Quantitative Issue

2019. 11. 18

코스피 단기예측 AI 모델

랜덤 포레스트 기법을 활용한 머신러닝 기반 모델

김동영, CFA
Quant Analyst
dy76.kim@samsung.com
02 2020 7839

원동은
Quant Analyst
de.won@samsung.com
02 2020 7982

본 자료는 먼저 머신러닝 기법인 결정 트리와 랜덤 포레스트의 이론과 사용방법을 설명한다. 그리고, 랜덤 포레스트 기법을 활용하여 코스피를 단기 예측하는 AI 모델을 소개하고 이에 대한 정보를 제공한다.

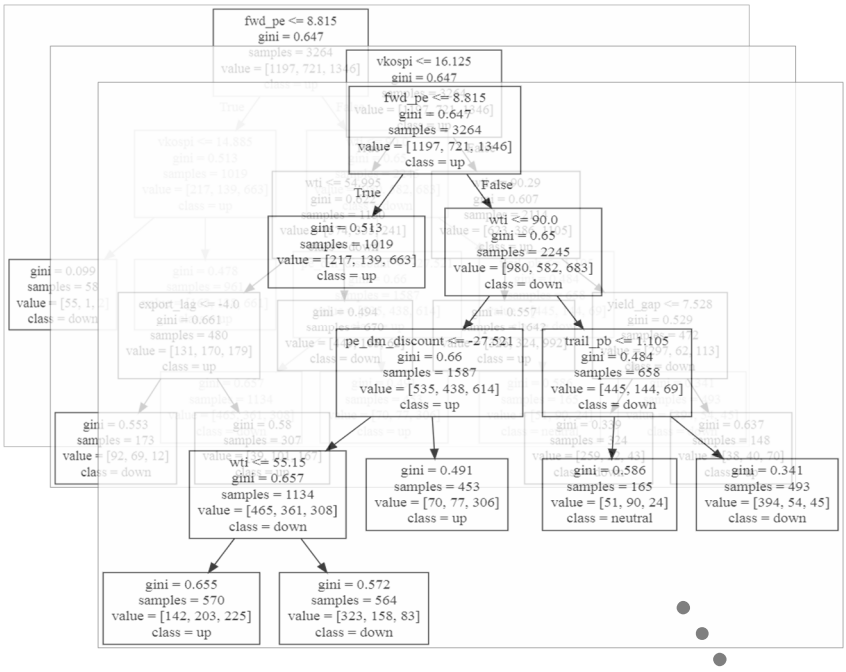
결정 트리 기법

결정 트리(Decision Tree, 의사결정나무라고도 함)는, 나무 형태의 판단 규칙 조합을 만들어서 기존 데이터를 학습하고 그 다음에 예측하는 알고리즘을 말한다. 쉽게 생각하면 어렸을 적의 '스무고개' 놀이와 같은 개념이다.

랜덤 포레스트 기법

앞서 언급한 '결정 트리' 모델에 배깅(bagging) 기법을 적용한 앙상블 모델을, 특별히 '랜덤 포레스트' 모델이라고 한다. 하나의 데이터 소스에서 랜덤하게 데이터들을 만들어 여러 개의 학습된 결정 트리 모델을 만들고 이를 종합하는 방식이다.

랜덤 포레스트 개념도



참고: 랜덤 포레스트는 수십 혹은 수백 개의 '결정 트리' 모델을 만든 다음, 이를 종합해서 판단함
자료: 삼성증권

랜덤 포레스트 기반 단기예측 모델

랜덤 포레스트 기법을 사용하여 코스피 시장을 단기예측하는 모델을 개발하였고, 모델 코드를 제시한다. 일별 시계열 데이터와 정교한 모델을 통해, 기존 예측 모델 대비 정확도가 크게 상향된 것으로 확인된다.

* 본 자료는 Python 언어와 sklearn 라이브러리를 기준으로 설명함

Contents

I. 머신러닝 기초

작년 글로벌 투자업계에서 화제가 된 뉴스가 하나 있었다. CFA 협회가 시험 커리큘럼에 머신러닝(Machine Learning) 내용을 새롭게 넣었다는 뉴스가 바로 그 것이다. 그리고 올해 들어서 CFA 협회는, 2020년 커리큘럼에 Machine Learning과 Big Data Projects에 대한 실무적이고 상세한 커리큘럼 내용을 추가했다. 이 것의 의미는 앞으로 CFA가 되기 위해서는 머신러닝에서 사용되는 주로 알고리즘이 어떤 것이 있는지, 어떤 프로세스를 통해 머신러닝을 실행하는지를 알아야 한다는 것이다. 그만큼 선진 투자업계에서는 머신러닝, 인공지능이 이미 투자 의사 결정에 밀접하게 사용되고 있는 상황이라고 하겠다. 이 것이 시사하는 바는 크다. 국내 투자업계도 신기술에 대한 좀 더 개방적인 자세가 필요하다.

머신러닝(기계학습)이라고 하는 것은 인공지능의 한 분야로, 경험적 데이터를 기반으로 기계(컴퓨터)가 학습을 하고 이를 통해 새로운 데이터를 예측할 수 있는 알고리즘 및 기술을 말한다.

일반적으로 머신러닝 시스템은 학습하는 형태에 따라서 '지도 학습, 비지도 학습, 강화 학습' 등의 범주로 주로 나눈다.

지도 학습(supervised learning)은 학습 데이터에 입력값과 그의 정답인 출력값이 같이 포함되어 있고, 이를 학습하는 형태다. 이를 통해 회귀분석이나 분류 작업을 하게 된다. 쉽게 말해, 과거의 입력값과 출력값을 가지고 패턴을 학습하여, 새로운 입력값이 들어오면 이에 맞는 출력값을 예측하는 방식이다. 지도 학습의 알고리즘에는 penalized regression, SVM, k-최근접 이웃, 결정 트리, 랜덤 포레스트, 나이브 베이지안 등이 있다. 본 리포트는 결정 트리와 랜덤 포레스트 기법에 대해서 상세한 설명을 실었다.

비지도 학습(unsupervised learning)은 입력값만으로 학습 데이터가 구성되어 있는 경우다. 비지도 학습에서는 군집화, 차원 축소 등의 일을 할 수 있다. 즉, 흩어져 있는 데이터들을 유사한 그룹을 묶는 작업 등을 하는 분야다. 관련 알고리즘에는 PCA, k-평균 군집 등이 있다.

강화 학습(reinforcement learning)은 에이전트가 환경을 관찰해서 행동을 실행하고 그 결과로 보상을 받는 환경을 만들어서, 가장 큰 보상을 얻기 위해 최상의 전략을 스스로 학습하게 만드는 알고리즘이다. 바둑대회에서 우승한 알파고가 강화 학습의 대표적인 예다.

머신러닝과 과적합(Overfitting) 이슈

앞에서 본 머닝러닝의 각각의 알고리즘도 중요하지만, 필자가 생각하기에 머신러닝 기법에서의 가장 큰 장점은 과적합(overfitting)에 대한 여러 대비책을 강구했다는 점이다.

'과적합'은 학습이 너무 과도하게 되어서 학습 데이터에 대해서는 높은 정확도를 보이지만, 실제 적용 시에는 성능이 떨어지고 예측이 틀리게 되는 현상을 말한다. 쉽게 말해서 "모델대로 했더니 과거까지는 맞았는데 미래는 안 맞더라" 혹은 "백테스팅에서는 잘 나오는데 실제 수익률은 안 나온다"라고 하는 현상을 말한다. 이런 과적합은 모든 '모형화' 작업에서 필연적으로 생길 수밖에 없는 문제다.

머신러닝 분야에서는 과적합을 최대한 막을 수 있도록 여러 가지 프로세스가 고안되어 사용되고 있다.

우선, 머신러닝 기법에서는 전체 사용 가능한 데이터셋을 훈련 데이터(training data)와 테스트 데이터(test data)로 나누는 것을 기본으로 하고 있다. 모델 학습 시에는 training data만을 사용한다. test data는 모델 학습에 전혀 사용하지 않는다. 모델의 실전 성능을 체크하는 out-of-sample 테스트에서 test data가 처음 사용된다. 이를 통해 현실적인 모델의 성능을 확인할 수 있다. training data를 넣은 모델 성능보다 test data를 넣은 모델 성능이 갑자기 떨어진다면, 현재 모델이 과적합된 것으로 의심해볼 수 있다. 보통 test data에는 전체 데이터의 20%~25%를 떼어놓는다. 여기서 test data를 따로 준비한다는 것은, 그만큼 데이터가 더 많이 필요하다는 뜻도 된다.

또한, 머신러닝은 검증 데이터(validation data)란 것을 따로 사용하는 경우가 많다. validation data는 모델의 검증과 튜닝을 위해 사용되는 데이터다. 예를 들어, A 모델과 B 모델 중에서 어떤 것을 선택할지 고민할 때, training data를 통해 모델 학습을 하고 validation data를 통해서는 A 모델과 B 모델의 실제 오차를 만들어서 이를 비교해 좋은 모델을 고를 수 있다. validation data는 중간 검증용 데이터라고 할 수 있다. 이와는 다르게, test data는 최종 선택된 모델에서 out-of-sample 오차(일반적인 모델 성능을 의미하는)를 확인할 때에만 사용된다. validation data에 너무 많은 데이터를 빼기지 않기 위해 일반적으로 교차 검증(cross-validation)이란 기법이 사용된다. 교차 검증법에서는, 우선 training data를 여러 서브셋으로 나눈다. 이 서브셋들의 조합을 만들어 여러 개의 training data, validation data를 조합을 다시 만들고, 이를 가지고 검증을 여러 번 반복하는 기법이다(챕터 3에서 상세 설명함).

이런 머신러닝 상의 training & test data 분리 기법, cross-validation 기법들이, 과적합되지 않고 안정적인 모델을 구축하는 데 실질적인 도움을 준다.

Contents

II. 결정 트리 기법

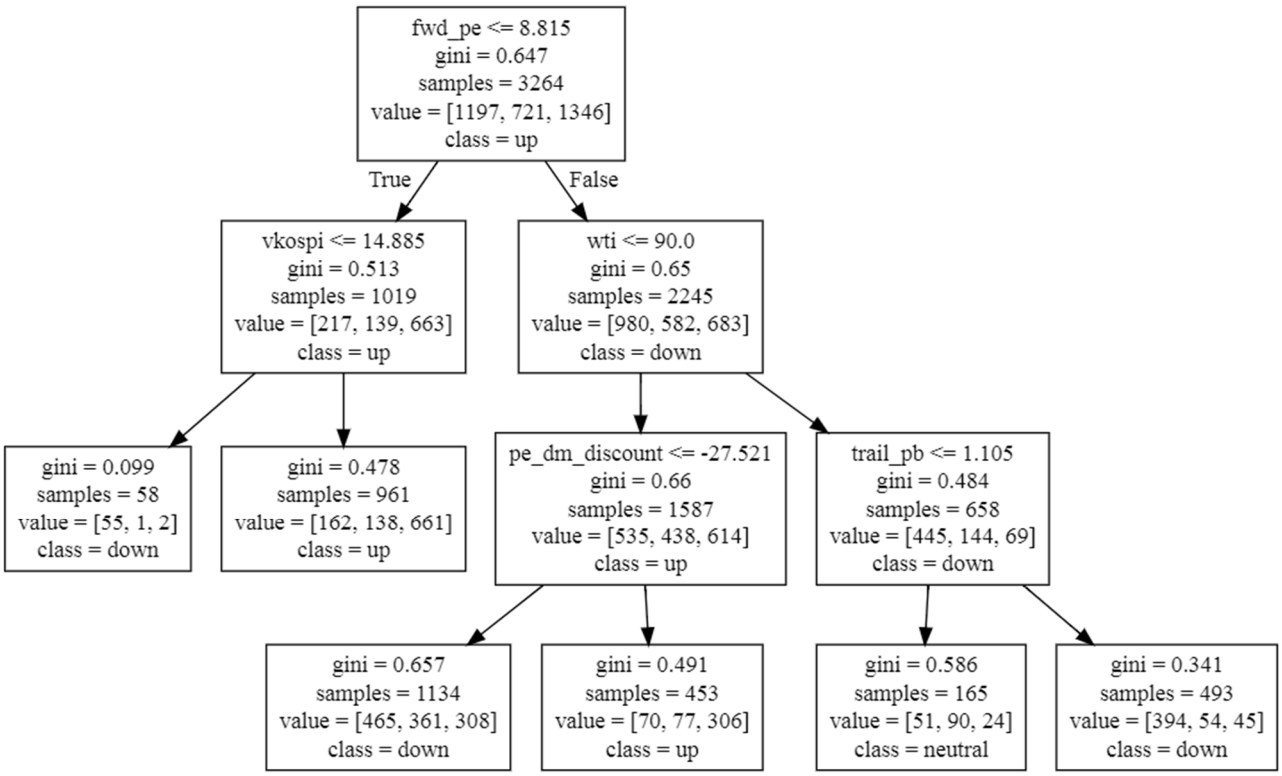
1. 결정 트리 모델

결정 트리 기법은 오래된 머신러닝 알고리즘이지만, 강력한 머신러닝 기법인 '랜덤 포레스트'의 기초 알고리즘이기 때문에 알아야 할 필요가 있다.

결정 트리(Decision Tree, 의사결정나무라고도 함)는, 나무 형태의 판단 규칙 조합을 만들어서 기존 데이터를 학습하고 그 다음에 예측하는 알고리즘을 말한다. 쉽게 생각하면 어렸을 적의 "스무고개" 놀이와 같은 개념이다.

결정 트리 알고리즘을 시각화한 예로 다음 그림을 보자.

결정 트리 예제: 코스피 단기 방향성 예측하기



자료: 삼성증권

위 예제는 학습이 완료된 결정 트리 모델이다. 이 샘플 모델의 역할은 여러 가지 시장 지표 데이터를 가지고, KOSPI 단기 방향성을 예측하는 것이다. 과거 20년간의 시장 지표 데이터들과 해당 시점 사후의 KOSPI 방향성 정보를 모델에다 대입하면, 학습 결과로 위와 같은 판단 구조가 생성된다.

샘플 모델의 예측 과정은 다음과 같다. 현시점의 방향성을 예측할 때, 맨 위의 시작 노드에서 출발하여 "Fwd P/E가 8.815배 이하인가?" 라는 질문을 체크한다. 현재 P/E가 8.815배 이하면 true에 해당하므로, 바로 아래 왼쪽 노드로 이동한다. 아래 왼쪽 노드에서는 "vkospi <= 14.885"의 질문을 체크한다. 여기서 만약 현재 VKOSPI 지수가 14.885보다 높으면, false인 세 번째 줄의 왼쪽 2번째 노드로 이동한다. 여기가 마지막 leaf node가 되고, "class=up" 즉, 시장 단기 방향성이 상승이라는 결론으로 예측이 끝난다. '스무고개' 놀이를 통해 사물을 맞추는 것과 같은 방식이다. 이렇게 결정 트리는 여러 가지 데이터를 가지고 출발하여, 수치 분기점을 통과하면서 하나의 결론에 도달하는 판단 구조를 가지고 있다.

2. 훈련 알고리즘

위의 트리는 샘플수가 총 3,264개인 학습 데이터(up=1,346개, neutral=721개, down=1,197개의 결과 데이터, 그때 당시의 시장지표 데이터들로 구성)를 가지고 모델을 학습하여 트리 구조의 판단 로직이 만들어진 것이다.

알고리즘에서 중요한 것은, 이런 트리 구조를 어떻게 만드느냐 하는 점이다.

결정 트리의 훈련 알고리즘은, 데이터셋의 '불순도'를 가장 크게 낮추는 방향으로 하나의 특성(변수)과 그의 임곗값을 순차적으로 찾아가는 방식이다. 쉽게 말하면, 무질서하게 섞여있는 결과 데이터들이 각 클래스별로 최대한 잘 분리되도록 분리 기준을 찾는 방법을 쓰는 것이다.

불순도(impurity)라 함은 데이터가 무질서하게 섞여있는 정도를 의미한다. 이진 분류 작업을 하려고 할 때 TRUE, FALSE 클래스가 섞여있거나, 위의 예제처럼 학습 데이터에서 up, neutral, down 결과값이 많이 섞여있으면 불순도가 높다라고 한다. 이런 불순도를 나타내는 지표는 여러 가지가 있는데, 대표적인 것이 지니 불순도다. 지니 불순도의 수식은 다음과 같다.

$$\text{지니 불순도 } G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

여기서, i는 i번째 노드, n은 결과값 클래스의 개수, k는 결과값의 k번째 클래스,

$p_{i,k}$는 i번째 노드에 있는 훈련 샘플 중 클래스 k에 속한 샘플의 비율임

지니 불순도는, 경제에서 이야기하는 소득불평등지수인 지니계수와 같은 뿌리다. 지니계수는 완전균등선과 로렌츠곡선이 이루는 면적의 비율로 0~1 사이의 값이다. 지니계수가 0이면 완전평등, 1이면 극단적 불평등을 의미한다. 지니 불순도도 0~1의 값을 가질 수 있다(클래스 개수에 따라 다름). 0은 가장 불순도가 낮은 균질적인 상황을 의미한다. 수치가 커질수록 불순도가 높고 섞여 있다는 뜻이다.

예를 들어, 결과 데이터의 클래스가 TRUE, FALSE만 있다고 하자. 첫 번째 데이터셋에 TRUE만 4개가 있다면 지니 불순도는 0이다($1 - ((4/4)^2 + (0/4)^2) = 0$). 두 번째 데이터셋에 TRUE가 3개 있고 FALE가 1개 있으면 지닌 불순도는 0.375로 커진다($1 - ((3/4)^2 + (1/4)^2) = 0.375$). 두 번째 데이터셋에서는 깔끔하게 한 클래스만 모여있는 것이 아니고 TRUE, FALSE가 뒤섞여 있으므로, 불순도가 커진 것이다.

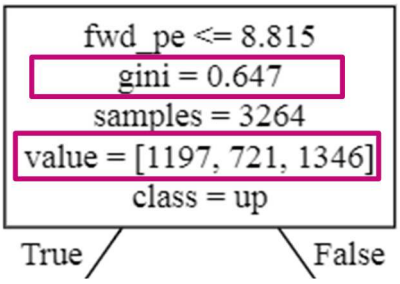
그래프 예제를 보자. 시작 노드를 보면, 총 3,264개 샘플에서 up이 1,346개, neutral이 721개, down이 1,197개로 서로 섞여 있다(그래프에서 value=[1197, 721, 1346]으로 표시. down/neutral/up 순서). 이에 따른 해당 노드에서의 불순도는 0.647로 나온다(그래프에서 gini=0.647로 표시되어 있음). 결과값들이 무질서하게 뭉쳐있다고 볼 수 있다.

$$1-\left(\left(\frac{1346}{3264}\right)^2+\left(\frac{721}{3264}\right)^2+\left(\frac{1197}{3264}\right)^2\right)=0.647$$

이 노드를 "fwd_pe<=8.815"란 기준으로 나누면, 1,019개 샘플과 2,245개 샘플을 가진 각각의 분할 데이터셋이 생긴다(그래프에서 2번째 줄의 노드). 2개 노드 각각의 불순도는 0.513, 0.65로 계산된다(그래프에서 수치 확인 가능). 분할 후의 전체 '종합' 불순도는 0.607로 계산된다.

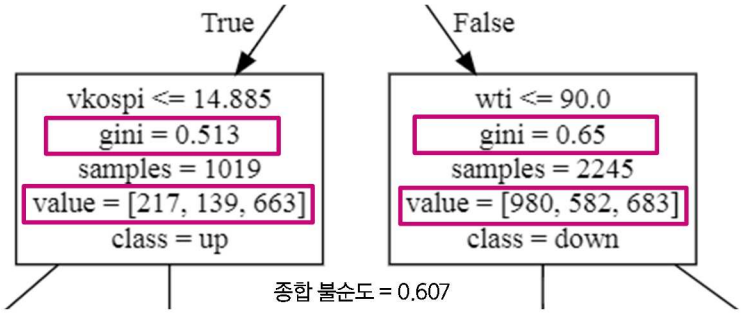
분할 후 '종합' 불순도 = $\left(\frac{1019}{3264}\right)\times 0.513+\left(\frac{2245}{3264}\right)\times 0.65=0.607$ (서브셋 샘플수로 가중평균)

시작 상태: 불순도 0.647



참고: 해당 노드(데이터셋)에 up/neutral/down 결과가 섞여있는 상황
자료: 삼성증권

첫번 째 분할 후: 불순도 0.607 → 자료가 더 잘 나눠졌다는 뜻



참고: 노드 분리 후, 왼쪽 노드에는 up결과가 많이 들어 있고(663개), 오른쪽 노드에는 down 결과가 많이 들어 있음(980개). 분할을 통해 각 클래스별로 분리됨
자료: 삼성증권

예제에서, 분할 전의 불순도 0.647는 분할 후의 불순도 0.607로 하락, 개선되었다. 왜일까? 첫 노드에서는 up, neutral, down이 모두 뭉쳐 있었지만, 분리된 노드에서는 구별이 더 잘 되었기 때문이다. 두 번째 줄 왼쪽 노드에는 up이 상대적으로 많이 모였고, 두 번째 줄 오른쪽 노드에는 down이 상대적으로 모였다. 이에 따라, 분할 후의 종합 불순도는 하락했다. 즉 "fwd_per<=8.815"란 기준은 up, neutral, down이 뭉쳐진 샘플들을 처음보다 조금 더 분리했음을 알 수 있다.

결정 트리는, 이렇게 불순도를 가장 최대한 낮출 수 있도록(자료 분류가 잘 되도록) 모든 특성(변수)과 모든 임곗값을 다 대입해서, 제일 좋은 기준을 하나씩 찾아가면서 트리 구조를 만들어낸다. 이 트리를 가지고, 향후 신규 데이터에 대해서 트리 형태 판단을 통해 분류(예측)를 하게 된다.

한편, 결정 트리 기법의 하이퍼파라미터[1]로는 criterion, max_depth, max_leaf_nodes 등등이 존재한다. criterion은 불순도를 계산하는 함수를 의미하며, 여기서 설명한 '지니 불순도'나 '엔트로피' 등이 사용된다. max_depth는 트리의 최대 깊이를 정하는 기준값이다. max_leaf_nodes는 끝단 노드(leaf node)의 최대 개수를 제한하는 역할을 한다.

[1] 모델 외적인 조정변수. 쉽게 말해 모델 구동을 위한 세팅 기준을 의미함. 예를 들어 선형회귀분석에서는 비용 함수로, 일반적인 RMSE(root mean square error, 평균 제곱근 오차)을 선택하거나 mean absolute error를 선택하는 것이 하이퍼파라미터에 해당함. 또한 다항 회귀 분석에서는 몇 차항까지를 사용할지를 결정하는 것이 하이퍼파라미터에 해당함

3. Python/sklearn에서의 사용법

(Python에 대한 기초 지식 보유를 가정함)

머신러닝 모델을 실제 구축할 때, 보통 Python 프로그래밍 언어를 가장 많이 사용한다. 특히 싸이킷런(scikit-learn) 라이브러리가 머신러닝에 관한 핵심적인 라이브러리다.

scikit-learn 라이브러리에서, 머신러닝 모델을 사용하는 방식은 보통 다음의 형태로 통일되어 있다.

```
import sklearn
dummy_clf = DummyClassifier() # 머신러닝 모델 객체 선언
dummy_clf.fit(X_train, y_train) # X_train 피처들과 y_train 레이블들을 가지고 학습
y_new = dummy_clf.predict(X_new) # X_new 피처를 가지고 모델에서 예측하기
```

sklearn의 지도학습 모델 객체들은 대부분 fit, predict, predict_proba 등의 메서드를 가진다.

fit 메서드는 학습 데이터의 판단근거 자료(입력값)와 분류결과 자료(출력값)를 input으로 받아서 모델 학습을 진행하는 메서드다. 머신러닝에서 판단근거 자료들은 피처(feature)라고 주로 칭하며, 회귀분석에서의 독립변수와 똑같은 역할을 한다. 분류결과 자료는 레이블(label)이라고 주로 칭하며, 회귀분석에서의 종속변수와 똑같은 역할을 한다. sklearn에서 보통 피처(feature) 자료를 X 변수로 많이 표현하며, 레이블(label) 자료를 y 변수로 많이 표현한다. 예를 들어, 피처가 m개인 훈련 데이터가 샘플이 n개 있다고 하자. 이 때, X 데이터는 (n×m)의 배열, y 데이터는 (n×1)의 배열로 표현이 된다.

predict 메서드는 새로운 X 데이터를 input 받아서 모델을 예측하는 일을 한다. predict_proba 메서드는, 모델 예측을 통해 분류작업에 의한 각 클래스가 될 확률을 자세히 보여주는 역할을 한다(결론만을 보여주는 predict와 비교했을 때, 판단 직전의 중간 단계 정보를 보여주는 역할임).

결정 트리 모델의 실제 구현 코드 샘플을 다음과 같이 만들었다.

결정 트리 모델 코드: TREE.py

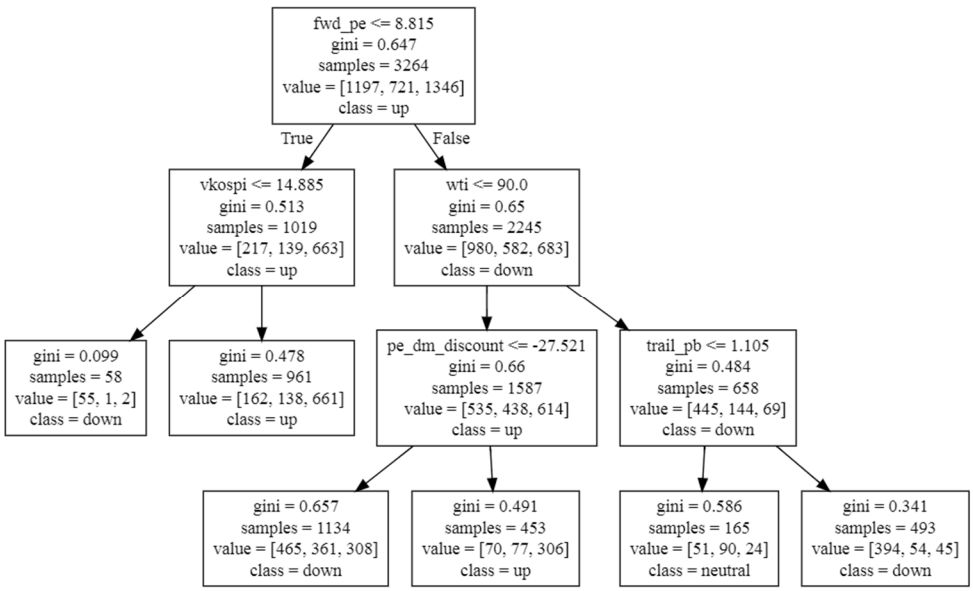
```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedShuffleSplit

# 1. daily raw data 가져오기
model_data = pd.read_excel("daily_data4_value.xlsx", sheet_name="raw", header=18, index_col=0)

# 2. features, label 전체데이터 생성
X = model_data.iloc[:, 1:]
X_names = X.columns
y = model_data["forward_stage"]
X = X[y.notna()]
```

```
y = y[y.notna()]

# 3. train, test 나누기
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=0)

for train_index, test_index in sss.split(X, y):  #n_splits 수만큼 반복
   X_train, X_test = X.iloc[train_index,], X.iloc[test_index,]
   y_train, y_test = y[train_index], y[test_index]

# 4. 결정 트리 샘플
print("\n", "decision tree learning...")
tree_clf = DecisionTreeClassifier(max_leaf_nodes=6) #max_leaf_nodes는 끝단의 leaf_node 수를 제한하는 것임
tree_clf.fit(X_train, y_train)
print("accuracy_score of test data", tree_clf.score(X_train, y_train))

# 5. Visualization
y_names = tree_clf.classes_
export_graphviz(tree_clf, out_file="CART_sample.dot", feature_names=X_names, class_names=y_names)
```

결정 트리 모델 결과는, export_graphviz 함수를 통해서 트리 구조를 쉽게 시각화할 수 있다. 코드 마지막 줄은 tree_clf 모델의 현재 학습된 트리구조를 'CART_sample.dot' 파일로 저장하라는 뜻이다. dot 파일은 graphviz 프로그램에서 사용하는 그래프 생성용 텍스트 파일이다. 인터넷에서 graphviz.msi 파일을 받아 설치한 다음, gvedit 프로그램에서 해당 dot 파일을 열면, 처음의 예제와 같은 트리 구조 그래프를 만들어낼 수 있다.

(Python 프로그램 전반의 코드 설명은 랜덤 포레스트 챕터에서 진행함)

코드 실행 결과: CART_sample.dot의 비주얼 그래프



자료: 삼성증권

한편 결정 트리의 특징은, 충분한 깊이의 트리가 주어진다면 항상 training data에 100% 정확히 일치하는 모델을 만들 수 있다는 점이다. 사실, 이는 과적합되었을 가능성이 높다는 뜻이며, out-of-sample에서 실패할 가능성이 높아진다는 뜻이다. 이 특징은 결정 트리의 대표적인 단점에 해당한다.

실질적으로 결정 트리는 단독으로는 잘 사용되지 않고 있으며, 랜덤 포레스트 모델의 하위 구성 요소로 대부분 사용되고 있는 상황이다.

Contents

III. 랜덤 포레스트 기법

랜덤 포레스트를 위해서는 먼저 앙상블 기법에 대한 이해가 필요하다.

앙상블 기법

'조화', '함께'의 뜻을 가진 앙상블(ensemble)은, 머신러닝 분야에서는 '여러 모델들을 활용해 종합하여 예측을 하는 기법'을 말한다. 적당한 예측력(성능)을 가진 모델을 한 개만 활용하는 것보다 여러 모델을 가지고 종합해서 결정하면 더 나은 의사 결정을 할 수 있다는 개념이다.

앙상블 기법의 종류로는 배깅, 페이스팅, 부스팅 등을 들 수 있다.

배깅(bagging)은 bootstrap aggregating의 줄임말이며, 훈련 세트에서 중복을 허용하여 샘플링하는 방식이다. 쉽게 말하면, 하나의 훈련 데이터를 가지고 여러 개의 훈련 데이터를 만들고, 이를 가지고 여러 모델을 만들어서 모델들을 종합하는 방식이다. 배깅에서는 훈련 세트에서 중복 허용을 통해 여러 샘플링 세트를 만들고, 이 샘플링들을 가지고 여러 개의 예측기(모델)를 훈련한다. 그 다음 예측할 때 이들 모델 예측의 평균을 계산해서 최종 예측을 하는 방식이다.

페이스팅(pasting)은 배깅과 유사한 방식인데, 훈련 세트에서 중복을 허용하지 않고 샘플링하는 방식이다. 중복 없이 여러 샘플링을 만들고, 이를 가지고 여러 모델을 훈련한 다음, 이들 모델 예측의 평균을 가지고 최종 예측하는 방식이다.

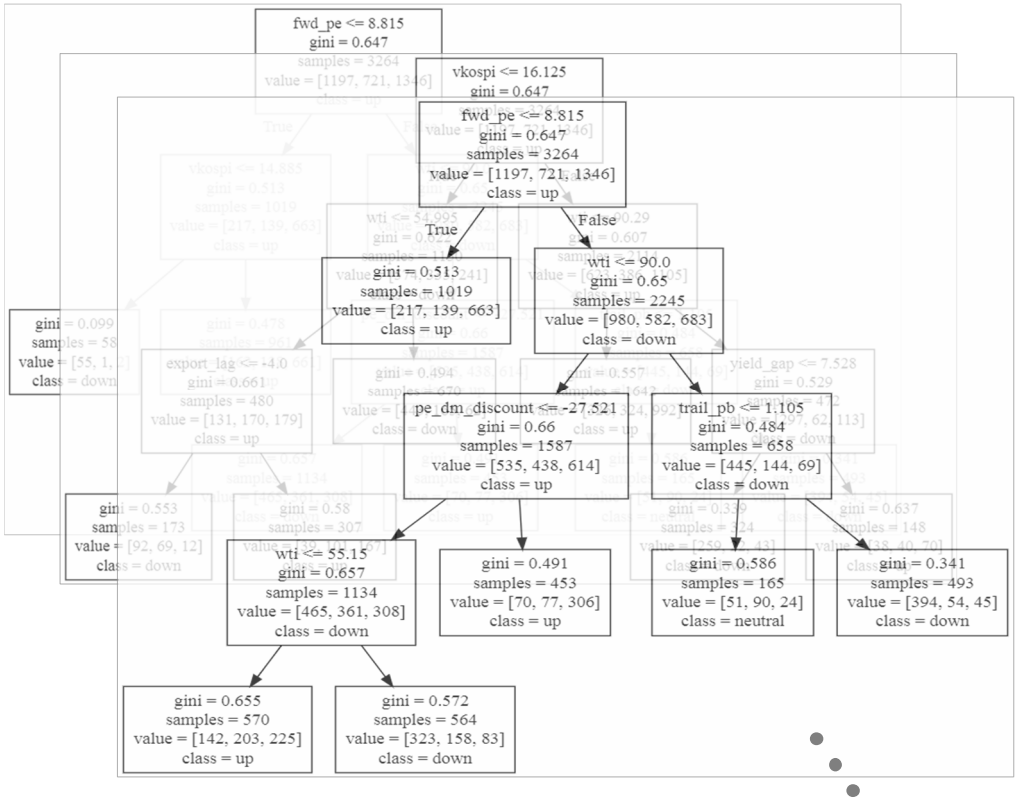
부스팅은 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 방식이다. 부스팅 기법 중에는 아다부스트(AdaBoost, Adaptive Boosting의 줄임말)가 대표적인 기법이다. 아다부스트 기법은 하나의 분류기를 훈련 세트에서 훈련시킨 다음 결과를 보고, 잘못 분류된 훈련 샘플의 가중치를 높인 다음 그 데이터를 그 다음의 분류기에 넣는 방식이다. 아다부스트는 순차적으로 분류기(모델)를 학습하는데 그 때 각 오류 샘플의 가중치를 높여서 넣어서, 결과적으로는 종합 모델의 정확도가 올라가도록 유도하게 된다.

랜덤 포레스트

앞서 언급한 '결정 트리' 모델에 배깅(bagging) 기법을 적용한 앙상블 모델을, 특별하게 '랜덤 포레스트(Random Forest)' 모델이라고 한다. 하나의 데이터 소스에서 랜덤하게 데이터들을 만들어 여러 개의 학습된 '결정 트리' 모델을 만들고 이를 종합하는 방식이다. 보통 랜덤 포레스트는 서로 다른 수십 개 혹은 수백 개의 결정 트리를 만들어서 사용한다. 트리를 여러 개 만들어서 숲을 만든다는 뜻이므로, 직관적인 작명법이라 하겠다.

앙상블 모델의 하나의 사례에 랜덤 포레스트라는 특별한 이름이 붙은 것은, 해당 기법이 상당히 예측력이 높으며 실무에서 많이 사용되는 중요한 모델이기 때문이다.

랜덤 포레스트 개념도



참고: 랜덤 포레스트는 수십 혹은 수백 개의 '결정 트리' 모델을 만든 다음, 이를 종합해서 판단함
자료: 삼성증권

랜덤 포레스트에서는 배깅 기법을 통해, 우선 훈련 데이터에서 중복을 허용한 여러 데이터 샘플을 만들어낸다. 이 때, 분산성을 더 키우기 위해 피처(독립변수)의 개수를 랜덤하게 줄인 데이터를 각각의 결정 트리 모델에 사용하는 기법도 적용하고 있다. 랜덤 포레스트의 가장 큰 장점은 앙상블 기법을 통해서 overfitting 문제를 감소시키고 또한 예측력이 올라간다는 것이다. 랜덤 포레스트는, 전기간 데이터만 가지고 학습하는 것이 아니라 몇몇 시계열 구간과 몇몇 주요 변수들을 뺀 샘플들로 수백 개의 결정 트리를 만들어 종합하는 방식이다. 이를 통해 일부 사건에 overfitting되지 않은 범용적인 판단 모델을 더 잘 만들게 된다.

랜덤 포레스트의 단점에는, 여러 개의 결정 트리 모델을 겹쳐서 사용하므로 전체적인 구조를 직관적으로 이해하기 힘든 '블랙 박스 유형'의 알고리즘이란 점을 들 수 있다.

Python/sklearn에서의 사용법

Python 언어, sklearn 라이브러리 환경에서의 랜덤 포레스트 모델 사용법은, 다른 머신러닝 모델 사용법과 동일하다.

```
from sklearn.ensemble import RandomForestClassifier

# 랜덤 포레스트 모델 선언
rnd_clf = RandomForestClassifier(n_estimators=100, max_leaf_nodes=40, max_features=8, n_jobs=-1)

# X_train 피처들과 y_train 레이블들을 가지고 학습
rnd_clf.fit(X_train, y_train)

# X_new 피처를 가지고 모델에서 예측하기
y_current_pred = rnd_clf.predict(X_new)
```

랜덤 포레스트 모델의 객체는 RandomForestClassifier다(sklearn.ensemble.RandomForestClassifier). 랜덤 포레스트에서도 fit 메서드를 통해서 모델 학습이 이뤄진다. fit 메서드에서는 독립변수 집합과 같은 성격인 'X_train' 피처들과 종속 변수 혹은 분류 결과를 의미하는 'y_train' 레이블을 입력받아서 학습을 진행한다. 새로운 피처를 가지고 예측할 때는 predict 메서드를 사용한다.

랜덤 포레스트의 하이퍼파라미터로는 모델 자체 파라미터인 n_estimators, max_features 등과 결정 트리의 파라미터인 criterion, max_depth, max_leaf_nodes 등을 모두 가지고 있다. 자체 파라미터인 n_estimators는 랜덤 포레스트에서 하위 결정 트리 몇 개를 생성할지 지정하는 옵션이다. max_features는 랜덤 포레스트에서 각각의 결정 트리를 만들 때 훈련 데이터에 사용되는 피처 개수의 상한을 지정하는 옵션이다. 여기서 언급한 파라미터들이 랜덤 포레스트 사용 시의 핵심적인 파라미터에 해당한다고 하겠다.

분류 성능 측정 지표: 정확도(accuracy), 정밀도(precision), 재현율(recall), F1 점수(F1 score)

분류 모델 상에서 '얼마나 잘 분류하는지'에 대한 성능 측정 지표로 가장 일반적인 것이 '정확도(accuracy)' 지표다. 본 리포트에서도 성능 지표로 정확도 지표를 기본으로 사용했다. 여기서는 정확도 지표와 함께, 다양한 성능 지표 개념인 혼동행렬, 정확도, 정밀도, 재현율, F1 점수에 대해서 같이 설명한다.

우선, 분류기의 성능을 행렬 형태로 표시한 것을 혼동행렬(confusion matrix)라고 한다. 이는 분류 모델에 의한 예측과 결과 조합의 개수를 행렬로 나타낸 것이다.

예를 들어 고양이 사진과 강아지 사진을 놓고 어떤 동물인지 판단하는 모델이 있다고 하자. 13장의 사진에 대해서 모델이 예측했던 동물과 실제 동물의 조합 개수를 아래와 같이 표의 수치로 나타낼 수 있다.

도표 1

		Actual class	
		Cat	Dog
Predicted class	Cat	5	2
	Dog	3	3

참고: 고양이로 예측했는데 실제 고양이였던 경우가 5번, 고양이로 예측했는데 실제 강아지였던 경우가 2번이라는 뜻임.
자료: Wikipedia

이런 표를 혼동행렬이라고 한다. 여기서 분류 질문을 "동물이 고양이인가?"로 살짝 바꾸면, 답이 Positive 혹은 Negative로 결정되는 일반적인 2항 분류 문제가 된다(아래 도표).

도표 2: 혼동행렬

		Actual class	
		Cat	Non-cat
Predicted class	Cat	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	3 True Negatives

주석: 여기서의 가로, 세로 배열은 Wikipedia의 설명 기준임. Python sklearn 라이브러리에서는 혼동행렬의 축이 반대로 되어 있음. Sklearn 라이브러리에서는 위쪽 column 기준이 예측 클래스이며, 왼쪽 row 기준이 실제 클래스임
자료: Wikipedia

일반적인 2항 분류 문제에서 예측치와 결과치의 조합은 TP, FP, FN, TN로 구분된다.

TP(True Positive): Positive로 예상했는데 실제 Positive여서 맞춘 경우
FP(False Positive): Positive로 예상했는데 실제는 Negative여서 틀린 경우 (Type I 에러)
TN(True Negative): Negative로 예상했는데 실제 Negative여서 맞춘 경우
FN(False Negative): Negative로 예상했는데 실제는 Positive여서 틀린 경우 (Type II 에러)

주석: 단어 본체가 항상 예측치 기준임

정확도(Accuracy)는 $ACC = (TP + TN)/(P + N) = (TP + TN)/(TP + TN + FP + FN)$다. 즉, Positive를 Positive로 정확히 맞힌 개수와 Negative를 Negative로 정확히 맞힌 개수의 합을 전체 샘플수로 나눈 값이다. 전체 샘플 중에서 정확하게 맞힌 개수의 비율을 뜻하며, 가장 기본적인 성능지표다.

2항 분류가 아닌 n-항 분류에서도 정확도 지표는 계산될 수 있다. 전체 샘플수 대비한 각 클래스를 적중한 케이스의 샘플수 비중으로 정확도(Accuracy) 지표를 계산한다.

정밀도(Precision)는 $PPV = TP/(TP + FP)$로, 모델이 Positive라고 예측한 것 전체 중에서 실제로 Positive를 맞힌 비율을 말한다. 자기(모델)가 한 말 중에 얼마나 정확하게 맞혔는지를 보여주는 지표라고 하겠다.

재현율(Recall)은 $TPR = TP/(TP + FN)$로, 실제로 Positive인 사실들 중에서 모델이 Positive라고 맞힌 비율을 말한다. 실제 현실에 대한 예측을 할 때, 얼마나 잘 재현했는지 혹은 얼마나 현실을 잘 반영했는지를 보여주는 지표라고 하겠다.

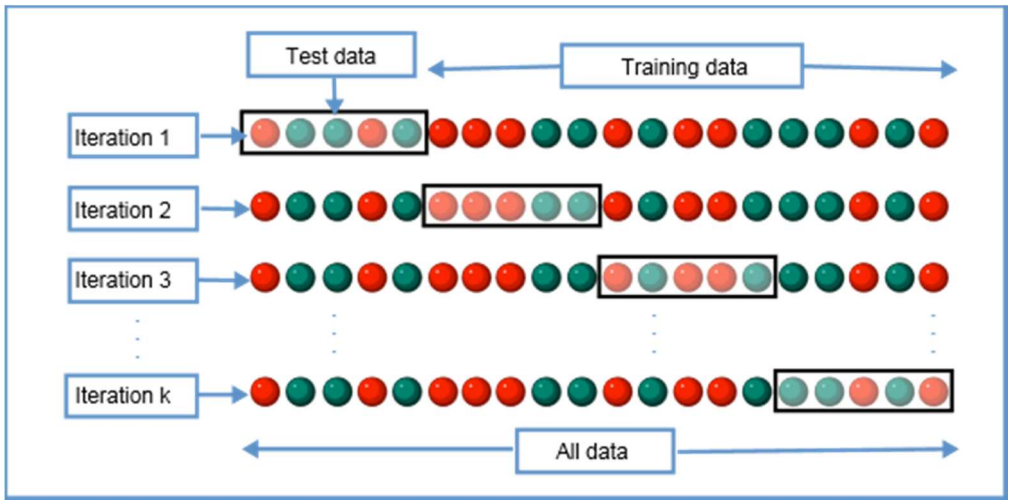
이와 관련해서, 모델이 아주 확실한 경우에만 Positive를 가끔 말한다면, 정밀도는 올라가고 재현율은 내려간다. 모델이 약간의 가능성만 있어도 Positive를 자주 말한다면, 정밀도는 내려가고 재현율은 올라간다. 따라서, 정밀도와 재현율은 서로 trade-off 관계인 것을 알 수 있다.

F1 점수(F1 score)는 $F_1 = 2/(\frac{1}{\text{정밀도}} + \frac{1}{\text{재현율}})$이어서, 정밀도와 재현율의 조화평균값이다. 둘을 종합해서 보는 지표다.

교차 검증(cross-validation) 기법

앞서 과적합(Overfitting) 이슈에서 설명한 교차 검증(cross-validation)에 대해 부연 설명하면 다음과 같다. 머신러닝에서 검증(validation)이란 여러 모델 중에서 최선을 선택하거나 현실적인 예측력을 확인하기 위해, 데이터를 훈련 데이터와 검증 데이터로 나눠서, 검증 데이터로 확인하는 과정을 말한다. 이 때 별도로 검증 데이터를 분리할 경우 샘플수가 줄어드는 문제가 생기기 때문에, 교차 검증 기법을 주로 많이 쓴다.

K-fold cross-validation 설명



자료: Wikipedia

K겹 교차 검증 기법에서는 우선 전체 데이터를 k개로 나눈다. 그리고, 첫 번째 시도에서는 2번~k번의 부분집합을 훈련 데이터로 쓰고, 1번 부분집합을 검증 데이터로 사용해 결과를 확인한다. 두 번째 시도에서는 1번과 3~k번의 부분집합을 훈련 데이터로 쓰고, 2번 부분집합을 검증 데이터로 사용한다. 이런 식으로 k번의 작업을 진행한다. 그리고, 그 결과를 모두 모아서 모델의 성능을 확인하는 구조다. 보통 k값으로는 10을 많이 쓴다. 이 기법은 전체 데이터 개수의 손실 없이 여러 번의 모델 성능 점검을 종합할 수 있기 때문에, 실무에서 자주 사용하고 있다.

```
from sklearn.model_selection import cross_val_score

dummy_scores = cross_val_score(dummy_clf, X_train, y_train, scoring="accuracy", cv=10)
```

파이썬/sklearn에서 교차 검증 기법은 cross_val_score 함수를 사용한다.

cross_val_score 함수의 input은 머신러닝 모델(dummy_clf), 피처 데이터 배열(X_train), 레이블 데이터 배열(y_train) 순으로 되어있다. cv 변수는 k개로 나누고 k번 반복하는 그 k값을 지정하는 변수다. scoring은 개별 머신러닝 모델에서 성능 평가기준으로 쓰는 스코어링 함수를 명시적으로 적는 부분이다. Accuracy 지표, RMSE 지표 등이 들어갈 수 있다.

cross_val_score의 리턴값에는, k번 작업에 따른 성능 평가 점수의 k개 배열이 들어간다. 위 코드에서는 dummy_scores에 저장된다. 보통 이 dummy_scores의 평균값을 모델의 검증단계 성능수치로 사용한다.

Contents

IV. 랜덤 포레스트 기반 단기예측 모델

필자는 랜덤 포레스트 기법을 사용하여 코스피 시장을 단기예측하는 모델을 개발했다. 여기서는 해당 모델의 개발 과정과 모델 예측 결과에 대해 상세히 설명할 계획이다.

AI 모델의 기본적인 아이디어는 이전에 나온 "코스피 단기예측 모델" 리포트(19/9/3)[2] 에서 출발했다. 단기예측 모델은 여러 가지 매크로 지표, 시장 투자 지표들을 활용해서 코스피의 향후 3개월 방향성을 예측하는 모델이다. 모델 예측의 결과는 '상승/보합/하락'의 3가지 투자의견으로 나타난다. KOSPI의 미래 3개월 수익률에 대해 4% 이상을 기대하면 '상승' 의견, 4%~0% 사이를 기대하면 '보합' 의견, 0% 미만을 기대하면 '하락' 의견을 제시하는 방식이다.

이 모델을 머신러닝 모델의 관점에서 보자. 과거 각 시점들에서의 여러 투자 지표들(예를 들어, G10 Economic Surprise Index나 수출 데이터 등)의 데이터가 실제 존재한다. 그리고, 그 시점 사후의 3개월간 KOSPI 실제 수익률 데이터도 존재한다. 3개월간 KOSPI 실제 수익률 데이터는 앞서의 4% / 0% 기준에 의해 '상승, 보합, 하락'이라는 3가지 단계로 변환될 수 있다.

그렇다면, 이 데이터들은 각종 투자지표들을 피처(독립변수)로 가지고, 상승/보합/하락이라는 단기 방향성 결과를 레이블(종속변수)로 가지는 지도 학습(supervised learning) 3항 분류 모델의 문제로 접근할 수 있다.

우선 예측 모델에 사용되는 피처 데이터 리스트는 다음과 같다.

피처 데이터 리스트

리스트	리스트
Citi G10 ESI	한국 P/E의 선진국 대비 할인율
Citi EM ESI	KOSPI Trailing P/B
원/달러 환율	FY2 이익조정비율 (1m)
한국 수출 증가율	FQ1 이익조정비율 (1m)
WTI 유가	FQ1 이익조정비율 (1w)
US Core PCE	투자의견 점수 변화율 (1w)
KOSPI Forward P/E	VKOSPI
Yield Gap	VIX
한국 P/E의 이머징 대비 할인율	Put-call volume ratio (20d)

참고: 기존 단기예측 모델의 데이터를 동일하게 사용. 다만, 일별 데이터 사용
자료: 삼성증권

[2] 리포트링크: http://nums.samsungpop.com/report/2019090210412521K_02_08.pdf

단기예측 AI 모델의 구성은 다음과 같다.

단기예측 AI 모델 구성

사용 알고리즘: 랜덤 포레스트 모델

시계열 자료 기간(샘플수): 2002/12/30부터 현재까지의 영업일별 시계열 자료. 약 4천 개.

피처(feature): 18종의 매크로, 밸류에이션, 어닝스 등의 투자지표 일별 수치

레이블(label): 해당일의 사후 KOSPI 방향성. 상승/보합/하락으로 지정됨. KOSPI의 사후 60영업일 수익률이 4% 이상이면 '상승', 4%~0% 사이면 '보합', 0% 미만이면 '하락' 결과로 기록.

(T일의 KOSPI 방향성은 'T+1일~T+61일 간의 사후 수익률'을 기준으로 삼음. 현실적인 투자 가능성을 위해 하루 뒤부터 투자하는 것으로 가정)

머신러닝 AI 모델에서는 자료 샘플을 일별 데이터를 사용함으로써 샘플수를 크게 증가시켰다.

머신러닝 기법의 핵심 경쟁력은 빅데이터, 즉 많은 데이터에서 나온다. Overfitting을 극복하기 위한 머신러닝의 많은 기법들이, 실제로는 input 데이터를 늘리면서 가능하게 된 기법들이다(cross-validation 등). 많은 데이터들을 확보하는 게 머신러닝에서 중요한 포인트며, 이번 AI 모델에서도 일별 데이터 확보를 통해 샘플수를 늘렸다.

피처와 레이블을 모두 담은 훈련 데이터셋은 다음의 형태로 관리된다.

훈련 데이터셋 예제

base_date	forward_stage	g10_esi	em_esi	krw	export_lag	wti	core_pce_lag	fwd_pe	yield_gap	pe_em_discount	pe_dm_discount	trail_pb	fy2_err_1m	fq1_err_1m	fq1_err_1w	recom_chg_1w	vkospi	vix	put_call_ratio
…																			
03-06-24	up	-7.9	2.9	1,189.8	3.5	28.8	0.08	7.1	9.9	-16.6	-52.6	0.91	3.5	0.4	-0.6	0.1	27.5	20.8	82.3
03-06-25	up	-3.1	2.7	1,187.1	3.5	30.0	0.08	7.3	9.7	-15.9	-51.7	0.93	3.5	0.4	-0.6	0.1	27.6	20.8	80.6
03-06-26	neutral	-12.6	3.2	1,186.6	3.5	29.0	0.08	7.3	9.5	-15.3	-51.5	0.93	3.5	0.4	-0.6	0.1	27.8	19.4	79.8
…																			

참고: 여기서 월간 통계지표인 export_lag, core_pce_lag에는 3일간 동일한 수치(최근 발표치)가 들어가 있는 것을 알 수 있음
자료: 삼성증권

각 일자별로(샘플수가 n개면 n개 행) 레이블 정보와 피처 정보가 각 열에 표시되어 있다. forward_stage는 레이블에 해당하며, 해당일의 사후 60영업일 수익률로 결정된 클래스다. 예를 들어 03/6/24의 forward_stage는, 다음 날인 03/6/26부터의 사후 60일간 수익률이 7.5%로 나와서 'up' 클래스로 지정되었다. 03/6/26의 forward_stage는 사후 60일간 수익률이 0~4% 사이여서 'neutral' 클래스로 지정되었다. 다음 g10_esi ~ put_call_ratio의 18개 열은 앞에서 정의한 투자지표 데이터가 동일한 순서대로 들어가는 피처(feature) 영역이다.

단기예측 AI 모델 전체 code

단기예측 AI 모델의 전체코드는 현재 1) 모델 학습 프로세스와 2) 예측 프로세스 두 부분으로 나누어져 있다.

모델 학습 프로세스

- Raw data 가져오기 (train, test data 나누기)
- 최적 하이퍼파라미터를 찾기 위한 모델 튜닝
- K-fold cross validation 실행으로 모델 성능 확인
- 최종 모델 학습
- test data로 최종 모델 성능 확인
- 모델 저장

예측 프로세스

- 모델 로드
- 현재 데이터로 예측하기

모델 학습 프로세스의 전체 코드는 다음과 같다.

모델 학습 프로세스 코드: MODEL_1_learning.py

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.metrics as mt
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.externals import joblib
from sklearn.metrics import confusion_matrix


# 1. daily raw data 가져오기
model_data = pd.read_excel("daily_data4_value.xlsx", sheet_name="raw", header=18, index_col=0)

# 2. features, label 전체데이터 생성
# X, y는 최근일까지 포함한 전 데이터. X_past, y_past는 결과가 확인된 61일전까지의 데이터
X = model_data.iloc[:, 1:]
X_names = X.columns
y = model_data["forward_stage"]

X_past = X[y.notna()]
y_past = y[y.notna()]


# 3. train, test 나누기
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
```

```
for train_index, test_index in sss.split(X_past, y_past):  #sss.split(~) 안에 n_splits 수만큼 준비됨
   X_train, X_test = X_past.iloc[train_index,], X_past.iloc[test_index,]
   y_train, y_test = y_past[train_index], y_past[test_index]


# ===== 랜덤포레스트 메인 =====


# 4. 모델 세부 튜닝: 최적 하이퍼파라미터 찾기
#rnd_clf = RandomForestClassifier(n_estimators=100, n_jobs=-1, random_state=42)
#
#param_dist_rf = {
#   'n_estimators':[50, 100, 500],
#   'max_leaf_nodes':[20, 30, 40, 50],
#   'max_features':[2, 4, 6, 8]
#}
#
#rnd_search = RandomizedSearchCV(rnd_clf, param_dist_rf, cv=10, random_state=42)
#rnd_search.fit(X_train, y_train)
#print(rnd_search.best_params_)


# 5. 학습 및 K-fold cross_validation 평가
rnd_clf = RandomForestClassifier(n_estimators=100, max_leaf_nodes=40, max_features=8, n_jobs=-1,
random_state=42)  #디폴트
rnd_scores = cross_val_score(rnd_clf, X_train, y_train, scoring="accuracy", cv=10)
print("\n<10-fold cross-validation>")
print("accuracy score mean: ", rnd_scores.mean())


# 6. 최종 모델 학습
rnd_clf.fit(X_train, y_train)
print("\n<AI model: machine learning done >")
print("accuracy_score of train data(0.8 of sample): ", rnd_clf.score(X_train, y_train))


# 7. test data 확인
print("accuracy_score of test data(0.2 of sample): ", rnd_clf.score(X_test, y_test))
#y_test_pred = rnd_clf.predict(X_test)
#print("accuracy_score of test data: ", mt.accuracy_score(y_test, y_test_pred))


# 8. confusion matrix 확인
y_test_pred = rnd_clf.predict(X_test)
cm1= confusion_matrix(y_test, y_test_pred, labels=["up","neutral","down"])
print("\n<Confusion matrix>")
print("(of test)")
print("up","neutral","down")
print(cm1)

cm2= confusion_matrix(y_past, rnd_clf.predict(X_past), labels=["up","neutral","down"])
print("(of all)")
print("up","neutral","down")
print(cm2)


# 9. 변수 중요도 체크
print("\n<Feature importance>")
for name, score in zip(X.columns, rnd_clf.feature_importances_):
   print(name, ": ", score)


# 10. backtesting용 과거의 예측데이터 생성
```

```
y_prediction = rnd_clf.predict(X)
y_pred = pd.Series(y_prediction, index=y.index)


# 11. 모델 저장
joblib.dump(rnd_clf, "forecast_model.pkl")
print("\n< AI model: save >")
```

선언부

머신러닝 라이브러리를 사용하기 위해, sklearn과 하위 class 등을 import하는 구문이다.

1. daily raw data 가져오기

앞의 '훈련 데이터셋 예제' 표에서 봤던 4,000여 개 행으로 된 데이터 엑셀파일(daily_data4_value.xlsx) 내용을 model_data 변수에 저장한다. 여기서 model_data는 pandas 라이브러리의 DataFrame 객체 변수다.

2. features, label 전체데이터 생성

모든 데이터가 들어있는 model_data에서 피처 데이터를 뽑아 X 변수에 저장하고, 레이블 데이터는 y 변수에 저장한다. X는 n×18 배열크기(shape)의 DataFrame 객체고, y는 n 배열크기의 Series 객체다. X_names는 X 데이터의 컬럼명(각 투자지표명)을 담고 있다. X, y는 최근일까지 포함한 전체 데이터다. 사후 방향성 정보가 없는 최근일 데이터(y 배열에서 공란으로 되어있는 부분)는 제외하고, 학습 가능한 데이터들을 X_past, y_past로 따로 저장한다.

3. train, test 나누기

Overfitting을 피하며 모델 성능을 확인하기 위해서, train data와 test data로 나누는 작업은 필수적이다. 이 때 StratifiedShuffleSplit 객체를 사용한다. 이 객체는 전체 데이터에서 train data와 test data를 골고루 분리하는 작업을 한다. 코드에서 test data의 사이즈는 전체의 0.2(20%)로 지정되었다. train_data를 가리키는 train_index 배열변수, test_data를 가리키는 test_index 배열변수를 만들고, 이를 통해 X_train, X_test, y_train, y_test 데이터를 각각 만든다.

4. 모델 세부 튜닝: 최적 하이퍼파라미터 찾기

랜덤 포레스트를 비롯한 많은 머신러닝 모델들은 여러 가지 하이퍼파라미터(모델 외적인 조정변수)를 가진다. 하이퍼파라미터의 설정에 따라서, 모델의 성능이 차이가 크게 날 수 있다. 따라서, 머신러닝 기법에서는 trial and error 방식으로 하이퍼파라미터를 조금씩 바꿔가면서 모델을 여러 번 테스트해, 최적의 파라미터 세팅을 찾는 기법을 많이 쓴다. 이 때 RandomizedSearchCV가 사용된다. RandomizedSearchCV는 최적 파라미터를 찾을 때 랜덤하게 적절한 수의 조합으로 테스트를 해주는 객체다.

코드에서 param_dist_rf 변수는 파라미터 세팅의 후보군이 들어가 있다. 여기서는 결정 트리 개수를 지정하는 n_estimators를 50번, 100번, 500번 중에서 선택하고, 피처 개수를 지정하는 max_features를 2, 4, 6, 8 중에서 고르는 등, 총 3*4*4=48개 조합 중에서 하이퍼파라미터를 고르도록 후보를 지정해놨다. rnd_search.fit을 통해 이를 확인한 결과가 rnd_search 객체에 저장되며, rnd_search._best_params_ 속성에서 그 결과를 확인한다. 실행 결과, 이번 랜덤 포레스트 모델의 최적 파라미터는 "n_estimators=100, max_leaf_nodes=40, max_features=8"로 확인되었다(뒤의 '코드 실행 결과' 내용에 표시).

5. 학습 및 K-fold cross_validation 평가

5번에는 K-fold cross_validation을 진행하여, 모델의 일반적인 예측력을 확인하는 단계다. 이 때 4번 코드에서 확인한 하이퍼파라미터 세팅이 사용되었다. cross_val_score가 교차 검증을 하는 함수이며, 이의 리턴값은 k번 검증한 성능지표 값의 배열이다. rnd_scores에 랜덤 포레스트를 10번 실행했을 때의 정확도(accuracy) 수치 배열이 들어가 있다. rnd_scores.mean()을 통해 이의 평균값을 확인한다.

6. 최종 모델 학습

여기서는 X_train과 y_train을 가지고 rnd_clf.fit을 통해서 최종적인 모델 학습을 진행한다.

7. test data 확인

test data를 통한 out-of-sample 테스트 과정이다. rnd_clf.score(X_test, y_test)를 사용하면, X_test와 y_test 데이터를 사용한 out-of-sample 테스트의 성능 수치를 확인하게 된다.

8. confusion matrix 확인

confusion_matrix 함수를 통해서 혼동행렬을 생성하는 코드다. sklearn 라이브러리는 Wikipedia와는 반대로, 행렬의 위쪽 column 기준이 예측 클래스이며, 왼쪽 row 기준이 결과 클래스이다.

9. 변수 중요도 체크

랜덤 포레스트는 블랙 박스 형태의 머신러닝 모델이다. 수십, 수백의 결정 트리 모델이 중첩되어 있기 때문에, 중간의 분류 작업 과정을 직관적으로 확인하기가 어렵다. 대신에, 최종 모델에서의 각 피처별 중요도 정보는 라이브러리에서 제공하므로 이를 확인할 수 있다. rnd_clf.feature_importances_는 각 피처들의 중요도 수치를 담고 있는 배열이다. 18개 지표를 피처로 쓰고 있으므로 결과도 18개 배열로 나오며, 각 피처들의 중요도를 모두 더하면 1이 된다. 변수의 중요도 수치가 클수록, 분류 판단에 중요하게 작용하는 변수라는 것을 의미한다.

10. backtesting용 예측데이터 생성

뒤에서 나올 AI모델 기반 투자 포트폴리오 백테스팅을 위한, 과거 예측치 생성 루틴이다.

11. 모델 저장

모델링한 상태를 파일로 저장하는 단계다. sklearn.externals.joblib을 사용하면 학습된 모델 정보를 pkl 파일로 쉽게 저장할 수 있다.

모델 학습 프로세스의 실행 결과는 다음과 같이 출력된다.

모델 학습 프로세스 코드 실행 결과

```
{'n_estimators': 100, 'max_leaf_nodes': 40, 'max_features': 8}

<10-fold cross-validation>
accuracy score mean: 0.8062374055627817

<AI model: machine learning done >
accuracy_score of train data(0.8 of sample): 0.8427615571776156
accuracy_score of test data(0.2 of sample): 0.81044957472661

<Confusion matrix>
(of test)
up neutral down
[[317  11  10]
 [ 52  89  41]
 [ 26  16 261]]
(of all)
up neutral down
[[1593   52   46]
 [ 227  504  176]
 [ 104   68 1341]]

<Feature importance>
g10_esi : 0.06775862397559076
em_esi : 0.049720132319309905
krw : 0.05245610408369035
export_lag : 0.06903322158373318
wti : 0.14177595378422583
core_pce_lag : 0.039037048164689093
fwd_pe : 0.12883474580452478
yield_gap : 0.04266947834950722
pe_em_discount : 0.04769161558448116
pe_dm_discount : 0.06439509460535199
trail_pb : 0.07115502560425827
fy2_err_1m : 0.022682389767492383
fq1_err_1m : 0.012277027718205295
fq1_err_1w : 0.008887778874414677
recom_chg_1w : 0.0042505330978163825
vkospi : 0.07770324450857719
vix : 0.051158622563615086
put_call_ratio : 0.04851335961051654

< AI model: save >
```

결과를 나눠서 설명하면 다음과 같다.

```
{'n_estimators': 100, 'max_leaf_nodes': 40, 'max_features': 8}
```

4번 모델 세부 튜닝 코드를 실제 실행했을 때 나타나는 결과다. 하이퍼파라미터 세팅에 대한 랜덤 검색을 실시한 결과, 다음 기준이 최적 옵션인 것으로 확인되었다. ['n_estimators': 100, 'max_leaf_nodes': 40, 'max_features': 8]

```
<10-fold cross-validation>
accuracy score mean: 0.8062374055627817
```

5번 코드 cross validation을 통해서, 모델 성능을 확인하고 출력한 결과다. Cross validation을 10번 실행했을 때의 모델의 정확도(accuracy) 지표가 평균 80.6%로 나왔다. 시장 방향성이 '상승'인지, '보합'인지, '하락'인지를 적중하는 비율이 전체에서 80.6%였다는 뜻이다.

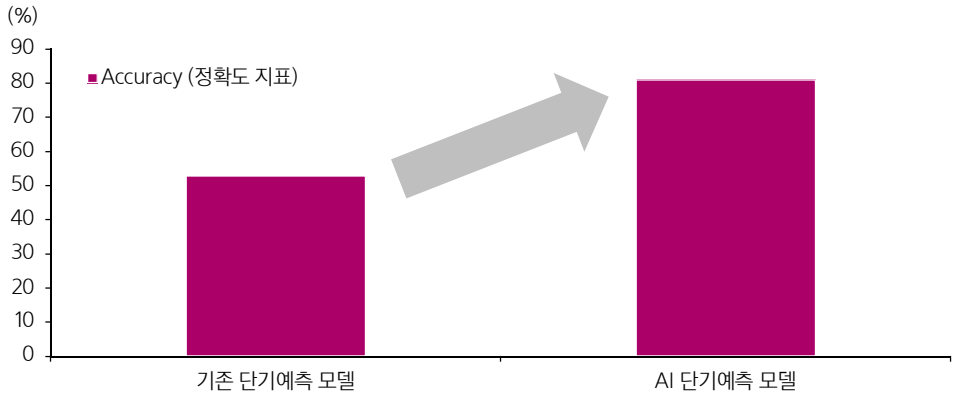
```
<AI model: machine learning done >
accuracy_score of train data(0.8 of sample): 0.8427615571776156
accuracy_score of test data(0.2 of sample): 0.81044957472661
```

6번, 7번 코드를 통해, 최종 모델 학습을 끝내고 train data 기준의 모델 성능과 test data 기준의 모델 성능을 확인했을 때의 결과다.

AI 모델에서 test data로 테스트한 결과, 정확도가 81.0%를 기록. 기존 모델 대비 예측력이 크게 상향되었음.

모델 학습에 사용되지 않은 신규 test data를 넣었을 때, 모델의 정확도 지표는 81.0%를 기록했다(전체 중에서 상승/보합/하락을 정확히 맞춘 비율). 이는 이전 리포트의 "코스피 단기예측 모델"이 52.8%의 정확도를 기록한 것에 비해, 크게 개선된 결과다(2019/9/3 "코스피 단기예측 모델" 리포트 참고). 더 정교한 로직을 사용하여, 과적합 이슈를 낮추면서도 정확도를 더 높인 모델이 만들어졌다는 것을 알 수 있다.

코스피 단기예측 AI 모델의 정확도 비교



자료: 삼성증권

```
<Confusion matrix>
(of test)
up neutral down
[[317  11  10]
 [ 52  89  41]
 [ 26  16 261]]
(of all)
up neutral down
[[1593   52   46]
 [ 227  504  176]
 [ 104   68 1341]]
```

8번 코드를 통해서 confusion matrix를 확인한 결과다. test data를 기준으로 한 혼동행렬이 먼저 출력되었다. 이 혼동행렬의 정보를 통해서, 정확도(accuracy) 수치가 [(317+89+261)/823 = 0.8104]로 계산되었다. 참고로, train data와 test data를 모두 포함한 전체 데이터에 대해서도 혼동행렬을 계산한 결과가 그 다음에 나온다. 참고로 전체 데이터 기준의 정확도(accuracy)는 83.6%로 확인된다.

```
<Feature importance>
g10_esi : 0.06775862397559076
em_esi : 0.049720132319309905
krw : 0.05245610408369035
…
put_call_ratio : 0.04851335961051654
```

rnd_clf.feature_importances_ 정보를 프린트하여, 랜덤 포레스트 모델에서 중요하게 사용된 key 지표들을 확인한 결과다. WTI가 0.141, Fwd P/E가 0.128로 단일 변수로는 가장 상위의 중요도를 기록했다. 그 외에 VKOSPI, 수출증가율, Trailing P/B 등이 0.06~0.08의 높은 중요도를 기록했다. 전체적으로 매크로 지표들과 밸류에이션 지표들의 중요도가 높은 것으로 확인된다.

다음으로, 모델 학습 및 세팅이 끝난 다음에 일별로 실행하는 예측 프로세스를 살펴보자. 예측 프로세스의 전체 코드는 다음과 같다.

예측 프로세스 코드: MODEL_2_routine.py

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.metrics as mt
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.externals import joblib
from sklearn.metrics import confusion_matrix

t = {'temp':'temp'}

# 1. 모델 로드
rnd_clf = joblib.load("forecast_model.pkl")
print("\n< AI model: load >")


# 2. new daily raw data 가져오기
# X, y는 최근일까지 포함한 전 데이터. X_past, y_past는 결과가 확인된 61일전까지의 데이터
model_data = pd.read_excel("daily_data4_value.xlsx", sheet_name="raw", header=18, index_col=0)
X = model_data.iloc[:, 1:]
X_names = X.columns
y = model_data["forward_stage"]

X_past = X[y.notna()]
y_past = y[y.notna()]


# 3. new daily raw data 전체 학습
rnd_clf.fit(X_past, y_past)
print("\n< AI model: machine learning done >")
print("accuracy_score of whole data: ", rnd_clf.score(X_past, y_past))


# 4. 현재(마지막) 데이터 표시
print("\n<Current status>")
```

```
for t['col'], t['score'] in zip(X.columns, X.iloc[-1]):
   print("{:20} : {:>8.3f}".format(t['col'], t['score']))
X_current = np.array(X.iloc[-1]).reshape(1,-1)


# 5. 현재 전망
print("\n< AI model: forecasting >")
y_current_pred = rnd_clf.predict(X_current)
print("forecast: ", y_current_pred)

# 현재전망의 확률표
prob_current = rnd_clf.predict_proba(X_current)
y_names = rnd_clf.classes_

print("\n[class] : [prob]")
for t['name'], t['prob'] in zip(y_names, prob_current[0]): #prob_current[0]에 1개의 현재전망이 들어가기 때문에
   print("{:7} : {:.2f}".format(t['name'], t['prob']))


# 6. 2019년 일별 전망치의 확률 변화
# 전기간 전망치 확률 데이터생성
prob = rnd_clf.predict_proba(X)
prob_df = pd.DataFrame(prob, index=y.index, columns=y_names)

# 2019년 전망치 확률의 그래프 출력
prob_2019 = prob_df['2019']
plt.bar(prob_2019.index,   prob_2019['up'],   label='up',   bottom=prob_2019['neutral']+prob_2019['down'],
color='r')
plt.bar(prob_2019.index, prob_2019['neutral'], label='neutral', bottom=prob_2019['down'], color='g')
plt.bar(prob_2019.index, prob_2019['down'], label='down', color='b')
plt.legend()
plt.show()
```

1. 모델 로드

joblib.load을 통해서 pkl 파일로 저장된 랜덤 포레스트 모델을 rnd_clf로 옮긴다.

2. new daily raw data 가져오기

현재 시장 전망을 하기 위한 지표 데이터를 새로 가져오는 부분이다. daily_data4_value.xlsx는 시장 전망을 위해서 어제자까지의 각 지표 데이터가 포함된 최근 데이터셋이다.

3. new daily raw data 전체 학습

계속 운영하는 모델이라고 하면, 날짜가 지나면서 데이터가 쌓일수록 최근 데이터를 계속 포함하는 식으로 훈련 데이터를 늘리는 방식이 유효하다. 3번은 이를 위해서 최근 정보를 포함한 전체 데이터를 재학습하는 루틴이다.

4. 현재(마지막) 데이터 표시

X.iloc[-1,]는 X 데이터프레임 변수에서 마지막 행 데이터를 지칭한다. 코드를 통해서 X_current라는 현재 지표 정보를 담은 1×m 크기의 numpy array가 만들어진다.

5. 현재 전망

y_current_pred = rnd_clf.predict(X_current)를 통해서, 현재 데이터를 가지고 시장 전망을 하게 된다. y_current_pred에 그 결과가 들어가고 'up', 'neutral', 'down' 3가지 결과 중 하나로 결정된다. predict_proba 메서드는 각 클래스별로 계산된 확률 수치를 직접 보여주는 메서드다. 해당 코드를 통해서, 상승, 보합, 하락 투자의견에 대한 각각의 확률을 보여준다.

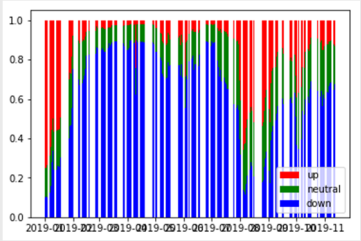
6. 2019년 일별 전망치의 확률 변화

predict_proba 메서드를 통해서, 2019년 연간으로 단기 방향성 전망의 확률이 어떻게 바뀌었는지를 차트로 보여주는 코드다. plt.bar를 통해서 누적 바차트 형태로 그래프를 그린다.

예측 프로세스의 실행 결과는 다음과 같이 출력된다. 이 결과는 11월 12일자 데이터를 넣어서 실행한 결과다.

예측 프로세스 코드 실행 결과

```
< AI model: load >

< AI model: machine learning done >
accuracy_score of whole data:  0.8382388713208465

<Current status>
g10_esi          :    1.100
em_esi           :  -21.500
krw              : 1160.800
export_lag       :  -14.700
wti              :   56.800
core_pce_lag     :    0.050
fwd_pe           :   11.190
yield_gap        :    7.373
pe_em_discount   :   -5.201
pe_dm_discount   :  -29.848
trail_pb         :    0.880
fy2_err_1m       :   -9.480
fq1_err_1m       :   -8.970
fq1_err_1w       :    0.960
recom_chg_1w     :   -0.020
vkospi           :   13.650
vix              :   12.680
put_call_ratio   :   95.414

< AI model: forecasting >
forecast:  ['down']

[class] : [prob]
down    : 0.67
neutral : 0.20
up      : 0.13
```



결과를 나눠서 설명하면 다음과 같다.

```
< AI model: load >

< AI model: machine learning done >
accuracy_score of whole data:  0.8382388713208465
```

1~3번 코드를 통해서, 모델을 다시 로드하고 현시점의 전체 데이터로 학습한 결과를 출력한 것이다. 전 데이터를 대상으로 한 정확도(accuracy) 수치는 83.8%로 확인된다(이런 성능지표는, 회귀분석에서의 결정계수와 비슷한 역할임).

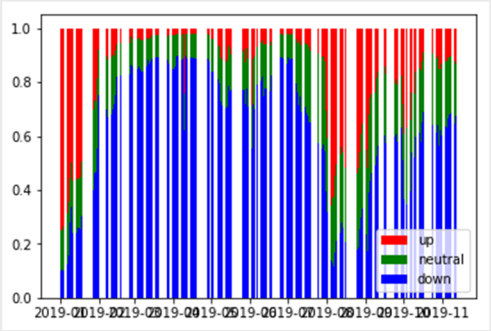
```
<Current status>
g10_esi          :    1.100
em_esi           :  -21.500
krw              : 1160.800
…
put_call_ratio     :   95.414
```

최근 일의 데이터를 입력 받아, 한번 더 표시한 것이다. 11월 12일자 데이터를 최종데이터로 넣었고, G10 ESI 지수=1.1포인트, EM ESI 지수=-21.5포인트 등이 확인된다.

```
< AI model: forecasting >
forecast:  ['down']

[class] : [prob]
down    : 0.67
neutral : 0.20
up      : 0.13
```

현재 AI 모델의 예측치는 '하락'으로 나타남. 모델에서 '하락' 확률이 67%, '보합' 확률이 20%, '상승' 확률이 13%로 나타남

입력 데이터를 바탕으로 한 모델의 예측 결과를 보여주고 있다. 현재 데이터로 모델 예측을 실행한 결과, 단기 방향성 '하락' 확률은 67%, '보합' 확률은 20%, '상승' 확률은 13%로 나왔다. 모델은 각 클래스별 확률이 제일 높은 것을 전망치로 쓰는 방식이다. 따라서, 최종 코스피 단기 방향성 전망은 '하락'으로 나타났다. 즉, 지금부터 향후 3개월간의 코스피 미래 수익률이 0% 미만을 기록할 것으로 예상한다.



올해 연중으로 시장 방향성에 대한 예측 확률의 변화를 그래프로 보여주고 있다. 연초 1월까지는 모델이 '상승' 확률을 높게 보았다가(빨간색), 2월부터는 '하락' 확률이 크게 증가하여 연중으로는 '하락'이 우위다. 8월말 경에 '상승' 확률이 잠깐 높아졌다가 지금은 다시 내려온 상황임을 알 수 있다.

이로써, 파이썬 기반 코스피 단기예측 AI 모델의 구성과 예측이 완성되었다.

백테스팅

모델의 예측 능력에 대한 기본 측정은, 앞서 정확도(accuracy) 수치를 통해서 이미 확인하였다(학습 프로세스 코드 실행 결과 설명 파트 – 정확도 지표 81.0%). 좀 더 눈에 보이는 증거를 위해서, AI 모델을 기반으로 실제 포트폴리오를 운용했을 때의 수익률 성과를 백테스팅 해보았다. AI 모델 기반 운용 백테스팅 방법론 또한, 기존 코스피 단기예측 모델 리포트 상의 방법론과 동일하다. 백테스팅 진행은 모두 Python으로 진행하였다.

백테스팅 방법론은 다음과 같다.

모델 기반 포트폴리오 운용 백테스팅

투자기간 및 주기: 2002년말~2019년 10월말(202개월 기간). 월간 리밸런싱.

투자전략: 매월말에 산출되는 모델 시그널을 바탕으로 한 달간 포지션을 유지함. 모델 시그널이 '상승' 시그널이면 한달 간 KOSPI 매수 포지션을 유지. '보합'이면 현금 보유 포지션 유지(0% 수익률). '하락' 이면 한달간 KOSPI (공)매도 포지션을 유지. 즉, KOSPI 단기 전망이 밝을 때는 KOSPI 지수를 long하는 전략을 유지하고, 단기 하락을 예상할 때는 KOSPI short을 통해 절대 수익을 추구하는 전략임.

참고: 단기예측 AI 모델은 일별로 생성되지만 실제 투자 결정 시에는 매월별로 하루의 정보(매월말 하루 전의 정보)만 사용함. 투자 예측은 3개월 전망 기준이지만, 실제 투자는 한달에 한번씩 이뤄짐.

Python으로 만든 백테스팅 코드는 다음과 같다.

백테스팅 코드: BACKTEST.py

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# -1. MDD 함수 정의
def MDD(list_values):
   mdd_value = 0

   for i in range(1, len(list_values)):
      bw_max = max(list_values[:i])
      curr = list_values[i]
      mdd = curr / bw_max - 1

      if mdd < mdd_value:
         mdd_value = mdd

   return  mdd_value


# 0. 사후방향성 클래스를 수치로 전환하는 함수 정의 (up=1, neutral=0, down=-1)
def convert_num(pred):
```

```
    pred_num = np.empty(len(pred))
    pred_num[pred=='up']=1
    pred_num[pred=='neutral']=0
    pred_num[pred=='down']=-1
    pred_num[pred.isna()]=np.NaN

    return pred_num


# 1. 월간 시장수익률 데이터 가져오기
#    데이터는 일자/월수익률(원수치)/코스피지수/월말하루전영업일로 구성
kdata = pd.read_excel("investing.xlsx", sheet_name="raw", header=13, index_col=0)

# 2. AI모델로 예측한 예측정보를 가져오기
#   기존 AI모델 학습 프로세스에서 만들어낸 y_pred 데이터를 조회함.
#   매월말에 예측했던 투자의견을 가져옴. 단, 실제 투자를 위해서, 월말하루전영업일 기준 자료를 가져옴

kdata['stage']=''

#kdata['stage'] = y_pred #말일자 아님
for index in kdata.index:
    kdata.loc[index,'stage']=y_pred[kdata.loc[index,'before_last']]


# 3. 전월말 투자의견 열 생성
kdata['pre_stage']= kdata['stage'].shift(1)
kdata['port_return']=0


# 4. 전략 수익률 생성
# 전월말 투자의견이 상승이면 코스피 long, 보합이면 Cash, 하락이면 코스피 Short 실행.
# 해당 전략에 따라 포트 월별수익률(port_return) 생성
kdata.loc[kdata['pre_stage']=='up', 'port_return'] = kdata['m_return']*1
kdata.loc[kdata['pre_stage'].isna(), 'port_return'] = kdata['m_return']*1
kdata.loc[kdata['pre_stage']=='neutral', 'port_return'] = 0
kdata.loc[kdata['pre_stage']=='down', 'port_return'] = kdata['m_return']*-1

# 코스피와 모델포트폴리오의 누적수익률(1에서 시작하는 인덱스 형태) 생성
kdata['kospi_cumul']=(1+kdata['m_return']).cumprod()
kdata['port_cumul']=(1+kdata['port_return']).cumprod()


# 5. 백테스팅 결과 기록(CAGR, 변동성, Sharpe ratio, MDD)
my_back = {'months':len(kdata)}

my_back['k_cumul_return_idx']=kdata['kospi_cumul'][-1]
my_back['k_cumul_return_pct']=(my_back['k_cumul_return_idx']-1)*100
my_back['k_cagr']=(my_back['k_cumul_return_idx']**(12/my_back['months']))-1
my_back['k_cagr_pct']=my_back['k_cagr']*100
my_back['k_vol_pct']=np.std(kdata['m_return'])*np.sqrt(12)*100
my_back['k_Sharpe']=my_back['k_cagr_pct']/my_back['k_vol_pct']
my_back['k_MDD']=MDD(kdata['kospi_cumul'])*100

my_back['port_cumul_return_idx']=kdata['port_cumul'][-1]
my_back['port_cumul_return_pct']=(my_back['port_cumul_return_idx']-1)*100
my_back['port_cagr']=(my_back['port_cumul_return_idx']**(12/my_back['months']))-1
my_back['port_cagr_pct']=my_back['port_cagr']*100
my_back['port_vol_pct']=np.std(kdata['port_return'])*np.sqrt(12)*100
my_back['port__Sharpe']=my_back['port_cagr_pct']/my_back['port_vol_pct']
my_back['port_MDD']=MDD(kdata['port_cumul'])*100


# 6. 백테스팅 결과 출력하기
```

```
print("<Backtesting result>")
for key, value in my_back.items():
   print("{:22}: {:>8.3f}".format(key, value))

# 포트폴리오 누적수익률 그래프
kdata['port_cumul'].plot()
plt.title('Portfolio performance index')
plt.ylabel('\'02/12/31 = 1')
plt.show()


# 7. 월말 모델전망치와 실제결과치 출력
# 월말 실제결과치 입력
for index in kdata.index:
   kdata.loc[index,'real_stage']=y[kdata.loc[index,'before_last']]

# 사후방향성 클래스를 수치로 변환
kdata['stage_num']=convert_num(kdata['stage'])
kdata['real_stage_num']=convert_num(kdata['real_stage'])

# 전망치와 결과치의 그래프 출력
kdata.plot(y=['stage_num', 'real_stage_num'], label=['model forecast','real direction'])
plt.title('Model forecast vs. Real direction')
plt.ylabel('up=1, neutral=0, down=-1')
plt.show()
```

-1. MDD 함수 정의

MDD 계산은 간단한 함수 정의를 통해서 계산할 수 있다.

0. 사후방향성 클래스를 수치로 전환하는 함수 정의

이 코드에서 상승/보합/하락의 클래스를 1/0/-1의 수치로 변환하는 함수를 정의하였다.

1. 월간 시장수익률 데이터 가져오기

미리 만들어진 시장수익률 데이터셋의 엑셀파일을 가져와서 kdata 데이터프레임을 만든다. 시장수익률 데이터셋은 다음과 같다. 자료는 base_date(당월말), m_return(월간 수익률), kospi(코스피지수), before_last(월말 하루전 영업일)로 구성되어 있다. before_last는 월말 하루전 날짜의 예측정보를 가져올 때 사용된다. AI 모델은 G10 ESI, WTI 등의 해외데이터를 사용하기 때문에, 실제 투자할 때는 하루 전의 데이터만 쓸 수 있다. 따라서, 월말에 투자를 집행할 때는 before_last에 들어있는 하루 전 기준의 예측데이터를 사용하여 투자를 결정하게 된다.

시장수익률 데이터셋(초기 kdata) 예제

base_date	m_return	kospi	before_last
2003-01-30	-0.057	591.86	2003-01-29
2003-02-28	-0.028	575.43	2003-02-27
2003-03-31	-0.069	535.70	2003-03-28
…			

참고: m_return(월간 수익률)은 % 형태가 아닌 원수치값 기준
자료: 삼성증권

2. AI모델로 예측한 예측정보를 가져오기

기존 AI모델 학습 프로세스에서 만들어낸 일별 예측 정보 y_pred를 조회하여, 매월말(하루전 기준)의 모델 예측값을 stage 열에 기록한다.

3. 전월말 투자의견 열 생성

kdata 데이터프레임(엑셀 표 개념)에 pre_stage라는 전월말 투자의견 데이터를 생성한다. 4번 스텝에서 배열연산을 쉽게 하기 위해 전월 데이터 열을 생성한 것이다.

4. 포트폴리오의 수익률 생성

포트폴리오 수익률 생성 단계다. 전월말 투자의견이 상승이면 코스피 수익률, 보합이면 0% 수익률, 하락이면 코스피 수익률*-1을 입력했다. 전월말 투자의견이 na인 경우는(2003/1/30 첫달만 해당) 포트폴리오 수익률을 KOSPI 수익률과 동일하게 설정했다.

가령, 투자전략 백테스팅 기준을 "보합의견이면 코스피 Long을 유지한다"는 식으로 만약 바꾼다고 한다면, 여기 코드를 변경하면 된다.

코스피의 누적수익률은 kospi_cumul 열로 생성하고, 포트폴리오의 누적수익률은 port_cumul 열로 생성했다(1에서 시작하는 인덱스 형태).

5. 백테스팅 결과 기록(CAGR, 변동성, Sharpe ratio, MDD)

백테스팅 결과를 정리하는 단계다. CAGR, 연변동성, Sharpe ratio, MDD 등을 계산했다(이 때, Sharpe ratio에서는 Rf=0을 가정함). Python의 Dictionary 자료형을 사용했다.

6. 백테스팅 결과 출력하기

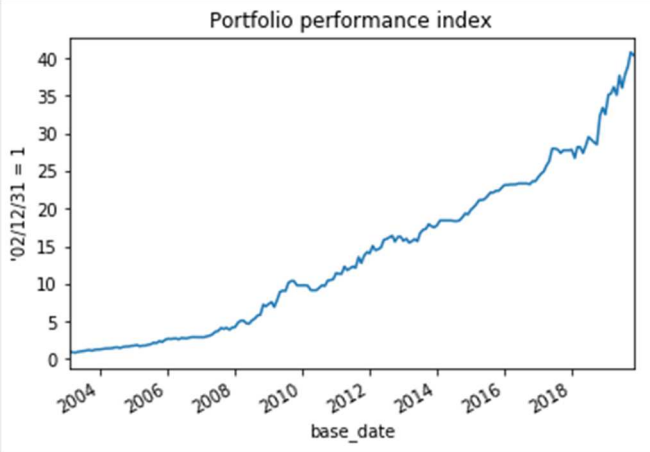
백테스팅 결과를 출력하는 단계다. 포트폴리오의 누적수익률 그래프도 같이 출력한다.

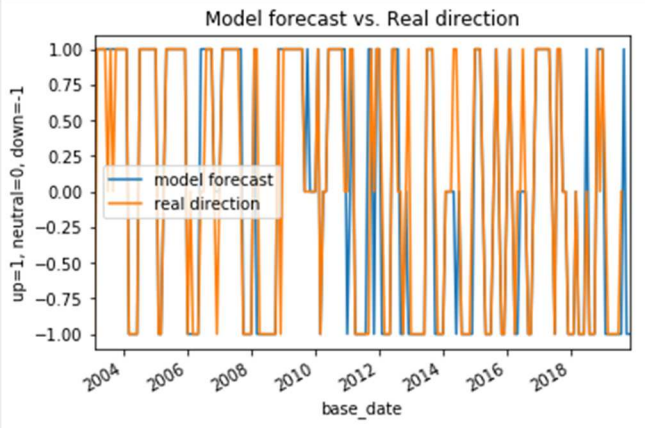
7. 월말 모델전망치와 실제결과치 출력

매월말에 모델이 예측했던 전망(상승/보합/하락)과 실제 사후결과를 비교해서 차트로 보여주는 코드다. 모델에서 예측했던 전망(상승/보합/하락)이 kdata['stage']에 들어 있고, 실제 사후결과는 kdata['real_stage']에 들어있다. 모델 예측 전망치를 '상승=1, 보합=0, 하락=-1'의 형태로 수치로 변환한 정보가 kdata['stage_num']에 들어 있다. 사후 결과의 수치 변환 정보는 kdata['real_stage_num']에 들어 있다. 이를 가지고 plot 차트를 만들었다.

백테스팅 코드 실행 결과

```
<Backtesting result>
months            : 202.000
k_cumul_return_idx   :   3.320
k_cumul_return_pct   : 232.001
k_cagr            :   0.074
k_cagr_pct         :   7.389
k_vol_pct          :  17.986
k_Sharpe           :   0.411
k_MDD             : -48.518
port_cumul_return_idx :  40.369
port_cumul_return_pct : 3936.893
port_cagr          :   0.246
port_cagr_pct       :  24.569
port_vol_pct        :  16.098
port__Sharpe        :   1.526
port_MDD           : -11.782
```





백테스팅 코드의 실행 결과는 위와 같이 나온다.

텍스트 부분은 my_back 딕셔너리의 데이터를 모두 출력함으로써, 백테스팅 결과 정보를 보여주고 있다. 도표로 다시 정리하면 다음과 같다.

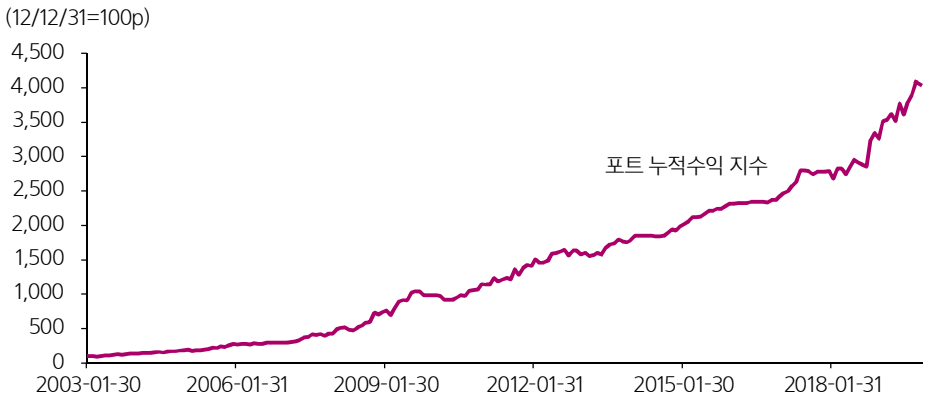
AI모델 기반 포트폴리오 백테스팅 결과

성과지표	KOSPI 투자 (k)	모델기반 포트폴리오 (port)
누적수익률 (%)	232.00	3,936.89
연환산 수익률 (%)	7.39	24.57
연환산표준편차 (%)	17.99	16.10
Sharpe ratio (Rf=0% 가정)	0.41	1.53
MDD (Maximum Draw Down, %)	-48.52	-11.78

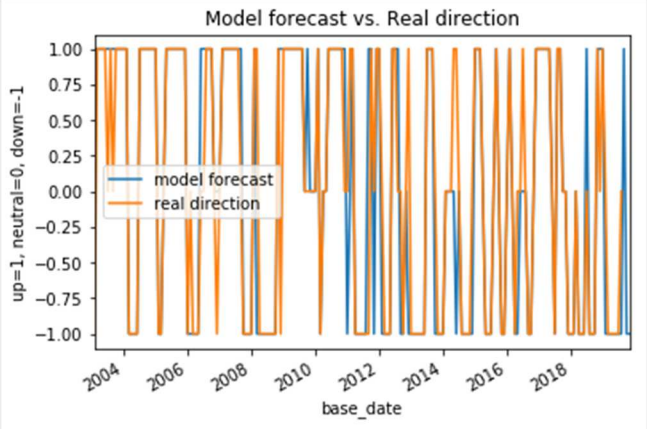
자료: 삼성증권

모델 기반 포트폴리오의 연환산 수익률은 24.57%이다. KOSPI Buy & Hold를 크게 아웃퍼폼하는 것으로 확인된다. 코스피 단기예측 모델 최초 리포트 상에서 모델 기반 포트폴리오의 연수익률이 23.2%가 나왔음을 비교해보면, 이번 모델 기반 포트폴리오의 투자 성과가 더 개선되었음을 알 수 있다.

단기예측 AI 모델 기반 포트폴리오, 누적수익 지수



참고: 기초를 100포인트로 지정하여 누적성과를 표시한 그래프. "port_cumul_return_idx : 40.369" 혹은 Python 출력 첫 번째 그래프와 동일한 결과임
자료: 삼성증권



코드 실행 결과 2번째 차트에서는, 전기간 동안 월별 모델 전망치와 실제 사후결과가 얼마나 일치했는지를 보여주는 차트다. 상승(사후 3개월 수익률 4% 이상) 국면이면 1로 표시하고, 보합(3개월 수익률 4~0%) 국면이면 0으로 표시, 하락(수익률 0% 미만) 국면이면 -1로 표시했다. 과거 데이터를 보면, 모델 전망치와 실제 결과치가 상당히 유사하게 움직였음을 눈으로 확인할 수 있다.

이상으로 모델 기반 전략의 백테스팅 결과까지 확인해 보았다.

후속 리포트에서는, 시장 단기예측을 위해 당초에 검토했던 200여 개의 투자지표들로 사용 데이터 후보를 넓혀볼 계획이다. 이를 통해서 더 많은 데이터를 통한 더 예측력이 높은 모델을 개발할 계획에 있다.

Contents

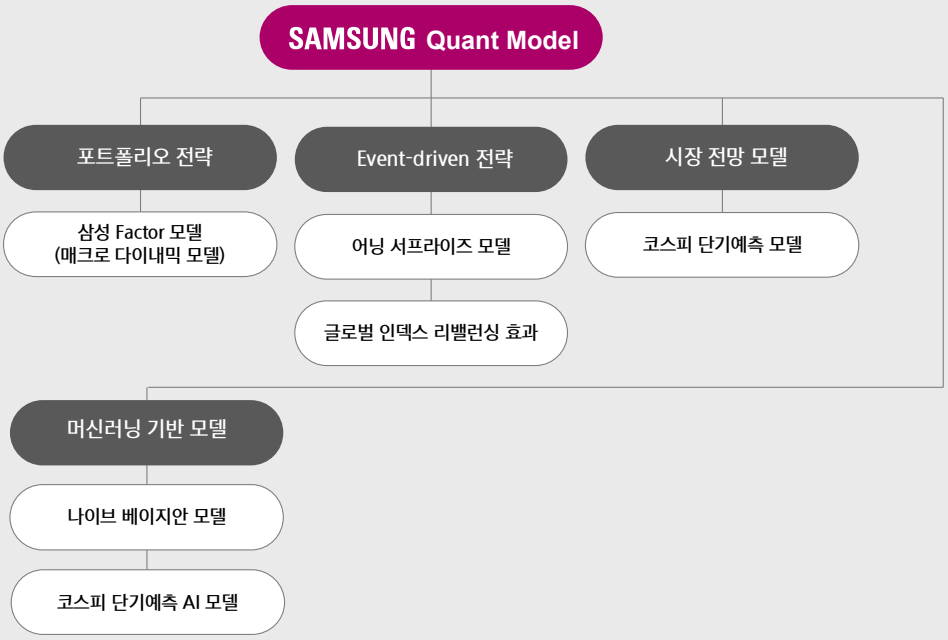
V. 마무리

이상으로 랜덤 포레스트 머신러닝 기법을 활용하여 코스피를 단기 전망하는 모델을 완성하였다. 필자는, 이를 활용하여 월별로 코스피 단기예측 결과를 제공할 계획이다.

이처럼 금융시장 예측 및 투자 업무에서 머신러닝과 같은 새로운 기술의 도입은 갈수록 커질 것으로 본다. 그럼 투자업에서 기계가 애널리스트를 완전히 대체할 것인가? 그건 아니라고 생각한다. 데이터를 잘 쓰기 위해서는 금융시장을 이해하는 애널리스트의 역할이 꼭 필요하다. 좋은 머신러닝 모델, 좋은 알고리즘이라도, 쓰레기 수치가 들어가면 쓰레기 결과가 나온다. 이번 리포트의 AI 모델은, 이전에 발간된 '코스피 단기예측 모델' 리포트에 많은 빚을 지고 있다. 시장 단기 흐름에 영향을 끼치는 주요 데이터를 선별하는 작업이 중요하고, 이 부분에서 오랜 경험 정보가 사용되었다. 데이터가 가짜 연관성을 보인 것은 아닌지 등의 질적인 체크가 항상 필요하다. 또한, 머신러닝 모델은 블랙 박스 형태를 가지는 경우가 많다. 모델이 잘 작동하는지에 대해서는 업무 전문가가 다각도로 데이터와 현실과의 관계에 대해서 고민하고 체크하는 작업이 필요하다.

머신러닝 모델 그 자체는 판단하는 영혼도, 직관적인 결정도 없다. 대량의 데이터를 쓰고 빠르게 처리함으로써 그런 단점을 커버하려고 노력하는 것이다. 머신러닝의 정량적 판단과 인간의 정성적 판단을 적절히 조화하는 것이, 시장에 대한 예측력을 좀 더 높이는 길이라고 판단한다.

삼성 퀀트 모델 소개



[모델별 최신 리포트]

삼성 Factor 모델:

18/9/18 "Advanced 매크로 다이내믹 모델: 팩터 로테이션 정교화"

19/4/15 "팩터 모델의 구조와 해설: Factor Models for Asset Returns"

19/10/16 "월간 팩터 포트폴리오: 2019년 10월 ~ 11월"

인덱스 리밸런싱 효과:

19/3/3 "MSCI의 중국 A주 비중 확대 결정: 확실해진 충격"

19/8/26 "FTSE 2019년 9월 정기변경 발표"

19/11/8 "MSCI 2019년 11월 정기변경 발표: 한국 비중 감소"

코스피 단기예측 모델:

19/9/3 "코스피 단기예측 모델: Part 1. 매크로 지표 설명"

19/11/4 "코스피 단기예측 모델: Part 3. 어닝스 및 리스크 지표 설명"

머신러닝 기반 모델:

16/1/4 "기계학습(Machine Learning)과 투자전략: 빅 데이터 기법 Naive Bayes Classifier의 활용"

19/11/18 " 코스피 단기예측 AI 모델: 랜덤 포레스트 기법을 활용한 머신러닝 기반 모델"

Compliance notice

- 본 보고서는 철저히 계량적 분석에 근거한 의견을 제시합니다. 따라서 당사의 대표 투자의견과 다를 수 있습니다.
- 본 조사분석자료의 애널리스트는 11월 15일 현재 위 조사분석자료에 언급된 종목의 지분을 보유하고 있지 않습니다.
- 당사는 11월 15일 현재 위 조사분석자료에 언급된 종목의 지분을 1% 이상 보유하고 있지 않습니다.
- 본 조사분석자료에는 외부의 부당한 압력이나 간섭없이 애널리스트의 의견이 정확하게 반영되었음을 확인합니다.
- 본 조사분석자료는 당사의 저작물로서 모든 저작권은 당사에게 있습니다.
- 본 조사분석자료는 당사의 동의없이 어떠한 경우에도 어떠한 형태로든 복제, 배포, 전송, 변형, 대여할 수 없습니다.
- 본 조사분석자료에 수록된 내용은 당사 리서치센터가 신뢰할 만한 자료 및 정보로부터 얻어진 것이나, 당사는 그 정확성이나 완전성을 보장할 수 없습니다. 따라서 어떠한 경우에도 본 자료는 고객의 주식투자의 결과에 대한 법적 책임소재에 대한 증빙자료로 사용될 수 없습니다.
- 본 조사분석자료는 기관투자가 등 제 3자에게 사전 제공된 사실이 없습니다.

삼성증권 SAMSUNG

삼성증권주식회사
06620 서울특별시 서초구 서초대로 74길 11 10층 리서치센터
02 2020 8000

지점 대표번호
1588 2323 / 1544 1544

고객 불편사항 접수
080 911 0900

samsung POP.com

신뢰에 가치로 답하다