connotation_clashers at SemEval-2022 Task 6: The effect of sentiment analysis on sarcasm detection

Patrick Hantsch and **Nadav Chkroun**
University of Tübingen
{patrick.hantsch, nadav.chkroun}@student.uni-tuebingen.de

Abstract

We investigated the influence of contradictory connotations of words or phrases occurring in sarcastic statements, causing those statements to convey the opposite of their literal meaning. Our approach was to perform a sentiment analysis in order to capture potential opposite sentiments within one sentence and use its results as additional information for a further classifier extracting general text features, testing this for a Convolutional Neural Network, as well as for a Support Vector Machine classifier, respectively.

We found that a more complex and sophisticated implementation of the sentiment analysis than just classifying the sentences as positive or negative is necessary, since our implementation showed a worse performance in both approaches than the respective classifier without using any sentiment analysis.

1 Introduction

According to Cambridge dictionary, sarcasm is defined as the use of remarks that mean the opposite of what they say, in order to hurt someone's feelings or to criticize something in a humorous way. While detecting sarcasm can be very difficult even beyond the text level, it is a real challenge to detect sarcasm in textual data. When having conversations in person, it is easier for people to detect sarcasm because they have additional information in form of the speaker's emphasis of words, his facial expressions and body gestures. All of this information is not available in written language. Yet, detecting sarcasm from text becomes more and more important, since everyday life nowadays takes place on social media platforms to a large extent, unfortunately including things like hate speech in order to offend people. Sarcasm is often used as a tool for that (Frenda, 2018), so operators need ways to detect and remove offensive material expressed in a sarcastic way in large amounts of messages and posts released by millions of people every day.

The shared task (described in Abu Farha et al., 2022) consisted of three sub tasks:

- Sub task A: Determining if a given input text is sarcastic or not
- Sub task B: Determining if a given input text belongs to a ironic speech category and if so, to which one exactly
- Sub task C: Given two input texts, one being sarcastic and the other one being its non-sarcastic rephrase, determining which is the sarcastic one

We only participated in sub task A of the shared task. Our strategy was to additionally perform a sentiment analysis on the given text data, using the contradictory nature of a sarcastic statement, between the actual utterance and its true meaning/intent. For example in the sentence "I love how ill i became last night...", a sentiment analysis might recognize the clash of connotations between the positive connotated word "love" and the negative connotated fact of "being ill". Van Hee et al. (2018) also mentioned the inversing effect of using irony (a form of sarcasm) in a sentence on the overall sentiment of that sentence and even included this notion in form of a class called "Verbal irony by means of a polarity contrast" in one of their sub tasks of the task they provided for the SemEval competition in 2018.

In an earlier work described in Poria et al. (2017) a similar approach to our strategy was used, combining several Convolutional Neural Networks (CNNs), pre-trained on recognizing and classifying sentiment, emotion and personality features respectively, with a Support Vector Machine classifier for classification only. However, for the best of our knowledge there was no earlier work investigating the impact of sentiment analysis on sarcasm detection for linear models. Inspired by the previously

Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022), pages 945 - 950

mentioned work, we combined the result of our sentiment analysis with all other features subtracted from the given text and investigated the influence of sentiment analysis on the final sarcasm detection for both a linear model, as well as for a Deep Learning model, also comparing the performance of both approaches.

2 Training Data

According to sub task A, our system should classify an input text as either sarcastic, or non-sarcastic. If, for example, the string "I love how it rains for the seventh day in a row" was fed as input to our system, the output would be either "1" (sarcastic), or "0" (non-sarcastic). As training data, two collections of tweets were given, one in English and one in Arabic. In table 1 you can see the amount of sarcastic and non-sarcastic tweets in both datasets.

	English	Arabic
total amount of tweets	**3468**	**3102**
`sarcastic`	867	745
`non-sarcastic`	2601	2357

Table 1: Amount of sarcastic and non-sarcastic tweets in training data

3 System Overview and Experimental Setup

Both approaches we decided to test for solving the task were implemented in Python 3.8. In terms of sentiment analysis we decided to focus on English, for which we tested each of those approaches both combined with a sentiment analysis and without it. The predictions contained in our official submission were created by our Deep Learning model not being combined with sentiment analysis, for both English and Arabic.

3.1 Official Submission

We chose to build a CNN as a Deep Learning Model for sarcasm detection. CNNs are several layers of convolutions with non-linear activation functions like ReLU (Brownlee, 2019) or tanh applied to the results.

3.1.1 Preprocessing

All tweets were split into a sarcastic and a non-sarcastic tweet collection. We used the tokenizer class from Keras library (Chollet et al., 2015), which allows vectorization of a text corpus, by turning each text into a sequence of integers, while calculating the maximum number of words to keep based on their frequency. In addition, we split the training data into training set (80%) and validation set (20%) for hyperparameter optimization. We were setting each tweet a length, and padding or truncating it, based on the length of 100 words, respectively.

3.1.2 Training

In our model, we converted words to vocabulary indices. We did not use pre-trained embeddings. Instead, the embedding was done by the embedding layer of our model. In total, 6 layers were used, which are described in table 2.

name of layer	parameters
embedding	vocab_size=8879, embedding_dim=32, input_length=100
convolution 1D	filters=128, kernel_size=3, activation= "relu"
pooling 1D	-
Dense	units=100 activation="relu"
Dense	units=32 activation="relu"
Dense	units=1 activation="sigmoid"

Table 2: Description of the model's architecture

We compiled the model using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.01. The loss function for the optimizer was Binary Cross Entropy, which compares each of the predicted probabilities to the actual class labels which can be either 0 or 1. For the tuning, random search was used to find the optimal hyperparameters. This method sets up a grid of hyperparameter values and selects random combinations. We tuned the parameters with the respective values as shown in table 3.

In total, 5 different hyperparameter settings were tried by our random search.

3.1.3 Evaluation

According to the official metrics, which was F1-score for the sarcastic class, our CNN trained on the English data scored 0.2024, unweighted mean precision being 0.5016 and unweighted mean recall being 0.5029. All in all, our CNN achieved a

Description	Name	Values
learning rate of the optimization algorithm	learning_rate	0.01, 0.001, 0.0001
length of the convolution window	kernel_size	3, 5
size of the dense layer	units	32-128 (interval: 16)
number of convolution units	filters	32-256 (interval: 32)

Table 3: Hyperparameters and values to which the random search was applied

bad rank (37/43). The CNN trained on the Arabic training data achieved a F1-score of 0.3013, a unweighted mean precision score of 0.5647 and the unweighted mean recall was 0.5954. It ranked 23rd of 32 total submissions.

3.2 CNN Influenced by Sentiment Analysis

The preprocessing used for the CNN we combined with the sentiment analysis is the same as for the CNN which created our official submission.

3.2.1 Sentiment Analysis & Transfer Learning

We implemented sentiment analysis for sarcasm detection, used as a "transfer learning" method. The sentiment analysis assigns each tweet a label of either being "positive" or "negative". The sentiment analysis dataset we used was the IMDB movie review sentiment classification dataset of Keras. It consists of 25000 movie reviews, labeled as 1 ("positive") or 0 ("negative"). We split it into a training set consisting of 20000 of the reviews and a test set containing 5000 reviews.

First, we trained our CNN model on the training set for sentiment analysis. We saved the weights and then trained the model on the sarcasm detection training set that we created previously. It is important to note that the sentiment analysis scores were used in a way that although the model labeled each sentence as a float number between 0 to 1. The final labels (0 or 1) were given to each sentence according to its proximity to 0.5.

3.2.2 Training

In order to get a more fine-tuned model for the combination with the sentiment analysis, we changed the model's architecture. The reason for this change is a result that another random search, which we conducted after we applied the transfer learning, picked for our combined model. The same hyperparameters and values were tested as in our first random search described in section 3.1.2. We changed the hyperparameters accordingly, leading to the architecture depicted in table 4.

name of layer	parameters
embedding	vocab_size=88584, embedding_dim=32, input_length=100
convolution 1D	filters=128, kernel_size=3, activation="relu" kernel_regularizer=l2(0.001)
dropout	rate=0.1
convolution 1D	filters=128, kernel_size=3, activation="relu" kernel_regularizer=l2(0.001)
dropout	rate=0.1
pooling 1D	-
dropout	rate=0.1
Dense	units=32 activation="relu" kernel_regularizer=l2(0.001)
dropout	rate=0.1
Dense	units=1 activation="sigmoid" kernel_regularizer=l2(0.001)

Table 4: Description of the model's architecture for the transfer learning

3.2.3 Evaluation

Our CNN which was influenced by the sentiment analysis performed worse than the CNN on its own. The F1 score on the sarcastic class was 0.1679. Unweighted mean precision was 0.5109, unweighted mean recall 0.5117.

3.3 Comparing Results of both CNN's

Comparing the scores of both the CNNs shows that the CNN augmented by the sentiment analysis performed worse than the CNN on its own, as shown in table 5.

3.4 Linear Model Approach

In the following subsections we discuss our linear model based approach. The preprocessing steps, as

model	f1-score	precision (unweighted mean)	recall (unweighted mean)
CNN without transfer learning of sentiment analysis	0.2024	0.5016	0.5029
CNN including transfer learning of sentiment analysis	0.1679	0.5109	0.5117

Table 5: Both CNN model's scores on the English test set

well as the implementation of sentiment analysis differ from what we did in our Deep Learning based approach.

3.4.1 Preprocessing

One challenge we were confronted with was a huge amount of data imbalance. The English data consisted of only 25% sarcastic tweets, the Arabic data also of around 24%, while the rest of the data was non-sarcastic, respectively. To account for this in our linear model approach, in each language respectively, first, all tweets were split into a sarcastic and a non-sarcastic tweet collection. After shuffling both collections, we chose 867 tweets each, i.e all sarcastic tweets were used, for the further training process. We put 694 tweets each (80%) into the training set, while keeping the other 173 tweets (20%) of each category as held-out data for testing. This way, we had a guaranteed sarcastic/ non-sarcastic ratio of 50% each to avoid bias due to the previously mentioned imbalance. After shuffling both the training set and the test set again, we obtained our final datasets.

3.4.2 Feature Extraction

We extracted the features from our training data by using a simple count vectorizer, while applying tf-idf weighting, both as implemented in scikit-learn (Pedregosa et al., 2011). The vectorizer considered word n-grams from unigrams up to 4-grams. This feature extraction process was used in the grid search we performed for choosing a linear classifier as described in section 3.4.3, as well as for training both with and without sentiment analysis, described in section 3.4.5.

3.4.3 Model Choice

For the linear model approach we chose three different linear models and performed a grid search on them, trying to optimize F1-score for the sarcastic class on a held-out dataset (20% of all training data, see Section 3.4.1), to find the best hyperparameter settings for fitting on our training data. The three models chosen were a Naive Bayes Classifier, a Support Vector Machine model and a Random Forest classifier. The following hyperparameters of the respective models were included in the grid search:

- For Naive Bayes Classifier:

Description	Name	Values
Parameter for add-k smoothing	alpha	0.1, 1.0, 2.0, 5.0, 10.0, 20.0, 50.0, 100.0

- For Support Vector Machine

Description	Name	Values
Probability estimates for classification	probability	True, False
Kernel type	kernel	linear, poly, rbf, sigmoid
Kernel coefficient	gamma	scale, auto
Degree of the polynomial kernel function	degree	0-6 (interval: 1)

- For Random Forest Classifier:

Description	Name	Values
Number of decision trees in the forest	n_estimators	10, 20, 50, 100, 200, 300, 400, 500
Minimum number of samples needed to split a node (make a decision)	min_samples_split	3, 5, 10, 20, 30, 50
Maximum depth of a tree	max_depth	None, 3, 5, 15, 25, 50, 100
Maximum number of features considered when looking for each split	max_features	3, 5, 10, 20

The F1-score of the sarcastic class was calculated using cross-validation. Features of the used

training data during the grid search were extracted as described in section 3.4.2. Both the grid search and the classifiers were implemented via scikit-learn. We chose the classifier which achieved the highest F1-score for the sarcastic class with its best hyperparameter settings respectively, obtained from the previously mentioned grid search. This classifier was the Naive Bayes Classifier. However, after making predictions on the organizer‘s test data, we decided to finally use the Support Vector Machine model, since almost all test instances were classified sarcastic by the Naive Bayes Classifier, while the predictions made by the Support Vector Machine model were mixed and seemed more reasonable. The best hyperparameter setting found for the Support Vector Machine classifier were its default values for all possible hyperparameters, as defined in scikit-learn.

3.4.4 Sentiment Analysis

To implement the sentiment analysis for English we used a pre-trained model from the Flair NLP library (Akbik et al., 2019). Flair is a NLP framework providing a lot of different models for several common NLP tasks, including sentiment analysis. We can feed an input text to the pre-trained Flair model to get a classification for the text being positive or negative, as well as a score indicating how confident the model was about the classification between 0.5 and 1.0.

Our idea for how to use this result to help us solve the task at hand was to feed each input sentence into the Flair model and obtain a sentiment prediction, "positive" or "negative", together with the model's confidence score. We then assign one of six categories to the respective sentence. We created three categories, each resembling a certain range of the model's confidence score for both the positive and negative class. Thus, we obtained the following six categories:

confidence score	**classified "positive"**	**classified "negative"**
> 0.95	very positive	very negative
0.75 - 0.95	quite positive	quite negative
< 0.75	rather positive	rather negative

Depending on the sentence's category we then created a one-hot encoded vector. For example, if a sentence would be categorized as very positive, the respective vector would be "[1 0 0 0 0 0]", with each integer resembling one of the categories and its value showing if it is the sentence's category (1) or not (0).

Applying this to each input sentence results in a matrix, which we attached in the training process (see section 3.4.5) to the feature matrix that was created by our feature extraction described in section 3.4.2.

3.4.5 Training

For training in both cases (with and without sentiment analysis), we preprocessed our training data as described in section 3.4.1 and obtained our feature matrix as described in section 3.4.2.

For training with sentiment analysis, we attached the sentiment matrix, obtained in the process described in section 3.4.4, horizontally. That means the first row of the sentiment matrix was appended to the first row of the feature matrix etc. such that the final matrix resembles all input sentences of the training set (rows) with all extracted features, including the result of our sentiment analysis (columns).

To get our final predictions we fit our Support Vector Machine classifier on our respective feature matrix, depending on training with or without sentiment analysis.

3.4.6 Results

The linear model trained without the input of our sentiment analysis scored 0.2738 according to the official metrics, which was F1-score for the sarcastic class. The model trained with the input of the sentiment analysis scored only 0.2721, so the model influenced by the sentiment analysis performed slightly worse.

model	**f1-score**	**precision (unweighted mean)**	**recall (unweighted mean)**
SVM with sentiment analysis	0.2721	0.5318	0.564
SVM without sentiment analysis	0.2738	0.5336	0.5673

Table 6: Scores of the linear model with and without sentiment analysis for the English test set

4 Conclusions

The Support Vector Machine model (SVM) performed better than the CNN both when sentiment analysis was included, as well as on its own. The SVM without sentiment analysis also showed the

highest F1 score of all four tested models. However, in general our results show that our ways of implementing sentiment analysis even had a negative impact on the classification for sarcasm on both systems.

One reason for the bad performance of the sentiment analysis could have been that we performed the sentiment analysis on the input sentences as a whole for both training and classifying. In order to recognize a contradiction within one sentence, as explained in Section 1, improvements might be achieved if the sentence is split in several sub parts and sentiment analysis is performed on those parts. This way, positive and negative classifications with high confidence scores might be observed, clearly indicating the aforementioned clash of connotations. Potential challenges that could come up in that case are to determine where to split the sentences exactly, or deciding which parts of even the sub parts the sentiment analysis should be performed on, since not all word types convey sentiment (e.g. stop words could be considered neutral). To account for this, alternatively to our approaches, a sentiment lexicon could be used to calculate scores on sub parts of sentences.

Even without splitting the input sentences, we could have implemented our idea better for both approaches. For example, for the Deep Learning approach, sentiment analysis datasets might be more useful, if their genre is the same or closer to our training data. An additional way to adjust our implementation for this approach would be to shift our decision boundary, classifying sentences contradictory, if output values of the sigmoid function between for example 0.35 and 0.65 can be observed, since clearly positive and negative sub parts might lead to a rather neutral sentiment in total.

Similarly, for the linear model approach we could have weighted the "rather positive" and the "rather negative" category as more important, compared to the more extreme categories.

Acknowledgements

We would like to thank Çağrı Çöltekin for his advice and support while creating this work. We would also like to thank everyone who reviewed an earlier version of this paper and provided helpful comments and suggestions for improvement.

References

Ibrahim Abu Farha, Silviu Oprea, Steven Wilson, and Walid Magdy. 2022. SemEval-2022 Task 6: iSarcasmEval, Intended Sarcasm Detection in English and Arabic. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Association for Computational Linguistics.

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. Flair: An easy-to-use framework for state-of-the-art in natural language processing. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59.

Jason Brownlee. 2019. A gentle introduction to the rectified linear unit (relu). https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

François Chollet et al. 2015. Keras. https://keras.io.

Simona Frenda. 2018. The role of sarcasm in hate speech. pages 14–16.

Cynthia Van Hee, Els Lefever, and Véronique Hoste. 2018. Semeval-2018 task 3: Irony detection in english tweets. In *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*. Association for Computational Linguistics.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. 2017. A deeper look into sarcastic tweets using deep convolutional neural networks.

Cambridge University Press. 2022. Cambridge dictionary.