Chapter 4
Sample Weights

01.

Motivation

1 동기

- **IID 문제**
  - 대부분의 통계 및 머신러닝 문헌은 **IID 가정에 근거**함
  - **그러나** 대부분의 **금융 데이터는 IID하지 않음**
    - 이는 금융 도메인에서 머신러닝 알고리즘의 성능 저하 원인 중 하나
- **목적**
  - 본 장에서는 IID 문제를 해결하고자 함
    - **1) 중첩된 결과를 교정**하는 방법을 통한 샘플링
    - **2) 가중값** 설계
      - 모든 관측이 똑같이 중요한 것은 아니기 때문

1 동기

- **구조**
  - **문제 인식**
    - 금융 시장은 IID하지 않음 **(4.2장)**
  - **해결 방안**
    - 1) **중첩된 결과를 교정**하는 방법을 통한 샘플링 **(4.5장)**
      - **고유성**을 통해 교정
        - 라벨별 고유성 구함**(4.4장)**
          - **공존** 라벨**(4.3장)**
    - **2) 가중값 설계 (4.6-4.8장)**
      - **수익률 기여도**에 따른 가중값 설계(**Return Attribution**)**(4.6장)**
      - **시간**-감쇄(**Time Decay**)**(4.7장)**
      - **부류** 가중값(**Class Weights**)(**4.8장)**

02.

Return Attribute

2 수익률 기여도

- **구조**
  - **문제 인식**
    - 금융 시장은 IID하지 않음 **(4.2장)**
  - **해결 방안**
    - 1) **중첩된 결과를 교정**하는 방법을 통한 샘플링 **(4.5장)**
      - **고유성**을 통해 교정
        - 라벨별 고유성 구함**(4.4장)**
          - **공존** 라벨**(4.3장)**
    - 2) 가중값 설계 (4.6-4.8장)
      - **수익률 기여도**에 따른 가중값 설계 (**Return Attribution**)**(4.6장)**
      - **시간**-감쇄**(4.7장)**
      - **부류** 가중값**(4.8장)**

2 수익률 기여도

- **배경**
  - 1) 고도로 중첩되는 결과는 비중첩 출력에 비해불균형한 가중값
    - 샘플링한 결과가 최대한 다양한 정보를 반영해야 함
      - 샘플링 결과가 서로 이질적인 정보를 갖고 있을수록 좋음
  - 2) 절대수익률이 클수록 더 큰 가중치를 둬야 함
    - 돈이 되는 정보에 더 관심을 둠
  - → **고유성**과 **절대 수익률**을 고려하는 함수를 사용해 관측값에 가중을 둬야 한다.

2 수익률 기여도

- **레이블**
  - **표본 가중값**
    - **목적**
      - 관측값의 가중값(w_i)을 **절대 로그 수익률**의 함수로 나타냄
    - **정의**
      - $\tilde{\boldsymbol{w}}_{\boldsymbol{i}}$: 표준화되지 않은 가중치
      - $\boldsymbol{w}_{\boldsymbol{i}}$: 표준화된 가중치
      - $\boldsymbol{r}_{\boldsymbol{t-1,t}} = \frac{\boldsymbol{p}_{\boldsymbol{t}}}{\boldsymbol{p}_{\boldsymbol{t-1}}} - \mathbf{1}$: $t-1$에서 t 시점 사이의 수익률
      - $\mathbf{1}_{\boldsymbol{t,i}}$: t 시점에 중첩이 되었으면 1, 없으면 0
      - $\boldsymbol{c}_{\boldsymbol{t}} = \sum_{i=1}^{I} \mathbf{1}_{\boldsymbol{t,i}}$: 시간 t에서 공존하는 레이블 개수
      - $\boldsymbol{I}$: 추출 횟수
    - **수식**
      - $\tilde{\boldsymbol{w}}_{\boldsymbol{i}} = \left| \sum_{t_{i,0}}^{t_{i,1}} \frac{\boldsymbol{r}_{\boldsymbol{t-1,t}}}{\boldsymbol{c}_{\boldsymbol{t}}} \right|$
      - $\boldsymbol{w}_{\boldsymbol{i}} = \tilde{w}_i I \left(\sum_{j=1}^{I} \tilde{w}_j \right)^{-1}$
        - $\sum w_i = I$

$$\mathbb{1} = \{1_{t,i}\} = \left.\overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}}^{i}\right\} t \qquad \boldsymbol{c} = \{c_t\} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```python
## 절대 수익률 기여도에 의한 표본 가중값 결정
def mpSampleW(t1, numCoEvents, close, molecule):
    # 수익률 기여에 따른 샘플 가중값 도출
    ret = np.log(close).diff()
    wght = pd.Series(index=molecule)
    for tIn, tOut in t1.loc[wght.index].iteritems():
        wght.loc[tIn] = (ret.loc[tIn:tOut] / numCoEvents.loc[tIn:tOut]).sum()
    return wght.abs()
```

2

수익률 기여도

- **코드 구현**
  - **인자**
    - t1
      - 코드 3.3. **getEvents** 함수(첫 번째 배리어가 도달한 시간 측정)으로부터 얻는 인자
      - **버티컬 배리어**의 타임 스탬프
    - numCoEvents
      - 코드 4.1. **mpNumCoEvents** 함수(레이블의 고유성)으로부터 얻는 인자
      - 각 레이블의 **고유성**
    - Close
      - 데이터로부터 얻은 **종가**
    - molecule
      - 코드 2.4 대칭 **cumsumfilter**(전체 길이가 h런 있을 경우 표본 추출)
        - h기간동안 런 있을 때 바 t 추출
      - h는 코드 3.1의 getDailyVol로부터 얻은 vol 값의 평균 사용

```python
## 절대 수익률 기여도에 의한 표본 가중값 결정
def mpSampleW(t1, numCoEvents, close, molecule):
    # 수익률 기여에 따른 샘플 가중값 도출
    ret = np.log(close).diff()
    wght = pd.Series(index=molecule)
    for tIn, tOut in t1.loc[wght.index].iteritems():
        wght.loc[tIn] = (ret.loc[tIn:tOut] / numCoEvents.loc[tIn:tOut]).sum()
    return wght.abs()
```

```python
trip_barr_events = getEvents(close, filtered_bars, ptSl=[1,1], trgt=daily_vol, minRet=0.01)
t1 = trip_barr_events["t1"]
trip_barr_events.head()
```

	t1	trgt	pt	sl
2020-05-20 09:08:00	2020-05-22 09:38:00	0.010565	1	1
2020-05-20 09:12:00	2020-05-22 09:38:00	0.011713	1	1
2020-05-20 09:16:00	2020-05-22 09:38:00	0.012666	1	1
2020-05-20 09:40:00	2020-05-22 10:07:00	0.014690	1	1
2020-05-20 10:10:00	2020-05-22 10:11:00	0.014602	1	1

```python
NumCoEvents = mpNumCoEvents(close.index, trip_barr_events['t1'], filtered_bars)
NumCoEvents.head()
```

```
시각
2020-05-20 09:08:00    1.0
2020-05-20 09:10:00    1.0
2020-05-20 09:12:00    2.0
2020-05-20 09:14:00    2.0
2020-05-20 09:16:00    3.0
```

```python
close.head()
```

```
시각
2020-05-18 09:00:00.990    47950
2020-05-18 09:02:00.000    47900
2020-05-18 09:04:00.000    47900
2020-05-18 09:06:00.000    48000
2020-05-18 09:08:00.000    48000
Name: close, dtype: int64
```

```python
molecule = getTEventS(close, daily_vol.mean())
molecule[:5]
```

```
DatetimeIndex(['2020-05-18 09:02:00', '2020-05-18 09:10:00',
               '2020-05-18 09:13:00', '2020-05-18 09:32:00',
               '2020-05-18 09:39:00'],
```

2

수익률 기여도

- **코드 구현**
  - **코드**
    - `ret = np.log(close).diff()`
      - log 수익률
    - `wght = pd.Series(index=molecule)`
      - cumsum 필터 기준 series 초기화
    - `for tIn, tOut in t1.loc[wght.index].iteritems():`

      `wght.loc[tIn] = abs((ret.loc[tIn:tOut] / numCoEvents.loc[tIn:tOut]).sum())`
      - $\tilde{w}_i = \left|\sum_{t=t_{i,0}}^{t_{i,1}} \frac{r_{t-1,t}}{c_t}\right|$
    - `w= *= w.shape[0] / w.sum()`
      - $w = \tilde{w}_i * I * \left(\sum_{j=1}^{I} \tilde{w}_j\right)^{-1}$

```
## 절대 수익률 기여도에 의한 표본 가중값 결정
def mpSampleW(t1, numCoEvents, close, molecule):
    # 수익률 기여에 따른 샘플 가중값 도출
    ret = np.log(close).diff()
    wght = pd.Series(index=molecule)
    for tIn, tOut in t1.loc[wght.index].iteritems():
        wght.loc[tIn] = (ret.loc[tIn:tOut] / numCoEvents.loc[tIn:tOut]).sum()
    return wght.abs()
```

```
w = mpSampleW(t1, NumCoEvents, close, filtered_bars)
w.tail()
```

```
2020-08-13 15:02:00    0.002755
2020-08-13 15:07:00    0.002944
2020-08-13 15:08:00    0.003133
2020-08-13 15:10:00    0.003347
2020-08-13 15:13:00    0.003591
dtype: float64
```

```
w.shape[0]
```

```
1758
```

```
w *= w.shape[0] / w.sum()
w.tail()
```

```
2020-08-13 15:02:00    5.886956
2020-08-13 15:07:00    6.291142
2020-08-13 15:08:00    6.696018
2020-08-13 15:10:00    7.152280
2020-08-13 15:13:00    7.674615
dtype: float64
```

03.

Time Decay

3

시간-감쇄

- **구조**
  - **문제 인식**
    - 금융 시장은 IID하지 않음 **(4.2장)**
  - **해결 방안**
    - 1) **중첩된 결과를 교정**하는 방법을 통한 샘플링 **(4.5장)**
      - **고유성**을 통해 교정
        - 라벨별 고유성 구함**(4.4장)**
          - **공존** 라벨**(4.3장)**
    - **2) 가중값 설계 (4.6-4.8장)**
      - **수익률 기여도**에 따른 가중값 설계**(4.6장)**
      - **시간**-감쇄**(4.7장)**
      - **부류** 가중값**(4.8장)**

3 시간-감쇄

- **배경**
  - **시장은 적응적 시스템**
    - 시장이 발달할수록 과거의 예제가 새로운 것보다 더 연관성이 떨어짐
    - 일반적으로 새로운 관측값을 얻게 되면 표본 가중값 감쇄
- **정의**
  - $\boldsymbol{d}[\boldsymbol{x}]$: 시간 감쇄 인자
    - $d[x] \geq 0, \forall x \in \left[0, \sum_{i=1}^{I} \bar{\boldsymbol{u}}_{\boldsymbol{i}}\right]$
    - $d\left[\sum_{i=1}^{I} \bar{\boldsymbol{u}}_{\boldsymbol{i}}\right] = 1$
      - 마지막 가중값에는 감쇄가 없음
  - $\bar{\boldsymbol{u}}_{\boldsymbol{i}}$: 레이블 i의 평균 고유성
  - $\boldsymbol{c} \in (-1, 1]$: 시간 감쇄 함수를 결정하는 사용자 정의 매개변수

3 시간-감쇄

- **전개**
  - If $c \in [0,1]$
    - $d[1] = c$
      - 선형 감쇄
  - If $c \in (-1, 0)$
    - $d[x] = 0 \ \forall x \le -c\sum\bar{u}_i$
    - $d[-c\sum\bar{u}_i] = 0$
    - $[-c\sum\bar{u}_i, \ \sum\bar{u}_i]$ 사이에서 선형 감쇄

- $d = \max\{0, a + bx\}$
  - $d = a + b\sum\bar{u} = 1 \rightarrow a = 1 - b\sum\bar{u}_i$
  - c에 따라
    - $(a)\ d = a + b0 = c \rightarrow b = (1-c)(\sum\bar{u}_i)^{-1}, \forall c \in [0, 1]$
    - $(b)\ d = a - bc\sum\bar{u}_i = 0 \rightarrow b = [(c+1)\sum\bar{u}_i]^{-1}, \forall c \in (-1,0)$

3

시간-감쇄

- 배경
  - 특징
    - $c = 1$
      - 시간 감쇄가 없음
    - $0 < c < 1$
      - 시간에 대해 가중값 감쇄가 선형이라는 의미
      - 가중값은 오래된 정도에 상관없이 양수의 가중값 부여
    - $c = 0$
      - 시간이 갈수록 가중값이 선형으로 0에 수렴한다는 의미
    - $c < 0$
      - 관측값 중 가장 오래된 부분인 cT는 0의 가중값을 받는다.

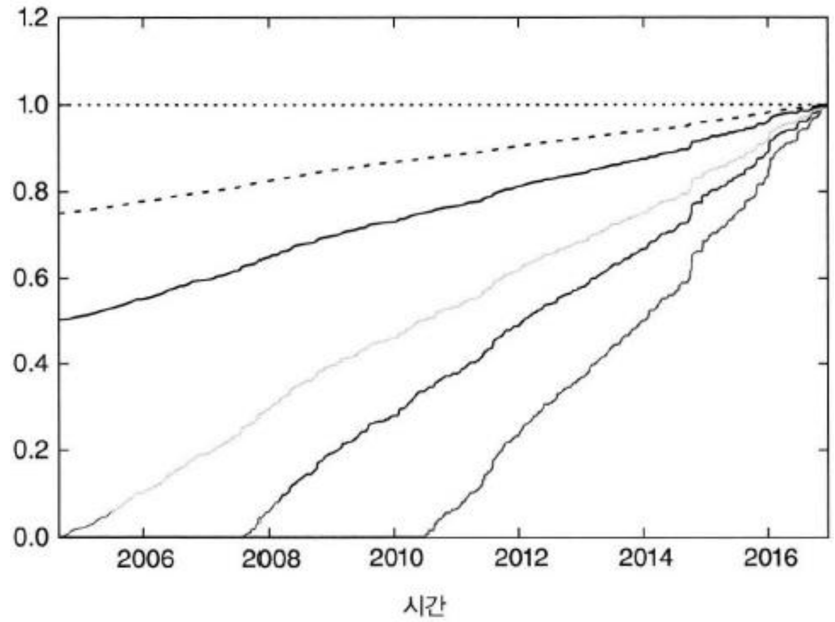

그림 4-3 구간-선형 시간-감쇄 요인

3 시간-감쇄

- **코드 구현**
  - **인자**
    - tw
      - 관측된 고유성
      - Code 4.2. mpSampleTW(이벤트 생명 주기 동안 평균 고유성) 를 통해 구함
    - clfLastW
      - 가장 오래된 관측값의 weight
      - 가장 최신 관측값의 weight = 1

```python
def getTimeDecay(tW, clfLastW=1.0):
    # 관측된 고유성(tW)에 구간-선형 감쇄 적용
    # 최신 관측값  weight = 1, 가장 오래된 관측값 : clfLastW
    clfW = tW.sort_index().cumsum()
    if clfLastW >= 0:
        slope = (1.0 - clfLastW) / clfW.iloc[-1]
    else:
        slope = 1 / ((clfLastW + 1) * clfW.iloc[-1])
    const = 1.0 - slope * clfW.iloc[-1]
    clfW = const + slope * clfW
    clfW[clfW < 0] = 0
    print(const, slope)
    return clfW
```

04.

부류 가중값

4

부류 가중값

- **구조**
  - **문제 인식**
    - 금융 시장은 IID하지 않음 **(4.2장)**
  - **해결 방안**
    - 1) **중첩된 결과를 교정**하는 방법을 통한 샘플링 **(4.5장)**
      - **고유성**을 통해 교정
        - 라벨별 고유성 구함**(4.4장)**
          - **공존** 라벨**(4.3장)**
    - 2) 가중값 설계 (4.6-4.8장)
      - **수익률 기여도**에 따른 가중값 설계**(4.6장)**
      - **시간**-감쇄**(4.7장)**
      - **부류** 가중값**(4.8장)**

4 부류 가중값

- **정의**
  - **얼마 없는 레이블에 가중값을 교정**
  - **중요한 부류의 빈도수가 낮을 경우 특히 중요함**
    - 이런 드문 레이블에 가중값을 더 높게 주지 않으면 흔한 레이블에 대해서만 정확도 극대화
- **방법**
  - Sklearn에 부류 가중값 파라미터
    - Class_weight[j]
- **전개**
  - 금융 응용에 있어서 분류기 알고리즘의 표준 레이블 : $\{-1, 1\}$
    - 0인 경우에는 중립 임계값인 0.5보다 약간 높거나 낮은 확률로 예측
    - Class_weight="balanced"
    - 관측값에 가중값을 재부여해 모든 부류가 동일한 빈도로 나타나도록 해줌
  - 배깅 분류기
    - Class_weight = "balanced_subsample"
    - Class_weight="balanced"가 전체 데이터 세트가 아닌 부트스트랩 표본에 적용

Thank you