Search models, datasets, users...

← Back to blog

Getting Started with Sentiment Analysis using Python

Published February 2, 2022

Update on GitHub

federicopascual
Federico Pascual

Sentiment analysis is the automated process of tagging data according to their sentiment, such as positive, negative and neutral. Sentiment analysis allows companies to analyze data at scale, detect insights and automate processes.

In the past, sentiment analysis used to be limited to researchers, machine learning engineers or data scientists with experience in natural language processing. However, the AI community has built awesome tools to democratize access to machine learning in recent years. Nowadays, you can use sentiment analysis with a few lines of code and no machine learning experience at all! 🤯

In this guide, you'll learn everything to get started with sentiment analysis using Python, including:

1. What is sentiment analysis?
2. How to use pre-trained sentiment analysis models with Python
3. How to build your own sentiment analysis model
4. How to analyze tweets with sentiment analysis

Let's get started! 🚀

1. What is Sentiment Analysis?

Sentiment analysis is a natural language processing technique that identifies the polarity of a given text. There are different flavors of sentiment analysis, but one of the most widely used techniques labels data into positive, negative and neutral. For example, let's take a look at these tweets mentioning @VerizonSupport:

- *"dear @verizonsupport your service is straight 💩 in dallas.. been with y'all over a decade and this is all time low for y'all. i'm talking no internet at all."* → Would be tagged as "Negative".

- *"@verizonsupport ive sent you a dm"* → would be tagged as "Neutral".

- *"thanks to michelle et al at @verizonsupport who helped push my no-show-phone problem along. order canceled successfully and ordered this for pickup today at the apple store in the mall."* → would be tagged as "Positive".

Sentiment analysis allows processing data at scale and in real-time. For example, do you want to analyze thousands of tweets, product reviews or support tickets? Instead of sorting through this data manually, you can use sentiment analysis to automatically understand how people are talking about a specific topic, get insights for data-driven decisions and automate business processes.

Sentiment analysis is used in a wide variety of applications, for example:

- Analyze social media mentions to understand how people are talking about your brand vs your competitors.
- Analyze feedback from surveys and product reviews to quickly get insights into what your customers like and dislike about your product.
- Analyze incoming support tickets in real-time to detect angry customers and act accordingly to prevent churn.

2. How to Use Pre-trained Sentiment Analysis Models with Python

Now that we have covered what sentiment analysis is, we are ready to play with some sentiment analysis models! 🎉

On the Hugging Face Hub, we are building the largest collection of models and datasets publicly available in order to democratize machine learning 🚀 . In the Hub, you can find more than 27,000 models shared by the AI community with state-of-the-art performances on tasks such as sentiment analysis, object detection, text generation, speech recognition and more. The Hub is free to use and most models have a widget that allows to test them directly on your browser!

There are more than 215 sentiment analysis models publicly available on the Hub and integrating them with Python just takes 5 lines of code:

```
pip install -q transformers
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
data = ["I love you", "I hate you"]
sentiment_pipeline(data)
```

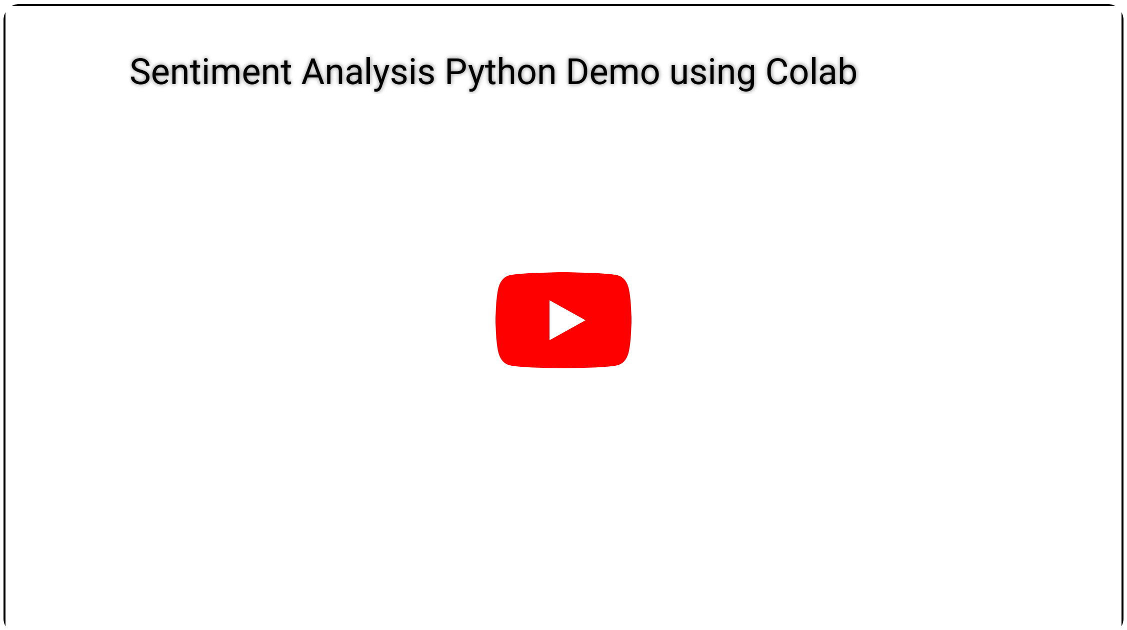
This code snippet uses the pipeline class to make predictions from models available in the Hub. It uses the default model for sentiment analysis to analyze the list of texts `data` and it outputs the following results:

```
[{'label': 'POSITIVE', 'score': 0.9998},
 {'label': 'NEGATIVE', 'score': 0.9991}]
```

You can use a specific sentiment analysis model that is better suited to your language or use case by providing the name of the model. For example, if you want a sentiment analysis model for tweets, you can specify the model id:

```
specific_model = pipeline(model="finiteautomata/bertweet-base-sentiment-analysis")
specific_model(data)
```

You can test these models with your own data using this Colab notebook:



The following are some popular models for sentiment analysis models available on the Hub that we recommend checking out:

- Twitter-roberta-base-sentiment is a roBERTa model trained on ~58M tweets and fine-tuned for sentiment analysis. Fine-tuning is the process of taking a pre-trained large language model (e.g. roBERTa in this case) and then tweaking it with additional training data to make it perform a second similar task (e.g. sentiment analysis).
- Bert-base-multilingual-uncased-sentiment is a model fine-tuned for sentiment analysis on product reviews in six languages: English, Dutch, German, French, Spanish and Italian.
- Distilbert-base-uncased-emotion is a model fine-tuned for detecting emotions in texts, including sadness, joy, love, anger, fear and surprise.

Are you interested in doing sentiment analysis in languages such as Spanish, French, Italian or German? On the Hub, you will find many models fine-tuned for different use cases and ~28 languages. You can check out the complete list of sentiment analysis models here and filter at the left according to the language of your interest.

3. Building Your Own Sentiment Analysis Model

Using pre-trained models publicly available on the Hub is a great way to get started right away with sentiment analysis. These models use deep learning architectures such as transformers that achieve state-of-the-art performance on sentiment analysis and other machine learning tasks. However, you can fine-tune a model with your own data to further improve the sentiment analysis results and get an extra boost of accuracy in your particular use case.

In this section, we'll go over two approaches on how to fine-tune a model for sentiment analysis with your own data and criteria. The first approach uses the Trainer API from the 🤗 Transformers, an open source library with 50K stars and 1K+ contributors and requires a bit more coding and experience. The second approach is a bit easier and more straightforward, it uses AutoNLP, a tool to automatically train, evaluate and deploy state-of-the-art NLP models without code or ML experience.

Let's dive in!

a. Fine-tuning model with Python

In this tutorial, you'll use the IMDB dataset to fine-tune a DistilBERT model for sentiment analysis.

The IMDB dataset contains 25,000 movie reviews labeled by sentiment for training a model and 25,000 movie reviews for testing it. DistilBERT is a smaller, faster and cheaper version of BERT. It has 40% smaller than BERT and runs 60% faster while preserving over 95% of BERT's performance. You'll use the IMDB dataset to fine-tune a DistilBERT model that is able to classify whether a movie review is positive or negative. Once you train the model, you will use it to analyze new data! ⚡

We have created this notebook so you can use it through this tutorial in Google Colab.

1. Activate GPU and Install Dependencies

As a first step, let's set up Google Colab to use a GPU (instead of CPU) to train the model much faster. You can do this by going to the menu, clicking on 'Runtime' > 'Change runtime type', and selecting 'GPU' as the Hardware accelerator. Once you do this, you should check if GPU is available on our notebook by running the following code:

```
import torch
torch.cuda.is_available()
```

Then, install the libraries you will be using in this tutorial:

```
!pip install datasets transformers huggingface_hub
```

You should also install `git-lfs` to use git in our model repository:

```
!apt-get install git-lfs
```

2. Preprocess data

You need data to fine-tune DistilBERT for sentiment analysis. So, let's use 🤗 Datasets library to download and preprocess the IMDB dataset so you can then use this data for training your model:

```
from datasets import load_dataset
imdb = load_dataset("imdb")
```

IMDB is a huge dataset, so let's create smaller datasets to enable faster training and testing:

```
small_train_dataset = imdb["train"].shuffle(seed=42).select([i for i in list(range
small_test_dataset = imdb["test"].shuffle(seed=42).select([i for i in list(range(3
```

To preprocess our data, you will use DistilBERT tokenizer:

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
```

Next, you will prepare the text inputs for the model for both splits of our dataset (training and test) by using the map method:

```
def preprocess_function(examples):
   return tokenizer(examples["text"], truncation=True)

tokenized_train = small_train_dataset.map(preprocess_function, batched=True)
tokenized_test = small_test_dataset.map(preprocess_function, batched=True)
```

To speed up training, let's use a data_collator to convert your training samples to PyTorch tensors and concatenate them with the correct amount of padding:

```
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

3. Training the model

Now that the preprocessing is done, you can go ahead and train your model 🚀

You will be throwing away the pretraining head of the DistilBERT model and replacing it with a classification head fine-tuned for sentiment analysis. This enables you to transfer the knowledge from DistilBERT to your custom model 🔥

For training, you will be using the Trainer API, which is optimized for fine-tuning Transformers🤗 models such as DistilBERT, BERT and RoBERTa.

First, let's define DistilBERT as your base model:

```
from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncase
```

Then, let's define the metrics you will be using to evaluate how good is your fine-tuned model (accuracy and f1 score):

```
import numpy as np
from datasets import load_metric

def compute_metrics(eval_pred):
   load_accuracy = load_metric("accuracy")
   load_f1 = load_metric("f1")

   logits, labels = eval_pred
   predictions = np.argmax(logits, axis=-1)
   accuracy = load_accuracy.compute(predictions=predictions, references=labels)["a
   f1 = load_f1.compute(predictions=predictions, references=labels)["f1"]
   return {"accuracy": accuracy, "f1": f1}
```

Next, let's login to your Hugging Face account so you can manage your model repositories. `notebook_login` will launch a widget in your notebook where you'll need to add your Hugging Face token:

```
from huggingface_hub import notebook_login
notebook_login()
```

You are almost there! Before training our model, you need to define the training arguments and define a Trainer with all the objects you constructed up to this point:

```
from transformers import TrainingArguments, Trainer

repo_name = "finetuning-sentiment-model-3000-samples"

training_args = TrainingArguments(
   output_dir=repo_name,
   learning_rate=2e-5,
   per_device_train_batch_size=16,
   per_device_eval_batch_size=16,
   num_train_epochs=2,
   weight_decay=0.01,
   save_strategy="epoch",
   push_to_hub=True,
)

trainer = Trainer(
   model=model,
   args=training_args,
   train_dataset=tokenized_train,
   eval_dataset=tokenized_test,
   tokenizer=tokenizer,
   data_collator=data_collator,
   compute_metrics=compute_metrics,
)
```

Now, it's time to fine-tune the model on the sentiment analysis dataset! 🙌 You just have to call the `train()` method of your Trainer:

```
trainer.train()
```

And voila! You fine-tuned a DistilBERT model for sentiment analysis! 🎉

Training time depends on the hardware you use and the number of samples in the dataset. In our case, it took almost 10 minutes using a GPU and fine-tuning the model with 3,000 samples. The more samples you use for training your model, the more accurate it will be but training could be significantly slower.

Next, let's compute the evaluation metrics to see how good your model is:

```
trainer.evaluate()
```

In our case, we got 88% accuracy and 89% f1 score. Quite good for a sentiment analysis model just trained with 3,000 samples!

4. Analyzing new data with the model

Now that you have trained a model for sentiment analysis, let's use it to analyze new data and get 🤖 predictions! This unlocks the power of machine learning; using a model to automatically analyze data at scale, in real-time ⚡

First, let's upload the model to the Hub:

```
trainer.push_to_hub()
```

Now that you have pushed the model to the Hub, you can use it pipeline class to analyze two new movie reviews and see how your model predicts its sentiment with just two lines of code 🤯 :

```
from transformers import pipeline

sentiment_model = pipeline(model="federicopascual/finetuning-sentiment-model-3000-
sentiment_model(["I love this move", "This movie sucks!"])
```

These are the predictions from our model:

```
[{'label': 'LABEL_1', 'score': 0.9558},
 {'label': 'LABEL_0', 'score': 0.9413}]
```

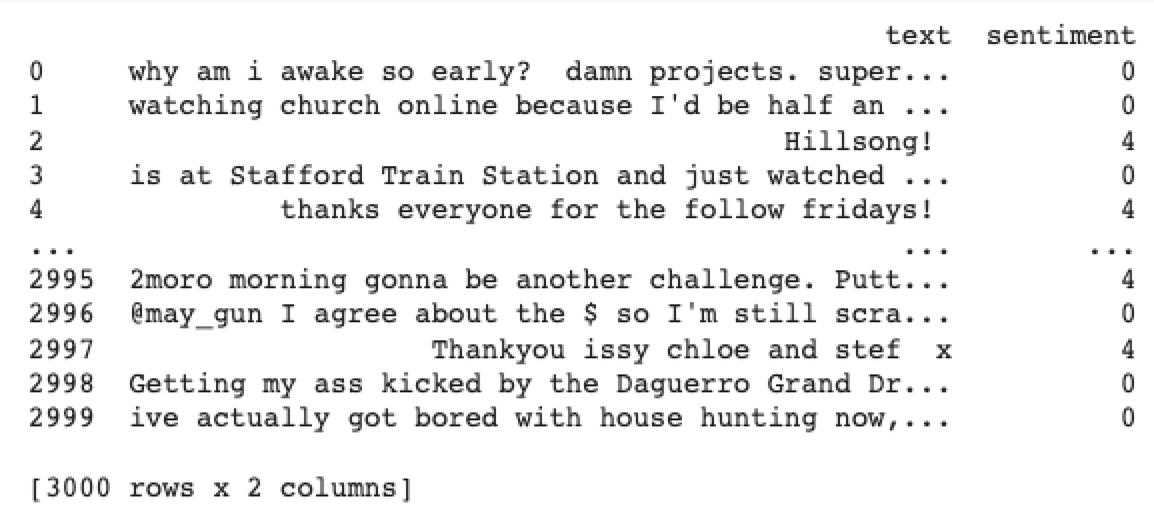
In the IMDB dataset, `Label 1` means positive and `Label 0` is negative. Quite good! 🔥

b. Training a sentiment model with AutoNLP

AutoNLP is a tool to train state-of-the-art machine learning models without code. It provides a friendly and easy-to-use user interface, where you can train custom models by simply uploading your data. AutoNLP will automatically fine-tune various pre-trained models with your data, take care of the hyperparameter tuning and find the best model for your use case. All models trained with AutoNLP are deployed and ready for production.

Training a sentiment analysis model using AutoNLP is super easy and it just takes a few clicks 🤯. Let's give it a try!

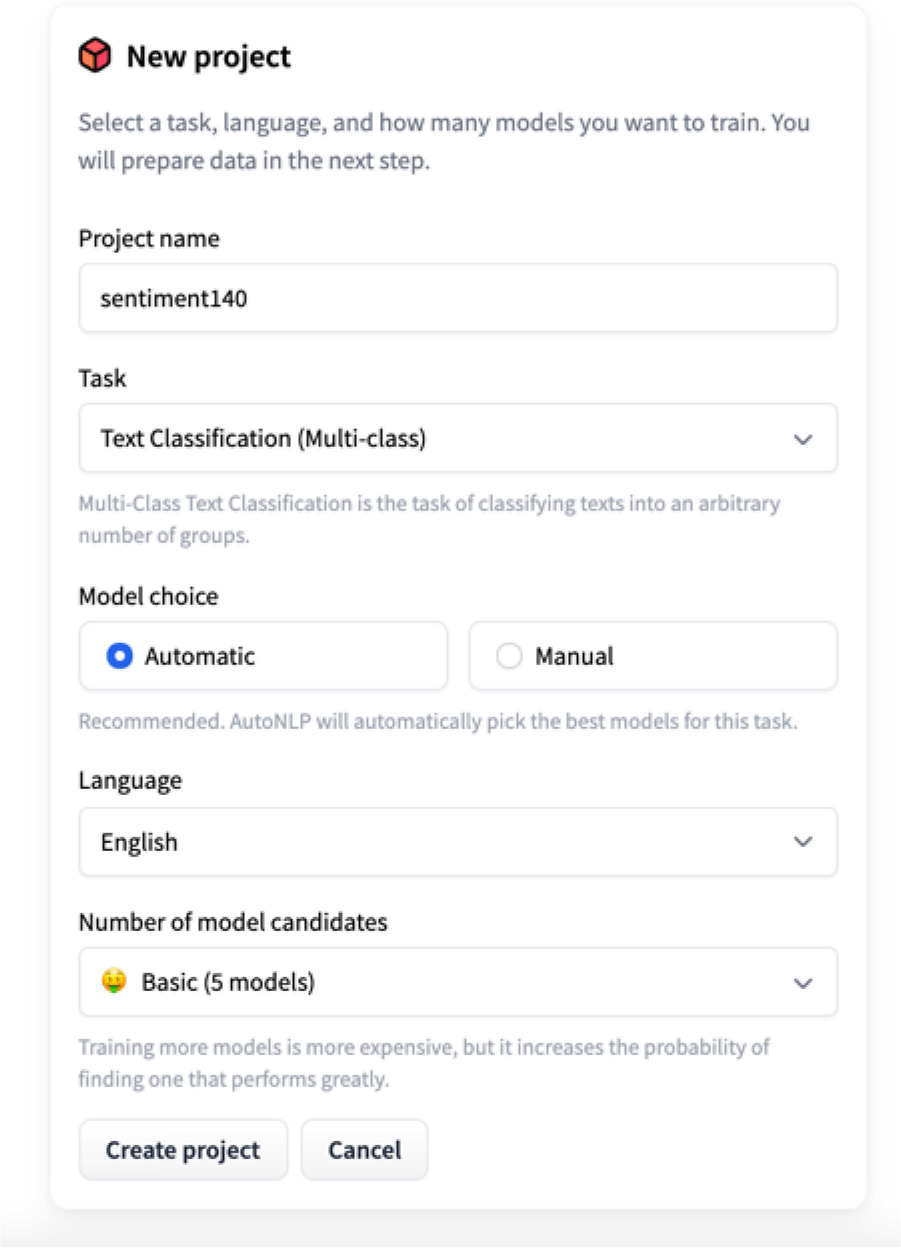
As a first step, let's get some data! You'll use Sentiment140, a popular sentiment analysis dataset that consists of Twitter messages labeled with 3 sentiments: 0 (negative), 2 (neutral), and 4 (positive). The dataset is quite big; it contains 1,600,000 tweets. As you don't need this amount of data to get your feet wet with AutoNLP and train your first models, we have prepared a smaller version of the Sentiment140 dataset with 3,000 samples that you can download from here. This is how the dataset looks like:



```
                                                   text  sentiment
0     why am i awake so early?  damn projects. super...          0
1     watching church online because I'd be half an ...          0
2                                             Hillsong!          4
3     is at Stafford Train Station and just watched ...          0
4              thanks everyone for the follow fridays!           4
...                                                 ...        ...
2995  2moro morning gonna be another challenge. Putt...          4
2996  @may_gun I agree about the $ so I'm still scra...          0
2997                    Thankyou issy chloe and stef  x          4
2998  Getting my ass kicked by the Daguerro Grand Dr...          0
2999  ive actually got bored with house hunting now,...          0

[3000 rows x 2 columns]
```

Sentiment 140 dataset

Next, let's create a new project on AutoNLP to train 5 candidate models:



New project

Select a task, language, and how many models you want to train. You will prepare data in the next step.

Project name

sentiment140

Task

Text Classification (Multi-class)

Multi-Class Text Classification is the task of classifying texts into an arbitrary number of groups.

Model choice

● Automatic ○ Manual

Recommended. AutoNLP will automatically pick the best models for this task.

Language

English

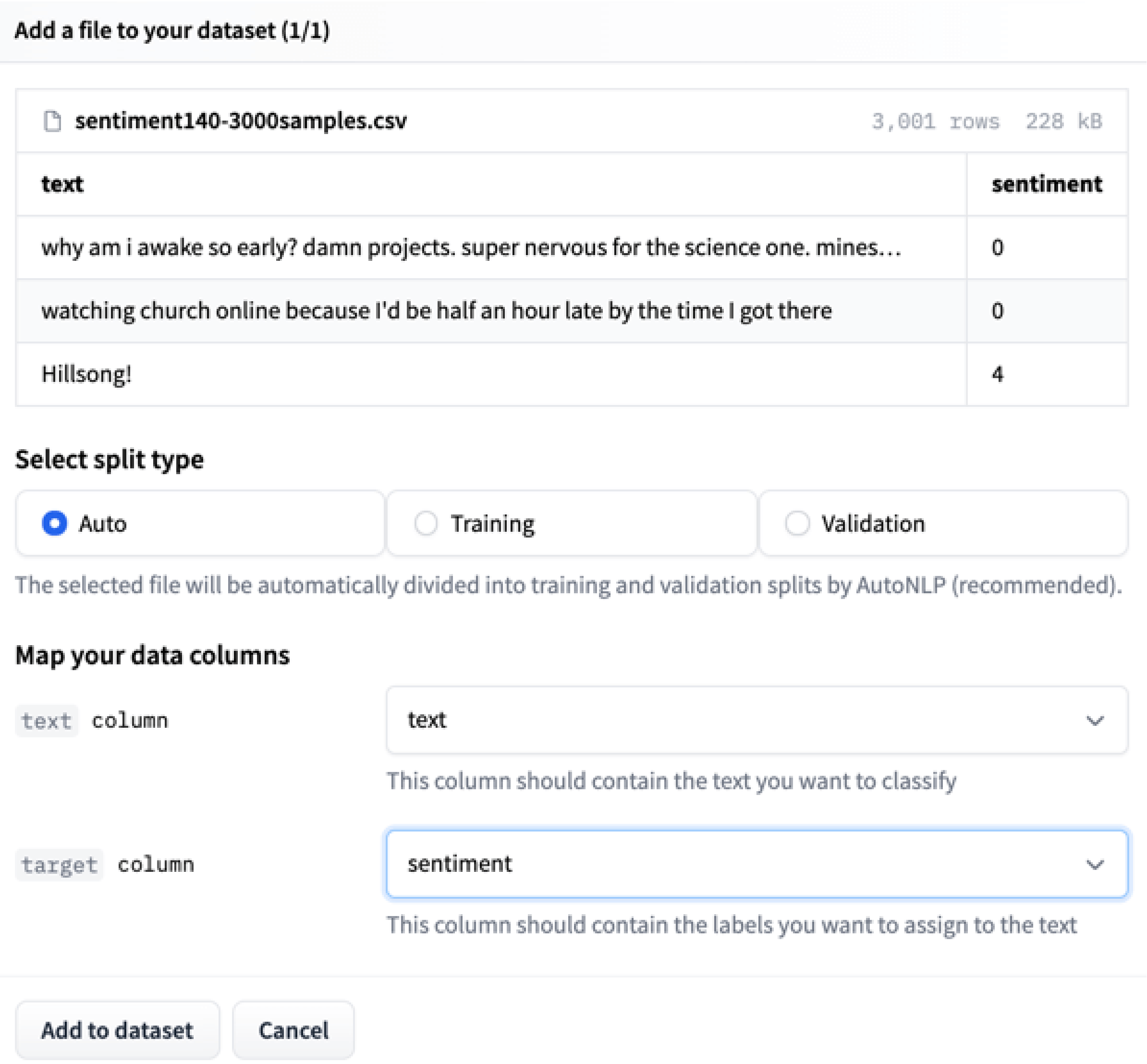
Number of model candidates

Basic (5 models)

Training more models is more expensive, but it increases the probability of finding one that performs greatly.

Creating a new project on AutoNLP

Then, upload the dataset and map the text column and target columns:



Add a file to your dataset (1/1)

sentiment140-3000samples.csv 3,001 rows 228 kB

text	sentiment
why am i awake so early? damn projects. super nervous for the science one. mines…	0
watching church online because I'd be half an hour late by the time I got there	0
Hillsong!	4

Select split type

● Auto ○ Training ○ Validation

The selected file will be automatically divided into training and validation splits by AutoNLP (recommended).

Map your data columns

`text` column: text

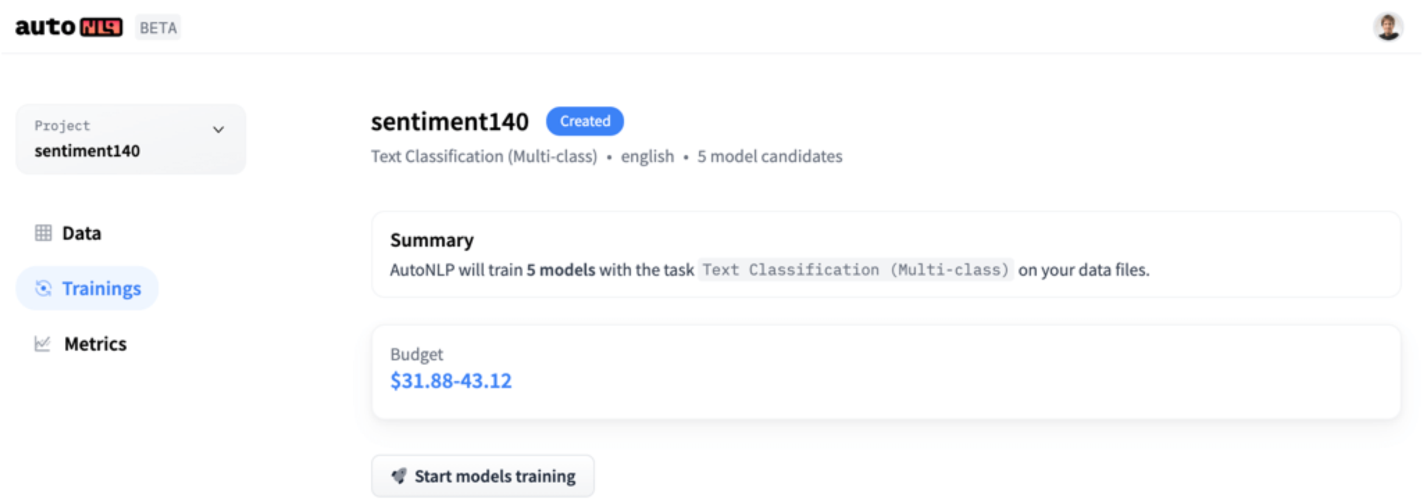
This column should contain the text you want to classify

`target` column: sentiment

This column should contain the labels you want to assign to the the text

Adding a dataset to AutoNLP

Once you add your dataset, go to the "Trainings" tab and accept the pricing to start training your models. AutoNLP pricing can be as low as $10 per model:



sentiment140 Created

Text Classification (Multi-class) • english • 5 model candidates

Summary

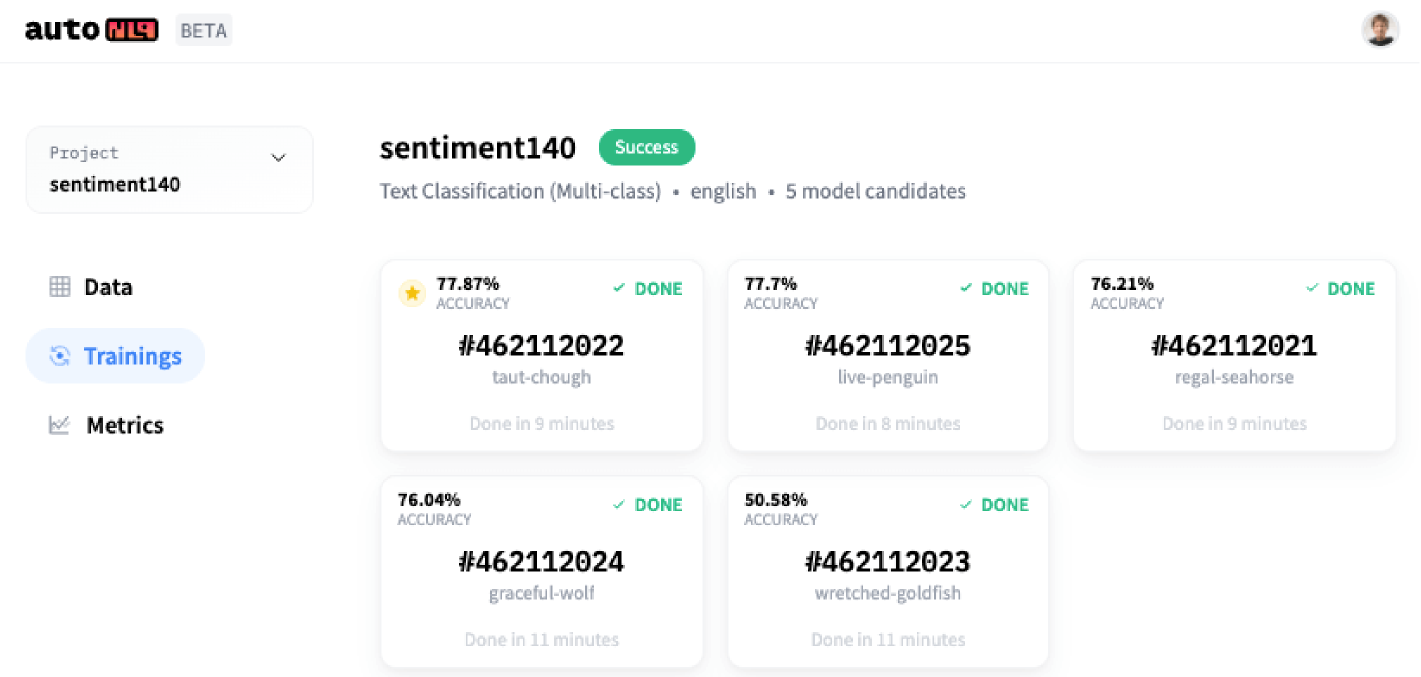
AutoNLP will train **5 models** with the task `Text Classification (Multi-class)` on your data files.

Budget

$31.88-43.12

Adding a dataset to AutoNLP

After a few minutes, AutoNLP has trained all models, showing the performance metrics for all of them:



sentiment140 Success

Text Classification (Multi-class) • english • 5 model candidates

77.87% ACCURACY ✓ DONE
#462112022
taut-chough
Done in 9 minutes

77.7% ACCURACY ✓ DONE
#462112025
live-penguin
Done in 8 minutes

76.21% ACCURACY ✓ DONE
#462112021
regal-seahorse
Done in 9 minutes

76.04% ACCURACY ✓ DONE
#462112024
graceful-wolf
Done in 11 minutes

50.58% ACCURACY ✓ DONE
#462112023
wretched-goldfish
Done in 11 minutes

Trained sentiment analysis models by AutoNLP

The best model has 77.87% accuracy 🔥 Pretty good for a sentiment analysis model for tweets trained with just 3,000 samples!

All these models are automatically uploaded to the Hub and deployed for production. You can use any of these models to start analyzing new data right away by using the pipeline class as shown in previous sections of this post.

4. Analyzing Tweets with Sentiment Analysis and Python

In this last section, you'll take what you have learned so far in this post and put it into practice with a fun little project: analyzing tweets about NFTs with sentiment analysis!

First, you'll use Tweepy, an easy-to-use Python library for getting tweets mentioning #NFTs using the Twitter API. Then, you will use a sentiment analysis model from the 🤗 Hub to analyze these tweets. Finally, you will create some visualizations to explore the results and find some interesting insights.

You can use this notebook to follow this tutorial. Let's jump into it!

1. Install dependencies

First, let's install all the libraries you will use in this tutorial:

```
!pip install -q transformers tweepy wordcloud matplotlib
```

2. Set up Twitter API credentials

Next, you will set up the credentials for interacting with the Twitter API. First, you'll need to sign up for a developer account on Twitter. Then, you have to create a new project and connect an app to get an API key and token. You can follow this step-by-step guide to get your credentials.

Once you have the API key and token, let's create a wrapper with Tweepy for interacting with the Twitter API:

```
import tweepy

# Add Twitter API key and secret
consumer_key = "XXXXXX"
consumer_secret = "XXXXXX"

# Handling authentication with Twitter
auth = tweepy.AppAuthHandler(consumer_key, consumer_secret)

# Create a wrapper for the Twitter API
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)
```

3. Search for tweets using Tweepy

At this point, you are ready to start using the Twitter API to collect tweets 🎉. You will use Tweepy Cursor to extract 1,000 tweets mentioning #NFTs:

```python
# Helper function for handling pagination in our search and handle rate limits
def limit_handled(cursor):
   while True:
       try:
           yield cursor.next()
       except tweepy.RateLimitError:
           print('Reached rate limite. Sleeping for >15 minutes')
           time.sleep(15 * 61)
       except StopIteration:
           break

# Define the term you will be using for searching tweets
query = '#NFTs'
query = query + ' -filter:retweets'

# Define how many tweets to get from the Twitter API
count = 1000

# Let's search for tweets using Tweepy
search = limit_handled(tweepy.Cursor(api.search,
                       q=query,
                       tweet_mode='extended',
                       lang='en',
                       result_type="recent").items(count))
```

4. Run sentiment analysis on the tweets

Now you can put our new skills to work and run sentiment analysis on your data! 🎉

You will use one of the models available on the Hub fine-tuned for sentiment analysis of tweets. Like in other sections of this post, you will use the pipeline class to make the predictions with this model:

```
from transformers import pipeline

# Set up the inference pipeline using a model from the 🤗 Hub
sentiment_analysis = pipeline(model="finiteautomata/bertweet-base-sentiment-analys

# Let's run the sentiment analysis on each tweet
tweets = []
for tweet in search:
   try:
     content = tweet.full_text
     sentiment = sentiment_analysis(content)
     tweets.append({'tweet': content, 'sentiment': sentiment[0]['label']})

   except:
     pass
```

5. Explore the results of sentiment analysis

How are people talking about NFTs on Twitter? Are they talking mostly positively or negatively? Let's explore the results of the sentiment analysis to find out!

First, let's load the results on a dataframe and see examples of tweets that were labeled for each sentiment:

```
import pandas as pd

# Load the data in a dataframe
df = pd.DataFrame(tweets)
pd.set_option('display.max_colwidth', None)

# Show a tweet for each sentiment
```

```
display(df[df["sentiment"] == 'POS'].head(1))
display(df[df["sentiment"] == 'NEU'].head(1))
display(df[df["sentiment"] == 'NEG'].head(1))
```

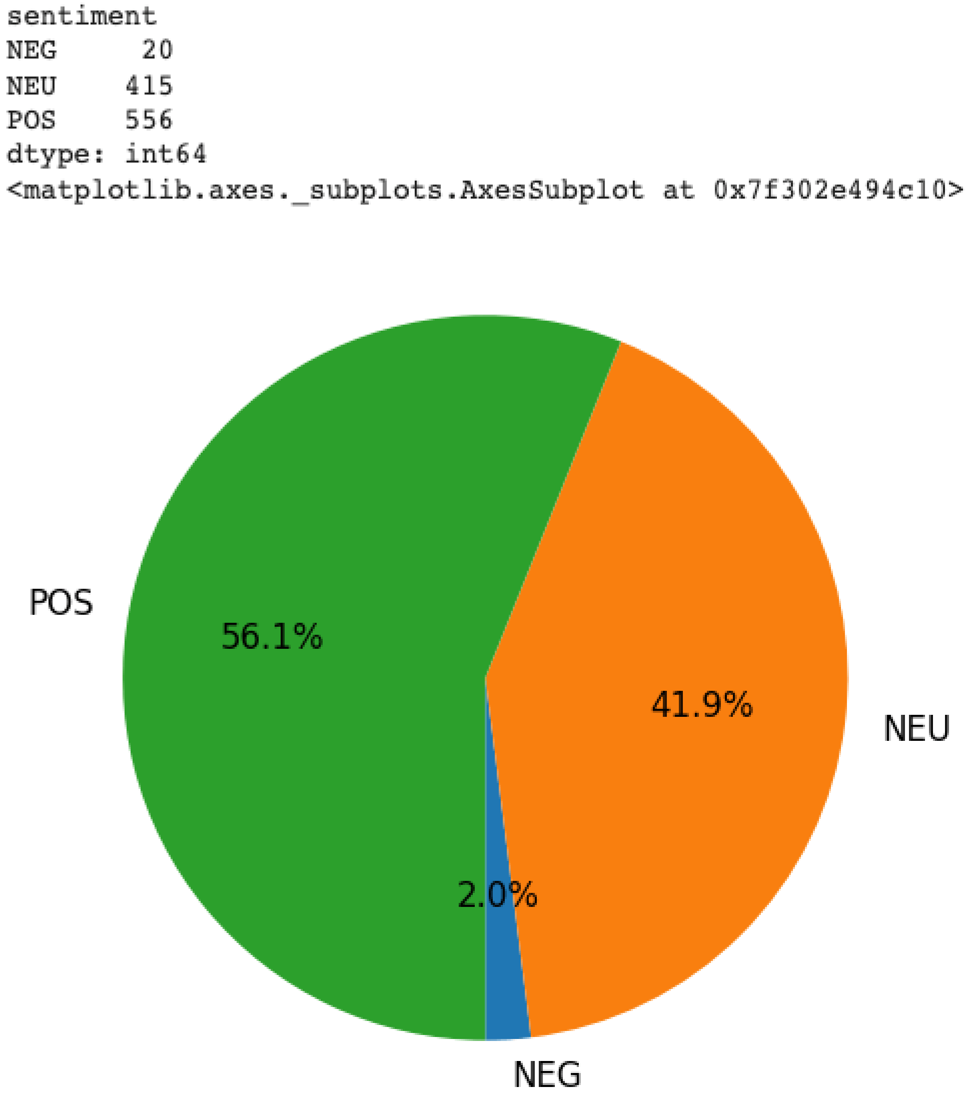
Output:

```
Tweet: @NFTGalIery Warm, exquisite and elegant palette of charming beauty Its pric


Tweet: How much our followers made on #Crypto in December:\n#DAPPRadar airdrop — $


Tweet: Stupid guy #2\nhttps://t.co/8yKzYjCYIl\n\n#NFT #NFTs #nftcollector #rarible
```

Then, let's see how many tweets you got for each sentiment and visualize these results:

```
# Let's count the number of tweets by sentiments
sentiment_counts = df.groupby(['sentiment']).size()
print(sentiment_counts)

# Let's visualize the sentiments
fig = plt.figure(figsize=(6,6), dpi=100)
ax = plt.subplot(111)
sentiment_counts.plot.pie(ax=ax, autopct='%1.1f%%', startangle=270, fontsize=12, l
```

Interestingly, most of the tweets about NFTs are positive (56.1%) and almost none are negative (2.0%):



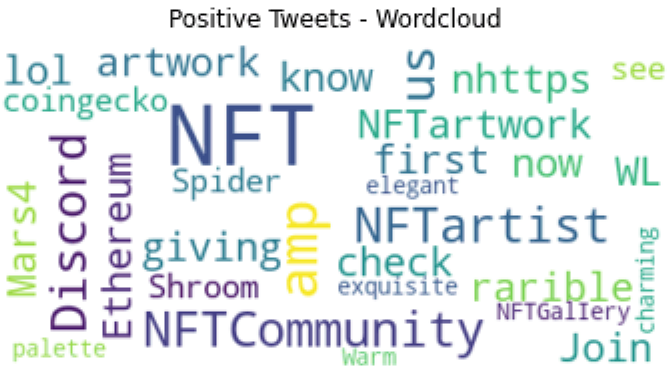
```
sentiment
NEG      20
NEU     415
POS     556
dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7f302e494c10>
```

Sentiment analysis result of NFTs tweets

Finally, let's see what words stand out for each sentiment by creating a word cloud:

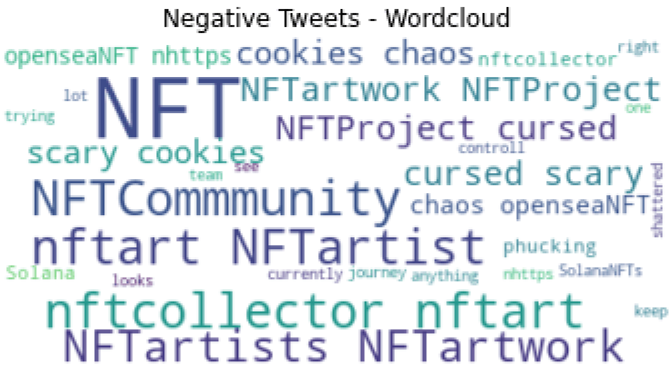
```
from wordcloud import WordCloud
from wordcloud import STOPWORDS

# Wordcloud with positive tweets
positive_tweets = df['tweet'][df["sentiment"] == 'POS']
stop_words = ["https", "co", "RT"] + list(STOPWORDS)
positive_wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="
plt.figure()
plt.title("Positive Tweets - Wordcloud")
plt.imshow(positive_wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()

# Wordcloud with negative tweets
negative_tweets = df['tweet'][df["sentiment"] == 'NEG']
stop_words = ["https", "co", "RT"] + list(STOPWORDS)
negative_wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="
plt.figure()
plt.title("Negative Tweets - Wordcloud")
plt.imshow(negative_wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Some of the words associated with positive tweets include Discord, Ethereum, Join, Mars4 and Shroom:



Word cloud for positive tweets

In contrast, words associated with negative tweets include: cookies chaos, Solana, and OpenseaNFT:



Word cloud for negative tweets

And that is it! With just a few lines of python code, you were able to collect tweets, analyze them with sentiment analysis and create some cool visualizations to analyze the results! Pretty cool, huh?

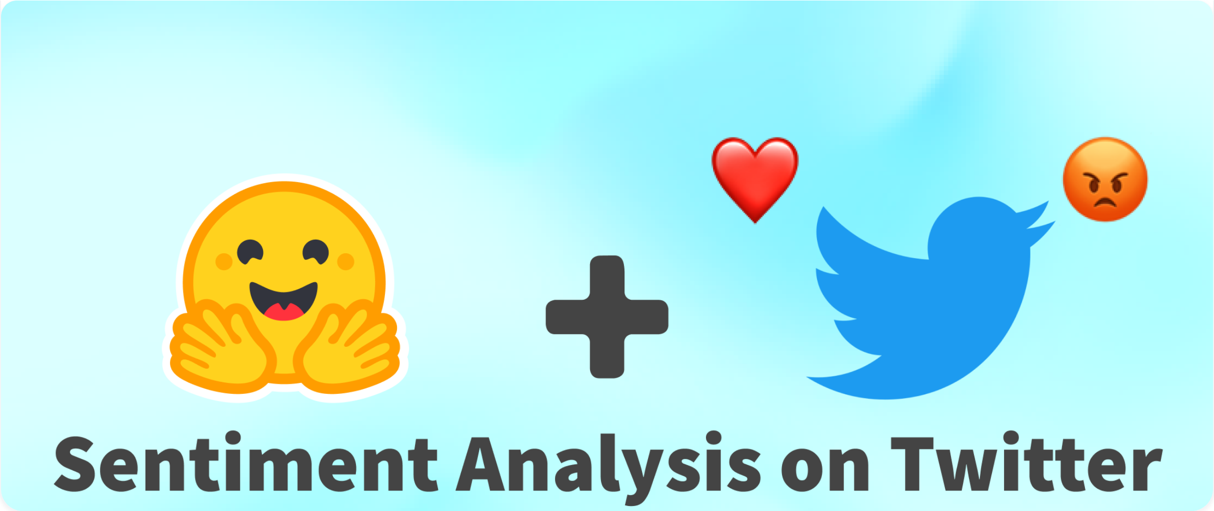
5. Wrapping up

Sentiment analysis with Python has never been easier! Tools such as 🤗 Transformers and the 🤗 Hub makes sentiment analysis accessible to all developers. You can use open source, pre-trained models for sentiment analysis in just a few lines of code 🔥

Do you want to train a custom model for sentiment analysis with your own data? Easy peasy! You can fine-tune a model using Trainer API to build on top of large language models and get state-of-the-art results. If you want something even easier, you can use AutoNLP to train custom machine learning models by simply uploading data.

If you have questions, the Hugging Face community can help answer and/or benefit from, please ask them in the Hugging Face forum. Also, join our discord server to talk with us and with the Hugging Face community.

More articles from our Blog



🤗

Company

TOS

Privacy

About

Jobs

Website

Models

Datasets

Spaces

Pricing

Docs