3. 레이블링

3.1 동기

- 2장에서는 비정형화된 데이터 셋으로 부터 feature 행렬 X를 구성하는 방안
- 3장은 레이블하는 방안을 알아보자.

3.2 고정기간기법 (The Fixed-Time Horizon Method)

- 특정 바로 추출된 데이터(바) $\{X_i\}_{i=1,\ldots,I}$에 대하여, 레이블은 $y_i \in \{-1, 0, 1\}$로 할당

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

  - τ: 상수 임계값
  - $t_{i,0}$: X_i가 발생한 직후의 바 인덱스
  - $t_{i,0+h}$: $t_{i,0}$ 이후 h번째 바 인덱스
  - $r_{t_{i,0},t_{i,0+h}}$: 바기간(bar horizon) h에 대한 return
- 대부분의 논문들은 타임 바를 사용하여 h는 고정된 기간을 의미.
  - (당시) 최신 레이블링 기법 유명한 논문은 Dixon et al., 2016
  - 하지만, 고정 기간 바의 사용은 아래와 같은 단점이 있음.

    a. 통계적 성질을 갖지 못함. (앞장에서 설명)

    b. 임계값 τ는 관측값의 변동성에 따라 변하지 않는다.
  - 해결 방안(비교적 더 나은)

    a. 수익률의 이동지수가중표준편차 $\sigma_{t,0}$를 기반으로 시간에 따라 변하는 임계값 τ를 설정

    b. 동분산성(homoscedasticity) 특징을 더 잘 반영하는 거래량바나 달러바를 사용.
  - 해결방안의 한계
    - 가격에 기반하여 레이블링한다는 단점(the path followed by price)으로 인하여 핵심 결함을 해결하지 못함
    - 모든 투자전략에는 stop-losss limit(손절 제한)이 있음. 손절 거래를 했을 포지션으로 부터 수익을 낸다는 것은 현실성이 없을 수 있으며, 레이블링 관점에서 이를 고려한 문헌은 거의 없음.

3.3 동적 임계값 계산 (Computing Dynamic Thresholds)

- 베팅에 내제된 리스크로 이익과 손절한도를 설정해보자. 현재의 변동성을 고려하여 너무 높은 수익을 원하거나 너무 낮은 수익을 설정하지 않는 것이 목적.
  - 아래 코드는 지수가중이동표준편차를 `span0` 기간 적용해 일별 변동성을 계산하는 코드. 해당 함수의 출력 값을 이용해 이후의 설명할 이익 손실의 상하한 값의 디폴트 값을 설정할 수 있다.

```
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]:]))
```

```
    try:
        df0=close.loc[df0.index]/close.loc[df0.values].values-1 # daily rets
    except Exception as e:
        print(f'error: {e}\nplease confirm no duplicate indices')
    df0=df0.ewm(span=span0).std().rename('dailyVol')
    return df0
```

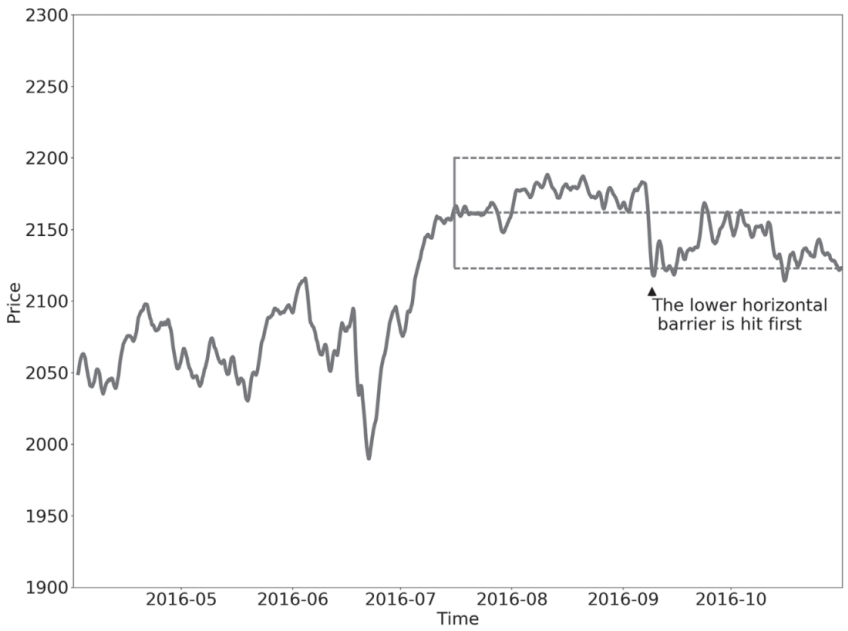
3.4 삼중 배리어 기법 (The Triple-Barrirer Method)

- 3가지 배리어(2 수평 배리어, 1 수직 배리어) 를 놓아 최초로 도달한 배리어의 관측값을 레이블링.
  - 두 수평 배리어: 추정 변동성에 의해 정해짐 (실제 변동 또는 내재 변동에 대한)
  - 수직 배리어: 포지션을 취한 후 지나간 바의 개수에 의해 정의 (expiration limit)
  - 상단 배리어에 먼저 도달 시 1로 레이블, 하단 배리어에 먼저 도달 시 -1로 레이블
  - 수직배리어 도달 시, 2가지 선택 가능 → 실험 필요
    - 손익 부호로 선정 (저자는 전자 선호)
    - 0으로 선정
- 삼중 배리어 기법은 경로의존(path-dependent). 관측값에 레이블을 정하는 것은 $[t_{i,0}, t_{i,0+h}]$를 고려한다는 것이고, h는 수직 배리어이다. $t_{i,1}$은 처음으로 도달한 배리어의 시간, 관측된 특성에 대한 수익률은 $r_{t_{i,0},t_{i,1}}$으로 나타냄. $t_{i,1} \leq t_{i,0} + h$ 이며, 수평배리어는 대칭일 필요는 없음.
- 코드

```
def applyPtSlOnT1(close,events,ptSl,molecule):
    # apply stop loss/profit taking, if it takes place before t1 (end of event)
        """
        close : 가격 (판다스 시리즈)
        events: 데이터프레임
            - t1: 수직배리어 타임스탬프
            - trgt: 수평 배리어의 단위 너비
        ptSl: 상하단 리스트 (음이 아닌 실수값 리스트)
                - ptSl[0]: trgt에 곱해 상단 배리어 너비 설정. 값이 0 이면 상단배리어가 없음.
        molecule: 단일 스레드에 의해 처리되는 이벤트 인덱스 부분집합 리스트(추수 설면)
        """
    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
                # earliest stop loss
        out.loc[loc,'sl']=df0[df0<sl[loc]].index.min()
                # earliest profit taking
        out.loc[loc,'pt']=df0[df0>pt[loc]].index.min()
    return out
```

- 배리어 설정을 [pt, sl, t1] 3가지 값으로 나타내고, 0은 배리어 비활성, 1은 활성이며 총 8가지로 설정 가능하다.

- 배리어 설정
  a. 유용한 설정
    `[1,1,1]`: 가장 표준이 되는 설정. 3가지 배리어의 탈줄 조건을 사용해, 이익 상한 손실 하한 보유기간의 제약이 있음을 의미
    `[0,1,1]`: 손절하지 않는다면 몇 개의 바 이후에 배리어를 탈출
    `[1,1,0]`: 손절하지 않는 한 이익을 실현. 얼마나 오래걸리든 포지션을 보유하므로 비현실 적일 수 있음.
  b. 비교적 비현실적인 설정
    `[0,0,1]`: 고정기간법(The Fixed-Time Horizon Method)와 동일. 해당 방안의 해결방안 적용시 유용할 수 있음
    `[1,0,1]`: 이익이 나거나 설정한 기간 전까지 포지션 유지. 중도 손실을 고려하지 않음.
    `[1,0,0]`: 이익이 날 때까지 포지션을 유지.
  c. 비논리적 설정
    `[0,1,0]`: 목표가 없는 설정으로 손절 전까지 보유
    `[0,0,0]`: 배리어가 없어 포지션 계속 고정. 레이블 발생하지 않음.
- 예시



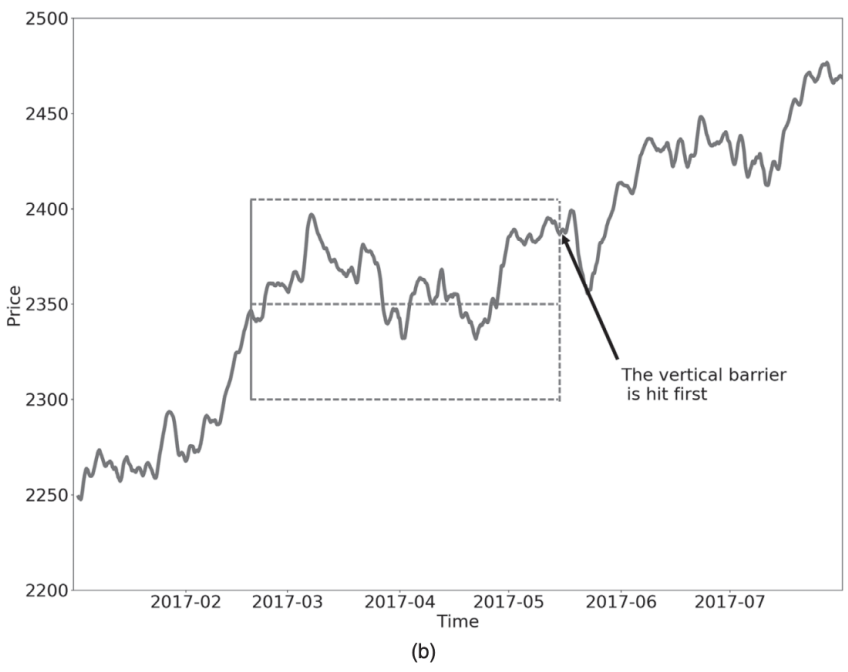


FIGURE 3.1 Two alternative configurations of the triple-barrier method

3.5 방향과 크기 파악(Learning Side and Size)

- ML 알고리즘이 베팅의 방향(매수 매도)과 크기(베팅의 정도)를 학습할 수 있도록 데이터에 레이블링하는 방안 설명.
- 최초 도달 시간 측정 코드 예시

```
def getEvents(close, tEvents, ptSl, trgt, minRet, numThreads, t1=False):
"""
```

```
최초 도달 시간 측정 코드
close: 가격
tEvents: 삼중배리어 시드가 될 타임스탬프 값을 가진 timeindex, 2.5절의 샘플링에 선정된 인덱스
ptSl: 상한 하한 배리어
t1: 수직 배리어의 타임스탬프를 가진 판다스 시리즈
trgt: 수익률의 절대값으로 표현한 목표 판다스 시리즈
minRet: 삼중 배리어 검색을 진행할 때 필요한 최소 목표 수익률
numThreads: 함수에서 현재 동시에 사용하고 있는 스레드 수
"""
    #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)
        ptSl = [ptSl[0],ptSl[0]] # (현재는 symmetric하다는 가정)
    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_)
    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
```

I = 1E6, h=1,000 인경우 계산해야하는 조건의 개수는 단일 금융 상품에 대해 10억개 일 수 있다. 이 때는 병렬계산이 필요하다.(20장)

`mpPandasObj` 함수는 다중 처리함수를 처리(20장에서 설명). 지금은 함수 `applyPtSlOnT1` 가 병렬로 실행되는 것만 기억하고, 이는 각 배리어가 도달한 시점의 타임스탬프를 반환. 베팅의 방향을 `ptsl` 인수로 전달, 항상 매수로 설정하며 최초로 배리어를 도착하는 부분은 방향과 무관할 수 있기 때문이다(?). 위 함수의 출력은 `t1`(최초 배리어 도달 타임스탬프)와 `trgt`(수평 배리어를 생성하고자 사용된 타겟)이다.

- 수직 배리어를 설정하는 코드 예시

```
def addVerticalBarrier(tEvents, close, numDays=1):
"""
tEvents 각 인덱스에 대해 그다음 가격 바 또는 numDays 며칠 이후의 타임스탬프를 바(설정 날짜 지난 후)를 반환
"""
    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]]))
    return t1
```

- 레이블 부여 코드 예시

event 데이터프레임과 가격을 기준으로 계산. `ret`는 최초 배리어에 도착했을 때 실현된 수익률이고, `bin`은 레이블이다.

```
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 메타 레이블링

- 얼마나 매수/매도 할지
- 매수 매도(베팅)의 방향을 설정하는 모델을 이미 갖고 있는 상황을 가정하자.
  - 현업에서는 얼마나 베팅할지도 정해야할 수 있다. ML 알고리즘을 통해 이를 학습하는 것은 쉽지 않을 수 있으며 많은 논문에서 다루지 않음.
  - 이러한 문제를 메타레이블링이라함. → 1차 모델(primary exogenous model)을 어떻게 사용할지 학습하기 때문
  - 위 `getEnvents` 함수의 side 옵션(1차 모델의 방향)을 추가하자. 이 값이 None이 아니라면 메타레이블이 작동하며, 이익 실현과 손절을 구분할 수 있다. 수평배리어는 이전처럼 대칭일 필요는 없음.

```
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=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
```

  - `getBins` 함수 역시 메타레이블을 다룰 수 있도록 확장하면 아래와 같다. out['bin']이 가능한 값은 {-1,0,1}이 아닌 {0,1}이다. 2차 ML모형은 베팅을 할지 말지만을 학습할 것이며 예측값의 확률을 통해 얼마나 베팅할지를 정할 수 있다.

```
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
```

3.7 메타레이블링을 이용하는 방법

- recall($tp/(tp+fn)$)지표는 가설검증의 검증력과 유사하다. f1 점수는 precision과 recall의 조화평균으로 분류기의 효율성(efficiency)을 측정
- 메타레이블링은 더 높은 F-1점수를 원할때 유용함.
  - 1차 모형은 precision이 높지 않더라도 recall이 높은 모형을 개발
  - 메타레이블링을 통해 낮은 precision을 보완
- 대부분의 positive는 1차 모델이 식별했으며, 메타레이블링은 False Positive를 걸러 F1-score를 개선한다.
  - 메타레이블링의 모델은 1차모델의 positive가 참인지 거짓인지를 판단하는 모형이며 이는, 베팅을 실행할지 말지를 결정하는 것이 목적이다.
- 메타레이블링이 강력한 도구인 이유
  - ML의 블랙박스 성질을 개선
  - 과적합 방지 (방향과 크기를 선정)
  - 베팅 크기 예측과 베팅 방향 예측을 분리하여 정교한 전략구조 설정 가능
  - 작은 베팅에 높은 정확도나 큰 베팅에 낮은 정확도를 갖는 것을 방지할 수 있음. (10장)

3.8 퀀터멘털 방법

- recap
  - 1차 모델의 예측을 통해 메타레이블 생성. (수평 배리어 대칭일 필요 없음)
  - 메타레이블에 대하여 2차 모델 학습
  - 1차 모델의 방향, 2차 모델 크기 설정
- 이 때, 1차모델을 ML 모형, 계량경제모형, 사람의 예측 등 다양한 모델로 만들 수 있음.

3.9 불필요한 레이블 제거

- ML 분류기는 클래스 불균형문제에 대하여 취약함.
- 아래 코드를 통해 클래스가 매우 적은 레이블을 제거할 수 있으며, 클래스가 2개가 남거나 임계값을 모두 넘을 때까지 반복.

```
def dropLabels(events, minPct=.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
```

References

- Marcos Lopez de Prado, 2018, Advances in Financial Machine Learning
- codes: https://github.com/BlackArbsCEO/Adv_Fin_ML_Exercises/blob/master/notebooks/Labeling and MetaLabeling for Supervised Classification.ipynb