Fake News Detection

Piyush Dugar

2/8/2018

Definition

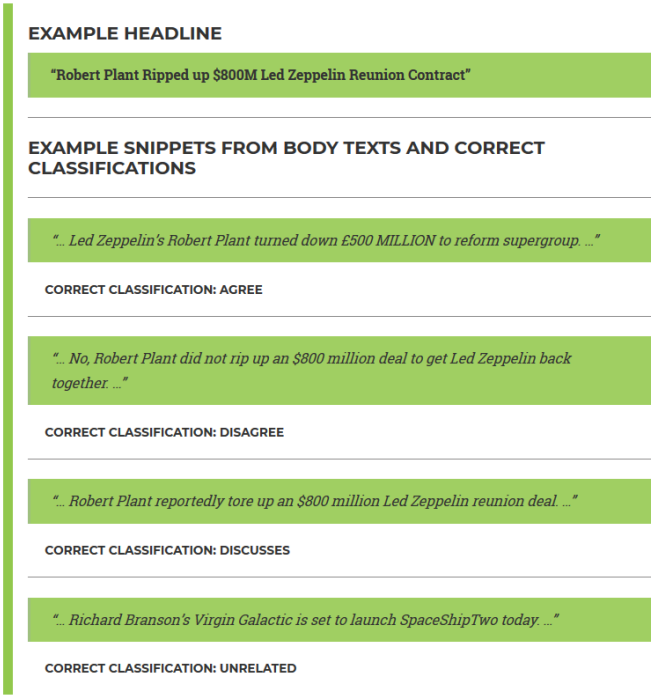
Project Overview

With the expansion of social media platforms, journalistic websites/organizations there has been an enormous increase in the number of available outlets for the society to grab the current affairs from. The major intention behind this is to produce false news that are eye catching to increase the number of users to a website or to produce news that misleads and manipulates people for purchasing/believing something that isn't real. A detector with good performance can help to decrease such instances and can be used at rating various websites/organizations for being trustworthy. Fake news challenge addresses this issues by making a first step through stance detection between headline and article.

In this project, I created a model that finds the similarity between the Headline and the article Body and classifies it into 4 classes: *unrelated, discuss, agree and disagree.* The model uses Bi-directional LSTM for summarizing the sequence and CNN for classifying the summarized sequence.

Problem Statement

To state the problem formally, the model is given a headline and an article corresponding to it. The model will find the probability of the pair belonging to each of the Stance class (*unrelated, discuss, agree and disagree*).



Metrics

I have used the following metrics

1. Mean accuracy
2. Mean Fscore
3. Inverse Weighted F-score: Sum of f-score of each class weighted by inverse of its sample representation. So majority class will have the lowest weight.
4. Mean Precision
5. Mean Recall

Since nearly 73% of the data is the *unrelated* (the majority class), even if we predict everything as *unrelated* we still can get 73% accuracy. Because of this imbalances in the data, we will focus more on the mean fscore, inverse weighted fscore, precision and recall rather than only focusing on Accuracy.

Analysis

Data Exploration

The dataset for this problem is collected from the fakenewschallenge.org . The dataset contains the headline, article pair and their corresponding **stance**: *unrelated, discuss, agree, disagree.* There are a total of 49972 training samples. The training data is divided into two files:

1. Train_stances.csv : It contains the headline, bodyID and stance
2. Train_body.csv : It contains the bodyId and the Body text

Testing data has 25413 unlabeled samples.

Following is one of the row in Training_stance.csv:

“””

```
Police find mass graves with at least '15 bodies' near Mexico town
where 43 students disappeared after police clash,712,unrelated
```

“””

The first field is the Headline, second is the BodyId and the third field is the stance.
Following is one of the row in Training_body.csv

“””

```
0,"A small meteorite crashed into a wooded area in Nicaragua's capital
of Managua overnight, the government said Sunday. Residents reported
```

hearing a mysterious boom that left a 16-foot deep crater near the city's airport, the Associated Press reports.

“”””

The first field is the BodyID, second is the body text
So, the first thing which is supposed to be done is to join the two files on BodyID.

The distribution of the data among the 4 classes is as follows:

We can clearly see that the data is imbalanced

	Unrelated	Discuss	Agree	Disagree
Percentage of training data	73.13	17.82	7.36	1.68

Following are the statistics for the number of words (including punctuations) in Headline and Article

1. The average Headline length is 11.12 and average article length is 369.69
2. The max Headline length is 40 and max article length is 4788
3. The min Headline length is 2 and min article length is 4
4. The median Headline length is 10 and median article length is 304

Exploratory Visualization

To find the most important words that occur in each Headline-Article pair corresponding to each class, I concatenated the Headline and Article and made the word-cloud for each of the class

For the unrelated class

For the discuss class

For the agree class

For the disagree class

There isn't much difference between the words used in different classes. On a whole, they are pretty much the same which is evident from the Word Cloud. This is the reason why using bag of words is a bad strategy for Fake news detection. More important is to understand and summarize the headline-article pair.

Algorithms and technique

Words cannot be used directly for training. They must be converted in a form that is understandable by the machine learning algorithm. Also, many times in articles and headlines, people use synonyms and related words, so the algorithm should be able to identify those similarities and differences. Therefore, I represented words in the form of word vector.
Word Vectors are vector representation of the words and they are obtained by training neural networks on large corpus of global words. By converting them to the word vectors, similarities between the words can be seen. For example,

wordVec('France') – wordVec('Paris') + wordVec('India') ≈ wordVec('NewDelhi').

Here, we can see that it follows logic and is able to find similarities and differences in the word representation. Another example,

wordVec('king') – wordVec('man') ≈ wordVec('queen').

After converting to the word vectors, I used the Bi-directional LSTM, which is the state of art model for sequence labelling problem. But it gets swept away by the majority class if the inherent data distribution is not balanced. So, instead of using Bi- LSTM as a classifier, I'm using it as a sequence summarization model which will summarize the headline-article pair in the form of a sequence that is a bit easier to classify. I then used CNN for the summarized sequence classification. CNN are really good at classification tasks and can brilliantly handle imbalances in

the data. But, both the models need large amount of training data as compared to traditional models like SVM, Logistics regression etc... Fortunately, we have around 49000 training samples, so this shouldn't be a proble. The final model will give a classification for each sample.

The following parameters are tuned to optimize the classifiers:

1. Number of Bi-LSTM units
2. Kernel size for CNN
3. Batch size
4. Optimizer for training

I used Nvidia 1050ti for faster computation. Although being not that powerful, I was still able to get good results in appreciable time.

During training, both the training and validation sets are loaded into RAM. After that, random batches are selected to be loaded into GPU for memory. The training is done in Batch-gradient descent mode .

Benchmark

Fakenewschallenge.org provides a benchmark model which was trained on hand coded features using Gradient boosting classifier. It achieves an accuracy score of 79.53 % on the test data. The goal is to achieve an accuracy above the baseline score. Their complete score can be found here: https://github.com/FakeNewsChallenge .

Methodology

Data preprocessing

The preprocessing done to prepare the data consists of the following steps

1. Read the Training_stance.csv file
2. Read the Training_body.csv file
3. Join the two lists based on the Body Id field

After we have joined the two files, we have three fields in each row . Following are some of the rows after the join

	articleBody	Headline	Stance
21792	When Apple unveiled its first smartwatch earli...	Rare Case: Seven Bosnian Girls, Aged 13 And 14...	unrelated
26245	Dylan Thomas found the spider had burrowed its...	Homeless man receives $100... then shocks ever...	unrelated
35214	Ryan Roche, 32, of Levi, was coaxed into parti...	Chugging eggnog at office party lands man in h...	agree
11975	DNA tests have confirmed that a daughter and a...	Amazon.com to open first physical store in Man...	unrelated
36896	BREAKING: Islamic State, in video, beheads Ame...	Fears journalist James Wright Foley beheaded b...	discuss
30180	(CNN) -- Boko Haram laughed off Nigeria's anno...	Reports: Jihadists Steal Commercial Jets, Rais...	unrelated
12147	Forget sweater weather and crisp autumn leaves...	CNN Plays Alleged Audiotape of Michael Brown S...	unrelated

The next task is making the word vector. I used the Global Word Vector of dimension 50 provided by Stanford University [link]. The word vectors are provided in the form of a txt file. I made a dictionary for the word vectors with key as words, punctuations, tabs, spaces and value as their corresponding word vector.

These are the steps I followed to convert the headline and article to their corresponding wordVector form

1. For each character in the text, check if it is Alphanumeric or is a space. If yes, then append it to a list. If not, then append a space. At the end we will have a cleaned string of the text. This is done for all the headline and Article text.
2. Split the headline and Article on spaces
3. Sum the word vectors of all the headline words (in case they are present in the wordVec dictionary) , let's call it as titleVec
4. Sum the word vectors of all the Article words (in case they are present in the wordVector dictionary) , let's call it as articleVec
5. Make another vector, diffVec as bodyVec – titleVec
6. Normalize the diffVec and it becomes the sample representation

The reason why I summed up all the all the wordVectors and not simply append them is because the dimensions were becoming very large and it needed much powerful GPU for the computation. Also, article length ranges from 4 to 4788 words. Definitely some limit had to be specified to constraint the number of wordsVectors to be appended. This would then result in loss of information. So, I decided to sum them all. Definitely, this would result in loss of information about the word ordering in the headline and in the text. But considering the computational cost, I think this is a valid constraint.

Implementation

I have used two models for the final classification: Bi-directional LSTM for headline-article pair text summarization and CNN for classification.

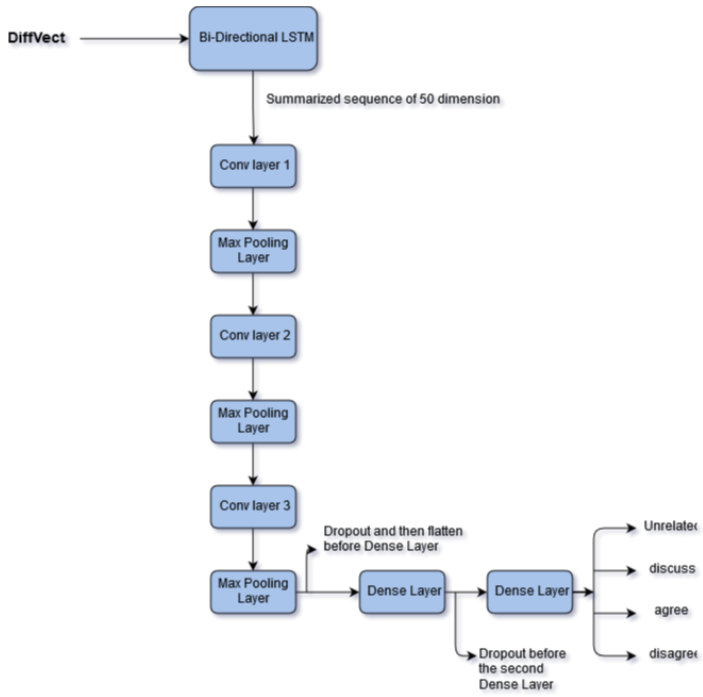
Bi-directional LSTM are great at finding long range dependencies. Bi-LSTM are enhanced version of the LSTM (Long Short-Term Memory) in which an LSTM is trained in both forward and backward pass sequentially. LSTM is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for remembering values over arbitrary time intervals and therefore the word "memory". Each gate can themselves be thought of a neural network . LSTM are very well known for its sequence summarization, i.e given a sequence , summarize it in a form that is best suitable for classification. Here, the benefit of using a Bi-lstm is that the model is trained on diffVec on both the directions, which is a sure shot benefit to summarize it. The bi-lstm returns a sequence of words of 50 dimensions each. The number of these word are equal to 2*number_of_bilstm_units .

CNN are great at classifying data. The input to the CNN are vectors of dimensions 50*(num of Bi-lstm units). Let's say that there are 64 bi-lstm units, then the input to the CNN would be a final vector of dimension 50*128 . This can be thought as an image and CNN are the best for classifying images.

After a series of CNN and Max pooling layers, the model outputs the classification probability for each class

Following flowchart depicts the overall model

The overall Architecture for classification



The above figure shows the overall model in a nutshell. The acronyms can be read as, Conv: Convolutional Layer, Dense: Fully Connected Layer

Also, since the dataset is imbalanced, I have specified a class weight which is inversely proportional to its percentage in the training data. The weights updates according to those proportion so that the model heavily penalizes if it wrongly classifies disagree but there's a small penalty when it wrongly classifies the unrelated class. I selected **Adam** as the optimizer as it is one of the most popular optimizer available and has proven to give good results. Also, it has 2nd order momentum and not just 1st order which again makes it one of the best optimizer for our case. Apart from that, there are certain parameters that are to be optimized

1. Number of Bi-directional LSTM units
2. Dimension of Kernels in each of the Convolutional layer
3. Batch Size for training the model

Refinement

I did a grid search on the hyperparameters to get the best possible models. Parameters for grid search were:

1. Number of Bi-LSTM: [32,64]
2. Dimension of kernel: [3,5,7]
3. Batch sizes: 256, 128, 64

The training set was split into training and validation with a 20% split.

I had the following callbacks for the model:

1. Early stop: Stop training if the *validation loss* doesn't decrease for 10 consecutive iterations
2. Checkpoint: Save the weights every time we get better results.

Some of the Adam's default parameters:
Adam uses a default learning rate of 0.001, beta_1(first order momentum) = 0.9 and beta_2 (second order momentum) = 0.999 as a starting point and then dynamically adjusts it. Also, I left the decay parameters as default = 0, because initially the model will learn the majority class and we don't want a decayed learning when it is learning to classify other classes.

My grid search optimization consisted of two steps

1. Find the best model among the following with a batch size of 100
    a. BI-lstm units = 32, kernel size = 3
    b. BI-lstm units = 32, kernel size = 5
    c. BI-lstm units = 32, kernel size = 7
    d. BI-lstm units = 64, kernel size = 3

e. BI-lstm units = 64, kernel size = 5
f. BI-lstm units = 64, kernel size = 7

2. For the best model from above, find the best batch size from 64,128 and 256

Results

Model Evaluation and Results

The training set was split into training and validation set. Grid search was performed and following were found to be the best optimized parameters:

1. Number of BI-LSTM units = 32
2. Size of kernel in CNN = 5
3. Batch size = 128

Models were trained for 100 epochs.

Following are the results on the Validation/ Test data

Model Architecture	Mean accuracy	Mean Precision	Mean Recall	Mean F-score	Inv - Wt F-Score
BI-Lstm:32, Kernels:3	0.8616	0.6878	0.5206	0.5505	0.3364
BI-Lstm:32, Kernels:5	0.8788	0.7110	0.5857	0.6149	0.4552
BI-Lstm:32, Kernels:7	0.8760	0.7750	0.5458	0.5616	0.2825
BI-Lstm:64, Kernels:3	0.8739	0.6988	0.5786	0.6076	0.4570
BI-Lstm:64, Kernels:5	0.8754	0.7461	0.5676	0.5909	0.3777
BI-Lstm:64, Kernels:7	0.8740	0.6652	0.5741	0.5899	0.3718

Table showing the grid search scores for different models. In this grid search we are optimizing for the Bi-Lstm units and the kernel dimension. The batch size is kept constant at 100. Acronyms are: Inv - Wt means Weighted fscore where the weights are inversely proportional to the sample representation

We will select the model which has the maximum Mean fscore and the Maximum Inverse weighted fscore. The reason being that the accuracy can be swayed by the majority class. It is evident that the two models: (BI-Lstm:32, Kernels:5) and (BI-Lstm:64, Kernels:3) performs the best and have scores nearly equal. **So, we will select the model (BI-Lstm:32, Kernels:5) as the best model as it is less complex that the other one and hence will suffer less from error due to variance**.

To find the batch size, I ran another grid search on the model (BI-Lstm:32, Kernels:5) with different batch sizes.

Following are the results on the Validation/ Test data

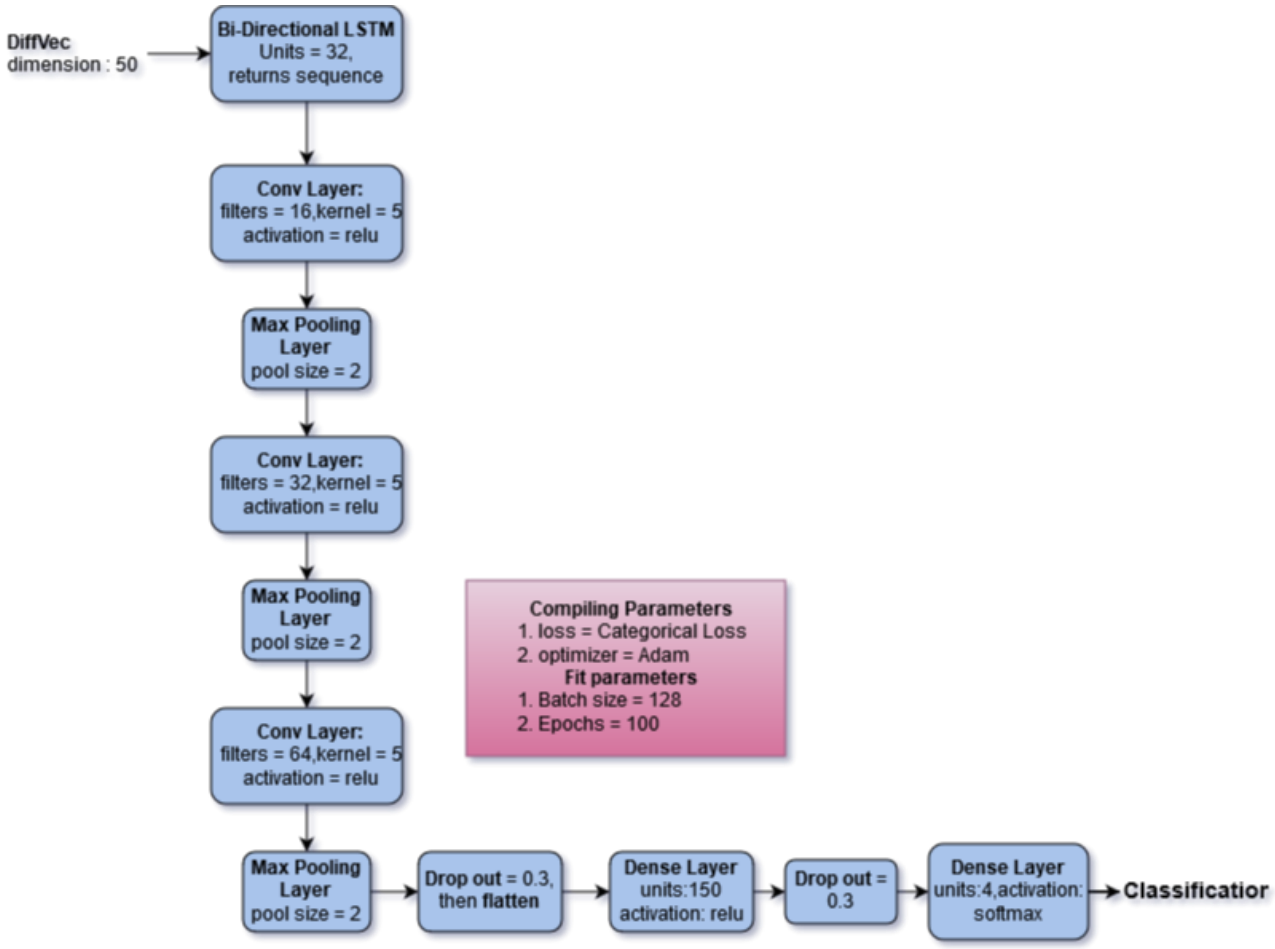
Model Architecture	Mean accuracy	Mean Precision	Mean Recall	Mean F-score	Inv - Wt F-Score
Batch size: 64	0.8727	0.6952	0.5756	0.6026	0.4306
Batch size: 128	0.8788	0.7110	0.5857	0.6149	0.4552
Batch size: 256	0.8640	0.6402	0.5373	0.5415	0.2141

Scores for different batch sizes for the model with 32 Bidirectional Lstm units and kernel dimension as 5

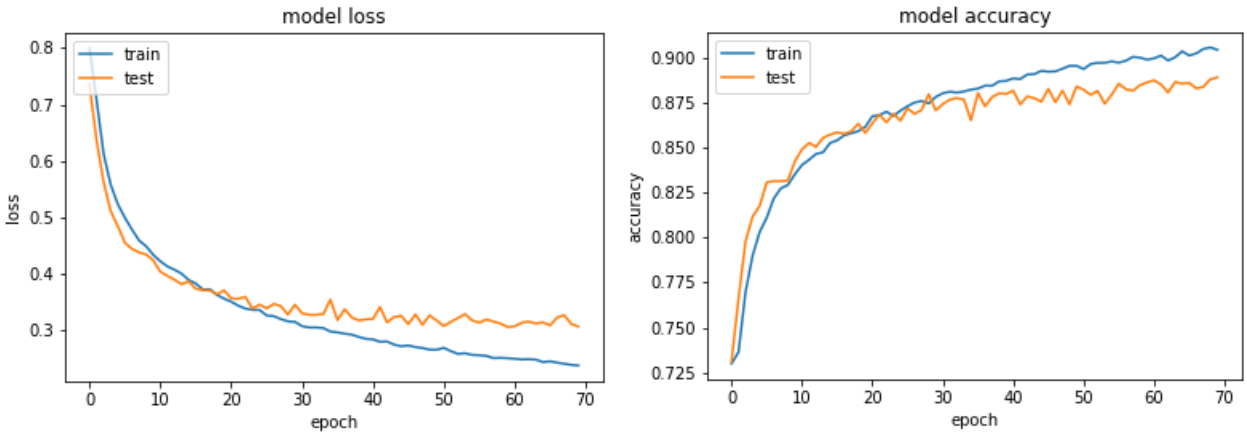
So, after the exhaustive grid search, the final optimized parameters are

1. **Bidirectional LSTM units: 32**
2. **Dimension of Kernel: 5**
3. **Batch size for training: 128**

The Final architecture with optimized parameters



Following are the Model loss and Model accuracy for the Training and Validation test cases with respect to epochs



As we can see, that the model starts to converge roughly after 30 to 40 epochs. The dropouts and the regularization prevented the overfitting and didn't let the accuracy decrease. So, we can be sure that this is the best model given the sample representation as *diffVec.*

Robustness

To find the robustness of the model, I perturbed the Validation data by random noise.

To achieve this let's define few terms:

1. Perturbation factor: The random noise will have values randomly between -(Perturbation factor)*(Max value of input data) to + ((Perturbation factor)*(Max value of input data)
2. Since the maximum Value of the input data is +1 and minimum value is -1, the random noise will have values between (-1*Pertub_factor,+1* Pertub_factor)

Perturbation factor I considered are: **0.01,0.001,0.0001**. Perturbation factor of 0.01 means that the noise maximum value is 1 % of the input data's maximum value and it is a lot! It's very high, but the model should still be able to make some basic predictions.

Perturbation Factor	**Mean accuracy**	**Mean Precision**	**Mean Recall**	**Mean F-score**	**Inv - Wt F-Score**
0.0001	0.8787	0.7109	0.5855	0.6148	0.4552
0.001	0.8784	0.7134	0.5866	0.6166	0.4626
0.01	0.8749	0.7062	0.5750	0.6048	0.4313

It is evident that even after a Perturbation factor of 0.01, the fscore just dropped by 0.01. This shows that the model is very stable and generalizes well. The algorithm focuses on the series of words and tries to summarize the article headline pair in a representation that classifies them well.

Justification

To verify how successful the algorithm is, I used the challenge's model, which they have provided on GitHub . They have made some hand coded features and trained a Gradient boosted classifier.

Using their baseline model, following are the results

Model Architecture	Mean accuracy	Mean Precision	Mean Recall	Mean F-score	Inv - Wt F-Score
Baseline Model	0.8773	0.6180	0.5016	0.4986	0.1850
Best Trained Model (Bi-LSTM + CNN)	0.8788	0.7110	0.5857	0.6149	0.4552

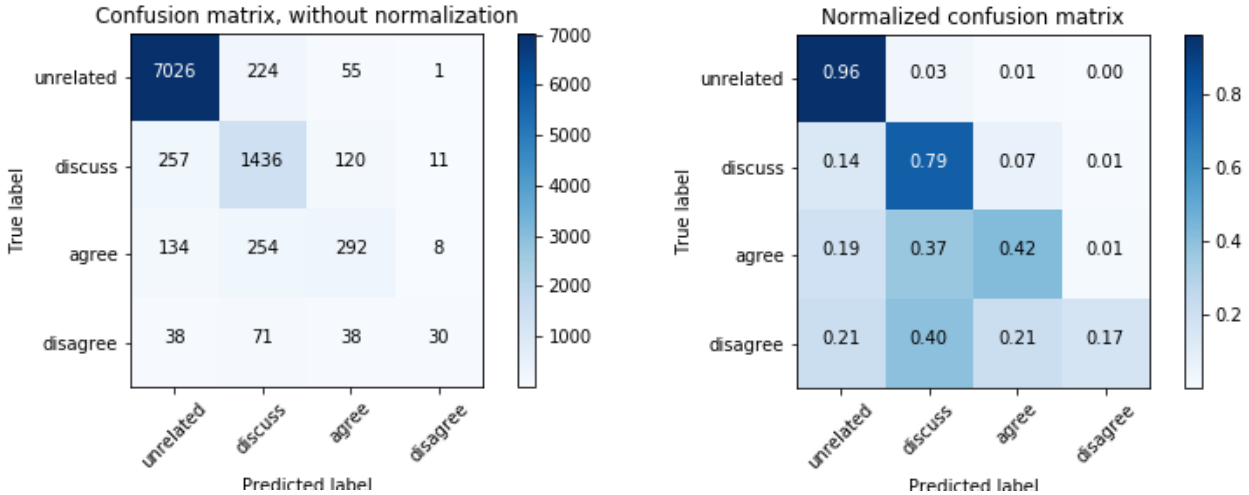
Following are the observations:

1. The baseline model has f-score and inverse weighted f-score much lower than our best trained model.
2. The accuracy of the Baseline Model is approximately same as our Best trained model. This explains why I emphasized on the need of not using Accuracy as a metric. Their Model is getting swept by the Majority class and making majority generalized model
3. It is evident from the Inverse Weighted score that our model performs much better than their model on minority classes

The trained model is performing much better when compared to the baseline. But, to solve the bigger problem and to actually put it into use, we need a much more complex representation of the input data (not summing up the word Vectors, using them as it is, which would result in each input begin a series of n-dimensions where n being the dimension of the word vectors used). To achieve this, a more powerful computing capacity and GPU is required (see the improvement section)

Conclusion

Free form visualization



One thing to note here is that if something is classified as fake/disagree, we should have high confidence for it. We just can't say that an article-headline is fake with some probability. This means that we should have a high precision in classifying something as disagree. If this is not the case, and an article-headline pair that is not fake and the model classifies it as Fake, then the model would be criticized. We can accept classifying some pair as not Fake when in case they are but cannot accept the opposite at any cost.

From the confusion plot, we can see that 20 out of 9818 pair were classified as disagree when actually they were not. This is 0.2%, which means we have 98.0% confidence in classifying something as disagree/fake.

We also got a good mean f-score of 0.6149 which means that the model is not just predicting everything as *unrelated* (majority class).

Reflection

The process used for this project can be summarized in the following steps:

1. An initial problem and dataset are found
2. The data was downloaded and processed (converting to word-vectors)
3. A benchmark was provided, and aim was to do much better than the baseline model
4. An initial architecture was decided for the model with proper reasoning

5. The classifier was trained using the data (multiple times, until a good set of features were found)
6. The scores of the final optimized model were compared against the baseline scores

I found steps 2,4 most difficult. I had to learn how to represent words, how to come up with an initial architecture and to decide what all parameters to optimize. As for the most interesting aspects of the project, I got learn about sequential models, word Vectors and CNN.

Improvement

1. **Data Pre-processing**: The highly imbalanced data must be considered before the classification task. To counter class imbalance, we could consider using down-sampling majority class or oversampling minority class.
2. **Using an Attention model**: Training with attention has shown to improve performance on numerous tasks where present input has dependents on particular processed sections. It could be a great experiment to explore this idea in this scenario.
3. **Considering article-headline pair in series of Word Vectors**: Instead of summing up all the word vectors, we can break them into batches of let's say n. We can sum the word vectors in individual batches. By this the input itself would be a vector of dimension nXm where n is the number of batches and m is the dimension of word vector used.
4. **Additional syntactic features**: It might be worthwhile to explore adding additional syntactic features such as structural dependencies or POS tags to see if this can boost the minority class performance.
5. **Using two classifiers**: Using the first classifier to classify into related or unrelated and second one to classify into discuss, agree, disagree. This way we can get rid of the models getting swiped away by the unrelated class

References

1. William Ferreira and Andreas Vlachos. Emergent: a novel data-set for stance classification. In HLT-NAACL, 2016.
2. Isabelle Augenstein, Tim Rocktaschel, Andreas Vlachos, and Kalina Bontcheva. Stance Detection with Bidirectional Conditional Encoding. Empirical Methods in Natural Language Processing, (2010):876–885,2016.
3. Bowman, Samuel R., et al. ”A large annotated corpus for learning natural language inference.” arXiv preprint arXiv:1508.05326, 2015.