قسمت الف) برای استفاده از قاعده‌ی بیز ساده، بر روی داده‌های iris باید احتمال زیر را بدست آوریم:

$$p(Species|SepalLength,SepalWidth,PetalLengths,PetalWidth) = p(SepalLength|Species)\ p(SepalWidth|Species)\ p(PetalLengths|Species)\ p(PetalWidth|Species)\ p(Species)\ /\ p(SepalLength,SepalWidth,PetalLengths,PetalWidth) \quad (1)$$

در این فرمول احتمال وقوع کلاس (رخداد) با داده‌شدن ۴ ویژگی محاسبه می‌شود؛ در واقع باید کلاسی انتخاب شود که این احتمال را بیشینه کند. بدلیل اینکه برای تمامی داده‌ها، احتمال مرزی (مخرج کسر، مربوط به ویژگی‌ها) یکسان است، می‌توان از محاسبه‌ی آن‌ها صرف‌نظر کرد. برای اینکه این فرمول را به جمعی از احتمال‌ها تبدیل کنیم، آن را از یک تابع خطی اکیداً صعودی (مثل لگاریتم) استفاده می‌کنیم. به دلیل اینکه داده‌ها از توزیع گاوسی تبعیت می‌کنند، برای محاسبه هرکدام از احتمالات، لازم است که واریانس و میانگین مربوط به آن توزیع گاوسی را داشته باشیم. برای مثال داریم:

$$p(SepalLength|Species) = 1/\sqrt{2\pi\sigma^2} \times \exp(-(SepalLength-\mu_{Species})\ /\ 2\sigma^2) \quad (2)$$

در تابع fit، احتمال وقوع هرکدام از سه کلاس را به همراه میانگین و واریانس ستون مربوطه‌اش بدست می‌آوریم. در تابع predict برای هر نمونه داده از دیتاست با توجه به مقدار ۴ ویژگی‌اش، یک کلاس اختصاص داده می‌شود (کلاسی که احتمال را بیشینه کرده است)؛ حال اختصاص کلاس با تابع predict_ صورت می‌پذیرد. برای اینکار، تمامی احتمال‌های موخر (به تعداد کلاس‌ها) را محاسبه می‌کنیم و بیشترین را در نظر می‌گیریم. محاسبه این posteriorها با توجه به نوع کلاس صورت می‌گیرد، به این صورت که لگاریتم احتمال وقوع کلاس را با تک‌تک لگاریتم‌های احتمال وقوع ویژگی به شرط کلاس را با هم جمع می‌کنیم. برای محاسبه احتمال وقوع کلاس از تابع pdf_ (مخففِ probability density function) استفاده شده که از میانگین و واریانس هرکدام از کلاس‌ها را استخراج کرده و فاصله‌ی مقدار ویژگی را با میانگین می‌سنجد (از فرمول شماره 2، که همان توزیع گاوسی است، استفاده می‌کند). برای محاسبه دقت (تابع accuracy)، تعداد داده‌هایی که کلاس‌هایی آن ها به درستی حدس زده شده است را به تعداد تمامی داده‌ها تقسیم می‌کنیم. برای نمایش بهتر شیوه‌ی عملکرد این روش، از ماتریس درهم‌ریختگی استفاده می‌کنیم. با تقسیم کردن مجموعه داده به تست و آموزش به دقت‌های متفاوتی می‌رسیم؛ بهترین مقدار برای تقسیم ۰/۱ است، که ۱۰ درصد از داده‌ها برای تست و ۹۰ درصد برای آموزش در نظر گرفته شده است و به دقت ۱۰۰٪ را برروی دادگان تست می‌دهد.

از **مزایای** این کلاس‌بند می‌توان به موارد زیر اشاره کرد:

۱. دسته‌بندی داده‌های آزمایشی آسان و سریع است. همچنین زمانی که تعداد دسته‌ها از دو بیشتر باشد نیز عملکرد خوبی از خودش نشان می‌دهد.

۲. زمانیکه شرط مستقل بودن برقرار باشد، یک کلاس‌بند بیز ساده عملکرد بهتری نسبت به مدل‌های دیگر مانند رگرسیون خطی دارد و به حجم آموزش کمی نیاز دارد.
۳. در حالتیکه ورودی‌هایمان دسته‌بندی شده باشند این روش عملکرد بهتری نسبت به حالتی دارد که ورودی‌هایمان عدد باشند. برای حالتی که ورودی عدد باشد به طور معمول اینطور فرض می‌شود که از توزیع نرمال پیروی می‌کنند که این خود فرض قوی‌ای به حساب می‌آید.
برای **معایب** نیز موارد زیر مطرح است:
۱. در صورتیکه ورودی‌مان دسته‌بندی شده باشد و در مرحله یادگیری دسته‌ای وجود داشته باشد که کلاس‌بند هیچ داده‌ای از آن دسته مشاهده نکرده باشد، کلاس‌بند احتمالی برابر صفر برای آن دسته در نظر می‌گیرد و قادر به دسته‌بندی کردن نخواهد بود (برای حل این مشکل می‌توان از تکنیک‌های هموارسازی مانند تخمین‌گر لاپلاس استفاده کرد).
۲. دستیابی به شرط مستقل بودن در دنیای واقعی تقریبا غیر ممکن است.
قسمت ب) در روش ترکیبی از چند کلاس‌بند استفاده می‌شود تا کلاس مربوطه مشخص گردد. در اینجا از درخت تصمیم به عنوان کلاس‌بند ترکیبی استفاده شده، که با استفاده از سه درخت می‌خواهیم عمل جداسازی را انجام دهیم. در کلاس RandomForest این درخت‌ها با تابع fit تشکیل می‌شوند. مراحل کار به این شکل است که ابتدا با تابع boostrap_sample یک زیرمجموعه تصادفی از داده‌ها (این زیرمجموعه هم اندازه مجموعه اصلیست، با این تفاوت که ممکن است بخشی از داده‌های آن تکرار شوند؛ با احتمال ۳۷٪ برخی از نمونه‌ها هرگز انتخاب نمی‌شوند) را برای آموزش دادن یکی از درخت‌ها، انتخاب می‌کنیم. سپس این مجموعه داده را برای ساخت درخت به تابع fit، در کلاس DesicionTree، می‌دهیم. تابع grow_tree در کلاس DesicionTree وظیفه ساخت درخت را بر عهده دارد. به هنگام ساختن درخت، در صورت وقوع سه شرط، شاخه‌زنی برای درخت متوقف می‌شود. اولین حالت هنگامی است که درخت به حداکثر عمق خود رسیده باشد (متغیر max_depth این آستانه را مشخص می‌کند)، در حالت دوم، اگر تنها یک کلاس برای دسته‌بندی باقی مانده باشد یا، در حالت سوم، که تعداد نمونه‌های باقی مانده برای دسته‌بندی کمتر از مقداری باشد که با متغیر min_samples_split تعیین کرده‌ایم. (متغیر min_samples_split برای جلوگیری از بیش‌برازش تعیین می‌شود. هر زمان که تعداد نمونه‌های باقی‌مانده برای دسته‌بندی کمتر از این مقدار شد، الگوریتم باید متوقف شود.) در این وضعیت برچسب کلاسی که بیشترین بار در بین داده‌های باقی مانده تکرار شده باشد به عنوان مقدار برگ درخت (کلاس نهایی) تعیین می‌گردد. (تابع most_common_label به ما برچسب کلاسی را برمی‌گرداند که در بین داده‌ها، بیشتر از همه تکرار شده باشد. از این کلاس، برای تعیین برگ درخت استفاده می‌کنیم.) برای اینکه تعیین کنیم که در هر نود در درخت کدام ویژگی قرار گیرد باید از بهره‌ی اطلاعاتی استفاده کنیم که با فرمول آنتروپی بدست می‌آید. در تابع best_criteria هربار یک ویژگی به صورت تصادفی، از میان ویژگی‌های باقی مانده، انتخاب و

بهره‌ی اطلاعاتی آن محاسبه می‌شود. در آخر این تابع به ما بهترین ویژگی را به همراه بهترین آستانه برای تشکیل نود برمی‌گرداند. best_criteria برای پیدا کردن بهترین آستانه به هنگام شاخه‌زنی، باید بهره‌ی اطلاعاتی را برای تک‌تک مقادیر منحصربه‌فرد آن ویژگی محاسبه کند و عددی که به ما بیشترین بهره را می‌دهد به عنوان آستانه برای آن ویژگی درنظر بگیرد. بهره اطلاعاتی با تابع information_gain محاسبه می‌شود؛ به این صورت که آنتروپی تمامی صفات (یا صفات باقی مانده در مراحل بعدی) به عنوان آنتروپی والد در نظر گرفته می‌شود، سپس تابع split ردیف‌هایی از ویژگی را که مقدارشان کمتر از حد آستانه است را در سمت چپ و آن‌هایی که مقدارشان از آستانه بیشترست را در سمت راست درخت قرار می‌دهد. (در اینجا مشخص می‌شود که درخت ما باینری است) با این تقسیم‌بندی مقدار آنتروپی فرزند با فرمول میانگین‌گیری وزن‌دار (قطعه کد مربوط به آن در زیر آورده شده است) محاسبه می‌شود و از آنتروپی والد کم شده تا بهره‌ی اطلاعاتی را به ما بدهد.

```
n = len(y)
n_l, n_r = len(left_idxs), len(right_idxs)
e_l, e_r = entropy(y[left_idxs]), entropy(y[right_idxs])
child_entropy = (n_l / n) * e_l + (n_r / n) * e_r
```

تابع entropy در ابتدا احتمال آماری را برای ورودی‌اش محاسبه می‌کند و سپس از فرمول زیر برای احتساب آنتروپی بهره می‌برد:

$$-\Sigma_i p_i \times \log_2(p_i) \qquad (3)$$

بعد این مراحل برای ویژگی‌های قرار گرفته در سمت چپ و راست، مجددا تابع grow_tree فراخوانی می‌شود تا اینکه هرکدام به شرط توقف برسند و برگ‌های درخت پیدا شود. (از تابع grow_tree به شیوه‌ی بازگشتی استفاده شده است) (درخت‌های تصمیم با الگوریتم ID3 تشکیل شده‌اند) به تعداد درخت‌های مشخص داده با متغیر n_trees ما n تا کلاس‌بندِ درخت تصمیم داریم.

برای سنجش دقت این کلاس‌بند ترکیبی از مجموعه داده تست استفاده می‌کنیم. با فراخوانی تابع predict از کلاس RandomForest، ابتدا کلاسی که هرکدام از درخت‌ها برای هر داده بدست آورده‌اند با پیمایش درخت بدست می‌آید. سپس کلاس یا برچسبی که توسط هرکدام از کلاس‌بندها (در اینجا درخت تصمیم) به تعداد بیشتری انتخاب شده باشد، به عنوان کلاس آن داده در نظر گرفته می‌شود. شیوه پیمایش درخت برای یافتن کلاس مربوط به داده‌ی مشاهده شده، به این صورت است که برای هر ورودی با ۴ ویژگی مشخص تابع traverse_tree به صورت بازگشتی روی هر ویژگی فراخوانی می‌شود تا به یک برگ، که همان کلاس ورودیست، برسد. سپس برای آن ورودی، کلاس بدست آمده را نسبت می‌دهد. تابع predict از کلاس DecisionTree، آرایه‌ای از کلاس‌های متناظر با ورودی‌ها را برمی‌گرداند. با داشتن کلاس هر داده‌یِ تست و پیش‌بینی کلاسِ مربوط به آن، از تابع accuracy، با همان تعریف آورده شده در قسمت الف، میزان دقت

این کلاس‌بند ترکیبی محاسبه می‌شود. برای مقایسه‌ی عملکرد از ماتریس درهم‌ریختگی استفاده شده است.

از **مزایای** این روش اینست با میانگین‌گیری یا ترکیب نتایج درختان تصمیم‌گیری مختلف، بر مشکل بیش‌برازش غلبه می‌کند. جنگل‌های تصادفی برای بازه وسیعی از عناصر داده، نسبت به درخت تصمیم‌گیری مجزا، عملکرد بهتری دارند. همچنین این الگوریتم نسبت به درخت تصمیم‌گیری مجزا واریانس کمتری دارد. جنگل‌های تصادفی بسیار منعطف هستند و دقت بسیار بالایی دارند. در الگوریتم جنگل تصادفی نیازی به مقیاس‌پذیری داده وجود ندارد، زیرا حتی بدون مقیاس‌بندی داده، دقت خوبی باقی خواهد ماند. حتی در صورت فقدان بخش بزرگی از داده، الگوریتم‌های جنگل تصادفی دقت بالایی خواهند داشت. این روش امکان مدیریت بسیار عالی داده‌های چند بعدی و دقت بالا حتی در داده‌های گم‌شده را محقق می‌کند. از **معایب** این الگوریتم می‌توان گفت که پیچیدگی، اصلی‌ترین عیب الگوریتم‌های جنگل تصادفی است، در حقیقت تعداد زیاد درخت‌ها می‌توانند الگوریتم را برای پیش‌بینی‌های جهان واقعی کند و غیر موثر کنند. ساخت جنگل‌های تصادفی بسیار سخت‌تر و زمان‌برتر از درختان تصمیم‌گیری است چون به جای یک درخت به چند درخت احتیاج داریم. برای پیاده‌سازی الگوریتم جنگل تصادفی، به منابع محاسباتی بیشتری نیاز است. در صورت وجود یک مجموعه بزرگ از درختان تصمیم‌گیری، جنگل تصادفی کمتر شهودی خواهد بود. روند پیش‌بینی با استفاده از جنگل‌های تصادفی در مقایسه با سایر الگوریتم‌ها بسیار زمان برتر است.

قسمت ج) در این الگوریتم فاز آموزش وجود ندارد و تنها با تعریف کردن مدل با مجموعه داده‌ها می‌توان از این شبکه استفاده کرد. برای پیش‌بینی کلاس داده جدید از فاصله‌ی اقلیدسی با فرمول زیر استفاده می‌شود:

$$d(p,q) = \sqrt{\Sigma_i (q_i - p_i)^2} \qquad (4)$$

با فراخوانی تابع predict_class برای هر ورودی دیده‌نشده (داده تست)، ابتدا فاصله‌ی اقلیدسی این نقطه با تک‌تک نقاط موجود در مجموعه آموزشی محاسبه می‌شود و برچسب (کلاس) k نقطه نزدیک به این نقطه برگردانده می‌شود، سپس برچسبی که بین این k نقطه بیشتر از بقیه تکرار شده باشد، به داده ورودی نسبت داده می‌شود. تابع get_k_neighbors وظیفه پیدا کردن k نقطه همسایه را بر عهده دارد.

از **مزایای** این الگوریتم می‌توان به موارد زیر اشاره کرد:

۱. سادگی الگوریتم
۲. تفسیر بسیار ساده
۳. دقت بالا
۴. چندمنظوره بودن
۵. قابلیت استفاده در طیف وسیعی از مسائل

۶. عدم نیاز به در اختیار داشتن فرضیات درباره‌ی داده، که مخصوصاً در خصوص داده‌های غیرخطی بسیار کاربردی است

معایب هم شامل موارد زیر می‌شوند:

۱. زمان حدس زدن طولانی

۲. محاسبات گران

۳. نیازمند حافظه زیاد چون باید تمامی داده‌های قبلی را ذخیره کند

۴. حساس به مقیاس داده و ویژگی‌های غیرمرتبط

۵. اگر k عدد بزرگی شود پیش‌بینی کند و زمان افزایش پیدا می‌کند

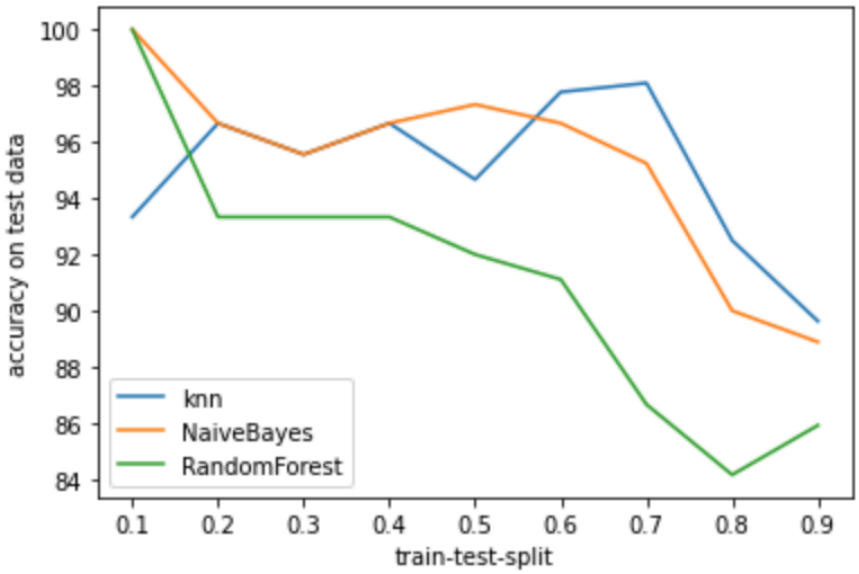
۶. از نظر همسایه‌ها به طور یکسان استفاده می‌کند، در حالی که باید هر همسایه بر اساس فاصله‌ای که به نمونه‌ی تست دارد، سهم متفاوتی در پروسه تصمیم‌گیری داشته باشد، همین موضوع باعث شده که الگوریتم knn به تعداد k بسیار حساس باشد

قسمت د) مزایا و معایب هر روش در قسمت مربوط به خودش توضیح داده شده است.

الگوریتم knn بدلیل فاز آموزشی ندارد، با اضافه شدن داده‌ی جدید به مشکلی بر نخواهد خورد و به اصطلاح قابلیت یادگیری آنلاین را دارد، اما در صورتیکه حجم داده ها بالا برود، حجم محاسبات برای پیش‌بینی نیز بسیار بالا رفته و مشکل‌ساز می‌شود. از طرفی یادگیری ترکیبی با جنگل تصادفی این قابلیت را ندارد و اگر حجم داده‌های آموزشی بالا رود، محاسبات برای آن به شدت پیچیده می‌شود اما به هنگام پیش‌بینی پیچیدگی زمانی پایین‌تری نسبت به دو الگوریتم دیگر دارد ($\log_2 n$) چون از درخت‌های تصمیم باینری استفاده می‌کند. الگوریتم بیز ساده نیز در مقابل knn بسیار سریع‌تر عمل می کند، چون knn دارای اجرای بلادرنگ (real-time execusion) می‌باشد. همچنین بیز ساده پارامتری است، اما knn و جنگل تصمیم پارامتری نمی‌باشند. هرس درخت‌های تصمیم در الگوریتم ترکیبی ممکن است برخی از مقادیر را در داده‌های آموزشی نادیده بگیرد، که منجر به افزایش دقت می‌شود. در روش knn هایپرپارامتر k و نوع فاصله‌سنج نیز باید تعیین شود (با هر k متفاوت به دقت‌های مختلفی می‌رسیم) که این خود نیازمند محاسبات بیشتری نسبت به دیگر الگوریتم‌هاست. اما این الگوریتم نسبت به داده‌های نویزی مقاوم‌ترست. جنگل از چند دسته از درختان تصمیم‌گیری ترکیبی ساخته شده که می‌توانند ویژگی‌های طبقه‌بندی را به خوبی استخراج کنند در صورتیکه برای طبقه‌بند بیز ابتداً باید ویژگی‌ها مشخص باشند. الگوریتم جنگل تصمیم می‌تواند، فضاهایی با ابعاد بالا و همچنین تعداد زیادی مثال آموزشی را مدیریت کند. بیز ساده، برخلاف knn، یک طبقه‌بند خطی است؛ به همین دلیل با اعمال برروی داده‌ها بزرگ سریع‌تر عمل می‌کند. در جایی که سرعت مهم است، بیز ساده به knn و جنگل تصمیم ارجح داده می‌شود. اما هنگامی که می‌خواهیم از بیش‌برازش جلوگیری کنیم، انتخاب ما جنگل تصمیم است. بیز ساده زمانیکه روی داده‌ها بزرگ اعمال می‌شود، بسیار دقیق عمل می‌کند. در صورتیکه که احتمال وقوع یکی از شرط‌ها در بیز ساده، صفر باشد، این الگوریتم بسیار بد عمل

خواهد کرد. بیز ساده در جایی درست کار می‌کند که مرز جداساز خطی، بیضوی یا سهمی‌گون باشد، در غیر این صورت دو الگوریتم دیگر انتخاب‌های بهتری هستند. knn و بیز ساده در جایی که اتفاقات نادر رخ می‌دهد بهتر از جنگل تصمیم عمل می‌کنند، چون با هرس کردن درخت‌های تصمیم صفت‌هایی که کم اتفاق می‌افتند را نادیده می‌گیرد. در جنگل تصمیم، درخت‌های تصمیم‌گیری برای توضیح و درک بهترین هستند. درخت‌های تصمیم دارای ویژگی‌های ساده برای شناسایی ویژگی مهم، کنترل کردن مقادیر از دست رفته و مقابله با موارد پرت هستند. جنگل‌های تصمیم عموما برای مواردی که تعداد کلاس‌ها پایین هستند، بیشتر استفاده می‌شود. برخلاف بیز ساده و knn، درخت‌های تصمیم می‌توانند مستقیماً با جدول داده‌ها، بدون هیچ طراحی قبلی، کار کنند.

با تقسیم‌های مختلف مجموعه دادگان این مسئله، به دو بخش آموزش و تست، در هر الگوریتم به دقت‌های متفاوتی می‌رسیم که نشان می‌دهد که الگوریتم جنگل تصادفی با کم شدن داده‌های آموزشی، دقتش بسیار کاهش پیدا می‌کند، حال آنکه این عمل برای knn و بیز ساده فرق چندانی ایجاد نمی‌کند. حتی برای knn تاثیر بهتری نیز دارد.



تهیه‌کننده: فاطمه طاهر ـ ۴۰۰۱۰۰۰۶۹۷