تمرین سری ۳ مبانی یادگیری ماشین

مهرانه مقتدائی‌فر ۹۷۲۲۲۰۸۶

۱. در روش SVM میخواهیم داده ها را در فضای ویژگی ها از یکدیگر تفکیک کنیم. در بعضی مواقع داده ها به صورت خطی از هم جدا میشوند و انجام اینکار ساده است، اما در بیشتر موارد داده ها به صورت خطی جداپذیر نیستند. برای آنکه بتوانیم درست ترین ابرصفحه را برای تفکیک اینگونه داده ها انتخاب کنیم، نیاز داریم که بعد فضا را افزایش دهیم. در بعد بالاتر این عمل راحت تر انجام خواهد شد. این افزایش بعد فضاها توسط kernel هایی انجام میشود. این kernel ها درواقع توابعی هستند که با استفاده از ضرب داخلی باعث میشوند تا محاسبات ساده تر و سریع تر انجام شود و راحت تر بتوانیم داده ها را به بعد فضای بیشتر منتقل کنیم.

کرنل ها انواع مختلفی دارند، معروف ترین آنها کرنل های چندجمله‌ای و کرنل radial basis function یا همان RBF است.

برخی از کرنل های معروف در svm را باهم بررسی میکنیم:

Polynomial kernel:

معمولا در پردازش تصویر ها کاربرد دارد و تابع آن بسیار ساده است:

$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d$$

که d درجه چند جمله‌ای را مشخص میکند.

Gaussian kernel:

در همه موارد استفاده میشود به خصوص زمانی که هیچ دانش اولیه‌ای درمورد داده نداریم.

$$K(x, y) = e^{-}(\frac{||x-y||^2}{2\sigma^2})$$

Gaussian Radial Basis Function(RBF):

مانند کرنل قبلی است اما تفاوت آن این است که شعاعی را مشخص میکند و دقیق تر از حالت قبلی عمل میکند.

$$K(x, y) = e^{-}(\gamma||x - y||^2)$$

به طوریکه $\gamma > 0$ و در برخی موارد قرار میدهند: $\gamma = \frac{1}{2}\ \sigma^2$

Sigmoid kernel:

این تابع معمولا در شبکه های عصبی کاربرد دارد و همانند آن است که یک شبکه perceptron ۲ لایه تشکیل داده باشیم و از این تابع به عنوان تابع فعالساز استفاده کنیم:

$$K(x, y) = tanh(\gamma.x^T y + r)$$

۲. در ابتدا دیتاستی که در تمرین قبل هم از آن استفاده کردیم را از کگل لود میکنیم و از آنجایی که نیازی به تغییر در آن نیست، به *preprocessing* نیازی نداریم و فقط داده ها را *scale* میکنیم. سپس مدل *SVM* را با استفاده از پکیج موجود در *sklearn* پیاده سازی میکنیم.

۳. کرنل دیفالت این روش *RBF* است که در سوال قبلی همان را ران کردیم. حالا با کرنل های دیگری مدل را تست میکنیم.

نمی توانیم از کرنل *precomputed* استفاده کنیم چون ورودی مربعی از ما میخواهد. به همین دلیل خطا میدهد.

کرنل بعدی که آن را مورد استفاده قرار دادیم *linear* بود که خروجی بسیار بهتر و ۹۷ درصدی به ما داد. اما استفاده از *sigmoid* دقت ۹۲ درصدی را به ما داد.

تا به حال ما از *one versus all* یا همان *one versus rest* استفاده میکردیم (به صورت *default*) ، حال با تغییر به *one versus one* مشاهده میکنیم، که البته تغییی در نتیجه مشاهده نشد.

کرنل *polynomial* را نیز امتحان کردیم، مشاهده میشود که دقت بسیار پایین آمد، درجه چندجمله‌ای در حالت *default* برابر با ۳ است اما وقتی درجه را به ۵ تغییر دادیم، دقت حتی کمتر هم شد و به ۵۵٪ کاهش یافت.

۵.

الف) در ابتدا با تعداد *bin* های ۴تایی کار را آغاز کردیم و در هر بازه از *bin* ها میانگین را قرار میدهیم و یک فیچر جدید تحت عنوان *average_battery* درست میکنیم و هر متغیری مربوط به هر بازه ای بود میانگینش را در آن قرار میدهیم. در ابتدا با تعداد *bin* ها با سایز برابر و ۴تا اجرا کردیم، سپس ۶تا و ۸تا. یک

بار هم سایز *bin* ها را نامساوی در نظر گرفتیم و با ۶ تا *bin* به سایز های نامساوی کار کردیم. نتایج پیاده سازی مدل *SVM* بر روی هر یک از آنها را در سوال ۶ بررسی میکنیم.

ب) در دیتاست داده شده، هیچکدام از داده ها *categorical* نیستند و به همین دلیل از *one hot encoding* استفاده نکردیم. اما با توجه به مشاهدات قبلی میدانیم که این کار برای تبدیل داده های *categorical* به داده های عددی لازم است. زیرا وقتی میخواهیم مدل را *fit* کنیم نیاز است تا داده های ورودی *numerical* باشند، به همین دلیل از این روش استفاده میکنیم.

ج) یکی از تبدیلات مهمی که در مهندسی داده ها از آن استفاده میشود همین *log transform* است این تبدیل باعث میشود تا داده های پرت را بتوانیم بهتر هندل کنیم و درواقع پس از این تبدیل، توزیع داده ها به سمت توزیع نرمال نزدیک تر میشود و به همین دلیل نتیجه بهتری خواهد داد. البته در این دیتاست از آنجایی که خیلی واریانس داده ها زیاد نیست، این تبدیل خیلی کمکی به ما نمیکند و نتیجه آن توسط مدل *SVM* مشخص است. پس از تبدیل لگاریتمی، دقت ما کاهش یافته است. برای آنکه داده های منفی برای ما مشکلی ایجاد نکنند، این داده ها را از مینیمم داده های موجود در هر ستون کم کردیم و با ۱ جمع کردیم تا تمامی داده های ما مثبت باشند و بتوانیم لگاریتم آنها را حساب کنیم.

د) متغیری تحت عنوان حجم با *width* و *depth* و *height* میسازیم و ستون های مربوط به این ۳ فیچر را از دیتاست پاک میکنیم و به جای آن حجم را قرار میدهیم. پس از اسکیل کردن و جداکردن داده های آموزشی و تست دقت را با *SVM* مشاهده میکنیم که کاهش یافته است (بخش مربوط به *SVM* را در سوال ۶ توضیح میدهیم)

۶. در کد برای سوال ۵ هر بخش را به صورت جداگانه قرار دادهایم و پس از آن مدل *SVM* را برای هر کدام از آنها پیاده سازی کردیم. همانطور که میبینیم عمل *binning* دقت مدل را تقریبا ۸۶٪ نشان میدهد و در مقایسه با بقیه بخش ها، دقت بیشتری دارد. عمل *log transform* دقت مدل را کاهش داده و به ۷۷٪ رسیده. اضافه کردن فیچر حجم نیز کمی دقت را پایینتر آورده و تقریبا ۸۳٪ شده است.
عمل *binning* بسیار بهتر از حالت های دیگر بود و با ۴ *bin* دقت مدل تقریبا ۸۶٪ بود و وقتی تعداد *bin* ها را به ۶ اقزایش دادیم دقت مدل به ۸۹٪ رسید که این نشان میدهد هرچه تعداد *bin* ها بیشتر باشد عملکرد مدل بهتر خواهد بود. اما سایز های نامساوی عملکرد خوبی ندارند.

حال یک بار هم عمل *bin* با ۶ قسمت با سایز مساوی، اجرای *log transform* و داشتن فیچر حجم مدل *SVM* را بر روی تمامی حالات پیاده سازی کردیم. نتیجه دقت به ۷۵٪ کاهش یافت. دلیل میتواند این باشد که دو بخش از سوال قبل میزان دقت را کاهش دادند به خصوص *log transform* و بهمیندلیل استفاده از آنها با یکدیگر نتیجه را بهتر نکرد.

۷. الگوریتم های زیادی برای ساخت درخت تصمیم در کتابخانه های مختلف مورد استفاده قرار میگیرد. تفاوت اصلی این الگوریتم ها در معیار اندازه گیری*impurity* ، روش *splitting* و هرس کردن درخت می‌باشد.

- الگوریتم *ID3* :

این الگوریتم با استفاده از دو معیار *entropy* و *gain* درخت را میسازد. با محاسبه*gain* ، آن ویژگی هایی که اطلاعات بیشتری دارند را در سطح بالاتر درخت قرار میدهد. هر بار که یک ویژگی در سطحی از درخت انتخاب شد، زیر درخت‌های آن نیز دقیقا به همان صورت انتخاب می‌شوند و در سطوح و گره‌های بعدی قرار می‌گیرند. هرچه در درخت پایین‌تر می‌رویم (به برگ‌ها نزدیک‌تر می‌شویم)، مجموعه داده‌ها برای محاسبه‌ی مقدار اطلاعات کمتر می‌شوند. این الگوریتم برای مقادیر پیوسته ساخته نشده بود و تنها مقادیر گسسته را تشخیص میداد. یعنی فقط مقادیری که عددی نیستند را میتوانست تفکیک کند.

-الگوریتم *C4.5* :

این الگوریتم، نقص الگوریتم قبلی را رفع میکند. یعنی میتواند مقادیر گسسته یا پیوسته را در ویژگی ها درک کند. عملکرد آن مشابه *ID3* است یعنی بر اساس *gain* بیشتر درخت را میسازد. نکته مثبتی که وجود دادر آن است که این مدل با *missing value* ها میتواند کار کند و مشکلی ایجاد نمیشود. همچنین این الگوریتم عمل هرس کردن را نیز پس از ساخت درخت انجام میدهد و باعث جلوگیری در *overfitting* نیز میشود. این الگوریتم با وزن دادن به برخی ویژگی ها نیز تاثیر آنها را بیشتر میکند که بسیار مفید است.

-الگوریتم *CART* :

یکی از محبوب ترین الگوریتم های مورد استفاده برای ساخت درخت است. مخفف *classification and regression tree*است که بر اساس ساخت درخت های دودویی(باینری) بنا شده است.این الگوریتم داده ها را به دسته های دوتایی تقسیم میکند و بر اساس آنها درخت دودویی میسازد. در واقع برای اینکه درخت *CART* تشخیص دهد که کدام ویژگی‌ها می‌تواند اطلاعات بیشتری را ارائه دهد از شاخص جینی استفاده کرده و برای هر ویژگی هر چقدر شاخص جینی کمتر باشد، یعنی آن ویژگی اطلاعات بیشتری را به ما می‌دهد و می‌تواند در درخت ساخته شده، بالاتر (یعنی نزدیک به ریشه) قرار بگیرد.

الگوریتم های دیگری نیز به نام های *C5.0*، *QUEST*، *CHAID*، *MARS* و ... وجود دارد.

۸. با استفاده از پکیج *sklearn* و استفاده از *DecisionTreeClassifier* به ساخت یک درخت تصمیم بر روی دیتاست اصلی پرداختیم و نتیجه نهایی با دقت ۸۳٪ است

۹. با تغییر دادن پارامتر های مدل درخت تصمیم، نتیجه نهایی متفاوت خواهد بود. همانطور که در کد نیز مشاهده میکنیم، میزان عمق درخت و تعداد نمونه های موجود در هر گره را تغییر دادهایم و مشاهده میکنیم که دقت مدل تغییر میکند. هرچه عمق مدل بیشتر باشد، به مدل اولیه نزدیک تر خواهیم بود اما اگر این عمق را کم بگذاریم، میزان دقت کم میشود.
در مودر تعداد نمونه ها در هر گره، اگر زیاد افزایش دهیم، نتیجه بدتر خواهد شد و هرچه کمتر باشد بهتر است و از یه حدی نباید بیشتر باشد.

۱۰. زمانی که داده های ما خیلی زیاد باشند، شاخه های درخت بسیار زیاد میشوند و تقسیم بندی ها طولانی هستند. عمل هرس کردن دقیقا مقابل عمل تقسیم کردن است و با این عمل، زیر گره هایی از درخت تصمیم حذف میشوند (شاخه هایی که موجب ایجاد داده های پرت در داده آموزشی شدهاند) درخت های هرس شده تمایل به کوچکتر بودن و پیچیدگی کمتر دارند و بنابراین به راحتی قابل فهم میباشند. آنها معمولا در طبقهبندی صحیح دادههای تست سریعتر و بهتر از درخت های هرس نشده عمل می کنند. در برخی الگوریتم های ساخت درخت تصمیم، هرس کردن جزئی از مراحل آنها است، اما در بعضی دیگر اینگونه نیست و تنها برای جلوگیری از *overfitting* و از بین بردن داده های نویز انجام میشود.

۱۲. همانطور که در کد هم مشاهده میکنیم، نتیجه *random forest* بهتر است، زیرا این مدل به جای آنکه از یک درخت تصمیم استفاده کند، از چندین درخت باهم استفاده میکند اما اینکه در هر درخت چگونه ویژگی ها را انتخاب میکند کاملا رندم است ولی به دلیل استفاده از چندین درخت و *ensemble* کردن، نتیجه نهایی بهتر خواهد بود.

۱۳. تفاوت هایی بین الگوریتم درخت تصمیم و شبکه های عصبی وجود دارد. این تفاوت ها برخی باعث برتری درخت تصمیم میشود و به همین خاطر هست که همچنان محبوب است. در مورد سرعت، تفسیر پذیری و دقت آنها صحبت میکنیم:
الگوریتم درخت تصمیم در یادگیری، سریعتر از شبکه های عصبی عمل میکند. دلیل آن است که درخت تصمیم، درجا ویژگی های ورودی که به نظر لازم نیست را از بین میبرد و با تعداد ویژگی های کمتری مدل را آموزش

میدهد اما شبکه عصبی از تمامی ویژگی ها استفاده میکند مگر آنکه در مرحله پردازش داده ها، از عملیات *feature selection* استفاده کنیم.

درخت تصمیم تفسیر پذیری بالایی دارد، زیرا در هر مرحله کاملا مشخص است که از کدام ویژگی استفاده میکند و چگونه جلو میرود. عملکرد مدل دقیقا مشخص است اما در شبکه عصبی چنین چیزی نیست و این شبکه تفسیر پذیر نیست

درخت تصمیم برای داده های کم دقت بهتری نسبت به شبکه عصبی دارد، اما اگر حجم داده ها و بعد آن ها افزایش یابد، عملکرد درخت تصمیم ضعیف میشود. روش هایی مانند *bagging* و *boosting* هستند که موجب میشوند تا دقت مدل افزایش یابد اما اگر سرعت و تفسیر پذیری مدل مهم باشد الگوریتم درخت تصمیم مفیدتر است.

۱۴. روش های متفاوتی برای استخراج قوانین از دیتاست وجود دارد. برخی از آنها مبتنی بر رشد و هرس کردن هستند.

روش *RIPPER*: این روش بر اساس استراتژی تقسیم و غلبه کار میکند، بدین صورت که *Incremental Reduced Error Pruning* یا همان *IREP* را اجرا میکند تا قوانین را استخراج کند و با استفاده از این قوانین استخراج شده، دو قانون دیگر از آنها پدید می آورد: *replacement rule* و *revision rule* حال با استفاده از معیار دیگری مدل تصمیم میگیرد که از بین این سه دسته قوانین کدام را انتخاب کند.

روش *CAMUR*: اسم کامل روش *Classifier with Alternative and MUltiple Rule-based models* هست به معنای یک طبقه بند که با استفاده از مدل های جایگزین و متعدد بر اساس قانون عمل میکند. این روش از الگوریتم *RIPPER* نیز بهره میگیرد و بر اساس آن تولید شده است. این الگوریتم توسط یک مدل *classification* و بر اساس محاسبات تکراری، چندین قوانین معادل را استخراج میکند. *CAMUR* شامل یک پایگاه داده و همچنین ابزاری برای *querying* نیز هست.