Post-Training the CryptoBERT model on the MLM task

```
!pip install -q transformers
!pip install emoji
!pip install pandas
!pip install numpy
```

```
     |████████████████████████████████| 4.7 MB 4.1 MB/s
     |████████████████████████████████| 120 kB 47.5 MB/s
     |████████████████████████████████| 6.6 MB 37.1 MB/s
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting emoji
  Downloading emoji-2.0.0.tar.gz (197 kB)
     |████████████████████████████████| 197 kB 4.2 MB/s
Building wheels for collected packages: emoji
  Building wheel for emoji (setup.py) ... done
  Created wheel for emoji: filename=emoji-2.0.0-py3-none-any.whl size=193022 s
  Stored in directory: /root/.cache/pip/wheels/ec/29/4d/3cfe7452ac7d8d83b1930f
Successfully built emoji
Installing collected packages: emoji
Successfully installed emoji-2.0.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packa
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
```

```
import numpy as np
import pandas as pd

import logging
import time

import random
import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F

from transformers import AutoTokenizer, AutoModelForMaskedLM

# max_len = 128
max_len = 32

epochs = 12


RANDOM_SEED = 2137
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
random.seed(RANDOM_SEED)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
torch.set_num_threads(16)
if torch.cuda.is_available():
    print(torch.cuda.get_device_name(0))
```

```
cpu
```

```
# path_posts = '/datasets/cryptobert-corpus/'
path_posts = '/content/drive/MyDrive/Data_Thesis_Temp/SA_files/'
# fn_bull = path_posts + 'posts-bull.txt'
# fn_net = path_posts + 'posts-net.txt'
# fn_bear = path_posts + 'posts-bear.txt'

# fn_bull_test = path_posts + 'posts-bull-test.txt'
# fn_net_test = path_posts + 'posts-net-test.txt'
# fn_bear_test = path_posts + 'posts-bear-test.txt'

# fn_bull_new = path_posts + 'posts-bull-new.txt'
# fn_net_new = path_posts + 'posts-net-new.txt'
# fn_bear_new = path_posts + 'posts-bear-new.txt'
# fn_red = path_posts + 'data-red.txt'
# fn_twit = path_posts + 'data-twit.txt'
# fn_tel = path_posts + 'data-tel.txt'
fn_data = path_posts + "bert-train-updated.txt"


config = open(fn_red, "r")
text_red = config.read().splitlines()
config.close()
config = open(fn_twit, "r")
text_twit = config.read().splitlines()
config.close()
config = open(fn_tel, "r")
text_tel = config.read().splitlines()
config.close()
config = open(fn_bull, "r")
text_bull_1 = config.read().splitlines()
config.close()
config = open(fn_bear, "r")
text_bear_1 = config.read().splitlines()
config.close()
config = open(fn_net, "r")
text_net_1 = config.read().splitlines()
config.close()

config = open(fn_bull_test, "r")
text_bull_2 = config.read().splitlines()
config.close()
config = open(fn_bear_test, "r")
text_bear_2 = config.read().splitlines()
config.close()
```

```
config = open(fn_net_test, "r")
text_net_2 = config.read().splitlines()
config.close()

config = open(fn_bull_new, "r")
text_bull_3 = config.read().splitlines()
config.close()
config = open(fn_bear_new, "r")
text_bear_3 = config.read().splitlines()
config.close()
config = open(fn_net_new, "r")
text_net_3 = config.read().splitlines()
config.close()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-4-1be982059fb6> in <module>
----> 1 config = open(fn_red, "r")
      2 text_red = config.read().splitlines()
      3 config.close()
      4 config = open(fn_twit, "r")
      5 text_twit = config.read().splitlines()

FileNotFoundError: [Errno 2] No such file or directory:
'/datasets/cryptobert-corpus/data-red.txt'
```

```
text = text_red + text_twit + text_tel + text_bull_1 + text_bear_1 + text_net_1 +
# df_text = text + text
# random.shuffle(df_text)


random.shuffle(text)


df_text = text[:100000]


config = open(fn_data, "r")
text = config.read().splitlines()
config.close()


df_counts = [word_count(t) for t in text]
```

```
table_counts = np.array(df_counts)


av_counts = np.mean(table_counts)


std_counts = np.std(table_counts)


sum_counts = sum(df_counts)


average_count = sum_counts / len(df_counts)


numeros = []
for i in range(2, 26):
  numeros.append(4*i)


fractions = []
for numio in numeros:
  count = num_count(df_counts, numio)
  frac = count/sample_size
  fractions.append(frac)
```

```
fractions
```

```
[0.6599768665097745,
 0.4383439107135093,
 0.2937699280984951,
 0.20519500533568313,
 0.14888213241361853,
 0.11174930769565211,
 0.08515870805440631,
 0.06657270928296942,
 0.05265645561697414,
 0.042212377837823646,
 0.03436532920885099,
 0.02811532651840972,
 0.023263922818342695,
 0.019474059626205156,
 0.01649477258022788,
 0.014059169906316683,
 0.012121621720513518,
 0.010547821194564534,
 0.009218097500730724,
 0.008073476166349338,
 0.007166905075635837,
 0.006317586575171574 5,
 0.005620654668303029,
 0.004977531586856699]
```

```
sample_size = len(text)
```

```
print('over 26: '+str(over_26) + " as fraction of sample: "+ str(over_26/sample_s
print('over 32: '+ str(over_32) + " as fraction of sample: "+ str(over_32/sample_
print('over 48: '+str(over_48) + " as fraction of sample: "+ str(over_48/sample_s
print('over 64: '+str(over_64) + " as fraction of sample: "+ str(over_64/sample_s
print('over 90: '+str(over_90) + " as fraction of sample: "+ str(over_90/sample_s
print('over 128: '+str(over_128) + " as fraction of sample: "+ str(over_128/sampl
```

```
over 26: 476676 as fraction of sample: 0.20519500533568313
over 32: 197827 as fraction of sample: 0.08515870805440631
over 48: 79832 as fraction of sample: 0.03436532920885099
over 64: 38318 as fraction of sample: 0.01649477258022788
over 90: 22916 as fraction of sample: 0.009864664347004075
over 128: 5207 as fraction of sample: 0.0022414604317878436
```

```
def word_count(str):
  words = str.split()
  count = len(words)
  return count


def num_count(list_num, bar):
  count = 0
  for num in list_num:
    if num >= bar:
      count = count +1
  return count


from transformers import AutoTokenizer
path_token = 'cardiffnlp/twitter-roberta-base'
tokenizer = AutoTokenizer.from_pretrained(path_token, use_fast = True)
```

```
Downloading config.json: 100% 565/565 [00:00<00:00, 11.5kB/s]
Downloading vocab.json: 100% 878k/878k [00:01<00:00, 1.12MB/s]
```

```
tokens = tokenizer('🤡')


tokens2 = tokenizer('🤣')


tokens['input_ids']
```

```
[0, 6569, 10470, 5543, 2]
```

```
tokens2['input_ids']
```

```
[0, 6569, 10470, 2469, 2]
```

```
tokenizer.decode(tokens['input_ids'][1])
```

```
'<unk>'
```

```
tokenizer.decode(tokens['input_ids'][3])
```

```
'[UNK]'
```

```
df_wc = [word_count(c) for c in df_text]


counts_tokens = []
for line in tokens['input_ids']:
  temp_c = len(line)
  counts_tokens.append(temp_c)


sum_words = sum(df_wc)
sum_toks = sum(counts_tokens)


toks_per_word = sum_toks/sum_words


df_min20 = [max(0, cc - 20) for cc in df_counts]


frac_cut = sum(df_min20)/sum(df_counts)


from transformers import AutoTokenizer, AutoModelForMaskedLM
path_model = '/datasets/cryptobert-checkpoint/'
path_token = '/datasets/cryptobert-tokenizer/'
tokenizer = AutoTokenizer.from_pretrained(path_token, use_fast = True)
model = AutoModelForMaskedLM.from_pretrained(path_model)
```

```
inputs = tokenizer(df_text, return_tensors='pt', max_length=max_len, truncation=T
inputs['labels'] = inputs.input_ids.detach().clone()
# create random array of floats with equal dimensions to input_ids tensor
rand = torch.rand(inputs.input_ids.shape)
# create mask array
mask_arr = (rand < 0.15) * (inputs.input_ids != 1) * (inputs.input_ids != 2) * (i

selection = []

for i in range(inputs.input_ids.shape[0]):
    selection.append(
        torch.flatten(mask_arr[i].nonzero()).tolist()
    )
for i in range(inputs.input_ids.shape[0]):
    inputs.input_ids[i, selection[i]] = 50264



class CryptoDataset(torch.utils.data.Dataset):
    def __init__(self, encodings):
        self.encodings = encodings
    def __getitem__(self, idx):
        return {key: torch.tensor(val[idx]) for key, val in self.encodings.items(
    def __len__(self):
        return len(self.encodings.input_ids)


dataset = CryptoDataset(inputs)


from transformers import TrainingArguments
path_out = 'datasets/cryptobert-out'
args = TrainingArguments(
    output_dir=path_out,
    per_device_train_batch_size=1504,
    num_train_epochs=epochs,
    learning_rate= 1e-4,
    save_total_limit = 5,
    gradient_accumulation_steps = 2,
    save_steps = 2290,
    fp16 = True,
    logging_steps = 2290
)
```

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=dataset
)


trainer.train()
```

Fine-Tuning BERT models for Sentiment Classification

```
!pip install -q transformers
!pip install emoji
!pip install pandas
!pip install numpy
```

```
import logging
import time
import torch
import pandas as pd
import numpy as np
from transformers import Trainer, TrainingArguments
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import random
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F

RANDOM_SEED = 2137

max_len = 64
num_classes = 3
epochs = 12

np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
random.seed(RANDOM_SEED)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
torch.set_num_threads(8)
if torch.cuda.is_available():
    print(torch.cuda.get_device_name(0))
    print(torch.cuda.get_device_properties(0))


path_posts = '/datasets/cryptobert-corpus/'

fn_bull = path_posts + 'posts-bull.txt'
fn_net = path_posts + 'posts-net.txt'
fn_bear = path_posts + 'posts-bear.txt'

fn_bull_test = path_posts + 'posts-bull-test.txt'
fn_net_test = path_posts + 'posts-net-test.txt'
fn_bear_test = path_posts + 'posts-bear-test.txt'

fn_bull_new = path_posts + 'posts-bull-new.txt'
fn_net_new = path_posts + 'posts-net-new.txt'
```

```
fn_bear_new = path_posts + 'posts-bear-new.txt'


config = open(fn_bull, "r")
text_bull_1 = config.read().splitlines()
config.close()
config = open(fn_bear, "r")
text_bear_1 = config.read().splitlines()
config.close()
config = open(fn_net, "r")
text_net_1 = config.read().splitlines()
config.close()

config = open(fn_bull_test, "r")
text_bull_2 = config.read().splitlines()
config.close()
config = open(fn_bear_test, "r")
text_bear_2 = config.read().splitlines()
config.close()
config = open(fn_net_test, "r")
text_net_2 = config.read().splitlines()
config.close()

config = open(fn_bull_new, "r")
text_bull_3 = config.read().splitlines()
config.close()
config = open(fn_bear_new, "r")
text_bear_3 = config.read().splitlines()
config.close()
config = open(fn_net_new, "r")
text_net_3 = config.read().splitlines()
config.close()
text_bear = text_bear_1 + text_bear_2 + text_bear_3
text_net = text_net_1 + text_net_2 + text_net_3
text_bull = text_bull_1 + text_bull_2 + text_bull_3

# The dataset needs to be balanced, so we limit the size to the smallest class -
len_class = len(text_bear)

# take a sample from net and bull, so that all 3 datasets have the same size
text_train_bear = text_bear.copy()
text_train_net = random.sample(text_net, len_class)
text_train_bull = random.sample(text_bull, len_class)

# generate labels for all 3 datasets, 0 for bear, 1 for neutral, 2 for bull
```

```
labels_bear = np.zeros(len_class, dtype = int)
labels_net = np.ones(len_class, dtype = int)
labels_bull = 2*np.ones(len_class, dtype = int)

# initialize dataframes with posts and labels and assign both to right columns
data_bear = pd.DataFrame()
data_net = pd.DataFrame()
data_bull = pd.DataFrame()

data_bear['text'] = pd.Series(text_train_bear)
data_bear['label'] = pd.Series(labels_bear)

data_net['text'] = pd.Series(text_train_net)
data_net['label'] = pd.Series(labels_net)

data_bull['text'] = pd.Series(text_train_bull)
data_bull['label'] = pd.Series(labels_bull)

# lastly concatenate the datasets into one, and shuffle
df = pd.concat((data_bear, data_net, data_bull))
df = df.sample(frac=1).reset_index(drop=True)

# Creating the dataset class
class StockTwitsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.ids = torch.tensor(encodings['input_ids'], dtype=torch.long)
        self.masks = torch.tensor(encodings['attention_mask'], dtype=torch.long)
        self.labels = torch.tensor(labels, dtype=torch.long)

    def __getitem__(self, idx):
        input_ids = self.ids[idx]
        attention_mask = self.masks[idx]
        labels = self.labels[idx]
        # item = {k: torch.tensor(v[idx]) for k, v in self.encodings.items()}
        # item["labels"] = torch.tensor([self.labels[idx]])
        # return item
        return {
      'input_ids': input_ids,
      'attention_mask': attention_mask,
      'labels': labels
    }

    def __len__(self):
        return len(self.labels)
```

```
# from transformers import AutoTokenizer, AutoModelForMaskedLM
path_model = '/datasets/cryptobert-mlm/'
# path_model = 'cardiffnlp/twitter-roberta-base'
path_token = '/datasets/cryptobert-tokenizer/'
tokenizer = AutoTokenizer.from_pretrained(path_token, use_fast = True)
model = AutoModelForSequenceClassification.from_pretrained(path_model, num_labels
new_id2label = {"0": "Bearish", "1": "Neutral", "2": "Bullish"}
new_label2id = {"Bearish": 0, "Neutral": 1, "Bullish": 2}
model.config.id2label = new_id2label
model.config.label2id = new_label2id
print(model.config)


df_train_X, df_val_X, df_train_y, df_val_y = train_test_split(df['text'], df['lab
encodings_train = tokenizer(df_train_X.tolist(), truncation=True, padding='max_le
encodings_val = tokenizer(df_val_X.tolist(), truncation=True, padding='max_length
dataset_train = StockTwitsDataset(encodings_train, df_train_y)
dataset_val = StockTwitsDataset(encodings_val, df_val_y)
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    # calculate accuracy using sklearn's function
    acc = accuracy_score(labels, preds)
    return {
      'accuracy': acc,
    }
```

```
path_save = '/notebooks/cps/cb-small'
training_args = TrainingArguments(
    output_dir=path_save,          # output directory
    num_train_epochs=epochs,              # total number of training epochs
    per_device_train_batch_size=1000,  # batch size per device during training
    per_device_eval_batch_size=1000,   # batch size for evaluation
    warmup_steps=500,                # number of warmup steps for learning rate s
    save_total_limit = 2,
    # dataloader_num_workers = 8,
    learning_rate= 1e-5,
    weight_decay=1e-6,               # strength of weight decay
    # logging_dir='./logs',            # directory for storing logs
    load_best_model_at_end=True,     # load the best model when finished training
    # but you can specify `metric_for_best_model` argument to change to accuracy
    # logging_steps=2000,               # log & save weights each logging_steps
    # save_steps=2000,
    logging_strategy = 'epoch',
    save_strategy = 'epoch',
    metric_for_best_model = 'accuracy',
    fp16 = True,
    evaluation_strategy = 'epoch'
    # evaluation_strategy="steps"     # evaluate each `logging_steps`
)

trainer = Trainer(
    model=model,                         # the instantiated Transformers model to
    args=training_args,                  # training arguments, defined above
    train_dataset=dataset_train,         # training dataset
    eval_dataset=dataset_val,          # evaluation dataset
    compute_metrics=compute_metrics,     # the callback that computes metrics of
)


# train the model
trainer.train()
# saving the fine tuned model & tokenizer
model.save_pretrained(path_save)
tokenizer.save_pretrained(path_save)
# evaluate the current model after training
trainer.evaluate()
```

BERT Sentiment Predictions

```
!pip install transformers
!pip install emoji
```

```
import random
import torch
import numpy as np
import pandas as pd

RANDOM_SEED = 2137

np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
random.seed(RANDOM_SEED)
```

```
import pandas as pd
path_test = '/content/drive/MyDrive/Data_Thesis_Temp/SA_files/test-data/'
fn_final_test = path_test + 'test-data-final.xlsx'
df = pd.read_excel(fn_final_test)
df_posts = df['text'].tolist()
df_val_y = df['label']
total_obs = len(df_val_y)
total_bear = len(df_val_y[df_val_y==0])
total_net = len(df_val_y[df_val_y==1])
total_bull = len(df_val_y[df_val_y==2])
```

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TextC
import torch
import time
import logging
import numpy as np
import pandas as pd
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
path_model_cb = '/content/drive/MyDrive/Data_Thesis_Temp/CryptoBERT/Sentiment-Cla
tokenizer_cb = AutoTokenizer.from_pretrained(path_model_cb, use_fast=True)
model_cb = AutoModelForSequenceClassification.from_pretrained(path_model_cb, num_
model_cb = model_cb.eval()
pipe_cb = TextClassificationPipeline(model=model_cb, tokenizer=tokenizer_cb, fram

preds_cryptobert_st = pipe_cb(df_posts)


cryptobert_st_labels = []
for pred in preds_cryptobert_st:
  temp_label = 1
  if pred['label']=='Bearish':
    temp_label = 0
  if pred['label']=='Bullish':
    temp_label = 2
  cryptobert_st_labels.append(temp_label)


df_met_cryptobert = compute_all_metrics(cryptobert_st_labels, df_val_y, model_nam
```

Function to compute all performance metrics in one go

```
# import pandas as pd
def compute_all_metrics(pred_labels, true_labels, model_name = 'Current Model'):
  correct_count = 0
  # tp = true positives, tn = true negatives, fp = false positives, fn = false ne
  tp_bear = 0
  tp_net = 0
  tp_bull = 0
  # tn = true negatives
  tn_bear = 0
  tn_net = 0
  tn_bull = 0
  # fp = false positives
```

```
    fp_bear = 0
    fp_net = 0
    fp_bull = 0
    # fn = false negatives
    fn_bear = 0
    fn_net = 0
    fn_bull = 0
    for lab_pred, lab_true in zip(pred_labels, true_labels):
      # count true positives
      if lab_pred == lab_true:
        correct_count += 1
        if(lab_pred == 0):
          tp_bear += 1
        if(lab_pred == 1):
          tp_net += 1
        if(lab_pred == 2):
          tp_bull += 1
      else:
        # count true negatives
        if (lab_pred != 0 and lab_true != 0):
          tn_bear += 1
        if (lab_pred != 1 and lab_true != 1):
          tn_net += 1
        if (lab_pred != 2 and lab_true != 2):
          tn_bull += 1
        # count false positives
        if(lab_pred == 0):
          fp_bear += 1
        if(lab_pred == 1):
          fp_net += 1
        if(lab_pred == 2):
          fp_bull += 1
        # count false negatives
        if(lab_true == 0):
          fn_bear += 1
        if(lab_true == 1):
          fn_net += 1
        if(lab_true == 2):
          fn_bull += 1
    # aggregate the results for macro computations:
    total_obs = len(true_labels)
    true_positives = tp_bear + tp_net + tp_bull
    true_negatives = tn_bear + tn_net + tn_bull
    false_positives = fp_bear + fp_net + fp_bull
    false_negatives = fn_bear + fn_net + fn_bull
```

```
    # calculate the performance measures
    acc = correct_count / total_obs


    prec_bull = tp_bull / (tp_bull + fp_bull)
    rec_bull = tp_bull / (tp_bull + fn_bull)
    f1_bull = (2*prec_bull*rec_bull)/(prec_bull+rec_bull)

    prec_net = tp_net / (tp_net + fp_net)
    rec_net = tp_net / (tp_net + fn_net)
    f1_net = (2*prec_net*rec_net)/(prec_net+rec_net)

    prec_bear = tp_bear / (tp_bear + fp_bear)
    rec_bear = tp_bear / (tp_bear + fn_bear)
    f1_bear = (2*prec_bear*rec_bear)/(prec_bear+rec_bear)


    macro_f1 = (f1_bear + f1_bull + f1_net) / 3
    macro_rec = (rec_bear + rec_bull + rec_net) / 3
    macro_prec = (prec_bear + prec_bull + rec_net) / 3


    df_measures = pd.DataFrame()
    df_measures['accuracy'] = [acc]
    df_measures['macro_f1'] = [macro_f1]
    df_measures['macro_precision'] = [macro_prec]
    df_measures['macro_recall'] = [macro_rec]

    df_measures['bull_precision'] = [prec_bull]
    df_measures['bull_recall'] = [rec_bull]
    df_measures['bull_f1'] = [f1_bull]

    df_measures['net_precision'] = [prec_net]
    df_measures['net_recall'] = [rec_net]
    df_measures['net_f1'] = [f1_net]

    df_measures['bear_precision'] = [prec_bear]
    df_measures['bear_recall'] = [rec_bear]
    df_measures['bear_f1'] = [f1_bear]

    print(f"Performance Measures for {model_name}:")
    print(df_measures)

    return df_measures
```

VADER prediction pipeline

[ ] ↳ 2 cells hidden