X1120004 Daniel Honies

Friday, 29. December 2023

Homework 2 - Kaggle Competition

Emotion Prediction of Tweets

The second homework Kaggle competition deals with predicting the emotion of tweets. There are 8 classes (or say emotions) in our dataset: anger, anticipation, disgust, fear, sadness, surprise, trust, and joy. Given input is as follows:

tweets_DM.json - Raw data from Twitter

emotion.csv - Lists the emotion labels per tweet_id

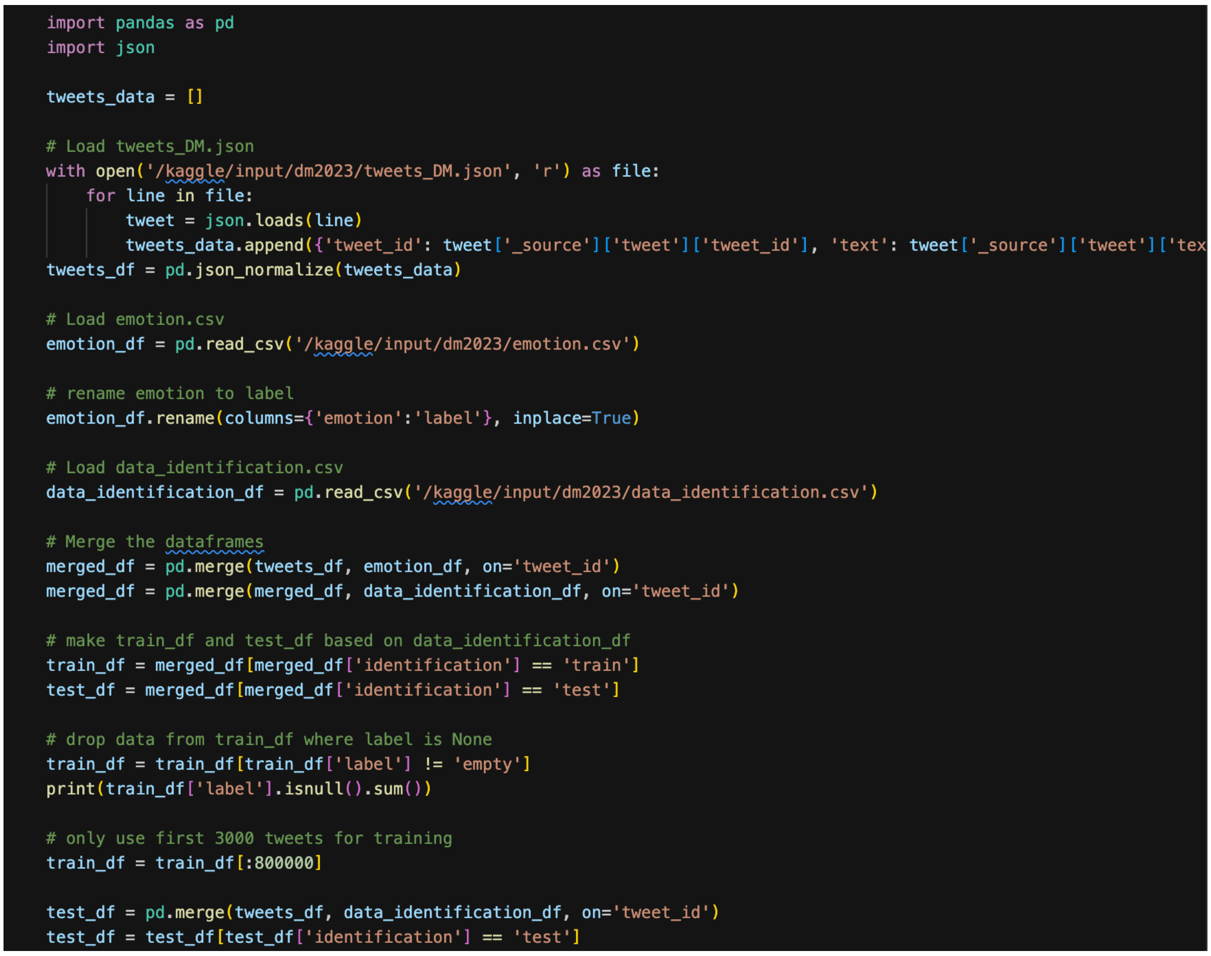
data_identification.csv - A file that identifies membership of training or testing set per tweet_id. Note that you won´t be

provided with the labels for the testing set, but you will have to predict for these when you make your submission.

sampleSubmission.csv - A submission format you should follow for submitting to the competition

Preparation of Data:

We use pandas to load, clean up and merge the input data through the following code:



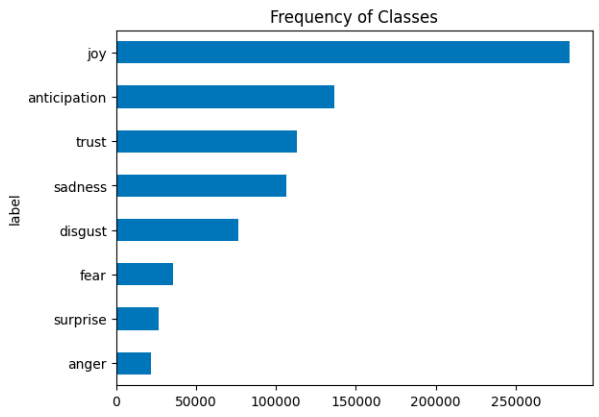
```
import pandas as pd
import json

tweets_data = []

# Load tweets_DM.json
with open('/kaggle/input/dm2023/tweets_DM.json', 'r') as file:
    for line in file:
        tweet = json.loads(line)
        tweets_data.append({'tweet_id': tweet['_source']['tweet']['tweet_id'], 'text': tweet['_source']['tweet']['tex
tweets_df = pd.json_normalize(tweets_data)

# Load emotion.csv
emotion_df = pd.read_csv('/kaggle/input/dm2023/emotion.csv')

# rename emotion to label
emotion_df.rename(columns={'emotion':'label'}, inplace=True)

# Load data_identification.csv
data_identification_df = pd.read_csv('/kaggle/input/dm2023/data_identification.csv')

# Merge the dataframes
merged_df = pd.merge(tweets_df, emotion_df, on='tweet_id')
merged_df = pd.merge(merged_df, data_identification_df, on='tweet_id')

# make train_df and test_df based on data_identification_df
train_df = merged_df[merged_df['identification'] == 'train']
test_df = merged_df[merged_df['identification'] == 'test']

# drop data from train_df where label is None
train_df = train_df[train_df['label'] != 'empty']
print(train_df['label'].isnull().sum())

# only use first 3000 tweets for training
train_df = train_df[:800000]

test_df = pd.merge(tweets_df, data_identification_df, on='tweet_id')
test_df = test_df[test_df['identification'] == 'test']
```

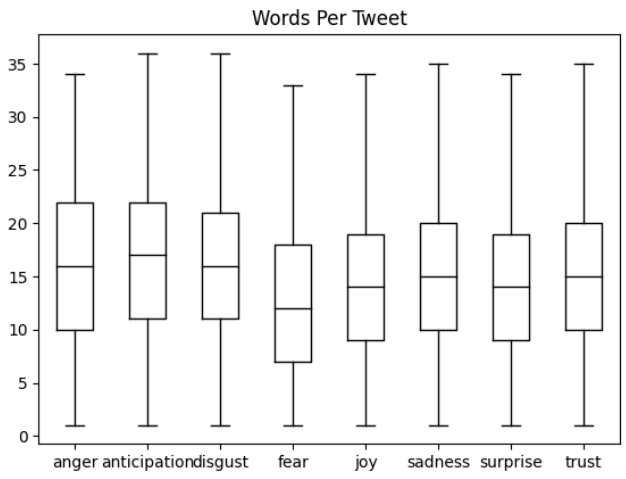
Splitting Data: The data is split into 'training' and 'test' sets based on the 'identification' column.

Limitation of Training Size: Due to the memory available in the Kaggle training environment we limit the training data frame to the first 800k rows.

Data Exploration

Following we can see the frequency as well as the length of the tweets



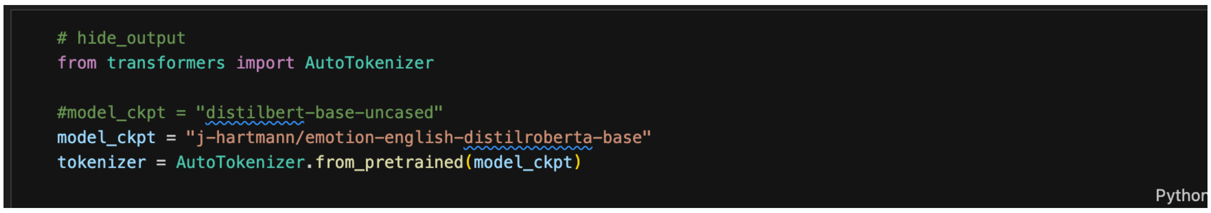


Model

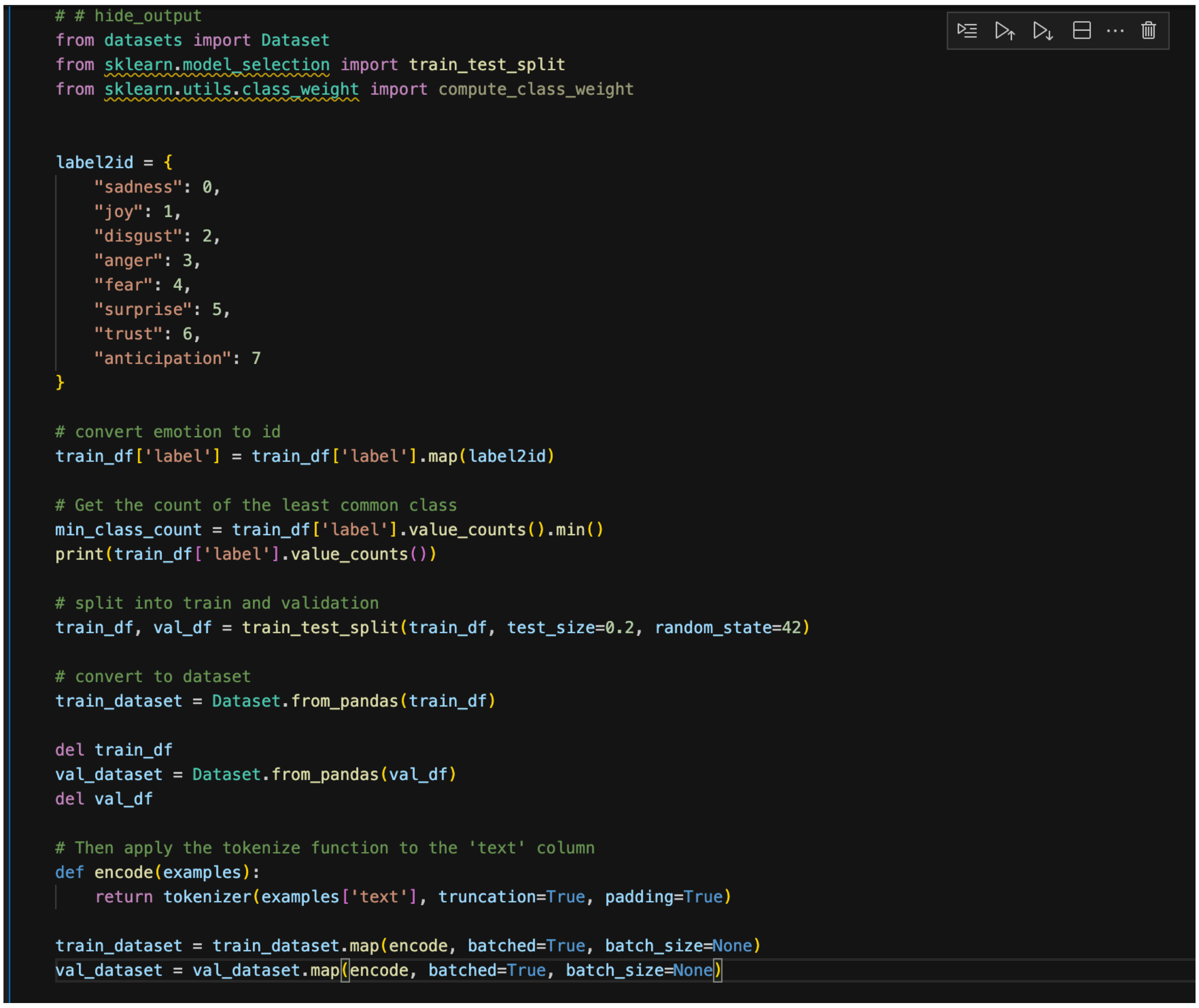
For our main model we use the transformers library and a custom trained emotion model from huggingface based on the RoBERTa LLM.

The RoBERTa (Robustly Optimized BERT Approach) language model is an enhanced version of the original BERT (Bidirectional Encoder Representations from Transformers) model, developed by Facebook AI. It modifies key hyperparameters in BERT, training the model on a much larger dataset and for a longer duration, leading to improved performance. RoBERTa differs from BERT in aspects like dynamic masking, removing the Next Sentence Prediction (NSP) task, and training with larger batch sizes and longer sequences. This model excels in a variety of natural language processing tasks, including sentiment analysis, question answering, and text classification, outperforming its predecessors in many benchmarks. RoBERTa's architecture allows it to understand and generate human-like text, making it a powerful tool in AI-driven language processing applications.

However the checkpoint file used is only trained on 7 emotion classes which are not the same as ours. We therefore have to retrain it. First we prepare the data for the transformers library by converting the labels to ids and splitting the training data into training and validation set. Finally we convert the data frames to transformer datasets.



```python
# hide_output
from transformers import AutoTokenizer

#model_ckpt = "distilbert-base-uncased"
model_ckpt = "j-hartmann/emotion-english-distilroberta-base"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
```

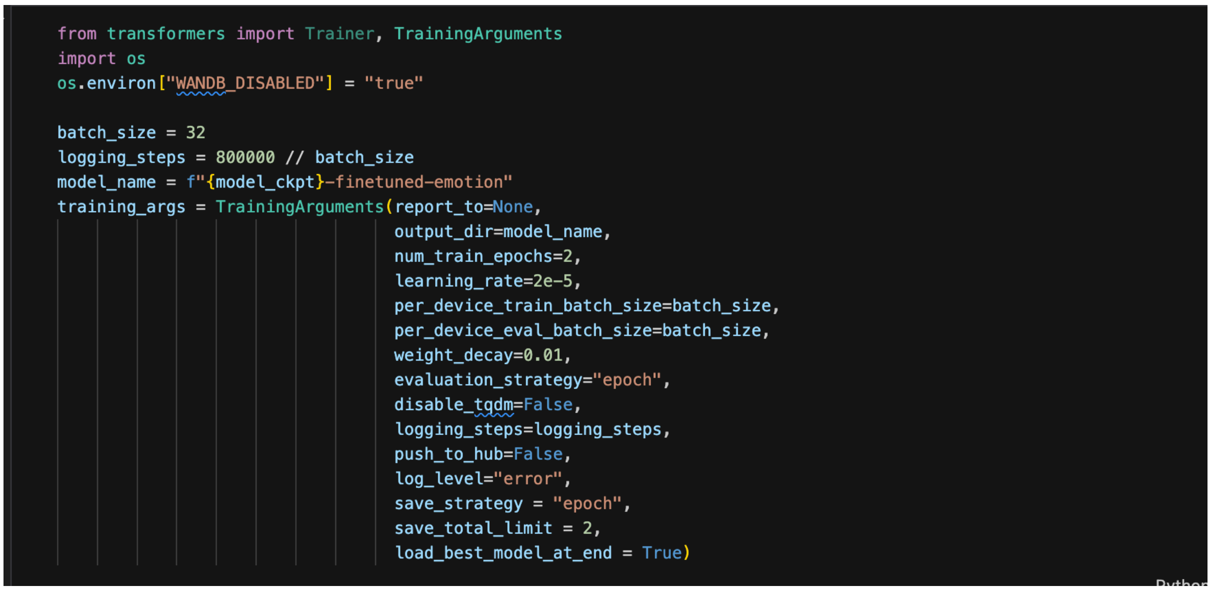


```python
# # hide_output
from datasets import Dataset
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight


label2id = {
    "sadness": 0,
    "joy": 1,
    "disgust": 2,
    "anger": 3,
    "fear": 4,
    "surprise": 5,
    "trust": 6,
    "anticipation": 7
}

# convert emotion to id
train_df['label'] = train_df['label'].map(label2id)

# Get the count of the least common class
min_class_count = train_df['label'].value_counts().min()
print(train_df['label'].value_counts())

# split into train and validation
train_df, val_df = train_test_split(train_df, test_size=0.2, random_state=42)

# convert to dataset
train_dataset = Dataset.from_pandas(train_df)

del train_df
val_dataset = Dataset.from_pandas(val_df)
del val_df

# Then apply the tokenize function to the 'text' column
def encode(examples):
    return tokenizer(examples['text'], truncation=True, padding=True)

train_dataset = train_dataset.map(encode, batched=True, batch_size=None)
val_dataset = val_dataset.map(encode, batched=True, batch_size=None)
```

For the tokeniser we use the provided AutoTokenizer by the transformers library.

Training

Training is done for 2 epochs. On kaggle it takes around 7 hours to finish.



```python
from transformers import Trainer, TrainingArguments
import os
os.environ["WANDB_DISABLED"] = "true"

batch_size = 32
logging_steps = 800000 // batch_size
model_name = f"{model_ckpt}-finetuned-emotion"
training_args = TrainingArguments(report_to=None,
                                  output_dir=model_name,
                                  num_train_epochs=2,
                                  learning_rate=2e-5,
                                  per_device_train_batch_size=batch_size,
                                  per_device_eval_batch_size=batch_size,
                                  weight_decay=0.01,
                                  evaluation_strategy="epoch",
                                  disable_tqdm=False,
                                  logging_steps=logging_steps,
                                  push_to_hub=False,
                                  log_level="error",
                                  save_strategy = "epoch",
                                  save_total_limit = 2,
                                  load_best_model_at_end = True)
```

Evaluation

On the validation set the scores are as follows:

'test_loss': 0.96634441614151,

'test_accuracy': 0.6529,

'test_f1': 0.6443129089481171,

'test_runtime': 790.6953,

'test_samples_per_second': 202.354,

'test_steps_per_second': 6.324

On the Kaggle Competition I scored 19th with an accuracy of 0.52947.

Improvements and Experiments

I tried several avenues to improve my score, considering the following

- Only consider hashtags
  - I preprocessed the tweet text to only consider the hashtags as the dataset description mentions that labels are given by hashtags. This resulted in a very similar but slightly lower score. I therefore decided to discard this approach
- Change training data size
  - At first I trained with less data so that I can experiment faster and don't always have to wait 7 hours for a score. In general increasing the size of the dataset resulted in better scores. However with a set bigger than 800k rows I ran into

memory issues on the Kaggle training. As I was satisfied with the score, I did not apply more memory saving options.

- Balance dataset
    - I tried to balance the dataset by limiting the size of rows for each class to the amount of samples that the smallest class had. This resulted in a dataset of around 200k rows and did not improve my score.
- Number of Epochs
    - I ran experiments with one, two and three epochs. Generally the score did not improve significantly after the second epoch and therefore did not justify the additional training time.