# CHAPTER 10
# Bet Sizing

## 10.1 MOTIVATION

There are fascinating parallels between strategy games and investing. Some of the best portfolio managers I have worked with are excellent poker players, perhaps more so than chess players. One reason is bet sizing, for which Texas Hold'em provides a great analogue and training ground. Your ML algorithm can achieve high accuracy, but if you do not size your bets properly, your investment strategy will inevitably lose money. In this chapter we will review a few approaches to size bets from ML predictions.

## 10.2 STRATEGY-INDEPENDENT BET SIZING APPROACHES

Consider two strategies on the same instrument. Let $m_{i,t} \in [-1, 1]$ be the bet size of strategy $i$ at time $t$, where $m_{i,t} = -1$ indicates a full short position and $m_{i,t} = 1$ indicates a full long position. Suppose that one strategy produced a sequence of bet sizes $[m_{1,1}, m_{1,2}, m_{1,3}] = [.5, 1, 0]$, as the market price followed a sequence $[p_1, p_2, p_3] = [1, .5, 1.25]$, where $p_t$ is the price at time $t$. The other strategy produced a sequence $[m_{2,1}, m_{2,2}, m_{2,3}] = [1, .5, 0]$, as it was forced to reduce its bet size once the market moved against the initial full position. Both strategies produced forecasts that turned out to be correct (the price increased by 25% between $p_1$ and $p_3$), however the first strategy made money (0.5) while the second strategy lost money (−.125).

We would prefer to size positions in such way that we reserve some cash for the possibility that the trading signal strengthens before it weakens. One option is to compute the series $c_t = c_{t,l} - c_{t,s}$, where $c_{t,l}$ is the number of concurrent long bets at time $t$, and $c_{t,s}$ is the number of concurrent short bets at time $t$. This bet concurrency is derived, for each side, similarly to how we computed label concurrency in Chapter 4 (recall the `t1` object, with overlapping time spans). We fit a mixture of two Gaussians on $\{c_t\}$, applying a method like the one described

in López de Prado and Foreman [2014]. Then, the bet size is derived as

$$m_t = \begin{cases} \dfrac{F[c_t] - F[0]}{1 - F[0]} & \text{if } c_t \geq 0 \\ \dfrac{F[c_t] - F[0]}{F[0]} & \text{if } c_t < 0 \end{cases}$$

where $F[x]$ is the CDF of the fitted mixture of two Gaussians for a value $x$. For example, we could size the bet as 0.9 when the probability of observing a signal of greater value is only 0.1. The stronger the signal, the smaller the probability that the signal becomes even stronger, hence the greater the bet size.

A second solution is to follow a budgeting approach. We compute the maximum number (or some other quantile) of concurrent long bets, $\max_i\{c_{i,l}\}$, and the maximum number of concurrent short bets, $\max_i\{c_{i,s}\}$. Then we derive the bet size as $m_t = c_{t,l}\frac{1}{\max_i\{c_{i,l}\}} - c_{t,s}\frac{1}{\max_i\{c_{i,s}\}}$, where $c_{t,l}$ is the number of concurrent long bets at time $t$, and $c_{t,s}$ is the number of concurrent short bets at time $t$. The goal is that the maximum position is not reached before the last concurrent signal is triggered.

A third approach is to apply meta-labeling, as we explained in Chapter 3. We fit a classifier, such as an SVC or RF, to determine the probability of misclassification, and use that probability to derive the bet size. [1] This approach has a couple of advantages: First, the ML algorithm that decides the bet sizes is independent of the primary model, allowing for the incorporation of features predictive of false positives (see Chapter 3). Second, the predicted probability can be directly translated into bet size. Let us see how.

# 10.3 BET SIZING FROM PREDICTED PROBABILITIES

Let us denote $p[x]$ the probability that label $x$ takes place. For two possible outcomes, $x \in \{-1, 1\}$, chapter 15 shows that the Sharpe ratio of the opportunity can be estimated as $z = \frac{p[x=1]-\frac{1}{2}}{\sqrt{p[x=1](1-p[x=1])}} = \frac{2p[x=1]-1}{2\sqrt{p[x=1](1-p[x=1])}}$, with $z \in (-\infty, +\infty)$. Assuming that the Sharpe ratio of opportunities follows a standard Gaussian distribution, we may derive the bet size as $m = 2Z[z] - 1$, where $m \in [-1, 1]$ and $Z[.]$ is the CDF of the standard Gaussian.

For more than two possible outcomes, we follow a one-versus-rest method. Let $X = \{-1,\ldots, 0, \ldots, 1\}$ be various labels associated with bet sizes, and $x \in X$ the predicted label. In other words, the label is identified by the bet size associated with it. For each label $i = 1,\ldots, \|X\|$, we estimate a probability $p_i$, with $\sum_{i=1}^{\|X\|} p_i = 1$. We define $\tilde{p} = \max_i\{p_i\}$ as the probability of $x$, and we would like to test for $H_0 : \tilde{p} = \frac{1}{\|X\|}$.[2] We compute the statistic $z = \frac{\tilde{p} - \frac{1}{\|X\|}}{\sqrt{\tilde{p}(1-\tilde{p})}}$, with $z \in [0., . + \infty)$. We derive the bet size as $m = x \underbrace{(2Z[z] - 1)}_{\in[0,1]}$, where $m \in [-1, 1]$ and $Z[z]$ regulates the size for a prediction $x$ (where the side is implied by $x$).

Figure 10.1 plots the bet size as a function of test statistic. Snippet 10.1 implements the translation from probabilities to bet size. It handles the possibility that the prediction comes from a meta-labeling estimator, as well from a standard labeling estimator. In step #2, it also averages active bets, and discretizes the final value, which we will explain in the following sections.

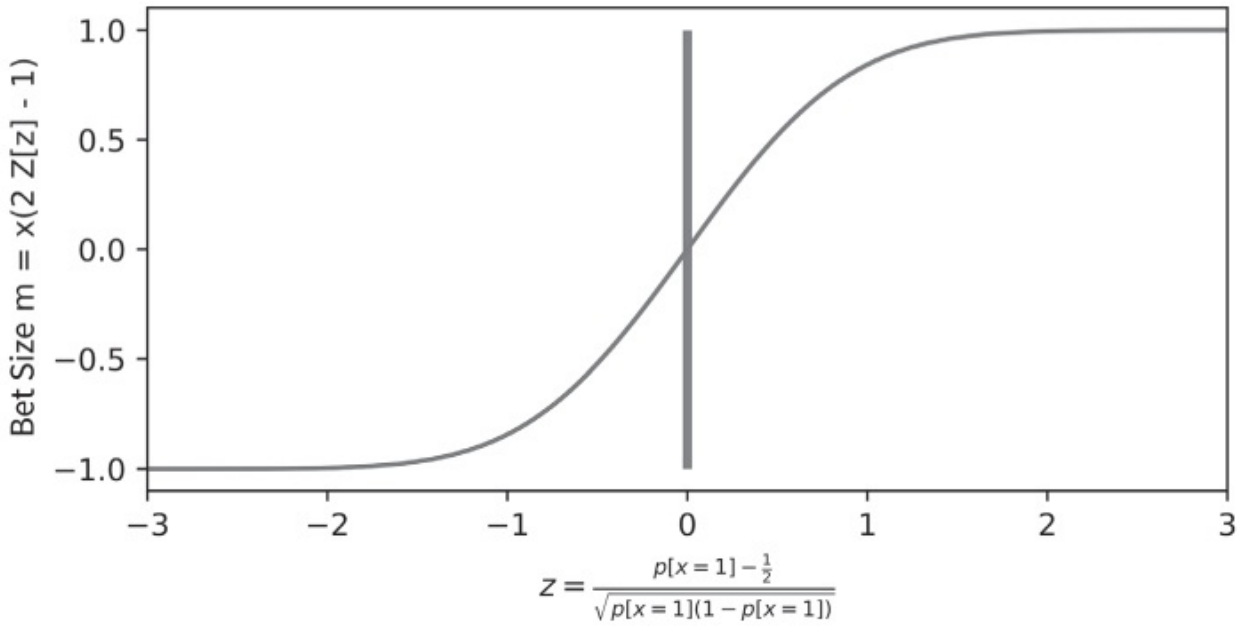


**FIGURE 10.1** Bet size from predicted probabilities

**SNIPPET 10.1 FROM PROBABILITIES TO BET SIZE**

```
def getSignal(events,stepSize,prob,pred,numClasses,numThreads,**kargs):
    # get signals from predictions
    if prob.shape[0]==0:return pd.Series()
    #1) generate signals from multinomial classification (one-vs-rest, OvR)
    signal0=(prob-1./numClasses)/(prob*(1.-prob))**.5 # t-value of OvR
    signal0=pred*(2*norm.cdf(signal0)-1) # signal=side*size
    if 'side' in events:signal0*=events.loc[signal0.index,'side'] # meta-labeling
    #2) compute average signal among those concurrently open
    df0=signal0.to_frame('signal').join(events[['t1']],how='left')
    df0=avgActiveSignals(df0,numThreads)
    signal1=discreteSignal(signal0=df0,stepSize=stepSize)
    return signal1
```

## 10.4 AVERAGING ACTIVE BETS

Every bet is associated with a holding period, spanning from the time it originated to the time the first barrier is touched, `t1` (see Chapter 3). One possible approach is to override an old bet as a new bet arrives; however, that is likely to lead to excessive turnover. A more sensible approach is to average all sizes across all bets still active at a given point in time. Snippet 10.2 illustrates one possible implementation of this idea.

### SNIPPET 10.2 BETS ARE AVERAGED AS LONG AS THEY ARE STILL ACTIVE

```
def avgActiveSignals(signals,numThreads):
    # compute the average signal among those active
    #1) time points where signals change (either one starts or one ends)
    tPnts=set(signals['t1'].dropna().values)
    tPnts=tPnts.union(signals.index.values)
    tPnts=list(tPnts);tPnts.sort()
    out=mpPandasObj(mpAvgActiveSignals,('molecule',tPnts),numThreads,signals=signals)
    return out
#-----------------------------------------
def mpAvgActiveSignals(signals,molecule):
  '''
  At time loc, average signal among those still active.
  Signal is active if:
      a) issued before or at loc AND
      b) loc before signal's endtime, or endtime is still unknown (NaT).
  '''
  out=pd.Series()
  for loc in molecule:
      df0=(signals.index.values<=loc)&((loc<signals['t1'])|pd.isnull(signals['t1']))
      act=signals[df0].index
      if len(act)>0:out[loc]=signals.loc[act,'signal'].mean()
      else:out[loc]=0 # no signals active at this time
  return out
```

# 10.5 SIZE DISCRETIZATION

Averaging reduces some of the excess turnover, but still it is likely that small trades will be triggered with every prediction. As this jitter would cause unnecessary overtrading, I suggest you discretize the bet size as $m^* = \text{round}\left[\frac{m}{d}\right] d$, where $d \in (0, ..1]$ determines the degree of discretization. Figure 10.2 illustrates the discretization of the bet size. Snippet 10.3 implements this notion.

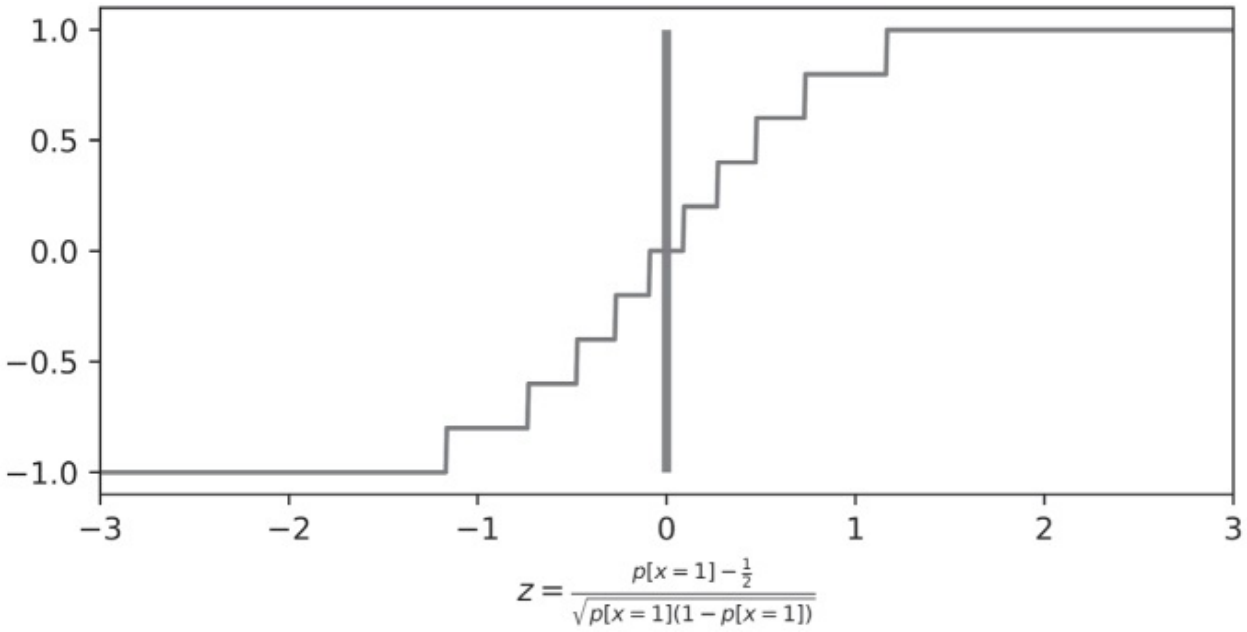


**FIGURE 10.2** Discretization of the bet size, d = 0.2

**SNIPPET 10.3 SIZE DISCRETIZATION TO PREVENT OVERTRADING**

```
def discreteSignal(signal0,stepSize):
    # discretize signal
    signal1=(signal0/stepSize).round()*stepSize # discretize
    signal1[signal1>1]=1 # cap
    signal1[signal1<-1]=-1 # floor
    return signal1
```

# 10.6 DYNAMIC BET SIZES AND LIMIT PRICES

Recall the triple-barrier labeling method presented in Chapter 3. Bar $i$ is formed at time $t_{i,0}$, at which point we forecast the first barrier that will be touched. That prediction implies a forecasted price, $\mathrm{E}_{t_{i,0}}[p_{t_{i,1}}]$, consistent with the barriers' settings. In the period elapsed until the outcome takes place, $t \in [t_{i,0}, t_{i,1}]$, the price $p_t$ fluctuates and additional forecasts may be formed, $\mathrm{E}_{t_{j,0}}[p_{t_{i,1}}]$, where $j \in [i + 1, I]$ and $t_{j,0} \leq t_{i,1}$. In Sections 10.4 and 10.5 we discussed methods for averaging the active bets and discretizing the bet size as new forecasts are formed. In this section we will introduce an approach to adjust bet sizes as

market price $p_t$ and forecast price $f_i$ fluctuate. In the process, we will derive the order's limit price.

Let $q_t$ be the current position, $Q$ the maximum absolute position size, and $\hat{q}_{i,t}$ the target position size associated with forecast $f_i$, such that

$$\hat{q}_{i,t} = \text{int}[m[\omega, f_i - p_t]Q]$$

$$m[\omega, x] = \frac{x}{\sqrt{\omega + x^2}}$$

where $m[\omega, x]$ is the bet size, $x = f_i - p_t$ is the divergence between the current market price and the forecast, $\omega$ is a coefficient that regulates the width of the sigmoid function, and Int[$x$] is the integer value of $x$. Note that for a real-valued price divergence $x$, $-1 < m[\omega, x] < 1$, the integer value $\hat{q}_{i,t}$ is bounded $-Q < \hat{q}_{i,t} < Q$.

The target position size $\hat{q}_{i,t}$ can be dynamically adjusted as $p_t$ changes. In particular, as $p_t \to f_i$ we get $\hat{q}_{i,t} \to 0$, because the algorithm wants to realize the gains. This implies a breakeven limit price $\bar{p}$ for the order size $\hat{q}_{i,t} - q_t$, to avoid realizing losses. In particular,

$$\bar{p} = \frac{1}{|\hat{q}_{i,t} - q_t|} \sum_{j=|q_t + \text{sgn}[\hat{q}_{i,t} - q_t]|}^{|\hat{q}_{i,t}|} L\left[f_i, \omega, \frac{j}{Q}\right]$$

where $L[f_i, \omega, m]$ is the inverse function of $m[\omega, f_i - p_t]$ with respect to $p_t$,

$$L[f_i, \omega, m] = f_i - m\sqrt{\frac{\omega}{1 - m^2}}$$

We do not need to worry about the case $m^2 = 1$, because $|\hat{q}_{i,t}| < 1$. Since this function is monotonic, the algorithm cannot realize losses as $p_t \to f_i$.

Let us calibrate ω. Given a user-defined pair ($x$, $m^*$), such that $x = f_i - p_t$ and $m^* = m[\omega, x]$, the inverse function of $m[\omega, x]$ with respect to $\omega$ is

$$\omega = x^2(m^{*-2} - 1)$$

Snippet 10.4 implements the algorithm that computes the dynamic position size

and limit prices as a function of $p_t$ and $f_i$. First, we calibrate the sigmoid function, so that it returns a bet size of $m^* = .95$ for a price divergence of $x = 10$. Second, we compute the target position $\hat{q}_{i,t}$ for a maximum position $Q = 100$, $f_i = 115$ and $p_t = 100$. If you try $f_i = 110$, you will get $\hat{q}_{i,t} = 95$, consistent with the calibration of $\omega$. Third, the limit price for this order of size $\hat{q}_{i,t} - q_t = 97$ is $p_t < 112.3657 < f_i$, which is between the current price and the forecasted price.

**SNIPPET 10.4 DYNAMIC POSITION SIZE AND LIMIT PRICE**

```
def betSize(w,x):
    return x*(w+x**2)**-.5
#-------------------------------------------------
def getTPos(w,f,mP,maxPos):
    return int(betSize(w,f-mP)*maxPos)
#-------------------------------------------------
def invPrice(f,w,m):
    return f-m*(w/(1-m**2))**.5
#-------------------------------------------------
def limitPrice(tPos,pos,f,w,maxPos):
    sgn=(1 if tPos>=pos else -1)
    lP=0
    for j in xrange(abs(pos+sgn),abs(tPos+1)):
        lP+=invPrice(f,w,j/float(maxPos))
    lP/=tPos-pos
    return lP
#-------------------------------------------------
def getW(x,m):
  # 0<alpha<1
    return x**2*(m**-2-1)
#-------------------------------------------------
def main():
    pos,maxPos,mP,f,wParams=0,100,100,115,{'divergence':10,'m':.95}
    w=getW(wParams['divergence'],wParams['m']) # calibrate w
    tPos=getTPos(w,f,mP,maxPos) # get tPos
    lP=limitPrice(tPos,pos,f,w,maxPos) # limit price for order
    return
#-------------------------------------------------
if __name__=='__main__':main()
```

As an alternative to the sigmoid function, we could have used a power function

$\tilde{m}[\omega, x] = \text{sgn}[x]|x|^{\omega}$, where $\omega \geq 0$, $x \in [-1, 1]$, which results in $\tilde{m}[\omega, x] \in [-1, 1]$. This alternative presents the advantages that:

- $\tilde{m}[\omega, -1] = -1, \tilde{m}[\omega, 1] = 1$.
- Curvature can be directly manipulated through $\omega$.
- For $\omega > 1$, the function goes from concave to convex, rather than the other way around, hence the function is almost flat around the inflexion point.

We leave the derivation of the equations for a power function as an exercise. Figure 10.3 plots the bet sizes (y-axis) as a function of price divergence $f - p_t$ (x-axis) for both the sigmoid and power functions.

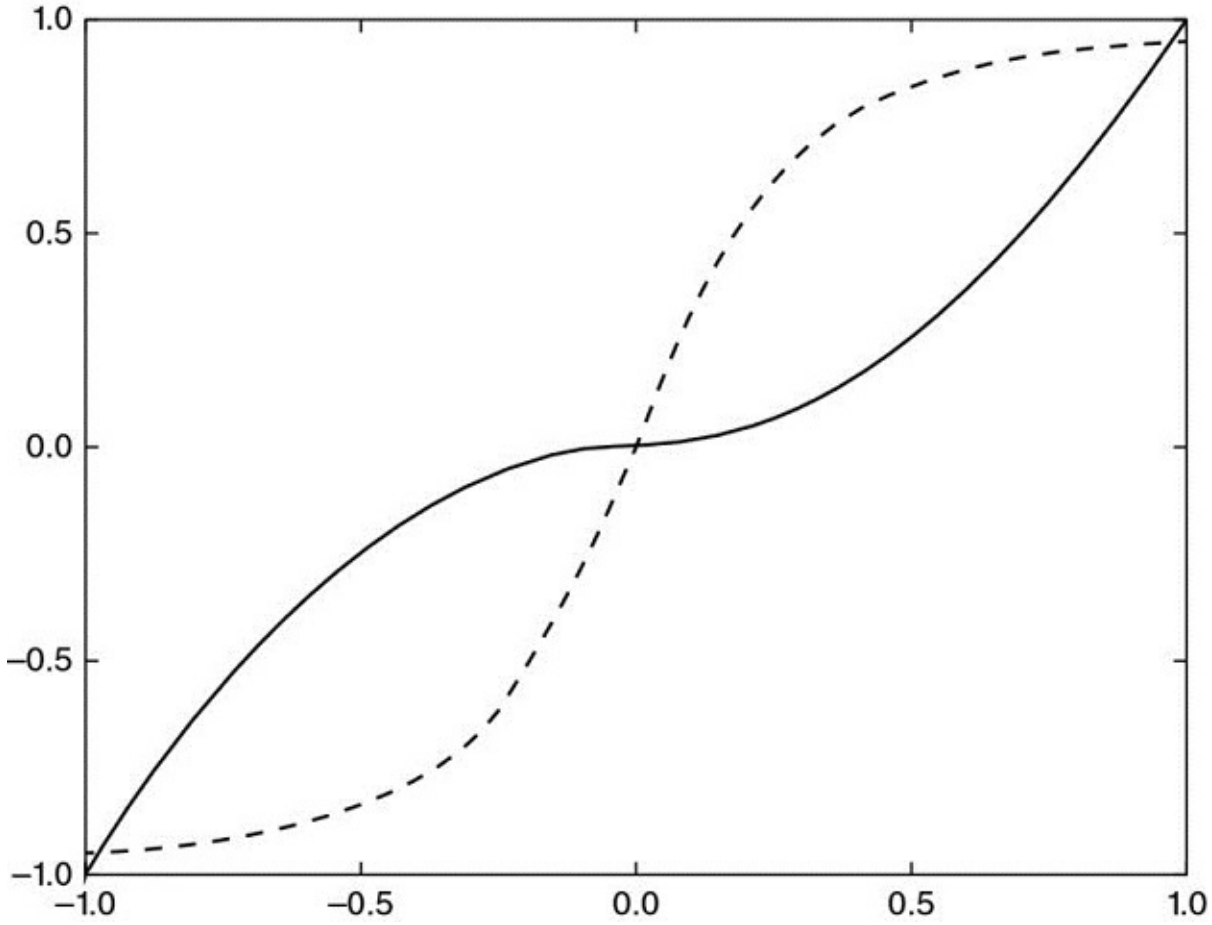


**FIGURE 10.3** $f[x] = sgn[x]|x|^2$ (concave to convex) and $f[x] = x(.1 + x^2)^{-.5}$ (convex to concave)

# EXERCISES

**10.1** Using the formulation in Section 10.3, plot the bet size ($m$) as a function of the maximum predicted probability ($\tilde{p}$) when $||X|| = 2, 3, \ldots, 10$.

**10.2** Draw 10,000 random numbers from a uniform distribution with bounds $U[.5, 1.]$.

a. Compute the bet sizes $m$ for $||X|| = 2$.

b. Assign 10,000 consecutive calendar days to the bet sizes.

c. Draw 10,000 random numbers from a uniform distribution with bounds $U[1, 25]$.

d. Form a pandas series indexed by the dates in 2.b, and with values equal to the index shifted forward the number of days in 2.c. This is a `t1` object similar to the ones we used in Chapter 3.

e. Compute the resulting average active bets, following Section 10.4.

**10.3** Using the `t1` object from exercise 2.d:

a. Determine the maximum number of concurrent long bets, $\bar{c}_l$.

b. Determine the maximum number of concurrent short bets, $\bar{c}_s$.

c. Derive the bet size as $m_t = c_{t,l}\frac{1}{\bar{c}_l} - c_{t,s}\frac{1}{\bar{c}_s}$, where $c_{t,\,l}$ is the number of concurrent long bets at time $t$, and $c_{t,\,s}$ is the number of concurrent short bets at time $t$.

**10.4** Using the `t1` object from exercise 2.d:

a. Compute the series $c_t = c_{t,\,l} - c_{t,\,s}$, where $c_{t,\,l}$ is the number of concurrent long bets at time $t$, and $c_{t,\,s}$ is the number of concurrent short bets at time $t$.

b. Fit a mixture of two Gaussians on $\{c_t\}$. You may want to use the method described in López de Prado and Foreman [2014].

c. Derive the bet size as $m_t = \begin{cases} \frac{F[c_t]-F[0]}{1-F[0]} & \text{if } c_t \geq 0 \\ \frac{F[c_t]-F[0]}{F[0]} & \text{if } c_t < 0 \end{cases}$, where $F[x]$ is the CDF of the fitted mixture of two Gaussians for a value $x$.

d. Explain how this series $\{m_t\}$ differ from the bet size series computed in exercise 3.

**10.5** Repeat exercise 1, where you discretize *m* with a `stepSize=.01`, `stepSize=.05`, and `stepSize=.1`.

**10.6** Rewrite the equations in Section 10.6, so that the bet size is determined by a power function rather than a sigmoid function.

**10.7** Modify Snippet 10.4 so that it implements the equations you derived in exercise 6.

# REFERENCES

López de Prado, M. and M. Foreman (2014): "A mixture of Gaussians approach to mathematical portfolio oversight: The EF3M algorithm." *Quantitative Finance*, Vol. 14, No. 5, pp. 913–930.

Wu, T., C. Lin and R. Weng (2004): "Probability estimates for multi-class classification by pairwise coupling." *Journal of Machine Learning Research*, Vol. 5, pp. 975–1005.

# BIBLIOGRAPHY

Allwein, E., R. Schapire, and Y. Singer (2001): "Reducing multiclass to binary: A unifying approach for margin classifiers." *Journal of Machine Learning Research*, Vol. 1, pp. 113–141.

Hastie, T. and R. Tibshirani (1998): "Classification by pairwise coupling." *The Annals of Statistics*, Vol. 26, No. 1, pp. 451–471.

Refregier, P. and F. Vallet (1991): "Probabilistic approach for multiclass classification with neural networks." Proceedings of International Conference on Artificial Networks, pp. 1003–1007.

# NOTES

[1] The references section lists a number of articles that explain how these probabilities are derived. Usually these probabilities incorporate information about the goodness of the fit, or confidence in the prediction. See Wu et al. [2004], and visit http://scikit-learn.org/stable/modules/svm.html#scores-and-probabilities.

[2] Uncertainty is absolute when all outcomes are equally likely.