2020

CS 410: Project Progress Report

Text Classification Competition

Team: The West Coasters

The West Coasters | 28-Nov-2020

Project Progress Report

<u>CS410: Text Information Systems</u>

<u>Final Project:</u> Text Classification Competition – Sarcasm detection

<u>Date:</u> Saturday, Nov. 29, 2020

Team members:

- **Marina Polupanova** (University of Illinois at Urbana-Champaign) - marinap2@illinois.edu
- **Savita Manghnani** (University of Illinois at Urbana-Champaign) - savitam2@illinois.edu
- **Tirthankar Bhakta** (University of Illinois at Urbana-Champaign) - tbhakta2@illinois.edu

---

<u>Approaches considered for 'Sarcasm Detection'</u>:

We have considered the following ideas to implement text classification in order to detect Sarcasm.

1. **Text classification and sarcasm detection with CNN**:

   We want to try a model, which can be made with parallel convolutional neural networks, where different kernel sizes used in parallel convolutional layers. This gives a multichannel input of text, that in fact uses different n-gram sizes.
   The network type was chosen, as over all the tutorials, it gave one of the best result on the text data, and also, logically, it should grab maximum information having in mind using different n-ngrams.
   The neural network will be built using TensorFlow and Keras libraries.

2. **Text classification and sarcasm detection with BERT**:
   The idea here is to build a classifier with BERT. BERT or Bidirectional Encoder Representations from Transformers is considered as a choice to do text classification as it can do compute vector-space representations of natural language that are suitable for use in deep learning models. The BERT family of models uses the Transformer encoder architecture to process each token of input text in the full context of all tokens before and after.

We plan to use Tensorflow and Keras libraries to create the basic machine learning models and then pre-train the dataset with BERT, fine-tune the model and use it on the test data.

3. **Text classification and sarcasm detection with DNN**:
   Tensorflow library contains multiple estimators which can be used without building the complex models. Here I tried one such pre-made estimator, DNNClassifier, available in tensorflow python library for the classification task. DNNClassifier is an Tensorflow implementation of Deep Neural Network model. It is capable of accepting text in its raw format than required preprocessing to convert in numbers. Paired with Adagard, a gradient based optimization, DNNClassifier provides high recall.

4. **Text classification and sarcasm detection with Bidirectional LSTM**
   Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. The classification task requires the model to learn from various language usages, tones in the sentences, etc. The amount of learning for this supervised task requires, neural network needs to memorize the decisions from short term to long term. LSTM, as the name suggests fits the requirements very well. As there is no pre-estimator or readily available model available for LSTM in Tensorflow, requires creation of neural network model using Tensorflow library.

Implementation & Current Status:

1. **Text classification and sarcasm detection with CNN:**

   The implementation steps for the model include following:
   a) Import required libraries – Tensorflow, Keras, Numpy, Pandas, Json, re, string
   b) Load the 'train.jsonl' and 'test.jsonl' file and read the file
   c) Create the train and validation labels and feature arrays from 'train.jsonl', and test label and feature arrays from 'test.jsonl'.
   d) Convert all the resulting arrays to tensors
   e) Pre-process all the feature train, test, and validation tensors to get a numeric input to Embedding layer and CNN.
   f) Create a model with Embedding and CNN layers
   g) Use model to train the data
   h) To evaluate the model
   i) To test the model on test data, and check the results vs Leaderboard.

   Status: Currently, the steps up to "e" are passed, now we are in the process of model creation and training.

Challenges: There was a challenge to load the data from jsonl files to tensors, and it was multiple time failing as it was the first our experience with Pandas and Tensorflow data structures. After some efforts, the expected input was received. Current challenge is to create the Embedding/CNN of a structure, which will count towards the highest score. Possible issue can be that we have chosen to merge "response" and "context" in one variable, but so far we haven't yet understood how to make it a multichannel tensor.

2. **Text classification and sarcasm detection with BERT:**

The idea is to use Tensorflow and Keras to create a ML model and use BERT pre-training models to perform pre-training on the data, fine tune it, optimize the output and create the final model.
Here are the implementation steps taken into consideration:

j) Import required libraries – Tensorflow, Keras, Numpy, Pandas, Jsonlines, Tensorflow-Hub
k) Load the 'train.jsonl' file and read the file
l) Create the train dataset from 'train.jsonl'.
m) Split the dataset into train and validation datasets
n) Load models from Tensorflow Hub
o) Choose BERT models and determine the best fit to fine-tune
p) Use BERT models to Pre-train the data
q) Build own model by combining BERT with a classifier
r) Train a model, including the preprocessing module, BERT encoder, data, and classifier
s) Use an Optimizer like Adaptive Moments to fine-tune the model
t) Run the test data and analyze data.

Status: Currently, we are in the process of creating the dataset to train the model. The BERT models for pre-training and training are selected and also the model is built.

Challenges: Using Tensorflow-Hub was giving an error – "ImportError: cannot import name 'feature_column_v2' from 'tensorflow-hub". Resolved the error by setting up a new environment and installing Tensorflow ver. 1.14. The current challenge is to create the expected dataset that BERT encoder and classifier can understand.

3. **Text classification and sarcasm detection with DNN**

Current State: DNNClassifier is a pre-made estimator available in tensorflow library. The classifier is capable of supporting multiple classes for categorization. Implementation steps:

a) Import required libraries, as it's a pre-made estimator not many libraries are required for making it work. Libraries it requires are tensorflow, numpy, pandas, sklearn, tensorflow_hub.

b) Load the training dataset and split it into training and validation datasets as Pandas dataframes.
c) Use TFHub sentence encoder to encode the datasets from Pandas dataframes.
d) Created embeddings from above step are fed to DNNClassifier for training and evaluation.
e) Use the trained model, categorize the test data set and generate answer.txt file.

The latest submission with DNNClassifier, LiveLab reports F1 measure of 0.7073684210526316.

Challenges: One of the challenge I faced with the approach was to embed text data to the classifier.
Next steps: Train the model with "Attention" to improve on the performance.

4. **Text classification and sarcasm detection with Bidirectional LSTM**

Current State: Unlike DNNClassifier where one does not have to build a model, LSTM requires creation of a neural network model. To create the model, we use tensorflow keras APIs. The generic APIs accept the data set in numeric format, thus preprocessing of input will not just involve cleaning of data but also converting data from text to numeric representation. Following steps were performed to achieve the task:

a) Import required libraries like tensorflow, numpy, sklearn, TextVectorization, keras layers and losses.
b) Load the training data set and split the data set into training and validation data sets.
c) Preprocess data set to remove unwanted characters and words; convert the data set into numeric format as expected by keras APIs.
d) Build neural network model with different layers including bidirectional LSTM.
e) Train and evaluate the model.
f) When satisfied with evaluation, use the model to predict for test data set.

The latest submission with this model resulted in F1 measure of 0.6917372881355932.

Challenges: Once the DNNClassifier implementation was done, this implementation did not take much time. The only challenge I faced was to vectorize the text in format understood by keras APIs.
Next Steps: Train the model with "Attention" to improve on the performance.