۱)

در این پروژه ابتدا با کمک کتابخانه sklearn یک decision tree میسازیم که با ۸۰ درصد دیتا داده شده train شده است. این دیتا دارای ۳۰۳ sample و ۱۲ feature است. سپس accuracy این درخت تصمیم را با تست روی ۲۰ درصد باقی مانده دیتا به وسیله خود تابع accuracy_score موجود در کتابخانه sklearn بدست می آوریم.

bootstrapping چیست و چه تاثیری بر روی واریانس و انحراف معیار استاندارد دارد؟

bootstrapping یک روش نمونه گیری است که در آن داده ها به صورت تصادفی و با امکان تکرار انتخاب می شوند. اگر train data ای که داریم را bootstrapped کنیم میزان واریانس و انحراف معیار کمتر خواهد شد.

Accuracy این درخت تصمیم با سه بار تست به صورت زیر است.

تست اول: ۵۳ درصد

تست دوم: ۵۶ درصد

تست سوم: ۵۸ درصد

میانگین: ۵۵.۶ درصد

۲.۱ ـ ۲.۲)

Bagging یعنی bootstrap + aggregating که یعنی استفاده از یک داده bootstrapped (که در بالا توضیح داده شد) و پیش بینی های مدل aggregated باشد که یعنی برای این منظور میتوان از هر مدلی استفاده کرد که یکی از این مدل ها میتواند درخت تصمیم باشد.

Overfitting چیست و چرا درخت تصمیم به آن حساس است؟ bagging سعی دارد چه مساله ای را حل کند؟

یک ماشین ممکن است در بعضی موارد آنقدر train data را خوب یاد بگیرد که فقط بتواند همان دیتا را درست پاسخ دهد و اگر موردی خارج از train data را بخواهد پیش بینی کند نتواند با دقت خوبی پیش بینی کند. در یک درخت تصمیم overfitting زمانی اتفاق می افتد که درخت تصمیم به گونه ای طراحی شده است که به طور کامل بر همه ی سمپل های داده تست fit شود. مشکلی که درخت های تصمیم دارند این است که با توجه به train data ای که به آنها داده میشود خروجی آنها میتواند بسیار متفاوت باشد (به همان دلیل overfitting) به همین واریانس بسیار

زیاد میشود. یکی از روش های از بین بردن این مشکل bagging است. در این روش به صورت تصادفی و با جایگزینی, از train data ای که داریم چند دسته سمپل N تایی (به گونه ای که معمولا N به اندازه ۶۰ درصد اندازه train data اصلی است.) از train data درست میکنیم (ایجاد bootstrapped data). و سپس برای هر دسته از سمپل ها یک درخت تصمیم جداگانه train میکنیم. سپس برای داده های تست, تست ها را به همه ی این درخت های تصمیم میدهیم و سپس هر جوابی که درخت های تصمیم بیشتری به آن رسیده بودند میشود جواب پیش بینی کلی مساله. با این روش یک train data ای که داشتم را به چند train data تبدیل کردیم و هر کدام از آنها را جداگانه train کردیم که با این کار جواب کلی دیگر بر یک نمونه train data بیش از حد fit نیست و برای همین با روش bagging میتوان مشکل overfitting را حل کرد.

برای پیاده سازی bagging در پروژه دیتا داده شده را به ۵ دسته ۱۵۰تایی تقسیم می کنیم و سپس برای هر دسته یک درخت تصمیم را train می کنیم. سپس هر داده تست را به این ۵ درخت میدهیم تا در مورد آن تصمیم بگیرند و هر تصمیمی که بیشتر گرفته شود میشود تصمیم نهایی.

دقت مدل در سه بار تست به صورت زیر است.
تست اول: ۵۶ درصد
تست دوم: ۶۰ درصد
تست سوم: ۶۰ درصد
میانگین: ۵۸ درصد

۲.۳)

در این بخش با شروع از اولین feature هر بار یکی از ویژگی ها را حذف میکنیم و درخت تصمیم برای دیتا باقی مانده را میسازیم.
با سه بار اجرای برنامه accuracy ها به شکل زیر بودند.

تست اول	تست دوم	تست سوم
0.6666666666666666	0.5833333333333334	0.6333333333333333
0.5333333333333333	0.48333333333333334	0.48333333333333334
0.5333333333333333	0.5666666666666667	0.5666666666666667
0.5666666666666667	0.55	0.5166666666666667
0.6166666666666667	0.6166666666666667	0.65
0.5666666666666667	0.5333333333333333	0.55
0.5833333333333334	0.5166666666666667	0.5333333333333333
0.5666666666666667	0.5666666666666667	0.55
0.5	0.5	0.5666666666666667
0.55	0.5166666666666667	0.5833333333333334
0.5333333333333333	0.5	0.6
0.5333333333333333	0.55	0.5
0.5833333333333334	0.5833333333333334	0.5833333333333334

که در بعضی موارد accuracy هر سه بار بیشتر شد. به نظر من با حذف ca کمترین میزان افت به وجود آمد.

۲.۴)

در این بخش پنج ویژگی را به طور تصادفی از ۱۲ ویژگی موجود انتخاب کرده و درخت تصمیم آن را میسازیم. Accuracy درخت ایجاد شده به صورت زیر است.

تست اول: ۶۵ درصد

تست دوم: ۵۵ درصد

تست سوم: ۴۱.۶ درصد

۲.۵)

برای ساخت random forest باید تعداد بیشتری از درخت هایی که در بخش ۲.۴ ایجاد کردیم, درست کنیم و سپس داده تست را به همه آنها بدهیم و هر نتیجه ای که بیشتر انتخاب شد میشود نتیجه نهایی. یعنی برای ساخت جنگل تصادفی باید تعداد زیادی درخت داشته باشیم که در هر کدام به صورت تصادفی بعضی ویژگی ها حذف شده است و با دیتا های باقی مانده آموزش دیده شده اند.

در این بخش یک جنگل با صد درخت ساختم که هر کدام از درخت ها با پنج feature که به صورت تصادفی از بین تمام feature ها انتخاب شده بودند train دیدند که با سه بار اجرا کردن برنامه accuracy ها به صورت زیر بودند.

تست اول: ۹۱.۶ درصد

تست دوم: ۹۵ درصد

تست سوم: ۱۰۰ درصد

میانگین: ۹۵.۵۳ درصد

و زمانی که به همین شکل یک جنگل با ۵۰۰ درخت ساختم در هر سه بار تست accuracy ۱۰۰ درصد شد.

ارتباط random forest و bagging چیست؟ random forest سعی دارد چه مساله ای را حل کند؟

همانطور که در بالاتر توضیح داده شده بود در boosting از روش های متفاوتی برای aggregation میتوان استفاده کرد و random forest یک الگوریتم bagging است ولی تفاوت آن با bagging tree این است که در bagging همه ی tree ها در نظر گرفته میشوند ولی در random forest بعضی از آنها به صورت رندم انتخاب میشوند و هر درخت تصمیم با توجه به ویژگی های رندم انتخاب شده برای همان درخت train می شود. از random forest هم برای classification و هم برای regression استفاده می شود و هم چنین accuracy بیشتری از یک درخت تصمیم خواهد داشت.

با مقایسه دقت های ثبت شده در قسمت های ۱ و ۲.۲ و ۲.۵ و با توجه به سوالات بالا چه نتیجه ای میگیرید؟

میزان accuracy یک درخت تصمیم ۵۵.۶ درصد شد و با استفاده از bagging توانستیم accuracy را به ۵۸ درصد برسانیم و با استفاده از random forest اگر از تعداد زیادی درخت استفاده میکردیم میتوانستیم به دقت بسیار بالایی برسیم مثلا با ۵۰۰ درخت تقریبا به accuracy ۱۰۰ درصد رسیدیم. (در روش bagging هم اگر بیشتر از ۵ دسته درست میکردیم به دقت بیشتری میرسیدیم ولی با توجه به صورت پروژه از ۵ دسته استفاده شد) همان طور که در قبل هم گفته شد با استفاده از bagging توانستیم از overfitting جلوگیری کنیم و با random forest هم توانستیم accuracy را بالاتر ببریم.