ML4H Project 2

May 16, 2023

Aloïs Thomas
alothomas@student.ethz.ch

Jaro Meyer
jarmeyer@student.ethz.ch

Disclaimer: Various questions and code snippets contain parts generated using ChatGPT

Part 1: Data exploration and pre-processing

Q1: Preprocessing

First we removed all rows where `TweetText` or `Sentiment` were `NaN`. We also made sure that there are no duplicated tweets in our dataset using the unique `TweetID`. We created a new `Sentiment_class` column that assigns a positive, neutral or negative sentiment label to every Tweet:

```
df = df.drop_duplicates()  # Remove duplicates
df = df.dropna()  # Remove rows with missing values

#Sentiment processing
df[['Positive Sentiment', 'Negative Sentiment']] = df['Sentiment'].str.split(' ',
expand=True)
df['Sentiment']   =   df['Positive   Sentiment'].astype(float)   +   df['Negative
Sentiment'].astype(float)
df = df.drop(df[df['Sentiment'] == 0].index)
df['Sentiment_class'] = df['Sentiment'] > 0
```

The next step was to preprocess the actual Tweet texts. They are converted to lowercase and all URLs, mentions, hashtags, and special characters are removed. The texts are then tokenized and after removing all stop words the tokens are lemmatized before getting concatenated to a string again. We decided not to apply spelling correction since Tweets often contain abbreviations and slang words which would be corrected incorrectly.

Following function was used to preprocess the text:

```
def preprocess_text(text):
    stop_words = stopwords.words('english')
    lemmatizer = WordNetLemmatizer()
    text = text.lower() # Convert all text to lowercase
    text = re.sub(r'http\S+', '', text) # Remove URLs
    text = re.sub(r'@[A-Za-z0-9]+', '', text) # Remove mentions
    text = re.sub(r'#[A-Za-z0-9]+', '', text) # Remove hashtags
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # Remove punctuation and special chars
    tokens = word_tokenize(text)
    filtered_tokens = [token for token in tokens if token not in stop_words] # Remove
stop words
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens] #
Lemmatize tokens
    preprocessed_text = ' '.join(lemmatized_tokens)
    return preprocessed_text
```

Q2: Exploratory data analysis

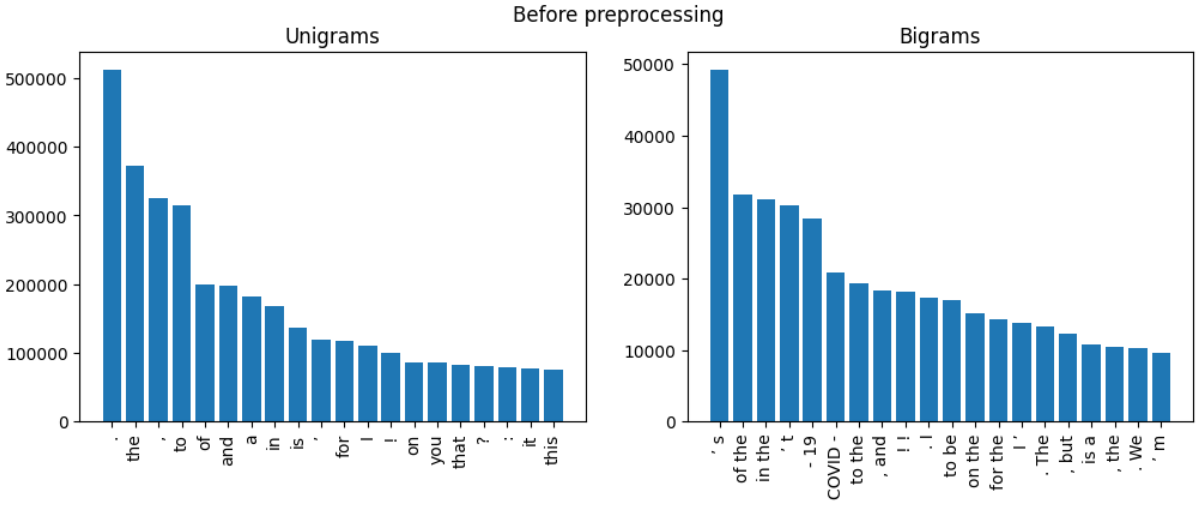

Figure 1: 20 most used Uni- and Bi-grams before preprocessing

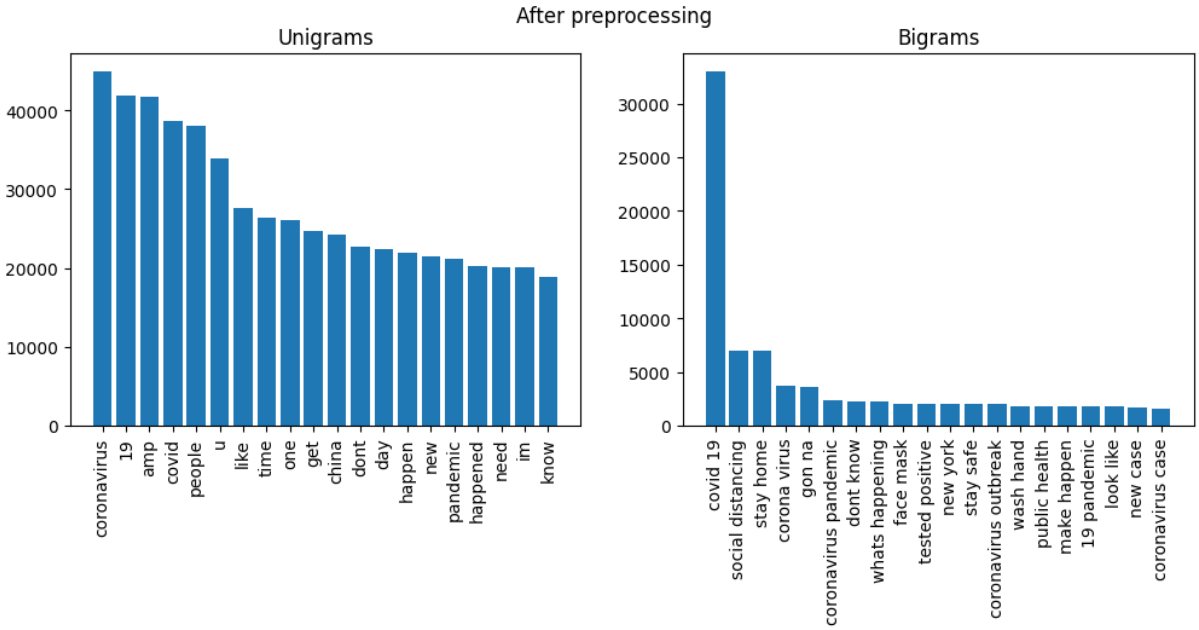

Figure 2: 20 most used Uni- and Bi-grams after preprocessing

Q3: Metric choice

Since we are dealing with a classification task we chose to use F1-score to measure the performance of different models/techniques. F1 is a good metric to use when one deals with balancing the trade-off between precision and recall and the cost of false positives and false negatives is similar for all classes, which is the case here.

In term of class balancing, as seen in Figure 3, there is a majority of neutral sentiment but the imbalance is not drastic. Thus, class imbalance should not impact the model performances too much.

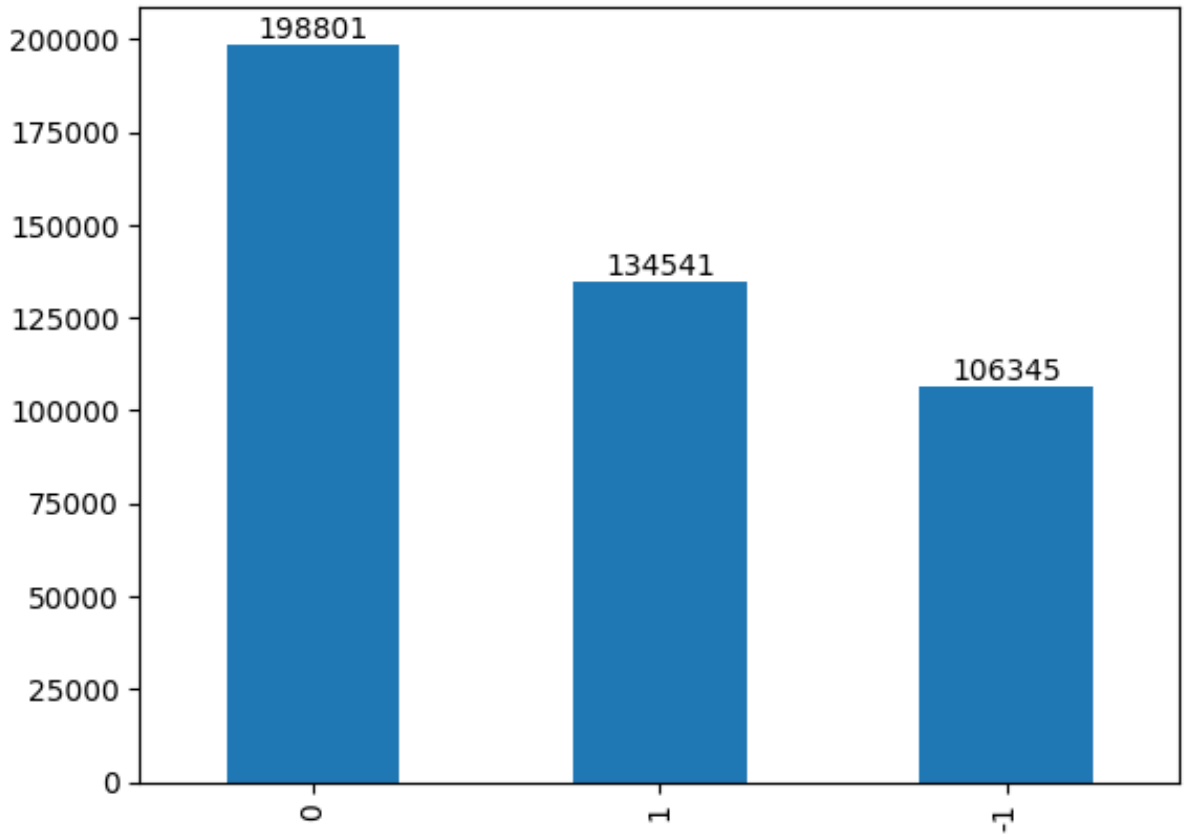

Figure 3: Overview of class balance with the computed sentiment labels

Q4: Dataset splitting

The original dataset was split into 64% training, 20% test, and 16% validation datasets. These splits were done randomly to avoid label shift over time.

```
X = df['TweetText']
y = df['Sentiment_class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2,
random_state=7)
```

Part 2: NLP learning based methods

Vader

Q1: Briefly explaining how this method works

Vader combines a lexicon approach using pre-existing dictionary of words and phrases that have been manually labeled with their sentiment scores with additional rules that take into account context and linguistic to further refine the sentiment scores. This allows it to work on text with informal language, misspellings, use of slang and emojis, and other social media specific characteristics.

Q2: Provide a code snippet detailing how to use it for our task

Since Vader was specificaly developed to work on unstructured social media text with an understanding of puctuation (e.g. multiple !'s provide extra emphasis), use of capitalization (e.g. ALL CAPS), and emoticons like :) and :D, making everyting lowercase and the removing of special characters is not needed and probably even hurting the performance. Removing stopwords is also not required since Vader does this by default. Stemming and lemmatization are also not strictly required since Vader was built to identify sentiment based on pretrained words/phrases, regardless of their morphology. Stemming and lemmatization could sometimes even change the meaning of words, and thus negatively impact the performance. Note that we used only a sample size of 10,000 tweets only as we will use this data size for the next methods for computational reasons.

The following code was used to apply Vader to the dataset:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

df = pd.concat([X_test, y_test], axis=1)

# Applying Vader to pre-processed dataset and using the Sentiment_class label created
in Part1 as a ground truth
analyzer = SentimentIntensityAnalyzer()
df['scores'] = df['TweetText'].apply(lambda tweet: analyzer.polarity_scores(tweet))
df['compound'] = df['scores'].apply(lambda scores: scores['compound'])

df['vader_sentiment'] = df['compound'].apply(lambda score: -1 if score <= -0.05 else
(1 if score >= 0.05 else 0))

accuracy = (df['vader_sentiment'] == df['Sentiment_class']).mean()
f1 = f1_score(df['Sentiment_class'], df['vader_sentiment'])
print(f"Accuracy: {accuracy:.2%}")
print(f"F1-score: {f1:.2%}")
```

Q3: Apply this method to our TweetsCOV19 dataset and comment on the performance obtained

By applying Vader to our pre-processed dataset (Test set), we get an accuracy of 62.60% and an F1-score of 0.62. Those performance metrics could probably be higher if we had used the dataset unprocessed as mentioned in Q2. However, those results could allow for further use of Vader for analysis of sentiment or to confirm the feasability of Tweet sentiment analysis on the data before trying more ressource intensive methods such as word embedding.

Word Embeddings

Q1: Bag of Words (BoW)

BoW converts text into a numerical feature vector by assigning each word from the training text a unique ID and then counting the frequency of previously seen words during embedding of new text,

ignoring the order of the words and their grammatical relationships. These word coutns are then returned as a (sparse) vector with dimension = #(distinct words in the training dataset). The variable `bag_vectors` contains the embedded vectors for all Tweets from the test dataset.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
bag_vectors_train  = vectorizer.fit_transform(X_train['TweetText'])
bag_vectors_test = vectorizer.transform(X_test['TweetText'])
```

Q2: TF-IDF

TF-IDF embedding is a method that assigns numerical vectors to words in a tweet based on their frequency and importance. It consists of two components: term frequency (TF) and inverse document frequency (IDF). TF counts how many times a word occurs in a tweet, while IDF weighs how common or rare a word is in a collection of tweets. The product of TF and IDF gives the TF-IDF value for each word in each tweet. A high TF-IDF value indicates that the word is frequent in the tweet but rare in the collection of tweets, implying more importance and relevance for the specific tweet. A low value implies less importance and informativeness for the specific tweet.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
tfidf_vectors_train = tfidf.fit_transform(X_train["TweetText"])
tfidf_vectors_test = tfidf.transform(X_test["TweetText"])
```

Q3: Word2Vec with CBOW or Skip-gram

Word2Vec is a method that learns word embeddings from a text. It uses a neural network to learn the semantic and syntactic relationships between words based on their co-occurrence patterns. For the COVID19 Twitter dataset, we chose to use Skip-Gram as it can learn more complex and nuanced meanings of words relevant to the context of the dataset. Moreover, the dataset may have rare and specific words related to Covid19 that Skip-Gram can better embed by predicting their context in a tweet. We use a 300-dimensional model created by Google. We also use zero vectors as a default for words that are not in the model vocabulary.

```
import gensim.downloader as api
model_vec = api.load('word2vec-google-news-300')

def get_word_vectors_vec(df):
    word_vectors = []
    for tweet in df:
        tweet_vectors = []
        for word in tweet.split():
            try:
                tweet_vectors.append((word, model_vec[word]))
            except KeyError:
                default_vector = (word, np.zeros(300))
                tweet_vectors.append(default_vector)
                continue
        word_vectors.append(tweet_vectors)
    return word_vectors
```

Q4: GloVe

GloVe is a method that learns word embeddings from global word-word co-occurrence statistics in a large corpus. It is an unsupervised learning algorithm developed by Stanford. GloVe has several benefits compared to other word embedding methods, such as Word2Vec or FastText. It can capture both

global and local information from the corpus, it can handle large vocabularies and corpora efficiently, and it can generate word vectors for unseen words in the training corpus. We use a 25-dimensional model tailored for Twitter. We also use zero vectors as a default for words that are not in the model vocabulary.

```
import gensim.downloader as api
model_glove = api.load('glove-twitter-25')

def get_word_vectors_glove(df):
    word_vectors = []
    for tweet in df:
        tweet_vectors = []
        for word in tweet.split():
            try:
                tweet_vectors.append((word, model_glove[word]))
            except KeyError:
                default_vector = (word, np.zeros(25))
                tweet_vectors.append(default_vector)
                continue
        word_vectors.append(tweet_vectors)
    return word_vectors
```

Q5: FastText

FastText is a method that decomposes words into subword n-grams and represents them as a sum of their n-gram vectors. This enables FastText to handle the word morphology and the rare or unknown words.

FastText uses a shallow neural network with one hidden layer and a softmax output layer to learn word embeddings. It can adopt either the skip-gram or the CBOW architecture. The skip-gram model learns to predict the surrounding words given a center word, while the CBOW model learns to predict the center word given the surrounding words.

FastText has several benefits compared to other word embedding and text classification methods, such as Word2Vec or GloVe. It can scale well to large vocabularies and corpora, it can create word vectors for new words using n-gram information, it can train word embeddings and text classifiers together or separately, and it can achieve state-of-the-art performance on various tasks. We used a 300-dimensional model trained on Wikipedia. We also used a zero vector as a default for words that were not in the model vocabulary, possibly due to suboptimal data preprocessing.

```
import gensim.downloader as api
ft_model = api.load('fasttext-wiki-news-subwords-300')

def get_word_vectors_ft(df):
    word_vectors = []
    for tweet in df:
        tweet_vectors = []
        for word in tweet.split():
            try:
                tweet_vectors.append((word, ft_model[word]))
            except KeyError:
                default_vector = (word, np.zeros(300))
                tweet_vectors.append(default_vector)
                continue
        word_vectors.append(tweet_vectors)
    return word_vectors
```

Q6: Visualization of embeddings

The plots show the visualization of the latent space of each embedding approach using t-SNE, a dimensionality reduction technique that preserves local distances, PCA and UMAP. The plots are colored by the sentiment labels (negative, neutral or positive) in a selection of 300 word vectors of tweets.

The plots reveal some differences in the quality of the semantic content of each embedding approach. For example, bag of words and TF-IDF have a more scattered and overlapping distribution of the sentiment labels, indicating that they are not very good at separating the positive and negative tweets more particularly on the PCA projection. Word2Vec, FastText, and GloVe have a more clustered and distinct distribution of the sentiment labels, indicating that they are better at capturing the polarity of the tweets. However, those clusters are not very easely distinguishable from each other meaning that those methods are not extremely efficient at separating the words of the tweets by their sentiment.

Overall these differences in the visualization of each method suggest that word embeddings are more suitable for downstream tasks such as sentiment analysis than bag of words or TF-IDF. Word embeddings can encode more semantic information and distinguish between positive, neutral and negative tweets better than bag of words or TF-IDF. Therefore, we can anticipate that word embeddings will have higher performance on sentiment analysis than bag of words or TF-IDF.

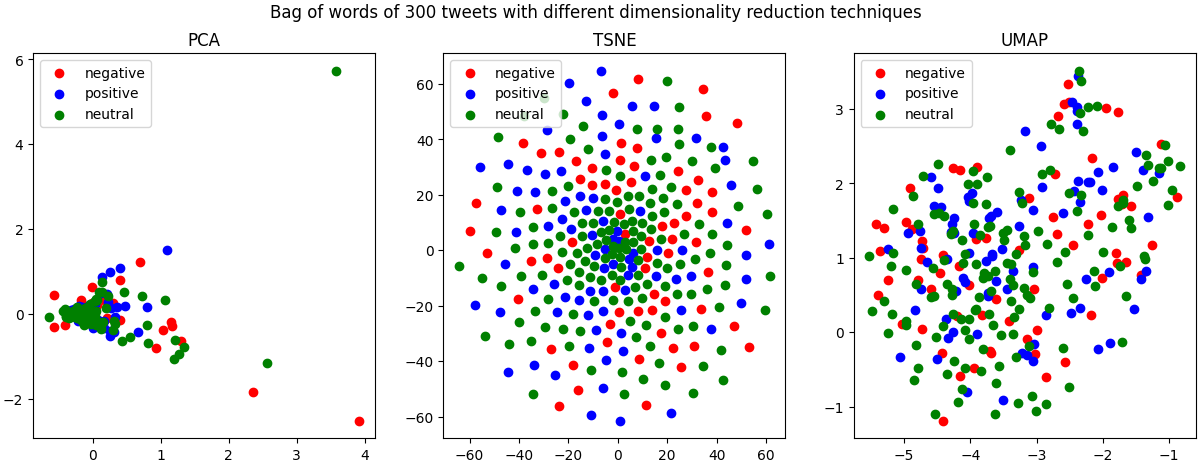

Figure 4: Bag of words embedding visualization

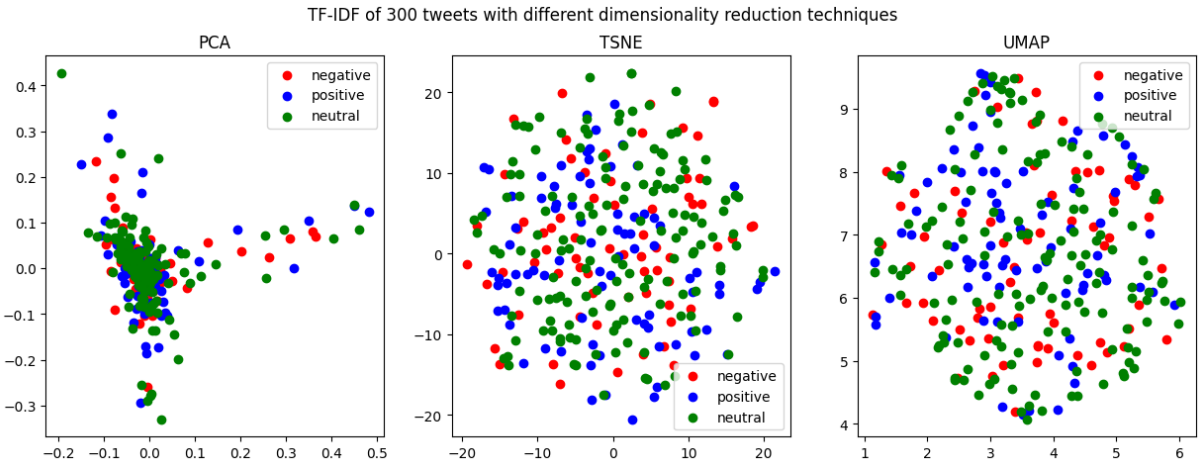

Figure 5: TF-IDF embedding visualization

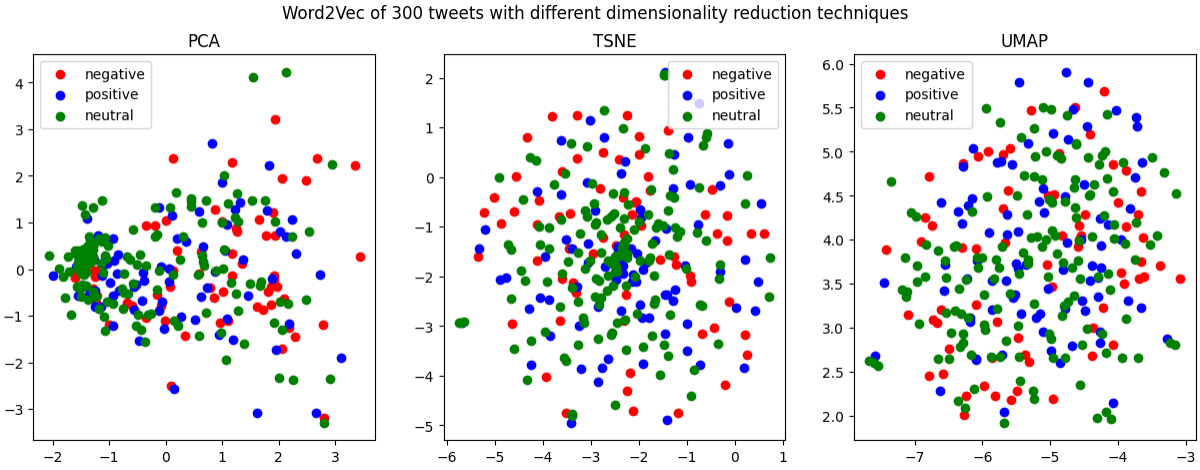

Figure 6: Word2Vec embedding visualization

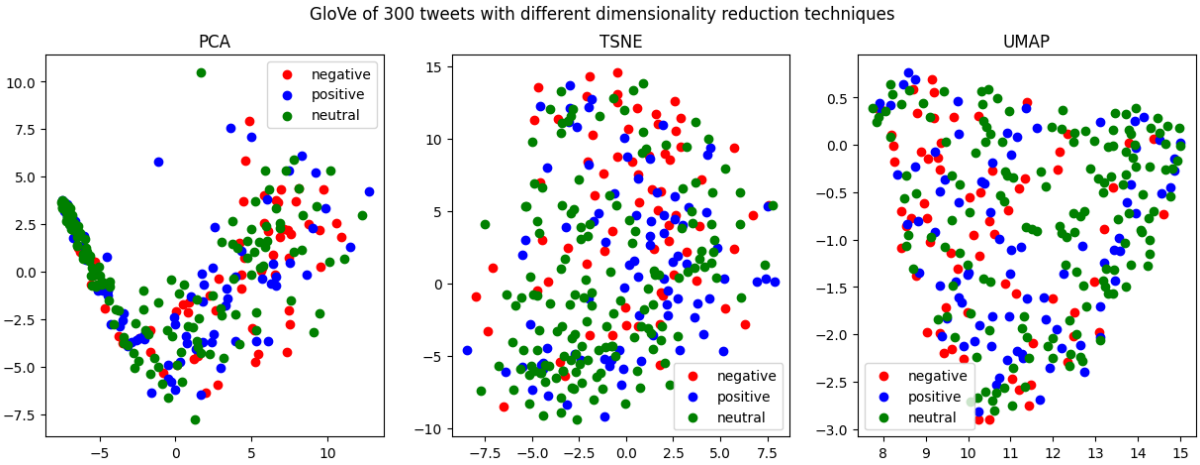

Figure 7: GloVe embedding visualization

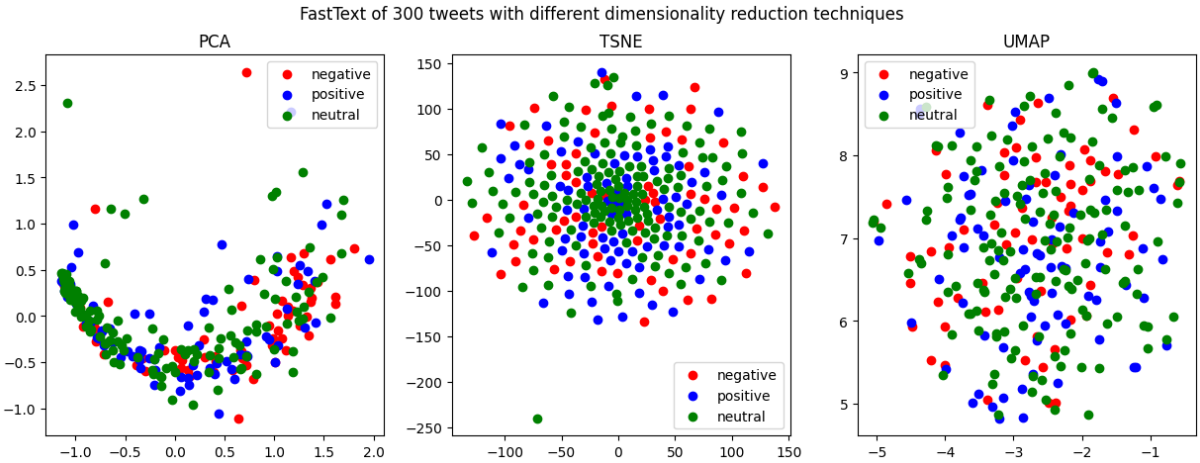

Figure 8: FastText embedding visualization

Q7: Tweet embeddings

As a way to do Tweet embedding, we used three different approaches: average embedding, TF weighting and SIF embedding.

Average embedding is a method that computes the tweet embedding as the mean of the word embeddings of the words in the tweet. This method is simple and fast, but it loses the word order and context information and does not account for the different importance of words.

```
def average_embedding(word_vectors):
  tweet_vector = np.mean([v for w, v in word_vectors], axis=0)
  return tweet_vector
```

TF weighting is a method that improves average embedding by multiplying each word embedding by its term frequency (TF) in the tweet. This method can capture the relevance of words by giving higher weights to more frequent words, but it still ignores the word order and context information and does not account for the global information of words.

```
def tf_weighting_embedding(word_vectors, freqs):
  tweet_vector = np.zeros(word_vectors[0][1].shape)
  for word, word_vector in word_vectors:
    tweet_vector += word_vector * freqs[word]
  return tweet_vector
```

SIF embedding is a method that further improves TF weighting by multiplying each word embedding by its inverse document frequency (IDF) in the corpus and applying principal component analysis (PCA) to remove some common components. This method can capture both local and global information of words by giving higher weights to more rare words and reducing noise and redundancy, but it still ignores the word order and context information and requires more computation and data.

```
def sif_embedding(word_vectors, freqs, a=1e-3):
  words = [w for w, v in word_vectors]
  vectors = np.array([v for w, v in word_vectors])
  weights = np.array([a / (a + freqs[w]) for w in words])
  weighted_average = np.sum(vectors * weights[:, np.newaxis], axis=0)
  weighted_average /= np.sum(weights)
  pca = PCA(n_components=1)
  pca.fit(vectors)
  u = pca.components_[0]
  sif_vector = weighted_average - u.dot(weighted_average) * u
  return sif_vector
```

Overall, we did not have to deal with out of vocabulary words with tweet embeddings because we dealt with this problem during the word embedding part with a default vector for missing words.

Q8: Classifier

We experimented with three classification models: Random Forest, Logistic Regression and SVM on all the aggregated datasets obtained from the different embedding and aggregation methods. For computational purposes, we used only a random selection of 10,000 tweets of the pre-processed dataset. The target classes were negative sentiment (-1), neutral sentiment (0) and positive sentiment (1) calculated from the original sentiment labels of the dataset. We used GridSearch to tune the hyperparameters of each model. For Random Forest, we adjusted the number of trees in the forest (n_estimators), the maximum depth of the tree (max_depth) and the splitting criterion (criterion). For Logistic Regression, we varied the inverse of regularization strength, the regularization type (penalty) and the optimization algorithm (solver). For SVM, we changed the regularization parameter, the kernel type (kernel), the kernel coefficient (gamma) and the degree of the polynomial kernel (degree).

We evaluated all the models on the test set using the F1-score metric. The results can be seen in Table 1.

	Random forest			Logistic regression			SVM		
	Average	**Weights**	**SIF**	**Average**	**Weights**	**SIF**	**Average**	**Weights**	**SIF**
Word2Vec	0.49	0.45	0.49	0.59	0.37	0.57	0.56	0.53	0.55
FastText	0.50	0.43	0.50	0.60	0.31	0.55	0.57	0.51	0.53
GloVe	0.49	0.42	0.41	0.46	0.39	0.38	0.54	0.42	0.39
Bow	0.70			0.72			0.53		
TF-IDF	0.67			0.72			0.57		

Table 1: Performance comparison [F1-score]

Q9: Performance comparison

The table shows that the best performing method is Bag of Words and TF-IDF with logistic regression, which achieve an F1-score of 0.72. The worst performing method is Word2Vec and FastText with TF weighting aggregation and logistic regression, which achieve an F1-score of 0.21. The other methods have varying performance depending on the embedding model, the aggregation method, and the classifier. Overall, bag of words and TF-IDF perform better than the pre-trained models probably because we did not train the models on our dataset which leads to a large decrease in performance for the pre-trained models.

From a computational point of view, word embeddings are more expensive than bag of words (BOW) and TF-IDF, as they require more memory and time to train and use. However, they can capture more semantic information in theory than BOW and TF-IDF, which rely on simple counts or frequencies of words. Word embeddings can also handle different types of tweets, such as informal language, slang, abbreviations and emojis.

One can notice that FastText seems to perform better than Word2Vec and GloVe overall. This can be explained by the design of FastText itself which can handle out-of-vocabulary words in theory where the two other methods do not.

If we had to select one approach among all, we would select Bag of words with logistic regression for this task. It has a good balance of performance and complexity, as they achieve an F1-score of 0.72. It also use a simple classifier that is easy to interpret and robust to noise. Furthermore by training the model on the entire dataset and not just a small sample of 10,000, we could improve the performance drastically.

Further potential extensions that could help improve the performance of the methods are possible. For example, using pre-trained embeddings on tweets or other domains could improve the quality and relevance of the embeddings. Further training the pre-trained models on the dataset and using more advanced aggregation methods such as attention or pooling could capture more information from the embeddings. Using more sophisticated classifiers such as neural networks or boosting could learn more complex patterns from the features.

Transformers

Q1: Transformer-based language models

Transformer LLMs can learn from large-scale, unlabeled text data through pre-training and can then be fine-tuned for specific tasks. They models consist of an encoder and a decoder. The encoder processes the input sequence and generates contextualized representations for each token in the sequence. The decoder takes the encoder's output and generates an output sequence, such as a translated sentence or a text completion. This process is guided by a self-attention mechanism which allows model to weigh the importance of different tokens in the input sequence when generating a representation for each

token. Self-attention helps the model to capture long-range dependencies instead of only relying on a small, local context window. To capture different types of dependencies and relationships, transformers can use multiple attention heads in parallel, each attending to different linear projections of the input sequence. Stacking multiple layers of self-attention and feed-forward networks together allows the model to capture increasingly complex patterns and dependencies as the information flows through the network.

Since transformer models don't have an inherent notion of word order, positional encodings manually have to be added to the input embeddings.

The main differences between BERT and RoBERTa are the objective and method of masking.

BERT was pre-trained on two different objectives, namely Masked Language Modeling (MLM) where a certain percentage of input words are randomly masked, and the model is trained to predict the masked words based on the surrounding context and Next Sentence Prediction (NSP) where the model is trained to predict whether two sentences appear consecutively in a document or not. The idea behind this choice is that by combining these two objectives, BERT learns contextualized word representations that capture both local and global dependencies within and between sentences. The MLM objective provides a bidirectional context understanding, while the NSP objective helps BERT learn relationships between sentences. This enables the model to perform well on various downstream NLP tasks, such as question answering, text classification, and named entity recognition. RoBERTa removes the NSP objective allowing RoBERTa to focus more on learning the relationships between words within sentences and avoiding potential biases introduced by the NSP objective. Additionally, RoBERTa is trained on a substantially larger corpus of unlabeled text data. RoBERTa also uses larger batch sizes during pre-training, which improves computational efficiency.

Model size plays a significant role in performance. Larger models with more layers and hidden units can capture more complex patterns and dependencies in the data but also require larger datasets for training. The number of attention heads and the attention dropout rate can als affect performance. Different training objectives can also affect the performance depending on the downstream task.

RoBERTa also introduces dynamic masking, where different mask patterns are dynamically created for each training epoch instead of pre-generateing maskings before training. This can lead to more robust representations being learnt.

GPT based models again differ from BERT/RoBERTa in multiple aspects. Most importantly they are trained using autoregressive language modeling, where the model always predicts the next word in a sequence given the preceding context, compared to MLM where the masked word can appear anywhere in the sentence with context extending on both sides. GPT models thus are unidirectional while both BERT and RoBERTa are bidirectional. The left-to-right direction makes GPT very good at generating coherent and contextually relevant text from scratch while bidirectionality helps BERT/RoBERTa in building a deeper understanding of the contextual relationships between words.

Additionaly GPT models have a fixed context window length, limiting the range of dependencies they can capture. BERT/RoBERTa, on the other hand, do not have such limitations and can take into account an enitre document of arbitrary length.

Q2: Scalability

Embedding-based approaches, such as word2vec and GloVe, are typically more scalable compared to transformer-based language models in terms of the number of parameters. Embedding-based models generate fixed-dimensional representations for each word or subword unit based on co-occurrence statistics or contextual information. These representations are independent of the input sequence

length, making them computationally efficient and scalable. Embedding-based work well with relatively smaller datasets.

On the other hand, transformer-based language models have a significantly larger number of parameters due to the self-attention mechanism and the deeper architectures. Models like BERT, RoBERTa, and GPT can have millions or even billions of parameters, enabling them to capture more intricate dependencies and nuances in the language. However, the increased number of parameters comes with increased computational requirements during training and inference, making transformer models more resource-intensive and requiring large-scale datasets to properly capture extensive language knowledge.

Q3: Code

For our transformer we chose to use a pretrained RoBERTa-base model from the Cardiff NLP group at Cardiff University [1]. This model has been trained on 124M tweets from January 2018 to December 2021, and finetuned for sentiment analysis with the TweetEval benchmark. This aligns well with the task we are trying to accomplish which is sentiment analysis on tweets. To begin with we used the model as is without any finetuning and applied it to our test dataset using following code snippet:

```
import tensorflow as tf
from transformers import AutoModelForSequenceClassification
from transformers import TFAutoModelForSequenceClassification
from transformers import AutoTokenizer, AutoConfig
from sklearn.metrics import f1_score

task='sentiment'
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment-latest"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
config = AutoConfig.from_pretrained(MODEL)
model = TFAutoModelForSequenceClassification.from_pretrained(MODEL)

X = tokenizer(X_test.to_list(), return_tensors='tf', padding=True)

dataset = tf.data.Dataset.from_tensor_slices((dict(X), y_test)).batch(batch_size=64)
output = model.predict(dataset)


scores = tf.nn.softmax(output.logits, axis=-1)
neg = scores[:,0]
pos = scores[:,2]
pred = pos > neg
labels = y_test.to_list()
f1_score(labels, pred)
```

Next we modified our code to allow fine-tuning of the last k layers on our training dataset:

```
import tensorflow as tf
from transformers import AutoModelForSequenceClassification
from transformers import TFAutoModelForSequenceClassification
from transformers import AutoTokenizer, AutoConfig
from sklearn.metrics import f1_score
k_last_layers = 2
task='sentiment'
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment-latest"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
config = AutoConfig.from_pretrained(MODEL)
```

```
train_encoded = tokenizer(X_train.to_list(), return_tensors='tf', padding=True)
train_ds          =          tf.data.Dataset.from_tensor_slices((dict(train_encoded),
y_train)).batch(batch_size=8)
train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)

val_encoded = tokenizer(X_val.to_list(), return_tensors='tf', padding=True)
val_ds            =            tf.data.Dataset.from_tensor_slices((dict(val_encoded),
y_val)).batch(batch_size=8)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)

test_encoded = tokenizer(X_test.to_list(), return_tensors='tf', padding=True)
test_ds           =           tf.data.Dataset.from_tensor_slices((dict(test_encoded),
y_test)).batch(batch_size=8)
test_ds = test_ds.prefetch(buffer_size=AUTOTUNE)

X = tokenizer(X_test.to_list(), return_tensors='tf', padding=True)
dataset = tf.data.Dataset.from_tensor_slices((dict(X), y_test)).batch(batch_size=64)


# training
model = TFAutoModelForSequenceClassification.from_pretrained(MODEL, num_labels=2)
# Freeze all layers except for the last k
for layer in model.layers[:-k_last_layers]:
    layer.trainable = False
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5, clipnorm=1.),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.metrics.SparseCategoricalAccuracy()],
    )
model.fit(train_ds, validation_data=val_ds, epochs=2)

# evaluation
output = model.predict(test_ds)
scores = tf.nn.softmax(output.logits, axis=-1)
pred = scores[:,1] > 0.5
labels = y_test.to_list()
f1_score(labels, pred)
```

Q4: Performance analysis

As Table 2 shows, running the RoBERTa model without any training results in a mediocre F1-score of 0.69. Already after fine-tuning only the classifier while freezing the transformer we achieve a significantly better score of 0.79. Even better performance was achieved by finetuning both transformer & classifier.

Method	F1-score
no fine-tuning	0.69
fine-tuning classifier only (1 epoch)	0.79
fine-tuning transformer & classifier (2 epochs)	0.85

Table 2: F1-scores on the test dataset

Possible approaches to improve performance:

- With unlimited resources it would be possible to train the whole model for more epochs instead of only training the last few layers and freezing the rest or training the whole model for only few

epochs. Our dataset should be large enough to increase the number of epochs without any overfitting happening.

- Another idea would be to try different embedding methods other than the one offered by the HuggingFace model. With unlimited resources we could train the model multiple times using different embedding methods and select the one which is achieving the best performance on our task.
- Larger transformer models with more trainable parameters could also be used but we are not entirely sure whether the dataset is large enough to efficiently train such models.

Q5: Transfer learning details

Table 2 shows that fine-tuning both the tranformer and the classifier leads to better downstream performance compared to only training the classification head. This means that the embeddings learnt by the pretrained model are not optimal for our downstream task or that our data is differently distributed than the original dataset that the RoBERTa model was trained on. Unfortunately the time required for training in every experiment didn't allow us to further investigate by freezing different layers of the transformer and comparing the performance.

Q6: Embedding analysis

Figure 9 depicts the 768-dimensional embeddings of our test dataset generated by the stock RoBERTa model without any fine-tuning. They were extracted from the last layer. We again used PCA, t-SNE, and UMAP to reduce the embeddings to 2d for easy visualization. As one can see there are two main clusters with a higher density in all three visualizations, but many samples also land between those clusters or even in the wrong cluster.

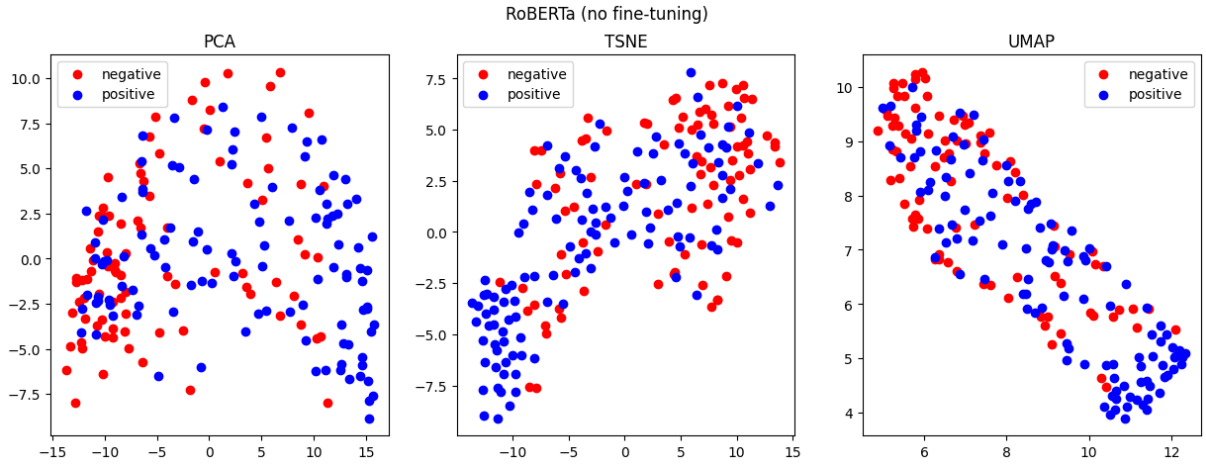

Figure 9: embeddings before fine-tuning

Figure 10 depicts the embeddings of the same samples but this time after the model has been fine-tuned for a single epoch. As one can see there is a lot more separation between the two main clusters, especially in the case of UMAP. Still some samples are in the wrong cluster meaning they will probably be falsly classified.

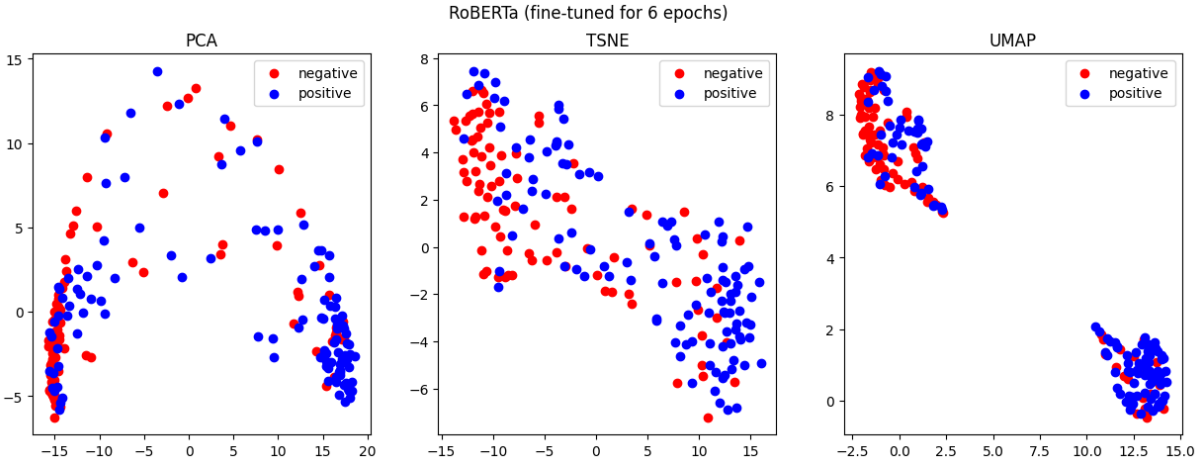

Figure 10: embeddings after fine-tuning (1 epoch)

After training the model for two epoch (Figure 11) the classes are nearly linearly seperable in the 2d representations. Unfortunately we were not able to get the UMAP library running in a Colab TPU instance.

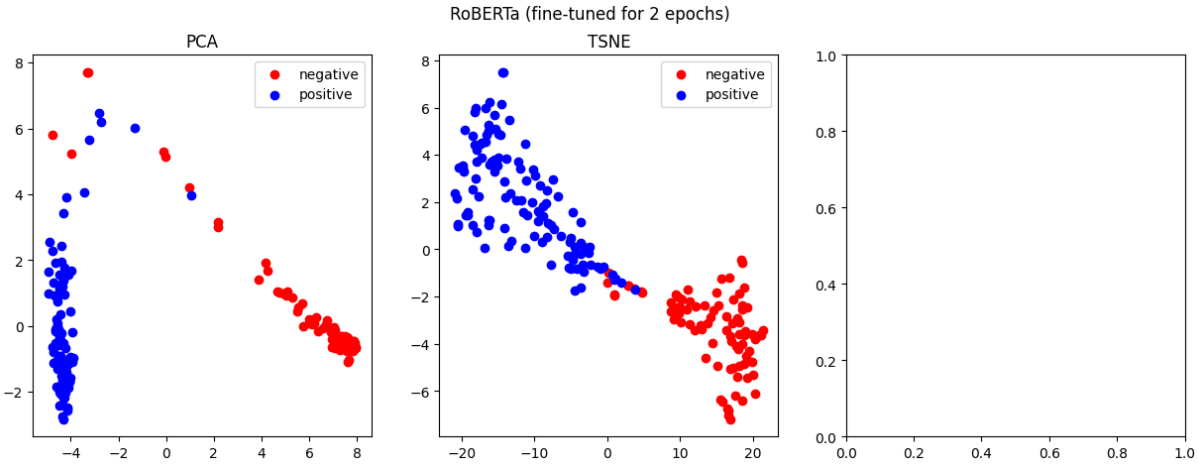

Figure 11: embeddings after fine-tuning (2 epochs)

Already after training for a single epoch the 2d visualizations show better separation of the classes compared to the pretrained model. This is also confirmed by the increase in metrics. RoBERTa embeddings win when compared to all results obtained by using traditional word embedding methods. This explains the better performance (F1-score of 0.79 compared to 0.72 for BoW + Logistic regression). Training the model for another epoch drastically improves separation.

In conclusion the class separation in our 2d representations of the embeddings clearly correlate with the F1-score in our downstream task.

Part 3: Downstream Global Health Analysis

Q1: Research question

The research question of our study is to examine the temporal and geographical variations of COVID-19 perception in Europe. We draw inspiration from the paper "Early warnings of COVID-19 outbreaks across Europe from social media" by Lopreite, M., Panzarasa, P., Puliga, M. et al, which used Twitter data to measure the public sentiment toward pneumonia [2]. In our analysis, we aim to explore how the sentiment toward COVID-19 changed across different regions in Europe over time, and to relate it to the events that occurred during those periods, such as restrictions, health announcements, and political events. This could provide useful insights for better managing pandemics at the country level, and for addressing the health and well-being of the population.

Q2: Method choice and design

The aim of the study is to investigate how the perception of COVID-19 has changed over time geographically in Europe using Twitter data. To achieve this goal, we use the pre-processed dataset of COVID-19 tweets from Part 1, which contains tweets collected from October 2019 to December 2020. We then apply the Vader Sentiment analyzer to each tweet to obtain a sentiment score ranging from -1 (negative) to 1 (positive), and classify the tweets into three categories: neutral (score = 0), positive (score > 0), and negative (score < 0). We chose Vader as our sentiment analysis tool because it showed a relatively good performance on the dataset in Part 2 Q2, with an accuracy of 61.88%, and it has a relatively low computational cost compared to other methods such as Bag of Words combined with logistic regression.

We also create a new feature `Coordinates` that provides the geographical coordinates of the user who wrote the tweet based on the `UserLocation` feature. To do this, we use the geolocator function from the Nominatim library, which converts a location name into a tuple of latitude and longitude. We then convert our dataframe into a geodataframe using the geopandas library, and set the coordinate reference system (CRS) to EPSG:4326, which corresponds to the World Geodetic System (WGS84).

Finally, we group and aggregate the geodataframe by quarter and country, and calculate the mean sentiment score for each group. We then join the aggregated geodataframe with a world geodataframe that contains the boundaries of the countries, and plot the mean sentiment score for each country using a color map. We create four plots, one for each quarter, to visualize how the perception of COVID-19 has changed over time geographically in Europe. We also plot the sentiments by country over time in a simple graph.

In terms of performance evaluation, we acknowledge that the accuracy of the Vader Sentiment analyzer is not very high, and that it may have some limitations in capturing the nuances and contexts of natural language. Therefore, we interpret the results with caution and consider other factors that may influence the perception of COVID-19, such as media coverage, government policies, and cultural differences.

Vader Sentiment analysis:

```
analyzer = SentimentIntensityAnalyzer()
df['scores'] = df['TweetText'].apply(lambda tweet: analyzer.polarity_scores(tweet))
df['compound'] = df['scores'].apply(lambda scores: scores['compound'])
df['vader_sentiment'] = df['compound'].apply(lambda score: -1 if score <= -0.05 else
(1 if score >= 0.05 else 0))

f1 = f1_score(df['Sentiment_class'], df['vader_sentiment'], average='macro')
print(f"Accuracy: {accuracy:.2%}")
print(f"F1-score: {f1:.2}")
```

Coordinate search for every tweet:

```
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut, GeocoderUnavailable, GeocoderServiceError

geolocator = Nominatim(user_agent="your_app_name")

def get_coordinates(location):
  try:
    result = geolocator.geocode(location)

    if result:
      return (result.latitude, result.longitude)
    else:
      return None
  except (GeocoderTimedOut, GeocoderUnavailable, GeocoderServiceError) as e:
    # Print the error message and return None
    print(e)
    return None
```

Agreggation of data by quarters and plotting:

```
# Import pandas and numpy
import pandas as pd
import numpy as np
import geopandas as gpd
from shapely.geometry import Point

df["date"] = pd.to_datetime(df["date"], format="%Y-%m-%d")
df["quarter"] = df["date"].dt.to_period("Q")
df["geometry"] = df["Coordinates"].apply(lambda x: Point(x))

gdf = gpd.GeoDataFrame(df, geometry="geometry")
gdf.crs = "EPSG:4326"
world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
merged = gpd.sjoin(gdf, world, how="left", op="within")

agg = merged.groupby(["quarter", "name"])["vader_sentiment"].mean().reset_index()
agg = world.merge(agg, on="name")
quarters = agg["quarter"].unique()

fig, axs = plt.subplots(2, 2, figsize=(40, 40))
axs = axs.flatten()

for i, q in enumerate(quarters):
 agg_q = agg[agg["quarter"] == q]
 agg_q.plot(column="vader_sentiment", cmap="viridis", ax=axs[i], legend=True)
 axs[i].set_title(q)
 # Set the x and y limits of the plot to zoom in on Europe
 axs[i].set_xlim(-25, 40)
 axs[i].set_ylim(30, 75)
 axs[i].set_aspect("equal")

plt.show()

agg2 = merged.groupby(["date", "name"])["vader_sentiment"].mean().reset_index()
agg2 = agg2.pivot(index="date", columns="name", values="vader_sentiment")
```

```
agg2.plot(kind="line", cmap="viridis", figsize=(20, 10), legend=True)
plt.show()
```

Q3: Results & Analysis

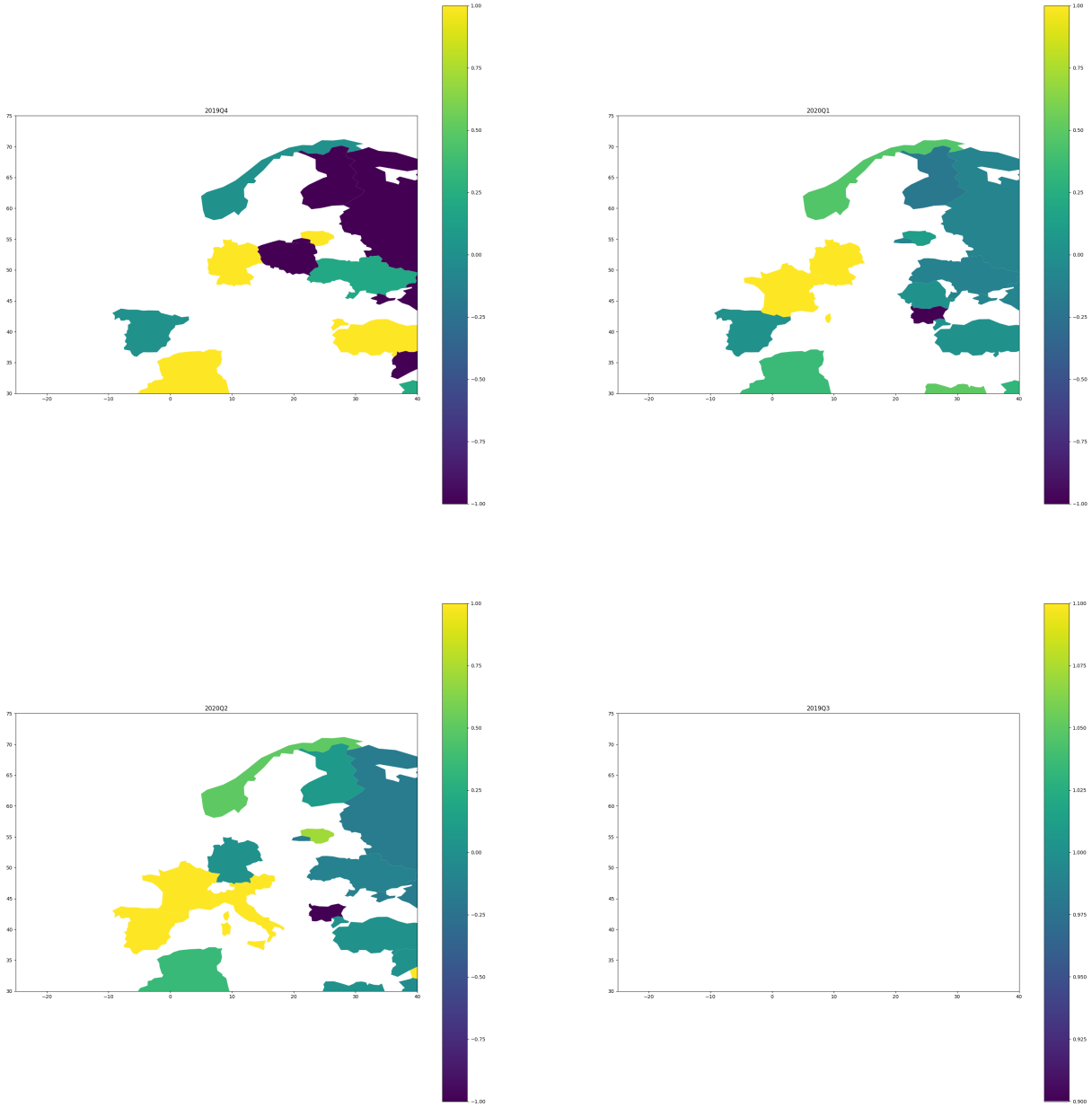

Figure 12: Sentiment evolution for European countries over quarters of time

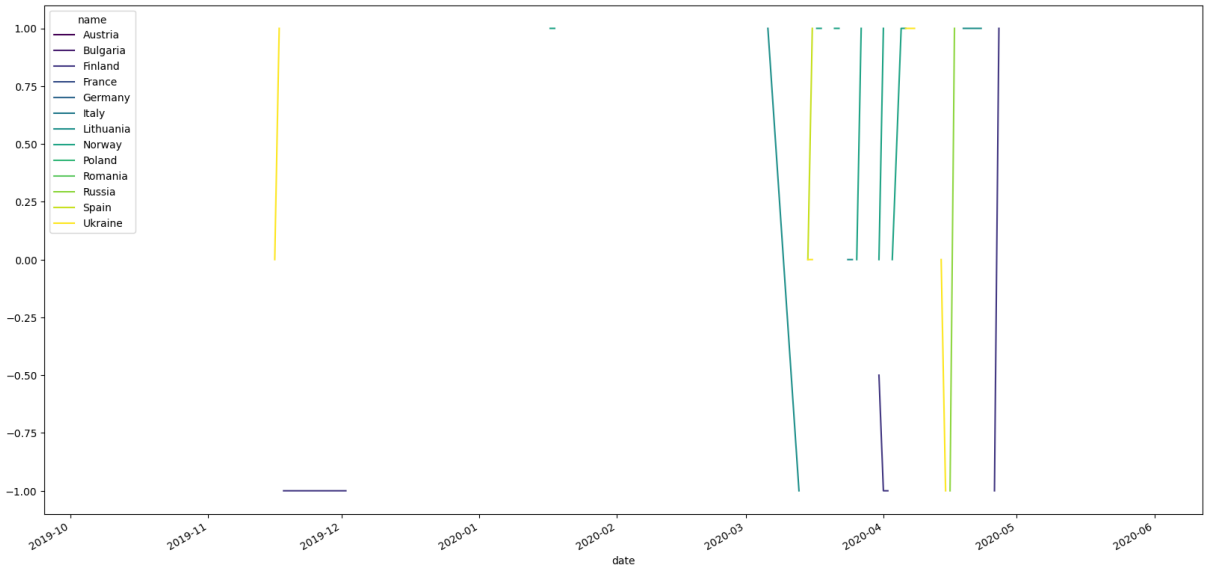

Figure 13: Sentiment evolution over time of european countries

The Figure 12 shows the temporal variation of COVID-19 sentiment in different European countries. We can identify some patterns of sentiment change that may be related to the governmental measures and policies implemented to contain the pandemic. For instance, Germany exhibits a decline in sentiment positivity from 2019-Q4 to 2020-Q2, which coincides with the periods of lockdown and social distancing imposed by the authorities. A similar trend can be observed for Algeria, which also enforced strict restrictions to curb the virus spread.

On the other hand, Norway displays an increase in sentiment positivity over the same time span, despite having a relatively low number of COVID-19 cases and deaths. This may reflect the effectiveness of the country's strategy of testing, tracing and isolating, which allowed for a gradual reopening of society and economy without compromising public health.

However, the Figure 13 reveals some limitations of our data and analysis. Since we only used a sample of 10,000 tweets, we may have missed some relevant information and introduced some biases in our results. Therefore, we cannot draw definitive conclusions or generalizations from our findings.

In summary, our analysis suggests that COVID-19 sentiment varies across countries and time, depending on the context and circumstances of each nation. The governmental responses and interventions may have a significant impact on the public perception and attitude towards the pandemic, as exemplified by the contrasting cases of Germany and Norway.

Q4: Comparison to literature

The paper by Lopreite et al uses Twitter data to uncover early-warning signals of COVID-19 outbreaks across Europe before the official announcements of local sources of infection were made. They show that unexpected levels of concerns about pneumonia were raised across several European countries, especially in the regions that eventually became the hotspots of infections.

Our results are partially consistent with this paper, as we also find that social media can be used to detect temporal variations of COVID-19 sentiment in different countries. However, we use a different keyword ("COVID-19") and a different sentiment analysis method (VADER) than the paper, which may explain some differences in our findings. For example, we find that some countries (such as Germany and Algeria) have a negative sentiment at the beginning of 2019-Q4 and a positive one in 2020-Q2, which is opposite to what the paper reports. This may be due to the fact that we capture the sentiment

towards COVID-19 specifically, while the paper captures the sentiment towards pneumonia in general, which may not be directly related to COVID-19.

Another possible reason for the discrepancy is that we use a smaller sample size (10,000 tweets) than the paper (over 1 million tweets), which may introduce some noise and bias in our results. We acknowledge this limitation and suggest that future work should use larger and more representative datasets to improve the reliability and validity of our analysis.

In conclusion, our results support the idea that social media can provide useful information for monitoring the spread and impact of COVID-19 across countries and time, but they also disagree with some specific findings of the paper by Lopreite et al. We attribute these differences to the different data sources, methods, and sample sizes used in our study and the paper, and we propose some directions for further research to address these issues.

Q5: Discussion

Our approach has some advantages against the one used in the paper:

- Our approach uses a more specific keyword ("COVID-19") than the paper ("pneumonia"), which may capture the sentiment towards the pandemic more accurately and avoid confounding factors such as seasonal flu or other respiratory diseases for our research question.
- Our approach uses a more robust and widely used sentiment analysis method (VADER) than the paper (a custom dictionary-based method), which may account for the nuances and variations of sentiment expression in different languages and contexts.
- Our approach uses a more recent and updated dataset (2019-Q4 to 2020-Q2) than the paper (2019-Q4 only), which may reflect the changes and dynamics of COVID-19 sentiment over time more comprehensively.

However, our approach also comes with some limitations:

- We use a smaller sample size (10,000 tweets) than the paper (over 1 million tweets), which may reduce the representativeness and generalizability of the results and introduce some noise and bias in the analysis.
- We use a single data source (Twitter COVID19 dataset) than the paper (Twitter and Google Trends), which may limit the diversity and coverage of the data and miss some relevant information and signals from other platforms or sources.
- We do not perform a deep geo-localization analysis of the data and trends, unlike the paper, which may prevent from identifying and comparing the regional patterns and variations of COVID-19 sentiment across countries which the paper does very well.

Q6: Summary & Conclusion

In this study, we investigated how the perception of COVID-19 has changed over time geographically in Europe using Twitter data. We applied the VADER sentiment analysis method to 10,000 tweets in seven languages that contained the keyword "COVID-19", and obtained a sentiment score for each tweet ranging from -1 (negative) to 1 (positive). We also extracted the geographical coordinates of the users who wrote the tweets based on their location names, and converted our dataframe into a geodataframe. We then grouped and aggregated the geodataframe by quarter and country, and calculated the mean sentiment score for each group. We joined the aggregated geodataframe with a world geodataframe that contained the boundaries of the countries, and plotted the mean sentiment score for each country using a color map. We created four plots, one for each quarter, to visualize how the perception of COVID-19 has changed over time geographically in Europe. We also plotted the sentiments by country over time in a simple graph.

Our analysis revealed that COVID-19 sentiment varies across countries and time, depending on the context and circumstances of each nation. We found that some countries (such as Germany and Algeria) have a negative sentiment at the beginning of 2019-Q4 and a positive one in 2020-Q2, while others (such as Norway) show the opposite trend. These results may reflect the different governmental responses and interventions to contain the pandemic, as well as the different media coverage, cultural differences, and public awareness of COVID-19.

We acknowledge that our analysis has some limitations, such as the small sample size, the single data source, and the mild accuracy of the sentiment analysis method. Therefore, we interpret our results with caution and suggest that future work should use larger and more representative datasets, multiple data sources, and more robust sentiment analysis methods to improve the reliability and validity of our findings.

Bibliography

[1] C. University, "Twitter-roberta-base for sentiment analysis - updated (2022)." (https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest?)

[2] M. Lopreite, P. Panzarasa, M. Puliga, and M. Riccaboni, "Early warnings of covid-19 outbreaks across europe from social media," *Scientific Reports*, vol. 11, no. 1, p. 1, 2021.