2022

An implementation of machine learning for security selection

By Sean Kessler

[CONVOLUTIONAL NEURAL NETWORK FOR MOMENTUM SYSTEMS]

This document is an overview of the implementation of the machine learning process for a momentum model

A Convolutional Neural Network for Security Selection

Table of Contents

Introduction **3**

Python Model Code **8**

Data Generation I **9**

Data Generation II **10**

Running Models **11**

Leveraging the Model in Client Code **12**

Bibliography **13**

A Convolutional Neural Network for Security Selection

Introduction

This is an introduction to the particular endeavor that I have undertaken in order to implement a modern machine learning approach to trading in the equities market.

I could have purchased an off-the-shelf tool and perhaps for the ordinary person this indeed is the way to go. A tool like TradeStation provides limitless possibilities in terms of model development with readily available data and canned models that can be tweaked to the authors delight.

I wanted this to be a learning process. I wanted to come to understand what it would take to lift something off the ground from scratch. I wanted to build the database, write the feeds, write the data access components, build the visualization into the front-end and write the models.

The feeds capture a variety of information. These are daily prices, ratings, dividend history, analyst price targets, yield curves, earnings announcements, news headlines, currency conversions, stick splits, income statement, balance sheet, statement of cashflows, SEC filings and a variety of economic indicators such as GDP.

I spent several years scaffolding a basic platform. It consisted of a MySQL database with a dozen or so tables and it had a nice front-end with a very fast visualization engine. The main view of the system was a hybrid multiple document interface that consisted of individual equities presented along a Bollinger band with the various standard deviation lines (+-1/+-2). The very first analytics I introduced into the system was Valuation page which was based on Phil Town's Rule #1 "Sticker Price" and "Margin of Safety" concept. I also built a FreeCashflow analysis page.

I then produced my first model. This really wasn't a model at all but rather a monthly ranking based on Phil Town's "Sticker Price" and "Margin of Safety". To that list I also added "Acquirers Multiple" ranking which I had got from Tobias Carlisle's book of the same name followed by Joel Greenblatt's magic number which is based on ROIC and Earnings Yield from his book "The Little Book that Still Beats The Market". All of these I presented in tabular form that could be reloaded from the database by selecting the appropriate analysis date from a dropdown menu.

This "system" worked fairly well but there were drawbacks. For starters the stock selection process was discretionary. This meant that the buying and selling of securities was totally up to the individual, me in this case, and how I was feeling about the data and the market on any given day. The second drawback was that this system really didn't work for the way that I wanted to trade. By definition the ranking process is deep-value in nature. This means that, unless one can time the market, one is usually buying into some level of weakness prior to an eventual upswing. There is a great payoff ahead but one needs to be patient and wait for it. The waiting process is generally not rewarded by periodic dividends either.

What I wanted was a simple model that I could run not daily, but weekly or perhaps monthly. I had a fulltime job and simply did not have the time during the day to stare at stock screens nor did I want to become a day trader. This system was not going to make me a millionaire overnight but it was going to produce money safely over time. This is what I wanted to build.

The first "real" model that I built was a monthly system I leveraged from the book "Quantitative Momentum" by Wesley Gray and Jack Vogel. This was mainly a stock selection system based on 52 week ranking of momentum. The basic idea behind this approach was a rotating bucket of stocks. For instance, one would purchase a bucket of stocks every month (say 3) for a number of months (say 3). On the fourth month the stocks purchased in the first month would be sold and another 3 stocks purchased. The system sold the stocks regardless of whether they made money or lost money. This was a long-only system. Built into the system was the ability to flee to safety if the market became volatile. I liked this system because it was automated. I disliked it because I found it painful to sell a stock after some months when it was clear that the stock was running up. Likewise, I found it difficult to hang on to a stock when it was clear that the stock was exhausted. At any rate, I was very disciplined with the system and to this day have never overridden it. I refer to this model as MGMomentum.

The second "real" model that I build was based on a book by Andreas Clenow "Stocks on the Move: Beating the Market with Hedge Fund Momentum Strategies". The approach here is also largely a stock selection process. I will refer to this model as CMMomentum. The model differs from MGMomentum largely in two ways. First, it incorporates a method to determine if a stock is overextended and over what horizon the overextension has occurred. It uses overextension to try to locate stocks that have run up too much over a short period of time. Secondly, the model uses the log() of the prices to put those in a time domain which it

then calculates a slope to determine a quality of returns score which is used as a rank. Clenow does not get into details about "how" to trade the model and so left to my own devices I incorporated an approach that was analogous to the MGMomentum model I had developed earlier. For example, I applied a similar bucketing approach to the trading strategy that I had implemented in the earlier model.

Of course I back-tested both of these models going back 10 years and while the MGMomentum model was a "Steady Freddie" the CMMomentum model was a raging bull. When is won it won big. When is lost it lost big. If you could hang on to the reigns it paid off. What was really missing in these two models was the notion of cutting your losses and letting your profits run. The two aforementioned models would simply buy securities, hold them for a specified amount of time and then sell them. It did not care if the stock was making significant gains nor if the stock was falling off a ledge.

I needed a better way to manage risk. I wanted to develop a system that would cut my losses quickly and also let my profits run. There were two books that influenced me at this time. The first book by Marc Minervini introduced a technique that Marc called
Volatility Price Contraction or VPC. This was a technical indicator of when a stock was about to make a major breakout and advance. I was fascinated by this elusive VPC. The second author was Van Tharp. Van Tharp introduced me to the concept of risk adjusted positions which up to this point I had really no concept of. Known better in his writings as "R" this was a discipline for predetermining what position risk was going to look like for any given trade and thus a method for exiting the trade early if things didn't go as planned or running to the end if things did go to according to plan.

Armed with this new knowledge I proceeded to develop yet a third model which I called CMTrend. This was a model that would incorporate all of what I had learned from Van Tharp in terms of position sizing and how to set and adjust stop limits for the duration of the holding period.

This third model differed significantly from the prior models in some very significant ways. Firstly, CMTrend had a candidate selection process which was separate and distinct from its position entry process. Second, in CMTrend every new position had a predefined stop limit that would be set when the position was opened. This stop limit would adjust upwards depending on both how long the position was held and also the true range of the price.

In summary and up to this point I have described three models…

MGMomentum – My steady Freddie.
CMMomentum – My raging bull
CMTrend – A newly backtested model that I had just begun trading.

Having developed these three models I turned back to re-examine VPC as put forth by Marc Minervini. Now Marc had published some examples of VPC. All in all I think I may have found among his writings three or four examples. I stared at these examples. I totally appreciated that Marc could probably identify these patterns in his sleep. I could not. I thought maybe I simply didn't have enough examples and so the patterns were just not emerging for me. I started to think about where I could find more examples of his VPC. I searched and searched and came up empty. What I did have at hand were all of the trade results that had accumulated from running my CMMomentum model. CMTrend had only been running in production for a few months at that time but CMMomentum had accumulated hundreds of trades.
I thought if somehow I could teach the CMMomentum model to learn from its past trades then maybe this could be something quite useful. I thought that even if I could not visually detect certain patterns of the winners vs. the losers than regardless; that patterns did indeed exist and a computer model could be developed to identify them.

I searched for machine learning books that were oriented towards computer vision. I took this approach because I had planned to give the model a picture of the Bollinger band with its depiction of volatility and standard deviation. I remember thinking that a computer should be able to hone in on the structure of that volatility pattern for the losers versus the winners.

I came across one such book by Mohamed Elgendy titled "Machine Learning for Vision Systems". There was a huge learning curve for me here. For starters even though I had a basic understanding of machine learning from some prior readings I had never undertaken to understand how it might be implemented in an actual system. I also needed to acquire an understanding of Python, Keras, NumPy, and a variety of other packages that comprised the heart of what a Machine Learning process was. Finally, I needed to understand how I might be able to incorporate an ML process into my existing model.

After spending quite a considerable amount of time I set forth to develop a CNN (Convolutional Neural Network) front end that fed into a Multilayer Perceptron or MLP.
I would train this network on Bollinger band from my top 20 and bottom 20 performers from the CMMomentum model. It was my hope that I could develop a model that would

understand and “nudge” the CMMomentum model towards trades that “looked” like successful predecessors.

The most difficult part of this process was that for the training I was going to need tens of thousands of input data where I only had hundreds. What I did to overcome this obstacle was to convolute my existing data so that from a given piece of information I would produce a hundred. This is how I trained the system.

I played around with a variety of models that I had learned in Elegency’s book. The one that proved to be the best for my set of conditions was the resnet50 network. I hosted this network in flask using Python and was able to incorporate it into my existing CMMomentum model by calling it over HTTP. A positive hit by the model would credit the score of the current candidate by 20%. So if a candidate exited the CMCandidate selection process with a score of 100 and then produced a positive hit in the Resnet50 model it wound up with an overall score of 120. This worked great in the backtests and so I started running it in production in 2021.

As of this writing (11/3/2022) the markets have been in a downtrend since June and two of my models have exited the market entirely. As a result I have not yet been able to assess the performance of the model over any length of time. I can say that prior to the general market downturn the model was performing exceptionally well.

Python Model Code

1) Network is resnet50. This is the current production model.

2) C:\Boneyard\DeepLearning\Data\0 This is the avoid data
C:\Boneyard\DeepLearning\Data\1This is the “good” data.
This is the location of data for training.

3) C:\Boneyard\DeepLearning\ModelInputData.
This is the data where the code generates the image data. The data must then be copied to the appropriate folder (/Data/0 /Data/1) and then the model run.
In general, the folder structure should look like this.

```
├── data
│   ├── class2
│   │   ├── result_image5.png
│   │   ├── result_image9.png
│   │   ├── result_image15.png
│   │   ├── result_image13.png
│   │   ├── result_image1.png
│   │   ├── result_image3.png
│   │   ├── result_image11.png
│   │   ├── result_image19.png
│   │   ├── result_image7.png
│   │   └── result_image17.png
│   └── class1
│       ├── result_image14.png
│       ├── result_image8.png
│       ├── result_image12.png
│       ├── result_image18.png
│       ├── result_image16.png
│       ├── result_image6.png
│       ├── result_image2.png
│       ├── result_image10.png
│       ├── result_image4.png
│       └── result_image0.png
```

The model_host.py will then produce a prediction in a range of [0..1].

(Example:model_host.py)

```
if type(predictions) == list:
    average_prediction = sum(predictions)/len(predictions)
    threshold_output = np.where(average_prediction > 0.5, 1, 0)
else :
    threshold_output = np.where(predictions > 0.5, 1, 0)
```

It does this because the activation in the final Dense layer of the resnet50.py module uses a sigmoid activation function.

(Example:resnet50.py)

```
if classes==1:
    x=Dense(classes,activation='sigmoid',name='fc'+str(classes))(x)
else:
    x=Dense(classes,activation='softmax',name='fc'+str(classes))(x)
```

4) C:\Boneyard\Documents.
This is the location of documentation for selection of training and test data

5) C:\Boneyard\CNN\Models.
This is the location of the various models.
Of particular importance are…
resnet50.py - This is the residual network model that is currently running in production
model_lenet5.py – This is the runner-up model to the production model.
Model_host.py – This is the Flask hosting platform for the models.

6) C:\Boneyard\CNN\Weights.
This is the location of the stored weights that are generated from model training.
Of particular importance are…
resnet50.h5 - Prodution weights for CMMomentum
lenet5.h5 – Runner-up.

Data Generation I

1) C:\Boneyard\DeepLearning\Documents\CMMomentumRunData\ CM10-31-2015_OI_10_1_MPBW_65_USEOVTRUE_V2.xlsm

a) The “Resnet” tab contains the various test run results for the resnet50 model. These are the final validation results and expectations.

b) The CM10-31-2015_OI_10_1_MPBW_65_US tab contains in column “U” the insert statements that should be copied into CNNProcessor.cs.

c) rows 6-40 which are the training data in the “avoid” scenario are used to feed Data[0].

d) rows 122-184 which are the training data sets in the “good” scenario are used to feed Data[1].

e) Columns R & S control which of the data are to be used for training and which data are to be used for validation.

Data Generation II

C:\Boneyard\MarketData\MarketDataLib\CNNProcessing\CNNProcessor.cs
The module contains the following methods…

a) GenerateTraining() : This method generates training data and puts the data into
C:\Boneyard\DeepLearning\ModelInputData\0
C:\Boneyard\DeepLearning\ModelInputData\1

b) GenerateGeneralizedData() : This method creates the validation cases and copies to
C:\Boneyard\DeepLearning\IndividualTestCases.

c) TestPredictAPI() : This method calls the resnet50 model (API) with the test cases and writes the results to the console. This output winds up in the “Resnet50” tab of the spreadsheet.

Running Models

C:\GIT\Keras
C:\GIT\absl
C:\Boneyard\CNN\Models models folder
C:\Boneyard\CNN\Weights weights folder

Since the residual network model (resnet50) is the current model being run in the CMMomentumModel it is described here, The other models should work similarly.

a) model_sk_resnet.py - This is the resnet50 model setup
  resnet50.py - This is the resnet50 model
Run the model with Visual Studio Code. The model weights will be generated in the same folder (resnet50.h5). Copy the weights file to C:\Boneyard\CNN\Weights if satisfied with results. The Flask host will load the model weights from C:\Boneyard\CNN\Weights.

b) model_host.py : This is the flask web host for the models. Currently, two models are hosted. These are ../weights/resnet50.h5 and ../weights/lenet5.h5

Notes

The model was trained with data from 2015-2021. It should be retrained periodically..although not sure what the period should be. For instance, nearly nothing performed well in 2022 so not sure that retraining in that years would add any insight to the model.

Leveraging the Model
Client Code

a) C:\Boneyard\MarketData\MarketDataLib\CNNProcessing\CNNClient.cs->Predict(..)
This is the main method used by the CMMomentum Generator.

b)
C:\Boneyard\MarketData\MarketDataLib\Generator\CMMomentum\CMMomentumGenerator.cs
in the PredictCandidate(CMCandidate cmCandidate, CMParams cmParams) method
1) A Bollinger band is created for the candidate and a grayscale image created.(i.e.)
dataProcessir.ProcessDate(testCase)

2) A prediction is then made.
(i.e.) cnnClient.Predict(resnet50,testCase.LastStream)

3) If a prediction result of 1 is returned then the score for the candidate is increased by UseCNNRewardPercentDecimal which is currently set to .20 or 20%.

Bibliography

Quantitative Momentum: A Practitioner's Guide to Building a Momentum-Based Stock Selection System
Jack R. Vogel and Wesley Gray
Wiley Finance

The Little Book That Still Beats the Market
Joel Greenblatt
Simon and Schuster

Trade Like a Stock Market Wizard
Marc Minervini
McGraw-Hill Education

Trade Your Way To Financial Freedom
Van K Tharp
McGraw Hill; 2nd edition (December 13, 2006)

Deep Learning for Vision Systems
Mohamed Elgendy
Manning Publications

Deep Learning
John D Kelleher
The MIT Press Essential Learning Series

Super Trader, Expanded Edition: Make Consistent Profits in Good and Bad Markets
Van K Tharp
McGraw Hill; 2nd edition (December 3, 2010)

Rule #1
Phil Town
Currency, August 28, 2007