İçindekiler

(Intro to Machine Learning)Makine Öğrenimine Giriş

Makine öğrenmesindeki temel fikirleri öğrenin ve ilk modellerinizi oluşturun.

How Models Work (Modeller Nasıl Çalışır):

Giriş:

Makine öğrenimi modellerinin nasıl çalıştığına ve nasıl kullanıldıklarına genel bir bakışla başlayacağız. Daha önce istatistiksel modelleme veya makine öğrenimi yaptıysanız bu temel görünebilir. Endişelenmeyin, yakında güçlü modeller oluşturmaya devam edeceğiz.

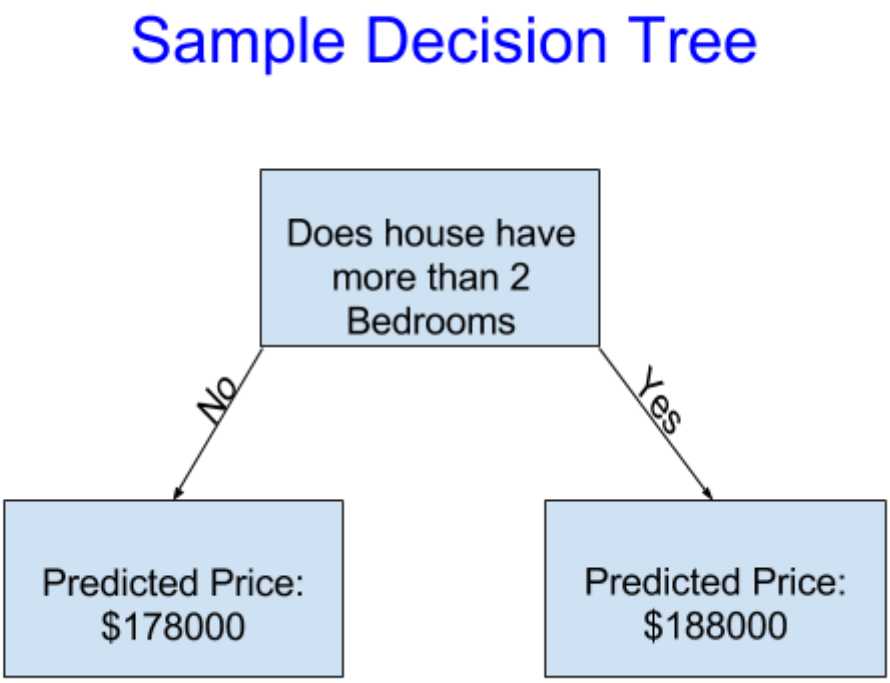
Bu mikro kurs, aşağıdaki senaryodan geçerken modeller oluşturmanızı sağlayacaktır:

Kuzeniniz gayrimenkul konusunda spekülasyonlar milyonlarca dolar kazandı. Veri bilimine gösterdiğiniz ilgi nedeniyle sizinle iş ortağı olmayı teklif etti. Parayı tedarik edecek ve çeşitli evlerin ne kadar değerli olduğunu tahmin eden modeller sunacaksınız.

Kuzeninize geçmişte gayrimenkul değerlerini nasıl tahmin ettiğini soruyorsunuz. Ve bunun sadece sezgi olduğunu söylüyor. Ancak daha fazla sorgulama, geçmişte gördüğü evlerden fiyat örüntülerini belirlediğini ve bu kalıpları düşündüğü yeni evler için tahminler yapmak için kullandığını ortaya koyuyor.

Makine öğrenimi de aynı şekilde çalışır. Karar Ağacı **(Decision Tree)** adlı bir modelle başlayacağız. Daha doğru tahminler veren meraklı modeller var. Ancak karar ağaçları'nın anlaşılması kolaydır ve bunlar veri bilimindeki en iyi modellerin bazıları için temel yapı taşıdır.

Basitlik için, mümkün olan en basit karar ağacıyla başlayacağız.



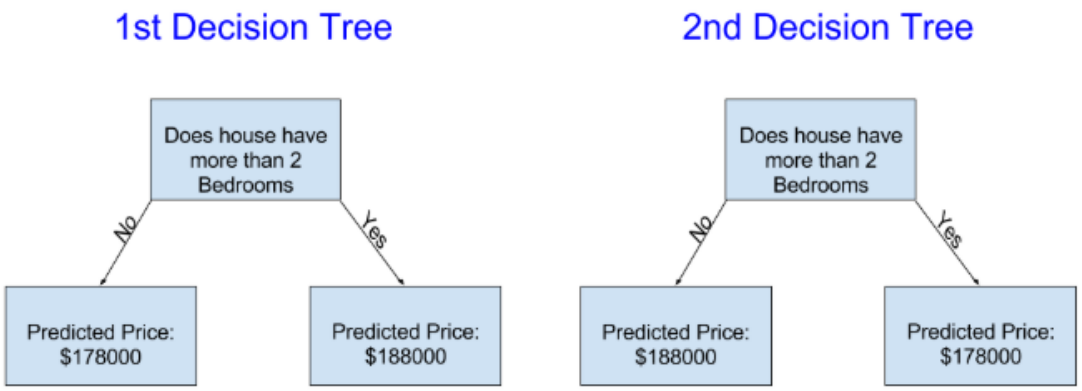
Evleri sadece iki kategoriye ayırır. Dikkate alınan herhangi bir ev için tahmini fiyat, aynı kategorideki evlerin tarihsel ortalama fiyatıdır.

Verileri, evlerin iki gruba nasıl ayrılacağına karar vermek için ve sonra her grupta öngörülen fiyatı belirlemek için kullanıyoruz. Verilerden pattern(desen) yakalamanın bu adımına**, modelin fit edilmesi (fitting)** veya **train edilmesi(training)** denir.

Modelin fit edilmesi için kullanılan verilere **training data** denir. Modelin nasıl fit edildiğine dair ayrıntılar (örneğin, verilerin nasıl bölüneceği) daha sonra kullanmak üzere kayıt edeceğimiz kadar karmaşıktır. Model fit edildikten sonra, yeni evlerin fiyatlarını **predict(tahmin)** edebilmek için yeni verilere uygulayabilirsiniz.

Decision Tree'nin Geliştirilmesi

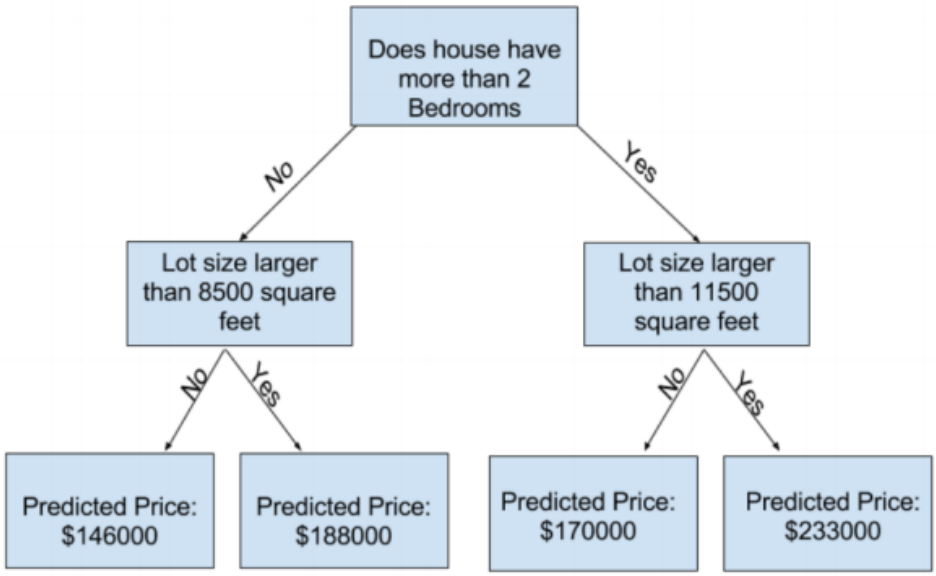
Aşağıdaki iki karardan hangisinin gayrimenkul eğitim verilerinin fit edilmesinden kaynaklanması daha olasıdır?



Soldaki karar ağacı (Karar Ağacı 1) muhtemelen daha mantıklıdır, çünkü daha fazla yatak odası olan evlerin daha az yatak odası olan evlerden daha yüksek fiyatlarla satılma eğiliminde olduğu gerçeğini yakalar.

Bu modelin en büyük eksikliği, banyo sayısı, lot büyüklüğü, yer vb. gibi ev fiyatını etkileyen çoğu faktörü yakalamamasıdır.

Daha fazla **"splits(bölme)"** olan bir ağaç kullanarak daha fazla faktör yakalayabilirsiniz. Bunlara **"deeper(daha derin)" ağaçlar** denir. Her evin toplam lot büyüklüğünü de dikkate alan bir karar ağacı şöyle görünebilir:



Herhangi bir evin fiyatını karar ağacından takip ederek, her zaman o evin özelliklerine karşılık gelen yolu seçerek tahmin edersiniz.

Ev için tahmini fiyat ağacın altındadır.

Altta tahmin yaptığımız noktaya **leaf(yaprak)** denir.

Yapraklardaki **splits(bölünmeler)** ve **values(değerler)** veriler tarafından belirlenecektir, bu nedenle çalışacağınız verileri kontrol etmenin zamanı geldi.

Basic Data Exploration (Temel Veri Keşfi)

Using Pandas to Get Familiar With Your Data (Verilerinizi Öğrenmek için Pandas kullanma):

Herhangi bir makine öğrenimi projesinin ilk adımı, verileri tanımaktır. Bunun için Pandas kütüphanesini kullanacaksınız. **Pandas**, bilim insanlarının verileri keşfetmek ve işlemek için kullandığı temel araçtır. Çoğu kişi **Pandas** kodlarında **pd** olarak kısaltılır. Bunu şu komutla yapıyoruz:

```
In [1]: import pandas as pd
```

Panda kütüphanesinin en önemli kısmı DataFrame'dir. Bir DataFrame, tablo olarak düşünebileceğiniz veri türünü tutar. Bu, Excel'deki bir sayfaya veya SQL veritabanındaki bir tabloya benzer.

Pandas, bu tür verilerle yapmak isteyeceğiniz birçok şey için güçlü yöntemlere sahiptir.

Örnek olarak, Avustralya, Melbourne'daki ev fiyatları hakkındaki verilere bakacağız(https://www.kaggle.com/dansbecker/melbourne-housing-snapshot).

Uygulamalı alıştırmalarda, aynı işlemleri Iowa'da ev fiyatları olan yeni bir veri kümesine uygulayacaksınız.

Örnek (Melbourne) verileri ../input/melbourne-housing-snapshot/melb_data.csv dosya yolundadır.

Verileri aşağıdaki komutlarla yükler ve inceleriz:

- # kolay erişim için dosya yolunu değişkene kaydet

```
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
```

- # verileri okuyun ve DataFrame'de melbourne _data başlıklı verileri depolayın

```
melbourne_data = pd.read_csv(melbourne_file_path)
```

```
melbourne_data.describe()
```

[14]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.000000	8205.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.416127	151.967650	1964.684217	-37.809203	144.995216	7454.417378
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990.669241	541.014538	37.273762	0.079260	0.103916	4378.581772
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000	-38.182550	144.431810	249.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000000	1940.000000	-37.856822	144.929600	4380.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000000	-37.802355	145.000100	6555.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.000000	1999.000000	-37.756400	145.058305	10331.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000	-37.408530	145.526350	21650.000000

Interpreting Data Description (Veri Açıklamalarını Yorumlama):

Sonuçlar, orijinal veri kümenizdeki her sütun için 8 sayı gösterir. İlk sayı, sayı, kaç satırın eksik olmayan değerleri olduğunu gösterir.

Eksik değerler birçok nedenden dolayı ortaya çıkar. Örneğin, 1 yatak odalı bir ev araştırılırken 2. yatak odasının boyutu toplanmaz. Eksik veriler konusuna geri döneceğiz.

İkinci değer, ortalama olan **mean'dir**. Bunun altında **std**, değerlerin sayısal olarak ne kadar yayıldığını ölçen standart sapmadır.

Min,% 25,**% 50**,**% 75** ve maksimum değerleri yorumlamak için, her sütunu en düşükten en yüksek değere doğru sıraladığınızı düşünün.

İlk (en küçük) değer **min**. Liste'nin dörtte birini incelerseniz, değerlerin **% 25**'inden daha büyük ve değerlerin **% 75**'inden daha küçük bir sayı bulacaksınız.

Bu% 25 değerdir ("25. **percentile**" olarak telaffuz edilir). 50. ve 75. yüzdelikler benzer şekilde tanımlanır ve **max**. En büyük sayıdır.

Excercise: Explore Your Data

Bu alıştırma, bir veri dosyasını okuma ve verilerle ilgili istatistikleri anlama yeteneğinizi test edecektir.

Daha sonraki alıştırmalarda, verileri filtrelemek, bir makine öğrenme modeli oluşturmak ve modelinizi yinelemeli olarak geliştirmek için teknikler uygulayacaksınız.

Kurs örnekleri Melbourne'den gelen verileri kullanır. Bu teknikleri kendi başınıza uygulayabilmeniz için, bunları yeni bir veri kümesine (Iowa'dan konut fiyatları) uygulamanız gerekecektir.

Step 1: Loading Data (Veri Yükleme)

Iowa veri dosyasını home_data adlı bir Pandas DataFrame'de okuyun.

```
import pandas as pd

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

# Fill in the line below to read the file into a variable home_data
home_data = pd.read_csv(iowa_file_path)

# Call line below with no argument to check that you've loaded the data correctly
step_1.check()
```

Correct

Step 2: Review The Data (Verileri Gözden Geçirme)

Verilerin özet istatistiklerini görüntülemek için öğrendiğiniz komutu kullanın. Ardından aşağıdaki soruları cevaplamak için değişkenleri doldurun

```
# Print summary statistics in next line
home_data.describe()
```

Out[3]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	...
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	...
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	...
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	...
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	...
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	...
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	...
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	...

8 rows × 38 columns

...	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	SalePrice
...	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
...	94.244521	46.660274	21.954110	3.409589	15.060959	2.758904	43.489041	6.321918	2007.815753	180921.195890
...	125.338794	66.256028	61.119149	29.317331	55.757415	40.177307	496.123024	2.703626	1.328095	79442.502883
...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	2006.000000	34900.000000
...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000	2007.000000	129975.000000
...	0.000000	25.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.000000	2008.000000	163000.000000
...	168.000000	68.000000	0.000000	0.000000	0.000000	0.000000	0.000000	8.000000	2009.000000	214000.000000
...	857.000000	547.000000	552.000000	508.000000	480.000000	738.000000	15500.000000	12.000000	2010.000000	755000.000000

[10]:
```
# What is the average lot size (rounded to nearest integer)?
avg_lot_size = 10517

# As of today, how old is the newest home (current year - the date in which it was built)
newest_home_age = 10

# Checks your answers
step_2.check()
```

Correct

Verilerinizi Düşünün

Verilerinizdeki en yeni ev o kadar yeni değil. Bunun için birkaç potansiyel açıklama:

1- Bu verilerin toplandığı yeni evler inşa etmediler.

2- Veriler uzun zaman önce toplanmıştır. Veri yayımından sonra inşa edilen evler görünmezdi.

Nedeni yukarıdaki 1. açıklama ise, bu, bu verilerle oluşturduğunuz modele olan güveninizi etkiler mi? 2. neden ise ne olur?

Hangi açıklamanın daha mantıklı olduğunu görmek için verileri nasıl inceleyebilirsiniz?

Your First Machine Learning Model:

Selecting Data for Modeling (Modelleme için Veri Seçmek):

Veri kümenizin, kafanızda canlanması veya güzelce ekrana yazdırmak için çok fazla değişkeni vardı. Bu başa çıkılamaz veri miktarını anlayabileceğiniz bir şeye nasıl ayırabilirsiniz?

Sezgimizi kullanarak birkaç değişken seçerek başlayacağız. Daha sonraki kurslar, değişkenleri otomatik olarak önceliklendirmek için istatistiksel teknikleri gösterecektir.

Değişkenleri / sütunları seçmek için veri kümesindeki tüm sütunların bir listesini görmemiz gerekir. Bu, DataFrame'in **columns** özelliği ile yapılır. (Aşağıdaki kodun alt satırı.)

[1]:
```
import pandas as pd
melbourne_file_path='melb_data.csv'
melbourne_data=pd.read_csv(melbourne_file_path)
melbourne_data.columns
```

[1]:
```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

Melbourne verilerinin bazı eksik değerleri vardır (bazı değişkenlerin kaydedilmediği bazı evler.)

```
[10]: melbourne_data.isna().sum()
```

```
[10]: Suburb             0
      Address            0
      Rooms              0
      Type               0
      Price              0
      Method             0
      SellerG            0
      Date               0
      Distance           0
      Postcode           0
      Bedroom2           0
      Bathroom           0
      Car               62
      Landsize           0
      BuildingArea    6450
      YearBuilt       5375
      CouncilArea     1369
      Lattitude          0
      Longtitude         0
      Regionname         0
      Propertycount      0
      dtype: int64
```

Daha sonraki bir derste eksik değerleri ele almayı öğreneceğiz.

Iowa verileriniz, kullandığınız sütunlarda eksik değerlere sahip değildi.

Şimdilik en basit seçeneği alacağız ve verilerimizden eksik değere sahip evleri düşüreceğiz.

dropna eksik değerleri düşürüyor (na'yı "mevcut değil" olarak düşünün)

```
[3]: melbourne_data=melbourne_data.dropna(axis=0)
```

Column'ların içinde kaçar tane eksik veri var ona baktık.

```
[8]: melbourne_data.isna().sum()

[8]: Suburb           0
     Address          0
     Rooms            0
     Type             0
     Price            0
     Method           0
     SellerG          0
     Date             0
     Distance         0
     Postcode         0
     Bedroom2         0
     Bathroom         0
     Car              0
     Landsize         0
     BuildingArea     0
     YearBuilt        0
     CouncilArea      0
     Lattitude        0
     Longtitude       0
     Regionname       0
     Propertycount    0
     dtype: int64
```

Verilerinizin bir alt kümesini seçmenin birçok yolu vardır. Pandas Micro-Course (https://www.kaggle.com/learn/pandas) bunları daha derinlemesine ele alıyor, ancak şimdilik iki yaklaşıma odaklanacağız.

1. "Prediction Target(Tahmin hedefi)"'ni seçmek için kullandığımız nokta gösterimi(dot notation)
2. "Features(Özellikleri)" seçmek için kullandığımız bir sütun listesiyle seçim yapma

Selecting The Prediction Target (Tahmin Hedefini Seçme)

dot-notation ile bir değişkeni(column) veri setinden çekebilirsiniz. Bu tek sütun, genel olarak yalnızca tek bir column'a sahip DataFrame benzeri bir **Seri'**de depolanır.

Tahmin etmek istediğimiz column'u seçmek için dot-notation kullanacağız, buna **prediction target** (tahmin hedefi) denir.

Kural olarak, prediction target (tahmin hedefi) **y** olarak adlandırılır.

Melbourne'deki ev fiyatlarını (price) kaydetmek için gereken kod.

```
y=melbourne_data.Price
y
```

```
0        1480000.0
1        1035000.0
2        1465000.0
3         850000.0
4        1600000.0
           ...
13575    1245000.0
13576    1031000.0
13577    1170000.0
13578    2500000.0
13579    1285000.0
Name: Price, Length: 13580, dtype: float64
```

Choosing "Features" (Özellik Seçimi):

Modelimize girilen sütunlara (ve daha sonra tahminlerde kullanılan sütunlara) "features (özellikler)" denir.

Bizim durumumuzda, bunlar ev fiyatını belirlemek için kullanılan sütunlar olacaktır.

Bazen, target(hedef) hariç tüm sütunları feature(özellik) olarak kullanırsınız. Diğer zamanlarda daha az özellik ile daha iyi olacaksınız.

Şimdilik, sadece birkaç özelliğe sahip bir model oluşturacağız.

Daha sonra, farklı özelliklerle oluşturulan modellerin nasıl tekrarlanacağını ve karşılaştırılacağını göreceksiniz.

Köşeli parantez içine sütun adlarının listesini yazarak birden fazla özellik seçiyoruz. Bu listedeki her öğe bir string (tırnak işaretli) olmalıdır.

Here is an example:

```
[15]: melbourne_features=['Rooms','Bathroom','Landsize','Lattitude','Longtitude']
```

Kural olarak, bu verilere X denir.

```
[16]: X=melbourne_data[melbourne_features]
```

[19]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
0	2	1.0	202.0	-37.79960	144.99840
1	2	1.0	156.0	-37.80790	144.99340
2	3	2.0	134.0	-37.80930	144.99440
3	3	2.0	94.0	-37.79690	144.99690
4	4	1.0	120.0	-37.80720	144.99410
...	...	...	...	...	...
13575	4	2.0	652.0	-37.90562	145.16761
13576	3	2.0	333.0	-37.85927	144.87904
13577	3	2.0	436.0	-37.85274	144.88738
13578	4	1.0	866.0	-37.85908	144.89299
13579	4	1.0	362.0	-37.81188	144.88449

13580 rows × 5 columns

En üstteki birkaç satırı gösteren **head** yöntemini ve **describe** yöntemini kullanarak konut fiyatlarını tahmin etmek için kullanacağımız verileri hızlı bir şekilde inceleyelim.

```
[20]: X.describe()
```

[20]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	13580.000000	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.534242	558.416127	-37.809203	144.995216
std	0.955748	0.691712	3990.669241	0.079260	0.103916
min	1.000000	0.000000	0.000000	-38.182550	144.431810
25%	2.000000	1.000000	177.000000	-37.856822	144.929600
50%	3.000000	1.000000	440.000000	-37.802355	145.000100
75%	3.000000	2.000000	651.000000	-37.756400	145.058305
max	10.000000	8.000000	433014.000000	-37.408530	145.526350

```
[21]: X.head()
```

[21]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
0	2	1.0	202.0	-37.7996	144.9984
1	2	1.0	156.0	-37.8079	144.9934
2	3	2.0	134.0	-37.8093	144.9944
3	3	2.0	94.0	-37.7969	144.9969
4	4	1.0	120.0	-37.8072	144.9941

Verilerinizi bu komutlarla görsel olarak kontrol etmek, bir veri bilim insanının işinin önemli bir parçasıdır. Veri kümesinde sıklıkla daha fazla incelemeyi hak eden sürprizler bulacaksınız.

Building Your Model (Model Oluşturma):

Modellerinizi oluşturmak için **scikit-learn** kütüphanesini kullanacaksınız.

Kodlama yaparken, bu kütüphane örnek kodda göreceğiniz gibi **sklearn** olarak yazılır.

Scikit-learn, tipik olarak DataFrames'da depolanan veri türlerini modellemek için en popüler kütüphanedir.

Bir model oluşturma ve kullanma adımları:

- **define :** Ne tür bir model olacak? Karar ağacı mı? Başka bir model mi? Model tipinin diğer bazı parametreleri de belirtilir.
- **fit :** Sağlanan verilerden pattern(desen) yakalayın. Bu modellemenin kalbidir.
- **predict :** Tahmin
- **evaluate :** Modelin tahminlerinin ne kadar doğru olduğu belirleyin.

İşte **scikit-learn** ile bir **Decision Tree**(Karar Ağaçları) modelini tanımlama ve modeli feature'lara ve target değişkene **fit** etme örneği.

- Modeli tanımlayın. Her çalıştırmada aynı sonuçları sağlamak için random_state için bir sayı belirtin

```
[23]: from sklearn.tree import DecisionTreeRegressor

#Modeli tanımlayın. Her çalıştırmada aynı sonuçları sağlamak için random_state için bir sayı belirtin
melbourne_model=DecisionTreeRegressor(random_state=1)
#Fit model
melbourne_model.fit(X,y)
```

```
[23]: DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, presort='deprecated',
                           random_state=1, splitter='best')
```

random_state: Kodu her çalıştırdığımızda aynı çıktıyı alabilmek için girdiğimiz bir ifade. Örneğin, validation ve training olarak datayı ayırırken Python her seferinde datayı farklı yerlerinden böler, bir random state değeri belirlediğimizde de her çalıştırdığımızda aynı şekilde bölmüş olur ve aynı sonucu vermiş olur. Farklı değerler verdiğinde farklı sonuçlar aldığını göreceksin.

En iyi karar ağacını bulma problemi NP-Complete olarak sınıflandırılan problemlerdendir. Bu tip problemlerin çözümlerinde sezgisel algoritmalar kullanılır. Sezgisel algoritmalarda her kullanıldıklarında en iyi çözümü bulabileceklerini garanti etmezler ve her seferinde farklı sonuçlar üretirler. Dolayısıyla her ağaç inşa ettiğinde ağaç yapısı değişiklik gösterecektir. Modeli her çalıştırdığında aynı ağacı elde etmek istersen **random_state** parametresini bir tamsayıya eşitlemen gerekir. Hangi tamsayıya eşitlediğinin bir önemi yok .

Birçok makine öğrenimi modeli, model eğitiminde bazı rasgeleliklere izin verir.

Random_state için bir sayı belirtmek, her çalıştırmada aynı sonuçları almanızı sağlar. Bu iyi bir uygulama olarak kabul edilir.

Herhangi bir sayı kullanabilirsiniz ve model kalitesi tam olarak hangi değeri seçtiğinize bağlı olmayacaktır.

Herhangi bir sayı kullanabilirsiniz ve model kalitesi tam olarak hangi değeri seçtiğinize bağlı olmayacaktır.

Uygulamada, halihazırda fiyatlarımız olan evler yerine piyasaya çıkan yeni evler için tahminler yapmak isteyeceksiniz.

Ancak, tahmin işlevinin nasıl çalıştığını görmek için egzersiz verilerinin ilk birkaç satırı için tahminler yapacağız.

```
[24]: print("Making predictions for the following 5 houses:")
      print(X.head())
      print("The predictions are")
      print(melbourne_model.predict(X.head()))
```

```
Making predictions for the following 5 houses:
   Rooms  Bathroom  Landsize  Lattitude  Longtitude
0      2       1.0     202.0   -37.7996    144.9984
1      2       1.0     156.0   -37.8079    144.9934
2      3       2.0     134.0   -37.8093    144.9944
3      3       2.0      94.0   -37.7969    144.9969
4      4       1.0     120.0   -37.8072    144.9941
The predictions are
[1480000. 1035000. 1465000.  850000. 1600000.]
```

Exercises: Your First Machine Learning Model

Step 1: Specify Prediction Target: (Tahmin Hedefi Belirtme)

"Satış fiyatı(sales price)" na karşılık gelen hedef değişkeni seçin. Bunu "y" adlı yeni bir değişkene kaydedin. İhtiyacınız olan sütunun adını bulmak için sütunların bir listesini yazdırmanız gerekir.

tahmin hedefinin adını bulmak için veri kümesindeki sütunların listesini yazdır

```
]:
# print the list of columns in the dataset to find the name of the prediction target
home_data.columns
```

```
[2]:
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

```
y =home_data.SalePrice

# Check your answer
step_1.check()
```

Aşağıdaki satırlar size bir ipucu veya çözüm gösterecektir.

step_1.hint()

step_1.solution()

Step 2: Create X

Şimdi, predictive feature'ları (tahmin özelliklerini) tutan X adında bir DataFrame oluşturacaksınız.

Orijinal verilerden yalnızca bazı sütunlar istediğiniz için, önce X'de istediğiniz sütunların adlarını içeren bir liste oluşturacaksınız.

Listede yalnızca aşağıdaki sütunları kullanacaksınız :

- LotArea
- YearBuilt
- 1stFlrSF
- 2ndFlrSF
- FullBath
- BedroomAbvGr
- TotRmsAbvGrd

Bu özellik listesini oluşturduktan sonra, modeli fit etmek için kullanacağınız DataFrame'i oluşturmak için kullanın.

[19]:
```
# Create the list of features below
feature_names = ['LotArea','YearBuilt','1stFlrSF','2ndFlrSF','FullBath','BedroomAbvGr','TotRmsAbvGrd']

# Select data corresponding to features in feature_names
X = home_data[feature_names]

# Check your answer
step_2.check()
```

Review Data

Bir model oluşturmadan önce, mantıklı göründüğünü doğrulamak için X 'e hızlı bir göz atın

[22]:
```
# Review data
# print description or statistics from X
print(X.describe())

# print the top few lines
print("\n",X.head())
```

```
            LotArea    YearBuilt     1stFlrSF     2ndFlrSF     FullBath  \
count   1460.000000  1460.000000  1460.000000  1460.000000  1460.000000
mean   10516.828082  1971.267808  1162.626712   346.992466     1.565068
std     9981.264932    30.202904   386.587738   436.528436     0.550916
min     1300.000000  1872.000000   334.000000     0.000000     0.000000
25%     7553.500000  1954.000000   882.000000     0.000000     1.000000
50%     9478.500000  1973.000000  1087.000000     0.000000     2.000000
75%    11601.500000  2000.000000  1391.250000   728.000000     2.000000
max   215245.000000  2010.000000  4692.000000  2065.000000     3.000000

       BedroomAbvGr  TotRmsAbvGrd
count   1460.000000   1460.000000
mean       2.866438      6.517808
std        0.815778      1.625393
min        0.000000      2.000000
25%        2.000000      5.000000
50%        3.000000      6.000000
75%        3.000000      7.000000
max        8.000000     14.000000

    LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
0     8450       2003       856       854         2             3
```

```
    LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
0     8450       2003       856       854         2             3
1     9600       1976      1262         0         2             3
2    11250       2001       920       866         2             3
3     9550       1915       961       756         1             3
4    14260       2000      1145      1053         2             4

   TotRmsAbvGrd
0             8
1             6
2             6
3             7
4             9
```

Step 3: Specify and Fit Model:

DecisionTreeRegressor oluştur ve iowa_model'e kaydet. Bu komutu çalıştırmak için **sklearn'de** ilgili import işlemini yaptığınızdan emin olun.

[23]:
```
from sklearn.tree import DecisionTreeRegressor
#modeli belirtin
#Model tekrarlanabilirliği için, modeli belirtirken random_state için sayısal bir değer belirleyin
iowa_model = DecisionTreeRegressor(random_state=1)

# Fit the model
iowa_model.fit(X,y)

# Check your answer
step_3.check()
```

Step 4: Make Predictions: (Tahmin Yapma)

Veri olarak **X**'i kullanarak modelin **predict** komutuyla tahminler yapın. Sonuçları **predictions** adı verilen bir değişkene kaydedin.

Notlar:

- `predict`: Regresyon, sınıflandırma, kümeleme gibi yöntemler kullanarak yapacağınız çalışmalarda tahmin edilen etiket bilgisini `predict` fonksiyonuyla elde edebilirsiniz. Sınıflandırma problemlerinde gözlemlerin sınıflara ait olma olasılıklarını elde etmek istiyorsanız `predict_proba` fonksiyonunu kullanmanız gerekiyor.

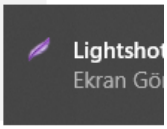
LinearRegression nesnesi ile modelimizi oluşturalım ve train datamız ile de modelimizi besleyelim.

```
model = LinearRegression()
model.fit(X_train, y_train)
```

Şimdi ise test datamızla modelimize tahmin ürettirelim.

```
prediction = model.predict(X_test)
print(prediction)
```

```
[ 55870.35248488 124217.47107547  53061.56678938 114854.85209045
55870.35248488 115791.11398896  63360.44767289  92384.56652643
63360.44767289 102683.44740994]
```



Model Validation(Model geçerliliği):

Bir model oluşturdunuz. Ama ne kadar iyi?

Bu derste, modelinizin kalitesini ölçmek için model doğrulamayı kullanmayı öğreneceksiniz.

Model kalitesini ölçmek, modellerinizi tekrar tekrar geliştirmenin anahtarıdır.

What is Model Validation: (Model Validation Nedir)

Oluşturduğunuz hemen hemen her modeli değerlendirmek isteyeceksiniz. Çoğu (hepsi olmasa da) uygulamada, model kalitesinin ilgili ölçüsü tahmini doğruluktur. Başka bir deyişle, modelin tahminleri gerçekte olanlara yakın olacak mı?

Birçok kişi tahmini doğruluğu ölçerken büyük bir hata yapar. "Training data" ile tahminler yaparlar ve bu tahminleri "Tranining data" daki hedef değerlerle karşılaştırırlar.

Bu yaklaşımla ilgili sorunu ve bir anda nasıl çözüleceğini göreceksiniz, ancak önce bunu nasıl yapacağımızı düşünelim.

Önce model kalitesini anlaşılabilir bir şekilde özetlemeniz gerekir. 10.000 ev için tahmini ve gerçek ev değerlerini karşılaştırırsanız, muhtemelen iyi ve kötü tahminlerin bir karışımını bulacaksınız. 10.000 tahmini ve gerçek değerin listesine bakmak anlamsız olacaktır. Bunu tek bir metrik olarak özetlemeliyiz.

Model kalitesini özetlemek için birçok metrik vardır, ancak **Mean Absolute Error**(ortalama mutlak hata) **(MAE** olarak da adlandırılır) olarak adlandırılan biriyle başlayacağız.

Bu metriği son kelimeyle başlayarak parçalayalım, hata.

Her ev için tahmin hatası:

```
error=actual-predicted
```

Yani, eğer bir ev 150.000 dolara mal olursa ve bunun 100.000 dolara mal olacağını tahmin ettiyseniz, hata 50.000 dolar. MAE metriği ile, her hatanın mutlak değerini alırız. Bu, her hatayı pozitif bir sayıya dönüştürür. Daha sonra bu mutlak hataların ortalamasını alırız. Bu bizim model kalitesinin ölçüsüdür. Düz İngilizce olarak, şu şekilde söylenebilir:

Ortalama olarak, tahminlerimiz yaklaşık X civarında.(On average, our predictions are off by about X.)

MAE'Yİ hesaplamak için önce bir modele ihtiyacımız var. Bu, code düğmesine tıklayarak inceleyebileceğiniz aşağıdaki gizli bir hücreye yerleştirilmiştir.

Bir modelimiz olduğunda, ortalama mutlak hatayı nasıl hesaplarız:

```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

```
434.71594577146544
```

```
# Data Loading Code Hidden Here
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 price 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.tree import DecisionTreeRegressor
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(X, y)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=Non
e,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

The Problem with "In-Sample" Scores("In-Sample(Örnek İçi)" Puanlarla İlgili Sorun)

Sadece hesapladığımız ölçü "örnek" puanı olarak adlandırılabilir. Hem modeli oluşturmak hem de değerlendirmek için tek bir "örnek" ev kullandık. İşte bu yüzden kötü.

Büyük emlak piyasasında, kapı rengi ev fiyatı ilgisiz olduğunu düşünün. Ancak, modeli oluşturmak için kullandığınız veri örneğinde, yeşil Kapılı tüm evler çok pahalıydı. Modelin işi, ev fiyatlarını tahmin eden desenleri(patterns) bulmaktır, bu yüzden bu deseni görecektir ve her zaman yeşil Kapılı evler için yüksek fiyatları tahmin edecektir.

Bu model eğitim verilerinden türetildiğinden, model eğitim verilerinde doğru görünecektir.

Ancak, model yeni veriler gördüğünde bu örüntü tutmazsa, model pratikte kullanıldığında çok yanlış olur.

Modellerin pratik değeri yeni veriler üzerinde tahmin yapmaktan geldiğinden, modeli oluşturmak için kullanılmayan veriler üzerindeki performansı ölçüyoruz.

Bunu yapmanın en basit yolu, bazı verileri model oluşturma sürecinden dışlamak ve daha önce görmediği veriler üzerindeki modelin doğruluğunu test etmek için bunları kullanmaktır.Bu verilere **validation data** denir.

Coding It

Scikit-learn Kütüphanesi, verileri iki parçaya bölmek için train_test_split işlevine sahiptir.

Bu verilerin bir kısmını modele uyacak şekilde eğitim verileri olarak kullanacağız ve diğer verileri mean_absolute_error hesaplamak için doğrulama verileri olarak kullanacağız.

Here is the code:

[41]:
```
from sklearn.model_selection import train_test_split
#verileri hem özellikler hem de hedef için eğitim ve doğrulama verilerine bölün
#Bölünmüş bir rasgele sayı üreteci dayanmaktadır. Sayısal bir değer sağlama
#random_state argümanı, her seferinde aynı bölünmeyi elde ettiğimizi garanti eder
#bu komut dosyasını çalıştırın.
train_X, val_X, train_y, val_y=train_test_split(X,y,random_state=0)
#Modeli tanımla
melbourne_model=DecisionTreeRegressor()
# fit model
melbourne_model.fit(train_X,train_y)
#doğrulama verilerinde öngörülen fiyatları alın
val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))
```

```
246802.63033873343
```

```
]:
```

Wow!

Örnek içi veriler için ortalama mutlak hatanız yaklaşık 500 dolardı. Örnek dışı 250.000 dolardan fazla.

Bu, neredeyse tamamen doğru olan bir model ile en pratik amaçlar için kullanılamayan bir model arasındaki farktır. Bir referans noktası olarak, doğrulama verilerindeki ortalama ev değeri 1,1 milyon dolar. Yani yeni verilerdeki hata ortalama ev değerinin dörtte biri kadardır.

Daha iyi özellikler veya farklı model türleri bulmak için deneme yapmak gibi bu modeli geliştirmenin birçok yolu vardır.

Exercises: Model Validation

Bir model yaptın. Bu alıştırmada modelinizin ne kadar iyi olduğunu test edeceksiniz.

Önceki alıştırmanın kaldığı kodlama ortamınızı ayarlamak için aşağıdaki hücreyi çalıştırın.

```
# Daha önce veri yüklemek için kullandığınız kod
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
# Okunacak dosya yolu
iowa_file_path='melb_data.csv'
home_data = pd.read_csv(iowa_file_path)
y = home_data.SalePrice
feature_columns = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[feature_columns]
# Modeli Belirle
iowa_model = DecisionTreeRegressor()
# Fit Model
iowa_model.fit(X, y)
print("First in-sample predictions:", iowa_model.predict(X.head()))
print("Actual target values for those homes:", y.head().tolist())
# Kod kontrolünü ayarlama
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex4 import *
print("Setup Complete")
```

Step 1: Split Your Data(Verilerinizi Bölün)

Verilerinizi bölmek için train_test_split işlevini kullanın.

Random_state = 1 argümanını verin, böylece kontrol fonksiyonları kodunuzu doğrularken ne bekleyeceğini bilir.

Geri çağırma, Özellikleri Veri Çerçevesi x yüklenir ve hedef yüklenir senin.

```
[13]:
#Import the train_test_split function and uncomment
from  sklearn.model_selection import train_test_split

# fill in and uncomment
train_X, val_X, train_y, val_y = train_test_split(X,y,random_state=1)

# Check your answer
step_1.check()
```

```
Correct
```

Step 2: Specify and Fit the Model

Bir DecisionTreeRegressor modeli oluşturun ve ilgili verilere uydurun. Modeli oluştururken random_state öğesini tekrar 1 olarak ayarlayın.

[15]:
```
# You imported DecisionTreeRegressor in your last exercise
# and that code has been copied to the setup code above. So, no need to
# import it again

# Specify the model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit iowa_model with the training data.
iowa_model.fit(train_X,train_y)

# Check your answer
step_2.check()
```

```
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
 262000.]
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
 262000.]
```

Step 3: Make Predictions with Validation data(Doğrulama verileriyle tahminler yapın)

[17]:
```
# Predict with all validation observations
val_predictions = val_predictions = iowa_model.predict(val_X)

# Check your answer
step_3.check()
```

Doğrulama verilerinden tahminlerinizi ve gerçek değerlerinizi inceleyin.

[21]:
```
# print the top few validation predictions
print(mean_absolute_error(val_y, val_predictions))
# print the top few actual prices from validation data
print(val_X.head())
```

```
29652.931506849316
      LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
258     12435       2001       963       829         2             3
267      8400       1939      1052       720         2             4
288      9819       1967       900         0         1             3
649      1936       1970       630         0         1             1
1233    12160       1959      1188         0         1             3

      TotRmsAbvGrd
258              7
267              8
288              5
649              3
1233             6
```

İçinde gördüğünüzden farklı olan ne fark edersiniz-örnek tahminler (bu sayfadaki en üst kod hücresinden sonra yazdırılır).

Doğrulama tahminlerinin neden örnek içi (veya eğitim) tahminlerinden farklı olduğunu hatırlıyor musunuz? Bu son dersten önemli bir fikir.

Step 4: Calculate the Mean Absolute Error in Validation Data(doğrulama verilerinde ortalama mutlak hatayı hesaplayın)

Step 4: Calculate the Mean Absolute Error in Validation Data

+ Code + Markdown

[]:
```
from sklearn.metrics import mean_absolute_error
val_mae = mean_absolute_error(val_y,val_predictions)

# uncomment following line to see the validation_mae
print(val_mae)


# Check your answer
step_4.check()
```

MAE sonucu iyi mi? Uygulamalar arasında geçerli olan değerlerin genel bir kuralı yoktur. Ancak bir sonraki adımda bu sayının nasıl kullanılacağını (ve geliştirileceğini) göreceksiniz.

Underfitting and Overfitting

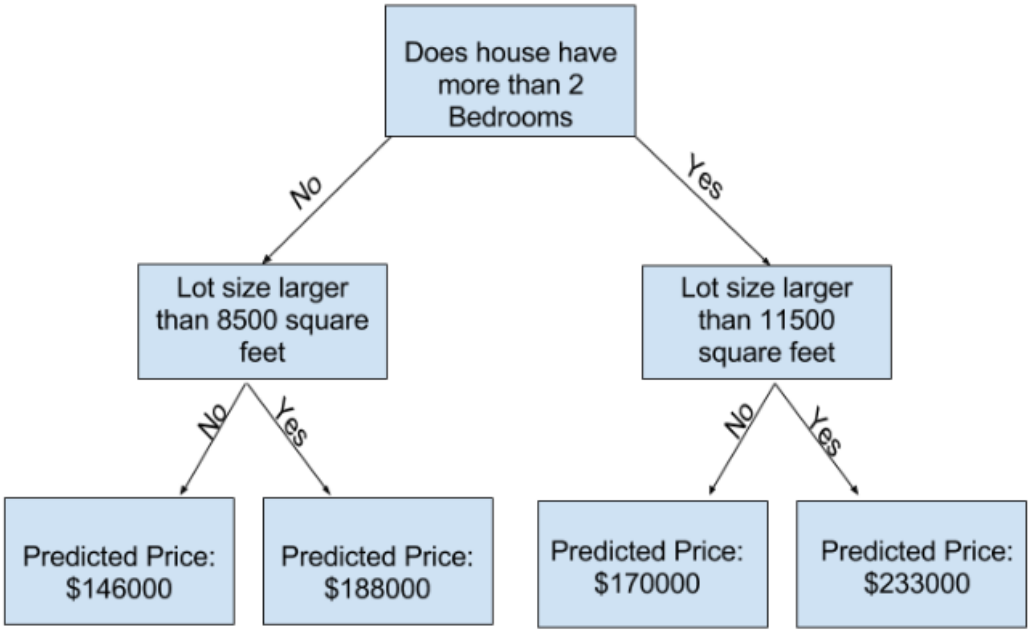
Bu adımın sonunda, uygun olmayan ve fazla uygunluk kavramlarını anlayacak ve modellerinizi daha doğru hale getirmek için bu fikirleri uygulayabileceksiniz.

Experimenting With Different Models

Artık model doğruluğunu ölçmenin güvenilir bir yoluna sahip olduğunuza göre, alternatif modelleri deneyebilir ve hangisinin en iyi tahminleri verdiğini görebilirsiniz. Peki modeller için hangi alternatifleriniz var?

Scikit-learn documentation'da(https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html)
karar ağacı modelinin birçok seçeneğe sahip olduğunu görebilirsiniz(uzun süre isteyeceğinizden veya ihtiyaç duyacağınızdan daha fazla).

En önemli seçenekler ağacın derinliğini belirler. Bu mikro kurstaki ilk dersten, bir ağacın derinliğinin, bir tahmine gelmeden önce kaç bölmenin yaptığının bir ölçüsü olduğunu hatırlayın. Bu nispeten derin olmayan bir ağaçtır.



Uygulamada, bir ağacın üst seviye (tüm evler) ve bir yaprak arasında 10 bölünmesi nadir değildir.

Uygulamada, bir ağacın üst seviye (tüm evler) ve bir yaprak arasında 10 bölünmesi nadir değildir. Ağaç derinleştikçe, veri kümesi daha az ev ile yapraklara dilimlenir. Bir ağacın yalnızca 1 bölünmesi varsa, verileri 2 gruba böler. Her grup tekrar bölünürse, 4 grup ev alırız. Bunların her birini tekrar bölmek 8 grup oluşturacaktır. Her seviyede daha fazla bölme ekleyerek grup sayısını ikiye katlamaya devam edersek, 10. seviyeye geldiğimizde 210 grup evimiz olacak. 1024 yaprak.

When we divide the houses amongst many leaves, we also have fewer houses in each leaf. Leaves with very few houses will make predictions that are quite close to those homes' actual values, but they may make very unreliable predictions for new data (because each prediction is based on only a few houses).

Bu, bir modelin eğitim verileriyle neredeyse mükemmel bir şekilde eşleştiği, ancak doğrulama ve diğer yeni verilerde zayıf olduğu overfitting adlı bir olgudur.

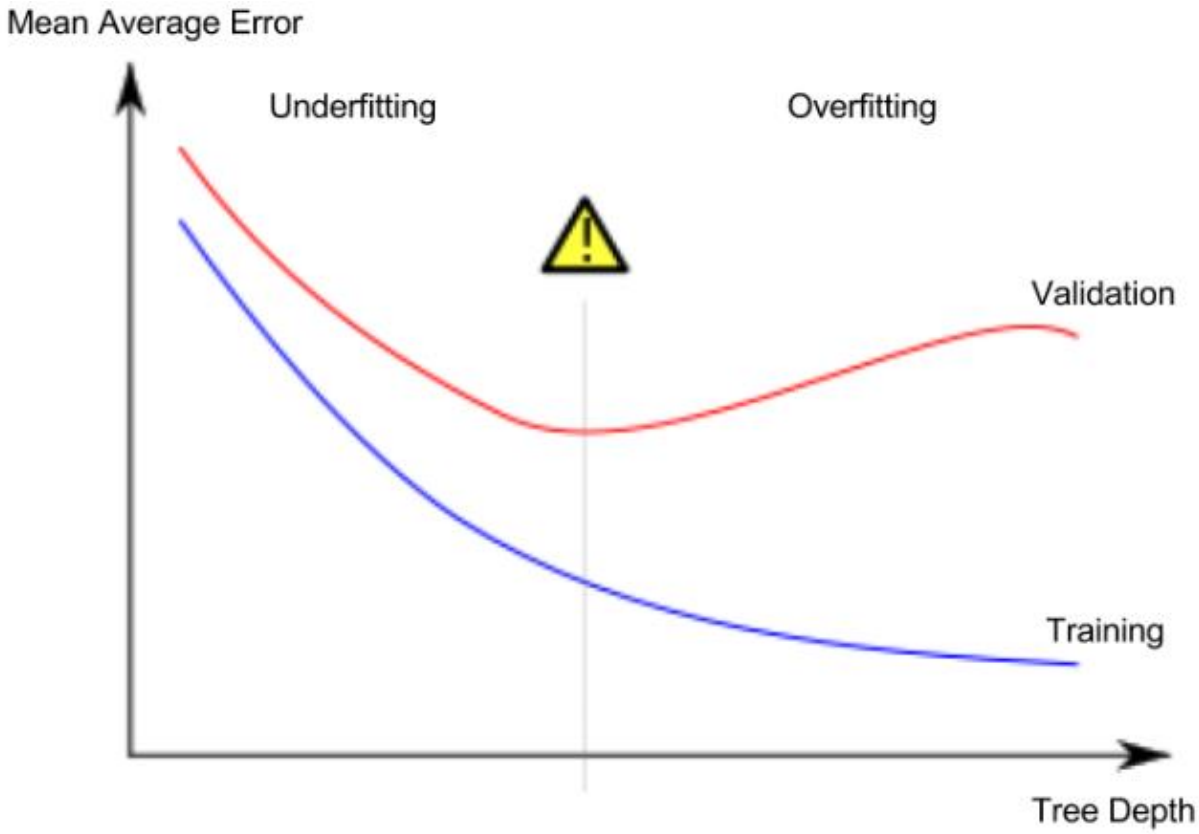
Ters tarafta, eğer ağacımızı çok yüzeysel yaparsak, evleri çok farklı gruplara ayırmaz.

Aşırı derecede, eğer bir ağaç evleri sadece 2 veya 4'e bölerse, her grup hala çok çeşitli evlere sahiptir.

Ortaya çıkan tahminler, eğitim verilerinde bile çoğu ev için çok uzak olabilir (ve aynı nedenden dolayı doğrulamada da kötü olacaktır). Bir model verilerde önemli ayrımlar(import distinctions) ve desenler(patterns) yakalamak için başarısız olduğunda, bu yüzden bile eğitim verilerinde(training data) kötü performans, bu underfitting denir.

Doğrulama verilerimizden tahmin ettiğimiz yeni veriler üzerindeki doğruluğu önemsediğimizden, underfitting ve overfitting arasındaki en etkili noktayı(sweet spot) bulmak istiyoruz.

Görsel olarak, (kırmızı) doğrulama eğrisinin düşük noktasını istiyoruz



Examples:

Ağaç derinliğini kontrol etmek için birkaç alternatif vardır ve birçoğu ağaçtaki bazı rotaların diğer rotalardan daha fazla derinliğe sahip olmasına izin verir.

Ancak max_leaf_nodes argümanı overfitting vs underfitting kontrol etmek için çok mantıklı bir yol sağlar.

Modelin yapmasına izin verdiğimiz daha fazla yaprak, yukarıdaki grafikteki underfitting alanından overfitting alanına daha fazla hareket ederiz.

Max_leaf_nodes için farklı değerlerden Mae puanlarını karşılaştırmaya yardımcı olmak için bir yardımcı program işlevi kullanabiliriz:

In [1]:

```
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_sta
te=0)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)
```

Veriler, daha önce gördüğünüz (ve daha önce yazdığınız) kodu kullanarak train_X, val_X, train_y ve val_y içine yüklenir.

Max_leaf_nodes için farklı değerlerle oluşturulmuş modellerin doğruluğunu karşılaştırmak için bir for-loop kullanabiliriz.

mae'yi farklı max_leaf_nodes değerleriyle karşılaştırın

```
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_no
des, 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, en uygun yaprak sayısıdır.

Sonuç:

Modeller şunlardan herhangi birine sahip olabilir.

- Overfitting: gelecekte tekrarlamayacak sahte pattern(desen)leri yakalamak , daha az doğru tahminlere yol açmak veya
- Underfittin: alakalı pattern'leri yakalayamama, yine daha az doğru tahminlere yol açma.

Bir aday modelin doğruluğunu(accuracy) ölçmek için model eğitiminde(train) kullanılmayan doğrulama(validation) verilerini kullanıyoruz. Bu, birçok aday modeli denememize ve en iyisini elde etmemizi sağlar.

Overfitting:

> *Burada **eğitim seti** geçmiş 10 yılın soruları, **model** sınavın geçmiş senelere benzeyeceğini düşünmeniz, **test seti** hiç görmediğimiz istatistik sınavı, **başarı kriteri** aldığınız not. Sınav soruları beklendiğiniz gibi gelmez de kötü not alırsanız bu olaya **overfitting** denir.*

Underfitting:

> *Hoca bu konuyu sormaz şu konuyu sormaz diye diye kafanıza göre konuları çalışmaktan vazgeçip düşük not alırsanız buna da underfitting denir.*
>
> *Diğer bir deyişle eğer modelimizi **eğitim (training)** veri seti üzerinde çok basit olarak kurguladıysak hiç görmediğimiz test verisi üzerinde başarısız tahminler (sallama) yaparız ve gerçek değerle tahmin ettiğimiz değer arasındaki fark çok olur.*

Exercise: Underfitting and Overfitting

Tekrarlamak:

İlk modelinizi oluşturdunuz ve şimdi daha iyi tahminler yapmak için ağacın boyutunu optimize etme zamanı. Önceki adımı bıraktığınız yerde kodlama ortamınızı ayarlamak için bu hücreyi çalıştırın.

```
# Code you have previously used to load data
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor


# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)

# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE: {:,.0f}".format(val_mae))

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex5 import *
print("\nSetup complete")
```

```
Validation MAE: 29,653

Setup complete
```

Get_mae fonksiyonunu kendiniz yazabilirsiniz. Şimdilik tedarik edeceğiz. Bu, bir önceki derste okuduğunuz işlevle aynıdır. Aşağıdaki hücreyi çalıştırmanız yeterlidir.

[ ]:

```
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)
```

Step 1: Compare Different Tree Sizes

Bir dizi olası değerden max_leaf_nodes için aşağıdaki değerleri çalıştıran bir döngü yazın.

Her max_leaf_nodes değerinde get_mae işlevini çağırın. Çıktıyı, verilerinizde en doğru modeli veren max_leaf_nodes değerini seçmenize izin verecek şekilde saklayın.

```
candidate_max_leaf_nodes = [5, 25, 50, 100, 250, 500]
# Candidate_max_leaf_nodes'dan ideal ağaç boyutunu bulmak için döngü yaz
for max_leaf_nodes in candidate_max_leaf_nodes:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes:{0} \t\t Mean Absolute Error:{1}".format(max_leaf_nodes, my_mae))
# Max_leaf_nodes en iyi değerini saklayın (5, 25, 50, 100, 250 veya 500 olacaktır)
best_size=100


# Check your answer
step_1.check()
```

```
Max leaf nodes:5          Mean Absolute Error:35044.51299744237
Max leaf nodes:25         Mean Absolute Error:29016.41319191076
Max leaf nodes:50         Mean Absolute Error:27405.930473214907
Max leaf nodes:100        Mean Absolute Error:27282.50803885739
Max leaf nodes:250        Mean Absolute Error:27893.822225701646
Max leaf nodes:500        Mean Absolute Error:29454.18598068598
```

Step 2: Fit Model Using All Data(Modeli tüm verileri kullanarak sığdır)

En iyi ağaç boyutunu biliyorsun. Bu modeli pratikte deploy edecek olsaydınız, tüm verileri kullanarak ve bu ağaç boyutunu koruyarak daha da doğru hale getirirsiniz.

Yani, tüm modelleme kararlarınızı verdiğiniz için doğrulama verilerini saklamanız gerekmez.

```
19]:
# Fill in argument to make optimal size and uncomment
final_model = DecisionTreeRegressor(max_leaf_nodes=best_tree_size,random_state=1)

# fit the final model and uncomment the next two lines
final_model.fit(X,y)

# Check your answer
step_2.check()
```

Bu modeli ayarladınız ve sonuçlarınızı geliştirdiniz. Ancak hala modern makine öğrenimi standartlarına göre çok karmaşık olmayan Decision Tree modellerini kullanıyoruz. Bir sonraki adımda, modellerinizi daha da geliştirmek için Random Forest kullanmayı öğreneceksiniz.

Random Forests:

Giriş:

Decision Tree sizi zor bir kararla baş başa bırakır. Çok sayıda yapraklı derin bir ağaç, her tahmin, yaprağındaki sadece birkaç evden gelen tarihsel verilerden geldiğinden fazla olacaktır. Ancak, az yapraklı sığ bir ağaç kötü performans gösterecektir, çünkü ham verilerdeki birçok farklılığı yakalayamaz.

Günümüzün en sofistike modelleme teknikleri bile, underfitting ve overfitting arasındaki bu gerilim ile karşı karşıyadır.

Ancak, birçok model daha iyi performans sağlayabilecek akıllı fikirlere sahiptir. Örnek olarak **Random Forest'a** bakacağız.

Random Forest birçok ağaç kullanır ve her bileşen ağacının tahminlerini ortalayarak bir tahmin yapar.

Genellikle tek bir karar ağacından çok daha iyi tahmin doğruluğu(predictive accuracy) vardır ve varsayılan parametrelerle iyi çalışır.

Modellemeye devam ederseniz, daha iyi performansa sahip daha fazla model öğrenebilirsiniz, ancak bunların çoğu doğru parametreleri almaya duyarlıdır.

Example

Verileri yüklemek için gereken kodu zaten birkaç kez gördünüz. Veri yüklemenin sonunda aşağıdaki değişkenler bulunur:

• train_X

• val_X

• train_y

• val_y

In [1]:

```
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
melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                      'YearBuilt', 'Lattitude', 'Longtitude']
X = melbourne_data[melbourne_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and target
# The split is based on a random number generator. Supplying a numeric value to
# the random_state argument guarantees we get the same split every time we
# run this script.
train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

scikit-learn kütüphanesinde decision tree modeli oluşturduğumuz gibi bu kez random forest modeli oluşturacağız. – **DecisionTreeRegressor** yerine **RandomTreeRegressor** kullanacağız.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/fores
t.py:245: FutureWarning: The default value of n_estimators wil
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)


202888.18157951365
```

Sonuç:

Daha da iyileştirilmesi muhtemeldir, ancak bu 250.000 olan en iyi karar ağacı hatası üzerinde büyük bir gelişmedir.

Single decision tree'nin maksimum derinliğini değiştirdiğimiz gibi Random Forest'ın da performansını değiştirmenize izin veren parametreler var.

Ancak Random Forest modellerinin en iyi özelliklerinden biri, bu ayarlama olmadan bile genellikle makul bir şekilde çalışmasıdır.

Yakında, doğru parametrelerle iyi ayarlandığında daha iyi performans sağlayan (ancak doğru model parametrelerini elde etmek için biraz beceri gerektiren) XGBoost modelini öğreneceksiniz.

Exercises: Random Forest

Şimdiye kadar yazdığımız kod:

```
# Code you have previously used to load data
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor


# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)

# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE when not specifying max_leaf_nodes: {:,.0f}".format(val_mae))

# Using best value for max_leaf_nodes
iowa_model = DecisionTreeRegressor(max_leaf_nodes=100, random_state=1)
iowa_model.fit(train_X, train_y)
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE for best value of max_leaf_nodes: {:,.0f}".format(val_mae))


# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex6 import *
print("\nSetup complete")
```

```
Validation MAE when not specifying max_leaf_nodes: 29,653
Validation MAE for best value of max_leaf_nodes: 27,283

Setup complete
```

Exercises

Veri bilimi her zaman bu kadar kolay değildir. Ancak Decision Tree'yi Random Forest ile değiştirmek kolay bir kazanç olacaktır.

Step 1: Use a Random Forest

[5]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
# Modeli tanımlayın. Random_state öğesini 1 olarak ayarla
rf_model = RandomForestRegressor(random_state=1)

# fit your model
rf_model.fit(train_X,train_y)
rf_model_predictions=rf_model.predict(val_X)
# Calculate the mean absolute error of your Random Forest model on the validation data
rf_val_mae =mean_absolute_error(val_y,rf_model_predictions)

print("Validation MAE for Random Forest Model: {}".format(rf_val_mae))

# Check your answer
step_1.check()
```

```
Validation MAE for Random Forest Model: 21857.15912981083
```

Şimdiye kadar, projenizin her adımında belirli talimatları izlediniz. Bu, temel fikirleri öğrenmeye ve ilk modelinizi oluşturmaya yardımcı oldu, ancak şimdi işleri kendi başınıza denemek için yeterince bilgi sahibisiniz.

Machine Learning yarışmaları, bağımsız olarak bir machine learning projesinde gezinirken kendi fikirlerinizi denemek ve daha fazla bilgi edinmek için harika bir yoldur.

Exercises: Machine Learning Competitions

Introduction

Makine öğrenimi yarışmaları, veri bilimi becerilerinizi geliştirmenin ve ilerlemenizi ölçmenin harika bir yoludur.

Bu alıştırmada, bir Kaggle yarışması için tahminler oluşturacak ve sunacaksınız.

Bu notebook'daki adımlar:

• Tüm verilerinizle Random Forest modeli oluşturun. (X ve y)

• Target(hedef) içermeyen "test" verilini okuyun. Random Forest modelinizle test verilerindeki ev fiyatlarını tahmin edin.

• Bu tahminleri yarışmaya gönderin ve puanınızı görün.

• İsteğe bağlı olarak, feature'lar ekleyerek veya modelinizi değiştirerek modelinizi geliştirip geliştiremeyeceğinizi görmek için tekrar deneyin. Daha sonra bunun rekabet lider panosunda nasıl etkilediğini görmek için yeniden gönderebilirsiniz.

Şimdiye kadar yazdığımız kod:

```
# Code you have previously used to load data
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Set up code checking
import os
if not os.path.exists('../input/train.csv'):
    os.symlink('../input/home-data-for-ml-course/train.csv', '../input/train.csv')
    os.symlink('../input/home-data-for-ml-course/test.csv', '../input/test.csv')
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex7 import *

# Path of the file to read. We changed the directory structure to simplify submitting to a competi
iowa_file_path = '../input/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbv
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)
```

```
# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE when not specifying max_leaf_nodes: {:,.0f}".format(val_mae))

# Using best value for max_leaf_nodes
iowa_model = DecisionTreeRegressor(max_leaf_nodes=100, random_state=1)
iowa_model.fit(train_X, train_y)
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE for best value of max_leaf_nodes: {:,.0f}".format(val_mae))

# Define the model. Set random_state to 1
rf_model = RandomForestRegressor(random_state=1)
rf_model.fit(train_X, train_y)
rf_val_predictions = rf_model.predict(val_X)
rf_val_mae = mean_absolute_error(rf_val_predictions, val_y)

print("Validation MAE for Random Forest Model: {:,.0f}".format(rf_val_mae))
```

```
Validation MAE when not specifying max_leaf_nodes: 29,653
Validation MAE for best value of max_leaf_nodes: 27,283
Validation MAE for Random Forest Model: 21,857
```

Creating a Model For the Competition

Random Forest modeli oluşturun ve tüm X ve y ile modeli eğitin.

In [2]:
```
# To improve accuracy, create a new Random Forest model which you will train on all training data
rf_model_on_full_data = RandomForestRegressor(random_state=1)

# fit rf_model_on_full_data on all data from the training data
rf_model_on_full_data.fit(X, y)
```

Out[2]:
```
RandomForestRegressor(random_state=1)
```

Make Predictions

"Test" verileri dosyasını okuyun. Tahmin yapmak için modelinizi uygulayın.

In [3]:

```
# path to file you will use for predictions
test_data_path = '../input/test.csv'

# read test data file using pandas
test_data = pd.read_csv(test_data_path)

# create test_X which comes from test_data but includes only the columns you used for prediction.
# The list of columns is stored in a variable called features
test_X = test_data[features]
# make predictions which we will submit.
test_preds = rf_model_on_full_data.predict(test_X)

# The lines below shows how to save predictions in format used for competition scoring
# Just uncomment them.
output = pd.DataFrame({'Id': test_data.Id,
                       'SalePrice': test_preds})

output.to_csv('submission.csv', index=False)
```

Modelinizi geliştirmenin birçok yolu vardır ve deneme yapmak bu noktada öğrenmenin harika bir yoludur. Modelinizi geliştirmenin en iyi yolu özellikler eklemektir. Sütun listesine bakın ve konut fiyatlarını nelerin etkileyebileceğini düşünün. Bazı özellikler, eksik değerler veya sayısal olmayan veri türleri gibi sorunlar nedeniyle hatalara neden olur.

KAYNAKLAR

- Kaggle – Intro to Machine Learnin Course
  https://www.kaggle.com/learn/intro-to-machine-learning
- https://medium.com/data-science-tr/overfitting-underfitting-cross-validation-b47dfda0cf4e
- http://www.veridefteri.com/2017/11/23/scikit-learn-ile-veri-analitigine-giris/