Using part of speech in tweet classification

Kaggle group name: POS-itive

Richard Wigren
17-909-383

Michiel Baptist
17-907-544

Gorm Rødder
17-908-963

***Abstract*—Sentiment analasys is an important field, which has gained in popularity. As social media becomes more and more an important factor in marketing, so does sentiment analasys. In this paper we present a sentiment analasys technique which improves on a classical model by using part of speech information.**

I. Introduction

The task of sentiment analysis consists of extracting subjective information, such as opinions and attitudes, from language [1]. It has been explored extensively in previous works [1], [2]. In this paper we present a supervised method for classification of short pieces of text in the form of tweets into either positive or negative sentiment. In section II we present our method in an incremental fashion and present three baseline methods against which we compare our model. In section III we present results from experiments we performed on the models described in section II. Finally in section IV we discuss the results of the experiments.

II. Models and Methods

In this section, after some definitions, we present three baseline methods, our vanilla model, and several improvements attempted incrementally. Note that our baselines are not taken from the programming exercises.

Tweet tokenization: In the following models we assume a tweet consists of a string. Before classification we usually have to tokenize the tweet. We consider a tokenization as a deterministic mapping:

$$t : \mathbb{S}_{140} \to \mathbb{T}^{l(i)} \tag{1}$$

Where $\mathbb{S}_{140}$ is the set of all possible strings of size less than 140 characters, $\mathbb{T}$ is the vocabulary with $|\mathbb{T}| = M$. $l(i)$ is an integer depending on the string being mapped. For notational purposes we write $t(x) = w_1, \ldots, w_n$ Not all models presented use the same tokenization function. A simple example of a tokenizer is the space tokenizer which splits a string at the spaces.

Word embedding: Word embeddings form a core concept in modern natural language understanding. Many models have been proposed in an attempt to capture semantic meaning of a word as a continuous vector [3]–[5]. We consider a word embedding as a deterministic mapping:

$$e : \mathbb{T} \to \mathbb{R}^l \tag{2}$$

where $\mathbb{T}$ is the vocabulary, l is the length of the real vector or the embedding size.

A. Baseline: Naive Bayes

The Naive Bayes classifier is well known and has been successfully applied in the past, such as spam filtering and text classification [6], [7]. The main idea of the naive Bayes classifier is that tweets with a positive sentiment might have a very different distribution over words than tweets with a negative sentiment. Concretely we consider the model:

$$P(c|w_1, \ldots, w_n) = \frac{P(c)P(w_1, \ldots, w_n|c)}{P(w_1, \ldots, w_n)} \tag{3}$$

Where $w_1, \ldots, w_n$ are the tokens of a tweet and c is the class to which the tweet belongs. In this case either positive or negative. Making the naive Bayes assumption we assume the tokens w_i are independent given the class to which it belongs, i.e.:

$$p(w_1, \ldots, w_n|c) = \prod_{i=1}^{n} P(w_i|c) \tag{4}$$

Assuming $W|c$ is a categorical random variable, we estimate $P(W = w_i|c = j) = \theta_{ij}$ using maximum likelihood with α-smoothing as in [7], i.e. we use the following estimates:

$$P(W = w_i|c = j) = \hat{\theta}_{ij} = \frac{N_{ij} + \alpha}{\sum_{i=1}^{M}(N_{ij} + \alpha)} \tag{5}$$

Where M is size of the vocabulary. The Bayes optimal classifier then becomes:

$$\hat{c} = \arg\max_{c} \; P(c|w_1, \ldots, w_n) \tag{6}$$

The naive Bayes classifier was implemented using the MultinomialNaiveBayes implementation of scikit-learn [8]. Our α value is set to 1.

B. Baseline: Averaged word embedding SVM

In order to leverage the semantic meaning captured by word embeddings we propose the following classifier as a baseline:

1) Tokenize the tweet: $t(x) = w_1, \ldots, w_n$
2) Embed the tokens: $e(w_i) = v_i \quad \forall i \leq n$
3) Average the word embeddings: $a = \frac{1}{n}\sum_{i=1}^{n} v_i$
4) Classify the tweet: $\hat{c} = c(a)$

Where c is a classifier. We use Stanford's GloVe embeddings trained on Twitter data [5] of size 200 and the implementation

of Random Forests from scikit-learn [8] as the classifier. Note this baseline has been suggested on the tutorial slides[1].

C. Baseline: Google sentence embeddings

In a similar vein as the previous baseline we attempt to capture semantic meaning using vectors of real numbers. However, we attempt to do this for the tweet entirely by using Google's Universal Sentence Encoder embeddings [9] and classifying the semantic meaning in the form of a continuous vector as either positive or negative.

The Google sentence embedding is a deterministic mapping $\mathbb{S} \rightarrow \mathbb{R}^{512}$, we then train a feed forward neural network classifier. Figure 1 shows the general architecture of this baseline. We attempted two different feed-forward architectures to classify the sentence embeddings:

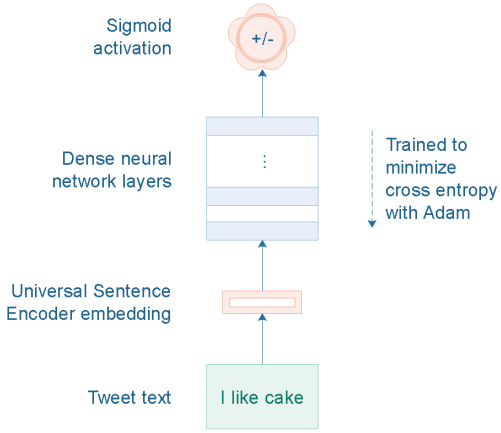

Fig. 1. The general structure of the sentence embedding baseline. Image created using Edraw Max[10].

1) 4 hidden layers of size 256, 256, 128 and 64. Each hidden layer has sigmoid activations. Output of size 1 with sigmoid activation.
2) 5 hidden layers of size 256, 256, 128, 128 and 128. Each hidden layer has sigmoid activations. Output of size 1 with sigmoid activation.

We use Tensorflow [11] to implement the different feed forward architectures. As mentioned previously we use Google's sentence embedding [9] to embed the tweets, their code has been made publically available on Tensorflow Hub[2]. We use Tensorflow's Adam optimizer to minimize for cross entropy. Section III shows results for different feed-forward architectures.

D. Model and Improvements

LSTM recurrent neural networks [12], [13] have been successfully applied in many areas when dealing with sequential data. Examples of good performance of LSTM networks can be found in machine translation [14], language modeling [15] and question answering [16]. In an attempt to leverage the predictive power of LSTM networks we propose to use a simple LSTM network as our vanilla model.

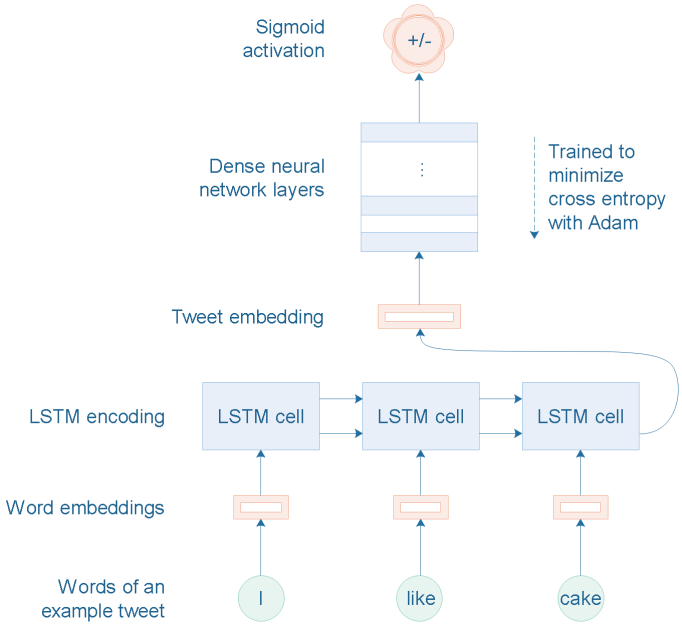

Fig. 2. The vanilla LSTM model architecture. Image created using Edraw Max[10].

Parameter	Value
Embedding	May vary
Cell layers	1
Cell Hidden size	128
Dense layers after output	2
Dense layer sizes	128, 128
Dense layer activations	Sigmoid, Sigmoid
Output layer size	1
Output activation	Sigmoid
Objective	(binary) cross entropy

TABLE I
OVERVIEW OF THE PARAMETERS OF THE VANILLA LSTM MODEL.

Figure 2 shows the vanilla architecture of an LSTM network, while table I shows the parameters used. The remainder of this section we discuss and justify possible improvements we attempted on the vanilla model described above.

Different embeddings: In order to understand the impact of embedding size of pre-trained embeddings we try embeddings of different sizes.

Parameters: In order to have consistency we use Stanford's pre-trained GloVe embeddings trained on Twitter data [5], of sizes 25, 50, 100 and 200 respectively. Results of the different sizes are shown in section III.

Principal Component Analysis (PCA): The core idea of PCA is to represent a set of vectors in a high dimensional space in a lower dimensional space. Doing so one can try and reduce statistical noise the data may contain.

Parameters: In an attempt to de-noise the high dimensional

[1]https://github.com/dalab/lecture_cil_public/blob/master/exercises/ex6/tutorial06.pdf

[2]https://tfhub.dev/google/universal-sentence-encoder/2

word embeddings we performed a PCA on Stanford's Twitter word embeddings of size 200 and only keep the 25 highest principal components. We then compare the results of using Stanford's Twitter word embeddings of size 25, results of this comparison can be seen in section III.

Part of Speech (POS): Word embeddings are a good way to capture semantic meaning on a per word basis. However, words might have similarities based on what type of word it is. For example a negatively sentimented tweet might on average contain more adjectives than a positively sentimented tweet. Additionally a word might change its meaning or impact based on its type or type of neighboring words. An example of this scenario lies in the sentences *I love cake!* and *I love you!*, *cake* being a noun and *you* a pronoun, one could argue that *love* has a substantially stronger impact in the latter case[3].

Following this thought process we attempt to improve the vanilla model by improving the word embeddings. We follow a similar approach as [17]. More concretely we improve the word embedding by first POS-tagging the tweet. Doing so every token of the tweet is associated with a part of speech, e.g. noun, verb, adverb, adjective,...Every POS tag is then associated with a so-called POS embedding. To form the improved embedding we simply concatenate each word embedding with its respective POS embedding. Figure 3 shows the overview of the POS model.

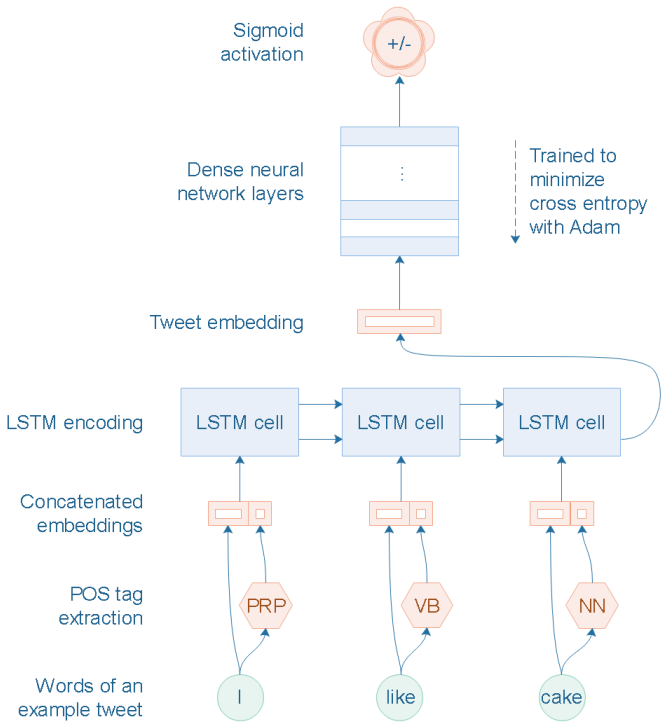

Fig. 3. The architecture of the POS LSTM model. Image created using Edraw Max [10].

[3]We make no strong claims, further research is needed.

Parameters: We use NLTK's [18] POS-tagger to tag the data. To embed the POS tags we use randomly initialized embeddings of size 50. The available POS tags are described by the Penn Treebank [19]. As word embeddings we use Stanford's pre-trained GloVe embeddings trained on Twitter data of size 200.

For our implementation of the vanilla model and extensions described above, we use Tensorflow [11]. We use NLTK [18] to tokenize the data. We use Tensorflow's Adam optimizer to minimize for cross-entropy for all models. The learning rate for all models is 0.001. During training we use dropout with a rate of 0.5.

III. Experiments and results

In this section we present results of different experimental setups.

A. Baselines

In the following we present the results of the baselines described in section II. In order to have a consistent and fair comparison of the baselines we use the same seed, we use the same split of 90% of the full training set as training data and 10% of the full training data as validation data. We show the test accuracy of the baselines we have tested using the test set, the results are available on Kaggle. Table II shows a summary of the results.

TABLE II
A summary of the baseline accuracies.

Baseline	Validation accuracy	Test accuracy
Naive Bayes	73.4	71.7
Average embedding	75.3	69.2
Sentence embedding, architecture 1	82.2	79.9
Sentence embedding, architecture 2	81.7	79.6

B. Model and improvements

Here we present the results of experimental setups of our vanilla model and the improvements suggested in section II. We use the same seed for each experiment, we use the same split of 90% of the full training set as training data and 10% of the full training data as validation data. The results shown are results of the best achieved validation loss for each model respectively. As a result not all models have trained for an equal amount of epochs. Figure III shows the results for the vanilla model and improvements **Important: Read the note at the end of the paper in section VI.**. For a closer comparison

TABLE III
Summary of the vanilla model and improvements.

Model	embedding	Val. accuracy	Test accuracy
Vanilla	Twitter GloVe 25 dim	86.3	85.4
Vanilla	Twitter GloVe 50 dim	87.2	86.2
Vanilla	Twitter GloVe 100 dim	87.6	87.0
Vanilla	Twitter GloVe 200 dim	88.0	87.0
Vanilla	PCA 25 components	85.6	NA
POS-model	Twitter GloVe 200 dim	88.1	86.6

of the vanilla model and the POS model we trained the models

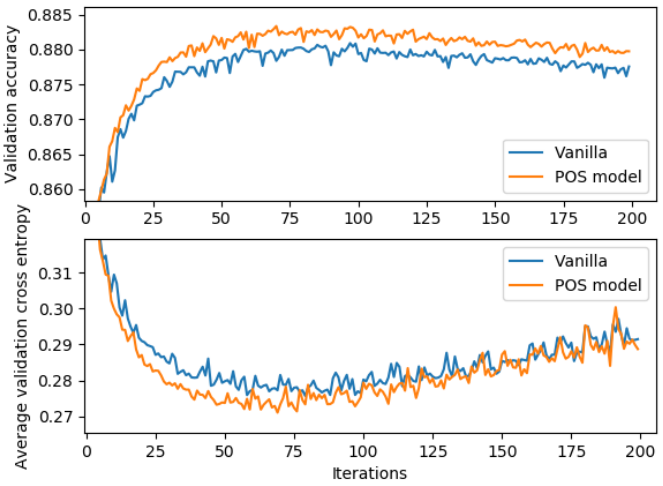

Fig. 4. Comparison of vanilla model and POS model on validation accuracy and loss.

and calculate the validation accuracy and loss in intervals while training. Concretely we calculate validation accuracy and loss 10 times per epoch and train for 20 epochs each. Both models use Stanfords GloVe embeddings trained on twitter data of size 200. The results can be seen in figure 4.

IV. Discussion

From table II we can see that the simple naive Bayes baseline managed to obtain a validation and test accuracy in the 70%. For such a simple model this baseline performed very decently. However, naive Bayes is not able to capture the structure of tweets. A positive tweet with a negation such as *don't worry, I don't hate cake!* can easily be interpreted as negatively sentimented since *worry* and *hate* are more likely to have a higher probability of occurring in a negatively sentimented tweet than positive. A similar performance is obtained by the second baseline, again scoring in the 70% range. This baseline again does not take into account the structure of the tweet in any way. Additionally this baseline is more prone to overfitting as can be seen by the test accuracy. From the same table however we see the sentence embedding baseline obtained an even higher validation accuracy. Both architectures managed to score in the 80% which is substantial for a baseline. The Google sentence embeddings in this case are somewhat more able to capture tweet structure than the naive Bayes baseline. However this model suffers tremendously from memory and computational related issues.

From table III we can clearly see the positive effect of embedding size on the validation accuracy. This shows that in some sense the semantic meaning of words are captured more accurately in higher dimensions. However the gain in accuracy seems to scale logarithmically with the increase in dimensions, meaning one has to make a trade-off between embedding dimensions and computational time, model complexity and validation accuracy.

We can see in an attempt to de-noise the data using a PCA the word embeddings became worse compared to Stanford's pre-trained Twitter embeddings of size 25. We assume this is the case since, while a dimension of 200 indeed captures more information about words, when applying PCA we keep less information than Stanford's pre-trained embedding of size 25.

However we are able to see a slight improvement when using the POS model over the vanilla model. From figure 4 we can see a consistent improvement in validation accuracy of the POS model over the vanilla model. However since the POS model has a higher model complexity it becomes more prone to over-fitting, as can be seen by the validation loss plot in 4. Here the validation loss of the POS model increases faster than the vanilla model. However considering the consistent improvement in validation accuracy and the lower validation loss, we conclude there is a definite improvement using the POS model over the vanilla model. As a result we conclude there is information in the POS tags of short texts.

V. Summary

Sentiment analysis is an important task in natural language understanding. In this paper we attempt to improve a classical LSTM classifier model by improving word embeddings. We do this by including information about the part of speech (POS). We concluded a noticeable increase in validation accuracy and a decrease in validation loss.

VI. Final Note

Please note, the validation accuracies of the POS model in table III is not the same as the validation accuracy of the model with chich we made the Kaggle predictions. This is because we failed to notice we commented out where we set the random seed. As a consequence we are not able to reproduce the validation accuracy and the test accuracy by the same model. Instead we have trained 2 separate models.

References

[1] B. Liu, "Sentiment analysis and subjectivity," in *Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca*, 2010.

[2] Z. Madhoushi, A. R. Hamdan, and S. Zainudin, "Sentiment analysis techniques in recent works," in *2015 Science and Information Conference (SAI)*, pp. 288–291, July 2015.

[3] Y. Li and T. Yang, "Word embedding for understanding natural language: A survey," 05 2018.

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[6] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. D. Spyropoulos, and P. Stamatopoulos, "Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach," *CoRR*, vol. cs.CL/0009009, 2000.

[7] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[8] 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, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y. Sung, B. Strope, and R. Kurzweil, "Universal sentence encoder," *CoRR*, vol. abs/1803.11175, 2018.
[10] https://www.edrawsoft.com/, "Edraw max."
[11] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.
[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.
[13] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, pp. 2451–2471, 1999.
[14] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.
[15] D. Shi, "A study on neural network language modeling," *CoRR*, vol. abs/1708.07252, 2017.
[16] D. Wang and E. Nyberg, "A long short-term memory model for answer sentence selection in question answering," in *ACL*, 2015.
[17] S. M. Rezaeinia, A. Ghodsi, and R. Rahmani, "Improving the accuracy of pre-trained word embeddings for sentiment analysis," *CoRR*, vol. abs/1711.08609, 2017.
[18] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, (Stroudsburg, PA, USA), pp. 63–70, Association for Computational Linguistics, 2002.
[19] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, pp. 313–330, June 1993.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Using part of speech in tweet classification

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	First name(s):
BAPTIST	MICHIEL
RØDDER	GORM
WIGREN	RICHARD

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	Signature(s)
HOEGAARDEN, BELGIUM ; 06/07/2018	Michiel Baptist
TRONDHEIM, NORWAY ; 06/07/2018	Gorm Rødder
STOCKHOLM, SWEDEN; 06/07/2018	Richard Wigren

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.