1 2022/6/27

1.1 Trading opportunities and costs

1.1.1 Moving Average crossover

Table 1: Close Price Comparison via MA Crossover

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-01-18	2271.889893	2017-01-30	2280.899902
2017-04-28	2384.199951	2017-05-01	2388.330078
2017-09-05	2457.850098	2017-09-08	2461.429932
2018-03-16	2752.010010	2018-03-22	2643.689941
2018-05-14	2730.129883	2018-05-18	2712.969971

1.1.2 Bollinger Bands

Table 2: Close Price Comparison via BB

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-03-21	2344.020020	2017-03-27	2341.590088
2017-04-13	2328.949951		
2017-06-29	2419.699951		
2017-07-06	2409.750000		
2017-08-10	2438.209961		
2017-08-17	2430.010010	2017-08-22	2452.510010
2018-02-05	2648.939941		
2018-02-08	2581.000000	2018-02-08	2581.000000
2018-03-22	2643.689941	2018-03-27	2612.620117
2018-06-27	2699.629883		
2018-10-10	2785.679932	2018-10-12	2767.129883
2018-10-24	2656.100098	2018-10-29	2641.250000
2018-12-17	2545.939941		
2018-12-19	2506.959961		

1.1.3 Moving average convergence diver

Table 3: Close Price Comparison via MACD

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-01-04	2270.750000		
2017-01-11	2275.320068	2017-01-17	2267.889893
2017-01-24	2280.070068		
2017-04-24	2374.149902	2017-04-26	2387.449951
2017-05-25	2415.070068	2017-05-30	2412.909912
2017-06-19	2453.459961		
2017-07-13	2447.830078	2017-07-18	2460.610107
2017-08-31	2471.649902	2017-08-09	2474.020020
		2017-09-06	2465.540039
2017-11-08	2594.379883		
2017-11-28	2627.040039	2017-11-28	2627.040039
2018-01-04	2723.989990		
2018-02-23	2747.300049	2018-02-27	2744.280029
2018-03-05	2720.939941	2018-03-09	2786.570068
2018-04-10	2656.870117	2018-04-16	2677.840088
2018-05-07	2672.629883		
2018-06-04	2746.870117		
2018-07-09	2784.169922	2018-07-12	2798.290039
2018-08-06	2850.399902	2018-08-09	2853.580078
2018-08-24	2874.689941	2018-08-29	2914.040039
2018-09-20	2930.750000		
2018-11-02	2723.060059	2018-11-07	2813.889893
2018-11-28	2743.790039	2018-12-03	2790.370117

1.2 Differences

1.2.1 Pure data (EMA/SMA)

start='2016-10-21'
rolling function and drop first fifty-one lines which have Nan values.
Using ewm function (adjust=False) to calculate EMA

1.2.2 Data prepossessing

MinMaxScaler() –Original closing price (columns:price, index:date)
MinMaxScaler() –Moving average price (columns:SMA/EMA, index:date)

train_test_split(data,test_size=0.2, random_state=1)

1.2.3 CNN

kernel_constraint=max_norm(max_norm_value):
if the L2-Norm of weights exceeds , scale whole weight matrix by a factor that reduces the norm to

kernel_initializer:
Weights are responsible for connection between the units
In He Uniform Initialization weights belong to uniform distribution in range as shown below

$$W \approx U(a,b)$$

$$a = -\sqrt{\frac{6}{f\_in + f\_out}}, \qquad b = \sqrt{\frac{6}{f\_in + f\_out}}$$

Figure 1: He Uniform Initialization

Conv1DTranspose

1.2.4 SVM

x_train_encode: the output without being inverted(401,1)
r: using output without being inverted

x_train: same
return:put same

1 to 1

2 2022/6/28

2.1 import data

1. download data from yfinance: start='2016-10-21',end='2019-1-1'
   - GSPC
   - Adj Close
   - start='2016-10-21'
   - end='2019-1-1'
   - index: date
   - column: adj close
2. SMA
   - for loop
   - .rolling().mean() function: default parameters
   - .dropna(): delete the fifty-one lines
   - after deleting, start='2017-01-03'('2017-01-01')
3. EMA
   - for loop
   - .ewm(adjust=False).mean()
   - min_periods: default=0
4. data: index=date columns= price,SMA...,EMA....
5. noisy data
   - MinMaxScaler(price)
   - for loop
   - (shape: (100,502,1))
   - train_test_split(test_size=0.2)
   - noisy train: (80,502,1)
   - noisy test: (20,502,1)

6. pure data
    - MinMaxScaler(SMA....EMA....)
    - for loop
    - (shape: (100,502,1))
    - train_test_split(test_size=0.2)
    - pure train: (80,502,1)
    - pure test: (20,502,1)

2.2 Training and test Model

1. MODEL

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 500, 128)          512

 conv1d_1 (Conv1D)           (None, 498, 32)           12320

 conv1d_transpose (Conv1DTra  (None, 500, 32)          3104
 nspose)

 conv1d_transpose_1 (Conv1DT  (None, 502, 128)         12416
 ranspose)

 conv1d_2 (Conv1D)           (None, 502, 1)            385

=================================================================
```

Figure 2: CNN-Autoencoder

   (a) Encoding
      - (Conv1D(128, kernel_size=3, kernel_constraint=max_norm(2.0), activation='relu', kernel_initializer='he_uniform', input_shape=input_shape))
      - (Conv1D(32, kernel_size=3, kernel_constraint=max_norm(2.0), activation='relu', kernel_initializer='he_uniform'))

   (b) Decoding
      - (Conv1DTranspose(32, kernel_size=3, kernel_constraint=max_norm(2.0), activation='relu', kernel_initializer='he_uniform'))
      - (Conv1DTranspose(128, kernel_size=3, kernel_constraint=max_norm(2.0), activation='relu', kernel_initializer='he_uniform'))

   (c) Output layer

- Conv1D(1, kernel_size=3, kernel_constraint=max_norm(2.0), activation='sigmoid', padding='same')

(d) Compile parameters

- optimizer='adam'
- loss='binary_crossentropy'
  Generally used for multi-category tasks

(e) Fit

- epochs = 20
- batch_size = 10

2. Reconstructions

- Using one line of noisy test(scaled)
- Using scaler_X which is fitted by price to inverse price and reconstructed price
- shuffle = False

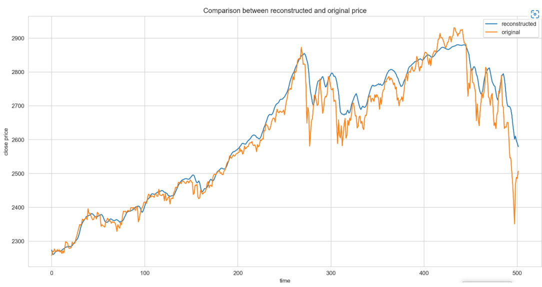

Figure 3: shuffle = False

- shuffle = True

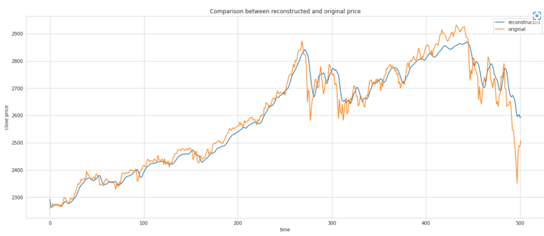

Figure 4: shuffle = True

2.3 SVM

1. F1 for original prices

    - Using scaled data to calculate log return and label the data. Using 80% to split train and test dataset without shuffling.
    - Using first 80% percentage of scaled data as train dataset.
    - X_train, Y_trains' shapes (401,1) (401,)
    - X_test, Y_test' shapes (101,1) (101,)

2. F1 for reconstructed prices

    - Same, but using reconstructed prices before being inversed

3. Shuffle = False

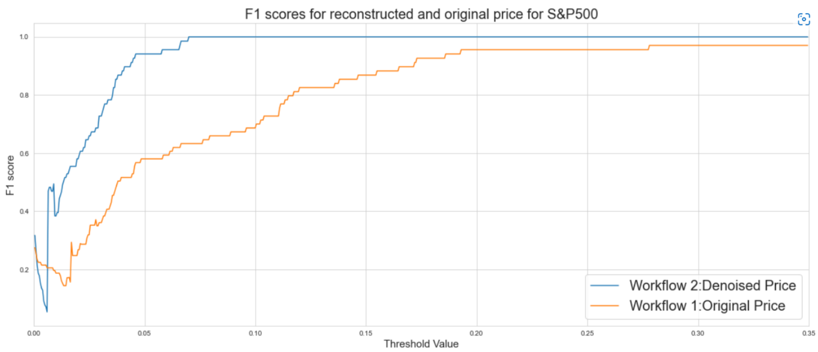

Figure 5: shuffle = False

4. Shuffle = True

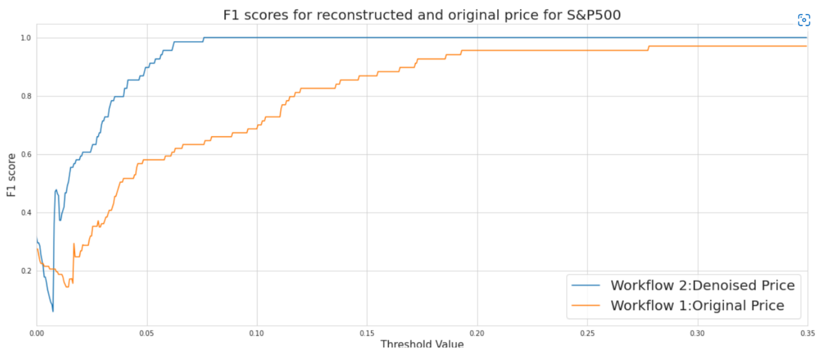

Figure 6: shuffle = True

2.4 Trading indicators

1. MA

    - Shuffle = True

Table 4: Close Price Comparison via MA Crossover

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-01-18	2271.889893	2017-01-30	2280.899902
2017-04-28	2384.199951	2017-05-01	2388.330078
2017-09-05	2457.850098	2017-09-08	2461.429932
2018-03-16	2752.010010	2018-03-22	2643.689941
2018-05-14	2730.129883	2018-05-18	2712.969971

    - Shuffle = False

Table 5: Close Price Comparison via MA Crossover

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-01-18	2271.889893	2017-01-19	2263.689941
2017-04-28	2384.199951	2017-05-02	2391.169922
2017-09-05	2457.850098	2017-09-08	2461.429932
2018-03-16	2752.010010		
2018-05-14	2730.129883	2018-05-17	2720.129883

2. BB

- Shuffle = True

Table 6: Close Price Comparison via BB

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-03-21	2344.020020	2017-03-27	2341.590088
2017-04-13	2328.949951		
2017-06-29	2419.699951		
2017-07-06	2409.750000		
2017-08-10	2438.209961		
2017-08-17	2430.010010	2017-08-22	2452.510010
2018-02-05	2648.939941		
2018-02-08	2581.000000	2018-02-08	2581.000000
2018-03-22	2643.689941	2018-03-27	2612.620117
2018-06-27	2699.629883		
2018-10-10	2785.679932	2018-10-12	2767.129883
2018-10-24	2656.100098	2018-10-29	2641.250000
2018-12-17	2545.939941		
2018-12-19	2506.959961		

- Shuffle = False

Table 7: Close Price Comparison via BB

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-03-21	2344.020020	2017-03-24	2343.979980
2017-04-13	2328.949951		
2017-06-29	2419.699951		
2017-07-06	2409.750000	2017-07-06	2409.750000
2017-08-10	2438.209961		
2017-08-17	2430.010010	2017-08-22	2452.510010
2018-02-05	2648.939941		
2018-02-08	2581.000000	2018-02-08	2581.000000
2018-03-22	2643.689941	2018-03-26	2658.550049
2018-06-27	2699.629883		
2018-10-10	2785.679932	2018-10-11	2728.370117
2018-10-24	2656.100098		
2018-12-17	2545.939941	2018-12-20	2467.419922
2018-12-19	2506.959961		

3. MACD

- Shuffle = True

Table 8: Close Price Comparison via MACD

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-01-04	2270.750000		
2017-01-11	2275.320068	2017-01-17	2267.889893
2017-01-24	2280.070068		
2017-04-24	2374.149902	2017-04-26	2387.449951
2017-05-25	2415.070068	2017-05-30	2412.909912
2017-06-19	2453.459961		
2017-07-13	2447.830078	2017-07-18	2460.610107
2017-08-31	2471.649902	2017-08-09	2474.020020
		2017-09-06	2465.540039
2017-11-08	2594.379883		
2017-11-28	2627.040039	2017-11-28	2627.040039
2018-01-04	2723.989990		
2018-02-23	2747.300049	2018-02-27	2744.280029
2018-03-05	2720.939941	2018-03-09	2786.570068
2018-04-10	2656.870117	2018-04-16	2677.840088
2018-05-07	2672.629883		
2018-06-04	2746.870117		
2018-07-09	2784.169922	2018-07-12	2798.290039
2018-08-06	2850.399902	2018-08-09	2853.580078
2018-08-24	2874.689941	2018-08-29	2914.040039
2018-09-20	2930.750000		
2018-11-02	2723.060059	2018-11-07	2813.889893
2018-11-28	2743.790039	2018-12-03	2790.370117

- Shuffle = False

Table 9: Close Price Comparison via MACD

Date	Buy with Original Signals	Date	Buy with Original Signals
2017-01-04	2270.750000	2017-01-09	2268.899902
2017-01-11	2275.320068		
2017-01-24	2280.070068		
2017-04-24	2374.149902	2017-04-26	2387.449951
2017-05-25	2415.070068	2017-05-31	2411.800049
2017-06-19	2453.459961		
2017-07-13	2447.830078	2017-07-18	2460.610107
2017-08-31	2471.649902	2017-09-06	2465.540039
2017-11-08	2594.379883		
2017-11-28	2627.040039	2017-11-28	2627.040039
2018-01-04	2723.989990	2018-01-05	2743.149902
2018-02-23	2747.300049	2018-02-27	2744.280029
2018-03-05	2720.939941		
2018-04-10	2656.870117	2018-04-16	2677.840088
2018-05-07	2672.629883		
2018-06-04	2746.870117		
2018-07-09	2784.169922	2018-07-11	2774.020020
2018-08-06	2850.399902		
2018-08-24	2874.689941	2018-08-28	2897.520020
2018-09-20	2930.750000		
2018-11-02	2723.060059	2018-11-07	2813.889893
2018-11-28	2743.790039	2018-11-30	2760.169922

3 2022/6/29

3.1 Import data

1. **Dataset:** Tabular Playground Series - Mar 2022(https://www.kaggle.com/code/ambrosm/tpsmar22-eda-which-makes-sense/data)

2. There are 12 roadways, 8 directions and 65 combinations of roadway with direction. This means that on average, a roadway has between 5 and 6 directions. The code below shows this for the training data; the test data has the same geography. There are no missing values here.

row id	time	x	y	direction	congestion
0	1991-04-01	0	0	EB	70
1	1991-04-01	0	0	NB	49

Table 10: dataset head

3. **Time:** There are 13059 time values in the training data. As 13059 * 65 = 848835, i.e. the length of the train dataframe, we know that at every point in time, the congestion is known for all 65 roadways.

4. Choose one road(EB00)
 13059 rows $\times$ 1 columns

time	congestion
1991-04-01	70
1991-04-01	70

Table 11: EB00 head

3.2 CNN model

1. **Split the train and test for XGBoost regression**
train_size = int(len(EB00) * 0.8)
train: 10447 rows $\times$ 1 columns(For CNN model)
test: 2612 rows $\times$ 1 columns

2. **Noisy data**

- Data: train
- Repeat 100 times
- delete first 51 columns
- Shape: 100 rows $\times$ 10396 columns
- Raw: SMA/EMA
- Columns: date

3. **Pure data**
    - Data: train
    - talib.SMA/EMA to calculate (range:(2,52))
    - delete first 51 columns which have Nan values
    - Shape: 100 rows $\times$ 10396 columns
    - Raw: SMA/EMA
    - Columns: date

4. **MinMaxScaler()**
    - fit_transform(EB00_train_noisy)
    - **Shape:**: (100, 10396, 1)
    - fit_transform(EB00_train_pure)
    - **Shape:**: (100, 10396, 1)

5. **train_test_split**
    - test_size=0.2
    - Shuffle = True
    - train_train (80, 10396, 1)

6. **Model structure**
    -

7. **Prediction**
    (a) EB00_train
        - MinMaxScaler()–fit_MM

- predict
- inverse(using MM)

(a) EB00_test

- MinMaxScaler()–fit_MM
- predict
- inverse(using MM)

3.3 XGBRegressor

1. EB00_train

   (a) Denoised to Denoised

   - Get denoised data by predicting EB00_train
   - Generate feature matrix using denoised data(shift)
   - Using TimeSeriesSplit(5)
   - Using the feature to regression the denoised data
   - Calculate average MSE
   - Before transfer back: 0.00086107798885202646
   - After: 8.209789848327636

   (b) Denoised to Original

   - Get denoised data by predicting EB00_train
   - Generate feature matrix using denoised data(shift)
   - Using TimeSeriesSplit(5)
   - Using the feature to regression the original data
   - Calculate average MSE
   - Before transfer back: 0.0159162247814175
   - After: 174.44955564114952

   (c) Original to Original

   - Generate feature matrix using original data EB00_train(shift)
   - Using TimeSeriesSplit(5)
   - Using the feature to regression the original data
   - Calculate average MSE
   - Before transfer back: 0.014528513660646O2

- After: 147.62879062077891

2. EB00_test

   (a) Denoised to Denoised

   - Get denoised data by predicting EB00_test
   - Generate feature matrix using denoised data(shift)
   - Using TimeSeriesSplit(5)
   - Using the feature to regression the denoised data
   - Calculate average MSE
   - Before transfer back:
   -

   (b) Denoised to Original

   - Get denoised data by predicting EB00_test
   - Generate feature matrix using denoised data(shift)
   - Using TimeSeriesSplit(5)
   - Using the feature to regression the original data
   - Calculate average MSE
   - Before transfer back:
   -

   (c) Original to Original

   - Generate feature matrix using original data EB00_test(shift)
   - Using TimeSeriesSplit(5)
   - Using the feature to regression the original data
   - Calculate average MSE
   - Before transfer back:
   -