Let's keep an eye on Twitter! - Deep Learning/NLP final project

Pedro Meseguer **Matteo Ferrari** **Jorge Enebral**

Abstract

This project involves building an AI model capable of receiving an embedding vector representing a specific tweet/phrase and returning a text containing the entities involved in the tweet and the entity's reputation risk. Our project development presents a unified model capable of performing Named Entity Recognition (NER) and Sentiment Analysis (SA). To achieve this, we have used an architecture with a combined loss function, weighted so that each task has an appropriate weight in the final result. The architecture used in turn displays an output with both results, allowing representations to be shared between them. The model also allows each task to be run separately and a specific model to be trained if only one or the other is desired. All this is done to be able to compare each task separately (NER and SA) with the model that combines both (NERSA). In addition, an automatic alert system is proposed, with a pre-trained text generation model that returns a personalized response based on the previous results.

1 Introduction

Tasks in Natural Language Processing (NLP), such as Sentiment Analysis (SA) and Named Entity Recognition (NER), are fundamental for text analysis. NER allows for the identification and classification of entities within a sentence (such as people, organizations, or locations), while SA evaluates the emotional tone or attitude of the text, classifying it as positive, negative, or neutral. These problems can be addressed separately—that is, analyzing sentiment on one hand and entities on the other, and then combining both results—but this approach can be somewhat inefficient. Sometimes, it is the combination of both that leads to a better-than-expected outcome, as they may share contextual information. This work explores a solution to this by proposing a unified model capable of handling both NER and SA in parallel, based on the intuition that recognizing entities can provide context for identifying sentiment, and vice versa. In particular, for this project, the data primarily consists of tweets, which is beneficial since both the entities and emotional tone are important in this context.

2 Prior Related Work

In recent years, both Named Entity Recognition (NER) and Sentiment Analysis (SA) have been extensively studied in the field of Natural Language Processing. Traditionally, they have been addressed independently, with approaches such as Conditional Random Fields (CRF) and Long Short-Term Memory (LSTM) networks for NER, and classification models based on embeddings and neural networks for SA. Early sentiment analysis techniques relied on simple methods such as word lexicons or bag-of-words representations. Later on, neural classifiers using Convolutional Neural Networks (CNNs) or LSTMs achieved significant improvements, especially on informal texts like tweets, which make up the dataset used in this work.

With the emergence of language models like BERT and similar architectures, the performance in both NER and SA has significantly improved. This improvement is due to the fact that these models better capture the context of all words in a sentence, processing text bidirectionally—not just from left to right, but also from right to left—allowing for a deeper understanding of the sentence. As a result, the field has progressed toward models capable of solving multiple tasks simultaneously through what is known as multi-task learning.

Along these lines, recent works such as [1] Hu, J. et al. (2025) have proposed unified architectures capable of simultaneously performing entity recognition and structured language understanding, thereby improving information extraction for more reliable outcomes.

Our work focuses on these more recent models, aiming for a joint architecture. However, it differentiates itself by focusing on a lighter and more easily trainable architecture, with an alert system based on the joint prediction of NER and SA.

3 Model

Generally, for tasks related to sentiment analysis and named entity recognition (NER), the most effective models (excluding transformers) are Recurrent Neural Networks (RNNs), as they store information from the past. Among RNNs, Long Short-Term Memory (LSTM) networks stand out, and this is the one we have used.

For our case, we initially proposed a basic model composed of a bidirectional LSTM and two linear layers that process the information in parallel, each specializing in its corresponding task: one for SA classification and the other for NER.

The use of a bidirectional LSTM over a unidirectional one is justified because, in addition to capturing better the context of the sentence/tweet (minimizing the amount of information the LSTM "forgets" across layers), for tasks like NER, it is capable of analyzing subsequent words. This is especially useful since many named entities consist of more than one word, and without the ability to analyze future context, the classification of the entity may not be accurate.

The linear layer for sentiment analysis is applied to each output of each neuron after it receives the previous hidden state and input, while the linear layer for entity recognition is applied to the last hidden state of the model, requiring the concatenation of the last forward and backward hidden states.

Regarding the loss functions, since both NER and SA are non-binary classification tasks, we used cross-entropy loss. Furthermore, since we observed that the training, validation, and test data are highly imbalanced for both NER and SA, we had to use the weight parameter, which allows us to use the CrossEntropyLoss function from Python's nn, adjusting the influence of each class on the loss function. The NER loss function also ignores indices of -1, as these represent padding.

We combine losses as:

$$\mathcal{L} = \lambda_{ner}\mathcal{L}ner + \lambda_{sa}\mathcal{L}sa, \tag{1}$$

with $\lambda ner = 0.35$, $\lambda_{sa} = 0.65$. We empirically tune these weights on validation data.

For optimizers, we used Adam, with two separate instances—one for each loss function—each with its own learning rate and weight decay.

Once the model was trained, evaluation could be done in two different ways. The first is the classic evaluation method, which calculates accuracy on the test set for both NER and SA. The second method involves using a manually written sentence as input for the model, which processes it, obtains its results, and generates a specific prompt. This prompt is then passed to a pre-trained model from Deep Seek ("deepseek-r1-distill-llama-70b"), which returns the final output. For the generative model, the prompt is based on the sentiment analysis and the entities mentioned. Below are the prompts for each detected sentiment (with an example sentence):

- Neutral SA: "Generate a message with the following entities (they have been mentioned in a tweet with neutral SA, DO NOT CONGRATULATE): argentina as a location entity, saudi as a location entity, arabia as a location entity. I want you to give the alert/congratulation/message between 3 symbols: — message — . DO NOT USE EMOJIS."
- Negative SA: "Generate an alert (the tweet speaks negatively of these entities, JUST THE ALERT, DO NOT REFERENCE THE USER NOR THE AI ASSISTANT) for: france as a location entity, croatia as a location entity. I want you to give the alert/congratulation/message between 3 symbols: — message — . DO NOT USE EMOJIS."
- Positive SA: "Generate a congratulatory message (the tweet speaks highly of these entities. I JUST WANT YOU TO MENTION THAT THEY HAVE BEEN REFERENCED POSITIVELY) for: brazil as a location entity, belgium as a location entity. I want you to give the alert/congratulation/message between 3 symbols: — message — . DO NOT USE EMOJIS."

The final part of the different prompts includes several limitations to steer the result in the desired direction. The definitive response must be enclosed between 3 symbols, which we identify through a regular expression.

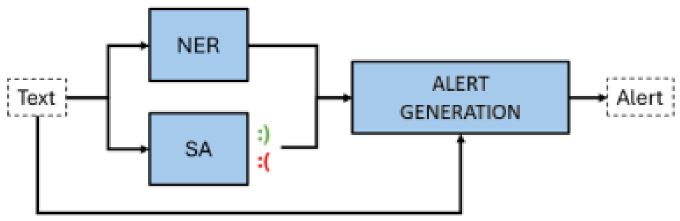


Figure 1: Joint NER and SA architecture with alert generator.

The goal of using this simple model was to check for possible programming errors and to obtain a general functional model to begin the testing process. To train this model, we designed a specific loss function for our project. This function combines two other cost functions and adjusts their weights accordingly.

4 Data Processing

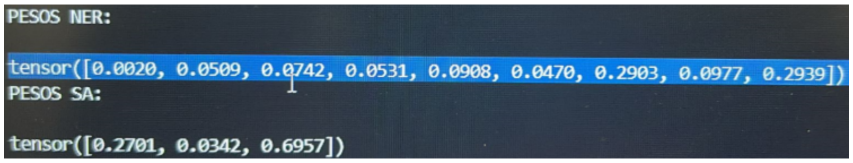
The data used for our model has been carefully manipulated to create the datasets and tailor them to ensure the model functions correctly.

To start, we implemented the function download data, which downloads the CoNLL-2003 corpus via load dataset("conll2003") and combines it with a pre-trained sentiment analyzer (pipeline("sentiment-analysis", model="cardiffnlp/twitter-roberta-base-sentiment")). Inside download data, we iterate through each partition ("train," "validation," "test"), clean and normalize the tokens (removing quotation marks, converting to lowercase), extract the NER tags, and generate the corresponding sentiment label using the CardiffNLP pipeline. The results are stored in three JSONL files (train.json, validation.json, test.json), containing the fields tokens, ner, and sa.

These files are then transformed into data loaders using the load data function, which applies several operations:

- word2idx: This function converts each token into its index within the GloVe vocabulary (skipping any tokens not found in the vocabulary) and aligns the NER tags accordingly. For padding management, 1 is added to each index, reserving 0 for sentence padding.
- collate fn: This function receives a batch of examples (tokens, ner tags, sa label), applies word2idx, filters out empty sequences, sorts by descending length, and applies dynamic padding (pad sequence) to both the sentences (with 0) and the NER tags (with -1). The function then creates the tensors inputs, labels ner, labels sa, and lengths, ready for the network.

For the embeddings, the load embeddings function downloads the ZIP file for GloVe Twitter-27B, decompresses it, and converts it to the Word2Vec format using glove2word2vec. It is then loaded via KeyedVectors.load word2vec format. Currently, we are using 50-dimensional vectors. For padding, a vector of zeros is added in the first row of the embedding matrix.



```
PESOS NER:

tensor([0.0020, 0.0509, 0.0742, 0.0531, 0.0908, 0.0470, 0.2903, 0.0977, 0.2939])
PESOS SA:

tensor([0.2701, 0.0342, 0.6957])
```

Figure 2: Weights of the tensors for each class.

5 Experiments

During the development of this, we needed to conduct a series of experiments to verify the functionality of various functions used in our model.

The first experiments we had to carry out were testing with the provided notebooks to see how the pre-trained SA and NER models worked, and to understand the data structure and how to use them in the model. For this, we experimented with the BERT Sentiment Analysis tutorial and examined the data.

We also used some other functions for the proper functioning of the model, taken from an assignment in the Natural Language Processing I course. This assignment used the functions word2idx and collate fn to identify whether a tweet was written by a human or a bot.

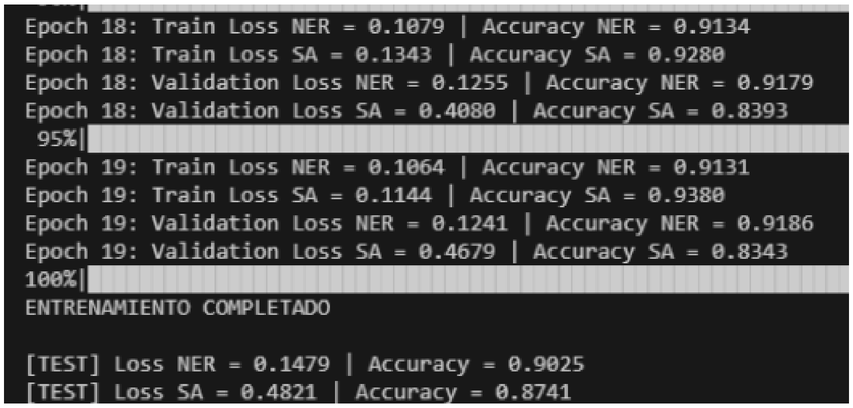
For the embeddings, we tested 50, 100, and 200 dimensions, with no clear improvement for high-dimensional embeddings.

At first, the SA model gave an accuracy of around 0.85, and the NER model 0.99, which seemed strange. The problem with NER was that the "out" category contained most of the words, so we had to resort to weighting. And when looking at the NER case, we realized that the same happened for SA, which always classified sentences that should have been labeled as 0 (negative) and 2 (positive) as 1 in the test set. For the weights, initially, we weighted the classes according to their frequency, giving more weight to the less frequent ones and less weight to the more frequent ones, obtaining what we seen in the Figure 2.

But after analyzing these weights in more detail, we realized that it was better to adjust them manually: for NER, we gave more weight to the first detection of entities (those starting with B), as otherwise, it was likely that there would be isolated entities in the middle with (I-AAA). We also know that it's important to categorize "OUT" as "OUT" because otherwise, it could trigger alerts for meaningless words. Therefore, we increased the weight of the first "OUT" category. For SA, we decided to be more conservative and slightly increase the weight of category 1 to avoid false alerts.

We tested hyperparameters, and after 20 epochs, the performance didn't improve. Since there was no overfitting, we left the weight decay and dropout at 0. The number of hidden layers is 1, as there was no improvement with 2. The hidden size is 32, which is smaller than the input dimension (50), as it needs to summarize the information.

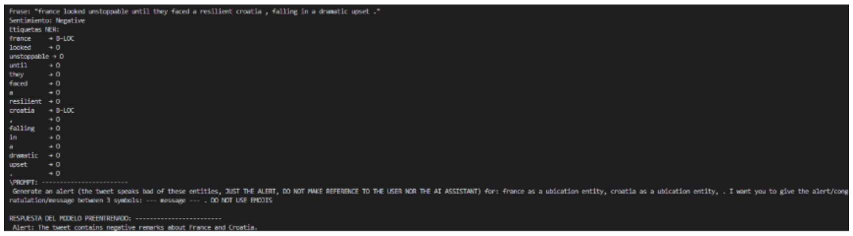
Regarding the loss weighting between NER and SA, it didn't vary much. We tried swapping values between 0.3-0.7, adding up to 1. We chose the values that experimentally gave the best results: 0.3 for NER and 0.7 for SA, also because, conceptually, it is more important for the alert generation to have the sentiment detection (SA) correct than to slightly improve the NER labels.



```
Epoch 18: Train Loss NER = 0.1079 | Accuracy NER = 0.9134
Epoch 18: Train Loss SA = 0.1343 | Accuracy SA = 0.9280
Epoch 18: Validation Loss NER = 0.1255 | Accuracy NER = 0.9179
Epoch 18: Validation Loss SA = 0.4080 | Accuracy SA = 0.8393
 95%|
Epoch 19: Train Loss NER = 0.1064 | Accuracy NER = 0.9131
Epoch 19: Train Loss SA = 0.1144 | Accuracy SA = 0.9380
Epoch 19: Validation Loss NER = 0.1241 | Accuracy NER = 0.9186
Epoch 19: Validation Loss SA = 0.4679 | Accuracy SA = 0.8343
100%|
ENTRENAMIENTO COMPLETADO

[TEST] Loss NER = 0.1479 | Accuracy = 0.9025
[TEST] Loss SA = 0.4821 | Accuracy = 0.8741
```

Figure 3: Results from the trained model.



```
Frase: "france looked unstoppable until they faced a resilient croatia , falling in a dramatic upset ."
Sentimiento: Negative
Etiquetas NER:
france      → B-LOC
looked      → O
unstoppable → O
until       → O
they        → O
faced       → O
a           → O
resilient   → O
croatia     → B-LOC
,           → O
falling     → O
in          → O
a           → O
dramatic    → O
upset       → O
.           → O
\PROMPT: -----------------------
 Generate an alert (the tweet speaks bad of these entities, JUST THE ALERT, DO NOT MAKE REFERENCE TO THE USER NOR THE AI ASSISTANT) for: france as a ubication entity, croatia as a ubication entity, . I want you to give the alert/cong
ratulation/message between 3 symbols: --- message --- . DO NOT USE EMOJIS

RESPUESTA DEL MODELO PREENTRENADO: -----------------------
 Alert: The tweet contains negative remarks about France and Croatia.
```

Figure 4: Results from the evaluate with a sentence with negative sentiment.

6 Results

After training the model, the results are seen in Figure 3. From the results, we can see that with 20 epochs, we achieve good accuracy for both NER and SA. However, as we mentioned earlier, a good accuracy does not necessarily imply that the model will function correctly. This is clearly observed with NER, where the dataset is completely imbalanced. If all cases are attributed to the same class without considering the others, the accuracy would still be high.

The final test still remains: manually input a sentence into the model to test it and see what alert it returns. In the evaluate.py module, there is an option to enter a sentence and obtain the different results of the process, including the original sentence (which should be in English), the sentiment, the NER labels, the prompt used for the generative AI, and finally, the model's response. An example with negative sentiment is attached in a photo seen in the Figure 4.

While generally, when only a single entity appears, the model responds correctly, when the tweet/text contains multiple entities, the model attributes a generalized sentiment despite the entities not sharing the same relationship (for example, in a football match where one team's performance is praised while the other is criticized).

7 Analysis and Conclusion

The results obtained reflect that the multitask approach to address Named Entity Recognition (NER) and Sentiment Analysis (SA) is effective. The model performs correctly in many cases, although, like any model, it can be significantly improved, as it still makes some errors. In terms of numbers, the accuracy achieved is quite good, and after testing it with the evaluation module, the model works reasonably well. However, as mentioned, these accuracy metrics do not necessarily indicate much on their own.

One of the most notable contributions is the model's ability to combine information extracted from both tasks to generate automated alerts and contextualized responses. In situations where there is only a single entity, the model responds adequately. However, a limitation is observed when dealing with more complex texts where multiple entities with different sentiments coexist. For this reason, future improvements should focus on handling cases involving multiple entities with differing sentiments.

The use of a simple architecture based on a bidirectional LSTM, along with pre-trained embeddings (GloVe), has kept the computational cost low and facilitated model interpretation. Additionally, if the model were to be improved, increasing the dimensions of the embeddings would be a key area for enhancement. In our project, we used 50-dimensional embeddings, but for more complex tasks, many more dimensions would be required.

Finally, integrating generative AI into the model to generate alerts is a key added value for the project, as it ultimately provides the most visible outcome for the client who would use it.

References

[1] Hu, J., Li, Z., Shen, M., Ai, H., Li, S., Zhang, J. (2025). Joint Automatic Speech Recognition and Structure Learning for Better Speech Understanding. https://arxiv.org/html/2501.07329v2S2