Tweet Tone Classifier

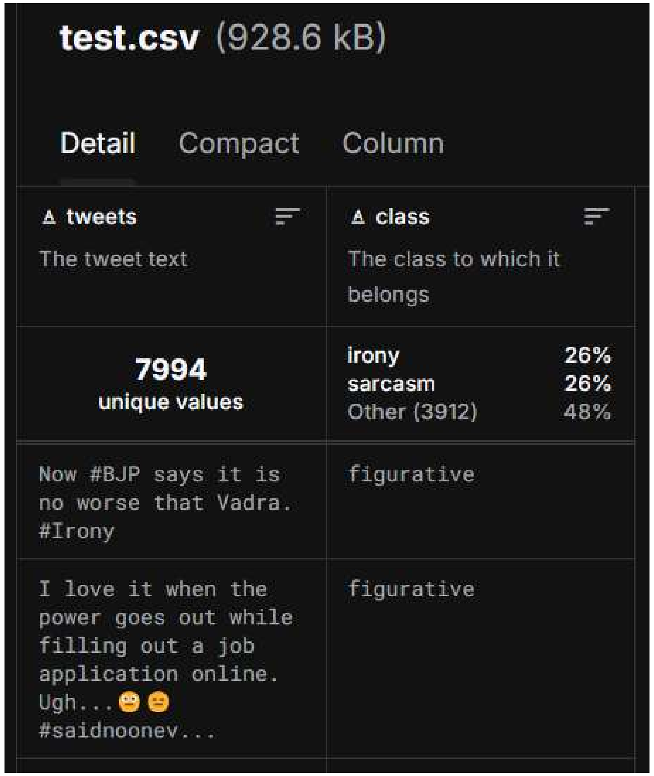
Project made by Brusamarello Michelangelo and Meloni Diego

THE DATASET

- The dataset consists of 2 files: train.csv and test.csv, both of these files contain 2 columns:
  - tweet: The text of the tweet, containing mentions, hashtags and links.
  - class: The label we are trying to predict.

THE DATASET

- The dataset consists of 2 files: train.csv and test.csv, both of these files contain 2 columns:
  - tweet: The text of the tweet, containing mentions, hashtags and links.
  - class: The label we are trying to predict.



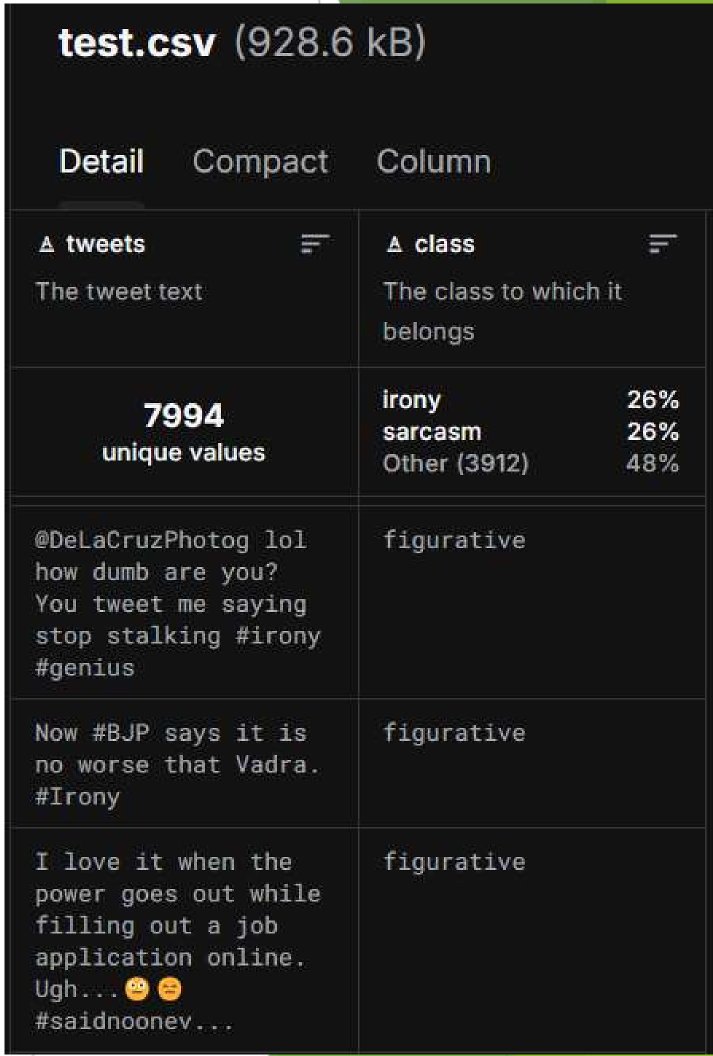
Our goal

- The aim of our project is to build a classifier that detects the tone of tweets, classifying them into one of four categories:
  - Ironic
  - Sarcastic
  - Figurative (ironic and sarcastic)
  - Regular

TWEETS AND INTERNET LANGUAGE

- As we can see tweets are not an easy language to work with, they can contain
  - Internet slang
  - User names (in mentions after @)
  - URL links
  - Hashtags (groups of sentences together without spaces, following the # symbol)
  - Emojis
  - Foreign characters

What preprocessing?



TWEETS AND INTERNET LANGUAGE

- As we can see tweets are not an easy language to work with, they can contain
  - Internet slang
  - User names (in mentions after @)
  - URL links
  - Hashtags (groups of sentences together without spaces, following the # symbol)
  - Emojis
  - Foreign characters

What preprocessing?

- We opted to remove all links, mentions, foreign characters and emojis since they only introduce noise in the dataset
- We want to keep the hashtags content because it might be useful for our classification task, but how do we tokenize them?

test.csv (928.6 kB)

Detail Compact Column

tweets The tweet text	class The class to which it belongs
7994 unique values	irony 26% sarcasm 26% Other (3912) 48%
@DeLaCruzPhotog lol how dumb are you? You tweet me saying stop stalking #irony #genius	figurative
Now #BJP says it is no worse that Vadra. #Irony	figurative
I love it when the power goes out while filling out a job application online. Ugh... #saidnoonev...	figurative

VOCABULARY

Word tokenization

- In general, with our dataset it would be problematic to use word tokenization (tweets contain plenty of internet slang, moreover do we count hashtags as unique words?), we would have too many words, and dealing with Out Of Vocabulary words (OOV) would be very hard

VOCABULARY

Word tokenization

- In general, with our dataset it would be problematic to use word tokenization (tweets contain plenty of internet slang, moreover do we count hashtags as unique words?), we would have too many words, and dealing with Out Of Vocabulary words (OOV) would be very hard

Subword tokenization

- Using subwords tokenization basically solves all our vocabulary problems, in fact now we can control the vocabulary size and since we search for subwords in the text, even if words are merged together it is no longer a problem for us.
- We used Byte Pair Encoding (BPE) to achieve this result

TWEET REPRESENTATION

Which representation do we choose?,
Which embedding method is better?

Count based

- TF IDF
- SVD (Truncated SVD)

Semantic based

- Word2Vec

COUNT BASED EMBEDDINGS

- We have tested how different types of embeddings perform on our dataset, in particular we have used TF-IDF and SVD.
- These methods are based on raw-count and co-occurrences, they provide an embedding which is relative to the tweet.
  - **TF-IDF** applies no dimensionality reduction, so it is more computationally expensive and we run it for less epochs
  - We apply **truncated SVD** on the previous embeddings to have some dimensionality reduction, in particular we have chosen to make embeddings of 50, 100 or 200 dimensions to see how much impact does this parameter have on the model's performance.

SEMANTIC EMBEDDINGS

- The main idea is that we would like similar words to have similar embeddings in our latent space, so they should capture (embed) some semantic meaning of the words.
- We have used Word2Vec to make this kind of embeddings

CLASSIFICATION

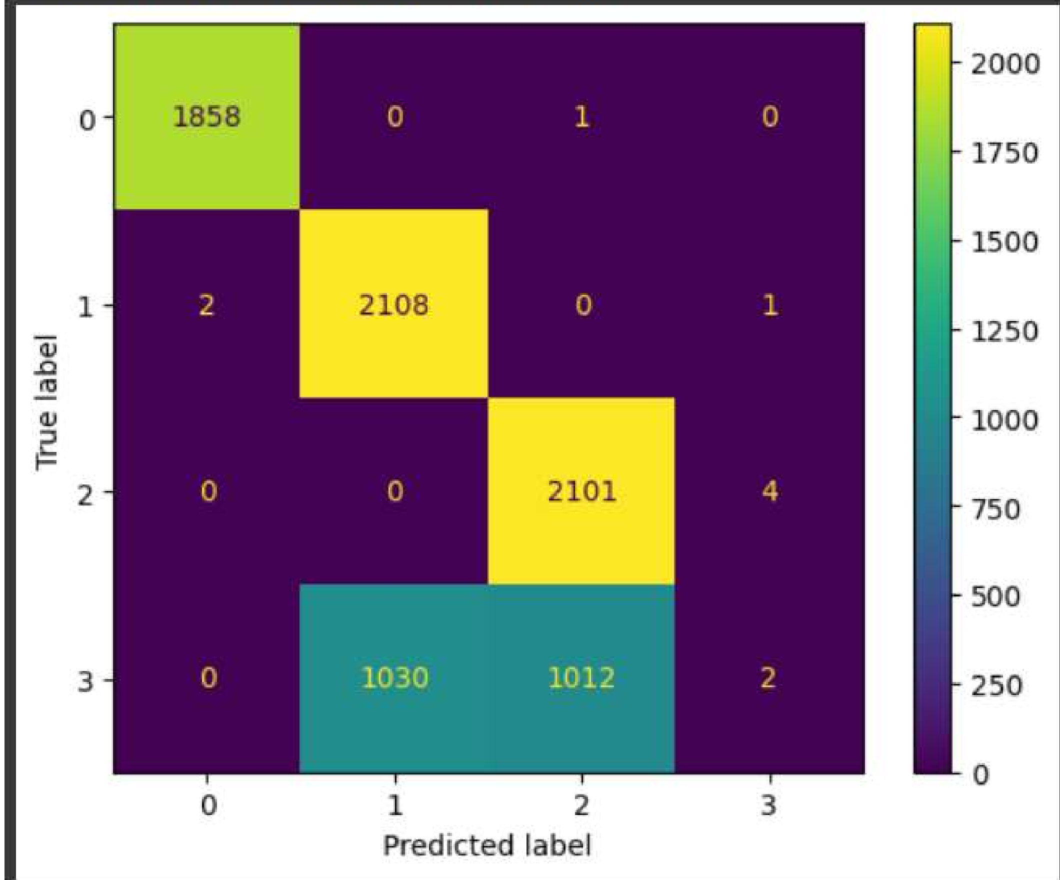
- For each of the embeddings we have seen we decided to train different models, both with or without regularization.
- Machine learning models work with numerical labels, not strings, so we map each class as follows:
  - 0 if the tweet is regular (neither sarcastic nor ironic)
  - 1 if the tweet is Ironic
  - 2 if the tweet is Sarcastic
  - 3 if the tweet is figurative (contains both irony and sarcasm)

Evaluation

- For each setup we
  - Computed precision, recall, and F1.
  - Displayed confusion matrices to visualize performance and misclassifications.

Anything wrong?

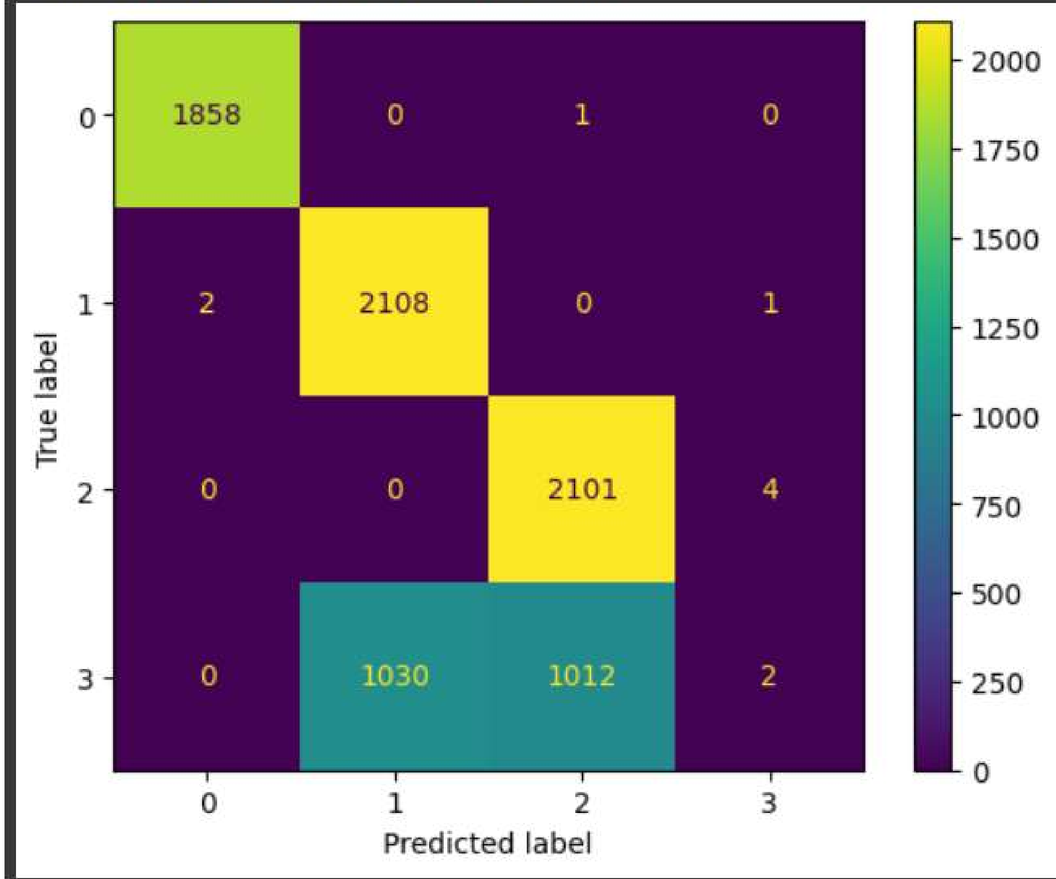
```
TFIDF BASED MODEL
Precision score with regularizer:0.6577748496853411, without regularizer:0.6182482974629323
Recall score with regularizer:0.7492797962522291, without regularizer:0.7419543270623782
F1 score with regularizer:0.6523698231729602, without regularizer:0.6484320322456364
CONFUSION MATRIX WITH REGULARIZED MODEL
```



Evaluation

- For each setup we
  - Computed precision, recall, and F1.
  - Displayed confusion matrices to visualize performance and misclassifications.

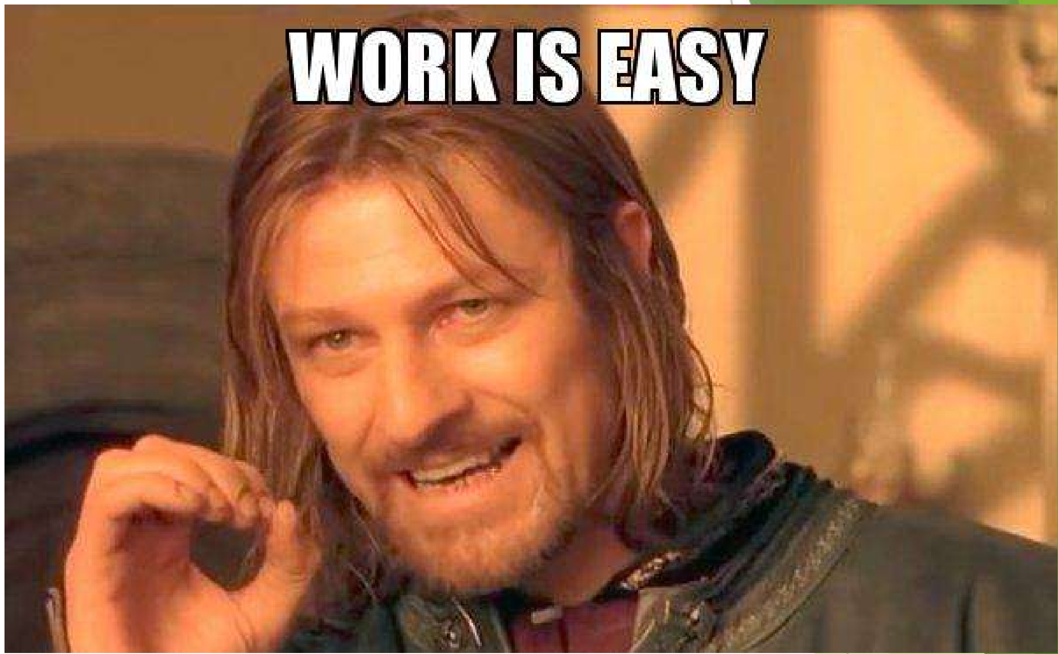
```
TFIDF BASED MODEL
Precision score with regularizer:0.6577748496853411, without regularizer:0.6182482974629323
Recall score with regularizer:0.7492797962522291, without regularizer:0.7419543270623782
F1 score with regularizer:0.6523698231729602, without regularizer:0.6484320322456364
CONFUSION MATRIX WITH REGULARIZED MODEL
```



Anything wrong?

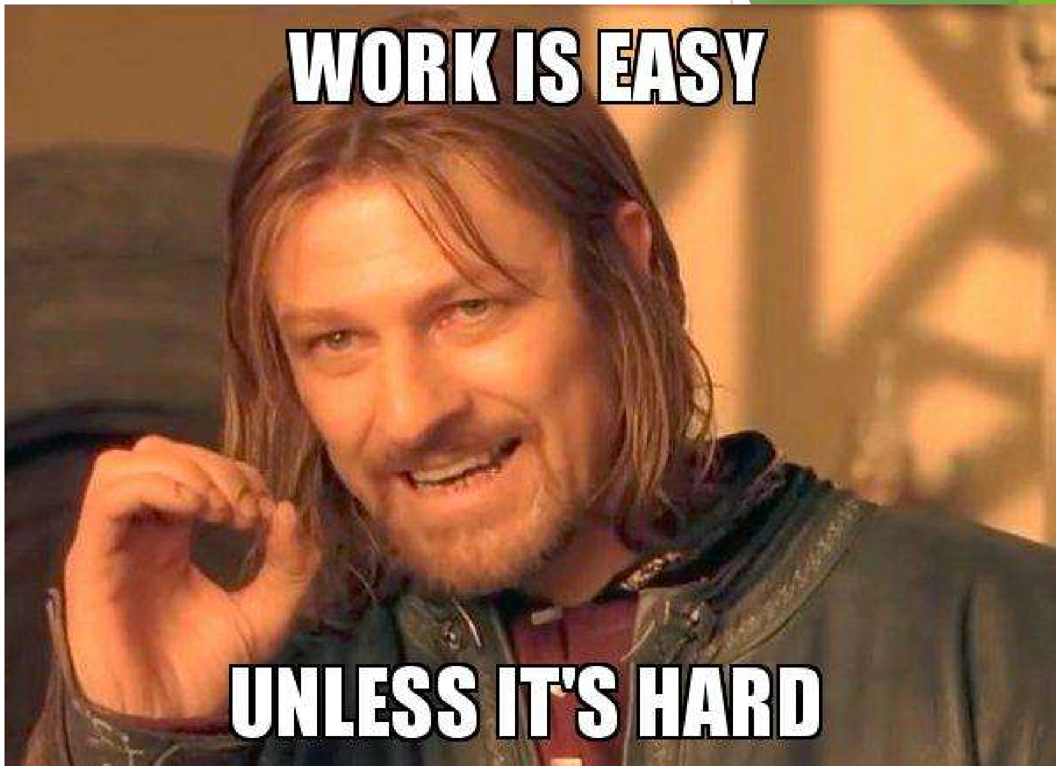
- We notice we have a suspicious situation, basically the model works flawlessly for all data except for those of figurative class. What is happening?

```
I love getting mail
saying that my
medical &amp;
personal information
has been
compromised.
#sarcasm
```



- As we can see the tweet contains #sarcasm at the end, this will be a very easy classification for our model

I love getting mail saying that my medical & personal information has been compromised. #sarcasm	figurative



- As we can see the tweet contains #sarcasm at the end, this will be a very easy classification for our model
- Turns out our dataset is even worse that we expected
- Many tweets contain either #sarcasm or #irony, which makes classifying those labels very easy.
- If none of those hashtags can be found the tweet should be regular, but very rarely we have both irony and sarcasm in the hashtags, making it almost impossible for the model to distinguish tweets containing either #sarcasm or #irony but which are actually figurative

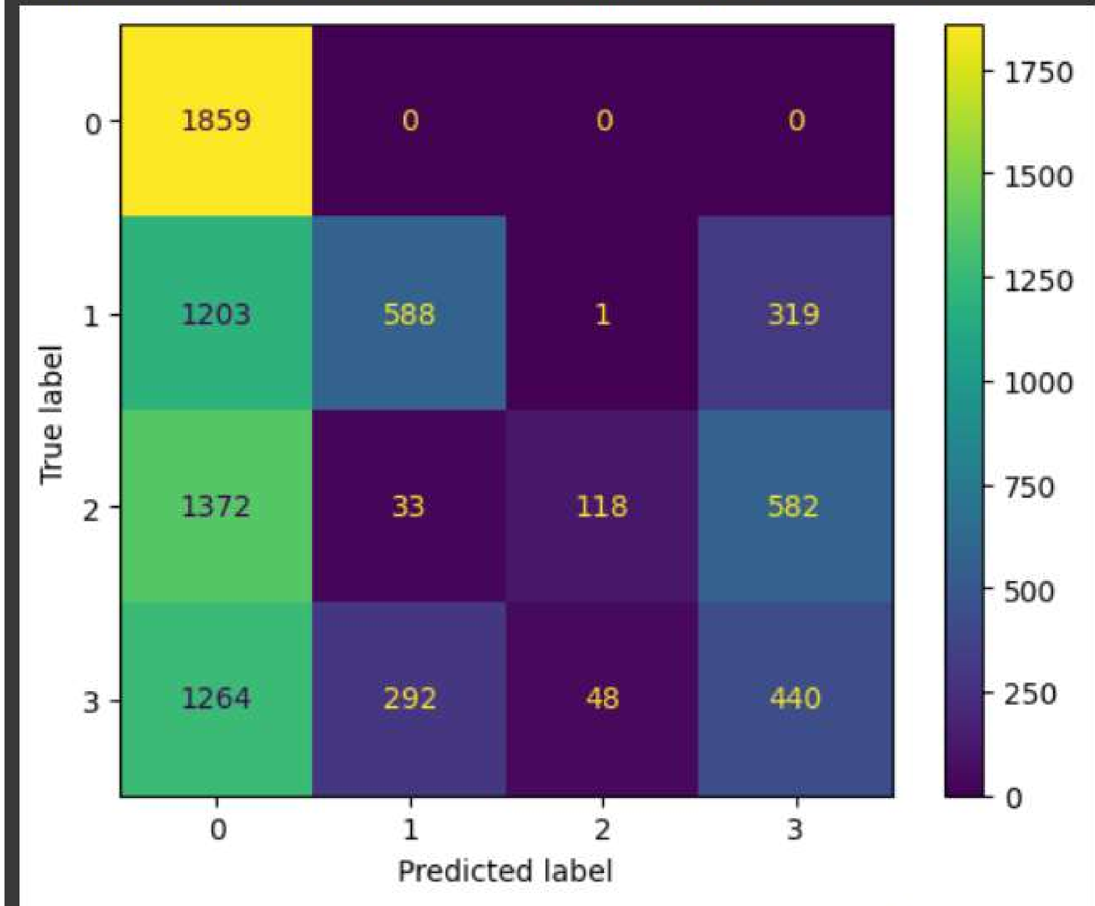
DEALING WITH MISCLASSIFICATION

- The model is clearly giving a lot of weight to whether it can use a user «hint» for his classification task.
- To produce a model which is not so dependent on the user telling which type of tweet he is making, we decided to do all previous steps again, but removing words «irony» and «sarcasm» from our dataset, so we are sure it can't have a bias toward those words.

RESULTS

- As we see the we had a noticable improvement to the missclassification of figurative tweets, but now the overall performance of the model has significantly dropped.
- It looks like the model is now struggling to find any irony or sarcasm and tends to assing the «regular» classification more often.

```
TFIDF BASED MODEL - NO IRONY & SARCASM
Precision score with regularizer:0.5012464172441904, without regularizer:0.45592474061554483
Recall score with regularizer:0.38746554270841305, without regularizer:0.49725549123147306
F1 score with regularizer:0.31118169095292925, without regularizer:0.45249130700139123
CONFUSION MATRIX WITH REGULARIZED MODEL - NO IRONY & SARCASM
```



OUR IDEAS

- We knew that removing the labels, the accuracy of the model would have dropped.
- One one hand the # are part of the tweets, so it seems fair to use them, on the other hand having a model that looks if the answer is in the question in not ideal.
- We think the very low performance in the classification could be due to the fact that the line between irony and sarcasm (and a combination of the two) is very blurry also for humans, there is no clear boundary so even the true label is in some ways subjective.
- Another explanation which is more likely is that tweets are very short and we are also removing some parts like emojis, links, foreign characters. It could be very difficult even for humans to reliably understand if a small sentence like the ones we have in the dataset is ironic, sarcastic, figurative or none.

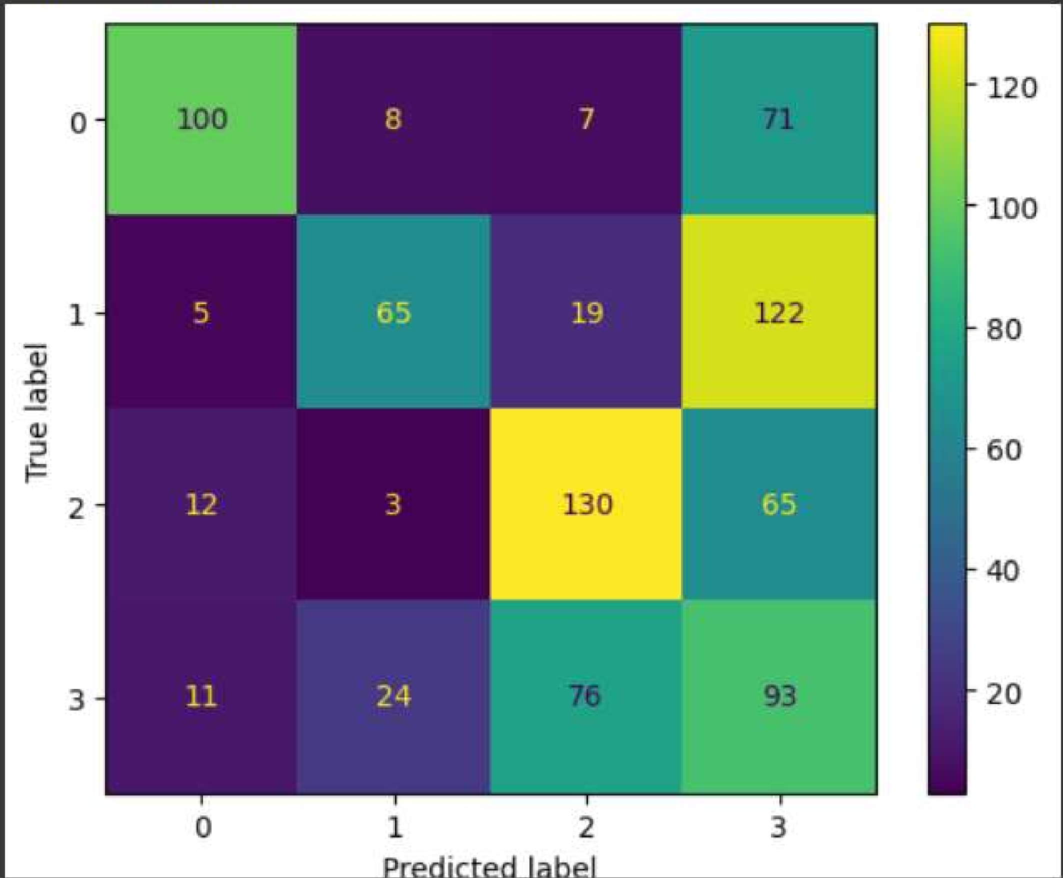
IS THERE MORE?

LLM PROMPTING

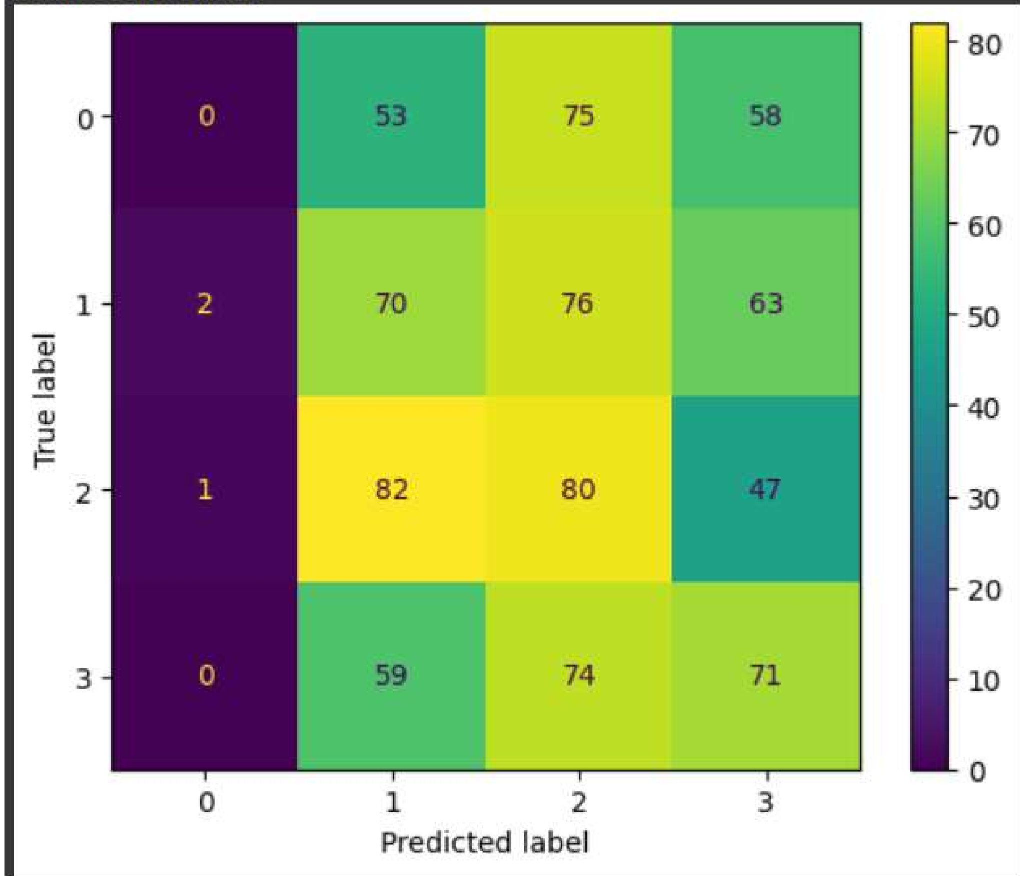
- To really see if the problem is our dataset or our model we decided to use another approach to produce our classification.
- Instead of training embeddings and using them on our dataset to produce a classification, we ask a pretrained LLM to produce the classification.
- Note that, since LLMs use an high amount of resources and we are limited by google Colab, we tested the models on just 10% of our test dataset (after shuffling it)
- We used two different ways to produce this classification:
  - **Zero shot prompting:** We give a tweet to our model and ask it to produce a single word classification just saying if the tweet is «Regular», «Sarcastic», «Ironic» or «Figurative». If the first word outputted by the model for a given tweet is none of those we count the asnwer as wrong because the model did not attend to the prompt, and we give it a random classification.
  - **Chain of thoughts prompting:** The model makes use of some «reasoning» producing a longer output, which we scan searching for the classification.

RESULTS

LLM ZERO-SHOT PROMPTING BASED MODEL
Precision score:0.5641380231358679
Recall score:0.48015531315721516
F1 score:0.49457988382004625
CONFUSION MATRIX



LLM CHAIN-OF-THOUGHT PROMPTING BASED MODEL
Precision score:0.2061294317064603
Recall score:0.26518628778525627
F1 score:0.23148955361952403
CONFUSION MATRIX

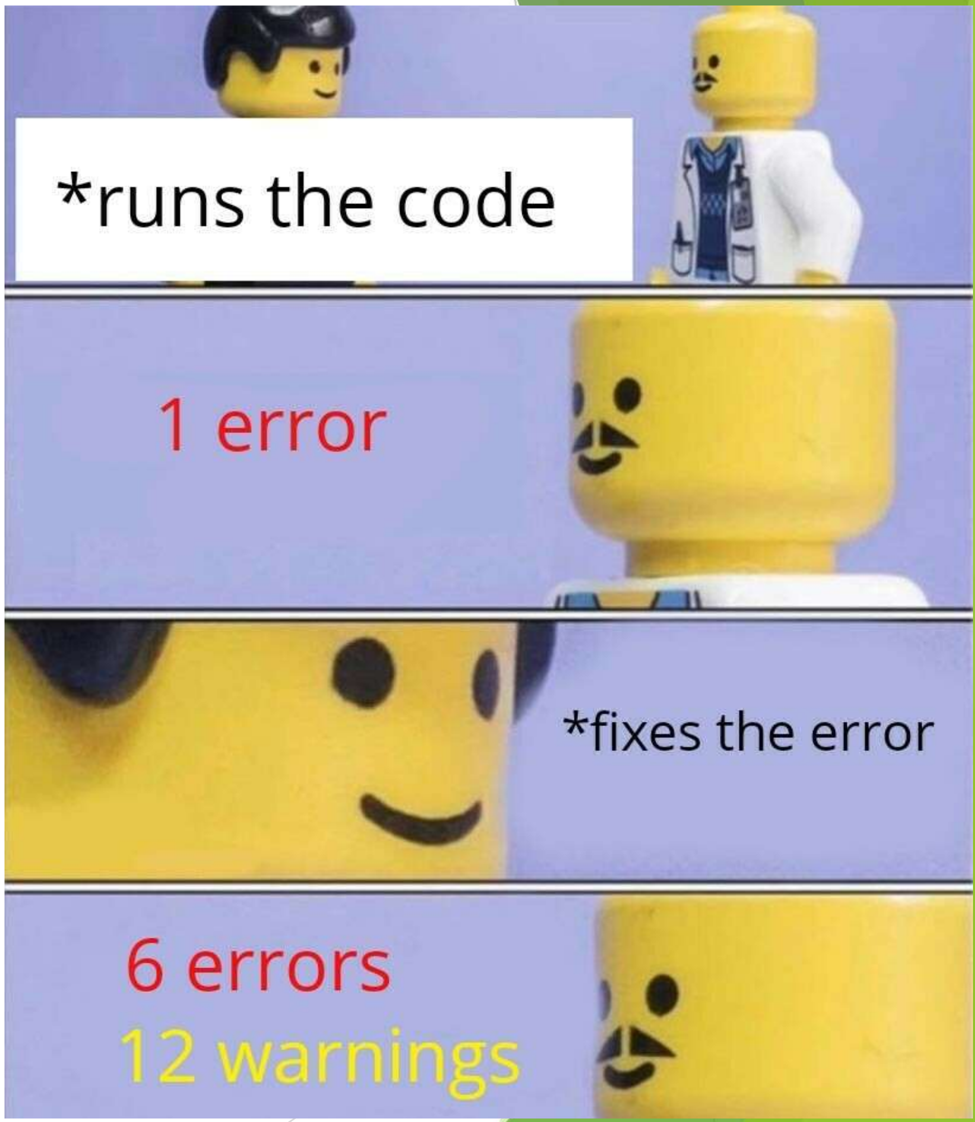


CONSIDERATIONS

- The very poor performance of LLM could be due to many factors:
  - The prompt might have not been ideal or could’ve been improved.
  - The model, having many options, felt like assinging the tweet to either «irony», «sarcasm» or a mixture of the two was a safer option in general
  - We only tested the model on few data because it takes already a very long time
  - We punished the model during zero-shot classification for not giving a direct answer
- There is not a clear winner in the classification because all regular model performs in similar ways, especially keeping the all the hastags words.
- SVD is a good compromize between performance and used resources.
- Overall we realized that our classification task presented many unseen challenges and it was definitely an interest project to develope

PROBLEMS

- The main problems of the project were:
  - **Google Colab resources limitation:** We initially also made embeddings using pure raw count, but we had to remove them from the project since running big sparse vectors is very expensive and Google Colab has a limited amount of Ram we can use.
  - **Libraries conflict and general setup steps:** We wanted to try to make the project locally but some libraries like tensorflow were creating some issues and in general it would have been more complicated to keep the project updated and word together on it.
  - **Transformer and tensorlow used different versions of numpy**, so to run the last part of the code we need to download the library, restart the session and run the necessary blocks to use the LLM prompting.



AI USAGE

- We used the help of generative AI like Copilot and ChatGPT to try to solve some problems like conflicts of the libraries, project setup using google drive and some other errors we did not know the cause of.

AI USAGE - Example

In our project the transformer library is not compatible with the version of numpy we are using, this is the version of transformers we are using:
! pip install transformers==4.48.0

Is there any version which could work with our model? We use for instance gensin library

We used version 4.48 because otherwise we had more problems, and the AI help in this case did not solve our issue

🚨 **Issue:**

You're using:

bash

```
!pip install transformers==4.48.0
```

But `transformers==4.48.0` **does not currently exist** (as of June 2025). The latest officially released version is around **4.41.2**.

So first, that's likely the root cause of a lot of version mismatches or install failures.

✅ **Recommended setup:**

bash

```
!pip install transformers==4.40.2
!pip install accelerate==0.29.2
```

> ✅ `transformers==4.40.2` is **compatible** with Phi-3 and has support for `BitsAndBytesConfig` for 4-bit quantization (which you use).

You should **also make sure** you're using:

bash

```
numpy >= 1.24.0
```

✅ Compatibility with `gensim`

`gensim` is usually quite flexible. Here's a safe environment setup that works for both LLMs and traditional NLP like Word2Vec:

Package	Version
`transformers`	`4.40.2`
`accelerate`	`0.29.2`
`torch`	`2.2.x`
`numpy`	`>=1.24.0`
`gensim`	`>=4.3.0`

Install with:

bash

```
pip install transformers==4.40.2 accelerate==0.29.2 gensim==4.3.2 numpy>=1.24
```