Financial Machine Learning Part 0: Bars

Maks Ivanov · Follow
Published in Towards Data Science
9 min read · Feb 27, 2019

Preface

Recently, I got my copy of Advances in Financial Machine Learning by Marcos Lopez de Prado. Lopez de Prado is a renowned quant researcher who has managed billions throughout his career. The book is an amazing resource to anyone interested in data science and finance, and it offers valuable insights into how advanced predictive techniques are applied to financial problems.

This post is the first of a series dedicated to applying the approaches introduced by Lopez de Prado to real (and occasionally, synthetic) datasets. My hope is that by writing these posts I can solidify my understanding of the material and share some lessons learned along the way.

Without further ado, let's proceed to the main subject of this post: bars.

wrong kind of bar

What are bars?

The first step to building a great model is aggregating the data into a convenient format for further analysis. "Bars" refer to a data representation that contains the most basic information about price movements of a financial asset. A typical bar may contain features like the timestamp, peak price, opening and closing prices, etc.

HLC bar chart (stockcharts.com)

Bars are typically the data format used as the input for training and testing your ML predictor. One can easily imagine that the way the raw data is aggregated can have a significant downstream effect on the entire model.

Motivation

Although it may seem intuitive to work with price observations at fixed time intervals, e.g. every day/hour/minute/etc., it is not a good idea. Information flow through markets is not uniformly distributed over time, and there are some periods heightened activity, e.g. in the hour following the market open, or right before a futures contract expires.

We must aim for a bar representation in which each bar contains the same amount of information, however time-based bars will oversample slow periods and undersample high activity periods. To avoid this problem, the idea is to sample observations as a function of market activity.

Setup

Using a trade book dataset, we will construct multiple types of bars for an actual financial instrument. I will use the data for BitCoin perpetual swap contract listed on BitMex as XBT, because talking about BitCoin is the exciting thing to do these days and also because the trade book data is available here. We will compare time bars vs. tick bars, volume bars, dollar bars, and dollar imbalance bars. Python 3 snippets are provided to follow along.

First, a bit of setup:

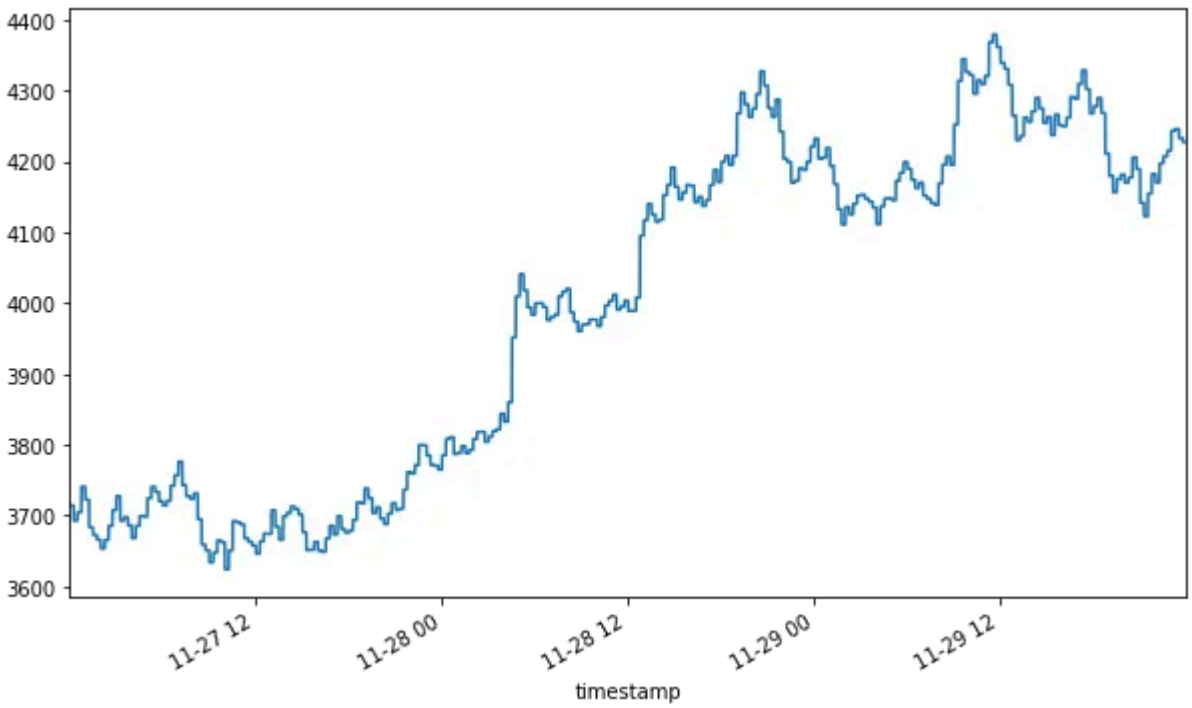
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from datetime import datetime

# raw trade data from https://public.bitmex.com/?prefix=data/trade/
data = pd.read_csv('data/20181127.csv')
data = data.append(pd.read_csv('data/20181128.csv')) # add a few
more days
data = data.append(pd.read_csv('data/20181129.csv'))
data = data[data.symbol == 'XBTUSD']
# timestamp parsing
```

```
data['timestamp'] = data.timestamp.map(lambda t:
datetime.strptime(t[:-3], "%Y-%m-%dD%H:%M:%S.%f"))
```

Time Bars

We've now loaded a few days worth of trade data for the XBTUSD ticker on BitMex. Let's see what the volume weighted average price looks like when computed in 15 minute intervals. As previously mentioned, this representation isn't synchronized to market information flow — however, we will use it as a benchmark to compare against.

```
def compute_vwap(df):
    q = df['foreignNotional']
    p = df['price']
    vwap = np.sum(p * q) / np.sum(q)
    df['vwap'] = vwap
    return df

data_timeidx = data.set_index('timestamp')
data_time_grp = data_timeidx.groupby(pd.Grouper(freq='15Min'))
num_time_bars = len(data_time_grp) # comes in handy later
data_time_vwap = data_time_grp.apply(compute_vwap)
```



XBT time bars

Note that we saved the number of bars in the final series. For comparing different methods, we want to make sure that we have roughly the same resolution so that the

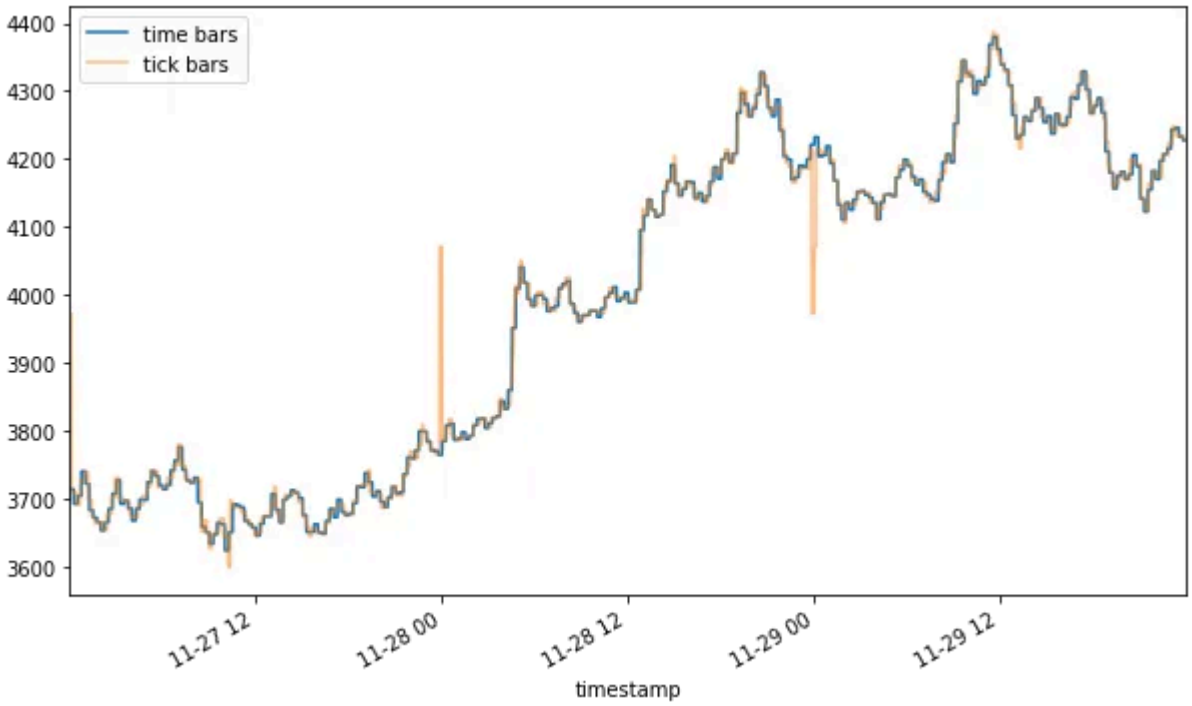
comparison is fair.

Tick Bars

The idea behind tick bars is to sample observations every N transactions, aka “ticks”, instead of fixed time buckets. This allows us to capture more information at times when many trades take place, and vice-versa.

```
total_ticks = len(data)
num_ticks_per_bar = total_ticks / num_time_bars
num_ticks_per_bar = round(num_ticks_per_bar, -3) # round to the
nearest thousand
data_tick_grp = data.reset_index().assign(grpId=lambda row:
row.index // num_ticks_per_bar)

data_tick_vwap =  data_tick_grp.groupby('grpId').apply(compute_vwap)
data_tick_vwap.set_index('timestamp', inplace=True)
```

How does this compare to the time bar series?



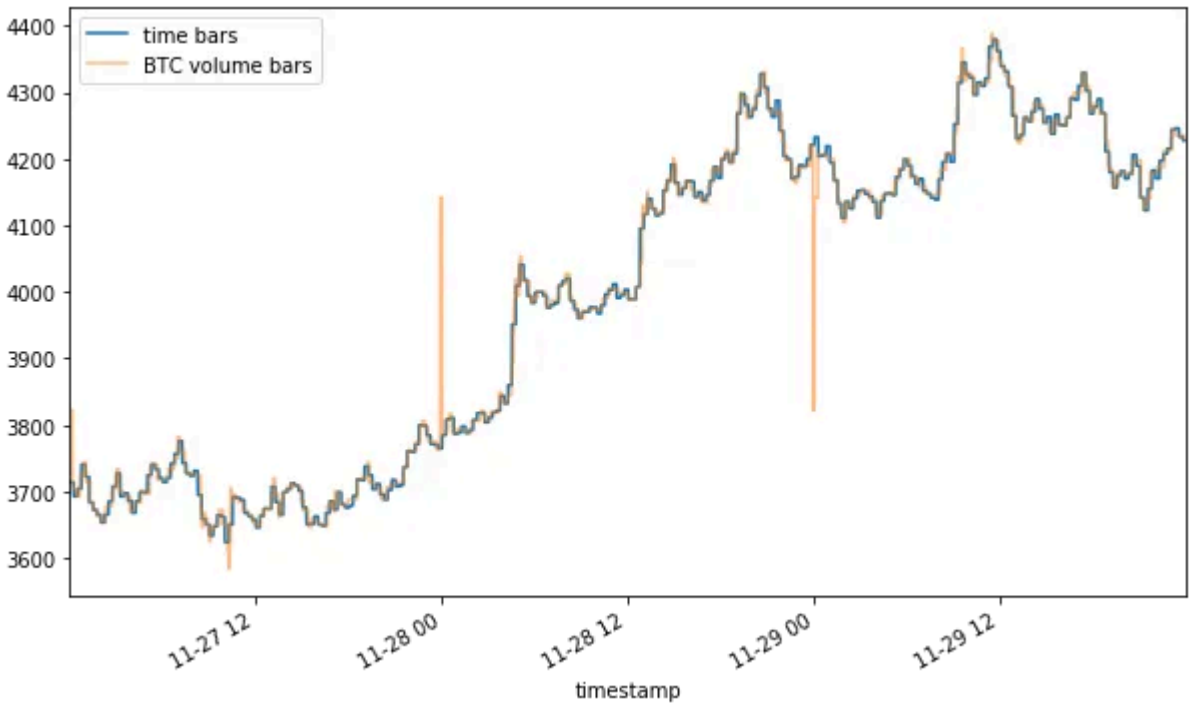
XBT time and tick bars

Plotting the two together, you may notice a flash rally and a flash crash (yellow) of ~10% that were hidden in the time bar representation (blue). Depending on your strategy, these two events could mean a huge trading opportunity (mean reversion) or a trading cost (slippage).

Volume Bars

One shortcoming of tick bars is that not all trades are equal. Consider that an order to buy 1000 contracts is executed as one transaction, and 10 orders for 100 contracts will count for 10 transactions. In light of this somewhat arbitrary distinction, it may make sense to sample observations for every N contracts exchanged independent of how many trades took place. Since XBT is a BTC swap contract, we will measure the volume in terms of BTC.

```
data_cm_vol = data.assign(cmVol=data['homeNotional'].cumsum())
total_vol = data_cm_vol.cmVol.values[-1]
vol_per_bar = total_vol / num_time_bars
vol_per_bar = round(vol_per_bar, -2) # round to the nearest hundred

data_vol_grp = data_cm_vol.assign(grpId=lambda row: row.cmVol //
vol_per_bar)

data_vol_vwap =  data_vol_grp.groupby('grpId').apply(compute_vwap)
data_vol_vwap.set_index('timestamp', inplace=True)
```



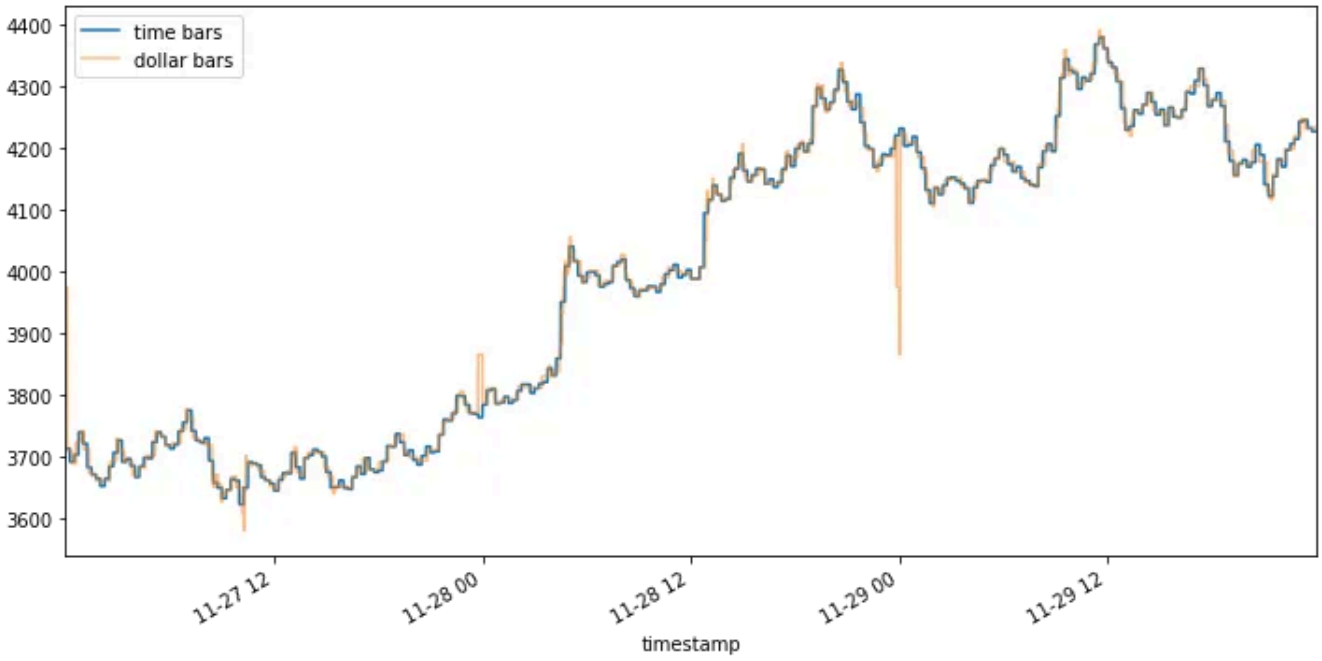
XBT time and volume bars

Note that the volume representation shows an even sharper rally and crash than the tick one (4100+ vs 4000+ peak and ~3800 vs 3900+ trough). By now it should become apparent that the method of aggregation chosen for your bars can affect the way your data is represented.

Dollar Bars

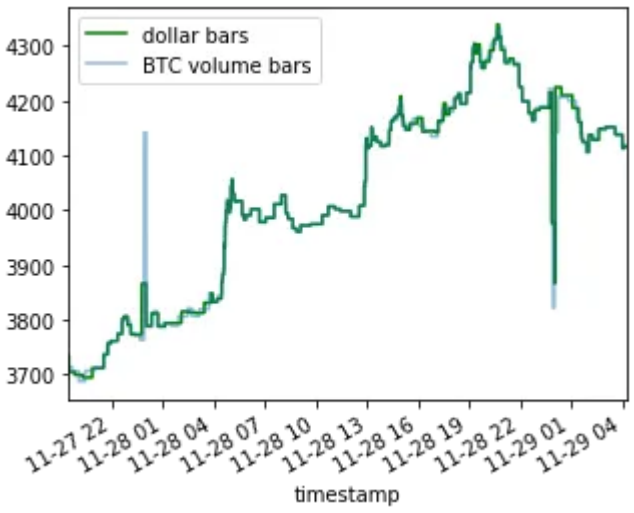
Even with the tiny dataset used here, you might notice that sampling the data as a function of number of BTC traded doesn't make sense when the value of BTC relative to USD moves more than 20% in just 3 days. Buying 1 BTC on the morning of 11–27 was a significantly different decision than buying 1 BTC the night of 11–29. Such price volatility is the rationale behind dollar bars — sampling as a function of dollars (or a currency of your choice) exchanged should in theory make the frequency more robust to value fluctuations.

```
# code omitted for brevity
# same as volume bars, except using data['foreignNotional'] instead
of data['homeNotional']
```



XBT dollar bars

Note that the BTC volume bars show nearly identical jumps around 11–28 00 and 11–29 00, however the initial spike in dollar bars on 11–28 00 looks relatively mild compared to the later one.



dollar bars vs BTC volume bars

This is a prime example of differences induced by sampling — even though many bitcoins have changed hands around 11–28 01, their dollar value was relatively lower at that time and so the event is represented as less severe.

Imbalance bars

Imbalance bars are the type of bars MLDP calls “information-driven”. These extend the ideas of alternative bars to more advanced approaches. Imbalance bars in particular try to sample when there is an unusual imbalance of buying/selling activity, which may imply information asymmetry between market participants. The rationale is that informed traders either buy or sell in large quantities, but rarely do both at the same time. Sampling when imbalance events occur allows us to focus on large moves and ignore less interesting periods.

Implementing Dollar Imbalance Bars

Implementing imbalance bars warrants a more detailed explanation. Given dollar volume and prices for each tick, the process is:

1. Compute signed flows:

- Compute tick direction (the sign of change in price).
- Multiply tick direction by tick volume.

2. Accumulate the imbalance bars :

- Starting from the first datapoint, step through the dataset and keep track of the cumulative signed flows (the imbalance).

- Take a sample whenever the absolute value of imbalance exceeds the expected imbalance threshold.
- Update the expectations of imbalance threshold as you see more data.

Let's expand each of these steps further.

1.1 Compute tick direction:

Given a sequence of N ticks $\{ (p[i], v[i]) \}$ for $i \in 1...N$ where $p[i]$ is the associated price and $v[i]$ is the dollar volume, we first compute change in price from tick to tick, and then define the sequence $\{b[i]\}$ for $i \in 1...N$:

$\Delta p[i] := p[i] - p[i-1]$

$b[i] := b[i-1]$ if $\Delta p[i] = 0$

$b[i] := \text{sign}(\Delta p[i])$ otherwise

Luckily in our dataset the tick directions are already given to us, we just need to convert them from strings to integers.

```
def convert_tick_direction(tick_direction):
    if tick_direction in ('PlusTick', 'ZeroPlusTick'):
        return 1
    elif tick_direction in ('MinusTick', 'ZeroMinusTick'):
        return -1
    else:
        raise ValueError('converting invalid input: '+
str(tick_direction))

data_timeidx['tickDirection'] =
data_timeidx.tickDirection.map(convert_tick_direction)
```

1.2 Compute signed flows at each tick:

Signed Flow$[i] := b[i] * v[i]$ is the dollar volume at step i

```
data_signed_flow = data_timeidx.assign(bv =
data_timeidx.tickDirection * data_timeidx.size)
```

2. Accumulate dollar imbalance bars

To compute dollar imbalance bars, we step forward through the data, tracking the imbalance since the last sample, and take a sample whenever the magnitude of the imbalance exceeds our expectations. The rule is expanded below.

Sample bar when:

$|Imbalance| \geq Expected\ imbalance$

where

*Exp. imbalance := (Expected # of ticks per bar) * |Expected imbalance per tick|*

We define the imbalance for a subset of t ticks as $\theta[t] := \sum b[i] * v[i]$ *for* $i \in 1 \ldots t$

Let T denote the number of ticks per bar, which is not constant. Then, $E_o[T]$ is the expected number of ticks per bar, which we estimate as the exponentially weighted moving average of T values from prior bars.

Finally, we estimate the expected imbalance per tick, $E_o[b*v]$, as the exponentially weighted moving average of b[i]*v[i] values from prior bars.

Putting it all together, we must step iterate over the dataset, and take samples every T^* ticks, defined as

$T^* := argmin(\ t\ )\ s.t.\ |\theta[t]| \geq E_o[T]\ * |E_o[b*v]|$

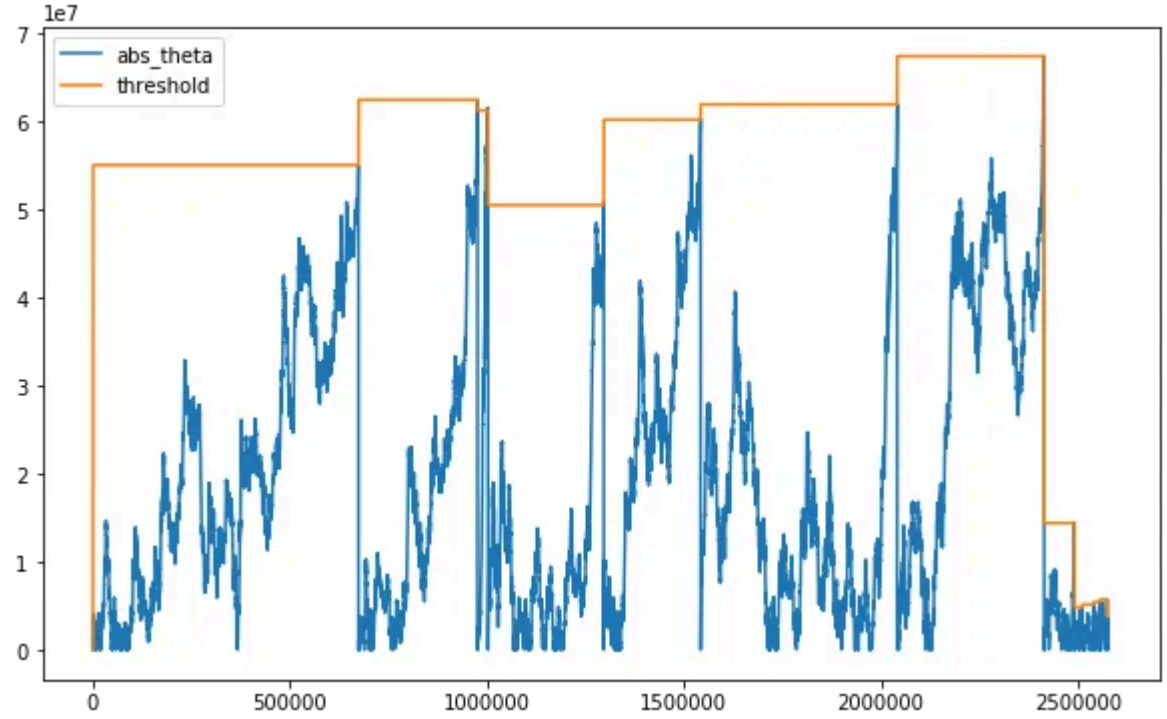
Important caveats of this procedure:

- At the start, we don't have any previous bars to base our estimates on, so we must come up with initial values for computing the first threshold.
- As the algorithm accumulates more bars, the EWMA estimates "forget" the initial values in favor of more recent ones. Make sure you set a high enough initial values so that the algorithm has a chance to "warm up" the estimates.
- The algorithm can be quite sensitive to the hyperparameters used for EWMA. Because there is no straightforward way to get the same number of bars as in the previous demos, we will just pick the most convenient/reasonable hyperparameters.

With that in mind, let's put the logic into code. I use a fast implementation of EWMA sourced from stackexchange.

```
from fast_ewma import _ewma

abs_Ebv_init = np.abs(data_signed_flow['bv'].mean())
E_T_init = 500000 # 500000 ticks to warm up

def compute_Ts(bvs, E_T_init, abs_Ebv_init):
    Ts, i_s = [], []
    i_prev, E_T, abs_Ebv  = 0, E_T_init, abs_Ebv_init

    n = bvs.shape[0]
    bvs_val = bvs.values.astype(np.float64)
    abs_thetas, thresholds = np.zeros(n), np.zeros(n)
    abs_thetas[0], cur_theta = np.abs(bvs_val[0]), bvs_val[0]

    for i in range(1, n):
        cur_theta += bvs_val[i]
        abs_theta = np.abs(cur_theta)
        abs_thetas[i] = abs_theta

        threshold = E_T * abs_Ebv
        thresholds[i] = threshold

        if abs_theta >= threshold:
            cur_theta = 0
            Ts.append(np.float64(i - i_prev))
            i_s.append(i)
            i_prev = i
            E_T = _ewma(np.array(Ts), window=np.int64(len(Ts)))[-1]
            abs_Ebv = np.abs( _ewma(bvs_val[:i],
window=np.int64(E_T_init * 3))[-1] ) # window of 3 bars

    return Ts, abs_thetas, thresholds, i_s

Ts, abs_thetas, thresholds, i_s = compute_Ts(data_signed_flow.bv,
E_T_init, abs_Ebv_init)
```

Let's plot $|\theta[t]|$ and the imbalance threshold ($E_0[T] * |E_0[b^*v]|$) to see what's going on.



Threshold vs. magnitude of imbalance

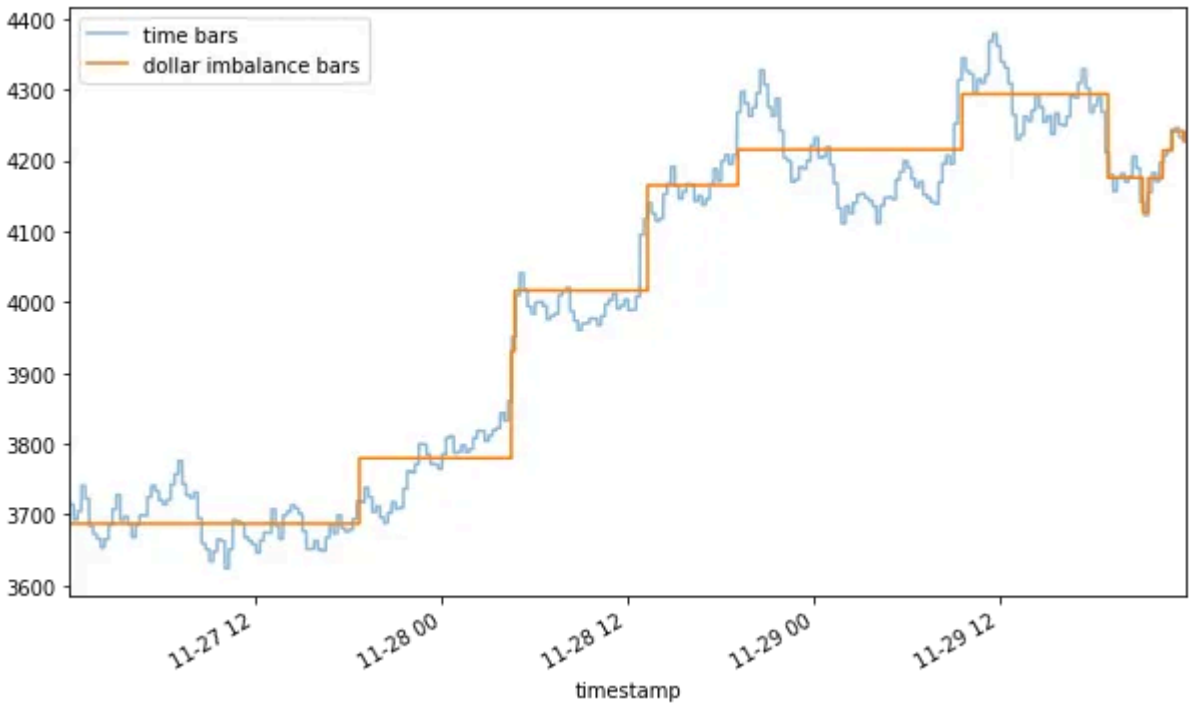
Seems like the sampling frequency is high near where the upward trend picks up and and also near where that same trend reverses. Before we can visualize the bars, we need to group the ticks accordingly.

Aggregate the ticks into groups based on computed boundaries

```
n = data_signed_flow.shape[0]
i_iter = iter(i_s + [n])
i_cur = i_iter.__next__()
grpId = np.zeros(n)

for i in range(1, n):
    if i <= i_cur:
        grpId[i] = grpId[i-1]
    else:
        grpId[i] = grpId[i-1] + 1
        i_cur = i_iter.__next__()
```

Putting it all together: Dollar Imbalance Bars

```
data_dollar_imb_grp = data_signed_flow.assign(grpId = grpId)
data_dollar_imb_vwap =
data_dollar_imb_grp.groupby('grpId').apply(compute_vwap).vwap
```



Dollar imbalance bars

We see that DIBs tend to sample when a change in trend is detected. It can be interpreted as DIBs containing the same amount of information about trend changes, which may help us to develop a model for trend following.

Summary

We've used a trade book dataset to compute time, tick, dollar, volume, and dollar imbalance bars on a BTC swap contract. Each alternative approach tells a slightly different story, and each has advantages that depend on the market microstructure and particular use cases.

To explore this further, consider measuring the statistical properties of each bar series such as kurtosis and serial correlation to see which bars would be easier to model with an ML algorithm. I hope you enjoyed this demo and please reach out if you catch a mistake or have any questions!

Machine Learning | Trading | Bitcoin | Visualization | AI

Follow

Written by Maks Ivanov

766 Followers · Writer for Towards Data Science

linkedin.com/in/maksivanov | Opinions are my own

More from Maks Ivanov and Towards Data Science

Maks Ivanov in Towards Data Science

Financial Machine Learning Part 1: Labels

Setting up a supervised learning problem

10 min read · Mar 18, 2019

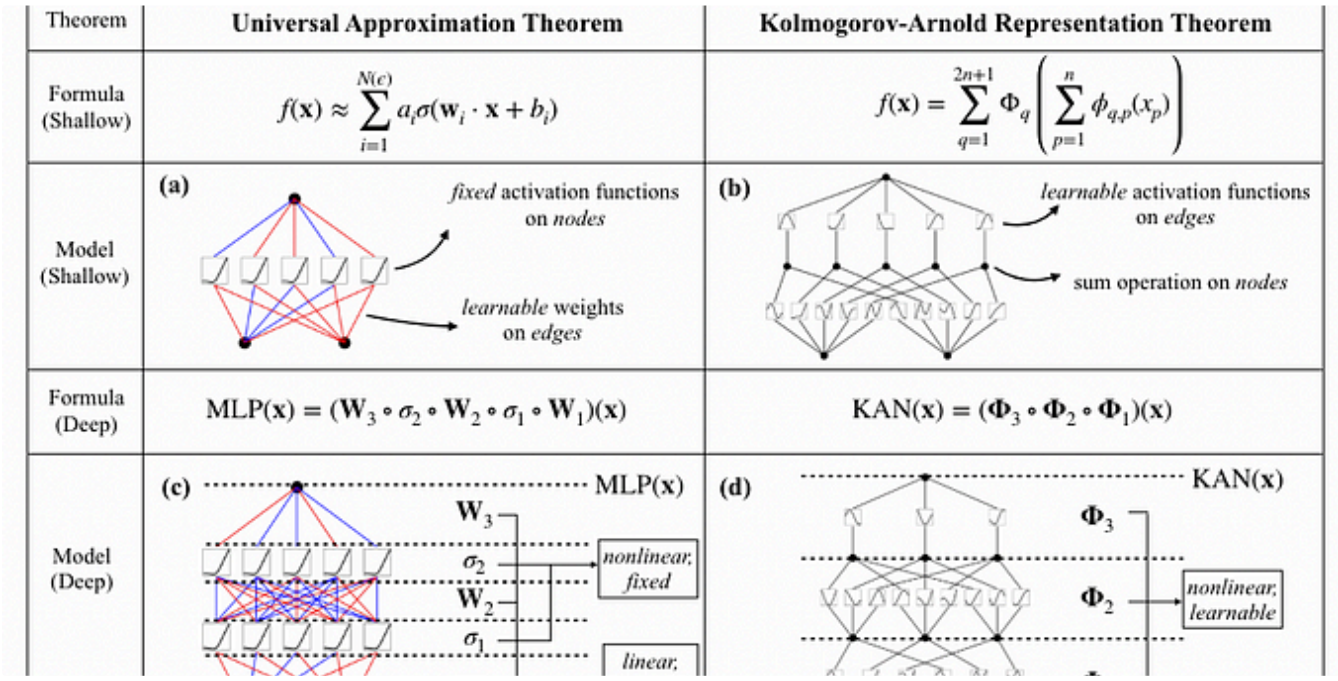
937 8

Torsten Walbaum in Towards Data Science

What 10 Years at Uber, Meta and Startups Taught Me About Data Analytics

Advice for Data Scientists and Managers

9 min read · May 30, 2024



Theo Wolf in Towards Data Science

Kolmogorov-Arnold Networks: the latest advance in Neural Networks, simply explained

The new type of network that is making waves in the ML world.

✦ · 9 min read · May 12, 2024

2.1K 18

Egor Howell in Towards Data Science

How I Use ChatGPT As A Data Scientist

How ChatGPT improved my productivity as a data scientist

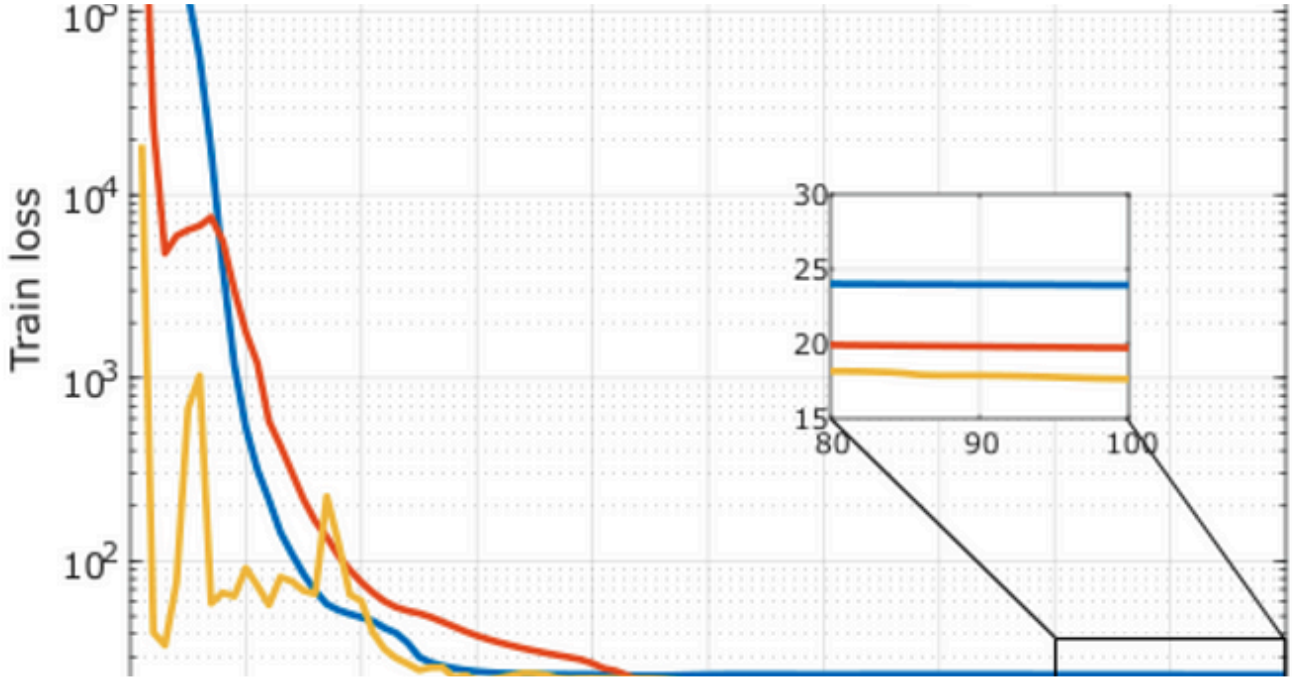
✦ · 8 min read · Jun 2, 2024

623 20

See all from Maks Ivanov

See all from Towards Data Science

Recommended from Medium



Sercan Bugra Gultekin

(2) Stock Price Prediction with ML in Python: GRU (Gated Recurrent Unit) Model

In our first series, I made a forecast with the LSTM model, now we will change our model and try the same experiment with GRU (Gated...

✦ · 3 min read · May 31, 2024

1

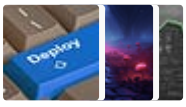
Cristian Velasquez

Riding the Waves of Stock Prices with Wavelet Transform Signals in Python

Towards Unlocking Market Signals for Clearer Trading Insights

· 9 min read · Sep 11, 2023

Lists



Predictive Modeling w/ Python

20 stories · 1316 saves

Practical Guides to Machine Learning

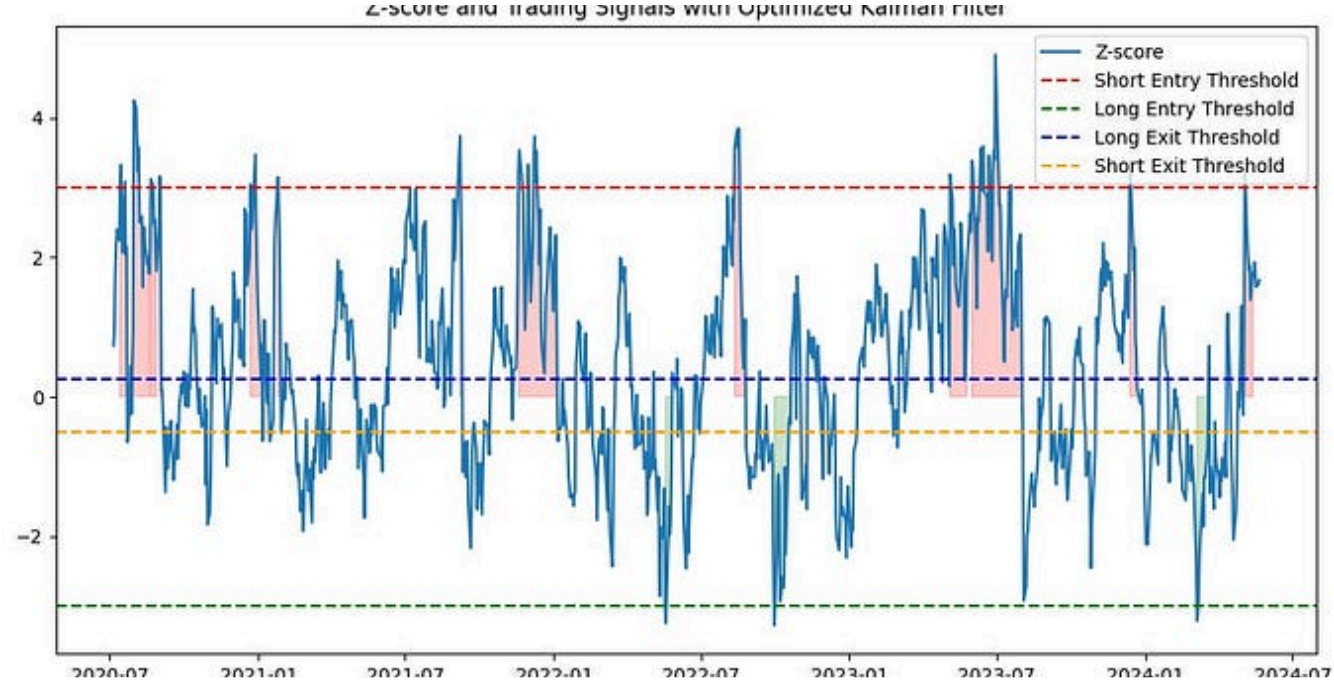
10 stories · 1580 saves

Natural Language Processing

1537 stories · 1071 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 412 saves



Serdar İlarslan

Implementing a Kalman Filter-Based Trading Strategy

Financial markets are inherently noisy and unpredictable, making it challenging for traders and investors to identify and capitalize on...

7 min read · May 25, 2024

163 2

Kaan Aslan

Time Series Forecasting with a Basic Transformer Model in PyTorch

Time series forecasting is an essential topic that's both challenging and rewarding, with a wide variety of techniques available to...

10 min read · Jan 11, 2024

132 2

Fisher Lok in InsiderFinance Wire

Trading from First Principle 3: Building a Machine Learning-Based Trading Strategy

This article proposed a standardized approach to produce a trading strategy based on machine learning models. A systematic approach is...

13 min read · Feb 13, 2024

177