

Title: Fine-Tuning Sentiment Analysis Models for Covid Vaccine

Tweet Sentiments: A Comprehensive Guide

Introduction



Figure 1

In recent years, sentiment analysis has become a crucial task in natural language processing (NLP), with applications in various domains, including social media analysis, customer feedback, and market research. In this article, we will explore the process of fine-tuning pre-trained transformer models for sentiment analysis on tweets related to Covid vaccines. We will walk through the steps of data preparation, model fine-tuning, evaluation, and deployment using Hugging Face's Transformers library and Streamlit, a powerful Python web app framework.

Important and Necessary Libraries

```
#Import Libraries
import os

#Data manipulation
import pandas as pd
import numpy as np
import datasets
from sklearn.model_selection import train_test_split
import collections
import emoji
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import re
import seaborn as sns

#finetuning
from transformers import AutoTokenizer
from transformers import TrainingArguments
from transformers import AutoModelForSequenceClassification
from transformers import Trainer

#Evaluation
from datasets import load_metric, load_dataset
from sklearn.metrics import f1_score
from huggingface_hub import notebook_login
```

Figure 2

In this app for sentiment analysis on Covid vaccine-related tweets, we utilize various libraries to perform data manipulation, fine-tuning, evaluation, and visualization tasks. Each library plays a crucial role in different stages of the app's development and helps us achieve the desired outcomes effectively. Let's briefly discuss the importance of each library:

1. **Pandas and NumPy:** Pandas and NumPy are fundamental libraries for data manipulation and analysis in Python. Pandas provides powerful data structures like DataFrames, which are essential for handling and exploring structured data. NumPy, on the other hand, offers support for numerical computations and array operations, making it highly efficient for mathematical tasks.
2. **Datasets:** The datasets library is an excellent tool for managing and loading data in a standard format. It allows us to access various datasets from the Hugging Face Hub, including our custom datasets, and easily prepare the data for training and evaluation.

3. **Sklearn's train_test_split:** The `train_test_split` function from Scikit-learn helps us split our dataset into training and evaluation subsets. It enables us to create separate datasets for training the models and evaluating their performance, ensuring a robust assessment.
4. **Collections:** The `collections` module provides specialized container datatypes, such as `defaultdict` and `Counter`, which are beneficial for counting occurrences and handling data structures more efficiently.
5. **Emoji:** The `emoji` library allows us to work with emojis in text data. In our sentiment analysis, emojis might convey emotions and add context to the tweets, so this library helps in handling them effectively.
6. **Matplotlib and Seaborn:** `Matplotlib` and `Seaborn` are visualization libraries used to create various plots and graphs to explore data patterns, distribution, and relationships. We can visualize sentiment distributions, word clouds, and other insightful visualizations using these libraries.
7. **WordCloud:** `WordCloud` is a library specifically designed for generating word clouds, which are useful for visually representing the most frequent words in the tweets. It helps identify common themes and sentiments in the text data.
8. **Regular Expressions (re):** Regular expressions are powerful tools for pattern matching and text processing. We can use `regex` in tasks like cleaning and preprocessing the tweets, removing URLs, special characters, and other noise.
9. **Transformers:** The `transformers` library is a significant component of our app as it provides pre-trained transformer models, such as BERT, DistilBERT, and RoBERTa, which are state-of-the-art models for NLP tasks. It also offers tools for tokenization, model fine-tuning, and inference.
10. **Hugging Face Hub:** The `huggingface_hub` library allows us to interact with the Hugging Face Model Hub, which is a repository of pre-trained models. It enables us to save, share, and load models easily, making model deployment and collaboration seamless.

11. **TrainingArguments:** TrainingArguments is a class from the transformers library that defines various training configurations and hyperparameters for model training. It allows us to specify the number of epochs, batch sizes, evaluation strategies, and more.
12. **Trainer:** Trainer is another class from the transformers library that simplifies the model training process. It takes care of training loops, evaluation, and logging, making it easier to train and evaluate our fine-tuned sentiment analysis models.

In summary, the libraries mentioned above are essential building blocks of our sentiment analysis app. They enable us to manipulate data, fine-tune models, evaluate performance, visualize insights, and interact with the Hugging Face Model Hub, resulting in a user-friendly and efficient application to analyze public sentiments towards Covid vaccines in tweets.

Dataset and EDA

The first step in any machine learning task is data collection and exploration. Our dataset consists of tweets related to Covid vaccines, along with corresponding sentiment labels. The sentiment labels indicate whether the tweet expresses a negative, neutral, or positive sentiment towards Covid vaccines.

After loading the dataset, we performed an exploratory data analysis (EDA) to gain insights into the data's distribution and potential challenges. We identified missing values, dropped irrelevant columns, and split the data into training and evaluation sets.

```
#Load data
train_df=pd.read_csv('/content/drive/MyDrive/data/Train.csv')

#check first 5

train_df
```

	tweet_id	safe_text	label	agreement
0	CL1KWCMY	Me & The Big Homie meanboy3000 #MEANBOY #M...	0.0	1.000000
1	E3303EME	I'm 100% thinking of devoting my career to pro...	1.0	1.000000
2	M4IVFSMS	#whatcausesautism VACCINES, DO NOT VACCINATE Y...	-1.0	1.000000
3	1DR6ROZ4	I mean if they immunize my kid with something ...	-1.0	1.000000
4	J77ENIIE	Thanks to <user> Catch me performing at La Nui...	0.0	1.000000
...
9996	IU0TIJDI	Living in a time where the sperm I used to was...	1.0	1.000000
9997	WKKPCJY6	<user> <user> In spite of all measles outbrea...	1.0	0.666667
9998	ST3A265H	Interesting trends in child immunization in Ok...	0.0	1.000000
9999	6Z27IJGD	CDC Says Measles Are At Highest Levels In Deca...	0.0	1.000000
10000	P6190L3Q	Pneumonia vaccine: for women w risk of pulmona...	1.0	0.666667

10001 rows × 4 columns

Figure 3

The code provided above loads the training data from a CSV file and displays the first 5 rows of the DataFrame. Let's briefly discuss the data and its columns:

1. `tweet_id`: A unique identifier for each tweet in the dataset.
2. `safe_text`: The actual text content of the tweet, which might contain user mentions, hashtags, and other text elements.
3. `label`: The sentiment label associated with each tweet. It is represented as a numeric value: 0.0 for negative sentiment, 1.0 for neutral sentiment, and -1.0 for positive sentiment.
4. `agreement`: A numerical value representing the agreement among human annotators for the labeled sentiment. It ranges from 0.0 to 1.0, where 1.0 indicates full agreement.

The dataset consists of 10001 rows, each corresponding to a single tweet. The sentiments in the dataset have been labeled by human annotators, and the agreement score indicates how much the annotators agreed on the given sentiment label.

This training data will be used to fine-tune the sentiment analysis models, enabling them to make predictions on new, unseen tweets

The code provided below generates a word cloud (as shown in figure 4) from the text data in the "safe_text" column of the training DataFrame. The word cloud visually represents the frequency of words in the text data, with more frequent words appearing larger and bolder in the cloud. Let's briefly discuss the word cloud visualization:

1. **Word Cloud:** A word cloud is a popular data visualization technique that displays the most frequently occurring words in a given text corpus. In this case, the text corpus is the "safe_text" column of the training DataFrame, which contains the text content of tweets.
2. **Word Frequency:** Words that appear more frequently in the "safe_text" column will have a larger presence in the word cloud, making them more visible. Conversely, less frequent words will appear smaller and may be less prominent.
3. **WordCloud Parameters:** The WordCloud object is created with specific parameters:
 - `width=800, height=400`: The dimensions of the word cloud image.
 - `max_font_size=200`: The maximum font size for the most frequent words in the cloud.
 - `background_color="white"`: The background color of the word cloud, set to white.
4. **Display:** The word cloud image is displayed using Matplotlib's **`imshow`** function. The **`interpolation='bilinear'`** parameter ensures that the word cloud is displayed smoothly without pixelation.
5. **`plt.axis("off")`:** This line removes the axis and tick labels from the plot, giving a clean and focused visualization of the word cloud.

The resulting word cloud visually represents the most common words in the tweets, providing an overview of the frequent topics or sentiments present in the dataset. Larger and bolder words in the cloud indicate words that appear more frequently in the tweets.

```
# Generate a word cloud from the message text data
text = " ".join(tweet for tweet in train_df.safe_text)
wordcloud = WordCloud(width=800, height=400, max_font_size=200, background_color="white").generate_from_text(text)

# Display the word cloud image
plt.figure(figsize=(16,8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

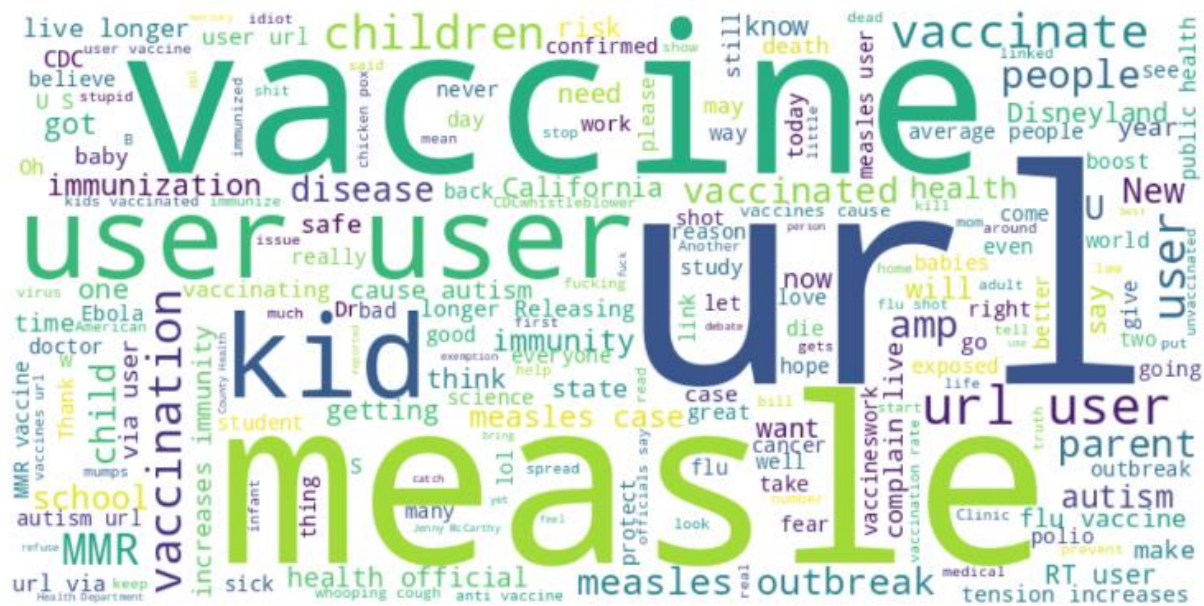


Figure 4

Visualize word frequency using bar chart

The code provided is used to visualize word frequency using a bar chart as shown in figure 5. It summarizes the top 20 most common words and their frequencies in the dataset.

Overall, the bar chart provides a visual representation of the top 20 most frequent words in the dataset, allowing users to quickly grasp the distribution of word occurrences and identify the most common words. This visualization is helpful for gaining insights into the prominent topics or sentiments present in the text data.

```
# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(top_words.keys(), top_words.values())
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 20 Most Common Words')
plt.xticks(rotation=45)
plt.show()
```

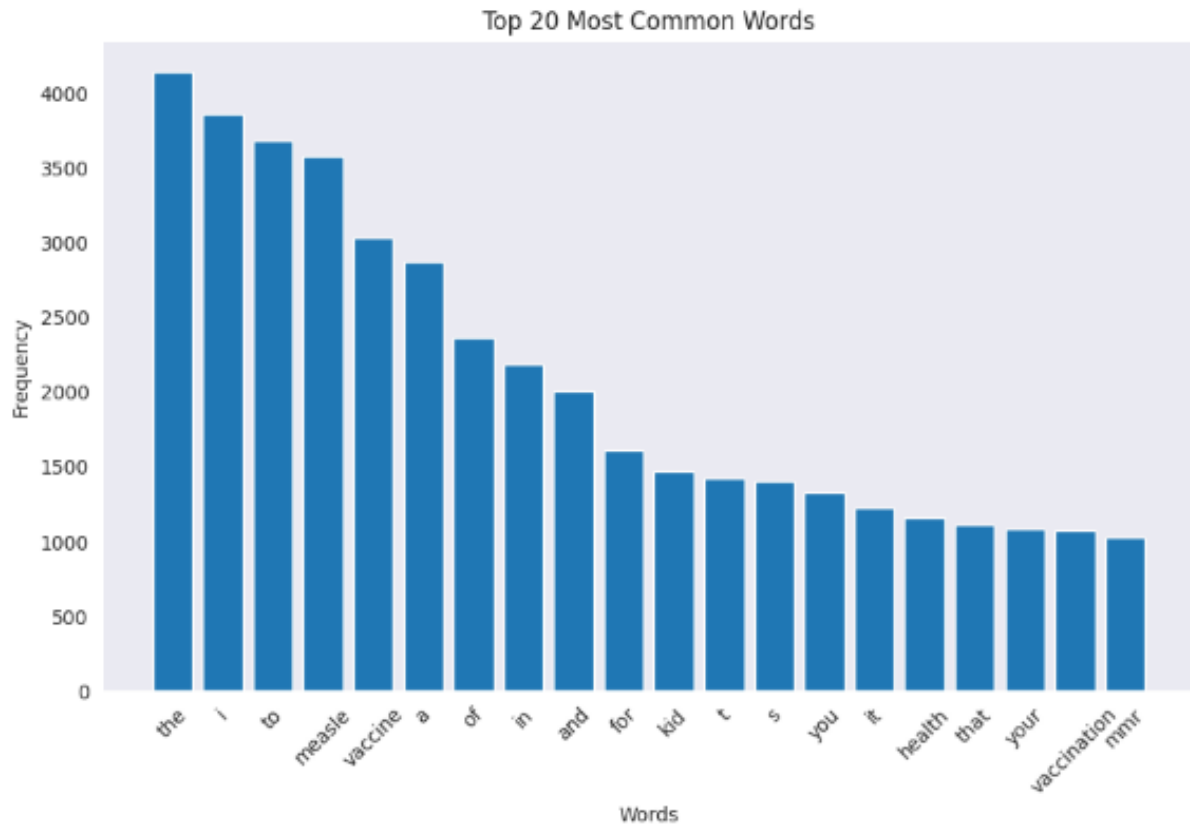


Figure 5

Check for the presence of Emojis: Emojis are a fundamental part of tweeting today

As expected we do have some emojis in our text. We can decide to fine tune algorithms that can handle emojis. Figure 6 below shows the code used to extract the emojis used our dataset and gives a summary of them as follows.

- The most frequently used emoji is 😊 with a count of 140. This emoji is commonly associated with laughter or amusement.
- The second most frequently used emoji is 💉 with a count of 63. This emoji represents a syringe and is often used in the context of healthcare, vaccinations, or medical topics.

- The third most frequently used emoji is 🤒 with a count of 58. This emoji depicts a face wearing a medical mask and is commonly used to represent sickness, health precautions, or protection.

```
]: # Create a list of all the extracted emojis
emojis_list = list(emojis)

# Count the frequency of each emoji in the list
emoji_dict = dict(collections.Counter(emojis_list))

# Sort the dictionary in descending order of frequency
emoji_dict = sorted(emoji_dict.items(), key=lambda x: x[1], reverse=True)

# Convert the dictionary to a pandas DataFrame
emoji_df = pd.DataFrame(emoji_dict, columns=['emoji', 'count'])

# Display the top 20 most frequently used emojis
print(emoji_df.head(10))
```

	emoji	count
0	😂	140
1	🔥	63
2	🤒	58
3	👉	52
4	😭	40
5	😓	39
6	💩	30
7	😞	29
8	🙏	27
9	👊	25

Figure 6

The bar plot in figure 7 gives a clear visualization of the distribution of the emojis.

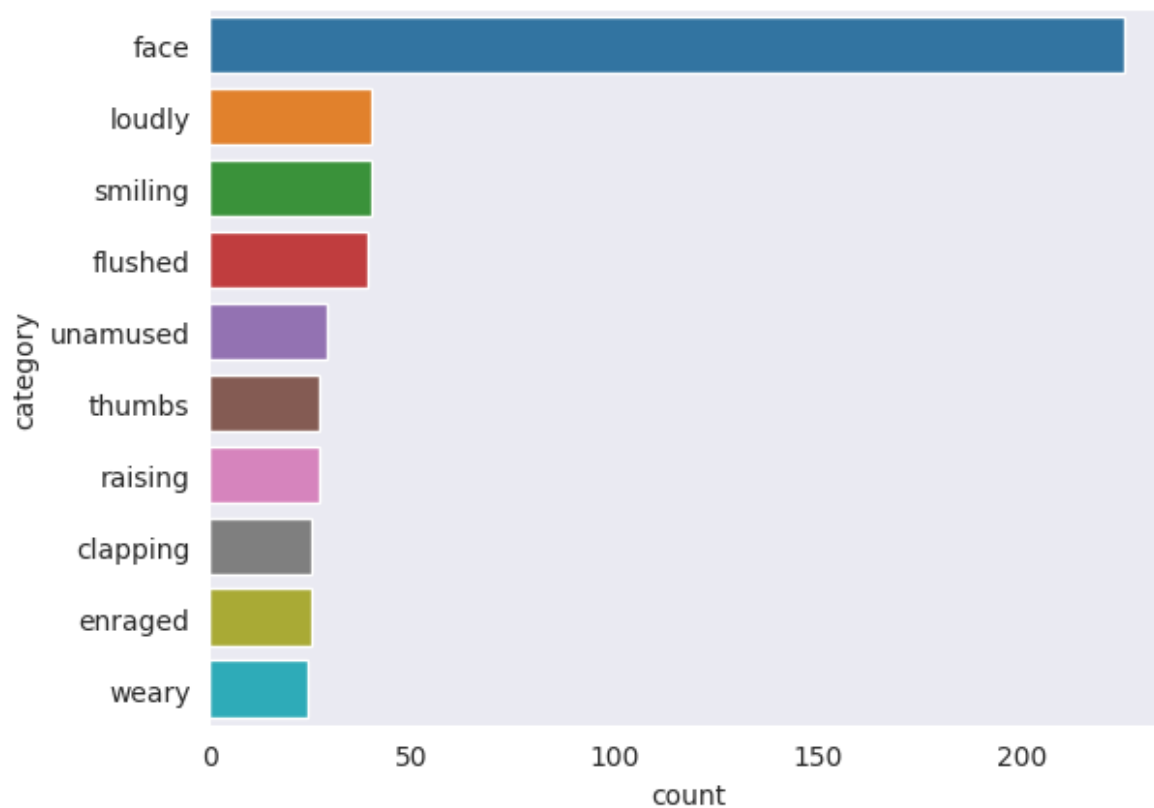


Figure 7

Distribution of Sentiment Labels

The code below is used to visualize the distribution of sentiment labels in the dataset using a bar chart as can be seen in figure 8.

Overall, the bar chart provides a visual representation of the distribution of sentiment labels in the dataset, showing the frequency of each sentiment category. This visualization is useful for understanding the balance of sentiment classes in the dataset and can be beneficial during the data exploration phase before training a sentiment analysis model.

```

1 [ ]: train_df['label'] = train_df['label'].astype('category')
      # Create a bar chart to visualize the distribution
      plt.figure(figsize=(8, 6))
      sns.barplot(x=train_df['label'].value_counts().index, y=train_df['label'].value_counts().values)
      plt.xlabel('Sentiment Label')
      plt.ylabel('Frequency')
      plt.title('Distribution of Sentiment Labels')
      plt.show()

```

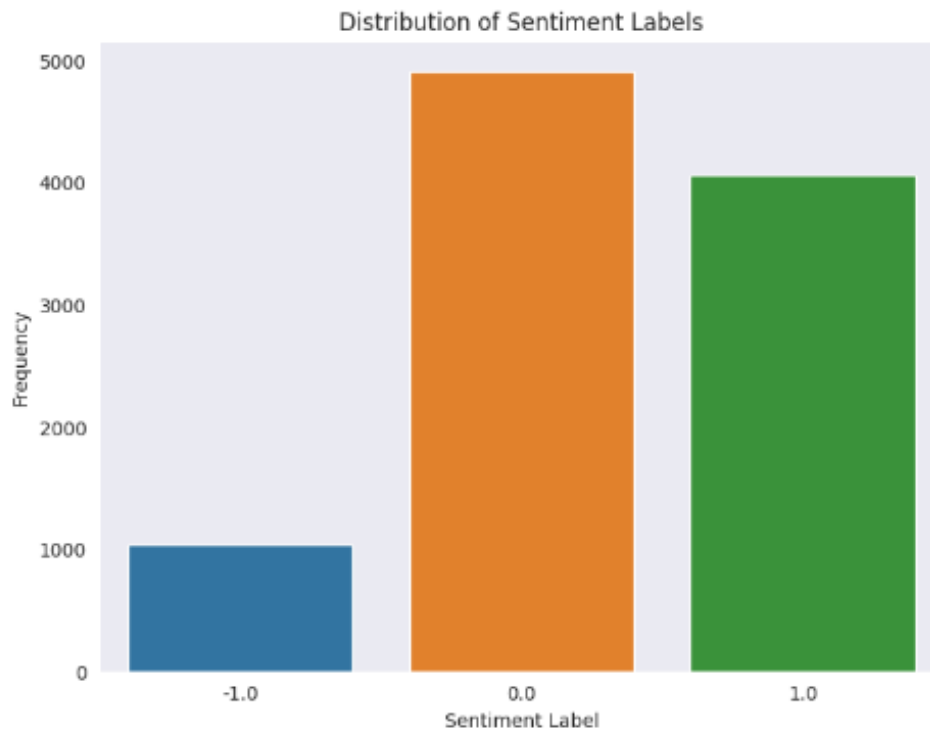


Figure 8

Visualize the distribution of sentiment labels based on the agreement level among annotators

The code below is used to plot a grouped bar chart showing the distribution of sentiment labels within each agreement level category. Overall, this grouped bar chart (figure 9) provides a clear visual representation of how sentiment labels are distributed across different agreement levels, allowing for better understanding and comparison of sentiment distribution within each category.

```
In [ ]: # Counting the sentiment labels within each agreement level category
label_counts = train_df.groupby(['agreement', 'label']).size().unstack()

# Plotting the distribution
fig, ax = plt.subplots()
agreement_levels = label_counts.index
labels = label_counts.columns
colors = ['red', 'blue', 'green']
sentiment_labels = ['negative', 'neutral', 'positive']

for i, sentiment_label in enumerate(labels):
    counts = label_counts[sentiment_label].fillna(0)
    ax.bar(agreement_levels, counts, label=f'{sentiment_labels[i]} ({sentiment_label})', color=colors[i])

ax.set_xlabel('Agreement Level')
ax.set_ylabel('Count')
ax.legend(title='Sentiment Label')

plt.title('Sentiment Distribution by Agreement Level')
plt.tight_layout()
plt.show()
```

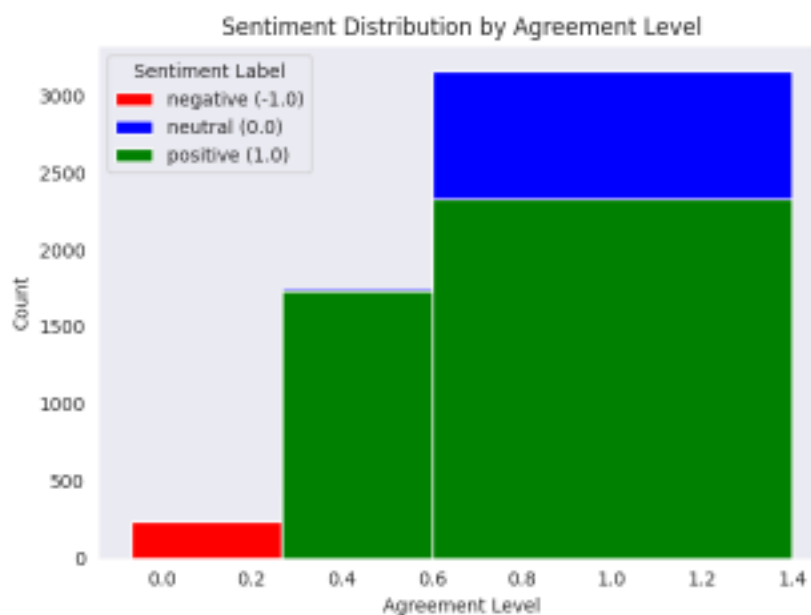


Figure 9

Keyword Frequencies by Sentiment Label

The chart in figure 11 allows us to compare the distribution and relative frequencies of keywords across different sentiment categories.

From the plot, we can summarize that:

- The words "measle(s)" are used most (over 4500 times) in the **neutral** sentiment label followed 2nd by "vaccine(s)" and 3rd by word "health".
- The words "vaccine(s)" are used most (about 2000 times) in the **positive** sentiment label followed 2nd by "measle(s)" and 3rd by word "vaccination".
- The words "vaccine(s)" are used most (about 1000 times) in the **negative** sentiment label followed 2nd by "measle(s)" and 3rd by word "vaccination".

```

1 [ ]: # Set the Labels for the sentiment Labels
      sentiment_labels = ['negative', 'neutral', 'positive']

      # Plot the stacked bar chart
      keyword_frequencies.plot(kind='bar', stacked=True, figsize=(10, 6))

      # Set the Labels and title
      plt.xlabel('Sentiment Label')
      plt.ylabel('Frequency')
      plt.title('Keyword Frequencies by Sentiment Label')

      # Show the Legend
      plt.legend(title='Keywords')

      # Set the x-axis tick Labels as the sentiment Labels
      plt.xticks(range(len(sentiment_labels)), sentiment_labels)

      # Show the plot
      plt.show()

```

Figure 10

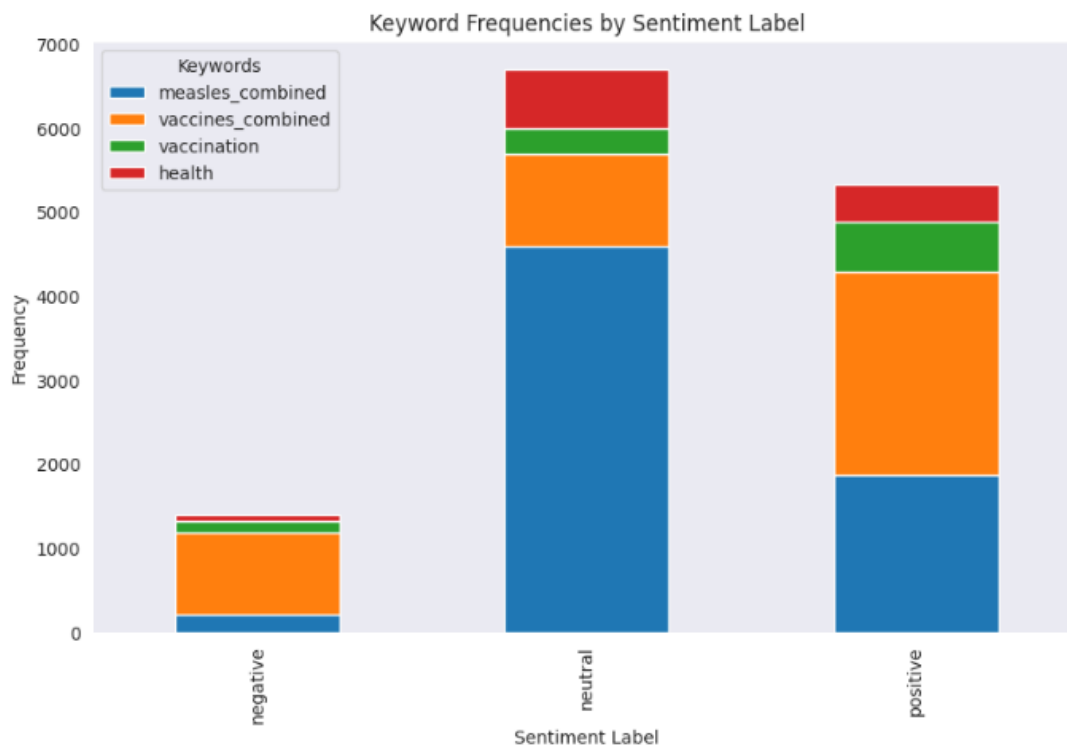


Figure 11

Model Fine-Tuning

For sentiment analysis, we used three pre-trained transformer models: BERT, DistilBERT, and RoBERTa. These models have shown excellent performance in various NLP tasks due to their attention mechanisms and deep contextual understanding of text.

To fine-tune the models for sentiment analysis, we used Hugging Face's Transformers library. We tokenized the text data using the model-specific tokenizer, transformed the sentiment labels into numerical representations, and prepared the data for training. We utilized the `TrainingArguments` class to define training configurations, including the number of epochs, evaluation strategy, and saving strategies.

Creating a Tokenizer instance

- The line of code below initializes a tokenizer for the "tweet_sentiments_analysis_roberta" model using the transformers library. The tokenizer is an essential component for processing text data and is used to convert input text into tokens that can be fed into the model for sentiment analysis.

```
#Create a tokenizer instance
tokenizer=AutoTokenizer.from_pretrained('distilbert-base-cased')

Downloading (...)okenizer_config.json:  0%|          | 0.00/29.0 [00:00<?, ?B/s]
Downloading (...)lve/main/config.json:  0%|          | 0.00/411 [00:00<?, ?B/s]
Downloading (...)solve/main/vocab.txt:  0%|          | 0.00/213k [00:00<?, ?B/s]
Downloading (...)main/tokenizer.json:  0%|          | 0.00/436k [00:00<?, ?B/s]
```

Figure 12

Functions to tokenize text and transform labels

The code below defines two functions: `tokenize_data` and `transform_label`, both of which are used to preprocess data before feeding it into a model for sentiment analysis.

1. `tokenize_data(data)`:
 - This function takes a data dictionary as input, which contains a key 'safe_text' representing the raw text to be tokenized.
 - It uses the tokenizer object (previously defined in the code) to tokenize the text data.

- The `padding='max_length'` parameter specifies that the tokenized sequences should be padded to the maximum length of the input data to ensure uniformity for model processing.
- The function then returns the tokenized data.

2. `transform_label(data)`:

- This function takes a data dictionary as input, which contains a key 'label' representing the sentiment label (e.g., -1 for Negative, 0 for Neutral, 1 for Positive).
- It maps the numerical sentiment labels to a unified label format suitable for model training and inference.
- The function first extracts the 'label' value from the input data.
- It then checks the value of 'label' and maps it to a numerical label representation (0 for Negative, 1 for Neutral, 2 for Positive).
- The function returns a new dictionary containing the transformed 'labels' with the numerical representation.

These functions play a crucial role in the data preprocessing pipeline before training the sentiment analysis model. The `tokenize_data` function converts the raw text data into tokenized sequences suitable for the model, while the `transform_label` function converts the original sentiment labels into numerical representations that the model can understand during training and evaluation.

```

#function to tokenize text
def tokenize_data(data):
    return tokenizer(data['safe_text'],padding='max_length')

#function to transform labels
def transform_label(data):

    #extract label
    label=data['label']
    num=0
    #create conditions
    if label==1: #Negative
        num=0
    elif label==0: #Neutral
        num=1
    else:
        num=2 #Positive
    return {'labels':num}

```

Figure 13

Tokenizer text and transform original sentiment labels

The code below is part of a data preprocessing pipeline using the datasets module. The pipeline prepares the dataset for sentiment analysis using tokenization and label transformation. The dataset has the columns named 'tweet_id', 'label', 'safe_text', and 'agreement'.

1. remove_columns:
 - This is a list of column names that will be removed from the dataset after tokenization and label transformation. These columns are not needed for model training and can be discarded.
2. datasets.map(tokenize_data, batched=True):
 - This line applies the tokenize_data function (previously defined in the code) to the dataset in a batched manner.
 - The tokenize_data function tokenizes the 'safe_text' column in the dataset, converting the raw text into tokenized sequences. The batched=True parameter indicates that the tokenization should be applied in batches to optimize efficiency.
3. datasets.map(transform_label, remove_columns=remove_columns):

- This line applies the `transform_label` function (previously defined in the code) to the dataset.
- The `transform_label` function transforms the original sentiment labels in the 'label' column to numerical representations (0 for Negative, 1 for Neutral, 2 for Positive).
- The `remove_columns=remove_columns` parameter ensures that the specified columns ('tweet_id', 'label', 'safe_text', 'agreement') are removed from the dataset after transformation, leaving only the tokenized text data and the numerical sentiment labels.

```

: #The following columns will be removed after tokenization
remove_columns = ['tweet_id', 'label', 'safe_text', 'agreement']

#Tokenize the text data
datasets=datasets.map(tokenize_data,batched=True)

#transform the labels
datasets=datasets.map(transform_label,remove_columns=remove_columns)

Map:   0%|          | 0/7999 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
Map:   0%|          | 0/7999 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]

```

Figure 14

Extract the train and eval datasets from datasets

The code below does the following:

1. `train_dataset = datasets['train'].shuffle(seed=0):`
 - This line creates the training dataset by extracting the 'train' subset from the `datasets` variable using the key 'train'.
 - The `shuffle(seed=0)` function is applied to the 'train' subset. This function randomly shuffles the examples in the training dataset to ensure that the order of examples seen during training is randomized, which can lead to better generalization. The `seed=0` parameter sets the random seed for reproducibility.

2. `eval_dataset = datasets['eval'].shuffle(seed=0):`

- This line creates the evaluation dataset in a similar way to the training dataset.
- The 'eval' subset is extracted from the `datasets` variable, and the `shuffle()` function is applied with the same random seed as in the previous line (`seed=0`).

```
: #extract train datasets
train_dataset=datasets['train'].shuffle(seed=0)
#extract eval datasets|
eval_dataset=datasets['eval'].shuffle(seed=0)
```

Figure 15

Define Metrics Function

The function below does the following:

1. `def compute_metrics(eval_preds)::`

- This is the definition of a Python function named `compute_metrics`.
- The function takes a single argument `eval_preds`, which is expected to be a tuple containing two arrays: logits and labels. These arrays are the predictions made by the model (logits) and the corresponding true labels (labels) for a batch of examples during evaluation.

2. `logits, labels = eval_preds:`

- This line unpacks the tuple `eval_preds` into two separate arrays `logits` and `labels`, representing the model's predictions and the true labels for a batch of examples, respectively.

3. `predictions = np.argmax(logits, axis=-1):`

- This line calculates the predicted class for each example in the batch by taking the index of the maximum value in the logits array along the last axis (`axis=-1`).
- The logits array contains raw model outputs for each class, and `np.argmax()` finds the index of the class with the highest probability for each example.

4. `f1 = f1_score(labels, predictions, average='weighted')`:
 - This line computes the weighted F1 score for the batch of examples.
 - The `f1_score()` function from the `sklearn.metrics` module is used for this calculation.
 - It takes the true labels `labels` and the predicted labels `predictions` as inputs and computes the F1 score.
 - The `average='weighted'` parameter specifies that the F1 score should be weighted by the number of samples in each class, which accounts for class imbalance.
5. `return {"f1-score": f1}`:
 - Finally, the function returns a dictionary containing the computed F1 score, with the key `"f1-score"` and the corresponding value `f1`

```
# Define the function to compute F1-score
def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=-1)
    f1 = f1_score(labels, predictions, average='weighted')
    return {"f1-score": f1}
```

Figure 16

Three models shall be trained in turns as follows:

1. 'bert-base-cased' and be named as: 'tweet_sentiments_analysis_bert'
2. 'distilbert-base-cased' and be named as: 'tweet_sentiments_analysis_distilbert'
3. 'roberta-base' and be named as: 'tweet_sentiments_analysis_roberta'

Set the training arguments

In summary, the `TrainingArguments` object is used to define various training configurations and hyperparameters, such as the number of training epochs, evaluation and saving strategies, and whether to push the model to the Hugging Face Model Hub after training. This object is then passed to the `Trainer` class for model training using the specified settings.

```
]: #set the training arguments
trainargs=TrainingArguments('tweet_sentiments_analysis_distilbert',
                             num_train_epochs=5,
                             evaluation_strategy="epoch",
                             save_strategy='epoch',
                             load_best_model_at_end=True,
                             push_to_hub=True)
```

Figure 17

Create an instance of the Model

The code below creates a pre-trained sequence classification model with three output labels and initializes it with the pre-trained weights from the model. This model can be fine-tuned for a specific classification task, such as sentiment analysis, with the appropriate training data.

```
: # Create an instance of the model model
model_name = 'distilbert-base-cased'
num_labels = 3
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=num_labels)

Downloading pytorch_model.bin: 0%|          | 0.00/263M [00:00<?, ?B/s]
```

Figure 18

Create trainer instance

The code below creates a Trainer instance with the necessary components, including the model, training and evaluation datasets, tokenizer, and a function for computing evaluation metrics. The Trainer instance can be used to train and evaluate the pre-trained sequence classification model on the specified datasets using the defined training arguments.

```
]: # Create a Trainer instance
trainer = Trainer(
    model=model,
    args=trainargs,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
```

Figure 19

Start training

The code below `trainer.train()` is used to initiate the training process of the pre-trained sequence classification model using the specified training dataset and training arguments.

When this code is executed, the model will start training for the specified number of epochs (defined in the training arguments) on the provided training dataset.

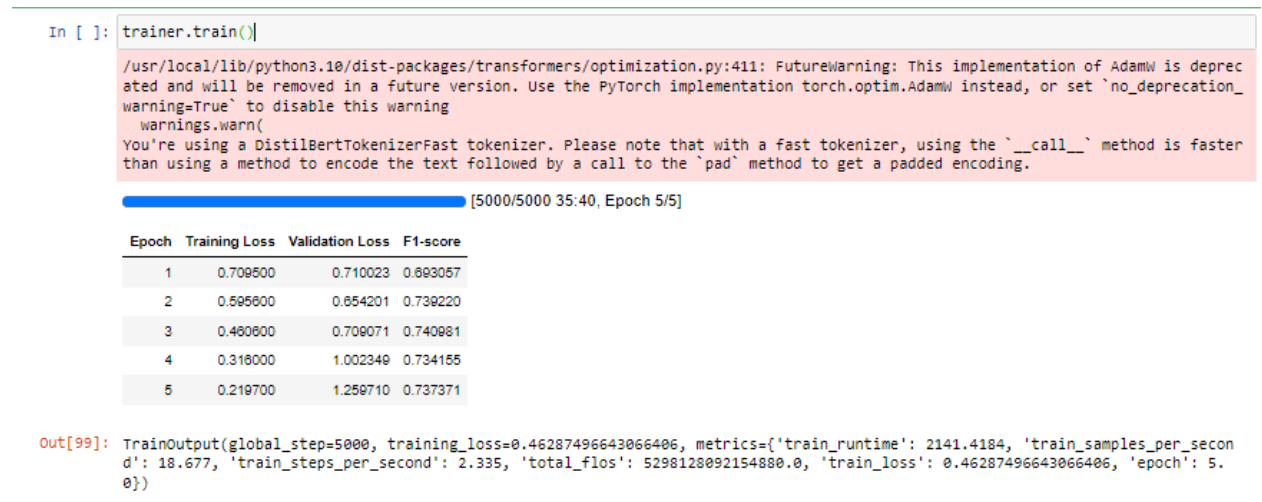


Figure 20



Training hyperparameters

The following hyperparameters were used during training:

- learning_rate: 5e-05
- train_batch_size: 8
- eval_batch_size: 8
- seed: 42
- optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- lr_scheduler_type: linear
- num_epochs: 5

Training results

Training Loss	Epoch	Step	Validation Loss	F1-score
0.6985	1.0	1000	0.6765	0.7147
0.5511	2.0	2000	0.6362	0.7618
0.3932	3.0	3000	0.7714	0.7556
0.2184	4.0	4000	1.2567	0.7530
0.1451	5.0	5000	1.4088	0.7474

Figure 21

Training hyperparameters

The following hyperparameters were used during training:

- learning_rate: 5e-05
- train_batch_size: 8
- eval_batch_size: 8
- seed: 42
- optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- lr_scheduler_type: linear
- num_epochs: 5

Training results


Training Loss	Epoch	Step	Validation Loss	F1-score
0.7293	1.0	1000	0.7054	0.6857
0.6175	2.0	2000	0.6039	0.7454
0.5132	3.0	3000	0.6426	0.7662
0.4113	4.0	4000	0.7244	0.7790
0.3092	5.0	5000	0.9855	0.7734

Figure 22

Model Evaluation

After fine-tuning the models, we evaluated their performance on the evaluation set. We computed evaluation metrics such as loss and F1-score, which measures the model's accuracy in classifying negative, neutral, and positive sentiments. The results showed that all three models achieved high F1-scores, indicating good performance on the sentiment analysis task.

```
: # Evaluate the model
results = trainer.evaluate()
print("DistilBERT model:")
print("Evaluation results:", results)
```



```
DistilBERT model:
Evaluation results: {'eval_loss': 0.6542006731033325, 'eval_f1-score': 0.7392202958287318, 'eval_runtime': 32.917, 'eval_sample
s_per_second': 60.759, 'eval_steps_per_second': 7.595, 'epoch': 5.0}
```


Figure 23

Streamlit Web App Deployment

Finally, we developed a user-friendly web application using Streamlit to interact with the fine-tuned models. The app allows users to input a tweet or choose from example tweets related to Covid vaccines. Users can also select one of the three fine-tuned models (BERT, DistilBERT, or RoBERTa) for sentiment analysis.

The app utilizes the Hugging Face Transformers library and the Streamlit framework to process the input tweet through the selected model. The model predicts the sentiment (negative, neutral, or positive) and provides the confidence level of the prediction. The results are displayed with corresponding emoji animations for a delightful user experience.

Figures 24 & 25 show the streamlit App in huggingface giving a provision of a dropdown button to select the model to run at any given time.



Covid Vaccine Tweet Sentiments

These models were trained to detect how a user feels about the covid vaccines based on their tweets(text)

Copy and paste a tweet or type one

I find it quite amusing how people ignore the effects of not taking the vaccine

Can't Type? Select an Example below

I hate the vaccines

Which model would you want to Use?

Bert

Predict


Sentiment Emoji	How this user feels about the vaccine	Confidence of this prediction
	POSITIVE	60.36%

Figure 24



Covid Vaccine Tweet Sentiments

These models were trained to detect how a user feels about the covid vaccines based on their tweets(text)

Copy and paste a tweet or type one

I find it quite amusing how people ignore the effects of not taking the vaccine

Can't Type? Select an Example below

I hate the vaccines

Which model would you want to Use?

Distilbert

Predict

Sentiment Emoji



How this user feels about the vaccine

POSITIVE

Confidence of this prediction

68.61%

Figure 25

Conclusion

In this article, we demonstrated the end-to-end process of fine-tuning pre-trained transformer models for sentiment analysis on tweets related to Covid vaccines. We used the Hugging Face Transformers library for model training and evaluation, and the Streamlit framework for web app development and deployment.

Sentiment analysis is a powerful tool for understanding public sentiment towards specific topics. By applying these techniques to Covid vaccine-related tweets, we can gain insights into public attitudes and emotions surrounding vaccination efforts. The web app allows users to interactively explore and analyze sentiments in real-time, making it a valuable tool for various stakeholders, including researchers, policymakers, and the general public.

As new data and models become available, researchers can further refine and improve the sentiment analysis process to provide more accurate insights into public perceptions and attitudes. By harnessing the power of NLP and web app development, we can continue to advance our understanding of public sentiment and drive positive change in response to public health challenges.

Thanks For Reading

Thanks for reading. Send me your thoughts and ideas. I look forward to chatting with you soon.

Email: jaroyajo@gmail.com

LinkedIn: <https://www.linkedin.com/in/jjaroya/>

GitHub: https://github.com/Jauloma/Career_Accelerator_P5-NLP