**Tom Pichard – Capstone write-up**

1. **Background and Introduction**

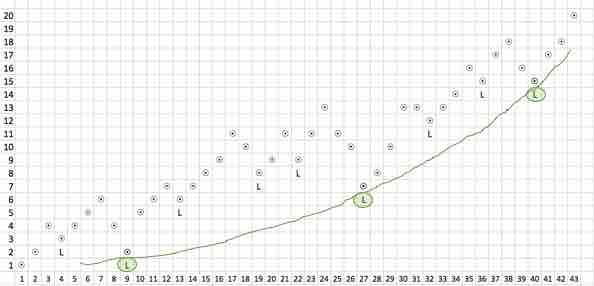
I have observed what I consider to be a non-traditional stock-commodity chart pattern which I call the parabolic acceleration curve. I witnessed it when working in the markets as an investment professional and associate it with accelerated gains after its completion. I have not validated this pattern from an objective data pattern analysis or system though.

The goal of this capstone was to code a solution that identifies this pattern, records its chart dates and presents it for examination to view/test any predicted future gains or not across a time series data sample.

The pattern is made up of successive lows in a stock-commodity progressing over time in a non linear channel as can be shown in two actual examples below.



A more idealized view of the pattern can be seen below, the key being not only higher lows but lows progressing upward in a non-linear pattern suggesting an acceleration or ‘wind-up’ in the making. I determined that initial parameters for the pattern would be approximated by three (3) successive lows.



1. **Data Fetching**

I was able to load historical stock time series data (IBM in this instance for the Capstone) leveraging the Yahoo\_Finance module (<https://pypi.python.org/pypi/yahoo-finance/1.2.1>) which allowed me to quickly retrieve IBM daily stock history of open, high, low and closes from January 2, 1962 consisting of 13,669 data rows. The module also natively creates a Pandas DataFrame with the time series which became the data baseline from which to perform subsequent transformations.

1. **Data Preparation**

I first had to re-sort and re-index the dataset as the import from Yahoo came in with the most current day at the top of the .csv file. The reason I did this was to conform to the processing protocol of charting applications including Matplotlib which process data in the source file from top to bottom leading to a left to right rendering of the time based data.

To note that I left the index as a sequence of increasing numbers starting from zero instead of integrating the dates into the index as is typically done with similar time-series. I did this for calculations purposes further downstream that will become clear in a later section.

During prep, I discovered that the only daily stock price that reflected historical continuity in value was the Adjusted\_Close. All other prices – the daily Open, High, Low, Close - were expressed in current prices.

The reason this was an issue is the following: in a nutshell, a stock’s value or price is based on the shares outstanding in the marketplace and when a company decides to split a stock the subsequent stock price after a split can fall dramatically i.e. for a stock trading at $100 and being split 2 for 1 the next day’s trading price will be $50.

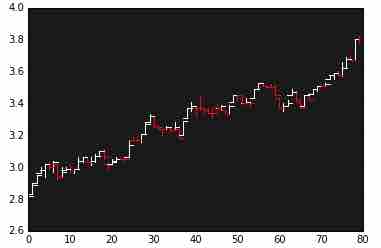
I basically had to write functions that re-adjusted the Open High Low Closes back to the Adjusted\_Close so that a continuous chart of true value could be generated and, for example, as shown below the Low of January 4 1962 would be in proportion to the Adjusted\_Close of 2.31 instead of being recorded as 570.99.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Symbol** | **Open** | **High** | **Low** | **Close** | **Adj\_Close** |
|  |  |  |  |  |  |  |
| 1962-01-02 | IBM | 578.49 | 578.49 | 572.00 | 572.00 | 2.32 |
| 1962-01-03 | IBM | 572.00 | 576.99 | 572.00 | 576.99 | 2.34 |
| 1962-01-04 | IBM | 576.99 | 576.99 | 570.99 | 571.25 | 2.31 |

1. **Data Visualization**

Next, in order to visualize results for spot checking and looking at patterns I had to find a method to draw bar charts for visualizing the stock over time.

I was able to leverage the Matplotlib Finance module found at (<http://matplotlib.org/api/finance_api.html> ) and generate historical bar charts.

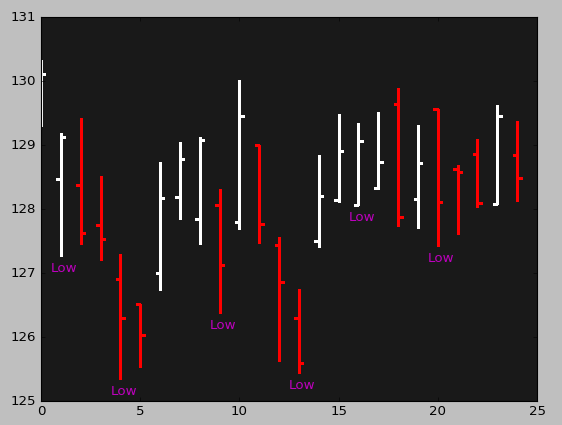


1. **Data Model Construction**

The next step was to identify each local low. I was able to code a simple enough algorithm to identify these and record them as a ‘LLOW’ in a separate DataFrame which was subsequently merged into the original Dataset.

The code up to here can be reviewed at <https://github.com/TomRene/TPCapstone/blob/master/Build001.ipynb>

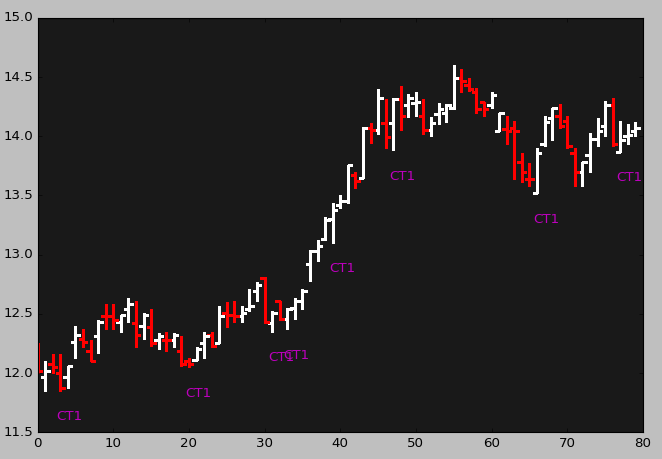
I then added text to the generated charts to visually validate on the algorithm’s accuracy.



The challenge I encountered in the next step of the model was how to model the pattern of 3 rising lows given that not all lows are created ‘equal’ in the sense that among the series of local lows which encompass all lows in the time-series some lows are more important than others with respect to their relative timeframe – please refer to the idealized pattern chart on page 1 where several local lows denoted with an ‘L’ are interspersed by the more major lows denoted by a circled L.

In the time I gave myself to figure this out I was not able to determine the (code) logic to identify these so I proceeded to code for any 3 lows contiguous to each other where each successive low was higher than the other i.e. the logic was that in the pattern I was coding to hunt for, the second low had to be higher than the first low and the third low had to be higher than the second low. I added these annotations on the day of the first low in the DataFrame and labeled these ‘CT1’ for Count Triplet first low.

I then made for the graphs to indicate these starting points of the pattern as shown in the example below where each ‘CT1’ denotes the beginning of a ‘3 consecutive rising lows pattern.’



I was not totally pleased with the noise surrounding the creation of these consecutive rising low patterns in the sense that sometimes there were 3 small lows connected that did not indicate the larger trend but I needed to move on….

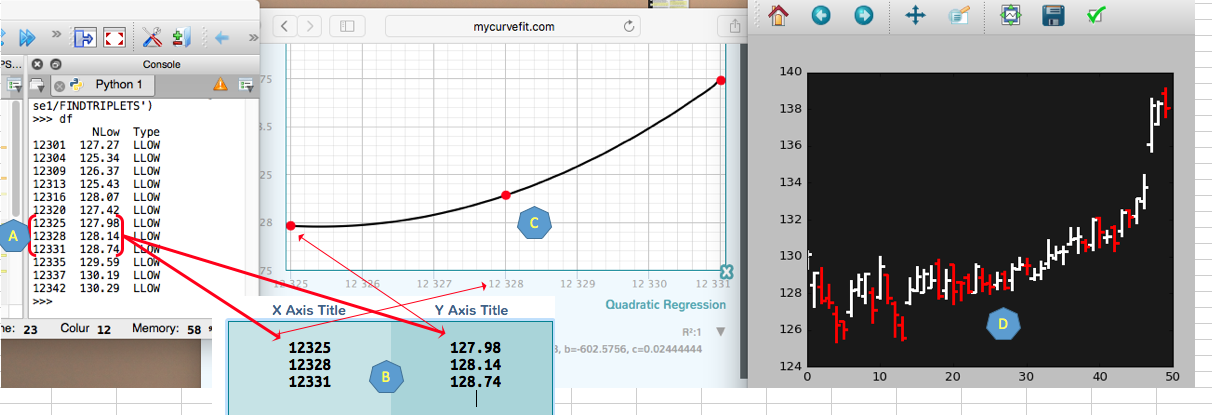
As a point of reflection and a call out to you, if you notice the first three CT1 patterns in the chart above – those starting at points ~3, ~20 and ~33 on the X axis – these three pattern-start points really resemble the pattern that I was originally looking for and you can perhaps see now that patterns can be fractal in the sense that patterns may resemble each other but at different orders of scale or magnitude and that some lows are really minor or short team versus others being more meaningful or of medium term relevance. So again, if you can imagine a method to identify rising intermediate lows or have any ideas surrounding this please share with me.

1. **Feature Construction**

At this point I went into the details of the three rising lows and imagined that the three low points would indeed be the minimum number of points to fit an exponential curve to…

I looked into this with curve fitting apps on the web. If you look at the picture below, you will notice:

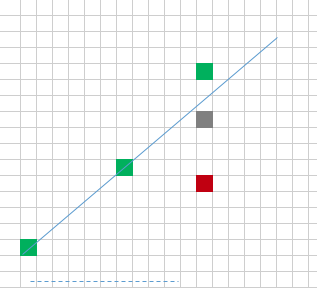
* In heptagon D, the actual chart pattern
* In heptagon A, the actual lows with their index placement
* In heptagon B, the inputs from A into a curve fitting approximator
* In heptagon C, the actual curve fitted with the quadratic equation regressed by the app.



In my effort to keep moving and simplify, I figure that I could approximate an exponential curve with three rising points as long as the third point was above the projected slope derived from the first two points. In the diagram below:

* The third green square fits this rule
* The gray square violates the rule but still fits the three higher lows framework
* The red square violates both rules.

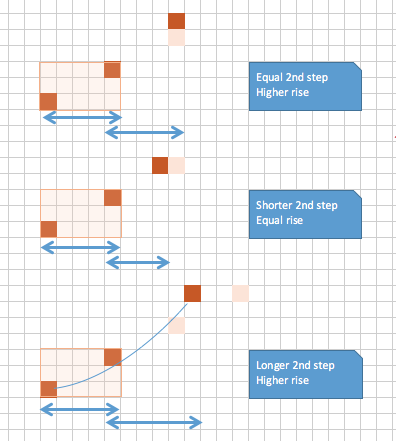
So the first metadata (or feature) point I added to the dataset was an Over Under attribute that identified if the CT1 pattern had its third low below or above the projected trendline from low1 and low2.



The next features I sought to extract from the pattern was the rise and run and slope for interval 1 (from low1 to low2) and interval 2 (from low2 to low3). I calculated these and added them as columns in the metadata dataset.

I added these because they were parameters embedded in the pattern and thought that it might be useful to have these as sub-components to compare. For instance, perhaps a successful pattern would contain a particular ratio of rise1 to rise2 and run1 to run2. This was purely an exploratory exercise but I coded and stored these features for each CT1 record.

The snapshot below illustrates this exploration:



1. **Target-Label Construction**

I have thought about constructing a buy sell method to record the predictive success of the model with a trailing stop and a market following rule to maximize profit but in the interest of time and at Jim’s suggestion I kept it simple and devised the following method to track the future price improvement/deterioration, the one subsequent to the ‘triplet low’ pattern.

Model elements

1. Entry Price: Record an entry price on the day following the third low at the Open of that day. (the reason the entry point is the day following the third low is because the third low can only be confirmed with the subsequent day’s stock performance – in retrospect, I made a mistake to use the Open because it is only at the Close that the following day can confirm that the third low of the previous day would stand and be counted as the third low. For matters of this proof of concept exercise, I don’t think its important.
2. Predictive Value 1: Take the length of the current CT1 pattern in days or index points and project this out from the day of the entry point. From that point forward, calculate the average of the next five days of Closes.
3. Predictive Value 2: Calculate another 5 day average of close prices starting at the end of the period for Predictive Value 1. I did this second value just to give some perspective and did an average of 5 days to smooth out any daily fluctuations to determine an average projected value.
4. Calculate the gross non-annualized % return: for the two predicted values, take each predicted value and divide its difference from the entry price and divide that by the entry price i.e. (Predicted Value – Entry Price) / Entry Price.

Summary stats

To give some perspective on the frequency of Local Lows and the Triplet pattern.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Stock** | **Rows/ Daily records** | **LLOWs (**Local Lows**)** | **Lows with a next higher low** | **CT1s (**3 consecutive higher lows**)** | **Os (**where the 3rd low is above the slope of interval 1**)** |
| IBM | 13669 | 2701 | 1387 | 714 | 340 |
|  | % of rows | 19.8% | 10.1% | 5.2% | 2.5% |
|  | % of previous |  | 51.4% | 51.5% | 47.6% |

The notebook for this second part of the project can be found at <https://github.com/TomRene/TPCapstone/blob/master/Build002.ipynb>

1. **Preliminary Review and Results**

* The Over Under indicator was not a meaningful distinction or feature as shown by the relative performance between the O and U with respect to win rate and Gain Loss for Predictive Value 1 or Scenario 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Over Under** | **Win Loss** | **# Sc1** | **%GainLoss** | Win % |
| O | L | 127 | (0.012450) |  |
| W | 213 | 0.015795 | 62.6% |
| U | L | 135 | (0.011281) |  |
| W | 239 | 0.017366 | 63.9% |
|  | Total | 714 |  |  |

In fact, disregarding that feature for Scenario 1 produced similar results for scenario 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Win Loss** | **# Sc1** | **%GainLoss** | Win % | **Total PL %** | **Net PL %** |
| L | 262 | (0.011848) |  | (3.1042) |  |
| W | 452 | 0.016626 | 63.3% | 7.5150 | 4.4108 |
| Total | 714 |  |  |  |  |

Net performance over the 54 years is a net 440% or a flat 8%.

This performance is actually encouraging because of the very gross nature of the performance measurement or exit rule/predictive window I constructed. The idea here is if a system was further refined to 1) control losses (using an algorithm for stop-losses) and ride the wins (using an algorithm to prolong positions in a rising market), I wouldn’t be surprised to be able to raise the % win by 25% or more.

Caveats to the analysis

* This was really a proof-of-concept using only one stock. Many more stocks would need to be scaled into the operationalization of this and the model would probably change as a result.

Questions to reviewers

* Could I use an estimator model using my metadata points/features to optimize the performance? Does this even make any sense due to the uncharacteristic nature of this project not utilizing estimators a-priori? But could any be used to tease out patterns or categories from the rise, run slope internal numbers?
* How could I code to find the string of the more major lows and connect them instead of finding just consecutive lows? Any thoughts on this would be appreciated ☺.

Next Steps

* Clean up the Git Hub and this document based on your feedback
* Add a feature or two if time permits
* Run the features through estimators but I really need to be convinced of this and what method to use let alone which model to use i.e. regression versus decision tree versus clustering?