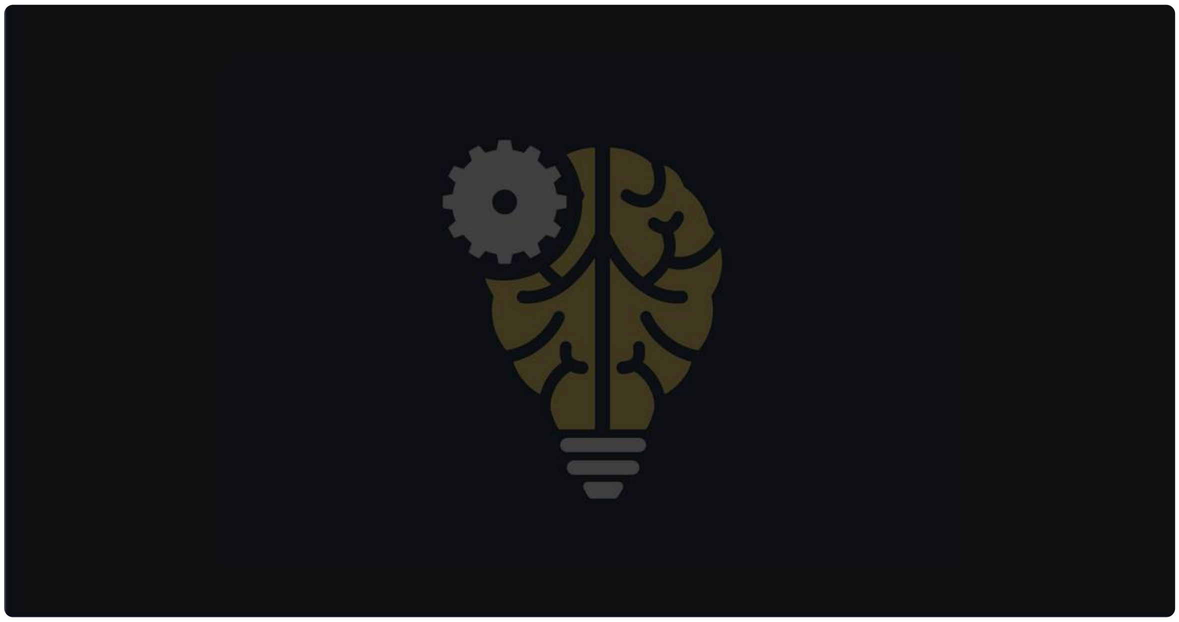
Trading Using Machine Learning In Python – SVM (Support Vector Machine)

On August 7, 2017 0 Comments

84 23 141 248 SHARES



By Varun Divakar

In this blog, I will show you how to implement a machine learning based trading strategy using the regime predictions made in the previous blog. Do read it, there is a special discount for you at the end of this.

There is one thing that you should keep in mind before you read this blog though: The algorithm is just for demonstration and should not be used for real trading without proper optimization.

Let me begin by explaining the agenda of the blog:

1. Create an unsupervised ML (machine learning) algorithm to predict the regimes.
2. Plot these regimes to visualize them.
3. Train a Support Vector Classifier algorithm with the regime as one of the features.
4. Use this Support Vector Classifier algorithm to predict the current day's trend at the Opening of the market.
5. Visualize the performance of this strategy on the test data.
6. Downloadable code for your benefit

Trading Using Machine Learning In Python – SVM (Support Vector Machine)

Import the Libraries and the Data:

First, I imported the necessary libraries. Please note that I have imported fix_yahoo_finance package, so I am able to pull data from yahoo. If you do not have this package, I suggest you install it first or change your data source to google.

```
from pandas_datareader import data as web
import numpy as np
import pandas as pd
from sklearn import mixture as mix
import seaborn as sns
import matplotlib.pyplot as plt
import talib as ta
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import fix_yahoo_finance
```

Next, I pulled the data of the same quote, 'SPY', which we used in the previous blog and saved it as a dataframe df. I chose the time period for this data to be from the year 2000.

```
df= web.get_data_yahoo('SPY',start= '2000-01-01', end='2017-08-01')
df=df[['Open','High','Low','Close']]
```

After this, I created indicators that can be used as features for training the algorithm.

But, before doing that I decided on the look back time period for these indicators. I chose a look back period of 10 days. You may try any other number that suits you. I chose 10 to check for the past 2 weeks of trading data and to avoid noise inherent in smaller look back periods.

Apart from the look back period let us also decide the test train split of the data. I prefer to give 80% data for training and remaining 20% data for testing. You can change this as per your need.

```
n = 10
t = 0.8
split =int(t*len(df))
```

Next, I shifted the High, Low and Close columns by 1, to access only the past data. After this, I created various technical indicators such as, RSI, SMA, ADX, Correlation, Parabolic SAR, and the Return of the past 1- day on an Open to Open basis.

```
df['high']=df['High'].shift(1)
df['low']=df['Low'].shift(1)
df['close']=df['Close'].shift(1)
df['RSI']=ta.RSI(np.array(df['close']), timeperiod=n)
df['SMA']= df['close'].rolling(window=n).mean()
df['Corr']= df['SMA'].rolling(window=n).corr(df['close'])
df['SAR']=ta.SAR(np.array(df['high']),np.array(df['low']),\
                  0.2,0.2)
df['ADX']=ta.ADX(np.array(df['high']),np.array(df['low']),\
                  np.array(df['close']), timeperiod =n)
df['Return']= np.log(df['Open']/df['Open'].shift(1))
```

Next, I printed the data frame.

```
print(df.head())
```

And it looked like this:

```
                  Open        High         Low     Close        high  \
Date
2000-01-03  148.250000  148.250000  143.875000  145.4375         NaN
2000-01-04  143.531204  144.062500  139.640594  139.7500  148.250000
2000-01-05  139.937500  141.531204  137.250000  140.0000  144.062500
2000-01-06  139.625000  141.500000  137.750000  137.7500  141.531204
2000-01-07  140.312500  145.750000  140.062500  145.7500  141.500000

                   low     close  RSI  SMA  Corr         SAR  ADX    Return
Date
2000-01-03         NaN       NaN  NaN  NaN   NaN         NaN  NaN       NaN
2000-01-04  143.875000  145.4375  NaN  NaN   NaN         NaN  NaN -0.032348
2000-01-05  139.640594  139.7500  NaN  NaN   NaN  148.250000  NaN -0.025357
2000-01-06  137.250000  140.0000  NaN  NaN   NaN  146.528119  NaN -0.002236
2000-01-07  137.750000  137.7500  NaN  NaN   NaN  144.672495  NaN  0.004912
```

As you can see, there are many NaN values. We need to either impute them or drop them. If you are new to the machine learning and want to learn about the imputer function, read this. I dropped the NaN values in this algorithm.

```
df=df.dropna()
```

In the next part of the code, I instantiated a StandardScaler function and created an unsupervised learning algorithm to make the regime prediction. I have discussed this in my previous blog, so I will not be going into these details again.

```
ss= StandardScaler()
unsup = mix.GaussianMixture(n_components=4,
                            covariance_type="spherical",
                            n_init=100,
                            random_state=42)
df=df.drop(['High','Low','Close'],axis=1)
unsup.fit(np.reshape(ss.fit_transform(df[:split]),(-1,df.shape[1])))
regime = unsup.predict(np.reshape(ss.fit_transform(df[split:]),\
                                                   (-1,df.shape[1])))

Regimes=pd.DataFrame(regime,columns=['Regime'],index=df[split:].index)\
                      .join(df[split:], how='inner')\
                           .assign(market_cu_return=df[split:]\
                                   .Return.cumsum())\
                                   .reset_index(drop=False)\
                                   .rename(columns={'index':'Date'})

order=[0,1,2,3]
fig = sns.FacetGrid(data=Regimes,hue='Regime',hue_order=order,aspect=2,size= 4)
fig.map(plt.scatter,'Date','market_cu_return', s=4).add_legend()
plt.show()

for i in order:
    print('Mean for regime %i: '%i,unsup.means_[i][0])
    print('Co-Variance for regime %i: '%i,(unsup.covariances_[i]))
```

Towards the end of the last blog, I printed the Mean and Covariance values for all the regimes and plotted the regimes. The new output with indicators as feature set would look like this: