یادگیری ماشین برای بیوانفورماتیک

تمرین دوم، بخش تئوری

جواد راضی (۴۰۱۲۰۴۳۵۴)

SVM For Classification

۱.۱

اگر دیتاپوینت حذف شده یک ساپورت وکتور باشد، آنگاه مرز تصمیم جابجا می‌شود. در صورتی که دیتاپوینت ساپورت وکتور نباشد، مرز تصمیم جابجا نخواهد شد. نزدیک‌ترین پوینت‌ها به مرز تصمیم‌گیری، ساپورت وکتور محسوب می‌شوند.

در رگرسیون لاجیستیک، حذف یک دیتاپوینت **منجر به جابجایی مرز تصمیم می‌شود.** چرا که در این متد، مرز تصمیم بر اساس تمام دیتاپوینت‌های موجود انتخاب می‌شود. البته میزان جابجایی بسته به میزان تاثیر آن دیتاپوینت خاص در مرز تصمیم دارد؛ اگر نقطه حذف شده نزدیک مرز باشد، میزان جابجایی بزرگ‌تر خواهد بود.

۱.۲

مقدار هر یک از متغیرهای slack به شکل زیر محاسبه می‌شود:

$$\xi_i = \max(1 - y_i(w^T * x_i), 0)$$

در صورتی که $y_i(w^T * x_i) < 1$ باشد، نقطه مربوطه در کران بالای اینستنس‌هایی که درست طبقه‌بندی نشده‌اند قرار گیرد. بنابراین، $\xi_i > 0$ این را نتیجه می‌دهد که اینستنس ممکن است درست طبقه‌بندی نشده باشد. در نتیجه، کران بالای اینستنس‌هایی که درست طبقه‌بندی نشده‌اند، **برابرست با تعداد متغیرهای slack (ξ_i) با مقدار مثبت.**

۱.۳

ضریب C، هایپرپارامتری است که در واقع تریدآف میان کمینه‌کردن خطای طبقه‌بندی، و بیشینه کردن فاصله از نقاط حاشیه‌ای را کنترل می‌کند.

در حالی که C نزدیک صفر است، الگوریتم به Soft-margin SVM تبدیل می‌شود. در این شرایط، هدف بیشینه کردن مارجین است. در این حالت، اجازه می‌دهیم برخی از اینستنس‌ها نادرست طبقه‌بندی شوند و اولویت، بیشینه‌کردن مارجین است. برای همین، الگوریتم در این حالت نسبت به outlierها کم‌تر حساس است.

در شرایطی که C بسیار بزرگ باشد، الگوریتم به Hard-margin SVM تبدیل می‌شود. در این حالت، مدل به هنگام ترین‌شدن محافظه‌کارانه عمل می‌کند و سعی می‌کند تمام اینستنس‌ها را درست طبقه‌بندی کند، حتی اگر این به معنای کوچک‌تر شدن مارجین باشد. در چنین شرایطی حساسیت به outlierها بیشتر می‌شود، چرا که پنالتی طبقه‌بندی نادرست بسیار بالاست و outlierها وزن بالایی در تغییر وزن‌ها دارند.

۱.۴

در حالتی که کلاس‌ها به صورت خطی قابل تمییزند، هر دو الگوریتم Hard SVM و Logistic Regression یک مرز تصمیم‌گیری تولید خواهند کرد. مرز تصمیم‌گیری Hard SVM، چون با هدف کمینه‌کردن خطای طبقه‌بندی دیتای ترین انتخاب شده، به outlierها حساس خواهد بود و ممکن است مدل overfit شود.

مرز تصمیم Hard SVM، یک مرز خطی خواهد بود که ممکن است به یادگیری یک مدل overfitشده منجر شود. مرز تصمیم Logistic Regression اما غیرخطی بوده و به توزیع دیتاپوینت‌ها بستگی خواهد داشت.

۱.۵

این بار نیز هر دو الگوریتم مرزی برای تمییز دو کلاس ایجاد خواهند کرد. اما در این حالت، Soft SVM هدف اصلی‌اش بیشینه کردن مارجین (فاصله مرز با دیتاپوینت‌ها) می‌باشد و در صورت نویزی بودن دیتا و وجود outlierها، این الگوریتم با اجازه دادن تعدادی طبقه‌بندی نادرست هنگام ترین‌شدن مدل، انعطاف بیشتری به خرج داده و بهتر از Hard SVM عمل خواهد کرد.

Composing Kernel Functions

۲.۱

$$K^{(1)}\ positive\ semi-definite \rightarrow\ \forall z\ z^T K^{(1)} z \geq 0$$

$$z^T K^{(1)} z \geq 0, c \geq 0 \rightarrow c\left(z^T K^{(1)} z\right) \geq 0 \rightarrow z^T\left(cK^{(1)}\right) z \geq 0$$
$$\rightarrow cK^{(1)} is\ positive\ semi-definite$$

۲.۲

$$z^T K z = z^T\left(K^{(1)} + K^{(2)}\right) z = z^T K^{(1)} z + z^T K^{(2)} z$$

$$Given: \left\{z^T K^{(1)} z \geq 0\ and\ z^T K^{(2)} z \geq 0\right\} \rightarrow\ z^T K^{(1)} z + z^T K^{(2)} z \geq 0 \rightarrow\ z^T K z \geq 0$$

۲.۳

از قضیه Mercer می‌دانیم که اگر K یک کرنل صحیح باشد، یک ضرب داخلی به صورت $K^{(1)}(x, x') = \phi^{(1)}(x)^T \phi^{(1)}(x')$ را بیان می‌کند. اکنون برای ضرب دو کرنل داریم:

$$K^{(1)} K^{(2)}(x, x') = \phi^{(1)}(x)^T \phi^{(1)}(x')\ \phi^{(2)}(x)^T \phi^{(2)}(x')$$
$$= \left(\sum_{i=1}^{n} \left(\phi_i^{(1)}(x)^T \phi_i^{(1)}(x')\right)\right)\left(\sum_{j=1}^{n} \phi_j^{(2)}(x)^T \phi_j^{(2)}(x')\ \right)$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} \left(\ \phi_i^{(1)}(x)^T \phi_j^{(2)}(x)\ \right)\left(\phi_i^{(1)}(x')^T \phi_i^{(2)}(x')\ \right)$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} \phi_{ij}^{(1,2)}(x) \phi_{ij}^{(1,2)}(x') = \phi^{(1,2)}(x)^T \phi^{(1,2)}(x')$$

بنابراین، ضرب دو کرنل را به صورت ضرب دو basis جدید نوشتیم و مطابق قضیه Mercer، ماتریس این کرنل نیز positive, semi-definite بوده و کرنل صحیح است.

۲.۴

۲.۵

K-Fold Cross Validation

۳.۱

در این پروسه، دیتا به صورت رندم به K گروه تقسیم می‌شود. بین گروه‌ها، جابجا می‌شویم و هربار یک گروه به عنوان دیتای تست (یا ولیدیشن) انتخاب شده و باقی گروه‌ها، دیتای ترین خواهند بود. با این تقسیم‌بندی، مدل ترین شده و عملکرد آن ارزیابی می‌گردد. در ایتریشن بعدی، مدل و وزن‌های آن باید ریست شوند و ترینینگ با دیتاست ترین و تست جدید، مجددا از صفر شروع شود. در نهایت، هر یک از K گروه، یک بار به عنوان دیتای تست، و K -1 بار به عنوان دیتای ترین استفاده خواهند شد.

۳.۲

(a

نسبت به Validation Set:

مزایا:

- به علت اینکه دیتا به چند گروه تقسیم‌شده و چند بار مدل ترین شده و عملکرد مدل‌ها میانگین گرفته می‌شود، این روش واریانس کمتری دارد و مخصوصا در دیتاست‌های کوچک، اثر تقسیم نامتعادل شانسی کمتر است.
- چون تمام دیتاست هم برای ترین، و هم برای Validation استفاده می‌شود، مدل ترین‌شده توسط این روش معمولا عملکرد بهتری خواهد داشت.

معایب:

- این روش نسبت به Validation Set، پیچیدگی محاسباتی بیشتری دارد. چرا که مدل K بار ترین خواهد شد. بر خلاف Validation Set که فقط یکبار داده‌ها تقسیم و مدل ترین خواهند شد.

(b

نسبت به Leave One Out Cross Validation:

مزایا:

- پیچیدگی محاسباتی کم‌تری نسبت به LOOCV دارد. چرا که در روش دومی، هر دیتاپوینت یکبار کنار گذاشته‌شده و به عنوان دیتای Validation استفاده می‌شود. پس مدل N بار (به تعداد دیتاپوینت‌ها) ترین می‌شود. این کار، مخصوصا در دیتاست‌های بزرگ، تفاوت بزرگی در زمان محاسباتی ایجاد می‌کند.

معایب:

- برای دیتاست‌های کوچک، مدل K-Fold نیز می‌تواند واریانس زیادی داشته باشد و به اندازه LOOCV تخمین‌مان از عملکرد مدل مناسب نباشد.

۳.۳

MCCV یک متد دیگر برای ارزیابی عملکرد مدل بر روی دیتای دیده‌نشده است. این متد نیز همانند K-Fold، برای زمان‌هایی که اندازه دیتاست محدود است کاربردیست. در این متد، دیتا چندین بار، به دو سابست ترین و تست تقسیم می‌شود. اما در هر تقسیم، هر سابست ترین یا تست، ممکن است با سابست تقسیم بندی اشتراک داشته باشد. به عبارت دیگر، یک دیتاپوینت می‌تواند چندین بار به عنوان دیتای ترین، و دیتای تست استفاده شود.

۳.۴

همان‌طور که گفته شد، تفاوت اساسی این متد با K-Fold این است که در K-Fold، دیتاست به K سابست که با یکدیگر Mutually Exclusive هستند تقسیم می‌شود. بنابراین، هر دیتاپوینت، دقیقا یکبار به عنوان دیتای تست، و K – 1 بار به عنوان دیتای ترین استفاده می‌شود. اما در MCCV، سابست‌ها لزومی ندارد که Mutually Exclusive باشند. بنابراین، یک دیتاپوینت ممکن است هیچ بار به عنوان دیتای ترین یا تست استفاده نشود، یا بالعکس، چندین بار به عنوان دیتای تست استفاده گردد.

به طور کلی، در انتخاب میان K-Fold و MCCV، در تریدآف میان بایاس و واریانس، انتخاب می‌کنیم که کدام را ترجیح می‌دهیم. متد K-Fold Cross Validation، به علت اینکه هر دیتاپوینت در آن دقیقا یکبار تست شده‌است، منجر به بایاس کمتری می‌شود. از سوی دیگر، MCCV،

تقسیم‌بندی‌های متنوع‌تری از دیتا را استفاده می‌کند که این کار می‌تواند به واریانس کم‌تر منجر شود و اثر تقسیم‌بندی شانسی کاهش یابد.

Hyperparameter Optimization

۴.۱

از هایپرپارامترهایی نظیر نرخ یادگیری، تعداد لایه‌های شبکه عصبی، تابع فعال‌سازی استفاده شده در لایه‌ها، تابع کرنل استفاده شده در SVM، مقدار C در SVM، تعداد تخمین‌گر های Random Forest استفاده شده‌است.

تعیین مقدار نامناسب برای این پارامترها، می‌تواند منجر به عملکرد ضعیف مدل در طبقه‌بندی یا پیش‌بینی شود. به طور خاص، هایپرپارامترهای نادرست می‌توانند منجر شوند مدل Overfit یا Underfit شود.

۴.۲

پارامترهای مدل، نظیر وزن‌های شبکه عصبی، هنگام فرایند ترین‌شدن آن یادگرفته می‌شوند. این پارامترها، به گونه‌ای یاد گرفته می‌شوند که تابع Loss، کمینه شود. هایپرپارامترهای مدل، قبل از فرایند یادگیری، به مدل داده می‌شوند. در فرایند Hyperparameter Optimization، بهترین مجموعه مقادیر هایپرپارامترها که منجر به بهترین عملکرد مدل روی دیتای Validation می‌گردد، یافت می‌شود.

۴.۳

یک راه برای اعمال دانش خودمان در خصوص هایپرپارامترها، استفاده از متد Bayesian Optimization است. در این متد، دانش خود در مورد مدل و هایپرپارامترهای آن را به صورت توزیع‌های احتمالی پیشین (Prior) اعمال کرده و این توزیع را به صورت Iterative، بر اساس عملکرد مدل بروزرسانی می‌کنیم.

راه دیگر، استفاده از دانش خود برای تخمین کران‌های بالا و پایین هایپرپارامترها، یا مقادیری که می‌توانند معقول باشند است. در واقع با این کار، فضای ترکیبات هایپرپارامترهای ممکن را با

استفاده از دانش خود narrow-down می‌کنیم. با کوچک‌شدن این فضا، می‌توان از روش‌هایی نظیر Grid Search استفاده کرد تا بهترین ترکیب از هایپرپارامترها بدست آید.

Decision Tree

۵.۱

می‌دانیم که برای هر فیچر، آنتروپی با فرمول زیر بدست می‌آید:

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

در گام نخست، باید آنتروپی تمام دیتاست یافت شود.

$$H(HeartAttack) = -p_{H=True} \log_2 P_{H=True} - p_{H=False} \log_2 P_{H=False}$$
$$= 0.67 * 0.58 + 0.33 * 1.58 = 0.92$$

در مرحله بعد، باید میزان Information Gain برای هر فیچر پیدا شود. این کار، با بدست آوردن میانگین وزن‌دار تفریق آنتروپی دیتاست برای یک مقدار معین فیچر، از آنتروپی گره پدر بدست می‌آید. در ادامه برای هر فیچر Information Gain یافت می‌شود:

ChestPain:

$$ChestPain = Yes => \{HeartAttack = yes: 1.0, HeartAttack = no: 0.0\}$$
$$ChestPain = No => \{HeartAttack = yes: 0.33, HeartAttack = no: 0.66\}$$

$$H(ChestPain = Yes) = -1.0 \log_2 1 - 0.0 \log_2 0.0 = 0$$
$$H(ChestPain = No) = 0.33 * 1.58 + 0.67 * 0.58 = 0.92$$
$$IG(ChestPain) = 0.5 * IG(ChestPain = Yes) + 0.5 * IG(ChestPain = No)$$
$$= 0.5 * (0.90 - 0.0) + 0.5 * (0.92 - 0.92) = 0.46$$

Male:

$$H(Male = yes) = 0.75 * 0.41 + 0.25 * 2 = 0.80$$
$$H(Male = no) = 1.0 * 0 + 0 = 0$$

$$IG(Male) = 0.67 * IG(Male = yes) + 0.33 * IG(Male = no)$$
$$= 0.67 * (0.92 - 0.8) + 0.33 * (0.92 - 0) = 0.384$$

Smokes:

$$H(Smokes = yes) = 0.75 * 0.41 + 0.25 * 2 = 0.80$$
$$H(Smokes = no) = 0.5 * 1 + 0.5 * 1 = 1$$
$$IG(Smokes) = 0.67 * IG(Smokes = yes) + 0.33 * IG(Smokes = no)$$
$$= 0.67 * (0.92 - 0.8) + 0.33 * (0.92 - 1.0) = 0.054$$

Exercises:

$$H(Exercises = yes) = 0.5 * 1.0 + 0.5 * 1.0 = 1$$
$$H(Exercises = no) = 1.0 * 0 + 0 = 0$$
$$IG(Exercises) = 0.67 * IG(Exercises = yes) + 0.33 * IG(Exercises = no)$$
$$= 0.67 * (0.92 - 1.0) + 0.33 * (0.92 - 0) = 0.25$$

بنابراین، مطابق محاسبات، بیشترین Information Gain را فیلد Chest Pain دارد و ریشه درخت، این فیچر خواهد بود.

اکنون برای سطح دوم، همین پروسه را تکرار می‌کنیم. برای کسانی که درد سینه دارند، مستقیما مدل سکته را پیش‌بینی خواهد کرد، چرا که در این حالت تمام دیتاپوینت‌ها یکسانند و مربوط به کسانی‌اند که سکته کرده‌اند.

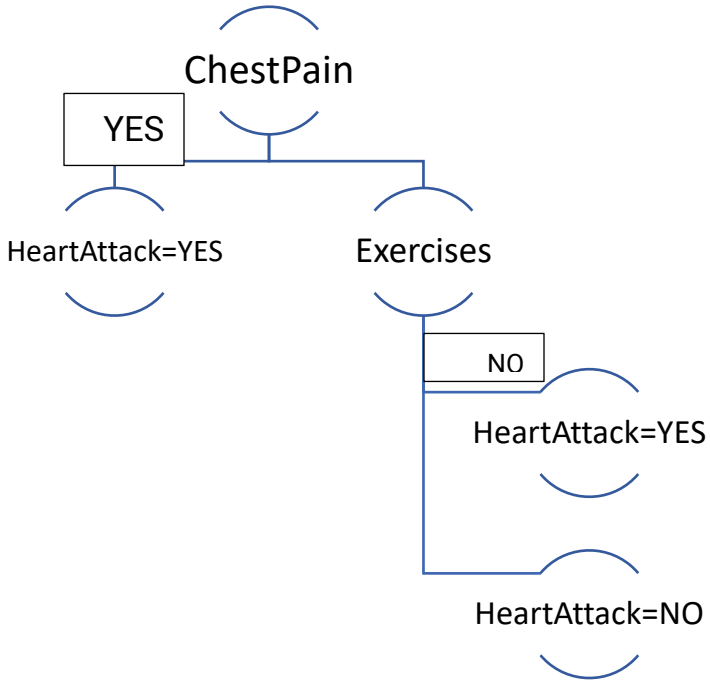
برای ۳ نفری که درد قفسه سینه ندارند، جدولی به شکل زیر خواهیم داشت:

Patient ID	Male	Smokes	Exercises	Heart Attack
3	No	Yes	No	Yes
4	Yes	No	Yes	No
6	Yes	Yes	Yes	No

در اینجا نیاز به محاسبه جزئی برای هریک از فیچرها نیست و با یک نگاه سرانگشتی نیز می‌توان تشخیص داد کدام فیچر بیشترین Information Gain را دارد؛ فیچر Male، به طور کامل سکته قلبی را پیش‌بینی می‌کند. فیچر Smoke، در شرایطی که فرد سیگار می‌کشد نمی‌تواند به تنهایی پیش‌بینی انجام دهد. فیچر Exercises اما مشابه فیچر Male، به تنهایی سکته قلبی را پیش‌بینی

می‌کند. بنابراین هریک از فیچرهای Male یا Exercises، Information Gain یکسانی دارند و می‌توانند انتخاب شوند. در اینجا ما فیچر Exercises را انتخاب می‌کنیم.

بنابراین درخت تصمیم نهایی به شکل زیر است:



۵.۲

گره مربوط به فیچر قفسه سینه، ریشه درخت است، و کسی که درد قفسه سینه دارد، مطابق درخت بالا، مستقیما سکته قلبی برایش پیش‌بینی خواهد شد.

AdaBoost Algorithm

۶.۱

فرض کنیم که توابعی که در مرحله t و t+1 انتخاب شده‌اند یکسانند. در این صورت، طبعا وزن اینستنس‌ها نیز یکسان است. این در حالی است که در مرحله t+1، الگوریتم AdaBoost، دیتا را بر اساس خطای طبقه‌بندی در مرحله t وزن‌دهی می‌کند. بنابراین، اینستنس‌هایی که در مرحله t

نادرست طبقه‌بندی شده‌اند، در مرحله t+1 باید وزن بالاتری داشته باشند. بنابراین، الگوریتم AdaBoost، هیچ‌گاه دو تابع یکسان را در دو مرحله متوالی انتخاب نمی‌کند.

۶.۲

$$< D_{t+1}(1), \dots, D_{t+1}(m)\ , y_i h_t(x_i) > \\ = \left(D_{t+1}(1) * y_1 h_t(x_1) \right) + \left(D_{t+1}(2) * y_2 h_t(x_2) \right) + \dots \\ + \left(D_{t+1}(m) * y_m h_t(x_m) \right)$$

از آنجایی که الگوریتم AdaBoost وزن‌های اینستنس‌هایی که نادرست طبقه‌بندی شده‌اند را آپدیت می‌کند، $D_{t+1}(k)$ فقط برای اینستنس‌هایی که نادرست طبقه‌بندی شده‌اند غیرصفر خواهد بود. از سوی دیگر $y_k h_t(x_k)$ فقط برای اینستنس‌هایی که درست طبقه‌بندی شده‌اند غیر صفر است. بنابراین، همواره یکی از کامپوننت‌های این دو وکتور صفر بوده، و ضرب داخلی این دو صفر خواهد شد.