

AUTOMATIC STANCE DETECTION

By: Abhinav Kumar Jha



Problem Statement

The problem is about “stance detection,” which involves comparing a headline with a body of text from a news article to determine what relationship (if any) exists between the two. There are 4 possible classifications:

- The article text agrees with the headline.
- The article text disagrees with the headline.
- The article text is a discussion of the headline, without taking a position on it.
- The article text is unrelated to the headline (i.e. it doesn't address the same topic).

Dataset Overview

The datasets for our task are provided by the Fake News Challenge organization. The complete training set consists of 50,000 “stance” tuples, with each tuple consisting of:

- A headline (word count 2-40)
- The (integer) ID of an article.
- Article Body. Length range from 2 to nearly 5000 words.
- The true stance of the headline with respect to the article. (This is one of the four classes outlined earlier: agree, disagree, discuss, and unrelated.)

<https://github.com/FakeNewsChallenge/fnc-1>

Features Extracted from text

1. Cosine Similarity
2. KL Divergence Score
3. Google News Cue Words Embedding
4. Google Word 2 Vec word embeddings
5. N Gram Model
 - 5.1 Unigram
 - 5.2 Bigram
 - 5.3 Trigram
6. Word Weightage scorers
 - 6.1 TF -IDF
 - 6.2 *Skip Gram Model*
 - 6.3 *Common Bag of Words*

Vector Representation

- Initially, each document (headlines and article bodies) are vectorised into tf-idf form.
- Before this is possible, we need to learn the idf weights of every word in the collection. This is done by going through all documents and building a dictionary of (word -> in how many documents does that word appears). This is the document frequency (df) score of each word.
- To get the idf score of each word, its df score is inverted and smoothed with the following function :

$$idf_{word} = \ln \left(\frac{1 + N \text{ (no. of docs in collection)}}{1 + df_{word} \text{ (no. of docs in which word occurs)}} \right) + 1$$



Computing tf-idf -:

- Then, we can compute the tf-idf values of each word in a document.
- First, the term frequency of the word is counted (e.g. if the word 'glove' appears twice in a document d, the tf of glove is 2), then this tf value is multiplied by the idf weight of said word (e.g. if the idf of 'glove' is 1.25, then its tf-idf value in d is $2 \times 1.25 = 2.5$).
- This tf-idf representation of each document is saved as a (word->tf-idf) value dictionary (e.g. 'glove'->2.5).

$$tfidf_{word,doc} = tf_{word,doc} (\text{no. of times word occurs in doc}) \times idf_{word}$$



GloVe: Global Vectors for Word Representation -:

- Then, to make these tf-idf representations comparable, we used GLoVe pre-trained word vectors (<https://nlp.stanford.edu/projects/glove/>) to convert these tf-idf representations to fixed-length vectors based on the learned 'meaning' of each word.
- GLoVe provides a mapping of six billion English words to a 50-dimensional vector, trained on Wikipedia and Gigaword.

converting a document to a GLoVe vector -:

- To convert a document to a GLoVe vector, the (scalar) tf-idf value of each word in the document is multiplied by the GLoVe vector associated with the word, which is summed together and normalised for document length:

$$glove_{doc} = \frac{\sum (glove_{word} \times tfidf_{word,doc}) \text{ (for each word in the document)}}{\sum (tfidf_{word,doc}) \text{ (for each word in the document)}}$$

(Note that $glove_{word}$ is a vector, while $tfidf_{word,doc}$ is a scalar value)

Computing cosine similarity of GLoVe vectors for all headline-body pairs -:

- The GLoVe vector representation of documents is used as a feature by calculating the cosine similarity of the GLoVe vector representation of each article's headline and body.
- The cosine similarity of two similar-length vectors are computed with:

$$\text{cosine_similarity} = \frac{H \cdot B}{|H||B|} = \frac{\sum_{i=0}^{\text{len}(\text{glove})} (H_i \times B_i)}{\sqrt{\sum_{i=0}^{\text{len}(\text{glove})} (H_i^2)} \times \sqrt{\sum_{i=0}^{\text{len}(\text{glove})} (B_i^2)}}$$

Where H = GLoVe vector representation of headline,

B = GLoVe vector representation of body, and
 $\text{len}(\text{glove})$ is the dimensionality of the GLoVe vector representation used (in this case 50).



Return values of Cosine Similarity -:


- This feature returns a real value between -1.0 and 1.0 .
- higher = words in headline and body are more similar
- lower = words in headline and body are more different.

Language Model Representation (KL-Divergence) -:

- Another feature that we are using is the KL-Divergence of the language model representations of the headline and the article body.
- This is a measure of how divergent (i.e. different) are the language (i.e. words) being used in the headline and the body.
- To convert a document to a (simple, unigram) language model, we compute the probability of each word occurring in the document, by using the occurrence of the word:

$$LM_{doc}(word) = \frac{tf_{word,doc} \text{ (no. of times word occurs in doc)} + eps}{|doc| \text{ (no. of words in doc)} + eps \times |V| \text{ (no. of unique words in headline AND body)}}$$

language model



Computing the KL-Divergence of language model (LM) representations of the headline and the body

- KL-divergence is a measure of how different two probability distributions are. In this case, KL-divergence is used to measure the divergence between the language model of each article's headline and body.
- This formula is defined as:

$$kl_divergence = \sum \left(LM_{headline}(word) \times \ln \left(\frac{LM_{headline}(word)}{LM_{body}(word)} \right) \right) \text{ (for each word in } V \text{)}$$

value is, the more divergent the language model (i.e. words used) in the article's headline and body are.

Additional Features (N-gram overlap) -:

- N-gram overlap is a measure of how many times n-grams that occur on the article's headline re-occur on the article's body.
- For each article (headline-body pair), I counted n-gram overlaps up to 3-grams (i.e. count no. of words in headline that re-occur on body + no. of sequence of 2-words in the headline that re-occur in body + no. of sequence of 3-words in the headline that re-occur in body).

$$3gram_overlap = \left(\frac{\text{no. of 1/2/3grams in headline that reoccur in body}}{\text{no. of words in body (article length)}} \right)^{\frac{1}{e}}$$

(higher = words in headline and body are more similar, lower = words are more different).

```
Grade [32.87415513356936]
Agree [0.08512874408828167]
Disagree [0.47776183644189385]
Discuss [0.05152329749103943]
Unrelated [0.4830235980162407]
Accuracy [0.9272872152048165]
(base) abhinavgabhinav:~/fake_news$
```

Dense layer Model architecture

In [8]: `model.summary()`

Layer (type)	Output Shape	Param #
=====	=====	=====
dense_1 (Dense)	(None, 32)	128
dense_2 (Dense)	(None, 128)	4224
dense_3 (Dense)	(None, 128)	16512
dense_4 (Dense)	(None, 128)	16512
dense_5 (Dense)	(None, 128)	16512
dense_6 (Dense)	(None, 64)	8256
dense_7 (Dense)	(None, 64)	4160
dense_8 (Dense)	(None, 32)	2080
dense_9 (Dense)	(None, 32)	1056
dense_10 (Dense)	(None, 16)	528
dense_11 (Dense)	(None, 16)	272
dense_12 (Dense)	(None, 4)	68
=====	=====	=====
Total params: 70,308		
Trainable params: 70,308		
Non-trainable params: 0		


```
[10]: from sklearn.model_selection import train_test_split as tts
xtrain,xtest,ytrain,ytest = tts(data,y_pred,test_size=0.2,random_state = 0)
```

```
[11]: history = model.fit(xtrain,
                           ytrain,
                           epochs=50,
                           batch_size=512,
                           validation_data=(xtrain, ytrain))
```

WARNING:tensorflow:From /home/abhinav/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

Train on 4478 samples, validate on 4478 samples

Epoch 1/50

4478/4478 [=====] - 1s 155us/step - loss: 1.3724 - acc: 0.5911 - val_loss: 1.3388 - val_acc: 0.7329

Epoch 2/50

4478/4478 [=====] - 0s 25us/step - loss: 1.2564 - acc: 0.7329 - val_loss: 1.0825 - val_acc: 0.7329

Epoch 3/50

4478/4478 [=====] - 0s 24us/step - loss: 0.9737 - acc: 0.7329 - val_loss: 0.7991 - val_acc: 0.7329

Epoch 4/50

4478/4478 [=====] - 0s 23us/step - loss: 0.7487 - acc: 0.7329 - val_loss: 0.6909 - val_acc: 0.7329

Epoch 5/50

4478/4478 [=====] - 0s 23us/step - loss: 0.6808 - acc: 0.7329 - val_loss: 0.6391 - val_acc: 0.7329

Epoch 6/50

```
Epoch 40/50
4478/4478 [=====] - 0s 23us/step - loss: 0.3308 - acc: 0.8790 - val_loss: 0.3274 - val_acc: 0.8810
Epoch 41/50
4478/4478 [=====] - 0s 24us/step - loss: 0.3281 - acc: 0.8803 - val_loss: 0.3272 - val_acc: 0.8819
Epoch 42/50
4478/4478 [=====] - 0s 23us/step - loss: 0.3288 - acc: 0.8805 - val_loss: 0.3263 - val_acc: 0.8814
Epoch 43/50
4478/4478 [=====] - 0s 23us/step - loss: 0.3279 - acc: 0.8823 - val_loss: 0.3340 - val_acc: 0.8785
Epoch 44/50
4478/4478 [=====] - 0s 30us/step - loss: 0.3307 - acc: 0.8794 - val_loss: 0.3267 - val_acc: 0.8805
Epoch 45/50
4478/4478 [=====] - 0s 24us/step - loss: 0.3286 - acc: 0.8808 - val_loss: 0.3266 - val_acc: 0.8819
Epoch 46/50
4478/4478 [=====] - 0s 31us/step - loss: 0.3273 - acc: 0.8810 - val_loss: 0.3258 - val_acc: 0.8814
Epoch 47/50
4478/4478 [=====] - 0s 28us/step - loss: 0.3274 - acc: 0.8812 - val_loss: 0.3277 - val_acc: 0.8799
Epoch 48/50
4478/4478 [=====] - 0s 29us/step - loss: 0.3279 - acc: 0.8812 - val_loss: 0.3291 - val_acc: 0.8810
Epoch 49/50
4478/4478 [=====] - 0s 28us/step - loss: 0.3289 - acc: 0.8808 - val_loss: 0.3252 - val_acc: 0.8812
Epoch 50/50
4478/4478 [=====] - 0s 34us/step - loss: 0.3267 - acc: 0.8810 - val_loss: 0.3256 - val_acc: 0.8801
```

```
In [12]: predictions = model.predict(xtest)
         score = model.evaluate(xtest,ytest)
```

1120/1120 [=====] - 0s 26us/step

```
In [13]: model_json=model.to_json()
         with open("model.json", "w") as json_file:
             json_file.write(model_json)
         # serialize weights to HDF5
         model.save_weights("model.h5")
         print("Saved model to disk")
```

Saved model to disk

```
In [14]: test_loss, test_acc = model.evaluate(xtest, ytest)

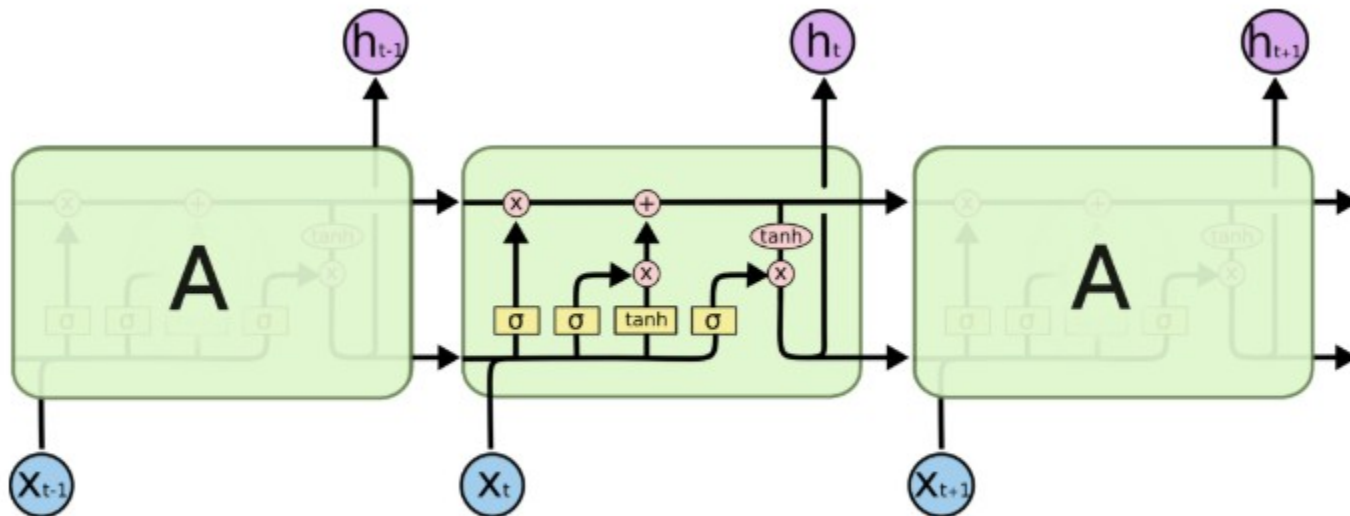
         print('Test accuracy:', test_acc)
```

1120/1120 [=====] - 0s 35us/step
Test accuracy: 0.8758928571428571

```
In [15]: print('Accuracy of the model obtained is ',score[1]*100)
```

Accuracy of the model obtained is 87.58928571428571

LSTM architecture



The repeating module in an LSTM contains four interacting layers.

Approach

1. Created word vectors of the headline and body separately using the GoogleNews word embeddings and make the feature matrix for the same.
2. We used the following model specifications while designing the model.

```
learning_rate = 0.01
epochs = 10
training_iters = 100000
batch_size = 50

*Network Parameters*
input_dimension = 300
inputs_to_hidden = 100
number_of_output_classes = 4
```

3. Using Two basic LSTM cell with **AdamOptimizer** for headline and body_lengths getting the word vector as input it will self generate it's features and creates the accuracy score learned from the vectored input.

LSTM RESULT

```
Activities Terminal Sun 18:20
abhinav@abhinav: ~/fake/word2vec

File Edit View Search Terminal Help
step is :40 cost is :1.1107158567646813 score is :83.24675324675324
step is :41 cost is :1.0649514195448027 score is :83.24675324675324
step is :42 cost is :1.066237101208423 score is :83.24675324675324
step is :43 cost is :1.0883094717261115 score is :83.24675324675324
step is :44 cost is :1.0431896594762866 score is :83.24675324675324
step is :45 cost is :1.1020611538695777 score is :83.24675324675324
step is :46 cost is :1.0801894483606231 score is :83.24675324675324
step is :47 cost is :1.0392551555464216 score is :83.24675324675324
step is :48 cost is :1.1175995022526464 score is :83.24675324675324
step is :49 cost is :1.0083624645357583 score is :83.24675324675324
step is :50 cost is :1.0500238886924982 score is :83.24675324675324
step is :51 cost is :0.9997003665713678 score is :83.24675324675324
step is :52 cost is :1.102634644011561 score is :83.24675324675324
step is :53 cost is :1.0196855973046641 score is :83.24675324675324
step is :54 cost is :1.0576689992137946 score is :83.24675324675324
step is :55 cost is :1.0453010152327857 score is :83.24675324675324
step is :56 cost is :1.0420613842192339 score is :83.24675324675324
step is :57 cost is :1.1250100275555686 score is :83.24675324675324
step is :58 cost is :1.0589890137267495 score is :83.24675324675324
step is :59 cost is :1.1128422061937067 score is :83.24675324675324
step is :60 cost is :1.0618462096993833 score is :83.24675324675324
step is :61 cost is :1.0732569926176654 score is :83.24675324675324
step is :62 cost is :1.0396540270390504 score is :83.24675324675324
step is :63 cost is :1.104985881447813 score is :83.24675324675324
step is :64 cost is :0.9843036456331116 score is :83.24675324675324
step is :65 cost is :0.9957362987430446 score is :83.24675324675324
step is :66 cost is :1.0659983523928007 score is :83.24675324675324
step is :67 cost is :0.9959333000135099 score is :83.24675324675324
step is :68 cost is :1.0282896178978411 score is :83.24675324675324
step is :69 cost is :1.1294844110611317 score is :83.24675324675324
step is :70 cost is :0.9854248854277589 score is :83.24675324675324
step is :71 cost is :1.0053001785432758 score is :83.24675324675324
step is :72 cost is :1.1133625656218133 score is :83.24675324675324
step is :73 cost is :1.005586523382103 score is :83.24675324675324
step is :74 cost is :1.0464827935557144 score is :83.24675324675324
Traceback (most recent call last):
```

Previous Works

- Ferreira and Vlachos [5] used the “Emergent” dataset and applied Logistic Regression to identify compare rumored claims against news articles that had been previously labeled by journalists with the goal of predicting the stance of the article towards the rumor.
- This team summarized each article into a headline and used a logistic regression model with features representing the article and claim to classify the combination of article and claim as either “for,” “against,” or “observing,” with a final accuracy level of 73%.

References

- TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <http://tensorflow.org/>.
- A. Agrawal, D. Chin, and K. Chen. Cosine siamese models for stance detection, 2017. URL <http://web.stanford.edu/class/cs224n/reports/2759862.pdf>
- B. Galbraith, H. Iqbal, H. van Veen, D. Rao, J. Thorne, and Y. Pan. A baseline implementation for FNC-1, 2017. URL <https://github.com/FakeNewsChallenge/fnc-1-baseline>.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014.
- Stance Detection in Fake News: A Combined Feature Representation. Bilal Ghanem, Paulo Rasom