

Task 2: Data Processing 1

Muhammad Ali, 103960437, COS30018, 23/08/2024.

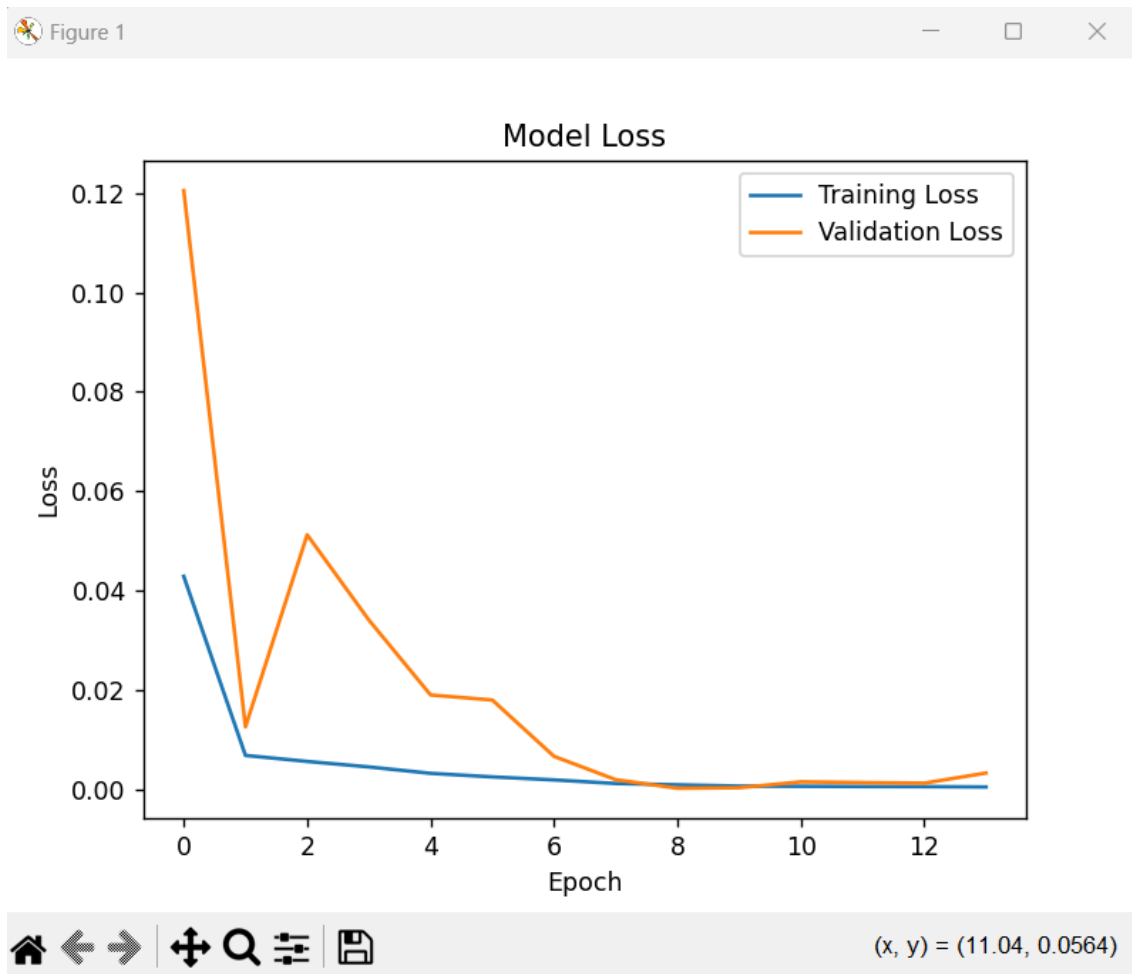
Requirements / Deliverables –

- Write a function that improves the data processing for the initial v0.1 stock prediction code.
- The function should load and process a dataset with the following requirements:
 - Allows you to specify the start date and the end date for the whole dataset as inputs.
 - Allows you to deal with the NaN issue in the data.
 - Allows you to use different methods to split the data into train/test data. E.g. You can split it according to some specified ratio of train/test and you can specify to split it by date or randomly.
 - Allows you to store the downloaded data on your local machine for future uses and to load the data locally to save time.
 - The function will also allow you to have an option to scale your feature columns and store the scalers in a data structure to allow future access to these scalers.

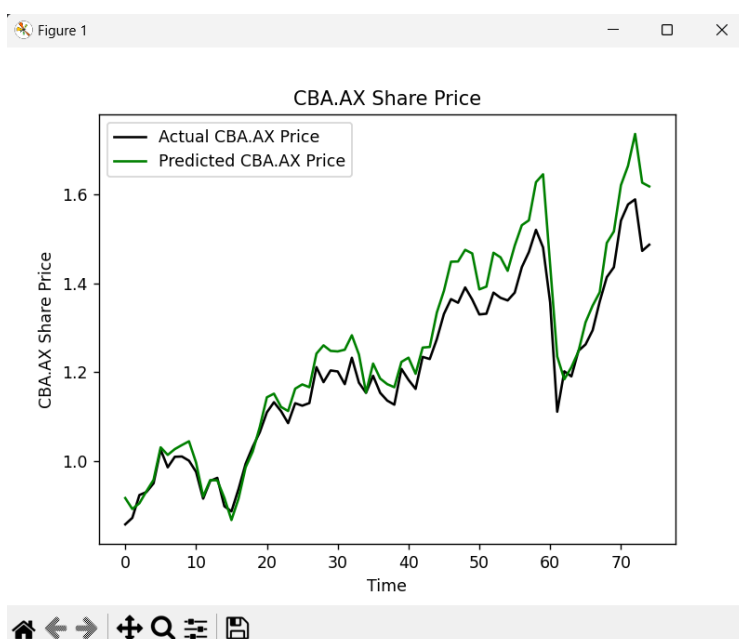
Result –

The provided function fulfils the requirements and uses an advanced data processing technique that utilizes Simple Moving Averages (SMA) and Relative Strength Index (RSI) for increased accuracy when training the model.

You must install the provided *requirements.txt* in repository folder Task 2.



The model now provides a graph that displays the model training loss and its validation loss (gradual decline is *good*).



Improve accuracy compared to the previous model. Accuracy can be further improved using more advanced techniques for architecture optimisation, data handling, and training techniques.

Load_and_process.py

```
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import yfinance as yf
from sklearn.model_selection import train_test_split

def add_technical_indicators(df):
    """
    Add technical indicators to the stock data DataFrame.
    This includes Simple Moving Average (SMA) and Relative Strength Index (RSI).

    Parameters:
    - df (pd.DataFrame): DataFrame containing stock data with at least a 'Close' column.

    Returns:
    - df (pd.DataFrame): DataFrame with added technical indicators.
    """
    # Simple Moving Average (SMA)
    df['SMA_20'] = df['Close'].rolling(window=20).mean()
    df['SMA_50'] = df['Close'].rolling(window=50).mean()

    # Relative Strength Index (RSI)
    delta = df['Close'].diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
    rs = gain / loss
    df['RSI'] = 100 - (100 / (1 + rs))

    df.fillna(0, inplace=True) # Fill NaNs resulting from rolling operations

    return df

def load_and_process_data_with_gap(ticker, start_date, end_date,
    handle_nan='drop',
    split_method='date', test_ratio=0.2,
    scale_data=True,
    save_local=False, load_local=False,
    local_dir='stock_data',
    feature_columns=None, gap_period='30D'):
    """
    Load and process stock market data with a gap between training and testing periods.
    This function fetches the stock data, processes it (e.g., handling NaNs, adding technical indicators),
    splits it into training and testing sets, and optionally scales the data.

    Parameters:
    - ticker (str): Stock ticker symbol.
    - start_date (str): Start date for data in 'YYYY-MM-DD' format.
    - end_date (str): End date for data in 'YYYY-MM-DD' format.
    - handle_nan (str): Method to handle NaN values ('drop', 'fill', 'none').
    - split_method (str): How to split the data ('date', 'random').
    - test_ratio (float): Ratio of the test set, if split method is
    """
```

```

'random'.
- scale_data (bool): Whether to scale the data.
- save_local (bool): Whether to save the data locally.
- load_local (bool): Whether to load data from a local file if
available.
- local_dir (str): Directory to save/load the data.
- feature_columns (list): List of columns to use as features.
- gap_period (str): Gap period (e.g., '30D' for 30 days) between the
training and testing periods.

Returns:
- X_train, X_test (np.ndarray): Training and testing feature matrices.
- y_train, y_test (np.ndarray): Training and testing target vectors.
- scaler (MinMaxScaler or None): The scaler object used for data
normalization, if scaling is enabled.
"""

# Load data from Yahoo Finance or local
if load_local and os.path.exists(os.path.join(local_dir,
f"{ticker}.csv")):
    df = pd.read_csv(os.path.join(local_dir, f"{ticker}.csv"),
index_col=0, parse_dates=True)
else:
    df = yf.download(ticker, start=start_date, end=end_date)
    if save_local:
        if not os.path.exists(local_dir):
            os.makedirs(local_dir)
        df.to_csv(os.path.join(local_dir, f"{ticker}.csv"))

# Handle NaN values
if handle_nan == 'drop':
    df.dropna(inplace=True)
elif handle_nan == 'fill':
    df.fillna(method='ffill', inplace=True)

# Add technical indicators to the DataFrame
df = add_technical_indicators(df)

# Determine features and target
if feature_columns is None:
    feature_columns = df.columns.tolist()
    feature_columns.remove('Close') # Assuming 'Close' is the target
by default

X = df[feature_columns].values
y = df['Close'].values

# Data splitting logic
if split_method == 'date':
    gap = pd.to_timedelta(gap_period).days
    test_start_idx = int(len(df) * (1 - test_ratio)) + gap
    X_train, X_test = X[:test_start_idx], X[test_start_idx:]
    y_train, y_test = y[:test_start_idx], y[test_start_idx:]
else:
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_ratio, shuffle=True, random_state=42)

# Data scaling
scaler = None
if scale_data:
    scaler = MinMaxScaler()

```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(y_train.reshape(-1, 1)).flatten()
y_test = y_scaler.transform(y_test.reshape(-1, 1)).flatten()

return X_train, X_test, y_train, y_test, scaler
```