# bitcoin

October 30, 2024

[9]:
```
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
from tensorflow.keras.optimizers import SGD
```

[10]:
```
# 1. Load Data
data = pd.read_csv('/content/drive/MyDrive/all_currencies.csv')  # Replace with␣
 ↪your data file
data['Direction'] = np.where(data['Close'].shift(-1) > data['Close'], 1, 0)  #␣
 ↪Define direction (1=up, 0=down)
```

[11]:
```
# 2. Data Preprocessing
features = data[['Open', 'High', 'Low', 'Close', 'Volume']].values
target = data['Direction'].values
```

[12]:
```
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)
```

[13]:
```
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target,␣
 ↪test_size=0.2, random_state=42)
```

[14]:
```
# 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]))
```

[15]:
```
# 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)
```

[16]:

```python
optimizer = SGD(learning_rate=0.01, momentum=0.9)  # You can adjust the learning rate if necessary
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

[17]:

```python
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2, verbose=1)
```

```
Epoch 1/10
12916/12916            71s 5ms/step
- accuracy: 0.5351 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 2/10
12916/12916            72s 6ms/step
- accuracy: 0.5342 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 3/10
12916/12916            68s 5ms/step
- accuracy: 0.5341 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 4/10
12916/12916            73s 6ms/step
- accuracy: 0.5345 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 5/10
12916/12916            82s 6ms/step
- accuracy: 0.5347 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 6/10
12916/12916            83s 6ms/step
- accuracy: 0.5333 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 7/10
12916/12916            82s 6ms/step
- accuracy: 0.5356 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 8/10
12916/12916            79s 6ms/step
- accuracy: 0.5356 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 9/10
12916/12916            75s 6ms/step
- accuracy: 0.5352 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
Epoch 10/10
12916/12916            91s 7ms/step
- accuracy: 0.5347 - loss: nan - val_accuracy: 0.5350 - val_loss: nan
```

[18]:

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

```
4037/4037            12s 3ms/step -
```

```
accuracy: 0.5337 - loss: nan
Accuracy: 53.27%
```

[19]:

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

```
4037/4037             11s 3ms/step
```