به نام خدا

مهرانه مقتدائی فر – ۹۷۲۲۲۰۸۶

تاریخ : ۲۹ / ۲ /۱۴۰۱

گزارش تمرین سری ۲ **درس: داده کاوی** **استاد درس: دکتر فراهانی**

بخش اول

۱. الگوریتم forward selection را پیاده سازی کرده‌ایم. این الگوریتم ابتدا از یک مجموعه خالی شروع میکند و با توجه به معیار auc فیچر ها را به این مجموعه اضافه میکند و تا جایی پیش میرویم که مشاهده کنیم auc در حال افزایش است. از یک جا به بعد با اضافه کردن فیچرها، این معیار کاهش خواهد یافت و دیگر الگوریتم متوقف میشود.
پیش از اجرای الگوریتم، کلاس price_range را از ۴ کلاس به ۲ کلاس تبدیل کردیم. سپس الگوریتم FS را برروی داده ها اعمال میکنیم.
۴ فیچر انتخاب میشود و این فیچر ها بهترین هایی است که تاثیر بیشتری در مدل خواهد داشت. فیچر های ram، px_height، battery_power و px_width بهترین فیچرها هستند. در قسمت بعد خواسته شده که مدل logistic regression را با استفاده از این ۴ فیچر اعمال کنیم. دقت مدل 99% شده است.

۳. از آنجایی که در بخش قبل ۴ فیچر انتخاب شد، پس تعداد component های PCA را نیز ۴ در نظر میگیریم. سپس برروی این مولفه ها، logistic reg را پیاده میکنیم، دقت مدل به 99.25% خواهد رسید.

۵. 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:

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

۶. با استفاده از دیتاست اصلی، روش *SVM* را برروس آنها پیاده کردیم. دقت مدل برروی این داده های *89%* است.

۷. 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.

مهندسی ویژگی

۹.

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 بعدی 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 بعدی توضیح میدهیم)

۱۰. در کد برای سوال قبل 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 به همین دلیل استفاده از آنها با یکدیگر نتیجه را بهتر نکرد.

۱۱. 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.

۱۲. 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

۱۳. 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.

۱۴. 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.

۱۵. روش bootstrapping درواقع یکی از روش های resampling داده است .Bootstrap انجام نمونه گریی با جایگذاری از یک نمونه اصیل به دفعات زیاد است. از یک نمونه ثابت با حجم محدود به دفعات زیاد نمونه گریی انجام میدهیم تا در نهایت بتوان با استفاده از نتایج کلیه ی دفعات نمونه گریی، در مجموع به یک توزیع نمونه ای دست یافت. به این دلیل جایگذاری انجام میدهیم که آن چیزهایی که نمونه گیری شده اند پس از انتخاب دوباره به مجموعه داده ها بازگردند و شانس انتخاب شدن در نمونه گیری های دیگر را نیز داشته باشند. Cross Validation نیز مانند boostrapping یک روش resampling است اما یکی از تفاوت های آن این است که CV بدون جایگذاری انجام میشود همچنین با تمامی دیتاست سر و کار دارد اما bootstrapping لزوما به همه ویژگی ها کار ندارد و تا هر زمان که بخواهیم میتوانیم با استفاده از آن، resampling انجام دهیم. از bootstrapping در کارهای مختلفی استفاده میکنیم، مثلا برای ارزیابی مدل یا روش های تجمعی(ensemble) یا حتی تخمین زدن بایاس و واریانس مدل استفاده میشود.

۱۶. *5*2 fold cross validation* به این معناست که *2-fold* را برای ۵ مرتبه تکرار میکنیم. به این دلیل از *2fold* استفاده میشود که مطمئن باشند هر مشاهده در مجموعه داده *train* یا *test* فقط برای یک بار آزمودن مدل استفاده میشود.

تسک های امتیازی:

۱. روش *Backward Selection* مشابه روش *Forward* که در سوال اول بخش قبل انجام دادیم است. عملی مشابه آن اما برعکس آن انجام میدهد. بدین صورت که از مجموعه تمامی فیچر ها شروع میکند و آنهایی که تاثیر کمتری دارند و باعث کاهش معیار *AUC* میشود را حذف میکند. با پیاده سازی این روش و اعمال آن برروی دیتاست برخی از فیچر ها انتخاب میشود. همچنین *Logistic Reg* را نیز برروی این فیچرها اعمال میکنیم. دقت مدل برروی داده های تست *92%* است.

۳- معیار *MCC* یکی از معیار های سنجش درستی مدل است. مانند معیار های *precision, recall* و ... که برای مدل های طبقه بند استفاده میکنیم. این معیار ها کم و بیش معایبی دارند که ممکن است دقت مدل را به خوبی به ما ندهند، به خصوص برای مدل های ۲ کلاسه. معیار دیگری برای *binary classification* همین معیار است. فرمول آن به صورت زیر است:

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$

۳ مقدار مختلف نیز به عنوان خروجی میتواند بدهد. مقدار ۱ یعنی طبقه بندی به درستی انجام شده است.($FP=FN=0$) مقدار -1 یعنی کاملا اشتباه کلاس بندی صورت گرفته و همیشه *misclassify* داریم.($TP=TN=0$) اگر مقدار 0 دهد یعنی طبقه بند ما به صورت کاملا رندم کار میکند.

بخش دوم

۱.

قاعده بیز به ما بیان میکند که چگونه یک احتمال شرطی را با اطلاعاتی که از قبل داریم محاسبه کنیم. از این فرمول برای تعیین احتمال شرطی رویدادها استفاده می شود. اساساً، قضیه بیز، احتمال وقوع یک رویداد را بر اساس دانش قبلی از شرایطی که ممکن است مربوط به رویداد باشد، توصیف می کند. فرمول کلی این قضیه به صورت زیر است:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

با محاسبه احتمال پیشین، احتمال پسین و *Likelihood* و با توجه به احتمال وقوع داده های پیشین، میتواند احتمال پسین را پیشبینی کند و در فرایند دسته بندی داده با توجه به احتمال کم یا زیاد وقوع آنها به ما کمک میکند.

دسته بند های *Naïve Bayes*:

۱. *Gaussian Naïve Bayes*

اگر مشاهدات و داده‌ها از نوع پیوسته باشند، از مدل احتمالی با توزیع گاوسی یا نرمال برای متغیرهای مربوط به شواهد می‌توانید استفاده کنید. در این حالت هر دسته یا گروه دارای توزیع گاوسی است. به این ترتیب اگر k دسته یا کلاس داشته باشیم می‌توانیم برای هر دسته میانگین و واریانس را محاسبه کرده و پارامترهای توزیع نرمال را برای آن‌ها برآورد کنیم. پس توزیع x در هر دسته نرمال فرض میشود و به همین علت احتمال وقوع x را میتوانیم به صورت زیر در نظر بگیریم:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

۲. *Multinomial Naïve Bayes*

این نوع دسته بند بیشتر برای متن ها و نوشته های طولانی و در بحث *NLP* کاربرد دارد. در این حالت برداری از n ویژگی داریم که از توزیع چند جمله‌ای پیروی میکنند و هر *feature* دارای احتمالات مختلف است. بر اساس شمارش کلمات و همچنین استفاده از توزیع *multinomial*؛ احتمال وقوع کلمات بعدی را پیشبینی میکند و در بحث هایی که داده های متنی داریم بسیار کاربرد دارد.

۳. *Bernoulli Naïve Bayes*

این نوع از دسته‌بند بیز بیشترین کاربرد را در دسته‌بندی متن‌های کوتاه داشته، به همین دلیل محبوبیت بیشتری نیز دارد. در این مدل در حالت چند متغیره، فرض بر این است که وجود یا ناموجود بودن یک ویژگی در نظر گرفته شود. در واقع مقادیر فیچر ها در این نوع دسته بند تنها حالت 0,1 یا باینری دارد. در آن از توزیع برنولی استفاده میشود و بیشتر برای مقادیر گسسته کاربرد دارد. از برنولی بیشتر برای پیدا کردن وجود یا عدم وجود یک کلمه در متن استفاده میشود ، برخلاف توزیع چندجمله‌ای که فرکانس وقوع یک کلمه را بررسی میکند.

همانطور که با این سه نوع دسته بند آشنا شدیم، متوجه شدیم که GNB در زمانی که داده ها به صورت پیوسته هستند و از توزیع نرمال پیروی میکنند بیشتر کاربرد دارد. MNB در زمانی که یک متن را میخواهیم دسته بندی کنیم به کار میرود و اینکه چقدر یک کلمه احتمال آمدن دارد را برای ما بیان میکند. BNB زمانی به کار میرود که فیچر ها حالت باینری داشته باشند و مقدار گسسته باشد.

پیاده سازی Gaussian Naïve Bayes

در بخش پیاده سازی این تمرین با استفاده از دیتاست داده شده باید به پیاده سازی GNB یک بار بدون پکیج و بار دیگه با استفاده از پکیج sklearn برپردازیم.

بخش بدون استفاده از پکیج، از منبع زیر گرفته شده است:

https://towardsdatascience.com/learning-by-implementing-gaussian-naive-bayes-3f0e3d2c01b2

همانطور که در کد نیز مشخص است، میانگین داده ها و انحراف معیار آنها مشخص میشود و سپس در تابع predict_proba با توجه به توزیع گاوسی(نرمال) احتمال وقوع هر داده بررسی میشود. تابع GNB که بدون استفاده از پکیج زدیم را برروی داده های آموزشی، آموزش داده و سپس برروس داده های تست آزمایش کردیم. نتایج زیر را داریم:

```
confussion matrix
[[20  9]
 [ 6 26]]
-----------------------------------------
Accuracy of Logistic Regression: 75.40983606557377

-----------------------------------------
              precision    recall  f1-score   support

           0       0.77      0.69      0.73        29
           1       0.74      0.81      0.78        32

    accuracy                           0.75        61
   macro avg       0.76      0.75      0.75        61
weighted avg       0.76      0.75      0.75        61
```

پس از پیاده سازی همین داده ها با استفاده از پکیج sklearn نیز به نتایج زیر رسیدیم:

```
confussion matrix
[[20  9]
 [ 6 26]]
-----------------------------------------
Accuracy of Logistic Regression: 75.40983606557377

-----------------------------------------
              precision    recall  f1-score   support

           0       0.77      0.69      0.73        29
           1       0.74      0.81      0.78        32

    accuracy                           0.75        61
   macro avg       0.76      0.75      0.75        61
weighted avg       0.76      0.75      0.75        61
```