Final Report

Ling-En Huang

Introduction

Project Overview:

Develop models to predict stock prices and the direction of price movement.

Objective:

Predict actual stock prices.

Predict the direction of stock price movement (up or down).

Methodology

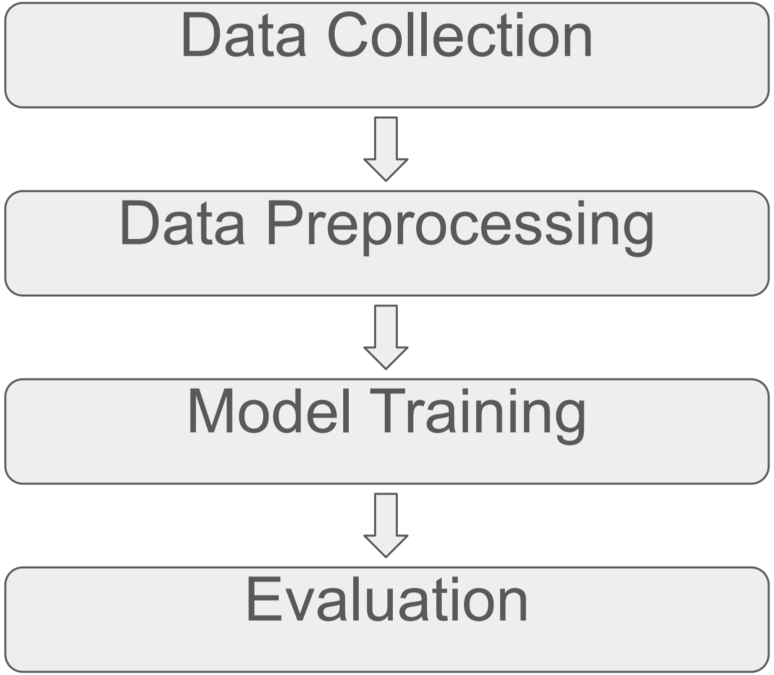
Approach:

- Data Collection: Historical stock prices, market indicators.
- Data Preprocessing: Masks, data selection
- Model Selection: Various machine learning models for prediction.

Tools and Frameworks:

- Python, PyTorch, Pandas, Numpy, Sklearn, Mplfinance, Pandas_ta.

Workflow


Data Collection
Data Preprocessing
Model Training
Evaluation

Data collection

Experimental Setup

Data Sources: 5 min SPY stock market data from Alpha Vantage

Data preprocessing: labeling data, filter data, normalization, etc.

Datasets:

5 min SPY data from 2018 to 2023

Labeling data

Average True Range [1] (ATR: is a technical indicator that measures how volatile a stock has been over past 14 bars)

$$\left(\frac{1}{n}\right)\sum_{i}^{n} \mathrm{TR}_i$$

where:

TR_i = Particular true range, such as first day's TR, then second, then third

n = Number of periods

$$\mathrm{TR} = \mathrm{Max}\,[(\mathrm{H} - \mathrm{L}), |\mathrm{H} - \mathrm{C}_p|, |\mathrm{L} - \mathrm{C}_p|]$$

where:

H = Today's high

L = Today's low

C_p = Yesterday's closing price

Max = Highest value of the three terms

so that:

$(\mathrm{H} - \mathrm{L})$ = Today's high minus the low

$|\mathrm{H} - \mathrm{C}_p|$ = Absolute value of today's high minus yesterday's closing price

$|\mathrm{L} - \mathrm{C}_p|$ = Absolute value of today's low minus yesterday's closing price

Labeling data

Label the candle as

+1 (Green) if the candle reaches **current closed price + 1 * ATR first**

-1 (Red) if the candle reaches **current closed price - 1 * ATR first**

Filter data (First mask)

1. Average bar range (ABR: average bar range for past 10 bars.):

    Highlight the Bar(with green and red color) that is larger than the 1.05 * ABR

Training / Testing data with the first mask (large bar)

Training : 2020, 2021 SPY 5 min data

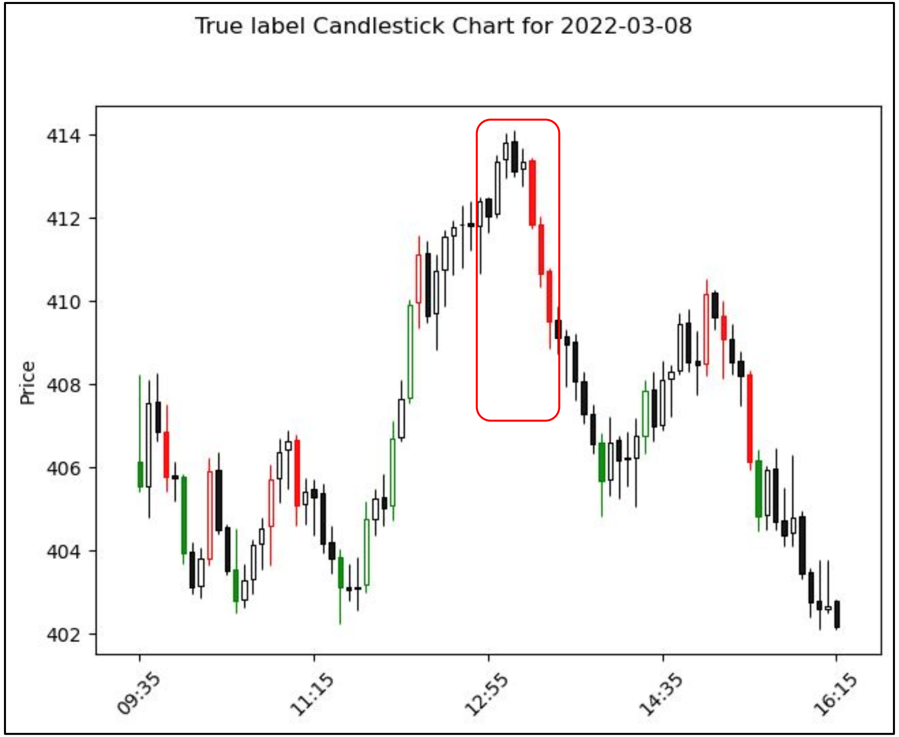
Testing : 2022, 2023 SPY 5 min data selected 50 days from this period.

Total: 1453

1	-1
750 (51.62%)	703 (48.38%)

Filter data (Second mask)

2. Non-overlap candle size is >= the whole candle size * 0.4



Testing Data with the second mask distribution

Total: 834

1	-1
420 (50.36%)	414 (49.64%)

Create features and labels (1)

Sequence length: 20, 30, 40, 50, 60

Take sequence = 10 for example:

Features

Open	High	Low	Close
241.9180	242.3220	241.5500	242.0110
242.2080	242.4310	241.8940	242.0350
242.2260	242.3490	241.7760	241.9090
242.0990	242.2770	241.7310	241.9490
242.1400	242.4940	241.7810	242.1800
242.3660	242.6930	242.0560	242.2970
242.4860	242.7380	242.1830	242.4240
242.6150	242.7200	242.2280	242.3430
242.5420	242.7750	242.2460	242.4150
242.2080	242.4310	241.8940	242.0350

Label = +1 or -1

Create features and labels (2)

Open	High	Low	Close
241.9180	242.3220	241.5500	242.0110
242.2080	242.4310	241.8940	242.0350
242.2260	242.3490	241.7760	241.9090
242.0990	242.2770	241.7310	241.9490
242.1400	242.4940	241.7810	242.1800
242.3660	242.6930	242.0560	242.2970
242.4860	242.7380	242.1830	242.4240
242.6150	242.7200	242.2280	242.3430
242.5420	242.7750	242.2460	242.4150
242.2080	242.4310	241.8940	242.0350

Difference *10
or
Difference *100

Features

Open	High	Low	Close
NaN	NaN	NaN	NaN
2.90	1.09	3.44	0.24
0.18	-0.82	-1.18	-1.26
-1.27	-0.72	-0.45	0.40
0.41	2.17	0.50	2.31
2.26	1.99	2.75	1.17
1.20	0.45	1.27	1.27
1.29	-0.18	0.45	-0.81
-0.73	0.55	0.18	0.72
0.68	-0.18	0.36	-0.05

Label = +1 or -1

Normalization

Standard Scaler from sklearn

```
scaler = StandardScaler()
for i in range(features.shape[0]):
    features[i] = scaler.fit_transform(features[i])
```

Min Max

```
scaler = MinMaxScaler()
for i in range(features.shape[0]):
    features[i] = scaler.fit_transform(features[i])
```

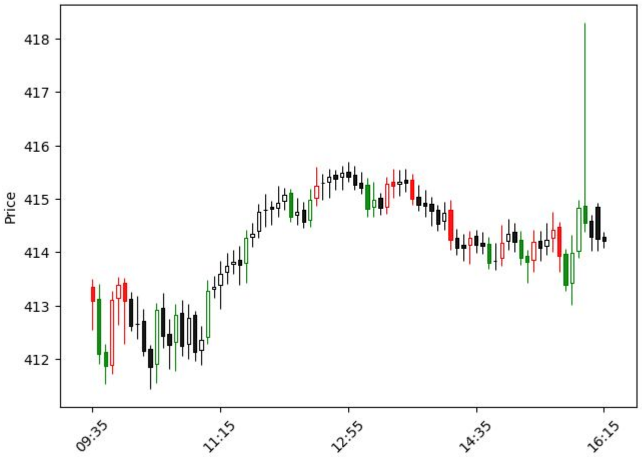
Do normalization to each individual columns.

Open	High	Low	Close
241.9180	242.3220	241.5500	242.0110
242.2080	242.4310	241.8940	242.0350
242.2260	242.3490	241.7760	241.9090
242.0990	242.2770	241.7310	241.9490
242.1400	242.4940	241.7810	242.1800
242.3660	242.6930	242.0560	242.2970
242.4860	242.7380	242.1830	242.4240
242.6150	242.7200	242.2280	242.3430
242.5420	242.7750	242.2460	242.4150
242.2080	242.4310	241.8940	242.0350

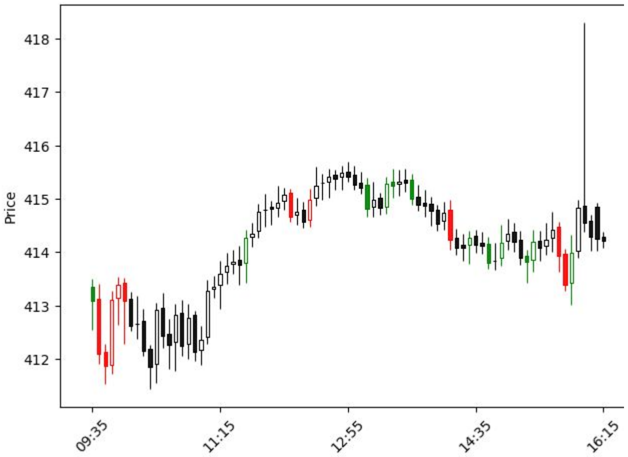
Metrics

Accuracy : when model make decisions, the right or wrong of that decisions.

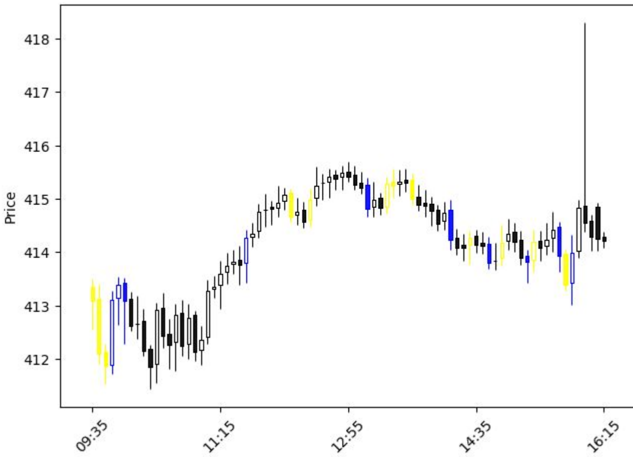
True label Candlestick Chart for 2021-07-08T00:00:00.000000000



Predicted Candlestick Chart for 2021-07-08T00:00:00.000000000



Predicted accuracy Chart for 2021-07-08T00:00:00.000000000



Models used

Long Short-Term Memory(LSTM)

Transformer

LSTM model design

```python
import torch
import torch.nn as nn

class LSTMModel(nn.Module):
    def __init__(self, hidden_dim=256, num_layers=2):
        super().__init__()
        self.input_projection = nn.Linear(4, hidden_dim // 4)
        self.act = nn.ReLU()
        self.r = nn.LSTM(hidden_dim // 4, hidden_dim, num_layers, batch_first=True, bidirectional=False, dropout=0.1)
        self.l = nn.Linear(1*num_layers*hidden_dim, 1)
        self.init_weights()

    def init_weights(self):
        for name, param in self.r.named_parameters():
            if 'bias' in name:
                nn.init.constant_(param, 0.0)
            elif 'weight_ih' in name:
                nn.init.kaiming_normal_(param)
            elif 'weight_hh' in name:
                nn.init.orthogonal_(param)

    def forward(self, x):
        batch_size = x.shape[0]
        x = self.act(self.input_projection(x))
        lstm_out, (h_n, c_n) = self.r(x)
        x = h_n.permute(1, 0, 2).flatten(1)
        x = self.l(x)
        return x
```

LSTM Hyper-parameters

Learning rate	Epochs	Input dim	Hidden dim	Output dim	hidden_layer	dropout	seq_length	Batch size
0.001	100	4	128	1	256	0.1	20~60	128

Adding the second mask (LSTM)

Seq_len	20	30	40	50	60
Training Accuracy	99.87%	99.98%	99.27%	99.40%	96.37%
Testing Accuracy(>=0.5)	51.80% T:834	49.28% T:834	49.88% T:834	53.36% T:834	48.80% T:834
Testing Accuracy(0.9, 0.1)	51.80% T:666	50.38% T:657	49.70% T:656	53.39% T:635	50.08% T:623
Testing Accuracy(0.8, 0.2)	51.99% T:729	50.28% T:718	49.37% T:711	52.91% T:722	49.64% T:695
Testing Accuracy(0.7, 0.3)	51.94% T:772	50.33% T:765	49.34% T:762	52.99% T:768	48.92% T:744

Change normalization method_MinMax

Seq_len	20	30	40	50	60
Training Accuracy	99.97%	99.41%	99.90%	98.79%	99.92%
Testing Accuracy(>=0.5)	51.80% T:834	49.40% T:834	50.60% T:834	52.40% T:834	51.44% T:834
Testing Accuracy(0.9, 0.1)	52.43% T:637	49.67% T:614	52.19% T:663	52.07% T:653	50.52% T:673
Testing Accuracy(0.8, 0.2)	52.79% T:716	50.51% T:691	51.73% T:723	52.52% T:714	50.89% T:731
Testing Accuracy(0.7, 0.3)	52.66% T:771	50.55% T:732	51.24% T:769	51.87% T:750	51.15% T:780

LSTM difference x 10 (without normalization)

Seq_len	20	30	40	50	60
Training Accuracy	100%	99.85%	98.96%	100%	99.96%
Testing Accuracy(>=0.5)	48.56% T:834	51.52% T:834	46.64% T:834	47.84% T:834	52.40% T:834
Testing Accuracy(0.9, 0.1)	49.93% T:699	50.15% T:688	48.07% T:622	47.34% T:676	52.77% T:667
Testing Accuracy(0.8, 0.2)	49.87% T:744	50.95% T:738	47.26% T:694	47.22% T:737	52.33% T:730
Testing Accuracy(0.7, 0.3)	48.97% T:778	51.73% T:779	46.52% T:748	47.48% T:775	52.39% T:773

LSTM difference x 100 (without normalization)

Seq_len	20	30	40	50	60
Training Accuracy	99.95%	99.95%	100%	99.87%	99.98%
Testing Accuracy(>=0.5)	51.32% T:834	51.68% T:834	50.48% T:834	52.52% T:834	49.64% T:834
Testing Accuracy(0.9, 0.1)	52.06% T:607	51.42% T:636	49.75% T:599	51.86% T:644	48.84% T:649
Testing Accuracy(0.8, 0.2)	51.89% T:688	50.93% T:701	49.63% T:679	52.83% T:725	48.95% T:711
Testing Accuracy(0.7, 0.3)	51.82% T:743	51.52% T:757	49.46% T:736	52.93% T:765	49.09% T:744

LSTM difference x10 no mask (without normalization)

Seq_len	20	30	40	50	60
Training Accuracy	96.75%	96.90%	97.68%	97.81%	96.70%
Testing Accuracy(>=0.5)	47.84% T:834	49.76% T:834	47.96% T:834	51.92% T:834	51.92% T:834
Testing Accuracy(0.9, 0.1)	48.31% T:650	49.85% T:670	47.96% T:663	51.88% T:640	49.62% T:663
Testing Accuracy(0.8, 0.2)	48.21% T:728	49.38% T:723	47.45% T:725	51.62% T:709	51.71% T:731
Testing Accuracy(0.7, 0.3)	47.77% T:762	49.35% T:764	47.23% T:775	52.05% T:757	51.95% T:770

LSTM difference x 10 standard scaler

Seq_len	20	30	40	50	60
Training Accuracy	100%	100%	99.98%	99.98%	100%
Testing Accuracy(>=0.5)	50.12% T:834	51.56% T:834	50.24% T:834	53.36% T:834	52.28% T:834
Testing Accuracy(0.9, 0.1)	50.44% T:678	50.36% T:693	51.35% T:668	52.58% T:679	50.98% T:663
Testing Accuracy(0.8, 0.2)	49.93% T:735	50.74% T:747	50.70% T:716	53.00% T:734	51.90% T:738
Testing Accuracy(0.7, 0.3)	49.81% T:775	51.33% T:787	50.65% T:768	53.16% T:775	51.86% T:781

LSTM difference x 10 min max

Seq_len	20	30	40	50	60
Training Accuracy	99.60%	97.19%	97.83%	99.25%	95.69%
Testing Accuracy(>=0.5)	50.24% T:834	49.40% T:834	52.76% T:834	49.04% T:834	49.88% T:834
Testing Accuracy(0.9, 0.1)	51.32% T:604	47.54% T:568	53.31% T:514	50.09% T:569	50.39% T:614
Testing Accuracy(0.8, 0.2)	50.95% T:685	48.13% T:669	52.26% T:616	49.63% T:671	50.92% T:597
Testing Accuracy(0.7, 0.3)	50.73% T:749	48.83% T:727	53.69% T:704	49.66% T:727	50.45% T:672

Adding the fifth feature:Volume, and Training data

Training : 2018, 2018, 2020, 2021 SPY 5 min data

Testing : 2022, 2023 SPY 5 min data selected 50 days from this period.

Learning rate	**Epochs**	Input dim	Hidden dim	Output dim	hidden_layer	dropout	seq_length	Batch size
0.001	**150**	4	128	1	256	0.1	20~60	128

Adding the fifth feature:Volume, and Training data

Seq_len	20	30	40	50	60
Training Accuracy	100%	99.30%	99.90%	98.79%	99.92%
Testing Accuracy(>=0.5)	48.08% T:834	50.48% T:834	50.60% T:834	52.40% T:834	51.44% T:834
Testing Accuracy(0.9, 0.1)	47.72% T:679	52.82% T:655	52.19% T:663	52.07% T:653	50.52% T:673
Testing Accuracy(0.8, 0.2)	47.80% T:728	52.26% T:718	51.73% T:723	52.52% T:714	50.89% T:731
Testing Accuracy(0.7, 0.3)	47.59% T:767	51.23% T:771	51.24% T:769	51.87% T:750	51.15% T:780

Transformer model design

```
class EncoderOnlyTransformerModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers, num_heads, dropout):
        super(EncoderOnlyTransformerModel, self).__init__()
        self.seq_len = sequence_length
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.positional_encoding = PositionalEncoding(hidden_dim, max_len=self.seq_len)
        self.encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_dim, nhead=num_heads, dropout=dropout)
        self.transformer_encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=num_layers)
        self.fc2 = nn.Linear(hidden_dim*self.seq_len, output_dim)

    def create_mask(self, seq_len, device):
        mask = nn.Transformer.generate_square_subsequent_mask(self=self, sz=seq_len).to(device)
        return mask

    def forward(self, src):
        src = self.fc1(src)  # Linear layer
        src = src.permute(1, 0, 2)  # Change shape to (seq_len, batch_size, hidden_dim)
        src = self.positional_encoding(src) # Apply positional encoding
        attention_mask = self.create_mask(src.size(0), src.device)
        output = self.transformer_encoder(src, mask=attention_mask)
        output = output.permute(1, 0, 2)  # Change shape back to (batch_size, seq_len, hidden_dim)
        output = output.reshape(output.size(0), -1)  # Flatten the sequence (batch_size, seq_len * hidden_dim)
        output = self.fc2(output)  # Classification layer
        return output
```

Attention mask [2]

```python
class EncoderOnlyTransformerModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers, num_heads, dropout):
        super(EncoderOnlyTransformerModel, self).__init__()
        self.seq_len = sequence_length
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.positional_encoding = PositionalEncoding(hidden_dim, max_len=self.seq_len)
        self.encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_dim, nhead=num_heads, dropout=dropout)
        self.transformer_encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=num_layers)
        self.fc2 = nn.Linear(hidden_dim*self.seq_len, output_dim)

    def create_mask(self, seq_len, device):
        mask = nn.Transformer.generate_square_subsequent_mask(self=self, sz=seq_len).to(device)
        return mask

    def forward(self, src):
        src = self.fc1(src)  # Linear layer
        src = src.permute(1, 0, 2)  # Change shape to (seq_len, batch_size, hidden_dim)
        src = self.positional_encoding(src) # Apply positional encoding
        attention_mask = self.create_mask(src.size(0), src.device)
        output = self.transformer_encoder(src, mask=attention_mask)
        output = output.permute(1, 0, 2)  # Change shape back to (batch_size, seq_len, hidden_dim)
        output = output.reshape(output.size(0), -1)  # Flatten the sequence (batch_size, seq_len * hidden_dim)
        output = self.fc2(output)  # Classification layer
        return output
```

Hyper-parameters

Learning rate	Epochs	Input dim	Hidden dim	Output dim	num_layers	num_heads	dropout	seq_length	Batch size
0.0001	500	4	128	1	1	4	0.1	20~60	64

Accuracy

Seq_len	20	30	40	50	60
Training Accuracy	98.75%	98.46%	98.40%	98.01%	97.51%
Testing Accuracy(>=0.5)	51.62% T:1453	48.93% T:1453	50.38% T:1453	50.58% T:1453	51.34% T:1453
Testing Accuracy(0.9, 0.1)	50.49% T:822	49.36% T:934	50.78% T:961	51.22% T:945	49.90% T:960
Testing Accuracy(0.8, 0.2)	51.77% T:1016	49.05% T:1105	50.72% T:1118	51.02% T:1125	51.09% T:1149
Testing Accuracy(0.7, 0.3)	51.70% T:1178	49.15% T:1237	50.80% T:1246	50.61% T:1235	51.54% T:1267

Hyper-parameters (adding more layers and dropout)

Learning rate	Epochs	Input dim	Hidden dim	Output dim	**num_layer**	num_heads	**dropout**	seq_length	Batch size
0.0001	500	4	128	1	**2**	4	**0.5**	20~60	64

Transformer with the first mask

Seq_len	20	30	40	50	60
Training Accuracy	82.20%	83.28%	87.63%	88.24%	90.27%
Testing Accuracy(>=0.5)	48.73% T:1453	50.38% T:1453	52.03% T:1453	51.20% T:1453	52.58% T:1453
Testing Accuracy(0.9, 0.1)	53.25% T:246	45.16% T:279	52.21% T:385	50.00% T:418	49.68% T:475
Testing Accuracy(0.8, 0.2)	50.72% T:483	47.78% T:519	53.24% T:633	52.71% T:683	51.85% T:756
Testing Accuracy(0.7, 0.3)	51.48% T:744	49.35% T:774	52.68% T:913	52.49% T:924	51.98% T:1010

Ensemble

Seq_len	20	30	40	50	60
Output probability	a	b	c	d	e

Ensemble probability
(a+b+c+d+e) / 5

Ensemble with the first mask

layer	1 layer	2 layer
Testing Accuracy(>=0.5)	50.79% T:1453	50.03% T:1453
Testing Accuracy(0.9, 0.1)	55.56% T:36	33.33% T:6
Testing Accuracy(0.8, 0.2)	51.37% T:146	40.43% T:47
Testing Accuracy(0.7, 0.3)	51.72% T:435	53.21% T:218
Testing Accuracy(0.6, 0.4)	49.37% T:869	50.43% T:698

Transformer with the second mask

Seq_len	20	30	40	50	60
Training Accuracy	92.02%	93.24%	95.16%	95.93%	96.37%
Testing Accuracy(>=0.5)	49.76% T:834	50.48% T:834	50.48% T:834	50.12% T:834	50.00% T:834
Testing Accuracy(0.9, 0.1)	48.86% T:307	47.51% T:301	51.16% T:346	48.98% T:392	49.68% T:433
Testing Accuracy(0.8, 0.2)	49.43% T:441	46.60% T:470	52.09% T:503	50.54% T:556	50.09% T:581
Testing Accuracy(0.7, 0.3)	49.74% T:571	49.07% T:593	50.08% T:613	50.84% T:653	50.52% T:673

Transformer Hyper-parameters

Learning rate	Epochs	Input dim	Hidden dim	Output dim	num_layers	num_heads	dropout	seq_length	Batch size
0.0001	2000	4	128	1	2	4	0.5	20~60	64

Transformer difference x 10 no mask, standard scaler

Seq_len	20	30	40	50	60
Training Accuracy	51.41%	51.29%	52.34%	51.46%	51.91%
Testing Accuracy(>=0.5)	51.20% T:834	46.16% T:834	50.12% T:834	49.88% T:834	50.24% T:834
Testing Accuracy(0.9, 0.1)	51.69% T:561	46.15% T:572	47.11% T:588	47.47% T:554	49.40% T:583
Testing Accuracy(0.8, 0.2)	51.45% T:653	45.84% T:661	48.13% T:669	49.22% T:644	50.00% T:666
Testing Accuracy(0.7, 0.3)	50.83% T:720	46.08% T:714	49.04% T:732	51.76% T:732	49.66% T:727

Positional Encoding

```python
class PositionalEncoding(nn.Module):
    def __init__(self, hidden_dim, max_len=5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, hidden_dim)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, hidden_dim, 2).float() * (-math.log(10000.0) / hidden_dim))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]
        return x
```

Transformer difference x 10 no mask standard scaler

Seq_len	20	30	40	50	60
Training Accuracy	**91.31%**	**93.64%**	**92.91%**	**87.30%**	**90.11%**
Testing Accuracy(>=0.5)	50.00% T:834	51.08% T:834	52.76% T:834	49.64% T:834	50.72% T:834
Testing Accuracy(0.9, 0.1)	46.13% T:336	52.97% T:404	50.46% T:434	48.21% T:614	50.00% T:642
Testing Accuracy(0.8, 0.2)	47.44% T:489	52.21% T:544	52.88% T:573	48.77% T:689	50.21% T:719
Testing Accuracy(0.7, 0.3)	49.50% T:606	50.62% T:640	52.97% T:657	49.40% T:751	50.39% T:776

Adding more training data

Training : 2018, 2018, 2020, 2021 SPY 5 min data

```
paths = [
spy_data/spy_2018_rth_data.csv",
spy_data/spy_2019_rth_data.csv",
spy_data/spy_2020_rth_data.csv",
spy_data/spy_2021_rth_data.csv",
spy_data/spy_2022_rth_data.csv",
spy_data/spy_2023_rth_data.csv",
```

Transformer difference x 10 standard scaler
Adding more training data with two masks

Seq_len	20	30	40	50	60
Training Accuracy	99.99%	100%	100%	100%	100%
Testing Accuracy(>=0.5)	48.56% T:834	49.64% T:834	49.16% T:834	51.20% T:834	51.92% T:834
Testing Accuracy(0.9, 0.1)	50.51% T:489	50.77% T:518	49.91% T:537	51.26% T:593	52.48% T:585
Testing Accuracy(0.8, 0.2)	49.92% T:611	50.00% T:626	49.37% T:632	52.23% T:674	53.25% T:676
Testing Accuracy(0.7, 0.3)	49.78% T:691	49.29% T:700	49.22% T:709	51.82% T:741	52.89% T:745

Discussion

Models did not perform as expected:

When the training loss goes down, the testing loss goes up instead, which shows that the training data is too noisy for the model to find any pattern. (shown in next slide)

Possible Reasons:

High Market Noise: The stock market has significant noise, making it difficult to predict accurately.

Overfitting: Models might have overfitted the training data, failing to generalize well to new data.

Feature Selection: Potentially important features might have been overlooked for example, the sequence length.

Problems encountered

LSTM

```
epoch: 0 train loss 0.6890728155771891
epoch: 0 val loss 0.6940886772523714
New best model found at epoch 0 with val loss 0.6940886772523714
epoch: 1 train loss 0.6893789440393447
epoch: 1 val loss 0.6931033786826246
New best model found at epoch 1 with val loss 0.6931033786826246
epoch: 2 train loss 0.6894668489694595
epoch: 2 val loss 0.6925991989496186
New best model found at epoch 2 with val loss 0.6925991989496186
epoch: 3 train loss 0.6892596185207367
epoch: 3 val loss 0.6923506292771167
New best model found at epoch 3 with val loss 0.6923506292771167
epoch: 4 train loss 0.6888898173967998
epoch: 4 val loss 0.6922032880032156
New best model found at epoch 4 with val loss 0.6922032880032156
epoch: 5 train loss 0.688542636235555
epoch: 5 val loss 0.6924364059928834
epoch: 6 train loss 0.6882103641827901
epoch: 6 val loss 0.6922846779109925
epoch: 7 train loss 0.6879297544558843
epoch: 7 val loss 0.6927107248719283
epoch: 8 train loss 0.6875005096197129
epoch: 8 val loss 0.6927516310233769
epoch: 9 train loss 0.6870632092158
epoch: 9 val loss 0.6936417381594501
...
epoch: 98 val loss 3.152350272719316
epoch: 99 train loss 0.01031502036882254
epoch: 99 val loss 3.103265445063433
```

Transformer

```
Epoch [1/500], Train Loss: 0.7254
epoch: 1 val loss 0.7013076060400234
New best model found at epoch 0 with val loss 0.7013076060400234
Epoch [2/500], Train Loss: 0.7063
epoch: 2 val loss 0.7028362441250658
Epoch [3/500], Train Loss: 0.7072
epoch: 3 val loss 0.7054279150925283
Epoch [4/500], Train Loss: 0.7059
epoch: 4 val loss 0.7061669225767842
Epoch [5/500], Train Loss: 0.7009
epoch: 5 val loss 0.7083898510519914
Epoch [6/500], Train Loss: 0.7006
epoch: 6 val loss 0.7119343027355164
Epoch [7/500], Train Loss: 0.6990
epoch: 7 val loss 0.7111155465831906
Epoch [8/500], Train Loss: 0.6992
epoch: 8 val loss 0.7124045448979055
Epoch [9/500], Train Loss: 0.6974
epoch: 9 val loss 0.711611934534208
Epoch [10/500], Train Loss: 0.7004
epoch: 10 val loss 0.7144954570635097
Epoch [11/500], Train Loss: 0.6981
epoch: 11 val loss 0.7163006663322449
Epoch [12/500], Train Loss: 0.6945
epoch: 12 val loss 0.7166589925608297
...
Epoch [499/500], Train Loss: 0.3065
epoch: 499 val loss 1.5637622470930805
Epoch [500/500], Train Loss: 0.3200
```

Discussion cont.

Learned:

Use of different deep learning models and techniques: for this project, I learned how to use pytorch with LSTM, Transformer. Besides, also learned transformer techniques like look ahead mask, making the time series prediction without the data leakage, positional encoding, making the model stronger so that can converge in training data.

Some strategies for daily traders: how to filter candles, how to get the relatively useful information in the candle chart.

Conclusion

I believe the problem is the highly randomness in stock market that cause the project challenging.

Future Work and Possible Improvements (information from one of my friend who is quantitative researcher from Points72):

Data filtering: using some independent indicators, different time dimension to find some useful information. However, this requires a lot of domain knowledge and data resource.

Investment target: In some unpopular investment target, which not so many hedge funds companies paying attention to, might have some chance to use deep learning method to get some benefits. However, if the target is unpopular, it might have problems of bid-ask spread which should be paid attention to.

Acknowledgement

Thanks to:

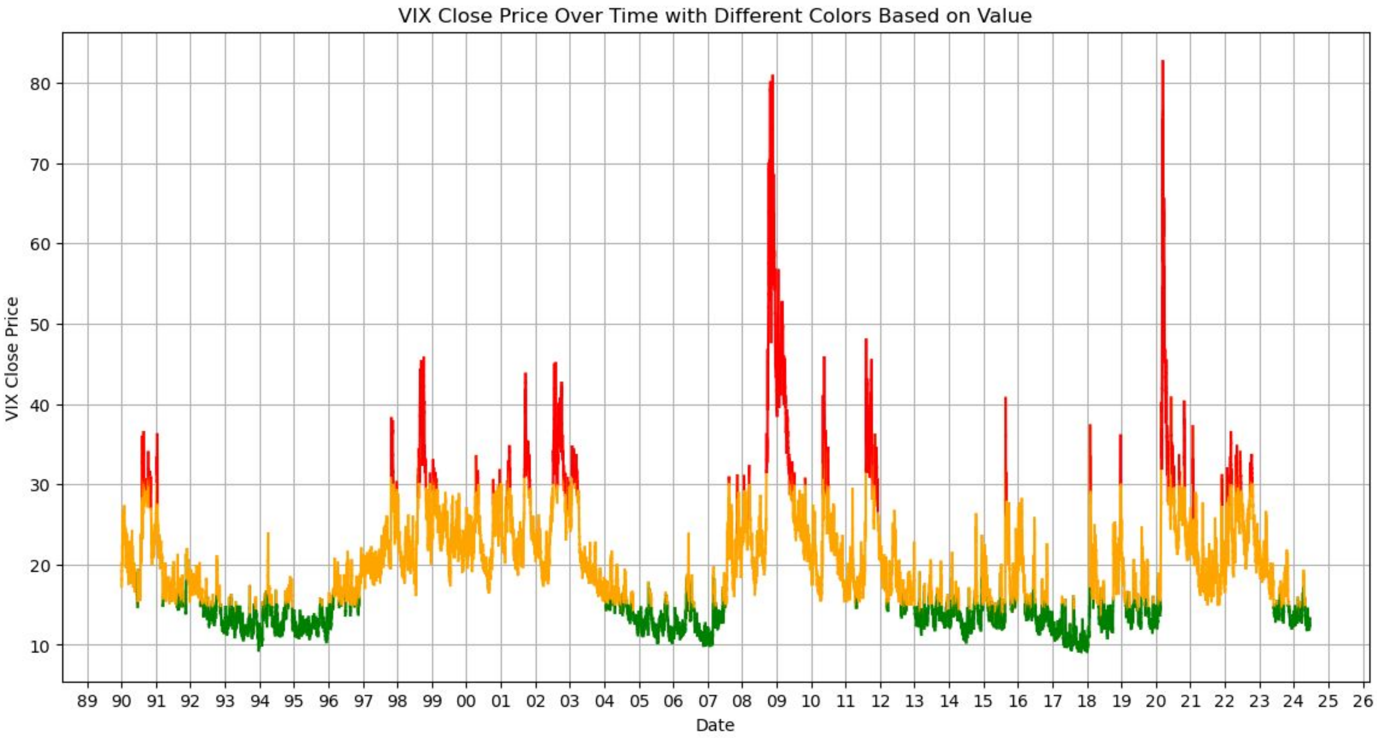
Supervisors: Prof. Marios Savvides, Ph.D. Han Zhang

Collaborators: Ph.D. Han Zhang

Appendix

How I sample the testing data

Using VIX distribution



Data set

5 minutes data of SPY from 2020 to 2023

Training data set: 2020

Testing data sampled from 2021 to 2023 with the VIX distribution

VIX < 15 = low VIX

15 <= VIX < 30 = mid VIX

VIX > 30 = high VIX

2021 to 2023 distribution

	Count	Percentage
Mid VIX	634	71.396396
Low VIX	199	22.409910
High VIX	55	6.193694

Autoregressive models' issues in stock market [3]

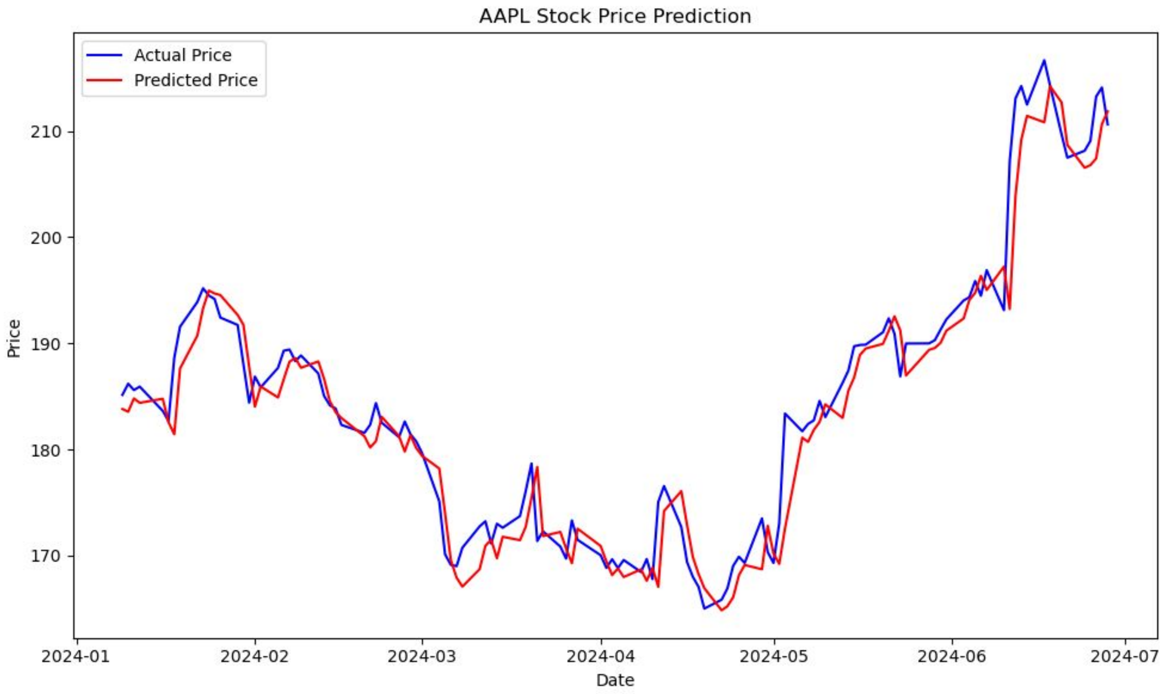
Normalization method design: using the futers testing data to do the Min Max, it's like you know the future's upper bound and lower bound and feed those info to the model.

Still need more discuss on how to do the normalization.

```python
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# Prepare data for LSTM
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data[['Close', 'SMA_20', 'EMA_20', 'RSI', 'MACD']])

# Function to create input sequences
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length), :])
        y.append(data[i + seq_length, 0])
    return np.array(X), np.array(y)

# Set sequence length and create train/test sets
seq_length = 1

X, y = create_sequences(scaled_data, seq_length)
train_size = int(len(X) * 0.8)

X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Result

There is a serious delay with the predicted price, and it's just like simply draw the line after the actual price.



Reference

[1]ATR https://www.investopedia.com/terms/a/atr.asp

[2]Attention mask https://medium.com/analytics-vidhya/understanding-attention-in-transformers-models-57bada0cce3e

[3]Autoregessive model with LSTM https://www.linkedin.com/feed/update/urn:li:activity:7224362320712364032?utm_source=share&utm_medium=member_desktop