**فصل 14. درخت‌ها 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**

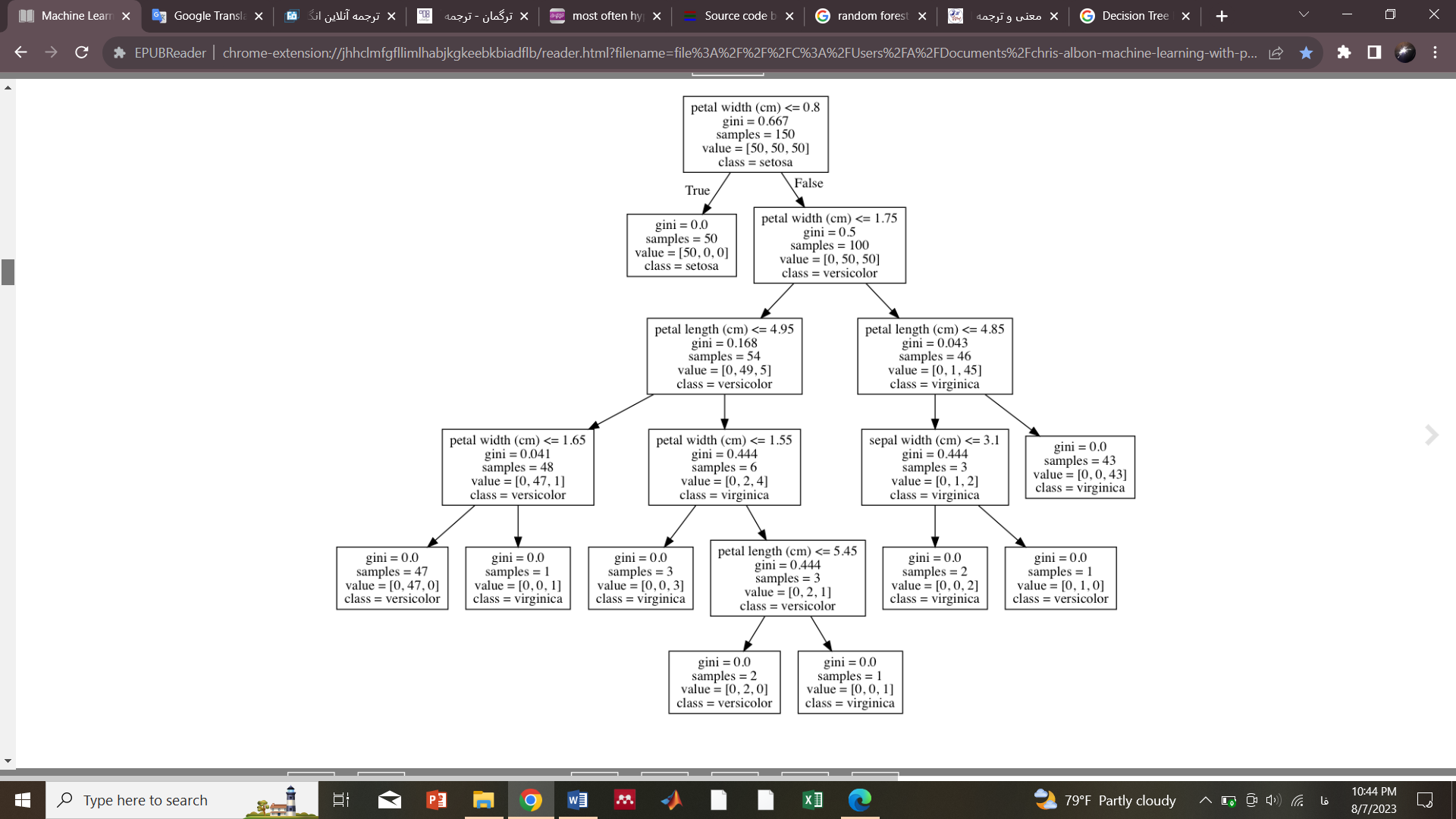
**مسئله**

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)

**14.6 ارزیابی جنگل‌های تصادفی[[1]](#footnote-1) با خطاهای خارج از کیسه[[2]](#footnote-2)**

**مسئله**

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

**راه‌حل**

امتیاز خارج از کیف مدل را محاسبه کنید:



**بحث**

در جنگل‌های تصادفی، هر درخت تصمیم با استفاده از زیرمجموعه‌ای از مشاهدات راه‌اندازی، آموزش داده می‌شود. این بدان معناست که برای هر درخت یک زیر مجموعه جداگانه از مشاهدات وجود دارد که برای آموزش آن درخت استفاده نمی‌شود. به این مشاهدات خارج از کیسه (OOB) می‌گویند. ما می‌توانیم از مشاهدات OOB به عنوان یک مجموعه تست برای ارزیابی عملکرد جنگل تصادفی خود استفاده کنیم.

برای هر مشاهده، الگوریتم یادگیری، ارزش واقعی مشاهدات را با پیش‌بینی زیرمجموعه‌ای از درختانی که با استفاده از آن مشاهده آموزش ندیده‌اند، مقایسه می‌کند. امتیاز کلی محاسبه می‌شود و یک معیار واحد از عملکرد یک جنگل تصادفی را ارائه می‌دهد. تخمین امتیاز OOB جایگزینی برای اعتبارسنجی متقابل است.

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

**14.7 شناسایی ویژگی­های مهم در جنگل­های تصادفی**

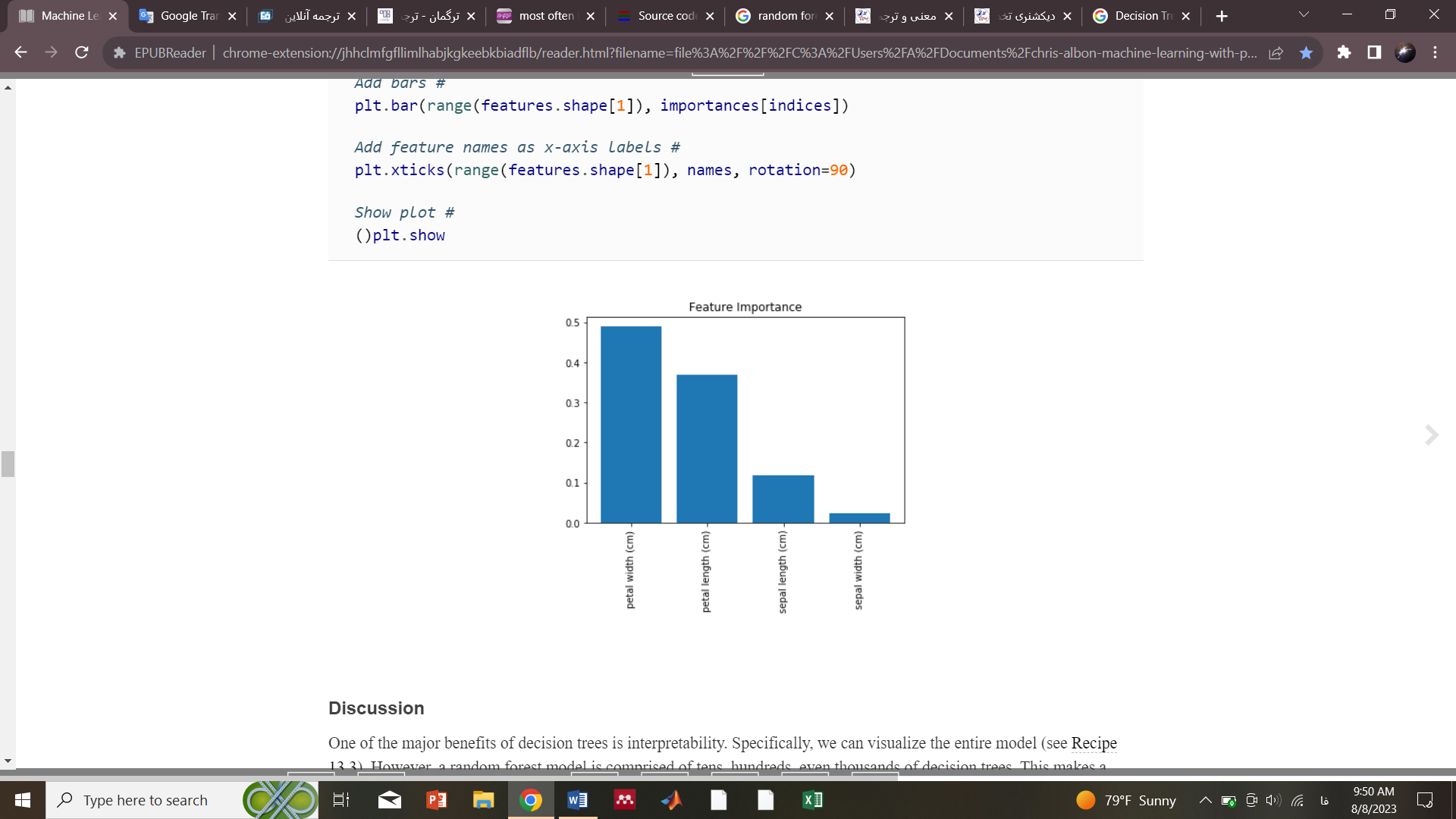
**مسئله**

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.

**14.8 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)

**14.9 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:





**14.10 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.

**۱۴.۱1. 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)

**14.12 آموزش مدل XGBoost**

**مسئله**

شما باید یک مدل درختی با قدرت پیش بینی بالا آموزش دهید.

**راه‌حل**

از کتابخانه xgboost پایتون استفاده کنید:



**بحث**

XGBoost (که مخفف Extreme Gradient Boosting است) یک الگوریتم تقویت گرادیان بسیار محبوب در فضای یادگیری ماشینی است. اگرچه همیشه یک مدل مبتنی بر درخت نیست، اما اغلب برای مجموعه‌ای از درخت‌های تصمیم استفاده می‌شود. به دلیل موفقیت گسترده در وب سایت مسابقه یادگیری ماشین Kaggle، محبوبیت زیادی به دست آورد و از آن زمان به بعد الگوریتمی‌قابل اعتماد برای بهبود عملکرد فراتر از جنگل‌های تصادفی معمولی یا ماشین‌های تقویت شده گرادیان بوده است.

اگرچه XGBoost به دلیل محاسباتی فشرده شناخته شده است، بهینه سازی عملکرد محاسباتی (مانند پشتیبانی از GPU) در چند سال گذشته، تکرار سریع با XGBoost را به طور قابل توجهی آسان کرده است، و زمانی که عملکرد آماری الزامی‌است، به عنوان یک الگوریتم رایج انتخاب می‌شود.

**همچنین ببینید:**

* [مستندات XGBoost](https://xgboost.readthedocs.io/en/stable/)

**14.13 بهبود عملکرد بلادرنگ با LightGBM**

**مسئله**

شما باید یک مدل مبتنی بر درخت تقویت شده با گرادیان را آموزش دهید که از نظر محاسباتی بهینه شده باشد.

**راه‌حل**

از کتابخانه ماشینی تقویت شده با گرادیان lightgbm استفاده کنید:



**بحث**

کتابخانه lightgbm برای ماشین‌های تقویت‌شده‌ی گرادیان استفاده می‌شود و برای زمان آموزش، استنتاج و پشتیبانی GPU بسیار بهینه شده است. به عنوان یک نتیجه از کارایی محاسباتی آن، اغلب در تولید و در تنظیمات در مقیاس بزرگ استفاده می‌شود. اگرچه استفاده از مدل‌های Sikit-Learn معمولا آسان‌تر است، برخی از کتابخانه‌ها، مانند lightgbm، می‌توانند زمانی مفید باشند که داده‌های بزرگ داشته باشیم یا از لحاظ زمان‌های آموزشی دقیق برای مدل، محدود شده‌اید.

**همچنین ببینید:**

* [مستندات LightGBM](https://lightgbm.readthedocs.io/en/latest/)
* [مستندات CatBoost (یکی دیگر از کتابخانه‌های بهینه شده برای GBM)](https://catboost.ai/en/docs/)

1. - Random Forests [↑](#footnote-ref-1)
2. - Out-of-Bag Erros [↑](#footnote-ref-2)