

## <제목 차례>

1. 데이터 사이언스 정의 4
  - 1.1. 빅데이터와 AI 5
  - 1.2. 데이터 사이언스 범위 15
  - 1.3. 데이터 사이언스 응용 25
  - 1.4. 데이터 사이언스 도구 46
2. 파이썬 핵심 48
  - 2.1. 파이썬 실행 환경 49
  - 2.2. 파이썬 기초 78
  - 2.3. NumPy 117
  - 2.4. Pandas 129
3. 데이터 탐색 148
  - 3.1. 데이터 읽기 149
  - 3.2. 탐색적 분석 156

3.3. 시각화	170
4. 데이터 전처리	190
4.1. 전처리 타입	191
4.2. 데이터 변환	205
4.3. 유사도	229
5. 클러스터링	250
6. 머신러닝	285
6.1. 머신러닝 목적	286
6.2. 머신러닝 유형	289
6.3. 머신러닝 프로세스	300
6.4. 머신러닝 동작	312
7. 예측 모델	336
7.1. 선형 회귀	337
7.2. 선형 분류	389

7.3.	kNN	442
7.4.	결정 트리	456
7.5.	랜덤 포레스트	508
7.6.	로지스틱 회귀	561
7.7.	모델 성능 평가	595
7.8.	분류 알고리즘 성능 비교	629
7.9.	특성공학	641
7.10.	모델 최적화	659

# **데이터 사이언스 개론**

**김 화 종**

1. 데이터 사이언스 정의
2. 파이썬 핵심
3. 데이터 탐색
4. 데이터 전처리
5. 클러스터링
6. 머신러닝
7. 예측 모델

# 1. 데이터 사이언스 정의

- 데이터 사이언스를 간단히 정의하면, “데이터를 분석하여 의미 있는 결과를 얻는 기술”라고 하겠다. 먼저 데이터 사이언스 관련 용어를 알아보겠다.

## 1.1. 빅데이터와 AI

### 데이터 기술의 부각

- 데이터 사이언스는 정보통신기술(ICT)을 기반으로 하는 최신 기술이다. 지금까지 ICT 기술은 전자공학, 컴퓨터공학, 통신공학을 기본으로 발전하였으며 이들은 각각 하드웨어, 소프트웨어, 데이터 송수신을 다루었다.
- 전자, 컴퓨터, 통신 기술의 목적은 결국 데이터를 잘 다루는 것이었다. 음악을 듣거나, 영화를 보거나, 블로그를 이용하거나 홈쇼핑을 하는 것은 음악, 비디오, 텍스트, 상품 거래, SNS에 관한 “데이터”를 효과적으로, 안전하게, 편리하게 다루는 것이 목적이었다.
- 이제 데이터 기술(data technology)의 중요성이 새롭게 부각되고 있다.

- 데이터 사이언스에서는 데이터를 효과적으로 다루고 분석하는 기술을 포함하는데, 예를 들어 필요한 데이터를 수집하고, 변환하고, 알고리즘을 적용하여 분석하는 과정이 포함된다.
- 이제 모든 산업이 데이터 기반으로 동작하고 있다. 예를 들어 3D 프린터가 특별히 관심을 끌게 된 이유는 물체를 데이터로 변환하여 먼 곳으로 즉시 전송하고, 쉽게 재가공할 수 있기 때문이다.

## 4차 산업혁명

- 1차 산업혁명은 엔진의 발명으로 인간 근력 에너지를 기계가 대체한 것으로 시작되었다.
- 2차 산업혁명은 에너지를 전달하는데 전기를 사용함으로써 가정이나 공장에서 에너지를 쉽게 공급받을 수 있게 한 에너지 전달혁명이다.
- 1, 2차 산업혁명은 에너지의 생산과 전달에서 혁신을 이루었으며 인간 근육 노동력의 한계를 극복하였다. 즉, 사람이 힘을 쓰지 않아도 에너지를 기계로부터 얻을 수 있게 한 것이 핵심이다.
- 이를 통해 인간은 근육보다 머리를 사용하는 일에 집중할 수 있게 되었다. 블루컬러 보다 화이트컬러 노동자를 더 필요로 하였다.

- 3차 산업혁명은 전자, 컴퓨터, 통신 기술에 기반한 정보통신(ICT) 기술의 발전으로 전 산업을 ICT 기반으로 만들었다. 인터넷, 검색, 스마트폰 등은 모든 활동의 기본이 되었다.
- 4차 산업혁명은 빅데이터 기반의 지능화로 인공지능(AI)의 발전과 함께 진행되고 있다.
- 3, 4차 산업혁명은 인간 지적 노동력의 한계를 극복함으로써 사람들을 지적노동에서 해방시킬 것이다. 마치 1, 2차 산업혁명이 육체 노동에서 인간을 해방시킨 것과 같다.
- 이제 화이트컬러 노동의 상당 부분이 AI로 대체될 것이다.

## AI의 확대

- AI는 사람 수준으로 보고, 듣고, 말하는 수준으로 발전하였으며 이미지 인식, 음성인식, 문장 작성, 실시간 번역 수준이 나날이 발전하고 있다.
- AI는 감정은 가지고 있지 않지만 사람의 감정을 파악하여 어떻게 대화를 나누면 사람이 좋아할 지 피악하는 능력은 갖추고 있다. 이미 챗봇과 애완용 로봇은 사람과 대화를 나누거나 위로를 하는 수준으로 발전하였다.
- 현재의 AI는 특정한, 한정된 업무를 잘 수행하는 AI로서, 바둑을 두는 AI, 인공지능 스피커, 자율주행자동차, 회계처리 등에 사용되고 있다.
- 이러한 AI를 약한(weak) 또는 좁은(narrow) AI라고 부른다. 약한 AI 만으로도 인간의 직업에 큰 변화가 올 것으로 예상된다.

- 미래에는 강한(strong) AI가 발명될 것으로 예상되며 이는 사람처럼 종합적인 판단력과 의사결정도 하는 AI가 될 것이다.
- 강한 AI가 발명되까지는 수십 년이 걸릴 것으로 예상하지만 약한 AI들도 매우 유용하게 사용되고 있으며 앞으로 거의 모든 산업, 경제, 문화, 창작, 비즈니스에 AI가 접목될 것으로 예상되고 있다.

## AI의 미래

- AI가 중요한 이유는 AI가 우리의 일자리를 변화시키기 때문이다. 우리는 AI를 미래를 잘 예측하고 준비해야 불필요하게 AI와의 경쟁을 피하고 개인의 역량을 갖출 수 있다.
- 이제 어느 누구도 기계, 엔진과 힘겨루기 하지 않는다. 앞으로 상당히 많은 영역에서 데이터를 분석하고, 판단하고, 일을 처리하는데 컴퓨터와 경쟁이 안 되는 시기가 올 것이다.
- 심지어 그림을 그리거나, 작곡을 하거나 소설을 쓰거나 뉴스를 작성하는 소위 창작의 영역에서도 AI가 사람의 영역을 상당히 침범할 것이다. 이제 사람은 새로운 영역을 발굴하고 AI의 도움을 받는 전략을 택해야 한다.

- AI로 인해 직업의 변화가 크게 나타날 날 것이며 사라지는 직업과 새로 만들어지는 직업을 예상할 수 있어야 한다.
- 자율주행차는 아직 불안하여 도입이 안 될 것으로 생각하는 사람도 많겠지만 이미 배달용 무인차가 운영되고 있다. 힘든 일, 반복적인 일, 단순한 처리는 점차 AI가 대신하게 될 것이다.
- 또한 감정에 흔들리지 않고 편견을 같지 않고 판단을 한다는 것도 AI의 강점이다. 사람은 자신의 지식, 경험과 선입견에 많이 좌우된다.
- 인간이 AI보다 잘 하는 일은 감성이 개입되어야 하는 일, 종합적인 판단이 필요한 일, 뜻 밖의 엉뚱한 생각을 필요로 하는 일 (진정한 창의적인 일), 자원 봉사 등의 분야가 될 것이다.

- AI 시스템을 만드는 일과 AI에게 일을 시키는 것은 사람이 담당하게 될 것이다. 그러나 이러한 AI를 만들고 시키는 일도 점차 AI가 수행하게 될 것이다. 오히려 사람은 앞으로 AI와 협력하고 때에 따라서는 타협해야 할 것이다.
- 미래에는 안전을 고려한 AI의 개발이 필수적 요소이다. 마치 원자력에서 가장 중요한 것이 재난, 비상시에도 동작해야 하는 것과 같다. 또한 AI의 윤리에 대한 준비가 반드시 필요하다.
- AI의 도입으로 신입사원 채용시에 지원자의 미래 건강을 예측하여 몸이 아플 사람은 취업이 어려울 수도 있다.
- AI 동작에 대한 책임을 누가 질 것인지, 효율성만 높이면 되는 것인지?

윤리적으로 옳다는 것의 기준을 누가 정할지 등에 대한 대비가 필요하다.

- AI가 정교해질수록 사람들은 AI 기기를 마치 생각과 감정이 있는 것으로 착각하고 기대하게 되는데 이러한 기대는 더 높아질 것이다. 이러한 문제에도 대비해야 한다.

## 1.2. 데이터 사이언스 범위

- 데이터 사이언스는 ICT 기술의 발전 특히 빅데이터와 AI의 발전과 함께 부각되는 과학기술 분야이다. 먼저 관련된 기술을 살펴보면 다음과 같다.

## 통계 분석

- 데이터의 특성을 파악하는 데 수학적인 근거를 사용하는 분야로서 특히, 샘플 데이터로부터 전체 데이터의 속성을 파악하는데 가치를 둔다.
- 예를 들어 양케이트 조사를 통해서 선거 결과를 예측하거나, 제품의 불량률을 예측하거나, 가설의 타당성을 입증하는 것을 중요시한다.
- 오차범위, 신뢰도 등을 계산하고 데이터 해석에 어떤 의미가 있는지 여부 등을 검증하는데 필요한 이론을 제시한다.

## 데이터 마이닝

- 이미 구축해서 보유하고 있는 데이터베이스에서 새로운 지식을 얻는 것을 말한다.
- 예를 들어 기업이 구축한 매출, 고객, 상품 관련 데이터베이스를 종합적으로 분석하여 고객에 대한 새로운 구매 특성을 파악하거나 제품의 반품 문제를 파악할 수 있다.
- 데이터 마이닝의 목적은 평범한 데이터에 숨겨져 있던 가치 있는 새로운 지식을 얻는 과정이라고 하겠다.

## 비즈니스 인텔리전스

- 데이터 분석을 통해 새로운 비즈니스 전략을 얻는 것을 말한다.
- 데이터 분석 결과 내년에는 붉은색 옷이 유행할 것이라고 예측하든지, 어떤 지역에 거주하는 여성은 파란색 옷을 좋아할 것이라든지 등을 예측하여 마케팅이나 상품개발, 홍보에 활용하는 것을 말한다.
- 지난 수십년간 비즈니스 인텔리전스를 얻기 위해서 데이터 마이닝 기술이 널리 사용되었고 최근에는 빅데이터 분석을 사용한다.

## 빅데이터 분석

- 기존의 통계적 분석 방법이나 수학적으로는 검증할 수 없지만 대량의 데이터를 분석하여 새로운 의미 있는 정보를 얻는 것을 말한다.
- 예를 들어 대량의 소셜 데이터를 분석하여 사회적인 트렌드를 파악하거나, 센서 데이터를 분석하여 전에는 미처 생각하지 못한 상황에서 기계가 고장나는 현상을 발견하기도 한다.
- 마케팅에서도 예전과 달리 10대, 20대의 성향 분류 정도가 아니라 구매 시각, 장소, 구매이력, 상품이용 내역 등 관련된 빅데이터 분석으로 매우 정교한 고객 세분화가 가능해진다.

# 인공지능(AI)

- 인공지능이란 컴퓨터가 마치 지능이 있는 것처럼 똑똑하게 일을 처리하는 것을 말한다. AI가 실현되기 위해서 특정한 구현 방법이 필요한 것이 아니라, 구현 방법에 불문하고 지능이 있는 것처럼 동작하는 것은 모두 인공지능이라고 할 수 있다.
- 최근에, 딥러닝 기반의 머신 러닝 기술이 급속히 발전하면서 컴퓨터가 사람처럼 보고, 듣고, 쓰는 능력을 갖추게 되면서 인공지능의 지능이 사람에 근접하게 발전하고 있다.

# 머신러닝

- 데이터를 분석하여 분류(classification)나 회귀(regression) 분석 등 예측 모델을 만들고 새로운 데이터로 모델을 학습시켜 모델의 성능을 점차 개선시키는 방법을 말한다.
- 즉, “사람”이 분석을 담당하는 것이 아니라 머신(컴퓨터)이 학습을 하여 분석 능력을 키우는 방법이다.
- 모델은 스팸 메일을 분류하는 모델, 날씨나 주가를 예측하는 모델, 기계의 장애를 예측하는 모델 등을 말하며 선형회귀 모델, 결정트리모델, 로지스틱 회귀 모델 등을 사용한다.
- 최근 AI 성능이 급속히 발달한 것은 머신 러닝의 발달 때문이다.

## 딥러닝

- 딥러닝은 머신러닝의 한 방법으로서, 모델을 구성하는데 신경망을 사용하는 방법을 말한다.
- 딥러닝을 사용하면서 이미지 분석, 텍스트 분석, 음성 인식, 대용량 센서 분석의 성능이 급격히 개선되었다.
- 딥러닝은 예측 모델을 만드는 데 그치지 않고 두 종료의 이미지를 합성하여 새로운 이미지를 창의적으로 만들거나, 주제만 주면 작곡을 하거나, 키워드만 주면 뉴스 기사를 작성하는 작업 등이 가능해졌다.

## 사물 인터넷

- 사물인터넷(IOT: Internet of Things)이란 TV, 냉장고, 세탁기, 보안장치, 난방장치, 동물 등에 센서를 부착하고 통신 기능을 부여하여 데이터를 지속적으로 수집하거나 장치를 제어하는 기술을 말한다.
- IOT는 가전 제품 사용 패턴 분석, 가축의 이동 분석, 건강상태 모니터링, 기계 장치의 고장 파악, 위험 조기 발견 등에 폭넓게 사용된다.
- 사물 인터넷이 중요하게 부각되는 이유는 바로 센서 빅데이터를 생산하고 수집하여 빅데이터 분석과 AI 서비스가 확대되기 때문이다.

## 데이터 사이언스

- 데이터 사이언스는 통계적 분석, 데이터 마이닝, 빅데이터 분석, 머신러닝, 딥러닝과 모두 관련된 기술을 다루는 연구 분야이다.
- 지금까지는 데이터를 다루는 기술이 각 영역별로 발전하였으나 최근 빅데이터라는 키워드로 데이터 분석 분야가 주목을 받았고 이들을 전체적으로 아우르는 학문 영역을 데이터 사이언스라고 한다.
- 데이터 사이언스에 대한 지식과 프로그래밍 능력을 갖추고 데이터 분석과 머신 러닝 문제를 해결하는 전문가를 데이터 사이언티스트(데이터 과학자)라고 부른다.

## 1.3. 데이터 사이언스 응용

- 데이터 사이언스는 AI 서비스 개발, 제품개발, 공장자동화, 마케팅, 과학기술 연구, 건강관리, 재난예방, 맞춤형 교육, 증거 기반 의학 등 거의 모든 분야에서 이용되고 있다.
- 데이터 사이언스의 대표적인 응용 사례를 살펴보겠다.

## 생활

- 인공지능 스피커를 통해서 음악듣기 추천, 전화걸기, 알람 설정, 날씨, 뉴스 듣기, 동화책 읽기, TV 채널 예약 등이 이루어지고 있다.
- 인공지능 스피커는 이용자의 데이터 (언제 어떤 질문을 했는지)를 지속적으로 분석하여 점차 지능적인 서비스를 한다.
- 어떤 노래를 언제 주로 듣는지, 유사한 성향의 친구들은 어떤 음악을 좋아하는지를 파악하여 음악 추천의 만족도를 높일 수 있다. 컴퓨터가 사람과 대화를 하는 서비스인 챗봇은 어떤 주어진 일을 수행하는 경우와, 일반적인 질문에 답을 하거나(한국의 수도는?), 또는 친구와 대화를 하듯이 잡담을 나누는 것도 가능하다.

- 대량의 대화를 분석하면서 챗봇의 모델의 성능이 향상되며 개인 비서 역할도 수행한다.
- 제품으로는 애플의 홈팟(Homepod) 스피커를 이용하는 시리(Siri), 구글 홈/Home) 스피커를 이용한 구글 어시스턴스(Assistant), 아마존 에코(Echo) 스피커를 이용하는 알렉사(Alexa), 마이크로소프트 코타나(Cortana), 일본 라인(LINE)의 웨이브(Wave) 스피커를 통한 클로바(Clova), 중국 마이크로소프트의 샤오아이스(Xioice), SK텔레콤의 누구(Nugu) 등이 있다.
- 자기가 좋아하는 사람이 타입을 계속 선택하며 가장 근접한 사람을 소개해주는 서비스도 있다(Qunme). 이 외에서도 영화추천, 음악추천 (spotify), 직장추천(Career track), 의사추천(Sensy) 등이 있다. 이외에도 길 안내를

하는 내비게이터, 상품주문 도우미, 행정서비스 안내, 콜센터 안내 등에  
데이터 사이언스가 폭넓게 사용된다.

## 스마트 홈

- 인공지능 스피커는 스마트 홈을 관리하는 창구역할을 하여 냉난방 조절, 보안장치 관리 등에 사용되며 가정에서 발생하는 데이터를 취합 분석하는 것이 필요하다.
- 스마트 가전제품을 관리|하되, 데이터 분석 등의 대용량 연산은 클라우드에서 수행하고 디바이스는 신호의 입출력만 담당한다.
- 방의 구조를 학습하는 청소기(Rumba), 식사 메뉴를 제안하는 냉장고, 취향에 맞는 프로그램을 추천하는 TV, 오븐, 커피 메이커, 전기밥솥, 에어컨, 냉장고이 소개되고 있다.

# 마케팅

- 마케팅 분야는 고객의 취향을 점차 정밀하게 분석하여 고객 세분화를 정교하게 하고 대상 고객을 좁혀서 마케팅하는 타겟 마케팅으로 발전하고 있다.
- ]정말로 구매를 할 만한 사람에게 만 광고를 보내기 위해서 과거 상품구매 이력, SNS 분석을 하고 적절한 상품을 추천한다.
- 고객이 SNS를 사용하고 있는 도중에 또는 매장을 둘러보는 도중에 현재의 요구를 파악하여 실시간 추천도 한다.
- 새로운 영화가 소개되면 여기에 얼마나 투자를 해야할 지 또는 몇 개의 개봉관에서 이 영화를 상영하는 것이 적절할지를 예측하기 위해서 영화 시나리오 대본, 주인공 분석, 예고편에 대한 고객 반응을 분석한다.

- 영화의 예고편인 트레일러(trailer)를 만드는 데에도 사람이 수십일 걸리던 작업을 며칠 만에 인공지능이 제작하기도 한다. 어떤 장면을 어떻게 보여줘야 광고 효과가 높을지를 예측하는 것이다.
- 콜센터에서는 고객 성향에 맞게 잘 대응할 적합한 직원을 배정하거나, 고객이 평소에 자주 궁금해 하는 정보를 바로 자동응답기로 안내해주기 주기 위해서 고객의 과거 이용 기록, 목소리 감성분석을 통해 까다로운 고객인지, 또는 지금 무슨 용건으로 전화를 했는지를 예측한다.
- 예를 들어 평소에 카드 납입액을 자주 물어본 고객이라면 기다리는 동안 미리 카드 납입액을 자동으로 알려준다.

## 로봇

- 공항, 호텔, 전시관 등에서 안내, 접수를 담당하는 안내 로봇이 선보이고 있다. 일본 헨나 호텔은 공통 모습의 안내 로봇이 유명하다. 객실까지 짐을 운반하는 로봇, 청소에도 사용된다. 인건비를 절약하여 숙박비가 준다.
- 장난감 로봇으로 안키(Anki)사의 코즈모(Cozmo)는 10cm 크기의 로봇으로 1천개 이상의 감정 표현이 가능하다.
- 여러 가지 용도로 활용할 수 있는 범용 로봇으로는 소프트뱅크의 페퍼(Pepper)가 있어 가정이나 기업에서 목적에 맞게 기능을 프로그래밍 할 수 있다. 페퍼에는 감정을 수치화 하여 표현하는 기능을 가지고 있다.
- 수거된 쓰레기를 선별하는 로봇도 실용화되었다. 핀란드의 젠로보틱스

(ZenRobotics)는 인공지능으로 로봇 팔을 동작시킨다.

## 금융

- 금융사에서는 개인이나 기업에 대출을 위한 신용 평가에 머신러닝 기법을 도입하고 있다. 과거의 대출 사고 사례, 개인의 금융 활동을 분석한다. 도난된 신용 카드 사용을 찾아내기 위해서 평소 정상적인 거래를 분석하고 이상 현상을 발견한다.
- 보험사는 사기성 보험 청구를 찾는데, 불법자금 세탁을 찾아내는데도 머신러닝이 사용된다.

## 핀테크

- 금융과 기술의 융합을 핀테크라고 하며 여기에 인공지능 도입이 시도되고

있다. 지금은 고객의 상품 추천, 서비스 대화 챗봇에 도입되는 정도이지만 고객의 신용평가, 대출한도 설정 등이 자동화되면 은행원의 감축과 매장 축소가 예상된다.

- 투자를 담당하는 트레이더 직원수가 이미 급격히 줄었고 자동주식거래 프로그램은 초고속으로 매매(High Frequency Trading)를 하여 수익을 낸다.
- 신용카드 부정 사용을 탐지하기 위해서 고객의 평소 카드 사용 이력을 분석하여, 특이한 거래가 발생하면 이를 의심하고 재 점검한다.

## 안전

- 테러리스트와 같은 위험 인물을 파악하거나 위험한 행동을 할 사람을 사전에 찾아내는데 머신러닝을 사용한다. 요일, 시간, 장소별로 범죄가 발생하는

패턴을 분석하여 미리 경찰관을 현장에 보내어 범죄 발생률을 낮추고 있다 (PredPol, Hunch Lab).

- 사람의 동작을 보고 범죄 발생을 예측하거나 얼굴 이미지를 분석하기도 한다. 컴퓨터가 범죄유형의 인물을 보고 학습을 하는 것이다. 미국의 원 컨선(One Concern)에서는 자연 재해 발생시에 예상되는 피해 범위를 시뮬레이션하고 피해 예측 지도를 생성하는 솔루션을 만들었다.

## 산업

- 공장 자동화, 생산 라인의 최적화, 상품의 물류과 재고 관리에는 오래전부터 IT와 소프트웨어 기술을 도입하고 있지만 이제 이러한 과정에 다양한 데이터 분석 기술을 도입하고 있다.
- 예를 들어 여러 부품 중에 필요한 부품을 선택하는 작업은 숙련된 사람만이

작업할 수 있었으나 이제 로봇도 몇 시간만 가르치면 숙련자 이상의 정확도로 선택을 할 수 있다.

- 생산된 제품중에 불량품을 찾아내는 일, 기계의 고장 발생을 예측하거나 원인을 예측하는 일이 가능하다. 식료품 공장에서는 음식을 품질을 판독하는데에도 사용된다.
- 스마트 공장(smart factory)에서는 발생하는 데이터를 상세히 분석함으로써 더 효과적인 생산, 유통, 재고관리를 하며 제품 자체의 품질 향상에도 이용하고 있다. 이를 산업 4.0(Industry 4.0)이라고도 부르면 독일, 일본에서 주도적으로 추진하고 있다.

## 에너지 관리

- 가정에 부착하는 스마트 미터기(smart meters)를 통해서 수집되는 데이터

를 보면, 가정에서 현재 어떤 가전 제품을 사용하고 있는지도 파악이 가능하며 불법적으로 전기를 도용하는 것도 찾아낼 수 있다. 정전을 예측하는데에도 사용된다.

- 전기 에너지의 효과적인 생산과 배분을 위한 에너지 관리 체계인 스마트 그리드에서는 전력 수요 피크 시간대에 전력이 부족하지 않게 하되 평소에는 남는 에너지를 최소화해야 한다.
- 이를 위해서는 전력 수요를 정확히 예측하여야 하며 신재생 에너지의 불규칙한 에너지 공급 현상도 예측할 수 있어야 한다. 에너지 생산시 이를 자판매하여 얻을 수익을 예측하는 데 사용한다. 에너지 사용의 효율화에도 인공지능 기술이 사용된다.
- 에너지 생산 다변화 대응에도 필요한데, 전력수요가 급증하는 시간에는

이를 대체할 다른 에너지원을 공급해야 한다 (Peaker plant 운영). 전력 수용 피크 타임은 시기, 장소에 따라 다르다.

- 가정에서는 일반적으로 가전제품 사용이 많은 저녁시간이지만 더운 날은 오후에(에어컨), 추운날은 아침에(히터), 피크 타임이 된다.
- 전기 자동차는 빅데이터의 역할을 하여 개인이 충전한 전기를 한전에 되파는 데에도 사용될 수 있다. 이때 적정한 판매 시점과 가격을 예측하는 데에도 빅데이터 분석을 사용한다.
- 도시 단위의 효율적인 에너지 이용을 위한 스마트 그리드도 인공지능으로 진화할 것이며 적절한 시간에 적절한 장치를 구동하는 제어를 지능적으로 수행하는 것이다.

## 자동차

- 전기 자동차의 사용이 늘어나면 전력 그리드에도 영향을 주기 시작할 것이다. 만일 전기 자동차들이 같은 시간대에 동시에 충전을 하면 전력 수요가 급증하여 전력공급에 문제가 생길 수 있다. 따라서 자동차들이 서로 충전 시간을 적절히 분산 조정할 수 있다면 전력 이용을 짧은 시간대에 몰리지 않게 조정할 수 있다.
- 일부 자동차 보험회사에서는 차량에 블랙박스를 달면 보험료를 감해 준다. 어떤 보험사는 더 나가서 블랙박스 데이터를 실시간으로 보험사로 보내주면, 예를 들어 급정거를 하는지, 급커브를 하는지 등의 정보를 제공하면 보험료를 더 감해준다.
- 보험사는 운전습관에 관한 실제 정보를 얻고 이를 통해 고객의 안전한

운전을 유도하며 동시에 사고를 줄일 방법을 찾는다. 우리나라 SKT에서는 티맵 사용자의 내비게이터 운행 기록을 보고 운전자의 안전운행 정도를 파악하여 이를 자동차 보험 상품과 연계하여 보험료를 절약하게 해주는 서비스를 하고 있다.

## 자율주행차

- 인공지능의 가장 혁신적인 성과는 자율주행차(자율차)의 상용화에서 나타날 것이다. 자율차의 목표는 모든 운전을 자동차가 스스로 하고 사람의 개입이 거의 필요 없는 완전 자율차이다.
- 벤츠, BMW, GM, 테슬라와 같은 자동차 회사는 모두 자율주행차를 개발하고 있다. 자동차 보험가입을 운전자가 가입하지 않고 자동차 제작회사가 가입하게 되면 자율차의 상용화가 급속이 확대될 것이며 사람은 교통사고 처리로부

터 해방될 것이다.

- 사람이 운전하지 않으면 교통사고 발생도 90% 이상 줄어들 것으로 예상하고 있다. 도로의 상황을 정확히 인식하고 지나가는 사람의 다음 행동을 예측하며 돌발 상황에 안전하게 실시간으로 대응하려면 인공지능 기술이 지금보다 더 발전해야 한다.
- 우리가 자율항법 장치를 따르는 비행기를 믿고 타듯이 자율차를 믿고 탈 시기가 올 것이다.
- 승용차에 앞서 일정한 루트를 운행하는 버스나, 순환버스, 트럭, 물류 배송차에 우선 도입될 것이다. 모든 차 자율차로 되면 운영이 수월하겠지만 당분간은 사람이 운전하는 자동차와 자율차가 혼재하는 상황에서 교통사고를 줄이는 기술 개발이 되어야 할 것이다.

## 건강

- 사람은 누구나 병이 들고 건강이 악화된다. 건강 분야는 피할 수 없이 개선책이 필요한 분야이며 특히 고령화 진행과 간병인 부족 현상으로 건강 분야에 인공지능과 도우미 로봇의 도입은 필수적으로 확대될 분야이다.
- 기본적으로는 몸에 지니고 다니는 (wearable) 측정 장치를 이용하여 혈압, 심장박동, 수면 패턴, 걸음수, 운동 패턴 등을 종합적으로 분석하여 운동이나 음식을 추천하는 건강관리 추천 서비스가 있다. 병의 진단을 예측하는 기능이 있다.
- 수치 분석과 x-레이 사진, 단층사진 판독 등을 돋는다. 수많은 사진을 보고 의사가 판단하는 것보다 컴퓨터의 이미지 처리 능력이 훨씬 앞서고 있다.

- 앞으로 개인별로 다른 정보를 활용하면 질병 예측과 치료방법 제시도 정확해 질 것이다. 의학논문과 사례를 컴퓨터에 학습시켜 진단 성능을 높이고 있으며 대표적으로 IBM의Watson을 이용한 시도가 이루어지고 있다.
- 또한 개인의 DNA 염기서열, 유전자 발현 분석을 통해서 개인별 질병 예측과 치료방법 제시도 상당히 정확하고 있다. 개인 간의 약 복용에 대한 효과 차이를 구분할 수 있다면 맞춤형 처방 또는 맞춤형 신약개발이 가능해진다.
- 이러한 분야의 성패는 데이터의 확보이다. 특히 어떤 경우가 질병인지를 표시해주어야 하는데, 이때 전문가 의사의 판독이 필요하기 때문에 학습에 필요한 데이터 수집 비용이 증가하는 문제가 있다.

- 구글에서는 의무전자기록을 분석하여 환자의 치료 결과를 예측하였는데, 주요 관심사항인 입원중 사망확률, 장기간 입원 확률, 퇴원후 30일 이내 재입원률, 퇴원시의 진단명을 예측하는 데 좋은 성과를 보였다.
- 심리 상담 분야에서도 사람 인간 의사보다 AI의사에게 더 솔직하게 상담하는 경향이 있다.
- 남캘리포니아대학교에서 개발한 가상의 여성 상담사 로봇(SimSensei)은 환자의 대화 등 언어적 요소 뿐 아니라 시선, 미소, 머리 움직임, 표정 등 비언어적 요소도 분석한다.
- 전통적으로 의료 분야는 전문가인 의사의 경험을 중요시하여 인공지능의 도입이 늦어질 것으로 예상되지만 다음과 같은 분야에서는 인공지능의

도입이 적극 추진되고 있다.

- 예를 들어, 판독에 많은 시간과 노력이 드는 분야, 문제의 난이도가 낮은 분야, 보험 수수가 낮은 분야, 그리고 인공지능 도입으로 비용감소와 효율이 분명히 향상되는 분야 등이다.
- 진단은 의사만 할 수 있기 때문에 아직은 진단 보조 도구로만 활용되지만 그 중요성은 점차 커지고 있다. 우리는 정확한 X-레이 사진과 초음파 사진이 진단에 도움이 되듯이 인공지능의 도움을 당연히 도입하게 될 것이다.
- 의료법에 관한 내용과, 개인 프라이버시 정보 보호에 대해 건강정보는 더 민감하므로 대책이 필요하다. 개인정보를 처음부터 익명화하고 판독이 불가하게 변환하여 사용해야 한다.

## 스포츠

- 선수의 동작과 활동을 분석하는데 널리 사용된다. 야구에서는 도루, 안타율 등을 분석하여 경기에 투입하는데 사용하고, 축구, 농구, 배구에서도 선수의 움직임을 모두 추적하여 선수의 컨디션, 운동량, 득점기회, 위치 예측 등을 수행한다.
- 아디다스의 miCoachElite 등이 있다. 감독과 코치는 인공지능의 도움을 받아 상대 선수의 공격 전략 등을 파악하고 있다.
- 경기 판독에서 인공지능이 사용되는데 고속으로 움직이는 체조 경기의 동작, 난이도 채점에 도입하였다.

## 1.4. 데이터 사이언스 도구

- 데이터 사이언스에서 사용할 수 있는 도구는 자바, C와 같은 일반 프로그래밍 언어와 R과 같은 통계 분석에 특화된 언어, 그리고 매트랩(matlab), SPSS, SAS와 같은 데이터 분석 도구가 있다.
- 최근에는 파이썬(python) 언어가 데이터 분석에 가장 많이 사용된다. 파이션이 인기가 있는 이유는 다음과 같다. 먼저 파이썬은 일반 프로그래밍 언어이면서 자바, C에 비해 문법이 간단하여 배우기가 쉽다.
- 파이썬은 R과 유사하게 데이터를 다루는데 편리한 기능을 제공하다. 구글에서 2015년에 공개한 데이터 분석 라이브러리이며 특히 딥러닝 모델을 구축하는데 널리 사용되는 텐서플로우는 파이썬을 인터페이스 언어로 사용

한다.

- 텐서플로우를 사용하려면 파이썬을 이용해야 한다. 한편 딥러닝 프로그램을 쉽게 작성할 수 있게 해주는 커라스(keras) 라이브러리도 파이썬으로 작성한다.
- 이와 같이 딥러닝을 이용한 프로그램 작성에서 파이썬을 기본 언어로 채택함으로써 이제 머신러닝 영역 뿐 아니라 일반 데이터 분석에서 폭넓게 파이썬을 도입하고 있다.

## 2. 파이썬 핵심

- 현재 데이터 사이언스 분야에서 가장 널리 사용되는 언어는 파이썬(python)이다. 파이썬의 장점은 문법이 간단하고 배우기 쉽다는 것이다. 파이썬에는 여러 사람이 작성한 무료 라이브러리가 방대하게 제공되고 있다.
- 여기서는 파이썬으로 작성된 데이터분석, 머신러닝, 딥러닝 예제들을 이해하는데 필요한 필수적인 파이썬 문법을 소개한다.

## 2.1. 파이썬 실행 환경

- 파이썬 개발 환경으로 가장 널리 사용되는 것이 쥬피터 노트북(Jupyter notebook)이다. 쥬피터 노트북을 사용하면 프로그램 코딩, 코드 블록단위의 동작확인, 실행결과 출력 화면 보관, 문서 작성 등을 편리하게 할 수 있다.
- 쥬피터 노트북의 편리한 점은 모든 동작을 웹기반으로 수행할 수 있다는 것이다. 즉, 임의의 컴퓨터에서 웹 브라우저를 통해서 서버의 개발 환경을 이용할 수 있다.
- 구글이 제공하는 colab을 쥬피터 노트북으로 무료로 사용할 수도 있다.

## 파이썬 설치

- 파이썬과 쥬피터 노트북은 아나콘다(Anaconda) 패키지로 한 번에 간편하게 설치할 수 있다. 아나콘다는 <http://anaconda.com> 사이트에서 다운로드할 수 있다. 화면 우측의 다운로드 버튼을 클릭한다.

Downloads

- 아나콘다는 윈도우, 맥, 리눅스 버전을 각각 제공하며 현재 파이선 버전 3.7을 제공한다. 윈도우 사용자의 경우 사용하는 PC가 32 비트 버전인지 64비트 버전인지를 먼저 PC 시스템에서 확인한 후 해당 버전을 다운로드 해야 한다.
- 아래는 아나콘다 사이트에서 다운로드 메뉴를 선택한 다음의 화면을 보였다.

파이썬 버전 2.7이 필요한 경우는 예전에 작성된 2.7 버전의 프로그램을 수행하는 것이 필요한 경우이다. 그 외에는 버전 3.7을 다운로드해야 한다.

**Python 3.7 version \***

 Download

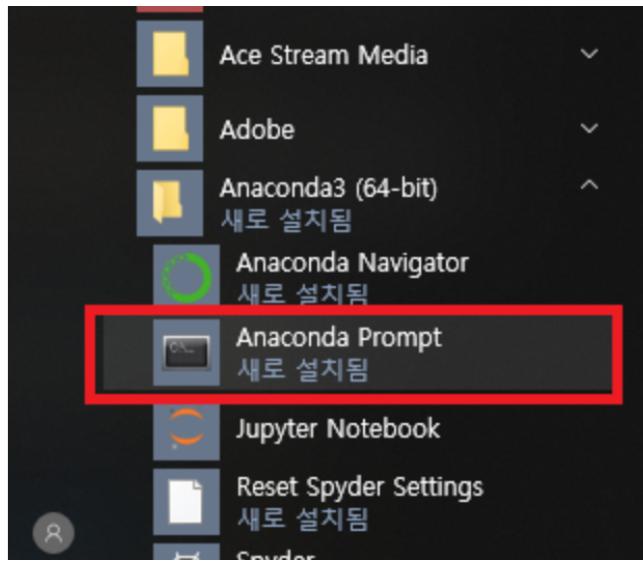
[64-Bit Graphical Installer \(633 MB\)](#) (?)  
[32-Bit Graphical Installer \(510 MB\)](#)

**Python 2.7 version \***

 Download

[64-Bit Graphical Installer \(580 MB\)](#) (?)  
[32-Bit Graphical Installer \(458 MB\)](#)

- 아나콘다를 다운받은 후에는 기본 옵션을 사용하여 설치를 한다. 설치를 마치면 윈도우의 경우 프로그램 찾기에서 아나콘다 명령창(Anaconda Prompt)를 실행한다



---

Anaconda Prompt

```
(base) C:\Users\Yunice>"C:\Users\Yunice\Desktop\Jupyter workplace"
'C:\Users\Yunice\Desktop\Jupyter workplace' is not recognized as
operable program or batch file.

(base) C:\Users\Yunice>
```

- 파이썬과 관련된 작업은 아나콘다 프롬프트에서 명령을 내리면 편리하다. 아나콘다 프롬프트는 윈도우 명령창과 같은 역할을 하되 파이선 환경에서 실행되는 것이다.
- 윈도우의 경우 아나콘다 프롬프트는 아래와 같이 (base)가 보이는데 이는 파이선 가상환경이 기본(base)로 설정되어 있다는 것이다.

(base) C:\Windows\Users\hjkim

- 가상환경은 개발 환경을 독립적으로 구성하는 것이 필요할 때 사용한다. 예를 들어 수행하는 프로젝트마다 추가로 필요한 패키지의 종류나 버전이 다른 경우 이를 각각 다른 가상환경에서 작업하면 마치 서로 다른 컴퓨터를

사용하듯이 환경 조건을 다르게 만들 수 있다.

- (base) 다음에 나타나는 폴더 경로 위치를 기억해 두는 것이 필요하다. 이 경로가 바로 쥬피터 노트북이 처음 시작되는 홈 디렉토리가 된다.
- 쥬피터 노트북은 이 것보다 상위의 폴더에 있는 파일은 검색이 안 된다. 이 폴더의 하위에 작업 폴더를 만들고 필요한 파일을 여기에 저장해 두어야 한다.
- 여기서 설치된 파이선의 버전을 다음과 같이 확인 할 수 있다. (현재 경로 이름은 생략함)

```
(base) python -V
```

- 파이썬에서는 아나콘다가 기본으로 제공하는 라이브러리 외에 새로운 라이브러리를 추가로 설치할 일이 많은데, 이때 아나콘다 프롬프트에서 pip (python install program)을 이용하여 추가 패키지를 설치할 수 있다.
  - 파이썬 버전 3에서는 pip대신 pip3를 사용하기도 한다. pip의 사용 예는 아래와 같다.

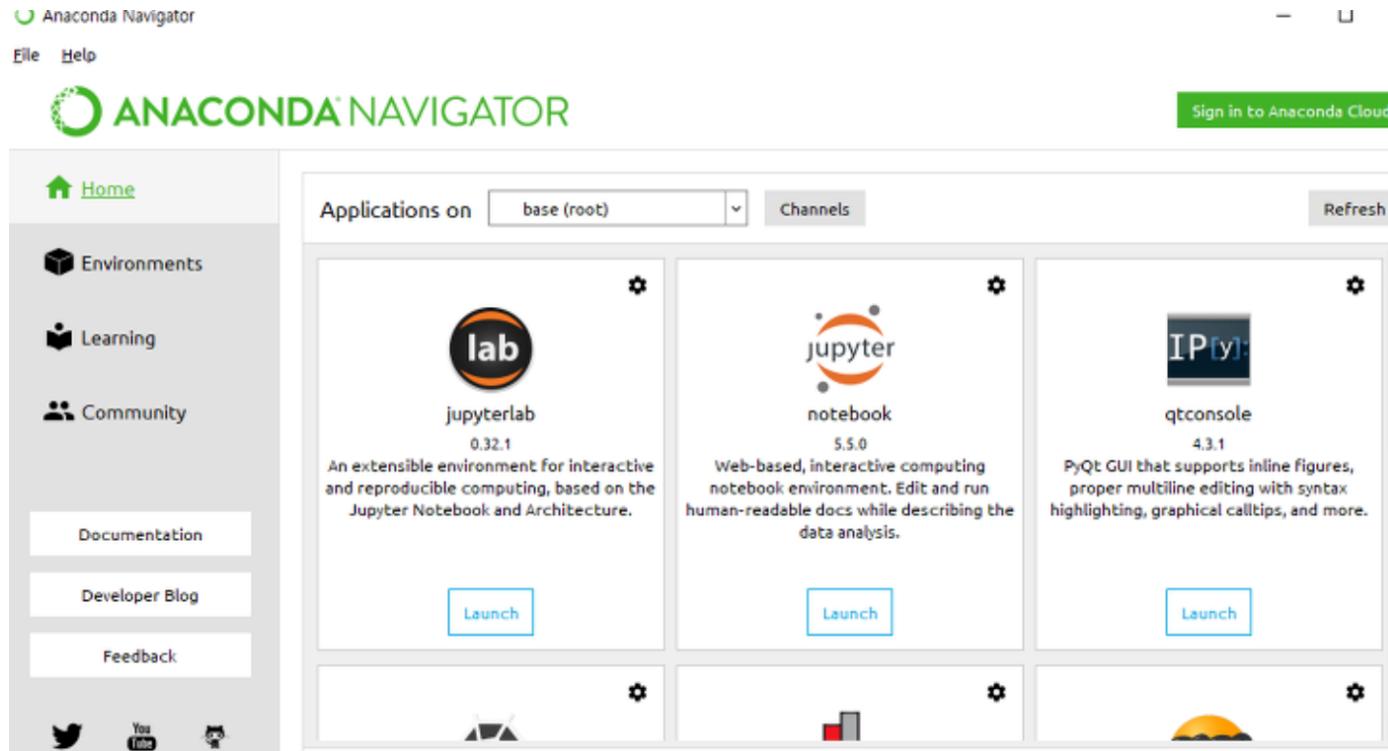
pip --version pip0 설치되었는지 확인

`pip install pkg` 특정 패키지 “pkg”를 설치한다

`pip install --upgrade pip` 패키지 pip 자체를 업그레이드한다

- 맥 컴퓨터나 리눅스에서는 아나콘다 프롬프트가 아니라 일반 명령창에서 pip 명령을 사용할 수 있다.

- 아나콘다 네비게이터를 사용할 수도 있다(속도가 느리다)



## 쥬피터 노트북 실행

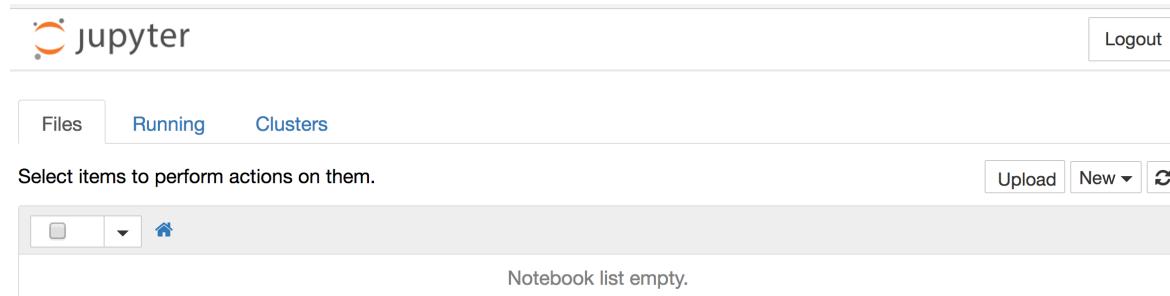
- 아나콘다를 설치하면 프로그램 메뉴에서 쥬피터 노트북(Jupyter Notebook) 프로그램을 실행할 수 있다. 또는 아나콘다 명령창(anaconda prompt)에서 jupyter notebook 명령으로 실행할 수 있다.

(base)jupyter notebook

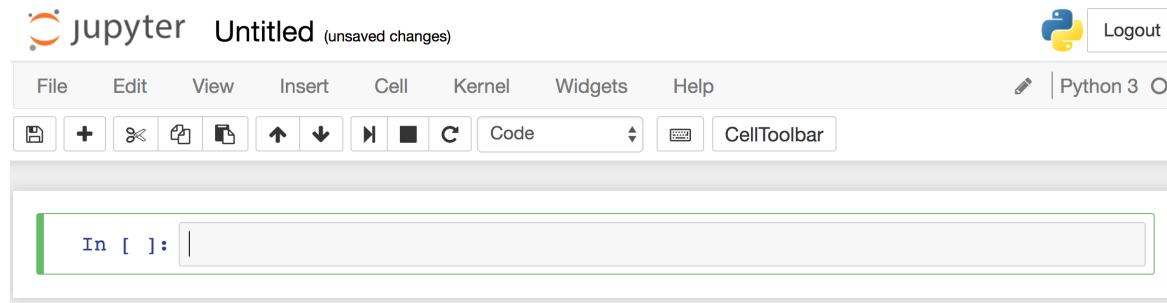
- 맥이나 리눅스에서는 일반 명령창에서 수행하면 된다.

\$ jupyter notebook

- 쥬피터 노트북은 프로그램 코드 작성, 실행 결과 화면 저장, 문서 작성 세가지 작업을 한 곳에서 지원하는 통합 개발 환경이다.
- 쥬피터 노트북은 웹기반으로 동작하며 자신의 PC가 로컬 서버 역할을 한다.
- 쥬피터 노트북을 실행하면 다음과 같은 화면이 나타난다.

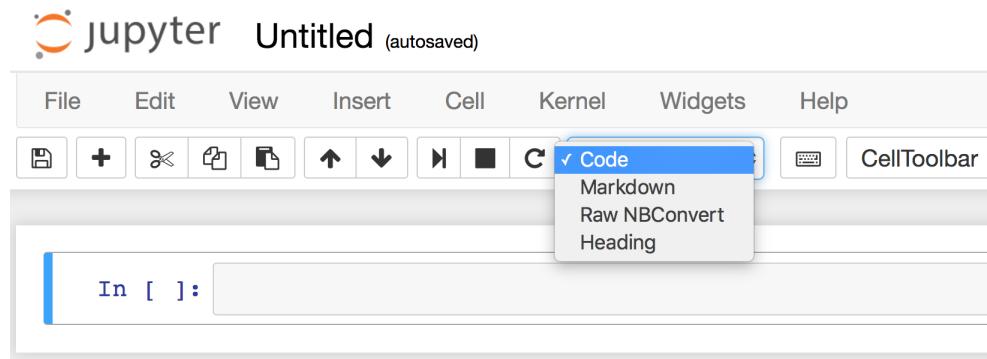


- 위의 쥬피터 화면에는 현재 폴더에 아무 파일도 없다는 것을 나타낸다. 여기서 ‘현재 폴더’란 쥬피터 노트북이 구동되는 홈 디렉토리를 말한다. 혹시 다른 곳으로 프로그램이나 데이터를 다운로드 받았으면 이 홈 디렉토리 또는 하위 폴더로 복사해야 한다.
- 홈디렉토리 위치는 아나콘다 명령창의 (base) 다음의 경로 주소를 참조하면 된다.
- Files 메뉴는 현재 폴더에 있는 모든 파일을 보여준다. 쥬피터에서 다루는 프로그램 파일은 노트북(notebook)이라고 부르며 확장자는 .ipynb이다.
- 새로운 노트북을 만들려면 맨 우측 New 버튼을 클릭하고 Python3 노트북 을 선택하면 된다. 아래와 같은 새로운 노트북 창이 나타난다.

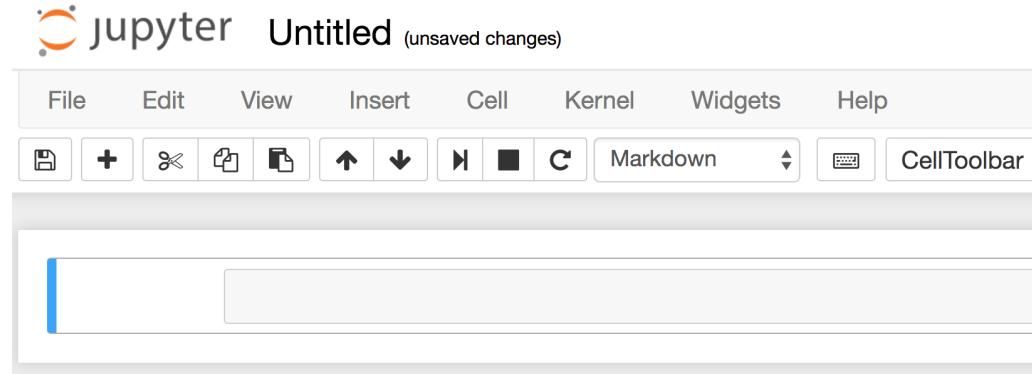


- 새로 만들어진 노트북의 이름은 “Untitled”이며 이 부분을 클릭하여 이름을 바꿀 수 있다. 위에서 In [ ] 부분을 셀이라고 하는데 현재 이 셀의 타입이 Code로 되어 있다(툴바 중간에 Code라는 단어가 보임).
- 이렇게 만들어진 코드 셀(code cell)에 파이썬 코드를 입력할 수 있다.
- 셀 타입을 문서 작성용으로 바꿀 수 있다. 아래와 같이 Code 우측의

드롭다운 메뉴를 누르면 Markdown을 선택할 수 있는 메뉴가 나타난다.



- 여기서 Markdown을 선택하면 아래와 같이 In [ ] 없는 입력창이 나타난다. 이를 마크다운 셀이라고 하는데 이 셀은 문서를 만드는데 사용된다. 즉 쥬피터 노트북의 셀은 코드 영역과 문서 영역으로 구분된다.



- 위의 마크다운 셀의 내부를 클릭하면 문서 내용을 입력할 수 있는데, 예를 들어 아래와 같은 내용을 입력하였다고 하자.

The screenshot shows a Jupyter Notebook window titled "Untitled (unsaved changes)". The toolbar includes standard options like File, Edit, View, Insert, Kernel, Widgets, and Help, along with specific notebook controls for cell selection, execution, and output. A "CellToolbar" button is also present. The main content area displays a Markdown cell containing the following text:

**# 파이선 예제**  
##### 파이선 프로그래밍을 배우는 첫 단계

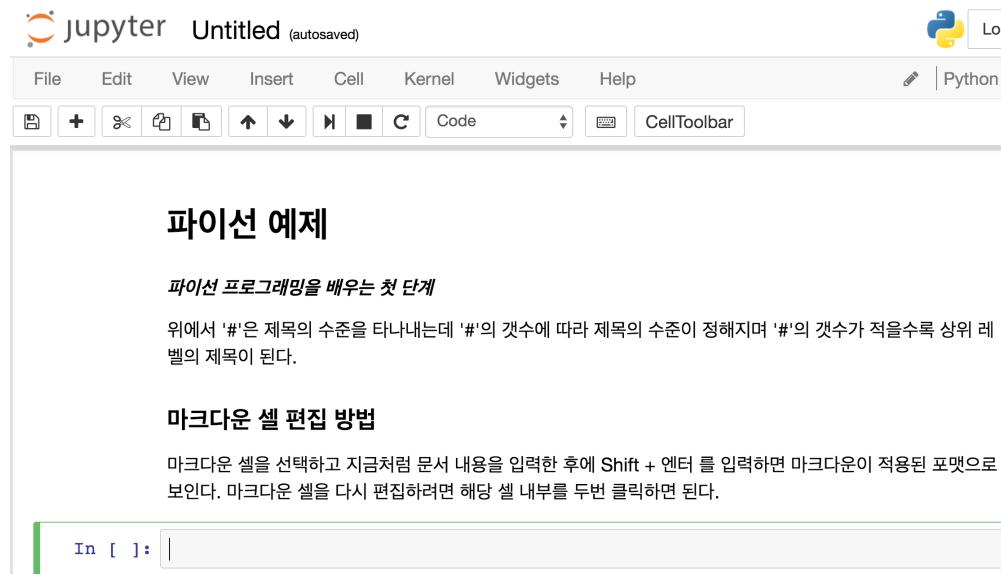
위에서 '#'은 제목의 수준을 나타내는데 '#'의 갯수에 따라 제목의 수준이 정해지며 '#'의 갯수가 적을수록 상위 레벨의 제목이 된다.

**### 마크다운 셀 편집 방법**  
마크다운 셀을 선택하고 지금처럼 문서 내용을 입력한 후에 Shift + 엔터 를 입력하면 마크다운이 적용된 포맷으로 보인다.  
마크다운 셀을 다시 편집하려면 해당 셀 내부를 두번 클릭하면 된다.

- 마크다운은 간단히 서식을 지정하는 방식인데 제목의 수준을 '#'의 개수로 나타낸다. '#'의 갯수가 적을수록 상위 레벨의 제목이 된다. '#' 기호 뒤에

스페이스가 하나 있어야 한다.

- 문서 내용을 입력한 후에 “Shift + 엔터”를 입력하면 셀이 실행된 결과 출력 포맷이 보인다(아래 그림 참조). 마크다운 셀의 내용을 편집하려면 해당 마크다운 셀 내부를 두 번 클릭하면 된다.



- 코드 셀에는 파이썬 코드를 입력할 수 있으며 입력된 코드를 실행하려면 “Shift + 엔터”를 입력하면 된다. “Shift + 엔터” 후에는 커서가 아래 셀로 이동하며 아래 셀이 없으면 아래에 새로운 셀을 생성한다.

The screenshot shows a Jupyter Notebook window titled "Untitled (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help, along with various icons for file operations and cell management. A sub-toolbar below the main one includes icons for file, cell creation, cell deletion, cell selection, cell execution, cell copy/paste, and cell search, followed by "Code" and "CellToolbar".

**마크다운 셀 편집 방법**

마크다운 셀을 선택하고 지금처럼 문서 내용을 입력한 후에 Shift + 엔터 를 입력 보인다. 마크다운 셀을 다시 편집하려면 해당 셀 내부를 두번 클릭하면 된다.

```
In [9]: x = (1, 2, 3, 4, 5)
         print (x)
         print (x*3)

(1, 2, 3, 4, 5)
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

In [ ]:

- 셀을 실행시키지 않고 편집 모드에서 빠져나오려면 Esc 키를 누르거나 화면의 셀 외부 다른 곳을 클릭하면 된다.
- 셀 편집 모드에서 빠져 나오면 명령 모드가 된다. 편집 모드의 셀은 초록색이고 명령 모드는 파란색으로 보인다. 명령 모드에서 a나 b 키를 입력하면 빈 셀을 현재 셀의 위(above)나 아래(below)에 하나 추가한다.
- 마크다운 셀이나 코드 셀을 실행할 때 Shift + 엔터 가 아니라 Ctrl + 엔터 키를 사용하면 셀을 실행한 후 커서의 위치가 다음 셀로 이동하지 않고 현재 셀에 남아 있다.
- 인접한 두 개의 셀을 합하려면 Shft + m 명령을 입력한다. 하나의 셀을 두 개로 분할하려면 분할하고 싶은 위치에서 Shift + Ctrl + '-' 키를

입력하면 된다.

- 윈도우에서는 가끔 이 명령이 스크린 축소 명령으로 인식된다. 그 때는 셀을 나누려면 메뉴에서 Edit – Split Cell을 선택한다.

## 파이썬 라이브러리

- 파이썬의 장점은 유용한 라이브러리가 많다는 것이다. 기본적인 라이브러리는 numpy, pandas, matplotlib, scikit-learn 등이 있으며 딥러닝 모델을 이용하려면 텐서플로우, 케라스 등을 추가로 설치해야 한다.

### **numpy**

- numpy는 Numerical Python의 의미로 계산 속도를 높여주는 라이브러리를 제공한다. 데이터 처리에 많이 사용되는 다차원 어레이(매트릭스)를 제공하며 매트릭스를 사용하면 계산 속도가 빠르다. numpy는 내부적으로 C 언어로 작성되어 있다.

### **pandas**

- pandas 패키지는 데이터를 마이크로소프트의 액셀을 사용하듯이 테이블 구조의 데이터를 편리하게 조작하는데 쓰이는 패키지다.
- numpy가 매트릭스의 연산 (곱셈, 덧셈 등)을 빠르게 처리하기 위한 것이라면 판다스는 테이블 구조의 데이터의 컬럼 추가, 컬럼 삭제, 조건에 맞는 행 추출 등을 편리하게 수행할 수 있게 한다.
- 판다스는 테이블 구조의 데이터를 담기 위해서 데이터프레임(DataFrame)을 제공한다.

## **matplotlib**

- matplotlib는 그림을 그리는데 사용되는 라이브러리이다. 히스토그램, 박스 플롯, 직선 그리기, 산포도(scatter plot) 등을 그리는 함수를 제공한다.

## **scikit-learn**

- scikit-learn은 간단히 sklearn이라고 부르며 선형회귀, 결정트리, 랜덤포레스트 등 머신러닝 알고리즘들을 포함하는 라이브러리이다.

## **프로그램 개발 환경**

- 파이썬과 쥬피터 노트북 등 프로그램 실행 환경 구축 환경은 다음과 같이 세가지 경우로 나눌 수 있다.

### **개인 PC에서 사용하는 방법**

- 개인 PC에 파이썬을 설치하면 속도는 느리지만 개인이 독립적으로 사용할 수 있다는 장점이 있다. 운영체제는 윈도우, 맥, 리눅스로 나누어진다.

### **원격 서버를 사용하는 경우**

- 원격 서버에서 실행되는 파이썬 서버를 쥬피터 노트북을 통해서 사용하며 개인용 PC는 접속용 웹 브라우저를 실행하는 터미널로만 사용하는 방법이다.

- 원격 서버는 보통 리눅스 운영체제로 구축한다. 원격 서버도 실험실이나 회사에 전용 서버를 구축하는 경우도 있고, 아마존, 구글, 마이크로소프트 등의 클라우드 서버를 이용하는 방법도 있다.

## 구글 colab

<https://colab.research.google.com>

- 구글이 제공하는 코랩(colab)은 파이썬 쥬피터 노트북 환경을 무료로 제공한다. 코랩에서는 GPU도 사용할 수 있어 신경망 프로그램을 구동할 때 속도가 빠르다.
- 코랩은 구글 colab 사이트에 접속하여 사용할 수 있는데 프로그램 환경설정

내용과 업로드한 데이터를 12시간만 유지해준다는 단점이 있다. 즉, 12시간이 지나면 서버에서 데이터가 사라진다.

- 코랩에 파일을 업로드하거나 코랩에서 작성한 프로그램을 저장하는 방법은 여러 가지가 가능하다. 자신의 컴퓨터, 구글 드라이브, 또는 github에서 파일을 읽거나 저장하는 것이 가능하다.

# github

- <https://github.com/StillWork>

## 가상환경 구성

- 파이썬 개발 환경을 개인이 혼자 사용하는 것이 아니라면 사용자마다 설치하는 라이브러리가 다르므로 서로 다른 실행 환경을 요구할 수 있다.
- 또는 한 사람이 사용하더라도 수행하는 프로젝트마다 필요로 하는 라이브러리나 버전이 다를 수 있어 프로젝트마다 서로 다른 파이썬 실행환경을 구축할 필요가 있다.
- 이를 위해서 파이썬 가상환경 (virtual environment) 설정을 이용하면 편리하다. 가상 환경 설정이 처음에는 조금 복잡해 보이지만 안전하고 일관성 있는 프로그램 개발에 도움이 된다.
- 리눅스에서 가상 환경을 최초로 설정하는 절차는 다음과 같다. (리눅스

서버를 사용하지 않는 경우, 즉 윈도우나 맥을 사용하는 경우는 별도의 블로그 등을 참조하기 바란다.)

- 먼저 파이썬 관련 프로그램 설치 프로그램을 최신 버전으로 업데이트 한 후 virtualenv 패키지를 설치한다.

```
pip3 install --upgrade pip
```

```
pip3 install --user --upgrade virtualenv
```

- 홈 폴더에서 아래와 같은 명령으로 가상환경으로 운영할 하위 폴더를 만든다.

```
export ML_PATH="$HOME/textbook"
```

```
mkdir -p $ML_PATH
```

```
cd $ML_PATH  
virtualenv env
```

- 이렇게 만들어진 가상 환경을 활성화 하려면 아래와 같이 activate 명령을 수행한다.

```
cd $ML_PATH  
source env/bin/activate
```

- 이제 가상 환경으로 전환되며, 필요한 파이썬 관련 패키지를 여기서 설치하면 이는 이 가상 환경에만 설치된 것으로 인식된다.

## 2.2. 파이썬 기초

- 여기서는 이 책의 내용을 이해하는데 필요한 최소한의 파이썬 문법을 소개한다.
- 정수, 소수, 문자열 등 기본 변수와 리스트, 튜플, 딕셔너리를 소개한다.

## 기본 변수

- 파이썬의 기본 변수는 정수, 소수, 불리언, 문자열 등이 있다.
- 파이썬에서는 변수명과 변수 타입을 미리 정의할 필요 없이 임의로 즉시 만들어 사용할 수 있다.
- 단, 기본적으로 사용하는 예약어(if, for, True 등)는 변수로 사용할 수 없다.
- 아래는 변수 a, x, y, z에 여러 가지 기본 변수 값을 배정하는 것을 보였다.  
값 배정(assign)시에는 “=” 기호를 사용한다.

a = 0.5

```
x = 3  
y = True  
z = "홍길동"  
print (a,x,y,z)    # 출력: 0.5 3 True 홍길동
```

- 문자열은 ‘ ’ 또는 “ ” 으로 묶어서 표시하며 여러 줄(line)으로 된 문자열은  
“ ” 으로 묶어서 표시한다.

multi = """  
이것은 여러 줄로  
구성된 문자열을 표시하기 위한  
방법입니다. """

```
print(multi)
```

# 결과는 아래와 같다

이것은 여러 줄로  
구성된 문자열을 표시하기 위한  
방법입니다.

## 논리값

- 논리값은 True 또는 False 중에 하나의 값을 갖는다 (대소문자를 구분한다).
- 파이썬에서는 논리값이 마치 숫자인 것처럼 연산에 바로 사용할 수 있는데, 이때는 True는 1로, False는 0으로 변환되어 계산에 사용된다. 아래에서 x의 값은 30이고 y의 값은 True인데 이들을 더하면 y값이 1로 자동 변환된다.

```
print (x + y)          # 출력: 4
```

## 리스트

- 여러 데이터를 하나의 변수에 저장하는 방법으로 리스트가 사용된다. 여러 데이터를 나열(Isit)했다는 의미이다. 리스트는 정수, 소수, 문자, 논리값 등 임의의 데이터 타입을 담을 수 있다. 리스트를 만들려면 [ ]를 사용한다.

```
list_1 = [x, y, z]  
print(list_1)      # [3, True, '홍길동']
```

- 리스트 내에 들어있는 항목의 갯수는 len() 함수로 구한다.

```
print(len(list_1))  # 출력: 3
```

## in

- 리스트 내에 어떤 항목이 들어 있는지를 확인하는데 `in`을 사용된다. 해당 항목이 들어 있으면 `True`를 리턴한다.

```
print(3 in list_1)      # True  
print(4 in list_1)      # False
```

- 리스트에 항목을 추가하려면 `append()`를 사용한다.

```
list_1.append(100)  
print(list_1)           # [3, True, '홍길동', 100]
```

- `append()`를 실행하면 리스트 `list_1`의 내용이 바뀌는 것을 주의해야 한다.

## range

- 일정한 범위의 숫자로 구성된 리스트를 한 번에 만드는 편리한 함수가 있다. range()를 사용하면 다음과 같이 일정 간격(디폴트 간격은 1임)의 숫자 리스트를 만든다.

```
x2 = range(10)      # 리스트 [0,1,2,3,4,5,6,7,8,9]
```

- 초기 값과 간격을 10이 아니라 임의의 값으로 줄 수 있다. 아래는 초기값이 10이고 마지막 값이 18인 정수의 리스트를 만든다. x3의 내용을 모두 출력해 보기 위해서 for 문을 사용하는 예를 보였다.

- for 문의 조건문 뒤, 반복 수행되는 영역 앞에는 반드시 “:”가 있어야 한다. 그리고 반복 수행되는 블록은 반드시 동일한 크기로 들여쓰기(indent)를 해주어야 한다.

```
x3 = range(10, 20, 2)

for i in x3:

    print(i)          # 10, 12, 14, 16, 18
```

- 리스트 내의 특정 위치의 값을 얻으려면 인덱싱(indexing)을 사용한다. 인덱스 값은 [ ] 내에 입력한다. 아래에서 변수 x3의 인덱스 “0”은 첫 번째 리스트 값인 10을 얻고 2는 세 번째 값을 얻는다. 인덱스로 -1를

쓰면 맨 뒤의 값을 얻는다. -2면 뒤에서 두 번째 항목을 리턴한다.

```
print(x3[0])      # 10  
print(x3[2])      # 14  
print(x3[-1])     # 18  
print(x3[-2])     # 16
```

## 리스트 내부 for

- 리스트 내부에서 for문을 사용할 수 있다. 아래는 변수 x를 0부터 4까지 사용하여 리스트를 만드는 코드로서 이 코드를 [ ] 내부에서 작성할 수 있다.

```
list_2 = [x*10 for x in range(5)]  
print(list_2)      # [0, 10, 20, 30, 40]
```

## 튜플

- 튜플(tuple)도 리스트와 유사하게 여러 항목의 집합체를 나타낸다. 튜플이 리스트와 다른 점은 항목의 값을 바꿀 수 없다는 것이다. 즉, 튜플은 상수화된 리스트라고 볼 수 있다.
- 튜플을 사용하는 이유는 항목의 내용을 더 이상 내용을 변경하는 것을 방지하는 목적으로 있고, 처리속도를 빠르게 한다는 장점도 있다.
- 튜플을 만들려면 ( )을 사용하거나 아무 괄호를 넣지 않아도 된다.

```
t1 = (1,2,3)
```

```
print(t1)           # (1, 2, 3)
```

```
t2 = 4,5,6
```

```
print(t2)           # (4, 5, 6)
```

# 사전(dictionary)

- 딕셔너리는 모든 항목이 항상 “키(key)”와 “값(value)” 두 가지로 짝을 이루어 구성되는 데이터 형식이다.
- 딕셔너리 형식 데이터를 만들 때는 { }를 사용한다. 항목의 값은 키 값을 인자로 하여 얻을 수 있다. 아래는 이름과 나이를 각각 키와 값으로 정한 딕셔너리 예이다.

```
name_age = {"kim": 20, "lee": 25}
```

```
name_age["kim"]           # 20
```

- 어떤 딕셔너리 내에 특정한 키 값이 들어 있는지 아닌지를 in으로 확인할 수 있다. 들어 있으면 True 아니면 False를 리턴한다.

```
print("kim" in name_age)      # True  
print("park" in name_age)     # False
```

- 딕셔너리에 새로운 항목을 추가하려면 “키” 값을 인자로 주면서 새로운 값을 배정하면 된다.

```
name_age["song"] = 35  
print(name_age)    # {'kim': 20, 'lee': 25, 'song': 35}
```

- 딕셔너리에 어떤 키가 있는지 보거나 또는 값들만을 출력할 수 있다. 아래와 같이 각각 keys(), values() 함수를 사용하여 키와, 값을 각각 얻을 수 있다.

```
name_age.keys()      # dict_keys(['kim', 'lee', 'song'])  
name_age.values()    # dict_values([20, 25, 35])
```

- 딕셔너리에 대해 items() 함수를 사용하면 딕셔너리 각 항목을 튜플로 만들고 이 항목들을 모두 리스트로 리턴한다. 딕셔너리를 튜플로 바꾸는 이유는 튜플로 만듬으로써 검색 속도를 빠르게 하기 위해서이다.

```
name_age.items() # dict_items([('song', 35), ('lee', 25), ('kim', 20)])
```

## if-else

- if-else 문을 사용하면 조건에 따라 프로그램의 흐름을 정할 수 있다. elif을 사용하면 조건을 두 가지가 아니라 여러 가지 조합으로 나누어 적용할 수 있다. elif 는 else if의 뜻이다.

x=3

```
if x > 10:  
    print("x > 10")  
  
elif x > 3:  
    print("10 >= x < 3")  
  
else:  
    print(" x < 3")
```

# for

- for를 사용하여 어떤 조건이 만족되는 동안 작업을 수행할 수 있다. 아래는 변수 x의 값으로 튜플 (1, 3, 5, 10)의 값을 각각 사용하여 필요한 작업을 수행한다.

```
for x in (1,3,5,10):  
    print(x, "'s sqaure = ", x*x)
```

```
# 출력  
1 's sqaure = 1  
3 's sqaure = 9  
5 's sqaure = 25  
10 's sqaure = 100
```

## sort()

- 리스트 항목의 내용들을 값의 크기 순으로 정렬할 때 sort()를 사용한다. 기본적으로 오름차순으로 정렬한다. sort()를 실행하고 나면 리스트의 내용이 바뀌는 것을 주의해야 한다.

```
x = [2,1,3,5,4]
```

```
x.sort()
```

```
print(x)      # [1, 2, 3, 4, 5]
```

- 만일 원래 리스트의 내용이 변경되지 않게 하려면 sorted() 메소드를 사용하고 새로운 변수에 배정하면 된다.

```
x = [2,1,3,5,4]
y = sorted(x)
print(y)          # y = [1, 2, 3, 4, 5]
print(x)          # x = [2, 1, 3, 5, 4] 원래 값을 유지
```

- sort()는 기본적으로 오름차순으로 정렬한다. 내림차순으로 정렬하려면 속성값 reverse에 True를 설정한다.

```
x = [2,1,3,5,4]
x.sort(reverse=True)
print(x)          # x = [5, 4, 3, 2, 1]
```

- sort\_values()는 데이터프레임을 정렬하는 명령어로 특정 열을 기준으로

정렬 할 수 있다. by에는 기준 열, ascending은 오름차순, 내림차순 정렬을 선택하는 변수이다.

```
df = pd.DataFrame({'순서': [1, 3, 2],  
                   '이름': ['park', 'lee', 'choi'],  
                   '나이': [30, 20, 40]})  
  
df.sort_values(by=['순서'], ascending=True)  
  
>>  
  
    나이   순서     이름  
0   30    1   park  
2   40    2   choi  
1   20    3   lee
```

## 들여쓰기

- 파이썬에서는 if, elif, else, for, def 문에 내부에 해당하는 블록(단락)을 구분하기 위해서 들여쓰기를 사용한다. 들여쓰기는 보통 탭을 사용하지만 탭의 크기는 블록 내에서 통일 시켜야 한다.
- 예를 들어 탭으로 4개의 공란을 사용한다면 모든 탭의 크기를 4로 일정하게 해야 한다.
- 스페이스나 리턴은 ( ) 코드 블록 내부나 { } 블록 내부에서는 무시되므로 코드의 길이가 긴 경우 보기 쉽게, 여러 줄로 표시해도 동작은 동일하다.
- 아래는 2차원 리스트를 만들면서 보기 좋게 여러 줄로 표시한 예이다.

```
x = [[1,2,3],
```

```
[4,5,6],  
[7,8,9]]  
print(x)           # [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

- ( ) 나 { } 괄호가 없는 경우에 코드를 여러 줄로 나누어 쓰려면 아래와 같이 문장 끝에 역 슬래시 “\”를 사용해서 줄이 연속되는 것을 나타낼 수 있다.

```
x = 3 + 4 + 5 + 6 \  
     + 7
```

## 모듈 설치

- 파이썬의 기본 기능 외에 추가 기능을 사용하려면 해당 모듈을 설치해야 한다. 모듈을 서버에 설치하는 것은 한 번만 하면 되는데 pip install을 사용하여 설치한다.
- 설치된 모듈을 프로그램에서 사용하려면 프로그램 첫 부분에서 이를 import로 불러와야 한다.
- 예를 들어 아래는 파이썬에서 그림을 그리는데 사용하는 matplotlib 패키지의 pyplot모듈을 불러오며 이를 plt라는 약자로 대신 사용한다는 것을 나타낸다.

```
import matplotlib.pyplot as plt
```

## 함수 정의

- 사용자가 임의의 기능을 함수를 정의하려면 def를 사용한다. 아래를 함수 이름이 double이고 인자로 받은 값의 두배를 리턴하는 함수이다.

```
def double(x):  
    return x*2  
double(5)      # 결과는 10
```

- 함수의 인자 값으로 디폴트 값을 지정할 수 있다. 아래와 같이 함수를 정의하면 함수를 호출할 때 인자 값을 주지 않으면 디폴트로 인자 x의 값에 100이 배정된다.

```
def double(x=100):  
    return x*2  
  
double()      # 200
```

- 함수 실행 결과로 두 개 이상의 값을 리턴할 때에는 리턴 값을 튜플로 처리하면 편리하다. 아래 함수는 리턴 값이 2배수와 3배수 두 개로 구성되는데 이 두 값이 각각 변수 x와 y로 배정된다.

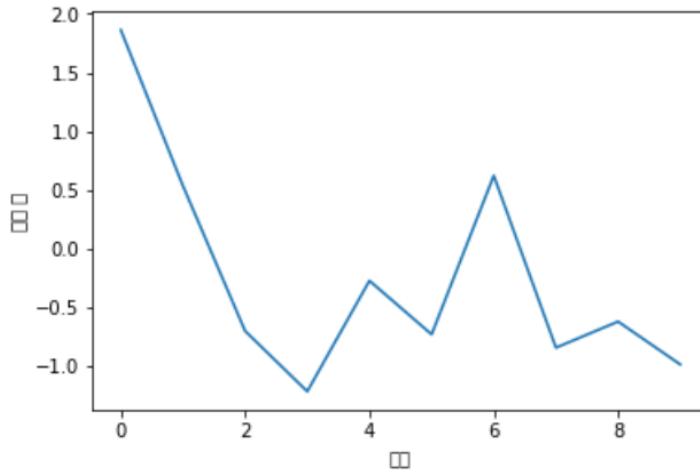
```
def double_triple(x):  
    return x*2, x*3  
  
x, y = double_triple(4)  
print(x, y)      # x = 8, y = 12
```

# matplotlib

- 파이썬의 시각화 라이브러리이다. 아래는 10개의 랜덤한 숫자를 만들고 이들을 선으로 연결하는 코드이다.

```
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
n = 10  
y = np.random.randn(n)  
plt.plot(range(n), y)  
plt.xlabel('시간')  
plt.ylabel('랜덤 값')
```



- 위 코드에서 `%matplotlib inline` 의 의미는 그래프를 현재 쥬피터 노트북 화면에 직접 나타나게 그리라는 뜻이다.
- x축과 y축 라벨에 한글이 출력되지 않았다. matplotlib에서 한글이 보이도록 하려면 다음과 같은 코드를 먼저 실행해 주어야 한다.

```
import platform  
from matplotlib import font_manager, rc  
import matplotlib  
  
# '-' 부호가 제대로 표시되게 하는 설정  
matplotlib.rcParams['axes.unicode_minus'] = False  
  
# 운영 체제마다 한글이 보이게 하는 설정  
# 윈도우  
if platform.system() == 'Windows':  
    path = "c:\\Windows\\Fonts\\malgun.ttf"  
    font_name = font_manager.FontProperties(fname=path).get_name()  
    rc('font', family=font_name)  
  
# 맥
```

```
elif platform.system() == 'Darwin':  
    rc('font', family='AppleGothic')  
# 리눅스  
elif platform.system() == 'Linux':  
    rc('font', family='NanumBarunGothic')
```

## random

- 랜덤하게 발생하는 숫자를 데이터 분석에서 자주 사용한다. 먼저 0~1 사이의 랜덤 숫자를 5개 출력하는 코드는 아래와 같다.

```
import numpy as np  
np.random.rand(3)  
=>  
array([0.00505926, 0.54385491, 0.92803911])
```

- 랜덤 숫자 발생을 이용하여 원주율 ( $\pi$ )를 시뮬레이션으로 구하는 방법을 소개하겠다. 원의 면적인  $\pi * r^{**2}$  이므로 아래 그림과 같이 반경이 1인 원과 사각형의 관계에서 랜덤하게 0~1 사이의 값을 두 개씩 만들고 이를

각각  $(x, y)$  좌표하고 하면, 이 점이 원 내부에 들어올 확률을 실제 개수를 세어서 구할 수 있다. 확률은 면적의 비율이므로  $\pi$ 를 다음과 같이 구할 수 있다.

$$\pi = 4 * (\text{랜덤한 점이 원 내부에 들어올 확률})$$

- 이를 프로그램으로 구현하면 다음과 같다. 아래에서 랜덤하게 1만개의 샘플 점을 생성하였다.  $n$ 의 개수를 늘리면 더 정확한  $\pi$  값을 시뮬레이션으로 구할 수 있게 된다. 이러한 기법을 몬테카를로 방식이라고 한다.

```
import numpy as np  
import matplotlib.pyplot as plt
```

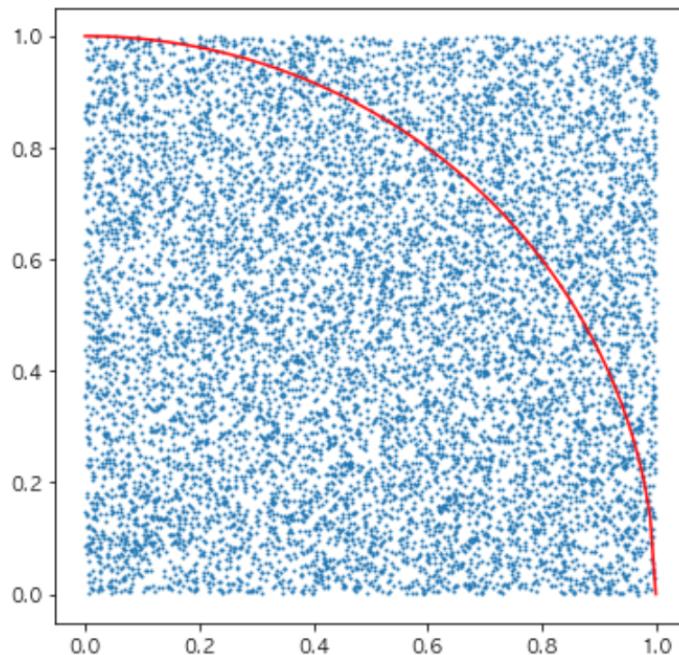
```
n = 10000
x = np.random.rand(n)
y = np.random.rand(n)
plt.figure(figsize=(6,6))
plt.scatter(x,y, s=1)
xx=np.linspace(0,1,100)
plt.plot(xx, (1-xx*xx)**0.5, c='r')
```

pi = ((x\*\*2 + y\*\*2) < 1).mean()\*4

pi

=>

3.1512



- 소수가 아니라 정수를 랜덤하게 생성하려면 randint를 사용한다. 아래는 1~6 사이의 정수 10개를 랜덤하게 얻는 코드이다.

```
np.random.randint(1,7,10)
```

=>

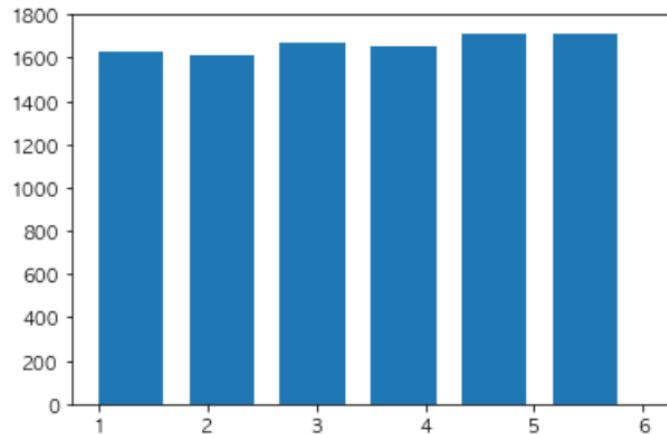
```
array([3, 2, 4, 6, 4, 1, 5, 1, 6, 5])
```

- 아래는 주사위를 여러번 던져서 1~6 사이의 눈금이 몇 번 나오는지를 확인하는 코드이다 (아래에서는 10000번 주사위를 던졌다)

```
x = np.random.randint(1,7, 10000)
```

```
plt.hist(x, bins = 6, width=0.6)
```

```
plt.show()
```



- 이번에는 주사위를 두 개를 동시에 던져서 나오는 값의 합의 분포를 구해보겠다. 두 개를 던지면 2~12의 값이 나오는데 이를 1만 번 시행한 경우의 히스토그램은 아래와 같다.

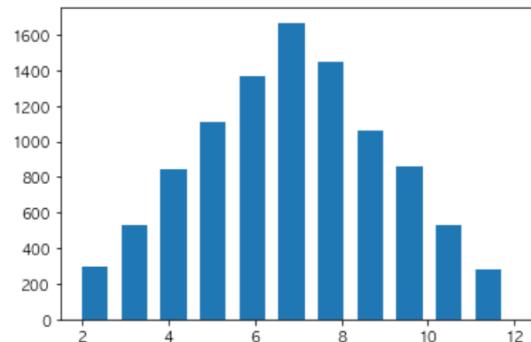
$z = 0$

$n = 2 \ # \text{ number of indices thrown}$

```

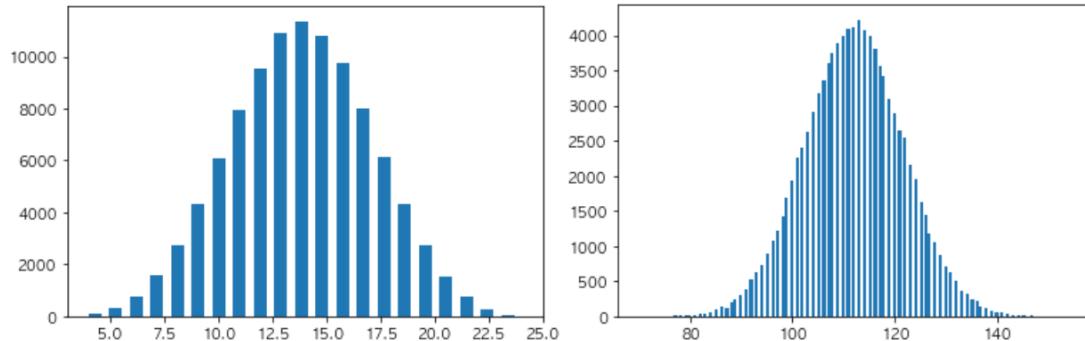
for i in range(n):
    x = np.random.randint(1,7, 10000)
    z = z + x
plt.hist(z,bins = n*5+1, width=0.6)
plt.show()

```



- 만일 주사위를 4개와 32개를 던져서 나오는 값의 합을 측정하면 아래와

같이 정규분포 모습으로 수렴하는 것을 알 수 있다. 주사위 4개의 합은 4~24의 분포를, 32개의 합은 32~192의 분포를 갖는다 (여기서는  $n=32$ 로 하고 10만 번 시행한 결과이다.)



- 표준 정규분포를 따르는 랜덤 숫자를 얻으려면 `randn`을 사용한다. 아래는 평균이 0, 표준편차가 1인 랜덤 숫자 3개를 얻는 코드이다.

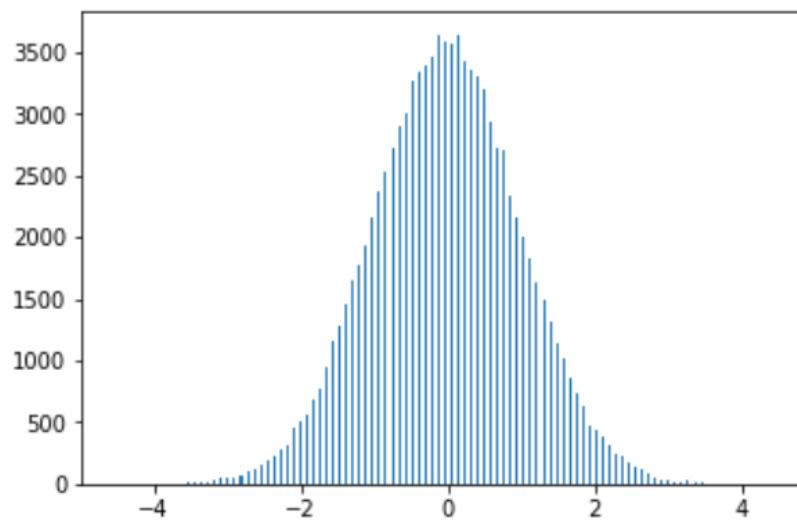
```
np.random.randn(3)
```

=>

```
array([ 0.27768694, -2.79107554,  0.48945396])
```

- 아래는 정규분포를 갖는 10만개의 랜덤 숫자를 발생시키고 이의 분포를 히스토그램을 그린 것이다.

```
x = np.random.randn(100000)  
plt.hist(x, bins = 100, width=0.03)  
plt.show()
```



## 2.3. NumPy

- 여러 항목으로 구성된 데이터를 다루려면 앞에서 배운 파이썬의 리스트나 튜플을 사용할 수 있다. 그러나 다루어야 할 데이터 모두 숫자인 경우는 계산 속도를 개선하기 위해서 NumPy 모듈을 사용한다.
- NumPy는 Numerical Python의 줄임말로서 수학에서 벡터(여러 항목으로 된 1차원 배열)나 매트릭스(2차원, 3차원 등 여러 차원으로 구성된 배열)와 같은 다차원 배열(array)를 계산하는데 유리하다.
- 데이터 분석에서는 숫자 배열을 다루는 경우가 많으며 NumPy 라이브러리가 제공하는 다차원 배열(ndarray: n dimensional array)를 사용하면 앞에서 소개한 리스트보다 계산을 할 때 편리하고 속도도 빠르다.

- 리스트와 달리 배열에서 각 항목은 모두 같은 타입인 숫자(정수나 소수 등)이어야 한다.
- 아래는 numpy를 np라는 이름으로 사용하는 것으로 정하면서 import하는 것을 보였다.
- 배열을 기존의 리스트로부터 만들 수 있는데 아래에서 리스트 list를 만들고 여기에 3을 곱한 결과와 리스트를 배열 arr로 바꾸고 여기에 3을 곱했을 때의 결과를 비교했다.

```
import numpy as np # numpy 라이브러리를 np라는 이름으로 사용  
list = [1, 2, 10]  
print(list * 3)    # [1, 2, 10, 1, 2, 10, 1, 2, 10]
```

```
arr = np.array(list)  
print(arr * 3)      # [ 3  6 30]  
print(arr + 100)     # [101 102 110]
```

- 위의 결과를 보면 리스트에 숫자를 곱하면 동일한 리스트가 숫자 만큼 복사되며, 배열에 숫자를 곱하면 리스트와 달리 모든 항목에 적용되는 것을 알 수 있다.
- 이렇게 하나의 숫자(스칼라)가 배열의 항목 개수만큼 자동으로 확대되는 것을 벡터화라고 한다.
- 벡터화는 곱셈 뿐 아니라 덧셈에도 적용되는데, 위해서 배열에 100을 더하면 배열의 모든 항목에 100이 더해진다. 배열도 [ ]로 묶어서 표시된다.

## 2차원 배열

- 2차원 리스트로부터 2차원 배열을 만드는 예를 아래에 보였다.

```
data = [[1,2,3],[4,5,6],[7,8,9]]  
print(data)      # [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
print(type(data))    # list
```

```
arr2 = np.array(data) # list를 ndarray로 변환  
print(arr2)  
print(type(arr2))    # ndarray
```

```
# 출력  
# [[1 2 3]]
```

```
# [4 5 6]
```

```
# [7 8 9]]
```

- 배열의 특정 위치를 인덱싱하여 얻을 수 있는데 아래는 arr2의 두 번째 행의 세 번째 열의 값인 6을 얻는다. 아래 두 가지 액세스 방법이 동일한 결과를 얻는다.

```
arr2[1,2]      # 6
```

```
arr2[1][2]     # 6
```

- 다차원 배열에서 한 차원이 낮은 배열을 얻으려면 인자 수를 한 개 작게 주면 된다. 예를 들어 아래는 2차원 배열인 arr2에 첫번째 행 또는 세

번째 행을 얻는 것을 보였다.

```
arr2[0]          # array([1, 2, 3])  
arr2[2]          # array([7, 8, 9])
```

- 행의 일부 또는 열의 일부를 얻을 수 있는데 이 때는 슬라이스(slice)를 사용하면 된다. 아래는 행에 소는 2 이전까지 즉, 행 0, 1을 택하고, 열에서는 1 이후 즉, 열 1, 2를 택하는 예를 보였다.

```
arr2[:2,1:]      # array([[2, 3],  
#                  [5, 6]])
```

- 슬라이스를 사용하면 다양하게 배열 각 차원 데이터에 대해 일정한 범위를 선택할 수 있는데 아래에 몇 가지 사용 예를 보였다.
  - [:] 전체
  - [0:n] 0번째부터 n-1번째까지, 즉 n번 항목은 포함하지 않는다.
  - [:5] 0번째부터 4번째까지, 5번은 포함하지 않는다.
  - [2:] 2번째부터 끝까지
  - [-1] 제일 끝에 있는 항목
  - [-2] 제일 끝에서 두번째 항목

## arange

- 일정한 범위의 숫자를 자동으로 생성할 때 파이썬이 기본으로 제공하는 range가 있었다. 이와 유사하게 NumPy가 일정한 숫자의 범위를 제공하는 함수 arange가 있다.
- 0~11 사이의 숫자를 얻고 이를 4x3 크기의 2차원 배열로 재구성하는 것을 아래에 보였다. reshape 함수는 배열의 차원을 생성, 변경할 때 사용한다.

```
data = np.arange(12).reshape(4,3)
data          #array([[ 0,  1,  2],
#                  [ 3,  4,  5],
#                  [ 6,  7,  8],
```

```
# [ 9, 10, 11]])
```

- 이 data 배열에서 항목의 값이 4보다 작은 것을 찾아서 이를 모두 0으로 대체하는 코드를 아래에 보였다.

```
data[data < 4] = 0
```

```
data
```

```
# 출력
```

```
#array([[ 0,  0,  0],  
       [ 0,  0,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11]])
```

- 위의 코드에서 `data < 4` 부분의 동작 결과는 True 또는 False 들로 구성된 2차원 배열을 얻게 되고 이 배열을 `data[ ]`의 인자로 다시 적용함으로써 True에 해당되는 `data` 배열의 항목만 0으로 대체된 새로운 배열을 얻는다.

```
data < 4  
# 출력  
array([[ True,  True,  True],  
       [ True,  True, False],  
       [False, False, False],  
       [False, False, False]])
```

- 2차원 배열에 들어 있는 모든 수의 평균을 구하려면 mean()을 사용한다. 예를 들어 data 배열의 내용을 아래와 같은데, 모든 항목값의 평균은 5.5가 된다.

```
data      #array([[ 0,  1,  2],  
#                 [ 3,  4,  5],  
#                 [ 6,  7,  8],  
#                 [ 9, 10, 11]])
```

```
data.mean()      # 5.5
```

- 배열 전체의 평균이 아니고 각 행 또는 각 열에 대해서 평균을 구하는

방법을 아래에 보였다. `mean()` 함수의 속성값 `axis`를 0으로 지정하면 열을 기준으로 평균 값을 구하고, 1로 지정하면 행을 기준으로 평균값을 구한다. 아래에서 결과 4.5는 1, 3, 6, 9의 평균값이고 5.5는 1, 4, 7, 10의 평균값, 6.5는 2, 5, 8, 11의 평균값이다.

```
data.mean(axis=0)      # array([ 4.5,  5.5,  6.5])  
data.mean(axis=1)      # array([ 1.,   4.,   7.,  10.])
```

## 2.4. Pandas

- 파이썬에서 데이터를 편리하게 다루기 위해 테이블(표) 구조로 데이터를 처리하는 경우가 많다. 이를 위해서 피이썬의 pandas 패키지를 사용하며 이 패키지가 제공하는 데이터프레임(DataFrame)을 이용한다.
- 데이터프레임은 액셀의 스프레드시트와 같은 2차원 테이블 구조로 데이터를 다룬다. pandas는 ‘구조화된 데이터 분석’의 의미인 panel data analysis의 줄임말이다.

## 데이터프레임

- 데이터프레임에는 숫자, 문자열, 불리언 등 임의의 타입의 데이터를 담을 수 있다. 아래는 pandas를 pd라는 이름으로 사용하겠다고 선언하고 DataFrame 함수를 이용하여 딕셔너리 타입의 데이터로부터 데이터프레임을 만드는 예를 보였다.

```
import pandas as pd  
  
import numpy as np  
  
dic = {'city': ['서울', '부산', '대전', '대구', '광주'],  
       'year': [2017, 2017, 2018, 2018, 2018],  
       'temp': [18, 20, 19, 21, 20]}  
  
data = pd.DataFrame(dic)  
  
print(data)
```

==>

	city	temp	year
0	서울	18	2017
1	부산	20	2017
2	대전	19	2018
3	대구	21	2018
4	광주	20	2018

- 데이터프레임을 만들면 인덱싱 번호가 자동으로 부여되면 번호는 0부터 시작한다.
- 컬럼의 배치는 알파벳순으로 정렬되는데, 컬럼의 순서를 바꾸려면 컬럼명을 원하는 순서로 된 리스트로 만들어서 인자로 주면 된다.

```
data[['year', 'city', 'temp']]
```

	year	city	temp
0	2017	서울	18
1	2017	부산	20
2	2018	대전	19
3	2018	대구	21
4	2018	광주	20

- 인덱스를 임의의 이름으로 지정할 수 있다.

```
data.index = ['a','b','c','d','e'] ; data
```

	city	year	temp
<b>a</b>	서울	2017	18
<b>b</b>	부산	2017	20
<b>c</b>	대전	2018	19
<b>d</b>	대구	2018	21
<b>e</b>	광주	2018	20

- 컬럼 이름을 변경할 수 있다.

```
data.columns = ['도시', '연도', '날씨'] ; data
```

	도시	연도	날씨
<b>a</b>	서울	2017	18
<b>b</b>	부산	2017	20
<b>c</b>	대전	2018	19
<b>d</b>	대구	2018	21
<b>e</b>	광주	2018	20

- 데이터프레임에서 특정한 컬럼(열)의 내용만 얻으려면 컬럼 이름을 [ ] 내의 인자로 지정하는 방법이 있고, 또는 컬럼 이름을 속성 값으로 취급하여 “.” 연산을 이용하는 방법도 있다.
  - 컬럼명으로 접근(data['연도'])
  - 속성값으로 접근(data.연도)

- 데이터프레임에서 특정 행(row)을 얻는 방법에는 두 가지가 있다. 먼저 인덱스를 사용하는 방법이 있는데 이때는 loc()을 사용한다.

```
data.loc['b']  
year      2017  
city      부산  
temp      20  
Name: b, dtype: object
```

- 인덱스가 아니라 행의 위치를 지정하는 방법이 있는데 이때는 iloc()을 사용한다. “:”를 사용하여 행의 범위를 지정할 수도 있다.

```
data.iloc[1:3]
```

```
## 출력
```

	year	city	temp
b	2017	부산	20
c	2018	대전	19

- 인덱스를 임의의 컬럼으로 재배정할 수 있다.

```
data.set_index(['도시'], inplace=True) ; data
```

	<b>year</b>	<b>temp</b>
<b>city</b>		
서울	2017	18
부산	2017	20
대전	2018	19
대구	2018	21
광주	2018	20

## 행 출력

```
data.loc['서울']
```

==>

연도 2017

날씨 18

Name: 서울, dtype: int64

```
data.iloc[1:2]
```

==>

연도 날씨

도시

---

부산 2017 20

## 컬럼 추가

- 새로운 컬럼을 추가하려면 현재 없는 컬럼명을 인자로 주면 새로운 컬럼이 자동으로 생성된다.

```
cars = [50,40,20,30,10]
```

```
data['car'] = cars ; data
```

```
==>
```

연도 날씨 car				
도시				
서울	2017	18	50	
부산	2017	20	40	
대전	2018	19	20	
대구	2018	21	30	
광주	2018	20	10	

```
data['high'] = data.car >= 30 ; data
```

	연도	날씨	car	high
도시				
서울	2017	18	50	True
부산	2017	20	40	True
대전	2018	19	20	False
대구	2018	21	30	True
광주	2018	20	10	False

- 특정 조건에 맞는 항목을 찾는 방법을 소개하겠다. 기온(temp)이 20도 이상인 도시를 찾고 이를 구분하기 위해서 high 컬럼을 추가하여 해당 도시를 True로 표시하는 코드를 아래에 보였다.

```
data['high'] = data.temp >= 20
```

```
print(data)
```

==>

	year	city	temp	crime	high	car
a	2017	서울	18	50	False	50
b	2017	부산	20	40	True	40
c	2018	대전	19	20	False	20
d	2018	대구	21	30	True	30
e	2018	광주	20	10	True	10

- 위 코드에서 `data.temp >= 20` 부분은 기온이 20이상인 부분을 찾아 해당 부분을 True로, 해당하지 않는 부분을 False로 구성한 리스트를 만든다. 이 리스트 내용을 새로운 이름의 컬럼 `high`으로 만든 결과이다.

- 특정 컬럼을 삭제하려면 drop을 사용한다.
- drop 함수를 호출 할 때 인자 값으로 0이나 1을 줄 수 있는데 1의 의미는 컬럼(열)을 기준으로 삭제를 하라는 뜻이다.

```
data.drop(['car', 'high'], 1)
```

==>

### 연도 날씨

도시		
서울	2017	18
부산	2017	20
대전	2018	19
대구	2018	21
광주	2018	20

## apply

- 함수를 편리하게 데이터에 일괄 적용하는 방법으로 apply가 있다. 아래는 (최대값-최소값)을 계산하는 함수를 정의하고 이를 데이터에 적용하는 예이다.

```
f = lambda x: x.max() - x.min()
```

```
df = pd.DataFrame(np.arange(12).reshape(4, 3), columns=['A', 'B', 'C'], index=['a', 'b', 'c', 'd'])
```

```
print(df)
```

```
==>
```

	A	B	C
a	0	1	2

```
b 3 4 5  
c 6 7 8  
d 9 10 11
```

```
df.apply(f)
```

```
A 9  
B 9  
C 9
```

```
dtype: int64
```

- 행에 대해서 함수를 적용하려면 (즉, 좌에서 우로 수행) 인자 axis=1로 지정한다.

```
df.apply(f, axis=1)
```

```
##
```

```
a    2
```

```
b    2
```

```
c    2
```

```
d    2
```

```
dtype: int64
```

# Series

- 판다스는 시리즈(Series) 객체를 제공하는데 이를 컬럼이 하나뿐인 데이터프레임이다. 즉, 특수한 구조의 데이터프레임이라고 할 수 있다.

```
region = pd.Series(['서울', '부산', '대전', '대구', '광주'],
index=['1','2','3','4','5'])

print(region)

##

1    서울
2    부산
3    대전
4    대구
5    광주
```

### 3. 데이터 탐색

- 분석할 데이터의 전체적인 특성을 살펴보는 것을 데이터 탐색이라고 한다.
- 데이터 탐색에는 데이터를 시각화하여 그래프로 그려보는 방법이 기본적으로 사용된다.
- 히스토그램, 박스 플롯, 막대그래프, 스캐터 플롯 사용법을 배운다.

## 3.1. 데이터 읽기

### 데이터 타입

- 데이터 타입은 크게 나누어 문자형, 수치형, 바이너리형, 논리형이 있다. 이름, 주소, 텍스트 본문 등은 문자형이고, 온도 습도 등 센서 측정값은 숫자형이다.
- 바이너리형은 오디오, 비디오, 실행파일 등 비트 단위로 구성된 파일이다. 논리형은 True 또는 False등 논리 데이터를 말한다.

문자형: “Hello World”, “대한민국”, ...

수치형: 1, 5, 10, 3.14, 0.9, ...

바이너리형: 0100100101010101...

논리형: True, True, False, True, ...

- 수치형 데이터는 다시 범주형(categorical), 순서형(ordinal), 연속형(continuous)으로 나눌 수 있다. 범주형은 클래스를 구분하는 데이터로 예를 들어 성별, 국가명, 요일, 사람 이름 등을 구분하는데 사용되는 숫자이다.

## 범주형

- 범주형은 문자로도 표현되지만 편의상 숫자로 대체하여 표현하기도 한다. 예를 들어 월요일=1, 화요일=2, 수요일=3 등으로 표현하기도 한다. 또는 남성=1, 여성=0으로 표현하기도 한다.
- 범주형 데이터는 특정 클래스를 지칭하는 것이므로 덧셈이나 뺄셈이 의미가 없다.

## 순서형

- 순서가 의미를 가지는 데이터를 순서형 데이터라고 한다. 여성의 옷 사이즈를 나타내는 44, 55, 66 같은 숫자, 달력의 1일, 2일, 3일 등이 순서형 데이터이다.
- 순서형 데이터에서는 덧셈이나 뺄셈이 의미가 없다. 옷 사이즈 66에서 44를 뺀 22라는 숫자는 순서형으로서 데이터의 의미가 없다.

## 연속형

- 연속형 데이터는 무게, 길이, 온도, 압력, 속도, 화폐 단위와 같이 숫자의 양이 어떤 의미를 가지는 데이터를 말한다. 연속형 데이터는 덧셈과 뺄셈의 결과가 계속 같은 연속형 데이터로서 의미를 갖는다. 예를 들어 기온 3도에

4도를 더한 7도는 여전히 온도로서 의미가 있다.

## 정형과 비정형 데이터

- 데이터를 다른 측면에서 정형(structured), 비정형(unstructured), 그리고 반정형(semi-structured)의 데이터로 구분한다.
- 정형 데이터란 데이터의 포맷이 정해져 있는 데이터를 말한다. 예를 들어 액셀 서식으로 저장된 데이터가 대표적인 정형 데이터이다. 대부분의 통계 데이터, 기업의 매출기록 등 서식이 정해진 데이터는 정형 데이터이다.
- 비정형 데이터란 정형 데이터와 달리 일정한 포맷을 가지지 않은 데이터를 말한다. 블로그 글, 트윗 등 텍스트 문자와, 오디오나 비디오 데이터는 비정형 데이터이다.

- 반정형 데이터란 데이터 포맷이 정형 데이터처럼 완전하게 정의되어 있지는 않지만 조금만 처리를 하면 정형화된 정보를 추출할 수 있는 데이터를 말한다.
- 주기적으로 측정되는 센서데이터, 시간대별로 측정한 웹 사용자의 기록 등이 해당된다.
- 컴퓨터는 궁극적으로 숫자로 표현된 데이터만 처리할 수 있다. 텍스트, 오디오, 비디오 등의 데이터는 숫자로 표현을 바꾸어야만 처리할 수 있다.

## 웹 데이터 받기

- 데이터 분석의 시작은 어디선가 데이터를 가져오는 작업에서 시작한다. USB 메모리 드라이브로 파일을 복사하기도 하고 웹에서 데이터를 다운로드 받거나, 웹 크롤링으로 데이터를 직접 수집하기도 한다.
- 먼저 웹에서 파일을 다운로드하는 경우를 소개하겠다.

## 전력판매 데이터

- 아래 주소에서 국내 전력 판매 데이터를 다운로드받을 수 있다.  
전력판매량(시도별/용도별) 액셀 파일 <https://goo.gl/Cx8Rzw>
- 위 데이터는 전력거래소(<http://kpx.or.kr>)에서 받은 자료로 전국 광역

지자체별로 주거용, 공공용, 서비스업 등 용도별 전력 사용 데이터를 제공한다 (전력거래소 – EPSIS 전력통계정보 – 판매 – 시도별 용도별 – 수치정보 보기 – EXCEL 메뉴).

- 다운 받은 파일을 현재 작업중인 (쥬피터 노트북이 실행되는) 폴더 아래에 data 폴더를 만들고 data 폴더로 복사한다.

## 3.2. 탐색적 분석

### 정의

- 데이터 탐색이란 본격적인 데이터 분석이나 머신러닝을 수행하기에 앞서 데이터의 전체적인 특성을 살펴보는 것을 말한다. 데이터 탐색 자체도 중요한 데이터 분석의 하나이며 이를 탐색적 데이터 분석 (exploratory data analysis: EDA)라고 한다.
- 데이터 탐색과정에서 주로 시각화(visualization)을 사용한다. 본격적인 데이터 분석에 앞서 수집한 데이터가 분석에 적절한지 알아보는 과정이다.
- 기본적인 통계적 특성 파악하며 숫자형 데이터의 평균, 최대값, 최소값, 표준편차, 분산 등을 알아본다. 탐색적 분석에는 시각화 도구 이용한다.

## 데이터프레임으로 읽기

- 파이썬에서 데이터 탐색을 위해서 데이터를 데이터프레임에 데이터를 담고 분석하면 편리하다. 판다스가 제공하는 read 관련 함수를 사용하면 파일을 데이터프레임 형태로 바로 변환하면서 읽을 수 있다.
- 아래에 다운로드 받은 엑셀 형태의 파일을 read\_excel 함수로 읽어 데이터 프레임 power\_data에 저장하고 데이터의 모양(shape)를 확인했다.

```
power_data = pd.read_excel('data/시도별_용도별.xls')
```

```
print(power_data.shape)
```

=> (19, 28)

- 위의 결과를 보면 power\_data 데이터 프레임의 구조가 19행, 28열로 된 것을 알 수 있다. power\_data의 상위 5개의 행을 출력하면 아래와 같다 (아래 결과 화면을 보면 총 28개의 열이 있는데 전체가 보이지는 않는다).

power\_data.head(5)

구분	주거용	공공용	서비스업	업무용합계	농림어업	광업	제조업	식료품제조	섬유,의류	...
0 강원	1940933	1400421	6203749	7604170	607139	398287	6002286	546621	13027	...
1 개성	0	0	0	0	0	0	0	0	0	...
2 경기	16587710	5533662	33434551	38968213	2371347	317263	56603327	2544420	2109963	...
3 경남	4260988	1427560	8667737	10095297	2141813	95989	18053778	932743	346974	...
4 경북	3302463	1578115	8487402	10065517	1747462	224568	30115601	566071	3780171	...

- 데이터프레임에서 마지막의 n개의 행을 출력하려면 tail(n)함수를 사용하면 된다.
- 위 결과를 보면 좌측에, 0, 1, 2, ..의 인덱스 번호가 자동으로 생성된 것을 알 수 있다. 데이터프레임에는 이와 같이 인덱스가 자동으로 만들어진다.
- 먼저 어떤 열이 있는지 확인하려면 데이터프레임의 column 속성을 보면 된다. 아래에 power\_data의 총 28개 열의 이름을 모두 출력했다. (출력을 =>로 표시하겠다)

```
power_data.columns  
=>
```

```
Index(['구분', '주거용', '공공용', '서비스업', '업무용합계', '농림어업', '광업', '제조업', '식료품제조', '섬유,의류', '목재,나무', '펄프,종이', '출판,인쇄', '석유,화학', '의료,광학', '요업', '차금속', '조립금속', '기타기계', '사무기기', '전기기기', '영상,음향', '자동차', '기타운송', '가구및기타', '재생재료', '산업용합계', '합계'],dtype='object')
```

## 인덱스 변경

- power\_data의 인덱스를 자동으로 배정된 0, 1, 2, ... 값이 아니라 광역지자체 이름(위에서 ‘구분’ 컬럼)으로 변경하고 이 데이터프레임 이름을 power로 변경하여 사용하겠다.

```
power = power_data.set_index('구분')
power.head(5)
```

구분	주거용	공공용	서비스업	업무용합계	농림어업	광업	제조업	식료품제조	섬유,의류	목재,나무	...
강원	1940933	1400421	6203749	7604170	607139	398287	6002286	546621	13027	19147	...
개성	0	0	0	0	0	0	0	0	0	0	...
경기	16587710	5533662	33434551	38968213	2371347	317263	56603327	2544420	2109963	529274	...
경남	4260988	1427560	8667737	10095297	2141813	95989	18053778	932743	346974	60160	...
경북	3302463	1578115	8487402	10065517	1747462	224568	30115601	566071	3780171	72680	...

- 위 결과를 보면 이제 인덱스가 0,1,2,3,...에서 자체 이름으로 변경된 것을 알 수 있다.
- 데이터 프레임의 헤더와 인덱스를 제거하고 내용만 보려면 values 속성을 보면 된다.

```
power_data.values
```

```
=>
```

```
array([[1940933, 1400421, 6203749, 7604170, 607139, 398287, 6002286, 546621,  
13027, 19147, 24382, 7727, 175323, 84397,  
3695776, 1038913, 39477, 35063, 2019, 38062, 43986, 113448,  
108629, 12872, 3418, 7007712, 16552816],  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0],  
[16587710, 5533662, 33434551, 38968213, 2371347, 317263,
```

```
...
```

## 헤더 옵션

- 엑셀 파일의 상위 몇 개의 행을 제거하고 읽을 필요가 있다. 이 때는 `read_excel` 함수의 인자로 `headers`를 사용하면 된다. `headers=2`로 하면 상위 두 개의 행을 제거하고 읽는다.

## csv 파일

- 엑셀 파일이 아니라 콤마로 항목들이 구분된 파일(csv 파일이라고 한다)을 읽으려면 `read_csv`를 사용한다. 구분자가 콤마가 아닌 경우 (탭이나 스페이스 등) `read_csv` 호출할 때 구분자(delimiter)를 지정할 수 있다.

## 행 삭제

- 위의 power 데이터를 보면 개성 데이터는 들어있지 않다. 그리고 맨 아래 행인 ‘합계’는 전체의 합을 나타내므로 이 두 행을 삭제하고 데이터를 분석해보겠다.
- 행을 제거하려면 drop()을 사용하고 인자로 행을 입력하면 된다. 여러개를 제거하려면 리스트를 사용한다.

```
power = power.drop(['개성', '합계'], errors='ignore')
```

- 위에서 errors 옵션은 혹시 drop() 수행중에 오류가 발생하여도 무시하라는 뜻이다.

## info()

- 데이터프레임의 각 열에 대한 정보를 보려면 info()를 사용한다.

power.info()

=>

```
<class 'pandas.core.frame.DataFrame'>
```

Index: 17 entries, 강원 to 충북

Data columns (total 27 columns):

주거용 17 non-null int64

공공용 17 non-null int64

서비스업 17 non-null int64

...

# describe()

- 각 열에 대해서 최소치, 최대치, 평균, 분산 등 기초통계 정보를 보려면 describe()를 사용한다.

power.describe()

	주거용	공공용	서비스업	업무용합계	농림어업	광업	제조업
<b>count</b>	1.700000e+01	1.700000e+01	1.700000e+01	1.700000e+01	1.700000e+01	17.000000	1.700000e+01
<b>mean</b>	3.912786e+06	1.388502e+06	8.291312e+06	9.679813e+06	9.400665e+05	102720.176471	1.523205e+07
<b>std</b>	4.310227e+06	1.298531e+06	8.708909e+06	9.981623e+06	1.010972e+06	122502.337918	1.524290e+07
<b>min</b>	3.849030e+05	2.996750e+05	6.454240e+05	9.450990e+05	1.515000e+04	2898.000000	2.415370e+05
<b>25%</b>	1.940933e+06	8.263960e+05	3.955921e+06	4.910602e+06	7.460800e+04	14019.000000	2.910768e+06
<b>50%</b>	2.326183e+06	1.089613e+06	5.690659e+06	6.654683e+06	6.071390e+05	71529.000000	1.236782e+07
<b>75%</b>	3.856852e+06	1.400421e+06	7.582169e+06	8.888045e+06	1.747462e+06	139856.000000	2.145393e+07
<b>max</b>	1.658771e+07	5.533662e+06	3.343455e+07	3.896821e+07	3.096126e+06	398287.000000	5.660333e+07

- 위에서 min과 max는 각각 최소값과 최대값을 나타낸다. 50%은 중앙값(median)을 나타내고 mean은 평균값을 나타낸다.
- 중앙값은 전체 데이터를 크기순으로 배열했을 때 중간에 있는 값을 말한다. 예를 들어 전체 항목 수가 101개이면 51번째 데이터 값이 중앙값이다.
- 전체 항목수가 짝수이면 메디안은 가운데에 있는 두 개의 평균을 알려준다. 예를 들어 전체 항목 수가 100개이면 50번째와 51번째 데이터 값의 평균치가 중앙값이다.
- 25%와 75%는 각각 1사분위(first quartile) 값과 3사분위(third quartile) 값을 나타낸다. 1사분위란 자료의 전체 값 중에서 앞에서부터 25%에 해당하는 값을, 3사분위는 앞에서부터 75% 위치의 값이다.

## **counter()**

- 빈도수를 구해서 사전형식으로 만들어 준다.

## **corr()**

- 변수간의 상관관계 계수를 구한다. 1에 가까울수록 변수 간에 양의 상관관계를 가지며 -1에 가까울수록 음의 상관관계를 가진다.
- 상관관계의 타입을 method 옵션으로 선택할 수 있는데 pearson, kendall, spearman 등이 널리 사용된다.

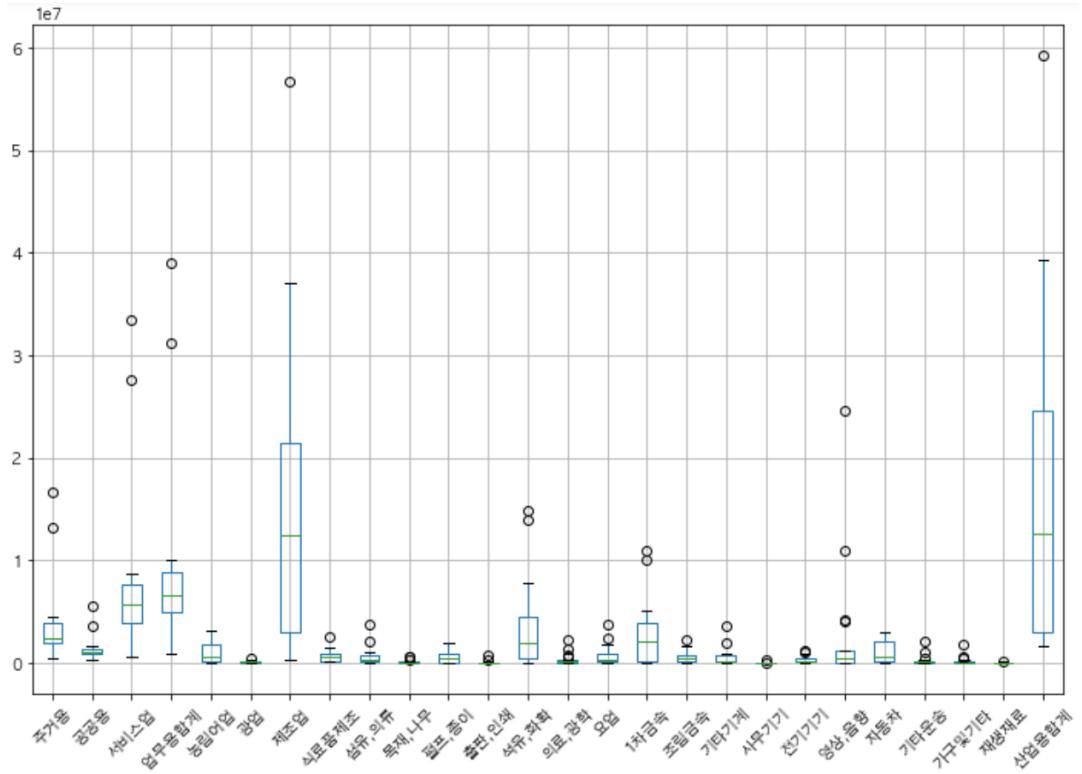
### 3.3. 시각화

- 데이터의 특성을 파악하기 위해서 박스플롯, 바차트, 히스토그램, 스캐터 플롯 등을 자주 사용한다.
- 시각화를 하는 목적은 데이터 내재되어 있는 의미를 찾아낼 수 있고 데이터 탐색 뿐만 아니라 분석 결과를 고객에게 설명할 때에도 필요하다.

## 박스 플롯

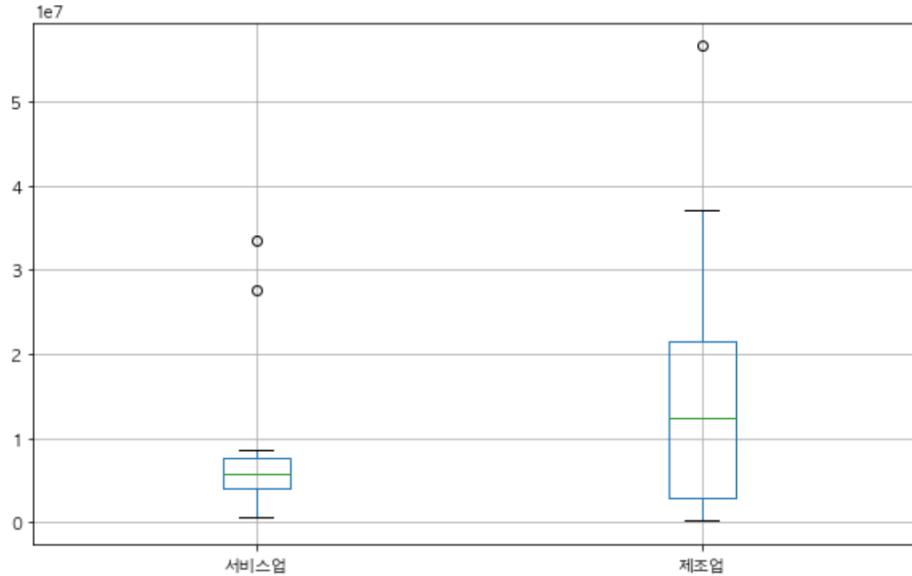
- 수치형 데이터의 분포를 한눈에 볼 때 사용된다. `boxplot()` 함수로 그릴 수 있다. 열에서 전체 전력량의 ‘합계’ 컬럼을 제거하고 `power` 데이터프레임 전체 열에 대한 박스 플롯을 그린 것이다.

```
power.drop('합계', axis=1).boxplot(figsize=(10,6))  
plt.xticks(rotation=45)
```



- 서비스업과 제조업 전기 판매량에 대해서만 박스플롯을 그리면 다음과 같다.

```
power[['서비스업','제조업']].boxplot(figsize=(10,6))
```



- 위 박스 플롯에서 가로방향 실선의 의미는 위에서부터 최대치, 3사분위 값(75%), 평균값, 1사분위 값(25%), 그리고 최소치를 나타낸다. 3사분위값

과 1사분위 값의 차이를 IQR이라고 부른다.

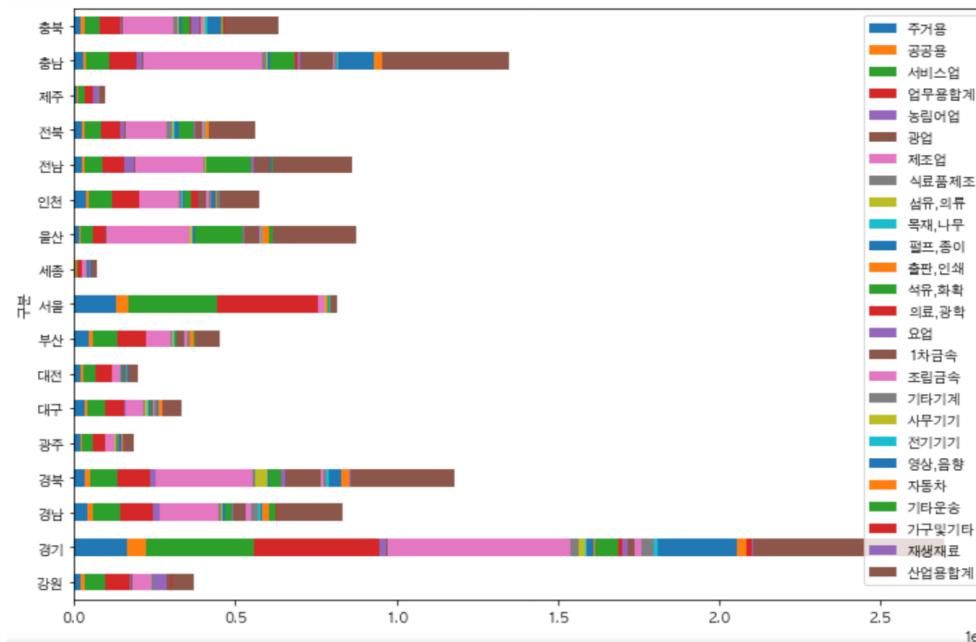
- 박스플롯 위 또는 아래에 나타나는 ‘o’ 기호는 이상치(outlier)를 표시한다. 이상치로 표시하는 기준은 보통 1사분위 값보다  $1.5 \times \text{IQR}$  이상 더 작거나, 또는 3사분위 값보다  $1.5 \times \text{IQR}$  이상 더 큰 값을 기준으로 삼는다.
- 즉, 다른 값들에 비해 너무 작거나 너무 큰 값을 이상치로 보는 것이다.

# 바플롯

- power 열에서 ‘합계’ 컬럼을 삭제하고 power 의 바 플롯을 그려보겠다.

```
power = power.drop('합계', axis=1)
```

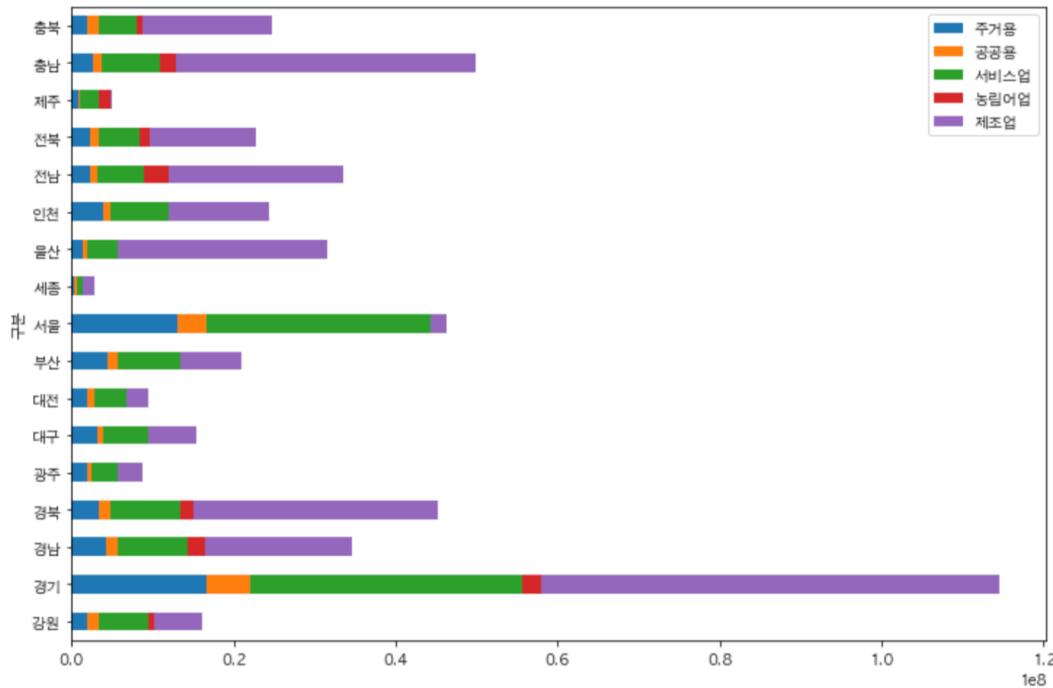
```
power.plot(kind='barh', figsize=(12,8), stacked=True)
```



- 임의로 5 개 열에 대해서만 바플롯을 그려보겠다.

```
sample = ['주거용', '공공용', '서비스업', '농림어업', '제조업']
```

```
power[sample].plot(kind='barh', figsize=(12,8), stacked=True)
```



## 산포도

- 산포도(scatter plot)이란 두 개의 변수(특성) 관계를 점으로 나타내는 그래프이다. 여기서는 서비스업과 제조업 두 개의 특성 관계를 산포도로 그려보겠다.
- 먼저 두 개의 열로만 구성된 데이터프레임 power를 아래와 같이 만들었다.

```
power = power[['서비스업', '제조업']]  
power.head(5)
```

	서비스업	제조업
구분		
강원	6203749	6002286
경기	33434551	56603327
경남	8667737	18053778
경북	8487402	30115601
광주	3174973	2910768

```

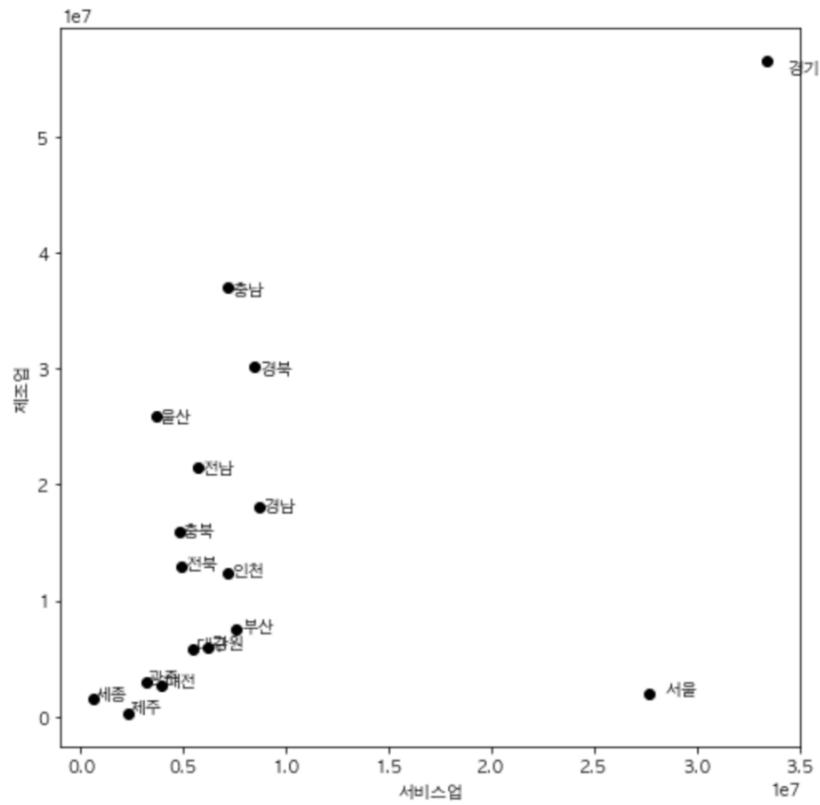
plt.figure(figsize=(8,8))

plt.scatter(power['서비스업'], power['제조업'], c='k', marker='o')
plt.xlabel('서비스업')
plt.ylabel('제조업')

for n in range(power.shape[0]):

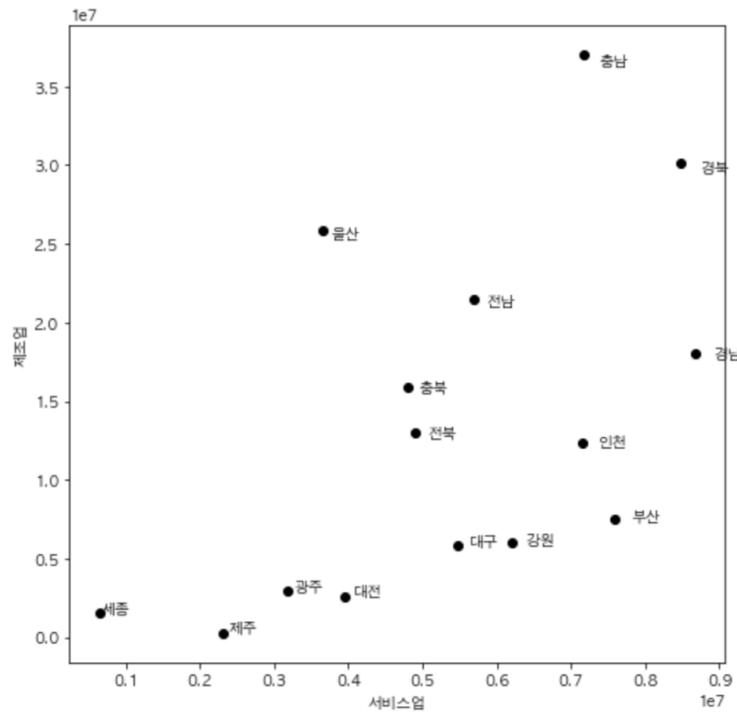
    plt.text(power['서비스업'][n]*1.03, power['제조업'][n]*0.98, power.index[n])

```



- 경기와 서울의 전력사용량이 큰 것을 알 수 있다. 이 두 지역의 데이터를 삭제하고 다른 지자체를 좀 더 상세히 비교하겠다.

```
power = power.drop(['경기', '서울'])
```



# 기타 시각화 함수

## 산포도 매트릭스

- 여러 변수간의 산점도를 한번에 그리기 위해서 산포도 매트릭스를 사용한다. 산포도 매트릭스는 scatter\_matrix()를 사용한다.

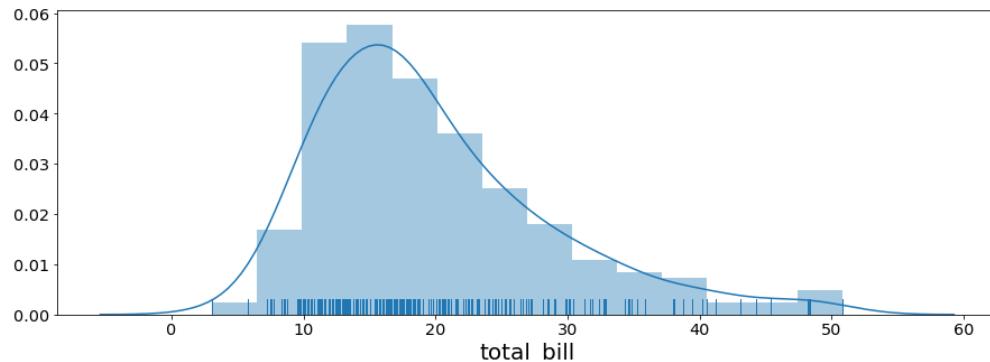
## seaborn

- 시본(seaborn)은 Matplotlib의 기능을 확장하여 다양한 색상과 표현을 추가로 제공하는 시각화 패키지이다.
- 시본은 pip install seaborn 으로 설치할 수 있다. 시각화할 데이터는 seaborn에서 제공하는 tip 데이터를 활용한다.

## distplot()

- 히스토그램을 그려주며 kde 옵션 선택으로 밀도그래프와 rug 옵션선택으로 데이터의 분포 위치를 표시해 주는 기능을 설정할 수 있다.

```
data = sns.load_dataset("tips")
sns.distplot(data['total_bill'], kde=True, rug=True)
plt.show()
```

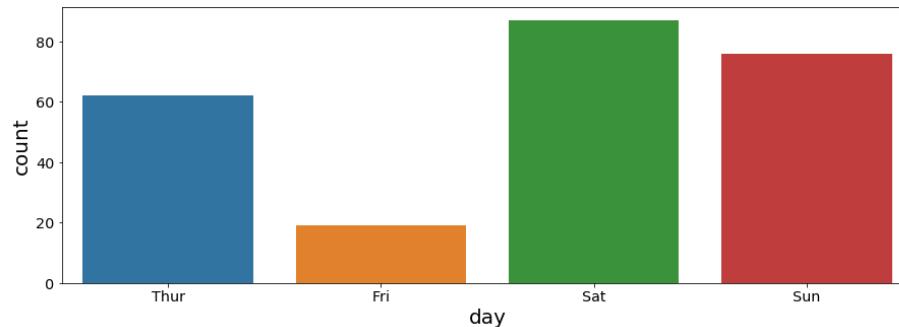


## countplot()

- 각 카테고리 별로 데이터 분포를 알려준다. data 인자에는 데이터가 들어있는 데이터프레임을 지정하고 x에는 카테고리 컬럼 이름을 입력한다.

```
sns.countplot(x='day', data=data)
```

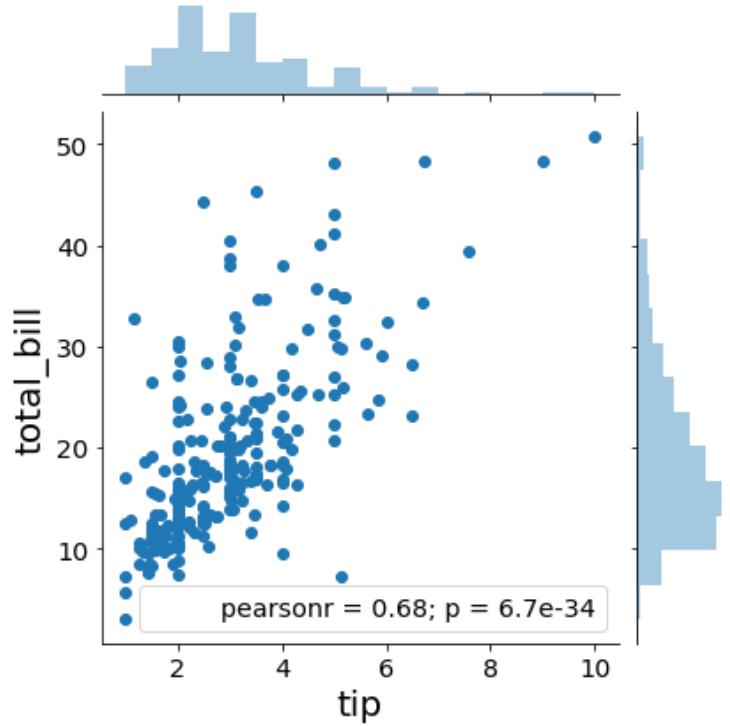
```
plt.show()
```



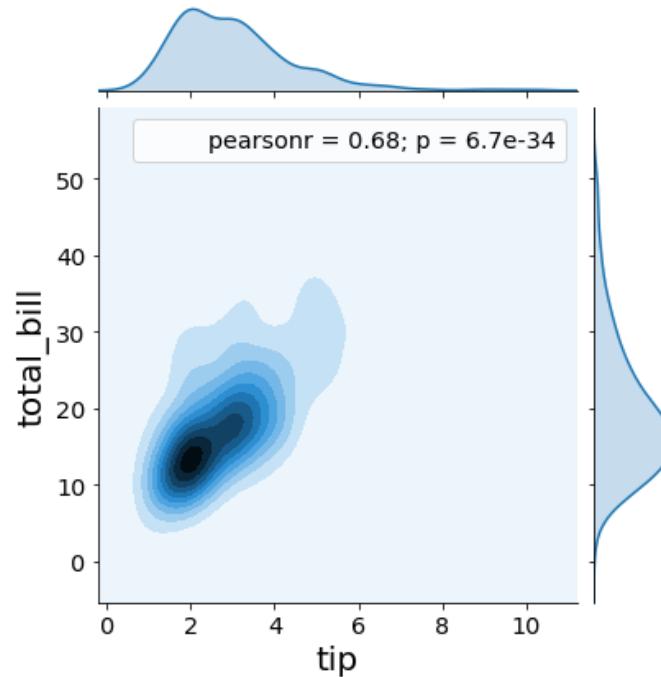
## jointplot()

- 산포도(스캐터 플롯)을 그려준다. jointplot 을 사용하면 가장자리에 각 변수의 히스토그램을 추가로 그려준다. x, y에 컬럼 명을 지정해 두 컬럼 간의 상관관계를 살펴볼 수 있다. 또한, kind=kde 옵션을 선택하면 커널 밀도 히스토그램을 그릴 수 있다.

```
sns.jointplot(x="tip", y="total_bill", data=data)  
plt.show()
```



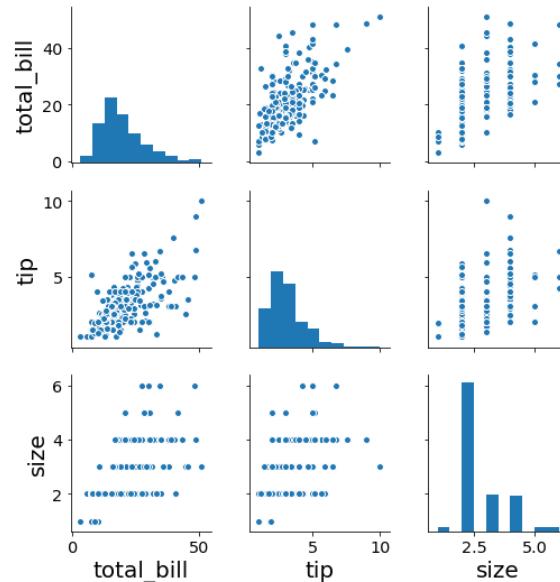
```
sns.jointplot(x="tip", y="total_bill", data=data, kind="kde")
plt.show()
```



## pairplot()

- 여러 변수간의 산포도를 매트릭스 형태로 그려준다. matplotlib의 scatter\_matrix()와 같은 그래프를 그려준다.

```
sns.pairplot(data)
```



## 히트맵

- 히트맵(heatmap)은 수치를 색상의 진한 정도로 표현한다. pivot\_table을 사용하면 피벗테이블을 생성할 수 있는데 아래는 요일별 팀을 준 사람의 성별 수를 나타낸 피벗 테이블이다.

```
data = data.pivot_table(index="day", columns="sex", aggfunc="size")
print(data)
##
sex      Male   Female
day
Thur      30      32
Fri       10       9
```

Sat 59 28

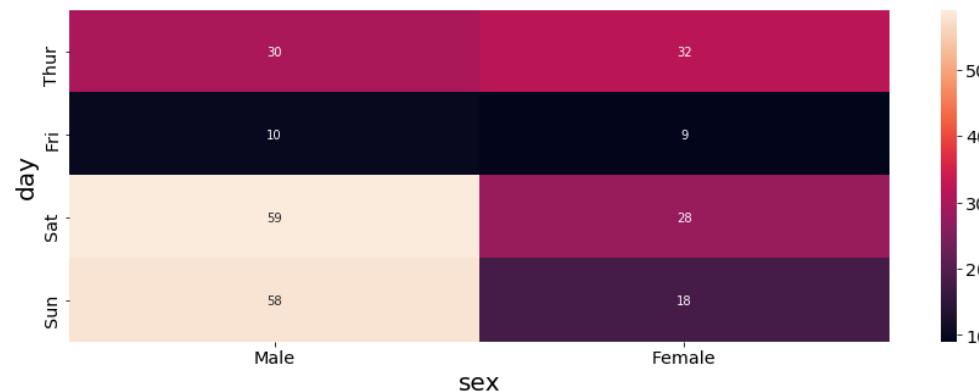
Sun 58 18

- 위 정보를 이용해 히트맵을 다음과 같이 그릴 수 있다.

```
sns.heatmap(data, annot=True, fmt="d")
```

```
plt.show()
```

```
##
```



## 4. 데이터 전처리

- 데이터 전처리(preprocessing)란 데이터를 분석에 사용할 때 성능이 더 좋게 나오도록 데이터를 수정하거나 형태를 변형하는 것을 말한다.
- 수집한 데이터를 머신러닝 등에 바로 사용할 수 있는 경우는 거의 없다. 수집한 데이터가 너무 크면 이를 한 번에 분석하기 어려우므로 적절한 크기로 줄여야 한다.
- 데이터가 비정형 이라면 이를 정형 데이터로 바꾸어야 한다. 이미지나 텍스트와 같은 비정형 데이터의 의미를 컴퓨터가 바로 다룰 수는 없다.

## 4.1. 전처리 타입

- 데이터를 사용하려면 중간에 데이터가 빠졌거나, 틀린 값이 들어 있거나, 데이터의 단위가 틀릴 수가 있다(m와 인치, kg과 파운드 등 다른 단위가 혼재). 범주형 데이터의 경우 카테고리를 나타내는 표현으로 바꾸기도 한다.
- 빠진 값을 채워 넣거나 오류를 수정하는 것을 데이터 정제(cleaning)이라고도 한다.
- 데이터를 변형하는 것은 로그를 적용하거나, 역수를 취하거나 정규분포에 맞게 구간을 설정하기도 한다.
- 컴퓨터는 단어의 개념을 수치나 범주형 데이터로 환산해 주어야 처리할

수 있다.

- 예를 들어 월요일은 1로, 화요일을 2로 코딩할 수 있다. 또는 수치 데이터의 분포를 정규화(normalize)하기도 한다.
- 예를 들어 10점 만점으로 처리한 것과 100점 만점으로 처리한 데이터를 같이 활용하려면 동일한 분포로 바꾸어야 한다.

## 결측치 처리

- 데이터 전처리에서 가장 중요한 과정은 빠진 값 즉, 결측치(missing value)를 처리하는 것이다.
- 결측치를 처리하는 방법은 크게 세가지가 있다.
- 결측치가 포함되어 있는 샘플 항목을 모두 버리거나, 결측치를 적절한 다른 값으로 대체하거나, 아니면 결측치를 NA(not available)라고 표시하여 다음의 데이터 처리 단계로 그대로 넘겨주는 방법도 있다.

결측치가 포함된 샘플 항목을 모두 버리는 방법

결측치를 적절한 값으로 대체

분석 단계로 결측치 처리를 넘김

- 결측치가 들어 있는 항목을 삭제하더라도 크게 문제가 되지 않으면 결측치를 버리는 방법이 가장 처리하기가 쉽다. 그런데 결측치가 들어 있는 항목의 비중이 크면 이를 무시할 수 없다.
- 예를 들어 데이터가 1백만 건이 있는데 이중 100건 정도에서 결측치가 발생했다면 이들을 모두 제외해도 분석에 큰 영향이 없을 것이다. 그런데 1000개 샘플 중에 100개 데이터에서 문제가 있다면 삭제할 수 없다.
- 단순히 실수로 값을 입력하지 않은 것인지, 어떤 오류에 의해 값이 빠진 것인지를 파악해야 한다.
- 결측치를 다른 값으로 대체하는 경우에는 어떤 값으로 대체할지를 선택해야 한다.

- 대체값으로 전체 평균값을 사용할 수도 있고, 인접한 값으로 추정치를 계산하는 수도 있다. 숫자이면 0을 채택할 수도 있고, 미리 정한 최소값으로 정해줄 수도 있다.
- 결측치를 대체할 때는 분석 결과가 달라질 수 있는 것에 항상 주의를 해야 한다.
- 결측치를 다른 값으로 대체한 경우 이러한 사실을 표시하는 별도의 범주형 변수를 새로 정의하는 방법이 편리하다. 나중에 이 변수를 보고 결측값이 있었고 다른 값으로 대체되었다는 사실을 추적할 수 있다.
- 결측치의 처리를 분석 단계로 넘긴다는 것은 전처리 단계에서는 아무 조치도 취하지 않는 것을 말한다. 데이터가 빠져있으면 없는대로 둔다는 뜻이다.

- 이러한 경우는 결측치 처리를 미리 일괄적으로 할 것이 아니라, 다음 처리 단계로 넘기는 것이 안전할 수 있다.
- 아래는 결측치를 처리하는 코드 예이다. 결측치는 파이썬에 읽을 때 NA로 읽힌다. 아래는 (3, 4) 크기의, 정규분포를 갖는 랜덤 숫자를 발생시켰다.

```
from numpy import nan as NA
from pandas import DataFrame
df = pd.DataFrame(np.random.randn(3, 4))
print(df)
##
0          1          2          3
0  0.124076 -1.791666  0.569558  1.055151
```

```
1 0.302909 -1.750992 -0.830812 -2.060737  
2 1.406866  0.127084 -0.822918 -1.378994
```

- 강제 df[1][2] 위치의 값을 결측치로 변경했다. np.nan은 값이 존재하지 않는다는 사실 즉 결측치인 것을 나타낸다 (nan는 not a number를 나타냄)

```
df[1][2] = np.nan  
print(df)  
##  
          0           1           2           3  
0 0.124076 -1.791666  0.569558  1.055151  
1 0.302909 -1.750992 -0.830812 -2.060737
```

```
2 1.406866      NaN -0.822918 -1.378994
```

- `NaN`이 하나라도 들어 있는 행을 모두 제거 하려면 `dropna()`를 사용한다.

```
cleaned = df.dropna()  
print(cleaned)  
##  
  
          0           1           2           3  
0 0.124076 -1.791666  0.569558  1.055151  
1 0.302909 -1.750992 -0.830812 -2.060737
```

- 결측치를 다른 값으로 대체하는 방법이 여러 가지가 있는데 우선 0으로 채우려면 `fillna(0)`을 사용한다.

```
df_2=df.fillna(0)
print(df_2)
##
```

	0	1	2	3
0	0.124076	-1.791666	0.569558	1.055151
1	0.302909	-1.750992	-0.830812	-2.060737
2	1.406866	0.000000	-0.822918	-1.378994

- 결측치를 평균값으로 채우려면 아래와 같이 평균값을 계산해서 인자로 넣어주면 된다.

```
data2 = data.fillna(data.mean())
```

- 여기서 평균을 구하는 방법이 행에 대한 평균 또는 열에 대한 평균으로 구분할 수 있다. 이 때에는 `mean(0)`, 또는 `mean(1)`과 같이 열 기준 또는 행 기준을 정해주어야 한다.
- 열을 기준으로 연산을 수행할 때 인자를 0으로 하고, 행을 기준으로 수행할 때 인자를 1로 사용한다. 디폴트로는 열을 기준으로 평균을 구한다.
- 결측치를 바로 앞에(forward) 있는 값으로 대신 채우려면 아래와 같이 `method` 인자를 ‘`ffill`’로 설정하면 된다. (다시 결측치를 강제로 만들고 실행한다). 아래에 보면 결측치였던 `df[1][2]` 값이 그 앞의 값인 `-1.750992`로 변경된 것을 알 수 있다.

```
df[1][2] = np.nan  
df_2=df.fillna(method='ffill')  
print(df_2)  
##  
  
0           1           2           3  
0  0.124076 -1.791666  0.569558  1.055151  
1  0.302909 -1.750992 -0.830812 -2.060737  
2  1.406866 -1.750992 10.000000 -1.378994
```

## 틀린 값 처리

- 틀린(invalid) 데이터란 확실히 잘 못된 값이 들어있는 것을 말한다. 양수여야 하는데 음수가 있거나, 숫자가 있어야 할 곳에 문자열이 있는 경우를 말한다.
- 예를 들어 키를 나타내는데 3.7m가 입력되었다면 이는 틀린 데이터이다. 틀린값을 처리하는 방법도 크게 나누면 결측치를 처리하는 방법과 같이 세가지이다.

틀린 값이 포함된 항목을 모두 버리는 방법

틀린 값을 적절한 값으로 대체

분석 단계로 틀린 값 처리를 넘김

- 결측치는 값이 없는 것이므로 명백하게 발견할 수가 있다. 그러나 틀린값은 바로 발견할 수가 없고 프로그램에 의해서 찾아내야 한다. 값이 들어 있기는 한데 틀린 것이므로 명확한 기준에 따라 찾아내야 한다.

## 이상치 처리

- 이상치(outlier)란 값의 범위가 일반적인 범위를 벗어나 특별한 값을 갖는 것을 말한다. 이상치와 틀린 값은 다르다. 예를 들어 학교에서 키를 측정했는데 3.7m가 나오면 틀린 값이지만, 2.0m가 나왔다면 이상치이다. 2.0m인 학생은 극히 드물지만 있을 수 있는 값이기 때문이다.
- 신용카드 내역을 조사하는데 월평균 100만원 쓰는 사람이 어느 달에 2천만 원을 썼으면 이는 이상치이다.
- 이상치를 찾아내는 것을 이상치 검출(detection)이라고 하는데 이는 도난당한 카드의 사용을 찾아내거나, 불법으로 보험료를 청구하는 것을 찾거나, 기계가 이상한 동작을 하는 것을 찾는데 사용된다. 즉, 이상치 검출은 데이터 전처리에서 다루는 것이 아니며, 데이터 분석 단계로 넘겨야 한다.

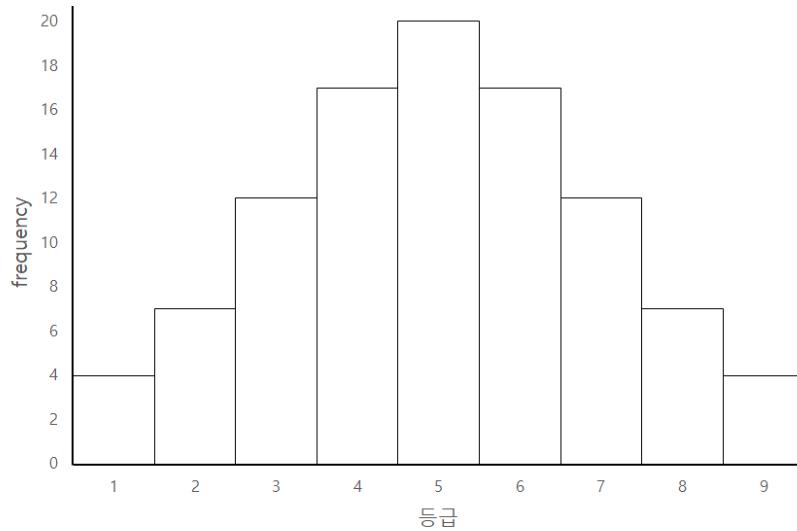
## 4.2. 데이터 변환

- 데이터를 주어진 그대로 사용하지 않고 다른 형태로 바꾸어 사용하는 것이 필요한 경우가 많다.
- 같은 성적을 나타내는데 A, B, C 등 학점으로 표현하거나 100점 만점으로 환산하기도 한다 (97, 94, 91 등).
- 범주형 변환, 로그변환, 역수변환 등을 설명한다.

## 범주형으로 변환

- 수치 데이터를 범주형으로 변환하여 사용하는 경우가 많다. 나이를 구간으로 나누는 경우, 10대, 20대, 30대, 40대 등으로 나눌 수 있다.
- 이렇게 하는 이유는 정확한 나이를 알 필요가 없고 연령대만 알아도 충분하기 때문이다. 너무 세세하게 구분하는 것이 오히려 혼란스러울 수가 있다. 연간 소득을 고소득층, 중간층, 저소득층으로 나누기도 한다.
- 수치형 데이터를 범주형으로 변환할 때 각 구간의 범위를 균등하게 정할 수도 있고 서로 다른 범위를 정할 수도 있다.
- 예를 들어 고등학교 내신 성적을 9등급으로 나누는 경우에는 균등하지 않은 범위를 사용한다. 아래 그림은 내신성적 등급 분포를 나타냈다. 이

그림을 보면 1등급과 9등급은 각각 4%, 2와 8 등급은 15% 등으로 등급 내의 분량이 다르다.



고등학교 내신 성적의 1~9 등급의 히스토그램

- 구간의 크기를 균등하게 배정하는 방법과 균등하지 않게 배정하는 것 중 어떤 것이 적합한지는 분석 목적에 따라 다르다.
- 학생의 실력을 나타내는 등급의 경우는 석차에 따라 균등하게 배분하는 것보다 정규 분포와 유사하게 나누는 것이 더 자연스럽다고 본 것이다.
- 여기서 자연스럽다고 하는 것은 사람이 느끼는 실력 차이가 등급 수치를 보고 느끼는 것과 자연스럽게 일치한다는 뜻이다.
- 예를 들어 6등급과 4등급 학생의 실력 차이와 4등급과 2등급 학생의 실력 차이가 같게 느껴져야 한다.
- 범주를 잘 나누는 기준은 최종적으로 이러한 데이터를 사용한 머신러닝 모델이 잘 동작하는지를 보고 판단해야 한다.

## 범주형 변수 코딩

- 예를 들어 요일을 1, 2, 3, 4, 5, 6, 7 등으로 표시한 경우 이 변수를 컴퓨터가 연산(덧셈이나 곱셈)을 할 수 있는 숫자로 인식해서는 안 된다. 이 숫자를 범주형(카테고리형)으로 분명하게 처리되어야 한다.
- 컴퓨터가 범주형(카테고리형) 변수를 분명히 인식하게 하는 방법의 하나로 하나의 특성(컬럼)만 1이 될 수 있고 다른 특성은 모두 0으로 코딩하는 방법을 사용할 수 있는데 이를 원핫인코딩(one hot encoding)이라고 한다.
- 데이터프레임 형식이면 판다스가 제공하는 `get_dummies()`를 사용하면 카테고리형 변수들을 원핫인코딩으로 만들어준다.

```
n_samples = 10  
height = 3*np.random.randn(n_samples).round() + 170  
nationality = np.random.randint(0,3,n_samples)  
  
df = pd.DataFrame(list(zip(height, nationality)),  
                  columns=["height","nationality"])  
df.head()
```

	<b>height</b>	<b>nationality</b>
<b>0</b>	167.0	1
<b>1</b>	167.0	0
<b>2</b>	167.0	1
<b>3</b>	167.0	2
<b>4</b>	167.0	0

```
nat = pd.get_dummies(df['nationality'], prefix='nat_')
new_df = pd.concat([df, nat], axis=1)
new_df.head()
```

	height	nationality	nat_0	nat_1	nat_2
0	170.0		0	1	0
1	173.0		1	0	1
2	170.0		1	0	1
3	164.0		1	0	1
4	170.0		0	1	0

```
new_df.drop('nationality', 1, inplace=True)
```

```
nat_categ = pd.Categorical(nationality)
```

```
nat_categ
```

```
==>
```

```
[0, 1, 1, 1, 0, 2, 0, 1, 1, 2]
```

```
Categories (3, int64): [0, 1, 2]
```

## 스케일링

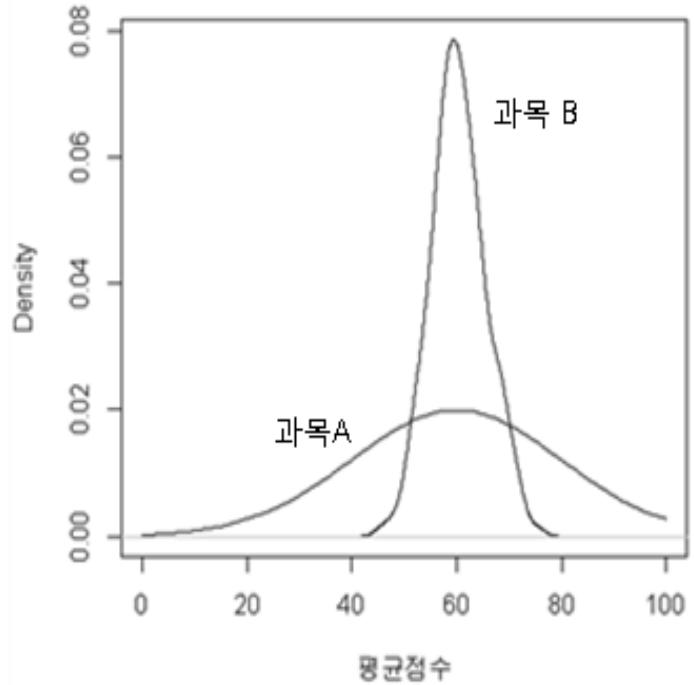
- 스케일링(scaling)이란 원래 데이터가 갖는 값의 범위를 다르게 조정하는 작업을 말한다. 스케일링을 하는 이유는 여러 가지 특성 변수를 같이 사용하여 머신러닝을 할 때 각 특성의 중요도를 갖게 맞추기 위해서이다.
- 예를 들어 모든 시험은 100점 만점으로 환산해야 동일한 비중으로 취급되며, 어떤 과목은 50점 만점, 어떤 과목을 80점 만점이면 동일한 조건으로 특성이 반영되지 않는다.

### 최소–최대 스케일링

- 주어진 값의 최소값을 0으로 최대값을 1로 조정하는 것을 최소–최대(min–max) 스케일링이라고 한다. MinMaxScaler() 함수를 사용한다.

## 표준화 스케일링

- 주어진 샘플의 평균치를 0이 되도록, 표준편차가 1이 되도록 변환하는 스케일링을 표준화(standardization)라고 하며 z-score 정규화라고도 한다.
- 예를 들어 어떤 두 과목이 모두 100점 만점이고 학급 평균도 60점으로 같으나 점수 분포가 아래와 같이 표준편차가 다르다고 하자. 과목 A의 표준편차는 20점이고, 과목 B의 표준 편차는 5점이라고 하자. 즉, 과목A의 성적이 더 넓게 퍼져있다.



평균은 60점으로 같으나 표준편차가 20인 과목 A와 표준편차가 5인 과목 B의  
확률 분포

- 두 학생 a, b의 성적이 각각 아래와 같다고 할 때 누가 더 공부를 잘 하는 학생일까?

	과목 A	과목 B	평균
학생 a	90	80	85
학생 b	80	90	85

- 위 두 학생 성적의 산술 평균은 모두 85점으로 같다. 과목 B는 편차가 작으나 이것의 의미는 대부분의 학생이 60점 근처에 모여 있다는 것이고 따라서 고득점을 받기가 매우 어려운 과목인 것을 나타낸다.
- 이러한 과목의 점수 분포 특성을 고려하면 학생 b가 어려운 과목에서 90점을 받았으므로 더 우수한 학생이라는 생각이 든다.

- 이러한 문제를 정확히 설명하려면 원 점수가 아니라 표준편차를 고려한 점수를 사용해야 하며 이때 표준 변환을 사용한다. 즉, 각 점수가 평균에서 얼마나 떨어져 있는지를 표준 편차를 기준으로 나누어 비교하려는 것이다.

$$z = \frac{x - u}{\sigma}$$

- $x$ 는 원래 값이고,  $\sigma$ 는 표준편차,  $u$ 는 평균이다.  $z$  변환을 한 이후의 데이터 분포는 평균은 0 표준편차는 1이 된다.
- 학생 성적에  $z$ 변환을 적용해 보면 학생 a의 과목 A의  $z$ 변환 성적은  $(90-60)/20 = 1.5$  점이 되고, 과목 B의  $z$ 변환 성적은  $(80-60)/5 = 4$  점이다.

- 같은 방법으로 학생 b의 z변환 성적과 평균을 계산하면 아래와 같다.

	과목 A	과목 B	평균
학생 a	1.5	4	2.75
학생 b	1	6	3.5

- 이 결과를 보면 학생 b의 z변환 후 점수의 평균이 3.5로서 학생 a 보다 더 높다는 것을 알 수 있다. 그 이유는 과목 B에서 고득점이 어려운데 상대적으로 학생 b가 6점으로 더 높은 점수를 받았기 때문이다.
- 표준 스케일링을 하려면 StandardScaler() 함수를 사용한다. 최소-최대 스케일링의 결과는 모두 0~1 사이의 값을 얻지만, 표준화 스케일링을 수행하여 10이상의 큰 값도 발생할 수 있다.

- 아래는 키와 몸무게로 구성된 데이터프레임을 표준 스케일링하는 예이다.

```
height = 3*np.random.randn(n_samples).round() + 170  
weight = 4*np.random.randn(n_samples).round() + 70
```

```
X = pd.DataFrame(list(zip(height, weight)))  
X.head()
```

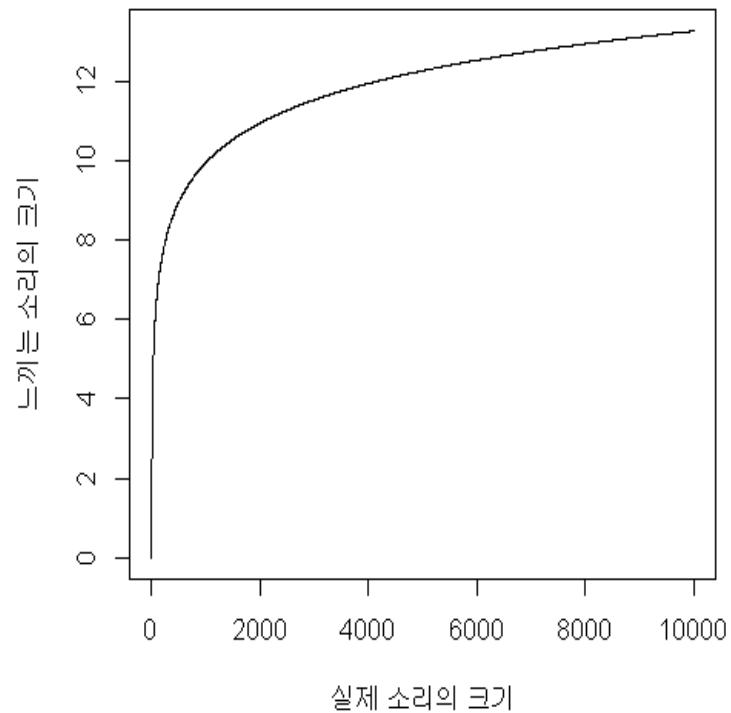
	0	1
0	170.0	66.0
1	170.0	70.0
2	170.0	74.0
3	173.0	78.0
4	173.0	74.0

```
from sklearn.preprocessing import StandardScaler  
X_std = StandardScaler().fit_transform(X);X_std  
=>  
array([[-0.46852129,  0.73730873],  
      [-0.46852129, -0.08192319],  
      [-0.46852129,  1.55654065],  
      [ 1.09321633, -0.08192319],  
      [-2.0302589 ,  1.55654065],  
      [-0.46852129, -0.08192319],  
      [ 1.09321633, -0.08192319],  
      [ 1.09321633, -0.90115511],  
      [-0.46852129, -1.72038703],  
      [ 1.09321633, -0.90115511]])
```

## 로그 변환

- 로그(log) 변환이란 원래 값에 로그를 취한 값을 사용하는 것을 말한다. 데이터 분석에서 로그를 취하는 것이 필요하고 더 의미가 맞는 경우가 있는데 두 가지 경우를 설명하겠다.
- 먼저 로그 변환이 필요한 경우는 사람이 느끼는 감각의 경우이다. 소리, 빛, 압력, 냄새, 금전적인 수입 등 생물학적인 자극에 대해서는 로그를 취한 이후의 값에 대해서 사람들이 느끼는 변화량이 선형적이라는 특성이 있다.
- 예를 들어 스피커 소리를 들을 때 소리가 2배, 3배, 4배 크게 들리게 하려면 실제 소리의 크기는 지수 함수로 키워야 한다.

- 아래 그래프에서 실제 소리 크기와 사람이 느끼는 소리의 증가는 로그 함수의 관계를 갖는 것을 나타냈다.
- 아래 그래프에서 x축이 자극의 세기를 나타내는데 여기에 로그를 취해서 그래프를 그리면 그래프가 선형적인 모양을 갖게 된다. 즉, 입력을 x축에 출력을 y축에 그리는 경우 x축을 지수 스케일로 그리면 입력과 출력의 관계가 선형적으로 된다.

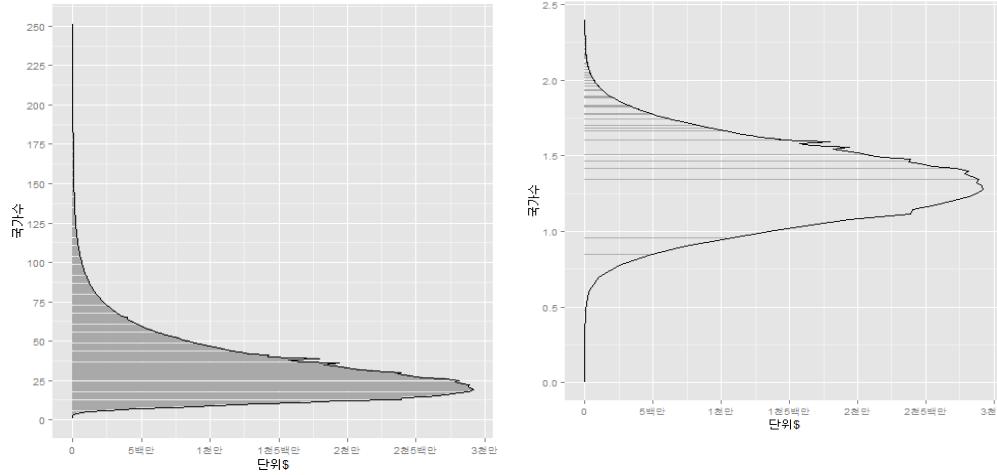


실제 소리의 크기와 느낌의 차이는 로그 관계

- 따라서 이러한 신호는 로그를 취한 값을 사용해야 선형적인 분석 모델이 잘 맞는다.
- 로그함수  $\log(x)$ 를 미분하면  $1/x$  이 된다. 이 것의 의미는 로그 형태로 변화하는 신호의 기울기(입력의 변화대 출력의 변화량)은 현재의 양에 반비례하는 것을 나타냈다.
- 예를 들어 현재 보유하고 있는 돈이 많으면 많을수록 같은 양의 수입이 늘어도 이에 따른 기쁨의 증가분은  $1/x$  에 비례하므로 현재 보유량에 반비례 한다. 소리, 압력, 온도에 대한 반응도 비슷한 패턴을 따른다.
- 두 번째 경우로, 데이터에 로그를 취하면 그 분포가 정규 분포에 가깝게 분포하는 경우가 있다. 이러한 분포를 로그정규분포(log-normal

distribution)를 갖는다고 한다. 예를 들어 국가별 수출액은 로그정규분포를 갖는다.

- 아래 그림에 y축은 연간 수출액이고 x축은 해당 국가수를 나타낸다. 좌측 그림을 보면 수출액이 많은 국가는 미국, 중국, 일본 등 숫자가 적고, 수출액이 적은 국가들의 숫자는 매우 많은 분포를 갖는다.
- x축 수출국가의 수에 로그를 취하여 그래프를 그리면 우측과 같이 정규분포처럼 나타난다.



국가별 수출액 히스토그램 (a) 원시데이터 (b) 로그로 변환한 데이터

- 데이터가 정규분포를 따르는 경우에 그 데이터의 속성을 파악하기가 쉬워지고 미래 샘플의 분포를 예상하기가 수월해진다. 즉, 예측 가능한 모델을 만들기가 쉬워진다.

## 역수 변환

- 어떤 수치를 그대로 사용하지 않고 역수를 사용하면 선형적인 특성을 가지게 되어 선형 모델로 분석도 가능하고 의미를 해석하기가 쉬워지는 경우가 있다.
- 예를 들어 자동차의 마일리지는 연료 1리터로 몇 km를 가는지를 나타낸다. 자동차의 특성을 기술하기 위해서 마일리지를 변수로 택했다면 1리터로 달릴 수 있는 ‘거리’가 특성이 된다.
- 한편, 자동차가 100km를 주행하는데 몇 리터의 연료가 필요한지를 수치로 나타내기도 한다.
- 미국의 경우 이를 주로 표시하며 이를 연비라고 하자. 연비를 자동차의

효율을 나타내는 특성 변수로 택했다면 이 때는 같은 거리를 달리기 위해서 몇 리터의 연료가 필요한지를 나타내는 ‘비용’이 특성이 된다.

- 이 두가지 마일리지와 연비는 모두 자동차의 같은 성능을 나타내지만 서로 역수의 관계이다.
- 분석 목적에 따라 어떤 변수를 사용하는 것이 더 직관적인지 또는 선형적인 영향을 주는지를 보고 판단해야 한다.
- 예를 들어 자동차가 얼마나 멀리 갈 수 있는지를 아는 것이 더 필요하면 마일리지를 특성으로 특성으로 사용하는 것이 좋을 것이다.

## 4.3. 유사도

- 머신러닝에서는 샘플들간의 유사도(similarity) 또는 거리(distance)를 측정하는 것이 필수적이다.
- 예를 들어 새로 도착한 메일이 스팸 메일인지 정상 메일인지를 구분하려면 이 메일이 스팸에 가까운지 아니면 정상 메일에 가까운지를 측정해야 한다.
- 좋아하는 음악을 추천하거나 책을 추천하려면 두 아이템 또는 두 사람이 서로 얼마나 가까운지를 측정할 수 있어야 한다.
- 이렇게 샘플 항목간의 유사한 정도를 수치로 나타낸 것을 유사도라고 하며 유사도의 반대값 즉, 서로 얼마나 거리가 먼지를 측정한 것을 거리라고 한다.

- 유사도 또는 거리를 명확하게 구하기 위해서 데이터를 다른 형태로 변환하는 것도 필요하다.

## 유사도 예시

- 다음과 같이 키, 몸무게, 나이가 서로 다른 세 명의 사람이 있는데 이 특성만을 기준으로 아래 세 명 중에 어느 두 명이 더 가까운 사람이라고 할 수 있을까?

A: 키=174cm, 몸무게=70kg, 나이=21세

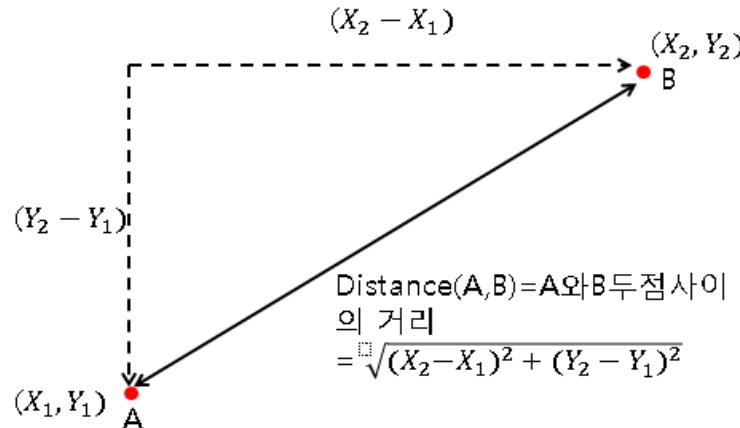
B: 키=170cm, 몸무게=61kg, 나이=27세

C: 키=162cm, 몸무게=73kg, 나이=29세

- 키만 보면 A와 B가 가깝고, 몸무게만 보면 A와 C가 가깝고, 나이만 보면 B와 C가 가깝다. 따라서 위의 정보만으로는 어느 두 명이 가장 가깝다고 말하기가 어려워 보인다.
- 샘플의 특성으로 키, 몸무게, 나이 세 개를 사용했는데 이들의 성격과 단위가 달라서 주어진 수치를 그대로 비교해서는 유사도를 측정하기가 어렵다.
- 이렇게 성격과 분포가 다른 데이터들을 사용하여 거리를 계산할 때 주로 사용하는 방법이 평균과 표준 편차를 구하고 표준 스케일링을 하는 것이다.

## 공간 거리

- 유사도 또는 거리를 측정하는 가장 일반적으로 방법은 샘플이 2차원 공간상의 점(point)라고 가정하고 이들 간의 거리를 유클리디안(Euclidian) 거리로 구하는 것이다.
- 2차원 평면에서는 두 점 A, B 사이의 거리는 피타고라스의 정리로 다음과 같이 구할 수 있다.



2차원 공간에서 유클리디안 거리를 구하는 방법 (n 차원으로 확대 가능)

- 특성이 3개 이상이 경우도 다차원 공간상의 거리를 다음과 같이 정의하여 구할 수 있다. 예를 들어 두 샘플 p와 q의 특성의 수가 n개이면 n차원 공간상의 유클리디언 거리는 다음과 같이 구한다.

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

- 위에서  $p_i$ 와  $q_i$ 는  $n$ 차원의 특성값이다.
- 앞에서 소개한 3명의 거리를 유클리디안 거리를 적용해서 계산해 보자.
- 먼저 키, 몸무게, 나이의 분포가 정규분포라고 가정하고 이를 표준 스케일링을 한 후 세 명의 거리를 구하겠다. 키, 몸무게, 나이의 표준편차가 각각 4cm, 3kg, 2세 라고 하자.
- 실제 키, 몸무게, 나이의 차이를 이 표준편차로 나누어서(즉, 정규화해서)

A, B, C 세 사람간의 거리를 구하면 다음과 같다.

$$d_{(A,B)} = \sqrt{((174-170)/4)^2 + ((70-61)/3)^2 + ((21-27)/2)^2} = \sqrt{19}$$

$$d_{(A,C)} = \sqrt{((174-162)/4)^2 + ((70-73)/3)^2 + ((21-29)/2)^2} = \sqrt{26}$$

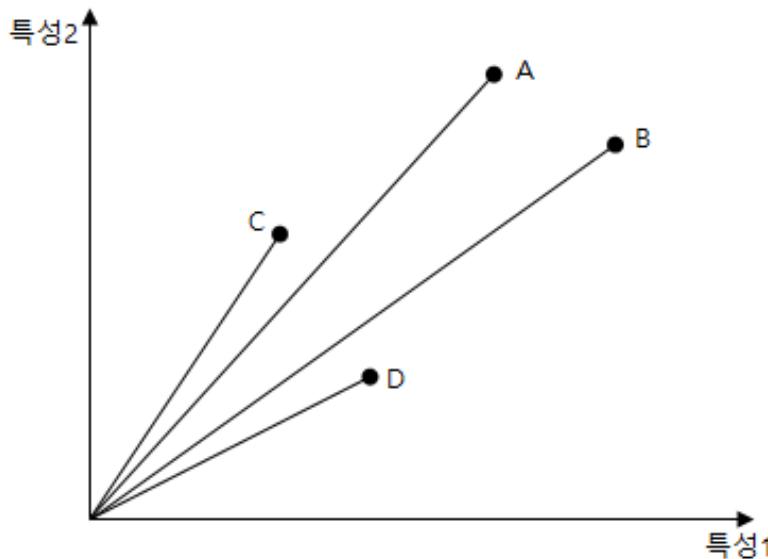
$$d_{(B,C)} = \sqrt{((170-162)/4)^2 + ((61-73)/3)^2 + ((27-29)/2)^2} = \sqrt{21}$$

- 위의 결과를 보면 A와 B가 가장 가깝고, A와 C가 가장 먼 것을 알 수 있다.
- 항목간의 유사도는 어디에 관심을 두는가에 따라 다르게 정의할 수 있다. 신체조건이 비슷한 것을 중점적으로 볼 것인지, 어떤 영화나 음식을 좋아하는지 취향을 볼 것인지, 아니면 둘 다 고려할 것인지에 따라 달라진다.

## 코사인 유사도

- 앞에서 설명한 유클리디안 거리는 가상 공간 상에 두 점의 기하학적인 거리를 계산한 것이다. 이러한 절대적인 "거리"를 구하는 대신 공간상의 두 점이 만드는 각도가 적을수록 서로 가깝다고 하는 것이 더 타당한 경우가 있다.
- 즉, 공간상에 두 점의 물리적인 거리가 멀더라도 두 점이 가리키는 방향이 같으면 서로 비슷하다고 보는 것이다.
- 예를 들어 아래 그림에서 공간상의 유클리디언 거리를 보면 A와 B가 가깝고 C와 D가 가까운 것으로 보인다. 그러나 방향성을 보면 A와 C 사이의 그리고 B와 D 사이의 각도가 작으므로 A와 C 그리고 B와 D가 더 가깝다고 할 수 있다.

- 이러한 방향성의 유사도를 코사인(cosine) 유사도라고 한다.

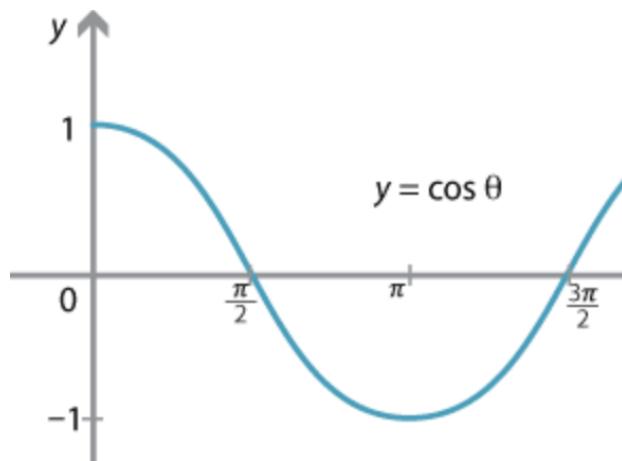


코사인 유사도에 의하면 A와 C가 가깝고 B와 D가 가깝다

- 두 명의 아빠와 두 아들이 있을 때 얼굴의 크기만을 기준으로 비교한다면 어른들이 서로 유사하고 어린이가 서로 유사하겠지만 얼굴의 생김새를 보면 각 아빠와 아들이 서로 닮았을 것이다.
- 즉, 크기보다 모양이(방향이) 유사한 정도를 평가하려면 유clidean 거리가 아니라 코사인 유사도를 사용해야 한다.
- 코사인 유사도는 아래와 같이 정의하는데 이는 두 점  $x, y$  사이의 각도의  $\cos()$  값으로 정의된다. 이 값은 벡터의 내적(inner product)를 절대값으로 나누어 구할 수 있다.

$$s_{\cos}(x,y) = \frac{X \cdot Y}{|X||Y|}$$

- 유사도로 두 점 사이의 각도의  $\cos()$ 을 택한 이유는 다음과 같다. 두 점이 정확히 같은 방향을 가리키면 둘 사이의 각도가 0이 되고  $\cos(0)=1$ 으로 유사도가 1이 되어 유사도로 사용하기가 적합하다.



- 또한 두 점이 서로 직각 방향이면 둘 사이의 각도는 90도이고  $\cos(90)=0$ 이 되어 유사도가 0이라고 표현할 수 있다.

- 만일 두 점이 공간상에 서로 반대 방향을 가리키고 있으면 두 점 사이의 각도는 180도이고 이 때  $\cos(180) = -1$  이다. 따라서 코사인 유사도는  $-1 \sim 1$  사이의 값을 가질 수 있다.
- 코사인 유사도는 텍스트 분석을 할 때 하나의 글에서 같은 단어가 얼마나 같이 자주 등장하는지 또는 반대 성향의 단어가 얼마나 자주 등장하는지를 등을 반영할 때 사용된다.

## 자카드 유사도

- 자카드(Jaccard) 유사도는 영화, 책, 음악을 추천하려고 할 때 비슷한 취향의 사람들을 찾을 때 많이 사용된다. 영화 추천 사이트에서 특정 고객에게 영화를 추천하기 위해서 그 사람과 영화 보는 취향이 비슷한 사람들을 먼저 파악해야 한다.
- 예를 들어 D라는 사람과 영화 보는 취향이 가장 비슷한 사람으로 A, B, C 세 명을 찾았다고 하자.
- D가 영화 추천을 요청하면 A, B, C가 이미 보고 재미있었다고 추천한 영화들 중에서 D가 아직 보지 않은 영화를 추천해주면 된다.
- 이러한 영화 추천 시스템을 운영하려면 사람들 간의 "영화 보는 취향에

"따른" 유사도를 측정해야 한다. 이를 위해서 같은 영화를 본 것이 얼마나 많이 겹치는지를 보고 유사도를 측정할 수 있다.

- 예를 들어 지난 1년 동안 국내에 개봉된 영화가 500편이라고 하고 이중에 A와 B가 각각 본 영화중에서 겹치는 영화가 5편이라고 하자.
- A와 B가 같은 영화를 본 비율은 전체 개봉 영화 수 대비  $5/500 = 0.01$  이 된다. 한편 A와 C가 본 영화중 겹치는 영화가 10편이라고 하면 전체 개봉영화 대비 A와 C가 같이 본 영화 비율은  $10/500 = 0.02$ 가 된다.
- 두 사람이 같이 영화를 본 비율만 유사도로 사용한다면  $0.01 < 0.02$  이므로 A와 B 사이보다 A와 C가 영화에 대한 취향이 더 가깝다고 할 수 있을 것이다.

- 그런데 이렇게 유사도를 정의하는 것은 좋은 선택이 아니다. 두 사람이 같이 본 영화의 수만 측정해서는 안되고 각자 영화를 얼마나 봤는지 절대량을 함께 고려해야 보다 정확한 유사도를 나타낼 수 있다.
- 예를 들어 A, B, C가 각각 지난해 본 영화의 총 개수가 20편, 50편, 200편이라고 하자, 그러면 A와 B는 C에 비해 영화를 거의 안보는 사람들이 다.
- 그 점을 고려하면 A와 B가 같은 영화를 5편이나 봤다는 것은 A와 B의 취향이 상대적으로 더 비슷하다고 해야 할 것이다. 영화를 거의 안 본다는 상황을 고려해야 한다.
- 이러한 점을 고려한 유사도가 자카드 유사도이며 자카드 유사도는 아래와

같이 정의한다.

- 어떤 두 항목이 겹치는 부분의 절대량만을 보지 않고, 두 항목의 공통부분이 얼마나 많은지를 고려하여 이에 대한 상대적인 값을 자카드 유사도로 사용한다. 샘플  $X, Y$  사이의 자카드 유사도  $J(X, Y)$ 는 다음과 같이 정의된다.

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

- A와 B 그리고 A와 C 사이의 자카드 유사도를 구하면 아래와 같다.

$$J(A, B) = 5 / (20 + 50) = 0.07$$

$$J(A, C) = 10 / (50 + 200) = 0.04$$

- 이를 보면 A와 B 사이의 유사도가 A와 C 사이 보다 더 큰 것을 알 수 있다. 그 이유는 A와 B가 지난해에 본 영화의 전체 수가 20편과 50편으로 C의 200편에 비해 워낙 적으므로 이를 고려하면 A와 B가 더 취향이 비슷하다고 보는 것이 타당하기 때문이다.

## 해밍 거리

- 두 개의 벡터에 동일한 패턴이 얼마나 겹치는지를 비교할 때 해밍 거리 (Hamming distance)를 자주 사용한다.
- 예를 들어 DNA 시퀀스에서 일정 구간 내에 동일한 패턴이 얼마나 있는지를 비교할 때 사용된다. 예를 들어 아래와 같은 세 가지 코드 패턴이 있다고 하자.

코드 A: a c d t c d s

코드 B: a c f r s d c

코드 C: d s f r v x w

- 각 코드는 총 7개의 변수가 있는데 코드 A와 코드 B는 같은 변수가 겹치는 곳이 3개이므로(a, c, d) 해밍 유사도는 3/70이고 거리는 4/70이다. 코드 B와 코드 C는 겹치는 변수가 2개이므로(f, r) 해밍 유사도는 2/70이고 거리는 5/70이다.

## 유사도와 거리

- 지금까지 유사도를 설명했는데 유사도는 보통  $0 \sim 1$  사이의 값을 갖도록 정의한다. (코사인 유사도는  $-1 \sim 1$  사이의 값을 갖는다).
- 유사도가 1에 가까울수록 서로 가까운 것이고 0에 가까울수록 유사성이 없다고 한다. 유사도의 상대 개념으로 거리(distance)를 사용할 때는 다음과 같이 거리( $d$ )와 유사도( $s$ )와 관계를 정의한다.

$$d = 1 - s$$

- 거리도  $0 \sim 1$  사이의 값을 가지며,  $s=0$ 일 때 즉, 유사성이 없을 때 거리는  $d=1$ 이 되고  $s$ 가 1에 가까울수록 거리는  $d=0$ 에 수렴한다.

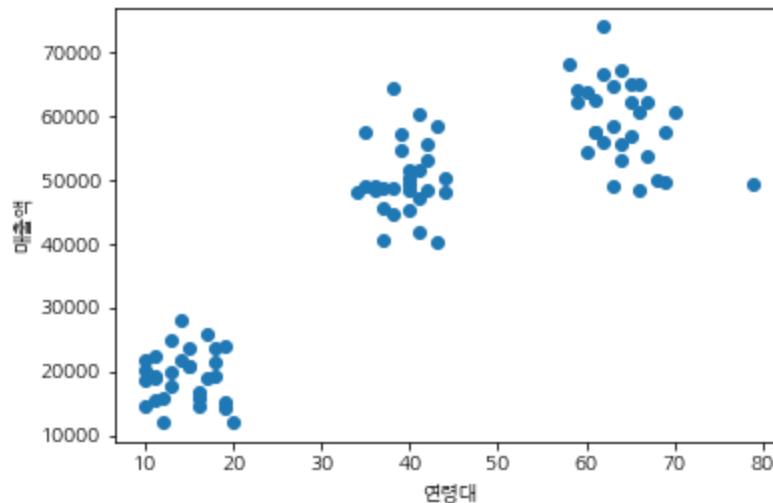
- 머신러닝에서는 계산의 편의에 따라 거리를 먼저 계산하고 유사도를 구하기도 하고, 유사도를 먼저 계산하고 거리를 구하기도 한다.
- 유사도나 거리를 구하기 전에 스케일링이 해야 하는 경우가 많다. 앞에서 키와 몸무게를 정규분포로 보고 변환을 해주어야 했다.

## 5. 클러스터링

- 클러스터링(군집화: clustering)란 성격이 비슷한 항목들을 그룹으로 묶는 작업을 말한다.
- 군집화는 대표적인 비지도학습이다.
- 분류나 회귀 등 정답을 예측하는 지도학습과 달리, 비지도 학습이란 정답이 없이 데이터로부터 중요한 의미를 찾아내는 머신러닝이다.
- 특성의 수를 줄이는 주성분분석(PCA)도 비지도 학습이다.

## 군집화 개요

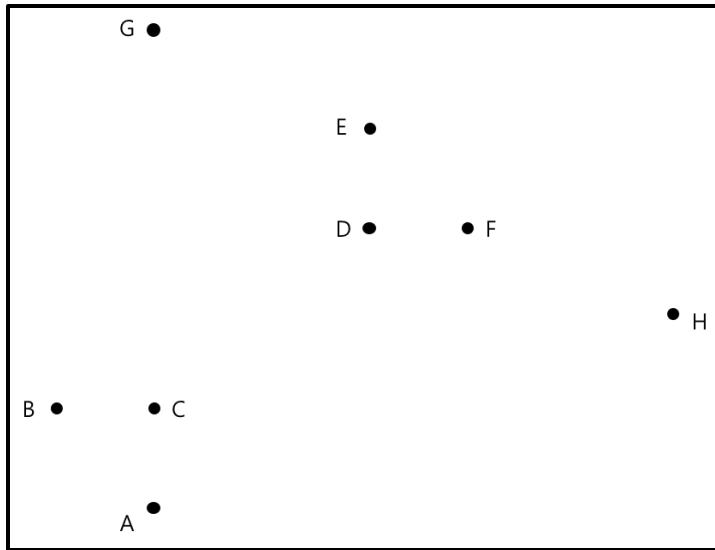
- 지난 한 달간 어떤 문구점에서 구매한 사람을 매출액을 기준으로 나누어 보았더니 아래와 같은 분포를 보였다고 하자. 고객을 몇 개의 그룹으로 나누는 것이 타당할까? 연령대와 평균 매출액을 보면 대략 세 개의 그룹으로 나누는 것이 자연스러울 것이다.



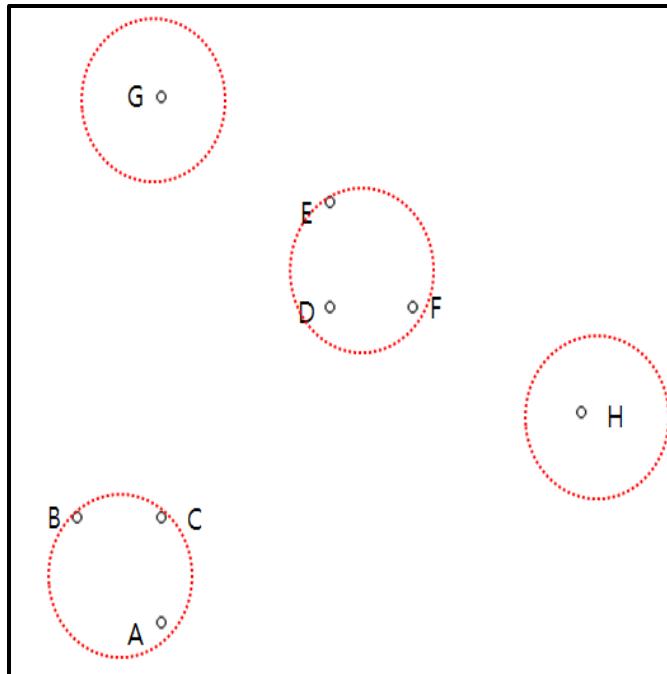
- 고객을 공통된 특징을 갖는 그룹으로 나누는 이유는 고객의 타입을 나누어 마케팅에 대응하기 위해서이다.
- 다른 예로 새로 개봉하는 영화가 관객을 얼마나 동원할지 예측하는 모델을 만든다고 하자. 여기에는 얼마나 유명한 주연 배우가 등장하는지, 감독의 등급은 얼마인지 등을 고려해야 할 것이다.
- 이 때 배우의 등급을 A, B, C 세 등급으로 나누는 것이 좋을지 10 등급으로 나누는 것이 좋을지의 판단이 필요하다, 몇 개의 등급이 좋을지 정답이 정해진 것은 아니지만 나눌 그룹의 개수는 나중에 이를 사용할 예측이나 추천 시스템의 성능에 영향을 미친다. 감독의 등급도 마찬가지이다.
- 군집화에서는 나눌 적절한 집단의 수, 즉 클러스터 수를 찾는 것이

가장 중요하다.

- 예를 들어 아래 그림과 같이 2차원 공간상에 A~H의 위치에 있는 항목을 서로 가까이 있는 항목들로 나누려고 한다. 몇 개의 그룹으로 나누는 것이 적절할까?



- 사람이 눈으로 판단한다면 아마 3개 또는 4개의 그룹으로 나누는 것이 적절할 것이다. 아래는  $k=4$ 인 때의 군집화를 나타냈다.



- 군집화의 기본 조건은 같은 그룹 내의 항목들은 서로 속성이 비슷해야 하고 다른 그룹에 속한 항목들 간에는 가능한 속성이 서로 많이 달라야 한다.

## 적절한 그룹 수( $k$ ) 선택

- 군집화에서 가장 어려운 부분이 적절한 클러스터 수인  $k$  값을 정하는 것이다.
- 예를 들어 고객 집단을 단골, 보통집단, 불만집단 등 세 개로 나누기로 미리 정해졌다면  $k$  값이 3이 된다.
- $k$  값은 최소치인 1부터 최대치인  $N$ 을 가질 수 있다. 여기서  $N$ 은 전체 샘플의 수이다. 적절한 클러스트의 수를 구할 때 각 클러스터에 속한 샘플들의 동질성(homogeniety)을 고려해야 한다. 즉, 비슷한 것들이 잘 모여 있다면 동질성이 높다고 하겠다.
- $k=N$ 으로 설정했다면 즉, 항목 수만큼 클러스터를 만들면 각 그룹에

항목이 한 개씩만 들어 있으므로 동질성은 최고이다.

- 반면에  $k=1$ 로 설정하여 클러스터를 하나만 만들면 모든 항목이 하나의 클러스터에 모여 있으므로 동질성은 최악이다.
- $k$ 가 너무 작으면 여러 가지 성격을 가진 항목들이 너무 섞여 있는 현상이 발생하고,  $k$ 를 너무 크게 하면 항목들을 세밀하게 구분은 할 수 있으나 그룹 수가 너무 많아져서 군집화를 하는 의미가 없어진다.
- 따라서 적절한 클러스터 수를 택해야 하는데 선택의 한 방법으로 동질성이 충분히 만족되는 적절히 큰  $k$  값을 택해야 한다.

# 전력 사용패턴 예

- 지역별 전력 판매량 데이터를 가지고 클러스터링을 하는 예를 소개하겠다.  
전력 사용 데이터를 읽고 서비스업과 제조업 분야의 전력 판매량을 두  
개의 특성으로 2차원 평면에 지역을 나타내는 코드는 아래와 같다.

## 데이터

```
import platform  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib import rc, font_manager  
%matplotlib inline  
power_data = pd.read_excel('data/시도별_용도별.xls')
```

```
print(power_data.shape)
```

=>

(19,28)

```
power_data = power_data.set_index('구분')
```

```
power_data = power_data.drop(['합계', '개성', '경기', '서울'], errors='ignore')
```

```
power_data = power_data.drop('합계', axis=1)
```

```
power = power_data[['서비스업', '제조업']]
```

```
power.head(5)
```

	서비스업	제조업
구분		
강원	6203749	6002286
경남	8667737	18053778
경북	8487402	30115601
광주	3174973	2910768
대구	5470438	5862633

## 덴드로그램

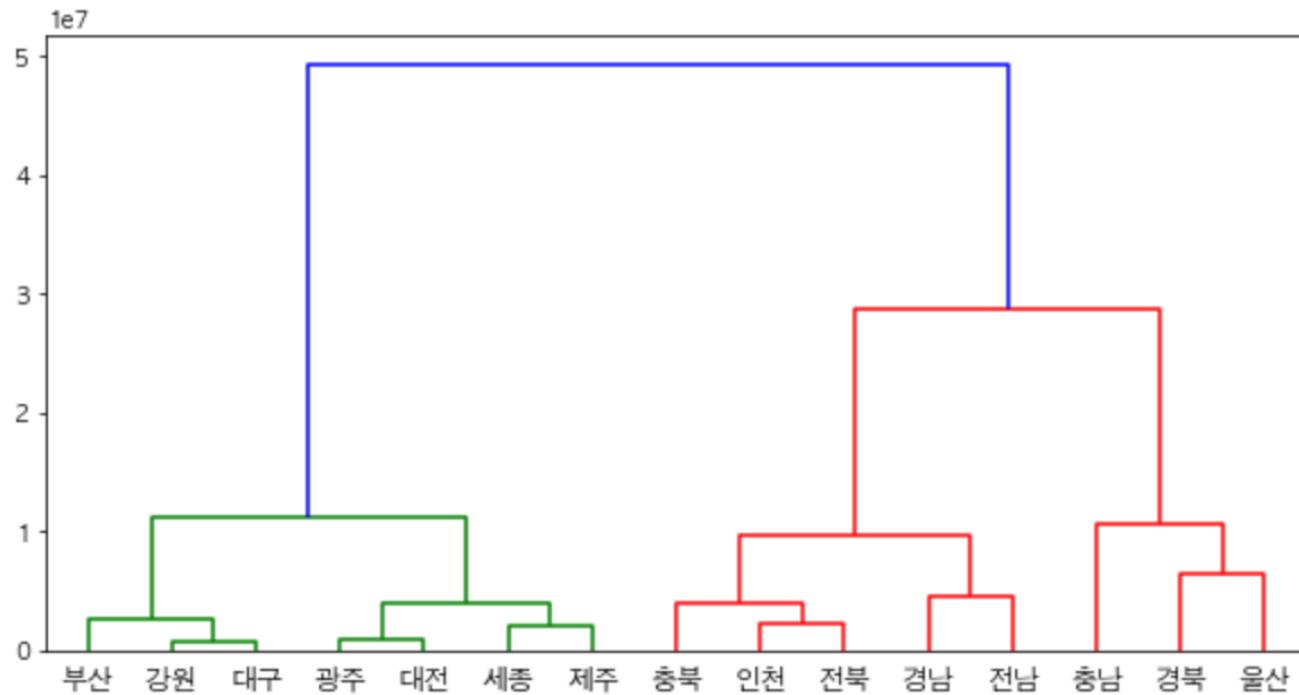
```
from scipy.cluster.hierarchy import dendrogram, linkage

plt.figure(figsize=(10, 5))

link_dist = linkage(power, metric='euclidean', method='ward')
```

```
dendrogram(link_dist, labels=power.index)
```

```
plt.show()
```



## 클러스터링

```
from sklearn.cluster import KMeans  
kmeans = KMeans(n_clusters=4).fit(power)  
  
print(kmeans.n_clusters)  
centers = kmeans.cluster_centers_  
=>  
4  
  
kmeans.labels_  
=>  
array([3, 3, 1, 2, 3, 2, 3, 2, 0, 3, 0, 0, 2, 1, 0], dtype=int32)
```

```
power.loc[:, '클러스터']=kmeans.labels_
power
```

구분	서비스업	제조업	클러스터
강원	6203749	6002286	0
경남	8667737	18053778	2
경북	8487402	30115601	3
광주	3174973	2910768	0
대구	5470438	5862633	0

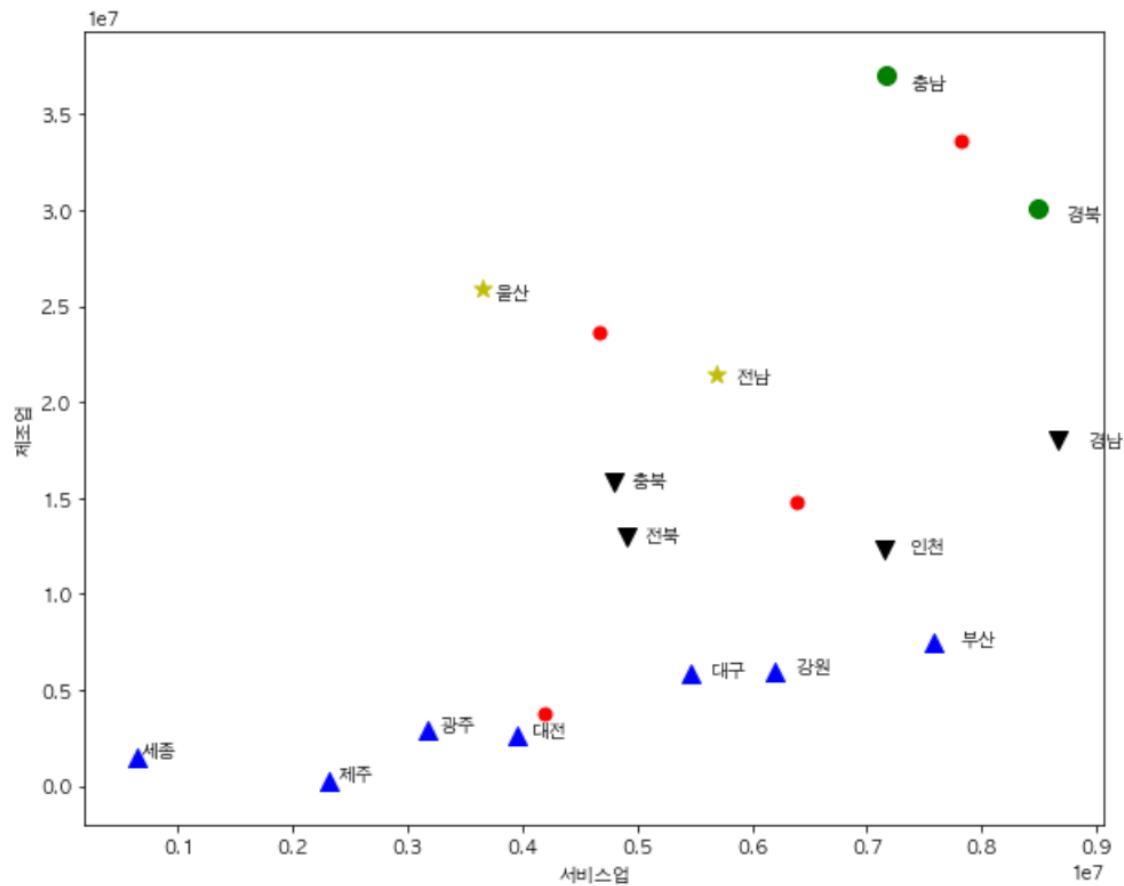
```
my_markers=['*','^', 'o','v']
```

```
my_color =['y','b','g','k']
```

```
plt.figure(figsize=(10, 8))
```

```
plt.xlabel('서비스업')
plt.ylabel('제조업')
for n in range(power.shape[0]):
    label = kmeans.labels_[n]
    plt.scatter(power['서비스업'][n], power['제조업'][n], c=my_color[label],
marker=my_markers[label], s=100)
    plt.text(power['서비스업'][n]*1.03, power['제조업'][n]*0.98, power.index[n])

for i in range(kmeans.n_clusters):
    plt.scatter(centers[i][0], centers[i][1], c = 'r', s= 50)
```



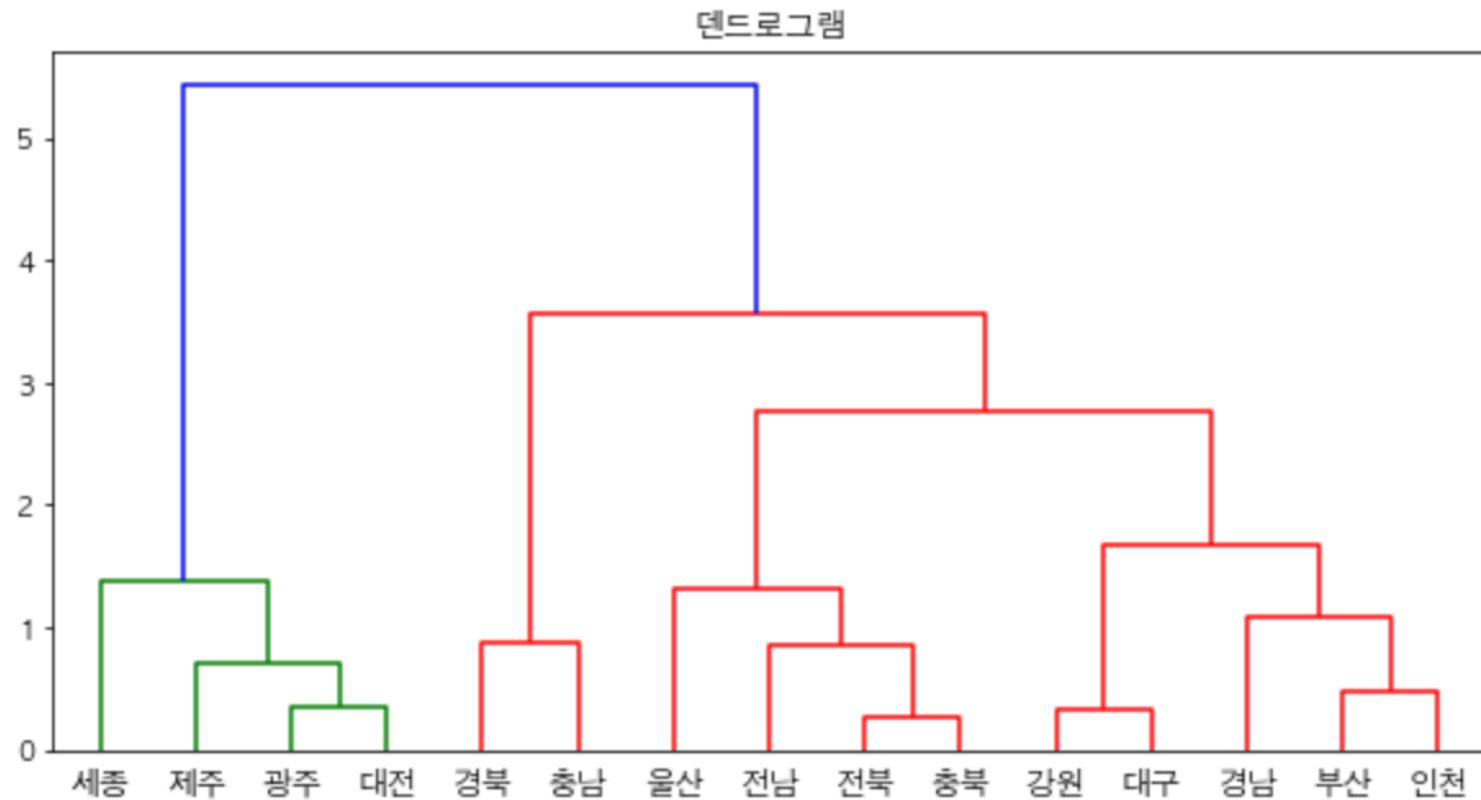
## 스케일링

```
from sklearn.preprocessing import StandardScaler  
import warnings  
warnings.simplefilter('ignore')  
  
scaler = StandardScaler()  
power[:] = scaler.fit_transform(power[:])
```

## 스케일링 이후의 덴드로그램

```
Z = linkage(power, metric='euclidean', method='ward')  
# 유클리드 거리를 이용해 Linkage Matrix를 생성  
plt.figure(figsize=(10, 5))  
plt.title('덴드로그램')  
dendrogram(Z, labels=power.index)
```

```
plt.show()
```

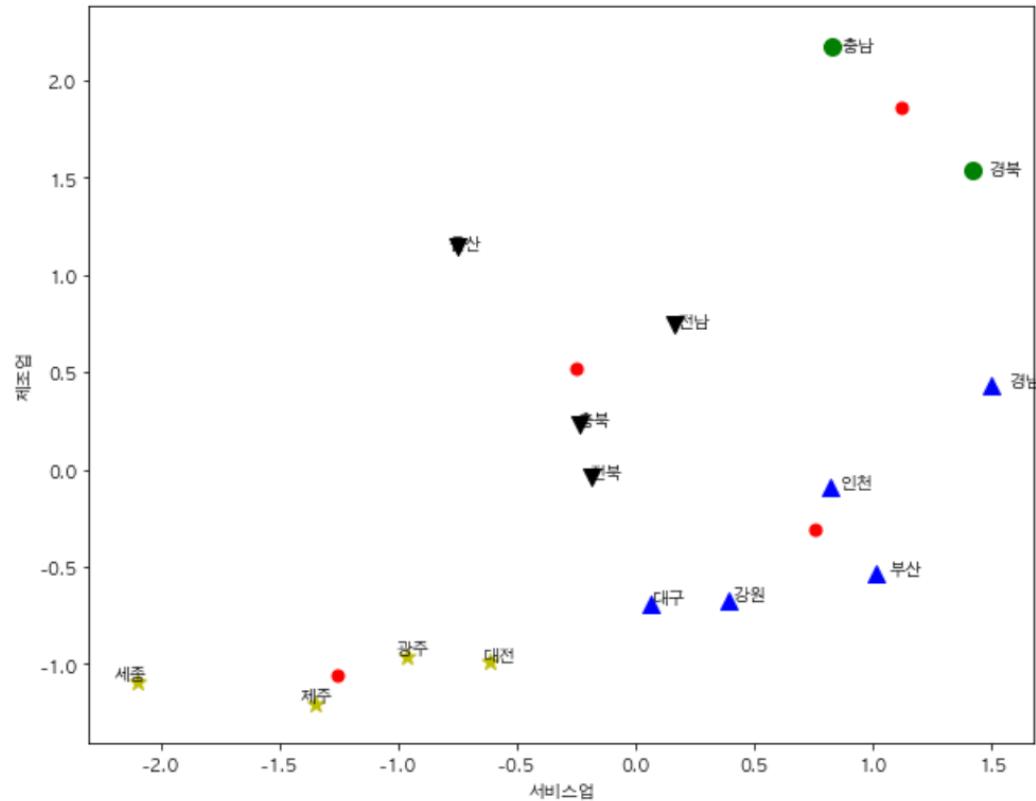


## 클러스터링

```
k = KMeans(n_clusters= 4).fit(power)
centers = k.cluster_centers_
plt.clf()
plt.figure(figsize=(10, 8))
plt.xlabel('서비스업')
plt.ylabel('제조업')

for n in range(power.shape[0]):
    label = k.labels_[n]
    plt.scatter(power['서비스업'][n], power['제조업'][n], c=my_color[label],
marker=my_markers[label], s=100)
    plt.text(power['서비스업'][n]*1.05, power['제조업'][n]*0.99, power.index[n])
```

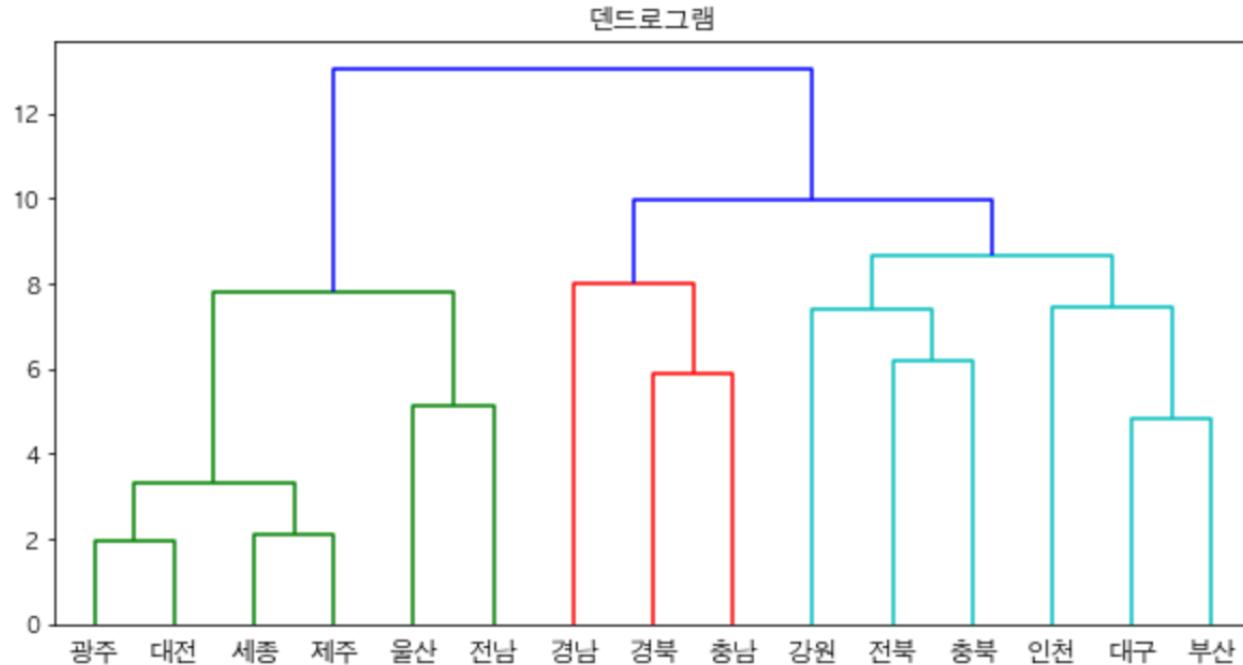
```
for i in range(k.n_clusters):  
    plt.scatter(centers[i][0], centers[i][1], c = 'r', s= 50)
```



## 모든 특성을 고려

```
power3 = power_data.drop(['업무용합계','산업용합계'], axis=1, errors='ignore')
power3[:] = scaler.fit_transform(power3[:])
```

```
Z = linkage(power3, metric='euclidean', method='ward')
# 유클리드 거리를 이용해 Linkage Matrix를 생성
plt.figure(figsize=(10, 5))
plt.title('덴드로그램')
dendrogram(Z, labels=power3.index)
plt.show()
```



```
k = KMeans(n_clusters= 4).fit(power3)
```

```
n_samples = power3.shape[0]  
print(n_samples)
```

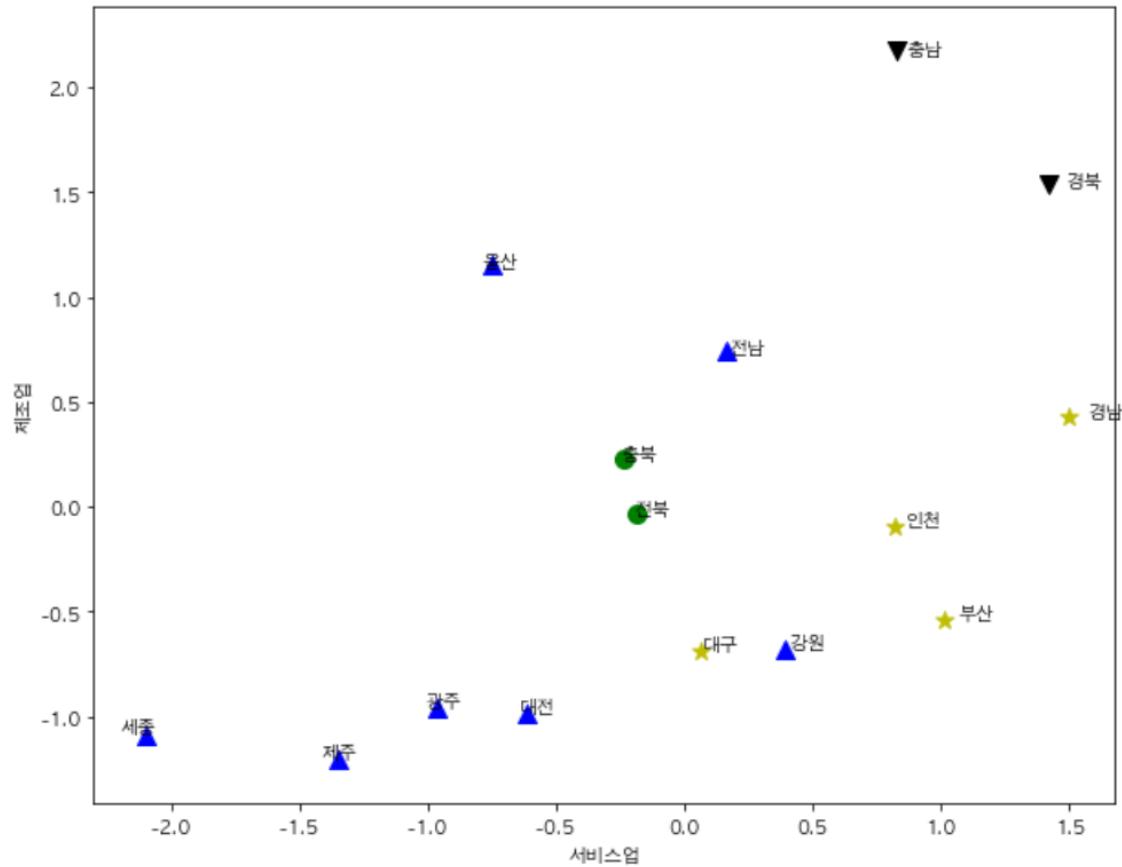
```
plt.figure(figsize=(10, 8))

plt.xlabel('서비스업')
plt.ylabel('제조업')

for n in range(n_samples):
    label = k.labels_[n]

    plt.scatter(power3['서비스업'][n], power3['제조업'][n], c=my_color[label],
marker=my_markers[label], s=100)

    plt.text(power3['서비스업'][n]*1.05, power3['제조업'][n]*0.99, power3.index[n])
```



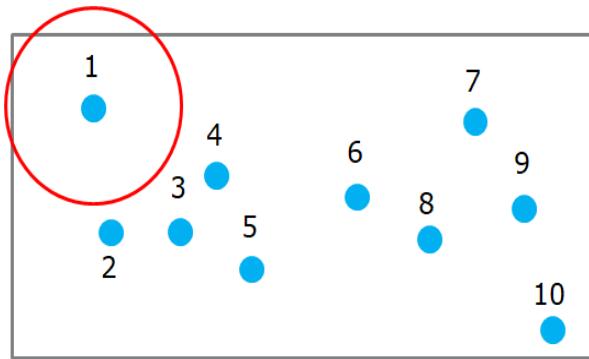
## k-means 알고리즘

- 군집화를 수행하는데 k-means 알고리즘이 널리 사용된다.
- k-means 알고리즘 동작 원리는 매우 간단하다. 전체 항목을  $k$  개의 그룹으로 나누되 각 그룹 내에 들어 있는 항목들 간의 거리는 가능한 작고, 서로 다른 그룹에 들어 있는 항목들 간의 거리는 가능한 멀게 나눈다.
- k-means 알고리즘의 동작 순서는 다음과 같다. 먼저 특성 변수 공간상에 임의의  $k$  개의 초기 지점을 정한다. 이를 클러스터 중점(cluster center)이라고 한다. 초기 클러스터 중점은 임의의  $k$  개의 샘플을 선정한다.

- 다음에는 남아 있는 샘플 중에 클러스터 중점과 가까운 샘플을 포함시켜 조금 더 큰 클러스터를 만든다. 현재 각 클러스터에 포함된 항목의 위치들의 평균(무게 중심)을 구해서 이를 새로운 클러스터 중점으로 변경한다.
- 새로 설정된 중점을 중심으로 하여 가까운 샘플들을 모아서 전체 항목을 고려한 중점을 다시 만든다.
- 이제 근처의 다른 중점과 거리가 가까운 샘플들은 클러스터의 소속을 변경한다. 이제 소속이 변경된 항목들을 반영하여 중점을 다시 구한다.
- 이와 같이 클러스터의 무게 중심을 바꾸는 작업을 계속 진행하여 더 이상 클러스터의 모양이 바뀌지 않을 때까지 수행하면 군집화가 완료된다.

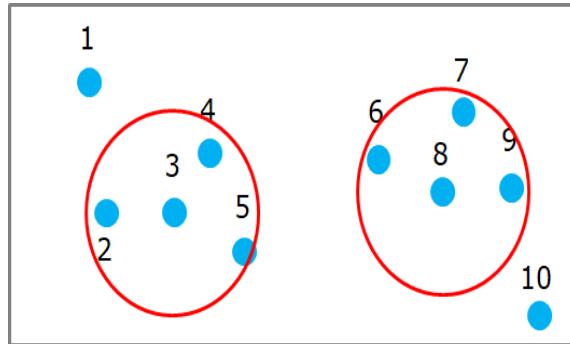
# DBSCAN

- DBSCAN은 밀도 기반 클러스터링 알고리즘이다. k-means처럼 단순히 거리만을 기준으로 군집화를 하는 것이 아니라 “가까이 있는 샘플들은 같은 군집에 속한다”는 원칙으로 군집을 차례로 넓혀가는 방식이다.
- 샘플들의 몰려 있는 정도 즉, 밀도가 높은 부분을 중심으로 인접한 샘플들을 포함시켜 나가는 방식이다. 어떠한 한 점을 기준점으로 반경  $r$ 내에 점이  $n$ 개 이상 있으면 하나의 군집으로 인식하는 방식이다.
- 예를 들어 아래와 같이 10개의 데이터가 분포되어 있다고 하자. 여기에 반지름이  $r$ 인 원을 1번 데이터부터 10번 데이터를 각각 중심으로 그리고 원 안에 포함된 샘플의 수를 세어본다.

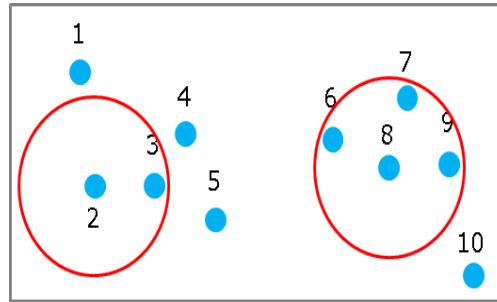


- 1번 데이터를 중심으로 보면 반지름  $r$ 인 원 안에 점이 1번을 제외하고는 없다. 군집이 되기 위한 최소기준이 예를 들어  $n=4$ 라면 이 데이터는 노이즈 데이터(noise point)라 되며 클러스터에서 제외한다.
- 3번과 8번 데이터를 중심으로 보면 원 안에 4개의 점이 있으며 이러한 데이터를 코어 데이터(core point)라고 한다. 코어 데이터들은 스스로

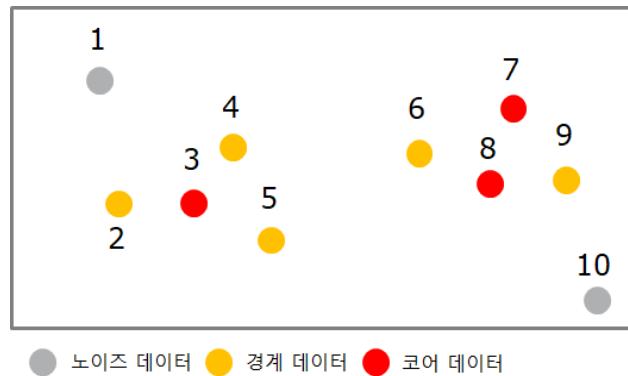
클러스터를 형성할 수 있다.



- 2번과 8번 데이터는 최소 기준인 4개의 데이터를 포함하는 것에는 충족하지 못하지만 코어 데이터인 3번 데이터를 포함한다. 이런 데이터를 경계 데이터 (border point)라고 하며 인접한 군집에 포함시킨다.



- 이러한 방식으로 반지름  $r$ 인 원을 이용해 코어 데이터, 경계 데이터, 노이즈 데이터들을 분류해보면 아래 그림과 같은 결과를 얻을 수 있다.

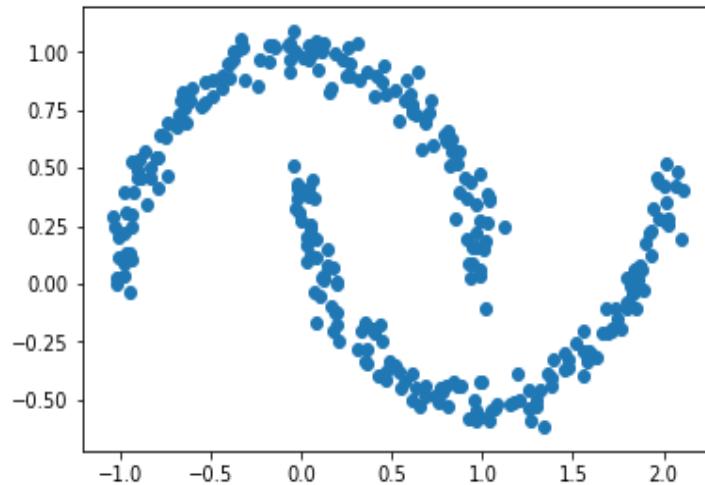


## two-moons 데이터

- DBSCAN이 유용한 것을 보이는 예를 들어보겠다. 아래의 코드는 make\_moons 데이터 생성함수로 데이터를 만들고 이를 각각 KMeans와 DBSCAN 알고리즘을 적용하였다.
- 원에 속하는 데이터의 개수가 10개인 DBSCAN 모델을 만들었고 fit\_predict를 사용하여 모델을 훈련시킨다.

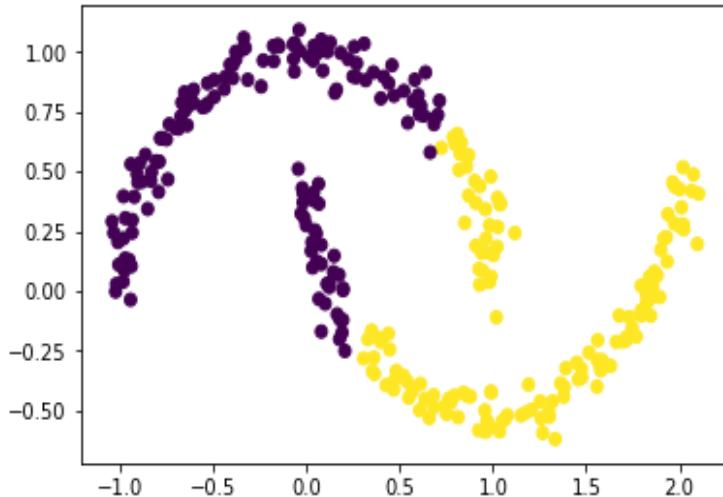
```
%matplotlib inline  
import numpy as np  
import matplotlib  
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_moons  
from sklearn.cluster import KMeans  
from sklearn.cluster import DBSCAN  
  
X, y = make_moons(n_samples=300, noise=0.05, random_state=11)  
plt.scatter(X[:,0], X[:,1])  
plt.show()  
=>
```



- 위 그림에 보듯이 make\_moons 데이터는 달이 두 개 서로 엎어져 겹친 모양이다. 이 모양을 보면 서로 연결되어 있는 모양이 하나의 클러스터로 나누어져야 하는데 kmeans 알고리즘을 적용하면 거리가 가까운 점들을 하나의 클러스터로 묶기 때문에 우리가 원하는 결과를 얻지 못한다.

```
kmeans = KMeans(n_clusters=2)  
predict = kmeans.fit_predict(X)  
plt.scatter(X[:,0], X[:,1],c=predict)  
#출력
```

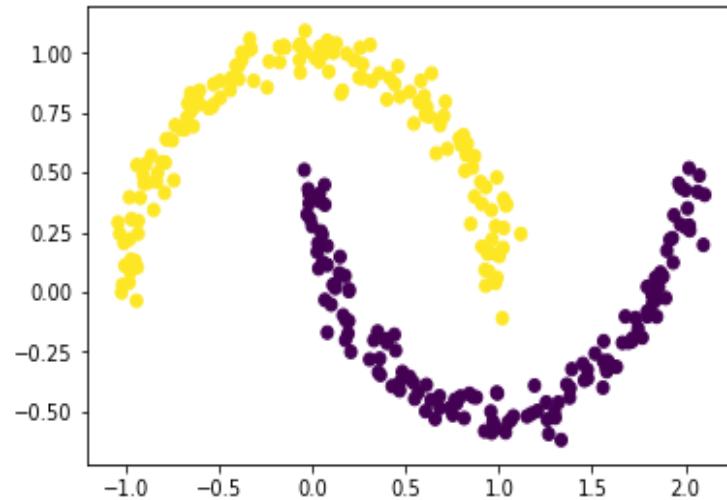


- kmeans 알고리즘 대신 DBSCAN을 사용하면 두 개의 달 모양을 따라 클러스터가 만들 수 있다.

```
dbscan = DBSCAN(eps=0.2, min_samples=10, metric='euclidean')
predict = dbscan.fit_predict(X)
```

```
plt.scatter(X[:,0], X[:,1],c=predict)
```

#출력



# 6. 머신러닝

- 머신러닝이란 컴퓨터가 데이터를 분석하여 중요한 의미를 추출하거나 미래를 예측하는 모델을 만드는 기술을 말한다.
- 머신러닝을 사용하는 목적 즉, 머신러닝으로 문제를 해결하는 유형을 다음과 같이 세 가지로 나눌 수 있다.

예측(prediction)

설명(description)

추천(recommendation)

- 머신러닝 구현 유형은 정답을 알고 학습하는지 아닌지에 따라 크게 지도학습 또는 비지도 학습으로 나누어진다.

## 6.1. 머신러닝 목적

### 예측

- 새로운 샘플에 대한 미래 값을 예측하는 것을 말하며 예측에는 회귀(regression)와 분류(classification) 두 가지가 있다.
- 회귀는 수치를 예측하는 것을 말하는데 내일의 날씨 예측, 주가 예측, 병에 걸릴 확률이 얼마일지, 다음 달 가게의 매출이 얼마일지 등을 예측하는 모델을 만든다.
- 분류는 주어진 샘플이 어느 카테고리에 속하는지를 예측하는 것이며 수신한 메일이 스팸인지 아닌지를 구분하는 것, 이번 은행 대출이 부도가 날지 아닐지, 누가 우수 고객인지를 예측하는 것 등이다.

## 설명

- 설명이란 어떤 현상의 원인을 데이터 분석을 통해 설명하는 것을 말한다. 어떤 상품이 많이 팔렸다면 그 이유를 파악하는 것 등이다.
- 예를 들어 슈퍼마켓에서 어떤 품목들이 자주 같이 판매되는지 패턴을 찾아내는 것은 일종의 서술형 모델이다. 유사한 특성을 가진 항목들을 함께 묶는 것을 군집화라고 하는데 군집화도 서술형 모델이다.
- 설명적 분석은 데이터를 이해하는 것이라고 할 수 있는데 마켓 리서치, 고객의 행동전환 파악, 탐색적 분석을 포함한다. 이를 통해서 새로운 인사이트를 얻도록 한다.

## **추천**

- 추천이란 주어진 조건에서 최적의 선택을 제시하는 것으로 의사결정을 돋는 것을 말한다. 이를 위해서 설명 및 예측 모델을 활용한다.
- 예를 들어, 상품추천, 영화추천, 음악추천, 약 추천, 네비게이터, 검색엔진, 자율차의 운행, 알파고와 같은 게임 플레이어, 보험 사기청구 거절 등이 해당된다.

## 6.2. 머신러닝 유형

- 지도학습, 비지도 학습, 강화학습의 차이를 설명하겠다.

### 지도 학습

- 앞에서 머신러닝은 회귀(regression)나 분류 등 예측에 주로 사용한다고 했다. 예측은 모두 시간이 지나면 정답을 확인할 수 있고 머신러닝 모델의 성능을 정확히 평가할 수 있다.
- 이와 같이 정답을 알고 학습하는 유형의 머신러닝을 지도학습(supervised learning)이라고 한다.
- 정답에 해당하는 값을 목적변수 (target variable) 또는 레이블(label)이라고 부른다.

- 레이블은 회귀분석에서는 수치로 주어지고 분류에서는 카테고리 변수로 표현된다.
- 예를 들어 스팸메일 분류를 학습시키려면 어떤 메일이 스팸이었는지 정답 샘플도 같이 주어져야 한다.

## 회귀

- 회귀분석은 경제지표 예측, 사회학 연구, 마케팅, 의학에서 치료의 효과를 분석하는데 사용되며, 재난시 피해액 산정, 선거 결과 예측, 범죄 발생 예측 등 광범위하게 사용된다.
- 회귀분석에서 사용하는 대표적인 알고리즘은 선형회귀, kNN, SVM, 로지스틱 회귀, 랜덤 포레스트, 신경망 등이다.

## 분류

- 분류(classification)란 어떤 항목(item)이 어느 그룹에 속하는지를 판별하는 기능을 말한다.
- 예를 들어 새로 도착한 메일이 스팸인지 아닌지를 판별하는 것 등을 말한다. 우수 고객을 찾아내거나, 충성심 높은 신입사원을 선발하거나, 투자할 좋은 회사를 구분하는 작업은 분류 문제이다.
- 매장에 들어오는 사람을 보고 이 사람이 물건을 살 고객인지, 단순히 구경만 할 고객인지, 아니면 항의하러 들어오는 고객인지를 판단하여 적절하게 대응하려고 한다면 매장에 들어오는 고객의 타입을 분류해야 한다.
- 광고 안내문이나 기념품을 잠재 고객에게 보내는 경우에도 기왕이면 구매할

확률이 높은 고객을 찾아서 보내는 것이 좋을 것이다. 그러려면 과거의 구매이력 데이터 분석이 필요하고 SNS 등 고객의 다른 활동 데이터를 같이 분석하여 우수 고객을 분류하는 것이 필요하다.

- 분류에 사용되는 대표적인 알고리즘으로는 확률적인 모델을 이용하는 베이시안 알고리즘이 널리 사용된다. 베이시안 모델은 사건들이 서로 어떤 영향을 주는지를 분석한다.
- 스팸 메일을 찾는 데에 베이시안 알고리즘이 주로 사용되는데 예를 들어 “급매”라는 단어가 들어 있는 메일은 스팸일 확률이 얼마인지를 미리 측정해 두고 이를 스팸 메일 필터링에 사용하는 것이다.
- 만일 “급매” 단어와 함께 “카드결재” 단어가 같이 들어 있으면 스팸일 확률이 더 올라간다고 모델을 만들 수 있다.

## 비지도학습

- 지도학습은 훈련을 하는 데에도 정답이 필요하고, 테스트를 할 때에도 정답과 맞추어 성능을 평가한다. 이와 달리 비지도학습(unsupervised learning)은 정답이 없이 학습하는 유형이다.
- 즉, 훈련 데이터에 목적변수가 포함되지 않다. 데이터의 특성을 기술하는 서술형 모델이 비지도 학습이며 군집화(clustering), 연관분석, 시각화, 데이터변환, 차원 축소 등이 비지도 학습에 해당된다.

## 강화 학습

- 강화학습(reinforcement learning)은 머신러닝 모델이 어느 방향으로 만들어져야 하는지 방향성만 알려주는 학습 방법이다.
- 입력 샘플마다 정답을 있어 답을 알려주는 것이 아니지만 시간이 흐르면서 모델이 바람직한 방향으로 가고 있는지를 알려줄 수 있고 이를 통해서 학습하는 방법이다.
- 예를 들어 게임을 하는 강화학습 모델을 만든다면 매 입력시마다 답을 주지는 못하지만 게임을 이기고 있는지 아니면 지고 있는지를 알려줌으로써 스스로 게임을 잘 수행하는 방법을 터득하게 한다.
- 강화학습도 크게 보면 지도 학습이지만 매 샘플마다 정답을 알려줄 수

없다는 차이가 있다. 강화학습에서는 일정 기간동안의 행동(action)에 대해 보상(reward)을 해줌으로써 잘 하고 있는지, 잘 못하고 있는지를 알려주며 학습을 시킨다.

- 예를 들어 로봇이 혼자 그네를 타는 방법, 전자 게임을 하는 방법, 바둑을 두는 방법의 학습에 사용된다.
- 2017년에 우리나라 이세돌을 이긴 알파고(Alpha Go) 바둑 프로그램은 강화학습 방법을 사용하여 개발되었다.

## 머신러닝의 특징

- 머신러닝 이전의 컴퓨터 프로그램과 머신러닝을 어떤 면에서 다른가? 예전에는 컴퓨터는 프로그래머가 코딩한 대로만 동작을 했다. 계산을 빨리 하든지, 이미지를 처리하든지, 정해진 알고리즘대로 빠르고 정확하게 동작하는 일이 대부분이었다.
- 이에 반해 머신러닝에서는 컴퓨터가 데이터를 보면서 점차 성능을 향상시킬 수 있다. 이제 컴퓨터가 데이터를 보고 스스로 기능을 향상시키는 방법을 찾아낸 것이다.
- 머신러닝은 예측이나 설명을 수행하기 위해서 모델을 사용하는데 모델이란 예를 들어 스팸 메일을 찾아내는 모델, 누가 게임에서 이길지 예측하는 모델, 내일 날씨를 예측하는 모델 등을 말한다.

## 머신러닝과 인공지능

- 인공지능(AI: artificial intelligence)이란 컴퓨터가 마치 지능이 있는 것처럼 똑똑하게 동작하는 것을 통칭한다.
- 1960년대 컴퓨터가 보급되기 시작하면서 “사람과 같이 생각하는” 인공지능 구현을 시도하였는데 이러한 접근 방법은 성과를 내지 못했는데 그 이유는 사람처럼 생각하는 것을 프로그램으로 구현하는 것이 불가능하기 때문이다.
- 어린이도 한 눈에 고양이와 강아지를 구분하지만 이를 컴퓨터 알고리즘으로 구현하기는 매우 어려웠다. 그러나 데이터 기반으로 동작하는, 즉 머신러닝을 사용하는 인공지능 구현 방법은 2000년대 이후 급속히 발전하면서 인공지능이 꽤 만족할만한 수준으로 작동하기 시작했다.

## 빅데이터와 인공지능

- 인공지능은 빅데이터 공급으로 급속히 발전하였다. 지금까지 가치 있는 정보는 주로 사람이 만들어냈다. 뉴스, SNS 데이터, 블로그, 통계자료, 음악, 영화 등 콘텐츠를 구성하는 데이터는 사람이 만들어 냈다.
- 그러나 앞으로는 사람이 직접 만드는 데이터보다 센서와 정보기기들이 생산하는 데이터가 급격히 늘어날 것이다.
- 온도, 습도와 같은 과학적 측정 데이터로부터, 소음, 카메라, 오염도, 교통상황, 인구밀도 변화, 약물사용 통계 등 다양한 데이터가 자동으로 만들어질 것이다.
- 이렇게 사물이 데이터를 생산하며 인터넷으로 연결되는 것을 사물 인터넷

(IOT: Internet of Things)이라고 한다. 사물인터넷은 빅데이터 생산과 인공지능의 발전에 데이터 공급차원에서 점차 많은 역할을 담당할 것이다.

## 6.3. 머신러닝 프로세스

- 머신러닝 모델을 만들고 결과를 적용하는 절차를 문제 정의, 전략 수립, 데이터 수집, 머신러닝 모델 구현, 결과 적용 등 다섯 단계로 구성할 수 있다.

## 문제 정의

- 머신러닝 프로세스에서 가장 먼저 수행해야 하며 가장 중요한 과정은 해결할 문제를 명확하게 정의하는 것이다.
- 주어진 문제가 데이터를 확보하고 분석하면 과연 해결할 수 있는 문제인지 를 먼저 파악해야 한다. 처음부터 문제가 아닌 것을 해결하려고 시도하는 경우가 있으며 아무리 데이터를 수집해도 답을 얻을 수 없는 문제도 있다.
- 문제는 조건, 조직, 시기, 장소 등에 따라 다를 수 있다. 문제를 정의할 때 주의할 것은 큰 문제를 한 번에 해결하려면 힘들다는 것이다.
- 최종적으로 해결해야 할 큰 문제를 정의하는 것을 필요하지만 이를 머신러닝 모델이 해결할 수 있는 작은 문제로 나누어 접근하는 전략이 필요하다.

- 예를 들어 기업의 수익 감소 문제를 한 번에 해결하는 모델은 만들기가 어렵다. 대신 배송이 늦어지는 이유를 찾아내거나, 고객 불만이 발생하는 원인을 찾거나, 반품이 많은 이유를 분석하는 것 등으로 나누어 다루어야 한다.

## 전략 수립

- 해결할 문제를 정한 후에는 이 문제를 해결하는 과정을 하나의 프로젝트라고 보고 프로젝트 수행 전략을 수립해야 한다.
- 관련 부서는 어디인지, 프로젝트를 누가 주도할지, 필요한 데이터는 누가 가지고 있는지, 주어진 예산과 기간은 얼마인지 등을 파악해야 한다.
- 전략을 잘 세우는 첫 단계는 문제를 다시 검토하는 것이다. 왜 그것이 문제인지를 다시 생각해봐야 한다. 고객의 근본적인 고민이 무엇인지, 현재는 그 문제를 해결하기 위해서 어떤 방법을 사용하고 있는지, 그리고 그 방법이 왜 잘 동작하지 않고 있는지를 다시 검토하여 문제의 본질을 파악해야 한다.

- 다음에는 문제 해결의 목표를 명확히 정해야 한다. 막연히 비용절감, 좋은 아이디어 도출, 서비스 개선과 같은 불분명한 목표가 아니라, 비용을 몇 % 줄일지 등 모델이 달성해야 할 구체적인 목표를 정해야 한다.
- 이런 목표가 있어야만 개발을 종료할 수 있는 조건을 정할 수 있다. 아니면 학습 과정은 끝없이 계속된다. 목표가 명확하지 않으면 참여자들은 각각 기대 수준이 다르게 되고 나중에 서로 불만이 될 수가 있다.
- 다음에는 가용 자원을 파악해야 한다. 사용할 수 있는 데이터에는 무엇이 있는지, 인적 자원은 어떤지 등을 리스트를 만든다. 데이터 수집 비용이 많이 들지 아니면 공짜로 얻을 수 있는지도 알아야 한다. 데이터를 가지고 있는 부서와 협조가 잘 것인지도 알아야 한다.

투입할 수 있는 예산은?

최종 목표 산출물의 구체성은?

프로젝트 기간은?

사용할 수 있는 데이터의 종류는?

인적 자원은? (내부 인력 또는 외부 전문가 활용)

컴퓨팅 자원은? (클라우드 사용)

데이터 수집 비용은?(무료, 유료)

관계자와의 협조 및 갈등 요소는?

보안 대책은?

- 관련 데이터 조사는 데이터의 목록을 작성하는 것에서 시작한다. 이 문제 해결에 직접 필요할 거라고 예상한 데이터 뿐 아니라 이 문제와 조금이라도 관련된 데이터에는 어떤 것들이 있는지를 파악해야 한다.

- 의외로 예상하지 않은 데이터로부터 유용한 도움을 받는 수가 있다. 분석에 직접 도움이 되는 데이터를 구하기 어려운 경우에는 우회적으로 어떤 데이터로부터 원하는 정보를 추정할 수 있는지도 검토해 봐야 한다.

## 데이터 수집

- 머신러닝에 필요한 데이터를 실제로 준비하는 과정이다. 어떤 데이터가 수집 가능한지, 실제로 수집되는 데이터가 쓸만한지, 이정도 데이터면 분량이 충분한지, 데이터 품질은 만족할만한지 등을 파악해야 한다.
- 데이터 수집에서 가장 중요한 것은 핵심적인 데이터가 없으면 프로젝트 자체가 의미가 없는 경우가 많다는 것이다. 어떤 핵심 데이터를 얻을 수 있다고 가정했는데 실제로는 수집하기 어려운 경우도 있다.
- 또는 데이터 품질이 나빠서 분석이 곤란할 수도 있다. 핵심 데이터를 얻지 못한다면 전략을 다시 세워야 한다.
- 데이터를 수집하면 수집한 데이터의 전체 모습을 살펴보는 것이 필요하며

데이터 자체에 문제가 있는지도 봐야 한다. 수집된 데이터에는 빠진 데이터가 있거나, 데이터에 오류가 많으면 이들의 처리 비용이 많이 들 수 있다.

## 머신러닝 모델 구현

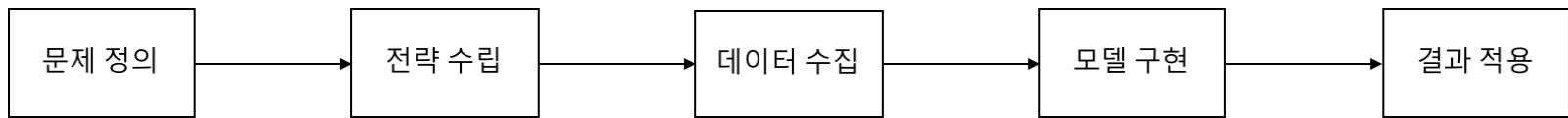
- 최종적인 머신러닝 모델 구조를 결정하기 전에 일부 샘플 데이터를 사용해서 여러 가지 머신러닝 모델을 시도해 보는 것이 필요하다.
- 처음부터 완전한 모델을 만들려고 하지 말고 초기에는 기본적인 동작만 수행해 보고 상세한 모델은 더 많은 데이터를 학습하면서 개선시켜야 한다.
- 초기의 모델을 통해서 실제로 필요한 데이터의 범위를 다시 정하기도 한다. 초기 결과를 보고 수집해야 할 데이터의 종류가 달라질 수 있고, 또 얼마나 상세한 데이터가 필요한지 등이 달라질 수 있다.
- 예를 들어 하루 한 번의 평균 값만 있으면 될 분석에 1분마다 측정한 데이터를 사용할 필요는 없다.

## 결과 적용

- 머신러닝의 최종 목적은 모델을 실제 상황에 적용하는 것이다. 모델을 실전에서 어떻게 이용할지도 미리 시뮬레이션 해봐야 한다.
- 머신러닝 모델은 한 번에 만족할 만한 성과를 내지 못하는 경우가 많다. 실전에 적용되면서, 즉, 새로운 실제 데이터가 계속 추가 입력되면서 모델의 성능이 진화하도록 해야 한다. 모델이 진화하도록 설계되지 않았다면 제대로 시스템을 구현하지 못한 것이다.

## 프로세스 요약

- 앞에서 설명한 머신러닝 프로세스를 정리하면 다음과 같다.



문제 정의 - 해결하려는 문제를 명확히 정의하는 것

전략 수립 - 문제 해결을 위해 어떤 데이터를 어떻게 사용할지를 정하는 것

데이터 수집 - 머신러닝에 필요한 데이터를 수집하는 것

모델 구현 - 분류, 회귀, 설명, 추천 등을 위한 머신러닝 모델을 구현하는 것

결과 적용 - 머신러닝 모델을 실제 상황에 적용하고 성능을 개선하는 것

## 6.4. 머신러닝 동작

- 머신러닝에서는 좋은 모델을 만드는 것이 핵심이다. 모델이란 회귀나 분류를 수행하는 컴퓨터 프로그램(알고리즘)을 말한다.

### 머신러닝 모델

- 우리가 모델을 사용하는 역사는 오래되었다. 수학이나 과학에서는 어떤 현상을 설명하는 모델로 수식을 주로 사용하였다.
- 예를 들어 모든 질량을 가진 모든 물체는 서로 끌어당긴다는 만유인력 법칙은 두 물체의 질량에 각각 비례하고, 두 물체의 거리의 자승에 반비례하는 수식으로 표현된다.
- 이 모델 덕분에 자유낙하 하는 물체의 속도를 정확하게 예측할 수 있으며

많은 기구를 설계할 수 있다. 수식은 가장 명확한 모델이다.

- 머신러닝에서는 이렇게 간결한 수식으로 표현되는 모델을 만드는 것은 아니지만 데이터에 기반하여 동작하는 모델을 사용한다.
- 현실 세계의 많은 현상은 수식으로 간단히 모델링하기 어렵고 과학적으로 증명할 수는 없다.
- 어느 고객이 불만이 많을 것인지, 어떤 영화가 관객을 많이 동원할지, 어떤 물건이 많이 팔리지, 어떤 메일이 스팸일지 등을 예측하는 모델은 수식으로 만들기는 어렵지만 컴퓨터 프로그램으로 구현하는 것을 가능하며 이제 그 성능이 꽤 쓸만하게 발전하고 있다.
- 머신러닝은 바로 이러한 데이터에 기반한 모델을 만드는 것이다.

## 모델 구조와 파라미터

- 사람의 뒷 모습을 멀리서 보고 남녀를 구분하는 모델을 생각해보자. 머리카락 길이가 20cm 이상이면 여성이라고 판단하는 모델이 잘 맞을까? 어느 지역에서는 이 모델이 잘 맞을 수 있지만 다른 곳에서는 30cm 이상으로 해야 정확도가 올라갈 수도 있다.
- 여기서 “머리카락 길이를 보고 남녀를 구분한다”는 것은 모델의 구조를 정한 것이고, 남녀를 구분하는 적정한 “머리카락의 길이”(위에서 20cm)는 모델 파라미터라고 한다. 즉, 모델은 구조와 파라미터로 구성된다.

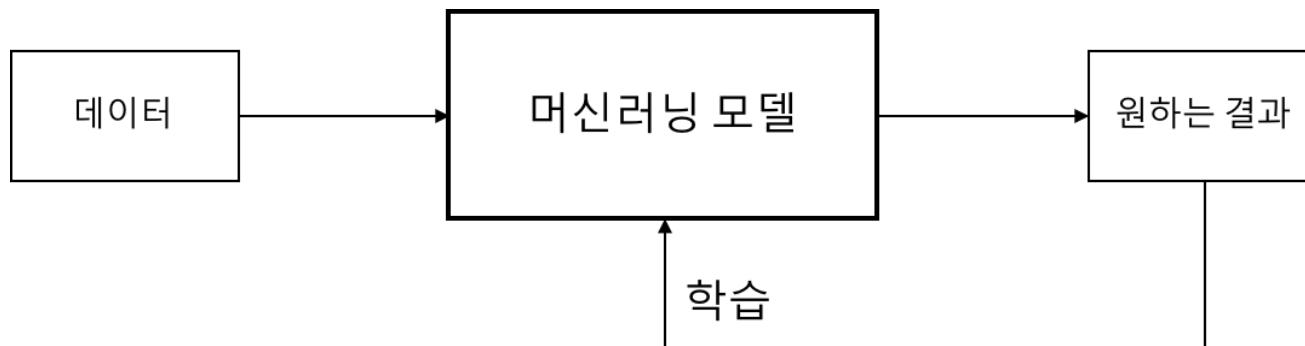
모델 구조: 모델의 동작을 규정하는 방법

모델 파라미터: 모델이 잘 동작하도록 정한 가중치 등 계수

- 모델의 구조는 프로그래머가 선택한다. 주어진 모델을 사용하여, 데이터를 기반으로 적절한 파라미터를 찾는 것은 머신러닝 프로그램이 수행한다.

## 머신러닝 기본 동작

- 머신러닝의 공통된 동작을 다음 그림과 같이 설명할 수 있다.
- 주어진 학습 데이터를 보고 원하는 동작을 잘 수행하는 모델을 만드는 것이 머신러닝의 기본적인 목표이다. 여기서 모델이 잘 동작한다는 것은 회귀나 분류를 정확하게 하는 것을 말한다.



# 훈련과 검증

## 훈련

- 모델이 데이터를 이용하여 학습하는 과정을 훈련 (training)이라고 한다. 모델 파라미터 값 (예를 들어 선형 회귀에서 가중치의 값)의 초기값은 보통 랜덤한 값을 준다.
- 따라서 초기 모델의 출력은 실제값과 다른 값을 예측하게 된다. 그러나 최적화 알고리즘에 의해서 파라미터값(예를 들어 가중치)을 최적화 알고리즘에 의해서 파라미터값(예를 들어 가중치)을 계속 갱신하여 모델의 예측값이 실제값에 수렴하도록 하는 것을 훈련이라고 한다.
- 모델 학습에 사용하는 데이터를 훈련 데이터 (training data)라고 한다.

- 모델을 훈련시킨 후에는 모델이 제대로 동작하는지 검증을 해야 하는데 이때는 모델을 만드는 데 사용하지 않은 새로운 데이터로 검증해야 한다.
- 연습문제로 수학 실력을 키우지만 실력을 제대로 평가하려면 연습문제가 아닌 새로운 문제로 테스트를 해야 하는 것과 같다.
- 머신러닝은 항상 모델을 훈련하는 과정과 새로운 데이터로 테스트를 하여 모델이 쓸만한지를 검증하는 두 단계로 이루어진다.

모델 학습 - 최적의 파라미터를 학습하는 과정

모델 검증 - 모델이 잘 동작하는지 확인하는 과정

## 검증

- 모델이 동작이 얼마나 우수한지를 검증할 때는 훈련 데이터로 해서는 안되며 훈련에 사용하지 않은, 새로운 검증 데이터(validation data)를 사용해야 한다.
- 보통 검증 데이터를 따로 제공하지 않으므로 훈련에 사용할 데이터의 일부를 검증용으로 미리 확보해야 한다. 이렇게 훈련에 사용하지 않고 남겨 두었다가 모델이 제대로 동작하는지 테스트할 때 사용하는 데이터를 hold-out 데이터라고 한다.
- 검증 데이터를 사용하여 최적의 모델이 만들어졌는지를 확인해야 한다. 머신러닝에 사용되는 데이터는 다음과 같이 세 가지로 분류된다.

훈련 데이터 - 모델 파라미터를 훈련하는데 사용

검증 데이터 - 과대적합이나 과소적합을 검사하고 최적 모델 구조를 찾는데 사용

테스트 데이터 - 모델의 성능을 최종적으로 테스트 하는데 사용

- 모델을 튜닝하는데 즉 적절한 환경변수인 하이퍼 파라미터를 찾기 위해서 검증 데이터를 사용하는 것이며 테스트 데이터는 모델의 최종 성능을 확인하기 위해서 사용된다.
- 가끔 검증 데이터를 테스트 데이터라고 부르기도 하는데 정확히는 위와 같은 차이가 있다.

## 과대적합

- 모델이 훈련 데이터에 대해서만 잘 동작하도록 훈련되어 새로운 데이터에 대해서는 오히려 잘 동작하지 못하는 경우를 과대적합(over fitting)되었다고 한다.
- 과대적합은 주어진 훈련 데이터를 너무 세밀하게 학습에 반영하여 발생하는 현상이다.
- 과대적합된 모델은 훈련 데이터에 대해서는 매우 우수한 성능을 보이지만 일반성이 떨어진다.
- 예를 들어 특정 학기에 붉은색 구두를 신은 학생들의 성적이 모두 A+를 받았다고 해서 항상 이것이 맞지 않는 것과 같다.

## 일반화

- 머신러닝에서는 과대적합을 피해서 일반적으로 잘 동작하게 모델을 만드는 것이 매우 중요하다. 이를 모델의 일반화(generalization)라고 하는데 모델이 새로운 임의의 일반적인 데이터에 대해서도 잘 동작하도록 해야 한다.
- 과대적합이 발생하는 원인은 훈련 데이터가 너무 적어서 모델이 학습을 충분히 할 수 없는 경우에 발생한다. 과대적합을 줄이려면 훈련 데이터를 많이 확보하여 다양한 경우를 대비하여 일반적인 모델을 만들어야 한다.

## 규제화

- 만일 학습할 데이터가 부족하다면 모델 구조를 좀 단순하게 만들어야 한다. 이렇게 모델이 일반성을 갖도록 기능을 제한하는 것을 모델에 제약을 가한다고 하여 규제화(regularization)라고 한다.
- 머신러닝에서는 일반화된 모델을 만든 것이 중요하며 적절한 규제 기법을 사용해야 한다.

## 과소적합

- 과대적합과 반대로 모델이 너무 간단하여 성능이 미흡한 경우를 과소적합 (under fitting)이라고 한다.
- 예를 들어 뒷모습만 보고 남녀를 구분하는 경우에, 남성대 여성의 숫자 비율이 9:1이라는 평균치를 미리 알고 있었다면, 아무런 알고리즘도 수행하지 않고 90%의 확률로 남성이로 예측하는 단순한 모델을 생각할 수 있다.
- 이는 과소적합한 모델로 개선의 여지가 있다. 회귀의 경우도 예를 들어 몸무게를 추정하는 경우에 몸무게 = (키 - 110)과 같은 간단한 모델을 사용한다면 너무 단순한 모델이 된다.
- 과소적합을 피하려면 좀 더 상세한 모델 구조를 사용해야 한다. 머신러닝에서

는 과대적합과 과소적합을 모두 피해야 하며 최적의 예측을 수행하는 모델을 만드는 것이 중요하다.

## 데이터의 대표성

- 학습에 사용할 훈련 데이터를 준비할 때에는 훈련 데이터가 미래에 나타날 가능성이 있는 모든 데이터의 특징을 반영하도록 구성해야 한다.
- 예를 들어 투표 결과를 예측하는 것이라면 실제 인구구성에 비례한 남녀구성비, 지역적인 구성비를 고려하여 훈련 데이터를 확보해야 한다.
- 전 국민대상 양케이트를 조사할 때에도 1천명을 대상으로 조사하는 이유는 적어도 1천명의 샘플이 있어야 연령대별, 성별, 지역별 특성을 골고루 포함하는 샘플을 얻을 수 있기 때문이다.
- 이렇게 실제 데이터 분포를 고려하여 훈련 데이터를 준비하는 것을 데이터의 대표성(representative)을 고려한다고 한다. 균형을 갖춘 데이터를 확보하

는 것은 매우 중요하며 잘 못된 데이터를 입력하면 잘 못 판단하는 모델을 만들게 된다.

- 이러한 내용을 고려하여 데이터를 수집하는 것을 계층적 샘플링(stratified sampling)이라고 한다. 예를 들어 어느 학교의 남녀학생 비율이 8:2라고 하자, 학생들에게 의견수렴을 정확히 하려면 샘플로 택한 학생의 남녀 비율이 8:2와 비슷하도록 준비해야 한다.

## 모델 구축 절차

- 머신러닝을 이용하는 과정은 다음과 같은 과정을 따른다. 먼저 해결할 문제에 적합한 머신러닝 모델을 선택한다. 모델에는 선형회귀모델, 결정트리, 신경망, SVM, 랜덤포레스트 등이 있다. 수행할 작업이 지도학습인지 비지도 학습인지도 파악해야 한다.
- 모델을 선택했으면 훈련 데이터로 모델을 학습시킨다. 학습이란 달리 표현하면 모델을 통한 예측 값이 실제 값과 잘 맞도록 하는 모델 파라미터를 구하는 것이다 (지도학습의 경우).
- 비지도 학습의 경우는 객관적인 성능 평가가 있는 것이 아니며 사람이 우선 판단을 해야 한다. 모델 학습 후에는 이 모델이 과대적합되었는지 또는 과소적합인지를 판단해야 한다.

- 과대적합이면 모델을 더 일반화해야 하고, 과소적합이면 모델을 더 상세하게 설계해야 한다. 마지막으로 모델을 실제 테스트 데이터에 적용하여 모델의 성능을 평가한다.
- 머신러닝의 일반적인 과정을 정리하면 다음과 같다.
  - 1) 모델 선택: 머신러닝 구조(알고리즘)를 선택
  - 2) 모델 학습(fit): 훈련 데이터를 이용하여 최적의 모델 파라미터를 구한다
  - 3) 모델 검증: 과대적합이나 과소적합을 검증하고 최적의 모델 하이퍼 파라미터를 선택
  - 4) 모델 평가: 테스트 데이터에 모델을 적용(predict)하고 성능을 평가(score)

## 모델 선택

- 문제가 주어지면 처음에 할 일은 적절한 모델을 선택하는 것이다. 문제에 따라 최적의 모델은 다르다. 문제를 보고 바로 적절한 모델을 찾아내는 것은 많은 경험과 실험이 있어야 할 수 있는 일이다.

## 머신러닝 알고리즘

- 주요 머신러닝 알고리즘을 아래 표에 나열했다. 참고로 분류와 회귀 알고리즘은 서로 교차하여 사용할 수 있다. 예를 들어, 선형 회귀 알고리즘은 수치 예측인 회귀에만 사용하는 것이 라 분류에도 사용할 수 있다. 결정 트리 알고리즘도 분류에만 사용하는 것이 아니라 회귀에도 사용할 수 있다.

## 머신러닝 유형별 대표적 알고리즘

머신러닝 유형		알고리즘
지도학습	분류	kNN, 베이즈, 결정 트리, 랜덤포레스트, 로지스틱회귀, 그라디언트부스팅, 신경망
	회귀	선형회귀분석, SVM, 신경망
비지도학습	군집화	k-means, DBSCAN
	변환	스케일링, 정규화, 로그변환
	차원축소	PCA, 시각화

## 모델의 동작 성능

- 정확도 등 모델의 동작 성능 외에 모델의 동작 속도도 중요한 요소이다. 모델을 학습하는데 걸리는 시간과 작성된 모델을 실제로 데이터를 분류 또는 예측하는데 걸리는 시간 두 가지를 최소화해야 한다.
- 일반적으로 모델이 정교하고 복잡할수록 성능은 좋아지지만 모델을 만들거나 적용하는데 시간이 오래 걸린다.

학습 시간 - 모델을 만드는데 걸리는 시간

동작 속도 - 모델을 적용하는데 걸리는 시간

## 심슨 패러독스

- 통계 분석은 결과 해석 방법에 따라 서로 상반된 결과를 얻기도 한다.
- 두 회사가 각각 자사 제품의 불만률이 더 낮다고 주장한 예를 보자. A사와 B사가 서울과 춘천에서 판매한 전체 제품 1100개를 조사해보니, A사 제품의 불만률은 3%, B사의 불만률은 8%였다. 즉, A사 제품이 불만율이 전체적으로 낮았다. A사는 이를 대대적으로 홍보했다.
- 그런데, 같은 데이터를 가지고 B사에서 다시 분석을 해보았는데, 서울과 춘천에서의 통계를 나누어서 각각의 통계치를 보았다. 서울에서의 불만률이 A사가 10%, B사는 8%로 B사의 실적이 더 좋았고, 춘천에서도 A사의 불만률은 2%, B사의 불만률은 1%로 B사의 실적이 역시 좋았다.

- 즉, 서울과 춘천 두 도시에서 모두 B사가 승리했으니 B사의 제품이 우수하다고 볼 수 있고 이를 홍보했다. 이들은 모두 같은 데이터를 가지고 분석을 했는데 어떻게 정 반대의 결과를 얻었을까?

두 도시에서 A사와 B사의 불량률 비교와 전체 불량률 비교

도시	A 사	B 사
서울	판매량 90, 불량품 10 (불량률 = $10/100 = 10\%$ )	판매량 920, 불량품 80 (불량률 = $80/1000 = 8\%$ )
춘천	판매량 980, 불량품 20 (불량률 = $20/1000 = 2\%$ )	판매량 99, 불량품 1 (불량률 = $1/1000 = 1\%$ )
	A사 총 불량률 $30/1100 = 3\%$	B사 총 불량률 $81/1100 = 8\%$

- 앞 표의 맨 아래 줄에 나타낸대로 전체 제품 1100개의 결과를 비교하면 A사는 총 불량률이 3%이고 B사는 8%인 것이 맞다. 그러나 각 도시별로 비교하면 두 도시에서 모두 B사의 불량률이 A사보다 낮다.
- 이렇게 언뜻 이해가 잘 되지 않는 결과를 얻게 된 원인은 A사는 춘천에서 제품을 많이 팔았고(1000개) B사는 서울에서 물건을 더 많이 팔았기 때문이다. 그리고 제품을 많이 판매한 곳의 불량률이 각각 2%와 8%로 큰 차이가 났기 때문에 전체적으로 이것이 결정적인 영향을 준 것이다.
- 각자의 주장에 틀린 점은 없다. 단지 데이터의 해석 방법이 달랐을 뿐이다.
- 이렇게 같은 데이터를 가지고도 분석 방법에 따라 해석이 달라질 수 있는 예를 심슨(Simpson) 파라독스라고 하며 이러한 일은 종종 벌어진다.

## 7. 예측 모델

- 지도학습은 정답을 예측하는데 사용된다. 회귀, 분류 모델을 만드는 방법과 대표적인 예측 모델 알고리즘을 설명하겠다.

## 7.1. 선형 회귀

- 선형 회귀 모델은 가장 간단하면서도 성능이 우수하여 오랫동안 널리 사용된 모델이다. 먼저 선형 시스템의 특징을 살펴보겠다.

### 선형 모델

- 입력과 출력이  $y = ax + b$ 의 선형관계인 시스템을 선형 시스템이라고 한다.
- 커피 자판기는 선형 시스템인가? 입력이 어느 한도를 넘으면 선형 관계가 성립하지 않는다. 일부분 구간에서만 선형이라고 가정할 수 있다.
- 선형 시스템이어야 정확한 예측이 가능하다. 앞에서 로그변환, 역수변환, 정규분포를 가정한 구간 나누기 등을 수행하여 입력 신호를 변형한 이유가

바로 선형적인 입출력 관계를 맞추기 위해서이다.

- 선형 회귀 모델을 포함한 모든 머신러닝 모델에서는 예측값과 실제 값과의 차이, 즉 오차를 줄이는 방향으로 모델을 최적화 한다.

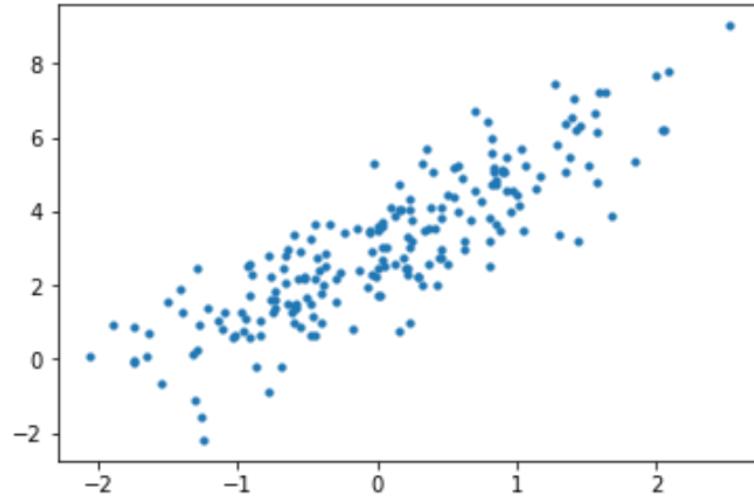
## 예제

- 평균이 0이고 표준편차가 차 1일 랜덤수 200개를 발생하고 이를 벡터  $x$ 에 담는다. 종속변수는  $y = 2x + 3$  의 관계로 얻고 이를 스캐터 플롯으로 그리면 아래와 같다.

```
import numpy as np  
import matplotlib.pyplot as plt
```

```
n_samples = 200
x = np.random.randn(n_samples)

# 계수 및 절편
w = 2
b = 3
# 노이지
y = w*x + b + np.random.randn(n_samples)
plt.scatter(x,y, s=10)
```



- 이를 sklearn 라이브러리를 사용하여 계수와 절편을 구하면 아래와 같다.

```
leg = LinearRegression()  
leg.fit(x.reshape(-1,1), y)
```

```
print(leg.coef_)

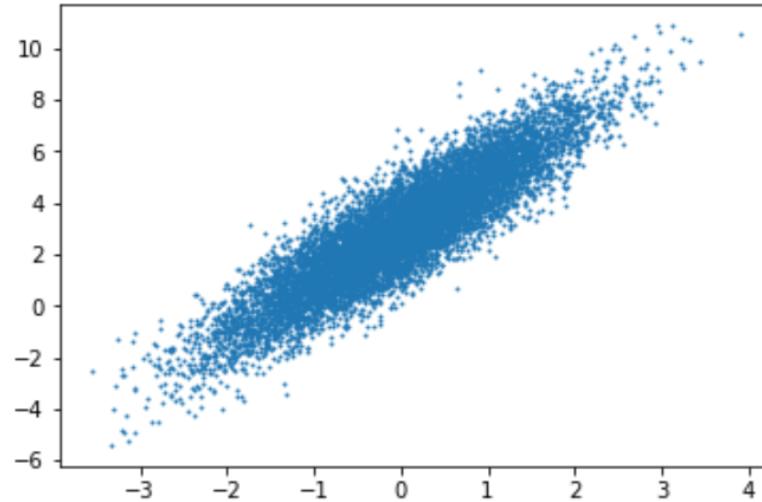
print(leg.intercept_)

=>

[2.05278049]

2.980708846242261
```

- 위의 결과를 보면 2, 3의 근사 값을 구한 것을 알 수 있다.
- 만일 같은 분포를 갖는 데이터 10000개를 사용하면 아래와 같은 분포를 가지면 파라미터를 더 정확하게 예측할 수 있다.



```
n_samples = 200  
...  
print(leg.coef_)  
print(leg.intercept_)  
[2.00139813]  
3.0032040293016733
```

## 학습속도 고려

- 변수를 2개로 하고  $y = 2 * x_1 + 3 * x_2 + 4 + \text{noise}$  로 종속 변수를 200개씩 만든 경우의 학습 프로그램은 아래와 같다.

```
n_samples = 200
```

```
x1 = np.random.randn(n_samples)
```

```
x2 = np.random.randn(n_samples)
```

```
y = 2 * x1 + 3 * x2 + 4 + np.random.randn(n_samples)
```

```
leg = LinearRegression()
```

```
X = pd.DataFrame({'x1':x1, 'x2':x2})
```

```
leg.fit(X, y)
```

```
print(leg.coef_)

print(leg.intercept_)

=>

[1.96016685 3.09308127]

3.9467896173580646
```

## 직접 구현하는 예

- 이포크를 10회 실행하고 학습률은 임의로 0.7로 설정하였다.

```
num_epoch = 10

lr = 0.7
```

```
w1 = np.random.uniform()
w2 = np.random.uniform()
b = np.random.uniform()

for epoch in range(num_epoch):
    y_pred = w1*x1 + w2*x2 + b
    error = np.abs(y_pred - y).mean()
    print(f'{epoch:2} w1 = {w1:.6f}, w2 = {w2:.6f}, b = {b:.6f} , error = {error:.6f}')

    w1 = w1 - lr*((y_pred - y)* x1).mean()
    w2 = w2 - lr*((y_pred - y)* x2).mean()
    b = b - lr*((y_pred - y)).mean()

=>

0 w1 = 0.531587, w2 = 0.257861, b = 0.458581 , error = 3.726568
```

1 w1 = 0.888426, w2 = 1.795199, b = 2.876638 , error = 1.601875  
2 w1 = 1.424048, w2 = 2.421399, b = 3.571785 , error = 0.998551  
3 w1 = 1.705597, w2 = 2.740087, b = 3.808276 , error = 0.822332  
4 w1 = 1.838922, w2 = 2.908563, b = 3.894370 , error = 0.780016  
5 w1 = 1.901678, w2 = 2.997302, b = 3.926604 , error = 0.769709  
6 w1 = 1.931594, w2 = 3.043638, b = 3.938878 , error = 0.765982  
7 w1 = 1.946063, w2 = 3.067660, b = 3.943625 , error = 0.764516  
8 w1 = 1.953149, w2 = 3.080048, b = 3.945494 , error = 0.764313  
9 w1 = 1.956654, w2 = 3.086413, b = 3.946246 , error = 0.764228

## 손실 함수

- 모델의 학습 속도를 높이기 위해서 실제로는 오차 자체를 줄이는 것이 아니라 오차로부터 계산된 손실함수(loss function)을 정의하고 이를 줄이는 방향으로 모델을 학습시킨다.
- 손실함수는 한 번 측정한 값으로 계산하기도 하지만 보통 배치 단위로 측정하고 배치 단위로 학습을 한다.
- 한 샘플마다 오차를 측정하고 손실함수를 구하고 학습을 하는 것이 아니라 예를 들면 20 개의 배치 단위로 측정하고 학습을 한다.
- 회귀분석에서 주로 사용하는 손실함수로는 오차 자승의 합의 평균치(MSE: mean square error)이다. MSE를 수식으로 표현하면 다음과 같다.

$$MSE = \sum_{k=1}^N (y - \hat{y})^2$$

- 위의 식에서는 배치 크기를 N명으로 하는 예를 들었으나 배치 크기를 얼마로 선택하는 것이 좋은지는 상황에 따라 다르다. 한명 단위로 예측하고 학습할 수도 있고 100명 단위로 할 수도 있다.
- 배치 크기와 같은 변수를 하이퍼 파라미터라고 한다. 하이퍼 파라미터는 일종의 환경 변수로 사람이 선택하는 변수이며 학습으로 갱신하는 계수는 아니다. 학습으로 갱신되는 변수는 “파라미터”라고 부른다.
- 머신러닝이란 손실함수를 줄이는 방향으로 모델의 파라미터를 조정하는

작업이라고 정의할 수 있다.

- 여기서 파라미터를 조절하는 작업을 최적화(optimizer)라고 부른다. 최적이라는 의미는 모델 예측 값이 실제 값을 최대한 잘 맞추도록 파라미터를 최적화한다는 의미를 갖는다.
- 그런데 오차를 줄인다면 절대치의 합을 줄여도 되는데 오차의 자승을 구하고 이를 더한 후에 평균을 구하고 루트를 적용하는 이유는 무엇일까? 그 이유는 오차가 크게 나는 샘플에게 페널티를 강하게 주기 위해서이다.

## 다중 선형회귀

- 위의 예와 같이 하나의 변수 (위에서는 연령) 만을 특성으로 사용하지 않고 여러 특성을 이용한 회귀 예측이 가능한데 이를 다중 선형 회귀 (multiple linear regression)라고 한다.
- 예를 들어 혈압을 예측하는데 연령 뿐 아니라, 몸무게를 같이 고려하여 더 정확한 값을 예측할 수 있다. 다중 선형 회귀식은 아래와 같다.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$$

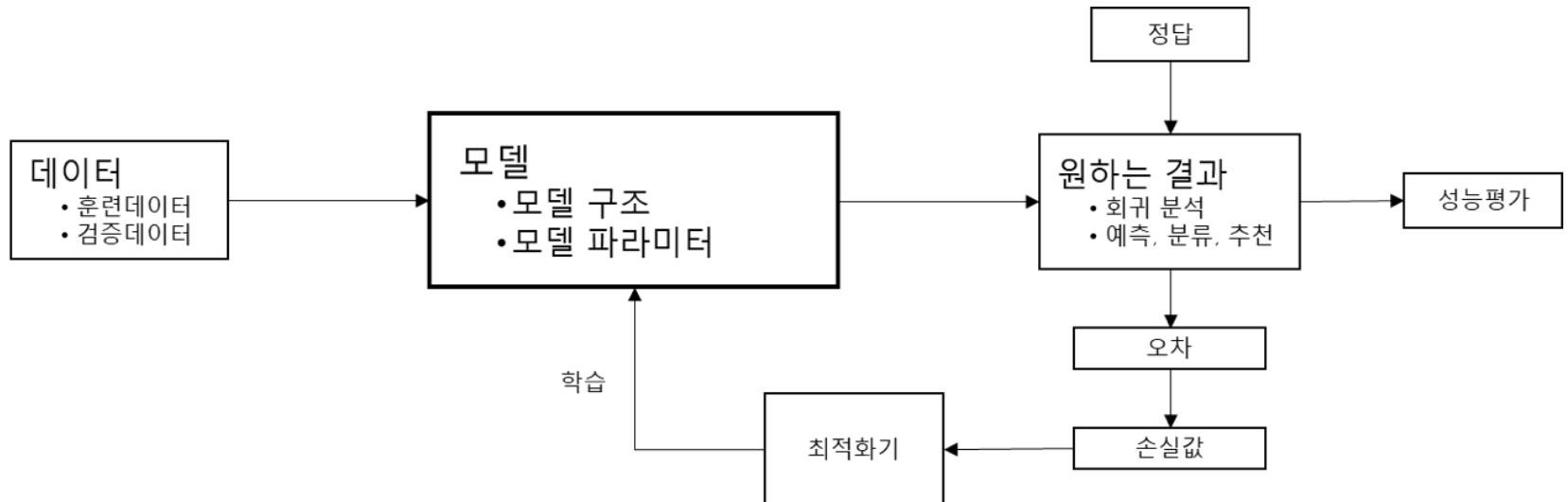
- 위에서  $p$ 는 회귀분석에 사용되는 변수의 총 갯수이다 (위의 예에서는 혈압을 예측하는데 연령, 몸무게의 2개 항목을 사용하므로  $p=2$ 가 됨). 혈압은 종속 변수이고 연령, 몸무게가 독립 변수가 된다.

## **독립변수 종속변수**

- 종속 변수를 다른 표현으로 목적(target) 변수, outcome 변수, response 변수, 또는 label이라고도 부르며, 독립 변수를 predictor, 설명 변수 (explanatory variable) 또는 feature라고도 한다.

# 머신러닝 모델

- 앞에서 설명한 오차, 손실함수, 최적화, 모델 파라미터의 관계를 아래 그림에 나타냈다.



## 배치와 이포크

- 예를 들어 총 1000개의 데이터로 예측 모델을 만들 때 한번에 1000개의 데이터를 모두 입력하여 손실함수를 구하고 학습을 시키려면 컴퓨터 용량이 부족하다.
- 따라서 일정 크기의 배치 단위로 나누어 학습을 시키고 MSE와 같은 손실함수도 배치 단위로 계산한다. 예를 들어 배치 크기를 50 샘플로 하였으면 1000개의 데이터를 20번에 나누어 학습시킬 수 있다. 즉, 한 번에 학습하는 샘플 수를 배치크기라고 한다.
- 학습에 주어진 1,000개의 데이터 셋을 모두 학습에 사용하였어도 아직 최적의 모델 파라미터를 찾지 못했으면 주어진 데이터를 다시 학습에 여러 번 반복하여 사용할 필요가 있다.

- 우리가 수학문제집을 한번만 풀어보고 학습하는 것이 아니라 여러번 반복하여 문제를 풀 듯이, 머신러닝도 주어진 데이터를 여러 번 재사용하여 학습을 한다면 필요한 만큼 데이터 사용을 반복해야 한다.
- 주어진 데이터 전체를 한번 사용하는 것을 이포크(epoch)라고 한다.

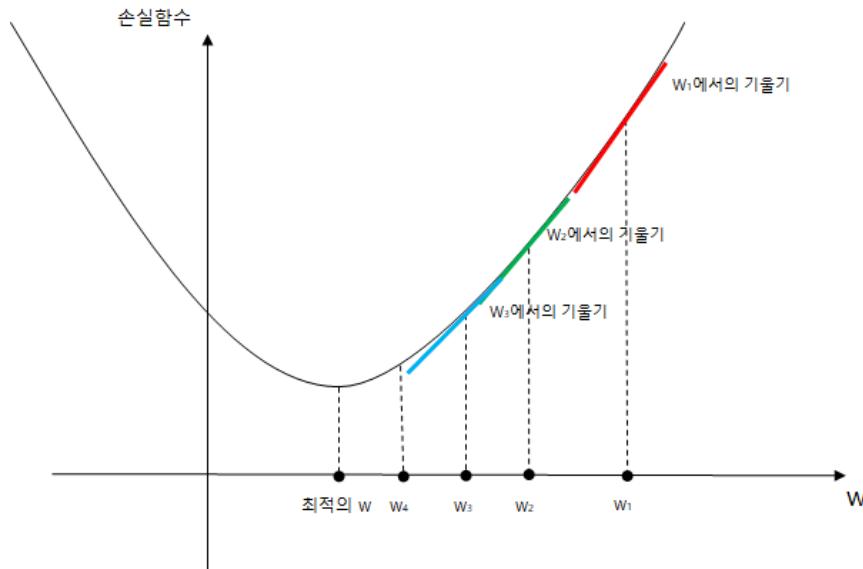
## 모델 최적화

- 모델의 최적화란 학습을 통하여 파라미터 (예를 들어 선형회귀 모델의 가중치)를 최적의 값으로 수렴시키는 것을 말한다.
- 여기서 최적의 값이란 손실함수를 가장 줄이는 계수 값을 말한다. 최적화 알고리즘으로는 경사하강법(GD: Gradient Descent)이 널리 사용되는데 이는 선형회귀 모델뿐 아니라 대부분의 머신러닝에서 사용하는 기본적인 최적화 방법이다.

## 경사하강법

- 경사하강법(GD)이란 손실함수를 계수에 관한 그래프로 그렸을 때 최소값으로 빨리 도달하기 위해서는 현재 위치에서의 기울기(미분값)에 비례하여 반대방향으로 이동하는 방식을 말한다.
- 아래 그림은 계수에 대해서 손실함수를 간단히 2차원으로 가정하고 그린 것이다(실제로는 특성의 수만큼 높은 고차원의 공간에서 이루어진다).
- 현재 계수의 값의 위치가  $(t_1)$ 이라고 하자. 여기서 기울기가 현재 (+)값이고 기울기가 큰 편이다. 이때는 음의 방향(좌측)으로 이동하여 다음번 계수를 선택해야 한다. 다음 위치( $t_2$ )에서는 기울기가 아직 (+)지만 크기는 조금 감소했다.

- 따라서 좌측으로 다음 계수를 정하되 이동하는 폭을 좁혀야 한다. 최저점에 도달하면 기울기가 0이 되며 더 이상 이동할 필요가 없다.



경사강하법의 동작원리. 가중치는  $W_1$ 에서 출발하여 최적의 값에 도달한다.

- 경사하강법의 동작을 수식으로 표현하면 다음과 같다.

$$W_i = W_{i-1} - \eta \text{Grad}(i)$$

- 위에서  $\text{Grad}(i)$ 는 시점  $i$ 에서 손실함수의 기울기이고  $\eta$ 는 학습 속도를 조절하는 변수로 학습률(learning rate)이라고 한다. 계수  $W$ 의 초기 값은 보통 랜덤하게 준다.
- 경사하강법(GD)의 원리를 비유하면, 예를 들어 산에서 낮은 곳으로 내려오려면 경사가 내려가는 방향 (즉 음의 방향)으로 이동하되 기울기의 크기에 비례하여 빨리 움직이는 것과 같다.
- 기울기가 크다면 산 정상에 있는 셈이니 빨리 움직이고 기울기가 거의

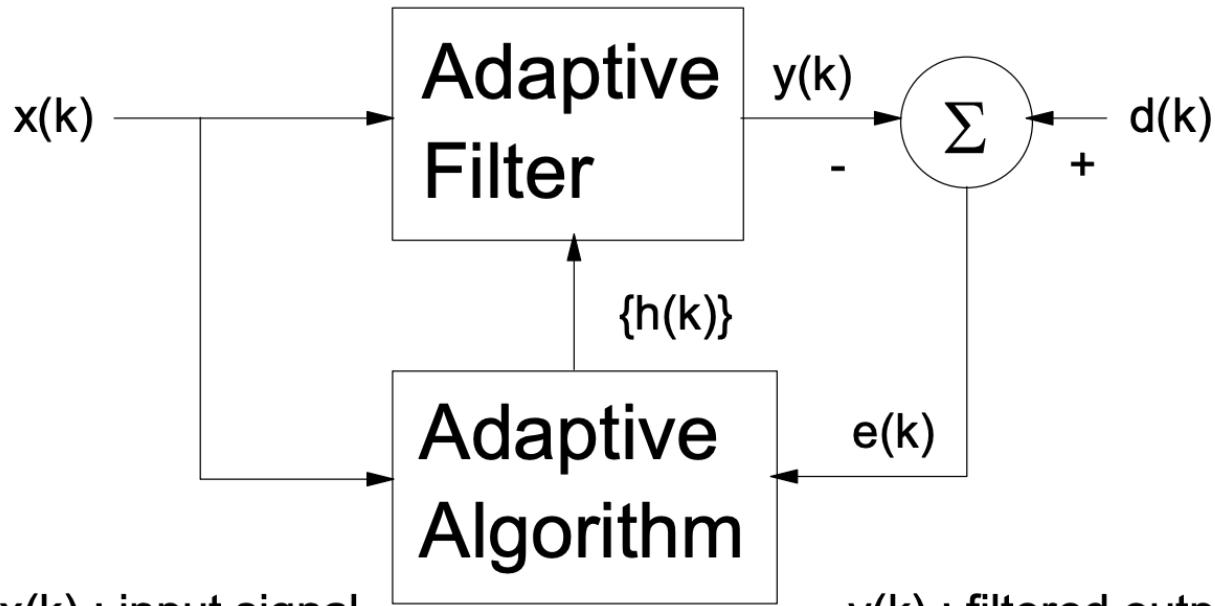
없다면 평지에 다 내려 온 것이라고 보는 것과 같다. 즉, 계수를 업데이트 할 때 학습률에 기울기를 곱한 값만큼 변화를 주는 방법을 GD 방법이라고 한다.

- 계수를 업데이트 하는 속도를 조정하는 학습률이 너무 작으면 수렴하는데 시간이 오래 걸리지만 최저점에 도달했을 때 흔들림 없이 안정적인 값을 얻게 되고, 반대로 학습률을 너무 크게 정하면 학습하는 속도는 빠르나 자칫하면 최저점으로 수렴하지 못하고 발산하거나 수렴하더라도 흔들리는 오차가 남아있을 수 있다.
- 학습 스케줄(learning schedule) 기법을 사용하면 초기에는 학습률을 크게 정하고 (학습률을 빠르게 하고) 오차가 줄어들면 학습률을 줄여서 안정상태(steady state)의 오차를 줄이는 방법을 사용할 수 있다.

- 마치 현재 수돗물의 온도를 조절하기 위해서 얼마나 손잡이를 돌려야 하는지 를 계산하는 것과 같다. 이때 사람마다 손잡이를 돌리는 속도가 다를 수 있다. 천천히 돌리는 것과 너무 빠르게 돌리는 것 모두 문제가 있으며 적절한 크기의 속도를 택해야 한다.
- 주의할 것은 경사하강법을 적용하려면 특성 변수들을 모두 스케일링해야 한다.
- 특성 값마다 크기의 편차가 크면 특정 변수에 너무 종속되어 동작할 수 있고 이로 인해 수렴속도가 직선이 되지 않고 오래 걸릴 수가 있다. 즉, 경사하강법에서는 모든 특성 변수에 대해 비슷한 크기의 기울기를 갖게 해야 한다.

- 경사하강법(GD)은 크게 배치(Batch) GD, 확률적(Stochastic) GD (SGD) 두 가지가 있다. 일반적으로 배치 GD방식을 많이 상용하는데 이는 적절한 크기의 배치단위로 입력 신호를 나누어 경사하강법을 적용하는 방식이다.
- SGD (확률적 경사하강법)은 한 번에 한 샘플씩 랜덤하게 골라서 훈련에 사용하는 방법이다. 즉 샘플을 하나만 보고 계수를 조정한다.
- 계산량이 적어 동작속도가 빠르고, 랜덤한 방향으로 학습을 하므로 전역 최소치를 가능성이 높아진다. 처리해야 할 데이터 양이 매우 많아도 데이터를 하나씩 처리하므로 동작에 문제가 없다. 그러나 SGD 방법은 매 샘플이 너무 랜덤하여 방향성을 잃고 수렴하는데 시간이 오래 걸릴 가능성도 있다.

## 경사하강법 수식



$x(k)$  : input signal

$y(k)$  : filtered output

$d(k)$  : desired response

$h(k)$  : impulse response of adaptive filter

The cost function may be  $E\{e^2(k)\}$  or  $\sum_{k=0}^{N-1} e^2(k)$

$$\begin{aligned}
\underline{W}(n+1) &= \underline{W}(n) - \mu \frac{\partial e^2(n)}{\partial \underline{W}(n)} \\
&= \underline{W}(n) - \mu \frac{\partial e^2(n)}{\partial e(n)} \cdot \frac{\partial e(n)}{\partial \underline{W}(n)} \\
&= \underline{W}(n) - 2\mu e(n) \cdot \frac{\partial [d(n) - \underline{W}^T(n) \cdot \underline{X}(n)]}{\partial \underline{W}(n)}, \\
&= \underline{W}(n) + 2\mu e(n) \underline{X}(n)
\end{aligned}$$

$$\frac{\partial \underline{A}^T \cdot \underline{B}}{\partial \underline{A}} = \underline{B}$$

## 성능지표

- 모델의 성능을 객관적으로 비교 평가하려면 수치화된 기준이 있어야 한다. 손실함수를 정하는 목적은 모델을 훈련시킬 때의 기준으로 삼기 위해서이다. 모델은 손실함수를 최소화 하는 방향으로 학습한다.
- 모델 성능지표란 이렇게 만든 모델이 궁극적으로 얼마나 잘 동작하는지를 평가하는 척도이다.
- 예를 들어 과속단속을 하여 교통사고율을 줄이는 것이 목적인 경우, 자동차의 속도를 측정하여 과속단속을 하는 것은 손실함수를 줄이는 것이다.
- 그러나 과속을 줄이는 작업을 통해서 궁극적으로 교통사고를 줄이는 것이 목적인 것이다. 따라서 평가 지표는 교통사고율을 측정해야 한다.

- 회귀에서는 손실함수로 MSE를 주로 사용한다고 하였다. 그런데 MSE 값은 성능 지표로 사용하기에는 부족하다.
- 예를 들어 키를 예측하였는데 MSE 값이 5.7cm이 나왔다고 하면 이것의 성능이 얼마나 우수한지 다른 경우와 비교하기 어렵다. 몸무게를 예측하였는데 MSE값이 3.8kg이라면 얼마나 우수한 것인가?
- 키와 몸무게의 MSE의 의미가 다르므로 MSE만 봐서는 데이터 성격에 따라서 모델의 성능이 얼마나 좋은지 객관적으로 평가하기가 어렵다. 이를 성능 측정 지표로 사용하기가 어렵다.

## R-sqaurer

- 회귀분석에서는 동일한 개념의 수치로 모델의 성능을 비교하기 위해서  $R^2$ 값을 주로 사용한다.
- 모델이 실제 값을 잘 예측하면  $R^2$ 는 1로 수렴하고 모델이 평균치 예측 정도의 수준으로 대충 예측하면  $R^2$ 는 0에 가까운 값이 된다.
- $R^2$ 의 정의는 아래와 같다.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y}_i)^2}$$

- 위에서  $y_i$ 는 실제 값,  $\hat{y}_i$ 는 예측한 값,  $\bar{y}$ 는 샘플의 평균 값을 나타낸다.
- 예를 들어 키를 예측한다고 할 때, 미리 알고 있는 키의 평균치가 170cm이면, 모든 예측을 동일하게 170이라고 예측하는 모델을 생각해보자. 위에서 모든  $\hat{y}_i$ 가 평균치인  $\bar{y}$ 가 되므로 분수식의 분자와 분모가 같아서  $R^2 = 0.0$  된다.
- 만일 모델이 매우 정확해서 모든 예측이 실제값과 완전히 일치하는 경우는 실제값  $y_i$ 와 예측치  $\hat{y}_i$ 가 일치하므로 분수식의 분자가 0이 되고 따라서  $R^2$  값은 1.0이 된다.
- 즉, 예측을 평균치로 하면, 즉, 가장 기본적으로 예측을 하면  $R^2 = 0.0$

되고, 모든 예측을 실제값으로 정확히 하면  $R^2=1$  이 된다.

- 만일 평균치 예측만도 못한 모델을 만들면 분수식의 분자가 분모보다 커질 수 있고  $R^2$ 는 음수가 된다.
- 회귀분석의 성능 평가로  $R^2$ 을 사용하면 측정하는 데이터의 종류와 값의 크기 범위와 관계없이 성능 평가를 객관적으로 할수 있게 된다.
- 손실함수는 모델이 학습할 때 줄이는 대상이면 작을수록 바람직한 방향이다. 성능 지표(metric)로서 이는 클수록 좋은 값으로 정한다.
- 아래에 회귀분석에서 손실함수와 성능평가 지표에 대한 정의와, 대표적인 값을 정리했다.

손실함수와 성능평가지표의 정의와 자주 사용되는 예

	손실함수	성능평가지표
정의	손실함수를 줄이는 방향으로 모델이 학습을 함	성능을 높이는 것이 머신러닝을 사용하는 목적임
회귀 모델의 대표적인 값	MSE (오차 자승의 평균치)	$R^2$

## 선형 회귀 예제

- 붓꽃 데이터를 사용하여 꽃받침의 길이를 보고, 꽃잎의 길이를 수치로 예측하는 프로그램은 아래와 같다.

### 데이터 준비

```
from sklearn import datasets  
iris = datasets.load_iris()  
print(iris.feature_names)  
## 출력  
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

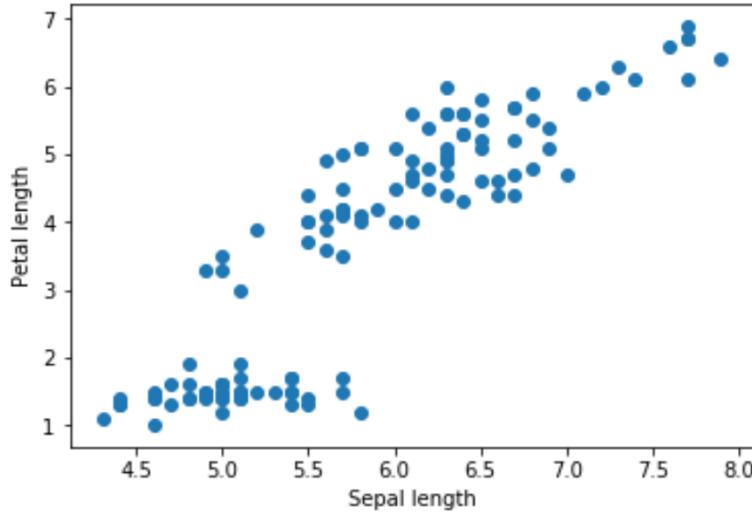
- 네 가지 속성을 알 수 있다. 독립변수로 꽃받침의 길이를 선택하고, 예측할

종속변수로 꽃잎의 길이를 선택하겠다. 샘플로 앞의 5개만 출력하면 아래와 같다.

```
X_all = iris.data  
X = X_all[:, 0]  
y = X_all[:, 2]  
  
print(X[0:5])  
print(y[0:5])  
# 출력  
[ 5.1  4.9  4.7  4.6  5. ]  
[ 1.4  1.4  1.3  1.5  1.4]
```

- 훈련과 테스트 데이터를 나눈다. 훈련 데이터의 분포를 그려보았다.

```
from sklearn.cross_validation import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)  
  
%matplotlib inline  
import matplotlib.pyplot as plt  
  
plt.scatter(X_train, y_train, marker='o')  
plt.xlabel("Sepal length")  
plt.ylabel("Petal length")
```



## 모델 구축

- 선형 모델인 LinearRegression를 사용하여 모델을 학습시키는 코드이다. LinearRegression는 손실함수로 오차의 자승의 합의 평균인 MSE를 사용하며 경사하강법의 최적화 알고리즘을 사용하는 1차 선형 모델이다.

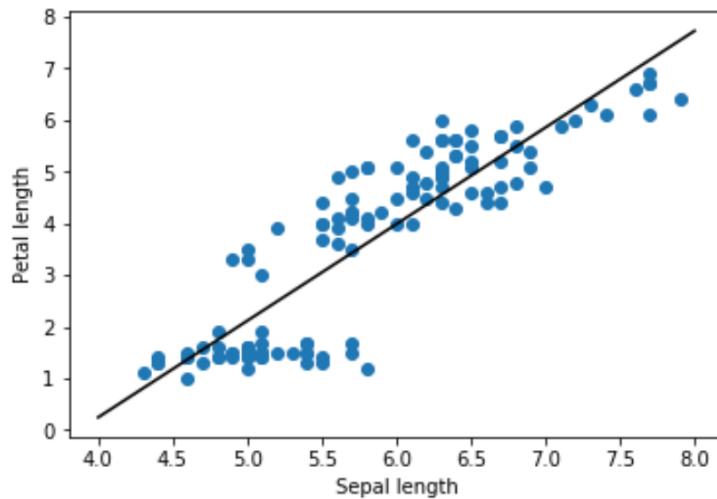
- 학습된 1차 선형 모델의 계수(기울기)와 절편을 구하면 아래와 같다.

```
from sklearn.linear_model import LinearRegression  
linr=LinearRegression()  
linr.fit(X_train.reshape(-1,1), y_train.reshape(-1,1))  
print("Score : {:.3f}".format(linr.score(X_test.reshape(-1,1), y_test)))  
  
## Score: 0.653  
  
print(linr.coef_)  
print(linr.intercept_)  
## 출력  
[[ 1.8699969]]
```

[-7.23331523]

```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt

plt.scatter(X_train, y_train, marker='o')
plt.xlabel("Sepal length")
plt.ylabel("Petal length")
XX = np.linspace(4, 8, 200)
plt.plot(XX, linr.coef_ * XX + linr.intercept_, "k-")
```



- 위에서는 score를 측정하였고 이는 R<sup>2</sup> 값을 알려준다.

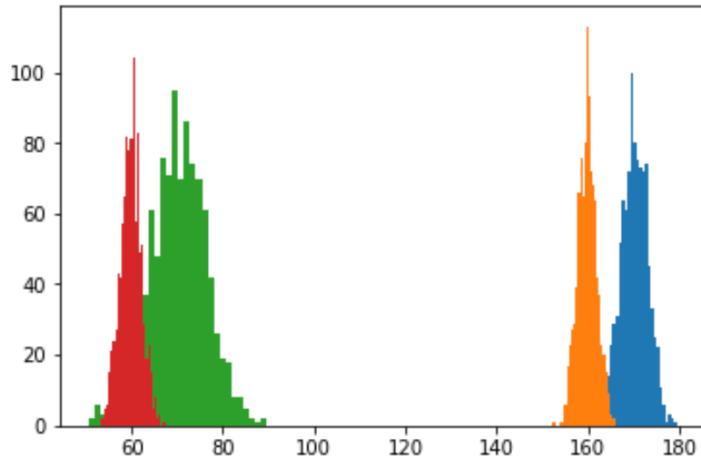
## 성별에 따른 몸무게 예측

- 키로부터 몸무게를 예측하는 예를 들어보겠다. 먼저 가상의 샘플 데이터를 만들어 사용하겠다.

```
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.linear_model import LinearRegression, LogisticRegression  
from sklearn.cross_validation import cross_val_score, train_test_split
```

```
n_samples = 1000  
x1 = 3*np.random.randn(n_samples) + 170  
x2 = 2*np.random.randn(n_samples) + 160
```

```
y1 = 2*x1 - 270 + 2*np.random.randn(n_samples) # 평균 70kg  
y2 = 1*x2 - 100 + np.random.randn(n_samples) # 평균 60kg  
plt.hist(x1, bins=30)  
plt.hist(x2, bins=30)  
  
plt.hist(y1, bins=30)  
plt.hist(y2, bins=30)  
plt.show()
```



- 남성에 대해서만 예측하는 모델을 만드는 코드는 아래와 같다. 결과를 보면, 계수 2과 절편 -270을 정확히 찾았으며 score도 0.89로 높다.

```
X_train, X_test, y_train, y_test = train_test_split(x1, y1, test_size=0.2)
leg1 = LinearRegression()
```

```
leg1.fit(X_train.reshape(-1,1), y_train)

print(leg1.coef_)
print(leg1.intercept_)
```

```
print(leg1.score(X_test.reshape(-1,1), y_test))
```

=>

[2.00043952]

-270.11660782913015

0.8931096360424833

- 남성과 여성 데이터를 합하여 회귀 분석을 하면 아래와 같다.

```
x = np.concatenate((x1, x2))
```

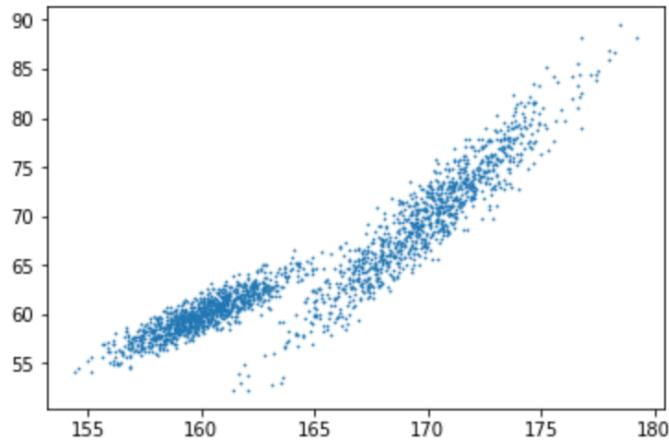
```
y = np.concatenate((y1, y2))
```

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
leg1 = LinearRegression()
leg1.fit(X_train.reshape(-1,1), y_train)

print(leg1.coef_)
print(leg1.intercept_)
print(leg1.score(X_test.reshape(-1,1), y_test))

=>
[1.13000512]
-121.48115678968445
0.8574293668531869
```

- plt.scatter(x, y, s=0.5)로 산점도를 그려보면 다음과 같다.



- 성별 컬럼을 추가하여 회귀분석을 하면 성능이 개선된 것을 알 수 있다.

```
X1 = pd.DataFrame({'height':x1, 'sex':0})  
X2 = pd.DataFrame({'height':x2, 'sex':1})  
X = pd.concat([X1, X2], ignore_index=True)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
leg = LinearRegression()
leg.fit(X_train, y_train)

print(leg.coef_)
print(leg.intercept_)
print(leg.score(X_test, y_test))

=>

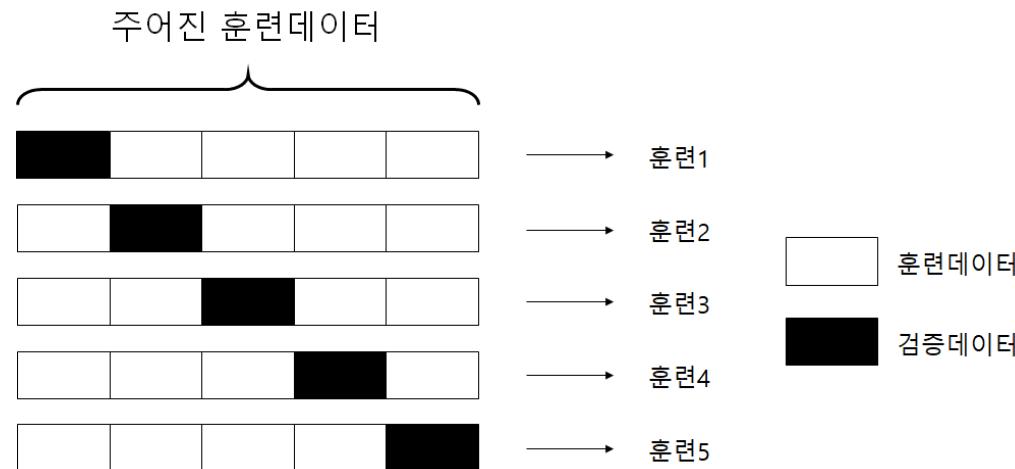
[1.68741033 6.89591868]
-216.88724022019412
0.9139554091334032
```

- 위의 결과를 보면  $1.687 * 키 + 6.896 * 성별 - 216.88$  로 선형 모델을 만든 것을 알 수 있고 score는 0.91로 높아졌다. 성별은 남성이면 0, 여성이면 1의 값을 갖는다.

## 교차 검증

- 앞의 예에서는 주어진 샘플데이터를 훈련 및 테스트 데이터로 나누는데 각각 70% 및 30% 비율로 나누었고 이를 랜덤하게 나누었다. 이를 위해서 sklearn이 제공하는 `train_test_split()`함수를 사용하였다.
- 그러나 이렇게 한번만 나누는 방식은 주어진 데이터 중 훈련에 사용되는 부분이 랜덤하게 정해지고 단 한번만 사용되므로 머신러닝 모델의 일반성이 떨어지는 단점이 있다. 즉, 주어진 데이터를 충분히 활용하지 못한다.
- 이러한 단점을 해결하기 위해서 교차 검증(cross validation)을 널리 사용한다. 교차검증이란 주어진 샘플 데이터를 훈련용 및 테스트 용으로 한번만 나누는 것이 아니라, 훈련 데이터를 다시 검증용 데이터로 나누어서 여러 번 검증하는 방법이다.

- 교차검증에서는 검증 데이터를 여러 세트 만들어야 하는데 아래 그림에 소개한 K-fold 교차 검증이 가장 널리 사용된다.



검증 데이터를 교대로 사용하는 교차검증의 개념

- 위의 그림에서는 주어진 전체 훈련 데이터를 5개의 그룹으로 나누고 그 중 네 파트 즉, 80%는 훈련용으로 사용하고 나머지 한 파트(20%)를 검증용으로 사용한다.
- 이렇게 하는 이유는 주어진 데이터 전체를 골고루 검증용으로 사용하여 모델의 동작을 보다 정교하게 확인하기 위해서이다. 교차검증의 개수를 보통 K로 표시하며 K는 5~10을 주로 사용한다.
- 검증데이터를 fold라고도 부르며 이 방식을 K-fold 교차검증이라고 부른다. 사이킷런에서는 `cross_val_score()` 함수가 교차검증을 자동으로 수행한다.
- 교차검증에서 주의할 것은 K개의 점수가 골고루 나오면 안정적인 모델이지만 만일 K개의 점수의 편차가 크면 모델이 불안정하다고 볼 수 있다.

검증 데이터를 어떤 부분을 택하는 가에 따라서 모델의 성능 차이가 크게 나는 것은 모델이 불안정하다는 뜻이다.

- 교차검증을 수행하는 목적은 만든 모델이 잘 동작하는지 성능을 검증하는 것이 목적이지 모델 자체를 잘 학습시키는 것은 아니라는 것을 주의해야 한다.
- 즉, 모델을 학습시키는 것은 모델선택, 알고리즘 선택, fit 등 별도의 작업에서 이루어지고 선택된 모델이 여러 가지 데이터 조건에도 잘 동작하는지 일반성을 갖는지를 확인하기 위해서 교차 검증을 하는 것이다.
- 위에 예제에 대해 교차검증을 해보면 아래와 같다.

```
from sklearn.cross_validation import cross_val_score, KFold
```

```
cv = KFold(X.shape[0], 5, shuffle=True)
print(cross_val_score(linr, X.reshape(-1,1), y.reshape(-1,1), cv=cv))
[ 0.70670544  0.80840397  0.83490685  0.61473755  0.75023557]
```

## 7.2. 선형 분류

- 두 가지 카테고리를 나누는 작업을 이진 분류(binary classification)라고 하고 세 개 이상의 클래스를 나누는 작업을 다중 분류(multiclass classification)라고 한다.
- 모든 분류 알고리즘(모델)은 기본적으로 이진 분류를 수행할 수 있다. 다중 분류 작업이 필요한 경우에는 이진 분류 알고리즘을 여러 번 사용하면 다중 분류 작업을 할 수 있다.
- 예를 들어 A, B, C를 나누려면 A와 {B, C}를 나누는 이진 분류기와 B와 {A, C}를 나누는 이진 분류기 등 총 3번의 이진 분류를 수행하는 방법이 가능하다.

- 목적 변수가 세 가지 이상의 값을 가지는 경우 sklean은 이를 자동으로 다중 분류라고 파악하고 내부적으로는 이진 분류를 여러 번 수행하여 다중 분류를 수행한다.
- 한편 랜덤포레스트나 나이브 베이즈와 같이 한 번에 다중 분류가 가능한 알고리즘도 있다.

## 붓꽃 이진 분류

- 여기서는 꽃 잎과 꽃 받침의 크기를 보고 붓꽃의 종류를 분류하는 프로그램을 작성하겠다. scikit-learn 라이브러리가 제공하는 데이터를 다음과 같이 출력해볼 수 있다.

```
from sklearn.datasets import load_iris  
iris = load_iris()  
print(type(iris)) # <class 'sklearn.utils.Bunch'>
```

- 사이킷런이 제공하는 붓꽃 데이터 셋인 iris는 번치(bunch) 타입의 데이터 구조이다. 번치 타입은 임의의 데이터들을 묶어서 하나로 저장하는데 유용한 방법이다.

- iris 데이터 셋이 포함하고 있는 내용 중에 속성의 이름들을 feature\_names 이름으로 제공하며 이를 다음과 같이 출력해보면 꽃받침의 길이, 꽃받침의 폭, 꽃잎의 길이, 꽃잎의 폭 등 네가지 속성으로 구성된 것을 알 수 있다.

```
print(iris.feature_names)
##
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

- iris의 타겟 변수들 즉, 꽃의 종류를 포함하고 있는 변수명은 target\_names이며 이를 확인해 보면 아래와 같다.

```
print(iris.target_names)
```

```
# ['setosa' 'versicolor' 'virginica']
```

- 타겟 변수에는 실제로 0, 1, 2 등의 값이 들어있는데 이 변수의 의미가 위의 target\_names가 가리키는 순서에 대응하는 이름을 가진다는 것을 나타낸다.
- 즉, 타겟 값이 0이면 이는 세토사(setosa)임을 나타낸다. 속성 데이터를 X 변수에, 타겟 데이터를 y 변수에 저장하고 각각의 크기를 shape 함수로 확인하면 아래와 같다. 샘플의 수는 150개이고, 속성의 종류는 4가지인 것을 알 수 있다.

```
X, y = iris.data, iris.target
```

```
print(X.shape, type(X)) # (150, 4) <class 'numpy.ndarray'>
```

```
print(y.shape, type(y)) # (150,) <class 'numpy.ndarray'>
```

- 샘플 3개만 아래와 같이 출력해 볼 수 있다.

```
print(X[0:3])  
## 출력  
[[ 5.1  3.5  1.4  0.2]  
 [ 4.9  3.   1.4  0.2]  
 [ 4.7  3.2  1.3  0.2]]
```

- 타겟 변수를 출력하면 아래와 같다.

```
print(y)  
## 출력
```

- 샘플마다 주어진 4개의 속성을 다 사용하지 않고 앞의 두개의 속성(꽃 받침의 길이와 넓이)만 사용하여 분류를 수행해보겠다. 앞의 두 컬럼만 취하여 새로운 변수 X2에 저장하였다.

```
X2 = X[:, :2]
```

- 세 개의 샘플만 출력해 보면 아래와 같이 두 개의 컬럼만 가진 것을 볼

수 있다.

```
print(X2[0:3])
```

## 출력

```
[[ 5.1  3.5]
```

```
[ 4.9  3. ]
```

```
[ 4.7  3.2]]
```

- 이제 데이터를 훈련 데이터와 테스트 데이터로 나누기 위해서 `train_test_split()` 함수를 사용하겠다. 아래에서 입력 데이터로 `X`가 아니라 `X2`를 사용한 것을 주의해야 한다.

```
from sklearn.cross_validation import train_test_split
```

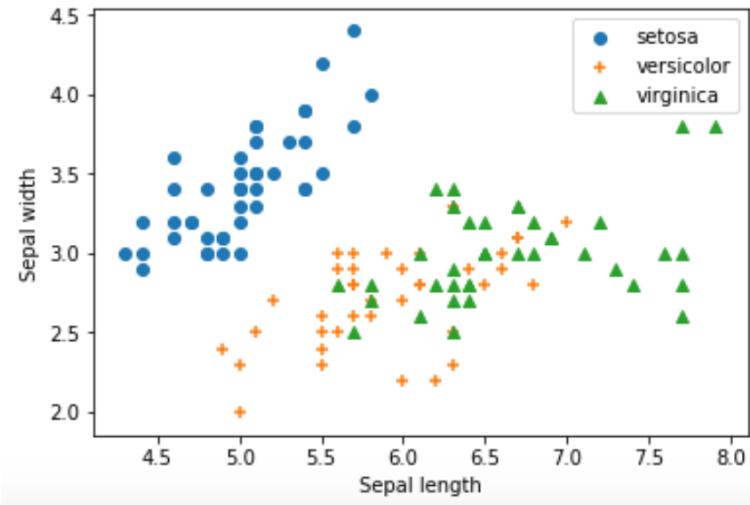
```
X_train, X_test, y_train, y_test = train_test_split(X2, y, test_size=0.2, random_state=7)
```

- 훈련 데이터로 80%를 사용한다는 의미이다. train\_test\_split 랜덤하게 데이터를 섞은 후에 훈련 데이터와 테스트 데이터를 나누는데, random\_state 값은 아무 값이나 써도 되는데, train\_test\_split 랜덤하게 데이터를 섞는 순서를 다음번 프로그램 실행시에도 같게 유지하려면 같은 random\_state 값을 써주면 된다(위에서 7을 사용했음).
- 훈련 데이터의 분포를 그래프로 그려보겠다. 여기서 가로와 세로축은 두 개의 속성 값을 나타내며, 각 샘플의 모양이 세가지 봇꽃의 종류를 구분한다.

```
%matplotlib inline  
import matplotlib.pyplot as plt
```

```
markers = ['o', '+', '^']  
for i in range(3):
```

```
xs = X_train[:, 0][y_train == i]
ys = X_train[:, 1][y_train == i]
plt.scatter(xs, ys, marker=markers[i])
plt.legend(iris.target_names)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
```



- 위 그림에서는 주어진 데이터의 속성 4개를 다 사용하지 않고 앞의 두 개의 속성에 대해서 타겟 값들을 표시해보았다(x축과 y 축). 학습에 사용할 속성으로 2개만 사용한 이유는 2차원 평면에서 보기 편하도록 임의로 2개의 속성을 택한 것 뿐이다. 실제로는 속성을 2개만 사용할 이유는 없다.
- 위 그림을 보면 2개의 속성만 사용해도 Setosa 타입(그림에서 원형으로 표시)은 다른 두 가지와 쉽게 구분할 수 있다는 것을 알 수 있다.
- 직선을 잘 선택하면 두 영역을 쉽게 구분할 수 있으며 이를 달리 표현하면 선형 모델로도 잘 나누어질 것으로 보인다.
- 여기서 타겟 변수를 3개로 나누어 진 것을, 세토사Setosa인 것(타겟=0)과

세토사가 아닌 것(타겟=1, 2) 두 가지를 구분하는 이진 분류 문제로 바꾸어 보겠다.

- 즉, Setosat인지 아닌지만 구분하는 간단한 문제로 바꾼다. 이를 위해 타겟 값이 2인 것을 모두 1로 바꾸어 사용하겠다. 즉, 1이나 2를 하나의 클래스로 분류(세토사가 아닌 것)하겠다는 의미이다.
- 아래의 코드에서는 타겟 값이 2인 것을 1로 바꾸는 작업을 하였다 (세토사가 아닌 것을 모두 1로 표시함).

```
y2 = y.copy()          # y의 복사본을 만든다  
y2[(y2==2)] = 1 # y중에 2의 값을 모두 1로 바꾼다  
y2
```

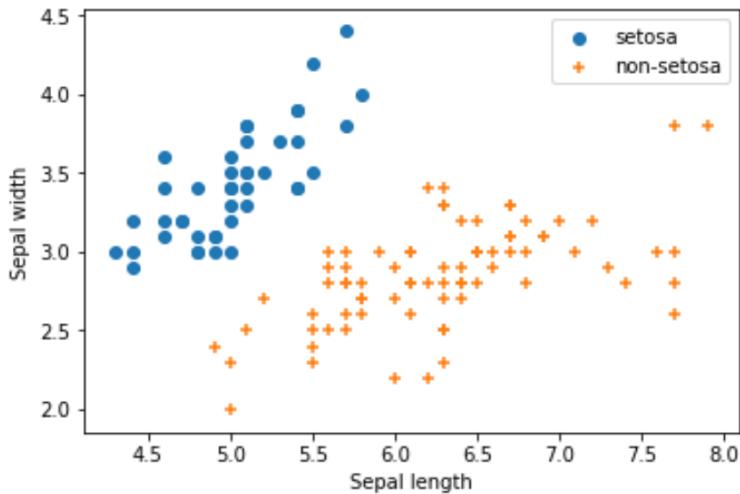
- $y$  대신  $y_2$ 를 사용하여 훈련 데이터를 다시 만든다.

```
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=7)
```

- 앞에서 소개한 그래프를 그리는 프로그램을 다시 실행하면 다음과 같은 그래프를 얻는다.

```
markers = ['o', '+', '^']
```

```
for i in range(3):
    xs = X_train[:, 0][y_train == i]
    ys = X_train[:, 1][y_train == i]
    plt.scatter(xs, ys, marker=markers[i])
binary_names = ['setosa', 'non-setosa']
plt.legend(binary_names)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
```



- 위 그래프를 보면, 세토사와 세토사가 아닌 것을 잘 나눌 수 있을 것으로 예상된다.
- 이제 학습을 하기 위해서 최적화 알고리즘을 선택해야 하는데 가장 널리 사용되는 경사하강법 (Gradient Descent, GD)을 사용하겠다.

```
from sklearn.linear_model import SGDClassifier
```

```
clf = SGDClassifier()  
clf.fit(X_train, y_train)
```

## 분류의 손실함수

- 앞에서 회귀 모델에서 가장 널리 사용되는 손실함수로 MSE를 소개했다. 그런데 분류 작업에서는 손실함수로 MSE를 사용할 수 없다. 왜냐하면 분류에서는 맞거나 틀리는 사실만 측정할 수 있으며 오차와 같은 어떤 수치를 맞추는 것이 아니기 때문이다.
- 분류 모델에서는 얼마나 맞게 분류했지를 계산하는 정확도(accuracy)를 손실함수로 사용할 수 있다.
- 예를 들어 100명에 대해 남녀 분류를 시도하였으나 96명을 맞추고 4명을 틀렸다면 정확도는 0.960이고 오류율은 0.04가 된다. 그런데 분류할 데이터 카테고리의 분포가 균일하지 않으면 정확도나 오류율을 손실함수로 사용하는데 문제가 있다.

- 예를 들어 남자가 50명, 여자가 50명이 있는 그룹에서 남자는 1명을 잘 못 분류하고 여자는 3명을 잘 못 분류했다고 하면 정확도는 0.96이다.
- 그런데 만일 남자가 95명, 여자가 5명이 있는 그룹에서 동일하게 남자는 1명을 잘 못 분류하고 여자는 3명을 잘 못 분류했다고 하면, 정확도는 여전히 0.96이다.
- 그러나 여성 5명 중에 3명을 잘 못 분류한 것은 뭔가 모델의 성능이 매우 나쁜 것인데 이러한 점이 손실함수에 반영되지 않았다.
- 정확도를 손실함수로 사용할 때의 이러한 문제점을 해결하기 위해서 도입한 개념이 크로스 엔트로피(cross entropy)이다. 크로스 엔트로피는 다음과 같이 정의한다.

$$CE = \sum_i p_i \log\left(\frac{1}{p_i}\right)$$

- 위에서  $p$ 는 어떤 사건이 일어날 실제 확률이고,  $\hat{p}$ 는 예측한 확률이다.  
앞의 남녀 구분 예에 적용하면 다음과 같다.

1) 남녀가 50명씩 같은 경우

$$CE = -0.5 \times \log\left(\frac{49}{50}\right) - 0.5 \times \log\left(\frac{47}{50}\right) = 0.02687$$

2) 남자가 95명 여자가 5명인 경우

$$CE = -0.95 \times \log\left(\frac{94}{95}\right) - 0.05 \times \log\left(\frac{2}{5}\right) = 0.17609$$

- 다른 케이스로, 남자가 95명, 여자가 5명이 있는 그룹에서 남자는 4명을 잘못 분류하고 여자는 모두 정확히 맞추었다고 하자. 정확도는 여전히 0.96이지만 크로스 엔트로피는 다음과 같은 값을 갖는다.

$$CE = -0.95 \times \log\left(\frac{91}{95}\right) - 0.05 \times \log\left(\frac{5}{5}\right) = 0.01774$$

- 위의 결과를 보면 정확도는 두 경우 모두 0.96으로 같은 값이지만, 크로스 엔트로피는 두 번째 경우는 첫 번째 보다 큰 값을 갖는다.
- 이상적이 분류 모델 즉, 100% 정확히 분류를 하면 로그 값이 10이 되어 크로스 엔트로피는 0이 된다.

- 이렇게 데이터의 분포가 비대칭인 상황은 질병을 진단하는 경우에 자주 발생한다.
- 예를 들어 1만명 중에 정상인이 9990명이고 암환자가 10명이라고 할 때, 모든 사람을 정상이라고 진단해도 평균 0.999의 정확도로 예측할 수 있다.
- 그러나 소수인 10명의 환자는 하나도 예측하지 못하게 된다. 이러한 상황에서는 크로스 엔트로피를 손실함수로 사용하여 모델을 학습시켜야 분포가 소수인 부분에 대해서도 성능을 개선하게 된다.
- 여기서는 남녀 구분 등 두 가지 카테고리를 분류하는 예를 들었지만 크로스 엔트로피는 구분할 대상이 둘 이상인 임의의 수, 예를 들어 1000가지 물체를 구분하는 경우 등에도 동일한 개념으로 사용된다.

## 결정경계

- 선형 분류는 아래의 부등식으로 동작한다. 아래 1차 선형 판별식에서 x는 꽃받침의 길이, y는 꽃받침의 폭을 나타낸다.

$$ax + by + c > 0$$

$$y > (-a/b)x - c/b$$

- 이를 그리기 위해서 다음과 같이 계수를 출력해 보았다.

```
a = clf.coef_[0,0]
```

```
b = clf.coef_[0,1]
```

```
c = clf.intercept_[0]
```

```
print(a, b, c)
## 103.189493433 -172.607879925 -35.2970136277

import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt

markers = ['o', '+', '^']
for i in range(3):
    xs = X_train[:, 0][y_train == i]
    ys = X_train[:, 1][y_train == i]
    plt.scatter(xs, ys, marker=markers[i])

binary_names = ['setosa', 'non-setosa']
```

```
plt.legend(binary_names)
```

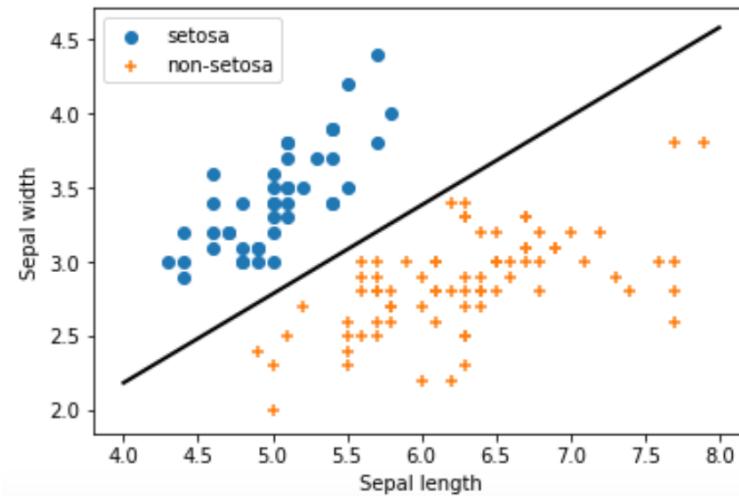
```
plt.xlabel("Sepal length")
```

```
plt.ylabel("Sepal width")
```

```
XX = np.linspace(4, 8, 200)
```

```
# 결정 경계선
```

```
plt.plot(XX, (-a/b * XX + -c/b), "k-", linewidth=2)
```



- 이제 임의의 새로운 샘플 데이터에 대해 어떻게 분류하는지를 확인해 보겠다.
- 꽃 받침의 길이와 폭이 각각, [4.5, 3.5], [6.5, 2.5]인 두 가지 경우의 예상되는 봉꽃의 종류를 예측하면 아래와 같이 각각 세토사 및 세토사가 아닌 것으로 분류한 것을 알 수 있다.

```
print(clf.predict([[4.5, 3.5]])) # 0
```

```
# 세토사로 예측했다
```

```
print(clf.predict([[6.5, 2.5]])) # 1
```

```
# 세토사가 아닌 것으로 예측했다
```

- 꽃받침의 길이와 폭의 속성 그래프 상에서 아래 두 점의 좌표의 위치를

보면 대략 어떤 클래스에 해당할지 눈으로 봐도 알 수는 있다.

- 이제 테스트 데이터에 대해서 예측을 모두 시도하고 이 데이터를 기반으로 정확도를 계산해 보겠다.
- 예측을 수행하려면 모델 (아래에서는 clf)이 제공하는 함수 predict()를 사용하면 예측을 수행한다. 예측한 결과는 세토사로 예측하면 0, 아닌 것으로 예측하면 1일 들어가는데 이 값들이 y\_test\_pred 어레이에 저장된다.
- 이제 이 예측값과 실제 테스트 데이터의 타겟 값이 저장된 y\_test 값을 비교하여 정확도를 구하는 코드는 아래와 같다.

```
y_test_pred = clf.predict(X_test)  
print(metrics.accuracy_score(y_test, y_test_pred))
```

```
## 출력
```

```
0.9
```

- 위에서 accuracy\_score() 함수의 출력 값이 0.9인데 이는 30개 테스트 데이터 중에 27개를 정확히 분류하고 세 개를 틀리게 분류한 결과이다.  
 $27/30 = 0.9.$
- 붓꽃 분류에서 앞의 두가의 속성만 사용하여 분류를 수행해보겠다 (4개를 다 사용하지 않고)

```
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.datasets import load_iris
```

```
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import cross_val_score, KFold

from sklearn.svm import SVC

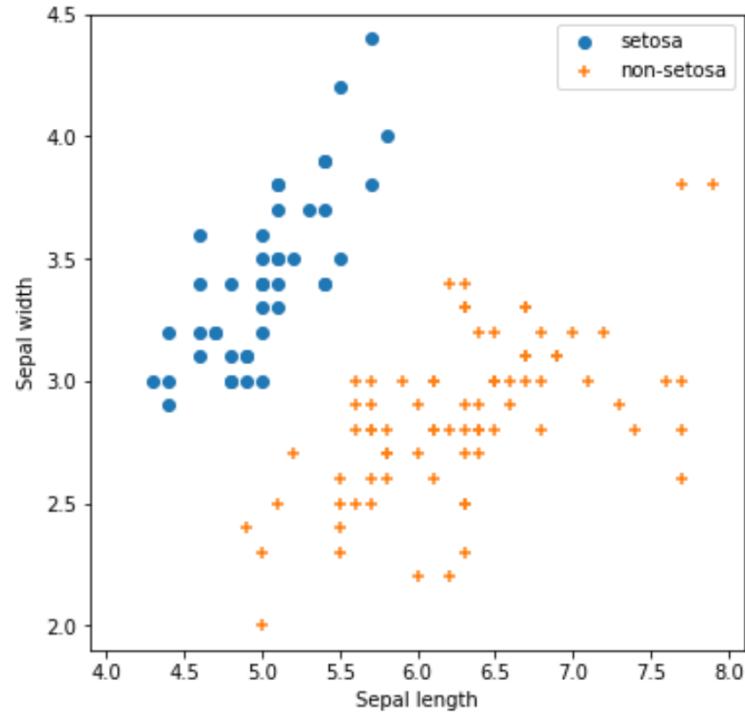
%matplotlib inline

iris = load_iris()
X, y = iris.data, iris.target

X2 = X[:, :2]
y2 = y.copy()      # y의 복사본을 만든다
y2[(y2==2)] = 1   # y중에 2의 값을 모두 1로 바꾼다 (세토사가 아닌 것 = 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=7)

plt.figure(figsize=(6,6))
plt.xlim(3.9,8.1)
plt.ylim(1.9,4.5)
markers = ['o', '+', '^']
for i in range(3):
    xs = X_train[:, 0][y_train == i]
    ys = X_train[:, 1][y_train == i]
    plt.scatter(xs, ys, marker=markers[i])
binary_names = ['setosa', 'non-setosa']
plt.legend(binary_names)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
```



- 먼저 선형 분류를 그리면 다음과 같다.

```
from sklearn.linear_model import SGDClassifier  
clf = SGDClassifier()  
clf.fit(X_train, y_train)
```

```
# threshold = ax + by + c > 0  
# y > (-a/b)x - c/b
```

```
a = clf.coef_[0,0]  
b = clf.coef_[0,1]  
c = clf.intercept_[0]
```

```
plt.figure(figsize=(6,6))  
plt.xlim(3.9,8.1)  
plt.ylim(1.9,4.5)
```

```
markers = ['o', '+', '^']

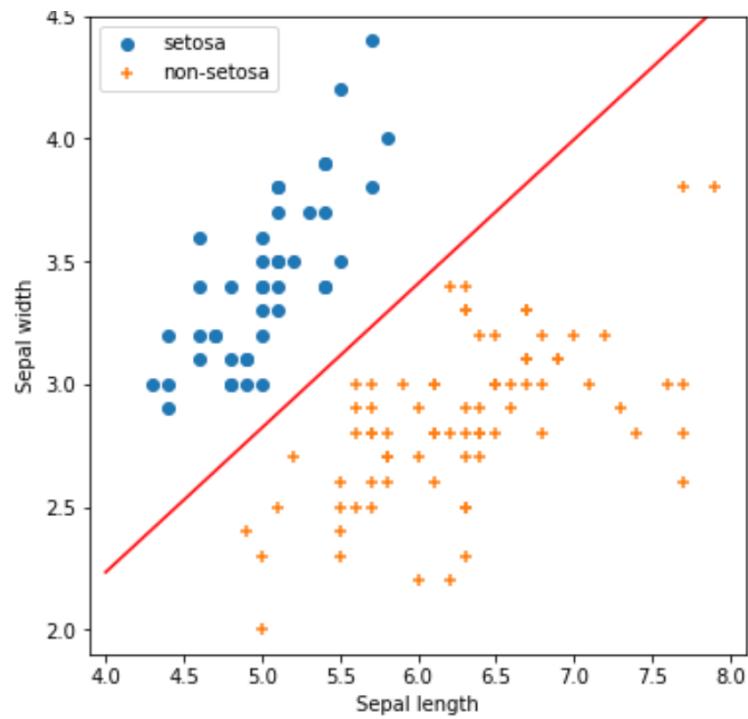
for i in range(3):
    xs = X_train[:, 0][y_train == i]
    ys = X_train[:, 1][y_train == i]
    plt.scatter(xs, ys, marker=markers[i])

# plt.legend(iris.target_names)

binary_names = ['setosa', 'non-setosa']
plt.legend(binary_names)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
```

```
XX = np.linspace(4, 8, 200)

# 결정 경계선
plt.plot(XX, (-a/b * XX + -c/b), "r-")
```



## 교차검증

- 아래는 세토사인지 아닌지만 구분하는 이진 분류 예에서 만든 모델(clf)을 사용하여 K=5인 교차 검증을 수행하는 코드이다.

```
from sklearn.cross_validation import cross_val_score, KFold  
cv = KFold(X2.shape[0], 5, shuffle=True)  
print(cross_val_score(clf, X2, y2, cv=cv))  
  
## 결과  
[ 0.96666667  0.96666667  1.          1.          0.86666667]
```

## 4가지 속성 사용

- 앞의 예에서 X, y 가 아니라 X2, y2를 사용한 이유는 이진 분류용으로

만든 데이터를 사용하기 위해서이다. 만일 4개의 속성을 모두 사용하고 세 가지 붓꽃을 분류하는 모델을 교차검증하려면 아래와 같이 X, y를 사용하면 된다.

- 4개의 속성을 모두 사용하여 세가지 붓꽃을 분류하는 프로그램은 아래와 같다.

```
from sklearn import datasets  
from sklearn.cross_validation import train_test_split  
from sklearn.linear_model import SGDClassifier  
from sklearn import metrics  
  
iris = datasets.load_iris()  
X, y = iris.data, iris.target  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
```

```
clf_all = SGDClassifier()
clf_all.fit(X_train, y_train)
y_test_pred = clf_all.predict(X_test)
print(metrics.accuracy_score(y_test, y_test_pred))
```

## 출력

0.8

```
cv = KFold(X.shape[0], 5, shuffle=True)
print(cross_val_score(clf_all, X, y, cv=cv))
```

## 결과

[ 0.73333333 0.66666667 0.93333333 0.76666667 0.93333333]

# 서포트벡터머신 (SVM)

- 서포트 벡터 머신(Support Vector Machine, SVM)은 이름이 좀 생소하지만 가장 널리 사용되는 예측 모델이다. SVM은 선형 모델의 성능을 개선하였다.
- SVM의 기본적인 아이디어는 분류시에 경계면을 가능한 일반화 하는 것이다. 이진 분류를 예를 들면, 각각의 카테고리에 속한 샘플들을 단순히 나누기만 하는 것이 아니라 가능한 거리를 멀리 나눌 수 있는 경계면을 찾는 작업을 한다.

## SVC 예시

```
# SVM Classifier model  
svm_clf = SVC(kernel="linear", C=float("inf"))
```

```
svm_clf.fit(X_train, y_train)
```

```
plt.figure(figsize=(6,6))
```

```
plt.xlim(3.9,8.1)
```

```
plt.ylim(1.9,4.5)
```

```
w = svm_clf.coef_[0]
```

```
v = svm_clf.intercept_[0]
```

```
decision_boundary = -w[0]/w[1] * XX - v/w[1]
```

```
margin = 1/w[1]
```

```
gutter_up = decision_boundary + margin
```

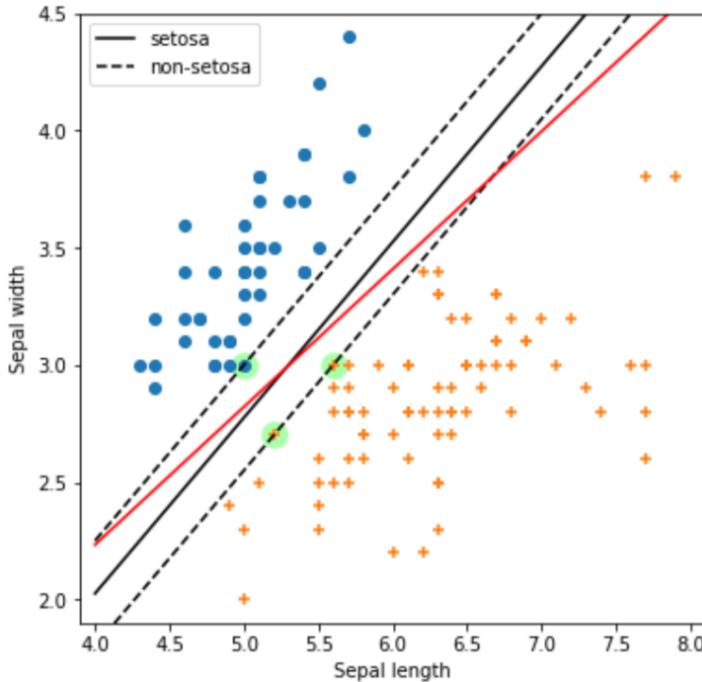
```
gutter_down = decision_boundary - margin
```

```
svs = svm_clf.support_vectors_
plt.scatter(svs[:, 0], svs[:, 1], s=180, facecolors='#AAFFAA')
plt.plot(XX, decision_boundary, "k-")
plt.plot(XX, gutter_up, "k--")
plt.plot(XX, gutter_down, "k--")

markers = ['o', '+', '^']
for i in range(3):
    xs = X_train[:, 0][y_train == i]
    ys = X_train[:, 1][y_train == i]
    plt.scatter(xs, ys, marker=markers[i])
binary_names = ['setosa', 'non-setosa']
plt.legend(binary_names)
```

```
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")

XX = np.linspace(4, 8, 30)
# 선형분류 결정 경계선
plt.plot(XX, (-a/b * XX + -c/b), "r-")
```



- 위의 그림에서 원형과 십자가를 구분하는데는 선형 분류 직선이 모두 분류를 수행하는데 문제가 없어 보인다. 그러나 이러한 경계면은 새로운 샘플 데이터에 대해서는 잘 동작하지 않을 것이다.

- 검정 실선의 SVM이 만든 경계선을 사용하는 것이 보다 일반적으로 보인다. 이를 개념적으로 설명하면 두께가 없는 선으로 경계를 만드는 것이 아니라 우측 그림의 점선으로 이루어진 두께가 있는 굽은 경계면을 만들고 그 두꺼운 경계면의 중앙을 지나는 선을 택하는 방법이 타당해보이다.
- 결정경계 주변에 있는 샘플들과 결정경계와의 거리 (margin)를 최대화 하는 방향으로 설정한다. 이를 마진을 최대화한다고 한다. (maximizing the margin).
- 이러한 방식으로, 즉, 가능한 샘플들을 두꺼운 선으로 나누는 것으로 제한하면 더 일반적인, 안정적인 경계선을 얻을 수 있으며 이것이 SVM 알고리즘의 기본 개념이다.

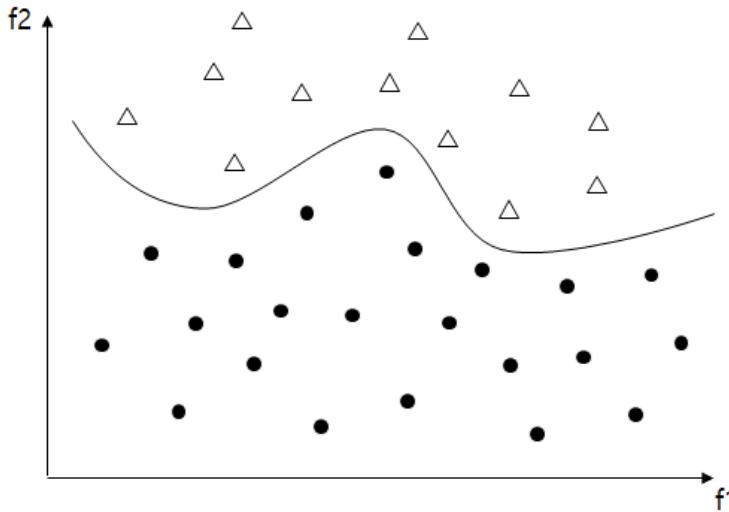
- SVM은 이를 통해 선형모델보다 일반화된 모델을 만들고 과대적합을 피하게 된다.
- 서포트 벡터, 즉 두꺼운 경계 선과 만나는 점이 세 개 있다. 이의 좌표를 출력하면 아래와 같다.

SVS

```
##  
array([[ 5. ,  3. ],  
       [ 5.2,  2.7],  
       [ 5.6,  3. ]])
```

## 커널 방식

- SVM은 커널 방식을 제공하는데 주어진 속성을 그대로 사용하지 않고 이의 2승, 3승, 4승 등 고차원의 속성을 내부적으로 만들어서 사용하는 방식이다.
- 예를 들면 아래 그림과 같은 샘플 분포에서 선형적인 (직선)의 경계를 사용하는 것이 아니라, 점선으로 된 곡선의 경계를 찾을 수있다면 분류를 더 잘 할 것이다.



커널 기법을 이용하여 비선형적인 결정 경계(하이퍼 평면)를 만들 수 있다

- 고차원의 새로운 공간으로 매핑하기 위해서 (1차 함수가 아니라) 2차 함수 이상의 고차항 들을 사용하는데 이를 커널기법(kerner trick)이라고 한다.
- SVM에 커널 트릭 기법을 도입함으로써 비선형 함수를 도입하고 이것이

선형 모델을 개선하는 주요 요소이다.

- 비선형 요소는 속성의 2차항, 3차항 등 다행 차수를 도입하는 것으로서 다행 함수를 사용함으로써 복잡한 경계면을 모델링할 수 있다. 물론 이렇게 비선형 요소를 추가로 고려함으로써 계산량은 매우 늘어난다.
- 커널 기법중에 방사 기저 함수(Radial Basis Function, RBF) 방식이 가장 효율적으로 알려져 있으며 RBF가 현재 사이킥런 패키지의 SVM 모델의 기본 커널로 사용되고 있다.
- SVM 방식은 성능이 우수하여 랜덤 포레스트와 같이 가장 널리 사용되고 있다. 주의 할 것은 SVM은 선형 모델과 마찬가지로 속성에 계수를 곱하고 덧셈을 하는 연산에 기반하므로 여러 속성을 함께 사용하려면 반드시 스케일링을 해야 한다.

- 키와 몸무게는 차원이 다른, 성격이 다른 속성인데 여기에 곱셈과 덧셈을 하여 경계를 만들 때에는 스케일링을 하여 절대 값의 차이가 분류나 예측에 영향을 주지 않도록 해야 한다.

## 유방암 예측 데이터

- 위스콘신 대학에서 유방암 진단에 사용된 데이터를 공개했다. 데이터 구조를 보면 아래와 같다. 유방암 진단을 위해 다양한 속성들로 진단을 내렸고 타겟은 M과 B로 이진 클래스를 가지고 있으며 M은 malignant(악성), B는 benign(양성)으로 구분되어 있다.
- 데이터를 아래 사이트에 다운로드 받을 수 있다.

<https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/kernels>

```
import pandas as pd  
data = pd.read_csv('data/breast_cancer.csv')  
print(data.shape)  
data.columns  
  
## 출력  
(569, 33)  
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',  
'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave  
points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se',  
'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave  
points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst',  
'texture_worst', 'perimeter_worst', 'area_worst', 'compactness_worst',  
'concavity_worst', 'concave points_worst', 'symmetry_worst',
```

```
'fractal_dimension_worst', 'Unnamed: 32'], dtype='object')
```

```
data.head(5)
```

	<b>id</b>	<b>diagnosis</b>	<b>radius_mean</b>	<b>texture_mean</b>	<b>perimeter_mean</b>	<b>area_mean</b>	<b>smoothness_mean</b>	<b>compactness_mean</b>
<b>0</b>	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
<b>1</b>	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
<b>2</b>	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
<b>3</b>	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390
<b>4</b>	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280

- 속성을 설명하면 다음과 같다. id는 환자 고유번호이고 diagnosis는 암 판정 결과를 나타내는데 M은 악성(malignant) B는 양성(benign)을 나타낸다. 양성 샘플은 357개 악성 샘플은 212개 이다.

- 30개의 속성을 제공하는데, radius\_mean: 평균 반경, texture\_mean: 평균 질감 등을 포함한다. 내용을 보면 10개 속성에 대해 각각 평균, 표준 오차, 가장 큰 값 등 각각 3개의 값을 제공하여 총 30개의 데이터를 제공한다.

## 유방암 예측

- 커널 기법을 사용하는 예를 유방암 예측에 적용하면 아래와 같다.

```
from sklearn.svm import SVC  
from sklearn.model_selection import train_test_split  
import pandas as pd  
import numpy as np  
  
y = pd.get_dummies(data['diagnosis'])
```

```
x = cancerData.drop(['Unnamed: 32','id','diagnosis'], axis = 1 )  
drop_list = ['perimeter_mean','radius_mean','compactness_mean','concave  
points_mean','radius_se','perimeter_se','radius_worst','perimeter_worst','compactne  
ss_worst','concave points_worst','compactness_se','concave  
points_se','texture_worst','area_worst']
```

```
x = data.drop(['Unnamed: 32','id','diagnosis'], axis = 1 )  
drop_list = ['perimeter_mean','radius_mean','compactness_mean',  
            'concave points_mean','radius_se','perimeter_se',  
            'radius_worst','perimeter_worst','compactne ss_worst',  
            'concave points_worst','compactness_se',  
            'concave points_se','texture_worst','area_worst']  
  
x = x.drop(drop_list, axis = 1, errors='ignore')
```

# 커널 호출

```
from sklearn.multiclass import OneVsRestClassifier  
classifier = OneVsRestClassifier(SVC(kernel='rbf', C=10,  
gamma=0.1,probability=True))  
classifier = classifier.fit(X_train, y_train)  
classifier.score(X_test, y_test)  
>> 0.97777777
```

```
clf = SGDClassifier(max_iter=1000)  
clf.fit(X_train, y_train)  
clf.score(X_test, y_test)  
>> 0.8444444444
```

## 분류의 성능지표

- 분류에서는 손실함수로 크로스엔트로피를 주로 사용하고 성능 지표로는 정확도(accuracy), 정밀도(precision), 재현률(recall), F1-지수 등이 사용된다.
- 손실함수와 성능지표에 대한 정의와 대표적인 값을 아래 표에 정리하였다.

	손실함수	성능평가지표
정의	손실함수를 줄이는 방향으로 모델이 학습을 함	성능을 높이는 것이 머신러닝을 사용하는 목적임
회귀 모델의 대표적인 값	MSE (오차 자승의 평균치)	$R^2$
분류 모델의 대표적인 값	크로스 엔트로피	정확도, 정밀도, 재현률, F1점수

## 7.3. kNN

- kNN(k-nearest neighbor) 알고리즘은 주어진 샘플의 특성 공간상에서 거리가 가장 가까운 이웃(neighbor)을 k개 선택하고 이들 레이블의 평균치로 이 샘플이 속할 카테고리를 예측하는 방식이다.
- kNN은 직관적으로 이해하기 쉬운 알고리즘으로서 추천 시스템에서 많이 사용되는데 적절한 추천을 하기 위해서 추천을 요청한 사람의 성향을 특성들로 파악하고 그 사람과 가장 성향이 유사한 k명의 사람들이 좋아하는 품목을 추천하는 방식을 사용한다.
- kNN알고리즘을 협업 필터링(collaborative filtering)이라고도 부른다.

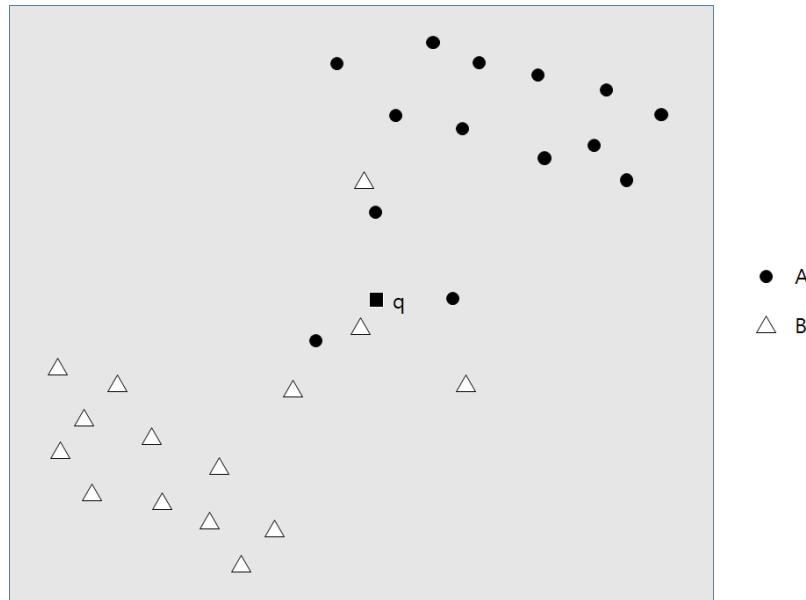
## 개념

- 예를 들어 서점에서 어떤 사람(A)에게 도서를 추천한다고 하자. 서점에서는 그 사람과 독서 취향이 비슷한 사람들 그룹을 먼저 찾는다. 서점에서는 평소 고객들이 어떤 책들을 즐겨 보는지, 그리고 책에 대한 평가는 어땠었는지를 보고 취향이 비슷한 사람들을 구분해 낼 수 있다.
- 이제 A와 취향이 비슷한 k명의 사람들 사이에서 평이 가장 좋은 책 중에 아직 A가 보지 않은 책을 추천하면 된다.
- 여기서 협업이라는 말을 쓰는 이유는 여러 사람(샘플)의 협력을 이용하기 때문이다. 취향이 비슷한 '사람'을 기반으로 하는 협업 필터링을 사용자-기반 협업 필터링(user-based collaborative filtering)이라고 한다.

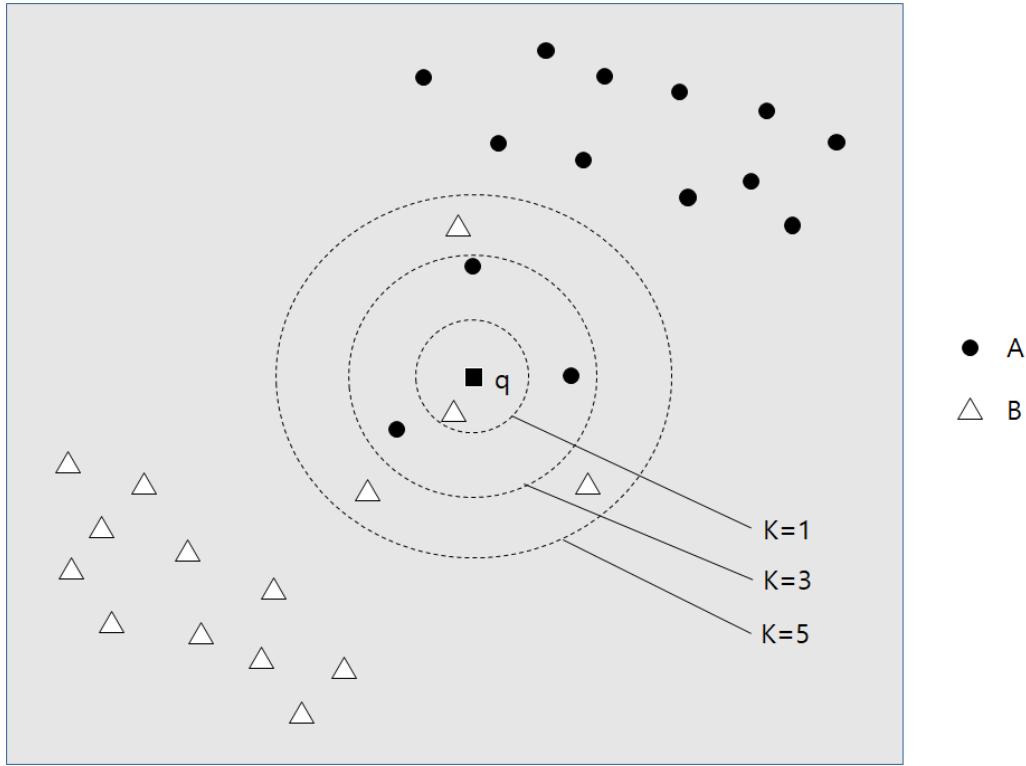
- 반면에 상품을 기반으로 하는 방법이 있다. 예를 들어 a라는 상품을 좋아하는 사람들은 대부분 b나 c 상품도 같이 좋아한다고 하자. 그러면 상품 입장에서 유사한 상품 그룹을 만들 수 있을 것이다.
- 무협지를 좋아하는 사람은 다른 무협지를 향상 좋아하고 연애 소설도 좋아한다는 식으로 책이라는 상품을 기준으로 비슷한 상품 그룹을 만들고, 어떤 사람이 새로운 상품 추천을 원하면 성격이 비슷한 상품 중에 그 사람이 아직 구매하지 않은 것을 추천하면 된다.
- 이러한 방식을 아이템-기반 협업 필터링(item-based collaborative filtering)이라고 한다.

## kNN 동작

- 예를 들어 아래 그림에서  $q$  위치의 샘플 데이터는 그룹 A와 B 중 어느 그룹이라고 판단해야 할까?



- 그러나 주변의 샘플 하나만 고려하는 것 보다는 몇 개 샘플의 평균을 사용하는 것이 타당할 것이다.
- 아래 그림에 여러 가지  $k$  값에 대한 경우를 나타냈다. 만일  $k=4$ 로 설정하면 이 때의 점선 원 내에는 그룹 B인 점(즉, 원)이 3개이고, 그룹 A인 점(세모)가 한개이므로  $q$ 는 그룹 B로 판정될 것이다.
- $k=7$ 로 선택하면 이 원 안에는 그룹A가 4개 그룹B가 3개 이므로 이때는 다시  $q$ 가 그룹 A로 분류되어야 한다. 이렇게  $k$  값을 어떻게 선택하는 가에 따라  $q$ 가 어느 그룹에 속하는지가 달라질 수 있다.



kNN에서  $k$  값을 달리 정했을 때( $k=1, 3, 5$ ) 샘플  $q$ 가 그룹 A와 B중 어디에 속하는지가  
달라지는 예

- $k$  값을 너무 작게 잡으면 주변 데이터에 너무 예민하게 반응하고  $k$  값을 너무 크게 잡으면 주변에 너무 많은 데이터의 평균치를 사용하므로 분류가 무뎌진다.
- 극단적으로  $k=N$  (전체 샘플 수)로 잡으면 항상 전체 데이터의 평균치 값을 예측하게 된다.
- 영화 추천에서  $k=N$ 으로 한다면 이는 평균적으로 가장 많은 사람들이 본 영화 즉, 종합 베스트셀러를 추천하는 것과 같다. 나에게 맞는 추천이라고 할 수 없다.
- $k$ 값을 작게 잡으면 노이즈에 민감하나 정확도는 올라가고  $k$ 를 크게 잡을수록 노이즈에 강하나 정밀한 예측이 어렵다.

- 여기서 노이즈란 좋은 모델을 만드는데 도움이 되지 않고 혼란스러운 정보를 제공하는 샘플을 말한다. 모델을 만드는데 도움이 되는 정보를 시그널이라고 부른다.
- kNN 알고리즘의 장점은 알고리즘 개념이 명확하고 예측 성능도 좋다는 것이다. 모델링에 필요한 하이퍼 파라미터도  $k$ 값 하나만 고려하면 된다. 모델 훈련 시간도 거의 필요 없고 특성 변수들만 잘 선정하면 된다.
- kNN의 단점은 훈련시간이 거의 없는 것에 비해 분류를 처리하는 시간, 즉 알고리즘을 수행하는 시간이 길다는 것이다. 왜냐하면 확보한 샘플들을 모두 비교해서 어떤 그룹에 가까운지를 새롭게 계산해야 하기 때문이다.
- 더욱이 새로운 샘플이 계속 추가될 때마다 가까운 이웃이 달라진다. 모델

자체는 “가까운 이웃을 찾는다”는 간단한 원리로 동작하지만, 누구와 가까운지를 미리 계산해둘 수 없고 비교할 샘플이 나타나야만 그때부터 계산을 한다. 따라서 분류에 시간도 걸리고 계산량이 많아진다.

- 이렇게 나중에 계산량이 많은 종류의 알고리즘을 게으른(lazy) 알고리즘이라고 한다.
- 지금까지 설명한 kNN 알고리즘에서는 단지 k개 이웃의 샘플들을 동일한 조건으로 고려하였다. 샘플간의 거리는 고려하지 않았는데 가까이 있는 항목에 대해서는 가중치를 크게 하는 방법도 있다.

## 붓꽃 예제

- 붓꽃 분류를 kNN으로 수행해보겠다. 먼저 필요한 패키지를 호출하고 붓꽃 데이터를 로드한다.

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from

from sklearn.datasets import load_iris
iris = load_iris()
X, y = iris.data, iris.target
```

- 네 가지 특성을 모두 사용하고 세 가지 붓꽃을 분류하는 데, 선형분류와 kNN의 성능을 비교해 보겠다.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
clf = SGDClassifier()
```

```
knn = KNeighborsClassifier(n_neighbors=3)
```

```
knn.fit(X_train, y_train)
```

```
print("kNN score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
clf.fit(X_train, y_train)
```

```
print("Linear Reg score: {:.2f}".format(clf.score(X_test, y_test)))
```

```
##  
kNN score: 0.91  
Linear Reg score: 0.60
```

- k 값을 1부터 9까지 바꾸면서 kNN의 성능을 비교하면 아래와 같다.

```
for i in range(1,10):  
    knn = KNeighborsClassifier(n_neighbors=i)  
    knn.fit(X_train,y_train)  
    print("K가", i, "일때 정확도: {:.4f}".format(knn.score(X_test, y_test)))
```

```
##  
K가 1 일때 정확도: 0.91
```

K가 2 일때 정확도: 0.91

K가 3 일때 정확도: 0.91

K가 4 일때 정확도: 0.96

K가 5 일때 정확도: 0.93

K가 6 일때 정확도: 0.93

K가 7 일때 정확도: 0.91

K가 8 일때 정확도: 0.91

K가 9 일때 정확도: 0.91

- kNN 모델에서는  $k$  값이 작을수록 과대적합될 가능성이 커진다.  $k=1$ 로 설정하면 새로운 샘플과 바로 인접한 특성을 갖는 데이터의 레이블을 분류 결과로 예측한다. 이는 매우 민감하여 잘 맞을 수도 있지만, 노이즈를 피할 수 없다.

- 따라서 예측이 매우 정확할 가능성은 높지만 동시에 노이즈로 인해 엉뚱한 값을 예측할 수도 있다.
- 이러한 현상은 모델이 너무 정교해서 과대적합한 결과이며 예측값이 넓게 퍼져서 나타나므로 분산에 의한 오류가 크다고 한다.
- 반대로  $k$ 를 크게 정할수록 평균치로 예측하는 성향이 많아지며 모델이 과소적합이 된다. 예를 들어 극단적으로  $k=N$  (모집단 수)로 정하면 항상 전체 평균치로 예측을 한다.
- 영화 추천에서 많은 사람이 재미있다고 하는 영화를 추천해주는 것과 같다.
- 모델이 성의 없이 평균을 따라가는 구조이며 이는 매우 간단한, 과소적합된 모델이다. 개선이 필요한 모델이다.

## 7.4. 결정 트리

- 앞에서 소개한 선형회귀 모델은 특성들을 대상으로 곱셈과 덧셈 등 연산을 하고 그 값을 기준으로 회귀나 분류를 예측했다.
- 결정 트리(decision tree)는 이와 달리 각 특성을 독립적으로 하나씩 검토하여 분류 작업을 수행한다. 마치 스무고개 하여 분류 예측을 하듯이 한 번에 하나의 특성을 따져보는 방법이다.
- 결정 트리는 주로 분류에 사용되지만 회귀에도 사용할 수 있다. 예를 들어 분류용 모델은 DecisionTreeClassifier가 있고 회귀분석 모델로는 DecisionTreeRegressor가 제공된다.

### 회귀와 분류 알고리즘

- 앞에서 분류와 회귀는 각각 목적이 다르다고 설명했다. 분류는 카테고리를 예측하는 것이고 회귀는 수치를 예측하는 것이다. 그리고 분류와 회귀를 위한 여러 알고리즘들이 소개되었다.
- 그러나 이 알고리즘들은 서로 다른 용도로도 사용될 수 있다. 회귀를 위해 만든 알고리즘을 분류에 사용할 수 있고, 분류를 위해 만든 알고리즘을 회귀에 사용할 수 있다.
- 예를 들어 내일 비올 확률이 70%라고 예측하는 것은 분류문제인가 회귀문제인가? 숫자를 예측하므로 회귀문제로 볼 수 있지만, 비올 확률일 0%, 10%, 20%, ..., 100% 등 총 11개의 클래스로 나누는 분류문제로 볼 수도 있다.

- 실제로 많은 경우에 회귀분석과 분류는 같은 알고리즘을 사용할 수 있다. 즉 선형회귀 알고리즘은 회귀 분석 뿐 아니라 분류에도 사용할 수 있다.
- kNN이나 결정트리, 랜덤 포레스트, SVM, 신경망 등도 회귀와 분류에 모두 사용할 수 있다는 것을 기억하기 바란다. 즉, 최초에 알고리즘을 만들때는 주로 회귀 또는 분류 중 하나의 목적에 사용하는 것을 목적으로 만들었지만 다른 목적으로 대부분 사용할 수 있다.

## 동작 원리

- 결정 트리에서 핵심이 되는 부분은 가장 효과적인 분류를 위해서 어떤 변수를 가지고 판별을 할지 결정하는 것이다.
- 이 판별은 트리를 내려가면서 계속되는데 매 단계마다 어떤 변수를 기준으로 분류를 하는 것이 가장 효과적인지를 찾아야 한다.
- 그룹을 효과적으로 “잘 나누는 것”의 기준은 그룹을 나눈 후에 생성되는 하위 그룹들에 가능하면 같은 종류의 아이템들이 모이는지이다.
- 한 그룹에 같은 종류의 아이템이 많이 모일수록 순수(pure)하다고 하는데, 만일 나누어진 하위 그룹이 100% 같은 항목들로만 구성되면, 순도(purity)가 100%라고 한다.

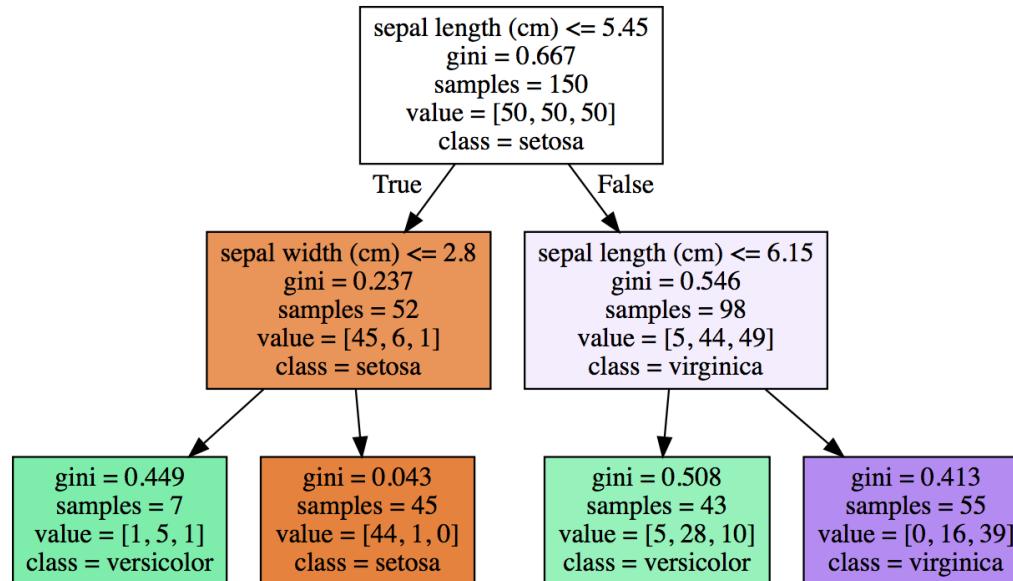
## 결정 트리 예제

- 붓꽃을 분류하는데 결정 트리를 사용해보겠다.
- 먼저 붓꽃 파일을 읽고 속성으로 앞의 두 열, 꽃받침의 길이와 폭 두 개의 속성을 사용하고 타겟변수로는 붓꽃의 종류를 사용한다.

```
from sklearn.datasets import load_iris  
iris = load_iris()  
X = iris.data[:, :2]  
y = iris.target
```

- 아래는 결정 트리 모델을 생성하고 훈련 데이터로 훈련시키는 코드이다. 트리의 깊이는 2로 제한하였다.

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth = 2)
clf.fit(X, y)
```



- 결정 트리 첫 번째 노드를 보면(이 노드를 루트 노드라고 부른다) 꽃받침의 길이가 5.45cm보다 짧은지(Treu) 아닌지(False)를 보고 구분을 한다. 나누기 전에는 총 150개의 샘플이 있었고, 세 개 클래스별로 각각 50개씩 샘플이 들어 있는 것을 나타낸다.
- value [ ] 부분에 이 결정 과정을 수행하기 전의 클래스별 항목의 개수를 나타낸다. 지니 계수는 0.667로 높은 편이다.
- 나누어지는 하위 트리의 좌 우 측에 같은 클래스들이 가능한 많이 모이도록, 즉, 지니 계수가 작아지도록 트리를 만들어 준다.
- 나누어진 첫 번째 단계의 좌측 트리의 value[ ]를 보면 총 52개 샘플중에 세토사가 다수인 45개인 것을 알 수 있다. 트리가 여기서 멈추어야 한다면

이 박스에 속한 샘플들의 대표는 세토사로 분류되는 것이 타당하다.

- 이번에는 꽃받침의 폭이 2.8cm보다 짧은지(True, 좌측 박스로 이동), 아닌지 (false, 우측 박스로 이동)를 판단한다. 지니 계수는 0.237로 작아졌다.
- 이제 베실로카와 세토사 두 개의 그룹으로 나누었으며 3개의 샘플을 제외하고 모두 제대로 분류가 되었다.
- 결정 트리 동작을 그림으로 나타내면 아래와 같다. 루트노드에서 나누는 작업은 꽃받침의 길이 (x 축)에서 5.45cm를 기준으로 작은 쪽에는 세토사가 주로 들어 있고, 우측에는 버지니카가 주로 들어 있다고 구분하였다(실선).
- 1차(자식) 노드중 좌측 노드에서는 꽃 받침의 폭이 2.8cm보다 작으면

베시컬러가, 2.8보다 크면 세토사인 것으로 분류했다(점선). 1차(자식) 노드 중 우측 노드에서는 꽃 받침의 길이가 6.15cm보다 작으면 베시컬러가, 6.15보다 크면 버니니카인 것으로 분류했다(점선).

- 이와 같이 나누어야 지니 불순도가 가장 작은 값을 갖는 조건이 된다. 즉, 분류가 잘 된 기준이었다.

```
%matplotlib inline  
import matplotlib.pyplot as plt  
import numpy as np  
  
plt.xlim(4, 8.5)  
plt.ylim(1.5, 4.5)
```

```
markers = ['o', '+', '^']

for i in range(3):
    xs = X[:, 0][y == i]
    ys = X[:, 1][y == i]
    plt.scatter(xs, ys, marker=markers[i])
```

```
plt.legend(iris.target_names)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
```

```
# 결정 트리 경계선: 실선은 루트 노드 점선은 자식 노드
xx = np.linspace(5.45, 5.45, 20)
yy = np.linspace(1.5, 4.5, 20)
```

```
plt.plot(xx, yy, '-k') # 검정색 실선
```

```
xx = np.linspace(4, 5.45, 20)
```

```
yy = np.linspace(2.8, 2.8, 20)
```

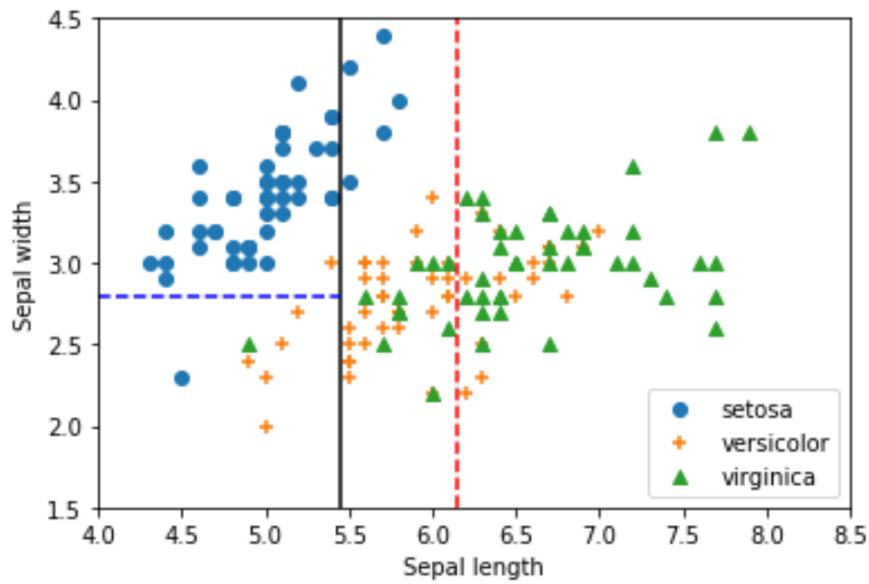
```
plt.plot(xx, yy, '--b') # 파란색 점선
```

```
xx = np.linspace(6.15, 6.15, 20)
```

```
yy = np.linspace(1.5, 4.5, 20)
```

```
plt.plot(xx, yy, '--r') # 빨간색 점선
```

```
## 출력
```



## 클래스 확률

- 학습된 결정 트리를 이용해서 주어진 임의의 붓꽃 샘플이 어느 클래스에 속할지 예측하는 방법은 아래와 같이 predict()를 사용하면 된다. 아래는 꽃받침의 길이와 폭이 (5.5, 4)이면 클래스 1, 즉, 베시컬러로 예측한다는 것을 보여주는 코드이다.

```
print(clf.predict([[5.5, 4]])) # [1]
```

- 위와 같이 주어진 샘플이 어느 하나의 클래스에 속하는지를 예측하는 것 외에, 이 샘플이 모든 클래스에 속할 확률이 각각 얼마인지를 계산하는 것도 가능하다.

- 이렇게 샘플이 각 클래스에 속할 확률을 알려주면 단순하게 하나의 클래스로 분류한 것보다 유용하게 활용할 수가 있다. 예를 들어 소프트 투표(soft voting)를 도입하면 보다 정확한 다중 분류를 수행할 수 있다. 각 클래스에 속할 확률을 구하려면 predict\_proba() 함수를 사용하면 된다.

```
print(clf.predict_proba([[5.5, 4]]))  
## 출력  
[[ 0.11627907  0.65116279  0.23255814]]
```

- 위 결과의 의미는 이 샘플이 세토사, 베시컬러, 버지니카로 분류된 확률이 각각 [0.11627907 0.65116279 0.23255814]인 것을 알려준다.

## 판별 기준

- 결정 트리는 나누어지는 그룹의 순도가 가장 높아지도록 그룹을 나누어야 한다. 그룹의 순도를 표현하는 데 지니(Gini) 계수 또는 엔트로피(entropy)가 주로 사용된다.

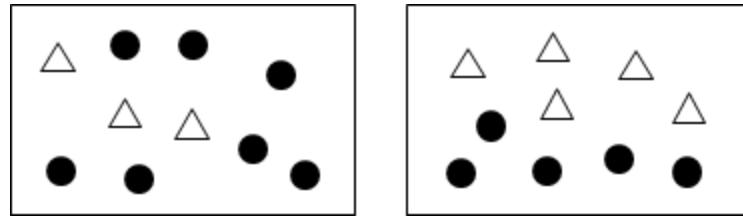
### 지니 계수

- Gini 계수는 다음과 같이 정의된다.

$$Gini = 1 - \sum_{k=1}^m p_k^2$$

- 위에서  $P_k$  는 카테고리  $k$ 에 대한 분포 확률을 말한다. 예를 들어 아래와

같이 원과 세모 두 카테고리 샘플들이 들어 있는 두 개의 그룹의 지니 계수를 구하면 다음과 같다.



**좌측 박스:** 지니( $7:3$ ) =  $1 - \left[ \left( \frac{7}{10} \right)^2 + \left( \frac{3}{10} \right)^2 \right] = 1 - (0.49 + 0.09) = 0.42$

**우측 박스:** 지니( $5:5$ ) =  $1 - \left[ \left( \frac{5}{10} \right)^2 + \left( \frac{5}{10} \right)^2 \right] = 1 - (0.25 + 0.25) = 0.5$

- 위 결과를 보면 7:3으로 나누어진 그룹의 지니 계수가 작게 나타났다.

지니계수를 작을수록 순도가 높은 것이다 만일 10개 모두 같은 카테고리라면 지니 계수는  $1-[1] = 0.0$  된다.

## 엔트로피

- 엔트로피(entropy)도 지니와 유사하게 순도를 나타낸데 사용된다. 한 노드(그룹)에 여러 클래스가 골고루 균일하게 섞여 있을 때는 엔트로피가 가장 높고, 동종의 클래스로 모여 있을수록 엔트로피가 낮다.
- 엔트로피의 정의는 아래와 같으며 위와 같은 분류시의 엔트로피를 구하면 아래와 같다.

$$Entropy = - \sum_{k=1}^m p_k \log_2(p_k)$$

$$\text{좌측}(7:3) = -[0.7 \cdot \log_2(0.7) + 0.3 \cdot \log_2(0.3)] = -[(-0.36) + (-0.52)] = -(-0.88) = 0.88$$

$$\text{우측}(5:5) = -[0.5 \cdot \log_2(0.5) + 0.5 \cdot \log_2(0.5)] = -[(-0.5) + (-0.5)] = -(-1) = 1$$

- 지니를 사용하든 엔트로피를 사용하든 결정 트리 모델 성능에는 큰 차이가 없다. 단 지니의 계산 속도가 조금 빠르고 (로그를 계산하지 않으므로), 엔트로피를 사용하면 트리가 좀 더 잘 나누어지는 성향이 있다.

## 정보량과 엔트로피

- 엔트로피를 위와 같이 정의한 데에는 정보량의 개념에서 출발했다. 정보량이란 어떤 데이터가 포함하고 있는 정보의 총 가치를 나타낸다. 정보량을 표현하기 위해 그 사건이 발생할 확률을 사용했다.
- 예를 들어 내일 해가 동쪽에서 뜬다는 사건은 정보량이 얼마나 될까? 발생할 확률이 1인 사건은 아무런 정보가 없는 것과 같다. 복권에 당첨된 사실은 정보량은 매우 크다고 하겠다.
- 즉, 정보량은 일어날 확률의 역수에 비례한다고 가정하여 어떤 사건이 발생할 확률이  $p$ 이면 그 사건의 정보량을 다음과 같이 정의하였다.

$$\log\left(\frac{1}{p}\right)$$

- 여기서 로그를 붙인 이유는 사람이 어떤 현상에 대해 실제로 느끼는 체감 수치를 나타내기 위해서이다. 같은 량의 감정의 변화를 느끼려면 절대량이 아니라 현재 내가 로그에 비례하는 변화가 주어져야 한다.
- 이러한 현상은 사람이 자연적으로 느끼는 느낌의 양을 수학적 모델로 설명할 때 자주 등장한다. 이는 소리, 빛, 냄새에 대한 사람의 반응 등에서도 비슷하게 나타나는 현상이다.
- 따라서 어떤 사건의 "정보량의 기대치"를 구하려면 그 사건의 정보량  $\log(1/p)$ 에 그 사건이 발생할 확률( $p$ )을 곱해주어야 하며 이를 식으로

표현하면 아래와 같다.

$$p \log\left(\frac{1}{p}\right) = -p \log(p)$$

- 위와 같이 어떤 사건이 갖는 정보량 기대치를 엔트로피(entropy)라고 부른다.
- 위 식의 의미를 생각해보자. 어떤 사건이 발생할 확률이 거의 없으면 즉,  $p \approx 0$ 이면  $1/p$  값은 매우 커지만 이 사건이 발생할 확률  $p$ 가 거의 0이 되어 엔트로피는 0이 된다. 반면에  $p$ 가 1에 가까우면 즉, 발생이 거의 확실하면  $1/p \approx 10$  되고  $\log(10) \approx 0$ 으로 엔트로피 값은 거의 0이 된다.

- 즉, 엔트로피는  $p$ 가 너무 작아도 0으로 수렴하고  $p$ 가 너무 커서 1이 되어도 0으로 수렴한다. 아래 그림에 확률  $p$ 에 대한 엔트로피를 나타냈다. 엔트로피는  $p=0.5$ 일 때 가장 크다. 결정 트리 모델에서 보면 이때가 순도가 가장 낮은 때이며 엔트로피를 줄이는 방향으로 학습을 시키는 것이다.

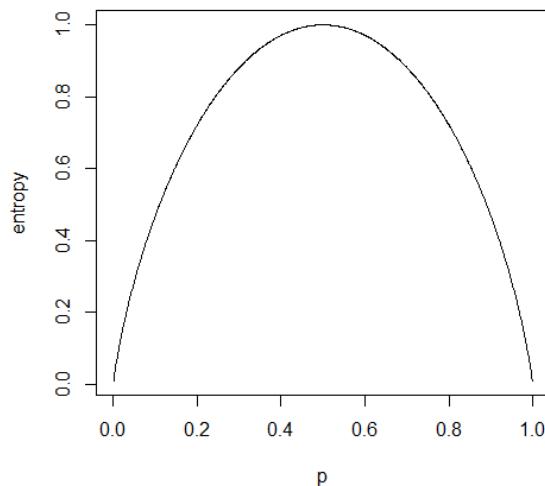


그림 5-8. 사건의 발생  
확률과 엔트로피

## 트리 종료 조건

- 결정 트리 모델에서는 트리를 언제까지 나눌지, 즉, 트리 만드는 작업을 언제 종료할지를 정해주어야 한다. 결정 트리를 계속 만들어 상세하게 분류를 하면 언젠가는 훈련 데이터에 대해서 100% 순도의 분류가 가능하다.
- 즉, 모든 마지막 노드에는 모두 동일한 클래스의 샘플들로만 들어 있게 만들 수 있다. 그런데 이렇게 100% 분류를 하는 트리는 과대적합된 것이므로 테스트 데이터에 대해서는 성능이 오히려 떨어지게 된다.
- 결정 트리 모델은 트리를 만드는 깊이를 제한하지 않으면 과대적합될 위험이 높으므로 주의해야 한다. 검증 데이터에 대해 최적의 성능을 내려면 적절한 깊이까지만 트리를 만들어야 한다.

- 결정 트리에서는 과대적합을 피하기 위해서 기본적으로 트리의 깊이 수를 제한한다. 물론 최대 깊이까지 도달하지 않았어도 모든 항목의 분류가 완성되면 그 가지는 더 이상 분류를 수행하지 않는다.
- 이외에 트리 생성의 종료 조건을 지정하는 대표적인 하이퍼 파라미터는 다음과 같다.

max\_depth: 트리의 최대 깊이 (이보다 깊은 트리를 만들지 않는다)

max\_leaf\_nodes: 리프 노드의 최대 수 (리프 노드를 이보다 많이 만들지 않는다)

min\_samples\_split: 분할하기 위한 최소 샘플수 (이보다 작으면 분할하지 않는다)

min\_samples\_leaf: 리프 노드에 포함될 최소 샘플수 (이보다 작은 노드는

만들지 않는다)

max\_features: 최대 특성수 (분할할 때 이보다 적은 수의 특성만 사용한다)

- 한편 트리의 깊이를 적절한 값보다 너무 작게 제한하면 과소적합이 된다. 과대접합과 과소적합을 모두 피하려면 트리의 깊이 등을 바꾸면서 언제 검증 데이터의 성능이 최적이 되는지를 봐야 한다.
- 일반적으로는 이러한 작업을 수동으로 할 필요는 없고 자동으로 최적의 값을 찾아준다.

## 유방암 분류

- 유방암 데이터에 대하여 DecisionTreeClassifier를 사용하여 유방암 진단 결과인 M(악성: malignant), B(양성: benign)을 분류하는 모델이다. 훈련 데이터와 검증 데이터는 train\_test\_split으로 분리하였다.

```
from sklearn.datasets import load_breast_cancer  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model_selection import train_test_split  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd
```

```
cancer = load_breast_cancer()
np.random.seed(9)
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
print(clf.score(X_test, y_test))
=>
0.916083916083916
```

- 테스트 데이터에 대해서 91.6%의 정확도를 보였다.

## 내부 변수

- 결정 트리 모델을 만든 후에, 어떤 특성이 결정 트리를 생성할 때 중요한 역할을 했는지 비중을 파악할 수 있다. 이 결과를 보고 중요하지 않은 특성은 향후에 제외하기도 한다.
- 유방암 예측의 경우 분류에 비중있게 사용된 특성 값을 알려주는 내부변수를 출력해 보는 예를 소개하겠다. 유방암 예측 데이터는 `sklearn.datasets`에 들어있다.
- 결정 트리를 만드는데 각 특성이 기여한 정도는 모델의 내부 변수인 `feature_importances_`에서 확인할 수 있다.
- 아래는 유방암 데이터의 각 특징의 이름과 모델에서 가지고 있는 중요도와

매칭시켜서 킨 후 리스트로 만들어서 출력하는 함수이다. 하나의 리스트로 합치는데에는 zip() 함수를 이용했고 소수점 4째 자리까지만 표시되게 하였다. 리스트는 앞의 10개만 출력했다.

```
list(zip(cancer.feature_names, clf.feature_importances_.round(4)))[10]  
=>
```

```
[('mean radius', 0.0),  
 ('mean texture', 0.0417),  
 ('mean perimeter', 0.0),  
 ('mean area', 0.0),  
 ('mean smoothness', 0.0),  
 ('mean compactness', 0.0),
```

```
('mean concavity', 0.0),  
('mean concave points', 0.0426),  
('mean symmetry', 0.0114),  
('mean fractal dimension', 0.0)]
```

- 중요도가 높은 순서로 표시하기 위해서 데이터 프레임을 만들고 중요도가 높은 상위 10개 특성을 출력하면 아래와 같다.

```
df =  
pd.DataFrame({'feature':cancer.feature_names,'importance':clf.feature_importances_ })
```

```
df=df.sort_values('importance', ascending=False)
```

```
print(df.head(10))
```

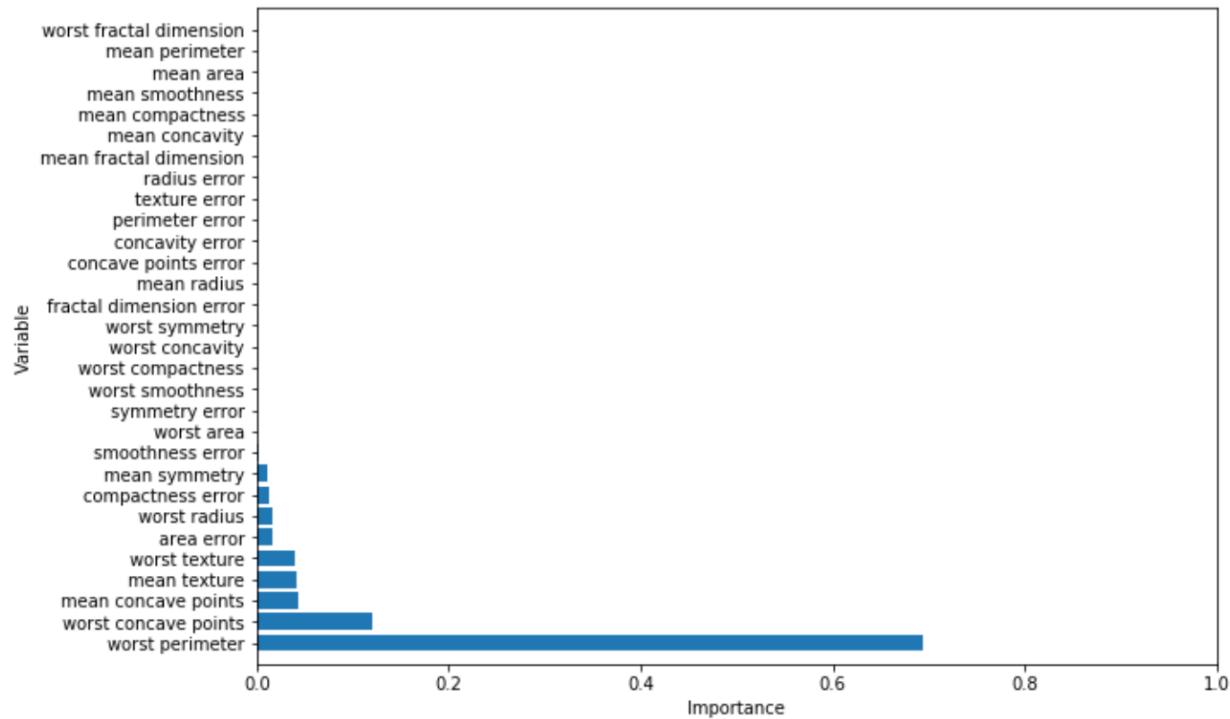
```
=>
```

	feature	importance
22	worst perimeter	0.694689

27	worst concave points	0.121068
7	mean concave points	0.042647
1	mean texture	0.041720
21	worst texture	0.039639
13	area error	0.017216
20	worst radius	0.017188
15	compactness error	0.012042
8	mean symmetry	0.011405
14	smoothness error	0.002385

- 위의 결과를 보면 worst radius가 트리를 만드는데 가장 기여를 많이 한 것을 알 수 있다. 아래는 각 특성의 기여도를 바 차트로 그리는 코드이다. 바 차트를 수평(horizontal)로 그리기 위해서 barh 함수를 사용하였다.

```
x = df.feature  
y = df.importance  
ypos = np.arange(len(x))  
  
plt.figure(figsize=(10,7))  
plt.barh(x, y)  
plt.yticks(ypos, x)  
plt.xlabel('Importance')  
plt.ylabel('Variable')  
plt.xlim(0, 1)  
plt.ylim(-1, len(x))  
plt.show()
```



## 결정 트리의 특징

- 결정 트리는 거의 모든 종류의 분류에서 사용할 수 있는 범용 모델이다. 결정 트리의 가장 큰 장점은 알고리즘의 동작을 쉽게 남에게 설명할 수 있다는 것이다.
- 예를 들어 고객에게 왜 대출을 해주지 않았는지, 당신의 신용도가 왜 낮은지, 왜 합격시키지 않았는지 등 이유를 설명해야 할 때 유용하다.
- 결정 트리의 또 다른 장점은 특성 변수의 스케일링이 필요 없다는 것이다. 트리 모델에서는 변수간의 연산이 없기 때문이다.
- 각 노드에서는 한 번에 한 특성을 검토하여 어떤 기준으로 트리를 나누면 순도가 올라가는지만 점검하면 되므로 다른 특성 값과의 관계를 계산할

필요가 없다. 따라서 정규화 등 스케일링이 필요 없다.

- 그러나 결정 트리는 훈련데이터가 바뀌면 모델의 구조가 달라지는 단점이 있다. 즉, 훈련 데이터에 따라 바뀌는 모델을 남에게 설명하기 어려울 수도 있다.

## 회귀에 사용

- 결정 트리 알고리즘을 분류가 아니라 회귀분석에도 사용할 수 있다. 이때는 DecisionTreeRegressor()를 사용하면 된다.
- 결정 트리를 회귀에 사용하면 각 노드에서 클래스를 예측하는 것이 아니라 목적변수의 수치 값을 예측하는 것만 다르다. 예측하는 타겟 값은 마지막 가지 노드에 있는 샘플의 평균치로 예측한다.
- 남녀 몸무게 예측 프로그램에서 결정트리를 사용하여 다음과 같이 회귀 분석을 할 수 있다.

```
from sklearn.tree import DecisionTreeRegressor  
dec_reg = DecisionTreeRegressor()
```

```
dec_reg.fit(X_train, y_train)  
print(dec_reg.score(X_test, y_test))  
=>  
0.893923784590028
```

## 타이타닉 예

- 결정 트리를 적용하는 예로 타이타닉에서 생존자를 예측하는 문제를 풀어보겠다. 먼저 타이타닉 데이터를 읽어 오는 코드는 아래와 같다. 데이터는 data 폴더에 미리 다운 받았다고 가정한다.
- 훈련 데이터는 총 891명의 데이터이고 상위 몇 명의 정보를 보면 아래와 같다. 데이터를 읽을 때 승객 고유번호를 인덱스로 사용하도록 지정하였다.

```
import pandas as pd
```

```
train = pd.read_csv("./data/titanic_train.csv", index_col=["PassengerId"])
```

```
print(train.shape) # (891, 11)
```

```
train.head()
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId											
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- 타이타닉 데이터 속성 정보는 아래와 같다.

Survival – 생존여부: 0 = No, 1 = Yes

Pclass – 티켓 등급: 1 = 1st, 2 = 2nd, 3 = 3rd

Sex – 성별: male, female

Age – 나이

SibSp – 동승한 형제, 배우자 수

Parch – 부모와 자녀의 수

Ticket – 티켓 번호

Fare – 승선 요금

Cabin – 캐빈(객실) 번호

Embarked – 승선한 항구: C = Cherbourg, Q = Queenstown, S = Southampton

- 테스트 데이터를 읽는다.

```
test = pd.read_csv("./data/titanic_test.csv", index_col=["PassengerId"])
```

- 성별을 카테고리 변수로 표현하는 컬럼을 추가한다.

```
train.loc[train["Sex"] == "male", "Sex_encode"] = 0
```

```
train.loc[train["Sex"] == "female", "Sex_encode"] = 1
```

```
print(train.shape)
```

```
train[["Sex", "Sex_encode"]].head()
```

- 테스트 데이터에 대해서도 같은 작업을 한다.

	<b>Sex</b>	<b>Sex_encode</b>
<b>PassengerId</b>		
1	male	0.0
2	female	1.0
3	female	1.0
4	female	1.0
5	male	0.0

```
test.loc[test["Sex"] == "male", "Sex_encode"] = 0
```

```
test.loc[test["Sex"] == "female", "Sex_encode"] = 1
```

```
print(test.shape)  
test[["Sex", "Sex_encode"]].head()
```

- 요금이 없는 항목을 찾아보니 하나가 있다.

```
test[pd.isnull(test["Fare"])]  
## 출력
```

Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId									
1044	3 Storey, Mr. Thomas	male	60.5	0	0	3701	NaN	NaN	S

- 요금이 비어 있는 항목은 요금의 평균 값으로 채운다.

```
mean_fare = train["Fare"].mean()  
print("평균 요금 = ${0:.3f}".format(mean_fare))  
## 출력  
평균 요금 = $32.204
```

```
test.loc[pd.isnull(test["Fare"])]["Fare"] = mean_fare  
test[pd.isnull(test["Fare"])]  
## 출력
```

Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
<b>PassengerId</b>									

- 승선한 항구 이름은 카테고리 변수이므로 이를 카테고리형 변수로 바꾸기

위해서 더미 코딩을 한다. 아래와 같이 앞에 접두어로 “Embarked”를 붙인 세 개의 특성을 새로 만었다.

```
train_embarked = pd.get_dummies(train["Embarked"], prefix="Embarked")  
train_embarked.head()
```

	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1

- 테스트 데이터에 대해서도 같은 처리를 한다.

```
test_embarked = pd.get_dummies(test["Embarked"], prefix="Embarked")
test_embarked.head()
##
```

PassengerId	Embarked_C	Embarked_Q	Embarked_S
892	0	1	0
893	0	0	1
894	0	1	0
895	0	0	1
896	0	0	1

- 새로 만든 컬럼을 기존의 훈련 및 테스트 데이터에 붙인다. 테스트 데이터의 내용을 확인해 보았다.

```
train = pd.concat([train, train_embarked], axis=1)
```

```
test = pd.concat([test, test_embarked], axis=1)  
test[["Embarked", "Embarked_C", "Embarked_Q", "Embarked_S"]].head()  
##
```

	Embarked	Embarked_C	Embarked_Q	Embarked_S
PassengerId				
892	Q	0	1	0
893	S	0	0	1
894	Q	0	1	0
895	S	0	0	1
896	S	0	0	1

- 일부 데이터로 먼저 학습하기 위해서 일부 클래스, 성별, 요금, 탑승한항구 컬럼만 선택했다.

```
feature_names = ["Pclass", "Sex_encode", "Fare",
```

```
"Embarked_C", "Embarked_Q", "Embarked_S"]
```

```
X_train = train[feature_names]  
y_train = train["Survived"]  
from sklearn.tree import DecisionTreeClassifier  
model = DecisionTreeClassifier(max_depth=2,random_state=7)  
model.fit(X_train, y_train)
```

- 결과를 그래프로 그려본다.

```
from sklearn.tree import export_graphviz  
import graphviz
```

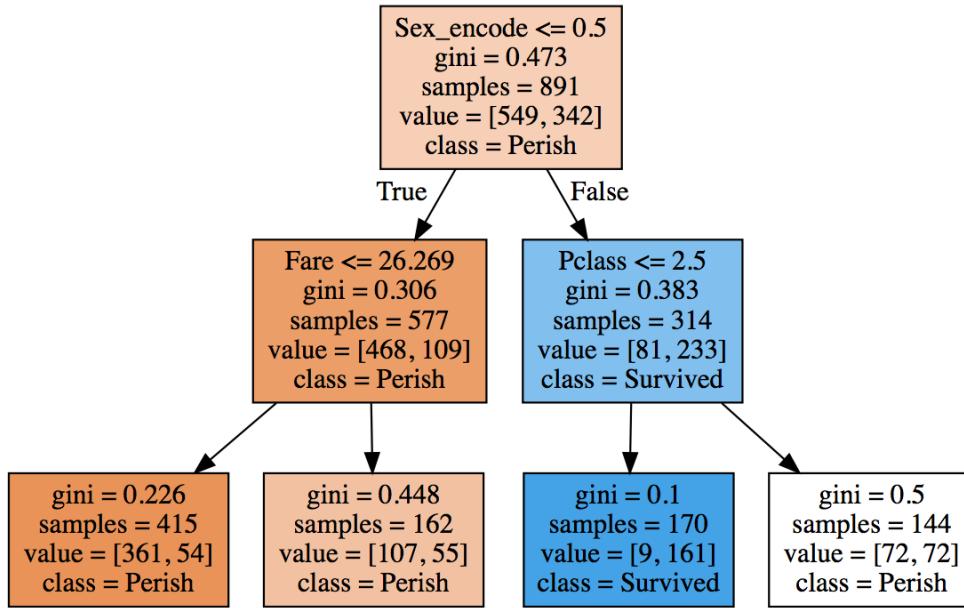
```
export_graphviz(model,  
                feature_names=feature_names,  
                class_names=["Perish", "Survived"],  
                out_file="decision-tree.dot")
```

```
with open("decision-tree.dot") as f:
```

```
    dot_graph = f.read()
```

```
graphviz.Source(dot_graph)
```

```
## 결과
```



- 트리 구조를 간단히 만들어 보기 위해서 깊이를 2단계 까지만 수행했다.  
위 그림을 보면 첫 번째 루트 노드에서는 성별을 판단했다.
- 성별 코드가 0.5 보다 작으면 남성("0")을 가리키고 이것이 True이면

좌측 노드로 가게 된다. 그림에서 좌측 노드는 2단계에서 모두 사망으로 분류되었다. 여성이면 우측으로 가고 객실의 등급이 2.5보다 적으면 (즉, 10이나 2등급이면) 살아날 확률이 큰 것으로 분류되는 것을 알 수 있다.

- 이제 테스트 데이터를 가지고 예측을 수행하면 아래와 같다. 테스트 데이터 앞의 20명의 속성 값과 생존을 예측한 결과를 보였다.

```
X_test = test[feature_names]  
prediction = model.predict(X_test)
```

```
X20 = X_test[:20]  
X20["Survived"] = prediction[:20]  
X20  
##
```

	Pclass	Sex_encode	Fare	Embarked_C	Embarked_Q	Embarked_S	Survived
PassengerId							
892	3	0.0	7.8292	0	1	0	0
893	3	1.0	7.0000	0	0	1	0
894	2	0.0	9.6875	0	1	0	0
895	3	0.0	8.6625	0	0	1	0
896	3	1.0	12.2875	0	0	1	0
897	3	0.0	9.2250	0	0	1	0
898	3	1.0	7.6292	0	1	0	0
899	2	0.0	29.0000	0	0	1	0
900	3	1.0	7.2292	1	0	0	0
901	3	0.0	24.1500	0	0	1	0
902	3	0.0	7.8958	0	0	1	0
903	1	0.0	26.0000	0	0	1	0
904	1	1.0	82.2667	0	0	1	1
905	2	0.0	26.0000	0	0	1	0
906	1	1.0	61.1750	0	0	1	1
907	2	1.0	27.7208	1	0	0	1
908	2	0.0	12.3500	0	1	0	0
909	3	0.0	7.2250	1	0	0	0
910	3	1.0	7.9250	0	0	1	0
911	3	1.0	7.2250	1	0	0	0

- 위의 결과를 보면 앞의 20명 중에 생존으로 예측한 사람은 모두 여성인 것을 알 수 있다. (904, 906, 907 번 승객이 모두 여성이다.)

## 7.5. 랜덤 포레스트

- 결정 트리의 성능을 개선한 방법으로 랜덤 포레스트(Random Forest) 방식이 있다. 이 방식은 비교적 간단한 구조의 결정 트리 들을 수십~수백개를 랜덤하게 만들고 각 결정 트리의 동작 결과의 평균치를 구하는 방법이다.
- 이렇게 여러 개의 모델을 만들고 평균을 구하는 방식을 앙상블(ensemble) 방법이라고 하며 하나의 모델만 만드는 것보다 좋은 성능을 보인다.

## 동작 원리

- 랜덤 포레스트에서는 주어진 훈련 데이터를 모두 한 번에 사용해서 하나의 최상의 트리 모델을 만드는 방식이 아니라, 데이터의 일부 또는 속성의 일부만 랜덤하게 선택하여 결정 트리를 다양하게 만들고 그 결과의 평균치를 취하는 방식이다.
- 나무(tree)가 많이 모였다는 의미로 숲(forest)라는 용어를 사용했다.
- 예를 들어 어떤 사람이 얼마나 훌륭한지를 평가한다고 할 때, 그 사람을 아는 모든 사람에게(예를 들어 1000명에게) 모두 묻고 수집된 모든 데이터를 사용하여 한 번에 판단하는 것보다, 그 사람을 아는 사람들을 랜덤한 조합으로 (예를 들어 50명씩 선택) 총 300개의 조합을 만들어 평가를 하고 이들 평가 점수의 평균치를 구하는 것과 같다.

- 여기서 1000명 중에 50명씩을 랜덤하게 선택하는 경우의 수는 무수히 많은 조합이 가능하다. 또한 각 결정 트리를 만들 때 사용하는 속성을 선택하는 데에도 랜덤한 선택을 함으로써 보다 일반화된 동작을 할 수 있게 한다.
- 다른 예로 나의 모든 건강 정보를 가장 훌륭한 의사 한명에게만 주고 진단을 받는 것보다 나의 건강정보의 일부를 랜덤하게 선택하는 것을 100번 수행하여(예를 들면 배깅 방법으로), 100명의 의사에게 각각 진단을 구하고 이들의 평균치(양상블)을 최종 진단으로 정하는 방식이 더 우수하다는 것이다.
- 이렇게 샘플도 랜덤하게 선택하고, 속성도 랜덤하게 선택하여 다수의 결정 트리를 만들고 이의 평균을 구하면 단일 결정 트리를 사용하는 것보다

안정적이고 우수한 성능을 낸다.

- 랜덤 포레스트의 단점은 가장 우수한 결정 트리를 하나만 만드는 것이 아니므로 모델의 동작을 한가지 트리를 기준으로 설명할 수 없다는 것이다. 또한 하나의 결정 트리를 만드는 방식보다 계산량이 많아진다.

## 양상블 기법

- 양상블 기법에서는 여러개의 작은 모델의 평균 값을 구하거나 투표를 통하여 최적의 값을 찾는 절차가 필요하다. 이는 랜덤 포레스트 뿐 아니라, 다양한 양상블 기법에서 공통으로 필요한 절차이다.
- 먼저 여러 모델의 결과를 가지고 최적의 타겟 변수를 찾아내기 위해서 투표가 필요한 경우가 많은데 투표에는 직접 투표와 간접 투표가 있다.

### 직접투표와 간접투표

- 직접투표(hard voting)란 각 세부 모델이 예측한 최종 클래스에 1점으로 주고 가장 많은 점수를 받은 클래스를 최종 타겟 변수라고 분류를 하는 방법을 말한다.

- 반면에 간접투표(soft voting)란 각 세부 모델이 하나의 결과만 선택하는 것이 아니라 각 클래스에 속할 확률을 제공하고 이 확률을 모두 더해서 가장 큰 값을 받은 대상을 최종 타겟 변수로 분류 하는 방법을 말한다.
- 예를 들어 세 개의 세부 모델 A, B, C를 사용하여 어떤 샘플이 어떤 클래스에 속할지 분류하는 작업을 나누어 수행했다고 하자. 이 샘플이 두 개의 클래스 P 또는 Q에 속할 될 확률을 각각 아래와 같이 예측했다고 하자.

	P일 확률	Q일 확률	판정결과 (직접투표)
세부 모델 A	0.9	0.1	P
세부 모델 B	0.4	0.6	Q
세부 모델 C	0.3	0.7	Q
확률의 평균	$(1.6)/3 = 0.533$	$(1.4)/3 = 0.456$	

- 세부 모델 A는 이 샘플을 P라고 예측했고, 세부 모델 B와 C는 클래스 Q라고 예측했다. 이를 직접 투표 방식으로 판단하며 이 샘플은 Q를 두 표 얻었으므로 이 분류 예측 결과는 Q라고 판정한다.
- 그러나 간접 투표를 하면 P일 확률의 총합은 1.6, Q일 확률의 총합은 1.4가 되어 최종 판정은 P라고 판정하게 된다.

- 이와 같이 직접 투표와 간접 투표의 양상을 결과가 다르게 나타날 수 있다.
- 간접 투표를 하려면 각 세부 모델이 클래스별 추정 확률 값을 제공하는 기능이 있어야 한다. 모델에 따라서는 확률 예측을 제공하지 않는 경우도 있다.
- 양상을 방법을 분류가 아니라 회귀분석에도 사용할 수 있는데, 이때는 투표를 하는 대신 예측된 여러 수치 값의 평균치를 최종 예측값으로 제시한다.
- 어떤 방법이 더 나은 방법일까? 이는 문제의 성격에 따라 다르다. 또한 처리의 복잡도도 고려해야 한다.

## 배깅

- 모델을 학습시키기 위한 훈련 데이터를 선택할 때 원래 데이터 셋에서 데이터를 취하는 방법으로 배깅 방식이 널리 사용된다.
- 배깅이란 bootstrap aggregation의 줄임말이며 전체 훈련 데이터에서 “중복을 허용”하여 데이터를 샘플링을 하는 방법이다.
- 중복을 허용하므로 같은 데이터가 중복되어 선택될 수 있다. 배깅을 bootstrap resampling의 줄임말로 부트스트래핑이라고도 부른다.
- 배깅과 달리 주어진 원래 데이터에서 중복을 허용하지 않고, 즉, 한 번 샘플링 된 것은 다음 샘플링에서 제외하는 방식은 페이스팅(pasting)이라고 한다.

- 배깅으로 샘플을 추하면 전체 데이터 중에서 평균 63%의 데이터만 선택이 된다.
- (참고)  $n$ 개의 샘플에서 랜덤하게 하나를 선택할 때 어떤 샘플이 선택되지 않을 확률은  $(1-1/n)$ 이다. 이를  $n$ 번 시행해도 계속 선택되지 않을 확률은 독립사건이  $n$  번 일어날 확률이므로  $(1-1/n)^n$ 이 된다.  $n$ 이 무한대로 커지면(즉, 샘플의 수가 커지면) 이 값은  $(1 - 1/e)=0.6321$ 로 수렴한다.)
- 배깅을 수행하면 학습에 선택되지 않는 샘플은 평균 37%가 되는데 이 샘플을 oob(out of bag) 샘플이라고 한다. 이 oob 데이터를 검증 데이터로 사용하면 별도로 검증 데이터를 나누지 않고도 평가를 할 수 있어 편리하고 공정한 평가를 할 수 있게 된다.

- 결정 트리 구조에 배깅을 적용한 방식이 바로 랜덤 포레스트 모델이다.
- 랜덤 포레스트에서는 또한 각 속성들을 분석에 모두 사용하지 않고 일부만 랜덤하게 선택하게 한다. 이는 max\_features 인자로 지정하는데 이 값을 0.5로 하면 주어진 특성 수의 50 %를 랜덤하게 선택하여 분석에 사용한다.
- 이를 auto로 지정하면 전체 특성수의 제곱근의 수만 사용한다. 예를 들어 특성의 수가 100개가 있었다면 이중에 랜덤하게 선택된 10개의 속성만 사용한다.

## 특성 중요도

- 랜덤 포레스트도 결정 트리에서와 같이 어떤 특성 값이 트리를 잘 나누는데 많이 기여를 많이 했는지를 알려준다. 여기서 잘 나누는 것의 기준은 트리를 나눈 후에 불순도가 얼마나 그 이전의 트리 구조에 비해 감소했는지를 평가한다.
- 특성 중요도 값은 내부 변수 `feature_importances_`에 들어 있다. 랜덤 포레스트를 사용하면 결정 트리를 사용한 경우와 비교하여 특성들이 골고루 사용된 것을 알 수 있다.

## 부스팅 알고리즘

- 간단한 학습 모델들을 여러 개 사용하여 성능을 향상 시키는 양상을 방법 중에 부스팅 알고리즘이 있다.
- 랜덤 포레스트 방식은 간단한 결정 트리를 다수 만들어 각각 독립적으로 실행시킨 후에 이들을 평균을 구하는 것이며, 부스팅 알고리즘은 앞의 모델을 보고 성능을 순차적으로 점차 개선하는 방식으로 동작한다.
- 학습하는 동안 모델 자체의 기능을 점차 개선한다고 하여 부스팅이라는 이름을 붙였다. 부스팅에는 아다(ada) 부스트와 그라디언트 부스트가 널리 사용된다.
- 아다부스트(adaptive boosting)에서는 앞에서 사용한 세부 모델에서 과소

적합했던 샘플, 즉 분류에 실패한 샘플의 가중치를 높여주는 것이다. 즉, 소외되었던 샘플을 주목하여 학습을 다시 시키는 방식이다.

- 부스팅 방법을 “연속된” 학습기법이라고도 부르는데, 앞의 세부모델이 결과를 보고 두 번째 세부 모델을 만들 수 있으므로 순차적으로 작업이 이루어진다. 따라서 작업을 여러 디바이스로 나누어 병렬로 처리하지는 못한다. 랜덤 포레스트는 병렬작업이 가능하다.
- 부스팅 알고리즘은 결정 트리에만 적용되는 것이 아니라 임의의 학습 모델에 적용될 수 있다.

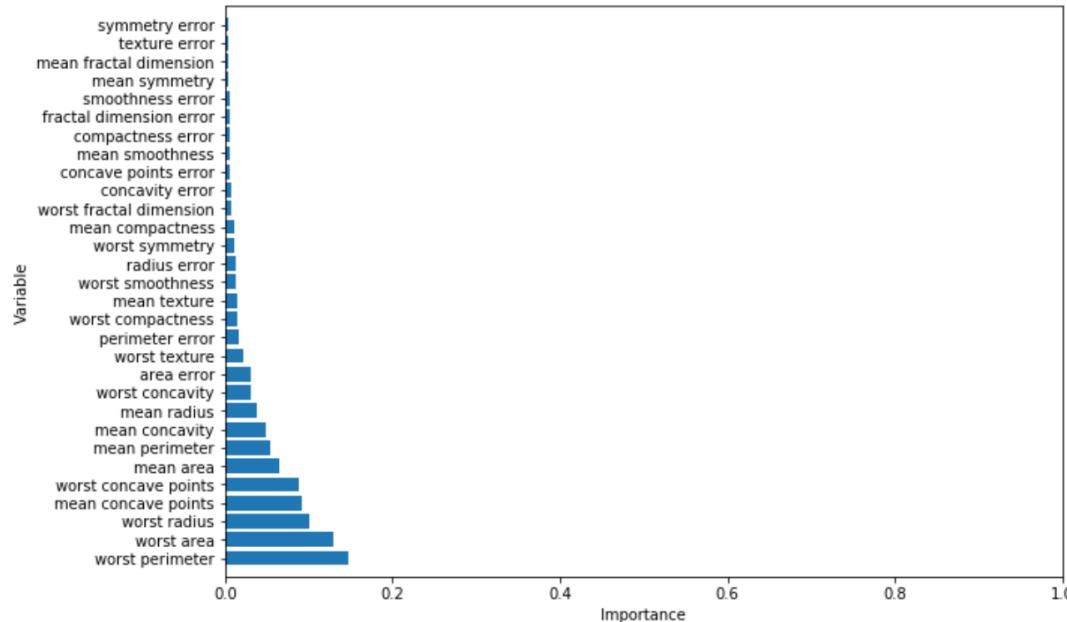
## 유방암 예측 예

- 앞에서 결정 트리를 사용하여 유방암 예측 문제를 풀어보았다. 여기서는 같은 데이터를 가지고 랜덤포레스트를 적용해 보겠다.

```
from sklearn.datasets import load_breast_cancer  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model_selection import train_test_split  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
cancer = load_breast_cancer()  
np.random.seed(9)
```

```
X_train, X_test, y_train, y_test = train_test_split(  
    cancer.data, cancer.target, stratify=cancer.target)  
  
rfc = RandomForestClassifier(n_estimators=500)  
rfc.fit(X_train, y_train)  
print(rfc.score(X_test, y_test))  
  
df =  
pd.DataFrame({'feature':cancer.feature_names,'importance':rfc.feature_importances_ })  
df=df.sort_values('importance', ascending=False)  
x = df.feature  
y = df.importance  
ypos = np.arange(len(x))
```

```
plt.figure(figsize=(10,7))
plt.barh(x, y)
plt.yticks(ypos, x)
plt.xlabel('Importance')
plt.ylabel('Variable')
plt.xlim(0, 1)
plt.ylim(-1, len(x))
plt.show()
=>
0.951048951048951
```



- 앞에서 설명한 결정 트리를 사용한 경우와 비교해보면 더 다양한 특성들을 사용한 것을 알 수 있다.

# 자전거 대여수 예측 예

- 여기서는 랜덤포레스트 알고리즘을 활용하는 예로 자전거 대여수를 예측하는 예를 소개한다. 이 예제는 캐글에서 소개된 것이다. 자전거 대여 기록 데이터는 아래 주소에서 다운받을 수 있다. 데이터를 data 폴더에 저장한 것을 가정하였다.

## 데이터 읽기

<https://goo.gl/s8qSL5>

```
import os, os.path, shutil  
import numpy as np  
import pandas as pd  
import seaborn as sns
```

```
import matplotlib.pyplot as plt  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model_selection import cross_val_score  
from sklearn.metrics import make_scorer  
%matplotlib inline
```

```
train = pd.read_csv("data/bike_train.csv", parse_dates=["datetime"])  
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

- 데이터의 기본적인 속성을 보면 다음과 같다.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10886 entries, 0 to 10885  
Data columns (total 12 columns):  
 datetime    10886 non-null datetime64[ns]  
 season      10886 non-null int64  
 holiday     10886 non-null int64  
 workingday  10886 non-null int64  
 weather      10886 non-null int64  
 temp         10886 non-null float64  
 atemp        10886 non-null float64
```

```
humidity      10886 non-null int64
windspeed     10886 non-null float64
casual        10886 non-null int64
registered    10886 non-null int64
count         10886 non-null int64
dtypes: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.6 KB
```

```
train["d-year"] = train["datetime"].dt.year
train["d-month"] = train["datetime"].dt.month
train["d-day"] = train["datetime"].dt.day
train["d-hour"] = train["datetime"].dt.hour
train["d-minute"] = train["datetime"].dt.minute
train["d-second"] = train["datetime"].dt.second
```

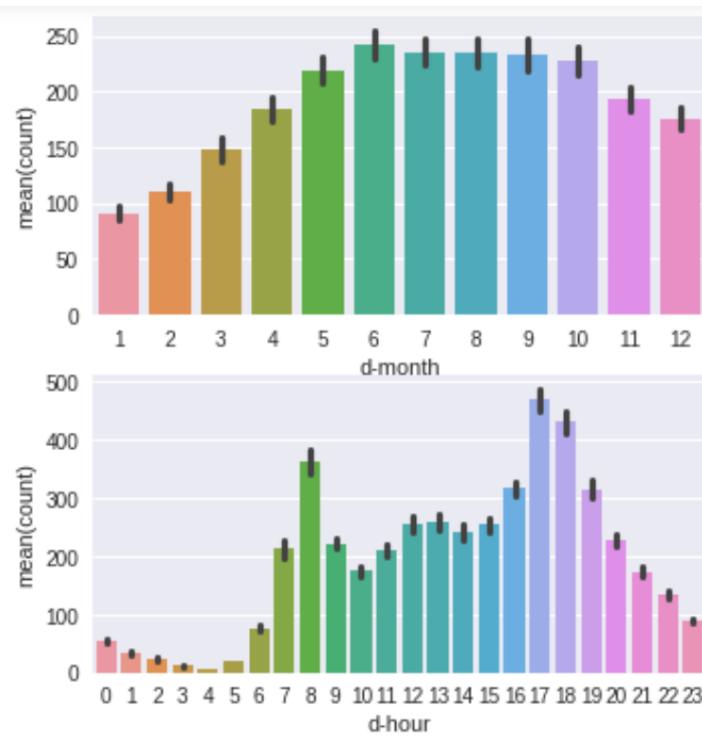
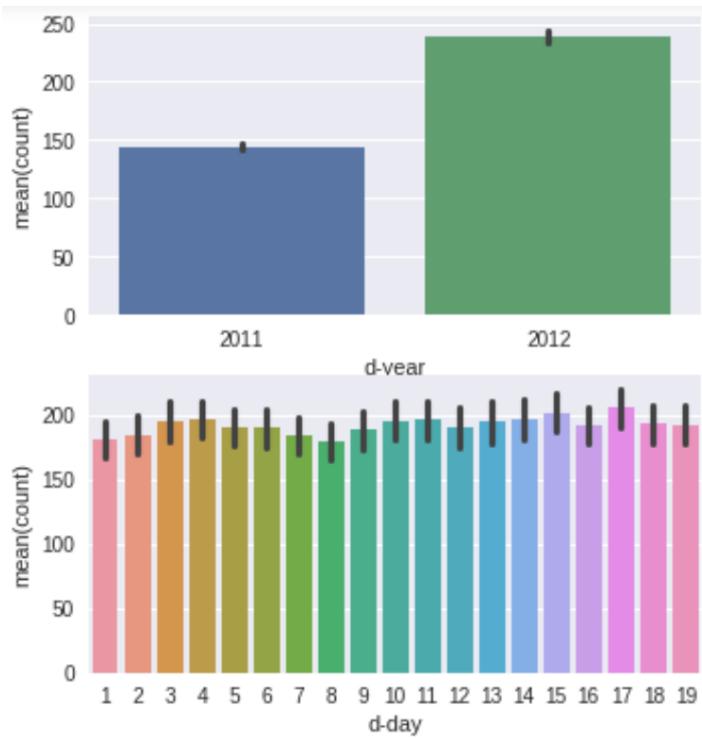
```
train[["datetime", "d-year", "d-month", "d-day", "d-hour", "d-minute",  
"d-second"]].head()
```

	datetime	d-year	d-month	d-day	d-hour	d-minute	d-second
<b>0</b>	2011-01-01 00:00:00	2011		1	1	0	0
<b>1</b>	2011-01-01 01:00:00	2011		1	1	1	0
<b>2</b>	2011-01-01 02:00:00	2011		1	1	2	0
<b>3</b>	2011-01-01 03:00:00	2011		1	1	3	0
<b>4</b>	2011-01-01 04:00:00	2011		1	1	4	0

```
figure, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2)
```

```
figure.set_size_inches(12, 6)

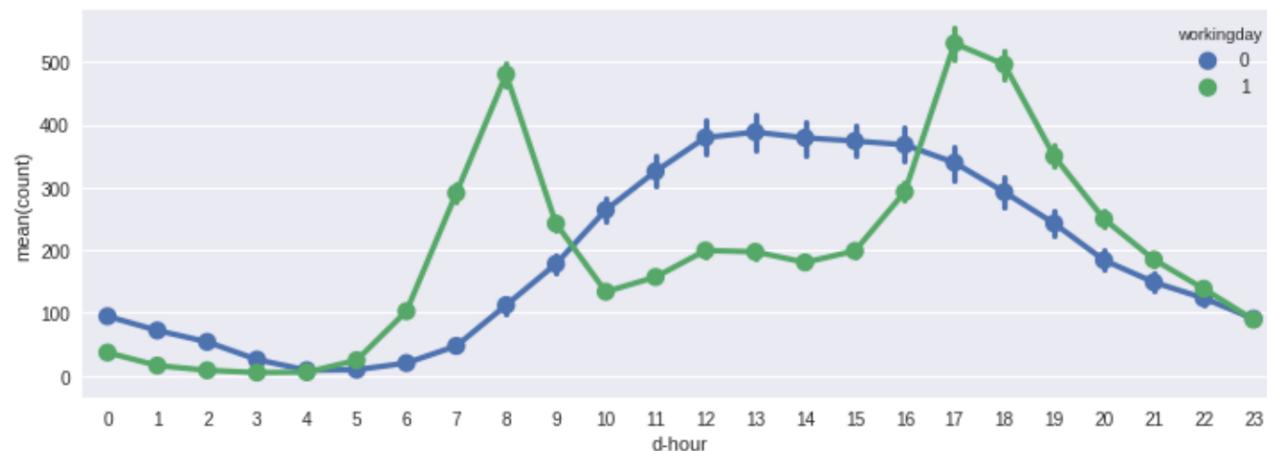
sns.barplot(data=train, x="d-year", y="count", ax=ax1)
sns.barplot(data=train, x="d-month", y="count", ax=ax2)
sns.barplot(data=train, x="d-day", y="count", ax=ax3)
sns.barplot(data=train, x="d-hour", y="count", ax=ax4)
```



- 주중(workingday==1)에는 출근 시간과 퇴근 시간에 자전거를 많이 대여하고, 주말(workingday==0)에는 오후 시간에 자전거를 많이 대여하는 것을 알 수 있다.

```
plt.figure(figsize=(12,4))
```

```
sns.pointplot(data=train, x="d-hour", y="count", hue="workingday")
```



- 이제 요일별 특성을 나누어보겠다.

```
train["d-dayofweek"] = train["datetime"].dt.dayofweek
```

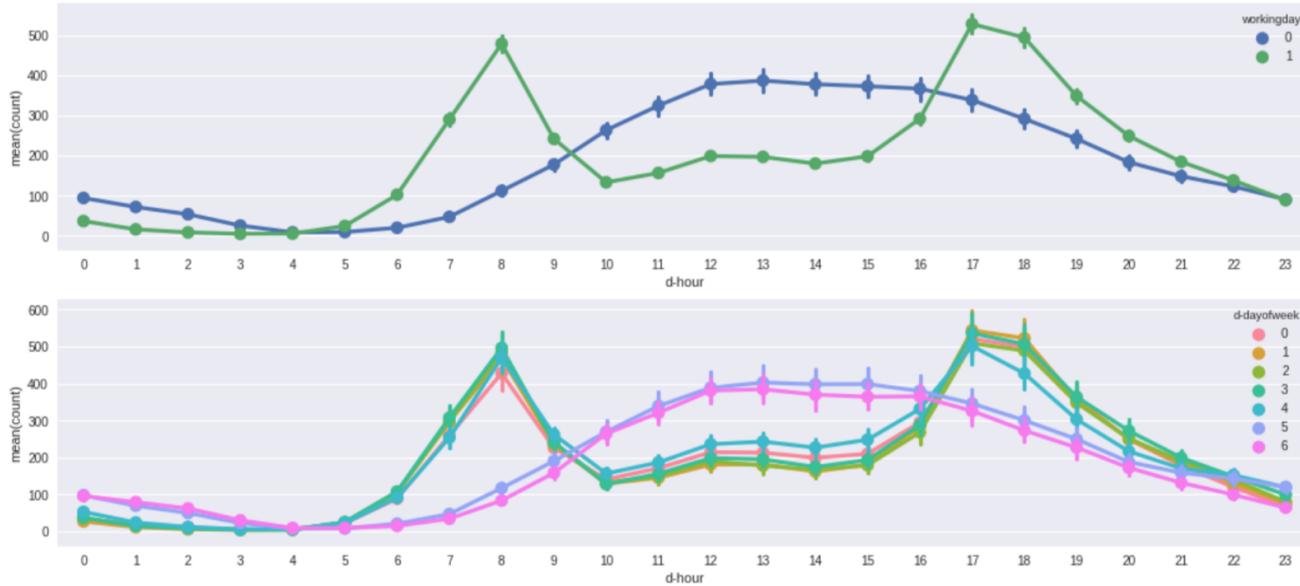
```
print(train.shape)
```

```
train[["datetime", "d-dayofweek"]].head()
```

	datetime	d-dayofweek
0	2011-01-01 00:00:00	5
1	2011-01-01 01:00:00	5
2	2011-01-01 02:00:00	5
3	2011-01-01 03:00:00	5
4	2011-01-01 04:00:00	5

```
figure, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
figure.set_size_inches(18, 8)
```

```
sns.pointplot(data=train, x="d-hour", y="count", hue="workingday", ax=ax1)
sns.pointplot(data=train, x="d-hour", y="count", hue="d-dayofweek", ax=ax2)
```



- 훈련에 사용할 특성을 아래와 같이 선택하였다.

```
features = ["season", "holiday", "workingday", "weather",
           "temp", "atemp", "humidity", "windspeed",
```

```
"d-year", "d-hour", "d-dayofweek"]
```

- 레이블 변수는 아래와 같이 실제 대여수에 로그를 취한 값을 택했다.

```
y_train = train["count"]
```

```
y_train = np.log(y_train + 1)
```

- 성능지표를 rmse에서 rmsle로 변경하여 사용한다.

```
def rmsle(predict, actual):  
    predict = np.array(predict)  
    actual = np.array(actual)
```

```
    log_predict = predict + 1
```

```
    log_actual = actual + 1
```

```
difference = log_predict - log_actual  
difference = np.square(difference)
```

```
mean_difference = difference.mean()  
score = np.sqrt(mean_difference)  
return score
```

```
rmsle_scorer = make_scorer(rmsle)
```

## 그리드 탐색

- 그리드 탐색을 먼저 수행해보겠다.

```
n_estimators = 30
```

```
max_depth_list = [10, 20, 30, 50, 100]
```

```
max_features_list = [0.1, 0.3, 0.5, 0.7, 0.9]
```

```
hyperparameters_list = []
```

```
for max_depth in max_depth_list:
```

```
    for max_features in max_features_list:
```

```
        model = RandomForestRegressor(n_estimators=n_estimators,
```

```
        max_depth=max_depth,  
        max_features=max_features,  
        random_state=11,  
        n_jobs=-1)  
  
score = cross_val_score(model, X_train, y_train, cv=10, \  
                       scoring=rmsle_scorer).mean()  
  
hyperparameters_list.append({  
    'score': score,  
    'n_estimators': n_estimators,  
    'max_depth': max_depth,  
    'max_features': max_features,  
})
```

```
print("Score = {0:.5f}".format(score))
```

- 성능이 좋은 순으로 나열해보겠다. rmsle는 작을수록 성능이 좋은 것으로 손실 함수로 사용된다.

```
hyperparameters_list = pd.DataFrame.from_dict(hyperparameters_list)
```

```
hyperparameters_list = hyperparameters_list.sort_values(by="score")
```

```
print(hyperparameters_list.shape)
```

```
hyperparameters_list.head()
```

	<b>max_depth</b>	<b>max_features</b>	<b>n_estimators</b>	<b>score</b>
<b>9</b>	20	0.9	30	0.389932
<b>14</b>	30	0.9	30	0.392554
<b>24</b>	100	0.9	30	0.392963
<b>19</b>	50	0.9	30	0.392963
<b>8</b>	20	0.7	30	0.396058

## 랜덤 탐색

- 랜덤 탐색은 아래와 같이 수행한다.

```
hyperparameters_list = []
```

```
n_estimators = 30
```

```
num_epoch = 25
```

```
for epoch in range(num_epoch):
```

```
    max_depth = np.random.randint(low=10, high=100)
```

```
    max_features = np.random.uniform(low=0.1, high=1.0)
```

```
model = RandomForestRegressor(n_estimators=n_estimators,
```

```
        max_depth=max_depth,  
        max_features=max_features,  
        random_state=11,  
        n_jobs=-1)  
  
score = cross_val_score(model, X_train, y_train, cv=10, \  
                       scoring=rmsle_scorer).mean()  
  
hyperparameters_list.append({  
    'score': score,  
    'n_estimators': n_estimators,  
    'max_depth': max_depth,  
    'max_features': max_features,  
})
```

```
print("Score = {:.5f}".format(score))
```

- 같은 방법으로 hyperparameters\_list 리스트를 점수 순으로 나열해 보면 아래와 같다.

	<b>max_depth</b>	<b>max_features</b>	<b>n_estimators</b>	<b>score</b>
	<b>3</b>	92	0.739906	30 0.393622
	<b>15</b>	67	0.744231	30 0.393622
	<b>17</b>	49	0.673060	30 0.395155
	<b>22</b>	41	0.558215	30 0.395571
	<b>24</b>	80	0.612872	30 0.395571

## 상세 검색

- 이제 범위를 좁혀 상세 검색을 하겠다.

```
hyperparameters_list = []
```

```
n_estimators = 50
```

```
num_epoch = 25
```

```
for epoch in range(num_epoch):
```

```
    max_depth = np.random.randint(low=40, high=100)
```

```
    max_features = np.random.uniform(low=0.5, high=1.0)
```

```
model = RandomForestRegressor(n_estimators=n_estimators,
```

```
        max_depth=max_depth,  
        max_features=max_features,  
        random_state=11,  
        n_jobs=-1)  
  
score = cross_val_score(model, X_train, y_train, cv=10, \  
                       scoring=rmsle_scoring).mean()  
  
hyperparameters_list.append({  
    'score': score,  
    'n_estimators': n_estimators,  
    'max_depth': max_depth,  
    'max_features': max_features,  
})
```

```
print("Score = {:.5f}".format(score))
```

- 점수순으로 나열하면 아래와 같다.

	<b>max_depth</b>	<b>max_features</b>	<b>n_estimators</b>	<b>score</b>
	<b>16</b>	66	0.798244	50 0.389918
	<b>23</b>	95	0.759411	50 0.389918
	<b>10</b>	63	0.755530	50 0.389918
	<b>0</b>	43	0.693175	50 0.391426
	<b>21</b>	96	0.659266	50 0.391426

## 최종 모델

- 최종 모델을 다음과 같이 선택한다.

```
model = RandomForestRegressor(n_estimators=300,  
                             max_depth=70,  
                             max_features=0.7,  
                             random_state=11,  
                             n_jobs=-1)
```

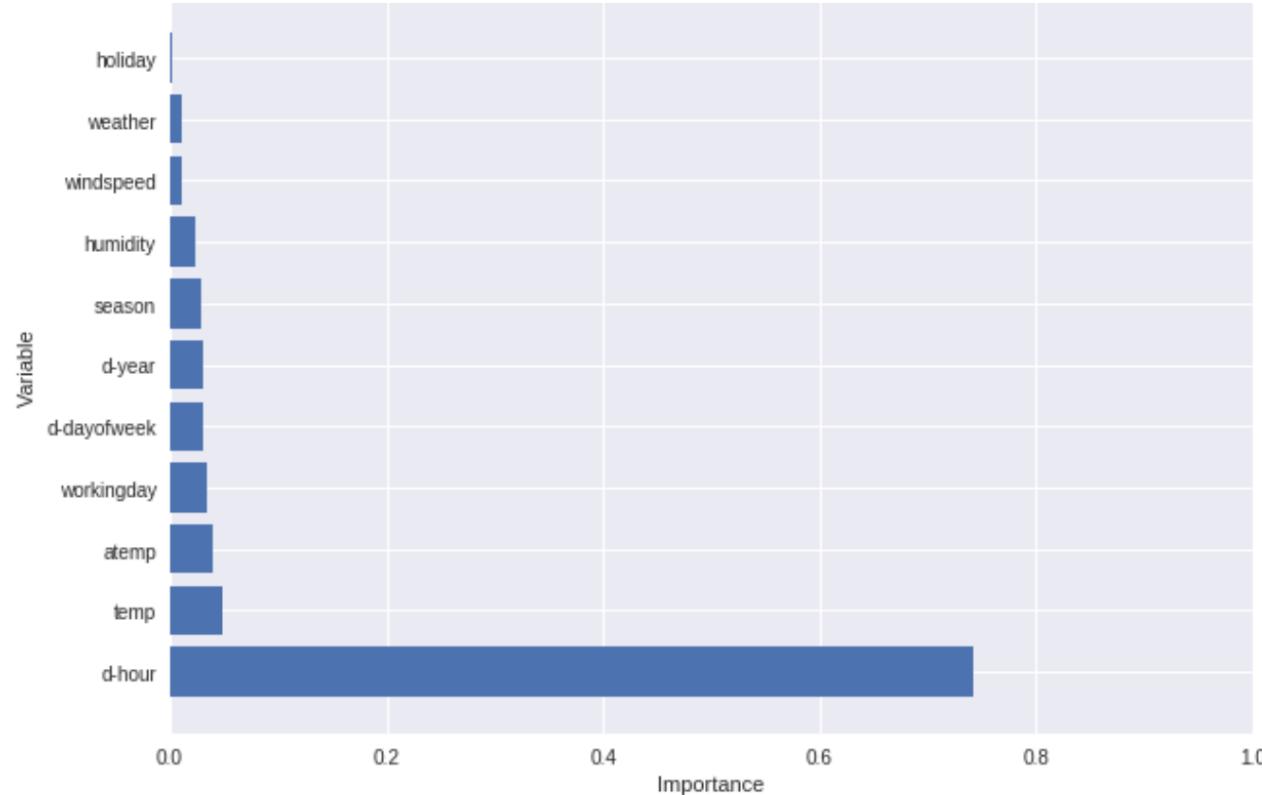
```
score = cross_val_score(model, X_train, y_train, cv=10, \  
                        scoring=rmsle_scorer).mean()  
  
print("Score = {:.5f}".format(score))  
  
=>  
Score = 0.38834
```

## 주요 특성 변수

```
df = pd.DataFrame({'feature':features,'importance':model.feature_importances_ })  
df=df.sort_values('importance', ascending=False)  
x = df.feature  
y = df.importance  
ypos = np.arange(len(x))  
  
plt.figure(figsize=(10,7))  
plt.barh(x, y)  
plt.yticks(ypos, x)  
plt.xlabel('Importance')  
plt.ylabel('Variable')  
plt.xlim(0, 1)
```

```
plt.ylim(-1, len(x))
```

```
plt.show()
```



## 그리드 탐색

- 하이퍼 파라미터가 가질 수 있는 전체 범위를 몇 개의 구간으로 나누어 일일이 하나씩 점검해보는 방식이다.
- 예를 들어, SVC 모델을 사용할 때 gamma 변수와 C 변수의 값을 각각 5가지, 4가지로 나누고 총  $4 \times 5 = 20$  가지 경우의 수만큼 하이퍼 파라미터를 바꾸면서 최적의 성능을 내는 조건을 찾는 방법이다.
- 총 20가지 경우의 수를 수행하면서 최고의 score를 얻는 gamma와 C 값을 찾는다.
- 머신러닝 알고리즘별로 설정 가능한 하이퍼 파라미터에는 다음과 같은 것들이 있다.

kNN: k값

결정 트리: 트리의 깊이, 분류 조건, 분류를 위한 최소 샘플수

SVM: 커널 타입, 커널 계수, 규제화 파라미터(감마, C)

랜덤포레스트: 트리수, 사용할 특성수, 분리 조건, 분류할 최소 샘플수

부스팅: 트리수, 학습률, 트리깊이, 분할할 조건, 분류할 최소 샘플 수

- 또한 과대적합을 피하기 위해서 수행하는 규제화용 파라미터도 하이퍼파라미터이다. 커널 SVM에서 감마 파라미터를 조절했었다. 신경망에서는 드롭아웃 등이 규제화 파라미터에 해당된다.

## GridSearchCV

- GridSearchCV() 함수를 사용하면 그리드 탐색을 하며 동시에 교차검증을 수행하여 예상되는 성능을 측정하기에 편리하다. GridSearchCV()로 생성한 모델 객체는 fit, predict, score 함수를 제공하며 fit을 호출할 때, 여러 파라미터 조합에 대해서 교차검증을 수행한다.
- 유방암 예측 모델에 그리드 탐색을 적용해보겠다.

```
from sklearn.datasets import load_breast_cancer  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model_selection import train_test_split  
from sklearn import model_selection, svm, metrics
```

```
from sklearn.grid_search import GridSearchCV  
  
import numpy as np
```

## 데이터

```
cancer = load_breast_cancer()  
np.random.seed(9)  
X_train, X_test, y_train, y_test = train_test_split(  
    cancer.data, cancer.target, stratify=cancer.target)  
  
# 랜덤포레스트  
rcf = RandomForestClassifier(n_estimators=100)  
rcf.fit(X_train, y_train)  
rcf.score(X_test, y_test)
```

## 그리드 탐색

```
#그리드 서치의 매개변수를 설정한다(C, gamma)
```

```
params = [{"C": [1,10,100,1000], "kernel":["linear"]},  
          {"C": [1,10,100,1000], "kernel":["rbf"], "gamma":[0.001, 0.0001]}]
```

```
#그리드 서치 실행
```

```
clf = GridSearchCV(svm.SVC(), params, n_jobs=-1 )
```

```
clf.fit(X_train, y_train)
```

```
print('최적값 :', clf.best_estimator_)
```

```
print('최적 score :', clf.best_score_)
```

```
#테스트 데이터로 최종 평가
```

```
score = clf.score(X_test, y_test)
```

```
print('최종 평가 =',score)
```

```
=>
```

최적값 : SVC(C=10, cache\_size=200, class\_weight=None, coef0=0.0,  
decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='linear',  
max\_iter=-1, probability=False, random\_state=None, shrinking=True, tol=0.001,  
verbose=False)

최적 score : 0.9507042253521126

최종 평가 = 0.965034965034965

- 최고의 성능평가를 얻는 하이퍼 파라미터 조합의 값을 얻은 후에는 전체 주어진 데이터를 가지고 훈련을 한번 더 수행하여 최종 모델을 만든다.
- 선택된 최적의 하이퍼 파라미터 값은 best\_estimator\_ 내부 변수에 저장되

며, 평가점수는 `best_score_` 에 저장된다.

## 랜덤 탐색

- 그리드 탐색은 개념은 단순하나 여러 경우의 수를 모두 탐색하는데 시간도 오래 걸리고 최적의 값을 놓치는 경우가 있다. 격자 모양의 파라미터 조합의 값 사이에 실제로 최적의 하이퍼 파라미터 값이 있었다면 이를 찾아낼 방법이 없다.
- 이렇게 최적값을 놓치는 문제점을 개선하고 탐색 시간도 줄이는 방법으로 하이퍼 파라미터를 랜덤하게 탐색하는 방법이 있다. 랜덤 탐색은 두 단계로 이루어진다.
- 먼저 규칙적으로 하이퍼 파라미터 값을 지정하는 것이 아니라 일정한 범위 내에서 랜덤하게 하이퍼 파라미터를 선택하여 성능을 실험하여 대체로 어떤 영역에서 성능이 좋은지를 찾는다.

- 다음에는 이 영역을 중심으로 세밀하게 탐색을 한다. 전체 범위를 대상으로 균일한 조건으로 탐색하는 것이 아니라, 크게 보고 성능이 좋은 영역을 먼저 찾고 그 영역 내에서 정밀하게 탐색을 하는 방법이다. 이를 위해서는 RandomizedSearchCV() 함수를 사용한다.

## 7.6. 로지스틱 회귀

### 개념

- 로지스틱 회귀분석(logistic regression)은 임의의 범위를 갖는 값으로부터 0과 1사이의 값을 예측하거나 이진 분류에 사용하는 알고리즘이다.
- 예를 들어 어떤 시험에 합격하기 위해서 공부한 시간과 실제로 합격여부의 데이터가 아래와 같다고 하자.

합격자 공부시간:

```
pass_time = [3, 8, 9, 9, 9.5, 10, 12, 14, 14.5, 15, 16, 16, 16.5, 17, 17, 17, 17.5, 20, 20, 20]
```

불합격자 공부시간:

```
fail_time = [1, 2, 2.1, 2.6, 2.7, 2.8, 2.9, 3, 3.2, 3.4, 3.5, 3.6, 3, 5, 5.2, 5.4, 6, 6.5, 7, 8]
```

- 합격자 및 불합격자의 데이터를 입력 X와 레이블 y로 각각 합치는 코드는 아래와 같다.

```
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.linear_model import LogisticRegression  
%matplotlib inline
```

```
X = np.hstack((pass_time, fail_time))  
y1=[1]*len(pass_time)  
y0=[0]*len(fail_time)
```

```
y = np.hstack((y1,y0))
```

(X,y)

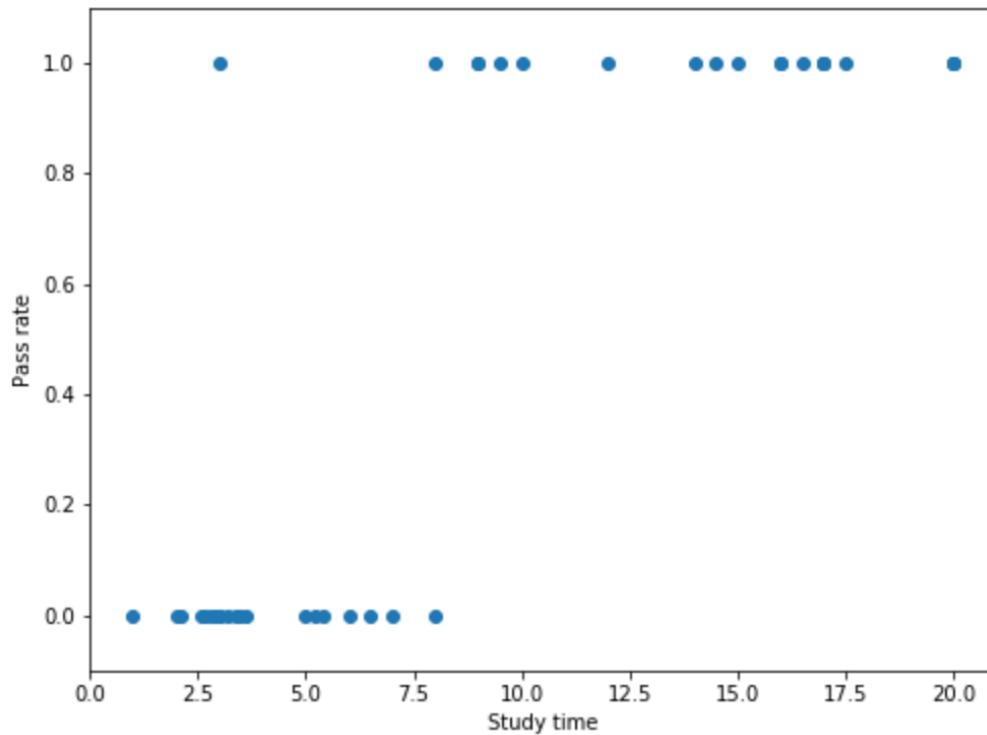
=>

```
(array([ 3. ,  8. ,  9. ,  9. ,  9.5, 10. , 12. , 14. , 14.5, 15. , 16. , 16. , 16.5, 17.  
       , 17. , 17. , 17.5, 20. , 20. , 20. , 1. , 2. , 2.1, 2.6, 2.7, 2.8, 2.9, 3. , 3.2,  
       3.4, 3.5, 3.6, 3. , 5. , 5.2, 5.4, 6. , 6.5, 7. , 8. ]),  
 array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,  
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0]))
```

- 이를 그래프로 그리면 다음과 같다.

```
fig = plt.figure(figsize=(8,6))  
plt.xlim(0, 21)
```

```
plt.ylim(-0.1, 1.1)
plt.xlabel("Study time")
plt.ylabel("Pass rate")
plt.scatter(X, y)
plt.show()
```

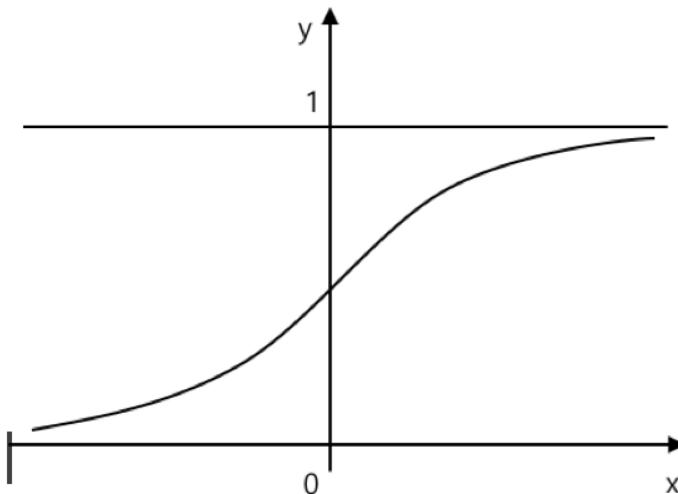


- 이와 같이 공부시간과 합격의 패턴이 알려진 시험에서 만일 어떤 사람이 10시간을 공부한다면 이 사람을 합격 예정자로 분류할 수 있을까? 또는 합격할 확률은 얼마로 예상할 수 있을까?

- 이러한 관계를 모델링하는데 로지스틱 회귀분석이 사용된다.
- 위와 같은 학습시간 대 합격의 관계는 S형 커브로 매핑하며 모델이 잘 동작한다는 것이 알려졌다. 선형 회귀에서 입출력의 관계를 직선으로 매핑한 것과 같이 여기서는 S형 커브를 사용하려는 것이다.
- S 커브로는 아래와 같이 시그모이드 함수를 사용한다.

$$p = \frac{1}{1 + e^{-y}}$$

- 시그모이드 함수는 아래와 같은 모양을 가지며, 입력값이  $(-\infty \sim \infty)$  범위에 대해 출력은  $0 \sim 1$  사이의 값으로 매핑된다.



- 시험을 본 결과는 합격과 불합격 두 가지 상태 밖에 없지만, 로지스틱 회귀분석을 하여 파라미터  $a$ 와  $b$ 를 구하면 공부시간과 시험 결과의 관계를 S곡선으로 매핑할 수 있게 된다.
- 즉, 로지스틱 회귀분석 모델에서는 아래와 같은 모델을 사용하여 입력 데이터( $x$ : 공부시간)로부터 최적의 합격률( $p$ )를 매핑하는 파라미터  $a$ ,  $b$

값을 학습하는 모델이 로지스틱 회귀분석 모델이다.

$$p = \frac{1}{1 + e^{-(ax+b)}}$$

- 이 모델을 이용하면 어떤 사건이 발생할지 아닐지의 확률을 독립변수  $x$ 로부터 추정할 수 있다.
- 위의 데이터로부터 파라미터  $a$ ,  $b$ 를 구하는 코드는 아래와 같다.

```
model = LogisticRegression()  
model.fit(x.reshape(-1, 1), y.reshape(-1, 1))
```

```
print(model.coef_)

print(model.intercept_)
```

=>

```
[[0.34566022]]
[-2.41671027]
```

- 이제 7시간 공부한 사람이 합격할 확률은 아래와 같이 구할 수 있다.

```
model.predict([7.0])
```

=>

```
array([1])
```

- 클래스 확률을 구하면 아래와 같다.

```
model.predict_proba(7)
```

=>

```
array([[0.49927219, 0.50072781]])
```

```
model.predict_proba(10)
```

=>

```
array([[0.26117085, 0.73882915]])
```

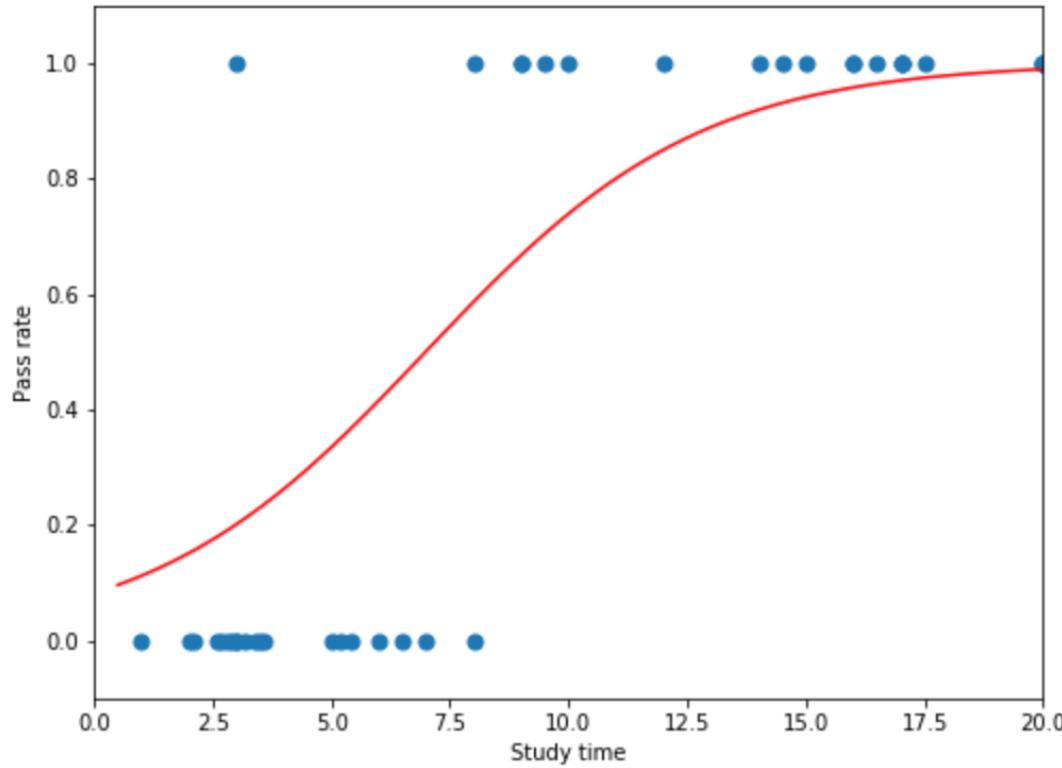
- 위의 로지스틱 모델을 나타내는 시그모이드 그래프를 그리는 코드는 아래와 같다.

```
# 시그모이드 함수 정의
```

```
def logreg(z):  
    return 1 / (1+np.exp(-z))
```

```
fig = plt.figure(figsize=(8,6))
plt.xlim(0, 20)
plt.ylim(-0.1, 1.1)
plt.xlabel('Study time')
plt.ylabel('Pass rate')
plt.scatter(X, y, s=50)

XX = np.linspace(0.5, 21, 100)
yy = logreg(model.coef_*XX + model.intercept_)[0]
plt.plot(XX, yy, c='r')
```



- 로지스틱 회귀는 이름은 회귀이지만 분류 문제이므로 손실함수로 크로스엔트로피를 사용한다.

# 유방암 예제

## 데이터 다운로드

- 데이터를 다음 사이트에서 다운받는다.

<https://goo.gl/u9yHpQ>

```
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import matplotlib.gridspec as gridspec  
import seaborn as sns  
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.cross_validation import KFold    # K-fold cross validation
from sklearn import metrics
```

```
data = pd.read_csv('data/breast_cancer.csv')
```

```
print(data.shape)
```

=>

(569, 33)

- 데이터 샘플은 569개이고 컬럼 수는 33개이다.

```
data.head(5)
```

	<b>id</b>	<b>diagnosis</b>	<b>radius_mean</b>	<b>texture_mean</b>	<b>perimeter_mean</b>	<b>area_mean</b>
<b>0</b>	842302	M	17.99	10.38	122.80	1001.0
<b>1</b>	842517	M	20.57	17.77	132.90	1326.0
<b>2</b>	84300903	M	19.69	21.25	130.00	1203.0
<b>3</b>	84348301	M	11.42	20.38	77.58	386.1
<b>4</b>	84358402	M	20.29	14.34	135.10	1297.0

- 유방암 진단 데이터는 총 30개의 특성을 제공하는데, 여기서는 이중에 두 개의 속성으로 "radius\_mean", "texture\_mean"을 그리고 레이블로 "diagnosis"을 사용하겠다.

```
data['diagnosis'] = data['diagnosis'].map({'M':1,'B':0})
```

```
df = data[["diagnosis", "radius_mean", "texture_mean"]]  
df.head()
```

	<b>diagnosis</b>	<b>radius_mean</b>	<b>texture_mean</b>
<b>0</b>	1	17.99	10.38
<b>1</b>	1	20.57	17.77
<b>2</b>	1	19.69	21.25
<b>3</b>	1	11.42	20.38
<b>4</b>	1	20.29	14.34

- 레이블로 진단 결과 컬럼을 설정하고, 머신러닝 모델로 로지스틱 회귀를

사용하겠다. 모델 변수 이름을 model라고 하였다.

```
model=LogisticRegression()
```

```
y = df["diagnosis"]
```

## **radius\_mean 특성만 사용**

- 속성으로 "radius\_mean" 하나만 사용하여 유방암 진단을 하는 로지스틱 모델을 학습시키고 이를 테스트 데이터에 적용하면 아래와 같은 성능이 나온다.

```
features = ["radius_mean"]
```

```
X = df[features]
```

```
np.random.seed(11)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

```
model.fit(X_train, y_train)  
print("Score: {:.2%}".format(model.score(X_test, y_test)))
```

```
## Score: 85.96%
```

- 계수 a, b를 출력해 보면 다음과 같다.

```
print(model.coef_)  
print(model.intercept_)  
=>  
[[0.48325363]]
```

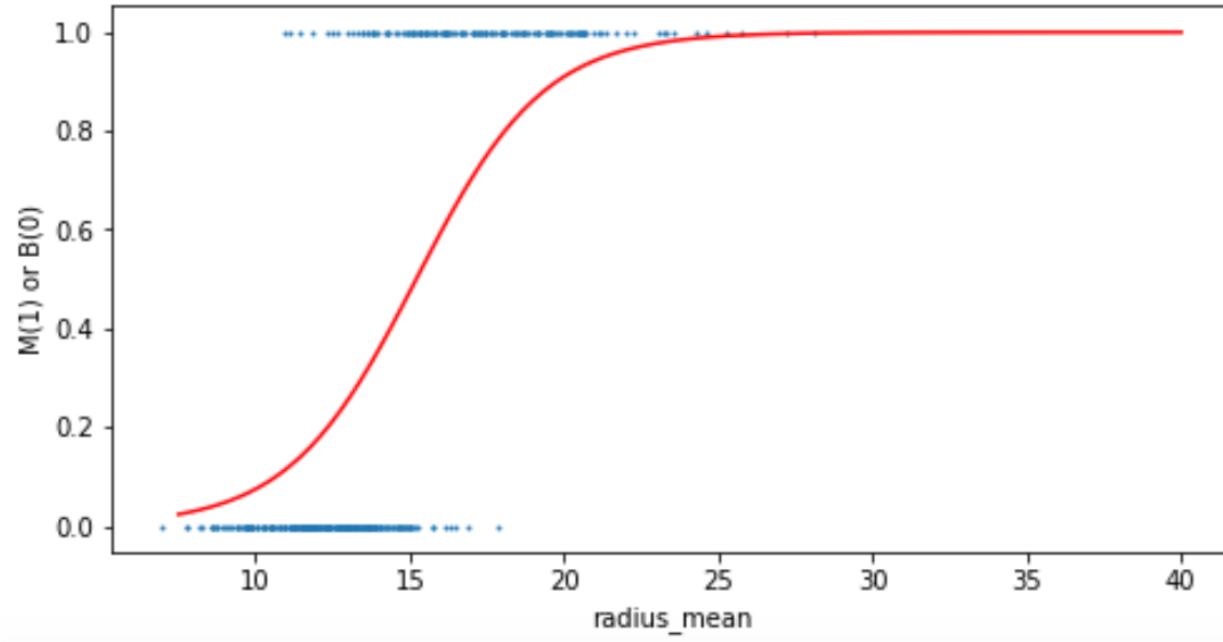
[-7.3485004]

- 이러한 계수 값을 갖는 시그모이드 함수를 그래프로 그리면 다음과 같다.

```
plt.figure(figsize=(8,4))
plt.scatter(X_train, y_train, s=1)
plt.xlabel("radius_mean")
plt.ylabel("M(1) or B(0)")
```

```
def logreg(x):
    return 1 / (1 + np.exp(-x))
```

```
XX = np.linspace(7.5, 40, 100)
plt.plot(XX, logreg(model.coef_ * XX + model.intercept_[0]), c='r')
```



## **texture\_mean 특성만 사용**

- 아래는 속성으로 "texture\_mean" 하나만 사용하는 경우의 성능을 출력했

다.

```
features = ["texture_mean"]
X = df[features]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

```
model.fit(X_train, y_train)
print("Score: {:.2%}".format(model.score(X_test, y_test)))
```

=>

Score: 66.08%

- 계수를 보면 아래과 같다. a의 값이 작은 것을 알 수 있다. a의 값이 작을수록 시그모이드 함수가 옆으로 퍼져서 나타난다.

```
print(model.coef_)

print(model.intercept_)

=>

[[0.1657761]]

[-3.74274468]
```

- 그래프로 그리면 다음과 같다.

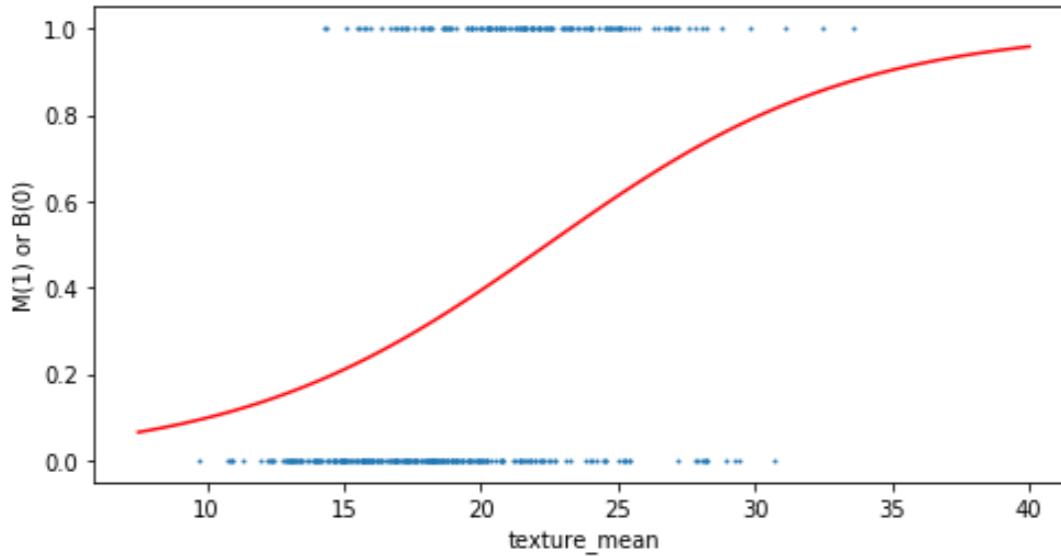
```
plt.scatter(X_train, y_train, s=1)

plt.xlabel("texture_mean")

plt.ylabel("M(1) or B(0)")
```

```
XX = np.linspace(7.5, 40, 100)
```

```
plt.plot(XX, logreg(model.coef_ * XX + model.intercept_[0], c='r')
```



- 위 그래프를 보면 texture\_mean으로는 변별력이 떨어지는 것을 알 수 있다.

## 두 개의 특성 사용

- 이제 두 개의 속성을 모두 사용할 때의 로지스틱 회귀를 구해보겠다.

```
features = ["radius_mean", "texture_mean"]
```

```
X = df[features]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

```
model.fit(X_train, y_train)
```

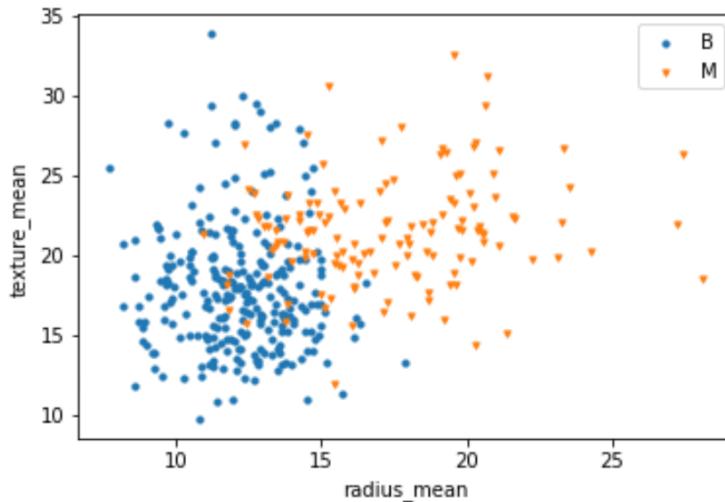
```
print("Score: {:.2%}".format(model.score(X_test, y_test)))
```

```
=>
```

```
Score: 88.89%
```

- 두 개의 속성을 사용하여 성능이 개선된 것을 알 수 있다. 이 두 속성을 2차원 평면에 표시하고 암 진단 결과(레이블)를 그래프로 그리면 다음과 같다.

```
markers = ['o', 'v']
label = ["B", "M"]
for i in range(2):
    xs = X_train["radius_mean"][y_train == i]
    ys = X_train["texture_mean"][y_train == i]
    plt.scatter(xs, ys, marker=markers[i], s=10)
plt.legend(label)
plt.xlabel("radius_mean")
plt.ylabel("texture_mean")
```



## 모든 특성 사용

- 주어진 30개의 모든 속성을 다 사용하여 로지스틱 회귀 분석을 수행해 보겠다. 33개 특성중에 불필요한 3개의 특성을 제거하여 입력  $X$ 를 만든다.

```
data.drop('id',axis=1,inplace=True)  
data.drop('Unnamed: 32',axis=1,inplace=True)  
X=data.drop('diagnosis', axis=1)
```

- 이 X를 사용하여 성능을 측정하면 아래와 같다. 모델 이름을 model\_all로 정했다.

```
np.random.seed(11)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)  
model_all = LogisticRegression()  
model_all.fit(X_train, y_train)  
print("Score: {:.2%}".format(model_all.score(X_test, y_test)))  
=>  
Score: 93.57%
```

## kNN과 성능 비교

- 같은 데이터를 사용하여 kNN으로 수행하면 아래와 같은 성능을 얻는다.

```
for i in range(1,21,2):
```

```
    knn = KNeighborsClassifier(n_neighbors=i)
```

```
    knn.fit(X_train, y_train)
```

```
    print("K = ", i, "-> Score: {:.2%}".format(knn.score(X_test, y_test)))
```

```
=>
```

```
K = 1 -> Score: 90.06%
```

```
K = 3 -> Score: 91.81%
```

```
K = 5 -> Score: 93.57%
```

```
K = 7 -> Score: 92.98%
```

```
K = 9 -> Score: 92.98%
```

$K = 11 \rightarrow \text{Score: } 92.40\%$

$K = 13 \rightarrow \text{Score: } 92.40\%$

$K = 15 \rightarrow \text{Score: } 91.81\%$

$K = 17 \rightarrow \text{Score: } 91.81\%$

$K = 19 \rightarrow \text{Score: } 91.81\%$

- $k$  가 5일 때 성능이 가장 좋은 것으로 나타났다.

## 결정트리와 랜덤포레스트 비교

```
from sklearn.tree import DecisionTreeClassifier  
tree = DecisionTreeClassifier()  
tree.fit(X_train, y_train)  
print("결정트리 -> {:.2%}".format(tree.score(X_test, y_test)))  
=>  
결정트리 -> 92.98%
```

```
from sklearn.ensemble import RandomForestClassifier  
rfc = RandomForestClassifier(n_estimators=300)  
rfc.fit(X_train, y_train)
```

```
print("랜덤포레스트 -> : {:.2%}".format(rfc.score(X_test, y_test)))
```

=>

랜덤포레스트 : 97.66%

▪

- 랜덤포레스트를 사용할 때 성능이 가장 좋은 것을 알 수 있다.

## 다항 로지스틱 회귀

- 앞에서는 이진 분류, 즉 합격/불합격 등 두 개의 레이블을 가진 경우에 로지스틱 회귀를 사용하는 예를 소개했다. 그런데 2개가 아니라 3개 이상의 클래스 중에 하나를 예측해야 하는 경우는 로지스틱 회귀를 그대로 사용할 수 없다.
- 다항 분류를 처리하려면 다항 로지스틱 회귀(multinomial logistic regression) 또는 소프트맥스 (softmax) 함수를 이용한다.
- 다항 로지스틱스에서는 우선 각 클래스로 분류될 가능성을 나타내는 점수를 구하고 이 점수들을 사용하여 상대적인 확률을 구하는 소프트맥스 함수를 적용한다.

- 소프트맥스 함수의 정의는 아래와 같다. 여기서  $\hat{p}_k$ 는 클래스  $k$ 에 속할 확률이고,  $x$ 는 주어진 샘플을 의미하며,  $s_k(x)$ 는 소프트맥스 회귀모델이 각 클래스  $k$ 에 대한 점수이다.

$$\hat{p}_k = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_{j=i}^K \exp(s_j(x))}$$

- 예를 들어 사람의 얼굴을 보고 한국인, 중국인, 일본인 등 세가지 국적 카테고리를 구분하는 문제가 있다고 하자. 학습을 통하여 각각 한, 중, 일 국적일 점수가 1.5, 2.0, 1.8 이었다고 하자. 이 점수만 보면 이 사람은 중국인일 점수가 가장 높다.

- 그러나 이를 확률처럼 상대적으로 나타내고 싶다면 각 점수의 합이 0~1 사이가 되면서 세가지 점수의 합이 1이 되도록 변환하는 것이 필요하다. 이를 위해 사용하기 적당한 함수가 소프트맥스이다.
- 위의 이 점수를 소프트맥스 함수로 바꾸면 0.23, 0.43, 0.34이 되며 이를 각각 한, 중, 일 국적일 “확률”로 사용한다.

## 7.7. 모델 성능 평가

### 분류 평가 지표

- 학습으로 만든 분류 모델의 성능이 얼마나 우수한지를 알기 위해서 위에서 앞의 예제에서는 `score()` 함수를 이용하여 정확도(accuracy)를 출력해 보았다. 그런데 정확도만 측정해서는 분류의 성능을 제대로 평가하기 어려울 때가 있다.
- 예를 들어 발생할 확률이 0.1%인 어떤 병을 검진하는 예를 생각해 보자. 예측 모델을 간단하게 만들어서, “모두 정상”이라고 단순하게 판정하는 모델을 사용해도 정확도만 보면 맞출 확률이 99.9%가 된다(평균적으로 99.9%는 정상이므로).

- 이렇게 발생 빈도가 심하게 비 대칭적인 경우는 정확도만 봐서는 성능을 파악할 수 없고 가끔 발생하는 병을 얼마나 잘 찾아내는 지도 확인해 봐야 한다.
- 즉, 정확도 하나만 측정해서는 이러한 모델의 성능을 정확히 표현하기 어렵다. 만일 “모두 정상”이라고 하는 단순한 모델을 사용한다면 정확도는 99.9%이지만 실제로 병을 찾아낼 확률은 0%가 된다.
- 이러한 잘못된 평가를 피하기 위해서 분류에서는 정확도 뿐 아니라 정밀도 (precision), 재현율(recall), F1점수(F1-score) 등의 지표를 사용한다.
- 파이썬의 sklearn패키지에서 이들 값을 출력해주는 함수가 classification\_report()이다. 아래는 앞에서 소개한 스팸 메일 분류기 모델

의 성능을 출력해주는 코드는 아래와 같다.

```
y_predict = clf.predict(X_test)  
print(metrics.classification_report(y_test, y_predict))  
=>  


|             | precision | recall | f1-score | support |
|-------------|-----------|--------|----------|---------|
| ham         | 0.99      | 0.80   | 0.88     | 1227    |
| spam        | 0.39      | 0.94   | 0.55     | 167     |
| avg / total | 0.92      | 0.82   | 0.84     | 1394    |


```

- 위의 출력을 보면 정상 메일을 찾는 것을 기준으로 정밀도, 재현률, F1점수가 각각 0.99, 0.80, 0.88인 것을 나타낸다. 스팸 메일을 찾는 것을 기준으로 정밀도, 재현률, F1점수는 각각 0.39, 0.94, 0.55이다.
- 정상 메시지와 스팸 메시지의 개수를 support로 표시한다. support를 고려한 가중 평균 값을 아래에 나타냈다.
- 이 숫자들의 의미를 정확히 이해하려면 먼저 혼돈 매트릭스(confusion matrix)를 이해해야 한다.

## 혼돈 매트릭스

- 혼돈 매트릭스(confusion matrix)란 분류의 결과가 잘 맞았는지를 평가하는 채점표와 같다. 이진 분류 즉, 두 가지 카테고리를 분류하는 경우 예측한 내용과 실제 타겟 값이 맞는지를 따져보면 총 네 가지 경우의 수가 존재한다.
- 정상 메시지인 사건을 positive라고 표현하고, 정상이 아닌 경우를 negative라고 정의하겠다. 그리고 예측 결과가 맞으면 True, 틀리면 False라고 수식어를 붙이면 아래와 같이 네 가지 경우의 수가 발생하며 각각 True positive(TP) 등의 이름을 사용한다.

정상이라고 예측했는데 실제로 정상인 경우의 수 , True positive (TP)  
정상이라고 예측했는데 실제는 스팸인 경우의 수, False positive (FP)

스팸이라고 예측했는데 실제는 정상인 경우의 수, False negative (FN)  
스팸이라고 예측했는데 실제로도 스팸인 경우의 수, True negative (TN)

- 이 용어들을 보면 처음에는 좀 혼란스럽다. 이 용어들을 쉽게 이해하는 방법을 소개하겠다. 첫 번째 단어인 True나 False는 예측의 결과를 나타낸다. 즉, 앞의 단어가 True인 것은 결과를 “맞춘 것”이고, 앞에 False를 붙인 것은 예측이 틀린 경우를 나타낸다.
- 뒤의 단어는 예측한 대상을 구분한다. “positive”이면 찾고자 하는 카테고리를 나타낸다. 위의 예에서는 정상을 찾는 작업을 positive라고 정의했다. “negative”이면 스팸을 예측하는 경우를 나타낸다.
- 이 내용을 정리하면, 뒤의 단어는 예측, 앞의 단어는 결과를 구분한다.

예를 들어 False positive는 정상이라고 예측했는데 (positive) 결과는 틀린(False) 경우의 수를 말한다.

- 위에 설명한 혼돈 매트릭스 정의를 표로 그리면 아래와 같다.

혼돈 매트릭스 (열은 예측을, 행은 결과를 나타낸다)

실제 \ 예측	p로 예측	n으로 예측
실제로는 p	True positive (TP)	False negative (FN)
실제로는 n	False positive (FP)	True negative (TN)

- sklearn이 제공하는 confusion\_matrix() 함수를 사용하면 혼돈 매트릭스를 출력할 수 있는데, 아래에 해당 코드를 보였다.

```
print(metrics.confusion_matrix(y_test, y_predict))
```

=>

[[981 246]

[ 10 157]]

- 이 출력 결과는 앞에서 설명한 혼돈 매트릭스와 같으면 각 행과 열의 의미는 아래와 같다.

### 혼돈 매트릭스

실제 \ 예측	정상(ham)로 예측	스팸으로 예측
실제로 정상	True positive (TP) = 981	False negative (FN) = 246
실제로 스팸	False positive (FP)= 10	True negative (TN)= 157

## 정확도(accuracy)

- 정확도란 positive 및 negative 두 가지에 대해서 정확히 예측한 비율을 말한다. 정확도는 혼돈 매트릭스에서 True-에 해당하는 부분 즉, 정확하게 예측한 비율을 말한다. 정확도의 정의는 아래와 같다.

$$\begin{aligned}\text{정확도} &= (\text{positive 또는 negative를 맞춘 수}) / (\text{전체 샘플 수 } N) \\ &= (TP+TF) / (TP + FP + FN + TN)\end{aligned}$$

- 위에서 전체 샘플의 수( $N$ )는 혼돈 매트릭스 네 가지 경우의 수를 모두 더한 값이다.  $N = TP + FP + FN + TN$ . 앞의 예제에서는 전체 테스트 샘플 수는 1394개이며 정확히 맞춘 수는  $981+157 = 1138$  개이므로 정확도는  $27/1394 = 0.81640$ 이다.

## 정밀도(precision)

- 정밀도란 머신러닝 모델이 positive라고 예측한 수에 실제로 positive인 비율을 말한다. 정밀도의 정의는 아래와 같다.

$$\begin{aligned}\text{정밀도} &= (\text{positive를 맞춘 수}) / (\text{positive라고 예측한 수}) \\ &= \text{TP}/(\text{TP+FP})\end{aligned}$$

- 앞의 예에서는 정밀도는  $981/(981+10) = 0.99$ 가 된다. 정밀도의 의미는 모델이 positive라고 주장한 중에 실제로 맞춘 경우의 비율을 말한다.

## 재현률(recall)

- 재현률은 테스트 데이터 중 전체 positive (즉, 세토사) 중에 실제로 모델이 positive라고 찾아낸 비율을 말한다. 재현률은 관심 있는 대상을 얼마나 찾아내는지를 나타낸다. 재현율의 정의는 아래와 같다.

$$\begin{aligned}\text{재현률} &= (\text{positive를 맞춘 수}) / (\text{positive 전체 수}) \\ &= \text{TP} / (\text{TP+FN})\end{aligned}$$

- 앞의 예에서 재현율은  $981/(981+246) = 0.800$ 이었다.

## F1 지표

- 앞에서 세 가지 성능 지표를 설명했는데, 정상 메시지(타겟 변수 “0”) 클래스에 대한 정확도, 정밀도, 재현률이 각각 0.82. 0.99, 0.80 이었다. 이들 각각의 의미가 다른데 지표를 어떻게 해석해야 할까?
- 재현률과 정밀도 두 성능평가 지표를 동시에 높이려면 실력이 좋아야 한다. 또는 머신러닝 모델을 잘 만들어야 한다. 이 두 지표를 동시에 높이는 머신러닝 모델이 좋은 모델이라고 할 수 있다.
- 이를 위해서 이 두 지표를 함께 반영한 새로운 지표가 필요하게 되었는데 이를 위해서 F1 점수를 도입하였다.
- F1 점수는 정밀도와 재현률의 조화평균(harmonic average)이다.

$$F1 = 2 \times \text{precision} \times \text{recall} / (\text{precision} + \text{recall})$$

- 어떤 두 숫자  $a$ ,  $b$ 의 산술평균  $c$ 는 아래와 같이 구한다.

산술평균:  $c = (a+b)/2$

- 예를 들어, 어떤 학생 A의 영어와 국어 성적이 각각 30점 90점이면 A 학생의 산술평균 점수는 60점이다  $(30+90)/2 = 60$ .
- 한편 두 숫자의 조화 평균은 각 수의 역수의 산술 평균을 구하고 이의 역수를 구하는 것을 말한다. 즉,  $a$ 와  $b$ 의 조화 평균  $c$ 는 아래와 같이 구한다.

기하평균:  $\frac{1}{c} = \frac{\left(\frac{1}{a} + \frac{1}{b}\right)}{2}$

$$c = \frac{2ab}{a+b}$$

- 학생 A의 성적의 조화 평균은 계산해 보면 45점이 된다. 한편 다른 학생 B의 영어와 국어 성적이 각각 50점 70점이었다고 하자.
- B 학생의 두 과목 산술평균과 조화 평균은 각각, 60점, 58점이 된다. 산술평균만 보면 두 학생의 실력은 모두 60점으로 같다.

- 그러나 조화 평균을 계산해 보면 45점과 58점으로 다르게 나타난다.
- 조화평균은 두 평가 지표(위에서 영어와 국어 점수)를 동시에 고려하되 낮은 점수의 영향을 더 많이 반영하는 (즉, 낮은 점수에 페널티를 더 주는) 것이 필요할 때 자주 사용되는 방식이다.
- F1 점수는 같은 목적으로 도입되었는데, 정밀도와 재현률의 조화평균을 사용하여 분류의 성능을 종합적으로 측정하기 위한 지표이다.
- 이제 정밀도나 재현률을 각각 보는 것 외에 두 값을 반영하여 계산된 F1점수를 보면 한번에 성능을 평가할 수 있게 되었다.
- 한편 재현률을 민감도(senditivity)라고 부르기도 한다.

- 앞에서 소개한 성능 평가 지표를 정리하면 다음과 같다.

정확도 accuracy - 질병 유, 무를 맞춘 비율

재현률 recall - 질병이 있는 사람중에 질병이 있다고 정확히 판단한 비율

정밀도 precision - 질병이 있다고 진단한 중에 실제 질병이 있는 비율

특이도 specificity - 질병이 없는 사람들을 대상으로 정확히 질병이 없다고 판단하는 비율 (정밀도와 같은 개념)

F1 score - 정밀도와 재현률(민감도)의 조화 평균값

ROC - 민감도와 특이도를 그래프로 그린 것

## 비용 최소화

- 머신러닝 모델을 사용하여 최종적으로 얻고자 하는 목표치를 정해야 하고 모델을 궁극적으로 이를 달성해야 한다. 오차를 줄이는 것 자체가 목표가 아니라 예를 들면 수익 증대, 사망률 감소, 교통사고 감소량 등을 말한다.
- 즉, 오차나, 정확도를 높이는 것은 중간과정일 뿐이며 이것이 최종 목표치를 향상시킨다는 보장은 없다.
- 분류의 결과가 어떤 이득(benefit)과 비용(cost)을 발생하는지를 따져보고 의사결정을 돋는 것이 데이터 분석의 목적이다. 즉, 분류 확률을 높이는 것 자체가 중요한 것이 아니라 분류 결과의 기대치를 구해봐야 한다.
- 암 진단의 경우 분류 결과에 따라 발생할 수 있는 이득 또는 비용은 다음과

같이 생각해볼 수 있다

- 1) 암이라고 예측했는데 실제로 암에 걸린 경우 - 환자는 치료를 계속 받고 병원은 수익을 낸다
  - 2) 암이라고 예측했는데 실제는 암에 걸리지 않은 경우 - 환자는 정밀검사를 받겠지만 암이 아닌 것에 안도한다
  - 3) 암이 아니라고 예측했는데 실제는 암인 경우 - 환자가 항의를 할 것이고 손해배상을 청구할 수도 있다
  - 4) 암이 아니라고 예측했는데 실제로도 암이 아닌 경우 - 환자는 진료가 정확하다고 믿고 다음에도 이 병원을 찾는다
- 
- 위 1)의 경우 수익이 발생하며 이를 편의상 200만원이라고 하자(E1), 2)의 경우 병원은 약간의 손실이 생기며 이를 -3만원이라고 하자(E2), 2)의

경우는 병원에 큰 손실이 생기며 이를 -500만원이라고 하자(E3), 4)의 경우 환자를 더 유치하므로 평균 이득이 2만원이라고 하자(E4) (이 내용은 임의로 정한 것이며 병원 정책에 따라 달라지는 값이다).

## 동적 성능 평가

- 앞에서 설명한 컴퓨터 매트릭스를 통해서 분류의 정확도(accuracy), 재현률, 정밀도(precision) 등을 구할 수 있다.
- 알고리즘의 동작을 좀 더 세밀하게 평가하기 위해서는 최종 분류 결과만 보는 것이 아니라 분류한 순서를 평가하는 방법이 필요하다.

## ROC 곡선

- ROC(receiver operating characteristics) 곡선은 분류 모델의 성능을 분류 예측한 순서까지 고려하여 자세히 알아볼 수 있는 유용한 수단이다.
- 예측 결과를 순서로 제시한 것이 실제 값과 얼마나 순서에 따라 잘 맞는지는 검증하기 위해서 2차원 그래프로 그린 것이다.
- ROC 커브는 예측한 순으로 한 행씩 내려오면서 정답과 오류를 차례대로 확인하는 방식으로 그린다. ROC 커브는 (0,0) 점에서 시작하는데, 한 행씩 체크를 하여 정답을 맞추었으면 y축 위로 한 칸 이동하고, 정답을 맞추지 못했으면 x축 방향으로 한 칸 이동한다.
- 여기서 x축은 false positive rate를 나타내며, y축은 true positive rate를

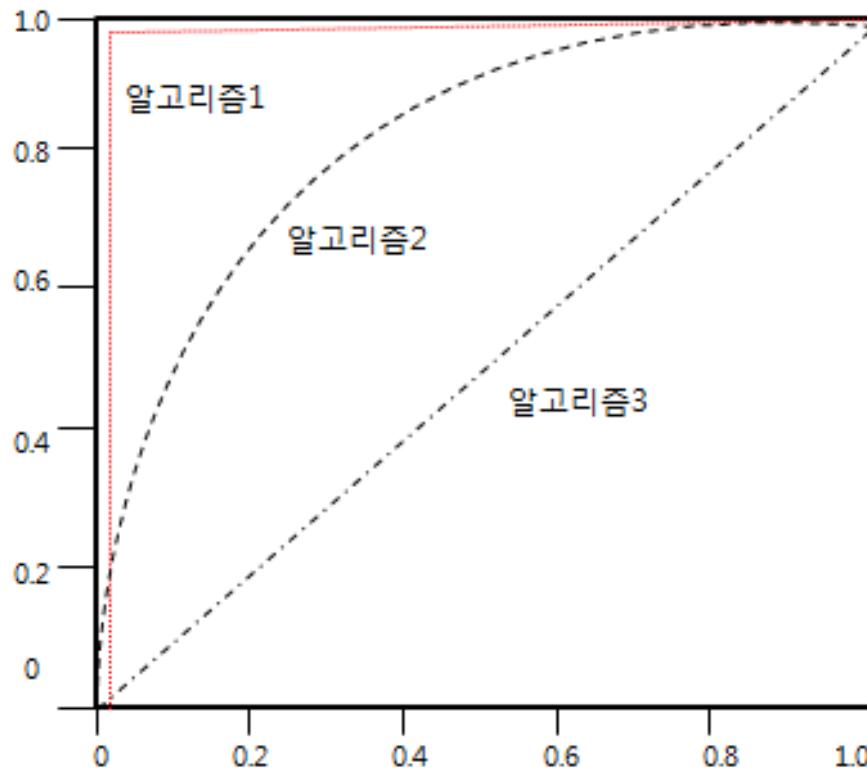
나타낸다.

## AUC(Area Under the Curve)

- ROC 커브를 그려보면 분류 전체의 성능을 한 눈에 알 수 있다. 그래프가 초기에 위 방향으로 올라갈수록 분류를 잘 한 것이다.
- 그러나 ROC를 이용하여 알고리즘을 평가하려면 항상 2차원 그래프를 그려 봐야 하므로 평가를 수치화하기에는 불편한 점이 있다.
- 예측 알고리즘의 성능을 간단히 수치로 나타내기 위해서 ROC 그래프의 면적을 계산하는 방법을 사용한다. 이 면적의 크기를 AUC(Area Under Curve)라고 한다.
- 우수한 알고리즘일수록 초반에 y축 윗 방향으로 이동하므로 ROC 커브의 면적이 넓어질 것이다. 이상적으로 분류를 잘 예측한 경우 (위 그림에서

BEST의 경우) AUC는 최대값으로서 1이 된다 (정사각형 그래프의 면적은 1이므로).

- 아래 그림에서 대각선 방향의 점선은 기본적인 예측 모델을 나타내는데 이는 분류 모델이 아무런 알고리즘도 적용하지 않고 랜덤하게 예측한 경우를 나타낸다. 이 때 AUC는 0.5를 갖는다.
- 컴퓨터 매트릭스는 순서의 어디까지를 양성이라고 판정할지 정하는 것에 따라 다르게 나타나다. 예를 들어 최종적으로 암이라고 판단한 순서를 어디까지로 정할지 기준값을 정하는 데 따라 컴퓨터 매트릭스가 달라진다.



알고리즘 성능에 따라 AUC의 값이 달라지는 예

## ROC 예제

- 남녀 각각 10명씩 있는 20명을 상대로 남녀를 예측하는 예를 들어보겠다.

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc  
from sklearn.preprocessing import StandardScaler, LabelEncoder  
%matplotlib inline
```

- 여성이라고 판단한 평가 점수(score)의 순서가 다음과 같다고 가정하겠다.

```
y_score = np.linspace(99, 60, 20).round(1)  
print(y_score)
```

=>

```
[99. 96.9 94.9 92.8 90.8 88.7 86.7 84.6 82.6 80.5 78.5 76.4 74.4 72.3  
70.3 68.2 66.2 64.1 62.1 60. ]
```

- 이 점수를 근거로 상위 8명을 여성이라고 분류 예측했다고 하자.

```
y_pred=[1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0]
```

```
len(y_pred)
```

```
y_pred.count(1), y_pred.count(0)
```

=>

```
(8, 12)
```

- 위의 점수 순으로 배치하여 확인한 실제 값(여성과 남성의) 분포가 아래와

같다고 하자.

```
y_test=[1,1,0,1,0,1,1,1,0,0,1,0,1,1,0,1,0,0,0]
```

```
y_test.count(1), y_test.count(0)
```

=>

```
(10, 10)
```

- 이 데이터를 바탕으로 혼돈 매트릭스를 구하면 아래와 같다.

```
confusion_matrix(y_test, y_pred)
```

=>

```
array([[8, 2],
```

```
       [4, 6]])
```

- 정밀도, 리콜, f1점수를 구하면 아래와 같다.

```
print(classification_report(y_test, y_pred))
```

=>

	precision	recall	f1-score	support
0	0.67	0.80	0.73	10
1	0.75	0.60	0.67	10
avg / total	0.71	0.70	0.70	20

- 이제 분류 예측한 순서를 평가해보겠다. 먼저 점수, 예측값, 실제값을 하나의 데이터프레임으로 만들어보겠다.

```

result = pd.DataFrame(list(zip(y_score, y_pred, y_test)),
                      columns=['score', 'predict', 'real'])

result['correct'] = (result.predict == result.real)

result.head(10)

```

	score	predict	real	correct
0	99.0	1	1	True
1	96.9	1	1	True
2	94.9	1	0	False
3	92.8	1	1	True
4	90.8	1	0	False
5	88.7	1	1	True
6	86.7	1	1	True
7	84.6	1	1	True
8	82.6	0	0	True
9	80.5	0	0	True

- 이제 점수를 고려하여 ROC 커브와 AUC를 구하면 아래와 같다.

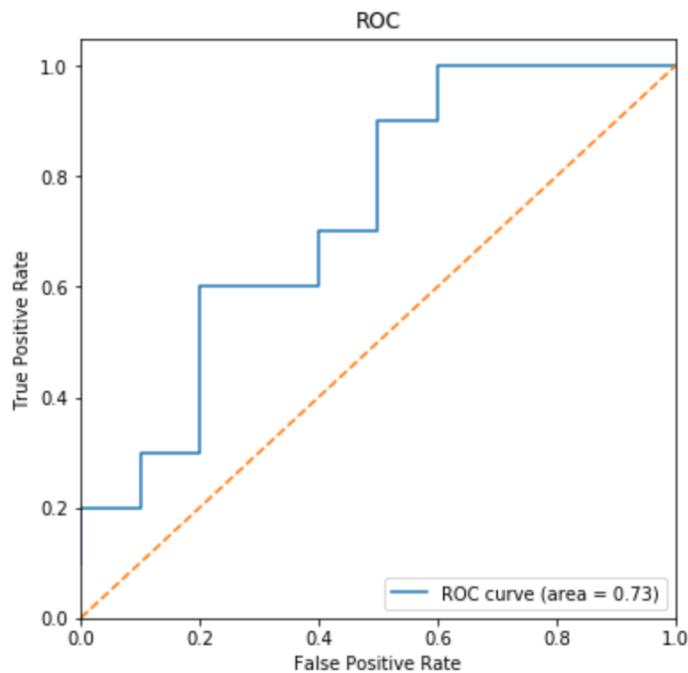
```
fpr = dict()
tpr = dict()
roc_auc = dict()

fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('ROC')
```

```
plt.legend(loc="lower right")
```



## 다중 분류

- 분류 알고리즘은 기본적으로 이진 분류를 수행한다. 즉, 두 개의 카테고리를 나누는 기능을 수행하다. 여러개의 카테고리를 가질 수 있는 입력 데이터를 분류하는 것을 다중분류하고 한다.
- 사이킷 런의 이진 분류 함수들을 다중 분류를 지원하는데 이는 내부적으로 이진 분류를 확장해서 수행한다. 카테고리를 하나씩 선택하고 이 카테고리인 것과 아닌 것 등 두 가지로 나누는 이진 분류를 수행하면 첫 번째 카테고리를 구분할 수 있다.
- 이러한 방식을 One-versus-Rest (OvR) 방법이라고 한다. 즉, 이진 분류 알고리즘을 여러번 적용하면 다중 분류를 수행할 수 있다.

- 다중 분류에서 주어진 샘플이 최종적으로 어느 클래스에 속하는지 알려면, 즉, 분류 결과만 알려면 predict()를 사용하면 된다. 그러나 다중 클래스 각각에 해당할 확률을 알려면 decision\_function() 함수 또는 predict\_proba()를 사용해야 한다 (모델에 따라 지원하는 함수명이 다르다)

cross\_val\_predict()

predict() # 해당 클래스를 예측한다

decision\_function() # 각 클래스에 해당하는 점수를 알려준다

predict\_proba() # 각 클래스에 해당하는 확률을 알려준다

## 7.8. 분류 알고리즘 성능 비교

- 포도주 품질을 예측하는 문제를 다양한 알고리즘으로 비교하겠다. 혼돈 매트릭스와 ROC 커브를 그려보겠다. 포도주의 특성은 다음과 같다. 레이블은 0~10점의 범위를 갖는 quality 값을 예측하는 것이다.
  - fixed acidity - 결합 산도
  - volatile acidity - 휘발성 산도
  - citric acid - 시트르산
  - residual sugar - 잔류 설탕
  - chlorides - 염화물
  - free sulfur dioxide - 자유 이산화황
  - total sulfur dioxide - 총 이산화황

- density - 밀도
- pH - pH
- sulphates - 황산염
- alcohol - 알코올
- quality - 품질 (0 ~ 10 점)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
```

```

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
%matplotlib inline
wine = pd.read_csv('data/winequality-red.csv')
print(wine.shape)
wine.head(5)
=>
(1599, 12)

```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51

```
wine['quality'].value_counts()
```

=>

5 681

6 638

7 199

4 53

8 18

3 10

- 이진 분류 문제로 바꾸기 위해서 포도주의 품질이 좋고 나쁜 것을 나누는 기준 설정을 7, 8 점을 상품, 6점 이하를 하품으로 나누겠다. 이를 위해서 6.5를 기준으로 bad(0) good(1)으로 나눈다. 이것은 임의로 나눈 것이다.

```
my_bins = (2.5, 6.5, 8.5)
groups = [0, 1]
wine['qual'] = pd.cut(wine['quality'], bins = my_bins, labels = groups)
```

```
wine['qual'].value_counts()
```

```
X = wine.drop(['quality', 'qual'], axis = 1)
y = wine['qual']
```

```
y.value_counts()
```

```
=>
```

0	1382
1	217

- 표준 스케일링을 한 후 다양한 알고리즘 적용시의 점수(정확도)를 비교하면 아래와 같다.

```
sc = StandardScaler()
```

```
X = sc.fit_transform(X)
```

```
np.random.seed(11)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

```
# 선형 분류
```

```
sgd = SGDClassifier()
```

```
sgd.fit(X_train, y_train)
```

```
sgd.score(X_test,y_test)
```

```
=> 0.83125
```

```
# 결정 트리
```

```
from sklearn import tree
```

```
clf = tree.DecisionTreeClassifier()
```

```
clf = clf.fit(X_train, y_train)
```

```
clf.score(X_test,y_test)
```

```
=> 0.871875
```

```
# 랜덤 포레스트
```

```
rfc = RandomForestClassifier(n_estimators=300)
```

```
rfc.fit(X_train, y_train)
```

```
rfc.score(X_test,y_test)
```

```
=> .91875
```

```
# SVC  
svc = SVC()  
svc.fit(X_train, y_train)  
svc.score(X_test,y_test)  
=> 0.878125
```

```
# 로지스틱 회귀  
log = LogisticRegression()  
log.fit(X_train, y_train)  
log.score(X_test,y_test)  
=> 0.871875
```

## 혼돈 매트릭스

```
y_pred = sgd.predict(X_test)  
confusion_matrix(y_test, y_pred)  
=>  
array([[235,  34],  
       [ 20,  31]])
```

## ROC 커브

- 먼저 분류에 사용된 내부 확률 값을 decision\_function로 구한다. 이를 이용하여 ROC 커브를 아래와 같이 그릴 수 있다.

```
y_score = sgd.decision_function(X_test)
```

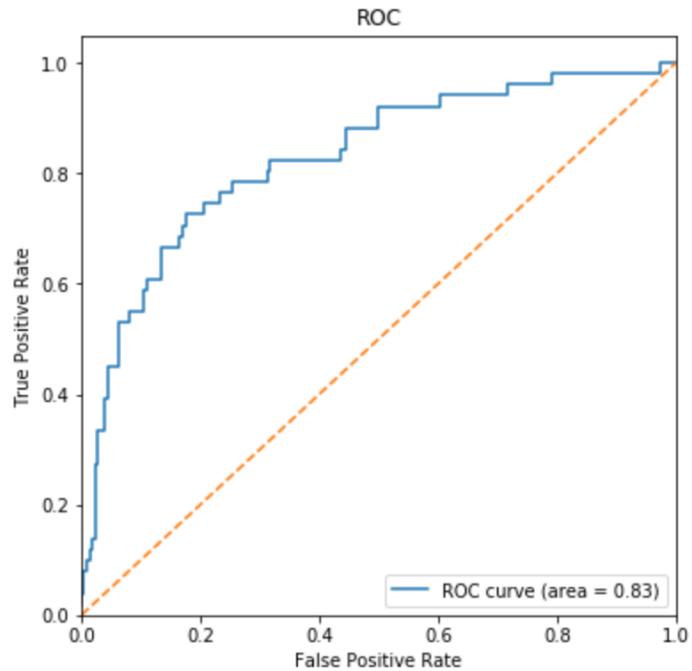
```
result = pd.DataFrame(list(zip(y_score, y_pred, y_test)),  
                      columns=['score', 'predict', 'real'])  
  
result['correct'] = (result.predict == result.real)  
  
result.head()
```

	score	predict	real	correct
<b>0</b>	-6.181080	0	0	True
<b>1</b>	-24.022043	0	0	True
<b>2</b>	-20.107734	0	0	True
<b>3</b>	-14.720284	0	0	True
<b>4</b>	-47.366560	0	0	True

```
fpr = dict()  
tpr = dict()  
roc_auc = dict()  
  
fpr, tpr, _ = roc_curve(y_test, y_score)
```

```
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")
```



## 7.9. 특성공학

- 머신러닝에서는 기존의 여러 특성 중에 분석에 가장 영향력이 있는 특성을 선택하는 것이 필요할 때가 있다.
- 예를 들어 학생의 학업 능력을 평가하는데 모든 과목의 성적을 다 사용하지 않고 국어와 수학 성적이 대표성이 있다면 국어와 수학 두 과목의 성적만 평가 점수로 사용할 수 있다.
- 이와 같이 머신러닝에서 사용할 특성을 잘 선택하는 것을 특성 공학이라고 한다. 유효한 특성을 잘 선택하면 학습 속도도 빨라지고 성능(정확도 등)도 좋아진다.

## 차원축소

- 차원축소란 머신러닝의 성능을 떨어뜨리지 않으면서 특성의 수를 줄이는 기술을 말한다.
- 차원축소가 필요한 이유는 계산량과 메모리 사용량을 줄이기 위해서이다. 또한 샘플의 특징을 보기 좋게 시각화하기 위해서도 차원축소가 필요하다.

## selectPercentile()

- 어떤 속성을 선택하는 것이 좋을지를 컴퓨터가 자동으로 찾아 주는 방법으로 가장 널리 사용되는 방법은 목적 변수와 상관관계가 높은 변수들을 찾는 것이다.
- 상관관계가 높은 순서대로 속성들을 나열하고 상위 몇 %까지의 특성을 선택해 주는 함수인 selectPercentile()을 사용하면 분석에 가장 효과적인 속성을 찾을 수 있다.
- 머신러닝에 별로 도움이 되지 않는 특성들을 제거함으로써 모델 개발 시간을 줄일 수 있고 남는 시간에 다양한 시도를 해 볼 수 있다.
- 아래는 영향력이 큰 특성을 상위 20%까지 찾아주는 코드이다. 유방암

데이터를 사용하였다.

- 먼저 주어진 30개 특성을 모두 사용하는 경우와 상위 20%의 유용한 특성만 (즉, 6개) 사용할 때의 성능을 비교하겠다.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import feature_selection
from sklearn.feature_selection import SelectPercentile
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
from sklearn.manifold import TSNE
from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
np.random.seed(11)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

X\_train.shape

=>

(398, 30)

- 30개의 특성을 모두 사용하고 랜덤 포레스트를 사용한 경우의 분류 성능은 다음과 같다.

```
rfc = RandomForestClassifier(n_estimators=300)
```

```
rfc.fit(X_train, y_train)
```

```
rfc.score(X_test,y_test).round(4)
```

```
=>
```

```
0.9649
```

- 상위 20%의 유효한 특성 즉 6개의 특성만 선택하여 분류를 수행하는 코드는 아래의 코드와 같이 SelectPercentile 함수를 사용한다.
- 첫 번째 인자에 데이터를 검정하여 나눌 기준인 카이제곱(chi2)을, 두

번째 인자에 퍼센트를 입력하면 전체 데이터의 상위 20% 만을 추출한다.

- 선택된 컬럼의 이름을 알 수 없으므로 fs.get\_support를 사용하여 선택된 열을 불리언 형식으로 표현하고(True, False, ...) 이를 사용하여 선택된 컬럼의 이름을 얻고 데이터 프레임으로 변환하여 출력했다.

```
# 특성의 이름을 컬럼 명으로 지정
```

```
X_train = pd.DataFrame(X_train, columns=data.feature_names)
```

```
# 상위 20%의 유효한 특성만 선택 (6개))
```

```
fs = SelectPercentile(feature_selection.chi2, percentile = 20)
```

```
X_train_P = fs.fit_transform(X_train, y_train)
```

```
X_train.columns[fs.get_support()]
```

```
=>
```

```
ndex(['mean perimeter', 'mean area', 'area error', 'worst radius',
      'worst perimeter', 'worst area'],
      dtype='object')
```

```
columns_new=X_train.columns[fs.get_support()]
X_test_P = fs.transform(X_test)
```

```
rfc_P = RandomForestClassifier(n_estimators=300)
```

```
rfc_P.fit(X_train_P, y_train)
```

```
rfc_P.score(X_test_P,y_test).round(4)
```

```
=>
```

```
0.9357
```

- 20%의 특성만 사용해도 성능이 별로 떨어지지 않은 것을 알 수 있다.

심지어 6%만 선택하여 특성을 두 개만 선택해도 성능이 0.924가 되는 것을 확인할 수 있다. 30개에서 단 2개만 사용해도 성능이 크게 떨어지지 않았다.

- 특성 선택은 기본적으로 카이제곱 검증을 사용하여 가장 관련성이 높은 속성을 찾아준다. 카이제곱 검증은 상관관계를 계산하여 우연히 어떤 현상이 발생한 것인지 아니면 충분히 연관성이 있는지를 알려주는 방법이다.

## 주성분 분석

- 주성분 분석(principal component analysis: PCA)이란 여러 속성들을 조합하여 이들을 대표할 수 있는 적은 수의 특성을(이를 주성분이라고 한다)를 찾아내는 작업을 말한다. 이다.
- 주성분을 도출하는데 가장 많이 사용하는 방법은 기존의 속성들의 선형 조합으로 새로운 변수를 정의하는 방식을 많이 사용한다.
- 예를 들면 국어, 영어, 수학 성적에 각각 가중치를 0.4, 0.3, 0.3을 곱한 값이 수업 능력이고, 독서량, 운동량, 친구 수에 각각 0.5, 0.2, 0.3을 곱하여 얻은 수를 활동지수라고 정의하면 학생의 능력을 수업능력과 활동 지수 두 개의 주성분으로 나타내는 셈이다.

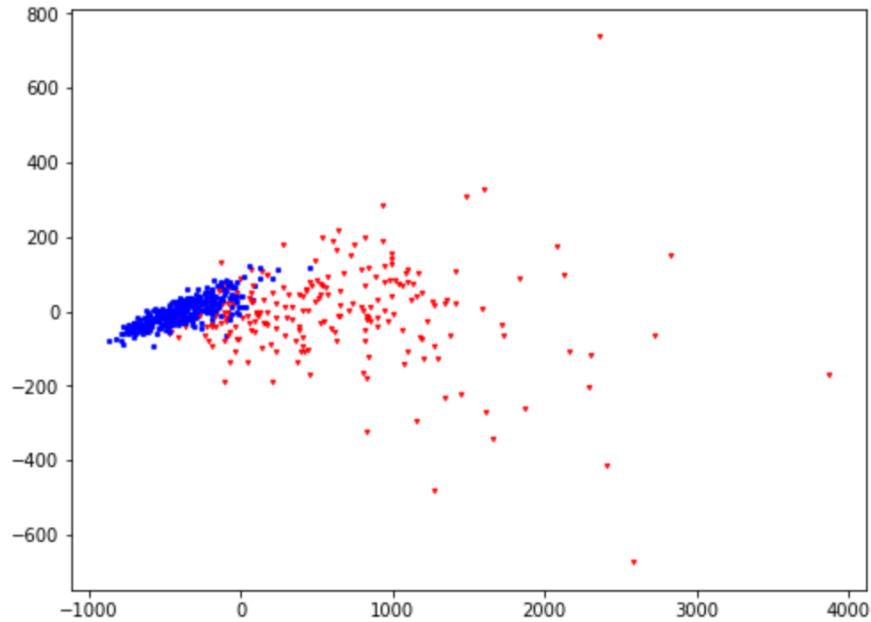
- 기존의 속성 값을 어떤 비율로 가중합산을 해야 가장 적절한 주성분을 얻을 수 있는지는 컴퓨터가 여러 가지 방법으로 조합을 만들어서 최적의 조합을 찾아준다.
- 주성분 분석을 하려면 최종적으로 필요한 주성분의 개수는 알려주어야 한다.

## pca 분석

- 아래는 유방암 데이터를 대상으로 PCA 분석을 수행한 결과이다. `pca.fit_transform` 함수를 사용하여 `pca`로 분석하였다.
- 이 예제에서 2개의 변수로 차원을 축소시키고 그라프로 2차원에 표현해보기 위해서 아래의 코드를 사용하였다. `m`은 마커의 모양을 결정 `c`는 색깔을

결정한다. 각 점들의 marker를 결정해 해서 for문을 사용하여 각 점들을 스캐터 플롯으로 그렸다.

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X)
m = ['v', 'o']
c = ['r', 'b']
plt.figure(figsize=(8,6))
for i in range(len(y)):
    plt.scatter(pca_result[:,0][i],pca_result[:,1][i], marker=m[y[i]], c=c[y[i]], s=5)
plt.show()
```



- 두 개의 주성분을 만들기 위해서 기존의 30개의 특성에 각각 어떤 가중치를 곱하였는지를 알려면 내부변수 `pca.components_`를 보면 된다. 아래는 소수점 3째 자리까지만 값을 출력했다.

```
pca.components_.round(3)
```

=>

```
array([[ 0.005,  0.002,  0.035,  0.517,  0.    ,  0.    ,  0.    ,  0.    ,  0.    , -0.
       ,  0.    , -0.    ,  0.002,  0.056, -0.    ,  0.    ,  0.    ,  0.    , -0.    , -0.    ,
       0.007,  0.003,  0.049,  0.852,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
      [ 0.009, -0.003,  0.063,  0.852, -0.    , -0.    ,  0.    ,  0.    , -0.    , -0.    ,
       -0.    ,  0.    ,  0.001,  0.008,  0.    ,  0.    ,  0.    ,  0.    ,  0.    , -0.001,
       -0.013, -0.    , -0.52 , -0.    , -0.    , -0.    , -0.    , -0.    , -0.    ]])
```

- 각 성분의 명칭은 아래와 같이 확인할 수 있다. (유방암 데이터에서 제공하는 특성 이름임)

```
cancer.feature_names
```

=>

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean
```

```
smoothness', 'mean compactness', 'mean concavity', 'mean concave points', 'mean symmetry', 'mean fractal dimension', 'radius error', 'texture error', 'perimeter error', 'area error', 'smoothness error', 'compactness error', 'concavity error', 'concave points error', 'symmetry error', 'fractal dimension error', 'worst radius', 'worst texture', 'worst perimeter', 'worst area', 'worst smoothness', 'worst compactness', 'worst concavity', 'worst concave points', 'worst symmetry', 'worst fractal dimension'],  
dtype='<U23')
```

- 또한 explained\_variance\_ratio\_에는 각 주성분 요소들이 얼마나 데이터를 잘 설명하는지를 볼 수 있다. 2개의 주성분만으로도 약 99.8%의 데이터를 표현한다고 나타났다.

```
pca.explained_variance_ratio_, sum(pca.explained_variance_ratio_)
```

```
=>
```

```
(array([0.98204467, 0.01617649]), 0.9982211613741719)
```

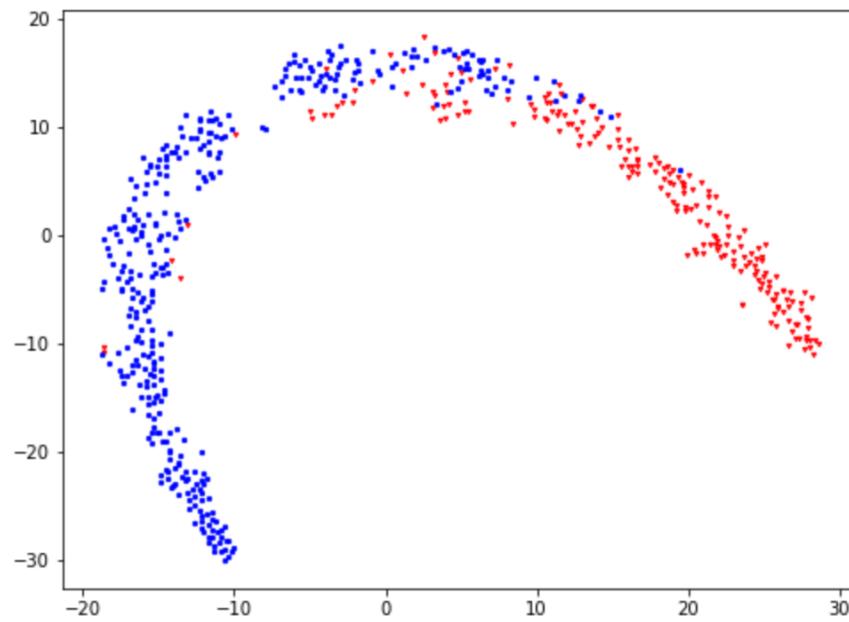
## t-SNE

- 데이터의 특성을 한 눈에 파악하는데 시각화가 매우 유용하다. 그러나 실제 세계의 데이터는 속성의 차원이 높으므로 이러한 다차원 공간에 속성들의 관계를 그리는 것은 불가능하다.
- 사람은 2차원 또는 3차원 공간에서의 시각화만 인식할 수 있다. 명확한 시각화를 위해서 데이터를 2, 3 차원 이하로 변환하는 작업도 비지도 학습의 일종이다.
- 시각화를 위해서 고차원의 특성을 가진 데이터를 저차원으로 축소하는 기술로 t-SNE가 널리 사용된다. 아래의 두 링크는 사이트에서 t-SNE를 사용함으로서 데이터의 표현을 얼마나 잘 해낼 수 있는지 보여주는 예시이다.

- 아래의 코드는 유방암 데이터를 t-SNE 시각화기법으로 데이터의 차원을 축소하여 표현한 것이다.

```
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=1000)
tsne_results = tsne.fit_transform(X)

m = ['v','o']
c = ['r','b']
plt.figure(figsize=(8,6))
for i in range(len(y)):
    plt.scatter(tsne_results[:,0][i],tsne_results[:,1][i], marker=m[y[i]], c=c[y[i]], s=5)
plt.show()
```



## 7.10. 모델 최적화

- 최적의 머신러닝 모델을 만드는 과정은 모델을 설계하고, 파라미터를 학습하고, 성능이 최적화 되도록 하이퍼 파라미터를 선정하는 단계로 구성된다.

모델 구조(알고리즘) 선택

파라미터 학습

하이퍼 파라미터 선택

- 모델 구조를 선택하는 것은 선형회귀, kNN, 결정트리, 랜덤포레스트, SVM, 로지스틱회귀, 그리고 뒤에서 배울 신경망등을 선택하는 과정이다.

- 파라미터 학습은 훈련 데이터를 사용하여 모델을 데이터에 맞추는 (fitting) 하는 것이다.
- 최적의 성능을 내는 모델을 완성하려면 모델의 구조를 구성하는 하이퍼 파라미터의 최적값을 찾아야 한다.
- 이 과정은 과대적합과 과소적합을 피하면서 성능평가지표 (정확도 등)을 최대로 얻는 하이퍼 파라미터를 찾아야 한다.

## 일반화

- 모델을 구성하는 환경 변수들을 하이퍼 파라미터라고 하는데, kNN알고리즘에서 k값, 선형회귀에서 계수의 개수, 결정트리에서 트리의 깊이, 랜덤포레스트에서 특성 선택비율, SVC에서 규제화 변수 등을 말한다.
- 과대적합을 피하면서 즉, 모델을 일반화하면서 높은 성능을 내는지를 검증하는데 사용하는 데이터가 검증 데이터이다. 검증 데이터는 파라미터를 학습시키는 “훈련에 사용되지 않은” 데이터여야 한다.
- 검증용 데이터와 테스트 데이터는 모델의 동작을 확인하는데 사용된다는 점에서는 같지만 목적에서 차이가 있다.
- 테스트 데이터는 모델을 최종적으로 만든 다음에 오로지 성능을 “평가”하기

위해서 사용되는 데이터인 반면, 검증 데이터는 모델을 만드는 “과정”에서 하이퍼 파라미터를 최적화 하고 과대적합 등을 검사하는데 사용하는 데이터이다.

- 모델 성능을 검증을 할 때에는 대부분 교차검증을 수행한다. 즉, 주어진 훈련 데이터 전체를 다시 훈련용과 검증용으로 나누되 나누는 방법을 달리하여 5~10회 정도 교차로 검증 데이터를 사용하는 방법이다.
- 최적의 하이퍼 파라미터를 찾는 일반적인 규칙은 없으며 일일이 여러 가지 하이퍼 파라미터 값들을 시도해봐야 한다. 뒤에서 그리드 탐색과 랜덤 탐색 방법을 소개하겠다.

## 릿지 규제

- 모델을 만들 때 훈련 데이터가 부족하면 과대적합하기 쉽다. 적은 수의 샘플로 학습을 시키기 때문이다. 과대적합을 피하려면, 즉 특정 데이터에 대해서만 잘 동작하는 모델이 아니라, 처음 보는 새로운 데이터에 대해서도 일반적으로 잘 동작하는 모델을 만드는 것이 중요하다.
- 모델을 일반화 하는 방법으로 손실함수를 MSE와 같이 간단한 형태를 사용하지 않고 좀 더 규제를 추가하는 방법이 널리 사용된다.
- 그 중에 대표적인 방법으로 손실함수에 MSE 항목과 함께 계수의 크기 자체도 줄이도록 하는 방법이 효과적이어서 널리 사용되며 이를 릿지 규제라고 부른다.

- 릿지 규제에서는 학습을 하는 동안에 손실함수로 다음과 같이 추가 항목을 더한 변형된 식을 사용한다.

$$J(W) = MSE(W) + \alpha \frac{1}{2} \sum_{i=1}^n W_i^2$$

- 위 식을 보면 앞에서 소개한 일반적인 MSE 항목에 더하여 계수의 자승의 합을 손실함수에 추가하였다. 이것의 의미는 계수 자체가 가능하면 작은 값이 되기를 원한다는 것이다.
- 알파 값은 리지 규제를 위해서 추가한 항의 비중을 얼마나 크게 할지를 정하는 하이퍼 파라미터이다. 알파 값은 일반화의 정도를 나타내는데 규제를 얼마나 강하게 적용할지를 정한다.

- 알파가 0이면 원래의 MSE만 사용하는 것이고 규제를 하지 않는 것이다. 알파를 10 정도로 크게 정하면 규제를 매우 강하게 하여 계수의 크기를 줄이는 데만 주력하고 MSE를 최소화 하는 것은 무시하겠다는 뜻이다.
- 계수 자승의 전체 합을 줄이면 왜 모델이 일반화하게 될까? 이는 여러 계수들을 가능한 골고루 반영하라는 의미를 갖는다. 왜냐하면 계수의 자승의 합을 줄이려면 각 계수의 크기를 줄여야 전체 자승의 합이 최소화되기 때문이다.
- 이는 마치 외부 경시대회에 내보낼 대표 학생을 선발하는데 국영수만 잘 하는 학생보다 전과목을 골고루 잘 하는 학생이 일반적으로 더 유리하다고 보는 것과 같다.

- 특정한 경우에는 (특히 학습할 샘플의 수가 적을 때), 국영수 세과목을 잘 하는 학생이 외부 경시대회에서 우수한상을 받았지만, 경시대회 문제가 사회, 과학, 예술 분야 중 어떤 응용문제가 나올지 모르는 상황이라면 전과목을 잘하는 학생이 일반적으로 더 시험을 잘 볼 수 있는 것에 비유할 수 있다.
- 예를 들어, 국어, 영어, 수학, 사회, 과학 과목의 가중치가 각각 [0.3, 0.3, 0.2, 0.1, 0.1]일 때와 [0.2, 0.2, 0.2, 0.2, 0.2]일 때 계수 자승의 합은 다음과 같이 차이가 난다.

$$\begin{aligned}
 & (0.3)^2 + (0.3)^2 + (0.2)^2 + (0.1)^2 + (0.1)^2 \\
 & = 0.09 + 0.09 + 0.04 + 0.01 + 0.01 = 0.24
 \end{aligned}$$

$$(0.2)^2 + (0.2)^2 + (0.2)^2 + (0.2)^2 + (0.2)^2 = 0.04 * 5 = 0.2$$

- 릿지 규제를 적용하면 국영수를 상대적으로 잘 하는 학생의 손실함수가 전과목을 골고루 잘 하는 학생보다 커지는 효과를 준다.
- 릿지 규제는 선형회귀에서만 사용되는 것이 아니라 SVM, 신경망 등 다른 머신러닝 모델에서도 사용할 수 있다. 릿지 규제는 L2 규제라고도 부른다 (자승의 합을 최소화 하므로).
- 아래는 붓꽃을 분류하는데 일반적인 선형회귀를 사용하는 것과 릿지 규제를 적용한 방법을 소개했다.

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
```

```
# 선형회귀  
linr = LinearRegression().fit(X_train, y_train)  
# 리지 규제  
ridge = Ridge().fit(X_train, y_train)
```

## 라쏘 규제

- 릿지 규제와 같이 모델의 일반화를 위한 규제의 다른 방법으로 라소(Lasso) 규제도 널리 사용된다, 라소 규제에서는 모든 계수의 절대치들의 합을 추가로 더하는 방법이다 (자승을 취하지 않음).

$$J(W) = MSE(W) + \alpha \sum_{i=1}^n |W_i|$$

- 이는 얼핏보면 릿지 규제와 유사한 것처럼 보이지만 사실은 반대의 효과를 얻는다. 릿지 규제에서는 특별히 비중이 큰 계수를 지양하고, 가능한 여러 가중치를 골고루 반영하는 효과를 얻었다.
- 그러나 라쏘 규제를 적용하면 절대값이 작은 계수가 먼저 사라지는 효과를

얻는다.

- 왜냐하면 절대치의 합을 줄이는 것이 목적이면 어떤 계수를 줄이든 동등한 효과를 얻으므로 이미 작은 값을 가진 계수들이 먼저 사라지게 된다. 라쏘 규제는 L1 규제라고도 부른다 (1차항인 절대치의 합을 최소화 하므로).
- 예를 들어 몸무게가 평균 60을 넘는, 65, 70, 90kg으로 구성된 세명에게 오차의 자승의 합을 줄이라고 하면 평균에서 멀리 벗어난 90kg인 사람이 열심히 몸무게를 줄여야 한다.
- 그러나 절대치의 합을 줄이라고 하면 누가 줄이든지 상관이 없다. 65kg인 사람이 5kg을 줄이는 것과 90kg인 사람이 5kg을 줄이는 것과 효과가 같다. 릿지 규제에서는 큰 값을 먼저 줄이게 되어 작은 값의 변수가 살아남을 가능성이 높아진다.

- 반면에 라쏘 규제에서는 절대값만 줄이면 되므로 어떤 변수를 줄이든 상관이 없고 따라서 적은 값이 사라질 (영향력을 잃을) 가능성이 크다.
- 결과적으로 라쏘 모델은 중요한 역할을 하지 않는 계수들을 제거해주는 효과를 얻으며 분석에서 다룰 특성 변수의 수를 줄이는 데 사용된다.
- 릿지와 라쏘 규제는 반대의 효과를 얻는다. 기본적으로 일반화를 위해서 릿지 규제가 널리 사용되며 변수의 개수를 줄이는 것이 필요한 때 라쏘 규제를 사용한다. 라쏘 규제의 사용법은 모델명칭만 바꾸어 주면 된다.

```
# 라소 규제
```

```
lasso = Lasso().fit(X_train, y_train)
```

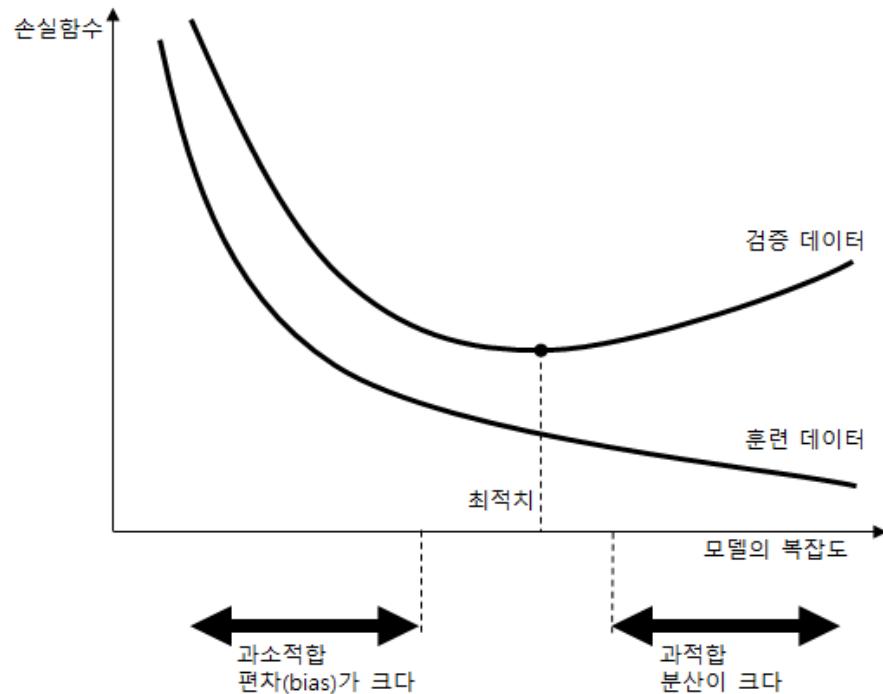
## 엘라스틱넷

- 릿지 규제와 라쏘 규제가 동시에 필요한 경우가 있다. 즉, 특정 계수가 크게 영향을 주는 것도 피하고 싶고(L2 규제), 동시에 영향력이 작은 계수의 수를 줄이는 것이 필요할 수가 있다(L1 규제).
- 이 때는 손실함수로 MSE와 L2 규제, 그리고 L1 규제 세 가지 항목을 모두 합하여 사용하는 방법이 있다. 이를 엘라스틱넷이라고 한다.
- 아래에서 알파는 일반화의 정도를 조정하는 하이퍼 파라미터이고 감마는 L2와 L1 규제의 반영 비중을 조절하는 하이퍼 파라미터이다.

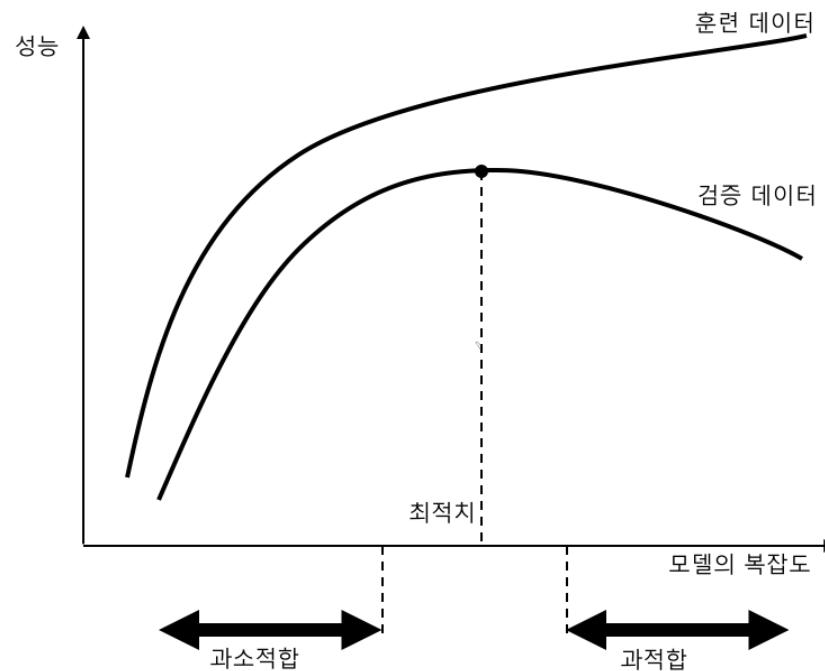
$$J(\theta) = MSE(\theta) + \gamma \alpha \sum_{i=1}^n |\theta_i| + \frac{1-\gamma}{2} \alpha \sum_{i=1}^n \theta_i^2$$

## 과대적합 과소적합

- 과대적합은 주어진 학습 데이터에 모델을 너무 잘 맞춘 것을 말한다. 과대적합을 검증하기 위해서 현재 학습한 모델이 일반성이 있는지 스스로 자체 검증을 해봐야 한다.
- 이를 위해서 주어진 데이터를 훈련 데이터와 검증 데이터로 나눈 후, 훈련 데이터에 대한 성능과 검증 데이터에 대한 성능을 비교하면 현재 모델이 과대적합인지 아닌지 알 수 있다.
- 아래 그림에서 모델이 복잡해질수록 과대적합이 되어 검증 오류가 증가하는 것을 볼 수 있다. 반면에 모델이 너무 간단하면 과소적합이 되어 성능이 나빠진다. 이들의 관계를 아래 그림에 나타냈다.



모델의 복잡도에 따른 손실함수의 변화



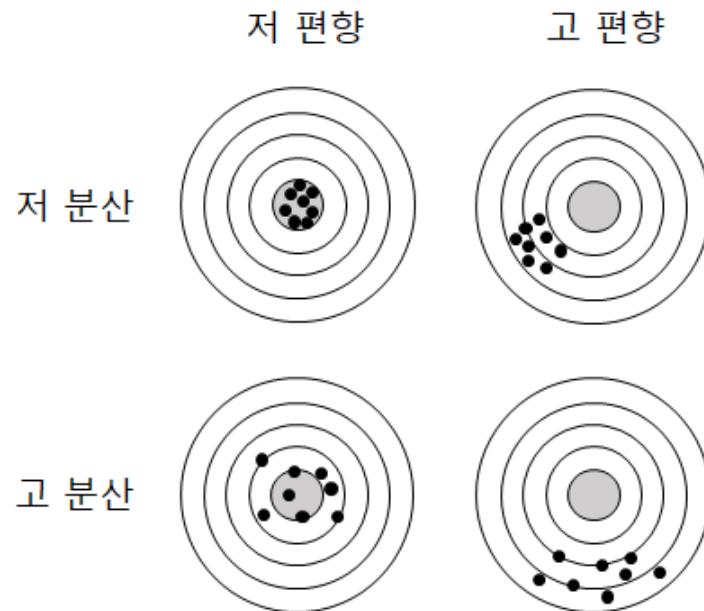
모델의 복잡도에 따른 성능(정확도 등)의 변화

## 편향과 분산

- 예측 모델에서 발생하는 오차는 분산(variance)과 편향(bias) 두 가지 성분으로 설명할 수 있다. 분산이란 모델이 너무 복잡하거나 학습데이터 민감하게 반응하여 예측 값이 산발적으로 나타나는 것이다. 편향이란 모델 자체가 부정확하여 피할 수 없이 발생하는 오차를 말한다.
- 예를 들어 kNN알고리즘에서  $k=1$ 인 경우는 모델이 학습 데이터에 민감하게 반응하여 예측의 변화가 커지므로 분산 성분이 증가한다. 그러나 편향은 발생하지 않는다. 이러한 경우를 과대적합되었다고 한다.
- 반면에  $k=N$ 인 경우는 모델은 항상 평균치로 동일하게 예측하므로 분산은 거의 발생하지 않는다. 그러나 모델 자체가 부정확하여 편향이 커진다. 모든 사람의 키를 평균치로 예측하면, 예측치는 평균치로 동일하므로 분산은

없어지지만 편향 오차가 클 수 밖에 없다.

- 이는 과소적합의 현상이다. 과대적합-과소적합 그리고 분산-편향의 관계를 아래 그림에 나타냈다.



- 모델이 훈련 데이터에 너무 종속적이거나 정교하면 ( $k=1$  인 경우) 분산이 늘어나고 편향은 줄어든다 (위 그림에서 좌 하). 모델이 너무 단순하면 ( $k=N$ 인 경우) 분산을 줄어드나 모델이 부족하여 생긴 편향을 증가한다.
- 위 그림에서 최상의 모델은 좌상의 위치이다. 최악의 모델은 우하이다. 좌하는 과대적합한 경우의 결과이고, 우상의 모델은 과소적합하여 평균치를 예측하는 경우이다.
- 같은 조건에서는 즉, 주어진 훈련 데이터가 같고 같은 컴퓨팅 자원을 이용한다면 하이퍼 파라미터를 잘 선택해서 과대적합과 과소적합을 모두 피하는 모델을 만들어야 한다.
- 결정 트리에서는 트리의 구조가 너무 복잡하면, 예를 들어 트리의 깊이

(depth)를 깊게 설계하면 과대적합이 되기 쉽고, 트리를 너무 얕게 제한하면 과소적합하기 쉽다.