CS-433: Project 2
Tweet Sentiment Analysis

Daniel Tavares Agostinho, Thomas Castiglione, Jeremy Di Dio

***Abstract*—In this project, we developed multiple machine learning models for the purpose of conducting sentiment analysis on Twitter posts. The objective was to classify the posts as either positive or negative. To achieve this, we employed some state-of-the-art transformer models, including BERT and RoBERTa, for the classification task. The performance of our best model was evaluated on the testing set, yielding an accuracy of 0.892.**

I. Introduction

Twitter is a social media platform that allows users to communicate their thoughts and opinions through short messages known as tweets. These tweets have a maximum length of 280 characters and can be shared with other users on the platform. Twitter generates a vast amount of data, with approximately 340 million tweets being published per day in 2012 and an estimated 500 million tweets per day in 2022, according to sources [1], [2].

The objective of this project is to classify the sentiment of tweets into positive and negative categories using machine learning classification models. This process, known as sentiment analysis, can be utilized by brands, researchers, and organizations to gain insight into the general sentiment surrounding a particular topic or event. To achieve this goal, we trained several machine learning classification models on a specific labelled dataset.

II. Dataset and Preprocessing

A. Dataset

The dataset used for this project consists of about 2 million tweets that are labelled as positive or negative, depending on the smileys present. The dataset is well balanced as we have about 1 million samples for both classes.

B. Data processing

Tweets have multiple specificities that we first need to address before classification. Here are the pre-processing transformation we choose to apply to our dataset:

1) Removing tweet-specific words
In a tweet, users can reply to other users, they can share websites and they can embed another tweet, which is called a retweet. If present, these actions are visible and are represented by specific words: ”user”, ”url” and ”rt” respectively. However, since the distribution of these keywords was similar in both positive and negative sentiment categories, we determined it necessary to remove them from our analysis.
2) Removing stopwords
Stopwords are words that are commonly used but do not add much meaning to the sentence. A minimal set of these words can be considered as determiners, conjunctions and prepositions. However, we chose to keep stopwords with a negation interpretation as they could add context in negative sentences. For example, we have removed ”can” but kept ”cannot”.
3) Removing the punctuation
We decide to remove the punctuation to focus the sentiment analysis on the words themselves.
4) Removing camel case words
We decide to remove camel case words (group of words without spaces where each word starts with a capital letter) to make the data more consistent and only keep common language words.
5) Removing words with digits
Without any specific dictionary to translate word with digits inside to common words, we decide to remove them to keep the data consistent.
6) Putting everything to lower case :
Again, to keep data consistent we decide to put everything in lower case.
7) Removing repeating char
Since tweets are not known for their use of good English, we had to deal with spelling mistakes or other bad writing. We have chosen to replace all repeated characters with a single one. For example, ”Let’s goooo” will be replaced by ”Let’s go”.
8) Translation from slang
Using a slang dictionary [3], we translated slang words to common words.
9) Expand contractions
By the informal nature of most tweets and the limited number of characters available, a lot of contractions are used. We expand those contractions to the original sentence or word they represent.
10) Lemmatization
Lemmatization is a process where a word is replaced by its ”base word”. For example ”running” would be replaced by ”run”. In order to reduce words to their core meaning we lemmatize.

III. Methodology

Textual data is not suitable for training machine learning algorithms, so we must convert it into a numerical form. One way to do this is through word embedding, which represents words as vectors that contain the meaning of the words within a given corpus. There are two options for obtaining word embeddings: training them from scratch or using pre-trained embeddings. Training from scratch requires a large corpus and a significant amount of time to be effective. In this project, both options were considered.

A. TF-IDF

TF-IDF or "Term Frequency Inverse Document Frequency" [4] is a numerical statistic that is used to reflect how important a word is to a document in a corpus.

The term frequency (TF) of a word is the number of times it appears in a document. The inverse document frequency (IDF) is a measure of how common a word is across all documents. It is calculated by dividing the total number of documents by the number of documents that contain the word, and then taking the logarithm of that result.

The TF-IDF weight for each word i in document j is therefore computed as

$$w_{i,j} = TF_{i,j} \times \log \frac{N}{df_i}$$

Therefore, using this method of word embedding, we transformed each of our samples into numerical vectors and we applied different machine learning classification algorithms. The classifiers we used are the following: Bernoulli Naive Bayes, Linear Support Vector Machine, and Multinomial Naive Bayes.

B. Word2Vec

Word2Vec was created by a team of researchers at Google in 2013 [5]. Two models can be used to learn the embeddings. Continuous bag of words (CBOW) is a model where the previous and next words are used to predict the middle word. Using an averaged embedding of the input words, going through a hidden layer, it ouputs a V-dimension vector, where V is the size of the vocabulary, to predict the probability of each word to be the middle one. Depending on the results, it updates the embedding. The other is the Countinuous Skip-gram model. It does the opposite by using the current word to predict previous and following. Learning embeddings using Word2Vec can be done on very large corpus of word and relations between word have a linear behaviour (ie. $Vect(king) - Vect(man) + Vect(woman) \approx Vect(Queen)$). Therefore, using this method of word embedding, we transformed each of our samples into numerical vectors. We used a sliding window of 5 words for the Word2Vec algorithm. We applied different machine learning classification algorithms. The classifiers we used are the following: Linear Support Vector Machine and Bernouilli Naive Bayes with vector dimension of 100 and Sequential neuronal network with two hidden layer with dropout using sigmoid as activation function, cross entropy loss as loss function, Adam as optimizer and softmax as final activation function with vector dimension of 200.

C. GloVE

GloVE (Global Vector) is a word embedding method developed by Stanford University researchers [6]. To transform words into vectors, it first converts a corpus of text into a co-occurence matrix X where each word appearing in the corpus adds a line and a column. Meaning a corpus with $|N|$ different words will have a co-occurence matrix X of shape NxN. An entry X_{ij} counts how many time $word_i$ appears next to $word_j$. Once the matrix is created we can use it to find P_{ik}/P_{jk} where P_{ik} is the co-occurence probability of $word_i$ and $word_k$. By solving $F(w_i, w_j, \tilde{w_k}) = P_{ik}/P_{jk}$ and some development steps the cost function $J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - \log X_{ij})^2$ to minimize is found. $f(x)$ is a weighting function that makes rare co-occurence carry less information that frequent ones, w_i and b_i are word vector and bias for $word_i$, $\tilde{w_j}$ and $\tilde{b_j}$ are the context word and bias for $word_j$. Using GloVE is useful because it takes into account the context of each word to build its word embedding by using the co-occurrence matrix. It adds more global dependencies and performs better than most of the other word embeddings existing prior its creation. We used a pretrained GloVe embedding "twitter.27B" trained on 2B tweets, with a 1.2M vocabulary. The classifiers we used are the following: Bernouilli Naive Bayes, Linear Support Vector Machine, Logistic Regression with a vector dimension of 100 and Sequential neuronal network with two hidden layer with dropout using sigmoid as activation function, cross entropy loss as loss function, Adam as optimizer and softmax as final activation function with vector dimension of 200.

D. Transformers

The transformer model is a type of neural network architecture introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017[7]. It was specifically designed for natural language processing tasks, such as machine translation, language modeling, and text classification.

A transformer is composed of an encoder and a decoder, both of which are organized into series of layers that process the input data. These layers typically include a self-attention mechanism, which allows the model to take into account distant interactions between elements in the input data. This mechanism is implemented through the use of *queries*, *keys*, and *values*. The *queries* are used to assess the relevance of the input data for a specific task, the *keys* identify the relevant portions of the input data, and the *values* represent the data that is utilized to complete the task. The attention mechanism can be represented mathematically as follows:

Given an input sequence $X = [x_1, x_2, ..., x_n]$, where each x_i is a vector representing an element in the sequence, the attention mechanism computes the attention weights as follows:

$$\alpha_{i,j} = \frac{\exp(score(q_j, k_i))}{\sum_{k=1}^{n} \exp(score(q_j, k_k))}$$

Where, $\alpha_{i,j}$ is the attention score between the ith and jth elements in the input sequence, q_j is the query vector for the jth query, k_i is the key vector for the ith element in the input sequence, and $score(q_j, k_i)$ is a function that measures the relevance of the query to the key.

The transformer model then uses the attention weights to compute a weighted sum of the values in the input sequence as follows:

$$v_j = \sum_{i=1}^{n} \alpha_{i,j} x_i$$

Where v_j is the output of the attention mechanism for the jth query and x_i is the value vector for the ith element in the input sequence.

This allows the model to effectively capture long-range interactions in the data and perform well on tasks that require to understand the context and relationships between words in a sentence. The output of the attention mechanism is then processed by a feed-forward neural network, normalized, and passed to the next layer. Figure 1 below summarizes the transformer architecture.

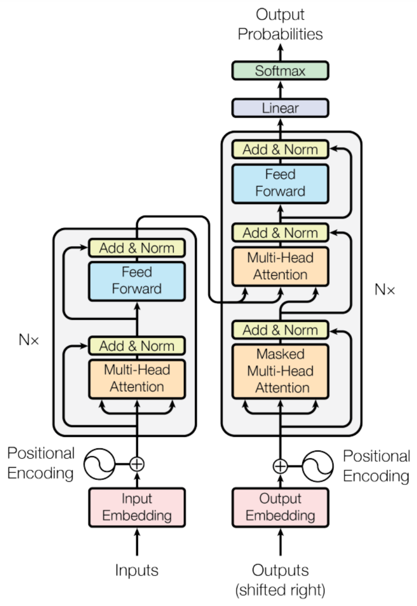


Figure 1. Transformer architecture from Vaswani et al. 2017 [7]. The input sequence is processed by the encoder (left) and passed to the decoder (right) along with the decoder output of the previous time step.

1) BERT: BERT (Bidirectional Encoder Representations from Transformers) is a natural language processing (NLP) model developed by Google in 2018 [8]. It was designed to pre-train deep neural network language models on a large dataset and then refine these models on smaller datasets for specific tasks such as language translation or text classification.

BERT builds on the Transformer model and introduces the concept of bidirectional learning, allowing the model to take into account the left and right context of a word, rather than just the left context as was previously the case. As a result, BERT became the first transformer-based model to achieve state-of-the-art results on a wide range of NLP tasks, including question answering, language translation and text classification. It has since been used as the basis for many other NLP models and has had a significant impact on the field of natural language processing.

For this project, we used a BERT model pretrained on English language using a masked language modeling (MLM) objective available on Hugging Face. It should be noted that in order to use this pre-trained model for the classification task, we added a multi-layer perceptron (MLP) layer to the end of the model.

2) XLNet: XLNet [9] was released by Google following BERT in order to address its limitations. BERT is based on an autoencoding pretraining by corrupting inputs with masks and learning to solve them. In XLNet paper they argue that this approach neglects the dependency between masked positions. Therefore they based XLNet on a autoregressive pretraining that uses all of the possible permutations of words in the sentence to predict the next one.

It was shown to beat BERT in multiples tasks, but with the drawback of being a lot more computationally intensive. A drawback that has consequences when computational resources for training are limited like in this project.

For classification we used a XLNet based model finetuned for sequence classification. Sequence classification is a task that involves predicting a label for a given input. The model is available on Hugging Face.

3) RoBERTa: RoBERTa [10] ("Robustly Optimized BERT Pretraining Approach") is a state-of-the-art language model developed by Facebook AI that is based on the BERT model and has been further improved through optimization and fine-tuning.

RoBERTa has achieved superior performance to other language models on a variety of natural language processing tasks, such as answering questions, translating languages, and classifying text. Additionally, it is able to learn and adapt to new tasks quickly, making it useful for various applications. For this project, we used a RoBERTa based model trained on around 58M tweets and finetuned for sentiment analysis with the TweetEval benchmark [11] available on Hugging Face. This model was already modified in order to take a tweet in input and output a probabilistic sentiment prediction as positive, neutral or negative. We further fine-tuned this model using our own dataset and modified the output interpretation to consider the maximum probability between positive and negative to be our discrete prediction.

E. Implementation

We used the Kaggle platform to fine-tune our transformer models, as they consist of large neural networks with many parameters. Fortunately for us, the Kaggle platform offers free runtime on two NVIDIA T4 GPUs, which allowed us to significantly reduce the fine-tuning time.

IV. Results

In this section, we present the experimental results of the numerous classification methods presented above. For our first machine learning models, we employed a k-fold cross-validation with 5 folds. Regarding the transformer models, cross-validation was computationally expensive, so we used 90% of the data as training set and 10% as validation set.

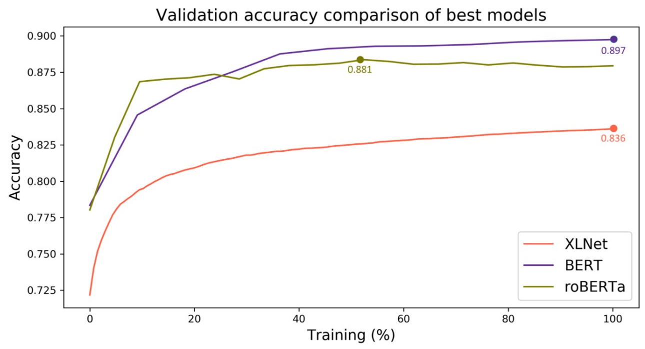
The implementation of the TF-IDF (Term Frequency-Inverse Document Frequency) embedding has resulted in a maximum accuracy of 0.8184 when using a linear support vector machine (SVM) model, as depicted in Table I. In regards to the selection of hyperparameters, we observed that the default values consistently yielded the best performance. The decision to employ linear classifiers was made due to the size of the TF-IDF matrix being too large for more computationally intensive models. Despite this, the results obtained through this method are still satisfactory and could potentially be improved by utilizing more complex models. However, we have elected to explore more promising alternatives, such as transformers, rather than pursuing this approach.

We observe that using a pretrained GloVE embedding on a large dataset of tweets resultes in better performance than using a Word2Vec embedding that was trained from scratch on our dataset. Using the same neuronal network, but with a change of embedding we observe in I that the accuracy increases by $\approx 7\%$. But we notice that using simple models has its limitations. Indeed, all of our models struggle to exceed 80% of accuracy. Only two models succeeded. Those models provide a good baseline but we need to turn ourselves to other ones if we want to reach better results.

Model	**BNB**	**LogReg**	**LSVM**	**MNB**	**NN**
TF-IDF	**0.75**	/	**0.82**	**0.80**	/
Word2Vec	0.63	/	0.74	/	0.74
GloVE	0.67	**0.75**	0.66	/	**0.80**

Table I
ACCURACY OF DIFFERENT MACHINE LEARNING MODELS

As anticipated, our transformer-based model demonstrated improved performance compared to previous methods, as shown in Figure IV, which illustrates the evolution of accuracy for each transformer-based model with respect to the percentage of fine-tuning applied.



The BERT base uncased model achieved the highest accuracy, with a score of 0.897 on the validation set and 0.892 on the testing set. While the XLNet model performed better than the baseline models, it was limited by the training resources available and only reached an accuracy of nearly 0.84 with 30% of the data. The RoBERTa-based model also showed improved performance compared to the baseline models, achieving a maximum accuracy of 0.881 on the validation set. However, this was lower than the performance achieved by the BERT base uncased model.

We also note that in the implementation of our transformer-based model, we found that utilizing all of the pre-processing methods described in the data processing section did not yield optimal results. As a result, we only applied steps 3, 5, 7, and 8 in order to maximize the performance of the model. It is worth noting that transformer-based models generally benefit from having more context available, and our decision to exclude certain pre-processing steps was made in service of this goal.

V. Discussion

Transformers are state-of-the-art models in NLP. As expected they performed better than simple machine algorithm. However, it can still be challenging to accurately classify the sentiment of a tweet using these models because some sentiment information may not be explicitly present in the tweet itself. As said in the Dataset section, the sentiment of a tweet is labelled depending on the presence of the smiley. But the use of a happy or a sad smiley doesn't always reflect the same sentiment of the sentence. For example ironic use of a smiley would be detected as the opposite sentiment as it should. Additionally, the context in which the tweet was posted is not taken into account, which can influence the sentiment of the tweet. Some sentences can carry different sentiments depending on the context they are posted. To improve the accuracy of sentiment classification in tweets, one potential approach is to incorporate user-labeled tweets. In addition, having the contextual information, such as the context in which the tweet was posted and the presence of previous and/or following tweets could be a way to improve the classification. It could be interesting to see if models could catch ironic tweets and to which extend having the previous and/or following tweets could help to rightly classify the sentiment.

VI. Conclusion

In this project, we present a comparative analysis of various text embedding and machine learning models that aim to perform a sentimental classification of tweets. We found that the state-of-the-art model for natural language processing, namely the transformer-based models, outperformed our baseline model, resulting in significant performance improvements. Furthermore, we found that the transformer-based models, in particular BERT and RoBERTa, significantly improve the accuracy of the classification task compared to the baseline model. Among these models, BERT achieved the highest accuracy. Overall, our results demonstrate the effectiveness of transformer-based models for natural language processing tasks and their potential to improve the accuracy of classification tasks on tweets.

References

[1] T. Inc. Twitter turns six. [Online]. Available: https://blog.twitter.com/official/en_us/a/2012/twitter-turns-six.html

[2] I. L. Stats. Twitter usage statistics. [Online]. Available: https://www.internetlivestats.com/twitter-statistics/

[3] M. MBAYE. Up-to-date list of slangs for text preprocessing. [Online]. Available: https://www.kaggle.com/code/nmaguette/up-to-date-list-of-slangs-for-text-preprocessing

[4] G. Salton and M. J. McGill, "Introduction to modern information retrieval," 1986.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[6] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *EMNLP*, vol. 14, 2014, pp. 1532–1543. [Online]. Available: https://nlp.stanford.edu/pubs/glove.pdf

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[9] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1906.08237

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[11] F. Barbieri, J. Camacho-Collados, L. Neves, and L. E. Anke, "Tweeteval: Unified benchmark and comparative evaluation for tweet classification," *CoRR*, vol. abs/2010.12421, 2020. [Online]. Available: https://arxiv.org/abs/2010.12421