**فصل ١١. ارزیابی مدل**

**١١.٠. مقدمه**

در این فصل به بررسی استراتژی‌های ارزیابی کیفیت مدل‌های ایجاد شده از طریق الگوریتم­های یادگیری می‌پردازیم. شاید عجیب به نظر برسد که در مورد ارزیابی مدل، قبل از بحث در مورد چگونگی ایجاد آنها صحبت کنیم اما روشی برای رفع این سردرگمی وجود دارد. مدل­ها تنها به اندازه‌ی کیفیت پیش‌بینی­هایشان مفید هستند و بنابراین اساساً هدف ما ایجاد مدل(که آسان است) نیست، بلکه ایجاد مدل­های با کیفیت بالا (که سخت است) است. بنابراین، قبل از اینکه هزاران الگوریتم یادگیری را بررسی کنیم، ابتدا نحوه ارزیابی مدل­های تولید شده توسط آن­ها را بررسی می­کنیم.

**١١.١. مدل­های اعتبار سنجی متقاطع (متقابل)**

**مسئله**

می­خواهید ارزیابی کنید که مدل شما در دنیای واقعی چقدر خوب کار خواهد کرد.

**راه حل**

یک خط انتقال داده ایجاد کنید که داده­ها را پیش‌پردازش می­کند، مدل را آموزش می­دهد و سپس آن را با استفاده از اعتبارسنجی متقابل ارزیابی می­کند:



****

**بحث**

در نگاه اول، ارزیابی مدل­های یادگیری تحت نظارت ممکن است ساده به نظر برسد به این صورت که یک مدل را آموزش دهید و سپس با استفاده از معیارهای عملکرد (دقت، مربع خطاها و غیره) میزان صحت عملکرد آن را بررسی کنید. با این حال، این رویکرد اساساً ناقص است. اگر مدلی را با استفاده از داده­های خود آموزش دهیم و سپس ارزیابی کنیم که چقدر روی آن داده­ها خوب عمل کرده است، به هدف مورد نظر خود نمی­رسیم. هدف ما ارزیابی این نیست که مدل تا چه حد بر روی داده­های آموزشی ما خوب عمل می­کند، بلکه این است که تا چه حد بر روی داده­هایی که قبلا ندیده است (به عنوان مثال، یک مشتری جدید، یک جرم جدید، یک تصویر جدید) خوب عمل می­کند. به همین دلیل، روش ارزیابی باید به ما کمک کند تا بفهمیم مدل­ها تا چه حد قادر به پیش‌بینی از داده­هایی هستند که تاکنون ندیده است.

یک استراتژی ممکن است نگه داشتن بخشی از داده­ها برای آزمایش باشد. این روش اعتبار سنجی(نگه داشتن) نامیده می‌شود. در اعتبارسنجی(validation)، مشاهدات ما (ویژگی‌ها و اهداف) به دو مجموعه تقسیم می‌شوند که به طور سنتی مجموعه آموزشی و مجموعه آزمون(تست) نامیده می‌شوند. مجموعه تست را انتخاب می‌کنیم و کنار می­گذاریم و وانمود می­کنیم که قبلاً آن را ندیده‌ایم. سپس مدل خود را با استفاده از مجموعه آموزشی خود، با استفاده از ویژگی­ها و بردار هدف آموزش می­دهیم تا به مدل یاد دهیم که چگونه بهترین پیش‌بینی را انجام دهد. در نهایت، با ارزیابی اینکه مدل آموزش دیده ما در مجموعه آموزشی چگونه در مجموعه آزمایشی عمل می­کند شبیه سازی می­کنیم که تا به حال، داده­های خارجی را ندیده‌ایم. با این حال، رویکرد اعتبارسنجی دو نقطه ضعف عمده دارد. اولین مورد اینکه، عملکرد مدل می­تواند بسیار به این وابسته باشد که مشاهدات کمی برای مجموعه‌ی تست انتخاب شده است. مورد دوم اینکه، مدل با استفاده از تمام داده­های موجود آموزش داده نمی­شود، و بر روی تمام داده­های موجود ارزیابی نمی­شود.

یک استراتژی بهتر، که بر این نقاط ضعف غلبه می‌کند، اعتبارسنجی متقاطع k-fold cross.validation (KFCV) نامیده می‌شود. در KFCV، داده­ها را به k بخش به نام "fold"(دسته) تقسیم می‌کنیم. سپس مدل با استفاده از k-1 دسته که در یک مجموعه آموزشی ترکیب می‌شوند، آموزش داده می‌شود و سپس آخرین دسته به عنوان یک مجموعه تستی استفاده می‌شود و مدل با این دسته تست می‌شود. ما این­کار را k بار، هر بار با استفاده از یک fold متفاوت به عنوان مجموعه تست تکرار می­کنیم. سپس عملکرد مدل برای هر یک از k تکرار میانگین گیری می­شود تا یک اندازه‌گیری کلی ایجاد شود.

در راه حل خود، اعتبارسنجی متقاطع k-fold را با استفاده از 10 fold انجام دادیم و نتایج ارزیابی را به cv\_results ارائه کردیم:





هنگام استفاده از KFCV سه نکته‌ی مهم وجود دارد که باید در نظر بگیریم. اول، KFCV فرض می‌کند که هر مشاهده، مستقل از دیگری ایجاد شده است(یعنی داده‌ها به طور یکسان توزیع شده‌اند [IID]). اگر داده‌ها IID هستند، ایده خوبی است که هنگام تخصیص به foldها، مشاهدات را جا به جا کنید. در sikit learn می‌توانیم shuffle=True­­ را برای انجام جابجایی تنظیم کنیم.

دوم، هنگامی که از KFCV برای ارزیابی یک دسته‌بندی‌کننده استفاده می‌کنیم، اغلب داشتن fold­های حاوی درصد یکسان مشاهدات از هر یک از کلاس‌های هدف مختلف(به نام k-fold دسته‌بندی شده) مفید است. به عنوان مثال، اگر بردار هدف ما حاوی جنسیت باشد و 80 درصد مشاهدات مذکر باشد، در این صورت هر دسته شامل 80 درصد مشاهدات مرد و 20 درصد زنان خواهد بود. در scikit-learn، می‌توانیم اعتبارسنجی متقاطع k-fold دسته‌بندی شده را با جایگزین کردن کلاس KFold با StratifiedKFold انجام دهیم.

در نهایت، هنگامی که از مجموعه‌ی اعتبارسنجی متقاطع استفاده می‌کنیم، پیش‌پردازش داده‌ها براساس مجموعه‌ی آموزشی و سپس اعمال این تبدیلات برای مجموعه‌ی آموزشی و تست مهم است. برای مثال، هنگامی که با هدف استاندارد سازی، standardizer، برازش می‌کنیم، تنها میانگین و واریانس مجموعه‌ی آموزشی را محاسبه می‌کنیم. سپس آن تبدیل(با استفاده از transform) را برای مجموعه­های آموزشی و تست اعمال می­کنیم:



دلیل این امر این است که ما وانمود می­کنیم که مجموعه‌ی آزمایشی، داده‌ی ناشناخته است. اگر ما هر دو پیش پردازش خود را با استفاده از مشاهدات هر دو مجموعه‌ی آموزشی و تست هماهنگ کنیم، برخی از اطلاعات مجموعه‌ی تست به مجموعه‌ی آموزشی ما تراوش می‌کند. این قانون برای هر مرحله‌ی پیش‌پردازش مانند انتخاب ویژگی اعمال می‌شود.

روند کاری پکیج pipeline در scikit-learn انجام این کار را با استفاده از تکنیک­های اعتبارسنجی متقابل آسان می­کند. ابتدا یک خط لوله ایجاد می­کنیم که داده­ها را از قبل پردازش می­کند(به عنوان مثال، استانداردساز) و سپس یک مدل(رگرسیون لوجستیکی، لاجیت) را آموزش می­دهد:



سپس KFCV را با استفاده از آن روند کاری اجرا می‌کنیم و scikit همه کارها را برای ما انجام می­دهد:



cross\_val\_score دارای سه پارامتر شایان ذکر است که ما در مورد آنها صحبت نکردیم. cv روش اعتبارسنجی متقاطع ما را تعیین می‌کند. K-fold تا کنون رایج‌ترین روش است، اما موارد دیگری نیز وجود دارد، مانند leave-one-out-cross-validation که در آن تعداد fold­های k برابر با تعداد مشاهدات است. پارامتر scoring معیار ما را برای موفقیت تعریف می‌کند که تعدادی از آن‌ها در دستور العمل‌های دیگر در این فصل مورد بحث قرار می‌گیرند. در نهایت، n\_jobs=-1 به scikit-learn استفاده از هر هسته‌ی موجود را بیان می­کند. به عنوان مثال، اگر کامپیوتر شما چهار هسته دارد(یک عدد رایج برای لپ تاپ­ها)، پس scikit-learn از هر چهار هسته به صورت هم زمان برای سرعت بخشیدن به عملیات استفاده خواهد کرد.

**برای مطالعه بیشتر**

* [Why every statistician should know about cross.validation](https://robjhyndman.com/hyndsight/crossvalidation/)
* [Cross.Validation Gone Wrong](https://betatim.github.io/posts/cross-validation-gone-wrong/)

**١١.٢. ایجاد یک مدل رگرسیون پایه**

**مسئله**

شما یک مدل رگرسیون پایه ساده می­خواهید تا با مدل خود مقایسه کنید.

**راه حل**

برای ایجاد یک مدل ساده که بتوان از آن به عنوان یک خط پایه استفاده کرد، از دستور DummyRegressor در scikit-learn استفاده کنید:





برای مقایسه، مدل خود را آموزش می‌دهیم و امتیاز عملکرد را ارزیابی می‌کنیم:





**بحث**

DummyRegressor به ما اجازه می‌دهد تا یک مدل بسیار ساده ایجاد کنیم که بتوانیم از آن به عنوان یک پایه(اساس) برای مقایسه با مدل واقعی خود استفاده کنیم. این امر اغلب می‌تواند برای شبیه‌سازی یک فرآیند پیش‌بینی "ساده" در یک محصول یا سیستم مفید باشد. به عنوان مثال، ممکن است یک محصول در اصل طوری کدگذاری شده باشد تا تصور کنید که همه کاربران جدید در ماه اول، صرف نظر از ویژگی­های آن­ها، ۱۰۰ دلار هزینه خواهند کرد. اگر این فرض را در یک مدل پایه رمزگذاری کنیم، می‌توانیم مزایای استفاده از رویکرد یادگیری ماشین را به طور مشخص بیان کنیم.

DummyRegressor از پارامتر strategy برای تعیین روش پیش‌بینی، شامل مقدار میانگین یا میانه در مجموعه‌ی آموزشی استفاده می‌کند. علاوه بر این، اگر strategy را ثابت(constant) قرار دهیم و از پارامتر ثابت(constant) استفاده کنیم، می­توانیم رگرسیون ساخته‌شده را برای پیش‌بینی مقداری ثابت برای هر مشاهده تنظیم کنیم:





یک نکته کوچک در مورد score. به طور پیش‌فرض، score مقدار امتیاز ضریب توضیح(R.squared،R2) را برمی­گرداند:

که در آن ، ارزش واقعی مشاهده‌ی هدف است، ، مقدار پیش‌بینی‌شده است و ، مقدار میانگین برای بردار هدف است. هر چه به ۱ نزدیک‌تر باشد، واریانس در بردار هدف که توسط ویژگی­ها توضیح داده می­شود، بیشتر است.

**١١.٣. ایجاد یک مدل دسته‌بندی پایه**

**مسئله**

شما یک دسته‌بندی پایه‌ی ساده می‌خواهید تا با مدل خود مقایسه کنید.

**راه حل**

از DummyClassifier در scikit-learn استفاده کنید.





با مقایسه‍ی دسته‌بندی پایه با دسته‌بندی آموزش‌دیده خود، می­توانیم بهبود را ببینیم:





**بحث**

معیار معمول عملکرد یک دسته‌بندی‌کننده این است که چقدر بهتر از حدس تصادفی است. Dummy Classifier در scikit-learn این مقایسه را آسان می­کند. پارامتر strategy به ما چند گزینه برای تولید مقادیر می‌دهد. دو استراتژی بسیار مفید وجود دارد. ابتدا، stratifiedپیش‌بینی‌هایی را انجام می‌دهد که متناسب با نسبت‌های کلاس بردار هدف مجموعه آموزشی است(به عنوان مثال، اگر 20٪ از مشاهدات در داده‌های آموزشی زنان باشند، DummyClassifier زنان را در 20٪ موارد پیش‌بینی می‌کند). دوم، uniform، پیش‌بینی­ها را به طور یکنواخت و تصادفی بین کلاس­های مختلف تولید می‌کند. به عنوان مثال، اگر ۲۰ % مشاهدات، زن و ۸۰ % مرد باشند، uniform پیش‌بینی­هایی را ایجاد می­کند که ۵۰ % زن و ۵۰ % مرد هستند.

**برای مطالعه بیشتر**

* [scikit-learn documentation: DummyClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html)

**۱۱.۴. ارزیابی پیش‌بینی­های دسته‌بندی‌کننده دودویی**

**مسئله**

با توجه به یک مدل دسته‌بندی آموزش دیده، می­خواهید کیفیت آن را ارزیابی کنید.

**راه حل**

از scikit-learn’s cross\_val\_score برای انجام اعتبارسنجی متقاطع استفاده کنید در حالی که از پارامتر scoring برای تعریف یک معیار از تعدادی معیار عملکرد، از جمله صحت، دقت، فراخوانی و امتیاز استفاده کنید.

دقت یک معیار عملکرد رایج است. به بیان ساده، نسبت مشاهداتی است که به درستی پیش‌بینی‌شده اند:

که در آن:

TP تعداد مثبت­های حقیقی است. مشاهداتی که بخشی از طبقه مثبت هستند(بیماری دارند، محصول را خریده اند و غیره) و اینکه ما به درستی پیش‌بینی کرده‌ایم.

TN تعداد منفی­های حقیقی است. مشاهداتی که بخشی از طبقه منفی هستند (بیماری ندارند، محصول را نخریده اند و غیره) و اینکه ما به درستی پیش‌بینی کرده‌ایم.

FP تعداد مثبت­های کاذب است. همچنین خطای نوع اول(*I*) نیز نامیده می­شود. این مشاهدات پیش‌بینی‌شده از کلاس مثبت هستند که در واقع اشتباه پیش‌بینی‌شده‌اند و از کلاس منفی هستند.

FN تعداد منفی­های کاذب است. همچنین خطای نوع دوم(II ) نیز نامیده می­شود. این مشاهدات پیش‌بینی‌شده از کلاس منفی هستند که در واقع اشتباه پیش‌بینی‌شده‌اند و از کلاس مثبت هستند.

ما می­توانیم صحت را در اعتبارسنجی متقاطع سه fold(تعداد پیش‌فرض foldها) با تنظیم scoring='accuracy' اندازه‌گیری کنیم:





جذابیت صحت در این است که یک توضیح شهودی و ساده‌ دارد: نسبت مشاهداتی که به درستی پیش‌بینی‌شده اند. با این حال، در دنیای واقعی، اغلب داده­های ما دارای کلاس­های نامتعادل هستند(به عنوان مثال، 99.9٪ مشاهدات از کلاس 1 و تنها 0.1٪ کلاس 2 هستند). در حضور کلاس­های نامتعادل، صحت از پارادوکسی رنج می‌برد که در آن یک مدل، بسیار دقیق است اما فاقد قدرت پیش‌بینی است. به عنوان مثال، تصور کنید ما در حال تلاش برای پیش‌بینی وجود یک سرطان بسیار نادر هستیم که در 0.1٪ از جمعیت رخ می­دهد. پس از آموزش مدل متوجه می‌شویم که صحت آن ۹۵ درصد است. این در حالی است که ۹۹.۹ درصد افراد به این سرطان مبتلا نیستند: اگر ما به سادگی مدلی بسازیم که "پیش‌بینی کرده باشد" که هیچ‌کس چنین شکلی از سرطان ندارد، مدل ساده‌ی ما ۴.۹ درصد دقیق‌تر خواهد بود، اما به وضوح قادر به پیش‌بینی هیچ چیز نیست. به همین دلیل، ما اغلب برای استفاده از معیارهای دیگر مانند دقت، فراخوانی و امتیاز تحریک می‌شویم.

دقت، نسبت هر مشاهده‌ای است که پیش‌بینی می­شود مثبت باشد که در واقع مثبت است. ما می­توانیم آن را به عنوان یک نویز اندازه‌گیری در پیش‌بینی­های خود در نظر بگیریم . یعنی وقتی ما پیش‌بینی می­کنیم چیزی مثبت است، چقدر احتمال دارد که درست بگوییم. مدل­هایی که دقت بالایی دارند به این دلیل بدبین هستند که تنها زمانی یک مشاهده را پیش‌بینی می­کنند که در مورد آن بسیار مطمئن باشند. به طور رسمی، دقت عبارت است از:





فراخوانی(بازیابی) نسبت هر مشاهده‌ی مثبت است که واقعا ً مثبت است و توانایی مدل برای شناسایی مشاهده کلاس مثبت را، اندازه‌گیری می‌کند. مدل­هایی با فراخوانی(بازیابی) بالا خوش‌بین هستند که بار پایینی برای پیش‌بینی اینکه یک مشاهده در کلاس مثبت است دارند:





اگر این اولین باری است که با دقت و یادآوری(recall) مواجه می­شوید، قابل درک است اگر کمی طول بکشد تا آن­ها را به طور کامل درک کنید. این یکی از معایب دقت است؛ دقت و یادآوری کم‌تر شهودی هستند. تقریبا همیشه نوعی تعادل بین دقت و یادآوری(recall) می­خواهیم و این نقش با امتیاز F1 پر می­شود. امتیاز F1 یک میانگین هارمونیک است(نوعی میانگین که برای نسبت­ها استفاده می­شود):

این معیاری از درستی است که در پیش‌بینی مثبت به دست می­آید. یعنی از مشاهداتی که به عنوان مثبت برچسب‌گذاری شده اند، چه تعداد واقعا مثبت هستند:





**بحث**

به عنوان یک معیار ارزیابی، صحت، ویژگی­های ارزشمندی دارد، به خصوص شهود ساده‌ی آن. با این حال، معیارهای بهتر، اغلب شامل تعادل استفاده از دقت(percision) و یادآوری(recall) است . یعنی توازنی بین خوش بینی و بدبینی مدل ما. F1 نشان دهنده تعادل بین بازیابی و دقت است، جایی که سهم نسبی هر دو برابر است.

همچنین برای استفاده از cross\_val\_score، اگر مقادیر y حقیقی و مقادیر y پیش‌بینی‌شده را داشته باشیم، می‌توانیم معیارهایی مانند دقت و یادآوری را مستقیما محاسبه کنیم:





**برای مطالعه بیشتر**

* [Accuracy paradox](https://en.wikipedia.org/wiki/Accuracy_paradox)

**١١.٥. ارزیابی آستانه­های دسته‌بندی‌کننده‌ی دودویی**

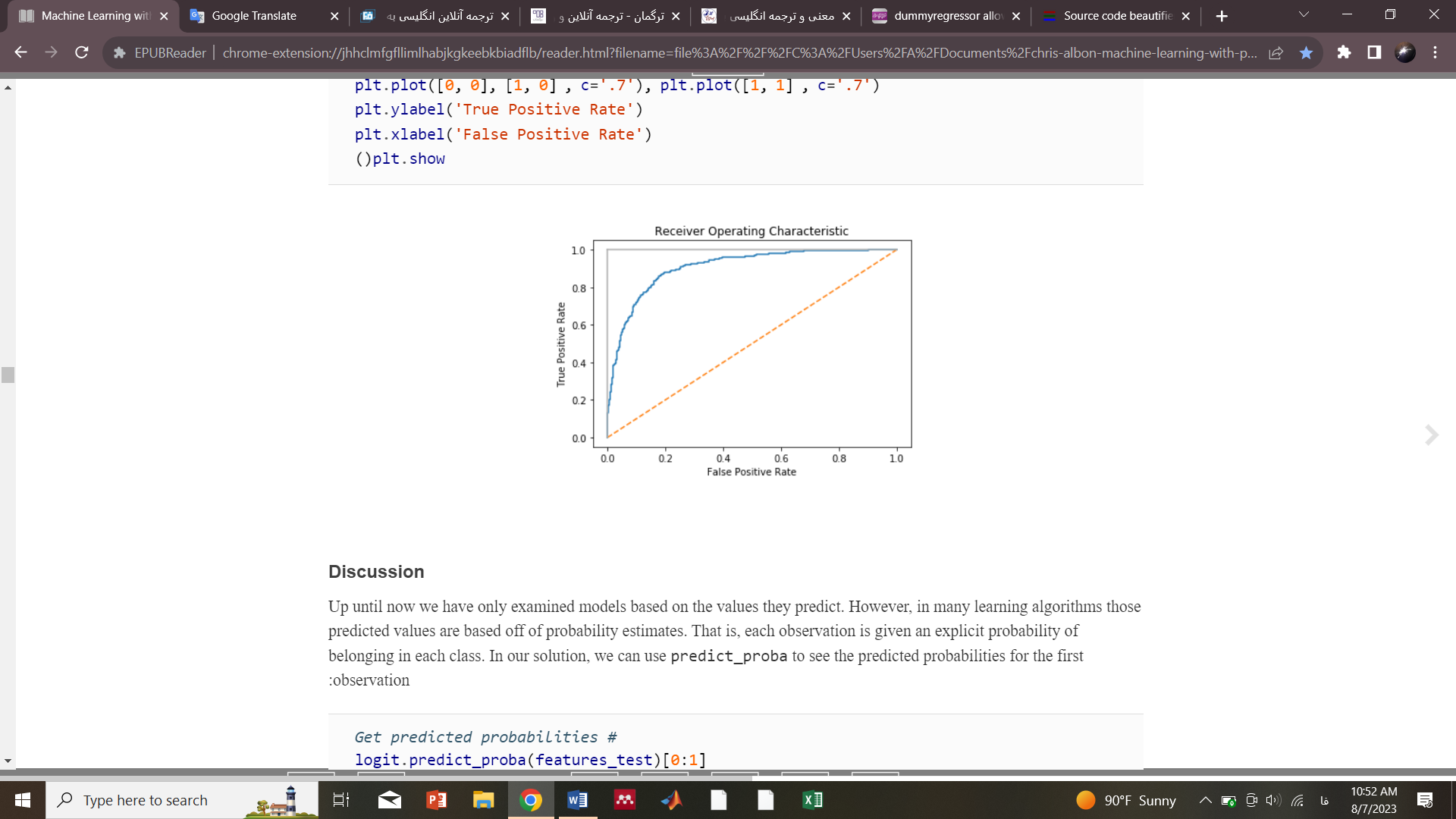
**مسئله**

می­خواهید یک دسته‌بندی‌کننده‌ی باینری و آستانه­های احتمالی مختلف را ارزیابی کنید.

**راه حل**

منحنی دریافت‌کننده مشخصه‌ی عملیاتی (ROC) یک روش رایج برای ارزیابی کیفیت یک دسته‌بندی‌کننده‌ی باینری(دودویی) است. ROC حضور مثبت­های حقیقی و مثبت­های کاذب را در هر آستانه احتمال مقایسه می‌کند (یعنی احتمال اینکه یک مشاهده در آن یک کلاس باشد). با رسم منحنی ROC، می‌توانیم عملکرد مدل را مشاهده کنیم. دسته‌بندی‌کننده‌ای که هر مشاهده را به درستی پیش‌بینی می­کند، مانند خط خاکستری روشن در نمودار زیر است که بلافاصله به سمت بالا می­رود. یک دسته‌بندی‌کننده که به صورت تصادفی پیش‌بینی می­کند، به صورت خط مورب ظاهر خواهد شد. هر چه مدل بهتر باشد، به خط ثابت نزدیک‌تر است. در scikit-learn، می‌توانیم از roc\_curve برای محاسبه مثبت‌های درست و نادرست در هر آستانه استفاده کنیم، سپس آنها را رسم کنیم:





**بحث**

تا کنون ما فقط مدل­ها را بر اساس مقادیری که پیش‌بینی می­کنند بررسی کرده‌ایم. با این حال، در بسیاری از الگوریتم­های یادگیری، آن مقادیر پیش‌بینی‌شده براساس برآورده‌ای از احتمالات هستند. یعنی به هر مشاهده یک احتمال صریح وابسته به هر کلاس داده می­شود. در این راه حل، ما می­توانیم ازpredict\_proba  استفاده کنیم تا احتمال­های پیش‌بینی‌شده برای اولین مشاهده را ببینیم:





ما می­توانیم کلاس­ها را با استفاده از classes\_ ببینیم:





در این مثال، اولین مشاهده دارای 87% احتمال قرار گرفتن در کلاس منفی (0) و 13% احتمال قرار گرفتن در کلاس مثبت (1) است. به طور پیش‌فرض، scikit-learn پیش‌بینی می­کند که یک مشاهده بخشی از کلاس مثبت است اگر احتمال آن بیشتر از ۰.۵ باشد (که آستانه نامیده می­شود).با این حال، به جای یک حد وسط، اغلب می‌خواهیم به طور صریح مدل خود را برای استفاده از یک آستانه‌ی متفاوت به دلایل متعدد به کار ببریم. به عنوان مثال، اگر مثبت کاذب برای کار ما بسیار پرهزینه باشد، ممکن است مدلی را ترجیح دهیم که آستانه احتمال بالایی دارد. ما نمی‌توانیم برخی از موارد مثبت را پیش‌بینی کنیم، اما وقتی یک مشاهده مثبت پیش‌بینی می‌شود، می‌توانیم بسیار مطمئن باشیم که پیش‌بینی درست است. این موازنه در نرخ مثبت حقیقی (TPR) و نرخ مثبت کاذب (FPR) نشان داده می­شود. نرخ مثبت حقیقی تعداد مشاهداتی است که به درستی پیش‌بینی‌شده اند و تقسیم بر تمام مشاهدات مثبت حقیقی می­شود:

نرخ مثبت کاذب، تعداد مثبت­های پیش‌بینی‌شده نادرست تقسیم بر تمام مشاهدات منفی واقعی است:

منحنی ROC نشان‌دهنده‌ی TPR و FPR مربوطه برای هر آستانه‌ی احتمال است. به عنوان مثال، در راه حل ما، یک آستانه تقریباً 0.50 دارای TPR 0.81 and an FPR of\0.15 است:





با این حال، اگر آستانه را به ۸۰ % افزایش دهیم (یعنی، میزان اطمینان مدل را قبل از اینکه یک مشاهده را مثبت پیش‌بینی کند افزایش دهیم)TPR به طور قابل توجهی کاهش می­یابد و به همین صورت FPR نیز کاهش می­یابد:





این به این دلیل است که نیاز بیشتر ما برای پیش‌بینی در کلاس مثبت باعث شده‌است که مدل تعدادی از مشاهدات مثبت (TPR پایین‌تر) را شناسایی نکند، بلکه نویز حاصل از مشاهدات منفی را به صورت مثبت (FPR پایین‌تر) کاهش دهد.

منحنی ROC علاوه بر اینکه می‌تواند مبادله‌ی بین TPR و FPR را تجسم کند، می‌تواند به عنوان یک معیار کلی برای یک مدل نیز استفاده شود. هر چه یک مدل بهتر باشد، منحنی بالاتر و در نتیجه مساحت زیر منحنی بیشتر است. به همین دلیل، محاسبه‌ی مساحت زیر منحنی ROC (AUCROC) برای مقایسه در مورد برابری کلی یک مدل در تمام آستانه‌های ممکن، معمول است. هر چه AUCROC به 1 نزدیکتر باشد، مدل بهتر است. در scikit-learn می‌توانیم AUCROC را با استفاده از roc\_auc\_score محاسبه کنیم:





**برای مطالعه بیشتر**

* [ROC Curves in Python and R](https://community.alteryx.com/t5/Data-Science/ROC-Curves-in-Python-and-R/ba-p/138430)
* [The Area Under an ROC Curve](https://darwin.unmc.edu/dxtests/roc3.htm)

**١١.٦. ارزیابی پیش‌بینی­های دسته‌بندی‌کننده چند طبقه**

**مسئله**

مدلی دارید که سه یا چند کلاس را پیش‌بینی می­کند و می­خواهید عملکرد آن را ارزیابی کنید.

**راه حل**

از اعتبارسنجی متقاطع با معیار ارزیابی‌ای که قادر به مدیریت بیش از دو کلاس است استفاده کنید:





**بحث**

وقتی کلاس‌های متعادلی داریم (مثلاً تعداد تقریباً مساوی از مشاهدات در هر کلاس از بردار هدف)، صحت(accuracy)، درست مانند تنظیمات کلاس باینری، یک انتخاب ساده و قابل تفسیر برای یک متریک ارزیابی است. صحت عبارت است از تعداد پیش‌بینی­های صحیح تقسیم بر تعداد مشاهدات که درست مانند تنظیمات باینری عمل می­کند. با این حال، زمانی که کلاس‌های نامتعادل داریم (یک سناریوی رایج)، باید به استفاده از معیارهای ارزیابی دیگر تمایل داشته باشیم.

بسیاری از معیارهای داخلی scikit-learn برای ارزیابی طبقه‌بندی‌کننده‌های باینری هستند. با این حال، بسیاری از این معیارها را می­توان برای استفاده در زمانی که بیش از دو کلاس داریم گسترش داد. دقت، یادآوری و امتیاز F1 معیارهای مفیدی هستند که قبلا به طور مفصل در دستورالعمل­های قبلی پوشش داده شده اند. در حالی که همه‌ی آنها در ابتدا برای طبقه‌بندی‌کننده‌های باینری طراحی شده بودند، می‌توانیم با در نظر گرفتن داده‌هایمان به‌عنوان مجموعه‌ای از کلاس‌های باینری، آن‌ها را در تنظیمات چند طبقه اعمال کنیم. انجام این کار، ما را قادر می­سازد تا معیارها را برای هر کلاس به گونه‌ای اعمال کنیم که گویی تنها کلاس موجود در داده­ها است، و سپس با میانگین‌گیری از آن­ها امتیازات ارزیابی را برای تمام کلاس­ها جمع آوری کنیم:





در این کد، \_macro به روش استفاده شده برای میانگین نمرات ارزیابی از کلاس­ها اشاره دارد:

macro میانگین امتیازات متریک را برای هر کلاس محاسبه می­کند و هر کلاس را به طور مساوی وزن می­کند.

weighted میانگین امتیازات متریک را برای هر کلاس محاسبه می­کند و هر کلاس را متناسب با اندازه‌ی آن در داده‌ها وزن می­کند.

micro میانگین امتیازات متریک را برای هر ترکیب کلاس مشاهده محاسبه می­کند.

**١١.٧. نمایش عملکرد یک دسته‌بندی‌کننده**

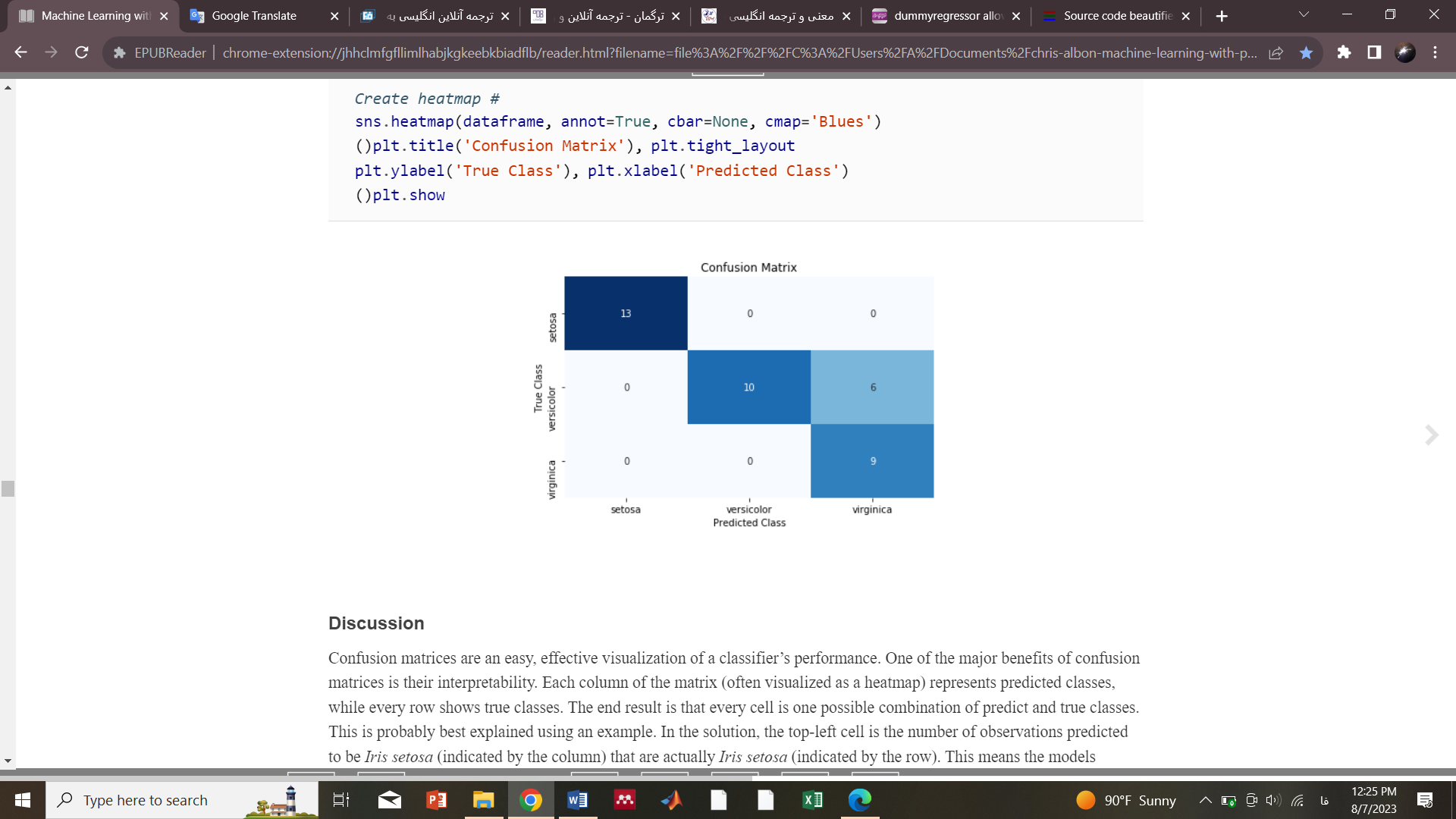
**مسئله**

با توجه به کلاس­های پیش‌بینی‌شده و کلاس­های واقعی داده­های آزمون، شما می­خواهید کیفیت مدل را به صورت بصری مقایسه کنید.

**راه حل**

از یک ماتریس درهم‌ریختگی(Confusion Matrix) استفاده کنید که کلاس­های پیش‌بینی‌شده و کلاس­های واقعی را مقایسه می­کند:





**بحث**

ماتریس­های درهم‌ریختگی نمایشی آسان و موثر از عملکرد یک دسته‌بندی‌کننده هستند. یکی از مزایای عمده‌ی این ماتریس­ها، تفسیرپذیری آنهاست. هر ستون از ماتریس (که اغلب به صورت یک نقشه حرارتی تجسم می­شود) کلاس­های پیش‌بینی‌شده را نشان می­دهد، در حالی که هر سطر کلاس­های واقعی را نشان می­دهد. نتیجه نهایی این است که هر سلول یک ترکیب ممکن از کلاس‌های پیش‌بینی و واقعی است. احتمالا ً بهتر است با استفاده از یک مثال توضیح داده شود. در این راه حل، سلول بالا سمت چپ، تعداد مشاهداتی است که پیش‌بینی می­شود زنبق هستند (با ستون نشان داده می­شود) که در واقعیت نیز زنبق (نشان داده شده توسط ردیف) هستند. این یعنی مدل­ها به درستی تمام گل­های زنبق را پیش‌بینی کردند. با این حال، این مدل در پیش‌بینی میخک به خوبی عمل نمی­کند. سلول پایین سمت راست نشان می‌دهد که مدل با موفقیت یازده مشاهدات Iris virginica را پیش‌بینی کرده، اما (با نگاه کردن به یک سلول بالا) یک گل را پیش‌بینی کرد که در واقع رنگارنگ زنبق بود.

سه نکته در مورد ماتریس­های درهم‌ریختگی قابل توجه است. اول، یک مدل کامل مقادیری در امتداد قطر و همچنین مقدار صفر در هر جای دیگر خواهد داشت. یک مدل بد، نشان می­دهد که تعداد مشاهدات به طور یکنواخت در اطراف سلول­ها پخش شده. دوم، یک ماتریس درهم‌ریختگی به ما اجازه می­دهد تا نه تنها ببینیم که مدل کجا اشتباه بوده است، بلکه ببینیم که چگونه اشتباه بوده است. یعنی می‌توانیم به الگوهای دسته‌بندی نادرست نگاه کنیم. به عنوان مثال، مدل ما به آسانی زنبق و میخک را تشخیص داد اما میخک و زنبق رنگین‌کمانی را به دشواری تشخیص می­داد. سوم، ماتریس­های درهم‌ریختگی با هر تعداد کلاس کار می­کنند (اگر چه اگر یک میلیون کلاس در بردار هدف خود داشته باشیم، تجسم ماتریس در هم ریخنگی ممکن است دشوار باشد).

**برای مطالعه بیشتر**

* [Confusion matrix, Wikipedia](https://en.wikipedia.org/wiki/Confusion_matrix" \t "_blank)
* [scikit-learn documentation: Confusion Matrix](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

**١١.٨.ارزیابی مدل­های رگرسیون**

**مسئله**

شما می­خواهید عملکرد یک مدل رگرسیون را ارزیابی کنید.

**راه حل**

استفاده از میانگین مربعات خطا (MSE):





یکی دیگر از معیارهای رایج رگرسیون، ضریب تعیین، R2 است:





**بحث**

MSE یکی از رایج­ترین معیارهای ارزیابی برای مدل­های رگرسیون است. MSE عبارت است از:

که در آن n تعداد مشاهدات، yi مقدار حقیقی هدفی است که می­خواهیم برای مشاهده i پیش‌بینی کنیم و مقدار پیش‌بینی‌شده توسط مدل برای yi است. MSE اندازه‌گیری مجموع مربع تمام فواصل بین مقادیر پیش‌بینی‌شده و واقعی است. هر چه مقدار MSE بیشتر باشد، خطای مجذور کل بیشتر و در نتیجه مدل بدتر می­شود. تعدادی از مزایای ریاضی برای مجموع مربعات خطا وجود دارند؛ از جمله اینکه تمام مقادیر خطا را مجبور می­کند که مثبت باشند؛ اما یک مفهوم اغلب محقق نشده این است که مربع کردن، چند خطای بزرگ را بیشتر از بسیاری از خطاهای کوچک جریمه می­کند(به چشم می‌آورد). حتی اگر قدر مطلق خطاها یکسان باشد. برای مثال، دو مدل A و B را تصور کنید که هر کدام دارای دو مشاهده هستند:

مدل A دارای خطاهای 0 و 10 است و بنابراین MSE آن 100= 02 + 102 است.

مدل B هر کدام دو خطای 5 دارد و بنابراین MSE آن 50= 52 + 52  است.

هر دو مدل خطای کل یکسان برابر با 10 دارند؛ با این حال، MSE مدل (MSE = 100) A را بدتر از مدل B (MSE=50) در نظر می­گیرد. در عمل این مفهوم به ندرت یک مساله است (و در واقع می‌تواند از لحاظ نظری سودمند باشد) و MSE به عنوان یک معیار ارزیابی کاملا خوب عمل می­کند.

یک نکته‌ی مهم: به طور پیش‌فرض در آرگومان‌های scikit-learn پارامتر امتیازدهی فرض می­شود که مقادیر بالاتر، بهتر از مقادیر پایین­تر هستند. با این حال، این مورد برای MSE صادق نیست، که در آن مقادیر بالاتر به معنای مدل بدتر است. به همین دلیل، scikit-learn با استفاده از آرگومان neg\_mean\_squared\_errorبه MSE منفی نگاه می­کند.

یک معیار ارزیابی رگرسیون جایگزین رایج R2 است، که مقدار واریانس در بردار هدف را اندازه‌گیری می‌کند که توسط مدل توضیح داده می­شود:

در این فرمول، yi مقدار هدف واقعی مشاهده i ام است، مقدار پیش‌بینی‌شده برای مشاهده iام است و مقدار میانگین بردار هدف است. هرچه به 1.0 نزدیکتر باشد، مدل بهتر است.

برای مطالعه بیشتر

* [Mean squared error, Wikipedia](https://en.wikipedia.org/wiki/Mean_squared_error)
* [Coefficient of determination](https://en.wikipedia.org/wiki/Coefficient_of_determination)

**۱۱.۹. ارزیابی مدل­های خوشه‌بندی**

**مسئله**

شما از یک الگوریتم یادگیری بدون نظارت برای خوشه‌بندی داده­های خود استفاده کرده‌اید. حالا می­خواهید بدانید که مدل شما چقدر خوب عمل کرده است.

**راه حل**

پاسخ کوتاه این است که احتمالا نمی­توانید، حداقل نه آنطور که مد نظر دارید.

به این ترتیب، یک گزینه‌ی ارزیابی خوشه‌بندی با استفاده از ضرایب سیلوئت است که کیفیت خوشه‌ها را اندازه‌گیری می‌کند:





**بحث**

ارزیابی مدل نظارت شده، پیش‌بینی­ها (به عنوان مثال، کلاس­ها یا مقادیر کمی) را با مقادیر واقعی متناظر در بردار هدف مقایسه می­کند. با این حال، رایج‌ترین انگیزه برای استفاده از روش‌های خوشه‌بندی این است که داده‌های شما بردار هدف ندارند. تعدادی از معیارهای ارزیابی خوشه‌بندی وجود دارند که نیاز به بردار هدف دارند، اما باز هم، استفاده از روش­های یادگیری بدون نظارت مانند خوشه‌بندی زمانی که شما یک بردار هدف در دسترس دارید، احتمالا به طور غیرضروری خود را ناتوان می­کند.

در حالی که اگر بردار هدف نداشته باشیم نمی­توانیم پیش‌بینی­ها را در مقابل مقادیر واقعی ارزیابی کنیم و می­توانیم ماهیت خود خوشه­ها را ارزیابی کنیم. به طور شهودی، ما می­توانیم خوشه­های "خوب" را تصور کنیم که فاصله­های بسیار کمی بین مشاهدات در یک خوشه (یعنی خوشه­های متراکم) و فاصله­های زیاد بین خوشه­های مختلف (یعنی خوشه­های به خوبی جدا شده) دارند. ضرایب سیلوئت(Silhouette)، یک مقدار واحد برای اندازه‌گیری هر دو صفت فراهم می‌کنند. ضریب سیلوئت مشاهده i ام برابر است با:

که در آن si ضریب سیلوئت برای مشاهده‌ی iام، میانگین فاصله بین i و تمام مشاهدات یک کلاس و bi میانگین فاصله بین i و تمام مشاهدات از نزدیک‌ترین خوشه یک کلاس متفاوت است. مقدار برگردانده شده توسط silhouette\_score میانگین ضریب سیلوئت برای همه مشاهدات است. ضرایب سیلوئت بین 1- و 1 است که 1 نشان دهنده خوشه­های متراکم و به خوبی جدا شده است.

**برای مطالعه بیشتر**

* [scikit-learn documentation: silhouette\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html)

**١١.١٠.ایجاد معیار ارزیابی ویژه**

**مسئله**

شما می­خواهید یک مدل را با استفاده از معیاری که ایجاد کرده‌اید ارزیابی کنید.

**راه حل**

اندازه را به عنوان یک تابع ایجاد کنید و آن را با استفاده از make\_scorerدر scikit-learn به یک تابع امتیاز دهنده تبدیل کنید:





**بحث**

در حالی که scikit-learn تعدادی معیار داخلی برای ارزیابی عملکرد مدل را دارد، اغلب تعریف معیارهای خود نیز مفید است. scikit-learn این کار را با استفاده از make\_scorer آسان می­کند. ابتدا، تابعی را تعریف می‌کنیم که دو آرگومان می‌گیرد. بردار هدف اصلی و مقادیر پیش‌بینی‌شده‌ی ما، و مقداری امتیاز را به دست می‌آورد (در خروجی به ما میدهد). دوم، ما از make\_scorer برای ایجاد یک امتیازدهنده استفاده می‌کنیم، و مطمئن می‌شویم که نمره‌های بالاتر یا پایین‌تر مطلوب هستند (با استفاده از پارامتر greater\_is\_better). معیار سفارشی (ویژه) در راه حل (custom\_metric) یک مثال اسباب بازی است زیرا به سادگی یک معیار داخلی برای محاسبه امتیاز 2R را پوشش می­دهد. در یک موقعیت واقعی، ما تابع custom\_metric را با هر متریک سفارشی‌ای که می‌خواهیم جایگزین می‌کنیم. با این حال، مشاهده می­شود که متریک سفارشی‌ای که 2R را محاسبه می­کند، با مقایسه نتایج با روش ساخته شده‌ی scikit-learn’s r2\_score کار می­کند:



**برای مطالعه بیشتر**

* [scikit-learn documentation: make\_scorer](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html)

**۱۱.۱۱. نمایش تاثیر اندازه مجموعه آموزشی**

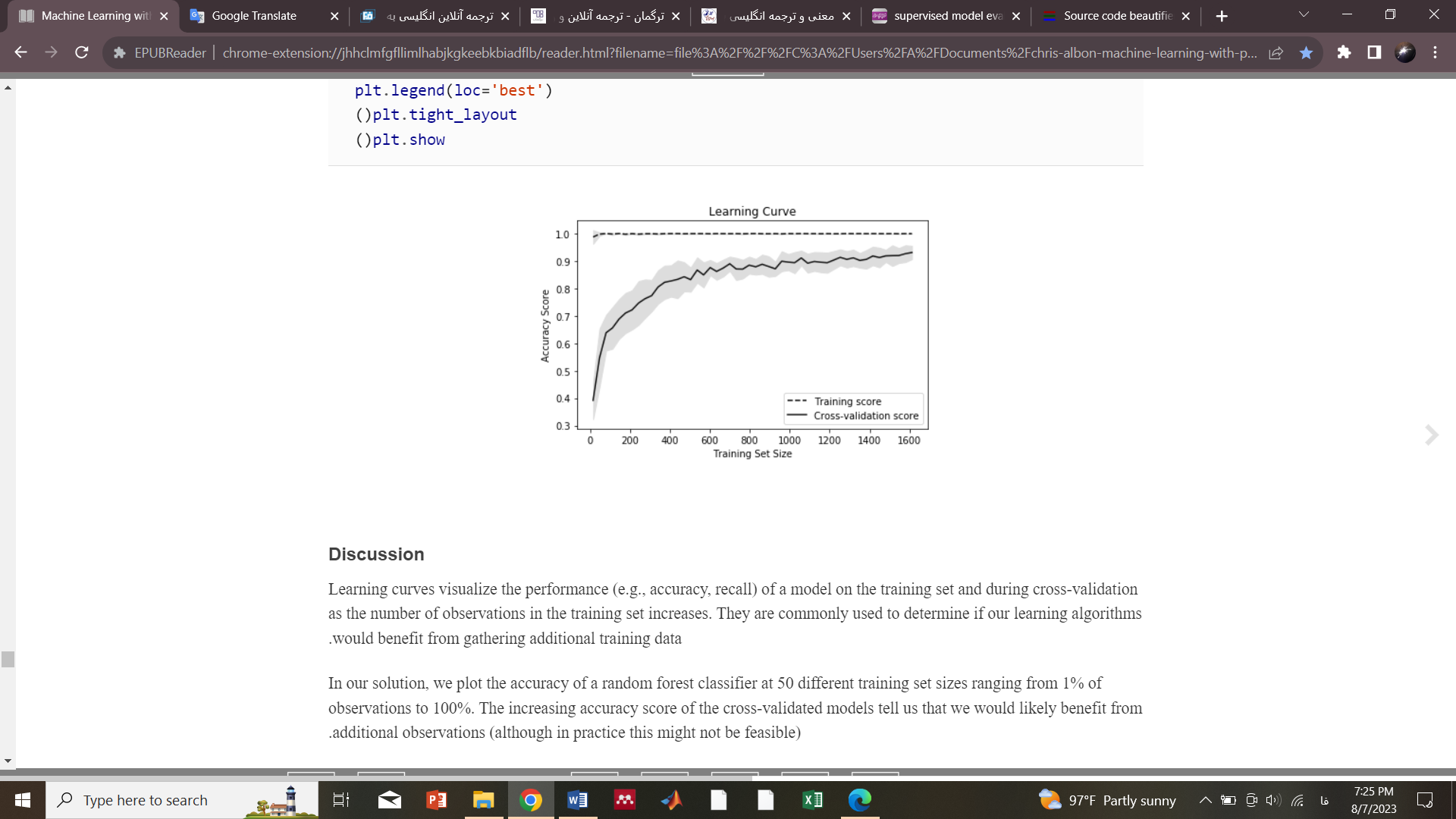
**مسئله**

شما می­خواهید تأثیر تعداد مشاهدات در مجموعه آموزشی خود را بر روی برخی متریک­ها (صحت، F1 و غیره) ارزیابی کنید.

**راه حل**

منحنی یادگیری را رسم کنید:





**بحث**

منحنی­های یادگیری، عملکرد (به عنوان مثال، صحت، یادآوری) یک مدل را در مجموعه آموزشی و در طول اعتبارسنجی متقابل با افزایش تعداد مشاهدات در مجموعه آموزشی، تصویرسازی می­کنند. آن‌ها معمولا ً برای تعیین اینکه آیا الگوریتم‌های یادگیری ما از جمع‌آوری داده‌های آموزشی اضافی بهره خواهند برد، استفاده می‌شوند.

در راه حل ما، ما دقت یک طبقه‌بندی‌کننده جنگل تصادفی(random forest) را در 50 اندازه‌ی مجموعه‌ی آموزشی مختلف از 1٪ مشاهدات تا 100٪ ترسیم می‌کنیم. افزایش امتیاز صحت مدل­های تایید شده‌ی متقاطع به ما می­گوید که ما احتمالا از مشاهدات اضافی بهره خواهیم برد (اگرچه این مورد در عمل ممکن است عملی نباشد).

برای مطالعه بیشتر

* [scikit-learn documentation: Learning Curve](https://scikit-learn.org/stable/modules/learning_curve.html" \l "learning-curve)

**١١.١٢.ایجاد یک گزارش متنی از معیارهای ارزیابی**

**مسئله**

شما یک توصیف سریع از عملکرد یک دسته‌بندی می­خواهید.

**راه حل**

از classification\_report در scikit-learn استفاده کنید:





**بحث**

classification\_report راهکاری سریع برای ما فراهم می‌کند تا برخی از معیارهای ارزیابی رایج از جمله دقت، یادآوری و امتیاز F1 را ببینیم (که قبلاً در این فصل توضیح داده شد). پشتیبانی(support) به تعداد مشاهدات در هر کلاس اشاره دارد.

**برای مطالعه بیشتر**

* [Precision and recall](https://en.wikipedia.org/wiki/Precision_and_recall)

**١١.١٣. تجسم اثر مقادیر فراپارامتر**

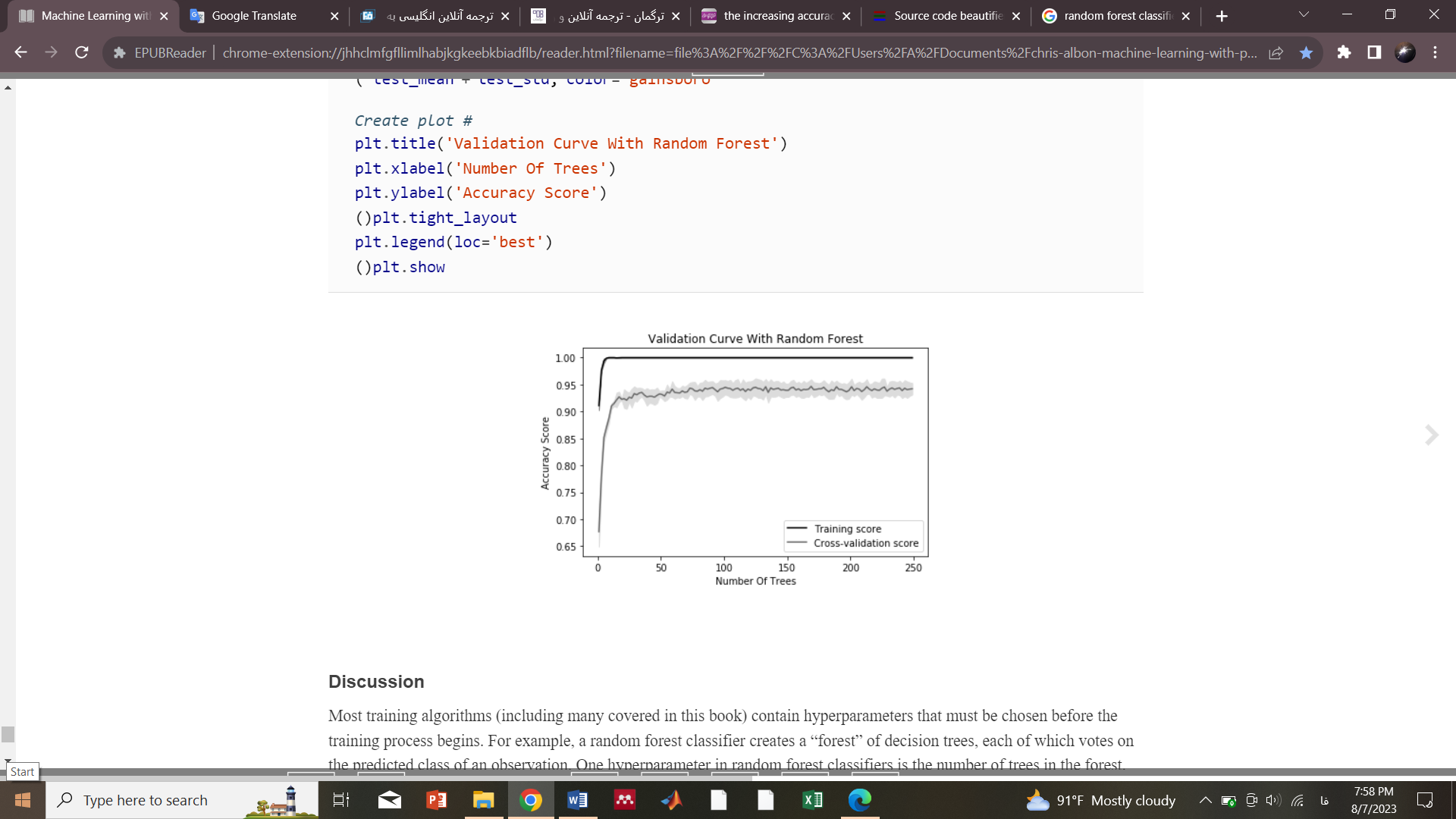
**مسئله**

شما قصد دارید متوجه چگونگی تغییر عملکرد یک مدل با تغییر مقدار برخی ­هایپرپارامترها شوید.

**راه حل**

منحنی اعتبار‌سنجی را رسم کنید:





**بحث**

بیشتر الگوریتم‌های آموزشی (از جمله بسیاری از الگوریتم‌های مورد بحث در این کتاب) حاوی هایپرپارامترهایی هستند که باید قبل از شروع فرآیند آموزش انتخاب شوند. برای مثال، یک دسته‌بندی‌کننده‌ی جنگل تصادفی(random forest) یک "جنگل" از درختان تصمیم ایجاد می­کند که هر کدام از آن­ها به کلاس پیش‌بینی‌شده‌ی یک مشاهده رای می­دهند. یک هایپرپارامتر در دسته‌بندی‌کننده­های جنگل تصادفی، تعداد درختان در جنگل است. اغلب مقادیر هایپرپارامتر در طول انتخاب مدل انتخاب می­شوند (به فصل 12 مراجعه کنید). با این حال، گاهی اوقات تصویرسازی نحوه‌ی تغییر عملکرد مدل با تغییر مقدار هایپرپارامتر مفید است. در راه حل ما، ما تغییرات در صحت را برای یک دسته‌بندی‌کننده‌ی جنگل تصادفی برای مجموعه‌ی آموزشی و در طول اعتبارسنجی متقابل با افزایش تعداد درختان رسم می­کنیم. هنگامی که تعداد کمی درخت داریم، هم امتیاز آموزش و هم امتیاز اعتبارسنجی متقاطع پایین است، که نشان می‌دهد مدل ضعیف است. با افزایش تعداد درختان به ۲۵۰ عدد، دقت هر دو سطح کاهش می­یابد که نشان می­دهد احتمالا ارزش چندانی در هزینه محاسباتی آموزش یک جنگل عظیم وجود ندارد.

در scikit-learn می‌توانیم منحنی اعتبارسنجی را با استفاده از validation\_curve محاسبه کنیم که شامل سه پارامتر مهم است:

aram\_name نام هایپرپارامتر برای تغییر است.

param\_range مقدار هایپرپارامتر مورد استفاده است.

scoring معیار ارزیابی مورد استفاده برای قضاوت در مورد مدل است.

**برای مطالعه بیشتر**

* [scikit-learn documentation: Validation Curve](https://scikit-learn.org/stable/modules/learning_curve.html" \l "validation-curve)

**فصل 14. درختان و جنگل­ها**

**١٤.٠. مقدمه**

الگوریتم‌های یادگیری مبتنی بر درخت، خانواده‌ی گسترده و محبوبی از روش‌های غیرپارامتریک و نظارت شده‌ی مرتبط به دسته‌بندی و رگرسیون هستند. اساس یادگیری مبتنی بر درخت، درخت تصمیم‌گیری است که در آن یک سری از قوانین تصمیم‌گیری (به عنوان مثال، "اگر جنسیت آن­ها مرد باشد...") به هم زنجیر می­شوند. نتیجه به طور مبهم شبیه یک درخت وارونه به نظر می‌رسد؛ با اولین قانون تصمیم‌گیری در بالا و قوانین تصمیم‌گیری بعدی که در زیر گسترش می‌یابند. در یک درخت تصمیم‌گیری، هر قاعده‌ی تصمیم‌گیری در یک گره تصمیم‌گیری رخ می­دهد و قواعد شاخه­هایی ایجاد می­کنند که منجر به گره­های جدید می­شوند. شاخه‌ای که در انتهای قاعده‌ی تصمیم‌گیری نباشد، برگ نامیده می­شود.

یکی از دلایل محبوبیت مدل‌های مبتنی بر درخت، قابلیت تفسیر آن‌ها است. در واقع، درخت­های تصمیم‌گیری را می­توان به معنای واقعی کلمه به شکل کامل آن­ها رسم کرد (دستورالعمل ۱۴.۳ را ببینید) تا یک مدل بسیار شهودی ایجاد شود. از این سیستم درختی پایه، طیف گسترده‌ای از پسوندها از جنگل‌های تصادفی گرفته تا متراکم کردن را داریم. در این فصل نحوه‌ی آموزش، مدیریت، تنظیم، تجسم و ارزیابی تعدادی از مدل‌های مبتنی بر درخت را پوشش خواهیم داد.

**١٤.١.آموزش یک دسته‌بندی درخت تصمیم**

**مسئله**

شما باید یک دسته‌بندی‌کننده را با استفاده از درخت تصمیم آموزش دهید.

**راه حل**

از DecisionTreeClassifier در scikit-learn استفاده کنید.



**بحث**

فراگیران درخت تصمیم‌گیری تلاش می­کنند تا یک قانون تصمیم‌گیری پیدا کنند که بیش­ترین کاهش ناخالصی را در یک گره ایجاد می­کند. در حالی که تعدادی اندازه‌گیری ناخالصی وجود دارد، به طور پیش‌فرض DecisionTreeClassifier از ناخالصی جینی استفاده می­کند:

که در آن G(t) ناخالصی جینی در گره t و pi نسبت مشاهدات کلاس c در گره t است. این فرآیند یافتن قوانین تصمیم‌گیری که باعث ایجاد شکاف برای افزایش ناخالصی می‌شوند، به صورت بازگشتی تکرار می‌شود تا زمانی که همه‌ی گره‌های برگ خالص باشند ( یعنی فقط یک کلاس ) یا یک برش دلخواه بدست آید.

DecisionTreeClassifier در scikit-learn، مانند سایر روش­های یادگیری عمل می­کند. به این صورت که پس از آموزش مدل با استفاده از برازش می­توانیم از مدل برای پیش‌بینی کلاس یک مشاهده استفاده کنیم:





همچنین می­توانیم احتمالات کلاس پیش‌بینی‌شده مشاهده را ببینیم:





در نهایت اگر بخواهیم از یک اندازه‌گیری ناخالصی متفاوت استفاده کنیم می­توانیم از پارامتر معیار استفاده کنیم:



**برای مطالعه بیشتر**

* [Decision Tree Learning، Princeton](https://www.cs.princeton.edu/courses/archive/spr07/cos424/papers/mitchell-dectrees.pdf)

**١٤.٢. آموزش رگرسیون درختی تصمیم‌گیری**

**مسئله**

باید یک مدل رگرسیون را با استفاده از درخت تصمیم آموزش دهید.

**راه حل**

از DecisionTreeRegressor در scikit-learn استفاده کنید.



**بحث**

رگرسیون درخت تصمیم‌گیری مشابه دسته‌بندی درخت تصمیم‌گیری عمل می­کند؛ با این حال، به جای کاهش ناخالصی جینی یا آنتروپی، شکاف­های بالقوه به طور پیش‌فرض براساس میزان کاهش خطای مربع میانگین(MSE) اندازه‌گیری می­شوند:

که در آن yi مقدار واقعی هدف و مقدار پیش‌بینی‌شده است. در scikit-learn، رگرسیون درخت تصمیم را می­توان با استفاده از DecisionTreeRegressor انجام داد. زمانی که درخت تصمیم‌گیری را آموزش دادیم، می‌توانیم از آن برای پیش‌بینی مقدار هدف برای مشاهده استفاده کنیم:



درست مانند DecisionTreeClassifier می‌توانیم از پارامتر criterion برای انتخاب اندازه‌گیری مطلوب کیفیت تقسیم استفاده کنیم. برای مثال، می‌توانیم درختی بسازیم که تقسیم‌های آن، میانگین خطای مطلق (MAE) را کاهش می‌دهد:



**برای مطالعه بیشتر**

* [Decision Tree Regression، scikit-learn](https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html)

**١٤.٣.نمایش یک مدل درخت تصمیم**

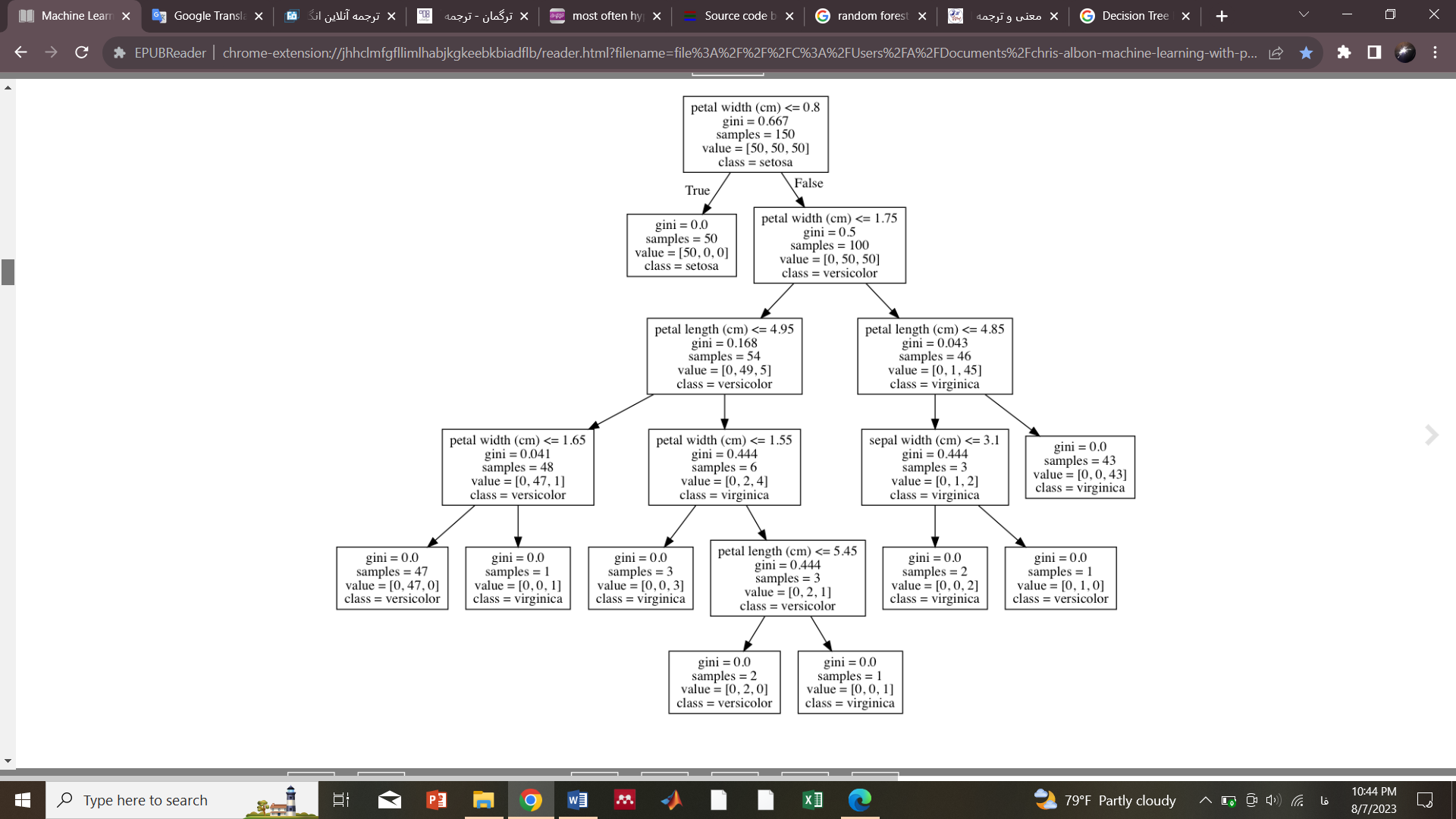
**مسئله**

شما باید یک مدل ایجاد شده توسط الگوریتم یادگیری درخت تصمیم‌گیری را به تصویر بکشید.

**راه حل**

مدل درخت تصمیم را به فرمت DOT صادر کنید، سپس آن را به تصویر بکشید:





**بحث**

یکی از مزایای طبقه‌بندی‌کننده‌های درخت تصمیم این است که می‌توانیم کل مدل آموزش‌دیده را تجسم کنیم. درخت‌های تصمیم‌گیری یکی از قابل تفسیرترین مدل‌ها در یادگیری ماشین هستند. در راه حل خود، مدل آموزش دیده‌ی خود را به قالب DOT (یک زبان توصیف گراف) صادر کردیم و سپس از آن برای ترسیم نمودار استفاده کردیم.

اگر به گره ریشه نگاه کنیم، می­بینیم که قانون تصمیم‌گیری این است که اگر عرض گلبرگ کمتر یا مساوی 0.8 باشد، به شاخه سمت چپ بروید. اگر نه، به شاخه سمت راست بروید. همچنین می­توانیم شاخص ناخالصی جینی (۰.۶۶۷)، تعداد مشاهدات (۱۵۰) و تعداد مشاهدات در هر کلاس (۵۰، ۵۰، ۵۰) را ببینیم و کلاس مشاهدات در صورت توقف در آن گره (setosa) پیش‌بینی می‌شود. همچنین می­‌توان مشاهده کرد که در گره یادگیرنده، یک قاعده تصمیم‌گیری منفرد (petal width (cm) <= 0.8) قادر به شناسایی کامل تمام مشاهدات کلاس setosa است. علاوه بر این، با یک قانون تصمیم‌گیری دیگر با همان ویژگی (petal width (cm) <= 1.75) درخت تصمیم می­تواند 144 مشاهده از 150 مشاهده را به درستی دسته‌بندی کند.

اگر بخواهیم از درخت تصمیم‌گیری در برنامه­ها یا گزارش­های دیگر استفاده کنیم، می­توانیم به راحتی تصویرسازی را به صورت PDF یا یک تصویر PNG صادر کنیم.









در حالی که این راه حل یک دسته‌بندی‌کننده‌ی درخت تصمیم را به تصویر می­کشد، می­تواند به راحتی برای به تصویر کشیدن یک رگرسیون درخت تصمیم نیز استفاده شود.

نکته: کاربران macOS ممکن است برای اجرای کد قبلی مجبور به نصب اپلیکیشن GraphViz باشند. این کار را می­توانید با استفاده از دستور Homebrew به این شکل brew install graphviz انجام دهید. برای آموزش نصب Homebrew به وب سایت Homebrew مراجعه کنید.

**برای مطالعه بیشتر**

* [Homebrew](https://brew.sh/)

**۱۴.۴. آموزش یک دسته‌بندی‌کننده جنگل تصادفی**

**مسئله**

شما می­خواهید یک مدل دسته‌بندی را با استفاده از "جنگل" از درختان تصمیم تصادفی آموزش دهید.

**راه حل**

یک مدل دسته‌بندی تصادفی جنگل را با استفاده ازRandomForestClassifier در scikit-learn آموزش دهید:



**بحث**

یک مشکل رایج در مورد درخت­های تصمیم این است که آنها تمایل دارند داده­های آموزشی را خیلی نزدیک به هم برازش کنند (به عنوان مثال، بیش‌برازش(overfitting)). این امر باعث استفاده‌ی گسترده از یک روش یادگیری گروهی به نام جنگل تصادفی شده است. در یک جنگل تصادفی، بسیاری از درختان تصمیم‌گیری آموزش دیده اند، اما هر درخت تنها یک نمونه بوت‌استرپ شده از مشاهدات را دریافت می­کند (یعنی یک نمونه تصادفی از مشاهدات با جایگزینی که با تعداد مشاهدات اصلی مطابقت دارد) و هر گره تنها یک زیر مجموعه از ویژگی­ها را در هنگام تعیین بهترین تقسیم در نظر می­گیرد. این جنگل از درخت­های تصمیم تصادفی (به این جهت نامش این است) به تعیین کلاس پیش‌بینی‌شده رای می­دهد. همانطور که می­بینیم با مقایسه این راه حل با دستورالعمل 14.1، RandomForestClassifier در scikit-learn مشابه به DecisionTreeClassifier عمل می­کند:





RandomForestClassifier همچنین از بسیاری از پارامترهای مشابه به DecisionTreeClassifier استفاده می­کند. به عنوان مثال می­توانیم معیار کیفیت تقسیم مورد استفاده را تغییر دهیم:



با این حال، RandomForestClassifier به عنوان یک جنگل به جای درخت تصمیم‌گیری فردی، پارامترهای خاصی دارد که یا منحصر به جنگل­های تصادفی هستند یا به ویژه، مهم هستند. اول، پارامتر max\_features تعیین‌کننده‌ی حداکثر تعداد مشخصه‌ها است که باید در هر گره در نظر گرفته شود و تعدادی آرگومان شامل اعداد صحیح (تعداد مشخصه‌ها)، شناور­ها (درصد مشخصه‌ها) و sqrt (ریشه دوم تعداد مشخصه‌ها) را می‌گیرد. به طور پیش‌فرض، max\_features روی auto تنظیم شده است که مانند sqrt عمل می­کند. دوم، پارامتر bootstrap به ما اجازه می‌دهد تا تعیین کنیم که آیا زیر مجموعه‌ی مشاهدات در نظر گرفته شده برای یک درخت با استفاده از نمونه برداری با جایگزینی (تنظیمات پیش‌فرض) یا بدون جایگزینی ایجاد می­شود. سوم، n\_estimators تعداد درخت­های تصمیم را برای قرارگیری در جنگل تنظیم می­کند. در دستور العمل 10.4، ما n\_estimators را به عنوان یک فراپارامتر در نظر گرفتیم و اثر افزایش تعداد درختان را بر روی یک معیار ارزیابی به تصویر کشیدیم. در نهایت، هرچند مختص طبقه‌بندی‌کننده‌های جنگل تصادفی نیست، چون ما به طور موثر بسیاری از مدل‌های درخت تصمیم را آموزش می‌دهیم، اغلب استفاده از تمام هسته‌های موجود با تنظیم n\_jobs=.1 مفید است.

**برای مطالعه بیشتر**

* [Random Forests، Berkeley Statistics](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

**١٤.٥. آموزش رگرسیون جنگل تصادفی**

**مسئله**

می­خواهید یک مدل رگرسیون را با استفاده از یک "جنگل" از درختان تصمیم‌گیری تصادفی آموزش دهید.

**راه حل**

یک مدل رگرسیون جنگل تصادفی را با استفاده از RandomForestRegressor در scikit-learn آموزش دهید:



**بحث**

درست مانند این که چگونه می­توانیم جنگلی از دسته‌بندی‌کننده­های درخت تصمیم بسازیم، می­توانیم جنگلی از رگرسیون کننده­های درخت تصمیم بسازیم که در آن هر درخت از یک زیر مجموعه‌ی خود راه اندازی شده که از مشاهدات استفاده می­کند و در هر گره قانون تصمیم­گیری تنها یک زیر مجموعه از ویژگی­ها را در نظر می­گیرد. مانند RandomForestClassifier ما پارامترهای مهم و خاصی داریم:

max\_features حداکثر تعداد ویژگی­هایی را که باید در هر گره در نظر گرفته شود، تعیین می­کند. پیش‌فرض آن ویژگی­ها هستند، که در آن p تعداد کل ویژگی­ها است.

bootstrap تعیین می­کند که آیا نمونه‌برداری با جایگزینی انجام شود یا خیر. پیش‌فرض True است.

n\_estimators تعداد درخت­های تصمیم را برای ساخت تعیین می­کند. پیش‌فرض 10 است.

**برای مطالعه بیشتر**

* [RandomForestRegressor، scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

**١٤.٦.شناسایی ویژگی­های مهم در جنگل­های تصادفی**

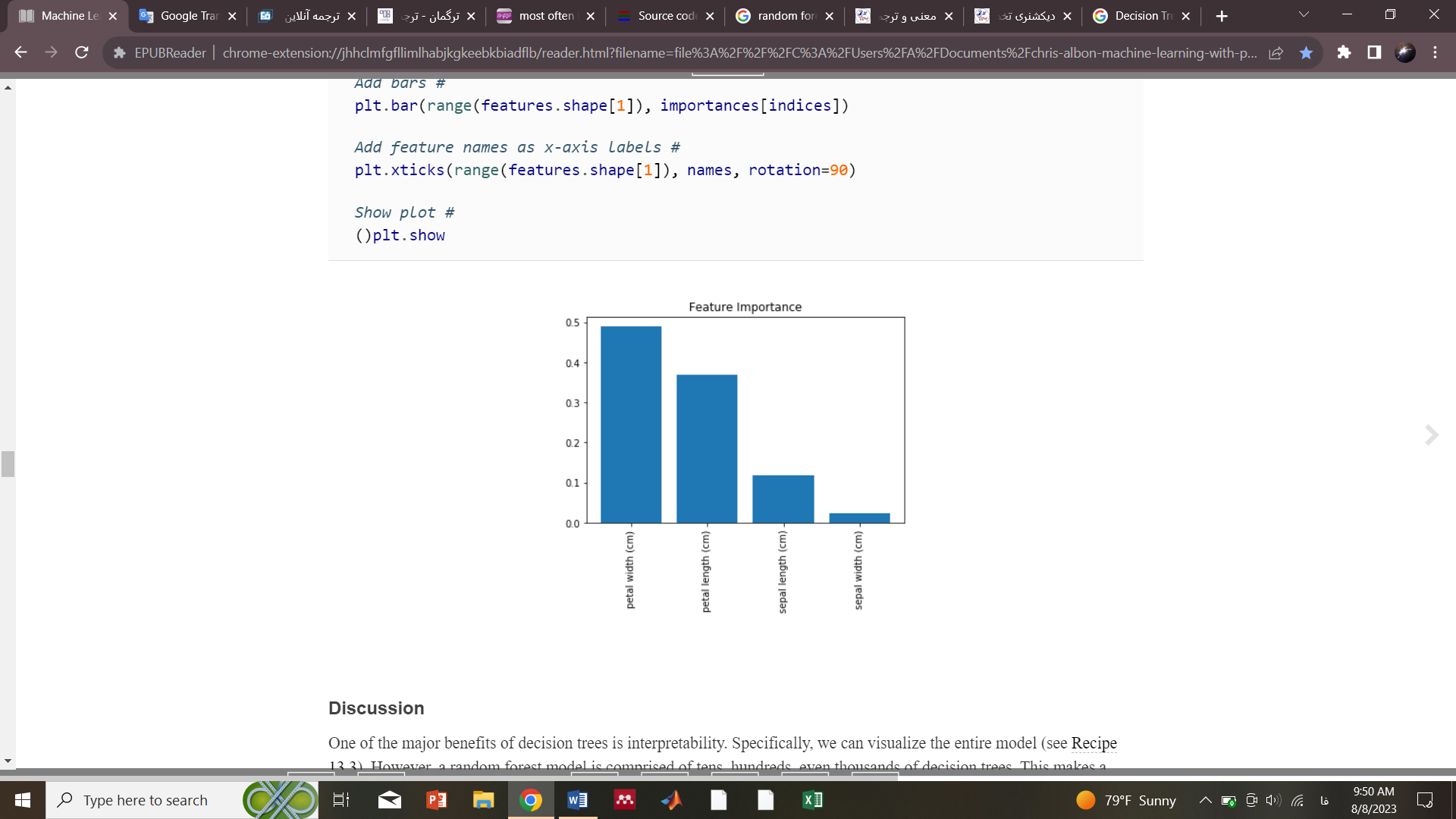
**مسئله**

شما باید بدانید که کدام ویژگی­ها در یک مدل جنگل تصادفی بیشترین اهمیت را دارند.

**راه حل**

اهمیت هر ویژگی را محاسبه کنید و به تصویر بکشید:





**بحث**

یکی از مزایای اصلی درخت تصمیم، تفسیرپذیری است. به طور خاص، ما می­توانیم کل مدل را تصویر کنیم (به دستور العمل 13.3 مراجعه کنید). با این حال، یک مدل جنگل تصادفی از ده­ها، صدها و حتی هزاران درخت تصمیم تشکیل شده است. این امر تجسم ساده و شهودی یک مدل جنگل تصادفی را غیر عملی می‌کند. با این حال، گزینه دیگری وجود دارد: ما می­توانیم اهمیت نسبی هر ویژگی را با هم مقایسه کنیم (و به تصویر بکشیم).

در دستورالعمل ۱۳.۳، ما یک مدل دسته‌بندی‌کننده‌ی درخت تصمیم‌گیری را تجسم کردیم و دیدیم که قوانین تصمیم‌گیری تنها براساس عرض گلبرگ قادر به دسته‌بندی صحیح بسیاری از مشاهدات هستند. به طور مستقیم می‌توان گفت که این به این معنی است که عرض گلبرگ ویژگی مهمی در دسته‌بندی‌کننده‌ی ما است. به طور رسمی­تر، ویژگی­هایی با شکاف­هایی که میانگین کاهش بیشتری در ناخالصی دارند (به عنوان مثال، ناخالصی جینی یا آنتروپی در دسته‌بندی‌کننده­ها و واریانس در رگرسیون­ها) مهم­تر در نظر گرفته می­شوند.

با این حال، در مورد اهمیت ویژگی­ها باید دو نکته را در نظر داشته باشید. ابتدا، scikit-learn نیاز دارد که ما ویژگی­های قطعی اسمی را به چندین ویژگی باینری تقسیم کنیم. این امر باعث گسترش اهمیت آن ویژگی در تمام ویژگی‌های باینری می‌شود و اغلب می‌تواند باعث شود که هر یک از ویژگی‌ها، بی‌اهمیت به نظر برسد، حتی زمانی که ویژگی دسته‌بندی اسمی اصلی بسیار مهم است. ثانیا، اگر دو ویژگی به شدت با هم مرتبط باشند، یک ویژگی اهمیت بیشتری پیدا می­کند و ویژگی دیگر اهمیت بسیار کمتری پیدا می­کند، که اگر در نظر گرفته نشود، پیامدهایی برای تفسیر دارد. درscikit-learn  دسته‌بندی و رگرسیون درختان تصمیم‌گیری و جنگل­های تصادفی می‌توانند اهمیت نسبی هر ویژگی را با استفاده از روش feature\_importances\_ گزارش کنند:





هر چه عدد بالاتر باشد، ویژگی مهم­تر است (مجموع امتیازات اعتبار به ۱ می‌رسد). با ترسیم این مقادیر می­توانیم تفسیرپذیری را به مدل­های جنگل تصادفی خود اضافه کنیم.

**۱۴.۷. انتخاب ویژگی­های مهم در جنگل­های تصادفی**

**مسئله**

شما باید انتخاب ویژگی را در یک جنگل تصادفی انجام دهید.

**راه حل**

ویژگی­های مهم را شناسایی کنید و مدل را تنها با استفاده از مهمترین ویژگی­ها بازآموزی کنید:



**بحث**

شرایطی وجود دارد که ممکن است بخواهیم تعداد ویژگی­های مدل خود را کاهش دهیم. برای مثال، ممکن است بخواهیم واریانس مدل را کاهش دهیم یا ممکن است بخواهیم تفسیرپذیری را تنها با در نظر گرفتن مهم­ترین ویژگی­ها بهبود بخشیم.

در scikit-learn ما می­توانیم از یک جریان کار ساده و دو مرحله‌ای برای ایجاد یک مدل با ویژگی­های کاهش یافته استفاده کنیم. ابتدا یک مدل جنگل تصادفی را با استفاده از تمام ویژگی­ها آموزش می­دهیم. سپس از این مدل برای شناسایی مهم­ترین ویژگی­ها استفاده می­کنیم. در مرحله بعد، یک ماتریس ویژگی جدید ایجاد می­کنیم که فقط این ویژگی­ها را شامل می­شود. ما در راه حل خود از روش SelectFromModel برای ایجاد یک ماتریس ویژگی استفاده کرده‌ایم که تنها شامل ویژگی­هایی با اهمیت بیشتر یا برابر با مقدار threshold است. در نهایت، ما یک مدل جدید با استفاده از این ویژگی­ها ایجاد کردیم.

لازم به ذکر است که دو هشدار برای این رویکرد وجود دارد. مورد اول اینکه ویژگی‌های دسته‌بندی اسمی که یک‌بار کدگذاری شده‌اند، اهمیت ویژگی را در بین ویژگی‌های باینری کاهش می‌دهند. دوم، اهمیت ویژگی‌های بسیار هم‌بسته به طور موثر به یک ویژگی اختصاص داده می‌شود و به طور یکنواخت در هر دو ویژگی توزیع نمی‌شود.

**برای مطالعه بیشتر**

* [Variable selection using Random Forests، Robin Genuer، Jean.Michel Poggi، Christine Tuleau.Malot](http://bit.ly/2FvG70D)

**١٤.٨.مدیریت کلاس‌های نامتعادل**

**مسئله**

یک بردار هدف با کلاس­های بسیار نامتعادل دارید و می­خواهید یک مدل جنگل تصادفی را آموزش دهید.

**راه حل**

درخت تصمیم یا مدل جنگل تصادفی را با class\_weight='balanced’ آموزش دهید:

**بحث**

کلاس­های نامتعادل مشکلی رایج در زمان انجام یادگیری ماشین در دنیای واقعی هستند. وجود کلاس­های نامتعادل می­تواند عملکرد مدل ما را کاهش دهد. در واقع ما در مورد چند استراتژی برای رسیدگی به کلاس­های نامتعادل در طول پیش‌پردازش در دستور ۱۷.۵ صحبت کرده‌ایم. با این حال، بسیاری از الگوریتم‌های یادگیری در scikit-learn با روش‌های داخلی برای اصلاح کلاس‌های نامتعادل همراه هستند. ما می‌توانیم RandomForestClassifier را برای اصلاح کلاس‌های نامتعادل با استفاده از پارامتر class\_weight تنظیم کنیم. اگر یک فرهنگ لغت به شکل نام کلاس و وزن­های دلخواه مربوطه (به عنوان مثال{'male': 0.2, 'female': 0.8}) ارائه شود، RandomForestClassifier کلاس­ها را بر این اساس وزن‌دهی خواهد کرد. با این حال، اغلب balanced یک آرگومان مفیدتر است که در آن کلاس­ها به طور خودکار متناسب با تعداد دفعاتی که در داده­ها ظاهر می­شوند، وزن‌دهی می­شوند:

که در آن wj وزن کلاس j، n تعداد مشاهدات، nj تعداد مشاهدات در کلاس j و k تعداد کل کلاس­ها است. به عنوان مثال، در حل این مسئله، ما به ترتیب 2 کلاس (k)، 110 مشاهده (n) و 10 و 100 مشاهده در هر کلاس (nj) داریم. اگر کلاس­ها را با استفاده از 'class\_weight='balanced وزن‌دهی کنیم، کلاس کوچکتر وزن بیشتری دارد:





در حالی که کلاس بزرگتر وزن کمتری دارد:





**١٤.٩.کنترل اندازه درخت**

**مسئله**

شما می­خواهید به صورت دستی ساختار و اندازه‌ی درخت تصمیم را تعیین کنید.

**راه حل**

از پارامترهای ساختار درخت در الگوریتم‌های یادگیری مبتنی بر درخت scikit-learn استفاده کنید:



**بحث**

الگوریتم‌های یادگیری مبتنی بر درخت scikit-learn دارای تکنیک‌های متنوعی برای کنترل اندازه‌ی درخت‌های تصمیم هستند. این موارد از طریق پارامترها قابل دسترسی هستند:

max\_depth

حداکثر عمق درخت است. اگر مقدار آن برابر با None باشد، درخت تا زمانی که همه برگ­ها خالص باشند رشد می‌کند. اگر یک عدد صحیح باشد، درخت به طور موثر تا آن عمق "هرس" می­شود.

min\_samples\_split

حداقل تعداد مشاهدات در یک گره قبل از تقسیم آن گره است. اگر یک عدد صحیح به عنوان آرگومان عرضه شود حداقل خام را تعیین می­کند، در حالی که اگر یک عدد اعشاری به آن داده شود حداقل درصد کل مشاهدات در نظر گرفته می‌شود.

min\_samples\_leaf

حداقل تعداد مشاهداتی که باید در یک برگ باشد را مشخص می‌کند. از همان آرگومان­های min\_samples\_split استفاده می­کند.

max\_leaf\_nodes

حداکثر تعداد برگ را مشخص می‌کند.

min\_impurity\_split

حداقل کاهش ناخالصی مورد نیاز قبل از انجام تقسیم است.

در حالی که دانستن وجود این پارامترها مفید است، به احتمال زیاد ما فقط از max\_depth و min\_impurity\_split استفاده خواهیم کرد زیرا درختان کم‌عمق­تر (که گاهی اوقات کنده نامیده می­شود) مدل­های ساده­تری هستند و بنابراین واریانس کمتری دارند.

**۱۴.۱۰. بهبود عملکرد از طریق تقویت**

**مسئله**

شما به مدلی با عملکرد بهتر از درختان تصمیم یا جنگل­های تصادفی نیاز دارید.

**راه حل**

یک مدل تقویت شده را با استفاده از AdaBoostClassifier یا AdaBoostRegressor آموزش دهید:



**بحث**

در جنگل تصادفی، یک گروه از درختان تصمیم‌گیری تصادفی، بردار هدف را پیش‌بینی می‌کنند. یک رویکرد جایگزین و اغلب قدرتمندتر، تقویت نامیده می­شود. در یک شکل از تقویت که AdaBoost نامیده می­شود، ما به طور مکرر یک سری از مدل­های ضعیف را آموزش می­دهیم (اغلب یک درخت تصمیم‌گیری کم عمق، گاهی اوقات یک کنده نامیده می­شود). هر تکرار، اولویت بالاتری به مشاهداتی می­دهد که مدل قبلی به طور نادرست پیش‌بینی کرده بود. به طور خاص، در AdaBoost:

1. به هر مشاهده‌ی xi، مقدار وزن اولیه اختصاص دهید، ، که در آن n تعداد کل مشاهدات در داده­ها است.
2. یک مدل "ضعیف" روی داده­ها آموزش دهید.
3. برای هر مشاهده:
4. اگر مدل ضعیف xi را به درستی پیش‌بینی کند، wi افزایش می­یابد.
5. اگر مدل ضعیف xi را اشتباه پیش‌بینی کند، wi کاهش می­یابد.
6. یک مدل ضعیف جدید آموزش دهید که در آن مشاهدات با wi بزرگتر، اولویت بیشتری داشته‌باشند.
7. مراحل ۴ و ۵ را تکرار کنید تا زمانی که داده­ها کاملا پیش‌بینی شوند یا تعدادی از مدل­های ضعیف آموزش داده شوند.

نتیجه نهایی یک مدل تجمعی است که در آن مدل­های ضعیف فردی بر مشاهدات دشوارتر (از دیدگاه پیش‌بینی) تمرکز می­کنند. در scikit-learn، می‌توانیم AdaBoost را با استفاده از AdaBoostClassifier یا AdaBoostRegressor پیاده‌سازی کنیم. مهمترین پارامترها نیز base\_estimator، n\_estimators و learning\_rate هستند:

base\_estimator الگوریتم یادگیری برای آموزش مدل­های ضعیف است. این مورد تقریبا همیشه نیازی به تغییر نخواهد داشت زیرا تا کنون رایج­ترین یادگیرنده­ای که با AdaBoost استفاده می­شود یک درخت تصمیم‌گیری است. این آرگومان، آرگومان پیش‌فرض این پارامتر است.

n\_estimators تعداد مدل­هایی است که باید به صورت تکراری آموزش داده شوند.

Learn\_rate سهم هر مدل در وزن و پیش‌فرض آن 1 است. کاهش سرعت یادگیری به این معنی است که وزن­ها تا حد کمی افزایش یا کاهش خواهند یافت و مدل را مجبور به آموزش کندتر می­کنند (اما گاهی اوقات منجر به امتیاز عملکرد بهتر می­شوند).

ضرر، منحصر به AdaBoostRegressor است و عملکرد کاهش را برای به‌روزرسانی وزن‌ها تنظیم می‌کند. این مورد به طور پیش‌فرض روی یک تابع زیان خطی است اما می‌توان آن را به مربع یا نمایی تغییر داد.

**برای مطالعه بیشتر**

* [Explaining AdaBoost، Robert E. Schapire](http://bit.ly/2FCS30E)

**۱۴.۱۱. ارزیابی جنگل­های تصادفی با خطاهای خارج از کیسه**

**مسئله**

شما باید یک مدل جنگل تصادفی را بدون استفاده از اعتبارسنجی متقاطع ارزیابی کنید.

**راه حل**

امتیاز out-of-bag را برای مدل محاسبه کنید:



****

**بحث**

در جنگل­های تصادفی، هر درخت تصمیم‌گیری با استفاده از زیر مجموعه‌ای از مشاهدات بوت‌استرپ شده آموزش داده می­شود. این بدان معناست که برای هر درخت یک زیر مجموعه‌ی جداگانه از مشاهدات وجود دارد که برای آموزش از آن درخت استفاده نمی‌شود. به این مشاهدات خارج از کیسه (OOB) می­گویند. ما می­توانیم از مشاهدات OOB به عنوان یک مجموعه‌ی تست برای ارزیابی عملکرد جنگل تصادفی خود استفاده کنیم. برای هر مشاهده، الگوریتم یادگیری مقدار واقعی مشاهده را با پیش‌بینی زیر مجموعه‌ای از درختان که با استفاده از آن مشاهده آموزش ندیده‌اند، مقایسه می­کند. امتیاز کلی محاسبه می­شود و یک معیار واحد از عملکرد یک جنگل تصادفی را فراهم می­کند. تخمین امتیاز OOB جایگزینی برای اعتبارسنجی متقاطع است.

در scikit-learn، با تنظیم oob\_score=True در شیء جنگل تصادفی (یعنی RandomForestClassifier) می‌توانیم امتیازهای یک جنگل تصادفی را OOB کنیم. امتیاز را می­توان با استفاده از oob\_score\_ بازیابی کرد.