Support Vector Machines com Python

1 Support Vector Machines com Python

1.1 Importando bibliotecas

[51]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1.2 Obter dados

Usaremos o conjunto de dados de câncer de mama incorporado da Scikit Learn. Podemos obter com a função load:

[52]:
```
from sklearn.datasets import load_breast_cancer
```

[54]:
```
cancer = load_breast_cancer()
```

O conjunto de dados é apresentado em uma forma de dicionário:

[55]:
```
cancer.keys()
```

[55]:
```
dict_keys(['DESCR', 'target', 'data', 'target_names', 'feature_names'])
```

Podemos pegar informações e arrays deste dicionário para configurar nosso dataframe e entender os recursos:

[4]:
```
print(cancer['DESCR'])
```

```
Breast Cancer Wisconsin (Diagnostic) Database

Notes
-----
Data Set Characteristics:
    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class
```

:Attribute Information:
    - radius (mean of distances from center to points on the perimeter)
    - texture (standard deviation of gray-scale values)
    - perimeter
    - area
    - smoothness (local variation in radius lengths)
    - compactness (perimeter^2 / area - 1.0)
    - concavity (severity of concave portions of the contour)
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension ("coastline approximation" - 1)

    The mean, standard error, and "worst" or largest (mean of the three
    largest values) of these features were computed for each image,
    resulting in 30 features. For instance, field 3 is Mean Radius, field
    13 is Radius SE, field 23 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54

```
    perimeter (worst):                    50.41   251.2
    area (worst):                         185.2   4254.0
    smoothness (worst):                   0.071   0.223
    compactness (worst):                  0.027   1.058
    concavity (worst):                    0.0     1.252
    concave points (worst):               0.0     0.291
    symmetry (worst):                     0.156   0.664
    fractal dimension (worst):            0.055   0.208
    ================================== ======= ========

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.
A few of the images can be found at
http://www.cs.wisc.edu/~street/images/

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

```
References
----------
   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages
570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning
techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77
(1994)
     163-171.
```

[56]:
```
cancer['feature_names']
```

[56]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension'],
      dtype='<U23')
```

1.3 Configurando o DataFrame

[12]:
```
df_feat = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df_feat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
mean radius                569 non-null float64
mean texture               569 non-null float64
mean perimeter             569 non-null float64
mean area                  569 non-null float64
mean smoothness            569 non-null float64
mean compactness           569 non-null float64
mean concavity             569 non-null float64
mean concave points        569 non-null float64
```

```
mean symmetry                569 non-null float64
mean fractal dimension       569 non-null float64
radius error                 569 non-null float64
texture error                569 non-null float64
perimeter error              569 non-null float64
area error                   569 non-null float64
smoothness error             569 non-null float64
compactness error            569 non-null float64
concavity error              569 non-null float64
concave points error         569 non-null float64
symmetry error               569 non-null float64
fractal dimension error      569 non-null float64
worst radius                 569 non-null float64
worst texture                569 non-null float64
worst perimeter              569 non-null float64
worst area                   569 non-null float64
worst smoothness             569 non-null float64
worst compactness            569 non-null float64
worst concavity              569 non-null float64
worst concave points         569 non-null float64
worst symmetry               569 non-null float64
worst fractal dimension      569 non-null float64
dtypes: float64(30)
memory usage: 133.4 KB
```

[14]:
```
cancer['target']
```

[14]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

[16]:
```
df_target = pd.DataFrame(cancer['target'],columns=['Cancer'])
```

Agora vamos verificar o dataframe:

[8]:
```
df.head()
```

[8]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.3001	0.14710	0.2419	
1	0.07864	0.0869	0.07017	0.1812	
2	0.15990	0.1974	0.12790	0.2069	
3	0.28390	0.2414	0.10520	0.2597	
4	0.13280	0.1980	0.10430	0.1809	

	mean fractal dimension	…	worst radius	\
0	0.07871	…	25.38	
1	0.05667	…	24.99	
2	0.05999	…	23.57	
3	0.09744	…	14.91	
4	0.05883	…	22.54	

	worst texture	worst perimeter	worst area	worst smoothness	\
0	17.33	184.60	2019.0	0.1622	
1	23.41	158.80	1956.0	0.1238	
2	25.53	152.50	1709.0	0.1444	
3	26.50	98.87	567.7	0.2098	
4	16.67	152.20	1575.0	0.1374	

	worst compactness	worst concavity	worst concave points	worst symmetry	\
0	0.6656	0.7119	0.2654	0.4601	
1	0.1866	0.2416	0.1860	0.2750	
2	0.4245	0.4504	0.2430	0.3613	
3	0.8663	0.6869	0.2575	0.6638	
4	0.2050	0.4000	0.1625	0.2364	

```
   worst fractal dimension
0                  0.11890
1                  0.08902
2                  0.08758
3                  0.17300
4                  0.07678

[5 rows x 30 columns]
```

2 Análise de dados exploratórios

Nós ignoraremos a parte Data Viz para esta leitura, pois existem tantos parâmetros que são difíceis de interpretar se você não tem conhecimento e domínio de câncer ou células tumorais. No seu projeto você terá mais oportunidades para visualizar os dados.

2.1 Divisão treino-teste

[57]:
```
from sklearn.model_selection import train_test_split
```

[58]:
```
X_train, X_test, y_train, y_test = train_test_split(df_feat, np.
↪ravel(df_target), test_size=0.30, random_state=101)
```

3 Treinando o Support Vector Classifier

[59]:
```
from sklearn.svm import SVC
```

[60]:
```
model = SVC()
```

[61]:
```
model.fit(X_train,y_train)
```

[61]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

3.1 Previsões e avaliações

Agora vamos prever o uso do modelo treinado.

[27]:
```
predictions = model.predict(X_test)
```

[45]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[46]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[  0  66]
 [  0 105]]
```

[62]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00        66
          1       0.61      1.00      0.76       105

avg / total       0.38      0.61      0.47       171


/Users/marci/anaconda/lib/python3.5/site-
packages/sklearn/metrics/classification.py:1074: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

Note que estamos classificando tudo em uma única classe! Isso significa que nosso modelo precisa ter parâmetros ajustados (também pode ajudar a normalizar os dados).

Podemos procurar por parâmetros usando um GridSearch!

4 Gridsearch

Encontrar os parâmetros certos (como o que o C ou os valores de gama para usar) é uma tarefa complicada! Mas, felizmente, podemos ser um pouco preguiçosos e apenas tentar um monte de combinações e ver o que funciona melhor! Essa idéia de criar uma "grade" de parâmetros e apenas experimentar todas as combinações possíveis é chamada Gridsearch, esse método é comum o suficiente para que o Scikit-learn tenha essa funcionalidade incorporada no GridSearchCV!

O GridSearchCV usa um dicionário que descreve os parâmetros que devem ser testados e um modelo para treinar. A grade de parâmetros é definida como um dicionário, onde as chaves são os parâmetros e os valores são as configurações a serem testadas.

[63]:
```
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001],␣
↪'kernel': ['rbf']}
```

[64]:
```
from sklearn.model_selection import GridSearchCV
```

Uma das grandes coisas do GridSearchCV é que é um meta-estimador. Ele pega um estimador como SVC e cria um novo estimador, que se comporta exatamente da mesma forma - neste caso, como um classificador. Você deve adicionar refit=True e escolher detalhadamente para número desejado, maior o número, mais detalhado.

Você deve adicionar refit=True e escolher um número para o parâmetro verbose. Quanto maior o número, mais detalhamento teremos.

[65]:
```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
```

O que o fit faz é um pouco mais complicado do que o habital. Primeiro, ele executa o mesmo loop com validação cruzada, para encontrar a melhor combinação de parâmetros. Uma vez que tenha a melhor combinação, ele corre novamente em todos os dados para fitá-los (sem validação cruzada), para construir um único modelo novo usando a melhor configuração de parâmetro.

[40]:
```
# Talvez demore um pouco
grid.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
[CV] gamma=1, C=0.1, kernel=rbf …
[CV] … gamma=1, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=0.1, kernel=rbf …
[CV] … gamma=1, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=0.1, kernel=rbf …
[CV] … gamma=1, C=0.1, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.1, C=0.1, kernel=rbf …
[CV] … gamma=0.1, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=0.1, kernel=rbf …
[CV] … gamma=0.1, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=0.1, kernel=rbf …
[CV] … gamma=0.1, C=0.1, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.01, C=0.1, kernel=rbf …
[CV] … gamma=0.01, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=0.1, kernel=rbf …
[CV] … gamma=0.01, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=0.1, kernel=rbf …
[CV] … gamma=0.01, C=0.1, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.001, C=0.1, kernel=rbf …
[CV] … gamma=0.001, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.001, C=0.1, kernel=rbf …
[CV] … gamma=0.001, C=0.1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.001, C=0.1, kernel=rbf …
[CV] … gamma=0.001, C=0.1, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.0001, C=0.1, kernel=rbf …
[CV] … gamma=0.0001, C=0.1, kernel=rbf, score=0.902256 -   0.0s
[CV] gamma=0.0001, C=0.1, kernel=rbf …
[CV] … gamma=0.0001, C=0.1, kernel=rbf, score=0.962406 -   0.0s
[CV] gamma=0.0001, C=0.1, kernel=rbf …
[CV] … gamma=0.0001, C=0.1, kernel=rbf, score=0.916667 -   0.0s
[CV] gamma=1, C=1, kernel=rbf …
[CV] … gamma=1, C=1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=1, kernel=rbf …
[CV] … gamma=1, C=1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=1, kernel=rbf …
[CV] … gamma=1, C=1, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.1, C=1, kernel=rbf …
[CV] … gamma=0.1, C=1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=1, kernel=rbf …
```

```
[CV] … gamma=0.1, C=1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=1, kernel=rbf …
[CV] … gamma=0.1, C=1, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.01, C=1, kernel=rbf …
[CV] … gamma=0.01, C=1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=1, kernel=rbf …
[CV] … gamma=0.01, C=1, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=1, kernel=rbf …
[CV] … gamma=0.01, C=1, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.001, C=1, kernel=rbf …
[CV] … gamma=0.001, C=1, kernel=rbf, score=0.902256 -   0.0s
[CV] gamma=0.001, C=1, kernel=rbf …
[CV] … gamma=0.001, C=1, kernel=rbf, score=0.939850 -   0.0s
[CV] gamma=0.001, C=1, kernel=rbf …
[CV] … gamma=0.001, C=1, kernel=rbf, score=0.954545 -   0.0s
[CV] gamma=0.0001, C=1, kernel=rbf …
[CV] … gamma=0.0001, C=1, kernel=rbf, score=0.939850 -   0.0s
[CV] gamma=0.0001, C=1, kernel=rbf …
[CV] … gamma=0.0001, C=1, kernel=rbf, score=0.969925 -   0.0s
[CV] gamma=0.0001, C=1, kernel=rbf …
[CV] … gamma=0.0001, C=1, kernel=rbf, score=0.946970 -   0.0s
[CV] gamma=1, C=10, kernel=rbf …
[CV] … gamma=1, C=10, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=10, kernel=rbf …
[CV] … gamma=1, C=10, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=10, kernel=rbf …
[CV] … gamma=1, C=10, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.1, C=10, kernel=rbf …
[CV] … gamma=0.1, C=10, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=10, kernel=rbf …
[CV] … gamma=0.1, C=10, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=10, kernel=rbf …
[CV] … gamma=0.1, C=10, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.01, C=10, kernel=rbf …
[CV] … gamma=0.01, C=10, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=10, kernel=rbf …
[CV] … gamma=0.01, C=10, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=10, kernel=rbf …
[CV] … gamma=0.01, C=10, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.001, C=10, kernel=rbf …
[CV] … gamma=0.001, C=10, kernel=rbf, score=0.894737 -   0.0s
[CV] gamma=0.001, C=10, kernel=rbf …
[CV] … gamma=0.001, C=10, kernel=rbf, score=0.932331 -   0.0s
[CV] gamma=0.001, C=10, kernel=rbf …
[CV] … gamma=0.001, C=10, kernel=rbf, score=0.916667 -   0.0s
[CV] gamma=0.0001, C=10, kernel=rbf …
[CV] … gamma=0.0001, C=10, kernel=rbf, score=0.932331 -   0.0s
[CV] gamma=0.0001, C=10, kernel=rbf …
```

```
[CV] … gamma=0.0001, C=10, kernel=rbf, score=0.969925 -   0.0s
[CV] gamma=0.0001, C=10, kernel=rbf …
[CV] … gamma=0.0001, C=10, kernel=rbf, score=0.962121 -   0.0s
[CV] gamma=1, C=100, kernel=rbf …
[CV] … gamma=1, C=100, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=100, kernel=rbf …
[CV] … gamma=1, C=100, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=100, kernel=rbf …
[CV] … gamma=1, C=100, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.1, C=100, kernel=rbf …
[CV] … gamma=0.1, C=100, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=100, kernel=rbf …
[CV] … gamma=0.1, C=100, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=100, kernel=rbf …
[CV] … gamma=0.1, C=100, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.01, C=100, kernel=rbf …
[CV] … gamma=0.01, C=100, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=100, kernel=rbf …
[CV] … gamma=0.01, C=100, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=100, kernel=rbf …
[CV] … gamma=0.01, C=100, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.001, C=100, kernel=rbf …
[CV] … gamma=0.001, C=100, kernel=rbf, score=0.894737 -   0.0s
[CV] gamma=0.001, C=100, kernel=rbf …
[CV] … gamma=0.001, C=100, kernel=rbf, score=0.932331 -   0.0s
[CV] gamma=0.001, C=100, kernel=rbf …
[CV] … gamma=0.001, C=100, kernel=rbf, score=0.916667 -   0.0s
[CV] gamma=0.0001, C=100, kernel=rbf …
[CV] … gamma=0.0001, C=100, kernel=rbf, score=0.917293 -   0.0s
[CV] gamma=0.0001, C=100, kernel=rbf …
[CV] … gamma=0.0001, C=100, kernel=rbf, score=0.977444 -   0.0s
[CV] gamma=0.0001, C=100, kernel=rbf …
[CV] … gamma=0.0001, C=100, kernel=rbf, score=0.939394 -   0.0s
[CV] gamma=1, C=1000, kernel=rbf …
[CV] … gamma=1, C=1000, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=1000, kernel=rbf …
[CV] … gamma=1, C=1000, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=1, C=1000, kernel=rbf …
[CV] … gamma=1, C=1000, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.1, C=1000, kernel=rbf …
[CV] … gamma=0.1, C=1000, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=1000, kernel=rbf …
[CV] … gamma=0.1, C=1000, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.1, C=1000, kernel=rbf …
[CV] … gamma=0.1, C=1000, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.01, C=1000, kernel=rbf …
[CV] … gamma=0.01, C=1000, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=1000, kernel=rbf …
```

```
[CV] … gamma=0.01, C=1000, kernel=rbf, score=0.631579 -   0.0s
[CV] gamma=0.01, C=1000, kernel=rbf …
[CV] … gamma=0.01, C=1000, kernel=rbf, score=0.636364 -   0.0s
[CV] gamma=0.001, C=1000, kernel=rbf …
[CV] … gamma=0.001, C=1000, kernel=rbf, score=0.894737 -   0.0s
[CV] gamma=0.001, C=1000, kernel=rbf …
[CV] … gamma=0.001, C=1000, kernel=rbf, score=0.932331 -   0.0s
[CV] gamma=0.001, C=1000, kernel=rbf …
[CV] … gamma=0.001, C=1000, kernel=rbf, score=0.916667 -   0.0s

[Parallel(n_jobs=1)]: Done  31 tasks       | elapsed:    0.3s
[Parallel(n_jobs=1)]: Done  75 out of  75 | elapsed:    0.8s finished


[CV] gamma=0.0001, C=1000, kernel=rbf …
[CV] … gamma=0.0001, C=1000, kernel=rbf, score=0.909774 -   0.0s
[CV] gamma=0.0001, C=1000, kernel=rbf …
[CV] … gamma=0.0001, C=1000, kernel=rbf, score=0.969925 -   0.0s
[CV] gamma=0.0001, C=1000, kernel=rbf …
[CV] … gamma=0.0001, C=1000, kernel=rbf, score=0.931818 -   0.0s
```

[40]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'C': [0.1, 1, 10,
100, 1000], 'kernel': ['rbf']},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=3)
```

Você pode inspecionar os melhores parâmetros encontrados pelo GridSearchCV no atributo best_params_ e o melhor estimador no melhor atributo *estimator*:

[41]:
```
grid.best_params_
```

[41]:
```
{'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
```

[ ]:
```
grid.best_estimator_
```

Então você pode re-executar previsões neste objeto da grade exatamente como você faria com um modelo normal.

[48]:
```
grid_predictions = grid.predict(X_test)
```

[49]:
```
print(confusion_matrix(y_test,grid_predictions))
```

```
[[ 60   6]
 [  3 102]]
```

[50]:
```
print(classification_report(y_test,grid_predictions))
```

```
             precision    recall  f1-score   support

          0       0.95      0.91      0.93        66
          1       0.94      0.97      0.96       105

avg / total       0.95      0.95      0.95       171
```