به نام خدا

گزارش پروژه پایانی

یادگیری ماشین

پروژه پیش‌بینی بیماری پارکینسون

بهار باطنی ۸۱۰۱۹۹۳۲۶

امیرمحمد رنجبر پازکی ۸۱۰۱۹۹۳۴۰

دانشکده برق و کامپیوتر دانشگاه تهران

زمستان ۱۳۹۹

• خلاصه مقاله:

عنوان مقاله:

Computational Diagnosis of Parkinson’s Disease Directly from Natural Speech using Machine Learning Techniques (2014)

بیماری پارکینسون یک بیماری است که باعث از بین رفتن نورون‌های مغزی می‌شود. این بیماری با علائمی مانند ویژگی‌های صورت "ماسکه"، کندی حرکات، کاهش قدرت بدنی و لرزش در حرکات شناخته می‌شود. این بیماری ۱ درصد افراد بالای ۶۵ سال را درگیر می‌کند. این بیماری در حال گسترش در دنیاست. زمانی که این بیماری تشخیص داده‌می‌شود حدود ۶۰ درصد از نورون‌ها از بین رفته‌اند و حدود ۸۰ درصد دوپامین آزاد شده‌است. پس دو هدف اصلی برای تشخیص این بیماری وجود دارد: ۱) تشخیص زودهنگام بیماری ۲) بررسی و ارزیابی میزان تاثیر درمان

در حال حاضر این ارزیابی توسط Unified Parkinson’s Disease Rating Scale (UPDRS) انجام می‌شود. این تست روانی توسط یک درمانگر انجام می‌شود و به همین دلیل متاثر از سوگیری‌های فردی است و دقیق نیست و ممکن است با واقعیت فاصله داشته باشد.

یک ایده درباره تشخیص این بیماری این است که این بیماری خیلی سریع در گفتار فرد تاثیر می‌گذارد و به همین دلیل، بررسی ویژگی‌های مربوط به سیگنال گفتار فرد می‌تواند راه خوبی برای تشخیص این بیماری باشد. ویژگی‌های انتخاب شده در این کار ویژگی‌های مربوط به تلفظ حروف صدادار هستند. این ویژگی‌های به شدت حساس به تغییرات عضلات صورت هستند. این ویژگی‌ها نمایانگر حرکات زبان، لب‌ها و فک هستند. در بیماری پارکینسون این اندام‌ها دچار اختلال حرکتی می‌شوند و به همین دلیل، این حروف به صورت مرکزی تلفظ می‌شوند. به این معنا که فرکانس‌های بالا فرکانس پایین پیدا می‌کنند و بالعکس، فرکانس‌های پایین فرکانس بالا می‌یابند. در پژوهش‌های پیشین، این گروه موفق شده‌اند با استخراج سه ویژگی معیار برای مرکزی شدن حروف صدادار تعریف کنند. با استفاده از یادگیری ماشین این گروه به دقت بالا دست یافته بودند.

در این پژوهش کار گذشته را توسعه داده‌اند. دو دستاورد نسبت به پژوهش پیشین در این پژوهش دیده می‌شود: ۱. به دنبال استخراج ویژگی‌ها بدون دخالت متخصصان رفته‌اند. ۲. توانایی تشخیص درجه‌های مختلف این بیماری فراهم شده‌است.

در این پژوهش از ۴۳ بیمار و ۹ فرد عادی مجموعه داده تهیه شده‌است. این مجموعه داده به این صورت است که هر یک از این افراد یک پاراگراف مشخص را می‌خوانند. این پاراگراف در تست‌های مربوط به تشخیص بیماری در صدا استفاده می‌شود و به نوعی است که توزیع حروف

صدادار و بی‌صدا برای استخراج ویژگی متوازن باشد. این ۴۳ بیمار توزیعی به این شکل دارند:
۴ درجه ۱، ۴ درجه ۱/۵، ۱۸ درجه ۲، ۹ درجه ۲/۵، ۷ درجه ۳، ۱ درجه ۴
این درجه‌های توسط تست سنتی به بیماران نسبت داده‌شده‌است.
از نوع نگارش این گونه برمی‌آید که این داده توسط خود محققان این پژوهش جمع‌آوری شده‌است اما مستقیما به این موضوع اشاره نشده‌است.
ابتدا بر روی این صداها یک فرآیند پنجره سازی اجرا شده‌است. این فرآیند به این صورت است که سیگنال به پنجره‌های با طول ۲۰ میلی‌ثانیه و با میزان هم‌پوشانی ۵۰ درصد تقسیم شده‌است. این کار برای بررسی سیگنال و نمونه‌برداری نیاز است.
در مرحله پیش پردازش، ابتدا مقدار DC سیگنال حذف شده‌است تا میانگین آن صفر شود. قدرت سیگنال در هر پنجره نرمالیزه شده‌است. سپس، این سیگنال فیلتر شده‌است تا فرکانس‌هایی که در محدوده گویش انسان نیست از آن حذف شود.
در بحث استخراج و انتخاب ویژگی، دو فاکتور موردتوجه قرار گرفته‌است. ویژگی‌هایی که بتوانند anomalyها را تشخیص بدهند و همچنین، ویژگی‌های استاندارد مورد استفاده در سیستم‌های تشخیص صدا باشد. این ویژگی‌ها عبارتند از:)لازم به ذکر است این انتخاب توسط متخصصان حوزه انجام شده‌است و خودکار نبوده‌است.)
۱. میزان لرزش سیگنال‌های صوتی: از گوینده‌ای به گوینده دیگر تغییر می‌کند. این ویژگی توسط الگوریتمی مشابه autocorrelation به دست‌آمده‌است.
۲. انرژی کوتاه مدت: بیانگر اندازه تغییرات است.
۳. Zero crossing rate: این ویژگی بیانگر تعداد تغییرات علامت سیگنال صوتی در یک frame است.

$$ZCR = \frac{1}{2}\Sigma_{n=1}^{N-1} \mid sgn(x[n]) - sgn(x[n-1]) \mid$$

۴. میانگین و انحراف معیار مقادیر ZCR: این مقادیر توسط فاصله دو صفر متوالي ZCR محاسبه می‌شود. این معیار در کنار خود ZCR می‌تواند بیانگر نویزی بودن یا نامتعارفی‌ها در صدا باشد.
۵. Mel Frequency Cepstral Coefficients (kDCC): این معیار نشان‌دهنده پاسخ فرکانسی صدای انسان است که پس از اجرای چند تبدیل به دست می‌آید.
در این پژوهش از classification بین هر دو درجه بیماری برای حل مسئله استفاده شده‌است. مدل استفاده‌شده برای این کار(support vector machine(SVM است. در این الگوریتم از C¯SVC kernel و تابع RBF برای یک طبقه‌بندی دو کلاسه استفاده

شده‌است.این الگوریتم به دلیل قدرت تعمیم‌دهی بالا روی نمونه‌های کوچک انتخاب شده‌است. از grid search با یک نسبت چندجمله‌ای برای تخمین پارامترهای آزاد مدل استفاده شده‌است. از هر مریض ۲۴ درصد frameها بدون جایگذاری نمونه‌برداری شده‌اند. سپس، این frameها oversample شده‌اند تا مشکل نامتعادل بودن داده‌ها برطرف شود. (تعداد بیماران در گروه‌های مختلف متفاوت است.)

عملکرد مدل با استفاده از cross validation سنجیده‌شده‌است. این عملکرد به صورت میزان صحت، میزان خطای false alarm و false negative بین هر دو درجه بیماری سنجیدی شده‌است. دقت‌های مدل بد نیست و این که به همراه دقت دو معیار دیگر نیز گزارش شده‌است خوب است. بهتر بود این کار چندین بار اجرا می‌شد و خطا به صورت میانگین و انحراف معیار گزارش می‌شد تا اثر نمونه‌گیری خاص از آن حذف می‌شد. به عنوان مثال از k¯fold cross validation استفاده می‌شد.

دقت کلی مدل ۸۱.۸ درصد ارزیابی شده‌است. این سیستم به طور خودکار می‌تواند بیماری پارکینسون را تشخیص دهد. این متد نشان داد که می‌توان به جای استفاده از متخصصان از پردازش سیگنال ویژگی‌ها را استخراج کرد. در کارهای پیش‌رو می‌توان به سراغ استخراج کاملا خودکار ویژگی‌ها رفت و از متریک‌های مطرح در تشخیص صدا استفاده نکرد. روش‌های پیش پردازش پیشرفته‌تر مانند حذف silent windowها ممکن است مفید باشند. نتایج دقت دسته‌بندی‌های مختلف در شکل زیر آمده‌است.

		Class 1					
		1	***1.5***	***2***	***2.5***	***3***	***4***
Class 2	***0***	**83.39** (17.47, 15.75)	**83.3** (18.46, 14.93)	**79.6** (19.77, 21.01)	**78.51** (22.54, 20.43)	**86.3** (10.57, 17.45)	**76.18** (32, 15.63)
	1		**78.41** (18.2, 24.97)	**79.08** (21.92, 19.92)	**81.7** (17.71, 18.89)	**85.14** (15.13, 14.58)	**85.57** (16.05, 12.80)
	1.5			**83.39** (16.93, 16.29)	**84.03** (15.04, 16.89)	**74.45** (22.78, 28.30)	**86.1** (15.42, 12.38)
	2				**76.7** (22.35, 24.24)	**83.37** (15.64, 17.6)	**81.11** (24.76, 13.01)
	2.5					**85.81** (13.51, 14.86)	**79.49** (23.92, 17.1)
	3						**86.57** (17.52, 9.33)

نتایج دسته‌بندی دو کلاسه بین درجه‌های مختلف بیماری

• بررسی و انتخاب معیار مقایسه عملکرد مدل‌ها:

معیارهای زیر برای ارزیابی عملکرد در این پروژه انتخاب شده‌اند:

• Accuracy یا همان دقت:

این معیار ساده‌ترین معیار است که نشان‌دهنده میزان طبقه‌بندی درست داده‌هاست و از تقسیم تعداد پیش‌بینی‌های درست بر کل تعداد پیش‌بینی‌ها به دست می‌آید. این معیار برای کلاس‌های غیرمتوازن مناسب نیست و در مجموع، نمی‌تواند اطلاعات جزئیات خطاها مانند نوع آن‌ها را در اختیار ما بگذارد. این معیار دید کلی خوبی از دقت مدل به ما می‌دهد.

• Confusion matrix:

این ماتریس نشان‌دهنده تعداد پیش‌بینی‌ها و label واقعی به ازای همه‌ی حالات است. در حالت دو کلاسه این ماتریس شامل چهار مقدار است:

True positive:

بیمارانی که واقعا پارکینسون داشته‌اند و درست تشخیص داده‌شده‌اند.

True negative:

بیمارانی که واقعا پارکینسون نداشته‌اند و به درستی سالم تشخیص داده‌شده‌اند.

False positive(False alarm):

بیمارانی که واقعا پارکینسون نداشته‌اند و به اشتباه مبتلا تشخیص داده‌شده‌اند.

False negative:

بیمارانی که واقعا پارکینسون داشته‌اند و به اشتباه سالم تشخیص داده‌شده‌اند.

با بررسی این ماتریس می‌توان نوع دقیق خطا را فهمید. این معیار در مسائل با داده نامتوزان می‌تواند بسیار مفید باشد.

• ROC curve & AUC:

این معیار یا همان منحنی Reciever Operation Characteristic از رسم نرخ True positive بر حسب نرخ False positive به‌دست‌می‌آید. یک معیار مناسب برای این نمودار سطح زیر آن است که با نام AUC(Area Under Curve) شناخته می‌شود. هر چه سطح زیر این نمودار بیشتر باشد و نمودار به خط افقی در بالا نزدیک‌تر باشد، قدرت مدل در تفکیک دو کلاس بیشتر است و مدل بهتر است و دقت بالاتری دارد.

مقایسه AUC به دلیل عدد بودن آن بسیار راحت‌تر است. بالاتر بودن این عدد به این معناست که مدل ما به ازای true positiveهای زیاد false positiveهای کمی دارد و دقت خوبی از خود نشان‌داده‌است.

• F1 Score:

این معیار عددی معیار precision و recall را در دل خود جای داده‌است. در حقیقت این معیار میانگین هارمونیک این دو معیار است.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

- روش‌های پیش‌پردازش و انتخاب ویژگی:

۱. عمل یکه‌سازی یا Normalization به این معنی عوض کردن مقیاس یک فیچر برحسب بازه مورد قبول برای مقادیر آن است. در واقع در یکه‌سازی، مقادیر یک فیچر همه به بازه‌ی 0 تا 1 متناظر می شوند. علت لزوم این امر این است که فیچر های مختلف به خاطر تفاوت در مقیاس، به طور متفاوتی روی طبقه بندی اثر می گذارند و ممکن است بایاس ایجاد کنند. به عنوان مثال فرض کنید قصد در تخمین قیمت خانه ها داشته باشیم و به عنوان فیچر، تعداد اتاق و مساحت خانه برحسب متر مربع را در نظر گرفته باشیم. در این صورت ممکن است مدل در خانه ی 3 اتاقه‌ی 120 متری را به خانه‌ی 2 اتاقه 120 متری مشابه‌تر در نظر بگیرد تا خانه 3 اتاقه 122 متری. (مخصوصا در مدل هایی مثل K¯Nearest Neighbors که به طور خاص از روی فاصله داده ها را طبقه بندی می کنند.) به همین علت، داده‌ها یکه‌سازی می شوند تا به اصطلاح ارزش یکسانی به آن ها داده شود.

البته یک روش دیگر برای حل این موضوع، Standardization است ولی در صورتی استفاده می شوند که بدانیم مثلا داده ما از یک توزیع گوسی حاصل شده است، ولی در این مسئله از آن جا که توزیع داده ها را نمی دانیم چندان استفاده ای ندارد.

برای یکه‌سازی داده ها، از کتابخانه Scikit Learn استفاده می کنیم. یکی از روش های پر استفاده برای یکه‌سازی، استفاده از کلاس MinMaxScaler در این کتابخانه است که همه داده ها را با یک مقیاس یکسان به عددی بین 0 یا 1 متناظر می کند، به طوری که مینیمم داده ی دریافتی برابر 0 و ماکسیمم داده برابر 1 شود. همچنین در این نوع scaler، فاصله بین داده ها حفظ می شود یعنی دو جفت داده هم فاصله پس از scale شدن هم فاصله می‌مانند.

۲. مجموعه داده‌ی موجود در این سوال نامتوازن یا imbalance است، به این معنی که تعداد نمونه‌های موجود در کلاس های مختلف، به طور محسوس متفاوت است. در چنین مواقعی، ممکن است به مشکلاتی از جمله accuracy paradox برخورد کنیم، یعنی ممکن است مدل از accuracy خیلی بالایی برخوردار باشد ولی در واقعیت عملکرد خوبی نداشته باشد. به عنوان مثال، ممکن است همه نمونه ها را در کلاس بزرگتر طبقه بندی کند اما به علت وجود تعداد بسیار بیشتری از چنین نمونه هایی، accuracy بالایی داشته باشد. برای حل چنین مشکلاتی، روش ها مختلفی موجود است که به بررسی برخی از آن ها می پردازیم:

- هنگامی که مجموعه داده نامتوازن داریم و امکان به دست آوردن نمونه های بیشتر از کلاس اقلیت نداریم، یکی از راه حل های موجود استفاده از metric های دیگر (در کنار accuracy) برای سنجش مدل است. به عنوان مثال ماتریس سردرگمی (Confusion Matrix) یک معیار مناسب است که می تواند نشان بدهد دقیقا چه کلاس هایی را به اشتباه به عنوان چه کلاسی طبقه بندی کرده ایم. مثلا اگر تعداد زیادی از کلاس اقلیت را اشتباها به جای کلاس اکثریت در نظر گرفته باشیم، حتی اگر accuracy بالا باشد این مشکل خود را در ماتریس سردرگمی نشان می دهد. به این صورت می توانیم متوجه شویم که طبقه بند به سمت طبقه بندی به عنوان کلاس اکثریت بایاس دارد. از معیار های مورد استفاده دیگر برای تشخیص این موضوع می توان به Recall، Precision و F1 Score اشاره کرد، که همه از روی ماتریس سردرگمی قابل محاسبه هستند. همچنین می توان از رسم نمودار ROC استفاده کرد.
- یک روش دیگر استفاده از resampling است، به این صورت که یا نمونه های کلاس اقلیت را کپی می کنیم تا تعدادشان افزایش یابد، یا از کلاس اکثریت حذف می کنیم. هر یک از این دو روش نقاط قوت و ضعف خود را دارند.
- مشکل مجموعه داده نامتوازن را می توان به جای استفاده از resampling، با generate کردن نمونه های synthetic حل کرد. برای این کار، می توان هر فیچر را به صورت رندوم از کلاس اقلیت sample کرد. مثلا، می شود از Naïve Bayes استفاده کرد تا هر فیچر را به صورت مستقل نمونه برداری کرد. (یک تکنیک مورد استفاده برای تولید نمونه ها، استفاده از Synthetic Minority Oversampling Technique یا SkOTE است.) وقتی نمونه های جدید تولید می کنیم، نمونه ها برعکس resampling متفاوت از نمونه های موجود هستند که بسیار مهم است، ولی در عین حال روابط غیر خطی بین نمونه ها ممکن است حفظ نشود. به عنوان مثال، اگر فیچر ها با هم correlation داشته باشند، نمونه برداری مستقل از هر کدام و کنار هم قرار دادن آن ها ممکن است داده غیر واقعی یا حتی غلط (عضو کلاس اکثریت) تولید کند.
- علاوه بر این روش ها، می توان برای طبقه بندی از الگوریتم هایی بهره برد که در برابر داده نامتوازن بهتر عمل می کنند (مانند درخت تصمیم)، یا تابع هزینه را به گونه ای تغییر داد که پنالتی بیشتری برای طبقه بندی اشتباه کلاس اقلیت داشته باشیم و از بایاس شدن مدل جلوگیری کنیم. با توجه به اینکه طبقه‌بندهای مورد استفاده در این سوال در صورت سوال مشخص شده اند، انجام چنین کاری در اینجا ممکن نیست.

- هنگامی که تفاوت در اندازه کلاس ها بسیار قابل توجه باشد، به عنوان مثال در مسائلی مثل fraud detection، می توان کلا به سوال به صورت anomaly detection نگاه کرد و نه یک مسئله طبقه بندی، که در آن از روش هایی استفاده می شود که به طور خاص با داده های نامتوازن سر و کار دارند. البته اینجا تفاوت اندازه کلاس ها چندان زیاد نیست.

در نهایت، با توجه به اینکه ما قادر به تغییر الگوریتم های طبقه بندی نیستیم (چراکه در صورت سوال مشخص شده اند)، و همچنین نمی خواهیم نمونه جدید تولید کنیم (چراکه تعداد بالایی فیچر داریم که بسیاری از آن ها با هم correlation دارند) یا بایستی از resample کردن داده استفاده کنیم و یا از معیار هایی دیگری به جز accuracy استفاده کنیم. انجام resampling مشکلات جدیدی به همراه دارد، چراکه در صورت undersample کردن کلاس اکثریت (حذف نمونه ها) ریسک از دست رفتن اطلاعات مهم را به همراه دارد که می تواند عمبکر طبقه بند را به علت نبود داده کافی برای train، به شدت دچار افت کند. از طرف دیگر oversample کردن کلاس اقلیت می تواند باعث overfit شدن مدل شود، با توجه به اینکه تعدادی داده دارند چند بار در آموزش تاثیر داده می شوند.

با توجه به این موضوع، ما مشکل نامتوازن بودن مجموعه داده را با چک کردن معیارهای اضافی (به جز accuracy) حل کرده ایم که می توانند نشان دهند که واقعا عملکرد مدل تا چه حد خوب است.

۳. در بخش کاهش ابعاد، از روش های گفته‌شده در صورت سوال استفاده می کنیم تا فیچر های redundant یا correlated را حذف کنیم و به این صورت نه تنها عملکرد مدل را ارتقا دهیم، بلکه زمان اجرا را نیز کاهش دهیم. همچنین با این کار از curse of dimension جلوگیری کرده و با کم کردن فیچر های موجود، از overfitting جلوگیری می کنیم. حال هر یک از روش های ذکر شده را مختصرا بررسی می کنیم.

- LDA یا Linear Discriminant Analysis: در این روش، همانطور که در کلاس بررسی شد، به تولید ترکیبی خطی از فیچر های موجود می پردازیم که بتواند به خوبی کلاس ها را از هم جدا کند. به این منظور مقادیر و بردار های ویژه ی مجموعه داده را به دست آورده و در ماتریس پراکندگی درون کلاسی و بین کلاسی قرار می دهیم. سپس بردار های ویژه ای را در نظر می گیریم که مقدار ویژه متناظرشان از بقیه بزرگتر باشد، چراکه این ها حامل اطلاعات بیشتری در مورد پراکندگی داده هستند.

در پیاده سازی این روش، از کلاس LinearDiscriminantAnalysis در کتابخانه Scikit Learn استفاده کردیم، که آن را روی داده های train آموزش دادیم و سپس با فراخوانی متد transform، هر دو بخش train و test را کاهش بعد دادیم.

- ICA یا Independent Component Analysis: در استفاده از ICA، فرض می شود که هر داده مخلوطی از کامپوننت های مستقل از هم است و تلاش می کند این کامپوننت ها را بیابد. به این منظور، فرض می کنیم که این کامپوننت ها از تعدادی source آمده اند، که سیگنال های هر یک مستقل از دیگری است و توزیع غیر گوسی دارد. سپس فرض می شود که روی این سیگنال ها یک ترکیب خطی انجام شده است و سعی در بازیابی سیگنال های اصلی داریم. با توجه به افزایش دقت با افزایش تعداد ویژگی‌ها تعداد ویژگی‌ها در حد مقدار بهینه PCA یعنی ۸۰ ویژگی درنظر گرفته‌شد.

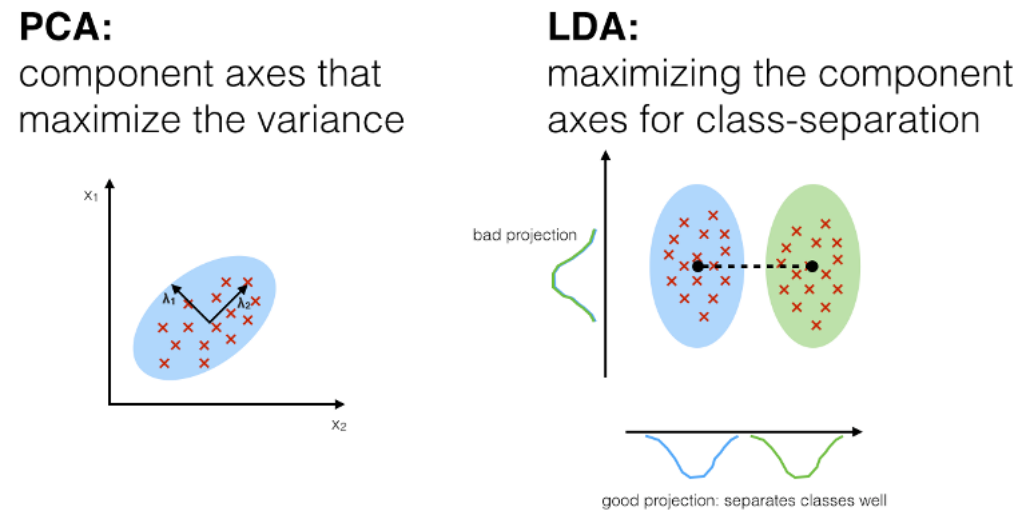
در پیاده سازی این روش، از کلاس FastICA کتابخانه ی Scikit Learn استفاده می کنیم، که یک پیاده سازی سریع از A. Hyvarinen and E. Oja، Independent Component Analysis: Algorithms and Applications، Neural Networks، 13(4–5)، 2000، pp. 411–430 است.

- PCA بدون Whitening: در PCA یا Principal Component Analysis داده را روی یک تعداد محور عمود بر هم project می کنیم، با این هدف که واریانس داده ها روی این محور ها بزرگ باشد. این محور های عمود بر هم یا principal components نامیده می شوند. اولین محور در نظر گرفته شده بزرگترین واریانس ممکن را دارد، و باقی محور ها هم بزرگترین واریانس ممکن به شرط عمود بودن محور ها را دارند. به طور کلی، PCA به شدت به مقیاس فیچر ها حساس است و باید حتما فیچر ها یکه‌سازی شده باشند.

برای حل، ابتدا ماتریس کواریانس فیچر ها را در نظر گرفته و آن را به مقادیر و بردار های ویژه decompose می کنیم. سپس بردار های ویژه ای را انتخاب می کنیم که بزرگترین مقدار ویژه ها را دارند.

برخلاف LDA، این روش ممکن است در جداسازی کلاس ها خوب عمل نکند. در حالت کلی، PCA در شرایطی خوب عمل می کند که فیچر ها یک رابطه خطی بین خود دارند.

برای پیاده سازی، از کلاس PCA کتابخانه Scikit Learn استفاده می کنیم، که از روش پیاده سازی Lapak، Halko et al. 2009 استفاده می کند.

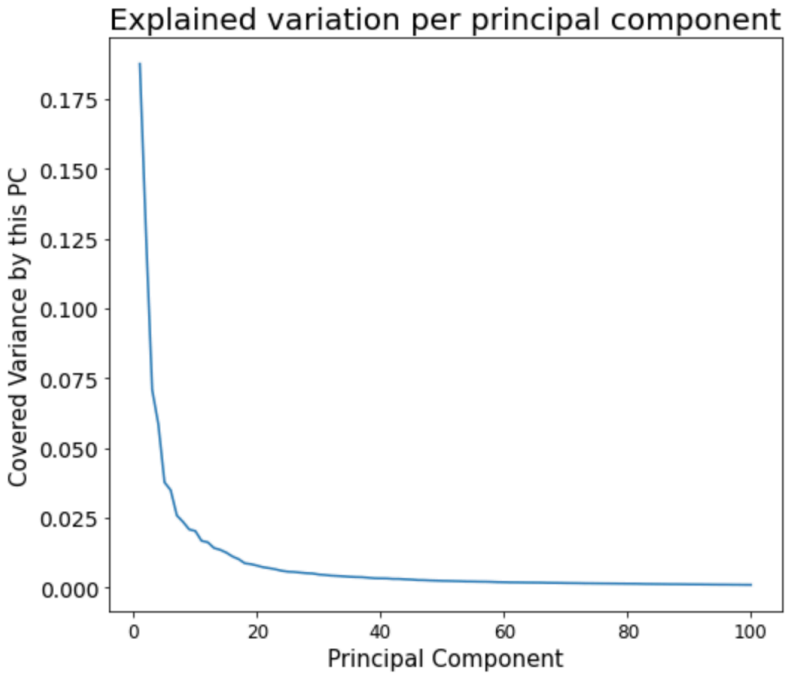


تفاوت LDA و PCA

نکته مهم در استفاده از این روش، توجه به میزان واریانس روی هر بعد است. همانطور که نمودار زیر دیده می شود، برای اضافه شدن هر principal component میزان explained variance رسم شده است و این مقدار بیشتری تاثیر خود را پس از یک بعد دارد، یعنی بعد اول واریانس بسیاری را در خود نشان داده است و باقی ابعاد چندان تاثیر گذار نبوده اند. به همین دلیل تنها از 80 بعد استفاده کرده ایم و کاهش بعد نهایی از 753 بعد به 80 بعد انجام شده است.

با تقلیل ابعاد به 80 بعد، به 0.9019115414677887 برای explained variance ratio رسیده ایم.

```
<Figure size 432x288 with 0 Axes>
```




```
Explained variation per principal component for the first ten PCs: [0.18756015 0.12677269 0.07084639 0.05847021 0.0376564
 0.02578891 0.02348846 0.02078483 0.02020834 0.01669565 0.01620448
 0.01416225 0.01349809 0.01249936 0.01111477 0.01015756 0.00867646
 0.00836132 0.00781394]
0.9019115414677887
```

نمودار میزان توضیح‌داده‌شدن واریانس توسط ۱۰۰
کامپوننت اول به همراه میزان واریانس ۸۰ کامپوننت اول

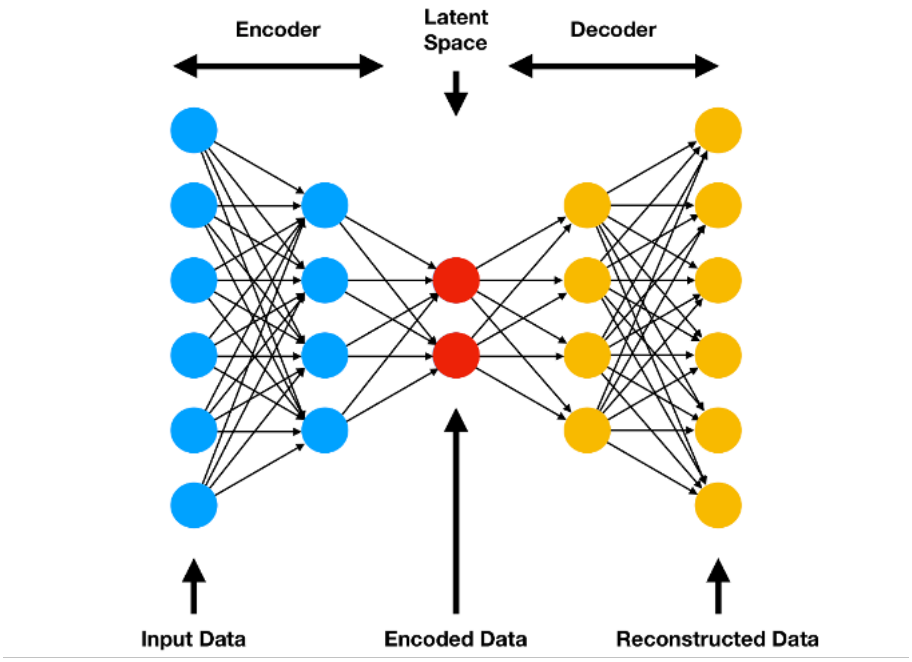
- PCA با Whitening: این روش مشابه روش قبل است و کماکان از همان پیاده سازی استفاده می کنیم، ولی عملیات whitening هم به آن اضافه شده است (که PCA کتابخانه Scikit Learn آن را پشتیبانی می کند). هدف در whitening این است که فیچر های موجود کمتر redundant باشند و correlation کمتری با هم داشته باشند. به این منظور، بردار ها در رادیکال تعداد نمونه ها ضرب شده و سپس بر singular value ها تقسیم می شوند تا فیچر های uncorrelated با واریانس واحد حاصل شود.
- Sequential Backward Feature Elimination: همانطور که در کلاس بررسی شد، در این روش ابتدا از تمامی فیچر های موجود استفاده شده و طبقه بندی انجام می شود و درستی آن سنجیده می شود. سپس امتحان می کنیم که کدام یک از فیچرها در صورت حذف بیشتر باعث بهتر شدن عملکرد طبقه بند می شود. آن فیچر را حذف کرده و مجددا پروسه را روی فیچر های باقی مانده تکرار می کنیم.

برای پیاده سازی این الگوریتم، از کلاس RFE از کتابخانه Scikit Learn استفاده می کنیم، که با دریافت یک مدل، یک به یک فیچر هایی که بیشتر باعث بهتر شدن عملکرد می شوند را حذف می کند.

همچنین، در این پیاده سازی به عنوان مثال از مدل SVR برای طبقه‌بندی روی فیچرها استفاده می کنیم تا بتوانیم دقت مدل با ویژگی‌های مانده را برای مقایسه مدل‌ها مختلف به‌دست‌آوریم.

- Autoencoder: اتوانکودر یا خودرمزنگارها، نوعی از شبکه های عصبی هستند که برای encode کردن داده و حتی ساخت داده جدید نیز استفاده می شوند. ساختار کلی این شبکه ها به این صورت است که در ورودی تمام فیچر ها را در نظر می گیریم و در خروجی نیز انتظار داریم تمام خروجی‌ها ساخته شوند، و در لایه های دیگر شبکه، تعداد کمتر یا بیشتری فیچر راس داریم. به این صورت در واقع شبکه آموزش می یابد تا داده اولیه را از روی هر لایه ی دلخواهی با داشتن وزن ها بسازد. حال در صورتی که در میانه‌ی شبکه، یک لایه داشته باشیم که تعداد کمتری راس داشته باشد، شبکه می تواند پس از آموزش دیدن و با داشتن وزن ها، وردی را داده و آن را با ابعاد کمتر از لایه میانی دریافت کند، یا از لایه میانی شروع کند و ورودی را دوباره بسازد.

از طرفی چون شبکه سعی دارد خطای بازیابی داده را کمینه کند، بعد میانی که داده encode شده را داراست، شامل اطلاعات کافی برای ساخت فیچر ها با خطای کم است.

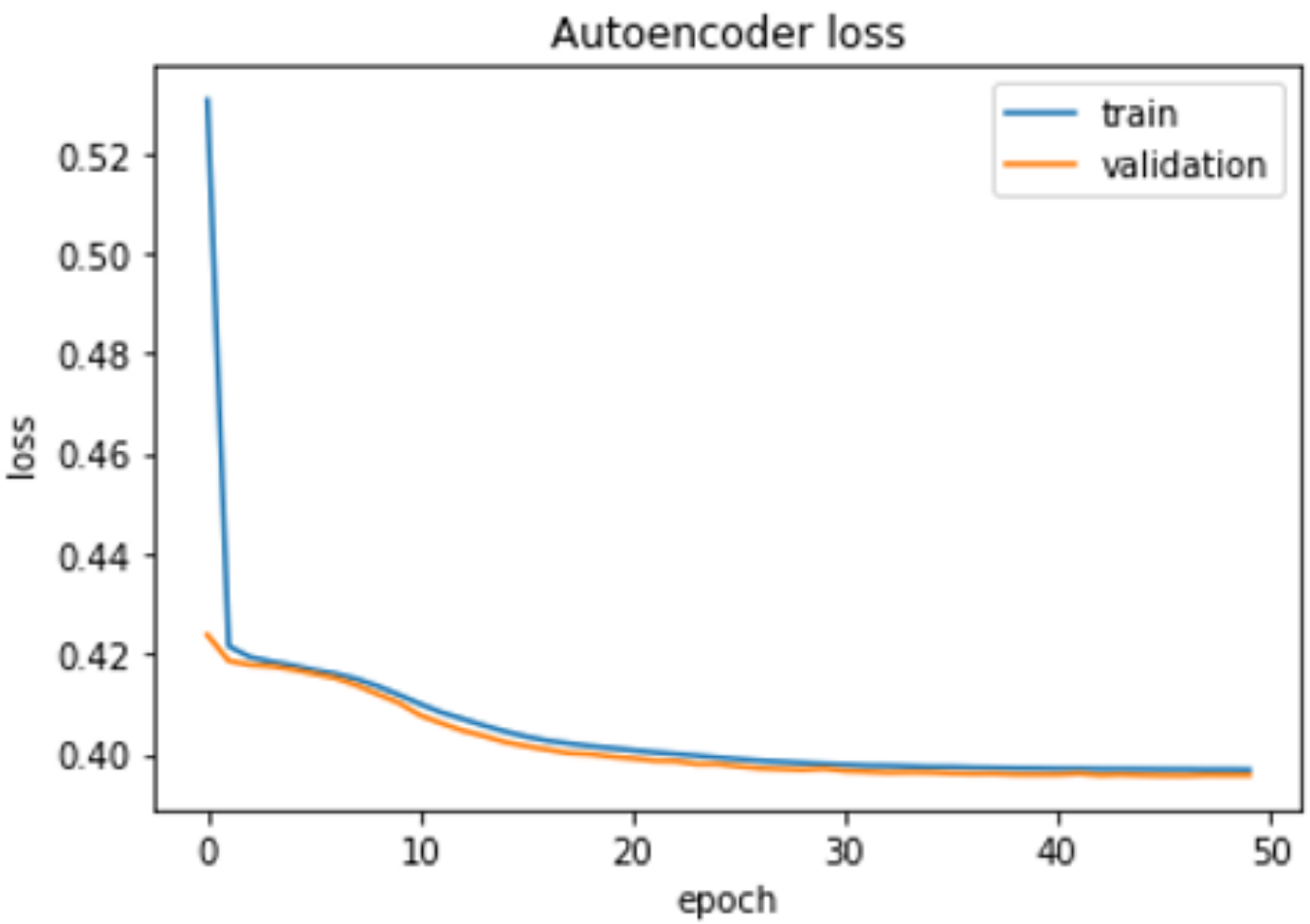


تصویر شماتیک AutoEncoder

برای پیاده سازی این شبکه، از مدل های کتابخانه keras برای تعریف و آموزش شبکه عصبی استفاده می کنیم. در ابتدا، یک لایه Input در نظر گرفته، و سپس از یک لایه ی Dense برای تبدیل به داده ی Code شده استفاده می کنیم. (لایه Dense شامل یال از همه راس ها یه لایه قبل به همه ی راس های این لایه می باشد.) می توان تعداد لایه بیشتری را نیز در نظر گرفت و یک autoencoder عمیق ساخت. سپس مجددا یک لایه Dense دیگر برای عملیات decode تعریف می کنیم و شبکه را روی داده های مجموعه train آموزش می دهیم.

برای آموزش شبکه از بهینه ساز Adam استفاده می کنیم و 10 درصد داده را برای validation مشخص می کنیم. سپس داده را برای 50 epoch آموزش می دهیم. با آزمون و خطا کردن، یعنی افزودن لایه میانی و تغییر batch size و دیگر پارامتر ها، در نهایت بهترین حالت را در نظر گرفتیم.

در ادامه نمودار میزان خطای train و validation را برای شبکه در طول 50 epoch آورده ایم که همانطور که دیده می شود خطای train و validation در نهایت بسیار نزدیک است (در صورت فاصله زیاد این دو، دچار overfit هستیم). همچنین، بعد از حدود 35 epoch، دیگر کاهش خاصی در مقدار loss نداشته ایم که نشان می دهد احتمالا مدل به خوبی fit شده و ادامه دادن فرآیند آموزش احتمالا تاثیر زیادی در میزان خطای نهایی ندارد.



میزان loss مدل به ازای epochهای
مختلف روی داده‌ی آموزش و ارزیابی

• طبقه بندی Generative:

مدل‌های generative مدل‌هایی هستند که تلاش می کنند تابع توزیع چگالی احتمال داده را برای هر کلاس به دست آورند و سپس از روی آن تابع، احتمال شرطی تعلق یک داده به یک کلاس را محاسبه کنند. احتمال شرطی با یک marginalization ساده و استفاده از قاعده بیز از روی احتمال توام قابل محاسبه است، و خود احتمال توام داده بسیار بیشتری از احتمال شرطی با خود دارد، اما دقیقا به همین علت برای محاسبه آن به داده بیشتری نیاز داریم، چراکه برای محاسبه درست احتمال توام باید به حد کافی از همه حالت ها نمونه دیده باشیم تا بتوانیم از روی آن ها به چگالی احتمال پی ببریم.

یکی از خوبی های این مدل ها این است که چون توزیع چگالی احتمال را تمام و کمال تخمین می زنند، می توانند برای تولید داده جدید نیز استفاده شوند و به همین دلیل generative نام دارند.

در این بخش چند روش طبقه‌بندی Generative که در صورت سوال خواسته شده را پیاده سازی می کنیم.

- طبقه بندی با پنجره پارزن: در پنجره پارزن به این صورت عمل می شود که سعی در تخمین چگالی احتمال برای هر ورودی دلخواه داریم. در این روش با فرض اینکه توزیع داده ها را نمی دانیم، برای تخمین به نقاط نزدیک به نقطه هدف و فاصله آن ها نگاه می کنیم و برحسب این فاصله میزان اثر هر نقطه در جواب را مشخص می کنیم.

$$P(\omega_i \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid w_i) \cdot P(\omega_i)}{p(\boldsymbol{x})}$$

$$\Rightarrow \textit{posterior probability} = \frac{\textit{likelihood} \cdot \textit{prior probability}}{\textit{evidence}}$$

قاعده‌ی بیز

حال در صورتی که روی هر یک از کلاس ها یک پنجره پارزن تعریف کنیم، با استفاده از قانون بیز، می توان یک طبقه بند بهینه بیز تعریف کرد.

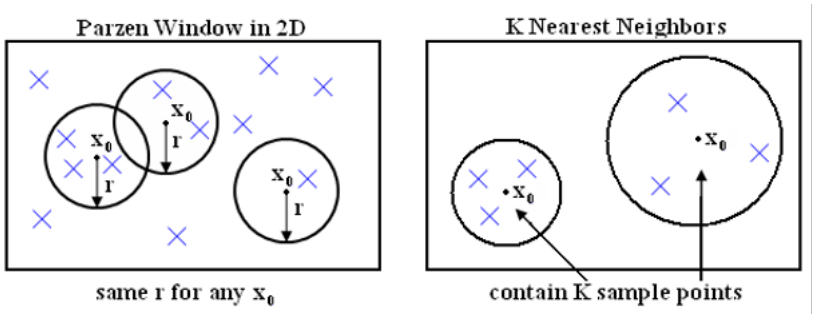
برای پیاده سازی این طبقه بند، از کلاس KernelDensity کتابخانه Scikit Learn استفاده می کنیم. این کلاس توانایی تخمین چگالی احتمال برای داده جدید را از روی داده های قبلی دارد. همچنین به صورت پیشفرض از تابع گوسی به عنوان پنجره استفاده می کند. (وزن داده شده به هر sample نسب به فاصله sample هدف از آن به صورت گوسی است.) سپس کافی است دو instance مختلف از این مدل تعریف کنیم که هر یک روی داده های یکی از کلاس ها آموزش ببیند و تخمین چگالی آن کلاس را برای sample جدید ارائه دهد.

حال می توان از قاعده بیز استفاده کرد تا برای هر sample جدید، با استفاده از احتمال پسین آن روی هر یک از کلاس ها، تخمین زد که این داده مربوط به کدام کلاس است.

همچنین برای محاسبه احتمال پسین، نیاز به داشتن احتمال پیشین داریم ولی در اینجا آن را نداریم. می توان آن را از روی sample های دسته train محاسبه کرد یا آن را برای هر دو کلاس مساوی در نظر گرفت. البته در حالت کلی احتمال داشتن و نداشتن این بیماری مساوی نیست و منطقا اکثر افراد سالم هستند ولی با توجه به اینکه هدف این مدل را نمی دانیم نمی توان با اطمینان نظر داد. به عنوان مثال ممکن است این مدل تنها برای افراد مشکوک به بیماری استفاده شود که واقعا نیمی از آن ها بیمار باشند. با توجه به اینکه این اطلاعات را نداریم از همان احتمال پیشین داده های train استفاده کردیم، که حدود 0.74 احتمال بیمار نبودن است.

- طبقه بندی با روش knn: در روش knn ایده همانند پارزن تخمین چگالی احتمال در هر نقطه دلخواه و سپس استفاده از طبقه بند بهینه بیزی است، اما به جای اینکه پنجره

مشخصی با اندازه های ثابت داشته باشیم، همیشه اندازه پنجره را به گونه ای تعریف می کنیم که تعداد ثابتی از همسایه های یک نقطه در آن قرار بگیرند.



تفاوت روش پارزن با knn

برای پیاده سازی این روش، کافی است از مدل kneighborsClassifier از کتابخانه Scikit Learn استفاده کنیم، که خود با استفاده از تخمین چگالی از طریق الگوریتم knn، عمل طبقه بندی را نیز انجام می دهد. همچنین برای اندازه مقدار k، با آزمون خطا مقادیر مختلفی را امتحان کردیم ولی در نهایت از k=3 نتایج مناسبی دریافت کردیم.

- طبقه بندی با استفاده از gmm: در طبقه بند gmm، هر کلاس را به صورت یک ترکیب از تعدادی توزیع گوسی (Multivariate) در نظر می گیریم. برای به دست آوردن این توزیع ها، پارامتر های آن ها یعنی وکتور میانگین و ماتریس واریانس هر یک را از روش EM تخمین می زنیم و سعی می کنیم به ماکسیمم likelihood برسیم، به این معنی که توزیعی را به دست بیاوریم که احتمال به دست آمدن این نمونه ها از آن، بیشینه باشد. سپس می توان به ازای هر داده جدید، با استفاده از این تخمین ها، چگالی احتمال متعلق به هر کلاس را به دست آورد. در نهایت با مقایسه این احتمال ها از طریق فرمول بیز، می توان کلاسی را انتخاب کرد که احتمال تعلق داده جدید به آن، بیشتر باشد.

برای پیاده سازی آن، از مدل GaussianMixture از کتابخانه ScikitLearn استفاده می کنیم و برای هر کلاس یک مدل با تعداد کامپوننت مشخص آموزش می دهیم. (پس از آزمون و خطا، بهترین نتیجه از قرار دادن 10 کامپوننت برای هر Gaussian Mixture به دست آمد.) در نهایت نیز مشابه قبل چگالی احتمال هر کلاس را برای هر نمونه از داده تست به دست آورده و از روی آن طبقه بندی را انجام می دهیم.

- طبقه‌بندی Discriminative:

مدل‌های generative را پیش تر توضیح دادیم و گفتیم این مدل ها احتمال توام (توزیع چگالی احتمال هر کلاس برای هر نقطه) را محاسبه می کنند و از روی آن احتمال شرطی را به دست می آورند. به خاطر همین موضوع، داده بسیار زیادی برای محاسبه آن ها لازم است. در طرف دیگر، مدل های discriminative اما خود احتمال شرطی را مستقیما تخمین می زنند و در واقع مرز بین کلاس ها را به دست می آورند. این مرز حتی می تواند به صورت نرم باشد و اجازه خطا بدهد.

این موضوع که احتمال شرطی را مستقیما محاسبه می کنیم باعث می شود که در مقایسه با روش های discriminative، به اطلاعات بسیار کمتری برای تخمین نیاز داشته باشند و مخصوصا وقتی مجموعه داده کوچکی داریم، مثل همین سوال، استفاده از آن ها بسیار مناسب است. در مقایسه روش ها نیز به خوبی دیده می شود که روش های discriminative به نتایج خوبی دست یافته اند. در نتیجه گیری نهایی نیز مدل discriminative را انتخاب می کنیم.

همچنین نقطه قوت مدل های generative در تولید داده جدید است که در این مسئله به چنین چیزی نیاز نداریم.

پیاده سازی چند مدل discriminative در صورت سوال خواسته شده است که توضیح هر یک در ادامه آمده است.

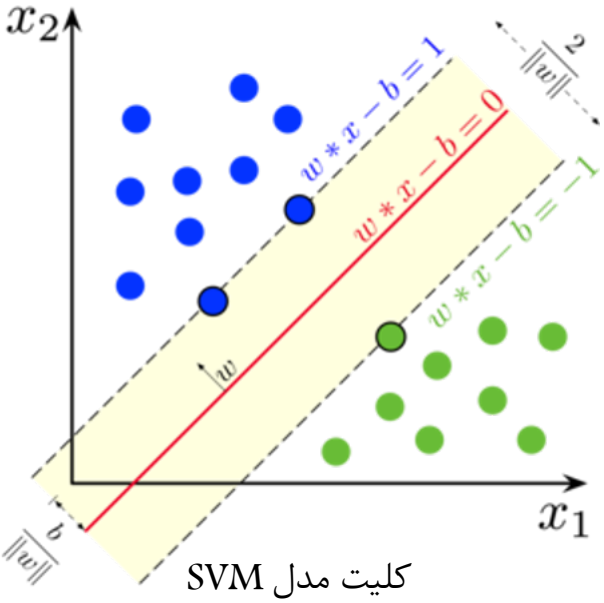
- Logistic Regression: روش Logistic Regression نوعی آنالیز regression است که در مواقعی که خروجی باینری است استفاده می شود. این مدل یک مدل خطی برای classification است که در آن احتمال یک اتفاق به صورت یک تابع logistic مدل می شود.

برای پیاده سازی آن، از کتابخانه Scikit Learn استفاده کردیم که ابتدا یک regularization انجام می دهد و سپس برحسب اینکه چه نوع regularization ای انجام داده، تابع هزینه را تعریف می کند. مثلا در حالت default، عمل regularization انجام شده l2 است و تابع هزینه آن مطابق زیر به دست می آید:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

تابع هزینه

- SVM یا support¯vector machine: یک مدل یادگیری supervised است که یک یا مجموعه ای از ابرصفحه ها می سازد که برای طبقه بندی یا regression استفاده می شوند. این مدل با این فرض طبقه بندی را انجام می دهد که در یک طبقه بندی مناسب، ابرصفحه بیشترین فاصله را از نزدیکترین نقاط datapoint به آن دارد (که support¯ vector نامیده می شوند). به این منظور، یک margin تعریف کرده که فاصله ای است که از اطراف ابرصفحه خالی است، و سعی در بیشینه کردن آن دارد. همانطور که در کلاس مطرح شد، این مدل همچنین امکان تعریف یک C دارد که برای بزرگتر کردن margin، اجازه تعداد محدودی خطا در طبقه بندی یا ورود datapoint ها به margin را بدهد و از این طریق تلاش دارد مانع overfit شدن به نویز ها شود.



کلیت مدل SVM

برای پیاده سازی این روش، از کلاس SVC از کتابخانه Scikit Learn استفاده می کنیم. هنگام fit کردن، ابرصفحه جدا کننده را به دست می آورد (که در واقع ضرایب سازنده ابرصفحه در فیلد coefficients قرار دارد). همچنین زمان اجرای فرآیند یادگیری با رابطه توان 2 با بالارفتن تعداد نمونه ها رابطه دارد.

- Decision Tree: همانطور که در کلاس صحبت شد، مدل درخت تصمیم به این صورت است که هر بار راس پدر برحسب یک خصوصیت به فرزندان شکسته می شود. با تکرار این عمل یک درخت به دست می آید که هر نمونه در یکی از برگ ها قرار می گیرد. این عمل تا جایی ادامه می یابد تا به تعدادی برگ برسیم و برای برگ یک برچسب تعیین کنیم. سپس در طبقه بندی یک نمونه جدید، برحسب اینکه این نمونه در کدام برگ قرار می گیرد، طبقه

آن نمونه را پیش بینی می کنیم. همچنین برای پیدا کردن خصوصیت مناسب برای شکستن راس پدر، راه های مختلفی موجود است که مثلا تصمیم گیری برحسب مقایسه آنتروپی اطلاعات حاصل از شکستن راس بر حسب هر خصوصیت، یک روش پر استفاده است.

برای پیاده سازی، از کلاس DecisionTreeCalssifier از کتابخانه Scikit Learn استفاده کردیم. در حالت دیفالت، این درخت تصمیم تا بیشترین عمق ادامه می دهد، و درخت هیچ هرسی ندارد، اما در صورت استفاده از هرش میزان مصرف حافظه کاهش می یابد. ما پس از آزمون و خطای چند حالت، از حالت بدون هرس استفاده کردیم.

- knn: روش knn را پیش تر توضیح دادیم.
- mlp یا Multilayer Perceptron: در این روش، یک شبکه چند لایه داریم (یعنی علاوه بر لایه های ورودی و خروجی، حداقل یک لایه پنهان داریم) که هر راس آن یک پرسپترون است. پرسپترون نورونی است که با دریافت ورودی ها و ضرب آن در وزن های موجود، در نهایت با استفاده از تابع activation، این مقدار به خروجی تبدیل می شود.

ما برای پیاده سازی mlp از MlpClassifier در کتابخانه Scikit Learn استفاده کردیم. به عنوان تابع activation از relu و برای solver از adam استفاده کردیم. تابع relu، مخفف Rectified Linear Unit است که در واقع همان مقدار max(0، x) را به ازای ورودی x بر می گرداند. بهینه ساز adam نیز همان stochastic gradient based optimizer است که برای آپدیت کردن وزن ها به کار می رود. برای آموزش تا 3000 iteration ادامه دادیم و در نهایت به نتایج قابل توجهی دست یافتیم که در ادامه به آن ها اشاره شده و با دیگر نتایج مقایسه شده است.

- RBF یا Radial Basis Function: این روش به این صورت است که یک تابع تعریف می کنیم که مقدار آن به فاصله نقطه ورودی از یک نقطه ثابت بستگی دارد. این تابع می توان به شکل های مختلفی تعریف شود، مثلا فاصله گوسی یا فاصله منهتنی یا غیره.

برای پیاده سازی روش RBF، از کلاس RBF کتابخانه Scikit Learn استفاده کردیم. کرنل RBF در این کتابخانه به صورت زیر تعریف می شود:

که مقدار l در آن، اندازه مقیاس است که مقداری مثبت است، و می تواند یک اسکالر یا یک وکتور هم اندازه ورودی باشد. این کرنل بی نهایت مشتق پذیر است، که نشان می دهد انحام

Gaussian Process با این کرنل به عنوان تابع کواریانس از همه مرتبه ها mean square derivation دارد و در نتیجه بسیار نرم است.

این تابع را با length scale = 1 اجرا کردیم (البته دقت کنید که در ابتدا فیچر ها را یکه کرده بودیم.)

- **مقایسه روش‌ها:**

در مقایسه روش های مذکور، ابتدا هر یک از آن ها را به ازای هر یک از روش های کاهش ابعاد اجرا کرده و سپس معیارهای اندازه گیری را در هر یک به دست آوردیم. همانطور که گفته شد، معیار های ارزیابی مورد بررسی accuracy، F1 score و AUC هستند. همچنین ROC را به ازای هر یک رسم کردیم تا دید بهتری دریافت کنیم. (نمودارهای ROC برای طولانی نشدن گزارش و بودن AUC به عنوان نماینده در گزارش آورده نشده‌اند ولی در notebook مربوط به کدها قابل مشاهده‌اند.)

در جداول زیر، این معیار ها به ازای هر یک از روش های کاهش ابعاد و طبقه‌بندی آمده‌اند:

جدول Accuracy مدل‌های مختلف به ازای
روش‌های کاهش بعد مختلف

Accuracy Table	Parzen	KNN	GMM	Logistic Regression	SVM	Decision Tree	MLP	RBF
LDA	0.5921	0.5921	0.4473	0.5921	0.5526	0.5921	0.5394	0.5921
ICA	0.7236	0.8684	0.7631	0.7236	0.8026	0.75	0.8289	0.8026
PCA with Whitening	0.8815	0.9210	0.7631	0.7236	0.8026	0.7105	0.8421	0.8026
PCA without Whitening	0.8815	0.9210	0.7631	0.7236	0.8026	0.7105	0.8421	0.8026
SBF	0.7236	0.7368	0.8157	0.7631	0.7763	0.6973	0.7763	0.7894
Autoencoder	0.8815	0.8552	0.75	0.8026	0.7894	0.75	0.7894	0.7236

جدول AUC مدل‌های مختلف به ازای
روش‌های کاهش بعد مختلف

AUC Table	Parzen	KNN	GMM	Logistic Regression	SVM	Decision Tree	MLP	RBF
LDA	0.5415	0.5415	0.5004	0.6173	0.4259	0.5415	0.6164	0.4891
ICA	0.5000	0.9129	0.76277	0.8164	0.6870	0.7242	0.8943	0.8744
PCA with Whitening	0.8593	0.9424	0.5861	0.7974	0.6261	0.5792	0.9194	0.8510
PCA without Whitening	0.8593	0.9424	0.5861	0.7974	0.6870	0.5792	0.9194	0.8510
SBF	0.5000	0.7376	0.7255	0.7480	0.6099	0.6142	0.7186	0.7515
Autoencoder	0.8298	0.9285	0.7389	0.7662	0.6632	0.7095	0.8701	0.7480

جدول F1 Score مدل‌های مختلف به ازای
روش‌های کاهش بعد مختلف

F1 Score Table	Parzen	KNN	GMM	Logistic Regression	SVM	Decision Tree	MLP	RBF
LDA	0.6990	0.6990	0.5000	0.6990	0.6964	0.6990	0.6236	0.6990
ICA	0.8396	0.9107	0.8235	0.8396	0.8739	0.8190	0.8849	0.8739
PCA with Whitening	0.9174	0.9454	0.8571	0.8173	0.8739	0.8135	0.8947	0.8739
PCA without Whitening	0.9174	0.9454	0.8571	0.8173	0.8739	0.8135	0.8947	0.8739
SBF	0.8396	0.8245	0.8793	0.8548	0.8640	0.7927	0.8617	0.8666
Autoencoder	0.9203	0.8990	0.8155	0.8717	0.8666	0.8224	0.8620	0.8396

جدول Confusion Matrix مدل‌های
مختلف به ازای روش‌های کاهش بعد مختلف

Confusion Matrix	Parzen		KNN		GMM		Logistic Regression		SVM		Decision Tree		MLP		RBF	
LDA	9	12	9	12	13	8	9	12	3	18	9	12	12	9	9	12
	19	36	19	36	34	21	19	36	16	39	19	36	26	29	19	36
ICA	0	21	15	6	16	5	0	21	9	12	14	7	13	8	9	12
	0	55	4	51	13	42	0	55	3	52	12	43	5	50	3	52
PCA with Whiten	17	4	18	3	4	17	8	13	9	12	6	15	13	8	9	12
	5	50	3	52	1	54	8	47	3	52	7	48	4	51	3	52
PCA without Whitening	17	4	18	3	4	17	8	13	9	12	6	15	13	8	9	12
	5	50	3	52	1	54	8	47	3	52	7	48	4	51	3	52
SBF	0	21	9	12	11	10	5	16	5	16	9	12	6	15	8	13
	0	55	8	47	4	51	2	53	1	54	11	44	2	53	3	52
Autoencoder	15	6	16	5	15	6	10	11	8	13	13	8	10	11	0	21
	3	52	6	49	13	42	4	51	3	52	11	44	5	50	0	55

همانطور که میبینید، در خیلی از موارد accuracy معیار دقیقی نیست، که یکی از علت های این امر همانطور که ذکر شد نامتوازن بودن کلاس هاست. به عنوان مثال، برای روش Parzen با کاهش ابعاد 72 accuracy، ICA است که چندان بد به نظر نمی آید ولی با نگاه به ماتریس آشفتگی متوجه می شویم که در واقع همیشه تخمین سالم بودن داشته و اصلا عملکرد خوبی نداشته. این موضوع همچنین در سایر متریک های نوشته شده نیز دیده می شود؛ مثلا F1 score از روش های دیگر مثل PCA کمتر هستند و همچنین شاخص AUC بسیار کم و برابر 0.5 است.

همچنین در موارد بسیاری میزان accuracy چند روش یکسان است ولی دیدن باقی شاخص ها دید بسیار بهتری از عملکرد این روش ها و برتری آن ها بر یکدیگر به ما می دهد.

نکته قابل توجه دیگر اینکه Whitening داشتن و نداشتن تفاوت چندانی در این تعداد بعد ایجاد نکرده است و معیار های مورد استفاده نشان از عملکرد مشابه این دو کاهش بعد دارند.

همچنین LDA عملکرد چندان خوبی از خود بروز نداده است. علت می تواند این باشد که هنگامی که روش LDA که تلاش می کند داده ها را روی یک بعد نشان دهد (به شرط اینکه تلاش بر این باشد که داده های بین کلاسی به خوبی از هم جدا شوند و درون کلاسی ها

نزدیک باشند) احتمالا یک بعد کافی نبوده و تناظر داده ها به روی یک خط اطلاعات مهمی را حذف می کند که برای دسته بندی درست آن ها نهایی است.

بهترین روش انتخابی ما، روش knn با استفاده از یکی از روش های PCA برای کاهش بعد است. این روش از چند نظر بر سایر روش ها برتری دارد که در تحلیل معیار ها مشخص می شود. اولین موضوع قابل توجه در مورد این روش، میزان accuracy به نسبت بالای آن است. در این روش accuracy، حدود 92 درصد است که بالاترین accuracy به دست آمده است. (هرچند در چند روش دیگر نیز accuracy نزدیکی حاصل شده است.) اما، همانطور که گفته شد accuracy به تنهایی معیار گویایی نیست و هر چند در یک بررسی اولیه می تواند کمک کننده باشد، در مقایسه روش ها می تواند ما را دچار اشتباه کند (مخصوصا در شرایطی مثل این مسئله که داده ها نامتوازن هستند و امکان وجود بایاس به سمت کلاس اکثریت است).

دومین علت این تصمیم، در توجه به معیار AUC مشخص می‌شود. همانطور که در جدول مشخص است، این روش در میان تمام روش های طبقه بندی و کاهش بعد، بیشترین شاخص AUC را دارد. همانطور که پیش تر توضیح دادیم، این معیار می تواند نشان دهنده جداپذیری کلاس ها باشد، چراکه هر چه کلاس ها جدا پذیری بهتری داشته باشند، میزان خمیدگی نمودار ROC افزایش یافته که در نتیجه آن افزایش مساحت زیر نمودار یا همان AUC را داریم و این مقدار به 1 نزدیک تر خواهد بود. در روش knn با کاهش بعد PCA، مقدار چشمگیر 0.9424 را شاهد هستیم که نشان می دهد در این روش کلاس ها تا حد خوبی از یکدیگر جدا شده‌اند.

در بررسی شاخص F1 score، در knn با کاهش بعد PCA، مقدار این شاخص از تمام روش‌ها بیشتر است؛ همچنین مشاهده می شود که در روش های parzen و mlp، مقادیر مشابهی را شاهد هستیم. این موضوع نشان می دهد که روش های parzen و mlp هنگامی که با کاهش بعد های PCA with/without whitening و ICA داده را آماده سازی می کنیم، نتایج خوبی از خود نشان می دهند. اما این معیار خود تابعی از Confusion Matrix است و با دقت در خود Confusion Matrix علت سوم انتخاب این روش را می توان دید.

در Confusion Matrix های به دست آمده، موضوع مهمی که دیده می شود کم بودن تشخیص صحیح فرد بیمار است. این موضوع در این سوال به صورت خاص اهمیت زیادی دارد؛ چراکه در تشخیص بیماری در صورت اینکه فرد سالم به اشتباه بیمار تشخیص داده شود، تنها هزینه ای که داده می شود احتمالا هزینه بررسی بیشتر پزشک متخصص است، اما از طرف دیگر در صورت تشخیص اشتباع فرد بیمار به عنوان سالم، این فرد احتمالا بررسی

بیشتری برای اطمینان از سلامت خود انجام نمی دهد و این اعتماد او باعث تشخیص دیرهنگام بیماری شده که در نهایت تاثیر منفی به سزایی در فرآیند درمان و کیفیت زندگی فرد دارد.

حال با دقت در این Confusion Matrix ها دیده می شود که روش knn با کاهش بعد PCA، نه تنها به بهترین accuracy و AUC و همچنین F1 score به نسبت بالایی دست یافته است، بلکه کمترین تشخیص اشتباه فرد بیمار به عنوان سالم را دارد و برعکس بقیه روش ها که به خاطر نامتوازن بودن کلاس ها، به سمت تشخیص به عنوان کلاس اکثریت متمایل شده اند، این روش تا حد خوبی تعادل را حفظ کرده و با وجود اینکه تشخیص اشتباه در هر دو طرف کم دارد، ولی کلاس اقلیت را بهتر از بقیه روش ها درست تشخیص داده است.

برای مثال، روش SVM با PCA، با وجود accuracy به نسبت قابل قبول و اینکه تنها 3 فرد سالم را بیمار در نظر گرفته است، ولی خطای زیادی در تشخیص فرد بیمار به عنوان سالم دارد 12 نفر از 21 نفر بیمار را سالم تشخیص داده است و عملا این مدل کاربردی ندارد. از طرف دیگر، روش انتخابی ما یعنی knn با PCA، تنها 3 نفر بیمار از 21 نفر را به اشتباه تشخیص داده است.

• اجرای kfold cross validation برای مدل نهایی:

این کار با استفاده از تابع k_fold_cross_validation انجام می‌شود که با گرفتن یک روش کاهش بعد و یک طبقه‌بند، آن را با ۵ fold ارزیابی می‌کند. به این صورت که داده را به پنج قسمت تقسیم می‌کند و سپس، به ازای هر ترکیب چهارتایی مدل را آموزش می‌دهد و به ازای دسته‌ی آخر می‌آزماید. در نهایت، میانگین دقت‌ها به همراه واریانس آن‌ها گزارش می‌شود که از کم بودن واریانس نیز مطمئن شویم.(نبود خطای واریانس) در این جا واریانس قابل قبول است. این نوع گزارش خطا داده کامل از خطای مدل و قدرت تعمیم‌دهی آن به ما می‌دهد.

```
Accuracies Mean: 0.8928720808644126
Accuracies std: 0.023764576186638988
```

میانگین و واریانس دقت مدل knn
حاصل از 5−fold cross validation

• روش‌های یادگیری تجمیعی:

- اهمیت یادگیری تجمیعی: در روش‌های یادگیری تجمیعی، یک مدل از ترکیب دو یا چند مدل مختلف ایجاد می شود. این مدل ها ممکن است از یک نوع یا حتی از انواع مختلف

باشند. ترکیب مدل ها می تواند به صورت روش های آماری مانند میانگین گیری (میانگین تخمین نهایی) یا با اطلاعات بیشتر، در نظر گرفتن میزان صحت هر مدل بسته به شرایط باشد. این روش ها در شرایطی که مهم ترین موضوع تنها عملکرد مدل نهایی باشد، استفاده می شوند و دو سود بسیار مهم دارند:

اول، در بهبود عملکرد نهایی نقش به سزایی دارند چراکه می توانند به عملکردی بهتر از هر یک از مدل ها به تنهایی داشته باشند.

دوم، می توانند از پخشی یا پراکندگی تخمین های یک مدل جلوگیری کنند. به عبارت دیگر، در بحث bias–variance trade–off، تجمیع مدل ها می تواند باعث کاهش واریانس به قیمت افزایش bias شوند. در نتیجه این اتفاق، مدل نهایی robust خواهد بود.

به عنوان مثال، در نظر بگیرید که عملکرد یک مدل در روش k–fold cross validation می تواند بایاس زیادی از خود نشان دهد. معمولا عملکرد میانگین سنجیده می شود ولی در صورتی می توانیم به مدل اعتماد کنیم که عملکرد آن واریانس بالایی نیز نداشته باشد. یک روش تجمیعی راحت، آموزش چند باره مدل با داده train و سپس ترکیب خروجی های آن هاست، مثلا انتخاب میانگین در regression یا انتخاب رای اکثریت در classification. تفاوت در مدل ها می تواند از stochastic بودن خود الگوریتم یا بخش بندی داده train یا تغییر در خود مدل ایجاد شود. در نتیجه این کار، واریانس عملکرد مدل ترکیبی بسیار کمتر خواهد بود و بهترین و بدترین عملکرد نیز نزدیک به میانگین عملکرد خواهند بود.

همانطور که گفتیم، مسئله دیگر در استفاده از مدل های تجمعی، بهبود عملکرد است. در واقع این علت اصلی استفاده از روش های تجمیعی است و معمولا توسط برندگان مسابقات یادگیری ماشین استفاده شده است. در واقع برای بهبود عملکرد، در صورتی از مدل تجمیعی استفاده می شود که مدل تجمیعی به صورت میانگین بهتر از هر یک از مدل های سازنده اش عمل کند. (وگرنه از مدل سازنده ای که از آن بهتر بود استفاده می کردیم.) هرچند این هدف همواره محقق نمی شود، و در مواردی که یکی از مدل های تشکیل دهنده بهتر از باقی مدل ها عمل کند و باقی مدل ها نتواند سبب بهبود عملکرد شوند، یا مدل تجمیعی نتواند به درستی از خروجی آن ها استفاده کند و باعث کاهش عملکرد شود.

- روش Bagging: روش bagging، یا همان bootstrap aggregating، یک روش ساده ولی قدرمند برای یادگیری تجمعی است. bagging، استفاده از bootstrapping برای مدل های با واریانس بالا مانند درخت تصمیم است.

برای توضیح این روش، ابتدا در مورد bootstrapping توضیح می دهیم. Bootstrapping یک نمونه برداری تصادفی با جایگذاری است که در آن یک زیرمجموعه از مجموعه داده اصلی انتخاب می شود (که اعضای آن می توانند تکراری باشند). مثلا فرض کنید لازم داشته باشیم میانگین یک مجموعه داده را تخمین بزنیم. یک راه این است که با روش bootstrapping، تعدادی زیر مجموعه با تکرار کوچک در نظر بگیریم و میانگین هر یک را محاسبه کنیم. سپس این میانگین ها را میانگین بگیریم. به این صورت تخمینی از میانگین مجموعه اصلی خواهیم داشت.

در bagging، از همین ایده استفاده می شود. فرض کنید N مشاهده از M فیچر داریم. ابتدا یک نمونه از مشاهده به صورت bootstrapping انتخاب می کنیم. (نمونه برداری با جایگذاری)

سپس، یک زیر مجموعه از فیچر ها را انتخاب کرده تا مدلی را با مشاهدات و فیچر های محدود آموزش دهیم.

حال فیچری را انتخاب می کنیم که داده train را بهتر split می کند (انتخاب بهترین split مانند درخت تصمیم).

این عملیات را تکرار می کنیم و هر مدل را به صورت موازی آموزش می دهیم.

در نهایت، با استفاده از aggregate کردن تخمین هر مدل، یک تخمین نهایی ارائه می دهیم.

با استفاده از bagging، دیگر کمتر نگران این هستیم که یک درخت خاص overfit شود. پس می توان درخت های تصمیم را تا عمق زیادی پیش برد و آن ها را هرس نکرد. (یعنی لزومی ندارد در برخی راس ها به اجبار و برای جلوگیری از overfit شدن، دیگر split نداشته باشیم و می توانیم تا جایی پیش برویم که هر راس شامل تعداد کمی نمونه باشد.) در نتیجه هر درخت به تنهایی واریانس بالا و بایاس پایین دارد، چراکه به نوعی overfit شده است، ولی از ترکیب آن ها واریانس کاهش می یابد.

برای انتخاب تعداد درخت های مورد استفاده نیز می توان از تعداد کم درخت شروع کرد و تا جایی درخت ها را افزایش داد که کماکان شاهد بهبود عملکرد با افزایش تعداد درخت هستیم، و پس از آن متوقف شد.

- برای پیاده‌سازی روش یادگیری تجمعی از سه رویکرد استفاده کرده‌ایم.

۱. پیاده‌سازی شخصی از bagging با مدل‌های مختلف:

در تابع BaggingClassifier ابتدا یک زیرنمونه از داده برداشته‌شد و سپس، سه مدل mlp، SVM و Knn آموزش داده‌شدند و پس از پیش‌بینی هر یک، رای اکثریت به عنوان پیش‌بینی نهایی در نظرگرفته‌شد.

```
Accuracies Mean: 0.8955036598117811
Accuracies std: 0.022704080633762985
```

میانگین و واریانس دقت مدل
BaggingClassifier

همانطور که مشاهده‌می‌شود این مدل از مدل نهایی که بهترین مدل در میان سایرین بود، میانگین دقت بهتری دارد و واریانس آن نیز کمی کمتر شده‌است. (۰/۵ درصد افزایش میانگین دقت و ۰.۳ درصد کاهش واریانس)

۲. استفاده از BaggingClassifier کتابخانه sklearn:

این تابع از کتابخانه sklearn یک کلاس به عنوان base estimator می‌گیرد و به تعداد از آن می‌سازد و با قسمت‌کردن داده آن‌ها را آموزش می‌دهد. نتایج این مدل به صورت زیر است.

```
Accuracies Mean: 0.8981962356221679
Accuracies std: 0.030155060531297 96
```

میانگین و واریانس دقت مدل
BaggingClassifier کتابخانه sklearn

همانطور که مشاهده‌می‌شود این مدل از مدل نهایی که بهترین مدل در میان سایرین بود، میانگین دقت بهتری دارد ولی واریانس آن کمی بیشتر شده‌است. (۰.۸ درصد افزایش میانگین دقت و ۰.۷ درصد افزایش واریانس) این ممکن است به دلیل به اندازه کافی خوب بودن knn باشد که افزایش چشمگیری را شاهد نیستیم. با اجرا بر روی مدل ضعیف‌تری مانند decision tree نتیجه به صورت زیر خواهد بود.

```
Bagging:
Accuracies Mean: 0.8187521784593935
Accuracies std: 0.029363020942906982
Single Decision Tree
Accuracies Mean: 0.773841059602649
Accuracies std: 0.043823849686266736
```

میانگین و واریانس دقت مدل BaggingClassifier کتابخانه
sklearn با استفاده از درخت تصمیم به عنوان پایه

همانطور که می‌بینید این کار باعث شد تا هم میانگین دقت به طرز چشمگیری افزایش یابد(حدود ۵ درصد) و هم واریانس کاهش خوبی داشته‌باشد.(حدود ۱/۵ درصد)

۳. استفاده از GradientBoostingClassifier کتابخانه sklearn:

در روش boosting سعی می‌شود تا هر قسمت داده با کلاس خاصی مدل شود و در نهایت، این مدل‌ها با یکدیگر ترکیب شوند تا مدل خوب به‌دست‌آید.

برای این کار از کلاس GradientBoostingClassifier کتابخانه sklearn استفاده‌شده‌است. نتیجه به صورت زیر به‌دست‌آمد.

```
Accuracies Mean: 0.8571540606483096
Accuracies std: 0.028220723211463503
```

میانگین و واریانس دقت مدل BoostingClassifier
کتابخانه sklearn

همانطور که مشاهده‌می‌کنید میانگین و واریانس دقت قابل‌قبولی از این مدل دیده‌می‌شود اما از بهترین دقت مدل‌های ما بهتر نبوده‌است. البته این مدل پارامترهای زیادی دارد که با برخی از آن‌ها آزمون خطایی بازی شد و دقت مدل کمی بالاتر آمد.

+ تمامی کدها به همراه خروجی‌ها(اعم از نمودارهای ROC) در نوت‌بوک ارسالی به همراه گزارش آمده‌است. این نوت‌بوک خروجی حاصل از google colab است که لینک آن به صورت زیر است.

https://colab.research.google.com/drive/1cEy8i07fbUF1ILrjN5a28ugQxNyc1S3a?usp=sharing

در این notebook بخش‌ها از یکدیگر جداشده‌اند و از روی عناوین می‌توانید به بخش مربوطه برسید.