Sarcasm Detection with News Headlines Data

Zhe Zhou, Xinyao Mo
Course: ANLY-521 Computational Linguistics with Advanced Python
Instructor: Dr. Elizabeth Merkhofer
29th April 2020

***Abstract*—Humans have a social nature. Social nature means that we interact with each other in positive, friendly ways, and it also means that we know how to manipulate others in a very negative way. Sarcasm, which is both positively funny and negatively nasty, plays an important part in human social interaction. Sarcasm detection is a key task for many natural language processing tasks. To date, most approaches to sarcasm detection have treated the task primarily as a text categorization problem. Sarcasm, however, can be expressed in very subtle ways and requires a deeper understanding of natural language that standard text categorization techniques cannot grasp. In this project, we will use various approaches from classic machine learning classification algorithms with TF-IDF text matrix to word embedding based recurrent neural network approach.**

I. Introduction

Sarcasm detection (as shown in Figure 1) is a very narrow research field in NLP, a specific case of sentiment analysis where instead of detecting a sentiment in the whole spectrum, the focus is on sarcasm. Therefore the task of this field is to detect if a given text is sarcastic or not. The first problem we come across is that, unlike in sentiment analysis where the sentiment categories are very clearly defined (love objectively has a positive sentiment, hate a negative sentiment no matter who you ask or what language you speak), the borders of sarcasm are not that well defined. And it is crucial that before starting to detect it, to have a notion of what sarcasm is. Moreover, someone being sarcastic does not mean the other person perceiving it as the speaker intended. This subjectivity will have implications in the performance of our machine learning models.

II. Related Work

A substantial literature exists around sarcasm detection. Many of the prior studies focus on the analysis of Twitter posts, which lend themselves well to sarcasm detection with NLP methods because they are available in large quantities, they tend to correspond roughly to a single utterance, and users hashtags in tweets (e.g., sarcasm, not) can provide imperfect but useful labels. A central theme of this literature is that bringing in contextual features helps performance. Ghosh and Veale[1] present a combination LSTM (long short-term memory RNN) architecture that takes as inputs user affect inferred from recent tweets as well as the text of the tweet and that of the parent tweet. When a tweet was addressed to someone by name, the name of the addressee was included in the text representation of the tweet, providing a loose link between interlocutors.

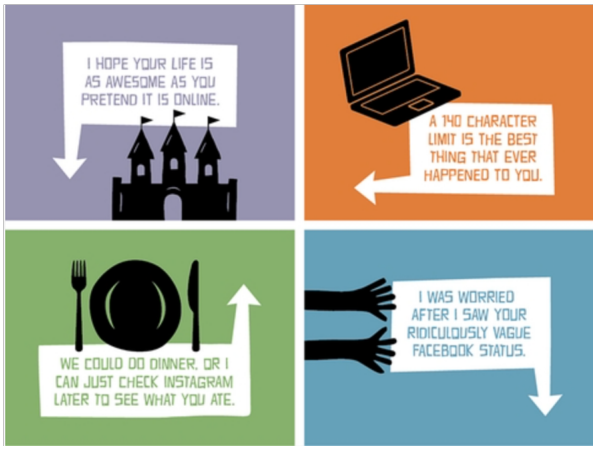


Fig. 1: Simple sarcasm example

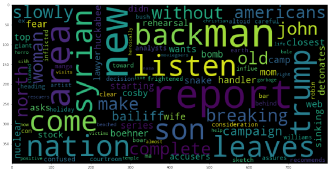
The work that is closest to our own is that of Kolchinski and Potts[2], who also experiment on the News Headline dataset. Their BiLSTM model learns author, forum, and text embeddings, and they show that all three kinds of representation contribute positively to the overall performance. We take a much simpler approach to different pre-trained Global word embeddings and do not include either forum embeddings or author embeddings, and we apply LSTM insdead of BiGRU. Finally, we report comparable performance according to different embedding and algorithms.

III. Dataset

We have given the News Headlines Dataset on Kaggle and our goal is to predict whether a given text is sarcastic or not. This News Headlines dataset for Sarcasm Detection is collected from two news website. TheOnion aims at producing sarcastic versions of current events and we collected all the headlines from News in Brief and News in Photos categories (which are sarcastic). We collect real (and non-sarcastic) news headlines from HuffPost. A wordcloud example is shown in Figure 2, which is made by the records with sarcasm and non-sarcasm label.

This dataset has three columns:

- Articlelink (type: Object): contains links to the news articles



(a)

(b)

Fig. 2: (a) Word-Cloud of Sarcasm Label Data (b) Word-Cloud of Non-Sarcasm Label Data

- Headline (type: Object): contains headlines of the news articles.
- issarcastic (type: int64): contains 0 for nonsarcastic text, and 1 for sarcastic text.

This reasons why we chose this dataset over the Twitter dataset are:

- Since news headlines are written by professionals in a formal manner, there are no spelling mistakes and informal usage. This reduces the sparsity and also increases the chance of finding pre-trained embeddings.
- Since the sole purpose of TheOnion is to publish sarcastic news, we get high-quality labels with much less noise as compared to Twitter datasets.
- Unlike tweets which are replies to other tweets, the news headlines we obtained are self-contained. This would help us in teasing apart the real sarcastic elements.

IV. Methodology

A. Classic Machine Learning Algorithms

- **Gaussian Naive Bayes:** Naive Bayes is a family of algorithms based on applying Bayes theorem with a strong(naive) assumption, that every feature is independent of the others, in order to predict the category of a given sample. They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{1}$$

$$P(y|x_1, x_2, ..., x_n) = \frac{P(x_1|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)} \tag{2}$$

- **Logistic Regression:** Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In contrast with Bayes Classifier where the coefficients are calculated in theory, logistic regression is a parametric model with the coefficients are learned during the training process on the TFITF text matrix.

$$P(y = 1) = \frac{1}{1 + e^{-(wx+b)}} \tag{3}$$

$$P(y = 0) = \frac{e^{-(wx+b)}}{1 + e^{-(wx+b)}} \tag{4}$$

- **Linear Support Vector Classifier :** The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin.
- **Random Forest:** Random forests (Figure 3) are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees habit of overfitting to their training set. Random Forest can be applied on both one-hot encoded or TF-IDF matrix.

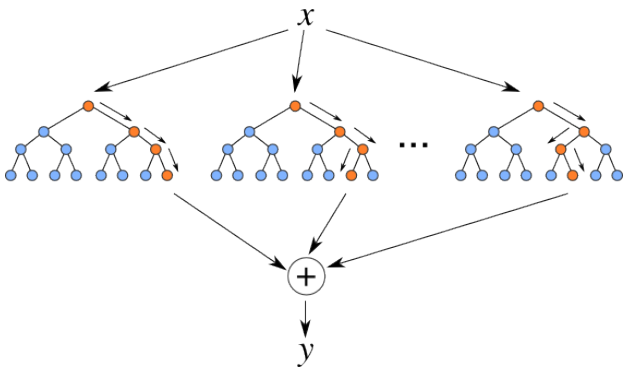


Fig. 3: Simple Random Forest Illustration

B. LSTM Model

LSTM is a specific type of reccurent neural network. The inputs to the LSTM model are text, which are split into words and punctuation marks, and are converted to word vectors. The nal states of the two directions of the LSTM are concatenated with each other and run through either a single fully-connected linear layer. The output of the nal linear layer is fed through a sigmoid function which outputs the estimated probability of sarcasm. Figure 4 can better illustrate this process.

C. Text Preprocessing

In all cases, the raw comment data was tokenized into words and punctuation marks, with components of contractions treated as individual words, then the tokens are sent to normalization and noise removal processes in the following order:

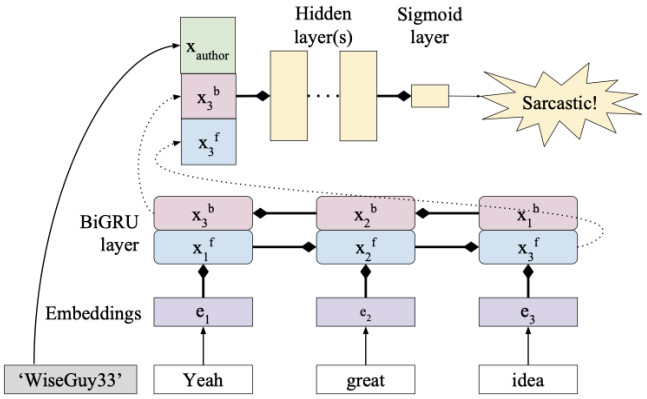


Fig. 4: LSTM Illustration

- Remove HTML tags
- Remove extra whitespaces
- Remove special characters
- Convert number words to numeric form
- Remove stopwords
- Lemmatization

D. Model Training

Classic machine learning classification models are trained with TF-IDF matrix which is based on preprocessed tokens using TfidfVectorizer from sklearn. Randomsearch is applied to random forest model for hyperparameter tuning.

As for the LSTM model, we apply two different pretrained word embedding methods to transfer preprocessed tokens into word vectors. First method is GloVe(glove.6B.100d) from Stanford University. Second method is Fasttext(crawl-300d-2M) from Facebook.

There are quite a few hyper parameter of LSTM that could be tuned to improve the results:

- Learning Rate
- Batch Size
- Sequence Length
- Clip Gradients at Value
- Decay Rate
- GPU Memory

V. Results

A. Classification Results

TABLE I: Results on Training set

Models	Accuracy	Precision	Recall	F1
Gaussian Naive Bayes	0.78	0.7	0.9	0.79
Linear SVM	0.92	0.91	0.9	0.91
Logistic Regression	0.88	0.87	0.86	0.87
Random Forest	0.99	0.99	0.98	0.99
LSTM (GloVe)	0.96	0.96	0.95	0.95
LSTM (FastText)	0.98	0.98	0.97	0.97

TABLE II: Results on Testing set

Models	Accuracy	Precision	Recall	F1
Gaussian Naive Bayes	0.70	0.62	0.81	0.70
Linear SVM	0.82	0.81	0.79	0.80
Logistic Regression	0.82	0.81	0.79	0.80
Random Forest	0.79	0.78	0.74	0.76
LSTM (GloVe)	0.86	0.84	0.83	0.83
LSTM (FastText)	0.84	0.83	0.82	0.82

According to the classification results Table I and Table II, LSTM has overall the best performance. GloVe and FastText word embedding do make slight difference in LSTM results. More precisely speaking, LSTM with GloVe embedding has slightly higher testing accuracy whereas LSTM with FastText embedding has slightly higher training accuracy.

Among those classic machine learning algorithms, SVM and Logistic Regression have the best testing accuracy. However, Logistic Regression tends to generalize better than SVM. Random Forest seems overfitting on the training data, tree proning would help solve this issue.

B. LSTM Training Results Visualization

Following Figure 5 to Figure 8 illustrate the training/testing loss and accuracy visualizations.

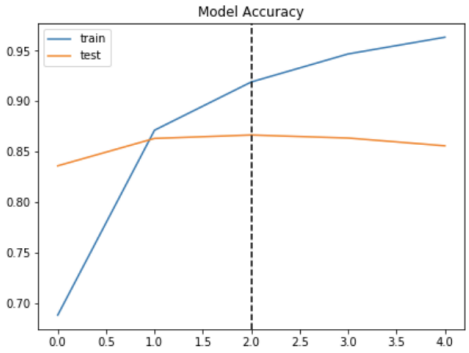


Fig. 5: LSTM (GloVe) Accuracy

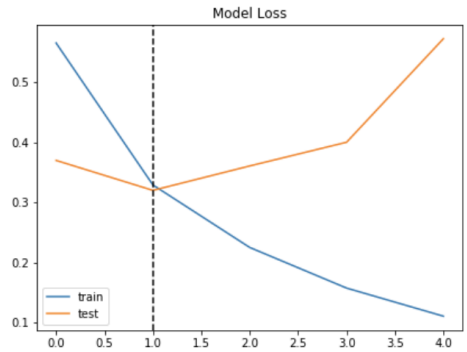


Fig. 6: LSTM (GloVe) Loss

According to Figure 6 and Figure 8, our training loss converges quite well. The model with GloVe embedding

converges to around 0.1, the model with FastText embedding converges to around 0.06. Testing loss both show a very clear U shape, and both have the elbow point happened at first epoch with value around 0.32.

According to Figure 5 and Figure 7, the model with GloVe embedding does slightly better job on training accuracy than the model with FastText embbeding, former has around 0.96 whereas latter has 0.98. In terms of testing accuracy, there are not much difference between these two methods, both curve tend to be flat and has maximum value around 0.86. However the maximum testing accuracy of GloVe happened at second epoch whereas that of FastText happened at first epoch.

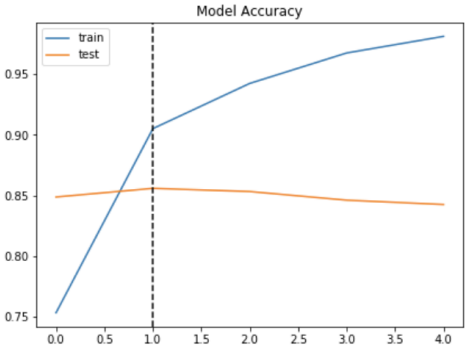


Fig. 7: LSTM (FastText) Accuracy

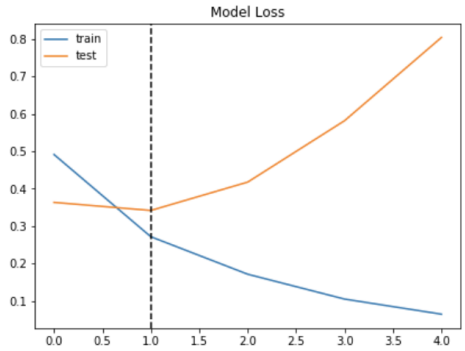


Fig. 8: LSTM (FastText) Loss

VI. Discussion

As discussed in the introduction, sarcasm is very much topic-dependent and highly contextual. For example, let us consider the sentence I am so glad to see England played very well, I can now sleep well :P. Unless one knows that England actually did not play well in that game, it is not possible to spot the sarcastic nature of this sentence. Another important fact is that an n-grams model cannot perform well on unseen data unless it is trained on a very large corpus. If most of the n-grams extracted from the unseen data are not in the vocabulary of the already trained n-grams model, in fact, the model will produce a very sparse feature vector representation of the dataset. Instead, we use the word2vec embeddings as the source of the features, as word2vec allows for the computation of similarities between unseen data and training data. However, because words may have multiple senses, some embeddings may lead to error. For example, in the sentence Great. Relationship advice from one of Americas most wanted, different interpretation of the word 'wanted' can lead to great difference.

Among those classic machine learning models, Gaussian Naive Bayes has the poorest performance. It might because that this assumption, that is, the features are strictly independent, does not apply to sarcasm detection.Models perfomance may be further improved if more features are added, such as topic and length.

Due to the lack of time, when tuning the hyperparameter, we only tried Randomsearch and moderate Gridsearch. We believe a thorough Gridsearch can further improve our Random Forest and LSTM model.

VII. References

[1] Y. Alex Kolchinski, Christopher Potts. Representing Social Media Users for Sarcasm Detection, EMNLP 2018.

[2] Aniruddha Ghosh and Tony Veale. 2017. Magnets for sarcasm: Making sarcasm detection timely, contextual and very personal. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 482491.

[3] Silvio Amir, Byron C Wallace, Hao Lyu, and Paula Carvalho Mario J Silva. 2016. Modelling context with user embeddings for sarcasm detection in social media

[4] Santiago Castro, Devamanyu Hazarika, Vernica Prez-Rosas, Roger Zimmermann, Rada Mihalcea and Soujanya Poria. 2019. Towards Multimodal Sarcasm Detection (An Obviously Perfect Paper). ACL 2019

[5] Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. 2016. A deeper look into sarcastic tweets using deep convolutional neural networks.

[6] Bamman, David and Noah A. Smith. 2015. Contextualized Sarcasm Detection on Twitter. ICWSM 2015

[7] N. Majumder, S. Poria, H. Peng, N. Chhaya, E. Cambria and A. Gelbukh. 2019. Sentiment and Sarcasm Classification With Multitask Learning. IEEE 2019.

[8] Reid Swanson, Stephanie Lukin, Luke Eisenberg, Thomas Chase Corcoran, Marilyn A. Walker. 2017. Getting Reliable Annotations for Sarcasm in Online Dialogues.