HW2 Solutions

Machine learning | spring 1400 | Dr.abdi

Teacher Assistant:

Zahra Dehghani

Student name : **Amin Fathi**

Student id : **400722102**

فهرست

شرح پروژه و معرفی بردار پشتیبان (svm) ۳

توضیح کرنل ها ۳

کرنل linear ۳

کرنل sigmoid ۴

کرنل rbf ۵

کرنل Polynomial ۸

بخش اول ۸

Example 1 ..... ۹

Example 2 ..... ۱۲

Example 3 ..... ۱۶

Example 4 ..... ۱۹

چالش ها و نتایج بخش اول ۲۲

بخش دوم ۲۳

چالش ها و نتایج بخش دوم ۲۷

بخش سوم ۲۸

چالش ها و نتایج بخش سوم ۳۰

منابع ۳۱

شرح پروژه و معرفی بردار پشتیبان (svm) :

ماشین بردار پشتیبان یکی از روش های یادگیری با نظارت است که از آن برای طبقه بندی و رگرسیون استفاده می شود ؛ هدف از الگورتیم ماشین بردار پشتیبان یافتن یک ابر صفحه (hyperplane) n بعدی (n تعداد ویژگی ها است) که نقاط داده را طبقه بندی کند ؛ نقاط داده ای که در سمت های مختلف ابر صفحه قرار دارند به عنوان کلاس های مختلف طبقه بندی می‌شوند؛ همانطور که اشاره شد ابعاد ابر صفحه را با استفاده از تعداد ویژگی ها (features) تعیین می‌کنیم، به طور مثال اگر تعداد ویژگی‌ها برابر ۲ باشد در نتیجه ابر صفحه در واقع یک خط است و اگر تعداد ویژگی‌ها ۳ باشد در واقع ابر صفحه یک صفحه دو بعدی (plane) در فضای ویژگی های سه بعدی است .

در انتخاب مرز بین دو دسته نقاط داده می‌توان hyperplane یا ابر صفحه‌های متفاوتی را انتخاب کرد اما هدف ما در svm یافتن بهینه‌ترین آن هاست که بیشترین حاشیه (margin) را از داده‌های هر دو کلاس‌ها داشته باشد ، بالا بودن margin اطمینان بیشتری را برای طبقه بندی داده‌های به ما می‌دهد.

در این تمرین از ابزارها و کتابخانه‌های آماده SVM برای اشنایی با قابلیت‌های دسته‌بندی SVM استفاده می‌کنیم. در بخش اول، چند مسئله دو کلاسه تعریف کرده و با استفاده از الگوریتم SVM با هسته‌ها (Kernel) و پارامترهای مختلف، آن‌ها را دسته‌بندی می‌کنیم. در حین دسته‌بندی، از ما خواسته شده است که علاوه‌بر خط جداکننده، Marginها را نیز رسم کنیم. در بخش دوم نیز به دسته‌بندی پایگاه داده‌ی MNIST می‌پردازیم. پارامترها و kernelهای مختلف را در این جا نیز به کار می‌بریم. در بخش آخر نیز، از پایگاه داده مربوط به ۵ کاراکتر از کاراکترهای موجود در پلاک خودرو ایران استفاده می‌کنیم. این تصاویر از دوربین‌های واقعی نصب شده برای تشخیص پلاک خودرو استخراج شده‌اند. کیفیت پایین برخی از تصاویر در اثر حرکت خودرو، کثیفی شیشه یا لنز دوربین، مخدوش بودن پلاک و سایر عوامل محیطی است. از ما خواسته شده تا با استفاده از الگوریتم SVM، به تشخیص و دسته‌بندی کاراکترهای ۲، ۳، ۷، س و ص بپردازیم.

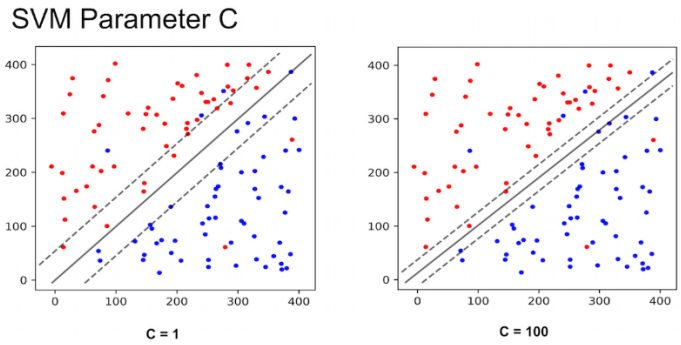
توضیح کرنل ها :

کرنل linear :

این کرنل ساده ترین کرنل در svm است که بر اساس ضرب داخلی ابعاد نقاط به علاوه یک مقدار ثابت c نقاط را طبقه بندی میکند ، به طور واضح تر این کرنل مشخصات یک هایپر پلین را به دست می آورد که طرفین آن هایپر پلین در کلاس های متفاوت طبقه بندی می شودند

$$k(x, y) = x^T y + c$$

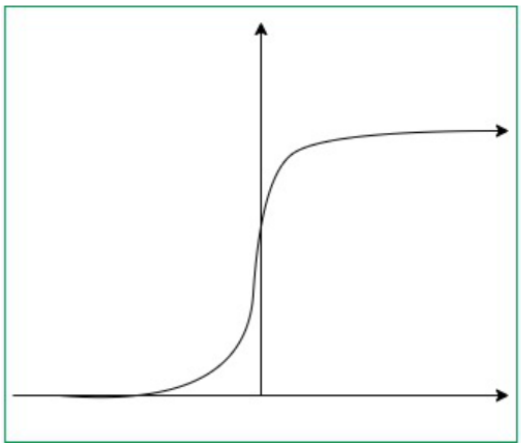
پارامتر C در این کرنل نقش شیر تعادل بین طبقه بندی درست و افزایش margin را دارد ، به طوری که هر چه مقدار آن بزرگتر انتخاب شود مدل به سمت طبقه بندی درست تر می رود و هر چه مقدار آن کمتر باشد به سمت افزایش دامنه margin سوق پیدا میکند



کرنل sigmoid :

این تابع معادل یک مدل پرسپترون دو لایه شبکه عصبی است که به عنوان یک تابع فعال سازی برای نورون های مصنوعی استفاده می شود. ،

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



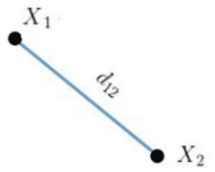
Sigmoid Kernel Graph

کرنل rbf :

این کرنل بر اساس فاصله بین نقاط (دو نقطه X1 و X2 را در نظر بگیرید) آن ها را کلاس بندی می کند و فرمول آن به شکل زیر است .

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

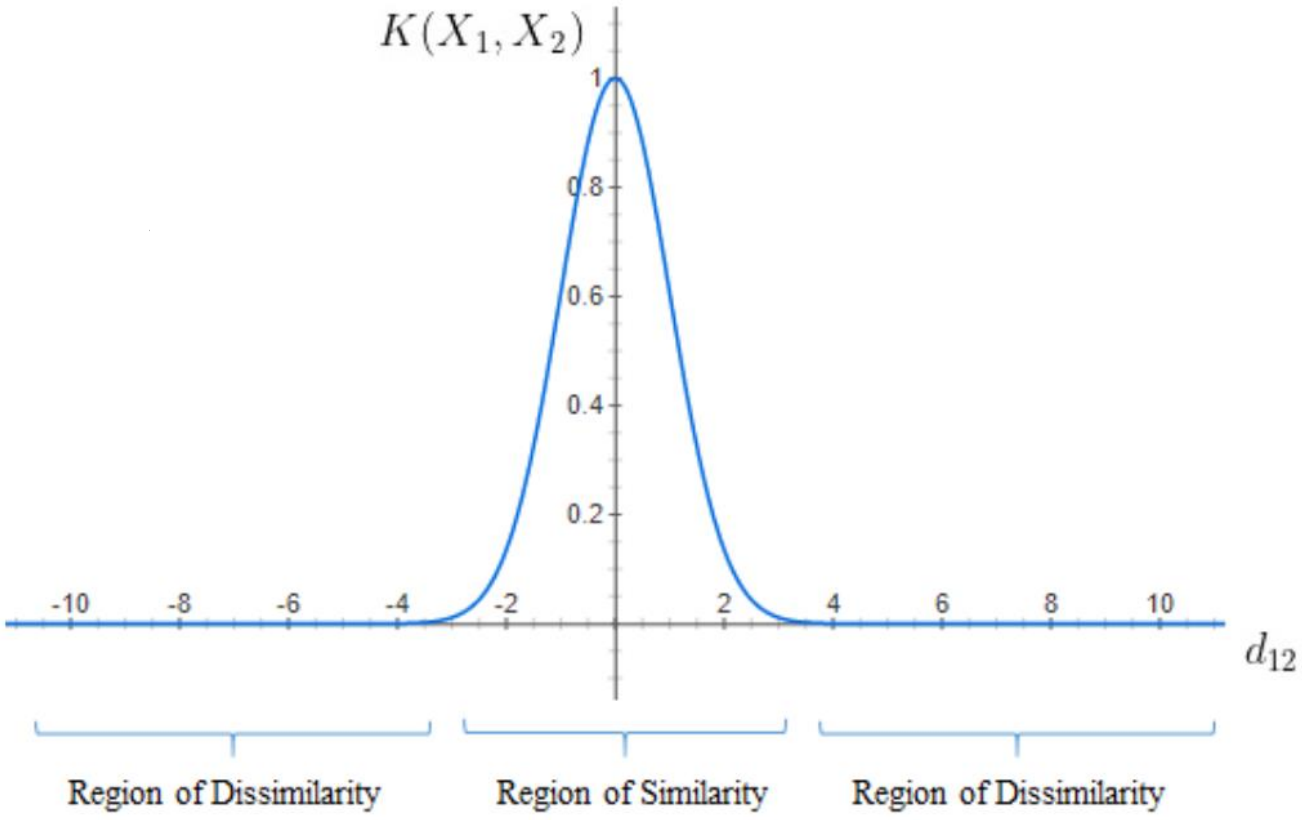
در فرمول بالا صورت کسر فاصله اقلیدسی بین نقاط بوده و مخرج واریانس است



$$d_{12} = ||X_1 - X_2||^2$$

$$K(X_1, X_2) = exp(-\frac{d_{12}}{2\sigma^2})$$

طبق فرمول بالا ماکسیمم مقداری که کرنل rbf برمیگرداند مربوط به نقاطی است که فاصله ان ها از همدیگر 0 می باشد و درنتیجه حاصل کرنل برابر 1 است و مینیمم این مقدار مربوط به نقاطی است که فاصله آن ها از همدیگر زیاد بوده و درنتیجه خروجی کرنل برابر 0 است ، نمودار زیر به خوبی این قضیه را نمایش می دهد (مقدار واریانس در شکل زیر برابر 1 است)



البته هر چه مقدار واریانس بزرگتر در نظر گرفته شود ، کرنل rbf محدوده وسیع تری را به عنوان نقاط مشابه با نقاط مد نظر ما در نظر میگیرد ،به طور مثال شکل زیر را در نظر بگیرید :

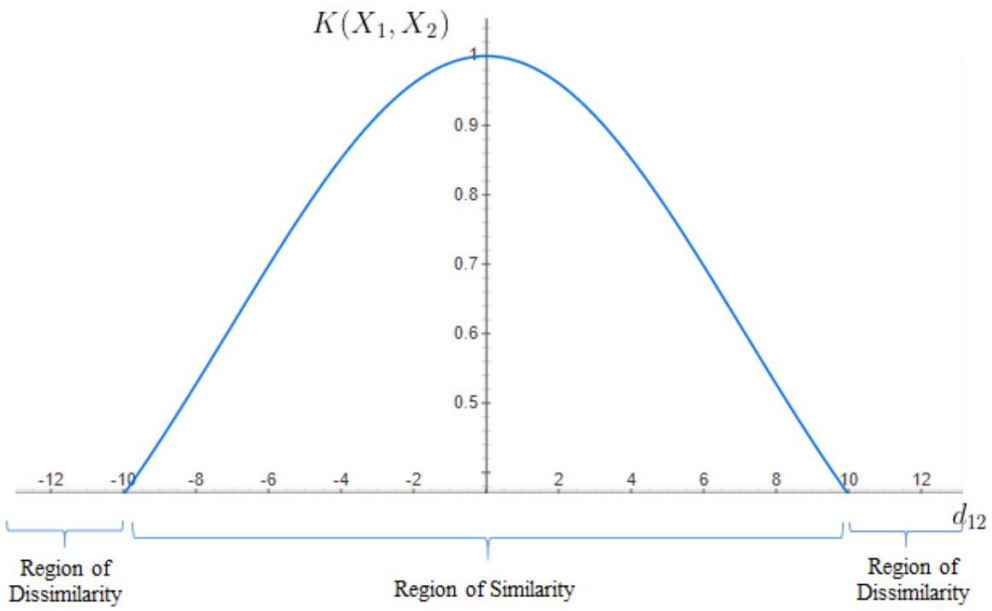
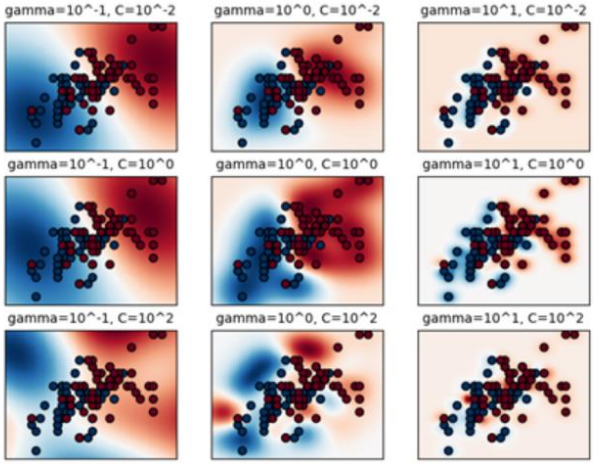


Fig 5: RBF Kernel for σ = 10 [Image by Author]

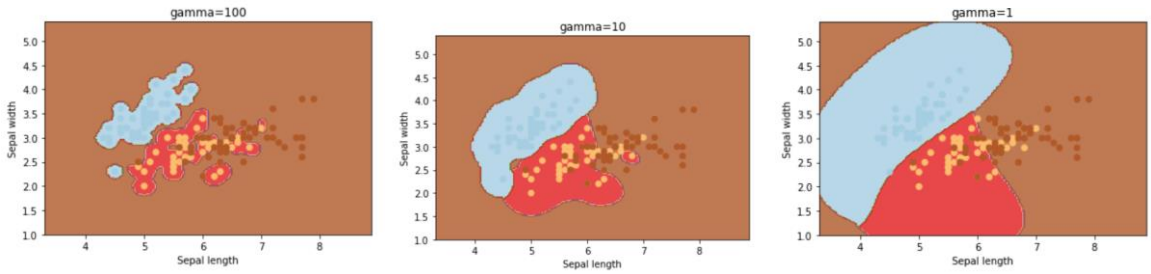
در نحوه پیاده سازی در کتابخانه sklearn نیز باید دقت داشت که مقدار گاما که در کد پیاده سازی باید تنظیم شود با مقدار واریانس نسبت عکس دارد

$$\gamma \propto \frac{1}{\sigma}$$

همانطور که در بالا دیدیم ، مقدار واریانس کمتر ، طبقه بندی سخت گیرانه تر را در پی دارد و طبق فرمول بالا این بدان معناست که هر چه مقدار گاما را بیشتر در نظر بگیریم ، طبقه بندی سخت گیرانه تری را تجربه خواهیم کرد که این ممکن است به اور فیت شدن مدل بیانجامد ، از طرف دیگر کم بودن بیش از حد مقدار گاما می تواند منجر به این شود که مدل مد نظر به خوبی الگوریتم دادگان را تشخیص ندهد و بسیار محدود باشد و از پس پیچیدگی دادگان بر نیاید و چه بسا منطقه تاثیر هر بردار پشتیبان کل مجموعه دادگان آموزشی باشد . پارامتر C هم نقش شیر تعادل بین طبقه بندی درست و افزایش margin را دارد ، به طوری که هر چه مقدار آن بزرگتر انتخاب شود مدل به سمت طبقه بندی درست تر می رود و هر چه مقدار آن کمتر باشد به سمت افزایش دامنه margin سوق پیدا میکند



تاثیر گاما و C بر کرنل rbf



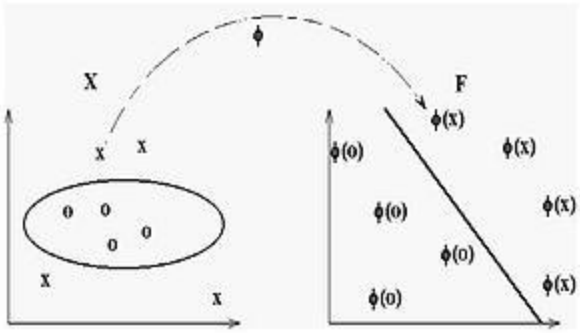
شمایی دیگر از نحوه تاثیر مقدار گاما در طبقه بندی ها

کرنل Polynomial :

یکی از کرنل ها مورد استفاده در svm است ، فرمول آن به شکل زیر است

$$K(x, y) = (x^\top y + c)^d \qquad K(x, y) = \langle \varphi(x), \varphi(y) \rangle$$

در واقع این کرنل ضرب داخلی x,y (فضای ویژگی ها) را به توان d رسانده این عمل پیچیدگی را بالا برده و برای داده های با پیچیدگی بالاتر بهتر است ، در واقع یک نگاشت فضایی رخ میدهد ، به طور مثال به شکل زیر نگاه کنید ، بردار svm در فضای ورودی یک بیضی است اما در واقع svm به کمک یک polynomial با درجه دو و با تنظیم پارامتر c یک بیضی را پیاده کرده که با نگاشت به فضای خطی توانسته طبقه بندی کند



بخش اول:

ابتدا تمامی کتابخانه‌های مورد نیاز را import می‌کنیم. سپس به ایجاد یک مجموعه داده‌ی دودویی می‌پردازیم. در این تابع یک DataFrame با سه ستون x، y و target ساخته شده است. همان‌طور که می‌دانیم، x و y نشان‌دهنده‌ی مختصات هر نقطه و target نشان‌دهنده‌ی برچسب نقطه است که در صورت سوال بیان شده است که برچسب‌ها یا ۱ و یا ۱- هستند.

```
def generate_binary_dataset(min_value, max_value, size, positive_condition):
    data = pd.DataFrame(np.concatenate((np.random.uniform(min_value, max_value, (size, 2)),-np.ones((size, 1))), axis=1),  columns=['x', 'y', 'target'])
    data.loc[positive_condition(data.x, data.y), 'target'] = 1
    return data
```

در صورت سوال از ما خواسته شده بود که علاوه بر خط جداکننده، margin را نیز در هنگام دسته‌بندی داده‌ها رسم کنیم. تابع plot_separator نیز در این راستا عمل می‌کند. خط جداکننده و دو margin اطراف آن را با خط‌چین رسم می‌کند.

```python
def plot_separator(svc):
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = svc.decision_function(xy).reshape(XX.shape)
    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
    ax.scatter(svc.support_vectors_[:, 0], svc.support_vectors_[:, 1], s=100, linewidth=1, label="support vectors", facecolors='none', edgecolors='k')
```

در تابع زیر، مسئله دودویی رسم می‌شود. به‌عبارتی می‌توان گفت که خطوط دو کلاس ۱ و ۱- با دو رنگ متقاوت در یک صفحه رسم می‌شوند. این خطوط به صورت تصادفی و با اعمال توابع خطی و غیرخطی بر روی آن‌ها ایجاد شده‌اند که در ادامه به توضیح کامل‌تر آن خواهیم پرداخت. حال یک شرط نیز بیان شده است که اگر از الگوریتم SVM استفاده شده باشد، تابع plot_separator فراخونی می‌شود تا علاوه بر خود داده‌ها، خط جداکننده و margin نیز رسم شود.

```python
def plot_binary(data, svc=None):
    plt.figure(figsize=(15, 15))
    plt.scatter(data.x, data.y, c=data.target, label="data", s=20, cmap=plt.cm.seismic)

    if svc:
        plot_separator(svc)

    plt.grid()
    plt.legend()
```

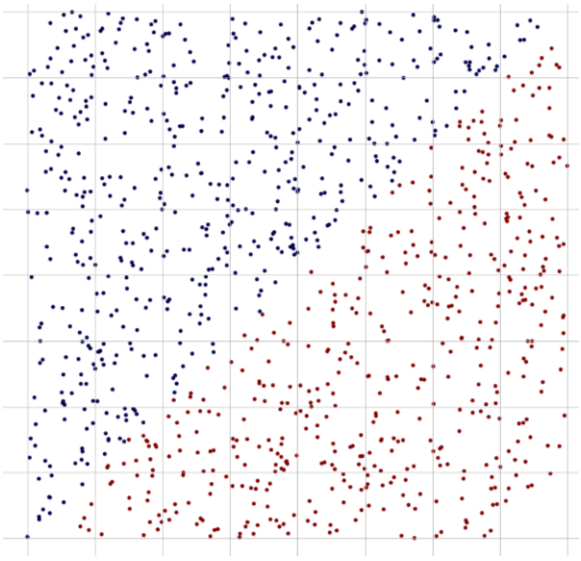
پیش از ادامه دادن بحث لازم به ذکر است در سه تابع گذشته از ایده لینک شماره ۱ استفاده شده است .

Example 1

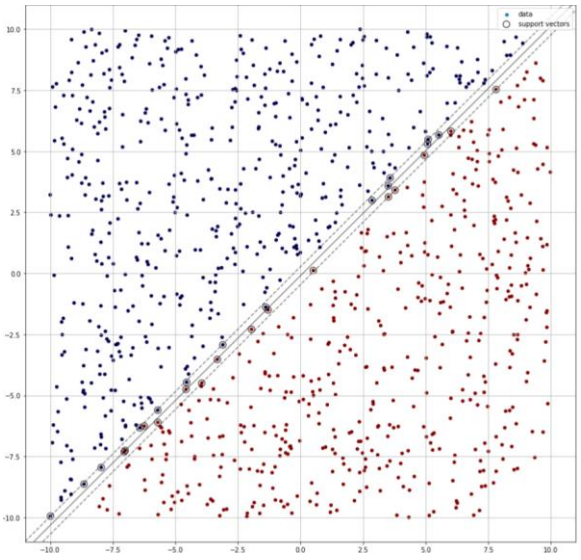
در ادامه به سراغ ساختن مجموعه داده‌ها و اجرای الگوریتم SVM بر روی آن‌ها می‌رویم. ساده‌ترین حالت، مجموعه داده خطی است. به این صورت که ۱۰۰۰ نقطه به صورت تصادفی در بازه ۱۰- تا ۱۰ تولید می‌شوند و الگوریتم SVM با کرنل‌ها و پارامترهای مختلف بر روی آن‌ها اعمال می‌شود. در ادامه ابتدا DataFrame مجموعه داده ایجاد شده و نمودار آن را نشان می‌دهیم و پس از آن نیز به سراغ نتایج حاصل از پیاده‌سازی الگوریتم SVM با کرنل‌ها و پارامترهای مختلف می‌رویم.

```python
dataset = generate_binary_dataset(-10, 10, 1000, lambda x, y: x  > y)
dataset.head()
```

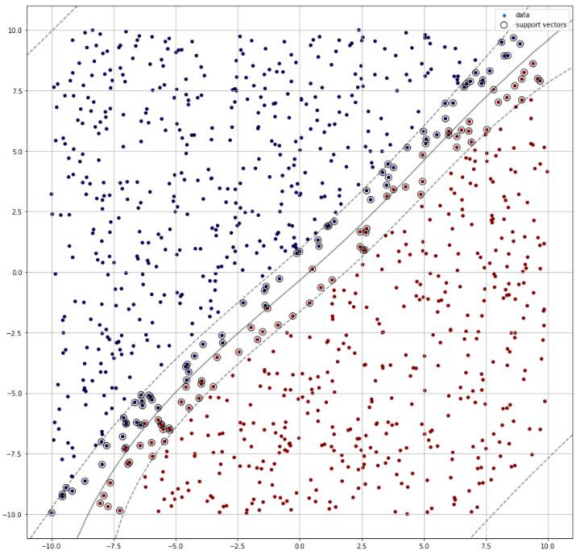
	x	y	target
0	-6.325697	-0.095887	-1.0
1	-5.451583	-0.950370	-1.0
2	6.269285	-9.412425	1.0
3	-7.792884	-5.845229	-1.0
4	6.466569	-0.609774	1.0



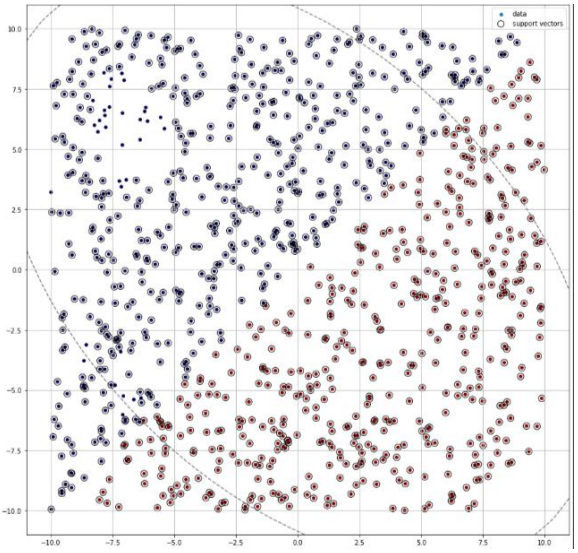
ابتدا به پیاده‌سازی الگوریتم SVM با kernel = linear می‌پردازیم. نتایج حاصل در زیر نشان داده شده است. از آنجا که همانطور که در شکل مشاهده میشود داده‌های ما خطی هستند به این معنا که می توان نقاط قرمز را از آبی با ترسیم خط بین این دو مجموعه نقاط جدا کرد فلذا کرنل خطی به راحتی و با دقت بسیار مناسب می‌تواند آن‌ها را دسته‌بندی کند. Score محاسبه شده برای این کرنل بر روی این مجموعه داده برابر است با score = 0.997. در زیر نمودار آن که شامل خط جداکننده و margin است، نشان داده شده است.



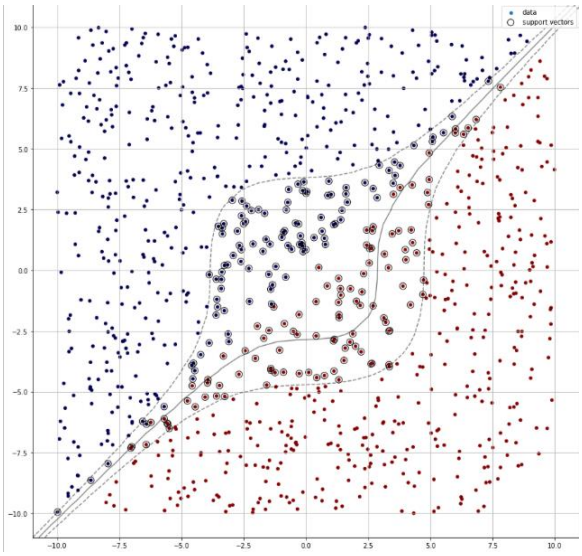
با توجه به اینکه داده های قرمز و ابی در ناحیه های متمرکز مخصوص به خود قرار دارند کرنل rbf هم عملکرد مناسبی به جا گذاشته است ؛ score مربوط به این کرنل برابر است با :0.992 و شکل مربوط به این کرنل را در زیر مشاهده می کنید :



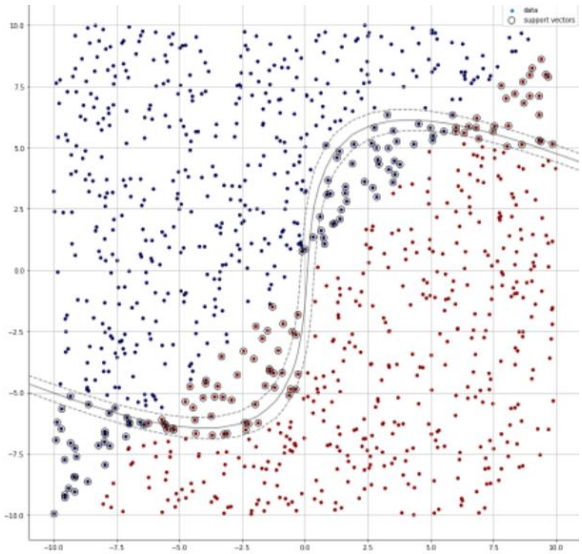
کرنل پولی نومیال درجه زوج بر روی این مجموعه داده عملکرد مناسبی ندارد و پولی نومیال درجه دو score برابر با 0.52 ثبت کرده است ؛ نمودار مربوط به این کرنل را در شکل زیر مشاهده می کنید :



برخلاف پولی نومیال با درجه زوج ، پولی نومیال با درجه فرد به خوبی از پس دادگان بر آمده و score برابر با 0.96 ثبت کرده است ؛ نمودار مربوط به این کرنل را در شکلی زیر مشاهده می کنید :



Score مربوط به کرنل سیگموید هم برابر 0.874 شده است و نمودار مربوط به این کرنل را در قسمت زیر مشاهده می کنید :



همانطور که مشاهده می شود بهترین کرنل برای این داده ، کرنل خطی می باشد

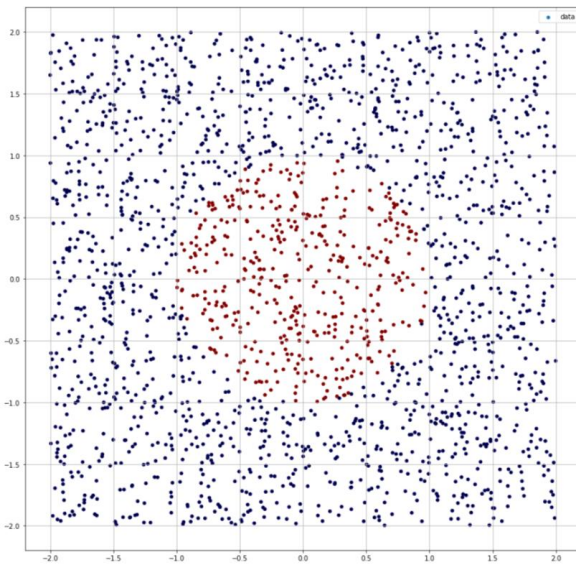
kernel	linear	rbf	Poly(degree =2)	Poly(degree =3)	sigmoid
score	0.997	0.992	0.52	0.96	0.874

Example 2

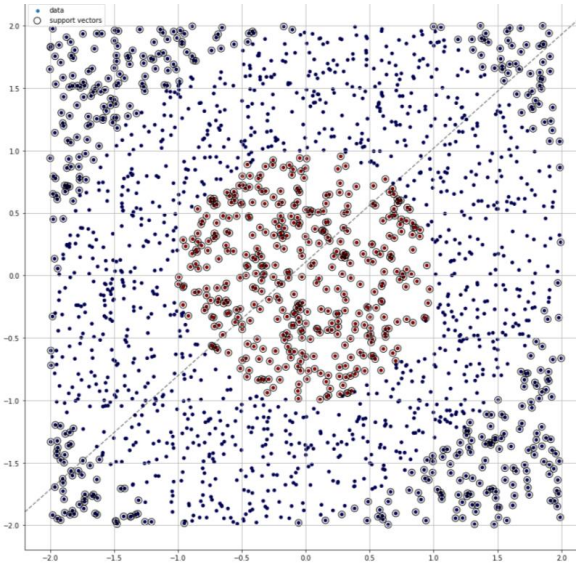
مجموعه داده بعدی، ۲۰۰۰ نقطه در بازه‌ی ۲- تا ۲ است. تابع اعمال شده تابع دایره است. همان‌طور که در زیر مشاهده می‌شود، فرمول ساخت دایره بر روی این مجموعه داده اعمال شده است و نقاط قرمز دایره به شعاع کمتر از ۱ می باشند . DataFrame ایجاد شده برای آن نیز در زیر مشاهده می‌شود.

```
dataset = generate_binary_dataset(-2, 2, 2000, lambda x, y: x ** 2 + y ** 2 < 1)
dataset.head()
```

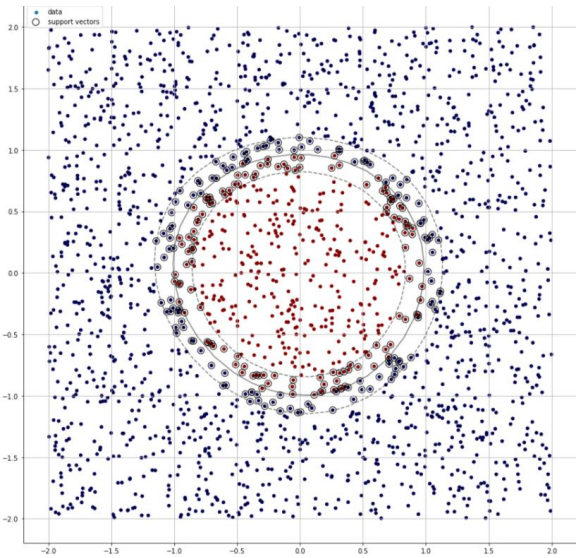
	x	y	target
0	-0.196275	0.476149	1.0
1	0.551684	-0.259606	1.0
2	-1.182871	-1.030645	-1.0
3	1.128409	1.052292	-1.0
4	-0.805193	-1.544665	-1.0



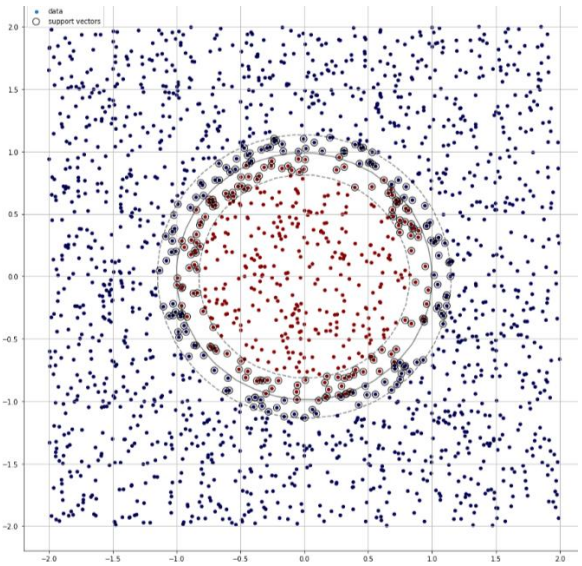
به دلیل غیرخطی بودن این تابع (تابع دایره‌ای)، اجرای الگوریتم SVM با kernel = linear، نتیجه قابل قبولی از دسته‌بندی نقاط را به ما ارائه نخواهد داد؛ زیرا همان‌طور که می‌دانید، کرنل linear برای دسته‌بندی داده‌های خطی استفاده می‌شودو برای این مساله هم score کرنل خطی برابر است با 0.7875 و شکل مربوط به این کرنل را در زیر مشاهده می کنیم :



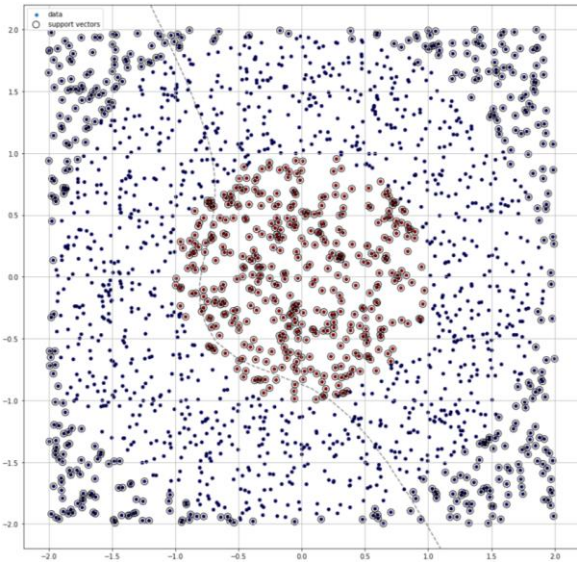
چنانچه برای این داده‌های غیرخطی که به صورت دایره ای توزیع یافته اند، الگوریتم SVM با kernel = rbf را اجرا می‌کنیم. score حاصل از اجرای این الگوریتم برابر است با 0.995 که مشاهده می‌شود که با scoreای نزدیک به 100 این داده‌ها دسته‌بندی شده‌اند. در شکل زیر خط جداکننده به همراه margin ایجاد شده توسط اجرای الگوریتم نشان داده شده است.



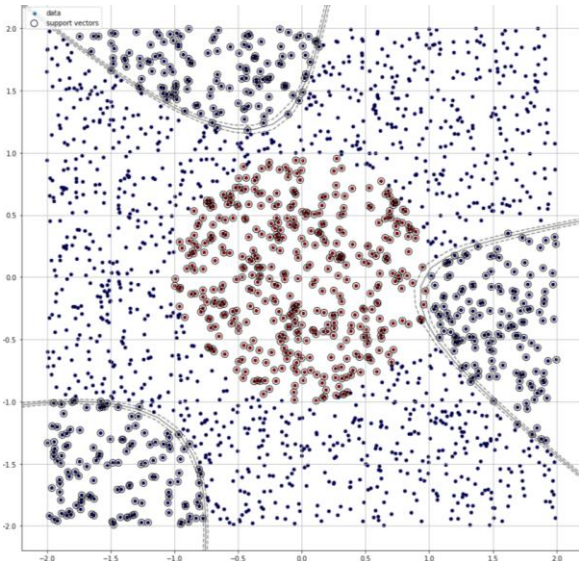
همانطور که پیشبینی می شد پولی نومیال درجه ۲ هم مانند rbf عملکرد بسیار خوبی را داشته است به طوری که score برابر با 0.997 ثبت کرده است



پولی نومیال درجه فرد (درجه ۳) اما عملکرد چندان قابل قبولی نداشته به طوری که score برابر با 0.7875 ثبت کرده است



کرنل سیگموید هم ضعیف ترین عملکرد مربوطه را داشته و score برابر 0.5845 را ثبت کرده است و نمودار مربوط به این کرنل را در شکل زیر مشاهده می کنید :



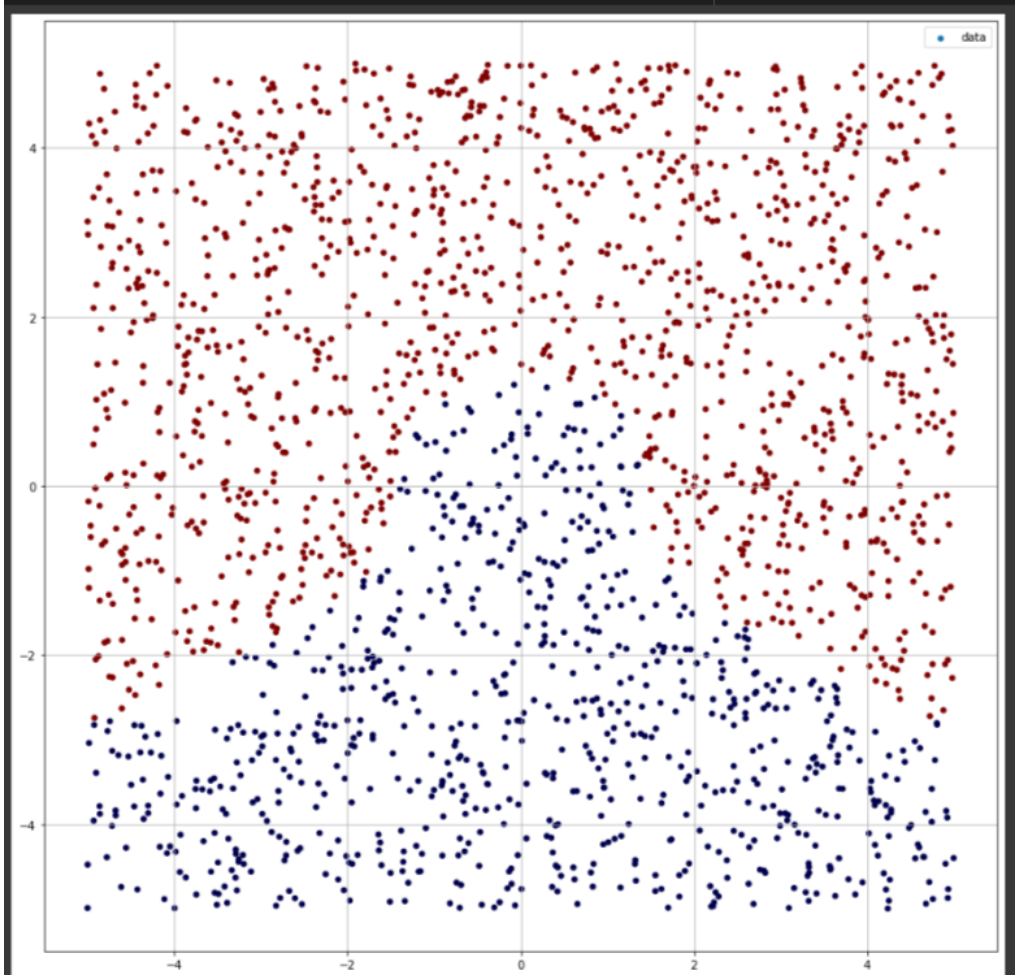
kernel	linear	rbf	Poly(degree =2)	Poly(degree =3)	sigmoid
score	0.7875	0.995	0.997	0.7875	0.5845

کرنل پولی نومیال با اختلاف بسیار اندک از کرنل rbf بهترین عملکرد را بر روی دادگان به دست اورد .

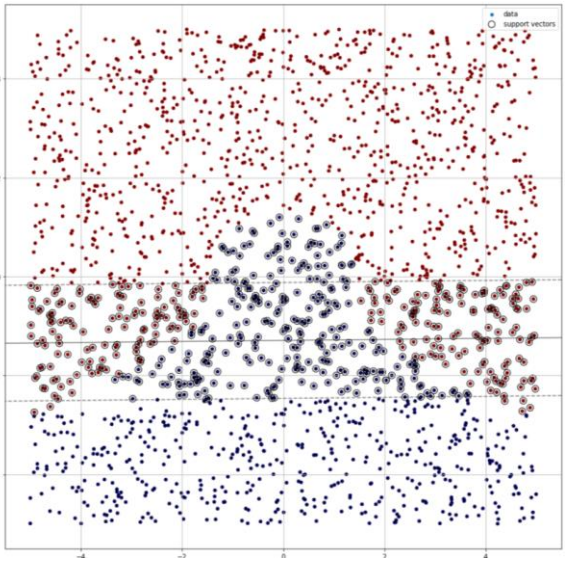
Example 3

در قسمت بعدی باز یک تابع غیرخطی دیگر روی ۲۰۰۰ داده در بازه ۵- تا ۵ اعمال شده است که شکل توزیع آن در پایین آورده شده است.

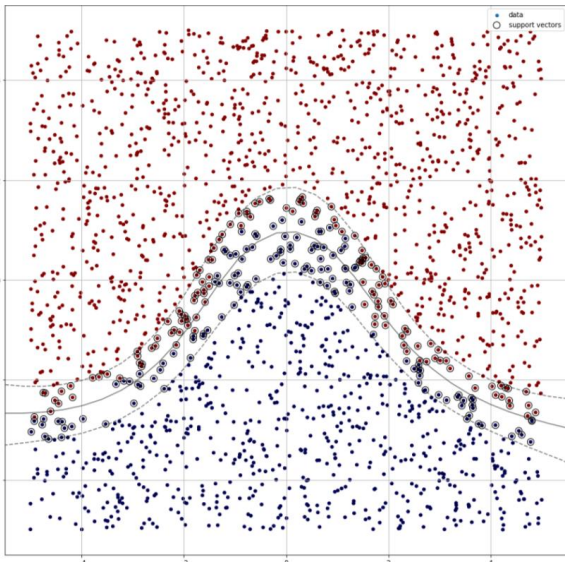
```
dataset = generate_binary_dataset(-5, 5, 2000, lambda x, y: x ** 2 +  y**3 > 2)
dataset.head()
plot_binary(dataset)
```



الگوریتم SVM به همراه کرنل خطی اجرا میکنیم که بر خلاف پیش بینی نگارنده عملکرد نسبتا قابل قبولی به دست آورده است و score مربوط به آن برابر است با 0.8835 ؛ نمودار مربوط به این کرنل را در شکل زیر مشاهده می کنید :

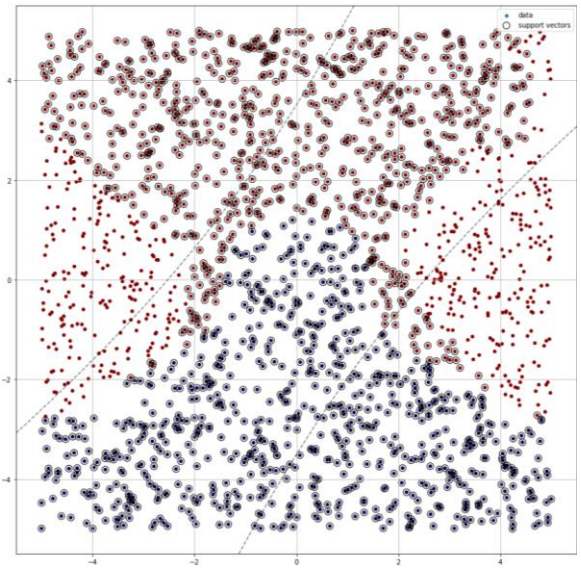


در مورد کرنل rbf با توجه به شکل گوسی دادگان میتوان حدس زد که عملکرد مناسبی بر روی این دادگان دارد و به طور مثال score مربوط به این کرنل برابر است با 0.9815 و شکل مربوط به این کرنل را در زیر مشاهده می کنید :

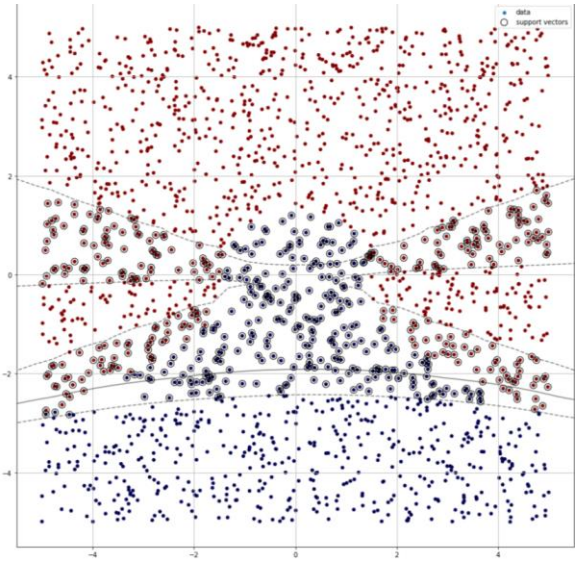


علاوه‌بر کرنل rbf، الگوریتم را با kernel = poly نیز بر روی داده‌ها اجرا می‌کنیم. درجه (degree) را هم برابر ۲ قرار می‌دهیم. score بدست آمده با استفاده از این کرنل برابر با 0.6155 شده است. همانطور که مشاهده می‌شود، این score نسبت به score بدست آمده در هنگام استفاده از کرنل rbf، بسیار پایین است.

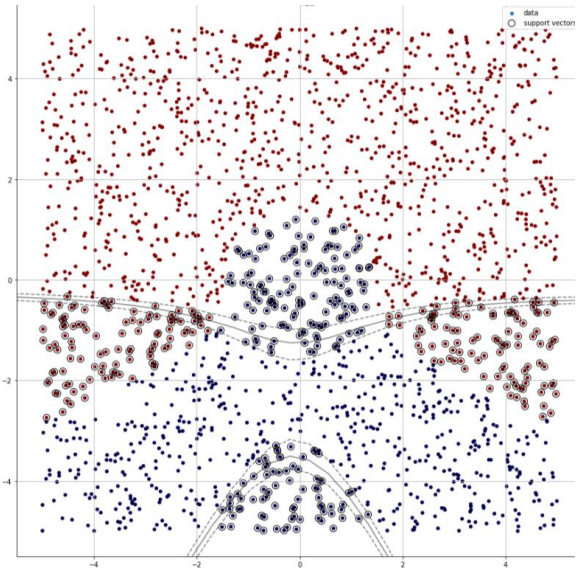
اگر به نمودار آن نیز دقت شود، متوجه می‌شویم که جداسازی داده‌ها به خوبی کرنل rbf نیست:



اگر دوباره از همین کرنل poly اما این بار با degree = 3 الگوریتم را اجرا کنیم، طبق پیش بینی پولی نومیال درجه فرد مشاهده می‌شود که score بدست آمده در این حالت برابر با 0.889 می‌شود و آن این است که درجه‌ی فرد تابع پلی نومیال استفاده شده که بر روی دادگان گوسی عملکرد بهتری نسبت به درجه زوج دارد و در این جا score بدست آمده به میزان قابل توجهی افزایش پیدا کرده و به score بدست آمده از کرنل rbf نیز نزدیک شده است. بنابراین استفاده از پارامترهای مختلف در نتیجه‌ی اجرای الگوریتم بسیار تاثیرگذار است و در شکل زیر نمودار های مربوطه را در زیر مشاهده می کنید :



در آخر نیز از کرنل سیگموید استفاده می‌شود. score حاصل برای این کرنل برابر است با 0.814 و در زیر نمودار های مربوطه را مشاهده می کنید :



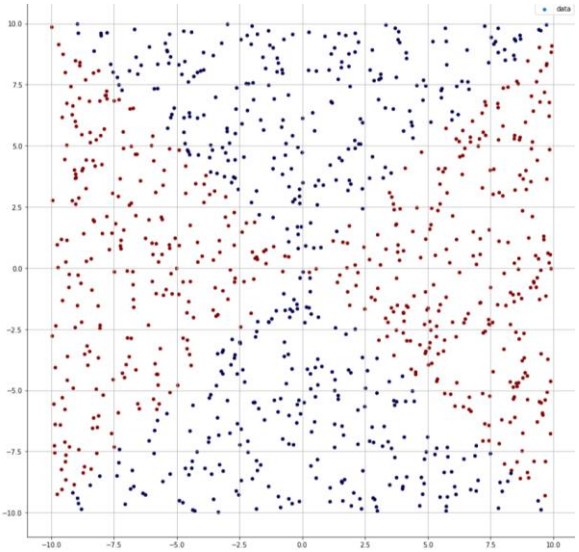
kernel	linear	rbf	Poly(degree =2)	Poly(degree =3)	sigmoid
score	0.8835	0.9815	0.6155	0.889	0.814

همانطور که مشاهده می شود کرنل rbf بهترین عملکرد را در مجموعه دادگان شماره ۳ (که حالت گوسی داشت) داشت .

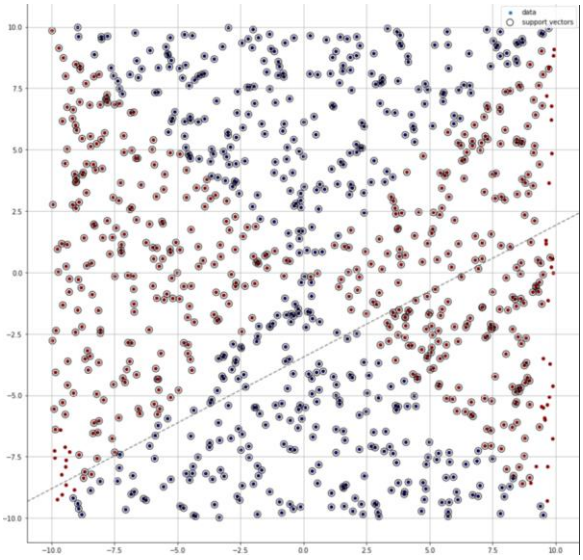
Example 4

در قسمت بعدی از تابع دیگری استفاده شده است. این تابع بر روی ۱۰۰۰ نقطه در بازه ۱۰- تا ۱۰ اعمال شده است.

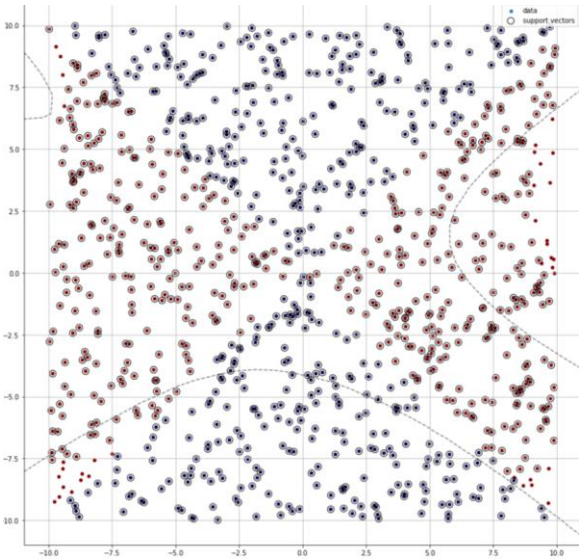
```
dataset = generate_binary_dataset(-10, 10, 1000, lambda x, y: x ** 2 - y**2 > 0)
dataset.head()
plot_binary(dataset)
```



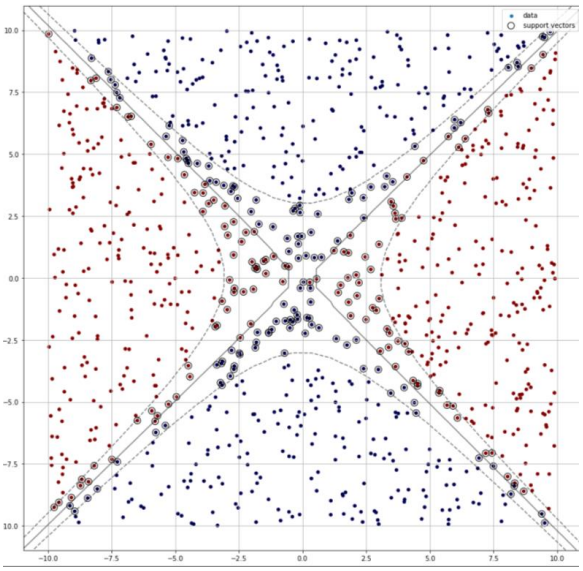
ابتدا الگوریتم SVM را با کرنل linear اجرا می کنیم ، score مربوط به آن برابر شده است با 0.523 و نمودار مربوط به این کرنل به شرح زیر است :



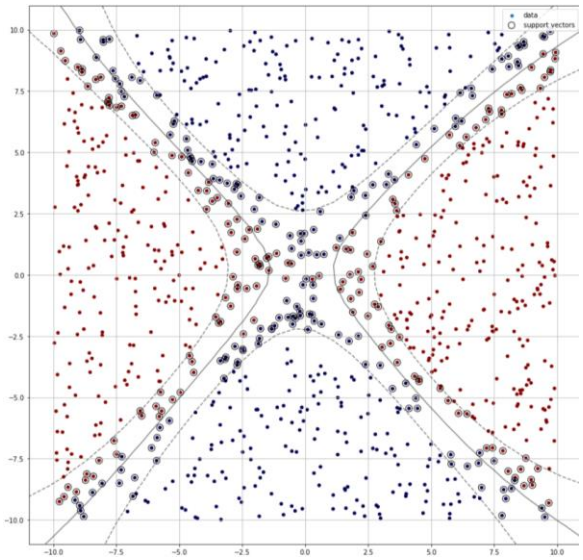
سپس کرنل پولی نومیال را با درجه 3 اجرا می‌کنیم که score ای برابر با 0.523 را خروجی می‌دهد.



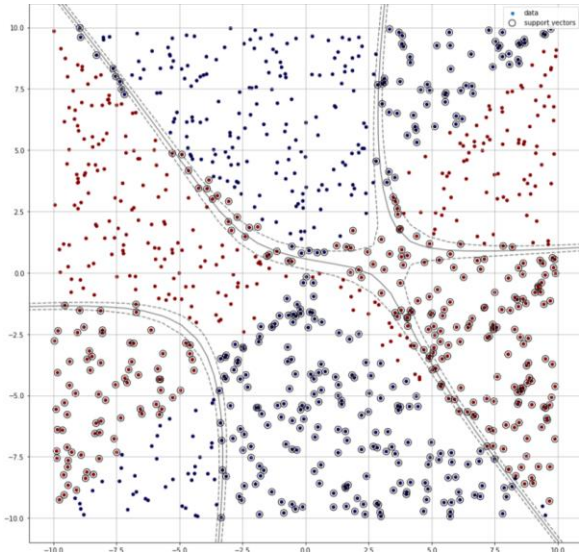
در ادامه نیز همان کرنل پولی نومیال را اما ایندفعه با degree = 2 اجرا می‌کنیم و می‌بینیم که score بدست آمده برای آن برابر است با 0.987 ، نمودار مربوط به این کرنل را در زیر مشاهده می کنید



چنانچه نیز بر روی این مجموعه دادگان کرنل rbf را اعمال کنیم ، score ای برابر با 0.972 به دست خواهیم آورد



ضعیف ترین عملکرد هم مربوط به کرنل sigmoid می باشد که score آن برابر است با 0.518 و نمودار مربوط به آن را در شکلی زیر مشاهده می کنید :



kernel	linear	rbf	Poly(degree =2)	Poly(degree =3)	sigmoid
score	0.523	0.972	0.987	0.523	0.518

در این مجموعه داده نیز پولی نومیال درجه زوج (درجه ۲) عملکرد بهتری نسبت به سایر کرنل ها داشته ولی فاصله کرنل rbf نیز با این کرنل بسیار کم بوده و عملکرد مناسبی به جا گذاشته است .

چالش ها و نتایج بخش اول :

در کل نتایجی که به دست آمد تا حدودی قابل پیش بینی بود ، به طور مثال اینکه بر روی داده های مثال شماره ۱ کرنل خطی عملکرد بسیار قابل قبولی داشته باشد و یا بر روی دادگان شماره دو که به صورت دایره توزیع شده بودند کرنل rbf عملکرد بسیار مناسبی داشته و بر روی دادگان شماره سه که حالت گوسی داشته کرنل rbf بهترین عملکرد را دارد و بر روی دادگان شماره ۴ نیز کرنل پولی نومیال با درجه زوج عملکرد بهتری داشته ؛ اما از نکات قابل توجه های این قسمت اینکه در هر ۴ حالت کرنل پولی نومیال و کرنل rbf عملکرد بالایی داشتند ، در مورد مثال قوسی و خطی (مثال های ۱ و ۳) درجه کرنل پلی نومیال فرد مناسب تر از زوج است و در مثال های دایره و تابع مثال شماره ۴ که درجه ۲ می باشد ، درجه پولی نومیال زوج درجه مناسب تری بوده و در کل نیز کرنل rbf عملکرد مناسبتری بر هر ۴ مثال به جای گذاشته است می توان علت این رفتار در فلسفه کرنل rbf یافت ؛ چرا که این کرنل بر اساس فاصله بین نقاط (دو نقطه X1 و X2 را در نظر بگیرید) کلاس بندی می کند و داده هایی هم که ما در این قسمت ایجاد کردیم معمولا در کنار داده های هم لیبل خود بودند پیچیدگی نسبتا پایین و توزیع نسبتا یکنواختی داشتند ، در مورد کرنل خطی هم این کرنل مشخصا فقط در مورد داده هایی که بتوان آن ها را به صورت خطی از هم جدا کرد عملکرد مناسبی داشتند . کرنل سیگموید نیز برای مجموعه دادگانی که به شکل تابع سیگموید توزیع یافته اند مناسب تر می باشد که البته هیچ

یک از دادگان ما به این شکل نبودند و شاید بتوان با کمی اغماض دادگان شماره ۱ را به عنوان عملکرد نه چندان بد این کرنل ثبت کرد .

از چالش هایی هم که در این مرحله مواجه شدم ، نحوه رسم برخی نمودار ها مربوط به مدل هایی که به درستی از پس طبقه بندی بر نیامدند بود ، به طور مثال کرنل خطی در مجموعه دادگان ۴ یا کرنل پولی نومیال درجه زوج در مثال اول ، که البته به نظر بنده این مشکل ریشه در نوع داده ها و کرنل مورد استفاده میدانم در نوع کد زدن چرا که کرنل های نامبرده برای آن مثال ها مناسب نبوده و توانایی تفکیک لازم را نداشتند .

بخش دوم:

در این بخش مجموعه داده‌های MNIST برای یادگیری و آزمون درنظر گرفته شده که شامل یک سری تصویر از اعداد دستنویس است. در این قسمت قصد داریم با استفاده از الگوریتم SVM و اعمال کرنل‌ها و پارامترهای مختلف، این کاراکترها را شناسایی و دسته‌بندی کنیم. لازم به ذکر است این قسمت از پروژه در ترم پیش و در پروژه طراحی پترن پیاده سازی شده است و از آن در این بخش استفاده کرده ام .

تصاویر به صورت نرمال و هم‌اندازه به همراه برچسب (عدد واقعی) در اختیار ما قرار داده شده است. این مجموعه دادگان به دو بخش آموزش (train) و آزمون (test) تقسیم شده و از طریق آدرس زیر قابل دسترس است:

http://yann.lecun.com/exdb/mnist/

مجموعه دادگان MNIST یک مجموعه آموزش شامل ۶۰۰۰۰ تصویر و یک مجموعه آزمون شامل ۱۰۰۰۰ تصویر دارد. این پایگاه داده یکی از پایگاه‌های محبوب برای آموزش تشخیص الگو به شمار می‌رود.

فایل‌ها در فرمت IDX قرار دارند. IDX یک فرمت ساده برای بردارها و ماتریس‌های چندبعدی از انواع مختلف عددی است. قالب اصلی فایل‌ها به صورت زیر است:

- دو بایت ابتدایی همواره برابر صفر است.
- بایت سوم نوع داده را مشخص می‌کند.
- بایت چهارم تعداد ابعاد بردار یا ماتریس (۱ برای بردارها و ۲ برای ماتریس‌ها) را نشان می‌دهد.

ابتدا داده‌ها را از سایت معرفی شده از بالا فراخوانی کرده و ذخیره می‌کنیم.

سپس با توجه به فرمت بیان شده، داده‌ها را از شکل ابتدایی خود خارج می‌کنیم.

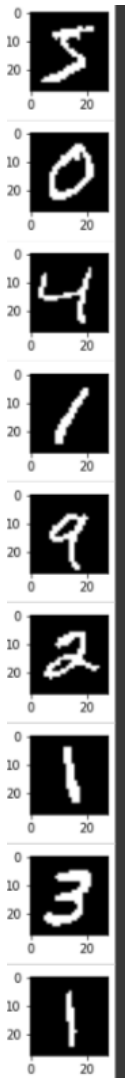
```python
def load_mnist(train_data=True, test_data=False):
    RESOURCES = [
        'train-images-idx3-ubyte.gz',
        'train-labels-idx1-ubyte.gz',
        't10k-images-idx3-ubyte.gz',
        't10k-labels-idx1-ubyte.gz']
    if (os.path.isdir('data') == 0):
        os.mkdir('data')
    if (os.path.isdir('data/mnist') == 0):
        os.mkdir('data/mnist')
    for name in RESOURCES:
        if (os.path.isfile('data/mnist/'+name) == 0):
            url = 'http://yann.lecun.com/exdb/mnist/'+name
            r = requests.get(url, allow_redirects=True)
            open('data/mnist/'+name, 'wb').write(r.content)
    return get_images(train_data, test_data), get_labels(train_data, test_data)
    #train_data = get_images(train_data)
    #test_data = get_images(test_data)
    #train_lable = get_labels(train_data)
    #test_lable = get_labels(test_data)
    #return train_data, test_data, train_lable, test_lable
```

```python
def get_images(train_data=True, test_data=False):
    to_return = []
    if train_data:
        with gzip.open('data/mnist/train-images-idx3-ubyte.gz', 'r') as f:
            # first 4 bytes is a magic number
            magic_number = int.from_bytes(f.read(4), 'big')
            # second 4 bytes is the number of images
            image_count = int.from_bytes(f.read(4), 'big')
            # third 4 bytes is the row count
            row_count = int.from_bytes(f.read(4), 'big')
            # fourth 4 bytes is the column count
            column_count = int.from_bytes(f.read(4), 'big')
            # rest is the image pixel data, each pixel is stored as an unsigned byte
            # pixel values are 0 to 255
            image_data = f.read()
            train_images = np.frombuffer(image_data, dtype=np.uint8)\
                .reshape((image_count, row_count, column_count))
            to_return.append(np.where(train_images > 127, 1, 0))

    if test_data:
        with gzip.open('data/mnist/t10k-images-idx3-ubyte.gz', 'r') as f:
            # first 4 bytes is a magic number
            magic_number = int.from_bytes(f.read(4), 'big')
            # second 4 bytes is the number of images
            image_count = int.from_bytes(f.read(4), 'big')
            # third 4 bytes is the row count
            row_count = int.from_bytes(f.read(4), 'big')
            # fourth 4 bytes is the column count
            column_count = int.from_bytes(f.read(4), 'big')
            # rest is the image pixel data, each pixel is stored as an unsigned byte
            # pixel values are 0 to 255
            image_data = f.read()
            test_images = np.frombuffer(image_data, dtype=np.uint8)\
                .reshape((image_count, row_count, column_count))
            to_return.append(np.where(test_images > 127, 1, 0))
    arr_return = np.array(to_return[0])
    return arr_return
```

در ادامه تعدادی از تصاویر درون این مجموعه داده نشان داده شده است.

```python
from matplotlib import pyplot
for i in range(9):
  pyplot.subplot(330 + 1 + i)
  pyplot.imshow(train_images[i], cmap=pyplot.get_cmap('gray'))
  pyplot.show()
```



در ادامه پیکسل‌های هر تصویر را در یک DataFrame ذخیره می‌کنیم. یک DataFrame با ۷۸۴ ستون که هر ستون بیانگر یک پیکسل از تصاویر است.

```
def imgtodf(img):
  arr = np.empty((0 ,784), int)
  for i in img:
    array_1d = np.array(i).flatten()
    array_1d = array_1d.reshape(1,784)
    arr = np.append(arr, np.array(array_1d), axis=0)
    df = pd.DataFrame(arr)
    print
  return df
```

این DataFrameهای ساخته شده را در قالب csv ذخیره کرده تا در هنگام نیاز از آن‌ها استفاده کنیم.

در ادامه به سراغ پیاده‌سازی الگوریتم SVM با کرنل‌ها و پارامترهای مختلف می‌رویم. لازم به ذکر است تغییر مقدار پارامتر گاما بسیار زمان بر بود و با توجه به آنتن دهی اینترنت بار ها مختل شده و تنها یک بار و برای کرنل rbf اجرا کردیم که مقدار accuracy برابر است با 83.61 درصد که نسبت به کرنل بدون تغییر گاما و با همان مقدار C مقدار به مراتب کمتری است .

```
Accuracy: 0.8361
CPU times: user 4h 18min 28s, sys: 17.8 s, total: 4h 18min 46s
Wall time: 4h 17min 50s
```

در ادامه مقدار ماتریس درهم‌ریختگی (confusion matrix)، دقت (Precison)، فراخوانی (Recall) و معیار f1 را بدست می‌آوریم. تعریف تمامی موارد ذکر شده در بالا آورده شده است (برای سایر کرنل ها نیز این مقادیر به دست آمده است ولی برای پرهیز از شلوغ تر شدن داک ، برای مشاهده آن ها به فایلipynb. در پیوست مراجعه شود)

```
confusion  [[ 818    0  156    2    0    2    1    0    1    0]
 [   0 1107   16    2    1    0    2    0    6    1]
 [   0    0 1022    0    0    0    0    1    9    0]
 [   0    0  173  827    0    1    0    2    6    1]
 [   0    0  176    0  795    0    1    0    2    8]
 [   0    0  192   19    0  677    0    0    4    0]
 [   3    1  237    0    0    6  709    0    2    0]
 [   0    2  165    0    1    2    0  838    3   17]
 [   0    0  223    5    1    1    0    0  744    0]
 [   0    2  145   12   15    2    0    5    4  824]]
Precision: 0.9200691739976633
Recal : 0.8361
f1 score   0.8579899903053473
```

در ادامه برای کرنل های rbf , lineaer , poly degree =2 , poly degree = 3 , poly degree = 4 , sigmoid و با مقادیر c=10 وc = 0.1 و c = 1 محاسبه کردیم با توجه به این موضوع که زمان مورد نیاز برای اجرا برابر ۴ ساعت و ۱۷ دقیقه شد و دلیل این میزان طولانی شدن زمان اجرا برای تغییر fپارامتر گاما از حالت دیفالت بود، به همین دلیل در ادامه روند اجرا دیگر این پارامتر را تغییر ندادیم و از همان حالت دیفالت استفاده شده است.

همچنین قابل ذکر است که مقدار random_state را یک int معرفی می‌کنیم (فرقی نمی‌کند چه عددی) تا هر بار که تابع را اجرا می‌کنیم، از مجموعه داده (آموزشی و آزمایشی) متفاوتی استفاده نکند و داده‌ها فیکس باشند تا نتایج بدست آمده نیز فیکس بوده و تغییری نکند.

نتیجه اجرای مراحل گفته شده در این قسمت به شرح زیر است :

برای کرنل rbf :

C	0.1	1	10
Accuracy	95.34	97.69	98.15

برای کرنل linear :

C	0.1	1	10
Accuracy	91.02	90.33	87.01

برای کرنل poly و درجه ۲ :

C	0.1	1	10
Accuracy	95.46	97.48	97.8

برای کرنل poly و درجه ۳ :

C	0.1	1	10
Accuracy	94.91	97.29	97.54

برای کرنل poly و درجه ۴ :

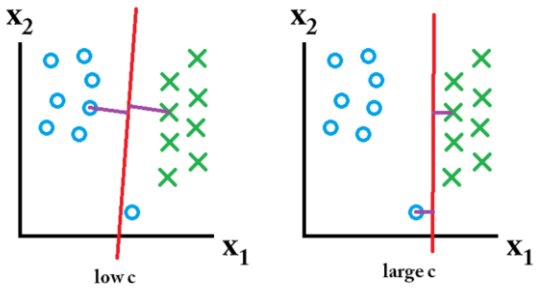
C	0.1	1	10
Accuracy	93.02	96.17	96.71

برای کرنل سیگموید :

C	0.1	1	10
Accuracy	87.32	79.78	77.77

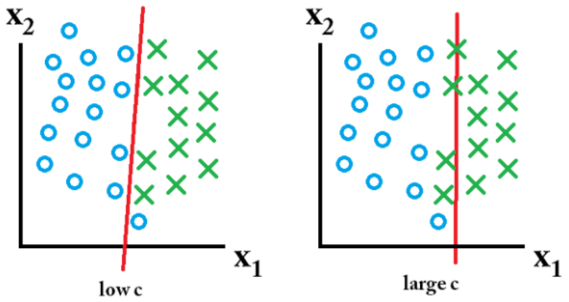
چالش ها و نتایج بخش دوم :

طبق آنچه در مقدمه گفته شد با افزایش مقدار C مقدار accuracy بالاتر رفته که این مورد برای کرنل های poly و rbf صادق است اما برای کرنل های سیگموید و linear نه برای توضیح این قضیه به گمان نگارنده اتفاقی همانند اتفاق زیر رخ داده است :



در مواردی مانند روبه رو ، hyperplane با c بزرگتر به سمت سختگیری بیشتر برای طبقه بندی درست تر می رود که این باعث بالا رفتن مقدار accuracy می شود .

اما در مواردی هم مانند شکل زیر hyperplane مد نظر ما با سخت گیری بیشتر برای طبقه بندی درست accuracy پایین تری ثبت می کند چرا که هدف آن به حداق رساندن FP است (FALSE POSITIVE)



در کل هم میتوان گفت که کرنل rbf با مقدار c=10 عملکرد بهتری نسبت به بقیه کرنل ها دارد .

بخش سوم :

در ابتدا فایل مربوط به این بخش را به صورت فایل rar. در گوگل درایو آپلود کرده تا در محیط کولب از آن استفاده شود .

با کد زیر ، فایل مربوطه را از گوگل درایو در محیط کولب دانلود میکنیم

```
!gdown --id 1mQhnAAP5djmRjmN5rC7KI_-L537Z5Iva
```

سپس آن را unrar می کنیم .

```
!unrar x './persian_LPR.rar'
```

سپس برای تبدیل عکس ها به داده دیتا فریم از تابع dataframeproducer استفاده می کنیم .

```
def dataframeproducer(path , label):
  list1 = []
  dim = (1 , 256 )
  for i in glob.glob(path) :
    image = cv.imread(i , 0)
    image = cv.resize(image , dim)
    list1.append(image)
  num = np.asarray(list1)
  result = num[:, :, 0]
  # print(result.shape)
  df = pd.DataFrame(result)
  # # df['target'] = label
  df['target'] = label
  return df
```

در این تابع دو ورودی را میگیریم ، اولی مسیری است از محیط کولب که فایل تصاویر در آن ها وجود دارد و دومی لیبلی است که میخواهیم به آن تصاویر اختصاص بدهیم ، در ابتدا برای تمام تصاویر موجود در مسیر(path که به عنوان ورودی داده ایم)تصاویر را به صورت flatten در آورده و همه آن ها را در لیست ذخیره می کنیم ، سپس لیست تصاویر را که شامل تصاویر اعداد ۲ و ۳ و۷ و حروف س و ص است را به صورت دیتا فریم در آورده و با توجه به برچسپی که دارند به آن ها لیبل می زنیم .

داده گان را پس از نرمال سازی به نسبت 0.3 برای دادگان تست و 0.7 برای دادگان train در آورده و با استفاده از svm به طبقه بندی آن ها میپردازیم (لازم به ذکر است با توجه به اینکه کرنل rbf با c=10 در قسمت قبل عملکرد بهتری داشته در اینجا نیز از این کرنل استفاده کردیم)

```
x = dataset.drop('target', axis=1).values.astype('float32') / 255
y = dataset.target.values

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3, random_state=1)

model_sec3 = SVC(kernel='rbf' , C = 10 )
model_sec3.fit(xtrain, ytrain)
ypred = model_sec3.predict(xtest)
print("Accuracy is :", metrics.accuracy_score(ytest, ypred))
```

```
Accuracy is : 0.8111111111111111
```

در نهایت مقدار accuracy برابر شد با 81.11 درصد

در ادامه از کراس ولیدیشن نیز استفاده شد (k = 5) که نتایج آن به شرح زیر است :

```
'test_accuracy': array([0.8       , 0.78571429, 0.76666667, 0.81428571, 0.82380952]),
```

چالش ها و نتایج بخش سوم :

در مورد تبدیل دادگان به دیتا فریم با چالش هایی رو به رو شدم که به مدد stackoverflow حل شدند ، لازم به ذکر است یکی دیگر از چالش ها نحوه لیبل زدن به دادگان بود که در ابتدا با مشکلاتی در مورد استفاده از کاراکتر ها رو به رو شده بودم و مجبور به استفاده از اعداد برای دادگان (اعداد ۰ و ۱ برای حروف س و ص) شدم ولی پس از تغییر نحوه کد زنی و استفاده از numpy توانستم از کاراکتر ها هم در لیبل گزاری دادگان استفاده کنم .

منابع :

-١

https://github.com/ahmadsalimi/AiProjects/blob/master/5.%20SVM/SVMClassification.ipynb

-٢

Polynomial kernel - Wikipedia

-٣

In Depth: Parameter tuning for SVC | by Mohtadi Ben Fraj | All things AI | Medium

-٤

sklearn.svm.LinearSVC — scikit-learn 1.0.2 documentation

– ٥

Radial Basis Function (RBF) Kernel: The Go-To Kernel | by Sushanth Sreenivasa | Towards Data Science

– ٦

Major Kernel Functions in Support Vector Machine (SVM) - GeeksforGeeks

– ٧

what is SVM ?, What is RBF kernel, what is Polynomial kernel | FAUN Publication

-٨

Creating linear kernel SVM in Python - GeeksforGeeks

-٩

Kernel Functions for Machine Learning Applications – César Souza (crsouza.com)

– ١٠

https://stackoverflow.com/questions/50765211/three-dimensional-pandas-dataframe-error-must-pass-2-d-input

– ١١

https://stackoverflow.com/questions/41966514/how-fast-in-python-change-3-channel-rgb-color-image-to-1-channel-gray

– ١٢

https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html

https://stackoverflow.com/questions/37152031/numpy-remove-a-dimension-from-np-array