

Machine Learning Study (Boosting 기법 이해)

Xgboot를 이해하기 위해 필요한 개념들을 정리

Decision Tree, Ensemble(bagging vs boosting) (Adaboost, gbm, xgboost, lightgbm) 등

2017.11

freepsw

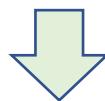
요즘 Kaggle에서 유명한 Xgboost가 뭘까?



Ensemble중 하나인 Boosting기법?



Ensemble 유형인 Bagging과 Boosting 차이는?



왜 Bagging이 low bias, high variance 모델인가?



Boosting 기법은 어떤게 있나?



Bias 와 Variance 관계는?

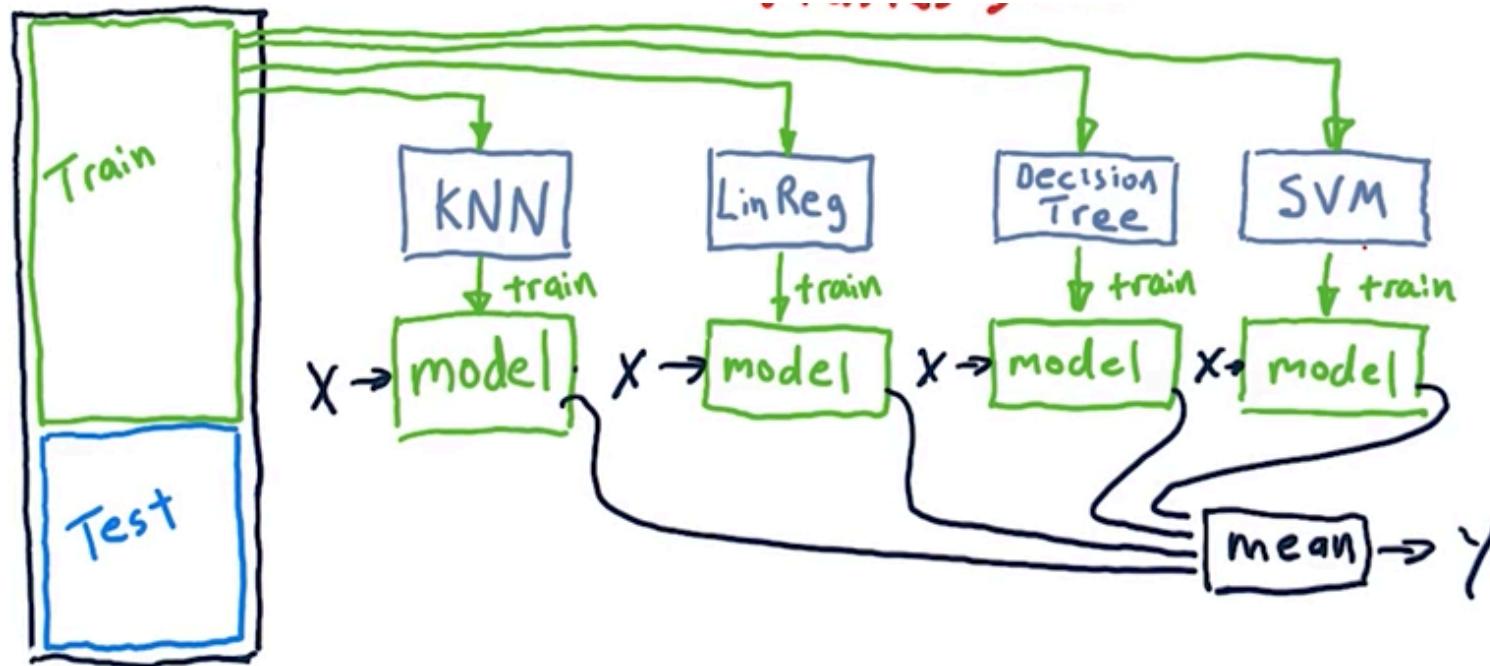


Xgboost에서 사용하는 CART 알고리즘은?

1. Ensenbles (앙상블) 이란?

여러 모델을 이용하여 데이터를 학습하고, 모든 모델의 예측결과를 평균하여 예측

앙상블 기법의 개념



Why Ensenbles?

- Error 최소화
 - 다양한 모델의 결과를 종합하여 전반적으로 오류를 줄여줌
- Overfitting 감소
 - 각 모델별로 bias가 존재함.
 - 이렇게 다양한 bias를 종합하여 결과를 생성하게 되어, overfitting을 줄여줌
- Low Bias, High Variance
 - Variance를 줄이기 위한 기법

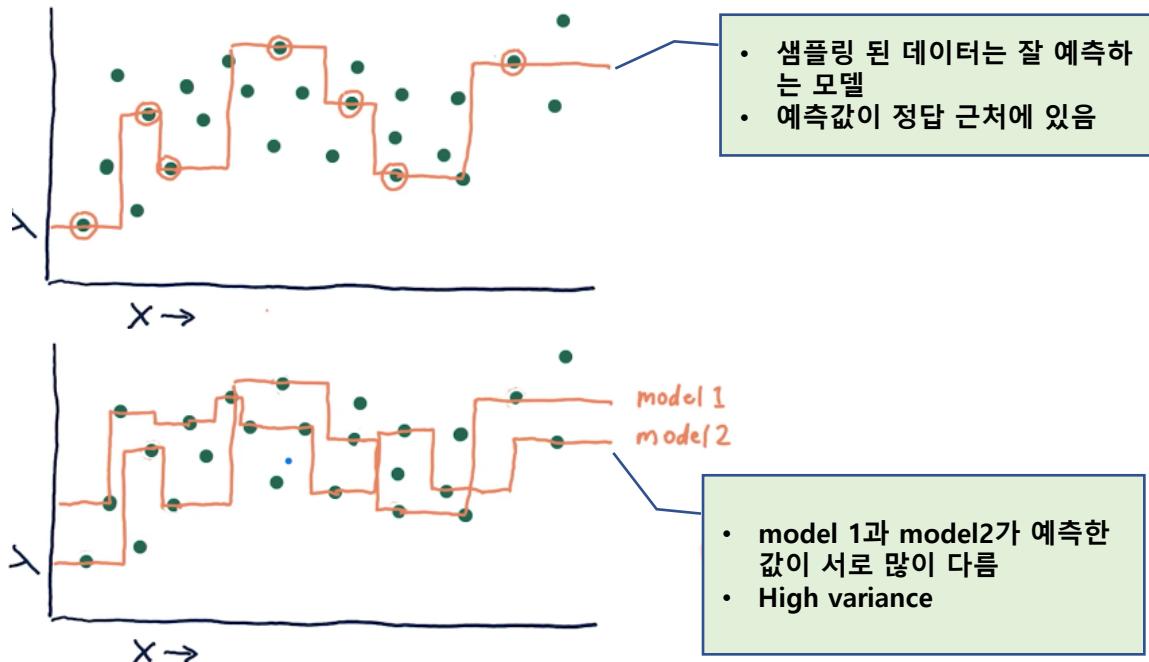
왜 Low Bias, High Variance인가?

1. Ensembles은 왜 Low Bias, High Variance인가?

Bagging에서 Regression모델을 예로 들면...

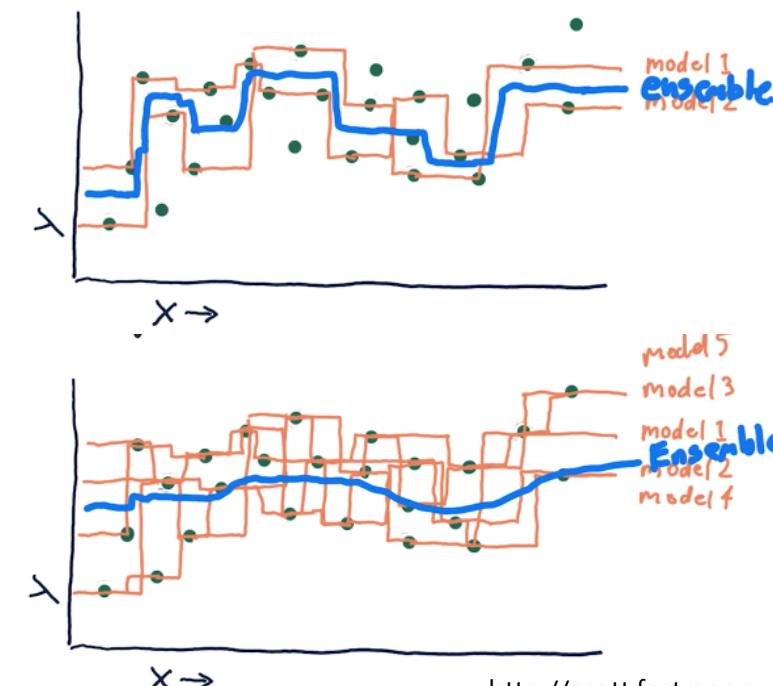
샘플링을 통해 모델을 학습하면...

- 하나의 모델만 본다면...
 - Low Bias : model1과 2가 정답과 가까운 거리에 위치
- 여러 개의 모델을 함께 보면...
 - High variance : 각 모델별로 예측한 값의 차이가 크다



Variance를 줄이려면?

- 모든 모델이 예측한 값의 평균을 사용하자
- 아래의 예시를 보면,
- 모델이 많을 수록 평균 값을 가진 모델이 실제 데이터와 유사

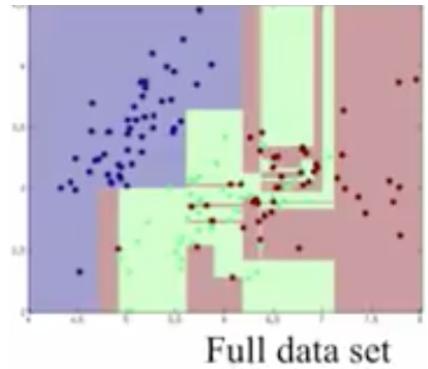


1. Ensembles은 왜 Low Bias, High Variance인가?

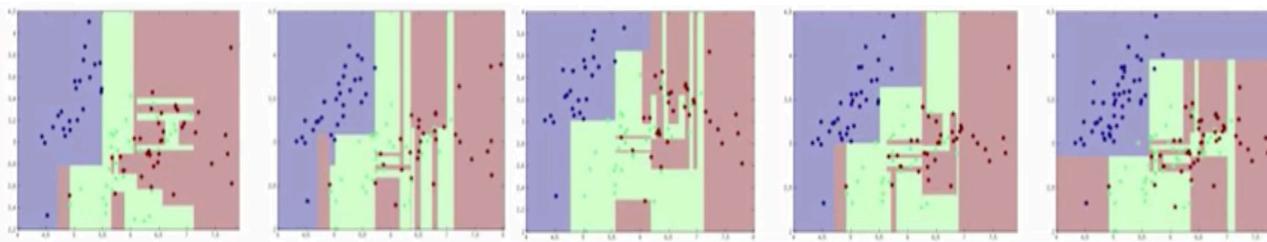
Bagging에서 Decision Tree 모델을 사용한 예시

샘플링을 통해 모델을 학습하면...

- 실제 데이터를 분류한 Decision Tree

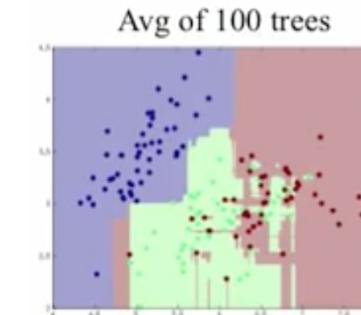


- 샘플링된 데이터를 이용한 Decision Tree
 - 일부 데이터만 이용하여,
 - 전체를 제대로 분류하지 못함.

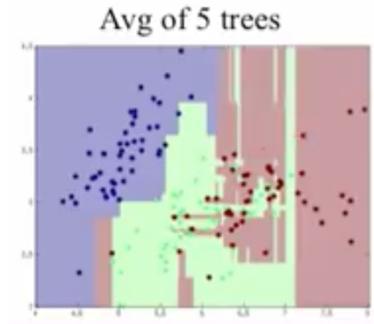


Variance를 줄이려면?

- 모든 모델이 예측한 값의 평균을 사용하자
- 아래의 예시를 보면,
- 모델이 많을 수록 평균 값을 가진 모델이 실제 데이터와 유사



- 100 개의 모델의 예측값을 이용한 것이
- 실제 데이터 분류 모델과 더 유사함.



[참고] bias vs variance (1/4)

학습한 모델의 예측 오류는 크게 2개(Bias, Variance)오류로 이루어짐

Bias

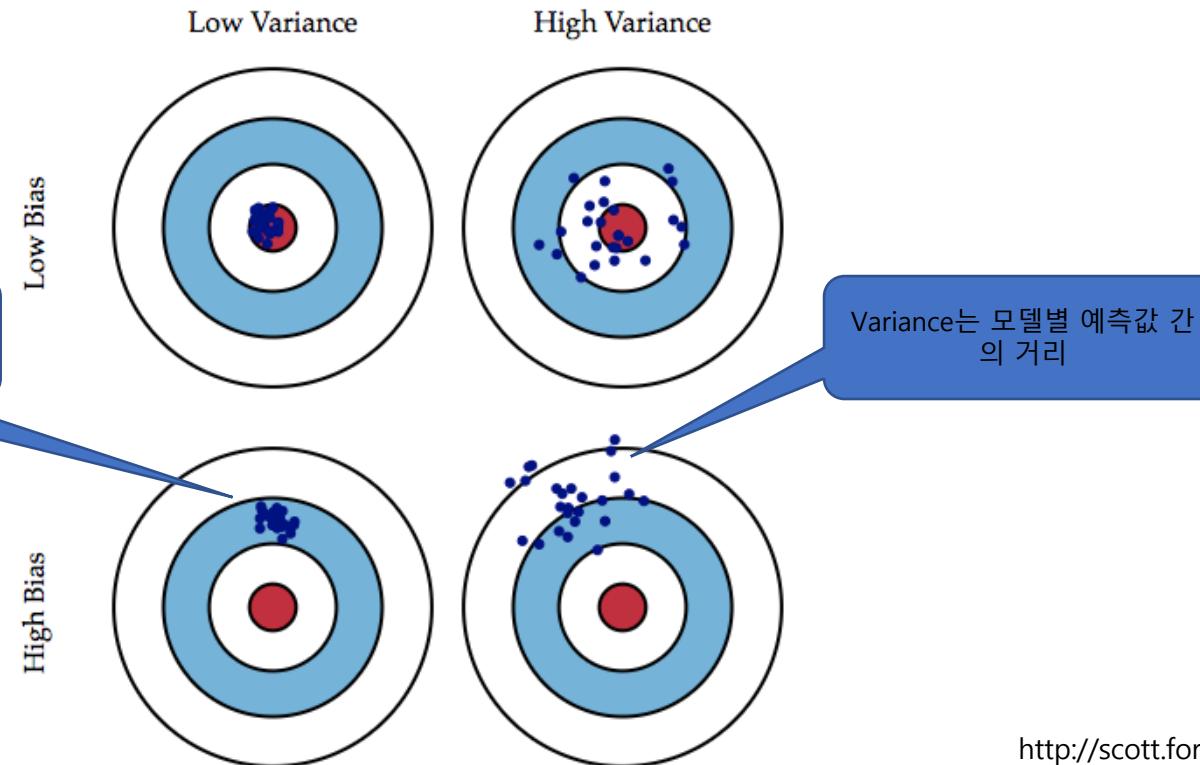
[Error due to Bias]

- Bias로 인한 에러는 예측값과 실제값 간의 차이
- (당연한 소리.. 그럼 뭐가 다른가?)
- 모델 학습시 여러 데이터를 사용하고, 반복하여 새로운 모델로 학습하면, 예측값들의 범위를 확인 할 수 있다.
- Bias는 이 예측값들의 범위가 정답과 얼마나 멀리 있는지 측정

Variance

[Error due to Variance]

- 주어진 데이터로 학습한 모델이 예측한 값의 변동성 (분산, variance)
- 만약 여러 모델로 학습을 반복한다고 가정하면,
- Variance는 학습 된 모델별로 예측한 값들의 차이를 측정



[참고] bias vs variance (2/4)

수학적 정의 및 예시

수학적 정의

- Y : 예측 값
- X : 변수
- ϵ : 에러, 에러의 분포는 정규분포 $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$.
- $Y = f(X) + \epsilon$: 예측값과 변수간의 관계 ($f(X)$ 는 정답을 추출하는 함수)
- $f'(X)$: 모델 학습을 통해 정답을 예측하는 함수
- 예측 모델의 에러 : $Err(x) = E[(Y - \hat{f}(x))^2]$

예측값과 예측값 평균간의 거리

- 위 공식을 세분화하면 bias와 variance로 구성된 공식으로 변환

$$Err(x) = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma_\epsilon^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

정답과 예측값 간의 거리 측정

어떤 모델로도 제거 불가능한 오류.
현실 세계에서는 완벽한 모델이 없으므로, 오류가 있다는 가정

예시 (투표 예측)

- 목적 : 유권자들이 어떤 정당을 투표할지 예측
- 데이터 : 전화번호부에서 랜덤으로 50명 추출
- 측정 방법
 - 어떤 정당에 투표할지 질문 및 결과 취합
 - 여당(13), 야당(16), 무응답(21)

	Voting Republican	Voting Democratic	Non-Respondent	Total
	13	16	21	50

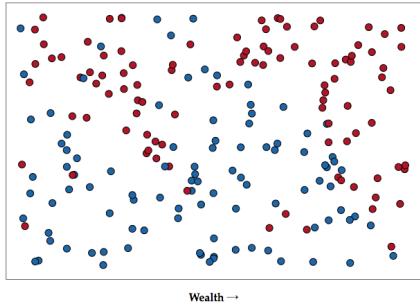
- [결과]
 - 야당이 승리 예상 $\rightarrow 16/(13+16)$
 - 실제는 여당이 압승
- 예측이 틀린 이유가 뭘까?
 - Bias (전화번호부)
 - 전화번호부에 있는 사용자만 대상 (특정 계층에 치우침)
 - 여러가지 모델과 데이터를 샘플링해서 반복 학습해도
 - 특정 계층(전화번호부에 있는)만 대변하는 결과가 추출됨.
 - 단, 여러번 반복하면, variance는 낮아짐.(예측값이 서로 비슷)
 - Variance (작은 샘플 데이터)
 - 작은 데이터로 학습한 모델의 예측값의 차이가 큼
 - 샘플을 더 많이 확보하면, variance가 줄어 듬

[참고] bias vs variance (3/4)

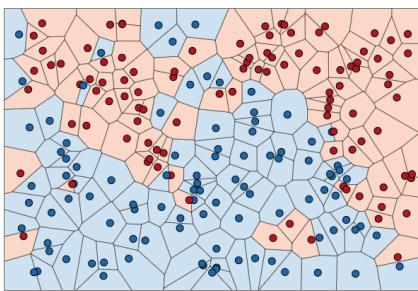
Bias와 Variance간의 Trade-off 이해

Voter Party Registration 예측 예시

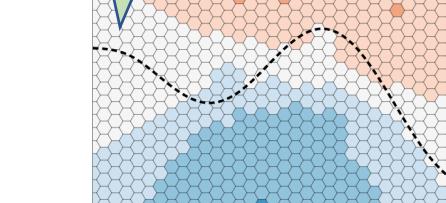
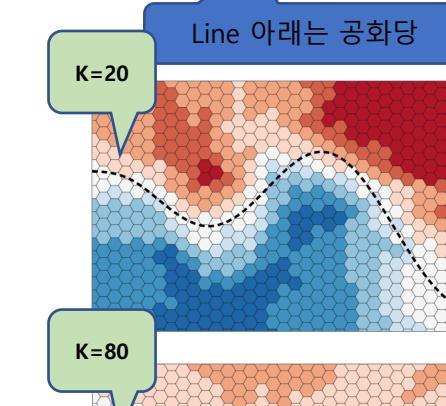
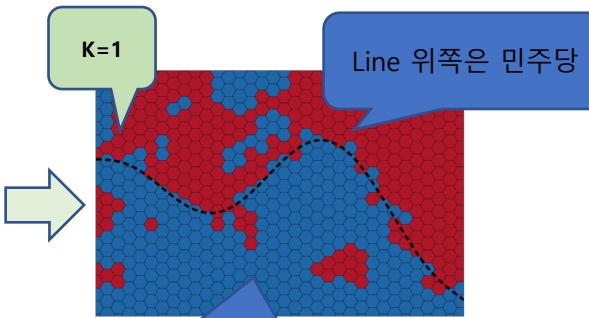
- 3개의 속성(소속정당, 자산, 종교)으로 구분된 투표자 정보



모델 학습

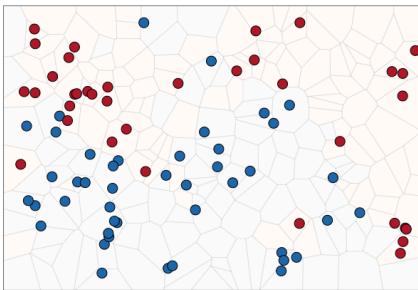


- K의 갯수에 따라 예측값이 달라짐



- 빨강 : 민주당
- 파랑 : 공화당

학습된 모델로
신규 유권자 정당 예측



Bias vs Variance

- K 증가

- 각 예측에 더 많은 유권자 정보의 평균값 사용
- 따라서, 예측곡선이 부드러워짐.

- K 감소

- 1인 경우, 가장 가까운 유권자 정보만 고려하여, 정당별로 고립되거나 급격한 예측곡선 생성됨

[Variance 관점]

- Small K → variance 증가 (모델간의 예측값 차이 큼)
 - 왜 그럴까?

- 가장 가까운 값을 이용하여 예측하므로,- 신규 데이터가 추가되면, 학습한 모델의 예측값이 급격하게 변함 (추가된 데이터 주변)
- 따라서 모델간의 예측값 차이가 커짐

- Large K → variance 감소

- 여러 개(K)의 학습데이터를 이용하여,- 신규 데이터 추가되어도 예측값 간에 큰 변경없음

[Bias 관점]

- Small K → Bias 감소 (정답과 예측값이 유사)
 - 왜 그럴까?

- 가장 가까운 값을 이용하여,- 예측값이 정답과 근처에 있을 가능성 높음

- Large K → Bias 증가 (예측값이 정답과 많이 다름)

- 모델선형 주변으로 예측하지 못하는 데이터가 많아짐. → 예측값과 정답간의 거리가 멀어짐

[참고] bias vs variance (4/4)

Bias와 Variance 활용하여 모델 복잡도 최적화 하기

모델 최적화시 고려사항

[Bias 만 줄이면 예측 정확도가 올라갈까?]

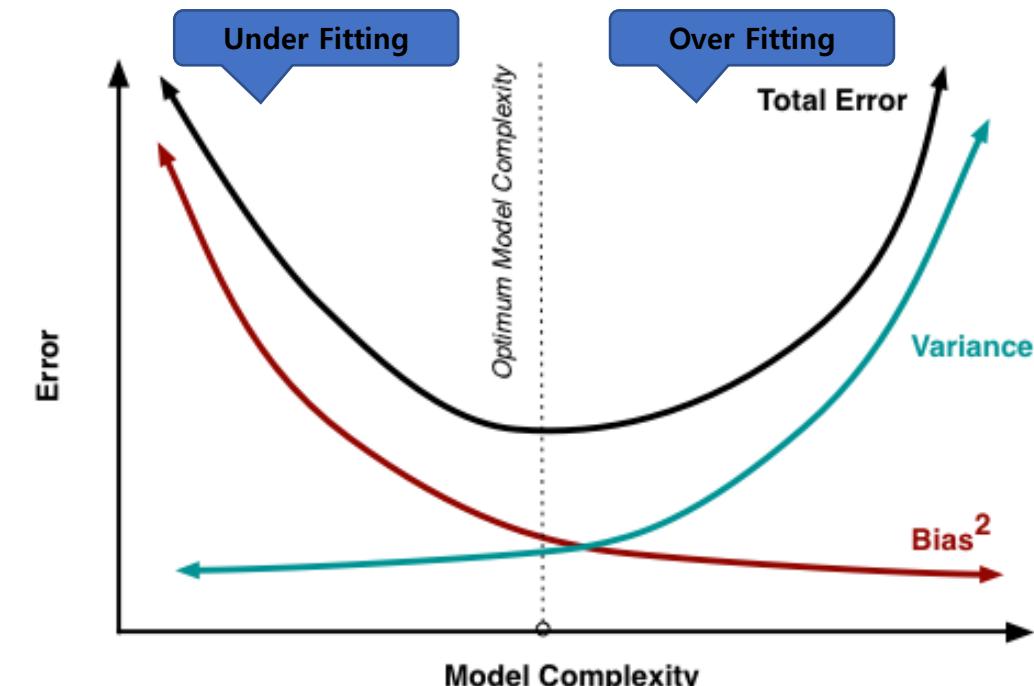
- 많은 사람들이 Bias가 줄어야 정답을 예측할 확률이 높다고 생각한다.
- 만약 긴 시간동안 모델이 예측한 값의 평균을 생각하면 맞는 말일 것이다.
- 하지만, 실제 모델은 하나의 모델로 정확도를 평가하므로 Bias만 줄이는 것이 정확도를 높이지 못한다.
- 따라서, Bias와 Variance를 동시에 고려하여 모델 복잡도를 관리해야 함

[Bagging / Resampling의 효과]

- 위 방식은 모델 복잡도에서 variance를 줄여준다. 어떻게?
- 원본 data set에서 샘플링을 통해 여러개의 data set을 생성하고,
- 각 data set으로 모델을 학습한다.
- 이 과정에서 각 모델의 bias는 낮아지고(정답과 유사),
- variance는 높아진다. (각 모델 별로 정답의 차이가 큼)
- 그럼 어떻게 variance를 낮출까?
- Bagging에서는 각 모델의 예측값을 평균하여 예측값을 만든다.
- → 이 평균하는 과정을 통해서 각 모델간의 차이를 줄여서 variance 감소

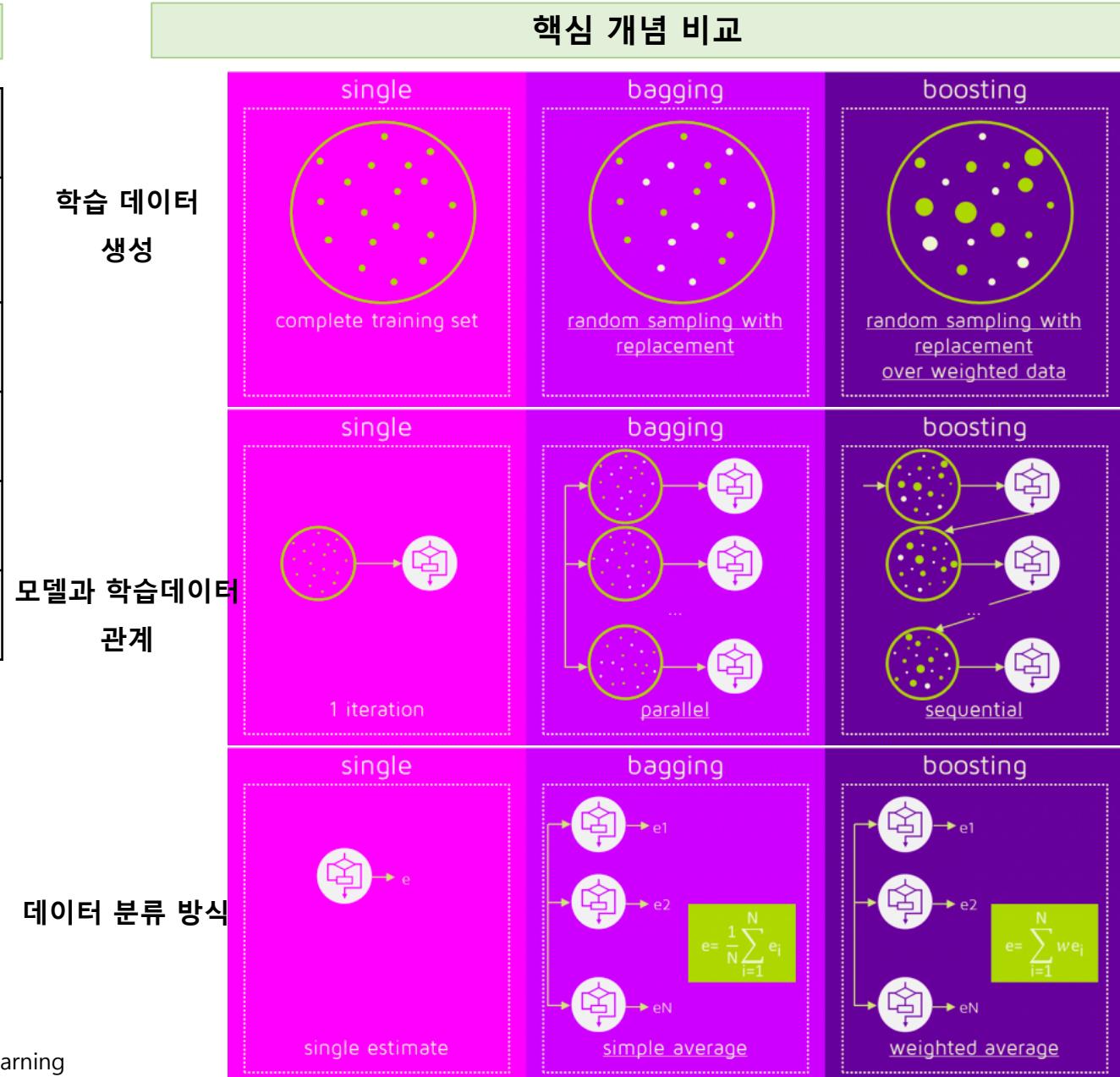
Bias & Variance Trade off

- 모델의 복잡도는 bias가 감소하고, variance가 증가하면서 발생한다.
- 따라서 모델의 복잡도가 최소화 되는 수준에서 bias와 variance를 선택



1. Bagging vs Boosting는 무엇이 다른가?

특징 비교		
비교	Bagging	Boosting
특징	병렬 양상을 모델 (각 모델은 서로 독립적)	연속 양상을 (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



<https://people.cs.pitt.edu/~milos/courses/cs2750-Spring04/lectures/class23.pdf>

<https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/>

<https://stats.stackexchange.com/questions/18891/bagging-boosting-and-stacking-in-machine-learning>

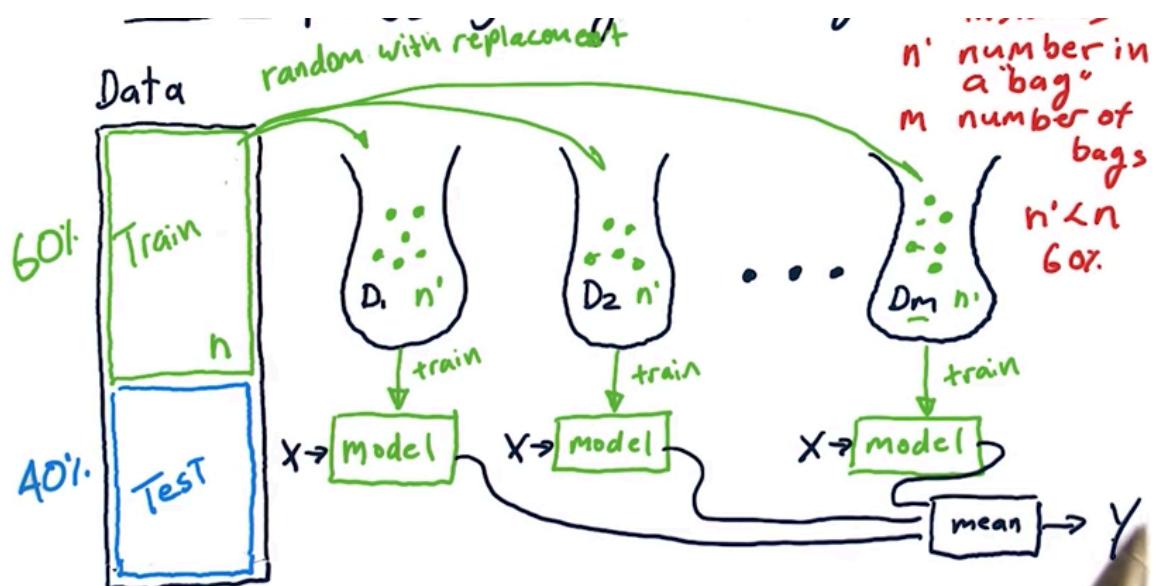
2. Bagging – Bootstrap aggregating

동일한 모델을 사용하고, 데이터만 분할하여 여러개 모델을 학습 (앙상블기법)

Bagging의 개념

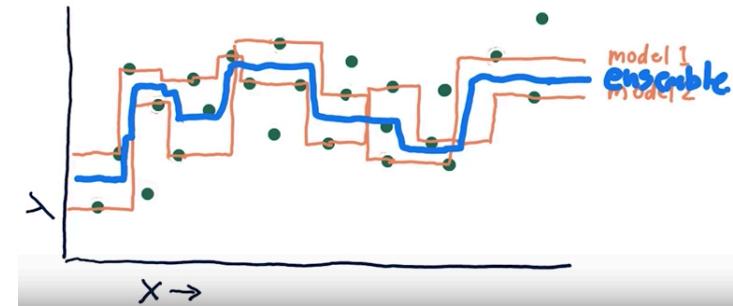
학습데이터를 랜덤으로 샘플링하여 여러개의 bag으로 분할하고, 각 bag별로 모델을 학습한 후, 각 결과를 합하여 최종 결과를 추출

- n : 전체 학습 데이터 수
- n' : bag에 포함된 데이터 수, 전체 데이터 중 샘플링된 데이터
- m : bag의 갯수, 학습할 모델별로 샘플링된 데이터 셋



어떻게 예측정확도를 높이나?

- 아래 그림에서 model1은 하나의 bag으로 학습된 모델이다.
- 각 모델별로 보면, 학습 데이터에 overfitting되어 테스트 데이터로 검증하면 예측성능이 낮다. (high variance)
- Bagging은 이렇게 weak model을 여러개 결합하여, 전체적으로 high variance \rightarrow low variance로 변하면서 예측성능을 향상한다.
- 아래 그림을 보면 여러개 모델이 서로 보완하면서 예측한다.
- Bagging은 linear 모델에는 잘 사용하지 않는데, 굳이 데이터를 샘플링하여 여러개 모델을 만들 필요가 없다.
- 이미 더 좋은 linear모델이 있음.



2. Random forest가 Tree correlation을 어떻게 해결하는가?

특정 feature가 정답에 많은 영향을 줄 때, 모든 tree들이 비슷한 결과를 도출하는
Tree correlation 문제 해결

Bagging의 이슈

- Bagged tree에서는 데이터를 샘플링하여 최적의 모델을 만든다. 그리고 이 과정을 n번 반복한다.
- 이렇게 만들어진 모델을 테스트 데이터로 검증하면, 당연히 서로 아주 다른 결과가 나온다. (학습된 데이터가 서로 다르니까)
- 샘플링 된 데이터로 학습하여 high variance가 발생하는 것을 방지하기 위해서, bagging에서는 여러개의 학습모델의 결과를 합하여 최종 결과를 도출한다.
- Bagging 과정에서 한 가지 이슈는 tree들이 얼마나 비슷하게 생성되는지 고려하지 못하는 것이다. (**tree correlation**)
- 예를 들어, 데이터 중에서 특정 변수(feature)가 정답에 미치는 영향이 아주 커서 cost(RSS)를 많이 줄여준다고 가정하자.
- 이 feature는 모든 tree가 공유하게 되므로, 대부분의 tree에서 동일한 결과를 예측하게 되는 현상이 발생한다. → tree correlation이 높아짐.

Tree correlation 해결 방안

[Random forest]

- 데이터 샘플링 시에 일부 feature들만 랜덤으로 선택한다.
- 따라서 모든 모델들은 서로 다른 feature로 학습하게 되고, 이로 인해 tree correlation이 줄어든다.

<https://stats.stackexchange.com/questions/295868/why-is-tree-correlation-a-problem-when-working-with-bagging>

<https://stats.stackexchange.com/questions/18891/bagging-boosting-and-stacking-in-machine-learning> - 모델간 비교 (자료 좋음)

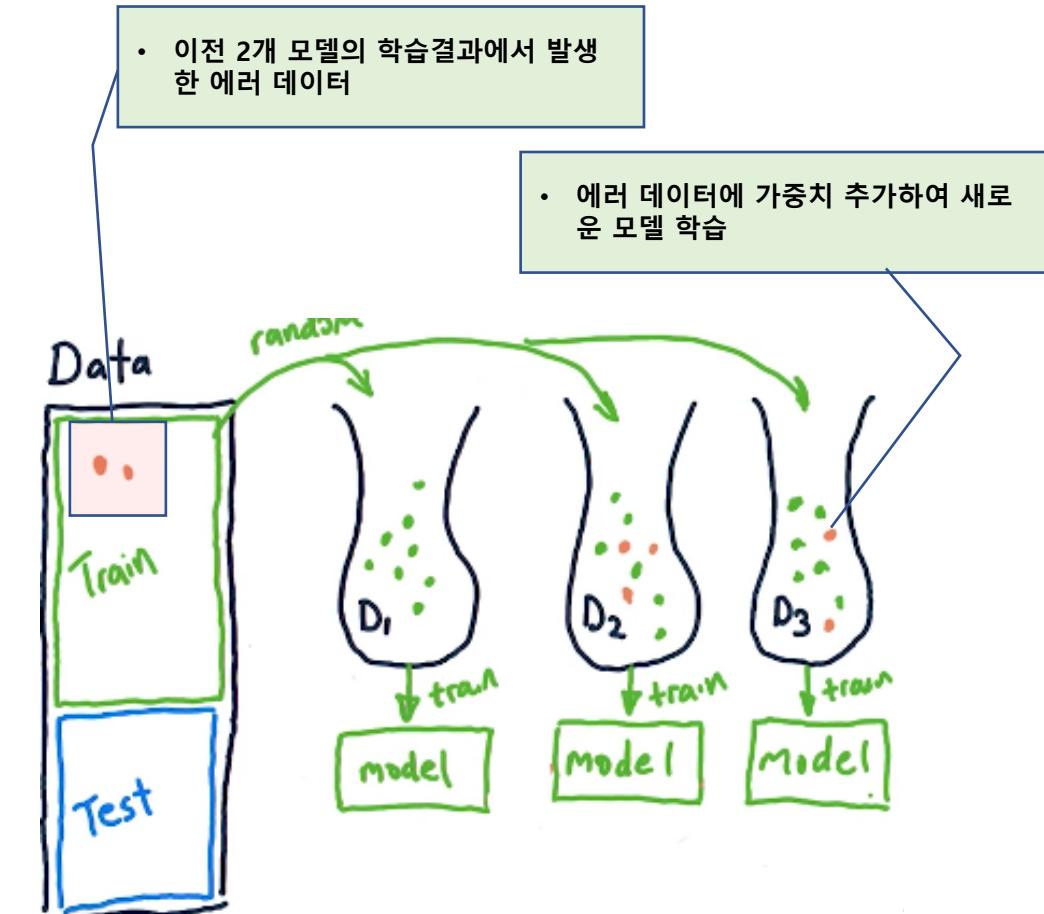
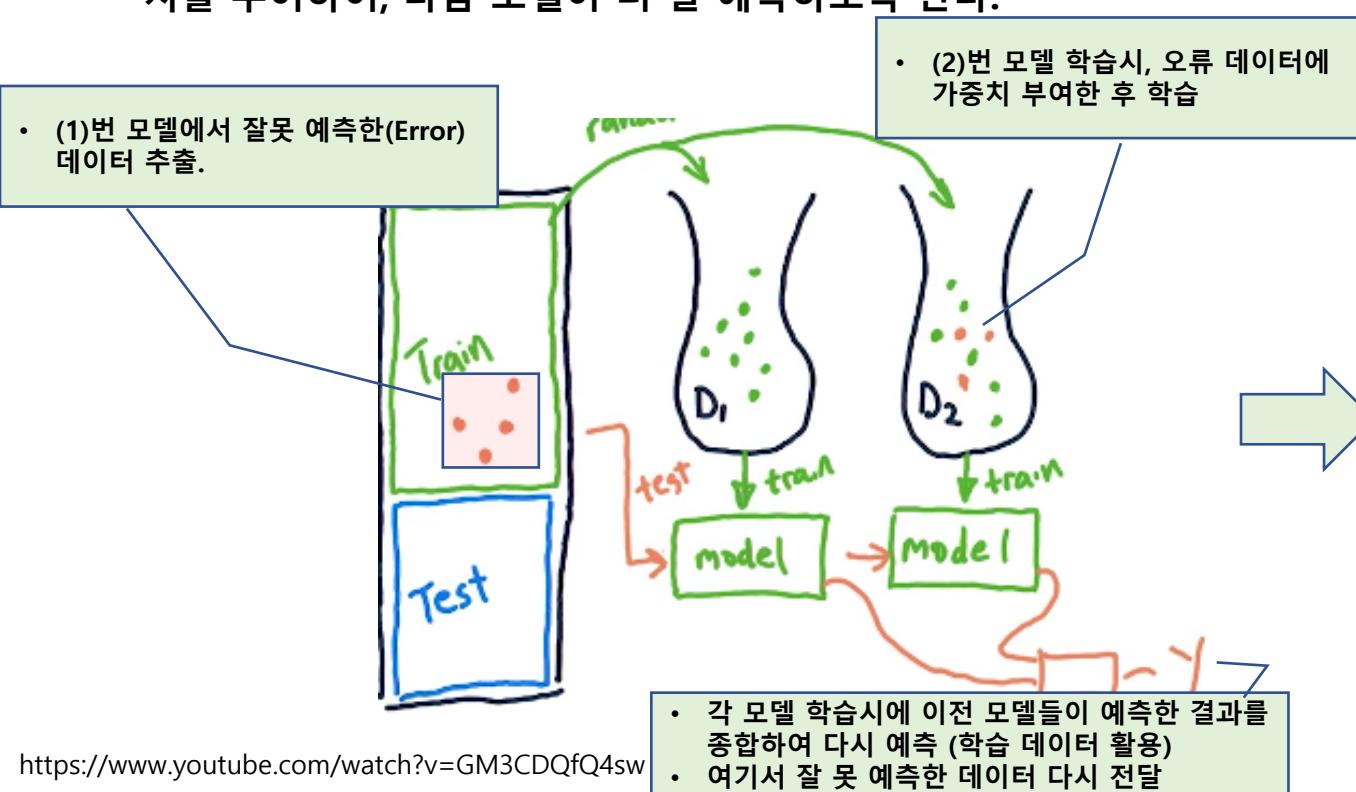
<http://operatingsystems.tistory.com/entry/Data-Mining-Ensemble-Bagging-and-Boosting>

3. Boosting

Bagging의 변형으로, 모델이 잘 예측하지 못하는 부분을 개선하기 위한 모델

Ada Boost (Adaptive, '애다'로 발음)

- Bagging에서 데이터를 단순히 샘플링해서 각 모델에 적용다면,
- Boosting은 이전 모델들이 예측하지 못한 Error 데이터에 가중치를 부여하여, 다음 모델이 더 잘 예측하도록 한다.



3. Boosting

Boosting 알고리즘

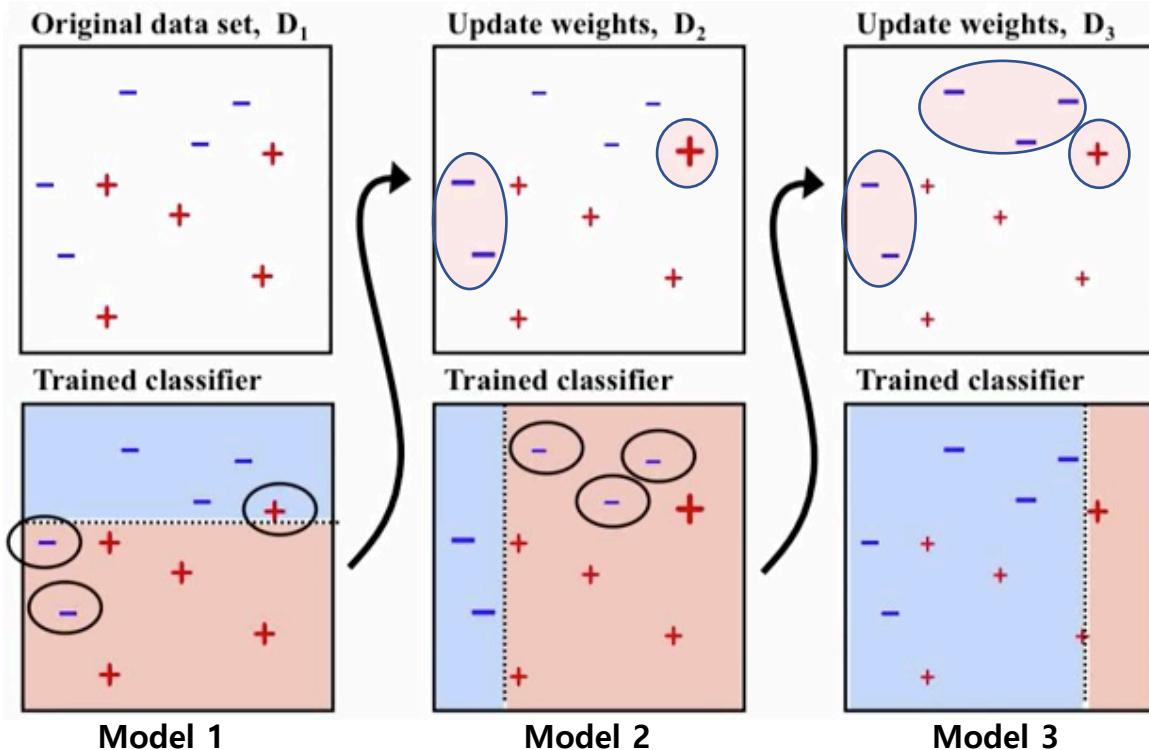
알고리즘	특징	비고
AdaBoost	<ul style="list-style-type: none">다수결을 통한 정답 분류 및 오답에 가중치 부여	
GBM	<ul style="list-style-type: none">Loss Function의 gradient를 통해 오답에 가중치 부여	gradient boosting.pdf
Xgboost	<ul style="list-style-type: none">GBM 대비 성능향상시스템 자원 효율적 활용 (CPU, Mem)Kaggle을 통한 성능 검증 (많은 상위 랭커가 사용)	2014년 공개 boosting-algorithm-xgboost
Light GBM	<ul style="list-style-type: none">Xgboost 대비 성능향상 및 자원소모 최소화Xgboost가 처리하지 못하는 대용량 데이터 학습 가능Approximates the split (근사치의 분할)을 통한 성능 향상	2016년 공개 light-gbm-vs-xgboost

3-1. AdaBoost (Adaptive Boosting)

AdaBoost를 이용하여 데이터를 분류하는 예시

Boosting Example

- Model1에서 잘못 예측한 데이터에 가중치를 부여
- Model2는 잘못 예측한 데이터를 분류하는데 더 집중
- Model3는 Model1, 2가 잘못 예측한 데이터를 분류하는데 집중



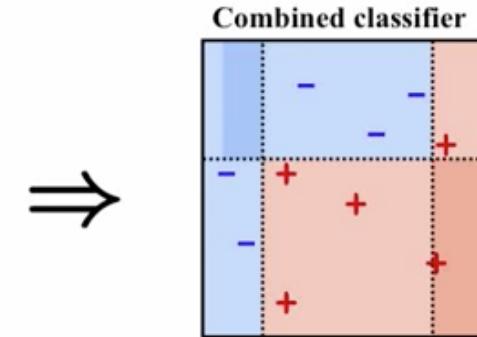
각 모델별 가중치를 고려한 예측 모델

- Cost Function : 가중치(W)를 반영하여 계산

$$J(\theta) = \sum_i w_i J_i(\theta, x^{(i)})$$

- 3개의 모델별로 계산된 가중치를 합산하여 최종 모델을 생성

$$.33 * \begin{array}{|c|}\hline \text{light blue} \\ \hline \text{light orange} \\ \hline \end{array} + .57 * \begin{array}{|c|}\hline \text{light orange} \\ \hline \text{light blue} \\ \hline \end{array} + .42 * \begin{array}{|c|}\hline \text{light blue} \\ \hline \text{light orange} \\ \hline \end{array} \geq 0$$



1-node decision trees
"decision stumps"
very simple classifiers

3-1. AdaBoost

AdaBoost 학습을 위한 개념

오류에 초점을 맞춘 모델 학습

- 각 weak 모델에서 학습할 데이터 선택
- 모든 데이터의 가중치 초기화 (동일한 값)
- 1회 학습 후 예측 오류(ε) 계산, 가중치(α) 계산, 가중치(D) 갱신
- 반복 회수별로 가중치 갱신
- 모든 모델이 위의 단계를 수행할 때 까지 반복

• 가중치(D) : 모든 train 데이터에 적용 (초기값 동일)

• 오류 (ε) : $\frac{\text{오류 데이터}}{\text{전체 학습 데이터}}$, 각 모델의 오류

• 모델별 가중치(α) : $\frac{1}{2} \ln \left(\frac{1-\varepsilon}{\varepsilon} \right)$, 오류를 기반으로 계산

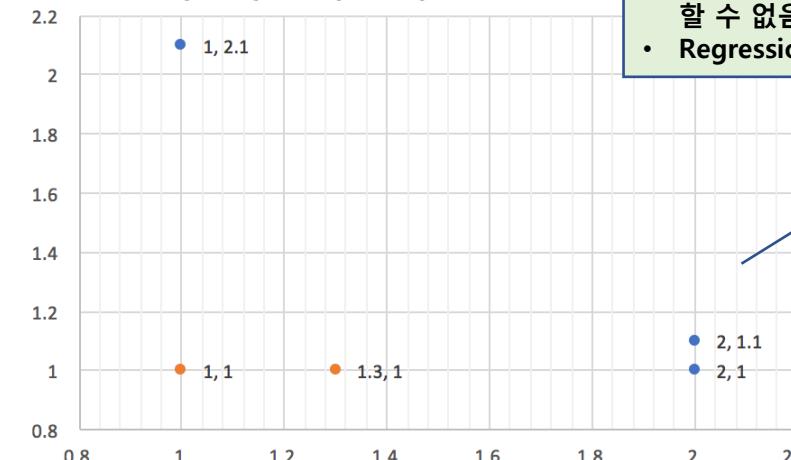
• $D_i^{(t+1)} : \frac{D_i^{(t)} e^{-\alpha}}{\text{Sum}(D)}$ (예측이 맞는 경우)

• $D_i^{(t+1)} : \frac{D_i^{(t)} e^{\alpha}}{\text{Sum}(D)}$ (예측이 틀린 경우)

Weak Classifier (약한 분류기)

- 학습할 데이터 : dataMat
- 정답 데이터 : classLabels

```
>>> dataMat, classLabels = adaboost.loadSimpData()
>>> dataMat
matrix([[ 1. ,  2.1],
       [ 2. ,  1.1],
       [ 1.3,  1. ],
       [ 1. ,  1. ],
       [ 2. ,  1. ]])
>>> classLabels
[1.0, 1.0, -1.0, -1.0, 1.0]
```



- 하나의 축으로 2종류의 데이터를 분할 할 수 없음 (45 problem)
- Regression은 쉽게 가능

- 이를 의사결정 트리로는 위 데이터를 해결하지 못함

3-1. AdaBoost

AdaBoost에서 사용할 약한 분류기 예시

약한 분류기 학습 절차

```
>>> mat(ones((5,1))/5)
matrix([[ 0.2],
       [ 0.2],
       [ 0.2],
       [ 0.2],
       [ 0.2]])

>>> D = mat(ones((5,1))/5)
>>> adaboost.buildStump(dataMat, classLabels, D)
split: dim 0, thresh 0.90, thresh ineqal: lt, the weighted error is 0.400
split: dim 0, thresh 0.90, thresh ineqal: gt, the weighted error is 0.600
split: dim 0, thresh 1.00, thresh ineqal: lt, the weighted error is 0.400
split: dim 0, thresh 1.00, thresh ineqal: gt, the weighted error is 0.600
```

- 가중치 초기화 (0.2)

- Feature의 값을 단계별로 증가시키면서
- 오류 계산

```
def buildStump(dataArr, classLabels):
    dataMatrix = mat(dataArr); labelMat = mat(classLabels).T
    m, n = shape(dataMatrix)
    numSteps = 10.0; bestStump = {} ; bestClasEst = mat(zeros((m,1)))
    minError = inf #init error sum, to +infinity
    for i in range(n):#loop over all dimensions (feature 수 만큼 반복)
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();
        stepSize = (rangeMax-rangeMin)/numSteps
        for j in range(-1,int(numSteps)+1):#loop over all range in current dimension
            for inequal in ['lt', 'gt']: #go over less than and greater than
                threshVal = (rangeMin + float(j) * stepSize)
                predictedVals = stumpClassify(dataMatrix,i,threshVal,inequal)#call stump classify with i, j, lessThan
                errArr = mat(ones((m,1)))
                errArr[predictedVals == labelMat] = 0
                weightedError = D.T*errArr #calc total error multiplied by D
                print "split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, inequal, weightedError)
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst
```

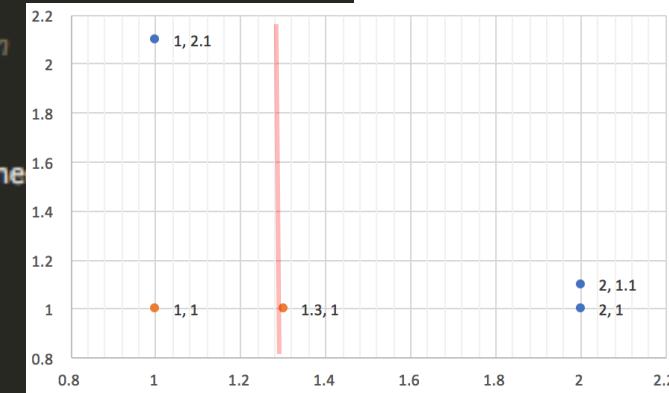
학습 결과 출력

```
>>> bestStump, minError
({'dim': 0, 'ineq': 'lt', 'thresh': 1.3}, matrix([[ 0.2]]))

>>> bestClasEst
array([[-1. ,  1. , -1. , -1. ,  1.]])
```

- Feature 0 번째의 값이 1.3보다 큰 분류 기의
- 예측 오류가 0.2로 제일 낮음

- 실제 예측한 결과값
- 정답과 비교해 보면 첫번째만 -1로
- 틀리게 예측함



3-1. AdaBoost

AdaBoost 학습 알고리즘 구현

학습 알고리즘

```
def adaBoostTrainDS(dataArr,classLabels,numIt=40):
    weakClassArr = []
    m = shape(dataArr)[0]
    D = mat(ones((m,1))/m) #init D to all equal
    aggClassEst = mat(zeros((m,1)))
    for i in range(numIt):
        bestStump,error,classEst = buildStump(dataArr,classLabels,D)#build Stump
        print "D:",D.T
        #calc alpha, throw in max(error,eps) to account for error=0
        alpha = float(0.5*log((1.0-error)/max(error,1e-16)))
        bestStump['alpha'] = alpha
        weakClassArr.append(bestStump) #store Stump Params in Array
        print "classEst: ",classEst.T
        #exponent for D calc, getting messy
        expon = multiply(-1*alpha*mat(classLabels).T,classEst)
        D = multiply(D,exp(expon)) #Calc New D for next iteration
        D = D/D.sum()
        #calc training error of all classifiers,
        #if this is 0 quit for loop early (use break)
        aggClassEst += alpha*classEst
        print "aggClassEst: ",aggClassEst.T
        aggErrors = multiply(sign(aggClassEst) != mat(classLabels).T,ones((m,1)))
        errorRate = aggErrors.sum()/m
        print "total error: ",errorRate
        if errorRate == 0.0: break
    return weakClassArr,aggClassEst
```

```
>>> classLabels
```

```
[1.0, 1.0, -1.0, -1.0, 1.0]
```

```
classifierArray = adaboost.adaBoostTrainDS(datMat, classLabels, 9)
```

학습 결과 출력

- 학습데이터(datMat)의 가장 에러가 낮은 모델 선택 buildStump()
- 모델 가중치(α) 계산
- 데이터 가중치(D) 업데이트
- 선택된 모델에 가중치를 곱하여 예측값 계산
- 이전 모델의 결과값에 누적 (합산)
- 예측값과 정답 비교하여 오류 계산
- 오류가 없으면 종료

```
D: [[ 0.2  0.2  0.2  0.2  0.2]]
classEst: [[-1.  1. -1. -1.  1.]]
aggClassEst: [[-0.69314718  0.69314718 -0.69314718 -0.69314718  0.69314718]]
total error:  0.2
D: [[ 0.5  0.125  0.125  0.125  0.125]]
classEst: [[ 1.  1. -1. -1. -1.]]
aggClassEst: [[ 0.27980789  1.66610226 -1.66610226 -1.66610226 -0.27980789]]
total error:  0.2
D: [[ 0.28571429  0.07142857  0.07142857  0.07142857  0.5      ]]
classEst: [[ 1.  1.  1.  1.  1.]]
aggClassEst: [[ 1.17568763  2.56198199 -0.77022252 -0.77022252  0.61607184]]
total error:  0.0
```

- 첫번째 학습 : 첫번째 값을 -1로 잘못 예측 (가중치 값이 -0.69) → 가중치 D에 0.5 부여
- 두번째 학습 : 마지막 값을 -1로 잘못 예측 (가중치 -0.27) → 가중치 D에 0.5 부여
- 세번째 학습 : 오류 없음.
→ 이전 학습에서 틀린 데이터에 가중치를 추가하면서, 모델의 정확도 향상

3-1. AdaBoost

AdaBoost 검사 알고리즘 구현

약한 분류기 학습 절차

```
def adaClassify(datToClass,classifierArr):
    dataMatrix = mat(datToClass)#do stuff similar to last aggClassEst in adaBoostT
    m = shape(dataMatrix)[0]
    aggClassEst = mat(zeros((m,1)))
    for i in range(len(classifierArr)):
        classEst = stumpClassify(dataMatrix,classifierArr[0][i]['dim'],
                                  classifierArr[0][i]['thresh'],
                                  classifierArr[0][i]['ineq']) #call stump classify
        aggClassEst += classifierArr[0][i]['alpha']*classEst
        print aggClassEst
    return sign(aggClassEst)
```

학습 결과 출력

- **dataToClass** : 검사할 데이터
- **classifierArr** : 학습결과 (weak model의 집합)
- Weak model의 갯수만큼 반복
 - 각 weak model별 feature와 임계값을 기준으로 분류 및 예측
 - 예측 결과를 모델의 가중치를 곱함
 - 전체 모델의 결과를 누적 계산 (aggClassEst)
- 최종 누적결과를 sign함수로 변환 (-1 or 1)
- Weak model의 개수가 많아지면, 좀 더 정확한 예측이 가능

```
datMat, classLabels = adaboost.loadSimpData()
classifierArray = adaboost.adaBoostTrainDS(datMat, classLabels, 9)

>>> adaboost.adaClassify([0,0], classifierArray)
[[ -0.69314718]
 [ -1.66610226]
 matrix([[ -1.]])

>>> adaboost.adaClassify([[5,5],[0,0]], classifierArray)
[[ 0.69314718]
 [-0.69314718]
 [[ 1.66610226]
 [-1.66610226]]
 matrix([[ 1.],
 [-1.]])
```

3-2. GBM(Gradient Boosting)

Gradient Boosting의 개념 및 학습 절차

개념	고려사항
<ul style="list-style-type: none">AdaBoost과 기본 개념은 동일하고,가중치(D)를 계산하는 방식에서Gradient Descent를 이용하여 최적의 파라미터를 찾아낸다.	<ul style="list-style-type: none">2가지 의문점<ul style="list-style-type: none">정말 white noise가 아닌 error가 식별가능해?만약 가능하다면, 예측 정확도가 거의 100% 가능?Boosting은 과적합 가능성성이 높아서, 적절한 시점에 멈춰야 함

[Easy Example]

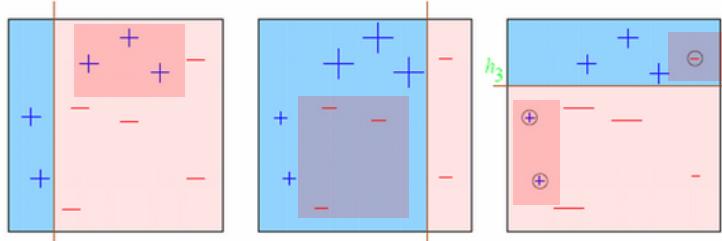
- Model의 예측정확도 80%인 함수 $Y = M(x) + \text{error}$
- 만약 error를 줄일 수 있다면? (error가 Y와 연관성이 있을 경우)
- 정확도 84%로 증가
 - 이렇게 error를 세분화하여 $\text{error} = G(x) + \text{error}_2$
 - $\text{error}_2 = H(x) + \text{error}_3$
 - 정리한 모델의 함수 $Y = M(x) + G(x) + H(x) + \text{error}_3$
- 각 함수별 최적 weight를 찾으면, 예측정확도는 더 높아짐.
- Gradient Descent 알고리즘으로 최적 weight 계산

$$Y = \alpha * M(x) + \beta * G(x) + \gamma * H(x) + \text{error}_4$$

3-2. GBM(Gradient Boosting)

Gradient Boosting의 개념 및 학습 절차

Gradient Descent를 이용한 weight 계산



- 1번 Weak model에서는 3개의 오분류(에러)가 발생
- 2번은 3개 에러를 제대로 분류하기 위해 가중치 부여. (다시 3개 에러 생김)
- 3번은 다시 3개 에러를 해결하기 위한 모델 생성 (다시 3개 에러 발생)
- 최적의 weight(가중치)를 찾을 때 까지 반복

[그럼 어떻게 가중치를 부여할까?]

- 초기 데이터 가중치(D) = $1/n$ (n : 전체 학습 데이터 개수)
- 오류 (ε) : $\frac{\text{오류 데이터}}{\text{전체 학습 데이터}}$, 각 모델의 오류
- 모델의 가중치(α) = $\frac{1}{2} \ln \left(\frac{1-\varepsilon}{\varepsilon} \right)$
- Weak model의 함수 : $h(t)$
- 가중치 업데이트(D) :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

데이터 가중치는 어떻게 최적화하나?

- α : 는 learning rate 역할을 수행, 지수함수의
- y : 정답 (1 or -1)
- $h(x)$: 모델의 예측값

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

- \exp 함수의 인자값(α)으로 학습의 방향(최적의 weight 탐색)을 확인
- 만약 학습이 잘못 되고 있다면, $-\alpha y_i h_t(x_i)$
 - $-\alpha * 1 * -1 = \alpha$ (정답은 1, 예측은 -1)
 - $-\alpha * -1 * 1 = \alpha$ (정답은 -1, 예측은 1)
- 만약 학습이 잘되고 있으면
 - $-\alpha * 1 * 1 = -\alpha$ (정답은 1, 예측은 1)
- 따라서, 예측이 틀리면 가중치(D) 증가

3-3. XGBoost (eXtreme Gradient Boosting)

XGBoost의 개념

개념

- XGBoost ?
 - GBM + 분산/병렬 처리
 - 지도학습으로 변수(x)를 학습하여 정답(y)을 예측

- Xgboost가 지원하는 모델
 - Binary classification
 - Multiclass classification
 - Regression
 - Learning to Rank

지도학습 용어

- Model : 변수(x)로 정답(y)를 예측하는 함수 $\hat{y}_i = \sum_j \theta_j x_{ij}$
- θ : 세타(가중치, 파라미터)
 - 학습을 통해 정답을 잘 예측하도록 조정

- Objective Function
 - 학습 데이터에 최적화된 파라미터 찾는 함수
 - Training Loss + Regularization

$$Obj(\Theta) = L(\theta) + \Omega(\Theta)$$

- L : Training Loss (Loss Function)

$$L(\theta) = \sum (y_i - \hat{y}_i)^2$$

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})]$$

Logistic regression

- Ω : Regularization

- 모델의 복잡도를 조절 (과적합 방지)

<http://xgboost.readthedocs.io/en/latest/model.html>

<https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/>

<https://www.slideshare.net/ShuaiZhang33/rg-xgboost20170306>

3-3. XGBoost (eXtreme Gradient Boosting)

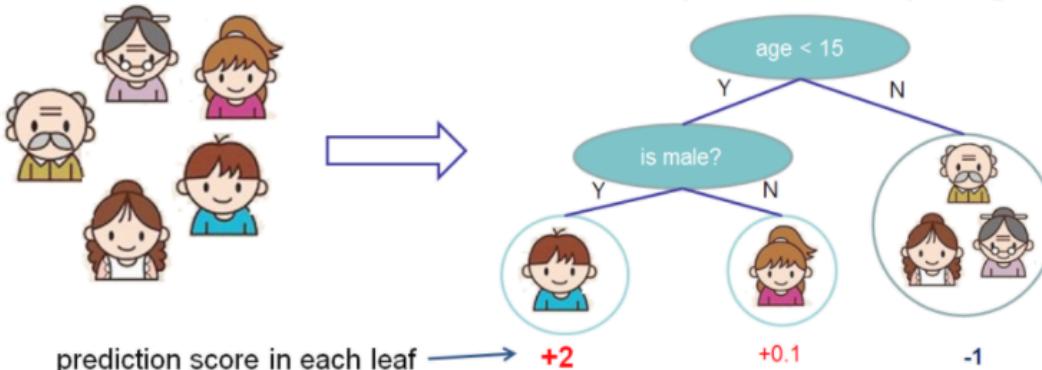
1) Tree Ensemble의 model 이해 (XGBoost에서 사용하는 모델로 CART의 집합)

CART 예시

- 이제 xgboost에서 사용하는 모델(tree ensemble)에 대한 소개
- Tree ensemble은 여러 개의 CART model로 구성
- 컴퓨터 게임을 좋아하는지에 따라 사람들을 CART로 분류한 예시
- 여러개 잎(leaf)으로 분류하고, 각 leaf별 점수(score) 할당 (CART의 특징)
 - 기존 Decision Tree는 Leaf에 결정값(decision value)만 존재함.
- 이 Score를 이용해서 분류외에 풍부한 해석이 가능(설명력)

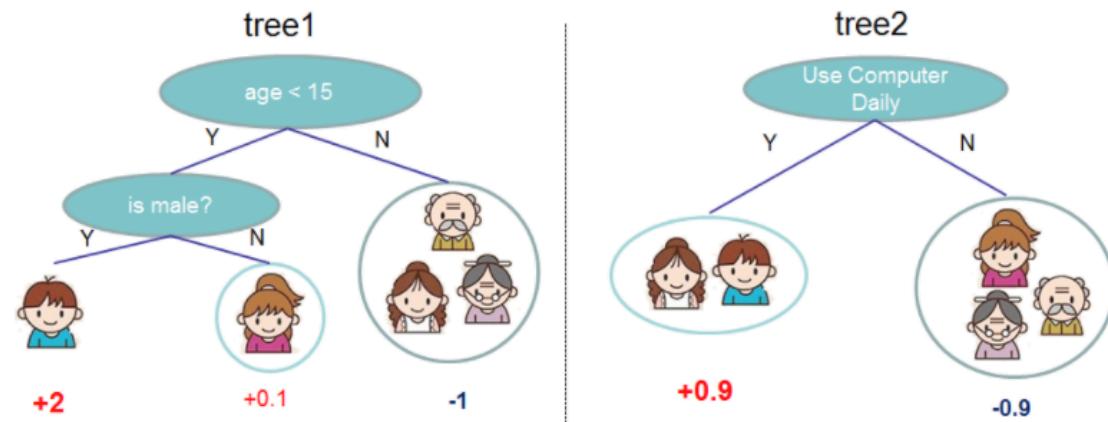
Input: age, gender, occupation, ...

Does the person like computer games



Tree Ensemble (단일 모델의 학습 복제)

- 2개의 tree를 이용하여 양상을 모델을 생성



- 2개 tree의 개별 예측점수(score)를 합산하여 최종 score를 계산
- 양상을 모델을 통해 tree간 모델을 상호보완

$$f(\text{boy}) = 2 + 0.9 = 2.9 \quad f(\text{elderly man}) = -1 - 0.9 = -1.9$$

3-3. XGBoost (eXtreme Gradient Boosting)

2) Tree Ensemble의 모델의 이해

CART 예시

- 선형 모델에서 학습의 목적은
 - Training loss를 최적화 → 학습 데이터에 모델이 최적화 되었는지 측정
 - Regularization 최적화 → 모델 단순화(복잡도 감소)

$$\hat{y}_i = \sum_j \theta_j x_{ij} \quad Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

Training Loss

Regularization

- K개의 Tree가 있다고 가정하면,

- 예측 모델은

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), f_k \in \mathcal{F}$$

- K : tree 개수 (예시는 2개)

- F : CART의 모든 regression tree 세트

- f : F 공간의 함수

- 최적화 함수는

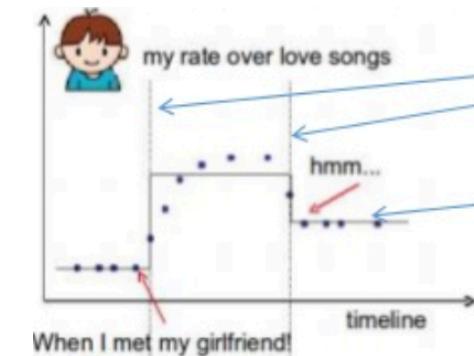
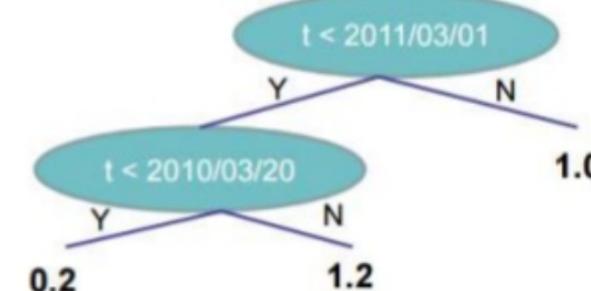
$$obj(\theta) = \sum_i l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

- 학습할 파라미터(Weight)는

- 각 tree의 구조와 leaf의 score → 수식은 $\Theta = \{f_1, f_2, \dots, f_K\}$
- 여기서는 weight(값)을 학습하는 대신, 함수(tree)를 학습한다!!

어떻게 function(tree)을 학습할까?

- 아래 예시는 시간을 기준으로 분할된 regression tree 모델이다



Splitting
positions

The height
in each
segment

3-3. XGBoost (eXtreme Gradient Boosting)

2) Xgboost의 학습방법 (여러 tree의 결과를 반영)

Tree Boosting

- 어떻게 Tree를 학습할까?
 - Objective Function 정의 → 최적화 (optimization)

- 아래와 같은 objective 함수가 있다고 가정

$$\text{Training Loss (Loss Function)} \quad \text{obj} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \quad \text{Regularization}$$

- 그럼 tree는 어떤 parameter가 있을까?

- 각 트리별로 트리의 구조와 leaf score를 가진 함수 f 있음

- 이제 가지고 있는 트리에 대한 parameter를 최적화 하자!

- 그런데, 기존 최적화(SGD) 함수는 한번에 여러개의 트리 최적화가 어려움

- 따라서, 기존 model 함수를 변경하여 한번에 하나의 트리를 추가하여

- 각 단계(t)별 결과값을 예측

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Additive Training

- 각 단계에서 어떤 트리를 선택해야 할까?

- 각 단계에서 트리의 Loss를 계산 (단계별 Loss 확인 가능)

$$\begin{aligned} \text{obj}^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant} \end{aligned}$$

- 만약 Loss Function이 MSE라면 아래와 같은 수식이 도출

$$\begin{aligned} \text{obj}^{(t)} &= \sum_{i=1}^n (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n [2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + \text{constant} \end{aligned}$$



Logistic regression과 같이 복잡한 Loss Function은
단순한 수식으로 정의하기 어려움

(Taylor 급수 활용)

3-3. XGBoost (eXtreme Gradient Boosting)

3) Tayler 급수를 이용하여 다양한 loss function 지원

다양한 Loss function 지원

- Objective 함수가 너무 복잡해짐 (Logistic regression등 다른 loss function)

- Tayler expansion을 이용하여 2차 다항식으로 단순화

$$\text{obj}^{(t)} = \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

- g 와 h 함수는 아래와 같이 정의

gradient

Hessian
(2차 미분)

다양한 Loss Function을 대입

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) -$$

$$h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$$

- 기존 공식에서 상수를 제거하면 아래의 t 단계의 최종 공식 도출

$$\sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

- 위 공식이 새로운 트리에 대한 학습을 최적화 할 수 있다.

- 이렇게 공식을 단순화해서 xgboost가 가지는 장점은

- g, h 함수만 수정하면, 다양한 loss function을 지원 가능함.

모델 복잡도 (Regularization 정의)

- 모델의 복잡도(regularization)는 ?

- 먼저 트리 함수 $f_{(x)}$ 를 아래와 같이 정의

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \dots, T\}$$

- w : leaf 노드의 score

- q : leaf 노드에 할당되는 함수

- T : leaf 노드의 개수

- XGBoost에서 모델의 복잡도 수식

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

- 기존 Decision tree 학습에서 regularization은 잘 사용하지 않음

- 보통 impurity(불순도)에 초점을 맞추어 학습하였으나,

- 최근에는 regularization을 활용하여 모델의 복잡도를 조정함

3-3. XGBoost (eXtreme Gradient Boosting)

4) Tree의 leaf에 Objective value (score)를 부여

t 번째 트리의 objective value(score, loss) 계산

- 복잡한 Objective 함수를 아래의 과정을 거쳐서 단순하게 수식으로 정리

$$\sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

\downarrow

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \dots, T\}$$
$$Obj^{(t)} \approx \sum_{i=1}^n [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$
$$= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$$

$I_j = \{i | q(x_i) = j\}$ j 번째 leaf에 할당된 데이터 포인트의 인덱스 세트

- 마지막 라인의 공식에서 변경된 것은
- 같은 leaf에 속한 모든 데이터 포인트는 같은 score를 가지므로,
- 합계를 구하는 수식을 변경함

- 수식을 좀 더 단순하게 해보자

- 아래 2 변수로 수식을 대입하면, 조금 더 단순해 진다.

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

$$obj^{(t)} = \sum_{j=1}^T [G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2] + \gamma T$$

- 여기서 w 는 서로 독립적인 변수이므로,
- $q_{(x)}$ (leaf에 할당된 함수)에 최적화된 w 계산

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

- 최소의 Objective 계산 가능

$$obj^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

27

w_j^* 는 어떻게
도출되었나?

w에 대입

- 이 objective value를 통해 모델(트리 구조, $q_{(x)}$) 평가 가능 (얼마나 좋은지)
- 낮을수록 더 좋은 트리구조를 의미함.

3-3. XGBoost (eXtreme Gradient Boosting)

5) 트리를 분리하는 알고리즘

트리구조를 학습해보자

- 이상적으로는 가능한 모든 트리를 생성하고, 이중 가장 최적의 트리를 선택.
- 하지만, 실제 그렇게 적용하기 어려우므로,
- 한번에 1단계의 트리를 최적화한다.
- 구체적으로 설명하면, 하나의 leaf를 2개로 분리하고 Score를 측정
 - 아래의 공식으로 score 측정 가능

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

원쪽 Leaf Score 오른쪽 Leaf Score 원본 Leaf Score Regularization

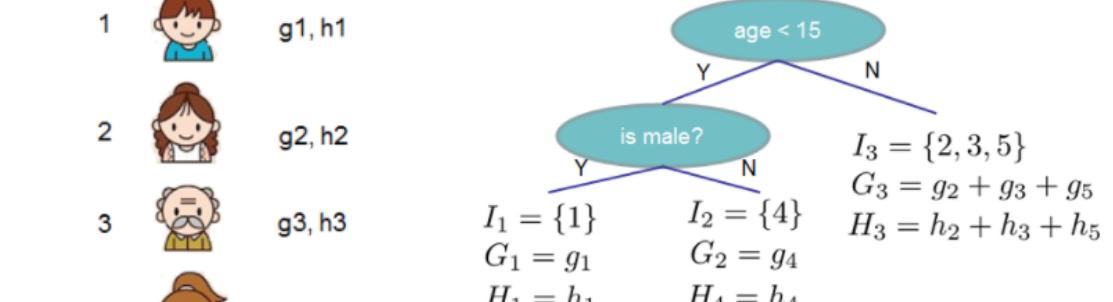
- 위 공식에서 중요한 사실은 만약 Node(leaf)를 분리한 Score(Gain)이 γ 보다 작다면, Node를 분리하지 않는 것이 좋다는 의미임. → Pruning 지원

어떻게 트리를 최적으로 분리할까?

- 아래 데이터에서 “age < 15” 를 기준으로 데이터를 2개로 분리 → Gain 확인

Instance index gradient statistics

1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



- 분리된 데이터는 Gain 공식을 이용해 빠르게 계산 (정렬된 순서대로)
 - 이때 g,h는 사전에 정의되어 있어야 함

			\rightarrow
g1, h1	g4, h4	g2, h2	g5, h5
$G_L = g_1 + g_4$		$G_R = g_2 + g_3 + g_5$	

3-3. XGBoost (eXtreme Gradient Boosting)

5) 트리를 분리하는 알고리즘

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I , instance set of current node

Input: d , feature dimension

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$

for $k = 1$ **to** m **do**

$G_L \leftarrow 0, H_L \leftarrow 0$

for j in $\text{sorted}(I, \text{ by } \mathbf{x}_{jk})$ **do**

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

end

end

Output: Split with max score

- For each node, enumerate over all features
 - For each feature, sorted the instances by feature value
 - Use a linear scan to decide the best split along that feature
 - Take the best split solution along all the features

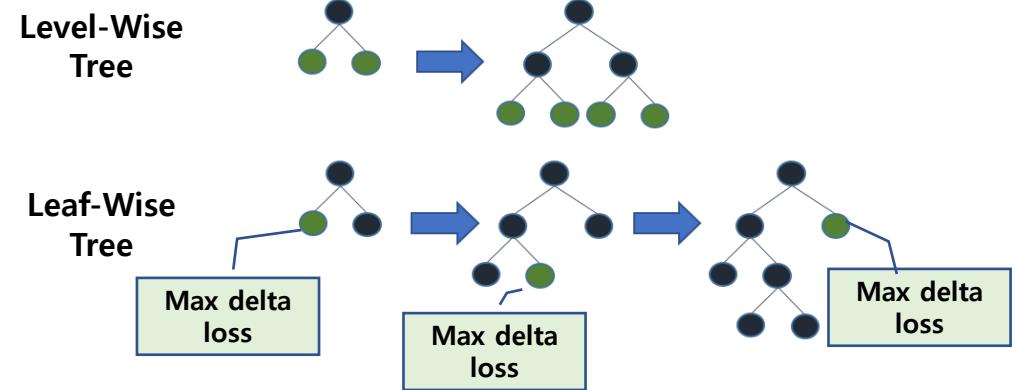
3-4. Light GBM

Light GBM 개념

개념

- Light GBM ?
 - Decision Tree 알고리즘기반의 GBM 프레임워크 (빠르고, 높은 성능)
 - Ranking, classification 등의 문제에 활용
- 무엇이 다른가?
 - Leaf-wise로 tree를 성장(수직 방향) , 다른 알고리즘 (Level-wise)
 - 최대 delta loss의 leaf를 성장
 - 동일한 leaf를 성장할때, Leaf-wise가 loss를 더 줄일 수 있다.
- 왜 Light GBM이 인기 있나?
 - 대량의 데이터를 병렬로 빠르게 학습가능 (Low Memory, GPU 활용가능)
 - 예측정확도가 더 높음(Leaf-wise tree의 장점 → 과접합에 민감)
- 얼마나 빠른가? ([link](#)) : XGBoost 대비 2~10배 (동일한 파라미터 설정시)
- 그렇게 좋은데 왜 많이 안쓰지?
 - Light GBM이 설치된 툴이 많이 없음. XGBoost(2014), Light GBM(2016)
- 어디에 활용해야 하나?
 - Leaf-wise Tree는 overfitting에 민감하여, 대량의 데이터 학습에 적합
 - 적어도 10,000 건 이상

Level-wise vs Leaf-wise



- **Level-wise**
 - 각 노드는 root노드와 가까운 노드를 우선 순회, 수평 성장
 - XGBoost, Random Forest
- **Leaf-wise**
 - 가장 Loss변화가 큰 노드에서 데이터를 분할하여 성장, 수직 성장
 - 학습 데이터가 많은 경우 뛰어난 성능
 - Light GBM, XGBoost

3-4. Light GBM

Light GBM에서 데이터를 잘 분할하는 속성을 병렬로 찾는 방법

어떻게 Best Split(최적 분할값)을 찾는가?

- Boosted Decision Tree 학습의 성능 요인
 - Leaf(Node)의 최적 분할값을 찾기 위한 연산비용!
 - 기존 알고리즘은 각 leaf의 정확한 분할을 위해 모든 데이터 탐색
 - Light GBM은 feature들의 histogram을 만들어 **근사치의 분할**을 수행
- Approximates the split (근사치의 분할)
 - 학습 데이터량이 증가하면서,
 - 기존 Decision tree의 학습을 병렬화 하였으나, Communication비용 증가
 - 이를 해결하기 위해,
 - 분할된 데이터별로 병렬로 학습한 결과에 대한
 - local voting과 global voting 개념을 이용하여 comm 비용 최소화
 - Local voting에서는 가장 최적의 상위 2개 속성 선택
 - Global voting은 local voting의 속성들 중 상위 2개 속성 선택
 - 이로 인한 성능 향상
 - <https://arxiv.org/abs/1611.01276>

Efficient Parallel Algorithm for Decision Tree

- Approximates the split (근사치의 분할)
- 학습 데이터량이 증가하면서,
- 기존 Decision tree의 학습을 병렬화 하였으나, Communication비용 증가
- 이를 해결하기 위해,
 - 분할된 데이터별로 병렬로 학습한 결과에 대한
 - local voting과 global voting 개념을 이용하여 comm 비용 최소화
 - Local voting에서는 가장 최적의 상위 2개 속성 선택
 - Global voting은 local voting의 속성들 중 상위 2개 속성 선택
 - 결과적으로 데이터를 잘(근사치) 분할 할 수 있는 속성 2개가 선택됨.
- 장점
 - 분할된 알고리즘간 comm 비용 최소화
 - 병렬 처리를 위한 scal out 효과적
 - 이로 인한 성능 향상
 - <https://arxiv.org/abs/1611.01276>

**Xgboost에서
weak model로 사용하는 CART ?**

4. Decision Tree

주요 특징 비교

	ID3	CART
분류기준	Shannon entropy Information gain	Gini Index
분류 방식	<ul style="list-style-type: none">Field별 값을 기준으로 분할Grade란는 field에 (A, B, C)라는 값이 있으면, 각 속성별로 데이터를 분류함 (총 3번)	<ul style="list-style-type: none">이진 트리를 사용하여,1개의 field의 값을 특정 기준값을 기준으로크면 right, 작으면 left로 분류
지원 모델	Classification	Classification + Regression
과접합 방지	X	O (Leaf 데이터 갯수, 최소 변화량)
트리	트리	이진트리
회기모델 지원 (수치 데이터)	X	O
장점	구현 용이	부등호 질의가능 사후 가지치기 가능 (Leaf Node 통합) 데이터 해석이 용이(설명력)
단점	수치형 속성 사용불가 카테고리 속성이 많은 경우, Tree가 깊어짐 (가지가 많아지기 때문)	학습데이터가 충분해야 함 (과적합 유발) → 배깅/부스팅 활용

- Entropy**
 - 예를 들어, 동물을 분류할때 “포유류인가?”라는 질문으로 먼저 분류하고,
 - “원숭이인가?”라는 구체적인 질문으로 분류하는 것이 효과적 → 정보이득이 높음
- Gini**
 - 분류된 노드의 데이터 불순도 측정 (얼마나 많은 종류가 있는가?)
 - 만약 노드에 1가지 종류의 데이터만 있다면, Purity(순수) 상태
 - 따라서 불순도를 낮추는 것이 최적의 분류를 위한 feature를 찾는 것

https://www.google.co.kr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&ved=0ahUKEwjSpo6Gm8_XAhVJE7wKHbBPCWUQFghTMAc&url=http%3A%2F%2F164.125.35.15%2Fvisbic_course%2F%3Fmodule%3Dfile%26act%3DprocFileDownload%26file_srl%3D5316722%26sid%3Dd2e0ad2c6357507c6c3dff47e2f4cc92&usg=AOvVaw1RxDnLYHQFpStpHJDGXGLc

<https://www.quora.com/What-are-the-differences-between-ID3-C4-5-and-CART>

<https://stackoverflow.com/questions/9979461/different-decision-tree-algorithms-with-comparison-of-complexity-or-performance>

1. Decision Tree (ID3)

한번에 하나의 속성(feature)으로 데이터 분류하기

어떻게 분류 기준(속성)을 선택할까?

정보이득 계산(entropy)

- 모든 속성에 대하여 정보이득을 계산하여 결정
 - 정보이득 계산 방법 : entropy, gini 방식
- 각 노드별로 위와 같이 정보이득 계산값으로 선택된 속성 선택

- 분류하기 전과 분류 후의 변화량
- 계산값이 가장 높은 값을 선택 (변화량이 큰것)

- [Entropy 방법]
 - 정보에 대한 기대값 계산 공식
 - $I(x_i) = p(x_i) * \log_2 p(x_i)$
 - $p(x_i)$: x_i 항목이 선택될 확률
 - 모든 속성(feature)에 대하여 entropy값 계산하여 합산
 - $H = -\sum_{i=1}^n p(x_i) * \log_2 p(x_i)$
 - n : 속성 개수
 - 값이 높다는 것은 데이터가 혼잡하다는 의미.
 - 속성에 있는 값이 다양하다는 의미

```
>>> myDat  
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
```

```
def calcShannonEnt(dataSet):  
    numEntries = len(dataSet)  
    labelCounts = {}  
    for featVec in dataSet: #the the number of unique elements and their occurrence  
        currentLabel = featVec[-1]  
        if currentLabel not in labelCounts.keys(): labelCounts[currentLabel] = 0  
        labelCounts[currentLabel] += 1  
    shannonEnt = 0.0  
    for key in labelCounts:  
        print labelCounts[key], "/", numEntries  
        prob = float(labelCounts[key])/numEntries  
        shannonEnt -= prob * log(prob,2) #log base 2  
    return shannonEnt
```

```
>>> trees.calcShannonEnt(myDat)
```

```
2 / 5  
3 / 5  
0.9709505944546686
```

- 속성 값별로 count한 값
- 이 값이 크면 계산 값도 커진다

- 'yes' : 2 개
- 'no' : 3 개

1. Decision Tree (ID3)

모든 속성에 대하여 데이터 집합 분할하기

데이터 집합 분할

- Feature별로 데이터를 분할
 - 아래 2개의 feature와 1개의 label('yes', 'no')

```
>>> myDat  
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
```

- 이를 feature에 포함된 값 별로 데이터를 분할한다.
 - 예를 들어 첫번째 feature (index 0)의 값이 1인것
 - [1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no']
 - 제외 → [0, 1, 'no'], [0, 1, 'no']
 - 이 방식으로 feature별, feature에 포함된 값 별로 데이터 분할

데이터 분할 예시

```
def splitDataSet(dataSet, axis, value):  
    retDataSet = []  
    for featVec in dataSet:  
        if featVec[axis] == value:  
            reducedFeatVec = featVec[:axis]      #chop out axis used for splitting  
            reducedFeatVec.extend(featVec[axis+1:])  
            print "reducedFeatVec.extend: ", reducedFeatVec  
            retDataSet.append(reducedFeatVec)  
    return retDataSet
```

```
>>> trees.splitDataSet(myDat, 0, 1)  
reducedFeatVec.extend: [1, 'yes']  
reducedFeatVec.extend: [1, 'yes']  
reducedFeatVec.extend: [0, 'no']  
[[1, 'yes'], [1, 'yes'], [0, 'no']]
```

- 0 : field의 index
- 1 : 0번째 field 값이 1인 것만 분할

```
>>> trees.splitDataSet(myDat, 0, 0)  
reducedFeatVec.extend: [1, 'no']  
reducedFeatVec.extend: [1, 'no']  
[[1, 'no'], [1, 'no']]
```

- 1번째 field의 값이 1인 데이터만 분할

1. Decision Tree (ID3)

데이터 분할시 가장 좋은 속성 선택하기

데이터 집합 분할

```
>>> myDat  
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]  
  
def chooseBestFeatureToSplit(dataSet):  
    numFeatures = len(dataSet[0]) - 1      #the last column is used for the labels  
    baseEntropy = calcShannonEnt(dataSet)  
    bestInfoGain = 0.0; bestFeature = -1  
    for i in range(numFeatures):          #iterate over all the features  
        featList = [example[i] for example in dataSet]#create a list of all the examples of this feature  
        print "Feature List : ", featList  
        uniqueVals = set(featList)           #get a set of unique values  
        newEntropy = 0.0  
        for value in uniqueVals:  
            subDataSet = splitDataSet(dataSet, i, value)  
            prob = len(subDataSet)/float(len(dataSet))  
            newEntropy += prob * calcShannonEnt(subDataSet)  
        infoGain = baseEntropy - newEntropy      #calculate the info gain  
        print "infoGain : ", infoGain  
        if (infoGain > bestInfoGain):          #compare this to the best  
            bestInfoGain = infoGain             #if better than current best  
            bestFeature = i  
    return bestFeature                      #returns an integer  
  
>>> trees.chooseBestFeatureToSplit(myDat)
```

- 0번째 feature가 가장 데이터를 잘 분할
- 정보획득이 가장 큼

작업 흐름

1. 각 Feature에 포함된 값 추출 (featList)
2. featList 중복제거 → uniqueList
3. 각 feature를 속성값 기준으로 분할
4. 각 feature별 분할된 데이터의 entropy 계산
5. 원본 데이터 entropy - 분할된 데이터 entropy
 - Feature별로 entropy가 계산됨
 - 차이가 가장 큰 값이
 - 데이터를 가장 잘 분할
6. 가장 차이가 큰 feature 선택

```
Feature List : [1, 1, 1, 0, 0]  
reducedFeatVec.extend: [1, 'no']  
reducedFeatVec.extend: [1, 'no']  
2 / 2  
reducedFeatVec.extend: [1, 'yes']  
reducedFeatVec.extend: [1, 'yes']  
reducedFeatVec.extend: [0, 'no']  
2 / 3  
1 / 3  
infoGain :  0.419973094022  
Feature List : [1, 1, 0, 1, 1]  
reducedFeatVec.extend: [1, 'no']  
1 / 1  
reducedFeatVec.extend: [1, 'yes']  
reducedFeatVec.extend: [1, 'yes']  
reducedFeatVec.extend: [0, 'no']  
reducedFeatVec.extend: [0, 'no']  
2 / 4  
2 / 4  
infoGain :  0.170950594455
```

[F1 feature 분류]
• 속성 1 : YES(2), NO(1)
• 속성 0 : NO(2)
→ F1으로 분류하는 것이 효과적

F1	F2	Label
1	1	YES
1	1	YES
1	0	NO
0	1	NO
0	1	NO

[F2 feature 분류]
• 속성 1 : YES(2), NO(2)
• 속성 0 : NO(1)

1. Decision Tree (ID3)

이제 트리를 만들어 보자 (재귀호출)

데이터 분할을 반복하여, tree를 생성

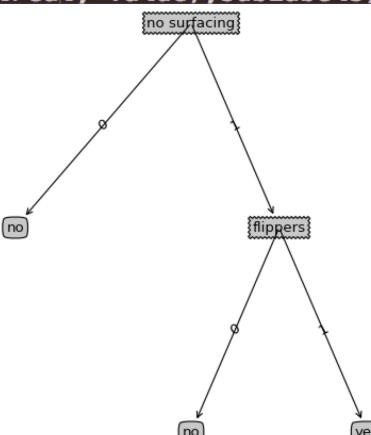
```
>>> myDat  
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]  
  
def createTree(dataSet, labels):  
    classList = [example[-1] for example in dataSet]  
    if classList.count(classList[0]) == len(classList):  
        return classList[0] #stop splitting when all of the classes are equal  
    if len(dataSet[0]) == 1: #stop splitting when there are no more features in dataSet  
        return majorityCnt(classList)  
    bestFeat = chooseBestFeatureToSplit(dataSet)  
    bestFeatLabel = labels[bestFeat]  
    myTree = {bestFeatLabel:{}}  
    del(labels[bestFeat])  
    featValues = [example[bestFeat] for example in dataSet]  
    uniqueVals = set(featValues)  
    for value in uniqueVals:  
        subLabels = labels[:].copy() #copy all of labels, so trees don't mess up existing labels  
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)  
    return myTree
```

[단점]

- 학습 데이터와 완벽히 일치 → 과적합 → 가지치기 필요
- 수치형 데이터 처리 불가 → 이산형으로 변환

작업 흐름

- 분류된 데이터의 label이 동일하면 종지
 - classList : myDat에서 정답(yes, no)만 추출(['yes', 'yes', 'no', 'no', 'no'])
 - 모든 정답이 동일하면 종지(['yes', 'yes'])
- dataSet(myDat)에 분류할 속성이 없으면 종지
 - classList에 가장 많은 정답(yes or no)을 반환
- 정보이득이 높는 속성(field) 선택
 - Label 값을 함께 저장
 - ['no surfacing', 'flippers']
 - 선택된 label은 삭제
- 선택된 속성의 값 별로 tree 생성
 - 여기서는 '0', '1' 별로 tree 생성
 - 0인 경우 tree는 모두 'no'
 - 1인 경우 tree는 다시 최적의 속성 선택

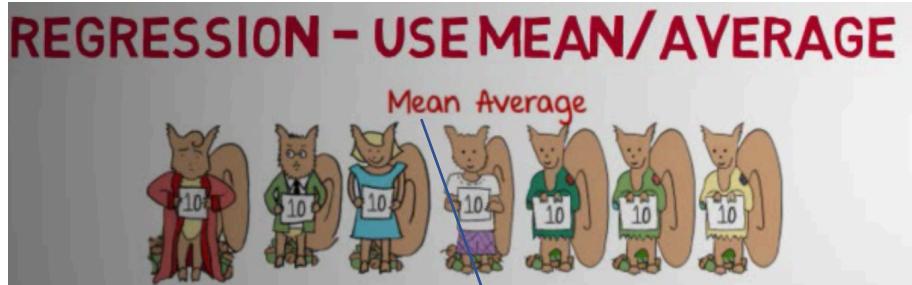


1. Decision Tree (CART)

Classification And Regression Trees, 트리 기반 회기

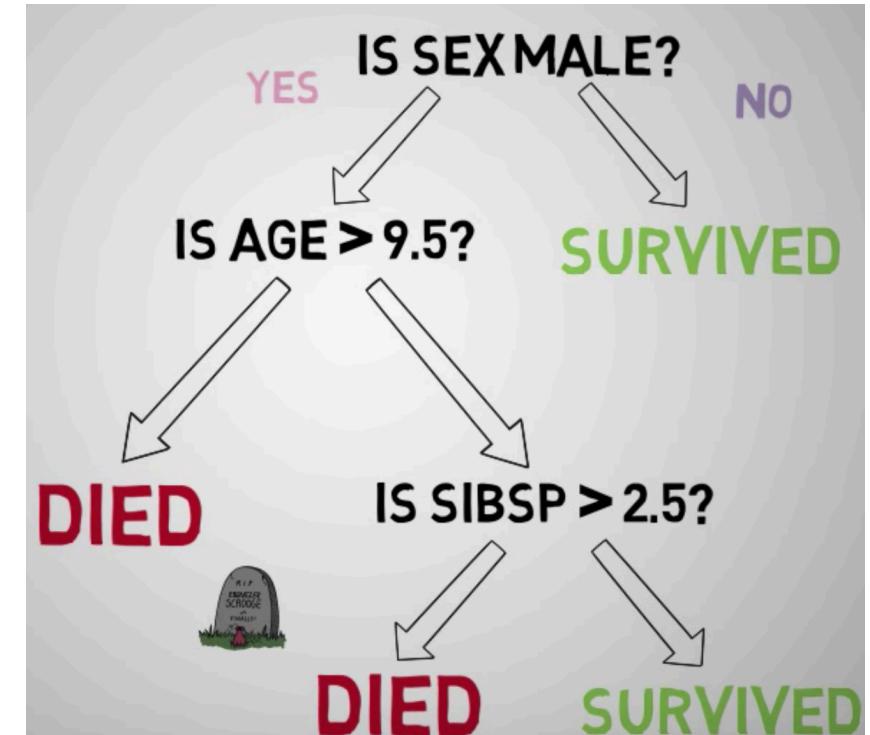
Classification vs Regression Tree 비교

- Classification : 카테고리컬 변수에 사용
- Regression : 연속형 변수에 사용



함수(알고리즘)을 어떻게 트리로 표현할까?

- 많이 사용하는 타이타닉 생존자를 분류하는 예시를 보자
- 수치형 값을 기준으로 3개의 알고리즘(함수)으로 데이터를 분류



- 수치형 값의 평균으로 분류
• 선형으로 수치값을 분류할 때.

1. Decision Tree (CART)

Classification And Regression Trees, 트리 기반 회기

트리를 성장하는 방식

1) FEATURES TO CHOOSE

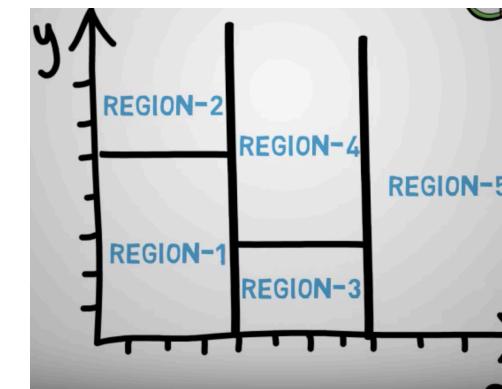
2) CONDITIONS FOR SPLITTING

3) KNOWING WHEN TO STOP

4) PRUNING

트리는 어디서 분류할지 어떻게 결정하는가?

- 하나의 알고리즘으로 모든 데이터 분류 불가(여러개 알고리즘 사용)
- 먼저 상위노드를 2개 이상의 노드로 분류
- 하위노드를 많이 생성하면, 하위노드의 데이터의 불순도가 낮아짐(한가지 유형의 데이터만 분류됨)
 - → 즉, 더 많은 질문을 할 수록 정답에 가까워짐 (과적합 가능성도 높아짐)
- 트리는 노드에 있는 모든 변수(중복제거)를 기준으로 데이터를 분류하고, 이 중에서 가장 불순도가 낮은 변수를 선택.

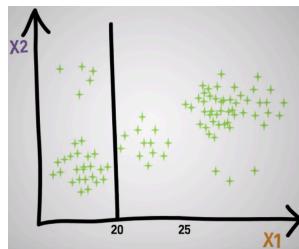
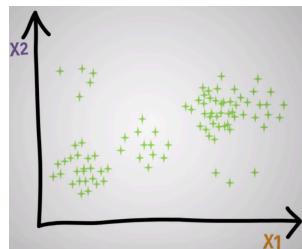


1. Decision Tree (CART)

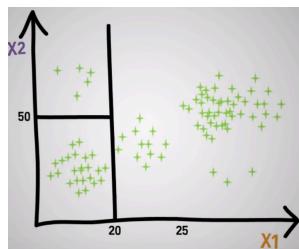
Classification And Regression Trees, 트리 기반 회기

실제 CART를 이용한 분류 예시

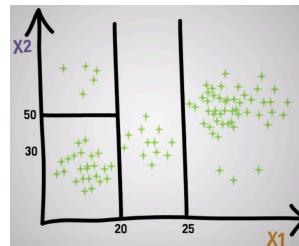
- 아래와 같이 임의 데이터를 생성
- 속성(feature) : X1, X2



- X1의 20을 기준으로 데이터 분류하는 함수 추가

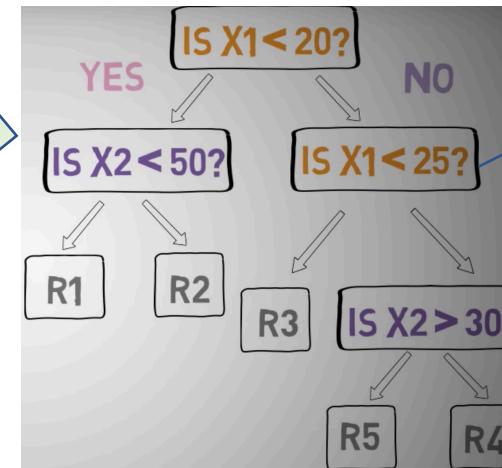
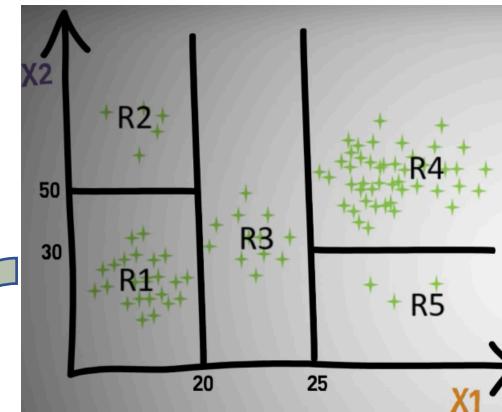


- 위 분류에 X2의 50을 기준으로 분류하는 함수 추가



분류 알고리즘을 트리로 표현

- 데이터를 분류한 함수(알고리즘)를 노드로 표현



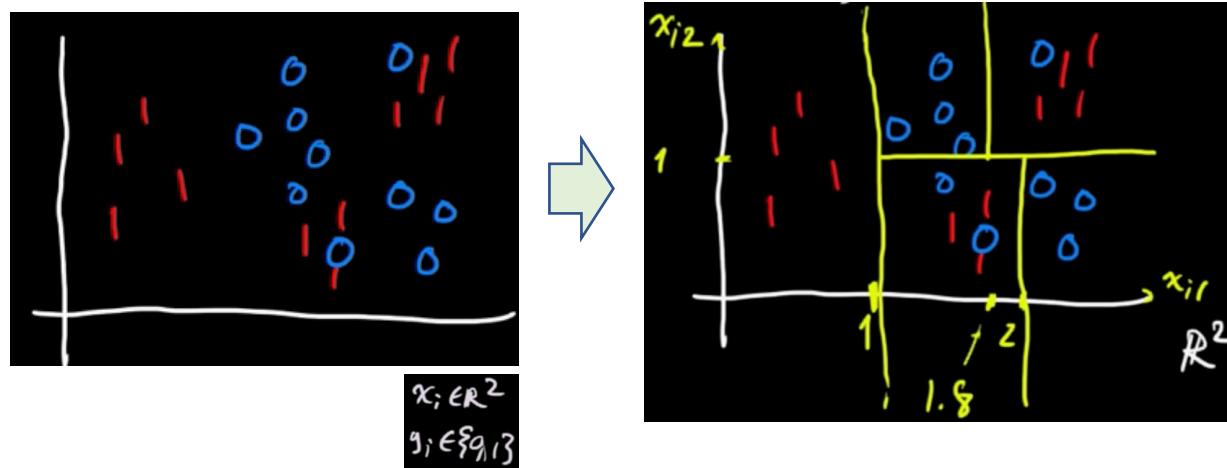
- 하나의 트리가 많은 함수(노드)로 구성되며,
- 노드가 많아질 수록 데이터를 더 잘 분류하게 됨. (과적합 주의)

1. Decision Tree (CART)

각 leaf node의 학습 에러(불순도)를 최소화 하는 Binary Tre

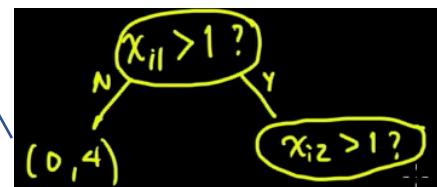
Classification Tree 예시

- 아래와 같이 임의 데이터를 생성
- Decision Tree는 데이터를 2개로 분할(Binary split)한다.
 - 이때 분류된 노드의 에러(불순도)를 최소화하는 분류기준을 선택



- 데이터를 binary로 분류하면, 아래과 같은 트리가 생성됨

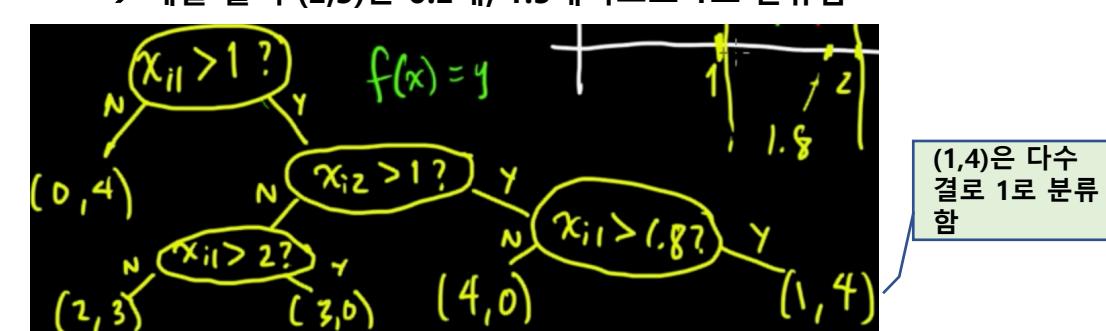
불순도 = 0
모두 1로 구성됨
더 이상 분류할 필요 없음



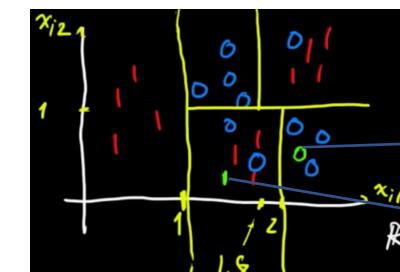
불순도 = 0:10개, 1:7개
불순도를 최소화하기 위해
데이터 분류

분류 알고리즘을 트리로 표현

- 그런데 특정 leaf node는 불순도가 높은 것이 보인다
- 이런 leaf에서는 어떻게 데이터를 분류할까?
 - 다수결을 이용하여 분류
 - 예를 들어 (2,3)은 0:2개, 1:3개이므로 1로 분류함



그럼 새로운 데이터가 입력되면 어떻게 분류할까?



(3,0)은 0이 3개이므로 "0"으로 분류

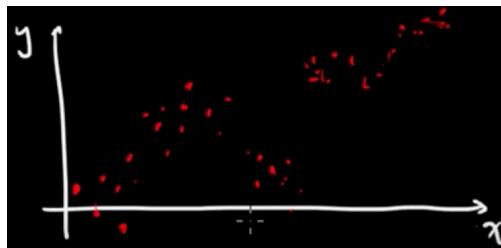
(2,3)은 0이 2개, 1이 3개 이므로 "1"로 분류

1. Decision Tree (CART)

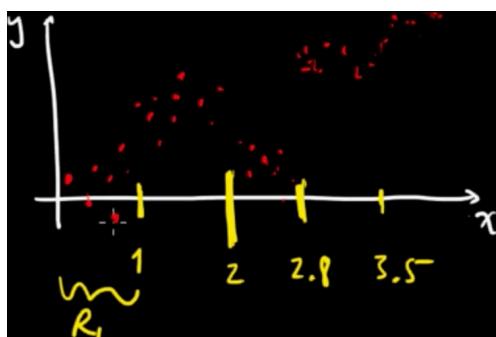
Regression Tree에서 에러를 측정하는 방법

Regression Tree 예시

- 아래와 같이 임의 데이터를 생성

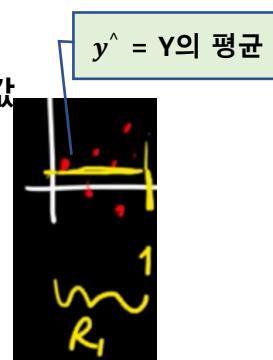


- 초기 학습시 데이터를 임의로 분류 (아래 그림은 사람이 직관적으로 선택)
 - 변수(X,Y)에 포함된 모든 값을
 - 임의로 선택



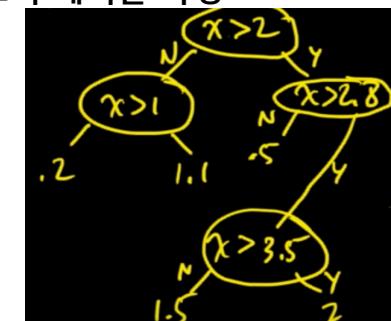
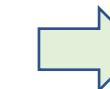
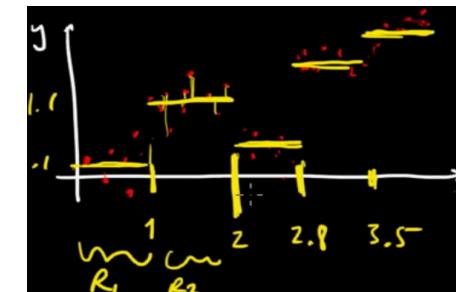
- 분류한 노드의 에러를 계산 (MSE 활용)
 - 그럼 예측값(\hat{y})는 어떻게 계산하나?
 - 분류된 데이터(R_1)의 평균값 → 여기서는 x와 y의 평균값
- 결국 분류된 데이터의 평균값과 실제값의 차이를 최소화!
 - 이는 학습을 통해 최적의 영역(R)을 찾는 것이다.

$$\sum_{i: x_i < 1} (y_i - \hat{y})^2$$

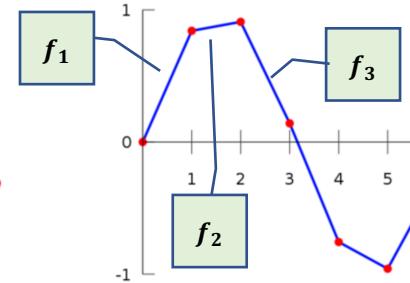
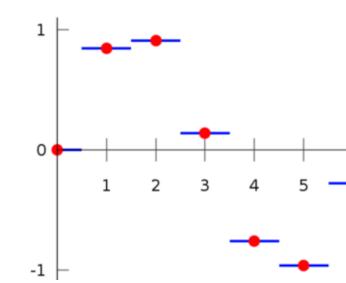


최종 분류된 트리의 결과 및 에러측정 (Score)

- 모든 학습 데이터 속성의 값을 기준으로 분류를 수행
- 속성을 선택하여 분류할 때마다 분류된 노드의 에러를 측정



- 만약 새로운 (x) 입력되면 트리는 어떻게 분류할까?
 - Piecewise constant (사이채움, 보간법)을 활용
 - 예를 들어, 신규 변수(x)에 대응하는 함수를 찾을 때,
 - x 와 가장 가까운 x_i 를 찾고,
 - x_i 에 대응하는 함수(f_i)를 x 의 함수로 사용



1. Decision Tree (CART)

Growing a regression tree (트리를 어떻게 성장시킬 것인가?)

Greedy 알고리즘을 이용한 최적의 트리 생성

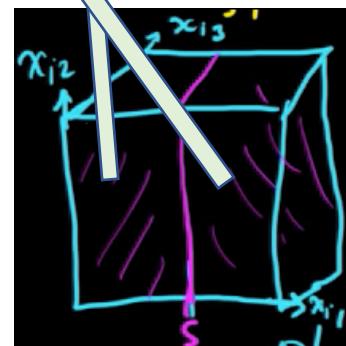
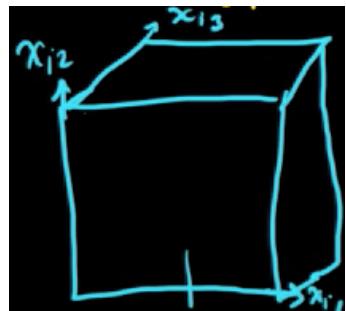
- 학습 에러를 최소화하는 최적의 분류 선택
 - 속성 : $x_i = (x_{i1}, x_{i2}, \dots, x_{id})$ 이 있는 경우, 학습에러 공식이다
 - S : 분류 기준

$$\min \sum_{j=1: x_{ij} > S} (y_j - \bar{y}_j)^2 + \min \sum_{j: x_{ij} \leq S} (y_j - \bar{y}_j)^2$$

속성값이 S보다 큰 경우,
분류된 값과 예측값(속성
의 평균)의 차이

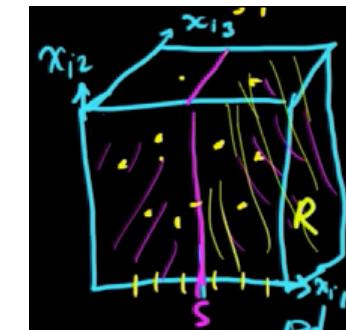
속성값이 S보다 큰 경우,
분류된 값과 예측값(속성
의 평균)의 차이

- 아래 데이터를 분류하는 예시를 보면
 - x_{i1} , 속성의 S값을 기준으로 데이터를 분류한다.

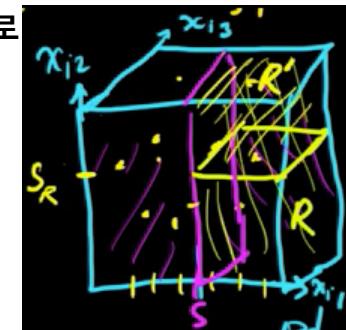
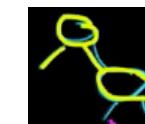


트리는 어떻게 성장하는가?

- 첫 번째 분류시
 - 분류된 데이터의 학습오류 최소화 되는
 - J (속성의 순서)와 S(분류 기준)을 선택
 - 왼쪽 아래의 그림에서는 J=1,
 - S는 값이 정의되지 않음



- 두 번째 분류시
 - 첫 번째 분류된 노드를 다시 동일한 기준으로
 - 학습오류가 최소화 되도록 J, S를 선택
- 모든 데이터가 분류될 때 까지 반복
 - 과접합 가능



언제 트리의 성장을 멈추는가?

- 분류된 leaf node에 데이터가 1개만 있는 경우
- 지정한 최소 개수보다 적은 경우

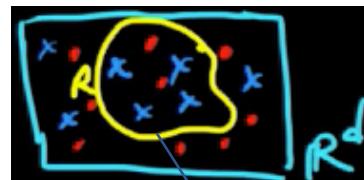
1. Decision Tree (CART)

Growing a classification tree (분류 트리를 어떻게 성장시킬 것인가?)

Classification Tree의 분류 결과 측정

- 데이터를 분류하고, 분류결과를 판단하는 방법(에러를 측정)
- E_R (불순도, Error)
 - R 영역으로 분류된 포인트(x_i) 중, 잘못 분류된 비율
 - 이때 잘못 분류된 기준은 다수결을 기준으로 선정
 - 여기서는 2/6 (아래 공식 참고) \rightarrow 0로 분류된 항목이 2개
- N_R : 전체 데이터 개수 (분류된 영역에 있는)
- x_i : 분류된 영역에 포함된 데이터
- E_R 공식

$$\min_y \frac{1}{N_R} \sum_{i: x_i \in R} I(y_i \neq y)$$



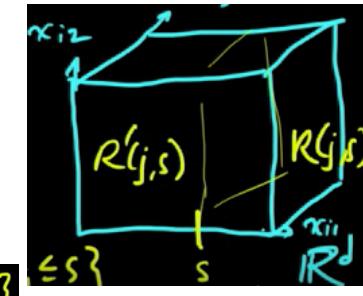
다수결로 분류한 결과와
실제 값이 다른 개수 계산

R 영역의 다수결 결과는 X
X:4, O:2로 X로 분류

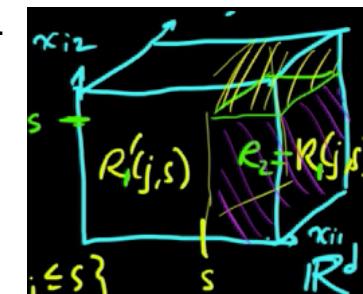
트리는 어떻게 성장하는가?

- 첫 번째 분류시
 - 분류된 데이터의 학습오류 최소화 되는
 - j (속성)와 S (분류 기준)을 선택
 - $N_{R(j,S)} E_{R(j,S)} + N_{R'(j,S)} E_{R'(j,S)}$ 최소화
 - 여기서 $j=1$, S 는 특정 값

$$R'_{(j, S)} = \{x_i : x_{ij} \leq S\} \quad R_{(j, S)} = \{x_i : x_{ij} > S\}$$



- 두 번째 분류시
 - 첫 번째 분류된 노드를 다시 동일한 기준으로
 - 학습오류가 최소화 되도록 j, S 를 선택
- 모든 데이터가 분류될 때 까지 반복
 - 과접합 가능



언제 트리의 성장을 멈추는가?

- 분류된 leaf node에 데이터가 1개만 있는 경우
- \rightarrow 지정한 최소 개수보다 적은 경우

1. Decision Tree (CART, 회귀트리 모델)

연속적이고, 이산적인 속성으로 트리 구축하기

트리 생성

```
def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1,4)):#assume dataSet is
    feat, val = chooseBestSplit(dataSet, leafType, errType, ops)#choose the best split
    if feat == None: return val #if the splitting hit a stop condition return val
    retTree = {}
    retTree['spInd'] = feat
    retTree['spVal'] = val
    lSet, rSet = binSplitDataSet(dataSet, feat, val)
    retTree['left'] = createTree(lSet, leafType, errType, ops)
    retTree['right'] = createTree(rSet, leafType, errType, ops)
    return retTree
```

```
def binSplitDataSet(dataSet, feature, value):
    mat0 = dataSet[nonzero(dataSet[:,feature] > value)[0],:]
    mat1 = dataSet[nonzero(dataSet[:,feature] <= value)[0],:]
    return mat0,mat1
```

```
>>> tstMat
matrix([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> print nonzero(tstMat[:,1] > 0.5)[0]
[1]
>>> tstMat[nonzero(tstMat[:,1] > 0.5)[0],:]
matrix([[ 0.,  1.,  0.]])
>>> tstMat[1,:]
matrix([[ 0.,  1.,  0.]])
>>> tstMat[nonzero(tstMat[:,1] > 0.5)[0],:][0]
matrix([[ 0.,  1.,  0.]])
```

- 코드 오류 해결
index 0 is out of bounds
for axis 0 with size 0

트리 생성 방식

- 분할에 가장 적합한 feature 선택
 - 더 이상 분할할 feature가 없으면, 단말노드
 - 분할 가능하면, 양쪽에 트리 생성 → 재귀호출

수치형 데이터를 어떻게 분할할까?

- MSE(Mean Square Error)를 이용한 데이터 복잡성 측정
- 해당 feature 값들의 평균을 계산하고,
- 평균 제곱오류의 합으로 복잡성 측정
- 값이 클수록 복잡함 (분류기준에 부적합)

- Nonzero 리턴값 = 행 array, 열 array
- Array[1], array[0]의 의미는 행1, 열0 번째에 0이 아닌 값이 있다는 것
- $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

1. Decision Tree (CART, 회귀트리 모델)

오차범위와 최소 데이터 개수를 이용하여 최적의 분류 속성(field) 선택

최적의 속성 선택 과정

```
def regLeaf(dataSet):#returns the value used for each leaf
    return mean(dataSet[:, -1])
def regErr(dataSet):
    return var(dataSet[:, -1]) * shape(dataSet)[0]
def chooseBestSplit(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
    toLS = ops[0]; toLN = ops[1]
    #if all the target variables are the same value: quit and return value
    if len(set(dataSet[:, -1].T.tolist()[0])) == 1: #exit cond 1
        return None, leafType(dataSet)
    m, n = shape(dataSet)
    #the choice of the best feature is driven by Reduction in RSS error from mean
    S = errType(dataSet)
    bestS = inf; bestIndex = 0; bestValue = 0
    for featIndex in range(n-1):
        for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
            mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
            if (shape(mat0)[0] < toLN) or (shape(mat1)[0] < toLN): continue
            newS = errType(mat0) + errType(mat1)
            if newS < bestS:
                bestIndex = featIndex
                bestValue = splitVal
                bestS = newS
    #if the decrease (S-bestS) is less than a threshold don't do the split
    if (S - bestS) < toLS:
        return None, leafType(dataSet) #exit cond 2
    mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
    if (shape(mat0)[0] < toLN) or (shape(mat1)[0] < toLN): #exit cond 3
        return None, leafType(dataSet)
    return bestIndex, bestValue#returns the best feature to split on
    #and the value used for that split
```

- 코드 오류 해결
unhashable type: 'matrix'

1

3

6

입력 파라미터

- leafType (regLeaf) : 데이터의 평균 계산 함수
- errType (regErr) : 데이터의 MSE 계산 함수
- 오차범위 (tolS)
 - 오류가 오차범위 이내로 개선되면,
 - 데이터를 분할하지 않음. (과적합 방지)
- 데이터 최소개수 (tolN)
 - 분할할 데이터가 너무 적지 않도록 설정 (과적합 방지)

함수 설명

- 데이터 값이 모두 같은지 확인 (같으면 트리가 없음, None 리턴)
- 현재 데이터의 오차(MSE) 계산 (S)
- 속성 개수만큼 순환
 - 중복제거한 속성 값 만큼 순환
 - 데이터 이진분할 (binSplitDataSet)
 - 분할한 좌/우 데이터 집합이 최소 데이터 개수보다 적으면, 해당 속성값은 선택하지 않는다.
- 오차(MSE)가 가장 적은 속성과 속성값 선택 (bestIndex, splitVal)
- 기존 오차(S)와 선택속성의 오차(bestS)가 허용범위 이내라면 트리생성 취소
- 선택된 속성과 속성값으로 데이터를 분할하고, 최소개수가 아닌지 확인

1. Decision Tree (CART, 모델트리)

회기에 의하여 트리를 선택하는 모델 (분할된 데이터 각각에 대한 선형회귀 모델)

어떻게 데이터를 분할 하는가?

- 분할된 데이터 집합의 오류를 측정
[어떻게 ?]
- 분할된 데이터에 맞는 선형모델 생성
- 모델의 예측값과 정답을 비교하여 오류 계산
- **modelLeaf** : 데이터의 모델 가중치 파라미터 리턴
- **modelErr** : 모델의 예측값과 정답간의 오차를 리턴

```
def linearSolve(dataSet):    #helper function used in two places
m,n = shape(dataSet)
X = mat(ones((m,n))); Y = mat(ones((m,1)))#create a copy of data with 1 in 0th position
X[:,1:n] = dataSet[:,0:n-1]; Y = dataSet[:, -1]#and strip out Y
xTx = X.T*X
if linalg.det(xTx) == 0.0:
    raise NameError('This matrix is singular, cannot do inverse,\ntry increasing the second value of ops')
ws = xTx.I * (X.T * Y)
return ws,X,Y

def modelLeaf(dataSet):#create linear model and return coeficients
ws,X,Y = linearSolve(dataSet)
return ws

def modelErr(dataSet):
ws,X,Y = linearSolve(dataSet)
yHat = X * ws
return sum(power(Y - yHat,2))
```

선형 모델 생성 및 오류 계산

```
def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1,4)):#assume dataSet is
feat, val = chooseBestSplit(dataSet, leafType, errType, ops)#choose the best split
if feat == None: return val #if the splitting hit a stop condition return val
retTree = {}
retTree['spInd'] = feat
retTree['spVal'] = val
lSet, rSet = binSplitDataSet(dataSet, feat, val)
retTree['left'] = createTree(lSet, leafType, errType, ops)
retTree['right'] = createTree(rSet, leafType, errType, ops)
return retTree
```

- 기존 **createTree**함수를 그대로 활용
- 전달하는 파라미터(**leafTyp**, **errTyp**)만
- (**modelLeaf**, **modelErr**)로 변경하여 호출

• 회귀 트리

```
trainMat = mat(regTrees.loadDataSet('bikeSpeedVsIq_train.txt'))
testMat = mat(regTrees.loadDataSet('bikeSpeedVsIq_test.txt'))
myTree  = regTrees.createTree(trainMat, ops=(1, 20))
myTree  = regTrees.createTree(trainMat, regTrees.modelLeaf, regTrees.modelErr, (1,20))
```

• 모델 트리

- **chooseBestSplit**에서 분할 오류값을 측정

END