Market Regime Detection using Hidden Markov Model

Author : Fahim Khan

Mentor : Nilesh Khandelwal

September 24, 2017

Contents

Chapter 1

Objective

There is always challenge for quantitative trader to find out the frequent behaviour of financial market due to change in government policy,negative news item,regulatory environment and other macroeconomics effects. Such periods are known as Market Regime. These various regimes lead to adjustments of asset returns via shifts in their means, variances, autocorrelation and covariances. This impacts the effectiveness of time series methods that rely on stationarity. There is a clear need to effectively detect these regimes. This aids optimal deployment of quantitative trading strategies and tuning the parameters within them.

This project is an attempt to find out such market regime and accordingly adjust the strategy. The pricipal method used to detect market regime is known as Hidden Markov Model which is a statistical time series techinique.

Chapter 2

What is Hidden Markov Model?

Before knowing about Hidden Markov Model, it important to understand Markov Model. The Markov Model is a stochastic state space model involves random transitions between states where the probability of the jump is only dependent upon the current state, rather than any of the previous states. The model is said to possess the Markov Property and is thus **memoryless**.

Markov Models can be categorised into four broad classes depending upon the autonomy of the system and whether all or part of the information about the system can be observed at each state.

	Fully Observable	Partially Observable
Autonomous	Markov Chain	Hidden Markov Model
Controlled	Markov Decision Process	Partially Observable Markov Decision Process

If model is both autonomous and fully observable. It cannot be modified by actions of an agent as in the controlled processes and all information is available from the model at any point in time.

If the model is fully autonomous but only partially observable then it is known as a Hidden Markov Model. In such a model there are underlying latent states and probability transitions between them but they are not directly observable. Instead these latent states influence the observations.

The HMM is more familiar in the speech recognition community and communication systems, but during the last years gained acceptance in finance as well as economics and management science.

Chapter 3

Implementation details

3.1 Programing Language

The project has been implemented in Python version 2.7. Python being a open source language is most popular for data analysis as it has wide range of available packages for data analysis,machine learning and statistical analysis. It is very popular language and easy to use.

3.2 Packages

The list of main packages used in this projects are as follows.

1. Python (v2.7)
2. Pandas (0.18.1)
3. Numpy (1.13.1)

4. Matplotlib (2.0.2)

5. hmlearn (0.2.0)

3.3 Strategy

As the main focus is on detection of market regime ,so I have used moving average strategy. As in case of HMM we can only have partially observable data, it become very important choose your observable data very carefully. For Moving average strategy I have choosen the daily returns as observable variable. We can also have daily std deviation ,daily volatility as observable variables.

In future work, I am going to add few more strategy with different observable variable.

3.4 Data

I have taken 120 days interaday data from Zerodha for the following script.

1. NIFTY50 Index
2. HDFCBANK
3. ICICIBANK
4. KOTAKBANK
5. ONGC
6. INFY
7. RELIANCE
8. HDFC

9. LT
10. IOC
11. SBIN
12. HINDUNILVR
13. MARUTI
14. ITC
15. TCS

It is always advisable to go for interaday data to get better result for detecting market regime using HMM.

Chapter 4

Backtesting Code

All the backtesting code is written in **backtest.py** file. The file is divided into two class namely **FinancialData** and **BacktestBase**.

4.1 FinancialData

This class get all the data required for backtesting. The class has following function.

- **init :** It initialize with ticker name. Also profolio class object is initialized here.
- **get_data :** All data reading happen in this function. Here we are loading the data in the pandas dataframe from csv file for the given ticker. You can load it from database as well.
- **plot_data :** This function plot the data on given column. By default it plot on close data.

4.2 BacktestBase

This class inherit all the property of FinacialData. Which means I can access the get_data and plot function from this class as well. This class has follwing function.

- **init :** It initialize with initial amount invested, amount after all backtesting,no of trades ,units ,position and transaction cost based on broker. I am calculating brokerage charge as per zerodha brokerage. So I am not using ftc and ptc.
- **get_trade_price :** This function gives the current trade price based on number of units.Also it add the transaction cost to it.
- **get_date_price :** This functon give the price of security at current data index level.
- **get_txn_cost :** Based on the amount invested it calculates the transaction cost.This function may change based on different broker.
- **get_trade_units :** This function gives the number of units you can buy based on the available amount.
- **place_buy_order :** This function place the buy order and accordingly modify the trade details,units and available amount.
- **place_sell_order :** This function place the sell order and accordingly modify the trade details,units and available amount.
- **trade_stats :** This function give the trade status once the backtesting is completed. Here I am calculating,initial amount invested, final balance, performance,number of trades and sharpe ratio. You can calculate your own stat here.

4.3 Portfolio

This class is written in **portfolio.py**. It actuallly being called from **trad_stat** function for each and every symbol. This class is reponsible for maintaing the portfolio details of every symbol on which we want to run tht backtesting strategy. Her we can find out which symbol is performing better for our backtesting strategy and we can focus on those symbol in live trading.

Portfolio class has following methods.

- **init :** It initialize portfolio details which hold column name for our portfolio stats.
- **add_portfolio_details :** This function add the portfolio details for every individual ticker.
- **show_portfolio_details :** This function actually store the portfolio details object in the pickel file and save on disk which can be load at anytime for comaparision , presentation etc.

Chapter 5

Training model for HMM

It is important to split the data into training and test data into proper ratio. As I have only 120 days interaday data it was difficult for me to have 70:30 ratio so I have decided to split it into 50:50 ratio. Please note that more the amount of data better would be the result. If you have large data the I would suggest you to go with 70:30 split which can be optimized.

The file used for training model is **hmm_regime_training_model.py**. This code actually split the data into training and test data in 50:50 ratio. To create the model **hmmlearn** package of python has been used. Please note that while creating the model it is important to choose the observable variable very carefully as on this variable model is going to be fit.Here in moving average startegy the observable variable is **daily returns**. We can also use **standard deviation, volatility, forward volatilty(for options) etc.**

Once the model is created, it is dumped to a file using same **pickle** module of python. It creates file for each and every symbol and store it into **Model** directory.

The code snippet for training HMM model is as follow.

```python
import warnings
import pandas as pd
import datetime
import numpy as np
import pickle
from hmmlearn.hmm import GaussianHMM

data_location = "../Data/IntraDay/"
train_test_split_ratio  = 0.5


# Hides deprecation warnings for sklearn
warnings.filterwarnings("ignore")
tickerList = ["HDFCBANK","ICICIBANK","KOTAKBANK","ONGC","INFY","RELIANCE","HDFC","LT","IOC","SBIN","HINDUNILVR","MARUTI","ITC","TCS"]

for ticker in tickerList:
  model_path = "../Models/hmm_model_"+ticker+".pkl"
  data = pd.read_csv(data_location+ticker+"-EQ.csv",index_col='Date',names=['Date','Open','High','Low','Close','Volume'],skiprows=1)
  data["Returns"] = data["Close"].pct_change()
  data.dropna(inplace=True)
  #Reverse data to get proper ascending order
  data =  data.iloc[::-1]

  ##Split data into training and test data
  training_data_len = int(len(data)*train_test_split_ratio) ##TO get training data
  end_date = data.index[training_data_len]
  start_date = data.index[training_data_len+1]
  training_df = data[:end_date]
  test_df =  data[start_date:]
```

```
# print training_df.tail()
# print test_df.head()

##Observable valriable on which hmm model will fits
returns = np.column_stack([training_df["Returns"]])

# Create the Gaussian Hidden markov Model and fit it
# to the returns cloumns fo training data, outputting a score
hmm_model = GaussianHMM(n_components=2, covariance_type="full", n_iter=1000).fit(returns)

#Dump model in a file to use it later on
pickle.dump(hmm_model, open(model_path, "wb"))
```

Chapter 6

Strategy code without using HMM

The file used to create moving average strategy is **moving_average.py**. If we run this file , it will run the moving average strategy on all the listed security given in the list. Also, you can declare inital amount to be invested. The portfolio object will give you the final status and also save it on disk for future use.

```
tickerList = ["HDFCBANK","ICICIBANK","KOTAKBANK","ONGC","INFY","RELIANCE","HDFC","LT","IOC","SBIN","HINDUNILVR",
  "MARUTI","ITC","TCS"]

  initial_investement_amount = 10000

  for ticker in tickerList:
    sma = MovingAverageStrategy(ticker,initial_investement_amount)
    sma.run(50,250)

  portOBJ = Portfolio()
  portOBJ.show_portfolio_details("port_without_hmm.pkl")
```

6.1 MovingAverageStrategy

This class implement the moving average strategy. The class inherits the BacktestBase which inherits FinancialData class. So, basically it has access to data as well as backtesting class.

It only has **run** method which is called to execute strategy on a given symbol and data. It copy the data into another data frame and do not touch origianl data frame.It split the data into 50:50 ratio same as we did it in training mode section. Please note that, here we only have to run the strategy on testing data only even though we are using moving average without HMM. Because if we run the strategy on whole data then it would be useless to compare the result between startegy with HMM and strategy without HMM as strategy with HMM will run on test data only as training data will be used for training model.

The complete code for moving average is as follow.

```
import pandas as pd
import numpy as np

from backtest import BacktestBase
from portfolio import Portfolio

train_test_split_ratio  = 0.5


class MovingAverageStrategy(BacktestBase):
  def run(self,SMA1,SMA2):
    msg = 'Running SMA strategy  for %s  |SMA1 =%d  |SMA2 = %d  | ftc  = %f|ptc  = %f'
    msg = msg%(self.ticker,SMA1,SMA2,self.ftc,self.ptc)
    self.position = 0
    self.amount = self.initial_amount
    self.trades = 0


```

```
    #Data Preparation
    self.data_run = self.data.copy()
    self.data_run['SMA1'] = self.data_run['Close'].rolling(SMA1).mean()
    self.data_run['SMA2'] = self.data_run['Close'].rolling(SMA2).mean()
    self.data_run["Returns"] = self.data_run["Close"].pct_change()
    self.data_run.dropna(inplace=True)


    ##Split data into training and test data
    training_data_len = int(len(self.data_run)*train_test_split_ratio) ##TO get training
    data
    end_date = self.data_run.index[training_data_len]
    start_date = self.data_run.index[training_data_len+1]
    training_df = self.data_run[:end_date]
    test_df =  self.data_run[start_date:]

    #Running for test data only
    self.data_run = test_df



    # print self.data_run.head()

    ########Signals
    Signals = pd.DataFrame(index=self.data_run.index)
    Signals["PnL"] = 0
    Signals["Trade"] = 0
    Signals["Units"] = 0

    for bar in range(0,len(self.data_run)):
      if self.position == 0:
        if self.data_run['SMA1'].ix[bar] > self.data_run['SMA2'].ix[bar]:
          self.place_buy_order(bar,amount=self.amount)
```

```
          self.position = 1 #Take Position
          Signals["Trade"].ix[bar] = 1
          Signals["Units"].ix[bar] = int(self.units)
        else:
          Signals["Trade"].ix[bar] = 0

      elif self.position == 1:
        if self.data_run['SMA1'].ix[bar] < self.data_run['SMA2'].ix[bar]:
          Signals["Units"].ix[bar] = int(self.units)
          self.place_sell_order(bar,units=self.units)
          self.position = 0   #Market nuetral
          Signals["Trade"].ix[bar] = -1
        else:
          Signals["Trade"].ix[bar] = 0


    ##Squaring off if holds any stock without checking any condition
    if self.position == 1:
      Signals["Units"].ix[bar] = int(self.units)
      self.place_sell_order(bar,units=self.units)
      self.position = 0   #Market nuetral
      Signals["Trade"].ix[bar] = -1


    ##Concatenating both dataframe
    frames = [self.data_run,Signals]
    self.final_dataframe = pd.concat(frames,axis=1, join_axes=[self.data_run.index])


    ##PnL
    price=0.
    for bar in range(0,len(self.final_dataframe)):
```

```
      if self.final_dataframe['Trade'][bar] == 1:
        price = self.get_trade_price(bar,self.final_dataframe['Units'][bar])
      elif self.final_dataframe['Trade'][bar] == -1:
        self.final_dataframe['PnL'].ix[bar] = self.get_trade_price(bar,self.final_dataframe[
    'Units'][bar]) - price

    self.trade_stats(bar,self.final_dataframe)

if __name__ == "__main__":
  tickerList = ["HDFCBANK","ICICIBANK","KOTAKBANK","ONGC","INFY","RELIANCE","HDFC","LT","IOC
    ","SBIN","HINDUNILVR",
  "MARUTI","ITC","TCS"]

  initial_investement_amount = 10000

  for ticker in tickerList:
    sma = MovingAverageStrategy(ticker,initial_investement_amount)
    sma.run(50,250)

  portOBJ = Portfolio()
  portOBJ.show_portfolio_details("port_without_hmm.pkl")
```

Chapter 7

Strategy code with using HMM

The file used to create moving average strategy with using HMM is **moving_average_using_hmm.py**. If we run this file , it will run strategy on all the listed security given in the list. Also, you can declare inital amount to be invested. The portfolio object will give you the final status and also save it on disk for future use.

It first load the model which we have trained in previouse chapter. It keeps adding daily returns to a list from test data.And whenever the strategy go for trade it actually first check if any regime is detected by feeding daily returns(Observable data) to HMM model. And if model got any regime the strategy would not trade and also it will sqaure off any holdings as well.

The function to detect regime is as follow.

```
def regime_detection(self,daily_returns):
    #Converting daily return list to numpy array
    daily_returns = np.column_stack([np.array(daily_returns)])
    hidden_state = self.hmm_model.predict(daily_returns)[-1]
    return hidden_state
```

If the function returns 1 it will not trade and squareoff if it has any holdings. Otherwise it will work as normal strategy.

The complete code snippet for moving average strategy with using HMM is as follow.

```

import pandas as pd
import numpy as np
import pickle

from backtest import BacktestBase
from portfolio import Portfolio



train_test_split_ratio  = 0.5

class MovingAverageStrategy(BacktestBase):
  def run(self,SMA1,SMA2,model_path):
    daily_returns = []
    msg = 'Running SMA strategy for %s |SMA1 =%d |SMA2 = %d | ftc = %f|ptc = %f'
    msg = msg%(self.ticker,SMA1,SMA2,self.ftc,self.ptc)
    self.position = 0
    self.amount = self.initial_amount
    self.trades = 0

    self.hmm_model = pickle.load(open(model_path, "rb"))
```

```


    #Data Preparation
    self.data_run = self.data.copy()
    self.data_run['SMA1'] = self.data_run['Close'].rolling(SMA1).mean()
    self.data_run['SMA2'] = self.data_run['Close'].rolling(SMA2).mean()
    self.data_run["Returns"] = self.data_run["Close"].pct_change()
    self.data_run.dropna(inplace=True)


    ##Split data into training and test data
    training_data_len = int(len(self.data_run)*train_test_split_ratio) ##TO get training
    data
    end_date = self.data_run.index[training_data_len]
    start_date = self.data_run.index[training_data_len+1]
    training_df = self.data_run[:end_date]
    test_df =  self.data_run[start_date:]

    #Running for test data only
    self.data_run = test_df



    # print self.data_run.head()

    #######Signals
    Signals = pd.DataFrame(index=self.data_run.index)
    Signals["PnL"] = 0
    Signals["Trade"] = 0
    Signals["Units"] = 0

    for bar in range(0,len(self.data_run)):
      ##Storing observable varibale.In our case it is daily returns.
```

```
daily_returns.append(self.data_run['Returns'].ix[bar])

regime = self.regime_detection(daily_returns)

if self.position == 0:
  if regime == 1:
    pass
  else:
    if self.data_run['SMA1'].ix[bar] > self.data_run['SMA2'].ix[bar]:
      self.place_buy_order(bar,amount=self.amount)
      self.position = 1 #Take Position
      Signals["Trade"].ix[bar] = 1
      Signals["Units"].ix[bar] = int(self.units)
    else:
      Signals["Trade"].ix[bar] = 0

elif self.position == 1:
  if regime == 1:
    print "Sell Without condition"
    Signals["Units"].ix[bar] = int(self.units)
    self.place_sell_order(bar,units=self.units)
    self.position = 0   #Market nuetral
    Signals["Trade"].ix[bar] = -1
  else:
    if self.data_run['SMA1'].ix[bar] < self.data_run['SMA2'].ix[bar]:
      Signals["Units"].ix[bar] = int(self.units)
      self.place_sell_order(bar,units=self.units)
      self.position = 0   #Market nuetral
      Signals["Trade"].ix[bar] = -1
    else:
      Signals["Trade"].ix[bar] = 0

```

```



    ##Squaring off if holds any stock without checking any condition
    if self.position == 1:
      Signals["Units"].ix[bar] = int(self.units)
      self.place_sell_order(bar,units=self.units)
      self.position = 0   #Market nuetral
      Signals["Trade"].ix[bar] = -1


    ##Concatenating both dataframe
    frames = [self.data_run,Signals]
    self.final_dataframe = pd.concat(frames,axis=1, join_axes=[self.data_run.index])


    ##PnL
    price=0.

    for bar in range(0,len(self.final_dataframe)):
      if self.final_dataframe['Trade'][bar] == 1:
        price = self.get_trade_price(bar,self.final_dataframe['Units'][bar])
      elif self.final_dataframe['Trade'][bar] == -1:
        self.final_dataframe['PnL'].ix[bar] = self.get_trade_price(bar,self.final_dataframe['Units'][bar]) - price


    self.trade_stats(bar,self.final_dataframe)

  def regime_detection(self,daily_returns):
    #Converting daily return list to numpy array
    daily_returns = np.column_stack([np.array(daily_returns)])
```

```
        hidden_state = self.hmm_model.predict(daily_returns)[-1]
        return hidden_state


if __name__ == "__main__":
    tickerList = ["HDFCBANK","ICICIBANK","KOTAKBANK","ONGC","INFY","RELIANCE","HDFC","LT","IOC","SBIN","HINDUNILVR",
    "MARUTI","ITC","TCS"]

    initial_investement_amount = 10000

    for ticker in tickerList:
        model_path = "../Models/hmm_model_"+ticker+".pkl"
        sma = MovingAverageStrategy(ticker,initial_investement_amount)
        sma.run(50,250,model_path)


    portOBJ = Portfolio()
    portOBJ.show_portfolio_details("port_with_hmm.pkl")
```

Chapter 8

Findings

The portfolio details has been saved as pickle object which can be loaded and see the result. To read the result please look at file **Read_Portfolio.py**

This file just load the result file and print it in human readable format. The code snippet is as follows.

```
import pandas as pd

port_without_hmm = pd.read_pickle("port_without.pkl")
port_with_hmm = pd.read_pickle("port_with.pkl")

print "#######Portfolio Wihtout using HMM##########"
print port_without_hmm

print "###############Portfolio with using HMM############"
print port_with_hmm
```

The result for moving average strategy without using HMM shown in figure 8.1.

	Invested Amount	Final Amount	Number of Trades	PnL	Sharpe Ratio	Performance
HDFCBANK	10000	10560.77899	12	560.77899	0.612232617292	5.6077899
ICICIBANK	10000	9989.614016	18	-10.385984	0.00532111748876	-0.10385984
KOTAKBANK	10000	10308.6009	20	308.6009	0.291930663207	3.086009
ONGC	10000	9198.240765	20	-801.759235	-0.545169050789	-8.01759235
INFY	10000	9718.72285	16	-281.27715	-0.223085075074	-2.8127715
RELIANCE	10000	11642.677855	12	1642.677855	0.475795629632	16.42677855
HDFC	10000	10256.49715	16	256.49715	0.271698593302	2.5649715
LT	10000	9766.627696	18	-233.372304	-0.13980268666	-2.33372304
IOC	10000	8794.833325	22	-1205.166675	-0.602970840762	-12.05166675
SBIN	10000	10440.14991	18	440.14991	0.196555503072	4.4014991
HINDUNILVR	10000	10645.769665	18	645.769665	0.259279075127	6.45769665
MARUTI	10000	10306.55655	14	306.55655	0.318665447398	3.0655655
ITC	10000	9957.459135	20	-42.540865	-0.0117827870183	-0.42540865
TCS	10000	10168.23993	16	168.23993	0.160543912718	1.6823993

Figure 8.1: Result without HMM.

The result for moving average strategy with using HMM shown in figure 8.2.

	Invested Amount	Final Amount	Number of Trades	PnL	Sharpe Ratio	Performance
HDFCBANK	10000	10539.82183	40	539.82183	0.433746837408	5.3982183
ICICIBANK	10000	9987.842031	28	-12.157969	0.0104600380918	-0.12157969
KOTAKBANK	10000	10153.75595	54	153.75595	0.137388576591	1.5375595
ONGC	10000	9226.209245	94	-773.790755	-0.326931551136	-7.73790755
INFY	10000	9814.2199	70	-185.7801	-0.0600898110903	-1.857801
RELIANCE	10000	11523.737215	24	1523.737215	0.474966781913	15.23737215
HDFC	10000	10058.3863	28	58.3863	0.156290783519	0.583863
LT	10000	9990.057544	24	-9.942456	0.0204548447429	-0.09942456
IOC	10000	8779.5593	30	-1220.4407	-0.55453045179	-12.204407
SBIN	10000	9939.5073	26	-60.4927	-0.179452317552	-0.604927
HINDUNILVR	10000	10457.693785	78	457.693785	0.271372324115	4.57693785
MARUTI	10000	10307.042875	36	307.042875	0.308808102866	3.07042875
ITC	10000	9829.2418	36	-170.7582	-0.0365560054164	-1.707582
TCS	10000	10202.795155	58	202.795155	0.13035605233	2.02795155

Figure 8.2: Result with HMM.

Chapter 9

Future Work

Chapter 10

Reference