Predicting Stock Return Volatility using Support Vector Regression

Carter Zhang, Gleb Dunaev, Guoyi Jiang, Lanxi Zhang

Fordham University

Abstract

In this project, we present our research process and detailed approach to predict S&P stocks daily return volatility using Support Vector Regression (SVR). We perform model hyperparameter optimization through manually randomization and, more systematically, GridSearch Cross Validation based on historical daily stock data in 2016 and 2017. We then predict the 20-day stock return volatility on the 2018 stock data on a daily rolling basis, which result in roughly 250 predictions. These predicted volatilities, along with the benchmark metric which is the T-1 realized volatilities, are compared with the real realized volatility to gauge their respective predictive power.

Scope and Assumptions

Given the diversified nature of S&P 500 index, we can treat the 500 as our proxy for the market. We use the daily data instead of the monthly data as we believe monthly data would be too generalized to produce a valid model. Additionally, as a common measure, we train out SVR model using 20-day stock (log)return volatility (aggregating day 1 to day 20) and make the prediction of the next day's 20-day return volatility (aggregating day 2 to day 21).

Project Design

1. Data

We first obtain stock historical daily data from 2016-01-01 to 2018-12-31 using from Yahoo Finance API in Python. We manually calculate the 20-day return volatility, the 20-day high/low adjusted close prices, and the 5-day high/low adjusted close prices. We then standardize the 20- and 5-day high/low by dividing these values by the adjusted close price. After these steps, the data frame as the following fields:

- **Basic Predictor Variable**: *Ticker, formatted_date, adjclose, daily_return, high_1m, low_1m, high_5d, low_5d, high_1m_pct, low_1m_pct, high_5d_pct, low_5d_pct*
- **Key Predicter Variable**: *ret20_vol*, the sample standard deviation of the stock's daily log return, from d_1 to d_{20}
- **Response Variable**: *ret20_vol_tmr*, same as *ret20_vol* of tomorrow, from d_2 to d_{21}

Finally, we connect to our local SQL database and load in the data.

2. Hyperparameter Optimization

There following are hyperparameters in the SVM model that are commonly optimized:

- $\boldsymbol{Kernel}$: Specifies the kernel type to be used in the algorithm, can be "linear", "poly", "rbf", "sigmoid". "rbf" stands for Radial Basis Function.
- $Gamma$: ["Scale", "auto"]. Kernel coefficient for "rbf", "poly" and "sigmoid".
- C: Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared L2 penalty.
- $Epsilon$: specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value.
- $Shrinking$: whether to use the shrinking Heuristic. (default = "True")

Notice the gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. The C parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy.

Based on our research, we decide to optimize "Kernel", "Gamma", "C", and "Epsilon" as they cover most of the variability in the model. We optimize these parameters two-fold: first by randomizing subsampling from the training sample, and second by performing GridSearch with 5-fold Cross Validation on each sampled subset. First, on the dates of the training set covering 2016 to 2017, we randomly select 15 dates $\{d_1, \dots d_{15}\}$ and perform GridSearchCV around these 15 dates, resulting in 15 different fitted SVR models. Each model is generated by the GridSearchCV algorithm from exhaustively trying every combination of the hyperparameters with 5-fold Cross Validation and keeping the combination that has the best result. Hence, each one of the 15 models potentially has different hyperparameters. To select the best one of the 15, the models are fitted on their following day's $(d_i + 1)_{i=1,\dots,15}$ data and predict on the next day's $(d_i + 2)_{i=1,\dots,15}$ return volatility. The model with the minimum MAE/MSE from their follow-on predictions is selected as the final model for prediction.

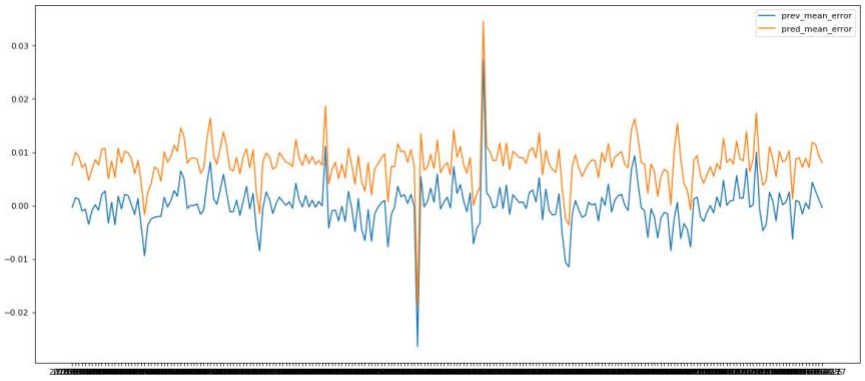
GridSearchCV is a commonly used method to test through the candidate parameters and select the optimal combination. From our online research and our previous project experience, we believe it is justifiable to use GridSearchCV when fitting on non-homogenous data that we have. Additionally, we assume there is low correlation across time hence the same fitted SVR model can be reapplied to predict on future data. Following this logic, we randomly select 15 dates from a range of approximately 250 trading days in 2016 and 2017 to generalize the GridSearch result.

3. Prediction

With a fitted model, we move forward each trading day in 2018 by predicting the next day's volatility based on today's *daily_return, ret20_vol* (an already aggregated metric of the 20-day stock return), *high_1m_pct, low_1m_pct, high_5d_pct, low_5d_pct*. For benchmarking, we use today's realized volatility as benchmark prediction. Through the 252 trading days of 2018 with roughly 500 stocks entries on each trading. On each day, the model predicts the next-day volatility for the 500 stocks and compare with the realized volatility on the next day. We keep track of the average prediction error and the prediction MSE for both our model's predictions and for the beachmark through the 252 trading days. At the end of 2018, we store the result into SQL database.

4. Result

We see that the model produces worse predictions compared to the benchmark $T-1$ realized volatility, but exhibits close (almost parallel) correlation with the benchmark. It roughly makes the same prediction as the benchmark does but at a shifted level. This result is not surprising as the model construction is not sophisticated and the data chose are hardly indicative of future volatility. Given the significant level of noise and variance in the market, the result from the SVR model aligns with our expectation.



5. Appendix: Python Code

5.1 Data Retrieval

```
#!/usr/bin/env python
# coding: utf-8

# Data Retrieval

import numpy as np
import pandas as pd
from yahoofinancials import YahooFinancials
import requests
import mysql.connector as mysql

from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://root:fordham@localhost:3306/project')
con = engine.connect()

symbol = pd.read_csv("symbol.csv")
tickers = list(symbol['Ticker'])
tickers.sort()
start_date = '2016-01-01'
end_date = '2018-12-31'
method = 'daily'

yahoo = YahooFinancials('AAPL')
x = yahoo.get_historical_price_data(start_date, end_date, method)
x['AAPL'].keys()


df = pd.DataFrame()
i = 0
for ticker in tickers:
    ticker = symbol.at[i,'Ticker']
    #check if ticker is valid
    link = requests.get('http://finance.yahoo.com/quote/' + ticker)
    if link.url.find("lookup") < 0:
        yahoo = YahooFinancials(ticker)
        price = yahoo.get_historical_price_data(start_date,end_date,method)
        #check if ticker has data
        if len(price[ticker])==6:
            #check if ticker has full historical daily prices between 2016 and 2018
            if (len(price[ticker]["prices"]) > 600): # 80% of trading days 2016-2018
                prices = pd.DataFrame(price[ticker]['prices'])
                n = len(prices)
                prices['daily_return'] = [np.log(prices['adjclose'][i]/prices['adjclose'][i-1])
if i > 0 else np.nan for i in range(n)]
                prices['ret20_vol'] = [np.std(prices['daily_return'][(i-19):(i+1)],ddof=19) if
(i >= 20) and (i+1 < n) else np.nan for i in range(n)]
                prices['ret20_vol_tmr'] = prices['ret20_vol'].shift(-1)
                prices['next_ret20_vol'] = [np.std(prices['daily_return'][(i+1):(i+21)], ddof=19)
if (i >= 0) and (i+21) < n else np.nan for i in range(n)]
                prices['high_1m'] = [np.max(prices['adjclose'][(i-19):i]) if i >= 19 else np.nan
for i in range(n)]
```

```
                prices['low_1m'] = [np.min(prices['adjclose'][(i-19):i]) if i >= 19 else np.nan
for i in range(n)]
                prices['high_5d'] = [np.max(prices['adjclose'][(i-4):i]) if i >= 4 else np.nan
for i in range(n)]
                prices['low_5d'] = [np.max(prices['adjclose'][(i-4):i]) if i >= 4 else np.nan for
i in range(n)]
                prices = prices.drop(columns=['date'])
                prices['Ticker'] = ticker
                df = df.append(prices[['Ticker', 'formatted_date', 'adjclose', 'daily_return',
'ret20_vol', 'ret20_vol_tmr', 'next_ret20_vol',                                    'high_1m',
'low_1m', 'high_5d', 'low_5d']], ignore_index = True)

    i += 1
    if (i%10 == 0):
        print(ticker)

df.to_excel("2016_2018_daily_data_v2.xlsx", index=False)
df.to_csv("2016_2018_daily_data_v2.csv", index=False)
print("\nDone.")
```

5.2 Main Project

```
#!/usr/bin/env python
# coding: utf-8


import pandas as pd
import numpy as np
import scipy.stats
import datetime
import random
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.svm import SVR
import mysql.connector as mysql
from sqlalchemy import create_engine

from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)



engine = create_engine('mysql+pymysql://root:fordham@localhost:3306/project')
con = engine.connect()

from sqlalchemy import event

def add_own_encoders(conn, cursor, query, *args):
    cursor.connection.encoders[np.float64] = lambda value, encoders: float(value)

event.listen(engine, "before_cursor_execute", add_own_encoders)


#data = pd.read_csv('project_data.csv')

query = ''' select * from project_data '''

data = pd.read_sql_query(query, con)

#data.drop(columns=['Unnamed: 0'], inplace=True)
data['high_1m_pct'] = data['high_1m']/data['adjclose']
data['low_1m_pct'] = data['low_1m']/data['adjclose']
data['high_5d_pct'] = data['high_5d']/data['adjclose']
data['low_5d_pct'] = data['low_5d']/data['adjclose']
data.head()

data = pd.DataFrame(data)
data.columns

dates = list(set(data['formatted_date']))
dates.sort()
#dates = list(map(lambda x:datetime.datetime.strptime(x, "%Y-%m-%d"), dates))

date2016D1 = "2016-01-01" #datetime.datetime.strptime('2016-01-01', '%Y-%m-%d')
date2017D1 = "2017-01-01" #datetime.datetime.strptime('2017-01-01', '%Y-%m-%d')
date2018D1 = "2018-01-01" #datetime.datetime.strptime('2018-01-01', '%Y-%m-%d')
```

```
dates2016 = list(filter(lambda x: (x >= date2016D1) & (x < date2017D1), dates))
dates2017 = list(filter(lambda x: (x >= date2017D1) & (x < date2018D1), dates))
dates2018 = list(filter(lambda x: x >= date2018D1, dates))


data2016 = data[data['formatted_date'].isin(dates2016)]
data2017 = data[data['formatted_date'].isin(dates2017)]
data2018 = data[data['formatted_date'].isin(dates2018)]
dataPre2018 = data[data['formatted_date'].isin(dates2016+dates2017)]


tickers = list(set(dataPre2018['Ticker']))
train_dates = list(set(dataPre2018['formatted_date']))
train_dates.sort()

# Generate 10 different 20-day snippets for training
train_dates_subset = [train_dates[i] for i in random.sample(list(range(len(train_dates)-1)), 15)]


from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
simplefilter(action='ignore', category=DeprecationWarning)

params = list()
models = list()
count = 0

for date in train_dates_subset:
    curr_date = date
    next_date = train_dates[train_dates.index(curr_date)+1]
    test_df = dataPre2018[dataPre2018['formatted_date']==next_date]
    train_df = dataPre2018[dataPre2018['formatted_date']==curr_date]

    X_train = np.array(train_df[['daily_return', 'ret20_vol', 'high_1m_pct', 'low_1m_pct',
'high_5d_pct', 'low_5d_pct']])

    X_test = np.array(test_df[['daily_return', 'ret20_vol', 'high_1m_pct', 'low_1m_pct',
'high_5d_pct', 'low_5d_pct']])

    y_train = np.array(train_df['ret20_vol_tmr'])
    y_test = np.array(test_df['ret20_vol_tmr'])


    # Set the parameters by cross-validation
    tuned_parameters = {'kernel': ('linear', 'rbf', 'poly'),
                        'C':[0.1, 5, 10],
                        'gamma': [1e-1, 1],
                        'epsilon':[0.01, 0.1, 1]}


    svr = SVR()
    clf = GridSearchCV(svr, tuned_parameters, scoring='neg_mean_squared_error', cv=5)
    clf.fit(X_train, y_train)

    y_true, y_pred = y_test, clf.predict(X_test)
    print("Best parameters set found on development set: {}".format(clf.best_params_))
```

```
    print("Mean Absolute Error on test set: %3f" % MAE(y_test, y_pred))
    print("Mean Squared Error on test set: %3f" % MSE(y_test, y_pred))

    param = clf.best_params_
    param['MAE'] = MAE(y_test, y_pred)
    param['MSE'] = MSE(y_test, y_pred)
    param['id'] = count
    models.append(clf)
    count += 1

    params.append(param)

print()
print("Done")


best_param_MAE = dict()
best_param_MSE = dict()
min_MSE = -1
min_MAE = -1
for param in params:
    if min_MSE < 0 and min_MAE < 0:
        min_MSE = param['MSE']
        min_MAE = param['MAE']
        best_param_MAE = best_param_MSE = param
        continue
    if param['MAE'] < min_MAE:
        min_MAE = param['MAE']
        best_param_MAE = param
    if param['MSE'] < min_MSE:
        min_MSE = param['MSE']
        best_param_MSE = param

print("Best parameter based on MAE from GridSearchCV:")
print(best_param_MAE)
print("Best parameter based on MSE from GridSearchCV:")
print(best_param_MSE)


best_model = models[best_param_MSE["id"]]

dates2017.sort()
dates2018.sort()

pred_data = data[data['formatted_date'].isin(dates2017[-1:] + dates2018)]
pred_data['prev_vol'] = pred_data['ret20_vol'].shift(1)
pred_dates = list(set(pred_data['formatted_date']))
pred_dates.sort()
n_dates = len(pred_dates)


result_df = pd.DataFrame(index=dates2018,

columns=['date','prev_mean_error','pred_mean_error','prev_MSE','pred_MSE'])

for i in range(n_dates-1):
    fit_date = pred_dates[i]
```

```
    pred_date = pred_dates[i+1]

    fit_df = pred_data[pred_data['formatted_date']==fit_date]
    test_df = pred_data[pred_data['formatted_date']==pred_date]
    common_stocks = set(fit_df['Ticker']).intersection(set(test_df['Ticker']))
    fit_df = fit_df[fit_df['Ticker'].isin(common_stocks)]
    test_df = test_df[test_df['Ticker'].isin(common_stocks)]
    X = np.array(fit_df[['daily_return', 'ret20_vol', 'high_1m_pct', 'low_1m_pct', 'high_5d_pct',
'low_5d_pct']])
    y_true = np.array(fit_df[['ret20_vol_tmr']])
    y_prev = np.array(fit_df[['ret20_vol']])
    y_pred = best_model.predict(X)
    prev_avg_error = np.mean(y_prev - y_true)
    pred_avg_error = np.mean(y_pred - y_true)
    prev_MSE = MSE(y_true, y_prev)
    pred_MSE = MSE(y_true, y_pred)

    result_df.loc[pred_date] = [pred_date, prev_avg_error, pred_avg_error, prev_MSE, pred_MSE]

result_df.index = list(range(len(result_df)))
display(result_df)


import matplotlib.pyplot as plt
fig=plt.figure(figsize=(18, 8), dpi= 80, facecolor='w', edgecolor='k')
plt.plot( 'date', 'prev_mean_error', data=result_df)
plt.plot( 'date', 'pred_mean_error', data=result_df)
plt.legend()
plt.show()


## drop_table_cmd = '''IF OBJECT_ID('Result') IS NOT NULL DROP TABLE Result; '''
# con.execute(drop_table_cmd)

#create_table_cmd = '''
#CREATE TABLE Result
#(
#  date VARCHAR(10) NOT NULL,
#  PrevRealVol_Error FLOAT NULL,
#  PredVol_Error FLOAT NULL,
#  PrevRealVol_MSE FLOAT NULL,
#  PredVol_MSE FLOAT NULL,
#  PRIMARY KEY (date)
#)
#'''

#con.execute(create_table_cmd)

result_df.to_sql('Result', con=engine, index=False, if_exists = 'replace')
```

References

[1] D. Keith Sill, “Predicting Stock Market Volatility”, *Business Review*, Jan.-Feb. 1993, phil.frb.org/-/media/research-and-data/publications/business-review/1993/brjf98ks.pdf
[2] Stephen Marra, “Predicting Volatility”, *Investment Research,* Dec. 2015, lazardassetmanagement.com/docs/-m0-/22430/predictingvolatility_lazardresearch_en.pdf