

دانشگاه صنعتی خواجه نصیرالدین طوسی  
دانشکده مهندسی برق

## درس مبانی سیستم‌های هوشمند

استاد: دکتر مهدی علیاری

مینی پروژه دوم

محمدامین محمدیون شبستری	نام و نام خانوادگی
۴۰۱۲۲۵۰۳	شماره دانشجویی
محمد سبحان سخایی	نام و نام خانوادگی
۴۰۱۱۹۲۵۳	شماره دانشجویی
مبینا یوسفی مقدم	نام و نام خانوادگی
۴۰۱۲۴۰۹۳	شماره دانشجویی
۱۴۰۴ آذر	تاریخ
لینک‌های مربوط به کد و دیتاست :	
کد + دیتاست   لینک گیت‌هاب	



## فهرست مطالب

۶	پرسش یک- سخت حاشیه، حاشیه هندسی، دوگان و بردارهای پشتیبان	۱
۶	فرم اولیه سخت حاشیه (Hard Margin Primal Form)	۱.۱
۶	اثبات اینکه پهنای حاشیه برابر است با $\frac{2}{\ w\ }$	۲.۱
۷	نوشتن و محاسبه مسئله دوگان (Dual Problem)	۳.۱
۸	چرا بردارهای پشتیبان (Support Vectors) تنها داده‌هایی با $\alpha_i > 0$ هستند؟	۴.۱
پرسش دو- حاشیه نرم با نرم <b>Hinge Loss</b> (Hinge Loss)		
۸	فرم اولیه حاشیه نرم با متغیرهای کمکی (Slack Variables)	۱.۲
۹	اثبات دقیق ریاضی: همارزی فرم اولیه با کمینه‌سازی Hinge Loss	۲.۲
۹	گام اول: تعریف مسئله مقید	۱.۲.۲
۹	گام دوم: تحلیل متغیر کمکی بهینه ( $\epsilon^*$ )	۲.۲.۲
۱۰	گام سوم: تحلیل حالات مختلف	۳.۲.۲
۱۰	گام چهارم: جایگذاری و نتیجه‌گیری نهایی	۴.۲.۲
۱۱	تحلیل دقیق نقش پارامتر $C$ (پارامتر جریمه)	۳.۲
۱۱	حالت اول: بزرگ $C$ (سخت‌گیرانه)	۱.۳.۲
۱۱	حالت دوم: کوچک $C$ (ملایم)	۲.۳.۲
۱۲	جدول خلاصه تأثیر $C$	۳.۳.۲
۱۲	توضیح شرایط KKT	۴.۲
پرسش سه- نرم <b>L2</b> و تفاوت دوگان با <b>L1</b>		
۱۳	فرم اولیه حاشیه نرم L2 (Soft Margin-L2)	۱.۳
۱۳	محاسبه مسئله دوگان L2	۲.۳
۱۳	تحلیل تفاوت کران‌های $\alpha_i$ و اثرهای آن	۳.۳
۱۴	منشأ ریاضی تفاوت کران‌ها	۱.۳.۳
۱۴	تأثیر بردارهای پشتیبان و تنکی (Sparsity)	۲.۳.۳
۱۵	تأثیر بر پایداری در برابر داده‌های پرت (Outliers)	۳.۳.۳
۱۵	کرنل معنبر و ارتباط با نگاشت ویژگی	۴.۳
۱۵	اثبات اعتبار کرنل‌ها با خواص بستاری (Closure Properties)	۵.۳
۱۶	تحلیل جامع شرایط KKT برای L2-SVM	۶.۳
۱۶	۱. ایستایی (Stationarity)	۱.۶.۳
۱۶	۲. موجه بودن اولیه (Primal Feasibility)	۲.۶.۳
۱۷	۳. موجه بودن دوگان (Dual Feasibility)	۳.۶.۳
۱۷	۴. شرط مکمل بودن (Complementary Slackness)	۴.۶.۳
۱۷	تحلیل نهایی و نتیجه‌گیری از ترکیب شرایط	۵.۶.۳



۱۸	پرسش پنجم - محاسبه دستی SVM سخت حاشیه در $R^2$ با نمونه‌های بیشتر	۴
۱۸	۱.۴ بخش (آ): حدس ابرصفحه حداکثر حاشیه و معادله مرز تصمیم	۱۴
۱۸	۲.۴ بخش (ب): محاسبه دستی ضرایب $w$ و $b$	۲۴
۱۹	۳.۴ بخش (ج): بردارهای پشتیبان، مقدار حاشیه هندسی و فاصله دو ابرصفحه	۳۴
۲۰	پرسش ششم - طبقه‌بندی و دسته‌بندی غیرخطی	۵
۲۱	۱.۵ انتخاب مدل مناسب	۵
۲۱	۲.۵ تحلیل ضعف مدل‌های سراسری (MLP) و اثبات ریاضی	۲۵
۲۱	۱.۲.۵ ۱. تحلیل و اثبات ریاضی رفتار نامناسب MLP	۲۵
۲۲	۲.۲.۵ ۲. تحلیل و اثبات ریاضی رفتار مناسب RBF	۲۵
۲۲	۳.۵ خلاصه و نتیجه‌گیری نهایی	۳۵
۲۳	بخش اول - پیاده‌سازی RBFNN	۶
۲۳	۱.۶ ایجاد دیتابست	۶
۲۷	۲.۶ تقسیم داده‌ها به دو بخش آموزش و تست	۶
۲۸	۳.۶ مساله تقریب تابع	۶
۲۸	۴.۶ بخش دوم: پیاده‌سازی RBFNN ایستا	۶
۲۹	۵.۶ تعریف تابع RBF فعال‌ساز گاوی و تست یک ورودی	۶
۳۰	۶.۶ بخش سوم: آموزش از طریق حداقل مربعات خطی (LLS)	۶
۳۶	۷.۶ سریع بودن LLS نسبت به backpropagation	۶
۳۷	۸.۶ آموزش مدل با تغییر تعداد مراکز و ثابت گرفتن مقدار واریانس نورون‌ها	۶
۳۹	۹.۶ آموزش مدل با تغییر واریانس و ثابت گرفتن تعداد نورون‌ها	۶
۴۲	۱۰.۶ چالش‌ها	۶
۴۳	۱۱.۶ بخش چهارم: تلاش برای حل چالش‌ها	۶
۴۵	۱۲.۶ نقش پارامتر $\lambda$ در L <sub>2</sub> Regularization	۶
۴۷	۱۳.۶ بخش پنجم:	۶
۵۷	۱۴.۶ نتایج بهتر kmeans clustering نسبت به روش تصادفی	۶



## فهرست تصاویر

۶	نمایش هندسی حاشیه در ماشین بردار پشتیبان (SVM) و اثبات پهنه‌ای آن برابر با $\frac{2}{\ w\ }$	۱
۲۰	دیتاست همراه با داده آموزشی و تست	۲
۲۴	۵ سطر اول	۳
۲۶	ترسیم خروجی به همراه ورودی $(t-1)y$ و نمایش موقعیت $y(t)$ نسبت به $y(t-1)$ برای ۲۵۰ نمونه اول	۴
	نمایش سه بعدی خروجی نسبت به ویژگی‌های $y(t-1)$ و $y(t-2)$ به همراه رسم نمودار توزیع دیتا برای پیدا کردن نقطه تکینگی	۵
۲۶	ماتریس $\phi_{train}$	۶
۳۲	بردار $\alpha$	۷
۳۴	خروجی مربوط به پیش‌بینی مدل به همراه خروجی واقعی برای دیتای آموزش	۸
۳۵	خروجی مربوط به پیش‌بینی مدل به همراه خروجی واقعی برای دیتای تست	۹
۳۸	نمایش تغییرات RMSE نسبت به تغییرات $k$	۱۰
۴۰	نمایش تغییرات RMSE نسبت به تغییرات $\sigma$	۱۱
۴۱	نمایش خروجی پیش‌بینی شده به همراه خروجی واقعی برای بهترین مقدار $k$ و $\sigma$	۱۲
۴۴	نمودار خروجی واقعی و خروجی پیش‌بینی شده بدون رگولاrizیشن برای مجموعه تست	۱۳
۴۷	تغییرات RMSE نسبت به تغییرات $\lambda$	۱۴
۵۰	تغییرات RMSE نسبت به تغییرات $k$ در kmeans	۱۵
۵۲	تغییرات RMSE نسبت به تغییرات $\sigma$ در kmeans	۱۶
۵۳	خروجی واقعی و پیش‌بینی مدل با استفاده از روش kmeans در بهینه‌ترین پارامترها	۱۷
۵۵	خروجی واقعی و پیش‌بینی مدل با استفاده از دو استراتژی kmeans و NearestNeighbor	۱۸
۵۶	خروجی واقعی و خروجی پیش‌بینی شده با استفاده از kmeans(sklearn)	۱۹
۵۶	خروجی واقعی و خروجی پیش‌بینی شده با استفاده از NearestNeighbor(sklearn)	۲۰



## فهرست جداول

۱۲	.....	مقایسه تأثیر مقادیر مختلف پارامتر $C$ بر رفتار مدل SVM	۱
۳۷	.....	مقادیر $k$ ، سیگما و RMSE برای داده‌های آموزش و تست	۲
۳۹	.....	مقادیر سیگما و RMSE برای داده‌های آموزش و تست	۳
۴۶	.....	مقادیر $\lambda$ و RMSE برای داده‌های آموزش و تست	۴
۵۰	.....	مقادیر $K$ ، سیگما و RMSE برای داده‌های آموزش و تست	۵
۵۲	.....	ارزیابی مدل آموزش دیده به وسیله KMeans به ازای تغییرات $K$	۶



## فهرست برنامه‌ها

۲۳	تولید ۱۰۰ نمونه برای ایجاد دیتاست beam ball	۱
۲۵	تحلیل و بررسی اثر ویژگی‌ها بر روی خروجی	۲
۲۷	تقسیم داده‌ها به آموزش و تست	۳
۲۹	تابع RBF فعال‌ساز گاوسی	۴
۲۹	تست تابع RBF فعال‌ساز گاوسی	۵
۳۰	تابع سازنده ماتریس $\phi$	۶
۳۰	تابع محاسبه $\sigma$	۷
۳۱	به دست آوردن مراکز	۸
۳۱	نرم‌السازی	۹
۳۲	نرم‌السازی	۱۰
۳۲	محاسبه ماتریس phi_train	۱۱
۳۳	پیش‌بینی خروجی‌های مدل بر روی داده‌های آموزش و تست	۱۲
۳۳	محاسبه RMSE	۱۳
۳۳	محاسبه RMSE	۱۴
۳۳	نمایش خروجی پیش‌بینی شده به همراه خروجی واقعی برای دیتای آموزش	۱۵
۳۴	نمایش خروجی پیش‌بینی شده به همراه خروجی واقعی برای دیتای تست	۱۶
۳۷	آموزش و ارزیابی مدل با تغییر تعداد مراکز و ثابت گرفتن مقدار واریانس نورون‌ها	۱۷
۳۸	تغییرات RMSE نسبت به تغییرات $k$	۱۸
۳۹	تغییرات RMSE نسبت به تغییرات $\sigma$	۱۹
۴۰	رسم تغییرات RMSE نسبت به تغییرات $\sigma$	۲۰
۴۱	کد کامل inference بر روی مجموعه تست و ارزیابی آن	۲۱
۴۳	تابع آموزش RBFNN static با استفاده از Regularization L2	۲۲
۴۳	آموزش مدل و محاسبه خروجی‌ها بر روی مجموعه آموزش و تست	۲۳
۴۳	محاسبه مقدار RMSE بر روی آموزش و تست	۲۴
۴۴	ترسیم نمودار خروجی واقعی و پیش‌بینی شده بر روی مجموعه تست	۲۵
۴۵	تغییرات RMSE نسبت به تغییرات $\lambda$	۲۶
۴۷	تابع پیاده‌سازی شده برای kmeans	۲۷
۴۹	استفاده از روش kmeans و آموزش مدل و ارزیابی آن با استفاده از تغییرات $k$	۲۸
۵۰	استفاده از روش kmeans و آموزش مدل و ارزیابی آن با استفاده از تغییر واریانس	۲۹
۵۲	آموزش مدل و ارزیابی آن با پارامترهای بهینه در روش kmeans	۳۰
۵۴	تابع سیگمای تطبیقی	۳۱
۵۴	تابع محاسبه ماتریس $\phi$ برای سیگمای تطبیقی	۳۲
۵۴	آموزش مدل و ارزیابی آن با استفاده از دو استراتژی	۳۳

## ۱ پرسش یک- سخت حاشیه، حاشیه هندسی، دوگان و بردارهای پشتیبان

### ۱.۱ فرم اولیه سخت حاشیه (Hard Margin Primal Form)

در ماشین بردار پشتیبان با حاشیه سخت، فرض بر این است که داده‌ها به صورت خطی کاملاً جداپذیر هستند. هدف پیدا کردن ابرصفحه‌ای است که حاشیه بین دو کلاس را بیشینه کند. معادله ابرصفحه تصمیم به صورت  $w^T x + b = 0$  است. مسئله بهینه‌سازی اولیه (Primal) به صورت زیر نوشته می‌شود:

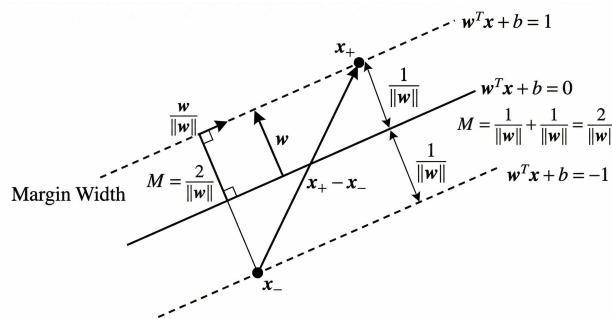
$$\begin{aligned} & \underset{w,b}{\text{minimize}} \quad \frac{1}{2} \|w\|^2 \\ & \text{to subject} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \dots, N \end{aligned} \quad (1)$$

### ۲.۱ اثبات اینکه پهنای حاشیه برابر است با $2/\|w\|$

ما دو ابرصفحه مرزی داریم که بر روی بردارهای پشتیبان قرار می‌گیرند:

۱. برای کلاس مثبت:  $w^T x + b = 1$

۲. برای کلاس منفی:  $w^T x + b = -1$



شکل ۱: نمایش هندسی حاشیه در ماشین بردار پشتیبان (SVM) و اثبات پهنای آن برابر با  $2/\|w\|$ .

فرض کنید  $x_+$  یک نقطه روی ابرصفحه مثبت و  $x_-$  نزدیکترین نقطه به آن روی ابرصفحه منفی باشد. فاصله عمودی (حاشیه هندسی) بین این دو صفحه برابر است با تصویر بردار واصل  $(x_+ - x_-)^T$  بر روی جهت نرمال صفحه. بردار نرمال واحد برابر است با  $\frac{w}{\|w\|}$ . بنابراین پهنای حاشیه  $M$  برابر است با:

$$M = (x_+ - x_-)^T \frac{w}{\|w\|} = \frac{w^T x_+ - w^T x_-}{\|w\|} \quad (2)$$

از معادلات ابرصفحه‌ها داریم:

$$w^T x_+ = 1 - b \quad (3)$$

$$w^T x_- = -1 - b \quad (4)$$



با جایگذاری در معادله حاشیه:

$$M = \frac{(1-b) - (-1-b)}{\|w\|} = \frac{1-b+1+b}{\|w\|} = \frac{2}{\|w\|} \quad (5)$$

این اثبات نشان می‌دهد که بیشینه کردن  $M$  معادل کمینه کردن  $\|w\|$  است.

### ۳.۱ نوشتمن و محاسبه مسئله دوگان (Dual Problem)

برای حل مسئله مقید فوق، از روش ضرایب لاغرانژ استفاده می‌کنیم.تابع لاغرانژین  $\mathcal{L}$  را با معرفی ضرایب لاغرانژ  $0 \geq \alpha_i$  برای هر قید تشكیل می‌دهیم:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^N \alpha_i[y_i(w^T x_i + b) - 1] \quad (6)$$

برای به دست آوردن فرم دوگان، باید مشتقات جزئی  $\mathcal{L}$  را نسبت به متغیرهای اولیه  $(w, b)$  برابر صفر قرار دهیم.

گام ۱: مشتق نسبت به  $w$

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^N \alpha_i y_i x_i = 0 \implies w = \sum_{i=1}^N \alpha_i y_i x_i \quad (7)$$

گام ۲: مشتق نسبت به  $b$

$$\frac{\partial \mathcal{L}}{\partial b} = - \sum_{i=1}^N \alpha_i y_i = 0 \implies \sum_{i=1}^N \alpha_i y_i = 0 \quad (8)$$

گام ۳: جایگذاری در لاغرانژین

حال مقادیر به دست آمده را در معادله اصلی جایگذاری می‌کنیم. جمله اول ( $\frac{1}{2}\|w\|^2$ ):

$$\frac{1}{2}w^T w = \frac{1}{2} \left( \sum_{i=1}^N \alpha_i y_i x_i \right)^T \left( \sum_{j=1}^N \alpha_j y_j x_j \right) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i^T x_j) \quad (9)$$

جمله دوم (بسط سیگما):

$$- \sum_{i=1}^N \alpha_i y_i (w^T x_i + b) + \sum_{i=1}^N \alpha_i \quad (10)$$



$$= - \sum_{i=1}^N \alpha_i y_i \left( \left( \sum_{j=1}^N \alpha_j y_j x_j \right)^T x_i \right) - b \underbrace{\sum_{i=1}^N \alpha_i y_i}_{0} + \sum_{i=1}^N \alpha_i \quad (11)$$

$$= - \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_j^T x_i) + \sum_{i=1}^N \alpha_i \quad (12)$$

با جمع کردن جملات:

$$\mathcal{L}_D(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i^T x_j) \quad (13)$$

بنابراین مسئله دوگان نهایی عبارت است از:

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} \quad \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i^T x_j) \\ & \text{to subject} \quad \alpha_i \geq 0, \quad \forall i \\ & \quad \sum_{i=1}^N \alpha_i y_i = 0 \end{aligned} \quad (14)$$

## ۴.۱ چرا بردارهای پشتیبان (Support Vectors) تنها داده‌هایی با $\alpha_i > 0$ هستند؟

این موضوع مستقیماً از شرط Complementary Slackness در شرایط KKT ناشی می‌شود. این شرط بیان می‌کند که حاصل ضرب ضریب لاگرانژ در مقدار محدودیت باید صفر باشد:

$$\alpha_i [y_i (w^T x_i + b) - 1] = 0 \quad (15)$$

این معادله دو حالت دارد:

۱. اگر  $1 > y_i (w^T x_i + b) > 0$  باشد (داده به درستی دسته‌بندی شده و خارج از حاشیه است)، آنگاه عبارت داخل براکت مخالف صفر است. برای برقاری تساوی، الزاماً  $\alpha_i = 0$  باشد.

۲. اگر داده دقیقاً روی مرز حاشیه باشد، یعنی  $y_i (w^T x_i + b) = 1$  باشد، آنگاه عبارت داخل براکت صفر است و  $\alpha_i$  می‌تواند مقداری مثبت داشته باشد.

از آنجا که بردار وزن نهایی  $w = \sum \alpha_i y_i x_i$  است، تنها نقاطی که  $\alpha_i \neq 0$  دارند در ساختن مرز تصمیم نقش دارند. به این نقاط بردارهای پشتیبان می‌گویند.

## ۲ پرسش دو- حاشیه نرم با نرم Hinge Loss و هینج لاس (Hinge Loss)

### ۱.۲ فرم اولیه حاشیه نرم با متغیرهای کمکی (Slack Variables)

برای داده‌هایی که خطی جداپذیر نیستند، متغیرهای کمکی  $0 \leq \xi_i$  را معرفی می‌کنیم. فرمول به صورت زیر است:



$$\begin{aligned} & \underset{w, b, \xi}{\text{minimize}} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ & \text{to subject} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad i = 1, \dots, N \\ & \quad \xi_i \geq 0 \end{aligned} \quad (16)$$

## ۲.۲ اثبات دقیق ریاضی: همارزی فرم اولیه با کمینه‌سازی Hinge Loss

هدف این بخش نشان دادن این است که مسئله بهینه‌سازی مقید (با متغیرهای کمکی) دقیقاً معادل با یک مسئله بهینه‌سازی نامقید است که در آن تابع زیان Hinge Loss کمینه می‌شود.

### ۱.۲.۲ گام اول: تعریف مسئله مقید

مسئله استاندارد Soft Margin SVM (فرم اولیه) به صورت زیر تعریف می‌شود:

$$\begin{aligned} & \underset{w, b, \xi}{\text{minimize}} \quad J_{\text{primal}}(w, \xi) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ & \text{to subject} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \forall i \\ & \quad \xi_i \geq 0, \quad \forall i \end{aligned} \quad (17)$$

در این مسئله، ما به دنبال پیدا کردن مقادیر  $w$ ,  $b$  و  $\xi$  هستیم که تابع هزینه را کمینه کنند.

### ۲.۲.۲ گام دوم: تحلیل متغیر کمکی بهینه ( $\xi_i^*$ )

فرض کنید مقادیر  $w$  و  $b$  ثابت هستند. می‌خواهیم بینیم برای این  $w$  و  $b$  خاص، بهترین مقدار برای  $\xi$  چه باید باشد تا هزینه کل ( $J_{\text{primal}}$ ) کمینه شود. از آنجا که ضریب  $C$  مثبت است، برای کمینه کردن تابع هدف، باید  $\sum \xi_i$  کمینه شود. پس برای هر نمونه  $i$ ، باید کوچکترین مقدار ممکن برای  $\xi_i$  را پیدا کیم که قیود مسئله را نقض نکند.

قیود حاکم بر  $\xi_i$  را بازنویسی می‌کنیم:

۱. قید دسته‌بندی:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \implies \xi_i \geq 1 - y_i(w^T x_i + b) \quad (18)$$

۲. قید مثبت بودن:

$$\xi_i \geq 0 \quad (19)$$

بنابراین، متغیر  $\xi_i$  باید هم‌مان بزرگتر از صفر و بزرگتر از مقدار  $(1 - y_i(w^T x_i + b))$  باشد. برای کمینه کردن  $\xi_i$ ، باید کوچکترین عددی را انتخاب کنیم که هر دو شرط را ارضاء کند. ریاضیات آن به صورت زیر است:

$$\xi_i^* = \max(0, 1 - y_i(w^T x_i + b)) \quad (20)$$



### ۳.۲.۲ گام سوم: تحلیل حالات مختلف

بیایید این مقدار بهینه را برای دو وضعیت ممکن داده‌ها بررسی کنیم:  
 حالت (الف) داده به درستی و با حاشیه مطمئن دسته‌بندی شده باشد:  
 در این حالت  $y_i(w^T x_i + b) \geq 1$  است.

$$\implies 1 - y_i(w^T x_i + b) \leq 0 \quad (21)$$

چون مقدار عبارت منفی (یا صفر) است و  $y_i$  باید مثبت باشد ( $\max(0, \text{negative})$ ), پس:

$$\xi_i^* = 0 \quad (22)$$

این منطقی است؛ وقتی داده صحیح است، نیازی به متغیر کمکی نداریم.  
 حالت (ب) داده غلط دسته‌بندی شده یا درون حاشیه است (نقض حاشیه):  
 در این حالت  $y_i(w^T x_i + b) < 1$  است.

$$\implies 1 - y_i(w^T x_i + b) > 0 \quad (23)$$

چون مقدار عبارت مثبت است، ماکسیمم بین صفر و یک عدد مثبت، همان عدد مثبت است:

$$\xi_i^* = 1 - y_i(w^T x_i + b) \quad (24)$$

در اینجا  $\xi_i^*$  دقیقاً برابر با مقدار "کمبود" یا خطای فاصله تا حاشیه است.

### ۴.۲.۲ گام چهارم: جایگذاری و نتیجه‌گیری نهایی

حال که مقدار بهینه  $\xi_i^*$  را برحسب  $w$  و  $b$  پیدا کردیم، آن را در تابع هدف اصلی (معادله ۱۷) جایگذاری می‌کنیم تا متغیر  $\xi$  حذف شود:

$$J_{unconstrained}(w, b) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \underbrace{\max(0, 1 - y_i(w^T x_i + b))}_{\text{Loss Hinge}} \quad (25)$$

ترم دوم دقیقاً تعریف تابع زیان Hinge Loss است که معمولاً به صورت  $L(y, f(x)) = \max(0, 1 - yf(x))$  نمایش داده می‌شود.  
 نتیجه: مسئله Soft Margin SVM از نظر ریاضی دقیقاً معادل است با مسئله رگولاریزاسیون تیخونوف (Tikhonov Regularization) که در آن تابع زیان برابر با Hinge Loss و جریمه پیچیدگی مدل برابر با نرم L2 وزن‌ها می‌باشد:

$$\min_{w,b} \left( \sum_{i=1}^N L_{\text{Hinge}}(y_i, f(x_i)) + \lambda \|w\|^2 \right) \quad (26)$$

(که در اینجا  $\lambda = \frac{1}{2C}$  است).



## ۳.۲ تحلیل دقیق نقش پارامتر C (پارامتر جریمه)

پارامتر  $C$  در تابع هدف SVM نقش یک ضریب تنظیم‌گر (Regularization Coefficient) معکوس را ایفا می‌کند. تابع هدف را به یاد بیاورید:

$$J(w, \xi) = \underbrace{\frac{1}{2} \|w\|^2}_{\text{کنترل پیچیدگی مدل (حاشیه)}} + C \sum_{i=1}^N \xi_i \quad (27)$$

کنترل خطای آموزش

این پارامتر یک "موازنۀ" (Trade-off) حیاتی بین دو هدف متضاد ایجاد می‌کند:

۱. ساده نگه داشتن مدل (با بیشینه کردن حاشیه یا کمینه کردن  $\|w\|$ ).

۲. کاهش خطای دسته‌بندی روی داده‌های آموزشی (با کمینه کردن مجموع  $\xi_i$ ).

در ادامه تأثیر مقادیر مختلف  $C$  را با جزئیات بررسی می‌کنیم:

### ۱.۳.۲ حالت اول: $C$ بزرگ (سخت‌گیرانه)

وقتی مقدار  $C$  بزرگ انتخاب می‌شود (مثلاً  $C \rightarrow \infty$ ):

- استدلال ریاضی: در تابع هزینه، وزن عبارت  $\sum \xi_i$  بسیار زیاد می‌شود. بنابراین، الگوریتم بهینه‌سازی تمام تلاش خود را می‌کند تا  $\xi_i$  را صفر کند (یا به صفر بسیار نزدیک کند). هرگونه خطای دسته‌بندی هزینه سنگینی به مدل تحمیل می‌کند.

- هنده حاشیه: برای اینکه خطاهای صفر شوند، مرز تصمیم مجبور است خود را با جزئیات دقیق داده‌ها و حتی نویزها تطبیق دهد. این باعث می‌شود حاشیه بسیار باریک شود.

- Bias/Variance**

- **Low Bias** : خطای مدل روی داده‌های آموزشی بسیار کم است.

- **High Variance** : مدل به شدت به داده‌های آموزشی وابسته می‌شود و با تغییر اندکی در داده‌ها، مرز تصمیم تغییر زیادی می‌کند. این یعنی خطر بیش‌بازش (Overfitting) بالاست.

- تعداد بردارهای پشتیبان: چون حاشیه باریک است، نقاط کمی درون حاشیه یا روی مرز قرار می‌گیرند ( فقط نقاطی که بسیار نزدیک به مرز هستند). بنابراین معمولاً تعداد بردارهای پشتیبان کم است.

### ۲.۳.۲ حالت دوم: $C$ کوچک (مالایم)

وقتی مقدار  $C$  کوچک انتخاب می‌شود (مثلاً  $C \rightarrow 0$ ):

- استدلال ریاضی: در تابع هزینه، اهمیت ترم خطای  $\sum \xi_i$  کاهش می‌یابد و اولویت اصلی به کمینه کردن  $\|w\|$  (پهن کردن حاشیه) داده می‌شود. الگوریتم حاضر است خطاهای دسته‌بندی (Misclassification) را پذیرید تا در عوض حاشیه بزرگتری داشته باشد.

- هنده حاشیه: مرز تصمیم صاف‌تر (Smoother) و ساده‌تر می‌شود و حاشیه پهن‌تر می‌گردد.



### • وضعیت Bias/Variance

- High Bias** - ممکن است خطای مدل روی داده‌های آموزشی زیاد شود (چون سختگیری کم شده است).
- Low Variance** - مدل پایدارتر است و تعمیم‌پذیری (Generalization) بهتری روی داده‌های دیده نشده دارد. اگر  $C$  بیش از حد کوچک باشد، خطر کمبازش (Underfitting) وجود دارد.
- تعداد بردارهای پشتیبان: نکته مهم و گاهًا متناقض‌نما اینجاست که وقتی حاشیه پهن می‌شود، تعداد داده‌هایی که «درون حاشیه» قرار می‌گیرند (یعنی شرط  $1 < w^T x_i + b$  را دارند) بیشتر می‌شود. طبق تعریف KKT، تمامی این نقاط بردار پشتیبان هستند. پس در  $C$  کوچک، تعداد بردارهای پشتیبان معمولاً زیاد است.

### ۳.۳.۲ جدول خلاصه تأثیر $C$

تعداد بردارهای پشتیبان	واریانس	خطای آموزش	پهنای حاشیه	مقدار $C$
کم	(Overfit)	کم	باریک	بزرگ
زیاد	(Underfit)	زیاد	پهن	کوچک

جدول ۱: مقایسه تأثیر مقادیر مختلف پارامتر  $C$  بر رفتار مدل SVM

### ۴.۲ توضیح شرایط KKT

شرایط KKT شروط لازم برای بهینگی در مسائل مقید هستند. چهار شرط اصلی عبارتند از:

۱. ایستایی (Stationarity): گرادیان تابع لاگرانژ نسبت به متغیرهای اولیه باید صفر باشد.

$$\nabla f(x) + \sum \lambda_i \nabla g_i(x) = 0 \quad (28)$$

۲. موجه بودن اولیه (Primal Feasibility): جواب باید قیود اصلی مسئله را ارضاء کند ( $g_i(x) \leq 0$ ).

۳. موجه بودن دوگان (Dual Feasibility): ضرایب لاگرانژ برای قیود نامساوی باید نامنفی باشند ( $\lambda_i \geq 0$ ).

۴. مکمل بودن (Complementary Slackness): برای هر قید، یا ضریب لاگرانژ صفر است یا قید فعال است (روی مرز).

$$\lambda_i g_i(x) = 0 \quad (29)$$

مثال ساده برای درک

فرض کنید می‌خواهیم  $x^2$  را کمینه کنیم به شرطی که  $3 - x \leq 3$  (یا  $0 \leq x \leq 3$ ).

- ۱. ایستایی:  $.2x + \lambda(-1) = 0 \implies \lambda = 2x$

- ۲. مکمل بودن:  $\lambda(3 - x) = 0$

اگر  $\lambda = 0$  باشد،  $x = 0$  می‌شود که قید  $3 - x \geq 3$  را نقض می‌کند. پس باید  $0 \leq x \leq 3$  باشد، یعنی  $3 - x = 0$ . در نتیجه  $6 = 2(3) = 2\lambda$  که مثبت است و شرط دوگان را ارضاء می‌کند.



### ۳ پرسش سه - نرم L2 و تفاوت دوگان با L1

#### ۱.۳ فرم اولیه حاشیه نرم L2 (Soft Margin-L2)

در اینجا مجموع مربعات متغیرهای کمکی را کمینه می‌کنیم. این کار جریمه سنگین‌تری برای خطاهای بزرگ در نظر می‌گیرد.

$$\begin{aligned} & \underset{w, b, \xi}{\text{minimize}} \quad \frac{1}{2} \|w\|^2 + \frac{C}{2} \sum_{i=1}^N \xi_i^2 \\ & \text{to subject} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \end{aligned} \quad (30)$$

#### ۲.۳ محاسبه مسئله دوگان L2

لاگرانژین را تشکیل می‌دهیم:

$$\mathcal{L} = \frac{1}{2} \|w\|^2 + \frac{C}{2} \sum_{i=1}^N \xi_i^2 - \sum_{i=1}^N \alpha_i [y_i(w^T x_i + b) - 1 + \xi_i] \quad (31)$$

مشتقات را صفر قرار می‌دهیم:

$$\frac{\partial \mathcal{L}}{\partial w} = 0 \implies w = \sum \alpha_i y_i x_i \quad (32)$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \implies \sum \alpha_i y_i = 0 \quad (33)$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C \xi_i - \alpha_i = 0 \implies \xi_i = \frac{\alpha_i}{C} \quad (34)$$

جایگذاری در لاگرانژین: ترمی که تغییر می‌کند بخش مربوط به  $\xi$  است:

$$\frac{C}{2} \sum \xi_i^2 - \sum \alpha_i \xi_i = \frac{C}{2} \sum \left( \frac{\alpha_i}{C} \right)^2 - \sum \alpha_i \left( \frac{\alpha_i}{C} \right) = \frac{1}{2C} \sum \alpha_i^2 - \frac{1}{C} \sum \alpha_i^2 = -\frac{1}{2C} \sum \alpha_i^2 \quad (35)$$

این عبارت را می‌توان با استفاده از دلتای کرونکر ( $\delta_{ij}$ ) در فرم دوگان ادغام کرد:

$$\mathcal{L}_D(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \left( x_i^T x_j + \frac{1}{C} \delta_{ij} \right) \quad (36)$$

#### ۳.۳ تحلیل تفاوت کران‌های $\alpha_i$ و اثرهای آن

یکی از تفاوت‌های بنیادی بین L1-SVM (حاشیه نرم استاندارد) و L2-SVM (حاشیه نرم مربعی)، در محدودیت‌هایی است که بر ضرایب لاغرانژ ( $\alpha_i$ ) اعمال می‌شود. این تفاوت مستقیماً از نحوه تعریف متغیرهای کمکی ( $\xi$ ) در تابع هدف اولیه ناشی می‌شود.



### ۱.۳.۳ منشاً ریاضی تفاوت کران‌ها

برای درک اینکه چرا کران‌ها متفاوت‌اند، باید به مشتقات جزئی لاگرانژین نسبت به متغیر  $\xi_i$  نگاه کنیم:

#### ۱. در مدل L1-SVM

تابع هدف شامل  $\sum_i C \xi_i$  است و قید  $0 \leq \xi_i \leq C$  وجود دارد (با ضریب لاگرانژ  $\mu_i$ ). بخشی از لاگرانژین که شامل  $\xi_i$  است:

$$L = \dots + C \sum \xi_i - \sum \alpha_i \xi_i - \sum \mu_i \xi_i \quad (37)$$

مشتق نسبت به  $\xi_i$ :

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \implies \alpha_i = C - \mu_i \quad (38)$$

چون طبق شرایط KKT باید  $0 \leq \mu_i \leq C$  باشد، پس لزوماً  $\alpha_i \leq C$  خواهد بود. همچنین چون  $0 \leq \alpha_i \leq C$  است، قید نهایی به صورت زیر است:

$$0 \leq \alpha_i \leq C \quad (\text{Box Constraint}) \quad (39)$$

#### ۲. در مدل L2-SVM

تابع هدف شامل  $\frac{C}{2} \sum \xi_i^2$  است و نیازی به قید صریح  $0 \leq \xi_i \leq C$  نیست. بخشی از لاگرانژین که شامل  $\xi_i$  است:

$$L = \dots + \frac{C}{2} \sum \xi_i^2 - \sum \alpha_i \xi_i \quad (40)$$

مشتق نسبت به  $\xi_i$ :

$$\frac{\partial L}{\partial \xi_i} = C \xi_i - \alpha_i = 0 \implies \xi_i = \frac{\alpha_i}{C} \quad (41)$$

در اینجا  $\alpha_i$  (خطای مدل) می‌تواند هر مقدار مثبت بزرگی داشته باشد (برای داده‌های پرت دور). چون  $\alpha_i$  متناسب با  $\xi_i$  است، هیچ کران بالایی برای  $\alpha_i$  وجود ندارد.

$$0 \leq \alpha_i < \infty \quad (42)$$

### ۲.۳.۳ تأثیر بردارهای پشتیبان و تنکی (Sparsity)

تفاوت در کران‌ها منجر به تفاوت چشمگیر در تعداد و ماهیت بردارهای پشتیبان می‌شود:

#### • خاصیت تنکی در L1-SVM: در این مدل، بردارهای پشتیبان سه دسته‌اند:

۱. داده‌های صحیح دور از مرز ( $\xi_i = 0$ ):

۲. داده‌های روی حاشیه ( $0 < \xi_i < C$ ):

۳. داده‌های ناهنجار یا خطدار ( $\xi_i > C$ ):

در این حالت، تأثیر داده‌های پرت «اشباع» می‌شود. یعنی اگر یک داده خیلی پرت باشد، زور آن بیشتر از  $C$  نمی‌شود. این باعث می‌شود جواب‌ها معمولاً تُنک (Sparse) باشند (تعداد زیادی  $\alpha$  صفر هستند).



- فقدان تنکی در **L2-SVM** (چگالی): در این مدل رابطه  $\alpha_i = C\xi_i$  برقرار است. این یعنی هر داده‌ای که حتی مقدار بسیار ناچیزی خطای داشته باشد (که در مسائل واقعی تقریباً همه داده‌ها کمی خطای دارند)، دارای  $\alpha_i > 0$  خواهد بود. علاوه بر این، در فرم دوگان دیدیم که عبارت  $\frac{1}{C}$  به قطر ماتریس کرنل اضافه می‌شود ( $K(x, x) + \frac{1}{C}$ ). این شبیه به رگرسیون Ridge عمل می‌کند و باعث می‌شود ماتریس حتماً معکوس‌پذیر و مثبت معین باشد. نتیجه این است که بردار جواب  $\alpha$  معمولاً چگال (Dense) است؛ یعنی تمام داده‌های آموزشی تبدیل به بردار پشتیبان می‌شوند.

### ۳.۳.۳ تأثیر بر پایداری در برابر داده‌های پرت (Outliers)

- مدل **L1** ( مقاومتر): اگر یک داده پرت (Outlier) خیلی دور داشته باشیم، مدل L1 فقط با نیروی ثابت  $C$  توسط آن کشیده می‌شود. مدل می‌گوید: «می‌دانم این داده غلط است، اما بیشتر از حد مشخصی ( $C$ ) به آن اهمیت نمی‌دهم.»
- مدل **L2** (حساس‌تر): اگر یک داده پرت دور داشته باشیم،  $\xi_i$  بزرگ می‌شود و چون  $\alpha_i = C\xi_i$ ، ضریب لاغرانژ آن نیز بسیار بزرگ می‌شود. یعنی داده پرت با نیروی عظیم و نامحدودی مرز تصمیم را به سمت خود می‌کشد. به همین دلیل L2-SVM نسبت به نویز و داده‌های پرت حساس‌تر است.

### ۴.۳ کرنل معتبر و ارتباط با نگاشت ویژگی

یکتابع  $K(x, z)$  یک کرنل معتبر است اگر شرط Mercer را ارضا کند، یعنی ماتریس گرام آن همواره نیمه معین مثبت (PSD) باشد. طبق قضیه مرس، اگر  $K$  یک کرنل معتبر باشد، نگاشتی  $(x)\phi$  به یک فضای ویژگی (احتمالاً با ابعاد نامتناهی) وجود دارد به طوری که:

$$K(x, z) = \langle \phi(x), \phi(z) \rangle \quad (43)$$

### ۵.۳ اثبات اعتبار کرنل‌ها با خواص بستاری (Closure Properties)

خواص بستاری: اگر  $K_1, K_2$  معتبر باشند، مجموع، ضرب، و حاصلضرب در اسکالر مثبت آنها نیز معتبر است. همچنین  $\exp(K_1)$  معتبر است.

- کرنل چندجمله‌ای:  $K(x, z) = (x^T z + c)^d$
- یک کرنل خطی معتبر است.  $c$  ثابت معتبر است.
- مجموع آنها  $(x^T z + c)$  طبق خاصیت جمع معتبر است.
- توان  $d$  یعنی  $d$  بار ضرب کرنل در خودش، که طبق خاصیت ضرب معتبر است.

کرنل **RBF**:  $K(x, z) = \exp(-\gamma \|x - z\|^2)$

بسط نرم اقلیدسی:

$$\|x - z\|^2 = \|x\|^2 + \|z\|^2 - 2x^T z \quad (44)$$



جایگذاری در نمایی:

$$K(x, z) = \underbrace{\exp(-\gamma \|x\|^2)}_{f(x)} \underbrace{\exp(-\gamma \|z\|^2)}_{f(z)} \underbrace{\exp(2\gamma x^T z)}_{\text{Kernel}} \quad (45)$$

عبارت  $2\gamma x^T z$  یک کرنل خطی است، پس  $\exp(-\gamma \|x\|^2)$  آن معتبر است. ضرب کرنل معتبر در توابع  $f(x)f(z)$  نیز طبق خواص کرنل‌ها معتبر است (تغییر مقیاس در فضای ویژگی).

### ۶.۳ تحلیل جامع شرایط KKT برای L2-SVM

شرایط کاروش-کون-تاکر (KKT) شروط لازم و کافی برای بهینگی در این مسئله محدب هستند. برای استخراج این شرایط، ابتدا تابع لاگرانژین مسئله L2-SVM را یادآوری می‌کنیم:

$$\mathcal{L}(w, b, \xi, \alpha) = \frac{1}{2} \|w\|^2 + \frac{C}{2} \sum_{i=1}^N \xi_i^2 - \sum_{i=1}^N \alpha_i [y_i(w^T x_i + b) - 1 + \xi_i] \quad (46)$$

چهار شرط KKT به شرح زیر استخراج و تفسیر می‌شوند:

#### ۱. ایستایی (Stationarity) ۲.۶.۳

گرادیان لاگرانژین نسبت به متغیرهای اولیه  $(\xi, b, w)$  باید صفر باشد. این شرط ساختار بهینه مدل را تعیین می‌کند.

• مشتق نسبت به  $w$ :

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^N \alpha_i y_i x_i = 0 \implies w^* = \sum_{i=1}^N \alpha_i y_i x_i \quad (47)$$

(این رابطه مشابه L1-SVM است و نشان می‌دهد بردار وزن ترکیبی از داده‌هاست).

• مشتق نسبت به  $b$ :

$$\frac{\partial \mathcal{L}}{\partial b} = - \sum_{i=1}^N \alpha_i y_i = 0 \implies \sum_{i=1}^N \alpha_i y_i = 0 \quad (48)$$

• مشتق نسبت به  $\xi_i$  (نکته کلیدی):

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C\xi_i - \alpha_i = 0 \implies \xi_i = \frac{\alpha_i}{C} \quad \text{یا} \quad \alpha_i = C\xi_i \quad (49)$$

تفسیر: این معادله مهم‌ترین تفاوت با L1 است. در اینجا مقدار ضریب لاگرانژ  $(\alpha_i)$  دقیقاً متناسب با مقدار خطای داده  $(\xi_i)$  است. یعنی «هر چه خطای بیشتر باشد، اهمیت آن داده در ساخت مدل  $(\alpha)$  بیشتر می‌شود».

#### ۲. موجه بودن اولیه (Primal Feasibility) ۲.۶.۳

جواب بهینه باید قیود اصلی مسئله را نقض نکند:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \forall i \quad (50)$$

این شرط تضمین می‌کند که داده‌ها با در نظر گرفتن حاشیه و خطای مجاز، در سمت درست قرار دارند.



### ۳.۶.۳. موجه بودن دوگان (Dual Feasibility)

ضرایب لاگرانژ متاظر با قیود نامساوی باید نامنفی باشند:

$$\alpha_i \geq 0, \quad \forall i \quad (51)$$

نکته مهم این است که برخلاف L1-SVM، اینجا شرط  $C \leq \alpha_i$  وجود ندارد. زیرا  $\alpha_i$  متناسب با  $\xi_i$  است و  $\xi_i$  می‌تواند هر عدد بزرگی باشد.

### ۴. شرط مکمل بودن (Complementary Slackness) ۴.۶.۳

حاصل ضرب ضریب لاگرانژ در مقدار قید باید صفر باشد. این شرط وضعیت بردارهای پشتیبان را مشخص می‌کند:

$$\alpha_i \times [y_i(w^T x_i + b) - 1 + \xi_i] = 0 \quad (52)$$

### ۵.۶.۳ تحلیل نهایی و نتیجه‌گیری از ترکیب شرایط

با ترکیب شرط ایستایی ( $\alpha_i = C\xi_i$ ) و شرط مکمل بودن، می‌توانیم وضعیت داده‌ها را دقیقاً دسته‌بندی کنیم:

۱. داده‌های درست دسته‌بندی شده (خارج از حاشیه):

- در این حالت خطاطا صفر است ( $\xi_i = 0$ ).
- طبق رابطه ایستایی،  $\alpha_i = C(0) = 0$ .
- این داده‌ها بردار پشتیبان نیستند و در مدل تأثیری ندارند.

۲. داده‌های دارای خطاطا یا روی مرز (بردارهای پشتیبان):

- اگر داده‌ای حتی اندکی خطاطا داشته باشد ( $0 < \xi_i$ ) یا دقیقاً روی مرز باشد.
- طبق رابطه ایستایی،  $\alpha_i = C\xi_i > 0$  می‌شود.
- طبق شرط مکمل، چون  $0 \neq \alpha_i$  است، عبارت داخل براکت باید صفر باشد:

$$y_i(w^T x_i + b) = 1 - \xi_i = 1 - \frac{\alpha_i}{C} \quad (53)$$

نتیجه مهم: در L2-SVM، هر داده‌ای که نتواند دقیقاً و کامل شرط حاشیه سخت ( $yf(x) \geq 1$ ) را ارضاء کند، بلا فاصله تبدیل به یک بردار پشتیبان با وزن  $\alpha_i > 0$  می‌شود. از آنجا که در داده‌های واقعی نویز وجود دارد، اکثر داده‌ها کمی خطاطا دارند، بنابراین معمولاً راه حل‌های غیرتُنک (Non-Sparse) تولید می‌کند که در آن تقریباً همه داده‌ها بردار پشتیبان هستند.



## ۴ پرسش پنج - محاسبه دستی SVM سخت حاشیه در $R^2$ با نمونه‌های بیشتر

### ۱.۴ بخش (آ): حدس ابرصفحه حداکثر حاشیه و معادله مرز تصمیم

تحلیل هندسی:

اگر به مختصات داده‌ها دقت کنیم، یک تقارن کامل نسبت به محور  $y$  (یا همان خط  $x_1 = 0$ ) وجود دارد:

- کلاس مثبت (۱+): تمام نقاط دارای مولفه اول  $2 \geq x_1$  هستند. کمترین مقدار  $x_1$  در این کلاس برابر با ۲ است (نقاط  $(-1, 2), (2, 1)$ ). $((2, 0), (2, 1))$ .

- کلاس منفی (۱-): تمام نقاط دارای مولفه اول  $-2 \leq x_1$  هستند. بیشترین مقدار  $x_1$  در این کلاس برابر با -۲ است (نقاط  $((-2, 0), (-2, -1), (-2, 1))$ ).

چون داده‌ها نسبت به مبدأ متقارن هستند و فاصله «لبه» کلاس مثبت تا مبدأ برابر با فاصله «لبه» کلاس منفی تا مبدأ است، ابرصفحه جداکننده بهینه (مرز تصمیم) دقیقاً در وسط این دو لبه قرار می‌گیرد.

- لبه کلاس مثبت: خط  $x_1 = 2$

- لبه کلاس منفی: خط  $x_1 = -2$

- وسط این دو خط: خط  $x_1 = 0$  (همان محور عمودی)

معادله مرز تصمیم:

معادله خط عمودی گذرنده از مبدأ به صورت زیر است:

$$x_1 = 0 \quad (54)$$

### ۲.۴ بخش (ب): محاسبه دستی ضرایب $w$ و $b$

در SVM سخت حاشیه، هدف ما پیدا کردن  $w$  و  $b$  است به طوری که شرایط زیر برقرار باشد:

$$w^T x_+ + b = 1 \quad (\text{برای بردارهای پشتیبان کلاس مثبت}) \quad (55)$$

$$w^T x_- + b = -1 \quad (\text{برای بردارهای پشتیبان کلاس منفی}) \quad (56)$$

انتخاب بردارهای پشتیبان:

نزدیک‌ترین نقاط به مرز تصمیم، بردارهای پشتیبان هستند.

- از کلاس مثبت، نقطه  $(2, 0)$  را انتخاب می‌کنیم (چون روی لبه ۲ است).

- از کلاس منفی، نقطه  $(-2, 0)$  را انتخاب می‌کنیم (چون روی لبه -۲ است).

تشکیل دستگاه معادلات:

$$\text{فرض کنید} \quad w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$



معادله ۱ (برای نقطه مثبت):  $(2, 0)$

$$w_1(2) + w_2(0) + b = 1 \implies 2w_1 + b = 1 \quad (57)$$

معادله ۲ (برای نقطه منفی):  $(-2, 0)$

$$w_1(-2) + w_2(0) + b = -1 \implies -2w_1 + b = -1 \quad (58)$$

حل دستگاه:

اگر دو معادله بالا را با هم جمع کنیم:

$$(2w_1 + b) + (-2w_1 + b) = 1 + (-1) \implies 2b = 0 \implies b = 0 \quad (59)$$

حال مقدار  $b = 0$  را در معادله اول جایگذاری می‌کنیم:

$$2w_1 + 0 = 1 \implies w_1 = \frac{1}{2} = 0.5 \quad (60)$$

برای پیدا کردن  $w_2$ , باید توجه کنیم که مرز تصمیم عمودی است، یعنی تغییر در  $x_2$  (محور عمودی) تاثیری در کلاس‌بندی ندارد.  
حال با استفاده از یکی دیگر از نقاط روی لبه (مثلاً  $(1, 0)$ ) از کلاس مثبت آن را محاسبه خواهیم کرد:

$$w_1(2) + w_2(1) + b = 1 \implies 0.5(2) + w_2(1) + 0 = 1 \implies 1 + w_2 = 1 \implies w_2 = 0 \quad (61)$$

نتیجه نهایی ضرایب:

$$w = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \quad b = 0 \quad (62)$$

معادله ابرصفحه به فرم  $w^T x + b = 0$  می‌شود:

$$0.5x_1 + 0x_2 + 0 = 0 \implies 0.5x_1 = 0 \implies x_1 = 0 \quad (63)$$

(که با حدس هندسی ما مطابقت دارد).

### ۳.۴ بخش (ج): بردارهای پشتیبان، مقدار حاشیه هندسی و فاصله دو ابرصفحه

۱. مشخص کردن بردارهای پشتیبان:

بردارهای پشتیبان نقاطی هستند که دقیقاً روی مرزهای حاشیه قرار دارند، یعنی در معادله  $1 = w^T x_i + b$  صدق می‌کنند. مرزهای حاشیه خطوط  $x_1 = 2$  و  $x_1 = -2$  هستند. تمام نقاطی که روی این دو خط قرار دارند، بردار پشتیبان هستند.

• کلاس مثبت:  $(2, 0), (2, 1), (2, -1)$

• کلاس منفی:  $(-2, 0), (-2, -1), (-2, 1)$



بنابراین ۶ نقطه ذکر شده در بالا، همگی بردارهای پشتیبان هستند. (نقاط  $(0, 3)$  و  $(3, 0)$  اگر وجود داشته باشند، دورتر هستند و بردار پشتیبان نیستند).

#### ۲. مقدار حاشیه هندسی (Geometric Margin)

حاشیه هندسی برابر است با فاصله مرز تصمیم تا نزدیکترین داده (یک طرف حاشیه). فرمول آن  $\frac{1}{\|w\|}$  است.

$$\|w\| = \sqrt{w_1^2 + w_2^2} = \sqrt{0.5^2 + 0^2} = 0.5 \quad (64)$$

$$\text{Margin Geometric} = \frac{1}{\|w\|} = \frac{1}{0.5} = 2 \quad (65)$$

(این عدد منطقی است، چون فاصله خط  $x = 0$  تا اولین داده‌ها در  $x = 2$  برابر با ۲ است).

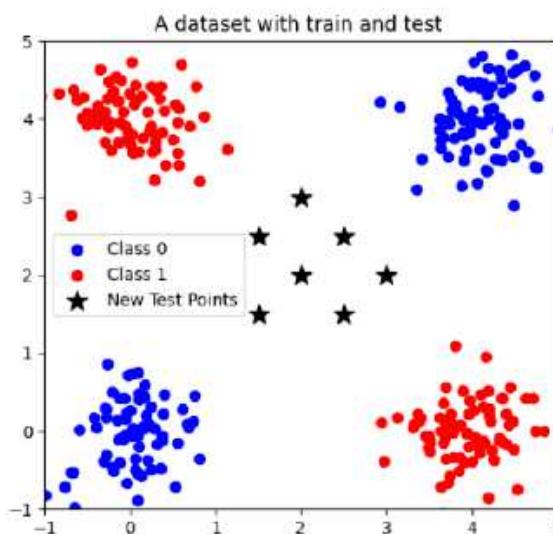
۳. فاصله دو ابرصفحه  $1$

این فاصله برابر با کل عرض حاشیه (Margin Width) است که فرمول آن  $\frac{2}{\|w\|}$  می‌باشد.

$$\text{Width Margin Total} = \frac{2}{\|w\|} = \frac{2}{0.5} = 4 \quad (66)$$

(این نیز منطقی است، چون فاصله بین خط  $x = 2$  و خط  $x = -2$  برابر با ۴ است).

## ۵ پرسش شش - طبقه‌بندی و دسته‌بندی غیرخطی



شکل ۲: دیتاست همراه با داده آموزشی و تست



## ۱.۵ انتخاب مدل مناسب

بهترین مدل برای حل این مسئله، شبکه عصبی با توابع پایه شعاعی (RBF Networks) است.

دلیل مفهومی:

داده‌های آموزشی در این مسئله دارای ویژگی «توزیع محلی» (Locality) هستند؛ به این معنی که داده‌ها در چهار خوشه (Cluster) مجزا در گوشه‌های فضای ویژگی متتمرکز شده‌اند و ناحیه مرکزی (محل قرارگیری ستاره‌های سیاه) فاقد هرگونه داده آموزشی است. شبکه‌های RBF بر اساس «فاصله» تصمیم‌گیری می‌کنند. اگر یک ورودی جدید (مانند ستاره‌های سیاه مرکزی) فاصله زیادی با داده‌های آموزشی داشته باشد، شبکه RBF به طور طبیعی خروجی نزدیک به صفر می‌دهد و آن را به عنوان ناحیه ناشناخته تشخیص می‌دهد. در مقابل، شبکه‌های MLP (با توابع فعال‌سازی Sigmoid یا ReLU) فضای را با ابرصفحه‌ها برش می‌دهند. این خطوط جداکننده معمولاً تا بین نهایت امتداد دارند و از ناحیه مرکزی عبور می‌کنند. در نتیجه، شبکه MLP برای داده‌های مرکزی که هرگز ندیده است، با اطمینان بالا (والبته غلط) یک کلاس را نسبت می‌دهد.

## ۲.۵ تحلیل ضعف مدل‌های سراسری (MLP) و اثبات ریاضی

در این بخش، تفاوت بنیادی بین «تقریب سراسری» (Local Approximation) در MLP و «تقریب محلی» (Global Approximation) در RBF را با استفاده از ریاضیات اثبات می‌کنیم.

### ۱. تحلیل و اثبات ریاضی رفتار نامناسب MLP

شبکه‌های MLP استاندارد مبتنی بر ضرب داخلی ( $w^T x$ ) هستند. این ویژگی باعث می‌شود که تصمیم‌گیری آن‌ها ماهیت سراسری داشته باشد.

تعريف ریاضی نرون MLP: فرض کنید خروجی یک نرون با تابع فعال‌سازی  $\sigma$  (مثالاً Sigmoid) به صورت زیر باشد:

$$y = \sigma(w^T x + b) \quad (67)$$

این معادله، فضای را توسط ابرصفحه  $w^T x + b = 0$  به دو نیم فضای تقسیم می‌کند.

اثبات رفتار در ناحیه مرکزی (پدیده Extrapolation): فرض کنید شبکه یادگرفته است که کلاسی در ناحیه  $x_1 > x_2 > \dots > x_n$  (خوشه بالا-راست) وجود دارد. برای پوشش این ناحیه، شبکه خطی رسم می‌کند که برای مقادیر بزرگ  $x$  فعال شود. رفتار حدی این تابع در بین نهایت به صورت زیر است:

$$\lim_{x \rightarrow +\infty} \sigma(wx + b) = 1, \quad \lim_{x \rightarrow -\infty} \sigma(wx + b) = 0 \quad (68)$$

این تابع در تمام فضای بینایی از منفی بی نهایت تا مثبت بی نهایت مقداری بین ۰ و ۱ دارد. اگر داده تست (ستاره سیاه) در مبدأ مختصات  $(0, 0)$  باشد و هیچ مرز آموزشی مسدودکننده‌ای دقیقاً در اطراف آن نباشد، مقدار فعالیت نرون‌ها غیر صفر خواهد بود. معمولاً برای حل مسئله چهار گوشه، MLP دو خط موازی رسم می‌کند. فضای بین این دو خط (که مرکز را شامل می‌شود) دارای فعالیت بالاست:

$$\text{Output}_{\text{MLP}}(0, 0) \neq 0 \quad (69)$$



این یعنی MLP برای ناحیه‌ای که هیچ داده‌ای ندیده است، «نظر قطعی» می‌دهد. این پدیده «برون‌یابی» (Extrapolation) نام دارد و در اینجا خطرناک است.

## ۲.۲.۵ تحلیل و اثبات ریاضی رفتار مناسب RBF

شبکه‌های RBF بر اساس نرم اقلیدسی (فاصله) کار می‌کنند. این ویژگی باعث «محلى» بودن آن‌ها می‌شود. تعریف ریاضی نرون RBF: خروجی شبکه تابعی از فاصله ورودی  $x$  تا مراکز یاد گرفته شده  $c_i$  است:

$$f(x) = \sum_{i=1}^N w_i \phi(\|x - c_i\|) \quad (70)$$

که معمولاً  $\phi$  یک تابع گوسی است:

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad (71)$$

اثبات خاموشی در ناحیه ناشناخته (Decay Property): فرض کنید مراکز  $c_i$  در چهار گوشه مختصات قرار دارند (داده‌های آموزشی). حال ورودی تست جدید (ستاره سیاه) در مرکز مختصات  $x_{test} = (0, 0)$  قرار می‌گیرد. فاصله اقلیدسی این نقطه تا هر یک از مراکز  $c_i$  زیاد است ( $r \gg 0$ ). حد تابع گوسی در فاصله دور برابر است با:

$$\lim_{r \rightarrow \infty} \exp\left(-\frac{r^2}{2\sigma^2}\right) = 0 \quad (72)$$

بنابراین خروجی کل شبکه برای نقطه مرکزی برابر است با:

$$f(0, 0) \approx \sum w_i \cdot (0) = 0 \quad (73)$$

نتیجه اثبات: ریاضیات نشان می‌دهد که تابع گوسی دارای خاصیت «میرا بودن» (Decay) است. وقتی ورودی از مراکز داده‌های آموزشی دور می‌شود، فعالیت نرون‌ها به سمت صفر میل می‌کند. این یعنی شبکه RBF در مواجهه با ستاره‌های سیاه در مرکز، خروجی صفر می‌دهد که تفسیر آن «عدم تعلق به هیچ یک از کلاس‌ها» یا «عدم اطمینان» است.

## ۳.۵ خلاصه و نتیجه‌گیری نهایی

مدل	نوع تقریب	ریاضیات پایه	رفتار در ناحیه مرکزی (ستاره‌ها)
MLP	سراسری (Global)	ضرب داخلی $w^T x$	مقدار بالا و غیر صفر (توهم آگاهی)
RBF	محلى (Local)	فاصله اقلیدسی $\ x - c\ $	میل به صفر (تشخیص صحیح ناشناخته)

بنابراین، ضعف مدل‌های MLP در تفکیک خوشه‌های محلی این است که مرزهای تصمیم آن‌ها نامحدود (Unbounded) است، در حالی که RBF مرزهای تصمیم بسته و محدود (Bounded) ایجاد می‌کند که برای داده‌های خوشه‌ای و جزیره‌ای ایده‌آل است.



## ۶ بخش اول - پیاده‌سازی RBFF

این بخش شامل تولید داده‌ها، پیاده‌سازی شبکه عصبی RBF است، آموزش با روش کمترین مربعات (LLS) و  $L_2$ -regularization، تحلیل نتایج برای پارامترهای مختلف  $K$  و  $\sigma$  است. بدین منظور ابتدا همانطور که از ما خواسته شده است دیتاست را به صورت زیر تولید می‌نماییم:

### ۱.۶ ایجاد دیتاست

سیگنال ورودی ( $u(t)$ ) به صورت سینوسی تعریف شده است:

$$u(t) = \sin(2\pi t/250), \quad (74)$$

و مقادیر اولیه  $y(0) = y(1) = 0$  در نظر گرفته می‌شوند. تکرار برای  $t = 999$  تا  $t = 2$  انجام می‌شود تا دنباله  $y(t)$  تولید شود. سپس مجموعه داده نمونه‌ها به صورت زوج‌های ورودی-هدف ساخته می‌شود که در هر نمونه

$$\mathbf{x}(t) = \begin{bmatrix} y(t-1) \\ y(t-2) \\ u(t-1) \end{bmatrix}, \quad \text{target} = y(t) \quad (75)$$

می‌باشد.

کد پایتون زده شده برای تولید این دیتاست به صورت زیر می‌باشد:

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4 import seaborn as sns
5
6 np.random.seed(42)
7 samples = 1000
8
9 total_steps = samples + 10 # plus 10 to prevent index error
10 y = np.zeros(total_steps)
11 u = np.zeros(total_steps)
12
13 # initial conditions
14 y[0] = 0
15 y[1] = 0
16
17 # input signal: sine wave
18 t_ = np.arange(total_steps)
19 u = np.sin(2 * np.pi * t_/250)
20
21 data_ = []
22 t = 2 # start from t=2 as per problem statement

```



```

۲۳ count = 0
۲۴
۲۵ while count < samples:
۲۶     # system formula: y(t) = y(t-1)/(1 + y(t-2)) + u(t-1)^3
۲۷     denom = 1 + y[t - 2]
۲۸
۲۹     # avoid division by zero
۳۰     if abs(denom) < 1e-7:
۳۱         denom = 1e-7
۳۲
۳۳     num = y[t-1]
۳۴     p1 = num / denom
۳۵     p2 = u[t-1]**3
۳۶     y[t] = p1 + p2
۳۷
۳۸     # create dataset row
۳۹     data_rows = {
۴۰         'Y[t-1]': y[t-1],
۴۱         'Y[t-2]': y[t-2],
۴۲         'U[t-1]': u[t-1],
۴۳         'Y[t]': y[t]
۴۴     }
۴۵
۴۶     data_.append(data_rows)
۴۷     count += 1
۴۸     t += 1
۴۹
۵۰ df = pd.DataFrame(data_)
۵۱ df.head()
۵۲
۵۳ df.to_csv('1000Sample_data_ball&beam.csv', index=False)

```

۱: تولید ۱۰۰۰ نمونه برای ایجاد دیتاست beam ball Code

۵ سطر اول دیتاست تولیدی به صورت زیر می‌باشد:

...	Y[t-1]	Y[t-2]	U[t-1]	Y[t]
0	0.000000	0.000000	0.025130	0.000016
1	0.000016	0.000000	0.050244	0.000143
2	0.000143	0.000016	0.075327	0.000570
3	0.000570	0.000143	0.100362	0.001581
4	0.001581	0.000570	0.125333	0.003549

شکل ۳: ۵ سطر اول



قبل از اینکه به بخش بعدی برویم مقداری ویژگی‌ها و تاثیر آنها بر روی خروجی را بررسی می‌نماییم. این بررسی به صورت ترسیم نمودارهایی به شکل زیر انجام می‌پذیرد. کد زده شده و نمودارهای گرفته شده در این بخش به ترتیب ارائه می‌گردند:

```
 1 data = pd.read_csv('1000Sample_data_ball&beam.csv')
 2 df = data
 3 X = data[['Y[t-1]', 'Y[t-2]', 'U[t-1]']]
 4 Y = data[['Y[t]]']
 5 # yekam plot mikonam data haro baresinkonam
 6 plt.figure(figsize=(16, 12))
 7
 8 #raftar zamani ro negah konim ba ye bakhshi az data
 9 # mikham bebinam aya out nesbat be input vakoneshi dare?
10 ax1 = plt.subplot(2, 2, 1)
11 subset = df.iloc[:250]
12 ax1.plot(subset.index, subset['Y[t]'], label='Output y(t)', color="#1f77b4", linewidth=2)
13 ax1.plot(subset.index, subset['U[t-1]'], label='Input u(t-1)', color="#ff7f0e", linestyle='--', alpha=0.7)
14 ax1.set_title('1. Time Series Response (First 250 samples)', fontsize=14)
15 ax1.set_xlabel('Time Step')
16 ax1.set_ylabel('Amplitude')
17 ax1.legend()
18
19
20
21 #phas ro barresi konam. rabete recurrent y(t) va y(t-1)
22 ax2 = plt.subplot(2, 2, 2)
23 sc = ax2.scatter(df['Y[t-1]'], df['Y[t]'], c=df['U[t-1]'], cmap='viridis', alpha=0.6, s=15)
24 ax2.set_title('2. Phase Portrait (y[t] vs y[t-1])\nColor = Input u', fontsize=14)
25 ax2.set_xlabel('Previous Position y(t-1)')
26 ax2.set_ylabel('Current Position y(t)')
27 plt.colorbar(sc, ax=ax2, label='Input u(t-1)')
28
29
30 #barresi 3d data
31 ax3 = plt.subplot(2, 2, 3, projection='3d')
32 p3d = ax3.scatter(df['Y[t-1]'], df['Y[t-2]'], df['Y[t]'], c=df['Y[t]'], cmap='plasma', s=5)
33 ax3.set_title('3. 3D Manifold (The Learning Target)', fontsize=14)
34 ax3.set_xlabel('y(t-1)')
35 ax3.set_ylabel('y(t-2)')
36 ax3.set_zlabel('y(t)')
37 ax3.view_init(elev=20, azim=45)
38
39
40 # statistical distribution data ro check konam
41 ax4 = plt.subplot(2, 2, 4)
42 sns.histplot(df['Y[t]'], kde=True, color='purple', ax=ax4, label='Output y(t)')
```



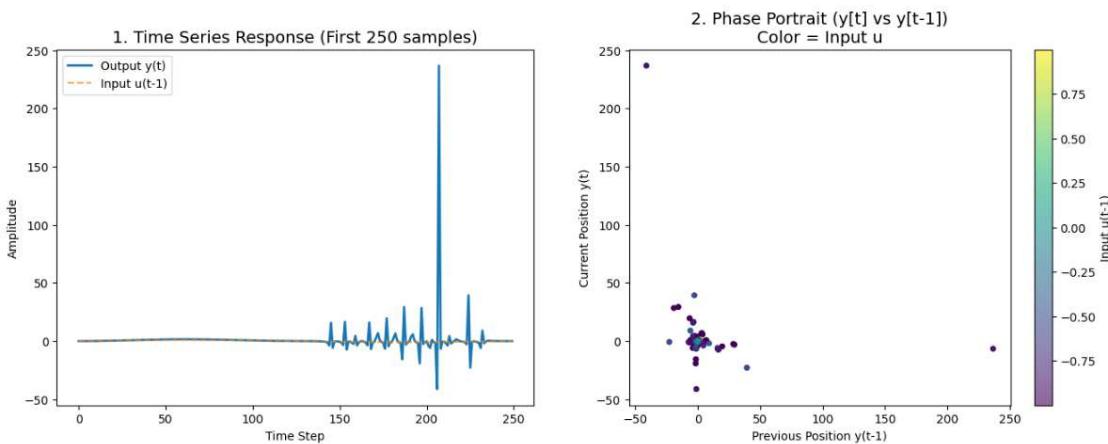
```

۴۳ #                                         )Singularity)
۴۴ ax4.axvline(-1, color='red', linestyle='--', label='Singularity Limit (-1)')
۴۵ ax4.set_title('4. Data Distribution & Safety Check', fontsize=14)
۴۶ ax4.set_xlabel('Value')
۴۷ ax4.legend()

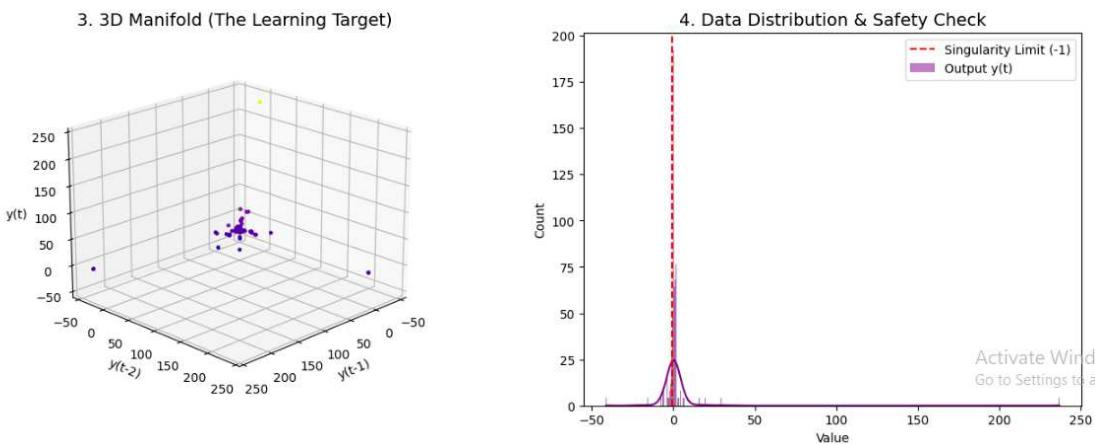
```

## ۲: تحلیل و بررسی اثر ویژگی‌ها بر روی خروجی Code

خروجی‌های گرفته شده به صورت زیر می‌باشند:



شکل ۴: ترسیم خروجی به همراه ورودی  $(t-1)$  و نمایش موقعیت  $y(t)$  نسبت به  $y(t-1)$  برای ۲۵۰ نمونه اول



شکل ۵: نمایش سه بعدی خروجی نسبت به ویژگی‌های  $y(t-1)$  و  $y(t-2)$  به همراه رسم نمودار توزیع دیتا برای پیدا کردن نقطه تکینگی

نمودار اول:

در ۱۴۰ گام اول، مقدار  $u$  بین  $-0.5$  تا  $0.5$  نوسان می‌کند. این منطقه "امن" است (دور از  $-1$ ). در حوالی گام ۱۴۰، احتمالاً به دلیل جمع شدن اثر ورودی‌ها، مقدار  $u$  کمی پایین‌تر می‌رود و به مثلاً  $-0.8$  یا  $-0.9$  می‌رسد. ناگهان اثر "مخرج کوچک" فعال می‌شود. سیستم یک لگد محکم می‌خورد و  $u$  به عدد مثبت ۳۰ پرتاپ می‌شود.



چون سیستم دینامیک است (به گذشته وابسته است)، این عدد بزرگ ۳۰ در گام بعدی وارد معادله می‌شود و سیستم را کاملاً بثبات می‌کند (نوسانات شدید مثبت و منفی پی درپی).

#### نمودار دوم و سوم:

هسته مرکزی: آن توده بنفس تیره، رفتار "سالم" سیستم است. جایی که توپ به آرامی حرکت می‌کند.

نقاط دوردست (Outliers): آن نقاط زرد و سبز که خیلی دور افتاده‌اند، همان لحظات انفجار هستند.

#### نمودار چهارم:

داده‌های بنفس رنگ از خط قرمز عبور کرده‌اند.

این یعنی توپ فرضی ما در این شبیه‌سازی، از نقطه فیزیکی غیرممکن عبور کرده است. در واقعیت فیزیکی، شاید این به معنای برخورد توپ با انتهای میله یا شکستن مکانیزم باشد.

سیستم بارها و بارها به نقطه تکینگی (Singularity) برخورد کرده است.

Singularity یا تکینگی به نقطه‌ای گفته می‌شود که در آن، یک تابع یا معادله رفتار "خوش‌تعريف" خود را از دست می‌دهد؛ یعنی مقدارش به سمت بینهایت می‌رود یا اصلاً قابل محاسبه نیست.

## ۲.۶ تقسیم داده‌ها به دو بخش آموزش و تست

در این بخش سوال از ما خواسته است که ۷۰۰ نمونه اول را به عنوان آموزش و ۳۰۰ نمونه بعدی را به عنوان تست استفاده نماییم. با توجه به اینکه مدل اصلی کاملاً وابستگی نسبت به مقادیر گام‌های قبلی خود دارد، باید توجه نماییم که دقیقاً به ترتیب ۷۰۰ نمونه اول به مدل داده شود تا یادگیری به شکل درستی صورت پذیرد. بنابراین در کد زیر shuffle=False قرار می‌دهیم.

کد زده شده به صورت زیر می‌باشد:

```
1 # data hamon ro motabegh khaste soal split kardam
2
3 from sklearn.model_selection import train_test_split
4 X = data[["Y[t-1]", "Y[t-2]", "U[t-1]"]].values
5 Y = data[["Y[t]"]].values
6
7 X_train, X_test, y_train, y_test = train_test_split(
8     X, Y,
9     test_size=0.3,
10    random_state=42,
11    shuffle=False
12 )
13
14 print(f"{X_train.shape}")
15 print(f"{X_test.shape}")
```

۳: Code تقسیم داده‌ها به آموزش و تست

خروجی کد بالا به صورت زیر می‌باشد:

(700, 3), (300, 3)



### ۳.۶ مساله تقریب تابع

این مساله یک مساله تقریب تابع است چرا که ما در تلاش هستیم سیستمی را که مدل دینامیکی آن را داریم و به صورت فرمول در صورت سوال بیان شده است به وسیله RBFNN تقریب بزنیم. در واقع به دنبال این هستیم که به وسیله RBFNN رابطه زیر را به دست بیاوریم و مدلی داشته باشیم که با گرفتن سه ورودی خروجی را برای ما محاسبه کند و دیگر نیازی به استفاده از فرمول زیر نداشته باشیم به عبارت دیگر به جای اینکه فرمول محور مساله را حل کنیم در تلاش هستیم که با دیتا این مساله را حل کنیم. منطقی است که مقداری خطای هنگام تخمین زدن این مدل به وسیله RBFNN وجود دارد و این مدل به صورت کامل نمی‌تواند هنگام گرفتن ورودی، خروجی اصلی را پیش‌بینی کند.

$$y(t) = \frac{y(t-1)}{1+y(t-2)} + u(t-1)^3 \quad (76)$$

### ۴.۶ بخش دوم: پیاده‌سازی RBFNN ایستا

شبکه‌ی RBFNN ایستا حافظه ندارد اما سیستم مورد نظر ما یک سیستم دینامیکی است.

راه حل: استفاده از مدل NARX

برای ایستا کردن سیستم، ورودی‌های گذشته را به عنوان ورودی جدید شبکه تعریف می‌کنیم. به این ترتیب بردار ورودی مدل به صورت زیر خواهد بود:

$$X(t) = [y(t-1), y(t-2), u(t-1)]$$

در شبکه‌ی RBF این بردار تنها یک ورودی سه‌بعدی است و هیچ مفهوم زمانی در آن وجود ندارد. به این ترتیب مدل دینامیکی به یک نگاشت کاملاً ایستا تبدیل می‌شود:

$$f(X(t)) = y(t)$$

به عبارت دیگر، شبکه تنها یک تابع ایستا را تقریب می‌زند، اما از آنجا که ورودی شامل نمونه‌های گذشته‌ی سیستم است، رفتار دینامیکی اصلی حفظ می‌شود. این کاری است که تا به اینجا انجام دادیم.

ساختمان شبکه RBFNN به صورت زیر می‌باشد:

$$f(x) = \sum_{k=1}^K \alpha_k \phi_k(x)$$

$$\phi_k(x) = \exp\left(-\frac{1}{\sigma_k^2} \|x - \mu_k\|^2\right)$$

حال در این بخش از ما خواسته شده تا نقش مرکز نورون ( $\mu_k$ ) و عرض ( $\sigma_k$ ) آن را توضیح دهیم و بیان نماییم که کم و زیاد شدن هر پارامتر چگونه بر پاسخ نورون به یک ورودی اثر می‌گذارد.

تابع فعال‌سازی گوسی در شبکه‌های RBF به صورت بالا تعریف نمودیم.

در این رابطه دو پارامتر اصلی وجود دارد: مرکز نورون  $\mu_k$  و عرض یا پهنه‌ای آن  $\sigma_k$ .

۱. مرکز نورون ( $\mu_k$ )

مرکز نورون نشان می‌دهد که این نورون در کدام نقطه از فضای ورودی بیشترین تحریک را دارد. اگر ورودی برابر مرکز باشد:

$$x = \mu_k \quad \Rightarrow \quad \phi_k(x) = 1$$



در نتیجه تغییرات زیر رخ می‌دهد:

- هرچه ورودی  $x$  به  $\mu_k$  نزدیک‌تر باشد مقدار  $\phi$  بزرگ‌تر است.
- دور شدن ورودی از مرکز باعث کاهش سریع مقدار تحریک نورون می‌شود.
- تغییر  $\mu_k$  موقعیت نورون را در فضای ورودی جابه‌جا می‌کند.

## ۲. عرض نورون ( $\sigma_k$ )

پارامتر  $\sigma_k$  میزان گستردگی یا پهنایتابع گوسی را تعیین می‌کند.

اثر افزایش  $\sigma_k$ :

- تابع گوسی پهن‌تر می‌شود.
- نورون نسبت به ناحیه بزرگ‌تری از ورودی‌ها پاسخ می‌دهد.
- حساسیت نورون کمتر می‌شود و رفتار آن عمومی‌تر می‌گردد.

اثر کاهش  $\sigma_k$ :

- تابع گوسی تیز‌تر و باریک‌تر می‌شود.
- نورون تنها به ورودی‌هایی که بسیار نزدیک  $\mu_k$  هستند پاسخ قوی می‌دهد.
- حساسیت نورون بیشتر شده و رفتار آن انتخاب‌گرانه‌تر می‌شود.

## ۵.۶ تعریف تابع RBF فعال‌ساز گاوسی و تست یک ورودی

تابع RBF فعال‌ساز گاوسی را در کد زیر تعریف نمودیم:

```

1 def rbf_kernel (x, mu, sigma):
2     euclidean_distances = np.sum((x - mu) ** 2 )
3     return np.exp(-euclidean_distances / (sigma ** 2))

```

۴: تابع RBF فعال‌ساز گاوسی

حال همانطور که سوال از ما خواسته برای ورودی به مرکز یکسان  $[1, 1]$  و عرض ۱ تابع را تست می‌نماییم. کد زیر زده شده است:

```

1 x_test = np.array([1, 1])
2 mu_test = np.array([1, 1])
3 sigma = 1.0
4
5 result = rbf_kernel(x_test, mu_test, sigma)

```



۶ result

## Code ۵: تست تابع RBF فعال‌ساز گاوسی

همانطور که بالاتر اشاره کردیم، چون ورودی و مرکز دقیقاً بر روی یکدیگر قرار دارند، فاصله اقلیدسی بینشان برابر با صفر خواهد شد و پاسخ نهایی تابع فعال‌ساز گاوسی برابر با یک خواهد شد که پاسخ این موضوع را تایید می‌کند. این خروجی نیز به این معناست که نورون بیشترین تحریک را نسبت به ورودی خود دارد و هنگامی که این ورودی را بیند با ماکریم مقدار خود فعال خواهد شد. هرچه از این مقدار که برابر با یک است فاصله بگیریم نورون تحریک کمتری را نسبت به آن ورودی نشان داده است.

## 6.6 بخش سوم: آموزش از طریق حداقل مربعات خطی (LLS)

در این بخش ابتدا از ما خواسته شده است تا با زدن کدی ماتریس  $\phi$  را تشکیل دهیم. تابعی که برای ما این ماتریس را می‌سازد به صورت زیر خواهد بود:

```
1 def phi_matrix (x , centers, sigma):
2     n = x.shape[0] #x = n*3
3     k = centers.shape[0] #k = k*3
4     phi = np.zeros((n,k))
5
6     for i in range(n):
7         for j in range(k):
8             phi[i, j] = rbf_kernel(x[i], centers[j], sigma)
9
10    return phi
```

Code 6: تابع سازنده ماتریس  $\phi$ 

مقدار  $K$  را برابر با  $200$  قرار می‌دهیم. همانطور که صورت سوال از ما خواسته است مقدار  $\sigma_k$  را از میانگین فاصله بین مرکز حساب می‌نماییم. این را به عنوان یک نقطه شروع در نظر می‌گیریم تا توانیم مدل خودمان را بسازیم. این کار به صورت تابع زیر نوشته شده است:

```
1 def sigma_(centers):
2     k = centers.shape[0]
3     if k <= 1:
4         return 1
5
6     total_distance = 0
7     count = 0
8
9     for i in range(k):
10        for j in range(i+1, k):
11            dist = np.sqrt(np.sum((centers[i] - centers[j])**2))
12            total_distance += dist
13            count += 1
14
15     average_distance = total_distance / count
```



```
16     return average_distance
```

۷: تابع محاسبه  $\sigma$  Code

۲۰۰ مرکز به صورت تصادفی توسط کد زیر انتخاب شده‌اند:

```
1 k = 200
2 np.random.seed(42)
3 random_indices = np.random.choice(len(X_train), k, replace=False)
4 centers = X_train[random_indices]
5 centers
```

## ۸: به دست آوردن مرکز

مقدار سیگما را با دادن این مرکز به تابع `sigma` محاسبه می‌کنیم. خروجی این تابع برابر با ۰.۸۳۲۸۵۴۷۴۷۰۱۴۳۴۲۸ است. پیش از آنکه داده‌های آموزش، مرکز و  $\sigma_k$  را به تابع `phi_matrix` بدهیم، باید روی همه داده‌ها نرمال‌سازی انجام دهیم. این نرمال‌سازی روی خروجی‌ها نیز انجام می‌شود؛ زیرا هنگامی که بخواهیم با روش LLS بردار  $\alpha$  را به دست آوریم، بزرگی مقادیر خروجی باعث می‌شود وزن‌ها نیز بزرگ شوند. در نتیجه حساسیت مدل برای تولید خروجی افزایش می‌یابد و تغییرات اندک در ورودی باعث تغییرات بزرگ در خروجی می‌شود. برای جلوگیری از چنین حساسیتی، نرمال‌سازی روی خروجی‌ها نیز اعمال می‌شود.

```
1 # data ro normalize mikonam mikonam if needed
2
3 min_val = np.min(X_train, axis=0)
4 max_val = np.max(X_train, axis=0)
5
6 range_val = max_val - min_val
7 range_val[range_val == 0] = 1
8
9 def normalize(data, min_v, range_v):
10     scaled_0_1 = (data - min_v) / range_v
11     scaled_neg1_pos1 = (scaled_0_1 * 2) - 1
12     return scaled_neg1_pos1
13
14 X_train = normalize(X_train, min_val, range_val)
15
16 X_test = normalize(X_test, min_val, range_val)
17
18
19 y_min = np.min(y_train)
20 y_max = np.max(y_train)
21 y_range = y_max - y_min
22
23 def normalize_y(data, min_v, range_v):
24     return (data - min_v) / range_v
25
26 y_train = normalize_y(y_train, y_min, y_range)
```



```
۷۷ y_test = normalize_y(y_test, y_min, y_range)
```

## ۹: نرمالسازی Code

حال مقادیر آموزش، مراکز و  $\sigma_k$  را به تابع `phi_train` بدهست می‌آید:

```
۱ phi_train = phi_matrix(X_train, centers, sigma)
۲ phi_train
```

## ۱۰: نرمالسازی Code

```
*** array([[0.41643706, 0.99999986, 0.63362214, ..., 0.5530614 , 0.33677008,
0.34659468],
[0.39322435, 0.99909084, 0.60831228, ..., 0.57881015, 0.35831273,
0.36845745],
[0.37065895, 0.99637332, 0.58298433, ..., 0.60462421, 0.38051216,
0.39095864],
...,
[0.91597307, 0.23636468, 0.69012244, ..., 0.02091587, 0.00663176,
0.00705494],
[0.85748546, 0.22697192, 0.74053719, ..., 0.02040121, 0.00651462,
0.00692766],
[0.88921723, 0.2381707 , 0.69490902, ..., 0.02173352, 0.00698442,
0.0074247 ]])
```

شکل ۶: ماتریس `phi_train`

در ادامه با کد زیر مقدار  $\alpha$  به دست می‌آید:

```
۱ # NumPy Linear Algebra Pseudo-Inverse = (phi_transpose * phi)^-1 * phi_transpose
۲ # calculate weights by lls and using psuedo inverse
۳ alpha = np.linalg.pinv(phi_train).dot(y_train)
۴ alpha.shape
```

۱۱: محاسبه ماتریس `phi_train` Code

ابعاد  $\alpha$  به صورت زیر است:

(200, 1)

ابعاد `phi_train` به صورت زیر است:

(700, 200)

۵ مقدار اول بردار  $\alpha$  به صورت زیر است:

```
array([-4.05909051e+09, -1.32861853e+10,  2.76020470e+07, -2.10113678e+07,
-4.36529036e+09])
```

شکل ۷: بردار  $\alpha$



در ادامه مقدار ریشه میانگین مربعات خطأ را پس از پیش‌بینی خروجی‌های مدل آموزش دیده روی داده‌های آموزش و تست به دست می‌آوریم:

```

1 y_pred_train = phi_train.dot(alpha)
2
3 phi_test = phi_matrix(X_test, centers, sigma)
4 y_pred_test = phi_test.dot(alpha)
```

۱۲: پیش‌بینی خروجی‌های مدل بر روی داده‌های آموزش و تست

تابعی که برای محاسبه RMSE نوشته شده به صورت زیر است:

```

1 def rmse(y_true, y_pred):
2     mse = np.mean((y_true - y_pred)**2)
3     return np.sqrt(mse)
```

RMSE: ۱۳: محاسبه

```

1 rmse_train = rmse(y_train, y_pred_train)
2 rmse_test = rmse(y_test, y_pred_test)
3
4 print(f"Result for Train: {rmse_train}")
5 print(f"Result for Test: {rmse_test}")
```

RMSE: ۱۴: محاسبه

Result for Train: 0.011833206150038801 (۷۷)

Result for Test: 0.015397151728820144 (۷۸)

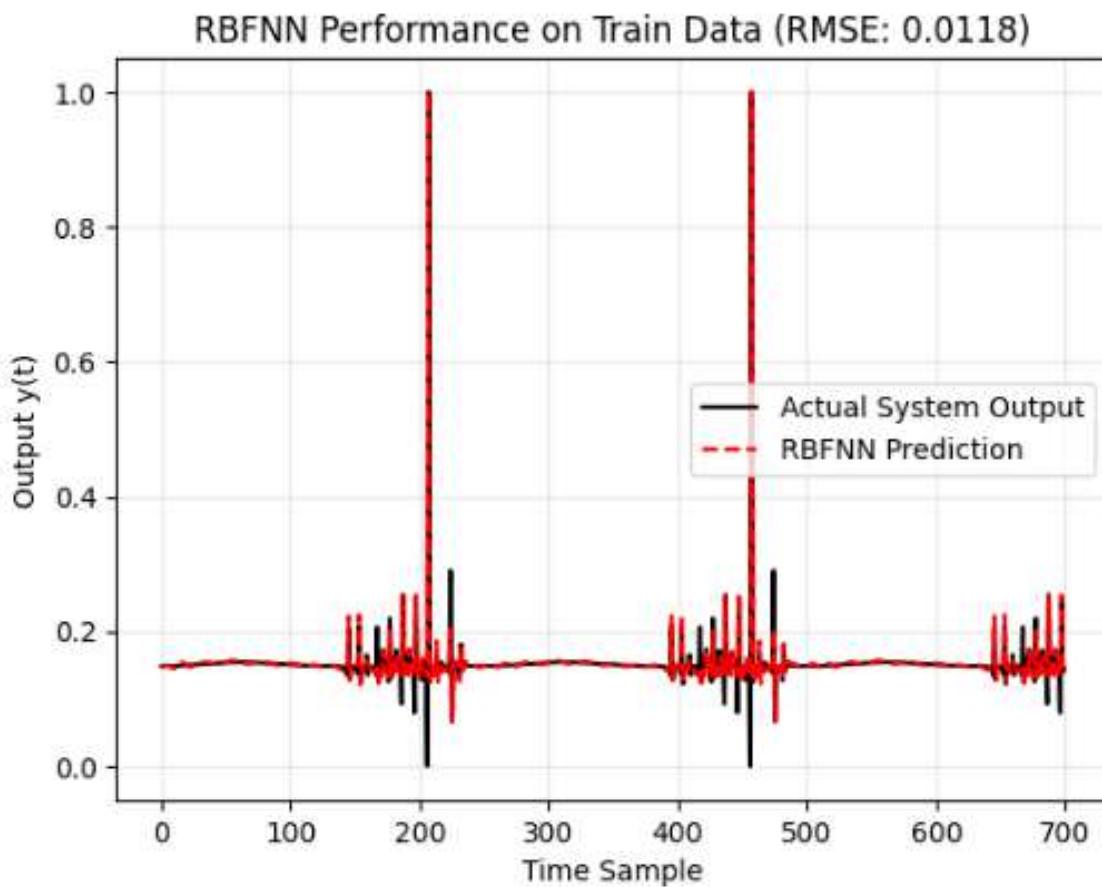
حال نمودار خروجی مربوط پیش‌بینی مدل و خروجی واقعی را با کد زیر به دست می‌آوریم:

```

1 limit = 700
2 plt.plot(y_train[:limit], label='Actual System Output', color='black', linewidth=1.5)
3 plt.plot(y_pred_train[:limit], label='RBFNN Prediction', color='red', linestyle='--',
4 linewidth=1.5)
5
6 plt.title(f'RBFNN Performance on Train Data (RMSE: {rmse_train:.4f})')
7 plt.xlabel('Time Sample')
8 plt.ylabel('Output y(t)')
9 plt.legend()
10 plt.grid(True, alpha=0.3)
11 plt.show()
```

۱۵: نمایش خروجی پیش‌بینی شده به همراه خروجی واقعی برای دیتای آموزش

خروجی این کد به صورت زیر خواهد بود:



شکل ۸: خروجی مربوط به پیش‌بینی مدل به همراه خروجی واقعی برای دیتای آموزش

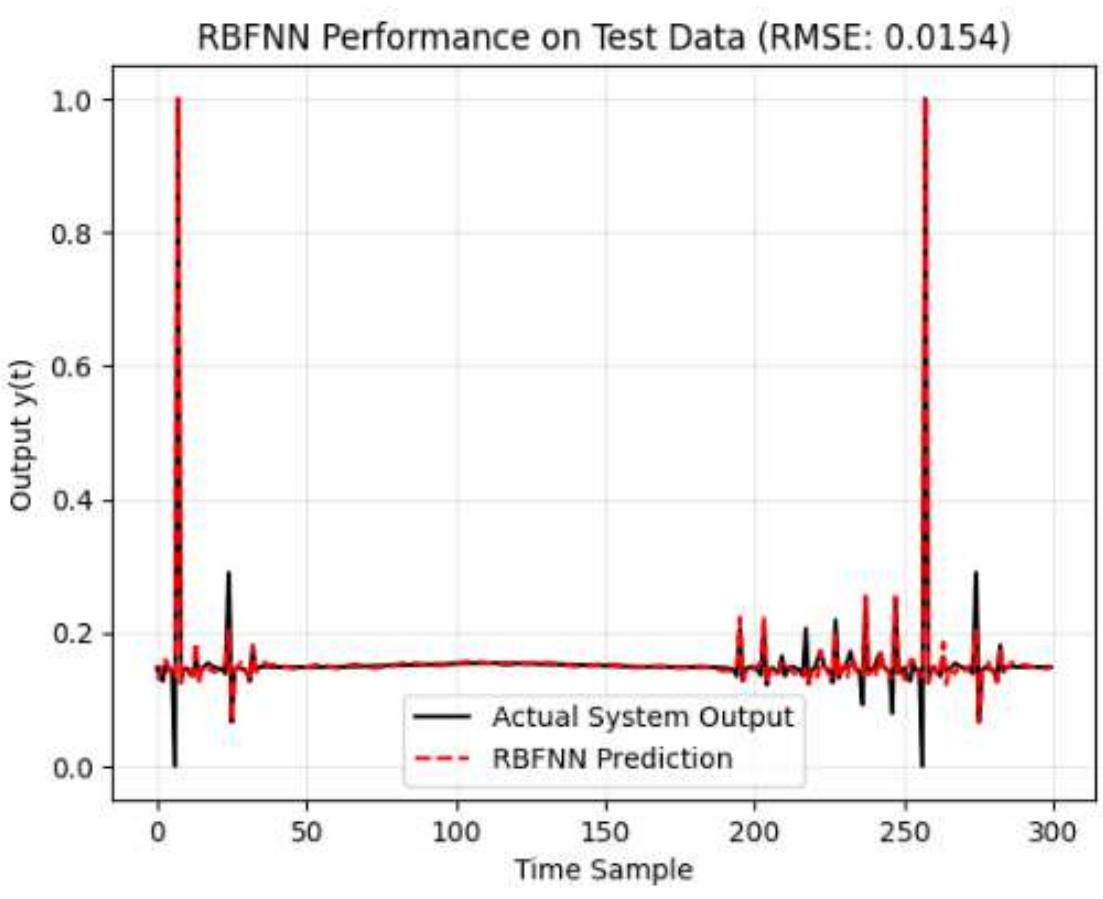
برای دیتای تست نیز داریم:

```

` 1 limit = 300
` 2 plt.plot(y_test[:limit], label='Actual System Output', color='black', linewidth=1.5)
` 3 plt.plot(y_pred_test[:limit], label='RBFNN Prediction', color='red', linestyle='--',
`           linewidth=1.5)

` 4
` 5 plt.title(f'RBFNN Performance on Test Data (RMSE: {rmse_test:.4f})')
` 6 plt.xlabel('Time Sample')
` 7 plt.ylabel('Output y(t)')
` 8 plt.legend()
` 9 plt.grid(True, alpha=0.3)
`10 plt.show()
```

۱۶: نمایش خروجی پیش‌بینی شده به همراه خروجی واقعی برای دیتای تست



شکل ۹: خروجی مربوط به پیش‌بینی مدل به همراه خروجی واقعی برای دیتای تست

همانطور که می‌بینیم تا حد بسیار زیادی مدل توانسته است خروجی را به خوبی پیش‌بینی کند و نقطه چین تقریباً بر روی خطوط مشکی افتاده است اما با این حال در جاهایی نتوانسته است خروجی درست را تشخیص دهد که بیشترین آن در زمان‌هایی است که نوسانات زیادی با دامنه متوسط و یا نسبتاً کم داریم. در فرکانس بالا با بزرگترین دامنه نیز توانسته‌ایم که خروجی را به خوبی پیش‌بینی نماییم. دلیلی که این اتفاق می‌افتد به شرح زیر است:

نقاطی از داده که دارای نوسان با دامنه متوسط یا کم هستند معمولاً حالتی را ایجاد می‌کنند که در آن

$$y(t-2) \approx -1,$$

و در نتیجه سیستم در نزدیکی یک تکینگی قرار می‌گیرد. در چنین شرایطی جملهٔ

$$\frac{y(t-1)}{1 + y(t-2)}$$

رفتاری بسیار حساس و شبیه‌انفجاری پیدا می‌کند؛ زیرا هرچه  $y(t-2) + 1$  به صفر نزدیک‌تر شود، خروجی تابع نسبت به کوچکترین تغییرات در  $y(t-1)$  یا  $y(t-2)$  واکنش بسیار شدیدی نشان خواهد داد.

این حساسیت عددی سبب می‌شود که حتی خطاهای بسیار کوچک در ورودی شبکه یا نبود داده آموزشی کافی در این ناحیهٔ خاص، به خطاهای خروجی بسیار بزرگ تبدیل شوند. به همین دلیل شبکه RBF که ساختاری استادار و باسته به مراکز  $\mu_k$  و پهناهای  $\sigma_k$  است، نمی‌تواند این ناحیهٔ محلی بسیار حساس را به خوبی تقریب بزند و در نتیجه خطأ در این بخش افزایش می‌یابد.



به بیان کوتاه‌تر، هرگاه

$$1 + y(t-2) \rightarrow 0,$$

تابع هدف شبیه سیار تند و رفتاری غیرخطی شدید پیدا می‌کند؛ بنابراین شبکه RBF با تنظیمات فعلی قادر به مدل‌سازی دقیق این قسمت نیست. این موضوع کاملاً مطابق با مشاهده تجربی است که بیشترین خطاهای در زمان‌هایی رخ داده‌اند که نوسانات خروجی دامنه متوسط یا کم داشته‌اند؛ زیرا همین نوسانات می‌توانند مقدار  $(2-t)y$  را در حوالی ۱- قرار دهند که همان ناحیه تکینگی سیستم است.

## ۷.۶ سریع بودن LLS نسبت به backpropagation

فرآیند آموزش شبکه‌های RBF مبتنی بر روش کمترین مربعات خطی (Linear Least Squares, LLS) (به مراتب سریع‌تر از آموزش مبتنی بر گرادیان‌کاهشی و پس‌انتشار خطای پرسپترون‌های چندلایه) است. دلیل این تفاوت، ماهیت کاملاً متفاوت مسئله‌ی بهینه‌سازی در دو مدل است.

در شبکه‌ی RBF، پس از تعیین مراکز و پهناهای توابع شعاعی، خروجی شبکه نسبت به وزن‌ها کاملاً خطی است و به صورت زیر نوشته می‌شود:

$$\hat{y}(t) = \sum_{k=1}^K \alpha_k \phi_k(X(t)).$$

به دلیل خطی بودن مدل نسبت به ضرایب  $\alpha_k$ ، تابع خطایک تابع سه‌می‌گون و کاملاً محدب است و نقطه‌ی بهینه‌ی وزن‌ها را می‌توان به صورت مستقیم و بدون نیاز به الگوریتم‌های تکراری از رابطه‌ی بسته‌ی زیر به دست آورد:

$$\alpha = (\Phi^T \Phi)^{-1} \Phi^T Y.$$

این ویژگی باعث می‌شود آموزش تنها با یک بار حل یک دستگاه معادلات خطی انجام شود؛ بنابراین:

- نیازی به تکرارهای متعدد نیست
- نیازی به انتخاب نرخ یادگیری وجود ندارد
- مسئله هیچ مینیمم محلی ندارد
- پاسخ دقیق در یک مرحله محاسبه می‌شود

اما در پرسپترون‌های چندلایه به دلیل وجود توابع فعال‌سازی غیرخطی و ترکیب چندلایه‌ای وزن‌ها، خروجی شبکه نسبت به وزن‌ها یک تابع غیرخطی و غیرمحدب است. در نتیجه امکان یافتن پاسخ با یک رابطه‌ی بسته وجود ندارد و ناچار باید از الگوریتم پس‌انتشار خطای گرادیان‌کاهشی استفاده شود که شامل:

- تکرارهای فراوان روی کل داده‌ها
- تنظیم حساس نرخ یادگیری
- احتمال گیر افتادن در مینیمم‌های محلی
- بار محاسباتی بالا در هر دوره‌ی آموزشی (ایپاک)

است.

در نتیجه، تفاوت بنیادین در خطی یا غیرخطی بودن مسئله‌ی بهینه‌سازی باعث می‌شود آموزش RBF با LLS چندین مرتبه سریع‌تر از آموزش مبتنی بر Backpropagation در پرسپترون‌های چندلایه باشد.



## ۸.۶ آموزش مدل با تغییر تعداد مراکز و ثابت گرفتن مقدار واریانس نورون‌ها

در این بخش شبکه را برای مقادیر مختلف  $k$  آموزش می‌دهیم. از همان اعدادی که برای  $k$  در صورت سوال پیشنهاد شده است استفاده می‌نماییم. همچنین مقدار  $\sigma$  را همان مقداری که در بخش قبل به دست آوردیم قرار می‌دهیم. کد زیر بدین منظور زده شده است:

```

1 k_list = [10, 20, 30, 40, 50, 100, 150, 200]
2 rmse_results = []
3 models_history = []
4 for k in k_list:
5     n_samples = X_train.shape[0]
6     actual_k = min(k, n_samples)
7     indices = np.random.choice(n_samples, actual_k, replace=False)
8     centers = X_train[indices]
9     sigma = sigma
10
11    phi_train = phi_matrix(X_train, centers, sigma)
12    alpha = np.linalg.pinv(phi_train).dot(y_train)
13    phi_test = phi_matrix(X_test, centers, sigma)
14    y_pred_test = phi_test.dot(alpha)
15    y_pred_train = phi_train.dot(alpha)
16    rmse_test = rmse(y_test, y_pred_test)
17    rmse_train = rmse(y_train, y_pred_train)
18    rmse_results.append(rmse_test)
19
20 print(f'{actual_k} | {sigma} | {rmse_test} | {rmse_train}')

```

۱۷: آموزش و ارزیابی مدل با تغییر تعداد مراکز و ثابت گرفتن مقدار واریانس نورون‌ها

خروجی گرفته شده از این کد به صورت زیر است:

k	Sigma	RMSE (Test)	RMSE (Train)
10	0.8328547470143473	0.0479508013176663984	0.03571671716798888
20	0.8328547470143473	0.0493843343249443116	0.03638187962721965
30	0.8328547470143473	0.021862064440383852	0.018363385278170675
40	0.8328547470143473	0.018025824640748774	0.014604579491309814
50	0.8328547470143473	0.01898482146367215	0.015497604616402351
100	0.8328547470143473	0.016369211097626165	0.013286653758441149
150	0.8328547470143473	0.016083691099683697	0.012924436965165376
200	0.8328547470143473	0.015282920418543145	0.011646556273110931

جدول ۲: مقادیر  $k$ ، سیگما و RMSE برای داده‌های آموزش و تست



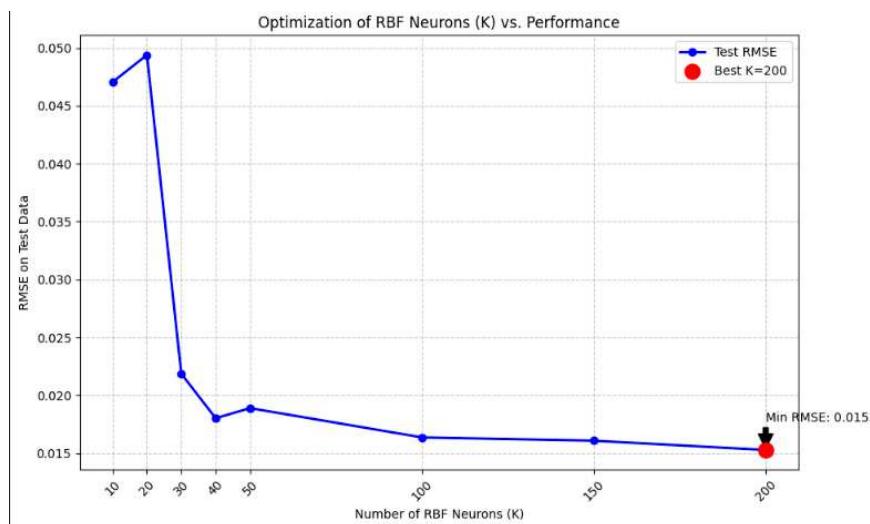
همانطور که مشاهده می‌شود با زیاد شدن تعداد مراکز که معادل است با زیاد شدن تعداد نورون‌ها، مقدار ریشه میانگین مربعات خطأ برای هم مجموعه آموزش و هم مجموعه تست کاهش پیدا می‌کند که این بدان معناست که هرچه تعداد نورون‌ها بیشتر می‌شود انگار بهتر می‌توانیم دامنه‌های متوسط در فرکانس‌های بالا را دنبال نماییم. در زیر کدی به منظور نمایش تغییرات RMSE نسبت به تغییرات  $k$  نوشته شده است:

```

1 plt.figure(figsize=(10, 6))
2 plt.plot(k_list, rmse_results, marker='o', linestyle='--', color='b', linewidth=2, label='Test
RMSE')
3
4 plt.scatter(best_k, best_rmse, color='red', s=150, zorder=5, label=f'Best K={best_k}')
5 plt.annotate(f'Min RMSE: {best_rmse:.4f}',
6 xy=(best_k, best_rmse),
7 xytext=(best_k, best_rmse + 0.05 * max(rmse_results)),
8 arrowprops=dict(facecolor='black', shrink=0.05))
9
10 plt.title('Optimization of RBF Neurons (K) vs. Performance')
11 plt.xlabel('Number of RBF Neurons (K)')
12 plt.ylabel('RMSE on Test Data')
13 plt.xticks(k_list, rotation=45)
14 plt.grid(True, linestyle='--', alpha=0.7)
15 plt.legend()
16 plt.tight_layout()
17 plt.show()
```

۱۸ Code: تغییرات RMSE نسبت به تغییرات  $k$

خروجی این کد به صورت زیر است:



شکل ۱۰: نمایش تغییرات RMSE نسبت به تغییرات  $k$

همانطور که مشاهده می‌شود بهترین مقدار برای  $k$  برابر با ۲۰۰ می‌باشد.



هنگامی که مقدار  $k$  خیلی بزرگ باشد تعداد نورون‌ها زیاد می‌شوند و مدل کاملاً روی این دیتاست تنظیم می‌شود و عملابتعادی پذیری خود را تا حد زیادی از دست می‌دهد. مثلاً اگر ورودی‌های ما نویزی شود مدل به خوبی نمی‌تواند خروجی را تشخیص دهد چرا که خیلی روی دیتاستی که ما برایش ساختیم فیت شده است و عملابتعادی برازش رخ داده است و اگر به حد زیادی کوچک باشد مدل هنوز به اندازه کافی آموزش ندیده است و مساله کم برازش رخ می‌دهد. در این دیتاستی که ما داریم به دلیل اینکه دیتا را به صورت کاملاً تزویج و بدون نویز خودمان ساختیم این مسائل دیده نمی‌شوند و هرچقدر هم تعداد نورون‌ها را بالا می‌بریم نتیجه بر روی تست نیز مطلوب است اما در عمل چنین موضوعاتی مطرح است و اگر دیتاست پیچیدگی‌های خاص خود را داشته باشد این موارد رخ می‌دهند.

## ۹.۶ آموزش مدل با تغییر واریانس و ثابت گرفتن تعداد نورون‌ها

در این بخش مقدار  $k$  را بر روی بهترین مقدار خود که همان  $200$  است تنظیم می‌نماییم. حال شبکه را با مقادیر مختلف  $\sigma$  که در صورت سوال داده شده است، آموزش می‌دهیم. سپس مقادیر RMSE را برای مجموعه آموزش و تست برای هر  $\sigma$  محاسبه می‌نماییم. درنهایت بهترین مقدار  $\sigma$  را ارائه می‌دهیم.

```

1 np.random.seed(42)
2 indices = np.random.choice(X_train.shape[0], best_k, replace=False)
3 fixed_centers = X_train[indices]
4
5 sigma_values = [0.5, 1, 2.5, 5, 7, 10]
6 rmse_results = []
7 for sigma in sigma_values:
8     phi_train = phi_matrix(X_train, fixed_centers, sigma)
9     alpha = np.linalg.pinv(phi_train).dot(y_train)
10    phi_test = phi_matrix(X_test, fixed_centers, sigma)
11    y_pred_test = phi_test.dot(alpha)
12    y_pred_train = phi_train.dot(alpha)
13    rmse_test = rmse(y_test, y_pred_test)
14    rmse_train = rmse(y_train, y_pred_train)
15    print(f"\{sigma} | {rmse_test} | {rmse_train}")
16    rmse_results.append(rmse_test)

```

۱۹: تغییرات RMSE نسبت به تغییرات  $\sigma$  Code

Sigma	RMSE (Test)	RMSE (Train)
0.5	0.013868243747038985	0.010730625539615696
1	0.015656442921451637	0.0123472083734454289
2.5	0.01778103595023957	0.014260578186100041
5	0.018471178927640806	0.014606478217888467
7	0.018646016837585166	0.014720791311617232
10	0.019367540023925413	0.015561147944709808

جدول ۳: مقادیر سیگما و RMSE برای داده‌های آموزش و تست

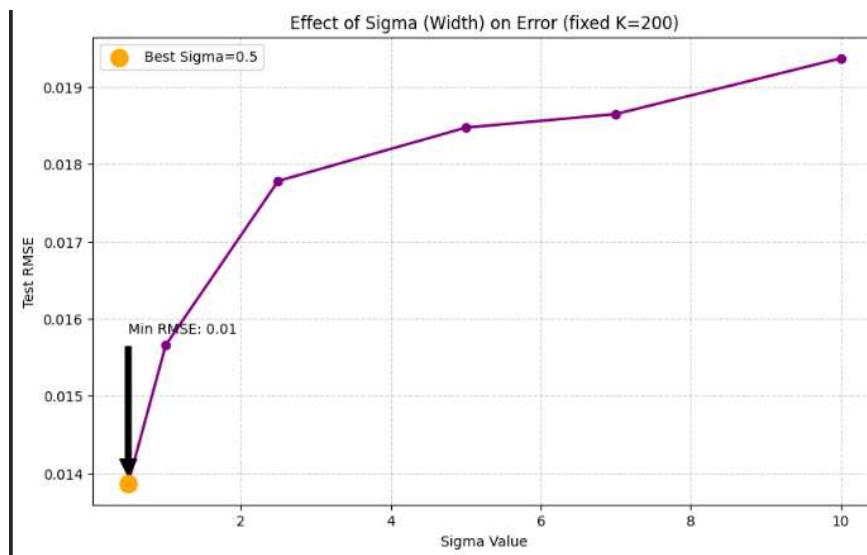


همانطور که مشاهده می‌نماییم بهترین مقدار  $\sigma$  که کمترین مقدار RMSE را برای مجموعه تست ایجاد می‌کند برابر با ۰.۵ است و با افزایش مقدار واریانس دقت ما بر روی مجموعه تست کاهش می‌یابد. کد زده شده برای نمایش تغییرات RMSE نسبت به تغییرات  $\sigma$  به صورت زیر خواهد بود.

```
 1 plt.figure(figsize=(10, 6))
 2 plt.plot(sigma_values, rmse_results, marker='o', linestyle='-', color='purple', linewidth=2)
 3
 4 plt.scatter(best_sigma, min_rmse, color='orange', s=150, zorder=5, label=f'Best Sigma={best_sigma}')
 5 plt.annotate(f'Min RMSE: {min_rmse:.2f}',
 6             xy=(best_sigma, min_rmse),
 7             xytext=(best_sigma, min_rmse + 0.1 * max(rmse_results)),
 8             arrowprops=dict(facecolor='black', shrink=0.05))
 9
10 plt.title(f'Effect of Sigma (Width) on Error (fixed K={best_k})')
11 plt.xlabel('Sigma Value')
12 plt.ylabel('Test RMSE')
13 plt.grid(True, linestyle='--', alpha=0.6)
14 plt.legend()
15 plt.show()
```

۲۰: رسم تغییرات RMSE نسبت به تغییرات  $\sigma$  Code

خروجی این کد نمودار شکل زیر می‌باشد:



شکل ۱۱: نمایش تغییرات RMSE نسبت به تغییرات  $\sigma$

همانطور که مشاهده می‌نماییم بهترین مقدار  $\sigma$  برابر با ۰.۵ می‌باشد. حال از مدل بر روی بهترین مقدار  $\sigma$  پیش‌بینی می‌گیریم و آن را به صورت نموداری نمایش می‌دهیم. این پیش‌بینی بر روی مجموعه تست گرفته شده است. کد زیر در این بخش زده شده است:



```

1   k = best_k # best k
2   sigma = best_sigma # best
3   k, sigma
4   np.random.seed(42)
5   random_indices = np.random.choice(len(X_train), k , replace=False)
6   centers_final = X_train[random_indices]
7   centers_final.shape
8   phi_train_final = phi_matrix(X_train, centers_final, sigma)
9   phi_train_final.shape
10  alpha = np.linalg.pinv(phi_train_final).dot(y_train)
11  alpha.shape
12  y_pred_train_final = phi_train_final.dot(alpha)
13  phi_test_final = phi_matrix(X_test, centers_final, sigma)
14  y_pred_test_final = phi_test_final.dot(alpha)
15  rmse_train = rmse(y_train, y_pred_train_final)
16  rmse_test = rmse(y_test, y_pred_test_final)
17
18  print(f"Result for Train: {rmse_train}")
19  print(f"Result for Test: {rmse_test}")
20  plt.figure(figsize=(12, 6))
21  plt.plot(y_test, 'k-', alpha=0.6, label='Actual Data (Peaks)')
22  plt.plot(y_pred_test_final, 'r--', linewidth=2, label='Random Centers + Fixed Sigma (The Red
Result)')
23  plt.title(f'Result with Random Centers & Fixed Sigma={sigma}')
24  plt.legend()
25  plt.show()

```

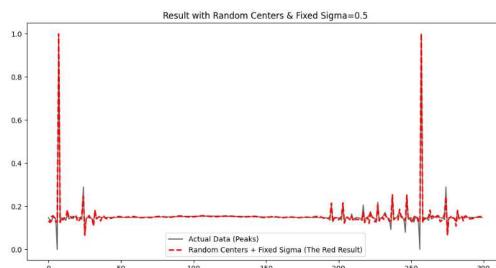
۲۱: کد کامل inference بر روی مجموعه تست و ارزیابی آن

مقادیر RMSE به شرح زیر هستند:

$$\text{Result for Train: } 0.010730625539615696 \quad (79)$$

$$\text{Result for Test: } 0.013868243747038985 \quad (80)$$

همانطور که مشاهده می‌شود، در این حالت مقدار خطای کاهش یافته است به طوریکه به مقدار یک هزارم بر روی ترین و به مقدار دو هزارم بر روی تست بهبود داشته‌ایم. نموداری که در این بخش برای تغییرات RMSE نسبت به  $\sigma$  است، به شرح زیر می‌باشد:



شکل ۱۲: نمایش خروجی پیش‌بینی شده به همراه خروجی واقعی برای بهترین مقدار  $k$  و  $\sigma$



پارامتر  $\sigma$  نقش تعیین‌کننده‌ای در میزان گستردگی یا تمرکز تابع شعاعی دارد و به طور مستقیم بر توانایی تعمیم‌پذیری شبکه‌ی RBF اثر می‌گذارد. رفتار شبکه در دو حالت حدی  $\sigma$  بسیار کوچک و بسیار بزرگ به صورت زیر است:

۱. زمانی که  $\sigma$  بسیار کوچک باشد در این حالت تابع شعاعی

$$\phi_k(x) = \exp\left(-\frac{\|x - \mu_k\|^2}{\sigma_k^2}\right)$$

بسیار تیز و موضعی می‌شود؛ به‌طوری که تنها ورودی‌هایی را فعال می‌کند که فاصله‌ی بسیار کمی با مرکز  $\mu_k$  دارند. نتیجه این است که:

- هر نورون تنها یک ناحیه‌ی بسیار کوچک از فضنا را پوشش می‌دهد؛

- شبکه برای نمونه‌های آموزشی مقدارهای دقیقی یاد می‌گیرد (پدیده‌ی حافظه‌ی صرف)؛

- کوچک‌ترین تغییر در ورودی باعث می‌شود نورون فعال نشود و شبکه نتواند خروجی درست تولید کند.

به همین دلیل در این حالت شبکه دچار بیش‌برازش می‌شود و توانایی تعمیم‌دهی آن به شدت کاهش می‌یابد، خصوصاً در نواحی که داده نوسانات متوسط یا کم دارند و سیستم رفتار حساس‌تری نشان می‌دهد.

۲. زمانی که  $\sigma$  بسیار بزرگ باشد: در این حالت تابع شعاعی بسیار پهن می‌شود و تقریباً برای تمامی ورودی‌ها مقدار نزدیک به ۱ می‌گیرد:

$$\phi_k(x) \approx 1.$$

در نتیجه:

- نورون‌ها رفتار تقریباً یکسانی پیدا می‌کنند،

- تفکیک‌پذیری مکانی از بین می‌رود،

- شبکه فقط یک تقریب «کلی و صاف» از داده تولید می‌کند.

در این وضعیت شبکه دچار کم‌برازش می‌شود و نمی‌تواند تغییرات موضعی تابع هدف را بازسازی کند؛ به‌خصوص در نواحی حساس (مثلًا نزدیک تکینگی در  $y(t-2) - y(t-1)$  دقت افت می‌کند).

## ۱۰.۶ چالش‌ها

در ابتدا این تصور وجود داشت که افزایش تعداد نورون‌ها و کاهش مقدار  $\sigma$  باعث افزایش حساسیت نورون‌ها شده و در نتیجه مدل دچار بیش‌برازش (Overfitting) می‌شود. بر همین اساس انتظار داشتیم در نتایج نیز چنین رفتاری مشاهده شود. اما با تحقیق و بررسی بیشتر متوجه شدیم که از آنجایی که داده‌ها مستقیماً از روی مدل سیستم تولید شده‌اند و این داده‌ها بسیار تمیز و بدون نویز هستند، شرایط بروز بیش‌برازش در این مسئله متفاوت است. به‌گونه‌ای که هرچه نورون‌ها حساس‌تر می‌شوند، مدل توانایی بهتری در تقریب تابع هدف پیدا کرده و خطأ بر روی مجموعهٔ تست نیز کاهش می‌یابد.

همچنین مشاهده کردیم که با افزایش تعداد نورون‌ها و کاهش واریانس، نورون‌هایی شکل می‌گیرند که قادرند نمایانگر برخی فرکانس‌های خاص باشند؛ بنابراین مدل در آن فرکانس‌ها پیش‌بینی دقیق‌تری ارائه می‌دهد و در مجموع عملکرد بهتری در بازسازی سیگنال خواهد داشت. با این حال، مسئله تعمیم‌پذیری همچنان مطرح است. به عنوان مثال، در صورت ورود داده‌های نویزی به مدل، نتایج کیفیت مطلوبی نخواهند داشت و مدل به سرعت وارد ناحیهٔ بیش‌برازش می‌شود.

شایان ذکر است که در برخی مراحل نیز الگوریتم با مشکلاتی مواجه شد که با همکاری و بررسی مشترک، این مسائل برطرف گردیدند.



## ۱۱.۶ بخش چهارم: تلاش برای حل چالش‌ها

همانطور که سوال از ما خواسته است تابعی برای آموزش مدل RBFNN static با استفاده از L<sup>2</sup> Regularization می‌نویسیم. کد مربوط به این تابع به صورت زیر است:

```

1 def train_rbf_l2(phi_train, y_train, lam):
2     k = phi_train.shape[1]
3     I = np.eye(k)
4     A = phi_train.T @ phi_train + lam * I #darim: Φ(Φ + I)
5     B = phi_train.T @ y_train #darim Φ y
6     alpha = np.linalg.solve(A, B) # inv(A) @ B
7
8     return alpha

```

۲۲: تابع آموزش RBFNN static با استفاده از L<sup>2</sup> Code

از آنجاییکه سوال از ما خواسته است که مقدار  $k$  بزرگ باشد، مقدار آن را روی ۷۰۰ تنظیم نمودیم. عملاً داریم کل داده‌های آموزش را به عنوان یک مرکز در نظر می‌گیریم. همچنین مقدار  $\sigma$  را روی بهترین مقداری که از بخش‌های قبلی به دست آوردهیم، سمت می‌نماییم. این مقدار برابر با ۰.۵ بود.

در ادامه با توجه به الگوریتمی که داشتیم پیش می‌رویم و بردار وزن‌های  $\alpha_k$  و همچنین خروجی‌ها را روی مجموعه آموزش و تست به دست می‌آوریم.

```

1 np.random.seed(42)
2 random_indices = np.random.choice(len(X_train), k, replace=False)
3 centers = X_train[random_indices]
4 centers.shape
5 phi_train_c = phi_matrix(X_train, centers, sigma)
6 alpha = np.linalg.pinv(phi_train_c).dot(y_train)
7 y_pred_train_c = phi_train_c.dot(alpha)
8 phi_test_c = phi_matrix(X_test, centers, sigma)
9 y_pred_test_c = phi_test_c.dot(alpha)

```

۲۳: آموزش مدل و محاسبه خروجی‌ها بر روی مجموعه آموزش و تست

حال مدل آموزش دیده را بر روی مجموعه تست ارزیابی می‌نماییم. در اینجا هنوز مقدار ضریب رگولاریزیشن را اعمال ننمودیم و داریم بدون ضریب رگولاریزیشن مساله را حل می‌نماییم. مقدار RMSE و خروجی را بر روی مجموعه تست محاسبه می‌نماییم:

```

1 rmse_train = rmse(y_train, y_pred_train_c)
2 rmse_test = rmse(y_test, y_pred_test_c)
3
4 print(f"Result for Train: {rmse_train}")
5 print(f"Result for Test: {rmse_test}")

```

۲۴: محاسبه مقدار RMSE بر روی آموزش و تست



نتیجه این کد به صورت زیر است:

Result for Train = 0.006960215405499155

Result for Test = 0.008988554526157296

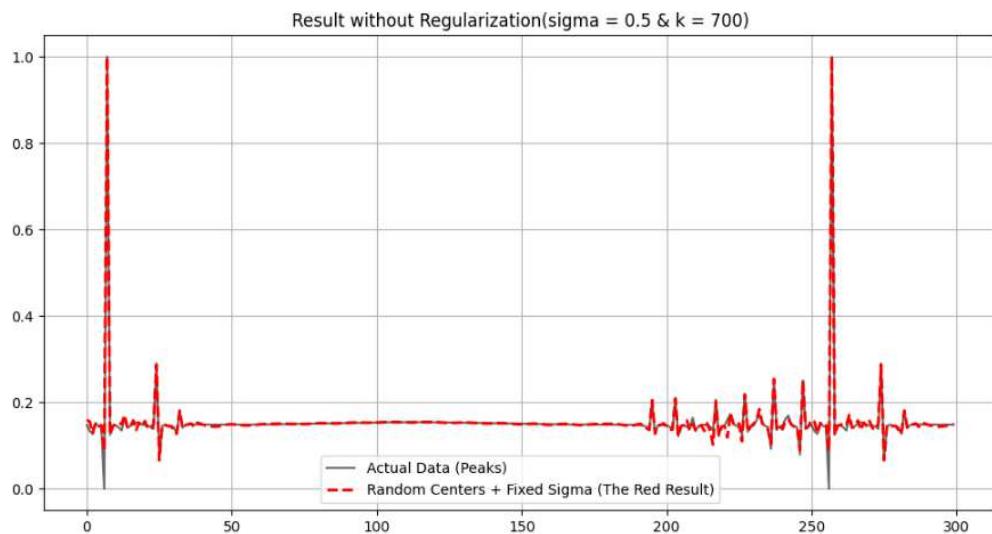
حال ترسیم خروجی بر روی تست:

```

1 plt.figure(figsize=(12, 6))
2 plt.plot(y_test, 'k-', alpha=0.6, label='Actual Data (Peaks)')
3 plt.plot(y_pred_test_c, 'r--', linewidth=2, label='Random Centers + Fixed Sigma (The Red
Result)')
4 plt.title(f'Result without Regularization(sigma = {sigma} & k = {k})')
5 plt.legend()
6 plt.grid(True)
7 plt.show()

```

۲۵ Code : ترسیم نمودار خروجی واقعی و پیش‌بینی شده بر روی مجموعه تست



شکل ۱۳: نمودار خروجی واقعی و خروجی پیش‌بینی شده بدون رگولاریزیشن برای مجموعه تست

همانطور که مشاهده می‌نماییم مقدار خطای حد بسیار زیادی نسبت به بخش‌های قبل کاهش یافته است و همانطور که اشاره کردیم بالا رفتن تعداد نورون‌ها باعث می‌شود که پیش‌بینی دقیق‌تری داشته باشیم هرچند که تعمیم‌پذیری کاهش می‌یابد. در این دیتابست این موضوع را مشاهده نمی‌کنیم چون با دیتابی تمیز و بدون نویز مواجه هستیم.

حال رگولاریزیشن با مقدار 0.1 را اعمال می‌نماییم. و تمامی مراحلی که انجام دادیم را این بار با این ضریب نیز تکرار می‌کنیم. تمام کل زده شده در این بخش از به دست آوردن مراکز تا محاسبه مقدار خطای نمایش خروجی‌ها به صورت زیر است:

اگر در نمودار آخری که به دست آورده‌یم نیز توجه کنیم، در نواحی که نوسانات شدیدی داریم دقت را از دست دادیم که این دقیقاً به همان مفهوم کاهش یافتن حساسیت نورون‌ها اشاره دارد.



## ۱۲.۶ نقش پارامتر $\lambda$ در Regularization

در روش L2 Regularization که در این پروژه استفاده شده، پارامتر  $\lambda$  به عنوان یک پارامتر تنظیمی برای کنترل مقدار جریمه به کار می‌رود. رابطه‌ای که در این روش برای بهینه‌سازی وزن‌ها استفاده می‌شود به صورت زیر است:

$$(\Phi^T \Phi + \lambda I) \alpha = \Phi^T Y$$

نقش  $\lambda$  این است که با افزایش مقدار آن، میزان جریمه‌ای که به وزن‌ها تعلق می‌گیرد بیشتر می‌شود. در واقع،  $\lambda$  با افزایش، وزن‌های بزرگ‌تر را جریمه می‌کند و مدل را مجبور به ساده‌سازی می‌کند. این ساده‌سازی باعث می‌شود که مدل کمتر قادر به یادگیری جزئیات دقیق داده‌ها شود، که می‌تواند به کاهش Overfitting و بهتر شدن تعمیم‌پذیری در داده‌های جدید منجر شود.

با این حال، در این پروژه و با توجه به ویژگی‌های خاص دیتاست، مشاهده شده که با افزایش مقدار  $\lambda$ ، نتایج مدل کاهش می‌یابد. این اتفاق به احتمال زیاد به دلیل این است که دیتاست ما از کیفیت بالایی برخوردار است و داده‌ها به طور دقیق و مرتب به مدل وارد می‌شوند. در چنین شرایطی، اعمال جریمه برای وزن‌ها می‌تواند مانع از یادگیری دقیق و حساس مدل شود، زیرا مدل می‌تواند ویژگی‌های مهم و پیچیده داده‌ها را که برای پیش‌بینی ضروری هستند، نادیده بگیرد.

در نتیجه، وقتی مقدار  $\lambda$  خیلی بزرگ می‌شود، مدل به طور بیش از حد ساده شده و نمی‌تواند به خوبی از جزئیات داده‌ها بهره‌برداری کند. اما وقتی مقدار  $\lambda$  کاهش یابد یا اصلاً از آن استفاده نشود، مدل قادر خواهد بود ویژگی‌های دقیق‌تری را از داده‌های تمیز و مرتب یاد بگیرد و نتایج بهتری داشته باشد. این نکته تأکید بر این دارد که در دیتاست‌های تمیز و خوب، معمولاً استفاده از Regularization کمتری نیاز است.

به عبارت دیگر، با کاهش  $\lambda$  و نبودن آن، مدل به نورون‌ها و ویژگی‌های حساس‌تر توجه بیشتری دارد و این باعث بهبود عملکرد مدل در این پروژه شده است. در کد زیر نیز نشان دادیم که با افزایش مقدار ضریب رگولاrizیشن، مقدار RMSE افزایش می‌یابد.

```

lambda_values = [0.001, 0.01, 0.1, 1, 10]

rmse_train_list = []
rmse_test_list = []

phi_train = phi_matrix(X_train, centers, sigma)
phi_test = phi_matrix(X_test, centers, sigma)

for lam in lambda_values:
    alpha = train_rbf_12(phi_train, y_train, lam)
    y_train_pred = phi_train @ alpha
    y_test_pred = phi_test @ alpha

    rmse_train = rmse(y_train, y_train_pred)
    rmse_test = rmse(y_test, y_test_pred)

```



```

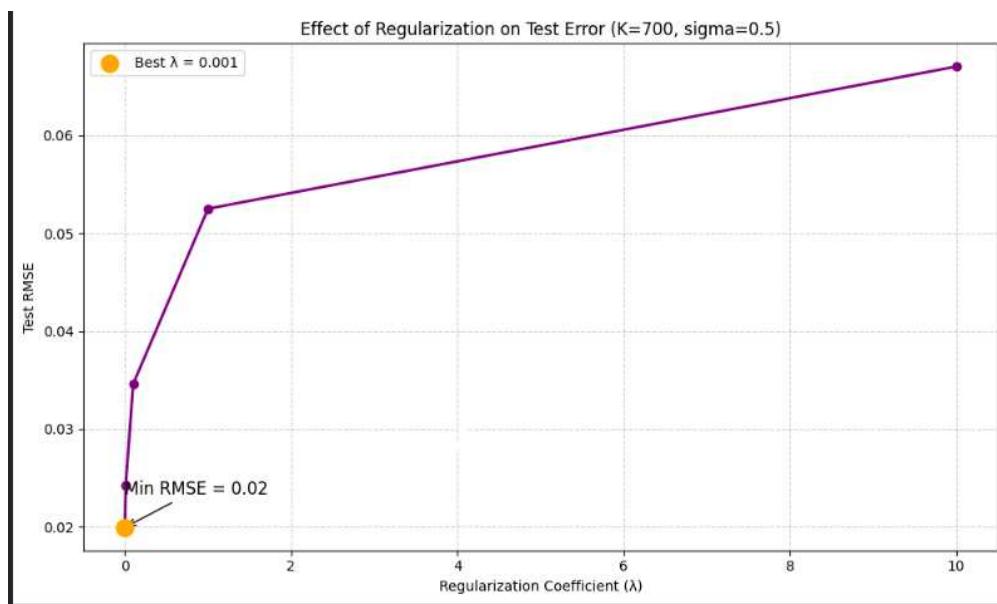
    ۲۲
    ۲۳ rmse_train_list.append(rmse_train)
    ۲۴ rmse_test_list.append(rmse_test)
    ۲۵
    ۲۶ print(f" = {lam} | RMSE_train = {rmse_train:.4f} | RMSE_test = {rmse_test:.4f}")
    ۲۷ min_rmse = min(rmse_test_list)
    ۲۸ best_L_idx = rmse_test_list.index(min_rmse)
    ۲۹ best_L = lambda_values[best_L_idx]
    ۳۰ plt.figure(figsize=(10, 6))
    ۳۱ plt.plot(lambda_values, rmse_test_list, marker='o', linestyle='-', color='purple', linewidth
    =2)
    ۳۲ plt.scatter(best_L, min_rmse, color='orange', s=150, zorder=5, label=f'Best = {best_L}')
    ۳۳ plt.annotate(
    ۳۴     f"Min RMSE = {min_rmse:.2f}",
    ۳۵     xy=(best_L, min_rmse),
    ۳۶     xytext=(best_L * 1.2, min_rmse + 0.05 * max(rmse_test_list)),
    ۳۷     arrowprops=dict(arrowstyle="->", color="black"),
    ۳۸     fontsize=12
    ۳۹ )
    ۴۰
    ۴۱ plt.title(f'Effect of Regularization on Test Error (K={k}, sigma={sigma})')
    ۴۲ plt.xlabel('Regularization Coefficient ()')
    ۴۳ plt.ylabel('Test RMSE')
    ۴۴ plt.grid(True, linestyle='--', alpha=0.6)
    ۴۵ plt.legend()
    ۴۶ plt.tight_layout()
    ۴۷ plt.show()

```

۲۶ Code: تغییرات RMSE نسبت به تغییرات  $\lambda$ 

$\lambda$	RMSE (Test)	RMSE (Train)
0.001	0.0200	0.0163
0.01	0.0243	0.0194
0.1	0.0346	0.0260
1	0.0525	0.0368
10	0.0671	0.0457

جدول ۴: مقادیر  $\lambda$  و RMSE برای داده‌های آموزش و تست



شکل ۱۴: تغییرات RMSE نسبت به تغییرات  $\lambda$

### ۱۳.۶ بخش پنجم:

تا به اینجا به این شکل پیش رفته که پس از اینکه تعداد مرکز را انتخاب نمودیم به روش تصادفی این مرکز را انتخاب می‌کردیم. در این بخش سوال از ما خواسته که با استفاده از روش clusteringkmeans مرکز را انتخاب نماییم. ما از دو روش برای پیاده‌سازی این موضوع استفاده کردیم. اولین روش پیاده‌سازیتابع kmeans است و روش دوم استفاده از کتابخانه sklearn است. ما تلاش کردیم که تا حد زیادی تابعمان را مشابه کتابخانه sklearn پیاده‌سازی نماییم مثلاً این کتابخانه هنگامی که میخواهد مرکز اولیه را مشخص کند از روشی موسوم به kmeans++ استفاده می‌کند که این کار را مانیز انجام دادیم اما این کتابخانه در تعداد دوره بیشتری فاصله بین مرکز را چک می‌کند و برای رسیدن به جواب بهینه جستجوی بیشتری را انجام می‌دهد. این کار در بار کمتری توسط تابع ما انجام می‌شود. این موضوع باعث می‌شود تا نتایجی که کتابخانه sklearn می‌دهد، بهتر باشد. همچنین تابعی که توسط این کتابخانه توسعه یافته شده در کل بهینه‌تر است. توابعی که ما خودمان پیاده‌سازی نمودیم به صورت زیر هستند:

```

1 import numpy as np
2
3 def kmeans_plus_plus_init(X, k, random_state=42):
4     np.random.seed(random_state)
5     n_samples = X.shape[0]
6
7     first_idx = np.random.choice(n_samples)
8     centroids = [X[first_idx]]
9
10    # remaining k-1 centroids
11    for _ in range(1, k):
12        # distances to the nearest chosen centroid
13        distances = np.array([

```



```
14         min(np.linalg.norm(x - c)**2 for c in centroids)
15     for x in X
16   ]
17
18
19     probs = distances / distances.sum()
20
21     # next centroid
22     next_idx = np.random.choice(n_samples, p=probs)
23     centroids.append(X[next_idx])
24
25
26     return np.array(centroids)
27 def KMeans (X, k , max_iter = 100, tol = 1e-4):
28
29   ...
30
31   tol smaller = alg zood stop mishe, speed bala, deghat kam
32   tol bozorgtar = alg sakht migire, time bishtari migire, deghat ziyad, ta vaghti edami
33   mide ke center ha kamel fix beshan
34
35   ...
36
37   n_samples, n_features = X.shape
38   np.random.seed(42)
39
40   '''random_indices = np.random.choice(n_samples, k , replace = False)
41   centroids = X[random_indices]'''
42   centroids = kmeans_plus_plus_init(X, k)
43
44
45   for i in range(max_iter):
46     old_centroids = centroids.copy()
47     distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis = 2)
48     # X shape: (N, 1, D) | Centroids shape: (1, K, D) -> Dist shape: (N, K)
49     # distance hame noghat to center ha mohasebe shodan
50
51
52     cluster_labels = np.argmin(distances, axis = 1)
53     # index nazdiktarin center baraye har dade ro peyda konim
54     new_centroids = np.zeros((k, n_features))
55     for cluster_index in range(k):
56       cluster_points = X[cluster_labels == cluster_index ]
57       if len(cluster_points) > 0:
58         new_centroids[cluster_index] = cluster_points.mean(axis = 0)
59       else:
60         new_centroids[cluster_index] = old_centroids[cluster_index]
61
62
63     centroids = new_centroids
64
65
66     shift = np.linalg.norm(centroids - old_centroids)
67     if shift < tol:
68       print(f"K-Means converged at iteration {i}")
```



```
58         break
59     return centroids
```

## ۲۷ Code: تابع پیاده‌سازی شده برای kmeans

به ازای مقادیر مختلفی از  $k$  و با استفاده از تابع `kmeans` مدل را آموزش می‌دهیم. به ازای مقادیر مختلف  $k$  مقدار RMSE را برای مجموعه آموزش و تست محاسبه می‌نماییم. کد زیر بدین منظور زده شده است:

```
1  k_list = [10, 20, 30, 40, 50, 100, 150, 200]
2  rmse_results = []
3  models_history = []
4
5  for k in k_list:
6      n_samples = X_train.shape[0]
7      actual_k = min(k, n_samples)
8      centers = KMeans(X_train, k)
9      sigma = sigma_(centers)
10
11     phi_train = phi_matrix(X_train, centers, sigma)
12     alpha = np.linalg.pinv(phi_train).dot(y_train)
13     phi_test = phi_matrix(X_test, centers, sigma)
14     y_pred_test = phi_test.dot(alpha)
15     y_pred_train = phi_train.dot(alpha)
16     rmse_test = rmse(y_test, y_pred_test)
17     rmse_train = rmse(y_train, y_pred_train)
18     rmse_results.append(rmse_test)
19
20     print(f'{actual_k} | {sigma} | {rmse_test} | {rmse_train}')
21
22 best_rmse = min(rmse_results)
23 best_k_index = rmse_results.index(best_rmse)
24 best_k = k_list[best_k_index]
25 plt.figure(figsize=(10, 6))
26 plt.plot(k_list, rmse_results, marker='o', linestyle='--', color='b', linewidth=2, label='Test
RMSE')
27
28 plt.scatter(best_k, best_rmse, color='red', s=150, zorder=5, label=f'Best K={best_k}')
29 plt.annotate(f'Min RMSE: {best_rmse:.4f}',
30             xy=(best_k, best_rmse),
31             xytext=(best_k, best_rmse + 0.05 * max(rmse_results)),
32             arrowprops=dict(facecolor='black', shrink=0.05))
33
34 plt.title('Optimization of RBF Neurons (K) vs. Performance')
35 plt.xlabel('Number of RBF Neurons (K)')
36 plt.ylabel('RMSE on Test Data')
37 plt.xticks(k_list, rotation=45)
38 plt.grid(True, linestyle='--', alpha=0.7)
```



```

39 plt.legend()
40 plt.tight_layout()
41 plt.show()

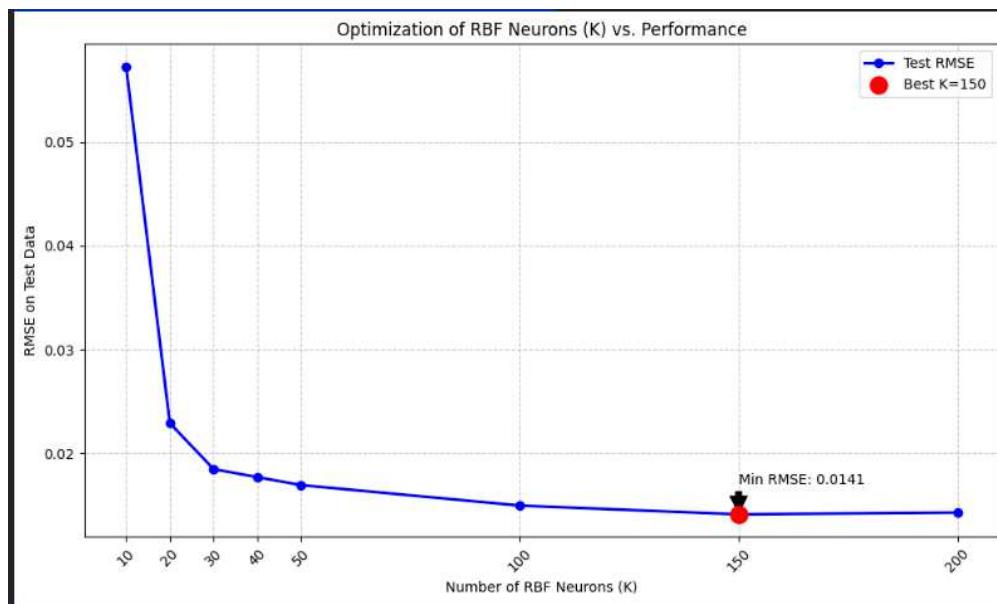
```

۲۸ Code: استفاده از روش kmeans و آموزش مدل و ارزیابی آن با استفاده از تغییرات k

در این کد وزن‌های بهینه نیز محاسبه شده‌اند. در ادامه مقادیر RMSE را به ازای مقادیر مختلف K گزارش می‌نماییم:

K	Sigma	RMSE (Test)	RMSE (Train)
10	1.2923699512374591	0.05726544637977595	0.040215290239696326
20	1.0426049634478782	0.02290459889777694	0.018968454426305462
30	0.9638175748306979	0.018479762797385627	0.015109533378511572
40	0.9083346266495169	0.01770602738950625	0.014296087678879445
50	0.8833895590841012	0.016941774290500377	0.013706808088290078
100	0.8064545161097252	0.014968979883198622	0.011432736312711168
150	0.7965168566729466	0.014124253224053895	0.010861987984709668
200	0.8379196034125181	0.014295950806403961	0.010837743365333604

جدول ۵: مقادیر K، سیگما و RMSE برای داده‌های آموزش و تست



شکل ۱۵: تغییرات RMSE نسبت به تغییرات k در kmeans

در اینجا بهترین مقدار k را ذخیره نمودیم و سپس در بخش بعد از این مقدار ثابت استفاده نمودیم و سیگما را تغییر دادیم تا بهترین مقدار برای سیگما را نیز بیابیم:

```
1 k = best_k
```

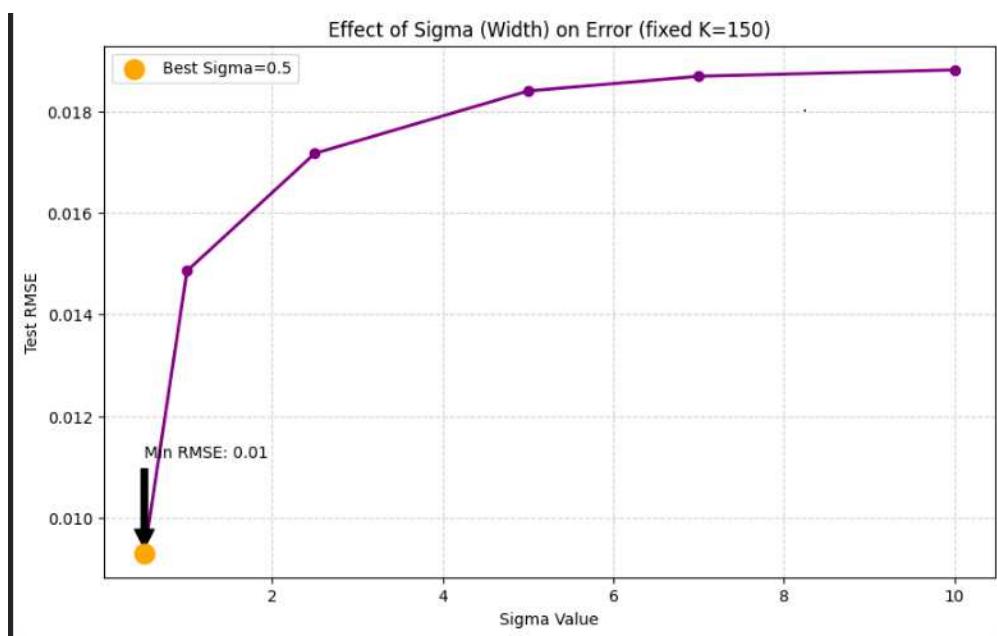


```
 1 kmeans_centers = KMeans(X_train, k)
 2 np.random.seed(42)
 3 fixed_centers = kmeans_centers
 4
 5 sigma_values = [0.5, 1, 2.5, 5, 7, 10]
 6 rmse_results = []
 7 for sigma in sigma_values:
 8     phi_train = phi_matrix(X_train, fixed_centers, sigma)
 9     alpha = np.linalg.pinv(phi_train).dot(y_train)
10     phi_test = phi_matrix(X_test, fixed_centers, sigma)
11     y_pred_test = phi_test.dot(alpha)
12     y_pred_train = phi_train.dot(alpha)
13     rmse_test = rmse(y_test, y_pred_test)
14     rmse_train = rmse(y_train, y_pred_train)
15     print(f"{sigma} | {rmse_test} | {rmse_train}")
16     rmse_results.append(rmse_test)
17 min_rmse = min(rmse_results)
18 best_sigma_idx = rmse_results.index(min_rmse)
19 best_sigma = sigma_values[best_sigma_idx]
20 plt.figure(figsize=(10, 6))
21 plt.plot(sigma_values, rmse_results, marker='o', linestyle='-', color='purple', linewidth=2)
22
23 plt.scatter(best_sigma, min_rmse, color='orange', s=150, zorder=5, label=f'Best Sigma={best_sigma}')
24 plt.annotate(f'Min RMSE: {min_rmse:.2f}',
25             xy=(best_sigma, min_rmse),
26             xytext=(best_sigma, min_rmse + 0.1 * max(rmse_results)),
27             arrowprops=dict(facecolor='black', shrink=0.05))
28
29
30 plt.title(f'Effect of Sigma (Width) on Error (fixed K={best_k})')
31 plt.xlabel('Sigma Value')
32 plt.ylabel('Test RMSE')
33 plt.grid(True, linestyle='--', alpha=0.6)
34 plt.legend()
35 plt.show()
```

۲۹: استفاده از روش kmeans و آموزش مدل و ارزیابی آن با استفاده از تغییر واریانس

Sigma	RMSE (Test)	RMSE (Train)
0.5	0.00930164406991945	0.0070320890925147945
1	0.014856774332816796	0.011336133360322524
2.5	0.01717075086915333	0.01362060118395568
5	0.018397865322470734	0.014535418075107885
7	0.018687089998828888	0.014736587762437043
10	0.018813118260311706	0.01480464083502848

جدول ۶: ارزیابی مدل آموزش دیده به وسیله KMeans به ازای تغییرات K



شکل ۱۶: تغییرات RMSE نسبت به تغییرات  $\sigma$  در kmeans

حال از پارامترهای بهینه‌ای که به دست آوردیم استفاده می‌نماییم تا بهترین مدل خود را با این روش آموزش دهیم. این کار به صورت زیر انجام می‌شود:

```

1 kmeans_centers.shape
2 sigma = best_sigma
3 sigma
4 phi_train_k = phi_matrix(X_train, kmeans_centers, sigma)
5 phi_train_k.shape
6 alpha = np.linalg.pinv(phi_train_k).dot(y_train)
7 alpha.shape
8 y_pred_train_k = phi_train_k.dot(alpha)
9 phi_test_k = phi_matrix(X_test, kmeans_centers, sigma)
10 y_pred_test_k = phi_test_k.dot(alpha)
11 rmse_train = rmse(y_train, y_pred_train_k)

```



```

12 rmse_test = rmse(y_test, y_pred_test_k)
13
14 print(f"Result for Train: {rmse_train}")
15 print(f"Result for Test: {rmse_test}")
16 plt.figure(figsize=(12, 6))
17 plt.plot(y_test, 'k-', alpha=0.6, label='Actual Data (Peaks)')
18 plt.plot(y_pred_test_k, 'r--', linewidth=2, label='Clustering K_means Centers + Fixed Sigma
(The Red Result)')
19 plt.title(f'Result with Clustering K_means on Test(sigma = {sigma} & k = {k})')
20 plt.legend()
21 plt.grid(True)
22 plt.show()

```

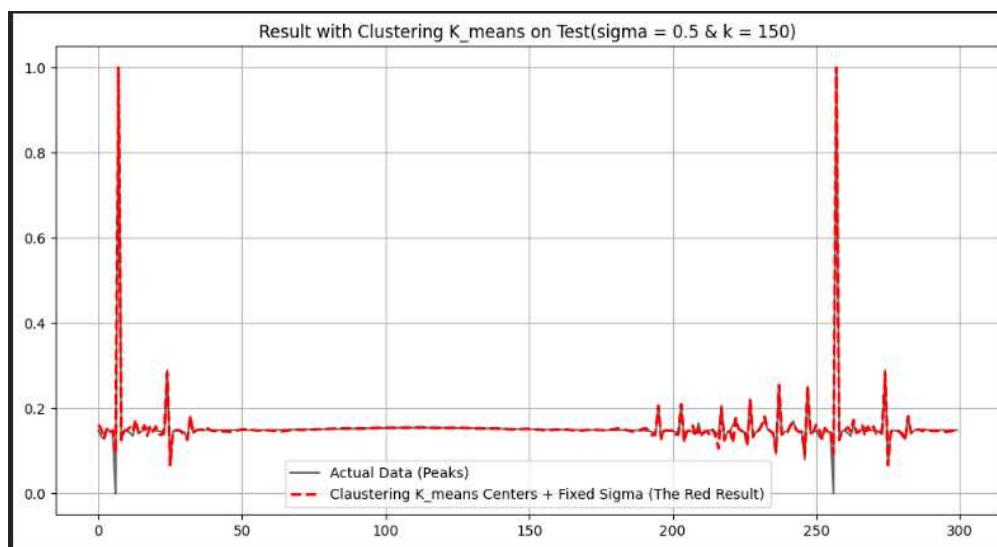
### ۳۰: آموزش مدل و ارزیابی آن با پارامترهای بهینه در روش kmeans

نتایج RMSE را برای مجموعه آموزش و تست به صورت زیر به دست می‌آید:

Result for Train = 0.0070320890925147945

Result for Test = 0.00930164406991945

خروجی پیش‌بینی شده به همراه خروجی واقعی برای مجموعه تست به صورت زیر به دست می‌آید:



شکل ۱۷: خروجی واقعی و پیش‌بینی مدل با استفاده از روش kmeans در بهینه‌ترین پارامترها

به وضوح دیده می‌شود که هنگامی که از این روش استفاده می‌نماییم بهبود بسیار زیادی در نتایجمان حاصل می‌شود چرا که روش kmeans بهینه‌ترین مرکز را برای به دست آوردن بهترین مدل فراهم می‌کند و این روش بهتر از به دست آوردن مرکز به شیوه تصادفی است. در ادامه از استراتژی NearestNeighbor استفاده می‌نماییم. در این استراتژی مقادیر  $\sigma$  به صورت منحصر به فرد برای هر نورون محاسبه می‌شود. بنابراین به جای اینکه تمامی نورون‌ها مقادیر واریانس ثابتی داشته باشند، مقادیر مشخصی را به خود اختصاص می‌دهند. همانطور که صورت سوال گفته است این تابع را محاسبه می‌نماییم:



```

1 def adaptive_sigma(centers, p = 2):
2     ...
3     data moterakem bahse va center ha be ham nazdic bashan sigma kam mishe , deghat mire bala
4     data parakande bashe va center ha fasele dashte bashan sigma bozorg mishe to
5     generalization save beshe
6     ...
7     ...
8     k_ = centers.shape[0]
9     sigmas = np.zeros(k_)
10
11    for i in range (k_):
12        diff = centers - centers[i]
13        distances = np.sqrt(np.sum(diff**2, axis=1))
14
15        sorted_indices = np.argsort(distances)
16        sorted_distances = distances[sorted_indices]
17
18        nearest_neighbor = sorted_distances[1 : p + 1] # index 0 mishe khodesh
19        if len(nearest_neighbor) > 0 :
20            sigmas[i] = np.mean(nearest_neighbor)
21        else:
22            sigmas[i] = 1 # in case we have just 1 center
23
24
25    return sigmas

```

## ۳۱: تابع سیگمای تطبیقی

ماتریس  $\phi$  نیز می‌بایست به گونه دیگری محاسبه شود چرا که اکنون هر نورون واریانس مخصوص به خود را دارد و ما باید هر ورودی را با مرکز و واریانس مشخص به خود فاصله اقلیدسیش را محاسبه نماییم و سپس وارد آن رابطه نمایی بکنیم.

```

1 def phi_matrix_variable_sigma(X, centers, sigmas):
2     n = X.shape[0]
3     k_ = centers.shape[0]
4     phi = np.zeros((n, k_))
5
6     for i in range(n):
7         for j in range(k_):
8             diff = X[i] - centers[j]
9             phi[i, j] = np.exp(-(np.sum(diff**2)) / (sigmas[j]**2))
10
11    return phi

```

۳۲: تابع محاسبه ماتریس  $\phi$  برای سیگمای تطبیقی

حال با استفاده از این دو استراتژی مدل را آموزش می‌دهیم تا بینیم نتایج چگونه تغییر می‌کند:

```

1 sigmas = adaptive_sigma(kmeans_centers, p=2)
2

```

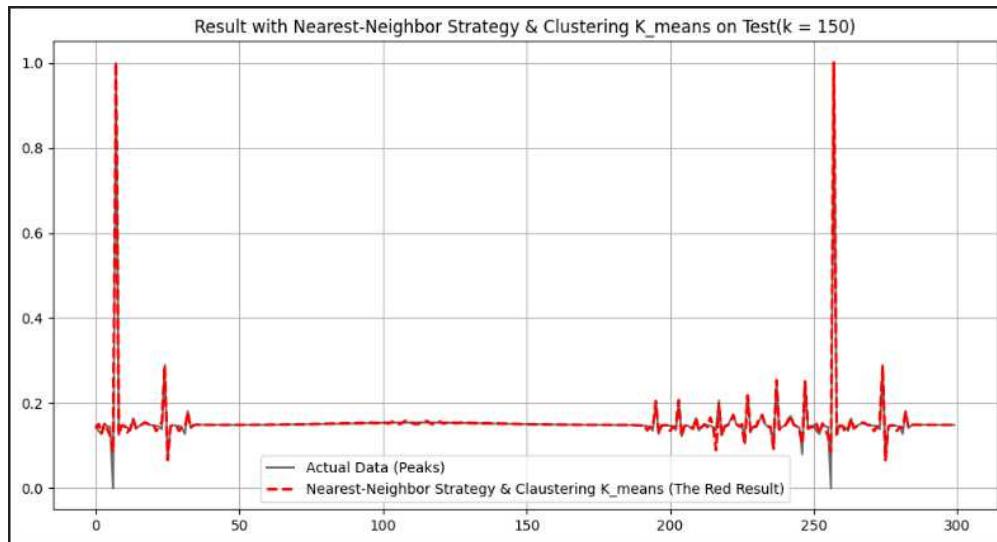


```

3   phi_train = phi_matrix_variable_sigma(X_train, kmeans_centers, sigmas)
4
5   phi_test   = phi_matrix_variable_sigma(X_test,   kmeans_centers, sigmas)
6
7   alpha = np.linalg.pinv(phi_train).dot(y_train)
8
9   y_train_pred = phi_train.dot(alpha)
10
11 y_test_pred  = phi_test.dot(alpha)
12
13
14 rmse_train = rmse(y_train, y_train_pred)
15 rmse_test  = rmse(y_test, y_test_pred)
16
17 print("Train RMSE:", rmse_train)
18 print("Test RMSE:", rmse_test)

```

۳۳: آموزش مدل و ارزیابی آن با استفاده از دو استراتژی Code



شکل ۱۸: خروجی واقعی و پیش‌بینی مدل با استفاده از دو استراتژی kmeans و NearestNeighbor

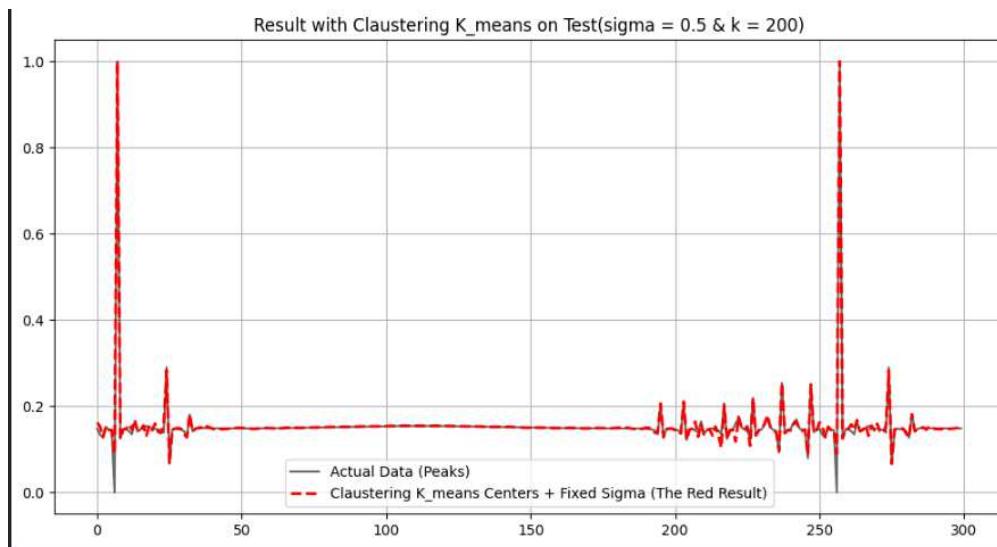
همانطور که مشاهده می‌نماییم هنگامی که حساسیت نورون‌ها بیشتر می‌شوند باز هم نتایج بهتری را داریم چرا که وقتی برای هر نورون به طور منحصر به فرد مقدار واریانس را محاسبه می‌نماییم، به مقدار خیلی زیادی روی دیتای آموزش فیت می‌شویم و عملاً بیش برآش می‌باشد رخ دهد اما همانطور که گفتیم به دلیل تمیز بودن دیتایی که در اختیار داریم نتایج مجدد بهتر می‌شوند.

تمامی موارد فوق را مجدداً با استفاده از کتابخانه `sklearn` نیز تکرار می‌نماییم. نتایج بهتری به دست می‌آید. در اینجا ما به ازای  $k=150$  و  $k=200$  این کتابخانه را مورد آزمون قرار دادیم. در نهایت متوجه شدیم که به ازای مقدار  $k=200$  بهترین نتیجه با کمترین مقدار

حاصل می‌شود. خروجی‌ها به صورت زیر به دست می‌آیند:

Result for Train = 0.0070110763718428965

Result for Test = 0.009160098255812849

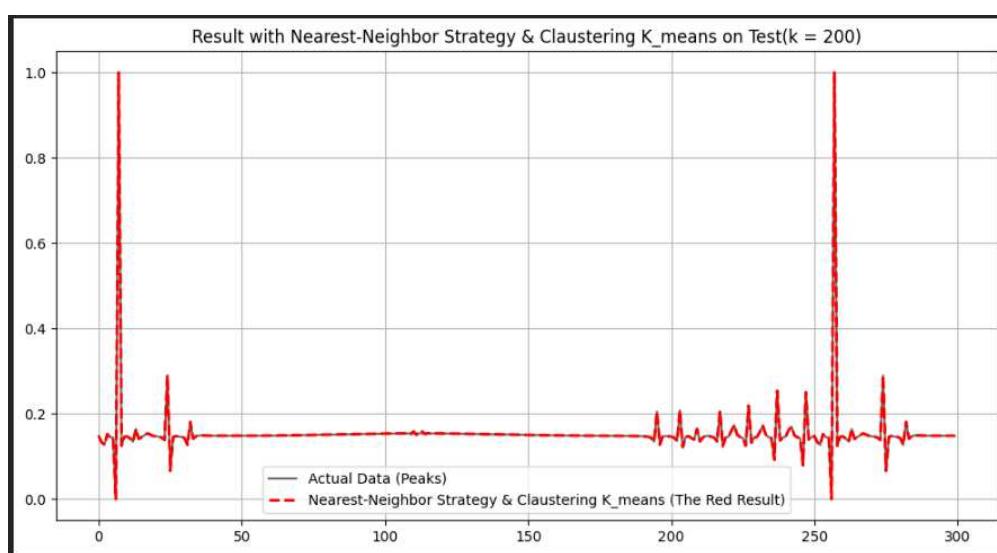


شکل ۱۹: خروجی واقعی و خروجی پیش‌بینی شده با استفاده از kmeans(sklearn)

نتایج بعد از اعمال NearestNeighbor به صورت زیر حاصل می‌شود:

Train RMSE = 0.0005480327488016946

Test RMSE = 0.00048613151684708466



شکل ۲۰: خروجی واقعی و خروجی پیش‌بینی شده با استفاده از kmeans(sklearn) و NearestNeighbor



همانطور که مشاهده می‌شود به طور کامل توانستیم مقادیر تست را پیش‌بینی نماییم و مقدار خطأ به صورت زیادی پایین است.

#### ۱۴.۶ نتایج بهتر kmeans clustering نسبت به روش تصادفی

در شبکه‌های RBF، انتخاب مناسب مراکز  $\mu_k$  نقش اساسی در کیفیت تقریب تابع و توانایی تعمیم‌پذیری دارد. روش K-Means از نظر تئوری و عملی بر انتخاب تصادفی مراکز برتزی دارد. دلایل این برتری به صورت زیر بیان می‌شود:

##### ۱. پوشش بهینه‌تر فضای ورودی

الگوریتم K-Means مراکزی را انتخاب می‌کند که "میانگین خوش‌های واقعی داده" هستند؛ یعنی نقاطی که بیشترین تراکم داده در اطراف آنها قرار دارد. بنابراین:

$$\mu_k \approx \text{cluster of Centroid}_k.$$

این کار باعث می‌شود توابع RBF دقیقاً در نواحی مهم و پرتردد داده فعال شوند. در مقابل، انتخاب تصادفی ممکن است مراکزی را در نواحی دورافتاده یا کم‌اهمیت قرار دهد و بخش‌های مهم از فضای بدون پوشش بمانند.

##### ۲. کاهش تداخل و هم‌پوشانی غیرضروری

در K-Means، فاصله‌ی بین مراکز به طور طبیعی بیشینه می‌شود زیرا مراکز در خوش‌های متفاوت داده قرار می‌گیرند. این موضوع باعث می‌شود توابع RBF تا حد ممکن نواحی جداگانه‌ای را پوشش دهند. اما در انتخاب تصادفی ممکن است چند مرکز در یک ناحیه نزدیک به هم انتخاب شوند که منجر به تداخل زیاد و کاهش کارایی تقریب می‌شود.

##### ۳. کاهش خطای بازسازی داده

از دیدگاه بهینه‌سازی، K-Means مراکزی را انتخاب می‌کند که مجموع فاصلهٔ مربعی داده‌ها تا نزدیک‌ترین مرکز را کمینه می‌کند:

$$\min_{\mu_1, \dots, \mu_K} \sum_{i=1}^N \min_k \|x_i - \mu_k\|^2.$$

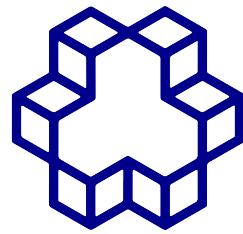
این تابع هزینه، معیار دقیقی برای «بهترین جای‌گذاری مراکز جهت بازسازی داده» است. انتخاب تصادفی هیچ تضمینی برای کوچک بودن این خطأ ندارد.

##### ۴. بهبود کیفیت تقریب موضعی RBF

در شبکه‌های RBF هر نورون تنها یک ناحیه محلی را مدل می‌کند. زمانی که مراکز با K-Means تعیین شوند:

$$\phi_k(x) = \exp\left(-\frac{\|x - \mu_k\|^2}{\sigma_k^2}\right)$$

دقیقاً در نواحی داده فعال می‌شوند و مدل توانایی بیشتری در یادگیری رفتارهای موضعی تابع هدف دارد. انتخاب تصادفی ممکن است باعث شود برخی نورون‌ها هرگز فعال نشوند یا فعال‌سازی آن‌ها بسیار کم باشد.



دانشگاه صنعتی خواجه نصیرالدین طوسی  
دانشکده مهندسی برق

## درس مبانی سیستم‌های هوشمند

استاد: دکتر مهدی علیاری

مینی پروژه دوم بخش دوم

محمدامین محمدیون شبستری	نام و نام خانوادگی
۴۰۱۲۲۵۰۳	شماره دانشجویی
محمد سبحان سخایی	نام و نام خانوادگی
۴۰۱۱۹۲۵۳	شماره دانشجویی
مبینا یوسفی مقدم	نام و نام خانوادگی
۴۰۱۲۴۰۹۳	شماره دانشجویی
۱۴۰۴ آذر	تاریخ
لینک‌های مربوط به کد و دیتاست :	
کد + دیتاست   لینک گیت‌هاب	



## فهرست مطالب

۵	۱ بخش دوم - پیاده‌سازی RBFNN تطبیقی با الهام از M-RAN	۱	
۵	۱.۱ برسی M-RAN	۱.۱	
۵	۱.۱.۱ استراتژی هرس (Pruning Strategy)	۱.۱.۱	
۶	۲.۱ بخش اول - یادگیری ترتیبی و معیار رشد	۲.۱	
۶	۱.۲.۱ تحلیل کد و منطق رشد	۱.۲.۱	
۶	۲.۲.۱ تحلیل دقیق معیارها و پرسش‌های مفهومی	۲.۲.۱	
۷	۳.۱ بخش دوم - امتیازی - پیاده‌سازی RBFNN تطبیقی با استراتژی هرس	۳.۱	
۷	۱.۳.۱ ساختار و مقداردهی اولیه کلاس (Initialization)	۱.۳.۱	
۸	۲.۳.۱ هسته محاسباتی و پیش‌بینی (Kernel , Prediction)	۲.۳.۱	
۸	۳.۳.۱ چرخه یادگیری ترتیبی (Sequential Learning Loop)	۳.۳.۱	
۹	۴.۳.۱ تحلیل نتایج و مقایسه عملکرد	۴.۳.۱	
۹	۵.۳.۱ تحلیل نمودارهای عملکرد شبکه M-RAN	۵.۳.۱	
۱۰	۶.۳.۱ نتیجه‌گیری نهایی	۶.۳.۱	
۱۰	کدنویسی - خطی و مطالعه پارامتر $\epsilon$		۲
۱۰	۱.۲ توضیحات پایگاهداده Breast Cancer Wisconsin (Diagnostic)	۱.۲	
۱۷	۲.۲ PCA	۲.۲	
۱۸	۳.۲ تقسیم دیتا به آموزش، اعتبارسنجی و تست	۳.۲	
۱۹	۴.۲ آموزش و گزارش حاشیه	۴.۲	
۲۱	۵.۲ مطالعه $C$	۵.۲	
۲۳	۶.۲ نتایج آزمایش SVM با هسته خطی برای مقادیر مختلف $C$	۶.۲	
۲۳	۱.۶.۲ مرور روابط نظری SVM	۱.۶.۲	
۲۳	۲.۶.۲ تحلیل تجربی	۲.۶.۲	
۲۴	۳.۶.۲ جمع‌بندی نهایی	۳.۶.۲	
۲۵	۷.۲ نمودارهای ارزیابی	۷.۲	
۲۶	۸.۲ تفسیر نمودارهای ROC و Precision-Recall برای مدل SVM با پارامتر $C = 0.1$	۸.۲	



## فهرست تصاویر

نمودارهای عملکرد M-RAN: (بالا) نوسانات تعداد نورون‌ها، (وسط) پیش‌بینی روی داده تست، (پایین) نرخ کاهش اپسیلون.	۱
.....	۱۰
توزيع کلاس‌ها	۲
.....	۱۶
2D PCA	۳
.....	۱۸
نتایج ارزیابی بر روی مجموعه اعتبارسنجی و تست	۴
.....	۲۱
جدول نتایج برای مطالعه پارامتر C	۵
.....	۲۵
Precision-Recall و ROC منحنی‌های	۶
.....	۲۶



## فهرست جداول

۹	.....	مقایسه عملکرد مدل‌های ایستا و تطبیقی	۱
۱۹	.....	ابعاد مجموعه‌های داده	۲



## فهرست برنامه‌ها

۱۲	.....	بالا آوردن دیتا بر روی colab	۱
۱۲	.....	خواندن و نمایس ۵ سطر اول دیتاست	۲
۱۳	.....	ترسیم ماتریس همبستگی بین ویژگی‌ها	۳
۱۴	.....	ترسیم نمودارهای pair plot	۴
۱۵	.....	توزیع کلاس‌ها	۵
۱۶	.....	نرم‌السازی و Encode متغیر هدف	۶
۱۷	.....	2D PCA	۷
۱۸	.....	2D PCA	۸
۱۹	.....	آموزش مدل	۹
۱۹	.....	محاسبه وزن‌های مدل و حاشیه	۱۰
۱۹	.....	محاسبه خروجی‌ها	۱۱
۲۰	.....	تابع ارزیابی	۱۲
۲۰	.....	محاسبه شاخص‌های ارزیابی	۱۳
۲۱	.....	آموزش مدل از طریق مقادیر مختلف C و گرفتن جدول خواسته شده	۱۴
۲۲	.....	تابع ارزیابی برای مطالعه پارامتر C	۱۵
۲۵	Precision-Recall	آموزش مدل با بهترین مقدار پارامتر C و به دست آوردن منحنی‌های ROC و Precision-Recall	۱۶



## ۱ بخش دوم - پیاده‌سازی RBFNN تطبیقی با الهام از M-RAN

### ۱.۱ بررسی M-RAN

الگوریتم M-RAN یا Minimal Resource Allocation Network نسخه تکامل یافته‌ای از الگوریتم RAN است. هدف اصلی این الگوریتم، ایجاد یک شبکه عصبی با تعداد بهینه نورون‌ها برای یادگیری یک نگاشت است، به طوری که شبکه نه بیش از حد بزرگ باشد و نه کارایی خود را از دست بدهد.

#### تفاوت با RAN و مفاهیم پایه

در شبکه‌های RBF کلاسیک، تعداد مراکز (نورون‌های مخفی) معمولاً ثابت است. الگوریتم RAN پیشنهاد داد که شبکه از صفر شروع شود و با ورود داده‌های جدید، به صورت پویا رشد کند.

- الگوریتم RAN: استراتژی اصلی آن «رشد» (Growth) است. یعنی اگر خطای شبکه زیاد باشد و داده ورودی از مراکز فعلی دور باشد، یک نورون جدید اضافه می‌کند. مشکل RAN این است که مکانیزمی برای حذف نورون‌ها ندارد؛ بنابراین تعداد نورون‌ها همواره افزایش می‌یابد که می‌تواند منجر به افزونگی (Redundancy) و بیش‌برازش (Overfitting) شود.
- الگوریتم M-RAN: تفاوت کلیدی M-RAN در اضافه کردن مکانیزم «هرس» (Pruning) است. این الگوریتم علاوه بر رشد، به طور مداوم بررسی می‌کند که آیا نورونی وجود دارد که سهم ناچیزی در خروجی داشته باشد؟ اگر چنین باشد، آن را حذف می‌کند تا ساختار شبکه «مینیمال» باقی بماند.

#### ۱.۱.۱ استراتژی هرس (Pruning Strategy)

استراتژی هرس برای جلوگیری از رشد بی‌رویه شبکه استفاده می‌شود. ایده اصلی این است که اگر فعالیت (Activity) یا سهم یک نورون در خروجی برای مدت زمان مشخصی کمتر از یک حد آستانه باشد، آن نورون زائد تشخیص داده شده و حذف می‌شود. برای فرمول‌بندی ریاضی، فرض کنید خروجی  $k$ -امین نورون مخفی برای ورودی  $x$  برابر با  $\phi_k(x)$  و وزن خروجی آن  $w_k$  باشد. سهم نسبی این نورون در خروجی کل شبکه به صورت زیر محاسبه می‌شود:

$$r_k^{(n)} = \frac{\|w_k^{(n)}\| \phi_k(x^{(n)})}{\max_{j=1\dots K} (\|w_j^{(n)}\| \phi_j(x^{(n)}))} \quad (1)$$

در این رابطه:

- $r_k^{(n)}$  نسبت خروجی نورون  $k$  به ماکزیمم خروجی تمام نورون‌ها در لحظه  $n$  است.
- این معیار نشان می‌دهد که نورون  $k$  در مقایسه با قوی‌ترین نورون فعال، چقدر اهمیت دارد.

الگوریتم هرس با پنجه‌های لغزان (Sliding Window): تصمیم برای حذف یک نورون نباید لحظه‌ای باشد. در M-RAN، اگر نسبت  $r_k$  برای  $M$  مشاهده متواتی (یک پنجه‌های زمانی) کمتر از آستانه  $\delta$  باشد، آنگاه نورون  $k$  حذف می‌شود:



$$\text{If } r_k^{(n-m)} < \delta, \quad \forall m \in \{0, 1, \dots, M-1\} \implies \text{Prune unit } k \quad (2)$$

این کار باعث می‌شود نورون‌هایی که زمانی مفید بوده‌اند اما اکنون (به دلیل تغییر توزیع داده‌ها یا همپوشانی با نورون‌های جدید) بی‌استفاده شده‌اند، از حافظه آزاد شوند.

## ۲.۱ بخش اول - یادگیری ترتیبی و معیار رشد

در این بخش تابعی طراحی شده است (`check`) که تصمیم می‌گیرد آیا شبکه برای یادگیری نمونه جدید نیاز به اضافه کردن یک نورون جدید (Growth) دارد یا خیر. این تصمیم بر اساس دو معیار کلیدی گرفته می‌شود: «خطای پیش‌بینی» و «تازگی داده» (فاصله).

### ۱.۲.۱ تحلیل کد و منطق رشد

کد ارائه شده مراحل زیر را برای تصمیم‌گیری طی می‌کند:

۱. محاسبه خطای ابتدا قدر مطلق خطای پیش‌بینی محاسبه می‌شود:

$$E = |y_{\text{true}} - y_{\text{pred}}| \quad (3)$$

اگر  $E \leq e_{\min}$  باشد، یعنی شبکه عملکرد خوبی دارد و تابع `False` بر می‌گردد.

۲. بررسی وجود نورون: اگر تعداد مرکز صفر باشد (`len(centers) == 0`), بلا فاصله `True` بر می‌گردد تا اولین نورون ساخته شود.

۳. محاسبه فاصله (معیار تازگی): فاصله اقلیدسی داده جدید  $x$  از تمام مرکز موجود  $c_k$  محاسبه شده و کمترین فاصله یافت می‌شود:

$$d_{\min} = \min_k \|x - c_k\| \quad (4)$$

۴. تصمیم نهایی: اگر فاصله  $d_{\min}$  بیشتر از شعاع اثرگذاری  $\epsilon$  (epsilon) باشد، تابع `True` بر می‌گردد.

بنابراین شرط نهایی برای رشد شبکه به صورت منطقی زیر است:

$$\text{Growth} \iff (E > e_{\min}) \text{ AND } (d_{\min} > \epsilon) \quad (5)$$

### ۲.۲.۱ تحلیل دقیق معیارها و پرسش‌های مفهومی

۱. چرا هر دو معیار مهم هستند؟ این دو معیار تعادل بین «یادگیری» و «تعییم» را برقرار می‌کنند:

- معیار خطای  $(E > e_{\min})$ : نشان‌دهنده نیاز عملکردی است. می‌گوید «شبکه در اینجا اشتباه کرده است، پس باید کاری انجام دهیم».

- معیار فاصله  $(d_{\min} > \epsilon)$ : نشان‌دهنده نیاز ساختاری است. می‌گوید «شبکه در اینجا هیچ نماینده‌ای (نورونی) ندارد».



ترکیب این دو تضمین می‌کند که ما فقط زمانی ساختار شبکه را تغییر می‌دهیم که هم عملکرد ضعیف باشد و هم دانش قبلی در آن ناحیه وجود نداشته باشد.

۲. چه اتفاقی می‌افتد اگر فقط از معیار خطا استفاده کنیم؟ اگر شرط فاصله را حذف کنیم، شبکه برای هر داده‌ای که خطای آن زیاد باشد، یک نورون جدید می‌سازد.

- مشکل: فرض کید یک داده نویزی (Outlier) دقیقاً کنار یکی از مراکز قبلی قرار بگیرد. فاصله کم است اما خطای زیاد است. اگر فقط به خطا نگاه کنیم، یک نورون جدید روی نورون قبلی می‌سازیم تا نویز را حفظ کند. این منجر به بیشبرازش شدید (Overfitting) و رشد انفجاری تعداد نورون‌ها می‌شود. در چنین حالتی راه درست، آپدیت وزن‌هاست نه ساخت نورون جدید.

۳. چه اتفاقی می‌افتد اگر فقط از معیار تازگی (فاصله) استفاده کنیم؟ اگر شرط خطا را حذف کنیم، شبکه صرفاً هندسه فضای را پر می‌کند.

- مشکل: فرض کنید تابع هدف در یک ناحیه وسیع مقدار ثابتی دارد (مثلاً صفر). شبکه یک نورون می‌سازد و به درستی صفر را پیش‌بینی می‌کند. حال داده‌ای جدید در فاصله دور می‌آید که مقدارش همچنان صفر است. پیش‌بینی شبکه صحیح است (خطای کم است)، اما چون فاصله زیاد است، اگر فقط معیار فاصله باشد، شبکه بیهوده یک نورون جدید می‌سازد. این منجر به هدر رفت منابع (Waste of Resources) می‌شود.

۴. تأثیر افزایش و کاهش پارامتر  $\epsilon$  (epsilon): پارامتر  $\epsilon$  رزولوشن (Resolution) یا دقت فضایی شبکه را کنترل می‌کند.

- افزایش  $\epsilon$ : شرط  $d_{\min} < \epsilon$  سخت‌تر ارضا می‌شود.

- تعداد نورون‌ها کمتر می‌شود (شبکه تنک‌تر).

- همپوشانی توابع پایه بیشتر شده و مدل کلی نگرتر (General) می‌شود.

- خطر Underfitting (کم‌برازش) برای توابع پیچیده وجود دارد.

- کاهش  $\epsilon$ : شرط  $d_{\min} > \epsilon$  راحت‌تر ارضا می‌شود.

- تعداد نورون‌ها بیشتر می‌شود (شبکه متراکم‌تر).

- شبکه می‌تواند جزئیات فرکانس بالا و تغییرات سریع تابع را یاد بگیرد.

- خطر Overfitting افزایش می‌یابد.

### ۳.۱ بخش دوم - امتیازی - پیاده‌سازی RBFNN تطبیقی با استراتژی هرس

در این بخش، به طراحی و پیاده‌سازی یک کلاس پایتون برای شبکه عصبی RBF تطبیقی با الهام از الگوریتم M-RAN می‌پردازیم. هدف اصلی این کلاس، ایجاد یک مدل یادگیری است که بتواند ساختار خود را متناسب با پیچیدگی داده‌ها تنظیم کند؛ یعنی در صورت نیاز نورون جدید اضافه کند (رشد) و در صورت عدم کارایی، نورون‌های زائد را حذف کند (هرس).

#### ۱.۳.۱ ساختار و مقداردهی اولیه کلاس (Initialization)

کلاس MRAN\_rbfnn به گونه‌ای طراحی شده است که با یک شبکه خالی شروع به کار می‌کند. در متد سازنده (`__init__`)، پارامترهای حیاتی الگوریتم مقداردهی می‌شوند:



- لیست‌های ساختاری: لیست‌های centers و sigmas به صورت خالی تعریف می‌شوند تا شبکه بدون هیچ پیش‌فرضی از داده‌ها آغاز به کار کند.
- پارامترهای تطبیقی: پارامتر epsilon که شعاع همسایگی و رزولوشن شبکه را تعیین می‌کند، با مقدار اولیه epsilon\_max آغاز می‌شود و در طول زمان با نرخ decay\_rate کاهش می‌یابد تا شبکه ابتدا کلیات و سپس جزئیات را یاد بگیرد.
- پارامترهای هرس: آستانه prun\_thresh و پنجره زمانی prun\_window برای شناسایی نورون‌های کم‌اهمیت تنظیم می‌شوند.

### ۲.۳.۱ هسته محاسباتی و پیش‌بینی (Kernel, Prediction)

محاسبات اصلی شبکه در دو متده gaussian\_kernel و predict انجام می‌شود:

- هسته گوسی: فعالیت هر نورون مخفی با استفاده از تابع پایه شعاعی گوسی محاسبه می‌شود. این تابع میزان شباهت (فاصله اقلیدسی معکوس) بین ورودی و مرکز نورون را اندازه‌گیری می‌کند.
- پیش‌بینی: خروجی نهایی شبکه حاصل ترکیب خطی خروجی‌های توابع پایه و وزن‌های سیناپسی است. مکانیزم پیش‌بینی به گونه‌ای پیاده‌سازی شده است که حتی در صورت خالی بودن شبکه (در اولین گام)، با بازگرداندن مقدار صفر، پایداری کد را حفظ کند.

### ۳.۳.۱ چرخه یادگیری ترتیبی (Sequential Learning Loop)

متده train قلب تپنده الگوریتم است که فرآیند یادگیری نظارت شده را به صورت آنلاین (نمونه به نمونه) اجرا می‌کند. در هر گام، خطای پیش‌بینی لحظه‌ای ( $e_n$ ) محاسبه شده و سپس شبکه وارد فاز تصمیم‌گیری می‌شود.

#### ۱. استراتژی رشد (Growth Strategy):

تصمیم برای اضافه کردن نورون جدید بر اساس دو معیار همزمان گرفته می‌شود:

- معیار خطای آیا قدر مطلق خطای بیشتر از  $e_{min}$  است؟

- معیار تازگی: آیا فاصله داده ورودی تا نزدیک‌ترین مرکز موجود، بیشتر از  $\epsilon$  است؟

اگر هر دو شرط برقرار باشند، یک نورون جدید با مرکزی برابر با داده ورودی و وزنی متناسب با خطای ایجاد می‌شود. عرض این نورون ( $\sigma$ ) نیز با توجه به فاصله تا نزدیک‌ترین همسایه و پارامتر همپوشانی  $\eta$  تنظیم می‌شود تا پوشش مناسبی در فضای ایجاد کند.

#### ۲. به روزرسانی پارامترها (Parameter Update):

اگر شرایط رشد محقق نشود، به این معنی است که ساختار فعلی پتانسیل یادگیری را دارد اما نیاز به تنظیم دارد. در این حالت، وزن‌های خروجی با استفاده از قانون گرادیان نزولی LMS به روزرسانی می‌شوند ( $\Delta w = \eta \cdot e \cdot \phi$ ). برای حفظ پایداری و تطابق با نسخه ساده‌شده M-RAN، مراکز و عرض‌ها در این فاز ثابت نگه داشته می‌شوند.

#### ۳. استراتژی هرس (Pruning Strategy):

در انتهای هر گام، شبکه خود را پالایش می‌کند. سهم نسبی هر نورون در خروجی کل محاسبه می‌شود. اگر این سهم کمتر از حد آستانه prun\_thresh باشد، یک شمارنده برای آن نورون فعال می‌شود. چنانچه نورونی برای مدت زمان prun\_window (مثلاً ۶۰ گام متوالی) بی‌استفاده باقی بماند، به عنوان «بار اضافه» تشخیص داده شده و به طور کامل از ساختار شبکه حذف می‌شود. این مکانیزم کلیدی‌ترین بخش برای مینیمال نگه داشتن توپولوژی شبکه است.



### ۴.۳.۱ تحلیل نتایج و مقایسه عملکرد

الف) مقایسه آماری مدل‌ها:

مدل	استراتژی آموزش	تعداد نورون	RMSE تست	وضعیت
Static RBF	K-Means + LLS	۱۵۰	۰.۰۰۰۴۸	عالی
M-RAN	Sequential + Shuffle	۱۸	۰.۰۲۶۸	خوب
M-RAN	Sequential (No Shuffle)	۸	۰.۰۸۰۷	ضعیف

جدول ۱: مقایسه عملکرد مدل‌های ایستا و تطبیقی

تحلیل دقیق در برابر پیچیدگی (Accuracy vs. Complexity): شبکه ایستا اگرچه خطای بسیار پایینی دارد، اما استفاده از ۱۵۰ نورون برای این سیستم نشان‌دهنده Over-parameterization است. در مقابل، شبکه M-RAN می‌توانسته است با کاهش ۸۸ درصدی تعداد نورون‌ها (تنهای ۱۸ نورون)، به خطای قابل قبول (۰.۰۲۶۸) برسد. این نشان می‌دهد که شبکه تطبیقی موفق شده است دینامیک اصلی سیستم را به شکلی بسیار فشرده‌تر استخراج کند.

ب) تأثیر حیاتی شافل کردن (Shuffling): آزمایش‌های نشان می‌دهد که بر زدن داده‌ها نقش حیاتی در موفقیت M-RAN دارد. در حالت بدون شافل، شبکه دچار «فراموشی فاجعه‌بار» (Catastrophic Forgetting) می‌شود و تنها روی آخرین داده‌ها تمرکز می‌کند (خطای بالا). اما با شافل کردن، شبکه مجبور می‌شود توزیع کلی فضای ایجادگیر که منجر به رشد منطقی نورون‌ها (تا ۱۸ عدد) و کاهش چشمگیر خطای می‌شود.

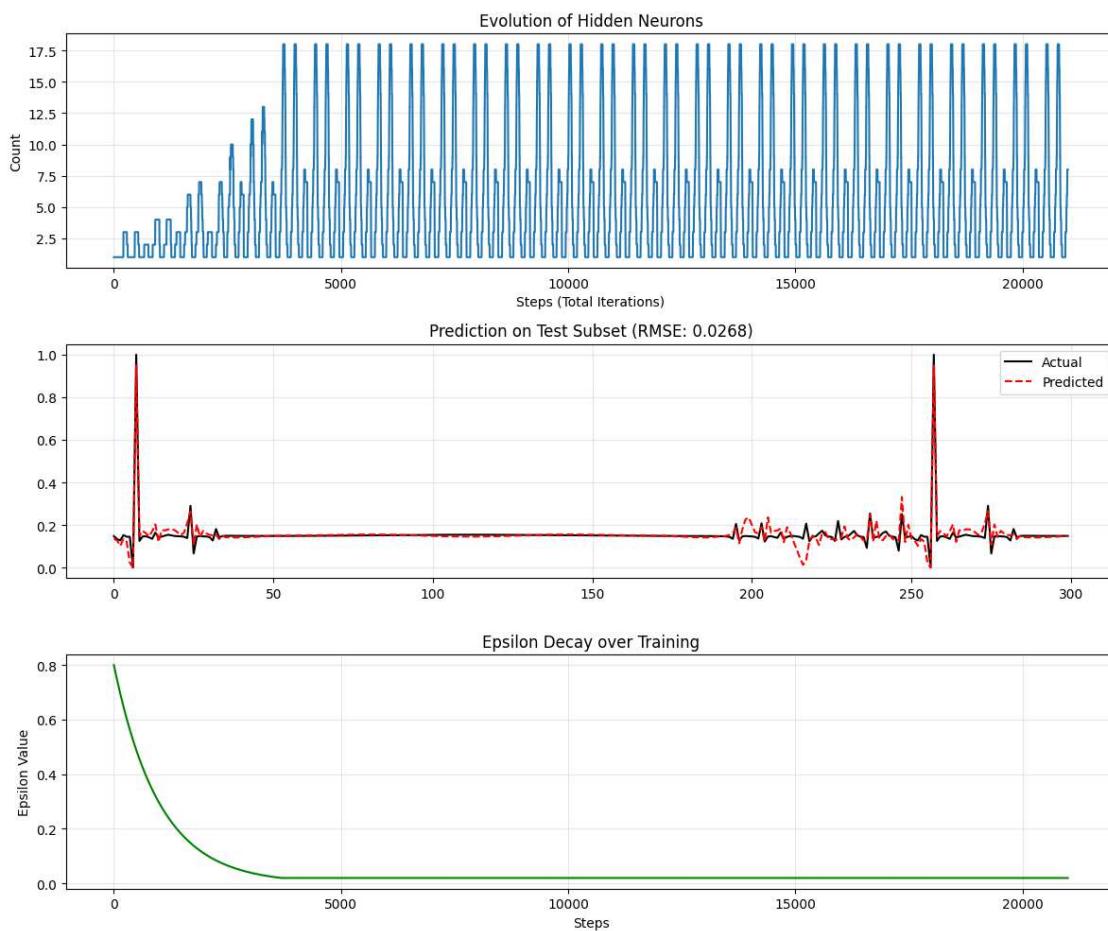
### ۵.۳.۱ تحلیل نمودارهای عملکرد شبکه M-RAN

نمودارهای حاصل از آموزش شبکه M-RAN حاوی نکات مهمی درباره دینامیک یادگیری هستند:

۱. تکامل تعداد نورون‌ها (Neuron Evolution): نمودار اول رفتاری دندان‌ارهای (Sawtooth) را نشان می‌دهد؛ تعداد نورون‌ها بالا می‌رود و سپس به شدت افت می‌کند. این نوسان شدید نتیجه مستقیم تعامل بین «شافل کردن داده‌ها» و «پنجه هرس» است. زمانی که شبکه داده‌های مربوط به یک ناحیه خاص را می‌بیند، نورون‌های نواحی دیگر غیرفعال شده و توسط مکانیزم هرس حذف می‌شوند. با تغییر نوع داده‌ها، شبکه مجبور به بازسازی (رشد مجدد) آن نورون‌ها می‌شود. نکته مثبت اینجاست که سقف تعداد نورون‌ها روی عدد ۱۸ تثبیت شده است که نشان‌دهنده رسیدن به «ظرفیت یادگیری» مورد نیاز است.

۲. پیش‌بینی روی داده‌های تست (Prediction): نمودار دوم انطباق بسیار خوب خط پیش‌بینی (قرمز) بر داده‌های واقعی (مشکی) را نشان می‌دهد. خطای جزئی مشاهده شده در نقاط پیک (تغییرات تیز) ناشی از مقدار  $\text{epsilon}_{\min}$  است. شبکه برای حفظ سادگی مدل، از مدل کردن جزئیات فرکانس بالا که کوچکتر از شاعع همسایگی استفاده نموده است. این رفتار نشان‌دهنده تعمیم‌پذیری (Generalization) مناسب مدل و عدم بیش‌برازش روی نویزهای است.

۳. کاهش اپسیلون (Epsilon Decay): منحنی سوم نشان می‌دهد که پارامتر  $\text{epsilon}$  پس از حدود ۴۰۰۰ گام به مقدار حداقل خود رسیده و ثابت شده است. همزمانی تثبیت اپسیلون با تثبیت سقف تعداد نورون‌ها در نمودار اول، نشان می‌دهد که شبکه فاز «اکتشاف» (Exploration) را با موفقیت پشت سر گذاشته و وارد فاز بهره‌برداری و تنظیم دقیق شده است.



شکل ۱: نمودارهای عملکرد M-RAN: (بالا) نوسانات تعداد نورون‌ها، (وسط) پیش‌بینی روی داده تست، (پایین) نرخ کاهش اپسیلون.

### ۶.۳.۱ نتیجه‌گیری نهایی

آزمایش و پیاده‌سازی M-RAN نشان داد که این الگوریتم راهکاری قدرتمند برای مسائلی است که محدودیت منابع دارند. اگرچه دقت مطلق آن کمی پایین‌تر از روش‌های دسته‌ای سنگین است، اما وزن‌هایی مانند Minimal Topology (کاهش حجم محاسبات و حافظه)، قابلیت Online Learning (تطبیق با تغییرات محیط) و سرعت بالای استنتاج، آن را به گزینه‌ای ایده‌آل برای کاربردهای بلاذرنگ و سیستم‌های نهفته (Embedded Systems) تبدیل می‌کند.

## ۲ کدنویسی - **svm** خطی و مطالعه پارامتر **c**

### ۱.۲ توضیحات پایگاهداده **Breast Cancer Wisconsin (Diagnostic)**

این پایگاهداده بر اساس تصویر دیجیتال‌سازی شده نمونه (Fine Needle Aspirate FNA) از توده پستان ایجاد شده است. در این تصاویر، ویژگی‌هایی استخراج می‌شوند که مشخصه‌های هسته سلول‌های موجود در نمونه را توصیف می‌کنند. فضای سه بعدی مورد استفاده در تحلیل این داده‌ها نخستین بار در پژوهش زیر معرفی شده است:



Lin-Two of Discrimination Programming Linear "Robust Mangasarian, L. O. and Bennett P. K. .۳۴-۲۳) ۱۹۹۲( Software, and Methods Optimization Sets," Inseparable early

این پایگاه از طریق سرور FTP دانشگاه ویسکانسین قابل دریافت است:

ftp.cs.wisc.edu •

• مسیر: math-prog/cpo-dataset/machine-learn/WDBC/

همچنین نسخه رسمی آن در مخزن UCI Machine Learning Repository در دسترس است:

[https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

## ساختار داده‌ها

۱. شماره شناسایی (ID)

۲. تشخیص نهایی  $M = \text{بدخیم}, B = \text{خوش خیم}$

۳. ۳۰ ویژگی اندازه‌گیری شده

## ویژگی‌ها

برای هر هسته سلولی، ده ویژگی اصلی اندازه‌گیری شده است:

۱. شعاع: میانگین فاصله از مرکز تا محیط

۲. بافت: انحراف معیار شدت روشناختی

۳. محیط

۴. مساحت

۵. صافی: میزان تغییرات موضعی شعاع

۶. فشردگی:  $\frac{\text{perimeter}^2}{\text{area}} - 1$

۷. فرورفتگی: شدت نواحی با تغیر

۸. نقاط فرورفتگی: تعداد نواحی مکعب

۹. تقارن

۱۰. بعد فرکتال: تقریب «خط ساحلی» منهای یک

برای هر یک از این ویژگی‌ها سه مقدار محاسبه شده است:



- میانگین (Mean)

- خطای استاندارد (Standard Error)

- بدترین مقدار (Worst)، که میانگین سه مقدار بیشینه است

در نتیجه، در مجموع ۳۰ ویژگی عددی برای هر نمونه ثبت شده است. نمونه‌هایی از فیلدها:

- فیلد ۳: شعاع میانگین (Mean Radius)

- فیلد ۱۳: خطای استاندارد شعاع (Radius SE)

- فیلد ۲۳: بدترین شعاع (Worst Radius)

تمام ویژگی‌ها با دقت چهار رقم معنادار ذخیره شده‌اند و هیچ مقدار گمشده‌ای در پایگاهداده وجود ندارد.

## توزیع کلاس‌ها

- ۳۵۷ نمونه خوش‌خیم

- ۲۱۲ نمونه بدخیم

در بالا توضیحاتی مربوط به دیتاست ارائه گردید. این توضیحات از سایت kaggle آورده شده است.  
حال که با جزئیات دیتاست و کارکرد آن آشنا شدیم به سراغ آوردن این دیتا بر روی colab می‌رویم. کد زیر بدین منظور زده شده است:

```

1 import kagglehub
2
3 # Download latest version
4 path = kagglehub.dataset_download("uciml/breast-cancer-wisconsin-data")
5
6 print("Path to datasetfiles:", path)

```

۱: بالا آوردن دیتا بر روی colab Code

حال که دیتا را بالا آوردیم، به وسیله کتابخانه pandas آن را می‌خوانیم و سپس ۵ سطر اول آن را نمایش می‌دهیم.

```

1 import pandas as pd
2 import os
3
4 files = os.listdir(path)
5 print("Files:", files)
6
7 csv_file = [f for f in files if f.endswith(".csv")][0]
8 csv_path = os.path.join(path, csv_file)
9
10 df = pd.read_csv(csv_path)

```



۱۲ df.head()

## ۲: خواندن و نمایش ۵ سطر اول دیتاست

مشاهده می‌نماییم که این دیتاست ۳۳ ستون دارد. ۳۰ ستون که ویژگی‌های مربوط به بیماری می‌باشند، یک ستون مربوط به شماره شناسایی بیماران، یک ستون تشخیص‌های مربوط به خوش‌خیم و یا بدخیم بودن بیماری و یک ستون نیز به دلیل یک ویرگولی که در انتهای ستون‌های ویژگی‌ها زده است، اضافه آمده. این ستون که با نام Unnamed است را در ادامه حذف می‌نماییم. قبل از آنکه به سراغ پاسخ به سوال‌ها برویم مقداری ویژگی‌هایی که در دیتاست وجود دارند را بررسی می‌نماییم.

باتابع df.describe() میانگین، انحراف معیار، مینیمم، ماکریمم و دیگر ویژگی‌های عددی مربوط به هر ستون ویژگی را بررسی می‌نماییم. همانطور که مشاهده می‌شود یکسری از ویژگی‌ها مثل mean و worst area دارای انحراف معیار زیادی هستند. ویژگی‌هایی که پراکنده‌تری دارند معمولاً ویژگی‌هایی هستند که بهتر می‌توانند ما را در تفکیک این دو کلاس یاری نمایند. در این دیتاست نیز از این ویژگی‌ها زیاد داریم. همچنین همانطور که در توضیحات ابتدایی نیز اشاره شد، این دیتاست داده‌گمشده ندارد که این موضوع را نیز با تابع df.info() مجدداً بررسی نمودیم. برای اینکه با همبستگی بین ویژگی‌ها بیشتر آشنا شویم و بینیم ویژگی‌ها چقدر بر روی یکدیگر تاثیر می‌گذارند و PCA می‌توانند تفکیک مناسبی را در دو بعد ایجاد کند یا نه به سراغ ترسیم ماتریس همبستگی بین ویژگی‌ها رفیم. هم به صورت عددی و هم به صورت نقشه حرارتی این ماتریس‌ها را گرفتیم:

```
 1 import seaborn as sns
 2 import matplotlib.pyplot as plt
 3
 4 if "id" in df.columns:
 5     df = df.drop(columns=["id"])
 6
 7
 8 df = df.loc[:, ~df.columns.str.contains("^Unnamed")]
 9
10
11 target = df["diagnosis"]
12
13
14 features = df.drop(columns=["diagnosis"])
15
16
17 corr = features.corr()
18
19
20 plt.figure(figsize=(18, 18))
21 sns.heatmap(corr, cmap="coolwarm", annot=False, square=True)
22 plt.title("Feature Correlation Matrix", fontsize=18)
23 plt.show()
24 numeric_df = df.select_dtypes(include=['int64', 'float64'])
25 corr_matrix = numeric_df.corr()
26
27 plt.figure(figsize=(20, 20))
28 sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
```



```

۲۹ plt.title('Heatmap of Feature Correlations With Seaborn Library', fontsize=14)
۳۰ plt.tight_layout()
۳۱ plt.savefig('/content/HeatmapofFeatureCorrelationsWithSeabornLibrary.png', dpi=300,
bbox_inches='tight')
۳۲ plt.show()

```

### ۳: ترسیم ماتریس همبستگی بین ویژگی‌ها

همانطور که از کد نیز مشخص است ستون‌های id و Unnamed کنار گذاشته شده‌اند. از آنجاییکه تصاویر ایجاد شده بسیار بزرگ هستند از آوردنشان در گزارش خودداری می‌نماییم و در کد نیز می‌توان آنها را مشاهده نمود. همانطور که مشاهده می‌شود خیلی از ویژگی‌ها با بدیگر همبستگی زیادی دارند. برای نمونه ویژگی‌های radius mean و perimeter mean با هم همبستگی زیادی دارند یا مثلا area و perimeter mean. البته انتظار نیز داریم که این اتفاق بیفتند به دلیل اینکه فرمول‌های مساحت و محیط از روی ساع و قطر دایره محاسبه می‌شوند، به وجود آمدن چنین هبستگی‌هایی طبیعی است. بسیاری از ویژگی‌هایی که در این دیتاست آمده‌اند نیز از طریق چنین فرمول‌هایی به هم مربوط می‌شوند بنابراین به نظر می‌رسد برای تفکیک این دو کلاس از یکدیگر نیازی نیست که از تمامی ویژگی‌هایی که در این دیتاست آمده‌اند استفاده نماییم چرا که بسیاری از آنها ویژگی‌های اضافی هستند که اطلاعاتشان در ستون‌های ویژگی دیگر وجود دارد. برای اینکه نسبت به این تحلیل خود مطمئن شویم برای هر دو ویژگی که در سه دسته میانگین، بدترین مقدار و خطای استاندارد قرار دارند نمودار pairplot ترسیم نمودیم. در این نمودارها نیز همبستگی بین یکسری از ویژگی‌ها دیده می‌شود و همچنین دیده این دو ویژگی در فضای وجود دارند که بتوانند تفکیک مناسبی را بر روی این کلاس برای تقسیم‌بندی ایجاد نمایند. به دلیل اینکه ابعاد این تصاویر بسیار بزرگ هستند از آوردنشان در گزارش خودداری نمودیم اما خروجی آنها در کد مشخص است. تمامی این نمودارها به ما نشان می‌دهند که انتظار این را داشته باشیم که PCA نیز تفکیک مناسبی را به ما بدهد. کد زده شده برای نمودارهای pair plot به صورت زیر است:

```

۱ mean_features = [
۲     'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean',
۳     'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', '
۴     fractal_dimension_mean'
۵ ]
۶
۷
۸
۹ sns.set(style="ticks")
۱۰ plt.figure(figsize=(15, 15))
۱۱ sns.pairplot(pairplot_df, hue='diagnosis', diag_kind='hist', corner=False)
۱۲ plt.suptitle("Pairplot of 10 Mean Features", fontsize=20, y=1.02)
۱۳ plt.show()
۱۴ worst_features = [
۱۵     'radius_worst',
۱۶     'texture_worst',
۱۷     'perimeter_worst',
۱۸     'area_worst',
۱۹     'smoothness_worst',
۲۰     'compactness_worst',
۲۱     'concavity_worst',

```



```
۲۲     'concave points_worst',
۲۳     'symmetry_worst',
۲۴     'fractal_dimension_worst'
۲۵ ]
۲۶
۲۷ pairplot_df = df[worst_features + ['diagnosis']]
۲۸
۲۹
۳۰ sns.set(style="ticks")
۳۱ plt.figure(figsize=(15, 15))
۳۲ sns.pairplot(pairplot_df, hue='diagnosis', diag_kind='hist', corner=False)
۳۳ plt.suptitle("Pairplot of 10 Worst Features", fontsize=20, y=1.02)
۳۴ plt.show()
۳۵ se_features = [
۳۶     'radius_se',
۳۷     'texture_se',
۳۸     'perimeter_se',
۳۹     'area_se',
۴۰     'smoothness_se',
۴۱     'compactness_se',
۴۲     'concavity_se',
۴۳     'concave points_se',
۴۴     'symmetry_se',
۴۵     'fractal_dimension_se'
۴۶ ]
۴۷ pairplot_df = df[se_features + ['diagnosis']]
۴۸
۴۹
۵۰ sns.set(style="ticks")
۵۱ plt.figure(figsize=(15, 15))
۵۲ sns.pairplot(pairplot_df, hue='diagnosis', diag_kind='hist', corner=False)
۵۳ plt.suptitle("Pairplot of 10 SE Features", fontsize=20, y=1.02)
۵۴ plt.show()
```

#### ۴: ترسیم نمودارهای pair plot

همچنین یک نمودار دایره‌ای برای نمایش توزیع کلاس‌ها ترسیم شده است. این نمودار نشان می‌دهد که کلاس خوش خیم دیتای بیشتری نسبت به کلاس بد خیم دارد که این موضوع یعنی یک عدم تعادلی در توزیع کلاس‌ها وجود دارد. این عدم تعادل می‌تواند باعث بایاس مدل روی یک کلاس شود اما در ادامه می‌بینیم که از آنچاییکه فضای ویژگی‌ها تفکیک مناسبی دارند این عدم تعادل تاثیر آنچنانی در پیش‌بینی مدل نخواهد گذاشت و مدل svm خطی به خوبی می‌تواند فضای این دو کلاس را با ایجاد ابر صفحه‌هایی از یکدیگر تفکیک نماید.

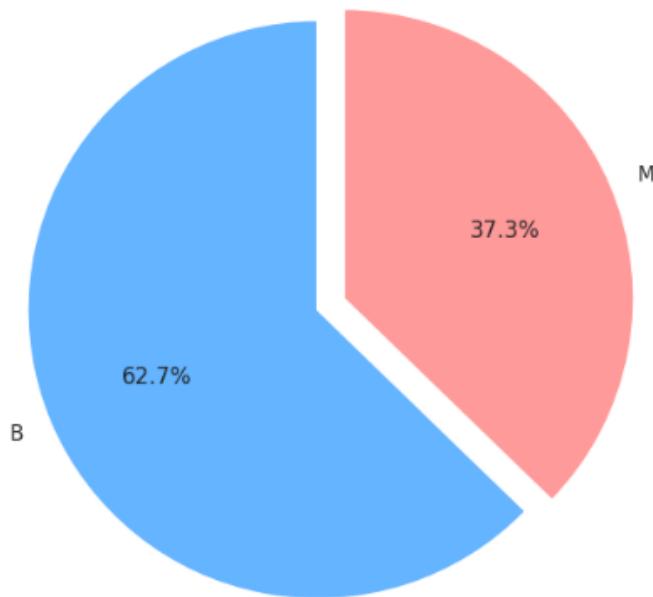
```
۱ class_counts = df['diagnosis'].value_counts()
۲ print("Number of samples per class:")
۳ print(class_counts)
۴
۵
```



```
 6 plt.figure(figsize=(7, 7))
 7 colors = ['#66b3ff', '#ff9999']
 8 plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=90, colors=
 9 colors, explode=(0.05, 0.05))
10 plt.title("Distribution of Diagnosis Classes", fontsize=16)
11 plt.show()
```

شکل ۵: توزیع کلاس‌ها

Distribution of Diagnosis Classes



شکل ۲: توزیع کلاس‌ها

در ادامه بر روی ویرگی‌ها یک نرمالسازی استاندارد و همچنین متغیر هدف را نیز Encode می‌نماییم. متغیر هدف از آنجاییکه از نوع ویرگی‌های دسته‌ای می‌باشد باید به ویرگی عددی تبدیل شود و عملاً تبدیل به صفر و یک شود. کلاس خوش خیم را لیل صفر و کلاس بدخیم را لیل صفر می‌زنیم. کد زیر بدین منظور زده شده است:

```
1 from sklearn.preprocessing import StandardScaler, LabelEncoder
2
3 X = features
4 y = target
5
6 le = LabelEncoder()
7 y_encoded = le.fit_transform(y)
8
9 scaler = StandardScaler()
```



```
۱۰ X_scaled = scaler.fit_transform(X)
```

۶: نرمالسازی و Encode متغیر هدف

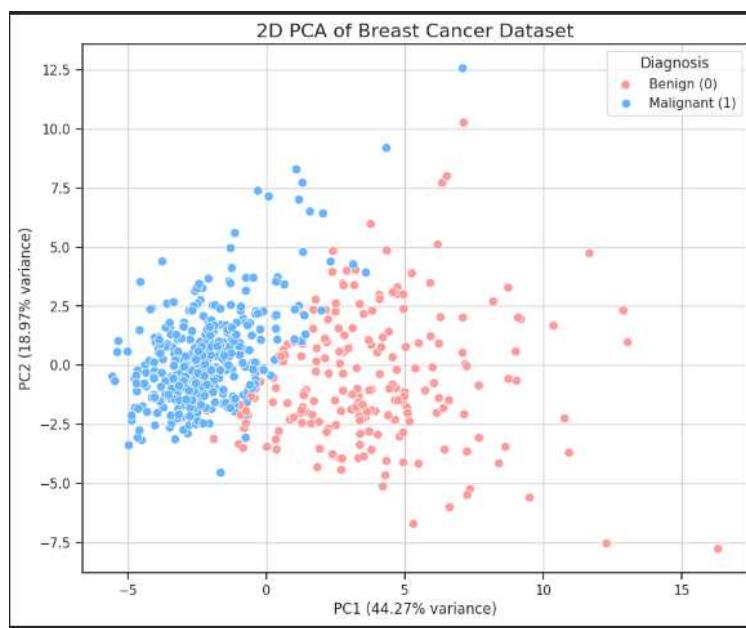
## PCA ۲.۲

در ادامه یک PCA دو بعدی ترتیب می‌دهیم.

```
۱ from sklearn.decomposition import PCA
  ۲ import matplotlib.pyplot as plt
  ۳
  ۴ pca = PCA(n_components=2)
  ۵ X_pca = pca.fit_transform(X_scaled)
  ۶
  ۷
  ۸ pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
  ۹ pca_df['target'] = y_encoded
  ۱۰
  ۱۱ plt.figure(figsize=(10, 8))
  ۱۲ sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='target', palette=['#66b3ff', '#ff9999'],
  ۱۳ s=60)
  ۱۴ plt.title('2D PCA of Breast Cancer Dataset', fontsize=16)
  ۱۵ plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]*100:.2f}% variance)')
  ۱۶ plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]*100:.2f}% variance)')
  ۱۷ plt.legend(title='Diagnosis', labels=['Benign (0)', 'Malignant (1)'])
  ۱۸ plt.grid(True)
  ۱۹ plt.show()
```

2D PCA :v Code

توزیع دیتای مربوط به هر کلاس در فضای دو بعدی به صورت زیر خواهد بود:



شکل ۳: 2D PCA

## ۳.۲ تقسیم دیتا به آموزش، اعتبارسنجی و تست

در کل زیر این تقسیم صورت گرفته است. همانطور که صورت سوال گفته تقسیم‌بندی به صورت ۶۰/۲۰/۲۰ صورت گرفته است و مقدار random state=53 در نظر گرفته شده است چرا که دو رقم آخر شماره دانشجویی یکی از اعضای گروه برابر با ۵۳ است.

```

1  from sklearn.model_selection import train_test_split
2
3  X_train, X_temp, y_train, y_temp = train_test_split(
4      X_scaled, y_encoded, test_size=0.4, random_state=53, stratify=y_encoded
5  )
6
7
8  X_val, X_test, y_val, y_test = train_test_split(
9      X_temp, y_temp, test_size=0.5, random_state=53, stratify=y_temp
10 )
11
12
13 print("Train shape:", X_train.shape, y_train.shape)
14 print("Validation shape:", X_val.shape, y_val.shape)
15 print("Test shape:", X_test.shape, y_test.shape)

```

2D PCA :Λ Code

خروجی کد بالا به صورت زیر است:



نمونه‌ها	ویژگی‌ها	مجموعه داده
۳۴۱	۳۰	Train
۱۱۴	۳۰	Validation
۱۱۴	۳۰	Test

جدول ۲: ابعاد مجموعه‌های داده

## ۴.۲ آموزش و گزارش حاشیه

همانطور که سوال از ما خواسته است روی فضای اصلی مدل را آموزش می‌دهیم. از تابع `svc` و `kernel` خطی استفاده می‌نماییم. کد زیر به منظور آموزش مدل زده شده است:

```

1 from sklearn.svm import SVC
2 from sklearn.metrics import accuracy_score, precision_score, recall_score,
roc_auc_score
3
4
5 svc = SVC(kernel='linear', probability=True, random_state=53)
6 svc.fit(X_train, y_train)
```

### ۹ آموزش مدل

در بالا نیز مقدار `random state` برابر با 53 قرار داده شده است. پس از آموزش مدل مقدار حاشیه را به دست می‌آوریم. البته باید توجه داشت که این مقدار برابر با نصف مقدار حاشیه است یعنی فاصله نیمی از بردارهای پشتیبان از خط وسط. اگر بخواهیم فاصله بین بردارهای پشتیبان را به دست بیاوریم، باید این عدد در دو ضرب شود.

```

1 import numpy as np
2 w = svc.coef_[0]
3
4
5
6 margin = 1 / np.linalg.norm(w)
7 print("Margin (1 / ||w||):", margin)
```

### ۱۰ محاسبه وزن‌های مدل و حاشیه

خروجی کد بالا به صورت زیر است:

$$\text{Margin} = \frac{1}{\|w\|} \approx 0.3361$$

حال بر روی مجموعه داده اعتبارسنجی و تست شاخص‌های Accuracy، Precision، Recall، F1 و ROC-AUC را اندازه‌گیری می‌نماییم. برای اینکه این شاخص‌ها را برای هر کلاس نیز به صورت دقیق داشته باشیم از دستور `classification Report` استفاده می‌نماییم. مقادیر ROC-AUC را نیز جدا محاسبه می‌نماییم چرا که این دستور این معیار ارزیابی را محاسبه نمی‌نماید. کدهای زده شده در این بخش به صورت زیر است:

```
1 y_val_pred = svc.predict(X_val)
```



```
 1 y_test_pred = svc.predict(X_test)
 2
 3
 4
 5 y_val_prob = svc.predict_proba(X_val)[:, 1]
 6 y_test_prob = svc.predict_proba(X_test)[:, 1]
```

### ۱۱: محاسبه خروجی‌ها

```
 1 from sklearn.metrics import classification_report, roc_auc_score
 2
 3 def evaluate(y_true, y_pred, y_prob):
 4     report = classification_report(y_true, y_pred, output_dict=True)
 5     roc_auc = roc_auc_score(y_true, y_prob)
 6
 7
 8     return report, roc_auc
```

### ۱۲: تابع ارزیابی

```
 1 report_val, roc_val = evaluate(y_val, y_val_pred, y_val_prob)
 2 report_test, roc_test = evaluate(y_test, y_test_pred, y_test_prob)
 3
 4 print("Validation Classification Report:")
 5 print(classification_report(y_val, y_val_pred))
 6
 7
 8 print("Validation ROC-AUC:", roc_val)
 9
10 print("\nTest Classification Report:")
11 print(classification_report(y_test, y_test_pred))
12
13 print("Test ROC-AUC:", roc_test)
```

### ۱۳: محاسبه شاخص‌های ارزیابی



```

Validation Classification Report:
precision    recall   f1-score   support
          0       0.97      1.00      0.99      72
          1       1.00      0.95      0.98      42

accuracy                           0.98      114
macro avg       0.99      0.98      0.98      114
weighted avg    0.98      0.98      0.98      114

Validation ROC-AUC: 0.9983465608465609

Test Classification Report:
precision    recall   f1-score   support
          0       0.95      1.00      0.97      71
          1       1.00      0.91      0.95      43

accuracy                           0.96      114
macro avg       0.97      0.95      0.96      114
weighted avg    0.97      0.96      0.96      114

Test ROC-AUC: 0.987880773010154

```

شکل ۴: نتایج ارزیابی بر روی مجموعه اعتبارسنجی و تست

همانطور که مشاهده می‌شود نتایج بسیار خوبی بر روی مجموعه اعتبارسنجی و تست به دست آمده است و ما توانستیم به شکل بسیار خوبی این دو کلاس را از یکدیگر تفکیک نماییم.

## ۵.۲ مطالعه C

در این بخش به ورودی تابع SVC پارامتر C را نیز می‌دهیم. به ازای مقادیر مختلف C که در صورت سوال آمده است، مدل را آموزش می‌دهیم و شاخص‌های ارزیابی را بر روی مجموعه اعتبارسنجی و تست به دست می‌آوریم. همچنین مقدار حاشیه و تعداد بردارهای پشتیبان را به ازای مقادیر مختلف C محاسبه می‌نماییم و همه را در یک جدول ارائه می‌دهیم. کد زیر در این بخش زده شده است:

```

C_values = [0.01, 0.1, 1, 10, 100]
#
results = []
for C in C_values:
    svc = SVC(kernel='linear', C=C, probability=True, random_state=53)
    svc.fit(X_train, y_train)
    #
    w = svc.coef_[0]
    margin = 1 / np.linalg.norm(w)
    #
    y_val_pred = svc.predict(X_val)
    y_test_pred = svc.predict(X_test)
    #

```



```

14 y_val_prob = svc.predict_proba(X_val)[:, 1]
15 y_test_prob = svc.predict_proba(X_test)[:, 1]
16
17 val_metrics = evaluate(y_val, y_val_pred, y_val_prob)
18 test_metrics = evaluate(y_test, y_test_pred, y_test_prob)
19
20 sv_count = svc.n_support_.sum()
21
22 results.append({
23     "C": C,
24     "Margin (1/||w||)": margin,
25
26     "Val Accuracy": val_metrics["Accuracy"],
27     "Val Precision": val_metrics["Precision_macro"],
28     "Val Recall": val_metrics["Recall_macro"],
29     "Val F1": val_metrics["F1_macro"],
30     "Val ROC-AUC": val_metrics["ROC-AUC"],
31
32     "Test Accuracy": test_metrics["Accuracy"],
33     "Test Precision": test_metrics["Precision_macro"],
34     "Test Recall": test_metrics["Recall_macro"],
35     "Test F1": test_metrics["F1_macro"],
36     "Test ROC-AUC": test_metrics["ROC-AUC"],
37
38     "Support Vectors": sv_count
39 })
40 df_results = pd.DataFrame(results)
df_results

```

#### ۱۴: آموزش مدل از طریق مقادیر مختلف C و گرفتن جدول خواسته شده

تابع ارزیابی نیز به شکل زیر برای این بخش تغییر کرده است چرا که برای یکسری شاخص‌ها که برای هر کلاس محاسبه می‌شود مقدار macro avg را محاسبه نمودیم و در داخل جدول آوردهیم.

```

1 from sklearn.metrics import classification_report, roc_auc_score, accuracy_score
2
3 def evaluate(y_true, y_pred, y_prob):
4     report = classification_report(y_true, y_pred, output_dict=True)
5
6     metrics = {
7         "Accuracy": accuracy_score(y_true, y_pred),
8         "Precision_macro": report["macro avg"]["precision"],
9         "Recall_macro": report["macro avg"]["recall"],
10        "F1_macro": report["macro avg"]["f1-score"],
11        "ROC-AUC": roc_auc_score(y_true, y_prob)
12    }

```



۱۳ return metrics

۱۵: تابع ارزیابی برای مطالعه پارامتر  $C$ ۶.۲ نتایج آزمایش SVM با هسته خطی برای مقادیر مختلف  $C$ 

در این بخش تأثیر مقادیر مختلف  $C$  در طبقه‌بندی SVM خطی بررسی شده است. تمام معیارهای ارزیابی در مجموعه اعتبارسنجی و تست محاسبه شده‌اند و همچنین مقدار margin و تعداد بردارهای پشتیبان نیز برای تحلیل بهتر رفتار مدل اندازه‌گیری شده است.

## ۶.۲.۱ مرور روابط نظری SVM

مسئله SVM نرم (Soft-Margin) در فرم primal به صورت زیر تعریف می‌شود:

$$\min_{w,b,\{\xi_i\}} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad \text{s.t. } y_i(w^\top x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

پارامتر تنظیم‌پذیر  $C$  نقش کلیدی دارد:

•  $C$  کوچک: تأکید بر بزرگ بودن margin  $\Rightarrow$  مدل ساده‌تر، بایاس بالا، واریانس کم.

•  $C$  بزرگ: تأکید بر کوچک بودن خطاهای آموزشی  $\Rightarrow$  مدل پیچیده‌تر، بایاس پایین، واریانس بالا.

برای SVM خطی، مقدار margin برابر است با:

$$\text{margin} = \frac{1}{\|w\|}.$$

## ۶.۲.۲ تحلیل تجربی

۱) رفتار margin با افزایش  $C$ 

مقدار margin با افزایش  $C$  به طور پیوسته کاهش یافته است. این روند نشان‌دهنده انتقال مدل از یک مرز تصمیم نرم و منعطف (برای کوچک) به مرز تصمیم سخت و فیت‌شده روی داده‌ها (برای  $C$  بزرگ) است.

۲) تغییر تعداد بردارهای پشتیبان

با افزایش  $C$ ، تعداد بردارهای پشتیبان کاهش یافته است. این امر مطابق تئوری است: در  $C$  کوچک، نمونه‌های بیشتری اجازه خطا دارند و وارد حاشیه می‌شوند، اما در  $C$  بزرگ مدل سخت‌گیرتر می‌شود و تنها نقاط نزدیک به مرز تصمیم به صورت SV باقی می‌مانند.

۳) تحلیل بایاس-واریانس (به همراه محاسبات)

برای تحلیل تجربی بایاس-واریانس، اختلاف بین عملکرد اعتبارسنجی و تست بررسی شده است. اختلاف کم نشان‌دهنده واریانس پایین و اختلاف زیاد معمولاً نشان‌دهنده واریانس بالاتر است.

برای هر مقدار  $C$  داریم:

$$\Delta(C) = \text{Accuracy Val}(C) - \text{Accuracy Test}(C).$$



با توجه به مقادیر به دست آمده:

$$\Delta(0.01) = 0.9825 - 0.9649 = 0.0176,$$

$$\Delta(0.1) = 0.9912 - 0.9649 = 0.0263,$$

$$\Delta(1) = 0.9737 - 0.9649 = 0.0088,$$

$$\Delta(10) = 0.9561 - 0.9649 = -0.0088,$$

$$\Delta(100) = 0.9561 - 0.9561 = 0.$$

تفسیر:

- بزرگترین اختلاف در  $C = 0.1$  مشاهده می‌شود؛ هرچند مدل بهترین عملکرد را دارد اما نسبت به مقدار  $C = 1$  واریانس بیشتری نشان می‌دهد.
- برای  $C = 1$ ، اختلاف بسیار کوچک است و نشان دهنده تعادل مناسب بایاس-واریانس می‌باشد.
- در  $C = 10$  و  $C = 100$ ، عملکرد ولیدیشن از تست کمتر شده که معمولاً نشانه افزایش خطای بایاس است. این مقادیر نشان دهنده مدل ساده‌تری هستند که نسبت به داده‌های آموزشی کمتر واکنش نشان می‌دهند (underfitting جزئی).

#### ۴) جمع‌بندی تحلیل بایاس-واریانس

- مقادیر کوچک  $C$  (مثل 0.01): margin بزرگ، SV زیاد، مدل ساده  $\Rightarrow$  بایاس بالا، واریانس پایین.
- مقادیر متوسط  $C$  (مثل 0.1): بهترین عملکرد کلی اما اختلاف ولیدیشن-تست کمی بیشتر است  $\Rightarrow$  واریانس نسبتاً بیشتر.
- مقادیر بزرگ  $C$  (۱۰ و ۱۰۰): بسیار کوچک، SV اندک اما کاهش دقت ولیدیشن و تست  $\Rightarrow$  بایاس بالا، مدل بیش از حد سخت و محدود.

#### ۳.۶.۲ جمع‌بندی نهایی

- رفتار کاهش margin و کاهش تعداد بردارهای پشتیبان با افزایش  $C$  کاملاً مطابق تئوری SVM است.
- مقدار  $C = 0.1$  از نظر عملکرد، بهترین حالت مشاهده شده است.
- مقدار  $C = 1$  از نظر تعادل بایاس-واریانس بهترین حالت است.

جدول این بخش به صورت زیر است:



C	Margin ( $1/\ w\ $ )	Val Accuracy	Val Precision	Val Recall	Val F1	Val ROC-AUC	Test Accuracy	Test Precision	Test Recall	Test F1	Test ROC-AUC	Support Vectors
0	0.0100	1.5067	0.9825	0.9865	0.9762	0.9810	0.9983	0.9649	0.9733	0.9535	0.9619	0.9879
1	0.1000	0.6668	0.9912	0.9932	0.9881	0.9905	0.9980	0.9649	0.9673	0.9581	0.9623	0.9879
2	1.0000	0.3361	0.9737	0.9741	0.9692	0.9716	0.9960	0.9649	0.9673	0.9581	0.9623	0.9807
3	10.0000	0.1428	0.9561	0.9605	0.9454	0.9521	0.9940	0.9649	0.9627	0.9627	0.9627	0.9872
4	100.0000	0.1006	0.9561	0.9551	0.9504	0.9526	0.9940	0.9561	0.9516	0.9556	0.9535	0.9866

شکل ۵: جدول نتایج برای مطالعه پارامتر C

## ۷.۲ نمودارهای ارزیابی

همانطور که در بخش قبل دیدیم بهترین پارامتری که می‌توان برای C انتخاب نمود، برابر با 0.1 می‌باشد. چراکه نتایج ارزیابی روی مجموعه اعتبارسنجی و تست برای این مقدار از C بالاتر است.

برای این مقدار مدل را مجددآموزش می‌دهیم و همانطور که سوال از ما خواسته منحنی‌های ROC و Precision-Recall را ترسیم می‌نماییم.

```

1 best_C = 0.1
2 svc_best = SVC(kernel='linear', C=best_C, probability=True, random_state=53)
3 svc_best.fit(X_train, y_train)
4 from sklearn.metrics import roc_curve, auc, precision_recall_curve, average_precision_score
5 fpr, tpr, thresholds_roc = roc_curve(y_test, y_test_prob)
6 roc_auc = auc(fpr, tpr)

7
8
9 precision, recall, thresholds_pr = precision_recall_curve(y_test, y_test_prob)
10 avg_precision = average_precision_score(y_test, y_test_prob)
11 plt.figure(figsize=(12,5))

12
13 plt.subplot(1, 2, 1)
14 plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
15 plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
16 plt.xlabel('False Positive Rate')
17 plt.ylabel('True Positive Rate')
18 plt.title('ROC Curve')
19 plt.grid(True)
20 plt.legend(loc="lower right")

21
22
23 plt.subplot(1, 2, 2)
24 plt.plot(recall, precision, color='green', lw=2, label=f'Precision-Recall curve (AP = {
25 avg_precision:.2f})')
26 plt.xlabel('Recall')
27 plt.ylabel('Precision')
28 plt.title('Precision-Recall Curve')
29 plt.grid(True)

```



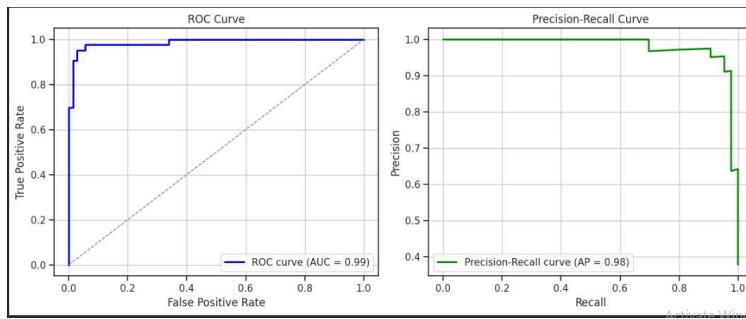
```

۲۹ plt.legend(loc="lower left")
۳۰
۳۱ plt.tight_layout()
۳۲ plt.show()

```

۱۶: آموزش مدل با بهترین مقدار پارامتر  $C$  و به دست آوردن منحنی‌های ROC و Precision-Recall

خروجی این کد به صورت زیر است:



شکل ۶: منحنی‌های ROC و Precision-Recall

## ۸.۲ تفسیر نمودارهای ROC و Precision-Recall برای مدل SVM با پارامتر $C = 0.1$

در این بخش، عملکرد مدل SVM با هسته خطی و مقدار تنظیم‌گر  $C = 0.1$  با استفاده از دو نمودار استاندارد یعنی Curve ROC و Curve Precision-Recall تحلیل می‌شود. این مقدار از  $C$  پیش‌تر به عنوان یکی از بهترین انتخاب‌ها از نظر توازن میان بایاس و واریانس شناسایی شده بود، و نمودارهای بالا کیفیت این انتخاب را تأیید می‌کنند.

### ۱. تفسیر منحنی ROC

نمودار ROC نرخ مثبت درست (TPR) را در برابر نرخ مثبت کاذب (FPR) نشان می‌دهد. چند مشاهده مهم از نمودار:

- تقریباً تمام نقاط منحنی به ناحیه بالا-چپ نزدیک شده‌اند؛ این یعنی مدل توانسته است با خطای بسیار کم، نمونه‌های مثبت را درست تشخیص دهد.
- مقدار مساحت زیر منحنی (AUC) برابر با **0.99** گزارش شده است که نشان‌دهنده عملکرد تقریباً ایده‌آل است.
- شبیه تند در بخش آغازین منحنی (FPR) نزدیک به صفر) بیانگر حساسیت بالا بدون افزایش قابل توجه در نرخ خطای مثبت کاذب است.

به طور خلاصه، منحنی ROC تأیید می‌کند که مدل قدرت تفکیک بسیار بالایی بین دو کلاس دارد و تصمیم‌مرز آن به خوبی نمونه‌های مثبت و منفی را جدا می‌کند.



## ۲. تفسیر منحنی Precision-Recall

از آنجا که در این مسئله توزیع نمونه‌ها می‌تواند نامتوازن باشد یا اهمیت کلاس مثبت بیشتر باشد، بررسی نمودار PR نیز ضروری است. مشاهده‌های کلیدی عبارت‌اند از:

- مقدار  $AP = 0.98$  به دست آمده که نشان‌دهنده دقت تجمعی بسیار بالا در بازیابی نمونه‌های مثبت است.
- بخش ابتدایی منحنی، تقریباً در مقدار  $Precision = 1.0$  قرار دارد؛ به این معنی که مدل در بسیاری از آستانه‌ها بدون هیچ خطای مثبت کاذب نمونه‌های مثبت را تشخیص داده است.
- کاهش بسیار ملایم در هنگام افزایش Recall نشان می‌دهد که مدل مجبور نشده برای بازیابی بیشتر نمونه‌ها دقت را قربانی کند؛ که این رفتار عموماً فقط در مدل‌های با تفکیک‌پذیری بالا دیده می‌شود.

در مجموع، منحنی PR نشان می‌دهد که مدل حتی در سناریوهای نامتوازن نیز عملکرد بسیار مطلوبی دارد و خطای نوع اول (False Negative) در آن به خوبی کنترل شده است. نوع دوم (Positive) در آن به خوبی کنترل شده است.

## جمع‌بندی

ترکیب دو شاخص  $ROC-AUC = 0.99$  و  $AP = 0.98$  نشان می‌دهد که مدل SVM با مقدار  $C = 0.1$ :

- از نظر تفکیک کلاس‌ها بسیار قوی است،
- همزمان Precision و Recall بالایی دارد،
- در تمام آستانه‌ها عملکرد پایدار و قابل اعتماد ارائه می‌دهد،
- و انتخاب  $C = 0.1$  برای این مجموعه داده یک trade-off bias-variance مناسب ایجاد کرده است.

بنابراین، ارزیابی‌های مبتنی بر ROC و PR به طور کامل نشان می‌دهند که مدل در حالت  $C = 0.1$  بهینه‌ترین عملکرد را نسبت به سایر مقادیر آزمایش شده ارائه داده است.