Sentiment Analysis on Twitter Data

July 29, 2023

Friedrich Answin Daniel Motz
daniel.motz@uni-jena.de
Friedrich-Schiller-Universität Jena

1 Introduction

From a philosophical standpoint, it's an almost comic endeavour to teach a machine to accurately interpret human language and which feelings they intend to induce in their fellow human beings. It gets more absurd upon reflection, for which purposes sentiment analysis is being conducted: mostly product success metrics, which track specific groups of users, possibly over time, to reinforce the relationship between a brand and said users.

Data scraped from microblogging plattforms (but also social media and chat apps) consist of abbreviated, exaggerated, highly contextual (e.g. a response to another tweet), informal and misspelt language, which makes it difficult to classify.

1.1 In this paper

This work's goal is to examine which methods are useful in achieving a higher accuracy on the given dataset.

2 Methods

Unless stated otherwise, an `sklearn LogisticRegression`-Classifier and Word2Vec [1, 2] was used.

2.1 Considerations

Previous research suggests, that part-of-speech features might not be suited for this task [3], so this work only uses other methods suggested, namely: removal of stop words, lemmatisation, replacing capitalised words with emphasising prepositions and their lowercase form, removal of links and mentions. A priori, this should put more focus on which words are being used. This stands in comparison to understand what is being said.

2.2 No Pre-Processing

As a baseline the data was used as is (with all special characters, quotes, @-signs, …). **Score = 0.736**

2.3 Pre-Processing

Extensive Pre-Processing. The first approach was to filter everything except for words which might contain a hint sentiment. This involved removing links, user mentions (like `@username58`), special characters (everything that's not alphanumeric) and also stopwords (for which NLTK's list of stopwords[4] was used) like 'of', 'whom', 'had', 'same' and 'too'. All words were transformed to lowercase. Then the words were lemmatised to make messages more comparable. There was no sign that prepending emphasis words like 'very' or 'much' to words in caps or with repeating letters (e.g. "NO" and "looooove"). The score with prepended emphasis words was about **0.724**. Without emphasis: **Score = 0.731**.

Medium Pre-Processing. In this approach only URLs, mentions, and special characters were removed. **Score = 0.732**

Issues with Pre-Processing. Plenty of manual adjustments are required to get satisfying results. Furthermore, poorly written regular expressions might create a bias themselves. Some occurences might be filtered, while others are not, of course, in disregard of the target sentiment. *Lemmatisation* and *porter stemming* [5] have not proven to be helpful in reducing the task complexity. No meaningful improvement could be noted.

2.4 Word Mapping

High Similarity. Such low scores lead to believe, that the mapping is just not differentiating enough or that the messages simply cannot be very easily be differentiated, without interpreting the meaning (e.g. with a LLM) of what is being said. Using UMAP [6] Figure 1 and Figure 2 show 2D-reductions of the mappings.

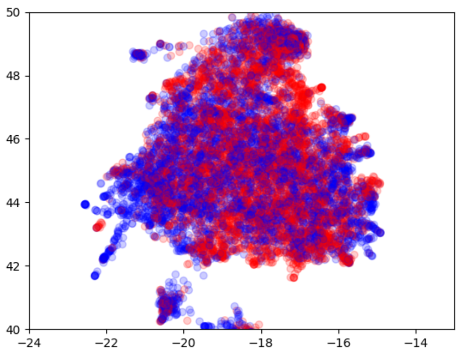

Figure 1: Word2Vec of cleaned text

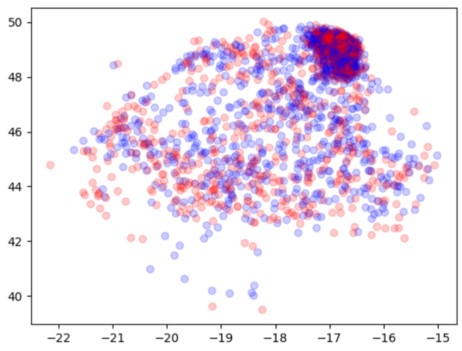

Figure 2: Word2Vec unprocessed text

The images use transparent points which illustrate how densely the vectors are distributed. Figure 2 shows a heap of alternating messages in the top right corner. However, both Figure 2 and Figure 1 show how evenly distributed the text messages are.

Lexicon and Rule-based approaches. The VADER [7] score consist of *positive*, *negative* and a *neutral* score $\in [0, 1]$ where higher is more intense. VADER also provides a compound score, the calculation of which can be taken from the documentation. The positive, negative and neutral score were used as input vectors, whcih sadly only performed at **Score $\approx$ 0.664**. However, a combination of Word2Vec with VADER resulted in a minimal increase of ≈ 0.011 from 0.741 to **Score = 0.752**. Each Word2Vec mapping of a sentence (a vector) was additively moved by $c \cdot s_{\text{vader}}$, where $c \in [100, 1000]$ was found to be optimal close to 1000, and s_{vader} is the compound VADER score. The hope was to dilute the heaps of words, which was, in part, achieved. This can also be seen in Figure 3 where there are two homogenous regions.

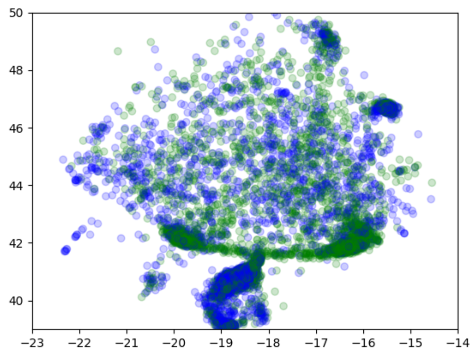

Figure 3: VADER x Word2Vec of cleaned text

Complexity Reduction. In addition to stemming, removing words below a certain frequency proved helpful in reducing the task's complexity and accuracy, with a minimal **Loss** $\approx$ **0.002**, but time improvements of about 10%.

Removal of neutral words. Based on VADER's evaluation, words with a neutral score ≥ 0.1 were left out. This method showed to remove stop words, which we already established were helpful. Table 1 shows messages which could not be matched correctly. For example "yay yay" does indeed sound like a message with good intent. However, the original text says:

> Woke up on time today. Yay. Maths first, then French! not so 'Yay.'

This illustrates how important context of certain words is. And while extensive pre-processing might reduce complexity, it negatively impacts accuracy, **Score** $=$ **0.66**.

2.5 Ensemble Classifiers

In the hope that homogenous areas produced by the combination of Word2Vec and VADER could be better predicted by Decision Trees or ensembles, both methods were tested the provided training data with a simple train-test split. Figure 4 shows how VADER gives slight increases in test accuracy, while not impacting train accuracy. It can be observed, that overfitted models, such as *7 Tree Ensemble* increase accuracy by ≈ 0.025.

Generally, *9 Logistic Regression* provided the best results without any overfitting. Technically speaking, *6 Extra Trees* had the highest accuracy on test data, while however massively overfitting.

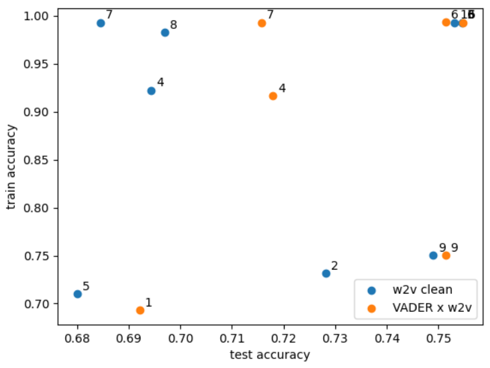

Figure 4: Comparison of accuracies of different classifiers on Word2Vec and Word2Vec with VADER

3 Summary

The methods used did not prove effective. The highest improvement of ≈ 0.012, which was achieved by combining VADER and Word2Vec (on cleaned data), is very low compared to the effort. It should be explored how other text mapping methods behave on the data set.

4 Appendix

target	**text**
1	serious
0	yay yay
0	hope best
1	help
1	like
1	laugh
1	cut hand laugh
0	true die
0	okay better

Table 1: Mismatched messages taken from the training set

Classifier	**Number**
AdaBoostClassifier n=10	1
AdaBoostClassifier n=100	2
AdaBoostClassifier n=50	3
Bagging n=10	4
DecisionTreeClassifier d=10	5
ExtraTrees n=500	6
ExtraTrees n=10	7
Gaussian Naive Bayes	8
LogisticRegression	9
RandomForest n=10	10
RandomForest n=100	11
RandomForest n=500	12
RandomForest n=1000	13

Table 2: Classifiers Reference, where n is the number of estimators and d is `max_depth`

For full test results, look for `results.csv` in the submission folder.

Bibliography

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[2] "Gensim." https://radimrehurek.com/gensim/models/word2vec.html

[3] E. Kouloumpis, T. Wilson, and J. Moore, "Twitter sentiment analysis: The good the bad and the Omg!," 2011.

[4] "Natural language toolkit." https://www.nltk.org/index.html

[5] "Porter stemmer." https://tartarus.org/martin/PorterStemmer/

[6] "Uniform manifold approxiation and projection." https://pypi.org/project/umap-learn/

[7] C. Hutto, and E. Gilbert, "A parsimonious rule-based model for sentiment analysis of social media text," 2014. [Online]. Available: https://github.com/cjhutto/vaderSentiment