Comparison of Various Machine Learning Algorithms in a Context of Sarcasm Detection

Computer Engineering Department, San Jose State University, San Jose, CA

Liubov Tovbin
liubov.tovbin@sjsu.edu

Asel Salmenbayeva
asel.salmenbayeva@sjsu.edu

Akshata Deo
akshata.deo@sjsu.edu

Sachin Sharma
sachin.sharma@sjsu.edu

Abstract— An abundance of written information in the form of microblogging and online reviews motivates the research on opinion mining and sentiment analysis. However, the presence of sarcastic remarks complicates the opinion mining task. Sarcasm detection is a binary classification problem. The availability of labeled datasets allows an application of supervised machine learning technics. In this paper, we investigate the performance of several binary classifiers, which are the most popular among researchers. The dataset for this project is the most extensive corpus of sarcasm available as of May 2019. For the feature extraction, Bag-of-Words method and Term Frequency-Inverse Document Frequency vectorization are used. The performance evaluation metrics are accuracy, precision, recall, and f1 score. The results show that the Support Vector Machine linear classifier with Bootstrap Aggregation performs the best overall.

Keywords— NLP, sarcasm, opinion mining, sentiment analysis, binary classification, logistic regression, SVM, Decision Tree, bagging, Random Forest

I. Introduction

Sarcasm is a form of expression used of oxymoron or exaggeration for mockery or to convey contempt towards others. Humans utilize various hand gestures, facial expression, heavy tones or voice to express sarcasm. However in textual data (social media tweets, chats, product review), these features are not available, makes sarcasm detection a herculean task.

We have chosen this topic due to growing interest in this field, blistering growth of textual data and need for opinion mining and sentiment analysis in this field.

In this paper, we study popular machine learning techniques (Support Vector Machines, Decision Trees, Logistic Regression, Decision Tree, Ensemble Methods) in the context of sarcasm detection.

The rest of the paper is organized as follows:

- Section 2 → Background/Literature Review of sarcasm detection
- Section 3 → Technical Approach for problem solution
- Section 4 → Feature Selection and Extraction
- Section 5 → Experiments and Results
- Section 6 → Conclusion
- Section 6 → Acknowledgement
- Section 7 → References

II. Related Work

Sarcasm detection can be seen as a binary classification problem. The objective is to determine whether or not a given sentence a sarcastic one. Also, the availability of labeled datasets makes it a supervised learning problem.

The following approaches for sarcasm detection as a supervised binary classification problem are mentioned in the literature [1], [2]:

- *Rule-Based*
- *Decision Tree*
- *Neural Network*
- *SVM*
- *Logistic Regression*
- *Naive Bayes*
- *Bayesian Network*
- *Maximum Entropy*

The popular choice of labeled datasets among researchers is the Amazon or Twitter datasets. Most of the work between the years 2010 and 2016 made use of these two datasets [1], [2], [3]. Sulis et al. [4] use the 12,532 tweets dataset from Twitter. Buschmeier et al. [5] use a dataset of 1,254 Amazon reviews.

In 2017, however, a new labeled dataset from Reddit was made available by researchers from Princeton University. In comparison with Amazon and Twitter datasets, the Reddit dataset has a clear advantage — a size. It contains 1.3 million labeled sarcastic statements [6]. The novelty and the size are the two reasons this dataset was chosen for analysis in this project.

The next section provides an overview of the technical solutions utilized in the presented work.

III. Technical Approach

3.1 Logistic Regression (LR)

Logistic Regression is one of the widely used predictive analysis algorithms. It utilizes the following "soft margin" formula to build a model and define the relationship between the binary dependent variable and one or more independent variables:

$$\theta(s) = \frac{1}{1+exp(-s)}$$

3.2 Support Vector Machine (SVM)

SVM is a binary classification algorithm whose objective is to find the maximum margin hyperplane that distinctly separates the data points.

3.3 Decision Tree (DT)

Decision Tree algorithm utilizes a tree model where each node represents a feature which separates the data so that the information gain from the child nodes would be maximal. It has achieved broad popularity due to its simplicity and easy approach to make decisions.

3.4 Ensemble methods

Ensemble methods combine several classifiers to a single predictive model. This approach allows us to attain several key objectives, such as improved performance and reduced variance while keeping the tradeoff with bias.

- Bootstrap Aggregation ("bagging")
- Random Forest
- XGBoost

Bootstrap Aggregation

Bootstrap Aggregation is the procedure of selecting different "bags" of data for training. Each bag is a replaceable sample from the initial dataset, and the models trained on them are parallelizable. The final hypothesis gets elected by voting, averaging, or another predefined method of choice.

Random Forest

Random Forest is a classification technique which makes use of multiple decision trees to classify a dataset. In this process, the mean or mode of the dataset is taken to select the best among them. Random forest employees randomness in addition to bagging to reduces the variance.

XGBoost

XGBoost stands for eXtreme Gradient Boosting, an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. XGBoost provides a parallel tree boosting to solve complex problems

IV. Data Preparation

4.1 Data Set

The largest sarcasm detection dataset parsed from Reddit[1]. We have chosen Reddit over Twitter and Amazon because it has not been much exploited as that of the other two, according to our research. Only regression has been performed on the Reddit dataset. So, in addition to that we have applied a couple of other models as well, and some ensembling methods(that we will discuss in shortly). Apart from this, the dataset which we have parsed from Reddit is already balanced, that is the number of comments with each label is almost equal. As shown in Figure 1 below, 505,405 comments are non-sarcastic, 505,368 comments are sarcastic, that makes a total of 1,010,773 comments.

```
In [9]:  1  train_df['label'].value_counts()

Out[9]:  0    505405
         1    505368
         Name: label, dtype: int64
```

Figure 1. The number of data in each class

We have downloaded the dataset posted on Kaggle, due to the simplicity of the process. Then we have uploaded and run our kaggle.json file on google *colab* to avoid the slow process of our personal computers. For this purpose, we imported files from google.colab. Then we imported pandas to neatly tabularise the downloaded .csv files. Figure 2 depicts the part of the dataset.

```
In [0]: !rm sarcasm.zip

In [0]: import pandas as pd
        sarcasm_data = pd.read_csv("train-balanced-sarcasm.csv")

In [0]: sarcasm_data.head(10)

Out[0]:
```

	label	comment	author	subreddit	score	ups	downs	date	created_utc	parent_comment
0	0	NC and NH.	Trumpbart	politics	2	-1	-1	2016-10	2016-10-16 23:55:23	Yeah, I get that argument. At this point, I'd ...
1	0	You do know west teams play against west teams...	Shbshb906	nba	-4	-1	-1	2016-11	2016-11-01 00:24:10	The blazers and Mavericks (The wests 5 and 6 s...
2	0	They were underdogs earlier today, but since G...	Creepeth	nfl	3	3	0	2016-09	2016-09-22 21:45:37	They're favored to win.
3	0	This meme isn't funny none of the "new york ni...	icebrotha	BlackPeopleTwitter	-8	-1	-1	2016-10	2016-10-18 21:03:47	deadass don't kill my buzz
4	0	I could use one of those tools.	cush2push	MaddenUltimateTeam	6	-1	-1	2016-12	2016-12-30 17:00:13	Yep can confirm I saw the tool they use for th...

Figure 2. Top 4 rows of the dataset

After checking the number of rows for each column, it came to notice that labels and comments had some difference. This shows that various comments were just blank. So, to reduce the complexity of the problem, we decided to remove all those rows with blank comments. The difference is highlighted in the below snapshot, Figure 3.

```
In [6]:  1  train_df.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 1010826 entries, 0 to 1010825
         Data columns (total 10 columns):
         label           1010826 non-null int64
         comment         1010773 non-null object
         author          1010826 non-null object
         subreddit       1010826 non-null object
         score           1010826 non-null int64
         ups             1010826 non-null int64
         downs           1010826 non-null int64
         date            1010826 non-null object
         created_utc     1010826 non-null object
         parent_comment  1010826 non-null object
         dtypes: int64(4), object(6)
         memory usage: 77.1+ MB
```

Figure 3. The unmatched number of comment and labels

And then we dropped all the columns except label and comments. So now, we have just two columns in our dataset, labels, and comments, As shown in snapshot 4 below.

```
In [5]: 1 train_df.head()
```

Out[5]:

	label	comment
0	0	NC and NH.
1	0	You do know west teams play against west teams...
2	0	They were underdogs earlier today, but since G...
3	0	This meme isn't funny none of the "new york ni...
4	0	I could use one of those tools.

Figure 4. Dataset, cleared from redundant data

4.2 Feature Extraction

Tokenization

The applied tokenization technique is called "*Bag-of-Words*" (BoW). Each distinct word represents a distinct *token*. *TfidfVectorizer* from *Scikit-Learn* library has a built-in tokenizer that utilizes a BoW approach. However, for NLP purposes, the *nltk* library works better. Indeed, we get 167,435 distinct words before versus 131022 after the *nltk* preprocessing. That is almost 22% reduction in the number of tokens.

Since the number of tokens directly translate to the number of features, it should be reduced as much as possible. The feature reduction is achieved by ignoring upper case letters, punctuation, and stop words such as articles, prepositions, conjunctions. The example for the first two comments from dataset shown below in Figure 5.

```
NC and NH.
nc nh

You do know west teams play against west teams more than east teams right?
you know west team play west team east team right
```

Figure 5. Example of tokenization

Vectorization

To apply a classifier, each data entry needs to be represented as a numerical vector. Let us say that the number of tokens in the training set is *N*. Thus, each comment in the corpus gets a representation as a vector of length *N*. Each coordinate in the vector should be a number that corresponds to a certain word in the comment.

The procedure of converting non-numerical features into numerical representation is called *vectorization*. Three popular approaches to vectorization of the BoW model exist:

- *Frequency Encoding*
  Each coordinate in the vector is the count of times that a corresponding word appears in a given comment. For NLP purposes, the frequency encoding might not be the best, since the common words get the most weight, but do not offer much insight.
- *One-Hot Encoding*
  A Boolean vector representation, where each 1 or 0 denote whether or not a corresponding word appears in a given comment.
- *TFIDF Encoding*
  Term Frequency-Inverse Document Frequency normalizes the frequency of a word in one comment with respect to the total frequency in a document. This approach gives more weight to topic-specific words, which seems more logical in our case.

4.3 Train Test Split

Now, we will divide the complete dataset for training and testing purpose. We have used train_test_split from sklearn.model_selection library to randomly split the data. After executing the code, we got 33% of the total 1,010,773, that is 333,555 vectors reserved for training. Vector's length is 131,022 as of the number of features. Now, after vectorizing the test dataset we got 677,218 vectors reserved for testing and vector's length is 167,435, amount of features. The vectorization of the training and testing dataset shown below in Table 1 and 2.

Table 1: Dataset split

Dataset	Number of data points
Train	677,218
Test	333,555

Table 2: Feature sets

Feature set	Number of features
Unigram	131,022
Trigram	6,268,559

V. EXPERIMENT

For the purpose of providing more accurate evaluation, the initial dataset was split into training and testing data so that 33% could be used to evaluate the out-of-sample error. We have applied error measure metrics such as accuracy, precision, recall, and f1 score.

We have tried various algorithms with features extracted with TFIDF vectorizer and One-hot encoder. Since one-hot encoder does not consider the frequency of the words, the results we got from it are much worse - about 50% from each model.

In Table 3, we compare the accuracy we got from each method using unigrams - each word individually and trigrams - a sequence of 3 words. In fact, the range of 1 to 3 words was used for trigrams so even if the performance slightly improved, yet the memory usage, as well as computation time, increased dramatically.

Table 3: Accuracy scores of every model using unigrams and trigrams as features

Model	Unigrams	Trigrams
Bagging (logistic regression)	65.85%	67.83%
Bagging (SVM)	65.47%	68.04%
Random Forest	65.27%	67.52%
XGBoost	64.25%	67.76%

For further investigation, we've plotted training and testing curves over epoch for XGBoost. Epoch corresponds to the number of estimators (trees). XGBoost is the only model such a graph is possible since bagging and random forest are fully parallelizable. Figure 6 describes overfitting we 2000 estimators and learning rate 0.5. This proves that even if in-sample-error is steeply going down out-of-sample error could rise on the other hand.

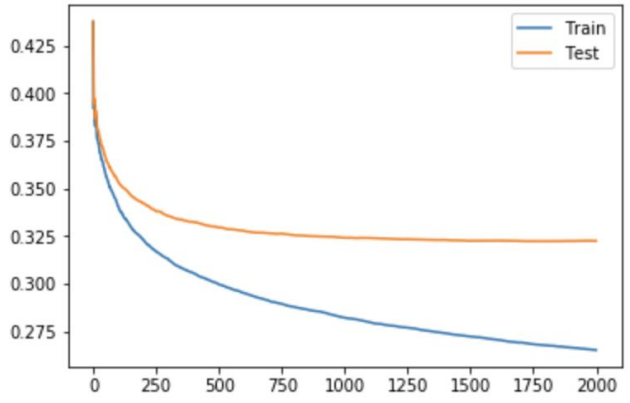

Figure 6. Train and test error curves over 2000 epochs

We tried to avoid overfitting after that with a learning rate of 0.05 and 500 epoch as shown in Figure 7.

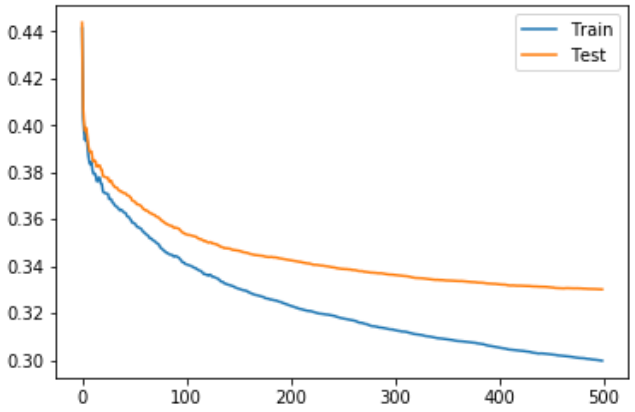

Figure 7. Train and test error over 500 epoch

As the next step, we decided to study precision and recall. Every model outputs quite similar precision-recall curves. Larger recall we have, smaller precision will get. Figure 8 shows normal behavior for precision and recall metrics, thus, the thing to point out here is that maximum precision achieved is 0.7 and the average precision is only 0.62. Since Precision = True Positive / (True Positive + False Positive), that means the fraction sarcasm we have predicted right over all of the comments predicted as sarcasm is still not sufficient and we ended up having some comments falsely classified as sarcasm.

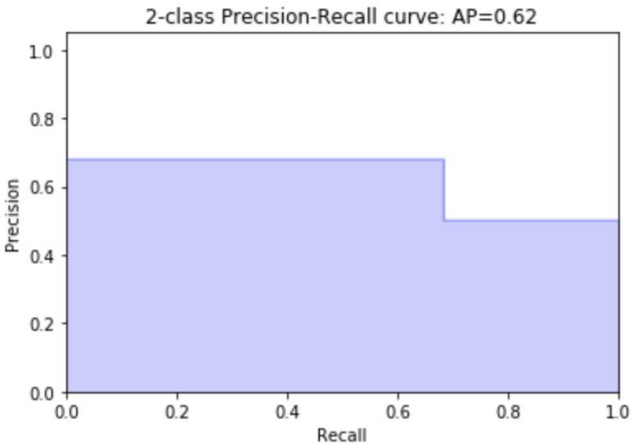

Figure 8. Precision-Recall curve

Finally, in Table 4 we provide the comparison of each precision, recall and f1 scores. The scores for Bagging with SVM are fairly balanced and resulting up with a better f1 score.

Table 4: Precision, recall, f1 scores for each model

Model	Precision	Recall	F1 score
Bagging (logistic regression)	0.69	0.63	0.66
Bagging (SVM)	0.68	0.68	0.68
Random Forest	0.7	0.59	0.64
XGBoost	0.70	0.60	0.64

VI. Conclusion

Research in sarcasm has grown dramatically in the past few years. It is the emerging field in data mining, which requires much deeper insight. We have gone through prominent research articles in this area to explore various techniques for this. In this paper, we focussed on the comparison of the performance of some Machine Learning algorithms on detecting sarcastic comments.

We have achieved the best accuracy and f1 score of 68% using Bagging SVM. However, we have also confirmed that other models are not far behind and have similar scoring. Our study also confirmed that the sequences of words,

trigrams, provide better results than just isolated words, unigrams.

Sarcasm detection is an extensive area of research and there are various difficult challenges that still makes complete sarcasm detection an elusive task. one such challenge is ambiguous definition of sarcasm. The definition of sarcasm is different based on various parameters such as culture, age, etcetera. Other challenge lies in the fact that sarcastic and offensive sentences often only have borderline differences between them. A silly sarcastic comment for one person may be an outright offensive one for the other. All of these nuances make sarcasm detection an interesting area of continuous research.

One of the future enhancement in our project could be to categorize the detected sarcastic comments into various categories such as outright offensive, moderately offensive and benign. One more future enhancement would be to introduce factors such as age and geography to detect sarcasm in a more robust way. Last, but not the least, to apply Recurrent Neural Network to get more efficient and accurate results since text data is a sequence by its nature.

Contribution

Team Members	Contribution
Liubov Tovbin (013747890)	Proposed the topic, found the related articles and synthesized them for the project proposal. Coding: performed feature extraction. Final report: wrote an abstract, related work and feature extraction sections.
Asel Salmenbayeva (013720291)	Found the dataset, proposed using XGBoost Coding: Random Forest, XGBoost, plotted graphs Final report: analyzed the results and wrote the experiment section
Akshata Deo (012565761)	Writing work: Contributed in preparing a project proposal, class presentation and final report (data preparation and conclusion), github README.md Coding work: Performed bagging with Logistic Regression. Performed bagging with SVM.
Sachin Sharma (013718146)	Coding - Bootstrapping with SVM (Bagging) Final Report- Introduction to problem, Technical Approach

Acknowledgment

We want to thank our professor Dr. Mahima Agumbe Suresh for sharing her expert knowledge and igniting our interest in machine learning.

References

[1] S. G. Wicana, T. Y. İbisoglu, U. Yavanoglu, "A Review on Sarcasm Detection from Machine-Learning Perspective," 2*017 IEEE 11th International Conference on Semantic Computing,* DOI 10.1109/ICSC.2017.74

[2] D. D. Anandkumar, P. Nikita, "A Comprehensive Study of Classification Techniques for Sarcasm Detection on Textual Data," *International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) - 2016,* DOI 10.1109/ICEEOT.2016.7755036

[3] S. Raghav, E. Kumar, "Review of Automatic Sarcasm Detection," *2017 2nd International Conference on Telecommunication and Networks (TEL-NET 2017),* DOI 10.1109/TEL-NET.2017.8343562

[4] E. Sulis, D. I. Hernandez Farias, P. Rosso, V. Patti, G. Ruffo, "Figurative Messages and Affect in Twitter: Differences Between #irony, #sarcasm, and #not," *Knowledge-Based Systems*, Volume 108, p. 132–143, 2016, DOI 10.1016/j.knosys.2016.05.035

[5] K. Buschmeier, P. Cimiano ve R. Klinger, "An Impact Analysis of Features in a Classification Approach to Irony Detection in Product Reviews," *WASSA*, Baltimore, 2014

[6] M. Khodak, N. Saunshi, K. Vodrahalli, "A Large Self-Annotated Corpus for Sarcasm," 2017, https://arxiv.org/abs/1704.05579