발표자 : 이상목

전략명 : Bag of Beta - Random Forest 알고리즘 활용 종목 단기 방향성 예측

전략의 아이디어 :

베타는 시장 대비 변동성을 나타내 주는 지표입니다.

따라서, 관측하는 시점과 기간에 따라 다른 값을 보여주게 됩니다.

관측 시점에 따라 베타를 변하게 하는 원인은 따로 존재하고 있을 것 입니다.

그렇다면, 한 시점을 기준으로 다양한 기간의 베타를 관찰하는 것 만으로도 투자의사 결정이 가능할지 궁금했습니다.

베타에 영향을 끼치는 요소를 특정하기는 굉장히 까다로운 작업이라 생각됩니다.

하지만 머신러닝 알고리즘은 이를 통해 새로운 시각을 얻어낼 수도 있지 않을까?

라는 생각으로 시도해 보게 되었습니다.

전략의 선택 이유 :

전부터 머신러닝 알고리즘에 금융 데이터를 활용하여 결과를 얻어보고 싶었습니다.

데이터들의 학습을 통해 간단하게 결과를 얻어낼 수 있지만,

'왜 알고리즘이 이런 결정을 내렸는가?'

에 대한 사항은 설명하기 힘든 단점을 가지고 있는 것이 머신러닝 알고리즘입니다.

그래서 특징적인 다양한 팩터를 사용하여 이를 활용하기 보단,

의미를 가지고 있겠지만 사람이 직접 관계를 해석하기 힘든 팩터를 사용해 보려 하였습니다.

종목별로 다양한 시점의 베타를 관측하여 이 다양한 시점의 베타들만으로

머신러닝 알고리즘이 인사이트를 얻어낼 수 있을지 확인해 보고자 합니다.

NLP에서 사용하는 Bag of Words model에서 이름을 빌려

Bag of Beta라고 이름을 붙여 보았습니다.

구체적인 전략 실행 방법 :

코스피 종목만 활용

1년에 가까운 영업일 230일간의 다양한 과거 베타를 관측
모의투자 기간과 비슷한 50 영업일 이후의 단기 방향성을 구분하여 분류 문제로 해결합니다.

Features

관측한 베타는 (기간 베타)와 (스텝 베타)로, 총 46+45=91개, 구간은 다음과 같습니다.
※제가 임의로 지은 이름입니다.

[0~230, 5~230, 10~230, … , 220~230, 225~230, 220~225, 215~220, 210~215, 205~210, … , 5~10, 0~5]

(기간 베타)
관측 시점을 기준으로 다양한 기간의 베타를 관측합니다.
과거 230, 225, 220, 215, 210, 205, …, 15, 10, 5 일간의 기간 베타들을 구합니다.(총 46개)

(스텝 베타)
230일 이전부터 관측 시점까지 5일씩의 베타를 구합니다.
0~5, 5~10, 10~15, …, 210~215, 215~220, 220~225일간 5일씩의 베타를 구합니다.(총 45개)
225~230 구간의 베타는 기간 베타에 중복되는 부분이 있으므로 제외합니다.

구현 코드는 첨부의 getBoB() 함수로 정의되어 있습니다.

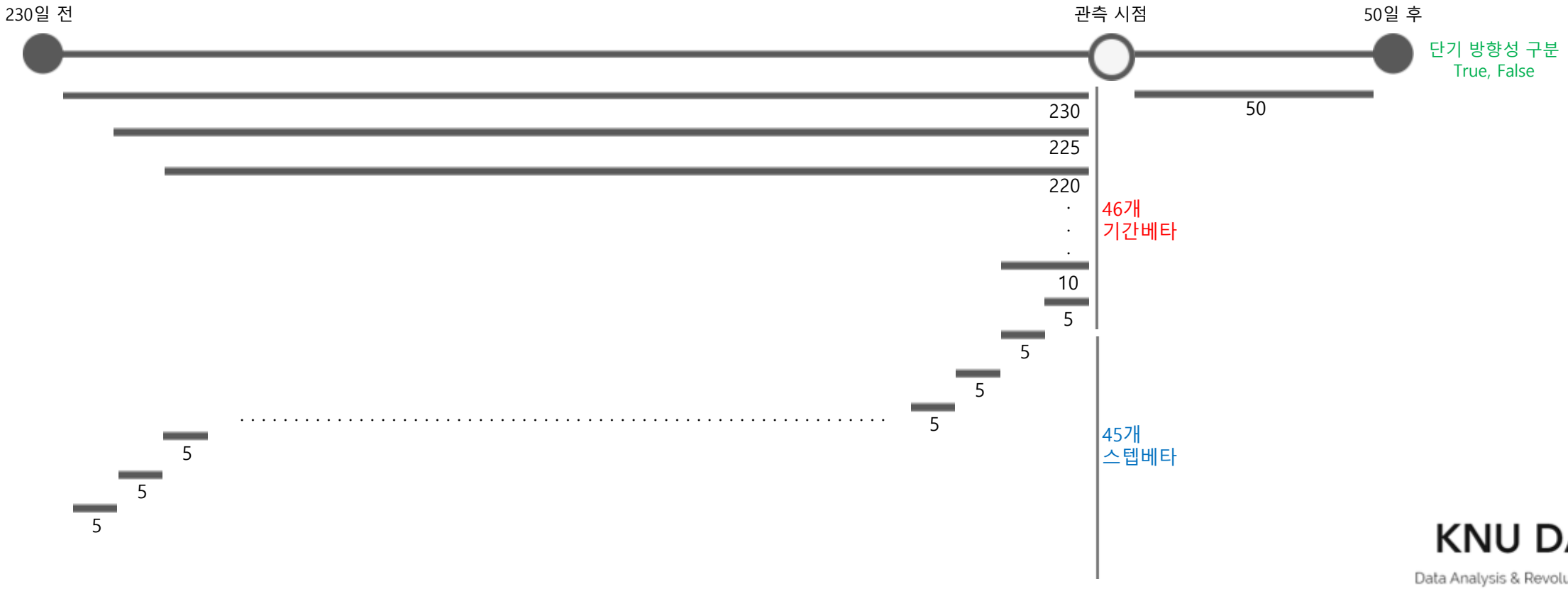
구체적인 전략 실행 방법 cont. :

Result

관측 시점을 기준으로 50영업일 이후의 단기 방향성을 구분합니다.

(50 영업일 이후 수익률 > 0) and (50 영업일 이후 수익률 > 50 영업일 이후 시장 수익률)

두 가지 조건 모두 만족하는 경우를 기준으로 단기 방향성을 구분합니다.



KNU DART

Data Analysis & Revolutionary Think

구체적인 전략 실행 방법 cont. :

학습용 데이터 생성

```
%%time
#230시점 생성
index_list = []
BoB_list = []
for period in period_list:
  print(period)
  for i, ticker in enumerate(pricedata_kospi.columns):
    print('₩r%d/%d'%(i+1, len(pricedata_kospi.columns)), end='')
    state, getedBoB = getBoB(ticker, period)
    index_list.append(state)
    BoB_list.append(getedBoB)
  print()
```

```
230
784/784
CPU times: user 2min 5s, sys: 1.98 s, total: 2min 7s
Wall time: 2min 4s
```

```
BagOfBeta_df_230 = pd.DataFrame(BoB_list, index = index_list)
```

```
classified_list = []
for period in period_list:
  print(period)
  for i, ticker in enumerate(pricedata_kospi.columns):
    print('₩r%d/%d'%(i+1, len(pricedata_kospi.columns)), end='')
    classified_list.append(getClassified(ticker, period))
  print()
```

```
230
784/784
```

```
BagOfBeta_df_230['result'] = classified_list
BagOfBeta_df_230_droped = BagOfBeta_df_230.dropna()
```

1. 한 시점을 기준으로 코스피 784개 종목에 대한 Bag of Beta 생성. 1개 시점당 약 2분 소요
2. result셋 생성
3. NaN값 제거(거래 중지, 추후 상장 등의 이슈로 발생)

A025890_2016-12-08	1.106322	1.072390
A010580_2016-12-08	1.749855	1.757707
A008500_2016-12-08	0.230774	0.232657

712 rows × 92 columns

예시
230 시점(2016-12-08)으로 만들어진 데이터
row - NaN값 제거로 인해 784 -> 712
col - 91개 관측 베타 + result -> 92

KNU DART
Data Analysis & Revolutionary Think

구체적인 전략 실행 방법 cont. :

학습용 데이터들

A025890_2016-12-08	1.106322	1.072390
A010580_2016-12-08	1.749855	1.757707
A008500_2016-12-08	0.230774	0.232657

712 rows × 92 columns

230

A025890_2017-11-17	0.267079	0.263221
A010580_2017-11-17	0.508732	0.557618
A008500_2017-11-17	0.069427	0.073402

1439 rows × 92 columns

230_460

A025890_2019-10-07	0.658543	0.703296
A010580_2019-10-07	0.619922	0.644286
A008500_2019-10-07	0.345035	0.311154

2893 rows × 92 columns

230_460_690_920

관측 기간이 겹치지 않도록 230일 단위로 관측 시점 설정
시점을 추가한 다양한 데이터 셋을 활용
시점이 추가될수록 데이터 셋의 크기가 증가
시점 별 데이터 셋으로 각기 모델을 학습하여 변화 확인

하이퍼 파라미터

```
1 %%time
2 rfc_model_230_460_690_920_1150 = RandomForestClassifier(n_estimators=1000, random_state=777, max_depth = 500, min_samples_split = 0.001)
3 rfc_model_230_460_690_920_1150.fit(train_x, train_y)
```

```
CPU times: user 18.4 s, sys: 60.8 ms, total: 18.5 s
Wall time: 18.5 s
```

(n_estimators=1000, random_state=777, max_depth = 500, min_samples_split = 0.001)
230_460_690_920_1150 기준(row - 3620) 학습에 약 19초 소요(Colab, CPU)

구체적인 전략 실행 방법 cont. :

모델 정확도 개선

```
1 pred = rfc_model_230.predict(test_x)
2
3 print("Accuracy: %s" % str(rfc_model_230.score(test_x, test_y)))
4 print("Confusion Matrix")
5 print(confusion_matrix(pred, test_y, labels=[True, False]))
```

```
Accuracy: 0.6433566433566433
Confusion Matrix
[[54 24]
 [27 38]]
```

230

```
1 pred = rfc_model_230_460.predict(test_x)
2
3 print("Accuracy: %s" % str(rfc_model_230_460.score(test_x, test_y)))
4 print("Confusion Matrix")
5 print(confusion_matrix(pred, test_y, labels=[True, False]))
```

```
Accuracy: 0.8298611111111112
Confusion Matrix
[[127  31]
 [ 18 112]]
```

230_460

```
1 pred = rfc_model_230_460_690_920.predict(test_x)
2
3 print("Accuracy: %s" % str(rfc_model_230_460_690_920.score(test_x, test_y)))
4 print("Confusion Matrix")
5 print(confusion_matrix(pred, test_y, labels=[True, False]))
```

```
Accuracy: 0.92573402417962
Confusion Matrix
[[304  32]
 [ 11 232]]
```

230_460_690_920

※ Confusion Matrix(혼동 행렬)
- 어떤 정보를 무엇으로 몇 개나 판단했는지 보여줌

	True	False
True	304	32
False	11	232

True를 True로 판단 - 304
True를 False로 판단 - 32
False를 True로 판단 - 11
False를 False로 판단 - 232

학습 데이터가 늘어날수록 정확도가 개선되는 모습 확인 가능(train : test = 8 : 2)
하지만 result 셋이 관측일로부터 50영업일 후의 정보로, 미래참조 이슈에서 자유롭지 못함

구체적인 전략 실행 방법 cont. :

모델 검증

```
test_x = BagOfBeta_df_1150_droped[BagOfBeta_df_1150_droped.columns[:-1]]
test_y = BagOfBeta_df_1150_droped[BagOfBeta_df_1150_droped.columns[-1]]
```

각기 모델들과 전혀 겹치지 않는 1150시점의 데이터를 관측하여 각 모델별로 정확도 검증

```
pred = rfc_model_230.predict(test_x)

print("Accuracy: %s" % str(rfc_model_230.score(test_x, test_y)))
print("Confusion Matrix")
print(confusion_matrix(pred, test_y, labels=[True, False]))
```

```
Accuracy: 0.7565337001375516
Confusion Matrix
[[279  84]
 [ 93 271]]
```

230

```
pred = rfc_model_230_460.predict(test_x)

print("Accuracy: %s" % str(rfc_model_230_460.score(test_x, test_y)))
print("Confusion Matrix")
print(confusion_matrix(pred, test_y, labels=[True, False]))
```

```
Accuracy: 0.9284731774415406
Confusion Matrix
[[362  42]
 [ 10 313]]
```

230_460

```
pred = rfc_model_230_460_690_920.predict(test_x)

print("Accuracy: %s" % str(rfc_model_230_460_690_920.score(test_x, test_y)))
print("Confusion Matrix")
print(confusion_matrix(pred, test_y, labels=[True, False]))
```

```
Accuracy: 0.9614855570839065
Confusion Matrix
[[368  24]
 [  4 331]]
```

230_460_690_920

미래참조 이슈에서 자유로운 데이터를 활용해도
모델이 실제로 유효하게 분류하였음을 확인

데이터가 많아질수록 정확도가 높아짐을 확인

가장 많은 데이터를 학습한 모델의 경우
727개 데이터 중 699개를 정확하게 분류해냄

구체적인 전략 실행 방법 cont. :

스크리닝

정확도를 최대한 높이기 위해 230, 460, 690, 920, 1150 시점의 데이터를 모두 학습한 모델 사용

가장 최근 시점(2021-03-29, 데이터 다운로드일)의 기간 베타를 관측하여 예측 결과 획득

```
pred = rfc_model_230_460_690_920_1150.predict(BagOfBeta_df_1284_droped)
```

369개의 종목이 스크리닝 되었음
너무 많으므로 실제 모의투자에서는 대표본 30여개를 추출하여 투자 예정.

```
1 len(screened)
```

```
369
```

결론 :

단기 방향성 예측이지만, 베타만 활용한 결과인데 정확도가 비정상적으로 높다고 느껴짐
놓친 부분이 있을 것으로 예상

하이퍼 파라미터는 휴리스틱하게 접근해서 조금씩 바꿔보며 결과가 좋아지는 방향으로 튜닝함

학습은 20초 정도밖에 안 걸리는데 베타 계산이 너무 오래 걸려서 힘들었음..

시작하면서 좋은 결과가 있기 힘들 것이라 예상했는데, 의외로 결과가 좋아서 놀라움

실제 50일 후 스크린 된 369개 종목 중 얼마나 정확히 맞추었을 지 기대됨

단기 방향성에 대한 예측만 있고, 수치 정보는 없는 점은 아쉬움

피드백 대 환영 및 대 감사합니다

KNU DART
Data Analysis & Revolutionary Think

Appendix :

김동영 애널리스트님의 랜덤 포레스트 기법을 활용한 머신러닝 기반 모델
https://www.samsungpop.com/common.do?cmd=down&saveKey=research.pdf&fileName=5020/20191105150227789K_01_32.pdf&contentType=application/pdf

이하는 코드 및 결과 주피터 노트북 문서
작업 사항 포함 - colab
https://colab.research.google.com/github/SNMHZ/DART_8TH/blob/master/mock_investing/2021DART_%EB%AA%A8%EC%9D%98%ED%88%AC%EC%9E%90_Bag_of_Beta_%EC%9E%91%EC%97%85%EC%83%81%ED%99%A9%ED%8F%AC%ED%95%A8.ipynb?hl=ko ※제 코딩 과정이 모두 포함되어 있습니다.

핵심 코드 모음 - colab
https://colab.research.google.com/github/SNMHZ/DART_8TH/blob/master/mock_investing/2021DART_%EB%AA%A8%EC%9D%98%ED%88%AC%EC%9E%90_Bag_of_Beta_%ED%95%B5%EC%8B%AC%EC%BD%94%EB%93%9C%EB%AA%A8%EC%9D%8C.ipynb?hl=ko ※주요 코드만 뽑아 주석이 달려있습니다.

github.com/SNMHZ/DART_8TH