GDSC - Titanic Machine Learining Models

- 지도학습 알고리즘
  - 분류 (Classification) : 샘플을 몇 개의 클래스 중 하나로 분류
  - 회귀 (Regression) : 임의의 어떤 숫자를 예측

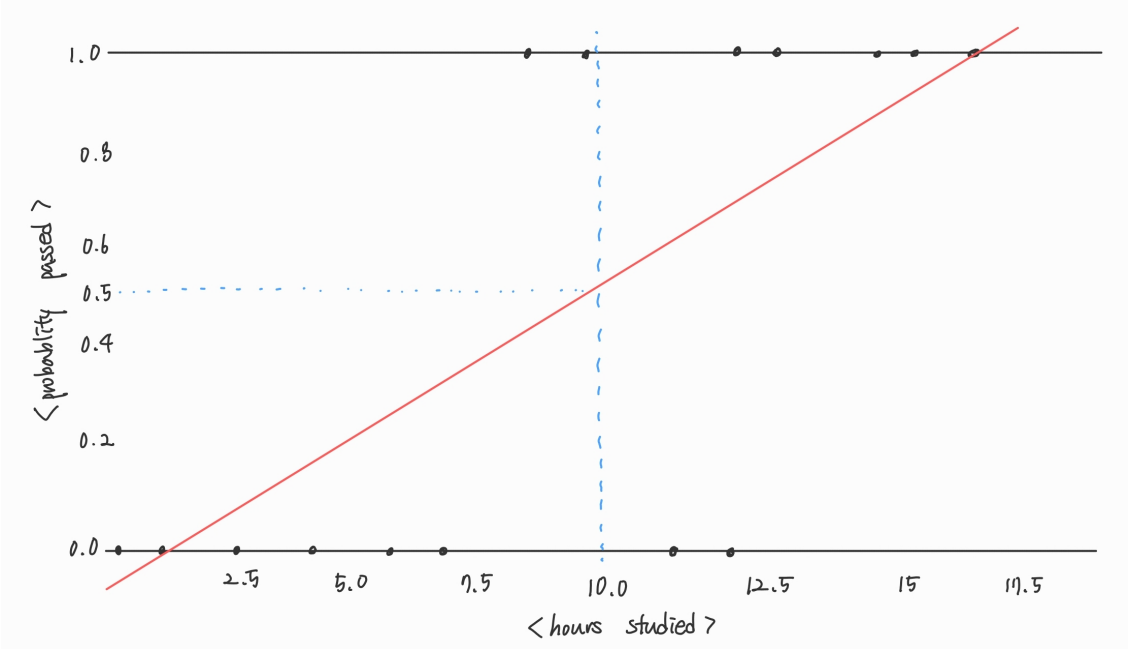
- **Logistic Regression**

  회귀를 사용하여 데이터가 어떤 범주에 속할 확률을 0 ~ 1 사이의 값으로 예측하고, 그 확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해주는 지도 학습

  ex) 스팸 메일 : 어떤 메일 받았을 때 스팸일 확률이 0.5 이상이면 spam 으로 분류, 확률이 0.5 보다 작은 경우 ham 으로 분류

  → 2진 분류 (binary claasification) : 2개의 범주 중 하나에 속하도록 결정

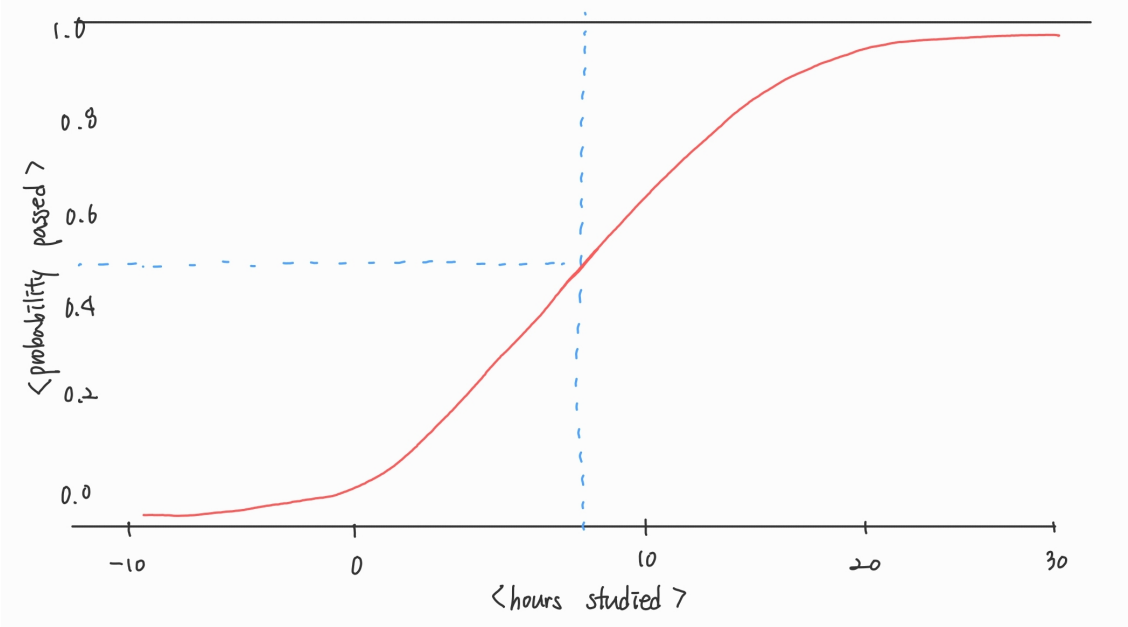
  - ex) 공부한 시간에 따라 시험에 합격할 확률이 달라짐
    - 선형 회귀를 사용



      공부한 시간이 적으면 시험에 통과 못하고, 공부한 시간이 많으면 시험에 통과한다는 식으로 설명 가능

      but 확률이 음과 양의 방향으로 무한대까지 뻗어 나감

    - 로지스틱 회귀 사용



시험에 합격할 확률이 0과 1 사이의 값으로 그려짐

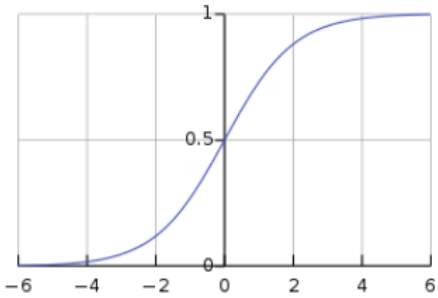
- 로지스틱 회귀에서 데이터가 특정 범주에 속할 확률을 예측하기 위해 거쳐야 할 단계
  1. 모든 속성 (feature)들의 계수 (coefficient)와 절편 (intercept)을 0으로 초기화
  2. 각 속성들의 값(value)에 계수(coefficient)를 곱해서 log-odds 를 구함
  3. log-odds 를 sigmoid 함수에 넣어서 [0, 1] 사이의 범위의 확률을 구함

- log-odds
  - odds = P(event occurring) / P(event not occurring) : 사건이 발생할 확률 / 사건이 발생하지 않을 확률
  - log-odds = log(odds)
  - log-odds를 구하는 방식
    - 선형 회귀에서의 예측값

      H(x) = Wx + b → 예측값 = 계수(coefficient) * 속성(feature) + 절편(intercept)
    - 로지스틱 회귀에서의 예측값

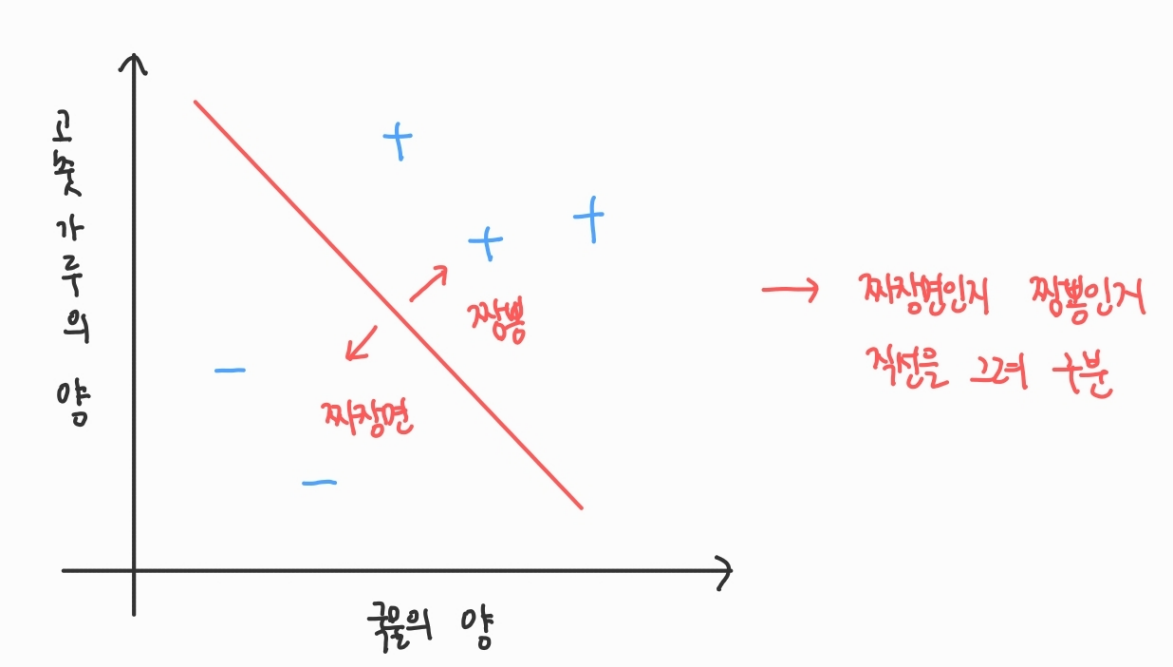
      z = b0 + b1x1 + … + bnxn 과 같은 선형결합 → ‘내적’ / ‘점곱’ 을 dot product 방식으로 log-odds 구함

      구한 log-odds 를 sigmoid 함수에 넣어서 0 과 1 사이의 확률로 만들어준다
      - sigmoid(x) = 1 / (1 + e^(-x))



- **Support Vector Machine**
  - 기본 작동 원리
    1. 두 카테고리 중 어느 하나에 속한 데이터 집합이 주어졌을 때,
    2. 주어진 데이터 집합을 바탕으로 새로운 데이터가 어느 카테고리에 속하는지 판단하는 비확률적 이진 선형분류 모델
    3. 만들어진 분류모델을 데이터가 사상된 공간에서 경계로 표현됨 → 그중 가장 큰 폭을 가진 경계를 찾음

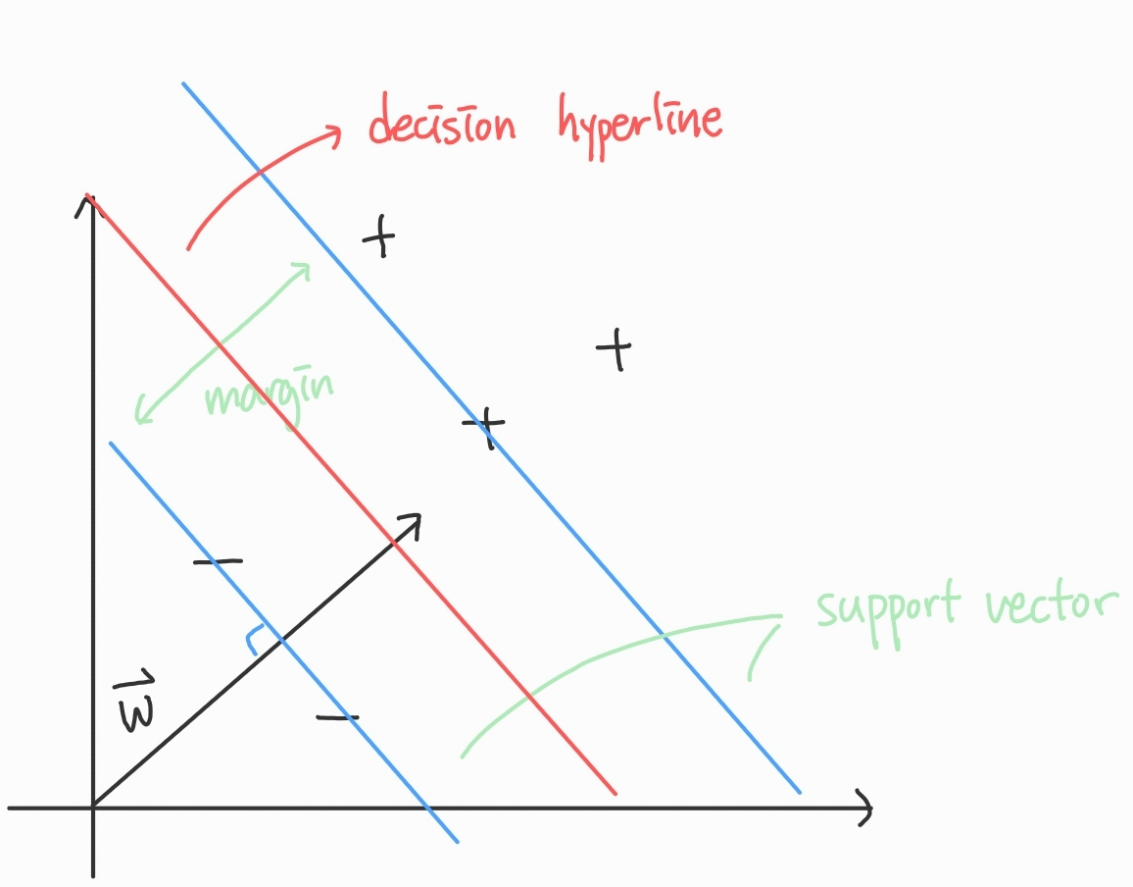
ex) 짜장면인지 짬뽕인지 직선을 그어 구분 (by. 국물의 양, 고춧가루의 양)



두 부류 사이의 여백이 가장 넓어지면 (margin 최대) 그 둘을 가장 잘 분류한 것

위와 같이 선형모델로 구분할 수도, 비선형모델로 구분할 수도 있음

- SVM 의 dicision rule



가중치 벡터 w 와 직교하면서 margin이 최대가 되는 선형을 찾아야 함

- support vector : 두 카테고리에 해당되는 data set 들의 최외각에 있는 샘플들

  supprot vector를 통해서 margin 을 구할 수 있음
- margin : support vector 를 통해 구한 두 카테고리 사이의 거리, 최대화하는 것이 목표
- decision hyperline : dataset을 분리하는 직선, 결정 초평면
  - 초평면 : n차원 공간에서의 초평면은 n-1차원의 subspace

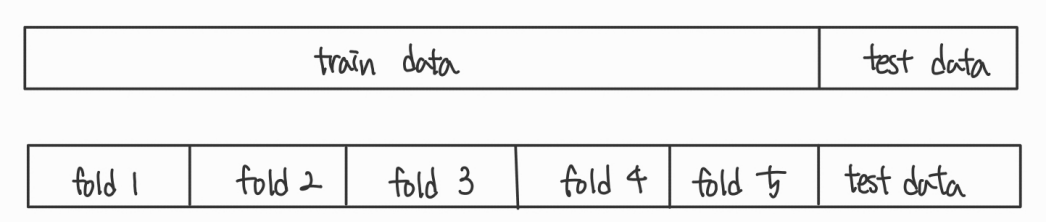
- train set 에서 최적의 decision hyperline을 찾아 test에 적용하였을 때 해당 데이터가 직선의 위 와 아래 중 어느 쪽에 있느냐에 따라서 분류하는 방법

- **KNN**

  K-NN : 데이터로부터 거리가 가까운 'k'개의 다른 데이터의 레이블을 참조하여 분류하는 알고리즘

  데이터 간 거리 측정에는 유클리디안 거리 계산법 사용 (피타고라스의 정리. 최소거리)
  - 장점
    - 단순하고 효율적
    - 훈련 단계가 빠름
    - 수치 기반 데이터 분류 작업에서 성능이 우수
  - 단점

- 모델을 생성하지 않기 때문에 특징과 클래스간 관계를 이해하는 데 제한적
- 데이터가 많아지면 분류가 느려짐
- 더미 코딩을 통한 명목 틍징 및 누락 데이터를 위한 추가 처리가 필요 (ex. male → 0, female → 1)
- 적절한 k의 선택이 중요하다
  - k 값이 너무 작은 경우
    - 거리가 가까운 데이터들만을 기준삼아 새로운 데이터 분류
    - 분류 기준이 too much strict
    - train set 에서의 정확도는 높지만, test set 에서는 에러가 높고 정확도가 낮음
    - overfitting (과대적합) 모델이 될 가능성이 높음
  - k 값이 너무 큰 경우
    - 분류 기준이 too much general
    - 새로운 데이터의 분류에 대해 정밀하게 판단하지 못함
    - test set 에 대한 정확도가 낮음
    - underfitting (과소적합) 모델이 될 가능성이 높음
  - k 값을 구하는 방법 (여러가지 중 하나)
    - 기본적으로 k는 홀수로 정하는 것이 좋음, 짝수로 k를 정하면 k/2 : k/2 의 상황이 생길 수 있어 어느 쪽인지 판단할 수 없는 상황이 생기기 때문
    - Cross-Validation (교차검증법)



기존에 나눈 train-set과 test-set에서 train-set을 다시 여러 개의 fold로 나누어, 그 안에서 1차적인 test를 하는 것

- 과정
  - k=1, fold는 5개의 부분으로 나눈다고 가정
    1. fold1 + fold2 + fold3 + fold4 로 모델 만들고 fold5로 test 하여 accuracy 냄
    2. fold1 + fold2 + fold3 + fold5 로 모델 만들고 fold4로 test 하여 accuracy 냄
    3. 5번의 시도로 5개의 accuracy 구하고 평균냄 → k=1 일때의 accuracy
    4. 위와 같은 방법을 k를 바꿔가며 accuracy 를 내고, accuracy 가 가장 좋은 k 를 선

- **Gaussian Naive Bayes (가우시안 나이브 베이즈)**

  기본적으로 조건부 확률을 이용한 모델

  ex) 테니스를 좋아하는 사람이 있다. 이 사람이 1. 날씨가 좋고, 2. 습도가 낮은 날에 테니스를 칠 확률은?

  → 1. 이 사람이 테니스를 많이 칠 수록, 2. 테니스를 쳤을 때 날씨가 좋고 습도가 낮은 경우가 많으면

  → 문제에 대한 답의 확률이 높아짐

- 베이즈 정리 (사전 확률로부터 사후 확률을 구함)

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

- 나이브 베이즈 종류
  - 다항 나이브 베이즈 : 설명변수가 범주형 변수일 때 사용
  - 베르누이 나이브 베이즈 : 설명변수가 범주형 변수일 때, 그 범주가 이진형일 경우 사용
  - **가우시안 나이브 베이즈** : 설명변수가 연속형 변수일 때 사용

    y 의 범주가 1 ~ C고 정해져 있을 때, y=c에서 X=x의 정규분포 표현식

$$P(X = x|y = c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} exp(-\frac{(x_i - \mu_c)^2}{2\sigma_c^2})$$

    조건부 독립으로 표현하면 다음과 같다

$$P(y = c|X1 = x1, ..., Xj = xj) = \frac{P(X1 = x1, ..., Xj = xj|y = c)P(y = c))}{P(X1 = x1, ..., Xj = xj)}$$

    분모는 항상 같은 조건을 나타내는 집합임

    분류 모델의 선택 편의를 위해 분자만 비교

$$P(X1 = x1, .., Xj = xj|y = c)P(y = c) = \prod_{j=1}^{J} P(Xj = xj|y = c)P(y = c)$$

    조건부독립에 정규분포 대입 (최종식)

$$= \prod_{j=1}^{J} \frac{1}{\sqrt{2\pi\sigma_c^2}} exp(-\frac{(x_i - \mu_c)^2}{2\sigma_c^2}) P(y = c)$$

    즉, 정규분포를 가정한 표본들을 대상으로 조건부 독립을 나타냄

    분모는 항상 같은 값을 갖기 때문에 분자의 값이 가장 큰 경우, 즉 확률이 가장 높은 경우를 '선택'하는 방식

- 장점
  - 간단하고 빠르며 효율적
  - 잡음과 누락 데이터를 잘 처리함
  - 데이터 크기에 상관없이 잘 작동
  - 예측을 위한 추정 확률을 쉽게 얻을 수 있음
- 단점

    - 모든 특징이 동등하게 중요하고, 독립이라는 가정이 잘못된 경우가 자주 있음
    - 수치 특징이 많은 데이터셋에는 이상적이지 않음

- **Perceptron**
    - 인간의 신경 세포 하나를 흉내낸 알고리즘 - 학습 데이터가 선형적으로 분리될 수 있을 때 적합
    - 다수의 신호 (input)을 입력받아서 하나의 output 신호를 출력
    - 뉴런에서 신호 전달하는 축삭돌기처럼 신호 전달하는 역할을 퍼셉트론에서는 weight가 함
    - Weight(가중치)
        - 각각의 입력신호에 부여
        - 입력신호와의 계산하고 신호의 총하비 정해진 임계값(theta)을 넘었을 때 1을 출력
            - 뉴런 활성화(activation)으로 표현, 넘지 못하면 0또는 -1 출력
        - 입력신호에 부여된 weight가 클수록 해당 신호가 중요한 신호임
        - weight값을 정하는 작업이 중요함
    - 퍼셉트론 학습 방법
        - 처음에는 임의로 설정된 weight로 시작 → 학습을 반복하면서 가장 데이터를 선형적으로 잘 분리하는 선을 찾음 (weight adjust 가중치 조정)
    - 퍼셉트론 수식

Output

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > \theta \Rightarrow 1$$

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq \theta \Rightarrow 0$$

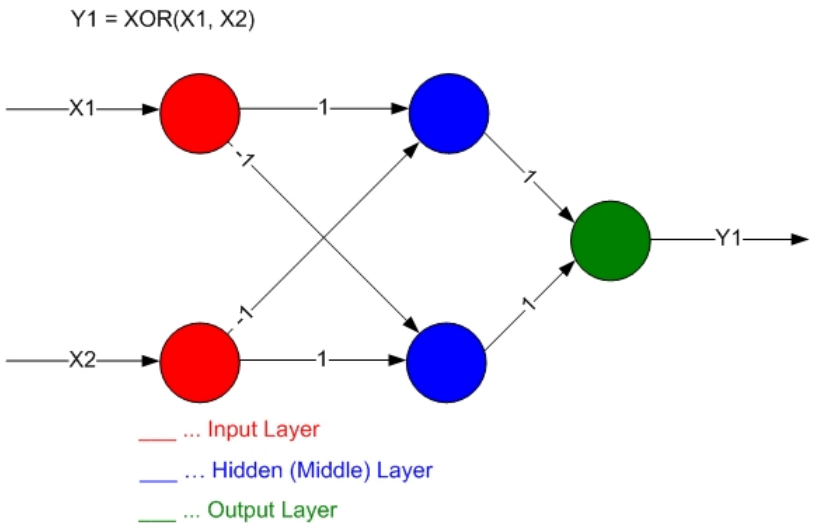
        - theta : 임계값
            - theta 값이 높을수록 분류의 기준이 엄격한 것
        - 세타θ를 -b로 치환하여 좌변으로 넘기면
            - b + w1x1 + w2x2 <0 => 0
            - b + w1x1 + w2x2 >=0 => 1

            - b : 편향 (bias)
                - theta 값이 높을수록 bias 값은 낮아짐 (-theta = bias 이기 때문)
                - 편향이 높을수록 모델이 간단해지는 경향이 있음(변수가 적고 일반화되는 경우) - underfitting의 위험
    - 가중치 : 입력신호가 결과 출력에 주는 영향도 조절하는 매개변수
    - 편향 : 뉴런이 얼마나 쉽게 활성화되느냐를 조정하는 매개변수

    - 한계
        - 선형분류만 가능하기 때문에 XOR 게이트와 같은 것은 분류 할 수 없음

x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0

- 다층 퍼셉트론을 통한 한계 극복
  - 여러 개의 layer로 선형 분류할 수 없는 것들을 선형 분류의 조합으로 풀어나감



- **Linear SVC**
  - Support Vector Machine 중 선형 모델

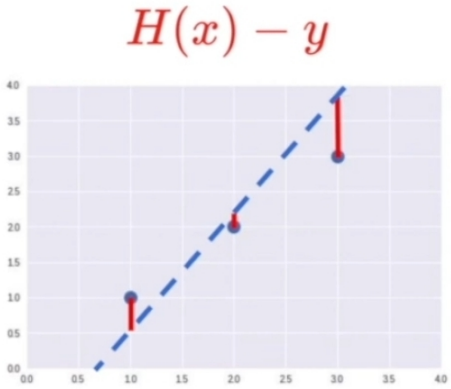
- **Stochastic Gradient Descent (확률적 경사 하강법)**
  - Gradient Descent Algorithm
    - cost 함수

Cost

How **fit** the line to our (training) data

$$\frac{(H(x_1)-y_1)^2+(H(x_2)-y_2)^2+(H(x_3)-y_3)^2}{3}$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx_i - y_i)^2$$



cost = loss (error)

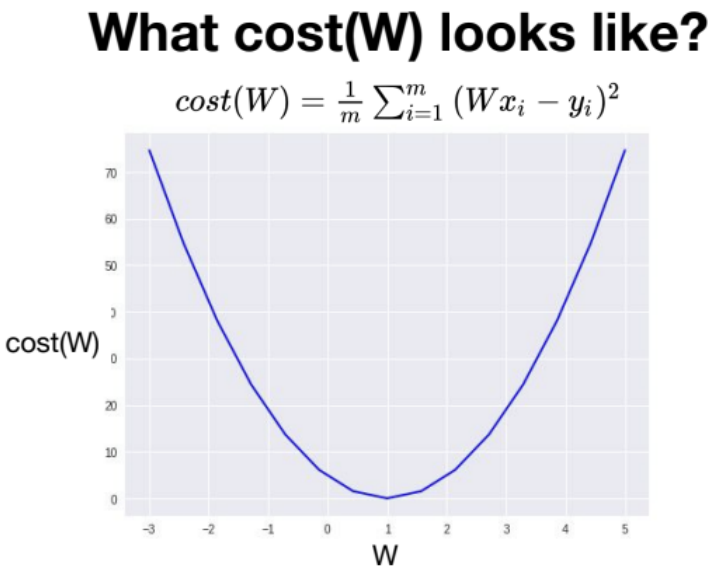
예측값과 실제 데이터와의 차이, -값과 +값이 모두 있기 때문에 제곱한 값을 사용함

What cost(W) looks like?

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx_i - y_i)^2$$

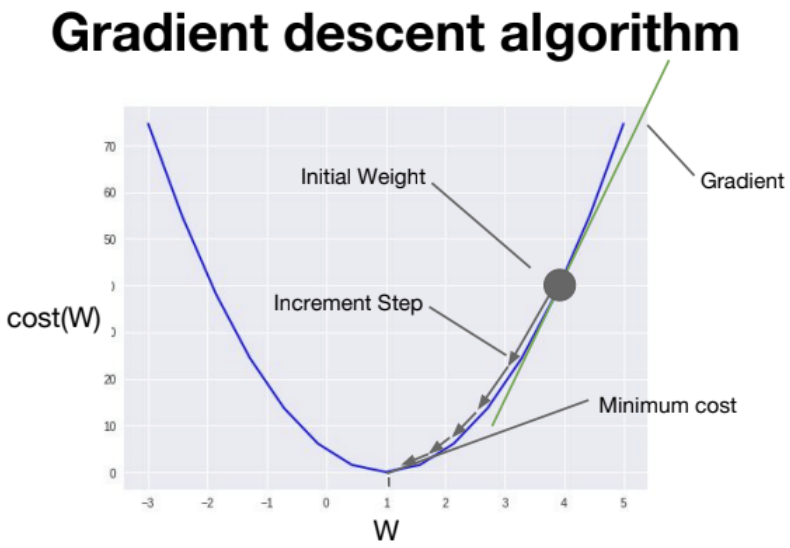
x	y
1	1
2	2
3	3

- W = 0, cost(W) = 4.67
  $\frac{1}{3}((0*1-1)^2+(0*2-2)^2+(0*3-3)^2))$
- W = 1, cost(W) = 0
  $\frac{1}{3}((1*1-1)^2+(1*2-2)^2+(1*3-3)^2))$
- W = 2, cost(W) = 4.67
  $\frac{1}{3}((2*1-1)^2+(2*2-2)^2+(2*3-3)^2))$
- W = 3, cost(W) = 18.67
  $\frac{1}{3}((3*1-1)^2+(3*2-2)^2+(3*3-3)^2))$

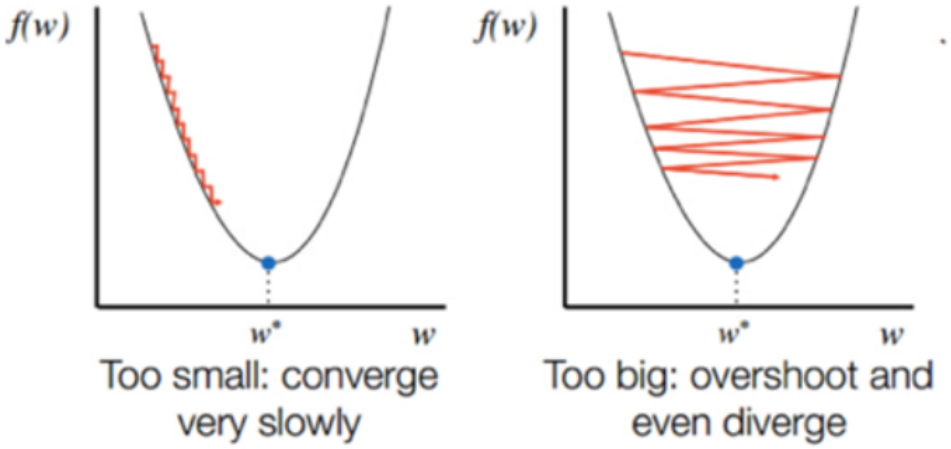


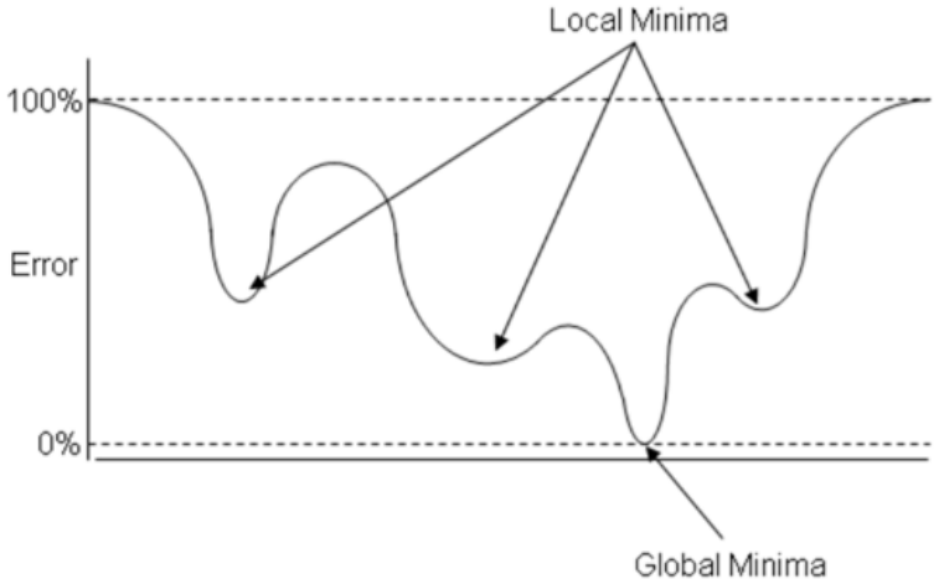
- cost 값을 최소화 하는 방법 : Gradient Descent Algorithm (경사 하강 알고리즘)

  그래프를 미분한 기울기 값을 사용하여 그래프의 경사를 따라 내려가면서 최저점을 찾도록 설계된 알고리즘

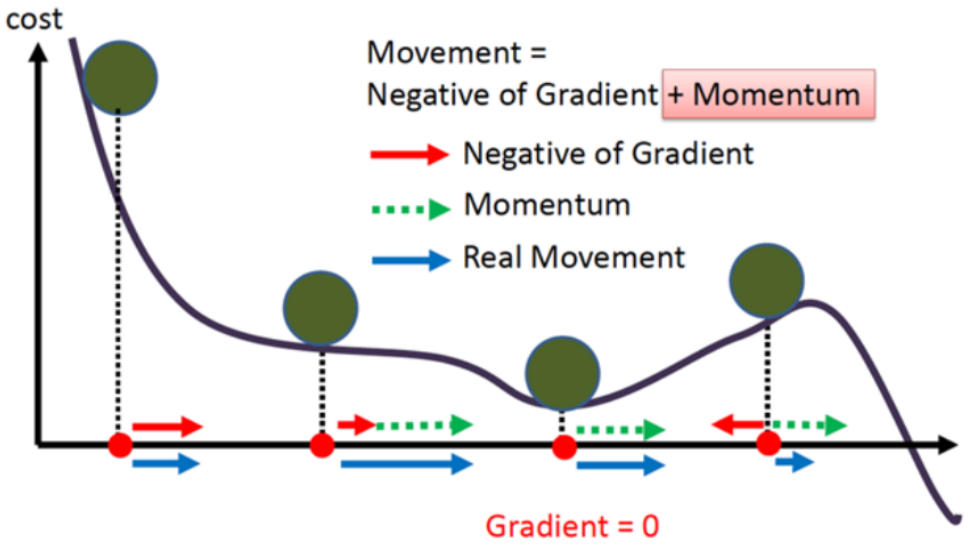


  - learning rate(학습률) : 학습 시 스텝의 크기





- 학습률이 작을 경우
  - 알고리즘의 결과가 0에 수렴하기 위해 반복해야 하는 값이 많음 → 학습시간이 오래 걸림
  - local minimum 에 수렴 가능
- 학습률이 클 경우
  - 학습 시간이 적게 걸림
  - 스텝이 너무 커서 global minimum 을 가로질러 반대편으로 건너뛰어 최소값에서 멀어질 수 있음
- 해결 방법 : Momentum



- 기울기에 관성을 부여하여 작은 기울기는 쉽게 넘어갈 수 있도록 만든 함수
- 지역 최소값을 탈출할 수 있게 해줌
- 모멘텀을 사용하지 않으면 아주 작을 언덕도 빠져나오는 데 오랜 시간이 걸리거나 빠져나올 수 없음

→ 대부분의 데이터 세트는 크기가 굉장히 큼, 이 경우 경사 하강 알고리즘을 사용하면 모든 데이터에 대한 손실 함수를 계산해야 해서 시간이 굉장히 오래걸림 (중복 데이터도 많을 것)

→ Stochasitc Gradient Descent (확률적 경사 하강법) 고안

- **Stochasitc Gradient Descent (확률적 경사 하강법)**
  - 데이터 세트에서 무작위로 선택한 데이터 하나를 가지고 학습
  - 매 step에서 딱 한 개의 샘플을 무작위 선택, 샘플에 대한 기울기 계산
  - batch size 가 1인 경사하강법 알고리즘
  - 반복이 충분하면 효과가 있지만 노이즈가 심함 → mini-batch gradient descent 알고리즘 고안

    → 노이즈는 많이 생기지만 지속적으로 무작위 데이터를 골라 학습시키면 훨씬 적은 데이터로 중요한 평균값을 추정할 수 있음 (모집단에서 샘플링하는 행위를 무수히 많이 했을 때 모집단의 평균에 수렴하는 원리와 같음)
  - epoch (에포크) : 훈련 세트를 한 번 모두 사용하는 과정
    - 1 epoch : 전체 학습 데이터셋이 신경망을 한 번 통과함
    - 에포크를 높일수록 적합한 파라미터를 찾을 확률이 올라감 (손실값이 내려감)
    - 지나치게 에포크를 높이면 train 데이터셋에 overfitting 되어 다른 데이터에 대해 제대로 된 예측 불가
- Mini-Batch Gradient Descent (미니 배치 경사 하강법)
  - batch size 를 줄이고, 확률적 경사하강법을 사용
  - ex) 학습 데이터 : 1000개, batch size : 100이면,

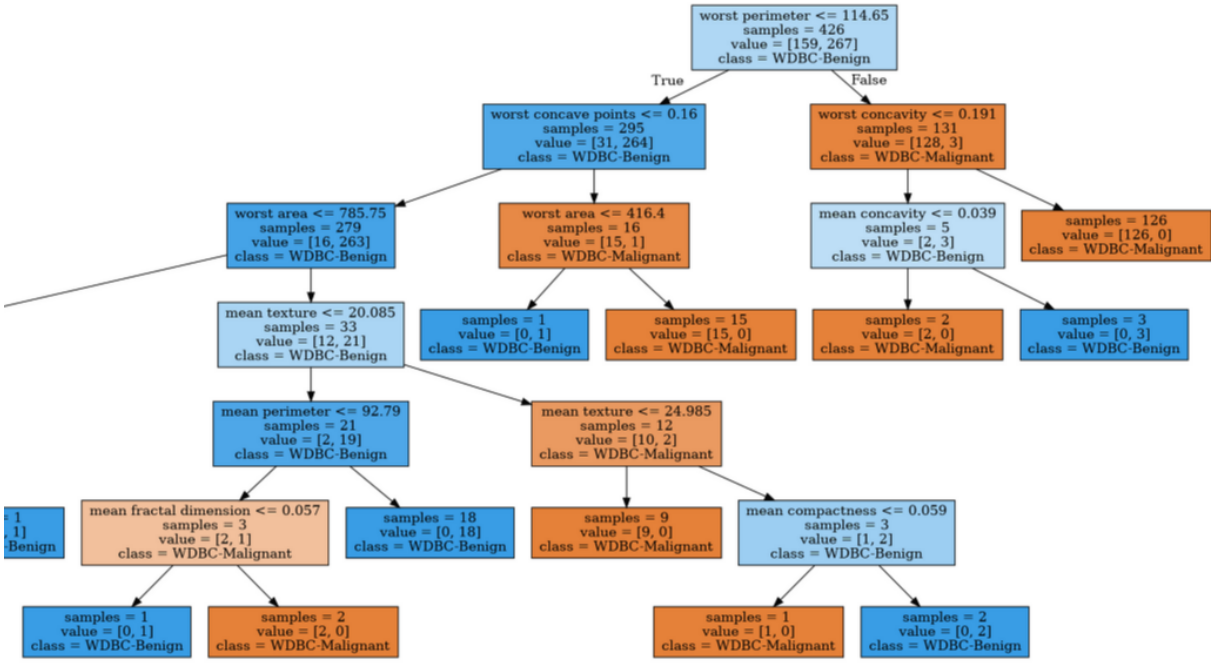
    총 10개의 mini batch가 나옴

    mini batch 하나당 한번씩 SGD (확률적 경사 하강법) 진행

    1 에포크당 총 10번의 SGD 진행

- **Decision Tree**

- flowchart 와 같은 구조
- 데이터에 있는 규칙을 통해 데이터셋을 분류/예측



루트 노드에서부터 분기를 이루면서 가장 많은 정보를 담을 수 있도록 계층적으로 영역을 분할해 감

→ 각 분할된 영역이 한 개의 타깃값(하나의 클래스 / 하나의 회귀 분석 결과)를 가질 때까지 반복

찾은 리프 노드들의 훈련 데이터 평균값이 최종 결과

- 노드 : 각 네모칸
- 리프 노드 : 마지막 노드, 자식이 없는 노드

- 트리 분할 방법
  - CART 알고리즘 (Classification And Regression Tree)
    - 지니 계수를 사용함

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

C = 사건의 개수, pi = 해당 split 에서 사건이 발생할 확률

- 지니계수는 데이터의 통계적 분산정도를 정량화해서 표현한 값
- 0 (complete equality) ~ 1 (complete inequeality)

  가장 낮은 지니계수를 가진 feature 가 결정트리에서 루트 노드가 되는 것 (low gini coefficient)

  - Iterative Dichotomiser 3 (ID3 알고리즘)

- 불순도를 엔트로피를 이용한 정보 이들으로 계산 - 독립변수가 모두 범주형일때만 가능
- 엔트로피 : 주어진 집합의 혼잡도
  - 서로 다른 값이 섞여있으면 1에 가깝고, 같은 값만 존재할수록 0에 가까움

$$Ent(D) = -\sum_{k=1}^{n} p_k \log_2 p_k \quad where \quad p_k = \frac{|k-class \;\; set \;\; in \;\; sample|}{|sample|}$$

  - D : 주어진 데이터들의 집합
  - k-class set in sample : D 에서의 k클래스에 속하는 레코드 수
  - |sample| : 주어진 데이터들의 집합의 데이터 개수
- 정보이득 : 분할 전 엔트로피와 분할 후 엔트로피의 차
- Ent(D)에서 해당 피처에 속하는 부분집합들에 대한 엔트로피 값을 뺌으로써 정보이득을 구할 수 있음
- 정보이득이 높은 노드를 우선적으로 선택

  - 장점
    - 알고리즘이 쉽고 직관적
    - 시각화 편리
    - 균일도만 신경쓰면 되기 때문에 feature 의 스케일링이나 정규화 필요 x
  - 단점
    - 과적합 발생 확률 많음
      - feature 가 많고 다양한 데이터 가질수록 트리의 길이가 커지고 복잡해짐 → 실제 상황에 유연하게 대처하기 어려움
      - 파라미터 튜닝으로 트리의 크기를 사전에 제한하는 것이 성능에 도움

- **Random Forest**
  - 의사결정 트리(decision tree)의 오버피팅 한계를 극복하기 위한 전략
  - 훈련을 통해 구성해놓은 다수의 나무들로부터 분류결과를 취합해서 결론을 얻는 방법
  - 몇몇의 나무들이 오버피팅을 보이더라도 다수의 나무를 기반으로 예측하기 때문에 그 영향력이 줄어듦
  - 배깅 (bagging) : 각 나무들을 생성하는 방법
    - ex) 학습 데이터가 1000개의 행이 있다고하면, 임의로 100개씩 행을 선택해서 의사결정 트리를 만듦
    - 중복 허용(트리를 만들고, 트리를 만든 행을 다시 가방에 집어넣는 방식)
    - bagging features : 트리를 만들 때 사용될 속성들을 제한 → 각 나무에 다양성

      ↔ 의사결정 트리에서 트리를 만들 때는 정보 획득량이 가장 많은 속성을 선택하여 그걸 기준으로 데이터 분할 → 각 분할에서 전체 속성들 중 일부만 고려하여 트리 작성하도록 하는 전

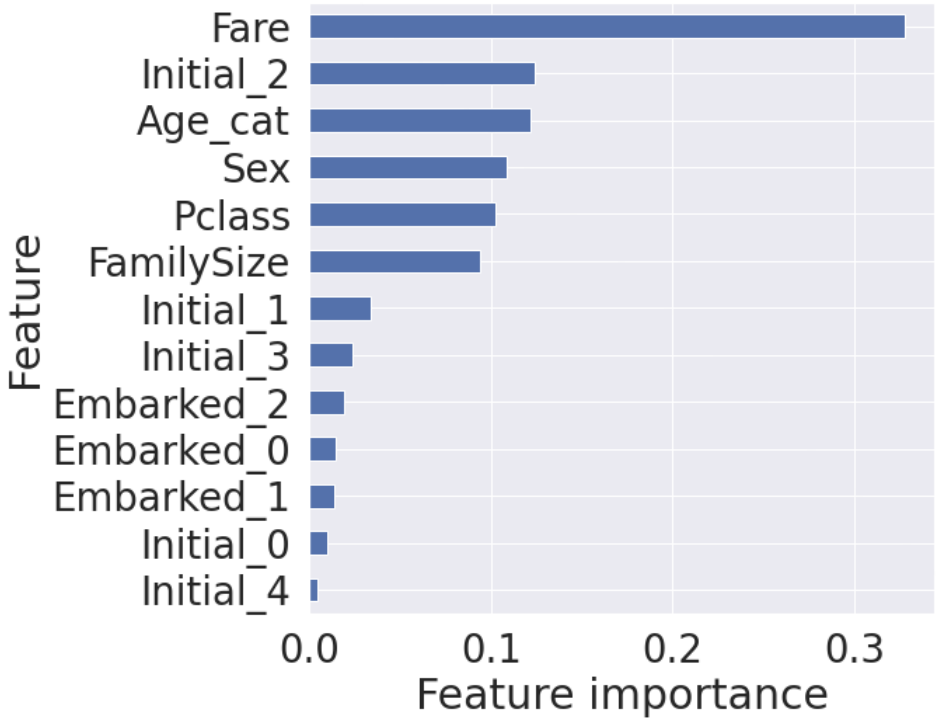
	Model	Score
3	Random Forest	86.76
8	Decision Tree	86.76
1	KNN	84.74
2	Logistic Regression	80.36
7	Linear SVC	79.12
5	Perceptron	78.34
0	Support Vector Machines	78.23
6	Stochastic Gradient Decent	75.20
4	Naive Bayes	72.28

- 선형으로 데이터들을 분리해야하는 logistic regression 부터는 score 가 갑자기 떨어진 것을 확인할 수 있음
- titanic과 같은 이진분류에서는 decision tree 를 활용하는 방법이 가장 효과적인 것을 알 수 있음
- Stochasitc Gradient Decent 에서는 아마 local minimum에 빠진 데이터들이 있을 것
- naive bayes 의 경우 수치 특성이 많은 데이터에 적합하지 않고, 무엇보다 모든 특징이 동등하게 중요하고 독립이라는 가정 때문에 각각의 feature 가 다른 중요도를 가지는 titanic 의 경우에 부합하지 않은 듯 하다

```
from pandas import Series

feature_importance = model.feature_importances_
Series_feat_imp = Series(feature_importance, index=df_test.columns)
```

```
plt.figure(figsize=(8, 8))
Series_feat_imp.sort_values(ascending=True).plot.barh()
plt.xlabel('Feature importance')
plt.ylabel('Feature')
plt.show()
```



실제로 feature 별로 중요도가 다르기 때문에 이 점이 고려되지 않아 가장 정확도가 낮게 나온 것 같다