$$J = \frac{1}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_j \in Y_2} (y_i - y_j)^2$$

$$\rightarrow J = \frac{1}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_j \in Y_2} \left[ (y_i - m_1) - (y_j - m_2) + (m_1 - m_2) \right]^2 =$$

$$\frac{1}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_j \in Y_2} \left[ (y_i - m_1)^2 + (y_j - m_2)^2 + (m_1 - m_2)^2 - 2(y_i - m_1)(y_j - m_2) \right.$$

$$\left. + 2(y_i - m_1)(m_1 - m_2) - 2(y_j - m_2)(m_1 - m_2) \right] =$$

$$\frac{1}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_j \in Y_2} (y_i - m_1)^2 + \frac{1}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_j \in Y_2} (y_j - m_2)^2 + \frac{n_1 n_2}{n_1 n_2} (m_1 - m_2)^2$$

جمع اضلاع از سیانتین ۰ است

$$\underbrace{- \frac{1}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_j \in Y_2} 2(y_i - m_1)(y_j - m_2)}_{0} + \underbrace{\frac{(m_1 - m_2)}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_i \in Y_2} 2(y_i - m_1)}_{0} \quad \rightarrow \sum (x - \bar{x}) = 0$$

$$\underbrace{- \frac{(m_1 - m_2)}{n_1 n_2} \sum_{y_i \in Y_1} \sum_{y_j \in Y_2} 2(y_j - m_2)}_{0} = \frac{1}{n_1 n_2} n_2 s_1^2 + \frac{1}{n_1 n_2} n_1 s_2^2 + (m_1 - m_2)^2 =$$

$$\boxed{\frac{1}{n_1} s_1^2 + \frac{1}{n_2} s_2^2 + (m_1 - m_2)^2}$$

الف) مطابق تعریف روبه رو می توان دید که $S_B$ از جمع $c$ بردار به دست آمده است بنابراین می توان گفت ماتریس رنک آن می تواند $c$ باشد اما نکته ای که وجود دارد این است که تمام این $c$ بردار از هم مستقل نیستند چرا که معادله ی دیگری به شکل زیر نیز وجود دارد:

$$S_B = \sum_{K=1}^{c} N_K (\mu_K - \mu)(\mu_K - \mu)^T$$

$$\mu = \frac{1}{N} \sum_{K=1}^{c} N_K \mu_K$$

این معادله بیان می کند که می توانیم یکی از $\mu_K$ ها با داشتن بقیه حساب کنیم و داریم:

$$N_j \mu_j = N\mu - \sum_{\substack{K=1 \\ K \neq j}}^{c} N_K \mu_K$$

← بنابراین حداقل می توان یکی از اعضا را بر حسب بقیه نوشت

← $Rank(S_B) \leq c-1$

درصورتی که $\mu_i$ ها مستقل از هم باشند دین آنگاه داشتن $c-1$ وجود داشته باشد: $Rank(S_B) = c-1$

ب)

$$Rank(S_\omega^{-1} S_B) \leq \min\{Rank(S_\omega^{-1}), Rank(S_B)\}$$

$$Rank(S_B) \leq c-1$$

$$\rightarrow \boxed{Rank(S_\omega^{-1} S_B) \leq c-1}$$

$S_\omega = \sum_{i=1}^{c} S_i$ ← چون معلوم شده است پس نول رنک بوده و رنک آن همان $c$ است

ج)

$$S_T = S_\omega + S_B$$

$$S_T = \sum_{x} (x-m)(x-m)^t =$$

$$\sum_{i=1}^{c} \sum_{x \in D_i} (x - m_i + m_i - m)(x - m_i + m_i - m)^t =$$ مجموع ماتریس بیانگین برابر ۰ است

$$\sum_{i=1}^{c} \sum_{x \in D_i} (x-m_i)(x-m_i)^t + \sum_{i=1}^{c} \sum_{x \in D_i} (m_i - m)(m_i - m)^t + \sum_{i=1}^{c} \cancel{\sum_{x \in D_i} (x-m_i)(m_i-m)^t}^{0}$$

$$+ \sum_{i=1}^{c} \cancel{\sum_{x \in D_i} (m_i - m)(x-m_i)^t}^{0} = S_\omega + \sum_{i=1}^{c} n_i (m_i - m)(m_i - m)^t = \boxed{S_\omega + S_B}$$

(6)

الف)

در روش generative تابع احتمال توام فضای ویژگی و کلاس $p(x, w_i)$ را به دست آورده و سپس از روی آن توابع شرطی $(p(w_j|k))$ را جهت تعریف مدل به دست می‌آوریم. مشکل این روش این است که برای به دست آوردن تابع توزیع توام تعداد نمونه زیادی می خواهیم و با مشکل کمبود نمونه مواجه هستیم.

در روش discriminative یک فرم توابع شرطی $(p(w_j|k))$ را تخمین می‌زنیم. در این روش اطلاعات کمتری داریم ولی برای ساخت مدل به تعداد نمونه‌های کمتری نیاز است و مدل به دست آمده ساده‌تر خواهد بود.

ب)

one vs one

برای به دست آوردن مرز تصمیم ساده‌ترین حالت دسته‌بندی c تا کلاس‌های دوتایی می‌بینیم به این صورت که هر بار یک کلاس را در مقایسه با سایر کلاس‌ها در نظر می‌گیریم. ایرادی‌های این روش این است که نقاط مبهم زیادی دارد

one vs rest

در این روش دوبه‌دو تمام pair های موجود را در نظر می‌گیریم و مرزها را مشخص می‌کنیم. ایرادی‌های این روش این است که به تعداد $\frac{c(c-1)}{2}$ دسته‌بندی باید انجام داد که از روش قبل بیشتر است و زمان طولانی‌تری می‌برد اما فضاهای مبهم به وجود آمده در این روش از روش قبلی کمتر است.

linear machine

این روش از discriminant function ها استفاده می‌کند و هر داده را به کلاسی نسبت می‌دهد که بیشترین امتیاز را به دست آورده است. از مزیت‌های این روش این است که ناحیه تصمیم آن convex است و معمولا برای مسائلی استفاده می‌شود که conditional density در آن unimodal باشد

در این روش اگرچه $\frac{c(c-1)}{2}$ جهت از نواحی وجود دارد اما معمولا مرزهای تصمیم کمتری داریم و نقاط مبهم نیز در آن وجود ندارد

ج)

در مسئله primal به ازای هر دوری تعداد کمینه برای w پیدا می‌شود و برای دسته بندی لازم است تا $w^t x$ به طور مستقیم پیدا شود، در صورتی که ابعاد مسئله بالا باشد محاسبات این روش بسیار زمان بر خواهد بود.

در مسئله dual پس از پیدا کردن $a_i$ها $w^t x$ پیدا می‌شود و از آن جایی که $a_i$ها در تمام نقاط به جز supporting vector ها صفر هستند محاسبات به شدت ساده خواهد شد. در واقع این روش هنگامی که supporting vector های کمی داریم بسیار سریع تر از روش اول عمل می‌کند. علاوه بر این مسئله dual هموار کردن را نیز دارا می‌کند و می‌توان مسائل غیر خطی را نیز با استفاده به ابعاد جدید حل کرد.

(7)

الف)

$$k(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle$$

$$\|\varphi(x_i) - \varphi(x_j)\|^2 = \langle \varphi(x_i), \varphi(x_i) \rangle + \langle \varphi(x_j), \varphi(x_j) \rangle - 2\langle \varphi(x_i), \varphi(x_j) \rangle =$$

$$k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j) = 1 + 1 - 2\exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right) < 2$$

ب)

5) $k(x, y) = \exp(k_1(x, y))$

$\rightarrow \exp(k_1(x,y)) = \sum_{i=0}^{\infty} \frac{1}{i!} (k_1(x,y))^i \rightarrow$ چند جمله‌ای با ضرایب مثبت به کرنل اعمال شده و نیز هم چنان مثبت معین باقی خواهد ماند

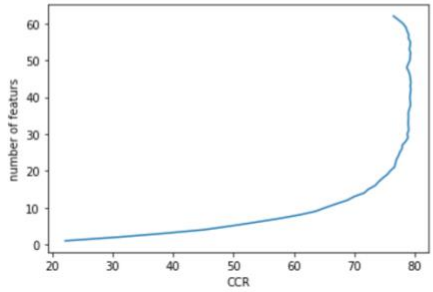
۱)

## forward selection

برای اجرای این الگوریتم ابتدا  از یک ست خالی ویژگی ها شروع کرده و در هر مرحله یک ویژگی به آن اضافه می کنیم. از میان ویژگی های اضافه شده ویژگی انتخاب می شود که بهترین score را داشته باشد. برای محاسبه ی score نیز از CCR استفاده می شود به این صورت که هر چه CCR به دست آمده برای آن مجموع ویژگی بیشتر باشد برای ما انتخاب بهتری است. این روند تا آن جا ادامه می یابد که تمام ویژگی ها اضافه شده باشند.
نمودار به دست آمده به صورت زیر است:

```
In [458]: x = [i for i in range(1,feature_size+1)]
          plt.plot(ccr,x)
          plt.ylabel('number of featurs')
          plt.xlabel('CCR')
          print("num ",ccr.index(max(ccr))+1)
          print("ccr",max(ccr))
```

```
num  42
ccr 79.36
```



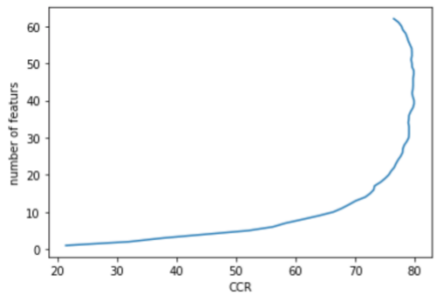

همان طور که مشاهد می شود با داشتن ۴۲ ویژگی می توان به بالاترین دقت رسید.

## Backward elimination

این روش نیز تشابه زیادی با روش قبل دارد با این تفاوت که در ابتدا از تمام ویژگی ها شروع کرده و در هر مرحله ویژگی ای حذف می شود که با حذف آن بیشترین امتیاز به دست آید.
نمودار به دست آمده پس از اجرای این الگوریم مطابق شکل زیر است:

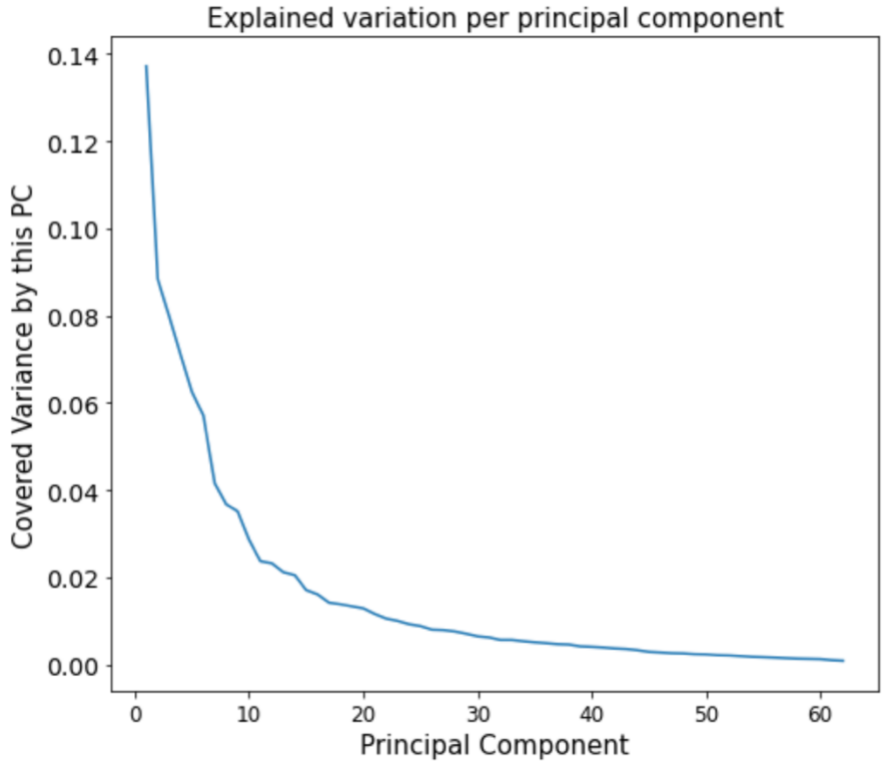
```
In [460]: x = [i for i in range(1,feature_size+1)]
          ccr.reverse()
          plt.plot(ccr,x)
          plt.ylabel('number of featurs')
          plt.xlabel('CCR')
          print("num ",ccr.index(max(ccr))+1)
          print("ccr",max(ccr))
```

```
num  39
ccr 79.88
```




در این روش با مجموع ۳۹ ویژگی می توان به بالاترین دقت رسید.

۲)
در این سوال هدف پیدا کردن جهت هایی است که ویژگی های مورد نظر بیشترین واریانس در آن راستا را داشته باشند. همچنین باید به این نکته توجه داشت که مقادیر ویژه در واقع بیان کننده میزان واریانس در راستای بردار ویژه است.نمودار به دست آمده برای ۶۲ کامپوننت به صورت زیر است:

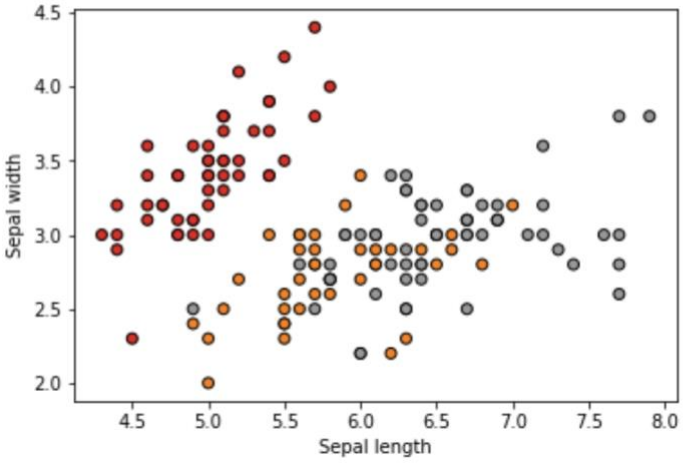



مشاهده می شود که که شیب نمودار تقریبا از ۲۵ کامپوننت به بعد بسیار کاهش یافته است و می توان گفت از این نقطه به بعد تغییرات اضافه شده بسیار کم و قابل چشم پوشی است. با اجرای pca و انتقال داده ها به بعد جدید به دقت خوبی میرسیم.

```
In [87]: optimal_component = 25
pca_mnist = PCA(n_components = optimal_component)

transformed_train = pca_mnist.fit_transform(df_train_data)
transformed_test = pca_mnist.transform(df_test_data)

clf = GaussianNB()
clf.fit(transformed_train, train_labels)
y_pred = clf.predict(transformed_test)
CCR = accuracy_score(test_labels, y_pred)*100
print(CCR)

79.12
```

مشاهده می شود که دقت با تنها 25 کامپوننت به دقت قسمت قبل نزدیک شده است که نشان می دهد این روش با ترکیب ویژگی های مختلف قدرت بسیار خوبی دارد و با ویژگی های کمتر می توان به دقت مشابه رسید.

۸)
داده های هر کلاس بر اساس دو ویژگی sepal width و sepal length به صورت زیر است:




a)
تصویر زیر فرمول های مربوط به کرنل های مختلف را نشان می دهد:

$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$
*Polynomial kernel equation*

$k(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$
*Gaussian kernel equation*

$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma\|\mathbf{x_i} - \mathbf{x_j}\|^2)$
*Gaussian radial basis function (RBF)*

$k(x, y) = \tanh(\alpha x^T y + c)$
*Sigmoid kernel equation*

$k(x, y) = \exp\left(-\frac{\|x-y\|}{\sigma}\right)$
*Laplace RBF kernel equation*

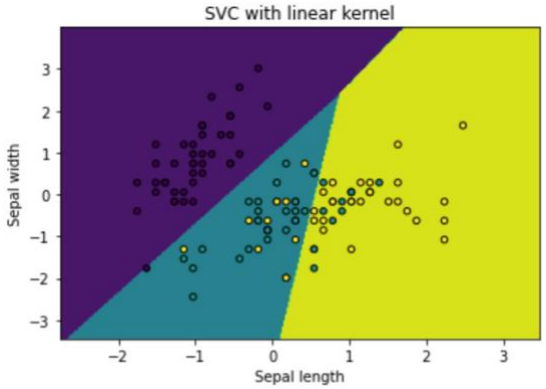
$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa\mathbf{x_i} \cdot \mathbf{x_j} + c)$
*Hyperbolic tangent kernel equation*

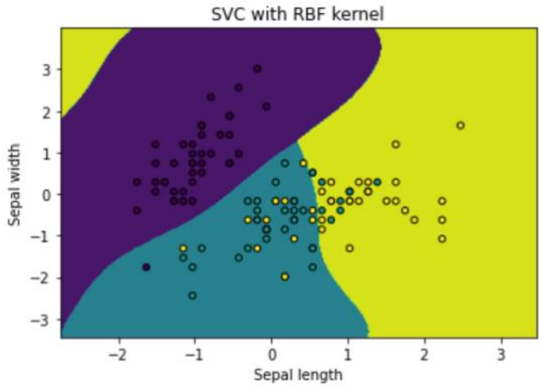
- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma\langle x, x' \rangle + r)^d$, where $d$ is specified by parameter `degree`, $r$ by `coef0`.
- rbf: $\exp(-\gamma\|x - x'\|^2)$, where $\gamma$ is specified by parameter `gamma`, must be greater than 0.

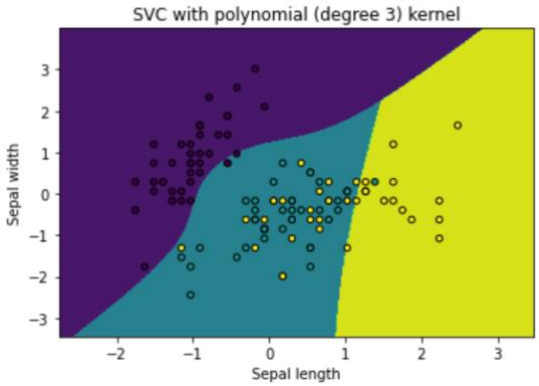
به کمک کرنل چند جمله ای می توان فضا را تا d بعد افزایش داد. این کرنل در پردازش تصویر بسیار کاربرد دارد و از آن استفاده می شود. این کرنل به ما اجازه می دهد که مدل های غیر خطی را نیز یاد بگیریم.
کرنل خطی زمانی خوب عمل می کند که داده ها به صورت خطی جدا پذیر باشند. علاوه بر این هنگامی که سرعت بالایی احتیاج داریم این کرنل می تواند به خوبی عمل کند.
کرنل rbf برای اهداف عمومی کاربرد دارد و هنگامی که هیچ دانش پیشینی در مورد داده ها نداریم معمولا از این کرنل استفاده می کنیم.

*توجه شود که در تمام مراحل موجود، داده ها در ابتدا استاندار شده اند.

نتایج حاصل از این طبقه بند ها به صورت زیر است:




SVC with linear kernel:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.83 | 0.83 | 0.83 | 12 |
| 2 | 0.80 | 0.80 | 0.80 | 10 |
| accuracy | | | 0.87 | 30 |
| macro avg | 0.88 | 0.88 | 0.88 | 30 |
| weighted avg | 0.87 | 0.87 | 0.87 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 10  2]
 [ 0  2  8]]
```

SVC with RBF kernel:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.85 | 0.92 | 0.88 | 12 |
| 2 | 0.89 | 0.80 | 0.84 | 10 |
| accuracy | | | 0.90 | 30 |
| macro avg | 0.91 | 0.91 | 0.91 | 30 |
| weighted avg | 0.90 | 0.90 | 0.90 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  2  8]]
```

SVC with polynomial (degree 3) kernel:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.58 | 0.92 | 0.71 | 12 |
| 2 | 0.67 | 0.20 | 0.31 | 10 |
| accuracy | | | 0.70 | 30 |
| macro avg | 0.75 | 0.71 | 0.67 | 30 |
| weighted avg | 0.72 | 0.70 | 0.65 | 30 |

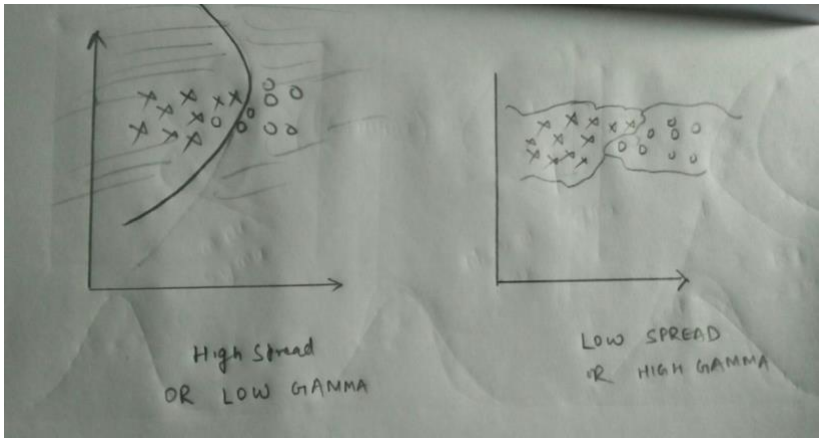
confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  8  2]]
```

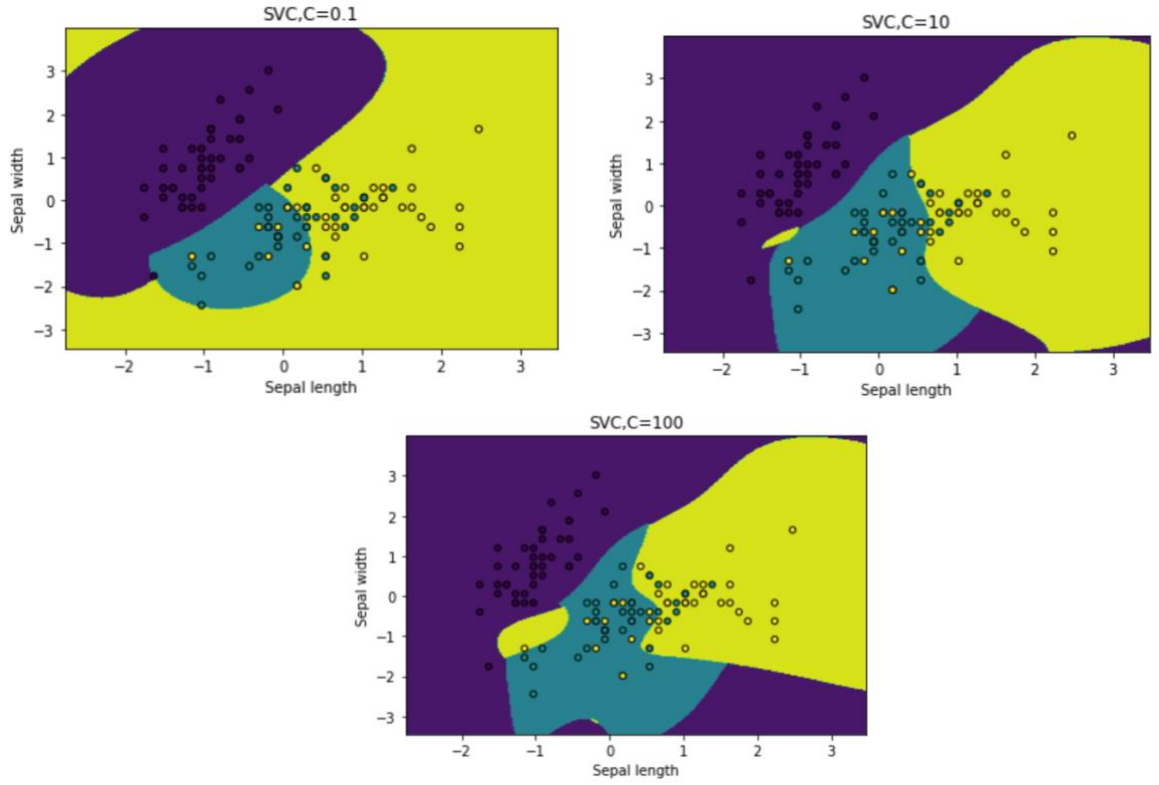
b)

از C برای کنترل خطا استفاده می شود. در واقع C بیان می کنیم که خطا در طبقه بندی به چه میزان برای ما هزینه دارد. هر چه این مقدار بیشتر باشد یعنی اجازه ی خطای کمتری داریم و به دنبال طبقه بندی صحیح بر روی داده های آموزشی هستیم به این معنی که مرز ها با سخت گیری بیشتری انتخاب می شوند.

از پارامتر گاما برای وزن دادن به انحنای مرز تصمیم استفاده می شود. هرچه مقدار گاما بیشتر باشد انحنای منحنی ها بیشتر است.




شکل زیر نتایج حاصل از امتحان سه C مختلف را نشان می دهد:



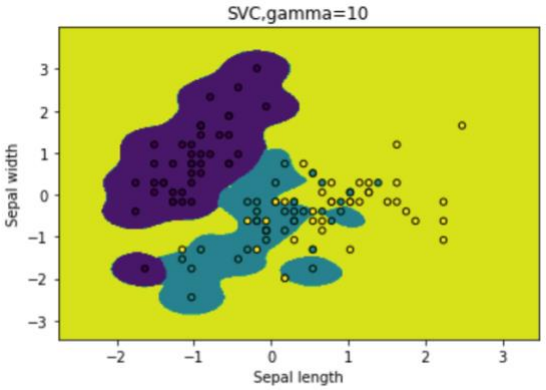
```
SVC,C=0.1:
classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      0.83      0.91        12
           2       0.83      1.00      0.91        10

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.94      0.93      0.93        30

confusion matrix:
[[ 8  0  0]
 [ 0 10  2]
 [ 0  0 10]]
```

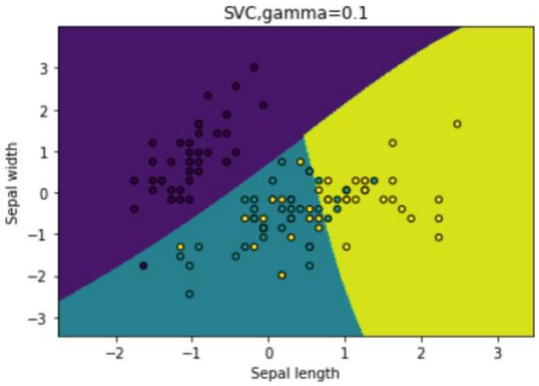
```
SVC,C=10:
classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.85      0.92      0.88        12
           2       0.89      0.80      0.84        10

    accuracy                           0.90        30
   macro avg       0.91      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30

confusion matrix:
[[ 8  0  0]
 [ 0 11  1]
 [ 0  2  8]]
```

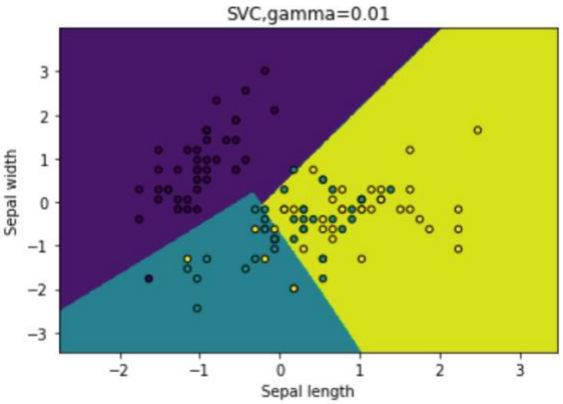
```
SVC,C=100:
classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.83      0.83      0.83        12
           2       0.80      0.80      0.80        10

    accuracy                           0.87        30
   macro avg       0.88      0.88      0.88        30
weighted avg       0.87      0.87      0.87        30

confusion matrix:
[[ 8  0  0]
 [ 0 10  2]
 [ 0  2  8]]
```

همان طور که مشاهده می شود با افزایش C سخت گیری بیشتری در انتخاب مرز تصمیم شده است و همین موضوع باعث شده مدل کمی بر روی داده های آموزشی فیت شود و برای داده های تست به دقت کمتری برسد.

شکل زیر نتایج حاصل از امتحان سه gamma مختلف را نشان می دهد:




SVC,gamma=10:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.92 | 0.92 | 0.92 | 12 |
| 2 | 0.90 | 0.90 | 0.90 | 10 |
| accuracy | | | 0.93 | 30 |
| macro avg | 0.94 | 0.94 | 0.94 | 30 |
| weighted avg | 0.93 | 0.93 | 0.93 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  1  9]]
```

SVC,gamma=0.1:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.73 | 0.92 | 0.81 | 12 |
| 2 | 0.86 | 0.60 | 0.71 | 10 |
| accuracy | | | 0.83 | 30 |
| macro avg | 0.86 | 0.84 | 0.84 | 30 |
| weighted avg | 0.85 | 0.83 | 0.83 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  4  6]]
```

SVC,gamma=0.01:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 1.00 | 0.92 | 0.96 | 12 |
| 2 | 0.91 | 1.00 | 0.95 | 10 |
| accuracy | | | 0.97 | 30 |
| macro avg | 0.97 | 0.97 | 0.97 | 30 |
| weighted avg | 0.97 | 0.97 | 0.97 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  0 10]]
```

همان طور که پیش تر نیز گفته شد مشاهده می شود که با افزایش گاما انحنای های مرز تصمیم بیشتر می شود و اطراف داده های آموزش محدود می شود.

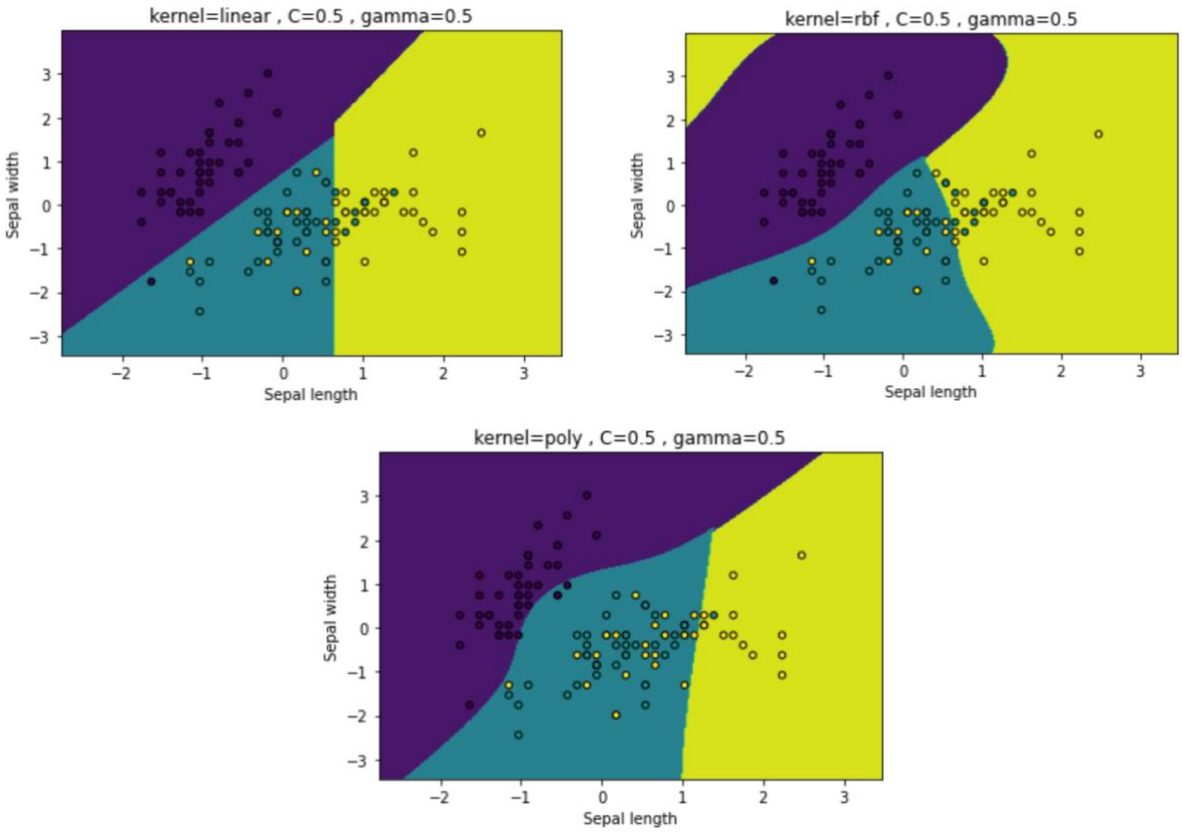
(c

```
from sklearn.model_selection import GridSearchCV

parameteres = {'kernel':['linear','rbf','poly'],'C':[0.5,1,10], 'gamma':[0.5,1,10]}
grid = GridSearchCV(svm.SVC(), param_grid=parameteres)
grid.fit(x_train, y_train)

print ('accuracy of the model is:',grid.score(x_test,y_test))
print ('best parameters of the model are:',grid.best_params_)
```

```
accuracy of the model is: 0.8666666666666667
best parameters of the model are: {'C': 0.5, 'gamma': 0.5, 'kernel': 'linear'}
```

با اعمال gridSearch می بینم که بهترین پارامتر ها مربوط به کرنل خطی با گاما و سی برابر 0.5 بوده است البته باید دقت داشت که لزوما اعداد به دست آمده بهترین جواب نیستند و بستگی به محدوده ورودی های مختلف ممکن است به پاسخ های متفاوتی برسیم.

با قرار دادن مقدار 0.5 برای C و gamma نتایج زیر به دست می آیند:



```
kernel=linear , C=0.5 , gamma=0.5:
classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.79      0.92      0.85        12
           2       0.88      0.70      0.78        10

    accuracy                           0.87        30
   macro avg       0.89      0.87      0.87        30
weighted avg       0.87      0.87      0.86        30

confusion matrix:
[[ 8  0  0]
 [ 0 11  1]
 [ 0  3  7]]
```

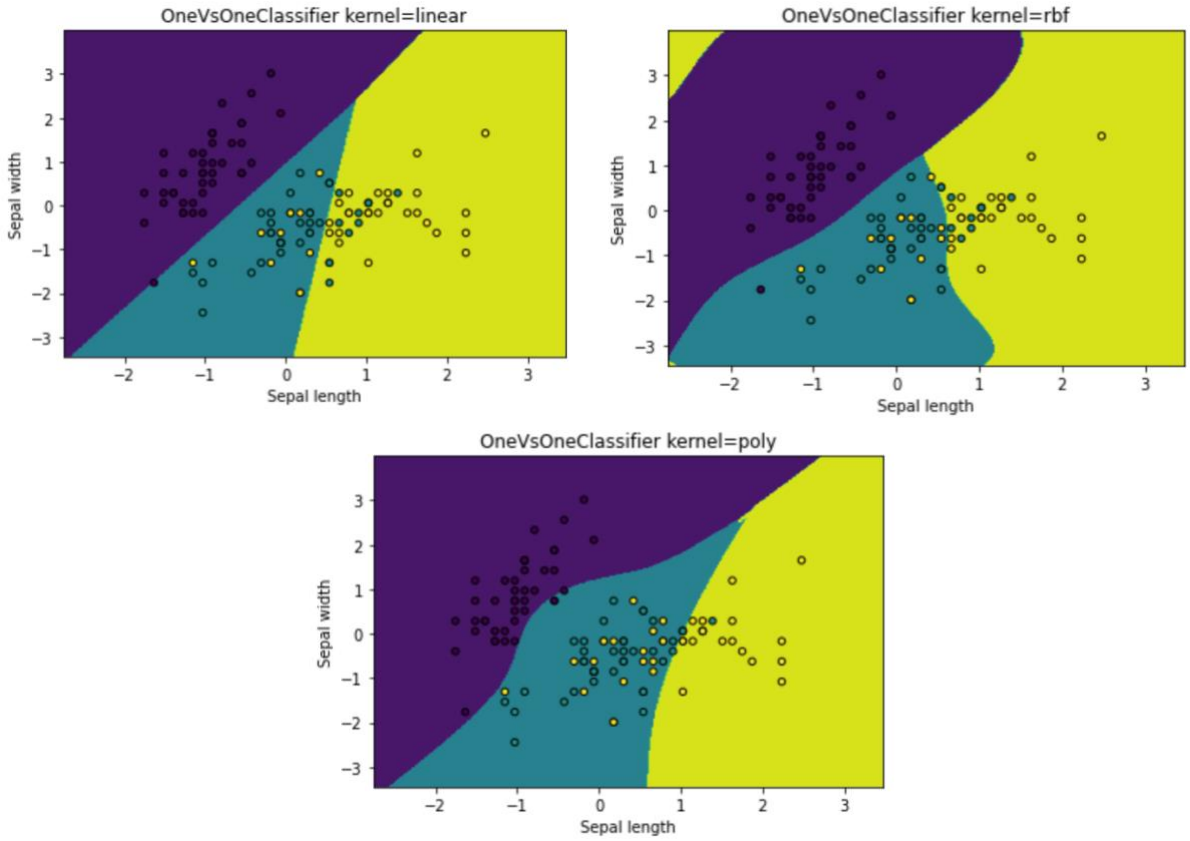
```
kernel=rbf , C=0.5 , gamma=0.5:
classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.79      0.92      0.85        12
           2       0.88      0.70      0.78        10

    accuracy                           0.87        30
   macro avg       0.89      0.87      0.87        30
weighted avg       0.87      0.87      0.86        30

confusion matrix:
[[ 8  0  0]
 [ 0 11  1]
 [ 0  3  7]]
```

```
kernel=poly , C=0.5 , gamma=0.5:
classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.60      1.00      0.75        12
           2       1.00      0.20      0.33        10

    accuracy                           0.73        30
   macro avg       0.87      0.73      0.69        30
weighted avg       0.84      0.73      0.68        30

confusion matrix:
[[ 8  0  0]
 [ 0 12  0]
 [ 0  8  2]]
```

مشاهده می شود که جوابی که gridSearch داده دقت بالایی دارد.

(d

**OneVsOne**




OneVsOneClassifier kernel=linear:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.83 | 0.83 | 0.83 | 12 |
| 2 | 0.80 | 0.80 | 0.80 | 10 |
| accuracy | | | 0.87 | 30 |
| macro avg | 0.88 | 0.88 | 0.88 | 30 |
| weighted avg | 0.87 | 0.87 | 0.87 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 10  2]
 [ 0  2  8]]
```

OneVsOneClassifier kernel=rbf:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.85 | 0.92 | 0.88 | 12 |
| 2 | 0.89 | 0.80 | 0.84 | 10 |
| accuracy | | | 0.90 | 30 |
| macro avg | 0.91 | 0.91 | 0.91 | 30 |
| weighted avg | 0.90 | 0.90 | 0.90 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  2  8]]
```

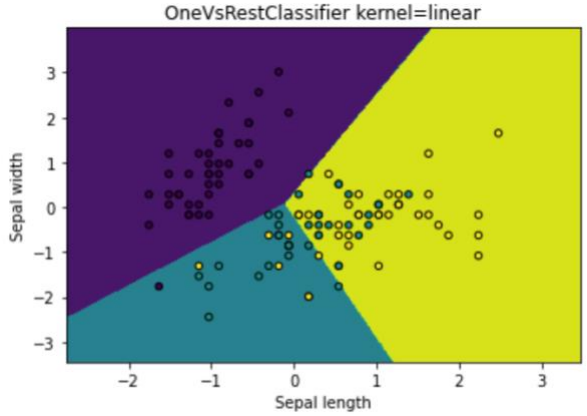
OneVsOneClassifier kernel=poly:
classification report:

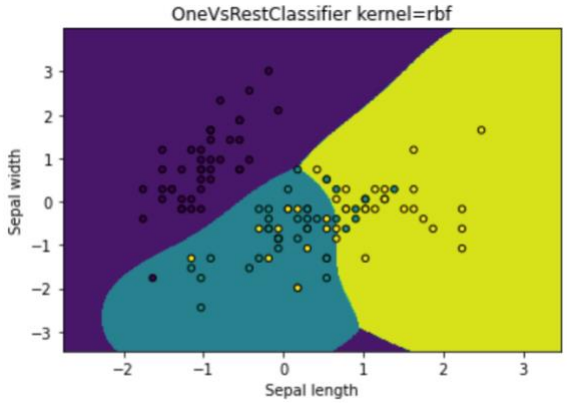
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.58 | 0.92 | 0.71 | 12 |
| 2 | 0.67 | 0.20 | 0.31 | 10 |
| accuracy | | | 0.70 | 30 |
| macro avg | 0.75 | 0.71 | 0.67 | 30 |
| weighted avg | 0.72 | 0.70 | 0.65 | 30 |

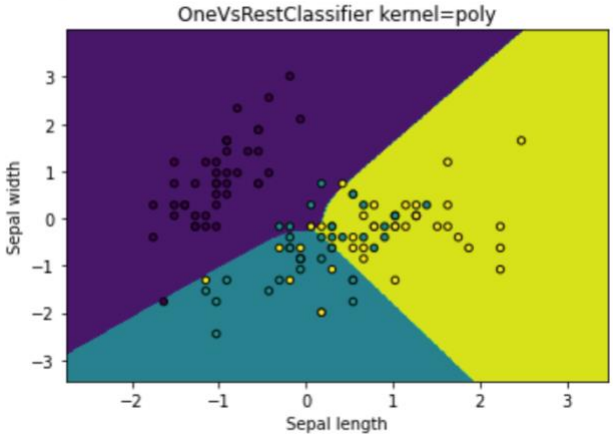
confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  8  2]]
```

## OneVsRest




OneVsRestClassifier kernel=linear:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.89 | 1.00 | 0.94 | 8 |
| 1 | 1.00 | 0.83 | 0.91 | 12 |
| 2 | 0.91 | 1.00 | 0.95 | 10 |
| accuracy | | | 0.93 | 30 |
| macro avg | 0.93 | 0.94 | 0.93 | 30 |
| weighted avg | 0.94 | 0.93 | 0.93 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 1 10  1]
 [ 0  0 10]]
```

OneVsRestClassifier kernel=rbf:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 8 |
| 1 | 0.79 | 0.92 | 0.85 | 12 |
| 2 | 0.88 | 0.70 | 0.78 | 10 |
| accuracy | | | 0.87 | 30 |
| macro avg | 0.89 | 0.87 | 0.87 | 30 |
| weighted avg | 0.87 | 0.87 | 0.86 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 0 11  1]
 [ 0  3  7]]
```

OneVsRestClassifier kernel=poly:
classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.80 | 1.00 | 0.89 | 8 |
| 1 | 1.00 | 0.75 | 0.86 | 12 |
| 2 | 0.91 | 1.00 | 0.95 | 10 |
| accuracy | | | 0.90 | 30 |
| macro avg | 0.90 | 0.92 | 0.90 | 30 |
| weighted avg | 0.92 | 0.90 | 0.90 | 30 |

confusion matrix:

```
[[ 8  0  0]
 [ 2  9  1]
 [ 0  0 10]]
```

می بینیم که در روش دوم به دقت های بهتری رسیدیم. در واقع در روش oneVsRest نقاط مبهم کمتری داریم اما مشکلی که این روش دارد این است که مسئله را باید به ازای تمام جفت کلاس ها حل کنیم که محاسبات بالایی نسبت به روش اول دارد.
علاوه بر این می بینیم که در روش اول منحنی مرزها پیچیده تر شده و شکست های بیشتری دارد.
باید توجه داشت که روش linear machine که از discriminant function ها استفاده می کند به نسبت دو روش دیگر اشکالات کمتری دارد.