CHAPTER 3
Labeling

3.1 MOTIVATION

In Chapter 2 we discussed how to produce a matrix *X* of financial features out of an unstructured dataset. Unsupervised learning algorithms can learn the patterns from that matrix *X*, for example whether it contains hierarchical clusters. On the other hand, supervised learning algorithms require that the rows in *X* are associated with an array of labels or values *y*, so that those labels or values can be predicted on unseen features samples. In this chapter we will discuss ways to label financial data.

3.2 THE FIXED-TIME HORIZON METHOD

As it relates to finance, virtually all ML papers label observations using the fixed-time horizon method. This method can be described as follows. Consider a features matrix *X* with *I* rows, $\{X_i\}_{i=1,\dots,I}$, drawn from some bars with index $t = 1,\dots,T$, where $I \leq T$. Chapter 2, Section 2.5 discussed sampling methods that produce the set of features $\{X_i\}_{i=1,\dots,I}$. An observation X_i is assigned a label $y_i \in \{-1, 0, 1\}$,

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

where τ is a pre-defined constant threshold, $t_{i,0}$ is the index of the bar immediately after X_i takes place, $t_{i,0} + h$ is the index of the *h*-th bar after $t_{i,0}$, and $r_{t_{i,0},t_{i,0}+h}$ is the price return over a bar horizon *h*,

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

Because the literature almost always works with time bars, *h* implies a fixed-time horizon. The bibliography section lists multiple ML studies, of which

Dixon et al. [2016] is a recent example of this labeling method. Despite its popularity, there are several reasons to avoid this approach in most cases. First, as we saw in Chapter 2, time bars do not exhibit good statistical properties. Second, the same threshold τ is applied regardless of the observed volatility. Suppose that $\tau = 1E - 2$, where sometimes we label an observation as $y_i = 1$ subject to a realized bar volatility of $\sigma_{t_{i,0}} = 1E - 4$ (e.g., during the night session), and sometimes $\sigma_{t_{i,0}} = 1E - 2$ (e.g., around the open). The large majority of labels will be 0, even if return $r_{t_{i,0},t_{i,0}+h}$ was predictable and statistically significant.

In other words, it is a very common error to label observations according to a fixed threshold on time bars. Here are a couple of better alternatives. First, label per a varying threshold $\sigma_{t_{i,0}}$, estimated using a rolling exponentially weighted standard deviation of returns. Second, use volume or dollar bars, as their volatilities are much closer to constant (homoscedasticity). But even these two improvements miss a key flaw of the fixed-time horizon method: the path followed by prices. Every investment strategy has stop-loss limits, whether they are self-imposed by the portfolio manager, enforced by the risk department, or triggered by a margin call. It is simply unrealistic to build a strategy that profits from positions that would have been stopped-out by the exchange. That virtually no publication accounts for that when labeling observations tells you something about the current state of the investment literature.

3.3 COMPUTING DYNAMIC THRESHOLDS

As argued in the previous section, in practice we want to set profit taking and stop-loss limits that are a function of the risks involved in a bet. Otherwise, sometimes we will be aiming too high ($\tau \gg \sigma_{t_{i,0}}$), and sometimes too low ($\tau \ll \sigma_{t_{i,0}}$), considering the prevailing volatility.

Snippet 3.1 computes the daily volatility at intraday estimation points, applying a span of `span0` days to an exponentially weighted moving standard deviation. See the pandas documentation for details on the `pandas.Series.ewm` function.

SNIPPET 3.1 DAILY VOLATILITY ESTIMATES

```
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
```

We can use the output of this function to set default profit taking and stop-loss limits throughout the rest of this chapter.

3.4 THE TRIPLE-BARRIER METHOD

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.

You may have noticed that the triple-barrier method is path-dependent. In order to label an observation, we must take into account the entire path spanning $[t_{i,0}, t_{i,0} + h]$, where h defines the vertical barrier (the expiration limit). We will denote $t_{i,1}$ the time of the first barrier touch, and the return associated with the observed feature is $r_{t_{i,0},t_{i,1}}$. For the sake of clarity, $t_{i,1} \leq t_{i,0} + h$ and the horizontal barriers are not necessarily symmetric.

Snippet 3.2 implements the triple-barrier method. The function receives four arguments:

- `close`: A pandas series of prices.
- `events`: A pandas dataframe, with columns,
    - `t1`: The timestamp of vertical barrier. When the value is `np.nan`, there

will not be a vertical barrier.
    - `trgt`: The unit width of the horizontal barriers.
- `ptSl`: A list of two non-negative float values:
    - `ptSl[0]`: The factor that multiplies `trgt` to set the width of the upper barrier. If 0, there will not be an upper barrier.
    - `ptSl[1]`: The factor that multiplies `trgt` to set the width of the lower barrier. If 0, there will not be a lower barrier.
- `molecule`: A list with the subset of event indices that will be processed by a single thread. Its use will become clear later on in the chapter.

SNIPPET 3.2 TRIPLE-BARRIER LABELING METHOD

```
def applyPtSlOnT1(close,events,ptSl,molecule):
    # apply stop loss/profit taking, if it takes place before t1 (end of event)
    events_=events.loc[molecule]
    out=events_[['t1']].copy(deep=True)
    if ptSl[0]>0:pt=ptSl[0]*events_['trgt']
    else:pt=pd.Series(index=events.index) # NaNs
    if ptSl[1]>0:sl=-ptSl[1]*events_['trgt']
    else:sl=pd.Series(index=events.index) # NaNs
    for loc,t1 in events_['t1'].fillna(close.index[-1]).iteritems():
        df0=close[loc:t1] # path prices
        df0=(df0/close[loc]-1)*events_.at[loc,'side'] # path returns
        out.loc[loc,'sl']=df0[df0<sl[loc]].index.min() # earliest stop loss.
        out.loc[loc,'pt']=df0[df0>pt[loc]].index.min() # earliest profit taking.
    return out
```

The output from this function is a pandas dataframe containing the timestamps (if any) at which each barrier was touched. As you can see from the previous description, the method considers the possibility that each of the three barriers may be disabled. Let us denote a barrier configuration by the triplet [`pt,sl,t1`], where a 0 means that the barrier is inactive and a 1 means that the barrier is active. The possible eight configurations are:

- Three useful configurations:
    - [1,1,1]: This is the standard setup, where we define three barrier exit conditions. We would like to realize a profit, but we have a maximum tolerance for losses and a holding period.
    - [0,1,1]: In this setup, we would like to exit after a number of bars, unless we are stopped-out.

    - [1,1,0]: Here we would like to take a profit as long as we are not stopped-out. This is somewhat unrealistic in that we are willing to hold the position for as long as it takes.
- Three less realistic configurations:
    - [0,0,1]: This is equivalent to the fixed-time horizon method. It may still be useful when applied to volume-, dollar-, or information-driven bars, and multiple forecasts are updated within the horizon.
    - [1,0,1]: A position is held until a profit is made or the maximum holding period is exceeded, without regard for the intermediate unrealized losses.
    - [1,0,0]: A position is held until a profit is made. It could mean being locked on a losing position for years.
- Two illogical configurations:
    - [0,1,0]: This is an aimless configuration, where we hold a position until we are stopped-out.
    - [0,0,0]: There are no barriers. The position is locked forever, and no label is generated.

Figure 3.1 shows two alternative configurations of the triple-barrier method. On the left, the configuration is [1,1,0], where the first barrier touched is the lower horizontal one. On the right, the configuration is [1,1,1], where the first barrier touched is the vertical one.

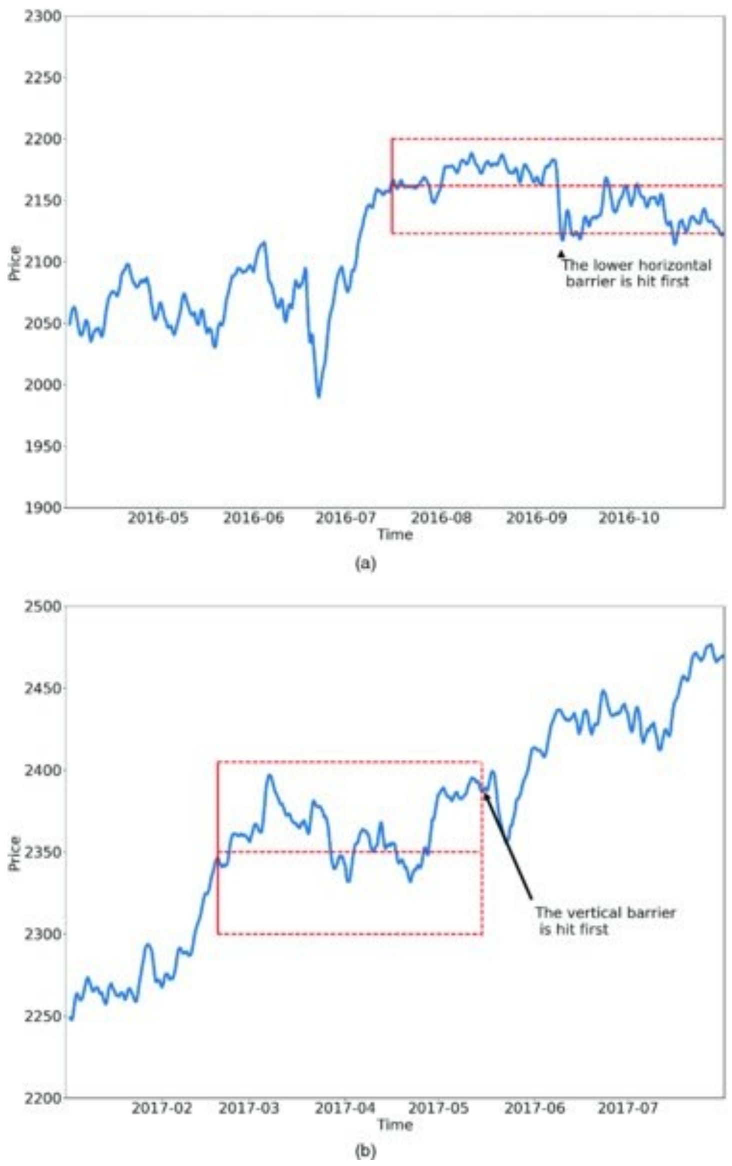

FIGURE 3.1 Two alternative configurations of the triple-barrier method

3.5 LEARNING SIDE AND SIZE

In this section we will discuss how to label examples so that an ML algorithm can learn both the side and the size of a bet. We are interested in learning the side of a bet when we do not have an underlying model to set the sign of our position (long or short). Under such circumstance, we cannot differentiate between a profit-taking barrier and a stop-loss barrier, since that requires knowledge of the side. Learning the side implies that either there are no horizontal barriers or that the horizontal barriers must be symmetric.

Snippet 3.3 implements the function `getEvents`, which finds the time of the first barrier touch. The function receives the following arguments:

- `close`: A pandas series of prices.
- `tEvents`: The pandas timeindex containing the timestamps that will seed every triple barrier. These are the timestamps selected by the sampling procedures discussed in Chapter 2, Section 2.5.
- `ptSl`: A non-negative float that sets the width of the two barriers. A 0 value means that the respective horizontal barrier (profit taking and/or stop loss) will be disabled.
- `t1`: A pandas series with the timestamps of the vertical barriers. We pass a `False` when we want to disable vertical barriers.
- `trgt`: A pandas series of targets, expressed in terms of absolute returns.
- `minRet`: The minimum target return required for running a triple barrier search.
- `numThreads`: The number of threads concurrently used by the function.

SNIPPET 3.3 GETTING THE TIME OF FIRST TOUCH

```python
def getEvents(close,tEvents,ptSl,trgt,minRet,numThreads,t1=False):
    #1) get target
    trgt=trgt.loc[tEvents]
    trgt=trgt[trgt>minRet] # minRet
    #2) get t1 (max holding period)
    if t1 is False:t1=pd.Series(pd.NaT,index=tEvents)
    #3) form events object, apply stop loss on t1
    side_=pd.Series(1.,index=trgt.index)
    events=pd.concat({'t1':t1,'trgt':trgt,'side':side_}, \
        axis=1).dropna(subset=['trgt'])
    df0=mpPandasObj(func=applyPtSlOnT1,pdObj=('molecule',events.index), \
        numThreads=numThreads,close=close,events=events,ptSl=[ptSl,ptSl])
    events['t1']=df0.dropna(how='all').min(axis=1) # pd.min ignores nan
    events=events.drop('side',axis=1)
    return events
```

Suppose that $I = 1E6$ and $h = 1E3$, then the number of conditions to evaluate is up to one billion on a single instrument. Many ML tasks are computationally expensive unless you are familiar with multi-threading, and this is one of them. Here is where parallel computing comes into play. Chapter 20 discusses a few multiprocessing functions that we will use throughout the book.

Function `mpPandasObj` calls a multiprocessing engine, which is explained in depth in Chapter 20. For the moment, you simply need to know that this function will execute `applyPtSlOnT1` in parallel. Function `applyPtSlOnT1` returns the timestamps at which each barrier is touched (if any). Then, the time of the first touch is the earliest time among the three returned by `applyPtSlOnT1`. Because we must learn the side of the bet, we have passed `ptSl = [ptSl,ptSl]` as argument, and we arbitrarily set the side to be always long (the horizontal barriers are symmetric, so the side is irrelevant to determining the time of the first touch). The output from this function is a pandas dataframe with columns:

- `t1`: The timestamp at which the first barrier is touched.
- `trgt`: The target that was used to generate the horizontal barriers.

Snippet 3.4 shows one way to define a vertical barrier. For each index in `tEvents`, it finds the timestamp of the next price bar at or immediately after a number of days `numDays`. This vertical barrier can be passed as optional argument `t1` in `getEvents`.

SNIPPET 3.4 ADDING A VERTICAL BARRIER

```
t1=close.index.searchsorted(tEvents+pd.Timedelta(days=numDays))
t1=t1[t1<close.shape[0]]
t1=pd.Series(close.index[t1],index=tEvents[:t1.shape[0]]) # NaNs at end
```

Finally, we can label the observations using the `getBins` function defined in Snippet 3.5. The arguments are the `events` dataframe we just discussed, and the `close` pandas series of prices. The output is a dataframe with columns:

- `ret`: The return realized at the time of the first touched barrier.
- `bin`: The label, $\{-1, 0, 1\}$, as a function of the sign of the outcome. The function can be easily adjusted to label as 0 those events when the vertical barrier was touched first, which we leave as an exercise.

SNIPPET 3.5 LABELING FOR SIDE AND SIZE

```
def getBins(events,close):
    #1) prices aligned with events
    events_=events.dropna(subset=['t1'])
    px=events_.index.union(events_['t1'].values).drop_duplicates()
    px=close.reindex(px,method='bfill')
#2) create out object
out=pd.DataFrame(index=events_.index)
out['ret']=px.loc[events_['t1'].values].values/px.loc[events_.index]-1
out['bin']=np.sign(out['ret'])
return out
```

3.6 META-LABELING

Suppose that you have a model for setting the side of the bet (long or short). You just need to learn the size of that bet, which includes the possibility of no bet at all (zero size). This is a situation that practitioners face regularly. We often know whether we want to buy or sell a product, and the only remaining question is how much money we should risk in such a bet. We do not want the ML algorithm to learn the side, just to tell us what is the appropriate size. At this point, it probably does not surprise you to hear that no book or paper has so far discussed this common problem. Thankfully, that misery ends here. I call this problem meta-labeling because we want to build a secondary ML model that learns how to use a primary exogenous model.

Rather than writing an entirely new `getEvents` function, we will make some adjustments to the previous code, in order to handle meta-labeling. First, we accept a new `side` optional argument (with default `None`), which contains the side of our bets as decided by the primary model. When `side` is not `None`, the function understands that meta-labeling is in play. Second, because now we know the side, we can effectively discriminate between profit taking and stop loss. The horizontal barriers do not need to be symmetric, as in Section 3.5. Argument `ptSl` is a list of two non-negative float values, where `ptSl[0]` is the factor that multiplies `trgt` to set the width of the upper barrier, and `ptSl[1]` is the factor that multiplies `trgt` to set the width of the lower barrier. When either is 0, the respective barrier is disabled. Snippet 3.6 implements these enhancements.

SNIPPET 3.6 EXPANDING GETEVENTS TO INCORPORATE META-LABELING

```
def getEvents(close,tEvents,ptSl,trgt,minRet,numThreads,t1=False,side=None):
    #1) get target
    trgt=trgt.loc[tEvents]
    trgt=trgt[trgt>minRet] # minRet
    #2) get t1 (max holding period)
    if t1 is False:t1=pd.Series(pd.NaT,index=tEvents)
    #3) form events object, apply stop loss on t1
    if side is None:side_,ptSl_=pd.Series(1.,index=trgt.index),[ptSl[0],ptSl[0]]
    else:side_,ptSl_=side.loc[trgt.index],ptSl[:2]
    events=pd.concat({'t1':t1,'trgt':trgt,'side':side_}, \
        axis=1).dropna(subset=['trgt'])
    df0=mpPandasObj(func=applyPtSlOnT1,pdObj=('molecule',events.index), \
        numThreads=numThreads,close=inst['Close'],events=events,ptSl=ptSl_)
    events['t1']=df0.dropna(how='all').min(axis=1) # pd.min ignores nan
    if side is None:events=events.drop('side',axis=1)
    return events
```

Likewise, we need to expand the `getBins` function, so that it handles meta-labeling. Snippet 3.7 implements the necessary changes.

SNIPPET 3.7 EXPANDING `GETBINS` TO INCORPORATE META-LABELING

```
def getBins(events,close):
    '''
    Compute event's outcome (including side information, if provided).
    events is a DataFrame where:
    —events.index is event's starttime
    —events['t1'] is event's endtime
    —events['trgt'] is event's target
    —events['side'] (optional) implies the algo's position side
    Case 1: ('side' not in events): bin in (-1,1) <—label by price action
    Case 2: ('side' in events): bin in (0,1) <—label by pnl (meta-labeling)
    '''
    #1) prices aligned with events
    events_=events.dropna(subset=['t1'])
    px=events_.index.union(events_['t1'].values).drop_duplicates()
    px=close.reindex(px,method='bfill')
    #2) create out object
    out=pd.DataFrame(index=events_.index)
    out['ret']=px.loc[events_['t1'].values].values/px.loc[events_.index]-1
    if 'side' in events_:out['ret']*=events_['side'] # meta-labeling
    out['bin']=np.sign(out['ret'])
    if 'side' in events_:out.loc[out['ret']<=0,'bin']=0 # meta-labeling
    return out
```

Now the possible values for labels in `out['bin']` are {0,1}, as opposed to the previous feasible values {−1,0,1}. The ML algorithm will be trained to decide whether to take the bet or pass, a purely binary prediction. When the predicted label is 1, we can use the probability of this secondary prediction to derive the size of the bet, where the side (sign) of the position has been set by the primary model.

3.7 HOW TO USE META-LABELING

Binary classification problems present a trade-off between type-I errors (false positives) and type-II errors (false negatives). In general, increasing the true positive rate of a binary classifier will tend to increase its false positive rate. The receiver operating characteristic (ROC) curve of a binary classifier measures the cost of increasing the true positive rate, in terms of accepting higher false positive rates.

Figure 3.2 illustrates the so-called “confusion matrix.” On a set of observations, there are items that exhibit a condition (positives, left rectangle), and items that do not exhibit a condition (negative, right rectangle). A binary classifier predicts that some items exhibit the condition (ellipse), where the TP area contains the true positives and the TN area contains the true negatives. This leads to two kinds of errors: false positives (FP) and false negatives (FN). “Precision” is the ratio between the TP area and the area in the ellipse. “Recall” is the ratio between the TP area and the area in the left rectangle. This notion of recall (aka true positive rate) is in the context of classification problems, the analogous to “power” in the context of hypothesis testing. “Accuracy” is the sum of the TP and TN areas divided by the overall set of items (square). In general, decreasing the FP area comes at a cost of increasing the FN area, because higher precision typically means fewer calls, hence lower recall. Still, there is some combination of precision and recall that maximizes the overall efficiency of the classifier. The F1-score measures the efficiency of a classifier as the harmonic average between precision and recall (more on this in Chapter 14).

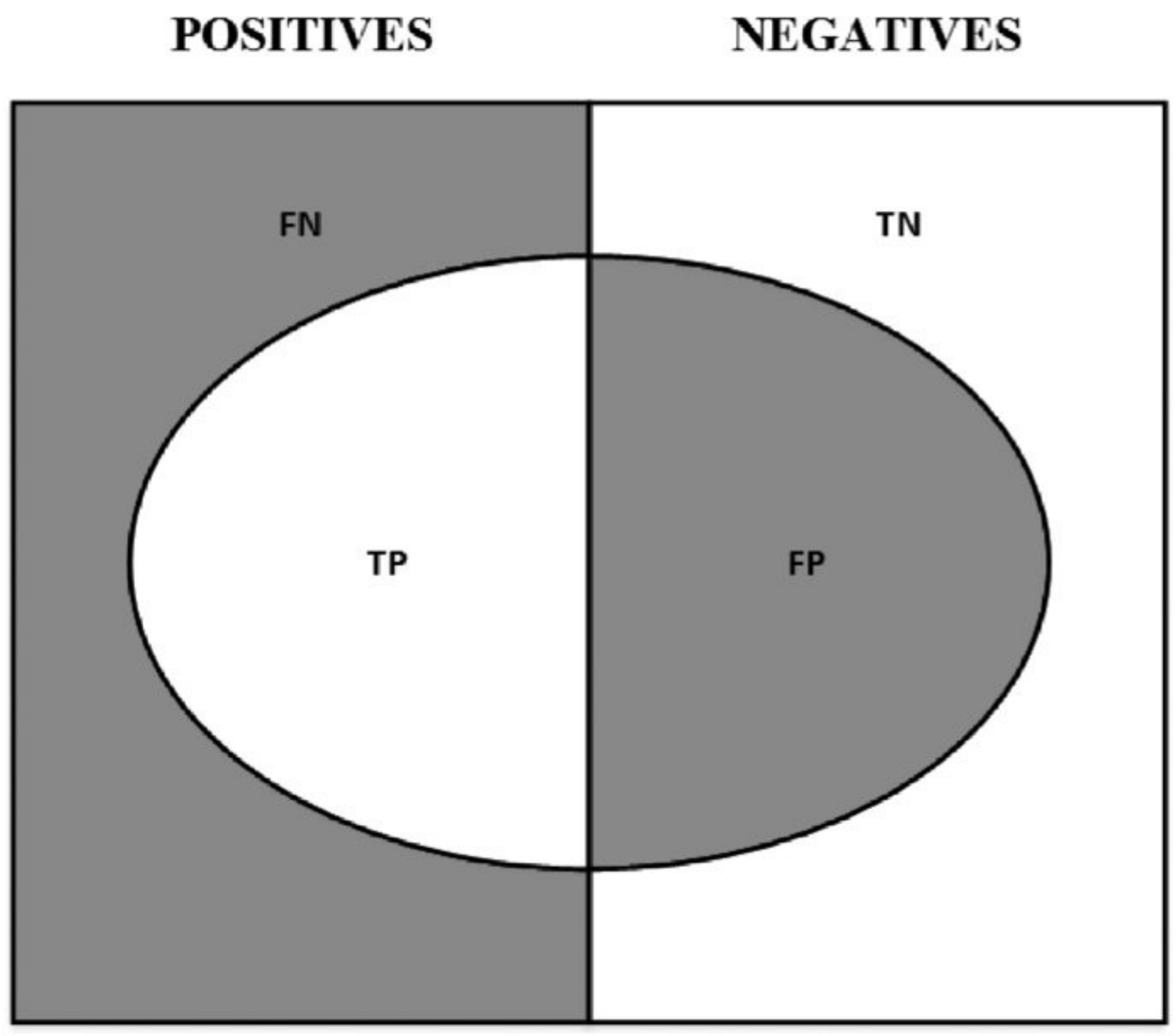

FIGURE 3.2 A visualization of the “confusion matrix”

Meta-labeling is particularly helpful when you want to achieve higher F1-scores. First, we build a model that achieves high recall, even if the precision is not particularly high. Second, we correct for the low precision by applying meta-labeling to the positives predicted by the primary model.

Meta-labeling will increase your F1-score by filtering out the false positives, where the majority of positives have already been identified by the primary model. Stated differently, the role of the secondary ML algorithm is to determine whether a positive from the primary (exogenous) model is true or false. It is *not* its purpose to come up with a betting opportunity. Its purpose is to determine whether we should act or pass on the opportunity that has been presented.

Meta-labeling is a very powerful tool to have in your arsenal, for four additional reasons. First, ML algorithms are often criticized as black boxes (see Chapter 1). Meta-labeling allows you to build an ML system on top of a white box (like a fundamental model founded on economic theory). This ability to transform a fundamental model into an ML model should make meta-labeling particularly useful to "quantamental" firms. Second, the effects of overfitting are limited when you apply meta-labeling, because ML will not decide the side of your bet, only the size. Third, by decoupling the side prediction from the size prediction, meta-labeling enables sophisticated strategy structures. For instance, consider that the features driving a rally may differ from the features driving a sell-off. In that case, you may want to develop an ML strategy exclusively for long positions, based on the buy recommendations of a primary model, and an ML strategy exclusively for short positions, based on the sell recommendations of an entirely different primary model. Fourth, achieving high accuracy on small bets and low accuracy on large bets will ruin you. As important as identifying good opportunities is to size them properly, so it makes sense to develop an ML algorithm solely focused on getting that critical decision (sizing) right. We will retake this fourth point in Chapter 10. In my experience, meta-labeling ML models can deliver more robust and reliable outcomes than standard labeling models.

3.8 THE QUANTAMENTAL WAY

You may have read in the press that many hedge funds are embracing the quantamental approach. A simple Google search will show reports that many hedge funds, including some of the most traditional ones, are investing tens of millions of dollars in technologies designed to combine human expertise with quantitative methods. It turns out, meta-labeling is exactly what these people have been waiting for. Let us see why.

Suppose that you have a series of features that you believe can forecast some prices, you just do not know how. Since you do not have a model to determine the side of each bet, you need to learn both side and size. You apply what you have learned in Section 3.5, and produce some labels based on the triple-barrier method with symmetric horizontal barriers. Now you are ready to fit your algorithm on a training set, and evaluate the accuracy of your forecasts on a testing set. Alternatively, you could do the following:

1. Use your forecasts from the primary model, and generate meta-labels. Remember, horizontal barriers do not need to be symmetric in this case.

2. Fit your model again on the same training set, but this time using the meta-labels you just generated.
3. Combine the “sides” from the first ML model with the “sizes” from the second ML model.

You can always add a meta-labeling layer to any primary model, whether that is an ML algorithm, an econometric equation, a technical trading rule, a fundamental analysis, etc. That includes forecasts generated by a human, solely based on his intuition. In that case, meta-labeling will help us figure out when we should pursue or dismiss a discretionary PM’s call. The features used by such meta-labeling ML algorithm could range from market information to biometric statistics to psychological assessments. For example, the meta-labeling ML algorithm could find that discretionary PMs tend to make particularly good calls when there is a structural break (Chapter 17), as they may be quicker to grasp a change in the market regime. Conversely, it may find that PMs under stress, as evidenced by fewer hours of sleep, fatigue, change in weight, etc. tend to make inaccurate predictions.[1] Many professions require regular psychological exams, and an ML meta-labeling algorithm may find that those scores are also relevant to assess our current degree of confidence on a PM’s predictions. Perhaps none of these factors affect discretionary PMs, and their brains operate independently from their emotional being, like cold calculating machines. My guess is that this is not the case, and therefore meta-labeling should become an essential ML technique for every discretionary hedge fund. In the near future, every discretionary hedge fund will become a quantamental firm, and meta-labeling offers them a clear path to make that transition.

3.9 DROPPING UNNECESSARY LABELS

Some ML classifiers do not perform well when classes are too imbalanced. In those circumstances, it is preferable to drop extremely rare labels and focus on the more common outcomes. Snippet 3.8 presents a procedure that recursively drops observations associated with extremely rare labels. Function `dropLabels` recursively eliminates those observations associated with classes that appear less than a fraction `minPct` of cases, unless there are only two classes left.

SNIPPET 3.8 DROPPING UNDER-POPULATED LABELS

```
def dropLabels(events,minPtc=.05):
    # apply weights, drop labels with insufficient examples
    while True:
        df0=events['bin'].value_counts(normalize=True)
        if df0.min()>minPct or df0.shape[0]<3:break
        print 'dropped label',df0.argmin(),df0.min()
        events=events[events['bin']!=df0.argmin()]
    return events
```

Incidentally, another reason you may want to drop unnecessary labels is this known sklearn bug: https://github.com/scikit-learn/scikit-learn/issues/8566. This sort of bug is a consequence of very fundamental assumptions in sklearn implementation, and resolving them is far from trivial. In this particular instance, the error stems from sklearn's decision to operate with standard numpy arrays rather than structured arrays or pandas objects. It is unlikely that there will be a fix by the time you are reading this chapter, or in the near future. In later chapters, we will study ways to circumvent these sorts of implementation errors, by building your own classes and expanding sklearn's functionality.

EXERCISES

3.1 Form dollar bars for E-mini S&P 500 futures:

a. Apply a symmetric CUSUM filter (Chapter 2, Section 2.5.2.1) where the threshold is the standard deviation of daily returns (Snippet 3.1).

b. Use Snippet 3.4 on a pandas series `t1`, where `numDays = 1`.

c. On those sampled features, apply the triple-barrier method, where `ptSl = [1,1]` and `t1` is the series you created in point 1.b.

d. Apply `getBins` to generate the labels.

3.2 From exercise 1, use Snippet 3.8 to drop rare labels.

3.3 Adjust the `getBins` function (Snippet 3.5) to return a 0 whenever the vertical barrier is the one touched first.

3.4 Develop a trend-following strategy based on a popular technical analysis statistic (e.g., crossing moving averages). For each observation, the model suggests a side, but not a size of the bet.

a. Derive meta-labels for `ptSl = [1,2]` and `t1` where `numDays = 1`. Use as `trgt` the daily standard deviation as computed by Snippet 3.1.

b. Train a random forest to decide whether to trade or not. Note: The decision is whether to trade or not, {0,1}, since the underlying model (the crossing moving average) has decided the side, {−1,1}.

3.5 Develop a mean-reverting strategy based on Bollinger bands. For each observation, the model suggests a side, but not a size of the bet.

a. Derive meta-labels for `ptSl = [0,2]` and `t1` where `numDays = 1`. Use as `trgt` the daily standard deviation as computed by Snippet 3.1.
b. Train a random forest to decide whether to trade or not. Use as features: volatility, serial correlation, and the crossing moving averages from exercise 2.
c. What is the accuracy of predictions from the primary model (i.e., if the secondary model does not filter the bets)? What are the precision, recall, and F1-scores?
d. What is the accuracy of predictions from the secondary model? What are the precision, recall, and F1-scores?

BIBLIOGRAPHY

Ahmed, N., A. Atiya, N. Gayar, and H. El-Shishiny (2010): “An empirical comparison of machine learning models for time series forecasting.” *Econometric Reviews*, Vol. 29, No. 5–6, pp. 594–621.

Ballings, M., D. van den Poel, N. Hespeels, and R. Gryp (2015): “Evaluating multiple classifiers for stock price direction prediction.” *Expert Systems with Applications*, Vol. 42, No. 20, pp. 7046–7056.

Bontempi, G., S. Taieb, and Y. Le Borgne (2012): “Machine learning strategies for time series forecasting.” *Lecture Notes in Business Information Processing*, Vol. 138, No. 1, pp. 62–77.

Booth, A., E. Gerding and F. McGroarty (2014): “Automated trading with performance weighted random forests and seasonality.” *Expert Systems with Applications*, Vol. 41, No. 8, pp. 3651–3661.

Cao, L. and F. Tay (2001): “Financial forecasting using support vector machines.” *Neural Computing & Applications*, Vol. 10, No. 2, pp. 184–192.

Cao, L., F. Tay and F. Hock (2003): “Support vector machine with adaptive parameters in financial time series forecasting.” *IEEE Transactions on Neural Networks*, Vol. 14, No. 6, pp. 1506–1518.

Cervelló-Royo, R., F. Guijarro, and K. Michniuk (2015): “Stock market trading rule based on pattern recognition and technical analysis: Forecasting the DJIA index with intraday data.” *Expert Systems with Applications*, Vol. 42, No. 14, pp. 5963–5975.

Chang, P., C. Fan and J. Lin (2011): “Trend discovery in financial time series data using a case-based fuzzy decision tree.” *Expert Systems with Applications*, Vol. 38, No. 5, pp. 6070–6080.

Kuan, C. and L. Tung (1995): “Forecasting exchange rates using feedforward and recurrent neural networks.” *Journal of Applied Econometrics*, Vol. 10, No. 4, pp. 347–364.

Creamer, G. and Y. Freund (2007): “A boosting approach for automated trading.” *Journal of Trading*, Vol. 2, No. 3, pp. 84–96.

Creamer, G. and Y. Freund (2010): “Automated trading with boosting and expert weighting.” *Quantitative Finance*, Vol. 10, No. 4, pp. 401–420.

Creamer, G., Y. Ren, Y. Sakamoto, and J. Nickerson (2016): “A textual analysis algorithm for the equity market: The European case.” *Journal of Investing*, Vol. 25, No. 3, pp. 105–116.

Dixon, M., D. Klabjan, and J. Bang (2016): “Classification-based financial markets prediction using deep neural networks.” *Algorithmic Finance*, forthcoming (2017). Available at SSRN: https://ssrn.com/abstract=2756331.

Dunis, C., and M. Williams (2002): “Modelling and trading the euro/US dollar exchange rate: Do neural network models perform better?” *Journal of Derivatives & Hedge Funds*, Vol. 8, No. 3, pp. 211–239.

Feuerriegel, S. and H. Prendinger (2016): “News-based trading strategies.” *Decision Support Systems*, Vol. 90, pp. 65–74.

Hsu, S., J. Hsieh, T. Chih, and K. Hsu (2009): “A two-stage architecture for stock price forecasting by integrating self-organizing map and support vector regression.” *Expert Systems with Applications*, Vol. 36, No. 4, pp. 7947–7951.

Huang, W., Y. Nakamori, and S. Wang (2005): “Forecasting stock market movement direction with support vector machine.” *Computers & Operations Research*, Vol. 32, No. 10, pp. 2513–2522.

Kara, Y., M. Boyacioglu, and O. Baykan (2011): “Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange.” *Expert Systems with Applications*, Vol. 38, No. 5, pp. 5311–5319.

Kim, K. (2003): “Financial time series forecasting using support vector machines.” *Neurocomputing*, Vol. 55, No. 1, pp. 307–319.

Krauss, C., X. Do, and N. Huck (2017): “Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500.” *European Journal of Operational Research*, Vol. 259, No. 2, pp. 689–702.

Laborda, R. and J. Laborda (2017): “Can tree-structured classifiers add value to the investor?” *Finance Research Letters*, Vol. 22 (August), pp. 211–226.

Nakamura, E. (2005): “Inflation forecasting using a neural network.” *Economics Letters*, Vol. 86, No. 3, pp. 373–378.

Olson, D. and C. Mossman (2003): “Neural network forecasts of Canadian stock returns using accounting ratios.” *International Journal of Forecasting*, Vol. 19, No. 3, pp. 453–465.

Patel, J., S. Sha, P. Thakkar, and K. Kotecha (2015): “Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques.” *Expert Systems with Applications*, Vol. 42, No. 1, pp. 259–268.

Patel, J., S. Sha, P. Thakkar, and K. Kotecha (2015): “Predicting stock market index using fusion of machine learning techniques.” *Expert Systems with Applications*, Vol. 42, No. 4, pp. 2162–2172.

Qin, Q., Q. Wang, J. Li, and S. Shuzhi (2013): “Linear and nonlinear trading models with gradient boosted random forests and application to Singapore Stock Market.” *Journal of Intelligent Learning Systems and Applications*, Vol. 5, No. 1, pp. 1–10.

Sorensen, E., K. Miller, and C. Ooi (2000): “The decision tree approach to stock selection.” *Journal of Portfolio Management*, Vol. 27, No. 1, pp. 42–52.

Theofilatos, K., S. Likothanassis, and A. Karathanasopoulos (2012): “Modeling and trading the EUR/USD exchange rate using machine learning techniques.” *Engineering, Technology & Applied Science Research*, Vol. 2, No. 5, pp. 269–272.

Trafalis, T. and H. Ince (2000): “Support vector machine for regression and applications to financial forecasting.” *Neural Networks*, Vol. 6, No. 1, pp. 348–353.

Trippi, R. and D. DeSieno (1992): “Trading equity index futures with a neural network.” *Journal of Portfolio Management*, Vol. 19, No. 1, pp. 27–33.

Tsai, C. and S. Wang (2009): “Stock price forecasting by hybrid machine learning techniques.” *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*, Vol. 1, No. 1, pp. 755–760.

Tsai, C., Y. Lin, D. Yen, and Y. Chen (2011): “Predicting stock returns by classifier ensembles.” *Applied Soft Computing*, Vol. 11, No. 2, pp. 2452–2459.

Wang, J. and S. Chan (2006): “Stock market trading rule discovery using two-layer bias decision tree.” *Expert Systems with Applications*, Vol. 30, No. 4, pp. 605–611.

Wang, Q., J. Li, Q. Qin, and S. Ge (2011): “Linear, adaptive and nonlinear trading models for Singapore Stock Market with random forests.” Proceedings of the 9th IEEE International Conference on Control and Automation, pp. 726–731.

Wei, P. and N. Wang (2016): “Wikipedia and stock return: Wikipedia usage pattern helps to predict the individual stock movement.” Proceedings of the 25th International Conference Companion on World Wide Web, Vol. 1, pp. 591–594.

Żbikowski, K. (2015): “Using volume weighted support vector machines with walk forward testing and feature selection for the purpose of creating stock trading strategy.” *Expert Systems with Applications*, Vol. 42, No. 4, pp. 1797–1805.

Zhang, G., B. Patuwo, and M. Hu (1998): “Forecasting with artificial neural networks: The state of the art.” *International Journal of Forecasting*, Vol. 14, No. 1, pp. 35–62.

Zhu, M., D. Philpotts and M. Stevenson (2012): “The benefits of tree-based models for stock selection.” *Journal of Asset Management*, Vol. 13, No. 6, pp. 437–448.

Zhu, M., D. Philpotts, R. Sparks, and J. Stevenson, Maxwell (2011): “A hybrid approach to combining CART and logistic regression for stock ranking.” *Journal of Portfolio Management*, Vol. 38, No. 1, pp. 100–109.

NOTE

[1] You are probably aware of at least one large hedge fund that monitors the emotional state of their research analysts on a daily basis.