

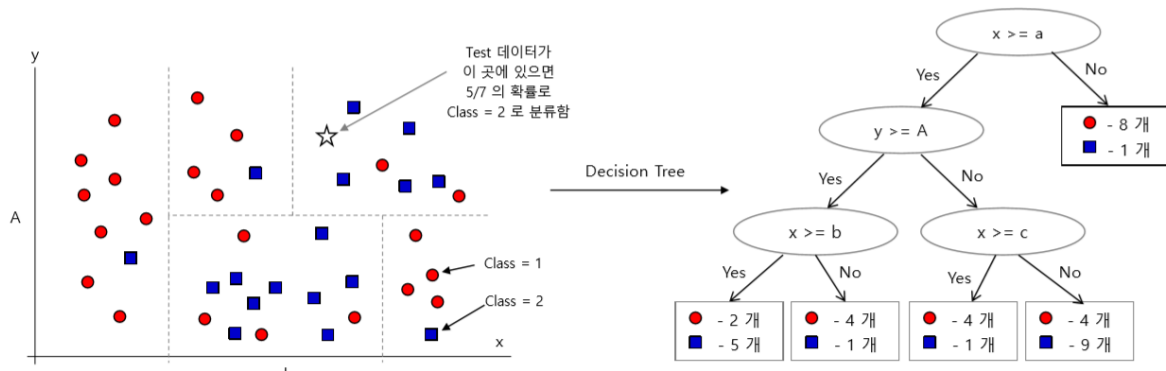
출처가 명시되지 않은 모든 자료(이미지 등)는 조성현 강사님 블로그 및 강의 자료 기반.

<< 머신러닝 - 분류 >>

[Decision Tree]

특정 기준에 따라 데이터에 구분선을 그어 나가며 분류하는 모델이다. 여러 개의 데이터를 가장 잘 구분할 수 있는 질문을 기준으로, 구분선을 긋는다.

이렇게 여러 가지 규칙을 순차적으로 적용하면서 데이터를 분할하다 보면, 다음과 같이 데이터를 나누는 기준을 트리 형태로 나타낼 수 있다.



- 구분선(decision boundary): 의사결정선. 데이터를 분류하기 위해 긋는 선.
- 분기: 구분선을 기준으로 의사결정나무가 나누어지는 것.
- 리프 노드: 마지막 노드.

새로운 데이터가 들어올 때마다 의사결정선을 긋는다. 계속해서 의사결정선을 그어 나가면서, 가장 잘 분류될 때까지 과정을 반복한다. 입력되는 데이터가 3차원 이상의 다차원 구조일 경우, 의사결정선으로 직선 대신 (초) 평면을 사용한다. 차원이 높아져도 구분선(혹은 면)을 그어 구분하면 되기 때문에, 직관적으로 이해하기 쉬운 분류 기법으로 자주 사용한다.

여기서는 분류 기법으로 설명하지만, 회귀 기법에도 사용할 수 있다. 아래에서 설명할 분할 알고리즘에서 선택하는 척도로 F 통계량, 분산의 감소량 등을 설정하면 된다.

1. 분할 알고리즘

데이터를 분할하기 위해 구분선(혹은 면)을 설정하는 것이 핵심이 되는 알고리즘인 만큼, 데이터를 가장 잘 분류하기 위해 **최적의 트리 분할 기준**을 고르는 것이 중요하다.

불순 척도

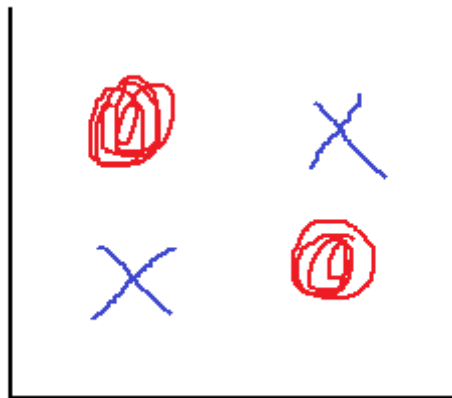
데이터 분할 시, 분할 전보다 후에 불순한 정도가 줄어드는 것이 좋다고 가정한다. 여기서의 '불순'이란, 데이터가 섞인 정도이다. 이질적인 데이터가 많을수록, 해당 노드에서 불순도가 높다고 본다. 즉, 부모 노드에서 자식 노드로 갈 때 불순도가 덜해지도록, 혹은 최대한 많이 감소하도록 해야 한다.

이러한 불순도를 측정하는 척도로 아래와 같이 **엔트로피 지수**와 **지니 지수**의 두 가지가 사용된다.

$$H(t) = - \sum_{i=1}^c p(i|t) \log_2 p(i|t) \quad \leftarrow \text{엔트로피}$$
$$Gini(t) = 1 - \sum_{i=1}^c p(i|t)^2 \quad \leftarrow \text{지니 지수}$$

1부터 c까지의 i는 클래스를 나타낸다

다음의 경우(동그라미 2개, 엑스 2개) 엔트로피와 지니 계수를 계산해 보자.



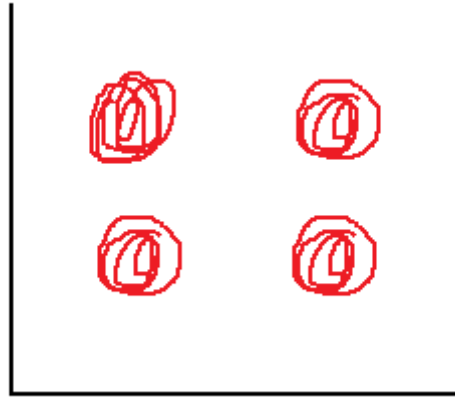
- 엔트로피

$$G_E = -\left(\frac{1}{2} \times \log_2 \frac{1}{2} + \frac{1}{2} \times \log_2 \frac{1}{2}\right)$$

- 지니

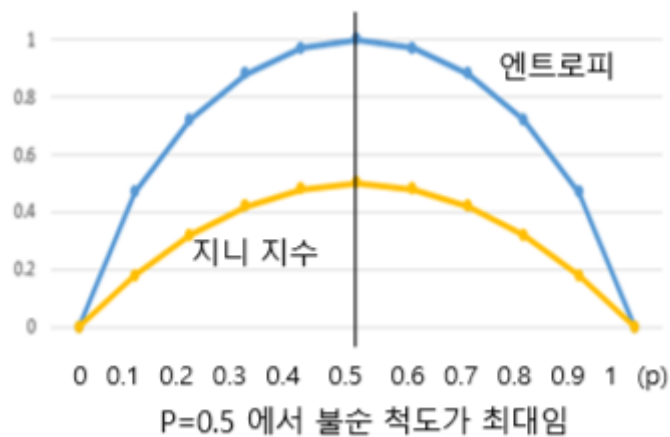
$$G_G = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2$$

계산해 보면 엔트로피 지수는 1, 지니 지수는 0.5가 됨을 알 수 있다. 각각의 수치는 데이터가 가장 불순할 때 나올 수 있는 수치들이다. 반대로 다음과 같이 데이터가 모두 순수한 경우는, 동일한 방법으로 계산하면 엔트로피 지수와 지니 지수가 모두 0이 된다.



이를 바탕으로 지니지수와 엔트로피 지수로 측정한 불순 척도의 분포를 그래프로 나타내면 다음과 같다.

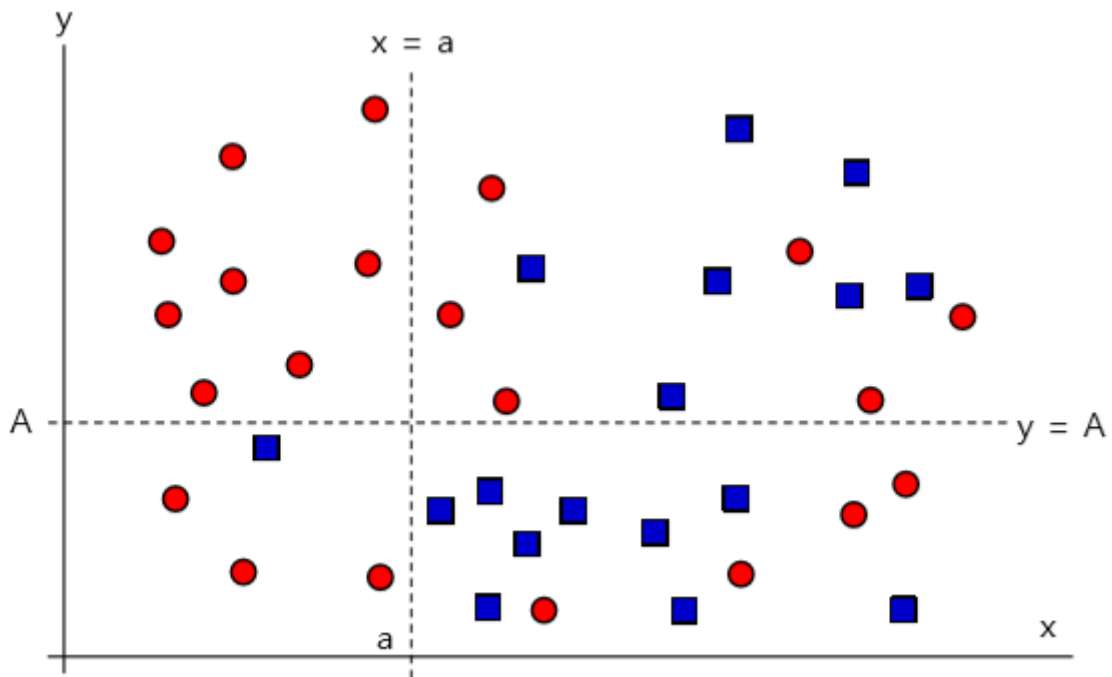
둘 모두 데이터가 절반씩 섞여 있을 때 불순 척도가 최대가 됨을 알 수 있다. 지니지수의 경우 0에서 0.5 사이의 값이, 엔트로피 지수의 경우 0에서 1사이의 값을 갖는다.



정보 획득

분할 시에는 어떠한 기준을 선택하든 불순 척도가 감소하도록 선택하는 것이 좋다고 했다. 불순도가 감소하는 것을 **정보 획득**이라고 한다. 따라서 트리 분할 기준을 한 마디로 다시 표현한다면, **정보 획득이 최대가 되도록 하는 것이 좋다.**

분할 선택의 과정을 다음의 그림을 예로 들어 설명하면 다음과 같다. (지니 지수를 기준으로 설명한다.)



초기 분할 선을 $x = a$ 로 선택하거나(좌우 분할), $y = A$ 로 선택한다(상하 분할)고 하자.

좌우 분할로 선택하는 경우, 왼쪽 영역의 지니 지수는 0.153, 오른쪽 영역의 지니 지수는 0.473이 된다. 각 경우의 지니 지수를 가중평균하면, 좌우의 선을 초기 분할선으로 선택하는 경우 지니 지수는 0.372가 된다. 완전히 불순할 때의 지니 지수가 0.5임을 감안한다면, 분할을 통해 0.128($=0.5-0.372$)의 정보량을 획득했다고 볼 수 있다.

같은 방식으로 상하 분할선을 선택한다고 하면, 위쪽 영역의 지니지수는 0.444, 아래쪽 영역의 지니 지수는 0.484가 된다. 가중평균한 값은 0.462이므로, 분할을 통해 0.038의 정보량을 획득했다.

정보 획득량을 계산할 때 가중평균을 취하는 이유는, 데이터의 수에 영향을 받지 않도록 하기 위함이다. 극단적으로 데이터가 1개만 존재하도록 분할선을 긋는다면, 가장 순수한 데이터 영역이 만들어질 것이다. 이처럼 데이터의 수를 적게 해놓고 순도가 높다고 하는 것을 방지하기 위해 가중평균의 과정이 필요하다.

이렇게 초기분할을 하는 경우를 비교했을 때, 불순도가 더 낮아지고, 정보량 획득이 많아지는 첫 번째 방법으로 분할선을 선택하는 것이 좋다.

이렇게 **정보 획득량이 더 많아지도록** 계속해서 트리를 분기해 나가는 것이 의사결정나무의 원리이다. (물론 무수히 많은 분기의 과정을 거칠 수 없기 때문에, 내부적으로 `quantile` 등의 알고리즘을 사용하여 분기한다. 지금 강의에서 그 단계까지 설명하지는 않는다.)

당연한 얘기지만, 분기 시 정보 획득량이 0보다 작으면 분할하면 안 된다. 분할해서 이득이 생겨야 분할하는 게 좋기 때문이다.

2. 특징

- 분류나 예측의 근거를 알 수 있으므로, 결정 과정에 대한 이해가 쉽다.
- 데이터의 차원이 높아져도(feature가 많아지더라도) 분류에 중요한 feature들을 제외할 수 있으므로, feature 선정 단계에서 크게 신경쓸 필요가 없다.(feature engineering에 큰 힘을 쏟지 않아도 된다는 의미인 듯하다.)
- 중요도 분석이 가능하다.

실습 1. breast cancer data: 모델 학습

다른 사이킷런 모델과 마찬가지로 다음의 과정을 거쳐 모델을 학습한다.

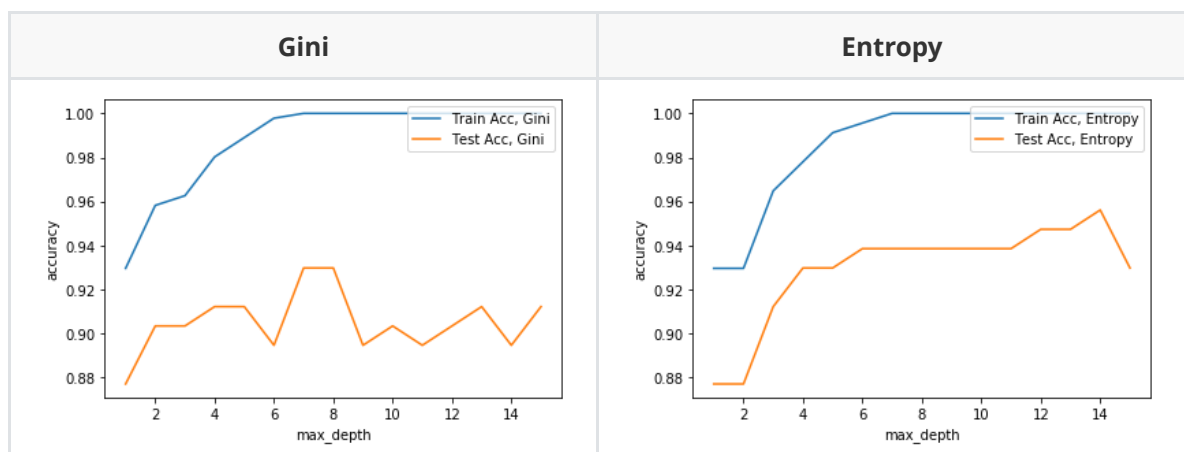
- `model.build`
- `model.fit` : 트리를 생성한다. 지니 지수나 엔트로피 지수 등 불순 척도를 계산하고, 학습 데이터를 이용해서 학습 데이터로 트리를 만든다.
- `model.predict` : 생성한 트리를 바탕으로 새로운 데이터가 어느 영역에 속할지(회귀의 경우에는 어떤 값을 가질지) 예측한다.

모델의 파라미터 중 중요한 것으로는 정보량 획득 계산의 기준을 무엇으로 할 것인지의 `criterion` 과, 트리를 어느 단계까지 생성해 나갈 것인지의 `max_depth` 가 있다.

`max_depth`의 경우, default 값은 `None` 이다. 마지막 노드에 데이터가 1개 남을 때까지 트리를 분기해 나간다. 이 경우, 당연히 마지막 노드에 데이터가 1개 남기 때문에, pure해진다.

3. 문제: 과적합

breast cancer 데이터 실습 결과를 보자.



train set에 대한 정확도가 어느 순간 이후 100%가 되는 것을 볼 수 있다. 자신의 데이터로 트리를 구성한 후, 그 데이터를 대입하니 잘 맞출 수밖에 없다.

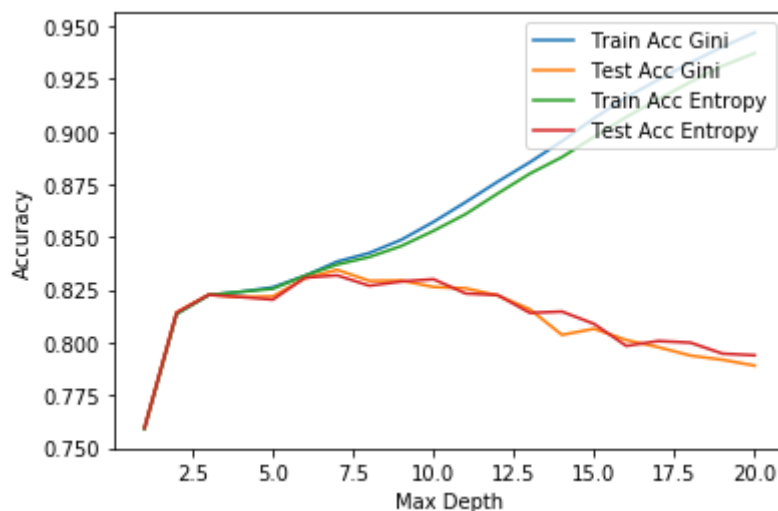
트리의 깊이가 깊어질수록, 의사결정나무 알고리즘은 맨 끝의 leaf node가 1개 남을 때까지 트리를 만들어 나간다. 깊이가 무한해질수록 순도가 높아지고, 학습 데이터를 정확히 맞출 수밖에 없는 구조라는 말이다. 그래서 max_depth를 늘리며 학습을 진행해 나갈수록 train set에 대한 정확도는 높아질 수밖에 없다.

그렇다면 이렇게 train accuracy가 높아지는 것이 좋은 상황일까? 그렇지 않다. **과적합**된 상황이다.

덱스가 깊어질수록 학습 데이터는 잘 설명하는데, 새로운 데이터는 제대로 맞추지 못한다는 의미다. 알고리즘이 학습 데이터를 달달 외워 버리는 상황이다. **적절한 덱스가 어디인지** 결정하는 일이 매우 중요하다.

실습 2. income dataset

income 데이터로 진행한 실습을 보더라도, 덱스가 깊어질수록 train accuracy는 100%에 가까워지지만, test accuracy는 7 부근에서 꺾이는 것을 볼 수 있다.



pd.Categorical

Pandas가 알아서 범주형 변수를 숫자 카테고리 코드로 바꿔 준다.

```
>>> pd.Categorical(income['workclass']) # 문자 형태의 범주형 변수를 Pandas는 어떻게 범주로 바라보는가.
```

```
Out[6]:
```

```
[State-gov, Self-emp-not-inc, Private, Private, Private, ..., Private, Private, Private, Private, Self-emp-inc]
```

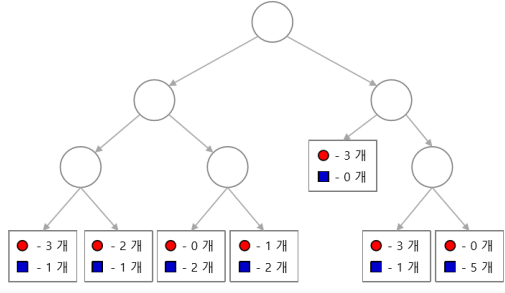
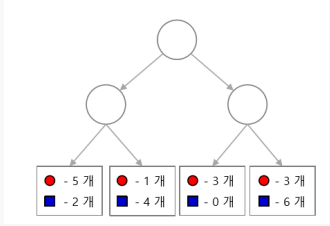
```
Length: 32561
```

```
Categories (9, object): [?, Federal-gov, Local-gov, Never-worked, ..., Self-emp-inc, Self-emp-not-inc, State-gov, Without-pay]
```

```
>>> pd.Categorical(income['workclass']).codes # 변경된 문자 형태 범주의 숫자 코드
```

```
Out[7]: array([7, 6, 4, ..., 4, 4, 5], dtype=int8)
```

교재의 간단한 예시를 살펴 보자.

		
트리 depth	3	2
리프 노드 개수	7	4
잘못 분류	4	6
에러율 (잘못 분류 된 개수/ 전체 데이 터 개수)	4/24	6/24

오른쪽 트리가 에러율이 더 높아서 왼쪽 트리가 더 좋은 트리라고 생각할 수 있다. depth가 깊어질수록 에러가 감소하는 것은 당연하다. 학습 데이터에 대해서는 그럴 수밖에 없다. *문제*는 시험 데이터이다. 학습의 이유는 새로운 데이터를 알아맞추기 위함이다. 그런데 덩스가 깊어질수록 시험 데이터에 대한 정확도가 어느 정도 증가하다가 낮아진다. 학습 데이터의 accuracy는 지속적으로 올라가지만, 시험 데이터의 accuracy는 증가하다가 감소한다.

결국, 에러가 적고, 정확도가 가장 높은 지점의 depth를 찾는 것이 중요하다!

4. 정규화

위에서 말한 것처럼 의사결정나무 알고리즘에서는 트리가 복잡해질수록 훈련 데이터에 과적합되는 문제가 발생한다. 그 외에도, 트리가 복잡해질수록, 아래 쪽 노드에 포함되는 데이터가 적어지며 통계적으로 의미 있는 결정을 내리기 어려워진다.

따라서 의사결정나무 알고리즘을 적용해 모델을 훈련할 때는 모델이 너무 복잡해지지 않도록 하는 것이 매우 중요하다. 이것이 모델 학습 및 구성의 큰 축을 이룬다.

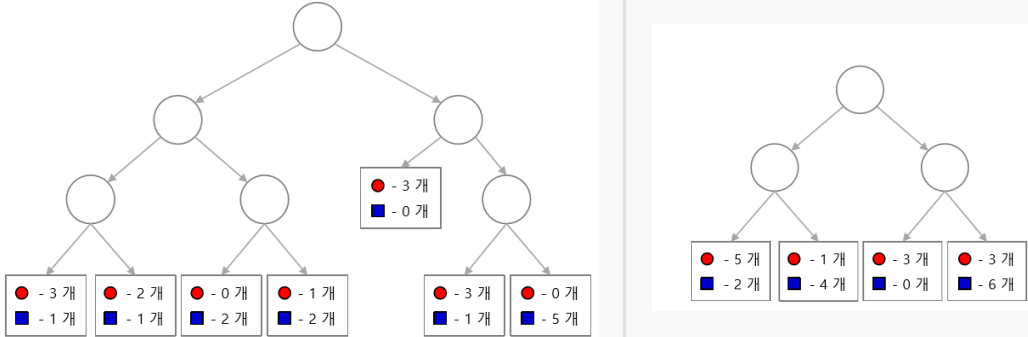
다시 위에서의 교재 예시로 돌아가 보자. 위의 그림에 있는 데이터는 모두 다 학습 데이터이다. 결국 여기서 측정된 **잘못 분류된 개수**, 즉, **에러율**은 학습 데이터에 대한 에러율이다. depth가 증가할 때 학습 데이터에 대한 에러율이 감소하는 것은 자연스러운 이야기이다. 하지만, 시험 데이터가 어떤지는 알 수 없다. 학습 데이터의 에러율만 가지고 좋다, 나쁘다 이야기할 수 없다.

이제 에러율을 수정한다. depth가 깊어질수록, 학습 데이터를 잘 맞추는 것은 당연하기 때문에, depth가 깊어질수록 벌점을 준다. 나무를 계속해서 분기시키지 않고도 잘 분류해낼 수록 더 좋은 트리라고 판단하겠다는 의미이다. 기존의 에러율 공식에 아래와 같이 penalty term으로 알파를 추가해 보자. 덩스가 클수록, 점수가 낮아지게 식을 표현한다.

$$e_{\alpha} = \frac{e + N \times \alpha}{D}$$

- 트리가 깊을수록 리프 노드의 수가 기하급수적으로 증가한다.
- 따라서 전체 데이터 개수 대비 리프 노드의 개수(N)를 벌점의 기준으로 삼는다.
- 이 때, 벌점을 얼마나 매길 것인지를 알파로 조정한다.

예컨대, 알파를 1로 하여 두 트리의 에러율을 다시 계산하면 다음과 같다.

	
수정 에 러 율	<div>11/24</div> <div>10/24</div>

오른쪽 트리에서 수정 에러율이 더 낮다. 학습 데이터는 왼쪽 트리가 더 잘 설명하지만, 시험 데이터에 대해서는 오른쪽 트리가 더 잘 설명할 여지가 있다. 왼쪽 트리의 과적합 상황을 오른쪽 트리에서처럼 적절하게 조정했다고 보면 된다.

알파가 0이면, 에러율 공식은 N/D 가 되기 때문에, 당연히 D를 최대화하려고 한다. 기존의 에러율 공식을 따른다면, 결정트리 알고리즘은 당연히 나무를 최대한으로 분기시키려 할 것이다.

이렇게 과적합을 조정하는 과정을 **정규화(regularization)**라고 한다. 규제항을 통해 일반화 성능을 향상시키는 방법이다. 페널티항을 벌점으로 추가해서 일반화 성능을 향상시키는 것이다. 이를 통해 과적합을 방지한다.

에러율 공식에서 알파를 규제항(regularization constant)라고 한다. 규제항의 크기를 조정함으로써 학습 뿐만 아니라 새로운 데이터까지 잘 설명하고자 한다. 위에서는 임의로 1.0을 적용했다. 분석자가 임의로 적용한 항이다. 결정나무 알고리즘을 적용해 모델을 설계하고자 할 경우, 분석자의 경험이나 노하우를 통해 reg. term을 조정하는 것이 매우 중요하다.

5. 가지치기

실제 결정트리의 알고리즘 설계에서 정규화(과적합 조정)는 가지치기를 통해 이루어진다. depth의 수나 알파의 크기를 통해 가지치기를 진행할 수 있다.

가지치기란, 트리가 너무 복잡해지지 않도록 단순화하는 과정이다. 트리 모델이 학습 데이터에 너무 적합하게 성장하지 않도록, *일반화* 특성을 향상시키는 과정이다.

검증데이터에 대한 오분류율이 증가하는 시점에서 적절히 가지치기를 수행해줘야 합니다.

마치 나뭇가지를 잘라내는 것과 같다는 의미에서 이러한 용어가 붙었습니다. 매우 직관적이죠. 다만 가지치기는 데이터를 버리는 개념이 아니고 분기를 합치는(merge) 개념으로 이해해야 합니다.

_출처: [ratsgo's blog](#)

훈련 데이터를 학습용과 검증용으로 나누어 일반화 특성이 좋아지는 지점에서 트리 성장을 멈추거나, 사전에 데전문가에게 의뢰하여 타당성이 없는 규칙을 배제하도록 할 수 있다.

종류

1) 사전 가지치기(pre-pruning)와, 2) 사후 가지치기(post-pruning)의 두 가지 방법이 있다.

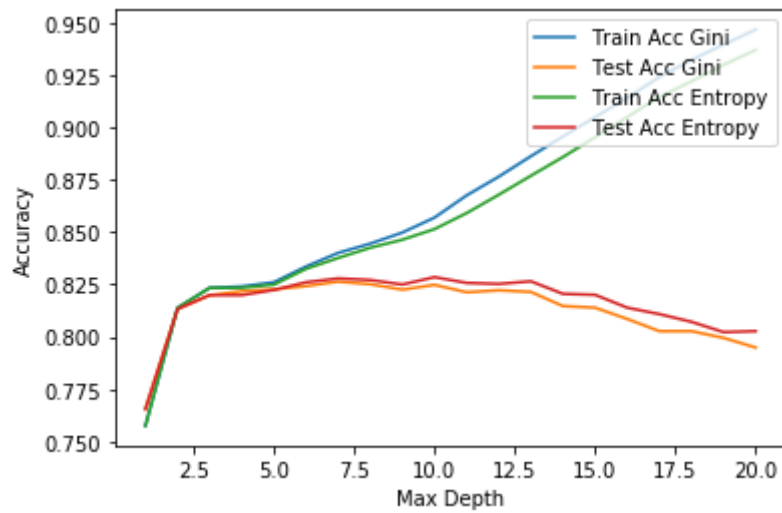
사전 가지치기: depth 조정

조기 정지규칙(stopping rule)에 의해 트리 성장을 멈추는 방법이다. 정지 규칙으로 사전에 마지막 트리의 깊이(depth)를 지정하거나, 마지막 노드의 데이터 수가 사전에 정해 놓은 임계치 이하로 떨어지지 않도록 지정할 수 있다.

실습 3. income dataset: pre-pruning

위에서 진행한 income dataset에 대한 Decision Tree 구성에 사전 가지치기를 적용한다.

depth를 1부터 시작해 점차 깊어지도록 한다. 간단한 트리에서 복잡한 트리로 가면서 test set accuracy가 어떻게 변화하는지 살펴 보고, 적절한 단계에서 가지치기를 중단한다.



사후 가지치기: 정규화항 조정

초기에는 트리를 최대 크기로 만들고, 이후 트리를 위쪽 방향으로 다듬어 가는 trimming 단계를 거치는 방법이다. 최초에 형성되는 트리가 최대 복잡도를 갖는다.

최대한 복잡하게 만든 다음, 필요 없는 가지를 잘라 내면서 간소화한다. 이런 측면에서 본다면, 진정한 의미에서의 가지치기라고도 할 수 있다.

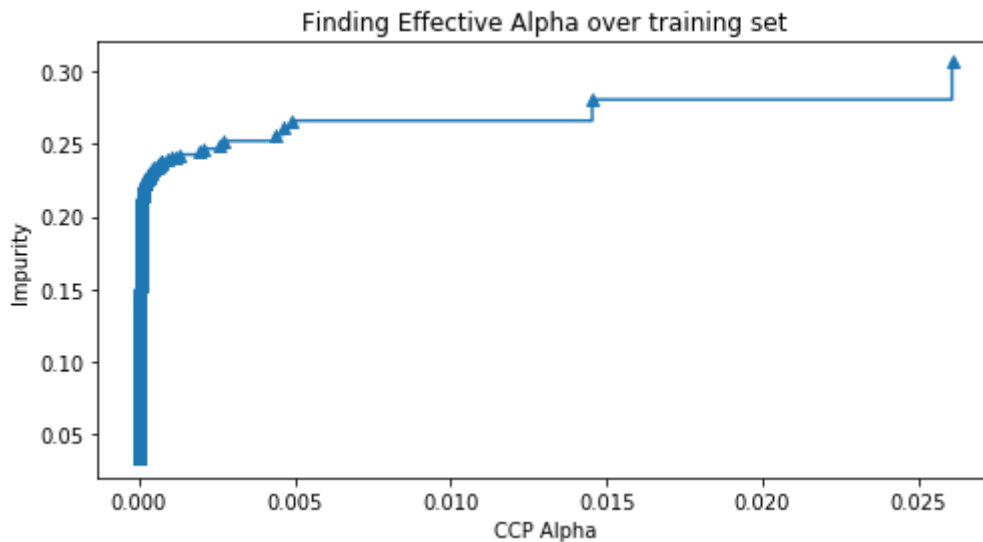
알파를 이용해 최적의 트리를 만드는 방법이다.

실습 4. income dataset: post-pruning

사이킷런에서 사후 가지치기는 `DecisionTreeClassifier` 함수에서 `ccp_alpha` 변수를 조정함으로써 이루어진다. `cost_complexity_pruning_path` 함수를 통해 정규화항(`ccp_alpha`)의 범위를 설정하고, 각각의 정규화항 크기에서의 불순도를 나타낼 수 있다.

```
from sklearn.tree import DecisionTreeClassifier

path = DecisionTreeClassifier().cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

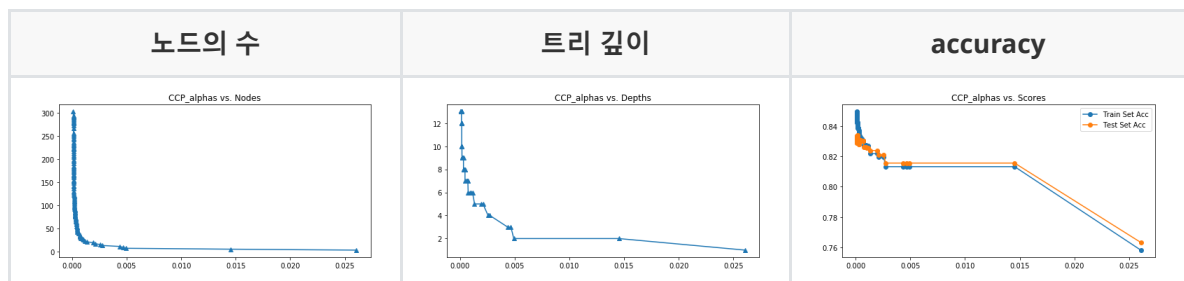


정규화항의 크기 변화에 따른 트리의 불순도 변화

알파의 크기가 증가할수록, 즉, 트리 깊이에 대한 가중치를 크게 둘수록 불순도가 커진다. 트리가 그만큼 분기되지 않기 때문에, 불순도가 증가하는 것이다.

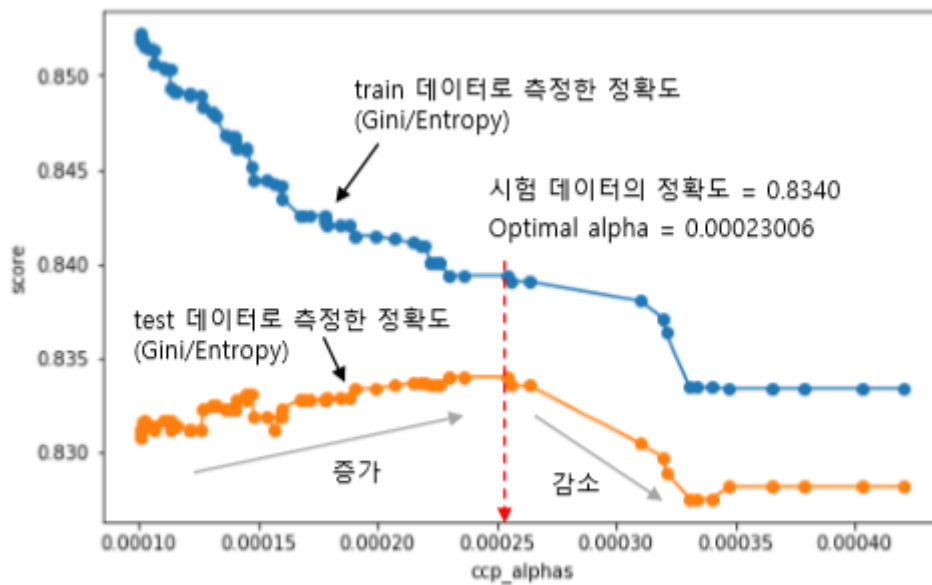
정규화항의 크기가 너무 작은 경우, 불순도의 분포가 매우 조밀하게 나타난다. 또한 마지막 트리에서는 depth가 0이 되므로, 트리가 분기되지 않게 된다.

시각화를 위해 크기가 너무 작은 `ccp_alpha`를 제외하고, 마지막 트리를 제외한 뒤 트리 노드의 수, 트리 깊이, test set에 대한 accuracy를 나타내면 다음과 같다.



alpha가 증가할 수록 노드의 수와 깊이가 감소한다. underfitting이다. depth에 대한 penalty를 크게 두기 때문에, 그만큼 트리의 구성이 단순해지는 것이다.

내 코드에서는 accuracy 변화가 정확하게 드러나지 않아 강사님의 자료를 참고해 accuracy 변화를 확인하면 다음과 같다.



alpha가 증가할수록 overfitting의 문제가 완화되기 때문에 train accuracy가 계속해서 감소한다. test set accuracy는 증가하다가 어느 순간 감소하는 모습을 보인다. 규제항의 크기가 너무 커도 트리의 분기 기준이 적절히 설정되지 않는 것을 알 수 있다.

두 그래프의 분포를 보고 적절한 alpha의 크기를 찾아내도록 하자.

6. 특성 중요도

의사결정나무 알고리즘은 하위 트리를 형성할 때 불순도가 최소가 되도록, 다시 말하면 정보량 획득이 최대가 되도록 feature들의 분기를 결정한다.

그렇기 때문에 각 분기 과정에서 각 feature가 얼마나 정보 불순도를 감소시켰는지 알아낼 수 있다. 각 분기 과정에서 feature가 분리됨으로써 정보를 얼마나 획득했는지 평균을 내고, 그 feature가 전체 트리를 만드는 데 얼마나 기여했는지 파악할 수 있다는 의미이다.

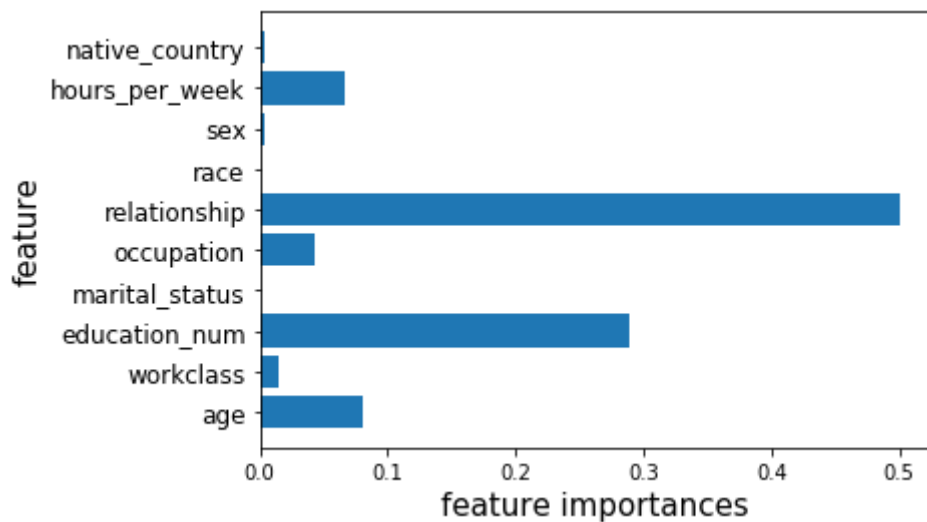
이러한 방식으로 모든 트리가 생성된 후, 각 feature의 중요도를 판단할 수 있다. 해당 feature가 부모 노드에서 자식 노드로 가는 데에 있어 불순도를 감소시키는 데에 크게 기여했다면, 그 feature는 중요한 것으로 볼 수 있다.

실습 5. income dataset: feature importance

사이킷런에서는 `DecisionTreeClassifier`의 `feature_importances_`를 통해 각 feature의 중요도를 배열로 반환해 준다.

income data에 대한 의사결정나무를 생성하고, 특성 중요도를 구해 시각화해 보자.

```
(...)
feature_importance = dt_clf.feature_importances_
(...)
```



column이랑 index 잘 맞춰주자. 제발....

특성 중요도를 통해 판단한 결과, 교육 연수(education_num)와 가족 관계(relationship)의 기여도가 높은 것으로 나타났다. 상대적으로 중요한 feature라고 판단할 수 있다.

7. 트리 그림

트리 분기 과정을 그림으로 나타낼 수도 있다. 강의에서는 실습 진행하지 않을 테니 찾아보자.

출처: <https://yamalab.tistory.com/31>

```
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image

dot_data = export_graphviz(`tree_variable_name`,
                           out_file=None,
                           feature_names=['feature_name_1', 'feature_name_2',
...],
                           class_names=['class_1', 'class_2', ...],
                           filled=True,
                           rounded=True,
                           special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

출처: [박태정 강사님 강의자료](#)

```
import pydotplus
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from IPython.display import Image
from sklearn import tree

iris = datasets.load_iris()
features, target = iris.data, iris.target
dt = DecisionTreeClassifier()
model = dt.fit(features, target)
dot_data = tree.export_graphviz(dt,
                                out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
graph.write_pdf('iris.pdf') # pdf 파일 생성
graph.write_png('iris.png') # png 파일 생성
```