USING NEURAL NETWORKS TO DETECT EXAMPLES OF AD HOMINEM IN POLITICS AND MICROBLOGGING PLATFORMS

Jordan Goetze
Computer Science
North Dakota State University
Fargo, North Dakota 58102
jordan.goetze@ndsu.edu

Abstract

With the presently ongoing race for the United States presidency, analysis of the currently participating candidates is under heavy analysis by politicians and civilians alike. A well noted factor of many of the candidate's oratory during debates are examples of the logical fallacy ad hominem. Ad hominem is a logical fallacy where instead of responding to another's argument, the person seeks to undermine the argument by attacking the personal character of the argument giver. Recently, candidates have begun to make use of microblogging services as a way to publicize their opinions and as such candidates often commit ad hominem against one another. The social nature of microblogging services commonly allows private users to comment upon candidate posts creating additional opportunities for ad hominem to be used.

1 Introduction

This paper will describe an attempt to utilize a neural network – specifically a Convolutional Neural Network(CNN) – to detect examples of the logical fallacy ad hominem in political microblog posts. This model could then be applied more generally to the analysis of arguments with special emphasis on political aguments such as debates or microblog conversations on networks such as Twitter. This model makes use of the word2vec model by Mokolov et al. [1] to train word embeddings which can then be used with a set of positive and negative examples of ad hominem by a CNN. The neural network should ideally be able to detect with a reasonable margin of error whether or not a sentence is an example of ad hominem based on the contextual likelihood learned by the model. One set of data will be collected for several separate but related experiments. The experiments will utilize microblog posts posted by presidential candidates on Twitter, that are then collected, and manually classified as either example or non-example of ad hominem.

2 Model

Our model is based heavily off of the Tensorflow implementation by Britz [2] which is in turn based on a model designed for gerneralized sentence classification by Kim [3]. We use a model derived from Kim’s because Kim’s model–and Britz’s derived model–was shown to be relatively adept at generalized classification tasks.

Significant differences between Britz’s and Kim’s models as reflected in our model include, the absence of pre-trained word vectors–Britz’s model trains it’s own word embeddings, and the absence of L2 norm constraints on weight vectors. Additionally Britz’s model utilizes only one non-static input channel where as Kim’s model uses one static input channel and one non-static input channel. Our model follows Britz’s with only a single, non-static input channel.

3 Dataset and Experimental Setup

The model was trained using 10-fold cross validation on classified microblog posts from Twitter (henceforth referred to as tweets).

Total Tweeets	5808
Negative Examples (not ad hominem)	4155
Positive Examples (ad hominem)	1653

Table 1: Tweet totals and positive/negative total example breakdown

Tweets were collected between January 2016 and March 2016. Tweets were collected from the Twitter feeds of six 2016 presidential candidates. This includes candidate posts as well as public responses to candidate posts made by Twitter users.

3.1 Dataset Preprocessing

All tweets collected were preprocessed to remove capitalization, excessive spacing, punctuation, HTML encoded greater-than and less-than ("<" and ">") symbols, and HTML encoded ampersands ("&"). All twitter user names–indicated by an "@" symbol followed by a user name, e.g. "@realDonaldTrump"–were replaced with the non-unique "<AT_NAME/>" token. All URLs were likewise replaced with a "<URL/>" token.

Example Tweet	Preprocessed Tweet
@HillaryClinton why always so #smug? https://t.co/eOU1rOaOlR	<AT_NAME/> why always so #smug <URL/>

Table 2: Preprocessing Example

3.2 Hyperparameters

As with Kim's model, little hypeparameter tuning was necessary for our model. As such our model was trained using many of the same hyperparameter settings as Kim reported: filter windows of 3, 4, and 5. Dropout rate of 0.5, and l_2 constraints of 3. We use a mini-batch size of 64, and 50 filters per filter size to prevent over fitting due to the small size of our dataset. Due to the fact that the model does not use pre-trained word vectors, we use 20 dimension word embeddings.

1.3 Model Variations

We experimened with three different variations of the model.

- **N-T-Names:** Our baseline model, where non-unique twitter names are removed and we initialize word vectors randomly to be modified during training.
- **U-T-Names:** Our baseline model but where twitter user names are replaced with a unique token such as "<AT_NAME_123/>".
- **G-N-Vecs:** Our Baseline model initialized with 300 dimensional pre-trained word vectors from word2vec trained on Google News articles [1]. All word vectors not provided with the pre-trained word2vec model are initialized randomly. Table 3 illustrates the breakdown between pre-trained word embeddings and randomly initialized word embeddings.

Total Words	8313	100% of words
Words From Pre-Trained Embeddings	5841	70.3% of words
Randomly Initialized Word Embeddings	2472	29.7% of words

Table 3: Comparison of pre-trained word vectors to randomly initialized word vectors.

4 Results and Discussion

The results of our models are listed in Table 4.

Model	Accuracy	Specificity	Sensitivity
N-T-Names	**87.7**	100	**39.4**
U-T-Names	87.3	100	37.7
G-N-Vecs	87.2	100	25.4

Table 4: Results of our models when trained and tested with 10-fold cross validation.

4.1 N-T-Names versus G-N-Vec

The G-N-Vecs model performs the worst of the three models. This comes as a surprise as Kim[3] reports a much greater degree of positive benefit from using the pre-trained vectors and even denotes surprise at the performance gains. In our model, we see a 36% margin of difference in sensitivity between the G-N-Vecs model and the N-T-Names model. A possible attribution for this difference may be the difference in word vector dimensions. The G-N-Vecs model was initialized with 300 dimension vectors while the other two models began with only 20 dimension vectors.

4.2 N-T-Names versus U-T-Names

The N-T-Names model was able to achieve a higher accuracy and sensitivity than the U-T-Names model which was rather unexpected. I had expected that both models would preform similarly because I thought the name token in both models would hold the same amount of significance whether specific (“<AT_NAME_123/>”) or generalized (“<AT_NAME/>”). As it turns out, this was not the case, the existence of a generalized name seems to hold more importance than a specific name. A potential test for this idea would be to replace each unique name in the original data set with a matching unique token, e.g. “hillary” is always “<AT_NAME_345/>” and “trump” is always “<AT_NAME_789/>”.

5 Conclusion

We have described an adapted model purposed and tested to detect instances of ad hominem in political tweets. Following Kim's example, little hyperparameter tuning was needed in order to produce reliable results. Additionally, further work may be done to produce alternative datasets with which to test the model–presidential debates for example–which would represent a more generalized test set. In total, we have successfully implemented and tested a model capable of detecting ad hominem with no prior knowledge of the generalized structure of a sentence containing ad hominem and were able to produce reasonably reliable results on a relatively small dataset.

6Acknowledgments

Special thanks to Dr. Anne Denton at the University of North Dakota Computer Science Department for overseeing this research project as part of an independent study course.

7 Source Code and Dataset

The source code for this project can be found at:

- https://github.com/jghibiki/NeuralAdHominem

The data set for this project can be found at:

- http://students.cs.ndsu.nodak.edu/~jgoetze/NeuralAdHominem/data.csv

References

[1] Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. In *Preceedings of ACL 2002*, October 2013.
[2] Britz. D. Implementation a CNN for Text Classification in Tensorflow. *WildML*, December 2015.
[3] Kim, Y. Convolutional Neural Networks for Sentence Classification. In *EMNLP 2014.* September 2014.
[4] Abadi et al. Vector Representations of Words. 2015.
[5] Goldberg and Levy. Word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. February 2014.