Rumour Detection and Stance Classification

Omkar Gurjar, Dhawal Jain, Jatin Paliwal, Mahathi Vempati

November 11, 2019

1 Introduction

Battling the surge of fake news is a multi-faceted, complicated project, a segment of which is to detect whether rumours that spread on social media are fake or real. One way in which this can be done is to analyse the public reaction to a rumour when broadcasted. The public often have information from a lot of other sources, and their collective reaction brings in valuable information in detecting whether a rumour is real or fake. In order to analyse public reaction, we take tweets and Reddit posts on a topic and study all the replies. We classify the replies into support, deny, query and comment (SDQC) and later use this to verify the original post. We have been given tagged data for training for this task.

2 Description of the Dataset

For training purposes, we have been provided with 9 categories about topics that have generated a lot of conversation in Twitter and Reddit - for example, the Charlie Hebdo shooting and the Sydney Seige. Each category contains around 120-150 tweets and their replies in the Twitter tree structure or Reddit comment tree structure and every reply is tagged with Support, Query, Comment, Deny. The test data also contains rumours that need to be classified as true or false. We have with us the training data from the 2017 and 2019 editions of the competition, and the test data from 2019.

The posts themseleves can be considered as trees, with parent and sibling posts providing context. The tagged data contains whether a given tweet is S, D, Q or C with respect to its immediate ancestor.

- 2019 Train and Test Data
- 2017 Train Data

3 Phase 1: Subtask A

This subtask was completed in the first phase of the project. We write the details here for completeness: The first subtask will deal with tracking how other sources orient to the accuracy of the report in question (that we need to eventually find the veracity of). We are provided with a tree-structured conversation formed of posts replying to the original rumourous post, where each post presents its own type of support with regard to the rumour. We frame this in terms of supporting, denying, querying or commenting on (SDQC) the claim. The goal is to label the type of interaction between a given statement and a reply post. We attempt two distinct methods to tackle this task:

3.1 Context Free Method

Repository: Transformers Repository

First, we attempt to classify tweets into S, D, Q, C without the use of parent or sibling tweets. Note that this attempt is substantiated as a tweet by itself could give reasonable information about its class. A question mark or a word like 'how', etc, immediately hints at a query, while negative connotations could refer to a denial.

For this approach, we use Transformers: an attention mechanism that learns contextual relations between words (or sub-words) in a text as opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, though it would be more accurate to say that its non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

We use SimpleTransformer: This library is based on the Pytorch Transformers library by HuggingFace. Using this library, you can quickly train and evaluate several transformer models. We first preprocess the data and convert the JSON tree structure to Tab separated values fed in a 2D array to the SimpleTransformer module.

The transformer then has two phases:

- The pre-train phase: which has already been done and is available - this is the phase where the transformer learns the structure of a language after being trained on a large corpus (eg: Wikipedia).
- The fine-tune phase: This is separate for every transformer, this is where we feed in our tagged data to the transformer and it learns the S, D, Q, C tags. For each of the 3 transformer models we used, fine tuning took about 5-6 hours.

We attempt three different transformer models here:

3.1.1 Bert

Details 12-layer, 768-hidden, 12-heads, 110M parameters. Trained on cased English text.The pre-training corpus for BERT is BooksCorpus (800M words) and English Wikipedia (2,500M words)

Accuracy: 70.71 %

3.1.2 Roberta

Details 125M parameters RoBERTa using the BERT-base architecture. RoBERTa uses 160 GB of text for pre-training, including 16GB of Books Corpus and English Wikipedia used in BERT. The additional data included CommonCrawl News dataset (63 million articles, 76 GB), Web text corpus (38 GB) and stories from Common Crawl (31 GB).

RoBERTa builds on BERTs language masking strategy, wherein the system learns to predict intentionally hidden sections of text within otherwise unannotated language examples.

RoBERTa, which was implemented in PyTorch, modifies key hyperparameters in BERT, including removing BERTs next-sentence pretraining objective, and training with much larger mini-batches and learning rates. This allows RoBERTa to improve on the masked language modeling objective compared with BERT and leads to better downstream task performance.

Accuracy: 73.74 %

3.1.3 XLNet

Details 12-layer, 768-hidden, 12-heads, 110M parameters. XLNet was trained with over 130 GB of textual data.

XLNet is a generalized autoregressive pretraining method. It is bidirectional and heavily relies on the pretraining phase for language structure compared to the other two models using a method called Permutation Language Modelling.

Accuracy: 77.6 %

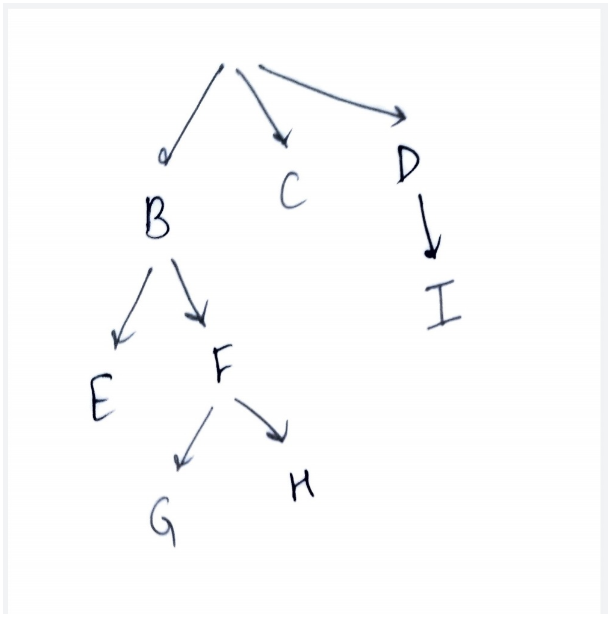
3.2 Contextual Method

Repository: branchLSTM Repository

Given the poor performance of our context-free models (note that a naive classifier that predicted 'Comment' – the majority class – all the time on the test set would have given an accuracy of 75.58 %), we move on to a contextual model.

The structure of the data in our project is in a tree format. Consider the example in the figure below.

Consider three starting tweets: B, C and D. All other tweets are replies to these. Now, for all the tweets in the sub-branches B, C and D, all



the tweets make up the context. For example, for tweet G - tweets B, E, F and H make up the context. However, one way to look at this data, as we need to do stance classification, is to ignore the context due to siblings and look at only parental context.

Therefore, the context for G is B $\rightarrow F \rightarrow G$. The entire tweet tree is converted to such branches. For example, if the tweet tree is as above, then the branches are:

$$B \rightarrow E$$

$$B \rightarrow F \rightarrow G$$

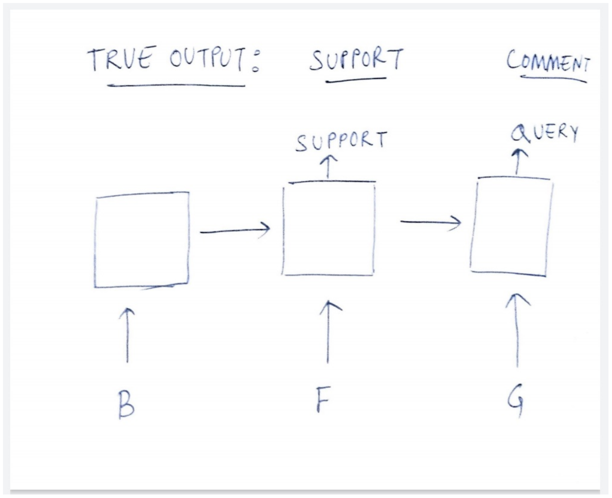
$$B \rightarrow F \rightarrow H$$

$$C$$

$$D \rightarrow I$$

Now, this data is an extremely convenient format to input into a Recurrent Neural Network as every post is tightly correlated to its immediate ancestor, but also needs to learn context from other posts in the branch. Consider the branch $B \rightarrow F \rightarrow G$:

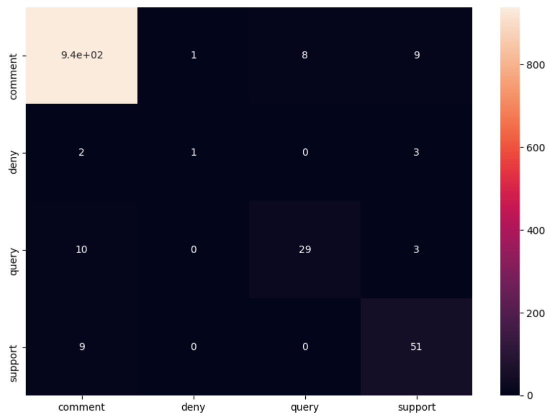
Now, F has a tag on its stance with respect to B, and G has a tag with its stance with respect to F. Say, F is tagged support and G is tagged comment. The branch is given as input in parts



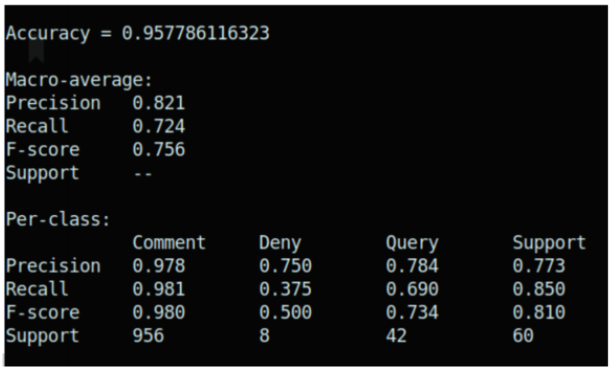
at various timesteps, and the loss function can be calculated based on every output.

Since RNNs are known for forgetting historical context, we can use an LSTM to get past this problem. Hence, if we do not care about sibling context, a branch-LSTM is an ideal neural network to use.

The Lasagne module constructs a suitable LSTM given input hyperparameters. We did not do a hyperparameter search but used the optimal values from the research paper in the repository. Once converted to branch structure, it is straightforward to feed it into the LSTM and train the model. Then, the test data is also converted to branches and tagged. We converted the data to the 2017 contest format, and the links to the data are available in the repository - which is the baseline code modified.



On running the model on the 2019 (Tweets

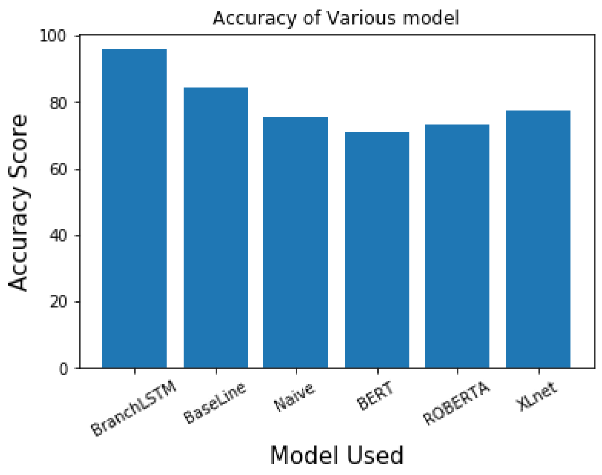

```
Accuracy = 0.957786116323

Macro-average:
Precision   0.821
Recall      0.724
F-score     0.756
Support     --

Per-class:
            Comment     Deny        Query       Support
Precision   0.978       0.750       0.784       0.773
Recall      0.981       0.375       0.690       0.850
F-score     0.980       0.500       0.734       0.810
Support     956         8           42          60
```

only) data, the results are as follows:

Accuracy: 95.77 %

F-score: 0.756

3.3 Model Comparison



This heavily beats the 84% accuracy of the winning model. While the two cannot be compared because we have only trained and tested on Twitter data as opposed to both Twitter and Reddit done by the winning submission, we are yet to study the reason for this accuracy.

4 Subtask B

The goal of the second subtask is to predict the veracity of a given rumour. The rumour is presented as a post reporting or querying a claim but deemed unsubstantiated at the time of release. Note that for each of these posts we also have the public opinion we have analysed in Task A. Given such a claim, the system should return a label describing the veracity of the rumour as true or false along with a confidence score.

Based on the rumour, we have to divide it into three classes - True (Class 0), False (Class 1) and Unverified (Class 2). Unlike the previous data, which had sequential dependence, the data here is in the form of a bag of words representing the tweet or the Reddit post itself and three more features comprising the support ratio, deny ratio and comment ratio from Task A (The query ratio is redundant.) We take two different approaches to do this classification. First, we use all available features, and in the second approach we completely discard the features describing the post and use only public reaction. For each approach, we use different types of classifiers.

4.1 Post Context and Public Reaction

The results when we use all features for various classifiers are as follows:

4.1.1 Linear SVC

4.1.2 MLP Classifier

4.1.3 Logistic Regression

4.1.4 Decision Tree

4.1.5 Random Forest Classifier

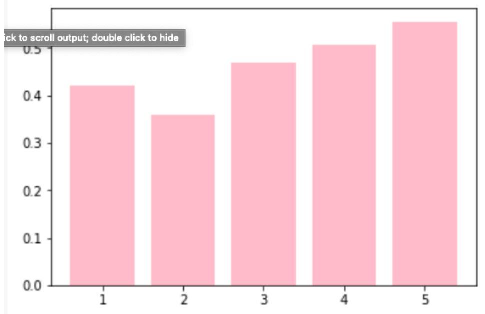
4.1.6 Model comparison

4.2 Only Using Public Reaction!

The results using only public reaction - The support, deny and comment percentages - just three features as opposed to 1750 in the first case to a post are as follows:

```
print("with tweet context")
plt.bar([1,2,3,4,5], X_axis, color = 'pink')
plt.show()
print("Graph Plotting the variation of accuracy for different models ")
```

```
with tweet context
```



```
Graph Plotting the variation of accuracy for different models
```

4.2.1 Linear SVC

4.2.2 MLP Classifier

4.2.3 Logistic Regression

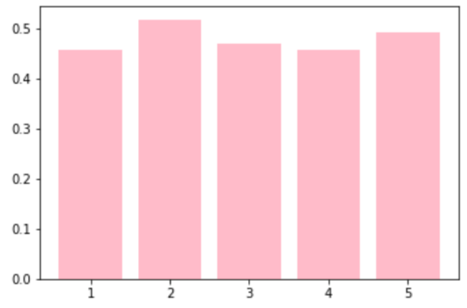
4.2.4 Decision Tree

4.2.5 Random Forest Classifier

4.2.6 Model comparison

```
print("without tweet context")
plt.bar([1,2,3,4,5], X_axis, color = 'pink')
plt.show()
print("Graph Plotting the variation of accuracy for different models ")
```

```
without tweet context
```



```
Graph Plotting the variation of accuracy for different models
```

The best accuracy using all the post features was , whereas the best accuracy using just public reaction is . While we expected public reaction to definitely improve the results, we didn't expect the post features themselves to be a hindrance. This beats the Baseline model by . The confidence scores for our best classifier - linear SVC are as follows: