4]:
```

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, SpaceTokenizer
from nltk.stem import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to C:\Users\User-
[nltk_data]     PC\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to C:\Users\User-
[nltk_data]     PC\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

The next code snippet performs the addition of context information which you should note **ONLY** includes the addition of the preceding CONTEXT sentence to the response and not the sentence before that as well (i.e. the model uses only one of the two additional CONTEXT sentences):

In [5]:

```
all_stopwords = stopwords.words('english')
tk = SpaceTokenizer()
ps = PorterStemmer()

for i in range(len(train_data_pd)):
    train_data_pd['response'][i]=train_data_pd['response'][i]+train_data_pd['context'][i][1
    train_data_pd['response'][i]=train_data_pd['response'][i].replace('@USER', '').strip().
    text_tokens = tk.tokenize(train_data_pd['response'][i])
    tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]
    test4=""
    for i in tokens_without_sw:
        test4 = test4 + " "+ps.stem(i)
    test4.strip()
    train_data_pd['response'][i]=test4


for i in range(len(test_data_pd)):
    test_data_pd['response'][i]=test_data_pd['response'][i]+test_data_pd['context'][i][1]
    test_data_pd['response'][i]=test_data_pd['response'][i].replace('@USER', '').strip().lo

    text_tokens = tk.tokenize(test_data_pd['response'][i])
    tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]
    test4=""
    for i in tokens_without_sw:
        test4 = test4 + " "+ps.stem(i)
    test4.strip()
    test_data_pd['response'][i]=test4



print("Converted response PD data to include Context Data")
print("Converted response to lowercase and removed stop words as well as @USER")
print("Converted response to stem words using PortStemmer")
```

```
Converted response PD data to include Context Data
Converted response to lowercase and removed stop words as well as @USER
Converted response to stem words using PortStemmer
```

Perform a random test print of the constructed response+context after NLP pre-processing to see if all is ok:

In [6]:

```
print(test_data_pd['response'][10])
```

```
define this way : 1 . desiring the good of the other ; wanting them to thriv
e / flourish , which means they'd get free from the attitudes you mention ;
2 . doing whatever's in your control / power to advance their good ; at leas
t * not * wishing them ill , * not * hating them .  ok , you ’ re right , bu
t how do you love someone who hates you , and wants you to not exist ? how d
o you love someone doesn ’ t share basic morals ?
```

Here we turn the labelled training data text format 'SARCASM' or 'NOT SARCASM' into a binary 1/0 list to for easier input into the models:

In [7]:

```
#Define the vector of actual results:
Actual_Results = []
for  l in data_train:
    if l['label'] == 'SARCASM':
        Actual_Results.append(1)
    else:
        Actual_Results.append(0)
```

Splitting of the project training dataset randomly

The code below splits the training dataset into a random (new) training and test dataset (i.e. only using the course project training data to train and test locally so far):

NOTE THAT RANDOM_STATE is set to zero to provide a fixed seed to the random generator for consistent results

In [8]:

```
#getting training dataset features and labels
features = train_data_pd['response']
labels = train_data_pd['label']
labels = Actual_Results


# Splitting of training data into train and test data
rawdata_train, rawdata_test, rawlabels_train, rawlabels_test = train_test_split(features, l

print("Training dataset split into this many train samples:")
print(len(rawdata_train))
print("Training dataset split into this many test/validation samples:")
print(len(rawdata_test))
```

```
Training dataset split into this many train samples:
4750
Training dataset split into this many test/validation samples:
250
```

Final Model : RoBERTa

The below code implements the SimpleTransformers.ai implementation of RoBERTa and calculates a number of metrics including:

1. Accuracy on both train and test sets
2. The precision, recall and F1 score on the training data
3. The precision, recall and F1 score on the test data

In [9]:

```
#Import the simpletransformers model library
from simpletransformers.classification import ClassificationModel,ClassificationArgs
```

In [10]:

```
#Create a training dataset in a Panda DataFrame format ready for the model
train_df = pd.DataFrame({
    'text': rawdata_train.str.replace('@USER', '', regex=False).str.strip(),
    'labels': rawlabels_train
})
```

In [11]:

```
#Create a test dataset in a Panda DataFrame format ready for the model
test_df = pd.DataFrame({
    'text': rawdata_test.str.replace('@USER', '', regex=False).str.strip(),
    'labels': rawlabels_test
})
```

Set up RoBERTa transformer model

The below code establishes and imports the RoBERTa transformer model. There are two possible options here, one including cuda support (if you have this available) and one excluding cuda support. The difference is primarily the speed of model training/estimation. the code has been set up to use or not use Cuda depending on the machine where the code is being executed (this is done through the torch.cuda.is_available() function which returns true if cuda is an option).

In [12]:

```python
# Create a ClassificationModel
# for training from scratch
import torch

cuda_available = torch.cuda.is_available()
print("Does system have CUDA support?")
print(cuda_available)

model = ClassificationModel('roberta', 'roberta-base', use_cuda=cuda_available) # You can s


# for loading my pre-trained model
#model = ClassificationModel('roberta', 'outputs/checkpoint-594-epoch-1', use_cuda=False)
```

```
Does system have CUDA support?
True

Some weights of the model checkpoint at roberta-base were not used when init
ializing RobertaForSequenceClassification: ['lm_head.bias', 'lm_head.dense.w
eight', 'lm_head.dense.bias', 'lm_head.layer_norm.weight', 'lm_head.layer_no
rm.bias', 'lm_head.decoder.weight']
- This IS expected if you are initializing RobertaForSequenceClassification
from the checkpoint of a model trained on another task or with another archi
tecture (e.g. initializing a BertForSequenceClassification model from a Bert
ForPreTraining model).
- This IS NOT expected if you are initializing RobertaForSequenceClassificat
ion from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a BertForSequenceCl
assification model).
Some weights of RobertaForSequenceClassification were not initialized from t
he model checkpoint at roberta-base and are newly initialized: ['classifier.
dense.weight', 'classifier.dense.bias', 'classifier.out_proj.weight', 'class
ifier.out_proj.bias']
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
```

Perform model training (this may take some time if non CUDA support):

In [14]:

```
model_args = {
    "reprocess_input_data": True,
    "overwrite_output_dir": True,
    "num_train_epochs": 1,
    "model_args.lazy_loading" : True
}

# Train the model
model.train_model(train_df,args=model_args)
```

```
100%                                          4750/4750 [01:03<00:00, 75.37it/s]

Epoch 1 of 1: 100%                                      1/1 [02:16<00:00, 136.05s/it]

Epochs 0/1. Running Loss:                                  594/594 [02:16<00:00,
0.8155: 100%                                               4.37it/s]
```

```
C:\Users\User-PC\anaconda3_SP2\envs\SP 36\lib\site-packages\torch\optim\lr_s
cheduler.py:216: UserWarning: Please also save or load the state of the opti
mizer when saving or loading the scheduler.
  warnings.warn(SAVE_STATE_WARNING, UserWarning)
```

Out[14]:

```
(594, 0.599046219102662)
```

Evaluate the model performance on the training dataset:

In [15]:

```
# Evaluate the model - training
result_train, model_outputs_train, wrong_predictions_train = model.eval_model(train_df,
                                                            f1=f1_score,
                                                            acc=accuracy_score,
                                                            prec= precision_score,
                                                            rec= recall_score)
```

```
100%                                          4750/4750 [01:34<00:00, 50.15it/s]

Running Evaluation:                                     594/594 [00:26<00:00,
100%                                                    22.69it/s]
```

Through printing of the below output of result_train, we can see relevant accuracy, precisionm, recall and F1 scores for the model based on predictions on the training dataset. A resulting f1 score of 0.84 is observed indicating a high degree of accuracy on the training dataset for the model priediction.

In [19]:

```
#Print training result metrics
result_train
```

Out[19]:

```
{'mcc': 0.6703606165142606,
 'tp': 2123,
 'tn': 1833,
 'fp': 538,
 'fn': 256,
 'f1': 0.8424603174603175,
 'acc': 0.8328421052631579,
 'prec': 0.7978203682826005,
 'rec': 0.8923917612442203,
 'eval_loss': 0.3645473464771553}
```

Next we need to evaluate the model performance on the test dataset (the test set randomly sampled from the trainign data...not the course test set for the leaderboard):

In [16]:

```
# Evaluate the model - testing
result_test, model_outputs_test, wrong_predictions_test = model.eval_model(test_df,
                                                            f1=f1_score,
                                                            acc=accuracy_score,
                                                            prec= precision_score,
                                                            rec= recall_score)
```

```
100%                                          250/250 [01:04<00:00, 3.85it/s]


Running Evaluation:                                      32/32 [00:07<00:00,
100%                                                     4.40it/s]
```

From printing the below result_test metrics, we can see that the model appears to perform relatively well with an F1 score 0f 0.81...slightly lower than training set but still relatively good. This is a good result and indicative of a model that hopefully was not overfit to the training data

In [22]:

```
#Print out the model performance metrics on the test set
result_test
```

Out[22]:

```
{'mcc': 0.6282554682487588,
 'tp': 104,
 'tn': 99,
 'fp': 30,
 'fn': 17,
 'f1': 0.8156862745098039,
 'acc': 0.812,
 'prec': 0.7761194029850746,
 'rec': 0.859504132231405,
 'eval_loss': 0.437514793011359 87}
```

Performing predictions on project test data for LiveDataLab

Perform predictions on the project test data for the competition for upload to LiveDataLab for grading

Perform RoBERTa predictions for project test set

In [17]:

```
# getting training dataset features and labels
features_test = test_data_pd['response']

predictions_test, raw_outputs_test = model.predict(features_test)
```

```
100%                                              1800/1800 [00:53<00:00, 33.38it/s]


100%                                              225/225 [00:03<00:00, 63.40it/s]
```

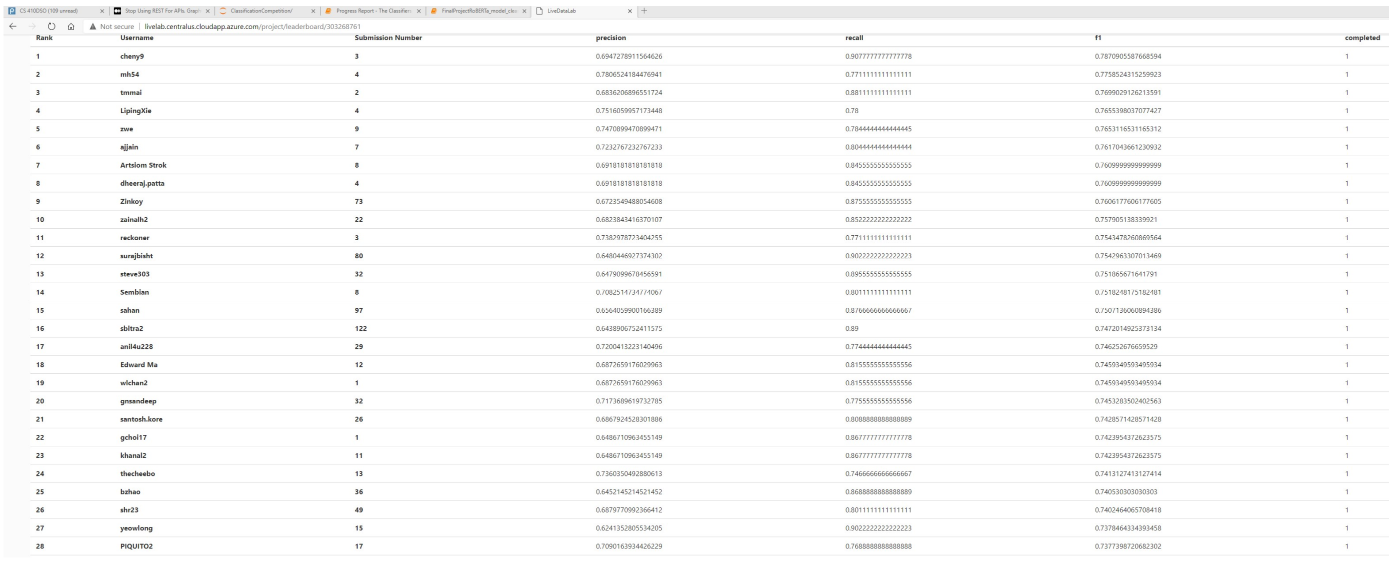
Write the predictions from the RoBERTa model to an output text file called 'RoBERTa_answers.txt' for storage and uploading the the course competition leaderboard:

In [24]:

```
#Writing the RoBERTa Classifier predictions to the output file: RoBERTa_answers.txt
y_pred = predictions_test
f = open("RoBERTa_answers.txt", "w")
for i in range(len(id_final_test)):
    i_result = y_pred[i]
    pred_id = id_final_test[i]
    if i_result == 1:
        f.write(pred_id + ',' + "SARCASM" +"\n")
    else:
        f.write(pred_id + ',' + "NOT_SARCASM" +"\n")
f.close()
```

By taking the resulting output file generated above and submission to the project LiveDataLab scoreboard, a resulting model score of F1 = 0.737 was achieved (beating the baseline score of 0.723).

A sample screenshot of the model is included below and can be seen under the username PIQUITO2 at rank 28



CS 410DSO (109 unread) | Stop Using REST For APIs. Graph | ClassificationCompetition/ | Progress Report - The Classifiers | FinalProjectRoBERTa_model_clea | LiveDataLab

Not secure | livelab.centralus.cloudapp.azure.com/project/leaderboard/303268761

Rank	Username	Submission Number	precision	recall	f1	completed
1	cheny9	3	0.6947278911564626	0.9077777777777778	0.7870905587668594	1
2	mh54	4	0.7806524184476941	0.7711111111111111	0.7758524315259923	1
3	tmmai	2	0.6836206896551724	0.8811111111111111	0.7699029126213591	1
4	LipingXie	4	0.7516059957173448	0.78	0.7655398037077427	1
5	zwe	9	0.7470899470899471	0.7844444444444445	0.7653116531165312	1
6	ajjain	7	0.7232767232767233	0.8044444444444444	0.7617043661230932	1
7	Artsiom Strok	8	0.6918181818181818	0.8455555555555555	0.7609999999999999	1
8	dheeraj.patta	4	0.6918181818181818	0.8455555555555555	0.7609999999999999	1
9	Zinkoy	73	0.6723549488054608	0.8755555555555555	0.7606177606177605	1
10	zainalh2	22	0.6823843416370107	0.8522222222222222	0.757905138339921	1
11	reckoner	3	0.7382978723404255	0.7711111111111111	0.7543478260869564	1
12	surajbisht	80	0.6480446927374302	0.9022222222222223	0.7542963307013469	1
13	steve303	32	0.6479099678456591	0.8955555555555555	0.751865671641791	1
14	Sembian	8	0.7082514734774067	0.8011111111111111	0.7518248175182481	1
15	sahan	97	0.6564059900166389	0.8766666666666667	0.7507136060894386	1
16	sbitra2	122	0.6438906752411575	0.89	0.7472014925373134	1
17	anil4u228	29	0.7200413223140496	0.7744444444444445	0.746252676659529	1
18	Edward Ma	12	0.6872659176029963	0.8155555555555556	0.7459349593495934	1
19	wlchan2	1	0.6872659176029963	0.8155555555555556	0.7459349593495934	1
20	gnsandeep	32	0.7173689619732785	0.7755555555555556	0.7453283502402563	1
21	santosh.kore	26	0.6867924528301886	0.8088888888888889	0.7428571428571428	1
22	gchoi17	1	0.6486710963455149	0.8677777777777778	0.7423954372623575	1
23	khanal2	11	0.6486710963455149	0.8677777777777778	0.7423954372623575	1
24	thecheebo	13	0.7360350492880613	0.7466666666666667	0.7413127413127414	1
25	bzhao	36	0.6452145214521452	0.8688888888888889	0.740530303030303	1
26	shr23	49	0.6879770992366412	0.8011111111111111	0.7402464065708418	1
27	yeowlong	15	0.6241352805534205	0.9022222222222223	0.7378464334393458	1
28	PIQUITO2	17	0.7090163934426229	0.7688888888888888	0.7377398720682302	1

In [ ]: