💜

CH2: Fundamentals of Machine Learning

2.1 Rule Based Machine Learning

해당 내용을 시작하기 앞서, 완벽한 세상이라는 시스템이 있다고 다음과 같은 가정을 하자.

- 관측 에러는 없다 (No observation errors, No inconsistent observations) $\rightarrow$ training data error와 noise 없음
- 일관적이지 않은 관측이 없다. 즉, 완벽히 관측되고 완벽히 일관적이다. (No stochastic elements in the system we obvere, Target function is deterministic)
- 정보가 모두 주어졌기 때문에, 시스템의 의사결정을 설명할 수 있다. (Full information in the observations to regenerate the system, target information is contained in hypothesis set)

정리하자면, 우리가 알아볼 세상에서는 관측 오차(Observation errors), 일관성이 없는 관측(Inconsistent observations), 어떤 확률론적 요소(Stochastic element)가 없으며, 우리가 관측하는 정보가 시스템에 있는 모든 정보다. 이 가정을 모두 만족하는 곳을 완벽한 세계(Perfect world)라고 한다.

이 완벽한 세계에서 관측한 데이터는 다음과 같을때, 어떠한 Factor가 나가서 노는데 (EnjoySpt)에 가장 큰 영향을 주는지 다양한 방법을 통해 알아보자.

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

2.1.1 Function Approximation

우리가 사용하는 데이터의 각각의 개체를 정확하게 알고 있으면 좋겠지만, 그 양이 무수히 많으면 모두 저장하는 것에 한계가 있다. 그래서, 데이터들이 일정한 경향성을 가지고 있다면, 이를 함수화시켜 놓으면, 필요할 때마다 실측값에 근사하여 사용할 수 있다. 이에 대한 장점은 다음과 같다.

- 실제로 가지고 있지 않은 데이터도 fit한 함수를 통해서 구할 수 있다
- 실제 데이터의 noise를 배제하고 학습 할 수 있다
- 고차원의 데이터도 효율적으로 저장이 가능하다

이렇게 각각의 데이터들을 모두 저장하는 것이 아니라 $ax^2 + bx + c$와 같이 식의 형태로 표현이 가능하다면, 방대한 영의 데이터를 저장하는 것 대신, parameter만 저장하는 방식을 선택할 수 있다. 그리고 이렇게 parameter를 이용하여 실제 값에 approximate하는 함수를 function approximator라고 한다. (자세한 내용을 알 고 싶으면 강화학습을 공부하자..., https://velog.io/@sjinu/개념정리-Introduction-to-Reinforcement-Learning)

뒷 부분을 진행하기 앞서 먼저, 용어를 정리해보자.

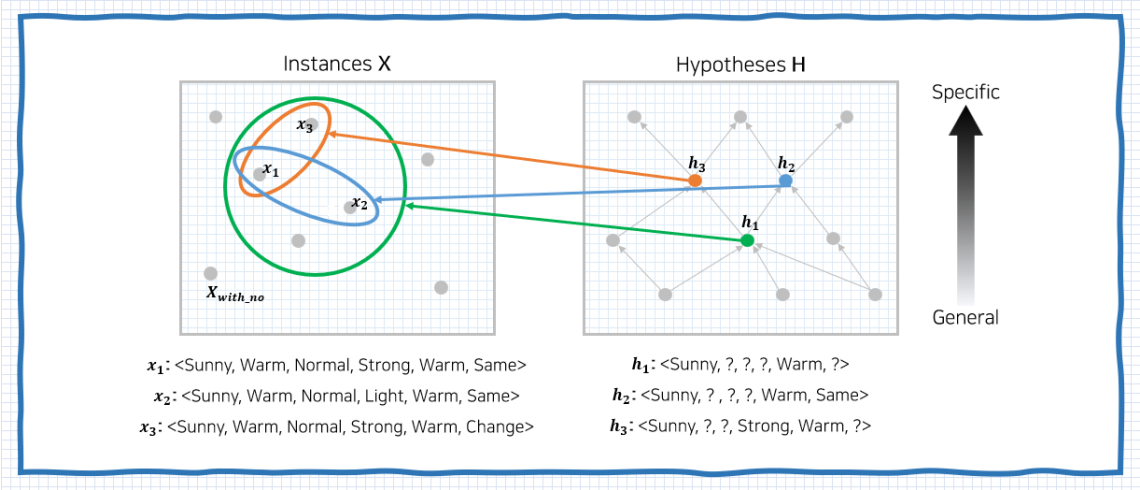
- Instance X: 하나의 sample을 의미, 위의 경우에서는 <Sunny, Warm, Normal, Strong, Warm, Same>이다.
- Label Y: 판단의 결과, 위의 경우에서는 Enjoy Sport의 Yes와 No이다
- Training Dataset: 여러개의 Instance X를 모아둔 것
- Hypothesis (H): 가설을 세워 Y를 추론하는 것으로, 여러 개의 가설을 세우는 것이 가능하다.

가설에 대해 조금더 알아보자면, **가설(Hypothesis)**이란, 특성값으로부터 특정 레이블을 도출할 가능성이 있는 함수다. 즉, A hypothesis h in H such that $h(x) = c(x)$ for all x in X 또는 A hypothesis h in H such that $h(x) = c(x)$ for all x in D로 나타낼 수도 있다. 위 데이터셋에서 도출할 수 있는 가설의 예시는 다음과 같다.

$$h_0 :< Sunny, Warm, ?, ?, ?, Same > \rightarrow Yes$$

위 가설에서 <Humid, Wind, Water, Forecst> 가 '?' 로 표시된 이유는 주어진 데이터셋을 통해서 도출해낼 수 없기 때문이다. Humid의 경우 인스턴스에서 2와 3에서 모두 'High'이지만 2번 인스턴스의 레이블은 'Yes', 3번 인스턴스의 레이블은 'No'이다. 이렇게 주어진 데이터셋 만으로는 해당 특성값이 레이블에 어떤 영향을 미치는지 알 수 없기 때문에 '?'로 표시한다. 그래서, '?'로 표시한 부분은 어느 값이든 다 들어갈 수 있다. 이렇게 결정되지 않은 특성 때문에 여러 종류의 가설이 가능하기 때문에, 데이터를 더욱 많이 수집하여 여러 가설로부터 하나의 목적 함수(Target function)을 찾는 것이 우리의 목적이다.

Graphical Representation of Function Approximation for Our Case



위의 그림을 보면 hypotheses space를 정의한 다음 instance space로 mapping하고 있는 것을 볼 수 있다. 사용된 notation의 뜻은 다음과 같다.

- x_i: input vector
- h_i: hypothesis / 가설을 설립하는 부분에서 특정 단어가 있으면, 그 단어를 만족해야 하고, 물음표가 있으면 그 부분은 무시가 가능하다.

위의 그림을 통해, 관측한 데이터의 개수에 따라 가설이 어떻게 변화하는지 알아보자. 위 그림에서 x_1, x_2, x_3을 관측하여 도출한 가설 h_1은 <Sunny, ?, ?, ?, Warm, ?>→ "YES"이다. 결정되지 않은 특성이 3개인 것을 알 수 있다. x_1, x_2만 관찰한 뒤에 가설을 세운 가설은 h_2로 <Sunny, ?, ?, ?, Warm, Same>→ "YES"이다. x_1, x_3를 관찰한 뒤 가설은 h_3으로, <Sunny, ?, ?, Strong, Warm, Same>→ "YES"이다.

이를 그림에 대입해서 보자면, h_1의 조건을 만족시키는 상황들이 초록색 원안에 있는 것을 확인할 수 있다. 즉, 모든 관측된 instance에 h_1는 만족된다는 것이다. 반대로, x_2는 가설 h_3에 따라 판단해보면 'Strong' 가준에 맞지 않기 때문에 No Enjoy Sport가 된다.

가설들 중 h_1은 상대적으로 여러 인스턴스로부터 도출된 일반적(General)인 가설이다. 즉, 관측된 데이터셋에 있는 인스턴스 말고도 해당 가설을 만족하는 인스턴스가 더 많을 수 있다. 반대로 h_2, h_3은 상대적으로 특수한(Specific)가설이다. 위의 그림을 보면, 화살표가 올라갈수록 가설이 general에서 specfic해지는 것을 볼 수 있다. General hypothesis의 경우, instance space의 넓이가 커서 더 많은 instance가 가설을 만족된다. 반대로, specific hypothesis의 경우, instance space의 넓이가 좁기 때문에 적은 양의 instance가 가설을 만족한다.

상대적으로 일반적인 가설과 특수한 가설 중 어떤 것이 더 좋을까?

> "기계 학습의 핵심은 **표현(representation)**과 **일반화(generalization)**에 있다. 표현이란 데이터의 평가이며, 일반화란 아직 알 수 없는 데이터에 대한 처리이다."

위의 인용으로부터, 특수한 가설을 만족할 수 있는 경우가 많지 않기 때문에, 상대적으로 일반적인 가설이 더 좋은 것을 알 수 있다. ~~하지만 우리가 알고 알고있다싶이 역시 가장 중요한 것은 양질의 데이터다... 데이터가 많으면 모든 것이 어느정도 해결된다...~~

이제 가설을 찾아나가기 위한 규칙 기반 학습 알고리즘에 대해 알아보자.

2.1.2 Find-S Algorithm

Find-S Algorithm은 specific한 가설로부터 general한 가설로 가설의 범위를 확장해나가는 알고리즘이다. 이 알고리즘은 아무런 특성도 하지 않은 영가설(Null hypothesis, h_o)로부터 시작해, postive한 레이블을 나타내는 경우를 하나씩 골라, 해당하는 특성을 기존의 가설과 비교해가며 판단한다. 기존의 가설과 새로운 인스턴스의 특성값이 다를 경우, 이 경우를 포함시키며 가설의 범위를 확장해 나간다. 즉, 특정한 convergance를 가진 h를 찾아나가는 과정이라고도 할 수 있다. 이를 알고리즘 적으로 표현하면 다음과 같다.

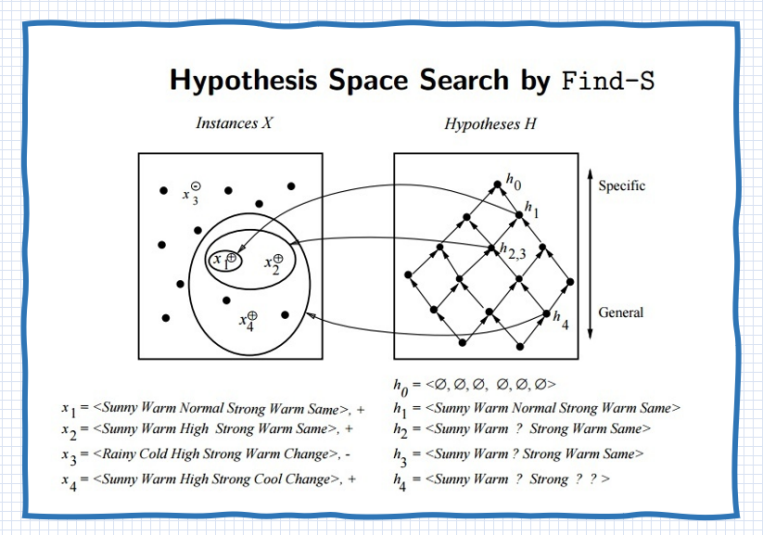
```
For instance x in D:
    if x > 0:
        for feature f in O:
            if f_1 in h== f_1 in x:
                Do Nothing
            Else:
                f_1 in h=f_1 in h U f_1 in x
```

```
Initialize h to the most specific hypothesis in H
    For each postivive training instance x
        For each attribute contraint a_1 in h
            If the constaint in h is satisfied by X THEN
                Do Nothing
            Else
                Replace a_i in h by next more general contraint satisfied by x
    Output hypothesis h
```

아래의 표에서 EnjoySpt가 'Yes'인 레이블을 나타내는 인스턴스는 1,2,4번이다. 이 3개의 인스턴스를 사용하여 Find-S Algorithm 과정을 진행해보자.

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

맨 처음은 영가설인 $h_0 =< \phi, \phi, \phi, \phi, \phi, \phi >$에서 시작한다. 새로운 인스턴스 1번에 의해 정해지는 가설 h_1은 <Sunny, Warm, Normal, Strong, Warm, Same>이다. 이 가설에 새로운 인스턴스 2번을 적용하면 가설 h_2는 <Sunny, Warm, ? , Strong, Warm, Same>으로 변한다. 즉, Humid 특성에 해당하는 가설이 변경되면서 가설의 범위가 확장되는 것을 볼 수 있다. 마지막으로 4번 인스턴스를 적용하면, 최종적으로 도출되는 가설은 h_3는 <Sunny, Warm, ? , Strong, ?, ?>이 된다. 이를 그림으로 나타내면 다음과 같다.



하지만, Find-S Algorithm은 consistent한 가설이라고 가정하므로 inconsistent한 가설에는 적합하지 않다. 또한, 너무 Specific한 h를 선택하게 되면 다른 가능한 가설들에 대해서 수렴하지 못하는 문제가 있다.

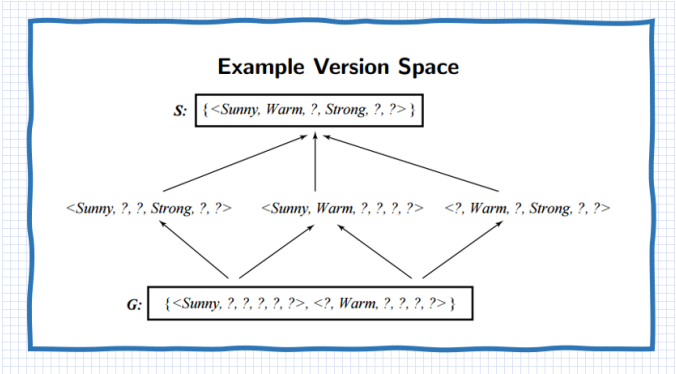
2.1.3 Version Space

앞선 예시에도 볼 수 있었다싶이, 일반화의 정도에 따라 규칙공간의 모든 가설들의 순서를 매길 수 있다. 훈련사례만 주어진 초기 상태에서 가장 구체적인(Sepcific)한 가설은 훈련사례 그 자체가 될 것이고, 가장 일반적인 가설은 영가설로써 (Null hypothesis) 모든 것을 다 포괄하는 가설이 된다. 이러한 가설들의 집합을 Version Space라고 한다. 즉, Version Sapce란, 위의 과정에서 생성가능한 모든 가설의 수를 모아둔 집합이다.

Version Space에는 가장 일반적인 가설을 나타내는 경계 (General Boundary) G와 가장 특수한 가설을 나타내는 경계 (Specific Boundary) S가 있다. 두 경계를 수식적으로 다음과 같이 나타낼 수 있다. 이는, 모든 가설은 general한 것과 specific 한 것 그 사이에 존재한다는 것이다.

$$VS_{H,D} = \{h \in H \mid \exists s \in S,\ \exists g \in G, g \geq h \geq s\} \ \ where\ x \geq y\ means\ x\ is\ more\ general\ or\ equal\ to\ y$$

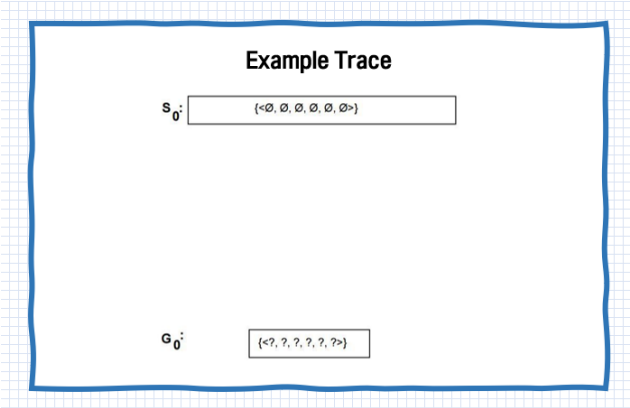
앞에서 알아본 것처럼 4개의 데이터셋으로부터 Find-S Algorithm을 통해 구할 수 있는 특수한 경계 S는 <Sunny, Warm, ?, Strong, ?, ?>이다. 그리고 가장 일반적인 경계 G는 <Sunny, ?, ?, ?, ?, ?>와 <?, Warm, ?, ?, ?, ?>이다. 이 사이에는 <Sunny, ?, ?, Strong, ?, ?>, <Sunny, Warm, ?, ?, ?, ?>, <?, Warm, ?, Strong, ?, ?>등의 많은 가설들이 존재할 수 있다.



하지만, 앞서 말했듯이, Find-S Algorithm은 너무 specific한 가설에 빠질 경우, 이를 개선할 수 있는 역추적 방식이 없다는 단점이 있다. 그렇다면 Find-S Algorithm 외에도 version space를 찾을 수 있는 다른 방법을 무엇일까?

2.1.4 Candidate Elimination Algorithm

후보 제거 알고리즘 (Candidatae Elimination Algorithm)은 Version Space를 구하기 위한 또다른 알고리즘이다. 이 알고리즘은 가장 Specific한 가설 $S_0 =< \phi, \phi, \phi, \phi, \phi, \phi >$와 가장 General한 가설 $G_o =<?, ?, ?, ?, ?, ? >$로부터 시작한다. 해당 알고리즘도 Find-S알고리즘과 마찬가지로 인스턴스를 하나씩 적용하여 두 가설 사이에 존재하는 후보 가설을 줄여나간다. 하지만, postive한 인스턴스만 사용하여 일반화를 진행한 Find-S Algorithm과 달리, 후보 제거 알고리즘은 positive와 negative한 인스턴스를 모두 사용한다.



후보 제거 알고리즘을 알고리즘적으로 표현하면 다음과 같다. 후보 제거 알고리즘이 진행되는 과정을 살펴보자면, 새로 적용할 인스턴스의 레이블이 postive하면 이 인스턴스를 만족시킬 수 있을 만큼만 specific boundary를 generalized한다. 이 과정에서 G를 분만족 시키는 가설이 있는 경우, 해당 조건을 제거한다. 반대로, 새로 적용할 인스턴의 레이블이 negative라면, 이 인스턴스가 negative로 분류될 수 있도록 일반 경계 G의 범위를 줄인다. 이 과정에서도 S에 새로 적용되는 인스턴스를 만족하는 조건이 있다면, 이 조건을 탈락시킨다.

```
For instance X in D:
    If y of x is positive:
        - Generalize S as much as needed to cover o in x
        - Remove any h in G, for which h(o) \neq y
    If y of x is negative:
        - Specialize G as much as needed to exclude o in x
        - Remove any h in S, for which h(o)=y
```

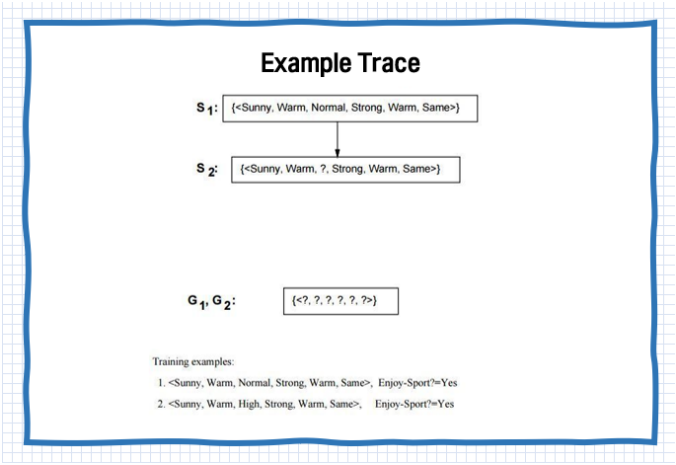
```
Input: Training set
Output:
    - G: Maximmaly general hypothesis in H (무조건 나감)
    - S: Maximmaly specific hypothesis in H (무조건 안나감)
Algorithm:
    If d>0:
        Remove from G any hypothesis inconsistent with d
        For each hypothesis s in S that is not consistent with d
            * Remove s from S
            * Add to S all minimal generalizations h of s such that (a) h is consistent with d, and (b) some member of G is more general t
            * Remove from S any hypothesis that is more general than another hypothesis in S
    If d<0:
        Remove from S any hypothesis inconsistent with d
        For each hypothesis g in G that is not consistent with d
            * Remove g from G * Add to G all minimal specializations h of g such that (a) h is consistent with d, and (b) some member of S
            * Remove from G any hypothesis that is less general than another hypothesis in G
```

이제, 이 후보제거 알고리즘을 위에서 사용한 예시를 통해 알아보자.

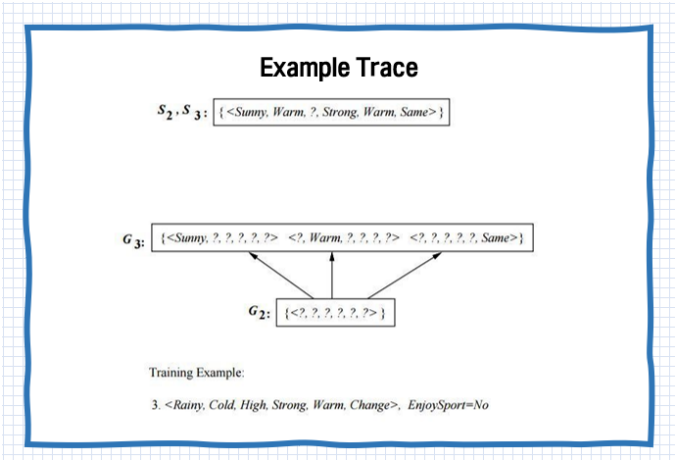
Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

첫 번째 인스턴스를 적용했을 때 각 경계는 다음과 같이 변한다: S_1: <Sunny , Warm, Normal, Strong, Warm, Same>, G_1:<?, ?, ?, ?, ?, ?>. 첫 번째 인스턴스의 레이블이 'Yes'이므로 이에 맞게 S를 조정하고 G를 만족하지 못하는 부분이 없으므로 그대로 유지한다.

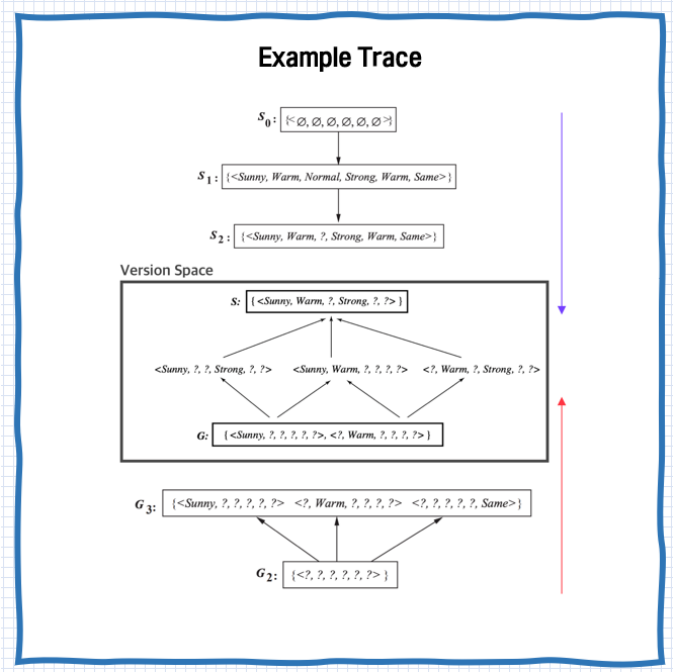
두 번째 인스턴스를 적용하면 S_2: <Sunny , Warm, ?, Strong, Warm, Same>, G_2:<?, ?, ?, ?, ?, ?>가 된다. 이는 앞선 과정과 동일하며, 여기까지는 Find-S Algorithm과도 유사한다. 첫번째와 두번째 인스턴스모두 positive하므로 S만 점점 General 해지는 것을 볼 수 있다.



세 번째 인스턴스는 앞의 두 경우와는 다르게 인스턴스가 negative한 경우다. 따라서, 세 번째 인스턴스를 적용한 가설은 S_3: <Sunny , Warm, ?, Strong, Warm, Same>, G_3:{<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>,<?, ?, ?, ?, ?, Same>}이 된다. 앞선 두 인스턴스와 세 번째 인스턴스는 'Sky','Temp',와 'Forecst'에서 다른 값을 나타낸다. 각 특성에서 세 번째 인스턴스의 특성값과 반대되는 특성값을 각각 G에 넣어주면 위와 같이 G를 구할 수 있다.



네 번째 인스턴스를 적용하면, S_4: <Sunny , Warm, ?, Strong, ?, ?>, G_4:{<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>}이 된다. 먼저 S쪽은 'Water'와 'Forecst'에서 기존 가설과 다르므로 '?' 처리를 통해 일반화를 진행했다. G는 네 번째 인스턴스에서 'Forecst'의 특성값이 가변적임에도 레이블이 'Yes'이므로 <?, ?, ?, ?, ?, Same> 가설을 탈락시켰다.



이렇게 도출된 version space를 다른 데이터에 적용하면 어떻게 될까?

Sky	Temp	Humid	Wind	Water	Forecst	Predict	EnjoySpt
Sunny	Warm	Normal	Strong	Cool	Change	?	Yes
Rainy	Cold	Normal	Light	Warm	Same	?	No
Sunny	Warm	Normal	Light	Warm	Same	?	X

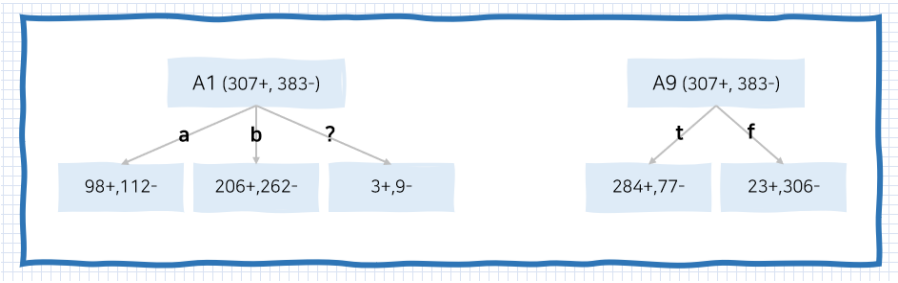
위의 표에서 첫 번째와 두 번째 인스턴스는 위에서 도출된 Version Space 밖에 존재하기 때문에 레이블을 결정할 수 있다. 첫 번째 인스턴스의 경우, S_4: <Sunny , Warm, ?, Strong, ?, ?>를 모두 만족시키므로 'Yes'로 레이블링 할 수 있다. 두 번째 인스턴스는 G_4:{<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>} 중 어느 것도 만족하지 않으므로 'No'로 레이블링 된다.

하지만, 마지막 인스턴스는 S_4, G_4 사이에 위치한다. 이 경우에는 함부로 레이블을 결정할 수 없다. 이런 경우에는 데이터의 수를 늘리면 된다. 완벽한 세계(Perfect world)에서는 데이터가 늘어나게 되면 후보 제거 알고리즘을 통해서 버전 공간(Version space)을 점점 줄여나갈 수 있고, 더욱 많아진다면 하나의 가설로 converge할 수도 있다. 하지만 현실세계에서는 모든 관측 데이터에 소음이 존재하고 우리가 관측하지 못하는 noise와 같은 특성이 개입될 수도 있기 때문에 후보 제거 알고리즘 등의 규칙 기반 방식으로는 현실에서 일어나는 모든 데이터에 대한 설명력이 떨어진다.

2.2 Decision Tree

규칙 기반 학습 (Rule based learning)에서는 관측된 데이터가 완벽한 세계 (Perfect World)에서 수집된 것으로 가정하고 진행하였다. 하지만, 우리가 살고 있는 세상에서는 모든 특성을 관측할 수 없고, 관측 오차도 발생한다. 그렇기 때문에, 규칙 기반 학습을 사용할 수 없으며, 더 robust한 방법과 hypothesis를 더 consise하게 세울 수 있는 모델을 찾아야 한다. 그 방법 중 하나가 의사 결정 나무 (Decision Tree)다.

이 부분에서는 UCI 신용평가 데이터셋을 분석하여 의사 결정 나무에 대해 알아볼 예정이다. 이 데이터셋에는 총 690개의 인스턴스가 있으며, 15개의 변수로 이루어져 있다. 아래의 사진은 학습 데이터셋이 첫 번째 변수(A1)과 아홉번째 변수(A9)로 나눴을 때 각각 어떤 결과로 나오는지 도식화 한 것이다.



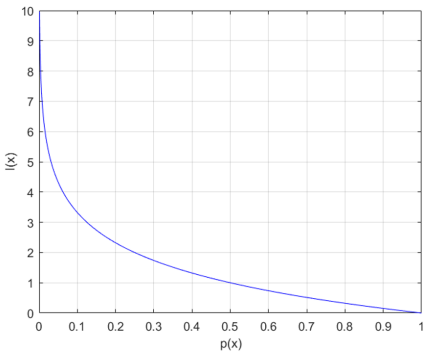
위의 이미지를 보면, 우리가 다루고자 할 데이터셋은 307개의 positive label을 가진 인스턴스와 383개의 negative label을 가진 인스턴스로 이루어져있다. 이를 A1의 특성으로 나눌 경우 왼쪽처럼 a노드에 98개의 positive, 112개의 negative 인스턴스가 배정되며, b도느에는 206개의 positive와 262개의 negative가 배정된다. 그리고 NA와 같이 어디에도 속하지 않는 12개의 값들이 존재한다. 이때, b 노드의 positve와 negative 인스턴스의 개수가 비슷하기 때문에, 분류할때 높은 uncertainty를 가진다. (만약 개수가 확 차이나면, 헷갈일 이유가 없이 잘 분류 된다고는 하지만, 데이터가 과하게 없으면 샘플링 해

야함...) 반대로, A9의 특성으로 필터링 할 경우에는 t 노드에 284개의 positive, 77개의 negative 인스턴스가 배정되며, f 노드에는 23개의 positive, 306개의 negative가 배정된다. 이를 통해, 데이터 분류에 사용한 변수 /feature에 따라 성능이 많이 상이한다는 것을 알 수 있다.

2.2.1 Entropy & Information Gain

만약, 위와 같이 변수에 따라 결과가 달라진다면, 어떤 변수를 기준으로 분류했을 때 더 좋은 기준이라고 할 수 있을까? 분류의 정도를 평가하기 위한 수치(Metric)으로 정보 엔트로피 혹은 섀넌 엔트로피(Sahnnon Entropy)가 있다. 정보 엔트로피가 무엇인지 일기 전에 정보 이론에 사용되는 정보량에 대해 알아보자.

정보량을 정의하는 방식은 많은데, 정보량은 이름 그래도 놀람의 정도라고 생각하면 된다. 이를 확률의 개념으로 설명하자면, 확률이 낮은 사건은 거의 일어나지 않기 때문에 정보량이 높다. 이를 통해, 정보량과 사건의 발생률이 반비례한다는 것을 알 수 있는데, 정보량I을 수식으로 표현하면 다음과 같다.

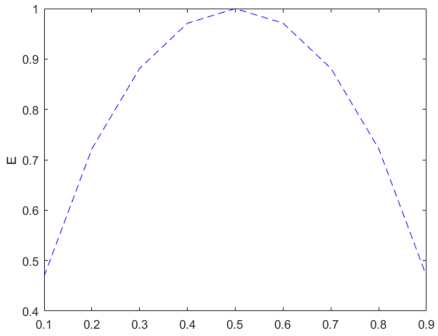


$$I(X) = \log_2 \frac{1}{p(X)}$$

위의 식에서 $p(X)$는 사건 X가 발생할 확률을 나타낸다. 이때, 로그 밑 b는 응용 분야에 따라 다르게 쓸 수 있는데, 대게는 2, e, 10을 사용한다. (각각을 사용했을 때의 정보량의 단위는 bit, nit, dit이다). 정보량을 정의할 때 왜 log를 사용할까?이는 정보에 대한 아래의 개념들은 만족시킬 수 있는 수식이어야 하기 때문이다.

- 확률값 / 확률밀도 값에 반비례하기 때문
- 두 사건의 정보량을 합치면 각 사건의 정보량을 합친 것과 같아야 함

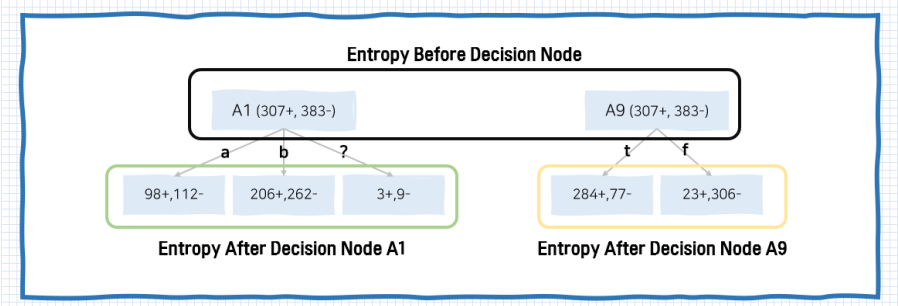
엔트로피 $H(x)$는 이 정보량의 평균을 나타내는 것으로 수식은 다음과 같다.



$$H(x) = -\sum_X P(X = x) \log_b P(X = x)$$

엔트로피는 데이터의 불확실성(Uncertainty)를 나타내므로 이 값이 높을수록 그 데이터셋이 더 불확실하다는 것을 의미한다. 이 엔트로피 공식에 조건부 확률을 적용하면, 조건부 엔트로피(Conditional Entropy)에 대한 식이 된다. 해당 식을 통해 X가 주어졌을 때 Y의 entropy를 계산할 수 있으며, $P(X = x)$에 대한 weighted sum이라는 것을 알 수 있다.

$$H(Y|X) = \sum_X P(X = x) H(Y|X = x) \log_b P(X = x) = \sum_X P(X = x) \{ -\sum_X P(Y = y|X = x) \log_b P(Y = y|X = x) \}$$



위의 식을 이용하여 원래 데이터셋의 엔트로피, A1과 A9로 분류했을 때의 엔트로피를 계산해보자.

$$H(Y) = \sum_{Y\in\{+,-\}} P(Y=y)\log_2 P(Y=y) = 0.991$$

$$H(Y|A1) = \sum_{Y\in\{a,b,?\}} \sum_{Y\in\{+,-\}} P(A1=x, Y=y)\log_2 \frac{P(A1=x)}{P(A1=x, Y=y)} = 0.989$$

$$H(Y|A9) = \sum_{Y\in\{t,f\}} \sum_{Y\in\{+,-\}} P(A9=x, Y=y)\log_2 \frac{P(A9=x)}{P(A9=x, Y=y)} = 0.566$$

정보 획득도(Information Gain, IG)란 엔트로피가 얼마나 줄었는지를 나타내는 지표로 데이터셋의 불확실성이 얼마나 줄었는지를 나타내는 수치이기도 하다. 정보 획득도를 구하는 식은 다음과 같고, 이를 위의 예시에도 적용해보자.

$$IG(Y, A_i) = H(Y) - H(Y|A_i)$$

$$IG(Y, A1) = H(Y) - H(Y|A1) = 0.991 - 0.989 = 0.002$$

$$IG(Y, A9) = H(Y) - H(Y|A9) = 0.991 - 0.566 = 0.425$$

앞서 entropy 지수는 낮으면 더 좋다고 했는데, A9의 특성으로 분리했을 때 A1의 특성으로 분리했을때보다 더 놓은 정보 획득도를 보이는 것을 확인할 수 있다. 이를 기준으로, 노드를 나눌 때, A1보다 A9가 더 좋은 변수임을 알 수 있다.

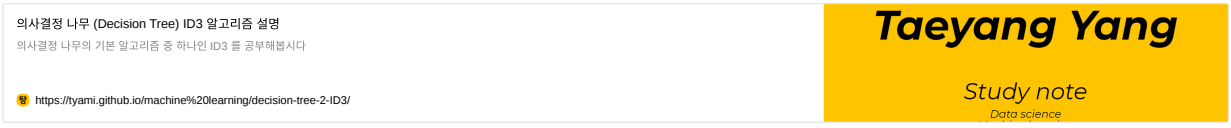
2.2.2 Variable Decision Tree

의사 결정 나무의 종류에도 여러가지가 있지만, 뒤에 갈 길이 바빠 짧게 설명하고 가겠다…

(1) ID3 (Iterative Dichotomiser 3)

ID3 알고리즘은 Iterative Dichotomiser 3의 약자로, Dichotomiser는 '이분하다'라는 뜻의 프랑스어로, 반복적으로 이분하는 알고리즘이다. 해당 알고리즘을 살펴보자면, 먼저 모든 인스턴스가 포함된 하나의 노드를 만든 뒤, 그 노드를 분류할 최적의 특성을 선택한다. 그리고 그 특성의 클래스에 맞게 인스턴스를 분리하여 새로운 노드에 배치한다. 한 노드에 담긴 인스턴스의 레이블이 모두 동일한 경우에 나누는 것을 종료한다. 가장 기본적인 의사 결정 나무 알고리즘이지만, 범주형 특성만을 다룰 수 있다는 단점이 있다. 또한 순수한 노드가 나올 때까지 가지를 나누게 되므로 과적합(Overfitting)이 쉽게 발생하는 문제점이있다.

ID3의 자세한 내용과 예시를 보고싶다면 다음 링크를 참고하자…



(2) C4.5

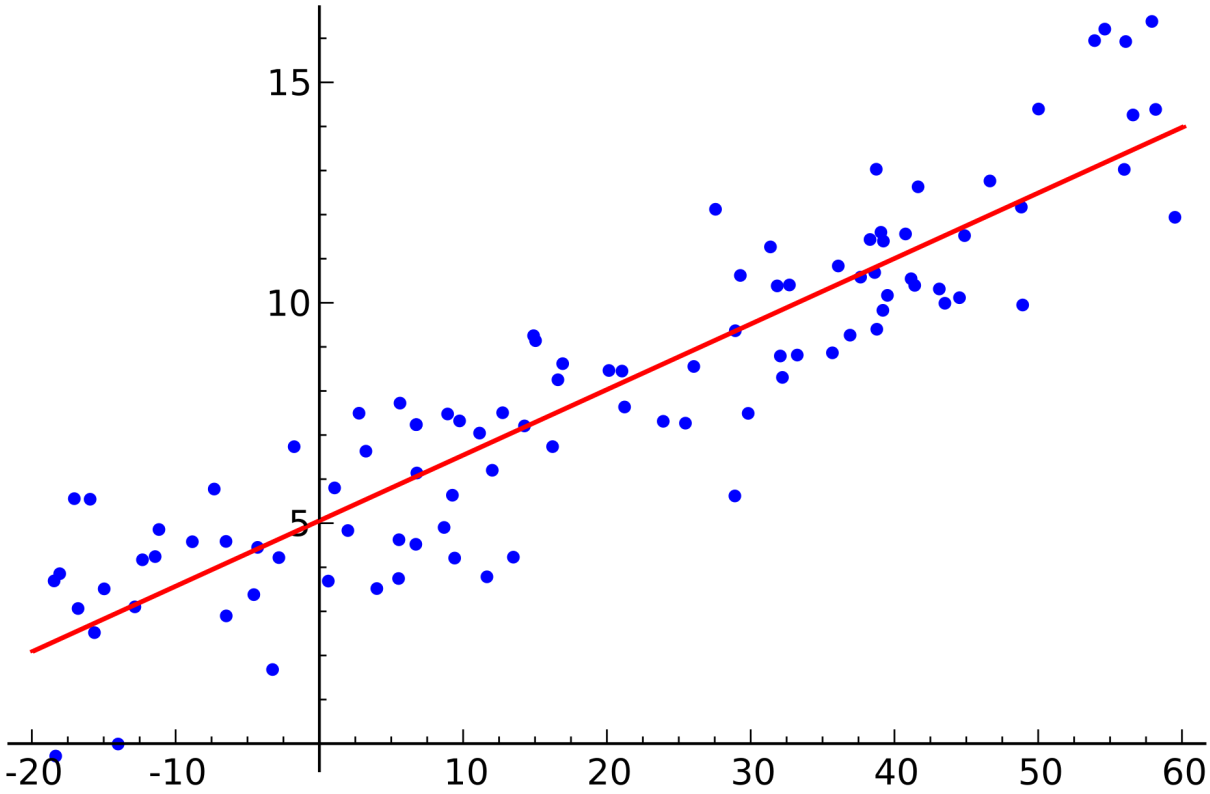
C4.5는 ID3알고리즘을 보완한 것으로 범주형 자료만 다룰수 있던 ID3에 비해 C4.5는 수치형 속성까지 다룰 수 있다. 그리고 해당 알고리즘에서는 pruning을 추가하여 트리가 깊어지는 것과 variance가 늘어나는 것이 방지할 수 있다. 또한, 결측치를 알아서 처리하고 속성마다 다른 가중치를 부여함으로서 계산을 효율적으로 진행할 수 있다.

(3) CART (Classification and Regression Tree)

잘 알고 있을 것이라고 믿는다..

2.3 Linear Regression

이 부분에서는 레이블이 범주형변수가 아닌 연속형 변수로 구성된 데이터셋을 알아보자.



위 그림에서 파란 점은 각 데이터의 연속형인 레이블을 표시한 것이고, 빨간색 선은 파란색 점을 가장 잘 설명할 수 있는 직선(또는 곡선)이다. 이 부분에서는, 특정 데이터셋의 특성(feature)과 레이블간에 선형 관계가 있을 것이라는 가설을 세웠을 때, 이 선현형의 관계를 예측하는 선형회귀(linear regression)에 대해 알아보자.

가장 기본적인 선형 회귀는 직선의 형태를 가지므로, 기울기를 나타내는 파라미터를 $\theta_i (i \geq 1)$라고 하고, 편향(Bias)를 나타내는 파라미터 θ_0라고 하면, 속성값 x_i에 대한 회귀 가설식 h를 다음과 같이 나타낼 수 있다.

$$h : \hat{f}(x;\theta) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i = \sum_{i=0}^{n} \theta_i x_i$$

위 식에서 x_i는 데이터로부터 주어져있기 때문에, 우리는 데이터에 더 잘 부합하는 θ를 찾아 더 좋은 가설을 만드는 것을 목표로 삼아야 한다.

2.3.1 Least Squared Method

최소제곱법(Least Squared Method)은 가장 적절한 θ를 찾는 방법 중 하나로, 행렬 연산을 기반으로 한다. 위에서 사용한 가설식 h를 행렬을 사용하여 다음과 같이 나타낼 수 있다. 이때, 각 행(Row)은 인스턴스의 특성값을 나타내며, D는 데이터셋 내에 있는 행의 개수가 된다.

$$h = \hat{f}(x;\theta) = \sum_{i=0}^{n} \theta_i x_i \rightarrow \hat{f} = X\theta$$

$$X = \begin{pmatrix} 1 & x_1^1 & \cdots & x_n^1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^D & \cdots & x_n^D \end{pmatrix}, \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$$

규칙 기반 학습에서도 봤던것 처럼 현실 세계는 완변한 세계와는 달리 다양한 노이즈가 존재하기 때문에, 현실 세계를 근사하는 함수 $f(\neq \hat{f})$에는 여러 에러ϵ에 대한 항을 포함해 한다.

$$f(x;\theta) = \sum_{i=0}^{n} \theta_i x_i + \epsilon \rightarrow f = X\theta + \epsilon = Y$$

우리의 목표는 θ를 조정하여 노이즈로부터 발생하는 에러를 최소화하는 것이다. 이를 수식으로 표현하면 다음과 같다.

$$\theta = \arg\min_{\theta}(\epsilon)^2 = argmin_{\theta}(f - \hat{f})^2$$

$f, \hat{f}$에 $Y, X\theta$를 각각 대입한 뒤 다음과 같이 변형할 수 있다.

$$\hat{\theta} = \arg\min_{\theta}(\epsilon)^2 = argmin_{\theta}(f - \hat{f})^2 = \arg\min_{\theta}(Y - X\theta)^2 = \arg\min_{\theta}(Y - X\theta)^T(Y - X\theta) = \arg\min_{\theta}(Y^T - \theta^T X^T)(Y - X\theta) = \arg\min_{\theta}(\theta^T X^T X\theta - 2\theta^T X^T Y + Y^T Y)$$

이제 이 식을 θ에 대해 미분한 후, 그 값이 0이 되는 θ를 찰아 $\hat{\theta}$를 구할 수 있다.

$$\nabla_\theta(\theta^{T}X^{T}X\theta - 2\theta^{T}X^{T}Y) = 0$$

$$2X^{T}X\theta - 2X^{T}Y = 0$$

$$\therefore \theta = (X^{T}X)^{-1}X^{T}Y$$

최소제곱법 덕분에 우리는 데이터셋이 주어지면 인스턴스의 특성값으로 구성된 행렬 X와 실제 레이블로 구성된 행렬 Y를 만든 뒤, 해당 데이터셋을 가장 잘 근사하는 선형 함수의 파라미터 θ를 구할 수 있다.

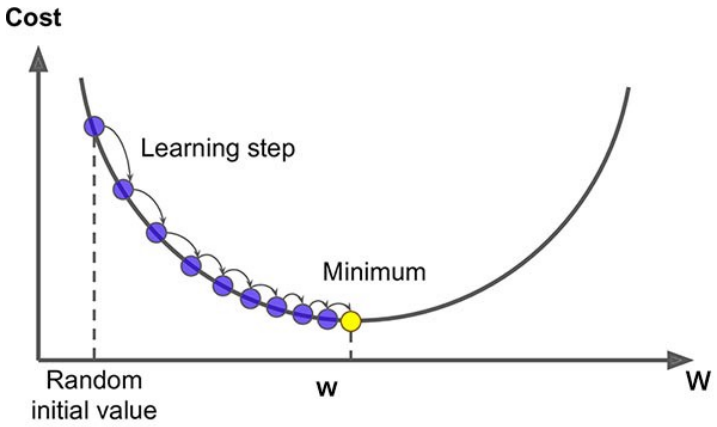
하지만, 최소제곱법은 행렬 연산에 기반하기 때문에, 데이터의 수에 따라 복잡도가 기하급수적으로 늘어나게 된다는 단점이 있다. 최소제곱법의 복잡도는 다음과 같다.

최소제곱법 복잡도	인스턴스(샘플) 수 m	특성/변수/feature 수 n
Big O	O(m)	$O(n^{2.4} \sim n^{3})$

위 표로부터 최소제곱법 알고리즘의 복잡도가 인스턴스의 개수에는 큰 영향을 받지 않지만, 특성의 개수에는 상당히 큰 영향을 받고 있음을 알 수 있다. 분석할 데이터셋이 인서턴스의 개수만 많고 feature의 개수는 적다면, 최소제곱법을 사용하는 것이 올바른 방법이 되겠지만, 데이터셋이 많은 feature/변수를 가지고 있다면, 최소제곱법을 사용하는 것은 많은 컴퓨팅 파워를 요구하기 때문에, 경사 하강법을 쓰는 것이 더욱 효율적일 것이다.

2.3.2 Graident Descent 경사하강법

경사 하강법은 최소제곱법과 같이 최적의 θ를 찾아가는 알고리즘으로 비용 함수(Cost function)이라는 개념을 사용한다. 비용 함수마다 약간의 차이가 있기는 하지만, 예측값과 실제값의 차이, 즉 $f - \hat{f}$라고 할 수 있다. 우리는 실제값과 가장 유사한 예측값을 알아내는 것이 목적이므로, 비용 함수를 최소화하는 방향으로 θ를 조정해 나간다.



경사 하강법에 본격적으로 들어가기 전에 선형 회귀에서 가장 흔히 사용되는 비용 함수인 평균 제곱 오차(Mean square error, MSE)의 수식은 다음과 같다.

$$MSE(X, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^{T}x^{i} - y^{i})^{2}$$

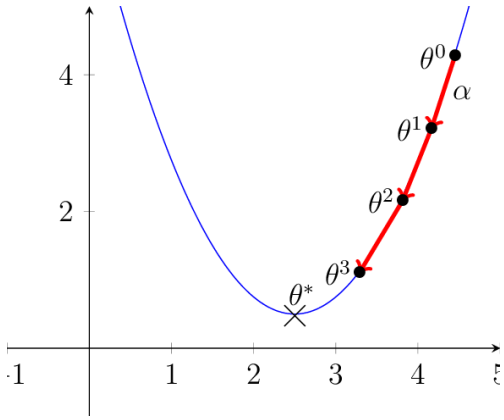
가장 기본적인 경사 하강법에 해당하는 배치 경사 하강법(Batch Gradient Descent)부터 알아보자. 경사 하강법에서 가장 중요한 것은 경사, 즉 그래디언트를 계산하는 것이다. 우리가 사용할 비용 함수인 평균 제곱 오차에 대한 파라미터 θ의 그래디언트는 편도함수(Partial derivative)를 이용하여 구할 수 있다. 편도함수의 수식은 아래와 같다.

$$\frac{\partial}{\partial\theta_j}MSE(\theta) = \frac{2}{m}\sum_{i=1}^{m}(\theta^{T}\cdot x^{i} - y^{i})x_j^{i}$$

이 식을 그래디언트 벡터를 통해 나타내면 다음과 같고, 이 식을 통해 비용함수가 최소화되는 점을 찾을 수 있다.

$$\nabla_\theta MSE(\theta) = \begin{bmatrix} \frac{\partial}{\partial\theta_0}MSE(\theta) \\ \frac{\partial}{\partial\theta_1}MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n}MSE(\theta) \end{bmatrix} = \frac{2}{m}\mathbf{X}^{T}\cdot(\mathbf{X}\cdot\theta - \mathbf{y})$$

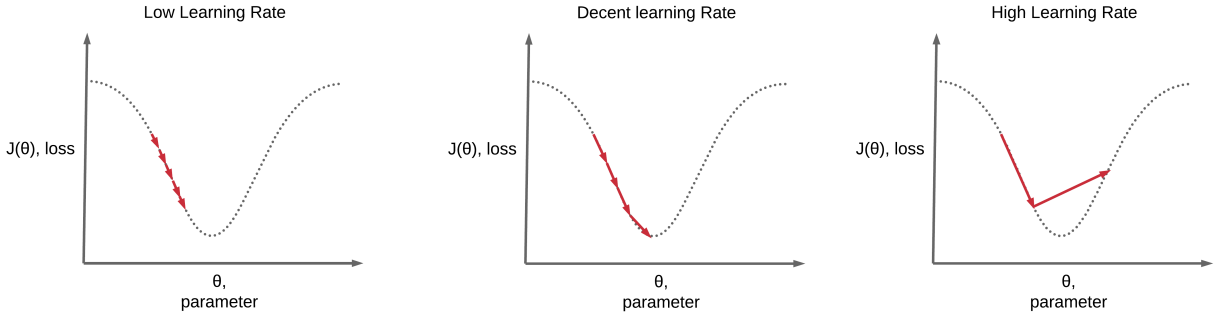
아래의 그림에서 θ^{*}은 비용함수가 최소가 되는 점으로, θ^{0}에서 시작하여 θ^{*}까지 $\theta^{1}, \theta^{2}, \theta^{3}$으로 나아갈수록 기울기가 점점 감소하는 것을 볼 수 있다.



이를 수식으로 나타내면 다음과 같다.

$$\theta^{d+1} = \theta^d - \eta \cdot \nabla_\theta MSE(\theta)$$

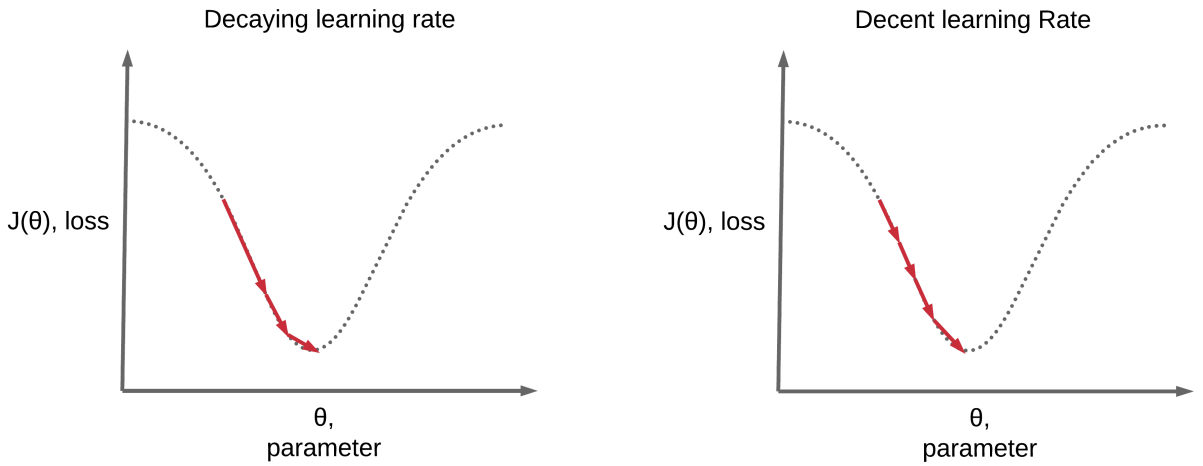
여기서 η란 학습률(learning rate)를 나타내는데, 이는 경사 하강법에서 가장 중요한 하이퍼파라미터(hyper paramter) 중 하나이다. 학습률은 경사 하강법에서 보폭의 크기를 결정하는 것으로 사용자가 결정한 값에 따라 달라진다.



위 그림에서 왼쪽의 경우는, 학습률을 너무 작게 설정했다. 학습률이 너무 작으면 특정 반복횟수 내에서 최소점을 찾지 못하는 경우가 있다. 물론 반복을 계속하게 되면 최소점에 다다를 수 있겠지만 너무 많은 시간과 컴퓨팅 자원을 소비하게 된다.

반대로, 오른쪽의 경우, 학습률을 너무 크게 설정했다. 학습률이 크면 빠르게 최소점으로 다가갈 수 있지만 진행 도중에 최소점을 지나쳐버리는 사고가 발생할 수 있다. 이런 문제 때문에 여러 학습률을 설정해보면서 가장 적절한 학습률을 찾아야 한다.

경우에 따라서는 아래 그림과 같이 처음에는 학습률을 크게 설정한 뒤 점점 감소시켜 나가는 학습률 감소(Learning rate decay) 방법을 사용하기도 한다. 아래 그림에서 학습률 감소 방법을 사용했을 때에 고정된 학습률을 사용했을 때보다 더 적은 반복수로 최솟값에 다다르는 것을 볼 수 있다.



배치 경사 하강법의 가장 큰 문제는 매번 모든 데이터셋에 대해 그래디언트를 계산한다는 것이다. 인스턴스의 개수가 많은 데이터셋에 배치 경사 하강법을 적용할 경우 시간이 오래 걸리고 컴퓨팅 자원을 많이 소모하게 된다.

확률적 경사 하강법(Stochastic Gradient Descent, SGD)은 이러한 문제를 개선하기 위한 알고리즘이다. 확률적 경사 하강법에서는 매 스텝마다 한 개의 샘플을 무작위로 선택한 뒤 하나의 샘플에 대해 그래디언트를 계산한다. 반복마다 사용되는 인스턴스의 숫자가 하나뿐이기 때문에 훨씬 빠른 속도로 계산을 해낼 수 있다. 하지만 인스턴스를 랜덤하게 하나만 선택하기 때문에 배치 경사 하강법보다 불안정하다는 단점이 있다.

미니배치 경사 하강법(Mini-batch Gradient Descent)은 두 방법의 합의점에 해당하는 경사 하강법 방식이다. 전체 데이터셋을 사용하거나 하나의 인스턴스만을 선택하지 않고, 작은 샘플 세트를 구성하여 경사하강법 알고리즘을 진행해 나가는 방식이다. 정리하자면, 다음과 같다.

알고리즘	샘플 수가 클 때	특성 수가 클 때	외부 메모리 학습 지원	스케일 조정 필요	하이퍼 파라미터 수
최소제곱법	빠름	느림	X	X	0
배치 경사 하강법	느림	빠름	X	O	2
확률적 경사 하강법	빠름	빠름	O	O	>=2
미니배치 경사 하강법	빠름	빠름	O	O	>=2