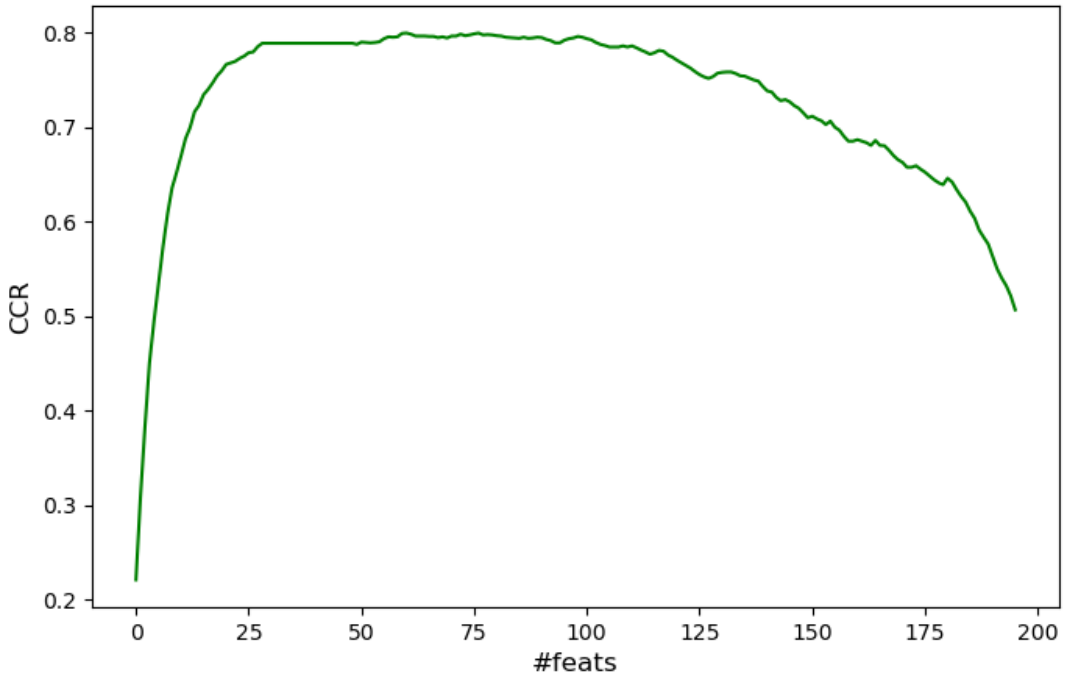
۱-من این دو الگوریتم رو بر روی کل دیتاست (یعنی هر ۱۹۶ feature) اجرا کردم. نتایج حاصل از اجرای هر یک به صورت زیر می‌باشد:

نتیجه‌ی Forward Selection:



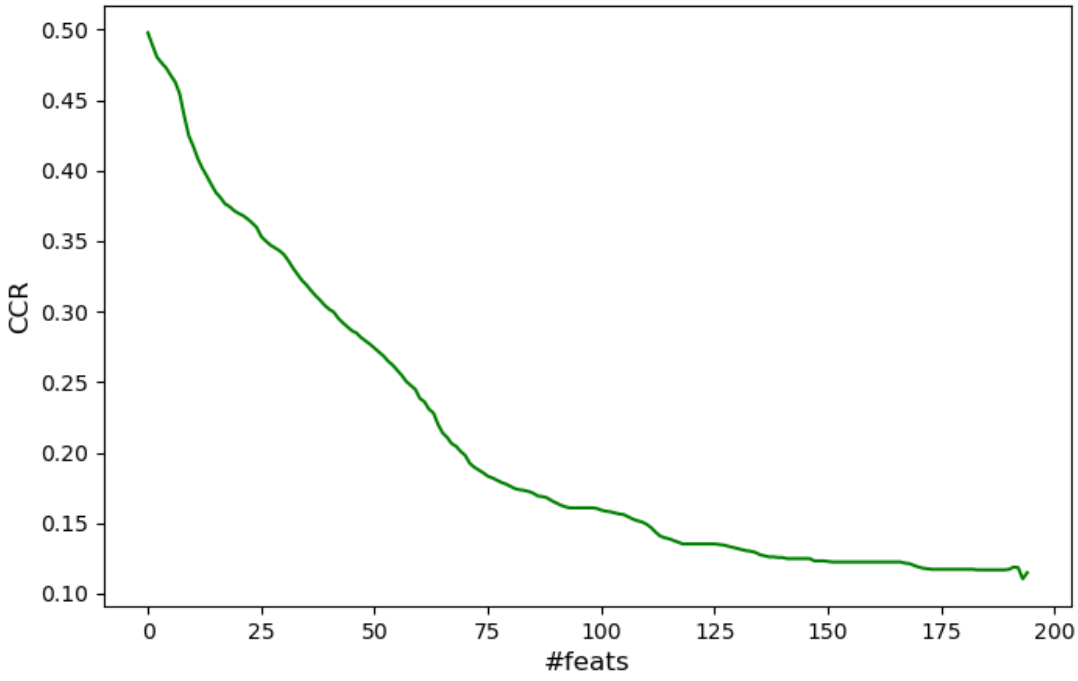
مقادیر CCR بیشینه و تعداد featureهای انتخابی که به ازای آن maximum CCR حاصل می‌شود به صورت زیر می‌باشد:

```
0.7996 61
```

یعنی میزان maximum CCR با استفاده از Forward Selection برابر 0.7996 بود و تعداد ویژگی‌هایی که با این الگوریتم منجر به این maximum CCR می‌شود برابر است با 61 ویژگی. و شماره‌ی ویژگی‌های انتخاب شده به صورت زیر می‌باشد:

```
[35, 102, 91, 106, 131, 89, 49, 104, 47, 65, 132, 63, 74, 133, 52, 162, 135,
76, 77, 78, 144, 149, 116, 61, 118, 64, 90, 75, 148, 0, 1, 2, 3, 12, 13, 14,
15, 27, 28, 84, 98, 154, 167, 168, 181, 182, 183, 194, 195, 120, 51, 134,
147, 80, 101, 103, 46, 60, 79, 48, 146]
```

نتیجه‌ی Backward Selection:

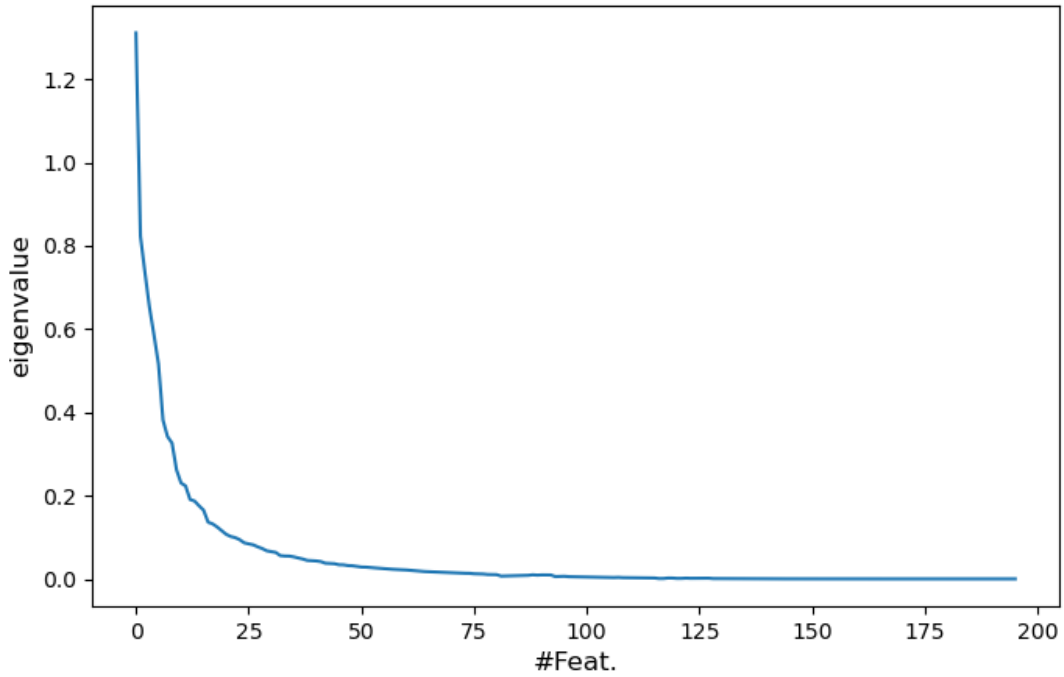



```
0.4976 196
```

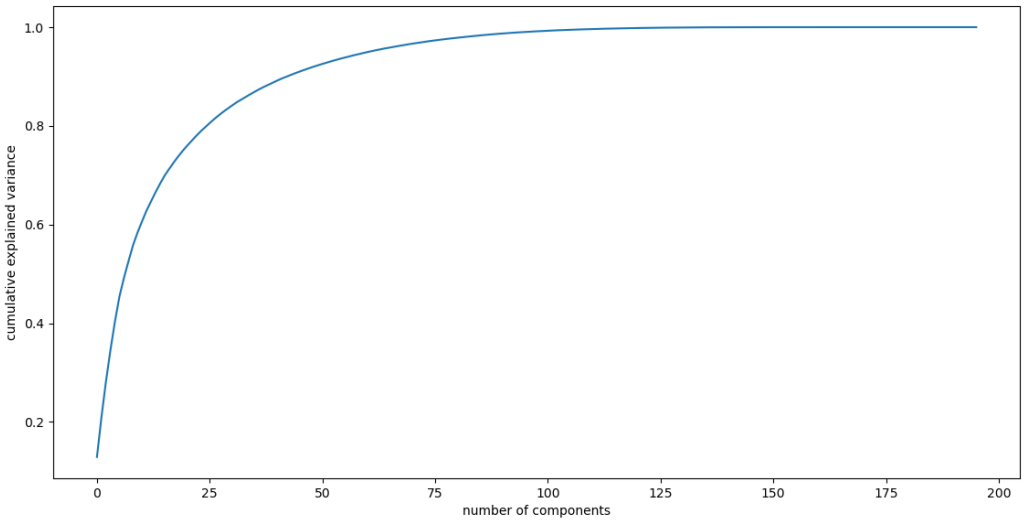
بیشترین میزان CCR برابر با 0.4976 می‌باشد که با انتخاب تمام ویژگی‌ها حاصل می‌شود. علت این تفاوت بارز در نتیجه‌ی FS و BS را می‌توان این گونه توضیح داد که: در backward selection مشکلی مشابه با مشکلی که در FS وجود داشت داریم. در BS پس از حذف ویژگی‌ها در مراحل اولیه در مراحل بعدی نمی‌توانیم ویژگی‌های حذف شده را reevaluate کنیم. و فقط می‌توانیم از مجموعه ویژگی‌های فعلی ویژگی کم کنیم.

۲- داده‌های test و train را split می‌کنیم. حال می‌خواهیم ماتریس کوواریانس را برای داده های train محاسبه کنیم. برای اینکار لازم است که داده‌ها centered شوند. یعنی میانگین هر ستون از ماتریس داده‌ها را محاسبه کرده(یعنی مقدار میانگین داده‌ها مربوط به هر feature) و سپس این ماتریس میانگین را از هر داده(هر سطر) کم می‌کنیم. با استفاده از تابع cov از پکیج numpy ماتریس کوواریانس را حساب کرده و با استفاده از تابع eig مقادیر و بردارهای ویژه‌ی این ماتریس را بدست می‌آوریم.

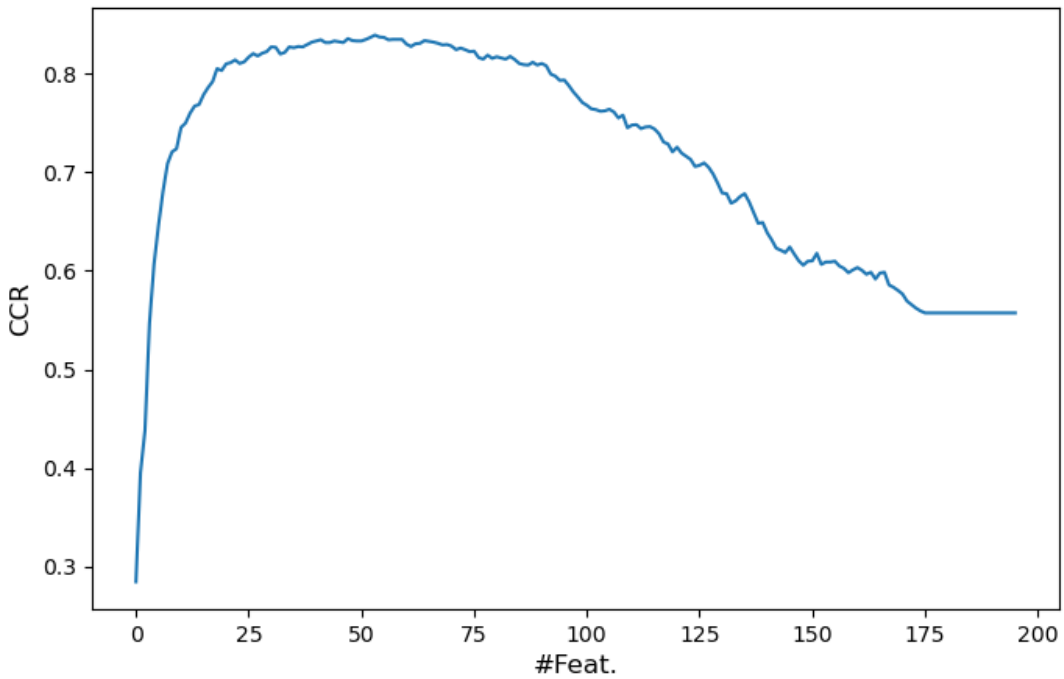
نمودار مقادیر ویژه ماتریس کواریانس بر حسب تعداد ویژگی به صورت زیر خواهد بود:



اما برای اینکه متوجه شویم که با چه تعداد ویژگی بیشترین مقدار ویژه حاصل می‌شود(در PCA به دنبال یافتن بردار ویژه‌ای هستیم که منتاظر با بیشترین مقدار ویژه می‌باشد)، باید نمودار explained variability تجمعی بر حسب تعداد ویژگی را رسم کنیم. این نمودار به ما می‌گوید که به ازای چه تعدادی از ویژگی‌ها بیشتر واریانس موجود در داده‌ها قابل توضیح است. این نمودار به صورت زیر خواهد بود:



به طور معمول درصد explained variance در حدود 80% کافی و مطلوب است. این میزان واریانس تجمعی با تعداد ویژگی حدود ۳۰ حاصل می‌شود. اما اگر بخواهیم این موضوع را با نمودار CCR بر حسب تعداد ویژگی بررسی کنیم، خواهیم داشت:



می‌بینیم که نتیجه‌گیری بالا با این نمودار هم همخوانی دارد. ما در حدود ۳۰ ویژگی، به مقدار حدود 0.8 برای CCR می‌رسیم. البته می‌بینیم که با تعداد حدوداً نزدیک 60 ویژگی بیشترین مقدار CCR را خواهیم داشت.

این نتایج به صورت شهودی از روی نمودارها بود. حال با استفاده از کد مقدار بیشینه CCR و تعداد ویژگی‌های بهینه را بدست می‌آوریم:

```
maximum CCR:  0.8388
optimum number of features:  53
```

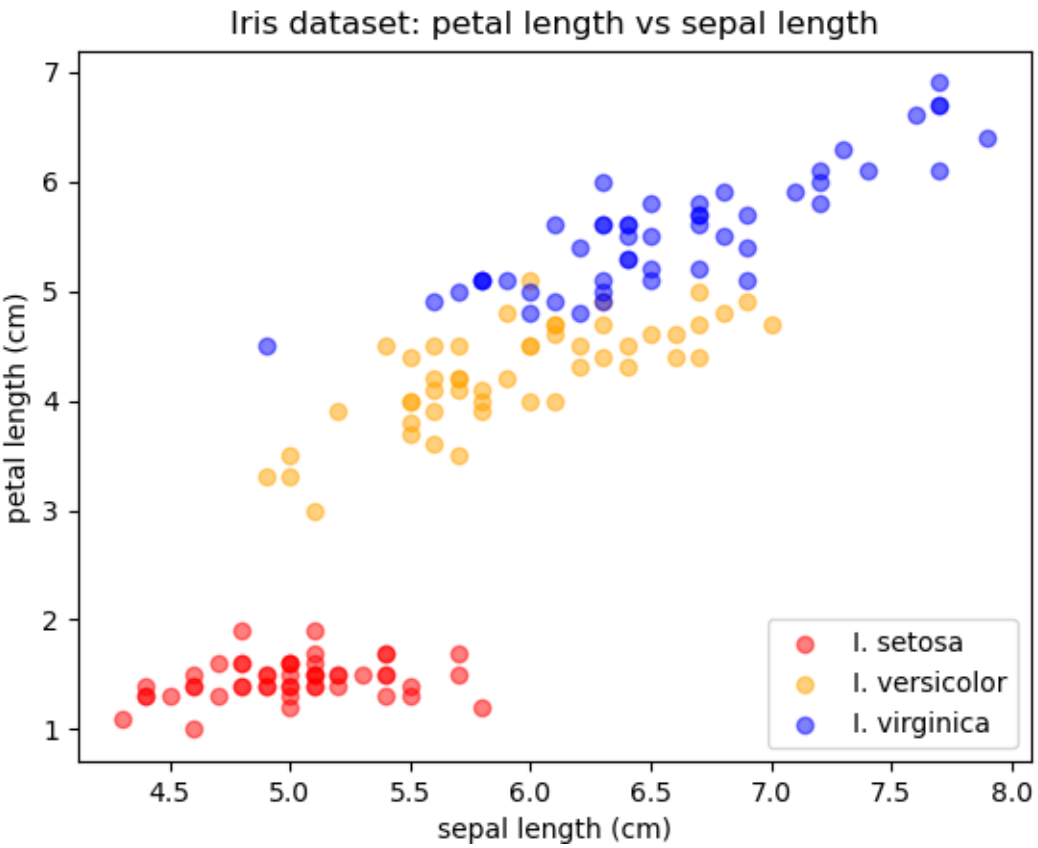
می‌بینیم که نتیجه‌ی بدست آمده برای تعداد بهینه ویژگی‌ها و بیشترین مقدار CCR با تحلیل شهودی ما همخوانی دارد. (تقریباً)

مقایسه با سوال ۱:

تعداد ویژگی‌های بدست آمده با الگوریتم Forward Selection(FS) در سوال ۱ برابر با 61 بوده که این تعداد ویژگی منجر به CCR= 0.7996 می‌شد. می‌بینیم که مقادیر به دست آمده برای CCR و تعداد بهینه‌ی ویژگی در این بخش اختلاف کمی با نتایج بدست آمده از FS دارند. دلیل این اختلاف می‌تواند این باشد که در الگوریتم FS ما امکان حذف ویژگی‌های انتخاب شده در مراحل قبل را نداریم. به عبارتی شاید مجموعه ویژگی های بهینه اصلا شامل تک ویژگی‌ای که منجر به بیشترین CCR نسبت به تک تک ویژگی‌های دیگر می‌شود نباشد. اما در FS ما حتما این ویژگی را در مرحله اول انتخاب می‌کنیم پس با اینکار امکان رسیدن به maximum CCR را به ازای مجموعه ویژگی‌های انتخابی خود می‌گیریم.

۸-

Scatter plot داده‌های کلاس‌های مختلف بر حسب دو ویژگی petal width وpetal length به صورت زیر می‌باشد:



الف) حال SVM را با kernelهای rbf وlinear و polynomial اعمال می‌کنیم و با استفاده از 10-fold cross validation نتایج آنها را نشان می‌دهیم(مقدار f1_score و ماتریس آشفتگی). در خروجی یک ماتریس سطری با اندازه ۳ داریم که درایه‌های آن نشان‌دهنده‌ی f1_score برای کلاس‌های ۰ و ۱ و ۲ می‌باشد.(به ترتیب کلاس های setosa وversicolor و virginica). و یک ماتریس ۳در۳ داریم که همان ماتریس آشفتگی می‌باشد.

برای linear kernel داریم:

```
linear
[1.   0.94 0.94]
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
```

برای polynomial kernel داریم:

```
poly
[1.   0.92 0.92]
[[50  0  0]
 [ 0 46  4]
 [ 0  4 46]]
```

برای rbf kernel داریم:

```
rbf
[1.         0.90196078 0.89795918]
[[50  0  0]
 [ 0 46  4]
 [ 0  6 44]]
```

در مورد این kernelها می‌توان گفت:

تفاوت این kernelها در ساخت decision boundary بین کلاس‌ها می‌باشد. به طور کلی کاربرد kernel functionها این است که دیتاست اصلی را به فضایی با تعداد ابعاد بیشتر ببرند به این امید که داده‌ها در آن فضای با بعد بیشتر خطی شوند. کاری که RBF kernel انجام می‌دهد این است که از یک ترکیب غیرخطی از ویژگی‌ها استفاده می‌کند تا داده‌ها را به یک فضای با ابعاد بزرگتر برده که بتوان داده‌ها را در آن فضا با linear decision boundary از هم جدا کرد. پس می‌توان نتیجه گرفت که linear kernel و polynomial kernel از نظر پیچیدگی زمانی نسبت به RBF به صرفه ترند اما دقت کمتری نیز دارند.

اما به طور کلی هیچ گارانتی‌ای وجود ندارد که یک kernel عملکرد بهتری از دیگری دارد. و باید بر اساس مسئله و دیتاست آن تصمیم بگیریم که کدام کرنل بهتر است و از آن استفاده کنیم.

برای دیتاست iris در این تمرین می‌بینیم که linear kernel عملکرد بهتری دارد.

ب) توضیح راجب پارامتر gamma و C:

پارامتر گاما مشخص می‌کند که اثر فاصله یک داده‌ی train تا چه حدی می‌تواند برسد. در واقع مقدار پارامتر گاما مشخص کننده‌ی میزان انحنایی است که می‌خواهیم decision boundary ما داشته باشد. مقادیر بالای گاما منجر به مدل‌های با بایاس بالا و واریانس کم (یعنی decision boundary با انحنای زیاد) و مقادیر پایین گاما منجر به مدل‌های با بایاس کم و واریانس زیاد (یعنی decision boundary با انحنای کم) می‌شود.

پارامتر C یک جریمه به ازای هر misclassified data point اضافه می کند. اگر مقدار C کم باشد، جریمه به ازای نقاط به اشتباه طبقه‌بندی شده نیز کم خواهد بود در نتیجه یک مرز تصمیم با مقدار margin بزرگتر انتخاب می‌شود اما در ازای آن تعداد نقاط به اشتباه طبقه‌بندی شده(misclassified points) بیشتر خواهد شد. اما اگر مقدار C بزرگ باشد، SVM سعی می‌کند تعداد نقاط به اشتباه طبقه‌بندی شده را کمینه کند. چرا که به ازای مقادیر بالای C، جریمه به

ازای نقاط به اشتباه طبقه‌بندی شده زیاد خواهد بود و درنتیجه decision boundary حاصل مقدار margin کوچکتری دارد.

Gamma:

خروجی(ماتریس سطری از f1_scoreها که در سوال ۲ هم توضیح داده شد و ماتریس آشفتگی) به ازای مقادیر مختلف گاما. در خروجی ابتدا مقدار گاما چاپ شده و سپس f1_score کلاس‌ها و بعد هم ماتریس آشفتگی:

```
0.01
[1.   0.72 0.72]
[[50  0  0]
 [ 0 36 14]
 [ 0 14 36]]
```

```
0.1
[1.         0.90196078 0.89795918]
[[50  0  0]
 [ 0 46  4]
 [ 0  6 44]]
```

```
1
[1.   0.92 0.92]
[[50  0  0]
 [ 0 46  4]
 [ 0  4 46]]
```

```
10
[0.94736842 0.88888889 0.8490566 ]
[[45  0  5]
 [ 0 44  6]
 [ 0  5 45]]
```

```
100
[0.41269841 0.04477612 0.11650485]
[[13 37  0]
 [ 0  3 47]
 [ 0 44  6]]
```

با افزایش مقدار Gamma از 0.01 به 1 عملکرد طبقه‌بند هم بهتر می‌شود اما با فاصله گرفتن زیاد آن از مقدار 1 (یعنی در Gamma= 100 و Gamma= 10) عملکرد طبقه‌بند کاهش می یابد که علت آن (همانطور که در توضیحات پارامتر Gamma در بالا اشاره کردم) overfit شدن مدل ما به داده‌ی train می باشد.

C:

خروجی(ماتریس سطری از f1_scoreها که در سوال ۲ هم توضیح داده شد و ماتریس آشفتگی) به ازای مقادیر مختلف C. در خروجی ابتدا مقدار C چاپ شده و سپس f1_score کلاس‌ها و بعد هم ماتریس آشفتگی:

```
0.01
[0. 0. 0.]
[[ 0 45  5]
 [12  0 38]
 [ 5 45  0]]
```

```
0.1
[1.         0.48076923 0.4375    ]
[[50  0  0]
 [ 0 25 25]
 [ 0 29 21]]
```

```
1
[1.         0.90196078 0.89795918]
[[50  0  0]
 [ 0 46  4]
 [ 0  6 44]]
```

```
10
[1.         0.93069307 0.92929293]
[[50  0  0]
 [ 0 47  3]
 [ 0  4 46]]
```

```
100
[1.         0.92783505 0.93203883]
[[50  0  0]
 [ 0 45  5]
 [ 0  2 48]]
```

با افزایش مقدار c از 0.01 به 10 عملکرد طبقه‌بند هم بهتر می‌شود اما با فاصله گرفتن زیاد آن از مقدار 10 (یعنی در C= 100) عملکرد طبقه‌بند کاهش می یابد که علت آن (همانطور که در توضیحات پارامتر C در بالا اشاره کردم) overfit شدن مدل ما به داده‌ی train می باشد.

ج) نتیجه‌ی grid search با استفاده از 10-fold به صورت زیر می‌باشد:

```
Best parameters from gridsearch: {'C': 1, 'gamma': 0.1, 'kernel': 'poly'}
CV score=0.980
{'C': 1, 'gamma': 0.1, 'kernel': 'poly'}
[1.         0.93877551 0.94117647]
[[50  0  0]
 [ 0 46  4]
 [ 0  2 48]]
```

بهترین مقادیر گزارش شده توسط grid search برای پارامترها و همچنین نوع kernel در خروجی بالا قابل مشاهده است. می‌بینیم که با انتخاب polynomial kernel و مقدار C = 1 و Gamma = 0.1 برای پارامترها بیشترین عملکرد مدل محقق می‌شود. همانطور که در ماتریس آشفتگی مشخص است تنها ۶ داده به اشتباه طبقه‌بندی شده اند.

د) نتایج روش one vs. all برای تک تک کرنل‌ها به صورت زیر می‌باشد:

```
linear
[0.98989899 0.8125     0.83809524]
[[49  1  0]
 [ 0 39 11]
 [ 0  6 44]]
```

```
poly
[1.         0.94949495 0.95049505]
[[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]
```

```
rbf
[1.         0.90384615 0.89583333]
[[50  0  0]
 [ 0 47  3]
 [ 0  7 43]]
```

می‌بینیم که با روش one vs. all، کرنل polynomial بهترین عملکرد را دارد.

نتایج روش one vs. one برای تک تک کرنل‌ها به صورت زیر می‌باشد:

```
linear
[1.   0.94 0.94]
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
.................................
poly
[1.         0.92929293 0.93069307]
[[50  0  0]
 [ 0 46  4]
 [ 0  3 47]]
.................................
rbf
[1.         0.90196078 0.89795918]
[[50  0  0]
 [ 0 46  4]
 [ 0  6 44]]
.................................
```

می‌بینیم که با روش one vs. one، کرنل linear بهترین عملکرد را دارد.

همانطور که در پاسخ سوالات تشریحی اشاره کردم، امکان ایجاد عدم تقارن در داده‌های کلاس‌های اقلیت(کلاس‌های با داده‌ی کم)، توسط one vs. one classifier خیلی کم است. در نتیجه انتظار می‌رود که این روش عملکرد بهتری نسبت به روش one vs. all داشته باشد (هر چند کندتر است).