**Optimizasyon Parametreleri**

**Random Forest Parameters**

**1) max\_features:** These are the maximum number of features Random Forest is allowed to try an individual tree. There are multiple options available in Python to assign maximum features. Here are a few of them :

1. Auto/None: This will simply take all the features which make sense in every tree. Here we simply do not put any restrictions on the individual tree.
2. sqrt: This option will take the square root of the total number of features in the individual runs. For instance, if the total number of variables is 100, we can only take 10 of them in an individual tree.”log2″ is another similar type of option for max\_features.
3. 0.2: This option allows the random forest to take 20% of the variables in the individual run. We can assign and value in the format “0. x” where we want x% of features to be considered.

*Bu parametre, her karar ağacında kullanılacak özelliklerin sayısını belirler. max\_features değerini arttırdıkça, karar ağaçları daha fazla özellik kullanır ve doğruluğu arttırır.*

**2) n\_estimators:** This is the number of trees you want to build before taking the maximum voting or averages of predictions. A higher number of trees gives you better performance but makes your code slower. You should choose as high a value as your processor can handle because this makes your predictions stronger and more stable.

*Bu parametre, kullanılan karar ağaçlarının sayısını belirler. Genellikle, n\_estimators değerini arttırdıkça modelin doğruluğu da artar.*

#### 3) min\_sample\_leaf: If you have built a decision tree before, you can appreciate the importance of minimum sample leaf size. The leaf is the end node of a decision tree. A smaller leaf makes the model more prone to capturing noise in train data. Generally, I prefer a minimum leaf size of more than 50. However, you should try multiple leaf sizes to find the most optimum for your use case.

#### *Bu parametre, her dalın en az örnek sayısını belirler. min\_samples\_leaf değerini arttırdıkça, karar ağaçları daha az dallı hale gelir ve overfitting riski azalır.*

#### 4) oob\_score: This is a random forest cross-validation method. It is very similar to leaving one out validation technique, however, this is so much faster. This method simply tags every observation used in different trees. And then it finds out a maximum vote score for every observation based on only trees which did not use this particular observation to train themselves.

#### Kullanımı:

model = RandomForestRegressor(n\_estimator = 100, oob\_score = TRUE, n\_jobs = -1, random\_state =50, max\_features = "auto", min\_samples\_leaf = 50)

#### 5) max\_depth: The *max\_depth* of a tree in Random Forest is defined as the longest path between the root node and the leaf node.

#### *Bu parametre, her karar ağacının derinliğini sınırlandırır. max\_depth'i arttırdıkça, karar ağaçları daha detaylı hale gelir ancak overfitting riski artar.*

**6***)* **criterion:**The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

**7)** **Min\_samples\_split:**This parameter decides the minimum number of samples required to split an internal node. Default value =2. The problem with such a small value is that the condition is checked on the terminal node. If the data points in the node exceed the value 2, then further splitting takes place. Whereas if a more lenient value like 6 is set, then the splitting will stop early and the decision tree won’t overfit on the data.

*Bu parametre, her dal için gerekli olan en az örnek sayısını belirler. min\_samples\_split değerini arttırdıkça, karar ağaçları daha az dallı hale gelir ve overfitting riski azalır.*

**8) Max\_leaf\_nodes:** With the help of this hyperparameter, a condition can be set on the splitting of the nodes in the tree. Thus, the growth of the tree gets automatically restricted.

**9) bootstrap :**Bu parametre, veri setinden rastgele seçilen örneklerin kullanılıp kullanılmayacağını belirler. Bootstrap=True ise, veri setinden rastgele seçilen örnekler kullanılır.

from sklearn.ensemble import RandomForestRegressor

# Eğitim ve test verilerinizi tanımlayın

X\_train, y\_train, X\_test, y\_test = ...

# Random Forest modelini tanımlayın

rf\_model = RandomForestRegressor(n\_estimators=100,

max\_depth=None,

min\_samples\_split=2,

min\_samples\_leaf=1,

max\_features='auto',

bootstrap=True)

# Modeli eğitin

rf\_model.fit(X\_train, y\_train)

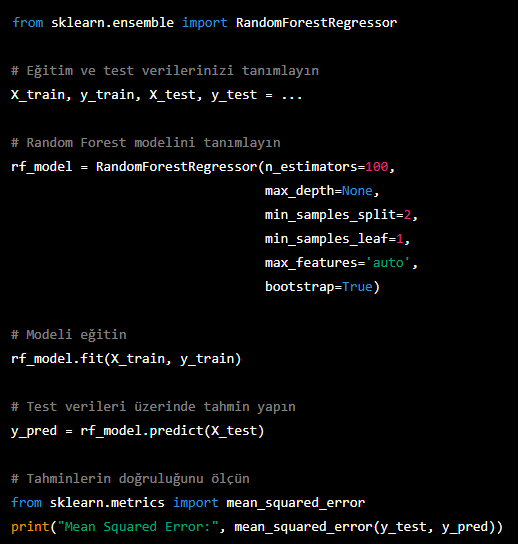
# Test verileri üzerinde tahmin yapın

y\_pred = rf\_model.predict(X\_test)

# Tahminlerin doğruluğunu ölçün

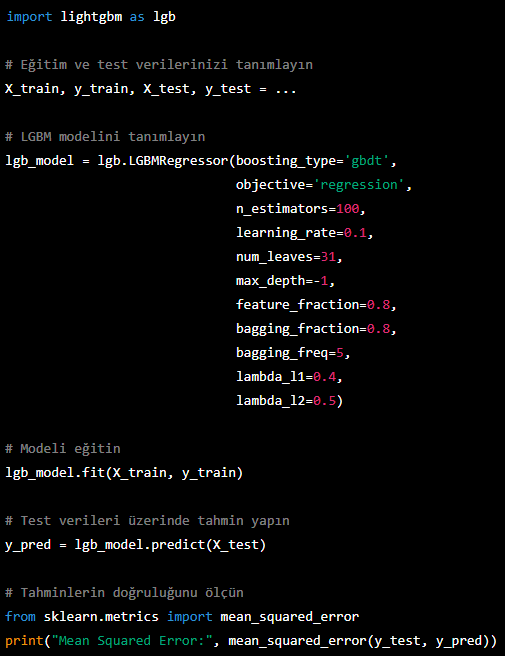
from sklearn.metrics import mean\_squared\_error

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

**

**Lightgbm (LGBM) Parameters**

1. **num\_leaves:** Bu parametre, ağaç dallarının sayısını belirler ve modelin doğruluğunu arttırır. Ancak, num\_leaves'i çok büyük yaparsanız, overfitting riski artar.
2. **max\_depth:** Bu parametre, ağaç derinliğini sınırlandırır ve overfitting riskini azaltır.
3. **feature\_fraction:** Bu parametre, rastgele seçilen özelliklerin yüzdesini belirler ve modelin doğruluğunu arttırır.
4. **bagging\_fraction:** Bu parametre, eğitim verisinden rastgele seçilen örneklerin yüzdesini belirler ve modelin doğruluğunu arttırır.
5. **lambda\_l1, lambda\_l2:** Bu parametreler, model ağırlıklarının L1 ve L2 regularizasyonunu belirler ve overfitting riskini azaltır.
6. **min\_data\_in\_leaf:**Prevents overfitting. It’s set depending on num\_leaves and the number of training samples. For a large dataset, it can be set to hundreds or thousands.
7. **learning\_rate:** the boosting learning rate
8. **n\_estimators:** number of boosted trees to fit — defaults to 200000

**Not: Bu parametrelerin tümünün uygun değerlerini bulmak için modelin eğitim ve test verileri üzerinde hiperparametre tuning yapmanız gerekir.**

import lightgbm as lgb

# Eğitim ve test verilerinizi tanımlayın

X\_train, y\_train, X\_test, y\_test = ...

# LGBM modelini tanımlayın

lgb\_model = lgb.LGBMRegressor(boosting\_type='gbdt',

objective='regression',

n\_estimators=100,

learning\_rate=0.1,

num\_leaves=31,

max\_depth=-1,

feature\_fraction=0.8,

bagging\_fraction=0.8,

bagging\_freq=5,

lambda\_l1=0.4,

lambda\_l2=0.5)

# Modeli eğitin

lgb\_model.fit(X\_train, y\_train)

# Test verileri üzerinde tahmin yapın

y\_pred = lgb\_model.predict(X\_test)

# Tahminlerin doğruluğunu ölçün

from sklearn.metrics import mean\_squared\_error

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))