CS 410 Text Information Systems Course Project Progress Report

**Team Name:**

Virginia DS

**Team Members:**

Huanzhen Hu NetID: [hh21@illinois.edu](mailto:hh21@illinois.edu) Role: Leader

Ying Zhang NetID: [ying12@illinois.edu](mailto:ying12@illinois.edu) Role: Member

**Topic Selected:**

Text Classification Competition

**What Have We Done?**

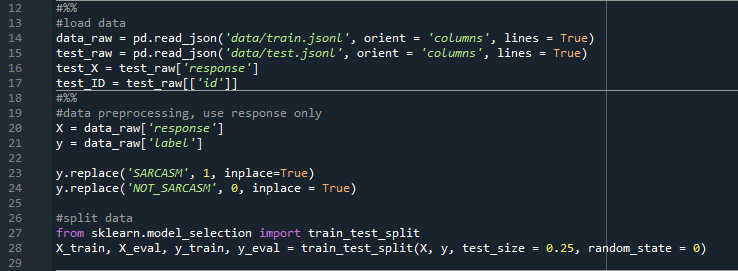
To Complete the classification task, we tried the classic tokenization method and machine learning techniques as described in the proposal. The implementation details are listed below:

**i. Data Preparation**

The provided dataset for training consists of three parts: response, context, and label. Response is the tweet document to be classified; Context is the conversation context of the response; And label contains two categories – “SARCASM” or “NOT\_SARCASM”. Since we haven’t figured out a good method to deal with the context information, in this stage, we only consider the response as input “X” and the label as input “y”. For convenience, we replaced “SARCASM” with integer 1 which indicates positive and replaced “NOT\_SARCASM” with integer 0 which indicates negative.

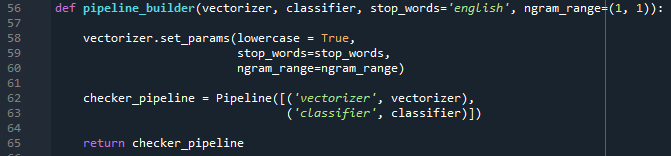
The provided dataset for testing consists of three parts: id, response, and context. To be consistent with the training data, we only utilized the response for classification.

Once the data was well-prepared, we randomly split training data into two parts – train and evaluation with a ratio of 3:1. Please check the code below for more details:

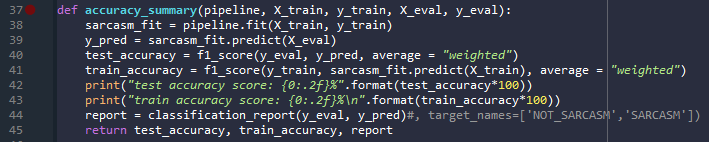


**ii. Model Training and Prediction**

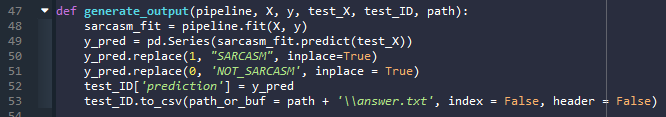
For a more efficient experiment, we built a pipeline to do this process. As shown in the code below, the pipeline is composed of two parts, a vectorizer and a classifier. The function also helped set key parameters of the vectorizer.



Once the pipeline constructed, we can feed training and evaluation data into the pipeline by calling function “accuracy\_summary” to check the accuracy report of the current model. We can also tune the model by recurrent calling this function.

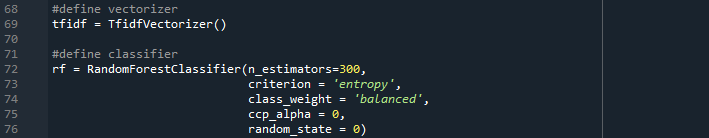


If the model is fine-tuned, we may call the function “generate\_output” to retrain the model with all training data and make the prediction on the testing data. The prediction result will be automatically generated as “answer.txt” which can be feed into LiveDataLab for further testing.



The main process is simple by following the three steps:

First, define tokenizer and classifier:



Second, build the pipeline with defined tokenizer and classifier:



Third, train the model and predict the result by feeding the data into the pipeline:



**iii. Results**

We chose TF-IDF as the vectorizer and we also chose three popular machine learning models as the classifier. The F-1 Score measure was used for evaluation. The results are shown in the figure below:

The results indicate all models overfitted data no matter how we added regularization terms which inferred the tokenized vocabulary matrix should be modified to ensure only key-performance words are counted. Since none of test results surpassed the baseline (0.723), we will no longer focus on traditional machine learning methods and move forward to neural network based models and BERT.

**What are We Doing Now?**

As we discussed in the previous section, we are focusing on neural network based models and BERT now. Since build and adjust such kind of models from scratch is highly time-consuming and unachievable, our work will start from previous research and try to make improvements. Two papers caught our attention. One used mixture neural networks, including CNNs, LSTM and DNNs, to detect sarcasm in Tweet, the other used BERT base model to do same job. Both reached SOTA in their work. We found the source code on GitHub provided by two authors. Now we are trying our best to fully understand the implementation of those codes and to make sure they can be successfully executed.

**What Challenges We Met?**

We found the source code on GitHub provided by two authors. However, those source codes are partially completed and contain bugs. After great works, the neural network can be executed for training but failed to predict on the test sets. For the BERT model, we already fully understood the implementations, but the code cannot be successfully executed. We will keep working on those codes. Once we done, we will try to deploy the model on our dataset and do further adjustments.