Twitter Sentiment Analysis

Mehdi Mezghani - Khalil Achache - Youssef Mamlouk

Abstract

In this paper, we propose several methods for training a model that can predict whether a tweet message originally contained a happy or sad smiley, based only on the text of the tweet. We investigate different machine learning (Logistic Regression), natural language processing (TF-IDF, CBOW, Skip-grams), and deep learning techniques (MLP, LSTM, Transfer Learning) to perform this task, and evaluate their performance in terms of accuracy. Our results show that performing transfer learning on BERT-based models while retraining all the layers gives the best results; in particular with BERTweet which gives a $91,4\%$ *accuracy on AIcrowd.*

1 Introcution:

Polarity classification is a crucial task in natural language processing, particularly when applied to social media platforms such as Twitter. On these platforms, users often express their opinions and emotions in short, messages, making it challenging for automated systems to accurately identify the sentiment behind them.
Twitter presents a unique challenge due to the informal and abbreviated nature of its messages. Traditional natural language processing techniques, which are typically trained on longer, more formal texts, can struggle to accurately classify the sentiment of tweets. Our model addresses this challenge by using a carefully designed set of features and a robust machine learning algorithm using a large dataset of 2.5 million tweets as a training data.

2 Data Preprocessing:

Data preprocessing is an essential step in natural language processing (NLP) when working with tweets. These messages are often informal, abbreviated, and contain slang, emojis, and other types of unconventional language, which can be difficult for NLP algorithms to process. Preprocessing helps to transform the raw text of tweets into a more structured and standardized form that is better suited for NLP algorithms.

A) Word Normalization:

First of all, we perform a **word-based tokenization**, which is a common approach that facilitate the manipulation of the text. By breaking down the tweet into smaller, more manageable units, our model can more easily identify patterns, relationships, and meaning within the tweet.
Case folding is beneficial when performing NLP on tweets because it helps to reduce the number of features and thus the performance of the algorithms. Tweets often contain words that are capitalized for emphasis, which can be difficult for NLP algorithms to process. By converting all words to a standard case, case folding reduces the number of unique words in the dataset.
Furthermore, we decided to **unpack contractions**, which involves expanding shortened forms of words (`e.g. "can't" → "can not"`) in order to transform the text into a more standardized and consistent form. This can make it easier for the algorithms to identify the structure and meaning of the text.
Besides, we use *ekphrasis* to **normalize** dates, numbers, money, etc. We map :

- `5` → ⟨`number`⟩
- `12/11/22 or 12 December 2021` → ⟨`date`⟩
- `12:30` → ⟨`time`⟩

B) Language specificities :

When performing natural language processing (NLP) on tweets, it is important to carefully address the unique challenges posed by this type of text data.
In particular, we had to pay attention to **elongated words**, which are words that have repeated letters to add emphasis. Thus, we used **spelling correction** based on probabilistic methods to transform them into a more standardized form.
Additionally, we **replaced slang** words with their standard equivalents (`e.g. "dunno" → "I do not know"`). This helps to reduce the number of unique words in the dataset.

C) Lemmatization :

We decided to perform **lemmatization**, which is the process of reducing words to their base form, also known as the lemma (`e.g. "was" → "be"`).
In fact, many words in a text dataset can have different forms, such as singular and plural nouns, or different verb tenses. These different forms can increase the number of unique words in the dataset, which can make the algorithms more computationally expensive to run.

We purposely did not remove the stop words since we noticed that our different models were slightly less accurate when doing so. It seems that they contain important information that is useful for determining the overall meaning and context of the tweets.

3 Regression using TF-IDF & N-grams:

In order to tackle our classification problem we decided to vectorize our input data using a **TF-IDF** matix and then feed it to a machine learning algorithm.
The underlying concept of TF-IDF is that words that are more important to a document should have a higher weight in the document representation. This is accomplished by multiplying the term frequency, which is the number of times a word appears in a document, by the inverse document frequency, which is a measure of how common the word is across all documents in the collection.
The motivation for using TF-IDF is that common words, such as "the" and "and," are not very useful for determining the sentiment of a tweet. These words appear frequently in most tweets and do not provide much information about the specific content of the tweet.

We combined this method to **N-grams**. This technique is also useful for sentiment analysis on Twitter because it can capture the context in which words are used. For example, if a tweet contains the word "sad" but also contains the 2-gram "not sad," it is more likely that the sentiment of the tweet is negative. We did a cross validation to determine the range of n-grams to consider **Figure 1**. In addition, we decided that only the grams that appears in at least 10 tweets will be added to the TF-IDF matrix. This prevents the explosion of the number of features to be fed to the machine learning algorithm, and filter the typos that made it through preprocessing process.

Finally since it is a classification problem we fed our TF-IDF matrix to a regularized logistic regression algorithm. More precisely we used the implementaion by scikit-learn with : `max_iter = 10000 and C=4`.

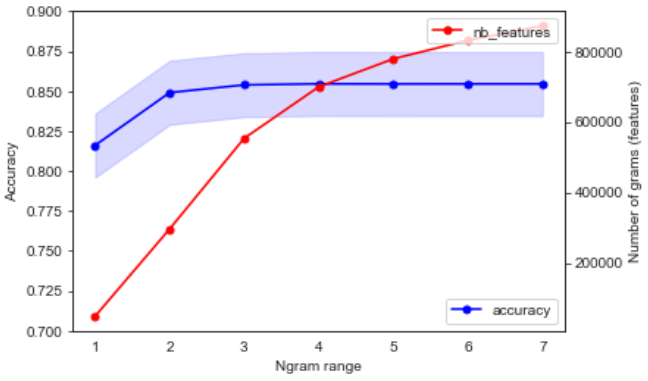

Figure 1: Model accuracy and complexity as a function of the N-grams range. Error bars represent Std.

From **Figure 1** we can see that the 3-grams are the last grams that significantly affect the accuracy ($\simeq 85\%$) of the model while presenting the advantage of being less parameterized (<600 000 features) than the following grams.

4 Word embedding & Neural Networks:

We will now focus on some more elaborate techniques for tweet vectorization as well as more parameterized machine learning algorithms.

4.1 Word embedding:

Word embedding is a way of representing words in a continuous vector space, where semantically similar words are represented by nearby points in the space. This can be useful for our task because it allows the model to capture the meaning of words and the relationships between them in a way that is more robust.

4.1.1 *CBOW* VS *Skip-gram* :

We implemented both **CBOW** and **Skip-gram** using `Word2Vec` in order to compare them. Both methods capture the meaning and context of words in the tweets, however, there are some differences between the two approaches that may make one more suitable for tweets than the other.

CBOW takes a window of surrounding words as the context for a target word, and tries to predict the target word based on the context. For example, if the target word is "Messi" and the context "is the goat", **CBOW** would try to predict the probability that "Messi" is the target word based on the context. The prediction is done using a softmax function, which assigns a probability to each word in the vocabulary based on the sum of the word embeddings of the context words. We update then the model using the gradient descent algorithm to minimize the error between the predicted probability and the true target word.

Skip-gram, on the other hand, takes a single target word and tries to predict the surrounding context words. So, using the same example, **Skip-gram** would take "Messi" as the target word and try to predict the probability of the surrounding words "is", "the" and "goat". We implement the prediction in a similar way as for **CBOW** but based on the word embedding of the target word instead of the word embeddings sum of context words.

Both these methods enable us to map each word in the tweet to a *unique* fixed length vector of numbers. We made the choice to set `vector_size` to 300, since it was proven to be the best dimension to capture the embeddings of the words when using **CBOW** and **Skip-gram**[1].
Then to find the embedding of a tweet we average the embeddings of each word composing it.
We wanted to study the influence of noise on both models. To do so we implemented various versions of **CBOW** and **Skip-gram** with different `min_count` which is the minimum number of tweets where a word has to appear so that we consider its embedding.

Then we fed the embedding of the tweet to a fully connected neural network with 2 hidden layers of size 200 and *ReLu* as an activation function. Here is a table summarizing the results of the different embedding techniques on AIcrowd when using the described architecture with a learning rate of 1×10^{-5} and 3 epochs and Cross Entropy Loss as criteria in **Table 1**.

	min_count			
	1	2	3	4
Skip-gram	$82,4\%$	$\underline{83,4\%}$	$81,9\%$	$81,9\%$
CBOW	$82,3\%$	$82,8\%$	$\underline{83,1\%}$	$\underline{83,1\%}$

Table 1: Accuracy as a function of the used embedding and the minimal number of tweets in which a word has to appear to count in the tweet embedding.

The table shows that both **CBOW** and **Skip-gram** models perform similarly on the classification task, with a slightly higher accuracy of 83.4% for the Skip-gram model when the `min_count` is 2, i.e. when we reduce the noise. It is also worth noting that in general, a higher `min_count` may lead to a smaller vocabulary and potentially less accurate models, as it may exclude rare words that may be important for understanding the context of a tweet. On the other hand, a lower `min_count` may include more noise in the model's vocabulary, potentially leading to less accurate models as well. Thus, **Skip-gram** performs better on our task that requires a more precise representation of the context despite being slower to train than **CBOW**.

4.1.2 BERT

Another interesting embedding technique we did consider is Bidirectional Encoder Representations from Transformers (**BERT**). It is a state-of-the-art NLP model that is a pre-trained deep neural network on a large dataset of unannotated text. The network processes the input text in a bidirectional manner, meaning that it takes into account both the context before and after each word. This is in contrast to traditional language models, which only consider the context before each word. By doing so, **BERT** is able to capture more contextual information about each word.

When generating embeddings with **BERT**, the input text is passed through the model, which processes it and produces a set of numerical values that represent the meaning of the input. For each tweet, the output is a set of vectors of constant length n composed of:

- A 'CLS' (Classification) token of length n which is a special symbol that is typically used in classification tasks.
- For a tweet of m words, a matrix of size $m \times n$ containing the vector representing the embedding of each word in the tweet.

It is important to note that with **BERT**, the same word gets a different embedding depending on the context surrounding it.

We tried two different pre-trained models, namely:

- **XtremeDistillBERT** is a pre-trained model that has been developed using a process called knowledge distillation and trained on data from the BERT model.
Knowledge distillation is a technique for training a smaller, more efficient model to replicate the behavior of a larger, more complex model, by transferring knowledge from the larger model to the smaller model through a process called "teacher-student learning."
These shallow models (called students) are then trained to mimic the output of BERT (called teachers) based on a transfer set.
In our application, it provides a way to compress the huge neural networks that we have with BERT into smaller ones. [2]

- **BERTweet** is another pre-trained language model that has been however specifically designed and trained for use on English tweets.
It has the same architecture as BERT-base, a widely used pre-trained model for natural language processing tasks, and is trained using the RoBERTa pre-training procedure, which is designed to improve the performance of the resulting model on downstream tasks. [3]

$\Longrightarrow$ It is true that **XtremeDistilBERT** is a more resource-efficient version of BERT, making it a good choice for resource-constrained environments.
However, **BERTweet** has been fine-tuned specifically for sentiment analysis on tweets. It may thus be more accurate at this task out-of-the-box than **XtremeDistilBERT**. We will later see if accuracy results confirm or not this hypothesis.

4.2 Transfer Learning:

Since both implementations of **BERT** only give us the word embedding and a classification token we had to perform transfer learning to adapt them to our polarity classification task. Thus we added some extra layers with various architectures to achieve our polarity classification task. The idea behind that is that the lower original layers of the neural network, which learn more general features, can be transferred to the second task, while the higher layers, which learn task-specific features, are re-trained on our tweet data-set.

Thus we considered two main directions to achieve our goal:

1. Retraining the entire network and modifying the weights of lower and higher layers.

2. Freezing the lower layers and learning the weight of the the higher ones using back propagation.

4.2.1 Retraining the entire network:

The features that we retrieve from the data after going through the network are the embeddings of each word an the 'CLS' token.

Our first approach was to average the word embeddings to get the tweet embedding. Thus we had two vectors of size 256 for **XtremeDistillBERT** and 1024 for **BERTweet** that we fed to a neural network composed of a linear layer with *ReLu* as an activation function, a dropout layer with rate of 0.25 and finally a second linear layer with again a *ReLu* as activation function, with each layer containing 500 nodes. At each step we calculated the Cross Entropy loss and used `AdamW` for the optimization.

A second approach was to take the row word embeddings and get them through a **bidirectional Long Short-Term Memory** (BiLSTM) model. A **BiLSTM** is a type of recurrent neural network (RNN) that processes input sequences in two directions: from past to future and from future to past. This allows the network to capture contextual information from both the beginning and the end of the input sequence. The LSTM will process the sequence of embeddings and output a sequence of hidden states, which can then be used as input, along with the 'CLS' token, to a final classification layers to predict the sentiment of the input text (2 hidden layers of 256 nodes and *ReLu* as activation function).

4.2.2 Freezing the lower layers:

By freezing the lower layers, we use BERT models for the embedding, without retraining them. Thus the models do no learn the optimal embedding for our task, i.e. the embedding remains suited for the original tasks the models where trained on. However this comes with some computational compensation since we only need to train the higher layers of our models since we can store the words embedding once and feed them to the added layers at each step. In this situation the 'CLS' token becomes meaning less since it is no longer suited for our task as mentioned in BERT documentation [4]. As for the models, we will use the same ones we described above.

4.2.3 Results:

Based on our results in **Table 2**, it appears that the performance of the XtremeDistilBERT (XDB) and BERTweet (BT) models varies depending on the features used as input, the neural network architecture, and the number of epochs of training.
Furthermore, accuracy tends to decrease when the model's weights are frozen. Overall, it appears that using tweet embeddings and the 'CLS' token as features, and training BERTweet with a fully connected architecture, is the most effective approach we found for sentiment analysis with an accuracy of 91.4% but it is also the most costly one.

Model & Arch.	Feature	Freeze	Epochs	Acc.
XDB & BiLSTM	we+'CLS'	No	4	88.1%
XDB & BiLSTM	we+'CLS'	Yes	4	76.2%
XDB & FC	te+'CLS'	No	2	87.8%
XDB & FC	te+'CLS'	Yes	4	78.5%
BT & FC	te+'CLS'	No	2	**91.4%**
BT & FC	te+'CLS'	Yes	2	81.3%

Table 2: Accuracy on AIcrowed as a function of the used Features, BERT models, neural network Architecture, the Freeze or not of lower layers and number of Epochs while keeping the learning rate at 10^{-5}.
(XDB) XtremeDistilBERT; (BT) BERTweet; (FC) fully connected neural network; (BiLSTM) bidirectional LSTM; (we) word; (te) tweet embedding (avg. of we);

5 Conclusion & further works:

To conclude, data preprocessing is an essential step in natural language processing when working with tweets, as these messages often contain unconventional language that can be difficult for NLP algorithms to process. We implemented several methods to achieve it including word normalization techniques such as case folding, contraction expansion, and handling of language specificities such as elongated words and slang. Additionally, lemmatization can be used to reduce words to their base form, which decreases the number of unique words in the dataset and make the algorithms more computationally efficient.

Surprisingly, embedding techniques like CBOW and Skip-gram combined with neural networks performed worse than TF-IDF fed to a Logistic regression. However, transformer methods, such as XtremeDistilBERT and BERTweet performed extremely well. The latter, with a fully connected network provided the best accuracy of 91.4%.
However it was disappointing to see that adding a BiLSTM after XtremeDistilBERT did not enhance the performances. Pooling the words embedding with a neural network does not offer a significant advantage on averaging them. This can be due to the presence of the 'CLS' token that already learn an appropriate pooling while retraining the whole model. We can suppose that the strength of the model comes from the original lower layers, and the added higher ones are not that important. To this end, it would be interesting to investigate more in depth fine tuning methods. In our work we treated the case where we froze or retrained all the lower layers. However recent researches have proven that freezing targeted parts of the networks may produce better results while improving computational efficiency [5].

References

[1] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[2] Subhabrata Mukherjee and Ahmed Hassan Awadallah. "XtremeDistil: Multi-stage Distillation for Massive Multilingual Models". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 2221–2234. DOI: 10.18653/v1/2020.acl-main.202. URL: https://aclanthology.org/2020.acl-main.202.

[3] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. "BERTweet: A pre-trained language model for English Tweets". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 9–14. DOI: 10.18653/v1/2020.emnlp-demos.2. URL: https://aclanthology.org/2020.emnlp-demos.2.

[4] *Bert*. URL: https://huggingface.co/transformers/v3.0.2/model_doc/bert.html#transformers.BertModel.

[5] Haojie Zhang et al. *Fine-Tuning Pre-Trained Language Models Effectively by Optimizing Subnetworks Adaptively*. 2022. DOI: 10.48550/ARXIV.2211.01642. URL: https://arxiv.org/abs/2211.01642.