بسمه تعالی

جناب آقای دکتر فراهانی

نام و نام خانوادگی: سید مهرداد دستوری

شماره دانشچویی: ۹۹۴۲۲۰۸۳

تمرین سری ۳واحد درسی داده کاوی

سوال ۱:

علت استفاده از کرنل ها :

علت استفاده از کرنل ها برای این است که داده ورودی را به یک داده دیگر و یک داده جدید در فضای جدید نگاشت کند، علت استفاده این است که چون svm در حالت کلی نمیتواند هر داده ای را جدا کند و بحث وکتورها برای همین از کرنل ها استفاده میشود تا انها را به فضای جدید نگاشت کنیم و بتوانیم از آنها استفاده کنیم، پس در حالت کلی وظیفه کرنل این است که داده را به عنوان ورودی بگیرد و آن را به شکل مورد نظر تبدیل کند،

کاربردهای هر کرنلی عموما با یک حالت و پس زمینه خاصی می آید، مثلا کرنل چندجمله ای برای پردازش تصاویر استفاده میشود، کرنل RBF برای اهداف عمومی، کرنل تانژانت هیپربولیک و همچنین کرنل سیگموئید برای شبکه های عصبی استفاده میشوند.

برخی از کرنل های رایج مورد استفاده در SVM ها و کاربرد های آن ها :

۱- کرنل چند جمله ای

این کرنل در پردازش تصویر پرکاربرد است.

۲- کرنل گاوسی

این یک کرنل برای اهداف عمومی است. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود ندارد استفاده می شود.

۳- تابع پایه شعاعی گاوسی (RBF)

این کرنلی برای اهداف عمومی کاربرد دارد. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود نداشته باشد، مورد استفاده قرار می گیرد

۴- کرنل RBF لاپلاس

این هم یک کرنل برای اهداف عمومی است. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود ندارد استفاده می شود.

۵- کرنل تانژانت هیپربولیک (tanh)

می توانیم از آن در شبکه های عصبی استفاده کنیم.

۶- کرنل سیگموئید

می توان این کرنل را در شبکه های عصبی مورد استفاده قرار داد.

۷- کرنل تابع بِسِل (Bessel) از نوع اول

ما می توانیم از آن برای حذف مقطع عرضی در توابع ریاضی استفاده کنیم.

۸- کرنل پایه شعاعی ANOVA

ما می توانیم از آن در مسائل رگرسیون استفاده کنیم.

۹- کرنل spline خطی بصورت یک بعدی

این کرنل، هنگام کار با بردارهای بزرگ داده پراکنده ، کاربرد زیادی دارد. این کرنل اغلب در دسته بندی متن مورد استفاده قرار می گیرد. کرنل spline همچنین در مسائل رگرسیون عملکرد خوبی دارد.

سوال ۲:

بر روی دیتاست کلاس بندی قیمت موبایل الگوریتم SVM را با تابع svm_func_base اجرا کردیم ، نحوه عملکرد این تابع بدین گونه است که svm اجرا شده، روی داده ها fit شده و predict و در نهایت تابع plot_results برای نتایج فراخوانی میشود تا نتایج را نشان بدهد

نتایج حاصل بعد از اجرا به این صورت میباشد که به صورت نمودار نمایش داده میشود و confusion_matrix نیز در خروجی چاپ میشود

accuracy = ۰.۸۵۲

percision = ۰.۸۵۵

recall = ۰.۸۵۳

f۱ = ۰.۸۵۳_score

سوال ۳:

برای حالت های مختلف ما یک تابع svm_func_۲ تعریف کردیم که در آن kernel = linear و خطی در نظر گرفته شد و gamma = auto مقداردهی شد که نتایج به صورت زیر است :

accuracy = ۰.۹۴۵

percision = ۰.۹۴۴

recall = ۰.۹۴۴

f۱ = ۰.۹۴۴_score

حالت بعدی با تابع svm_func_۳ که در آن kernel خطی و gamma = scale در نظر گرفته شد تعریف و اجرا کردیم

accuracy = ۰.۹۴۵

percision = ۰.۹۴۴

recall = ۰.۹۴۴

f۱ = ۰.۹۴۴_score

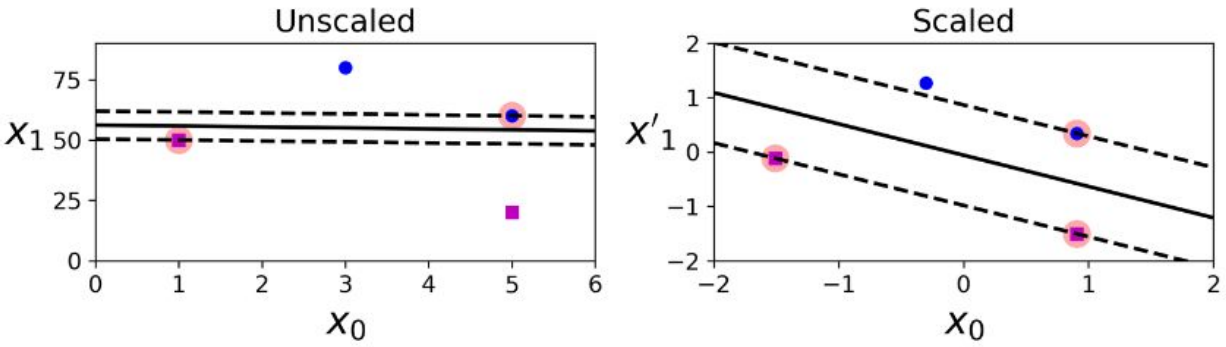
حالت بعدی با تابع svm_func_۴ که در آن kernel = poly و gamma = auto در نظر گرفته شد تعریف و اجرا کردیم

حالت بعدی با تابع svm_func_۵ که در آن kernel = poly و gamma = scale در نظر گرفته شد تعریف و اجرا کردیم

حالت بعدی با تابع svm_func_۶ که در آن پارامتر class_weight =balanced و decision_function_shape = ovo در نظر گرفته شده تعریف و اجرا شد

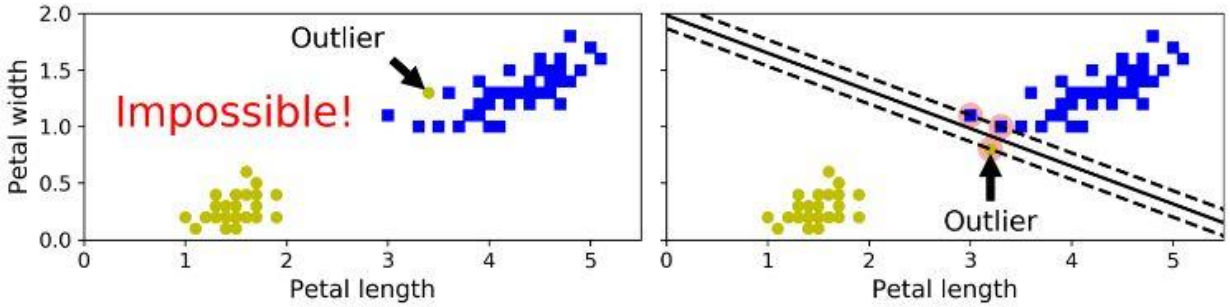
سوال ۴:

الگوریتم SVM به شدت به مقیاس داده ها حساس است. همون طور که در شکل سمت چپ می‌بینید مقیاس عمودی بیشتر از افقی هست بخاطر همین فضای خالی نزدیک به افق هست. بعد از تغییر مقیاس داده ها با استفاده از StandardScaler واقع در کتابخانه Scikit-Learn میبینیم که مرز های تصمیم گیری شکل بهتری پیدا کردند.



طبقه بندی حاشیه نرم

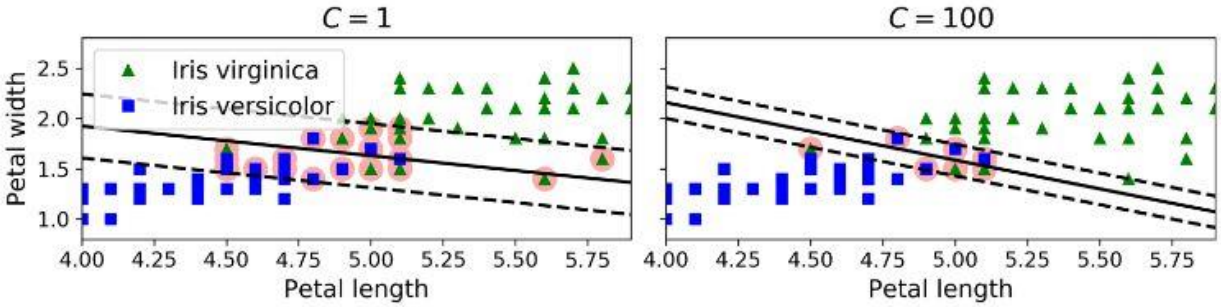
اگر تحمیل کنیم که تمام نمونه ها باید خارج از فضای خالی میانی باشند و حتما باید در سمت مناسب قرار بگیرند، در واقع روش Hard Margin Classification (طبقه بندی حاشیه سخت)نام دارد، این روش دو مشکل اساسی دارد، اول اینکه فقط برای داده‌هایی کار میکند که به‌صورت خطی جدا میشوند. دوم اینکه به داده های پرت به‌شدت حساس است. شکلی که می‌بینید دیتاست Iris رو نشان میدهد که در سمت چپ یک داده پرت وجود دارد و پیدا کردن یک Hard Margin برای آن غیر ممکن است. در سمت راست هم مرز تصمیم‌گیری با چیزی که در بالا دیدیم، خیلی متفاوت و احتمالا به‌خوبی نمیتواند عمومی‌سازی کند.



برای جلوگیری از این مشکلات، از یک مدل انعطاف پذیر تر استفاده می‌کنیم. هدف ما پیدا کردن یک تعادل مناسب بین بیشترین مقدار فضای خالی میانی و محدود کردن Margin Violations هست (نقض حاشیه یعنی قرار گرفتن نمونه ها در فضای میانی یا در سمت اشتباه) به این کار Soft Margin Classification میگویند

وقتی مدل های SVM رو با استفاده از Scikit-Learn می‌سازیم، یک سری هایپرپارامتر ها رو می‌توانیم تنظیم کنیم. یکی از این هایپرپارامتر ها C نام دارد. اگر مقدار C کم باشد، حاصل مدل

سمت چپ شکل زیر است، با مقدار زیاد به شکل سمت راست می‌رسیم. نقض حاشیه اتفاق خوبی نیست و بهتر است تا جایی که امکان دارد کمتر اتفاق بیفتد، اگرچه، در این مورد، درست است که مدل سمت چپ نقض حاشیه های زیادی دارد اما احتمالا بهتر عمومی‌سازی می‌کند. در واقع هایپرپارامتر C مقدار مجازات مدل برای هر نمونه‌ای را که به صورت اشتباه طبقه بندی میکند تعیین میکند. هر چقدر C کمتر باشد، حاشیه نرم تر میشود



سوال ۴ :

ما برای این کار تابع soft_margin_svm را تعریف کردیم که مقدار = c۰.۱ در نظر گرفتیم که بعد از فراخوانی نتیجه به صورت زیر حاصل شد :

accuracy = ۰.۸۸۵

percision = ۰.۸۸۷

recall = ۰.۸۸۵

f۱ = ۰.۸۸۵_score

همچنین hard_margin_svm را تعریف کردیم مقدار = c۱۰۰۰ برای آن در نظر گرفتیم که بعد از اجرا نتایج زیر حاصل شد :

accuracy = ۰.۹۵۸

percision = ۰.۹۵۷

recall = ۰.۹۵۷

۰.۹۵۷_score = ۱f

پس هر چقدر مقدار c بالاتر باشد دقت بیشتر است

سوال ۵:

آ)

بعد از انتخاب فیچر battery_power مقدار =bin۱۰ قرار دادیم و با دستور pd_cut این بین ها را اجرا کردیم نتیجه به این صورت شد که برای فیچر battery_power تقسیم بندی بازه انجام شد و داده ها دسته بندی شدند بعد از نرمالایز کردن داده ها و فراخوانی svm بر روی آن نتایج زیر حاصل شد :

۰.۹۴۲accuracy =

۰.۹۴۲percision =

۰.۹۴۱recall =

۰.۹۴۱_score = ۱f

یکبار دیگر با = ۲۰bin اینکار را کردیم و نتایج زیر حاصل شد

۰.۹۲۰accuracy =

۰.۹۱۹percision =

۰.۹۱۹recall =

۰.۹۱۹_score = ۱f

که در واقع دقت پایینتر شد

یکبار هم با = ۳۰bin اینکار را انجام دادیم و نتایج زیر حاصل شد :

۰.۹۱۲accuracy =

percision = ۰.۹۱۲

recall = ۰.۹۱۳

f۱ = ۰.۹۱۲_score

که باز هم کمتر شد پس = bin۱۰ نتایج بهتری دارد

ج)

ما در کد از log transform استفاده کردیم و از دیتاها لگاریتم گرفتیم و نتایج به صورت زیر حاصل شد :

accuracy = ۰.۹۱۰

percision = ۰.۹۰۹

recall = ۰.۹۰۸

f۱ = ۰.۹۰۸_score

داده هایی که نتیجه بی نهایت داشتند حذف شدند

علاوه بر log transform از تبدیل standard scaler transform نیز استفاده کردیم که نتایج به صورت زیر حاصل شد :

accuracy = ۰.۹۵۲

percision = ۰.۹۵۲

recall = ۰.۹۵۱

f۱ = ۰.۹۵۱_score

که نتایج بهتری نسبت به log transform دارد

د)

یک ویژگی به عنوان مساحت یا حجم گوشی ساختیم که px_weight و px_height را به آن دادیم و بعد از ضرب آن ها در هم یک ویژگی جدید مساحت ساخته شد که روی آن پیش بینی انجام میشود که نتایج به صورت زیر حاصل شد :

accuracy = ۰.۹۴۰

percision = ۰.۹۳۹

recall = ۰.۹۳۹

۱f_score = ۰.۹۳۹

سوال ۶:

الگوریتم svm را بر روی همه فیچرها اعمال کردیم و نتایج زیر حاصل شد :

accuracy = ۰.۸۲۵

percision = ۰.۸۲۷

recall = ۰.۸۲۳

۱f_score = ۰.۸۲۵

سوال ۷:

الگوریتم های مختلف درخت تصمیم:

الگوریتم ID۳ :

یکی از الگوریتم های بسیار ساده درخت تصمیم که در سال ۱۹۸۶ توسط Quinlan مطرح شده است. اطلاعات به دست آمده به عنوان معیار تفکیک به کار می رود. این الگوریتم هیچ فرایند هرس کردن را به کار نمی برد و مقادیر اسمی و مفقوده را مورد توجه قرار نمی دهد.

در این الگوریتم درخت تصمیم از بالا به پایین ساخته می‌شود. این الگوریتم با این سوال شروع می‌شود: کدام ویژگی باید در ریشه درخت مورد آزمایش، قرار بگیرد؟ برای یافتن جواب از معیار بهره اطلاعات استفاده می‌شود. با انتخاب این ویژگی، برای هر یک از مقادیر ممکن آن یک شاخه ایجاد شده و نمونه های آموزشی بر اساس ویژگی هر شاخه مرتب می‌شوند. سپس عملیات فوق برای نمونه های قرار گرفته در هر شاخه تکرار می شوند تا بهترین ویژگی برای گره بعدی انتخاب شود.

الگوریتم ۴.۵C :

این الگوریتم درخت تصمیم، تکامل یافته ۳ID است که در سال ۱۹۹۳ توسط Quinlan مطرح شده است که از معیار نسبت بهره(Gain ratio) استفاده می کند. الگوریتم هنگامی متوقف می‌شود که تعداد نمونه ها کمتر از مقدار مشخص شده‌ای باشد. این الگوریتم از تکنیک پس هرس استفاده می‌کند و همانند الگوریتم قبلی داده‌های عددی را نیز می پذیرد.

از نقاطِ ضعف الگوریتم ۳ID که در ۴.۵C رفع شده است می‌توان به موارد زیر اشاره کرد:

الگوریتم ۴.۵C می‌تواند مقادیر گسسته یا پیوسته را در ویژگی‌ها درک کندوالگوریتم ۴.۵C قادر است با وجود مقادیر گمشده نیز درخت تصمیم(decision tree) خود را بسازد، در حالی که الگوریتمی مانند ۳ID و بسیاری دیگر از الگوریتم‌های طبقه‌بندی نمی‌توانند با وجود مقادیر گمشده، مدلِ خود را بسازند.سومین موردی که باعث بهینه شدن الگوریتم ۴.۵C نسبت به ۳ID می‌شود، عملیاتِ هرس کردن جهت جلوگیری از بیش برازش می‌باشد. الگوریتم‌هایی مانند ۳ID به خاطر اینکه سعی دارند تا حد امکان شاخه و برگ داشته باشند (تا به نتیجه مورد نظر برسند) با احتمال بالاتری دارای پیچیدگی در ساخت مدل و این پیچیدگی در بسیاری از موارد الگوریتم را دچار بیش برازش و خطای بالا می‌کند.اما باعملیات هرس کردن درخت که در الگوریتم ۵ انجام می‌شود، می‌توان

مدل را به یک نقطه بهینه رساند که زیاد پیچیده نباشد (و البته زیاد هم ساده نباشد) و بیش برازش یا کم برازش(Underfitting) رخ ندهد.الگوریتم C۴.۵ این قابلیت را دارد که وزن‌های مختلف و غیر یکسانی را به برخی از ویژگی‌ها بدهد.

الگوریتم CART :

برای برقراری درخت های رگرسیون و دسته بندی از این الگوریتم استفاده می شود. در سال ۱۹۸۴توسط Breiman و همکارانش ارائه شده است. نکته حائز اهمیت این است که این الگوریتم درخت های باینری ایجاد می کند به طوری که از هر گره داخلی دو لبه از آن خارج می شود و درخت های بدست آمده توسط روش اثربخشی هزینه، هرس می شوند.

یکی از ویژگی های این الگوریتم، توانایی در تولید درخت های رگرسیون است. در این نوع از درخت ها برگ‌ها به جای کلاس مقدار واقعی را پیش بینی می کنند. الگوریتم برای تفکیک کننده ها، میزان مینیمم مربع خطا را جستجو می کند. در هر برگ، مقدار پیش بینی بر اساس میانگین خطای گره ها می باشد.

الگوریتم CHID :

این الگوریتم درخت تصمیم به جهت در نظرگرفتن مشخصه های اسمی در سال ۱۹۸۱ توسط Kass طراحی شده است. الگوریتم برای هر مشخصه ورودی یک جفت مقدار که حداقل تفاوت را با مشخصه هدف داشته باشد، پیدا می کند.

این الگوریتم با توجه به نوع برچسب کلاس از آزمون‌های مختلف آماری استفاده می‌کند. این الگوریتم هرگاه به حداکثر عمق تعریف شده‌ای برسد و یا تعداد نمونه‌ها در گره جاری از مقدار تعریف شده‌ای کمتر باشد، متوقف می‌شود. الگوریتم CHAID هیچگونه روش هرسی را اجرا نمی‌کند.

سوال ۸:

تابع DT_func_base را برای ساخت درخت تعریف کردیم که نتایج بعد از اجرا به صورت زیر حاصل شد:

accuracy = ۰.۸۴۰

percision = ۰.۸۳۷

recall = ۰.۸۳۹

f۱ = ۰.۸۳۸_score

سوال ۹:

تابع _DT_func۲ برای بررسی پارامترهای مختلف تعریف شد که در این تابع پارامتر criterion = gini و پارامتر max_depth = ۵ لحاظ شد و نتایج به صورت زیر حاصل شد :

accuracy = ۰.۸۲۸

percision = ۰.۸۲۸

recall = ۰.۸۲۵

f۱ = ۰.۸۲۲_score

تابع _DT_func۳ تعریف شد و پارامتر max_depth = ۲۰ و پارامتر criterion = gini قرار داده شد و نتایج به صورت زیر حاصل شد :

accuracy = ۰.۸۲۰

percision = ۰.۸۱۹

recall = ۰.۸۱۸

f۱ = ۰.۸۱۸_score

تابع _DT_func۴ تعریف شد و max_depth = ۵۰ و پارامتر criterion = gini قرار داده شد

تابع _DT_func۵ تعریف شد و پارامتر criterion = entropy و max_depth = ۵ قرار داده شد

تابع_DT_func۶ تعریف شد و پارامتر criterion = entropy و پارامتر max_depth = ۲۰ قرار داده شد

تابع _DT_func۷ تعریف شد و پارامتر criterion = entropy و max_depth = ۵ و پارامتر min_samples_split = ۵ قرار داده شد

تابع _DT_func۸ تعریف شد و پارامتر criterion = entropy و max_depth = ۵ و پارامتر min_samples_split = ۱۰ قرار داده شد

تابع _DT_func۹ تعریف شد و پارامتر criterion = entropy و max_depth = ۵ و پارامتر max_features = ۱۰ قرار داده شد

تابع _DT_func۱۰ تعریف شد و پارامتر criterion = entropy و max_depth = ۵ و پارامتر max_features = ۱۵ قرار داده شد و نتایج زیر حاصل شد :

accuracy = ۰.۷۳۸

percision = ۰.۷۲۵

recall = ۰.۷۲۷

f۱ = ۰.۷۲۵_score

که دقت کمتر شده

سوال ۱۰:

هرس کردن درخت تصمیم

هرس کردن درخت تصمیم مقابل عمل تقسیم کردن است و با هرس کردن زیر گره هایی در درخت تصمیم حذف می گردد. زمانی که یک درخت تصمیم ساخته می شود، تعدادی از شاخه ها ناهنجاری هایی در داده های آموزش منعکس می کنند که ناشی از داده های پرت و یا نویز است.

در برخی الگوریتم های ایجاد درخت، هرس کردن جزئی از الگوریتم محسوب می شود. در حالی که در برخی دیگر، تنها برای رفع مشکل بیش برازش از هرس کردن استفاده می شود.

چندین روش، معیارهای آماری را برای حذف کمتر شاخه‌های قابل اطمینان به کار می برند.
درخت های هرس شده تمایل به کوچک‌تر بودن و پیچیدگی کم تر دارند و بنابراین به راحتی قابل فهم می باشند. آن‌ها معمولا در طبقه‌بندی صحیح داده های تست سریع تر و بهتر از درخت های هرس نشده عمل می کنند.

دو رویکرد رایج برای هرس درخت به شرح ذیل وجود دارد:

پیش هرس (Pre pruning) :

در این رویکرد یک درخت به وسیله توقف های مکرر در مراحل اولیه ساخت درخت، هرس می شود. به محض ایجاد یک توقف گره به برگ تبدیل می شود.

هرس پسین (Post pruning) :

رویکرد هرس پسین درخت تصمیم رایج تر است به این صورت که زیر درخت ها از یک درخت رشد یافته کامل را حذف می‌کند. یک زیر درخت در یک گره به وسیله حذف کردن شاخه‌ها و جایگزینی آن‌ها با یک برگ، هرس می‌شود.

علت هرس کردن درخت : حذف داده های پرت یا ساده کردن مدل میباشد

سوال ۱۱:

نتیجه هرس کردن در کد

ما با معیار ccp_alpha در کد مشخص کردیم که چقدر هرس کردن اعمال شود، ۰.۰۱ یا ۰.۰۰۱ یا ... ، برای ۰.۰۱ نتیجه حدودا ۰.۷۵ میشود، با ۰.۰۰۱ که هرس کردیم دقت بالاتر رفت و نتیجه ۰.۸۲ را به ما داد پس تاثیرات مختلفی بر روی داده ها اعمال میشود

سوال ۱۲:

مجموعه ای از درختهای تصمیم random forest است که در انتها از ان درختها میانگین گیری میکند تا نتایج را پیش بینی کند، بعد از ساخت random forest و تخصیص مقادیر پارامترها نتایج مشخص میشود

سوال ۱۳:

از دیگر مزایای درخت تصمیم قابلیت تفسیرپذیری آن است. درخت های تصمیم ذات تفسیرپذیر و قابل درکی دارند که به استدلال آدمی شبیه است؛ .میتوان از درخت تشکیل شده یکسری قانون به صورت if-then استخراج کرد. از دیگر مزایای درخت های تصمیم میتوان به موارد زیر اشاره کرد:

درخت تصمیم توانایی کار با داده های پیوسته و گسسته را دارد .
-درخت تصمیم از نواحی تصمیم گیری ساده استفاده می کند .
-مقایسه های غیرضروری در این ساختار حذف می شوند .
-از ویژگی های متفاوت برای نمونه های مختلف استفاده میشود .
-نیازی به تخمین تابع توزیع نیست .
-آماده سازی داده ها برای یک درخت تصمیم، ساده یا غیر ضروری است .
-درخت تصمیم یک مدل جعبه سفید است. توصیف شرایط در درختان تصمیم به آسانی و با منطق بولی امکان پذیر است. در حالی که شبکه های عصبی به دلیل پیچیدگی در توصیف

نتایج آنها، مدل جعبه سیاه می باشند .
-تأیید یک مدل در درخت های تصمیم با استفاده از ارزیابی های آماری امکان پذیر است .
-ساختارهای درخت تصمیم برای تحلیل داده های بزرگ در زمان کوتاه قدرتمند می باشند .
-روابط غیرمنتظره یا نامعلوم را می یابند .
-درختهای تصمیم قادر به شناسایی تفاوت های زیر گروه ها می باشند .
-درختهای تصمیم قادر به سازگار کردن داده های فاقد مقدار هستند .
-روش های درخت تصمیم به ویژه در آشکار کردن تراکنش های پیچیده بین متغیرها بسیار توانمند هستند. هر شاخه ای از درخت می تواند شامل ترکیبات مختلفی از متغیرها باشد و متغیرهای یکسان می توانند بیش از یک بار در قسمتهای مختلف درخت ظاهر شوند. این امر می تواند مشخص کند که چگونه یک متغیر می تواند وابسته به متغیری دیگر باشد.

سوال ۱۴:

شماری از تکنیک‌های موفق و معروف یادگیری ماشین، نظیر طبقه‌بندی‌کننده‌های تقویتی، بردارهای پشتیبان و روش RIPPER در اصل برای مسائل دوکلاسه طراحی شده‌اند. البته لازم به ذکر است که روش RIPPER با هدف حل مسائل چندکلاسه تعریف شد اما این روش در واقع حاصل ترکیب دو روش ۸REP وIREP می‌باشد که هر دوی این روش‌ها در حوزه‌ی مسائل دوکلاسه تعریف شده‌اند. اما واقعیت این است که بسیاری از مسائل طبقه‌بندی در دنیای واقعی ابدا دوکلاسه نیستند بلکه متعلق به مسائل چندکلاسه می‌باشند. احتمال طبقه‌بندی نادرست در مسائل چندکلاسه بسیار بالاست و این احتمال با بالا رفتن تعداد کلاس‌ها، به سرعت افزایش می‌یابد، بنابراین واضح است که رسیدن به دقت بالا، در مسائل چندکلاسه، بسیار مشکل‌تر از مسائل دوکلاسه است.

تاکنون روش‌های موثر متفاوتی برای حل مسائل چندکلاسه ارائه شده است. این روش‌ها را می توان به دو دسته کلی تقسیم کرد:

روش‌های مبتنی بر تقسیم-و-تسخیر

روش‌های مبتنی بر تفکیک-و-تسخیر

دسته روش‌های تقسیم-و-تسخیر بر مبنای ایجاد درخت تصمیم‌گیری، بر روی داده‌های آموزشی استوار هستند. به این صورت که نقاط تصمیم‌گیری، در هر لایه از درخت تصمیم‌گیری، بر اساس آزمایش بر انواع ویژگی‌های داده، محاسبه می‌شود و داده‌ها بر اساس این آزمایش، طبقه‌بندی می‌شوند. این روند به صورت بازگشتی ، آنقدر انجام می‌شود تا زمانی که دقت مورد نظر کسب شود. از جمله الگوریتم‌های متعلق به این دسته، می‌توان به ID۳ و CART و C۴.۵ و توسعه یافته آن C۵.۰ اشاره کرد که پیچیده‌ترین روش‌های القای درختی ، در دو دهه اخیر هستند

در روش‌های مبتنی بر تفکیک-و-تسخیر، الگوریتم در هر زمان فقط بر یک کلاس تمرکز می‌کند. به بیان دیگر الگوریتم‌های این دسته سعی دارند که در هر زمان، قوانینی، را تولید کنند که بیشترین تعداد داده متعلق به کلاس جاری را به درستی پیدا کرده و تا حد ممکن داده‌های غیر از آن کلاس را نادیده بگیرند. پس از هر دور تولید قانون به روش مذکور، داده‌هایی که درست دسته‌بندی شده‌اند از داده‌های آموزشی حذف شده، سپس مابقی داده‌ها در دورهای آینده به صورت تکرار شونده، با روالی مشابه، مورد آموزش قرار می‌گیرند. الگوریتم‌هایی نظیر IREP و PRISM و RIPPER معروف‌ترین و موثرترین الگوریتم‌های این دسته می‌باشند.

سوال ۱۵:

بله میتوان از درخت تصمیم برای حل مسائل سری استفاده کرد ولی باید انها را به دیتای سری زمانی تبدیل کرد که عموما از Random forest برای این کار استفاده میکنند

سوال ۱۶:

برای تغییرات ستون date و حذف کاراکترهای اضافی از داده ها و تبدیل دیتاهای string به numeric باید بر روی انها پیش پردازش انجام شود که ما اینکار را در کد با استفاده از تابع bit_data انجام دادیم

سوال ۱۷:

توابع مختلف تعریف شده :

Linear_regression_func

که ابتدا یک ابجکت از آن ها ساخته میشود، سپس رویدادهای train را فیت میکنیم ، رویدادهای test را پیش بینی میکنیم و مقادیر mse را پیش بینی میکنیم، بعد مقادیر rmse وقتی که squared = false باشد حساب میشود ، مقدار mae و r۲ محاسبه میشوند، بعد مقدار دقت با تابع get_acc محاسبه شده و خروجی نهایتا در قالب اکسل ذخیره میشود

ridge_regression_func

lasso_regression_func

knn_regression_func

dt_regression_func

mlp_regression_func

random_frst_regression_func

bagging_regression_func

voting_regression_func

adaboost_regression_func

boosting_regression_func

نتایج بعد از اجرای این الگوریتم ها :

linear_regression_func

mse = ۰.۰۰۰

rmse = ۰.۰۰۸

mae = ۰.۰۰۴

r2 = ۱.۰۰

accuracy = ۹۹.۷۹۴

ridge_regression_func

mse = ۰.۰۰۰

rmse = ۰.۰۲۲

mae = ۰.۰۱۴

r2 = ۰.۹۹

accuracy = ۹۸.۷۶۵

dt_regression_func

mse = ۰.۰۵۹

rmse = ۰.۲۴۳

mae = ۰.۱۲۰

r2 = ۰.۱۹

accuracy = ۷۰.۹۸۸

votting_regression_func

mse = ۰.۰۱۵

rmse = ۰.۱۲۱

mae = ۰.۰۶۰

r۲ = ۰.۸۰

accuracy = ۷۲.۶۳۴

lasso_regression_func

mse = ۰.۱۵۵

rmse = ۰.۳۹۴

mae = ۰.۲۸۶

r۲ = - ۱.۱۱

accuracy = ۰

mlp_regression_func

mse = ۰.۰۰۳

rmse = ۰.۰۵۸

mae = ۰.۰۲۹

r۲ = ۰.۹۵

accuracy = ۷۸.۸۰۷

random_frst_regression_func

mse = ۰.۰۵۸

rmse = ۰.۲۴۲

mae = ۰.۱۱۸

r۲ = ۰.۱۹

accuracy = ۷۱.۳۹۹

bagging_regression_func

mse = ۰.۰۵۸

rmse = ۰.۲۴۲

mae = ۰.۱۱۸

r۲ = ۰.۱۹

accuracy = ۷۱.۳۹۹

voting_regression_func

mse = ۰.۰۱۴

rmse = ۰.۱۲۰

mae = ۰.۰۵۹

r۲ = ۰.۷۹

accuracy = ۷۲.۶۳۴

adaboost_regression_func

۰.۰۶۰mse =

۰.۲۴۶rmse =

۰.۱۲۲mae =

۰.۱۶= ۲r

۷۱.۳۹۹accuracy =

ما بر روی داده های بیت کوین دو مدل یادگیری عمیق زدیم : CNN و LSTM

برای کدهای یادگیری عمیق از بستر tensorflow و کتابخانه keras استفاده کردیم ما ابتدا یک لایه input ساختیم، داده های train را به آن پاس دادیم، shape داده را مشخص کردیم که داده ها چند ویژگی دارند، بعد مدل یادگیری عمیق را ساختیم و لایه input را به آن پاس دادیم و بقیه لایه ها را به آن اضافه کردیم ، پارامترهای هر لایه را مشخص کرده و به آن ها اختصاص دادیم، مثلا برای لایه کانولوشنال از فیلتر ۱۶ استفاده کردیم، سپس لایه D۱MaxPool و بعد لایه flatten را استفاده کردیم تا تبدیل به وکتورهای یک بعدی بشوند که بتوانیم لایه های dense را روی آنها اعمال کنیم ، بعد هم لایه های dense با نورون های مختلف استفاده کردیم و در انتها لایه خروجی با نام out_layer اورده شد که acctivation را برای آن sigmoid قرار دادیم چون رگرسیون داریم و تعداد نورون را ۱ قرار دادیم، بعد از اجرا یک مدل از CNN ساخته شد ، سپس مدل را کامپایل کردیم ، در کامپایل هم پارامترهای نوع optimizer و loss function و metric ها را مشخص کردیم و بعد از اجرا و کامپایل یک خروجی به صورت گرافیکی نمایش داده میشود که لایه ها و shape آنها و تعداد خروجی هایشان را شامل میشود

بعد از اینکار مدل را آموزش دادیم، برای اینکار ابتدا داده ها را shape میکنیم تا به صورت مناسب برای استفاده الگوریتم های یادگیری عمیق شود بعد مدل را fit میکنیم داده train و label را به آن پاس میدهیم و یک = epochs۱۰ به آن پاس میدهیم که مشخص کننده تعداد اجرا میباشد و batch_size هم برای اینکه بداند چندتا چندتا داده ها را پاس بدهد ، بعد از اجرا الگوریتم شروع به

train کردن مدل میکند و مدل را آموزش میدهد تا نهایتا بتوانیم از مدل برای پیش بینی استفاده کنیم، بعد از اجرا میبینم که مقدار loss و rmse تقریبا به صفر میرسد

بعد از آن برای تست مدل و پیش بینی دقت از تابع get_acc استفاده کردیم که بعد از فراخوانی predict نتایج زیر حاصل شد :

CNN_result

۰.۰۱۶mse =

۰.۱۲۷rmse =

۰.۰۷۵mae =

۰.۷۸= ۲r

۳۴.۵۶۸accuracy =

بعد از آن یک مدل LSTM ساختیم ، مجددا مشابه مدل قبل لایه ورودی ، input shape و تعداد ویژگی ها را مشخص کردیم، لایه های بعدی و لایه آخر مجددا کامپایل و رسم شد ، سپس آموزش و fit کردن مدل را انجام دادیم و بعد با تابع get_acc تست را انجام دادیم و نتایج زیر حاصل شد :

LSTM_result

۰.۰۰۴mse =

۰.۰۶۰rmse =

۰.۰۲۸mae =

۰.۹۵= ۲r

۸۲.۰۹۹accuracy =

ما برای این تست = ۱۰۰ephocs قرار دادیم که هر چه این عدد بالاتر باشد نتایج دقیقتر میشود و دقت بالاتری به دست می آید

سوال ۱۸:

استفاده از ایده ensemnle learning بر روی ۵ تا ار بهترین مدل ها و سپس اجراب تکنیک های bagging و votting و boosting نتایج به این صورت میباشد:

برای bagging از mlp به دلیل دقت بالا استفاده کردیم

برای votting از linear_regression و ridge استفاده کردیم و در boosting هم GradientBoostingRegressor استفاده شده

votting_model

۰.۰۰۰mse =

۰.۰۱۳rmse =

۰.۰۰۸mae =

۱.۰۰= ۲r

۹۹.۵۸۸accuracy =

که دقت بسیار بالایی دارد

bagging model

۰.۰۰۶mse =

۰.۰۷۴rmse =

۰.۰۳۸mae =

۰.۹۲= ۲r

۷۵.۳۰۹accuracy =

سوال ۱۹:

برای اجرای الگوریتم Adaboost بر روی دیتاست بیتکوین ما ۵ تابع مختلف با پارامترهای متفاوت با نام های _adaboost۱ تا _adaboost۵ تعریف کرده و اجرا کردیم که نتایج زیر حاصل شد :

Adaboost_regression_result_on_bit

mse = ۰.۲۴۸

rmse = ۰.۱۵۷

mae = ۰.۷۸۶

r۲ = ۰.۱۶

accuracy = ۷۱.۴

base_estimator_linear_regression

mse = ۰.۰۰۰

rmse = ۰.۸۰۰

mae = ۰.۴۰۰

r۲ = ۰.۹۹

accuracy = ۹۹.۸۰۰

n_stimators_۲۰۰

mse = ۰.۶۱۰

rmse = ۰.۲۴۳

mae = ۰.۱۲۳

r2 =۰.۱۶

accuracy =۷۱.۴۰۱

loss_square

mse =۰.۶۲۰

rmse =۰.۲۴۸

mae =۰.۱۲۴

r2 =۰.۱۶

accuracy =۷۱.۴۰۱

learning_rate_۱

mse =۰.۲۴۷

rmse =۰.۱۵۷

mae =۰.۷۸۳

r2 =۰.۱۶

accuracy =۷۱.۴۰۱

learning_rate_۰.۰۰۱

mse =۰.۷۴۰

rmse =۰.۲۷۲

mae =۰.۱۴۴

r2 =۰.۰۵

accuracy =۵۲.۱۰۱

سوال ۲۰:

برای اجرای الگوریتم Random forest بر روی دیتاست بیتکوین ما ۵ تابع مختلف با پارامترهای متفاوت با نام های random_frst_۱ تا random_frst_۵ تعریف کرده و اجرا کردیم که نتایج زیر حاصل شد :

Random_forest_default_parameters

mse =۰.۶۱۰

rmse =۰.۲۴۷

mae =۰.۱۲۳

r2 =۰.۱۷

accuracy =۷۱.۴۰۱

max_deep_۳

mse =۰.۷۴۰

rmse =۰.۲۷۱

mae =۰.۱۴۲

r2 =۰.۰۲

accuracy =۶۱.۵۰۱

max_deep_۹

mse = ٠.٥٩٠

rmse = ٠.٢٤٣

mae = ٠.١١٩

r٢ = ٠.١٩

accuracy = ٧١.٤٠١

max_features_sqrt

mse = ٠.٦٠٠

rmse = ٠.٢٤٦

mae = ٠.١٢١

r٢ = ٠.١٧

accuracy = ٧١.٤٠١

n_stimators_ ٣٠٠

mse = ٠.٥٩٠

rmse = ٠.٢٤٣

mae = ٠.١١٩

r٢ = ٠.١٩

accuracy = ٧١.٤٠١

سوال ۲۱:

ما برای دسته بندی پیش بینی ها که اگر مقدار واقعی با مقدار پیش بینی شده ۰.۱ یا ۱۰ درصد اختلاف داشت ان را قبول کند و اگر نبود نه ، تابعی به نام get_acc تعریف کردیم که دو مقدار واقعی و پیش بینی شده را به عنوان ورودی میگیرد و دقت اختلاف را محاسبه میکند تا مقدار پیش بینی شده قبول یا رد شود

سوال ۲۲ :

برای اینکار ما variation ها را ساختیم به این صورت که اگر قیمت نسبت به روز قبل افزایشی باشد مقدار = variation۱ و اگر کاهش یافته باشد مقدار = variation۰ شود ، بعد train و test را جدا کردیم و مدل را روی داده fit کردیم تا نتایج حاصل شود

سوال ۲۳:

برای این کار ابتدا مسئله را به یک Classification تبدیل کردیم و اگر جدول داده ها را ببینیم یک ستون به نام variation اضافه کردیم که اگر قیمت افزایشی باشد ۱ و اگر کاهشی باشد ۱- مقدار میگیرد، در واقع از این فیچر جدید استفاده میکنیم تا نتایج را ببینیم ، بعد از نرمالسازی داده ها و اجرا و پاس کردن داده ها به xgboost نتایج حاصل شد و دقت بالاتر رفت

mse = ۰.۰۰۰

rmse = ۰.۰۱۸

mae = ۰.۰۱۱

r۲ = ۱.۰۰

accuracy = ۹۵.۴۷۳

سوال ۲۶:

حجم داده **tournement** خیلی زیاد بود و مجبور به اجرا در کولب شدیم، اولا چون هر چقدر حجم داده زیاد باشد زمان فیت کردن مدل و یادگیری زیاد میشود ، برای همین ما یک **chunk_size** در تابع **read_tournement_data** برای خواندن بخشی از داده ها تعریف کرده و بر روی آن اجرا کردیم، تابع **read_train_data** نیز برای خواندن داده های تِرین به کار بردیم ، تابع **get_acc** برای محاسبه دقت و تابع **save_result** برای محاسبه نتایج به کار بردیم

ابتدا داده را خواندیم، **train** و **test** را جدا کردیم ، یک **onehot encoding** بر روی داده ها اعمال کردیم به دلیل داشتن دو ستون غیر عددی تا به داده های عددی تبدیل شوند تا الگوریتم ها بتوانند بر روی آنها پردازش انجام دهند، سپس سه تا مدل مختلف **xgboost** و **Linear regression** و **ridge regression** را روی این داده ها فراخوانی کردیم ،در نهایت مدل داده ها را فیت کرده و نتایج و خطای هر کدام را نشان میدهد و خروجی را ذخیره میکند

سوال ۲۷:

ما برای این کار از تکنیک **bagging** و با تابع **bagging_regression_func** استفاده کردیم، **votting** را دوبار فراخوانی کردیم یکبار با **linear_regression** یا **random_forest** و یکبار با **MLPRegressor** و **RandomForestRegressor** و همچنین **boosting** را دوبار فراخوانی کردیم یکبار با **GradientBoostingRegressor** و یکبار با **DicisionTreeRegressor** فراخوانی شدند و خروجی ها ذخیره شدند که بخشی از نتایج را مشاهده میکنید

votting

۰.۰۲۴mse =

۰.۱۵۵rmse =

۰.۱۱۲mae =

۰.۵۲= ۲r

۴۶.۷۰۰accuracy =

votting

mse = ۰.۰۱۴

rmse = ۰.۱۱۹

mae = ۰.۰۹۰

r۲ = ۰.۷۲

accuracy = ۴۲.۳۲۵

boosting

mse = ۰.۰۴۸

rmse = ۰.۲۲۰

mae = ۰.۱۵۳

r۲ = ۰.۰۳

accuracy = ۴۹.۹۲۰

boosting

mse = ۰.۰۴۸

rmse = ۰.۲۱۹

mae = ۰.۱۵۶

r۲ = ۰.۰۴

accuracy = ۴۵.۶۵۰

سپس برای هر کدام spearman_correlation با استفاده از مقادیری که مدل های ensemble برگراندند محاسبه کردیم، برای این کار کتابخانه spearman را ایمپورت کردیم و مقادیر پیش بینی شده و لیبل های واقعی را به آن پاس دادیم تا آن را محاسبه کند، برای مشخص کردن کورولیت بودن سمپل ها از پارامتر p = ۰.۰۵ استفاده کردیم که اگر مقادیر بیشتر از این مقدار بود کورولیت نبوده و اگر کمتر بود کورولیت هستند که نتایج زیر حاصل شدند ، طبیعی است که مقادیر کورولیت هستند چون مقادیر پیش بینی شده به مقادیر واقعی نزدیک هستند

votting by linear Spearman correlation coefficient : ۰.۸۸۶

Samples are correlated (reject H۰ = p (۰.۰۰۰

votting by mlp Spearman correlation coefficient : ۰.۸۷۱

Samples are correlated (reject H۰ = p (۰.۰۰۰

boosting by gradient descent Spearman correlation coefficient : ۰.۲۳۵

Samples are correlated (reject H۰ = p (۰.۰۰۰

boosting by dicision tree Spearman correlation coefficient : ۰.۱۶۱

Samples are correlated (reject H۰ = p (۰.۰۰۰