بخش اول:

در این بخش از تابع svc برای اجرای الگوریتم svm استفاده کردیم. پارامتر های مورد بررسی مقادیر C, gamma kernel , بوده اند. گاما پارامتر مرتبط با کرنل در کرنل های سیگموید، چند جمله ای و rbf است. که در بررسی ما هر یک از این پارامتر ها مقادیر زیر را دارا هستند:

```
C_values = [0.1, 1, 10]
gamma_values = [0.1, 1, 10]
kernels = ['linear', 'rbf','poly','sigmoid']
```

این مقادیر را به ازای داده هایی با اشکال مختلف امتحان کردیم. که در خروجی کد، میزان دقت، بردارهای پشتیبان، خط جدا کننده و margin را میتوانید مشاهده کنید.
تعداد نمونه ها در هر حالت 500 هست.

ایجاد نقاط با make_blobs:

به دلیل بزرگ بودن شکل آن را در اینجا اضافه نکردیم اما در فایل Ipyn قابل مشاهده هست.
همانطور که انتظار میرفت نقاط اولیه ساده هستند و تقسیم بندی آنها به راحتی و با حاشیه بزرگی قابل انجام است. به ازای هر مقداری از C و gamma ، تقسیم بندی این نقاط با کرنل خطی با دقت 100% قابل انجام است. و همه اینها یک خط جدا کننده را به عنوان خروجی داده اند. اما باقی کرنل ها یعنی sigmoid , rbf , poly صرفا به ازای مقادیر کم پارامتر های دیگر خروجی خوبی داده اند. هسته rbf اما به ازای مقادیر بزرگتر از 0.1 برای c و بزرگتر از 1 برای گاما اورفیت میشود و با وجود دقت زیاد، به وضوح قابل تعمیم نییست.

ایجاد نقاط اولیه با make moons:

به وضوح تقسیم بندی با هسته خطی جواب درستی نمیدهد . با هر مقداری از c یک خروجی یکسان داریم.
هسته rbf میتواند جواب خوبی بدهد. به ازای c و گامای بزرگ هم مطابق شکل مدل اوفیت شده.
بهترین جواب اما به ازای هسته poly رخ میدهد. و برای گاما 10.
هسته سیگموید هم که کلا پرت هست.

ایجاد نقاط اولیه با make_circles:

بهترین هسته برای این شکل از داده ها هسته rbf است که به ازای هر مقداری از c و gammaی بزرگ تر از 0.1 به خوبی و با دقت 100% تقسیم بندی انجام میشود.

ایجاد نقاط اولیه با logical and:

کلاس هر نقطه طبق AND طول و عرض آن مشخص میشود. برای کرنل rbf, gamma=1 , c=10 دقت یک داریم اما مطابق شکل مدل اورفیت شده. به صورت کلی هسته خطی با پارامترهای c=1 , gamma=10 بهترین جواب را میدهد.

ایجاد نقاط اولیه با logical_xor:

کلاس هر نقطه طبق xor طول و عرض نقاط را مشخص میکند. به ازای پارمتر های زیر بهتری جواب با دقت 97درصد را داریم:

c=10 , gamma=1

بخش دوم:

لود کردن و پیش پردازش تصاویر:

در این بخش مطابق طبقه بندی پروژه شبکه عصبی از داده های fashion mnist استفاده کردیم. که از داده های کراس استفاده شده. کلا 70000 عکس داریم با 10 کلاس. که 60000 تای آنها در دسته آموزشی قرار دارند.

عکس ها را در ابتدا نرمال کردیم با تقسیم پیکسل ها بر 255 . سپس ماتریس های هر عکس را به صورت یک آرایه یک بعدی فلت کردیم.

آموزش مدل ها:

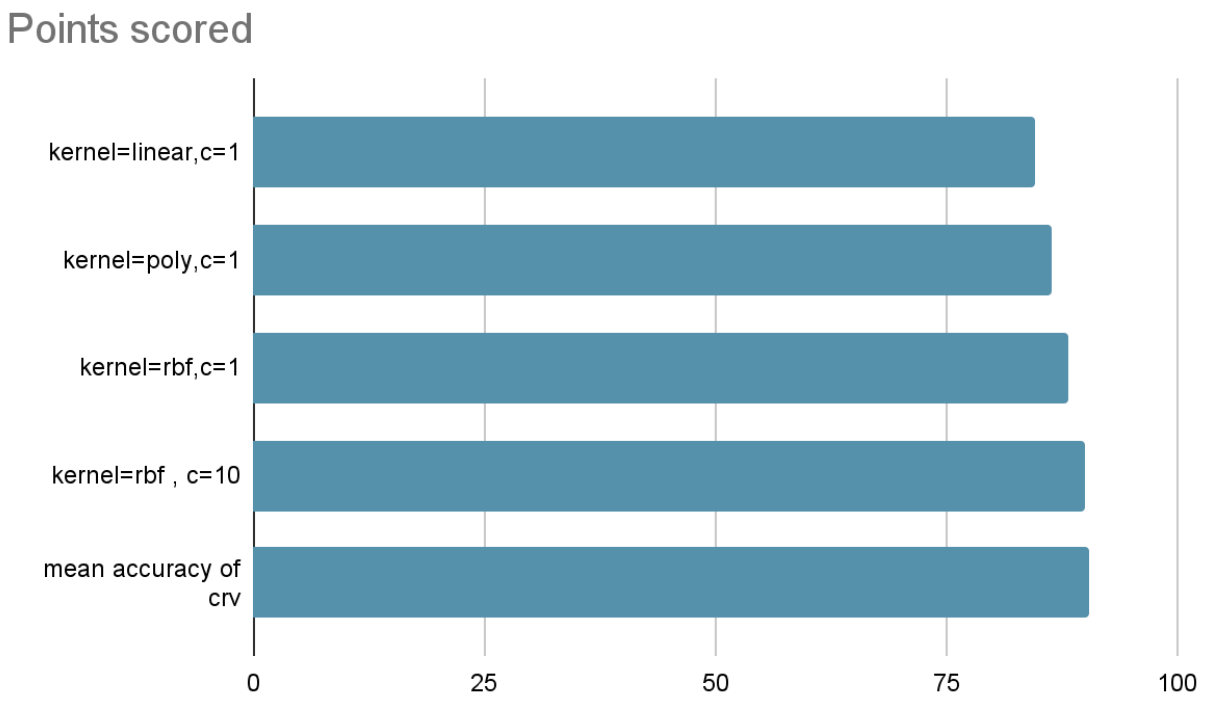
در این مرحله مدل هایی را با هسته های متفاوت آموزش دادیم. در انتها از کراس ولیدیشن هم استفاده کردیم که تعمیم پذیری مدل را بررسی کنیم که از این جهت قابل قبول است چرا مجموعه داده آموزشی بسیار زیاد است . این درحالیست که در شبکه عصبی خروجی که گرفتیم 86.9 درصد دقت داشت. در انتهای هر مدل تعدادی تصویر رندم از پایگاه داده با لیبل اصلی و پیش بینی شده آنها چاپ کردیم که با شهود بیشتری عملکرد مدل را ببینیم. مشکل اصلی آن برای افزایش دقت در تشخیص تیشرت از کت بوده که شباعت زیادی به هم دارند.

خروجی:

kernel=linear,c=1	84.64
kernel=poly,c=1	86.3
kernel=rbf,c=1	88.28
kernel=rbf , c=10	90.02

svm

mean accuracy of crv	90.5



بخش سوم:

چالش اصلی این بخش برای من در لود کردن داده ها در کگل بود.5 کلاس داشتیم که به هریک اعدادی از 0 تا 4 نسبت دادیم هرچند نوع لیبل در این روش svm اهمیتی ندارد. که از چه تایپی باشد. عکس ها و برچسب ها را در دو آرایه نامپای جدا ذخیره کردیم . روش های پیش پردازی بخش قبلی را روی این داده ها هم پیاده کردیم. به این ضورت که پیکسل ها را نرمال و سپس عکس ها را فلت کردیم. ابعاد تصویر ها 16 در 16 بوده که نسبت به تصاویر بخش قبلی کوچک تر بودند و سرعت آآموزش الگوریتم در این بخش بیشتر بود. در ضمن در هردو بخش تصاویر به صورت دیفالت در grayscale بودند و از این جهت این مرحله از پیش پردازش داده ها را انجام ندادیم اگر با عکس های رنگی مواجه بودیم باید این مرحله هم انجام میشد.

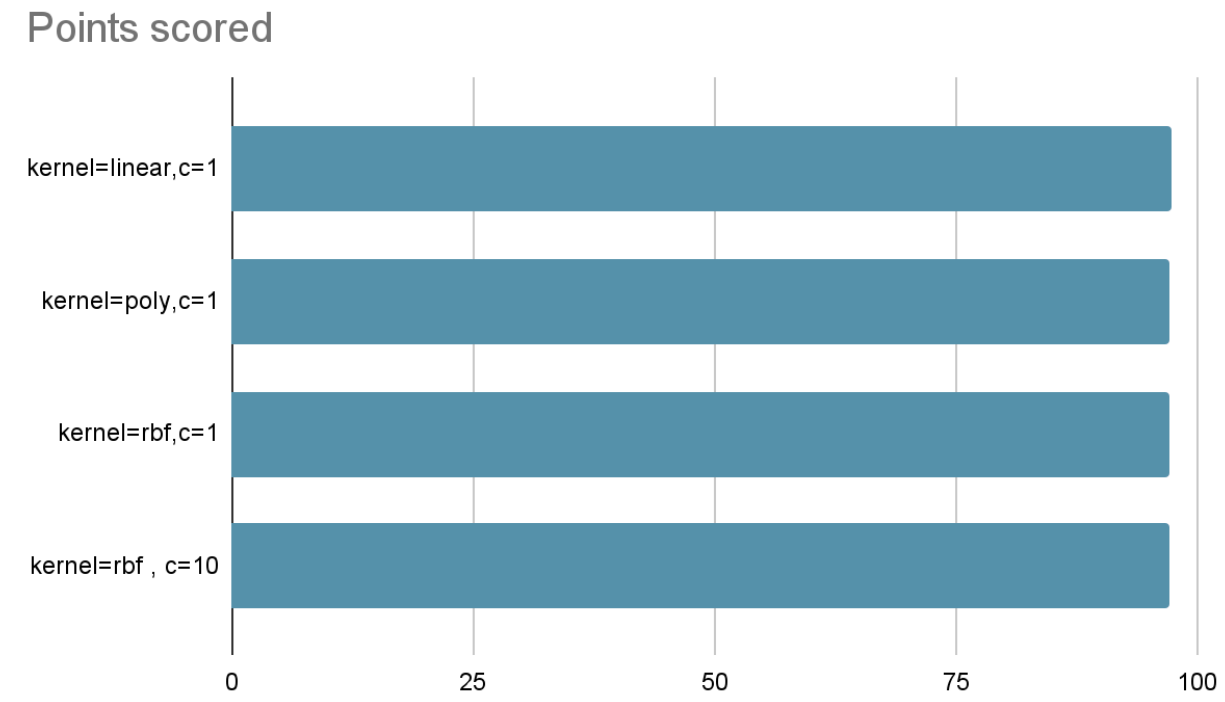
مانند مرحله قبل با هسته های مختلف آزمایش کردیم و خروجی های مختلفی از دقت مدل گرفتیم که در ادامه نشان داده خواهد شد.

مرحله خروجی هر مدل به ازای چند عکس رندم هم انجام شده که در فایل ipyn قابل مشاهده هست.

kernel=linear,	97.33

svm

c=1	
kernel=poly,c=1	97
kernel=rbf,c=1	97
kernel=rbf , c=10	97



چالش ها:

در ابتدا برای تقسیم بندی چند کلاسه به استفاده از بردار ویژه پشتیبان تصور کردم که کار سختی در پیش هست و باید یک الگوریتم را چند بار اجرا کرد. انتخاب اینکه از متد one vs one , one vs rest استفاده کنم تصمیم گیری سختی بود اما خوش شانس بودم که svc این موارد را هندل میکند و بنابراین پیچیدگی بخصوصی نداشت.