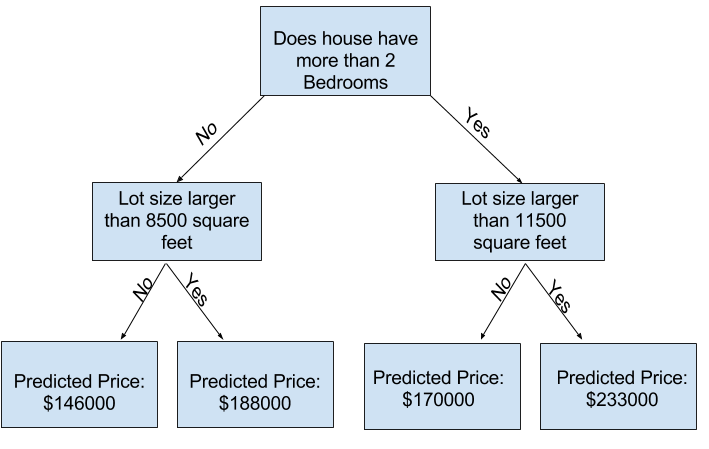
Bu adımın sonunda, yetersiz uyum ve aşırı uyum kavramlarını anlayacak ve bu fikirleri modellerinizi daha doğru hale getirmek için uygulayabileceksiniz.(overfitting,underfitting)

Experimenting With Different Models

Artık modelin doğruluğunu güvenilir bir şekilde ölçebildiğine göre, alternatif modellerle deneyler yapabilir ve en iyi tahminleri hangisinin verdiğini görebilirsin. Ama modeller için ne gibi alternatiflerin var?

Scikit-learn'in **dokümantasyonunda**, karar ağacı modelinin birçok seçeneği olduğunu görebilirsin (birçoklarını uzun bir süre ne isteyeceksin ne de ihtiyacın olacak). En önemli seçenekler, ağacın derinliğini belirler. Bu dersin ilk bölümünden hatırlarsan, bir ağacın derinliği, bir tahmine ulaşmadan önce yaptığı **bölme sayısının** bir ölçüsüdür. Bu, nispeten sığ bir ağaçtır.



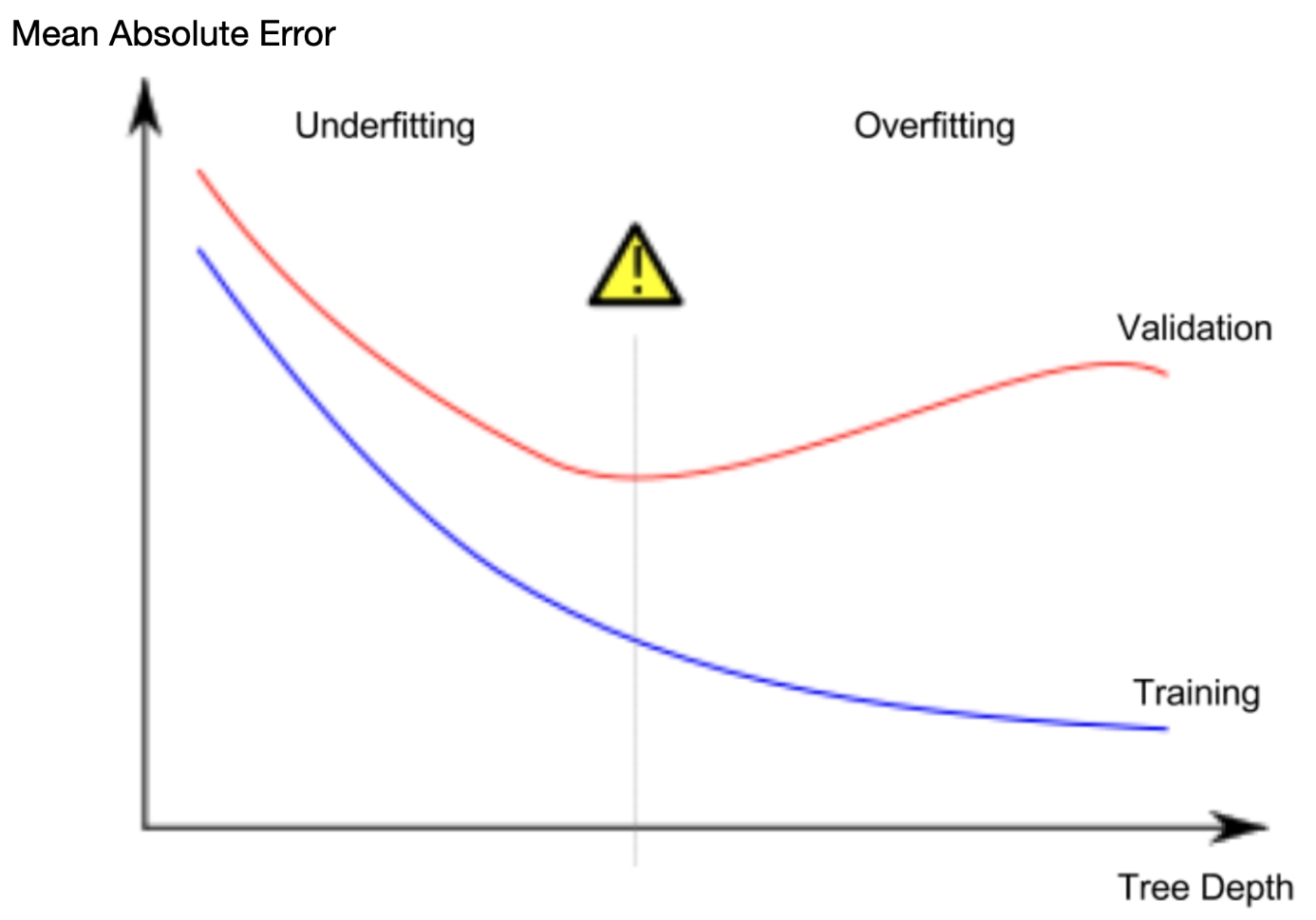
Pratikte, bir ağacın en üst seviyesi (tüm evler) ile bir yaprak arasında 10 bölme olması nadir görülen bir durum değildir. Ağaç derinleştikçe, veri kümesi daha az ev içeren yapraklara ayrılır. Eğer bir ağacın sadece 1 bölmesi olsaydı, veriyi 2 gruba ayırırdı. Eğer her grup tekrar bölünürse, 4 grup ev elde ederdik. Bunların her birini tekrar bölmek 8 grup oluştururdu. Her seviyede daha fazla bölme ekleyerek grup sayısını ikiye katlamaya devam edersek, 10. seviyeye geldiğimizde 210 grup evimiz olur. Bu da 1024 yaprak demektir.

Evleri birçok yaprak arasında böldüğümüzde, her yaprakta daha az ev olur. Çok az ev içeren yapraklar, o evlerin gerçek değerlerine oldukça yakın tahminler yapar, ancak yeni veriler için çok güvenilmez tahminler yapabilirler (çünkü her tahmin sadece birkaç eve dayanır).

Bu, **aşırı uyum (overfitting)** adı verilen bir fenomendir. Aşırı uyumda bir model, eğitim verilerine neredeyse mükemmel bir şekilde uyar, ancak doğrulama ve diğer yeni verilerde kötü performans gösterir. Diğer yandan, eğer ağacımızı çok sığ yaparsak, evleri çok belirgin gruplara ayırmaz.

Aşırı bir durumda, bir ağaç evleri sadece 2 veya 4 gruba ayırırsa, her grup hala çok çeşitli evler içerir. Ortaya çıkan tahminler, eğitim verilerindeki çoğu ev için bile (ve aynı nedenle doğrulama verilerinde de kötü olacaktır) gerçek değerlerden çok uzak olabilir. Bir modelin verilerdeki önemli ayrımları ve kalıpları yakalayamaması nedeniyle eğitim verilerinde bile kötü performans göstermesi durumuna **eksik uyum (underfitting)** denir.

## Yeni verilerdeki doğruluğu önemsediğimiz için, bunu doğrulama verilerimizden tahmin ederek, eksik uyum ile aşırı uyum arasındaki ideal noktayı bulmak isteriz. Görsel olarak, aşağıdaki şekildeki (kırmızı) doğrulama eğrisinin en düşük noktasını bulmak isteriz.



# Example[¶](https://www.kaggle.com/code/dansbecker/underfitting-and-overfitting" \l "Example" \t "_self)

Ağacın derinliğini kontrol etmek için birkaç alternatif vardır ve birçoğu, ağaçtaki bazı yolların diğer yollara göre daha derin olmasına izin verir. Ancak **max\_leaf\_nodes** argümanı, aşırı uyum ile eksik uyum arasında kontrol sağlamanın çok mantıklı bir yolunu sunar. Modele ne kadar çok yaprak oluşturma izni verirsek, yukarıdaki grafikteki eksik uyum alanından aşırı uyum alanına o kadar çok yaklaşırız.

Farklı max\_leaf\_nodes değerlerinden elde edilen MAE (Ortalama Mutlak Hata) skorlarını karşılaştırmaya yardımcı olacak bir yardımcı fonksiyon kullanabiliriz:

from sklearn.metrics import mean\_absolute\_error

from sklearn.tree import DecisionTreeRegressor

def get\_mae(max\_leaf\_nodes, train\_X, val\_X, train\_y, val\_y):

model = DecisionTreeRegressor(max\_leaf\_nodes=max\_leaf\_nodes, random\_state=0)

model.fit(train\_X, train\_y)

preds\_val = model.predict(val\_X)

mae = mean\_absolute\_error(val\_y, preds\_val)

return(mae)

Veriler train\_X, val\_X, train\_y ve val\_y'ye daha önce gördüğünüz (ve daha önce yazdığınız) kod kullanılarak yüklenir.

*# Data Loading Code Runs At This Point*

import pandas as pd

*# Load data*

melbourne\_file\_path = '../input/melbourne-housing-snapshot/melb\_data.csv'

melbourne\_data = pd.read\_csv(melbourne\_file\_path)

*# Filter rows with missing values*

filtered\_melbourne\_data = melbourne\_data.dropna(axis=0)

*# Choose target and features*

y = filtered\_melbourne\_data.Price

melbourne\_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',

'YearBuilt', 'Lattitude', 'Longtitude']

X = filtered\_melbourne\_data[melbourne\_features]

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

*# split data into training and validation data, for both features and target*

train\_X, val\_X, train\_y, val\_y = train\_test\_split(X, y,random\_state = 0)

Farklı max\_leaf\_nodes değerleriyle oluşturulan modellerin doğruluğunu karşılaştırmak için for döngüsü kullanabiliriz.

*# compare MAE with differing values of max\_leaf\_nodes*

for max\_leaf\_nodes **in** [5, 50, 500, 5000]:

my\_mae = get\_mae(max\_leaf\_nodes, train\_X, val\_X, train\_y, val\_y)

print("Max leaf nodes: **%d** **\t\t** Mean Absolute Error: **%d**" %(max\_leaf\_nodes, my\_mae))

Max leaf nodes: 5 Mean Absolute Error: 347380

Max leaf nodes: 50 Mean Absolute Error: 258171

Max leaf nodes: 500 Mean Absolute Error: 243495

Max leaf nodes: 5000 Mean Absolute Error: 254983

Listelenen seçeneklerden 500 yaprak ideal sayıdır.

# Conclusion

Bu konudan çıkarılacak sonuç şudur: Modeller iki sorundan muzdarip olabilir:

* **Aşırı Uyum (Overfitting):** Gelecekte tekrar etmeyecek olan sahte kalıpları yakalayarak daha az doğru tahminlere yol açar.
* **Eksik Uyum (Underfitting):** İlgili kalıpları yakalayamayarak yine daha az doğru tahminlere yol açar.

Aday modellerin doğruluğunu ölçmek için, modelin eğitiminde **kullanılmayan** **doğrulama (validation)** verilerini kullanırız. Bu, birçok farklı aday modeli denememize ve en iyi olanı seçmemize olanak tanır.