Krishna Shivkumar and Aleks Likhoshea
CS461: Machine Learning Principles
Sharon Levy

Sarcasm Detection Model Report

1. Introduction

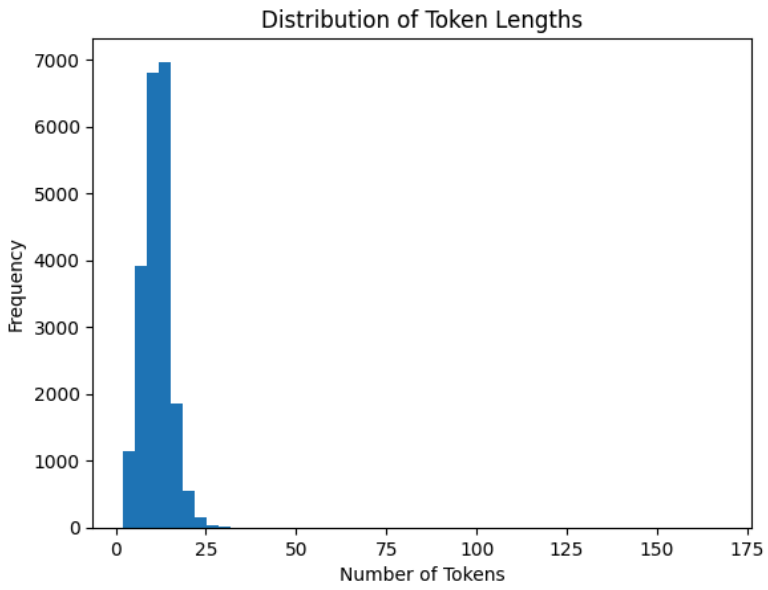
In this report, we'll cover how our group, Krishna Shivkumar and Aleks Likhoshva, worked to create a model that accurately and consistently determined whether a provided text sample was sarcastic or not.

In order to create our final model, we first cleaned our data and analyzed the best token length. We then tried numerous basic models , ranking them by accuracy, as well as different feature extraction methods and softwares. Finally, we combined the first order models together through different techniques and 2nd order models, while adjusting hyperparameters as needed, resulting in our final model that uses a bi-direction LSTM and a CNN model fed into a second order XGBoost model that predicts sarcasm at a 90% test accuracy.

Within our group, both of our members tested half the models and wrote approximately half the code. In addition, our group also worked on the report equally, with every section written by one member being reviewed and edited by the other.

2. Data Exploration & Preprocessing

This dataset had a single feature, the text itself. Since the text had a limited number of attributes, since "null" text wouldn't even exist as a row in the dataset and the lack of other features made finding preliminary relationships an impossibility, there was nearly no preliminary cleaning to be done. In fact, the provided dataset was already in all lower case, and a preliminary scan of all the text showed that a very small percentage of the text data had punctuation, with ending punctuation already being cut off. As such, the single variable that was evaluated was token length, as seen below.



This token value was then used for limiting the LSTM and CNN vector sizes, which slightly improved accuracy and moderately improved training time.

3. Feature Engineering

Our final model used word embeddings from the models provided by Gensim. While one of the preliminarily tested models, naive bayes to be exact, utilized TF-IDF, that was removed after further testing indicated that its utilization negatively impacted the final model. Overall, we chose word embeddings over TF-IDF as extraction method for 2 main reasons; the first was that word embeddings are good at determining semantic meaning, while TF-IDF is mainly good at determining sarcasm through the importance of specific words or word phrases, which holds less value when working with smaller amounts of text. As for the second reason, during testing our group determined using multiple models caused over-training issues, and between all the models we tested, naive bayes, and by proxy TF-IDF, were the least accurate and the least beneficial to our second order model.

For the LSTM, the sentences had to first be split into tokens before being vectorized as word embeddings. We initially used RegEx to split up the sentences, but we found a better alternative called Spacy which had a pretrained CNN which was superior at separating sentences. These tokens were then converted into 100 length vector embeddings using the Gensim model, after which they were extended to the length of 30 as determined by our preliminary data exploration for faster training and compatibility with tensorflow layers.

As for the CNN, n-gram's were utilized in order to find groups of words with greater semantic meaning, which improved the CNN's ability to find and recognize sarcastic text.

4. Model Architecture & Selection

During the process of testing to find the final model, 4 models were considered: a bidirectional LSTM, since they are known to be good at detecting sequential patterns while considering sentimental meaning, a text CNN since they are good for detecting local spatial patterns, a Naive Bayes model as a decent baseline and a potential secondary model, and XGBoost as a gradient decision tree model to compare with Naive Bayes.
These models had the following accuracy values:

Bidirection LSTM:	0.8861
Text CNN:	0.862
Naive Bayes:	0.753
XGBoost:	0.903

In the end, our final model is a Stacking Ensemble that combines a sequence-based model (LSTM) and a feature-based model (CNN) using a meta-learner. The primary model , the Bidirectional LSTM, used 64 units with Global Max Pooling to capture long-range dependencies and sequential context. The secondary model, a Multi-Channel CNN, featured three parallel branches, of kernel size 3, 5, and 7, with 128 filters each to detect specific sarcastic n-grams and phrases. The meta learner, blender, was an XGBoost Classifier that takes the probability outputs of the LSTM and CNN as input features to make the final prediction.

The main hyperparameters that were used were the number of nodes on the metalearner tree where learning rate was adjusted to be higher and the number of nodes was significantly reduced to simplify the model and prevent overfitting.

<u>Model Suitability</u>

The LSTM catches the sequential context, and overall phrase meaning due to its memory properties. CNN catches local patterns that the LSTM misses. The Ensemble combines these 2 models, allowing the model to succeed even if one base learner fails to detect the sarcasm pattern.

5. Training Methodology

The Adam optimizer was used for all neural network models (LSTM, CNN, Blender) due to its adaptive learning rate capabilities. The model was monitored and stopped when it to overfit with the use of 2 callbacks: Early Stopping, which monitored val_loss with a patience of 3 to stop training when generalization ceased to improve, restoring the best weights automatically, and Learning Rate Scheduler, where we used ReduceLROnPlateau to lower the learning rate by a factor of 0.2 if validation loss stagnated for 1 epoch.

Binary Cross-Entropy was the objective function for the LSTM, CNN, and Neural Blender, suitable for binary classification outputs. Log Loss was used as the evaluation metric for XGBoost and Naive Bayes. The primary metric for all models was Accuracy, with a secondary metric of validation/test loss.

The robustness of the model was tested with 2 validation strategies, hold-out validation and 5 fold cross-validation. Hold out validation, with the standard validation set (X_valid, y_valid) was used to test the individual base learners and to monitor overfitting, while cross validation was used via KerasClassifier and cross_val_score to ensure model stability across different data subsets.

From there, two levels of dropout were applied to prevent overfitting in the models, with .5 dropout for dense, fully connected, layers, and .2 dropout applied internally to LSTM sequences.

Finally, the 3 primary learning techniques utilized were sequence modeling, feature extraction, and multi-scale learning. For sequence learning, the LSTM used bidirectional learning to capture context from both past and future words simultaneously. Then, global max pooling was used in both the LSTM and CNN architecture to find strong sarcasm indicators throughout the text, and finally the CNN used parallel convolutional branches with kernel sizes of 3, 5, and 7 to capture different sized n-grams.

6. Experiments & Results

The baseline model performed poorly compared to the new model with a validation accuracy of 0.753. At first this model was included in the final stack result but was later removed since it hurt the accuracy by 0.02 suggesting that it was sending mixed signals to the meta learner. Adding additional features such as the number of punctuation in the sentence(!, ?, or “”) did not improve or add anything positive to the models.

Stacked Model Performance

Class	Precision	Recall	F1-Score	Accuracy
0	0.9140	0.9087	0.9113	-
1	0.8916	0.8977	0.8947	-
Overall	-	-	-	0.9037
Macro Avg	0.9028	0.9032	0.9030	-
Weighted Avg	0.9038	0.9037	0.9038	-

The model shows good results on the test set, resulting in 90.37% accuracy. It is a balanced model as observed by the macro average vs weighted average. The performance metric for class 0 is slightly better due to its higher representation in the dataset.

Confusion Matrix

	Predicted: 0	Predicted: 1
Actual: 0	478 (True Negative)	48 (False Positive)
Actual: 1	45 (False Negative)	395 (True Positive)

The model shows a very balanced performance. The false positive rate ($48/526 \approx 9\%$) and false negative rate ($45/440 \approx 10\%$) are nearly identical, indicating the model is not biased toward either class.

7. Discussion

Adding a CNN to the LSTM and combining into the final stacked model worked very well due to both models complementing each other on finding different patterns. Adding more models to the stack did not improve performance by a significant amount and in some cases even hurt model performance, and utilizing too many models on our limited data had a tendency to cause overfitting.

Furthermore, adding more features had no impact due to either the feature being useless since the models already catching the pattern or the signal itself being bad. It was also difficult to develop other features due to the input dataframe being stripped of capitalization and ending punctuation, which is incredibly important when running sentiment analysis for sarcasm detection. In addition, the text utilized for training and testing did not utilize emojis, which modern sarcastic text has a tendency to use.

Some significant improvement might result from using higher quality embedding or training our own embeddings on a larger set of data. In our model, we tried and chose not to use uncertainty sampling because while it did work to a degree, getting approximately 87.5% accuracy on the validation set, we determined that the training dataset was too small for uncertainty sampling to be viable, since directly using the entire training dataset gave similar accuracy rates while not having the overtraining issues that uncertainty sampling did. With a larger training dataset, uncertainty sampling would become a lot more viable.

Another significant improvement would come from using a transformer based architecture since they are better at catching patterns with the use of self attention. Another benefit of the structure is that we could train on a more modern and representative dataset to pick up more embeddings, instead of using a random vector for unknown words.

8. Conclusion

For this project, we developed a successful model which consists of a stacked ensemble model combining a Bidirectional LSTM and a Multi-Channel CNN fed into an XGBoost meta-learner. This model resulted in 90.37% accuracy, significantly outperforming the Naive Bayes implementation. This decently high result suggests that both memory and context is required for successful prediction of sarcasm.

Over the course of this project, we have discovered that it is important to either have large quantities of data with a highly complex model, or to use multiple models testing different patterns when attempting to predict sarcasm. Combining sequence-based (LSTM) and feature-based (CNN) architectures yielded better performance than either model individually by a significant margin. In addition, pre-trained word embeddings were superior to TF-IDF allowing the model to detect better patterns. As for the other models that were attempted, further testing showed that introducing models that provide weak context did not help overall accuracy, and may even reduce accuracy. Finally, an overview of the project indicated that the removal of punctuation and capitalization in the dataset likely hindered higher accuracy, as these are key markers of sarcastic tone.

9. References

Libraries used:

- pandas
- numpy
- spacy
- gensim
- tensorflow
- xgboost
- scikit-learn<1.6

This model, sourced from Gensim, was used to provide the embeddings for our data.

- glove-wiki-gigaword-100

Research done using Canvas Slides