AI for algorithmic trading: rethinking bars, labeling, and stationarity

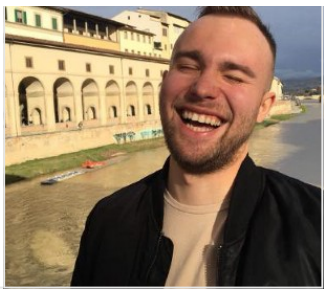
Image from https://www.tradingsetupsreview.com/trading-charts-without-time-range-tick-volume/

In a series of articles I was applying a very straightforward approach to forecast financial time series: take the whole dataset, using a sliding window approach generate X and Y, split it into historical and out-of-sample data, train some machine learning models to map X to Y and backtest simple long-short strategy. But as I showed in the last blog post I started to realize that pipeline for "normal" static data like images, text, audio, tabular data or even less chaotic time series can't be used for financial time series analysis.

The problem is not just that data is stochastic and difficult to forecast. It's all about the total misunderstanding of its inner nature, which influences dataset preparation, cross-validation, feature selection, and backtesting. In this article, we will concentrate on well known "bars", what's wrong with them and how to cook them correctly to feed later inside a machine learning model. After the recreation of the bars, we will discover several new ways to build inputs and outputs from them. Of course, we will compare approaches statistically and empirically. The ideas that I am showing here are highly influenced by **the book of Lopez de Prado**, that I recommend reading to anyone who wants to dig into more details. You also can find all the code here:

Mostly experiments based on "Advances in financial machine learning" book - Rachnog/Advanced-Deep-Trading

github.com



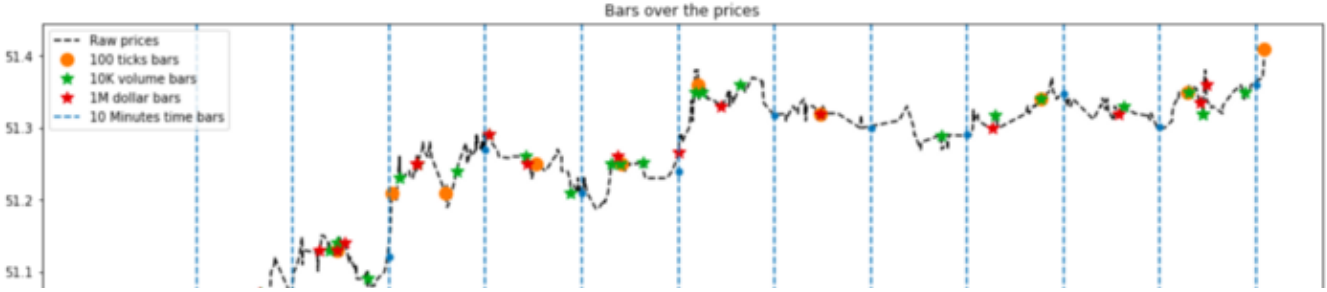
What's wrong with the candles?

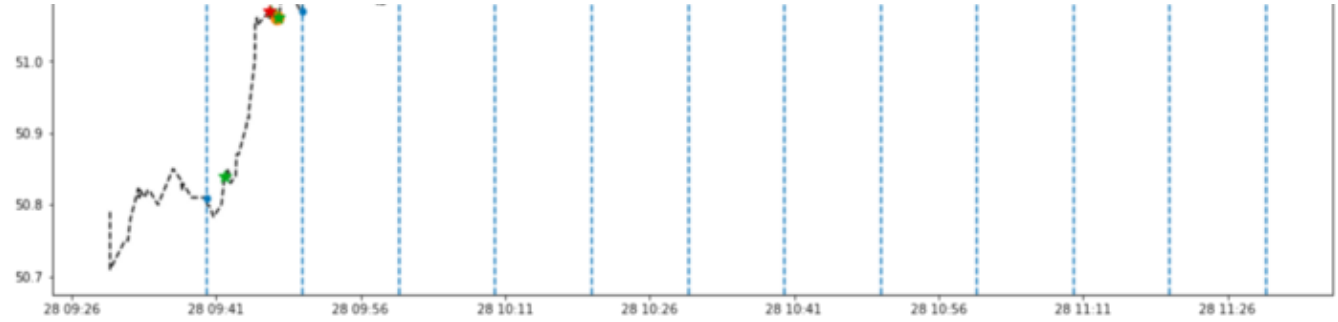
We are used to working with so-called "candles" data that represent open, high, low and close prices for some specific period of time (from minutes to days). There is one major problem with this approach: the market doesn't follow this timing rule. People don't place and do trades when some time comes or every N minutes/hours. Moreover, since modern markets are driven by algorithms, they definitely place their bets when they need to, **not when N seconds passed**. What happens if we sample historical data based on constant time intervals — we undersample when there are very active periods of trading and oversample when there are some periods with low activity. Last but not least, from all above mentioned follows (empirically), that bars sampled by time follow **"bad" statistical properties**: they have a low serial correlation, have outliers and fail distribution normality tests.

The solution lies in sampling these bars based on another kind of rules:

- **Tick bars:** sample an OHLC bar when N ticks appeared
- **Volume bars:** sample an OHLC bar when X shares of an asset have been traded
- **Dollar bars:** sample an OHLC bar when asset have been traded on Y dollars (or other currency)
- **Imbalance bars:** sample an OHLC bar when the ratio of ups/downs of prices or of buys/sells diverges from our expectations

You can have an intuition about how these bars are being produced looking on the picture below on a sample time series:

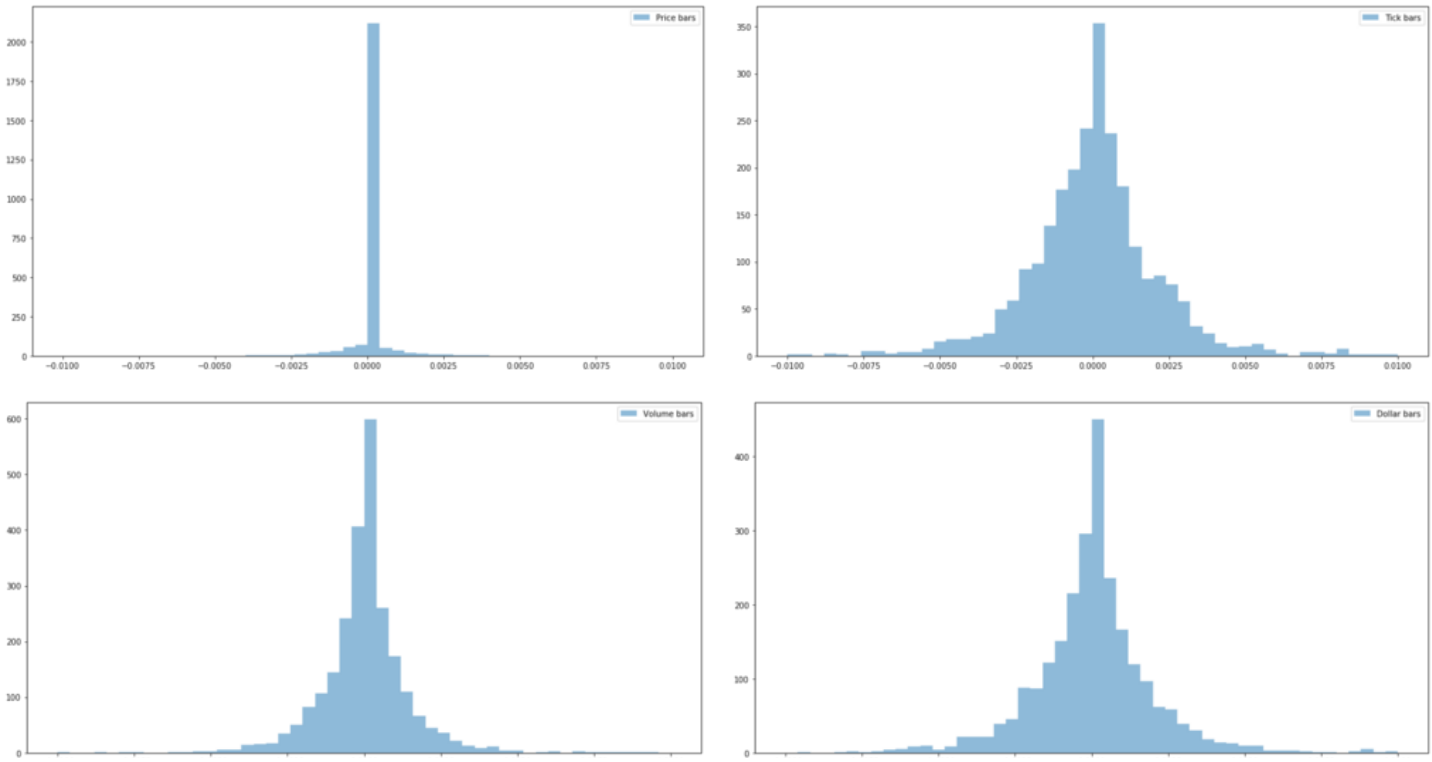




Comparison of some of the above-mentioned bars on IVE ticker

All above-mentioned approaches are designed to create bars with an **equal amount of information inside it**: tick bars have the same number of trades, but volume can be very different, so we maybe better sample volume bars, that have an equal amount of shares traded each. Dollar bars extend this ideas chain to have bars each having the same amount of market value exchanged. Imbalance bars can be applied to tick, volume or dollar bars. Let me explain the idea behind tick imbalance bars. Imbalance tick bars are samples from N ticks that follow similar prices trend.

As you can see, all these ideas rely heavily on market microstructure to rebuild the time series we already know a lot about. Let's check if these bars are really so good as they look like. They're not so many sources of market microstructure data that have bid, ask and size columns, I could just find this tutorial on the same topic, but I'll implement most of the methods myself. Here we have time bars sampled every 10 minutes, tick bars sampled every 100 bars, volume — every 10000 trades and dollar — each $1000000 traded (all over several months):



Comparison of distributions of returns with different bar rules

What about **statistical properties**? Below you can find a comparison of different bars with different parameters alongside their serial correlation, standard deviations, and normality tests.

Search this file…

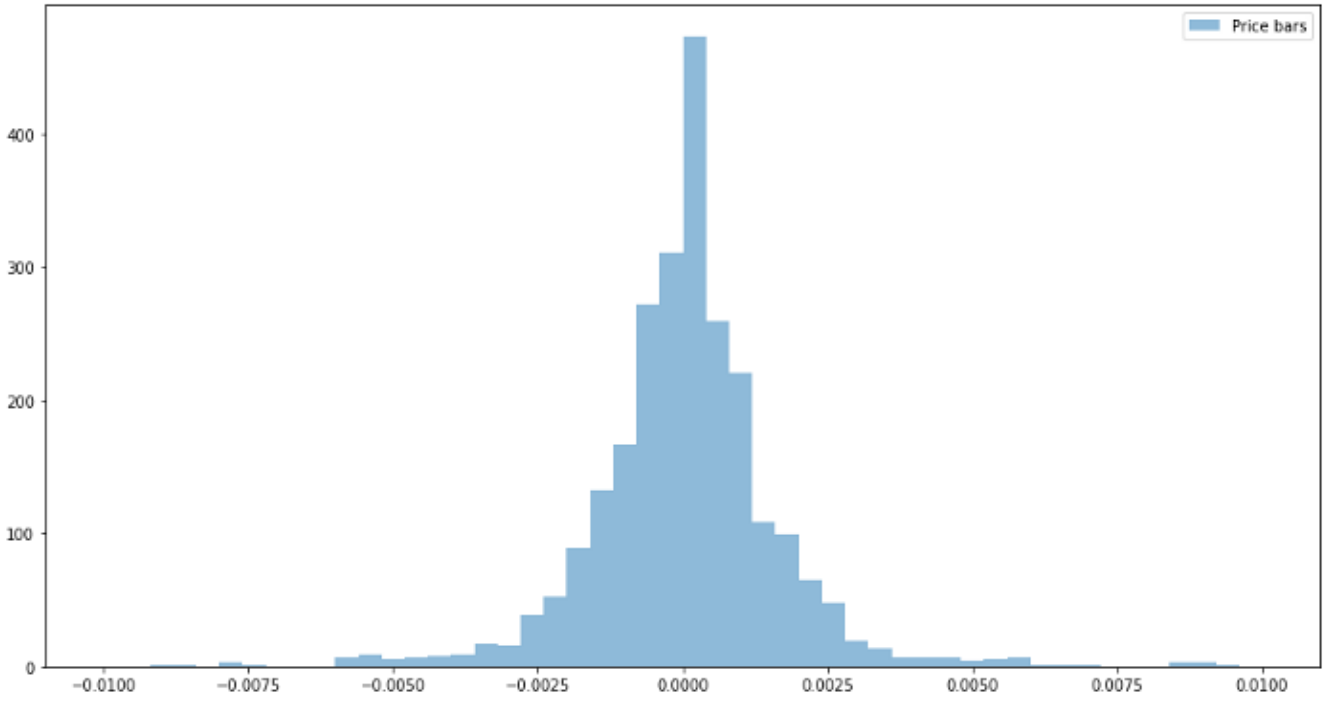
	Serial correlation	Variance	Jarque-Bera	Shapiro
Time bar	-0.0218	5.8e-07	24258229	0.3583
Tick bar	-0.0329	5.6e-06	17138	0.8961
Volume bar	-0.0226	2.5e-06	161134	0.8318
Dollar bar	-0.0377	4.8e-06	22318	0.8763

bars_tests.csv hosted with ❤ by **GitHub** **view raw**

As we can see, for every parameter **alternative bars behave better** than time bar (apart of the variance, which is very low for all anyway). How could I be using time bars before? :(

!UPDATE!

A GitHub user https://github.com/mpugna made a fix in the code, that makes time bars still very appealing:



The autocorrelation is -0.079, Jarque-Bera 78692.65, Shapiro 0.7924. Normality tests are still worse than in alternative bars, but at least it looks much more appealing now.

How to label dataset correctly?

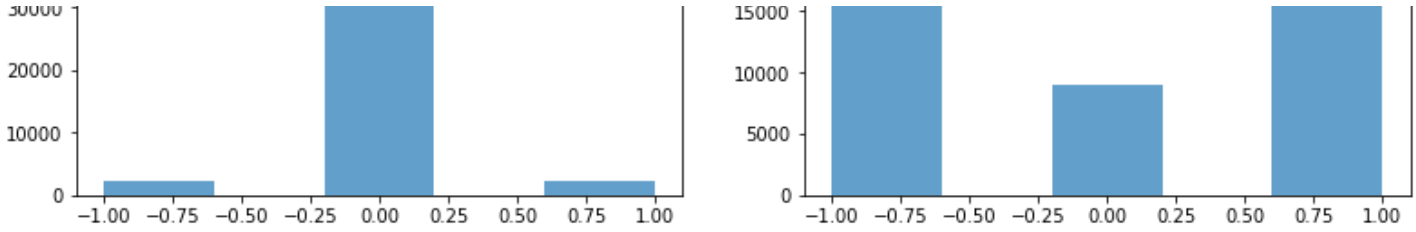
In my previous posts, I was predicting how the price will change after N bars pass. For example, I want to predict how the price will change after the next 30 minutes pass and go long or short accordingly to the forecast. But is it really how the practitioners and traders act? When they open the position after some signal, they keep in mind what their take profit goals and stop losses are. It means, that we care more about what happens inside these 30 minutes, but not what will happen right when they pass. Moreover, my profit and loss targets may change over time because of the volatility of the market. Also, since I am going to bet in some direction where market price will move I need to know how much should I bet. Last but not least, I am interested in the confidence of my prediction not to waste time on random signals from the point of view of my model.

Hence, I am interested in a new labeling method that includes:

- Taking into account the **rolling volatility** of the returns
- Taking into account **stop loss and take profit**
- Tells me not just side but **size of a bet**

Dynamic thresholds

Let's start first with the volatility of returns. Let's consider a simple framework where we still have a fixed horizon of forecasting and we want to label three classes: if the return of current price and the future is more than some threshold **T** (like r = 0.2 with T = 0.1), less than -T (like r = -0.12 with T = 0.1), or is not significant (e.g. sign of return is less than T, like r = 0.05 with T = 0.1). We can have this T fixed for the whole dataset or we can calculate it adaptively using the standard variation of returns. Let's check this idea and choose fixed T = 0.025 threshold for the return of horizon of 60 tick bars and compare it to T, that equals to the standard deviation of an absolute value of last 100 returns:

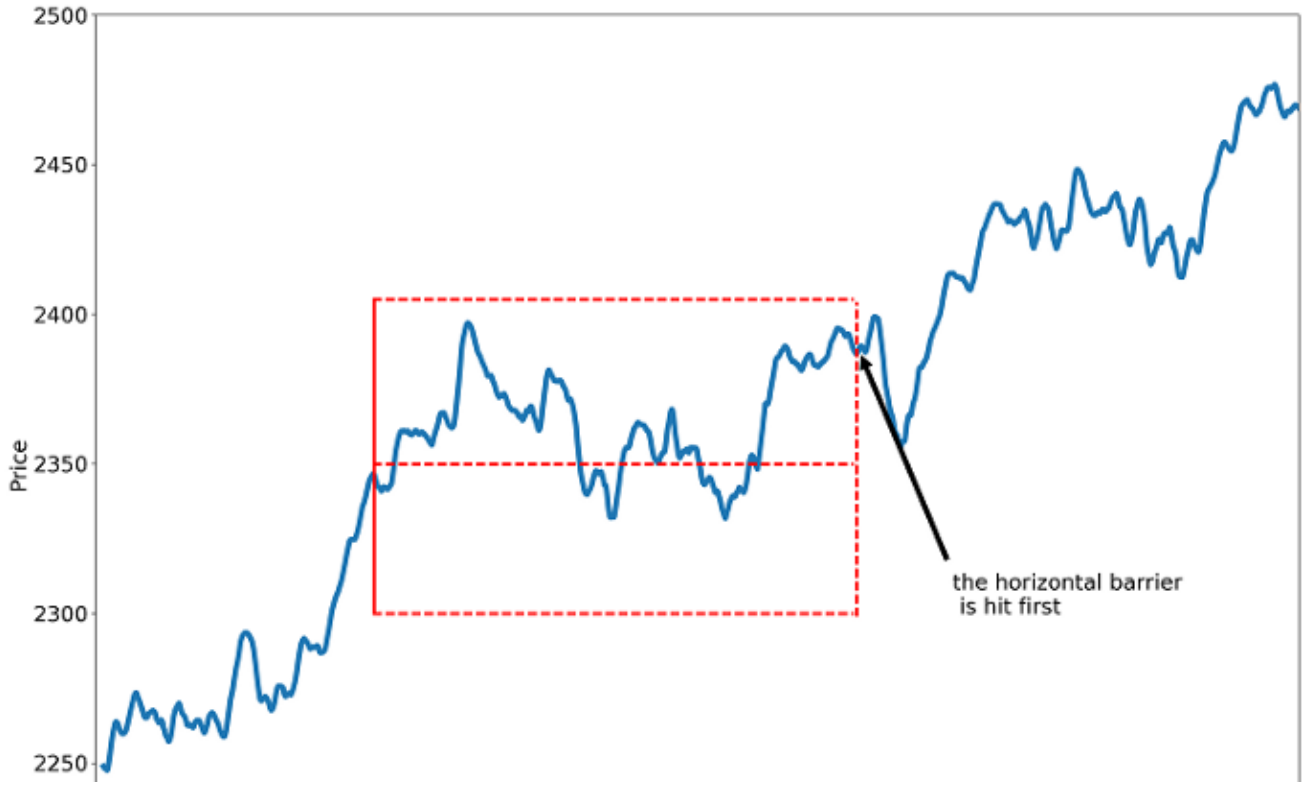


Histogram of labels for fixed threshold (left) and volatility driven labeling (right) for the fixed horizon

As we can see, volatility based labeling is still not very balanced, but fixed-threshold labeling **basically makes most of the labels 0** (where -1 is going down, 1 is going up and 0 — not significant movement). I just want to notice, that these labels were created while looping over bars with sliding window, which is not the most optimal way to sample data points, but we will come to this issue later.

Triple barrier labeling

Now let's research another kind of adaptiveness for labeling the dataset that is called in the book "**triple barrier method**". We want to know what will happen during next N bars — will we meet the stop-loss situation? Or maybe we supposed to take profit? Or maybe the price will just oscillate a little so we just better don't make a bet? Or maybe even combinations of these events? We can describe these three situations with three barriers: two horizontal (that represent stop loss and take profit) and one vertical which will mean the final horizon (fixed horizon in our previous example). Of course, these horizontal barriers shouldn't be symmetric (for example you're an aggressive player and your stop loss is pretty low comparing to the take profit goal).



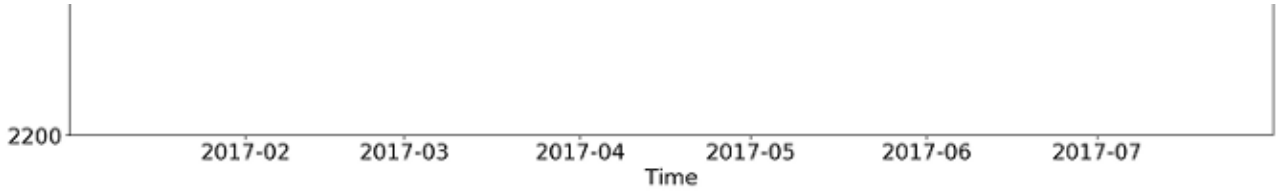


Image from http://www.quantresearch.info/Innovations.htm

Size of a bet and meta-labeling

Alright, we have some smart ways to label if the price will go up or down taking into account volatility and our stop losses and profit taking goals. But we still don't know how much we should bet (and if we should bet at all)! Lopez de Prado offers two-step labeling, where one label is responsible for the direction and second one — for the certainty of this bet (and, consequently, in size of the bet):

- Label each sample as the one, that will first go up or down using a dynamic threshold as a **preliminary stop loss and take profit barrier** (basically we want to know will the price go up or down first)
- After knowing this, we want to bet or no based on our **predefined stop loss and take profit goal**, so, if our first label says "up", we are checking if we will also hit the take profit goal and if we have confirmation, we set the second label as 1. If we have the first label "down" and we gonna hit the stop loss — we still label it as 1. Only if we don't have a correspondence between the direction from the first label and stop loss or take profit — we label it as zero.

And the coolest part. While preparing a dataset, we train on our set of features a model the predicts a side (label one). And we also train a second model, that has as input all previous features and label one as well. So after having the prediction of a direction from a first model, we want to know the level of a certainty with the second model — and based on its output that is from 0 to 1 — we make an appropriate bet. For example, if the first model says "up", but second says something like 0.05, it means that even the price will go up, most probably we won't hit the take profit goal.

What about memory and stationarity?

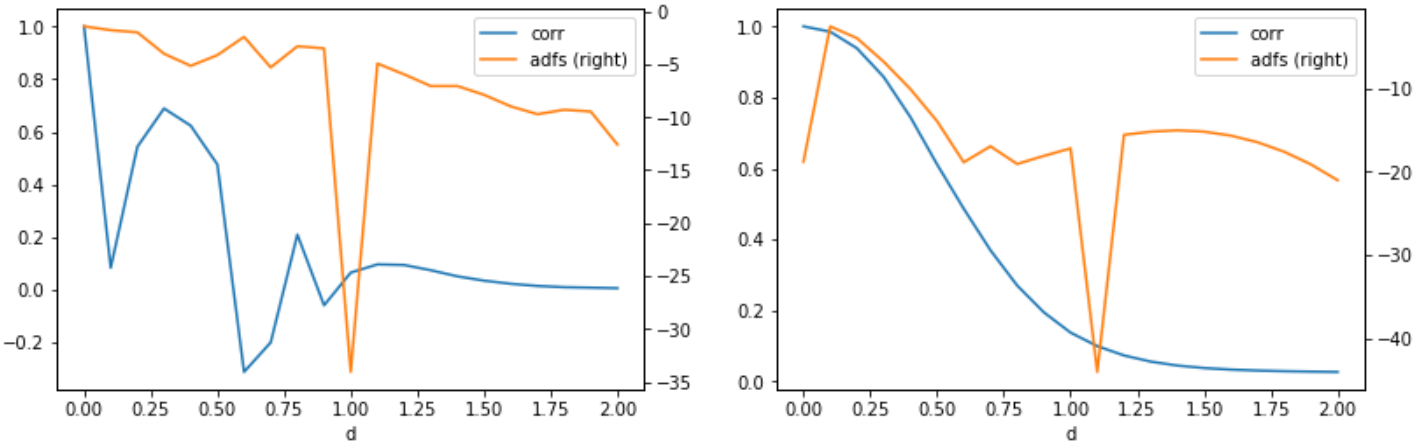
As we know, before passing data into any machine learning model we need to normalize or standardize or make it stationary in some other way. When we talking about financial time series we mostly perform differentiation with some lag (usually lag 1 and we call it **returns**). It indeed makes time series stationary, but what happens with the information inside this time series that supposed to have some memory? It's

just wiped out! Why? Because of this differentiation of lag 1 looks just one bar behind and doesn't know anything about what happened before and we do it consequently for all bars in our historical time window for the sake of stationarity. We still want to have a stationary time series, but without deleting all the useful memory in it… what if we only could differentiate it with an order less than 1…? And indeed we can and this is called **fractional differentiation**. I will leave more details to the main book or to other resources, let's just believe for now that we can differentiate time series with lag 0.1, 0.3, or 0.75, which supposedly will give us a bit more memory.

$$
\begin{aligned}
(1-B)^d &= \sum_{k=0}^{\infty} \binom{d}{k} (-B)^k \\
&= \sum_{k=0}^{\infty} \frac{\prod_{a=0}^{k-1}(d-a)\,(-B)^k}{k!} \\
&= 1 - dB + \frac{d(d-1)}{2!}B^2 - \cdots .
\end{aligned}
$$

Backshift operator of arbitrary order (from Wikipedia)

Let's measure how much information we keep as the correlation between log prices and different kinds of differentiation and stationarity via ADF test:



ADFs and correlation for tick bars (left) and time bars (right)

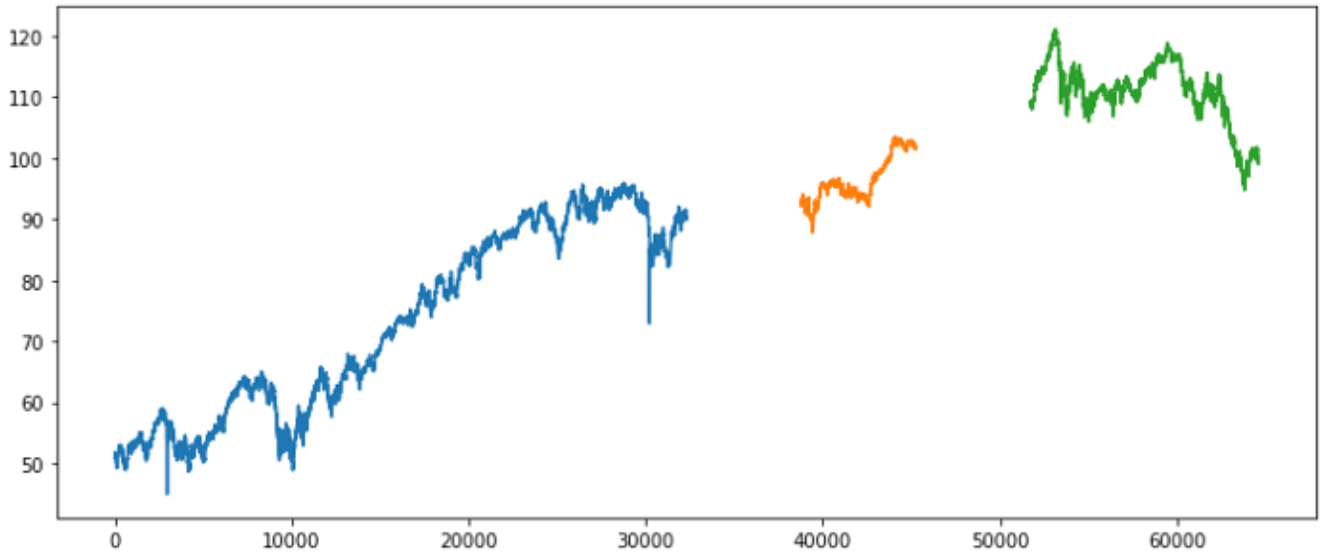
As we can see, in these tick bars time series, fractional differentiation won't give us better stationarity, but indeed has much more memory as correlation. In the case of **time bars**, we could improve our data a bit **with the use of differentiation of order 0.5–0.75**.

Feeding it all to the neural nets

Alright, now it's time to see what will happen when we will try to model our new bars with new objectives. I'll use very simple softmax regression as a classification algorithm (with dropout for regularization) and standardized close prices, volumes and returns over the historical window as features. The model in Keras looks like:

```
main_input = Input(shape=shape, name='main_input')
x = Flatten()(main_input)
x = Dropout(0.25)(x)
output = Dense(3, activation = "softmax")(x)
```

Long window length is 100, short — 50, forecasting horizon — 25 bars accordingly. Threshold T = 0.01. The dataset will consist of **tick bars**:



Tick bars for a train (blue), validation (yellow) and test set (green)

As you can see, there is some free space between sets, this is called *"embargo"* and helps to assure, that our model will work even a bit more into the future. We will study it in detail later, now let's just consider it as a more realistic split scenario. To deal with class imbalance, the weights for each class were calculated based on train and validation set.

Fixed threshold vs volatility threshold vs triple barrier

Our **baseline** will be just prediction of the price going up or down or not exceeding the threshold T. The classification report will be following (class 0 stands for down movement, 1 — for not significant movement and 2 — for up movement):

```
              precision    recall  f1-score   support
```

```
              0       0.09      0.32      0.14       899
              1       0.90      0.36      0.52     11116
              2       0.06      0.41      0.11       765

    micro avg       0.36      0.36      0.36     12780
    macro avg       0.35      0.36      0.25     12780
weighted avg        0.79      0.36      0.47     12780
```

Not very impressive, right? Not the imbalance in precision/recall and dataset label imbalance as well. What about the **volatility threshold**?

```
                  precision    recall  f1-score   support

              0       0.39      0.33      0.36      4922
              1       0.20      0.30      0.24      2896
              2       0.38      0.34      0.36      4962

    micro avg       0.33      0.33      0.33     12780
    macro avg       0.33      0.32      0.32     12780
weighted avg        0.35      0.33      0.33     12780
```

The numbers are getting better. We have no imbalance in precision/recall (which means that the classifier is fair and with an improvement of the algorithm itself and the features we will have trustworthy improvement in accuracy) and dataset imbalance is not that critical now. Okay, let's try **triple barrier now with symmetric take profit and stop loss at the value of rolling T based on volatility** like in the previous example:

```
                  precision    recall  f1-score   support

              0       0.50      0.28      0.36      6225
              1       0.48      0.47      0.47      6189
              2       0.02      0.22      0.04       366

    micro avg       0.37      0.37      0.37     12780
    macro avg       0.33      0.32      0.29     12780
weighted avg        0.48      0.37      0.41     12780
```

The average results more or less the same, because class 2, that here stands for the barrier touching the vertical line is very underpopulated. If we avoid it, the accuracies of up and down movements based on the take profit and stop loss will be higher than in previous examples, but with the slight imbalance appeared in the precision/recall.

Having these predictions as **meta-labels** for a binary prediction model that will learn **the side of the bet** it will have the following accuracies:

```
                  precision    recall  f1-score   support

           0       0.03      0.58      0.06       360
           1       0.98      0.52      0.68     12420

   micro avg       0.52      0.52      0.52     12780
   macro avg       0.51      0.55      0.37     12780
weighted avg       0.95      0.52      0.66     12780
```

Fractional differentiation vs integer differentiation

Let's now replace returns in the set of features with fractionally differentiated log prices with **d = 0.5** and see what will happen with the same experiments. **Baseline** (which is still not very impressive):

```
                  precision    recall  f1-score   support

           0       0.09      0.47      0.15       899
           1       0.88      0.45      0.60     11116
           2       0.05      0.16      0.08       765

   micro avg       0.44      0.44      0.44     12780
   macro avg       0.34      0.36      0.28     12780
weighted avg       0.78      0.44      0.54     12780
```

Volatility horizon is already a bit better than with "normal" differentiation:

```
                  precision    recall  f1-score   support

           0       0.41      0.37      0.39      4922
           1       0.24      0.28      0.26      2896
           2       0.40      0.40      0.40      4962

   micro avg       0.36      0.36      0.36     12780
   macro avg       0.35      0.35      0.35     12780
weighted avg       0.37      0.36      0.36     12780
```

And, finally, **triple barrier and its meta-labeling**:

```
                   precision    recall  f1-score   support

           0       0.49      0.37      0.42      6225
           1       0.49      0.44      0.47      6189
           2       0.04      0.25      0.06       366

   micro avg       0.40      0.40      0.40     12780
   macro avg       0.34      0.35      0.32     12780
weighted avg       0.48      0.40      0.43     12780

                   precision    recall  f1-score   support

           0       0.03      0.53      0.06       360
           1       0.98      0.54      0.69     12420

   micro avg       0.54      0.54      0.54     12780
   macro avg       0.50      0.54      0.38     12780
weighted avg       0.95      0.54      0.68     12780
```

I think we can indeed confirm, that fractional differentiation is an extremely useful transformation that allows saving more information from the original time series!

Conclusions

In this article, we have reviewed and redefined how the classical bars that we used to see on financial websites are created. We discovered their statistical properties and we can really agree that volume or dollar bars are much more appealing than standard time-based bars. We also have built several more realistic ways to label outputs based on the continuously changing volatility and predefined take profit and stop loss goals. Last but not least we reviewed a way to make financial time series stationary without losing all the memory in it. We also have evaluated all these experiments with data and labeling and we can actually see, that all these meaningful improvements really make results more stable and adequate. Of course, it's not a Holy Graal, but at least it is not that stupid as I and many others did before… :) Don't forget to check out the code in my repository!

P.S.

Follow me also on Facebook for AI articles that are too short for Medium, Instagram for personal stuff and Linkedin!