Hyperpartisan News Detection

Lan Anh Ngo Quy and Nicole Feldbaum

Abstract

In this project, we essentially tried out two main approaches for sentiment analysis for articles in the **Semeval-2019, Task 4: Hyperpartisan News Detection** training data set. The first approach was inspired by (Baziotis et al., 2017) and made use of the sequential representation of each article, where we use an embedding layer, and a 2-layer bidirectional LSTM with a simple attention mechanism. As a comparison to the neural network method, we used an ensemble learning approach by means of Random Forest, utilizing Bag of Words (BOW) embeddings. For both approaches to construct the word embedding, we rely on a word frequency dictionary with lowest values mapped to the most frequently occurred words, not including stop words. After experimenting with hyper-paramter tuning for the LSTM model and different methods of ensemble learning, we observed the best results (in terms of f-measure) with the Random Forest model, proving that while simple, the method of averaging a sentence's word vectors like BOW is still a very strong baseline.

1 Introduction

1.1 Task definition

Given an article, with its associated links, author, title, and text, we will try to predict whether the article is hyperpartisan or not. This task in interesting because there is a lot of information we can extract from an article such as:

- Article length: articles have different length, and to restrict the number of parameters for the LSTM model, we represented an article with its first 550 words.
- Word embeddings: there are a lot of unsupervised word embedding methods such as GloVe, word2vec, not to mentioned other approaches involving deep representation of words and sentences.
- Related information: could we gather any useful information about the article's sentiment using its title and attached links?

Detecting whether an article is hyperpartisan is important, especially in the wave of fake news and biased information which peaked during the time of the 2016 Presidential Election. If a reader could have the knowledge of whether an article is hyperpartisan, that awareness could be helpful for them to decide on the reliability of the source.

2 Previous Work

This task has already been approached by several groups, and our project was inspired by papers written by these groups, as well as other papers dealing with sentiment analysis and LSTMs:
DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis:
In a somewhat similar Semeval Task in 2014 for detecting sentiments of Twitter posts ranging from negative to neutral to positive, the authors of this paper achieved top results using a deep learning approach. They developed their method of data pre-processing by putting open and close tags denoting hashtags, all-caps, dates, and emoticons in tweets. The pre-processed data is then embedded with a multi-dimensional Twitter word embeddings, and trained on a 2-layer bidirectional LSTM model with attention mechanism. In this project, while we adopted the method of parsing for all-caps, hashtags and dates, we only selected the first 550 of these parsed words, and constructed our word embedding using the aggregation of all words including hashtags and all of the open/close tags, with the more frequent word gets

mapped to the lower number, and zero-padded at the beginning for articles having fewer than 550 parsed words. Our LSTM model is built based on the initial set of hyper-parameteters used by this paper, however, we tried out different architectures on the development set to figure out the best performing model.

This just in: fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news (Horne and Adali, 2017)

The authors of this research suggested some heuristics we could use to distinguish fake news from reliable sources could be looking into the title's features such as number of stop words, number of proper nouns, or the language complexity of the article by using part-of-speech related features. Our approach made use of some title features such as the length (which we turned into a length group (e.g. 5-10 words)), number of all-capitalized words, whether the title contains a question mark or an exclamation points, and number of stop words in the title.

Understanding LSTM Networks:

Andrej Karpathy's post on RNNs *The Unreasonable Effectiveness of Recurrent Neural Networks* (Karpathy), and Edwin Chen's blog-post *Exploring LSTMs* (Chen) provide a good tutorial on understanding LSTMs.

LSTMs consist of cell state (all available long-term memory), hidden state (working memory - how to focus on memories that would be immediately useful), forget state (remember vector - when new inputs come in, which information to forget/remember), input gate (save vector - determine how much of the input to let into cell state), and output gate (focus vector - which pieces of long-term memory should be kept, which should be discarded). Thus, while RNN uses only one equation to update its hidden state, each memory/attention mechanism of LSTM is a mini-brain of its own. LSTMs could be useful for the task of sentiment analysis because using the sequential relation of words in articles seems intuitively useful for inferring the sentiment. However, we note that there could be stop words or not so relevant words that wouldn't contribute to understanding the context, and thus as later described in methodology, we filtered out stop words during our feature processing step.

Glove: Global vectors for word representation:

Model	F-measure
Multinomial Naive Bayes with BOW features	0.687
Multinomial Naive Bayes with BOW and title features	0.685

Table 1: Preliminary results with naive Bayes model

For word embedding, particularly for our LSTM model, we made use of Glove's pretrained word vector, which was trained on a combination of 5 corpora from Wikipedia and Gigaword - containing 6 billion tokens in total. The training process obtained vector representations of words in an unsupervised learning manner by aggregating global word-word co-occurrence statistics from a corpus. As a result, the Glove vectors produced provide a meaningful substructure of each word with the properties combined from both global matrix factorization and local context window methods (Pennington et al., 2014).

3 Preliminary Results

We already have preliminary results from a Multinomial Naive Bayes classifier using a bag of words model as its only feature, and a Multinomial Naive Bayes classifier using features including a bag of words model, length of the title, whether or not the title contains a question mark, number of proper nouns in the title, and number of stop words in the title.

As can be seen from this chart, our multi-feature bag of words + title features classifier actually performed about the same as our single-feature Multinomial Naive Bayes classifier. We suspect that our classifier with bag of words + title is not performing better because there is no weighting of features in Multinomial Naive Bayes.

4 Methodology

We built and conducted parameter-tuning for LSTM and Random Forest models by splitting up two separate training - development sets from the initial set of intraining articles. Given the best-performing models for evaluation in terms of precision, recall and f-measure on validation articles, provided to us by the Hyperpartisan News Detection task organizers.

4.1 Preprocessing

Our preprocessing approach consists of mainly two steps: parsing the articles, and word-index mapping. Whether constructing features for the LSTM model, or other non-neural-network models, there is some difference in our choice of just having the features as words mapped to index, or using bag-of-words (BOW) features for each parsed article's content, top three associated hyperlinks, title-related features and number of censored words (words contained in the list of banned and swear words by Google).

4.1.1 Parsing

First step, we tokenized each article, and started appending alphabetic words that are not included in the list of nltk's stop words to a list, during which:

- For the sake of using word embedding later, we uncovered censored words (e.g. s**t)
- Parse for hashtags and add open/close tags surrounding the hashtagged word (e.g. #metoo → <hashtag>, metoo, </hashtag>)
- Parse for and add an <allcaps> symbol after a word which contains only capital letters
- Change all sets of strings identified as date-time related to <date>

While appending parsed words to a list, we capped this feature list to contain only up to a set number of words, and experimental results led us to decide on choosing 550 as the max length for parsing an article. For articles that did not reach the upper bound limit of 550 tokenized featu

4.1.2 Word-index mapping

From an aggregated list of all words occuring the training samples, we created a word-index dictionary, in which the more frequent a word occurred, the lower number it got assigned, starting from 1. After that, we mapped each article's word features from the previous step to the corresponding word index. For articles that did not as many as 550 word features, we padded 0 at the beginning to ensure that every article could be represented by an equal number of word-index features.

4.1.3 Model-dependent features

While for the LSTM model, the feature processing step is done, for non-neural-network models, from the preprocessed feature of previous steps,

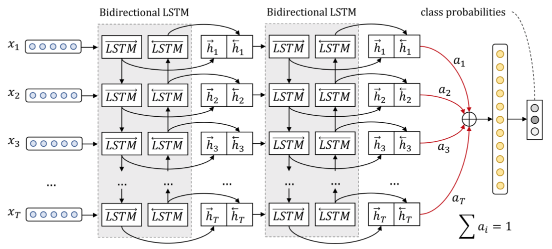

Figure 1: A 2-layer bidirectional LSTM with attention mechanism in the last layer (Baziotis et al., 2017)

- Construct a BOW feature vector for each article
- Record hyperlinks associated to an article, truncate each link to its principal domain (e.g. www.facebook.com), and choose 3 most frequent links (or padded with <empty_link>). From all articles' top 3 hyperlinks, we constructed, in a BOW manner, hyperlink features for each article
- Parse each article's title and record 5 title-related features: title's length (modded by 5), number of all-capitalized words, whether the title contains a question mark, whether it contains an exclamation point, and number of stop words in the title
- Add a feature for number of banned words occuring in each article within the max length (550)

4.2 LSTM Model

Our LSTM architecture consisted of an embedding layer, which we used a 300-dimensional word embedding, followed by a SpatialDropout1D layer, added in GaussianNoise of 0.3, n layers of LSTM with optional Dropout, an Attention Layer with optional Dropout, and finally a Dense layer with output dimension of We refer to this architecture as the *default* model, with which we conducted experiments a training set and evaluated on the development set by subsequently changing each model parameter.
First, on 200 training articles, we wanted to choose the best numbers of batch size, epochs, and the length of article to consider.
When testing batch size, number of epochs, and length of articles, we were only able to test on 300 training articles because we took so many data samples and LSTM model often takes a long duration to finish running. However, from this

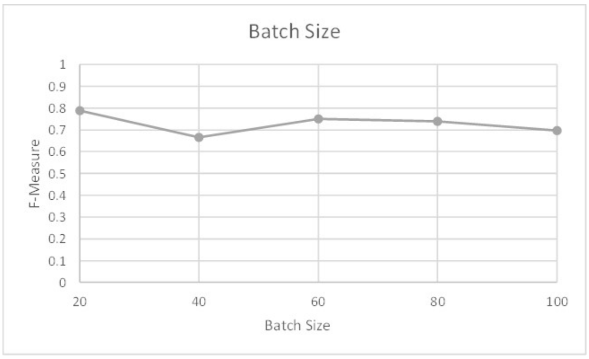

Figure 2: A graph depicting batch size and corresponding f-measure as tested from 20-100.

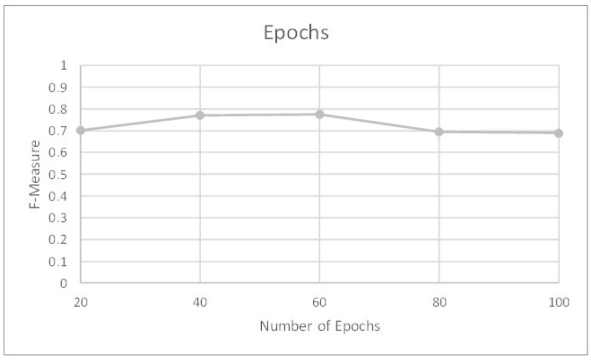

Figure 3: A graph depicting number of epochs and corresponding f-measure as tested from 20-100.

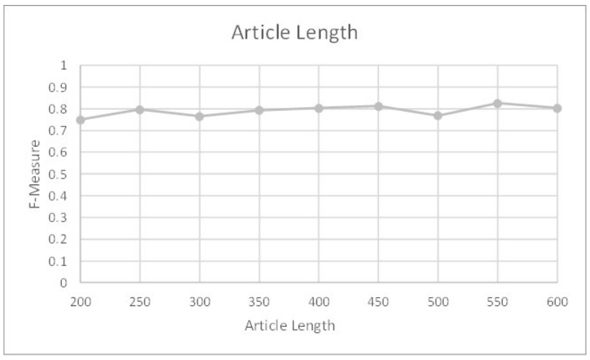

Figure 4: A graph depicting article length considered and corresponding f-measure as tested from 200-600.

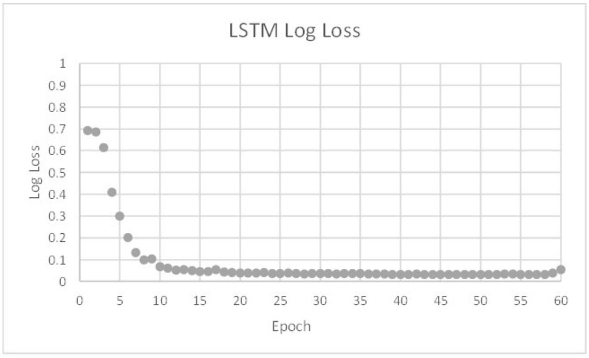

Figure 5: Log-loss of LSTM model by number of epochs

Model	F-measure
default	0.75
non-bidirectional	0.62
1-layer LSTM	0.61
3-layer LSTM	0.62
0 dropout rate	0.75
memory attention	0.49
0 dropout attention	**0.80**

Table 2: LSTM's parameter tuning experiments on 400 articles

Model	F-measure
default	0.75
0 dropout attention	**0.78**
Glove embedding	0.70

Table 3: LSTM's parameter tuning experiments on 2000 articles

small sample size, we saw no real trend in batch size, number of epochs, or article length. Despite not seeing a trend, we use the best values found in these charts (batch size: 20, number of epochs: 60, article length: 550).

With regards to the LSTM model's architecture, we conducted similar experiments on 400 training articles for numbers of LSTM layers (1-3 layers), whether to use bi-directional for LSTM layers or not, type of Attention mechanism (memory versus simple), whether to include a Dropout layer after Attention, and whether to include dropout layers in between any other types of layers. As shown in **Table 2**, we found that our classifier performed best with 2 layers of bi-directional LSTM layers, including Dropout layers before and after LSTM layers, but not after the Attention layer, and with "simple" attention mechanism.
Finally, on a larger training data set of 2000 articles, and evaluated on 200 articles, we found that without the dropout layer after the Attention layer performed the best on F-measure, as shown in **Table 3**. We noticed that using the pre-trained Glove vectors did not help with increasing the classifier's performance at all, this could be because all the additional features we got from parsing such as <allcaps>, <hashtag>, and <date>.

4.3 Random Forest Model

With the same set features described as in **4.1.3**, we evaluated the performance of models on a

Model	F-measure
AdaBoost	0.68
Random Forest	**0.722**
Bagged Random Forest	0.717

Table 4: Ensemble with bagging/boosting models' performance

Model	F-measure
Multinomial Naive Bayes (baseline)	0.69
LSTM	0.63
LSTM with Glove embedding	0.64
Random Forest	**0.84**

Table 5: Final experiments' results, trained on 10000 articles and evaluated on 1000 articles

training set of 2000 articles, evaluated on developement set of 200 articles. We used 3 different algorithms: adaboost (ensemble methods with boosting), random forest (ensemble methods with bagging), and a bagging version of random forest.

As a result, we decided to choose to classify using Random Forest over AdaBoost and Bagged Random Forest.

5 Results

Because of the time constraint, we were able to train the model on a training size of 10000 articles, and evaluated the result on 1000 articles in the test set. Using the baseline of multinomial naive Bayes model with BOW features as mentioned in our preliminary results section, we chose 3 other models to compare with one another: a LSTM model (without attention dropout), a similar LSTM architecture as previous but with Glove vector word embedding, and lastly a Random Forest model.

As shown in **Table 5**, our Random Forest model performed the best, and while significantly more arduous to implement than the baseline model, LSTM models unfortunately did not perform as well. For our random forest model, we were also able to extract the most important features used, 20 features with the highest scores are listed in **Table 6**. It is quite interesting to note that not only our BOW features (normal words), but also title features, all-capitalised words, and hyperlink features get included in this table.

	Feature		Feature
1	below	11	by
2	continue	12	globalpost.com
3	reading	13	says
4	AP	14	baptist
5	stop_words	15	window
6	title_length	16	NYSE
7	<allcaps>	17	said
8	<empty_link>	18	trump
9	reuters	19	NASDAQ
10	UPI	20	image

Table 6: 20 most important features, determined by Random Forest model on trained on 10000 articles, evaluated on 1000 articles

6 Discussion

After all experimenting with parsing methods, sequential model architecture using deep learning with LSTM models, or ensemble methods such as Random Forests, on the final training set of 10000 articles, we got the best results with Random Forest on 1000 evaluation articles. Parsing the articles and adding features seem to be the most useful step for increasing our classifier's performance, as demonstrated by highest F-score of random forest model, and the list of most important features by this model.

References

Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754. Association for Computational Linguistics.

Edwin Chen. Exploring lstms.

Benjamin D Horne and Sibel Adali. 2017. This just in: fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news. *arXiv preprint arXiv:1703.09398*.

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.