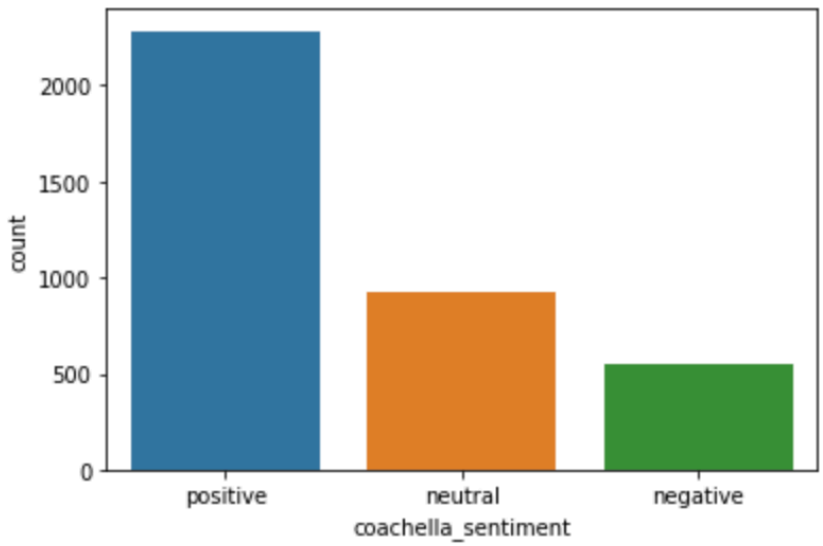
Twitter Sentiment Analysis on Music Festivals

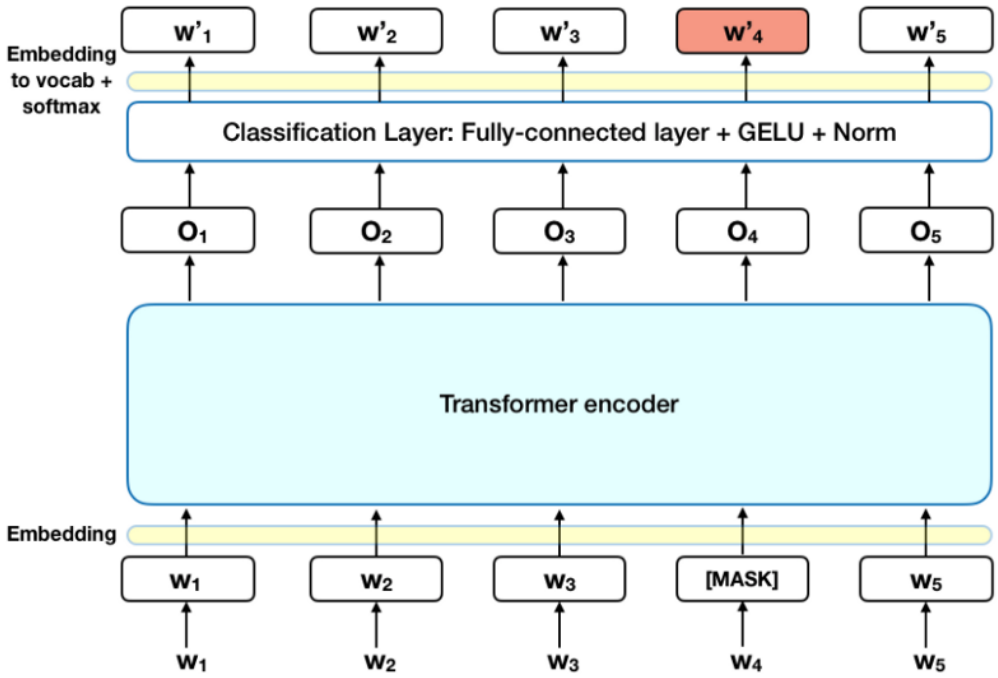
The music festival industry is fast growing and its growth can be tracked on an event by event basis through “hype” generated on social media. With the vast amount of data on websites like Twitter, we can attempt to predict how festival attendees feel about a festival before they even attend. We can then use this information to extrapolate attendance numbers as well as how much investment, such as in the form of catering, should go into the festival.

We start by importing the data into a pandas dataframe and performing a quick analysis on our dataset. We see that our most common sentiment is positive by a wide margin which indicates that our dataset is imbalanced towards this class. This has many implications when building an NLP model and we’ll keep that in mind as we train our model.



The text data in our dataset needs to be cleaned so that it can be tokenized for our model. We use a twitter preprocessor module as well as a regex expression that cleans the tweets of any mentions (“@“ symbol) as well as the hashtag and retweet markers. This leaves us with text with basic punctuation which are acceptable inputs for the model that we plan on using. We can visualize our text using a word cloud and after we filter out the most common words we can see the artists that have generated the most buzz on Twitter.

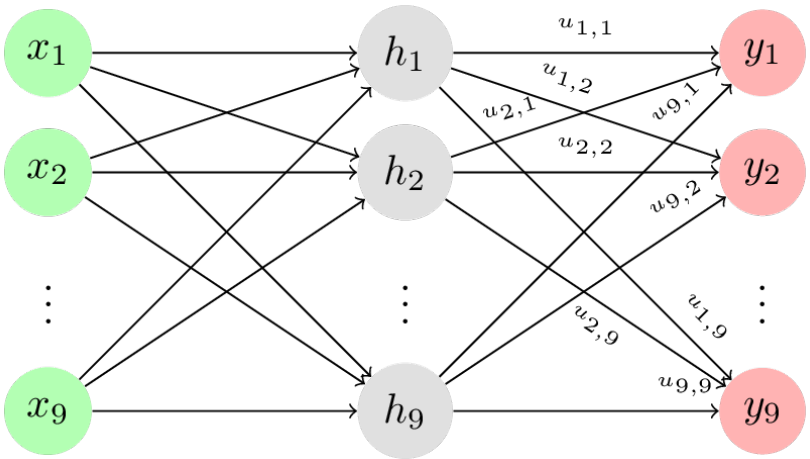
We decide to use the BERT (Bidirectional Encoder Representations from Transformers) model made by Google which makes use of the transformers mechanism. This means that it attempts to learn the contextual relationships between words in a text. We can see the basic structure of the model in the diagram below.



BERT is trained by randomly masking words in a given sentence and have the model attempt to predict the masked word. This occurs “bi-directionally” so the model is able to use the surrounding text in both directions to achieve state of the art results. We begin by tokenizing our Twitter text data and defining our Dataset class.

```
# We prepare our datasets for BERT by defining a PyTorch Dataset class
class TwitterDataset(Dataset):

    def __init__(self, tweets, targets, tokenizer, max_len):
        self.tweets = tweets
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.tweets)

    def __getitem__(self, item):
        tweet = str(self.tweets[item])
        target = self.targets[item]

        encoding = self.tokenizer.encode_plus(
            tweet,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            pad_to_max_length=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
          'tweet_text': tweet,
          'input_ids': encoding['input_ids'].flatten(),
          'attention_mask': encoding['attention_mask'].flatten(),
          'targets': torch.tensor(target, dtype=torch.long)
        }
```

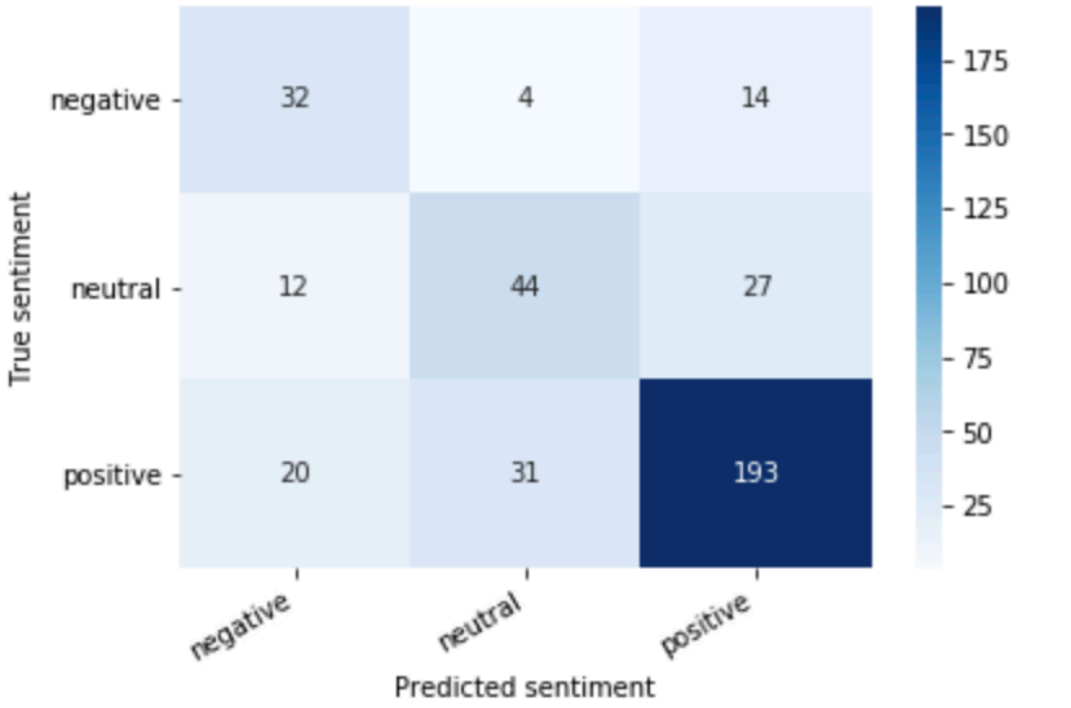
We then split our data into training, testing, and cross-validation sets using the train_test_split module and build our Sentiment Classifier which is a Bert cased model with an added dropout layer and a linear transformation output layer. The dropout layer randomly ignores some outputs in order make the training process more noisy and force the model to be more thorough in developing connections between nodes. In the figure below this can be understood as a purposefully dropping some of the h-nodes.



We train our model by splitting it into epochs and evaluating the loss and accuracy for both our training and cross-validation sets. This process is computationally intensive and took approximately 1 hour using just my CPU.

```
Epoch 1/5
----------
Train loss 0.9196283157128262 accuracy 0.5845234141481236
Val   loss 0.8324442766606808 accuracy 0.6409574468085106

Epoch 2/5
----------
Train loss 0.7021312635258374 accuracy 0.705745599468615
Val   loss 0.8030052720569074 accuracy 0.6728723404255319

Epoch 3/5
----------
Train loss 0.4693490761180085 accuracy 0.8226502822982398
Val   loss 0.9136608433909714 accuracy 0.675531914893617

Epoch 4/5
----------
Train loss 0.32478459853960934 accuracy 0.8894055131185653
Val   loss 1.018641819478944 accuracy 0.6888297872340425

Epoch 5/5
----------
Train loss 0.25012319019635715 accuracy 0.9169711059448689
Val   loss 1.0992481242865324 accuracy 0.6968085106382979

Wall time: 1h 8min 53s
```

After our model finishes training, we can compare the test set predicted sentiments with their actual values and plot it as a heatmap. The dark blue area of the map indicates that most of our positive sentiment tweets were correctly predicted positive and the adjacent squares tell us that most of our incorrectly predicted sentiments were predicted as positive. The pre-trained pipeline provided by Hugging Face performs at roughly 50% accuracy while our fine-tuned model achieves a test set accuracy of around 71%. This significant increase could be attributed to the model being better at understanding Twitter lingo and idiomatic expressions that were learned through the training data.



We also further apply our model to recent tweets regarding the upcoming music festival “dancefestopia” using a Twitter api query that stores and cleans the tweet to be evaluated by our model. We obtain the following results from a randomly selected sample of 100 tweets. While we would need to human label the actual sentiment of these tweets to determine the model’s performance, we can be safe in assuming that the model believes that people are generally enthusiastic and are looking forward to the festival. We can further improve our query to give us more insight into specifically what aspects of the festival they might be excited about.

```
Positive tweets percentage: 69.23076923076923 %
Negative tweets percentage: 10.256410256410257 %
Neutral tweets percentage: 20.512820512820515 %
```