دانشگاه صدا و سیمای جمهوری اسلامی ایران

نام و نام خانوادگی:

فاطمه رمضانی

استاد درس:

آقای دکتر راهیل مهدیان

درس:

یادگیری عمیق

شماره دانشجویی:

۴۰۲۱۵۴۱۵۰۴

موضوع مقاله:

On Deep Neural Networks for Detecting Heart Disease

شهریور ۱۴۰۳

❖ چکیده مقاله

بیمارهای قلبی علت اصلی مرگ و میر است و کارشناسان تخمین می‌زنند که تقریباً نیمی از تمامی حملات قلبی و سکته‌ها در افرادی رخ می‌دهد که در معرض خطر شناخته نشده‌اند. بنابراین، نیاز فوری به بهبود دقت تشخیص بیماری های قلبی وجود دارد. به همین منظور، ما به بررسی پتانسیل استفاده از تحلیل داده و به خصوص طراحی و استفاده از شبکه‌های عصبی عمیق برای تشخیص بیماری قلبی بر اساس داده‌های معمول بالینی می‌پردازیم.

مهم‌ترین دستاورد ما، طراحی ارزیابی و بهینه‌سازی معماری‌های شبکه های عصبی عمیق با عمق‌های افزایشی برای تشخیص بیماری قلبی است. این کار به کشف یک معماری جدید پنج لایه به نام *-ارزیابی قلب برای کاهش ریسک الگوریتمی و بهینه‌سازی پنج (HEARO-5)* - منجر شد که بهترین دقت پیش‌بینی را ارائه می‌دهد. طراحی HEARO-5 از regularization optimization استفاده می‌کند و به طور خودکار با داده‌های گمشده و/یا داده‌های پرت برخورد می‌کند.

برای ارزیابی و تنظیم معماری‌ها از اعتبارسنجی k-way cross-validation و همچنین Matthews correlation coefficient برای اندازه‌گیری کیفیت طبقه‌بندی‌های ما استفاده می‌کنیم. این مطالعه بر روی مجموعه داده‌های عمومی اطلاعات پزشکی Cleveland انجام شده است و ما توسعه‌های خود را به صورت متن‌باز در دسترس قرار می‌دهیم تا بیشتر به باز بودن و تحقیقات در زمینه استفاده از DNN در پزشکی کمک کنیم. معماری HEARO-5 با ارائه دقت ۹۹٪ و MCC برابر با ۰.۹۸، به طور قابل توجهی از تحقیقات منتشر شده فعلی در این زمینه پیشی می‌گیرد.

❖ مقدمه

بیماری قلبی علت اصلی مرگ ومیر در سراسر جهان است و سالانه بیست میلیون نفر را به کام مرگ می‌کشاند. تشخیص دقیق و زودهنگام می‌تواند تفاوت بین زندگی و مرگ برای افراد مبتلا به بیماری قلبی باشد. با این حال، پزشکان تقریباً یک سوم از بیماران را به اشتباه به عنوان غیرمبتلا به بیماری قلبی تشخیص می‌دهند و این باعث می‌شود این بیماران از درمان‌های نجات‌دهنده ممکن محروم شوند. این موضوع به نگرانی فزاینده‌ای تبدیل شده است، زیرا تعداد آمریکایی‌های مبتلا به نارسایی قلبی تا سال ۲۰۳۰ به میزان ۴۶ درصد افزایش خواهد یافت .تشخیص بیماری قلبی برای هر پزشکی چالش‌برانگیز است، زیرا در حالی که درد قفسه سینه و خستگی علائم شایع آترواسکلروز هستند، تا ۵۰ درصد از افراد هیچ علائمی از بیماری قلبی تا اولین حمله قلبی خود ندارند.

کشف نشانگرهای زیستی (برای بیماری قلبی) - شاخص‌های قابل اندازه‌گیری شدت یا حضور برخی از بیماری‌ها - مورد ترجیح است، اما در بسیاری از موارد نشانگرهای واضحی وجود ندارد و ممکن است نیاز به انجام و تحلیل چندین آزمایش باشد. بیشتر پزشکان از دستورالعمل‌های توصیه شده توسط انجمن قلب آمریکا استفاده می‌کنند که هشت عامل خطر شناخته شده مانند فشار خون بالا، کلسترول، سیگار کشیدن و دیابت را تست می‌کند. با این حال، این مدل ارزیابی خطر نقص دارد زیرا بر اساس فرض وجود رابطه خطی بین هر عامل خطر و نتیجه بیماری قلبی است، در حالی که این روابط پیچیده و با تعاملات غیرخطی هستند. ساده‌سازی بیش از حد، ممکن است باعث شود پزشکان در پیش‌بینی‌های خود اشتباه کنند یا عوامل مهمی را نادیده بگیرند که می‌تواند تعیین کند آیا بیمار درمان می‌شود یا خیر. پزشکان همچنین باید چگونگی تفسیر نتایج تشخیصی که بین بیماران متفاوت است و نیاز به تخصص زیاد دارد را بدانند.

استفاده از تکنیک‌های تحلیل داده‌های یادگیری ماشین می‌تواند نیاز به تخصص انسانی و امکان خطای انسانی را کاهش دهد و در عین حال دقت پیش‌بینی را افزایش دهد. الگوریتم‌های یادگیری ماشین مدل‌های پیش‌بینی انعطاف‌پذیری را بر اساس روابط یاد گرفته شده بین متغیرها در مجموعه داده ورودی اعمال می‌کنند. که می‌تواند از ساده‌سازی بیش از حد مدل‌های تشخیص ثابت مانند دستورالعمل‌های AHA جلوگیری کند. در واقع، الگوریتم شبکه عصبی با دقت ۷۶ درصد ثابت شده است که ۷.۶ درصد بیشتر از روش AHA به درستی رویدادها را پیش‌بینی می‌کند. در اینجا، ما به طور قابل توجهی بر نتایج بسیار امیدوارکننده یادگیری ماشین با طراحی و تنظیم معماری‌های شبکه عصبی عمیق با عمق‌های افزایشی برای تشخیص بیماری قلبی بر اساس داده‌های معمول بالینی بهبود می‌بخشیم. نشان می‌دهیم که یک طراحی انعطاف‌پذیر و تنظیم هایپرپارامترهای متعدد یک DNN می‌تواند تا ۹۹ درصد دقت داشته باشد. نتایج با استفاده از k-way cross-validation و همچنین Matthews correlation coefficient (MCC) برای اندازه‌گیری کیفیت طبقه‌بندی‌ها ارزیابی و اعتبارسنجی شد. بهترین نتایج بر روی یک معماری جدید پنج لایه DNN به نام- *Heart Evaluation for Algorithmic Risk-reduction and Optimization Five (HEARO-5)* - به دست آمد که بهینه‌سازی منظم‌سازی را به کار می‌گیرد و به طور خودکار با داده‌های گمشده و/یا داده‌های پرت برخورد می‌کند. دقت این شبکه برابر با ۹۹ درصد و MCC برابر با ۰.۹۸ است که به طور قابل توجهی از تحقیقات منتشر شده فعلی در این زمینه پیشی می‌گیرد و بیشتر جذابیت استفاده از تحلیل داده‌های ML در پزشکی تشخیصی را تأیید می‌کند.

❖ سوابق تحقیقات

تعدادی مقاله تحقیقاتی وجود دارند که از شبکه‌های عصبی مصنوعی برای بهبود تشخیص بیماری‌های قلبی استفاده می‌کنند. در یک مطالعه منتشر شده در مجله قلب و عروق، یو و همکاران نتیجه گرفتند که یک توپولوژی شبکه عصبی با دو لایه مخفی، یک مدل دقیق با دقت ۹۴ درصد بر روی داده‌های آزمایشی است. آنها در ساخت مدل خود برای طبقه‌بندی ویژگی‌ها قبل از تعیین تشخیص ممکن، بر تنوع عوامل خطر تمرکز کردند.

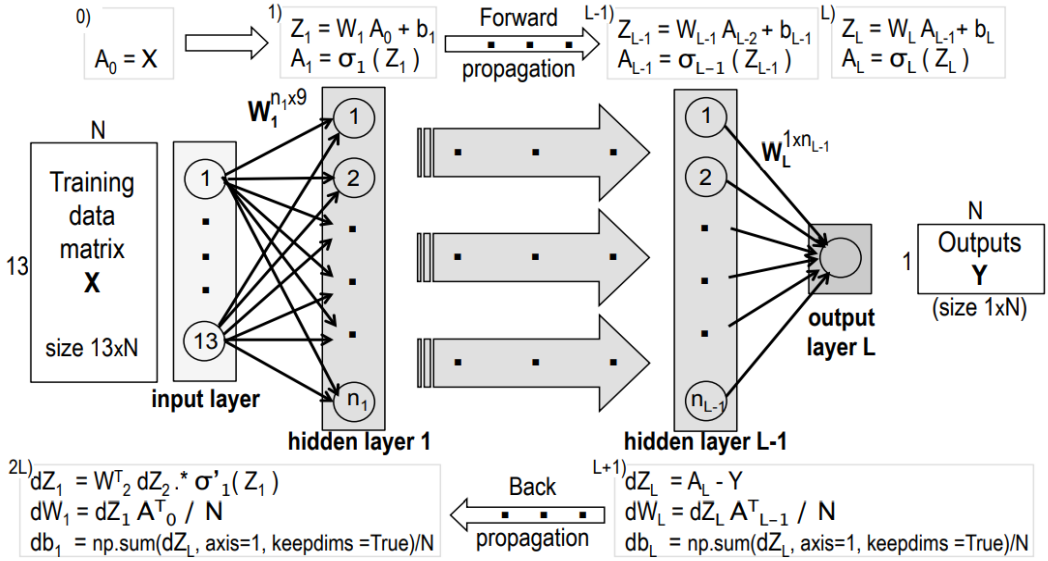
این مطالعه نتیجه‌گیری کرد که شبکه‌های عصبی روش موثری برای تحلیل موارد برای زمانی که ایجاد یک مدل ریاضی دقیق امکان‌پذیر نیست اما مجموعه‌ای نماینده از نمونه‌ها وجود دارد هستند. وینودهینی و همکاران با استفاده از این پژوهش، به طبقه‌بندی ویژگی‌ها با مدل‌های آماری مانند chi square پرداخته و سپس از شبکه عصبی به عنوان مدل پیش‌بینی استفاده کردند. این روش به طور کلی موفق بود، اما عملکرد ضعیف‌تری در مواجهه با ویژگی‌های اضافی نشان داد. مطالعه‌ای که در مجله پزشکی MHealth منتشر شده (لوه و همکاران) دقت شبکه‌های عصبی عمیق را با اثبات توانایی آنها در یادگیری از روابط غیرخطی در داده‌ها نشان می‌دهد. با این حال، آنها با مشکل بیش‌برازش مواجه شدند، زمانی که الگوریتم بیش از حد از داده‌های آموزشی یاد می‌گیرد و توانایی کمتری در اعمال خود بر داده‌های ناآشنا دارد(Over Fitting). پژوهشی منتشر شده در مجله مهندسی بهداشت و درمان به مشکل بیش‌برازش با رتبه‌بندی ویژگی‌ها، آموزش شبکه عصبی با هر رتبه‌بندی ویژگی، و سپس آموزش شبکه عصبی برای خروجی دادن یک تشخیص ممکن، کمک می‌کند .این روش کمک می‌کند تا شبکه از روابط مهم وزن‌دهی شده عددی در داده‌های آموزشی یاد بگیرد که می‌تواند به داده‌های ناآشنا نیز اعمال شود.

❖ پیشینه طراحی و پیاده سازی شبکه عصبی عمیق

➢ طراحی انعطاف‌پذیر شبکه عصبی عمیق

طراحی فریم‌ورک، نمادگذاری‌ها و مراحل محاسباتی اصلی که بررسی کرده‌ایم در شکل ۱ نشان داده شده است. همان‌طور که مشاهده می‌شود، شبکه عصبی به L لایه کاملاً متصل ($i = 1, ..., L$) با ni گره (یا نورون مصنوعی) در هر لایه سازماندهی شده است که با هم برای انجام prediction عمل می‌کنند. اتصالات بین لایه‌های i-1 و i توسط وزن‌های عددی نشان داده می‌شوند که در ماتریس Wi به اندازه $ni \times ni-1$ و بردار bi به طول ni ذخیره می‌شوند. بنابراین، اگر مقادیر ورودی برای لایه i که توسط مقادیر در گره‌های $ni-1$ لایه i-1 داده می‌شود، به‌صورت بردار $ai-1$ به اندازه $ni-1$ نشان داده شود، خروجی لایه i به‌صورت برداری به اندازه ni داده می‌شود که توسط ضرب ماتریس-بردار $Wiai-1 + bi$ محاسبه می‌شود.

از آنجا که آموزش به‌صورت موازی برای یک دسته از nb بردارها انجام می‌شود، ورودی‌های $ai-1$ به‌صورت ماتریس‌های $Ai-1$ به اندازه $ni-1 \times nb$ خواهند بود و خروجی‌ها توسط ضرب ماتریس-ماتریس $Zi = WiAi-1 + bi$ داده می‌شود، جایی که "+" بردار bi را به هر یک از ستون‌های nb ماتریس حاصل اضافه می‌کند.



شکل ۱: معماری DNN و مراحل محاسباتی اصلی برای آموزش آن پارامتریزه شده. داده‌های آموزشی X شامل ۱۳ ویژگی (داده‌های بالینی روتین برای هر بیمار؛ همچنین می‌توان پارامتریزه کرد) و N مثال آموزشی است. وزن‌ها W و بایاس‌ها b با استفاده از روش نزول گرادیان دسته‌ای تصادفی آموزش داده می‌شوند تا پیش‌بینی ها با نتایج داده شده "مطابقت" داشته باشند.

➢ اجزای اصلی سازنده شبکه‌های عصبی عمیق

فرآیند انتشار رو به جلو، که توسط مراحل ۰ تا L ارائه می‌شود، یک تابع فرضیه/پیش‌بینی غیرخطی HW,b(X) ≡ ALبرای ورودی‌های داده شده Xو وزن‌های ثابت Wو bرا نشان می‌دهد. وزن‌ها باید به گونه‌ای تغییر کنند که پیش‌بینی‌های HW,b(X)به نتایج داده شده/شناخته‌شده که در Yذخیره شده‌اند، نزدیک شوند. این به عنوان یک مسئله طبقه‌بندی شناخته می‌شود و یک مورد از یادگیری نظارت شده است. تغییر وزن‌ها به عنوان یک مسئله کمینه‌سازی روی یک تابع هزینه محدب J تعریف می‌شود، مثلاً:

$$\min_{W,b} J(W,b), \ \textbf{where} \ J(W,b) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log H_{W,b}(x_i) + (1-y_i)\log(1-H_{W,b}(x_i)).$$

این مسئله با استفاده از روش نزول گرادیان تصادفی دسته‌ای (batch stochastic gradient descent) حل می‌شود. این روش یک الگوریتم تکراری است که در هر مرحله یک دسته از nb نمونه‌های آموزشی را استفاده می‌کند. مشتقات تابع هزینه J نسبت به وزن‌ها W و b با استفاده از قاعده زنجیره‌ای برای مشتق‌گیری از ترکیبات توابع، از طریق لایه‌ها محاسبه می‌شوند. سپس این مشتقات از طریق مراحل انتشار رو به عقب L+1 تا L۲ محاسبه می‌شوند و برای تغییر وزن‌های مربوطه W_i و b_i در طی فرآیند تکراری آموزش برای هر لایه i استفاده می‌شوند به صورت:

$$W_i = W_i - \lambda dW_i, \quad b_i = b_i - \lambda db_i,$$

که در آن لاندا یک هایپرپارامتر است که به عنوان نرخ یادگیری شناخته می‌شود. توابع σ1, ..., σL توابع فعال‌سازی (که ممکن است برای لایه‌های مختلف شبکه متفاوت باشند) هستند و σ’ مشتقات آن‌ها هستند. ما از توابع فعال‌سازی ReLU، سیگموئید، tanh، و Leaky ReLU استفاده کرده‌ایم.

➢ بهینه‌سازی الگوریتمی: (Regularization)

Regularization یک تکنیک استاندارد است که با جریمه کردن مقادیر بزرگ وزن‌ها از (overfitting) جلوگیری می‌کند. شبکه‌های عصبی عمیق تمایل دارند به برخی از نقاط داده آموزشی مقادیر وزن بالاتری اختصاص دهند که این امر با واریانس بالا مطابقت دارد. تنظیم‌سازی به حل مشکل واریانس بالا در داده‌های آموزشی کمک می‌کند و می‌تواند دقت مدل روی داده‌های آزمایشی را بهبود بخشد. تنظیم‌سازی معمولاً با اضافه کردن یک جمله جریمه به فرم $\frac{\alpha}{2N}||W,b||^2$ به تابع هزینه J انجام می‌شود، جایی که ||W, b|| یک نُرم از وزن‌ها است، مانند L1 یا L2. پارامتر تنظیم‌سازی α جریمه‌ای برای وزن‌های بزرگ اعمال می‌کند، بنابراین اطمینان حاصل می‌شود که مدل بیش‌برازش نکند. یکی دیگر از مزایای Regularization این است که می‌تواند از یادگیری الگوریتم از داده‌های پرت (outliers) جلوگیری کند، که این امر برای مجموعه داده‌های کوچکی مانند مجموعه بیماران قلبی مورد استفاده در این تحقیق ضروری است. تنظیم‌سازی باعث می‌شود داده‌های پرت در مجموعه داده‌ها باقی بمانند، اما احتمال یادگیری الگوریتم از این مقادیر را کاهش می‌دهد. بنابراین، ما تنظیم‌سازی را به مدل خود اضافه می‌کنیم تا بهبودهای احتمالی در دقت را از طریق کاهش بیش‌برازش و به طور خودکار کاهش تأثیر هرگونه داده پرت بررسی کنیم.

➢ بهینه سازی هایپرپارامتر

یکی از چالش‌های کدنویسی یک شبکه عصبی، ساختاربندی آن به گونه‌ای است که هم دقیق و هم کارآمد باشد. باید تعیین کرد که از چند لایه استفاده شود، چند نود (نورون) در هر لایه به کار گرفته شود و غیره. این مسئله برای مثال در شبکه‌های عمیق کانولوشن برای تشخیص تصویر بسیار حیاتی بود، جایی که بهبود قابل توجهی نسبت به تنظیمات قبلی با افزایش عمق به ۱۶-۱۹ لایه وزنی، همانند شبکه محبوب VGG حاصل شد. در اینجا نیز متوجه شدیم که تنظیم عمق و تعداد نودها در هر لایه برای دقت مدل بسیار مهم است.

پارامترها پیکربندی شبکه و دقت آن را کنترل می‌کنند، بنابراین شبکه باید به‌طور کامل برای این پارامترها بهینه‌سازی و تنظیم شود. یک پیکربندی HEARO با لیست زیر از هایپرپارامترها تعیین می‌شود:

$$\mathbf{HEARO\_hparams} = [L,\ n1, ..., nL,\ \sigma 1, ..., \sigma L,\ \lambda,\ \alpha,\ nb,\ epochs],$$

که در آن، تعداد epoch تعداد تکرارهای آموزشی روی کل مجموعه آموزشی X است، σi تابع فعال‌سازی برای لایه i می‌باشد (۱، ۲، ۳ یا ۴ به ترتیب برای ReLU، سیگموئید، تانژانت هیپربولیک ، یا Leaky ReLU)، و بقیه موارد همان‌طور که در بالا توضیح داده شد هستند.

بنابراین، با توجه به لیست هایپرپارامترهای HEARO ،چارچوب HEARO به خودی خود بر روی مجموعه داده‌های آموزشی ورودی X و نتایج مشخص شده Y آموزش می‌بیند (و وزن‌های W و b را تعیین می‌کند)، و چالش اکنون انتخاب بهترین هایپرپارامترها است.

❖ روش‌شناسی بهینه‌سازی و معماری HEARO-5

➢ اطلاعات در مورد دیتاست

HEARO از داده‌های آموزشی و آزمایشی موجود در مخزن یادگیری ماشین دانشگاه کالیفرنیا، اروین (UCI) استفاده می‌کند. داده‌ها پیش‌پردازش شده‌اند، به‌طوری که مقادیر گمشده با مقدار -۱ جایگزین شده‌اند تا تأثیر زیادی بر مدل الگوریتم نگذارند. در مجموع حدود ۱۲ مقدار گمشده وجود داشت. همچنین مقیاس‌گذاری ویژگی‌ها به طول واحد انجام شد. این مجموعه داده که توسط بنیاد کلینیک کلیدز ارائه شده است، شامل ۷۵ ویژگی از اطلاعات پزشکی بیماران برای ۳۰۳ بیمار است. ویژگی‌های زیر به‌عنوان ورودی استفاده شده‌اند :

۱. سن ۲. جنسیت. ۳ نوع درد قفسه سینه ۴ .فشار خون در حالت استراحت ۵ .کلسترول ۶ .قند خون ناشتا

۷ .نتایج الکتروکاردیوگرافی در حالت استراحت ۸ .حداکثر ضربان قلب به‌دست‌آمده ۹ .آنژین ناشی از ورزش

۱۰ .افتST ۱۱ .شیب بخش ST اوج تمرین ۱۲. رگ‌های اصلی رنگ‌آمیزی شده با فلوروسکوپی

۱۳ .نتایج اسکن قلب با تالیوم

این ویژگی‌ها به‌عنوان ویژگی‌های بهینه توسط محققان دیگر با استفاده از این مجموعه داده انتخاب شده‌اند، زیرا به‌طور نزدیک‌ترین ارتباط را با بیماری قلبی دارند.

نوع درد قفسه سینه با عدد دسته‌بندی شده است، به طوری که عدد ۱ نمایانگر آنژین معمولی است که به دلیل ورزش یا استرس ایجاد می‌شود و عدد ۲ نمایانگر آنژین غیرمعمول است که به صورت ناراحتی مداوم در قفسه سینه بروز می‌کند. معیارهای رایج مانند فشار خون در حالت استراحت، کلسترول و قند خون ناشتا می‌توانند

نمایانگر سلامت عمومی بیمار و وضعیت رگ‌های خونی او باشند، که معمولاً با تجمع پلاک به عنوان نشانه‌ای از بیماری قلبی در حال توسعه شکل می‌گیرد. نتایج الکتروکاردیوگرام (ECG) نمایش‌های بصری از فعالیت قلب هستند و می‌توانند به پزشکان یا الگوریتم‌ها کمک کنند تا تعیین کنند که آیا قلب با نرخ طبیعی پمپ می‌کند یا گردش خون دچار مشکل است. یک ناهنجاری در موج ST-T می‌تواند با ارتفاع موج اندازه‌گیری شود و معمولاً چندین معنی دارد: آنوریسم بطنی، اسپاسم شریان کرونری، یا تنگی شریان. این‌ها همگی نشانه‌های نارسایی قلب هستند و بنابراین ویژگی‌های مهمی برای الگوریتمی که بیماری قلبی را تشخیص می‌دهد به حساب می‌آیند. شیب بخش ST اوج تمرین راه مشابهی برای ارزیابی بصری عملکرد قلب است زمانی که باید خون بیشتری را در طول ورزش به گردش درآورد. در مجموعه داده، این با مقادیر ۱، نمایانگر شیب رو به بالا، ۲، نمایانگر شیب مسطح، و ۳، نمایانگر شیب رو به پایین مشخص می‌شود. اسکن‌های قلبی با تالیوم شامل عبور یک رادیوایزوتوپ از طریق جریان خون و مشاهده جایی است که در بدن به آن می‌رسد. این می‌تواند نواحی قلبی که خون کافی دریافت نمی‌کنند را شناسایی کند. فلوروسکوپی ابزار ارزیابی مشابهی است که فعالیت برخی از قسمت‌های بدن، در این مورد قلب و رگ‌های خونی نزدیک به آن را پیگیری می‌کند. رگ‌هایی که توسط آزمایش‌های فلوروسکوپی هایلایت می‌شوند می‌توانند نشان‌دهنده وجود تجمع پلاک باشند.

این مجموعه داده به دلیل دسترسی عمومی آن انتخاب شده است که به نوبه خود قابلیت بازتولید نتایج را افزایش می‌دهد. HEARO از این سیزده ویژگی برای تشخیص بیماری قلبی استفاده می‌کند، زیرا این ویژگی‌ها از نظر تنوع، در دسترس بودن و توانایی‌شان در شناسایی بیماری قلبی در مراحل مختلف توسعه، بسیار مؤثر هستند. در حالی که روش‌های دقیق‌تری مانند فلوروسکوپی و اسکن‌های تالیوم معمولاً برای کسب اطلاعات بیشتر توسط پزشک درخواست می‌شوند و به شناسایی حضور یا عدم حضور بیماری قلبی کمک می‌کنند، ویژگی‌هایی مانند قند خون و کلسترول می‌توانند شواهدی نسبتاً ضعیف از فعالیت غیرطبیعی ارائه دهند. ترکیب این ویژگی‌ها می‌تواند مدلی فراهم کند که به‌طور مؤثر روابط بین وضعیت‌های مختلف بیمار و تشخیص بیماری قلبی را ارزیابی کند.

➢ ارزیابی دقت

علاوه بر اندازه‌گیری درصد دقت، ما از روش اعتبارسنجی K-fold cross validation برای ارزیابی دقیق‌تر دقت استفاده می‌کنیم. این روش استاندارد به ویژه برای ارزیابی دقیق پیش‌بینی‌ها زمانی که اندازه مجموعه داده‌های آموزشی کوچک است، مانند مورد ما، کاربرد دارد. این تکنیک همچنین به شناسایی احتمالات overfittingکمک می‌کند، که معمولاً با تنظیمات گسترده و استفاده از داده‌های کم ممکن است رخ دهد. برای حفظ نسبت تقریباً ۲:۱ بین داده‌های آموزشی و آزمایشی، عمدتاً از 3-fold cross validation استفاده می‌کنیم، که در آن دو بخش برای آموزش و یک بخش برای آزمایش اختصاص داده می‌شود.

علاوه بر این، ما از (Matthews Correlation Coefficient - MCC) برای تحلیل توانایی‌های تعمیم‌دهی الگوریتم استفاده می‌کنیم، به ویژه زمانی که مجموعه داده دارای توزیع کلاس نامتعادل است. در مجموعه داده مخزن یادگیری ماشین کلیدز، توزیع کلاس‌های دو حالت ممکن (۰ و ۱) به این صورت است: ۱۶۴ نمونه از کلاس '۰' و ۱۳۹ نمونه از کلاس '۱'. MCC ارزیابی می‌کند که الگوریتم چگونه در پیش‌بینی تمام نتایج ممکن داده‌ها عمل می‌کند، بدون توجه به نسبت آن‌ها در مجموعه داده. این روش به تحلیل دقیق‌تری از درصد دقت که ممکن است به دلیل عدم تعادل کلاس‌ها دچار سوگیری باشد، کمک می‌کند. فرمول تعریف MCC به صورت زیر است:

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

که در آن TP نمایانگر مثبت‌های واقعی (true positives)، TN نمایانگر منفی‌های واقعی (true negatives)، FP نمایانگر مثبت‌های کاذب (false positives) و FN نمایانگر منفی‌های کاذب (false negatives) است.

➢ روش بهینه سازی

چارچوب HEARO به ما این امکان را می‌دهد که به‌راحتی تنظیمات مختلف را بر اساس آزمون‌های دقت اجرا و مقایسه کنیم، که آن را به یک گزینه بسیار مناسب برای بهینه‌سازی و تنظیم تجربی تبدیل می‌کند. این فرآیند شامل تولید تعداد زیادی پیکربندی ممکن و اجرای آن‌ها بر روی یک پلتفرم مشخص به منظور کشف بهترین نتایج است.

اثربخشی بهینه‌سازی تجربی به پارامترهای انتخاب‌شده برای بهینه‌سازی و روش جستجوی هیوریستیک مورد استفاده بستگی دارد. یکی از معایب این روش، زمان‌بر بودن جستجو برای یافتن بهترین پیکربندی است، اما در مورد ما این مسئله مشکلی ایجاد نمی‌کند زیرا اندازه مجموعه داده بزرگ نیست. علاوه بر این، ما فضای جستجو را با انتخاب مقادیر زیر محدود کردیم: $L = 2..10$, $nL = 1$, $ni = 1..13$, $\lambda \in \{0.001, 0.01, 0.1\}$

تنظیم‌سازی L2 با $\alpha \in \{0, 0.7, 1\}$, $nb = N$, and $epochs = 6000$.

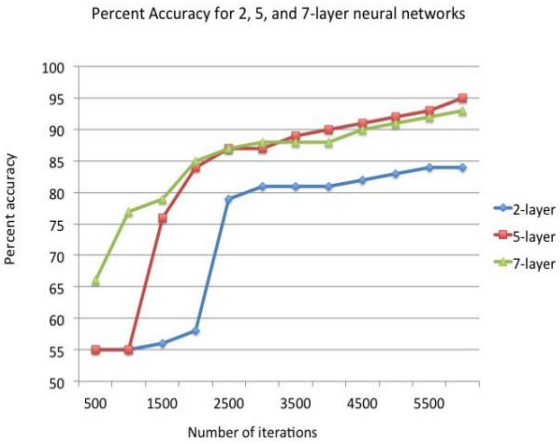
آزمایش کامل فضای جستجوی توصیف‌شده در بالا با استفاده از اسکریپت‌های خودکار پایتون برای تولید پیکربندی‌های مختلف و اجرای آن‌ها نشان داد که پیکربندی زیر:

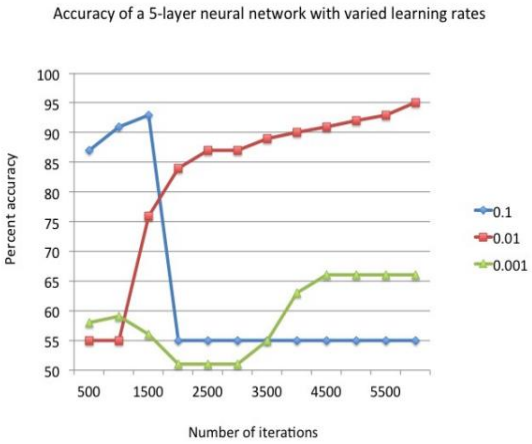
$$\textbf{HEARO-5} = [5, \quad 9,7,5,3,1, \quad 1,1,1,1,2, \quad 0.01, \quad 0.7, \quad 200, \quad 6000]$$

بهترین دقت را به دست می‌آورد، که در بخش نتایج به طور مفصل‌تری توضیح داده شده است.

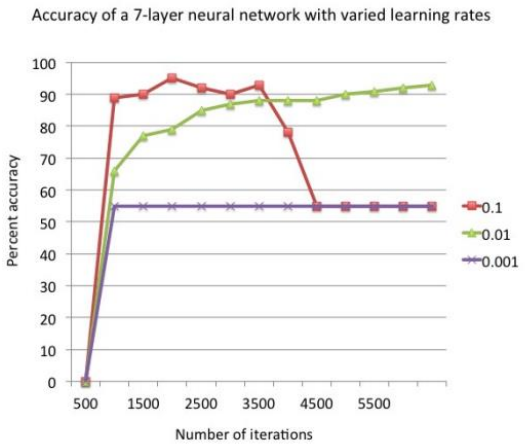
❖ نتایج

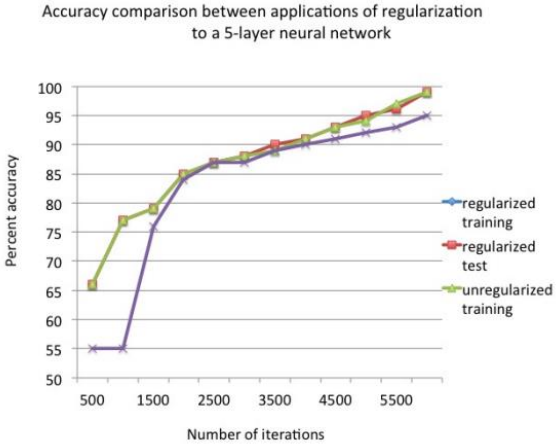
بهینه‌سازی تجربی در فضای جستجوی توصیف‌شده نشان داد که معماری HEARO-5 بالاترین دقت را ارائه می‌دهد. درصد دقت HEARO-5 با $\alpha = 0$ (بدون منظم‌سازی) در شکل سمت چپ، با دقت پیکربندی‌های دارای ۲ و ۷ لایه مقایسه شده است. تمام نمودارها عملکرد الگوریتم بر روی داده‌های آزمایشی را نشان می‌دهند، مگر اینکه خلاف آن ذکر شده باشد. شکل سمت راست، و شکل سمت چپ(شکل دوم)، تأثیر نرخ یادگیری λ را بر HEARO-5 و یک معماری HEARO با ۷ لایه به ترتیب نشان می‌دهند.





HEARO-5 با $\alpha = 0.7$ دقت ۹۹٪ بر روی داده‌های آزمایشی و Matthews correlation coefficient ۰.۹۸ را نشان داد. این نتایج در ، سمت راست نمایش داده شده است.





➢ مقایسه با نتایج منتشرشده قبلی

محققان استنفورد از یک شبکه عصبی کانولوشنی استفاده کردند و امتیازهای دقت، یادآوری و F1 معادل ۰.۸۰، ۰.۸۲ و ۰.۸۰ به دست آوردند. HEARO-5 نتایج بهتری را ارائه می‌دهد و به ترتیب دقت، یادآوری و F1 برابر با ۰.۹۸، ۱ و ۰.۹۹ را به دست می‌آورد. در سال ۲۰۱۶، Aravinthan و همکاران از یک دسته‌بند نایو بایز و شبکه عصبی مصنوعی برای این مجموعه داده استفاده کردند که دقت‌های ۸۱.۳٪ و ۸۲.۵٪ را به ترتیب به دست آوردند . مطالعه‌ای که در نشریه بین‌المللی برنامه‌های کامپیوتری (Marikani) منتشر شد، نتایج دقت ۹۵.۴٪ و ۹۶.۳٪ را برای الگوریتم‌های درخت تصمیم و جنگل تصادفی گزارش کرد.

جالب است که اشاره کنیم رگرسیون لجستیک نیز وجود دارد و واقعاً کمترین دقت را دارد، که احتمالاً به دلیل رویکرد آن در برازش داده‌های ویژگی‌های متغیر با همبستگی غیرخطی به بیماری قلبی است.

علاوه بر دقت، K-fold cross validation ما تأیید کردند که HEARO-5 به طور مؤثری overfitting را کاهش می‌دهد، به طوری که دقت اعتبارسنجی متقاطع تقریباً با دقت بر روی داده‌های آزمایشی با نسبت تعیین‌شده برابر بود.

MCC معادل ۰.۹۸ برای HEARO-5 نشان‌دهنده ارزیابی دقیق HEARO-5 از تمامی نتایج کلاس‌ها است. Matthews correlation coefficient از -۱ تا ۱ متغیر است، که در آن ۱ نمایانگر دقت کاملاً متوازن است. بنابراین، نتایج MCC معادل ۰.۹۸ و دقت ۹۹٪ نشان‌دهنده مدل تحلیلی جامع داده‌های الگوریتم است که به هیچ نتیجه خاصی متمایل نشده است.

➢ تاثیرات Regularization

در حالی که شبکه عصبی عمیق بدون تنظیم‌سازی تفاوتی بین دقت آموزش و دقت آزمایش (۹۹٪ در آموزش و ۹۳٪ در آزمایش) نشان می‌دهد، تنظیم سازی دقت داده‌های آزمایشی را به ۹۹٪ افزایش داد. تنظیم‌سازی با کاهش تأثیر نقاط پرت (ویا داده‌های گمشده) بر داده‌های آموزشی، دقت را بهبود بخشید. در یک مجموعه داده نسبتاً کوچک، نقاط پرت می‌توانند توانایی الگوریتم را برای یادگیری از روابط ثابت در داده‌های آموزشی مختل کنند و ارزش علمی اضافی ایجاد نکنند. بنابراین، با تنظیم تأثیر نقاط پرت بر یادگیری، تنظیم‌سازی توانایی الگوریتم را برای تعمیم بهبود می‌بخشد در حالی که استاندارد علمی یکسان را حفظ می‌کند. از آنجایی که تنظیم‌سازی موجب کاهش overfitting در داده‌های آموزشی می‌شود، دقت الگوریتم به طور معمول بر روی داده‌های آموزشی کاهش می‌یابد. در مورد HEARO-5 با $\alpha = 0$ (بدون تنظیم‌سازی)، دقت ۹۹٪ در آموزش با دقت نسبتاً پایین‌تر در آزمایش نشان‌دهنده Overfitting است. الگوریتم از نقاط پرت نویزی در داده‌های آموزشی یاد می‌گیرد که توانایی آن را برای تعمیم روابط گسترده‌تر در داده‌ها کاهش می‌دهد. یادگیری این روابط برای دقت در مجموعه داده‌های ناآشنا حیاتی است.

❖نتیجه‌گیری و مسیرهای آینده

این تحقیق به بررسی و نشان دادن پتانسیل استفاده از تحلیل داده‌های مبتنی بر شبکه‌های عصبی عمیق (DNN)برای تشخیص بیماری‌های قلبی بر اساس داده‌های کلینیکی معمول پرداخت. نتایج نشان می‌دهند که با استفاده از طراحی‌های انعطاف‌پذیر و تنظیم دقیق، تکنیک‌های تحلیل داده‌های DNN می‌توانند دقت بسیار بالایی ٪۹۹ دقت و MCC۰.۹۸ به دست آورند که به طور قابل توجهی از تحقیقات منتشرشده فعلی در این حوزه پیشی می‌گیرد و به تأسیس جذابیت استفاده از تحلیل داده‌های ML DNN در پزشکی تشخیصی کمک می‌کند.

در حالی که پیشرفت‌های کنونی عمدتاً تحقیقات با نتایج اثبات مفهومی عالی هستند، تحقیقات و توسعه بیشتری برای تبدیل آن به یک ابزار تشخیصی قابل اعتماد ضروری است، به طوری که پزشکان بتوانند به طور منظم از آن برای انجام تشخیص‌های دقیق‌تر استفاده کنند. تحقیقات بیشتری در حوزه تحلیل داده‌ها و تقاطع آن با تشخیص پزشکی مبتنی بر داده‌ها مورد نیاز است، شامل جستجوی خودکار بهترین ویژگی‌ها و همچنین گسترش یا کاهش ویژگی‌ها، مثلاً به دلیل عدم وجود داده‌های کلینیکی خاص. جهت‌گیری‌های آینده شامل گسترش این تحلیل به منظور ساخت مدل‌های جامع‌تری است که شامل تصاویربرداری‌های قلب و داده‌های تصویربرداری CT باشد. ویژگی‌های بیشتر می‌توانند داده‌های بیشتری برای الگوریتم فراهم کنند، مدل پیچیده‌تری ایجاد کنند و پیش‌بینی دقیق‌تر و جامع‌تری را تضمین کنند. یکی دیگر از زمینه‌های تحقیق آینده شامل استفاده از ابزارهای بهینه‌سازی سرعت و بک‌اندهای جبر خطی شتاب‌یافته مانند MagmaDNN برایGPU ها است تا توانایی الگوریتم را در پردازش مقادیر زیادی از داده‌ها و یافتن بهترین پیکربندی‌ها به صورت موازی بهبود بخشد.

❖ تحلیل و آنالیز کد

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import svm
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.models import Sequential
from sklearn.linear_model import LogisticRegression
from keras.layers import Dense
from keras import regularizers
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from google.colab import files
uploaded = files.upload()
```

import numpy as np:

این خط، کتابخانه NumPy را وارد می‌کند که برای عملیات‌های عددی پیشرفته و کار با آرایه‌های چندبعدی استفاده می‌شود. این کتابخانه ابزارهایی برای انجام محاسبات ریاضی و علمی فراهم می‌آورد.

import pandas as pd:

این خط، کتابخانه Pandas را وارد می‌کند که به مدیریت داده‌ها و تحلیل آن‌ها با استفاده از ساختارهای داده‌ای مانند DataFrame و Series اختصاص دارد. این ابزار برای بارگذاری، تمیز کردن و تجزیه و تحلیل داده‌ها بسیار مفید است.

import seaborn as sns:

با وارد کردنSeaborn ، ابزار قدرتمند برای ترسیم نمودارهای آماری و تجسم داده‌ها در اختیار شما قرار می‌گیرد Seaborn .بر پایه‌ی Matplotlib ساخته شده و برای ایجاد نمودارهای پیچیده و زیبا استفاده می‌شود.

from sklearn import svm:

این خط، ماژول svm از کتابخانه scikit-learn را وارد می‌کند. این ماژول شامل الگوریتم‌های ماشین بردار پشتیبان (SVM) است که برای مسائل دسته‌بندی و رگرسیون استفاده می‌شود.

from sklearn import preprocessing:

این خط، ماژول preprocessing از scikit-learn را وارد می‌کند که شامل ابزارهایی برای پیش‌پردازش داده‌ها، مانند نرمال‌سازی و استانداردسازی ویژگی‌ها است.

from sklearn.model_selection import train_test_split:

با وارد کردن تابعtrain_test_split ، می‌توانید داده‌ها را به دو مجموعه آموزش و تست تقسیم کنید. این کار برای ارزیابی عملکرد مدل‌ها ضروری است.

import matplotlib.pyplot as plt:

این خط، کتابخانه Matplotlib را وارد می‌کند که برای ترسیم نمودارها و تجسم داده‌ها استفاده می‌شودpyplot ،. زیرمجموعه‌ای از این کتابخانه، توابع ساده‌ای برای ایجاد نمودارهای مختلف فراهم می‌آورد.

from keras.models import Sequential:

با وارد کردن Sequential ازKeras ، می‌توانید مدل‌های شبکه‌های عصبی به صورت خطی و لایه به لایه بسازید. این مدل‌ها برای ایجاد شبکه‌های عصبی ساده مناسب هستند.

from sklearn.linear_model import LogisticRegression:

این خط، مدل LogisticRegression از scikit-learn را وارد می‌کند که یک الگوریتم محبوب برای مسائل دسته‌بندی است. این مدل به ویژه برای پیش‌بینی احتمال وقوع یک کلاس خاص استفاده می‌شود.

from keras.layers import Dense:

این خط، لایه Dense از Keras را وارد می‌کند. لایه Dense یک لایه کاملاً متصل است که به طور گسترده در شبکه‌های عصبی استفاده می‌شود و هر نورون در این لایه به همه نورون‌های لایه قبلی متصل است.

from keras import regularizers:

این خط، ماژول regularizers از Keras را وارد می‌کند که برای اضافه کردن تکنیک‌های جریمه‌گذاری به مدل‌های شبکه‌های عصبی به کار می‌رود. این تکنیک‌ها به کاهش خطر بیش‌برازش (overfitting) کمک می‌کنند.

from sklearn import metrics:

با وارد کردن ماژول metrics از scikit-learn ، می‌توانید از ابزارهای ارزیابی مدل، مانند دقت، بازخوانی و F1- برای سنجش عملکرد مدل‌ها استفاده کنید.

from sklearn.neighbors import KNeighborsClassifier:

این خط، مدل KNeighborsClassifier از scikit-learn را وارد می‌کند که برای دسته‌بندی داده‌ها بر اساس الگوریتم K نزدیک‌ترین همسایه استفاده می‌شود. این مدل به تعیین کلاس هر نمونه بر اساس نزدیک‌ترین نمونه‌های آموزشی کمک می‌کند.

from google.colab import files:

این خط، ماژول files از google.colab را وارد می‌کند. این ماژول به شما امکان می‌دهد فایل‌ها را از سیستم محلی خود در محیط Google Colab بارگذاری کنید.

uploaded = files.upload():

با اجرای این تابع، به شما این امکان را می‌دهد که فایل‌هایی را از سیستم محلی خود به محیط Google Colab آپلود کنید. فایل‌های آپلود شده در متغیر uploaded ذخیره می‌شوند و می‌توان به آن‌ها دسترسی پیدا کرد.

```
tdata = pd.read_csv("Integrated.csv",header=None,na_values=[-9])
```

این تابع از کتابخانه Pandas برای بارگذاری داده‌ها از فایل CSV استفاده می‌شود و آن را به یک DataFrame در کتابخانه Pandas تبدیل می‌کند.

```
new_data = tdata[[2,3,8,9,14,15,16,17,18,31,57]].copy()
#data = new_data.values
```

در این بخش از کد، شما یک زیرمجموعه از داده‌ها را از tdata (DataFrame) انتخاب شده و در متغیر جدیدی به نام new_data ذخیره شده اند. که در اینجا، ستون‌های با ایندکس‌های ۲و۳و۸و۹و۱۴و۱۵و۱۶و۱۷و۱۸و۳۱و۵۷ انتخاب شده‌اند. متد copy() از pandas برای ایجاد یک کپی از DataFrame انتخاب شده استفاده می‌شود. این کار به شما اطمینان می‌دهد که تغییرات بعدی روی new_data تاثیری بر tdata نخواهد داشت و برعکس.

```
new_data.columns = ['Age','Sex','Chest Pain','Blood Pressure','Smoking Years','Fasting Blood Sugar','Diabetes History',
                    'Family history Cornory','ECG','Pulse Rate','Target']
```

در این خط کد، نام ستون‌های DataFrame تغییر داده می‌شود تا با نام‌های توصیفی و واضح‌تری جایگزین شوند. این کار به شما کمک می‌کند تا با داده‌ها راحت‌تر کار کنید و اطلاعات هر ستون به راحتی قابل فهم باشد.

```
print(new_data.info())
```

در این خط کد، تابع info() از DataFrame فراخوانی می‌شود تا اطلاعات کلی درباره‌ی ساختار و محتوای آن نمایش داده شود. این اطلاعات به شما کمک می‌کند تا وضعیت کلی داده‌های خود را بررسی کنید و از ویژگی‌های آن‌ها آگاه شوید.

```
new_data['Blood Pressure'].fillna(new_data['Blood Pressure'].mean(),inplace=True)
new_data['Smoking Years'].fillna(new_data['Smoking Years'].mean(),inplace=True)
new_data['Fasting Blood Sugar'].fillna(new_data['Fasting Blood Sugar'].mean(),inplace=True)
new_data['Diabetes History'].fillna(new_data['Diabetes History'].mode()[0],inplace=True)
new_data['Family history Cornory'].fillna(new_data['Family history Cornory'].mode()[0],inplace=True)
new_data['ECG'].fillna(new_data['ECG'].mean(),inplace=True)
new_data['Pulse Rate'].fillna(new_data['Pulse Rate'].mean(),inplace=True)
```

در این خط کدها، مقادیر خالی (NaN) در ستون‌های مختلف DataFrame با مقادیر جایگزین پر می‌شوند. این کار به منظور اطمینان از کامل بودن داده‌ها برای تحلیل‌های بعدی انجام می‌شود. در اینجا از دو روش مختلف برای پر کردن مقادیر خالی استفاده شده است :میانگین و مد.

fillna() : این متد برای پر کردن مقادیر خالی استفاده می‌شود.

new_data[…].mean() : میانگین مقادیر موجود در این ستون محاسبه می‌شود و به عنوان مقدار جایگزین برای مقادیر خالی استفاده می‌شود.
new_data[…].mode() : مد مقادیر موجود در این ستون محاسبه می‌شود و به عنوان مقدار جایگزین برای مقادیر خالی استفاده می‌شود.

inplace=True : این گزینه باعث می‌شود که تغییرات مستقیماً بر روی DataFrame اصلی اعمال شود و نیازی به ایجاد یک کپی جدید نباشد.

```
print(new_data)
#print(new_data.info())
sns.set(style="ticks", color_codes=True)
ax = sns.pairplot(new_data,palette="husl")
plt.show()
#plt.savefig("Pair_Plot.jpg")
```

در این خط کدها، به تحلیل بصری داده‌ها پرداخته می‌شود تا درک بهتری از روابط بین ویژگی‌ها بدست آید. در اینجا از یک Pair Plot استفاده شده است که به صورت تصویری رابطه بین تمام ویژگی‌ها را نمایش می‌دهد .این ابزار به شناسایی الگوها، روندها، و نقاط غیرعادی کمک می‌کند . با استفاده از این ابزار، می‌توان به سادگی درک کرد که آیا ویژگی‌های مختلف با یکدیگر ارتباط معناداری دارند یا خیر و الگوهای پیچیده‌تر را شناسایی کرد.

```
print(new_data['Target'].value_counts())
```

این خط کد برای شمارش تعداد نمونه‌ها در هر دسته از ویژگی Targetدر DataFrame استفاده می‌شود. این ویژگی به طور معمول نشان‌دهنده برچسب‌های کلاس در مسائل طبقه‌بندی است و در اینجا نشان‌دهنده وجود یا عدم وجود بیماری قلبی است.

به طور کلی، هدف از این بررسی این است که ببینید آیا داده‌ها به طور متعادل بین کلاس‌های مختلف تقسیم شده‌اند یا یکی از کلاس‌ها بر دیگری غالب است. اگر توزیع کلاس‌ها نامتعادل باشد، ممکن است نیاز به تکنیک‌های خاصی برای مقابله با آن باشد، مانند استفاده از وزن‌دهی کلاس‌ها یا تکنیک‌های نمونه‌برداری.

```
new_data.replace({'Target' : 0}, 0, inplace=True)
new_data.replace({'Target' : 1}, 0, inplace=True)
new_data.replace({'Target' : 2}, 0, inplace=True)
new_data.replace({'Target' : 3}, 1, inplace=True)
new_data.replace({'Target' : 4}, 1, inplace=True)
```

این کد برای جایگزینی مقادیر موجود در ستون Target از DataFrame با مقادیر جدید استفاده می‌شود. برای ساده‌سازی تحلیل یا فرآیند مدل‌سازی انجام شده است. در اینجا، تمام مقادیر غیر از ۳و۴ به ۰ تبدیل شده‌اند و مقادیر ۳و۴ به ۱ تغییر یافته‌اند.

```
data = new_data.values
print(data)
#print(data.shape)
```

این کد برای استخراج مقادیر داده‌ها از DataFrame و سپس چاپ آنها استفاده می‌شود.

new_data.values : این ویژگی از DataFrame Pandas مقادیر موجود در DataFrame را به صورت یک آرایه NumPy باز می‌گرداند. در اینجا، data به یک آرایه دو بعدی تبدیل می‌شود که شامل مقادیر همه ستون‌ها و ردیف‌های DataFrame new_data است.

```
X = data[:,:-1]
#print(X)
y = data[:,-1]
#print(y)
y = y.reshape((y.shape[0],1))
```

این کد به منظور جدا کردن ویژگی‌ها و برچسب‌ها از آرایه data و آماده‌سازی آنها برای استفاده در مدل‌سازی نوشته شده است.

data[:,:-1] : این خط تمام سطرهای آرایه data و تمام ستون‌ها به جز آخرین ستون را انتخاب می‌کند. به عبارت دیگر، X شامل تمامی ویژگی‌ها (ویژگی‌های مستقل) از داده‌ها خواهد بود و آخرین ستون که معمولاً برچسب‌ها (کلاس‌ها) هستند را حذف می‌کند.

data[:,-1] : این خط تمام سطرهای آرایه data و تنها آخرین ستون را انتخاب می‌کند. به عبارت دیگر، y شامل تنها برچسب‌ها یا کلاس‌های هدف است که در آخرین ستون data قرار دارند.

y.reshape((y.shape[0],1)) : این خط شکل آرایه y را به صورت (n_samples, 1) تغییر می‌دهد، جایی که n_samples تعداد نمونه‌ها است. این تغییر شکل باعث می‌شود که y به صورت یک آرایه دو بعدی با یک ستون تبدیل شود. این کار معمولاً برای سازگار کردن شکل داده‌ها با نیازهای مدل‌های یادگیری ماشین انجام می‌شود، به ویژه زمانی که مدل‌های یادگیری ماشین انتظار دارند که برچسب‌ها به صورت یک آرایه دو بعدی ارائه شوند.

```
n_X = preprocessing.normalize(X)
n_y = preprocessing.normalize(y)
n_y = n_y.reshape((n_y.shape[0],))
```

این کد برای نرمال‌سازی داده‌ها و آماده‌سازی آنها نوشته شده است.

preprocessing.normalize(X) : این خط داده‌های ویژگی‌های X را نرمال‌سازی می‌کند. نرمال‌سازی به معنای مقیاس‌بندی داده‌ها به بازه‌ای خاص (معمولاً بین ۰ و ۱ یا به طور استاندارد با نرمال‌سازی واحد طول) است. این کار برای جلوگیری از تاثیر مقیاس‌های مختلف ویژگی‌ها بر روی مدل یادگیری ماشین انجام می‌شود. تابع normalize از کتابخانه sklearn.preprocessing این کار را انجام می‌دهد.

preprocessing.normalize(y) : این خط داده‌های برچسب‌ها y را نرمال‌سازی می‌کند.

n_y.reshape((n_y.shape[0],)) : این خط تغییر شکل n_y را از یک آرایه دو بعدی به یک آرایه یک بعدی انجام می‌دهد. این تغییر شکل برای اطمینان از اینکه n_y به صورت یک بعدی (با تنها یک بعد از نمونه‌ها) باشد، انجام می‌شود. معمولاً برچسب‌ها به صورت یک بعدی (یک ستون) نیاز هستند.

```
training_X, testing_X, training_y, testing_y = train_test_split(n_X,n_y,test_size=0.10,random_state=70)

print('Training data: '+str(training_X.shape) +' '+ str(training_y.shape))
print('Testing  data: '+str(testing_X.shape) +' '+str(testing_y.shape))
```

این کد برای تقسیم داده‌ها به مجموعه‌های آموزش و تست استفاده می‌شود و ابعاد هر مجموعه را چاپ می‌کند.

train_test_split(n_X, n_y, test_size=0.10, random_state=70) : این تابع از کتابخانه sklearn.model_selection برای تقسیم داده‌های ویژگی‌ها n_X و برچسب‌ها n_y به مجموعه‌های آموزش و تست استفاده می‌شود.

n_X : داده‌های ویژگی‌ها برای تقسیم.

n_y : داده‌های برچسب‌ها برای تقسیم.

test_size=0.10 : درصد داده‌ها که به مجموعه تست اختصاص داده می‌شود. در اینجا، ۱۰٪ از داده‌ها برای تست و ۹۰٪ برای آموزش استفاده می‌شود.

random_state=70 : مقدار تصادفی برای اطمینان از تکرارپذیری تقسیم داده‌ها. استفاده از مقدار ثابت باعث می‌شود که در هر بار اجرای کد، تقسیم داده‌ها به همان صورت انجام شود.

training_X.shape و training_y.shape : ابعاد داده‌های آموزش شامل تعداد نمونه‌ها و ویژگی‌ها برای X و تعداد نمونه‌ها برای y را چاپ می‌کند.

testing_X.shape و testing_y.shape : ابعاد داده‌های تست شامل تعداد نمونه‌ها و ویژگی‌ها برای X و تعداد نمونه‌ها برای y را چاپ می‌کند.

```
print('Support Vector Machine')
#clf  = svm.SVC(gamma='auto')
clf = svm.SVC(kernel='rbf',C=5,gamma='auto')
clf.fit(training_X,training_y)
#prediction = clf.predict(testing_X)
#print(prediction)
#print(testing_y)
r = clf.score(testing_X,testing_y)
print(r)
```

این کد برای آموزش و ارزیابی یک مدل **ماشین بردار پشتیبان** (SVM)با هسته رادیکال (RBF) استفاده می‌شود.

svm.SVC : این تابع از کتابخانه sklearn.svm برای ایجاد مدل ماشین بردار پشتیبان (SVM) استفاده می‌شود.
kernel='rbf' : نوع هسته (Kernel) مدلSVM در اینجا، از هسته تابع شعاعی پایه (Radial Basis Function) استفاده می‌شود.
C=5 : پارامتر تنظیم کننده (Regularization Parameter) که کنترل می‌کند چه مقدار از خطاهای آموزش را مجاز بدانیم. مقدار بالاتر باعث می‌شود مدل با داده‌های آموزش بهتر تطابق پیدا کند.

gamma='auto' : پارامتر تنظیم کننده برای هسته RBF که تعیین می‌کند تأثیر هر نمونه داده بر روی مدل چگونه است. مقدار 'auto' باعث می‌شود مقدار gamma به صورت خودکار انتخاب شود.

clf.fit(training_X, training_y) : این تابع مدل SVM را با داده‌های آموزشی training_X و برچسب‌های training_y آموزش می‌دهد.

clf.score(testing_X, testing_y) : این تابع دقت مدل را بر روی داده‌های تست testing_X و برچسب‌های testing_y محاسبه می‌کند. دقت نشان‌دهنده درصد نمونه‌های درست پیش‌بینی شده توسط مدل است.

```
print('Logistic Regression')
clf = LogisticRegression(random_state=0,solver='lbfgs',multi_class='multinomial')
clf.fit(training_X,training_y)
clf.predict(testing_X)
r = clf.score(testing_X,testing_y)
print(r)
```

این کد برای پیاده سازی Logistic regression است.

LogisticRegression :از کلاس رگرسیون لجستیک کتابخانه sklearn.linear_modelاستفاده می‌کند تا مدل رگرسیون لجستیک ساخته شود.

Random_state=0 : این پارامتر به منظور تولید نتایج قابل تکرار استفاده می‌شود. در اینجا مقدار ۰ انتخاب شده است.

solver='lbfgs' : این پارامتر نوع الگوریتم بهینه‌سازی را مشخص می‌کند. lbfgs یک الگوریتم متداول برای حل مدل‌های رگرسیون لجستیک است که برای مدل‌های چند کلاسه و داده‌های بزرگ مناسب است.

multi_class='multinomial' : این پارامتر مشخص می‌کند که مدل برای دسته‌بندی چند کلاسه (multi-class classification) تنظیم شده است. به این معنا که مدل می‌تواند بیش از دو کلاس را پیش‌بینی کند.

clf.fit(training_X, training_y) : این تابع مدل رگرسیون لجستیک را با استفاده از داده‌های آموزشی training_X و برچسب‌های training_y آموزش می‌دهد.

clf.predict(testing_X) : این تابع داده‌های تست testing_X را به مدل می‌دهد تا مدل بر اساس آموزش‌های قبلی، نتایج پیش‌بینی شده را خروجی دهد. در اینجا نتیجه پیش‌بینی چاپ نمی‌شود، اما این خط از نظر عملکرد مدل اهمیت دارد.

clf.score(testing_X, testing_y) : این تابع دقت مدل را بر روی داده‌های تست محاسبه می‌کند. دقت نشان‌دهنده درصد نمونه‌های درست پیش‌بینی شده توسط مدل است.

```
print('K Nearest Neighbors')
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(training_X,training_y)
y_prediction = knn.predict(testing_X)
score = metrics.accuracy_score(testing_y,y_prediction)
print(score)

training_y = np.expand_dims(training_y, axis=1)
```

KNeighborsClassifier : از کلاس K نزدیک‌ترین همسایه‌ها در کتابخانه sklearn.neighbors استفاده می‌شود تا یک مدل KNN ساخته شود.

n_neighbors=3 : این پارامتر تعداد همسایه‌های نزدیک را که باید در نظر گرفته شوند، مشخص می‌کند. در اینجا، ۳ به این معناست که مدل از ۳ نمونه نزدیک برای تصمیم‌گیری استفاده می‌کند.

knn.fit(training_X, training_y) : این تابع مدل KNN را با استفاده از داده‌های آموزشی training_X و برچسب‌های training_y آموزش می‌دهد.

knn.predict(testing_X) : این تابع داده‌های تست testing_X را به مدل می‌دهد تا مدل بر اساس آموزش‌های قبلی، نتایج پیش‌بینی شده را خروجی دهد. نتیجه پیش‌بینی شده در متغیر y_prediction ذخیره می‌شود.

metrics.accuracy_score(testing_y, y_prediction) : این تابع دقت مدل را بر اساس داده‌های واقعی testing_yو پیش‌بینی شده y_prediction محاسبه می‌کند. دقت نشان‌دهنده درصد نمونه‌های درست پیش‌بینی شده توسط مدل است.

np.expand_dims(training_y, axis=1) : این خط داده‌های training_y را یک بعدی می‌کند، به این معنا که یک بعد اضافی به آرایه training_y اضافه می‌کند. این کار برای سازگاری داده‌ها در مراحل بعدی ضروری است.

```
print('Multi Layer Perceptron Deep Learning Model')
model = Sequential()
model.add(Dense(units=10,activation='relu',input_dim=10,kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))
model.add(Dense(units=64,activation='relu',kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))
model.add(Dense(units=64,activation='relu',kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))
model.add(Dense(units=8,activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',optimizer='Adadelta',metrics=['accuracy'])
model.fit(training_X,training_y,epochs=200)

loss_and_metrics = model.evaluate(testing_X,testing_y)

print(loss_and_metrics)
```

این کد یک مدل Perceptronچندلایه یا مدل Deep Learning را با استفاده از کتابخانه Keras ایجاد، کامپایل، آموزش و ارزیابی می‌کند.

Sequential() : یک مدل ترتیبی ایجاد می‌کند که لایه‌ها را به صورت زنجیره‌ای به آن اضافه می‌کنیم.

Dense(units=10) : یک لایه کاملاً متصل (Dense) با ۱۰ نورون ایجاد می‌کند.
' activation='relu : از تابع فعال‌سازی ReLU برای این لایه استفاده می‌کند.
input_dim=10 : تعداد ویژگی‌های ورودی را به ۱۰ تنظیم می‌کند.
kernel_regularizer=regularizers.l2(0.01) : از منظم‌سازی L2 برای کاهش overfitting استفاده می‌کند.

activity_regularizer=regularizers.l1(0.01) : از منظم‌سازی L1 برای تنظیم فعالیت نورون‌ها استفاده می‌کند.

در ادامه لایه‌های کاملاً متصل با ۶۴ نورون را به مدل اضافه می‌کند. همانند لایه اول، از تابع فعال‌سازی ReLU و منظم‌سازی L2 و L1 استفاده می‌شود.

'activation='softmax' : از تابع فعال‌سازی Softmax برای خروجی چندکلاسه استفاده می‌کند. این تابع اعداد را به مقادیر احتمالی تبدیل می‌کند.

'loss='sparse_categorical_crossentropy : از تابع هزینه sparse categorical cross-entropy برای محاسبه خطا استفاده می‌کند. این تابع برای مسائل دسته‌بندی چندکلاسه مناسب است.
'optimizer='Adadelta : از الگوریتم بهینه‌سازی Adadelta برای به‌روزرسانی وزن‌ها استفاده می‌کند.

['metrics=['accuracy : معیار دقت را برای ارزیابی مدل استفاده می‌کند.

model.fit(training_X, training_y, epochs=200) : مدل را با استفاده از داده‌های آموزشی training_X و training_y آموزش می‌دهد. مدل برای ۲۰۰ تا Epoch آموزش می‌بیند.

model.evaluate(testing_X, testing_y) : عملکرد مدل را بر روی داده‌های تست ارزیابی می‌کند و مقادیر خطا و دقت را برمی‌گرداند.