📈PYTHON FOR FINANCE SERIES

Data Labelling

The Triple-barrier Method

Ke Gui Aug 30 · 12 min read ★

Photo by Dave Gandy under the Public Domain Dedication License

Warning: *There is no magical formula or Holy Grail here, though a new world might open the door for you.*

Note 1: How to install mlfinlab package without error messages can be found here.

Note 2: If you are reading Advances in Financial Machine Learning by Marcos Prado. ***7. Fractionally Differentiated Features*** is Chapter 5 about Fractionally Differentiated Features. ***8. Data Labelling*** is Chapter 3 about The Triple-barrier Method. *And* ***9. Meta-labeling*** *is Chapter 3.6 on page 50.* I have planned to go through each chapter step by step as I haven't found a very detailed explanation of those concepts in each chapter yet. Please stay tuned!

📈Python For Finance Series

1. Identifying Outliers
2. Identifying Outliers — Part Two
3. Identifying Outliers — Part Three
4. Stylized Facts
5. Feature Engineering & Feature Selection
6. Data Transformation
7. Fractionally Differentiated Features
8. Data Labelling
9. Meta-labeling and Stacking

out of three barriers introduced in Chapter 3 of Advances in Financial Machine Learning by Marcos Prado[1]. The conventional way to label the data is by using the next day (lagged) return with the fixed-time horizon method. This method can be described as follows.

$$y_i = \begin{cases} -1 & \text{if } r_{t_{i,0},t_{i,0}+h} < -\tau \\ 0 & \text{if } |r_{t_{i,0},t_{i,0}+h}| \leq \tau \\ 1 & \text{if } r_{t_{i,0},t_{i,0}+h} > \tau \end{cases}$$

and

$$r_{t_{i,0},t_{i,0}+h} = \frac{p_{t_{i,0}+h}}{p_{t_{i,0}}} - 1$$

There are several drawbacks about this popular conventional labelling method. First, time bars do not exhibit good statistical properties. Second, the same threshold τ is applied regardless of the observed volatility. Basically, labelling doesn't reflect the current state of the investment.

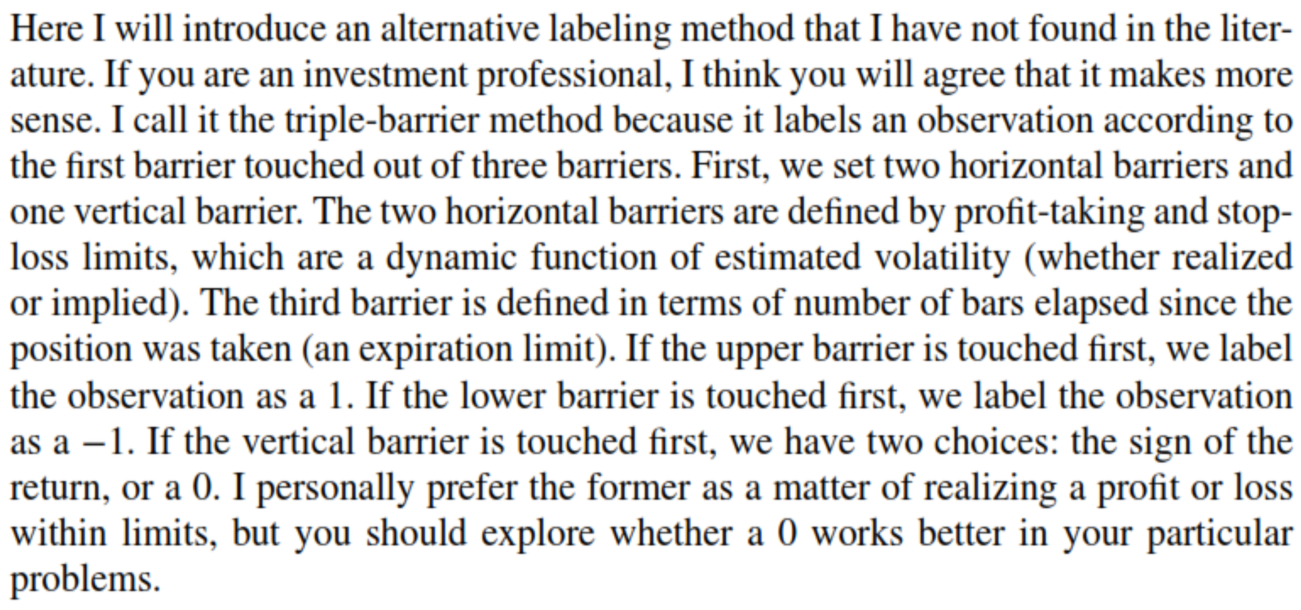
Moreover, in a real case, the chance is that you may not want to sell the next day. Therefore, triple-barrier method makes more sense in practice as it is path-dependent. You can make sound decisions depending on how many days you are planning to hold the stock and what's happening to the stock during that period.

The original code from Chapter 3 of Advances in Financial Machine Learning is created for high-frequency trading, using high-frequency data, and most are intraday data. If you are using daily data, we need to tweak the code a little bit. I also refracted most of the code from the book to make it beginner-friendly by heavily utilizing `padas DataFrame` structure to store all the information in one place. By this way, it makes life so much easier later on when you start to analysis or plot the data. At the meantime, I employed more complicated approaches such as Average True Range as the daily volatility. You can see all the code at the end of this article.

Intuition

window for you to make a buy or sell decision. If you haven't read it, you can always go back to here, here and here.

According to Advances in Financial Machine Learning by Marcos Prado[1], Triple Barrier method is:



Here I will introduce an alternative labeling method that I have not found in the literature. If you are an investment professional, I think you will agree that it makes more sense. I call it the triple-barrier method because it labels an observation according to the first barrier touched out of three barriers. First, we set two horizontal barriers and one vertical barrier. The two horizontal barriers are defined by profit-taking and stop-loss limits, which are a dynamic function of estimated volatility (whether realized or implied). The third barrier is defined in terms of number of bars elapsed since the position was taken (an expiration limit). If the upper barrier is touched first, we label the observation as a 1. If the lower barrier is touched first, we label the observation as a -1. If the vertical barrier is touched first, we have two choices: the sign of the return, or a 0. I personally prefer the former as a matter of realizing a profit or loss within limits, but you should explore whether a 0 works better in your particular problems.

Basically, what we are doing here is:

We will buy in a stock (let's say Apple) and hold it for 10 days. If the price is going down and trigger the stop loss alarm we exit at the stop-loss limit, or if the price is going up, we take the profit at a certain point. In an extreme case, the stock price goes sideway, we exit at a certain day after holding it for a while.

Assume we have a simple equity management rule:

- Never risk more than 2% of your total capital in a trade.
- Always look to trade only those opportunities where you will have a 3:1 earnings ratio.

Based on those simple rules, we make a trading plan before we put real money into any stocks. To infuse that trading plan into stock price movement, we need 3 barriers. What are those 3 barriers? 4 lines form a frame, defines a window as showing below.

The x-axis is the datetime, y-axis is the stock price. Line a,d belong to x-axis, which is the datatime index, and line b,c belong to y-axis which is the stock price.

a: starting date

b: stop-loss exit price

c: the profit-taking exit price

d: starting date + the number of days you are planning to hold it.

b and c don't have to be same. Remember we want to set profit-taking and stop-loss limits that are a function of the risks involved in a bet. And we are always looking to trade only those opportunities where you will have a 3:1 earnings ratio. Here to set $c = 3 * b$ will do the trick.

There are few videos on this topic, I just found one on YouTube.

OK, without further ado, let's dive in the code.

1. Data preparation

For consistency, in all the 📈Python for finance series, I will try to reuse the same data as much as I can. More details about data preparation can be found here, here and here or you can refer back to my previous article. Or if you like, you can ignore all the code below and use whatever clean data you have at hand, it won't affect the things we are going to do together.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn')

plt.rcParams['figure.figsize'] = [16, 9]
plt.rcParams['figure.dpi'] = 300
plt.rcParams['font.size'] = 20
plt.rcParams['axes.labelsize'] = 20
plt.rcParams['axes.titlesize'] = 24
plt.rcParams['xtick.labelsize'] = 16
```

```
import yfinance as yf

def get_data(symbols, begin_date=None,end_date=None):
    df = yf.download('AAPL', start = begin_date,
                     auto_adjust=True,#only download adjusted data
                     end= end_date)
    #my convention: always lowercase
    df.columns = ['open','high','low',
                  'close','volume']

    return df

Apple_stock = get_data('AAPL', '2000-01-01', '2010-12-31')
price = Apple_stock['close']
```

2. Daily Volatility

The original code (below) of getting daily volatility is for intraday data, which is consecutive data with no weekend, nonbusiness days, etc..

```
def getDailyVol(close,span0=100):
    # daily vol, reindexed to close
    df0=close.index.searchsorted(close.index-pd.Timedelta(days=1))
    df0=df0[df0>0]
    df0=pd.Series(close.index[df0 – 1],
                  index=close.index[close.shape[0]-df0.shape[0]:])
    df0=close.loc[df0.index]/close.loc[df0.values].values-1
    # daily returns
    df0=df0.ewm(span=span0).std()
    return df0
```

If you run this function, you will get an error message: `SyntaxError: invalid character in identifier`, that is because `close.index[df0–1]`. It can be fixed like this:

```
def getDailyVol(close,span0=100):
    # daily vol, reindexed to close
    df0=close.index.searchsorted(close.index-pd.Timedelta(days=1))
    df0=df0[df0>0]
    a = df0 -1 #using a variable to avoid the error message.
    df0=pd.Series(close.index[a],
                  index=close.index[close.shape[0]-df0.shape[0]:])
    df0=close.loc[df0.index]/close.loc[df0.values].values-1
    # daily returns
    df0=df0.ewm(span=span0).std()
```

If you use daily data instead of intraday data, you will end up with lots of duplicates as the date moved backwards 1 day and causing many NaN later on as many dates will be non-business days.

```
df0=close.index.searchsorted(close.index-pd.Timedelta(days=1))
pd.Series(df0).value_counts()
```

2766–2189 = 577 duplicates.

With daily data, we can use a simple percentage returns' Exponential weighted moving average (EWM) as the volatility.

```
def get_Daily_Volatility(close,span0=20):
    # simple percentage returns
    df0=close.pct_change()
    # 20 days, a month EWM's std as boundary
    df0=df0.ewm(span=span0).std()
    df0.dropna(inplace=True)
    return df0

df0 = get_Daily_Volatility(price)
df0
```

Depending upon the type of problem, we can choose more complicated approaches such as Average True Range (a technical analysis indicator that measures market volatility).

The formula for ATR is:

The first step in calculating ATR is to find a series of true range values for a stock price. The price range of an asset for a given trading day is simply its high minus its low, while the true range is current high less the current low; the absolute value of the current high less the previous close; and the absolute value of the current low less the previous close. The average true range is then a moving average, generally using 14 days, of the true ranges.

```
def get_atr(stock, win=14):

    atr_df = pd.Series(index=stock.index)
    high = pd.Series(Apple_stock.high.rolling( \
                     win, min_periods=win))
    low = pd.Series(Apple_stock.low.rolling( \
                    win, min_periods=win))
    close = pd.Series(Apple_stock.close.rolling( \
                      win, min_periods=win))
                    np.abs(low[i] - close[i])], \
                    axis=0)

        atr_df[i] = tr.sum() / win

    return  atr_df

get_atr(Apple_stock, 14)
atr_df
```

3. Triple-Barrier

Before we start to work on the barriers, a few parameters need to be decided.

```
#set the boundary of barriers, based on 20 days EWM
daily_volatility = get_Daily_Volatility(price)
# how many days we hold the stock which set the vertical barrier
t_final = 10
#the up and low boundary multipliers
upper_lower_multipliers = [2, 2]

#allign the index
prices = price[daily_volatility.index]
```

Here, I will use `pd.DataFrame` as the container to add all the information into one place.

```
def get_3_barriers():
    #create a container
    barriers = pd.DataFrame(columns=['days_passed',
              'price', 'vert_barrier', \
              'top_barrier', 'bottom_barrier'], \
               index = daily_volatility.index)
```

```
                    [daily_volatility.index[0] : day])

        #set the vertical barrier
        if (days_passed + t_final < len(daily_volatility.index) \
            and t_final != 0):
            vert_barrier = daily_volatility.index[
                                days_passed + t_final]
        else:
            vert_barrier = np.nan

        #set the top barrier
        if upper_lower_multipliers[0] > 0:
            top_barrier = prices.loc[day] + prices.loc[day] * \
                          upper_lower_multipliers[0] * vol
        else:
            #set it to NaNs
            top_barrier = pd.Series(index=prices.index)

        #set the bottom barrier
        if upper_lower_multipliers[1] > 0:
            bottom_barrier = prices.loc[day] - prices.loc[day] * \
                          upper_lower_multipliers[1] * vol
        else:
            #set it to NaNs
            bottom_barrier = pd.Series(index=prices.index)

        barriers.loc[day, ['days_passed', 'price',
        'vert_barrier','top_barrier', 'bottom_barrier']] = \
         days_passed, prices.loc[day], vert_barrier,
         top_barrier, bottom_barrier

    return barriers
```

Let’s have a look at all the barriers.

```
barriers = get_barriers()
barriers
```

Open in app

and have a close look at all the data information.

```
barriers.info()
```

Only the `vert_barrier` has 11 `NaN` value at the end as the `t_final` was set as 10 days.

The next step is to label each entry according to which barrier was touched first. I add a new column `'out'` to the end of `barriers`.

```
barriers['out'] = None
barriers.head()
```

Now, we can work on the labels.

```
def get_labels():
'''
start: first day of the window
end:last day of the window
price_initial: first day stock price
price_final:last day stock price
top_barrier: profit taking limit
bottom_barrier:stop loss limt
condition_pt:top_barrier touching conditon
condition_sl:bottom_barrier touching conditon

'''

    for i in range(len(barriers.index)):

        start = barriers.index[i]
        end = barriers.vert_barrier[i]

        if pd.notna(end):
            # assign the initial and final price
            price_initial = barriers.price[start]
            price_final = barriers.price[end]

            # assign the top and bottom barriers
            top_barrier = barriers.top_barrier[i]
            bottom_barrier = barriers.bottom_barrier[i]

            #set the profit taking and stop loss conditons
            condition_pt = (barriers.price[start: end] >= \
             top_barrier).any()
            condition_sl = (barriers.price[start: end] <= \
             bottom_barrier).any()

            #assign the labels
            if condition_pt:
                barriers['out'][i] = 1
            elif condition_sl:
                barriers['out'][i] = -1
            else:
```

```
                                           (price_final - price_initial)/ \
                                           (price_initial - bottom_barrier)],\
                                            key=abs)
     return

   get_labels()
   barriers
```

We can plot the ‘out’ to see its distribution.

```
   plt.plot(barriers.out,'bo')
```

and count how many profit taking and stop loss limit were triggered.

```
barriers.out.value_counts()
```

There are 1385 profit-taking and 837 stop-losing out of 2764 data points. 542 cases exit because time is up.

We can also pick a random date and show it on a graph.

```
fig,ax = plt.subplots()
ax.set(title='Apple stock price',
       xlabel='date', ylabel='price')
ax.plot(barriers.price[100: 200])

start = barriers.index[120]
end = barriers.vert_barrier[120]
upper_barrier = barriers.top_barrier[120]
lower_barrier = barriers.bottom_barrier[120]
```

```
                       (lower_barrier + upper_barrier)*0.5], 'r--');
ax.plot([start, start], [lower_barrier, upper_barrier], 'r-');
ax.plot([end, end], [lower_barrier, upper_barrier], 'r-');
```

We also can draw a dynamic graph with easy.

```
fig,ax = plt.subplots()
ax.set(title='Apple stock price',
       xlabel='date', ylabel='price')
ax.plot(barriers.price[100: 200])

start = barriers.index[120]
end = barriers.index[120+t_final]
upper_barrier = barriers.top_barrier[120]
lower_barrier = barriers.bottom_barrier[120]
ax.plot(barriers.index[120:120+t_final+1],
barriers.top_barrier[start:end], 'r--');
ax.plot(barriers.index[120:120+t_final+1],
barriers.bottom_barrier[start:end], 'r--');
ax.plot([start, end], [(lower_barrier + upper_barrier)*0.5, \
                       (lower_barrier + upper_barrier)*0.5], 'r--');
ax.plot([start, start], [lower_barrier, upper_barrier], 'r-');
ax.plot([end, end], [barriers.bottom_barrier[end],
barriers.top_barrier[end]], 'r-');
```

Open in app

Recap the parameters we have:

- Data: Apple 10-years stock price
- Hold for: no more than 10 days
- Profit-taking boundary: 2 times of 20 days return EWM std
- Stop-loss boundary: 2 times of 20 days return EWM std

The rule we expect in the real case:

- Always look to trade only those opportunities where you will have a 3:1 earn ratio.
- Never risk more than 2% of your total capital in a trade.

The first rule can be easily realized by setting `upper_lower_multipliers = [3, 1]` . The second one is about the trading size, the side times the size will enable us the calculate the risk (margin/edge). That will be meta-labelling in the next article. So, stay tuned!

Here is all the code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
plt.rcParams['figure.dpi'] = 300
plt.rcParams['font.size'] = 20
plt.rcParams['axes.labelsize'] = 20
plt.rcParams['axes.titlesize'] = 24
plt.rcParams['xtick.labelsize'] = 16
plt.rcParams['ytick.labelsize'] = 16
plt.rcParams['font.family'] = 'serif'

import yfinance as yf

def get_data(symbols, begin_date=None,end_date=None):
    df = yf.download('AAPL', start = begin_date,
                     auto_adjust=True,#only download adjusted data
                     end= end_date)
    #my convention: always lowercase
    df.columns = ['open','high','low',
                  'close','volume']

    return df

Apple_stock = get_data('AAPL', '2000-01-01', '2010-12-31')
price = Apple_stock['close']

def get_Daily_Volatility(close,span0=20):
    # simple percentage returns
    df0=close.pct_change()
    # 20 days, a month EWM's std as boundary
    df0=df0.ewm(span=span0).std()
    df0.dropna(inplace=True)
    return df0

df0 = get_Daily_Volatility(price)

def get_atr(stock, win=14):

    atr_df = pd.Series(index=stock.index)
    high = pd.Series(Apple_stock.high.rolling( \
                     win, min_periods=win))
    low = pd.Series(Apple_stock.low.rolling( \
                    win, min_periods=win))
    close = pd.Series(Apple_stock.close.rolling( \
                      win, min_periods=win))

    for i in range(len(stock.index)):
        tr=np.max([(high[i] - low[i]), \
                  np.abs(high[i] - close[i]), \
                  np.abs(low[i] - close[i])], \
                  axis=0)

    atr_df[i] = tr.sum() / win

    return  atr_df
```

Open in app

```
# how many days we hold the stock which set the vertical barrier
t_final = 10
#the up and low boundary multipliers
upper_lower_multipliers = [2, 2]

#allign the index
prices = price[daily_volatility.index]

def get_3_barriers():
    #create a container
    barriers = pd.DataFrame(columns=['days_passed',
              'price', 'vert_barrier', \
              'top_barrier', 'bottom_barrier'], \
               index = daily_volatility.index)

   for day, vol in daily_volatility.iteritems():
        days_passed = len(daily_volatility.loc \
                      [daily_volatility.index[0] : day])

        #set the vertical barrier
        if (days_passed + t_final < len(daily_volatility.index) \
            and t_final != 0):
            vert_barrier = daily_volatility.index[
                                 days_passed + t_final]
        else:
            vert_barrier = np.nan

        #set the top barrier
        if upper_lower_multipliers[0] > 0:
            top_barrier = prices.loc[day] + prices.loc[day] * \
                          upper_lower_multipliers[0] * vol
        else:
            #set it to NaNs
            top_barrier = pd.Series(index=prices.index)

        #set the bottom barrier
        if upper_lower_multipliers[1] > 0:
            bottom_barrier = prices.loc[day] - prices.loc[day] * \
                          upper_lower_multipliers[1] * vol
        else:
            #set it to NaNs
            bottom_barrier = pd.Series(index=prices.index)

        barriers.loc[day, ['days_passed', 'price', \
        'vert_barrier','top_barrier', 'bottom_barrier']] = \
         days_passed, prices.loc[day], vert_barrier, \
         top_barrier, bottom_barrier

return barriers

def get_labels():
'''
```

Open in app

```
price_final:last day stock price
top_barrier: profit taking limit
bottom_barrier:stop loss limt
condition_pt:top_barrier touching conditon
condition_sl:bottom_barrier touching conditon

'''

for i in range(len(barriers.index)):

start = barriers.index[i]
        end = barriers.vert_barrier[i]

if pd.notna(end):
            # assign the initial and final price
            price_initial = barriers.price[start]
            price_final = barriers.price[end]

# assign the top and bottom barriers
            top_barrier = barriers.top_barrier[i]
            bottom_barrier = barriers.bottom_barrier[i]

#set the profit taking and stop loss conditons
            condition_pt = (barriers.price[start: end] >= \
             top_barrier).any()
            condition_sl = (barriers.price[start: end] <= \
             bottom_barrier).any()
```

References

1. Introduction to "Advances in Financial Machine Learning" by Lopez de Prado

```
                          key=abs)
 return

get_labels()
barriers
```

Machine Learning Data Science Python Finance Artificial Intelligence

```
ax.set(title='Apple stock price',
       xlabel='date', ylabel='price')
ax.plot(barriers.price[100: 200])

start = barriers.index[120]
end = barriers.vert_barrier[120]
```

```
                              barrier, lower_barrier], 'r--');
              rt                 _barrier + upper_barrier)*0.5, \
                                 (lower_barrier + upper_barrier)*0.5], 'r--');
ax.plot([start, start], [lower_barrier, upper_barrier], 'r-');
ax.plot([end, end], [lower_barrier, upper_barrier], 'r-');

#dynamic graph
fig,ax = plt.subplots()
ax.set(title='Apple stock price',
       xlabel='date', ylabel='price')
ax.plot(barriers.price[100: 200])

start = barriers.index[120]
end = barriers.index[120+t_final]
upper_barrier = barriers.top_barrier[120]
lower_barrier = barriers.bottom_barrier[120]
ax.plot(barriers.index[120:120+t_final+1],
barriers.top_barrier[start:end], 'r--');
ax.plot(barriers.index[120:120+t_final+1],
barriers.bottom_barrier[start:end], 'r--');
ax.plot([start, end], [(lower_barrier + upper_barrier)*0.5, \
                       (lower_barrier + upper_barrier)*0.5], 'r--');
ax.plot([start, start], [lower_barrier, upper_barrier], 'r-');
ax.plot([end, end], [barriers.bottom_barrier[end],
barriers.top_barrier[end]], 'r-');
```