هوش مصنوعی

یادگیری ماشین

حانیه ناصری - 810196573

شرح پروژه

در این پروژه از روش های یادگیری ماشین برای پیش بینی برگشتن یا برنگشتن مشتری یک فروشگاه خاص به آن فروشگاه بهره می بریم.

هر ردیف داده شامل تعدادی ستون از خصوصیات فرد مربوطه (feature) و یک ستون برگشت/عدم برگشت (target) می باشد.

در ابتدای کار در فاز نخست داده ها را پیش پردازش می کنیم و آن ها را برای مدل کردن در دو فاز بعد آماده می کنیم. در دو فاز بعد از classifier های مختلف جمعی و غیر آن، برای مدل کردن داده های آموزش (train) و سپس تست کردن مدل ساخته شده روی داده های تست (test) بهره می بریم.

classifier های به کار رفته در این پروژه در فاز نخست،

K-nearest neighbour،

Decision tree و

Logistic regression

می باشند.

در فاز دوم پروژه نیز از تعدادی از روش های یادگیری جمعی مانند

Bagging،

Random Forest و

Hard voting

بهره می بریم.

فاز صفر - پیش پردازش داده

در قسمت اول فایل data.csv در پوشه محل پروژه را می خوانیم. از همه ستون های غیر 'Is Back'،که ستون target می باشد، ستون های 'CustomerID' و 'index' را حذف می کنیم و آن ها را به در آرایه feature ها نگه می داریم. از feature ها عبارت اند از:

Total Quantity, Total Price, Country, Date (Year/Month/Day), Purchase Count

ستون های count و price و quantity مقدار عددی دارند (numeric) اما country و Is Back به فرم string می باشند و برای محاسبه information gain آن ها، باید numeric شوند.

داده country از نوع categorial می باشد و شامل نام کشور های مختلف می باشد. برای numeric کردن آن، از oneHotEncoder استفاده کردیم. Encoder ذکر شده بدین صورت عمل می کند که هر کشور را به عنوان یک ستون جدا قرار داده و در هر ردیف dataset، ستون مربوط به آن کشور را 1 کرده و ستون مابقی کشور ها را 0 می کند. هم چنین ستون های دیگر را نیز که اسم کشور نیستند با همان مقدار خود نگه می دارد.

برای encode کردن ستون target که به صورت yes/no می باشد، از LabelEncoder استفاده کردیم تا yes/no را به فرم 1/0 تبدیل کند.

ستون Date را نیز 3 بخش کرده و هر کدام از Month ،Year و Day را یک ستون مجزا کردیم.

حال به توضیح information gain می پردازیم.

هر کدام از feature هایی که بیان کردیم بر دقت پیش بینی ما تاثیر گذارند و می توانیم با دانستن مقدارشان نوع داده target را تعیین کنیم. در واقع feature ها ی یک مشتری فروشگاه اطلاعاتی از او هستند که با دانستن آن ها می توانیم جواب برگشت/عدم برگشت مشتری به فروشگاه در آینده را بدهیم ولی هر کدام از feature ها با یک ضریب مشخص تاثیر گذارند و نمی توانیم ادعا کنیم که همگی به یک میزان تاثیر مثبت در پیش بینی دارند؛ به عنوان مثال تاثیر Total Quantity با تقریب خوبی از پارامتر Country بهتر است.

برای به دست آوردن این تاثیر (ضریب تاثیر گذاری) هر feature از information gain استفاده می کنیم.

gain یک پارامتر بر پارامتر دیگر، میزان تاثیر گذاری آن در تشخیص نوع (مقدار) پارامتر دوم را بیان می کند.

بدیهی است که هر مشتری customerID خاص خود را دارد و با دانستن customerID یک مشتری می توانیم کل اطلاعات او را به دست آورده و Is Back او را تشخیص دهیم (هر ردیف در dataset یک customerID یکتا دارد). پس gain شناسه مشتری بر داده هدف برگشت/عدم برگشت بالا می باشد ولی عکس آن اصلا درست نیست. یعنی با دانستن برگشت/عدم برگشت یک مشتری شناسه او قابل پیش بینی نیست. پس پارامتر customerID را مجموعه feature ها حذف می کنیم.

اما در مورد بقیه ستون های غیر target که آن ها را feature نام بردیم، این موضوع کاملا صادق نیست. یعنی مثلا از yes بودن پارامتر برگشت/عدم برگشت یک مشتری می توانیم حدس بزنیم که احتمالا count یا quantity بالایی در خرید قبلی داشته.

برای محاسبه information gain همه پارامتر ها از متد mutual_info_classif کتابخانه sklearn استفاده کردیم.

[0.11085386 0.11149918 0.00524362 0.02839502 0.06910112 0.01070025

0.02192575]

از اعداد می توان اینطور به نظر می رسد که information gain مربوط به total quantity و total price از ما بقی بیشتر است و تاثیر بیشتری در انتخاب جواب درست target می گذارند که قابل حدس است.

در مورد country تاثیر کمی را می بینیم و در تاریخ هم month اهمیت بیشتری دارد (به خاطر مراسم های خاصی که ممکن است در یک ماه خاص اتفاق افتد، نزدیک آخر سال بودن ماه و مناسب بودن زمان خرید ...)

فاز 1

در ابتدا با train_test_split از کتابخانه sklearn داده ها را دو دسته می کنیم (15 درصد را به test و 85 درصد به train اختصاص می دهیم). حال داده ها را با 3 classifier مختلف مدل کرده و تست می کنیم و معیار های دقت را نمودار می کنیم.

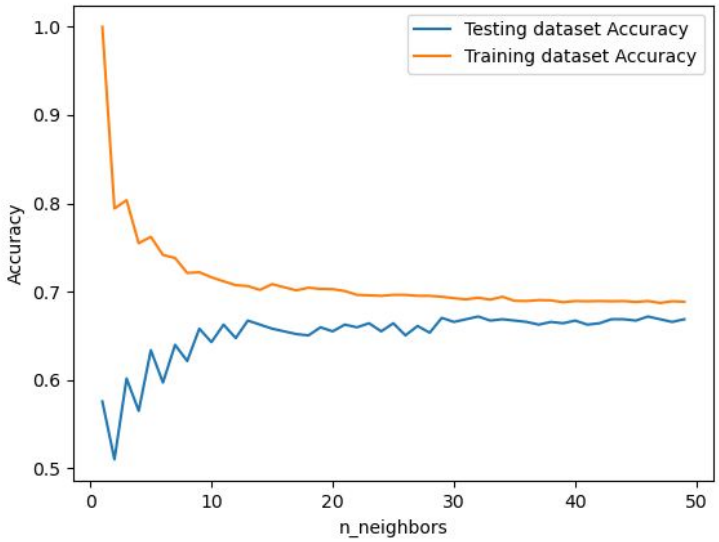
:KNN

در knn، به آموزش دادن و مدل کردن بر اساس داده هایی که از آموزش به دست می آیند نیاز نداریم، بلکه به ازای هر داده جدید، k نقطه نزدیک به او در داده های آموزش را پیدا می کنیم و بر اساس دسته های اون k داده vote گرفته و نتیجه را به نقطه جدید نسبت می دهیم.

برای مدل کردن knn از KNeighborsClassifier کتابخانه sklearn استفاده می کنیم و برای به دست آوردن k مناسب جهت ساختن classifier، از 1 تا 100 حرکت کرده و به ازای هر شماره ی بازه، مدل را ساخته و داده های train را روی آن fit می کنیم (x_train که feature های داده های آموزش و y_train که target های متناظر می باشد). آنگاه به ازای مدل هر شماره، داده های تست را روی آن predict می کنیم و accuracy پیشبینی را نگه میداریم.

در پایان از میان accuracy ها، ماکسیمم را انتخاب کرده و شماره مرتبط (از 1 تا 100) به آن را به عنوان k مطلوب برمیگردانیم.

پس از به دست آوردن k مناسب، مدل را بر حسب x_train و y_train ساخته و fit کرده و داده های تست را روی مدل تست میکنیم.



معیار های دقت داده های تست:

Recall : 0.977116704805492

Precision : 0.6756329113924051

Accuracy : 0.6717557251908397

معیار های دقت داده های آموزش:

Recall : 0.9764613573950569

Precision : 0.6975896860986547

Accuracy : 0.6930746429533818

در حالت بالا، پس از iterate از 1 تا 100، k مطلوب برابر با 32 شد که در شکل هم قابل حدس است.

در رابطه با شکل توجیه ما بدین صورت است:

در ابتدای نمودار که k عدد کوچکی می باشد (مثلا 1)، اگر یک داده ی train وارد کنیم تا target ان را پیش بینی کنیم، از آنجا که با تنها نزدیکترین داده train نزدیک به خود (خودش) مقایسه می شود، قطعا جواب درست می گیرد و به target درست می رسد. پس accuracy برابر با 100 درصد داریم.

اما از طرف دیگر داده تست، در k = 1، تنها با یک داده train که نزدیک خودش می باشد مقایسه می شود و چون این داده با داده های train غریبه است، حالت قبل به وجود نمی آید و اصلا حتی نمی توان تقریب خوبی زد که همان 1 داده که نزدیک ترین به او می باشد target یکسانی با او داشته باشد (مثلا ممکن است که اگر 3 تا نزدیک ترین را بررسی کنیم به جواب دیگری برسیم).

اما همانطور که k بیشتر می شود، دیگر داده train تنها با خود یا داده های کم اطراف خود مقایسه نمی شود، بلکه داده های دورتر هم در vote دخیل می شوند و ممکن است target غلط نماینده بیشتری داشته باشد و پیشبینی غلطی بکنیم. اما داده test از این افزایش k سود می برد چرا که با داده های train بیشتری سروکار دارد و اگر دو سه داده نزدیک آن target مخالف او داشته باشند فقط به آنها اکتفا نمی کند، بلکه به داده های دورتر هم سر می زند که ممکن است target یکسان با او داشته باشند.

پس دو نمودار test و train با افزایش k به هم نزدیک و نزدیکتر می شوند و متوسط به 70 درصد accuracy می رسند و همانطور که توضیح داده شد و از شکل پیداست بین accuracy های دو نمودار فاصله زیادی نمی افتد و به هم واگرا می شوند (مدل در k های بزرگتر از 15 به طور یکسان با دیتا های train و test برخورد می کند) پس overfit نداریم.

Decision tree

در ساختن و نمونه گیری از این classifier باز هم از کتابخانه sklearn کمک گرفتیم.

در این classifier همانطور که از اسمش پیداست، یک درخت متشکل از node های مختلف داریم. هر node بیانگر feature ای است که داده های رسیده به آن node بر اساس آن feature جدا شده اند.

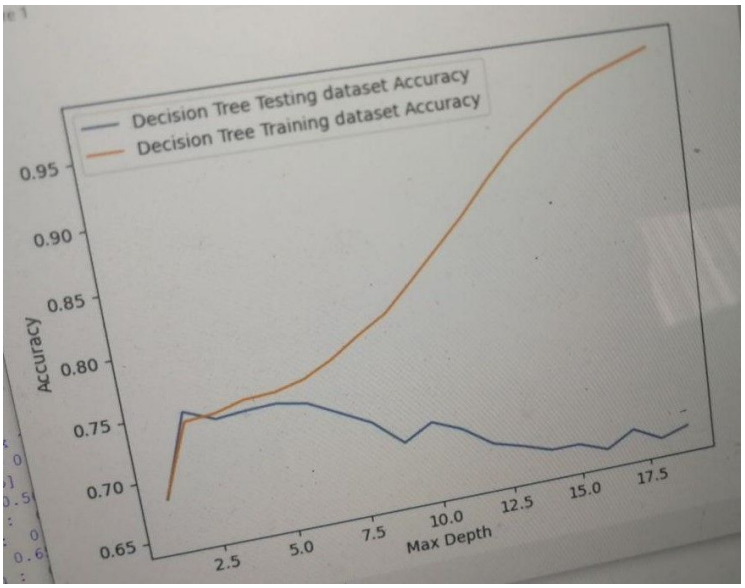
پس این مدل hyperparameter های مختلف مربوط به درخت مانند max_leaf_nodes ، max_depth و آلفا و ... را شامل می شود. در اینجا برای رفع overfit از دو hyperparameter به نام های max_depth و ccp_alpha استفاده کردیم. روش انتخاب آلفا و max_depth مناسب مطابق با توضیحاتی است که در knn اشاره کردیم (در بازه شماره های 1 تا 100 حرکت کرده و درخت با بهترین max_depth را پیدا می کنیم و در آن درخت با cost_complexity_pruning_path آرایه ای از آلفا ها به دست می آوریم که مبنای کار pruning در درخت می باشند.

برای هرس کردن درخت و بریدن نود های اضافی که کمکی به تشخیص درست target نمی کنند می توانیم از یک ضریب به نام آلفا استفاده کنیم و به ازای هر نود درخت pChance ای داشته باشیم که اگر بزرگتر از این مقدار آلفا بود هرس شده و از بین برود. البته جزییات این کار ها در توابع کتابخانه به کار رفته پیاده سازی شده اند و ما صرفا شمایی از نحوه هرس شدن درخت بیان کردیم. حال همانطور که گفته شده ابتدا آرایه ای از آلفا های مطلوب که در درخت با عمق k ما هرس به اندازه مشخصی را انجام می دهند به دست

می آوریم (آخرین آلفا در لیست، آن ضریبی است که کل درخت را هرس می کند و تنها 1 نود باقی می گذارد). حال از بین آلفا های به دست آمده بهترین آلفا را بر حسب بهترین accuracy به دست آمده انتخاب کرده و درخت را بر حسب آن ساخته و روی x_train و y_train مدل می کنیم.)

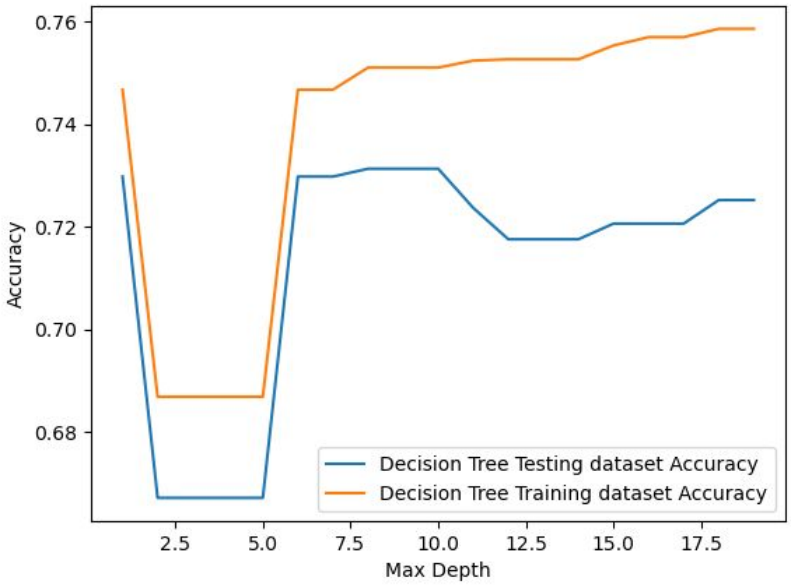
مشاهده می شود که عمل pruning دقت مدل را بالاتر برده و overfit را تا حد خوبی از بین می برد.

نمودار دقت داده تست و داده آموزش بر حسب accuracy بدون pruning:



در شکل بالا overfit زیادی که در عمق های بالاتر به وجود آمده را می بینیم به گونه ای که دقت داده آموزش به 1 میل می کند ولی دقت داده تست کم و کم تر می شود.

نمودار دقت داده تست و داده آموزش بر حسب accuracy بعد از pruning:



معیار های دقت داده های تست:

Recall : 0.9016018306636155

Precision : 0.7323420074349443

Accuracy : 0.7145038167938931

معیار های دقت داده های آموزش:

Recall : 0.9544919576304434

Precision : 0.76920644957319

Accuracy : 0.7720291026677445

مشاهده می شود که فاصله دقت دو نمودار دیگر از هم دور و دورتر نشده و ابتدا در عمق های کمتر، هرس باعث می شود که در داده آموزش سراغ نود های بد نرویم و دقت بهتر شود (چون قبلا درخت را بر اساس همین داده ها ساخته بودیم احتمالا در عمق کم اگر هرس کنیم همان چند تا نودی که غلط انداز بودند را بریده و دقتمان بالا می رود اما با افزایش عمق ممکن است بعضی شاخه های مفید تر هم هرس شوند و دقت کم شده و پس از آن از آنجا که تعداد feature ها کم و محدود است ابتدا وقتی max_depth را زیاد کنیم دقت بالاتر می رود و سپس هر چه ماکسیمم عمق بیشتر شود با توجه به محدود بودن تعداد feature ها با هرس کردن هم به یک میزان از دقت می رسیم)، حال آنکه در داده تست در عمق کم با هرس شانس دیدن همان چند نود تا آن عمق را هم میگیریم ولی با افزایش ماکسیمم عمق دقتش بالاتر می رود و همانطور که گفته شد بعد از مدتی به حالت ثابت می رسد (محدود بودن تعداد feature ها و برش ها در هرس و ....).

Logistic regression

درباره logistic regression می توان این را گفت که به ازای هر داده ورودی، بردار feature های آن را در بردار ضرایب آنها که طی آموزش به دست آورده ایم ضرب می کنیم و نتیجه را به تابع sigmoid می دهیم تا شانس وجود داده در یکی از دسته ها را تعیین کند. در واقع الگوریتم برای هر داده احتمال حضور او را در هر یک از دسته ها بیان می کند که اینجا چون target به صورت 0 و 1 است، دسته با شانس بالاتر برنده اعلام می شود.

از کتابخانه sklearn برای مدل سازی با logistic regression بهره بردیم و نتایج بدین شرح است:

معیار های دقت داده های تست:

Recall : 0.8466819221967964

Precision : 0.7597535934291582

Accuracy : 0.7190839694656489

معیار های دقت داده های آموزش:

Recall : 0.8787759905845429

Precision : 0.7788595271210014

Accuracy : 0.7453516572352465

در ادامه روش های آموزش گروهی را روی داده ها مشاهده می کنیم.

فاز 2

1)

بررسی bagging:

در این روش از داده های آموزش به تعدادی سمپل میگیریم تا آن ها را یکی یکی به یک classifier می دهیم تا آن را مدل کند.

برای یک داده جدید، او را به تک تک classifier ها می دهیم تا پیشبینی خود را انجام دهند و بعد از میان آن ها vote کرده و بیشتری را به عنوان کلاس این داده جدید در نظر میگیریم. بگینگ برای کم کردن overfit و از بین بردن واریانس زیاد به کار می رود.

بعد از تست کردن bagging روی مدل های knn و decision tree، اورفیت اندکی کم می شود و فاصله دقت train و test اندکی کم می شود، پس به عنوان مثال در مورد knn داریم:

: Test Results

KNN Score: 0.6717557251908397

Bagging Score: 0.667175572519084

: Train Results

KNN Score: 0.6930746429533818

Bagging Score: 0.6879547291835085

همانطور که مشاهده می شود، در مورد train دقت اندکی کمتر شده. البته توجه شود که در این پروژه تحلیلی برای موفقیت آمیز بودن یادگیری های گروهی ملزومات و نیازمندی های بیشتری لازم است لذا تاثیر رفع overfit و افزایش دقت را زیاد نمیبینیم.

برای bagging،

Hyperparameter های max_samples و max_features را 0.5 قرار دادیم و تعداد estimator ها را نیز 10 گرفتیم.

:Random forest

در random forest، یک نمونه از train data گرفته و سپس تعدادی درخت نمونه برداری می کنیم.

در هنگام ساخت هر درخت، تا زمانی که به کوچکترین نود نرسیده ایم، تعدادی از variable ها را برداشته و بهترین را از میان آن ها انتخاب می کنیم و نود را بر اساس آن میشکنیم. این عمل را به ازای تک تک نود ها تا کوچکترین نود از نظر سایز و برای هر درخت انجام می دهیم.

در نهایت مدلینگ را روی درخت های ساخته شده انجام می دهیم و نتیجه پیش بینی هر یک را نگه داشته و target ماکسیمم را انتخاب می کنیم.

Random forest را از کتابخانه sklearn می گیریم و با hyperparameter های مختلف امتحان می کنیم و نمودار های accuracy بر حسب hyperparameter را می کشیم

Test Results :

Tree Score: 0.667175572519084

Forest Score: 0.7221374045801526

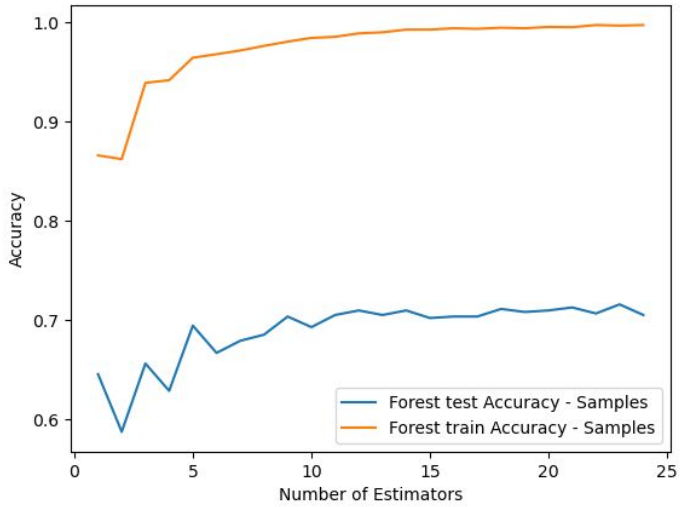
Train Results :

Tree Score: 0.6868768526003772

Forest Score: 0.7569388305039073

دیده می شود که accuracy پیشبینی در حالتی که از random forest استفاده کرده ایم نسبت به حالت محاسبه با یک decision tree بیشتر شده است و overfit نیز مشاهده نمی شود.

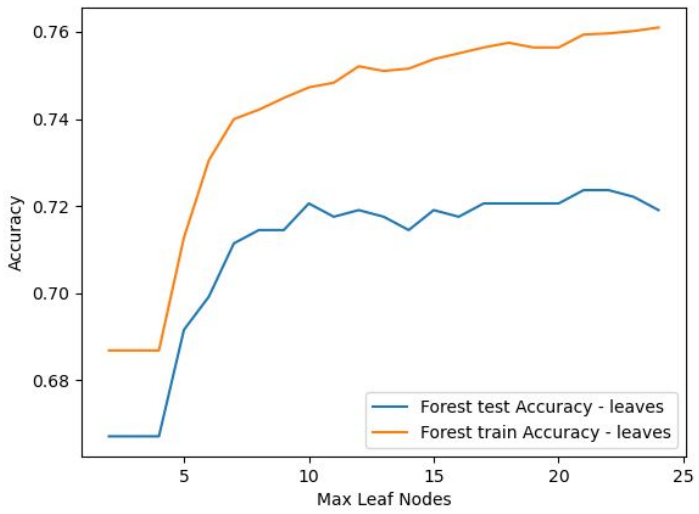
Random forest بر حسب number of estimators:



در بالا مشاهده می شود که چه در داده تست و چه در داده یادگیری، وقتی مدل کردن را از تعداد estimator

1 تا 15 اضافه می کنیم تعداد subtree های random forest بیشتر شده و یادگیری بهتر می شود (دقت بالاتر می رود). اما با افزایش بیشتر این تعداد estimator به نظر می آید که تعداد subtree های بیشتر دقت ما را خیلی بالاتر نمی برند و نتیجه vote از 20 درخت با نتیجه vote از 25 درخت تقریبا یکسان است و 5 درخت اضافه شده تاثیری بر تغییر target بیشترین ندارد.

Random forest بر حسب max leaf node:



در این نمودار تاثیر پارامتر max_leaf_nodes در مدل کردن randomforest را نشان داده ایم.

همانطور که در بالا مشاهده می شود، وقتی max leaf nodes را زیر 5 تا در نظر میگیریم، به درخت در واقع اجازه زیاد شکستن و تولید نود های بچه مناسب در سطح برگ را نمی دهیم و در سطح برگ که جواب های نهایی را داریم با محدودیت مواجه می شویم (شاید یکی از نود های پدر که اجازه شکستن به او را نداده ایم با شکستن خود information gain زیادی به وجود آورد)

پس وقتی تعداد max leaf nodes را بیشتر میکنیم فرصت تولید بچه های مناسب در سطح برگ را می دهیم و چه در داده تست و چه در داده train با افزایش دقت همراهیم. اما از leaf node 20 به بعد هر چه هم ماکسیمم تعداد نود های سطح برگ را هم بیشتر کنیم، از آنجا که تعداد feature ها محدود است، باز هم می توانیم به تعداد برگ مورد نیاز خود برسیم و نیاز به افزایش حد ماکسیمم این تعداد نیست. پس میبینیم که شیب افزایش دقت رو به کاهش رفته تا جایی که به حالت ثابت نزدیک می شود.

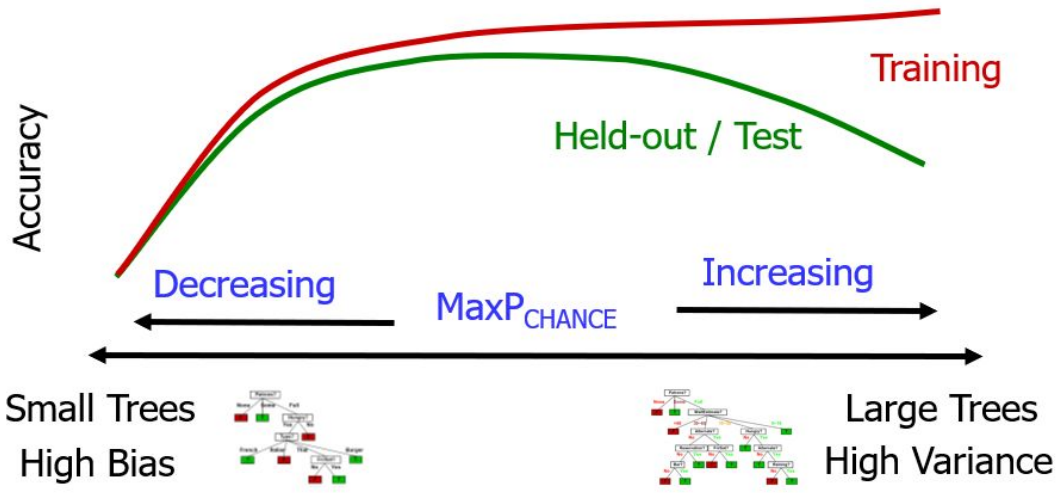
پارامتر های مورد بررسی n_estimators و max_leaf_nodes را از بازه 1 تا 25 برای هر کدام به RandomizedSearchCV کتابخانه sklearn دادیم تا بهترین عدد از هر range را انتخاب کند بگونه ای که بهترین دقت با کمترین overfit برای مدل randomForest به دست آید.

دیگر hyper parameter های random forest در کتابخانه sklearn عبارتند از:

- max_depth
- min_sample_split
- max_leaf_nodes
- Min_samples_leaf
- n_estimators
- max_sample (bootstrap sample)
- max_features

Bootstrapping چیست و چه تاثیری بر واریانس و بایاس دارد؟

Bootstraping از روش های افزایش دقت مدل و کاهش خطا ها و هم چنین کاهش واریانس و بایاس مدل می باشد.



در بالا شمایی از واریانس و بایاس بالا را می بینیم.

بایاس بالا زمانی مطرح می شود که پیش بینی accuracy کمی دارد و دقت پیش بینی پایین بوده است. به بیان دیگر چه در پیش بینی دسته آموزش و چه دسته تست، مدل ضعیف کار می کند و تعداد کمی از داده ها را به درستی تشخیص می دهیم.

اما واریانس بالا زمانی اتفاق می افتد که مدل روی داده های train که با آنها fit شده خوب جواب می دهد اما در هنگام ارزیابی با داده های تست accuracy پایین می گیریم یعنی مدل ما خیلی به سمت داده های آموزش سوگیری کرده.

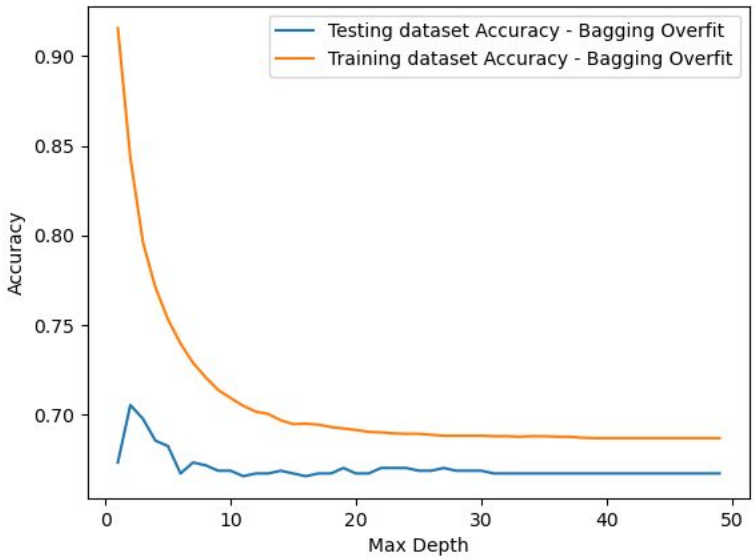
برای حل این مشکل روش های مختلف وجود دارد. یکی از آن ها bootstrapping می باشد که مبنای کار آن بر اساس رندم sample کردن data train با جایگزینی است.

در این روش تعدادی نمونه از داده های train با همان سایز می گیریم و هر کدام از داده های جدید نمونه برداری شده از داده اولیه را به یک classifier می دهیم تا مستقل از یکدیگر پیشبینی را انجام دهند. از آنجا که این روش از چندین classifier که موازی با هم و مستقلا عمل می کنند بهره می برد، خطا را به ویژه در مورد واریانس کاهش می دهد. چرا که اگر تنها یک classifier داشتیم و او به غلط یک نتیجه غلط را حدس میزد، ما هم نتیجه غلط را از کل الگوریتم می گرفتیم اما با این روش به عنوان مثال اگر 5 classifier داشته باشیم که هر کدام داده های train جدای از یکدیگر را دارند داشته باشیم، حتی اگر دو تا از classifier ها نتیجه غلط را پیشبینی کنند ما جواب درست را از الگوریتم میگیریم و طبیعتا دقت الگوریتم بالا می رود و داده های تست با سخت گیری بیشتری و از طرف چند نماینده classifier قضاوت می شوند. پس احتمالا به دقت داده های train نزدیک می شویم و واریانس کاهش می یابد اما بایاس بیشتر با کامل بودن و تمیز بودن data train ورودی سروکار دارد بگونه ای که classifier را از انواع مختلف حالت های رخ دادن یک target خاص تغذیه کند و داده های غلط و به درد نخور نداشته باشیم، هر چند bootstrapping هم در کاهش خطای بایاس بی تاثیر نیست.

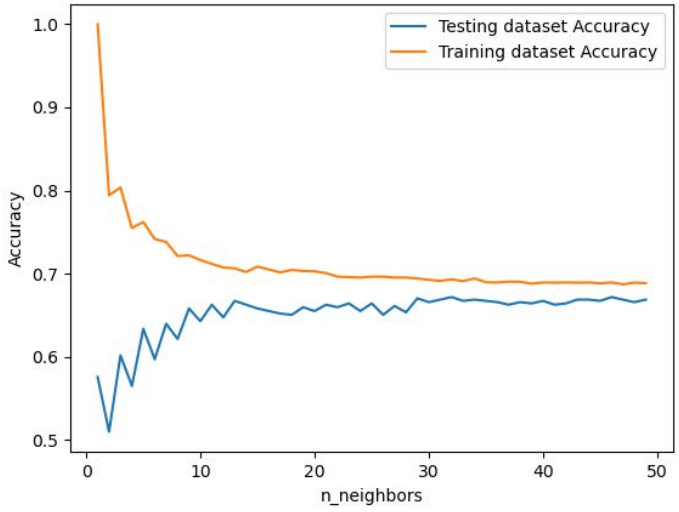
مشاهده تاثیر bagging بر overfitting

Bagging از روش های bootstraping می باشد و همانطور که در قسمت قبل توضیح دادیم برای کاهش overfit سودمند است.

در اینجا برای نشان دادن تاثیر bagging، مدل knn را به عنوان base_estimator به baddingClassifier داده و نمودار knn که در قسمت های قبل در بازه 1 تا 50 پارامتر k کشیده بودیم را با نمودار حاصل از bagging در بازه 1 تا 50 مقایسه می کنیم (به ازای همه شماره های بین 1 تا 50، knn را تشکیل می دهیم و به baggingClassifier به عنوان base_estimator می دهیم و نتیجه داده train و test را روی آن predict کرده و مطابق قبل در یک نمودار نشان می دهیم.



KNN WITH BAGGING



KNN WITHOUT BAGGING

در نمودار دوم مشاهده می کنیم که در k های کوچک زیر 10، تفاوت دقت train و test زیاد (در حد 40 درصد) بوده که کم کم و بعد از k برابر با 15 شروع به نزدیک شدن به هم می کنند (نمودار تست از درصد 50 به حدود 70 می رسد و نمودار train از 1 به 70)

حال آنکه در نمودار اول که با bagging یادگیری جمعی شده، به ازای k کوچک دقت نمودار test از مقداری نزدیک تر به 70 و همینطور نزدیک تر به نمودار train با دقت 90 شروع شده و کم کم دو نمودار به هم نزدیک تر شده اند. یعنی دور بودن بیش از حد دو نمودار از k برابر با 1 تا 15 در نمودار اول که با bagging یادگیری جمعی شده تا حد خوبی کمتر شده و دقت های دو نمودار نزدیک ترند.

البته تاثیر مشاهده شده خیلی زیاد نمی باشد.

Hard voting

Recall : 0.9954233409610984

Precision : 0.6744186046511628

Accuracy : 0.6763358778625954