Statistical Natural Language Processing: Project Final Report

Sarcasm detection

Lucas Swiniarski
NetID : ls4411
lucas.swiniarski@nyu.edu

Ruben Stern
NetId : rns365
rns365@nyu.edu

Abstract

We want to tackle the problem of sarcasm detection. We find this problem interesting for various reasons. Sarcasm is a challenging task as it requires a high level of language understanding to be detected. For instance, one sentence can be both sarcastic and not sarcastic depending on the context. Even for a normal human being, it can be quite hard to detect sarcasm. Sarcasm detection raises interesting questions: How important is the context of a sentence in its sarcastic meaning? Is sarcasm necessarily situation-bounded?

1 Introduction

Even if the text classification task have been broadly covered in the Natural Language Processing environment, not a lot of work have been done in the case of sarcasm detection. It corresponds to a very challenging task and even human classification is not completely accurate as sarcasm can be hard to detect and it can depends of it author and it context.

In [1], they collected a new dataset for sarcasm research and for training and evaluating systems for sarcasm detection. The advantage of this dataset lies in his size as it is 10 times bigger than any previous dataset designed for this specific task. Though it labels come from self annotation of comment authors on Reddit, the authors of this paper took care of reducing it noise while collecting this dataset. After harvesting this dataset, they also implemented some baselines and reached satisfactory performance with Bag-of-Bigrams model: 75.8% accuracy in a balanced setting.
Before [1], most of the work in this area corresponded to use Twitter data and manually annotate tweets. Then, datasets where way smaller and consequently performances where also smaller.

Furthermore, we shared our code on an open repository.[1]

2 Data sets

2.1 Self-Annotated Reddit Corpus

The dataset we are going to use is the Self-Annotated Reddit Corpus (SARC) which comes from [1]. It is a large corpus designed especially for sarcasm detection. It contains 533 million statements among which 1.3 million are labeled as sarcastic. This will allow us to train our model with both balanced and unbalanced methods.

This corpus has been extracted from Reddit, a social news aggregation, web content rating, and discussion website. Reddit's users can submit content such as text posts or direct links. They can

[1] https://github.com/lucas-swiniarski/Stat-nlp-project

then vote submissions up or down that determines their position on the page. Submissions with the most up-votes appear on the front page or the top of a category. In addition, content entries are organized by areas of interest called subreddits. For instance, subreddit topics are news, science, movies, music, books, food, ... Thanks to its 234 million monthly unique users and it 725 million comments, Reddit correspond to a huge source of text data. Indeed, SARC Data set have been built as a subset of comments from January 2009 to April 2017.

Furthermore, each statement in SARC has been labeled by the author (not an independent annotator) and is provided with a user, topic and conversation context (previous comments on the page). The common method used by authors for sarcasm annotation consists of putting the marker "/s" at the end of their statement. Such a method implies that these markers are noisy since many users do not make use of the marker, do not know it, or use it only when the sarcastic intent is not obvious. In order to reduce this noise, authors of [1] excluded comments that are answers of sarcastic comments as the label, in this case, is extremely noisy because authors agree or disagree with the previously expressed sarcasm with their own sarcasm without often labeling it. Another technique they used to decrease the noise of the SARC is to keep only comments from users that know about the "/s" sarcasm annotation which corresponds to users who have already used the marker previously.

In order to do a raw estimation of SARC's noise, they also labeled manually a sample of 500 comments tagged as sarcastic and 500 tagged as nonsarcastic, taking into account the full context of these comments. Thus, they tried to estimate the false positive rate and false negative rate of their data set. They found out a false positive rate of 1.0% and a false negative rate of 2.0%. The false positive rate remains reasonable while the false negative rate is significant as the sarcasm proportion is 0.25% which highlights a wide definition of sarcasm and indicates a need of methods that can handle noisy data in the unbalanced setting. In the balanced setting, it still corresponds to a small amount of noise.

2.2 South Park cartoon lines

We also plan to use a data set available on Kaggle: South Park cartoon lines.[2] It contains all the lines (70k) from the cartoon South Park annotated with the season, episode and speaker. Although this dataset is not labeled with sarcasm, we want to apply our model trained on SARC data in order to quantify how sarcastic each character is and rank them by sarcasm. Then, by doing some visualization and look manually at the produced labels, we want to check the transfer learning ability of our model.

3 Models

3.1 Linear Models

In the section, we will present the linear model used as a baseline and the techniques used to improve it.

3.1.1 Baseline

As a baseline, we decided to use a logistic regression using tf-idf counts as features of the comments. tf-idf, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. Indeed, it can be expressed as follows:

For each document d, for each term t (word):

$$tf - idf(d,t) = tf(d,t) \cdot idf(d,t)$$

$$idf(d,t) = log(\frac{1+n}{1+df(d,t)}) + 1$$

[2] https://www.kaggle.com/tovarischsukhov/southparklines

where n is the total number of documents and where tf(d, t) corresponds to the number of times that term t occurs in document d. This baseline gave us a 67% accuracy score on the test set which is satisfactory enough to interpret them by looking at words with highest and lowest weights. It also let us some space for improvements by tuning our model.

3.1.2 Improvements of Linear Models

In order to get better accuracy, we tried to tune the regularization factor of our model and the n-gram size to include in our tf-idf vectorizer at the same time.

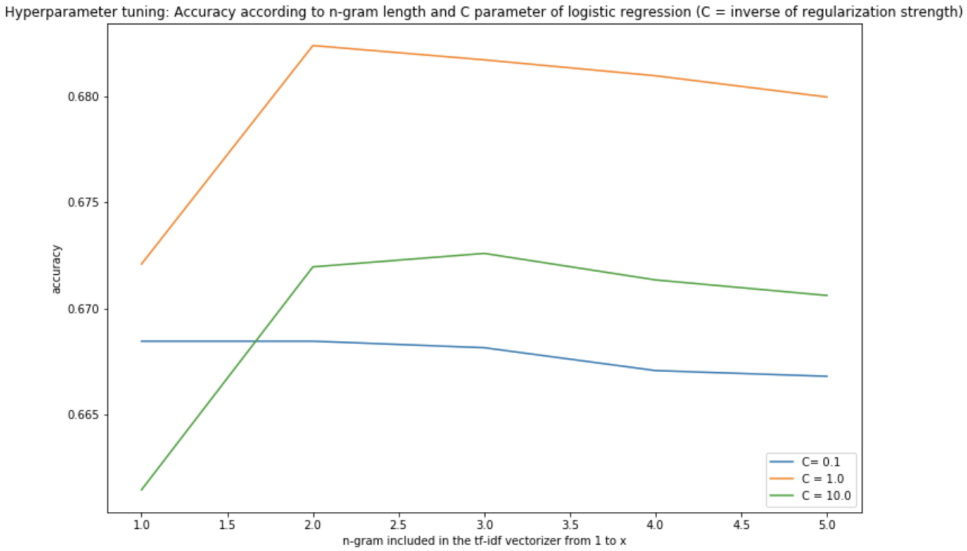

Figure 1: Hyper-parameters tuning of linear model

As it can be seen in Figure 1, it appears to be better to use only unigrams and bigrams, and to use a regularization strength of 1. Using these, we only improved our accuracy score of 1%.

Then, we wanted to try to include not only the comments in Reddit that we are assessing if it is sarcastic but also previous comments in the discussion so that our model can also "understand" sarcasm as a response to previous comments. In Figure 2, we can see the influence of the memory size (the number of previous sentences included in our model) on the accuracy. To do so, we implemented a different tf-idf vectorizer for each previous sentence: one for the sentence before the comment we are assessing, one for the second sentence before the one we are considering, one for the third, ...

Figure 2 shows that it drastically improve our model and that the larger the memory is, the better the accuracy is. Indeed, the best accuracy (77%) corresponds to using the eight previous sentences. We can deduct that some of the sarcasm really corresponds to a specific answer to previous comments and then it can only be understood by our model if we give it the previous comments.

3.2 Deep Learning Approach

In this section, we will present the data pre-processing, the models trained and their results, and our state-of-the-art model and hyper-parameters.

3.2.1 Pre-processing of the dataset

The first step before training a deep learning model is to pre-process the dataset. In order to do so, we ran once through the training set, ordering each word by their frequency. We then create a dictionary mapping words to integer index, ordered by frequency. We ordered by frequency in order to cross-validate the vocabulary size in later steps. In all, a small vocabulary size allows learning

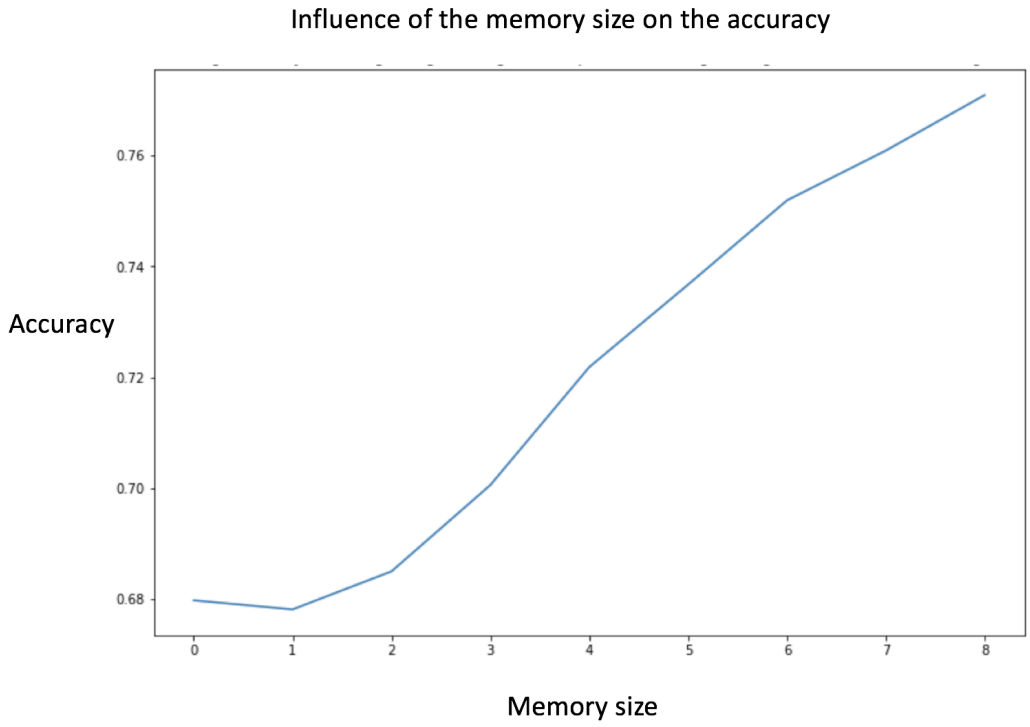

Figure 2: Influence of the memory size on the accuracy

good word embedding of each word, while a big vocabulary size might produce over-fitting with some words embedding being trained on only one sentence.

Parsing We first parsed the sentences only splitting a sentence into a list of words via split by space. It yielded us $1.2M$ unique words, which is huge for a dataset in English and is the consequence of our dataset: Reddit user often misspell, position uppercase letters randomly, etc. We then chose to use nltk word tokenizer, which reduced the number of unique words to $650K$. It's already a great improvement, due to splitting verbs conjugation for instance: $don't \rightarrow do + n't$. We chose to keep upper-cases words because they can convey a lot of information on the sarcasm of a sentence.

We then transformed our training set, validation set and testing set using this dictionary, storing the list of indexes for faster loading. Noting that every unseen word in the validation/test set is mapped to a $< UNK >$ token. Once again it's important to not train on the full training vocabulary size, otherwise, the $< UNK >$ token is never trained.

Secondly, we only consider a single past sentence when training those models, we could get improvements in the future by looking at more past sentences, but it allows us to train in a reasonable amount of time our model in this case.

Finally, our dataset is composed of approximately $1M$ training sentences, 5% of it are labeled sarcastic. In order to alleviate the problem of an unbalanced dataset, we chose to create a balanced training set out of it, using all sarcastic sentences and a randomly selected subset of unsarcastic sentences of the same size. While we loose information doing it, deep learning models already take hours to train on this small training set, thus given the limited resources available our result won't change drastically from using a smaller training set.

lose tackle the problem of unbalanced datasets, we could have done :

- Over-sampling strategy of the minority class, in our case sarcastic sentences.
- Weighted cross-entropy term, so it's more costly to make an error in a sarcastic sentence than a non-sarcastic one.

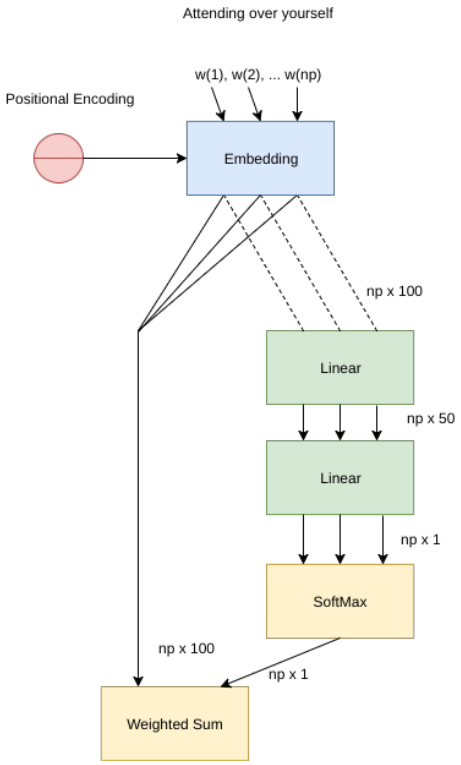

Figure 3: Self-attention module

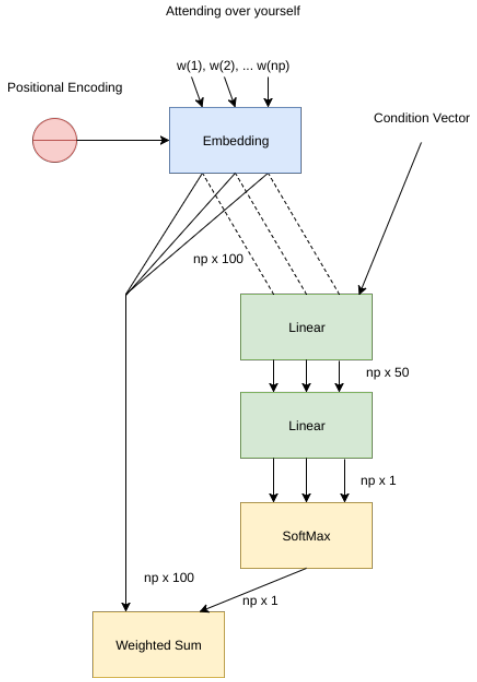

Figure 4: Conditional attention module

3.2.2 Self-Attention and Conditional Attention

We chose to study fully-attentional classification models, as explained in [3]. We started from the description of attention in [2], we used to types of modules: One we call self-attention, as it's done without conditional information. A second module we call conditional attention as it takes as input an array of vectors we attend on and a conditional information vector.

First, we don't have any positional information when using word embeddings because we don't use any recurrent neural networks to embed the sentence. Because of it, we used positional encoding as explained in [3]. This positional encoding works as follow: Given a word embedding of size d_{model} for instance, I concatenate to it a vector of size 50 filled with $\frac{pos}{10000^{2i/d_{model}}}$, and take the cosine of this value every odd number, sine every even number. The periodicity induced by cosine/sine is thought to help the model find periodical patterns in sentences.

3.2.3 GloVe Embedding

Our dataset, in the end, is reasonably big: We have $200K$ training sentences. It's often better to use pre-trained embedding matrices as we might not have enough information to learn embedding vectors. We thus downloaded GloVe[6] embeddings from their website [3] and used Wikipedia 2014 50-dimensional word embeddings. Out of the $650K$ unique words found in the train, $198K$ are matched with the GloVe embedding vocabulary. Also, it was important for us to use the nltk word tokenizer, as it's the one used in GloVe - thus make the vocabularies match better.

We then trained twice the same model, one with pre-trained glove embedding, the other with random initialization. We found little differences between the two models after training, both achieving 78% accuracy and similar training curves. We think this is due to the peculiar form of our dataset. Reddit users make a lot of typos, use upper-case letters in unusual spots, etc. while GloVe is trained on Wikipedia, a dataset without any of those problems. In Wikipedia, there are few to no typos, the

[3] https://nlp.stanford.edu/projects/glove/

vocabulary is clean. In all, we think there are too many differences between the two datasets to leverage the strength of a good pre-trained word embedding matrix. Further work should include pre-training Word2vec or GloVe or another word-embedding learner algorithm on our full dataset for a better initialization.

3.2.4 Other hyper-parameters used

Optimizers We tried 3 different optimizers: we implemented stochastic gradient descent without momentum, and with gradient clipping for our Recurrent Neural network model. For the attention model, we used SGD, RMSprop, and Adam. RMSprop and Adam yield similar very good results while SGD yields relatively poor results

Learning Rate The first iterations of our model used a low learning rate 10^{-3}, which was due to our gradient norm values being high in the embedding matrices. In order to alleviate this problem, we used batch normalization modules in our Multi-Layer Perceptron (MLP), which solved the issue and we got steady gradient norm at each layer, at a scale of approximately 10^{-2}. After this debugging step, our learning rate was always set to 10^{-1}.

Finally, we have a learning rate reduction policy, every 5 epoch we check if the validation loss is bigger than the best validation loss seen during training, if it's higher we divide by 2 the learning rate.

3.2.5 Recurrent Neural Network

Recurrent neural networks usually yield state-of-the-art results on natural language processing tasks. There are unfortunately some caveats of using RNNs :

- Recurrent Neural Networks are hard to train correctly. It necessitates gradient clipping to alleviate the exploding gradient problem, using more complex recurrent modules such as GRU [4] or LSTMs [5] to alleviate the vanishing gradient problem.
- It's overall slow to train due to its sequential nature on sentences.
- It gets state-of-the-art results with big architectures made of millions of parameters. We can't afford to train such architecture with the available computational resources.

We trained some recurrent architecture with GRU and LSTMs cells, using a hidden state size of 256, with a dropout rate of 0.5. This architecture is made of one bidirectional RNN running on the current and previous sentence, concatenate both last hidden states and feeding it to an MLP. The best results we could get was 68% test accuracy after 20 epochs.

3.2.6 Best model trained

For the deep learning part, we chose to focus on Attention models for various reasons :

- It's overall faster to train, as the computation of an Attention mechanism is fully parallelizable, computationally speaking. Compared to an RNN which needs to sequentially compute its hidden states.
- It's a good exercise to understand the attention mechanism.

We have chosen, for computational time purposes, to learn such model in the current sentence and the previous sentence. Taking into account a bigger chunk of the past of a conversation would increase the computational burden. Let's call p_1, p_{l_p} the words of the previous sentence, and $c_1, \cdots, c_{l_c}$ the words in the current sentence. We first embed each word to a vector and concatenate a positional embedding to this vector. Because we only use attention, we would lose the positional information without doing this step. Positional embedding is done in the same manner as [3]. Afterward, we attend over each sentence multiple times. The number of times attended is an hyper-parameter. This allows our model to have multiple vector representation of the sentence, which should encapsulate different meaning. We call this operation self-attention, as we attend the sentence over itself, selecting words based on their importance and position throughout the corpus. We then attend over the current

sentence conditioned on the attended vectors over the previous sentence, and vice-versa. The goal of this step is to learn dependencies between sentences. Certainly, when using sarcasm, the author will make allusion to what has previously been said. These 4 operations, namely 2 self-attention and 2 conditional attention, produce a vector representation of the sentence. This is then fed to a multi-layer perceptron (MLP) which will classify the sentence into sarcastic or not. Figure 5 shows the architecture of the network. $w_p(1), \cdots, w_p(n_p)$ are the words of the previous sentence, while $w_c(1), \cdots, w_c(n_c)$ are the words of the current sentence.

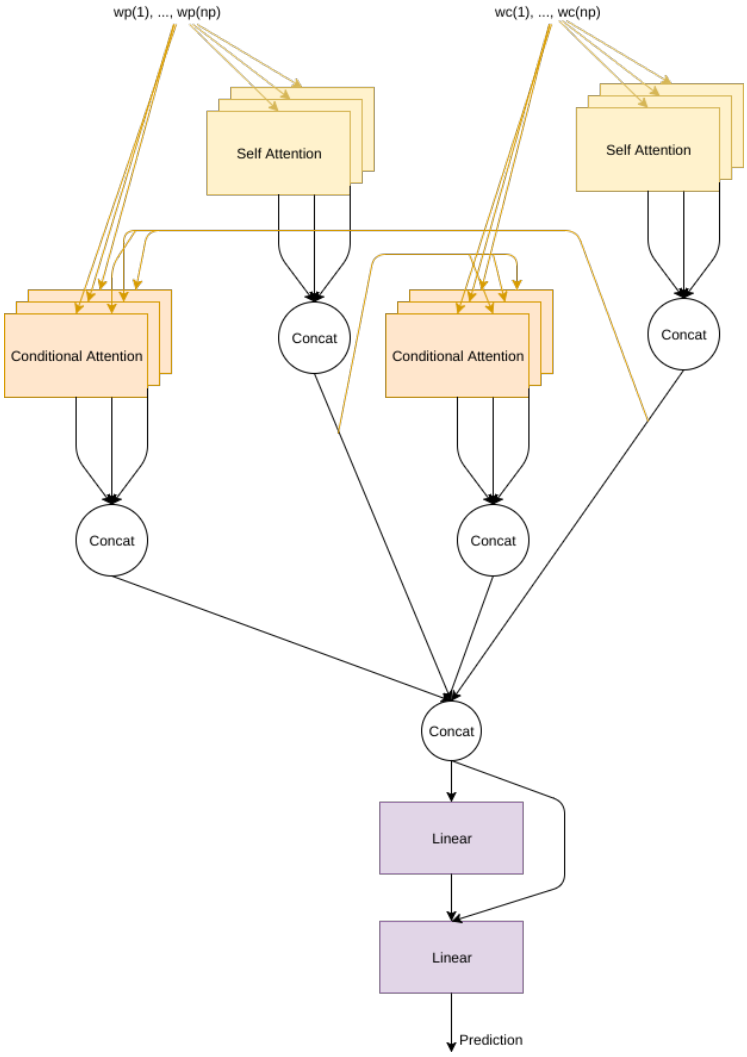

Figure 5: Attention-based Deep neural network.

The hyper-parameters of our best model are :

Embedding size	50
Vocabular size	500000
Attention Hidden layer size	256
Learning rate	0.1
Optimizer	Adam (β_1, β_2 unchanged)
Batch Size	128
Dropout Rate	0.5
Self-attention heads	7
Conditional Attention heads	5
Test Accuracy	79%

3.3 Learning Curve

We found the task at the end quite challenging to learn for the network. Its training loss is very slowly decreasing, and the network is overall generalizing quite well. Usually, neural networks tend to fit the training set dataset and the generalization part is rather challenging, which is not the case here.

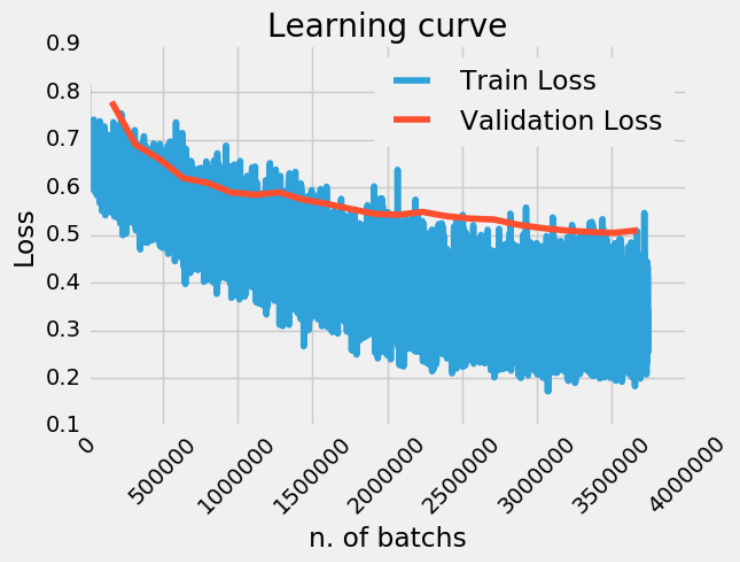

Figure 6: Learning curve of our deep learning model.

Figure 6 shows the learning curve of our model. While the variance of the training loss gets bigger across time, we think it can be explained in part by the dataset. Because of the difficulty of over-fitting the training set, we chose to show at every epoch some false-positive and false-negative results. It yields a good understanding of what the model has learned while allowing us to make improvements on its architecture. Here are some examples, where $pTrue$ is the label and predicted the probability of being sarcastic.

4 Results

4.1 Interpretation of Linear Models

The main advantage of linear models is that they could be easily interpreted. Indeed, it gives weights to every n-grams which corresponds to it sarcastic influence on the whole comments. In Figure 7, we are showing the bigrams that have the biggest influence on the prediction as they have the highest and lowest weights. Thus, using words characterizing such as "clearly", "totally", "sure", "obviously" are really associated to irony and sarcasm. On the contrary, using insults and slang language are associated with sincerity and no sarcasm.

4.2 South park dataset

South park dataset is a very good test for our model as :

- With some knowledge of the TV show Southpark, we can easily say whether our model is right or not.
- It shows how well our model generalizes to other datasets.
- The English displayed in South Park is close in Vocabulary to the one used in Reddit, except for the misspells.

We run our linear and deep learning model on the south park dataset and create a score for each character, which is the percentage of comical riposte said during the show.

bigrams with highest weights	bigrams with lower weights
obviously	itt
clearly	fuck yeah
yeah	reminds
totally	honestly
sure	feel like
good thing	fucking
confirmed	10 10
surely	shit
amirite	kinda
dare	florida
yea	fucks

Figure 7: Bigrams with highest and lowest weights

Character	bad	good	score	total
Cartman	4767.0	5016.0	51.272616	9783.0
Stan	3980.0	3785.0	48.744366	7765.0
Kyle	3506.0	3612.0	50.744591	7118.0
Butters	1230.0	1374.0	52.764977	2604.0
Randy	1142.0	1328.0	53.765182	2470.0
Mr. Garrison	480.0	522.0	52.095808	1002.0
Chef	487.0	432.0	47.007617	919.0
Kenny	674.0	209.0	23.669309	883.0
Sharon	428.0	434.0	50.348028	862.0
Mr. Mackey	277.0	357.0	56.309148	634.0
Gerald	279.0	349.0	55.573248	628.0
Jimmy	285.0	312.0	52.261307	597.0
Wendy	288.0	298.0	50.853242	586.0
Liane	286.0	296.0	50.859107	582.0
Sheila	271.0	296.0	52.204586	567.0
Jimbo	242.0	315.0	56.552962	557.0
Announcer	239.0	168.0	41.277641	407.0
Stephen	182.0	176.0	49.162011	358.0
Craig	163.0	163.0	50.000000	326.0
Clyde	121.0	198.0	62.068966	319.0
Jesus	156.0	156.0	50.000000	312.0

Figure 8: Deep Learning Model

	sarcasm_prediction	number of sentences
Character		
Crowd	0.391304	115
Kids	0.313869	137
Token	0.310469	277
Clyde	0.285714	315
The Boys	0.281818	110
Man	0.280952	210
Man 2	0.268293	123
Ike	0.265000	200
Kenny	0.257955	880
Man 1	0.257426	101
Craig	0.254601	326
Chris	0.250000	192
Mephesto	0.241135	141
Timmy	0.239544	263
Gerald	0.237560	623
Towelie	0.234848	132
Mark	0.232143	112
Satan	0.227723	202
Butters	0.226466	2592
Bebe	0.222727	220
Pete	0.216216	111
Yates	0.208333	120
Wendy	0.205832	583
Kyle	0.203926	7081
Jimmy	0.203704	594
Stan	0.201516	7652

Figure 9: Statistical Model

Deep Learning Model Generalization performance The rows are ranked by the number of said sentences, because a lot of characters only say one sentence in the whole show and, given it's funny, obtain a score of 100%. Here, with this classification, the main characters are first.

We see that Cartman is funnier than Stan, Kyle, but not Butters and Randy. Butters and Randy are a bit more anecdotic in the TV show but it makes sense that with fewer apparition they are overall funnier. Finally, Kenny has a low score, which aligns with the fact that he simply doesn't talk about the show because of his coat. Kenny inspires a comic of situation, more so than saying funny things, and our scorer reflects it.

Overall the scores make sense with the deep learning model.

Linear Model Generalization performance In Figure 9, sarcasm prediction columns corresponds to the percentage of sarcastic said sentences and rows are here sorted according to this column. In order to get rid of characters that says only a few sentences which are sarcastic, we put a threshold on the number of sentences that a character have to say for appearing in Figure 9.

We can see that the crowd and kids are the most sarcastic, which aligns to our perception as they do not speak a lot but represent an ironic opinion that comments the plot. For the main characters, they are ranked as follows (from teh most sarcastic to the less sarcastic): Kenny, Butters, Kyle, Stan, Cartman and Randy. Mostly, these results make sens except for the fact that Kenny is the first and that Cartman is among the last. For Cartman, as he is the one speaking the most, far from the others, it can be understandable that he has a lower score as his sarcastic comments are included in other comments that are not ironic.

5 Error analysis

5.1 Deep Learning

Error analysis for the deep learning part was done in 2 parts, visualizing attention and showing some false positive and false negative examples.

5.1.1 Visualizing the Attention mechanisms

Figure 10 shows the learned weights of attention. The weights very quickly in the learning process focused on, for each attention mechanism, a single word in the input sentence. Future work should include trying to get a less sharp attention behavior.

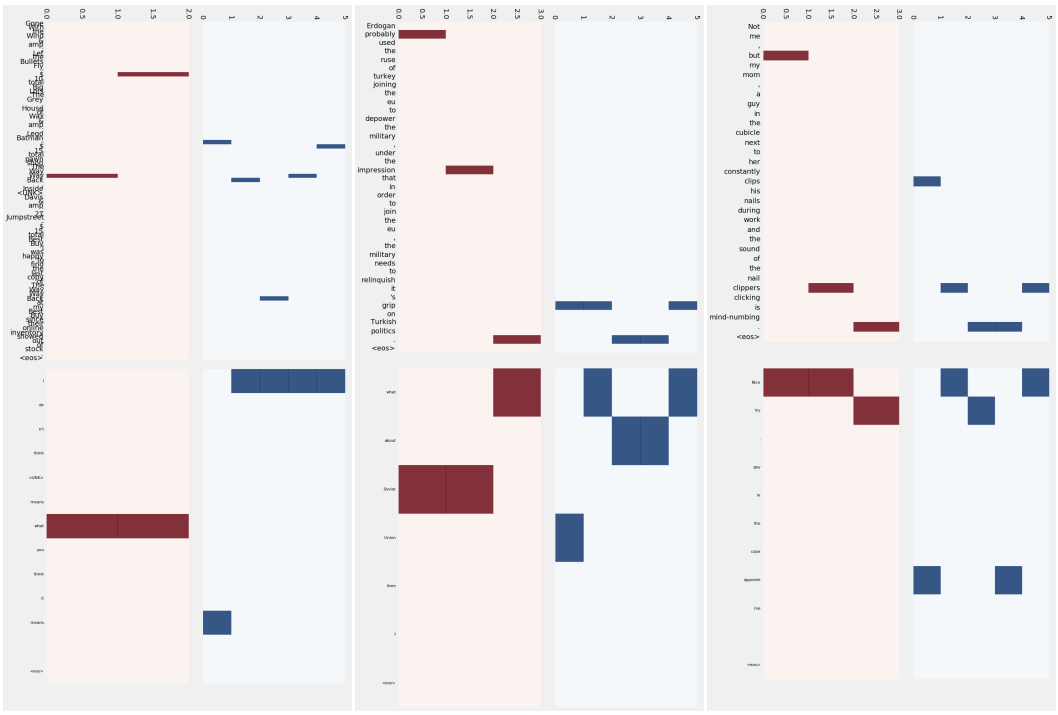

Figure 10: Visualizing learned attention weights. Blue : Self attention, Red : Conditional Attention

5.1.2 False-positive/negative examples

Previous Sentence	Current Sentence	Label	Predicted
Tomato tomatoes	And the overwhelming thought in my head on seeing this post as 1 is " WHO F***ING CARES ? ! "	0	0.62
But when you tune the game to the input device and give the reticle a small bit of resistance when moving over a target everyone 's all " oh no auto aim teh suck "	But there 's more skill in using a controller than a keyboard and mouse !	0	0.81
5 $ keys sells only for russia ip , and of course not only russian cheating in game , they have discomfort from cheaters too	I bought the game as a russian key for 4 csgo keys , please do n't do this PLEASE QQ	0	0.74

This shows a couple of examples where our model is, in our opinion, actually right. It's a great display of the caveats of learning sarcasm. It's often very subjective and thus quite hard to get a good training set. It might actually be easier to learn to predict sarcasm conditioned on the annotator.

6 Conclusion

Finally, we can say that we managed to achieve satisfactory results on the task of sarcasm detection. Indeed, we started with linear models that had good accuracy while having the advantage to be easily interpreted. Then, deep learning methods and especially attention got better results that are promising according to such a challenging task. In addition, using the South Park dataset, we have been able to evaluate the ability of our model to generalize on another dataset. According to our knowledge of South Park, it seems that it was performing pretty well.

References

[1] "A Large Self-Annotated Corpus for Sarcasm". 2017. URL: https://arxiv.org/abs/1704.05579.

[2] Cho et al. "Neural Machine Translation by jointly learning to align and translate". In: (2014). URL: https://arxiv.org/pdf/1409.0473.pdf.

[3] Vaswani et al. "Attention Is All You Need". In: (2017). URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=854DFFA350ABFD5D05171AF56922B036?doi=10.1.1.385.4410&rep=rep1&type=pdf.

[4] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[5] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: 9 (Dec. 1997), pp. 1735–80.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.