머신러닝을 이용한

알고리즘 트레이딩 시스템 개발

안명호, 류미현 지음

한빛미디어 Hanbit Media Inc

표지 사진 **임희진**

이 책의 표지는 임희진님이 보내 주신 풍경사진을 담았습니다.
리얼타임은 독자의 시선을 담은 풍경사진을 책 표지로 보여주고자 합니다.

사진 보내기 ebookwriter@hanbit.co.kr

머신러닝을 이용한 알고리즘 트레이딩 시스템 개발

초판 1쇄발행 2016년 5월 24일
초판 4쇄발행 2018년 2월 26일

지은이 안명호, 류미현 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 서대문구 연희로2길 62 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호 / **ISBN** 978-89-6848-818-4 13000

총괄 전태호 / **책임편집** 김창수 / **기획·편집** 정지연
디자인 표지/내지 여동일, 조판 최송실 / **제작** 박성우, 김정우
영업 김형진, 김진불, 조유미 / **마케팅** 박상용, 송경석, 변지영

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
잘못된 책은 구입하신 서점에서 교환해 드립니다. 책값은 뒤표지에 표시되어 있습니다.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(writer@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_ **안명호**

KAIST SW석사과정을 마쳤다.

어느 날 알게 된 머신러닝에 흠뻑 빠져 그동안 애지중지하던 클라우드를 버리고 머신러닝으로 전향하였다. 이제 더는 다른 기술은 관심을 두지 않고 머신러닝 한길만으로 정했기에 머신러닝을 공부하며 어려운 수식들을 다시 보느라 고생하고 있지만, 하루하루 배워가는 지식에 행복해하며 지내고 있다. 머신러닝으로 가마우지를 만들어 인생을 즐기려 노력하고 있으며, 그 결실이 완성되는 날 완벽한 경제적 자유를 누리고자 한다.

- Facebook https://www.facebook.com/james.ahn.9
- Homepage http://www.deepnumbers.com

지은이_ **류미현**

동국대 정보공학석사를 마쳤다.

머신러닝 책을 집필할 때만 해도 쉬엄쉬엄하는 마음으로 했는데, 딥마인드DeepMind의 데미스 하사비스Demis Hassabis가 '알파고AlphaGo'를 서울에 데려와 파문을 일으키고 가는 바람에 왠지 모를 조바심이 생겼다. 그동안 딥러닝에 대해 갸우뚱하던 생각도 무지의 소치로 치워두고 급상승한 호기심을 발판으로 깊숙이 들어가 보려 한다. 확신할 수 없을 때는 불안하고 머뭇거리게 되지만 다행히 누군가 그 길을 보여주면 그때라도 놓치지 말고 따라가는 게 낫지 않을까 생각한다.

패러다임의 변화, 소프트웨어에서 데이터로

바야흐로 데이터의 시대로 발전해가고 있다. 얼마 전까지만 하더라도 소프트웨어의 중요성을 강조하고 소프트웨어가 세상을 먹어치우고 있다는 다소 도발적인 문구가 많이 회자했다. 언론에서도 이에 발맞추어 소프트웨어가 우리의 삶과 산업에 미치는 영향을 보도하고 새로운 소프트웨어 기술을 소개하며 영향력 있는 개발자들의 인터뷰를 실었다. 당연히 소프트웨어가 중요하다는 점에서는 이견이 없다. 지금도 그렇고 앞으로도, 말할 나위 없이 소프트웨어는 중요하다.

이야기하고 싶은 것은 경쟁력과 가치다. 오픈소스의 확산은 소프트웨어 발전에 기념비적인 사건이라 할 수 있다. 이는 오픈소스가 소프트웨어에서 데이터로 패러다임을 변화시키는 핵심 동인이기 때문이다. 과거에는 소프트웨어 자체가 경쟁력이 될 수 있었다. 지금처럼 소프트웨어의 수가 많지도 않았고, 소스 코드가 공개된 소프트웨어의 수도 많지 않았다. 또한, 사용하는 소프트웨어의 기능과 품질이 우수할수록 경쟁력이 있었다.

현재는 오픈소스로 인해 사용할 만한 소프트웨어가 넘쳐나고 있다. 간단한 기능을 수행하는 오픈소스에서, 거대한 IT 서비스를 운영하는 데 사용할 정도로 강력한 오픈소스에 이르기까지 다양한 분야에 걸쳐있다.

더구나 최근의 오픈소스 프로젝트는 과거의 것과 비교할 수 없을 정도로 수준이 높아졌다. 구글, 페이스북처럼 전 세계 IT 산업을 이끄는 회사에서 자사의 필요로 개발한, 자사의 서비스에 사용하는 고품질, 고기능의 소프트웨어를 오픈소스로 앞다퉈 공개하고 있다. 얼마 전 구글에서 오픈소스로 공개한 TensorFlow[01]는 딥러닝Deep Learning 라이브러리로, 구글에서 머신러닝 관련 작업, 예를 들어 이미지 인식 등에 사용하는 것으

01 https://www.tensorflow.org/

로 기능이 떨어지거나 학습결과물의 품질이 떨어지는 조악한 것이 절대 아니다.

이처럼 오픈소스의 확산은 소프트웨어 자체가 더는 경쟁력이 될 수 없는 환경을 만들었다. 머신러닝을 구현하고 싶을 때 구글의 TensorFlow를 이용하면 세계 최고 수준의 머신러닝 라이브러리를 사용할 수 있다. 실시간 빅데이터 분석을 하고 싶을 때 링크드인의 Pinot[02]을 이용하면 역시 세계 최고 수준의 실시간 빅데이터 분석 프로그램을 쉽게 개발할 수 있다. 이렇게 몇 번의 검색만으로 최고 수준의 소프트웨어를 쉽게 얻는 환경이 되었다.

이런 환경에서 경쟁력은 이제 한 단계 끌어올려 져 '지능화'될 수밖에 없다. 기존의 소프트웨어가 어떤 기능을 제공하는 것이라면 지능화를 갖춘 소프트웨어는 자동화를 통해 번거로움을 없애주고, 판단을 통해 새로운 가치를 전달할 수 있게 된다. 이 책의 내용에 빗대어 주식으로 예를 들면, 기존의 주식 관련 소프트웨어는 "주식거래를 간편하게, 빨리, 원하는 시점에 할 수 있다"라는 기능 중심의 가치를 제공한다면, 지능화를 갖춘 소프트웨어는 "지금 xx 주식을 사면 10%의 이익을 볼 수 있다"라는 편익 중심의 가치를 제공한다.

지능화를 위해 필요한 것이 데이터고, 머신러닝을 이용하면 데이터를 지능으로 바꿀 수 있다. 대표적인 머신러닝 기술 중 하나인 딥러닝이 이에 대한 좋은 예다. 딥러닝은 1943년에 발표된 신경망Neural Network이라는 기술에 기반을 둔 것으로 2000년대 초반까지만 하더라도 낮은 성능과 기술상의 제약으로 인해 거의 사장되었었다. 하지만 2000년대 후반에 딥러닝으로 화려하게 부활했다. 알고리즘의 발전도 이에 중요한 역할을 했지만, 컴퓨팅 파워Computing Power와 특히 데이터가 없었다면 불가능했다.

머신러닝을 위해 필요한 요소는 알고리즘, 컴퓨팅 파워, 데이터 3가지다. 알고리즘은

---

02 https://github.com/linkedin/pinot/wiki

인터넷을 통해 비교적 쉽게 구할 수 있고, 컴퓨팅 파워 역시 비용만 있다면 아마존과 같은 퍼블릭 클라우드를 사용해 쉽게 해결할 수 있다.

하지만 앞의 2가지와 다르게 데이터는 그렇지 않다. 신경망을 비롯한 머신러닝 알고리즘들은 많은 데이터가 있을수록 더욱 좋은 성능을 보이기 때문에 양질의 데이터를 많이 가질수록 유리하다. 인터넷으로 인해 이전보다는 데이터를 구하기 쉬운 것은 사실이지만 데이터 오너십Data Ownership의 문제로 인터넷을 통해 구할 수 있는 데이터는 한정되어 있고, 신상정보나 카드사용명세처럼 중요한 가치를 가지고 있는 정보는 공개되지 않은 경우가 대부분이다.

결국, 쓸만한 양질의 데이터를 누가 많이 가지고 있느냐에 따라 지능화의 품질이 달라지는 것이다. 관심도 없는 제품광고를 시도 때도 없이 보내는 서비스보다는 내가 사고 싶은 물건에 대한 할인 광고를 보여주는 서비스가 사람들로부터 사랑받듯이, 앞으로의 싸움은 기능이 아닌 편익 중심으로 펼쳐질 수밖에 없고 여기에서 핵심은 단연코 데이터가 된다.

금융과 IT 그리고 머신러닝

금융은 숫자를 다루는 산업으로 IT 기술이 매우 중요한 역할을 한다. 입금하고 출금하고 내 계좌의 돈이 얼마나 있는지 확인하고 다른 사람에게 송금할 때, 이 모든 것이 IT 기술로 실행된다. 내가 송금한다고 해서 실제의 물리적 존재인 돈이 송금받는 사람에게 전달되는 것은 아니다. 돈을 찾기 전까지는 IT 시스템 어딘가의 데이터베이스에 기록되어 있는 나의 예금이 송금 액수만큼 줄어들고 송금받는 사람의 데이터베이스 기록의 예금이 해당 액수만큼 늘어나는 것이다.

IT 기술이 없다면 지금 우리가 누리는 여러 가지 금융서비스는 시간이 오래 걸리거나

불가능한 것이 많을 것이다. 그동안 금융에서는 IT 시스템을 금융거래에 관련된 데이터의 기록, 삭제, 수정 등의 기능 수행 목적으로 많이 활용했다. 하지만 최근에는 금융에서 IT 시스템을 한 차원 높은 수준으로 바라보기 시작했다.

간단히 말하면 머신러닝을 이용한 지능화라고 할 수 있다. 금융기관에서 가지고 있는 막대한 양의 금융 데이터를 단순 데이터베이스처럼 사용하는 것이 아니라, 이들 데이터에서 머신러닝을 이용해 위험도를 측정하고 앞으로 주가의 방향을 예측하고 새로운 사업기회를 찾는 등 이전과는 다른 시각과 기술로 활용하기 시작했다.

머신러닝은 많은 데이터가 필요하고 강력한 컴퓨팅 파워가 뒷받침되어야 하는데, 금융계는 이러한 조건들을 만족하는 환경을 이미 가지고 있다. 금융 데이터는 수치데이터고 여타의 데이터보다 정확해서 머신러닝에 적용하기 매우 좋은 조건을 가지고 있다. 예를 들어, 소셜분석을 하고자 하면 사람들이 작성한 글을 분석해 적절한 수치로 변환한 후에 머신러닝 알고리즘을 이용해 분석하거나 예측할 수 있는데, 분석하는 과정과 수치로 변환하는 과정에서 어느 정도의 손실과 오류는 필연적이다. 더구나 소셜분석으로 글에 쓰인 내용이 글쓴이의 생각을 100% 대변하는 것인지, 아니면 다른 이유로 그렇지 않은지를 정확하게 알기 힘들다.

이러한 흐름을 잘 보여주는 것이 세계적인 투자은행인 골드만삭스다. 비지니스인사이더의 2015년 4월 12일 기사를 보면 'Goldman Sachs is a tech company'[03]라는 제목으로 골드만삭스의 변신에 대해 자세히 소개하고 있다.

투자은행인 골드만삭스의 전 세계 직원 수는 33,000명인데, 이 중 9,000명이 엔지니어와 프로그래머라고 한다. 즉, 전 직원의 30%가 IT 관련 직종이다. 페이스북의 IT 관련 전체 직원 수가 9,199명이고 트위터의 전체 직원 수가 3,638명, 링크드인이

---

03 Jonathan Marino (2015). Goldman Sachs is a tech company. http://goo.gl/UruxtF

6,897명이라는 것을 생각해보면 골드만삭스의 IT 관련 인원이 얼마나 많은 수인지를 쉽게 짐작할 수 있다.

금융계에서의 대표적인 머신러닝 활용은 알고리즘 트레이딩Algorithmic Trading이다. 알고리즘 트레이딩은 사람이 아닌 프로그램에 의해 거래하는 것으로 이미 미국에서는 2012년에 전체 거래의 85%를 차지할 정도로 일반화되었다. 우리가 흔히 주식시장에서 얘기하는 외국인은 사람이 아닌, 기관이나 헤지펀드에서 사용하는 알고리즘 트레이딩 프로그램이라고 생각해도 결코 과언이 아니다.

금융계에서의 IT 활용 확대에 따라 이제는 전 세계 금융의 메카인 월스트리트에서도 MBA나 재무, 경제를 전공한 전통적인 이미지의 금융맨들이 아닌, 너무도 이질적으로 보이는 수학자, 천문학자, 물리학자, 컴퓨터과학자를 찾는 것이 쉬운 일이 되어버렸다. 머신러닝과 같은 IT 기술을 활용하는 것이 금융기관의 수익을 높여줄 수 있을 뿐만 아니라, 새로운 사업기회를 찾을 수 있고 일의 효율성을 극대화한다는 것이 오랫동안 증명되었기 때문에 금융과 머신러닝 같은 IT 기술의 조우는 더욱 많아질 것이다.

이 책을 집필한 이유

처음 머신러닝을 접했을 때 환상적인 개념과 놀라운 결과에 무척 흥분했었다. 예전에 수행한 프로젝트에서 물체를 인식하는 프로그램을 개발했었는데, 무척이나 힘들고 괴로운 과정이었다. 사람의 눈으로는 너무 쉽게 구별되는 물체를 프로그램으로 컴퓨터에 인식하게 하려니 정말 많은 수학 지식이 필요했고, 이를 코드로 구현하기도 쉽지 않았다.

물체 인식을 위한 모든 것을 하나하나 코드로 프로그램에 넣어야 했고, 발생할 수 있는 예외상황과 이미지 특성들도 역시 포함해야 했다. 다른 형태를 지닌 물체를 인식하게

하려면 앞서 했던 고생을 새롭게 반복해야 했다. 이런 고생을 하더라도 결과가 좋으면 행복했겠지만, 현실은 냉정했다. 특정한 조건에서는 인식률이 높았지만, 그 조건을 조금이라도 벗어나는 환경이 되면 인식률은 곤두박질하기가 부지기수였다.

그런데 머신러닝 예제로 여기저기서 쉽게 볼 수 있는 'MNIST Digit Recognition'을 보면 너무도 쉽게 숫자를 인식한다. 예제 코드도 길지 않은데, 정확도가 95~98%에 이르니 예전 기억을 떠올려보면 놀라운 세계였다. 그래서 머신러닝에 대한 설레는 기대감으로 여행을 시작했다. 어느 여행이나 마찬가지지만 여행안내서에 나오는 그 환상적인 절경들은 그냥 쉽게 만나는 것이 아니라는 것을 깨닫는 데 오랜 시간이 걸리지 않았다. 역시 예제는 예제일 뿐이라는 것을 다시 한 번 확인하는 계기가 되었다. 하지만 머신러닝에 대한 강한 끌림으로 여행을 포기하고 싶지 않았기 때문에 본격적인 공부를 시작했다.

필자의 기준으로 보면 머신러닝 책들은 크게 2가지 분류로 할 수 있었다. 첫 번째는 머신러닝에 대한 소개 차원의 책과 두 번째는 머신러닝의 수학적 배경을 설명하는 책이다. 전자의 책은 머신러닝 알고리즘에 대한 소개와 사용방법에 치중하여 머신러닝 사용법을 학습하는 데는 도움이 되었지만, 머신러닝의 개념을 이해하고 무엇을 하기에는 내용이 부족하다는 생각을 지울 수 없었다. 후자의 책은 반대로 너무 전문적이어서 머신러닝 알고리즘에 사용된 수학적 개념들과 각종 정리를 소개하는 데 치중했다. 머신러닝 알고리즘을 새로 개발하거나 혹은 기존 머신러닝 알고리즘을 개선해서 사용하려 한다면 이런 책들이 도움되지만, 머신러닝 알고리즘을 사용하는 데 중점을 둔 입장에서는 너무 지나친 감이 있었다.

목마른자가 우물을 파는 심정으로 머신러닝을 묵묵히 공부해본 결과, 머신러닝을 활용하는 데는 핵심적인 수학 개념에 대한 이해와 적용하려는 분야에 대한 도메인 지식 Domain Knowledge이 중요하다는 것을 깨닫게 되었다.

사실 머신러닝을 이용해 프로그램을 작성하는 데 머신러닝 알고리즘이 차지하는 비중은 그렇게 크지 않다. 중요한 것은 데이터에 대한 이해와 특성을 파악하는 것이다. 이런 과정을 데이터 탐색 분석EDA, Explonatory Data Analysis이라고 하는데, 이 과정을 제대로 수행하기 위해서는 통계와 확률에 대한 수학적 지식이 요구된다. EDA를 효과적으로 수행하는 데 도메인 지식이 있다면 시간을 대폭 단축할 수 있고, 문제를 단순화할 수 있다. 이런 과정을 거쳐 적용한 머신러닝이야말로 좋은 결과를 보여준다.

머신러닝을 공부하는 데 필요한 수학적 이론과 도메인 지식, 이를 구현한 코드를 한곳에서 볼 수 있는 책이라면 도움이 되지 않을까 하는 생각으로 이 책을 집필하게 되었다. 머신러닝을 제대로 활용하기 위해서는 데이터가 더욱 중요한 이슈라고 얘기했듯이, 머신러닝을 이야기하는 데 적용분야인 도메인을 정하지 않는 것은 반쪽짜리다. 그래서 적용대상으로 쉽게 데이터를 얻을 수 있으며, 데이터 자체에 대한 신뢰도가 높고 난도가 있는 주식을 선택했다.

주식시장은 예측이 어려운 대표적인 분야로, 머신러닝을 공부하는 데 필요한 모든 요소를 가지고 있다고 할 수 있다. 수학이론을 이용한 예측 모델의 작성, 작성한 모델을 위한 데이터 처리, 주가를 예측하기 위한 학습, 학습결과의 해석과 이를 개선하는 방법 등 머신러닝 전체 흐름을 경험하기에 좋다.

통계, 시계열Time Series, 알고리즘 트레이딩 등 책에서 다루는 주제는 각각 한 권의 책으로도 분량이 모자랄 만큼 방대한 내용을 가지고 있으나, 이 책의 목적상 알고리즘 트레이딩과 직접 연관되고 반드시 알아야 하는 사항들을 중심으로 서술했다.

부족한 내용은 별도의 책을 통해 지식을 습득해야 한다. 아울러 프로그래밍을 할 줄 아는 독자들을 대상으로 하므로 프로그래밍에 관련된 설명은 특별히 하지 않았다.

이 책의 구성

이 책은 크게 3부분으로 구성되어 있다.

Part 1은 머신러닝의 개요로, 머신러닝이 무엇인지와 머신러닝이 할 수 있는 것은 무엇인지, 머신러닝의 종류는 무엇이 있는지 등 머신러닝의 전반적인 개요를 설명한다.

1장 머신러닝이 첫 번째 Part인데, 특히 중요한 의미가 있는 내용은 머신러닝에 대한 개념 파악과 머신러닝이 해결할 수 있는 문제, 머신러닝 프로세스, 마지막으로 'No Free Lunch Theorem'이다.

Part 2는 알고리즘 트레이딩을 위한 수학적 배경지식으로 통계와 시계열을 다룬다. 알고리즘 트레이딩을 하려면 주식의 매도와 매수를 결정하는 '모델'이라는 것을 만들어야 한다. 이 모델을 만들기 위한 최소한의 통계 개념과 시계열 개념을 설명한다.

2장 통계는 모든 머신러닝의 가장 기초가 되는 것으로, 머신러닝을 올바르게 사용하고 활용하기 위해서 반드시 이해하고 넘어가야 하는 개념들인 표준편차, 히스토그램, 정규분포 등을 설명했다.

3장 시계열 데이터는 알고리즘 트레이딩의 이론적 토대가 되는 수학적 개념들을 소개한 것으로, 마지막 알고리즘 트레이딩 구현을 위해 필요한 내용이 최소한으로 담겨 있다.

Part 3은 실제로 간단한 알고리즘 트레이딩을 파이썬을 이용해 구현해본다. 머신러닝에 기반을 둔 모델과 시계열 이론에 기반을 둔 모델 2가지를 구현해보고, 구현된 결과에 대한 해석과 개선방법에 대해 다룬다.

4장 알고리즘 트레이딩에서는 국내에서 낯선 개념인 알고리즘 트레이딩이 무엇인지 소개하고, 알고리즘 트레이딩에 사용할 2가지 모델을 설명한다.

5장 알고리즘 트레이딩 시스템 구현은 파이썬과 라이브러리를 이용해 4장에서 설명한 2가지 모델을 실제로 구현해본다.

6장 성능 평가와 최적화는 5장에서 구현한 모델의 예측 성능을 평가하는 방법과 더 높은 예측 성능을 위해 어떻게 최적화해야 하는지를 설명한다.

이 책의 내용을 실습하는 데 필요한 소프트웨어와 하드웨어

머신러닝을 하려면 당연히 소프트웨어와 하드웨어가 필요하다. 머신러닝을 위한 하드웨어는 되도록 빠른 속도를 가진 CPU를 사용하는 것이 좋다. 머신러닝 알고리즘은 다른 소프트웨어보다 계산을 많이 하기 때문에 실행시간이 적게는 수 초에서 길게는 수 개월이 걸릴 수 있어서 CPU가 빠를수록 결과를 빨리 얻을 수 있다.

머신러닝을 본격적으로 해보고 싶다면 CPU보다는 GPU 사용을 강력히 추천한다. GPU는 수학과 과학 계산에 특화된 장치로, 매우 빠른 계산 속도를 보여준다. CPU는 직렬처리를 하지만, GPU는 병렬처리를 하므로 동일한 시간이라도 병렬처리하는 GPU의 수행속도가 훨씬 빠르다. 최신 CPU는 4코어, 8코어를 가지고 있지만, GPU는 수천 개의 코어를 가지고 있어 병렬로 처리했을 때 GPU는 동일한 시간에 수천 개의 코어로, 수천 개의 계산을 동시에 처리할 수 있다.

소프트웨어의 선택은 역시 하드웨어만큼 중요하다고 할 수 있다. 하드웨어는 사용하다가 더 좋은 성능이 필요하다면 업그레이드로 쉽게 해결할 수 있지만, 소프트웨어는 사용하는 라이브러리와 개발한 머신러닝 코드가 언어와 라이브러리에 종속되므로 신중한 선택이 필요하다.

과거에는 소프트웨어의 선택에서 속도라는 측면을 중요하게 여겼다. 엄청나게 많은 계산을 해야 하는 머신러닝은 실행시간이 길어서 오랜 시간 기다리는 것보다는 코드

의 결과를 되도록 빨리 보는 것이 의미가 있었기 때문이었다. 그래서 C, C++과 같은 언어를 많이 사용했다.

하지만 지금은 컴퓨팅 파워가 예전보다 많이 향상되었고, 단시간에 강력한 컴퓨팅 파워가 필요하다면 클라우드를 사용할 수도 있다. 결정적으로 GPU의 등장으로 속도의 매력이 이전보다는 상실되었다(이미지 인식 같은 분야는 아직도 많은 수행시간이 필요해 C++를 선호하기도 한다) 따라서 최근에는 라이브러리가 소프트웨어 선택에 중요한 이유가 되고 있다.

머신러닝 라이브러리는 필요한 머신러닝 알고리즘을 지원하고 데이터 처리를 위한 기능이 있는지, 업데이트가 지속해서 이뤄지는지 등을 고려해 선택하면 된다.

머신러닝 알고리즘을 직접 구현하는 것은 수학적 지식이 필요하고, 많은 시험과 최적화 등으로 검증해야 하는데, 새로운 머신러닝 알고리즘을 만들거나 기존의 머신러닝 알고리즘 자체를 개선할 목적이 아니라면 굳이 직접 개발할 필요는 없다.

머신러닝을 이용하는 데는 머신러닝 알고리즘에 대한 개념 이해와 사용법을 아는 것으로 충분하다. 머신러닝 프로그램 작성의 전체 프로세스를 생각해보면, 머신러닝 알고리즘 자체가 차지하는 비중과 시간은 10~20% 정도로 그다지 크지 않다. 데이터를 머신러닝에 사용할 수 있게 조작하는 전처리Pre-processing, 머신러닝의 결과를 분석하고 개선하기 위한 후처리Post-Processing 등의 과정이 오히려 더 중요하고 많은 시간이 필요하므로 사용하려는 라이브러리에서 이러한 기능들도 포함하고 있는지를 보는 것이 중요하다. 결국 전처리와 후처리에 의해 머신러닝 결과물의 품질이 결정되기 때문이다.

머신러닝 라이브러리를 선택할 때 GPU를 지원하는지 확인하는 것도 빠질 수 없는 점검사항이다. CPU와 GPU의 계산속도 차이는 작게는 수 배에서 크게는 수백, 수천 배 차이가 나기 때문에 머신러닝에 사용하려는 데이터의 크기가 크고 딥러닝 같은 계산량이 많은 알고리즘을 사용한다면 GPU는 더는 선택이 아닌 필수가 될 것이다.

이 책에서 알고리즘 트레이딩에 추천하는 하드웨어 환경과 소프트웨어를 정리하면 다음과 같다.

- **하드웨어** Intel i5 이상, 메모리 4GB, HDD 256GB 이상
- **OS** 파이썬을 사용할 수 있는 환경(Windows, OS X, Linux)
- **Database** MySQL, 주가 관련 데이터 저장용이다.
- **프로그래밍 언어** 파이썬, 파이썬은 강력한 언어로 웹 개발, 클라우드, 금융에 이르기까지 폭넓게 사용되고 있다. 언어가 간결하고 배우기 쉬우며 다양한 라이브러리를 가지고 있다.
- **라이브러리**

  NumPy: 고차원의 수학적 기능을 제공하는 오픈소스 파이썬 라이브러리다. NumPy의 핵심 기능은 ndarray인데, 이는 n 차원의 배열 데이터 클래스로 다차원배열을 유연하면서도 빠른 속도로 사용할 수 있다. NumPy는 각종 수학과 과학 연산에서 많이 사용되는 벡터나 스칼라로 활용할 수 있고, 데이터베이스와 연동해 사용할 수도 있다.

  SCIPY: 과학 연산에 필요한 기능을 제공하는 라이브러리로, 최적화, 선형대수, 적분, FFT 등의 기능을 제공한다.

  Pandas: 금융 데이터 처리용 라이브러리로, `DataFrame` 클래스를 이용해 시계열 금융 데이터 처리에 필요한 각종 기능이 있다.

  Matplotlib: 그래프를 그리거나 데이터를 시각화하는 데 필요한 풍부한 기능을 가지고 있는 라이브러리다. 그래프를 저장하거나 확대 또는 축소를 위한 간단한 UI도 제공한다.

  scikit-learn(sklearn): 파이썬 머신러닝 라이브러리로, 딥러닝을 제외한 현존하는 거의 모든 머신러닝 알고리즘이 구현되어 있고 데이터 처리와 머신러닝 학습결과를 분석하는 기능들도 포함되어 있다. 알고리즘과 관계없이 사용법이 일관되어 짧은 학습시간에 직관적으로 이용할 수 있는 파이썬의 대표적인 머신러닝 라이브러리다.

  Statsmodels: 파이썬 통계 라이브러리로 데이터 탐색, 통계적 모델 추정, 통계 테스트 등 다양한 통계 관련 기능을 지원한다.

Anaconda를 이용한 라이브러리 설치

Anaconda는 앞으로의 실습에 필요한 패키지들과 관련 프로그램을 모두 한 번에 설치해주는 프로그램으로, Windows, OS X, Linux를 지원한다. 파이썬에 익숙하지 않은 독자라면 파이썬 설치 프로그램인 pip 등을 이용해 필요한 라이브러리를 하나씩 설치하는 것이 어려울 수도 있다. 그래서 파이썬에 익숙하지 않은 독자라면 Anaconda를 이용해 매우 쉽게 필요한 실습 환경을 만들 수 있다.

Anaconda에 대한 자세한 설명은 홈페이지에서 확인할 수 있고 다운로드하려면 https://www.continuum.io/downloads 페이지를 방문하면 된다.

Anaconda 설치는 매우 간단해 다운로드한 파일을 실행하는 것이 전부다. Anaconda는 Python 2.7과 Python 3.5 버전 2가지를 제공하는데, Python 2.7 버전을 설치하기 바란다. 이 책에서 사용하는 라이브러리와 코드들은 모두 Python 2.7 버전에서 테스트했다.

Anaconda는 OS에 따라 설치되는 라이브러리에 차이가 있는데 Linux는 Anaconda에서 지원하는 라이브러리를 모두 설치할 수 있고, OS X는 상당 부분, Windows는 가장 적다. Anaconda에서 설치하는 모든 라이브러리 리스트를 보려면 http://docs.continuum.io/anaconda/pkg-docs 페이지를 참조하기 바란다.

예제 코드 다운로드

- http://www.hanbit.co.kr/exam/2803

차례

Part 1

머신러닝이 무엇이고 머신러닝이 할 수 있는 것과 머신러닝의 종류는 무엇이 있는지 등 머신러닝의 전반적인 개요를 설명한다.

chapter 1

머신러닝

1.1 머신러닝이란 무엇인가

머신러닝Machine Learning은 우리의 통념보다 오래된 기술이고, 이미 우리가 사용하는 많은 서비스와 상품에 직·간접적으로 활용되고 있다. 매일 쏟아지는 스팸메일로부터 우리를 보호하는 스팸메일 필터링 기술, 전자상거래 사이트에서 적합한 물건을 추천해주는 서비스, 애플의 시리와 같이 사람의 음성을 듣고 이를 인식하는 서비스 등 우리 곁에는 많은 서비스가 자리를 잡고 있다.

위키에서는 머신러닝[01]을 다음과 같이 정의하고 있다.

> *머신러닝은 인공지능의 패턴인식과 계산학습 이론에서 발전한 컴퓨터과학의 한 분야다. 머신러닝은 데이터로부터 학습하고, 예측할 수 있는 알고리즘을 연구한다. 이러한 알고리즘은 정적으로 주어진 프로그램이 아닌 입력된 데이터로부터 모델을 만들어 예측이나 결정을 내린다.*

여기서 알 수 있듯이 머신러닝의 정의에서 제일 중요한 부분은 주어진 데이터를 이용해 스스로 학습하고 적합한 모델을 만든다는 점이다. 일반적으로 컴퓨터로 어떤 일을 하게 하려면 사람이 컴퓨터에 소상히 Input(데이터는 무엇인지), Program(이런 데이터가 들어오면 어떻게 처리해야 하는지), Output(결과를 어떻게 표현해야 하는지) 등을 모두 정의해야 한다.

---

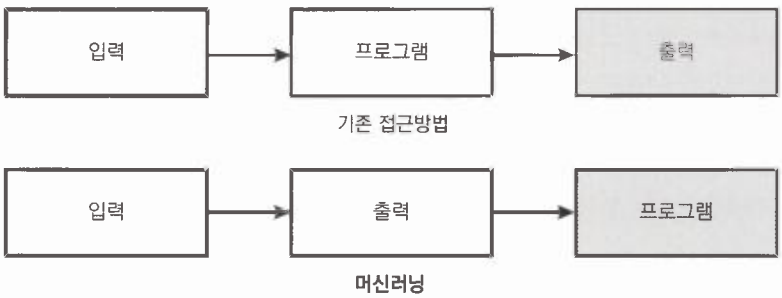
01 출처: https://en.wikipedia.org/wiki/Machine_learning

컴퓨터가 이해할 수 있도록 정의하는 것이 프로그래머가 프로그램을 개발하는 과정이며 Input(입력), Program(프로그램), Output(출력)을 어떻게 처리해야 하는지 명령어를 이용해 코딩한다. 컴퓨터는 지능이 없으므로 입력, 프로그램, 출력에 대한 상세한 내용을 논리적 모순 없이 프로그램으로 작성해야만 원하는 결과를 얻을 수 있다.

하지만 머신러닝에서는 전혀 다른 접근방법을 적용한다. 사람은 입력과 출력만 제공하고, 프로그램은 머신러닝 스스로 만들게 하는 것이다. 적절한 데이터가 준비되어 있다면 기존의 접근방법에서는 프로그램을 작성하는 데 상당한 노력과 시간을 보내야 하지만, 머신러닝 접근방법에서는 이러한 수고가 필요 없다. 원하는 결과를 머신러닝에 출력으로 지정하면 나머지 작업은 머신러닝에 의해 스스로 프로그램을 작성하기 때문에 사람은 충분한 양의 잘 정돈된 데이터와 머신러닝을 적용하는 데 필요한 컴퓨팅 파워만 제공해주면 된다.[02]

다음은 머신러닝의 접근방법과 기존 접근방법을 보여주는 그림으로, 각 접근방법이 어떻게 다른지를 확실히 보여준다.

그림 1-1 기존 접근방법과 머신러닝 접근방법 비교



머신러닝은 데이터에 기반을 두기 때문에 계산 통계Computational Statistics와 많은 연관이 있다. 데이터로부터 무엇을 학습한다는 것은 머신러닝 관점에서 보면, 주어

02 여기서 프로그램을 작성한다는 것은 스스로 모델을 만들고 적절한 파라미터를 설정하는 것을 의미한다.

진 데이터를 이용해 확률을 계산하고 특정 데이터를 줬을 때 이미 과거의 데이터를 이용해 계산된 확률로 결과치를 만들기 때문이다.

이러한 특성은 머신러닝에서 데이터가 얼마나 중요한지를 강력하게 보여준다. 아무리 좋은 머신러닝 알고리즘이라 하더라도 제공되는 데이터의 양이 적절하지 않거나 품질이 떨어진다면 결코 좋은 결과를 기대할 수 없다. 그 유명한 'Garbage in, Garbage Out' 원칙은 머신러닝에도 당연히 적용된다.

데이터가 머신러닝 결과물에 막대한 영향을 미치기 때문에 머신러닝에서는 데이터를 탐색하고 정리하는 데이터 마이닝Data Mining이 아주 중요하다.[03] 데이터 마이닝을 통해 입력 데이터로 사용할 적합한 입력변수를 선택하고, 이 입력변수에 빠진 데이터를 보충하거나 이상치Outlier를 제거하고, 적절한 양의 데이터를 선택하는 과정이 사실상 머신러닝에서 제일 중요한 과정이어서 그 중요성은 아무리 강조해도 지나치지 않다.

1.2 머신러닝의 장단점

머신러닝에 대한 개념이나 각종 성공사례를 처음 접하면 머신러닝이 마치 천하의 보검처럼 느껴 져 어떠한 문제라도 머신러닝으로 해결할 수 있을 것 같은 느낌을 받는다. 그리고 예제로 제공되는 코드들을 실행해보면 그리 길지 않은 코드로 신기하게 그림에서 문자를 인식하고 아이리스Iris 꽃의 종류를 분별하는 등, 기존 프로그래밍 방법으로는 막막한 것들을 너무도 쉽게 해내 놀랍기 그지없을 것이다.

하지만 머신러닝은 한 번 휘두르면 모든 적을 제압할 수 있는 천하의 보검이 아니라는 것을 분명히 알고, 역시 예제는 예제라는 만고의 진리를 깨닫게 되는 데는 그리 오랜 시간이 걸리지 않을 것이다. 이러한 결과는 머신러닝에 사용한 데이터가

03 머신러닝과 데이터 마이닝은 서로 유사한 점이 많아서 혼용되는 경우도 있지만, 이 책에서 데이터 마이닝은 탐색적 데이터 분석에 중점을 둔 것으로 설명한다.

잘못되었거나 엉뚱한 알고리즘을 적용하였거나 머신러닝에 적합하지 않은 문제 때문에 초래되었을 수 있다.

머신러닝을 제대로 사용하려면 머신러닝의 장단점을 파악해 해결하려는 문제가 머신러닝에 적합한 것인지, 적합하지 않은 것이라면 어떻게 문제를 재정의해야 하는지, 어떤 데이터를 사용해야 하는지 등을 고민해야 한다. 즉, 머신러닝의 장단점을 이해해야만 머신러닝으로 좋은 결과를 기대할 수 있다.

머신러닝의 장점

- 학습을 위한 지식 표현이 필요 없다. 컴퓨터가 지식을 이해하게 하기 위한 표현은 어려운 문제다.
- 충분한 데이터와 적합한 알고리즘을 사용한다면 사람이 만든 모델보다 좋은 결과를 보여줄 수 있다.
- 고도의 수학적 지식이나 프로그래밍 능력을 요구하지 않는다. 기본적인 지식과 능력으로도 충분히 머신러닝을 이용할 수 있다.
- 자동화가 가능하다. 프로그램으로 머신러닝을 학습시키고 최적의 파라미터를 찾아 그 결과에 대한 평가 등을 자동화하여 진행할 수 있다.
- 저렴하고 유연하다. 데이터를 제외한 나머지 과정은 자동화가 가능하므로 저렴하게 실행할 수 있다.
- 프로그램을 이용해 자신이 원하는 대로 사용할 수 있다.

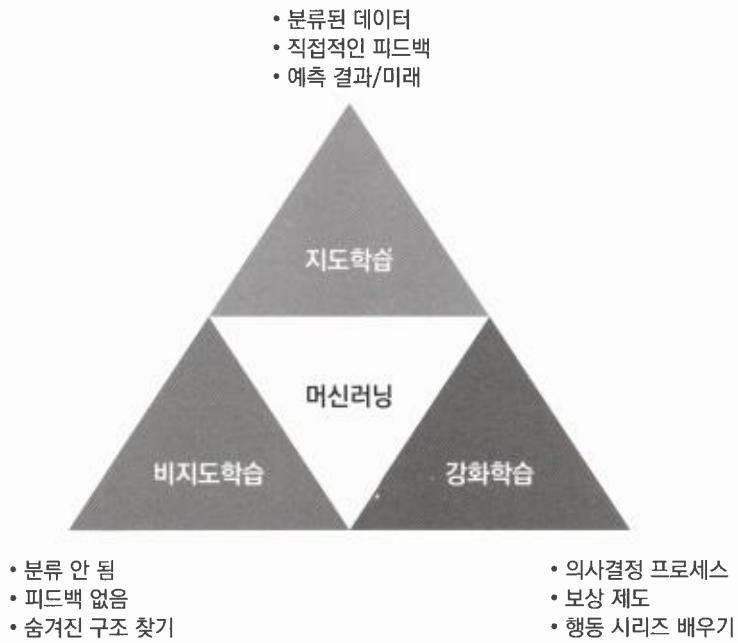
머신러닝의 단점

- 데이터 준비에 많은 노력이 든다. 지도학습Supervised Learning의 경우 모든 개별 데이터에 결과치를 만들어 주어야 한다.
- 오류 발생이 쉽다(Error prone). 일반적으로 정확도가 높은 모델을 만들기 어렵다.
- 생성된 모델이 블랙박스Black Box이기 때문에 이를 해석하기가 어렵다. 정확도를 높이려면 모델을 수정하거나 개선할 수 있어야 하는데, 대부분의 머신러닝 알고리즘은 학습결과로 생성된 모델을 이해하기 어렵고 모델 자체를 개선할 수 없다.
- 과적합Overfitting 문제가 종종 발생한다. 주어진 데이터에 너무 최적화되어 학습에 사용된 데이터에는 높은 예측력Predictve Power을 보여주지만, 그렇지 않은 데이터에는 좋지 않은 예측력을 보여준다.

1.3 머신러닝의 종류

머신러닝은 학습방법에 따라 크게 3가지 종류로 나눌 수 있다. 사람이 입력과 출력을 모두 제공하는 지도학습, 입력만 제공하는 비지도학습Unsupervised Learning, 어떤 환경에서 특정 목표를 달성하기 위해 스스로 학습하는 강화학습Reinforcement Learning이다. 현시점에서 활용빈도로 보면 지도학습이 가장 많고, 그 다음이 비지도학습, 마지막은 강화학습이라고 할 수 있다.

그림 1-2 머신러닝의 종류

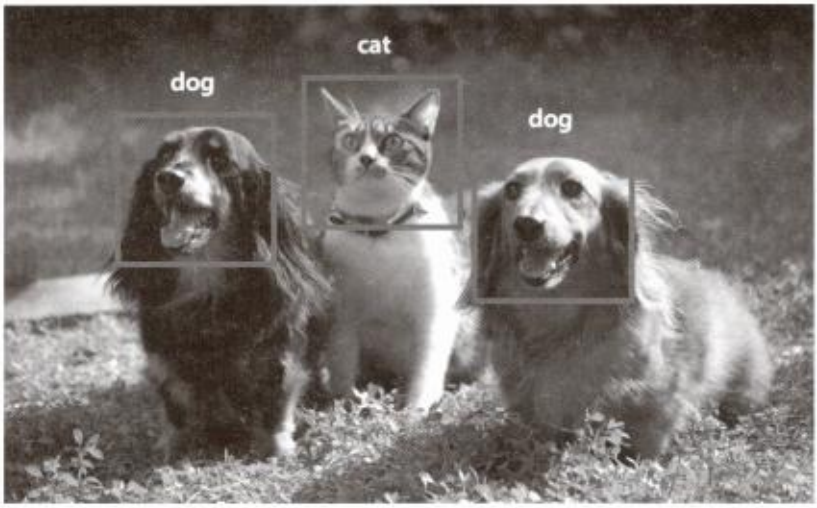


1.3.1 지도학습

지도학습은 가장 많이 활용되는 머신러닝의 종류로 스팸메일 필터링, OCR 문자 인식 등이 이에 해당한다. 지도학습은 머신러닝에 입력과 출력을 모두 제공해 학습하게 하는 것으로, 일종의 최적화Optimization 문제로 생각할 수 있다. 이는 지도학

습 알고리즘이 주어진 입력값을 분석해 출력값을 만들어내기 위한 최적의 모델을 만들기 때문이다. 예를 들어, 고양이의 이미지를 구분할 수 있는 머신러닝 프로그램을 개발한다고 하자. 지도학습은 입력과 출력을 모두 제공해야 하므로 고양이 사진과 함께 고양이라는 것을 알려주어야 한다. 고양이가 있는 사진에 '고양이'라는 출력을 같이 제공해야 한다.

그림 1-3 지도학습 예시



따라서 지도학습에서 하나의 데이터는 입력과 출력이 같이 묶여있는 튜플Tuple 형태로 이뤄진다.

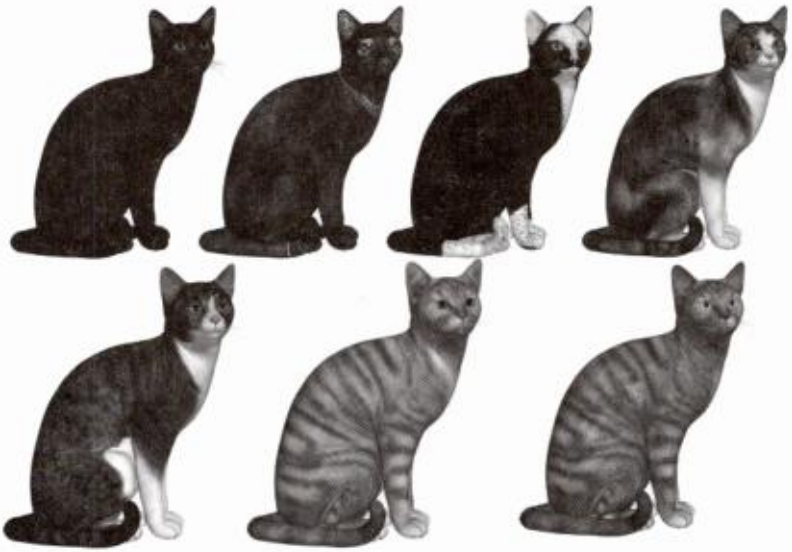
1.3.2 비지도학습

앞에서 설명한 지도학습이 입력을 분석해 출력을 만들어내는 최적화 문제라면 비지도학습은 입력 데이터의 구조를 파악하거나 관계를 분석하는 방법이라고 할 수 있다. 비지도학습은 '지식 발견Knowledge Discovery'라고도 하는데, 이는 학습결과로 생각하지 못한 지식을 발견하거나 입력 데이터 간의 그룹 또는 특성 등을 발견할 수 있어서다.

지도학습과 구분되는 또 하나의 특징은 학습결과에 대한 평가가 힘들다는 점이다. 이는 학습결과가 명확한 목적, 즉 출력이 없어서 평가기준을 잡을 수 없기 때문이다. 지도학습에서 데이터를 제공할 때 하나의 데이터마다 입력과 출력이 튜플로 제공되지만, 비지도학습에서는 출력 없이 입력만 제공된다.

앞의 지도학습에서는 고양이 사진과 이름이 주어졌지만, 다음 그림처럼 비지도학습에서는 이름 없이 고양이 사진만으로 학습이 이루어진다.

그림 1-4 비지도학습 예시



1.4 머신러닝이 할 수 있는 것

자동차 자율주행, 고양이 사진인식, 이미지 자막 넣기Image Captioning 등 최근 머신러닝이 보여주는 엄청난 성과를 보면 머신러닝이 고도의 수학적 배경과 정교한 알고리즘으로 무장해 내가 원하는 많은 문제를 해결할 것으로 믿을 수도 있다. 하지만 아직까지는 머신러닝이라는 마법으로 해결할 수 있는 문제가 많지 않으며, 그마저도 사람의 따뜻한 손길과 많은 헌신을 대가로 요구한다.

머신러닝을 적용해 무언가에 대한 결과를 얻는 것은 생각보다 많은 시간이 필요하다. 하나의 완성된 결과물을 얻으려면 수십 번, 수백 번 반복을 통해 조금씩 개선해가고, 때에 따라서는 모델을 새롭게 만들거나 전혀 새로운 시각으로 접근해야 한다. 더욱이 머신러닝으로 특정 문제를 해결하려면 그 특정 문제를 머신러닝에 적합한 형태로 바꿔야 원하는 결과를 얻을 수 있다. 따라서 머신러닝을 제대로 활용하려면 머신러닝이 잘 할 수 있는 것과 그렇지 않은 것을 아는 것이 중요하다.

스팸메일을 필터링하고, 글자와 음성을 인식하는 머신러닝을 보면 다양한 일을 처리할 수 있을 것이라 생각하겠지만, 크게 3가지 종류의 일을 처리할 수 있다. 변수 간의 관계를 파악하는 회귀Regression, 데이터를 분류하는 분류Classification, 데이터를 연관있는 것끼리 묶어주는 군집화Clustering다.

머신러닝은 이 3가지를 이용해 다양한 문제를 해결한다. 회귀와 분류는 모든 머신러닝 알고리즘의 기본이 되는 중요한 개념이므로 반드시 이해해야 한다.

1.4.1 회귀

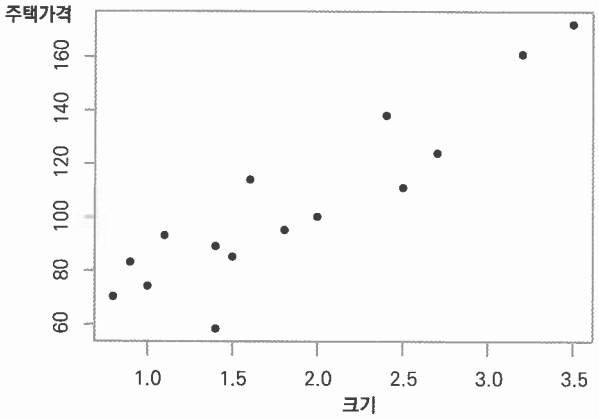
회귀는 연속적인 숫자Continuous Number 변수들 간의 상관관계를 파악하는 것으로, 특히 종속변수와 독립변수Independent Variable 사이의 연관성을 분석하는 것을 주된 목적으로 한다. 회귀에 대한 이해를 돕기 위해 주택가격에 대한 한 가지 예를 들어 설명하겠다.

살고 있는 집을 팔고 싶은데 도대체 얼마를 받아야 하는지를 알고 싶다고 가정하자. 그런데 팔려는 집과 동일한 크기의 주택은 사례가 없어서 주택가격을 얼마로 결정해야 하는지 애매하다면 다른 크기의 주택가격 데이터와 회귀 분석을 통해 합리적인 주택가격을 산정할 수 있다.

회귀 분석을 시작할 때 가장 먼저 해야 할 일은 데이터의 상관정도를 직관적으로 파악하게 종속변수과 독립변수의 산점도Scatter Plot를 그리는 것이다.

그림 1-5 주택가격과 크기의 산점도

크기	주택가격
0.80	70
0.90	83
1.00	74
1.10	93
1.40	89
1.40	58
1.50	85
1.60	114
1.80	95
2.00	100
2.40	138
2.50	111
2.70	124
3.20	172
3.50	172



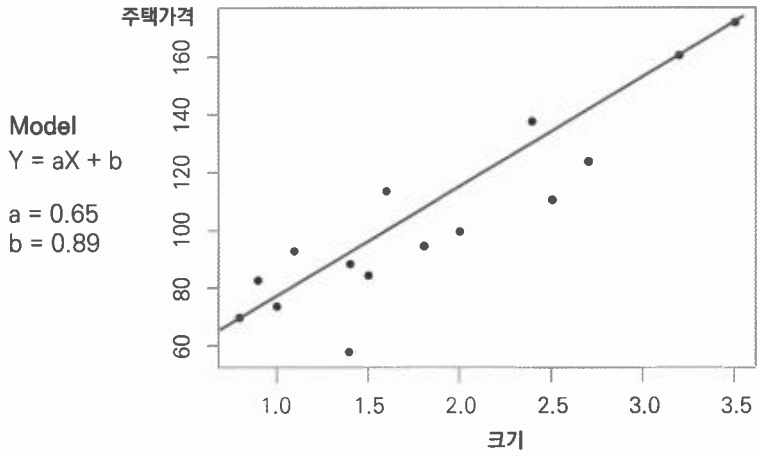
[그림 1-5]에서 알수 있듯이 주택가격과 크기는 일정한 관계, 즉 선형적인 관계가 있음을 쉽게 파악할 수 있다. 해결하고 싶은 문제는 크기에 따른 주택가격인데, 산점도를 보면 크기가 클수록 가격이 높아지는 선형적인 관계가 있으므로 다음과 같은 모델(수식)을 떠올릴 수 있다.

$$Y = aX + b$$

Y는 주택가격으로 '종속변수'라 하고, X는 크기로 '독립변수'라고 한다. 가정한 모델이 1차함수이므로 'a'는 기울기, 'b'는 절편을 의미한다. 주어진 데이터를 이용해 a 값과 b 값을 찾게 된다면 주택가격 결정을 위한 모델을 완성할 수 있고, 완성된 모델의 X에 팔려는 집의 크기를 입력하면 주택가격을 알 수 있다. 프로그램을 이용해 a의 값이 0.65, b의 값이 0.89일 때 앞서 보았던 산점도를 가장 잘 표현할 수 있다면 '주택가격 = 0.65×크기 + 0.89'라는 관계가 성립한다고 할 수 있다.

이제 주택가격과 크기의 관계를 알았으니 크기만 알면 앞의 식에 넣어 적절한 주택가격이 얼마인지를 산정할 수 있게 된다.

그림 1-6 회귀 분석



이처럼 주어진 변수 간의 상관관계를 파악하는 것이 회귀다. 회귀 문제의 예는 다음과 같다.

- 과거의 온도 데이터를 이용해 내일 온도를 예측한다.
- 주식시세 정보를 이용해 미래의 주식가격을 예측한다.
- 유동인구, 날씨, 가격정보 등을 이용해 음식점의 매출을 예측한다.
- 구매자의 나이와 연 소득을 이용해 특정 제품의 판매량을 예측한다.

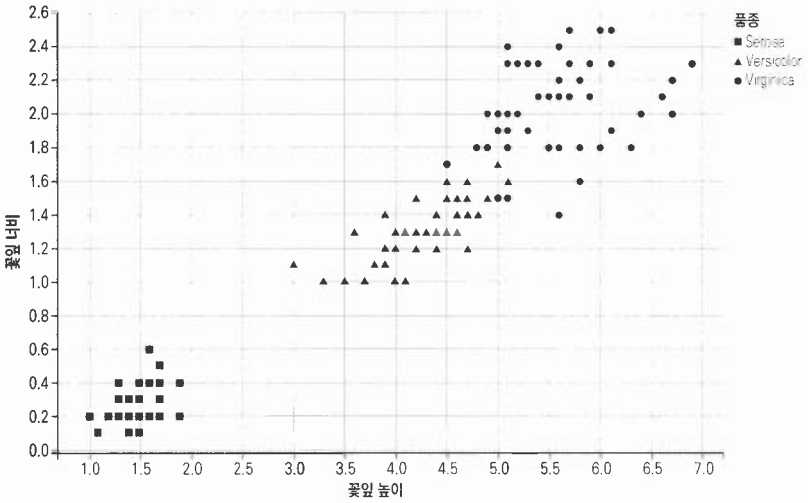
1.4.2 분류

분류는 단어가 의미하는 대로 데이터를 나누는 것이다. 분류에 대한 이해를 돕기 위해 아이리스의 예를 들어보겠다.

꽃잎의 너비와 높이 데이터를 이용해 주어진 아이리스가 Setosa, Virginica, Versicolor 3종류 중 어떤 품종에 속하는지를 판별해야 한다고 가정해보자. 이 문제는 앞의 회귀와는 다르게 어떤 값을 예측하는 것이 아니고 어떤 종류에 속하는지를 파악하는 문제다. 회귀 문제와 마찬가지로 분류 역시 산점도를 그려 아이리스 품종별로 꽃잎의 너비와 높이가 어떤 관계가 있는지를 파악해야 한다.

[그림 1-7]을 보면 Setosa 품종은 화면에 네모로 왼쪽 아래에 위치하고, Virginica는 원으로 오른쪽 위에 위치하며, 그 중간에 삼각형으로 Versicolor 품종이 있음을 알 수 있다. 해결하려는 문제는 주어진 꽃잎의 너비와 높이 데이터만으로 어떤 품종인지를 파악하는 것이므로 3가지 품종을 구분하는 방법이 필요하다. 만약 어떤 모델이 있고 이 모델을 이용해 꽃잎의 너비와 높이에 따라 품종을 구분할 수 있게 된다면 새로운 데이터로 너비와 높이 데이터만을 입력해도 품종을 구분할 수 있다.

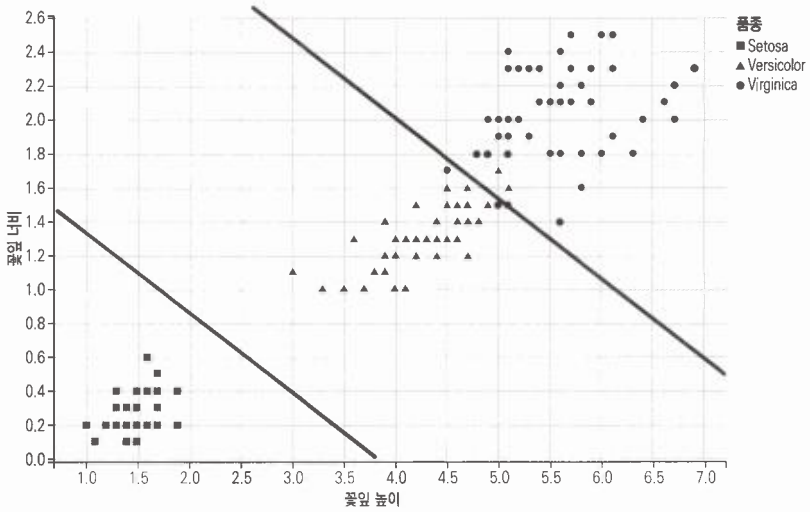
그림 1-7 아이리스의 산점도[04]



즉, [그림 1-8]처럼 2개의 선을 이용해 Setosa, Virginica, Versicolor로 영역을 나눈 다음에 새롭게 주어진 데이터가 3개의 영역 중 어디에 위치하는지를 안다면 자연스럽게 아이리스 품종을 분류할 수 있다. 따라서 분류는 주어진 데이터들을 이용해 아이리스를 잘 구분 지을 수 있는 2개의 'Y=aX + b'를 찾는 것이라 할 수 있다.

04 출처: http://blog.datacamp.com/machine-learning-in-r/

그림 1-8 최적 경계선



분류는 이와 같은 과정을 거쳐 데이터를 구분할 수 있게 하는 것으로 머신러닝에서 매우 광범위하게 사용되고 있다. 회귀는 연속적인 데이터Continuous Data에 적용할 수 있지만, 분류는 범주형 데이터Categorical Data에 적용할 수 있다.

분류 문제의 예는 다음과 같다.

- 스팸메일 분류
- 이미지 인식
- 음성 인식
- 질병 발생 여부 판별

1.4.3 군집화

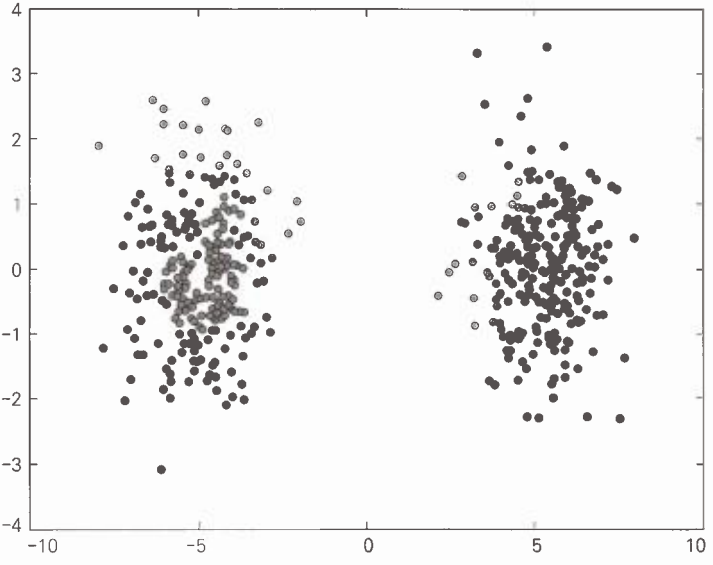
군집화는 데이터를 유사한 특성을 가진 무리Cluster로 묶는 것을 의미한다. 군집화는 비지도학습에서 사용하는 것으로 출력 데이터 없이 입력 데이터만으로 이뤄지며, 일반적으로 데이터의 특성을 파악하거나 이해하기 위해 많이 적용된다.

예를 들어, 마케팅 캠페인을 진행하는데 어떤 특성을 가진 사람들이 마케팅 캠페인에 반응하는지를 알고 싶다고 가정해보자. 마케팅 캠페인을 처음 시행해서 연관성 있는 데이터는 가지고 있지만, 어떤 기준으로 대상자를 선정하는 것이 좋을지 모른다면 이러한 문제를 해결하는 데 군집화가 효과적으로 적용될 수 있다.

군집화는 주어진 데이터들의 유사도를 계산해 비슷한 특성이 있다고 판단되는 것끼리 군집으로 분류하는 것으로, 이러한 작업을 효율적으로 할 수 있다. 마케팅에 반응한 사람들의 데이터를 모아 군집화를 실시하면 유사한 특성을 가진 사람들을 묶어 몇 개의 군집인지 알 수 있고, 각각의 군집에 속한 사람들의 공통점을 파악하면 원래의 문제인 마케팅에 반응하는 사람들의 유형과 그들의 특성을 발견할 수 있다.

군집화를 실시해 다음 그림과 같은 결과를 얻었다면 잠정적으로 마케팅 캠페인에 반응하는 사람의 군집은 2개가 되므로 이들 군집에 속한 사람들의 특성을 분석하면 된다.

그림 1-9 군집화 결과



군집화 문제의 예는 다음과 같다.

- 유사한 음악 취향을 가진 사용자를 묶는다.
- 천문학 데이터를 이용해 유사한 특성을 가진 별을 찾는다.
- 전자상거래 사용자가 좋아할 만한 물건을 추천한다.

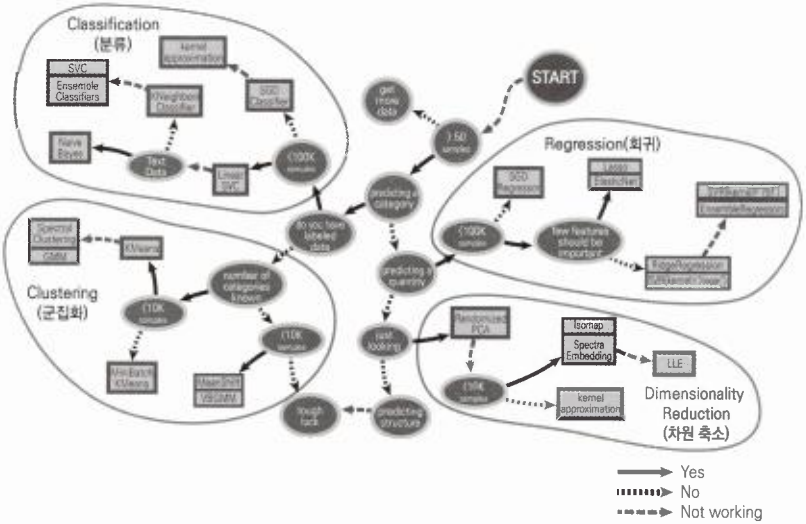
1.5 머신러닝 알고리즘

머신러닝을 실제로 적용하려면 머신러닝에 사용할 알고리즘을 선택해야 한다. 앞에서 설명한 바와 같이 머신러닝은 회귀, 분류, 군집화 3가지 종류의 문제를 해결할 수 있는데, 이들 문제에 적합한 각각의 알고리즘이 있다. 한 가지 알고리즘으로 모든 종류의 문제를 해결할 수 없고, 알고리즘마다 특성이 달라서 원하는 것이 무엇인지를 명확하게 하고, 그 결과를 얻기 위해서는 어떤 종류의 문제에 속하는지를 판별한 다음 머신러닝에 적용할 알고리즘을 선택해야 한다.

적합한 알고리즘을 선택하기 위해서는 데이터의 크기와 데이터 타입(문자, 숫자, 이미지 등), 데이터가 선형인지 비선형인지 등의 요소를 고려해야 한다. 데이터에 대한 사전지식 즉, 도메인 지식Domain Knowledge이 별로 없다면 최적의 결과를 보여주는 알고리즘을 선택하는 데 많은 시간과 노력이 필요할 수도 있다.

다음 그림은 파이썬의 유명한 머신러닝 라이브러리인 Scikit-learn에서 제공하는 알고리즘 Cheat- sheet로 머신러닝을 이용해 원하는 결과를 얻기 위한 일반적인 절차를 담고 있다. 'Start'부터 시작해 표시된 질문에 대한 응답으로 다음 과정이 차례로 제시되어 초심자들에게 어떤 머신러닝 알고리즘을 처음에 적용하고, 결과가 좋지 않은 경우 다음에 어떤 알고리즘을 적용해야 하는지를 알기 쉽게 표현하고 있다.

그림 1-10 Scikit-learn의 알고리즘 Cheat-sheet[05]



회귀, 분류, 군집화에 적용할 수 있는 대표적인 알고리즘은 다음과 같다.

회귀

- 선형 회귀Linear Regression
- SGD 회귀 Stochastic Gradient Descent Regression
- 서포트 벡터 회귀SVR, Support Vector Regression
- 랜덤 포레스트 회귀Random Forest Regression
- 베이지안 회귀Bayesian Regression
- 등위 회귀Isotonic Regression
- 베이지안 ARD 회귀Bayesian Automatic Relevance Determination Regression

분류

- 로지스틱 회귀Logistic Regression
- 서포트 벡터 머신SVM, Support Vector Machine

---

05 출처: http://scikit-learn.org/stable/tutorial/machine_learning_map/

- 랜덤 포레스트Random Forest
- 의사결정 트리Decision Tree
- 그레이디언트 부스팅 트리GBT, Gradient Boosting Tree
- SGD 분류기SGD Classifier
- AdaBoost

군집화

- K-평균(최적분리)K-means
- 스펙트럼 군집화Spectral Clustering
- 가우시안 혼합Gaussian Mixtures
- 병합식 군집화Agglomerative Clustering
- 친근도 전파Affinity Propagation
- 평균 이동Mean Shift

이외에도 많은 종류의 알고리즘이 있으므로 자신이 다루는 데이터 특성에 적합한 알고리즘을 선택해야 좋은 결과를 빨리 얻을 수 있다.

머신러닝 알고리즘의 수학적 이론과 그 내용을 아는 것은 특별한 경우가 아니라면 충분조건이지 필요조건은 아니다. 예측을 위한 모델링은 사람이 직접 하는 것이 아니라 머신러닝 알고리즘 스스로 만들어가도록 설계되어 있기 때문이다. 따라서 해당 머신러닝 알고리즘을 사용하는 개발자는 머신러닝 알고리즘의 개념과 사용법, 알고리즘 특성을 아는 것이 중요하다.

머신러닝으로 해결하려는 문제에 효과가 있다고 알려진 알고리즘이 없다면, 적용할 수 있는 모든 머신러닝 알고리즘을 하나씩 적용해 시행착오를 거쳐 가장 좋은 성능을 보여주는 것을 채택해 사용한다.

1.6 머신러닝 프로세스

머신러닝을 적용하는 프로세스는 머신러닝에서 데이터 다음으로 중요하다. 일반

적으로 머신러닝을 적용해 원하는 결과를 얻는 데는 많은 시간과 노력이 필요하다. 한두 번의 시도가 아니라 수십, 수백 번의 시행착오 끝에 모델을 완성하기 때문에 머신러닝을 행하는 절차, 즉, 머신러닝 프로세스가 결과물의 품질에 절대적인 영향을 미친다. 잘못된 머신러닝 프로세스는 작업의 효율성을 심각하게 저해하며, 올바른 방향으로 나아가지 못하게 할 수도 있다. 실제로 머신러닝을 이용하기 시작하고 경험이 쌓이기 시작하면 머신러닝 알고리즘과 학습은 전체 프로세스에서 많은 비중을 차지하는 부분이 아니라는 것을 알게 되고, 정작 중요한 것은 그 외의 것이라는 것을 뼈저리게 느끼게 될 것이다.

다음 그림은 머신러닝 프로세스를 도식화한 것이다.

그림 1-11 머신러닝 프로세스

각 과정은 다음과 같다.

첫 번째 전처리(1st Pre-Processing)

- 머신러닝 학습에 사용될 데이터인 학습용 데이터셋Training Dataset을 준비하는 과정이다.
- 학습에 사용할 데이터를 선별하고 빠진 데이터가 있는지를 확인하며, 데이터에 왜곡을 줄 수 있는 이상치Outlier 데이터를 처리한다. 필요에 따라서는 정규화Normalization 작업 등을 한다.

학습용 데이터셋(Training Dataset)

- 학습에 사용할 데이터를 만드는 과정으로, 지도학습이라면 입력과 출력, 비지도학습이라면 입력 데이터를 정해진 형태로 생성한다.

두 번째 전처리(2nd Pre-Processing)

- 학습용 데이터셋에 포함된 변수 중에 어떤 것을 학습에 사용할지를 결정하는데, 이 과정을 '피처 선택Feature Selection'이라고 한다. 학습에 지대한 영향을 미치는 중요한 과정이다.
- 데이터의 성격이나 적용하려는 알고리즘에 따라 피처 스케일링Feature Scaling같이 데이터를 일정 범위로 변환하는 작업도 이 과정에서 이뤄진다.
- 이 과정의 목적은 적용하려는 알고리즘에 더 쉽고 우수한 학습이 이뤄지도록 데이터를 만드는 것이다.

머신러닝 알고리즘 학습(Learning Algorithm Training)

- 선택한 머신러닝 알고리즘과 준비된 학습용 데이터셋을 이용해 학습해보는 과정으로, 프로그램을 통해 진행되며 사람의 개입은 거의 없다.

파라미터 최적화(Parameter Optimization)

- 선택한 머신러닝 알고리즘에서 적용 가능한 파라미터 값들을 조정해 결과물의 품질을 높이는 과정이다.
- 사람이 조정한 파라미터 값에 따라 학습결과의 품질을 파악해 최적의 파라미터를 찾거나 '격자 탐색Grid Search'을 이용해 머신러닝 프로그램에서 스스로 최적의 값을 찾도록 한다.

후처리(Post-Processing)

- 이 과정은 학습결과의 품질평가가 주된 목적이다.
- 대부분 머신러닝에 한 가지 알고리즘이나 모델만을 적용하지 않으므로 다수의 모델과 알고리즘 중 어떤 것이 가장 좋은 결과를 가져오는지, 각 알고리즘이 주어진 문제에 어떻게 반응하는지를 평가한다.

최종모델(Final Model)

- 최종으로 완성된 모델을 이용해 학습용 데이터셋이 아닌 실제 데이터에 적용한다.

[그림 1-11]에서도 알 수 있듯이 '파라미터 최적화'와 '후처리' 과정은 '첫 번째 전처리'나 '두 번째 전처리' 과정으로 다시 돌아가 원하는 결과가 나올 때까지 반복한다.

머신러닝을 이용한 문제 해결은 기본적으로 블랙박스를 사용하는 것과 비슷하다. 주어진 데이터의 종속변수와 독립변수의 상호관계를 모른 상태에서 이들 간의 관

계를 전적으로 머신러닝 알고리즘에 맡겨 상호관계를 파악하게 하는 것이어서 개발자로서는 그 결과만을 알 수 있을 뿐 그 과정은 알기 어렵다. 따라서 개발자가 해야 하는 것은 데이터를 가공하거나 파라미터를 조정하거나 피처 선택 등을 통해 학습결과물의 품질이 원하는 수준에 도달했는지 아닌지만을 파악할 수 있고, 모델이 완성되는 과정은 알 수도 없고 개입할 수도 없다. 그래서 머신러닝 프로세스를 지속해서 실행하며 시행착오를 통한 끊임없는 반복과 개선이 필요하다.

1.7 No free Lunch Theorem

'공짜 점심은 없다'는 의미의 'No Free Lunch Theorem'은 데이비드 울퍼트David Wolpert와 윌리엄 머크리디William Macready가 1997년에 발표한 「No Free Lunch Theorems for Optimization」[06]이라는 논문에 실린 것으로 머신러닝의 활용에 중요한 시사점을 제공하고 있다.

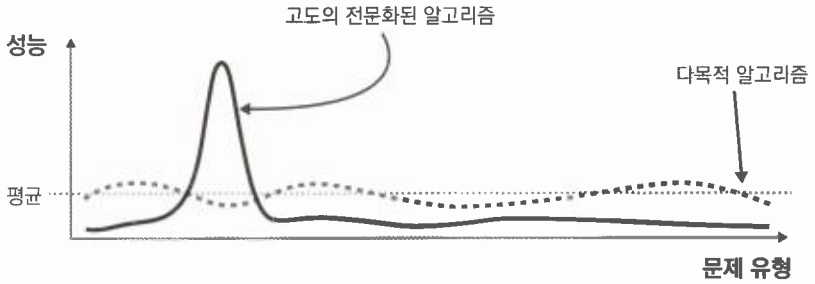
논문에서는 다음과 같이 적고 있다.

> *We have dubbed the associated results "No Free Lunch" theorems because they demonstrate that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems. ('No Free Lunch Theorem'은 간단히 말해 특정한 문제에 최적화된 알고리즘은 다른 문제에서는 그렇지 않다는 것을 수학적으로 증명한 정리다.)*

다음 그림은 이 내용을 표현한 것으로, 'No Free Lunch Theorem'이 무엇을 말하는지를 직관적으로 쉽게 이해할 수 있게 도와준다.

06 출처: Wolpert, D.H., Macready, W.G. (1997), No Free Lunch Theorems for Optimization, IEEE Transactions on Evolutionary Computation 1, 67. http://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf

그림 1-12 No free Lunch Theorem[07]



최적화라는 단어가 의미하는 것처럼 특정한 문제에 최적화되어 있다면 다른 문제에서는 좋은 결과를 보여줄 수 없는 것은 상식적으로 생각해도 당연하다. 하지만 머신러닝을 적용하는 데 있어 종종 범하는 실수는 하나의 머신러닝 프로그램에서 너무 많은 것을 바란다는 것이다.

예를 들어, 아이리스의 3품종을 분류하도록 최적화한 머신러닝 모델에 5품종을 포함한 데이터를 넣고 제대로 분류하기를 바라는 것은 'No Free Lunch Theorem'에서 강조하는대로 헛된 기대다. 3품종에 최적화된 모델이지 5품종에 최적화된 모델은 아니기 때문이다. 이런 경우에는 문제를 재정의해 5품종의 아이리스를 분류하도록 모델을 최적화하는 것이 올바른 접근이다.

머신러닝은 마법이 아니다. 학습용 데이터셋에서 종속변수와 독립변수의 관계를 분석하고 이를 기반으로 모델을 만드는 것이므로 철저하게 학습용 데이터셋에 종속된다. 따라서 언제나 그렇듯이 해결하려는 문제를 잘 정의하고, 연관된 데이터를 잘 정제하고, 머신러닝을 적용해 최적화해야 한다.

07 출처: http://ecmendenhall.blogspot.kr/2006_02_01_archive.html

Part 2

Part 2에서는 알고리즘 트레이딩을 위한 수학적 배경지식으로 통계와 시계열을 다룬다. 알고리즘 트레이딩을 하려면 주식의 매도와 매수를 결정하는 '모델'을 만들어야 하는데, 이 모델을 만들기 위한 최소한의 통계와 시계열 개념을 설명한다.

chapter 2

통계

2장에서는 통계 관련 주요 개념을 설명하고 직접 코드로 확인해 본다. 또한, 이를 위해 필요한 환경과 코딩에 사용할 데이터를 받는 방법을 다룬 후 실제로 코드를 작성해보겠다.

2.1 통계란

통계학은 데이터를 구성하고 요약하고 해석하는 데 필요한 수학적 이론과 방법들을 지칭하는 것이라 할 수 있다. 데이터의 크기가 일정 수준을 넘어가면 데이터의 성격, 특성 등을 이해하는 데 어려움을 겪게 된다. 이는 단순히 주어진 데이터의 값을 보는 것만으로는 전체 데이터의 모습이나 구조를 파악할 수 없기 때문이다. 통계학에는 이러한 평균, 중앙값, 최대최솟값, 분산, 표준편차, 분포 등 데이터의 모습과 특성을 파악하고 분석하며 표현하기 위한 여러 가지 이론과 도구가 있다.

통계학에서 데이터를 분석하는 데 사용하는 방법으로는 기술통계와 추론통계 2가지가 있다.

기술통계Descriptive Statistics는 데이터의 전체적인 모습을 파악하는 데 유용한 기법으로, 모든 데이터를 보지 않고 평균값, 최대최솟값, 표준편차 등의 값을 통해 데이터의 전체적 모습을 쉽고 빠르게 파악할 수 있다.

추론통계Inferential Statistics는 일부 데이터를 이용해 데이터 전체의 모습을 추정하는 것으로, 예를 들어 대한민국에 사는 전체 남성의 체중에 대한 통계치를 파악하고

싶다고 가정해보자. 현실적으로 생각해볼 때 대한민국에 사는 2,000만 명 이상 되는 남성의 체중을 모두 측정하는 것은 쉬운 일이 아니다. 그래서 대부분 일부 남성을 선택하고 이들의 체중을 측정해 대한민국 전체 남성의 데이터를 추정하는 방법을 쓰는데, 이것이 추론통계다.

기술통계와 추론통계는 서로 상보 관계에 있다. 기술통계가 데이터 전체의 모습을 간략히 요약하는 데 집중한다면, 추론통계는 제한된 수의 데이터를 통해 전체 데이터의 모습을 파악하기 때문이다.

2.2 통계가 머신러닝에서 중요한 이유

냉정하게 이야기하면 머신러닝 프로그램을 개발하는 데 통계를 전혀 모른다고 해도 문제가 될 것은 없다. 대개의 머신러닝 알고리즘은 적절한 데이터를 제공하고 적절한 파라미터만 지정한 후에 학습시키면 스스로 모델을 찾도록 설계되어 있다. 따라서 머신러닝 프로그램을 개발하는 과정은 먼저 학습시키려는 입력과 출력 데이터를 머신러닝 프로그램에 입력해 학습시키는 것으로 마무리되고, 그 후에는 학습된 머신러닝 프로그램에 입력 데이터를 넣어 활용하는 단계다. 그 결과치의 품질, 즉 정확도가 어느 정도냐에 대한 고민, 어떻게 하면 성능을 더 향상할 수 있을 것인가에 대한 관심이 없다면 통계는 그다지 중요하지 않을 수 있다.

앞장에서 설명한 머신러닝 프로세스를 잠시 되새겨보자. 머신러닝을 이용한다는 것은 전처리, 학습, 후처리 등의 과정을 거치는데, 이러한 과정이 머신러닝의 품질을 결정하는 데 중요한 역할을 한다고 얘기했다. 머신러닝 알고리즘의 성능을 결정하는 데 절대적인 것은 학습용 데이터셋이지 사용하려는 알고리즘이 아니다. 아무리 좋은 알고리즘이더라도 학습용 데이터셋이 너무 엉망이라면 결코 좋은 성능을 기대할 수 없다. 반대로 학습용 데이터셋이 아주 훌륭하다면 다른 알고리즘보다 우수하지 않은 알고리즘이라도 훌륭한 성능을 보여준다.

'Garbage-in, Garbage-out'의 원칙은 절대 깨지지 않는다!

머신러닝 프로세스에서 머신러닝 알고리즘에 직접 연관되지 않은 절차들, 즉 전처리, 후처리 등은 데이터를 만지는 과정이다. 분실자료Missing Data를 채워 넣고 이상치 등을 적절한 값으로 대체하거나 삭제하며 많은 독립 변수 중에 무엇을 선택할 것인가 등의 과정이 머신러닝 알고리즘보다 월등히 중요하고, 이들 과정은 알고리즘이 아닌 사람 손에서 이루어진다.

이러한 과정을 효과적으로 수행하는 데 필요한 것이 바로 통계다. 예를 들어, 이상치를 제거하고 싶다면 이상치에 대한 명확한 정의를 내려야 한다. 평균값의 3배 이상 크거나 작은 값을 이상치로 지정한다든가 1000 이상의 값을 가진 모든 데이터는 이상치로 취급하는 등의 결정을 해야 한다. 이상치는 머신러닝 알고리즘의 학습 과정에서 가중치 값을 왜곡할 수 있어서 간단히 결정할 문제는 아니므로 올바른 판단이 필요하다.

기술통계를 이용하면 이상치 문제를 쉽게 해결할 수 있다. 데이터 분포도를 이용해 이상치로 보이는 값들이 평균, 분산, 표준편차 등과 비교해 어떤 특성이 있는지, 전체 데이터에서 어느 정도를 차지하고 있는지 등을 알 수 있다. 또한, 어떤 결과치를 얻었다고 하면 해당 결과치가 의미가 있는 결과인지 아닌지를 판단하는 데도 역시 통계가 중요한 역할을 한다.

이처럼 통계학적 지식과 기법들은 데이터 처리를 위해 다양하게 활용될 수 있기 때문에 머신러닝과 떼려야 뗄 수 없는 긴밀한 연관성을 가지고 있다. 통계에 있는 많은 수식과 개념을 모두 이해할 필요는 없지만, 핵심적인 개념과 수식들은 숙지할 필요가 있다.

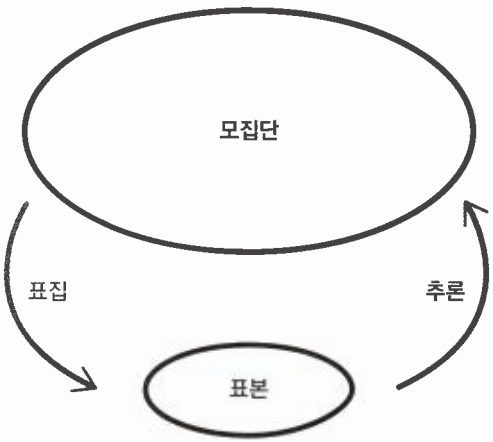
2.3 통계의 기본 개념과 용어

2.3.1 모집단과 표본

머신러닝으로 어떤 문제를 해결하려고 할 때, 학습용 데이터셋이 해결하려는 문제와 관련된 전체 데이터를 가졌는지 아니면 일부 데이터만을 가졌는지는 성능에 영향을 미칠 수밖에 없다. 상식적으로 생각해보면 전체 데이터가 일부 데이터를 사용한 것보다는 좋은 성능을 내는 것이 당연하다.

통계에서는 데이터의 범위에 대해 명확한 구분을 하는데, 관심있는 문제와 연관된 전체 집합을 '모집단Population', 전체 데이터로부터 추출된 일부 집합을 '표본Sample'이라고 한다. 예를 들어, 대한민국 전체 남성의 체중을 측정한다고 하면 모집단은 대한민국 전체 남성이 되고, 표본은 체중 측정을 위해 선택한 남성들이 된다.

그림 2-1 모집단과 표본



2.3.2 파라미터와 통계량

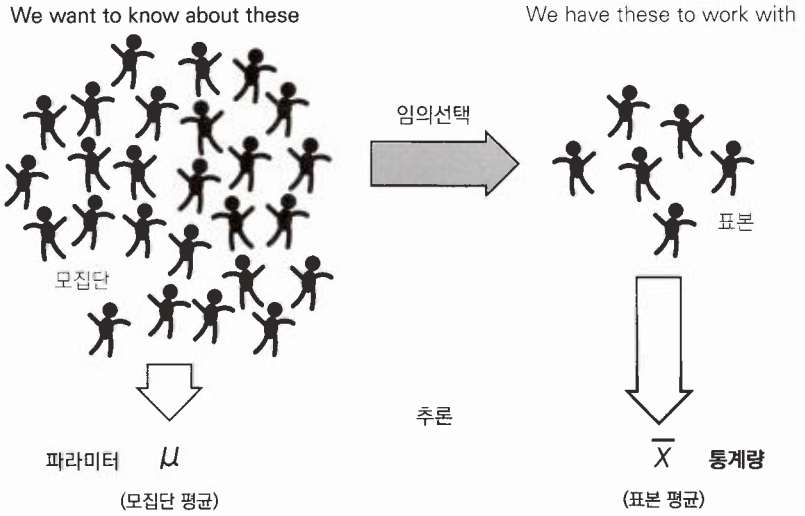
데이터를 하나하나 보면서 파악하기가 어렵기 때문에 다수의 데이터를 이야기할 때는 일반적으로 데이터의 특성을 설명해주는 평균과 표준편차 같은 값을 이용한

다. 이 값들은 전체적인 데이터의 모습이 어떠한지를 짧은 시간에 가늠할 수 있어서 많이 활용된다.

그런데 평균과 표준편차 같은 값들을 사용하는 데 그 데이터가 모집단으로 수집된 것인지 표본에서 얻었는지를 명확히 할 필요가 있다. 평균이라 하더라도 모집단의 평균과 표본의 평균은 다를 수밖에 없기 때문이다. 모집단의 평균은 평균으로 인정하고 사용할 수 있지만, 표본의 평균은 해당 표본을 추출한 모집단의 평균과 같지 않을 가능성이 매우 크다.

모집단으로부터 얻은 데이터의 성격을 나타내주는 지표, 예를 들어 평균, 분산 등을 '파라미터Parameter'라고 한다. 이와 달리 표본으로부터 얻은 것은 '통계량Statistic'이라고 한다. 데이터를 어디에서 얻었느냐는 중요한 의미가 있으므로 파라미터와 통계량은 구분해서 사용해야 한다.

그림 2-2 파라미터 vs 통계량



파라미터와 통계량에 대한 예로 대한민국 전체 남성의 평균체중을 구하는 문제를 생각해보자. 모집단의 평균값이 60kg이라고 하면, 해당 도메인의 전체 집합에서

구한 평균값이므로 60kg은 파라미터다. 그렇다면 표본으로부터 얻는 평균값인 통계량도 60kg이 될까?

그럴 수도 있고 그렇지 않을 수도 있다. 예를 들어, 표본을 만들기 위해 선택한 40명의 남성 중 상당수가 10세 미만이라면 표본으로부터 얻은 평균값은 60kg보다 훨씬 작을 것이다. 반대로 키가 180cm를 넘는 사람이 많다면 표본 평균값은 60kg보다 많이 나올 것이다.

이처럼 파라미터와 통계량은 다를 수 있으므로 이들을 구분해서 사용해야 한다.

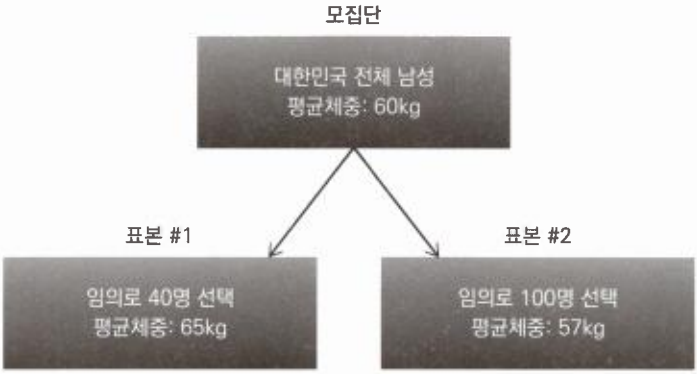
2.3.3 표집 오차

표집 오차Sampling Error는 모집단과 표본의 차이로 인해 발생하는 오차를 의미한다. 모든 데이터를 빠짐없이 수집해 모집단을 만드는 것은 비용적으로나 시간상으로 어려운 경우가 많아서 대개 표본을 이용한다. 표본을 만들기 위해 모집단으로부터 데이터를 선택하는 과정인 표집Sampling에서 모집단을 잘 대변할 수 있게 대상을 선택하는 것은 사실상 불가능한 경우가 많다. 아무리 많은 노력을 기울이고 지능적인 표집 방법을 사용한다 하더라도 표본과 모집단의 모든 특성이 100% 일치할 수는 없다.

예를 들어, 대한민국 전체 남성의 평균체중을 구하기 위해 임의로 40명과 100명을 선택해 표본 #1, 표본 #2를 만들었다고 하자. 이들 2개 표본의 평균체중을 구해보면, 표집 오차 때문에 모집단의 평균체중인 60kg이 아닌 65kg, 57kg처럼 값이 다르게 나온다.

모집단이 아닌 표본을 사용한다면 언제나 표집 오차가 존재하므로 이를 염두에 두고 데이터를 사용해야 한다. 머신러닝에 사용하는 모든 학습용 데이터셋이 과연 얼마나 모집단을 잘 반영하는지에 대해 검증해봐야 한다.

그림 2-3 표집 오차



2.3.4 종속변수와 독립변수

종속변수는 어떤 목적에 대한 결과를 파악하기 위해 사용하는 변수를 의미한다. 종속변수는 '반응변수Response Variable, 측정변수Measured Variable, 피예측변수Predicted Variable, 피설명변수Explained Variable, 출력변수Output Variable' 등 다양한 명칭을 가지고 있다. 독립변수는 '예측변수Predictor Variable, 특징Feature, 입력변수Input Variable, 설명변수Explanatory Variable' 등의 명칭으로 불리기도 한다. 예를 들어, 주택크기에 따라 가격이 결정된다고 하면, 주택크기가 독립변수, 가격이 종속변수가 된다.

2.3.5 연속변수와 이산변수

연속변수Continuous Variable 와 이산변수Discrete Variable은 통계처리를 하려는 값의 특성에 따라 분류한 것으로, 이에 대한 명확한 이해가 필요하다. 표기는 숫자로 되어 있더라도 그 숫자가 연속변수인 경우와 이산변수인 경우는 완전히 다르고, 이에 따라 적용할 수 있는 이론과 방법들이 달라진다.

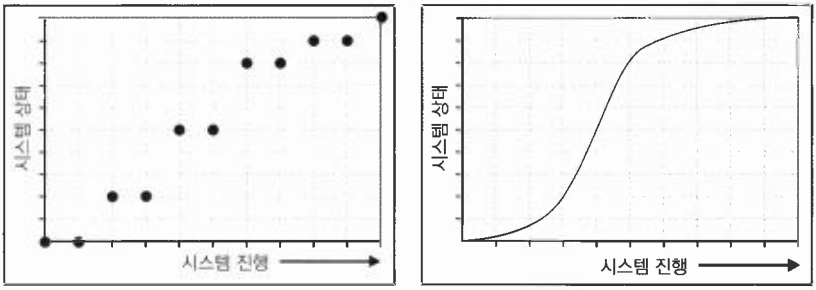
연속변수는 실수Real Number로 표현할 수 있는 변수를 지칭하는 것으로, 제일 중요한 성질은 특정 범위에서 허용되는 값이 '셀 수 없다Uncountable'는 점이다. 이러한 성질 때문에 연속변수라는 이름이 붙게 되었다. 예를 들어, 데이터가 0부터 100

까지의 실수라고 가정하면 실수는 1.01, 1.001, 1.0001, 1.000001과 같은 수가 끝없이 이어지므로 0~100 사이에 존재하는 값의 수는 셀 수가 없다.

이산변수는 연속변수와 정반대의 성질을 가진다. 연속변수는 셀 수 없지만, 이산변수는 셀 수 있다Countable. 예를 들어, 데이터가 0부터 100까지의 자연수Natural Number라면 자연수는 소수를 허용하지 않으므로 0~100 사이에는 0, 1, 2, 3, 4, …, 100처럼 총 101개의 값만이 존재할 수 있다.

다음 그림은 연속변수와 이산변수의 특성을 잘 나타내준다.

그림 2-4 이산변수와 연속변수



데이터를 볼 때 이산변수인지 연속변수인지를 명확하게 파악해야 하는 이유는 종류에 따라 적절한 방법을 선택해야 하기 때문이다. 예를 들어, 확률분포Probability Distribution를 파악하려 한다면 연속변수에서는 확률밀도함수Probability Density Function를 사용하지만, 이산변수라면 확률질량함수Probability Mass Function를 사용해야 한다.

독립변수가 이산변수인지 연속변수인지를 제대로 파악하지 않고 또는 잊어버리고 엉뚱한 방법을 적용하는 사례가 심심치 않게 있으므로 항상 유의해야 한다.

2.3.6 모델

우리는 여러 가지 목적을 가지고 데이터를 활용한다. 스팸메일을 분류하거나 주식 가격을 예측하거나 농산물의 수확량을 예측하는 등 삶을 영위하는 데 필요한 그

무언가를 만들기 위해 활용한다. 이외에도 다양한 목적이 있지만, 본질적인 의미에서 생각해보면 데이터를 통해 대상을 잘 이해하기 위함이 모든 것의 시작이라 할 수 있다.

모델Model은 대상을 설명하기 위한 표현 방법이라고 할 수 있다. 예를 들어, 주택크기와 가격을 예측하고 싶다면 이들 관계를 주택가격을 Y, 주택크기를 X로 놓고 다음 모델로 표현할 수 있다.

$$Y = aX + b$$

이 모델은 주택가격이 크기와 비례 관계에 있다는 것을 나타내는 것으로, 이 모델이 통계검정을 통해 올바른 모델이라는 것이 검증되면 이 모델을 이용해 주택크기만 알아도 주택가격이 얼마가 될지를 알 수 있게 된다. 예를 들어, 가지고 있는 데이터는 20평부터 30평까지의 주택크기와 주택가격 데이터인데, 주택크기가 40평인 주택의 가격을 알고 싶다고 하자. 먼저 해야 할 일은 주택크기와 주택가격을 설명하는 모델을 만들고, 가지고 있는 데이터로 학습시켜 모델의 완성에 필요한 a와 b의 값을 찾는 것이다. 모델이 완성되면 원래 알고 싶었던 40평 주택의 가격을 예측하기 위해 X에 '40'이라는 값을 넣어 Y를 구하면 이 값이 40평 주택의 가격이 된다.

모델은 독립변수가 어떻게 종속변수를 생성하는지를 보여주기 때문에 문제에 대해 더 잘 이해할 수 있도록 도와주고, 생각을 정리하고 개선하는 데 많은 도움을 준다.

2.4 준비사항

실습에 필요한 환경은 다음과 같다.

- **언어** Python 2.7

- **라이브러리** pandas, Numpy, SciPy, matplotlib
- **OS** 파이썬이 동작할 수 있는 OS

파이썬은 Windows, OS X, Linux 등 거의 모든 OS에서 동작하므로 자신이 사용하는 OS 환경에 파이썬과 라이브러리를 설치하면 실습을 위한 준비는 거의 마쳤다고 할 수 있다. 앞으로 사용하게 될 라이브러리인 pandas, Numpy, SciPy 등은 설치가 약간 까다로울 수 있으므로 Anaconda와 같은 전용 설치 프로그램을 이용하는 것도 좋다.

2.5 데이터 다운로드

이 책은 알고리즘 트레이딩에 관련된 책이므로 실습에 사용할 데이터는 당연히 주가 데이터를 다운로드해 사용한다. 주가 데이터는 야후 파이낸스(https://finance.yahoo.com/)에서 다운로드한다. pandas에는 야후 파이낸스에서 국내 주가 데이터를 다운로드하는 기능이 있어서 간단한 코드만으로도 필요한 데이터를 쉽게 구할 수 있다.

사용하는 방법도 매우 쉬워 데이터 수집을 원하는 일자와 종목코드를 입력하면 모든 것이 완료된다. 한 가지 주의할 점은 주식코드에 '.KS'라는 접미사를 붙여야 한다. 예를 들어, 삼성전자의 종목코드가 '005930'인데 야후 파이낸스를 이용해 주가 데이터를 받으려면 '005930.KS'라고 입력해야 한다.

다음 코드는 주가 데이터를 다운로드하는 download_stock_data( ) 함수다.

```
#-*- coding: utf-8 -*-
import pandas as pd
import pandas.io.data as web
import datetime

def download_stock_data(file_name,company_code,year1,month1,date1,year2,month2
,date2):
```

```
    start = datetime.datetime(year1, month1, date1)
    end = datetime.datetime(year2, month2, date2)
    df = web.DataReader("%s.KS" % (company_code), "yahoo", start, end)

    df.to_pickle(file_name)

    return df
```

download_stock_data( ) 함수의 인자는 다음과 같다.

- file_name 다운로드한 주가 데이터를 저장할 파일 이름을 지정한다.
- company_code 종목코드를 지정한다.
- year1/month1/date1 데이터를 다운로드할 시작일을 년, 월, 일 순서로 입력한다.
- year2/month2/date2 데이터를 다운로드할 마감일을 년, 월, 일 순서로 입력한다.

다음은 download_stock_data( ) 함수를 이용해 삼성전자의 2015년 1월 1일부터 2015년 11월 30일까지의 주가 데이터를 받는 코드다. 코드를 실행하면 주가 데이터가 저장된 'samsung.data'라는 이름의 파일이 생성된다.

```
download_stock_data('samsung.data','005930',2015,1,1,2015,11,30)
```

다음은 다운로드한 주가 데이터의 일부를 출력한 화면이다.

그림 2-5 삼성전자 주가 데이터

Date	Open	High	Low	Close	Volume	Adj Close
2015-11-02	1385000	1393000	1374000	1383000	386500	1383000
2015-11-03	1381000	1381000	1350000	1352000	301800	1352000
2015-11-04	1352000	1361000	1326000	1330000	281000	1330000
2015-11-05	1330000	1354000	1330000	1342000	173000	1342000
2015-11-06	1343000	1348000	1330000	1338000	164300	1338000
2015-11-09	1338000	1344000	1321000	1344000	185600	1344000
2015-11-10	1336000	1341000	1314000	1321000	197500	1321000
2015-11-11	1321000	1345000	1321000	1333000	140400	1333000
2015-11-12	1333000	1334000	1317000	1317000	157400	1317000
2015-11-13	1317000	1317000	1300000	1300000	177600	1300000
2015-11-16	1291000	1291000	1263000	1263000	275700	1263000
2015-11-17	1275000	1290000	1270000	1270000	186100	1270000
2015-11-18	1272000	1290000	1272000	1281000	167700	1281000
2015-11-19	1290000	1290000	1271000	1289000	192800	1289000

화면 상단을 보면 Open, High, Low, Close 등의 열 이름이 있는데, 각각의 의미는 다음과 같다.

- **Open** 시가
- **High** 고가
- **Low** 저가
- **Close** 종가
- **Volume** 거래량
- **Adj Close** 주식의 분할, 배당, 배분 등을 고려해 조정한 종가

야후 파이낸스에서 제공하는 주가 데이터는 일간 데이터로 시간이나 분, 초 단위의 주가 데이터는 제공하지 않는다.

2.6 데이터 로드

앞에서 download_stock_data( ) 함수를 이용해 다운로드한 주가 데이터를 불러와 이를 사용하는 함수는 load_stock_data( )이다. 이 함수는 pickle 파일로 저장된 주가 데이터를 읽어서 pandas DataFrame으로 반환한다.

```
def load_stock_data(file_name):
    df = pd.read_pickle(file_name)
    return df
```

앞으로 이 장에서 사용하는 데이터는 load_stock_data를 이용한다.

2.7 기초통계

기초통계Elementary Statistics에는 데이터의 전체적인 모습을 이해하는 데 매우 유용한 도구들이 있다. 주가 데이터와 머신러닝을 이용해 주식거래를 하고 싶다면 아마도 가장 먼저 해야 할 것은 주가 데이터의 특성을 파악하는 일이다.

국내 주식시장 전체의 특성은 어떤지, 관심 있는 회사의 주가는 어떤 모습인지를 파악하려면 주가 수치 데이터만으로는 알기 어렵다. 최솟값, 최댓값, 평균값, 분산, 히스토그램 등의 도구들을 이용하면 데이터의 범위가 얼마나 되는지, 변동성은 얼마나 되는지, 평균적인 주식가격이 얼마인지를 쉽게 파악할 수 있다.

다음 코드를 실행해 삼성전자 주식의 전체적인 모습을 살펴보자. pandas에는 describe( ) 함수가 있어 DataFrame에 저장된 데이터의 전체 모습을 별도의 코드 없이 쉽게 알 수 있다.

```
df = load_stock_data('samsung.data')
print df.describe()
```

그림 2-6 실행결과

	Open	High	Low	Close
count	238.000000	238.000000	238.000000	238.000000
mean	1301852.941176	1314323.529412	1288285.714286	1300966.386555
std	109294.744392	109067.742822	108619.542177	108850.490330
min	1068000.000000	1074000.000000	1033000.000000	1067000.000000
25%	1245000.000000	1260750.000000	1231250.000000	1252250.000000
50%	1315000.000000	1323500.000000	1300000.000000	1314000.000000
75%	1376500.000000	1390000.000000	1364000.000000	1376500.000000
max	1510000.000000	1510000.000000	1486000.000000	1503000.000000

실행결과 화면에 나온 열과 행의 의미는 다음과 같다.

- **count** 데이터 개수를 의미한다.
- **min** 데이터에서 가장 작은 값이다.
- **max** 데이터에 제일 큰 값이다.
- **mean** 평균값으로 산술평균을 의미한다.
- **std** standard deviation의 약자로 표준편차를 의미한다.
- **25%, 50%, 75%** 사분위수Quantile를 의미한다.

2015년 1월부터 11월까지의 삼성전자 주식 종가 데이터를 보면 데이터 개수가 238개, 저가가 1,067,000원, 고가가 1,503,000원, 평균은 1,300,966원임을 알 수 있다. 표준편차는 108,850이고, 3사분위수가 1,376,500임을 알 수 있다.

2.7.1 표준편차

표준편차Standard Deviation는 데이터의 분산 정도를 측정하는 데 매우 중요한 개념이다. 표준편차는 데이터값과 평균 사이의 거리를 계산해 데이터값들이 평균으로부터 얼마나 멀리 떨어져 있는지를 알려준다. 표준편차가 크다는 것은 많은 수의 데이터가 평균으로부터 멀리 있다는 것을 의미하고, 반대로 표준편차가 작다는 것은 데이터값들이 평균에 가까이 있음을 의미한다.

예를 들어, 대한민국 전체 남성의 평균체중이 50kg에 표준편차가 20이라고 하면 대부분 남성의 체중은 50kg - 20 = 30kg에서 50kg + 20 = 70kg 사이에 있다는 것을 의미한다.

표준편차값만 보아도 전체 데이터의 분포가 밀집되어 있는지 성긴지를 쉽게 알 수 있다. 표준편차의 값이 작을수록 데이터가 모여 있고, 값이 클수록 퍼져 있다. 표준편차 값이 '0'이라는 것은 모든 데이터가 동일한 값임을 의미하는 것으로 데이터가 한곳에 모두 모여있음을 보여준다. 반대로 표준편차가 무한대에 이를 정도로 크다면 데이터는 한곳에 모여있지 않고 모두 퍼져 있음을 나타낸다.

예를 들어, 삼성전자 주가분포와 한미약품 주가분포를 비교해보면 표준편차가 작은 삼성전자가 표준편차가 큰 한미약품보다 평균에 근접해 있음을 알 수 있다. 삼성전자 주가의 표준편차는 109,294고, 한미약품의 표준편차는 186,015로 한미약품 주가분포가 삼성전자 주가분포보다 넓게 퍼져 있는 것을 확인할 수 있다.

표준편차가 크다는 것은 그만큼 데이터의 변동이 크다는 것을 의미하기도 한다.

그림 2-7 삼성전자 주가분포

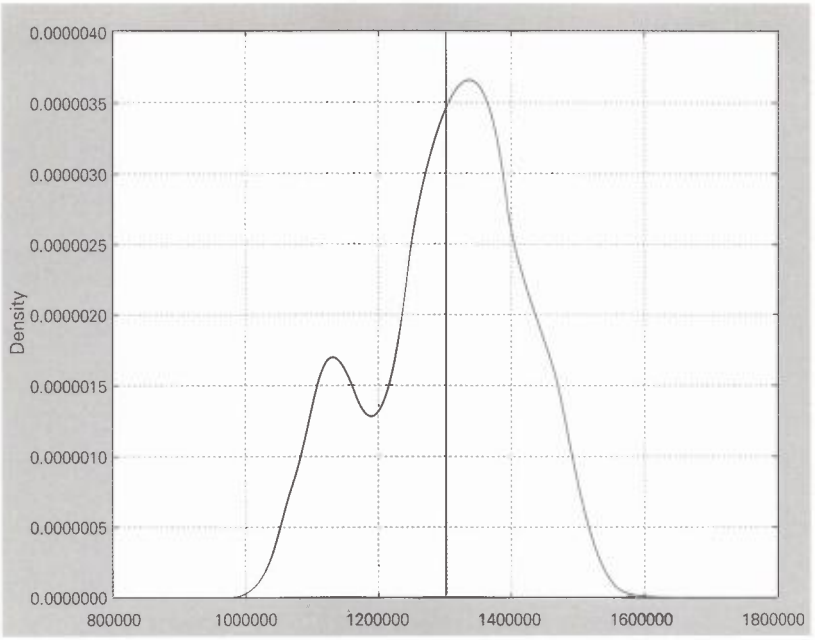
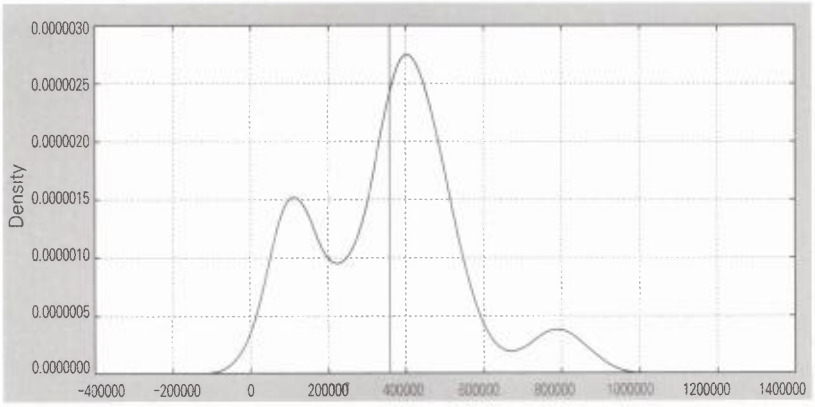


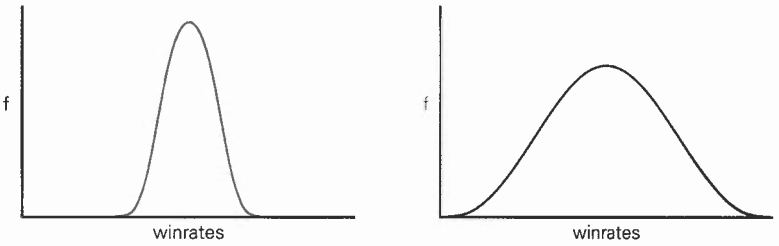
그림 2-8 한미약품 주가 분포



표준편차에는 확률과 연관된 또 다른 중요한 의미가 있다. 다음 그림처럼 표준편차 값을 알면 데이터의 분포 정도를 가늠할 수 있는데, 분포는 다른 관점에서 보면 어떤 사건이 발생한 빈도를 의미한다.

[그림 2-9]의 왼쪽 그래프처럼 표준편차의 값이 작으면 분포가 밀집되어 있다는 것인데, 이 말은 어떤 사건이 발생했을 때 평균과의 차이가 작은 값을 가질 확률이 높다는 것을 의미한다. 오른쪽 그래프처럼 표준편차의 값이 크면 특정 사건이 발생했을 때 평균값과 차이가 큰 값을 가질 확률이 높다는 것을 의미한다.

그림 2-9 표준편차



낮은 표준편차 높은 표준편차

표준편차는 다음 식으로 계산한다. 표준편차는 이 식에서 알 수 있듯이 데이터의 분포 정도를 나타내는 척도인 분산에 루트를 씌워 계산한 값이다. 따라서 표준편차를 제대로 이해하려면 분산이 무엇인지를 알아야 한다.

$$\text{표준편차} = \sqrt{\text{분산}}$$

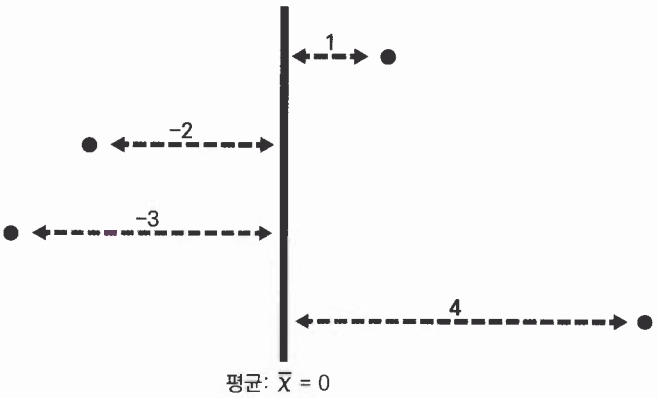
분산Variance은 분포에서 데이터가 퍼져 있는 정도를 나타내는 것으로, 데이터가 얼마나 밀집되어 있는지 넓게 퍼져있는지를 알려준다. 좀 더 자세히 말하면 분산은 평균으로부터 데이터들이 얼마나 멀리 떨어져 있는지를 알려주는 일종의 거리 개념으로, 표준편차와 유사하다.

분산식은 다음과 같다. '(X - 평균)'은 특정 값이 평균으로부터 얼마나 떨어져 있는가를 나타내는 것으로 '편차'라고 한다.

$$\text{분산} = \frac{\Sigma(x - \text{평균})^2}{\text{크기}}$$

다음 그림이 편차의 개념을 잘 나타내는데, 1, -2,-3,4와 같이 데이터값과 평균의 차이, 즉 회색 점선으로 표시한 것이 편차를 의미한다.

그림 2-10 분산 개념 설명도



편차는 데이터값에 따라 음수가 나올 수 있지만, 분산은 0보다 커야 하므로 제곱을 해서 양수가 되게 한다. 앞서 이야기했듯이 분산과 표준편차는 공식을 보아도 알 수 있듯이 유사한 개념이지만 분산보다는 표준편차가 많이 사용되는데, 이는 표준편차가 분산과 달리 데이터값과 같은 단위를 사용하기 때문이다.

표준편차는 데이터값이 평균에서 얼마나 밀집되어 있는지 퍼져 있는지, 즉 값의 변화가 얼마나 심한지를 파악할 수 있다. 아울러 위험도의 측정, 수익률 계산 등, 다방면에 활용되는 매우 주요한 지표로 여러 용도로 활용되기 때문에 그 개념을 확실하게 이해할 필요가 있다.

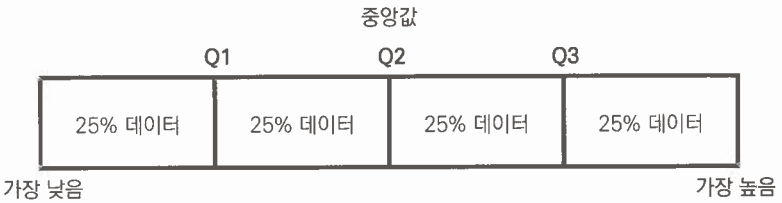
2.7.2 사분위수

사분위수Quartile는 데이터를 동일한 크기로 4등분, 즉 Q1-25%, Q2-50%, Q3-75%로 나눈 것이다. 데이터의 변화 정도를 알기 위해서 최댓값에서 최솟값을 빼서 구하는 '범위Range'를 사용한다.

범위 = 최댓값 - 최솟값

예를 들어, 데이터의 최솟값이 0이고 최댓값이 100이라면, 범위는 '100 - 0 = 100'이 된다. 즉, 데이터가 변할 수 있는 값의 범위가 100이 되는 것이다. 하지만 범위를 사용하면 전체가 아닌 부분별 데이터 범위를 알지 못한다. 이러한 단점을 보완하기 위해 사분위는 전체 구간을 4개로 나누어 범위를 계산해 데이터의 세부적인 모습을 알 수 있다.

그림 2-11 사분위수와 데이터 분포



사분위는 pandas의 quantile( ) 함수를 이용하면 쉽게 계산할 수 있다. 다음은 삼성전자 주가의 사분위를 구하는 코드다

```
df = load_stock_data('samsung.data')
print df.quantile([.25,.5,.75])
```

이 코드를 실행하면 다음과 같은 결과가 나온다.

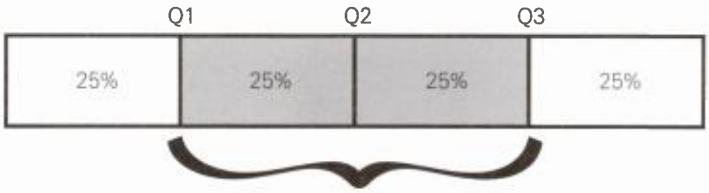
```
      Open     High     Low      Close    Volume  Adj Close
0.25  1245000  1260750  1231250  1252250  169725  1252250.0000
0.50  1315000  1323500  1300000  1314000  206350  1312971.8100
0.75  1376500  1390000  1364000  1376500  273725  1375422.9025
```

IQR

IQR[Interquartile Range, 사분위 간 범위]은 분포에서 하위 25%와 상위 25%를 제외한 중간에 있는 50%의 범위를 나타낸다. IQR을 구하는 식은 다음과 같다.

$$IQR = Q3 - Q1$$

그림 2-12 IQR



사분위 간 범위 = Q3 - Q1

IQR은 전체분포 중에서 가운데에 위치하고 있어 대다수 데이터가 어느 범위에 있는지를 가늠할 수 있게 한다. 사분위와 IQR은 히스토그램을 보지 않아도 직관적으로 전체 데이터가 어떻게 분포되어 있는지를 빠르게 파악할 수 있어서 유용하고, 또 구간별 데이터 분포를 알려주기 때문에 많이 사용한다.

2.7.3 히스토그램

히스토그램[Histogram]은 데이터의 구성과 패턴을 파악하는 데 많이 사용하는 방법의 하나로, 데이터의 분포를 그래프 형태로 나타낸 것이다. 히스토그램을 그리려면 Bin과 빈도[Frequency]가 필요하다. Bin은 전체 데이터를 겹치지 않는 일정 크기로 나눈 간격을 의미하고, 빈도는 선택한 Bin에 속한 데이터의 수를 나타낸다.

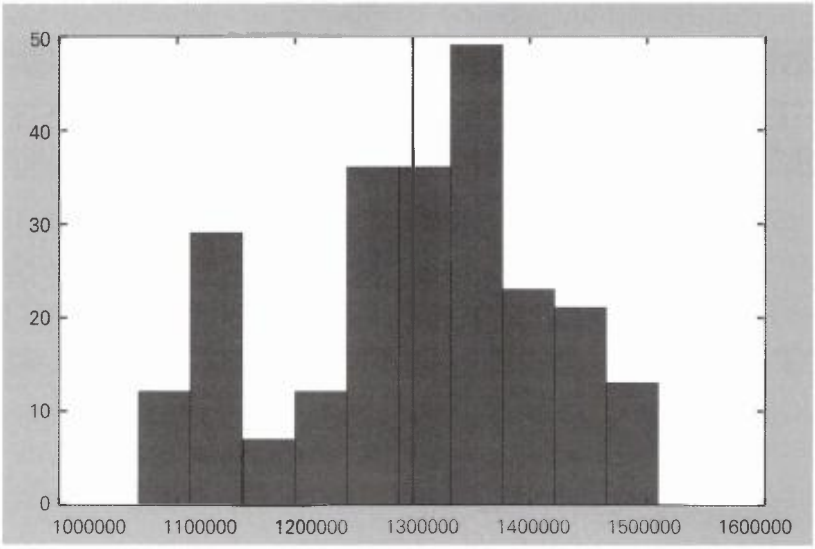
예를 들어, 2015년 1월부터 11월까지 삼성전자 주가의 히스토그램을 그려보자. 다음 코드는 matplotlib에서 제공하는 `hist()` 함수를 이용해 삼성전자 주가 히스토그램을 그리고 Bin과 빈도를 표시해준다.

```
df = load_stock_data('samsung.data')
```

```
(n, bins, patched) = plt.hist(df['Open'])
plt.axvline(df['Open'].mean(),color='red')
plt.show()
for index in range(len(n)):
    print "Bin : %0.f, Frequency = %0.f" % (bins[index],n[index])
```

실행결과는 다음과 같다.

그림 2-13 삼성전자 주가 히스토그램



Bin과 빈도의 결과는 다음과 같다.

```
Bin : 1068000, Count = 12
Bin : 1112200, Count = 29
Bin : 1156400, Count = 7
Bin : 1200600, Count = 12
Bin : 1244800, Count = 36
Bin : 1289000, Count = 36
Bin : 1333200, Count = 49
Bin : 1377400, Count = 23
Bin : 1421600, Count = 21
Bin : 1465800, Count = 13
```

그래프와 앞의 결과를 보면 삼성전자의 주가는 1,289,000~1,333,200 구간에서 49회로 최다빈도를 보여주고 있다.

히스토그램의 이해

히스토그램은 데이터 분포를 이해하는 데 필요한 많은 정보를 제공한다. 주요 항목은 다음과 같다.

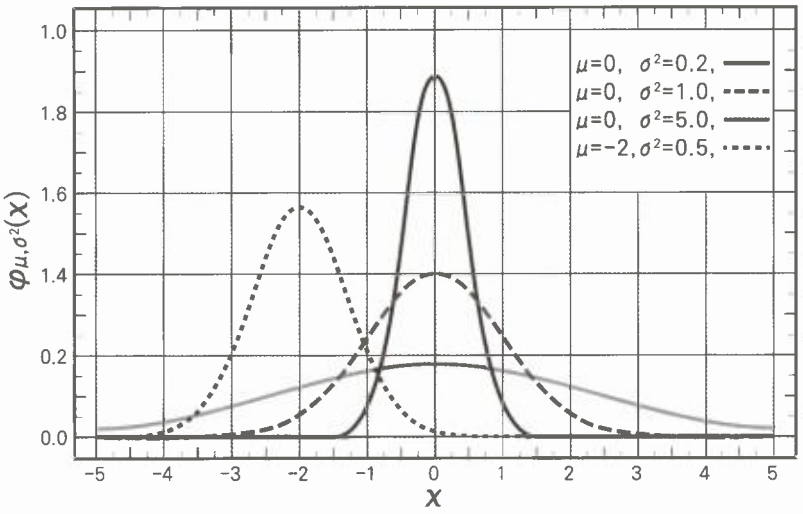
- **중심 성향**Central Tendency 데이터가 평균값을 중심으로 분포되어 있는가?
- **Modes** 데이터 분포상 하나 이상의 무리가 있는가?
- **Spread** 데이터가 어느 정도로 분산되어 있는가?
- **Tail** 하위 25%와 상위 25% 데이터 분포의 기울기 하락도가 완만한가 아니면 급한가?
- **이상치** 예외값이 분포도에 존재하는가?

[그림 2-13]을 살펴보면 가운데 굵은 선이 평균값인데, 최다빈도가 평균값이 속한 Bin이 아닌 곳에서 발생했기 때문에 평균값을 중심으로 분포되어 있다고 말하기 어려워 '중심 성향이 낮다'고 할 수 있다. 히스토그램의 형태를 보면 'Bin : 11122000과 Bin : 1333200'을 중심으로 2개의 무리가 있음을 알 수 있다. Spread와 Tail은 히스토그램에 2개의 무리가 있어서 이야기하기 힘들고, 그래프 상에서 이상치는 보이지 않음을 알 수 있다.

2.7.4 정규분포

정규분포Normal Distribution는 데이터의 중심을 나타내는 평균과 데이터의 밀집 정도를 나타내는 표준편차에 의해 결정되는 분포로, 다른 이름으로 가우스 분포Gaussian Distribution라고도 한다. 특히 평균이 0이고 표준편차가 1인 정규분포를 표준정규분포Standard Normal Distribution라고 한다.

그림 2-14 정규분포 그래프[01]



정규분포는 통계와 확률에서 중요한 개념이며, 다방면에서 활용된다. 독일의 유명한 수학자인 가우스Gauss가 물리학 실험을 할 때 발생한 계측오차에 대한 확률분포로 제시하였다. 재미있는 사실은 가우스는 실험 계측오차에 대한 확률분포로 발견했지만, 정규분포가 가우스의 실험과는 전혀 연관이 없어보이는 사회, 경제, 심리 등 많은 분야에 광범위하게 적용해 사용하고 있으며, 분포를 잘 모르는 상황이라면 정규분포를 따른다고 가정할 정도로 많은 사람의 지지를 받고 있다는 것이다.

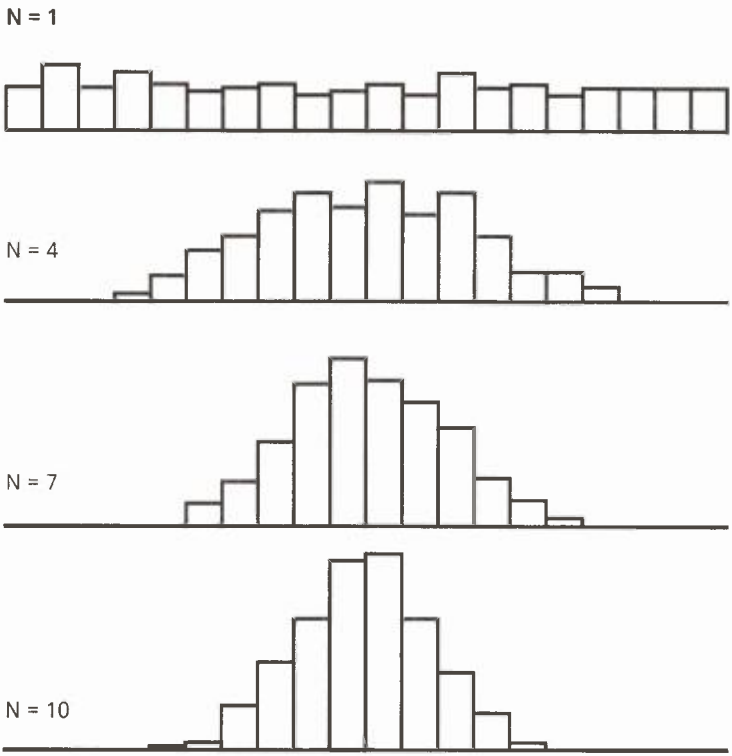
정규분포가 자연계에 많이 존재하는 이유는 '중심극한정리Central Limit Theorem' 때문이라고 밝혀졌다. 중심극한정리는 N이 적당히 크다면 동일한 확률분포를 가진 독립변수의 평균값은 정규분포에 가까워진다는 것으로, 통계와 확률에서 매우 중요한 정리다.

---

01 출처: https://goo.gl/BGGG83

[그림 2-15]는 N이 커질수록 정규분포를 따라가는 모습을 보여준다.

그림 2-15 N 값에 따른 정규분포 변화



정규분포가 자연계에서 흔히 발생하기는 하지만 정규분포를 따르지 않는 현상도 있을 수 있다. 정규분포 외에도 베르누이 분포Bernoulli Distribution, 기하분포Geometric Distribution, 포아송분포Poisson Distribution, 지수분포Exponential Distribution 등 많은 종류의 분포가 있다.

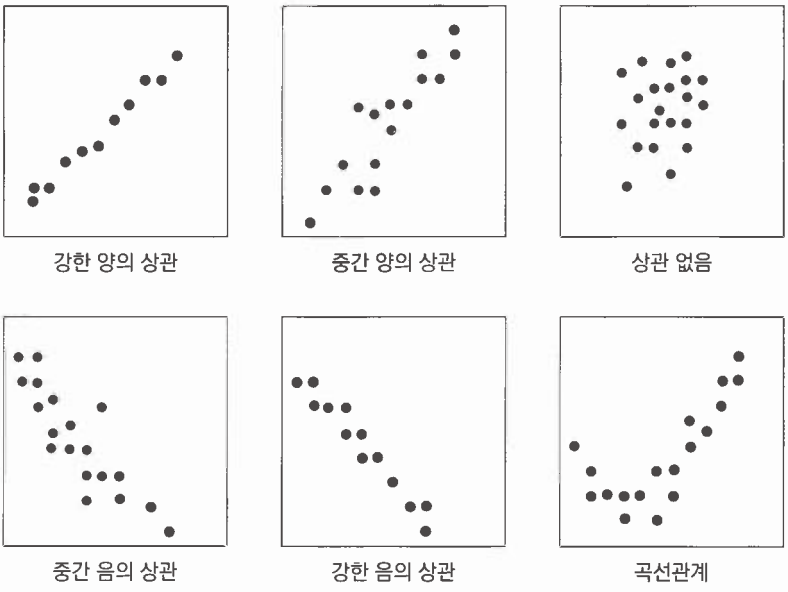
자신이 다루는 데이터가 어떤 분포와 유사한 성향을 보이는지를 파악해야 올바른 방법을 적용할 수 있으므로 신중하게 선택해 사용해야 한다.

2.7.5 산점도

산점도Scatter plot는 2개 변수를 직교좌표계에 그린 그래프로 변수 간의 관계를 파악하는 데 특히 유용하다. 산점도를 보면 2개 변수가 상관관계가 있는지 없는지, 있다면 양의 관계인지 음의 관계인지를 시각적으로 바로 파악할 수 있다. 어떤 형태의 선형적 패턴을 찾을 수 있으면 상관관계가 있을 가능성이 매우 크고, 그렇지 않고 여기저기 데이터가 무작위로 흩어져 있는 형태라면 선형적 상관관계가 없다고 할 수 있다.

다음 그림은 산점도를 이용한 대표적인 상관관계의 예를 보여준다.

그림 2-16 산점도와 상관관계



산점도를 해석할 때 한 가지 유의할 사항이 있다. 산점도가 강한 양의 상관관계를 나타낸다는 것은 2개의 변수가 어떤 인과관계Causation가 있다고 이야기하기 어렵다는 점이다. 특별한 관계없이 우연히 상관관계가 그래프에 나타났을 수도 있고, 2개의 변수 모두 다른 변수에 의해 상관관계가 나타났을 수도 있다.

예를 들어, 삼성전자 주가와 한미약품 주가를 산점도로 그려봤는데 양의 상관관계가 나타났다면, 산점도를 그리기 위해 선택한 기간만 우연히 상관관계가 나타났을 수도 있다. 또한, 오랫동안 상관관계가 유지된다면 상관관계가 있다고는 이야기할 수 있으나 삼성전자 주가 때문인지 한미약품 주가 때문인지는 산점도만으로는 판단하기 어렵다.

산점도 행렬

산점도는 기본으로 2개 변수의 값을 그래프에 표현하는 것으로, 여러 개의 변수가 있다 하더라도 변수 2개씩 짝을 지어 그래프를 그려야 한다. 머신러닝에서는 사용하는 변수의 수가 수십 개에 이르는 경우가 일반적인데, 이들 모든 변수에 대한 산점도를 일일이 지정해서 그리는 것은 매우 번거로운 일이다.

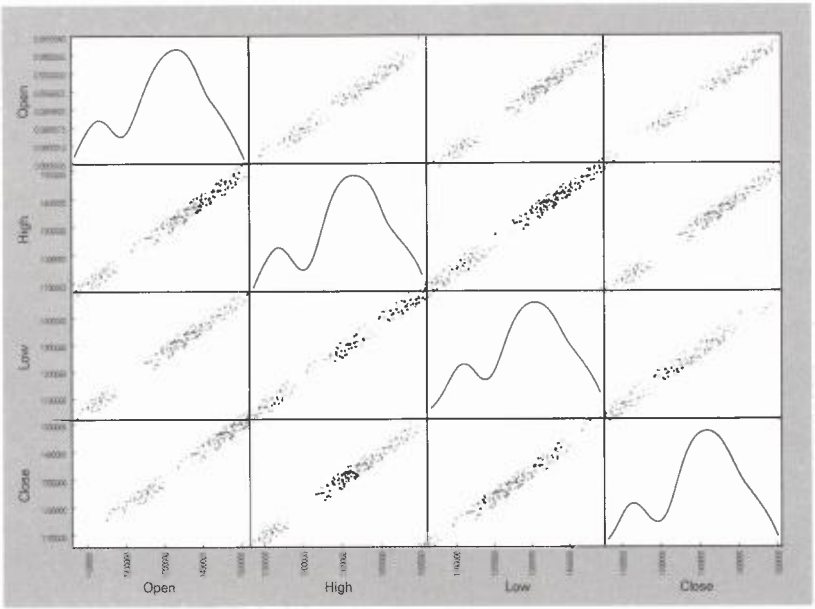
pandas에서는 다수의 변수에 대한 산점도를 한 번에 그려주는 scatter_matrix( )라는 함수를 제공한다. 이 함수를 이용하면 한 번에 모든 변수 간의 산점도를 그려주어서 매우 편리하게 사용할 수 있다.

앞서 받아온 삼성전자 주가 데이터에는 모두 6개의 변수가 있다. 이 변수 중 가격과 연관된 시가(Open), 고가(High), 저가(Low), 종가(Close) 4개 변수 간의 상관관계를 scatter_matrix( ) 함시수를 이용해 그려보자.

```
from pandas.tools.plotting import scatter_matrix
df = load_stock_data('samsung.data')
scatter_matrix(df[['Open','High','Low','Close']], alpha=0.2, figsize=(6, 6),
diagonal='kde')
plt.show()
```

앞의 코드를 실행하면 다음과 같은 산점도 행렬이 나타난다.

그림 2-17 삼성전자 주가 산점도 행렬



그림에서 알 수 있듯이 Open, High, Low, Close, 4개의 변수는 뚜렷한 패턴을 보여주며 강한 양의 상관관계가 있음을 쉽게 파악할 수 있다. Open 변수와 Close 변수의 상관관계를 파악하고 싶다면 화면 왼쪽 Y축에서 Close를 찾고 화면 아래쪽의 X축에서 Open을 찾으면 이 그래프가 Open과 Close 변수의 산점도다. 이와 같은 방식으로 다른 변수들의 산점도를 알 수 있다.

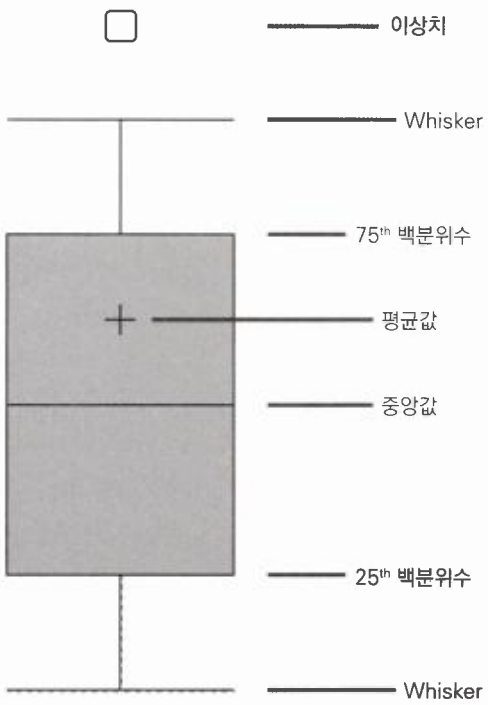
화면에서 대각선으로 나타난 분포는 변수별 분포도로 Y축 가장 위에 있는 Open 변수 옆의 그래프는 Open 변수의 분포도, 좀 더 자세히 말하면 커널밀도추정Kernel Density Estimation 그래프다.

커널밀도추정은 커널함수를 이용해 어떤 변수의 분포특성을 추정하는 것으로, 어떤 변수가 가질 수 있는 값의 범위와 해당 값을 가질 확률을 추정할 수 있게 한다.

2.7.6 상자그림

상자그림Box Plot은 중앙값, 평균값, 사분위, 이상치 값들을 한눈에 알아볼 수 있는 그래프로, 데이터의 전체 모습을 파악하는 데 매우 유용하게 사용할 수 있다. 먼저 상자그림의 구성을 다음 그림을 통해 파악해보자.

그림 2-18 상자그림



상자그림은 크게 5부분으로 구성되어 있다.

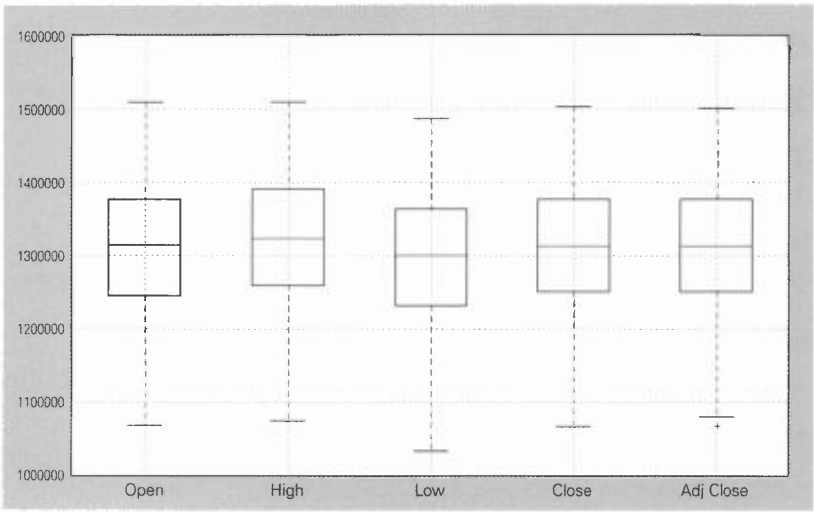
- **이상치** 양단의 Whisker에 속하지 않은 값으로 일반적인 데이터값이 아닌 값들이다.
- **Whisker** 하단의 Whisker는 'Q1 − 1.5 × IQR', 상단의 Whisker는 'Q1 + 1.5 × IQR'로 계산한 값이다.
- **Mean** 평균값
- **Median** 중앙값
- **Quartile** 25th **백분위수**Percentile는 사분위의 Q1이고 75th 백분위수는 Q3다.

pandas에서 상자그림을 지원하므로 매우 쉽게 그릴 수 있다. 예를 들어, 거래량 변수를 제외한 삼성전자 주가의 상자그림을 그리려면 다음과 같은 코드를 입력한다.

```
df = load_stock_data('samsung.data')
df[['Open','High','Low','Close','Adj Close']].plot(kind='box')
plt.show()
```

이 코드의 실행결과는 다음과 같다.

그림 2-19 삼성전자 주가 상자그림



이 그림을 보면 Low 변수의 값들이 다른 변수보다 대체로 낮은 값을 가지고 있고, Close 변수와 Adj Close는 중앙값과 Q1, Q3가 거의 같음을 알 수 있으며, 모든 변수에 이상치는 없다는 것을 알 수 있다.

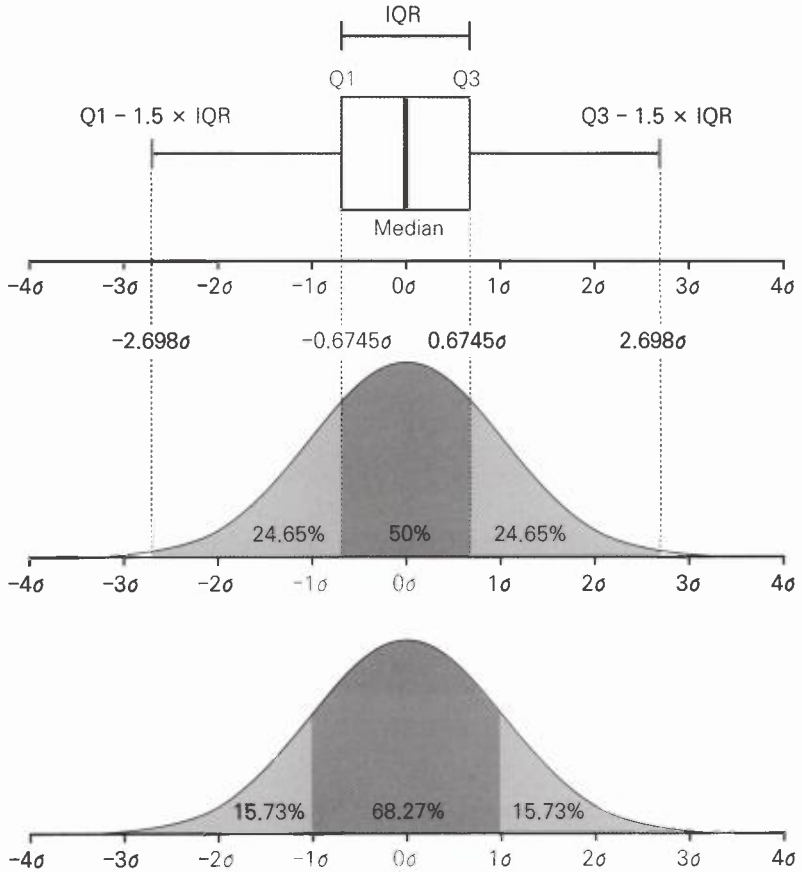
상자그림과 분포

상자그림은 데이터의 분포를 짐작하는 데도 유용하게 사용할 수 있다. 앞서 상자그림의 구성요소에서 살펴봤듯이 상자의 크기는 사분위의 Q1과 Q3에 의해 결정

되므로 전체 데이터 50%의 분포를 설명할 수 있는 IQR을 의미하고, 그 박스 안의 회색선은 중앙값을 나타낸다. 그리고 정규분포라고 가정하면, 2개의 Whisker는 나머지 상위와 하위 24.65%를 표시한다. 따라서 박스의 크기와 Whisker의 길이는 분포도를 능히 짐작할 수 있다.

다음 그림은 정규분포를 가정하고 상자그림과 분포를 그림으로 나타낸 것으로, 앞서 설명한 개념을 효과적으로 표현하고 있다.

그림 2-20 상자그림과 분포



chapter 3

시계열 데이터

알고리즘 트레이딩을 하기 위해서는 주가 데이터의 움직임을 잘 표현할 수 있는 모델을 만들어야 한다. 만들어진 모델은 주가의 움직임에 영향을 미치는 요소들Feature을 포함하고 있어야 하고, 이 요소 간의 관계 역시 수학적인 형태로 기술되어야 한다. 이러한 수학적 모델이 얼마나 잘 만들어졌느냐에 따라 주가 예측의 정확도가 결정되므로 좋은 모델을 만드는 것은 성공적인 알고리즘 트레이딩을 위한 아주 중요한 요소다.

하지만 알고리즘 트레이딩에 적용할 만한 수학적 모델을 만드는 것은 많은 노력과 시간, 이를 수행할 고급인력이 필요하므로 쉽게 만들기는 어렵다. 이런 측면에서 보면 사람이 직접 모델을 만들지 않아도 되는 머신러닝은 알고리즘 트레이딩에서 큰 강점이 있다고 이야기할 수도 있다. 적절한 데이터와 알고리즘을 선택해 실행하면 수많은 계산을 통해 알고리즘 내의 모델을 스스로 완성하므로 수학적 모델을 만드는 것보다 상대적으로 적은 노력과 비용으로 가능하다. 그러나 실제 금융계에서는 머신러닝이 블랙박스라는 점 때문에 아직 많이 사용하고 있지는 않다. 주로 기존의 수학적 모델을 메인으로 사용하고, 머신러닝은 서브로 사용한다.

2장에서도 설명했듯이 머신러닝은 모든 문제를 해결할 수 있는 마법의 상자가 아니다. 아무 생각 없이 가지고 있는 데이터를 모두 머신러닝에 넣은 후 그 결과물을 보면 역시 아무 생각도 할 수 없을 것이다. 거듭 이야기하지만, 머신러닝에서도 'Garbage in, Garbage Out'의 원칙은 무너지지 않는다. 특히 주가와 같은 금융 시계열 데이터Financial Time Series를 다룰 때는 머신러닝을 이용하더라도 전적으로

여기에 의지하기보다는 일정 수준 이상의 수학적 모델을 반드시 같이 활용하는 것이 필요하다.

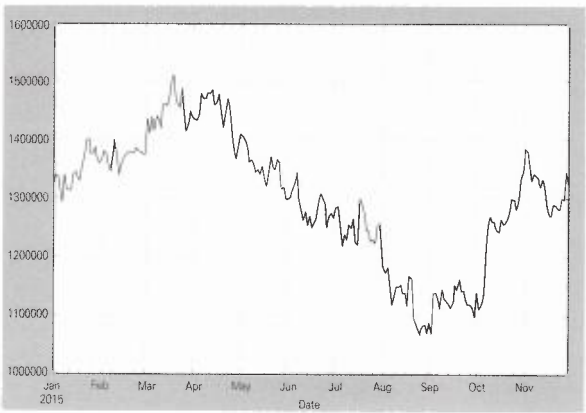
모델을 만드는 데 가장 우선으로 시작해야 할 것은 확보 가능한 데이터가 무엇인지를 파악하고, 데이터 내의 변수 간의 관계나 시간에 따른 변화를 파악하는 것이다. 자기상관Autocorrelation이나 자기공분산Autocovariance 같은 방법을 이용하면 시간에 따라 값이 어떻게 영향을 미치고, 그에 따른 결과가 어떻게 변화하는지를 알아내는 데 많은 도움이 된다.

이러한 분석을 통해 단순히 변수 간의 관계뿐만 아니라 시계열 데이터의 전체적인 구조를 추론하면 데이터에 대한 더 높은 이해도를 통해 예측의 정확성을 향상하게 하는 유용한 모델을 만들 수 있다. 이를 위해 필요한 개념과 방법들을 이번 장에서 살펴본다.

3.1 시계열 데이터

시계열 데이터Time Series는 일정 시간 간격으로 측정된, 순서를 갖는 데이터를 일컫는다. 즉, 시간의 흐름에 따라 값이 변하는 데이터들이 시계열 데이터인데, 예를 들어 주가 데이터, 온도 데이터, 상품판매량, 환율 데이터 등이 있다.

그림 3-1 시계열 데이터 예(삼성전자 주가)



시계열 데이터는 시간의 흐름에 따라 변수가 어떻게 변화하는지를 보여주는 것으로, 데이터에 순서가 있고, 그 값이 지속적으로 변화한다. 이는 시계열 데이터가 시간이라는 독립변수에 의해 변화하는 값, 즉 종속변수와의 관계를 보여주는 것이라고 생각할 수 있다.

시계열 데이터가 아니라면 데이터를 처리할 때 데이터의 순서를 바꾸거나 필요에 따라 재정렬해서 사용해도 무방하지만, 시계열 데이터는 시간의 변화에 따른 변수의 변화량을 의미하므로 순서를 뒤집거나 무작위로 데이터를 추출해 사용할 수 없다. 그리고 시간에 따른 종속변수의 변화량을 측정해야 해서 일정한 간격으로 측정해야 한다는 점도 중요하다. 시계열 데이터를 수집할 때는 반드시 동일한 조건과 방법으로 측정해 관측한 데이터가 객관성을 가질 수 있도록 해야 한다. 예를 들어, 동일한 측정방법이라 하더라도 측정 시간이 규칙적이지 않고 상황에 따라 1분 간격, 5분 간격, 10분 간격 등으로 다르다면 데이터의 객관성이 깨질 수 있다.

3.2 시계열 데이터 분석

시계열 데이터의 패턴을 파악하거나 패턴에 큰 영향을 미치는 요소를 찾는 등의 작업을 '시계열 데이터 분석Time Series Analysis'이라고 한다. 이 책의 목표인 알고리즘 트레이딩은 시계열 데이터인 주가 데이터를 다루기 때문에 시계열 데이터 분석이 매우 중요한 부분을 차지하고 있고, 앞으로 구축할 알고리즘 트레이딩 모델 역시 이 시계열 데이터 모델을 적용한다.

시계열 데이터 분석의 대표적인 목적은 다음과 같다.

- 시계열 데이터 패턴에 영향을 미치는 요소Feature를 찾는다.
- 과거의 데이터가 어떻게 미래의 데이터에 영향을 미치는지를 분석한다.
- 미래의 데이터를 예측한다.

이 책에서 시계열 데이터를 분석하는 궁극적인 목적은 미래의 데이터를 예측하는

것이다. 미래의 데이터를 예측하려면 시계열 데이터가 변화하는 패턴을 찾아야 하고, 발견된 패턴에 영향을 주는 요소를 추출하고 이 요소 간의 관계를 정립하는 모델을 만들어야 한다. 따라서 시계열 데이터 분석의 가장 중요한 목적은 모델을 만들기 위한 패턴과 요소를 찾는 것이라 정리할 수 있다.

3.3 주요 시계열 데이터의 특성

시계열 데이터를 다룰 때 내가 다루는 데이터가 어떤 시계열적 특성이 있는지를 파악하는 것은 매우 중요하다. 여러 수학자의 노력으로 이미 많은 모델이 개발되었고 지금도 개발되고 있어서 관심 있는 시계열 데이터의 특성을 명확히 파악할 수 있다면, 이에 걸맞는 모델을 적용해 좋은 결과를 가져올 수 있다. 예를 들어, 주말과 평일에 따라 데이터가 특정한 패턴을 가지고 있다면, 계절성Seasonality이 있다고 할 수 있고, 이런 특성을 가지고 있다는 것이 확인되면 Seasonal ARIMA 같이 검증된 모델을 적용할 수 있다.

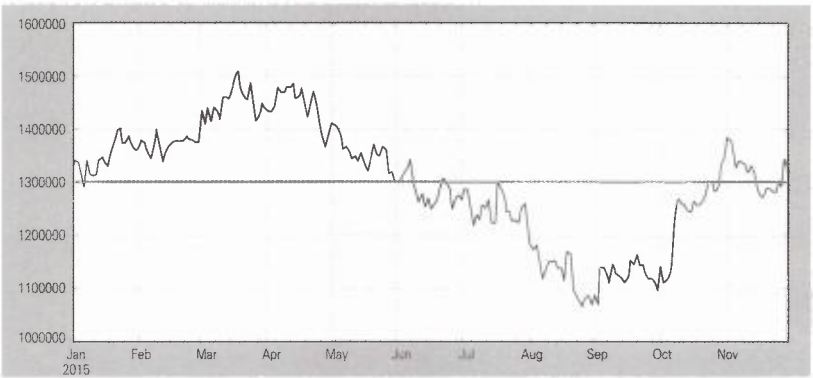
기억해야 할 시계열 데이터의 특성들은 다음과 같다.

- **Trend** 측정값이 시간의 흐름에 따라 증가나 감소 또는 반복 등의 일정한 패턴이나 경향을 가지고 있는가?
- **Seasonality** 일, 월, 년, 계절 등 일정 시간에 따라 지속해서 반복되는 패턴이 있는가?
- **Outliers** 다른 값들과 동떨어진 이상치를 관측할 수 있는가?
- **Long-run Cycle** 계절성과는 별도로 오랜 기간 반복되는 패턴이 있는가?
- **Constant Variance** 측정값이 일정한 수준 이내로 변동되는가 아니면 변동이 무작위로 발생하는가?
- **Abrupt Change** 급격한 변동을 보이는 데이터가 있는가?

시계열 데이터의 특성을 볼 때 데이터의 범위 또한 중요하다고 할 수 있다. 동일한 데이터라 하더라도 범위에 따라 일정 패턴이 보일 수도 있고 그렇지 않을 수도 있기 때문이다.

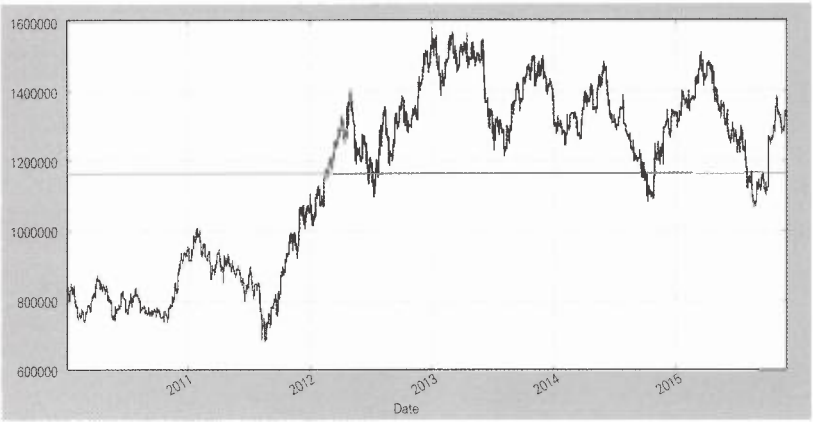
다음 그래프는 2015년 1월부터 11월까지의 삼성전자 주가로, 가운데 회색선으로 표시된 평균값을 기준으로 주가가 움직이는 것처럼 보인다. 그래프를 보면 1월부터 4월까지 상승하고 4월부터 9월까지 하락하다가 10월부터 다시 상승하는 모습을 보여주는데, 평균값은 1,301,852이고 Q1이 1,245,000, Q3가 1,376,500으로 변동폭은 크지 않다고 볼 수 있다. 2015년 삼성전자의 1분기 실적 발표일은 4월 29일, 2분기는 7월 7일이었는데 주가 그래프로 보면 4월 이후 주가가 하락하고, 7월 이후로는 하락폭이 더 커진것을 알 수 있어 계절성이 있다고 판단할 수 있다.

그림 3-2 삼성전자 주가(2015년 1월~11월)



다음 그래프는 삼성전자의 2010년~2015년의 주가 그래프를 보여주는 것으로 2015년 주가 그래프와는 다른 양상을 보여준다. 그래프상으로는 2012년을 전후해 주가가 큰폭으로 상승했고 그 이후에는 등락폭을 보이며, 2010년~2015년 평균치인 1,165,100을 상회하고 있음을 알 수 있다. 앞서 본 그래프와는 명백히 다른 패턴을 보여준다.

그림 3-3 삼성전자 주가(2010년~2015년)



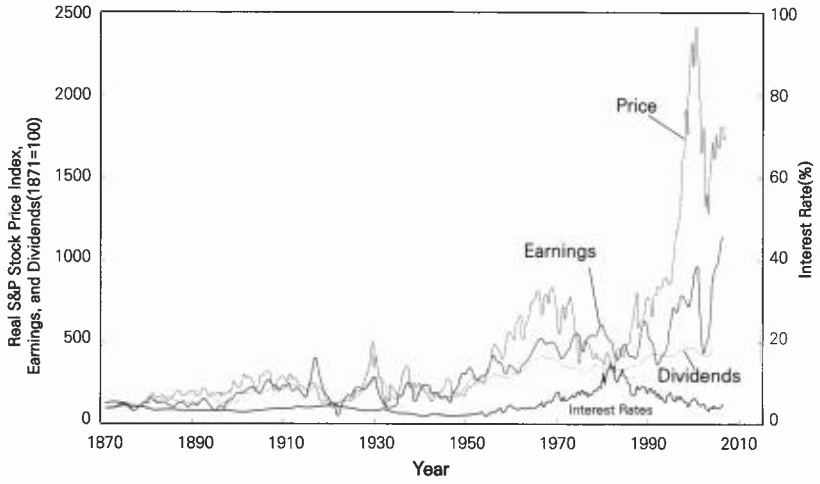
앞의 두 그래프에서 알 수 있듯이 시계열 데이터는 데이터 범위에 따라 동일한 데이터라 하더라도 패턴이나 트렌드를 전혀 다르게 인식할 수 있다는 것을 유의해서 분석해야 한다.

3.4 랜덤과정

랜덤과정Stochastic Process은 확률변수Random Variable가 시간의 흐름에 따라 변화한 값들을 일컫는다. 확률변수는 이름이 의미하는 특정한 값으로 귀결되거나 일정한 패턴을 보이지 않기 때문에 랜덤과정을 시간의 흐름에 따른 확률분포Probability Distribution라고도 생각할 수 있다.

랜덤과정의 반대 개념은 결정적 과정Deterministic Process으로, 명칭이 뜻하는 것처럼 모든 것이 결정되어 있다. 결정적 과정은 시간에 흐름에 대해 일정한 값을 갖지만, 랜덤과정에서는 무작위로 결정되어서 시간의 흐름에 따른 값의 변화가 일정하지 않다. 랜덤과정은 임의과정Random Process이라고도 하며 일정한 규칙 없이 움직이는 주가와 환율 같은 것이 대표적인 예다.

그림 3-4 주가 그래프[01]



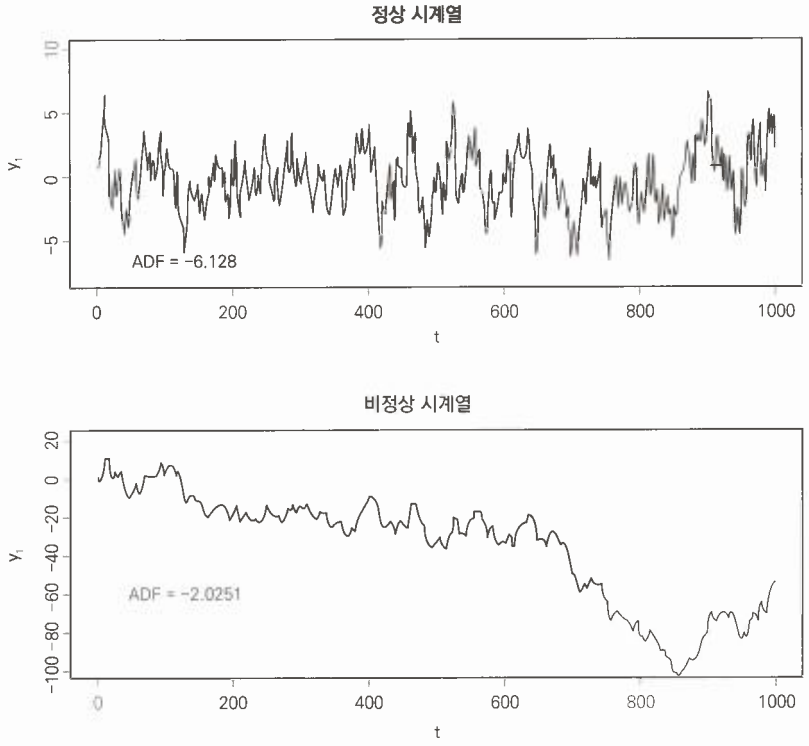
3.5 정상 시계열 데이터

정상성Stationarity은 평균과 분산 같은 통계적 특성이 시간에 대해 일정한 성질을 의미하고, 이러한 특성이 있는 랜덤과정을 정상과정Stationary Process이라고 한다. 정상과정은 우리가 살고 있는 현상계에서 발생하는 여러 가지 데이터를 설명할 수 있어서 자연과학이나 공학, 사회과학 등에서도 광범위하게 사용되는 중요한 개념이다.

[그림 3-5]는 정상과정 시계열 데이터와 비정상과정 시계열 데이터Non-Stationary Time Series을 비교한 것이다. 첫 번째 그래프는 데이터가 증가하거나 감소하지만, 평균값인 0을 기준으로 움직이는 패턴을 보이고, 데이터의 변동 또한 +5에서 -5 사이로 일정 수준을 넘지 않는 것을 확인할 수 있다. 두 번째 그래프는 평균값을 중심으로 데이터가 움직이지 않고, 변동 폭이 일정하지 않아 첫 번째 그래프와 같은 성질을 찾기 어렵다.

01 출처: By Frothy, CC BY-SA 4.0-3.0-2.5-2.0-1.0, https://goo.gl/qIf0tl

그림 3-5 정상과정 시계열 데이터 vs 비정상과정 시계열데이터



다뤄야 하는 데이터가 정상 시계열 데이터라는 것이 밝혀지면 행운이다. 데이터가 무작위로 움직이는 것이 아니라 일정한 통계적 특성에 따라 움직이기 때문에 기존의 알려진 시계열 데이터 모델을 사용하거나 직접 모델을 만들기만 하면 된다.

정상성은 주식과 같은 금융 시계열 데이터에서 많이 활용하고 있으며, 각종 수학적 이론을 제공하는 등 매우 중요한 위치를 차지하고 있다. 일반적으로 주식은 정상성이 없다고 알려졌음에도 불구하고 주가 데이터를 분석하는 데 사용하는 방법이 정상성과 연관된 것들이 많기 때문이다.

3.5.1 약한 정상성

정상성에는 약한 정상성Weak-sense Stationarity과 강한 정상성Strong-sense Stationarity이 있다. 약한 정상성은 'Wide-sense stationarity' 또는 'Second-order stationarity'라고도 하며 다음과 같이 정의할 수 있다.

> *평균함수Mean Function $m(t)$와 공분산함수Covariance Function $r(s,t)$가 시간에 대해 변하지 않는 성질을 '약한 정상성'이라고 하고, 이러한 과정을 '약한 정상성 과정Weak-sense Stationary Process'이라고 한다.*

앞의 정의에 따르면 약한 정상성은 다음과 같은 특성을 가진다.

- 일정한 평균Constant Mean
- 일정한 분산Constant Variance
- 시간에 독립적인 공분산

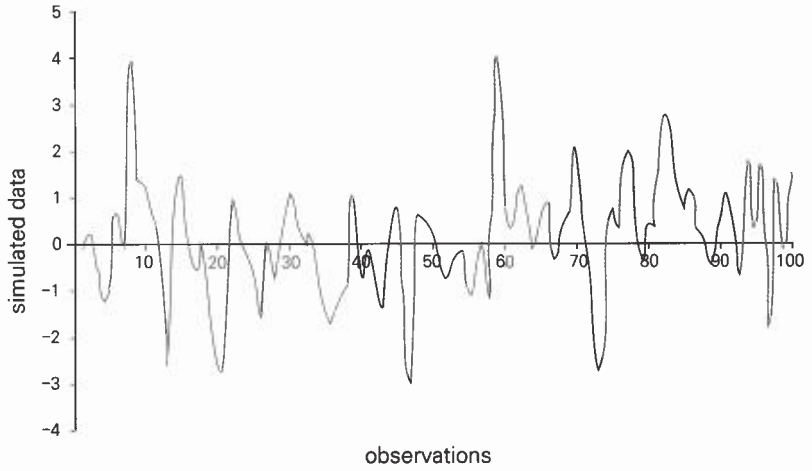
이 특성을 만족하려면, 어떤 랜덤과정이 있고 그 과정을 함수 $f(t)$라고 할 때 시간 t_1에 대해 다음과 같은 관계가 성립해야 한다.

$$f(t_1) = f(t_1+\tau)$$

이 수식은 t_1이라는 시간에 대한 $f(t_1)$값과 시간을 τ 만큼 이동시킨 시간에 대한 값 $f(t_1+\tau)$이 동일한 값을 가진다는 것을 의미하는데, 이는 앞서 설명한 것처럼 시간의 변화에 상관없이 일정한 통계적 특성을 가져야만 성립할 수 있다.

[그림 3-6]은 약한 정상성 시계열 데이터의 예로, 평균이 0이고 일정한 분산을 보여주고 있다.

그림 3-6 약한 정상성 시계열 데이터 예



3.6 랜덤과정에서의 기대값, 분산, 공분산

기대값Expectation과 분산Variance은 앞 장에서 살펴본 통계의 개념과 동일하며, 여기서는 랜덤과정에서 수식으로 어떻게 표현하는지를 알아보자.

기대값은 다음과 같이 표현하는데, E(x)는 모집단에서의 확률변수 x의 기대값이고 μ는 모집단 평균이다.

$$E(x) = \mu$$

분산은 데이터가 평균을 중심으로 얼마나 퍼져있는지를 나타내주는 수치로, 확률변수의 분산 정도(Spread)를 알 수 있다.

$$\sigma^2(x) = E[(x-\mu)^2]$$

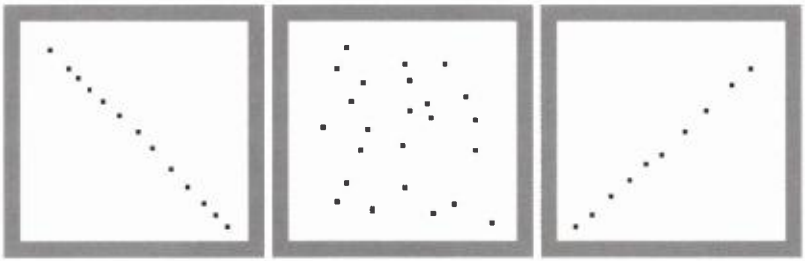
표준편차는 분산에 루트를 취해 데이터와 단위를 일치시킨 것으로, 분산을 구하면 간단히 계산할 수 있다.

3.6.1 공분산

공분산Covariance은 2개 변수의 상관정도를 나타내는 값으로, Cov(x,y)로 표기한다. 공분산을 계산한 수치는 2개 변수의 방향성을 파악하는 용도로 활용하며, 밀접한 정도를 나타내는 것은 아니다. 예를 들어, 공분산 값이 0보다 크면 2개의 변수가 모두 증가하는 경향을 보이고, 0보다 작다면 1개의 변수는 증가, 다른 1개의 변수는 감소하는 경향을 보이는 것을 알 수 있다. 하지만 공분산 값이 하나는 10, 다른 하나는 100이라고 할 때, 수치가 큰 100을 보여준 변수들이 더욱 상관관계가 높다고 얘기할 수는 없다.

다음 그림은 공분산 값에 따른 관계를 보여준다.

그림 3-7 공분산



큰 음의 공분산 | 0에 가까운 공분산 | 큰 양의 공분산

공분산 값의 해석 방법을 정리하면 다음과 같다.

- **Cov(x,y) > 0** x, y 변수가 모두 증가한다.
- **Cov(x,y) < 0** x가 증가하면 y는 감소한다. 또는 그 반대다.
- **Cov(x,y) = 0** x, y는 선형적인 관계가 없으며, 독립적인 관계에 있다.

모집단이 아닌 표본에서 공분산을 구하는 수식은 다음과 같다.

$$Cov(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

x, y는 공분산을 구하고자 하는 변수고, $\bar{x}$는 x의 표본 평균, $\bar{y}$는 y의 표본 평균이다. 이 식에서 표본의 데이터 수 n으로 나누지 않고 n-1을 사용한 이유는 공분산을 실제값과 추정값의 차이가 없는 비편향추정량으로 만들어주기 때문이다.

참고로 전체집합인 모집단에서 추출한 표본의 기대값이 다른 것을 '추정량의 편향Bias of an Estimator'이라고 하는데, 차이가 있는 것을 '편향추정량Biased Estimator', 차이가 없는 것을 '비편향추정량Unbiased Estimator'이라고 한다.

pandas를 이용해 삼성전자 종가와 한미약품 종가의 공분산 값을 구해보자. 다음 코드를 작성해 실행한다.

```
df_samsung = load_stock_data('samsung.data')
df_hanmi = load_stock_data('hanmi.data')
print df_samsung['Close'].cov(df_hanmi['Close'])
```

실행결과

```
-7384068478.53
```

공분산 값이 매우 큰 음수가 나왔는데, 앞서 설명한 것처럼 공분산의 해석에 있어 크기가 중요한 것이 아니라 방향성을 나타내는 부호가 중요하다. 음수이므로 삼성전자 종가와 한미약품 종가는 서로 다른 방향성을 가지고 있는 것을 알 수 있다.

3.7 상관

상관Correlation은 2개 변수 간에 어떤 선형적인 관계가 있는지를 분석하는 방법이다. 좀 더 자세하게 얘기하면 2개 변수의 공분산을 정규화한 것으로, 변수 간의 상관관계의 정도를 상관계수Correlation Coefficient로 나타낸다.

상관관계는 Cor(x,y)로 표시하며 다음과 같이 정의한다.

$$Cor(x,y)=\frac{\mathrm{Cov(x,y)}}{\mathrm{std(x)*std(y)}}$$

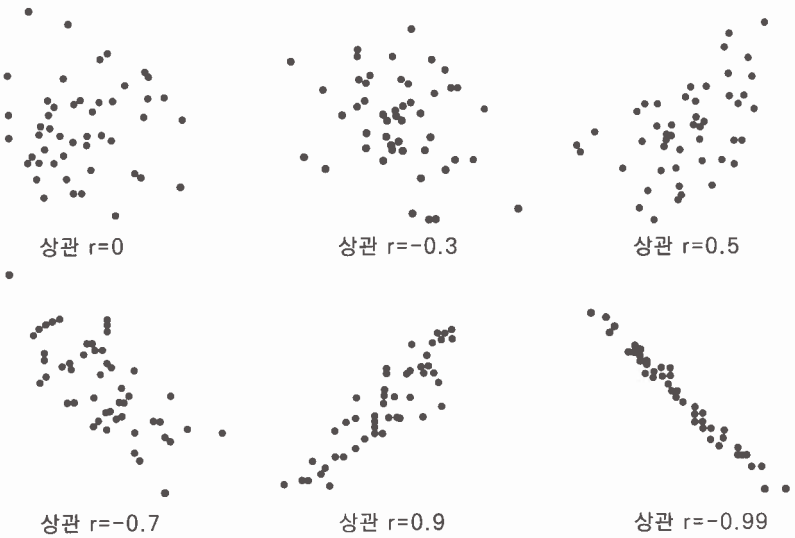
- std(x)는 표본변수 x의 표준편차를 나타내고, std(y)는 표본변수 y의 표준편차다.
- Cor(x,y)는 공분산을 정규화한 것이므로 [-1,1] 사이의 값을 가진다.

공분산과 달리 상관관계 값은 방향성과 함께 상관관계의 정도를 나타낸다. 값이 1에 가까울수록 양의 강한 상관관계, -1에 가까우면 음의 강한 상관관계를 의미하며, 0인 경우는 선형적으로 상관관계가 없다는 것을 의미한다.

상관관계 값을 해석 방법을 정리하면 다음과 같다.

- Cor(x,y) > 0 양의 상관관계
- Cor(x,y) = 1 x와 y가 동일
- Cor(x,y) < 0 음의 상관관계
- Cor(x,y) = -1 x와 y가 반대 방향으로 동일
- Cor(x,y) = 0 x와 y는 선형적인 상관관계가 없음

그림 3-8 상관관계



상관관계를 나타내는 상관계수는 피어슨Pearson, 스피어만Spearman, 켄달Kendal 등 여러 가지가 있으나 가장 많이 사용하는 것은 피어슨 상관계수다. pandas를 이용해 삼성전자 종가와 한미약품 종가의 피어슨 상관계수를 구해보자. 코드는 다음과 같다.

```
df_samsung = load_stock_data('samsung.data')
df_hanmi = load_stock_data('hanmi.data')
print df_samsung['Close'].corr(df_hanmi['Close'])
```

실행결과

```
-0.366482003208
```

피어슨 상관계수가 -0.366이므로 삼성전자 종가와 한미약품 종가는 앞서 공분산에서 확인한 바와 마찬가지로 서로 다른 방향성을 가지고 있으며, 계수값이 절대값 0.3보다 크므로 뚜렷한 음적 선형관계가 있다고 할 수 있다. 일반적으로 상관계수의 절대값이 0.3보다 작으면 약한 관계로, 0.7보다 크면 강한 관계로 해석한다.

3.8 자기공분산

자기공분산Autocovariance은 약한 정상성 과정 변수의 공분산함수로 다음과 같이 표현할 수 있다.

$$C(k) = \frac{1}{n}\sum^{n-k}(x_t - \bar{x})(x_{t+k} - \bar{x})$$

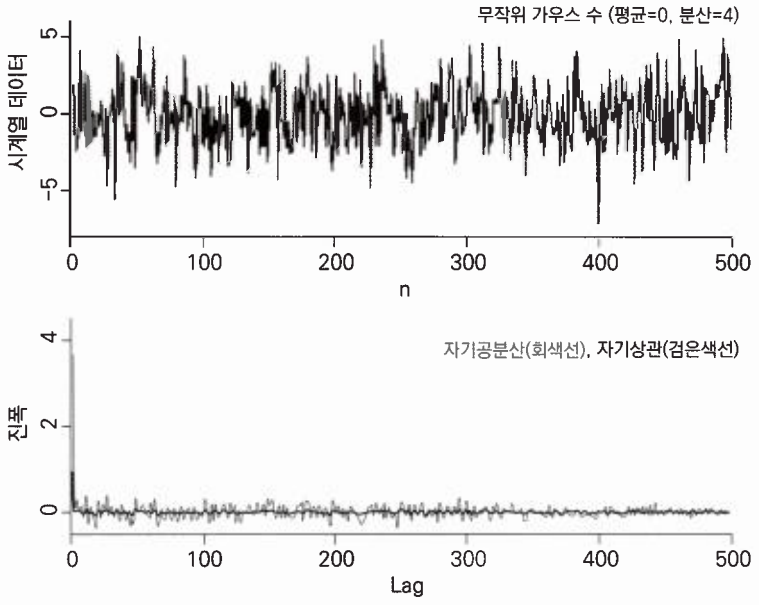
이 수식에서 C(k)는 Lag K에 대한 자기공분산함수를 지칭한다. Lag K는 x_t와 x_{t+k} 사이의 상관관계를 의미하는 것으로, 거칠게 말하면 2개의 값, x_t와 x_{t+k} 사이

의 공분산이다. Lag K를 구하는 함수인 C(k)의 식은 앞서 살펴본 공분산함수와 매우 유사한 형태를 보여준다.

$$Cov(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

공분산이 동일한 시간에서 2개 변수의 상관관계를 분석하는 것이라면, 자기공분산함수 C(k)는 서로 다른 2개의 시간, t와 t+k에 대한 변수 값의 공분산을 계산한다. 자기공분산함수의 주된 관심사는 시간에 따른 값들의 상관관계가 어떻게 되는가를 파악하는 것으로, 상관관계가 증가추세인지 하향추세인지, 상관관계의 변화폭은 얼마나 되는지를 살펴볼 수 있다.

그림 3-9 자기공분산함수



[그림 3-9]에서 첫 번째 그래프는 무작위로 생성한 가우스 숫자를 나타내며, -5부터 5까지 값이 변동하는 것을 알 수 있다. 두 번째 그래프의 회색선은 Lag K를 그

린 것으로, 상관관계가 증가나 감소 등의 추세가 보이지 않고 일정한 값에서 진동하며 시간이 지날수록 0에 수렴하는 경향이 있음을 알 수 있다. 이처럼 자기공분산함수는 시계열 데이터 Lag의 추세와 크기를 파악하는 데 유용하게 사용할 수 있다.

3.9 자기상관

자기상관Autocorrelation은 시계열 변수의 시간에 따른 자기상관관계를 나타내는 것으로 계열상관Serial Correlation 또는 교차자기상관Cross Autocorrelation이라고 한다. 상관이 특정 시간에 대한 변수 간의 상관관계라면, 자기상관은 시간의 변화에 따른 변수 간의 상관관계 변화가 주관심사다. 어떤 시계열 데이터가 일정한 패턴을 보인다는 것은 자기상관이 있다는 것을 의미하는 것으로, 시간에 따라 변수의 값이 자기상관성을 가지고 변화하므로 무작위가 아닌 일정한 패턴을 보여주는 것이다.

약한 정상성 과정의 Lag K를 구하기 위한 자기상관함수는 다음과 같다.

$$\rho(k) = \frac{C(k)}{C(0)}$$

이 수식을 보면 자기상관함수는 C(0)가 상수이기 때문에 자기공분산함수 C(k)에 의해 결정됨을 알 수 있다. C(0)는 첫 번째 공분산 값을 의미하므로 C(k) 함수를 초기값으로 정규화하는 의미로 해석할 수 있다. 따라서 x_t와 x_{t+k}의 상관관계를 나타내는 것임을 알 수 있다. 상관을 시계열 데이터로 확장한 것이 자기상관인만큼 기존 상관의 정의와 매우 유사하다.

자기상관함수는 또 다른 의미로도 해석할 수 있다. 앞서 설명한 바와 같이 자기상관함수의 값은 C(k)에 의해 결정되는데, C(k)를 단순화해 표현한 $(x_t - \bar{x})(x_{t+1} - \bar{x})$ 관계에서 보듯이 x_t와 x_{t+1}의 다음과 같은 비례관계로 결정된다는 것을 알 수 있다.

t 시간을 기준으로 하면 t보다 미래인 t+1 시간의 값을 Δ와 X_t로 예측 가능하다.

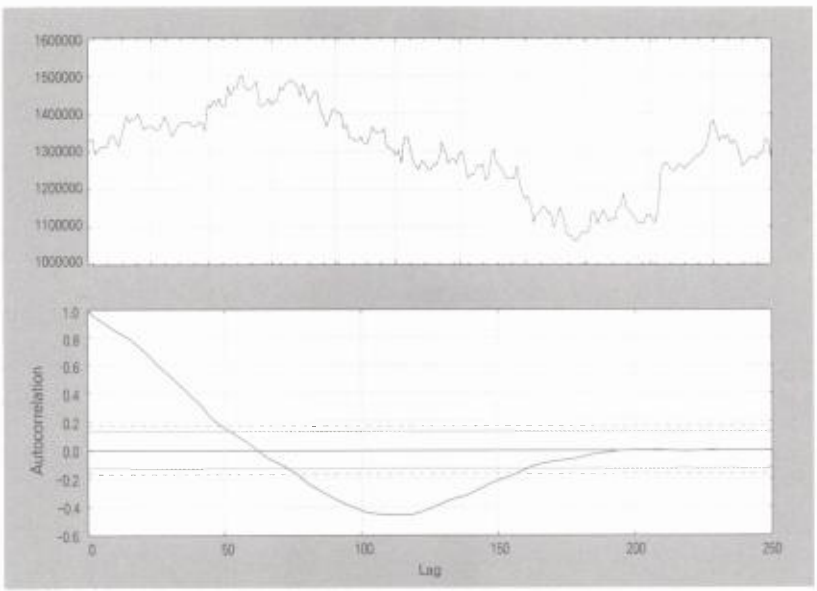
$$X_{t+1} = \Delta X_t$$

자기상관을 이용하면 해당 데이터의 무작위성Randomness를 파악할 수 있다. 자기상관 그래프를 그렸을 때 데이터가 0에 가까울수록 무작위성이 있는 시계열 데이터로 판단할 수 있고 0보다 큰 값을 가질수록 자기상관을 강하게 가진다고 할 수 있다. 자기상관은 알고리즘 트레이딩에서 매우 중요한 이론적 바탕이므로 확실히 이해해야 하는 중요한 개념이다.

pandas를 이용해 삼성전자 종가의 자기상관 그래프를 그려보자.

```
from pandas.tools.plotting import scatter_matrix,autocorrelation_plot

fig, axs = plt.subplots(2,1)
df_samsung['Close'].plot(ax=axs[0])
autocorrelation_plot(df_samsung['Close'],ax=axs[1])
plt.show()
```

그림 3-10 실행결과 - 삼성전자 종가의 자기상관 그래프



이 그래프를 보면 위쪽 그래프는 삼성전자 주식의 종가고, 아래쪽 그래프는 자기상관 그래프인데, 삼성전자 종가는 시간이 흘러감에 따라 0을 향해가는 경향을 보여주므로 무작위성이 있다는 것을 알 수 있다.

상관도표

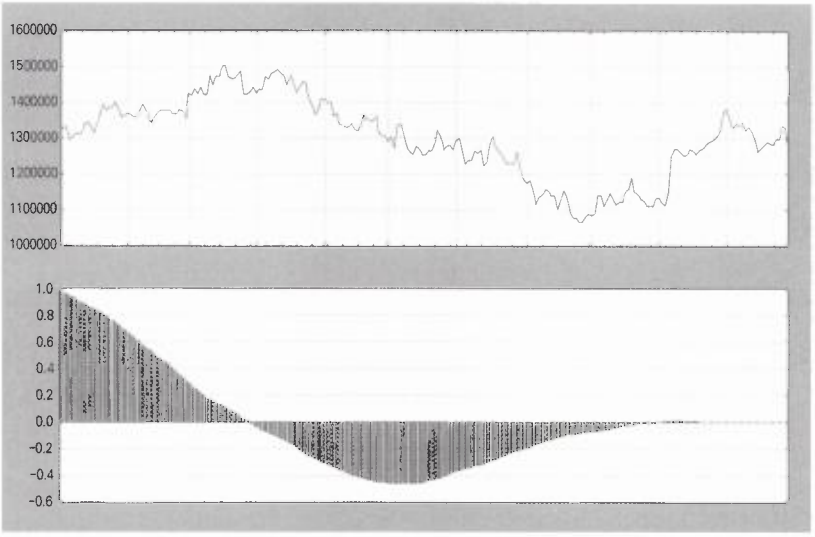
상관도표Correlogram는 자기상관함수를 Lag K의 순차적 값에 따라 그린 그래프로, '자기상관도Autocorrelation plot'라고도 한다. 자기상관 그래프와 동일하며 그래프 형태가 막대여서 선보다 크기를 시각적으로 잘 표현하므로 각각의 Lag 구조를 파악하는 데 도움을 주며, 당연하게도 자기상관 그래프처럼 무작위성을 파악하는 데도 유용하다. 알고리즘 트레이딩에서는 상관도표를 계절 효과인 계절성이나 결정적인 추세가 있는지를 감지하기 위해 많이 사용한다.

삼성전자 종가를 상관도표로 그려보자.

```
def get_autocorrelation_dataframe(series):
    def r(h):
        return ((data[:n - h] - mean) * (data[h:] - mean)).sum() / float(n) / c0

    n = len(series)
    data = np.asarray(series)
    mean = np.mean(data)
    c0 = np.sum((data - mean) ** 2) / float(n)
    x = np.arange(n) + 1
    y = lmap(r, x)
    df = pd.DataFrame(y, index=x)
    return df

df_samsung = load_stock_data('samsung.data')
df_samsung_corr = get_autocorrelation_dataframe(df_samsung['Close'])

fig, axs = plt.subplots(2,1)
axs[1].xaxis.set_visible(False)

df_samsung['Close'].plot(ax=axs[0])
df_samsung_corr[0].plot(kind='bar',ax=axs[1])

plt.show()
```

이 코드에서 get_autocorrelation_dataframe은 인자로 전달된 DataFrame에서 자기상관 값을 계산하고 이를 DataFrame으로 다시 돌려주는 함수다.

그림 3-11 실행결과 – 삼성전자 종가 상관도표



앞서 본 자기상관 그래프와 형태만 다를 뿐 동일한 그래프라는 것을 확인할 수 있다.

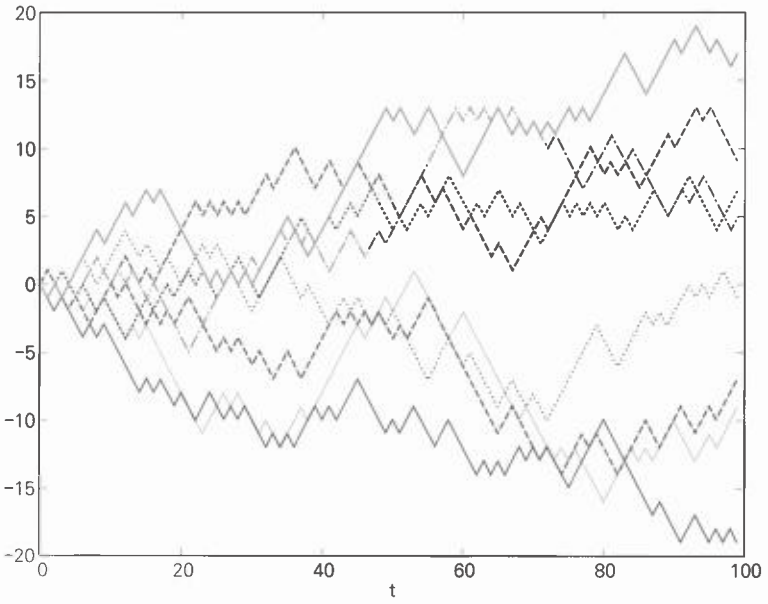
3.10 랜덤워크

랜덤워크Random Walk는 이전 행보와 독립적인 무작위 행보가 임의 방향으로 진행되는 것을 일컬으며 이산변수와 연속변수에 모두 사용할 수 있다. 예를 들어, 술에 잔뜩 취한 사람이 집을 향해 걸어가는 모습을 보면 앞으로 갈지 뒤로 갈지 또는 좌우로 갈지를 도저히 가늠할 수 없는 데, 이런 것을 '랜덤워크'라고 한다. 액체나 기체에서의 분자 운동, 주가의 움직임, 동전 던지기와 같은 것이 랜덤워크의 일종이며, 생태학, 경제학, 심리학, 물리학, 화학 등 그야말로 다양한 분야에서 폭넓게 활용되고 있다.

루이 베를리오즈Louis Bachelier가 1900년에 박사논문으로 발표한 「La Théorie de la Spéculation」에서 랜덤워크를 금융 시계열 데이터 모델로 제안한 후 많이 활용되기 시작했으며, 알고리즘 트레이딩에서도 역시 중요한 개념으로 다양한 이론과 방법에 적용되고 있다.

랜덤워크는 한 지점에서 다음 지점까지의 거리가 일정해 평균은 일정하지만 방향성은 무작위로 결정되어서 분산은 시간이 지남에 따라 증가하는 모습을 보이는 특징이 있다.

그림 3-12 랜덤워크



랜덤워크에는 표류 없는 랜덤워크Random-walk-without-drift와 표류하는 랜덤워크Random-walk-with-drift 2가지 모델이 있으며 이들의 차이는 분산의 변동성이다. 표류 없는 랜덤워크는 분산이 일정하지만, 표류하는 랜덤워크는 분산이 시간이 지남에 따라 확대되는 모델이다.

예를 들어, 술에 만취한 사람이 걷는 것을 랜덤워크라고 하면, 표류 없는 랜덤워크는 만취한 사람이 보폭을 일정하게 걷는 것이고, 표류하는 랜덤워크는 보폭이 일정하지 않은 것이라고 이야기할 수 있다.

3.10.1 기하적 브라운 운동

기하적 브라운 운동GBM, Geometric Brownian Motion 또는 지수 브라운 운동Exponential Brownian Motion은 랜덤워크의 일종으로, 표류drift 브라운 운동을 하는 랜덤과정이다. GBM은 다음과 같이 정의한다. 여기서 W(t)는 표준 브라운 운동이고, μ는 평균, z_0는 계수를 의미한다.

$$X(t) = z_0 \exp\left(\mu t + \sigma W(t)\right)$$

이 식에서 알 수 있듯이 임의변수의 값은 평균과 브라운 운동에 의해 결정된다. 일반적으로 GBM에서는 표류하는 랜덤워크 모델을 사용하는데, 이에 내포된 중요한 의미를 되새김해 볼 필요가 있다.

시간이 n에서 k만큼 지나 n+k가 될 때 표류가 불규칙하게 발생하고 정규분포를 따르는 것으로 가정하면 다음과 같은 식이 성립한다. r은 표류를 로그 단위로 표현한 것으로, 한 번의 움직임에 대한 비율증가값과 유사하다.

$$\ln(\hat{y}_{n+k}) = \ln(y_n) + kr$$

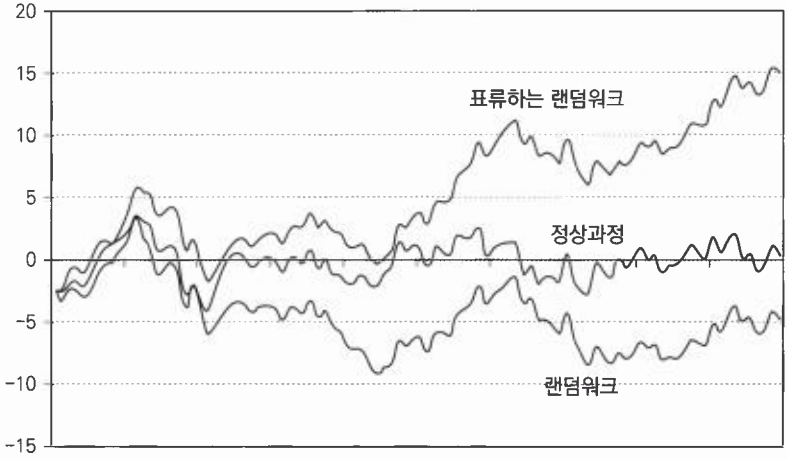
예를 들어, r = 0.0091이라고 하면 값의 변화 크기가 0.91%라는 것을 나타낸다. 이 식에서 자연 로그를 없애고 다시 기술하면 다음과 같고, 이 식에서는 r의 값에 따라 그래프의 기울기가 결정된다.

$$\hat{y}_{n+k} = y_n(1+r)^k$$

지수로 표현되는 식이어서 음수가 나올 수 없고, r 값이 기울기를 결정하고 전체 확률공간의 크기가 1이 되므로 GBM은 경제학과 금융 등에서 폭넓게 활용된다.

표류 없는 랜덤워크 모델은 긴 추세를 가지고 있지 않고, 일정 수준의 변동성 Volatility을 보여주지만, 표류하는 랜덤워크 모델은 지수함수 형태라서 우상향의 뚜렷한 긴 추세를 가지고 있고 변동성 역시 표류 없는 랜덤워크 모델보다 크다.

그림 3-13 랜덤워크



Part 3

실제로 간단한 알고리즘 트레이딩 시스템을 파이썬을 이용해 구현해본다. 머신러닝에 기반을 둔 모델과 시계열 이론에 기반을 둔 모델 2가지를 구현해보고, 구현된 결과에 대한 해석과 개선방법에 대해 다룬다.

chapter 4

알고리즘 트레이딩

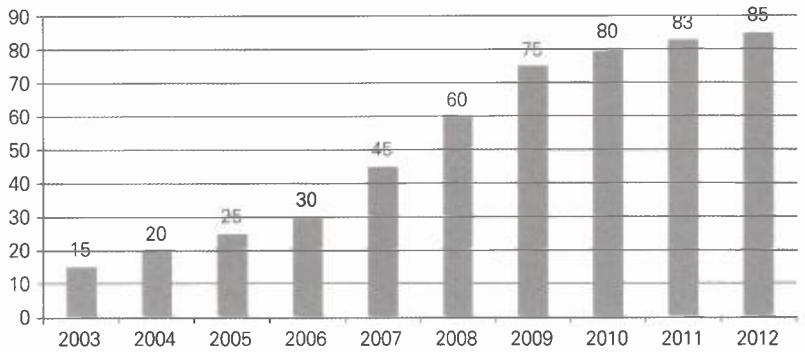
4.1 알고리즘 트레이딩 소개

알고리즘 트레이딩은 수학적 계산과 IT 시스템을 이용해 금융 상품을 거래하는 것으로 '시스템 트레이딩, Algo 트레이딩, Blackbox 트레이딩'이라고도 불린다. 알고리즘 트레이딩은 투자은행, 연기금, 헤지펀드, 증권회사 등 많은 곳에서 사용하고 있으며, 외국에서는 최근 몇 년 사이 수학적 지식과 IT 지식을 가진 개인들도 많이 참여하고 있다.

우리에게는 다소 생소할 수 있는 알고리즘 트레이딩은 이미 미국 금융시장의 모습을 많이 바꿔놓았다. 영화나 미국 드라마에서 보던 뉴욕증권거래소 객장의 모습은 그야말로 시골 장터처럼 활기가 넘치는 곳이었다. '트레이더'라고 부르는 사람들이 여기저기를 뛰어다니고 쉴 새 없이 전화로 통화하는 모습, 무슨 이유에서인지는 모르겠지만 환호성을 지르는 사람과 반대로 세상이 멸망한 것처럼 깊은 한숨을 내쉬는 사람들이 있는 그런 곳이었다.

하지만 2016년 현재 더는 이런 모습은 찾아볼 수 없고 적막만이 감도는 조용한 곳이 되었다. 대략 2007년부터 알고리즘 트레이딩의 확산에 따라 객장에서 거래를 사람이 아닌 기계가 대신하기 시작했다. 자료에 따르면 미국의 경우 2012년 알고리즘 트레이딩에 의한 거래량이 85%에 달할 만큼 알고리즘 트레이딩은 가파르게 증가해 왔다.

그림 4-1 연도별 알고리즘 트레이딩 비중 변화[01]



미국의 월스트리트는 2008년 금융위기 이후 시장환경의 악화로 이익창출이 어려워지고, 새롭게 도입된 각종 규제와 전 세계적인 저금리 기조가 맞물리면서 현 상황을 타개하는 데 많은 노력을 기울이고 있다. 새로운 이익창출과 비용절감, 특히 인건비를 줄이기 위한 대안으로 IT 기술을 적극적으로 활용하고 있으며, 알고리즘 트레이딩 역시 그 자구책의 하나다. 세계적 컨설팅업체인 맥킨지는 보고서에서 알고리즘 트레이딩과 같은 IT 기술을 도입하면 이익을 30% 정도 증가할 수 있다고 얘기했다. 이런저런 상황을 보면 앞으로 금융시장에서 알고리즘 트레이딩과 IT 기술들은 더욱 확고부동한 위치를 차지할 것으로 예상한다.

이처럼 알고리즘 트레이딩이 전 세계 금융의 중심지인 월스트리트에 성공적으로 안착할 수 있었던 여러 요인이 있지만, 가장 중요한 것은 단연코 수익을 내기 때문이다. 윌리엄 그로브William M. Grove, 데이비드 잘드David H. Zald 등이 작성한 「Clinical versus Mechanical Prediction: a meta-analysis」 논문[02]에 의하면 136건의 사례를 조사해보니 수학적 모델이 사람과 비슷하거나 사람보다 더 좋은 결과를 가져올 확률이 94%라고 한다.

01 출처: By Tertius51, CC BY-SA 3.0, https://goo.gl/1W32sq

02 참조: http://www.ncbi.nlm.nih.gov/pubmed/10752360

알고리즘 트레이딩은 주가의 움직임을 수학적으로 분석하고, 이를 잘 설명하고 예측할 수 있는 수학적 모델을 만들어 IT 기술로 구현한 것으로, 수치라는 객관적인 기준에 의해 거래한다. 수학적 모델을 만드는 과정에서 제일 중요한 핵심은 검증이다. 설계된 수학적 모델이 과거 데이터를 이용해 검증했을 때 통계적으로 유의미하다고 판단되는 모델만이 살아남아서 알고리즘 트레이딩으로 구현되기 때문에 주먹구구식으로 거래하는 사람보다는 좋은 결과를 가져올 수밖에 없다.

알고리즘 트레이딩은 금융거래를 'Art to Science'로 변화시킨 것으로, 그동안 사람들이 직관 또는 어떤 믿음에 근거해 거래하던 방식에 데이터에 의한 그리고 수학에 기반을 둔 과학적인 방식을 적용했고, 결과적으로 효과가 있음을 증명했다.

사람이 거래하는 전통적인 방식과 수학적 모델에 기반을 둔 알고리즘 트레이딩을 비교했을 때 알고리즘 트레이딩의 장점은 속도와 감정이 없다는 2가지로 요약할 수 있다.

첫 번째로 알고리즘 트레이딩은 사람과 비교할 수 없을 정도로 빠른 속도를 자랑한다. 매일 오전 9시부터 오후 3시까지 주식시장이 열리고, 이 주식시장에서는 다양한 생각을 하는 집단과 개인들이 자신의 선택에 따른 투자를 하고 있으며, 이러한 영향으로 주가는 상승과 하락을 무작위로 반복하며 빠른 속도로 복잡한 그래프를 그린다. 사람은 이러한 주가의 변화에 빠르게 대처하는 데 한계가 있을 수밖에 없다. 짧은 시간에 사야 할지 팔아야 할지 아니면 보유해야 할지를 결정하고 실제 거래까지 하기는 쉽지 않은 일이다.

알고리즘 트레이딩은 정해진 수학적 모델에 따라 빠르게 판단하고 거래할 수 있다. 이러한 알고리즘 트레이딩의 특성을 극대화한 것이 고빈도 매매HFT, High Frequency Trading다. HFT는 하루에도 수십 번에서 수백 번의 많은 거래를 통해 수익을 만들어내는 방식으로, 2010년 이후로 인기를 얻고 있다.

두 번째 장점인 무감정은 어찌 보면 앞서 언급한 속도보다 더 강력한 것으로, 수학

적 모델에 기반을 두고 결정하므로 투자의 큰 장애물인 감정이 개입할 여지가 없다. 사람에게 소중한 자산인 돈을 투자한다는 것은 위험을 수반하는 것이어서 손실에 대한 두려움이 본능적으로 생길 수밖에 없다. 투자금이 커지면 커질수록 손실에 대한 두려움은 폭발적으로 자라나 정상적인 판단을 어렵게 만든다. 대부분 아니 거의 모든 사람은 이러한 손실에 대한 두려움에서 벗어날 수 없으며, 결국 이 두려움이 합리적인 판단을 방해하고 손실로 이끈다.

기사를 보면 '개미'라 부르는 개인투자자들이 주식시장에서 대부분 손실을 보는데, 이는 종목 선정의 실패도 영향이 크겠지만 하루하루 주식의 등락폭에 따른 심리적 고통을 이기지 못하고 감정적으로 판단해 투자가 실패로 끝난다고 한다. 너무나 당연한 이야기지만, 알고리즘 트레이딩은 주어진 수학적 모델의 결과에 따라 팔지 살지 보유할지를 결정하기 때문에 감정이 영향을 전혀 미칠 수 없다.

알고리즘 트레이딩의 부정적인 면도 언급하지 않을 수 없다. 가장 많이 회자하는 것은 시장의 교란이다. 최근 금융시장에 알고리즘 트레이딩이 일반화되어 있는데, 너무 많은 거래를 발생시켜 시장의 변화가 빠르고 혼란스러워진다는 것이다. 이와 더불어 발생할 수 있는 문제는 어떤 특정 상황에, 우연의 일치인지 아니면 비슷한 알고리즘 트레이딩 모델을 사용해서 인지 모두 매도 주문을 내서 시장을 폭락시킬 가능성이다.

예를 들어, 주식거래에서 많은 종류의 알고리즘 트레이딩이 모멘텀Momentum을 이용한 모델을 사용하는데, 수학적 이론이 기본적으로 동일해서 어떤 신호가 주어지면 주식매도라는 유사한 결과를 도출해낼 수 있다. 이렇게 되면 매우 짧은 시간에 많은 양의 주식이 시장에 나오게 되고, 사려는 사람이 없다면 주식시장은 폭락하게 된다.

또 한 가지는 프로그램 오류나 수학적 모델 실패다. 알고리즘 트레이딩에 핵심적 역할을 하는 수학적 모델이 잘못 설계되었거나 알고리즘 트레이딩을 실현하기 위해 개발한 IT 시스템에 에러가 있다면 엄청난 손해를 초래한다. 국내에서는 한맥

투자증권이 2013년 12월 단 한 번의 오류로 2분 만에 460억 원이라는 손실을 봤고, 결과적으로 회사가 문을 닫게 되었다.

알고리즘 트레이딩은 머신러닝에 적합한 환경이다. 금융시장은 주식, 옵션, 선물, 환율 등 활용할 수 있는 많은 데이터가 있어서 머신러닝에 반드시 필요한 데이터를 충분히 공급할 수 있기 때문이다.

그러나 머신러닝에서는 아무리 훌륭한 알고리즘이 있다 하더라도 이를 학습시킬 데이터가 없다면 무용지물이다. 빅데이터의 시대에 접어들면서 데이터의 크기가 늘어나고 있고, 스마트폰과 IoT는 더욱 많은 데이터를 생성하며, 클라우드는 이들 데이터를 처리하는 데 필요한 충분한 컴퓨팅 파워를 제공한다. 머신러닝이 발전하기에는 더없이 좋은 환경이다.

4.2 인물로 살펴보는 알고리즘 트레이딩의 역사

알고리즘 트레이딩의 역사를 살펴보는 것은 매우 재밌을 뿐만 아니라 수많은 천재에 의해 어떻게 발전해왔는지를 알 수 있어서 의미가 있다. 알고리즘 트레이딩은 수학과 IT를 기반으로 발전해 금융공학의 발전과 궤를 같이한다. 알고리즘 트레이딩은 수많은 기관과 사람이 참여해 지금까지 발전시켜 왔지만, 필자가 개인적으로 가장 지대한 영향을 미쳤다고 생각하는 3명의 인물을 중심으로 설명하겠다.

4.2.1 에드워드 소프

에드워드 소프Edward Thorp[03]는 알고리즘 트레이딩의 1세대라고 할 수 있는데, 그가 월스트리트에서 최초로 수학과 IT 시스템을 이용해 펀드를 운영했기 때문이다. MIT 수학과 교수 출신인 그는 'Princeton

03 출처: http://edwardothorp.com/id1.html

-Newport Partners'라는 투자회사를 설립했고, 1970년부터 1998년 회사를 청산하기까지 단 한 번의 손실도 없이 연평균 20%라는 놀라운 수익률을 보여주었다.

주식에 조금이라도 관심 있는 사람이라면 누구나 아는 워런 버핏Warren Buffett의 연평균수익률이 21.6%고, 같은 기간 S&P 500의 연평균수익률이 8.84%라는 것을 생각해보면 소프가 보여준 수익률이 얼마나 대단한 것인지를 알 수 있다.

소프는 매우 특이한 이력을 가진 사람으로, 여러 가지 재밌는 이야깃거리를 가지고 있다. UCLA 물리학과 대학원 시절에 동료들과 쉽게 돈을 버는 방법에 관해 이야기하던 중 영감을 받아 카지노 룰렛에서 공이 어디에 멈출지를 예측할 수 있는 웨어러블 장치(최초의 웨어러블이라고도 한다)를 고안했다.

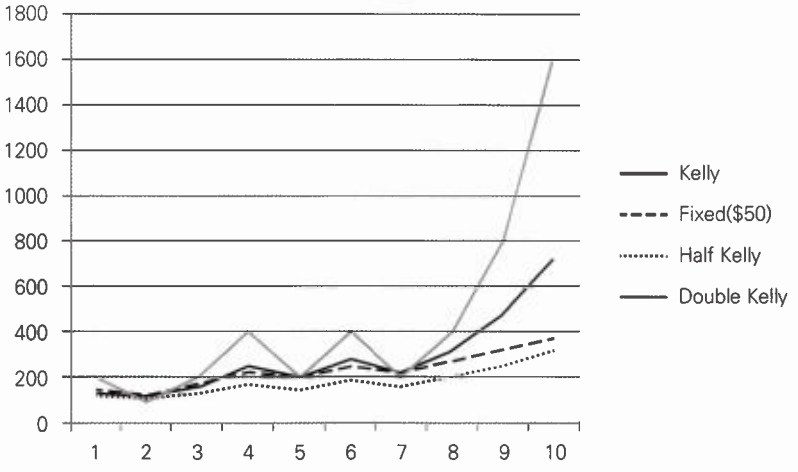
특히 카드 게임인 블랙잭은 소프의 인생에서 중요한 부분을 차지한다고 할 수 있다. 소프는 「부의 공식: 블랙잭 승리 전략Fortune's Formula | A Winning Strategy for Blackjack」이라는 제목의 논문을 발표했는데, 블랙잭 게임을 수학적으로 분석해 카지노 딜러를 이기는 방법을 설명한 것으로, 미국 전역의 도박시들에게 엄청난 인기를 얻었다. 소프가 제시한 방법은 딜러 데크의 상황에 따른 확률을 계산하는 것으로, 흔히 '카드 카운팅'으로 알려졌다. 소프는 이 방법을 소개하는 데 그치지 않고 직접 라스베이거스로 가서 많은 돈을 벌었다고 하니 케빈 스페이시가 출연한 2008년 개봉 영화 「21」의 실사 판이라 할 수 있다.

이후 그는 주식시장으로 전향했고 전 세계 금융사에 확실한 기록을 남겨놓은 LTCMLong-Term Capital Management의 설립자인 존 메리웨더John Meriwether와 Princeton-Newport Partners를 시작하였다. 소프는 켈리 방정식Kelly Criterion의 열렬한 신봉자로 투자금 위험 관리에 적극적으로 활용해 손실 없이 성공적으로 펀드를 운영할 수 있다고 이야기한다.

투자의 세계에서는 단 한 번의 실수로 모든 투자금을 날릴 수 있어서 위험 관리는 투자에서 매우 중요한 문제다. 위험을 없애는 가장 좋은 방법은 투자하지 않는 것이지만, 투자하지 않으면 아무런 이득도 발생하지 않기 때문에 위험을 짊어질 수밖에 없다. 문제는 얼마의 금액을 어디에 투자해야 더 높은 이익을 기대할 수 있으면서 위험을 낮출 수 있는가다.

켈리 방정식은 이런 문제에 답을 제시해 주는 것으로, 처음에는 투자수익률이 높지는 않지만 시간이 지날수록 수익이 급격하게 늘어나는 특성을 보여준다.

그림 4-2 켈리 방정식[04]



소프는 통계적 차익거래Statistical Arbitrage에 기반을 둔 알고리즘 트레이딩 시스템을 주로 활용한 것으로 알려졌다. 통계적 차익거래는 상관관계가 깊은 주식을 선택하고, 이들 간의 차이인 스프레드Spread를 통계적으로 분석해 적절한 헤지Hedge 전략을 통해 주식을 매도, 매수하는 방법으로, 페어트레이딩Pairs Trading이 대표적이다.

04 출처: https://www.biggerpockets.com/renewsblog/2013/12/05/kelly-criterion/

4.2.2 제임스 해리스 사이먼스

소프와 마찬가지로 제임스 해리스 사이먼스James Harris Simons[05]도 하버드 대학 수학 교수였다. 1982년, 그는 자신의 수학적 지식이 금융시장에 어떻게 적용되는지를 시험하기 위해 'Renaissance Technologies'라는 헤지펀드 회사를 세웠다. 제임스 사이먼스는 헤지펀드 역사상 가장 성공한 CEO라는 평가를 받을 정도로 엄청난 성공을 거두었으며, 2014년 전 세계 부호순위 88위로, 약 12조 8,500억 원의 재산을 가지고 있다. 대표적 펀드인 메달리온Medallion은 청산할 때까지 연평균수익률 36%라는 엄청난 기록을 세웠고, 특히 금융위기가 한창이던 2008년에는 80%의 이익을 거둬 많은 사람의 찬탄을 받았다.

제임스 사이먼스를 실제로 만나본 사람들은 한결같이 '수학에 미친 사람'이라고 얘기할 정도로 수학에 대한 애정이 깊고, 이러한 그의 애정은 금융시장에 적용되어 수학적 접근방법으로 펀드를 운용한다. 사이먼은 인터뷰를 통해 "펀드를 운용할 때 가장 먼저 하는 일은 수많은 과거 데이터를 수집해 분석하고, 분석을 통해 반복되는 공식을 찾고, 그 공식에 맞춰 펀드를 운영한다"고 이야기했다.

Renaissance Technologies는 이러한 생각을 바탕으로 운영하는 헤지펀드 회사로, 투자와 별로 연관이 없어 보이는 수학자, 물리학자, 천문학자, 컴퓨터과학자 등 금융을 모르는 사람들을 채용하고, 흔히 금융회사에서 많이 볼 수 있는 전통적인 기업분석이나 주식분석 등은 하지 않는다.

Renaissance Technologies는 헤지펀드 운용에서 금융지식보다는 수학과 IT 기술이 더 중요하기 때문에 관련 지식이 있는 사람을 채용하고, 그들에게 필요한 금융지식을 학습하게 해 펀드를 운용한다. 또한, 자사의 펀드 운용 방식이

---

05 출처: By Gert-Martin Greuel, CC BY-SA 2.0, https://goo.gl/pCf8cs

나 적용 모델 등을 외부에 공개하지 않기로 유명해 직원을 채용할 때 회사에서 얻은 내용은 일정 기간 외부에 공개하지 않는 비밀서약을 맺는다고 한다. 그래서 Renaissance Technologies에서 어떤 모델을 이용해 알고리즘 트레이딩을 하는지는 공개되지는 않았지만, 추세추종Trend Following에 기반을 둔 모델을 적극적으로 사용하는 것으로 알려졌다.

추세추종은 어떠한 긴 흐름의 추세가 시작되면 이를 파악해 상승 흐름에 올라탈 수 있다는 이론에 기반을 둔다. 추세추종은 오랜 역사를 가진 모델로 수학적 이론이 잘 정립되어 있고 이해하기도 쉬워 시장에서 널리 사용되며 세계에서 가장 큰 선물거래기관에서도 적용하고 있다.

그림 4-3 추세추종 예[06]

06 출처: http://goo.gl/AAXKTP

4.2.3 케네스 그리핀

케네스 C. 그리핀Kenneth C. Griffin[07]은 앞의 두 사람처럼 수학과 교수가 아닌 하버드대학교 경제학과 출신으로, 현역으로 활발히 활동하고 있는 헤지펀드 매니저다. 2014년 포브스 선정 미국 400대 부호 가운데 69위에 올랐고, 자산은 70억 달러로 추정된다. 어려서부터 컴퓨터 프로그래밍에 능했던 그리핀은 하버드대학교 재학 시절 소프가 저술한 책인 『Beat the Market: A Scientific Stock Market System』(Random House, 1967)이라는 책을 읽고 전환사채에 대한 아이디어를 얻어 'Convertible Hedge Fund # 1'이라는 합자회사를 시작했다.

그리핀은 거의 모든 금융기관과 투자가들이 엄청난 손실을 봤던 1987년 10월 19일, 블랙먼데이에 하버드대학교 기숙사에서 펀드를 운용하고 수익을 올리면서 월스트리트에서 명성을 얻기 시작했다. 그리고 1990년 대학교를 졸업하며 'Citadel'이라는 헤지펀드를 설립했다. Citadel은 가장 성공적인 헤지펀드라는 평가를 받고 있으며 2015년 기준으로 25조 달러의 자금을 운용하는 세계 최대의 헤지펀드 중 하나다. Citadel의 특성은 IT 기술을 적극적으로 활용한다는 점이다.

앞서 언급한 소프와 사이먼은 수학에 조예가 깊지만, 그리핀은 컴퓨터 프로그래밍에 익숙하기 때문인지는 몰라도 Citadel에는 트레이더보다 IT 인력이 더 많다고 한다. 또한, 모델의 개발, 검증, 실제 거래에 이르기까지의 많은 과정을 자동화하였고, HFT 영역을 개척했다는 평가를 받고 있다.

4.3 알고리즘 트레이딩 모델

알고리즘 트레이딩에서 모델은 주식의 가치를 평가하고, 이에 따라 매수, 매도, 보

07 출처: By Paul Elledge, CC BY-SA 3.0, https://goo.gl/Xo6h0F

유할지를 결정하기 때문에 위험 관리, 거래비용 관리, 포트폴리오 구성, 검증 등 알고리즘 트레이딩 시스템의 여러 영역 중에서도 매우 중요한 위치를 차지한다. 사실상 위험 관리, 포트폴리오 구성 등은 모델을 기반으로 적절한 전략을 세우기 때문에 알고리즘 트레이딩의 뼈대를 만든다고 할 수 있다.

알고리즘 트레이딩에 사용되는 모델은 평균 회귀Mean Reversion, 일간 모멘텀Interday Momentum, 추세추종Trend Following, 인덱스 펀드 재분배Index Fund Rebalancing 등 매우 다양하며 지금도 계속 새로운 모델이 개발되고 있다. 알고리즘 트레이딩에서는 시장 평균수익보다 월등한 수익을 내는 모델을 '알파 모델Alpha Model'이라고 하고, 시장 평균이나 이를 약간 웃도는 정도를 '베타모델Beta Model'이라고 한다. 알고리즘 트레이딩에서 통용하는 모델은 알파 모델을 의미하며, 크게 2가지 접근방법이 있다.

첫 번째는 'Theory-Driven'으로 어떤 모델을 가정하고 그 모델이 맞는지 틀리는지 검증을 통해 알파 모델을 완성해 나가는 방법이다. 이는 과학자들이 많이 쓰는 익숙한 방법이다.

두 번째는 'Data-Driven'으로 데이터를 분석해 패턴을 찾고 이를 알파 모델로 만드는 방법으로, 뚜렷한 가설이나 이론적 배경 없이 데이터를 분석하면서 지식을 쌓아가는 방식이다. 인간 게놈 프로젝트가 Data-Driven 방식으로 진행되었다.

그림 4-4 알파 모델의 접근 방법

Theory-Driven은 Top-down 접근방법이라 할 수 있고, Data-Driven은 Bottom-up 방식이라 할 수 있다. 알고리즘 트레이딩을 하는 많은 사람은 Theory-Driven 접근방법을 선호한다. 가장 큰 이유는 Theory-Driven이 훨씬 더 익숙하다는 것과 모델에 대한 이해도가 높다는 점이다.

Theory-Driven은 어떤 가설을 세우고, 이를 기반으로 알파 모델을 만들기 때문에 모델 설계자의 측면에서 보면 자신의 모델이 무엇인지, 여러 변수가 어떻게 영향을 미치는지를 잘 이해할 수 있다. 하지만 Data-Driven 접근방법은 가설을 세우지 않고 데이터 분석을 시작하는데, 뚜렷한 목적의식이 없다면 데이터를 분석할 때 이것의 의미를 찾기가 쉽지 않고, 더 나아가 어떤 패턴을 발견했더라도 이를 이해하기가 어렵다.

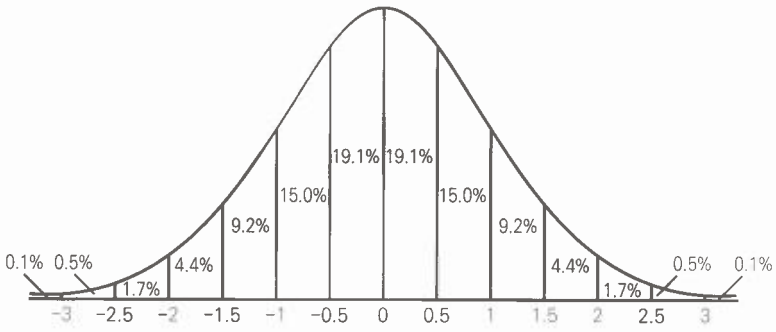
여기서는 Theory-Driven의 대표적 모델인 평균회귀와 Data-Driven에 머신러닝을 적용해 알파 모델을 만들어 본다.

4.4 평균회귀 모델

평균회귀 모델은 알고리즘 트레이딩에서 매우 중요한 모델의 하나로 많은 사람이 활용하는데, 이 모델의 기본 가정은 다음과 같다.

- 시계열 데이터는 과거의 평균값으로 회귀하려는 경향이 있다.
- 어떤 변수가 다음과 같은 정규분포를 따른다고 하면 평균에 가까이 갈 확률이 높아지고, 반대로 멀어질수록 확률이 낮아지는 것과 비슷한 이치라고 할 수 있다.

그림 4-5 정규분포도



예를 들어, 현시점의 주가가 평균주가보다 낮으면 주가가 올라가려 하고, 반대로 현주가가 평균주가보다 높으면 주가가 내려가려 하는 경향을 이용한 것이 평균회

귀 모델이다. 평균회귀 모델은 수학적으로도, 관념적으로도 알고 있어 이해하기 쉽고 우리 주변에서 어렵지 않게 찾아볼 수 있다.

만약 주가 데이터에 평균회귀 모델을 적용할 수 있다면, 언제 주식을 사야 하는지, 언제 팔아야 하는지를 쉽게 알아내 고수익을 내는 것은 어렵지 않다. 하지만 애석하게도 앞 장에서 이야기한 대로 주가 데이터는 무작위, 즉 랜덤워크라서 무턱대고 평균회귀 모델을 적용할 수는 없다.

4.4.1 평균회귀 테스트

평균회귀 모델을 적용하려면, 먼저 적용하려는 주가 데이터가 랜덤워크인지 아닌지를 파악해야 한다. 랜덤워크는 다음 행보가 이전 행보에 영향을 받지 않는 독립적인 사건이라는 것을 의미한다. 주가 데이터가 랜덤워크를 보인다는 것은 주가의 흐름이 이전의 데이터와 상관없이 결정된다는 의미인데, 이는 평균회귀 모델을 적용할 수 없다. 평균으로 회귀하려면 이전 데이터가 현재 데이터에 영향을 미쳐 평균으로 회귀하도록 해야 하기 때문이다. 즉, 비독립적이어야 한다. 평균회귀 모델을 적용할 수 있는 시계열인지 아닌지를 판별하기 위해 많이 사용하는 방법으로는 ADF Augmented Dickey-Fuller 테스트와 허스트 지수Hurst Exponent가 있다.

ADF 테스트

어떤 주가 데이터가 평균회귀 성질을 가지고 있다면 현주가는 다음의 주가가 어떻게 될지 예측할 수 있는 통계학적 정보를 포함하고 있다. 예를 들어, 현주가가 평균보다 낮다면 다음 주가의 값은 알 수 없지만, 주가가 위로 향하는 방향성을 가질 확률이 높다는 것을 알 수 있다. ADF 테스트는 어떤 시계열 데이터가 랜덤워크를 따른다는 가설을 세우고, 이 가설을 검증해 랜덤워크인지 아닌지를 판단한다는 방법이다.

다음과 같은 시계열 데이터가 있다고 가정하자. α는 상수고, β는 트랜드 계수Trend coefficient이며, $\Delta = y(t) - y(t-1)$ 관계를 만족한다.

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \cdots + \delta_{p-1} \Delta y_{t-p+1} + \varepsilon_t,$$

이 시계열 데이터가 랜덤워크라고 가정하면 y(t)와 y(t - 1)의 상관계수는 0이 되어야 한다. 랜덤워크는 정의에 따라 현재값이 이전값에 영향을 받지 않는 독립적인 특성이 있어서 y(t)와 y(t - 1)은 어떠한 상관관계도 없어야 하는데, 이렇게 상관관계가 없을 때 상관계수의 값은 0이 되기 때문이다.

ADF 테스트는 설명한 내용을 그대로 적용한 것으로, 앞의 식에서 $\gamma = 0$, $\alpha = 0$, $\beta = 0$을 대입해 가설검정Hypothesis Test을 실시하는 방법이다. 만약 $\gamma = 0$이라는 가설검정에 실패하면 시계열 데이터는 랜덤워크가 아니라는 것을 증명하게 되고, 랜덤워크가 아니라면 평균회귀 시계열이라는 결론을 얻게 된다.

pandas와 Statsmodels 라이브러리를 이용하면 ADF 테스트를 쉽게 만들 수 있다. 삼성전자 주가를 대상으로 ADF 테스트를 시행해 평균회귀 모델을 적용할 수 있는지를 알아보자.

```
import statsmodels.tsa.stattools as ts
df_samsung = load_stock_data('samsung.data')
adf_result = ts.adfuller(df_samsung["Close"])
pprint.pprint(adf_result)
```

실행결과

```
(-1.1376691628777393,
 0.69981858123217411,
 3,
 234,
 {'1%': -3.4586084859607156,
  '10%': -2.5733956592884799,
  '5%': -2.8739721592357208},
 5060.7362796028701)
```

실행결과의 첫 번째 값은 검정 통계량Test Statistic이고, 두 번째는 p-value, 네 번째는 데이터 수, 다섯 번째 딕셔너리는 가설검정을 위한 1%, 5%, 10% 기각값Critical Value을 나타낸다. $\gamma = 0$이라는 가설검정을 기각하려면 검정 통계량 값이 1%, 5%, 10% 기각값 중 어느 하나보다도 작아야 한다.

삼성전자 주가의 검정 통계량은 -1.137로 기각값 10%인 -2.57보다 크기 때문에 가설을 기각할 수 없다. 결론적으로 삼성전자 주가는 평균회귀 모델을 적용할 수 없다는 것을 알 수 있다. 참고로 `adfuller()` 함수에 대한 자세한 설명은 http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.stattools.adfuller.html에서 확인할 수 있다.

허스트 지수

정상과정은 평균과 표준편차가 일정해서 표류하는 랜덤워크인 기하적 브라운 운동GBM보다 천천히 값이 퍼져나가게Diffusion, 확산 된다. 주가 데이터는 시계열 데이터로, 시간에 따른 주식값의 변화를 확산이라고 생각할 수 있고, 이때 얼마나 빠르게 퍼져나가는지를 수학적 계산을 통해 측정할 수 있다. 허스트 지수Hurst Exponent의 핵심 아이디어는 분산Variance을 확산 속도로 치환할 수 있고, 그 값을 GBM의 확산 속도와 비교하면 랜덤워크인지 아니면 정상과정인지를 파악할 수 있다는 것이다.

주가의 확산속도는 다음과 같이 분산을 이용해 계산할 수 있다.

$$\mathbf{Var}(\tau) = [\,|\log(t+\tau) - \log(t)|^2\,]$$

GBM은 표류하는 랜덤워크 모델이어서 데이터가 충분히 크다면, τ의 분산은 τ에 비례해야 하므로 다음 식을 만족해야 한다.

$$[\,|\log(t+\tau) - \log(t)|^2\,] \sim \tau$$

이 식에서 사용한 '~'는 충분히 큰 데이터라면 어떤 값으로 수렴할 것이라는 의미다. 어떤 정상과정이 이 식을 만족하지 않는다면, 해당 정상과정은 평균회귀 성향이나 추세 성향Trending이 있다는 것을 알 수 있다.

허스트 지수는 τ를 확장해 다음과 같이 정의한다.

$$[|\log(t+\tau) - \log(t)|^2] \sim \tau^{2H}$$

이 식에서 τ의 지수인 2H에서 H를 허스트 지수라고 하는데, H의 크기에 따라 해당 시계열 데이터의 확산속도가 어떤 형태를 띠게 될지 짐작할 수 있다. 예를 들어, 어떤 시계열 데이터가 GBM이라고 가정하면 τ의 지수가 1이어야 하므로 2H = 1 -> H = 0.5의 값을 가져야 하고, GBM이 아니라면 H는 0 또는 0에 가까운 값을 가져야 한다. 이와 같이 계산한 허스트 지수의 값이 H < 0.5면 평균회귀, H > 0.5면 추세 성향이 있다는 것을 알 수 있다.

더 자세한 설명을 위해 허스트 지수를 정의한 식에서 필요한 부분만 발췌한 다음 함수를 보자.

$$f(x) = \tau^{2H}$$

이 함수에서 H에 각각 0, 0.5, 1의 값을 넣어보면 다음과 같다.

- **H = 0.5, $f(x) = \tau^{2\times0.5} = 1$**
  H = 0.5인 경우는 앞의 식과 동일한 형태가 되므로 GBM이라는 것을 알 수 있다.
- **H = 0, $f(x) = \tau^{2\times0} = 1$**
  H = 0이면 값이 1이 되므로 값이 평균회귀한다.
- **H = 1, $f(x) = \tau^2$**
  H = 1이면 지수함수이므로 발산하는 형태의 추세 성향이 있음을 알 수 있다.

허스트 지수는 정상과정이 평균회귀나 추세 성향이 있는지를 알 수 있을 뿐만 아니라 경향의 정도도 알 수 있어 매우 유용한 지수이다. 예를 들어, H값이 0에 가까울수록 평균회귀 성향이 강하고, 반대로 1에 가까울수록 추세 성향이 강하다고 판단할 수 있다.

이번에는 파이썬을 이용해 삼성전자 주가와 한미약품 주가의 허스트 지수를 구해보자.

```
def get_hurst_exponent(df):
    lags = range(2, 100)
    ts = np.log(df)

    tau = [np.sqrt(np.std(np.subtract(ts[lag:], ts[:-lag]))) for lag in lags]
    poly = np.polyfit(np.log(lags), np.log(tau), 1)

    result = poly[0]*2.0

    return result

df_samsung = load_stock_data('samsung.data')
df_hanmi = load_stock_data('hanmi.data')

hurst_samsung = get_hurst_exponent(df_samsung['Close'])
hurst_hanmi = get_hurst_exponent(df_hanmi['Close'])
print "Hurst Exponent : Samsung=%s, Hanmi=%s" % (hurst_samsung,hurst_hanmi)
```

실행결과

```
Hurst Exponent : Samsung=0.4089724019, Hanmi=0.578058846572
```

삼성전자 종가의 허스트 지수는 0.40, 한미약품은 0.57로 GBM에 가깝다는 것, 즉 뚜렷한 평균회귀 성향이나 추세 성향은 보이지 않는 것을 알 수 있다.

평균회귀의 Half-life

ADF 테스트와 허스트 지수에서 살펴봤듯이 삼성전자 주가와 한미약품 주가가 모두 평균회귀 모델을 적용하기에 부적합하다는 것을 알 수 있다. 주가 데이터 대부분이 평균회귀 모델 적용 가능 테스트에서 실패한다. 하지만 그렇다고 실망하기에는 이르다. 실제 알고리즘 트레이딩, 특히 2가지 테스트를 모두 통과하지 못한 주식종목에 평균회귀 모델을 적용해 고수익을 내는 알파 모델이 된 예도 흔치 않게 볼 수 있다.

이번 절에는 앞서 얘기한 2가지 테스트 통과에 실패했더라도, 평균회귀 모델을 적용할 수 있는지를 알 수 있는 Half-life 값을 알아보겠다.

Half-life는 값이 평균으로 회귀하는 데 걸리는 시간을 의미하는 것으로, 이 값을 이용해 평균회귀 모델을 적용할 수 있는 주가를 찾을 수 있다. 평균회귀 성향이 있는 랜덤과정을 오른스타인-우렌벡 과정Ornstein-Uhlenbeck Process이라고 하며 다음과 같은 확률미분Stochastic differential 방정식을 만족한다. λ는 평균회귀 속도고, μ는 평균, dW_t는 에러텀이다.

$$dx_t = \lambda(\mu - x_t)dt + \sigma dW_t$$

Half-life를 쉽게 구하기 위해 다음과 같은 오른스타인-우렌벡 과정을 가정해보자. y_0는 초기치를 나타내고, λ는 평균회귀 속도다.

$$f(t) = y_0 e^{-\lambda t}$$

이 프로세스는 평균회귀하기 때문에 시간 t의 1/2인 시간 $t_{1/2}$ 에 대해 다음과 같은 관계가 성립해야 한다.

$$f\left(t_{1/2}\right) = \frac{f(t)}{2}$$

앞의 2개의 식을 풀고 평균회귀를 위해 부호를 조정하면 다음과 결과를 얻을 수 있다.

$$Half - life, t_{1/2} = -\frac{ln2}{\lambda}$$

이 식을 통해 알 수 있듯이 Half-life는 평균회귀 속도인 λ와 반비례관계에 있음을 알 수 있다. λ가 커지면 속도가 빨라서 그만큼 빨리 평균으로 회귀하고, 반대로 작아지면 속도가 느려서 평균으로의 회귀가 늦다는 것을 의미한다.

half-life에서 의미하는 수치는 계산에 사용한 시간 단위기 때문에 계산에 사용한 데이터가 초 단위의 데이터라면 초 단위의 평균회귀 시간을, 시간 단위라면 시간 단위의 평균회귀 시간을 나타낸다는 것을 유의해야 한다. Half-life의 값이 크

고 작음은 알고리즘 트레이딩에 적용하는 전략에 따라 좋을 수도 나쁠 수도 있다. 예를 들어 Half-life의 값이 크다는 것은 장기간 지속하는 경향이 있다는 것을 의미할 수 도 있고, 반대로 작다는 것은 그만큼 주식값의 변동이 잦다는 것으로 해석할 수도 있기 때문에 전략에 맞춰 판단해야 한다.

파이썬을 이용해 삼성전자 주가와 한미약품 주가의 Half-life 값을 계산해보자.

```
def get_half_life(df):
    price = pd.Series(df)
    lagged_price = price.shift(1).fillna(method="bfill")
    delta = price - lagged_price
    beta = np.polyfit(lagged_price, delta, 1)[0]
    half_life = (-1*np.log(2)/beta)

    return half_life

df_samsung = load_stock_data('samsung.data')
df_hanmi = load_stock_data('hanmi.data')
half_life_samsung = get_half_life(df_samsung['Close'])
half_life_hanmi = get_half_life(df_hanmi['Close'])
print "Half_life : Samsung=%s, Hanmi=%s" % (half_life_samsung,half_life_hanmi)
```

실행결과

```
Half_life : Samsung=35.6008036921, Hanmi=1041.5719091
```

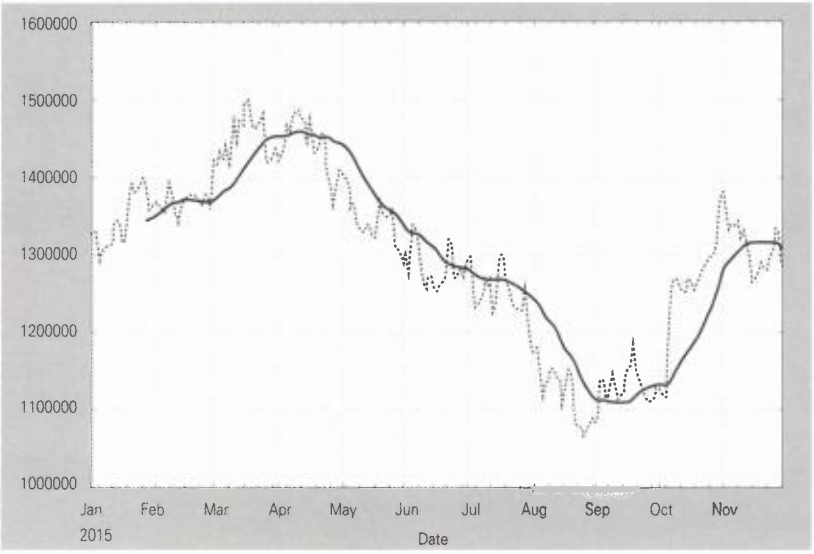
삼성전자 주가의 half-life 값은 35.60이고 한미약품 주가의 Half-life 값은 1041.57로, 삼성전자 주가가 한미약품 주가보다 더 빠르게 평균으로 회귀함을 알 수 있고, 한미약품은 값이 너무 커서 평균회귀 성향이 삼성전자보다 희박하다고 할 수 있다.

4.4.2 평균회귀 모델 구현

ADF 테스트, 허스트 지수, Half-life를 모두 통과한 종목을 찾았다면 이제 평균회귀 모델을 구현할 준비가 되었다. 평균회귀 모델은 실제 알고리즘 트레이딩에 적용할 수 있는 것으로, 무엇보다도 모델의 개념과 단순명료함이 매력적이다. 평균회귀 모델의 기본 개념은 주가가 평균보다 낮으면 주식을 매입하고, 반대로 평균보다 높으면 주식을 매도해 수익을 만드는 것이다.

다음 그림은 삼성전자 주가를 나타낸 것으로, 실선은 이동평균, 점선은 종가다. 예를 들어, 2015년 8월에는 실선으로 표시된 이동평균보다 종가가 낮아졌으니 주식을 매수하고, 10월 이후로 주가가 이동평균을 돌파했으므로 가지고 있던 주식을 매도하는 방법이다.

그림 4-6 이동평균 도표



평균회귀 모델을 완성하려면 다음의 3가지 사항을 결정해야 한다.

- **평균 정의** 주가 매도, 매수의 비교치로 사용될 평균을 어떻게 구할 것인가에 관한 내용이다. 평균을 과거 특정 기간의 주가로 구할 수도 있고, 아니면 이동평균을 이용해 계산할

수도 있다. 이동평균을 이용한다면 이동평균의 기간을 10일 단위로 할지, 30일 단위로 할지를 결정해야 한다. 평균은 매도·매수를 알려주는 일종의 신호(Indicator)로 활용된다.

- **매도·매수 기준** 예를 들어, 주가가 평균보다 높아지는 순간에 매도할지 평균과 주가의 차이가 2배 이상 될 때 매도할지를 결정하는 기준을 말한다. 평균과 마찬가지로 매도·매수 기준은 모델의 수익률에 막대한 영향을 미치는 중요한 요소다. 기준이 너무 낮다면 큰 이익을 보지 못하고 작은 이익에 만족해야 하며, 반대로 너무 높다면 매도나 매수 시기를 놓칠 수 있으니 쉽지 않은 결정이다.
- **데이터 선택** 알고리즘 트레이딩에 사용할 데이터로 무엇을 쓸 것인지 선택하는 것으로, 종가를 사용할지 시가를 사용할지 또는 현재가를 이용할지를 선택한다. 데이터 선택 역시 매우 중요한 문제로 간단히 선택하기에는 무게가 가볍지 않다.

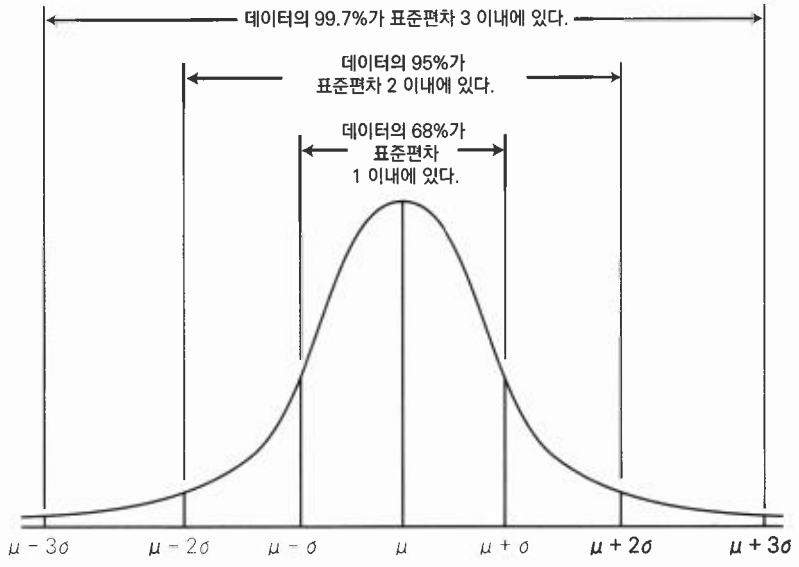
여기서는 다음과 같은 기준으로 평균회귀 모델을 구현한다. 매도·매수 기준에서 기준치를 표준편차로 선택한 것은 확률적으로 안전하다고 생각해서다.

- **평균 정의** 10일 이동평균
- **매도·매수 기준** 이동평균과 주가의 차이가 표준편차보다 크면 매도·매수 실행

  |주가 - 이동평균| > 표준편차일 때
  주가 - 이동평균 > 0, 매도
  주가 - 이동평균 < 0, 매수
- **데이터 선택** 종가 사용

[그림 4-7]은 정규분포를 나타내는 그래프인데, 어떤 값이 표준편차 이내에 있을 확률은 68%로 비교적 빈번한 트레이딩을 할 수 있다. 매수·매도 기준값을 표준편차보다 3배 이상 큰 경우로 설정하면 이러한 일이 발생할 확률이 '100 - 99.7 = 0.3%'로 매우 희귀한 일이 되어 트레이딩 횟수가 매우 적을 수 있다.

그림 4-7 정규분포



앞에서 설명한 평균회귀 모델의 유사코드Psuedo Code는 다음과 같다. 실제 파이썬을 이용한 코드는 '5장 알고리즘 트레이딩 시스템 구현'에서 자세히 소개하므로 여기서는 유사코드로 대체한다. 이 코드에서 rolling_mean( )은 이동평균을 구하는 함수고, rolling_std는 표준편차, price는 시가를 나타내는 변수다.

```
price_moving_average = rolloing_mean(종가,10)
price_standard_deviation = rolloing_std(종가,10)
diff = price- price_moving_average
if |diff| > price_standard_deviation:
    if diff>0:
        sell_stock()
    else:
        buy_stock()
```

4.5 머신러닝 모델

평균회귀 모델이 '정상성Stationarity'이라는 시계열 특성을 바탕으로 주가가 평균회귀할 것이라는 가설을 세우고 모델을 완성해나가는 것과는 대조적으로 머신러닝 모델은 특별한 이론 없이 데이터로부터 시작한다.

데이터로부터 모델을 만들기 시작한다는 것은 어떻게 보면 매우 어려운 과정의 시작이라고 할 수 있다. 가설을 세우고 모델을 만드는 것은 이미 어떤 뚜렷한 생각이나 믿음이 있는 것이어서 나머지 과정은 내 믿음이 맞는지 틀린 지를 확인하는 과정이다. 만약 잘못되었다는 것을 알게 되면 기존 가설을 수정하거나 새로운 가설을 만들면 된다.

하지만 머신러닝은 데이터를 통해 무엇인가를 찾아야 하는데, 때에 따라서는 목표가 무엇인지도 모르는 상태에서 진행해야 하고 이런 경우 어떻게 시작해야 하는지도 모호할 수 있다.

머신러닝에 대해 흔히 하는 착각 하나는 데이터만 입력하면 머신러닝이 알아서 원하는 결과를 만들어줄 것이라는 믿음이다. 앞서 얘기했지만, 머신러닝에 가장 중요한 것은 딥러닝, SVM, 랜덤 포레스트와 같은 알고리즘이나 머신러닝을 문제없이 실행할 수 있는 강력한 컴퓨팅 파워가 아니라 데이터다. 어떤 데이터를 머신러닝에 제공하느냐에 따라 머신러닝이 여러분의 얼굴에 웃음을 만들어 줄 수도, 한숨을 만들어낼 수도 있다.

머신러닝에 사용할 데이터를 준비하는 것은 사용자가 직접 해야 하는 것으로 주관적인 생각이 적극적으로 개입되는 과정이다. 내가 어떤 생각을 가지느냐에 따라 입력변수를 선택하고 이상치를 배제하고 빠진 데이터를 채워 넣고 적절한 형태로 가공하기 때문이다.

알고리즘 트레이딩에서 다행스러운 것은 머신러닝 목적이 매우 분명하고, 비교적 양질의 데이터를 얻기 쉽다는 점이다. 두말할 나위 없이 머신러닝을 활용하는 목

적은 머신러닝을 이용해 예측력이 높은 알파 모델을 만들고, 이를 알고리즘 트레이딩에 적용해 수익을 창출하는 것이다. 이를 위해 머신러닝의 일차적 목표는 수익의 창출기회를 알려주는 신뢰성 높은 알파 모델을 만드는 것이라 할 수 있다.

알파 모델을 만드는 데 입력변수의 선정만큼 중요한 일은 없을 것이다. 머신러닝 알고리즘은 입력 값 간의 관계를 나름의 방법을 통해 유추하고, 모델을 완성하기 때문에 출력변수에 영향을 미치는 입력변수가 많으면 많을수록(반드시 많다고 좋은 것은 아니다.) 예측력이 높아진다.

이러한 이유로 머신러닝 모델을 만들 때 원시 데이터 수집 후 가장 먼저 하는 일이 입력변수들을 살펴보는 일이다. 적절한 입력변수를 선택하면 이후 데이터를 적절히 가공한 후에 머신러닝 알고리즘으로 학습시켜 결과의 성능을 파악한다. 알고리즘 트레이딩에 사용할 머신러닝 모델을 개발하는 것도 이와 크게 다르지 않다.

4.5.1 특징 선택

특징 선택Feature Selection은 머신러닝에 사용할 변수(Feature)를 선택하는 것으로, 변수 선택Variable Selection 또는 속성 선택Attribute Selection이라고도 한다. 특징 선택은 출력변수와 연관성이 높은 입력변수를 선택해 예측력을 높이려는 목적도 있지만, 특징 선택을 진행해야 하는 사람이 이해하기 쉬운 입력변수를 선택하는 것도 또 다른 큰 목적이다.

머신러닝을 이용하더라도 가장 중요한 데이터는 사람이 만들어야 한다. 좋은 결과를 기대하려면 출력변수와 연관 있는 입력변수를 선별해야 하는데, 선별 과정에서 사람의 판단이 들어갈 수밖에 없다. 예를 들어, 주가 예측에 뉴스 데이터를 입력변수로 사용해 예측력을 높인다고 하자. 필요한 뉴스 데이터를 수집했다면 이제 특징 선택을 해야 한다. 모든 뉴스 데이터와 출력변수와의 상관관계를 분석했더니 경제뉴스와 연예뉴스가 예측력이 있는 것으로 결과가 나왔다면, 과연 어떤 결정을 내리는 것이 좋은 특징 선택이 될지 고심해야 한다.

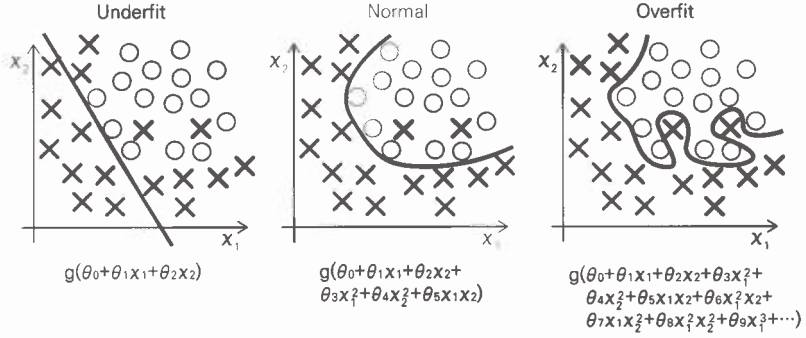
상식적으로 생각하면 경제뉴스가 주가 예측과 연관이 있는 것은 당연하므로 특징으로 선택해 사용할 수 있지만, 연예뉴스의 경우는 이야기가 다르다. 이 결과가 정말 연예뉴스가 주가예측과 연관이 있다는 것을 의미하는 것인지 아니면 데이터상에 오류가 있는 것인지, 상관관계 분석의 문제가 있는 것인지, 그것도 아니라면 우연에 의한 것인지 판단해야 한다.

이러한 결정은 사람이 내려야 해서 입력변수에 대한 이해도가 중요하다. 학습에 사용할 데이터가 많다면 더 좋은 결과를 얻을 수 있지 않을까 하는 생각을 할 수도 있는데, 생각과는 반대의 결과를 매우 빈번하게 보여준다.

머신러닝 알고리즘은 거칠게 이야기하면 주어진 데이터에 어떻게든 뜯어 맞춰 모델을 완성하는데, 머신러닝의 이러한 특성은 흔히 과적합Overfitting 문제를 일으킨다.

다음 그림을 보면 'x'와 'o'로 표시된 데이터를 구분하는 문제, 즉 분류classification다. 'Underfit'은 너무 단순화되어서 데이터를 구분하는 데 에러가 많고, 'Overfit'은 데이터에 너무 최적화되어 있어 예측력이 떨어지고, 이 둘 사이의 적절한 균형을 가진 분류는 가운데 'Normal'이다.

그림 4-8 과적합[08]



08 출처: http://0agr.ru/wiki/index.php/Overfitting

과적합은 주어진 데이터(seen data)에는 높은 예측력을 보여주지만, 학습에 사용되지 않은 데이터(흔히 unseen data라고 한다)에는 그다지 좋은 예측력을 보여주지 못한다. 따라서 적절한 출력변수와 상관관계가 있는 입력변수를 적절히 선택해주는 특징 선택이 중요하다.

알고리즘 트레이딩에서 사용할 수 있는 입력변수의 예를 몇 가지 들어보면 다음과 같다.

- 주가 데이터 삼성전자 주가, 한미약품 주가 등 주가 데이터 자체를 의미
- 거래량 데이터 주식의 거래량 데이터
- 지수 데이터 KOSPI 지수, KOSDAQ 지수 등
- 외부 데이터
  - 환율, 금리 등의 데이터
  - 뉴스 데이터 등 감성 분석Sentiment analysis에 필요한 데이터
- 기업 데이터 매출액, 영업이익률, PBR, PER, EPS 등의 데이터

이 외에도 다양한 입력변수가 있을 수 있으니 원하는 목적과 적용하려는 모델에 맞는 특징 선택을 할 수 있도록 많은 노력을 기울여야 한다.

4.5.2 가격이냐 방향이냐

머신러닝이 해결할 수 있는 문제는 원론적으로 회귀, 분류, 군집화라고 앞 장에서 이야기했다. 알고리즘 트레이딩에 적용할 머신러닝 모델을 만들기 위해서는 원하는 결과가 무엇인지를 명확히 하고, 결과에 맞는 문제로 정의해야 한다. 예를 들어, 머신러닝을 이용해 주가 자체를 예측하고 싶다면 회귀 문제로 정의해야 하고, 주가의 방향이 올라갈지 내려갈지를 알고 싶다면 분류 문제로 정의하는 것이 맞다.

정의한 문제의 성격에 따라 적용할 수 있는 머신러닝 알고리즘이 달라지고 데이터를 만드는 것 역시 영향을 받는다. 주가 자체를 예측하는 것은 주가의 흐름을 잘 나타낼 수 있는 어떤 패턴을 머신러닝으로 찾고 그 모델을 이용해 미래의 주가를

맞추는 것으로, 지금 1만 원인 주가가 다음에는 얼마가 될지를 예측하는 전형적인 회귀 문제다.

주가의 방향을 예측하는 것은 분류 문제로 정의하는 것이 더 예측력이 있을 수 있다. 주가는 랜덤워크라서 미래의 주가가 현재의 주가에 영향을 받지 않는 독립적인 관계에 있으므로 주가가 상승하거나 하락하는 확률은 각각 50%로 생각할 수 있다. 마치 동전 던지기를 하는 것처럼 상승 아니면 하락만 맞추면 된다. 이 책에서는 주가방향을 예측하는 분류 문제로 정의해 머신러닝 모델을 만들어 보겠다.

4.6 분류 모델

머신러닝 모델을 개발하기 위해 주가방향 예측을 분류 문제로 정의하고 이에 적절한 알고리즘을 선택한다. 분류는 'Yes or No'로 대답할 수 있는 문제들, 예를 들어 '내일 비가 올까 오지 않을까', '한국과 일본의 야구경기에서 한국이 이길까', '내일 삼성전자 주가가 오를까'와 같은 것들에 적용할 수 있다. 우리가 알고 싶은 문제의 상당수는 스팸메일 필터링, 텍스트 분류, 이미지 인식과 같은 것들이 많아서 분류는 머신러닝에서 매우 중요한 위치를 차지하고 있으며 많은 알고리즘과 활용 방법이 있다.

분류는 회귀와 마찬가지로 입력변수 간의 상관관계를 분석해 예측하지만, 결과값이 연속이 아닌 이산 값을 가진다는 것이 다르고, 이에 따라 회귀와는 다른 방식으로 예측한다. 여기서 분류에 사용되는 대표적인 알고리즘들을 소개하겠다.

4.6.1 로지스틱 회귀

로지스틱 회귀Logistic Regression는 이산변수 간의 상관관계를 나타내는 모델로, 0 또는 1의 이분형결과Binary Outcome(이분형결과 개념을 확장해 여러 개의 결과물을 가질 수도 있다)을 가지며, 로짓 회귀Logit Regression라고도 한다. 알고리즘에 회귀라는 명칭이 붙어서 회

귀의 한 종류라고 생각할 수도 있지만, 엄연한 분류 알고리즘으로 선형 회귀Linear Regression의 아이디어를 확장해 활용하기 때문에 이러한 명칭이 붙게 되었다.

로지스틱 회귀는 조건부 확률Conditional Probability로부터 출발한다. 입력변수 값에 따른 출력변수의 조건부 분포Conditional Distribution를 만들면, 출력변수의 값은 정확도에 대한 확률로 생각할 수 있다. 예를 들어, 주가방향 예측에 대한 로지스틱 회귀의 결과값이 0.65가 나왔다면 주가방향이 '65%의 확률로 올라갈 것'을 의미한다.

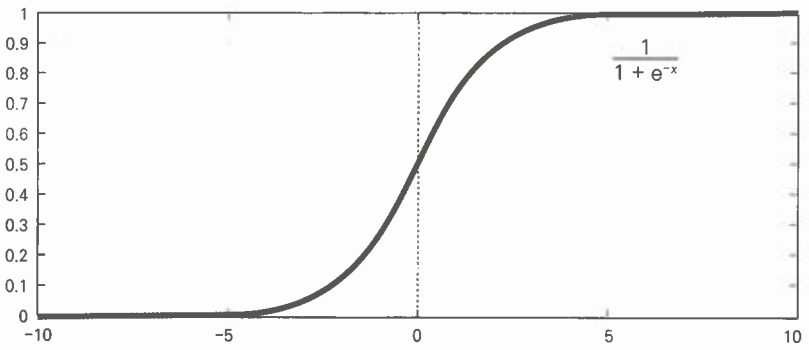
로지스틱 회귀를 이용하려면 확률함수 p(x)를 찾아야 하는데, p(x)는 선형함수이며 p(x)의 값은 0부터 1 사이라는 조건을 만족해야 한다. 로지스틱 회귀의 입력변수는 연속값과 이산값 모두 사용할 수 있지만, 출력변수의 값은 0과 1 사이에만 존재해야 한다. 하지만 선형회귀를 적용하면 그 값이 0과 1이 아닌 다양한 값이 나온다는 문제가 발생한다.

로지스틱 회귀는 이러한 문제를 해결하기 위해 로지스틱 함수Logistic Function를 적용한다.

$$f(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

로지스틱 함수의 그래프는 다음과 같다.

그림 4-9 로지스틱 함수



그래프에서 알 수 있듯이 x 값은 음수, 양수 등을 모두 사용할 수 있지만 출력값인 y는 0부터 1사이다. 로지스틱 회귀에 필요한 함수는 확률함수 p(x)이므로 로지스틱 함수를 출력변수 함수로 사용하면 값이 0부터 1사이에 위치하므로 앞서 이야기한 문제를 해결할 수 있다. 따라서 로지스틱 회귀 모델을 만드는 데 필요한 것은 출력함수인 로지스틱 함수의 기울기를 결정하는 2개의 파라미터 β_0와 β_1의 값을 찾는 것이다.

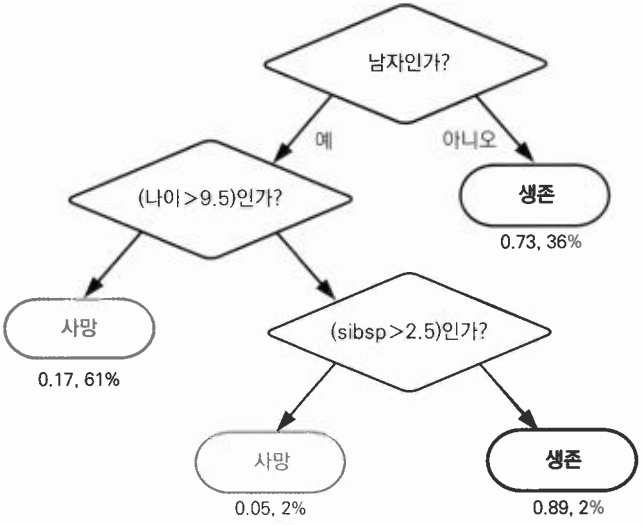
이와 같이 파라미터 값을 찾는 것을 모델피팅Model Fitting이라고 한다. 가장 많이 사용되는 방법은 MLEMaximum Likelihood Estimation로, 확률을 최대로 만들어주는 파라미터를 찾아준다. 파이썬 머신러닝 라이브러리인 Scikit-learn을 이용하면 모델피팅과 관련된 작업을 매우 쉽게 해결할 수 있다.

4.6.2 의사결정 트리와 랜덤 포레스트

의사결정 트리Decision Tree는 머신러닝 알고리즘의 한 종류로 입력변수의 특성을 트리 구조에 매핑해 분류하는데, 그 모양이 나무와 비슷해 '분류와 회귀 트리CART. Classification and Regression Tree'라고도 부른다. 회귀와 분류에 모두 적용할 수 있으며, 우수한 성능을 보여주어서 매우 인기있는 머신러닝 알고리즘이다. 의사결정 트리가 무엇인지는 [그림 4-10]을 보면 직관적으로 이해할 수 있다.

입력변수의 값에 따른 확률을 계산한 것이 의사결정 트리인데, 이를 도식화한 것이 [그림 4-10]이다. 입력변수의 특성, 예를 들어 탑승객의 나이가 9.5세 이상이라면 사망에 이를 확률이 61%이고, 남성이 아니라면 생존확률이 36%라는 식으로, 특성에 따른 확률값을 모두 계산한다.

그림 4-10 타이나닉호 탑승객의 생존 여부를 나타내는 의사결정 트리[09]



*sibsp: 탑승한 배우자와 자녀의 수

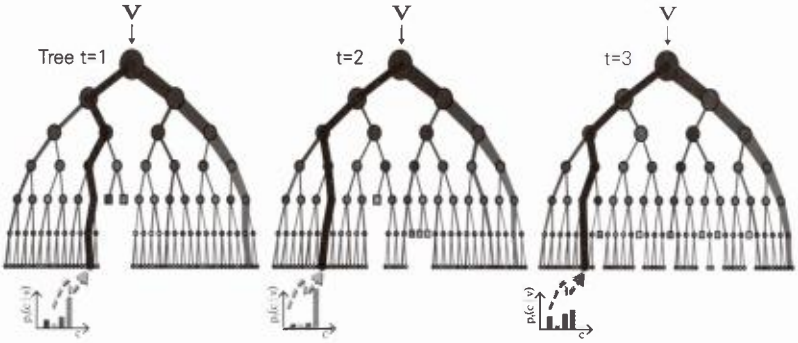
어떤 문제에 대한 확률을 계산하고 싶다면 그 값을 의사결정 트리에 넣어 사다리를 타듯이 내려오면 최종 확률을 알 수 있다. 의사결정 트리의 장점 중 하나는 이해하기 쉽다는 점이다. SVM이나 신경망Neural Network과 같은 머신러닝 알고리즘은 그 난해함 때문에 이해하기 어려운 블랙박스로 취급되는 데 비해 의사결정 트리는 [그림 4-10]에서 보듯이 구조가 이해하기 쉬운 형태로 되어 있다. 또한, 입력변수들의 중요도를 파악할 수도 있어 특징 선택에 활용될 수 있고, 대규모 데이터도 잘 처리한다.

랜덤 포레스트Randomr Forest는 여러 개의 의사결정 트리를 사용하는 앙상블Ensemble 학습기법이다. 머신러닝에서 앙상블 학습기법은 좀 더 좋은 결과를 얻기 위해 다수의 알고리즘이나 다수의 모델을 사용해 예측력을 높이는 방법이다.

다음 그림을 보자.

09 출처: By Ehwaz_k, CC BY-SA 3.0, https://goo.gl/lisqo2

그림 4-11 랜덤 포레스트[10]



랜덤 포레스트 출력 확률 $p(c \mid v) = \frac{1}{T}\sum_{t}^{T} p_t(c \mid v)$

[그림 4-11]에는 3개의 서로 다른 의사결정 트리가 있다. 랜덤 포레스트는 데이터가 들어오면 3개의 의사결정 트리를 이용해 예측하고 그 예측값을 통합해 최종적인 예측값을 내놓는 방식이다. 비유해서 설명하면 3명의 주식전문가에게 내일 주식이 상승할 것 같냐고 물어보았을 때 2명이 상승한다고 대답하고 1명은 하락한다고 대답한다면, 2/3가 상승이라고 대답했으므로 상승확률이 66.6%라고 계산하는 방식이다.

랜덤 포레스트는 상당히 우수한 머신러닝 알고리즘의 하나로, 의사결정 트리와 마찬가지로 회귀와 분류 문제에 모두 적용할 수 있으며 비교적 학습시간이 짧아서 다방면에서 많이 활용하고 있다.

4.6.3 SVM

SVM[Support Vector Machine]은 딥러닝이 지금처럼 맹위를 떨치기 전까지는 가장 높은 인기를 구가하던 머신러닝 알고리즘의 하나였다. 성능이 우수하고 선형과 비선형

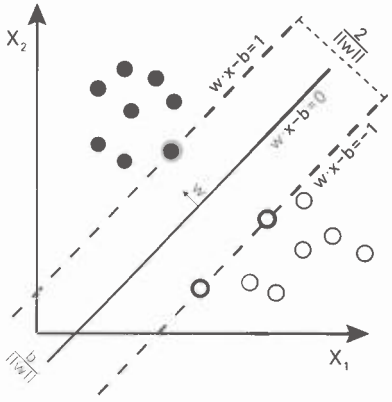
10 출처: http://goo.gl/xOZFwr

데이터에 모두 적용할 수 있으며 대용량 데이터도 처리할 수 있어서 여러 분야에서 많이 활용되었다.

SVM의 기본 아이디어는 주어진 데이터를 분류하기 위한 최적의 경계선을 찾고, 분류할 데이터가 들어왔을 때 경계선의 어느 쪽에 위치하느냐에 따라 분류하는 것이다. 최적경계선Decision Boundary은 분류하려는 데이터의 경계에 위치한 여러 경계선 중 가장 먼거리에 있는 경계선이 되며, 최적경계선 근처에 위치한 데이터들은 'Support Vector'라고 한다.

예를 들어, 아이리스의 품종을 분류하는 머신러닝 프로그램을 만든다고 하자. 문제를 단순화하기 위해 Setosa와 Versicolor의 2가지 종류로 구분한다고 가정하고, 검은 원은 Setosa, 하얀 원은 Versicolor를 나타낸다. 주어진 데이터를 산점도로 그려보면 다음과 같다.

그림 4-12 아이리스 품종의 산점도 SVM

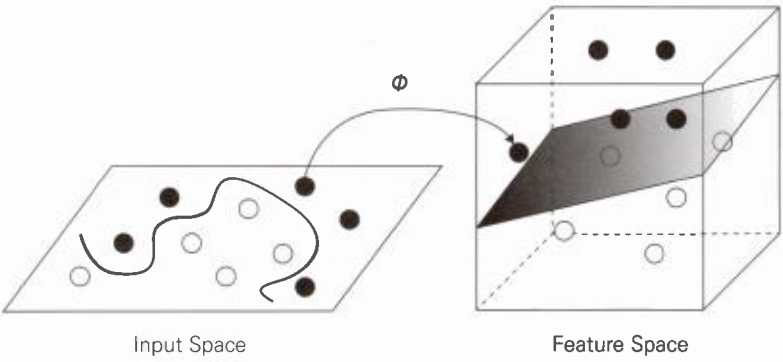


이런 상황에서 SVM은 Setosa와 Versicolor를 구분할 수 있는 여러 개의 선 중에서 거리 w가 가장 큰 값을 가지는 경계선을 최적경계선으로 선택하고 이 최적경계선을 결정하는 데 사용한 데이터들, 즉, 점선으로 표시된 2개의 선에 위치한 데이터들이 Support Vector가 된다.

최적경계선을 직선과 같은 선형으로 분리할 수 있다면 이와 같은 방법을 써서 분류하는 것이 가능하지만, 선형으로 분리되지 않는다면 커널 트릭Kernel Trick 기법을 써서 데이터를 분류한다.

[그림 4-13]은 커널 트릭의 개념을 나타내는 것으로, 흰색 원과 검은색 원을 분류하기 위해 SVM이 어떻게 하는지를 보여준다. 그림에서 알 수 있듯이 흰색 원과 흰색 원은 직선과 같은 형태의 선형적인 최적경계선으로 분류할 수 없다. 이런 경우에 SVM은 커널 트릭을 써서 원래의 데이터가 존재하는 Input Space를 다른 차원의 공간인 Feature Space로 변환해 선형적인 최적경계선을 찾는다. SVM은 이와 같은 방식으로 비선형 분류를 해결한다.

그림 4-13 커널 트릭[11]



SVM은 이해하기 어려운 알고리즘으로 알려졌지만, 사용자 입장에서는 기본 개념만 파악하면 Scikit-learn 라이브러리로 쉽게 SVM을 사용할 수 있다.

4.7 머신러닝 모델 구현

이번에는 앞서 설명한 3가지 머신러닝 알고리즘을 파이썬과 Scikit-learn 라이브러리를 이용해 주가방향을 예측할 수 있는 간단한 주가방향 예측변수Stock

11 출처: https://goo.gl/622NV9

Direction Predictor를 구현한다. Scikit-learn 라이브러리는 머신러닝의 많은 알고리즘과 관련 유틸리티를 포함하고 있어 머신러닝을 개발하는 데 필수라고 할 수 있고, 문서화가 잘 되어 있으며 라이브러리를 사용하는 방법도 일관성이 있어 개발 생산성 또한 높다.

만들려는 주가방향 예측변수의 기본 개념은 전날의 종가 또는 거래량 데이터를 이용해 다음날의 주가 방향을 예측하는 것이다. 사용자는 입력변수로 종가나 거래량 중 하나만 사용할 수도 있고, 2개를 모두 사용할 수도 있다. 또한, 하루 전의 데이터를 기준으로 예측할지 3일 전의 데이터를 기준으로 예측할지를 지정할 수 있다.

4.7.1 데이터셋

주가방향 예측변수의 학습과 테스트에 사용할 데이터셋은 앞서 야후 파이낸스에서 수집한 주가 데이터를 이용한다. 하나의 데이터셋을 만들고, 만들어진 데이터셋을 일정 비율로 나누어 학습과 테스트에 사용한다.

학습용 데이터셋은 주가방향 예측변수를 학습시키기 위해 사용하는 것으로, 이 데이터를 이용해 각 주가방향 예측변수 모델을 완성한다. 테스트용 데이터셋은 완성된 주가방향 예측변수를 테스트하기 위한 용도로, 학습용 데이터셋과 겹치지 않아야 하며, 당연히 학습에 이용하면 안 된다. 테스트용 데이터셋이 학습에 이용되면 테스트셋의 데이터가 해당 모델에 영향을 미쳐 올바른 테스트가 될 수 없다.

일반적으로 학습용 데이터셋과 테스트용 데이터셋은 전체를 100으로 봤을 때 75:25의 비율로 나누어 많이 사용한다. 필요에 따라서는 이 비율을 조정할 수 있으나 너무 적은 데이터로 학습시키면 과소적합UnderFitting 문제가 발생할 수 있고, 반대로 너무 많은 데이터로 학습시키면 과적합 문제가 발생할 수 있으며 주가방향 예측변수의 성능을 파악하는 데 필요한 테스트 자료가 부족해 테스트 결과를 신뢰할 수 없게 된다.

다음 코드는 데이터셋을 만드는 make_dataset( ) 함수다.

```
def make_dataset(df, time_lags=5):
    df_lag = pd.DataFrame(index=df.index)
    df_lag["Close"] = df["Close"]
    df_lag["Volume"] = df["Volume"]

    df_lag["Close_Lag%s" % str(time_lags)] = df["Close"].shift(time_lags)
    df_lag["Close_Lag%s_Change" % str(time_lags)] = df_lag["Close_Lag%s" %
str(time_lags)].pct_change()*100.0

    df_lag["Volume_Lag%s" % str(time_lags)] = df["Volume"].shift(time_lags)
    df_lag["Volume_Lag%s_Change" % str(time_lags)] = df_lag["Volume_Lag%s" %
str(time_lags)].pct_change()*100.0

    df_lag["Close_Direction"] = np.sign(df_lag["Close_Lag%s_Change" % str(time_
lags)])
    df_lag["Volume_Direction"] = np.sign(df_lag["Volume_Lag%s_Change" %
str(time_lags)])

    return df_lag.dropna(how='any')
```

make_dataset( ) 함수의 역할은 인자로 전달된 DataFrame의 데이터를 바탕으로 학습과 테스트에 사용할 DataFrame을 만들어 돌려주는 것이다.

df_lag["Close"]는 종가, df_lag["Volume"]은 거래량, df_lag["Close_Lag%s"]와 df_lag["Volume_Lag%"]는 사용자가 지정한 일자만큼의 종가, 거래량을 나타내는 입력변수다.

분류는 미래의 주가방향을 예측하기 위해 과거의 데이터를 이용하는데, time_lags 인자는 현재일 기준으로 며칠 전의 데이터를 이용할 것인가를 지정한다. 예를 들어, time_lags=1이라면 현재일 - 1, 즉 전날의 데이터를 기준으로 예측할 수 있도록 데이터를 만들고, time_lags=5라면 5일 전의 데이터로 예측한다.

pct_change( ) 함수는 pandas에서 제공하는 것으로, 주어진 데이터의 변화를 퍼센트로 계산하는 함수이다. 주가방향을 예측하는 분류는 지도학습이므로 입

력변수와 매치되는 출력변수를 만들어줘야 한다. 예를 들어, 종가가 1000이라면 이 값이 주가가 상승한 것인지 하락한 것인지를 출력변수로 만들어야 한다. 'Direction'이라는 접미사가 붙은 데이터가 출력변수다.

코드를 보면 Close_Direction과 Volume_Direction이 있는데, 앞은 주가의 방향을, 뒤는 거래량의 방향을 나타낸다. 예를 들어, 주가가 전날 대비 올랐으면 Close_Direction은 +1, 내렸으면 -1의 값을 가진다. 거래량 방향을 나타내는 Volume_Direction도 동일하다.

4.7.2 데이터셋 나누기

make_dataset( ) 함수를 이용해 주가방향 예측변수에 사용할 데이터셋을 만들었으면, 이를 학습에 사용할 데이터셋과 테스트에 사용할 데이터셋으로 나누어야 한다. split_dataset( ) 함수는 주어진 데이터셋을 사용자가 지정한 입력변수와 출력변수로 일정 비율로 나누어 준다.

```
def split_dataset(df,input_column_array,output_column,spllit_ratio):
    split_date = get_date_by_percent(df.index[0],df.index[df.shape[0]-1],spllit_
ratio)

    input_data = df[input_column_array]
    output_data = df[output_column]

    X_train = input_data[input_data.index < split_date]
    X_test = input_data[input_data.index >= split_date]
    Y_train = output_data[output_data.index < split_date]
    Y_test = output_data[output_data.index >= split_date]

    return X_train,X_test,Y_train,Y_test

def get_date_by_percent(start_date,end_date,percent):
    days = (end_date - start_date).days
    target_days = np.trunc(days * percent)
    target_date = start_date + datetime.timedelta(days=target_days)
    return target_date
```

input_column_array는 입력변수로 사용할 DataFrame의 칼럼 이름을 배열 형태로 전달하는 인수고, output_column은 출력변수로 사용할 칼럼 이름이다. split_ratio는 학습과 테스트로 나눌 비율을 지정한다. 예를 들어, input_column_array=["Close"], output_column="Close_Direction", split_ratio=0.75라고 인자를 설정하면, 입력변수로 df_lag["Close"]를 사용하고 출력변수는 df_lag["Close_Direction"], 전체 데이터셋의 75%를 학습용 데이터셋, 25%를 테스트용 데이터셋으로 나눈다.

get_date_by_percent() 함수는 주어진 데이터셋의 시작일과 마감일의 날수를 계산해 사용자가 지정한 split_ratio에 따라 계산해 날수를 돌려주는 함수다. 시계열 데이터이기 때문에 학습과 테스트에 사용할 데이터를 무작위로 추출하지 않고 시간을 기준으로 나눠야 한다.

X_train은 학습에 사용할 입력변수, Y_train은 학습에 사용할 출력변수고 X_test, Y_test는 테스트에 사용할 데이터셋이다.

4.7.3 주가방향 예측변수 작성

데이터셋이 모두 준비되었으니 이제 예측 프로그램을 만들어 볼 차례다. Scikit-learn은 사용하는 로지스틱 회귀, 랜덤 포레스트, SVM 등의 머신러닝 알고리즘과 상관없이 동일한 방법으로 예측 프로그램을 만들 수 있다. 너무 코드가 간단해서 보는 순간 직관적으로 이해할 수 있을 정도다.

```
def do_logistic_regression(x_train,y_train):
    classifier = LogisticRegression()
    classifier.fit(x_train, y_train)
    return classifier

def do_random_forest(x_train,y_train):
    classifier = RandomForestClassifier()
    classifier.fit(x_train, y_train)
    return classifier
```

```
def do_svm(x_train,y_train):
    classifier = SVC()
    classifier.fit(x_train, y_train)
    return classifier
```

do_logistic_regression( ) 함수는 logistic_regression 기반의 예측 프로그램을, do_random_forest( ) 함수는 random_forest 기반의 예측 프로그램을 생성하고 학습시킨 후에 반환한다. 코드에서 볼 수 있듯이 알고리즘에 상관없이 동일한 형태의 코딩 스타일이어서 매우 편리하다. 각 알고리즘의 파라미터는 생성 시에 인자로 전달할 수 있다. 예를 들어, SVM의 파라미터인 gamma와 C값을 지정하고 싶다면 다음 코드처럼 하면 된다.

```
classifier = SVC(gamma=0.001, C=100)
```

fit( ) 함수는 주어진 학습용 데이터셋을 알고리즘에 학습시키는 것으로 입력변수와 출력변수를 인자로 받는다.

4.7.4 주가방향 예측변수 실행 및 평가

앞서 설명한 내용들을 모두 코드로 작성하고 각각의 머신러닝 알고리즘에 대한 주가방향 예측력을 평가해보자.

```
def make_dataset(df, time_lags=5):
    df_lag = pd.DataFrame(index=df.index)
    df_lag["Close"] = df["Close"]
    df_lag["Volume"] = df["Volume"]

    df_lag["Close_Lag%s" % str(time_lags)] = df["Close"].shift(time_lags)
    df_lag["Close_Lag%s_Change" % str(time_lags)] = df_lag["Close_Lag%s" %
str(time_lags)].pct_change()*100.0

    df_lag["Volume_Lag%s" % str(time_lags)] = df["Volume"].shift(time_lags)
    df_lag["Volume_Lag%s_Change" % str(time_lags)] = df_lag["Volume_Lag%s" %
str(time_lags)].pct_change()*100.0
```

```
    df_lag["Close_Direction"] = np.sign(df_lag["Close_Lag%s_Change" % str(time_
lags)])
    df_lag["Volume_Direction"] = np.sign(df_lag["Volume_Lag%s_Change" %
str(time_lags)])

    return df_lag.dropna(how='any')

def split_dataset(df,input_column_array,output_column,spllit_ratio):

    split_date = get_date_by_percent(df.index[0],df.index[df.shape[0]-1],spllit_
ratio)

    input_data = df[input_column_array]
    output_data = df[output_column]

    X_train = input_data[input_data.index < split_date]
    X_test = input_data[input_data.index >= split_date]
    Y_train = output_data[output_data.index < split_date]
    Y_test = output_data[output_data.index >= split_date]

    return X_train,X_test,Y_train,Y_test

def get_date_by_percent(start_date,end_date,percent):
    days = (end_date - start_date).days
    target_days = np.trunc(days * percent)
    target_date = start_date + datetime.timedelta(days=target_days)
    return target_date

def do_logistic_regression(x_train,y_train):
    classifier = LogisticRegression()
    classifier.fit(x_train, y_train)
    return classifier

def do_random_forest(x_train,y_train):
    classifier = RandomForestClassifier()
    classifier.fit(x_train, y_train)
    return classifier

def do_svm(x_train,y_train):
    classifier = SVC()
    classifier.fit(x_train, y_train)
    return classifier

def test_classifier(classifier,x_test,y_test):
    pred = classifier.predict(x_test)

    hit_count = 0
    total_count = len(y_test)
    for index in range(total_count):
```

```
        if (pred[index]) == (y_test[index]):
            hit_count = hit_count + 1

    hit_ratio = hit_count/total_count
    score = classifier.score(x_test, y_test)

    return hit_ratio, score

if __name__ == "__main__":

    for time_lags in range(1,6):
        print "- Time Lags=%s" % (time_lags)

        for company in ['samsung','hanmi']:
            df_company = load_stock_data('%s.data'%(company))

            df_dataset = make_dataset(df_company,time_lags)
            X_train,X_test,Y_train,Y_test = split_dataset(df_dataset,["Close_
Lag%s"%(time_lags)],"Close_Direction",0.75)
            #print X_test

            lr_classifier = do_logistic_regression(X_train,Y_train)
            lr_hit_ratio, lr_score = test_classifier(lr_classifier,X_test,Y_test)

            rf_classifier = do_random_forest(X_train,Y_train)
            rf_hit_ratio, rf_score = test_classifier(rf_classifier,X_test,Y_test)

            svm_classifier = do_svm(X_train,Y_train)
            svm_hit_ratio, svm_score = test_classifier(svm_classifier,X_test,Y_test)

            print "%s : Hit Ratio - Logistic Regreesion=%0.2f, RandomForest=%0.2f,
SVM=%0.2f" % (company,lr_hit_ratio,rf_hit_ratio,svm_hit_ratio)
```

앞의 코드는 삼성전자와 한미약품의 주가를 입력변수로 산정하고 time_lags를 1부터 5까지 변화시켜 예측치의 적중률을 출력한 것으로, 결과는 다음과 같다.

실행결과 1

```
- Time Lags=1
samsung : Hit Ratio - Logistic Regreesion=0.50, RandomForest=0.55, SVM=0.55
hanmi : Hit Ratio - Logistic Regreesion=0.53, RandomForest=0.37, SVM=0.37
- Time Lags=2
samsung : Hit Ratio - Logistic Regreesion=0.51, RandomForest=0.46, SVM=0.46
hanmi : Hit Ratio - Logistic Regreesion=0.54, RandomForest=0.34, SVM=0.34
- Time Lags=3
samsung : Hit Ratio - Logistic Regreesion=0.49, RandomForest=0.49, SVM=0.49
```

```
hanmi : Hit Ratio - Logistic Regreesion=0.54, RandomForest=0.41, SVM=0.41
- Time Lags=4
samsung : Hit Ratio - Logistic Regreesion=0.41, RandomForest=0.51, SVM=0.51
hanmi : Hit Ratio - Logistic Regreesion=0.54, RandomForest=0.36, SVM=0.36
- Time Lags=5
samsung : Hit Ratio - Logistic Regreesion=0.41, RandomForest=0.49, SVM=0.49
hanmi : Hit Ratio - Logistic Regreesion=0.54, RandomForest=0.39, SVM=0.39
```

로지스틱 회귀의 경우 평균 54%의 적중률로 삼성전자보다 한미약품 주가에 더 높은 예측력을 보이고, 삼성전자의 경우 SVM과 랜덤 포레스트의 예측력이 큰 차이가 없음을 알 수 있다.

입력변수에 주가와 거래량, 2개의 입력변수를 넣고 실행하면 다음과 같은 결과를 보여준다.

실행결과 2

```
- Time Lags=1
samsung : Hit Ratio - Logistic Regreesion=0.47, RandomForest=0.50, SVM=0.50
hanmi : Hit Ratio - Logistic Regreesion=0.55, RandomForest=0.57, SVM=0.57
- Time Lags=2
samsung : Hit Ratio - Logistic Regreesion=0.47, RandomForest=0.53, SVM=0.53
hanmi : Hit Ratio - Logistic Regreesion=0.54, RandomForest=0.54, SVM=0.54
- Time Lags=3
samsung : Hit Ratio - Logistic Regreesion=0.46, RandomForest=0.54, SVM=0.54
hanmi : Hit Ratio - Logistic Regreesion=0.53, RandomForest=0.44, SVM=0.44
- Time Lags=4
samsung : Hit Ratio - Logistic Regreesion=0.49, RandomForest=0.44, SVM=0.44
hanmi : Hit Ratio - Logistic Regreesion=0.59, RandomForest=0.49, SVM=0.49
- Time Lags=5
samsung : Hit Ratio - Logistic Regreesion=0.47, RandomForest=0.49, SVM=0.49
hanmi : Hit Ratio - Logistic Regreesion=0.53, RandomForest=0.51, SVM=0.51
```

삼성전자의 경우 로지스틱 회귀의 평균적중률이 주가만 입력변수로 사용했을 때 46.4%, 주가와 거래량을 사용했을 때는 47.2%로 약 0.8% 향상되었음을 알 수 있다.

4.8 시간가치 감소 효과

앞에서 알고리즘 트레이딩을 위한 알파 모델을 만들기 위해 평균회귀와 머신러닝을 이용한 2가지 모델에 대한 개념과 내용을 살펴봤다. 살펴본 모델들은 거칠게 말해 단순 이론을 소개하기 위한 모델은 아니다. 특히 평균회귀 모델의 경우 기본적인 개념과 구현방법은 알고리즘 트레이딩에서 실제 투자에 사용하는 모델과 유사하다고 할 수 있다.

차이는 실제 투자 환경에 적용되는 모델의 경우 이를 좀 더 정교하게 만들고 알고리즘 트레이딩을 실행하기 위한 전략에 맞춰 최적화한 것이라고 할 수 있다. 독자의 노력과 아이디어로 소개한 모델들을 어떻게 개선하고 전략적으로 실행하느냐에 따라 좋은 실적을 보여주는 모델이 될 수도 있다.

최근 알고리즘 트레이딩의 추세는 복잡한 고도의 수학적 모델보다는 비교적 단순한 모델을 많이 사용한다. 그 이유가 바로 이 절의 제목인 '시간가치 감소 효과Time Decay Effect' 때문이다.

엄청난 수익을 안겨주는 모델을 우연히 또는 각고의 노력 끝에 찾았다고 하자. 모델 개발자는 내가 찾은 알파 모델이 지속해서 수익을 만들어 내길 바랄 것이다. 그러나 이러한 바람은 순진한 상상으로 현실 세계에서는 통용되지 않는다. 전 세계 누구도 하나의 알파 모델로 오랫동안 지속해서 수익을 낼 수는 없다.

알파 모델을 만든 후에 일정 기간 페이퍼 테스트를 거치고, 실제 거래에서 테스트와 튜닝을 위해 많은 노력을 기울인다. 그리고 그 이후부터 어느 정도 시간 동안 수익을 내기 시작하고, 서서히 수익성이 떨어진다. 이것이 일반적인 알파 모델들의 성장과 쇠퇴 모습이다. 이처럼 시간이 지날수록 처음에 가졌던 우위Edge가 퇴색되어가는 것을 일컬어 '시간가치 감소 효과'라고 한다.

알고리즘 트레이딩은 시간가치 감소 효과와의 싸움이라고 할 수 있다. 어떻게든 알파 모델을 만들고, 만들어진 알파 모델의 시간가치 감소를 최대한 저지하면서

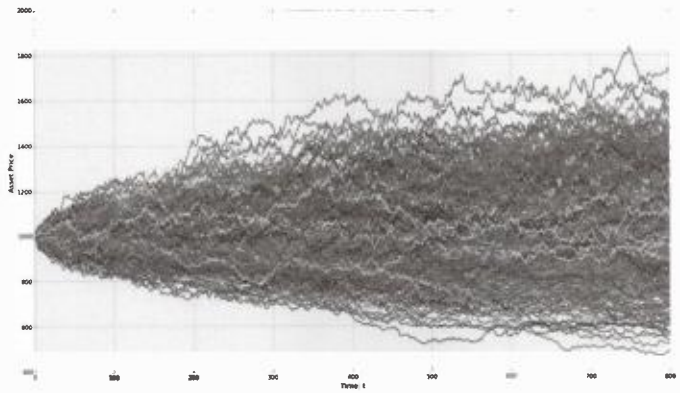
수익을 내고, 더는 우위가 없을 때는 새로운 알파 모델을 만들든지 실행전략을 수정하든지 아니면 다시 우위를 보이는 시간이 돌아올 때까지 기다려야 한다.

시간가치 감소 효과가 발생하는 이유는 여러 가지가 있지만 크게 2가지로 요약할 수 있다.

첫 번째, 알고리즘 트레이딩의 범람이라고 얘기할 수 있다. 앞서 설명했듯이 2012년에 알고리즘 트레이딩에 의한 거래량이 85%를 넘어섰을 정도로 많은 금융기관과 헤지펀드가 알고리즘을 사용하고 있다. 이들이 사용하는 알고리즘 트레이딩은 생각보다 그렇게 다양하지는 않아서 비슷한 수학적 이론과 모델들을 사용해 알고리즘 트레이딩을 실시하고 있다. 상황이 이렇다 보니 내가 찾은 알파 모델의 효과가 다른 알고리즘 트레이딩 시스템에 의해 희석되는 경향이 잦고, 또 힘들게 찾은 우위가 지속해서 유지되지 않는다. HFT가 인기를 얻고 있는 것도 이와 무관하지 않다.

두 번째, 주가가 표류하는 랜덤워크 모델을 따르기 때문이다. 표류하는 랜덤워크 모델은 다음 그림처럼 변동성이 일정하지 않고 계속 변화하는 모델로 처음에는 변동성이 작다가 시간이 지날수록 변동성이 커지는 특징이 있다.

그림 4-14 왜 예측하기 어려운가?[12]



12 출처: http://goo.gl/L2RGKr

알파 모델을 적용한 초기에는 상대적으로 변동성이 적기 때문에 예측력이 높지만, 시간이 흘러갈수록 변동성이 커져서 예측력은 당연히 낮아지게 된다.

지금 상황은 알고리즘 트레이딩의 일반화에 의해 앞서 얘기한 알고리즘 트레이딩의 범람이 표류하는 랜덤워크에 의한 변동성과 더불어 더욱더 많은 변동성을 일으키고 시간가치 감소 현상이 오히려 심화하고 있다고 생각한다. 어떤 이유에서든 알고리즘 트레이딩을 하겠다고 결심하는 순간, 여러분의 숙명은 모델과 알고리즘 트레이딩 시스템을 끊임없이 개선해 우위를 확보하는 것이다.

그렇다고 변동성을 나쁘게 볼 것은 아니다. 변동성이 있어서 알고리즘 트레이딩을 할 수 있으므로 오히려 고맙게 생각해야 한다. 변동성 없이 일정하게 주가와 각종 금융상품이 움직인다면 주식과 같은 금융시장은 애초부터 존재할 수 없었을 것이다. 오히려 변동성을 즐겨야 한다.

chapter 5

알고리즘 트레이딩 시스템 구현

이번 장에서는 간단한 알고리즘 트레이딩 시스템을 구현할 것이다. 앞서 설명한 평균회귀 모델과 머신러닝 모델을 모두 이용해 알고리즘 트레이딩을 적용할 만한 종목을 찾고, 해당 종목의 데이터로 모델을 완성하고, 실제 데이터를 입력해 어느 정도의 예측력을 갖는지 알아보겠다.

1절에서는 알고리즘 트레이딩 시스템의 개요에 대해 설명하고, 일반적인 알고리즘 트레이딩 시스템의 구성도와 파트별 기능을 설명한다. 또한, 이번 장에서 개발하려는 알고리즘 트레이딩 시스템의 전체적인 기능과 개요를 설명한다. 2절에서는 언어와 사용하는 라이브러리, 프로그래밍 환경에 대해 다루고, 3절부터는 알고리즘 트레이딩을 위해 필요한 데이터를 얻을 수 있는 소스와 방법들을 소개하고, 이를 `DataCrawler` 클래스로 구현한다.

이번 장과 다음 장은 구현에 관련된 내용이어서 코드가 많은데, 기초적인 사항은 건너뛰고 의미가 있는 코드 위주로 설명하겠다.

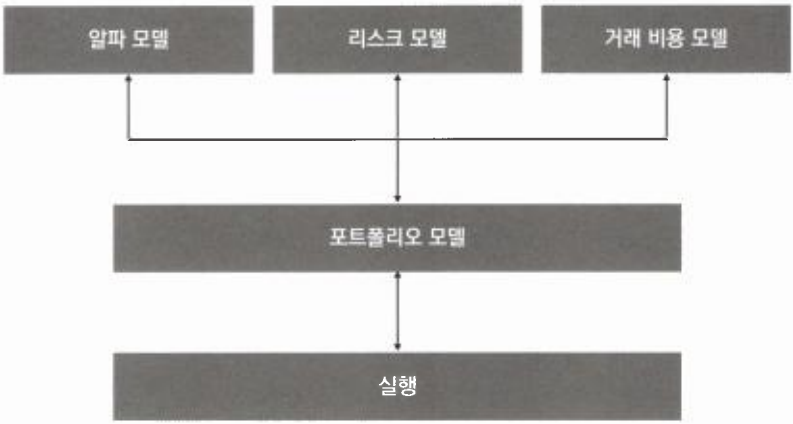
5.1 일반적인 알고리즘 트레이딩 시스템 구성

알고리즘 트레이딩은 이제 90% 이상의 거래를 처리할 정도로 금융에서 중추적인 역할을 담당하고 있다. 알고리즘 트레이딩 시스템은 사람의 개입 없이 정해진 모델에 따라 주식을 사고팔며, 적게는 수백에서 수억 원, 많게는 수천억 원의 돈이 움직인다. 이에 조그마한 프로그래밍 실수나 모델의 성능에 따라 엄청난 손실을

초래할 수도 있기 때문에 우리가 사용하는 그 어떠한 IT 시스템보다도 중대하다고 이야기해도 과장이 아니다.

그렇다면 월스트리트나 헤지펀드들이 수익 창출을 목적으로 사용하는 알고리즘 트레이딩의 실제 구성은 어떠할지 궁금하지 않을 수 없다. 다음 그림은 일반적으로 많이 사용하는 알고리즘 트레이딩 시스템의 기능과 구성도를 블록 다이어그램으로 나타낸 것이다.

그림 5-1 일반적인 알고리즘 트레이딩 시스템의 블록 다이어그램



알고리즘 트레이딩 시스템을 실제 구현하는 사람과 회사, 목적에 따라 세부 그림은 달라질 수 있으나 일반적으로 [그림 5-1]처럼 구성된다. 이 그림은 알고리즘 트레이딩에 직접 연관된 것만 표시한 것으로, 개발된 알고리즘 트레이딩 시스템의 성능을 파악하기 위한 백테스터Backtester나 데이터 저장을 위한 부분들은 생략하였다. 5가지 블록의 의미는 다음과 같다.

- **알파 모델** 주가나 주가 방향 등을 예측하기 위한 모델을 의미하며, 하나의 모델을 사용하는 경우도 있고 복수 개의 모델을 사용할 수도 있다.
- **리스크 모델**Risk Model 거래했을 때 예측이 틀렸다면 어느 정도의 손실을 보는지, 손실이 발생할 확률은 얼마인지를 계산하는 등 위험도를 측정하는 모델이다.

- 거래 비용 모델Transaction Cost Model 실제로 거래하면 그에 따른 수수료나 세금 등의 비용이 발생하는데, 이러한 모든 비용을 계산하는 모델이다.
- **포트폴리오 모델**Portfolio Model 알파 모델, 리스크 모델, 거래 비용 모델 이 3가지 모델의 결과값을 이용해 최종적으로 거래할지 말지를 결정하고, 거래의 규모는 얼마가 좋은지를 산정하는 모델이다.
- **실행**Execution 포트폴리오 모델의 결정에 따라 실제로 거래한다.

이 5가지 블록 중 가장 중요한 것은 포트폴리오 모델로, 알고리즘 트레이딩의 핵심이고 가장 많은 노하우가 들어가 있으며 최종으로 거래 여부와 거래 규모를 결정하므로 수익과 직결된 모듈이다.

알고리즘 트레이딩 시스템의 구성은 굉장히 재미있다. 끊임없이 거래 기회를 찾는 알파 모델과 끊임없이 거래 위험을 계산하는 리스크 모델이 공존하고, 포트폴리오 모델에서는 양쪽 모델의 수치를 보고 거래 여부를 결정하기 때문이다. 비유해서 설명하면 포트폴리오 모델은 최종결정권자라고 할 수 있고, 알파 모델은 긍정적인 견해로 끊임없이 수익을 낼 수 있다고 주장하는 사람, 리스크 모델은 부정적인 견해로 알파 모델이 제시한 거래 기회가 손실로 이어질 수 있고 그 확률이 얼마인지를 이야기하는 사람이라고 할 수 있다. 포트폴리오 모델은 양쪽의 이야기를 모두 듣고 자신만의 로직을 이용해 최종 거래 여부를 결정하는 것으로 인간사회에서 흔히 관찰되는 구조다.

최근에는 HFT의 인기로 인해 실행 모듈 역시 중요한 위치로 부상했다. 포트폴리오 모델에서 매입을 결정했다면 이를 얼마나 빨리, 원하는 수량을 살 수 있는가도 수익에 많은 영향을 미치기 때문이다. 예를 들어, 10,000주의 주식을 산다고 가정해보자. 10,000주의 주식은 적은 양이 아니므로 이를 한 번에 살 수 없는 경우가 많다. 이럴 때 몇 번에 나눠서 사야 하며, 어떻게 나눠서 사야 가장 수익을 극대화할 수 있는지를 고민해야 한다. 실행 모듈은 이러한 문제를 다루고 있다.

5.2 구현 시스템 개요

구현할 알고리즘 트레이딩 시스템은 앞 장에서 설명한 모델들을 실제 프로그램 코드로 작성하고 이를 테스트하는 데 초점을 맞춘 초보적인 수준이다. 앞서 설명한 모델들은 모든 종목을 대상으로 적용할 수 있는 것은 아니다. 특히 평균회귀 모델은 여러 종목 중 정상과정Stationary Process의 특성이 있는 종목을 찾고 이 종목들에 평균회귀 모델을 적용하는 것이 핵심적인 아이디어이므로 정상과정의 특성이 있는 종목을 찾아주는 그 무엇인가가 필요하다.

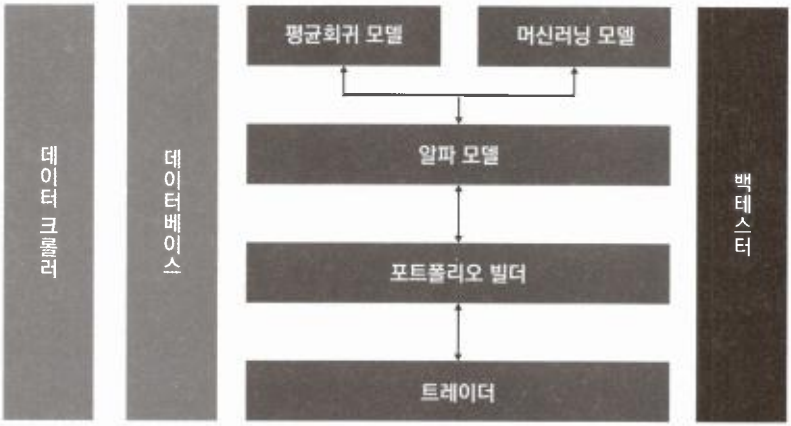
머신러닝의 경우에는 머신러닝 학습 후에 학습의 품질 정도를 나타내는 점수Score를 이용해 일정 기준으로 선별하는 방식을 활용한다. 머신러닝이든 평균회귀 모델을 적용하든 기본은 주가 관련 데이터를 받아와야 하므로 야후 파이낸스에서 필요한 주가 데이터를 다운로드하고 데이터베이스에 저장하는 기능 또한 필수다.

다운로드한 데이터로 알고리즘 트레이딩에 사용할 주식을 선택한 후에는 이를 과거 데이터에 적용해 어느 정도의 적중률을 보여주는지를 확인해야 한다. 평균회귀 모델은 ADF, 허스트 지수, Half-life를 이용해 비교적 알고리즘 트레이딩에 적합한 종목을 선택하고 머신러닝 모델은 점수 등을 이용해 우수한 성능을 보여주는 모델을 선별했다 하더라도, 알고리즘 트레이딩에 적용하려는 모델의 성능이 얼마나 되는지를 평가하는 데에는 한계가 있을 수밖에 없다. 이는 모델을 찾고 완성하기 위해 적절한 파라미터를 탐색하는 과정 등이 이미 지나간 과거에 대한 데이터지 내가 알고 싶은 미래에 대한 데이터Unseen Data가 아니기 때문이다.

모델을 완성하는 데 필요한 파라미터를 찾기 위해 과거 데이터에 맞춰 최적화하면, 그 모델은 과거 데이터에는 우수한 성능을 보이지만, 정작 관심 있는 미래 데이터에는 형편없는 결과를 보여주는 수가 왕왕 있다. 따라서 적절한 모델의 선정과 평가 또한 모델 개발만큼이나 중요한 부분이다.

여기서 간단히 구현할 알고리즘 트레이딩 시스템의 전체적인 모습은 다음과 같다.

그림 5-2 구현할 알고리즘 트레이딩 시스템의 블록 다이어그램



각 블록의 의미는 다음과 같다.

- **데이터 크롤러**Data Crawler 야후 파이낸스에서 지정한 주가 데이터를 다운로드하고, 이를 MySQL 데이터베이스에 저장한다.
- **데이터베이스** MySQL 데이터베이스로, 주가 데이터와 종목코드 데이터를 저장한다.
- **평균회귀 모델** 정상과정을 찾기 위한 ADF, 허스트 지수, Half-life 3가지 방법이 있다.
- **머신러닝 모델** 주가의 상승, 하락 방향을 예측하는 분류자Classifier로, 로지스틱 회귀, 랜덤 포레스트, SVM의 3가지 알고리즘이 있다.
- **알파 모델** 평균회귀 모델과 머신러닝 모델 등 알파 모델에 대한 추상적 모델이다.
- **포트폴리오 빌더**Portfolio Builder 평균회귀 모델과 머신러닝 모델을 이용해 알고리즘 트레이딩에 적합한 주식종목을 선정하고, 이 주식종목들을 관리한다.
- **트레이더**Trader 포트폴리오 빌더에 저장한 종목들에 대한 매도·매수 등의 거래를 한다.
- **백테스터**Backtester 과거 데이터를 적용해 예측 적중률이 얼마인지를 계산한다.

구현할 알고리즘 트레이딩 시스템은 실제 현업에서 투자에 사용하는 알고리즘 트레이딩보다 매우 간략화된 형태로 이 책에서 설명한 내용이 어떻게 알고리즘 트레이딩으로 연결되는지를 보여주는 데 주안점을 두었다는 것을 염두에 두기 바란다.

5.3 개발 환경

알고리즘 트레이딩 시스템을 개발하는 데 필요한 환경을 정리하면 다음과 같다.

- **언어** Python 2.7
- **운영체제** Linux, OS X, Windows 등 파이썬과 관련 라이브러리가 동작할 수 있는 환경
- **데이터베이스** MySQL
- **라이브러리**
  - pandas: 금융 데이터 라이브러리
  - Numpy: 과학 계산 라이브러리
  - matplotlib: 그래프 라이브러리
  - Beautiful Soup: 데이터 수집용 라이브러리
  - MySQLdb: MySQL 라이브러리
  - Scikit-learn: 머신러닝 라이브러리
  - Statsmodels: 통계 라이브러리

개발 환경은 파이썬과 라이브러리를 사용할 수 있는 환경이라면 선호도에 따라 선택하면 된다. 사용하는 라이브러리들은 대부분 비교적 쉽게 설치되고 특히 Anaconda와 같은 설치 프로그램을 이용하면 파이썬부터 관련 라이브러리까지 모두 한 번에 설치해주기 때문에 매우 편리하다.

5.4 데이터 크롤러 구현

알고리즘 트레이딩 시스템에서 첫 번째 개발할 부분은 주가 데이터를 수집하는 데이터 크롤러다. 데이터 크롤러는 pandas의 기능을 이용해 야후 파이낸스에서 데이터를 수집하고 MySQL에 저장한다. 야후 파이낸스에서 데이터를 수집하려면 다운로드하려는 종목의 코드값을 알아야 한다.

다운로드한 데이터는 시계열 데이터여서 '일자 + 데이터값'의 형식이고, 데이터 수집 방법과 데이터 소스에 따라 크기가 매우 커질 수도 있으므로 대용량 데이터

를 다루려고 한다면 MySQL 대신에 시계열 특화 데이터베이스인 openTSDB나 InfluxDB를 사용하는 것도 좋은 선택이 될 수 있다.

5.4.1 주식 종목코드 수집

야후 파이낸스에서 주가 데이터를 받아오려면 원하는 종목의 코드값을 입력해야 한다. 주식 종목코드는 코스콤[koscom] 사이트에서 받을 수 있다. 'http://datamall.koscom.co.kr/servlet/infoService/SearchIssue'를 브라우저 주소창에 입력하면 [그림 5-3]과 같은 화면이 나타나는데, 이 화면의 HTML을 분석해 종목코드를 추출할 수 있다. 추출된 종목코드와 종목명(회사명)은 컬렉션 클래스에 저장한다.

그림 5-3 종목코드 추출

다음은 [그림 5-3]의 HTML 중 종목코드와 종목명을 포함하는 일부분을 발췌한 내용이다.

```
<tr align="center">
<td colspan="8" style="PADDING-top: 10px;">
    <select name="stockCodeList" size="10" class="input_select"
style="height: 150px;" onDblClick="javascript:appendItem(document.searchFrm.
stockCodeList,document.searchFrm.selectedStockCodeList);">
    <option value="KR7060310000">(060310)3S</option>
    <option value="KR7095570008">(095570)AJ네트웍스</option>
    <option value="KR7068400001">(068400)AJ렌터카</option>
    <option value="KR7006840003">(006840)AK홀딩스</option>
    <option value="KRA006840144">(006840)AK홀딩스8R(폐지)</option>
    <option value="KR7054620000">(054620)AP시스템</option>
    <option value="KR7015671001">(015675)AP우주우B(폐지)</option>
    <option value="KR7015670003">(015670)AP우주통신(폐지)</option>
    <option value="KR7211270004">(211270)AP위성통신</option>
    <option value="KR7152100004">(152100)ARIRANG 200</option>
    <option value="KR7189400005">(189400)ARIRANG AC 월드(합성 H)</option>
    <option value="KR7141240002">(141240)ARIRANG K100EW</option>
    <option value="KR7122090004">(122090)ARIRANG KOSPI50</option>
```

Beautiful Soup는 HTML 파싱에 아주 유용하게 사용할 수 있는 라이브러리로, 주로 스크린 스크래핑Screen Scrapping에 아주 유용하게 사용할 수 있다(Beautiful Soup의 자세한 설명은 http://www.crummy.com/software/BeautifulSoup/를 참조하기 바란다).

requests.post( ) 함수로 종목검색창의 HTML을 받아와서 종목코드를 포함하는 부분을 HTML 태그로 검색하면 정규식이나 별도의 함수를 만들 필요 없이 간단하게 종목코드와 회사명을 가져올 수 있다.

downloadCode ( ) 함수는 코스콤 종목검색기에서 전체종목에 대한 코드값을 받아서 HTML로 반환한다. parseCodeHTML ( ) 함수는 인자로 전달된 HTML을 파싱하여 코드와 회사명을 추출하고, 코드를 관리하는 컬렉션 클래스인 StockCode에 추가한 뒤 컬렉션 클래스를 반환한다.

```
def downloadCode(self,market_type):
    url = 'http://datamall.koscom.co.kr/servlet/infoService/SearchIssue'
    html = requests.post(url, data={'flag':'SEARCH', 'marketDisabled': 'null',
'marketBit':market_type})
    return html.content

def parseCodeHTML(self,html,market_type):
    soup = BeautifulSoup.BeautifulSoup(html)
    options = soup.findAll('option')

    codes = StockCode()

    for a_option in options:
        #print a_tr
        if len(a_option)==0:
            continue

        code = a_option.text[1:7]
        company = a_option.text[8:]
        full_code = a_option.get('value')

    codes.add(market_type,code,full_code,company)

    return codes
```

StockCode는 종목코드를 관리하기 위한 것으로, 코드의 추가·삭제·검색과 종목수 등을 알려준다. 다음은 StockCode 컬렉션 클래스의 코드다.

```
class StockCode:
    def __init__(self):
        self.items = {}

    def count(self):
        return len(self.items)
```

```
    def clear(self):
        self.items.clear()

    def add(self,market_type,code,full_code,company):
        a_item = StockCodeItem(market_type,code,full_code,company)
        self.items[code] = a_item

    def remove(self,stock_code):
        del self.items[stock_code]

    def find(self,stock_code):
        return self.items[stock_code]

    def iterItems(self):
        return self.items.iteritems()

    def dump(self):
        index = 0
        for key,value in self.items.iteritems():
            print "%s : %s - Code=%s, Full Code=%s, Company=%s" % (index, value.
market_type, key, value.full_code, value.company)
            index += 1
```

수집된 데이터는 MySQL의 'codes' 테이블에 저장하며 스키마는 다음과 같다.

표 5-1 codes 테이블의 스키마

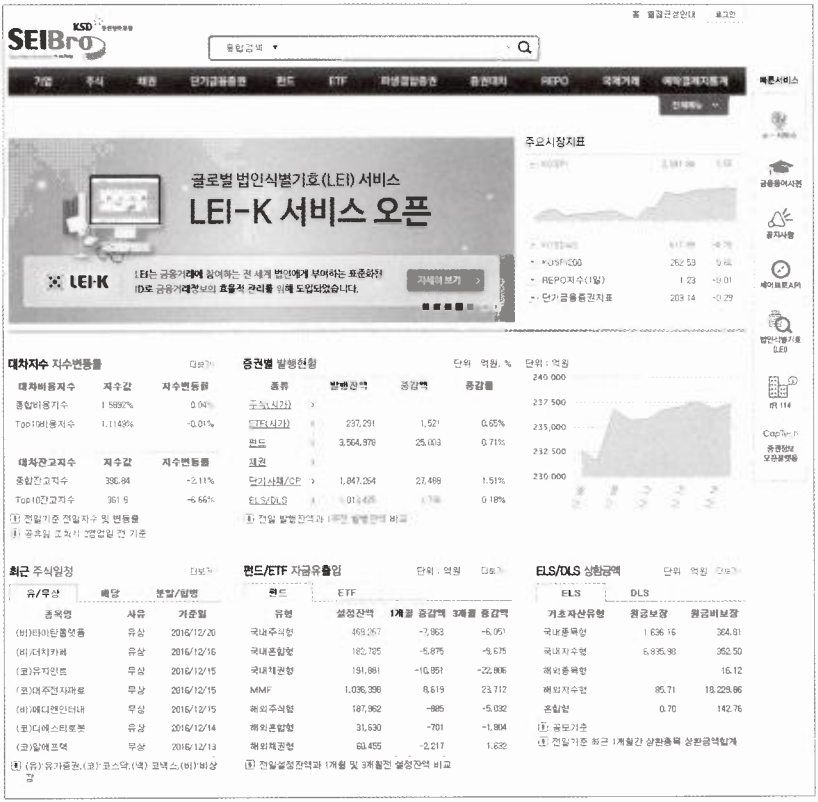
스키마	데이터 유형	설명
Id	integer	자동 증가(Auto Increment)
Last_update	varchar 8	마지막 데이터 업데이트 일자
Code	varchar 200	종목코드
Full_code	varchar 200	종목 전체코드
Market_type	integer,	시장종류, 1=코스피, 2=코스닥
Company	varchar 200	종목명

코드는 DataWriter 클래스의 updateCodeToDB( ) 함수를 이용해 codes 테이블에 저장한다.

5.4.2 주가 데이터 수집

주가 데이터는 pandas의 web.DataReader( ) 함수를 이용한다. 야후 파이낸스에서는 일 단위로 시가, 고가, 저가, 종가 등의 항목을 제공하고 있다. 한국예탁결제원에서는 오픈 API를 제공하는데, 이것을 이용하면 금융용어사전, 주식 정보서비스, 파생결합증권 정보서비스 등을 이용할 수 있다. 자세한 정보는 http://www.seibro.or.kr/을 방문하면 얻을 수 있다.

그림 5-4 한국예탁결제원 증권정보포털



데이터 수집 클래스인 DataCrawler의 downloadStockData( )는 인자로 주식 시장 종류, 종목코드, 데이터 수집 시작일, 마감일을 지정하면 야후 파이낸스에서 데이터를 받아오고 pandas DataFrame으로 반환한다. 야후 파이낸스에서 종목을 검색할 때 원래 종목코드에 '.KS'를 접미사로 붙어야 검색할 수 있는데, makeCode( ) 함수가 이러한 역할을 한다.

updateAllStockData( ) 함수는 주식시장 종류와 데이터 수집 시작일과 마감일을 지정하면 해당 기간 내의 모든 주가 데이터를 다운로드해 데이터베이스에 저장한다. 코스피의 경우 주식종목 수가 수백 개이므로 데이터 다운로드하는 데 시간이 걸린다.

```
def downloadStockData(self,market_type,code,year1,month1,date1,year2,month2,da
te2):
    def makeCode(market_type,code):
        if market_type==1:
            return "%s.KS" % (code)

            start = datetime(year1, month1, date1)
            end = datetime(year2, month2, date2)
        try:
            df = web.DataReader(makeCode(market_type,code), "yahoo", start, end)
            return df
        except:
            print "!!! Fatal Error Occurred"
            return None

    def updateAllStockData(self,market_type,year1,month1,date1,year2,month2,da
te2,start_index=1):
        print "Start Downloading Stock Data : %s , %s%s%s ~ %s%s%s" % (market_ty
pe,year1,month1,date1,year2,month2,date2)

        sql = "select * from codes"
        sql += " where market_type=%s" % (market_type)
        if start_index>1:
            sql += " and id>%s" % (start_index)
            rows = self.dbhandler.openSql(sql).fetchall()

            self.dbhandler.beginTrans()
```

```
            index = 1
            for a_row in rows:
                code = a_row[2]
                company = a_row[5]

                data_count = self.getDataCount(code)
                if data_count == 0:
                    print "... %s of %s : Downloading %s data " %
(index,len(rows),company)
                    df_data = self.downloadStockData(market_type,code,year1,mon
th1,date1,year2,month2,date2)
                    if df_data is not None:
                        df_data_indexed = df_data.reset_index()
                        self.dbwriter.updatePriceToDB(code,df_data_indexed)

                index += 1

            self.dbhandler.endTrans()

            print "Done!!!"
```

주가 데이터는 DataWriter 클래스의 updatePriceToDB( ) 함수를 통해 'prices' 테이블에 저장되며, 스키마는 다음과 같다.

표 5-2 prices 테이블의 스키마

스키마	데이터 유형	설명
Id	integer	자동 증가(Auto Increment)
Last_update	varchar 8	마지막 데이터 업데이트 일자
Price_date	datetime	주가일자
Code	varchar 200	종목코드
Price_open	integer	시가
Price_close	integer	종가
Price_high	integer	고가
Price_low	integer	저가
Price_adj_close	integer	보정된 종가
volume	integer	거래량

DataCrawler 클래스는 data_crawler.py 파일에 있으며, 이 클래스를 이용해 종목코드와 주가 데이터를 다운로드하려면 다음과 같이 코드를 실행한다.

```
crawler = DataCrawler()
crawler.updateAllCodes()
crawler.updateAllStockData(1,2010,1,1,2015,12,1)
```

이 코드는 DataCrawler를 이용해 모든 종목코드를 다운로드하고, 2010년 1월 1일부터 2015년 12월 1일까지의 코스피 시장 주가를 다운로드하는 예다.

5.5 알파 모델 구현

알파 모델은 수익을 내기 위한 기회를 포착하는 모델로, 여기서는 평균회귀 모델과 머신러닝 모델을 일컫는다. 이 모델들은 다음 2가지 중요한 기능이 있다.

- **모델 적합도 판별** 어떤 종목이 어느 모델에 적합한지를 결정한다.
- **포지션 결정** 매수·매도의 포지션을 결정하는데, 매수를 롱포지션Long Position, 매도를 숏포지션Short Position이라고 한다.

평균회귀 모델은 ADF, 허스트 지수, Half-life를 이용해 주식종목 적합도를 판별하고 머신러닝 모델은 점수를 이용한다. 포지션 결정은 주가 데이터가 주어졌을 때 매수해야 하는지, 매도해야 하는지를 선택하는 기능으로 추후 Trader 클래스와 연결되어 거래 시뮬레이션에 사용된다.

5.5.1 평균회귀 모델

다음 코드는 평균회귀 모델 클래스로, calcADF( ), calcHurstExponent( ), calcHalflife( )가 적합도 판별을 위한 함수들이다.

```
class MeanReversionModel(AlphaModel):
    def calcADF(self,df):
```

```
        adf_result = ts.adfuller(df)
        ciritical_values = adf_result[4]

        return adf_result[0], ciritical_values['1%'],ciritical_values['5%'],
ciritical_values['10%']

    def calcHurstExponent(self,df,lags_count=100):
        lags = range(2, lags_count)
        ts = np.log(df)

        tau = [np.sqrt(np.std(np.subtract(ts[lag:], ts[:-lag]))) for lag in lags]
        poly = np.polyfit(np.log(lags), np.log(tau), 1)

        result = poly[0]*2.0

        return result

    def calcHalfLife(self,df):
        price = pd.Series(df)
        lagged_price = price.shift(1).fillna(method="bfill")
        delta = price - lagged_price
        beta = np.polyfit(lagged_price, delta, 1)[0]
        half_life = (-1*np.log(2)/beta)

        return half_life

    def determinePosition(self,df,column,row_index,verbose=False):
        current_price = df.loc[row_index,column]

        df_moving_average = pd.rolling_mean(df.loc[0:row_
index,column],window=self.window_size)
        df_moving_average_std = pd.rolling_std(df.loc[0:row_
index,column],window=self.window_size)

        moving_average = df_moving_average[row_index]
        moving_average_std = df_moving_average_std[row_index]

        price_arbitrage = current_price - moving_average

        if verbose:
            print "diff=%s, price=%s, moving_average=%s, moving_average_std=%s"
% (price_arbitrage,current_price,moving_average,moving_average_std)

        if abs(price_arbitrage) > moving_average_std*self.threshold:

            if np.sign(price_arbitrage)>0:
                return SHORT
            else:
```

```
            return LONG
        return HOLD
```

determinePosition( ) 함수는 주어진 값에 따라 매수·매도 포지션을 결정하는데, 이동평균과 이동평균 표준편차값을 이용해 포지션을 결정한다. 예를 들어, 삼성전자의 현재 주가가 1,000,000이고, 이동평균은 900,000, 이동평균 표준편차가 80,000이라고 하자. 이동평균이 900,000이고 표준편차가 80,000이라는 것은 삼성전자 주가가 정규분포를 따른다고 가정해보면 주가가 900,000 ± 80,000 = 820,000~980,000의 범위에서 움직일 확률이 68.3%라는 것을 의미한다. 그런데 현재 주가가 1,000,000으로 980,000보다 크므로 앞으로 주가가 상승보다는 하락할 가능성이 더 크다고 판단할 수 있다. 따라서 이런 경우에는 매도 포지션을 갖도록 한다.

self.threshold는 이동평균과 현주가의 차이가 이동평균 표준편차보다 얼마나 크거나 작을 때 매수 또는 매도할지를 지정하는 계수로, 사용자가 지정해 확률을 조정할 수 있다. 예를 들어, self.threshold를 2로 지정하면 평균 -2s이므로 해당 범위 내에서 주가가 움직일 확률이 95.4%가 된다. 따라서 2로 지정한 값보다 현주가와 이동평균의 차이가 크다면 그 값은 매우 비정상적인 값으로, 이동평균보다 크다면 하락하고 반대로 이동평균값보다 작다면 상승하려는 경향이 매우 크다고 예측할 수 있다.

5.5.2 머신러닝 모델

머신러닝 모델 역시 앞서 얘기한 평균회귀 모델과 마찬가지로 각 예측변수의 성능을 파악하기 위한 calcScore( ) 함수와 매도·매수 포지션을 결정하기 위한 determinePosition( ) 함수를 가지고 있다.

calcScore( ) 함수는 Predictor 클래스의 trainAll( ) 함수를 이용해 학습결

과의 품질을 알려주는 점수를 얻는다. determinePosition( ) 함수는 보팅Voting 기법을 이용해 포지션을 결정한다. 보팅 기법은 여러 개의 머신러닝 알고리즘을 섞어서 쓰는 방법의 통칭인 앙상블Ensemble 기법의 하나로 'Voting'이라는 단어가 의미하는 것처럼 가장 많은 표를 얻은 결과가 최종 결과로 선정되는 다수결 방법이다.

예를 들어, 머신러닝 모델에는 로지스틱 회귀, 랜덤 포레스트, SVM의 3가지 예측 모델이 있는데, 값을 입력했더니 로지스틱 회귀는 상승, 랜덤 포레스트와 SVM은 하락이라는 예측치가 나왔다고 하자. 상승은 1개, 하락은 2개로 하락이 더 많으므로 최종 예측치는 '하락'이라고 결정하는 방법이다.

```
class MachineLearningModel(AlphaModel):
    def calcScore(self,split_ratio=0.75,time_lags=10):
        return self.predictor.trainAll(split_ratio=split_ratio,time_lags=time_
lags )

    def determinePosition(self,code,df,column,row_index,verbose=False):
        if (row_index-1) < 0:
            return HOLD

        current_price = df.loc[row_index-1,column]

        prediction_result = 0
        for a_predictor in ['logistic','rf','svm']:

            predictor = self.predictor.get(code,a_predictor)
            pred,pred_prob = predictor.predict([current_price])
            prediction_result += pred[0]

        if prediction_result>1:
            return LONG
        else:
            return SHORT
```

5.6 포트폴리오 빌더 구현

DataCrawler 클래스를 이용해 필요한 데이터를 다운로드했으면 이 데이터를 이용해 평균회귀 모델과 머신러닝 모델에 적합한 종목을 선택해야 한다. 한 가지 유의할 것은 알고리즘 트레이딩에서는 모든 종목을 선택하거나 무작위로 몇 종목을 선택해 트레이딩하지 않는다. 이는 굉장히 위험한 모험으로 반드시 피해야 하는 접근방법이다.

알고리즘 트레이딩은 수익을 낼 수 있는 알파모델을 만들고 트레이딩에 적용해 수익을 내는 방법인데, 모델을 만든다는 것 자체가 어떤 특성을 가진 주식종목만을 선별해야 수익을 낼 가능성이 크다. 예를 들어, 평균회귀 모델의 경우 핵심 아이디어는 많은 종목 중에 평균회귀 성향이 있는 종목이 있으니 이들 종목을 찾고, 여기에 평균회귀 모델을 적용하자는 가정으로부터 시작하는 것이다. 그런데 트레이딩할 종목이 평균회귀 성향이 없다면 평균회귀 모델을 적용했을 때의 수익률은 굳이 테스트할 필요가 없을 것이다.

머신러닝도 마찬가지 관점이다. 코스피에 여러 종목이 있지만, 이 종목 중에 머신러닝 모델이 우수한 성능을 보여주는 종목들만을 선별해 트레이딩하는 것은 이야기할 필요도 없는 너무 당연한 귀결이다. 따라서 종목 선정은 알고리즘 트레이딩의 성패를 좌우할 만큼 매우 중요한 과정이다.

구현하려는 알고리즘 트레이딩 시스템에서는 평균회귀 모델과 머신러닝 모델, 2가지를 적용하므로 이 두 모델에 적합한 종목을 각각 선별해야 한다. 이러한 역할을 하는 클래스가 PortfolioBuilder다. 주요한 역할은 모델별로 트레이딩에 사용할 종목들을 선정하고 선정된 종목들 관리하는 것이며, portfolio_builder.py 파일에 들어 있다.

PortpolioBuilder 클래스는 3단계로 종목선정을 하며, 절차는 평균회귀 모델과 머신러닝 모델 모두 동일하다.

1. 종목별 모델 적합성 평가
2. 적합성 평가 결과에 순위 지정
3. 정해진 순위에서 상위 n%의 종목 선정

모델에 따라 적합성 평가방법은 다르지만, 2번과 3번 순서는 개념상 거의 차이가 없다. 이렇게 최종으로 선택된 종목들이 트레이딩에 이용된다.

5.6.1 평균회귀 모델 종목 선정

앞장에서 살펴봤듯이 평균회귀 모델은 주가가 평균으로 회귀하려는 성향이 강한 종목에 쓸 수 있는 모델이다. 평균회귀 모델을 적용할 수 있을지, 없을지는 ADF 테스트와 허스트 지수, Half-life 계수를 구해보면 쉽게 알 수 있다. PortfolioBuilder의 doStationarityTest( ) 함수가 이 3가지 테스트를 이용해 평균회귀 성향이 어느 정도인지를 종목별로 계산하고 그 결과를 DataFrame으로 돌려주는 역할을 한다. ADF 테스트는 MeanReversionModel의 calcADF( ), 허스트 지수는 calcHurseExponent( ), Half-life는 calcHalf Life( ) 함수로 계산한다.

다음 코드에서 adf_5는 기각값 5%를, adf_10은 기각값 10%를 의미한다.

```
def doStationarityTest(self,column,lags_count=100):
    rows_code = self.dbreader.loadCodes(limit = self.config.get('data_limit'))

    test_result = {'code':[], 'company':[], 'adf_statistic':[], 'adf_1':[],'adf_5':[
],'adf_10':[], 'hurst':[],'halflife':[]}

    index = 1
    for a_row_code in rows_code:
        code = a_row_code[0]
        company = a_row_code[1]

        print "... %s of %s : Testing Stationarity on %s %s" % (index,len(rows_
code),code,company)

        a_df = self.loadDataFrame(code)
```

```
        a_df_column = a_df[column]

        if a_df_column.shape[0]>0:
            test_result['code'].append(code)
            test_result['company'].append(company)
            test_result['hurst'].append(self.mean_reversion_model.
calcHurstExponent(a_df_column,lags_count))
            test_result['halflife'].append(self. mean_reversion_model.
calcHalfLife(a_df_column))

            test_stat, adf_1,adf_5,adf_10 = self. mean_reversion_model.
calcADF(a_df_column)
            test_result['adf_statistic'].append(test_stat)
            test_result['adf_1'].append(adf_1)
            test_result['adf_5'].append(adf_5)
            test_result['adf_10'].append(adf_10)

        index += 1

    df_result = pd.DataFrame(test_result)

    return df_result
```

계산된 테스트 결과에 따라 종목별 적합도 순위는 rankStationarity ( ) 함수에서 담당한다. 이 함수는 각각의 테스트 결과의 전체 분포를 백분위수로 계산하고, 테스트 결과값이 전체 백분위수 중 어디에 위치하는지에 따라 순위를 매기는 방식을 취한다. 예를 들어, 삼성전자 종목의 ADF 값이 -1.5인데 이 값이 전체 20% 분위에 해당한다면 순위는 2위가 된다.

허스트 지수, Half-life 역시 동일한 방법으로 종목별 순위를 매긴다. assess ADF ( ), assessHurse ( ), assesHalflife ( )가 테스트별 순위를 지정하는 함수다.

```
def rankStationarity(self,df_stationarity):
    df_stationarity['rank_adf'] = 0
    df_stationarity['rank_hurst'] = 0
    df_stationarity['rank_halflife'] = 0

    halflife_percentile = np.percentile(df_stationarity['halflife'],
```

```
np.arange(0, 100, 10)) # quartiles

    for row_index in range(df_stationarity.shape[0]):
        df_stationarity.loc[row_index,'rank_adf'] = self.assessADF(df_
stationarity.loc[row_index,'adf_statistic'],df_stationarity.loc[row_
index,'adf_1'],df_stationarity.loc[row_index,'adf_5'],df_stationarity.loc[row_
index,'adf_10'])
        df_stationarity.loc[row_index,'rank_hurst'] = self.assessHurst(df_
stationarity.loc[row_index,'hurst'])
        df_stationarity.loc[row_index,'rank_halflife'] = self.
assessHalflife(halflife_percentile, df_stationarity.loc[row_index,'halflife'])
    df_stationarity['rank'] = df_stationarity['rank_adf'] + df_
stationarity['rank_hurst'] + df_stationarity['rank_halflife']

    return df_stationarity
```

테스트 순위 결과에 따라 종목 선정을 하는 함수가 buildUniverse( )다. 이 함수는 순위 정보의 백분위수를 구하고, 사용자가 지정한 비율에 해당하는 종목들만을 반환한다. 예를 들어, buildUniverse( ) 함수의 ratio 인자에 0.8이라고 입력하면, 모든 테스트 순위가 상위 80% 이상인 종목만을 최종 선택종목으로 반환한다.

```
def buildUniverse(self,df_stationarity,column,ratio):
    percentile_column = np.percentile(df_stationarity[column], np.arange(0,
100, 10))
    ratio_index = np.trunc(ratio * len(percentile_column))

    universe = {}

    for row_index in range(df_stationarity.shape[0]):
        percentile_index = getPercentileIndex(percentile_column, df_
stationarity.loc[row_index,column])
        if percentile_index >= ratio_index:
            universe[df_stationarity.loc[row_index,'code']] = df_stationarity.
loc[row_index,'company']

    return universe
```

다음은 portfolio 클래스를 이용해 평균회귀 모델을 선정하기 위한 코드다.

```
portfolio = PortfolioBuilder()

df_stationarity = portfolio.doStationarityTest('price_close')
df_rank = portfolio.rankStationarity(df_stationarity)
stationarity_codes = portfolio.buildUniverse(df_rank,'rank',0.8)
```

실행결과

```
{u'000157': u'두산2우B', u'000150': u'두산'}
```

이 코드는 종가 값(price_close)을 기준으로 ADF 테스트, 허스트 지수 등의 정상성 테스트를 시행하고 순위를 매긴 다음, 테스트 결과치가 상위 80% 이상인 항목들만을 돌려주는 예다.

5.6.2 머신러닝 모델 종목 선정

머신러닝 모델 역시 평균회귀 모델과 마찬가지로 머신러닝 모델이 좋은 성능을 보여줄 수 있는 종목을 선택하는 것이 수익을 내는 거대한 발걸음의 시작이라 할 수 있다. 하지만 머신러닝 모델은 평균회귀 모델처럼 수학이론을 바탕으로 하지 않기 때문에 상대적으로 선정에 적합한지, 그렇지 않은지를 판단하기 어렵다. 예를 들어, 평균회귀 모델은 ADF 테스트, 허스트 지수, Half-life 계수를 구하면 해당 종목이 평균회귀 성향이 있는지 아니면 랜덤워크인지를 어느 정도 파악할 수 있다.

머신러닝 모델은 이러한 수학 이론이 없고, 주어진 데이터에서 머신러닝 알고리즘으로 어떤 패턴을 찾는 것이어서 종목 선정에 대한 객관적 수치가 없다. 다시 말하면 머신러닝 모델은 학습결과가 좋다 나쁘다는 점수 값으로 알 수 있지만, 선택한 종목이 머신러닝에 적합한 특성이 있는 종목인지 아닌지는 알기 힘들다.

점수 값은 학습결과의 품질, 즉 학습에 사용한 데이터에 대한 성능을 나타내는 것으로, 평균회귀 모델 선정에 사용한 3가지 테스트는 종목별로 평균회귀 성향을 얼

마나 가졌는지를 알려주는 것과는 다른 성격이라는 것에 유의해야 한다.

doMachineLearningTest( ) 함수는 머신러닝에서 사용되는 3가지 알고리즘, 로지스틱 회귀, 랜덤 포레스트, SVM을 사용한 예측변수를 생성하고 학습시킨 뒤 종목별, 알고리즘별 점수를 DataFrame으로 반환한다.

```
def doMachineLearningTest(self,split_ratio=0.75,lags_count=10):
    return self.machine_learning_model.calcScore(split_ratio=split_ratio,time_
lags=lags_count )
```

앞의 코드를 봐서 알겠지만, 머신러닝의 경우 학습시킨 후 점수 값으로 판단해야 하기 때문에 MachineLearningModel 클래스의 calcScore( )라는 함수를 호출하였다. MachineLearningModel 클래스는 내부적으로 Predictors 클래스를 사용해 종목별, 알고리즘별로 예측변수를 생성, 학습, 저장, 관리한다. 예를 들어, 삼성전자 종목의 머신러닝 모델이라면 로지스틱 회귀, 랜덤 포레스트, SVM 3개의 예측모델을 생성하고 학습시키고, 추후 삼성전자 데이터로 학습시킨 머신러닝을 다시 불러와서 사용할 수 있어야 하는데, 이러한 것들을 모두 Predictors 클래스에서 처리한다.

Predictors 클래스의 trainAll( ) 함수는 학습에 사용할 데이터의 시간차인 time_lags와 사용자가 지정한 split_ratio로 학습용 데이터셋과 테스트용 데이터셋을 분리해 학습시킨다.

```
def trainAll(self, time_lags=5, split_ratio=0.75):
    rows_code = self.dbreader.loadCodes(self.config.get('data_limit'))

    test_result = {'code':[], 'company':[], 'logistic':[], 'rf':[], 'svm':[]}

    index = 1
    for a_row_code in rows_code:
        code = a_row_code[0]
        company = a_row_code[1]
```

```
        print "... %s of %s : Training Machine Learning on %s %s" %
(index,len(rows_code),code,company)

        df_dataset = self.makeLaggedDataset(code,self.config.get('start_
date'),self.config.get('end_date'), self.config.get('input_column'),self.config.
get('output_column'),time_lags=time_lags)

        if df_dataset.shape[0]>0:
            test_result['code'].append(code)
            test_result['company'].append(company)

            X_train,X_test,Y_train,Y_test = self.splitDataset(df_dataset,'price_
date',[self.config.get('input_column')],self.config.get('output_column'),split_
ratio)

            for a_clasifier in ['logistic','rf','svm']:
                predictor = self.createPredictor(a_clasifier)
                self.add(code,a_clasifier,predictor)

                predictor.train(X_train,Y_train)
                score = predictor.score(X_test,Y_test)

                test_result[a_clasifier].append(score)

                print "    predictor=%s, score=%s" % (a_clasifier,score)

        index += 1

    df_result = pd.DataFrame(test_result)

    return df_result
```

데이터베이스로부터 종목코드를 가져오는 함수인 loadCodes( )로 학습대상이 되는 종목코드들을 가져온다.

Predictors 클래스의 makeLaggedDataset( ) 함수는 start_date와 end_date로 지정된 일자에서 종목코드의 시간이 지연된 데이터셋을 만든다. 지연된 데이터셋은 학습에 사용할 데이터의 일자를 조정해 일부러 일자를 지연시킨 데이터를 가진다. 예를 들어, 머신러닝 모델은 주가의 방향이 상승할 것인지 하락할 것인지를 예측하는데, 예측하려면 학습된 머신러닝 모델(Predictor)에 데이터를 입력해주어야 한다. 하루 전 데이터를 이용해 오늘의 주가방향을 예측할 수도 있고, 5일 전 데이터를 이용할 수도 있다. 인자 time_lags=7을 입력하면

makeLaggedDataset( ) 함수는 7일 전 데이터를 이용해 오늘의 주가방향을 예측할 수 있는 데이터셋을 만든다.

splitDataset( ) 함수는 split_ratio 인자에 전달된 값에 따라 전체 데이터셋을 학습용 데이터셋과 테스트용 데이터셋으로 나눈다. 예를 들어, split_ratio=0.6이라고 지정하면 전체의 60%를 학습용 데이터셋으로, 40%를 테스트용 데이터셋으로 나눈다.

예측변수의 학습은 train( ) 함수를 이용하며, 학습결과의 품질은 score로 알 수 있다. score 값은 0에서 1사이의 실수를 가지며 1에 가까울수록 학습이 잘 이뤄졌음을 나타낸다. 학습결과를 score에 따라 순위를 매기는 함수는 rankMachineLearning( )이며, 코드는 다음과 같다.

```
def rankMachineLearning(self,df_machine_learning):
    def listed_columns(arr,prefix):
        result = []
        for a_item in arr:
            result.append( prefix % (a_item) )
            return result

    mr_models = ['logistic','rf','svm']

    for a_predictor in mr_models:
        df_machine_learning['rank_%s' % (a_predictor)] = 0

    percentiles = {}
    for a_predictor in mr_models:
        percentiles[a_predictor] = np.percentile(df_machine_learning[a_
predictor], np.arange(0, 100, 10))

        for row_index in range(df_machine_learning.shape[0]):
            df_machine_learning.loc[row_index,'rank_%s' % (a_predictor)] = self.
assessMachineLearning(percentiles[a_predictor],df_machine_learning.loc[row_
index,a_predictor])

    df_machine_learning['total_score'] = df_machine_learning[mr_models].
sum(axis=1)
    df_machine_learning['rank'] = df_machine_learning[ listed_columns(mr_
models,'rank_%s') ].sum(axis=1)

    return df_machine_learning
```

앞서 살펴본 평균회귀 모델에서 적용한 방법과 다름이 없고, 머신러닝 모델을 활용한 종목선택 함수는 buildUniverse ( )로 동일하다.

다음은 PorfolioBuilder 클래스를 이용해 평균회귀 모델과 머신러닝 모델로 종목을 선택하는 예제 코드다.

```
universe = Portfolio()
portfolio = PortfolioBuilder()

services.get('configurator').register('start_date','20150101')
services.get('configurator').register('end_date','20151130')
services.get('configurator').register('input_column','price_close')
services.get('configurator').register('output_column','indicator')
services.get('configurator').register('data_limit',10)

df_stationarity = portfolio.doStationarityTest('price_close')
df_rank = portfolio.rankStationarity(df_stationarity)
stationarity_codes = portfolio.buildUniverse(df_rank,'rank',0.8)

df_machine_result = portfolio.doMachineLearningTest( split_ratio=0.75,lags_
count=5 )
df_machine_rank = portfolio.rankMachineLearning(df_machine_result)
machine_codes = portfolio.buildUniverse(df_machine_rank,'rank',0.8)

universe.clear()
universe.makeUniverse('price_close','stationarity',stationarity_codes)
universe.makeUniverse('price_close','machine_learning',machine_codes)
universe.dump()
```

실행결과

```
>>> Portfolio.dump <<<
- model=machine_learning
... column=price_close : index=0, code=005610, company=삼립식품
... column=price_close : index=1, code=002270, company=롯데푸드
- model=stationarity
... column=price_close : index=0, code=000157, company=두산2우B
... column=price_close : index=1, code=000150, company=두산
--- Done ---
```

이 결과는 모델별로 선정된 종목을 표시한 것으로, 주어진 조건에서는 머신러닝 모델로 적합한 종목은 삼립식품과 롯데푸드, 평균회귀 모델에서는 두산2우B와 두산이 적합한 모델로 선정되었다.

services 클래스는 모델을 학습시키고 종목을 선택하는 데 필요한 환경 설정을 저장하고 관리하는 일종의 레지스트리로, 전역변수로 선언되어 모든 클래스에서 사용할 수 있다. 모델 선정에 연관된 주요한 환경변수는 다음과 같다.

- **start_date** 모델 선정에 사용할 데이터의 시작일
- **end_date** 모델 선정에 사용할 데이터의 마감일
- **input_column** 입력변수로 prices 테이블의 칼럼명을 사용한다. 예를 들어, 고가를 사용하고 싶다면 'price_high'를 입력한다.
- **output_column** 머신러닝에 사용할 출력변수 명칭을 지정하는 것으로 원하는 이름으로 지정한다.
- **data_limit** 모델 선정에 사용할 종목코드 수 지정, 전체 종목을 대상으로 하는 것은 시간이 오래 걸리므로 이 값을 이용해 수를 제한할 수 있다. 예를 들어, 10으로 입력하면, 최초의 10개 종목만 모델 선정에 사용하고, 0으로 입력하면 모든 종목을 모델 선정에 사용한다.

앞의 예에서는 종목선정을 위해 '20150101~20151130'의 데이터를 사용했는데 2010년 1월 1일부터 2015년 12월 30일까지로 변경하고 싶다면 다음과 같이 코드를 수정한다.

```
services.get('configurator').register('start_date','20100101')
services.get('configurator').register('end_date','20151230')
```

Portfolio 클래스는 PortfolioBuilder 클래스를 이용해 선정된 종목들을 저장하고 관리하기 위한 컬렉션 클래스로, 트레이딩에는 Portfolio 인스턴스를 인자로 전달한다.

5.7 트레이더 구현

Trader 클래스는 Portfolio 클래스에 담긴 종목들을 거래에 가상으로 적용해 각각의 모델에 따른 주식 매도·매수의 포지션을 기록하는 역할을 한다. 일반적인 알고리즘 트레이딩 시스템이라면 Trader 클래스가 매도·매수의 기록뿐만 아니라 실제 증권사에 연결해 포지션에 따라 거래하지만, 이 책에서는 이러한 기능은 포함하지 않는다. 많은 국내 증권사에서 주식거래 API를 지원하므로 관심 있는 독자들은 Trader 클래스를 참조하거나 확장해 실제 거래가 이뤄질 수 있도록 개선해보기 바란다.

Trader 클래스는 컬렉션 클래스로, 포지션 기록에 대한 것이어서 코드가 단순하다. Add( )는 새로운 거래 기록은 추가하는 함수고, dump( )는 기록된 거래 기록을 화면에 출력하는 함수다.

```
class MessTrader(BaseCollection):
    def setPortfolio(self, portfolio):
        self.portfolio = portfolio

    def add(self,model,code,row_index,position):
        if self.find(code) is None:
            self.items[code] = []

        a_item = TradeItem(model,code,row_index,position)
        self.items[code].append( a_item )

    def dump(self):
        print ">>> Portfolio.dump <<<"
        for key in self.items.keys():
            for a_item in self.items[key]:
                print "... model=%s, code=%s, row_index=%s, position=%s" % (a_
item.model,a_item.code,a_item.row_index,a_item.position)

        print "-- Done --"
```

chapter 6

성능 평가와 최적화

알고리즘 트레이딩 시스템을 만드는 것도 중요하지만, 더욱 관심을 쏟아야 하는 것은 만들어진 시스템에 대한 평가다. 개발된 시스템의 목적은 오류 없이 잘 동작하는 것뿐만 아니라, 트레이딩으로 수익을 내는 것이기 때문에 얼마나 많은 수익을 내줄 것인가를 평가하는 것이 핵심이라 할 수 있다. 일반적인 시스템의 경우에는 예상되는 시나리오를 상정하고, 테스트 프로그램으로 안정성과 성능을 측정하는 것이 가능하지만, 알고리즘 트레이딩은 그 경우가 많이 다르다.

알고리즘 트레이딩의 적용 도메인은 금융이고, 금융은 불확실성이 지배하는 세계다. 하락하던 주가가 갑자기 상승세로 반전될 수도 있고, 세계 어디에선가 테러가 발생해 일시에 전 세계 주식시장이 폭락할 수도 있다. 금융의 이러한 특성은 시나리오를 만들어서 테스트하기 어렵고, 설사 높은 수익성으로 테스트를 통과한 모델과 알고리즘 트레이딩 시스템이라 하더라도 실제 주식시장에서 수익을 안겨줄지 아니면 재앙이 될지는 개발 시스템을 실제 적용해보기 전까지는 아무도 모르는 일이다. 그래서 알고리즘 트레이딩은 알파 모델의 개발도 개발이지만, 개발된 알파 모델과 시스템을 평가하는 것이 더욱더 중요하고도 어렵다고 할 수 있다.

이 장에서는 알고리즘 트레이딩 시스템의 성능 측정에 많이 사용하는 대표적인 방법들과 머신러닝 모델의 성능 측정방법에 대해 알아본다. 또한, 알고리즘 트레이딩 시스템에 적용할 알파 모델을 선택할 때 유의사항 등을 설명하며, 마지막으로 알고리즘 트레이딩 시스템의 수익률을 좌우하는 파라미터 최적화Optimization에 대해 다룬다.

6.1 알고리즘 트레이딩 시스템의 성능 측정

검증된 수학 이론을 바탕으로 만든 강한 알파 모델도, 뛰어난 학습결과를 보여주는 알파 모델도, 적절한 성능 측정없이는 결코 알고리즘 트레이딩에 사용할 수 없다. 이것은 매우 당연하지만, 현실적으로 생각해보면 꽤 어려운, 정답이 없는 난제다. 과연 어떻게 성능을 측정하는 것이 알고리즘 트레이딩 시스템의 성능을 '적절히' 측정하는 것인지는 아무도 명확하게 이야기할 수 없다. 이유는 단연코 금융시장의 '불확실성' 때문이다.

알파 모델을 만들고 이를 적용한 알고리즘 트레이딩 시스템은 기본으로 우리가 이미 알고 있는 데이터, 즉 과거 데이터를 기반으로 개발한다. 그리고 이렇게 개발한 알고리즘 트레이딩 시스템의 성능을 평가하기 위해서는 또다시 과거의 데이터를 사용할 수밖에 없다. 과거 데이터만을 사용해 평가한 결과가 좋다면, 테스트한 알고리즘 트레이딩 시스템은 미래에 높은 수익을 가져온다고 할 수 있는 것인가? 반대로 평가 결과가 좋지 않다면, 미래에 손실을 초래할 것이라고 할 수 있는가?

경험에 의하면, 과거의 테스트 결과가 좋다고 해서 미래에 높은 수익을 가져오지 않는 경우가 종종 있고, 반대로 테스트 결과가 좋지 않았는데 현재 시점에서는 뜻밖에 좋은 수익을 보여주는 경우도 적지 않았다. 즉, 과거 데이터를 이용한 백테스팅 결과가 연관이 없다고는 할 수 없으나, 그 결과에 전적으로 의지해서 개발한 알고리즘 트레이딩 시스템을 이용해 돈을 투입하고 실제 거래에 사용하기에는 미심쩍다.

알고리즘 트레이딩 시스템의 성능 평가는 크게 3가지 목적을 가지고 실시한다.

구현 시스템의 수익성 평가

첫 번째 목적인 수익성 평가는 가장 기본적인 것으로, 어느 정도의 수익률을 예상할 수 있는지 조사한다.

구현 모델별 비교

두 번째는 시스템에 적용된 모델별 수익성 비교와 특성을 파악하기 위함이다. 특성이란 해당 모델의 매수·매도 포지션이 어떻게 변화하는지, 포지션 변화의 수는 얼마나 되는지 등을 말한다. 모델별 특성을 아는 것은 복수 개의 알파 모델을 적용할 때 중요한 문제다. 알파 모델을 하나만 사용할 때와 복수 개를 사용할 때, 어느 쪽이 더 높은 수익률을 보여줄지는 깊이 생각해 볼 문제다. 그야말로 엄청난 수익률을 보여주는 알파 모델을 찾았다고 하면 그 모델 하나로 즐거운 수익을 기대할 수 있겠으나, 현실에서는 만나기 어려운 행운이라고 할 수 있다. 그리고 설사 행운의 알파 모델을 찾았다 하더라도, 그 알파 모델이 얼마 동안이나 수익을 만들어 줄지는 모르는 일이다.

그래서 위험분산 차원에서 복수 개의 알파 모델을 많이 사용한다. 알파 모델마다 저마다의 이론적 배경이 있어서 서로 다른 특성을 보여주므로 한 모델의 수익률이 높지 않다면, 다른 모델이 수익을 발생해 손실 위험을 줄일 수 있다. 이렇게 여러 개의 알파 모델을 선택한다면 되도록 서로 다른 특성을 가진 알파 모델을 사용하는 것이 당연하다. 위험을 분산하기 위해 여러 개의 알파 모델을 사용하는데, 비슷한 특성을 가진 알파 모델들을 선택한다면 의미가 없기 때문이다.

시스템에 대한 자신감

세 번째 시스템에 대한 자신감은 심리적 요인이다. 그리고 알고리즘 트레이딩 시스템 성공에 의외로 중요한, 아니 어찌 보면 가장 중요한 요인이다. 알고리즘 트레이딩 시스템의 사용 여부는 결국 사람이 결정한다. 아무리 좋은 시스템이라 하더라도 사람이 사용하지 않으면 아무런 수익도 손실도 발생하지 않는다. 개발한 알고리즘 트레이딩 시스템은 수익을 내는 데 오랜 시간이 걸릴 수도 있고, 수익과 손실의 등락폭이 클 수도 있고, 오랫동안 계속 손실을 만들 수도 있다. 특별한 행운이 없다면 어떠한 알고리즘 트레이딩 시스템도 시작 첫날부터 만족할 만한 수익을 손실 없이 만들어주지는 못한다.

수익이 계속 떨어지고 잔액이 점점 줄어드는 모습을 보며, 알고리즘 트레이딩 시스템에 대한 믿음을 감내하는 것은 쉬운 일은 아니다. 이때 중요한 것이 개발자, 투자자의 알고리즘 트레이딩 시스템에 대한 믿음이다. 지금 적용한 모델은 특성상 일정 시간이 지나야 수익을 낸다는 것을 알고 있거나, 수익과 손실이 지속적으로 번갈아 가면서 발생하지만 결과적으로는 수익을 낼 수 있다는 것을 알고 있어야 가혹한 겨울을 견디고 따뜻한 봄을 맞이할 수 있다.

예를 들어, 추세추종 모델은 많은 거래에서 손실을 기록하지만 몇 번의 거래로 그동안의 손실을 만회할 만큼의 큰 수익을 내는 특성이 있다. 그런데 몇 번의 거래로 손해를 보고 해당 모델을 더는 사용하지 않는다면 추세추종 모델이 앞으로 가져오리라 예상되는 큰 수익을 스스로 외면하는 것과 같다(물론 예상했던 큰 수익이 발생하지 않을 수도 있다).

개발한 시스템에 대한 자신감이 없다면 약간의 손실만으로도 알고리즘 트레이딩 시스템을 중지시키고자 하는 유혹을 떨쳐버리기 힘들기 때문에 매우 중요한 평가 요소라 할 수 있다.

6.2 백테스팅

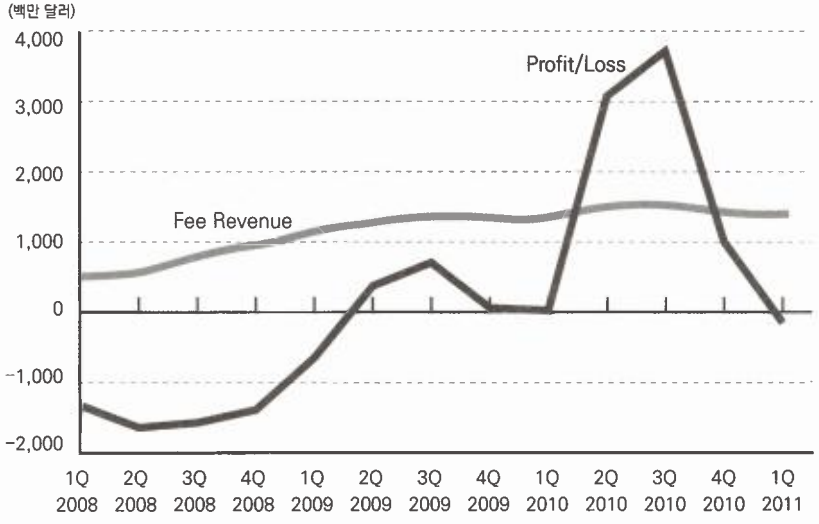
특정 기간의 과거 데이터로 알고리즘 트레이딩 시스템의 성능을 평가하는 것을 백테스팅Backtesting이라고 한다. 백테스팅은 알고리즘 트레이딩 시스템이 어느 정도의 예측력을 가졌는지, 수익률은 얼마나 되는지, 시스템 성능의 특성은 어떤 것인지를 파악하기 위해 시행하는 것으로 그 중요성을 아무리 강조해도 지나침이 없다.

백테스팅은 다양한 종류와 방법이 있고, 기관이나 헤지펀드에 따라 기존 것을 변형해 사용하는 경우도 많다. 대표적으로 많이 사용하는 Profit/Loss(손익, 수익률), Hit Ratio(적중률, 명중률), Drawdown(고점 대비 최대 손실률), Sharpe Ratio(샤프 지수) 4가지에 대해 알아보자.

6.2.1 Profit/Loss 테스트

알고리즘 트레이딩 시스템의 평가 방법 중 가장 많이 사용하는 것은 단연코 Profit/Loss 테스트다. 개발한 시스템에 특정 기간의 데이터를 입력해 트레이딩을 실시했을 때 발생하는 이익과 손실을 평가하는 것으로 간단히 이야기해 수익률을 의미한다. Profit/Loss는 2가지 방법으로 많이 사용하는데, 테스트 기간의 전체 수익률을 계산하는 방법과 일정 기간별로 수익률을 계산하는 방법이다. 다음 그래프는 분기별 수익률을 표현한 것이다.

그림 6-1 Profit/Loss 그래프[01]



6.2.2 Hit Ratio

Hit Ratio는 모델을 이용해 예측한 결과가 얼마나 맞는지 알려주는 정확도다. Hit Ratio는 모델에 따라 의미가 약간 다른데, 평균회귀 모델의 경우 매수·매도 포지션을 결정하는 데 이 포지션이 맞았는지를, 머신러닝 모델은 주가 방향의 예측 정확도를 의미한다.

01 출처: https://goo.gl/l5hPQz

예를 들어, 앞 장에서 구현한 머신러닝 모델은 주가 방향이 상승일지 하락일지를 예측하는데, Hit ratio=0.6이라는 것은 예측의 60%가 적중했다는 의미다. 평균회귀 모델에서 Hit ratio=0.5라는 것은 모델에 의해 결정된 포지션의 50%가 올바른 선택이었다는 것이다.

Hit Ratio는 과거 데이터를 이용하므로 특히 머신러닝 모델의 경우 학습에 사용했던 데이터와 테스트에 사용한 데이터가 중복되지 않도록 한다.

다음 getHitRatio( )는 일정 기간의 특정 종목 예측 정확도를 구하는 함수다.

```
def getHitRatio(self,name, code, start_date,end_date,lags_count=5):
    a_predictor = self.predictor.get(code,name)

    df_dataset = self.predictor.makeLaggedDataset(code,start_date,end_date,
self.config.get('input_column'), self.config.get('output_column'),lags_count )
    df_x_test = df_dataset[ [self.config.get('input_column')] ]
    df_y_true = df_dataset[ [self.config.get('output_column')] ].values

    df_y_pred,df_y_pred_probability = a_predictor.predict(df_x_test)

    hit_count = 0
    total_count = len(df_y_true)
    for index in range(total_count):
        if (df_y_pred[index] == df_y_true[index]):
            hit_count = hit_count + 1

    hit_ratio = hit_count/total_count
    print "hit_count=%s, total=%s, hit_ratio = %s" % (hit_count,total_
count,hit_ratio)

    return hit_ratio
```

실행결과

```
hit_count=8, total=14, hit_ratio = 0.571428571429
```

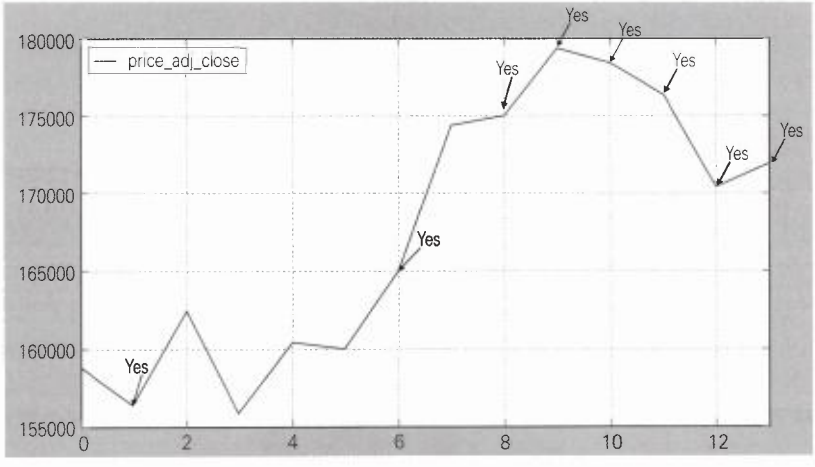
Hit Ratio는 모델별 예측 정확도를 나타내지만, 이는 참고용일 뿐 이 값을 기준으로 알고리즘 트레이딩 시스템의 성능을 평가하는 것은 섣부르다고 할 수 있다. 이

는 예측 정확도의 성격과 위험 관리에 대해 알 수 없기 때문이다. 실제 거래 데이터에서 언제 예측이 맞았고 틀리는가를 보는 것도 성능 평가에 많은 도움이 된다.

다음 drawHitRatio( ) 함수는 예측 결과에 따라 취한 매수·매도 포지션이 맞을 때는 'Yes'라는 표식을 나타내 주가 변동에 따른 알고리즘 트레이딩 시스템 예측 성능을 볼 수 있다.

```
def drawHitRatio(self,name, code, start_date,end_date,lags_count=5):
    a_predictor = self.predictor.get(code,name)

    df_dataset = self.predictor.makeLaggedDataset(code,start_date,end_date,
self.config.get('input_column'), self.config.get('output_column'),lags_count )
    df_x_test = df_dataset[ [self.config.get('input_column')] ]
    df_y_true = df_dataset[ [self.config.get('output_column')] ].values

    df_y_pred,df_y_pred_probability = a_predictor.predict(df_x_test)

    ax = df_dataset[ [self.config.get('input_column')] ].plot()

    for row_index in range(df_y_true.shape[0]):
        if (df_y_pred[row_index] == df_y_true[row_index]):
            ax.annotate('Yes', xy=(row_index, df_dataset.loc[ row_index, self.
config.get('input_column') ]), xytext=(10,30), textcoords='offset points', arro
wprops=dict(arrowstyle='-|>'))

    plt.show()
```

그림 6-2 실행결과 - Hit Ratio 도표

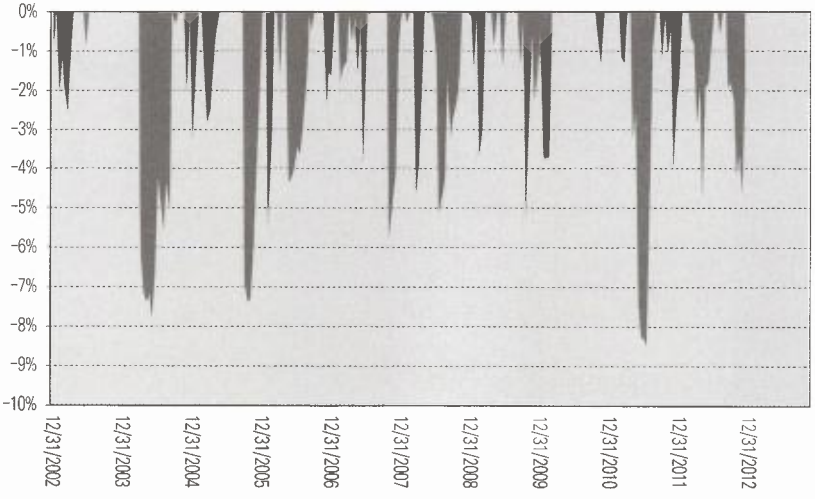


6.2.3 Drawdown

Hit Ratio가 예측 정확도를 나타내 시스템을 이용한 수익률을 짐작할 수 있다면, Drawdown은 알고리즘 트레이딩 시스템의 운영과 위험 관리에 매우 중요한 지표다. Drawdown은 잘못된 예측이 얼마나 되며, 그 기간이 얼마인지를 알려주어서 파산의 위험성에 대한 소중한 정보를 제공한다.

예를 들어, 10번의 트레이딩을 했고, 수익률이 20%라고 가정해보자. 이 수치만 보면 좋은 시스템이라 평가할 수 있으나 앞의 9번의 거래에서 모두 -80%의 손실을 보고, 마지막 1번의 거래에서 100%라는 큰 이익을 거둬 최종으로 20%의 수익률을 거뒀다고 하면 좋은 시스템이라 할 수 없다. 이는 연속적인 9번의 거래 손실에서 이미 회복할 수 없을 정도로 손실을 봐서 큰 이익이 기대되는 마지막 10번째에는 거래할 만한 자금이 없거나, 이미 파산해서 펀드가 해산되었을 수도 있기 때문이다. 아무리 엄청난 수익률을 안겨주는 알고리즘 트레이딩 시스템이라 하더라도 이 시스템을 운영하기 위한 자금이 필요하므로 Drawdown은 생존과 직결되는 매우 중요한 지표이다. 다음 그래프는 손실 관점에서 Drawdown으로 평가한 것으로, Y축은 손실률을 나타낸다.

그림 6-3 Drawdown 그래프



Drawdown 그래프에서 주의해서 봐야하는 사항은 3가지다.

Maximum Drawdown

최대 손실률로, 알고리즘 트레이딩 시스템을 운영하는 기관이나 헤지펀드에서 성능을 평가할 때 자주 이용되는 지표다. Maximum Drawdown 지표가 중요한 이유는 알고리즘 트레이딩의 특성상 단 한 번의 잘못된 거래로 파산할 수도 있기 때문이다. 실제로 이런 사례도 심심치 않게 발생하므로 항시 주의를 기울여야 한다.

Drawdown Duration

손실을 지속하는 기간을 의미하는 것으로, 주로 일 단위로 많이 이야기한다. 예를 들어, 일 단위의 Drawdown Duration이 5라면 5일 동안 지속적인 손실을 보았다는 의미다. Drawdown Duration 역시 중요한 지표로, 너무 오랫동안 손실을 지속하는 시스템은 자금적으로나 심리적으로나 견디기 어렵다. 적은 수익이라도 꾸준히 발생하게 하는 것이 좋은 시스템이라는 것은 누구나 동의할 것이다.

Drawdown Curve

손실의 경향을 알 수 있는 것으로, 시스템이 가지고 있는 손실 특성을 파악할 수 있다. 예를 들어, 손실이 지속해서 확대되는 경향이 있는지, 평균회귀 모델처럼 평균회귀 특성이 있는지 등을 알 수 있다.

Drawdown 지표는 알고리즘 트레이딩을 할 때 지속해서 확인해야 하는 것으로, 수익률과 더불어 알고리즘 트레이딩을 지속해야 할지 말지를 결정하는 데 중요한 역할을 한다.

6.2.4 Sharpe Ratio

위험을 고려한 성능을 나타내는 수치로, Sharpe Index, Sharpe Measure 혹은 Reward-to-variability Ratio로 불린다. Sharpe Ratio는 노벨상 수상자인 윌리엄 샤프William Sharpe가 1966년에 개발한 것으로, 금융에서 광범위하게 사

용되는 중요한 개념이다. Sharpe Ratio는 다음 식으로 계산할 수 있다. 이 식에서 R은 투자이익이고, R_b는 투자이익을 비교하기 위한 벤치마크 투자이익으로 위험 없는 은행이자율 등을 많이 사용한다.

$$\text{Sharpe Ratio} = \frac{E(R - R_b)}{\sqrt{Var(R - R_b)}}$$

수식을 간단히 풀어서 이야기하면 '기대값/기대값 표준편차'를 계산한 것이다. 기대값은 수익률이고, 기대값 표준편차는 수익률의 표준편차이므로 다르게 표현하면 '수익률/변동성'이라고 할 수 있다(필요에 따라 수익률 대신에 적중률과 같은 다른 개념을 사용할 수도 있다). 변동성은 위험을 의미하기 때문에 Sharpe Ratio 분석을 Reward/Risk 분석이라고도 부르는 이유가 이것이다.

모든 알고리즘 트레이딩 시스템은 수익률과 변동성이 있는데, 여러 개의 알고리즘 트레이딩 시스템 중에서 무엇이 좋은지 판단하기 어려울 때가 있다. 다음 표를 살펴보자.

표 6-1 시스템별 수익률과 변동성 비교

명칭	수익률	변동성
시스템 A	0.16	0.1
시스템 B	0.16	0.3
시스템 C	0.09	0.7
시스템 D	0.85	0.6

수익률을 놓고 보면 시스템 D가 0.85, 즉 85%로 가장 높고, 시스템 C는 0.09로 가장 낮다. 시스템 A와 시스템 B는 동일한 수익률을 보여주므로 어느 것이 더 좋다고 이야기 힘들다. 시스템 D의 수익률과 변동성을 살펴보면 시스템 D를 알고리즘 트레이딩에 사용하는 것이 좋을지 아닐지를 판단하기 힘들다. 수익률은 0.85로 높지만, 변동성이 0.6으로 시스템 A와 B보다 크기 때문이다. 변동성이 크면 그만큼 위험이 커지기 때문에 단 한 번의 잘못된 거래로 파산할 수 있다.

이럴 때 Sharpe Ratio를 이용하면 각 위험을 고려한 시스템 간의 성능을 직접 판단할 수 있다. [표 6-1]의 Sharpe Ratio를 수익률/변동성으로 계산해보면 다음과 같다.

표 6-2 [표 6-1]의 시스템별 Sharpe Ratio

명칭	수익률	변동성	Sharpe Ratio
시스템 A	0.16	0.1	0.16/0.1 = 1.6
시스템 B	0.16	0.3	0.16/0.3 = 0.53
시스템 C	0.09	0.7	0.09/0.7 = 0.12
시스템 D	0.85	0.6	0.85/0.6 = 1.41

Sharpe Ratio는 큰 값이 좋은 것으로, 시스템 A가 1.6으로 가장 높은 수치를, 시스템 C가 0.12로 가장 낮은 값을 보여준다. 시스템 A의 수익률은 0.16으로 시스템 D보다 낮지만, 변동성이 0.1로 시스템 D의 0.6보다 훨씬 낮다. 즉, 시스템 D보다 수익률은 낮지만, 안정성이 높아서 위험에 빠질 가능성이 적다는 것을 의미한다. 시스템 D의 Sharpe Ratio는 1.41로 변동성 0.6을 고려했을 때 높은 값이라 할 수 있는데, 수익률이 0.85로 매우 높으므로 위험을 고려하더라도 성능이 좋은 시스템이라는 것을 알 수 있다.

금융에서 변동성은 가장 무서운 적이라고 할 수 있다. 앞서 설명했지만, 알고리즘 트레이딩 시스템은 단 한 번의 거래로 파산할 수 있어서 수익률보다는 수익률이 적더라도 위험도가 낮은 시스템을 선택하는 것이 합리적인 선택이라 할 수 있다. Sharpe Ratio는 이익과 위험을 고려한 수치여서 시스템의 성능을 균형 잡힌 시각에서 알 수 있게 해준다.

6.3 머신러닝 모델 성능 측정

머신러닝 모델의 성능을 측정하는 방법은 문제의 성격에 따라 달라진다. 수치를 예측하는 회귀는 제곱근 평균 제곱오차RMSE, Root Mean Square Error 같이 예측치와 실측

치의 차이를 계산하는 방법을 사용하고, 알고리즘 트레이딩 시스템에 사용하는 머신러닝 모델과 같은 분류는 콘퓨전 매트릭스Confusion Matrix와 같이 분류하는 항목별 정확도나 재현율과 같은 특성을 계산하는 방법을 많이 사용한다.

Hit Ratio, Risk/Reward 테스트 등을 이용하면 수익률에 관련된 성능을 측정할 수 있지만, 머신러닝 모델 자체에 대한 것은 아니어서 이와는 별도로 머신러닝 모델의 성능을 측정할 필요가 있다. 머신러닝 모델은 주가의 방향을 예측하는 분류자인데, 상승과 하락 중 어느 쪽을 잘 맞추는지 정확도와 재현율은 얼마나 되는지를 알 수 있다면 모델이나 전략을 수정해 더 높은 수익률을 기대할 수 있기 때문이다.

머신러닝 모델 구현에 사용했던 sklearn 라이브러리는 분류자의 경우, 2가지 방법으로 예측할 수 있다. 첫 번째는 예측값이고, 두 번째는 예측에 대한 확률이다. 예를 들어, 주가가 상승하는 것을 '1', 하락하는 것을 '0'으로 학습시켰다고 하면 sklearn의 `predict()` 함수는 1 또는 0의 예측값을 반환하고, `predict_proba()` 함수는 '0.43, 0.57'처럼 각각의 예측에 대한 확률값을 제공한다. 확률값은 예측에 대한 머신러닝 알고리즘의 신뢰 수준Confidence Level을 알 수 있어서 단순히 예측값을 아는 것보다 유연하게 예측 결과값을 활용할 수 있다. 예를 들어, 1이 될 확률값이 0.57이라도 학습된 머신러닝 모델이 0을 더 잘 맞춘다면 예측값을 0으로 해석해 사용할 수 있다.

또한, 앞서 개발한 모델들은 각 머신러닝 알고리즘에 별도의 파라미터를 지정하지 않고 기본값을 사용했는데, 더 좋은 성능을 내게 하려면 적절한 파라미터를 설정해줘야 하고, 그러기 위해서도 구체적인 정보가 필요하다.

6.3.1 혼동 행렬

혼동 행렬Confusion Matrix은 항목별 분류 결과를 테이블 형태로 표현한 것으로, 분할표Contingency Table라고도 한다. 예를 들어, 종목코드가 006650인 대한유화를

2015년 1월 1부터 2015년 10월 30일까지 학습시킨 SVM 예측변수에 2015년 11월 1일부터 2015년 11월 19일까지의 예측을 혼동 행렬로 나타내면 다음과 같다.

표 6-3 혼동 행렬 예시

구분		Predicted	
		하락	상승
Actual	하락	2	6
	상승	1	5

'Predicted'는 예측치를 의미하고, 'Actual'은 실제값을 나타낸다. 예를 들어, Predicted 아래에 있는 '하락' 행은 예측변수가 하락이라고 예측한 수로 2 + 1 = 3건이고, 상승으로 예측한 경우는 6 + 5 = 11건이다. 따라서 예측변수의 총 예측 건수는 3 + 11 = 14건임을 알 수 있다.

'Actual'은 실제 상승수와 하락수를 표시한 것으로, 실제 하락수는 2 + 6 = 8건, 상승수는 1 + 5 = 6건, 따라서 총 건수는 8 + 6 = 14건으로, 예측변수의 예측 건수와 일치한다.

혼동 행렬에서 유심히 봐야 하는 부분은 값 '2'와 '5'다. '2'는 Predicted의 하락 행과 Actual의 하락 열의 교차 지점에 있는 값으로, 예측변수에서 하락이라고 예측한 값 중에 2건이 올바른 예측이었다는 것을 알려준다. 마찬가지로 '5'는 Predicted의 상승 행, Actual의 상승 열의 교차 지점에 있는 값으로, 상승이라고 예측한 값 중 5건이 올바른 예측이었음을 의미한다.

이때 '2'가 표시된 곳을 'True Negative', '5'가 있는 곳을 'True Positive'라고 한다. 그리고 '1'과 '6'은 잘못된 예측건수를 나타내므로 '1'은 잘못된 하락, 즉 'False Negative', '6'은 잘못된 상승, 즉 'False Positive'라고 한다.

혼동 행렬은 예측 정확도만으로 알 수 없는 머신러닝 모델의 예측 특성과 데이터

편향Bias 등의 추가적인 정보를 알 수 있어 매우 유용한 도구다. 혼동 행렬을 이용해 대한유화를 예측하는 SVM 예측변수의 특성을 살펴보자.

SVM 예측변수는 주가의 방향을 예측하는 이진 분류자Binary Classifier로, 예측 결과는 상승(1), 하락(0)의 2가지 값을 가진다. 전체적인 예측 정확도만을 보면 총 14건 중 7건을 올바르게 예측했으므로 7/14=0.5가 되어 2번 중 1번은 맞춘다고 생각할 수 있다.

먼저 데이터 편향이 있는지 확인해보자. 데이터 편향은 테스트에 사용된 데이터가 한쪽으로 치우쳐 예측결과를 제대로 평가할 수 없는 것을 말한다. 예를 들어, 테스트에 사용한 데이터의 개수가 100건인데 그중 90건의 데이터가 상승이고 10건의 데이터가 하락이라면, 머신러닝 모델이 상승이라고 많이 예측할수록 예측 정확도가 상승하게 된다. 이는 90%의 데이터가 상승이기 때문에 SVM 예측변수의 정확도는 상승이 많을수록 좋아질 수밖에 없다.

따라서 혼동 행렬을 이용해 사용된 데이터에 데이터 편향이 있는지 없는지를 확인하는 것이 순서다. [표 6-3]의 혼동 행렬에서 Actual의 하락은 2+6=8, 상승은 1+5=6으로 전체 데이터의 하락과 상승 비율은 57:43으로 적절한 수준이라고 할 수 있다.

이제는 모델의 예측 특성을 알아보자. 하락 예측에 대한 정확도는 'True Negative/(True Negative+False Negative)', 상승에 대한 정확도는 'True Positive/(True Positive+False Positive)'로 구할 수 있다.

이에 따라 [표 6-3] 혼동 행렬의 하락 예측 정확도는 $2/(2+1)=0.66$, 상승 예측 정확도는 $5/(5+6)=0.45$고, SVM 예측변수는 상승보다 하락에 대한 예측력이 높음을 알 수 있다. 따라서 SVM 예측변수를 사용할 때 하락 예측값이 더 정확하므로 포지션을 정할 때 '하락'에 기준을 맞추어 결정하는 것이 더 높은 수익률을 줄 것이다.

한 가지 더 살펴볼 사항은 SVM 예측변수의 상승과 하락에 대한 예측 건수다. 총 14건 중 상승은 11건, 하락은 3건으로, 상승에 대한 예측 비중이 하락보다 3배 이상 높다. SVM 예측변수는 상승에 대한 예측을 더 많이 하므로 매도보다는 매수 포지션을 더 많이 취할 것으로 예상할 수 있는데, 상승의 정확도가 0.45라서 사지 않아도 될 주식을 사게 되어 손실을 볼 가능성이 클 수 있다.

앞의 결과를 종합해서 학습에 사용된 데이터가 편향된 것은 아닌지, 파라미터 설정이 부적절한지, 적당한 크기의 데이터를 학습에 사용했는지를 판단해 볼 필요가 있다.

다음 그림은 혼동 행렬에 관련된 여러 가지 정의와 수식들을 한 번에 보여준다.

그림 6-4 혼동 행렬[02]

<table>
<tr><td></td><td></td><td colspan="2">True condition</td><td colspan="2"></td></tr>
<tr><td></td><td>Total population</td><td>Condition positive</td><td>Condition negative</td><td>Prevalence
$= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$</td><td></td></tr>
<tr><td rowspan="2">Predicted condition</td><td>Predicted condition positive</td><td>True positive</td><td>False positive
(Type I error)</td><td>Positive predictive value (PPV), Precision
$= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$</td><td>False discovery rate (FDR)
$= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$</td></tr>
<tr><td>Predicted condition negative</td><td>False negative
(Type II error)</td><td>True negative</td><td>False omission rate (FOR)
$= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$</td><td>Negative predictive value (NPV)
$= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$</td></tr>
<tr><td rowspan="2">Accuracy (ACC) =
$\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$</td><td>True positive rate (TPR), Sensitivity, Recall =
$\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$</td><td>False positive rate (FPR), Fall-out =
$\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$</td><td>Positive likelihood ratio (LR+)
$= \frac{TPR}{FPR}$</td><td rowspan="2">Diagnostic odds ratio (DOR)
$= \frac{LR+}{LR-}$</td></tr>
<tr><td>False negative rate (FNR), Miss rate =
$\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$</td><td>True negative rate (TNR), Specificity (SPC) =
$\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$</td><td>Negative likelihood ratio (LR-)
$= \frac{FNR}{TNR}$</td></tr>
</table>

6.3.2 Classification Report

Classification Report는 sklearn에서 제공하는 기능으로, 학습시킨 분류자들의 성능을 알기 쉽게 요약해준다. 혼동 행렬과 비슷한 것도 있지만, Recall, F1-score 등의 값을 제공해준다. 예를 들어, 앞의 혼동 행렬에서 언급한 대한유화 SVM 예측변수의 성능을 파악하기 위해 sklearn의 `classification`

02 참고: https://en.wikipedia.org/wiki/Confusion_matrix

_report( ) 함수를 실행하면 다음과 같은 결과를 보여준다.

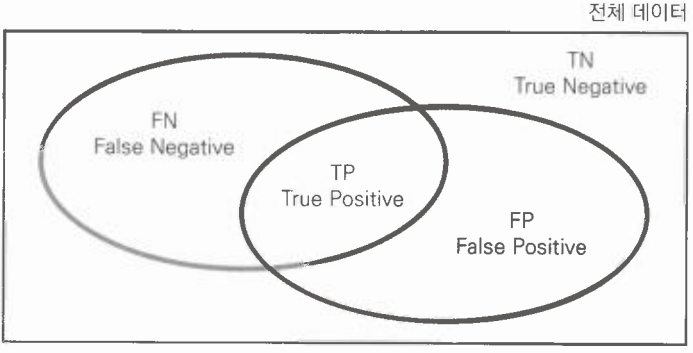
```
              precision  recall    f1-score   support
Down          0.60       0.38      0.46       8
Up            0.44       0.67      0.53       6
avg / total   0.53       0.50      0.49       14
```

결과에서 가장 왼쪽 행의 Down은 주가의 하락, Up은 주가의 상승을 의미한다. Precision은 정확도를 나타내는 것으로, True Negative와 True Positive가 있다. 예를 들어, Down의 Precision이 0.6인 것은 True Negative의 값이 0.6임을 의미하며, 'Precision = True Negative/(True Negative+False Negative)'로 계산한 값이다.

Recall은 민감도라고도 하는데, 무작위로 데이터를 입력해 예측했을 때 Positive 값으로 결과가 정확하게 나올 확률이다. recall의 개념은 학습된 분류자가 Positive와 Negative 둘 중 어디에 더 높은 예측력이 있는지를 알려주는 것으로, 이해를 확실히 할 필요가 있다.

다음 그림은 분류의 결과를 혼동 행렬에서 사용한 개념인 True Positive, True Negative, False Positive, False Negative의 4가지로 나누었을 때 각 영역을 벤다이어그램으로 표시한 것이다.

그림 6-5 Recall의 개념



이 그림에서 검은색 원은 분류자가 Positive로 예측한 영역이고, 회색 원은 Negative로 예측한 영역이다. Recall은 Positive를 정확하게 예측할 확률로 다음과 같이 계산한다.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Specificity(특이도)는 Recall과 반대 개념으로 Negative를 정확하게 예측할 확률이며, 수식도 정반대이다.

$$\text{특이도} = \frac{TN}{TN + FP}$$

Classification Report의 결과를 보면 주가 하락을 의미하는 Down의 Recall은 0.38, 상승인 Up은 0.67이다. 따라서 분류자는 하락보다 상승에 더 높은 정확도를 가지고 있다는 것을 알 수 있다.

F1-score는 Precision과 Recall의 조화평균값으로 다음과 같이 계산한다.

$$\text{F1 score} = \frac{2TP}{2TP + FP + FN}$$

F1-score가 의미하는 것은 Precision과 Recall의 수치를 고려했을 때 상승과 하락 중 어느 것이 예측력이 더 높은지를 알려주는 것으로, Down은 0.46, Up은 0.53으로 Up이 0.07 더 높은 예측력을 가짐을 알 수 있다.

마지막으로 Classification Report의 Support(지지도)는 각각의 예측값에 대한 수를 나타내는 것으로, Down의 Support 값이 8이라는 것은 총 8건의 값을 Down, 즉 하락으로 예측했다는 것이다.

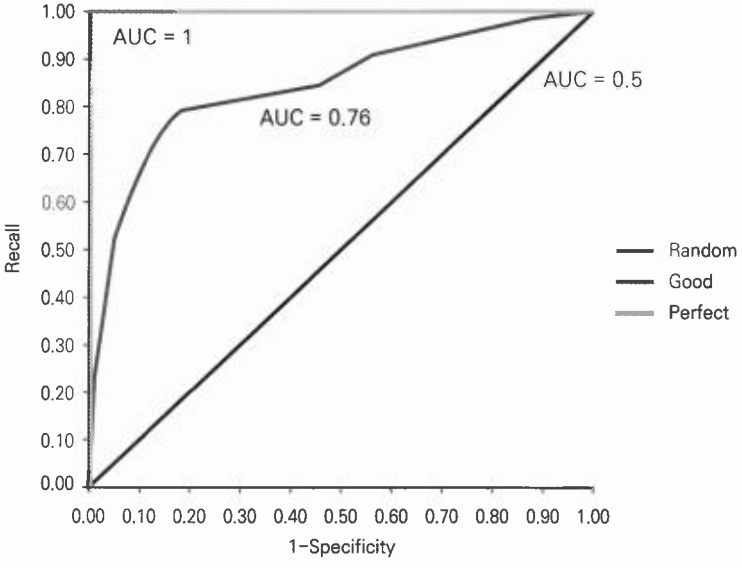
6.3.3 ROC 곡선

ROC 곡선Receiver Operating Characteristic Curve은 혼동 행렬, Classification Report

와 더불어 머신러닝 모델의 성능을 평가할 수 있는 또 하나의 유용한 도구다. ROC 곡선은 시각적으로 성능을 보여주어 직관적으로 이해하기 쉽고, 성능 평가에 사용한 데이터가 편향되었더라도 적용할 수 있다.

ROC 곡선은 Y축에 Recall, X축에 Specificity를 나타낸 그래프인데, 다음 그림을 한번 보자.

그림 6-6 ROC 곡선



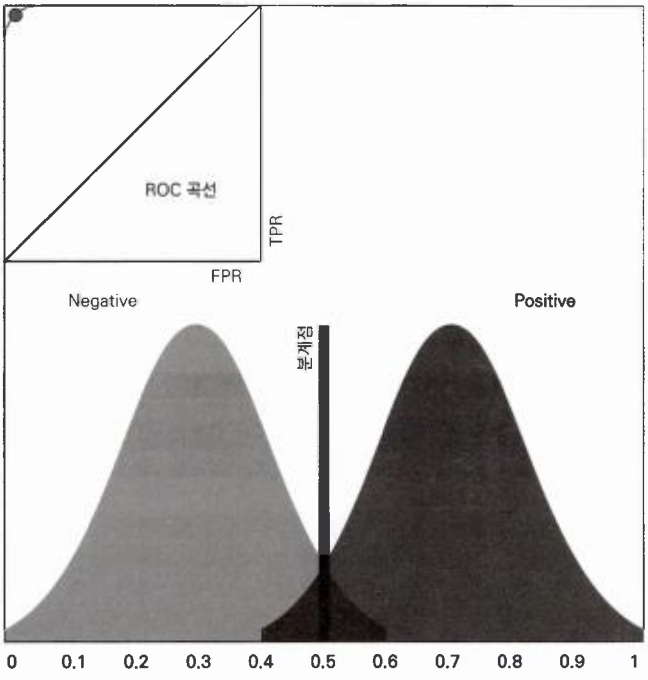
이 그림에서 세 선이 ROC 곡선이다. 가장 아래쪽 선은 테스트에 적용한 분류자의 예측 성능이 무작위(Random)로 선택한 것과 별 차이가 없을 때의 ROC 곡선이고, 가운데 곡선은 비교적 우수한 성능(Good)을 가진 분류자의 ROC 곡선, 그리고 가장 위쪽 선은 완벽한 성능(Perfect)을 가진 분류자의 ROC 곡선을 보여준다.

AUC[Area Under Curve]는 ROC 곡선의 면적을 의미하는 것으로, AUC의 값이 크면 클수록 우수한 성능을 가진다. 예를 들어, 무작위로 선택한 경우의 AUC=0.5고, 비교적 우수한 성능의 AUC=0.76, 완벽한 성능인 가장 위쪽 선의 AUC=1이 된다.

ROC 곡선은 학습시킨 분류자의 예측 성능에 대한 전체 궤적을 파악할 수 있으므로 핵심 개념이 무엇인지 반드시 이해해야 한다.

분류자를 이용해 예측하는 2가지 방법이 있는데, 첫 번째는 값 자체를 예측하는 것이고, 두 번째는 예측값에 대한 확률을 구하는 것이다. [그림 6-7]과 [그림 6-8]은 두 번째 방법인 확률을 이용해 주가의 상승과 하락, 2가지 값에 대한 예측 성능을 ROC 곡선으로 설명한 예로, [그림 6-7]은 우수한 성능을 보여주는 분류자의 ROC 곡선을 나타낸 예다.

그림 6-7 ROC 곡선 표본 - Good



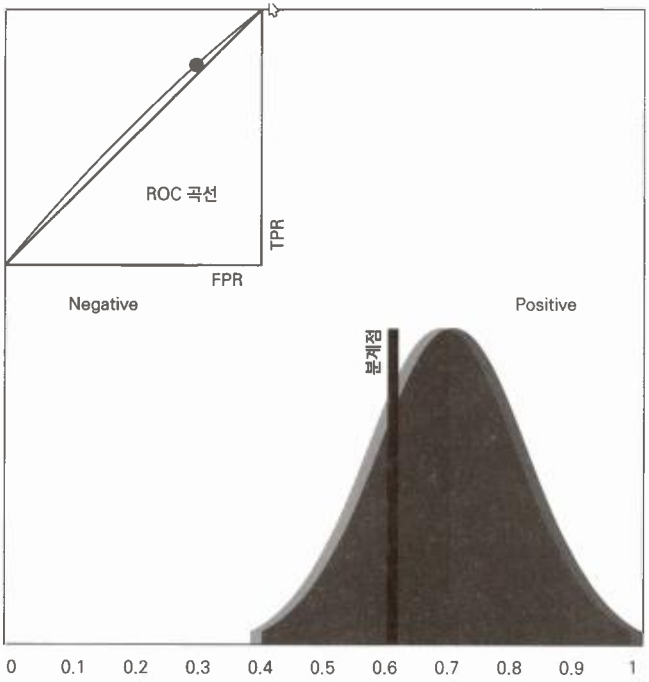
이 그래프에서 Negative를 나타내는 연한 회색 영역은 하락에 대한 확률값 분포로, 0~0.6 사이에 분포되어 있다. 진한 회색 영역은 Positive로, 상승에 대한 확률분포이며 범위는 0.4~1.0이다.

확률값을 이용해 상승과 하락을 결정한다면 상승과 하락을 결정짓는 기준값인 분계점Threshold를 구해야 한다. 예를 들어, 분계점이 0.4라고 하면 확률이 0~0.4 미만의 값들은 하락으로, 0.4~1.0 사이의 값은 상승으로 결정한다.

[그림 6-7]에서 두 곡선의 겹치는 영역이 적은데, 이것은 예측 성능이 우수하다는 것을 보여준다. 겹친 영역은 오류가 발생할 수 있는 부분인데, 그 영역이 작다는 것은 그만큼 오류 확률이 낮다는 것이기 때문이다. 그림 왼쪽 위의 ROC 곡선을 보아도 역시 우수하다는 것을 알 수 있다. 이런 경우에는 분계점 값을 0.5로 하면 비교적 적은 오류가 발생하고 우수한 예측 성능을 보이리라는 것을 쉽게 짐작할 수 있다. 또한, 상승과 하락에 대해서도 고르게 우수한 예측 성능을 보인다.

[그림 6-8]은 예측 성능이 좋지 않은 경우다.

그림 6-8 ROC 곡선 표본 - Bad



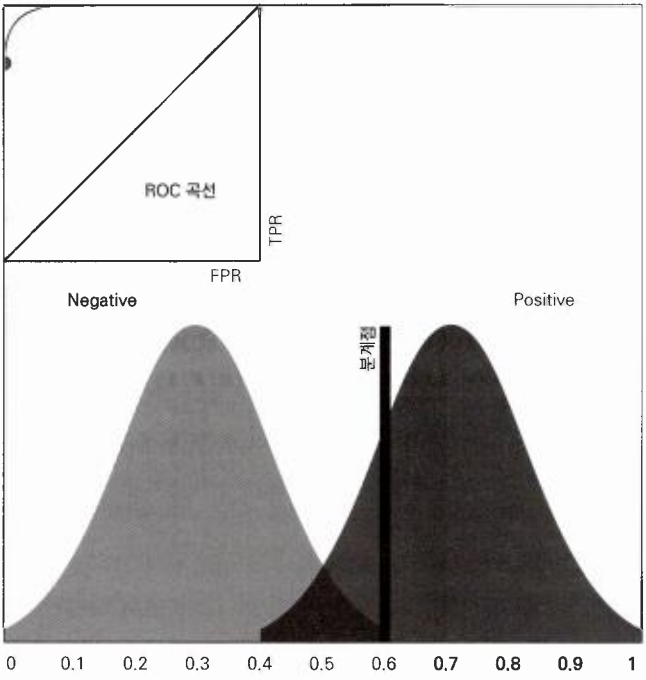
이 그래프는 두 곡선의 영역이 모두 0.4~1.0 사이에 위치하며, 상당 부분 겹쳐 있음을 확인할 수 있다. 겹친 영역은 오류가 발생할 수 있는 부분인데, 대부분 겹쳐 있으므로 예측 성능은 무작위로 예측하는 것과 별반 다르지 않을 것이고, 그림 왼쪽 위의 ROC 곡선 역시 무작위 ROC 곡선을 나타내는 선과 유사한 모습을 보여준다.

이처럼 ROC 곡선은 Recall에 대한 예측 성능이 Specificity에 따라 어떻게 달라지는지 전체 모습을 시각적으로 보여준다. Recall이 증가함에 따라 Specificity가 감소하는지 증가하는지, 변화가 있다면 그 변화의 정도가 완만한지 급격한지를 알 수 있다. 앞서 살펴본 혼동 행렬과 Classification Report는 Recall과 Specificity의 값이 무엇인지만을 알려줄 뿐 이들의 전체적인 궤적은 보여주지 않는다. ROC 곡선의 이러한 특성은 테스트한 분류자가 전 구간에 대해 안정적인 예측 성능을 가지는지 아니면 특정 영역에서만 좋은 예측 성능을 가지는지를 알 수 있게 한다.

아울러 ROC 곡선은 예측 성능을 높이는 데 적당한 분계점이 무엇인지를 결정하는 데도 사용할 수 있다. 예를 들어, 상승을 정확하게 예측하는 것이 하락을 예측하는 것보다 훨씬 더 중요하다면 분계점 값을 높거나 낮게 조정해 상승 예측 성능을 극대화할 수도 있다.

[그림 6-9]는 하락을 의미하는 확률분포(왼쪽 그래프)와 상승의 확률분포(오른쪽 그래프)를 나타낸다. 상승의 예측 성능을 높이기 위해, 일반적으로 많이 사용하는 0.5가 아닌 0.6을 사용했다고 가정해보자. 그래프에서 볼 수 있듯이 분계점이 0.6이 되면 연한 회색 영역과 겹치는 부분이 없어져서 상승 예측에 대한 정확도를 훨씬 더 높게 만들어주고, ROC 곡선 역시 완벽한 성능을 보여줄 때와 비슷하게 X, Y축 모두에 근접한 모습을 보여준다.

그림 6-9 ROC 곡선 – 보정

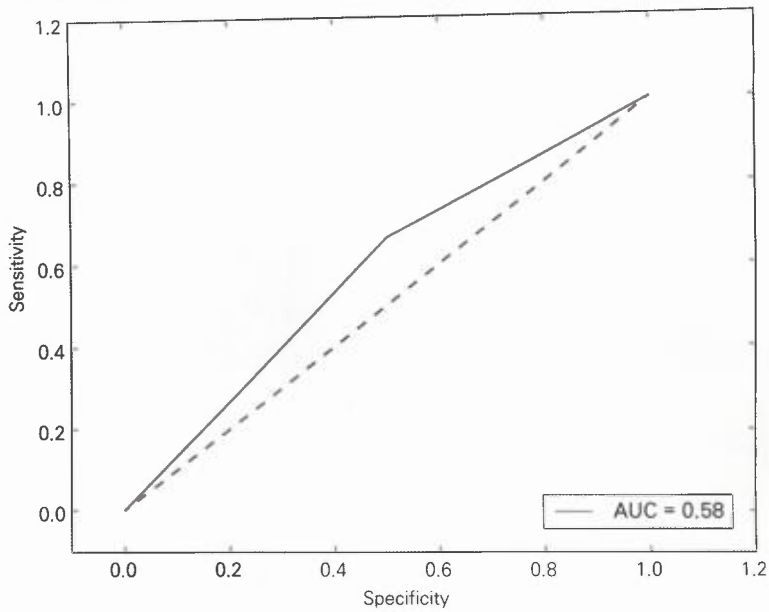


ROC 곡선을 이용해 혼동 행렬과 Classification Report를 설명할 때 사용한 대한유화 예측변수의 예측 성능을 파악해보자. ROC 곡선을 그리는 함수는 drawROC ( )로, 실제값과 예측값의 2개 인자를 가지고 있다.

```
def drawROC(self,y_true,y_pred):
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_true, y_
pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)

    plt.title('Receiver Operating Characteristic')
    plt.plot(false_positive_rate, true_positive_rate, 'b', label='AUC = %0.2f'%
roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0,1],[0,1],'r--')

    plt.xlim([-0.1,1.2])
```

```
plt.ylim([-0.1,1.2])
plt.ylabel('Sensitivity')
plt.xlabel('Specificity')
plt.show()
```

그림 6-10 실행결과 - ROC 곡선



그래프와 AUC = 0.58을 보면 SVM 예측변수의 성능이 좋지 않음을 알 수 있[r
따라서 분계점을 조정하거나 SVM 파라미터를 조정해 예측 성능을 향상할 [r
가 있다.

6.4 라이브 트레이딩 모니터링

라이브 트레이딩 모니터링Live Trading Monitoring은 실시간으로 알고리즘 트[r
템의 매수·매도를 모니터링하는 것으로, 시스템 운영에서 빠질 수 없
능이다. Pofit/Loss, Sharpe Ratio, ROC 곡선 등을 이용해 시스[r

성을 철저하게 파악해 선별한 시스템이라 하더라도 이 시스템이 수익을 안겨줄지 손실을 줄지는 아무도 장담할 수 없다. 이는 미래를 예측하는 일이기 때문이다.

그림 6-11 라이브 트레이딩 모니터링 예시[03]

금융시장이 표류하는 랜덤워크의 특성이 있지만, 최근 10~20년 동안 세계화와 IT 기술의 발달, 알고리즘 트레이딩의 확산 등으로 이러한 경향이 더욱 심해졌다. 이제는 마음만 먹으면 누구나 자신의 집에서 전 세계 금융 정보를 수집할 수 있고, 미국 주식시장, 홍콩 주식시장에 접속해 주식을 사고팔 수 있는 시대가 되었다. 마치 하나의 시장처럼 연결된 것이다. IT 기술의 발달로 전 세계 금융시장에 대한 접근을 선물로 얻었다면, 변동성의 심화는 재앙이 되었다. 우리나라가 아닌 다른 나라에서 발생한 사건 하나가 국내 주식시장의 폭락을 가져오기도 하고, 반대로 급등세를 만들기도 한다. 국내회사의 성장성을 예측하는 데 국내경제 상황보다는 해외경제 상황을 더 관심 있게 지켜봐야 하기도 한다.

이렇게 변동성이 심한 상황에서는 어떤 일이 발생할지 모르기 때문에 모든 거래를 알고리즘 트레이딩 시스템에 100% 맡기는 것은 큰 위험을 초래할 수 있다. 앞에서도 살펴봤지만, 알고리즘 트레이딩 시스템은 정해진 모델에 따라 사고팔기 때문

출처: https://goo.gl/oskwGf

에 주가에 영향을 미치는 정보를 알지 못하고, 예외 상황에 대해 올바르게 대처할 수 없다. 거듭 이야기하지만 예외적인 상황에서 단 한 번의 잘못된 거래는 모든 것을 파멸로 이끌 수 있다.

그래서 알고리즘 트레이딩 시스템에는 반드시 실시간으로 거래 상황을 모니터링하는 기능이 필요하다. 실시간으로 매수·매도 포지션을 점검하고, Drawdown 그래프를 보고 시스템이 이상 현상을 보이면 사람이 직접 대처하거나 알고리즘 트레이딩을 정지하도록 개발해야 한다.

6.5 파라미터 최적화

파라미터 최적화Parameter Optimization는 알파 모델에서 필요한 파라미터들을 가장 좋은 성능을 낼 수 있도록 최적화하는 것이다. 파라미터의 의미는 폭넓은데, 평균회귀 모델의 이동평균 기간을 5일 단위로 할지 10일 단위로 할지를 설정하는 것과 같은 모델과 직접 연관된 것도 있고, 머신러닝 모델처럼 학습에 사용할 데이터 시작일과 마감일을 1년으로 하면 좋을지 2년으로 하면 좋을지를 결정하는 것도 포함한다.

파라미터 최적화는 알고리즘 트레이딩 시스템의 성능에 결정적인 영향을 미친다. 앞에서 소개한 평균회귀 모델은 이미 검증된 모델로, 많은 기관과 헤지펀드가 이용해왔고 지금도 애용하는 모델이다. 하지만 앞에서 살펴본 결과를 보면 생각보다는 수익률이 높지 않았다. 그 이유는 모델이 잘못되거나 데이터의 오류라기보다는 파라미터가 최적화되지 않았기 때문이다. 아무리 좋은 모델이라 하더라도 적절한 파라미터가 설정되지 않으면 결코 좋은 결과를 기대할 수 없다.

앞에서 사용한 대한유화의 머신러닝 모델에서 모든 것이 동일한 조건에 time_lags 값만을 1, 5, 10으로 조정했을 때 Hit Ratio를 살펴보면 다음 표와 같다.

표 6-4 time_lags 값에 따른 Hit Ratio 변화

time_lags	Hit Ratio
1	0.5
5	0.57
10	0.55

time_lags가 5일 때는 1일 때보다 무려 0.07, 즉 7% 높은 예측력을 보여준다. 이는 오차범위 5%보다 높은 수치이므로 신뢰할 만한 결과라고 할 수 있다. 7%의 예측력 향상은 가볍게 볼 사항이 아니다.

카지노를 생각해보자. 카지노 바카라 게임의 1회당 기대율은 딜러 48.94%, 플레이어는 48.76%로 0.18%밖에 차이 나지 않는다. 하지만 이 무시할 만한 차이가 시간이 지날수록 대수의 법칙이 작동하며 엄청난 결과를 만들어낸다. 따라서 단지 time_lags의 값을 1에서 5로 바꿨을 때, 7%의 예측력 향상을 가져왔다는 것은 유의미한 변화라고 할 수 있다.

동일한 모델이라도 파라미터에 따라 매우 다른 수익의 양상을 보여주고, 알고리즘 트레이딩 시스템의 성능 역시 이 파라미터를 어떻게 최적의 값으로 채워 넣느냐에 따라 알고리즘 트레이딩 시스템의 설계 및 수준에서 차이가 나게 된다.

최근의 알고리즘 트레이딩 시스템의 경향은 금융시장의 유동성 심화와 타 알고리즘 트레이딩 시스템과의 경쟁으로 인해 그때그때의 시장 상황에 맞는 빠른 대처가 중요해지고 있다. 빨리 시장변화에 대처하려면 복잡한 모델은 적합하지 않으므로 단순하고 이해하기 쉬운 모델의 사용이 주류를 이루고 있다. 또한, 이러한 모델들은 많이 알려지고 검증된 모델들이 많아 실제 알고리즘 트레이딩 시스템에 적지 않게 활용된다.

이처럼 유사한 모델을 많은 기관과 헤지펀드에서 사용함에도 수익을 내는 곳과 그렇지 않은 곳이 생기는 이유는 파라미터 최적화도 큰 몫을 차지한다. 수익성이 좋

은 알고리즘 트레이딩 시스템을 만들려면 파라미터 최적화에 상당히 많은 시간과 노력을 투자해야 한다.

6.6 하이퍼파라미터 최적화

머신러닝 모델 파라미터의 최적화를 하이퍼파라미터 최적화Hyperparameter Optimization라고 한다. 머신러닝에는 크게 2가지, 모델 파라미터 최적화와 하이퍼파라미터 최적화가 있다. 모델 파라미터는 머신러닝 모델에서 필요한 파라미터로, 머신러닝 알고리즘이 스스로 찾는 값이다. 예를 들어, 로지스틱 회귀 모델이라면 다음과 같이 표현할 수 있다.

$$W^t * X = Y$$

X는 입력변수, Y는 출력변수, W^t는 로지스틱 회귀에서 찾아야 하는 벡터값인데, 이때 W^t가 모델 파라미터다.

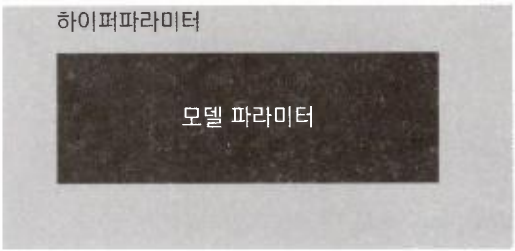
하이퍼파라미터는 머신러닝 모델을 실행할 때 설정하는 파라미터로, 모델 파라미터와 다르게 사람이 직접 값을 설정해줘야 한다. 예를 들어, sklearn의 로지스틱 회귀 모델에서 설정할 수 있는 값은 다음과 같다.[04]

- Penalty l1, l2
- Solver newton-cg, lbfgs, liblinear
- C float

하이퍼파라미터에서 설정한 값에 따라 모델 파라미터를 최적화하므로 하이퍼파라미터 최적화는 머신러닝 모델의 성능에 큰 영향을 미친다. 다음 그림은 하이퍼파라미터와 모델 파라미터와의 관계를 나타낸다.

04 로지스틱 회귀와 하이퍼파라미터에 대한 자세한 내용은 http://goo.gl/oCzs25 페이지를 참조하기 바란다.

그림 6-12 하이퍼파라미터와 모델 파라미터



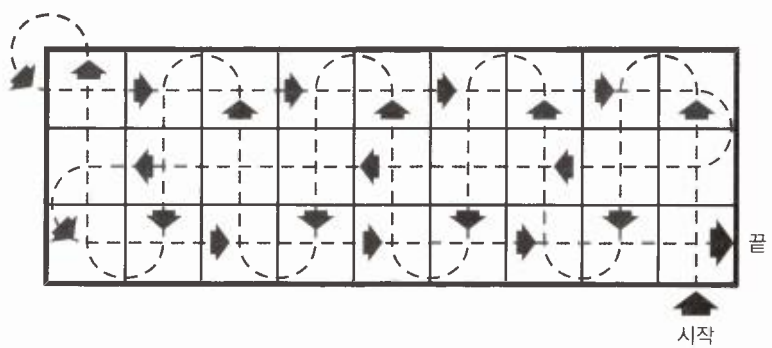
모델 파라미터 최적화는 하이퍼파라미터로 설정된 범위 내에서만 최적화를 시도하기 때문에 올바르게 설정되지 않은 하이퍼파라미터에서는 좋은 성능을 보여주기 힘들다. 따라서 모델 파라미터 최적화보다 중요한 것은 하이퍼파라미터 최적화라고 할 수 있다.

하이퍼파라미터 최적화의 대표적인 2가지 방법인 격자 탐색과 랜덤 탐색의 개념과 방법을 좀 더 살펴보자.

6.6.1 격자 탐색

격자 탐색Grid Search은 이름이 의미하는 것처럼 주어진 범위(격자) 내의 모든 값을 입력해 하이퍼파라미터를 최적화하는 방법으로, 다음 그림처럼 지나갈 수 있는 모든 길을 하나씩 지나가면서 가장 좋은 길이 어디인지를 찾는 방법이다.

그림 6-13 격자 탐색



예를 들어, 하이퍼파라미터 값으로 1,2,3,4,5 중 어떤 것이 좋은지 모르겠다면, 1,2,3,4,5 다섯 개의 값을 모두 입력해 그 결과를 보고 하이퍼파라미터를 최적화하는, 즉 선택하는 다소 무식한 방법이라 할 수 있다.

앞 장에서 학습시킨 랜덤 포레스트 예측변수의 최적화 하이퍼파라미터를 격자 탐색으로 찾아보자. optimizeHypermeter( ) 함수는 최적의 하이퍼파라미터 값을 찾기 위해 격자 탐색을 시행한다. 실질적인 격자 탐색을 실행하는 함수는 doGridSearch( )로, 가장 중요한 것은 굵게 표현된 grid_search.fit다. sklearn에서는 이와 같이 매우 간단하게 격자 탐색을 사용할 수 있다.

```
def optimizeHyperparameter(self,name, code, start_date,end_date,lags_count=5):
    a_predictor = self.predictor.get(code,name)

    df_dataset = self.predictor.makeLaggedDataset(code,start_date,end_date,
self.config.get('input_column'), self.config.get('output_column'),lags_count)

    X_train,X_test,Y_train,Y_test = self.predictor.splitDataset(df_
dataset,'price_date',[self.config.get('input_column')],self.config.get('output_
column'),split_ratio=0.8)

    param_grid = {"max_depth": [3, None],
                  "min_samples_split": [1, 3, 10],
                  "min_samples_leaf": [1, 3, 10],
                  "bootstrap": [True, False],
                  "criterion": ["gini", "entropy"]}

    a_predictor.doGridSearch(X_train.values,Y_train.values,param_grid)

def doGridSearch(self,x_train,y_train,param_grid):
    grid_search = GridSearchCV(self.classifier, param_grid=param_grid)
    grid_search.fit(x_train,y_train)

    for params, mean_score, scores in grid_search.grid_scores_:
        print("%0.3f (+/-%0.03f) for %r" % (mean_score, scores.std() * 2, params))
```

실행결과

```
0.500 (+/-0.070) for {'min_samples_split': 3, 'bootstrap': True, 'criterion':
'entropy', 'max_depth': 3, 'min_samples_leaf': 10}
```

```
0.500 (+/-0.070) for {'min_samples_split': 10, 'bootstrap': True, 'criterion':
'entropy', 'max_depth': 3, 'min_samples_leaf': 10}
0.505 (+/-0.077) for {'min_samples_split': 1, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 1}
0.505 (+/-0.077) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 1}
0.511 (+/-0.086) for {'min_samples_split': 10, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 1}
0.505 (+/-0.077) for {'min_samples_split': 1, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 3}
0.505 (+/-0.077) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 3}
0.511 (+/-0.086) for {'min_samples_split': 10, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 3}
0.511 (+/-0.086) for {'min_samples_split': 1, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 10}
0.511 (+/-0.086) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 10}
0.511 (+/-0.086) for {'min_samples_split': 10, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 10}
0.533 (+/-0.053) for {'min_samples_split': 1, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 1}
0.505 (+/-0.068) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 1}
0.543 (+/-0.032) for {'min_samples_split': 10, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 1}
0.533 (+/-0.001) for {'min_samples_split': 1, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 3}
0.533 (+/-0.001) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 3}
0.543 (+/-0.032) for {'min_samples_split': 10, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 3}
0.495 (+/-0.066) for {'min_samples_split': 1, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 10}
0.495 (+/-0.066) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 10}
0.495 (+/-0.066) for {'min_samples_split': 10, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf':
```

앞의 실행결과를 보면 한 줄이 시험한 하이퍼파라미터 값이고, 가장 앞의 숫자는 점수, 괄호 안은 표준편차 값이다. 점수와 std를 보고 최적의 하이퍼파라미터가 무엇인지를 찾을 수 있고, 또한 하이퍼파라미터 값이 변화함에 따라 점수와 표준편차가 어떻게 변하는지도 알아볼 수 있다.

6.6.2 랜덤 탐색

랜덤 탐색Random Search은 격자 탐색처럼 주어진 모든 값을 대입해 최적의 하이퍼파라미터를 찾는 것이 아니라, 주어진 값에서 표본을 추출하고 이들 값에 대해서만 하이퍼파라미터를 찾는 방법이다. 랜덤 탐색은 가능한 모든 조합의 하이퍼파라미터를 계산하지 않고 표본로 추출된 일부 값을 가지고 하이퍼파라미터를 계산하기 때문에 당연하게 격자 탐색보다 훨씬 적은 계산과 짧은 시간에 최선의 하이퍼파라미터를 찾을 수 있는 장점이 있다.

앞에서 '최적'이 아니라 '최선'의 하이퍼파라미터 값을 찾을 수 있다고 기술했는데, 그 이유를 알아둘 필요가 있다. 격자 탐색은 모든 경우의 수로 하이퍼파라미터 값을 찾지만, 랜덤 탐색은 전체집합인 모든 경우의 수에서 일부만을 표본으로 추출해 하이퍼파라미터 값을 찾기 때문에 반드시 최적의 하이퍼파라미터 값을 찾는다고 이야기할 수 없다. 부연 설명하면, 하이퍼파라미터를 찾기 위해 추출된 표본에 최적의 하이퍼파라미터를 찾을 수 있는 값이 포함되어 있다면 최적의 하이퍼파라미터를 찾을 수 있지만, 포함되어 있지 않다면 최적의 하이퍼파라미터는 찾을 수 없다.

랜덤 탐색에서 알아두어야 할 사실은 통상적으로 60개 이상의 표본을 이용한다면 랜덤 탐색의 성능이 격자 탐색보다 결코 성능이 떨어지지 않는다는 점이다. 제임스 버그스트라James Bergstra와 요슈아 벤지오Yoshua Bengio가 「Random Search for Hyperparameter Optimization」 논문[05]에서 이러한 사실을 밝혀냈다.

05 참조: http://goo.gl/efc8Qv

다음의 코드는 랜덤 탐색를 실시하는 예다.

```
def optimizeHyperparameterByRandomSearch(self,name, code, start_date,end_
date,lags_count=5):
    a_predictor = self.predictor.get(code,name)

    df_dataset = self.predictor.makeLaggedDataset(code,start_date,end_date,
self.config.get('input_column'), self.config.get('output_column'),lags_count)

    X_train,X_test,Y_train,Y_test = self.predictor.splitDataset(df_
dataset,'price_date',[self.config.get('input_column')],self.config.get('output_
column'),split_ratio=0.8)

    param_dist = {"max_depth": [3, None],
                  "min_samples_split": sp_randint(1, 11),
                  "min_samples_leaf": sp_randint(1, 11),
                  "bootstrap": [True, False],
                  "criterion": ["gini", "entropy"]}

    a_predictor.doRandomSearch(X_train.values,Y_train.values,param_dist,20)

def doRandomSearch(self,x_train,y_train,param_dist,iter_count):
    random_search =  RandomizedSearchCV(self.classifier, param_
distributions=param_dist, n_iter=iter_count)
    random_search.fit(x_train,y_train)

    for params, mean_score, scores in random_search.grid_scores_:
        print("%0.3f (+/-%0.03f) for %r" % (mean_score, scores.std() * 2, params))
```

실행결과

```
0.500 (+/-0.053) for {'min_samples_split': 4, 'bootstrap': True, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 4}
0.500 (+/-0.092) for {'min_samples_split': 10, 'bootstrap': True, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 3}
0.500 (+/-0.096) for {'min_samples_split': 3, 'bootstrap': True, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 8}
0.505 (+/-0.068) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 1}
0.495 (+/-0.066) for {'min_samples_split': 3, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 10}
0.505 (+/-0.077) for {'min_samples_split': 6, 'bootstrap': True, 'criterion':
'entropy', 'max_depth': 3, 'min_samples_leaf': 2}
0.495 (+/-0.082) for {'min_samples_split': 5, 'bootstrap': False, 'criterion':
'entropy', 'max_depth': 3, 'min_samples_leaf': 9}
```

```
0.511 (+/-0.086) for {'min_samples_split': 8, 'bootstrap': False, 'criterion':
'entropy', 'max_depth': 3, 'min_samples_leaf': 3}
0.495 (+/-0.066) for {'min_samples_split': 6, 'bootstrap': False, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 10}
0.495 (+/-0.082) for {'min_samples_split': 6, 'bootstrap': False, 'criterion':
'gini', 'max_depth': 3, 'min_samples_leaf': 8}
0.527 (+/-0.121) for {'min_samples_split': 7, 'bootstrap': True, 'criterion':
'gini', 'max_depth': None, 'min_samples_leaf': 6}
0.511 (+/-0.086) for {'min_samples_split': 10, 'bootstrap': False, 'criterion':
'entropy', 'max_depth': 3, 'min_samples_leaf': 4}
```

코드에서 알 수 있듯이 앞서 살펴본 격자 탐색과 코드가 거의 비슷하다. 차이가 있다면 GridSearchCV 대신에 RandomizedSearchCV 클래스를 사용하는 것과 값의 분포로 전달하는 것이다.

6.7 블랙 스완

블랙 스완Black Swan은 나심 니콜라스 탈레브Nassim Nicholas Taleb가 제시한 이론으로, 있을 수 없는 일이라고 생각한 일이 실제로 발생했을 때 예측하기 힘든 큰 충격이 발생한다는 내용을 담고 있다. 오스트레일리아에서 흑조가 발견되기 전까지 모든 사람은 백조만 있는 것으로 알고 있었다. 하지만 발견된 후에 과학계는 물론 많은 사람이 충격에 휩싸이게 되었다는 것으로 나심은 블랙 스완 이론을 설명하였다.

블랙 스완 이론은 서브프라임 사태와 같은 금융위기를 설명하는 데 도움이 되어 유명세를 얻게 되었다. 알고리즘 트레이딩에서 갑자기 블랙 스완 이론을 꺼내는 이유는 변동성과 확률에 관해 이야기하기 위해서다. 알고리즘 트레이딩 시스템은 어떠한 방법을 쓴다 하더라도 결론적으로 과거의 데이터를 이용해 미래를 예측해야 하는 숙명을 가지고 있다. 상승과 하락 같은 주가의 방향이든 주가 자체이든 간에 무엇인가를 예측하고, 그 결과에 따라 매수·매도를 하고, 수익을 창출하는 매커니즘을 가지고 있다.

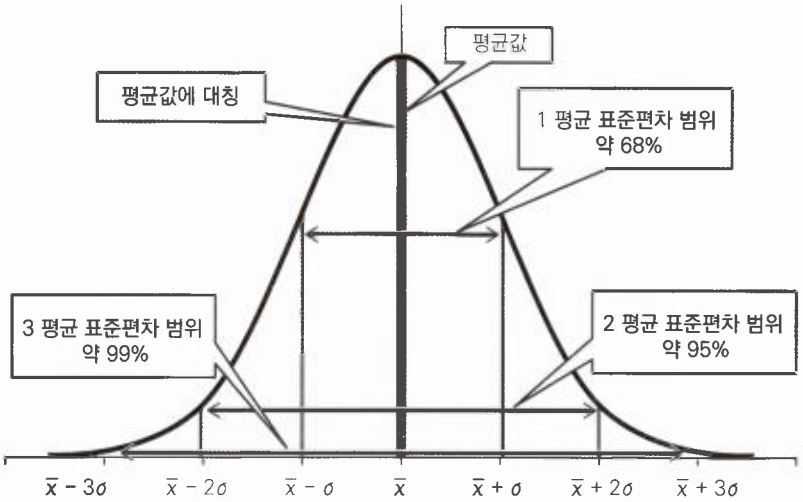
알고리즘 트레이딩에서, 특히 머신러닝 모델을 이용한 예측이라는 것은 어떤 일이 미래에 일어날 확률을 의미한다. 예를 들어, 주가 상승에 대한 확률값이 0.88이라면 매우 높은 신뢰도로 주가가 상승할 것이라 기대할 수 있지만, 반대로 생각해보면 주가가 하락하지 않을 확률은 1 - 0.88 = 0.12라는 것을 쉽게 알 수 있다.

자 그럼, 주식거래를 해야 하는데, 주식거래를 어떤 방식으로 할지를 결정해야 한다고 가정해보자. 개발한 알파 모델의 예측결과에 맞추어 매수·매도 포지션을 선택하는 것은 위험할 수 있다. 알파 모델은 기본적으로 매수·매도 기회를 발견해 알려주는 역할을 하기 때문에 상대적으로 매우 잦은 거래를 유도한다. 반대로 리스크 모델은 손실을 계산해서 되도록 거래하지 않게 하는 특성이 있다. 그렇다면 알고리즘 트레이딩 시스템에서 실제 매수·매도 포지션을 결정하는 포트폴리오 모델은 어떻게 동작하는 것이 위험을 최소화하면서 수익을 극대화할 수 있을까?

알고리즘 트레이딩 시스템에서 거래라는 것은 미래에 대한 예측을 바탕으로 시행하므로 100% 확실한 것은 아니다. 즉, 거래에 대한 위험과 보상이 공존하는 상황에서 결정해야 한다. 알고리즘 트레이딩에서 수익과 위험의 2가지 요소 중 우선순위가 높은 것은 망설임 없이 위험이다. 아무리 높은 수익이 기대된다 하더라도 한 번의 거래 실수로 상당 부분의 자산을 날려버릴 가능성이 크다면 거래하지 말아야 한다.

포트폴리오 모델에서의 핵심은 바로 이것이다. 수익 대비 위험에 따라 결정하는 Sharpe Ratio와 같은 개념으로 거래를 결정하는 것이 좋다. 그렇다면 위험은 어떻게 계산해야 하는가가 중요한 문제가 된다. 이때 사용하는 것이 확률분포다. 어떤 사건의 확률분포를 직접 구하거나 정규분포와 같은 것을 이용해 특정 사건이 발생할 확률값을 위험도의 측정으로 활용할 수 있다. 예를 들어, 확률분포에서 과거에 상승한 확률이 3%라면 상승은 거의 일어나지 않는 일이라고 생각해 위험도가 낮다고 가정하는 것이다.

그림 6-14 정규분포 그래프



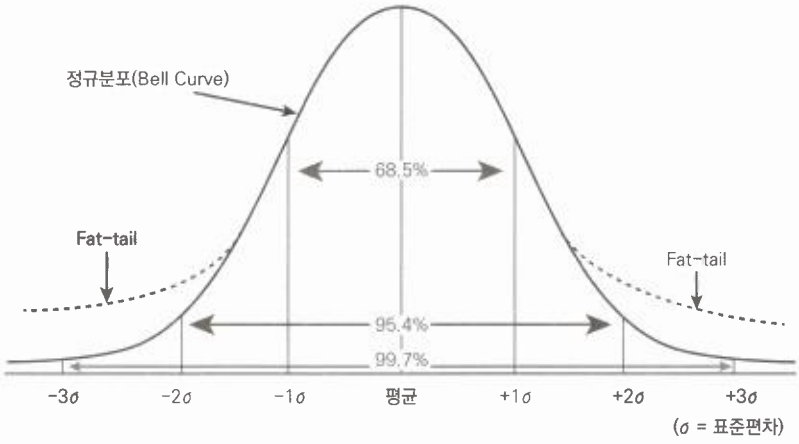
평균값과 표준편차를 가지고도 이런 판단은 얼마든지 가능하다. 예를 들어, 주가 평균이 10,000인데, 표준편차가 500이라면 주가가 표준편차의 2배 이하인 9,000 이하로 하락할 가능성은 5% 이하가 되어 발생하기 힘든 일이 된다. 알고리즘 트레이딩에서는 이런 방식으로 위험도를 계산한다.

그런데 재밌는 사실은, 현실에서는 발생 확률 5% 이하의 일이 생각보다 자주 일어난다는 점이다. 알고리즘 트레이딩 시스템은 수학과 IT에 깊은 조예가 있는 사람이 설계 및 개발하기 때문에 절대 허술하지 않고, 운영에 많은 돈을 투자해야 해서 위험 관리에 엄청나게 많은 노력을 투입하지만, 때때로 거대한 손실이 나는 것은 이와 무관하지 않다. 위험에 대한 계산은 앞에서 설명한 것처럼 'Thin-tailed' 분포를 가진 과거의 확률분포를 기반으로 계산하지만, 현실은 'Fat-tailed' 분포이기 때문이다.

'Fat-tailed' 분포는 확률분포의 양끝 가장자리가 두터운 것을 의미하는 것으로, 다음 그림처럼 정규분포도와 비교해보면 2σ(시그마), 3σ(시그마) 부분의 영역이 더

두터워 해당 영역이 발생할 확률이 높다는 것을 알 수 있다. 예를 들어, 정규분포에서는 2σ 이상의 값 변화가 일어날 확률이 100 - 95.4 = 4.6%라면 Fat-tailed 분포에서는 해당 확률이 5.6%, 8.7%처럼 4.6% 이상이 될 수 있다.

그림 6-15 Fat-tailed 분포 그래프



많은 수학적 이론과 방법이 정규분포를 가정하고 있는데, 정규분포는 Thin-tailed 분포로, 어떤 값이 〈평균값 + 표준편차×1〉의 범위에 있을 확률이 68.5%, 〈평균값 + 표준편차×2〉의 범위에 있을 확률이 95.4%, 〈평균값 + 표준편차×3〉의 범위에 있을 확률이 99.7%이다. Fat-tailed 분포는 정규분포와 달리 〈평균값 + 표준편차×2〉 또는 〈평균값 + 표준편차×3〉이 각각 정규분포의 95.4%보다 작은 90.4%, 정규분포의 99.7%보다 작은 96.3%가 될 수 있는 분포다(90.4%와 96.3%는 예를 들기 위한 수치로 Fat-tailed 분포의 실제 확률값은 아니다).

금융시장은 불확실성이 지배하는 세상이고, 이 세상에는 정규분포와 같은 Thin-tailed 분포가 아닌 Fat-tailed 분포가 더 적합한 이론으로 생각된다. 알고리즘 트레이딩 시스템을 설계 및 개발하려는 사람은 금융시장에서의 확률분포가 Fat-tailed 분포일 수도 있다는 것을 염두에 두고 개발할 필요가 있다.

chapter 7

마치며

이 책의 2가지 큰 주제인 머신러닝과 알고리즘 트레이딩은 내용이 방대하고 폭넓으며 다양한 분야의 지식이 필요하지만, 이 책에서는 입문자들을 위해 필수로 알아야 하는 핵심적인 개념과 이를 구현한 코드에 집중해 설명하였다. 머신러닝이라는 하나의 주제만으로도 책 한 권 분량은 쉽게 나올 수 있고, 알고리즘 트레이딩 역시 마찬가지다.

이 책을 집필하는 동안에 이세돌과 구글 딥마인 알파고의 바둑 대결이 있었다. 머신러닝을 공부하고 활용하는 사람으로서 이세돌과 알파고의 대결은 매우 흥미진진한 이벤트였고 과연 현재의 머신러닝 관련 기술이 얼마나 발전했는지를 보여주는 기회라서 바쁜 일정 사이사이에 4번의 대국을 지켜보았다. 머신러닝에 대해 비교적 잘 알고 있다고 자부했기에 알파고의 수준이 이세돌보다 뒤처질 거라고 생각했다. 그래서 당연히 이세돌의 압승을 예상해 첫 번째 경기는 보지 않았다. 알파고가 이겼다는 소식을 접하고, 하이라이트 영상을 보면서 대국의 전개 방향이 나의 예상과 다르다는 것을 아는 데는 시간이 얼마 걸리지 않았다. 두 번째 대국부터는 관심 있게 지켜보면서 알파고의 한 수 한 수에 감탄하고 말았다. 이세돌이 이길 거라고 생각한 이유는 Local Minima와 Global Minima의 밸런싱에 대한 것이었는데, 알파고는 이 문제를 너무 훌륭하게 잘 해결한 것으로 보였다. 머신러닝이 잘할 수 있는 일과 그 한계를 잘 알고 있다고 생각했는데, 알파고가 나의 편견을 여지없이 부숴버렸다. 생각보다 훨씬 더 다양한 문제에 머신러닝을 적용할 수 있고, 또 기술 수준도 많이 발전했다는 것을 느끼게 해준 고마운 계기였다.

오늘까지의 소프트웨어가 기능 중심의 발전을 해왔다면 내일부터의 소프트웨어는 지능 중심으로 혁혁한 발전을 할 것이라는 이제 자명해졌다. 알파고가 머신러닝에 대한 가능성을 보여준 것처럼 이제는 지능이라는 큰 목표가 확고히 세워져 다시 과거로 회귀할 수 없을 것이다.

머신러닝에 대한 중요성과 활용은 점차 확대되고, 우리의 삶 속에 하나둘씩 파고들 것이다. 그동안 사람이 직접 해야 했던 많은 일을 머신러닝으로 지능을 갖춘 소프트웨어들이 대신하게 될 것이고, 이러한 기술의 발전은 우리의 삶에 다시 재귀적으로 영향을 준다.

소프트웨어 개발자에게 머신러닝은 선택이 아닌 필수적인 기술 요소로 자리매김하게 될 것이며 사용자의 소프트웨어나 서비스 선택 기준 역시 기능 중심에서 편익 중심으로 바뀌게 될 것이다. 이 책에서 비중 있게 다룬 금융 분야는 머신러닝을 더욱더 많이 활용해 새로운 기술과 혁신을 끊임없이 만들어낼 것이다.

이 책의 목적은 머신러닝의 전체 흐름과 이를 구체적으로 어떻게 적용하는지를 보여주는 것이라서 나무보다는 숲에 주안점을 두어 내용에서는 설명이 부족한 부분과 소개하지 않은 개념들이 많다. 부디 부족한 부분은 관련 서적과 인터넷을 통해 보충하기를 바란다. 아무쪼록 저자의 작은 노력이 발화되어 지금보다 더 많은 IT 종사자들이 머신러닝과 금융에 많은 관심을 두기를 바란다.