Sarcasm Detection

Devashish Kamble

June 7, 2023

Abstract

Sarcasm, a form of communication that conveys meaning opposite to the literal interpretation of words, poses challenges for natural language processing tasks. In order to correctly understand people's true intention, being able to detect sarcasm is critical. The objective of this project was to explore and compare different machine learning approaches for effectively identifying and classifying sarcastic statements in textual data. I show that in comparison to traditional classifiers such as Support Vector Machines (SVM), Naive Bayes, and Random Forests; a simple BERT, short for Bidirectional Encoder Representations from Transformers [Devlin et al., 2018], classifier provides surprisingly good results for this task.

1 Introduction

Sarcasm, a prevalent form of linguistic expression, presents a unique challenge for natural language processing tasks due to its inherent irony and subtlety. Sarcasm involves the deliberate use of words to convey meanings opposite to their literal interpretation, making it difficult for machines to accurately comprehend and interpret such statements. Despite its widespread use in everyday communication, the detection of sarcasm remains a complex task for automated systems.

The objective of this project is to explore and compare different machine learning techniques for sarcasm detection in textual data. Detecting sarcasm is crucial in various domains, including sentiment analysis, social media analysis, and online content moderation. By developing effective sarcasm detection models, we can enhance the accuracy of sentiment analysis systems, improve customer feedback processing, and facilitate more nuanced understanding of textual data.

In this report, I present a comprehensive investigation into sarcasm detection using various machine learning approaches. The research involves the utilization of a carefully curated dataset containing a diverse collection of sarcastic and non-sarcastic statements sourced from news articles. I then employ a range of machine learning techniques, including traditional classifiers such as Support Vector Machines (SVM), Naive Bayes, and Random Forests, as well as more advanced Transformers-based model like BERT. These algorithms leverage different features extracted from the textual data, including lexical, syntactic, and contextual information, to discern patterns and identify sarcastic statements. The performance of each algorithm is evaluated using various evaluation metrics, including accuracy, precision, recall, and F1-score.

Through this research, I aim to contribute to the advancement of sarcasm detection in natural language processing. By identifying the most effective machine learning techniques and features for sarcasm detection, we can enhance the ability of automated systems to understand and interpret sarcastic statements. Additionally, this research provides insights into the challenges and limitations associated with sarcasm detection, including the influence of cultural and contextual factors on sarcasm interpretation.

The remainder of this report is structured as follows: Section 2 provides a comprehensive review of related work and prior research in the field of sarcasm detection. Section 3 details the methodology employed, including dataset collection, preprocessing steps, and the implementation of different machine learning techniques. Section 4 presents the experimental results and comparative analyses of the performance of each approach. Section 5 discusses the findings, limitations, and challenges encountered during the project. Finally, Section 6 concludes the report with recommendations for future research directions in sarcasm detection, emphasizing the potential for multimodal approaches and the incorporation of additional contextual cues for more robust sarcasm detection models.

2 Related Work

One of the first analyses has been made by Tepperman et al. (2006) [13] who aimed to recognize sarcasm in speech using prosodic, spectral, and contextual cues. Their research attracted the interest of many sentiment analysis experts. The first investigations made on text were focused on discovering lexical indicators and syntactic cues that could be used as features for sarcasm detection. Carvalho et al. (2009) [1] found that oral and gestural expressions such as emoticons and other keyboard characters are more predictive of sarcasm. Tsur et al. (2010) [14] found that exclamations like *yay!* or *great!* are good indicators on Amazon product reviews, and Davidov et al. (2010) [2] exploited syntactic patterns such as sarcasm hashtags to train classifiers using a semi-supervised technique.

Gonzalez-Ibanez et al. (2011) [4] performed Support Vector Machines and logistic regression on tweets using also emoticons and sarcastic hashtags as features. They also found positive and negative emotions to be strongly correlated with sarcasm. This theory was deepened by Riloff et al. (2013) [11] which developed a bootstrapping algorithm based on the opinion that sarcasm consists on a contrast between Positive Sentiment and a Negative Situation (e.g. *I love being ignored*). While the previous works focused on unigrams, Lukin and Walker (2017) [8] proposed a bootstrapping method too, based on the expansion of sarcastic N-grams cues, such as *no way*, *oh really*, etc.

However, sarcasm can not be considered only as a purely lexical and syntactic phenomenon. In fact, Wallace et al. (2014) [15] showed that many of the classifiers previously described fail when dealing with sentences where context is needed. Joshi et al. (2016) [6] proposed different kinds of word embeddings (Word2Vec, GloVe, LSA), augmented with other features on word vector-based similarity, to apprehend context in phrases with no sentiment words. Ghosh and Veale (2016) [3] proposed a concatenation of Convolutional Neural Network, Long-Short Term Memory Network and Deep Neural Network (CNN-LSTM-DNN) that outperformed many state-of-art methods based on text features. Hazarika et al. (2018) [5] proposed a framework able to detect contextual information with user embedding created through user profiling and discourse modeling from comments on Reddit.

In conclusion, prior research in sarcasm detection has witnessed a shift from rule-based and linguistic heuristics to machine learning approaches, particularly deep learning models. The utilization of RNNs, Transformers, and large pre-trained language models has shown promise in capturing the complex nature of sarcasm. The exploration of cross-lingual sarcasm detection and the development of robust evaluation metrics have further contributed to the advancement of the field. However, challenges such as contextual variations and cultural differences remain areas for future investigation. The present study aims to build upon prior research by exploring and comparing different machine learning techniques for sarcasm detection.

3 Data

3.1 Data Acquitsition

The **News Headlines** dataset for Sarcasm Detection [9] 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. 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. Furthermore, 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. Each record consists of three attributes:

1. $is\_sarcastic$: 1 if the record is sarcastic otherwise 0
2. $headline$: the headline of the news article
3. $article\_link$: link to the original news article. Useful in collecting supplementary data

3.2 Data Preprocessing

The data preprocessing phase involved cleaning and merging both datasets. The News Headlines dataset contained 55,328 headlines, including 29,970 neutral headlines from HuffPost and 25,358 satirical headlines from The Onion. Contractions are words or combinations of words that are shortened by dropping letters and replacing them by an apostrophe. In English contractions, we often drop the vowels from a word to form the contractions. The contractions are thus removed as the words play an important role in sentiment analysis. The data already contained sarcasm labels, so no preprocessing was necessary. The other preprocessing steps are explained below:

Check out for null values - Presence of null values in the dataset leads to inaccurate predictions. There are two approaches to handle this:

1. Delete that particular row - We will be using this method to handle null values.
2. Replace the null value with the mean, mode or median value of that column.
    (a) This approach cannot be used as our dataset contains only text.

Tokenization and remove punctuation - Process of splitting the sentences into words and also remove the punctuation in the sentences as they are of no importance for the given specific task.

Case conversion - Converting the case of the dataset to lowercase unless the case of the whole word is in uppercase.

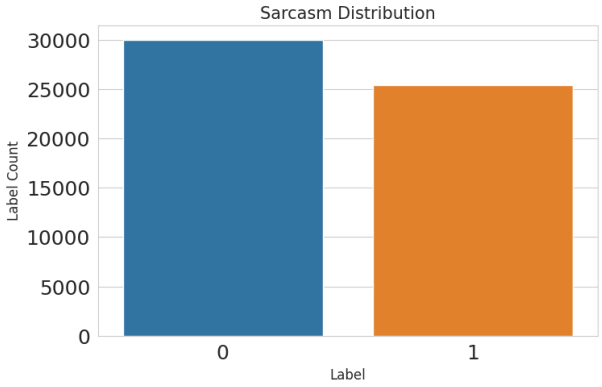
Stopword removal - These words are a set of commonly used words in a language. Some common English stop words are “a”, “the”, “is”, “are” and etc. The main idea behind this procedure is to remove low value information so as to focus on the important information.

Normalization - This is the process of transforming the text into its standard form. For example, the word “gooood” and “gud” can be transformed to “good”, “b4” can be transformed to “before”, “:)” can be transformed to “smile” its canonical form.

Noise removal - Removal of all pieces of text that might interfere with our text analysis phase. This is a highly domain dependent task. For the twitter dataset noise can be all special characters except hashtag.

Stemming - This is the process of converting the words to their root form for easy processing.

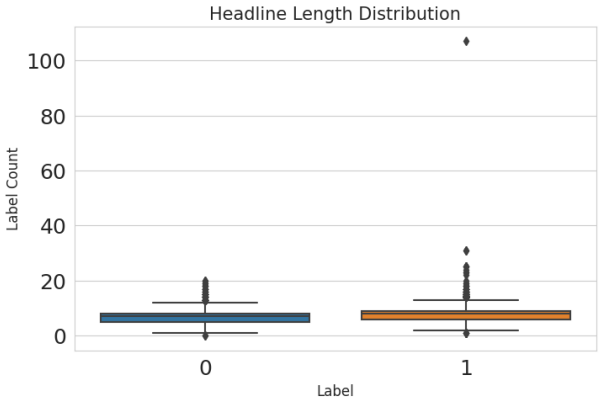
The ratio of the neutral to sarcastic headlines in the dataset is close to 1 (0.85), which means the dataset can be considered to be balanced as it makes training a model easier and helps prevent the model from becoming biased towards one class. This can be visualised in the figure below:



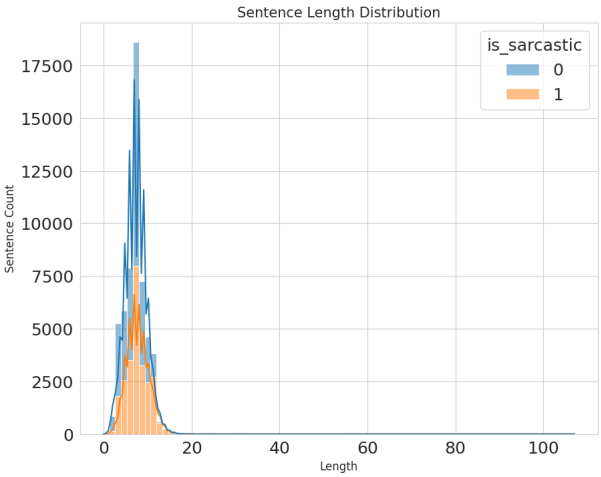
Distribution of Sarcasm Dataset

On cleaning the text in the dataset, I added the two columns holding the sentence length and character count in every sentence. On plotting the histogram and box-plot for sentence length, it is observed that there lies an outlier; a headline with 107 words which is quite unusual for any headline since most of distribution between 20 words. Hence, the outlier was then removed from the dataset.

The *Word Cloud* is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. They are used to quickly identify the most important themes or topics in a large body of text, to understand the overall sentiment or tone of a piece of writing and explore patterns or trends in data that contain textual information. In this project, the wordcloud library is used for generating it in Python. Also, I found the top 15 bigrams within the headlines and plotted them to understand their correlation.



Box-plot of Headline Length in Sarcasm Dataset



Histogram of Headline Length in Sarcasm Dataset

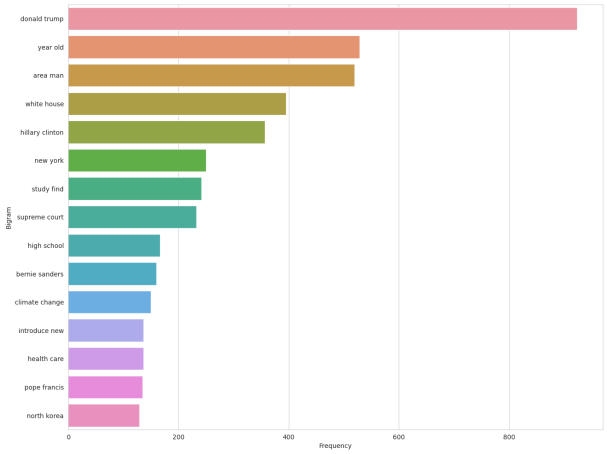
Word Cloud for Sarcastic Headlines

Since the most frequent words in this dataset are about Donald Trump, nation, American, White House, etc. it can be estimated that people tend to make sarcastic sentences about politics.

4 Models

4.1 Random Forest

Random forest is one of the supervised learning algorithms used in machine learning that is used in grouping and regression problems. Whereas the main usage is for the organization scenarios. When talking about a forest, we know it's full of trees and when there are more trees then it's considered to be the completely robust forest. Likewise, random forest algorithm in machine learning develops trees called as the decision trees out of the given examples and then finally predicts the best suited



Bar-plot of top 15 bigrams in Headlines

outcome. It generally dissolves the situation of overfitting by making the combination of the results of various decision trees. Random forest machine learning algorithm also undergo with large range of data products than one individual decision tree algorithm.

4.2 Gaussian Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm that can be used in several classification tasks. Typical applications of Naive Bayes are classification of documents, filtering spam, prediction and so on. This algorithm is based on the discoveries of Thomas Bayes and hence its name. Bayes Theorem tells us the Probability of an event happening (Y) given some observed evidence (X). The name *Naive* is used because the algorithm incorporates features in its model that are independent of each other. Any modifications in the value of one feature do not directly impact the value of any other feature of the algorithm. The main advantage of the Naive Bayes algorithm is that it is a simple yet powerful algorithm. Gaussian distribution, also known as Normal distribution, is a probability distribution commonly used to describe natural phenomena such as height or weight distribution in a population, etc. Gaussian Naive Bayes is a variation of Naive Bayes that assumes that data follows a normal distribution. It is based on the probabilistic model where the algorithm can be coded easily, and predictions did quickly in real-time. Hence this algorithm is the typical choice to solve real-world problems as it can be tuned to respond to user requests instantly.

4.3 Support Vector Machine

Support Vector Machine algorithm is one of the Supervised Learning algorithms, that is used for both organization and regression scenarios. Initially it was used for categorization problems in Machine Learning. The motive of the Support Vector Machine algorithm is to develop decision boundary that isolates n-dimensional space into categories in order that they can easily be placed to new data centre in the perfect class in the upcoming scenarios. Hyper plane is to be considered as the perfect decision boundary. Additional uses of this support vector machine algorithm includes text classification, image categorization and face detection etc.

4.4 K-Nearest Neighbor

The k-nearest neighbors (KNN) algorithm is a data classification method for estimating the likelihood that a data point will become a member of one group or another based on what group the data points nearest to it belong to. The k-nearest neighbor algorithm is a type of supervised machine

learning algorithm used to solve classification and regression problems. However, it's mainly used for classification problems. KNN is a lazy learning and non-parametric algorithm. It is called a lazy learning algorithm or lazy learner because it doesn't perform any training when you supply the training data. Instead, it just stores the data during the training time and doesn't perform any calculations. It doesn't build a model until a query is performed on the dataset. Also it is considered a non-parametric method because it doesn't make any assumptions about the underlying data distribution. Simply put, KNN tries to determine what group a data point belongs to by looking at the data points around it.

4.5 BERT

BERT, which stands for *Bidirectional Encoder Representations from Transformers*, relies on a transformer to learn the contextual relationships between the words in a text. The purpose of BERT is to generate a language representation model. Therefore, an encoder is needed in which an input tokenizer is used. After the encoding process, the tokens, as well as the masks and segments are obtained. Each of these will correspond to an input layer of the network. There are different versions of the BERT model, in this case TinyBERT is used as its small size is quick to train. For the training process, random input masking is applied independently to word pieces. The tokens, as well as the masks and segments obtained after the encoding process, correspond to the inputs of the BERT layer. The output of the BERT layer is then processed by the *tf_op_layer_strided_slice* layer, which performs the extraction of a straded slice of a tensor. Then, a Sigmoid layer with one single unit receives the output of this layer and obtains a probability of an instance being sarcasm.

It often occurs in the field of machine learning, that an algorithm performs incredibly well on the training dataset, but poorly on the test set. This common phenomenon is called overfitting. That is, the model has a high variance, which makes it difficult to generalize well on new data. To avoid overfitting, we apply different strategies such as dropout and early stopping. Dropout (Srivastava et al., 2014) [12] is the standard regularizer for deep neural networks in NLP. This regularization technique involves setting a probability of keeping certain nodes or not. A value between 0 and 1 is specified, which is the fraction of the input units to drop. It has been shown, that a dropout rate of 0.5 is effective in most scenarios (Kim, 2014) [7]. Therefore, there is a probability of 50 % that a node will be removed from the network. This, ultimately, results in a much simpler network that helps to prevent overfitting. Early stopping was also used to prevent the model from overfitting. Early stopping is a method in which an arbitrarily large number of training epochs are specified, and the training process is stopped once the model performance stops improving on the validation dataset. Therefore, loss in the validation dataset was monitored. The last best model is the one that is stored for posterior predictions.

4.6 Training

The models mentioned above are trained in order to predict the class of test samples. Scikit-learn (Pedregosa et al., 2011) [10] was used for training the classifier models. The HugggingFace framework provides us a shortcut to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you time from training a model from scratch. As aforementioned, I choose to fine-tune the TinyBert due to the limited hardware GPU for a classification task using PyTorch.

In order to fine-tune TinyBert for the task, the following hyperparameters were used:

1. *epochs* : 5

2. *batch_size* : 32
3. *max_seq_length* : 128
4. *learning_rate* : $1e-4$
5. *dropout* : 0.5
6. *loss_function* : Cross Entropy Loss
7. *optimizer* : Adam

As the environment to train and test the models, Google Colab was used with GPU activated. Google Colab is a Google Research product that enables running Python code on the browser for free with computational resources, such as GPU. The dataset as well as the code to replicate the experiments can be found in the GitHub repository https://github.com/devashishk99/Sarcasm-Detection.

5 Results

The model performance was evaluated by comparing accuracy, precision, recall, and F1 scores [16] across all the applied classifiers. Accuracy is defined as the ratio of correctly classified instances to the total number of instances.

$$Accuracy = (Correct\_predictions)/(Total\_predictions)$$

However, accuracy alone might not provide a complete picture of the performance. Precision is the ratio of true positive instances to the total number of true positive and false positive instances.

$$Precision = TP/(TP + FP)$$

At the same time, recall is the ratio of actual positive instances to the sum of actual positive and false negative instances.

$$Recall = TP/(TP + FN)$$

The F1-measure is then obtained by using the harmonic mean of precision and recall.

$$F1 = 2 * (Precision * Recall)/(Precision + Recall)$$

Also, the AUC-ROC score is calculated which is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between headlines being neutral or sarcastic.

The table below summarises the final results for each machine learning algorithm employed for the sarcasm detection task:

Model	*Accuracy*	*AUC-ROC*
Random Forest	0.95	0.95
Gaussian Naive Bayes	0.78	0.77
SVM	0.90	0.89
KNN	0.89	0.88
TinyBERT	0.97	0.95

Table 1: Model Performance

6 Discussion

This section discusses the performance of each model, compares their strengths and weaknesses, and explores possible reasons behind the observed results.

BERT, the transformer-based model, emerged as the top-performing model in our experiment for sarcasm detection. This outcome aligns with previous research that highlights the effectiveness of transformer models in capturing contextual information and understanding nuanced language patterns. BERT's ability to learn rich representations and leverage its pre-training on large corpora likely contributed to its superior performance. The model's strong performance suggests that contextual understanding plays a crucial role in accurately identifying sarcastic statements.

Following BERT, the Random Forest algorithm performed well, achieving competitive results in sarcasm detection. Random Forest's ensemble approach, combining multiple decision trees, allows it to capture complex relationships between features. This flexibility likely contributed to its success in identifying sarcastic statements. Random Forest models excel at handling high-dimensional feature spaces, making them well-suited for text classification tasks.

Support Vector Machines (SVM) demonstrated moderate performance in our experiment. SVMs aim to find an optimal hyperplane that separates different classes in the feature space. While SVMs have been widely used for text classification tasks, their performance can be affected by the choice of kernel function which in this case was the Linear kernel and hyperparameters. In the case of sarcasm detection, SVMs may struggle to capture the nuanced language patterns and context required for accurate identification of sarcastic statements.

K-Nearest Neighbors (KNN) exhibited relatively lower performance compared to the other models. KNN operates based on the assumption that similar instances are likely to have similar labels. In the context of sarcasm detection, the local nature of KNN may limit its ability to capture the complex relationships and contextual dependencies needed to accurately classify sarcastic statements. Additionally, the curse of dimensionality can impact KNN's performance, as it relies on distance metrics in high-dimensional spaces.

Gaussian Naive Bayes (GNB) displayed the lowest performance among the models evaluated. Naive Bayes models assume independence between features, which may not hold in the case of sarcasm detection where contextual and linguistic dependencies play a crucial role. GNB's simplicity and fast training speed make it suitable for certain tasks, but its limitations in capturing complex relationships likely affected its performance in sarcasm detection.

Considering the results, it is evident that the ability to capture contextual information and leverage complex language patterns plays a crucial role in sarcasm detection. The transformer-based BERT model outperformed the other models, demonstrating the importance of pre-training on large corpora and fine-tuning for specific tasks. However, it is important to note that the performance of each model can vary depending on the dataset, feature engineering, and hyperparameter tuning.

7 Conclusion

In the present study, we evaluated several machine learning models, including BERT, Random Forest, Support Vector Machines, K-Nearest Neighbors, and Gaussian Naive Bayes, for the task of Sarcasm Detection. Based on our experimental findings, we observed the following:

1. BERT, a transformer-based model, achieved the highest performance among the evaluated models. Its ability to capture contextual information and leverage pre-training on large corpora resulted in accurate identification of sarcastic statements.

2. Random Forest demonstrated competitive performance, leveraging its ensemble approach to capture complex relationships in the feature space.

3. Support Vector Machines exhibited moderate performance, while K-Nearest Neighbors and Gaussian Naive Bayes achieved comparatively lower accuracy in sarcasm detection.

Future research in sarcasm detection could explore techniques to combine the strengths of different models, such as ensemble methods or hybrid architectures. Additionally, investigating the impact of additional features, such as sentiment analysis or linguistic cues, may further enhance the accuracy of sarcasm detection systems. Furthermore, analyzing the misclassified instances and understanding the patterns leading to false positives and false negatives can provide valuable insights for improving the performance of sarcasm detection models.

The rate at which newer, better models are developed these days is quite a big feat. Thus, it would be interesting to put these models to test for the sarcasm detection task. At the same, instead of just using text-based datsets, multiple modality based datasets can be incorporated in order to get the most from the models.

References

[1] Paula Carvalho, Luis Sarmento, Mário Silva, and Eugénio Oliveira. Clues for detecting irony in user-generated contents: Oh...!! it's "so easy" ;-). *International Conference on Information and Knowledge Management, Proceedings*, 11 2009.

[2] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Semi-supervised recognition of sarcasm in Twitter and Amazon. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 107–116, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[3] Aniruddha Ghosh and Tony Veale. Fracking sarcasm using neural network. In *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 161–169, San Diego, California, June 2016. Association for Computational Linguistics.

[4] Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in Twitter: A closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 581–586, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[5] Devamanyu Hazarika, Soujanya Poria, Sruthi Gorantla, Erik Cambria, Roger Zimmermann, and Rada Mihalcea. Cascade: Contextual sarcasm detection in online discussion forums, 2018.

[6] Aditya Joshi, Vaibhav Tripathi, Kevin Patel, Pushpak Bhattacharyya, and Mark Carman. Are word embedding-based features useful for sarcasm detection? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1006–1011, Austin, Texas, November 2016. Association for Computational Linguistics.

[7] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[8] Stephanie Lukin and Marilyn Walker. Really? well. apparently bootstrapping improves the performance of sarcasm and nastiness classifiers for online dialogue. In *Proceedings of the Workshop on Language Analysis in Social Media*, pages 30–40, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[9] Rishabh Misra. News headlines dataset for sarcasm detection, 2022.

[10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[11] Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as contrast between a positive sentiment and negative situation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 704–714, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[13] Joseph Tepperman, David Traum, and Shrikanth Narayanan. Yeah right: Sarcasm recognition for spoken dialogue systems. 09 2006.

[14] Oren Tsur, Dmitry Davidov, and Ari Rappoport. Icwsm - a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. 01 2010.

[15] Byron C. Wallace, Do Kook Choe, Laura Kertz, and Eugene Charniak. Humans require context to infer ironic intent (so computers probably do, too). In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 512–516, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[16] Ethan Zhang and Yi Zhang. *F-Measure*, pages 1–1. 11 2016.