Text Classification
CS-433 - Machine Learning

Mathurin Chritin ; Pedro Pino ; Mia Zosso
Communication Systems ; Electrical Engineering ; Physics

***Abstract*—Abstract: This report addresses the task of sentiment classification in tweets, specifically predicting whether a tweet originally contained a positive or a negative smiley. The provided dataset includes pre-processed Twitter data with smileys removed. The baseline solution employs word embeddings to construct feature representations for text classification and performs then linear models on them. More sophisticated methods were then used for better performances. A fully connected neural network and a convolutional neural network were constructed. The process involves generating word vectors using GloVe and/or stemmed and/or lemmatized embeddings. For the second model, the three embeddings were used as the input channels which improved the performance of the model. The latter stood out with an accuracy of 0.83, a difference of at least 0.06 to the next best model, which was the CNN with only one channel.**

I. Introduction

In the era of social media, understanding sentiment in user-generated content is crucial for various applications such as the evaluation of the general reaction to a new product on the market. The task at hand involves predicting the sentiment of tweets. Various linear methods such as logistic regression have been used to classify the tweets into negative and positive ones, before using neural networks. A fully connected network and a convolutional neural network have been built. The first was done to understand what parameters of the linear layers have a major impact on the accuracy. This was later used in the CNN, which was then further optimized. In the latter, two distinct approaches were employed: initially utilizing a single embedding and subsequently implementing three different embeddings as separate input channels for enhanced model performance.

The dataset, available from the AIcrowd platform, includes labeled training tweets indicating the original presence of positive or negative smileys. Additionally, an unlabeled test set of 10,000 tweets was provided for prediction.

II. Models and Methods

A. Preprocessing

A big part of this research was dedicated to preprocessing: the final goal is to represent the tweets dataset in a computer-understandable manner. This presented several big challenges, firstly because a rather big preprocessing pipeline was needed for our model to work, and secondly because of the limited amount of computation we were able to access to, which represented in the end close to 60% of the workload for the preprocessing only.

1) Word stemming and lemmatization: A first intuition that emerged was the utilization of natural language processing techniques such as stemming and lemmatization. These vocabulary-shrinking techniques seemed promising since they allowed us to reduce the size of the vocabulary and extract the essence of the meaning of the words, while keeping the word positions in the tweet. The python library `NLTK` [1] provides powerful tools to stem and lemmatize a corpus of text. We applied these methods to our original tweet dataset : we now have 3 representations of each tweet, the original one, the stemmed one and the lemmatized one.

2) Word embeddings: We now get to the part where we translate text tweets to machine-understandable representations. In this report, we will not cover how embeddings work nor how to generate them and will refer to GloVe [2] embeddings. We opted for the repository GloVe tools instead of the originally furnished ones to get a better grasp and fine tune the parameters. We essentially adapted the script of the latest available version to generate the embeddings on our whole vocabulary, containing all positive and negative labelled tweets as well as the unlabeled ones concatenated in a single file. We bring to attention that we generated three word embedding dictionnaries for each of the original, stemmed, and lemmatized set of tweets. The goal is to get three different representation of each tweet. The chosen embedding size was 20, partly for memory consumption issues. After this step, we now have a dictionnary allowing to translate tweets (list of text words) to tensors (one embedding vector per word, repeat for all tweets).

For our research on fully-connected neural networks, the above preprocessing steps suffice. The final step was to transform text tweets to a single embedding vector, which corresponds to the sum of the embeddings of the tweet's contained words. We thus obtain a tensor of shape (20,) for each tweet, which is the input structure for the fully connected networks. However, to be able to train a CNN, some more preprocessing steps were implemented and described below.

3) Input generation: The final step of the CNN preprocessing consisted on building our input matrices for our model. For that, we have to overcome the problem of the variable length of the tweets. Indeed, the idea behind the use

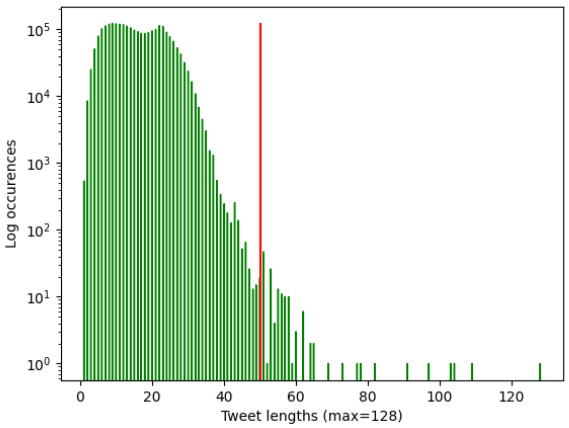

Figure 1. Tweet lengths on a log scale with the cutoff length in red.

of the CNN is for the network to be translational invariant, i.e. that no matter where in the sentence the word let us say "incredible" appears, it should most probably lead to a positive tweet. To do so, the tweets had to be loaded converting each word of the tweet to a vector (in this case of 20 features). Since the tweets were of different lengths, the shorter tweets were simply "padded" by adding at both ends of the tweet vectors of the mean features of the tweet. E.g. if the longest tweet is of size 6 and another one is of size 4, the shorter one was extended by adding to the left and to the right one vector (of size 20) consisting of the mean values of its 4 vectors (each one of size 20 as well of course). Looking at the histogram of the log-number of occurrences of the tweet lengths, we decided to ditch any tweet of more than 50 words, and set 50 as the standard size of all tweets : mean-padding was added before and after any tweet of length lower than 50 (see 1). Extending all tweets to a size of 128 (the longest one of the dataset) would not be sensible if only a small number of tweets were of such length. Here, the goal is to obtain a universal representation of the tweets, which would be of shape (3, 50, 20) : 3 channels (each one for a different embedding), 50 words (with padding if applicable), and a 20-dim vector for the each word which is the embedding itself. We thus obtain a uniform representation for each tweet, similarly to images that be all the same size. Finally, to help generalizing our trained model, we splitted the dataset in a training and a validation set, a common approach in machine learning. Another challenge that we have overcome is related to the in-memory size of the dataset : we decided to split the input data in 15 blocks so that we could load any block when needed. Saved on the disk as PyTorch tensors, the data is now ready to be fed to our CNN model.

B. Baseline model

1) Logistic regression: The logistic regression is very robust to unbalanced data and extreme values which can be an advantage in this task, since there might be a lot of words used very often by many users whereas some others with mispellings, or even invented ones would occur way less often. This method is also prone to overfitting, because it fits the noise as well. It may thus learn a decision boundary that perfectly separates the training data but does not generalize well to unseen data [3]. The LogisticRegression of Scikit-learn [4] with default parameters was used in order to get a very baseline model and have a starting point on what performances to expect.

C. Neural Network modeling

To handle our large dataset efficiently, we opted for iterative model training with batches, looking for a balance between computational constraints and learning dynamics. This approach allows us to navigate the trade-offs of noise, efficiency, and model generalization, ensuring practical feasibility with available GPU resources.

1) Fully Connected Neural Network: For all subsequent neural networks, the PyTorch [5] Python library was used extensively. The fully connected neural network was applied to a dataset constructed as follows: each word in every tweet was transformed into a 20-dimensional vector using its embeddings. Subsequently, the sum of these vectors was computed. This process resulted in each tweet being represented by a vector of 20 features, as described in greater detail in section II-A.

A standard architecture of a fully connected network has been implemented with 3 linear layers separated by dropouts and a ReLu activation function. The dropouts were added for the model to generalize better. Adding depth to the model provides the network with more capacity to learn complex relationships between words. The activation functions were added, because it enables the network to model non-linear relationships within the data, which is essential for capturing the complexity of natural language. A binary cross entropy loss was chosen, since the task at hand is a binary classification task. The loss was calculated using the BCEWithLogits loss of PyTorch which combines a Sigmoid layer and the BCELoss in one single class. The function takes care of the necessary activation and loss computation internally. It expects target between 0 and 1, so the labels had to be changed accordingly. It proved empirically more performing than the sigmoid activation function followed by the BCE loss. The optimizer was first chosen to be SGD, and tried with different learning rates to see the one that would adapt the best. As observed in the first project and in class, a bigger learning rate leans towards a faster converges (if not unstable) but worse precision, ideally we start with a sufficiently low big learning rate and smaller

at each iteration for a better precision. That's the reason why the Adam optimizer was our final choice.

2) Convolutional Neural Network:

General architecture: Our CNN approach uses our 3 channels in our input data. Indeed, three different embeddings were used to extract more features out of the tweets, as explained in section II-A3. These three channels differ from each other: each tweet is expressed in three different spaces and thus the classification should benefit from this augmented space. The architecture of the CNN was then changed accordingly, to be able to work with the three input channels. A pretty standard architecture was implemented for this task: two blocks of 1D convolution, followed by the activation function ReLu to introduce non-linearity, maximum pooling to extract the most important features, then two fully connected networks separated by dropouts as in the above section [6].

Loss function: We also use the Binary Cross-Entropy loss, which was performing well on the fully-connected network.

Hyper-parameters: We recall here the input dimensions (2048, 3, 50, 20), where 2048 is the batch size. We opt for 2048 because setting it lower was making the loss too unstable, and because we had blocks of $\approx 110'000$ tweets. For the convolution layers, a kernel size of 60 (3 multiplied by 20 the size of the embeddings) has been empirically chosen, corresponding to a 3-words kernel size. A kernel size of 1 does not make sense, and considering that most of the tweet have less than 30 words, setting it to more than 4 does not seem a good idea (too large). Inherently from the chosen kernel size, the stride is chosen to be 20 so that the convolution is made word by word. These parameters obviously depend on the chosen size of the embedding. Between the two fully connected layers we put dropouts for the model to be able to generalize better and not learn all the noise as in the fully connected neural network. For the optimization process, the exact same steps were done as in the above section.

III. Results

Model	Accuracy
Logistic regression	0.71
Fully connected neural network	0.75
Convolutional neural network 1 channel	0.77
Convolutional neural network 3 channels	0.83

Table I
THE ACCURACY OF THE DIFFERENT MODELS AS GRADED ON THE AICROWD PLATFORM.

In table I, it can be seen that the convolutional neural network with three channels stands out with its accuracy of 0.83.

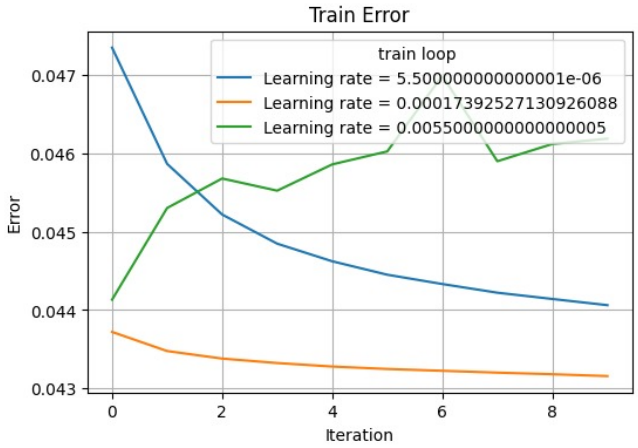

Figure 2. Three different learning rates were plotted to define which works best.

A. Fully connected neural network

The graph 2 depicts the loss evolution of a fully connected neural network using the Adam optimizer with varied initial learning rates. The first, a small learning rate, results in gradual convergence, requiring more epochs. The second, with a moderately larger rate, achieves faster convergence, indicating a balanced learning pace. The third, with a significantly larger rate, initially shows rapid improvement but experiences a subsequent increase in loss, signaling overshooting and model instability.

B. Convolutional neural network

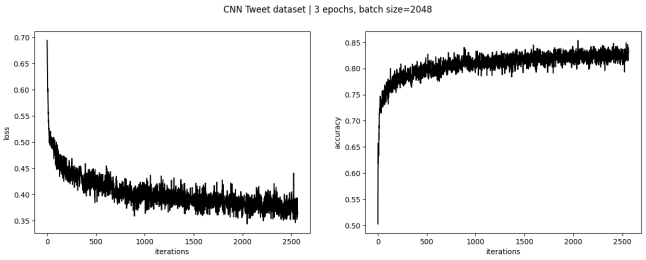

Figure 3. Train loss and accuracy over three epochs with 15 blocs.

In the left of figure 3, it can be seen that the loss of the model on the train set decreases with each iteration to converge to somewhere around 3.5. This is a good sign, since it indicates learning and optimization. Indeed, the model is updated for the loss to be smaller and smaller, meaning that the predicted data and the actual one agree more and more. The same conclusion can be reached from the right part of the same figure. The ascending accuracy indicates that the model is performing better and better on the data it was trained on.

IV. Discussion

As expected and shown in Table I, the more sophisticated the model, the better its performance. Convolutional neural

networks claimed the top two positions in our ranking, showcasing their effectiveness in handling the nuances of sentiment analysis. Notably, the CNN with three channels achieved an accuracy of 0.83, surpassing the next best model—a CNN with only one channel—by 0.06 points. These results also reflected on the validation set on which we obtained an F1-Score of 82.9%. This performance gap can be attributed to the multi-channel architecture, where each channel captures distinct textual features. Indeed, the three channels can be understood as representing different linguistic aspects, allowing the model to capture a more diverse range of features. For instance, these channels may focus on word patterns, subtle meanings, and expressions of sentiment. By incorporating information from these channels, the model gets better at understanding complex patterns, leading to improved overall performance.

A common approach in CNN that we could have explored is to implement skip connections [7], to allow the input to shortcut some layers, which helps optimizing the model faster. However, on a small network as the one we were working on, we judged it was not useful before some major architecture changes on the convolutional layers.

On the other hand, the fully connected neural network exhibited performance slightly worse than the second-best CNN. This result emphasizes the challenges fully connected networks have in grasping the local patterns, such as specific words or short phrases that contribute to the overall meaning. In fact, fully connected networks treat the input as a flattened vector, neglecting the spatial relationships present in the data. They also have a large number of parameters, which can lead to overfitting.

V. Ethical risks

In this project, we focused on developing a model that predicts correctly the given labels, but did not consider how the dataset was aggregated. Twitter is a very large platform with a wide variety of users, and we have no information concerning the origin of the tweets: are these tweets taken from a specific person's feed, which could be biased due to the tweet recommendation system of the platform? Also, while our focus was on the utilization of Glove embeddings for data preprocessing, it's important to note that these embeddings do not inherently consider the cultural origin of words.

The Glove embeddings rely on co-occurrence statistics within the training data, generalizing patterns learned from that data. However, if the training data exhibits biases toward a specific culture or community, these biases may be inherited and reflected in the embeddings. This raises ethical concerns, especially if the training data predominantly represents a certain linguistic style, potentially hindering the model's ability to recognize emotions expressed in different registers, such as slang.

For instance, if the training data predominantly consists of elaborate and sophisticated language, the model may struggle to recognize emotions conveyed through slang, which is more commonly used by specific demographics. This poses a risk of the model not effectively capturing the emotional nuances of individuals who predominantly use slang. Additionally, considering phenomena like irony, where language register is often modified to emphasize irony, a model trained on a limited class of the population might misinterpret language registers as indicators of irony.

Therefore, our ethical considerations are beyond the preprocessing stage, acknowledging that biases present in the training data can persist in subsequent model predictions. A detailed procedure should be written about the way the tweets have been aggregated in order to identify the potential biases it induces. If serious biases are identified, for example, if it is noted that a subset of the population is barely represented in the training data, an idea would be to weight the tweets depending on the population they belong to. The population subsets could be defined for example as the age of the user that tweets, and could be provided as metadata for each tweet, which would induce additional work when aggregating them. Finally, we should ensure the tweets are gathered in a uniform way and are not taken from a single source.

VI. Summary

In this report, two different neural networks were used to propose a solution to a classical text classification problem. A preprocessing pipeline was set up according to the needs of the models, which was presented extensively and can be run on the original dataset (see our associated Github repository). We discussed the performance of the model and could achieve a 83% accuracy on the platform, as well as the ethical implications arised by the dataset we worked on, questioning its origin. Although the performance on the testing dataset is not outstanding, a complete procedure has been developed to predict the emotion of a tweet from text input. To us, the next step should focus on exploring transformers neural networks, which is an on-going architecture that seems promising for this kind of text classification and prediction tasks.

References

[1] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc., 2009.

[2] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[3] N. Flammarion, "Classification," 2023.

[4] 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.

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[6] L. Voita, "Language modeling," 2023.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[8] W. McKinney, *pandas: powerful data analysis tools for Python*, pandas development team, 2022, https://pandas.pydata.org/.

[9] C. Harris, K. Millman, S. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. Smith, R. Kern, M. Picus, S. Hoyer, M. van Kerkwijk, M. Brett, A. Haldane, J. del Río, and M. Wiebe, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.