Sarcasm Detection - Final Report

Yuxing Wu
yw7vv@virginia.edu

Kai Qu
kq4ff@virginia.edu

Issac Li
il5fq@virginia.edu

1 Problem Statement

By the definition from The Free Dictionary, sarcasm is "a form of sarcasm that is intended to express contempt or ridicule." It is a kind of figurative language, which is widely used on social media(4). Understanding sarcasm correctly is the key to understand the true feelings and opinions expressed by users. But the nature of sarcasm makes it challenge for sentiment analysis and opinion mining. For example, one sentence could sound positive on the surface, but actually implies a negative sentiment. This led our interest to sarcasm detection.

The difficulties of detecting sarcasm lie in the fact that there are multiple types of sarcasms, and some of them are very subtle and do not have apparent patterns within the sentence. Sarcasms can be mainly categorized as 'verbal sarcasm by polarity contrast', 'situational ironies' and 'other types of ironies'. The first type of ironies is easier to identify compared to the second, as in the second type of sarcasm, there is no apparent sentiment shift , and it requires commonsense knowledge and logic to correctly identify the type. Moreover, some sarcasm require previous and latter contextual information, such as the twitter replies, to help identify them, which increases the difficulty of sarcasm detection.

To our knowledge, there are three popular ways to solve this task. They can be summarized as feature-engineering method, traditional machine learning method, and deep learning method (2). One of goals in this project is to try out different models associated with three methods above, with the focus on deep learning method as it generally achieves better result, compare and analyze final results in various metrics. Moreover, if there is time remaining, we would also like to fine-tune BERT on this task and compare its result on this task to the previous ones.

2 Expected Outcome

For this project, there are two datasets to be trained with. The large dataset contains 962,294 reddit comments either labelled with 1 or 0, indicating the comment is sarcastic or not. The small dataset contains 4,618 tweets with labels indicating the presence of sarcasm. For the logistic regression, as the baseline model, we expect it achieves around 70% accuracy on the test set of the large dataset. This expectation comes from an example that uses logistic regression model trained on the same dataset on kaggle. Since the small dataset contains less data, we expect the baseline model reaches around 60% accuracy on the test set of the small dataset. For the remaining models, it is expected that they will outperform the baseline performance, since they have larger capacity to model the desired function for this task, compared to the baseline model. Because contextual information is crucial for sarcasm detection, we also expect that models containing GRU or LSTM layers to have a better results than the Vanilla CNN models, as LSTM and GRU have reportedly good performance for processing entire sequence of input data. Finally, BERT (1) is expected to outperform all other models by a large margin.

3 Proposed Method

3.1 Traditional Machine Learning Models

We plan to try different traditional machine methods including SVM, Random Forest and Logistic regressions and find the baseline model for our project. In order to implement the models, we use Scikit-learn's libraries for data tokenization. We are going to try scikit-learn's CountVectorize and TFIDFVectorize and choose the one that gives a better result. During the data preprocessing, we will limit max features, deal with capitalization and stopword, and try different feature extractions.

3.1.1 Support Vector Machine

The first model we plan to try is suppert vector machines since SVM is one of the simplest model for classification. We are going to use Scikit-learn built-in SVM classifiers and plan to compare the performance of SVMs with different kernels, including Linear SVM, Polynomial SVM and RBF SVM.

3.1.2 Random Forest Model

The second traditional machine learning model we are trying is Random Forest. The Random Forest will build a decision tree. Each node model will split the data into two, a randomly selected feature that optimizes the split quality using the Gini criteria.

3.1.3 Logistic Regression Model

The last tradition machine learning model we plan to use is a logistic regression. When training the model, we will vary the solver, penalty and C to find the best logistic regression model.

3.2 Deep Learning Models

3.2.1 Vanilla CNN

For the vanilla CNN, we first have an embedding layer, then we have 4 2d-CNN layers with kernel size of $2 \times$ EmbeddingDim, $3 \times$ EmbeddingDim, $4 \times$ EmbeddingDim and $5 \times$ EmbeddingDim, and finally we have a linear layer and a dropout layer. The CNN kernels are like n-gram filters which we hope can capture the occurrence of different n-grams and their relation with the sarcasm sentiment.

3.2.2 GRU

This is just the vanilla multi-layer gated recurrent unit RNN implemented in pytorch.

3.2.3 GRU with Pooling

Instead of using the hidden state of the last layer for t = sequence length, we max-pool and average-pool the outputs for each t (from 0 to sequence length), concatenate them and feed them into the final linear layer. We hope this one can better capture the overall sentiment of the sentence.

3.2.4 CNN + GRU Concat

We simply concatenate the result from the CNN and the GRU for this model. Since CNN and RNN capture different features from sentences, we hope by concatenating them together, they can generalize to more sarcastic cases and better classify sarcastic sentences.

3.2.5 CNN + GRU Forward

This model contains two CNN layers and one GRU unit. The input text sequence is first fed to the CNN layers to produce n-gram like features. Then the features are proceeded into GRU layer and the final linear layer to make predictions. The purpose of CNN layers here is to reduce frequency variation and connect a subset of feature space that is shared across the entire input.

3.2.6 GRU + GRU

To better capture the information from the parent comment, we simply feed the parent comment into a GRU and use the last hidden state as the input for the GRU of the reply comment. The idea is that we hope the first GRU can capture the information of the parent comment and by using it as the initial hidden state for the second GRU, it can better capture the sarcasm in the comment.

3.2.7 BERT

We simply use the pretrained-pytorch-model from Huggingface. The tokenizers are replaced with the Bert tokenizer, and the rest are the same. Due to the very limited computational resource available, we only finetune the pretrained model on the twitter dataset and report the results on section 5.

4 Experimental Details

4.1 Dataset

We use two datasets to be trained with our models. The first dataset is a dataset used in Semeval 2018 task 3: Irony Detection in English Tweets(5). The training corpus contains 3,834 tweets, and the test set containing 784 tweets. Each tweet is labelled with either 1 or 0, indicating whether it is sarcastic or not. The second dataset is a dataset we found in Kaggle (3). It contains 962,294 reddit comments and their parrent comments. Each reddit comment is labelled with either 1 or 0, indicating it is sarcastic or not. The Reddit dataset is balanced, containing 50% of the comments labelled 1 (sarcastic) and 50% of the comments lablled 0 (non-sarcastic). We split the entire corpus into the training set and the test set, where the training set contains 800,000 reddit comments, and the test set contains 200,000 reddit comments.

4.2 Data Processing

We tokenize each sentence using nltk.tokenize.WordPunctTokenizer and map each token to a index. Afte tokenizing and reindexing, we assign each word to its corresponding twitter pretrained embedding and the glove embedding with 200 dimensions. For words not in the pretrained embeddings, we map it to '<UNK>'. Here we use two pretrained word embedding to investigate whether the choice of word embedding has an effect on the prediction accuracy. Then, since in each batch, the length of sentences are different, we pad each sentence with 0s to the longest length of sentence in the batch. Finally, we pack the padded sentences using pack_padded_sequence provided in PyTorch.

4.3 Parameters Tuning

- Hyperparameters:
  - $Learning\_rate = 0.001$
  - $Embed\_dimension = [400, 200]$
  - $Epochs = 10$
  - $BatchSize = 128$
  - $Optimizer = [SGD, Adam]$
  - $Dropout = [0.2, 0.5, None]$
  - CNN
    - $CNN\_filter\_size = [3, 5, 7, 9, 11, 13]$
    - $CNN\_num\_filters = 256$
  - $LSTM/GRU$
    - $LSTM\_hidden\_dim = 256$
    - $LSTM\_layers = [1, 2]$
- The twitter pretrained embedding has 400 dimensions, whereas the glove embedding we use has 200 dimensions.
- For the CNN+GRU Concat Model, all different filter sizes of CNN are used and the features produced by different filters are concatenated together. For the CNN+GRU Forward Model, only kernel of size 3 is used in the CNN layer.
- SGD and Adam are applied without using momentum and AdamGrad respectively.

5 Results and Analysis

The results of all the models trained on different datasets are reported in the tables below:

Table 1: The Result of Different Models on the Twitter Dataset

Model	Precision(%)	Recall(%)	F1(%)	Accuracy(%)
Random Forest	68.41	61.68	64.87	67.14
Logistic Regression	73.74	68.58	71.07	72.07
CNN	62.43	73.12	71.23	75.33
GRU	67.01	72.09	68.16	76.07
CNN-GRU-Pooled output	65.24	75.01	72.19	78.11
CNN-GRU-Concat	65.04	74.21	72.32	75.51
CNN-GRU-Forward	60.54	71.61	72.25	70.91
BERT	66.03	77.01	75.56	80.66

Table 2: The Result of Different Models on the Reddit Dataset

Model	Precision(%)	Recall(%)	F1(%)	Accuracy(%)
Random Forest	62.54	60.95	61.74	64.51
Logistic Regression	62.12	64.05	63.07	74.07
CNN	71.33	48.93	58.16	70.23
GRU	73.52	51.03	59.32	74.43
CNN-GRU-Pooled output	74.42	49.77	59.32	74.26
CNN-GRU-Concat	72.46	50.32	58.63	72.56
CNN-GRU-Forward	73.31	48.26	68.81	73.91
BERT	N/A	N/A	N/A	72.96

5.1 Traditional Machine Learning Models

After trying different tools from Scikit-learn libraries to perform word tokenization and feature extraction, we end up using Tfidfvectorizer with max features = 50000, N-grams = [1,2] and removing the words with the frequency lower than 3. In terms of our result, Random Forest model becomes our base-line mode. We tried SVMs; however, it gives the lowest accuracy of 58% and 0 on recall or precision. In addition, it took very long time to run on the larger Reddit data. Due to the limited computing resources, we end up giving up SVM models. Surprisingly, simple logistic regression performs better than our expectation. Especially, on the Reddit data, its performance is as good as deep learning methods.

5.2 Deep Learning Models

5.2.1 Word Embeddings

We tried the GloVe Embedding and the Twitter embedding as pre-trained embeddings for this project, and we found that for the 600k training data, there are 60k words GloVe Embedding can find and 40k words that it can't find. Except the numbers and usernames, this is still a large number. Therefore, we tried the Twitter embedding and around 80k words are found and 20k words are not found. We expected the Glove Embedding could do better than no-pre-trained-embedding and the Twitter Embedding could do better than the Glove embedding, but the results showed little difference in changing pre-trained word embeddings.

5.2.2 Data Size and Parent Comment

We can see from the table that our deep learning models have similar results. For the Reddit data, training them on less than 100k rows of data will provide a testing accuracy of around 70%(mostly below 70%). By increasing the training data to 600k and the testing data to 400k, we got the accuracy listed in the table, and they are all above 70%.

For the twitter data, since the training and testing data are not enough, it's hard to get good precision. However, the overall recall is high. This could be because the data are well pre-processed. The numbers are turned to "<num>" tags, and the usernames are replaced with "<username>" tags. What's more, there are many hashtags in the data, and many sarcastic sentences contain sarcasm-related hashtags which makes detecting sarcasm easier. We think this is why the average recall of this data is way higher than the Reddit data.

For all the results listed in the table of the Reddit data, we only used the reply comments to detect sarcasm and didn't use the parent comments. However, the reply comments carry limited information and sometimes are not sarcastic at all if you don't look at the parent comments. For example, "I really hope this was" is really hard to be classified as sarcastic. Therefore, we concatenate the parent comments and the reply comments and use the concatenated results as inputs. In order to separate the parent comment and the reply comment, we inserted a "<sep>" tag between the parent comment and the reply comment. However, there is still the limitation that the parent comments are usually longer than the reply comments, and the parent comment may weight more than the reply comments. We then tested on the concatenated data, and the accuracies of all the models improved by 1%.

5.2.3 CNN

The CNN model with kernel sizes from (2, Embedding Dimension) to (7, Embedding Dimension) has the worst result. This is expected because the CNN model will only capture local n-gram features, while sarcasm, in most cases, rises from bipolar words over long distances. However, given that it's only 2% below the average precision, we think it did a relatively good job. The reason might be because the comments are mostly shorter than 15 words, and having kernel sizes up to 7 can capture most of the features.

5.2.4 GRU

GRU unit RNN performs better than the CNN model as expected, and we can see that the GRU with pooling has better performance than the vanilla GRU using the hidden state. This meets our expectation that the concatenated result of the max-pooling and average-pooling gives a better summary of the sentence than the hidden state. This could be improved by having attention as weights and calculating the weighted sum, and we will talk about this in detail in the overall analysis section below.

5.2.5 Combinations of CNN and GRU

The model that uses the concatenated output of CNN and GRU didn't meet our expectation, because we hoped that it can detect sarcasm only CNN or only GRU cannot detect and thus increase accuracy, but it seems to be not as easy as just concatenating the outputs from the two models.

For the model that first uses a CNN to capture features and then feed the features into a GRU has similar performance to the vanilla GRU model. It seems that using CNN to generate better embeddings for input word sequences doesn't make a huge difference.

For the model that first runs the parent comment through a GRU and uses the resulting hidden state as the initial state for the second GRU, it has a higher accuracy, but not much. The reason could be because using a single hidden state of the parent comment can't capture enough information, and the second GRU can't make good use of this hidden state.

5.2.6 BERT

We expected BERT to perform really well on this task, but it takes too long to train.

5.2.7 Overall Analysis

The deep learning models have similar performance over both datasets. The low recalls of the deep-learning models indicate the difficulty of sarcasm detection. And by comparing the correctly-classified comments, we found that they didn't differ much from model to model. We guess that this is because sarcasm detection is a hard task, and the models we tried are basically just combinations

of CNN and RNN. They can tackle with the easy cases, but for many sarcastic sentences, they simply can't detect sarcasm due to their model structure. Even if we concatenate them together, it's still a hard task for RNN to capture the long-distance features. That's why attention should be a good mechanism to capture the relation between the words of the parent comment and the reply comment. By having the attention scores of the parent and reply comments, they can be used as weights to sum the hidden states and produce good summaries of the comments. And by concatenating the weighted-sum of the parent comment and the reply comment, it could produce a representation that better captures sarcasm than the models we implemented.

6 Conclusion

In this project, we experiment with different CNN and LSTM models on sarcasm detection with both the twitter dataset and the reddit dataset. Overall, our models have similar performance over both datasets. While the precision scores of some LSTM models are high, the results show that all models have relatively low recall scores. We suspect that this is because sarcasm detection is a hard task, and our models can only deal with the easy cases due to their limited structures. Since attention mechanism can help by assigning attention scores to both parent and reply comments, we speculate that it can help to produce good summary of sentences and therefore understand the relationship between words. Moreover, CNN performs better than our expectation, which is possibly due to the fact that a large portion of the sentences in both datasets are sentences short. For the future work, we will alleviate the issue of imbalance of sentence length by including longer comments or tweets to both datasets. Moreover, we will add attention mechanism to our decoders to see if it can improve the performance of sarcasm detection. We also plan to test out different kinds of attentions and analyze their effects on improving the performance on sarcasm detection.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[2] A. Joshi, P. Bhattacharyya, and M. J. Carman. Automatic sarcasm detection: A survey. *CoRR*, abs/1602.03426, 2016.

[3] M. Khodak, N. Saunshi, and K. Vodrahalli. A large self-annotated corpus for sarcasm. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[4] B. Liu. Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing*, 2010.

[5] C. Van Hee, E. Lefever, and V. Hoste. SemEval-2018 task 3: Irony detection in English tweets. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 39–50, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.