IMPACT OF EMOTION ON BITCOIN

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

INDIVIDUAL REPORT BY,

Mrunalini Devineni

DATS6312
Professor Amir Jafari
May 2, 2022

INTRODUCTION

In our project, we aim to analyze the impact of emotion of cryptocurrency market players on the price of Bitcoin using the sentiment classification abilities of transformer based natural language processing techniques.

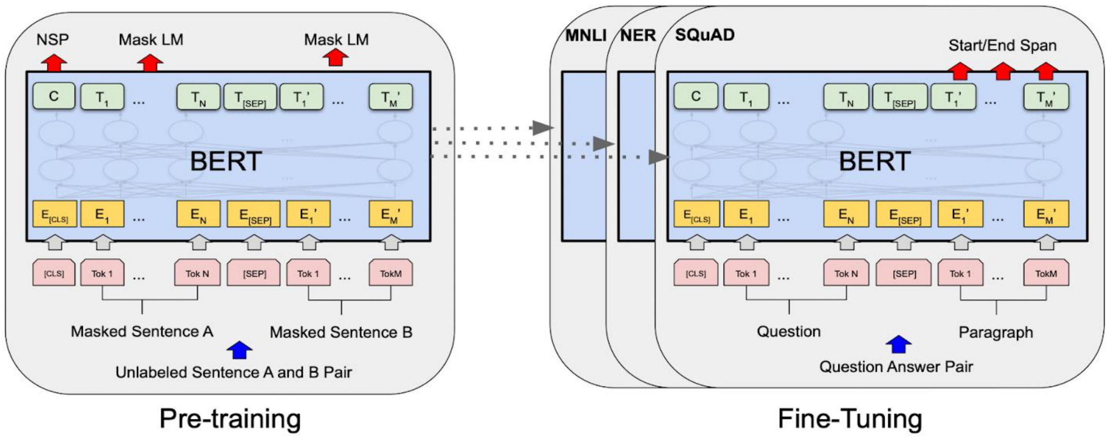
GOALS

1. Develop a dataset of Reddit comments from the r/cryptocurrency subreddit
2. Fine tune a transformer model on an emotion classification dataset
3. Repurpose the model to predict the emotion exhibited by Reddit comments
4. Investigate trends between certain emotions and the price of Bitcoin

INDIVIDUAL CONTRIBUTION:

I have worked on the BERT model for this project that includes running and understanding of the model. I have run the model on the emotions Dataset to first gather context understanding over labels and then repurpose the model to predict emotions on Reddit comments in order to understand what percent of context exhibits a particular emotion on a particular day. I have also done the writeup part of data, About BERT and its workflow and how it went through about in our project.

About the model, BERT, short for Bidirectional Encoder Representations from Transformers, is a Machine Learning (ML) model for natural language processing. The most common language tasks performed by it are sentiment analysis and name entity recognition. BERT has revolutionized the NLP space by performing better previous developed models. BERT uses transformer a mechanism that learns contextual relations between words in text. It consists of an encoder that reads the input and decoder that predicts the task. But Bert needs only encoder as its goal is to generate a language model. This is called bidirectional because it reads the entire sequence of words at once.



The model is trained on unlabeled data across many pre-training tasks during pre-training. The BERT model is fine-tuned using labeled data from downstream tasks after it is initialized with the pre-trained parameters. We pertain BERT using two supervised tasks:

1. Masked Language model: Here 15 percent of the words in each word sequence are substituted with a [MASK] token before being fed into BERT. Based on the context provided by the other, non-masked words in the sequence, the model then attempts to predict the original value of the masked words. Further, the BERT loss function considers only predicted masked values.
2. Next Sentence Prediction: The BERT training approach involves feeding the model pairs of sentences and learning to predict whether the second sentence in the pair is the next sentence in the original document.

Fine-tuning is simple since the transformer's mechanism allows BERT to mimic a wide range of downstream tasks just by swapping relevant input and outputs. For each task we simply plug in task specific input and output into BERT and fine tune all parameters end to end.

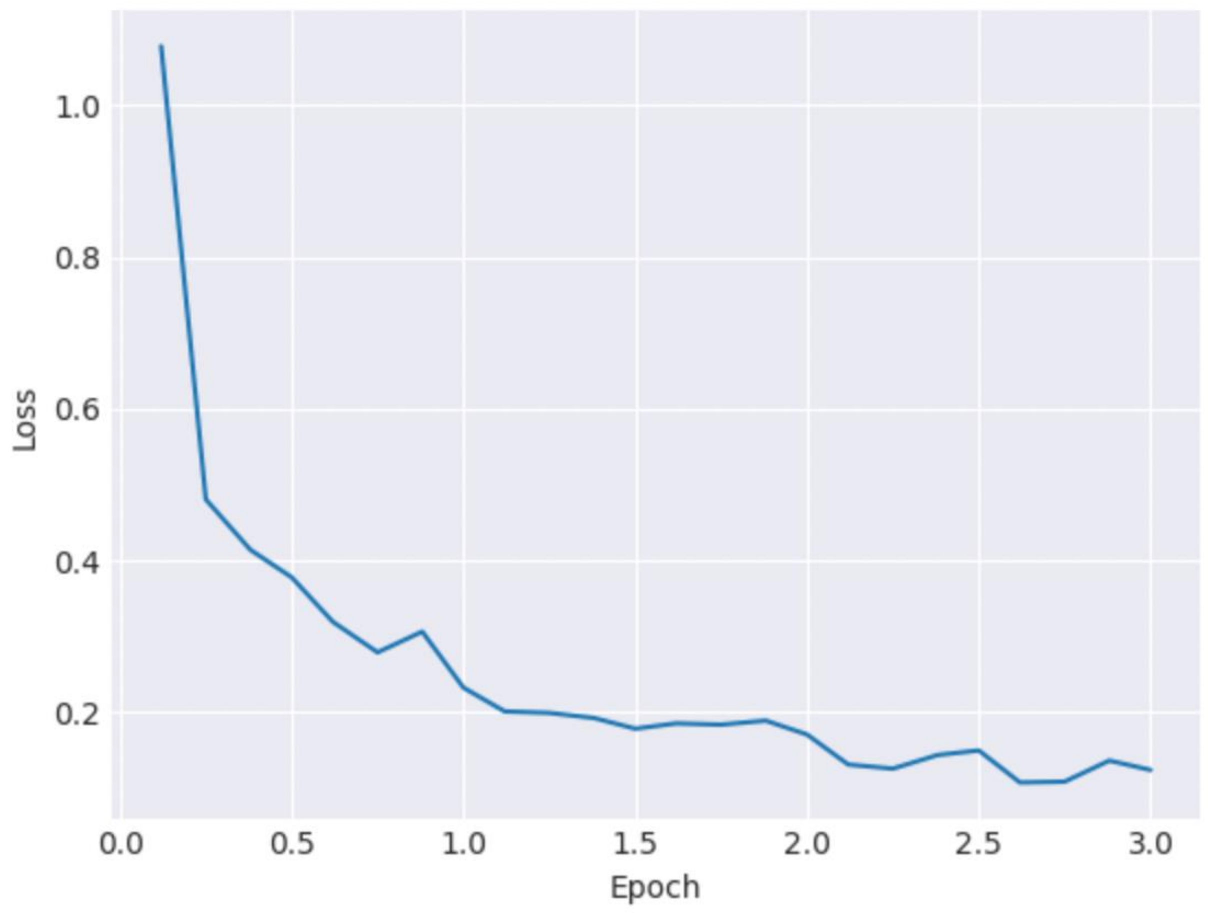
EXPERIMENT

Training BERT

We use the Hugging Face transformer APIs to train our model. We first load the emotions dataset, which has labels and texts for three splits: train, validations and test. We use the BERT tokenizer to convert our text inputs into integer encodings and Boolean attention masks. We also use a data collator to collate the encodings and convert them to the same token length. We then load a pre-trained BERT model and train it on our tokenized dataset according to the following hyperparameters.

Hyperparameters	**Values**
Learning rate	2e-5
Batch size	4
num_train_epochs	3
weight decay	0.01

We monitor the training by plotting the loss on the train set. Since the loss has not plateaued entirely, we can be confident that we have not overfitted our model.

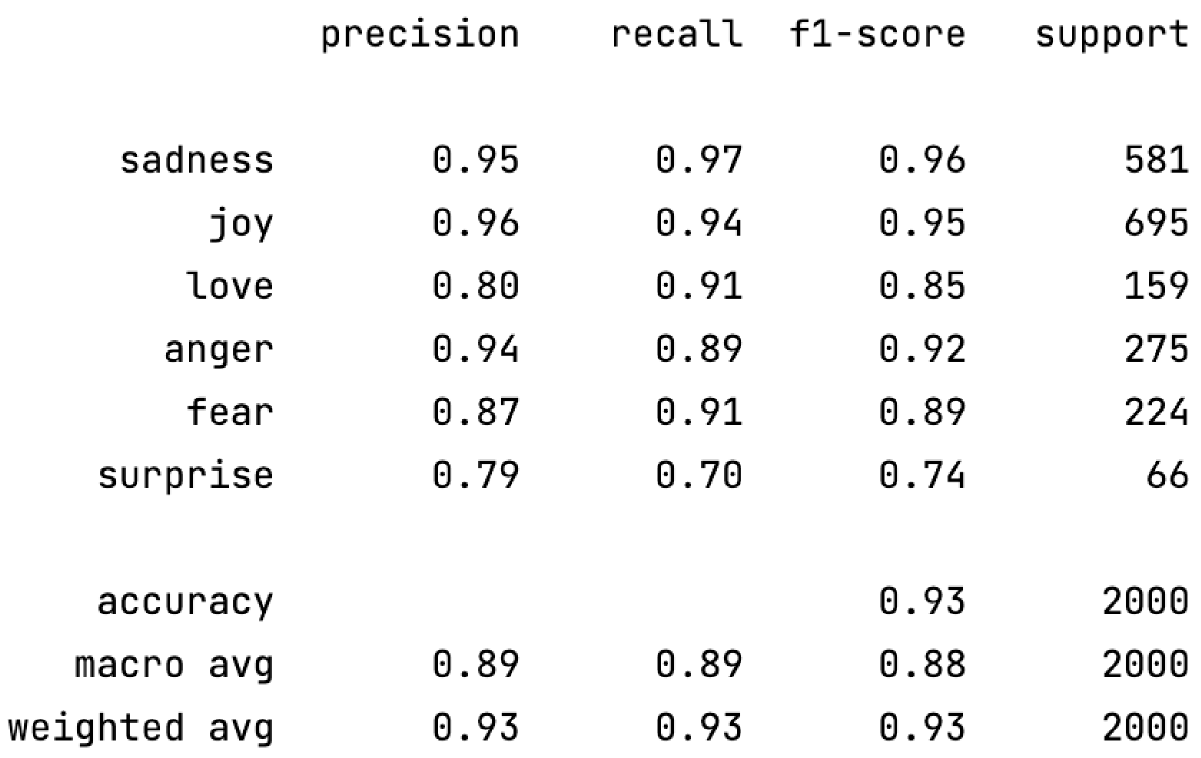


Since our evaluation strategy was epoch, we evaluated on the validation set at the end of every epoch.

Epoch	**Eval F1 Score**	**Eval Loss**
0	0.899	0.293
1	0.919	0.203
2	0.921	0.227

Testing BERT

In order to make sure our model has not overfitted, we run it on the unseen test dataset consisting of 2000 data points. The classification performance is as follows.



```
              precision    recall  f1-score   support

     sadness       0.95      0.97      0.96       581
         joy       0.96      0.94      0.95       695
        love       0.80      0.91      0.85       159
       anger       0.94      0.89      0.92       275
        fear       0.87      0.91      0.89       224
    surprise       0.79      0.70      0.74        66

    accuracy                           0.93      2000
   macro avg       0.89      0.89      0.88      2000
weighted avg       0.93      0.93      0.93      2000
```

The performance on the test set is like that on the validation set, leading to the conclusion that our model has not overfitted and is ready for downstream inference.

From the above we could see that sadness comprises 96% of the comments and surprise being the least of about 74% one of the reasons being there are very less comments supporting the emotion.

CONCLUSION:

In our project, we use the zero-shot classification power of transformer-based models to predict the emotion of Reddit comments, even though the model was trained on an entirely different dataset. The correlation between the percentage of comments exhibiting certain emotions and the price of Bitcoin indicates that these predictions were reliable. Furthermore, our work also shows that cryptocurrency market analysts can harness NLP techniques to advise investing, because cryptocurrency prices are more volatile than traditional stock prices.

REFERENCES

1. https://huggingface.co/datasets/emotion
2. https://huggingface.co/docs/transformers/training#train-in-native-pytorch
3. https://arxiv.org/pdf/1810.04805.pdf
4. https://dzone.com/articles/bert-transformers-how-do-they-work
5. https://github.com/yk/huggingface-nlp-demo/blob/master/demo.py

APEENDIX:

BERT model :

```
from datasets import load_dataset

emotion_raw = load_dataset("emotion")

from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")


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


emotion_tokenized = emotion_raw.map(tokenize_function, batched=True)

from transformers import DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```
from transformers import AutoModelForSequenceClassification,
TrainingArguments, Trainer

model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased",
num_labels=6)

from datasets import load_metric

metric = load_metric("accuracy")
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)

training_args = TrainingArguments(
    output_dir="model",
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=emotion_tokenized["train"],
    eval_dataset=emotion_tokenized["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics
)

trainer.train()
trainer.evaluate()

predictions = trainer.predict(emotion_tokenized['test'])

import numpy as np
predicted_labels = np.argmax(predictions.predictions, axis=-1)
target_labels = predictions.label_ids

from sklearn.metrics import accuracy_score, f1_score

f1_score(target_labels, predicted_labels, average='macro')
accuracy_score(target_labels, predicted_labels)
```