|  |
| --- |
| اکتشاف داده های پایه (Basic Data Exploration) |
| **الف) استفاده از pandas برای آشنایی با داده ها**  import pandas as pd  home\_file\_path=”../input/melb\_data.csv” #ست کردن مسیر فایل داده  home\_data=pd.read\_csv(home\_file\_path) # خواندن فایل داده  home\_data.describe() # مشاهده توصیفی از داده ها  **ب) تفسیر توصیف داده ها**  این توصیف شامل موارد زیر است:  count, mean, std, 25%, 50%, 75%, max |
| اولین مدل یادگیری ماشین توسط شما (Your First Machine Learning Model) |
| **الف) انتخاب داده برای مدلسازی**  برای انتخاب فیلدهای مورد استفاده در مدلسازی داده­ها، بهتر است اول با استفاده از ویژگی columns مجموعه داده، نام تمام فیلدها/ستونهای مجموعه داده را مشاهده کنیم. با استفاده از خروجی متد describe می­توانیم یک دید کلی نسبت به داده­ها و برخی نقایص آنها داشته باشیم.  import pandas as pd  home\_file\_path=”../input/home\_data.csv” #ست کردن مسیر فایل داده  home\_data=pd.read\_csv(home\_file\_path) # خواندن فایل داده  home\_data.describe() # مشاهده توصیفی از داده ها  print(home\_data.columns) # مشاهده نام همه فیلدها/ستونهای مجموعه داده  اگر در مجموعه داده دارای missing value باشیم، بعنوان یک راه حل ساده و ابتدایی می­توان رکوردهای حاوی این missing value ها را با استفاده از متد dropna حذف کرد. برای در خاطر ماندن آنرا بصورت drop not available در نظر بگیرید.  home\_data=home\_data.dropna(axis=0) #حذف رکوردهایی که برخی ستونهای آن فاقد مقدار هستند  **ب) انتخاب هدف/فیلد پیش­بینی**  فیلدی که می­خواهیم آنرا در مجموعه داده مورد پیش بینی قرار دهیم، در متغیر y قرار می دهیم. (نام این فیلد را از میان فهرستی که ویژگی columns در اختیارمان قرار می­دهد انتخاب می­کنیم.)  y=home\_data.Price  **پ) انتخاب ویژگی­ها**  در عمل، برای پیش­بینی مقادیر فیلد هدف نیاز به مقادیر همه فیلدهای دیگر نیست. با توجه به تجربه یا استفاده از روش­های مربوطه، نام فیلدهای مورد نظر برای پیش­بینی فیلد هدف انتخاب و در متغیر X قرار می­دهیم.  home\_features=[‘Rooms’,’BathSize’,’LandSize’]  X=home\_data[home\_features]  X.describe() # مشاهده توصیفی از داده ها  X.head() # مشاهده پنج رکورد اول داده ها  **ت) ساختن مدل**  برای مدلسازی از کتابخانه scikit-learn استفاده می­شود که در کد نویسی sklearn بکار می­بریم.  گام­های ساخت یک مدل عبارتند از:   * *تعریف (define)* : مدل از چه نوعی خواهد بود؟ درخت تصمیم؟ نوع دیگری از مدل؟ برخی پارامترهای دیگر برای انتخاب نوع مدل باید مشخص شوند. * *سازگاری/شایستگی (fit)* : گرفتن الگوها از داده های ارائه شده. این قلب مدل سازی است. * *پیش­بینی (predict)* : درست همان چیزی که به نظر می رسد. * *ارزیابی (evaluate)* : تعیین کنید پیش‌بینی‌های مدل چقدر دقیق هستند.   from sklearn.tree import DecisionTreeRegressor  home\_model=DecisionTreeRegressor(random\_state=1) # انتخاب مدل  home\_model.fit(X,y) # ایجاد مدل از داده ها  print(home\_model.predict(X)) # مشاهده پیش بینی  البته در کد بالا در متد predict باید یک مجموعه داده دیگر ارائه می­دادیم تا پیش­بینی شوند نه خود همان مجموعه­ای که یادگیری روی آن­ها انجام شده است. |
| صحت سنجی مدل (Model Validation) |
| شما می خواهید تقریباً هر مدلی را که می سازید ارزیابی کنید. در اکثر برنامه‌ها (اگرچه نه همه)، معیار مربوط به کیفیت مدل، دقت پیش‌بینی است. به عبارت دیگر، آیا پیش‌بینی‌های مدل به آنچه در واقع اتفاق می‌افتد نزدیک است؟  بسیاری از مردم هنگام اندازه گیری دقت پیش بینی اشتباه بزرگی مرتکب می شوند. آنها با داده های آموزشی خود پیش بینی می کنند و آن پیش بینی ها را با مقادیر هدف در داده های آموزشی مقایسه می کنند. ابتدا باید کیفیت مدل را به روشی قابل فهم خلاصه کنید. اگر مقادیر پیش بینی شده و واقعی قیمت خانه را برای 10000 خانه مقایسه کنید، احتمالاً ترکیبی از پیش بینی های خوب و بد را خواهید یافت. نگاه کردن به لیستی از 10000 مقدار پیش بینی شده و واقعی بی معنی خواهد بود. ما باید این را در یک متریک خلاصه کنیم.  معیارهای زیادی برای خلاصه کردن کیفیت مدل وجود دارد، اما ما با یکی به نام میانگین خطای مطلق (که MAE نیز نامیده می شود) شروع می کنیم.  error=abs(actual−predicted)  با متریک MAE، قدر مطلق هر خطا را می گیریم. این هر خطا را به یک عدد مثبت تبدیل می کند. سپس میانگین آن خطاهای مطلق را می گیریم. این معیار ما برای کیفیت مدل است. به زبان ساده می توان گفت: به طور متوسط، پیش‌بینی‌های ما چقدر با X تفاوت دارد.  from sklearn.metrics import mean\_absolute\_error  predicted\_home\_price=home\_model.predict(X)  mean\_absolute\_error(y, predicted\_home\_price)  مشکل با ارزش داخل نمونه (The Problem with "In-Sample" Scores)  اندازه گیری که ما انجام دادیم را می­توان نمره "در نمونه" نامید. ما از یک "نمونه" خانه برای ساختن مدل و ارزیابی آن استفاده کردیم. دلیل نامناسب بودن این روش را شرح می­دهیم.  تصور کنید که در بازار بزرگ املاک، رنگ درب به قیمت خانه ارتباطی ندارد. با این حال، در نمونه داده‌هایی که برای ساخت مدل استفاده کردید، همه خانه‌های دارای درهای سبز بسیار گران بودند. کار مدل پیدا کردن الگوهایی است که قیمت خانه ها را پیش بینی می­کند، لذا این الگو را می بیند و همیشه قیمت های بالایی را برای خانه هایی با درهای سبز پیش بینی می­کند. از آنجایی که این الگو از داده های آموزشی مشتق شده است، مدل در داده های آموزشی دقیق ظاهر می شود. اما اگر در داده‌های جدید این الگو برقرار نباشد، هنگام ارزیابی مدل، مدل بسیار نادرست خواهد بود.  از آنجایی که ارزش عملی مدل‌ها از پیش‌بینی داده‌های جدید ناشی می‌شود، عملکرد مدل را روی داده‌هایی که برای ساخت مدل استفاده نشده‌اند، اندازه‌گیری می‌کنیم. ساده ترین راه برای انجام این کار این است که برخی از داده ها را از فرآیند ساخت مدل حذف کنید و سپس از آنها برای آزمایش دقت مدل بر روی داده هایی که قبلاً ندیده استفاده کنید. به این داده ها داده های اعتبار سنجی (validation data) می­گویند.  کتابخانه scikit-learn یک تابع به نام train\_test\_split دارد تا داده ها را به دو قسمت تقسیم کند. ما از برخی از آن داده‌ها به عنوان داده‌های آموزشی برای ایجاد مدل استفاده کرده، از داده‌های دیگر به‌عنوان داده اعتبارسنجی برای محاسبه mean\_absolute\_error استفاده می‌کنیم.  from sklearn.model\_selection import train\_test\_split  train\_X,val\_X,train\_y,val\_y=train\_test\_split(X,y,random\_state=0)  home\_model=DecisionTreeRegressor(random\_state=1)  home\_model.fit(train\_X,train\_y)  val\_predictions=home\_model.predict(val\_X)  print(mean\_absolute\_error(val\_y,val\_predictions))  راه های زیادی برای بهبود این مدل وجود دارد، مانند آزمایش برای یافتن ویژگی های بهتر یا انواع مدل های مختلف. |
| (Underfitting and Overfitting) |
| مدل خود را برای عملکرد بهتر تنظیم کنید.  **آزمایش با مدل های مختلف**  اکنون که روش قابل اعتمادی برای اندازه گیری دقت مدل دارید، می توانید مدل های جایگزین را آزمایش کنید و ببینید کدام بهترین پیش بینی ها را ارائه می دهد. اما چه جایگزین هایی برای مدل ها دارید؟  شما می توانید در مستندات scikit-learn ببینید که مدل درخت تصمیم گزینه های زیادی دارد (بیشتر از چیزی که برای مدت طولانی می خواهید یا نیاز دارید). مهمترین گزینه ها عمق درخت را تعیین می کنند. از اولین درس در این دوره به یاد بیاورید که عمق یک درخت اندازه گیری تعداد انشعاب/شاخه­های آن قبل از رسیدن به یک پیش بینی است. این درخت نسبتا کم عمق است    در عمل، داشتن 10 انشعاب بین سطح بالایی (همه خانه ها) و یک برگ، غیر معمول نیست. با عمیق‌تر شدن درخت، مجموعه داده‌ها به برگ‌هایی با خانه‌های کمتر تقسیم می‌شوند. اگر درختی فقط 1 انشعاب داشته باشد، داده ها را به 2 گروه تقسیم می­کند. اگر هر گروه دوباره تقسیم شود، 4 گروه خانه خواهیم داشت. با تقسیم مجدد هر یک از آنها 8 گروه ایجاد می شود. اگر تعداد گروه ها را با اضافه کردن تقسیم های بیشتر در هر سطح دو برابر کنیم، 210 گروه از خانه­ها خواهیم داشت. در سطح 10 دارای 1024 برگ خواهیم بود.  وقتی خانه ها را بین چندین برگ تقسیم می­کنیم، تعداددخانه های کمتری نیز در هر برگ داریم. برگ‌هایی که خانه‌های بسیار کمی دارند، پیش‌بینی‌هایی می‌کنند که کاملاً به مقادیر واقعی آن خانه‌ها نزدیک است، اما ممکن است پیش‌بینی‌های بسیار غیرقابل اعتمادی برای داده‌های جدید انجام دهند (زیرا هر پیش‌بینی فقط بر اساس چند خانه است). این پدیده ای است به نام overfitting، که در آن یک مدل تقریباً با داده های آموزشی مطابقت دارد، اما در اعتبارسنجی و سایر داده های جدید ضعیف عمل می­کند. از طرف دیگر، اگر درختمان را خیلی کم عمق کنیم، خانه ها را به گروه های خیلی متمایز تقسیم نمی کند.  در نهایت، اگر درختی خانه‌ها را تنها به 2 یا 4 گروه تقسیم کند، هر گروه همچنان دارای تنوع زیادی از خانه‌ها است. پیش‌بینی‌های حاصل ممکن است برای اکثر خانه‌ها دور از دسترس باشد، حتی در داده‌های آموزشی (و به همین دلیل در اعتبارسنجی نیز بد خواهد بود). زمانی که یک مدل نتواند تمایزات و الگوهای مهم را در داده‌ها ثبت کند، حتی در آموزش داده‌ها نیز ضعیف عمل می‌کند، که به آن underfitting می‌گویند.  از آنجایی که به دقت پیش­بینی در داده‌های جدید اهمیت می‌دهیم، که می­توان آن را از روی داده‌های اعتبارسنجی خود تخمین بزنیم، می‌خواهیم نقطه مناسب بین underfitting و overfitting را پیدا کنیم. از نظر بصری، نقطه پایین منحنی اعتبارسنجی (قرمز) را در شکل زیر می خواهیم.    **مثال**  چند گزینه برای کنترل عمق درخت وجود دارد، و بسیاری از آنها اجازه می­دهند که برخی از مسیرها از طریق درخت عمق بیشتری نسبت به مسیرهای دیگر داشته باشند. اما آرگومان max\_leaf\_nodes راه بسیار معقولی را برای کنترل underfitting در مقابل overfitting ارائه می دهد. هرچه تعداد برگ های بیشتری را به مدل اجازه دهیم، بیشتر از ناحیه underfitting در نمودار بالا به ناحیه overfitting حرکت می­کنیم.  ما می توانیم از یک تابع بعنوان ابزاری برای کمک به مقایسه امتیازات MAE ناشی از مقادیر مختلف برای max\_leaf\_nodes استفاده کنیم:  from sklearn.metrics import mean\_absolute\_error  from sklearn.tree import DecisionTreeRegressor  def get\_mae(mad\_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)  می‌توانیم از یک حلقه for برای مقایسه دقت مدل‌های ساخته شده با مقادیر مختلف برای max\_leaf\_nodes استفاده کنیم.  # 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  با توجه به خروجی بدست آمده، انتخاب مقدار 500 برای max\_leaf\_nodes انتخابی بهینه است.  **نتیجه گیری**  مدل ها ممکن است از یکی از این موارد رنج ببرند:   * overfitting: ثبت الگوهای جعلی که در آینده تکرار نخواهند شد، که منجر به پیش بینی های کمتر دقیق می شود. * underfitting: شکست در یافتن الگوهای مربوطه موجود، که منجر به پیش بینی های کمتر دقیق می شود.   ما داده‌های اعتبارسنجی را که در آموزش مدل استفاده نمی‌شوندبکار می­گیریم تا دقت مدل کاندید را اندازه‌گیری کنیم. این به ما امکان می دهد بسیاری از مدل های کاندید را امتحان کنیم و بهترین را انتخاب کنیم. |
| جنگل­های تصادفی (Random Forests) |
| **مقدمه**  درختان تصمیم­گیری شما را با یک تصمیم دشوار رها می­کنند. یک درخت عمیق با تعداد زیادی برگ overfit می شود زیرا هر پیش بینی از داده های تاریخی فقط از چند خانه در برگ آن حاصل می شود. اما یک درخت کم عمق با چند برگ عملکرد ضعیفی خواهد داشت زیرا نمی تواند تمایزات زیادی را در داده های خام به دست آورد.  حتی پیچیده‌ترین تکنیک‌های مدل‌سازی امروزی نیز با این تنش بین overfitting و underfitting مواجه هستند. اما، بسیاری از مدل ها ایده های هوشمندانه ای دارند که می تواند منجر به عملکرد بهتر شود. به عنوان مثال به جنگل تصادفی نگاه خواهیم کرد.  جنگل تصادفی از درختان زیادی استفاده می‌کند و با میانگین‌گیری پیش‌بینی‌های هر درخت، پیش‌بینی می‌کند. به طور کلی دقت پیش‌بینی بسیار بهتری نسبت به یک درخت تصمیم دارد و با پارامترهای پیش‌فرض به خوبی کار می‌کند. اگر به مدل‌سازی ادامه دهید، می‌توانید مدل‌های بیشتری  را با عملکرد بهتر یاد بگیرید، اما بسیاری از آن‌ها به دریافت پارامترهای مناسب حساس هستند.  **مثال**  قبلاً کد بارگیری داده ها را چند بار دیده اید. در پایان بارگذاری داده ها، متغیرهای train\_X، val\_X، train\_y و val\_y را داشتیم:  import pandas as pd  # Load data  melbourne\_file\_path=”../input/melb\_data.csv”  melbourne\_data = pd.read\_csv(melbourne\_file\_path)  # Filter rows with missing values  melbourne\_data = melbourne\_data.dropna(axis=0)  # Choose the target and features  y = melbourne\_data.Price  melbourne\_features=[‘Rooms’, ‘Bathroom’, ‘Landsize’, ‘BuildingArea’, ‘YearBuilt’,  ‘Lattitude’, ‘Longtitude’]  X = melbourne\_data[melbourne\_features]  # 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)  ما یک مدل جنگل تصادفی مشابه نحوه ساخت درخت تصمیم در scikit-learn می سازیم - این بار از کلاس RandomForestRegressor به جای DecisionTreeRegressor استفاده می­کنیم.  from sklearn.ensemble import RandomForestRegressor  from sklearn.metrics import mean\_absolute\_error  forest\_model = RandomForestRegessor(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) # 191669.7536453626  **نتیجه گیری**  احتمالاً فضایی برای بهبود بیشتر وجود دارد، اما این یک پیشرفت بزرگ نسبت به بهترین خطای درخت تصمیم 250000 است. پارامترهایی وجود دارند که به شما امکان می دهند عملکرد جنگل تصادفی را به همان اندازه که ما حداکثر عمق درخت تصمیم را تغییر دادیم، تغییر دهید. اما یکی از بهترین ویژگی های مدل های Random Forest این است که به طور کلی حتی بدون این تنظیم به طور معقول کار می کنند. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |