An Exploration of Mini-Batch Optimization and Document Analysis through Sarcasm Detection

Ayodeji M. Lawal - Department of Mathematics, Senior

Seoyeon Kang - Department of Statistics, Junior

University of Wisconsin - Madison

Author Note

Audience: Peers in the same department

# Introduction

As human interaction occurs at an unprecedented scale, communication through various languages and mediums will play an ever-larger role in ensuring society functions properly. Specifically, as globalization has accelerated, the sheer magnitude of documentation and paperwork has reached a size by which human-only analysis is unfeasible; this presents an opportunity to shift to machine learning models as the optimal means of document analysis.

Detecting emotional tone in text and distinguishing fake news articles from reputable news articles are two fields closely related to our project. However, the task with which this project concerns itself is detecting sarcasm in news headlines. Using data acquired from Kaggle, what sorts of machine learning models can be built to classify headlines as sarcastic or non-sarcastic? How can they be optimized to improve accuracy and resource intensity?

This project focuses on normalizing training metrics for four types of machine learning models and evaluating their performance based on classification accuracies, training resource intensity, prediction resource intensity, and storage requirements. By controlling for unwanted variations and taking performance metrics as the models and networks attempt to complete the same task, we aim to learn about the intricacies of different types of models, what their strengths and weaknesses are, and how variations in the mini-batch heuristic affects their performance.

# 

# Data Structure

This project uses the second version of the dataset acquired from the “[News Headlines Dataset For Sarcasm Detection](https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection)” project prompt on Kaggle last updated on 2019-07-04. The news headlines in the dataset were sourced from two news websites: The Onion, a website aimed at producing sarcastic versions of current events, and the Huffington Post, an acclaimed newsite that serves as the non-sarcastic headline source.

Each record consists of three attributes:

1. **Labels: “is\_sarcastic”**
   1. In this column, a “1” indicates sarcastic headlines from the Onion and a “0” indicates non-sarcastic headlines from the Huffington Post.
2. **Features: “headline”**
   1. This column contains the headline of the news article. We will further discuss how this data is manipulated so ML algorithms can be applied.
3. **Further Info: “article\_link”**
   1. This column contains a link to the original news article. Useful in collecting supplementary data but four our purposes it is unnecessary and will be deleted.

Note: Data Samples and Visualizations can be found in Appendix B.

# 

# Data Pre Processing

For Machine Learning algorithms to be applied to our features, developers must ultimately convert them into a standard numerical vector format; those familiar with it can skip to the next section. The rest of the section provides a detailed explanation of the process. T

The first step in this process is basic data cleaning, i.e., deleting the URL column and removing brackets. The second step is stop-word removal; during this step, we remove words that are so common in the English language they provide little to no indication of information. Readers can find examples of stop-words in Appendix A.

Next, we convert inflectional and derivationally related forms of words into their base format; this process is called stemming and lemmatization. Despite seeming counterproductive to the task at hand, since it removes cues humans use to interpret sarcasm from sentences, it actually aids the model in identifying patterns by removing inconsistent word systems found across all languages.

The fourth and final step where the remaining base words are converted into a vector of numbers is called tokenization. Though there are many ways to tokenize text, our models use Word2Vec, a process of arranging words as positions in a vector space. This allows for similar words to be clustered closer together, and classification can often be done using principal component analysis. Despite being interesting in their own regard, those subjects are beyond the scope of this paper and this project.

# 

# Existing Models

In our paper, we examine [a Convolutional Neural Network (CNN) model](https://www.kaggle.com/madz2000/sarcasm-detection-with-glove-word2vec-83-accuracy) developed by Madhav Mathur. In addition, we also examine [a Recurrent Neural Network (RNN) model, a Random Forest Model, and a Support Vector Machine (SVM) model,](https://www.kaggle.com/zaeemahmed/sarcasm-with-rnn-random-forest-and-svm) all developed by Zaeem Ahmed. In addition to standardizing the datasets, we’ve also set the minibatch split to 0.2 and epochs to 30 or convergence for all neural networks as we will be varying these later. The classification accuracy of these models were reported in Kaggle; those results as well as other base model metrics recorded in our project can be found in Appendix C and D.

# 

# Platform

All scripts were run on Python 3.7.11 in the Google Colab IDE. All files are stored on the Lex Labs Google Drive, a copy of which is accessible to readers here (link yet to be determined).

# 

# The Experiment

After running all four of our models and finding that the RNN and CNN models produced the highest classification accuracy, we decided to move forward with those models for mini-batch experimentation. Our models were evaluated on four criteria: classification accuracy, training resource intensity, prediction resource intensity, and storage requirements. These evaluations occurred across mini-batch sizes of 200, 100, 75, 50, and 25. For each batch size, five model iterations were run, tested and averaged to produce values found in appendix E.

Classification accuracy measurements were compiled using the SkLearn.metrics library by earlier developers. Training and prediction resource intensity is measured via the Memory Profiler extension built into Google Colab. Using the Memory Profiler module, we measured the maximum amount of memory used when the model is trained, and when it makes predictions, referred to as peak memory. Also taken were measurements of the total time of execution when the model is trained or makes a prediction; this is further broken down into the time the processor spends running the code called CPU time and the time spent running the code in the operating system kernel called System Time.

Finally, we downloaded our models using the Load\_models package in Keras for the RNN and CNN and the Pickling library for the Random Forest Model and SVM and ascertained storage requirements by examining their file size in Windows File Explorer. The file size is measured in both “Size,” indicating the true file size, and “Size on Disk,” indicating the actual amount of space being used on the hard drive; “Size on Disk” differs based on how a storage disk is partitioned so for this project, we will focus on “Size.”

The reason for an investigation into the mini-batch heuristic in this manner is partly to discern what effect increasing or decreasing mini-batch size can have on performance. Beyond this, we aim to examine how resource-intensive nonoptimal mini-batch sizes can be and produce a clearer understanding of what real-world costs it could inquire as training sessions grow in length. During this course, we’ve seen training sessions last from 20 minutes to 40 minutes; however, in real-world applications, training sessions can sometimes last weeks, if not months. Even a 10% increase in runtime or memory usage for no real benefit to prediction accuracy can have high monetary costs.

We hypothesize that as the mini-batch size increases, the prediction accuracy will increase as larger datasets are more likely to capture the overall trend rather than overfit the data and be swayed by noise. We expect the downside to this is that runtime and memory usage of the program will increase in training and perhaps testing, as each iteration will have to deal with larger datasets than they otherwise would in smaller mini-batches.

# 

# Result - CNN Existing Model Metrics

CNN Training

| Peak Memory | 630.95 MiB |
| --- | --- |
| Increment | 101.20 MiB |
| CPU Times | 29.4 s |
| System Time | 609 ms |
| Total Time | 30 s |
| Wall Time | 17.4 s |

CNN Prediction

| Peak Memory | 631.86 MiB |
| --- | --- |
| Increment | 0.83 MiB |
| CPU Times | 2.46 s |
| System Time | 119 ms |
| Total Time | 2.58 s |
| Wall Time | 1.78 s |

CNN Accuracy

|  | Precision | Recall | F1-Score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.91 | 0.55 | 0.68 | 3001 |
| 1 | 0.65 | 0.94 | 0.77 | 2723 |
| Accuracy |  | | 0.73 | 5724 |
| Micro Avg | 0.78 | 0.74 | 0.73 | 5724 |
| Weighted Avg | 0.79 | 0.73 | 0.72 | 5724 |

CNN Size

| Size | 323 KB (331,537 bytes) |
| --- | --- |
| Size on Disk | 324 KB (331,776 bytes) |

# 

# Result - RNN Existing Model Metrics

RNN Training

| Peak Memory | 692.82 MiB |
| --- | --- |
| Increment | 0.48 MiB |
| CPU Times | 46.8s |
| System Time | 1.93s |
| Total Time | 48.7s |
| Wall Time | 27.9s |

RNN Prediction

| Peak Memory | 702.82 MiB |
| --- | --- |
| Increment | 1.21 MiB |
| CPU Times | 5.93s |
| System Time | 272ms |
| Total Time | 6.2s |
| Wall Time | 3.91s |

RNN Accuracy

|  | Precision | Recall | F1-Score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.89 | 0.70 | 0.78 | 2999 |
| 1 | 0.73 | 0.90 | 0.81 | 2725 |
| Accuracy | 0.80 (0.797) | | | 5724 |
| Micro Avg | 0.80 | 0.80 | 0.80 | 5724 |
| Weighted Avg | 0.80 | 0.80 | 0.80 | 5724 |

RNN Size

| Size | 798 KB (817,688 bytes) |
| --- | --- |
| Size on Disk | 800 KB (819,200 bytes) |

# 

# Result - Random Forest Existing Model Metrics

Random Forest Training

| Peak Memory | 1200.29 MiB |
| --- | --- |
| Increment | 0.00 MiB |
| CPU Times | 24.9s |
| System Time | 637ms |
| Total Time | 25.5s |
| Wall Time | 25.7s |

Random Forest Prediction

| Peak Memory | 1200.29 MiB |
| --- | --- |
| Increment | 0.00 MiB |
| CPU Times | 1.25s |
| System Time | 44ms |
| Total Time | 1.3s |
| Wall Time | 1.42s |

Random Forest Accuracy

|  | Precision | Recall | F1-Score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.66 | 0.71 | 0.68 | 2946 |
| 1 | 0.67 | 0.61 | 0.64 | 2778 |
| Accuracy | 0.66 (0.6624737945492662) | | | 5724 |
| Micro Avg | 0.66 | 0.66 | 0.66 | 5724 |
| Weighted Avg | 0.66 | 0.66 | 0.66 | 5724 |

Rand Forest Size

| Size | 399 MB (418,719,335 bytes) |
| --- | --- |
| Size on Disk | 399 MB (418,721,792 bytes) |

# 

# Result - SVM Existing Model Metrics

SVM Training

| Peak Memory | 818.44 MiB |
| --- | --- |
| Increment | 0.11 MiB |
| CPU Times | 32.9 s |
| System Time | 79.9 ms |
| Total Time | 33 s |
| Wall Time | 33 s |

SVM Prediction

| Peak Memory | 818.55 MiB |
| --- | --- |
| Increment | 0.05 MiB |
| CPU Times | 32.9 s |
| System Time | 57 ms |
| Total Time | 33 s |
| Wall Time | 32.9 s |

SVM Accuracy

|  | Precision | Recall | F1-Score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.54 | 0.86 | 0.66 | 2984 |
| 1 | 0.55 | 0.18 | 0.28 | 2740 |
| Accuracy |  | | 0.54 | 5724 |
| Micro Avg | 0.54 | 0.52 | 0.47 | 5724 |
| Weighted Avg | 0.54 | 0.54 | 0.48 | 5724 |

SVM Size

| Size | 511 MB (536,419,771 bytes) |
| --- | --- |
| Size on Disk | 511 MB (536,530,944 bytes) |

# 

# Result - Existing Model Confusion Matrices

CNN

array([[1645, 1356],

[167, 2556]])

RNN

array([[2108, 891],

[271, 2454]])

Random Forest

array([[2097, 849],

[1083, 1695]])

SVM

array([[2560, 466],

[2100, 598]])

# Result - Existing Model Classification Measures

Rounded to 4 decimal places

| Measure | CNN | RNN | Random Forest | SVM |
| --- | --- | --- | --- | --- |
| Sensitivity (Recall) | 0.9078 | 0.8861 | 0.6594 | 0.5494 |
| Specificity | 0.6534 | 0.7336 | 0.6663 | 0.5620 |
| Positive Predictive Value (Precision) | 0.5482 | 0.7029 | 0.7118 | 0.8460 |
| Negative Predictive Value | 0.9387 | 0.9006 | 0.6102 | 0.2216 |
| False Positive (Alarm) Rate | 0.3466 | 0.2664 | 0.3337 | 0.4380 |
| [False Discovery Rate](https://onlineconfusionmatrix.com/#measures) | 0.4518 | 0.2971 | 0.2882 | 0.1540 |
| False Negative Rate (Miss Ratio) | 0.0922 | 0.1139 | 0.3406 | 0.4506 |
| [Accuracy](https://onlineconfusionmatrix.com/#measures) | 0.7339 | 0.7970 | 0.6625 | 0.5517 |

# Result - CNN Mini-batch Performance Metrics

Rounded to 5 decimal places

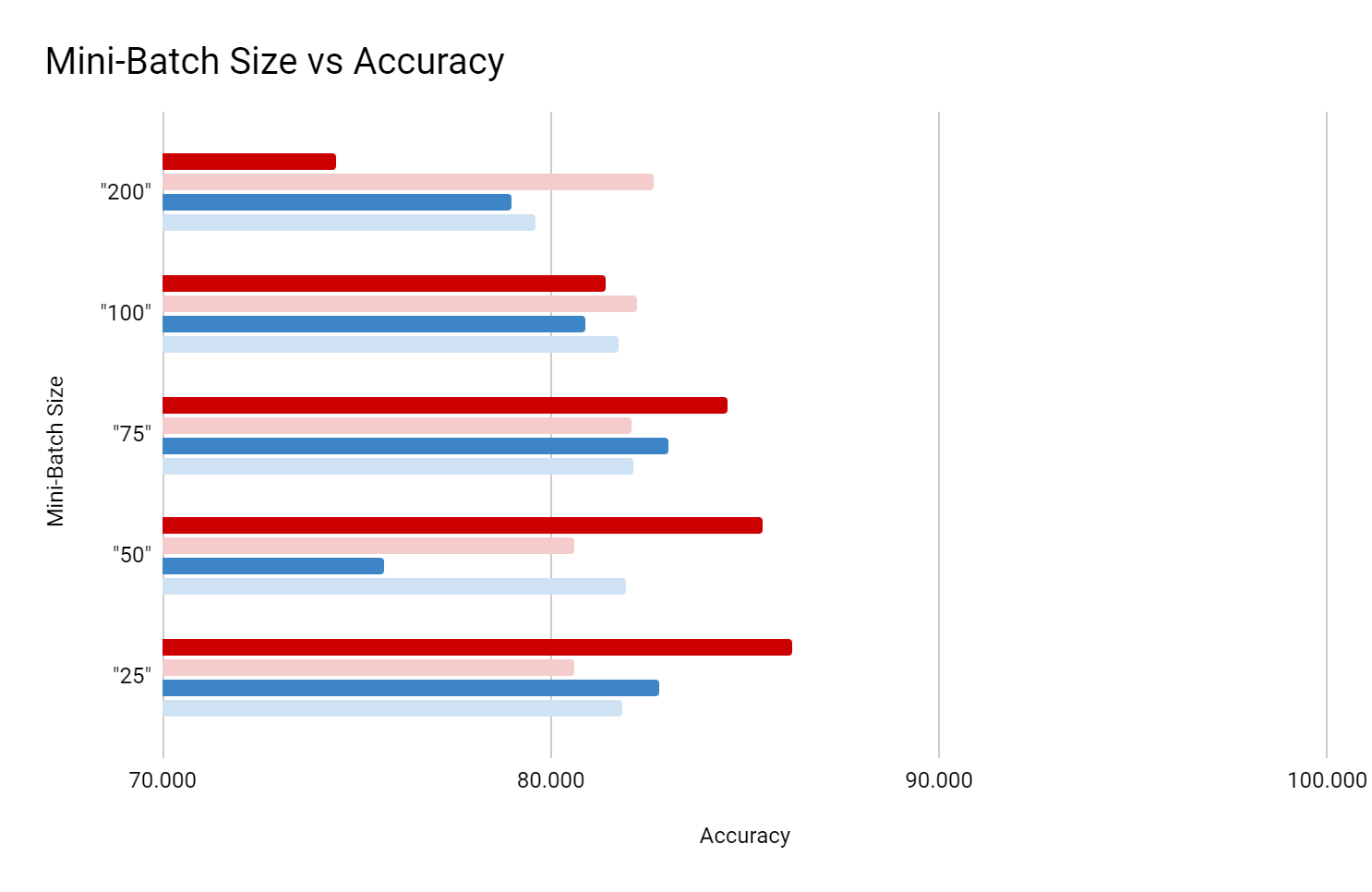
| Mini-Batch Size | 200 | 100 | 75 | 50 | 25 |
| --- | --- | --- | --- | --- | --- |
| Training  Accuracy | 0.74489 | 0.81425 | 0.84570 | 0.85475 | 0.86200 |
| Prediction  Accuracy | 0.82659 | 0.82208 | 0.82089 | 0.80580 | 0.80894 |
| Training  Runtime (s) | 13.799504 | 14.89530 | 15.46719 | 15.83304 | 18.03402 |
| Prediction  Runtime (s) | 1.30491 | 1.36940 | 1.31524 | 1.32560 | 1.371974 |
| Training  Memory-Use (MiB) | 665.35313 | 671.32266 | 671.39844 | 684.40703 | 684.40313 |
| Prediction  Memory-Use (MiB) | 665.33203 | 671.30703 | 671.37500 | 684.38672 | 684.39453 |

# Result - RNN Mini-batch Performance Metrics

Rounded to 5 decimal places

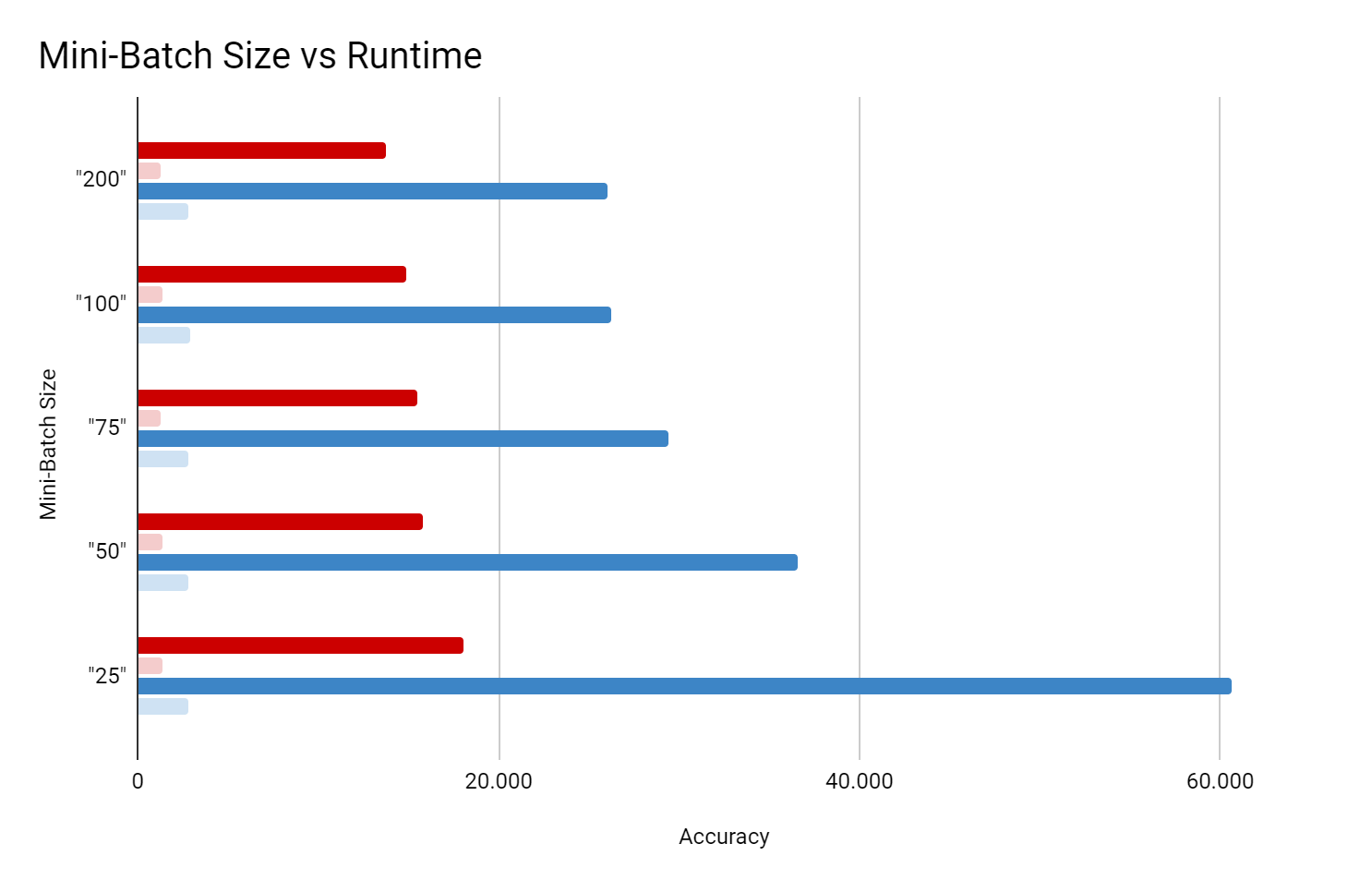
| Mini-Batch Size | 200 | 100 | 75 | 50 | 25 |
| --- | --- | --- | --- | --- | --- |
| Training  Accuracy | 0.78968 | 0.80903 | 0.83007 | 0.75702 | 0.82813 |
| Prediction  Accuracy | 0.79598 | 0.81754 | 0.82121 | 0.81943 | 0.81855 |
| Training  Runtime (s) | 26.01321 | 26.19866 | 29.44444 | 36.60399 | 60.59368 |
| Prediction  Runtime (s) | 2.78476 | 2.87182 | 2.83948 | 2.81994 | 2.80047 |
| Training  Memory-Use (MiB) | 863.01172 | 863.0508 | 863.04766 | 863.08594 | 863.11328 |
| Prediction  Memory-Use (MiB) | 863.10547 | 863.09766 | 863.0898 | 863.11328 | 863.10569 |

# Result - Mini-Batch Performance Graphs

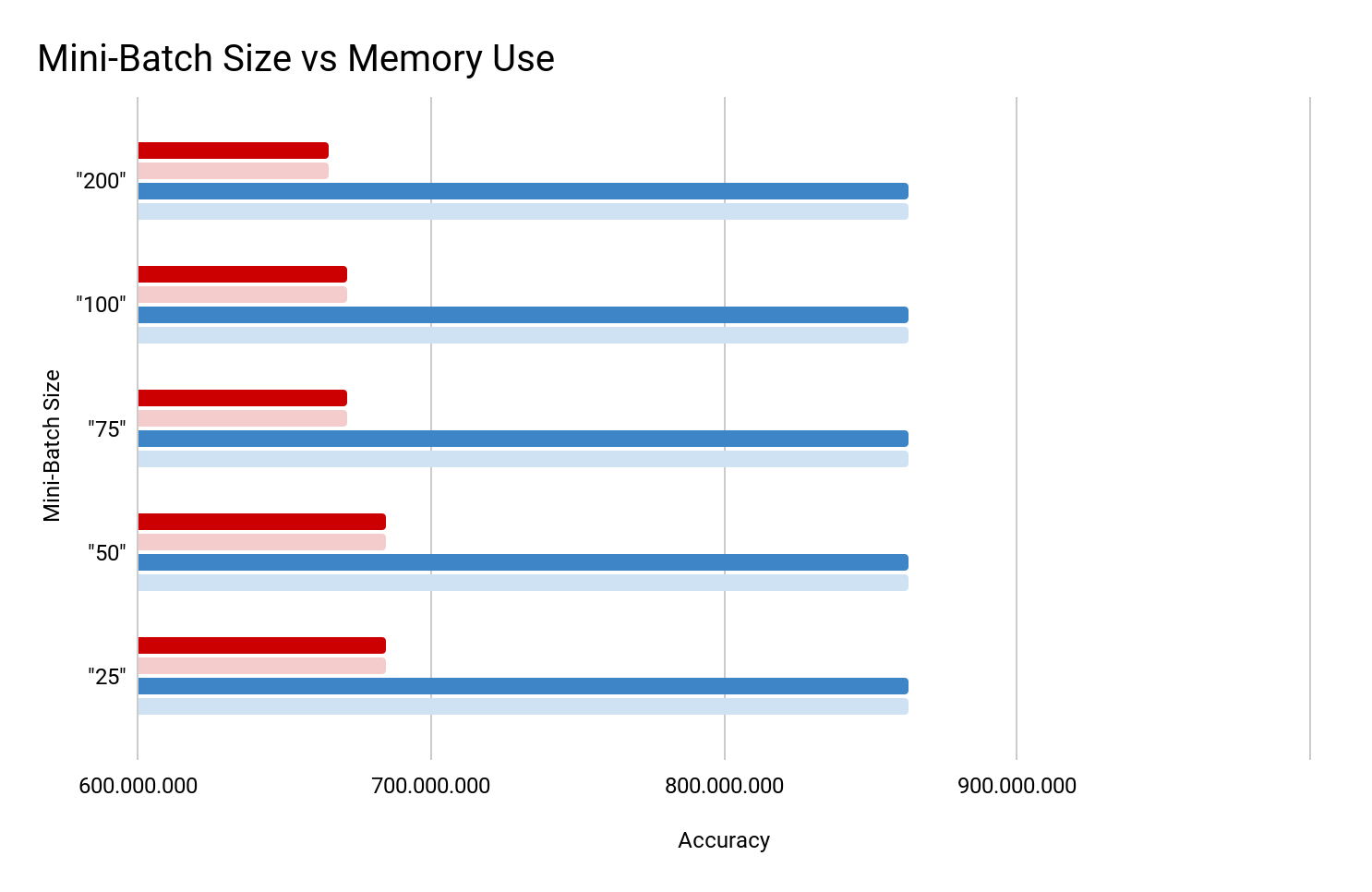


NOTE: Graph starts at 70% and goes to 100%

NOTE: Abnormality in mini-batch size 50, to be discussed in discussion



NOTE: Graph starts at 0s and goes to 60s



NOTE: Graph starts at 600 MiB and goes to 1,000 MiB

# Discussions

The defining metric by which models are ranked is, without a doubt, their accuracy. Our model's performance ranked from highest to lowest with 0.2 mini-batch split and 30 epochs or convergence setting is listed as follows: RNN (0.797) > CNN (0.734) > RF (0.662) > SVM (0.538). As intuition suggests, the two neural networks models, RNN and CNN, outperformed the Rand Forest and SVM models.

For a more detailed analysis of the performance of the classification models, we used a confusion matrix to calculate extra measures such as sensitivity, specificity, and precision. CNN and RNN detected sarcasm with the highest sensitivity (0.9078 and 0.8861). RNN also scored first in specificity with 0.7336. However, regarding the precision, the trend reversed as two non-neural network models had the highest value (SVM: 0.8460, Random Forest: 0.7118).

As machine learning occurs more and more on our phones, it is important to note the resource demand of each model. The CNN was the least resource-intensive model, with the smallest memory usage, shortest runtime, and smallest storage size. Here are some resource demand metrics and rankings:

Peak Memory : CNN (631.86 MiB) < RNN (702.82 MiB) < SVM (818.55 MiB) < RF (1200.29 MiB)

Total Time : CNN (2.58s) < RNN (6.2s) < RF (25.5s) < SVM (33s)

Size on Disk : CNN (324 KB) < RNN (800 KB) < RF (399 MB) < SVM (511 MB)

Though the lack of intensity in the deep neural networks compared to what could be perceived as a simpler model like the Support Vector Machine is a surprising result, this hints that neural network models are better equipped to detect sarcasm in news headlines.

By differing Mini-Batch Sizes, we were able to see some variations on Accuracy, Runtime, and Memory Use. For both CNN and RNN, the prediction accuracy was higher than training accuracy with large batch sizes (200 and 100) . Also, in the case of the CNN, when we increased the mini-batch size, the training accuracy decreased while the prediction accuracy increased. The reason for this is because as mini-batch size decreases there is more overfitting and a reduced ability to capture the general trend. This means that our hypothesis was wrong about the behavior of training accuracy but right about that of the prediction accuracy. RNN seemed to follow the same pattern, but it was hard to conclude since there was an abnormality in size 50 Mini-Batch experiment. Overall, CNN outperformed RNN in both training and prediction accuracy except for Mini-Batch size 200, where it only outperformed in prediction accuracy.

Runtime was recorded by taking the total execution time when the models were training or making predictions. As we increased the batch size, both CNN and RNN experienced a decrease in the training time. This was the reverse of what we hypothesized. For RNN, the change was more extreme than the training time dropped from 61s to 26s when we increased the batch size from 25 to 200. However, the prediction time was almost unaffected by the size of the mini-batches. Comparing the two models regarding the runtime, CNN ran almost twice as fast as RNN. Model runtime increases as mini-batch sizes decrease because epochs contain mode weight updating iterations which take up more time; i.e. a 100 sample data set would have 10 weight updates at mini-batch size 10 but only 2 weight updates at mini-batch size 50.

Our last performance metric was memory use. We calculated the maximum amount of memory used during training and prediction. Unlike our prediction, memory use was rarely affected by the mini-batch size variations. CNN did experience some decrease in its memory use while increasing the mini-batch sizes. However, the change was statistically insignificant. In both models, training and prediction used almost the same amount of memory. CNN was the more optimal model since it only used 665~684MiB while RNN used up to 863MiB. We believe this is because the models and functions they come with play a larger role and are held constant.

It is also important to understand that this experiment has a couple of drawbacks; specifically, future researchers could make three improvements to our experiment: sample size, number of iterations and algorithm efficiency. A sample size improvement would entail sampling minibatch sizes over a larger range or through finer increments. Increasing the number of iterations across which models are averaged from 5 to 100 would also be an easy improvement to make. A harder improvement would be deriving a method to collect all performance metrics at once rather than one at a time.

In addition to a small sample of possible heuristics, models, applications, and datasets; beyond the model itself, computational architecture like hardware, bit storage type, and operating systems are also aspects that developers can further investigate. However, despite not being an all-encompassing study, it shows that even a small amount of attention paid to how models are trained can yield impressive results.

This project serves as a small venture into computational complexity in Artificial Neural Networks. Machine learning is a subject built around optimization; usually, this is focused on optimizing weights and producing the best accuracy. However, as machine learning moves forward as a field, models must be optimal in accuracy and resource intensity.

# 

# References

Brownlee, Jason. “Save and Load Machine Learning Models in Python with Scikit-Learn.” *Machine Learning Mastery*, 27 Aug. 2020, machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/.

Ekstrand, Michael. “User Cpu Time vs System Cpu Time?” *Stack Overflow*, 1 May 1959, stackoverflow.com/questions/4310039/user-cpu-time-vs-system-cpu-time.

Frisco, Brice. “Python Tensorflow - Running MODEL.FIT Multiple Times WITHOUT Reinstantiating the Model.” *Stack Overflow*, 1 Nov. 1968, stackoverflow.com/questions/62120508/python-tensorflow-running-model-fit-multiple-times-without-reinstantiating-the.

godaygogodaygo. “How to Store the Result from %%TIMEIT CELL Magic?” *Stack Overflow*, 1 June 1965, stackoverflow.com/questions/41351041/how-to-store-the-result-from-timeit-cell-magic.

Kumar, Niranjan. “Introduction to Sequence Modeling Problems.” *Medium*, Towards Data Science, 17 Dec. 2019, towardsdatascience.com/introduction-to-sequence-modeling-problems-665817b7e583.

LudvigH, and Anshul Goyal. “What Is %Timeit in Python?” *Stack Overflow*, 1 Sept. 1963, stackoverflow.com/questions/29280470/what-is-timeit-in-python.

madz2000. “Sarcasm Detection with GloVe/Word2Vec(83%Accuracy).” *Kaggle*, Kaggle, 2 July 2020, www.kaggle.com/madz2000/sarcasm-detection-with-glove-word2vec-83-accuracy.

Misra, Rishabh. “News Headlines Dataset for Sarcasm Detection.” *Kaggle*, 3 July 2019, www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection.

Nicholson, Chris. “A Beginner's Guide to word2vec and Neural Word Embeddings.” *Pathmind*, 2020, wiki.pathmind.com/word2vec.

Sravyaysk. “Sarcasm Detection with Cnn, Rnn, LSTM, Bert.” *Kaggle*, Kaggle, 10 July 2021, www.kaggle.com/sravyaysk/sarcasm-detection-with-cnn-rnn-lstm-bert/execution.

“Stanford - Stemming and Lemmatization.” *Stemming and Lemmatization*, Cambridge University Press, 2008, nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.

# 

# Appendix A - Glossary

Stop Word Removal - “the”, “is”, “in”, “for”, “where”, “when”, “to”, “at” etc

Stemming & Lemmatization - The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is => be

car, cars, car's, cars' => car

The result of this mapping of text will be something like:

the boy's cars are different colors => the boy car be differ color

# 

# Appendix B - Data Visualization

Fig B.1 - A Small Sample of The Dataset

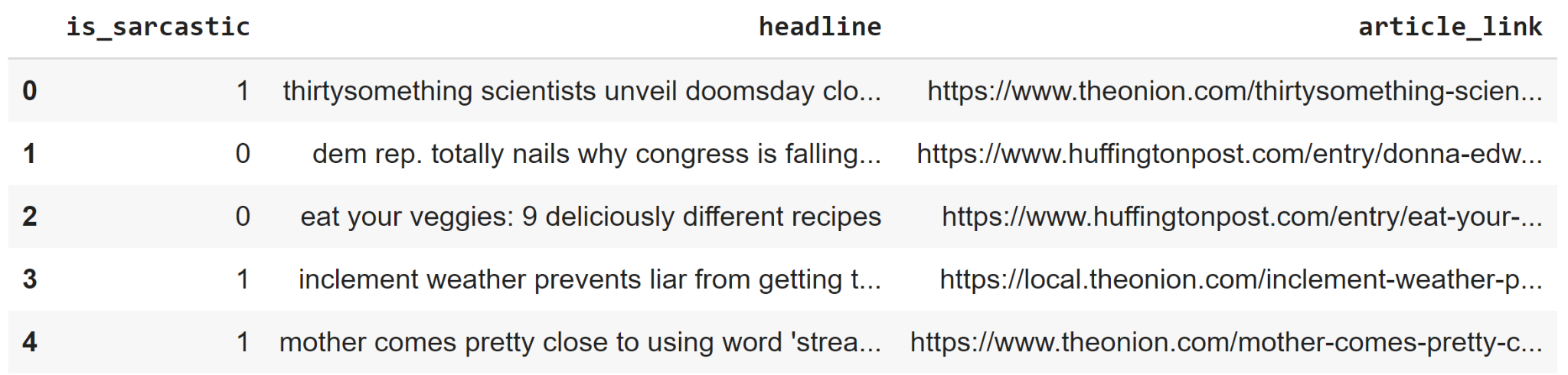
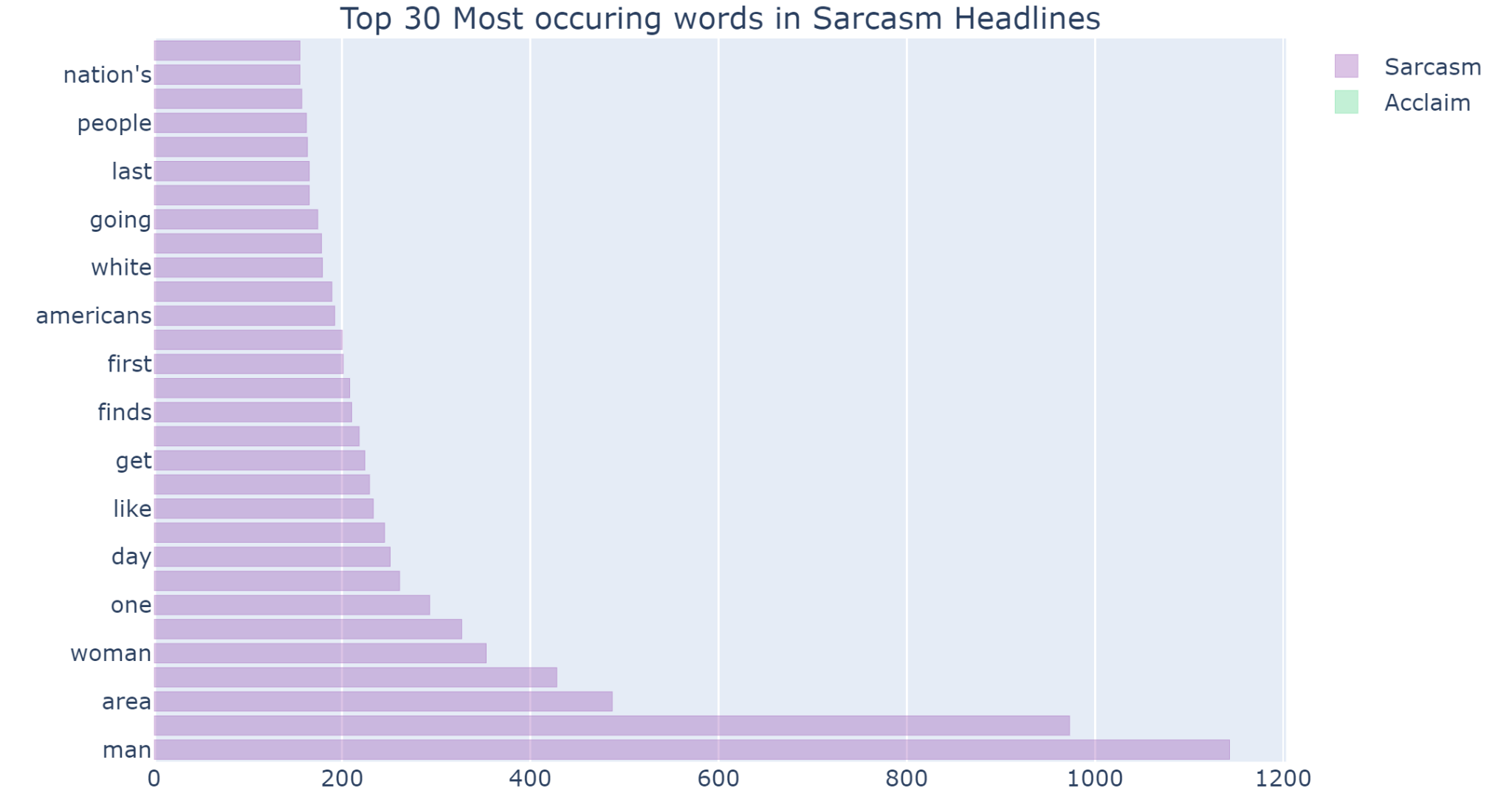
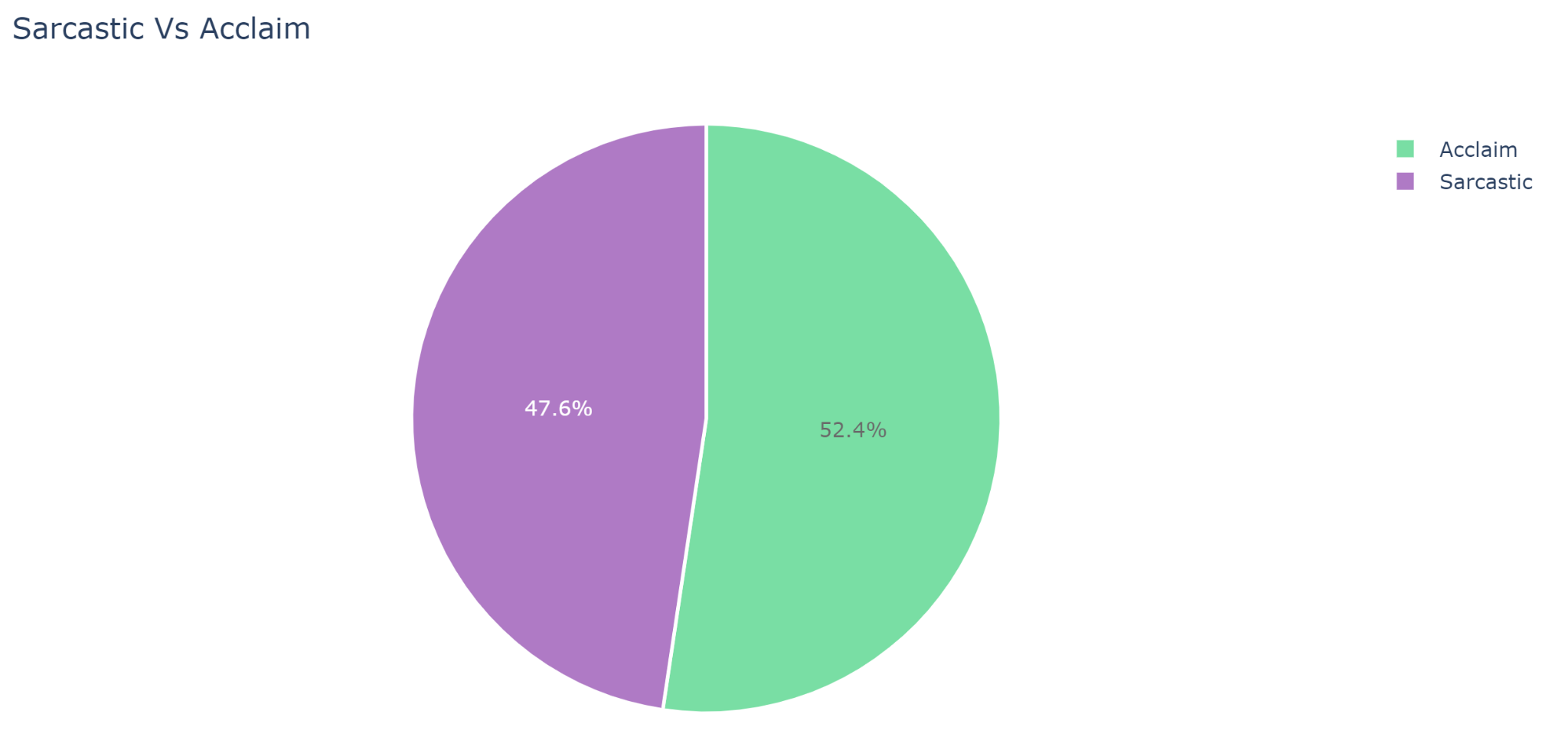
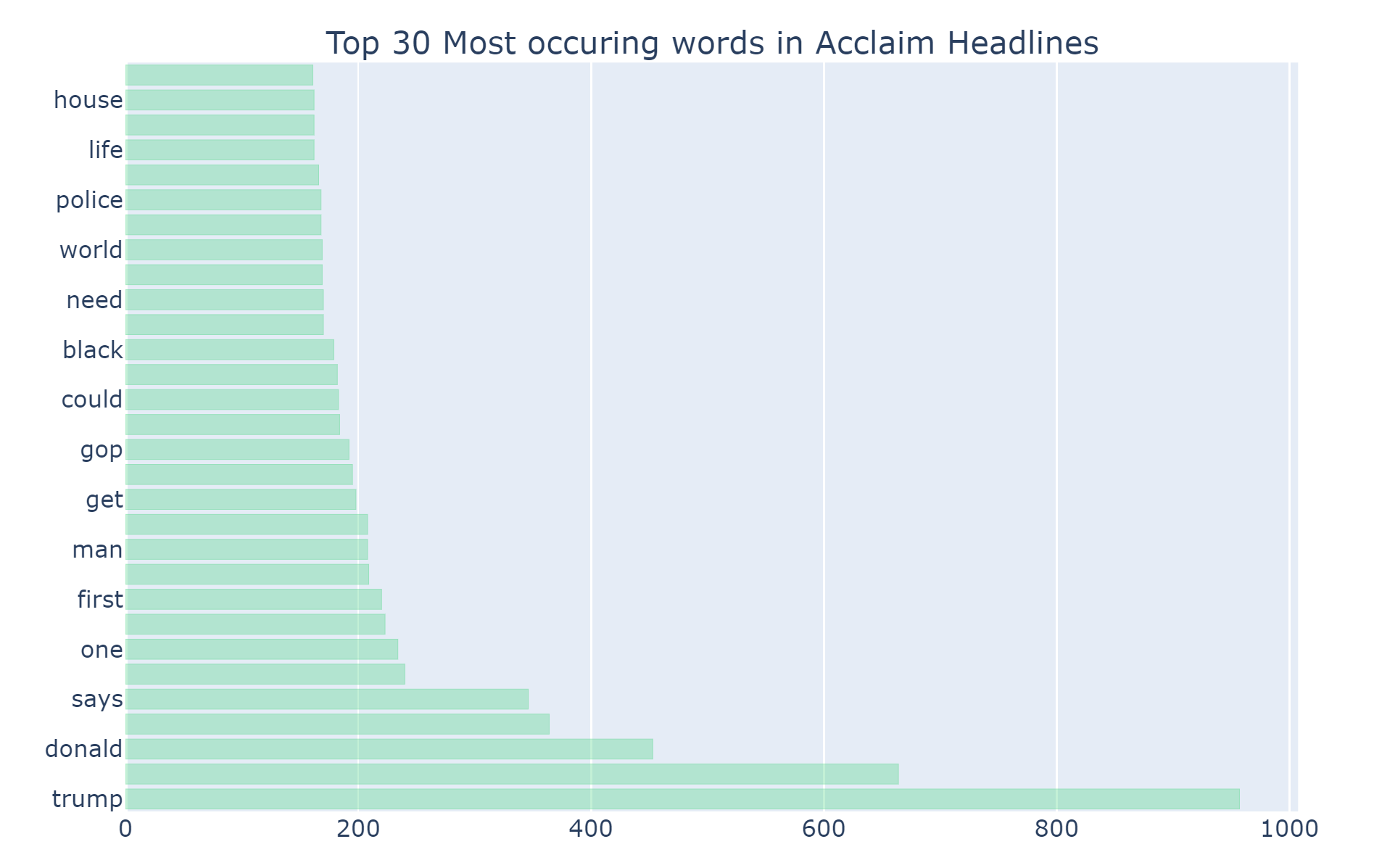


Fig B.2 - Sarcastic vs Non-Sarcastic (Acclaim) Labels



****