From Sentiment Analysis to Emotion Recognition: A NLP story

How we can use Machine Learning to recognize emotions from Social Media posts

Rodrigo Masaru Ohashi Follow
Jul 24, 2019 · 11 min read

> ***Note:*** *a Portuguese version of this article is available at "Da Análise de Sentimentos para o Reconhecimento de Emoções: Uma história PLN"*

Introduction

Recently, I've been doing some research in NLP (*Natural Language Processing*, a subfield of computer science, concerned about interactions between computers and human languages) and how to use it on our daily basis. There is a lot of work on fields like machine translation (Google Translator), dialogue agents (Chatbots), text classification (sentiment analysis, topic labeling) and many others.

Thinking about NLP data, it is possible to say that there is a lot of it, considering that millions of social media posts are being created every second. If that is not sufficient, there is a huge number of books, articles, and other sources.

Taking the social network, it has become a tool where a user can express his thoughts and feelings. Also, it's a good way to stay tuned to the events around the world. We could use its data and process it to get some interesting results.

At first, we could think about a sentiment analysis tool, where you can predict the polarity about a sentence. For instance, using the following text, got from a social media post:

> *Happy Friday my Lovelies. Spread some #kindness, #love, and #joy as you journey through your day.✨ #Shine your light and make a difference.💫 All my love.🖤 #Bkind #LoveLouder https://t.co/DJ5ddBLYYR*

We could say that it have a positive value. Thinking about a usage, this kind of tool can be used to review products on social media data. Another way to use it is to predict the reputation of a company based on what the users are saying.

Sentiment Analysis is a good tool if we just want to check the polarity of a sentence. Here we will go deeply, trying to predict the emotion that a post carries.

With emotion recognition, we can get more granular data, bringing the real thing that the user was feeling, at the moment of that specific post. It classifies the text into categories, representing the human emotions.

The first thing we need is the data itself.

Searching for the Data

Twitter Support
@TwitterSupport

😍 Twitter users love tweeting emoji from their phones –

Twitter users love tweeting emoji from their phones — now you can view these emoji on web! #emojiparty 🎉 😜 👯 🍻 🎈 🎤 🎮 🚀 🌉 ✨

1:42 PM · Apr 2, 2014 · Twitter for iPhone

4.5K Retweets 2.8K Likes

Tweet from @TwitterSupport

When working with sentiment analysis, there is a relatively large quantity of available datasets, like the *IMDb Movie Dataset*, with reviews from IMDb, and the *Stanford Sentiment Treebank*, with reviews from Rotten Tomatoes. The same thing doesn't happen to emotions. Some of the available ones are not very reliable, like the one from CrowdFlower/Figure Eight, because it uses a crowd source labeling system, and each source probably have its own way to do the classification. To work around this problem, based on some papers (see the references), we'll build our own emotion labeled dataset.

First of all, our focus will be on Twitter data. It can be fetched from the official API. You can grab the API credentials here. Using the library tweepy, we can access the API and search for any query. It returns some data, including the following:

- **id**: *Integer*. Identification number
- **created_at**: *Date Time*. Timestamp with the post creation time
- **screen_name**: *String*. Username
- **text**: *String*. Tweet text (max. 140 chars)
- **full_text**: *String*. Tweet text (max. 280 chars)

Our task is to identify the emotions from a tweet, so the data besides the text is irrelevant. As you can see, there are 2 fields containing the text: one with the old char limit, and another with the new one. When the length is above 140 chars (i.e. the old limit), the **text** field will be trimmed.

One of the downsides of the **full_text** field is that it doesn't support retweets. For that reason, we'll exclude it (retweets) from our analysis.

In supervised learning, the dataset must have a label that will be used as the class for the data. Unfortunately, there is no way to get it just from the API…

A good way to get the labels is doing it by hand, for each example, but it takes a lot of time. What if we could extract it from the text itself?

#Emotional tweets

In Social media, one common text pattern is the hashtags. Usually, the user shorten his intents using it inside his posts:

> *I feel good today! #joy*

We could extract the emotion searching for some hashtags related to the emotions. At first, the most reliable way to do it is using the value of the emotion as a hashtag (e.g. #joy).

Emojis? 😄😩😢😧

Another way to extract the emotion from the tweet is using emojis:

> *I can't believe I lost my other shoe* 😩

Like hashtags, they can express the intent of the user using a common pattern.

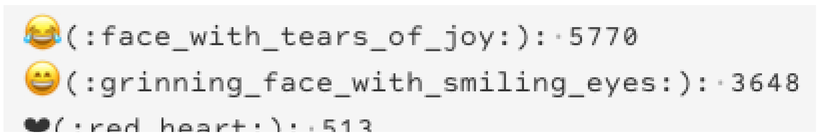
Unlikely when tweeting, it's not easy to write emojis on a console. To help us, there is a library called emoji. It converts our aliases, like *:thumbs_up:*, to the real emoji 👍.

Finding more hashtags and emojis

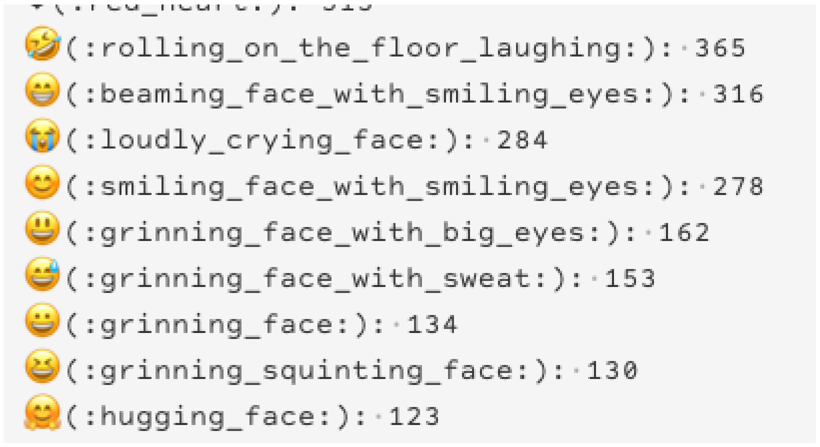
After using the known hashtags and emojis, we could explore our data and find more queries inside it.

Let's take the emotion *joy*. As the initial set of queries for that emotion, we'll use: *#joy*, the emotion itself*, #excited,* representing that a user is somehow animated*,* and the emojis 😂 and 😄 both used in place of a laugh.

After some data analysis using the collected data, we get the following emojis:



```
😂(:face_with_tears_of_joy:):·5770
😄(:grinning_face_with_smiling_eyes:):·3648
❤(:red_heart:):·513
```

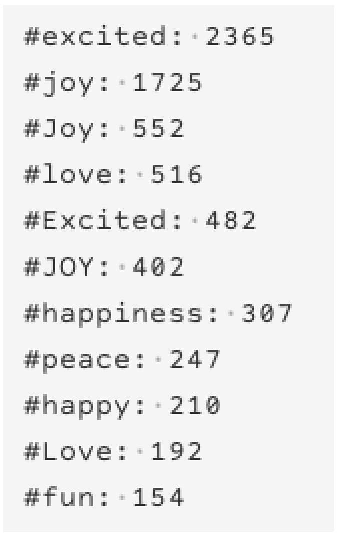


```
🤣(:rolling_on_the_floor_laughing:): 365
😁(:beaming_face_with_smiling_eyes:): 316
😭(:loudly_crying_face:): 284
😊(:smiling_face_with_smiling_eyes:): 278
😃(:grinning_face_with_big_eyes:): 162
😅(:grinning_face_with_sweat:): 153
😀(:grinning_face:): 134
😆(:grinning_squinting_face:): 130
🤗(:hugging_face:): 123
```

Emojis from the data collected with the initial set of queries

The first 2 rows are the initial queries. The rest are possible future queries.

Doing the same processing for the hashtags:



```
#excited: 2365
#joy: 1725
#Joy: 552
#love: 516
#Excited: 482
#JOY: 402
#happiness: 307
#peace: 247
#happy: 210
#Love: 192
#fun: 154
```

Hashtags from the data collected with the initial set of queries

As we discover more queries, they will be mapped to an emotion, inside a file that will be used to get more tweets later. This way, we'll build our emotion labeled dataset, until we reach a reasonable quantity of examples.

Validating the Data using Sentiment Analysis

The collected data was automatic labeled. To increase the trust on the labels, it's possible to use sentiment analysis and check the result. For instance, if a text have the label **anger**, we expect it to have a negative polarity result after predicting if from the model.

Also, let’s consider that the sentences classified with the same emotion must have a similar result values.

As dataset, we’ll use the Sentiment140, created by graduate students at Stanford University. The data is also collected from Twitter, and contains texts labeled as positive (4), negative (0) or neutral (2).

It’s important to note that it can’t prove that we labeled the examples correctly, but shows inconsistencies automatically.

Text Preprocessing

The first step is the text preprocessing. To clean the tweets, we’ll do as follows:

- **Convert to Lowercase**: Convert all characters from the text to lowercase
- **Remove special characters**: Remove links and usernames and transform emojis to text
- **Remove repetitions**: Remove char repetitions (e.g. whaaaaaat => what)
- **Remove Stop words**: Remove common stop words.

For the stop words step, it’s important to maintain negations (not, no, nor) to preserve the intention.

```
# Lowercasing
texts = texts.str.lower()

# Remove special chars
texts = texts.str.replace(r"(http|@)\S+", "")
texts = texts.apply(demojize)
texts = texts.str.replace(r"::", ": :")
texts = texts.str.replace(r"’", "'")
texts = texts.str.replace(r"[^a-z\':_]", " ")

# Remove repetitions
pattern = re.compile(r"(.)\1{2,}", re.DOTALL)
texts = texts.str.replace(pattern, r"\1")

# Transform short negation form
texts = texts.str.replace(r"(can't|cannot)", 'can not')
texts = texts.str.replace(r"n't", ' not')

# Remove stop words
stopwords = nltk.corpus.stopwords.words('english')
stopwords.remove('not')
stopwords.remove('nor')
stopwords.remove('no')
texts = texts.apply(
  lambda x: ' '.join([word for word in x.split() if word not in stopwords])
)
```

preprocess.py hosted with ❤ by GitHub view raw

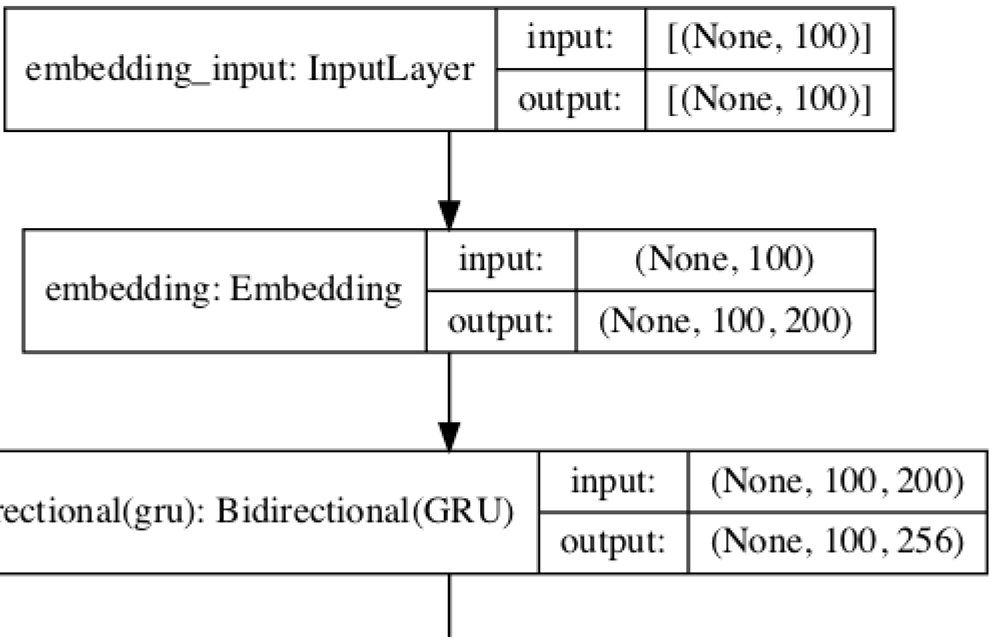
Tokenizer

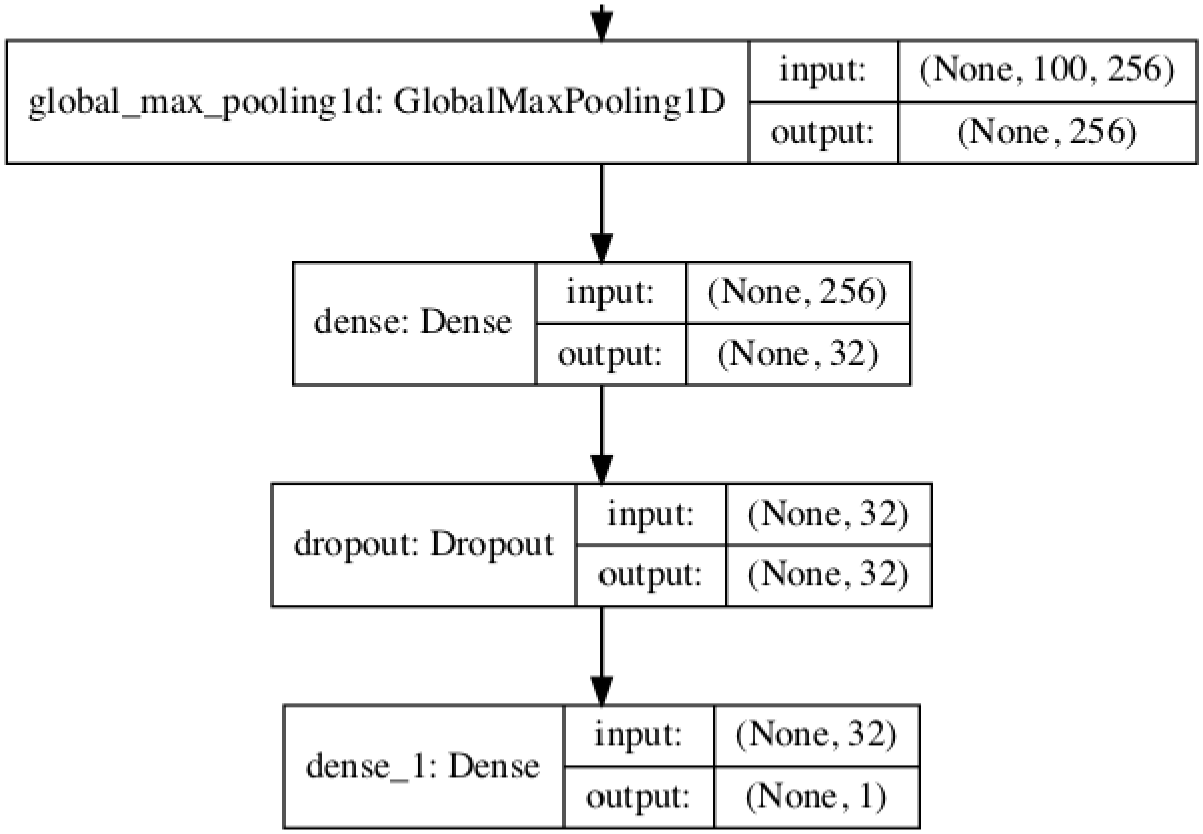
After the preprocessing, we need to transform the text corpus into a vector representation. For that task, there is a class inside the **Keras** library, simply called **Tokenizer**. It takes our data and return the desired representation that will be provided to the machine learning model.

The representation can be a one-hot vector (one value mapped to one position) or based on tf-idf score. We'll take the second option, using 10000 words.

Model

Next we define our machine learning model. For this task, a GRU (Gated Recurrent Unit) model could handle the prediction. It could be a LSTM (Long short-term memory)but there was no big difference between those 2.





Neural Network Architecture

The **Keras** implementation can be found at the GitHub repository in the end of this article.

Basically, we use a common network for this kind of task, training a non pre-trained embedding layer together. We could use pre-trained weights like GloVe or fastText, but the Twitter's data are a little bit different than the formal texts, so we'll train it from scratch.

About the output data, to simplify, we will only use the positive (4) as the value 1 and negative (0) as 0. This way, we can train a binary classification model. The more negative a tweet is, the less its result will be (near to 0).

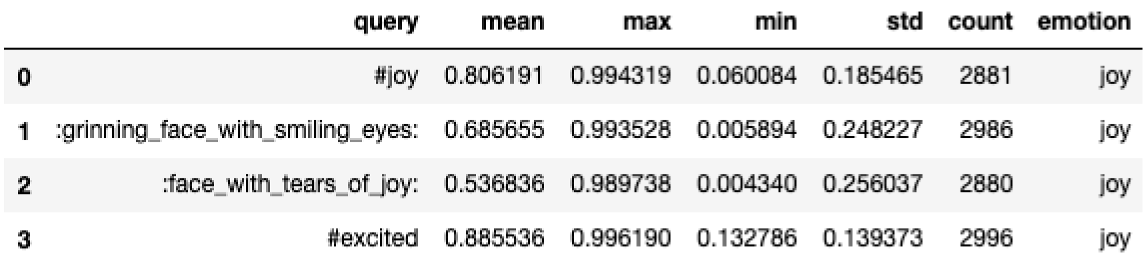
To improve the results, we could tweak the hyperparameters, like the embedding size and the *Bidirectional* layer size.

Validating queries

With the trained model, it's time to predict the polarity for the data fetched from Twitter. We'll use the file with the relations between the queries and the emotions to separate the data into categories (the emotions).

For our analysis, we'll use the mean, max, min and the standard deviation values. The result will be classified according to its class.

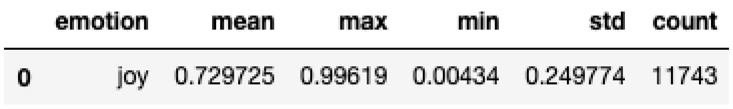
Let's check the data we collected before (with the label *joy*):



	query	mean	max	min	std	count	emotion
0	#joy	0.806191	0.994319	0.060084	0.185465	2881	joy
1	:grinning_face_with_smiling_eyes:	0.685655	0.993528	0.005894	0.248227	2986	joy
2	:face_with_tears_of_joy:	0.536836	0.989738	0.004340	0.256037	2880	joy
3	#excited	0.885536	0.996190	0.132786	0.139373	2996	joy

Sentiment analysis result for the collected data

Grouping all the results, and extracting the same values, we have the following:



	emotion	mean	max	min	std	count
0	joy	0.729725	0.99619	0.00434	0.249774	11743

Grouped results for the sentiment analysis

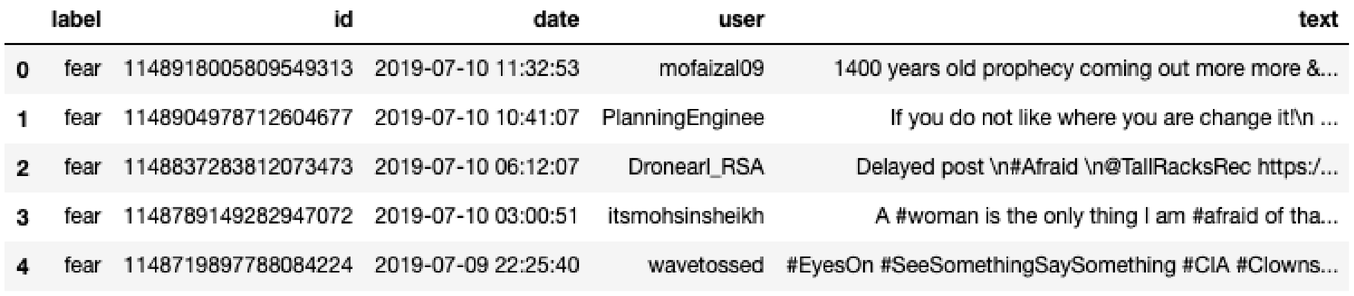
As we can see above, the mean value of the grouped result is more positive than negative. It's the expected value, since *#joy* can be classified as positive.

The query *:face_with_tears_of_joy:* (😂) have a lower mean value than the others. Comparing to the grouped result, *:grinning_face_with_smiling_eyes:* (😁) also have a result below the expectations. Probably these 2 emojis are used in other contexts too. On the other side, the hashtags (*#joy* and *#excited*) have values above the grouped mean value.

Let's take only the values that meets our expectations, considering the grouped mean value and the previously known polarity (positive in this case). We'll use the same methodology for all the other queries/emotions.

Checking the Data

After all the processing and filtering using the sentiment analysis results, a dataset with the following structure has been created:

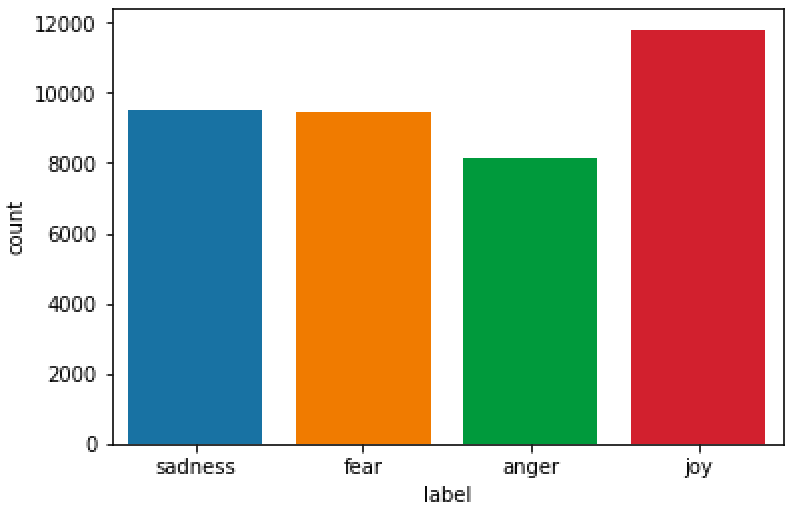


	label	id	date	user	text
0	fear	1148918005809549313	2019-07-10 11:32:53	mofaizal09	1400 years old prophecy coming out more more &...
1	fear	1148904978712604677	2019-07-10 10:41:07	PlanningEnginee	If you do not like where you are change it!\n ...
2	fear	1148837283812073473	2019-07-10 06:12:07	Dronearl_RSA	Delayed post \n#Afraid \n@TallRacksRec https:/...
3	fear	1148789149282947072	2019-07-10 03:00:51	itsmohsinsheikh	A #woman is the only thing I am #afraid of tha...
4	fear	1148719897788084224	2019-07-09 22:25:40	wavetossed	#EyesOn #SeeSomethingSaySomething #CIA #Clowns...

Emotion labeled dataset structure

The columns that we will focus are the label, with the emotion itself, and the text, containing the tweet data.

Let’s check the count of each emotion in the dataset:



Count of labels

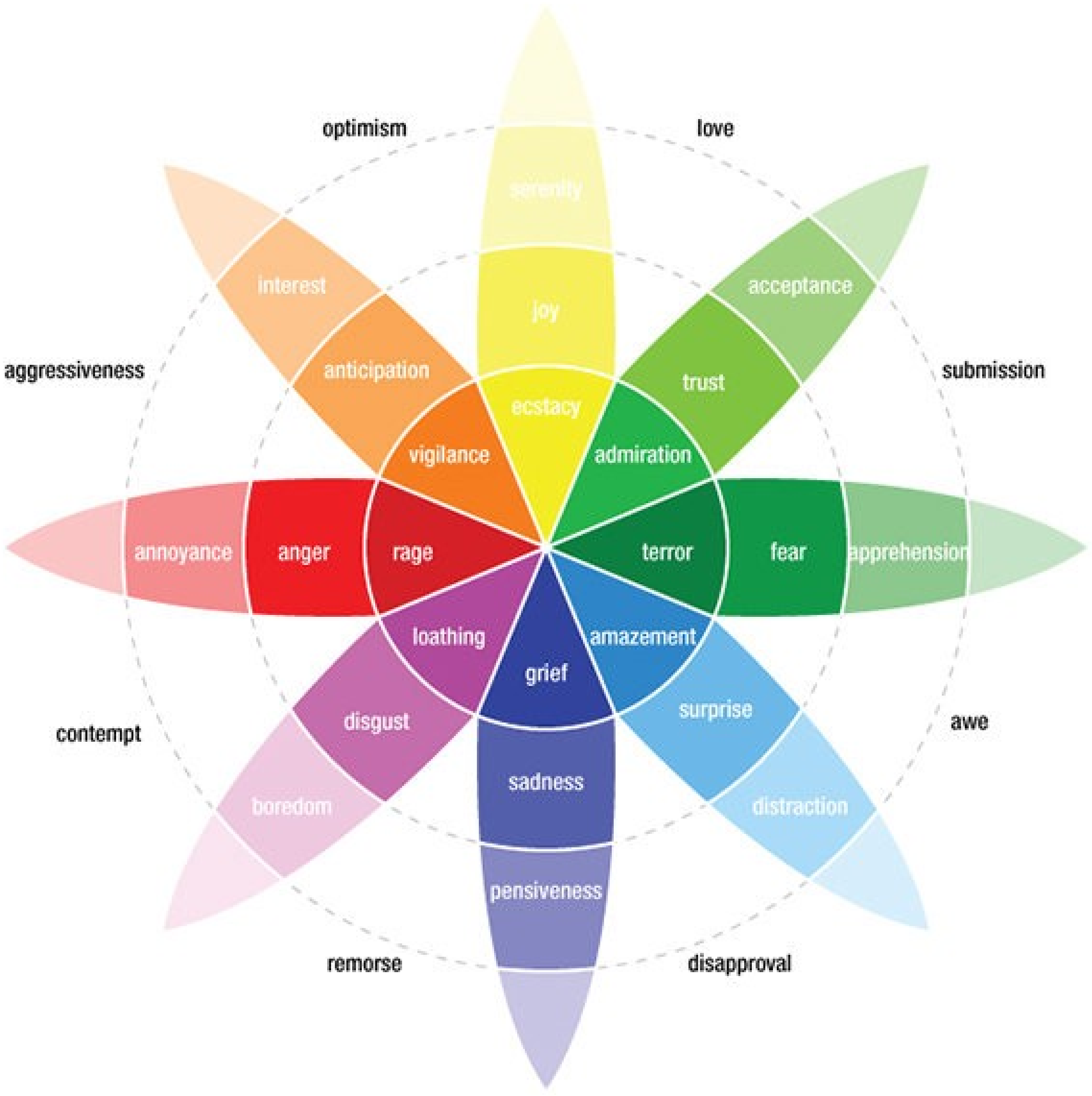
It’s always a good idea to train your models with a balanced dataset. If you see an inconsistency plotting the count graph, go back to the previous section and repeat the data gathering and analysis process until you get a balance between the labels.

Predicting Emotions

With the emotion annotated dataset in hands, we can proceed to the main and final task: build a model to predict the emotion of a text.

A lot of work from the sentiment analysis can be used here, with some minor changes. We’ll use the same tokenizer method, using the new data, and the same text preprocessing.

About the labels, there is a famous figure that represents the human emotions, called *Plutchik's Wheel of Emotions*.



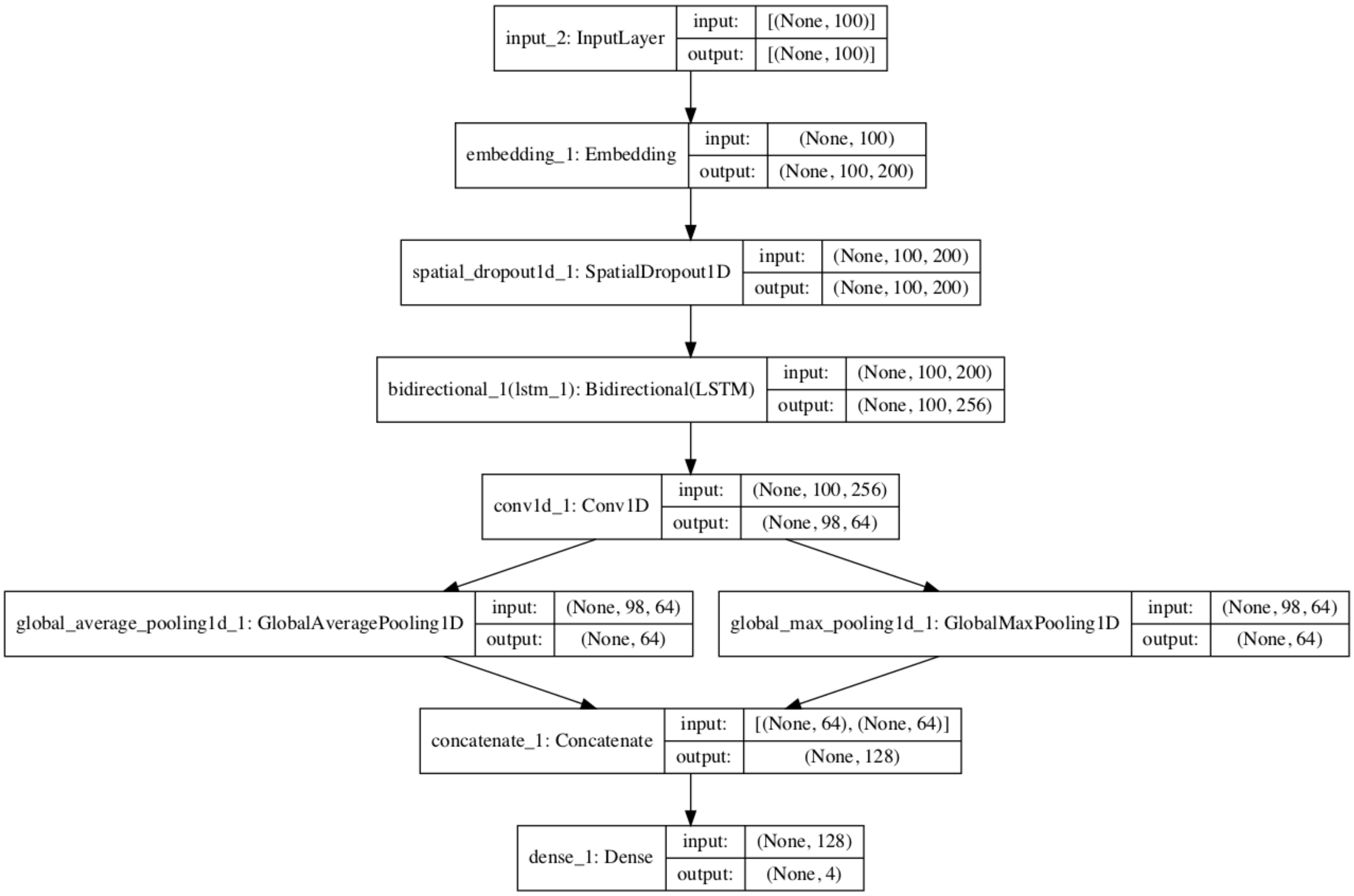
Plutchik's Wheel of emotions

As you can see, there is a large range of emotions. Here, to make the things simple, we'll use only 4: **joy**, **fear**, **sadness** and **anger**. Iterating over the process described on the previous section, you can get the labeled data for other emotions and use the same process to create a model to predict all the labels.

Model

For the machine learning model architecture, we'll use a Bidirectional **LSTM** with a **CNN** (Convolutional Neural Network)layer. The idea here is that the LSTM layer will grab the information about the context of the sentence and the CNN will extract local features.

The final architecture of the model is as follows:

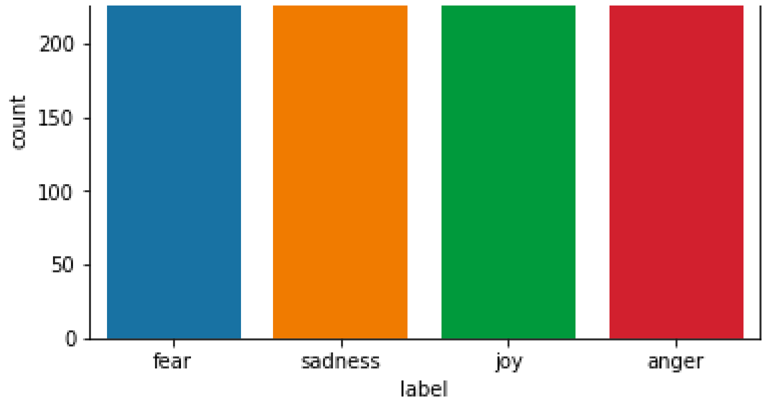


LSTM + CNN model architecture

We could tweak the embedding size, the LSTM units and the CNN filters and the kernel size to get better results.

Results

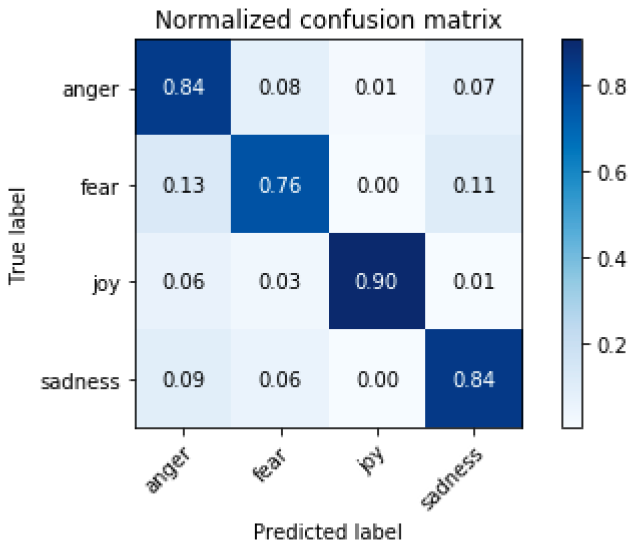
With the trained model, let's check the results. We'll use another dataset to do the test process. The data count is as follows:



Label count for the test dataset

It was generated using the same process described before.

Using our model, after predicting the labels and comparing to the real ones, the following confusion matrix was generated:



	anger	fear	joy	sadness
anger	0.84	0.08	0.01	0.07
fear	0.13	0.76	0.00	0.11
joy	0.06	0.03	0.90	0.01
sadness	0.09	0.06	0.00	0.84

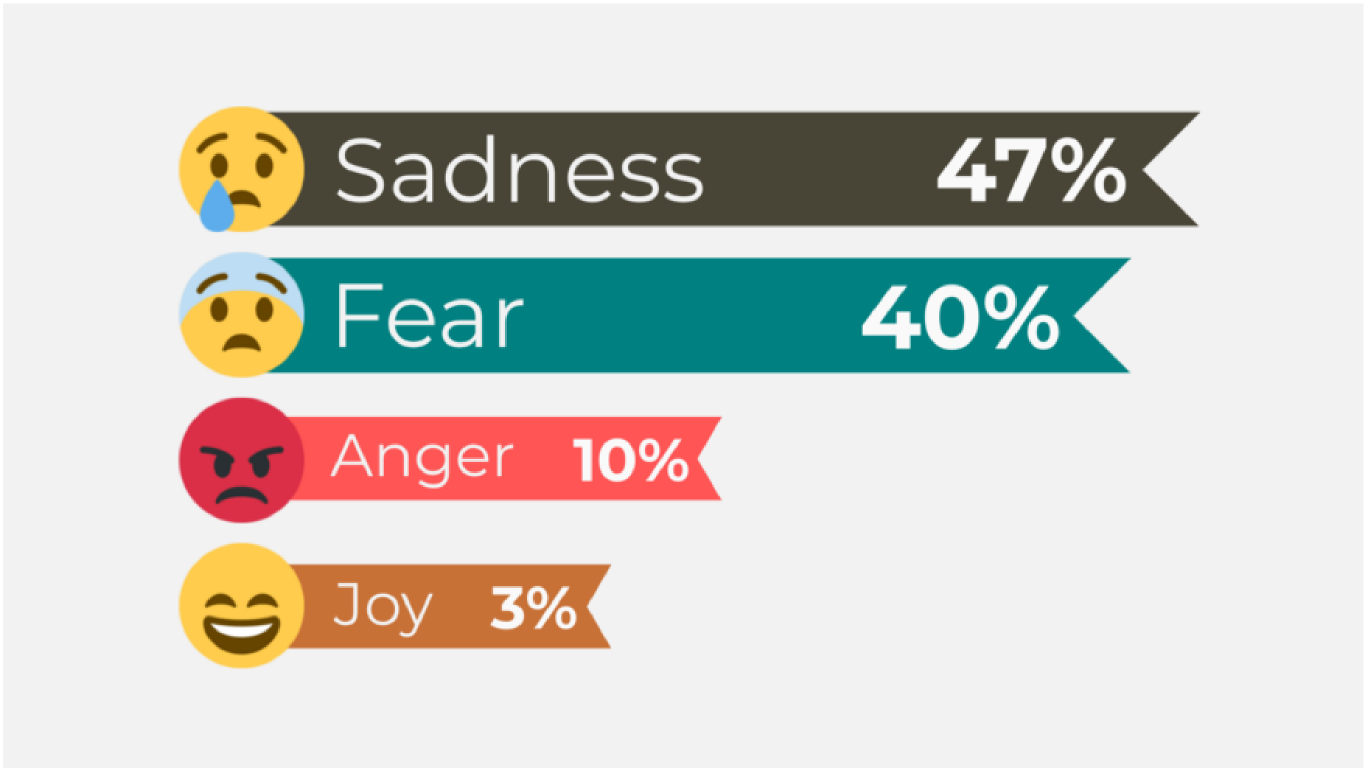
Confusion Matrix for the trained model

As you can see, we got some promising results.

Testing our Model

Finally we can test the model. Let’s take a trending topic from Twitter and use it as our query. At the time I was writing this article, Kyoto Animation (aka KyoAni), one of Japan’s most popular anime studios, was set ablaze, which killed at least 33 people and injured dozens more. Fans around the world had been posting supporting messages and lamenting what happened.

For the query “**kyoto animation**”, we expect the model to return results around the labels *“sadness”* and *“fear”*. After running the model on the data this is what it returns:



Results for the query **“kyoto animation”**

As you can see , our expectations meet the results.

Further Improvements

There are several points that could improve our model. First of all, if we build a test validation by hand, i.e. labeling each tweet one by one, the test set would reflect reality better.

Another thing that could be affecting negatively the results is threshold that we use to create the emotion labeled dataset, based on sentiment analysis result.

Finally, the number of examples in our dataset could be bigger. This way, the precision of the model could be better.

Wrapping Up

We did 3 tasks here: a process to build an emotion labeled dataset; a sentiment analysis tool to validate our data; an emotion recognition model to predict the emotion value that a tweet carry.

With that in hands, you could build a new kind of a review system, giving a more detailed version than a simple sentiment analysis. Also, it's possible to create systems to help the healthcare providers to identify some kind of mental illness before it's too late.

Lastly, as the problem can be interpreted as a text classification, the same model could be used to classify texts into other types of categories.

References

- Learning Emotion Indicators from Tweets: Hashtags, Hashtag Patterns, and Phrases
- Emoji as Emotion Tags for Tweets
- Twitter Sentiment Analysis using combined LSTM-CNN

rmohashi/emotion-from-tweet

Source code for the article "From Sentiment Analysis to Emotion Recognition: A NLP story" - rmohashi/emotion-from-tweet

github.com

Thanks to .

Machine Learning Artificial Intelligence Emotion Recognition Neural Networks