درخت ها در یادگیری ماشین

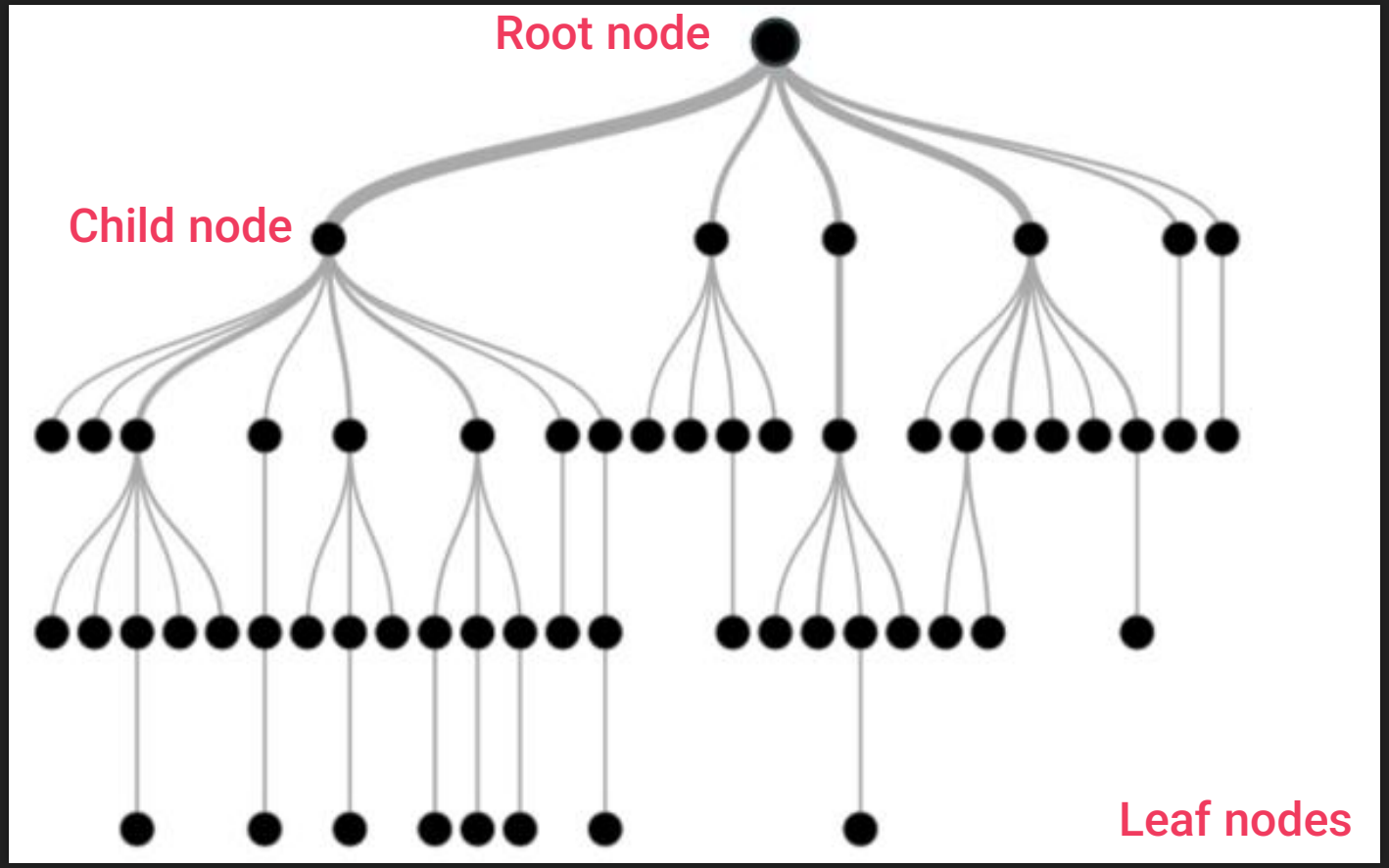
دوره پایتون و یادگیری ماشین

</Monolearn>

درخت تصمیم یا Decision Tree

- یک مدل یادگیری ماشین است که هم برای عمل Classification و هم Regression استفاده می شود.
- در این روش، فرآیند تصمیم گیری و یادگیری بر روی داده ها، به شکل یک درخت انجام می شود.
- این الگوریتم، ویژگی های مجموعه داده مورد نظر را از طریق استفاده از یک Cost function تقسیم بندی می کند.
- یکی از پارامتر های مهم این الگوریتم، عمق درخت است که با تنظیم مناسب آن می توان از Overfitting جلوگیری کرد.
- درخت در هنگام یادگیری، رشد کرده و ویژگی های نامرتبط با مسئله را در خود جای می دهد، از این رو طی عملیاتی با عنوان هرس کردن یا Pruning سعی می شود تا شاخه های اضافه (تصمیم های نامناسب گرفته شده بر روی داده ها) را ،طی فرآیند بهینه سازی، حذف کند.

ساختار کلی Decision Tree



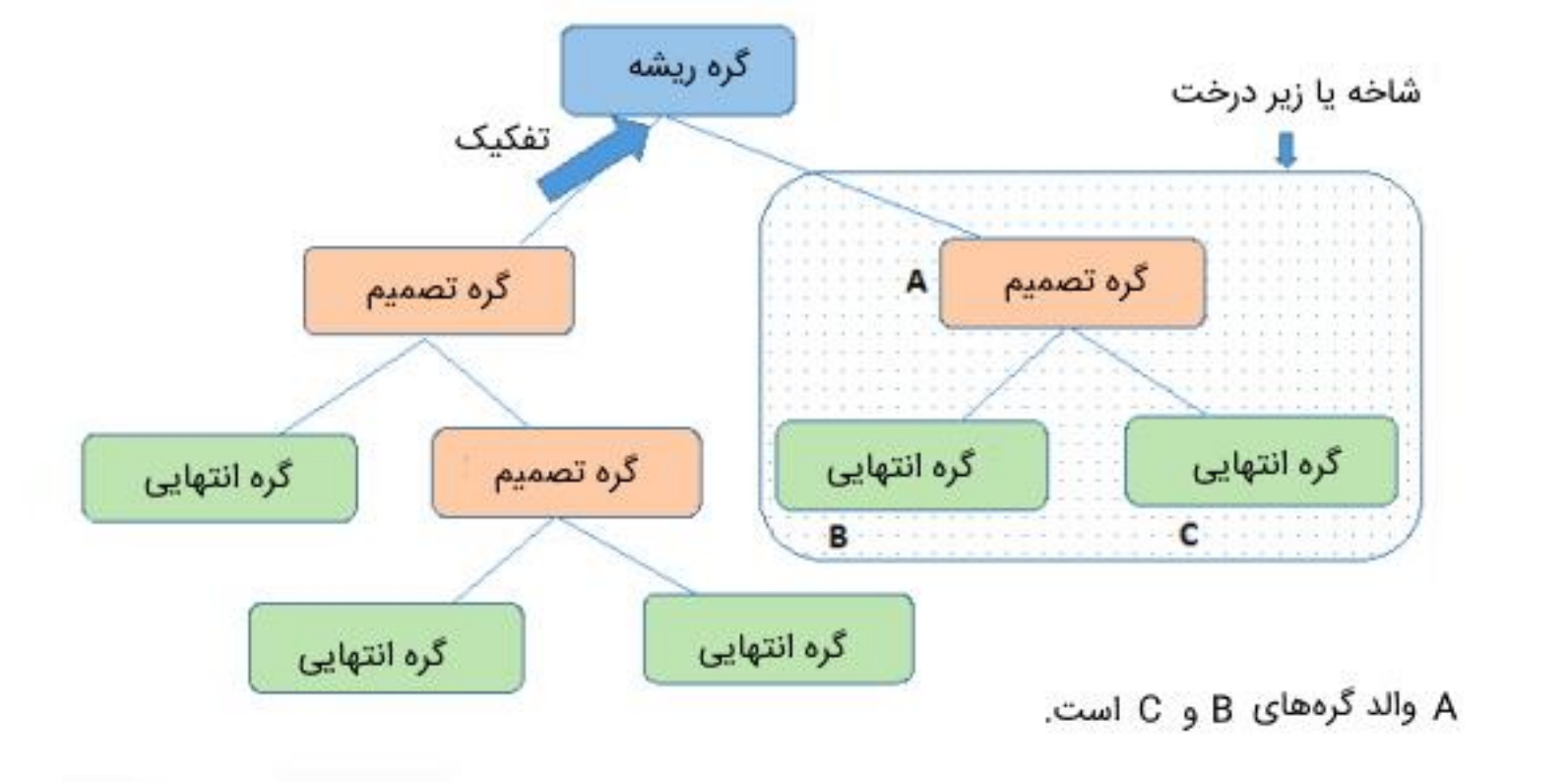
درخت تصمیم یا Decision Tree

- این الگوریتم، سادگی مناسبی برای نمایش فرآیند تصمیم گیری دارد و از این رو می توان از آن استفاده کرد.
- هر چند این سادگی می تواند با افزایش تصمیمات و افزایش عمق درخت کاسته شود.
- از طرفی، الگوریتم درخت تصمیم به سادگی به مشکل Overfitting دچار می شود.

برخی اصطلاحات در Decision Tree

- Root Node: گره ریشه، اولین و بالاترین گره درخت است. که گره نشان دهنده ی داده های مسئله است و می تواند به دو یا چند مجموعه ی همگن (Homogeneous) تقسیم شود.
- عمل Splitting: تفکیک یا تقسیم بندی، پردازشی برای تقسیم کردن گره ها به دو یا چند زیر گره (Sub-Node) است.
- Decision Node: هنگامی که یک زیر گره، به چند زیر گره دیگر تقسیم شود، به آن گره تصمیم می گویند.
- Leaf/Terminal Node: گره های انتهایی هر شاخه که دیگر قابلیت Split شدن ندارند (فرزند ندارند) را گره برگ یا انتهایی می گویند.
- Pruning: گاهی نیاز است که بخش هایی از درخت حذف گردد. هرس کردن در واقع حذف کردن برخی از گره های تصمیم است و به نوعی عکس عمل Splitting است.
- Branch/Sub-Tree: شاخه یا زیردرخت، به تمامی زیر بخش های درخت تصمیم، شاخه یا زیردرخت گفته می شود.
- Parent and Child: گره ای که به زیر گره های دیگر تقسیم می شود، گره ی والد (یا پدر) و زیر گره های حاصل از آن فرزند نامیده می شوند.

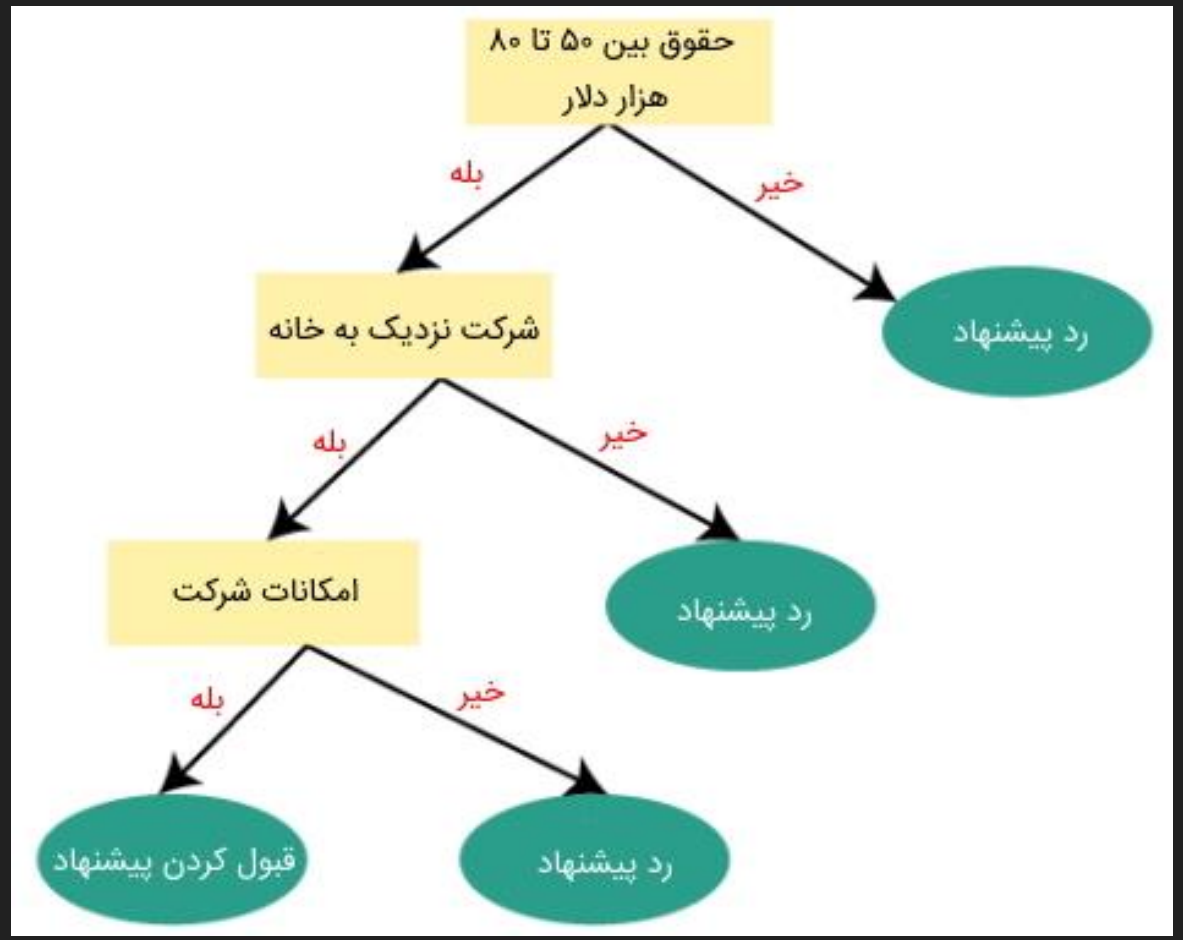
برخی اصطلاحات Decision Tree



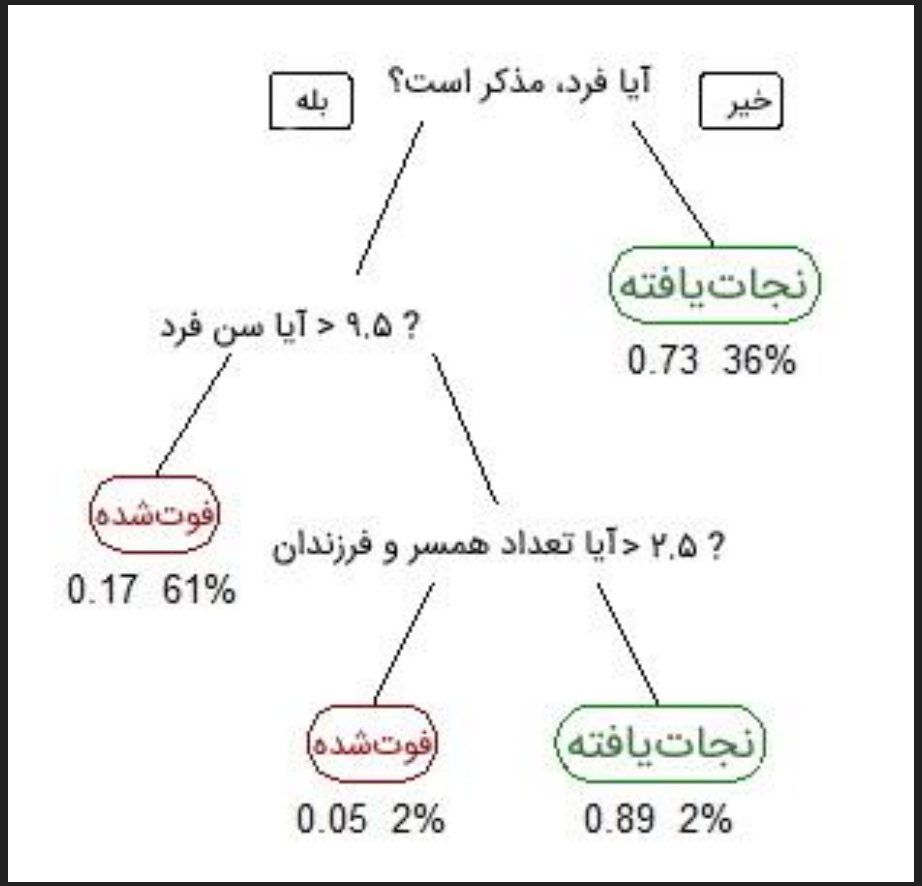
مراحل کار در Decision Tree

- مرحله اول: شروع روند کار الگوریتم درخت تصمیم از گره ریشه آغاز می‌شود که شامل مجموعه داده کامل مسئله است.
- مرحله دوم: با استفاده از روش «سنجیدن انتخاب ویژگی» (Attribute Selection Measure | ASM) بهترین ویژگی در مجموعه داده انتخاب می‌شود.
- مرحله سوم: تقسیم کردن گره ریشه به زیرمجموعه‌هایی که شامل مقادیر مناسب و ممکن برای بهترین ویژگی‌ها باشند.
- مرحله چهارم: تولید گره درخت تصمیمی که شامل بهترین ویژگی‌ها باشد.
- مرحله پنجم: با استفاده از زیرمجموعه‌های ایجاد شده از مجموعه داده در مرحله سوم این رویکرد، درخت‌های تصمیم جدید به صورت بازگشتی ایجاد می‌شوند. این روند تا جایی ادامه دارد که دیگر نمی‌توان گره‌ها را بیشتر طبقه‌بندی کرد و گره نهایی به عنوان گره برگ یا انتهایی به دست می‌آید.

یک مثال ساده از Decision Tree



مثال کشتی تایتانیک در Decision Tree



سنجش انتخاب ویژگی در Decision Tree

- یکی از موضوعات مهم در درخت تصمیم نحوه ی تصمیم گیری برای انتخاب بهترین ویژگی در گره ی ریشه و گره های فرعی است که برای حل این موضوع، الگوریتمی وجود دارد که به آن معیار/سنجش انتخاب ویژگی یا Attribute Selection Measure یا ASM گفته می شود.

- روش اول: بدست آوردن اطلاعات (Information Gain) (در برخی منابع به این روش، روش Entropy هم گفته می شود.)
- روش دوم: شاخص جینی (Gini Index)

روش Entropy در Decision Tree

- این روش بر اساس تغییرات آنتروپی پس از تقسیم بندی یک مجموعه داده بر اساس ویژگی ها عمل می کند.
- این روش محاسبه می کند که یک ویژگی چه مقدار اطلاعات درباره ی یک کلاس به ما می دهد. طبق مقادیر اطلاعات بدست آمده، گره ها تقسیم می شوند و درخت تصمیم ساخته خواهد شد. درخت تصمیم همواره سعی دارد تا بیشترین میزان اطلاعات را بدست آورد و برای همین آن گره های ویژگی که امکان دسترسی به اطلاعات بیشتر را فراهم می کنند، زودتر بررسی می کند.
- همچنین آنتروپی که معیاری برای اندازه گیری ناخالصی در یک ویژگی است و تصادفی بودن داده ها را بیان می کند.
- رابطه ی زیر فرمول بدست آوردن اطلاعات است:

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

- در این مقاله از وبسایت programstore می توانید یک مثال از نحوه محاسبه Information Gain را مشاهده کنید.

روش Gini Index در Decision Tree

- شاخص جینی یا Gini Index معیاری از میزان ناخالصی نسبت به خلوص ویژگی ها را بدست می آورد که در الگوریتم CART (Classification and Regression Tree) که یک درخت تصمیم برای مسائل Classification و Regression است استفاده می شود.
- Gini Index تنها برای مسائل Classification در الگوریتم CART کاربرد دارد.
- برای مسائل Regression در الگوریتم CART به جای Gini Index از تابع هزینه Sum Squared Error استفاده می شود.
- ویژگی ای که شاخص جینی کمتری داشته باشد نسبت به ویژگی ای که شاخص جینی بیشتری دارد، در ساخت درخت تصمیم ارجحیت دارد و زودتر انتخاب خواهد شد.
- فرمول محاسبه Gini Index به این صورت است:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

روش Gini Index در Decision Tree

- در فرمول Gini، اگر همه ی داده ها متعلق به یک کلاس باشند، گفته می شود که داده ها Pure هستند.
- مقدار Gini همواره بین ۰ و ۱ است. در صورتی که صفر باشد یعنی همگی داده ها متعلق به یک کلاس هستند و اگر ۱ باشد یعنی داده ها متعلق به کلاس های مختلف هستند.
- همچنین، Pi در این فرمول بیانگر احتمال متعلق بودن یک داده به کلاس مورد بررسی است.
- در این مقاله از وبسایت Medium می توانید یک مثال از نحوه ی محاسبه Gini را مشاهده کنید.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

تقسیم باینری بازگشت در Decision Tree

- برای تقسیم کردن شاخه ها یا اعمال Splitting نیاز به مکانیزمی داریم که با استفاده از تقسیم باینری بازگشت می توان این تقسیم بندی ها را به گونه ای انجام داد که از افزایش ابعاد درخت جلوگیری کرد.
- در این روش، در نقاط مختلف تقسیم، یک تابع هزینه بررسی می شود و تقسیم با بهترین مقدار هزینه یا همان کمترین مقدار هزینه انجام می گردد.
- در مثال کشتی تایتانیک، در هنگام شروع الگوریتم، تعداد زیادی ویژگی داریم که برای همه ی ویژگی ها مقدار تابع هزینه محاسبه می شود، و سپس ویژگی "جنسیت" که در این مثال کمترین میزان هزینه را دارد انتخاب می شود.
- این الگوریتم در واقع یک الگوریتم بازگشتی و از نوع Greedy است، چرا که سعی می کند در گره ریشه بهترین ویژگی پیش بینی کننده یا Classifier را انتخاب کند.

هزینه ی تقسیم در Decision Tree

- در مسائل Regression از فرمول زیر محاسبه می شود:

$$\text{Sum of Squares Error} \rightarrow \quad SSE = \sum (y - y')^2$$

- در مسائل Classification از فرمول زیر محاسبه می شود، که مقادیر بیشتر برای Gini Gain به معنای بهتر بودن آن Split یا تقسیم هست.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

توقف تقسیم در Decision Tree

- برای جلوگیری از بزرگ شدن بیش از حد درخت تصمیم و همچنین جلوگیری از مشکل Overfitting باید زمانی تقسیم کردن درخت را متوقف کرد.
- روش اول: با تنظیم حداقل تعداد ورودی های train برای استفاده در هر برگ می توان این کار را انجام داد. مثلاً در مورد تایتانیک شرط بگذاریم که فقط می توان ۱۰ مسافر را برای تصمیم گیری در مورد وضعیت زنده یا مرده بودن استفاده کرد و هر برگی که کمتر از ۱۰ مسافر داشته باشد را دیگر ادامه ندهیم.
- روش دوم: تعیین حداکثر عمق برای درخت است. حداکثر عمق، اندازه ی طولانی ترین مسیر از ریشه تا برگ است.

هرس کردن Decision Tree

- با هرس کردن می توان کارایی درخت تصمیم را افزایش داد. هرس کردن در واقع همان حذف شاخه هایی است که دارای ویژگی های کم اهمیت تر نسبت به هدف مسئله هستند. با اعمال این مورد، پیچیدگی درخت کاهش می یابد و قدرت و دقت پیش بینی الگوریتم با کاهش میزان Overfitting افزایش می یابد. همچنین، هرس کردن را می توان از ریشه و یا برگ ها آغاز کرد.
- روش هرس خطای کاهش یافته یا Reduced Error Pruning: ساده ترین نوع هرس است که از برگ ها شروع می شود و هر گره با بهترین کلاس را در برگ حذف می کند و این رویکرد تا زمانی که دقت مسئله کم نشود ادامه می یابد.
- روش هرس پیچیدگی هزینه یا Cost Complexity Pruning: این روش پیچیده تر از روش قبل است. و به کمک یک پارامتر Alpha به بررسی چگونگی حذف گره ها بر اساس اندازه ی زیر درخت ها (Sub-Tree) می پردازد. این روش با عنوان هرس ضعیف ترین پیوند یا Weakest Link Pruning هم شناخته می شود.

Decision Tree در سای کیت لرن

- در کتابخانه سای کیت لرن از DecisionTreeClassifier برای Classification می توان استفاده کرد.
- این الگوریتم هم برای مسائل دسته بندی باینری (لیبل خروجی شامل ۱ و ۱-) و هم برای دسته بندی چند کلاسه (مثلاً لیبل های ۱و ۲و ۳و۴) قابل استفاده است.

- در کتابخانه سای کیت لرن از DecisionTreeRegressor برای Regression می توان استفاده کرد.

مشکل اساسی Decision Tree

- در Decision Tree دقت با هر عمل Split افزایش می یابد و با Split های بیشتر می توان به دقت بالاتری رسید. حال در هنگام یادگیری امکان دارد که به سادگی، مدل یادگیری ماشین دچار Overfitting شود و این موضوع که چه موقع Overfitting رخ می دهد به وضوح مشخص نیست.
- باید حتماً سعی شود تا از Cross-validation جهت جلوگیری از این موضوع استفاده شود. همچنین، تنظیم بهینه هایپر پارامترها می تواند به این موضوع کمک کند.
- به دلیل وجود مشکل واریانس، درخت می توان ناپایدار باشد و تغییرات کوچکی در داده ها ممکن است منجر به ایجاد درختی کاملاً متفاوت شود.
- با افزایش تعداد ویژگی ها، پیچیدگی درخت افزایش می یابد و بهتر است در مسائل با تعداد ویژگی زیاد، ابتدا ابعاد مسئله را کاهش داد. (برای این کار می توان از الگوریتم PCA استفاده کرد.)

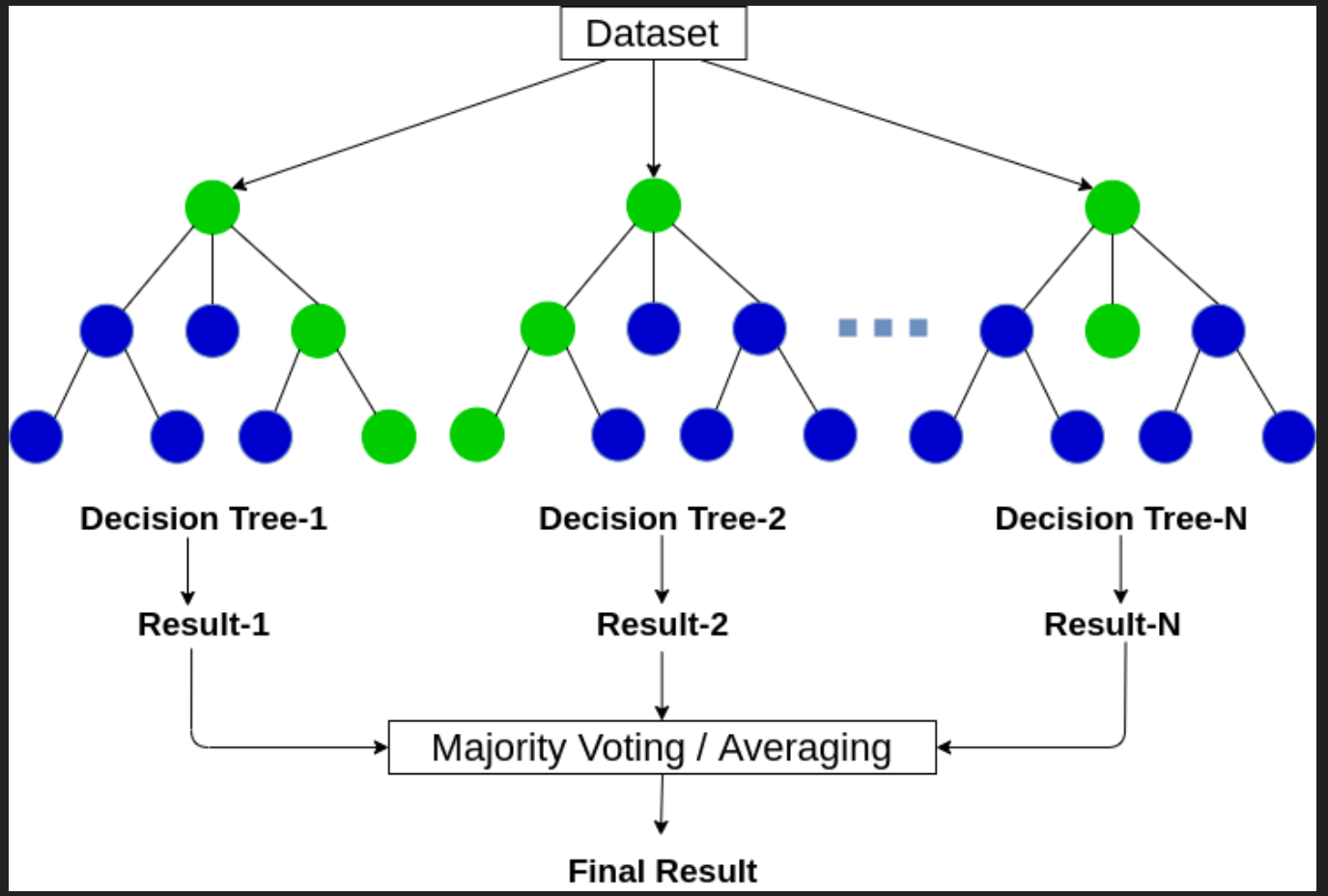
جنگل تصادفی یا Random Forest

- یک Black-box از مجموعه ای از Decision Tree هاست.
- معمولاً نتایج خوبی روی داده ها می دهد. (حتی بدون تنظیم پارامترهای بهینه!)
- می توان تعداد جنگل ها (forest) ها را تنظیم کرد. (n_estimators)
- می توان حداکثر تعداد ویژگی هایی که در ساخت Decision Tree ها استفاده می شود را مشخص کرد.
- نمی توان نحوه ی عملکرد Random این الگوریتم را تغییر داد. همچنین، نمی توان مشخص کرد که کدام ویژگی ها در هر درخت استفاده شود یا اینکه کدام داده ها در کدام درخت ها قرار بگیرد.
- در این الگوریتم، با افزایش تعداد درخت ها، دقت افزایش می یابد اما بعد از مدتی میزان دقت ثابت می شود.
- بر عکس Decision Tree، مدل ایجاد شده مشکل High Bias را ندارد و میزان واریانس مدل کم است.

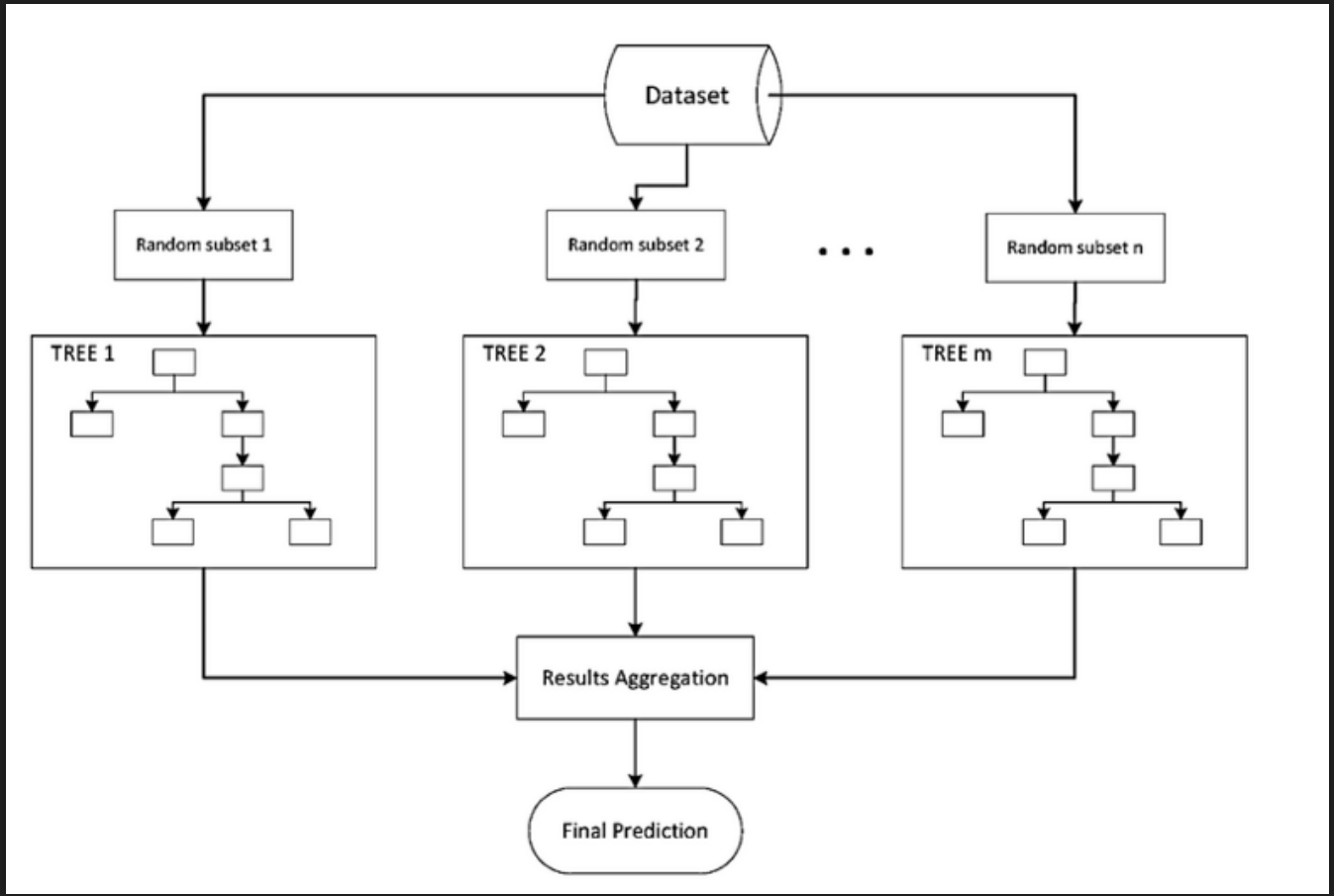
مفهوم Bagging و Ensemble در Random Forest

- در یادگیری ماشین زمانی که تعدادی مدل یادگیری ماشین را با یکدیگر ترکیب کرده و مدل بهینه جدیدی را بر اساس آن ترکیب بدست آوریم، می گویم که عمل Ensemble رخ داده است.
- در Random Forest در حقیقت شاهد Ensemble بر روی Decision Tree ها هستیم. در واقع این الگوریتم سعی می کند با ترکیب Decision Tree ها یک Random Forest بهینه ایجاد کند.
- عمل BAGGing یا Bootstrap AGGrigating قسمت اصلی نحوه ی کارکرد Random Forest است و یک نوع الگوریتم Ensemble است. به این صورت که در Random Forest داده ها به چندین زیر داده ی کوچکتر و رندوم تقسیم شده (Bootstraping) و بعد از این مرحله، یک Decision Tree بر روی هر زیر داده استفاده می شود و در نهایت خروجی همه ی Decision Tree ها طی مرحله ی AGGrigating جمع آوری شده و یک خروجی نهایی که در واقع حاصل رای گیری یا Voting در این مرحله است، بدست می آید. در شکل صفحه بعد این موضوع به وضوح قابل مشاهده است.

جنگل تصادفی یا Random Forest



مفهوم Bagging و Ensemble در Random Forest



چه موقع Decision Tree استفاده می شود؟

- برای مدل های ساده و قابل توصیف
- برای مدلی بدون پارامتر
- زمانی که نگران انتخاب ویژگی ها یا Regularization و مشکل Multi-collinearity نیستیم
- زمانی که Overfitting برایمان زیاد مهم نباشد!

چه موقع Random Forest استفاده می شود؟

- وقتی دقت بالاتری می خواهیم (و البته راحتی توصیف و بررسی مدل برایمان اهمیت ندارد.)
- زمانی که به دنبال کاهش واریانس (حل مشکل Overfitting) هستیم. لازم به ذکر است، که این مدل برای داده های جدید که تفاوت های غیرقابل منتظره ای با مقادیر داخل train دارند بهتر از Decision Tree عمل می کند. (میزان دقت الگوریتم بر روی داده های Unexpected بهتر از Decision Tree است.)
- محدودیت اصلی این الگوریتم این است که در صورت وجود داده های بسیار زیاد و همچنین درخت ها با تعداد بالا و یا عمق زیاد، الگوریتم بسیار کند اجرا می شود که برای حل برخی مسائل دنیای واقعی، استفاده از این الگوریتم را تقریباً غیر ممکن می کند.

منابع تکمیلی

https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/
https://www.kaggle.com/code/prashant111/bagging-vs-boosting
https://www.upgrad.com/blog/random-forest-vs-decision-tree/#:~:text=A%20decision%20tree%20combines%20some,forest%20model%20needs%20rigorous%20training.
https://mlu-explain.github.io/random-forest/
https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f
https://machinelearningmastery.com/bagging-and-random-forest-for-imbalanced-classification/#:~:text=Bagging%20is%20an%20ensemble%20algorithm,used%20in%20each%20data%20sample.

پایان

با تشکر از توجه تان، اوقات خوشی را برایتان آرزومندم.