به نام خدا

تمرین عملی پیاده‌سازی سوم هوش مصنوعی

استاد: دکتر آرش عبدی هجراندوست

پیاده‌سازی ماشین بردار پشتیبان (SVM)

محمدصادق مجیدی یزدی ۹۸۱۰۶۰۰۴

فهرست عناوین

نکات کلی و ملزومات اجرای پروژه **3**

بخش اول: تابع را بیابیم **5**

بخش دوم: تابع را با داده‌های نویزدار بیابیم **21**

بخش سوم: تابع را با چند ورودی بیابیم **25**

بخش چهارم: خطخطی را بیابیم **31**

بخش پنجم: این دیگر کدام عدد است؟ **36**

بخش ششم: بیرون کشیدن عدد از میان نویزها **43**

چالش‌ها 50

نکات کلی و ملزومات اجرای پروژه

در پیاده‌سازی پیش رو از کتابخانه‌های numpy و pandas برای کار با مجموعه‌داده‌ها و انجام محاسبات و عملیات‌های لازم استفاده شده است. همچنین از کتابخانه‌ی sklearn برای ساخت مدل‌ها، یادگیری آن‌ها& گزارش و تحلیل نتایج روی داده‌های آزمایشی استفاده شده است. از کتابخانه matplotlib برای نمایش گرافیکی نتایج مدل‌ها استفاده کرده‌ایم. از keras هم برای لود کردن mnist استفاده شده است. برای نصب این کتابخانه‌ها کافیست پس از ایجاد یک python virtual event و اتصال به آن محیط، دستور زیر را در خط فرمان وارد کنید تا عملیات نصب به‌طور خودکار انجام شود.

pip install -r requirements.txt [python3 pip install -r requirements.txt]

البته استفاده از یک محیط توسعه پایتون مانند pycharm باعث تسهیل در اجرای این مراحل خواهد شد. در صورت وجود مشکل اجرای دستا این دستورات می‌تواند کمک کننده باشد.

pip install numpy

pip install pandas

pip install scipy

pip install scikit-learn

pip install matplotlib

pip install tensorflow

pip install keras

پس از آماده شدن محیط اجرای برنامه در صورت استفاده از ترمینال برای اجرا کافیست دستور زیر را اجرا کنید.

python3 filename.py

با اجرای برنامه، خروجی برنامه به صورت گرافیکی تولید و به شما نمایش داده می‌شود و همچنین خروجی متنی گزارشات یادگیری به فرمتی خوانا در کنسول (یا ترمینال) چاپ می‌شود.

هر بخش از پروژه حاوی یک فایل اصلی کد پایتون به فرمت .py است. این امکان در نظر گرفته شده است که در صورت بروز مشکل هنگام دانلود دیتاست mnist از api های آماده keras، از فایل محلی عملیات خواندن دیتاست انجام شود.

بخش اول: جداسازی نقاط

ابتدا به ساختار کد و یک مقدمه کوچک درباره ماشین بردار پشتیبان و نکات آن می‌پردازیم.

```
def plot_binary_svc(features, cls, svc):
    fig = plt.figure(figsize=(10, 10))
    if isinstance(features, pd.DataFrame):
        x, y = features['x'], features['y']
    else:
        x, y = features[:, 0], features[:, 1]
    plt.scatter(x, y, c=cls, s=50, cmap='spring')
    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=25, linewidth=1, label='support vectors',
               facecolors='none', edgecolors='k')
    plt.show()
```

این تابع برای رسم نقاط مربوط به دیتاست دودویی مسئله دسته‌بندی را در صفحه و با رنگ مجزا رسم می‌کند. همچنین با دریافت مدل svm آموزش داده‌شده روی آن‌ها، خط جداکننده دسته‌بند را با یک خط پیوسته تیره و دو خط حاشیه (margin) را با خط تیره‌های مشکی رسم می‌کند. همچنین نقاط مربوط به بردارهای پشتیبان، را برای ایجاد تمایز، یک حاشیه تیره به آن‌ها نسبت می‌دهیم.

```
def make_classification_dataset(size, low_bound, high_bound, criteria):
    dataset = pd.DataFrame({
            'x': np.random.uniform(low_bound, high_bound, size),
            'y': np.random.uniform(low_bound, high_bound, size),
            'class': -np.ones(size)
        })
    dataset['class'][criteria(dataset.x, dataset.y)] = 1
    return dataset
```

این تابع یکی از منابع تولید ما برای نقاط دو دسته است. این تابع با گرفتن یک کران پایین و بالا برای بازه و یک سایز، به تعداد سایز داده‌شده در هر دو بعد نقطه تولید می‌کند. نهایتا با استفاده از تابع شرطی داده‌شده، نقاطی را که در شرط صدق می‌کنند را لیبل ۱ داده و نقاطی که صدق نمی‌کند را

به ۱- نسبت می‌دهیم. نهایتا نقاط تولیدشده با دو بعد فیچر ورودی و یک برچسب دودویی خروجی ۱ و ۱- را به صورت یک دیتاست خروجی می‌دهد.

دیگر منابع ما برای تولید نقاط دودویی برای دسته‌بندی، دیتاست‌سازهای آماده sklearn هستند که از دو تابع make_moons و make_blobs آن در ادامه استفاده شده است.

```
def svm_classification(X, cls, kernel, C=1, gamma='auto', degree=3):
    model = svm.SVC(kernel=kernel, C=C, gamma=gamma, degree=degree)
    model.fit(X, cls)
    plot_binary_svc(X, cls, model)
```

این تابع اصلی تولید مدل و یادگیری آن را انجام می‌دهد و نهایتا نتایج را نمایش می‌دهد. این تابع دیتاست و لیبل‌ها را در ورودی می‌گیرد. پارامتر kernel و پارامترهای C با پیش‌فرض ۱، gamma با پیش‌فرض auto، degree با پیش‌فرض ۳ هم جزو ورودی‌ها هستند.

از مدل آماده SVC که متعلق به کتابخانه sklearn استفاده می‌کنیم. پارامتراهای مهم آن‌را بررسی می‌کنیم. اولین پارامتر kernel است. در سراسر این پروژه از ۳ هسته پرتناوب linear (که در واقع همان بدون هسته است)، rbf و poly استفاده شده است.

هسته linear توضیح خاصی ندارد و صرفا همان فیچر ورودی را خروجی می‌دهد.

هسته rbf به صورت زیر عمل می‌کند:

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

که این کرنل ورودی‌ها را به فضایی با بعد بی‌نهایت نگاشت می‌کند. به کل آن مجموعه ضرایب ثابت gamma هم اطلاق می‌شود. این ضریب گاما اهیت زیادی در فرایند یادگیری با این هسته دارد. اگر مقدار gamma بسیار زیاد شود، مدل دچار overfit شدید شده و دقت در حالت آزمایش فاصله زیادی با آموزش می‌گیرد. از طرفی کم بودن این ضریب باعث می‌شود مدل نتوان اصلا به خوبی یاد بگیرد و underfit شود و دسته‌بندی به خوبی صورت نمی‌گیرد. پس تعیین پارامتر gamma هم اهمیت زیادی برای این هسته دارد. Gamma در دیگر هسته‌ها کاربردی ندارد و بی‌تاثیر است.

هسته دیگر poly است. به صورت زیر عمل می‌کند:

$$K(x, x') = (x.x' + coef0)^{degree}$$

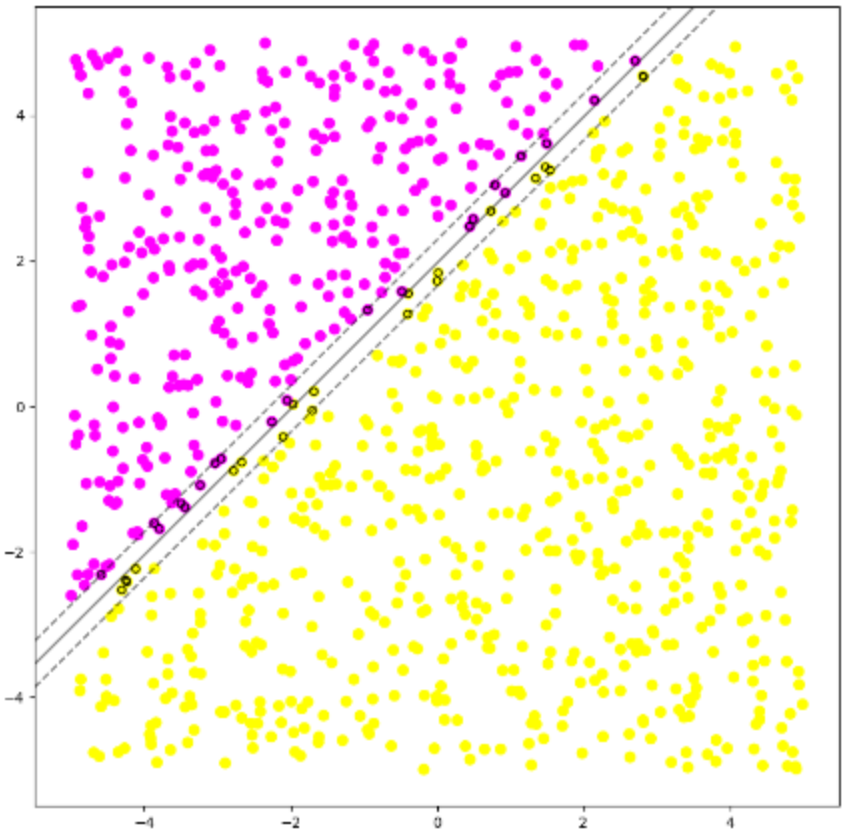
که degreeمان درجه چندجمله‌ای کرنل و coef0 جمله درجه ۰ یا همان ضریب ثابت چندجمله‌ای است. تغییرات degreee بر روی overfit شدن مدل تاثیر بسزایی دارند. چون در این پروژه خیلی به تغییرات این ضرایب برای کرنل poly نپرداختم و از این هسته زیاد استفاده نشده بیش‌تر به آن نمی‌پردازم.

ضریب C یک ضریب برای regularizatoin مدل و نرمال‌سازی آن برای جلوگیری از بیش‌برازش است. البته توجه می‌کنیم که افزایش بیش از حد آن مجددا موجب overfit و کاهش آن منجر به یادیری ضعیف و نتایج غیر قابل قبول می‌شود. این متغیر به صورت پیش‌فرض ۱ قرار داده‌شده است که در طیف گسترده‌ای از مسائل مقداری مناسب و قابل‌قبول است. البته بسته به مسئله این مقدار ممکن است فرق کند تا بتوانیم به مدل بهینه برسیم.

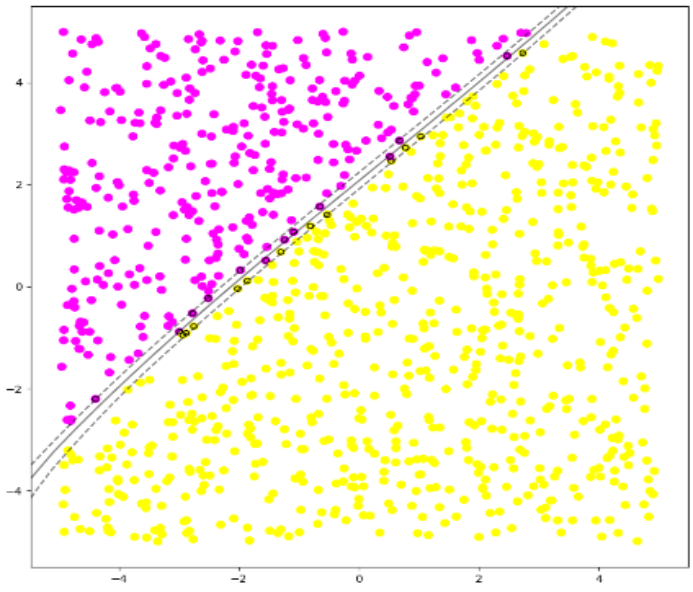
چندین تست کیس بررسی شده است. به هر تست به شکل عددید و رقمی نگاه می‌کنیم. تست‌های با دهگان یکسان در یک دسته بوده و دقیقا از یک مجموعه‌داده یکسان استفاده می‌کنند و صرفا در نوع هسته و پارامترهای دیگر تفاوت دارند.

دسته 0 -> از یک مجموعه داده کاملا خطی جداپذیر استفاده شده است. در این بخش بیشتر تفاوت هسته‌ها روی یک مجموعه داده خطی جداپذیر بررسی شده و به پارامترها پرداخته نشده است.

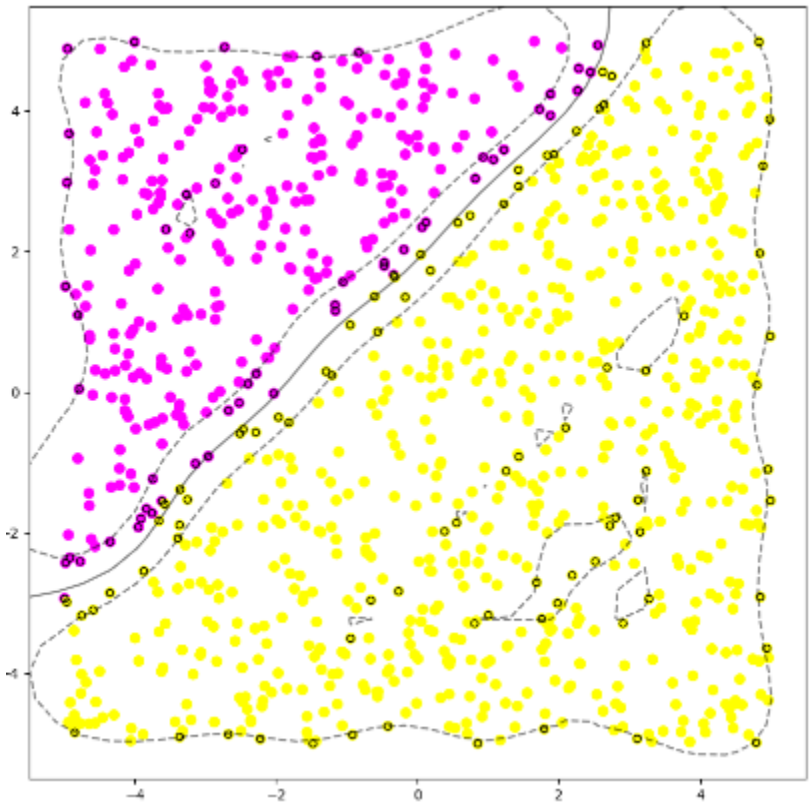
تست 00: با هسته linear و مقادیر پیش‌فرض پارامترها یعنی C=1 دسته‌بندی انجام می‌شود.



تست 01: با هسته poly و مقادیر پیش‌فرض ۰ برای ثابت چندجمله‌ای و درجه ۳ و c=1

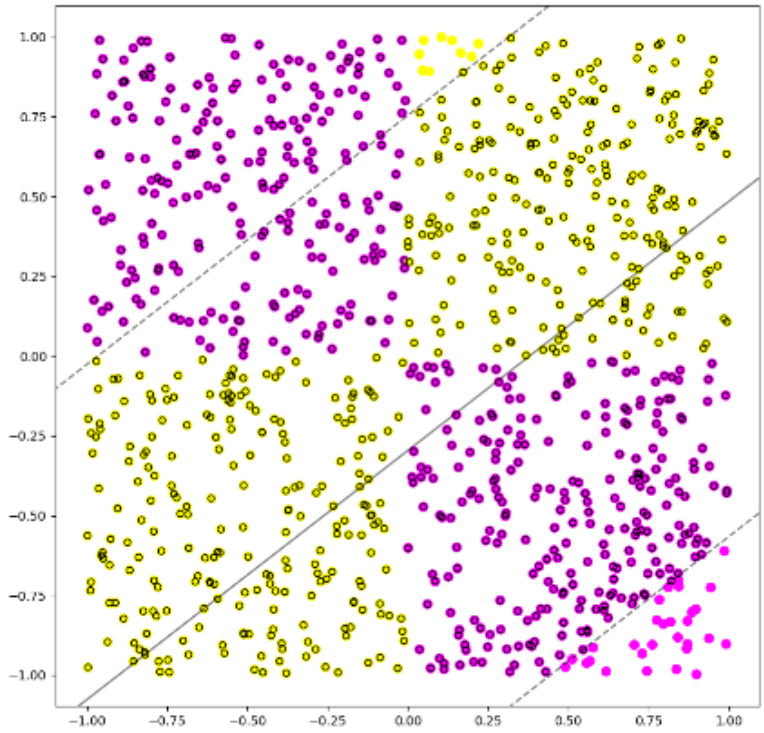


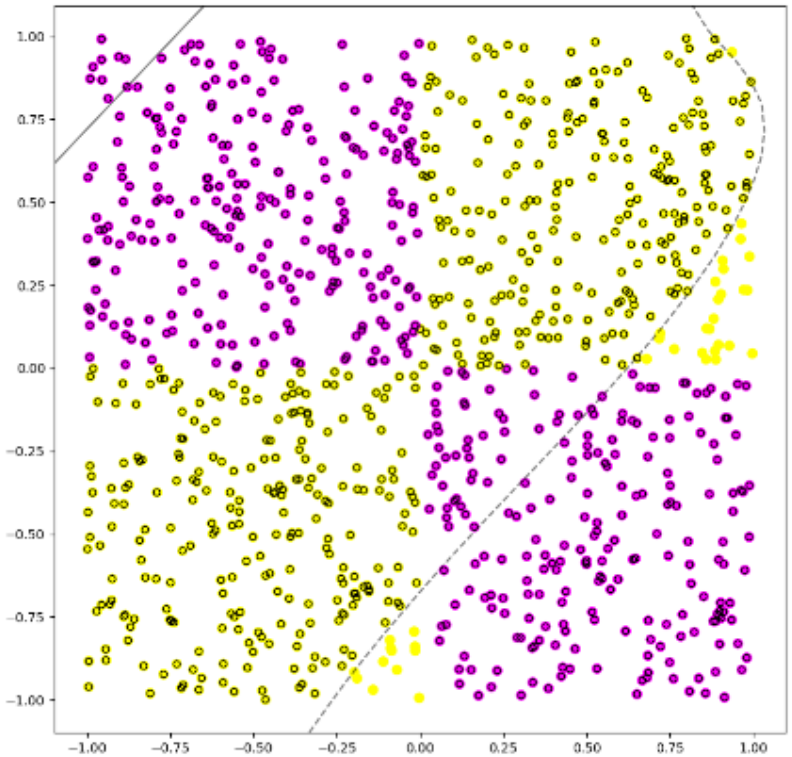
تست 03: هسته rbf با مقادیر پیش‌فرض

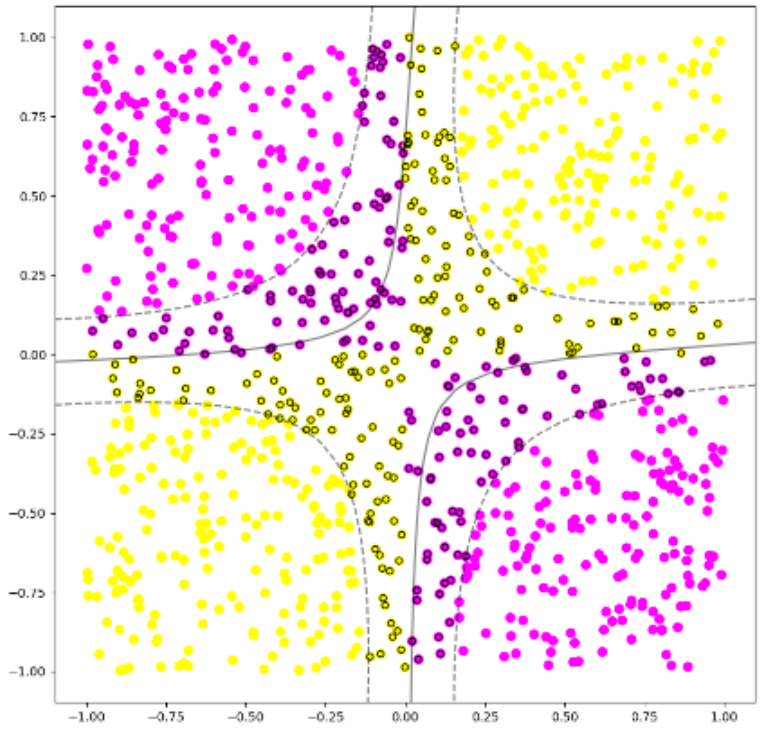


با توجه به این ۳ آزمایش برای یک دسته داده خطی جداپذیر، استفاده از svm بدون هسته مناسب‌تر و کم‌هزینه‌تر است و دقت کافی را دارد. البته polly هم عملکرد خوبی داشت و rbf هم خط جداکننده را درست یافته ولی همانطور که ز حاشیه‌ها مشخص است قدر بالای این هسته باعث ایجاد بیش‌برازش شده است.

دسته ۱: در این دسته تفاوت هسته‌ها با همین پارامترهای پیش‌فرض ولی این بار روی یک مجموعه‌داده ساده اما خطی جداناپذیر بررسی می‌کنیم. به ترتیب linear، polly و rbf بررسی می‌شوند.


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

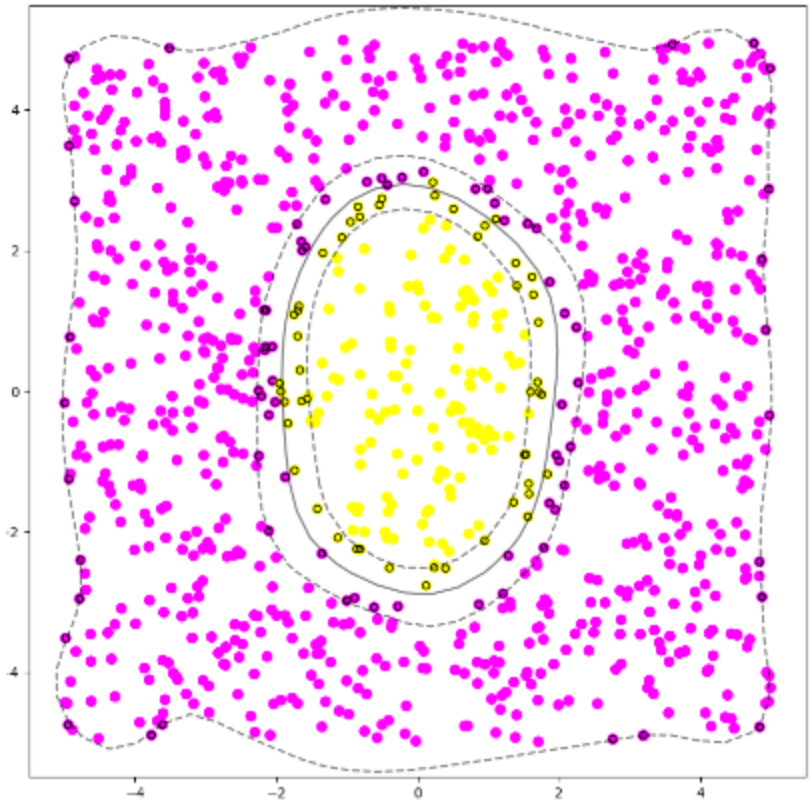


با توجه به این تست‌ها می‌توان نتیجه گرفت که هسته poly در مواجهه با این مجموعه جداناپذیر بسیار بد عمل کرد و انگار نیاز به تغییر پارامترها برای بهتر شدن دارد. مدل بدون هسته نتوانسه به خوبی عمل کند و همان‌طور که مشخص است یک مارجین زیاد ایجاد شده است. اما هسته rbf توانسته یک خط جداکننده نسبتا خوب را ارائه کند و دسته‌بندی را تا حد خوبی انجام دهد. پس در نگاه اول می‌توانیم نتیجه بگیریم هسته rbf در مواجهه با غیرخطیجداپذیری داده‌ها بهتر عمل می کند.

دسته 2 -> داده‌های یک دسته داخل بیضی و بقیه خارج آن هستند.

نکته جالب این است که با استفاده از هسته poly کد مدت زیادی در حال ران شدن بود و چیزی هم به نمایش نرسید. به نظر می‌آید در مواجهه با همچین مجموعه داده‌ای مشکلی در یادگیری این هسته ایجاد می‌شود. این هسته را با توجه به اینکه در کل خیلی علکرد مناسبی ندارد در ادامه بررسی نمی کنیم و بیشتر به هسته rbf و پارامترهای آن و بدون هسته می‌پردازیم.

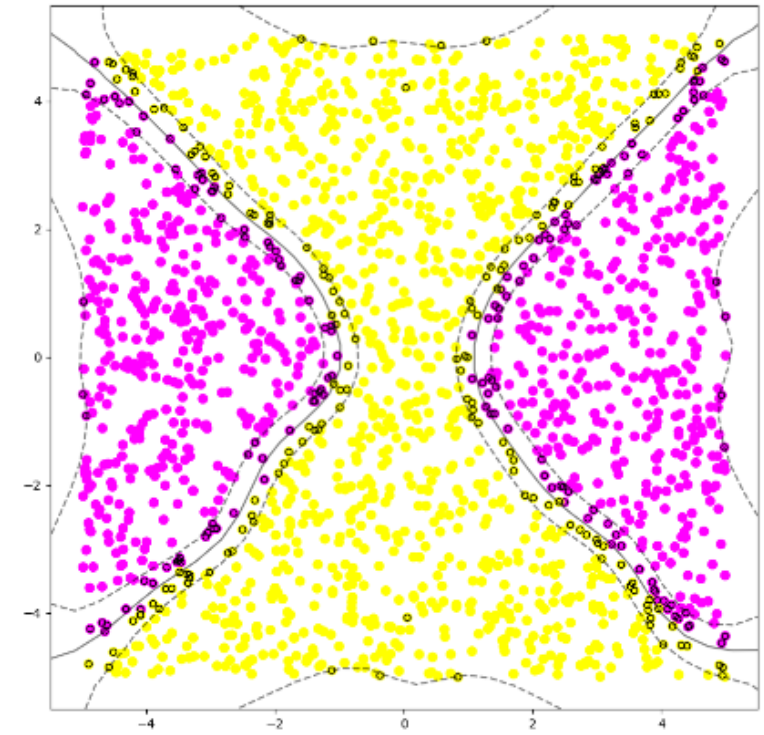
خروجی rbf در تست 22 برای این دادگان به صورت زیر است:

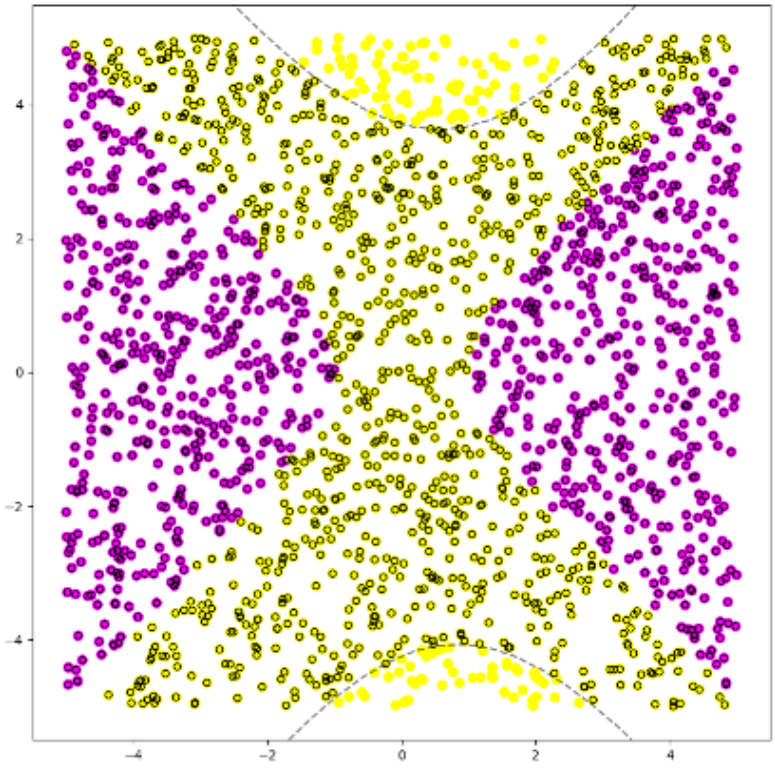


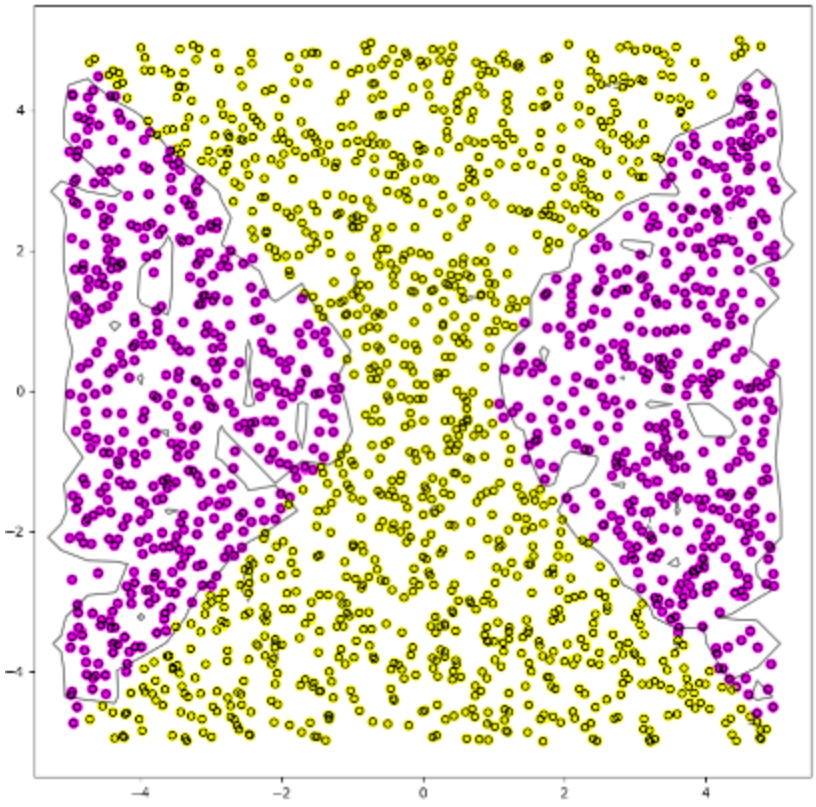
که خط جداکننده به خوبی تشخیص داده شده است.

دسته 3 ->

دادگان این دسته به صورت یک هذلولی از هم جدا شده‌اند. در این دسته قصد داریم با ثابت نگه داشتن هسته rbf، مقدار پارامتر gamma را تغییر داده و نتیجه را بررسی می‌کنیم. تصویر اول مربوط به مقدار auto، تصویر دوم مربوط به مقدار 0.001 و تصویر سوم مربوط به مقدار 100 برای پارامتر gamma است.

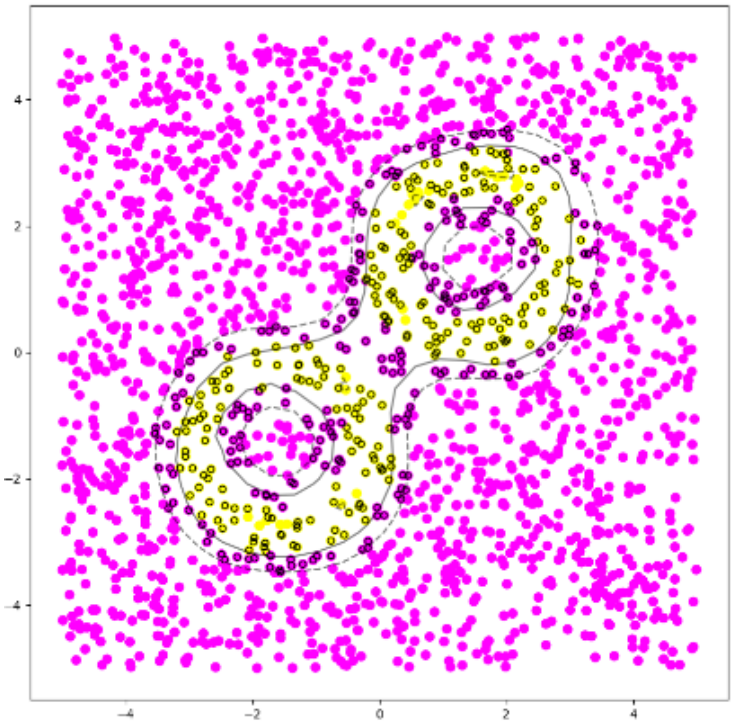

4
2
0
−2
−4
−4
−2
0
2
4

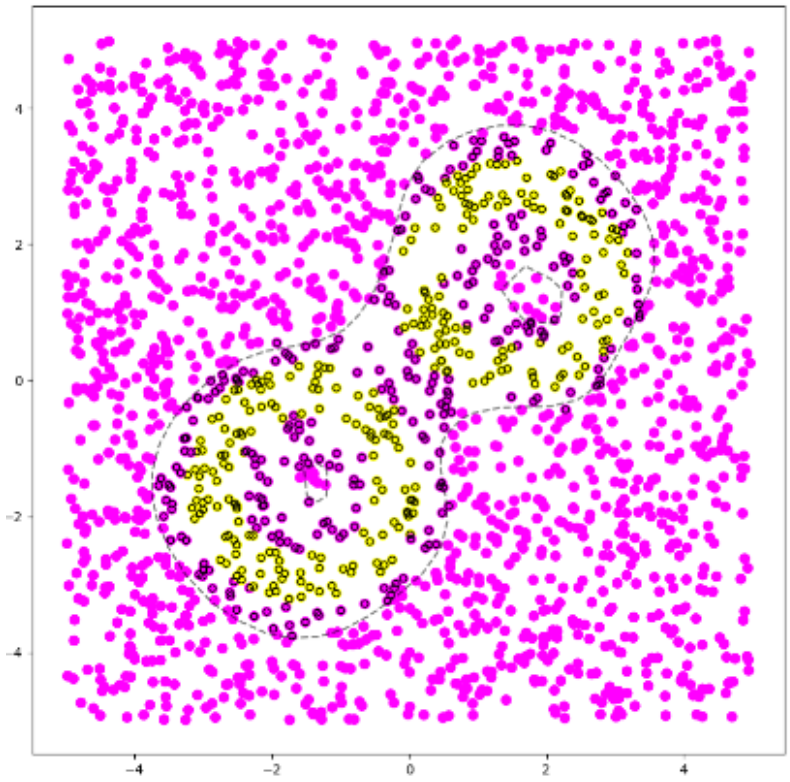

4
2
0
−2
−4
−4
−2
0
2
4

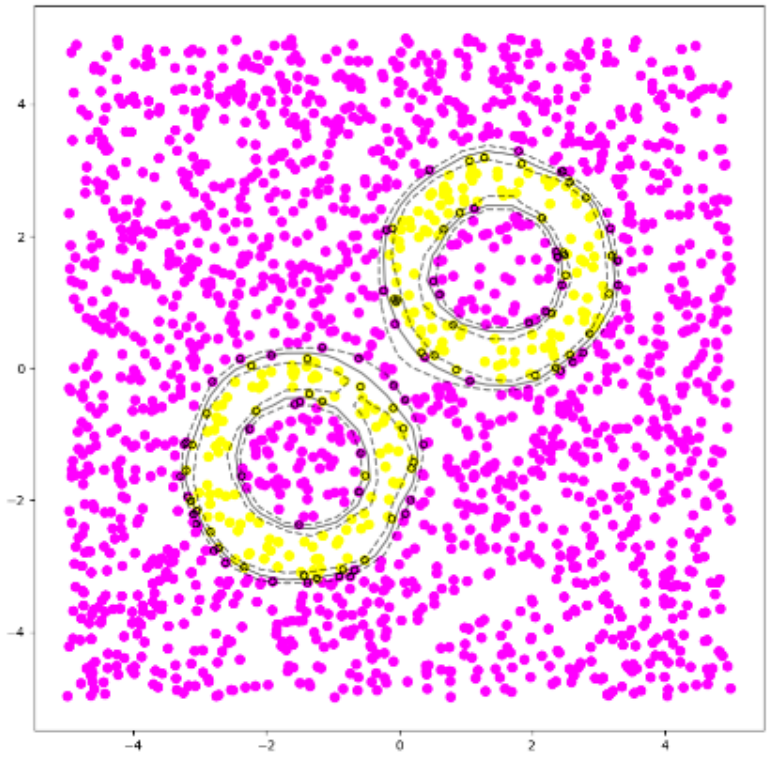


با توجه به این سه آزمایش مشاهده می‌کنیم که مقدار خیلی کم برای gamma باعث ایجاد underfit شده و مدل اصلا الگوی دسته‌ها را خوب یاد نمی‌گیرد. همچنین مقدار زیاد gamma در عکس سوم باعث ایجاد overfit زیاد شده و مدل پیچیدگی زیاد در دسته‌بندی در نظر می‌گیرد. پس نیاز به یک مقدار مناسب برای gamma است تا یادگیری به خوبی انجام شود.

دسته ۴ -> داده‌های این دسته به صورت دو حلقه با داده‌های مثبت و سایر نقاط با داده‌های منفی است. صرفا دسته‌بند با هسته rbf را با مقادیر مختلف C تست می‌کنیم تا تاثیر این ضریب منظم‌سازی بر یادگیری را ببینیم. تصویر اول مقدار ۱، تصویر دوم مقدار 0.001 و تصویر سوم مقدار 100 را برای پارامتر C نشان می‌دهد.

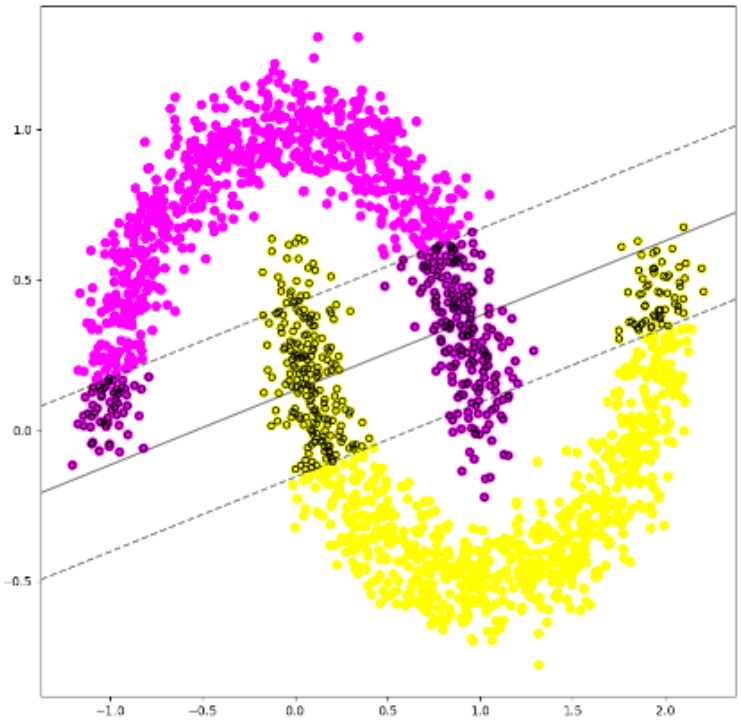

4
2
0
−2
−4
−4
−2
0
2
4

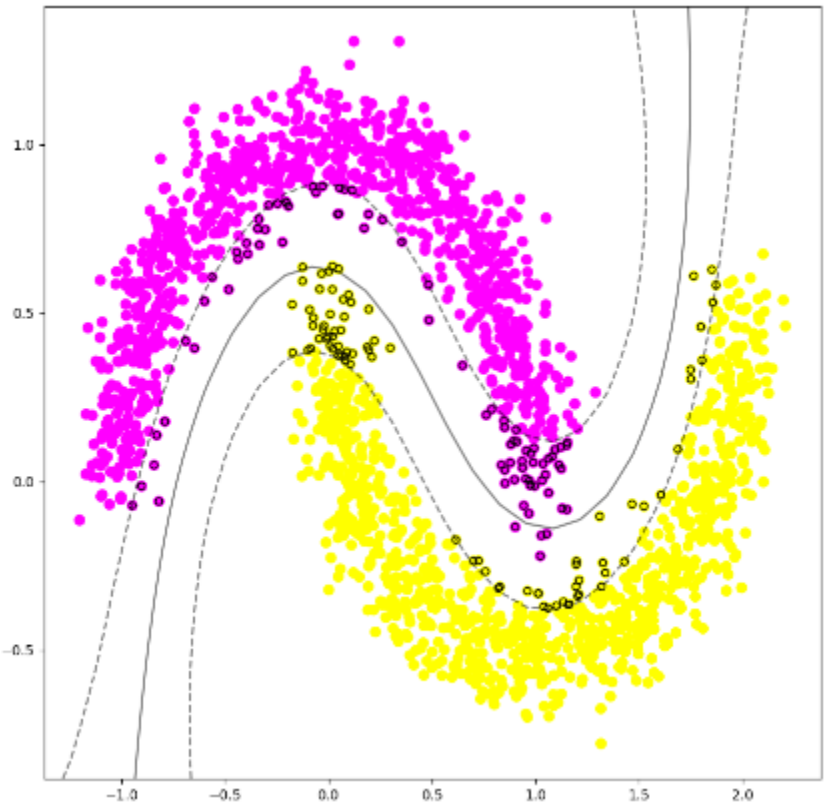

4
2
0
−2
−4
−4
−2
0
2
4

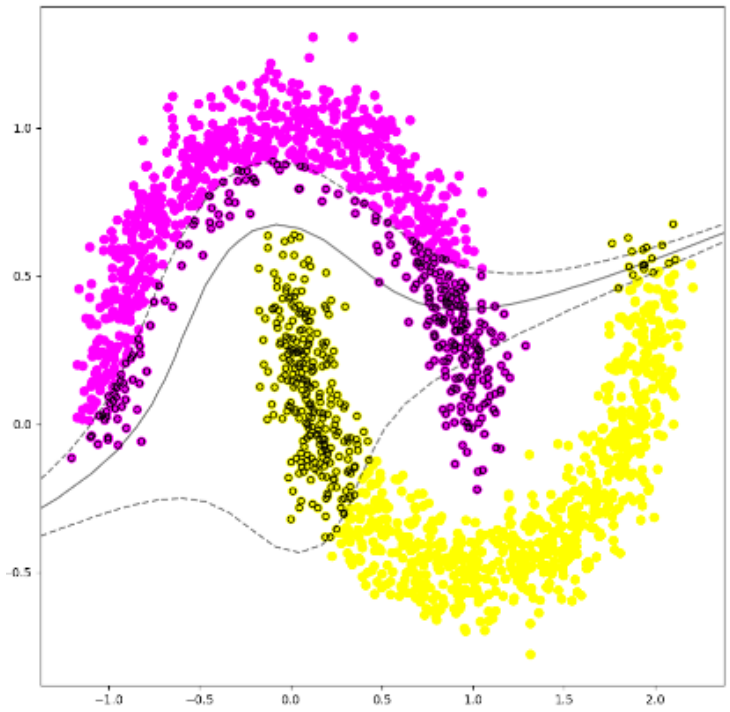


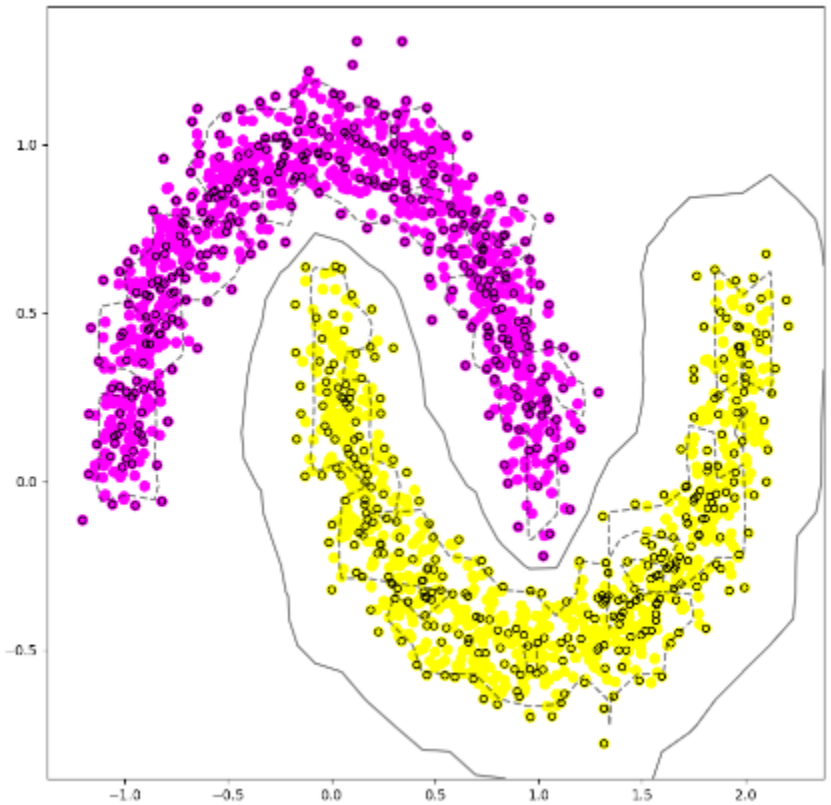
چیز جالبی مشاهده می‌شود. اولا که در حالت C برابر 0.001 بسیار کم یادگیری صورت گرفته و underfit رخ داده است. در حالت 1 نسبتا خوب کلاس‌ها جداسازی شده‌اند ولی داده‌های بین دو حلقه به خوبی کلاس‌بندی نشده‌اند. در حالت ۱۰۰ می‌بینیم که دسته‌ها دقیقا به صورت همان دو حلقه جداشده‌اند و دسته بندی خیلی خوب صورت گرفته است. البته کمی به سمت بیش‌برازش در حال حرکت است ولی نه آنچنان. اگر باز هم C را افزایش دهیم دچار بیش‌برازش می‌شویم.

دسته ۵ -> در این دسته داده‌ها به صورت دو هلال تو در تو که یکی کلاس ۱ و دیگری ۱- است هستند. به ترتیب svm را بدون هسته، با هسته rbf، با هسته poly همگی با پارامترهای دیفالت و یک مدل با هسته rbf و gamma 100 در تست‌های این دسته یاد گرفته‌شده‌اند.

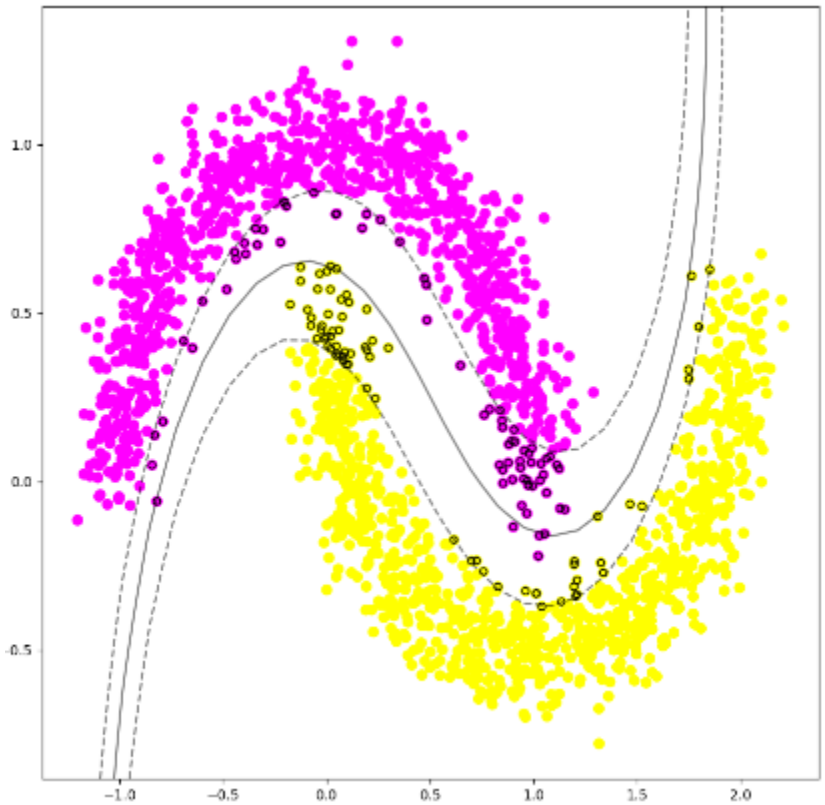

1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

نتایج به دست آمده در این آزمایش را تحلیل می‌کنیم. در تصویر اول که مدل بدون هسته است، چون داده‌ها با خط جدا نمی‌شوند خط جدا کننده سعی کرده است در بهترین مکان و شیب قرار بگیرد تا مقدار loss کمینه شود و تا جای ممکن دسته نقاط را درست تشخیص دهد که البته خیلی موفق نیست. در تصویر دوم مدل با هسته rbf به خوبی خط جداکننده را به همان شکل منحنی بین هلالی یافته و داده‌ها را عالی دسته‌بندی می‌کند. در تصویر سوم مدل با هسته poly با وجود تلاش زیاد که سعی کرده است خم جداکننده‌اش را به سمت خم مطلوب میل دهد، خیلی موفق نبوده و دسته‌بندی مطلوبی انجام نمی‌شود. در تصویر چهارم هم با اینکه دسته ها درست جدا شده‌اند اما اثر آشکاری از رخ دادن overfit قابل مشاهده است و این موضوع از شمایل خط جداکننده و حاشیه‌ها قابل دریافت است. البته این به این معنا نیست که هسته چندجمله‌ای به دردنخور است بلکه باید درجه و ثابت مناسب را برای آن بیابیم. مثلا اگر مقدار ضریب ثابت را در همان تست سوم این بخش از ۰ به ۱/۵ تغییر دهیم نتیجه زیر حاصل می‌شود.



که نتیجه‌ای بسیار مطلوب است. پس نقش مهم این دو پارامتر در هسته چندجمله‌ای محرز شد. تمام این موارد باید برای یک یادگیری خوب در نظر گرفته شود.

در مجموع آزمایش‌های انجام‌شده در این بخش، می‌توانیم نتیجه بگیریم که هسته rbf در اکثر موارد خطی و غیرخطی عملکردی مناسب دارد. البته باید کمی مراقب بیش‌برازش آن باشیم تا دردسرساز نشود چون هسته آن به شدت قدرتمند است. از طرف دیگر هسته چندجمله‌ای عملکردی نه‌چندان دلچسب در این دسته‌بندی‌ها داشته و نیاز است روی پارامترهای آن بیشتر تمرکز شود تا بتواند عملکرد خوب داشته باشد. بدون استفاده از هسته نیز علاوه بر داده‌های خطی، داده‌های خطی‌جداناپذیر هم نه با دقت عالی ولی تا حد خوبی از هم جدا می‌شوند و الگوی کلی کلاس‌ها یافت می‌شود که البته باز هم کیس‌هایی پیدا می‌شوند که عملکرد فاجعه‌ای را به ارمغان می‌آورند.

از نظر زمان اجرا در این بخش، تقریبا همه موارد به سرعت و در حد چند ثانیه اجرا می‌شدند ولی بدون استفاده از هسته سرعت محسوس‌تری نسبت به دو حالت دیگر وجود داشت.

بخش دوم: دسته‌بندی تصویر ارقام

این بخش کد نسبتا کوتاهی دارد. داده‌ها را دقیقا مانند پروژه شبکه عصبی از کتابخانه keras دانلود کرده و به همان شکل نرمال و مسطح می‌کنیم .در مجموع ۷۰۰۰۰ عکس داریم.

کد مدل نیز در زیر آمده است:

```
def k_fold_svm_classification(X, y, kernel, k_folds=5, C=1, gamma='auto'):
    clf = SVC(kernel=kernel, C=C, gamma=gamma)
    history = cross_validate(
        clf, X, y, cv=k_folds, scoring='accuracy', return_estimator=True, return_train_score=True
    )
    ests = history['estimator']
    train_scores = history['train_score']
    test_scores = history['test_score']

    print('-------------------------------------------------')
    print('train_scores: ')
    print(train_scores)
    print()
    print('test_scores: ')
    print(test_scores)
    print('-------------------------------------------------')
    best_index = np.argmax(test_scores)

    return ests[best_index], train_scores[best_index]
```

کلیت خود مدل svm و تنظیمات آن مانند بخش اول است که توضیح داده شد. در این بخش بیشتر تکنیک‌های یادگیری تغییر کرده است. از Cross Validation به صورت k-folds استفاده کرده‌ایم. با این تکنیک، مدل k بار، هر بار با سمپل گرفتن $\frac{k-1}{k}$ داده‌های آموزشی یادگیری را انجام داده و دسته‌بندی بهینه را می‌یابد و مقدار متریک دقت (accuracy) آن فولد را گزارش می‌کند. نهایتا مدل با بهترین متریک در بین همه فولدها به عنوان بهترین مدل یادگرفته‌شده برگردانده می‌شود. به طور پیش‌فرض تعداد fold را برابر ۵ گرفته‌ایم.

یادگیری را با صرفا ۱۰۰۰۰ نمونه رندوم ازداده‌های اصلی انجام می‌دهیم. چون روش SVM احتمالاتی نیستت،طبیعتش به گونه‌ای است که با افزایش داده‌های ورودی،زمان آن نیز به طرز چشمگیری افزایش پیدا می‌کند. اگر یادگیری را با همه داده‌های آموزشی انجام می‌دادم عملیات یادگیری با توجه به

قدرت کم لپتاپ من چند ساعت طول می‌کشید ولی با تعداد 10000تا با هسته‌های مختلف هم نهایتا تا ۱۰ دقیقه به جواب می‌رسیدم و دقت نسبتا خوبی هم داشت. در ادامه اجراهای متفاوت آن را بررسی می‌کنیم.

ابتدا مدل SVM را بدون هسته و با C برابر ۱ و folds-3 ران می‌کنیم. نتیجه زیر حاصل می‌شود.

```
------------------------------------------------
train_scores:
[0.99909991 0.99970001 0.99820009]

test_scores:
[0.90821836 0.90309031 0.91239124]
------------------------------------------------
test score 0.922
best train score 0.9982000899955003
```

با افزایش C دقت ترین به سمت ۱ می‌رود و دقت تست به سمت ۹۰ درصد رفته و بیش‌برازش پدید می‌آید. با کاهش آن نیز دقت هر دوی آموزش و آزمایش کم می‌شود.

با هسته rbf و C برابر 1 و گامای اتوماتیک یادگیری را انجام می‌دهیم.

```
test score 0.919
best train score 0.9191540422978851
```

دقت نزدیک 92 در هر دو دسته آموزش و آزمایش به دست می‌آید. این نشان می‌دهد که بیش‌برازش رخ نداده است. از ۳ فولد برای cross validation استفاده شده است.

این بار آموزش را ۲۰۰۰۰ داده انجام می‌دهیم و نتیجه را می‌بینیم:

```
------------------------------------------------
train_scores:
[0.93146667 0.92953333 0.92993333 0.9304    ]

test_scores:
[0.9178 0.9256 0.9258 0.9264]
------------------------------------------------
test score 0.932
best train score 0.9304
```

این بار دقت به 93.2 می‌رسد و همچنان overfit ندارم.
در کل با بهینه کردن C و gamma و استفاده از ۴ فولد و با استفاده از ۱۰۰۰۰ داده به دقت 94.8 روی داده‌های تست و بدون ایجاد overfit رسیدم. این اجرا حدود ۱۲ دقیقه از من زمان برد که زمان نسبتا زیادی است. حس می‌کنیم با داشتن تعداد داده‌های کامل دقت می‌تواند تا مقدار زیادی حتی نزدیک به ۹۸ یا ۹۹ برسد که در این گزارش توان اجرای آن را نداشتم. در کل svm با هسته مناسب و پارامترهای با دقت می‌تواند کلاس‌بندی خیلی خوب و مطلوبی انجام دهد. اگر بخواهیم از نظر زمانی نگاه کنیم، بدون هسته و در حالت linear در زمان کمتری می‌توانیم به دقت خوبی برسیم. اگر از فولد بیشتری استفاده کنیم مقدار دقت در صورتی که overfit نشده باشیم افزایش پیدا می‌کند. پارامتر C در همه حالات نقش مهمی در جلوگیری از overfit و underfit شدن دارد.
در مجموعه اگر مقایسه‌ای کلی بین شبکه عصبی و ماشین بردار پشتیبان انجام دهیم، از لحاظ تعداد پارامتر و وزن‌هایی که نیاز به یادگیری دارند،svm برتری بی چون و چرایی دارد زیرا روشی محاسباتی بوده و در طبیعت خود وزنی برای یادگیری ندارد و صرفا چند پارامتر کمکی مانند منظم‌ساز اسلک و پارامترهای مربوط به نوع هسته دارد در حالی که شبکه عصبی سراسر وزن و پارامتر است که منتظر یادگیری هستند. از لحاظ مدت زمان لازم برای یادگیری، وقتی کل داده‌ها را در پروژه به شبکه خود دادیم در مدتی نزدیک ۵ دقیقه یادگیری را انجام داد ولی svm با کل داده‌ها حداقل یک ساعت زمان لازم دارد. در اینجا شبکه عصبی برتری محسوسی دارد. در زمینه دقت هم، ممکن است با استفاده از کل داده‌ها دقت svm از شبکه عصبی بیشتر شود، ولی با همین تعداد داده کم دقت svm نزدیک 94 و دقت شبکه عصبی در حدود 96،97 می‌شود و خیلی ساده و در زمان کم به این دقت می‌رسد. بنابراین استفاده از شبکه عصبی در یک شرایط یکسان بسیار سودمندتر از استفاده از svm است. البته باید به این موضوع هم اشاره شود که مدل svm صرفا مناسب انجام دسته‌بندی است. اما شبکه عصبی برای تمام منظورها کاربرد دارد و خیلی مدل پرفایده‌تری است.

بخش سوم: دسته‌بندی پلاک‌ها

کد این بخش مشابه بخش قبل است. یادگیری را با ۳ مقدار متفاوت 0.1 و 1 و 10 برای C و ۳ مقدار gamma به شکل 0.001 و 0.1 و 1 برای هر ۳ کرنل انجام می‌دهیم. و برای هر کدام بهترین را به عنوان مدل دسته‌بند انتخاب و در گزارش می‌آوریم.

```
for i in range(3):
    for j in range(3):

        classifier, score = k_fold_svm_classification(X_train, y_train, k_folds=5, kernel='rbf', C=Cs[i], gamma=gammas[j])
        # classifier, score = k_fold_svm_classification(X_train, y_train, k_folds=5, kernel='linear', C=Cs[i], gamma=gammas
        # classifier, score = k_fold_svm_classification(X_train, y_train, k_folds=5, kernel='poly', C=Cs[i], gamma=gammas[j
        pred_y = classifier.predict(X_test)

        indices = np.random.choice(np.arange(len(X_test)), size=20)
        images = X_test[indices]
        predicted_digits = pred_y[indices]
        texts = [f'{predicted_digits[i]} - expected: {y_test[indices[i]]}' for i in range(len(indices))]
        draw(images, texts, 5, 4, image_size=16, scale=2)

        print(f'C = {Cs[i]}')
        print(f'gamma = {gammas[j]}')
        print(f'test score {accuracy_score(y_test, pred_y)}')
        print(f'best train score {score}')
        print('----------')
```

در آن بخش بعد از بهینه‌سازی پارامترها نیز تعداد ۲۰ داده را انتخاب و کلاس‌بندی روی آن‌ها را به نمایش گرافیکی در می‌آوریم. از 0.2 داده‌ها به عنوان داده آزمایشی و از سایرین برای آموزش استفاده شده است. همچنین Cross Validation در تمامی آزمایش‌ها با ۵ فولد انجام می‌شود.

برای هسته linear با تغییر C حول این ۳ مقدار (گاما بی‌تاثیر است) مقدار 1 برای C مناسب‌ترین است.

```
--------------------------------------------------
C = 1
gamma = 1
test score 0.96
best train score 0.9770833333333333
----------
--------------------------------------------------
```

دقت تست و آموزس به ترتیب ۹۶ و ۹۷.۷ است و بدون بیش‌برازش به دقت خوبی در دسته‌بندی پلاک‌ها رسیده‌ایم.

برای هسته poly با انجام ۹ آزمایش بهترین دقت برای مقدار 10 پارامتر C و مقدار0.01 برای گاما به دست آمد.

```
C = 10
gamma = 0.01
test score 0.9666666666666667
best train score 0.96875
----------
--------------------------------------------------
```

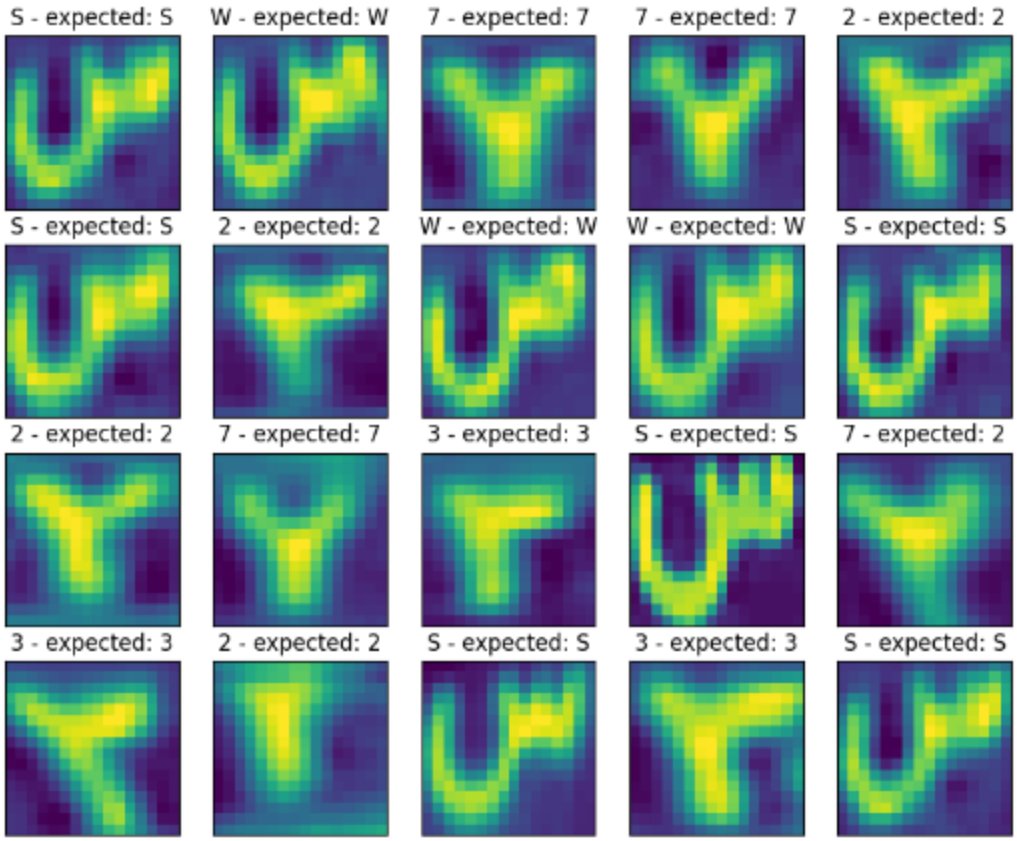
که دقت آموزش 96.8 و آزمایش 96.6 به دست آمده که بسیار مناسب و بدون بیش‌برازش است و در داده‌های تست بهتر از حالت بدون هسته عمل کرده است.

برای هسته rbf به ازای مقدار 10 برای C و مقدار 0.1 برای gamma به بهینه‌ترین حالت می‌رسیم.

```
C = 10
gamma = 0.1
test score 0.9833333333333333
best train score 0.9947916666666666
----------
```

که دقت 98.3 را برای داده‌های آزمایشی و دقت 99.4 را برای داده‌های آموزشی به دست داده است. این هسته با اختلاف عملکرد بهتری نسبت به هسته چندجمله‌ای و خطی دارد و در کل نشان می‌دهد این هسته انعطاف و قدرت زیادی برای حل مسائل دسته‌بندی دارد.

همچنین برای نمونه چندین داده تستی را به همراه مقادیر اصلی و پیش‌بینی‌شده نمایش داده و صحت آن را بررسی می‌کنیم.



در همه حالات مقدار پیش‌بینی با مقدار اصلی برابر است. به جز عکسی که در ستون اول از سمت راست و سطر سوم از بالا قرار دارد که مدل آن را ۷ پیش‌بینی کرده ولی مقدار مورد انتظار ما ۲ است. که البته از حق نگذریم تشخیصش خیلی سخت بود! مدل مجموعا با هر ۳ هسته به خصوص هسته rbf عملکرد بسیار مناسبی از خود روی این مدل نشان می‌دهد.

در این بخش تمرین یک نکته مهمی که استفاده کردیم و شرایط به ما اجازه می‌داد، آزمایش چندین مقدار محتمل مناسب برای پارامترهای تعیین کننده برای دستیابی به بهترین مقدار اولیه‌ای که بالاترین دقت را به ما بدهد است. این روش برای جستجوی مقدار مناسب پارامترهای به روش‌های بهتری هم قابل انجام است که البته اینجا نیاز نبود و سرچ روی همان ۳ مقدار C و ۳ مقدار gamma کفایت می کرد.

چالش‌ها

در این پروژه یک چالش خیلی اصلی مدت زمان زیاد اجرای هرباره یادگیری مخصوصا در بخش دوم بود که تست کردن و رفع اشتباهات و تغییر پارامترها را بسیار سخت و زمان‌بر می‌کرد و گاهی سیر رسیدن به یک مدل بهینه را برایم دشوار می‌کرد. با توجه به این که لپتاپم هم از لحاظ سخت‌افزاری کمی ضعف دارد این اجراها برایم مشکل‌ساز بود ولی به هر حال با کاهش تعداد داده‌های آموزشی به حدود ۱۰۰۰ تا این مشکل تا حدی برطرف شد که البته به قیمت کاهش دقت بود.

دیگر چالش من در بحث ساختن داده برای بخش اول تمرین بود که خوشبختانه با کمی سرچ برطرف شد.

یک چالش کوچک دیگر هم نحوه استفاده از دیتاست پلاک‌ها که در قالب یک فایل rar بود در numpy بود که خوشبختانه با کمی کار دستی و کد پایتون آن را به فرمت فشرده npz که مناسب نامپای است درآوردم و مساله حل شد. فایل npz را در کنار کد قرار داده‌ام و ارسال کردم تا هنگام اجرا مشکلی ایجاد نشود.

با تشکر از حوصله و توجه شما