Bu eğitimde, **gradyan artırma (gradient boosting)** ile modelleri nasıl oluşturacağınızı ve optimize edeceğinizi öğreneceksiniz. Bu yöntem, birçok Kaggle yarışmasına hakimdir ve çeşitli veri kümelerinde en iyi sonuçları elde eder.

Introduction

Bu kursun büyük bir bölümünde, tek bir karar ağacının tahminlerinin ortalamasını alarak daha iyi performans gösteren **rastgele orman (random forest)** yöntemiyle tahminler yaptınız.

Rastgele orman yöntemini bir "**topluluk yöntemi (ensemble method)**" olarak adlandırıyoruz. Tanım olarak, **topluluk yöntemleri**, birkaç modelin (örneğin, rastgele ormanlar durumunda birkaç ağacın) tahminlerini birleştirir.

Şimdi, **gradyan artırma (gradient boosting)** adı verilen başka bir topluluk yöntemini öğreneceğiz.

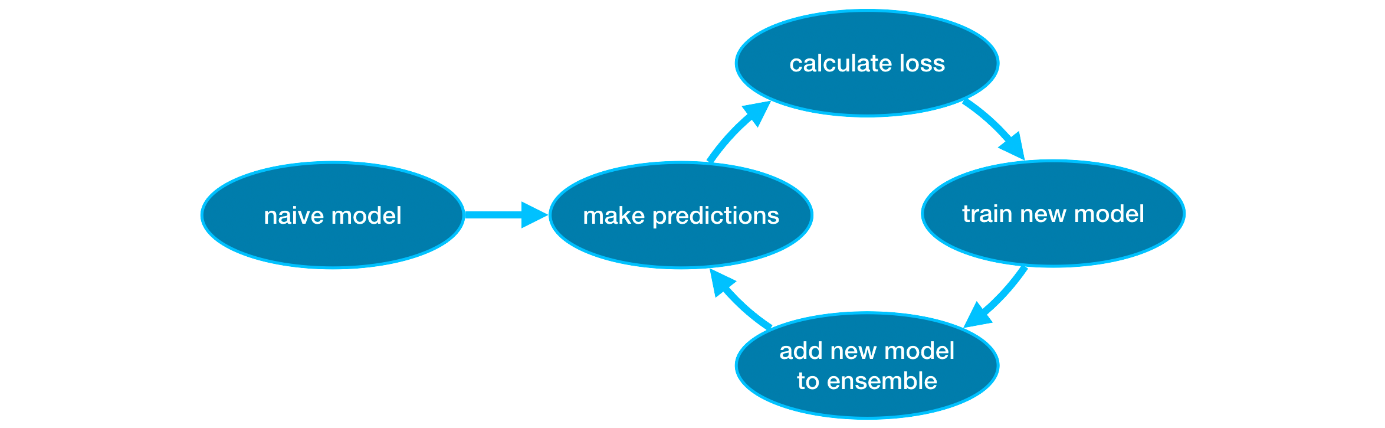
# Gradient Boosting

**Gradyan artırma (Gradient boosting)**, bir topluluğa (ensemble) aşamalı olarak modeller eklemek için döngülerden geçen bir yöntemdir.

Süreç, topluluğu, tahminleri oldukça basit olabilecek tek bir modelle başlatır. (Tahminleri çok yanlış olsa bile, topluluğa sonradan eklenen modeller bu hataları giderecektir.)

Ardından, döngü başlar:

* Önce, mevcut topluluğu kullanarak veri setindeki her gözlem için tahminler oluştururuz. Bir tahmin yapmak için topluluktaki tüm modellerin tahminlerini toplarız.
* Bu tahminler, bir kayıp fonksiyonunu (örneğin **ortalama karesel hata**) hesaplamak için kullanılır.
* Daha sonra, bu kayıp fonksiyonunu kullanarak topluluğa eklenecek yeni bir model oluştururuz. Özellikle, yeni modeli topluluğa eklemenin kaybı azaltacağı şekilde model parametrelerini belirleriz. (Ek Bilgi: "Gradient boosting"deki "gradyan" kelimesi, bu yeni modelin parametrelerini belirlemek için kayıp fonksiyonu üzerinde **gradyan inişini (gradient descent)** kullanmamıza atıfta bulunur.)
* Son olarak, yeni modeli topluluğa ekleriz ve...
* ... tekrarlarız!



# Example

Eğitim ve doğrulama verilerini X\_train, X\_valid, y\_train ve y\_valid'e yükleyerek başlıyoruz.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

*# Read the data*

data = pd.read\_csv('../input/melbourne-housing-snapshot/melb\_data.csv')

*# Select subset of predictors*

cols\_to\_use = ['Rooms', 'Distance', 'Landsize', 'BuildingArea', 'YearBuilt']

X = data[cols\_to\_use]

*# Select target*

y = data.Price

*# Separate data into training and validation sets*

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y)

Bu örnekte, **XGBoost** kütüphanesiyle çalışacaksınız. **XGBoost**, performans ve hıza odaklanan birkaç ek özellikle gradyan artırmanın bir uygulaması olan **extreme gradient boosting'in** kısaltmasıdır. (Scikit-learn'ün gradyan artırmanın başka bir versiyonu olsa da, XGBoost'un bazı teknik avantajları vardır.)

Bir sonraki kod hücresinde, XGBoost için scikit-learn API'sini (**xgboost.XGBRegressor**) içe aktarıyoruz. Bu, tıpkı scikit-learn'de olduğu gibi bir model oluşturmamıza ve eğitmemize olanak tanır. Çıktıda göreceğiniz gibi, **XGBRegressor** sınıfının ayarlanabilir birçok parametresi vardır; bunları yakında öğreneceksiniz!

from xgboost import XGBRegressor

my\_model = XGBRegressor()

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

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None,

colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1,

early\_stopping\_rounds=None, enable\_categorical=False,

eval\_metric=None, gamma=0, gpu\_id=-1, grow\_policy='depthwise',

importance\_type=None, interaction\_constraints='',

learning\_rate=0.300000012, max\_bin=256, max\_cat\_to\_onehot=4,

max\_delta\_step=0, max\_depth=6, max\_leaves=0, min\_child\_weight=1,

missing=nan, monotone\_constraints='()', n\_estimators=100, n\_jobs=0,

num\_parallel\_tree=1, predictor='auto', random\_state=0, reg\_alpha=0,

reg\_lambda=1, ...)

Ayrıca tahminlerde bulunuyoruz ve modeli değerlendiriyoruz.

from sklearn.metrics import mean\_absolute\_error

predictions = my\_model.predict(X\_valid)

print("Mean Absolute Error: " + str(mean\_absolute\_error(predictions, y\_valid)))

Mean Absolute Error: 241041.5160392121

# Parameter Tuning

XGBoost'un doğruluğu ve eğitim hızını önemli ölçüde etkileyebilecek birkaç parametresi vardır. Anlamanız gereken ilk parametreler şunlardır:

### n\_estimators

**n\_estimators**, yukarıda açıklanan modelleme döngüsünden kaç kez geçileceğini belirtir. Topluluğa (ensemble) dahil ettiğimiz model sayısına eşittir.

* Çok **düşük** bir değer **yetersiz öğrenmeye (underfitting)** neden olur, bu da hem eğitim verilerinde hem de test verilerinde hatalı tahminlere yol açar.
* Çok **yüksek** bir değer **aşırı öğrenmeye (overfitting)** neden olur, bu da eğitim verilerinde doğru, ancak test verilerinde (önemsediğimiz asıl veri) hatalı tahminlere yol açar.

Tipik değerler 100-1000 arasındadır, ancak bu, aşağıda tartışılan **learning\_rate** parametresine büyük ölçüde bağlıdır.

Toplulukta yer alacak model sayısını ayarlama kodu şöyledir:

my\_model = XGBRegressor(n\_estimators=500)

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

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None,

colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1,

early\_stopping\_rounds=None, enable\_categorical=False,

eval\_metric=None, gamma=0, gpu\_id=-1, grow\_policy='depthwise',

importance\_type=None, interaction\_constraints='',

learning\_rate=0.300000012, max\_bin=256, max\_cat\_to\_onehot=4,

max\_delta\_step=0, max\_depth=6, max\_leaves=0, min\_child\_weight=1,

missing=nan, monotone\_constraints='()', n\_estimators=500, n\_jobs=0,

num\_parallel\_tree=1, predictor='auto', random\_state=0, reg\_alpha=0,

reg\_lambda=1, ...)

## erken\_durdurma\_döngüsü (early\_stopping\_rounds)

**early\_stopping\_rounds**, n\_estimators için ideal değeri otomatik olarak bulmanın bir yolunu sunar. Erken durdurma, n\_estimators için belirlenen üst sınıra ulaşılmamış olsa bile, doğrulama skoru iyileşmeyi bıraktığında modelin yinelemeyi durdurmasına neden olur. **n\_estimators** için yüksek bir değer belirleyip, ardından yinelemeyi durdurmak için en uygun zamanı bulmak amacıyla **early\_stopping\_rounds** kullanmak akıllıca bir yaklaşımdır.

Rastgele şans, bazen doğrulama skorlarının iyileşmediği tek bir döngüye neden olabildiğinden, durmadan önce kaç tur üst üste kötüleşmeye izin vereceğinize dair bir sayı belirtmeniz gerekir. **early\_stopping\_rounds=5** belirlemek makul bir seçimdir. Bu durumda, 5 ardışık kötüleşen doğrulama skorundan sonra dururuz.

**early\_stopping\_rounds** kullanırken, doğrulama skorlarını hesaplamak için bir miktar veriyi ayırmanız da gerekir; bu, **eval\_set** parametresi ayarlanarak yapılır.

Yukarıdaki örneği, erken durdurmayı içerecek şekilde değiştirebiliriz:

my\_model = XGBRegressor(n\_estimators=500)

my\_model.fit(X\_train, y\_train,

early\_stopping\_rounds=5,

eval\_set=[(X\_valid, y\_valid)],

verbose=False)

/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:797: UserWarning: `early\_stopping\_rounds` in `fit` method is deprecated for better compatibility with scikit-learn, use `early\_stopping\_rounds` in constructor or`set\_params` instead.

UserWarning,

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None,

colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1,

early\_stopping\_rounds=None, enable\_categorical=False,

eval\_metric=None, gamma=0, gpu\_id=-1, grow\_policy='depthwise',

importance\_type=None, interaction\_constraints='',

learning\_rate=0.300000012, max\_bin=256, max\_cat\_to\_onehot=4,

max\_delta\_step=0, max\_depth=6, max\_leaves=0, min\_child\_weight=1,

missing=nan, monotone\_constraints='()', n\_estimators=500, n\_jobs=0,

num\_parallel\_tree=1, predictor='auto', random\_state=0, reg\_alpha=0,

reg\_lambda=1, ...)

Daha sonra tüm verilerinizle bir model oluşturmak isterseniz, n\_estimators değerini erken durdurma ile çalıştırdığınızda en uygun bulduğunuz değere ayarlayın.

### **learning\_rate**[**¶**](https://www.kaggle.com/code/alexisbcook/xgboost#learning_rate)

Tahminleri, her bir bileşen modelin tahminlerini basitçe toplayarak elde etmek yerine, her modelin tahminlerini küçük bir sayı ( **öğrenme oranı (learning rate)** olarak bilinir) ile çarptıktan sonra toplayabiliriz.

Bu, topluluğa eklediğimiz her ağacın bize daha az yardımcı olduğu anlamına gelir. Bu sayede, aşırı öğrenme (overfitting) olmadan **n\_estimators** için daha yüksek bir değer belirleyebiliriz. Erken durdurma (early stopping) kullanırsak, uygun ağaç sayısı otomatik olarak belirlenir.

Genel olarak, küçük bir öğrenme oranı ve büyük sayıda tahminci, daha doğru XGBoost modelleri sağlar, ancak döngüden daha fazla yineleme yaptığı için modelin eğitilmesi daha uzun sürer. Varsayılan olarak, XGBoost **learning\_rate=0.1** olarak ayarlar.

Yukarıdaki örneği öğrenme oranını değiştirecek şekilde düzenlemek, aşağıdaki kodu verir:

my\_model = XGBRegressor(n\_estimators=1000, learning\_rate=0.05)

my\_model.fit(X\_train, y\_train,

early\_stopping\_rounds=5,

eval\_set=[(X\_valid, y\_valid)],

verbose=False)

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None,

colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1,

early\_stopping\_rounds=None, enable\_categorical=False,

eval\_metric=None, gamma=0, gpu\_id=-1, grow\_policy='depthwise',

importance\_type=None, interaction\_constraints='',

learning\_rate=0.05, max\_bin=256, max\_cat\_to\_onehot=4,

max\_delta\_step=0, max\_depth=6, max\_leaves=0, min\_child\_weight=1,

missing=nan, monotone\_constraints='()', n\_estimators=1000,

n\_jobs=0, num\_parallel\_tree=1, predictor='auto', random\_state=0,

reg\_alpha=0, reg\_lambda=1, ...)

### **n\_jobs**

Çalışma süresinin önemli olduğu daha büyük veri setlerinde, modellerinizi daha hızlı oluşturmak için paralelliği kullanabilirsiniz. **n\_jobs** parametresini makinenizdeki çekirdek sayısına eşit olarak ayarlamak yaygın bir uygulamadır. Daha küçük veri setlerinde bu yardımcı olmaz.

Elde edilen model daha iyi olmayacaktır, bu nedenle eğitim süresini mikro düzeyde optimize etmek genellikle bir dikkat dağınıklığından başka bir şey değildir. Ancak, **fit** komutu sırasında uzun süre bekleyeceğiniz büyük veri setlerinde bu faydalıdır.

İşte değiştirilmiş örnek:

my\_model = XGBRegressor(n\_estimators=1000, learning\_rate=0.05, n\_jobs=4)

my\_model.fit(X\_train, y\_train,

early\_stopping\_rounds=5,

eval\_set=[(X\_valid, y\_valid)],

verbose=False)

# Conclusion

XGBoost, standart tablo verileriyle (Pandas DataFrames'te depoladığınız veri türü, resim ve video gibi daha sıra dışı veri türlerinin aksine) çalışmak için lider bir yazılım kütüphanesidir. Dikkatli parametre ayarlamasıyla son derece hassas modeller eğitebilirsiniz.