etherium

October 30, 2024

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.model_selection import train_test_split
```

[ ]:
```
# 1. Load Data
data = pd.read_csv('/content/drive/MyDrive/ETH-USD_5Y.csv')  # Replace with␣
↪your data file
data['Direction'] = np.where(data['Close'].shift(-1) > data['Close'], 1, 0)  #␣
↪Define direction (1=up, 0=down)
```

[ ]:
```
# 2. Data Preprocessing
features = data[['Open', 'High', 'Low', 'Close', 'Volume']].values
target = data['Direction'].values
```

[ ]:
```
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target,␣
↪test_size=0.2, random_state=42)
```

[ ]:
```
# 3. Reshape data for LSTM (if using LSTM, otherwise skip)
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

[ ]:
```
# 4. Build the Model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1],␣
↪X_train.shape[2])))
model.add(LSTM(50))
model.add(Dense(1, activation='sigmoid'))  # Binary classification (up or down)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
```

```
layer in the model instead.
  super().__init__(**kwargs)
```

[ ]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
# 5. Train the Model
history = model.fit(X_train, y_train, epochs=10, batch_size=32,␣
 ↪validation_data=(X_test, y_test))
```

```
Epoch 1/10
7/7             3s 69ms/step -
accuracy: 0.5508 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 2/10
7/7             0s 10ms/step -
accuracy: 0.4225 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 3/10
7/7             0s 10ms/step -
accuracy: 0.4278 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 4/10
7/7             0s 7ms/step -
accuracy: 0.4264 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 5/10
7/7             0s 8ms/step -
accuracy: 0.4302 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 6/10
7/7             0s 7ms/step -
accuracy: 0.4131 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 7/10
7/7             0s 7ms/step -
accuracy: 0.4105 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 8/10
7/7             0s 10ms/step -
accuracy: 0.4140 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 9/10
7/7             0s 11ms/step -
accuracy: 0.3784 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
Epoch 10/10
7/7             0s 14ms/step -
accuracy: 0.4074 - loss: nan - val_accuracy: 0.4906 - val_loss: nan
```

[ ]:
```
# 6. Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
2/2             0s 8ms/step -
accuracy: 0.4937 - loss: nan
Accuracy: 49.06%
```

[ ]:
```
# 7. Make Predictions
predictions = model.predict(X_test)
predicted_direction = (predictions > 0.5).astype(int)
```

```
2/2             1s 377ms/step
```