

# Natural Language Processing: Sentiment Analysis

Alexis BALESTRA, Gabriel LAFONT-MILLET

June 7, 2025

## Abstract

This report presents the methodology and results for a sentiment analysis challenge on classifying tweets into positive, negative, or neutral sentiments. Using a dataset from Figure Eight's platform, we benchmarked several models against the Macro F1-Score. Our investigation included parameter optimization for architectures such as the Qwen large language model and a Naive Bayes classifier, which yielded scores of 0.6937 and 0.7322, respectively. The most effective model was a Recurrent Neural Network (RNN), which achieved a final Macro F1-Score of 0.81, highlighting its strong capabilities for this NLP task.

## 1 Introduction

Text is the main way information is stored on the internet, which means that it is also the data type most usable for data analysis. In this paper, we will mainly focus on the sentiment analysis side of the natural language processing field. Our goal is to classify strings of text into the categories "positive", "neutral", or "negative" depending on the emotion of the message. The primary metric for success is the Macro F1-Score, which ensures a balanced evaluation across all sentiment classes.

## 2 Large Language Models (LLM)

We first attempted to use Large Language Models and more precisely Qwen2.5-Instruct with several different sizes. Given the processing time of LLM, we chose to do some data preparation by testing our LLMs only on a subsection of 124 tweets from the original dataset. We made sure that the subsection would be balanced. Given that there is no "training to do" for our LLM, there was no need to do any splitting.

We used 3 versions of the following prompt:

```
Classify the following message as either positive, neutral or negative,  
to do that answer only "positive", "neutral" or "negative", don't say anything else.  
Message: {}  
Answer:
```

We first tested a 0-shot version, then a one-shot version and finally a 3-shot version with one example for each of the possible emotions.

In order to cut down on inference costs, we decided to limit the amount of token prediction to one. This is why it is important that the model only answer with negative, neutral, or positive. This also makes it easier to classify the LLMs' answer as an integer, so that the classification is not stored as a string directly.

We also put the temperature of the LLM as 0, in order for the LLM's output to be deterministic. A higher temperature allows for more creativity at the cost of more random output. For the current task, creativity is useless, and a more random output is detrimental to the task's performance.

We got the following results with the zero-shot, one-shot, and three-shot prompts in Table 1.

The maximum Macro F1-Score is 0.6937 and is achieved by Qwen2.5-3B-Instruct, which has 3 billion parameters. The 7 billion parameter has approximately the same score, which shows that the issue isn't just computing power. It seems that the number of shots is most important for small models, since, as we can see, the 0.5B parameter has significantly improved with the number of shots.

Model	F1 Score Macro (Zero-shot)	One-shot	Three-shot
Qwen2.5-0.5B-Instruct	0.3297	0.4239	0.5518
Qwen2.5-3B-Instruct	0.6752	0.6937	0.6835
Qwen2.5-7B-Instruct	0.6588	0.6829	0.6852

Table 1: F1 score macro comparison of various Qwen models on the sentiment analysis task in zero-shot, one-shot, and three-shot settings.

In comparison, the larger model doesn't seem much influenced by the number of examples. A Macro F1-Score of 0.6937 isn't that good compared to the high inference cost for such large models. This seems to show that while LLMs are very good for general tasks, they aren't as good when it comes to more narrow tasks, especially when their inference cost is taken into account. However, there exist some smaller models with transformer-based architectures that can be fine-tuned to obtain very good performances on more narrow tasks, such as DistilBERT.

There is also another explanation for their poor performance. There is a bit of subjectivity when it comes to sentiment analysis. For instance, the message with id 088132673e "Just an observation: Aside from the riverwalk, there are pretty much no cute girls in downtown sa" is considered neutral in the dataset, but Qwen2.5-7B-Instruct in one-shot classifies it as negative. I would tend to agree with the classification of the LLM rather than with the dataset. The other models, which are trained on the dataset, perform better, but it might be due, in part, to learning the exact way the classification was done in the dataset, rather than truly being better at sentiment analysis.

### 3 Naive Bayes

A Naive Bayes classifier is a probabilistic machine learning algorithm based on Bayes' Theorem used for classification tasks. It operates on the "naive" assumption that the features it uses for prediction are all independent of one another, which makes it remarkably fast and effective, especially for text categorization.

We tested to change 3 parameters, with the results in Figure 1:

- **Max Features**: This parameter specifies the maximum number of features (vocabulary size) to build from the text data.

- **sublinear\_tf**: When set to True, this parameter applies a sublinear scaling to the term frequency (TF) component of the TF-IDF calculation. The TF-IDF calculation is used to determine how important a word is to a document in a dataset.

- **ngram**: This parameter defines the lower and upper boundary of the n values for different word n-grams (sequences of words) to be extracted. We always keep individual words as the minimum, but test with different maximums.

#### 3.1 Data Processing

Given that the dataset was fairly balanced, there was no need to do further balancing. We chose the split 0.7, 0.15, 0.15 for training, validation, and testing ratios. We also converted the text input to numeric features using TF-IDF, to make it in the correct format. Finally, we removed all of the English "stop words": common words that carry little meaning in text analysis, and are therefore useless for our algorithm. For example, we remove : "the", "is", "and", "in", "of", "to", "a", "that", "it", "on", etc...

#### 3.2 Max Features

We tested our Naive Bayes models with a number of max features between 1 thousand and 220 thousand, which is displayed in Figure 1 on a logarithmic scale. For most models, increasing the number of max features gives better results up to a number of max features of 40 thousand. After that, the performances seem to dive and then fluctuate a lot. This suggests that a number of max features of 40 thousand is an optimal range for feature selection before the inclusion of more features introduces noise or redundancy.

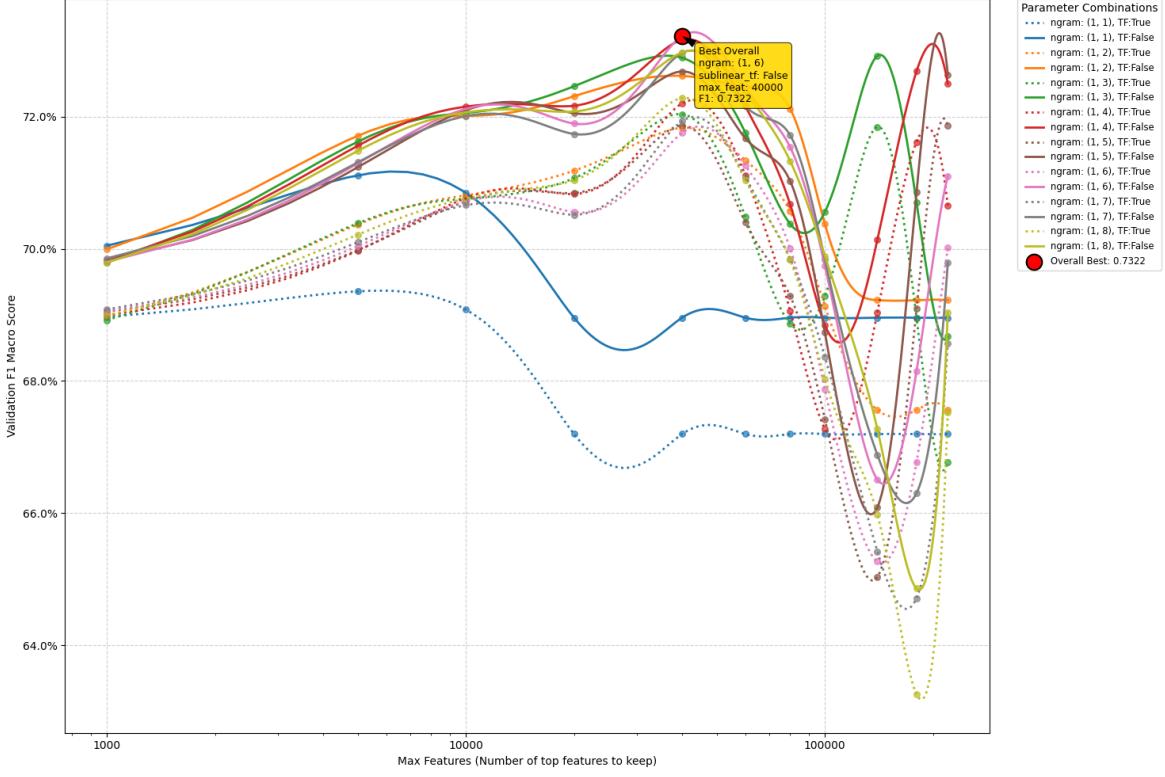


Figure 1: Effect of Max Features on Validation Macro F1 Score for Naive Bayes Classifiers

### 3.3 sublinear\_tf

Overall, the models with sublinear\_tf as true seem to perform worse than those with it kept as false. The logarithmic scaling of the parameter sublinear\_tf compresses the importance of very frequent terms, but in the context of tweets like in our dataset, very frequent terms can still be indicative of sentiment. This might explain the worse performances.

### 3.4 ngram

Models with a maximum ngram of 1 seem to perform significantly worse than the others. This shows that single words are too few to capture the full semantics of the message when it comes to sentiment. Having a large maximum n for ngrams allows the model to understand nuances such as "not good" or idiomatic expressions. However, for a maximum n for ngrams of two or more, the differences are way smaller and could be due to noise. When other parameters are taken into account, our best model has a maximum n for the ngram of 6, but the difference with other models is very small, highly dependent on other parameters, and could be due to noise.

### 3.5 Result

Our best model has a n for the ngrams between 1 and 6, it has the parameter sublinear\_tf as false, and has a maximum number of features of 40 thousand. It achieves a macro F1 score of 0.7322 on validation data and 0.7148 on test data, which is good, especially for such a simple model.

As seen in Table 2, the precision and recall are good for all the sentiment classes, although there is still a large margin for improvement.

Class	Precision	Recall
negative	0.75	0.64
neutral	0.67	0.72
positive	0.75	0.78

Table 2: Precision and Recall for each sentiment class of the best Naive Bayes classifier.

## 4 Recurrent Neural Network (RNN)

We then decided to test the efficacy of RNN to process text for sentiment analysis. A Recurrent Neural Network (RNN) is a type of neural network designed to process sequential data by maintaining an internal memory that allows it to use information from previous steps in the sequence to influence the processing of the current step.

One aspect we struggled with, for that particular kind of architecture, is having the model fall into a local minimum where it only predicts one output. Even if the data is quite balanced, we decided to implement weights for the cross-entropy loss function, which are inversely proportional to the frequency of the class. This did not fix the issue. After making several attempts at fixing this problem unsuccessfully by tuning and changing parameters, we decided to use Bidirectional Long Short-Term Memory, an advanced type of RNN, particularly effective at capturing long-range dependencies in sequences by processing data in both forward and backward directions. This fixed the problem of predicting only one class. The more complex bidirectional architecture seems to be able to learn more complex representations of the data, which prevents it, most of the time, from converging to a trivial, single-class output.

We made some tests to figure out the best parameters for this architecture. We decided to test the effect of the following parameters, with the results on Figure 2:

- **Number of LSTM units:** This parameter defines the number of hidden units in the LSTM layer. We tested with values of 64, 96, 128, 256, and 512.
- **Number of layers:** This specifies the number of recurrent layers to stack in the LSTM model, with configurations testing 1, 2, or 3 layers.
- **Learning rate (LR):** This controls the step size of the optimizer during training and is tested at values of 0.001 and 0.003.

### 4.1 Data Processing

Given that the dataset was fairly balanced, there was no need to do further balancing. We once again chose the split 0.7, 0.15, 0.15 for training, validation, and testing ratios. We converted the text data into tokens to make sure it is of the correct format for the LLM. We also padded the data for easier parallelization.

### 4.2 Number of LSTM units

The parameters we tested seem to be highly dependent on each other, which makes it harder to quantify the effect of a single parameter in isolation. The number of LSTM units seems to make the difference between the models larger. Some models perform worse, others perform better as the number of LSTM units grows. The model, which has a learning rate of 0.001 and a number of layers of 3, falls into the problem of predicting only one class. This shows that even with Bidirectional Long Short-Term Memory this problem can still happen.

### 4.3 Number of layers

While it is hard to determine precisely because of the high interdependence between the parameters, it seems like a higher number of layers might have an adverse effect on performance. Our best model has a number of layers of 1, while our worst model has a number of layers of 3.

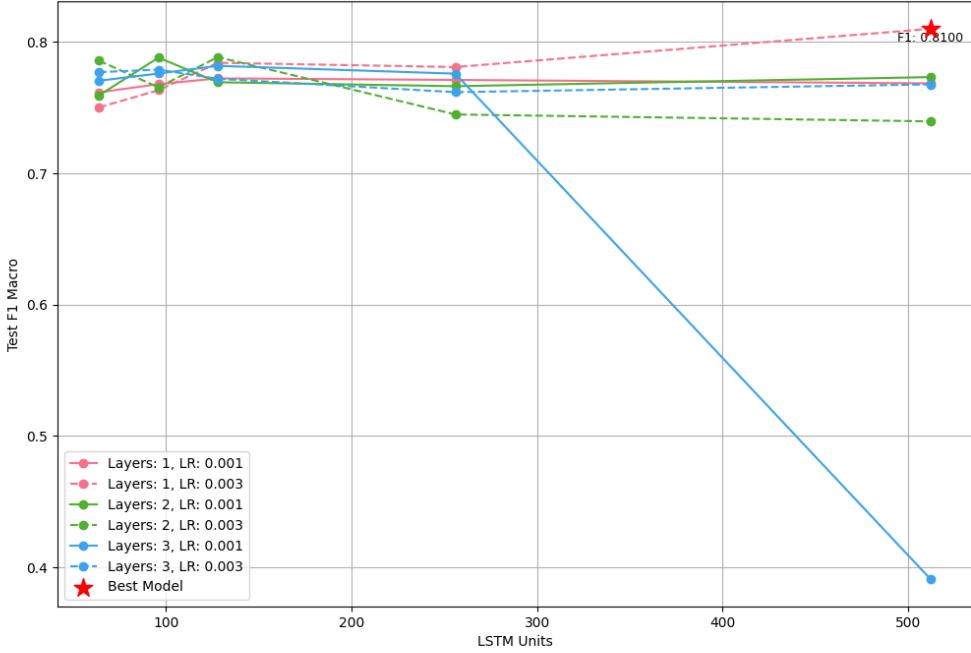


Figure 2: Effect of the number of LSTM Units on Validation Macro F1 Score for Recurrent Neural Networks

#### 4.4 Learning rate (LR)

It is hard to say what is best overall between a learning rate of 0.001 and 0.003. Our best model has a learning rate of 0.003, but it is hard to tell with regard to the other data points if this is just a coincidence or denotes some more general principle.

#### 4.5 Result

Our best model has an F1 macro score on test data of 0.8100 and achieves this result with a layer of 1, a learning rate of 0.003, and a number of LSTM units of 512. By training another model with the same parameters, we get an F1 macro score on test data of 0.7927. This suggests that while a F1 macro score of 0.8100 is very good, it is not consistent. It might be due to randomness during training, such as the random weight initialisation or the dropout layers. Overall, it is highly likely that the results in Figure 2 have a high variance. In order to get a better idea of the effect of those parameters, we would need to retrain the models several times in order to be able to calculate the standard deviation.

### 5 Conclusion

This report systematically evaluated three distinct approaches for the task of tweet sentiment analysis: a pre-trained Large Language Model (Qwen), a traditional Naive Bayes classifier, and a Recurrent Neural Network. Our objective was to identify the most effective architecture for classifying tweets as positive, negative, or neutral, benchmarked by the Macro F1-Score.

The Qwen LLM, despite its advanced general-purpose capabilities, achieved a moderate top score of 0.6937. Its high inference cost and lower performance on this specific task suggest it is not the optimal choice, even if its poor performance might be partly due to the dataset used. In contrast, the much simpler and computationally efficient Naive Bayes classifier proved to be a powerful baseline, achieving a respectable Macro F1-Score of 0.7322 on validation data. The most important metric for

that success was having a number of max features around 40 thousand, as well as having a maximum n for ngrams superior to or equal to two.

The most successful model was the Recurrent Neural Network, specifically a Bidirectional LSTM, which achieved at best a Macro F1-Score on test data of 0.8100. This architecture effectively captured the sequential nature of text and, after resolving initial training challenges, consistently outperformed the other models.

However, we noted that the RNN's performance exhibited some variance between training runs, likely due to random initialization and dropout. Future work could focus on enhancing the robustness of this result by training multiple models with different random seeds to establish a more stable performance baseline. Furthermore, exploring other advanced architectures, such as fine-tuning transformer-based models like DistilBERT, could potentially bridge the gap between specialized performance and the contextual understanding of LLMs.