TestStatic

April 13, 2020

[1]:

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error

from scipy import stats

import math

from datetime import datetime

import time

import warnings; warnings.simplefilter('ignore')
```

[2]:

```
# load the data
data = pd.read_csv('btc_ta.csv')
```

[3]:

```
# examine the features
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2638440 entries, 0 to 2638439
Data columns (total 74 columns):
Unnamed: 0                    int64
time                          int64
open                          float64
close                         float64
high                          float64
low                           float64
volume                        float64
volume_adi                    float64
volume_obv                    float64
volume_cmf                    float64
volume_fi                     float64
volume_em                     float64
```

```
volume_sma_em                float64
volume_vpt                   float64
volume_nvi                   float64
volatility_atr               float64
volatility_bbm               float64
volatility_bbh               float64
volatility_bbl               float64
volatility_bbw               float64
volatility_bbhi              float64
volatility_bbli              float64
volatility_kcc               float64
volatility_kch               float64
volatility_kcl               float64
volatility_kchi              float64
volatility_kcli              float64
volatility_dcl               float64
volatility_dch               float64
volatility_dchi              float64
volatility_dcli              float64
trend_macd                   float64
trend_macd_signal            float64
trend_macd_diff              float64
trend_ema_fast               float64
trend_ema_slow               float64
trend_adx                    float64
trend_adx_pos                float64
trend_adx_neg                float64
trend_vortex_ind_pos         float64
trend_vortex_ind_neg         float64
trend_vortex_ind_diff        float64
trend_trix                   float64
trend_mass_index             float64
trend_cci                    float64
trend_dpo                    float64
trend_kst                    float64
trend_kst_sig                float64
trend_kst_diff               float64
trend_ichimoku_a             float64
trend_ichimoku_b             float64
trend_visual_ichimoku_a      float64
trend_visual_ichimoku_b      float64
trend_aroon_up               float64
trend_aroon_down             float64
trend_aroon_ind              float64
trend_psar                   float64
trend_psar_up                float64
trend_psar_down              float64
trend_psar_up_indicator      float64
```

```
trend_psar_down_indicator    float64
momentum_rsi                 float64
momentum_mfi                 float64
momentum_tsi                 float64
momentum_uo                  float64
momentum_stoch               float64
momentum_stoch_signal        float64
momentum_wr                  float64
momentum_ao                  float64
momentum_kama                float64
momentum_roc                 float64
others_dr                    float64
others_dlr                   float64
others_cr                    float64
dtypes: float64(72), int64(2)
memory usage: 1.5 GB
```

[4]:

```
data.columns.to_series()[np.isinf(data).any()]
```

[4]:

```
trend_cci    trend_cci
dtype: object
```

[5]:

```
# create the target feature
data['nextClosingPrice'] = data['close'].shift(-1)

# drop the rows with 'None' in target column
data = data.dropna(subset=['nextClosingPrice'])
data = data.drop(['trend_psar_down', 'trend_psar_up', 'trend_cci'], axis=1)

# drop na values from feature extraction
data = data.dropna()

data['time'] = pd.to_datetime(data['time'], unit='ms')

data = data.reset_index()
```

[6]:

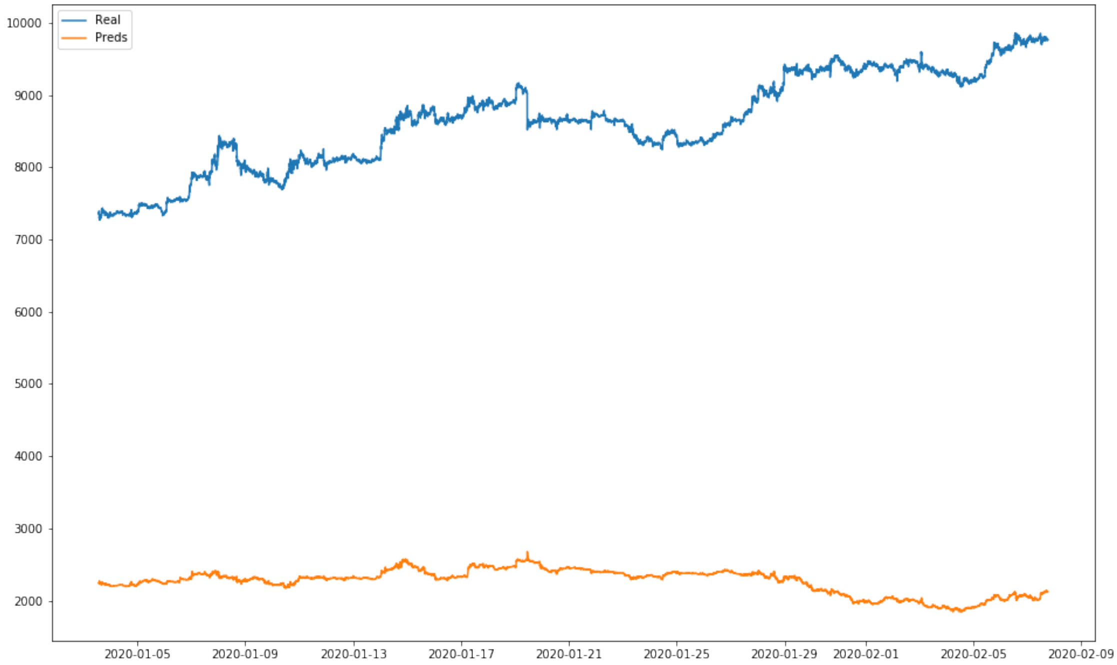
```
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from lightgbm import LGBMRegressor
from sklearn.svm import SVR
```

[7]:

```
def testModel(df, cutOff, modelName):

    # lists to store data, will concat to make result data frame
    rmseList = []
    predList = []
    realList = []
    predTimeList = []
    trainingTimeList = []

    # extract feature and test data
    X = df.drop(['Unnamed: 0', 'time', 'nextClosingPrice'], axis=1)
    y = df['nextClosingPrice']

    # split the data
    X_train, X_test = X[:cutOff], X[cutOff:]
    y_train, y_test = y[:cutOff], y[cutOff:]
    X_test, y_test = X_test.iloc[::2], y_test[::2]
    timeList = df['time'][cutOff:][::2]

    # start timer
    startTime = time.time()

    # create a new model
    if modelName == 'dt':
        model = DecisionTreeRegressor()
    elif modelName == 'xgb':
        model = XGBRegressor()
    elif modelName == 'lgbm':
        model = LGBMRegressor()
    elif modelName == 'lr':
        model = LinearRegression()
    elif modelName == 'ridge':
        model = Ridge(alpha=0.01)
    elif modelName == 'svr':
        model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)


    # train the model
    model.fit(X_train, y_train)

    # make a prediction
    y_pred = model.predict(X_test)

    endTime = time.time()

    # record time figures for result data frame
    trainingTime = endTime - startTime

    # result dictionary
    result_data = pd.DataFrame({'Timestamp': timeList,
                                "Real": y_test,
                                "Preds": y_pred,
                                "Difference": abs(y_test - y_pred)})

    print("")
    print("RMSE:{}".format(math.sqrt(mean_squared_error(y_test, y_pred))))
    print("Time to train:{}".format(trainingTime))

    # result plot
    plt.figure(figsize=(16,10))
    plt.plot('Timestamp', 'Real', data=result_data)
    plt.plot('Timestamp', 'Preds', data=result_data)
    plt.legend()
    plt.show()

    return result_data
```

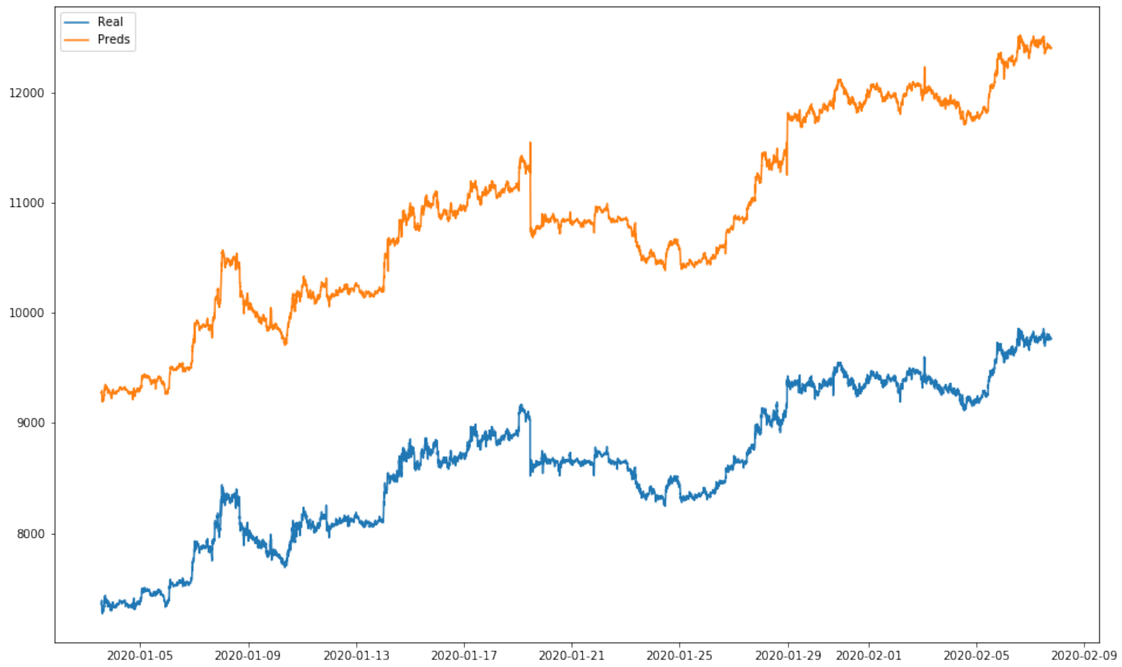
[8]:
```
lr_results = testModel(data, -50000, 'lr')
```

```
RMSE:6410.374610975496
Time to train:8.574902296066284
```



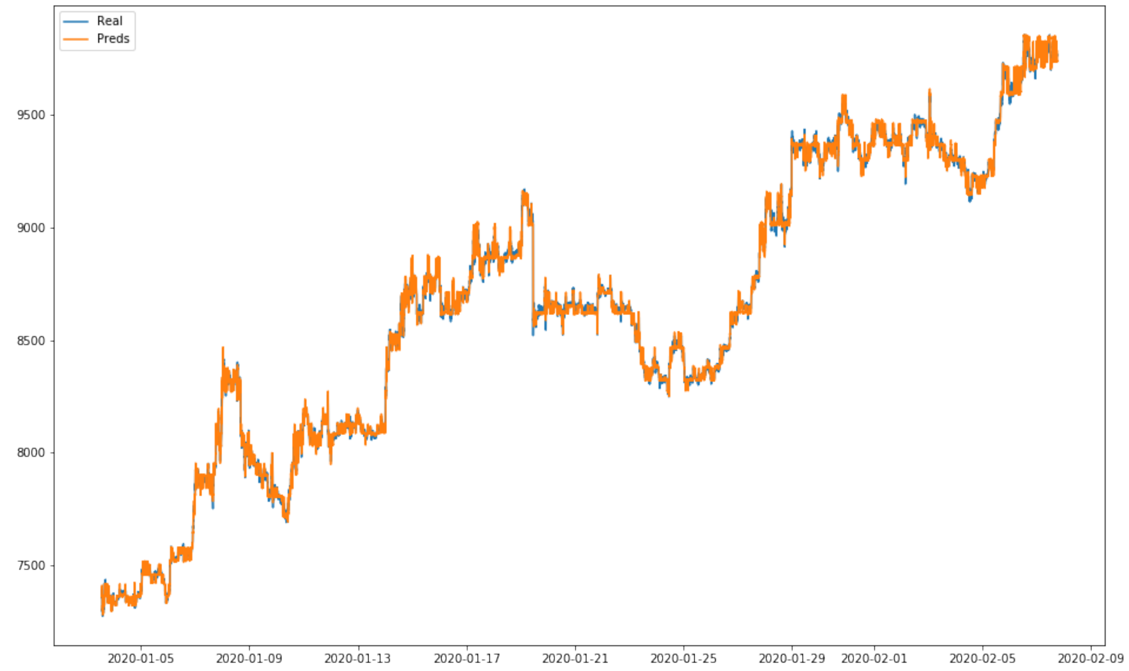
[9]:
```
ridge_results = testModel(data, -50000, 'ridge')
```

```
RMSE:2264.809269912674
Time to train:12.939939737319946
```



```
[10]: lgbm_results = testModel(data, -50000, 'lgbm')
```

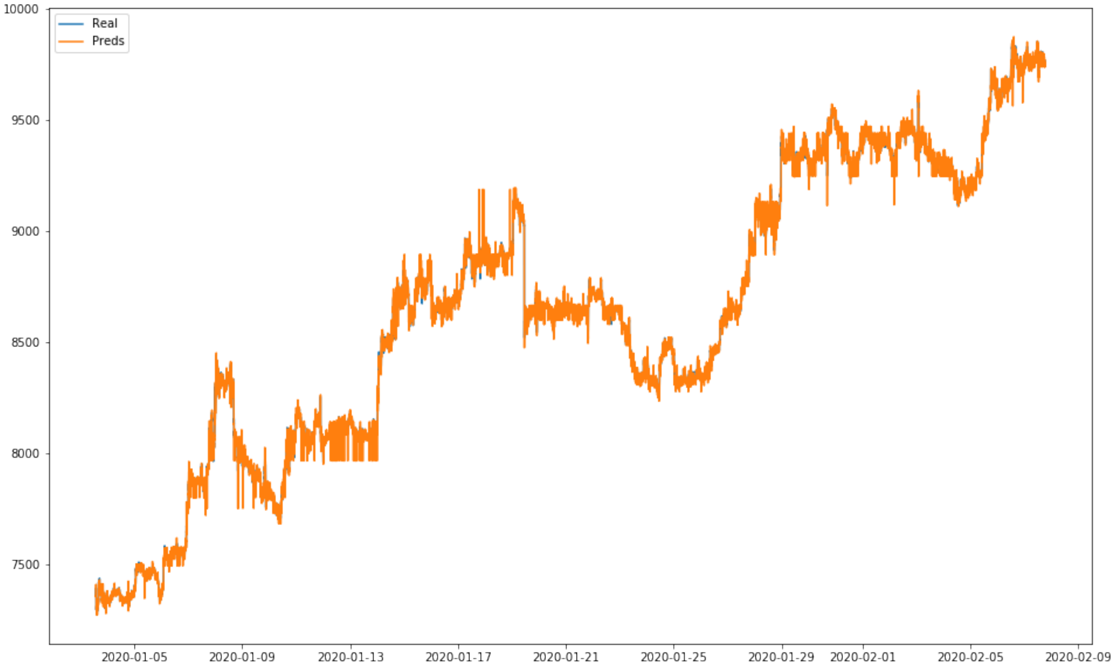
```
RMSE:19.003842236272593
Time to train:45.10588192939758
```



[11]:

```
dt_results = testModel(data, -50000, 'dt')
```

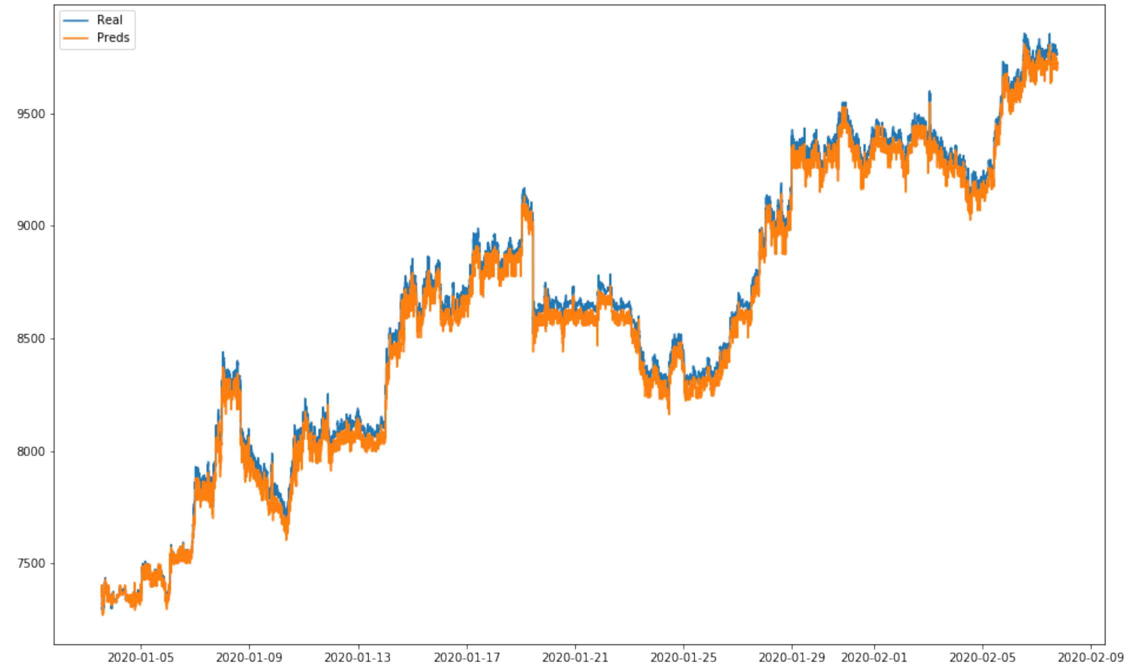
```
RMSE:22.891907136266546
Time to train:524.5222373008728
```



[12]:

```
xgb_results = testModel(data, -50000, 'xgb')
```

```
RMSE:51.40155509483932
Time to train:1550.1348748207092
```



```
[13]: pred_results = pd.DataFrame({'Timestamp': dt_results.Timestamp,
                                   'Real': dt_results.Real,
                                   'Dt_preds': dt_results.Preds,
                                   'Lgbm_preds': lgbm_results.Preds,
                                   'Xgb_preds': xgb_results.Preds,
                                   'Lr_preds': lr_results.Preds,
                                   'Ridge_preds': ridge_results.Preds})
```

```
[14]: import seaborn as sns
```

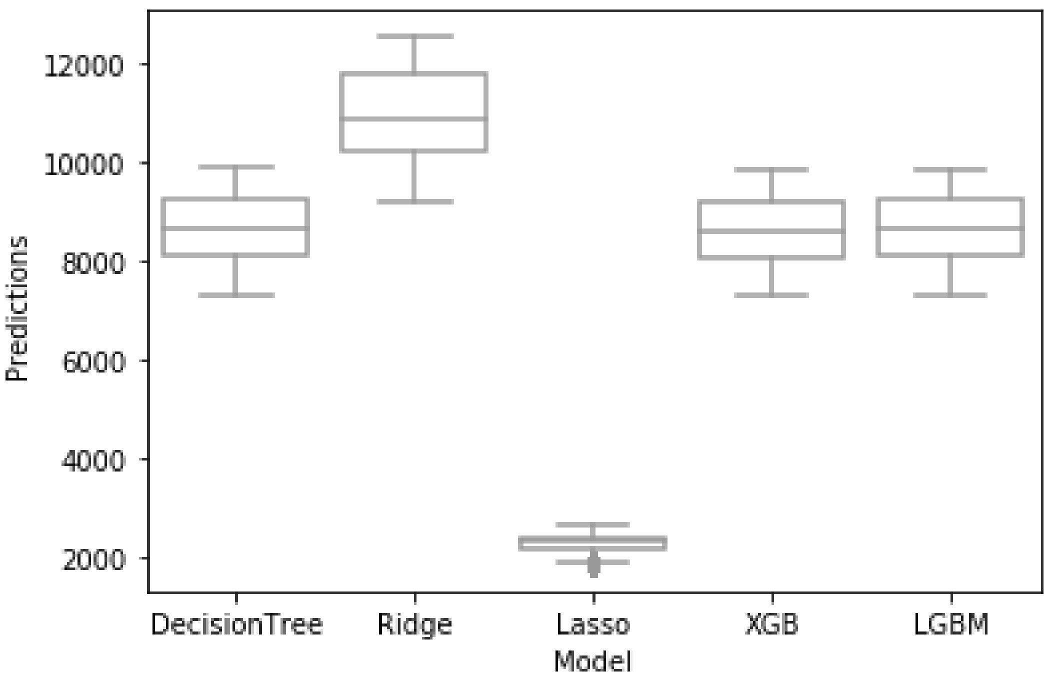
```
[15]: pred_results.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 2588113 to 2638111
Data columns (total 7 columns):
Timestamp      25000 non-null datetime64[ns]
Real           25000 non-null float64
Dt_preds       25000 non-null float64
Lgbm_preds     25000 non-null float64
Xgb_preds      25000 non-null float32
Lr_preds       25000 non-null float64
Ridge_preds    25000 non-null float64
dtypes: datetime64[ns](1), float32(1), float64(5)
memory usage: 1.2 MB
```

[16]:

```
display(pred_results.describe())

# Dataset:
a = pd.DataFrame({ 'Model' : np.repeat('DecisionTree',25000), 'Predictions':␣
 ↪pred_results.Dt_preds})
b = pd.DataFrame({ 'Model' : np.repeat('Ridge',25000), 'Predictions':␣
 ↪pred_results.Ridge_preds})
c = pd.DataFrame({ 'Model' : np.repeat('Lasso',25000), 'Predictions':␣
 ↪pred_results.Lr_preds})
d = pd.DataFrame({ 'Model' : np.repeat('XGB',25000), 'Predictions':␣
 ↪pred_results.Xgb_preds})
e = pd.DataFrame({ 'Model' : np.repeat('LGBM',25000), 'Predictions':␣
 ↪pred_results.Lgbm_preds})
# f = pd.DataFrame({ 'Model' : np.repeat('SVR',25000), 'Predictions':␣
 ↪pred_results.Svr_preds}

df=a.append(b).append(c).append(d).append(e)

# Usual boxplot
sns.boxplot(x='Model', y='Predictions', color='w', data=df)
plt.show()
```

```
               Real      Dt_preds    Lgbm_preds     Xgb_preds      Lr_preds  \
count  25000.000000  25000.000000  25000.000000  25000.000000  25000.000000
mean    8634.667031   8633.456115   8634.390758   8587.858398   2266.902934
std      647.442883    649.581945    649.267621    640.513000    170.879815
min     7275.000000   7272.600000   7279.750887   7271.422363   1846.205134
25%     8124.000000   8126.825000   8126.288991   8069.465454   2179.790624
50%     8650.000000   8653.894321   8624.104211   8596.123047   2312.853699
75%     9260.000000   9246.700000   9251.735517   9201.047607   2382.702386
max     9855.200000   9875.000000   9855.771909   9808.571289   2677.271965

        Ridge_preds
count  25000.000000
mean   10888.463127
std      860.828201
min     9191.698836
25%    10215.161950
50%    10842.470083
75%    11776.571253
max    12515.812328
```


12000
10000
8000
6000
4000
2000
Predictions
DecisionTree
Ridge
Lasso
XGB
LGBM
Model