

Fundamentals of Artificial Intelligence
CSCE 5210 – SPRING 2022 – Academic Project-
Increment -1

Signal Generator for making a market view for
Index Options of Indian Stock Market
Benchmark Index NIFTY 50

TEAM	
Veeranjaneyulu Muppala	11532514
Yash Zauwar	11439929

Abstract:

Nifty Index Options is a ZERO-SUM GAME – Which means if you are making gains then someone is making the same loss so if you understand how to read the data, you can make some fortune for yourselves. The motivation for this project is to make the most of the available data, developed for generating a BUY or SELL signal depending on performing some calculations and behavior of option chain data.

Introduction:

The NIFTY 50 is one of two main stock indices in India, the other being the BSE - SENSEX. It represents the weighted average of the index of the 50 largest Indian companies listed on the National Stock Exchange. NIFTY Options is derivative instrument of asset NIFTY. It has different strikes and multiple expiries.

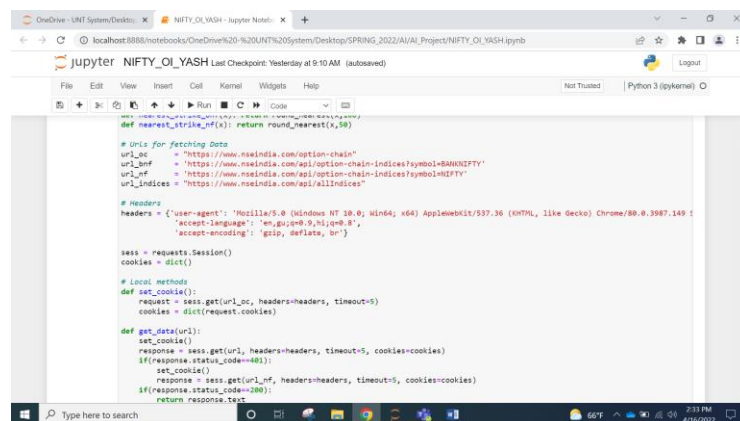
1 Problem specification:

There is a risk involved in options trading. The capital may blow out sometimes if gone wrong. To overcome this, to find the direction of market which is either bullish or bearish to exercise the particular option, the code generates the signal to BUY or SELL depending on the market sentiment.

1.1 Dataset:

The data is obtained from the National Stock Exchange Website of India. It consists of the strike price, change in Open Interest, Open Interest of both call and put options which all together is called as Option Chain Data.

Code for fetching the data:



```
def nearest_strike(nf): return round_nearest(x,50)

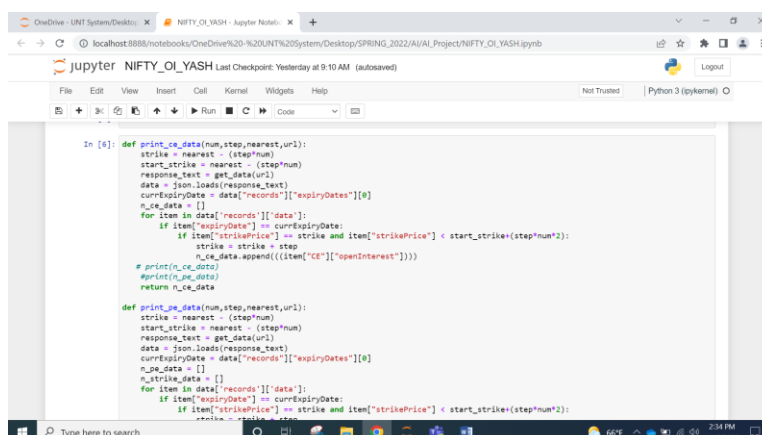
# urls for fetching data
url_pc = "https://www.nseindia.com/option-chain"
url_nf = "https://www.nseindia.com/api/option-chain-indices?symbol=BANKNIFTY"
url_nf = "https://www.nseindia.com/api/option-chain-indices?symbol=NIFTY"
url_indices = "https://www.nseindia.com/api/allIndices"

# Headers
headers = {'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.3987.149 ; accept-language: en-gb,en;q=0.9,h;en;q=0.8', 'accept-encoding': 'gzip, deflate, br'}

sess = requests.Session()
cookies = dict()

# Local methods
def set_cookie():
    request = sess.get(url_pc, headers=headers, timeout=5)
    cookies = dict(request.cookies)

def get_data(url):
    set_cookie()
    response = sess.get(url, headers=headers, timeout=5, cookies=cookies)
    if(response.status_code==200):
        set_cookie()
    response = sess.get(url_nf, headers=headers, timeout=5, cookies=cookies)
    if(response.status_code==200):
        return response.text
```



```
In [4]: def print_ce_data(num,step,nearest,url):
    strike = nearest - (step*num)
    start_strike = nearest - (step*num)
    response_text = get_data(url)
    data = json.loads(response_text)
    curExpiryDate = data["records"][0]["expiryDates"][0]
    n_ce_data = []
    for item in data["records"][1:]:
        if item["expiryDate"] == curExpiryDate:
            if item["strikePrice"] == strike and item["strikePrice"] < start_strike+(step*num*2):
                strike = strike + step
                n_ce_data.append([item["ce"],["openInterest"]])
    # print(n_ce_data)
    return n_ce_data

def print_pe_data(num,step,nearest,url):
    strike = nearest - (step*num)
    start_strike = nearest - (step*num)
    response_text = get_data(url)
    data = json.loads(response_text)
    curExpiryDate = data["records"][0]["expiryDates"][0]
    n_pe_data = []
    n_strike_data = []
    for item in data["records"][1:]:
        if item["expiryDate"] == curExpiryDate:
            if item["strikePrice"] == strike and item["strikePrice"] < start_strike+(step*num*2):
```

```
def get_data(url):
    response_text = get_data(url)
    data = json.loads(response_text)
    expiryDates = data["records"][0]["expiryDates"]
    n_ce_col_data = []
    for item in data["records"][0]["data"]:
        if item["expiryDate"] == expiryDates:
            if item["strikePrice"] == strike and item["strikePrice"] < start_strike(step*num/2):
                strike = strike + step
                n_ce_col_data.append((item["CE"]*["changeInOpenInterest"])))
    # print(n_ce_data)
    # print(n_pe_data)
    return n_ce_col_data

def print_pe_col_data(num, step, nearest, url):
    strike = nearest - (step*num)
    start_strike = nearest - (step*num)
    response_text = get_data(url)
    data = json.loads(response_text)
    expiryDates = data["records"][0]["expiryDates"]
    n_pe_col_data = []
    for item in data["records"][0]["data"]:
        if item["expiryDate"] == expiryDates:
            if item["strikePrice"] == strike and item["strikePrice"] < start_strike(step*num/2):
                strike = strike + step
                n_pe_col_data.append((item["PE"]*["changeInOpenInterest"])))
    # print(n_ce_data)
    # print(n_pe_data)
    return n_pe_col_data
```

```
def print_strike_data(num, step, nearest, url):
    strike = nearest
    start_strike = nearest
    response_text = get_data(url)
    data = json.loads(response_text)
    expiryDates = data["records"][0]["expiryDates"]
    n_strike_data = []
    for item in data["records"][0]["data"]:
        if item["expiryDate"] == expiryDates:
            if item["strikePrice"] == strike and item["strikePrice"] < start_strike(step*num/2):
                strike = strike + step
                n_strike_data.append((item["strikePrice"]*["changeInOpenInterest"])))
    # print(n_ce_data)
    # print(n_pe_data)
    return n_strike_data

In [7]: a = print_ce_data(10,50,nf_nearest,url_nf)
        #print(a)
        b = print_pe_data(10,50,nf_nearest,url_nf)
        #print(b)
        c = print_strike_data(10,50,nf_nearest,url_nf)
        #print(c)
        d = print_ce_col_data(10,50,nf_nearest,url_nf)
        #print(d)
        e = print_pe_col_data(10,50,nf_nearest,url_nf)
        #print(e)
        sum_ce_time = []
        sum_pe_time = []
        sum_ce = 0
        sum_pe = 0
```

```
sum_ce = sum_ce + e[1]
sum_pe = sum_pe + e[1]
print("CALL SIDE " + str(sum_ce))
for i in range(0, len(d)):
    sum_pe = sum_pe + d[i]
    print("PUT SIDE " + str(sum_pe))
for i in range(0, len(d)):
    sum_ce = sum_ce + d[i]
    print("Total Change of OI - CALL SIDE " + str(sum_ce))
for i in range(0, len(e)):
    sum_pe = sum_pe + e[i]
    print("Total Change of OI - PUT SIDE " + str(sum_pe))
#####
if sum_ce < sum_pe:
    print(strRed(str("SIGNAL GIVEN BY DATA IS BUY")))
else:
    print(strRed(str("SIGNAL GIVEN BY DATA IS SELL")))
```

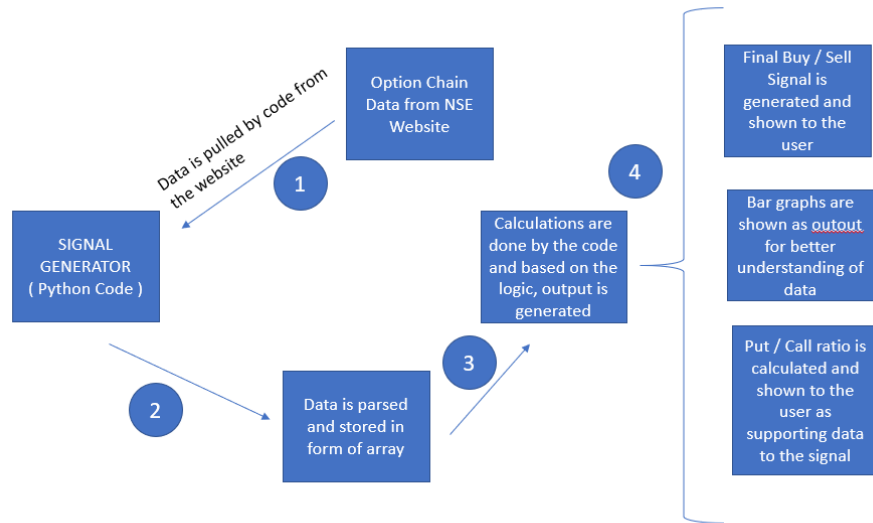
```
CALL SIDE 907189
PUT SIDE 612684
Total Change of OI - CALL SIDE 53479
Total Change of OI - PUT SIDE -66425
SIGNAL GIVEN BY DATA IS SELL
```

1.2 Problem analysis:

Most of the money in stock market options is exchanged from people with low capital to institutions, what we call as bigger hands. During initial days of trading, if you start without understanding what you are doing or the without having knowledge of market sentiment and how the price flow action takes place then that will be incurred with risk of losing money. To overcome this on analyzing the data, a particular pattern in options data can be seen which can be used to predict the zone in which the market is moving.

2 Design and Milestones

Design:



Milestones:

- I. Fetching data from the website using relevant cookies and headers
- II. Storing the data in arrays without any errors
- III. Performing calculations based on the logic
- IV. Final BUY/SELL signal to be generated
- V. PCR (Put Call Ratio) to be calculated
- VI. Bar graph of the CE & PE data to be shown

2.1 Proposed Method:

In this method, the signal is generated as per the calculation done by data. The change in Open Interest for the day is used as indicator and if there is a negative change or positive change on the call side or put side of the option then buy or sell is given accordingly.

For ex: In the below screenshot, the call side OI and put side OI is present.

The total change in OI is calculated and as there is a negative change in put side for that day which means puts are unwinding which is basically a sign of bearishness of the market hence the signal given is SELL. Likewise, if there is negative call side data then it is a sign of bullishness.

```
CALL SIDE 907189
PUT SIDE 612684
Total Change of OI - CALL SIDE 53470
Total Change of OI - PUT SIDE -66425
SIGNAL GIVEN BY DATA IS SELL
```

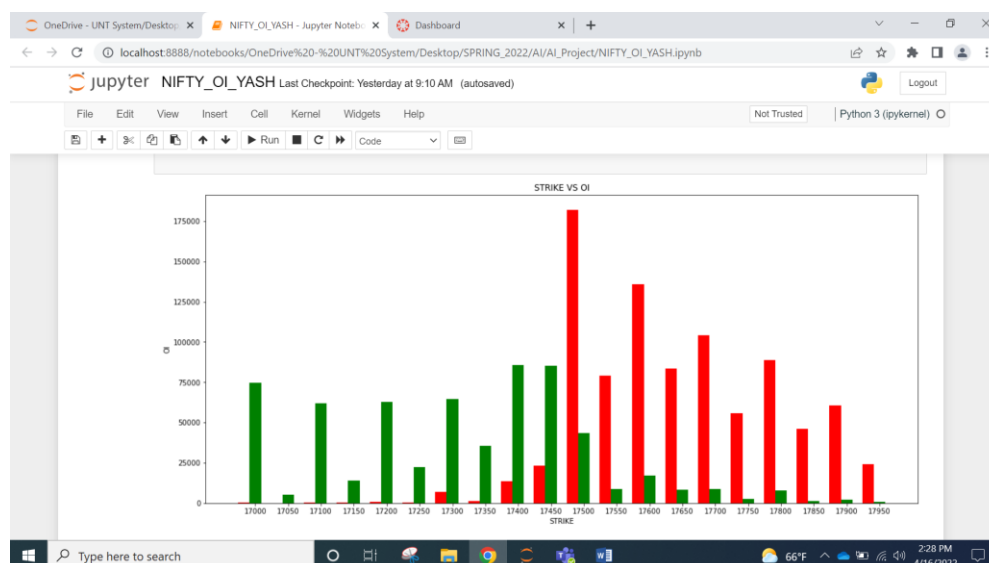
The bar graph is generated by the below code:

```
OneDrive - UNT System/Desktop x NIFTY_OI_YASH - Jupyter Noteb Dashboard x +
localhost:8888/notebooks/OneDrive%20-%20UNT%20System/Desktop/SPRING_2022/AI/AI_Project/NIFTY_OI_YASH.ipynb
jupyter NIFTY_OI_YASH Last Checkpoint: Yesterday at 9:10 AM (autosaved) Python 3 (ipykernel)
File Edit View Insert Cell Kernel Widgets Help Not Trusted
a = print_ce_data(10,50,nf_nearest,url_nf)
#print(a)
b = print_pe_data(10,50,nf_nearest,url_nf)
#print(b)
c=print_strike_data(10,50,nf_nearest,url_nf)
#print(c)
# data to plot
n_groups = 20
# create plot
#fig, ax = plt.subplots()
index = np.arange(20)
plt.figure(figsize=(18,8))
bar_width = 0.35
rects1 = plt.bar(index, a, bar_width,color='r',label='CE')
rects2 = plt.bar(index + bar_width, b, bar_width,color='g',label='PE')
plt.xlabel('STRIKE')
plt.ylabel('OI')
plt.title('STRIKE VS OI')
plt.xticks(index + bar_width, c)
plt.show()
```

Here in this part of code, we are using the inputs as strike price, total OI on call side and total OI on put side. A bar graph is then generated and shown as below.

The red color bar indicates resistance and green indicates support.

As per the below graph, it shows that the call OI is highest on 17500 strike and significantly less OI on put side which is a sign of bearishness again.



2.2 Data processing:

The data which is obtained from the website is processes in JSON and load as text in the project.

2.3 Experimental Settings:

This project is to be executed after 10:15 AM of the trading day. Though it gives output before the time but accuracy is less.

2.4 Validation methods:

Validations done in the code is expiry date. Currently the Expiry of option is Thursday of each week. In case it is holiday then the expiry is set to the previous trading day. To overcome the scenario, instead of setting the expiry date by using the calendar, we are obtaining it from the data from the National Stock Exchange website.

Limitations:

Few limitations of the project are

The signal is less accurate on Friday's as the option chain data is very less. The general weekday accuracy is around 70% and around 50% on Friday's. The accuracy on expiry day is around 50%.

The signal generated post 10:15 am IST has more clear view as majority of the positions will be created in the first trading hour.

Global factors like War like scenario's etc.... Any announcements in the global financial markets.

Future Work:

Calculation of PCR from the data is still in progress.

References:

<https://www.investopedia.com/trading/options-trading-volume-and-open-interest/>

<https://www.investopedia.com/financial-edge/0412/a-newbies-guide-to-reading-an-options-chain.aspx#:~:text=The%20order%20of%20columns%20in,dates%20have%20different%20options%20symbols.>

<https://www.nseindia.com/products-services/about-indices>

<https://medium.datadriveninvestor.com/python-utility-to-derive-nifty-support-and-resistance-zone-based-on-live-weekly-monthly-option-1b02bd3b31dd>

<https://dev.to/shahstavan/nse-option-chain-data-using-python-4fe5>

<https://www.nseindia.com/option-chain>

Disclaimer: The project is for personal research and experience and is intended to be educational material and trade executed based on the output of the project should be consulted with their financial advisor. Owners of the project are not responsible for any kind of financial loss.