Sarcasm Detection - A Comparison Into How Linguistic Categorization and Modeling Affects Sarcastic Classifications

Mark Biegel and Youssef Othman

Abstract

This paper details language processing methodology for classifying sarcasm within Twitter and Reddit posts on the internet. With an increase presence in social media, people are able to speak their mind and leave comments and opinions on others' posts. This leads to the dilemma of having posts containing sarcasm. Machine learning in conjunction with natural language processing techniques can be useful for determining and filtering out sarcastic commentary on these platforms. An experimentation is implemented to test different techniques, starting with pre-processing on various datasets and then implementation of four word-featurizers in three supervised models from Scikit-Learn. Area under the curve of the ROC graph is used to directly compare each featurizer head-to-head to determine which combination of vectorizer and model is best. Overall, the research and experimentation shows that Scikit-Learn's CountVectorizer featurization method used in a tuned Stochastic Gradient Descent model performs the best with classifying sarcasm from Twitter and Reddit posts.

1 Introduction

Twitter and Reddit have implemented methods to reduce cyber-bullying and make their platforms a friendly and engaging environment. Unfortunately, sarcastic responses can be undetectable by current methods; machine learning in combination with natural language processing techniques might be able to improve detection rate.

1.1 Defining Sarcasm

Sarcasm is a way for people to express their viewpoint by introducing negative remarks in opinionated commentary. Many factors should be recognized in sarcastic statements: specific punctuation, emojis, word choice, word sentiment, and pronunciation; all aspects are crucial to portraying a perspective using sarcasm (Zhang et al., 2016). When conversing with someone face-to-face, tone and facial expressions "...[point] out the intent of the speaker, whereas in textual communication, sarcasm is inherently ambiguous, and its identification and interpretation may be challenging even for humans" (Peled and Reichart, 2017). Overall, sarcasm is subjective and up to interpretation by each person, so a universal definition is hard to determine; having computers detect it requires numerous techniques and methods for an accurate implementation.

2 Purpose

2.1 Motivation

Sarcasm has become a staple of everyday life. People can come up with sarcastic quips that are funny, smart, and probably insulting to one person or another. Sarcasm, however, can mislead someone from the main point of an argument. Having a systematic and reliable way to determine sarcasm could be helpful with relieving tension on social media and make it a more friendly environment.

2.2 Fitting in with previous work

Numerous studies have touched on sarcasm detection, using a wide variety of techniques such as neural networks; however, few have explored the capabilities baseline featurizers from frameworks like Scikit-Learn offer. There seems to exists a knowledge gap in experimentation with how well these baseline linguistic featurizers are able to detect sarcasm. While neural models could very well be superior to any baseline featurizer, understanding how the baseline featurizers stack up against these other models could provide crucial information to show the importance of the more advanced neural networks for sarcasm detection.

This paper acknowledges the results from two studies using neural networks in order to compare them to the results of this paper's baseline feature experimentation.

2.2.1 Custom Neural Network Study

This study implements a custom neural network model to use sentiment analysis on sarcastic tweets. With their own datasets, this study shows an impressive 93% accuracy in sarcasm detection (Zhang et al., 2016).

2.2.2 Public Recurrent Neural Network Study

This other study implements a recurrent neural network. Using pre-defined datasets and a publicly available RNN, the study was able to achieve a maximum of 70% accuracy in their experimentation (Peled and Reichart, 2017).

2.2.3 Overall Goal

The purpose of this paper and experimentation is to shed light on the abilities of baseline featurizers. It utilizes experimentation to determine the best featurizer-and-model combination in order to be compared against these more-advanced models and ascertain the relevancy of the defined featurizes in sarcasm detection.

3 Methodology

There are numerous ways to find the best combination of featurizer and model. This experimentation utilizes four featurizers tested on three supervised models; these tools are chosen based on familiarity, popularity, and accessibility (through an open-sourced framework). The Scikit-Learning framework is solely used in this experimentation due to the collective experience the researchers have with it as well as its ease-of-use to novice machine learners.

3.1 Featurizers and Vectorizers

Since machine learning models can only process numerical data, words must be transformed into numerical representations. The following four featurizers and vectorizers are different approaches to numerically transforming categorical corpus in a systematic manner.

3.1.1 CountVectorizer

A CountVectorizer is one of the most basic methodologies for quantifying words in machine learning classification. It simply keeps track of the number of times a word is seen in a key-value format (Kumar et al., 2020) and convert the text into a vector format to be used in a model (Moeed et al., 2022).

Because of its simple implementation and effectiveness in model classification, CountVectorizer is included in the experimentation.

3.1.2 OneHotEncoder

OneHotEncoder is a binary featurizer that is useful for classification problems such as determining sarcasm in sentences. The model produces resulting vectors where each vector contains a "1" value at a specific index of the vector with every other indices containing zeros; thus, it gets its name of "one" from having only one bit in each vector true at a time. The purpose of OneHotEncoder is to create a representation for each unique word in the corpus; this allows the model to recognize patterns with certain vectors that represent individual words (Enireddy et al., 2022).

Because of its usefulness in classifications problems, OneHotEncoder has been chosen in the experimentation, as it is a more simple implementation of vectorization and is very popular.

3.1.3 TfidfVectorizer

TfidfVectorizer is a unique featurizer that goes a step further in terms of what is being vectorized in a CountVectorizer. A TfidfVectorizer attaches weights to words, allowing more meaningful words to have a higher weight-value while less important words get a lower weight-value. This is advantageous in word classification because unimportant words can skew how the model learns. For example, sentences that contain unimportant words like "at", "the", "of", and "that" don't provide much information towards sarcasm detection, so a TfidfVectorizer weighs these words less than other, more important words. Ideally, a TfidfVectorizer should perform better than a OneHotEncoder since it contains this better-suited language processing technique. (Paper and Paper, 2020)

3.1.4 HashingVectorizer

HashingVectorizer is similar to how a CountVectorizer operates; however, it stores the hash of each word along with a numerical value instead of storing the actual word in a key-value pair. The major benefit of this featurizer is to decrease runtime and memory used during computation. (Kumar et al., 2020)

This is chosen for the experiment to see if the HashingVectorizer differs in terms of accuracy and precision metrics compared to a CountVectorizer.

3.2 Supervised Learning Models

In order to showcase the efficacy of the four chosen featurizers, three supervised learning models are implemented based on popularity, familiarity, and classification-capabilities. The three models are Multinomial Naive Bayes, Stochastic Gradient Descent, and Logistic Regression. Multinomial Naive Bayes is useful in natural language processing classification, as it works well with classifying discrete, independent features. Stochastic Gradient Descent uses the gradient descent method of minimizing loss in classification. Logistic Regression uses logarithm as a regression function to model dependent variables based on probabilistic outcomes. Overall, all three models excel in classification.

3.3 Data sets

For any supervised machine learning task, access to large amounts of pre-processed and generalized data is important. A dataset that reflects real-world scenarios is ideal, as it reproduces what is experienced in the world. While a perfect dataset does not exist, a comprehensive corpus of Twitter and Reddit posts over various years will suffice for this experimentation. The dataset needs to be large and broad enough to avoid over-fitting the models and skewing the final results.

To create robust input data for the featurizers and the models, the experiment includes four datasets from Kaggle® search engine. The following four datasets are cleaned and then compiled together to make one large dataset to be split into individual training (80% of total dataset), development(10% of total dataset), and test sets (10% of total dataset):

- Twitter Set 1 - Uncleaned Sarcasm Tweets
- Twitter Set 2 - Cleaned Sarcasm Tweets
- Reddit Sarcasm Set
- Twitter News Headlines Sarcasm Set

In total, there are over one million different phrases from Twitter and Reddit in the final dataset which is plenty for generalization and for training the models. Once they were all read into a dataframe with their different column names, the comment and sarcastic classification columns were extracted and concatenated into one large dataframe that was consistent and could be used by the model.

3.4 Model Comparison Metric

Many metrics, such as accuracy, precision, and recall can be calculated to evaluate model performance; however, since the models are being compared to each other, a single, comprehensive metric needs to be used for consistency. The area under the receiver operating characteristic curve (ROC AUC) will be used for comparison. An ROC curve plots the true positive and false positive rates against each other in order compare and contrast a model's ability to correctly classify instances. When looking at the AUC of an ROC curve, a higher value is desired, indicating more area under the curve which translates to better classification between true positives and false positives. In this experiment, an ROC AUC score of 0.75 or higher is set as the target goal, as it is higher than being a random probabilistic event (an ROC AUC score of 0.5) and shows a significant consistency to classify sarcasm.

4 Experimentation

In order to provide a consistent experiment to determining the effectiveness of four featurizers on three supervised models, a systematic test bed is designed to thoroughly test the featurizers as well as chosen hyper-parameters of each model to find the best-in-class featurizer-model combination.

The experimentation was completed in a Jupyter Notebook for efficient data analysis, model manipulation, and debugging issues that occurred during development. Below is a link to a public Github repository containing the Jupyter Notebook experimentation:

Sarcasm Classification Comparison - GitHub

4.1 Process

With a defined test bed and input data, the Scikit-learn models and vectorizers were trained, tuned, and tested to determine effectiveness and a best-in-class classification on sarcasm detection for Twitter and Reddit posts.

4.1.1 Data Pre-Processing

Data sets from multiple sources were used for better generalization; however, each dataset had wildly different layouts, so big inconsistencies existed such that using regular parsing would've been inefficient and less effective. A decision was made to use 'Pandas' built-in function to read in dataframe; however, an issue with this is that some datasets

were in the CSV format while others were JSON format. Nonetheless, different functions simplified the data cleaning process. Once the data was in, the columns containing the sarcastic post or tweet and the sarcasm classification were selected to be used in the final dataset. Most datasets had more information than what the experimentation required, so these columns were not used to make processing faster. Thus, the final dataset is a single dataframe with the 2 columns containing the sarcastic post and the pre-determined label.

4.1.2 Feature Extraction

Using Scikit-Learn, a Count Vectorizer, One Hot Encoder, Tfidf Vectorizer, and Hashing vectorizer were implemented. A set of train, development, and test vectors are created by using the fit_transform() function for each vectorizer based on the tweet column from the dataframe. This allowed for each vectorizer to have vectors encoded for their specific implementation.

4.1.3 Training Models

Three models were implemented to evaluate the vectorizers. In order to find the best-in-class model and featurizer combination, the models' hyper-parameters were tuned via a systematic, iterative approach where each vectorizer ran on a specific combination of hyper-parameters in the model. Using the development set, all four vectorizers were tested on every combination of hyper-parameters, enabling a development ROC AUC score for each vectorizer in the specific model.

Once the best-in-class hyper-parameter combination was found for each vectorizer of each model, the parameter combination was ran on the test set, creating a test ROC AUC score for each vectorizer; this was stored in a dictionary so that all ROC AUC scores for each vectorizer in each model could be compared at the end. With this dictionary, the best featurizer for each model can be observed, and the best model and featurizer overall can be awarded.

4.2 Results and Discussion

With a consistent and systematic test bed, the metrics for each featurizer and model were generated, observed, and analyzed.

4.2.1 Scores

The ROC AUC scores for each model and vectorizer completed on the test set are tallied in **Table 1**.

	MultiNB	**SGD**	**Logistic**
CountVectorizer	**0.6243**	**0.6395**	**0.6330**
TfidfVectorizer	0.6158	0.6360	0.6298
OneHotEncoder	0.5084	0.5105	0.5083
Hash Vectorizer	N/A	0.5133	0.5149

Table 1: ROC AUC scores for each vectorizer and model combination

The data showcases that using a CountVectorizer in combination with a Stochastic Gradient Descent Model results in the best classification-ability to detect sarcasm in Twitter and Reddit posts. Nonetheless, the CountVectorizer performed the best across all three model implementations. It must be noted, however, that the highest ROC AUC score was only 0.6395, which is over 10% less than the target score of 0.75 for this experiment; there are many reasons and variables as to why the maximum score was poor.

4.2.2 Stochastic Gradient Descent

The Stochastic Gradient Descent's best-in-class vectorizer was the CountVectorizer, scoring an ROC AUC of 0.6395. This model in combination with a CountVectorizer ranked first out of the three model combinations tested; however, it only performed slightly better than the other two models' best-in-class combinations. An ROC AUC score of 0.6395 is closer to a random classifying score of 0.5 than it is to the target score of 0.75, indicating that the best result in this experiment shows a poor ability at sarcasm classification. Figure 1 shows the SGD ROC Curves for each of the four vectorizers where the CountVectorizer and Tfidfvectorizer handily outperform the OneHotEnocder and HashingVectorizer:

4.2.3 Logistic Regression

The Stochastic Gradient Descent's best-in-class vectorizer was the CountVectorizer, scoring an ROC AUC of 0.6330. This model in combination with a CountVectorizer ranked second out of the three model combinations tested. This model also did not meet the target threshold of 0.75 for ROC AUC score. Being ranked second, this model and vectorizer combination is around 0.5% off from the SGD best-in-class model, showcasing very similar performance between the two. Figure 2 shows the Logistic Regression ROC Curves for each of the four vectorizers where, again, the CountVectorizer and Tfidfvectorizer outperform the others:

Figure 1: **Stochastic Gradient Descent ROC Curve**

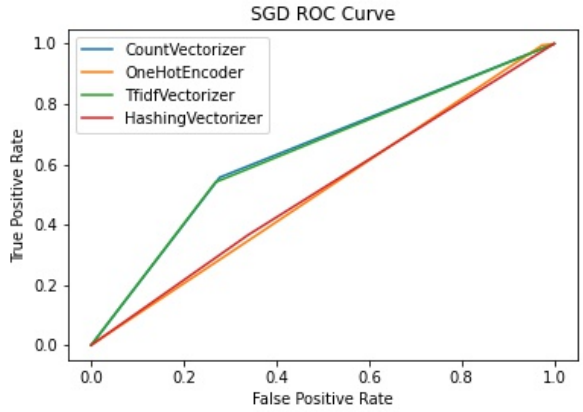
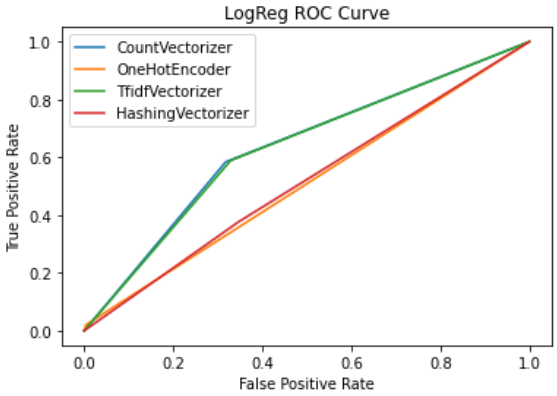


Figure 2: **Logistic Regression ROC Curve**



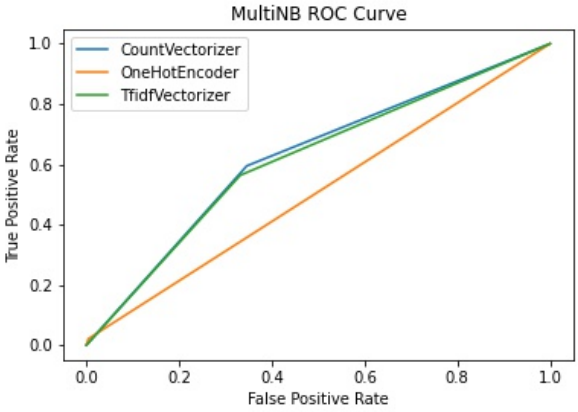
4.2.4 Multinomial Naive Bayes

The Multinomial Naive Bayes' best-in-class vectorizer was the CountVectorizer, scoring an ROC AUC of 0.6243. This model in combination with a CountVectorizer ranked third out of the three model combinations tested. Again, the ROC AUC score did not meet the target score of 0.75. While it ranked third, its ROC AUC score is around 1% off from the other two models which could lie within a margin of error. Furthermore, it should be noted that due to model incompatibilities, the hash vectorizer was not used on this model, so there are no statistics available. Figure 3 shows the Multinomial Naive Bayes ROC Curves for each of the three vectorizers where the CountVectorizer and Tfidfvectorizer outperform the OneHotEnocder:

4.3 Analysis

After hyper-parameter tuning and iterative combinations, each best-in-class model performed similarly to one another. Within each model, the

Figure 3: **Multinomial Naive Bayes ROC Curve**



CountVectorizer performed the best with Tfidfvectorizer following suit being within less than 1% in each instance. OneHotEncoder and HashVectorizer performed very poorly, both barely reaching above a random classification ROC AUC score of 0.5.

The CountVectorizer and TfidfVectorizer perform the best; however, they both resulted in low ROC AUC scores, signifying a poor ability in sarcasm classification. This could be a result of how the data was cleaned, how many data entries were used, over-fitting or under-fitting problems, as well as general incapabilities both models have with linguistic processing.

A reason for the poor results from OneHotEncoder may be due to emoticons within the training data. Because a OneHotEncoder creates a vector where a single entry is set to a value of 1, having emoticons in both sarcastic and non-sarcastic commentary may have made it difficult for models to find a pattern in this dataset, possibly misleading each model's classification.

Nonetheless, the HashVectorizer performed very poorly as well even though it uses a similar algorithm as the CountVectorizer. A HashVectorizer has better memory efficiency as long as there are no hash collision; however, if multiple posts happen to get the same hash value, this could skew the efficacy of the a model's ability to classify sarcasm within a sentence. On the other hand, the way the data is pre-processed in this experiment could be another leading factor as to why this featurization method performed poorly. Nonetheless, it is intriguing that this featurizer performed much worse than its counterpart.

Overall, the results shed additional light in a

specific field that has not been experimented in, showcasing how certain Sklearn models perform better than others as well as the overall poor classification of the generic Sklearn models in this specific implementation.

4.4 Experimental Limitations and Improvements

There were over a million data points in the dataset. The primary issue with having so much data was that we just assumed the pre-determined labels for training data were correct since we did not have time to go through and check each one. This isn't an issue with just our experimentation; it is an issue with all large pre-labeled data. The other major problem was that the term 'sarcasm' is a bit subjective, and having data from multiple sources means there will definitely inconsistencies. To improve on this, we would comb through the data ourselves and label it consistently, that way we know that all the information we are feeding our model is correct. Furthermore, the next big limitation was computing power for the experiment. Google Colab was limiting our runtime and length, we were unable to keep it running for a long time without being kicked out, and it was just pretty slow overall, limiting the number of vectorizers we ended up using.

4.5 Follow-up Work

As mentioned previously, we believe poor training data labeling could attribute subpar performance. Other than going through the data and making sure it is correctly labeled ourselves, we don't have too many options. Ideally, we would feed the model the same amount of sarcastic vs not sarcastic texts, that way our data is not skewed one way or the other. We believe doing that would impact our scores as it removes that chance of having a default class. Addressing the computing power limitation, we would ideally option for a subscription to Google Colab in order to run all training and testing on GPUs to speed up the process. We could even look into using Pytorch, as it is a much more mainstream and efficient library than Sklearn.

5 Conclusion

This paper outlines research and experimentation for determining the best Scikit-learn model and featurizer to classify sarcasm posts among Twitter and Reddit. Four featurizers and three models were implemented in a small experiment to observe generic featurization implemented into tuned models. The models' hyper-parameters were systematically tuned using a training set, and they were evaluated by predicting on a development and test set to find the best-in-class combination of featurizer and model. The best-in-class model and featurizer is a Stochastic Gradient Descent in with a CountVectorizer, achieving an ROC AUC score of 0.6395. While this combination was the best in the experiment, a score of 0.6395 is well below the target 0.75 score for this experiment, illustrating that the best-in-class model of this experiment is poor at sarcasm classification on this dataset. With hardware limitations and naive pre-processing methodology, we hope this work can be followed up with a more efficient dataset as well as looking at better featurizers and models (such as neural) models to increase classification capabilities.

In Conclusion, there is a lot left to be desired. Scikit-Learn's featurizer and models are a great way for beginners to understand natural language processing; however, in a small sample, they reveal poor classification-ability with sarcasm. Nonetheless, data pre-processing could be considered an even more important aspect than the chosen models or featurizers, as how the data is manipulated can greatly influence overall model performance. In the end, more-advanced natural language processing and machine learning models should be implemented to accurately determine sarcastic posts as it continues plagues online social media platforms.

References

Vamsidhar Enireddy, C Karthikeyan, and D Vijendra Babu. 2022. Onehotencoding and lstm-based deep learning models for protein secondary structure prediction. *Soft Computing*, 26(8):3825–3836.

Nripesh Kumar, Akash Harikrishnan, and Rajeswari Sridhar. 2020. Hash vectorizer based movie genre identification. In *Proceedings of ICETIT 2019*, pages 798–804. Springer.

Syed Abdul Moeed, Govindhula Sai Sangeetha, Mohammad Hafsa, Kota Manoj, Nuneti Prahaneetha, and Thopucherla Sujith. 2022. Polarity prediction on product reviews based on classification algorithms. In *2022 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, pages 1–6.

David Paper and David Paper. 2020. Classification from complex training sets. *Hands-on Scikit-Learn*

for Machine Learning Applications: Data Science Fundamentals with Python, pages 71–104.

Lotem Peled and Roi Reichart. 2017. Sarcasm sign: Interpreting sarcasm with sentiment based monolingual machine translation. *arXiv preprint arXiv:1704.06836*.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2016. Tweet sarcasm detection using deep neural network. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2449–2460, Osaka, Japan. The COLING 2016 Organizing Committee.