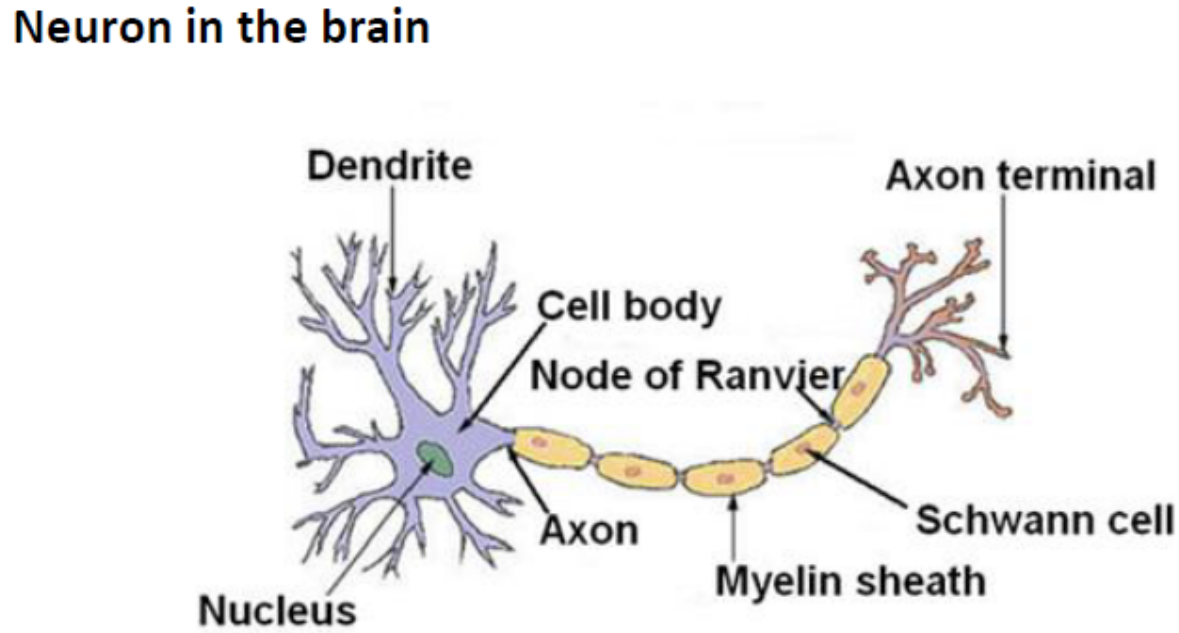
Neural Net

배경 : Non-linear hypothesis를 linear regression이나 logistic regression으로 제안한다면 feature수에 의해서 overfitting과 같은 부작용이 발생할 수 있다.(이부분에서 그냥 제곱텀 세제곱텀을 임의적으로 조절하면 overfitting 되지 않는 선에서 충분히 non-linear boundary를 만들어낼 수 있지 않을까? 생각을 해본다. 하지만 이 때 발생하는 문제가 그 feature을 어떻게 뽑아낼 것인가 하는 것이다. 뉴럴넷 알고리즘 자체가 feature까지 학습하여 가장 적절한 feature에 대한 가중치 값을 가지므로 이 방법이 non-linear boundary에서는 효과적이라고 할 수 있다.) 그래서 조금 더 효율적인 non-linear boundary classification을 구현하기 위해서 뇌의 신경세포를 모방하는 알고리즘인 neural net이 등장했다. 고 한다.

인간의 뇌에 있는 뉴런을 모방하자.

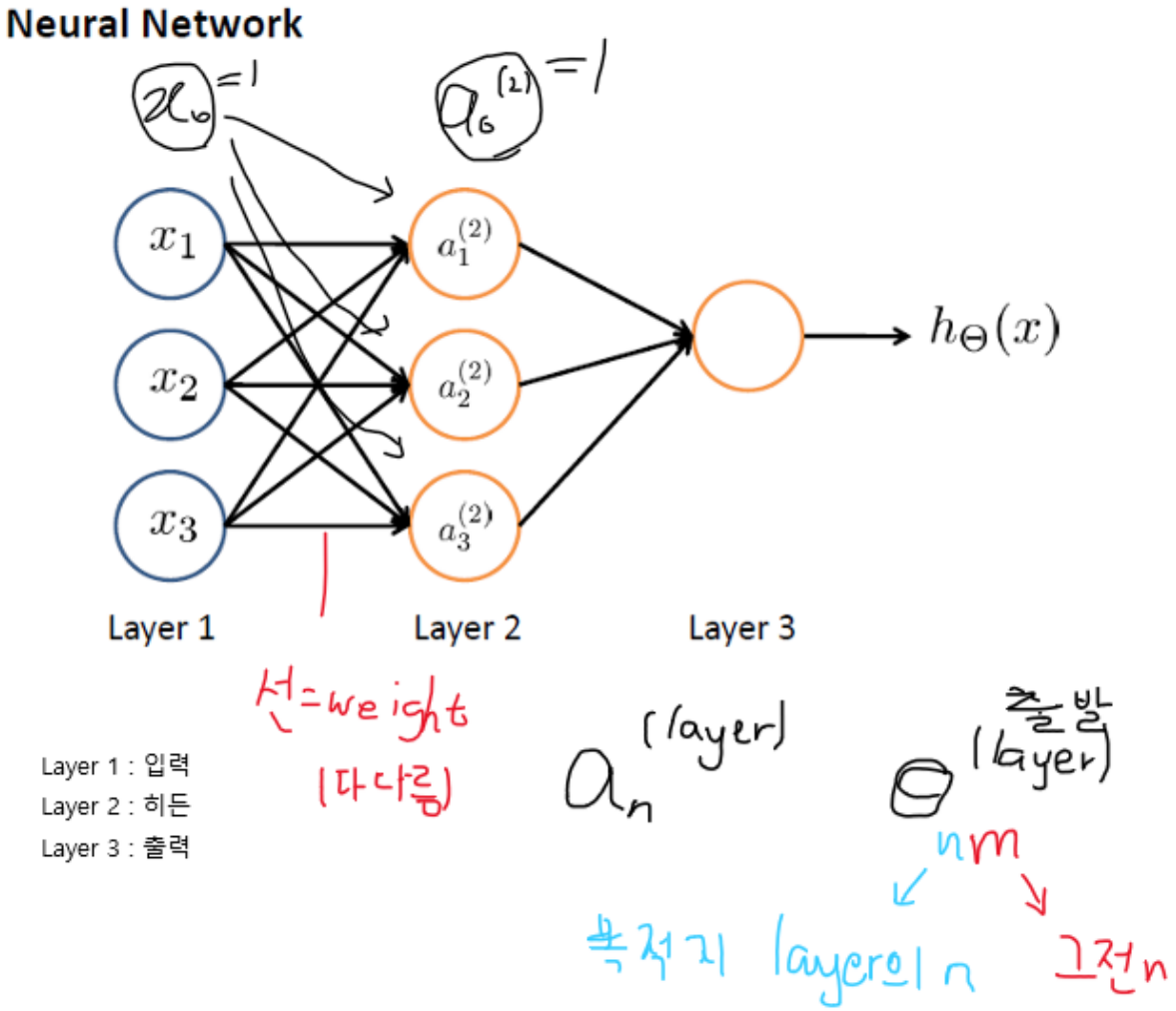


Dendrite : 입력 신호를 받아와서

Nucleus : 처리하고

Axon : 다른 뉴런으로 보내줌

Anyway, 뉴런을 수학적 모델로 나타내면 다음과 같다. 동그라미 하나가 뉴런이고 이 하나하나(input과 ouput 제외) 가 logistic regression, 즉 activation function이 된다. (이 때, output layer를 제외하고는 1의 값을 가지는 bias 텀을 추가하는데 이는 식을 일반화를 위해 하는 작업이다.)



가령, 2번째 layer의 첫번째 모델을 다음과 같이 표현할 수 있다.

z = 세타(x) (=

$$\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$$

)

a = sigmoid (그런데 시그모이드는 layer가 깊어질수록 gradient 값이 0으로 간다. 그래서 activate함수로 나중에는 ReLu를 제안한다.)

$$z_1^{(2)} = \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$$

$$a_1^{(2)} = g(z_1^{(2)})$$ -> 2번째 layer의 첫번째 모델

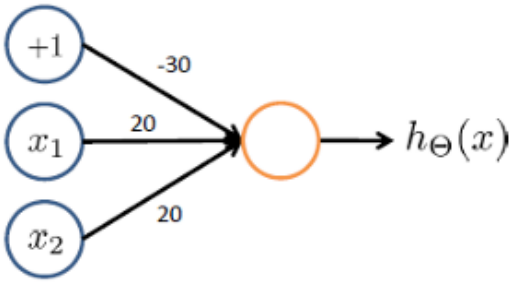
cf) 딥러닝 : layer 수가 많은 것

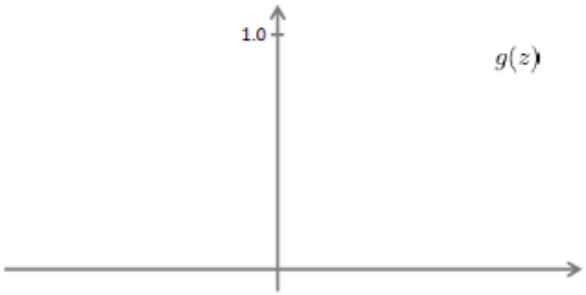
neural net으로 논리연산자를 구현해본다면 다음과 같은 weight값을 설정해줌으로써 And 연산자 역할을 수행할 수 있다.

Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$





x_1	x_2	$h_\Theta(x)$
0	0	0 (-30)
0	1	0 (-10)
1	0	0 (-10)
1	1	1 (10)

$$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$$

$x_1 = 0 \text{ and } x_2 = 0 \text{ then } g(-30) \approx 0$

$x_1 = 0 \text{ and } x_2 = 1 \text{ then } g(-10) \approx 0$

$x_1 = 1 \text{ and } x_2 = 0 \text{ then } g(-10) \approx 0$

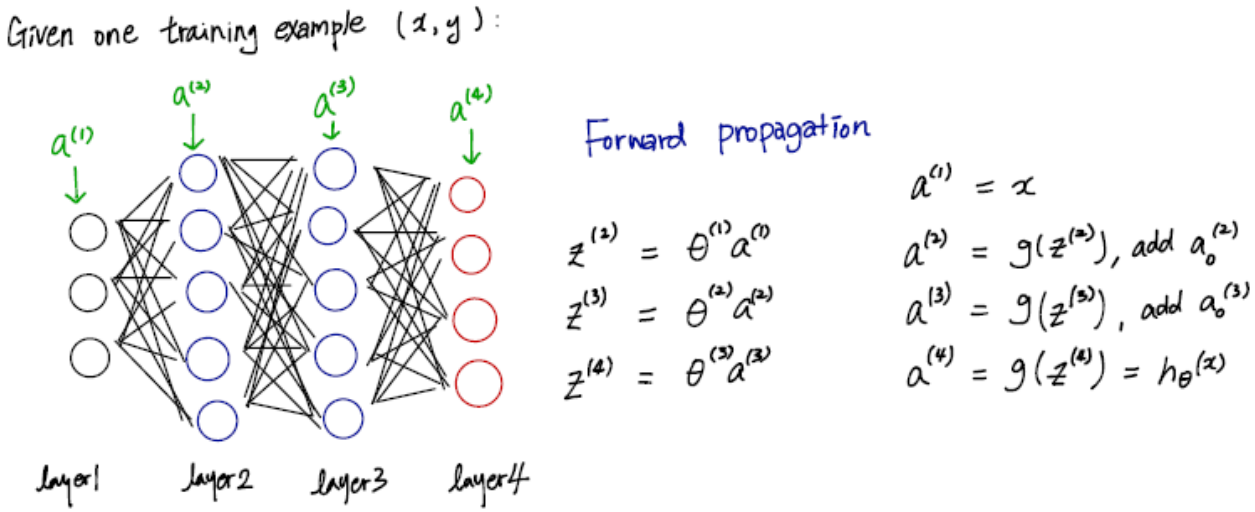
$x_1 = 1 \text{ and } x_2 = 1 \text{ then } g(10) \approx 1$

제대로 된 classification을 하는 hypothesis(최종적으로 예측한 값으로, n개로 classification하게 된다면 n개의 hypothesis가 나오게 된다.)는 늘 그래왔던 것처럼 적절한 세타 값을 찾는게 중요하다. 즉, cost function을 최소화하는 세타 값을 구해야한다. 이 때 나오는게 바로 forward propagation과 backward propagation이다.

Gradient Computation

Forward Propagation

: input layer에서 시작해 output이나 error 따위를 구하는 것.



다시 우리가 무엇을 하는지 정의하고 가겠다. 우리는 실제값과 예측값의 차이인 cost값을 최소화하기 위해서 layer들 사이의 세타값을 적절한 값으로 구해야한다. forward propagation의 경우, 예측값을 구하기 위해서 input layer부터 output layer까지 이전 layer에서 계산된 값을 받아서 다음 layer가 사용한다.

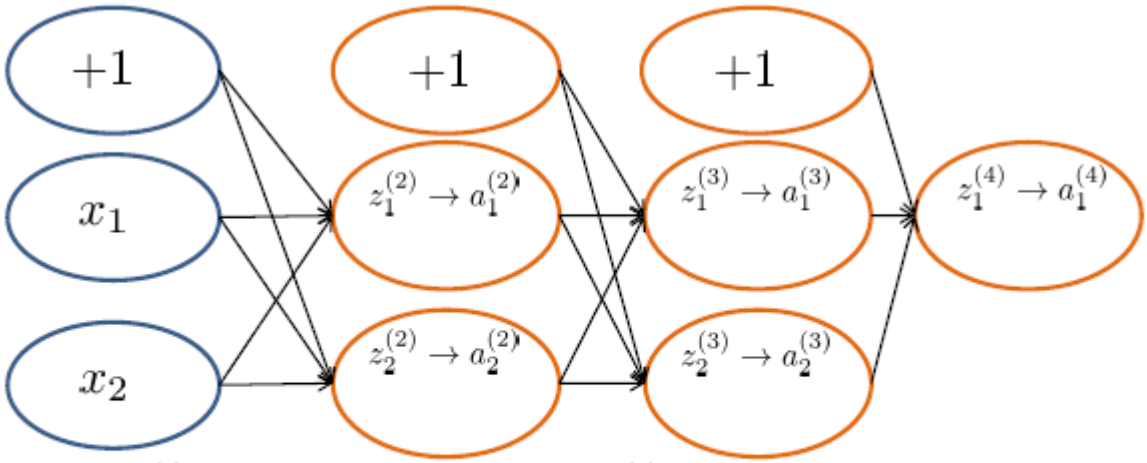
Backward Propagation

: output layer에서시작해 error값을 구하는 것.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

유도과정은 다음과 같다 :

https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/



$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).

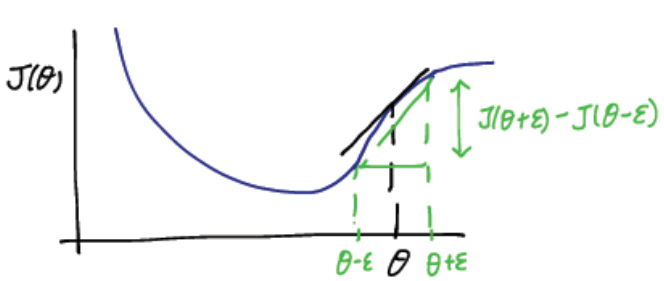
Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost(i)}$ (for $j \geq 0$), where

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

Gradient Checking

: 계산 방식이 워낙 복잡하다보니 미분해서 구한 기울기값이 직접 기울기를 계산한 값과 같은지를 확인해보는 것.

Numerical Estimation of Gradients



$\theta \in \mathbb{R}$

$\frac{\partial}{\partial\theta} J(\theta) \simeq \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$, $\varepsilon = 10^{-4}$

하지만 이 작업은 ' example수 * 세타수 ' 가 되어 비용이 많이들어 일부분만 해보고 판단한다.

cf) 만약 세타를 초기화할 때 모두 0으로 한다면 어떻게 될까?

=> 값이 다 같게 나온다. 입력, weigh, 결과, error 같아서 업데이트해도 같은 값으로 update된다.

Neural net 워크 플로우 정리

1. weight를 랜덤하게 초기화한다.
2. forward prop로 h(세타)를 구한다.
3. J(세타)를 구한다.

4. backward prop로 $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ 값을 구한다.
5. gradient checking로 기울기를 확인한다.
6. 최적화 방법(backward prop)으로 최적의 세타 값을 구한다.

Decision Tree

: 어떤 변수를 기준으로 classification하는 것.

// 한계 : decision boundary가 직선밖에 안된다.

1. IDE

기준 : entropy와 information gain

entropy는 혼합도(0: 낮다, 1: 높다)

information gain은 잘 나눠진 정도(상위노드 엔트로피 - 하위노드 엔트로피) 으로, 감소되는 entropy양이다.

entropy가 낮을 수록(가령,

예시1 - 밖에서 노는지 안노는지를 예측할 때 날씨로 하면 sunny일 때는 반반으로 1이고, rainy일때는 0이다. 이 때, rainy의 비중이 높다면 entropy가 엄청 낮아진다. 결국 날씨는 구분하는 기준으로 적합하다는 것이다.

예시2 - 월급이 높고 낮음으로 대출심사 통과패스를 결정한다고 할 때, 높을 때는 빌려주고, 낮을 때는 안 빌려준다. 이는 entropy가 엄청엄청 낮다. 높은 경우 낮은 경우 둘다 결정이 랜덤하지 않고 거의 정해져있다. 결국 이건 좋은 것이다.)

(그러니 entropy는 뭘로 봐야하겠는가! => 놀지 안 놀지도 잘 모르는 sunny의 비중이 크다는 것이고 이는 결코 좋은 분류기준이 아니다.) information gain이 높은것(A라는 기준에서 나뉠 수 있는 범주에서 혼합도가 낮은 것)이 좋은 것이다.

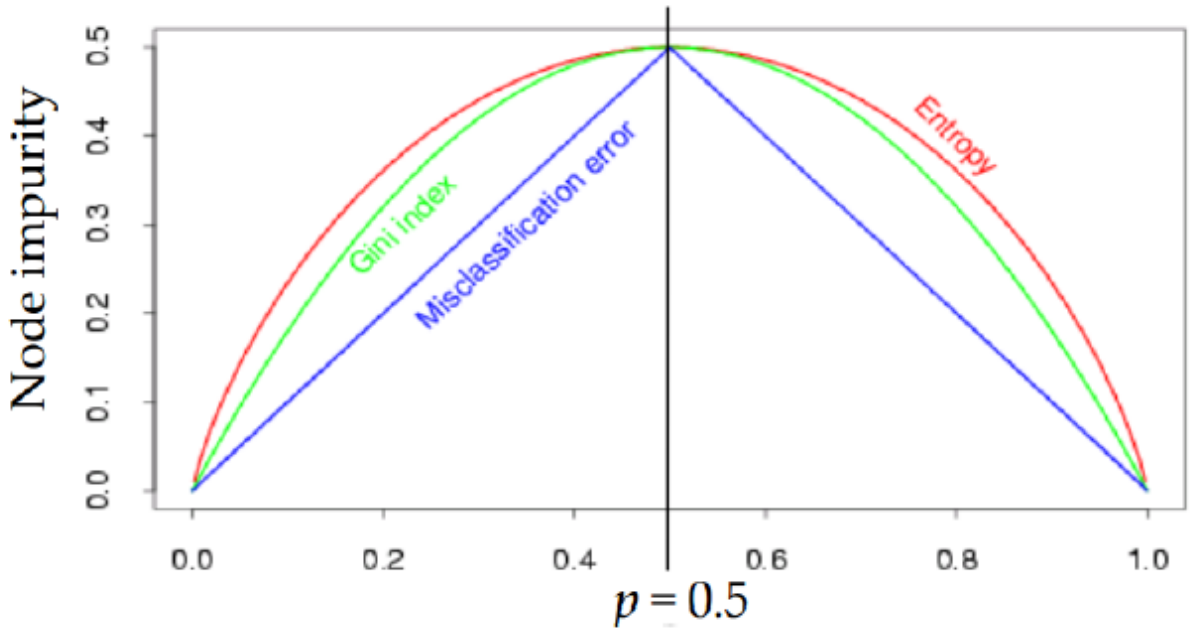
2. CART

기준 : gini index

어느 기준으로 분류를 했을 때, 한쪽을 기준으로 계산한다. '내가 yes라고 생각한 한쪽 면에 다른 불순물(no)이 있는 정도' 이다. 계산은 1 - (어느 한 결과의 비중)제곱과 - (다른 불순물의 비중)의 제곱으로 계산한다. 이는 만약 yes와 no를 잘 나눴다면 0이 됨을 뜻하고 gini index는 낮을수록 좋다. gini index의 값의 범위는 매번 달라지는데, 0부터 1-1/k 값이 최대값이다.(k는 classification 종류 갯수)

*entropy & gini index의 차이

: 거의 없다.



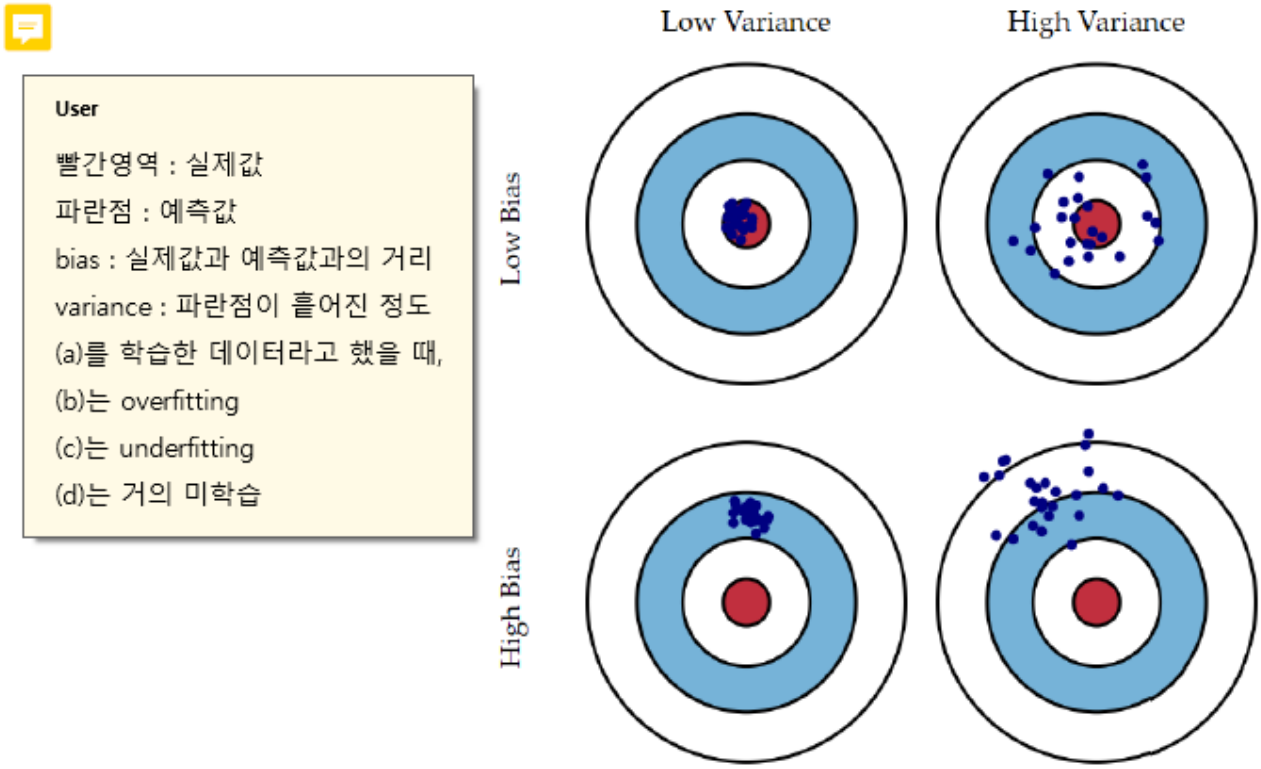
Bagging & Boosting

: 앙상블기법 아이디어이다. (underfitting & overfitting 문제를 해결하는데 초점)

앙상블 : 여러의견을 통합하거나 여러가지 결과를 합치는 방식

그 전에, bias(편향)과 variance(분산)을 알아야한다. 일단 기본적으로 이 둘은 error(안좋은 것) (이)다.

Bias and Variance



bias는 target을 잘못 생각하는 것 = 그냥 학습이 덜 된것이다. 즉, underfitting문제를 발생시키는 것으로, 잘못된 가정을 한 것이 원인이다. (추정값의 평균과 참 값들간의 차이) 그냥 학습이 덜된것.

variance는 예측값의 산포도. overfitting문제를 발생시키는 것.(추정 값의 평균과 추정 값들간의 차이)

cf) Overfitting = Bias-Variance Trade Off

Bagging

: bootstrap aggregating의 줄임말로, 원본 DATA에서 샘플을 뽑아서 모델링하고, 또 샘플을 뽑아서 모델링하고, 또 샘플을 뽑아서... 이것들의 집계로 decision하는 것이다. (여러 환경 전문가들을 데려다 놓고 해일이 올 것인지를 판단하게 해서 많은 쪽으로 결정을 내리는 것이다.)

=> 배깅은 overfitting(원인 : 높은 variance-새로운데이터를 예측 못하니까 산포도가 높다) 문제를 해결해준다.

ex) 디지즌 트리, 랜덤 포레스트

Boosting

: 원본 DATA에서 '샘플링->모델링' 을 반복하되 샘플링해서 모델링한 후 잘 예측하지 못한 점에 대해서 가중치를 주어 다시 샘플링한다. 아마 그 점은 디시즌 바운더리 근처의 점들일 것이다. 그래서 제대로 학습이 안되는 부분의 문제를 해결한다.

=> 부스팅은 underfitting(원인 : 높은 bias-학습이 덜된거) 문제를 해결해준다.

ex) AdaBoost, GBM, Xgboost, Light GBM

Bagging v.s. Boosting

비교	Bagging	Boosting
특징	병렬 앙상블 모델 (각 모델은 서로 독립적)	연속 앙상블 (이전 모델의 오류를 고려)
목적	Variance 감소	Bias 감소
적합한 상황	복잡한 모델 (High variance, Low bias)	Low variance, High bias 모델
대표 알고리즘	Random Forest	Gradient Boosting, AdaBoost
Sampling	Random Sampling	Random Sampling with weight on error

이렇게 기억하면 쉽겠지!

: 배깅은 Bag에 욕심껏 이것저것 넣어서 다 챙기니까 학습은 잘 됐는데 overfitting.

: 부스팅은 급하게 하느라 학습 덜됐어. underfitting.

Random Forest

: 수많은 디시즌 트리들이 모여서 숲을 이뤄 많은 트리들이 주장하는 걸 받아들인다. 배깅의 방법을 사용한 모델로서, observation(example)만 샘플링하는 것이 아니라, feature까지 샘플링해서 모델링하여 다수결로 결정한다.

1. 원본 DATA에서 몇 개의 트리를 만들 것인지 결정한다.
2. feature 들을 샘플링해서 트리를 만든다.
3. 트리들의 평균을 내서 의사를 결정한다.

그래서 변수를 어떻게 결정할지가 중요하다.

Proximity Measure

: 유사도, 근접도(거리) 측정, ex) 유클리디안, 맨하탄 등

[배깅과 부스팅 참고자료]

https://swalloow.github.io/bagging-boosting

Clustering - unsupervised learning

: label이 없는 상황에서 학습을 해야하기 때문에, clustering이 필요하다. clustering에는 크게 두가지 종류가 있는데, centroid를 기반으로 할 것인지, density를 기반으로 할 것인지로 구분된다.

cf) 클러스터링이란?

: 비슷한 것들을 모으는 것. 한 클러스터 안에서는 비슷할 수록 좋고, 클러스터들끼리는 성질이 다를수록 좋다. wss(within sum of square)값이 낮다는 것은 산포도가 낮다는 것이고 비슷한 것들끼리 잘 모여있다는 것을 의미한다. 분류 종류에 해당하는 k값이 커지면 커질수록 wss값은 낮아진다. 이 부분은 뒤에 설명이 되어있다.

Centroid-based clustering

: 어떠한 데이터들의 센터벡터를 기준으로 묶는 것. 노이즈에 취약(센트로이드 값이 평균값이기 때문에)

ex) K-means, RAM, CLARA,

Density-based clustering

: 특정 밀도(어떤 area내에 얼마나 있는가)를 기준으로 묶는 것.

ex DBSCAN

<구체적으로 확인>

1. k-means (centroid)

: k개로의 클러스터화 (k는내가 나눌 clustering의 개수(k개의 그룹))

: 프로세스

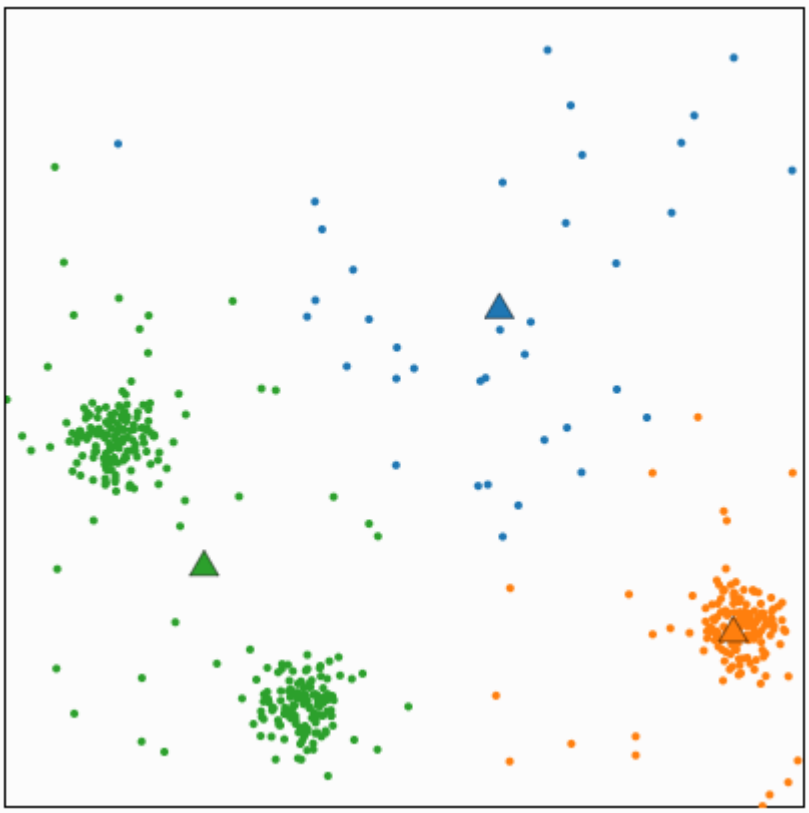
1. 시작하는 데이터 포인터-seed point (랜덤하게 지정)를 지정 - centroid
2. 다른 데이터 포인터 간의 거리를 계산
3. 거리의 평균에 해당하는 center point로 centroid를 이동시키고 (기존 seed point의 위치가 바뀜)
4. 이를 centroid가 변하지 않을 때까지 반복시키면 cluster가 나뉘게 됨.

특징 : 심플하다. 하지만 단점이 seed point가 local minimum에 빠져서 군집을 효율적으로 못하는 경우도 있어서

global optimize하고 있는지 확인. 또 적절한 k값을 알아야 제대로된 군집을 할 수 있다. 노이즈에 취약하다.

(하지만 실생활에서는 k값을 알고 있는 경우가 많음. 가령, 남녀 구분에 관해서라던지)

stanford.edu/class/ee103/visualizations/kmeans/kmeans.html



2. DBSCAN (density)

: 어떤 포인터를 두고 거기 어느 반경 안에 포인터가 몇 개가 있는지를 가지고 밀도를 계산

[용어]

- core point : 정해진 minpoint이상을 포인트를 가지는 포인트
- border point : 정해진 minpoint보다 작은 포인트를 가지는 포인트
- noise point : 둘다 아닌 포인트

특징 : k를 지정하지 않음. 임의의 모양을 가지는 데이터 형성에 유리하다. 멀러떨어져있는 데이터(노이즈)에 영향을 덜 받는다.

3. hierarchical clustering

: 클러스터의 클러스터화

특징 : 클러스터와 클러스터의 거리를 잴 때, 기준 포인트를 무엇으로 잡을지 결정해야한다. (ward's procedure방법이 wss가 줄어드는 방법으로 가장 잘 된다. distance matrix가 필요하다.)

<평가>

1. WSS

: 클러스터 내의 포인트에 대한 산포도. k(분류 갯수)가 커질수록 wss값은 작아진다.

2. ch index

: 클러스터 간의 거리 / 클러스터 내의 데이터 간의 거리 , 한 마디로, '얼마나 잘 나뉘었나.' 이며, 클수록 좋다.

(밑의 *생각해보기 확인!)

[용어]

tss : 전체 데이터에 대한 산포도. 센터와의 거리 제곱의 합. (일종의 분산값으로 'k값, 클러스터링' 과 상관 없음) wss : 클러스터 내의 포인트에 대한 산포도. k값과 관련, 클러스터링해서 각 클러스터의 센트로이드와 데이터 포인터와의 거리의 제곱의 합. bss : tss-wss 클러스터들 끼리의 산포도 ch index =

$$\frac{BSS(k)/_{k-1}}{WSS(k)/_{n-k}}$$

(n = 데이터 포인트 갯수, k = 클러스터 종류 갯수)

(k가 커지면 wss가 작아지지만 n-k역시 작아져서 어느정도의 균형을 이룸. bss(tss-wss)는 k가 커지면 tss는 상수, wss는 작아지니 커지게 된다. 하지만 그에 따라 k-1도 커져서 이 또한 어느정도의 균형을 이룬다.)

*생각해보기

1. 클러스터 간의 거리가 멀다(크다) = 다른 성향의 것들로 잘 분류를 한것이다.
2. 클러스터 내의 거리가 가깝다(작다) = 분류한 것들의 성향이 매우 비슷하다.
3. ch index가 크다. = 1번과 2번에 해당한다. = 잘 분류했다.

Unstructed data analysis

데이터의 스펙트럼은 3가지가 있다.

1. Structured : DB를 기록할 때, 각각의 변수에 대한 스키마(데이터 형식)에 맞는 경우만 입력이 가능하다. ex) 나이 변수에 0~100이면 121살의 정보를 입력하면 오류가 난다. Quality 가 높은 비싼 데이터. 은행, 공공기관 등
2. semi-structured : 형식은 있지만 값이 비어있거나 이상한 값이 들어있는 경우 ex) html이나 sml 등 형식은 있지만 그에 맞게 데이터가 들어가 있지는 않은 경우
3. unstructured : 일반적인 글들, 영상, 음성 등

cf) n-gram : n개의 token

cf) stop_words : is, the, of와 같이 핵심 단어로 판단하기 어려운 단어들

tf-idf

[tf] : term frequency로, 단어의 빈도라 할 수 있다. 하지만 이것만으로 해당 문서의 핵심이 무엇인지 파악하기는 힘들다.

[idf] : inverse document frequency로, 다른 문서에도 등장하는 단어인지 확인하는 것이라 할 수 있다. ' ln(전체문서의수/단어가나오는문서의수) ' 로 계산한다.

Recommender System

: 상품, 사람 등을 추천해주는 시스템으로, conversion rate(내가 필요한 물품이 있는데 마침 추천해주면 구입할 확률 높아짐), cross-selling(자전거 구입한 사람한테 예쁜 보호대 추천하는 것)과 같은 장점이 있다.

컨셉

의 종류는 크게 두가지이다.

1. content base filtering

: 상품 기반. 아디다스 브랜드만 사는 고객에게 아이다스 신상품 추천

장점

=> 이미 평가가 진행된 상품을 추천하다보니 퀄러티는 보장된다.

단점

=> 매번 같은 것만 추천해서 새로운 걸 추천하는것이 좀 제한적이다. 모든 상품을 수치화 한다는 것이 쉽지 않다.

2. collaborative(a.공동의) filtering

: 물건을 사는 패턴이 옆집 사람과 비슷하다. 그러면 옆집 사람이 사는 상품을 추천해줌

이 필터링 방법은 알고리즘으로, 더 자세히 알아볼 필요가 있다.

Collaborative filtering(CF)에 대해서 조금더!

1. Memory-based

: 유사한 사용자나 상품을 기반으로 추천함. ex) user-based CF, item-based CF

user-based CF

: 비슷한 사용자 top을 뽑아서 그들이 높게 평가한 상품에 대해 추천.(k명의 비슷한 사용자들을 찾고, 그들이 특정 상품에 대해 내린 평가를 평균을 내서 높은 것들을 추천함.)

User - based cf의 문제점

- 성능은 아주 좋지만 예측하고자 하는 사람의 정보가 많이 없거나, 완전 신상품은 평점 정보가 없어서 그래서 문제를 요약하면.

1. Cold Start Problem 1-1. User record x (어떤 고객인지 모를경우) => popularity로 해결 1-2. Item record x (신상품이라 평가가 없을 경우) => 노출시키기, 유사상품으로 가상평가, 평가단을 활용.
2. Computationally Expensive = not scalable 하다.

   (데이터 사이즈가 커져도 작업속도가 커지지 않아서 데이터가 커져도 가능한 것. 이 안되는거) 상품, 사용자가 많아질수록 계산양(비용, 시간)이 커진다.

item-based CF

: 방금은 사용자의 유사도로 비교하고 추천했는데 이번에는 아이템의 유사도를 가지고 추천하는 것. 가령, 사용자 A에게 추천한다고 했을 때, A의 구입 이력이 1, 5, 8인 경우, i3 상품에 대하여

i1 i5 i8 0 0.7 0.9 (i3 상품과 이전에 구입했던 상품과의 유사도) 2 4 5 (사용자가 내린 평가)

실제로 추천 점수를 계산해보면 다음과 같다.

(0x2+0.7x4+0.9x5) / (0+0.7+0.9) =4.56

만약 4.56이라는 수치가 다른 상품들에 비해 높은 점수라면 i3을 추천.

추천 정확도는 user-based보다는 떨어지나 계산이 덜 복잡해서 실제로는 이걸 쓴다.

(k=3 이렇게 설정을 하는데 각 상품마다 그 상품과 비슷한 top3만 활용을 해서 계산을 한다. 계산량을 줄일 수 있다. 또한 계산할 때, 관련이 없는건 계산에 포함 x)

S	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8
i_1	-	0.1	0	**0.3**	**0.2**	**0.4**	0	0.1
i_2	0.1	-	**0.8**	**0.9**	0	**0.2**	0.1	0
i_3	0	**0.8**	-	0	**0.4**	0.1	0.3	**0.5**
i_4	**0.3**	**0.9**	0	-	0	**0.3**	0	0.1
i_5	**0.2**	0	**0.7**	0	-	**0.2**	0.1	0
i_6	**0.4**	**0.2**	0.1	**0.3**	0.1	-	0	0.1
i_7	0	**0.1**	**0.3**	0	0	0	-	0
i_8	**0.1**	0	**0.9**	**0.1**	0	0.1	0	-
	-	0	4.56	2.75	-	2.67	0	-

2. Model-based

: 등급이 매겨지지 않은 항목에 대한 사용자의 등급을 예측하여 추천해주는 것.

(코드로 이해하기!)

지금까지 비슷한 성향의 것들을 기반으로 추천한다고 하는데, 비슷한 성향을 어떻게 측정할 것인지에 대해서 다루지를 않았다. 유사도는 다음의 세가지 방법으로 측정해볼 수 있다.

유사도 측정

1. Pearson correlation coefficient

:

$$\mathrm{sim}_{\mathrm{Pearson}}(\mathbf{x},\mathbf{y}) = \frac{\sum_{i\in I} x_i y_i - I\bar{\mathbf{x}}\bar{\mathbf{y}}}{(I-1)s_x s_y}$$

:

$$u_{ik} = \frac{\sum_j (v_{ij}-v_i)(v_{kj}-v_k)}{\sqrt{\sum_j (v_{ij}-v_i)^2 \sum_j (v_{kj}-v_k)^2}}$$

vij = i번째 사용자가 j상품을 얼마나 좋아했는지 vi = i번째 사용자가 전체 상품평가의 평균 j= i와 k둘다 사용한 상품

i1 i2 i3 i4

vi ? 3 5 1 vk 1 2 ? 4

vi = 9/3 = 3 vk = 7/3 = 2.3

j = {i2, i4} i2와 i4의 관계가 어떻게 되는가

2. Cosine similarity

:

$$\mathrm{sim}_{\mathrm{Cosine}}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|_2\|\mathbf{y}\|_2}$$

:

$$\cos(u_i,u_j) = \frac{\sum_{k=1}^{m} v_{ik}v_{jk}}{\sqrt{\sum_{k=1}^{m} v_{ik}^2 \sum_{k=1}^{m} v_{jk}^2}}$$

상관관계가 궁금한 상품에 대해서 각도를 가지고 similarity 계산 vi <3, 1> vk<2, 4> 텍스트 마이닝에서 tf-idf matrix가지고 많이 사용한다. 값이 크면 비슷, 값이 작으면 다른

3. Jaccard index

:

$$\mathrm{sim}_{\mathrm{Jaccard}}(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

, binary data에만 된다.!

궁금 해결선]

1. deep layer가 가능해진 이유

: ANN 기법의 문제(overfitting, gradient->0, 학습 시간 느림)이 ReLu 함수로 해결? or back prop로 해결?

=> 학습시간 느린게 아님. 기울기가 0으로 가서 업데이트가 안되는 것임. ReLu로 해결한것.

=> back prop은 gradient를 0으로 가게 한다.

2. neural net 등장 배경이 non-linear hypothesis에서 feature수에 의한 부작용?

: feature수를 선택적으로 잡으면 overfitting문제가 발생하지 않으면서 non-linear가능하지 않은지

=> 가능하지만 feature를 잘 선택하는 것이 쉽지 않음. 하지만 뉴럴넷은 feature도 학습하니 가장 효과적

3. forward 와 backward의 개념차이

: input layer에서 시작해 구한 output으로 error값을 구하는 것.

: output layer에서시작해 error값을 구하는 것.

둘다 세타optimize를 위한 방법인데 과정의 차이인지

=> forward는 input layer로 부터 예측하는 개념으로 output을 구해가는것.

=> backward는 output layer로 부터 학습하는 개념으로 값을 구해가는 것.

4. softmax와 cross entropy 개념이해

sofmax : multi classification을 하는 activation함수로, sigmoid가 positive일 확률 하나만을 가진다면 softmax는 여러 확률 값을 가진다.(sum이 1)

cross entropy : logistic cost function함수, multi,

5. RBM은 언제 사용되는(유용한) 것인지

: 신경망은 초기값이 중요한데(local minimum에 빠질까봐) 세타 초기값을 학습시켜서 찾는것.

6. entropy와 gini index 어떨때 한 쪽이 효과적인지

: 차이 거의 없음.

7. boosting이 왜 컴퓨터 비전의 일반적인 작업?

: 디시즌 바운더리 근처 것에 가중치를 주어 학습시키기 때문에 구분하는데 효과적임.

8. Proximity measure 개념이해 부족

: 트리마다 맨 끝의 external node는 예제가 비슷하다. 이를 이용하는 것.

9. k-means와 hierarchical 쓰려면 scaling이 필요하다?

: k-means와 hierarchical 모두 센트로이드와 데이터 포인터 간의 거리를 계산해서 가까운 데이터 포인터를 묶는 것인데, 거리를 계산할 때 스케일이 작은 건 묻힌다. 가령,

x축 = age, y축 = income이라고 했을 때, 실제로 포인트 잡아서 거리 계산하면, (30,10000), (40, 15000) age는 영향 없음.

10. k-means는 노이즈에 취약하고 DBSCAN은 노이즈의 영향을 덜 받는다?

: 노이즈란 멀리 떨어진 값으로 센트로이드가 평균으로 구하는데, 노이즈 때문에 왜곡된다.

11. wss를 한마디로 정의하면 산포도, ch index는? 둘은 클러스터링이 잘 되었는지를 평가하는 것?

: ch index는 클러스터 내의 거리 / 클러스터 간의 거리 로, 좋다 = 크다. / 그렇다.

12. 클러스터링이 무척 잘되면 또 overfitting인지?

: 비지도 학습은 overfitting이라는 개념이 없다. 정답이 없다.

13. deep layer에서 back propagation을 사용해서 gradient를 구할 때, 왜 gradient값은 0으로 가며, 이를 해결할 수 있는 방법은 무엇일까?

: 현재 각 노드들이 모두 logistic regression으로 activation함수를 sigmoid함수로 사용하고 있다. 그래서 output으로 나오는 값이 1보다 작은 값이며, back propagation으로 세타를 구하는 방식이 chain rule을 적용해 곱해지기 때문에 결국 1보다 작은 값이 계속해서 곱해져서 0으로 수렴하게 된다. 한마디로, 잘못된 타입의 non-linearity를 사용한 것이다. 이를 해결하기 위해서는 ReLU함수를 사용하게 되는데, 0보다 작은 지점에서는 값을 내지 않고, 0보다 큰 지점은 y=x 그래프로 output의 값을 조절한다.

14. 세타 초기화를 할 때, 모두 0으로 하면 발생하는 문제는?

: 한 번 weight를 업데이트할 때마다 같은 입력단에서 나온 weight들에는 같은 값이 들어간다. 업데이트 값이 같다.

15. drop out

: 노드들을 선택적으로 뽑아서 학습을 시키고 그걸 나중에 뭉쳐서 결과냄.

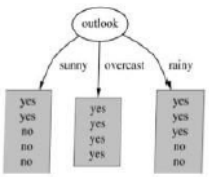
16. IDE

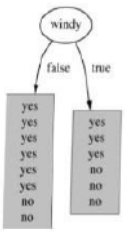
: entropy = - yes확률*log(yes확률)* - *no확률*log(no확률)

information gain = 상위 entropy - 하위 entropy(비중을 취해준다.)

Classification with using the ID3 algorithm

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no





E (Outlook=sunny) = $-\frac{2}{5}\log\left(\frac{2}{5}\right)-\frac{3}{5}\log\left(\frac{3}{5}\right)=0.971$

E (Outlook=overcast) = $-1\log(1)-0\log(0)=0$

E (Outlook=rainy) = $-\frac{3}{5}\log\left(\frac{3}{5}\right)-\frac{2}{5}\log\left(\frac{2}{5}\right)=0.971$

} H(S,Outlook)

Average Entropy information for Outlook

I (Outlook) = $\frac{5}{14}*0.971+\frac{4}{14}*0+\frac{5}{14}*0.971=0.693$

} $\sum_{t\in T} p(t)H(t)$

Gain (Outlook) = E(S) − I (outlook) = 0.94-.693 = 0.247 ⇨ $IG(A,S)=H(S)-\sum_{t\in T}p(t)H(t)$

E (Windy=false) = $-\frac{6}{8}\log\left(\frac{6}{8}\right)-\frac{2}{8}\log\left(\frac{2}{8}\right)=0.811$

E (Windy=true) = $-\frac{3}{6}\log\left(\frac{3}{6}\right)-\frac{3}{6}\log\left(\frac{3}{6}\right)=1$

Average entropy information for Windy

I (Windy) = $\frac{8}{14}*0.811+\frac{6}{14}*1=0.892$

Gain (Windy) = E(S) − *I* (Windy) = 0.94-0.892=0.048

9단원 28 페이지

Bagging v.s. Boosting

비교	Bagging	Boosting
특징	병렬 앙상블 모델 (각 모델은 서로 독립적)	연속 앙상블 (이전 모델의 오류를 고려)
목적	Variance 감소	Bias 감소
적합한 상황	복잡한 모델 (High variance, Low bias)	Low variance, High bias 모델
대표 알고리즘	Random Forest	Gradient Boosting, AdaBoost
Sampling	Random Sampling	Random Sampling with weight on error

배깅의 문제점 모든 feature를 가지고 가니까 특정 feature가 영향력이 너무 크면 모델링 하는데 트리 모델들이 비슷한 결과값을 낸다. 그래서 나온게 random forest

17. 문제점 두개

:

18. 랜덤 포레스트

: 그릴 트리 갯수를 정한다. 부트스트랩 샘플을 한다. 뽑은 부트스트랩 중에서도 랜덤하게 feature을 만들어보고, 그걸 합쳐서 판단을 하게 한다. 샘플링에 제외되었던 데이터를 기반으로 테스트 한다.

19. 변수의 중요성

: 한가지 변수를 끄고 모델을 만들어본다. = 열로 섞는다. 그러면 해당 feature에 대한 영향력을 조절할 수 있다.

20. 왜 unsupervised를 배워야할까?

: 애초에 label데이터가 잘 없다. 진료나 평가한 결과가 있는 데이터로서 흔하지 않음. 모든게 unlabeled 데이터이다. 숨어있는데 구조를 파악하기 위해서, 그냥 unsupervised 자체가 목적, 다음 단계를 위한(클러스터링 말고도 데이터를 학습시켜서 샘플을 뽑아내는데 그런 작업을 위한) 전처리 단계

21. kmeans

:

1. Initialize the center of the clusters	$\mu_i = \text{some value}, i = 1, \ldots, k$
2. Attribute the closest cluster to each data point	$\mathbf{c}_i = \{j : d(\mathbf{x}_j, \mu_i) \leq d(\mathbf{x}_j, \mu_l), l \neq i, j = 1, \ldots, n\}$
3. Set the position of each cluster to the mean of all data points belonging to that cluster	$\mu_i = \frac{1}{\lvert c_i \rvert} \sum_{j \in c_i} \mathbf{x}_j, \forall i$
4. Repeat steps 2-3 until convergence	
Notation	$\lvert \mathbf{c} \rvert$ = number of elements in $\mathbf{c}$

22. wss와 ch index(k값이 커진다고 무조건 ch index값이 커지는 거 아님.)

k값이 커질수록 wss떨어지는 폭은 둔화되는데 k값은 계속해서 커지는거야 그러니까 ch값은 작아지는 것이다. 그런데 ch가 큰게 좋은것이다! 그래서 decision이 필요해진다. 적정값을 찾아한다.

23. data 종류

: semi = format은 있지만 각 데이터가 그 format을 따르고 있지는 않다.

24. tf-idf

:

$$tf(term) = \frac{n_{term\ occurrence\ in\ a\ document}}{n_{all\ words\ occurrence\ in\ a\ document}}$$

$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

25. contents -based의 한계

: quality가 보장되지 않음. 그냥 비슷한 상품을 추천하는 것일뿐 질과는 관계가 없음.

: 새로운 물건을 추천할 가능성이 없다. no surprise

: 의미있는 특징을 뽑아내야하는 작업이 필요하다.

26. user based CF

1. Cold Start Problem 1-1. User record x => popularity 1-2. Item record x => 노출시키기, 유사상품으로 가상평가, 평가단을 활용.
2. Computationally Expensive = not scalable(데이터 사이즈가 커져도 작업속도가 커지지 않아서 데이터가 커져도 가능한 것. 이 안되는거)

(knn이 가지고 있는 단점을 가진다.) 2-1. 상품, 사용자가 많아질수록 계산양(비용, 시간)이 커진다.