리지 모델

- 계수를 추정할 때 약간 다른 값(quantity)을 최소화시키는 것 외에는, 리지 회귀(ridge regression)는 최소제곱법과 매우 유사한 형태이다.
- 특히, 리지 회귀(ridge regression) 계수 추정치는 다음 식을 최소화하는 계수추정치(beta값)를 갖는다.
- λ는 조절모수로 개별적으로 정해진다.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2,$$

리지와 라소 회귀

- 선형회귀는 잔차 제곱의 합계(RSS)만 최소화 하는 반면 리지와 라소회귀 분석은 튜닝 매개변수 λ로 계수를 정규화하기 위해 계수에 패널티(수축 패널티)를 적용한다.
- λ가 0이면 패널티에 영향이 없고 리지/라소 회귀는 선형 회귀와 동일한 결과를 도출한다.
- 반면 λ가 무한대로 가면 계수들은 0이 된다.
- 목표는 주어진 비용 제약조건하에서 잔차제곱 최소화를 수행하는 것이다.
- 각 λ마다 대응하는 s가 존재해 전체 목적함수가 페널티 요인을 가진 함수가 되도록 한다.

리지 회귀 분석

- 리지 회귀 분석은 최소 자승법이 높은 분산을 갖는 상황에서 잘 작동한다.
- 리지 모델은 2^p가지 모델이 필요한 최적 부분 집합 선택법에 비해 계산 자원을 적게 소모한다.
- 리지는 어떤 고정된 λ에 관해서도 단일 모델만 적합화하고 모델 적합화 절차도 매우 빠르다.
- 단점
  - 모든 예측 변수를 중요도에 따라 가중값만 축소시킬 뿐, 0값을 부여하지 않기 때문에 불필요한 변수가 제거 되지 않고 항상 남아있게 된다.
  - 라소회귀는 이 문제를 해결한다.

- 예측 변수가 아주 많은 경우 리지를 사용하면 정확성은 제공할 수 있을 지 몰라도 모든 변수를 포함하게 돼 모델의 간결한 표현을 위해서는 바람직하지 않다.
- 라소에는 이런 문제가 발생하지 않는다.
  - 불필요한 변수의 가중값에 0값을 부여해 제거해 버리기 때문이다.

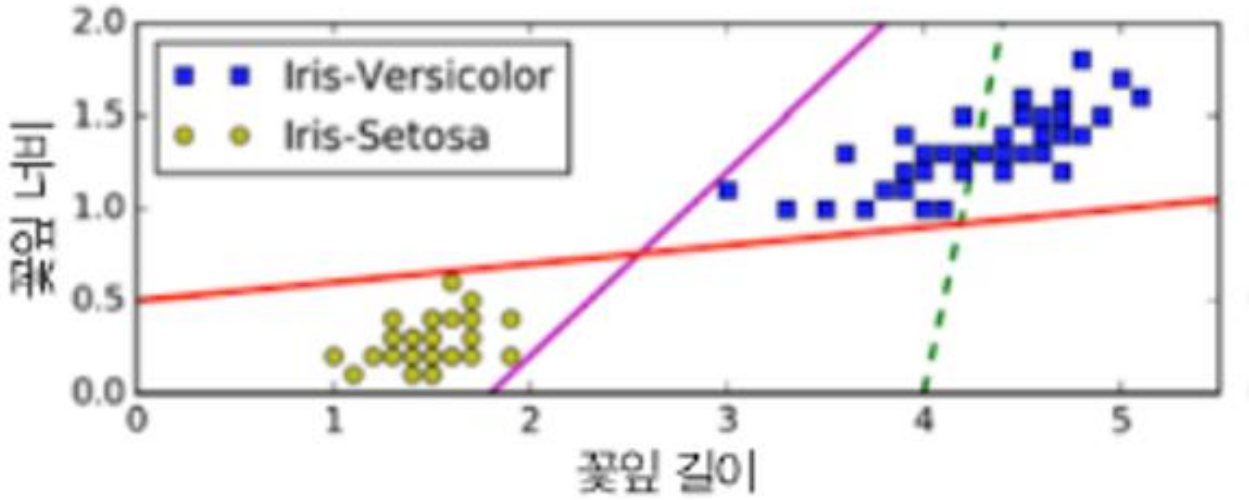
리지/라소 회귀 분석 머신러닝 예

- 리지 회귀 분석은 머신 러닝 모델로서 개별 독립 변수에 관해 어떠한 통계학적 진단도 수행하지 않는다.
- 단순히 테스트 데이터에 관해 적합화를 수행하고 그 정확성을 검사한다.
- 라소 회귀 분석은 계수의 제곱값을 최소화 하는 대신 계수의 절대값을 최소화한다.
  - 이렇게 함으로써 무의미한 변수를 제거 할 수 있는데, 모델을 상당히 간결하게 표현할 수 있다.

비지도학습

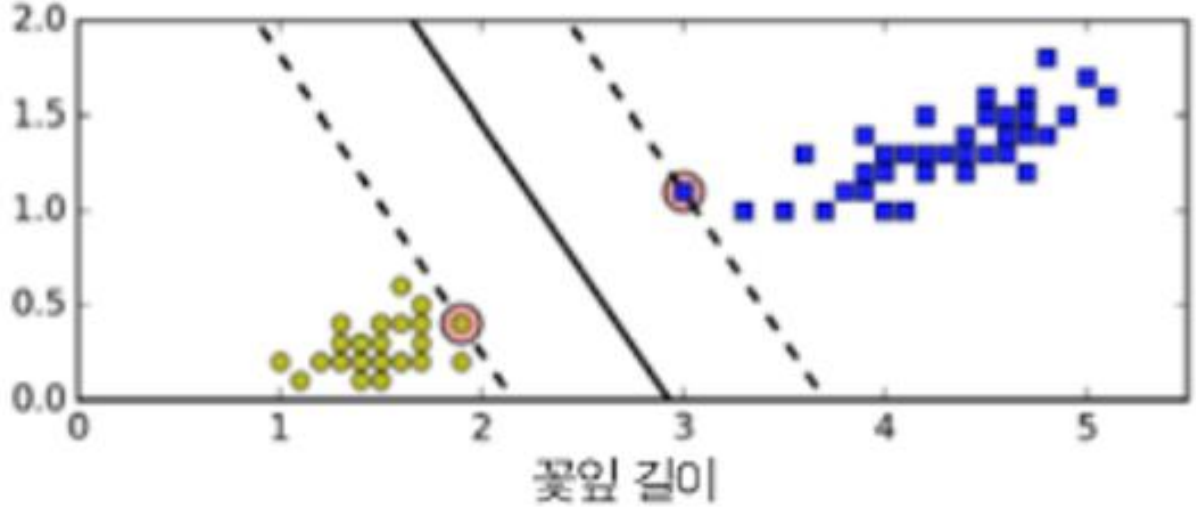
- 목표는 분류나 회귀 같은 기법을 사용할 목표 변수가 존재하지 않는 데이터로부터 숨어 있는 패턴이나 구조를 찾는 것이다.
- 그 결과 해석이 주관적이며 부류 예측이나 연속 변수 같은 간단한 분석 목표도 없기 때문에 힘들다.
- 비지도 학습의 결과에 관한 평가는 쉽지 않다. 범용적으로 통용되는 결과 검증 방법이 없기 때문이다.

SVM(서포트 벡터 머신)



- 점선은 두개의 데이터를 적절히 나누지 못했고 나머지 두 직선은 구분은 했지만 너무 가까워서 새로운 데이터가 들어갔을 때 알맞게 구분하기 어렵다.

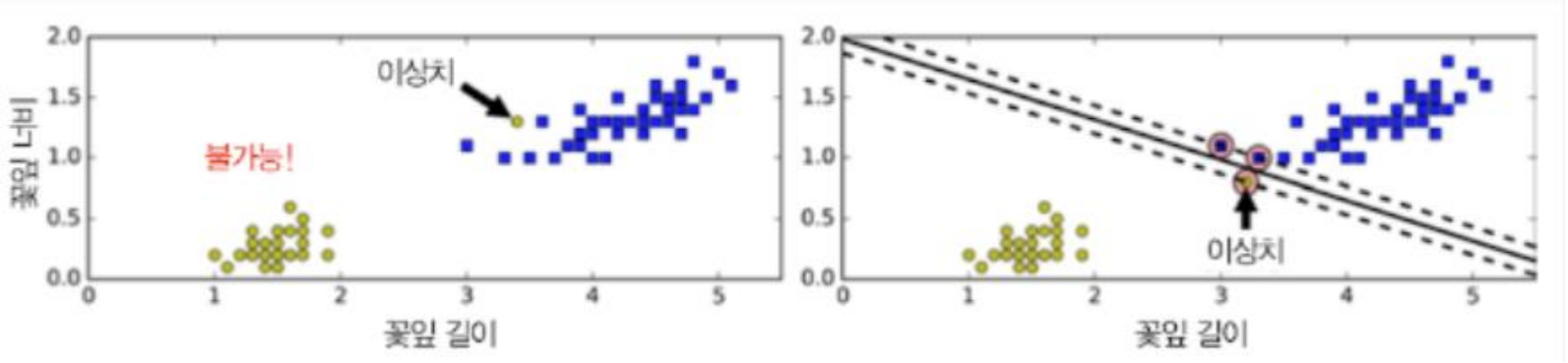
SVM(서포트 벡터 머신)



- 큰 직선은 일정한 거리를 두고 두개의 데이터를 잘 구분하고 있다.
- 여기서 일정한 거리를 마진이라고 한다.
- 마진에 데이터가 없고 실선으로부터 최대한 멀리 떨어지게 만드는 것을 라지 마진 분류라고 한다.
- 점선에 딱 걸치고 있는 데이터 두개를 기준으로 새로운 데이터가 들어왔을 때 구분시켜 준다. 이러한 기준이 되는 데이터를 서포트 벡터라고 한다.
- 서포터 백터 머신은 데이터들의 스케일을 맞추어 주는게 중요하다.

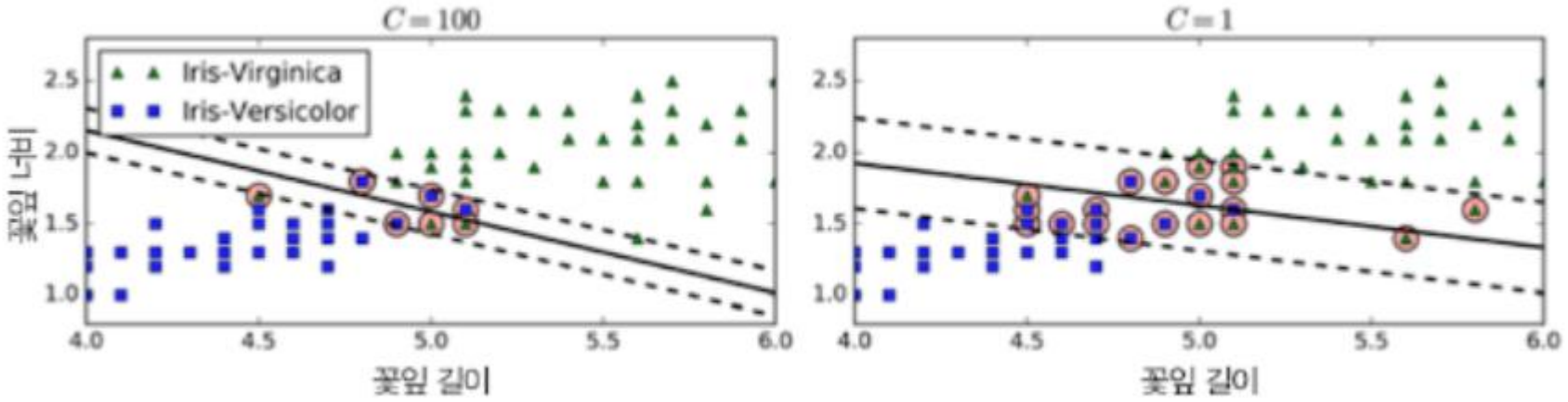
소프트 마진 분류

- 마진 바깥쪽으로 올바르게 분류하는 것을 하드 마진 분류라고 한다.
- 두가지 문제점
  - 선형적으로 구분 되어야하고
  - 이상치에 민감하다.



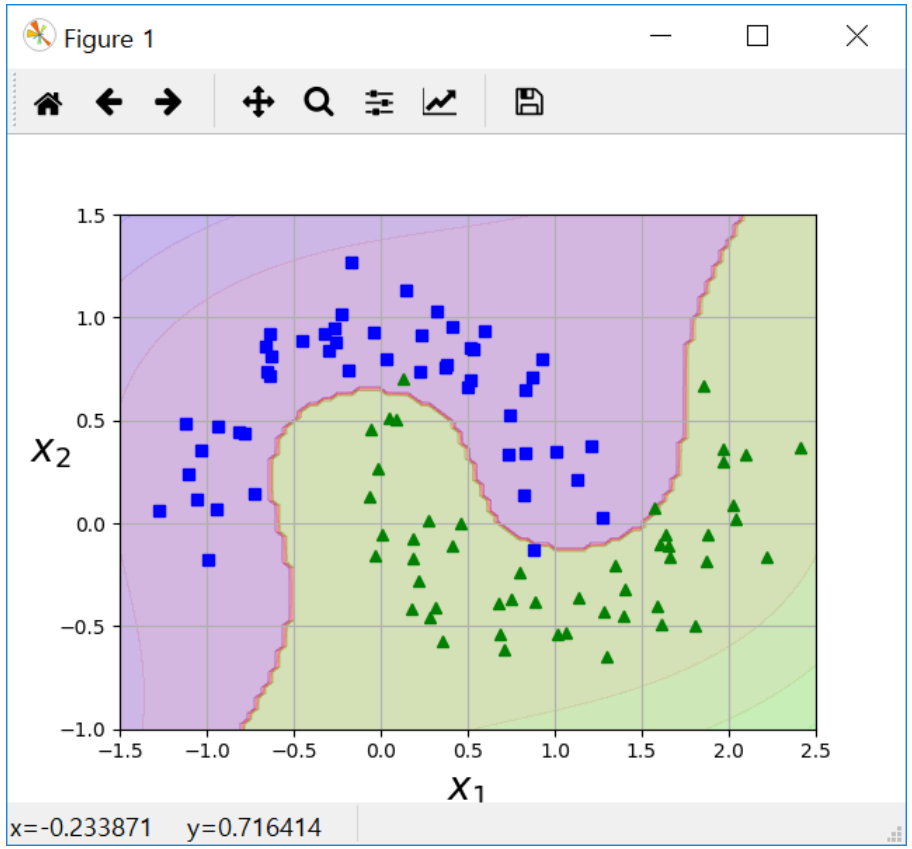
- 하드마진 분류를 사용할 수 없고, 앞의 그래프와 결정경계가 다르고 일반화를 잘 하지 못한 것 같다.
- 이런 문제를 피하기 위해서는 조금 더 유연한 모델이 필요하다. 이를 소프트 마진 분류라고 한다.
  - 최대한 마진거리를 유지
  - 어느 정도 마진 오류를 인정
    - 마진 오류: 마진 한중간에 데이터가 있거나 반대편에 데이터가 존재하는 경우
- 사이킷 런에서는 C라는 하이퍼 파라미터를 사용해서 균형을 맞추어 줄 수 있다.
  - C가 크다면 마진의 크기가 작아지고 마진 오류도 작아진다.
  - C가 작다면 마진의 크기가 커지고 마진 오류도 커진다.

- 만약 svm 모델이 과대적합이면 c를 감소시켜 조금 더 유연하게 만들어 줄 수 있고 과소 적합이라면 c를 증가시켜 타이트하게 만들어 줄 수 있다.



CODE 표현

- LinearSVC(C=1)를 이용하여 코드로 표현할 수 있다.



K-평균 군집화

- 관측 값을 서로 유사성이 높은 것끼리 묶어 다수의 그룹으로 만드는 것으로 동일 그룹 내 구성원 간의 유사성은 매우 높지만 다른 그룹의 구성원과의 유사성은 거의 없도록 하는 것이다.
- 데이터 세트 내부의 기본 패턴을 알아내거나 특정 성질을 가진 그룹을 생성할 때 흔히 사용하는 방식이다.
  - 사회 관계망의 경우, 커뮤니티를 알아낸 후 구성원 간의 끊어져 있는 관계를 연결하도록 추천하는 데 사용할 수 있다.
- K-평균 군집화 알고리즘은 반복적인 작업으로 각 클러스터의 현재 중심을 구성원의 평균 위치로 옮기고 중심이 옮겨진 위치로부터 가장 가까운 점들로 다시 구성원을 형성한 후 새로운 평균을 구하고 위치를 옮기는 작업을 반복한다.
- 이 반복은 클러스터의 중심이 더 이상 유의미한 변화가 없거나 미리 설정해둔 반복 횟수에 도달하면 멈춘다.

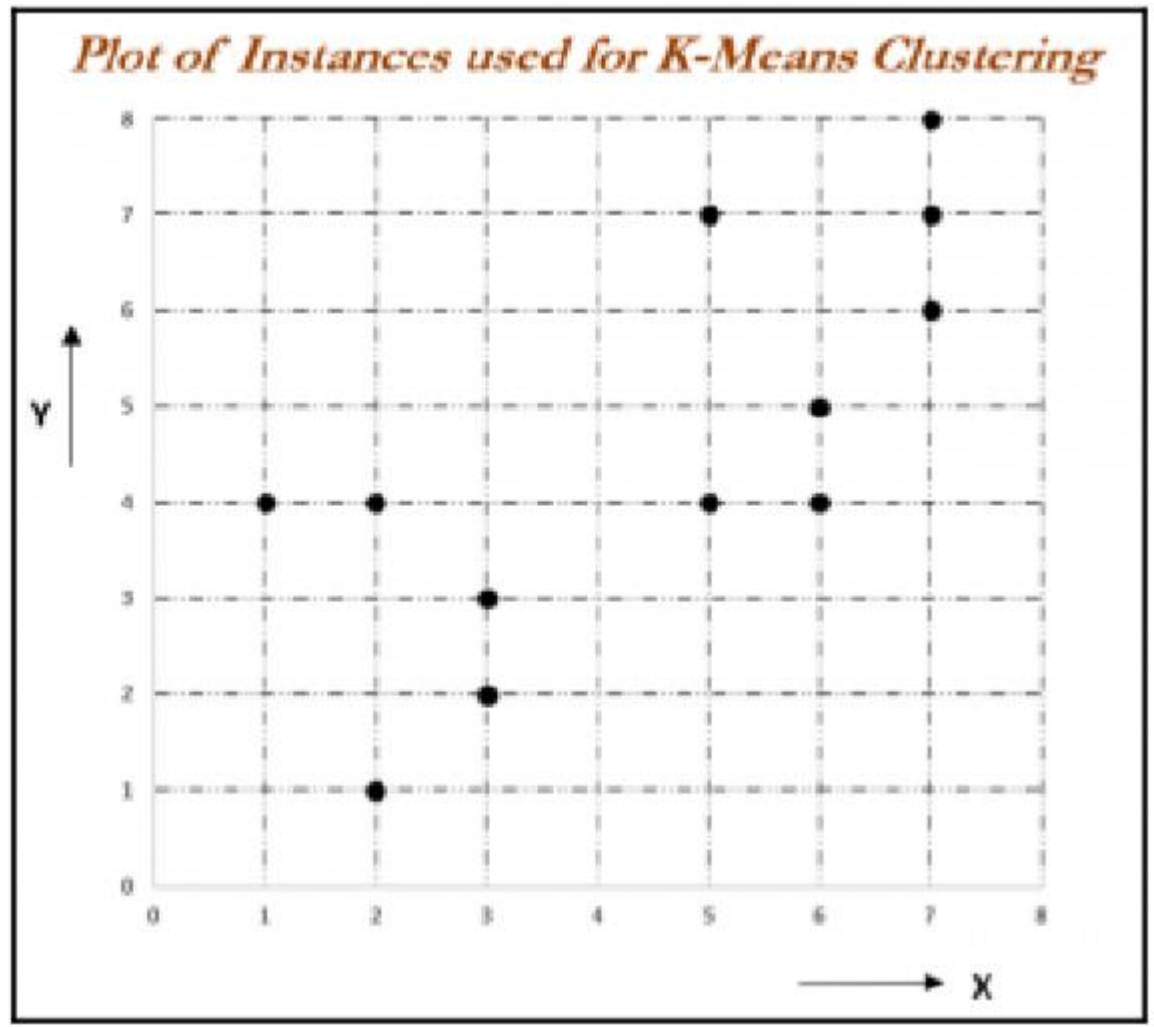
K-평균 군집화

- K-평균의 비용 함수는 클러스터의 중심과 클러스터에 속한 관측값 사이의 유클리디언 거리로 결정된다.
- 하나의 클러스터만 존재하는 경우를 생각해보자.
  - 이때는 모든 점이 하나뿐인 중심과의 거리를 구해 모든 점의 평균점과 비교한다.
  - 이제 클러스터가 2개가 되면 2개의 평균이 계산되고 인접도에 따라 일부는 1번, 나머지는 2번 클러스터에 배정된다. 그 후 비용 함수는 클러스터의 중심과 관측값 사이의 거리를 구해 계산한다.

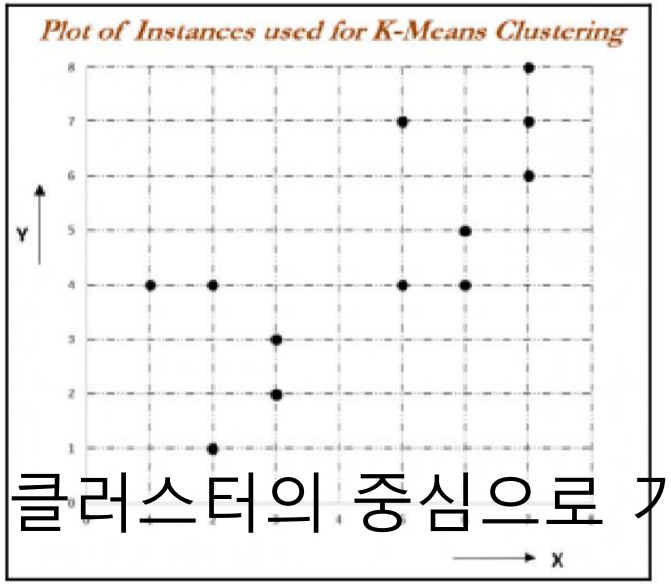
K-평균 군집화

- 12개의 점으로 구성된 데이터를 사용해 K-평균 군집화의 기본 작동 원리를 알아본다.
- 각 점은 X와 Y의 쌍으로 이뤄진다
- 주어진 과제는 주어진 데이터의 최적 군집을 결정하는 것이다.

점	X	Y
1	7	8
2	2	4
3	6	4
4	3	2
5	6	5
6	5	7
7	3	3
8	1	4
9	5	4
10	7	7
11	7	6
12	2	1

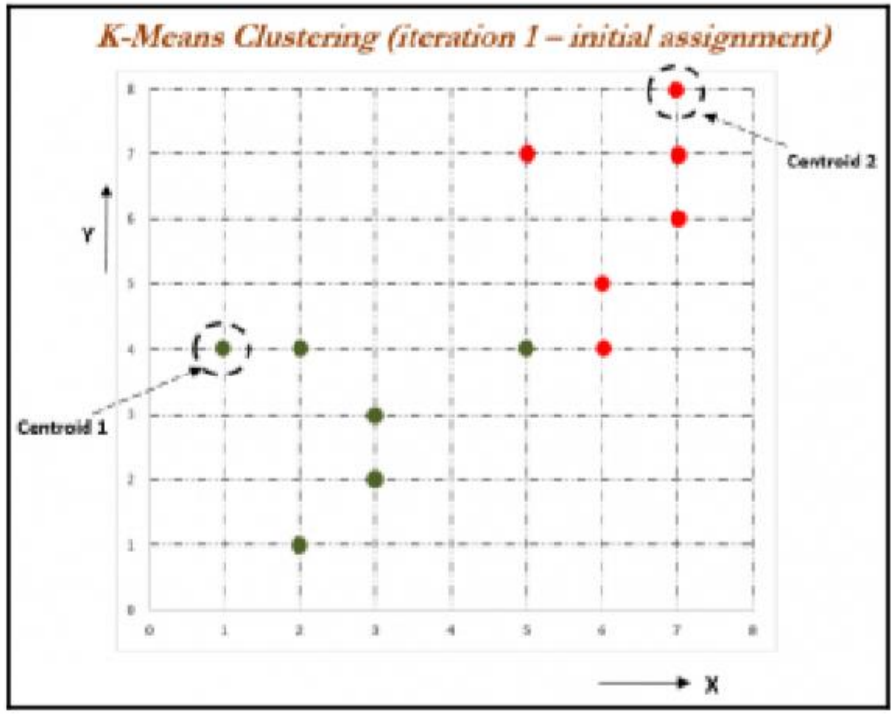

Plot of Instances used for K-Means Clustering
Y
X
0
1
2
3
4
5
6
7
8

- 2차원 평면에 점을 그려놓고 눈대중으로 살펴보면 대략 2개의 클러스터가 가능해 보인다.
- 그림의 가운데서 왼쪽 하단 방향을 한데 묶어 하나의 클러스터를 만들고 오른쪽 상단 방향을 또 하나로 묶어 다른 클러스터로 만들면 된다.
- 실제 응용에서는 변수가 아주 많거나 차원이 높아 이런 식의 시각화는 불가능하므로 이 문제를 근본적으로 해결해 줄 수 있는 수학적이며 알고리즘에 따른 방법이 필요하다.



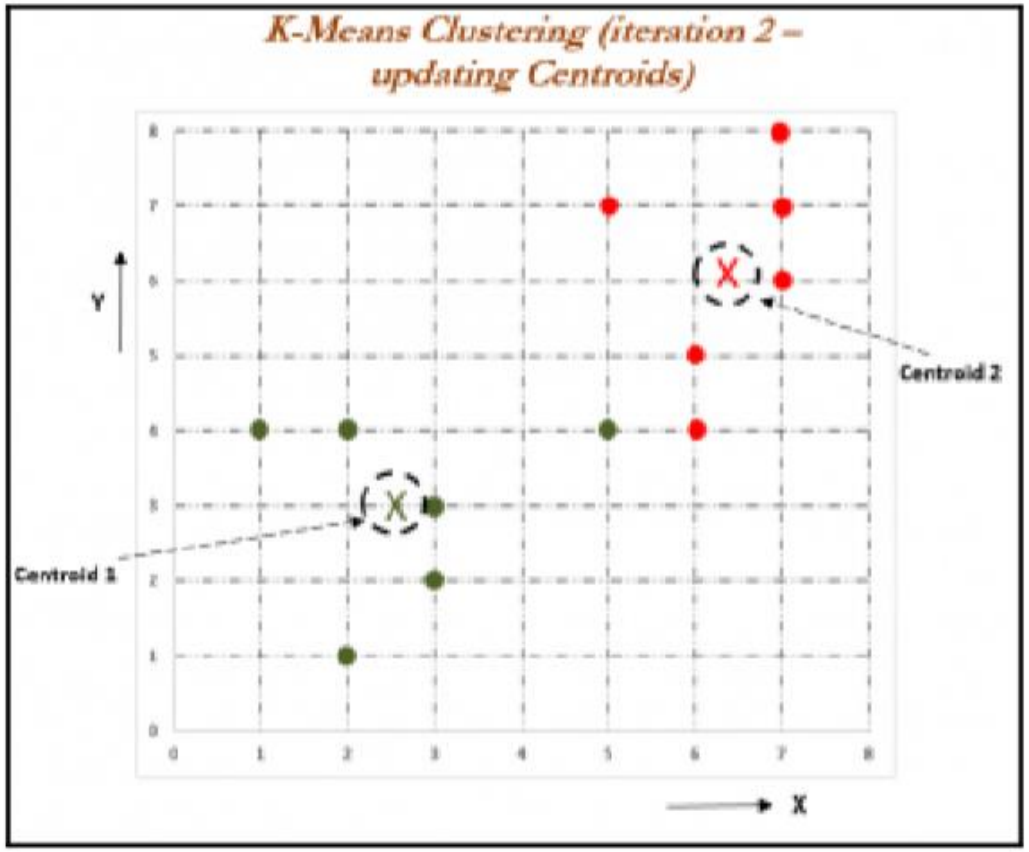
- 반복1
- 12개의 점들 중 2개를 임의로 골라 이 점을 2개 클러스터의 중심으로 가정해보자.
- 점1(X=7,Y=8)과 점8(X=1,Y=4)이 양 끝에 있는 것처럼 보이므로 이 두 점을 중심이라고 가정하자.
- 이제 각 점들과 이 점과의 유클리디언 거리를 구해 더 가까운 중심으로 각 점을 배정한다.
- 두점 A(X1,Y1)와 B(X2,Y2)거리
  - $\sqrt{(X2-X1)+(Y2-Y1)^2}$

- 예를 들어 점6의 경우에는 중심(중심1,중심2)들로 부터의 거리
  - 중심1과 점6의 거리 = $(5-1)^2+(7-4)^2 = \sqrt{25}=5$
  - 중심2와 점6과의 거리 = $(5-7)^2+(7-8)^2 = \sqrt{5}=2.24$
- 점들이 자신과 가까운 중심에 배정된 모습을 나타낸다.



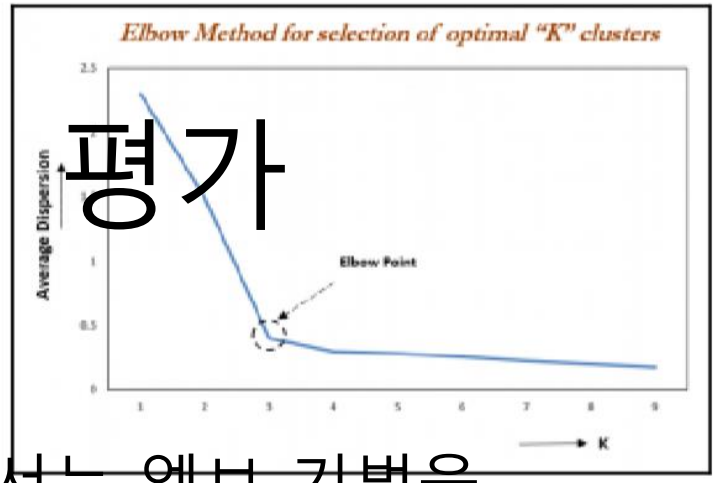
- 반복2
  - 각 클러스터의 새로운 중심을 구한다. 새로운 중심을 구하는 방법은 각 클러스터에 있는 모든 점을 좌표별로 평균한 점을 계산한다.
  - 중심1X = (2+3+3+1+5+2)/6 =2.67
  - 중심1Y = (4+2+3+4+4+1)/6 =3.0
  - 중심의 위치를 변경했으면 가장 가까운 점들로 클러스터를 다시 배정해야 한다.

  계산 결과 클러스터의 변화가 없으므로 결과는 수렴했다.



- 반복 3
  - 여기서 새로운 중심과 각 점들과의 유클리디언 거리를 구해 클러스터를 다시 배정한다.
  - 배정에 변화가 생기면 앞 과정을 다시 반복하고 이 반복은 더 이상의 변화가 없거나 미리 설정한 반복 횟수에 도달할 때까지 계속된다.
  - 각 점들과 새로운 중심들과 새로 계산된 거리

최적 클러스터 계수와 클러스터 평가



- 엘보기법
  - K-평균 군집화에서 최적 클러스터 개수를 찾기 위해서는 엘보 기법을 사용해야 한다.
  - K 값의 변화에 따른 비용 함수의 값을 그래프로 그린 것이다.
  - K 값이 증가하면 평균적인 왜곡이 줄어든다.
    - 각 클러스터의 구성원들이 줄어들수록 모든 구성원이 중심에 더 가까워지기 때문이다.
  - K 값의 증가에 따라 왜곡의 개선이 꾸준히 이뤄지다가 어느 시점이 되면 K 값이 증가해도 평균적인 왜곡의 개선이 급격히 저하되는 시점이 발생한다.
  - K 값 증가에 따른 왜곡 개선의 저하가 최대가 되는 지점을 엘보라 부르고, 더 이상 클러스터를 증가시키지 않아도 될 최적 시점으로 볼 수 있다.

- 실루엣 계수를 이용한 클러스터의 평가
  - 실루엣 계수는 클러스터가 어느 정도 밀집해 있는지 알아보는 척도이다.
  - 이 값이 높을 수록 클러스터의 품질이 우수하다는 것을 의미한다.
  - 클러스터들이 잘 분리돼 있으면 높은 값을 갖고, 서로 겹치면 낮은 값을 가지는데, -1부터 1사이의 어느 한 값을 가지며 값이 클수록 좋다.
  - 실루엣 계수는 각 개체(점)별로 계산하는데, 개체의 집합에 관해서는 개별 개체 점수의 평균을 계산한다.
    - $s = \frac{(b-a)}{\max(a,b)}$
  - a는 같은 클러스터 내의 다른 관측 값과의 평균 거리, b는 가장 가까운 다른 클러스터 내에 속한 관측값들과의 평균 거리이다.

데이터 예제

- 다음 코드로 class변수를 종속 변수로 분리해 그래프에 색상을 그릴 때 이용하고, 주어진 x에 관해 비지도학습 알고리즘을 적용한다. 따라서 어떠한 목표 변수도 주어지지 않는다.
- x_iris = iris.drop(['class'] ,axis=1)
- # drop() 의 경우 기본 값으로 행의 인덱스를 받기 때문에 열을 기준으로 삭제하고 싶을 경우 axis=1 값을 추가해주어야 합니다.
- y_iris = iris["class"]

```
C:\JIN\Anaconda3\envs\tf\python.exe "C:/Users/jin/PycharmProjects/alg/
   sepal_length  sepal_width     ...       petal_width        class
0           5.1          3.5     ...               0.2  Iris-setosa
1           4.9          3.0     ...               0.2  Iris-setosa
2           4.7          3.2     ...               0.2  Iris-setosa
3           4.6          3.1     ...               0.2  Iris-setosa
4           5.0          3.6     ...               0.2  Iris-setosa
```

데이터 예제

- 표본 척도를 사용해 세가지 클러스터를 구성한다. 그러나 실제로는 몇 개의 클러스터가 필요한지 사전에 알 수 있는 방법은 없다.
- 시행착오를 거쳐 숫자를 찾아낼 수 밖에 없다.
- 반복횟수를 300으로 설정한다고 하면, 이 값 역시 변경 가능하고 값의 변경에 따른 결과의 변화도 지켜볼 필요가 있다.

데이터 예제

- k_means_fit = KMeans(n_clusters=3 ,max_iter=300)
- k_means_fit.fit(x_iris)
- print ("₩nK-Means Clustering - Confusion Matrix₩n₩n“ ,pd.crosstab(y_iris ,k_means_fit.labels_ ,rownames = ["Actuall"] ,colnames = ["Predicted"]))
- print ("₩nSilhouette-score: %0.3f" % silhouette_score(x_iris, k_means_fit.labels_, metric='euclidean'))

```
K-Means Clustering - Confusion Matrix

 Predicted         0   1   2
Actuall
Iris-setosa       0  50   0
Iris-versicolor  48   0   2
Iris-virginica   14   0  36

Silhouette-score: 0.553
```

민감도 분석

- 이를 통해 실제로 몇 개의 클러스터를 만들어야 더 나은 구분이 가능한지 검사한다.
- 실루엣 계수 결과에 따르면 k가 2나 3일 때 다른 모든 경우보다 더 좋은 점수를 보인다.

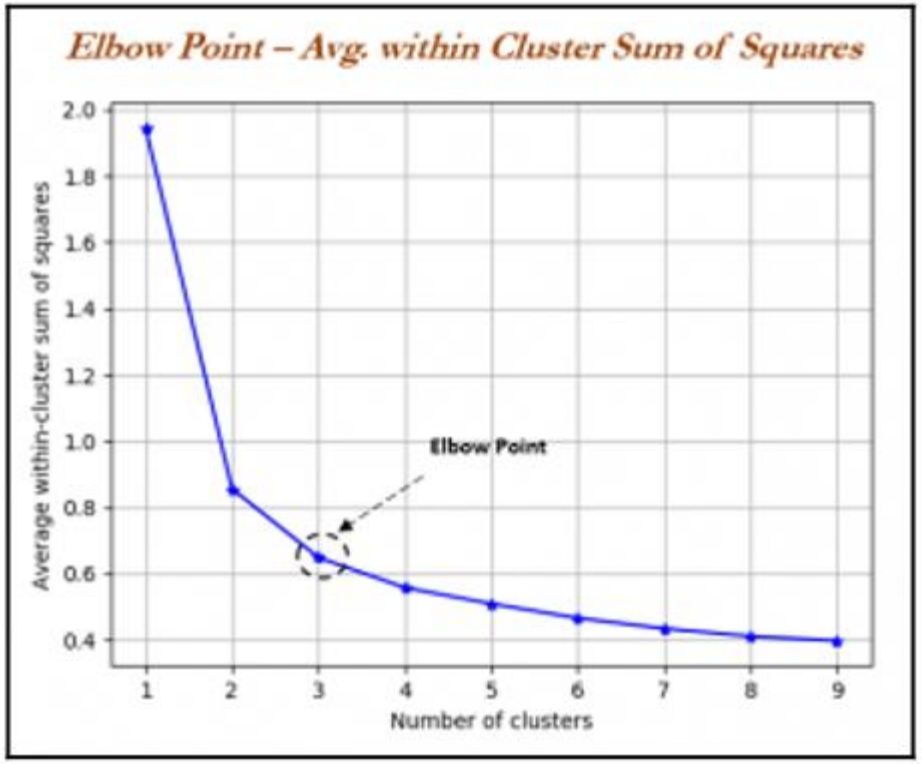
```
Silhouette-score: 0.553
For K value 2 ,Silhouette-score: 0.681
For K value 3 ,Silhouette-score: 0.553
For K value 4 ,Silhouette-score: 0.498
For K value 5 ,Silhouette-score: 0.489
For K value 6 ,Silhouette-score: 0.368
For K value 7 ,Silhouette-score: 0.351
For K value 8 ,Silhouette-score: 0.361
For K value 9 ,Silhouette-score: 0.333
```

- for k in range(2 ,10):
- k_means_fitk = KMeans(n_clusters=k ,max_iter=300)
- k_means_fitk.fit(x_iris)
- print ("For K value" ,k
- ,",Silhouette-score: %0.3f" % silhouette_score(x_iris, k_means_fitk.labels_, metric='euclidean'))

엘보 포인트 – 클러스터 내 제곱합의 평균

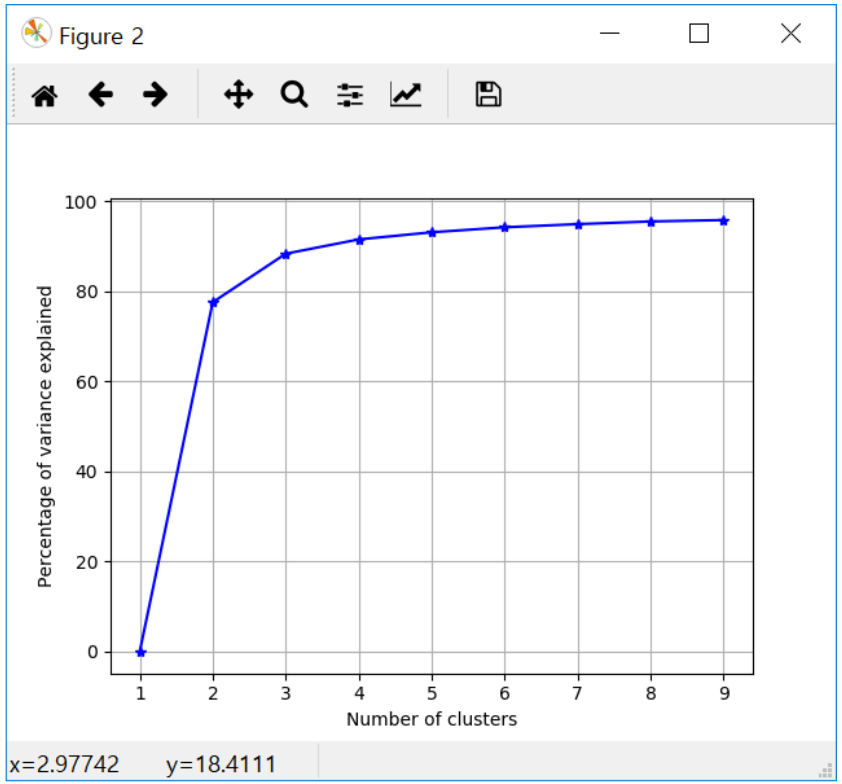
클러스터내의 평균 분산과 엘보 그래프도 그려보자.

엘보 그래프를 살펴보면 세 번째 값에서 기울기가 급격히 변하는 것을 알 수 있다. 최적 값k는 3으로 추정할 수 있다.



엘보 포인트 – 해석된 분산 퍼센티지

- 분산의 비율을 구한다.
- 최적클러스터 개수로 결정되기 위해서는 이 값이 최소한 80%를 넘어야 한다.
- 여기서도 k 값이 3일 때 해석된 분산 비율이 적절해 보인다.
- 세가지 척도(실루엣, 클러스터 내 평균, 해석된 분석 총 비율)을 종합해 보면 클러스터의 개수는 3이 이상적인 것으로 보인다.

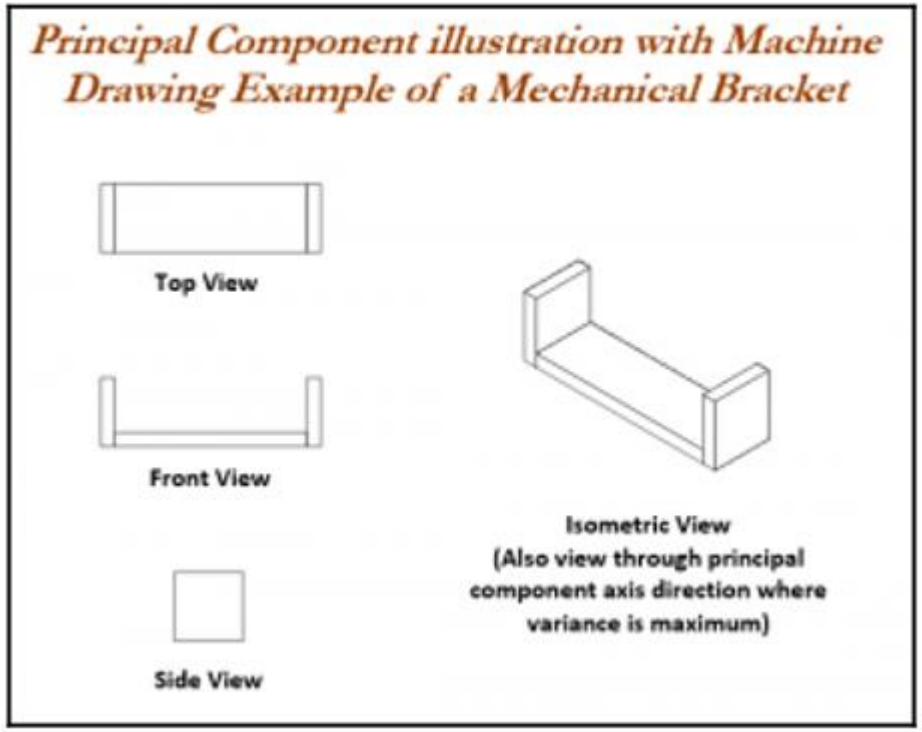


주성분 분석 - PCA

- 활용도가 높은 차원 축소 기술이다.
- 데이터를 낮은 차원으로 투영해 차원을 줄인다.
- 예를 들어 2차원의 데이터는 선으로 투영해 차원을 줄일 수 있다. 이 경우 각 점은 더 이상 좌표의 쌍이 아니라 하나의 값이 된다.
- 3차원의 데이터는 변수들을 평면에 투영해 그 차원을 줄일 수 있다.

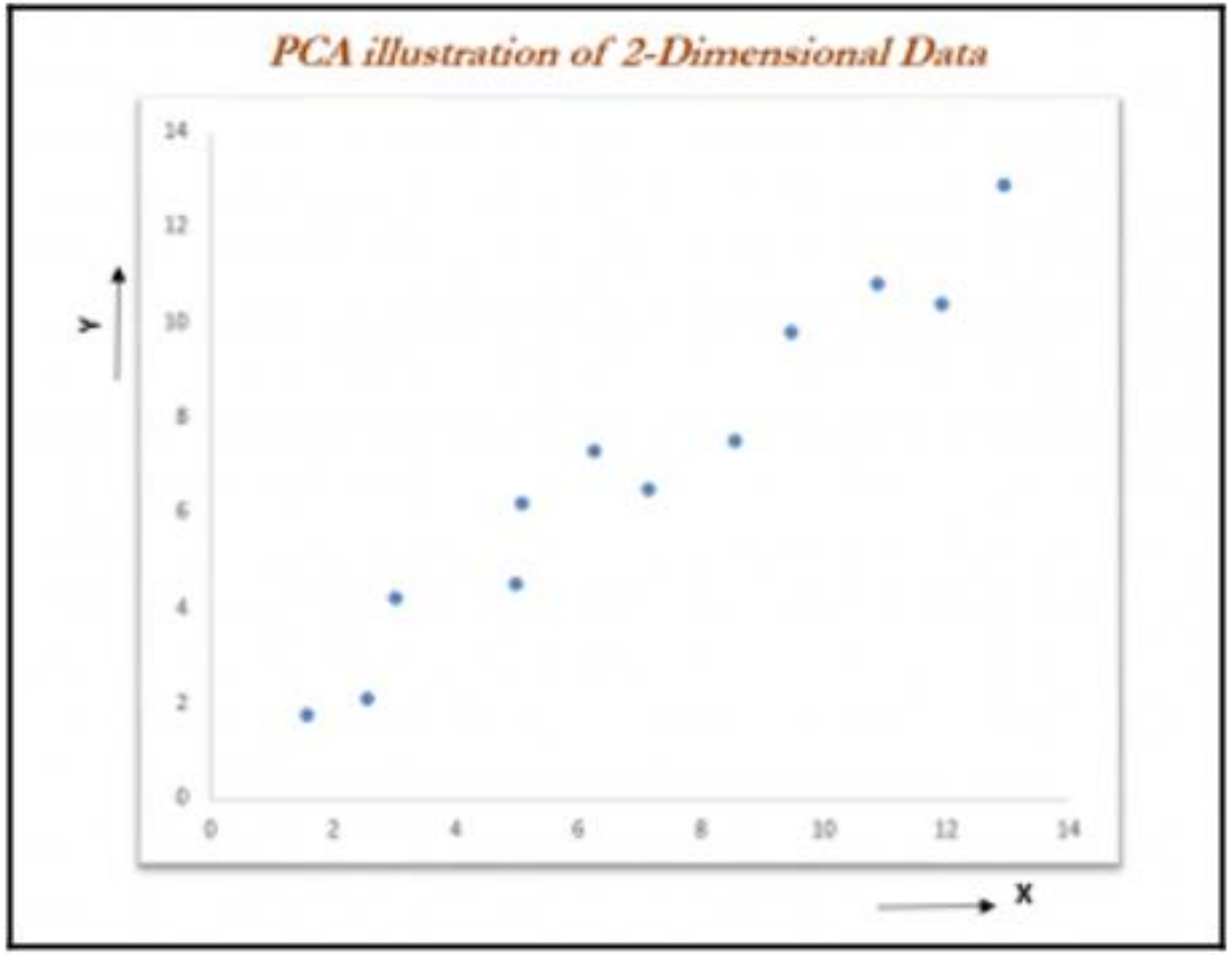
주성분 분석 - PCA

- 브래킷이라는 기계 모듈을 기계 공학에서 쓰는 기계 제도법으로 그린 것이다.
- 각각 상면도,정면도, 측면도이다.
- 오른쪽은 등각도가 나타나 있다.
  - 등각도를 통해 전체 생김새를 시각적으로 알 수 있다.
- 왼쪽 그림을 변수로 생각하면 오른쪽 그림은 모든 변이가 반영된 첫 번째 주성분으로 생각할 수 있다.



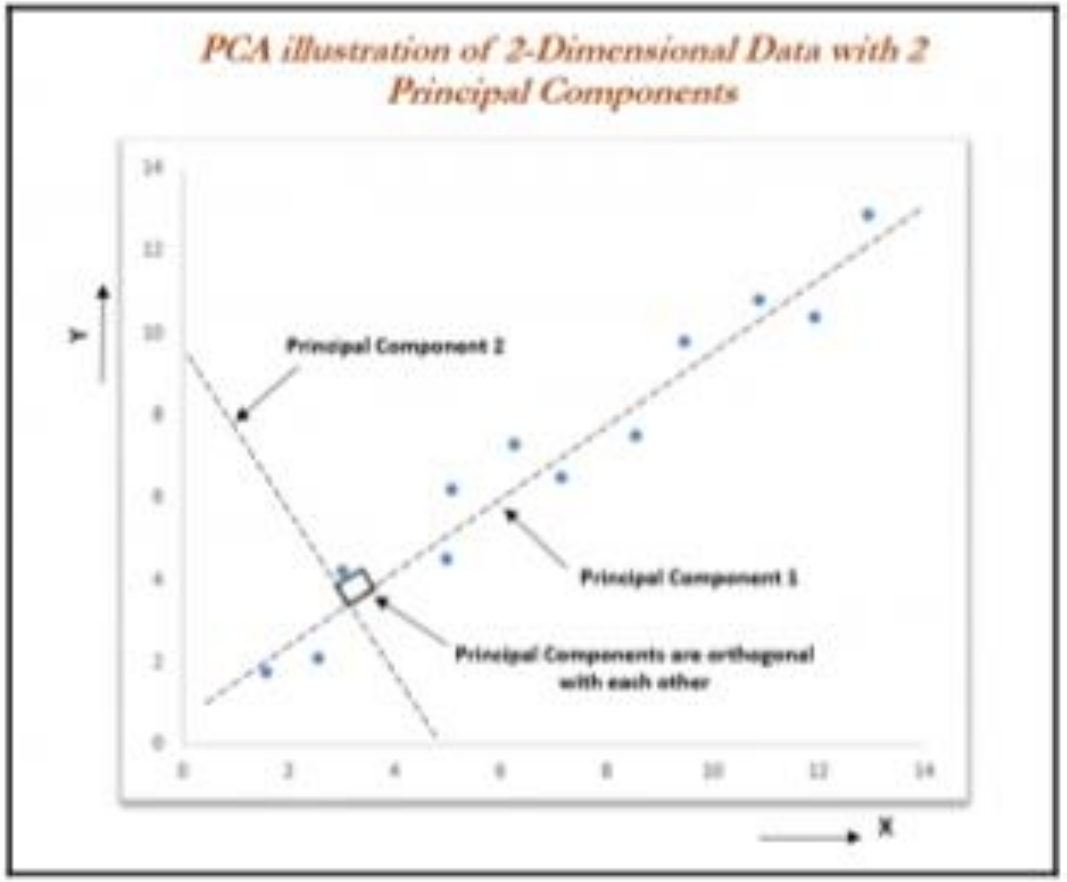
2차원 데이터 PCA예시

- 최종적으로 3개의 이미지를 모두 동일한 축으로 향하도록 방향을 바꾸면 완성된 하나의 이미지로 탄생한다.
  - 동일한 기법이 PCA분석에도 적용된다.
- 주성분 기법의 작동 원리
  - 실제 데이터가 X와 Y축으로 된 2차원 공간에 그려져 있다. 주성분은 데이터 중 가장 큰 변이를 가진 부분이다.



2개의 주성분으로 표현한 2차원 데이터 PCA예시

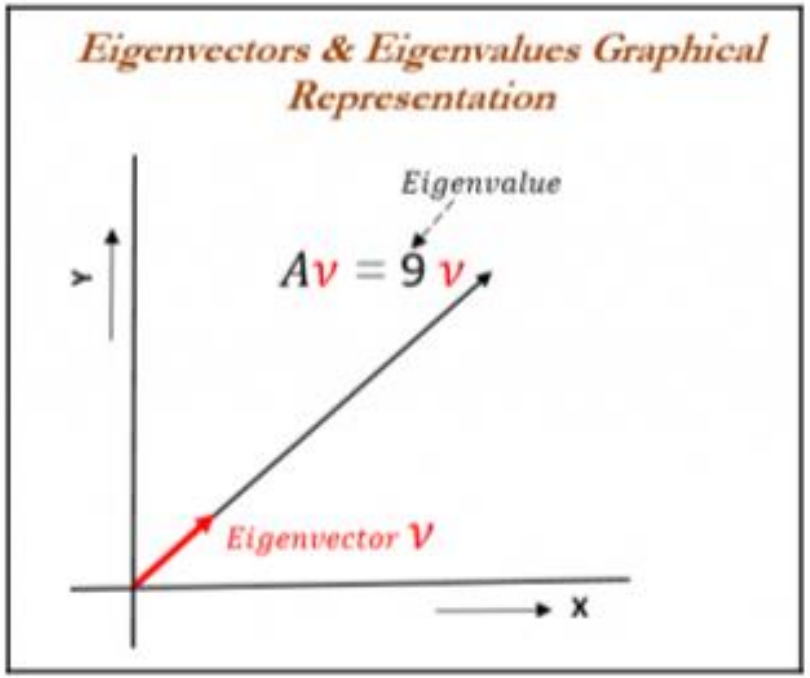
- 첫 주성분은 최대 분산을 표현
- 각 주성분들은 서로 직교해야 되므로 두 번째 주성분은 첫 번째 주성분과 직교하는 어느 지점이다.
  - 첫 번째 주성분 만으로도 데이터 전체를 잘 나타낼 수 있다.
  - 이런 성질이 데이터를 낮은 차원으로 표현할 때의 장점이다.
  - 공간이 절약되고, 데이터의 최대 분산을 찾을 수 있으며, 다음 단계의 지도학습에 활용할 수 있다.
  - 이런 점이 주성분 계산을 통해서 얻을 수 있는 핵심 장치이다.



- 고유벡터
  - 선형 변형에 관한 그 방향의 변화는 없고 단지 축소/확대/반전만 되는 축(방향)을 의미한다.
- 고유값
  - 그 변환이 일어나는 크기를 의미한다.
- 선형 변형에 관한 고유 벡터
  - 선형변형을 적용한 후에도 방향이 바뀌지 않는 0이 아닌 선형 벡터를 의미한다.

- A가 벡터 공간의 선형 변형 행렬이고, $\vec{V}$를 0이 아닌 벡터라 할 때 $A\vec{V}$가 $\vec{V}$의 스칼라 배수이면 $\vec{V}$를 A의 고유벡터라고 한다.
- $A\vec{V} = \lambda\vec{V}$
- $\vec{V}$는 고유벡터, A는 정방행렬, λ는 고유값으로 불리는 스칼라 값이다.
- 고유벡터의 방향은 A를 통해 변환 후에도 변화가 없고 고유값 만큼 크기만 변한다.
- 행렬을 자신의 고유벡터로 곱하면 그 결과는 고유 벡터를 스칼라배 한 것과 같다.
  - 이 방법은 원래 행렬을 아주 간결하게 나타낼 수 있는 방법이 된다.

2차원에서의 고유 벡터와 교유값



- 주어진 정방행렬로 부터 고유 벡터와 고유값을 계산하는 방법
  - 고유 벡터와 고유값은 정방 행렬에서만 계산 할 수 있다.
  - A $= \begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix}$
  - 정방 행렬 A와 고유벡터를 곱한 것은 고유값을 고유벡터에 곱한 것과 같다.
  - (A- λI) $\vec{V} = 0$
  - |A- λ*I| $= \begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = 0$

- 특성식에 따르면 행렬식(단위행렬과 고유값을 곱한 값과 데이터 값의 차이)은 0 값을 가져야 한다.
- $\begin{bmatrix} 2-\lambda & -4 \\ 4 & -6-\lambda \end{bmatrix}$ =(λ+2)(λ+2) =0

- 앞 행렬의 고유값은 모두 -2이다.
- 고유값을 이용해 식의 고유 벡터를 치환 할 수 있다.
- $A\vec{V} = \lambda\vec{V}$
- (A- λI) $\vec{V}$ = 0

- $\begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \vec{V} = \begin{bmatrix} 2-\lambda & -4 \\ 4 & -6-\lambda \end{bmatrix} \vec{V}$
- $\begin{bmatrix} 2-\lambda & -4 \\ 4 & -6-\lambda \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = 0$
- 앞 식에 고유 값을 대입하면
- $\begin{bmatrix} 2-(-2) & -4 \\ 4 & -6-(-2) \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = \begin{bmatrix} 4 & -4 \\ 4 & -4 \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = 0$
- 연립방정식 푸는 문제로 바뀌었다.
- $4^* v_{1,1} - 4^* v_{1,2} = 0$
- $4^* v_{1,1} - 4^* v_{1,2} = 0$

- 수식을 살펴보면 수식을 만족하는 어떠한 고유 벡터도 해가 될 수 있으므로 복수의 해가 존재한다는 것을 알 수 있다.
- 여기서는 벡터[1,1]을 사용해 검증해본다.
- 계산 결과 해가 맞는 것으로 볼 수 있다.
- $\begin{bmatrix} 2 & -4 \\ 4 & -6 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$
- PCA는 계산을 위해 단위 고유 벡터를 사용해야 한다.
- 따라서 노름을 사용해 나누거나 고유 벡터를 정규화 해야 한다.

- 유클리디언 노름 수식
  - $\|X\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$
- 출력 벡터의 노름
  - $\left\|\begin{bmatrix}1\\1\end{bmatrix}\right\| = \sqrt{1^2 + 1^2} = 0.7071$
- 단위 고유 벡터
  - $\begin{bmatrix}1\\1\end{bmatrix}/\sqrt{2} = \begin{bmatrix}07071\\0.7071\end{bmatrix}$

노름 (norm)

수의 크기를 의미하는 절댓값을 벡터공간으로 추상화한 개념을 노름이라고 한다. 두 점 사이의 거리를 절댓값을 이용하여 정의할 수 있듯이, 두 점 사이의 거리의 추상화한 개념인 거리(metric)함수를 노름을 이용하여 정의할 수 있다.

기본 원리로 된 PCA 작동 기법

- 2차원 점으로 구성된 예제를 통해 PCA 작동 방법을 설명한다.
- 목표는 2차원 데이터를 1차원(주성분)으로 축소하는 것이다.

점	X	Y
1	0.72	0.13
2	0.18	0.23
3	2.50	2.30
4	0.45	0.16
5	0.04	0.44
6	0.13	0.24
7	0.30	0.03
8	2.65	2.10
9	0.91	0.91
10	0.46	0.32
열 평균	0.83	0.69

- 첫 단계
  - 분석에 들어가기에 앞서 모든 관측 값으로부터 평균을 차감하는 것이다.
  - 이 방법은 변수의 크기 요소를 없애 차원에 관해 보다 균일한 값이 되도록 만들어준다.

점	X	Y
1	0.72-0.83 =-0.12	0.13-0.69 =-0.55
2	0.18	0.23
3	2.50	2.30
4	0.45	0.16
5	0.04	0.44
6	0.13	0.24
7	0.30	0.03
8	2.65	2.10
9	0.91	0.91
10	0.46	0.32

- 두 가지 기법을 사용해 주성분을 계산한다.
  - 데이터의 공분산 행렬

- 공분산
  - 두 변수가 함께 변하는 정도를 나타내는 척도이다. 두 변수 집합 사이의 상관관계의 강도를 측정하는 것이다.
  - 두 변수 간의 공분산이 0이라면 두 변수 사이의 상관관계는 존재하지 않는다는 결론을 내릴 수 있다.
  - 공분산을 구하는 공식
  - $COV(x,y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$
  - X와 Y 변수에 관해 공분산을 구하는 예제는 다음과 같다.
  - 전체 공분산 행렬은 2x2행렬이자 정방행렬이다.

- $COV(x, y)$=0.6972

- 공분산 행렬= C = $\begin{bmatrix} 0.91335 & 0.75969 \\ 0.75969 & 0.69702 \end{bmatrix}$

- 공분산 행렬이 정방 행렬이므로 고유 벡터와 고유값을 구할 수 있다.
- |A- λ*I|= $\begin{bmatrix} 0.91335 & 0.75969 \\ 0.75969 & 0.69702 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = 0$

- 고유값 = [1.5725 0.0378]
- 단위 고유 벡터 = $\begin{bmatrix} 0.7553 & -0.6553 \\ 0.6553 & 0.7553 \end{bmatrix}$

- 파이썬 코드를 통해 결과를 확인할 수 있다.

```
# Calculation of eigenvectors & eigenvalues
import numpy as np
w,v = np.linalg.eig(np.array([[ 0.91335 ,0.75969 ],[ 0.75969,0.69702]]))
print ("₩nEigen Values₩n",w)
print ("₩nEigen Vectors₩n",v)
```

```
C:\JIN\Anaconda3\envs\tf\python.exe C:/

Eigen Values
 [1.57253666 0.03783334]

Eigen Vectors
 [[ 0.75530088 -0.6553782 ]
 [ 0.6553782   0.75530088]]
```

- 고유 벡터와 고유 값을 구했으면 데이터를 주성분으로 투영할 수 있다.
- 2차원을 1차원으로 낮추려는 것이 가장 큰 고유값을 갖고 있는 첫 번째 고유 벡터를 첫 번 째 주성분으로 선택한다.

-0.12 -0.55 -0.45
-0.65 -0.46 -0.79
1.67 1.61 2.31
-0.38 -0.52 0.7533 -0.63
-0.80 0.6553 -0.76
-0.71
-0.53
1.82
0.07
-0.37

원시 2차원 데이터의 첫 번째 주성분을 1차원으로 사상하는 과정을 나타낸다.
고유값 1.5725를 통해 주성분은 원래 변수에 비해 57% 이상의 변이가 생겼다는 것을 알 수 있다.

scikit-learn을 활용한 필기체 숫자 인식에 PCA적용

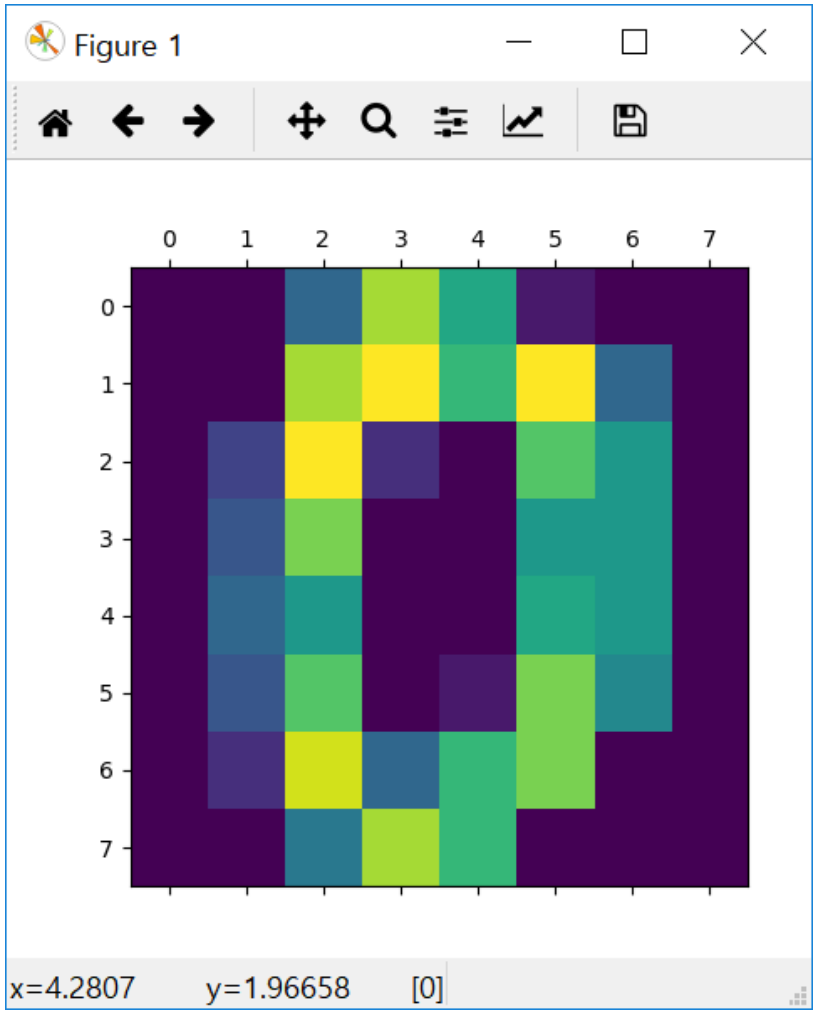
- 데이터에는 0부터 9사이의 필기체가 64개의 픽셀 강도값을 가진 특징 값들이 들어있다.
- PCA를 사용하는 기본아이디어는 64개의 특징 차원을 최소화해 보는 것이다.

```
#PCA - Principal Component Analysis
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits

digits = load_digits()
X = digits.data
y = digits.target

print (digits.data[0].reshape(8,8))
```

```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

- plt.show함수를 사용해 그래프 그려보자.

```
plt.matshow(digits.images[0])
plt.show()
```

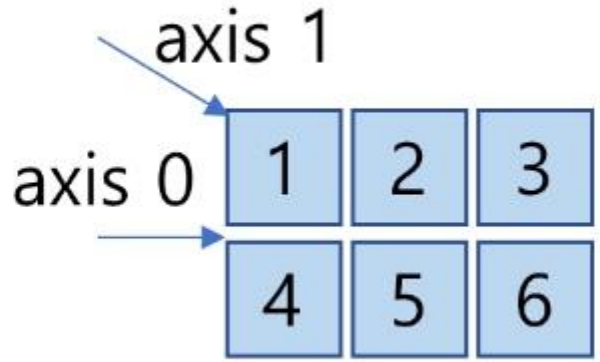


- PCA를 실행하기 전에 데이터 차원의 차이로 인한 문제를 없애기 위해 입력값의 크기를 비슷하게 조절할 필요가 있다.
- 예를 들어 고객 데이터에 PCA를 적용하는 경우 급여의 범위가 고객 나이의 범위에 비해 무척 크기 때문에 모든 변수를 비슷하게 맞추지 않으면 아주 큰 변수의 영향으로 인해 작은 값을 가진 나머지 변수의 영향을 모두 감쇠시켜 버리게 될 것이다.
- 모든 열에 관해 각각의 크기를 조절한다.

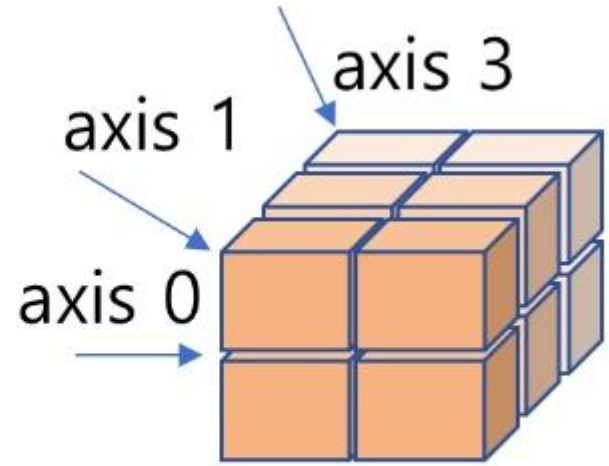
```
from sklearn.preprocessing import scale
X_scale = scale(X,axis=0)
```

- 참고
- NumPy 배열은 다차원 배열을 지원한다.
- NumPy 배열의 구조는 "**Shape**"으로 표현된다.
- Shape은 배열의 구조를 파이썬 튜플 자료형을 이용하여 정의한다.
  - 예를 들어 28X28 컬러 사진은 높이가 28, 폭이 28, 각 픽셀은 3개 채널(RGB)로 구성된 데이터 구조를 갖는다.
  - 즉 컬러 사진 데이터는 Shaep이 (28, 28, 3)인 3차원 배열이다.
  - 다차원 배열은 입체적인 데이터 구조를 가지며, 데이터의 차원은 여러 갈래의 데이터 방향을 갖는다.
  - 다차원 배열의 데이터 방향을 **axis**로 표현할 수 있다.
  - 행방향(높이), 열방향(폭), 채널 방향은 각각 axis=0, axis=1 그리고 axis=2로 지정된다.
  - NumPy 집계(Aggregation) 함수는 배열 데이터의 집계 방향을 지정하는 axis 옵션을 제공한다.

1D array

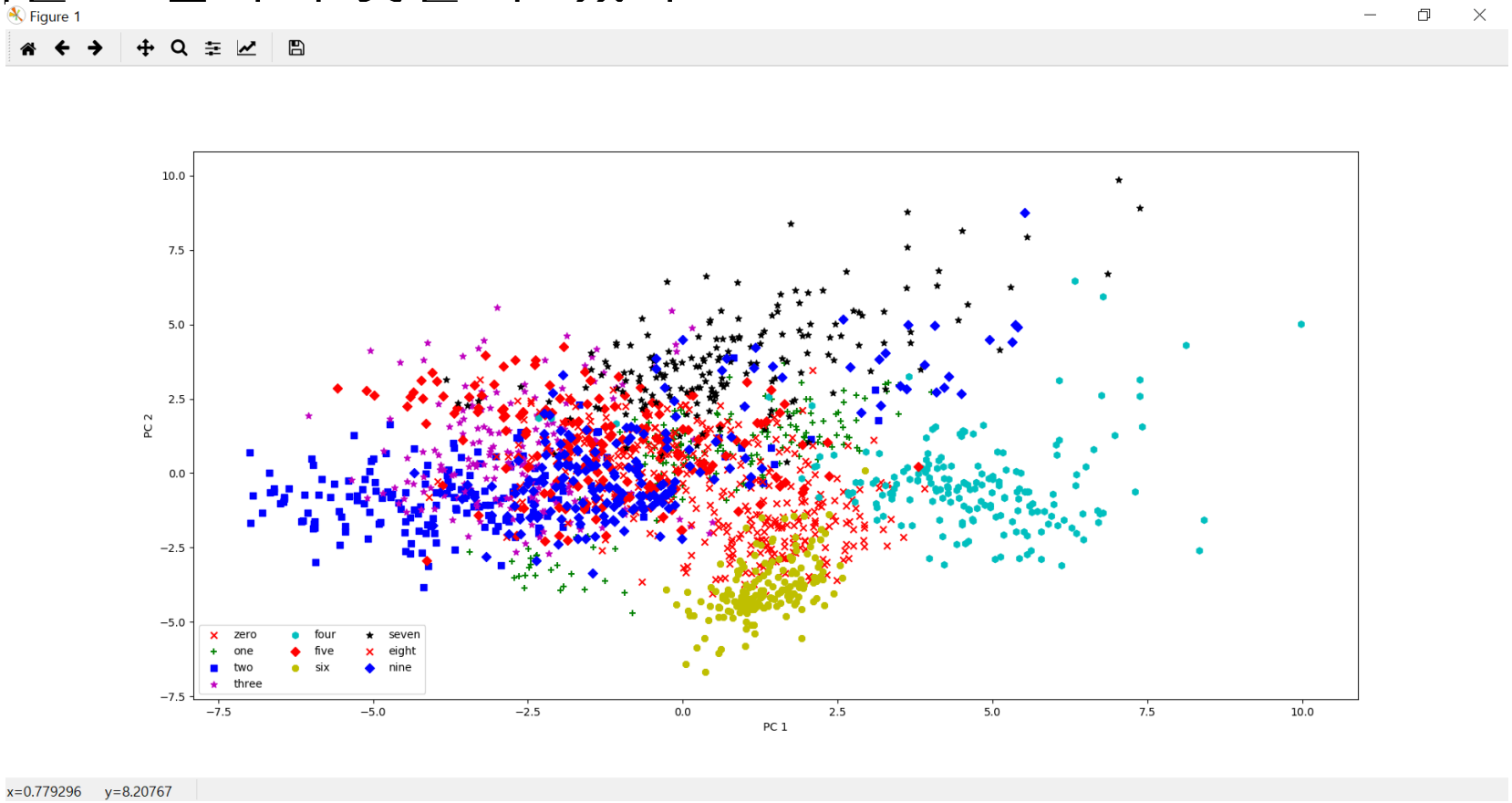


2D array

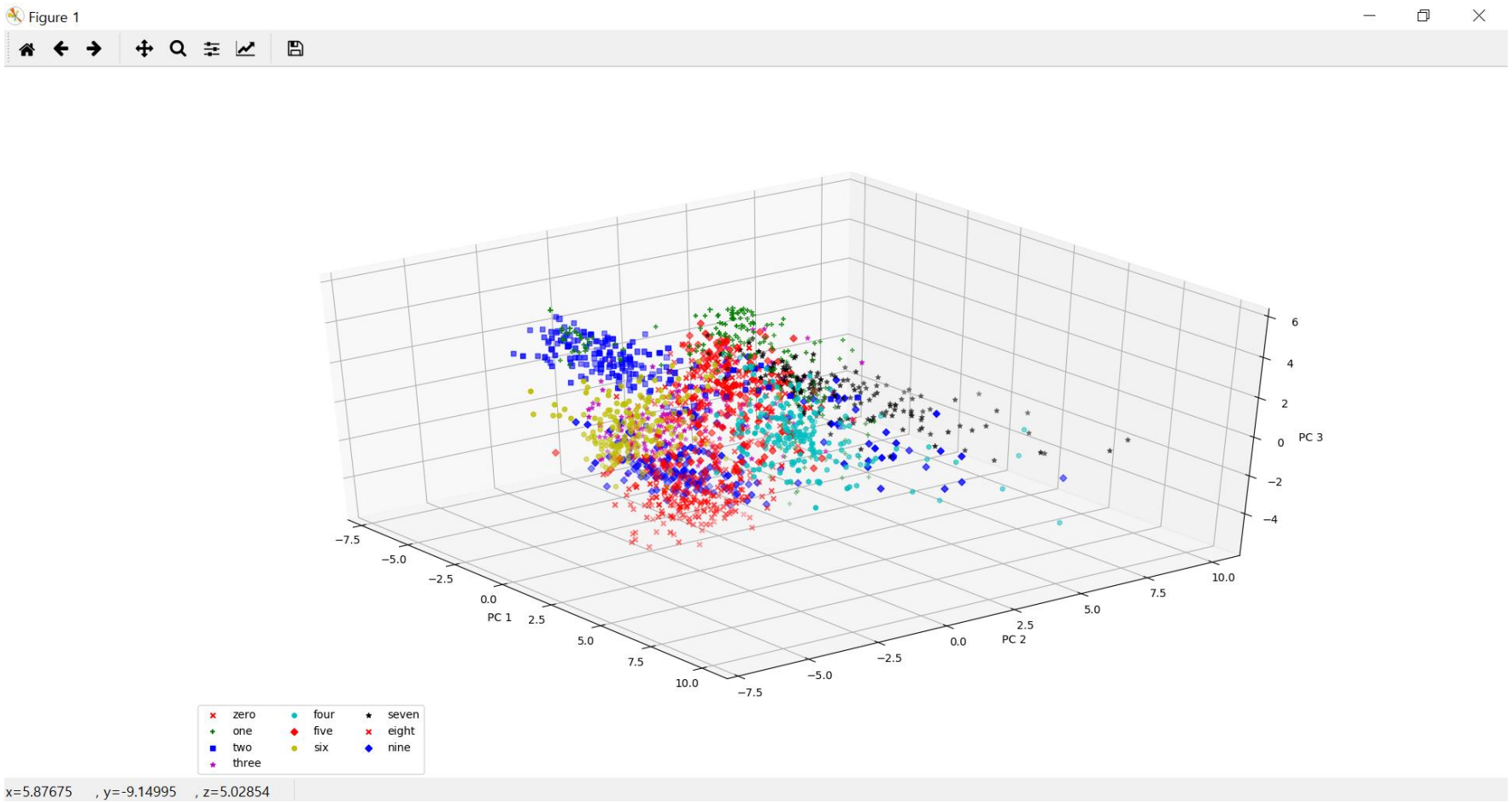


3D array

- 2개의 주성분을 활용해 2차원 그래프에서 성능을 표현한다.
  - 각 숫자의 인접한 정도와 분리된 정도를 대충 알아볼 수 있다.
  - 그래프상으로 6와 8은 매우 비슷하며, 4와 7은 중심 그룹으로부터 아주 멀리 분포하고 있다는 것 등을 알아볼 수 있다.
  - 그러나, 더 높은 차원의 PCA도 해볼 필요가 있다. 어떤 경우는 2차원만으로는 모든 변이를 표현하지 못할 수 있다.

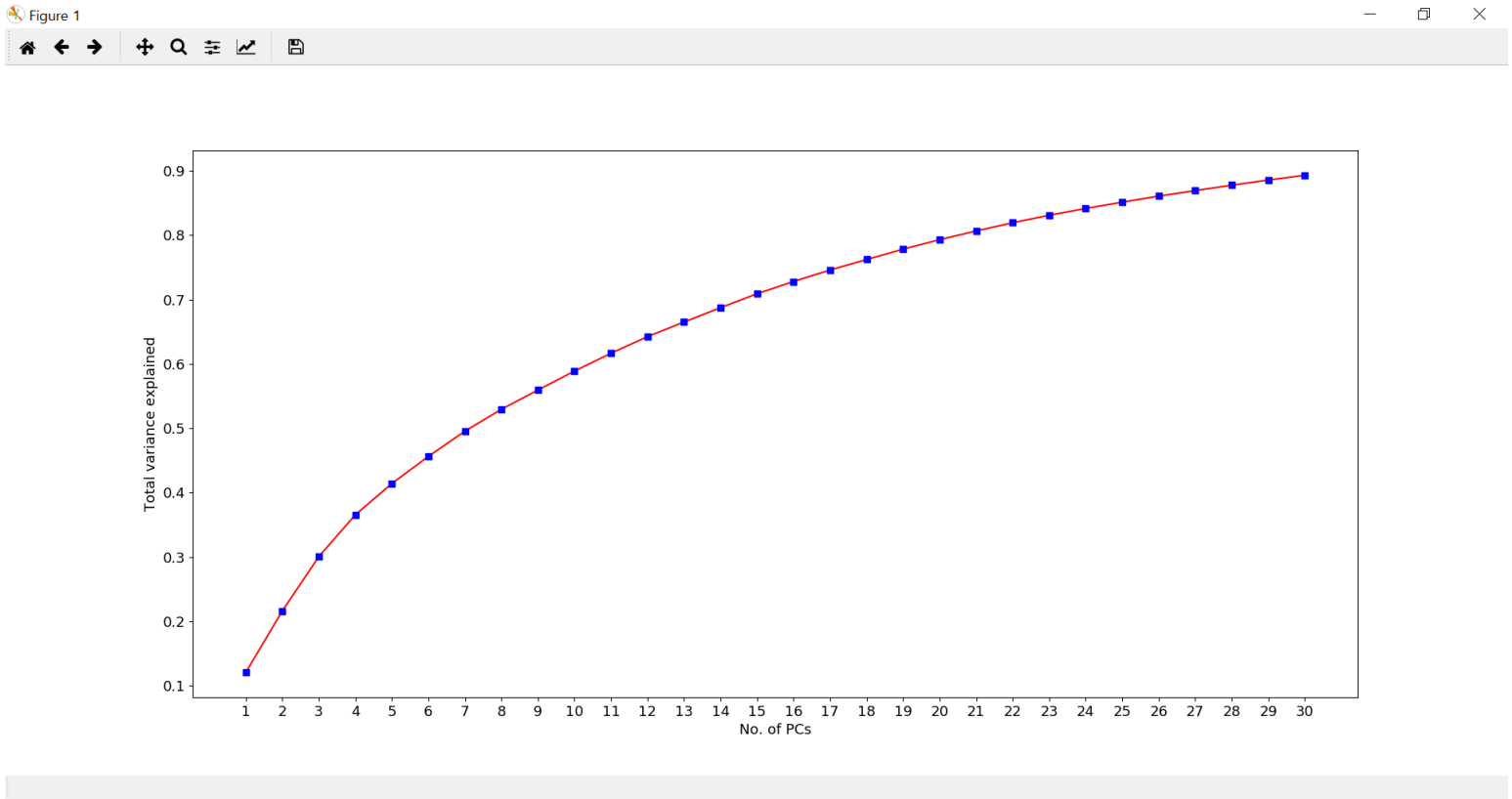


- 3개의 PCA를 적용해 3차원 공간상에 데이터를 투영해 좀 더 나은 분석을 해보자.
- 각 숫자 별로 차원 하나를 더 만들어야 하는 점을 제외하면 2개 PCA때와 아주 비슷하다.



- Matplot 장점
  - 대화형 UI를 제공하기 때문에 그림을 회전시켜 가며 다른 각도에서 관찰할 수 있다.
  - 3차원 그래프는 2차원 그래프와 유사하지만 더 많은 설명이 가능하다.
  - 숫자2는 왼쪽 극단에 치우쳐 있고, 0은 바닥에 위치해 있다.
  - 4는 오른쪽 상단에 분포하고 6은 주성분 1축에 더 가까이 분포하는 듯하다.
  - 각 숫자에 관해 그 분포를 시각적으로 파악할 수 있다.
- 몇 개의 PCA를 추출할 것인지는 비지도 러닝에 있어서는 주관식 문제와 같지만 어림할 수 있는 방법은 있다.
- 클러스터의 개수를 결정할 두 가지 기준
  - 전체 설명된 분산이 미미하게 감소하기 시작하는 시점
  - 설명된 분산이 전체의 80% 이상일 경우

- 주성분(PC)의 개수의 변화에 따라 전체 설명된 분산의 비율을 나타낸다.
- 주성분의 개수가 올라갈수록 설명된 분산의 비율도 증가할 것이다.
- 그러나 주성분의 개수는 작을수록 좋다.
  - 따라서 절충점은 분산의 설명 폭이 감소하기 시작하는 시점이 된다.



- 주성분의 개수가 10이 되는 지점에서 전체 분산의 해석 폭이 감소하기 시작하는 것을 볼 수 있다.
- 전체 분산이 80% 이상 해석된 지점은 주성분 개수가 21이 되는 지점이다.
- 해당 사업영역과 사용자에 따라 적절한 선택을 하면 된다.

PREVIEW

일상생활에서 앙상블

- 중한 병에 걸려 병원을 결정할 때
- 중요한 임무를 맡길 사람을 정할 때
- 비싼 물건을 살 때

←앙상블 방법ensemble method이란 여러 전문가로부터 얻은 복수의 의견을 적절한 방법으로 결합하여 좀 더 적합한 의사결정을 하는 접근방법

기계 학습에서의 앙상블

- 여러 예측기가 출력한 복수의 예측값을 적절한 방법으로 결합하여 더 높은 정확도의 예측값을 출력
- 두 가지 문제: 앙상블 생성과 앙상블 결합

12.1 동기와 원리

- 12.1.1 앙상블을 사용하는 이유
- 12.1.2 요소 분류기의 다양성

12.1.1 앙상블을 사용하는 이유

- 나쁜 운을 피할 수 있다.
- 성능 향상을 꾀할 수 있다.
- 데이터 양에 따른 어려움을 극복할 수 있다.
- 데이터 질에 따른 어려움을 극복할 수 있다.
- 다중 센서 시스템에 효과적이다.
- 점진 학습이 가능하다.

12.1.2 요소 분류기의 다양성

다양성의 중요성

- 틀리는 샘플이 다를수록 다양하다고 말함
- 요소 분류기가 비슷하면 앙상블 효과가 적음 (극단적으로 모두 같다면 효과 전무)

다양성 척도

- 식 (12.1)은 여러 척도 중 하나 (n은 샘플의 개수, T는 분류기의 개수, l_i는 i번째 샘플을 맞춘 분류기의 개수)

$$E = \frac{1}{n\left(T - \left\lceil \frac{T}{2} \right\rceil\right)} \sum_{i=1}^{n} \min(l_i, T - l_i) \qquad (12.1)$$

12.1.2 요소 분류기의 다양성

예제 12-1 **요소 분류기의 다양성을 측정하는 척도**

훈련 샘플 10개의 정답 레이블이 1111100000이라 가정하자. 3개 분류기 $c_1 \sim c_3$이 다음과 같이 동일한 분류 결과를 출력했다고 하자. 정확률은 모두 70%이다. 이 분류기들을 투표 방식으로 결합하면 70%의 정확률을 얻어 아무 이득이 없다. 요소 분류기의 다양성이 전혀 없는 경우이다.

$$\left.\begin{array}{l} c_1: 1011101100(0.7) \\ c_2: 1011101100(0.7) \\ c_3: 1011101100(0.7) \end{array}\right\} \Rightarrow 1011101100(0.7)$$

분류기 3개가 다음과 같이 서로 다르게 분류한 상황을 살펴보자. 이 상황에서 투표 기법으로 결합하면 100%의 정확률이 되어 30%의 성능 향상이 있다. 이와 같은 성능 향상은 요소 분류기가 다양하기 때문이다.

$$\left.\begin{array}{l} c_4: 1011101001(0.7) \\ c_5: 1110100110(0.7) \\ c_6: 0101110000(0.7) \end{array}\right\} \Rightarrow 1111100000(1.0)$$

식 (12.1)을 계산해 보자. 첫 번째 경우는 $l_1 \sim l_{10}$이 (3,0,3,3,3,3,0,0,3,3)이다. 따라서 다음과 같이 0이 된다.

$$E = \frac{1}{10 * 1}(0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0) = 0$$

두 번째 경우는 $l_1 \sim l_{10}$이 (2,2,2,2,3,2,2,2,2,2)이므로 다음과 같이 0.9이다.

$$E = \frac{1}{10 * 1}(1 + 1 + 1 + 1 + 0 + 1 + 1 + 1 + 1 + 1) = 0.9$$

12.1.2 요소 분류기의 다양성

다양한 요소 분류기를 만드는 방법

- 재샘플링resampling 방법
  - 훈련집합에서 일정 비율을 임의로 선택하는 일을 여러 번 반복
  - 배깅과 부스팅(12.2절)
- 부분 공간subspace 방법
  - 원래 특징 공간에서 일부를 임의로 선택하는 일을 여러 번 반복
  - 랜덤 포리스트(12.3절)

12.2 재샘플링resampling 기법

- 12.2.1 배깅
- 12.2.2 부스팅

12.2.1 배깅bagging

배깅

- 모델 선택 문제를 푸는 부트스트랩(1장의 [알고리즘 1-4]) 아이디어를 앙상블 생성에 적용
- 이름도 부트스트랩을 변형한 **b**ootstrap **agg**regat**ing**에서 유래
- 요소 분류기로 *k*-NN, SVM, 신경망처럼 안정적인 분류기보다 트리 분류기처럼 불안정한 분류기를 사용할 때 보다 효과적

알고리즘 12-1 배깅

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}, \mathbb{Y} = \{y_1, y_2, \cdots, y_n\}$, 샘플링 비율 p $(0 < p \leq 1)$
출력: 분류기 앙상블 $C = \{c_1, c_2, \cdots, c_T\}$

```
1  t = 0
2  C = ∅
3  repeat
4      t++
5      𝕏에서 샘플을 pn개 뽑아 𝕏′라 한다. 이때 대치를 허용한다.
6      𝕏′로 분류기 c_t를 학습한다.
7      C = C∪c_t
8  until(멈춤 조건)
9  T = t
```

12.2.2 부스팅boosting

부스팅

- 배깅은 가능한 한 분류기가 독립적이게 만듦 ⇔ 부스팅은 서로 연관성을 가지도록 만듦
- 부스팅은 t번째 분류기 c_t가 맞춘 샘플과 틀린 샘플로 구분 → 맞춘 샘플은 인식이 가능하므로 가중치를 낮추고 틀린 샘플은 여전히 까다로운 상대이므로 가중치를 높임
- 다음 분류기 c_{t+1}은 가중치가 높은 샘플을 잘 맞히도록 학습
- 메타 알고리즘이므로 구체적인 구현 방법은 여럿
- AdaBoost가 가장 널리 쓰임

12.2.2 부스팅

알고리즘 12-2 AdaBoost

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}, \mathbb{Y} = \{y_1, y_2, \cdots, y_n\}$
출력: 분류기 앙상블 $C = \{c_1, c_2, \cdots, c_T\}$, 분류기 신뢰도 $\alpha_t, t = 1,2, \cdots, T$

```
1  t = 1
2  C = ∅
3  for (i = 1 to n) w_ti = 1/n   // 처음에는 같은 가중치를 가지고 출발
4  repeat
5      w_t1~w_tn을 고려하여 분류기 c_t를 학습한다.   // 가중치가 큰 샘플을 더 중요하게 다룸
6      ε_t = 0    // 오류율
7      for (i = 1 to n)
8          if (c_t(x_i) ≠ y_i) ε_t = ε_t + w_ti    // c_t가 틀린 샘플의 가중치 합을 오류율로 취함
9      if (ε_t < 0.5)    // 임의 짐작보다 나은 분류기만 취함
10         α_t = 1/2 log((1−ε_t)/ε_t)    // c_t의 신뢰도
11         for (i = 1 to n)
12             if (c_t(x_i) ≠ y_i) w_t+1,i = w_ti * exp(α_t)    // 틀린 샘플의 가중치 높임
13             else w_t+1,i = w_ti * exp(−α_t)    // 맞힌 샘플의 가중치 낮춤
14         for (i = 1 to n) w_t+1,i = w_t+1,i / Σ_j=1,n w_t+1,j    // 가중치 정규화
15         C = C∪c_t
16         t++
17 until (멈춤조건)
18 T = t − 1
```

12.2.2 부스팅

[알고리즘 12-2] 부연 설명

- 가중치 관리
  - 라인 3에서 동일 가중치를 부여하고 출발
  - 라인 11~13은 맞힌 샘플은 가중치를 줄이고, 틀린 샘플은 높임
- 가중치를 고려한 학습
  - 재샘플링을 이용한 방법

5. // w_t를 고려하여 분류기 c_t를 학습한다.
   5.1 $\mathbb{X}$에서 qn개의 샘플을 대치를 허용하지 않고 샘플링하여 $\mathbb{X}'$를 구성한다. 이때 $\mathbf{x}_i$가 뽑힐 확률을 w_{ti}가 되도록 한다.
   5.2 $\mathbb{X}'$로 c_t를 학습한다.

  - 목적함수를 다시 정의하는 방법

$$J = \sum_{i=1}^{n} w_{ti} \| c_t(\mathbf{x}_i) - y_i \|_2^2 \qquad (12.2)$$

- 라인 6~8의 오류율 계산에서도 가중치 고려
- 라인 10은 요소 분류기의 신뢰도 계산

12.3 결정 트리와 랜덤 포리스트

- 12.3.1 결정 트리
- 12.3.2 랜덤 포리스트

12.3.1 결정 트리decision tree

- 결정 트리는 스무고개와 비슷한 원리
  - 스무고개 예) 생물인가요? $\xrightarrow{예}$ 식물인가요? $\xrightarrow{아니오}$ 뭍에 사나요? $\xrightarrow{아니오}$ 바다에 사나요? $\xrightarrow{예}$ 영리한가요? $\xrightarrow{예}$ 답은 돌고래죠? $\xrightarrow{예}$ 끝
  - 결정 트리는 알고리즘이 질문을 만들어야 함 (질문은 훈련집합을 이용하여 만듦)
- 노드에서의 질문
  - $x_i < \epsilon$?은 어느 쪽으로 분기할지 결정하는 데 사용하는 질문

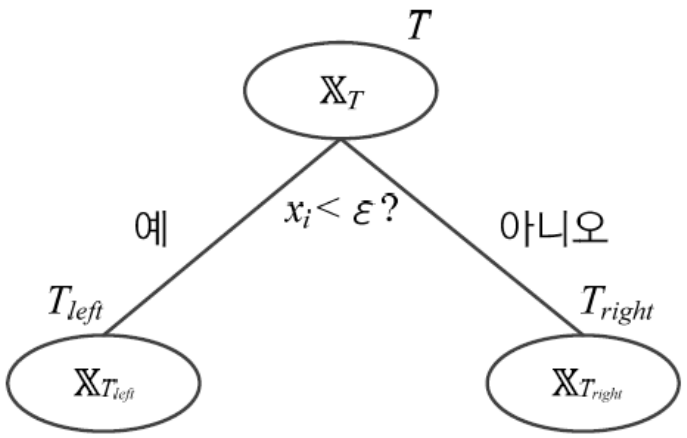


그림 12-1 노드의 분기

12.3.1 결정 트리

- 질문을 만드는 과정
  - ①가능한 모든 후보 질문 생성 → ②최적 질문 선택
  - 훈련집합을 이용함
- 후보 질문 생성 [예제 12-2]

예제 12-2 **후보 질문 생성**

온라인 쇼핑몰이 고객 정보를 3차원 특징 벡터로 표현한다고 하자. x_1은 직업, x_2는 선호 품목, x_3은 구매 성적이며, 값은 다음과 같다.

$$x_1 \in \{1(\text{디자이너}), 2(\text{스포츠맨}), 3(\text{교수}), 4(\text{의사}), 5(\text{공무원}), 6(\text{NGO}), 7(\text{무직})\}$$
$$x_2 \in \{1(\text{의류}), 2(\text{전자제품}), 3(\text{스포츠용품}), 4(\text{책}), 5(\text{음식})\}$$
$$x_3 \in \text{실수}$$

노드 T의 샘플집합 $\mathbb{X}_T$와 $\mathbb{Y}_T$가 다음과 같다고 하자.

$$\mathbb{X}_T = \{\mathbf{x}_1 = (3,1,50.6)^T,\ \mathbf{x}_2 = (2,3,72.8)^T,\ \mathbf{x}_3 = (3,5,88.7)^T,$$
$$\mathbf{x}_4 = (2,2,102.2)^T,\ \mathbf{x}_5 = (5,5,92.3)^T,\ \mathbf{x}_6 = (3,4,65.3)^T,$$
$$\mathbf{x}_7 = (2,3,67.8)^T,\ \mathbf{x}_8 = (7,1,47.8)^T,\ \mathbf{x}_9 = (2,3,45.6)^T\}$$
$$\mathbb{Y}_T = \{y_1 = 2,\ y_2 = 1,\ y_3 = 3,\ y_4 = 2,\ y_5 = 2,\ y_6 = 2,\ y_7 = 1,\ y_8 = 3,\ y_9 = 1\}$$

12.3.1 결정 트리

x_1과 x_2로 다음과 같이 후보 질문 9개가 생성된다.

x_1에 따른 후보 질문: $x_1=2?, x_1=3?, x_1=5?, x_1=7?$
x_2에 따른 후보 질문: $x_2=1?, x_2=2?, x_2=3?, x_2=4?, x_2=5?$

x_3은 실수이므로 부등식 질문을 만들어야 한다. 보통 훈련 샘플을 순서대로 나열한 다음 인접한 두 샘플의 중간을 임곗값으로 사용한다. x_3을 나열하면 45.6, 47.8, 50.6, 65.3, 67.8, 72.8, 88.7, 92.3, 102.20이다. 따라서 후보 질문은 다음과 같다.

x_3에 따른 후보 질문: $x_3<46.7?, x_3<49.2?, x_3<57.95?, x_3<66.55?, x_3<70.3?, x_3<80.75?, x_3<90.5?, x_3<97.25?$

17개의 후보 질문이 생성되었다.

12.3.1 결정 트리

■ 최적 질문을 선택하는 데 사용할 불순도

- 불순도impurity는 레이블 정보 $\mathbb{Y}_T$를 가지고 측정
- 세 가지 불순도 척도
  - $P(k|\mathbb{Y}_T)$는 $\mathbb{Y}_T$에서 부류 k가 발생할 확률 ($\mathbb{Y}_T$에서 부류 k의 샘플 수를 $\mathbb{Y}_T$의 전체 샘플 수로 나누어 계산)

$$\text{entropy}(\mathbb{Y}_T) = -\sum_{k=1}^{c} P(k|\mathbb{Y}_T)\log_2 P(k|\mathbb{Y}_T) \tag{12.3}$$

$$\text{gini}(\mathbb{Y}_T) = 1 - \sum_{k=1}^{c} P(k|\mathbb{Y}_T)^2 = \sum_{j \neq k} P(j|\mathbb{Y}_T)P(k|\mathbb{Y}_T) \tag{12.4}$$

$$\text{misclassification}(\mathbb{Y}_T) = 1 - \max_k P(k|\mathbb{Y}_T) \tag{12.5}$$

12.3.1 결정 트리

예제 12-3 **불순도 측정**

[예제 12-2]의 훈련집합을 다시 사용하자. 편의상 $\mathbb{Y}_T$를 다시 쓴다. 세 부류의 발생 확률은 다음과 같다.

$$\mathbb{Y}_T = \{y_1 = 2, y_2 = 1, y_3 = 3, y_4 = 2, y_5 = 2, y_6 = 2, y_7 = 1, y_8 = 3, y_9 = 1\}$$
$$P(1|\mathbb{Y}_T) = \frac{3}{9},\ P(2|\mathbb{Y}_T) = \frac{4}{9},\ P(3|\mathbb{Y}_T) = \frac{2}{9}$$

식 (12.3)~(12.5)의 엔트로피, 지니, 오분류 불순도를 계산하면 다음과 같다.

$$\text{entropy}(\mathbb{Y}_T) = -\left(\frac{3}{9}\log_2\frac{3}{9} + \frac{4}{9}\log_2\frac{4}{9} + \frac{2}{9}\log_2\frac{2}{9}\right) = 1.5305$$
$$\text{gini}(\mathbb{Y}_T) = 1 - \left(\left(\frac{3}{9}\right)^2 + \left(\frac{4}{9}\right)^2 + \left(\frac{2}{9}\right)^2\right) = 0.6420$$
$$\text{misclassification}(\mathbb{Y}_T) = 1 - \frac{4}{9} = 0.5556$$

12.3.1 결정 트리

■ 최적 질문 선택

- 질문은 $\mathbb{X}_T$를 $\mathbb{X}_{Tleft}$와 $\mathbb{X}_{Tright}$로 나누는데 ([그림 12-1]), 불순도를 많이 낮출수록 좋은 질문임
- 불순도를 낮추는 정도는 불순도 감소량으로 측정

$$delta\_im(q) = im(\mathbb{Y}_T) - \left(\frac{\left|\mathbb{Y}_{T_{left}}\right|}{|\mathbb{Y}_T|} im\left(\mathbb{Y}_{T_{left}}\right) + \frac{\left|\mathbb{Y}_{T_{right}}\right|}{|\mathbb{Y}_T|} im\left(\mathbb{Y}_{T_{right}}\right) \right) \tag{12.6}$$

$$twoing(q) = \frac{\left|\mathbb{Y}_{T_{left}}\right|}{|\mathbb{Y}_T|} \frac{\left|\mathbb{Y}_{T_{right}}\right|}{|\mathbb{Y}_T|} \left(\sum_{k=1}^{c} \left| P\left(k \middle| \mathbb{Y}_{T_{left}}\right) - P(k|\mathbb{Y}_{T_{right}}) \right| \right)^2 \tag{12.7}$$

12.3.1 결정 트리

예제 12-4 **불순도 감소량**

[예제 12-2]의 훈련집합을 사용한다. [그림 12-2]는 [예제 12-2]에서 생성한 17개 질문 중 x_2=3?에 대한 분기 결과를 보여 준다. 이 질문은 "선호 품목이 스포츠용품인가?"에 해당한다.

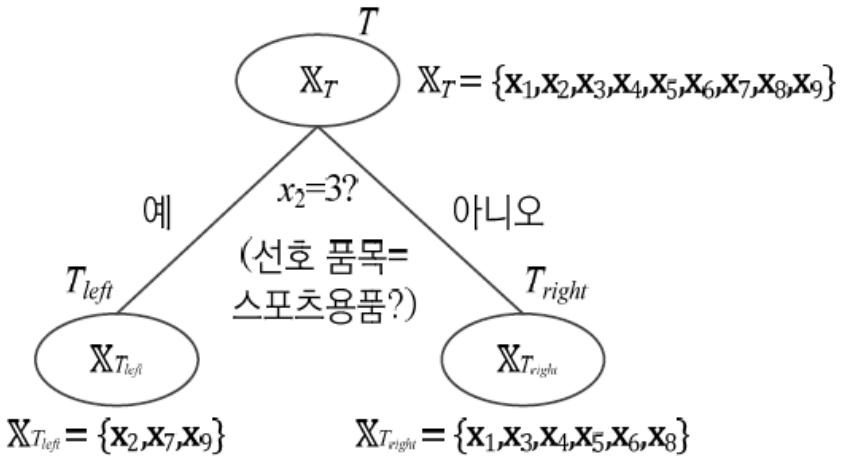


그림 12-2 질문에 따른 분기 결과

식 (12.6)으로 이 질문의 불순도 감소량을 계산하자. 불순도 계산은 식 (12.4)의 지니 불순도를 사용한다.

$$delta\_im("x_2 = 3?") = 0.6420 - \left(\frac{3}{9} * 0.0 + \frac{6}{9} * 0.4444\right) = 0.3457$$

12.3.1 결정 트리

■ 학습 알고리즘

알고리즘 12-3 결정 트리 학습

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}, \mathbb{Y} = \{y_1, y_2, \cdots, y_n\}$
출력: 결정 트리 R

```
1  노드 하나를 생성하고 R이라 한다.
2  SplitNode(R, 𝕏)

3  function SplitNode(T, 𝕏_T)
4      if (𝕏_T가 멈춤 조건을 만족)
5          T를 잎 노드로 설정하고 부류 또는 부류 확률분포를 할당한다.
6      else
7          T에서 후보 질문을 생성한다.
8          모든 후보 질문의 불순도 감소량을 측정한다.    // 식 (12.6) 또는 식 (12.7) 사용
9          불순도 감소량이 최대인 후보 질문 q̂를 선택한다.
10         q̂로 𝕏_T를 𝕏_Tleft와 𝕏_Tright로 나눈다.
11         새로운 노드 T_left와 T_right를 생성한다.
12         T.lchild = T_left
13         T.rchild = T_right
14         SplitNode(T_left, 𝕏_Tleft)
15         SplitNode(T_right, 𝕏_Tright)
16 end function
```

12.3.1 결정 트리

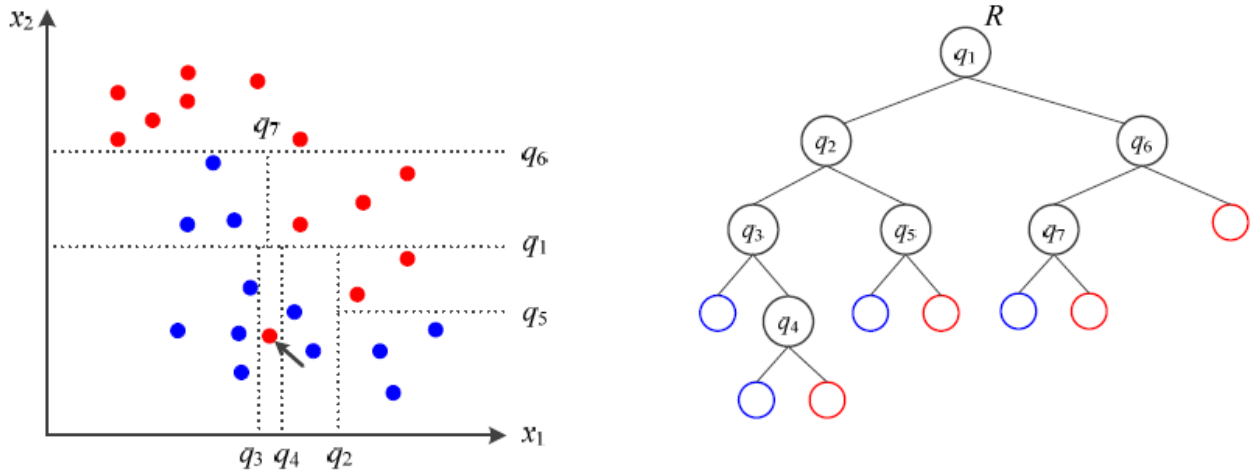
[알고리즘 12-3]의 부연 설명

라인 4의 멈춤 조건

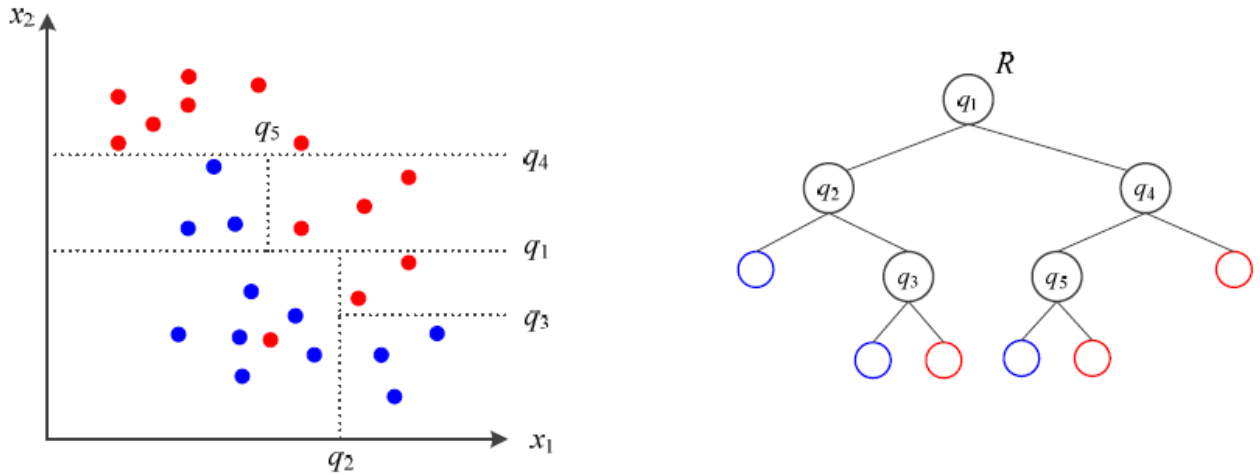
- $\mathbb{X}_T$에 속한 샘플이 모두 같은 부류이면 T를 잎 노드로 만들고 멈춤 → 이처럼 완전 동질일 때만 멈추면 과잉적합([그림 12-3(a)]의 예시)
- [그림 12-3(b)]는 멈춤 조건을 완화하여 과잉적합 회피(다음 조건을 and 또는 or로 결합)

조건 1: 불순도가 0이다.
조건 2: $\mathbb{X}_T$의 샘플 개수가 임곗값 이하이다.
조건 3: 라인 9가 결정한 $\hat{q}$의 불순도 감소량이 임곗값 이하이다.

12.3.1 결정 트리



(a) 과잉적합 상황



(b) 과잉적합 회피

그림 12-3 결정 트리의 공간 분할과 과잉적합 현상

12.3.1 결정 트리

[알고리즘 12-3]의 부연 설명

- 잎 노드 T에 부류 정보를 할당하는 일(라인 5)
  - $\mathbb{X}_T$의 샘플 부류를 조사하여 빈도가 가장 높은 부류로 결정
  - 또는 빈도수에 따라 확률분포 부여(예를 들어, [그림 12-3(b)]에 있는 q_2의 왼쪽 자식에게 $P(\text{파랑부류}) = \frac{6}{7}, P(\text{빨강부류}) = \frac{1}{7}$을 부여)
- 여러 가지 성능 향상 기법들
  - 트기 크기를 줄여주는 가지치기
  - 손실 특징을 처리하는 대리 분기
  - 여러 특징을 결합하여 질문을 만드는 기법 등

12.3.1 결정 트리

■ 예측 알고리즘

- 노드에서의 질문이 단순한 연산이고 질문의 개수가 트리의 최대 깊이를 넘지 않기 때문에 무척 빠르게 작동함

알고리즘 12–4 결정 트리를 이용한 예측

입력: 결정 트리 R, 테스트 샘플 **x**
출력: 샘플 **x**의 부류 k

```
1  T = R
2  while (T ≠ Nil)
3      x로 T의 질문을 계산하고 결과를 r이라 한다.
4      if (r = 예) T = T의 왼쪽 자식 노드
5      else T = T의 오른쪽 자식 노드
6      if (T가 잎 노드)
7          k = T의 부류
8          T = Nil
```

12.3.1 결정 트리

■ 결정 트리의 특성

- 계량 데이터뿐 아니라 비계량 데이터도 다룰 수 있음
- 분류 결과의 해석 가능성
  - 왜 이런 분류 결과를 냈는지 사용자에게 설명할 수 있음(예, 대출 거부 판정에 대한 이유를 설명)
- 불안정한 분류기
  - 일반적으로 단점이지만, 배깅에서는 유용한 특성으로 작용함

12.3.2 랜덤 포리스트random forest

- 결정 트리의 낮은 성능
  - 신경망이나 SVM 등에 비해 정확도가 낮다는 결정적인 약점 → 주로 앙상블의 요소 분류기로 사용
  - 랜덤 포리스트라는 앙상블 기법이 가장 널리 사용됨
- 랜덤 포리스트
  - 결정 트리를 배깅으로 결합

알고리즘 12-5 랜덤 포리스트

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, $\mathbb{Y} = \{y_1, y_2, \cdots, y_n\}$, 샘플링 비율 p $(0 < p \leq 1)$
출력: 분류기 앙상블 $C = \{c_1, c_2, \cdots, c_T\}$

```
1  t=0
2  C = ∅
3  repeat
4      t++
5      𝕏에서 pn개의 샘플을 뽑아 𝕏′라 한다. 이때 대치를 허용한다.
6      𝕏′를 훈련집합으로 하고 [알고리즘 12-3]의 '수정된' 버전으로 결정 트리 c_t를 학습한다.
7      C = C∪c_t
8  until (멈춤 조건)
9  T=t
```

12.3.2 랜덤 포리스트

[알고리즘 12-5]의 부연 설명

- 라인 5에서 임의 샘플링하므로 요소 분류기는 독립성을 가짐
- 라인 6은 독립성을 강화하려고 [알고리즘 12-3]의 수정 버전을 사용
  - 결정 트리의 성능을 희생하는 대신 앙상블의 독립성을 강화

7. T에서 후보 질문을 생성한다.
8. 모든 후보 질문의 불순도 감소량을 측정한다. // 식 (12.6) 또는 식 (12.7) 사용
9. 불순도 감소량이 최대인 후보 질문 $\hat{q}$를 선택한다.

⇓

7. 특징 벡터에서 임의로 특징 하나를 뽑는다. 뽑힌 특징을 x_j라 한다.
8. x_j의 최적 임곗값을 찾아 질문 $\hat{q}$를 만든다.

- 특징 각각의 중요도까지 측정 가능 → 특징 분석 또는 선택에 사용

12.3.2 랜덤 포리스트

- 랜덤 포리스트의 성공적인 활용 사례
  - 키넥트 센서로 획득한 RGBD 영상을 분석하여 사람의 자세를 추정하는 응용
    - 깊이 영상을 신체 부위로 분할하는 단계에 활용

그림 12-4 랜덤 포리스트를 이용한 신체 부위 분할

- 랜덤 포리스트와 딥러닝의 결합 → 12.5절

12.4 앙상블 결합

- 12.4.1 부류 레이블
- 12.4.2 부류 순위
- 12.4.3 부류 확률

- 예제
  - 부류 레이블: $(0,1,0)^T$는 두 번째 부류를 뜻함
  - 부류 순위: $(3,1,2)^T$는 두 번째 부류일 가능성이 가장 크고 그 다음으로 세 번째와 첫 번째 임을 뜻함
  - 부류 확률: $(0.1,0.7,0.2)^T$는 부류별 발생 확률을 제공

12.4.1 부류 레이블

- 유권자가 가장 선호하는 후보 1명에게 투표하는 방식에 해당
  - 분류기가 T개일 때, k번째 분류기의 출력을 $\mathbf{o}^{(k)} = \left(o_1^{(k)}, o_2^{(k)}, \cdots, o_c^{(k)}\right)^{\mathrm{T}}$라고 표기
  - 최다 득표majority voting 방식

$$\hat{\imath} = \underset{i}{\operatorname{argmax}} \sum_{k=1}^{T} o_i^{(k)} \qquad (12.8)$$

  - 보다 보수적인 방식으로서 과반 득표, 만장일치도 있음
  - 최다 득표, 과반 득표, 만장일치는 오른쪽으로 갈수록 보수적이며, 정확률이 높아지는 대신 기각률도 따라 높아짐

- 가중치 투표weighted voting
  - 분류기의 신뢰도(α_k는 k번째 분류기의 신뢰도)가 주어지는 경우(예, AdaBoost) 적용 가능 (명성이 높은 사람에게 더 큰 비중을 두는 방식에 비유)

$$\hat{\imath} = \underset{i}{\operatorname{argmax}} \sum_{k=1}^{T} \alpha_k o_i^{(k)} \qquad (12.9)$$

12.4.1 부류 레이블

예제 12–5 **투표와 가중 투표**

부류의 개수 $c=3$이고 분류기의 개수 $T=5$라고 하자. 분류기 신뢰도는 $\boldsymbol{\alpha} = (0.1,0.4,0.2,0.1,0.2)^{\mathrm{T}}$라 가정한다. 분류기 5개의 출력은 다음과 같다.

$$\mathbf{o}^{(1)} = (0,1,0)^{\mathrm{T}},\ \mathbf{o}^{(2)} = (0,0,1)^{\mathrm{T}},\ \mathbf{o}^{(3)} = (0,1,0)^{\mathrm{T}},\ \mathbf{o}^{(4)} = (0,1,0)^{\mathrm{T}},\ \mathbf{o}^{(5)} = (0,0,1)^{\mathrm{T}}$$

$$\Rightarrow \begin{cases} \text{투표: } \left(\sum_{k=1}^{T} o_1^{(k)}, \sum_{k=1}^{T} o_2^{(k)}, \sum_{k=1}^{T} o_3^{(k)}\right) = (0,3,2)^{\mathrm{T}} \rightarrow \hat{\imath} = 2 \\ \text{가중치 투표: } \left(\sum_{k=1}^{T} \alpha_k o_1^{(k)}, \sum_{k=1}^{T} \alpha_k o_2^{(k)}, \sum_{k=1}^{T} \alpha_k o_3^{(k)}\right) = (0,0.4,0.6)^{\mathrm{T}} \rightarrow \hat{\imath} = 3 \end{cases}$$

투표 방식에서는 두 번째 부류가 최다 득표했으므로 2번 부류로 분류한다. 하지만 가중 투표에서는 세 번째 부류가 가장 큰 값을 얻었으므로, 세 번째 부류로 분류한다. 신뢰도가 높은 두 번째 분류기가 세 번째 부류로 분류한 탓이다.

12.4.2 부류 순위

- 유권자가 c명의 후보에게 선호도에 따라 순서를 표기하는 방식에 해당
  - 주로 보르다 계수Borda count를 사용하여 결합($s_i^{(k)}$는 k번째 부류의 i번째 부류의 점수)

$$\hat{\imath} = \underset{i}{\operatorname{argmax}} \sum_{k=1}^{T} s_i^{(k)} \qquad (12.10)$$

  - 점수 벡터는 다음 두 가지 규칙 중 하나를 사용하여 순위 벡터로부터 구함
    - r 순위에 $c - r$점을 부여
    - 또는 r 순위에 $\frac{1}{r}$점을 부여
    - 예) 순위 벡터가 $(3,1,2)^{\mathrm{T}}$일 때, 첫 번째 방법은 $(0,2,1)^{\mathrm{T}}$, 두 번째 방법은 $(0.333,1.0,0.5)^{\mathrm{T}}$가 됨 ← 순위가 높은 부류에는 두 번째 방법이 유리
  - 보르다 계수는 일상생활에서도 많이 사용
    - 슬로베니아 등의 국가에서 국회의원 선출할 때
    - 유로비전 송 콘테스트 등

12.4.3 부류 확률

- 부류 확률은 가장 일반적인 형태의 정보
  - 단순한 방법은 부류 순위나 부류 레이블로 바꿔서 결합
  - 더 많은 정보를 내포하므로 그대로 활용함이 유리
  - 여러 가지 결합 규칙

$$\text{합: } \beta_i = \sum_{k=1}^{T} o_i^{(k)} \text{ 또는 평균: } \beta_i = \frac{1}{T}\sum_{k=1}^{T} o_i^{(k)} \tag{12.11}$$

$$\text{가중 합: } \beta_i = \sum_{k=1}^{T} \alpha_k o_i^{(k)} \tag{12.12}$$

$$\text{최대: } \beta_i = \max_k o_i^{(k)} \tag{12.13}$$

$$\text{최소: } \beta_i = \min_k o_i^{(k)} \tag{12.14}$$

$$\text{메디안: } \beta_i = \text{median}_k\, o_i^{(k)} \tag{12.15}$$

$$\text{곱: } \beta_i = \prod_{k=1}^{T} o_i^{(k)} \tag{12.16}$$

12.4.3 부류 확률

- 결합 규칙으로 $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_c)^{\mathrm{T}}$를 구한 후 다음 식으로 부류 결정

$$\hat{\imath} = \underset{i}{\operatorname{argmax}}\, \beta_i \qquad (12.17)$$

  - 보통 평균 규칙을 많이 사용하며, 분류기의 신뢰도가 있는 경우에는 가중 합 규칙을 주로 사용

12.5 딥러닝과 앙상블

- 12.5.1 평균 기법을 이용한 앙상블
- 12.5.2 암시적 앙상블
- 12.5.3 깊은 랜덤 포리스트

12.5.1 평균 기법을 이용한 앙상블

- AlexNet의 앙상블(2012년 ILSVRC 대회 우승)
  - 잘라내기와 반전을 통해 10장을 만들고, 식 (12.11)의 평균 결합 적용 ← 1차 앙상블
  - 서로 다른 매개변수를 가진 5개의 AlexNet에 식 (12.11)의 평균 결합 적용 ← 2차 앙상블 (18.2% 오류율 → 16.4%)

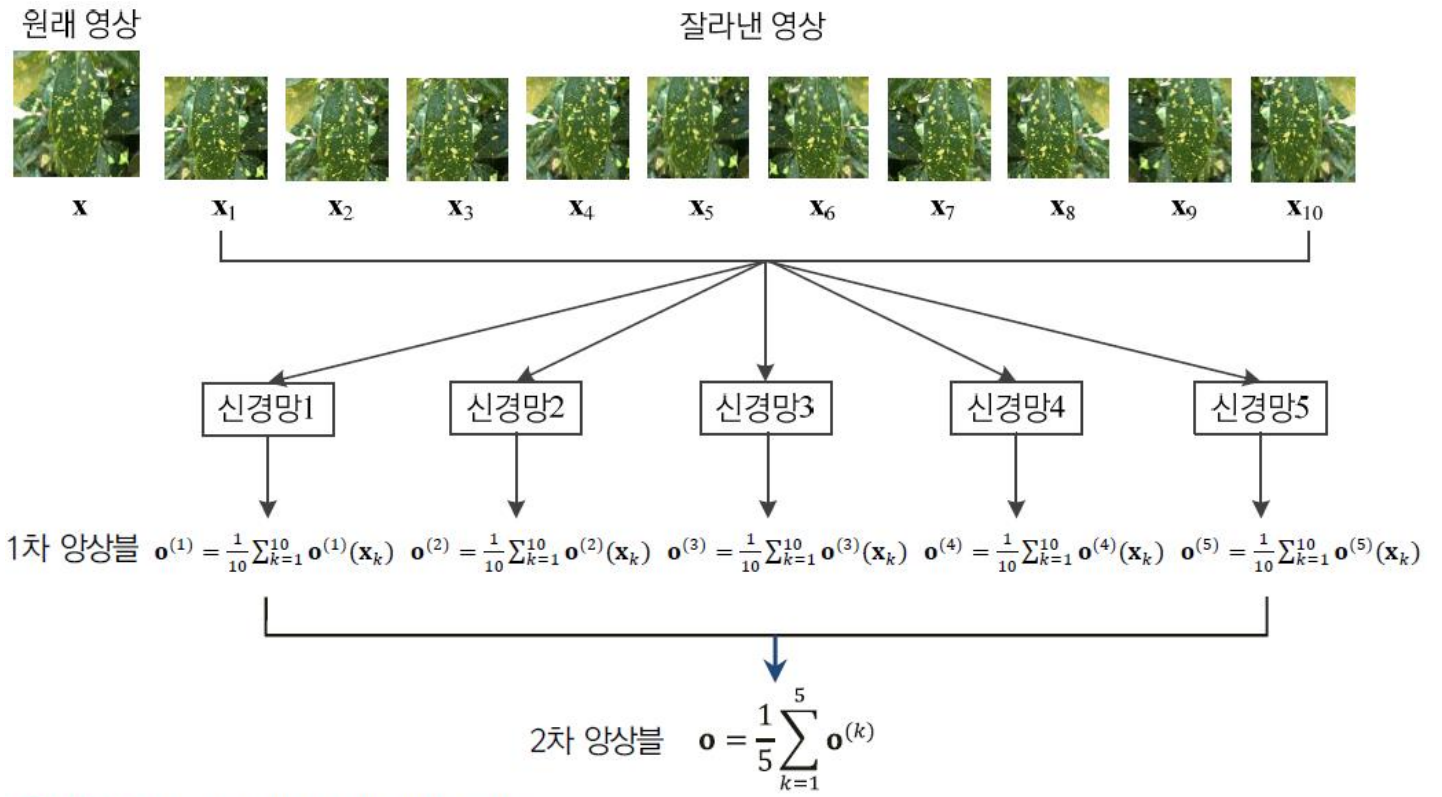


그림 12-5 AlexNet이 사용한 앙상블 기법

12.5.1 평균 기법을 이용한 앙상블

- GoogLeNet의 앙상블(2014년 ILSVRC 대회 우승)
  - 영상 잘라내기 144장, 신경망 7개 결합했을 때 오류율 6.67%(3.40%만큼 오류율 감소)

표 12-1 GoogLeNet의 앙상블에 의한 성능 향상

신경망 개수	영상 잘라내기 개수	계산 시간	5순위 오류율	오류율 감소
1	1	1	10.07%	0
1	10	10	9.15%	0.92
1	144	144	7.89%	2.18
7	1	7	8.09%	1.98
7	10	70	7.62%	2.45
7	144	1008	6.67%	3.40

  - 2014년 2위를 차지한 VGGNet의 오류율은 7.32%
    - 단일 분류기의 경우, VGGNet 7.0%, GoogLeNet 7.89%로 VGGNet이 우월 → VGGNet은 열등한 앙상블 기법을 사용한 탓에 2위로 밀린 셈

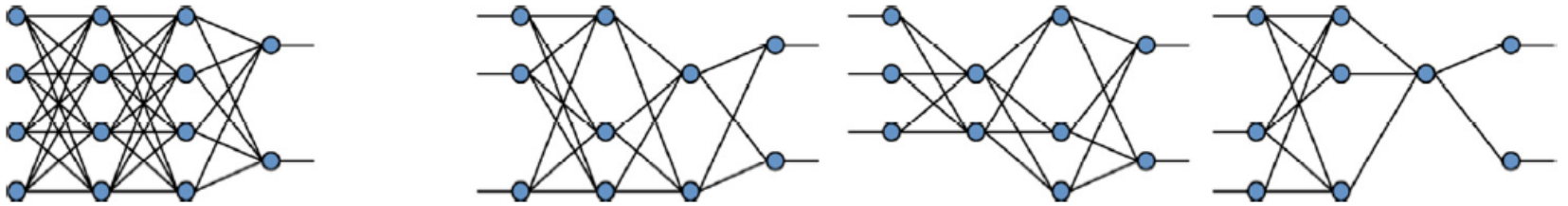
- ResNet의 앙상블(2015년 ILSVRC 대회 우승)
  - 신경망 하나만 사용했을 때 오류율 4.49% → 6개 결합 오류율 3.57%

12.5.2 암시적 앙상블

5.4.4절의 드롭아웃

- 학습 단계에서 서로 다른 구조의 부분 신경망 다수 생성([그림 5-27])
- 테스트 단계에서 여러 신경망의 출력을 평균하는 방식으로 앙상블 적용
- 실제로는 가중치를 공유함으로써 암시적으로 앙상블 효과를 거둠

← 구체적 내용은 5.4.4절 참조



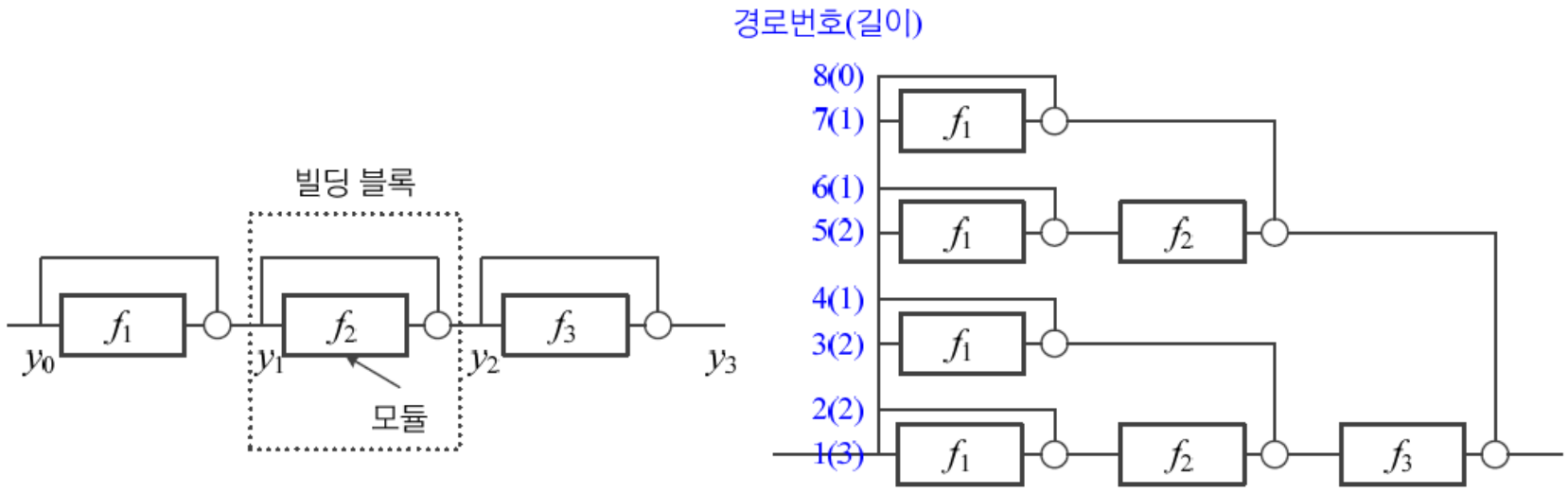
(a) 원래 신경망(4–4–4–2 구조) (b) 드롭아웃된 3개의 신경망 예시

그림 5–27 드롭아웃된 신경망

12.5.2 암시적 앙상블

ResNet의 지름길 연결을 새롭게 보기

- [그림 12-6(a)]에서 컨볼루션 2개 층을 모듈, 지름길까지 포함한 것을 빌딩볼록이라 부름 ([그림 4-29] 참조)



(a) 모듈과 빌딩 블록 (b) 여러 경로로 풀어헤침

그림 12-6 ResNet을 풀어헤침

12.5.2 암시적 앙상블

- [그림 12-6(a)]의 동작을 풀어 쓰면,

$$
\begin{aligned}
y_3 &= y_2 + f_3(y_2) \\
&= \left(y_1 + f_2(y_1)\right) + f_3\left(y_1 + f_2(y_1)\right) \\
&= \left(y_0 + f_1(y_0) + f_2\left(y_0 + f_1(y_0)\right)\right) + f_3\left(y_0 + f_1(y_0) + f_2\left(y_0 + f_1(y_0)\right)\right)
\end{aligned}
$$

  - 여러 경로를 풀어헤치면 [그림 12-6(b)] 가 됨
    - 모듈이 3개여서 2^3=8개의 경로 발생
    - 일반적으로 k개이면 2^k개의 경로 발생
  - 길이에 따른 경로의 개수 분포
    - 길이가 0, 1, 2, 3인 경로는 1, 3, 3, 1개
    - 일반적으로 이항분포를 따름

12.5.2 암시적 앙상블

- [Veit2016] 논문의 ResNet에 대한 새로운 해석
  - ResNet의 우수한 성능은 앙상블 효과에 기인(원래 논문 [He2016(a)]는 신경망이 깊은 탓으로 해석)
  - 모듈 하나를 제거하여도 성능 저하 거의 없음 (VGGNet은 층을 하나만 제거하여도 오류율이 90% 이상)
  - 제거한 모듈 수를 늘리면 성능이 서서히 저하
  - 오류 역전파 단계에서 그레이디언트는 대부분 짧은 경로를 흐름

  ← ResNet의 우수한 성능은 신경망의 깊이가 아니라 다수 경로의 앙상블 효과 때문임

12.5.2 암시적 앙상블

■ 스토캐스틱 깊이 기법[Huang2016]

- ResNet 학습에서 임의 선택된 빌딩블록에서 모듈을 제거하고 지름길만 남겨둔 채 가중치 갱신 → 컨볼루션 층을 배제하는 방식의 드롭아웃으로 간주할 수 있음
- 110층 신경망은 5.25%, 1202층은 4.91% 오류율로, 1000층을 넘어도 꾸준한 성능 향상(스토캐스틱 깊이를 적용하지 않았을 때는 너무 깊어지면 성능 저하 현상 나타남)

← 서로 다른 모듈을 제거한 아주 많은 신경망을 앙상블하는 효과로 해석

12.5.3 깊은 랜덤 포리스트

■ 랜덤 포리스트와 신경망의 장단점

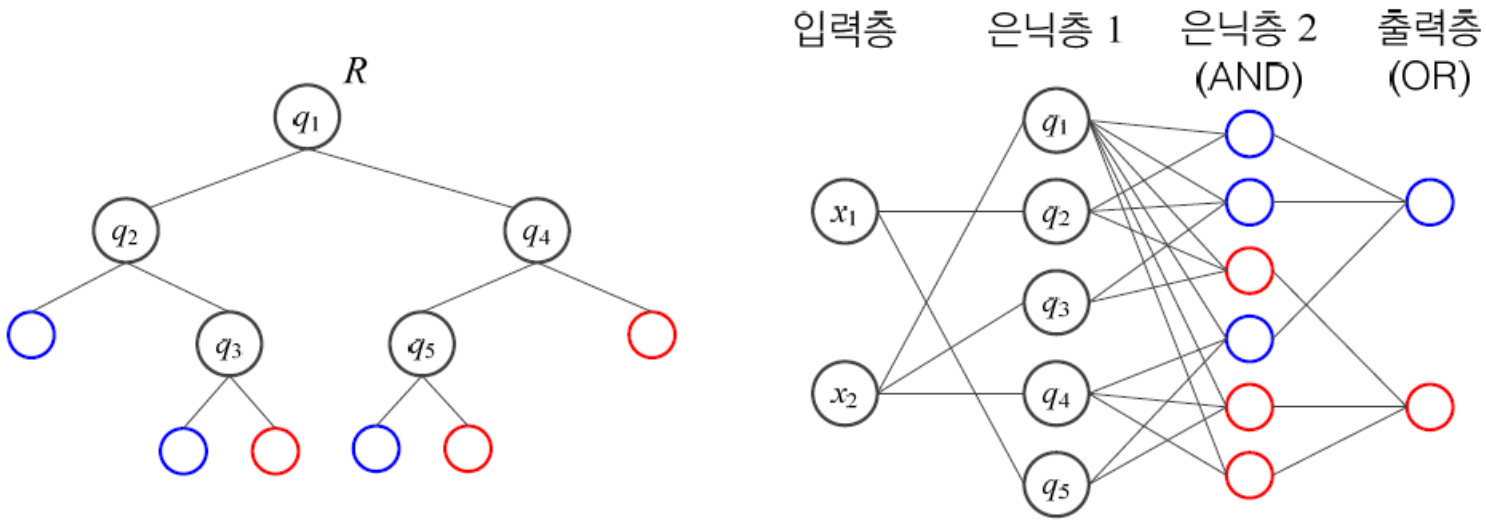
- 랜덤 포리스트는 작은 훈련집합으로도 학습이 잘되고, 예측 결과를 해석할 수 있는 장점
- 랜덤 포리스트는 표현 학습이 불가능하고, 노드에서의 의사 결정이 결정론적이라는 단점
- 랜덤 포리스트의 단점은 신경망의 장점이고 그 반대도 성립
- 최근 랜덤 포리스트에 신경망 특성을 부여하는 연구와 랜덤 포리스트와 딥러닝을 결합하는 연구 결과가 여럿 발표됨

■ 결정 트리의 학습 알고리즘을 개선한 연구 사례

- 노드의 질문 $x_i < \epsilon$?을 참거짓 판정하는 대신 시그모이드함수를 적용. 또한 학습을 마친 후 오류 역전파와 유사한 알고리즘을 이용하여 전역 최적화를 시도[Suarez1999]
- 랜덤 포리스트의 학습을 마친 후, 하나의 목적함수에 따라 모든 잎 노드를 전역 최적화 기법의 다시 조정[Ren2015]

12.5.3 깊은 랜덤 포리스트

- 결정 트리를 등가의 신경망으로 변환하는 연구 사례
  - 입력층에 특징 개수만큼 노드를 두고, 출력층에 부류 개수만큼 노드를 둠
  - 두 개의 은닉층
    - 첫 번째 은닉층에 내부 노드를 두고, 두 번째 은닉층에는 잎 노드를 둠
    - 첫 번째 은닉층의 노드는 자신의 자손 잎 노드와 연결
  - 두 번째 은닉층이 AND, 출력층이 OR 연산을 수행한다면 결정 트리와 신경망은 등가



(a) [그림 12-3(b)]의 결정 트리 (b) 등가의 신경망

그림 12-7 결정 트리를 등가의 신경망으로 변환

12.5.3 깊은 랜덤 포리스트

■ 랜덤 포리스트와 CNN을 상호 변환하는 알고리즘

- 학습된 랜덤 포리스트를 CNN으로 변환(CNN 초기화에 해당) → CNN을 학습한 후 랜덤 포리스트로 역변환 [Richmond2015]
  - 훈련집합이 작아도 학습이 잘되는 랜덤 포리스트의 장점과 표현 학습이 가능한 CNN의 장점을 결합한 접근방법
  - [그림 12-4]의 인체 분할 문제에서 성능 향상 입증
- 랜덤 포리스트와 CNN을 결합한 후 한꺼번에 학습하는 방법 [Kontschieder2015] ← 2015년 Marr 상을 수상

이미지 인식의 꽃, CNN 익히기

1 | 데이터 전처리

2 | 딥러닝 기본 프레임 만들기

3 | 더 깊은 딥러닝

4 | 컨볼루션 신경망(CNN)

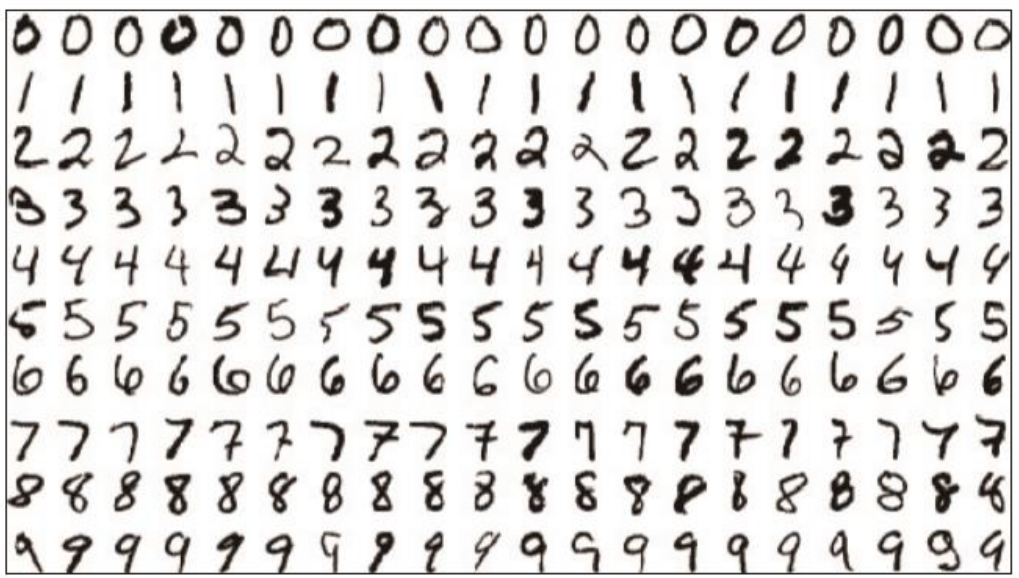
5 | 맥스 풀링

6 | 컨볼루션 신경망 실행하기

- **실습과제** MNIST 손글씨 인식

- 일반적인 사람에게 이 사진의 숫자를 읽어보라 하면 대부분 '504192'라고 읽을 것
- 그런데 컴퓨터에게 이 글씨를 읽게 하고 이 글씨가 어떤 의미인지를 알게 하는 과정은 쉽지 않음
- 숫자 5는 어떤 특징을 가졌고, 숫자 9는 6과 어떻게 다른지를 기계가 스스로 파악하여 정확하게 읽고 판단하게 만드는 것은 머신러닝의 오랜 진입 과제였음

- MNIST 데이터셋은 미국 국립표준기술원(NIST)이 고등학생과 인구조사국 직원 등이 쓴 손글씨를 이용해 만든 데이터로 구성되어 있음
- 70,000개의 글자 이미지에 각각 0부터 9까지 이름표를 붙인 데이터셋
- 머신러닝을 배우는 사람이라면 자신의 알고리즘과 다른 알고리즘의 성과를 비교해 보고자 한 번씩 도전해 보는 가장 유명한 데이터 중 하나



1 | 데이터 전처리

- MNIST 데이터는 케라스를 이용해 간단히 불러올 수 있음
- mnist.load_ data() 함수로 사용할 데이터를 불러옴

```
from keras.datasets import mnist
```

1 | 데이터 전처리

- 이때 불러온 이미지 데이터를 X로, 이 이미지에 0~9까지 붙인 이름표를 Y_class로 구분하여 명명하겠음
- 또한, 70,000개 중 학습에 사용될 부분은 train으로, 테스트에 사용될 부분은 test라는 이름으로 불러옴
  - 학습에 사용될 부분: X_train, Y_class_train
  - 테스트에 사용될 부분: X_test, Y_class_test

```
(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_data()
```

1 | 데이터 전처리

- 케라스의 MNIST 데이터는 총 70,000개의 이미지 중 60,000개를 학습용으로, 10,000개를 테스트용으로 미리 구분해 놓고 있음
- 이를 다음과 같이 확인할 수 있음

```
print("학습셋 이미지 수 : %d 개" % (X_train.shape[0]))
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))
```

```
학습셋 이미지 수: 60000 개
테스트셋 이미지 수: 10000 개
```

1 | 데이터 전처리

- 불러온 이미지 중 한 개만 다시 불러와 보자
- 이를 위해 먼저 matplotli 라이브러리를 불러옴
- imshow() 함수를 이용해 이미지를 출력할 수 있음
- 모든 이미지가 X_train에 저장되어 있으므로 X_train[0]을 통해 첫 번째 이미지를, cmap = 'Greys' 옵션을 지정해 흑백으로 출력되게 함

```
import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='Greys')
plt.show()
```

1 | 데이터 전처리

- 실행하면 그림 16-2와 같이 이미지가 출력됨

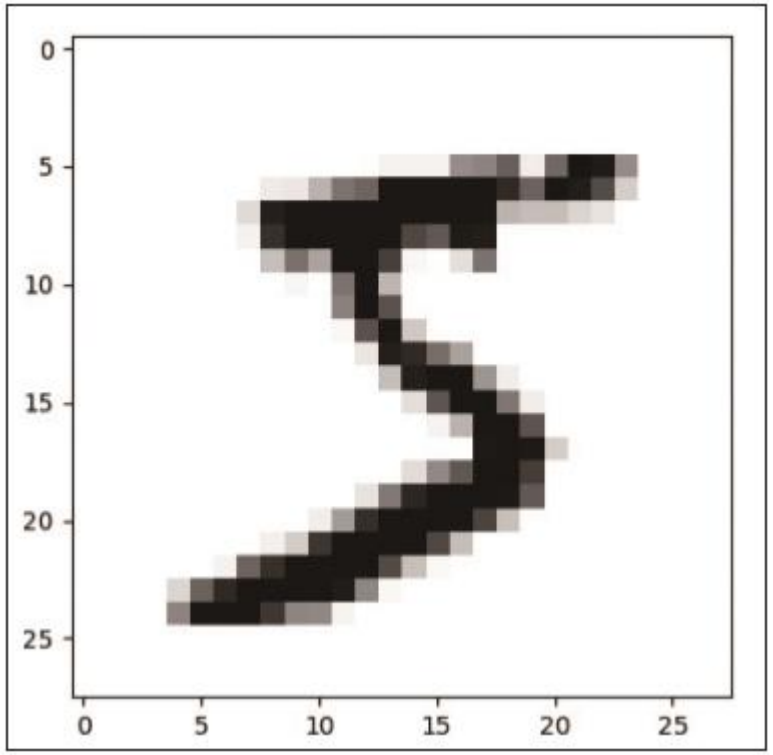


그림 16-2 MNIST 손글씨 데이터의 첫 번째 이미지

1 | 데이터 전처리

- 이 이미지를 컴퓨터는 어떻게 인식할까?
- 이 이미지는 가로 28 × 세로 28 = 총 784개의 픽셀로 이루어져 있음
- 각 픽셀은 밝기 정도에 따라 0부터 255까지 의 등급을 매김
- 흰색 배경이 0이라면 글씨가 들어간 곳은 1~255까지 숫자 중 하나로 채워져 긴 행렬로 이루어진 하나의 집합으로 변환됨
- 다음 코드로 이를 확인할 수 있음

```
for x in X_train[0]:
    for i in x:
        sys.stdout.write('%d\t' % i)
    sys.stdout.write('\n')
```

1 | 데이터 전처리

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	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	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	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	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	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0
0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0
0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0	0	0
0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	14	1	154	253	90	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	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	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	35	241	225	160	108	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	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	249	253	249	64	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	136	253	253	253	212	135	132	16	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	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	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

그림 16-3 그림의 각 좌표를 숫자로 표현해보기

1 | 데이터 전처리

- 이렇게 이미지는 다시 숫자의 집합으로 바뀌어 학습셋으로 사용됨
- 우리가 앞서 배운 여러 예제와 마찬가지로 속성을 담은 데이터를 딥러닝에 집어 넣고 클래스를 예측하는 문제로 전환시키는 것
- 28 × 28 = 784개의 속성을 이용해 0~9까지 10개 클래스 중 하나를 맞히는 문제가 됨

1 | 데이터 전처리

- 주어진 가로 28, 세로 28의 2차원 배열을 784개의 1차원 배열로 바꿔 주어야 함
- 이를 위해 reshape() 함수를 사용
- reshape(총 샘플 수, 1차원 속성의 수) 형식으로 지정
- 총 샘플 수는 앞서 사용한 X_train.shape[0] 을 이용하고, 1차원 속성의 수는 이미 살펴본 대로 784개

```
X_train = X_train.reshape(X_train.shape[0], 784)
```

1 | 데이터 전처리

- 케라스는 데이터를 0에서 1 사이의 값으로 변환한 다음 구동할 때 최적의 성능을 보임
- 따라서 현재 0~255 사이의 값으로 이루어진 값을 0~1 사이의 값으로 바꿔야 함
- 바꾸는 방법은 각 값을 255로 나누는 것
- 이렇게 데이터의 폭이 클 때 적절한 값으로 분산의 정도를 바꾸는 과정을 데이터 **정규화**(normalization)라고 함

1 | 데이터 전처리

- 현재 주어진 데이터의 값은 0부터 255까지의 정수로, 정규화를 위해 255로 나누어 주려면 먼저 이 값을 실수형으로 바꿔야 함
- 따라서 다음과 같이 astype() 함수를 이용해 실수형으로 바꾼 뒤 255로 나눔

```
X_train = X_train.astype('float64')
X_train = X_train / 255
```

- X_test에도 마찬가지로 이 작업을 적용
- 다음과 같이 한 번에 적용시킴

```
X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255
```

1 | 데이터 전처리

- 이제 숫자 이미지에 매겨진 이름을 확인해 보자
- 우리는 앞서 불러온 숫자 이미지가 5라는 것을 눈으로 보아 짐작할 수 있음
- 실제로 이 숫자의 레이블이 어떤지를 불러오고자 Y_class_train[0]을 다음과 같이 출력해 보자

```
print("class : %d " % (Y_class_train[0]))
```

- 그러면 이 숫자의 레이블 값인 5가 출력되는 것을 볼 수 있음

```
class : 5
```

1 | 데이터 전처리

- 12장에서 아이리스 품종을 예측할 때 딥러닝의 분류 문제를 해결하려면 원-핫 인코딩 방식을 적용해야 한다고 배웠음(149쪽 참조)
- 즉, 0~9까지의 정수형 값을 갖는 현재 형태에서 0 또는 1로만 이루어진 벡터로 값을 수정해야 함
- 예를 들어 class가 '3'이라면, [3]을 [0,0,1,0,0,0,0,0,0,0]로 바꿔 주어야 하는 것

1 | 데이터 전처리

- 지금 우리가 열어본 이미지의 class는 [5]였음
- 이를 [0,0,0,0,1,0,0,0,0,0] 로 바꿔야 함
- 이를 가능하게 해 주는 함수가 바로 np_utils.to_categorical() 함수
- to_categorical(클래스, 클래스의 개수)의 형식으로 지정

```
Y_train = np_utils.to_categorical(Y_class_train,10)
Y_test = np_utils.to_categorical(Y_class_test,10)
```

1 | 데이터 전처리

- 이제 변환된 값을 출력해 보자

```
print(Y_train[0])
```

- 아래와 같이 원-핫 인코딩이 적용된 것을 확인할 수 있음

```
[ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
```

- 이제 딥러닝을 실행할 준비를 모두 마쳤음

1 | 데이터 전처리

코드 16-1 MNIST 손글씨 인식하기: 데이터 전처리

- **예제 소스** deep_code/14_MNIST_Data.py

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

from keras.datasets import mnist
from keras.utils import np_utils

import numpy
import sys
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)
```

1 | 데이터 전처리

```
# MNIST 데이터셋 불러오기
(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_data()

print("학습셋 이미지 수 : %d 개" % (X_train.shape[0]))
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))

# 그래프로 확인
import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='Greys')
plt.show()

# 코드로 확인
for x in X_train[0]:
    for i in x:
        sys.stdout.write('%d\t' % i)
    sys.stdout.write('\n')
```

1 | 데이터 전처리

```
# 차원 변환 과정
X_train = X_train.reshape(X_train.shape[0], 784)
X_train = X_train.astype('float64')
X_train = X_train / 255

X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255

# 클래스 값 확인
print("class : %d " % (Y_class_train[0]))

# 바이너리화 과정
Y_train = np_utils.to_categorical(Y_class_train, 10)
Y_test = np_utils.to_categorical(Y_class_test, 10)

print(Y_train[0])
```

2 | 딥러닝 기본 프레임 만들기

- 이제 불러온 데이터를 실행할 차례
- 총 60,000개의 학습셋과 10,000개의 테스트셋을 불러와 속성 값을 지닌 X, 클래스 값을 지닌 Y로 구분하는 작업을 다시 한번 정리하면 다음과 같음

```
from keras.datasets import mnist

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255

Y_train = np_utils.to_categorical(Y_train, 10)
Y_test = np_utils.to_categorical(Y_test, 10)
```

2 | 딥러닝 기본 프레임 만들기

- 딥러닝을 실행하고자 프레임을 설정
- 총 784개의 속성이 있고 10개의 클래스가 있음
- 따라서 다음과 같이 딥러닝 프레임을 만들 수 있음

```
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

2 | 딥러닝 기본 프레임 만들기

- 입력 값(input_dim)이 784개, 은닉층이 512개 그리고 출력이 10개인 모델
- 활성화 함수로 은닉층에서는 relu를, 출력층에서는 softmax를 사용했음
- 딥러닝 실행 환경을 위해 오차 함수로 categorical_crossentropy, 최적화 함수로 adam을 사용

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

2 | 딥러닝 기본 프레임 만들기

- 모델의 실행에 앞서 모델의 성과를 저장하고 모델의 최적화 단계에서 학습을 자동 중단하게끔 설정
- 14장에서 배운 내용과 같음(189쪽 참조)
- 10회 이상 모델의 성과 향상이 없으면 자동으로 학습을 중단

```
import os
from keras.callbacks import ModelCheckpoint,EarlyStopping

MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1,
save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)
```

2 | 딥러닝 기본 프레임 만들기

- 샘플 200개를 모두 30번 실행하게끔 설정
- 테스트셋으로 최종 모델의 성과를 측정하여 그 값을 출력

```
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30,
batch_size=200, verbose=0, callbacks=[early_stopping_callback,checkpointer])

print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test) [1]))
```

2 | 딥러닝 기본 프레임 만들기

- 실행 결과를 그래프로 표현해 보려면 다음과 같이 설정

```
import matplotlib.pyplot as plt

y_vloss = history.history['val_loss']

# 학습셋의 오차
y_loss = history.history['loss']

# 그래프로 표현
x_len = numpy.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

2 | 딥러닝 기본 프레임 만들기

코드 16-2 MNIST 손글씨 인식하기: 기본 프레임

- **예제 소스** deep_code/15_MNIST_Simple.py

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint,EarlyStopping

import matplotlib.pyplot as plt
import numpy
import os
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)
```

2 | 딥러닝 기본 프레임 만들기

```
# MNIST 데이터 불러오기
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255

Y_train = np_utils.to_categorical(Y_train, 10)
Y_test = np_utils.to_categorical(Y_test, 10)

# 모델 프레임 설정
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))

# 모델 실행 환경 설정
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

2 | 딥러닝 기본 프레임 만들기

```
# 모델 최적화 설정
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1,
save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델의 실행
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30,
batch_size=200, verbose=0, callbacks=[early_stopping_callback,checkpointer])

# 테스트 정확도 출력
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test) [1]))
```

2 | 딥러닝 기본 프레임 만들기

```
# 테스트셋의 오차
y_vloss = history.history['val_loss']

# 학습셋의 오차
y_loss = history.history['loss']

# 그래프로 표현
x_len = numpy.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시
plt.legend(loc='upper right')
# plt.axis([0, 20, 0, 0.35])
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

2 | 딥러닝 기본 프레임 만들기

- 실행 결과

```
Epoch 00000: val_loss improved from inf to 0.15460, saving model to ./
model/00-0.1546.hdf5
Epoch 00001: val_loss improved from 0.15460 to 0.10492, saving model to ./
model/01-0.1049.hdf5
Epoch 00002: val_loss improved from 0.10492 to 0.08447, saving model to ./
model/02-0.0845.hdf5
Epoch 00003: val_loss improved from 0.08447 to 0.07896, saving model to ./
model/03-0.0790.hdf5
Epoch 00004: val_loss improved from 0.07896 to 0.06699, saving model to ./
model/04-0.0670.hdf5
Epoch 00005: val_loss improved from 0.06699 to 0.06388, saving model to ./
model/05-0.0639.hdf5
Epoch 00006: val_loss did not improve
Epoch 00007: val_loss improved from 0.06388 to 0.06291, saving model to ./
model/07-0.0629.hdf5
Epoch 00008: val_loss improved from 0.06291 to 0.05828, saving model to ./
model/08-0.0583.hdf5
```

2 | 딥러닝 기본 프레임 만들기

```
Epoch 00009: val_loss did not improve
Epoch 00010: val_loss did not improve
Epoch 00011: val_loss did not improve
Epoch 00012: val_loss did not improve
Epoch 00013: val_loss did not improve
Epoch 00014: val_loss did not improve
Epoch 00015: val_loss did not improve
Epoch 00016: val_loss did not improve
Epoch 00017: val_loss did not improve
Epoch 00018: val_loss did not improve
Epoch 00019: val_loss did not improve
 9920/10000 [============================>.] - ETA: 0s
 Test Accuracy: 0.9821
```

2 | 딥러닝 기본 프레임 만들기

- 20번째 실행에서 멈춘 것을 확인할 수 있음
- 베스트 모델은 10번째 에포크일 때이며, 이 모델의 테스트셋에 대한 정확도는 98.21%
- 함께 출력되는 그래프로 실행 내용을 확인할 수 있음

2 | 딥러닝 기본 프레임 만들기

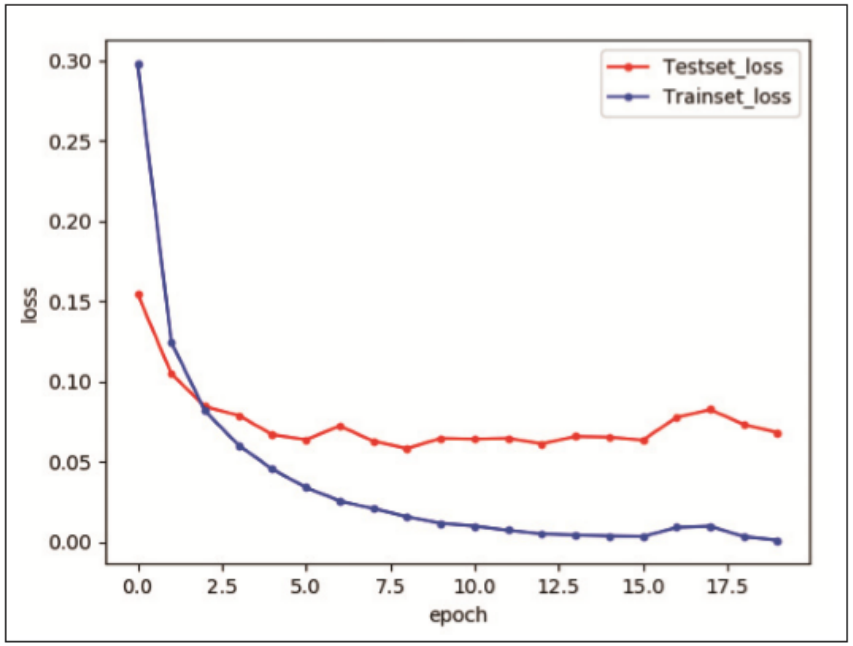


그림 16-4 학습이 진행될 때 학습셋과 테스트셋의 오차 변화

- 학습셋에 대한 오차는 계속해서 줄어듦
- 테스트셋의 과적합이 일어나기 전 학습을 끝낸 모습

3 | 더 깊은 딥러닝

- 98.21%의 정확도를 보인 딥러닝 프레임은 하나의 은닉층을 둔 아주 단순한 모델
- 이를 도식화해서 표현하면 그림 16-5와 같음

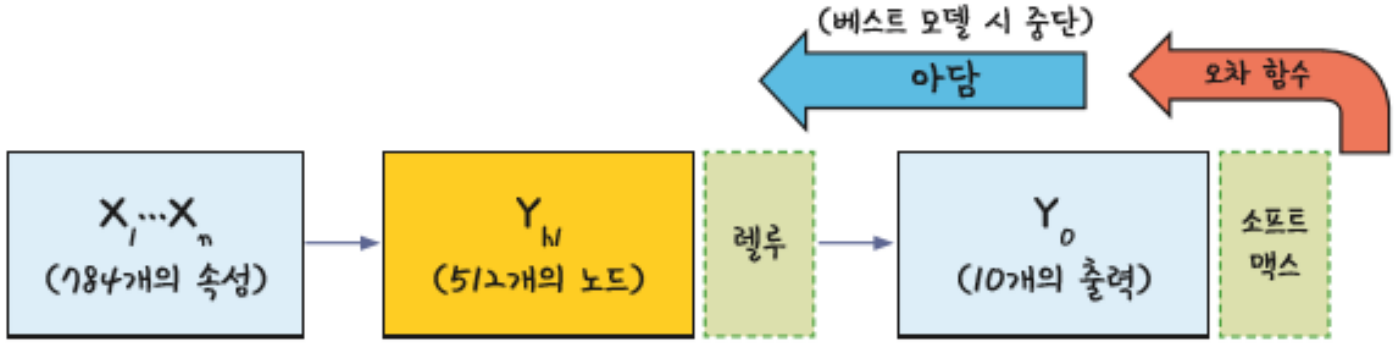


그림 16-5 은닉층이 하나인 딥러닝 모델의 도식

3 | 더 깊은 딥러닝

- 딥러닝은 이러한 기본 모델을 바탕으로, 프로젝트에 맞춰서 어떤 옵션을 더하고 어떤 층을 추가하느냐에 따라 성능이 좋아질 수 있음
- 지금부터는 기본 딥러닝 프레임에 이미지 인식 분야에서 강력한 성능을 보이는 컨볼루션 신경망(Convolutional Neural Network, CNN)을 얹어보자

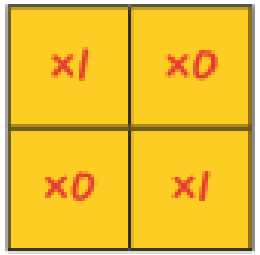
4 | 컨볼루션 신경망(CNN)

- 컨볼루션 신경망은 입력된 이미지에서 다시 한번 특징을 추출하기 위해 마스크 (필터, 윈도 또는 커널이라고도 함)를 도입하는 기법
- 예를 들어, 입력된 이미지가 다음과 같은 값을 가지고 있다고 하자

1	0	1	0
0	1	1	0
0	0	1	1
0	0	1	0

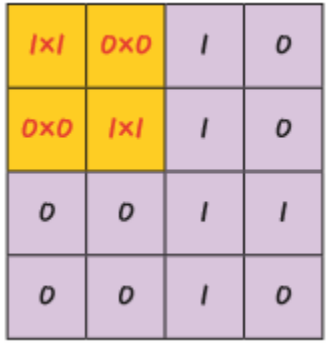
4 | 컨볼루션 신경망(CNN)

- 여기에 2×2 마스크를 준비함
- 각 칸에는 가중치가 들어있음
- 샘플 가중 치를 다음과 같이 ×1, ×0라고 하자



4 | 컨볼루션 신경망(CNN)

- 이제 마스크를 맨 왼쪽 위칸에 적용시켜 보자

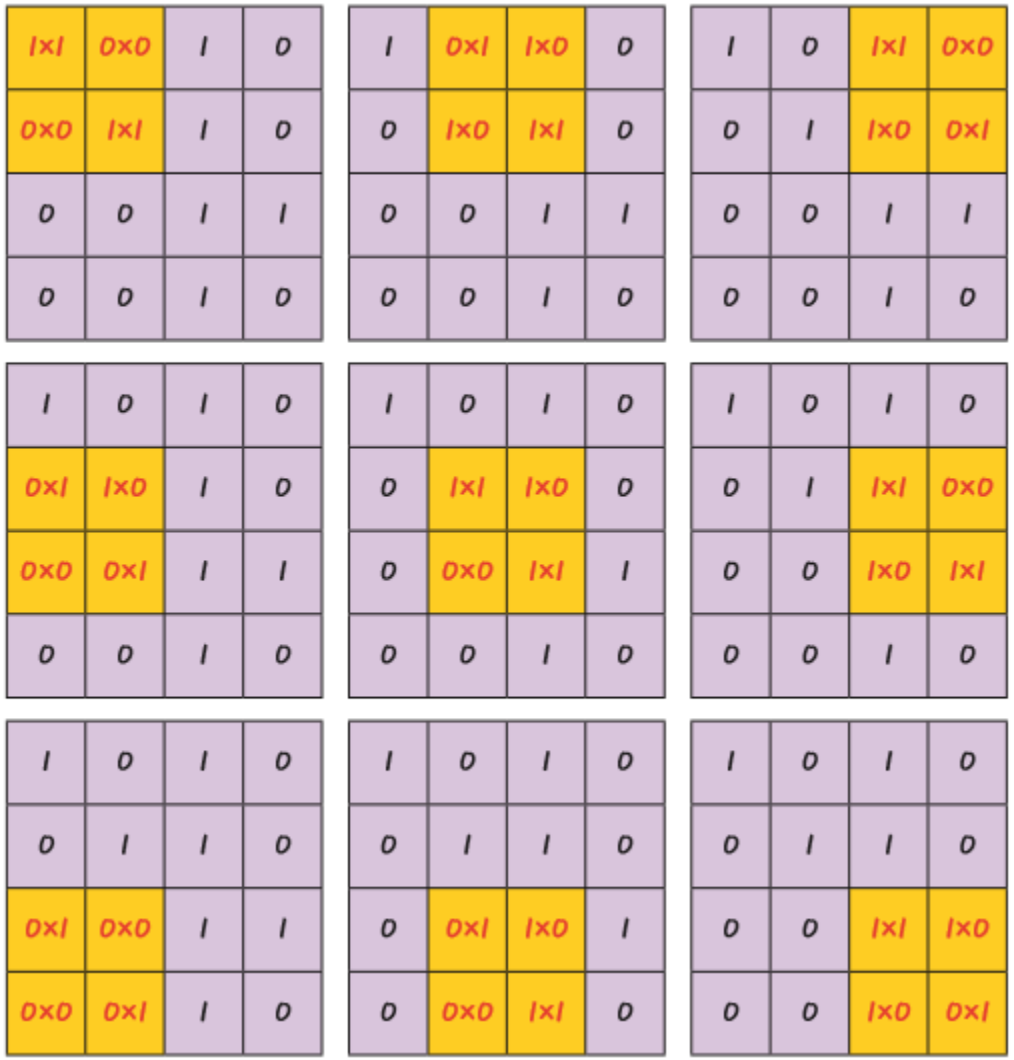


- 적용된 부분은 원래 있던 값에 가중치의 값을 곱해 줌
- 그 결과를 합하면 새로 추출된 값은 2가 됨

$$(1\times1)+(0\times0)+(0\times0)+(1\times1)=2$$

4 | 컨볼루션 신경망(CNN)

- 이 마스크를 한 칸씩 옮겨 모두 적용해 보자



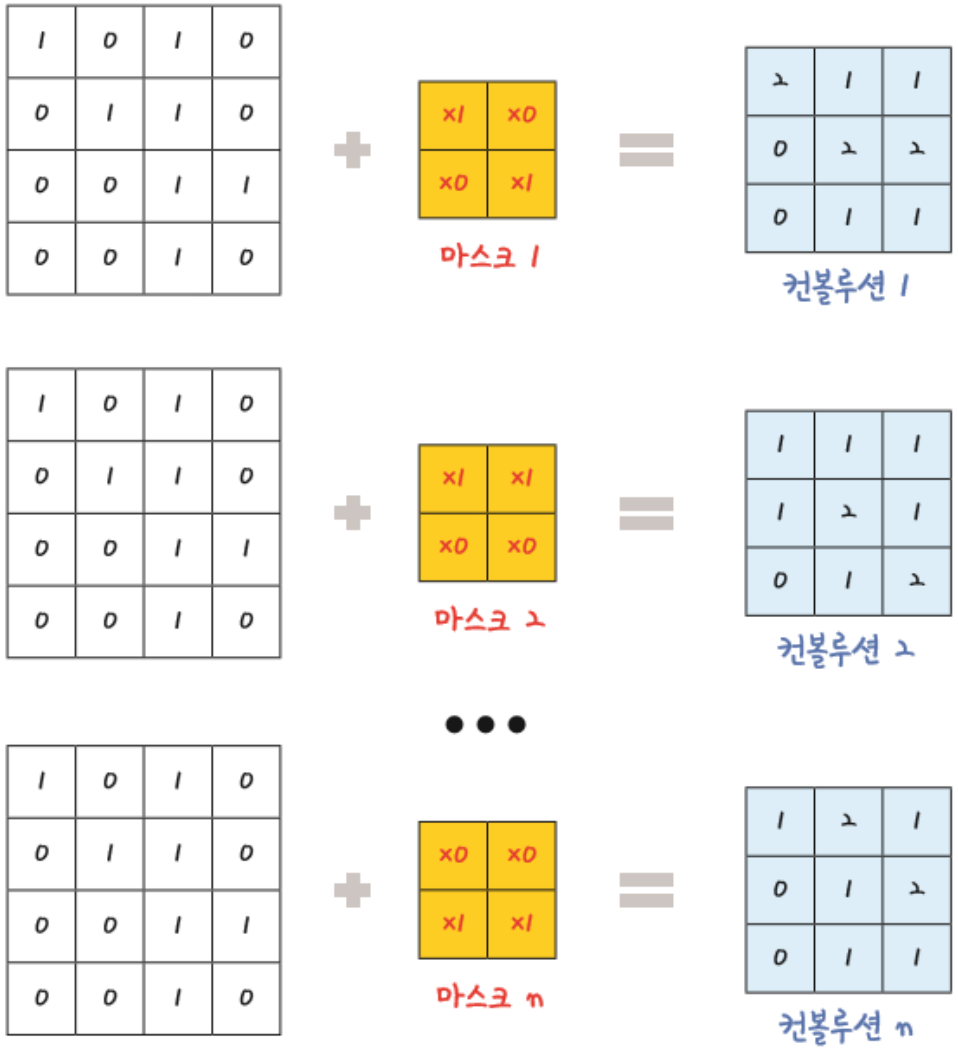
4 | 컨볼루션 신경망(CNN)

- 그 결과를 정리하면 다음과 같음

2	1	1
0	2	2
0	1	1

- 이렇게 해서 새롭게 만들어진 층을 컨볼루션(합성곱)이라고 부름
- 컨볼루션을 만들면 입력 데이터로부터 더욱 정교한 특징을 추출할 수 있음
- 이러한 마스크를 여러 개 만들 경우 여러 개의 컨볼루션이 만들어짐

4 | 컨볼루션 신경망(CNN)



4 | 컨볼루션 신경망(CNN)

- 케라스에서 컨볼루션 층을 추가하는 함수는 Conv2D()
- 다음과 같이 컨볼루션 층을 적용하여 MNIST 손글씨 인식률을 높여보자

```
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 1), activation='relu'))
```

→ 첫 번째 인자: 마스크를 몇 개 적용할지 정한다. 여기서는 32개의 마스크를 적용함

→ kernel_size: 마스크(커널)의 크기를 정한다. kernel_size=(행, 열) 형식으로 정하며, 여기서는 3×3 크기의 마스크를 사용하게끔 정한다

→ input_shape: Dense 층과 마찬가지로 맨 처음 층에는 입력되는 값을 알려 주어야 함 input_shape=(행, 열, 색상 또는 흑백) 형식으로 정한다. 만약 입력 이미지가 색상이면 3, 흑백이면 1을 지정.

→ activation: 활성화 함수를 정의

4 | 컨볼루션 신경망(CNN)

- 컨볼루션 층을 하나 더 추가하자
- 다음과 같이 마스크 64개를 적용한 새로운 컨볼루션 층을 추가할 수 있음

```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

4 | 컨볼루션 신경망(CNN)

- 컨볼루션 층을 추가한 도식을 그려보면 그림 16-6과 같음

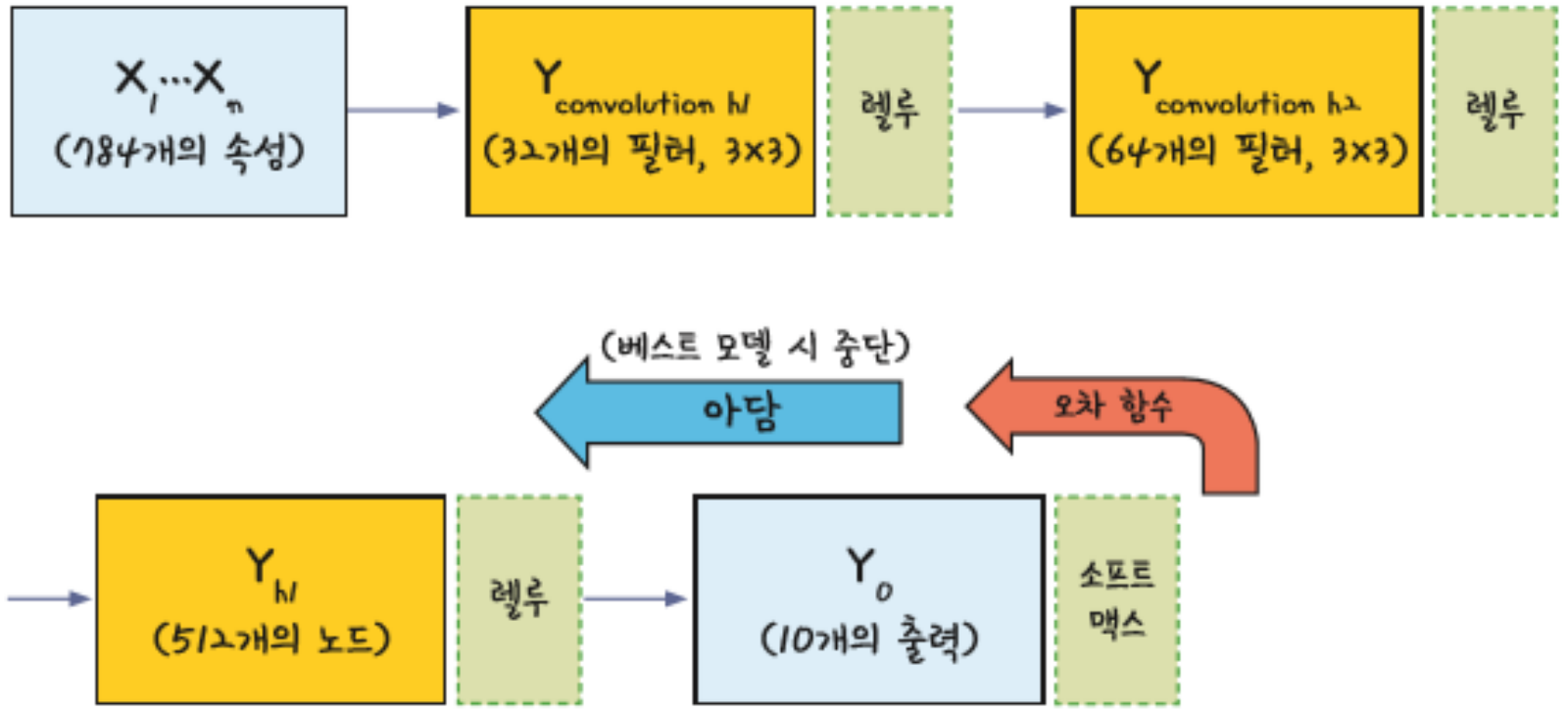


그림 16-6 컨볼루션 층의 적용

5 | 맥스 풀링

- 이제 여기에 맥스 풀링 층을 추가해 보자

- 앞서 컨볼루션 층을 통해 이미지 특징을 도출하였음
- 하지만 그 결과가 여전히 크고 복잡하면 이를 다시 한번 축소해야 함

  → 이 과정을 풀링(pooling) 또는 서브 샘플링(sub sampling)이라고 함

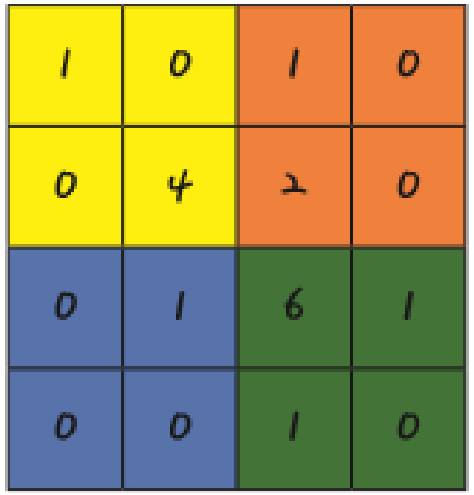
5 | 맥스 풀링

- 풀링 기법 중 가장 많이 사용되는 방법이 맥스 풀링(max pooling)
- 맥스 풀링은 정해진 구역 안에서 가장 큰 값만 다음 층으로 넘기고 나머지는 버림
- 예를 들어 다음과 같은 이미지가 있을때,

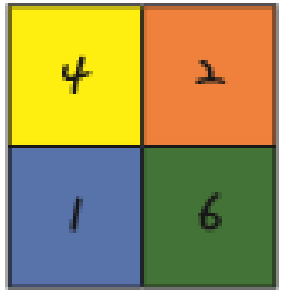
1	0	1	0
0	4	2	0
0	1	6	1
0	0	1	0

5 | 맥스 풀링

- 맥스 풀링을 적용하면 다음과 같이 구역을 나눔



- 그리고 각 구역에서 가장 큰 값을 추출



5 | 맥스 풀링

- 이 과정을 거쳐 불필요한 정보를 간추림
- 맥스 풀링은 MaxPooling2D() 함수를 사용해서 다음과 같이 적용할 수 있음

```
model.add(MaxPooling2D(pool_size=2))
```

5 | 맥스 풀링

- 여기서 pool_size는 풀링 창의 크기를 정하는 것으로, 2로 정하면 전체 크기가 절반으로 줄어듦

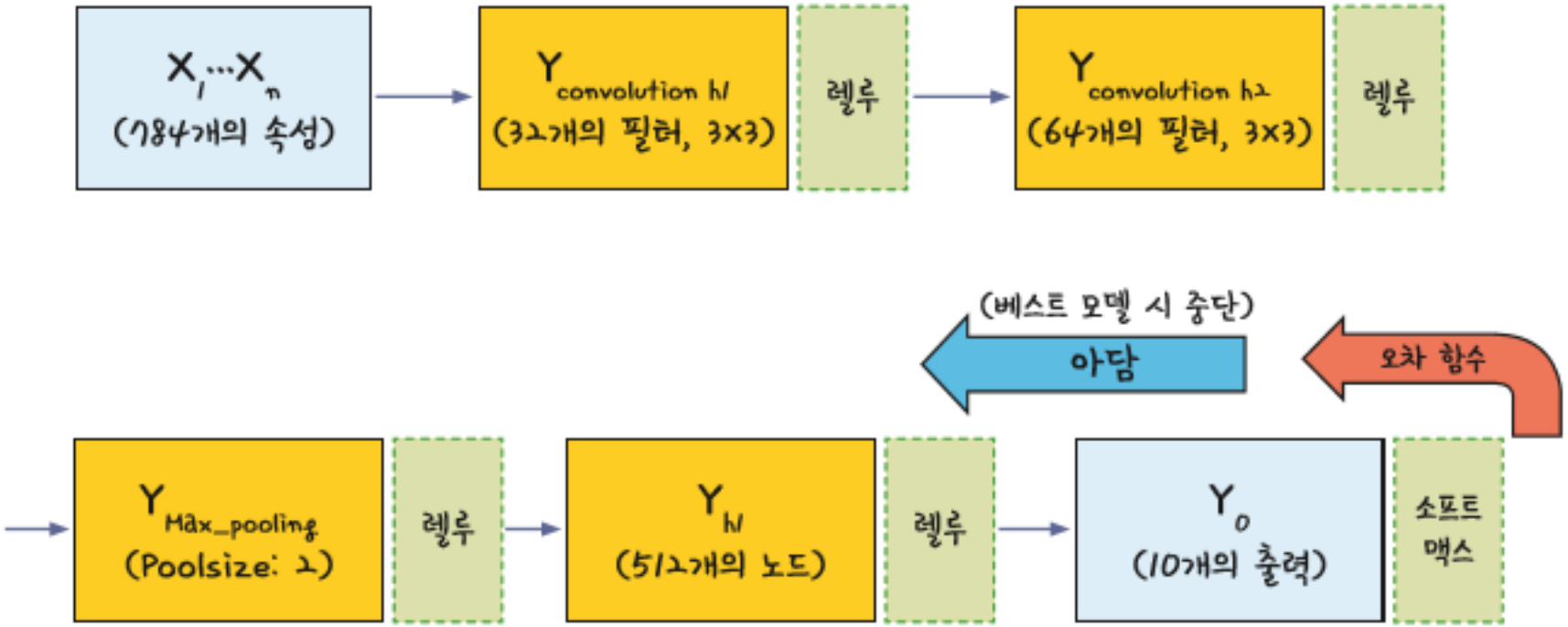


그림 16-7 맥스 풀링 층 추가

5 | 맥스 풀링

드롭아웃(drop out) , 플래튼(flatten)

- 노드가 많아지거나 층이 많아진다고 해서 학습이 무조건 좋아지는 것이 아니다

  → 과적합 발생!

- 과접합을 피하는 간단하지만 효과가 큰 기법이 바로 **드롭아웃**(drop out) 기법
- 드롭아웃은 은닉층에 배치된 노드 중 일부를 임의로 꺼주는 것

5 | 맥스 풀링

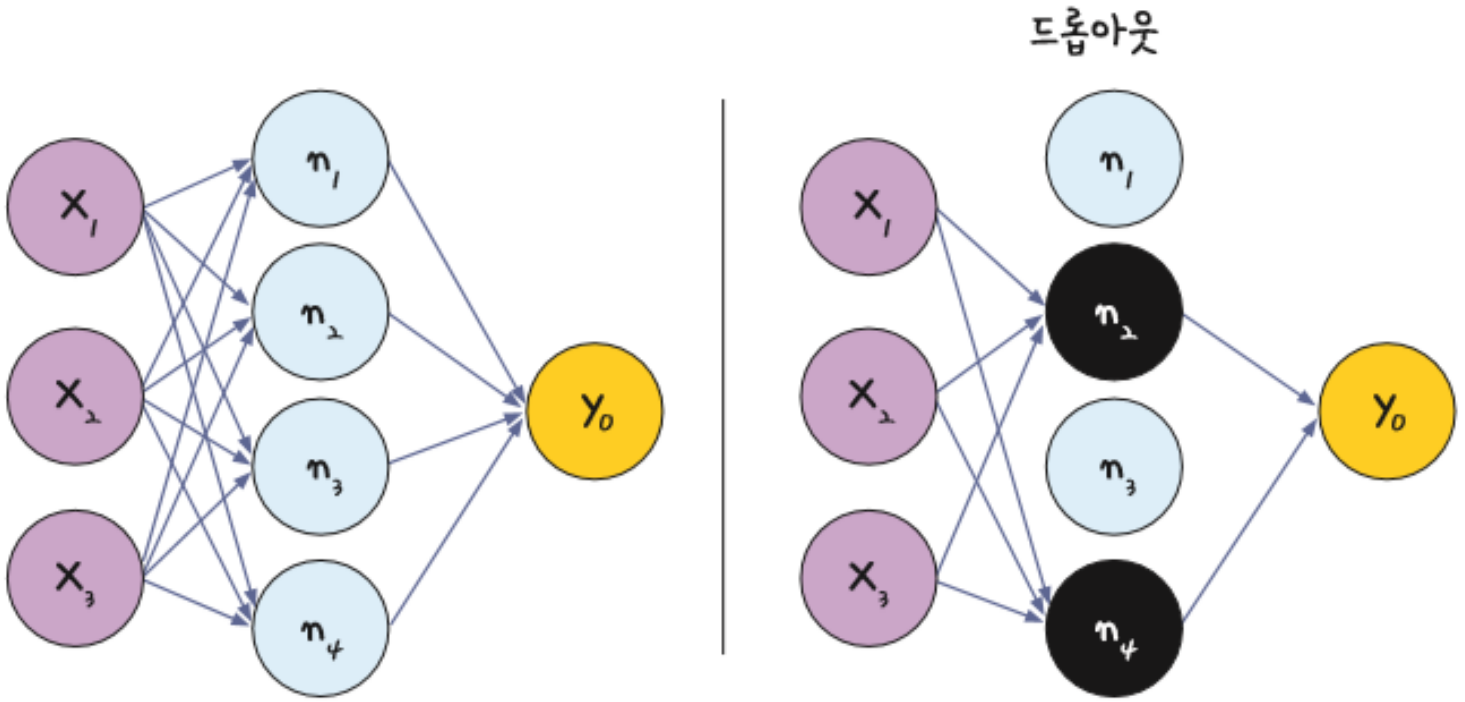


그림 16-8 드롭아웃의 개요, 검은색으로 표시된 노드는 계산하지 않는다.

5 | 맥스 풀링

- 이렇게 랜덤하게 노드를 끔으로써 학습 데이터에 지나치게 치우쳐서 학습되는 과적합을 방지할 수 있음
- 케라스는 이를 손쉽게 적용하도록 도와줌
- 예를 들어, 25%의 노드를 끄려면 다음과 같이 코드를 작성

```
model.add(Dropout(0.25))
```

5 | 맥스 풀링

- 이제 이러한 과정을 지나 다시 앞에서 Dense() 함수를 이용해 만들었던 기본 층에 연결해 보자
- 이때 주의할 점은 컨볼루션 층이나 맥스 풀링은 주어진 이미지를 2차원 배열인 채로 다룬다는 점
- 이를 1차원으로 바꿔주는 함수가 **플래튼** Flatten() 함수

```
model.add(Flatten())
```

5 | 맥스 풀링

- 이를 포함하여 새롭게 구현할 딥러닝 프레임을 그림 16-9와 같이 설정해 보자

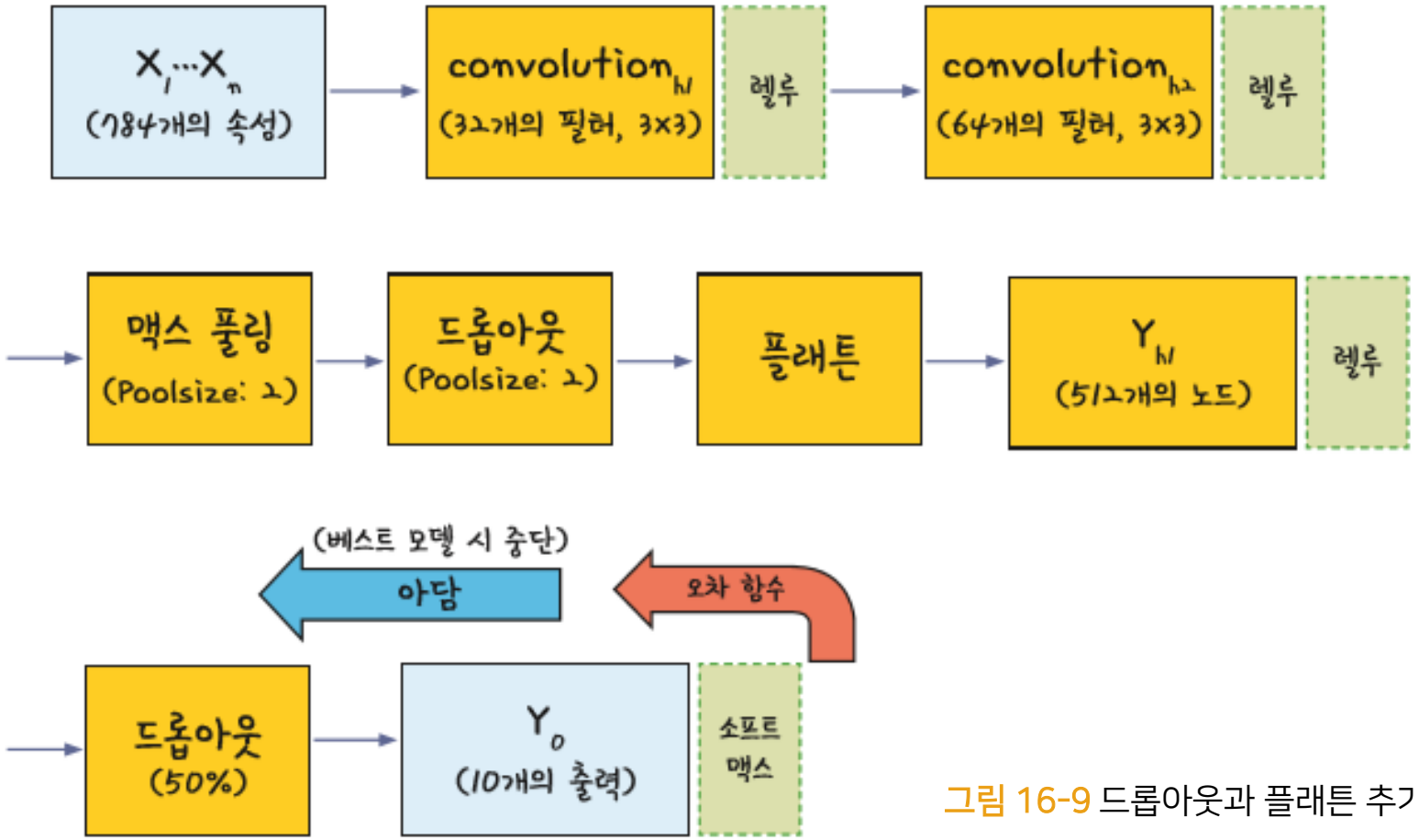


그림 16-9 드롭아웃과 플래튼 추가하기

6 | 컨볼루션 신경망 실행하기

- 지금까지 살펴본 내용을 코드로 작성해 보자
- 앞서 코드 16-2에서 만든 딥러닝 기본 프레임을 그대로 이용하되 model 설정 부분만 지금까지 나온 내용으로 바꿔주면 됨

6 | 컨볼루션 신경망 실행하기

코드 16-3 MNIST 손글씨 인식하기: 컨볼루션 신경망 적용

- **예제 소스** deep_code/16_MNIST_Deep.py

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.callbacks import ModelCheckpoint,EarlyStopping

import matplotlib.pyplot as plt
import numpy
import os
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)
```

6 | 컨볼루션 신경망 실행하기

```
# 데이터 불러오기
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255
Y_train = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)

# 컨볼루션 신경망 설정
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

6 | 컨볼루션 신경망 실행하기

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 모델 최적화 설정
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1,
save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델의 실행
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30,
batch_size=200, verbose=0, callbacks=[early_stopping_callback,checkpointer])
```

6 | 컨볼루션 신경망 실행하기

```
# 테스트 정확도 출력
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test) [1]))

# 테스트셋의 오차
y_vloss = history.history['val_loss']

# 학습셋의 오차
y_loss = history.history['loss']

# 그래프로 표현
x_len = numpy.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')
```

6 | 컨볼루션 신경망 실행하기

```
# 그래프에 그리드를 주고 레이블을 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

6 | 컨볼루션 신경망 실행하기

- 실행 결과

```
Epoch 00000: val_loss improved from inf to 0.06068, saving model to ./
model/00-0.0607.hdf5
Epoch 00001: val_loss improved from 0.06068 to 0.04409, saving model to ./
model/01-0.0441.hdf5
Epoch 00002: val_loss improved from 0.04409 to 0.03909, saving model to ./
model/02-0.0391.hdf5
Epoch 00003: val_loss improved from 0.03909 to 0.03188, saving model to ./
model/03-0.0319.hdf5
Epoch 00004: val_loss improved from 0.03188 to 0.02873, saving model to ./
model/04-0.0287.hdf5
Epoch 00005: val_loss did not improve
Epoch 00006: val_loss did not improve
Epoch 00007: val_loss did not improve
Epoch 00008: val_loss improved from 0.02873 to 0.02678, saving model to ./
model/08-0.0268.hdf5
Epoch 00009: val_loss did not improve
```

6 | 컨볼루션 신경망 실행하기

```
Epoch 00010: val_loss improved from 0.02678 to 0.02617, saving model to ./
model/10-0.0262.hdf5
Epoch 00011: val_loss did not improve
Epoch 00012: val_loss did not improve
Epoch 00013: val_loss improved from 0.02617 to 0.02454, saving model to ./
model/13-0.0245.hdf5
Epoch 00014: val_loss did not improve
Epoch 00015: val_loss did not improve
Epoch 00016: val_loss did not improve
Epoch 00017: val_loss did not improve
Epoch 00018: val_loss did not improve
Epoch 00019: val_loss did not improve
Epoch 00020: val_loss did not improve
Epoch 00021: val_loss did not improve
Epoch 00022: val_loss did not improve
Epoch 00023: val_loss did not improve
Epoch 00024: val_loss did not improve
Test Accuracy: 0.9928
```

6 | 컨볼루션 신경망 실행하기

- 12번째 에포크에서 베스트 모델을 만들었고 23번째 에포크에서 학습이 자동 중단됨
- 테스트 정확도가 99.28%로 향상됨
- 다음과 같이 함께 생성된 그래프로 학습 과정을 시각화할 수 있음

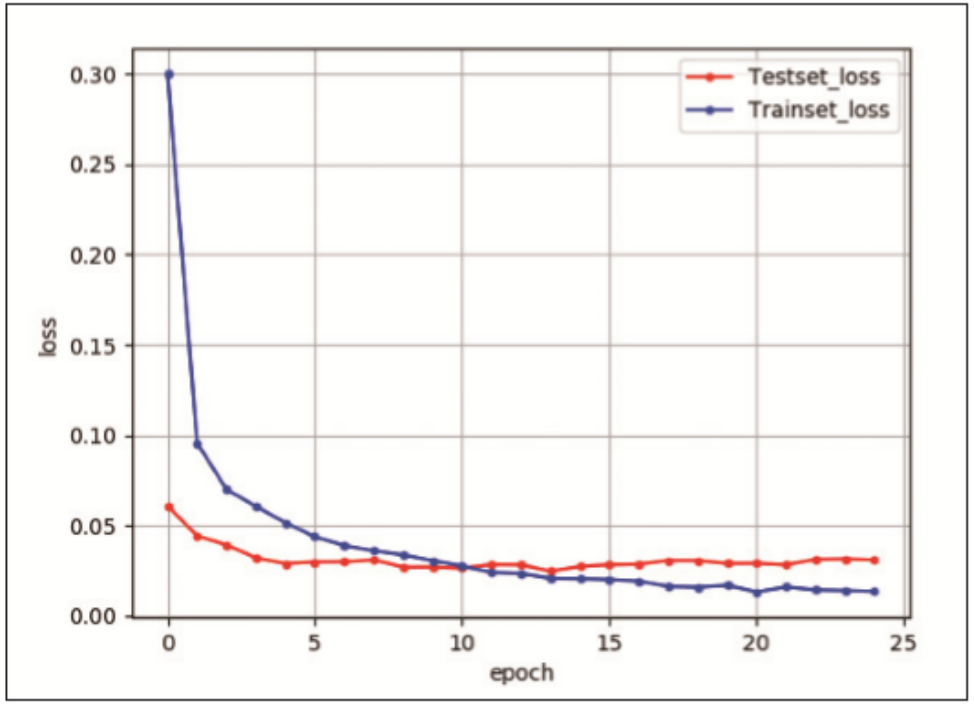


그림 16-10 학습의 진행에 따른 학습셋과 테스트셋의 오차 변화

6 | 컨볼루션 신경망 실행하기

- 0.9928, 즉 99.28%의 정확도는 10,000개의 테스트 이미지 중 9,928개를 맞추었다는 뜻
- 코드 16-2에서는 정확도가 98.21%였으므로 이보다 107개의 정답을 더 맞힌 것
- 100% 다 맞히지 못한 이유는 데이터 안에 다음과 같이 확인할 수 없는 글씨가 들어있었기 때문

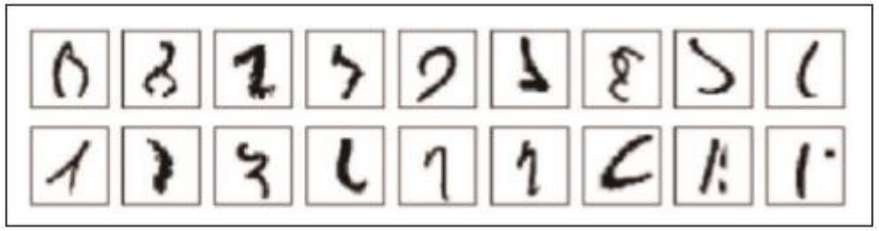


그림 16-11 알아내지 못한 숫자의 예

- 우리가 만든 딥러닝 모델은 이미 사람의 인식 정도와 같거나 이를 뛰어넘는 인식률을 보여 준다!