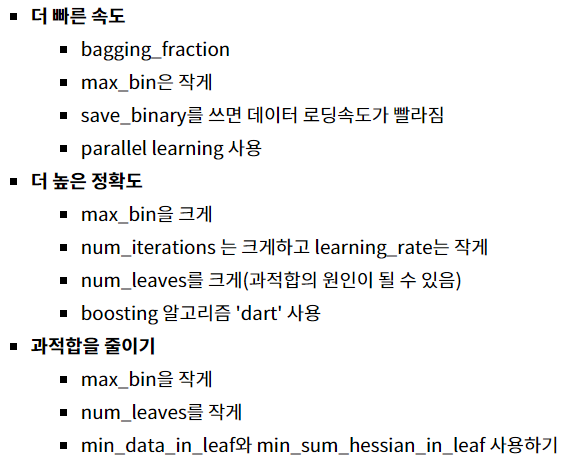
**LightGBM**

* lightGBM은 트리의 균형은 맞추지 않고 리프 노드를 지속적으로 분할하면서 진행
* 리프 노드를 max delta loss값을 가지는 리프 노드를 계속 분할하기 때문에 비대칭적이고 깊은 트리가 생성되지만 동일한 리프를 생성할 때 손실을 줄일 수 있다는 장점이 있음
* Max delta loss가 증가하도록 잎의 개수를 정함
* 메모리를 적게 차지하고 속도가 빠르다는 장점, 정확도가 높다는 장점, GPU를 지원해준다는 장점
* **과적합(overfitting)에 민감하여 데이터의 크기가 작을 경우 기존 머신러닝 알고리즘이 더 좋을 수 있음. 따라서 경험적으로 데이터의 개수(행 수)가 10,000개 이상일 때 추천함**
* 주요 하이퍼파라미터 (매우 복잡하고 100개 이상의 파라미터가 있다)
  + learning\_rate
  + num\_iterations
  + application = 모델의 종류, regression(default), binary, multiclass를 설정
  + device = cpu or gpu
  + metric = 지표 파라미터, mae, mse, binary\_logloss, multi\_logloss
  + max\_bin = feature 값의 최대 bin 수
  + categorical\_feature = 범주형 feature의 인덱스를 의미, 0,1,2이면 column0, column1, column2가 범주형 변수들임
  + ignore\_column = 범주형 feature로써 특정 column을 고려하지 않는 것, 그 변수들을 무시하는 것
  + Task = 데이터에 대해서 수행하고자 하는 임무를 구체화. train or predict
  + num\_leaves = 전체 Tree의 leave 수, default = 31
  + max\_depth = 트리의 최대 깊이. -1로 설정하면 제한없이 분기하며, feature가 많다면 크게 설정 -> 과적합이 된다고 생각된다면 줄이는 것을 추천
  + min\_data\_in\_leaf = 리프가 가지고 있는 최소한의 레코드 수. default = 20. 과적합 해결 시 주로 사용함
  + boosting = 기본값은 gbdt(Gradient Boosting Decision Tree), 정확도가 중요할 때는 딥러닝의 드랍아웃과 같은 dart를 사용 (rf: Random Forest)
  + num\_boost\_round = boosting iteration 수, 일반적을 100이상
  + bagging\_fraction = 배깅을 하기 위해서 데이터를 랜덤 샘플링하여 학습에 사용. 비율은 0~1이고 0은 포함하지 않음
  + feature\_fraction = Boosting이 Random Forest일 경우 사용. 1보다 작다면 LGBM은 매 iteration(tree)마다 다른 feature를 랜덤하게 추출하여 학습. 만약 0.8로 값을 설정하면 매 tree를 구성할 때, feature의 80%만 랜덤하게 선택. 과적합을 방지하기 위해 사용할 수 있으며 학습속도가 향상됨
  + min\_gain\_to\_split = 분기하기 위해 필요한 최소한의 gain을 의미. Tree에서 유용한 분기의 수를 컨트롤하는데 사용함
  + max\_cat\_gruop = 카테코리 수가 클 때, 과적합을 방지하는 분기 포인트를 찾음. lightgbm 알고리즘이 카테고리 그룹을 max\_cat\_gruop 그룹으로 합치고 그룹 경계선에서 분기 포인트를 찾음. default = 64
  + scale\_pos\_weight = 클래스 불균형의 데이터 셋에서 weight를 주는 방식으로 positive를 증가시킴. 기본값은 1이며 불균형의 정도에 따라 조절
  + early\_stopping\_round = Validation set에서 평가지표가 더 이상 향상되지 않으면 학습을 정지. 평가지표의 향상이 n round 이상 지속되면 학습을 정지함
  + lambda(lambda\_l1, lambda\_l2) = 정규화를 통해 과적합을 방지할 수 있지만, 정확도를 저하시킬 수도 있기 때문에 일반적으로 default 값인 0으로 둠



1. num\_leaves : Tree 모델의 복잡성을 컨트롤하는 주요 파라미터입니다. 이상적으로 num\_leaves 값은 2 ^ (max\_depth) 값보다 적거나 같아야 합니다. 이것보다 많은 값은 과적합을 유발할 것입니다.
2. min\_data\_in\_leaf : 큰 값으로 세팅하는 것은 Tree가 너무 깊게 확장되는 것을 막을 수 있지만 under-fitting 언더 피팅이 발생할 수도 있습니다. 관행적으로, 수백 또는 수천 개로 정하는 것이 큰 데이터 세트에 충분합니다.
3. max\_depth : Tree 깊이를 명확하게 제한하기 위해 max\_depth 값을 설정할 수도 있습니다.

**더 빠른 속도를 위하여 :**

* bagging\_fraction과 baggin\_freq 을 설정하여 bagging 을 적용하십시오
* feature\_fraction을 설정하여 feature sub-sampling을 하십시오
* 작은 max\_bin 값을 사용하십시오
* save\_binary 를 값을 통해 다가오는 학습에서 데이터 로딩 속도를 줄이십시오
* parallel learning 병렬 학습을 적용하십시오

**더 나은 정확도를 위해 :**

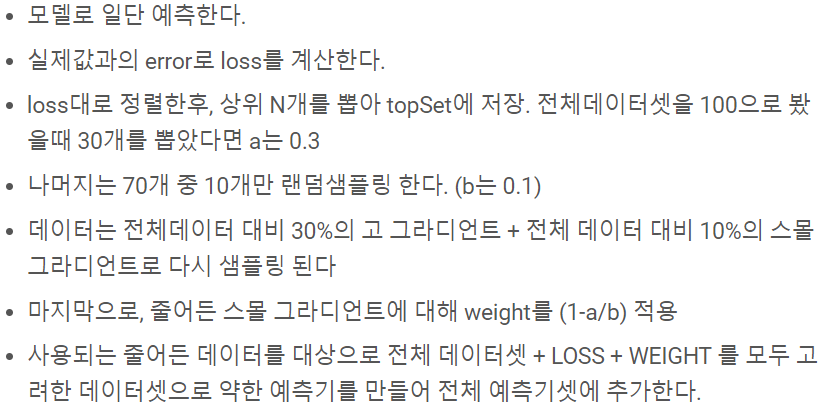
* 큰 max\_bin 값을 사용하십시오 (아마 속도는 느려질 수 있습니다)
* 작은 learning\_rate 값을 큰 num\_iterations 값과 함께 사용하십시오
* 큰 num\_leaves 값을 사용하십시오 (아마 과적합을 유발할 수도 있습니다)
* 더 큰 트레이닝 데이터를 사용하십시오
* dart 를 사용하십시오
* 범주형 feature를 사용하십시오

**과적합을 해결하기 위해 :**

* 작은 max\_bin 값을 사용하십시오
* 작은 num\_leaves 값을 사용하십시오
* min\_data\_in\_leaf 와 min\_sum\_hessian\_in\_leaf 파라미터를 사용하십시오
* bagging\_fraction 과 bagging\_freq 을 사용하여 bagging 을 적용하십시오
* feature\_fraction을 세팅하여 feature sub-sampling을 하십시오
* lambda\_l1, lambda\_l2 그리고 min\_gain\_to\_split 파라미터를 이용해 regularization (정규화) 를 적용하십시오
* max\_depth 를 설정해 Deep Tree 가 만들어지는 것을 방지하십시오
* GBDT(Gradient Boosting Decision Tree)

weight는 없지만, gradient가 있다. 데이터 개수를 내부적으로 줄여서 계산할 때, 큰 gradient를 가진 데이터는 놔두고 작은 gradient는 랜덤하게 드랍한다. 작은 gradient만 drop하므로 one-side sampling이라고 부른다. gradient는 잔차를 이용하는 것으로 이해하면 된다.

Gradient가 적다고 버리면, 데이터 분포자체가 왜곡되기 때문에 이 상태에서 훈련하면 정확도가 낮아지게 된다. 따라서, 낮은 gradient의 값들은 가져와서 버린 샘플만큼 뻥튀기한다. 1 – a / b 를 곱해서 수를 맞춰준다. (a는 탑n개의 비율, b는 샘플링n비율)



논문에서는 이 방법이 데이터가 많았을 때 정확도를 해치지 않음을 수학적으로 증명함

랜덤샘플링 하는 방법이 배깅과 맞닿아 있어서 좀 더 다양성을 높이고, 오히려 generalization이 잘되는 측면도 있을 수 있다.

* 라벨인코딩 = lightgbm은 카테고리형태의 타입을 넣으면 알아서 분기하는데, 라벨인코딩이 약간 정확도가 좀 더 높은 것으로 사람들이 인식하고 있다. 라벨인코딩을 해도 되는 경우가 확실한 경우 라벨인코딩을 쓰는 것이 나은 것으로 추측
* 파라미터 최적화 = 파라미터 최적화보다는 시간이 허용된다면 조금이나마 Feature Engineering을 고민하는 편이 더 나은 결과를 얻을 수 있다.
* <http://machinelearningkorea.com/2019/09/29/lightgbm-%ED%8C%8C%EB%9D%BC%EB%AF%B8%ED%84%B0/>
* 1등 코드에서의 light lgb 구동 방법

lgb\_train = training set

lgb\_valid = validation set

model = lgb.train(parameters, lgb\_train, valid\_sets = lgb\_valid, num\_boost\_round, early\_stopping\_rounds, vervose\_eval, feature\_name)

lgb\_model = lgb\_train(x\_train, x\_valid, y\_train, y\_valid, feature\_name=None, plot=False)

lgb\_test, lgb\_valid = lgb\_model.predict(x\_test).flatten(), lgb\_model.predict(x\_valid).flatten()

lgb\_test\_score, lgb\_valid\_score = r2\_score(y\_test, lgb\_test), r2\_score(y\_valid, lgb\_valid)

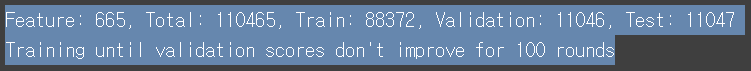
add\_data\_sets = additional\_data(weather data)

기상데이터를 포함해서 그냥 training set & validation set & test set 나눔

X\_ = (110465, 665)의 형태를 가지고 있음 ( 세대 구분 없이 concatenate )

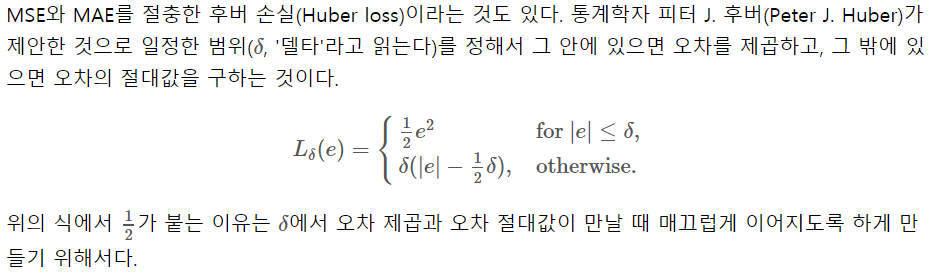
* 맞는지 틀린지 확실치는 않음

X\_.shape[1], X\_.shape[0], x\_train.shape[0], x\_valid.shape[0], x\_test.shape[0]



Loss function

1. huber loss



<https://en.wikipedia.org/wiki/Huber_loss>

1. ㅇㅇ

* feature\_fraction을 높이면 smape가 줄어듬 ( 모든 기간을 학습해서 그런 것으로 예상 )