# Using iSarcasm to Investigate Sarcasm Detection

CSE 476 - Honors Contract

Tara Paranjpe
Arizona State University
tparanjp@asu.edu

Github Link to Source Code: https://github.com/tparanjpe/sarcasmDetection

## I. Introduction

### Task Assigned

For the honors contract for this course, I chose to read about and investigate the task of sarcasm detection in the realm of Natural Language Processing. The iSarcasm dataset, created by Oprea et al. [1], was designed to improve the task of sarcasm detection after seeing the poor performance of other traditional datasets. I used this dataset to create a sarcasm detection model of my own. I learned traditional methods of feature extraction, classification model selection, and conclusion derivation from this honors contract exercise.

### General Approach

To approach this task, I began by cloning the iSarcasm Github repository and investigating the datasets. The datasets contained tweet identifiers, so the data needed to be extracted using the Twitter API. Once the datasets were created with textual context for the tweets, common features were extracted. Then, using SMOTE, a synthetic balancing technique, I balanced the dataset, and ran the train and test on 4 different classification models. With the prediction metrics, conclusions were derived and future works going forward were planned.

## II. Inspiration

In a paper written by Oprea et al. [1], I learned about the deficiencies of most traditional datasets with respect to sarcasm detection. The paper investigates the variety between intended and perceived sarcasm, stating that different cultures and individuals categorize sarcasm differently. Because of this, datasets that collect tweets manually can be difficult and do not always encapsulate the true meaning of the tweet (sarcastic or not?).

The paper also looks at datasets where tweets are labeled through a type of algorithm called “distant supervision”. This method grabs sarcastic tweets when they are indicated with the “#sarsam” tagline. This method can produce a lot of data, but it is not always accurate or encompassing everything.

This research paper proposes a new type of dataset, collected straight from each Twitter author directly. Outsourcing the manual collection, the authors asked twitter users to grab their own tweets (written and published by the author themselves) to ensure that manual labeling isn’t incorrect.

They tested their dataset on several different models, like LSTM, CNN, and SIARN. The results from the comparison of traditional models to the iSarcasm dataset showed higher F1-scores overall. They published this iSarcasm dataset to inspire more investigation into new and novel approaches to sarcasm detection.

## III. Model Creation

### Dataset Extraction

After gathering the datasets (in CSV format) from the iSarcasm repository, I investigated the data fields to understand how to use them. The train and test datasets contain the following fields:

```
tweet_id,sarcasm_label,sarcasm_type
```

`Tweet_id` is a unique identifier that Twitter recognizes as a tweet. Since this data is purely a string of numbers and doesn't have any tweet content for sarcasm analysis, I leveraged the Twitter API to extract the data. I applied for Elevated access to the API, and took the time to write a Python script that uses Tweepy, a Python library, to process the responses. Some tweets were unavailable and were removed from the dataset for the model.

The tweet data were retrieved from both the train and test datasets and written back into CSV files with the tweet data in string format. The `sarcasm_type` field was removed in order to keep the data simple and the model less convoluted.

### Feature Extraction

Inspired by the algorithms we learned in the course, I decided to use TF-IDF (term frequency-inverse document frequency) vectorization, count vectorization, and hashing vectorization. These were also algorithms supported and built into Sklearn, which I used for my models. To supplement my use of these algorithms, I researched what these features entailed.

**TF-IDF:** This is a type of feature that looks at the importance of specific words in a corpus. The higher this score, the more important the word to the corpus. With this technique, we evaluate word count and rarity of each word in the corpus.

**CountVectorizer:** This type of tool gets the count, or frequency, of each word in the corpus. This is a great way to evaluate which words are more likely to appear in sarcastic texts versus non-sarcastic texts.

**HashingVectorizer:** This type of processing strategy takes each word and maps it into a hash function, storing the data in a more sparse matrix, not indexed by the word.

Each of these feature extraction algorithms were applied to both the train and test dataset. The X-axis dataset was gathered from these algorithms, and the y-axis GOLD labels were extracted from the sarcasm_label column.

When analyzing the data balance of the labeled dataset, I observed that the distribution of "sarcastic" to "not sarcastic" data labels were wildly unbalanced. When the data is unbalanced, the model becomes biased toward a specific class and doesn't get trained appropriately. Running this through our models gave a wildly high accuracy and very low precision and recall. This clearly meant that the data imbalance was causing issues.

To address this deficit, we leveraged SMOTE, also known as Synthetic Minority Oversampling Technique, to balance the labels. This strategy inserts new rows of data to balance the dataset using a correlation line. The Results section of this report, including accuracy, precision, and recall.

### Machine Learning Models

The task at hand is to run binary classification with supervised learning to detect sarcasm. Using sklearn as the main Python package, I used 4 different machine learning models for analysis: k-Nearest Neighbors, Random Forest, AdaBoost, and Support Vector Machine. To understand and justify these classifier selections, I spent some time understanding how each classifier works. These models were chosen empirically based on my simple understanding of classifiers.

**k-Nearest Neighbors:** This classifier is simple, but important. Each data sample is represented on a correlation line, and each point is close to other similar points, but far away from dissimilar points. As each test input is given the model, it is placed on the correlation line and a result is returned.

**Random Forest:** This is a supervised learning algorithm that creates several decision trees to derive a result. Each tree generated doesn't produce the output, but will average out to the correct (or most likely) result.

**Ada-Boost:** Another word for this classifier is Adaptive Boosting. It is a derivation of Random Forest, where the trees only have one level. At each iteration, the weights are redistributed and classified again.

**Support Vector Machine:** Support Vector Machine uses support vectors that draw correlations closest to the similar data points. It is an efficient model in a high-dimensional space, but is difficult to train.

# IV. Results

| Model | Feature | Accuracy | Precision | Recall |
|---|---|---|---|---|
| KNN | TF-IDF | 0.25436 | 0.166944 | 0.87719 |
| | Count Vectorizer | 0.25 | 0.17476 | 0.94737 |
| | Hashing Vectorizer | 0.25436 | 0.92105 | 0.17241 |
| Random Forest | TF-IDF | 0.74273 | 0.248 | 0.27193 |
| | Count Vectorizer | 0.60465 | 0.46491 | 0.20076 |
| | Hashing Vectorizer | 0.55523 | 0.36842 | 0.15217 |
| Ada-Boost | TF-IDF | 0.75145 | 0.20175 | 0.22330 |
| | Count Vectorizer | 0.63953 | 0.36842 | 0.19266 |
| | Hashing Vectorizer | 0.73983 | 0.25439 | 0.23577 |
| SVM | TF-IDF | 0.79506 | 0.14035 | 0.27119 |
| | Count Vectorizer | 0.67297 | 0.2 | 0.32456 |
| | Hashing Vectorizer | 0.78197 | 0.275 | 0.19298 |

***Table 1*** - This table shows the accuracy, precision, and recall for the 4 different models on 3 different feature extraction algorithms.

It becomes clear from Table 1 that some models and some feature extraction algorithms are more promising than others. It appears that the Random Forest Classifier on CountVectorizer and Hashing Vectorizer produces the best accuracy. As we learned in class, accuracy isn't always the best indicator. So, looking at the precision and recall, we can see that they're not great scores, but better than the majority. This is a promising indicator that additional features and other models with tuned hyperparameters can improve the model. Furthermore, tuning the current models and their hyperparameters could optimize the models farther.

# V. Conclusions

### Overall Learnings

In totality, I learned so much about machine learning, NLP text processing techniques, and data balancing. I found it interesting to mess around with pre-made datasets and learn that data collection is just as important as data processing and feature extraction. It is also exciting to find real-world applications to the techniques we learned in class, like CountVectorizer and TF-IDF.

### Future Works

I hope to carry this work farther and see if I can introduce some neural networks to aid in the prediction and see if I can replicate the models created by the research paper. Additionally, I want to see if there are other text features that can be abstracted, or if concatenating two different features together can overall improve or harm the accuracy.

# VI. References

[1] S. Oprea and W. Magdy, "ISarcasm: A Dataset of Intended Sarcasm," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.