Maquinas de soporte vectorial

June 28, 2023

1 Máquinas de soporte vectorial (SVM)

1.1 Paquetes numpy y pandas

[1]:
```
import numpy as np
import pandas as pd
```

1.1.1 Para fines de estética en la salida, se desactivan las advertencias que pueda informar el intérprete Python

[2]:
```
import warnings
warnings.filterwarnings("ignore")
```

1.2 Paquetes para la construcción del gráfico

[3]:
```
# Paquetes para los gráficos
import matplotlib.pyplot as plt
import graphviz
```

1.3 Importación método para creación del conjunto de entrenamiento desde paquete *sklearn*

[4]:
```
from sklearn.model_selection import train_test_split
```

1.4 Paquete sklearn que contiene los métodos para árboles de decisión

[5]:
```
# Métodos para árboles de decisión desde sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn import svm, datasets
```

1.5 Lectura de los datos desde el archivo *datosAB.txt*

[6]:
```
# Esta es la opción para Jupyter Lab/Notebook
datos = pd.read_table("datosAB.txt", sep='\t')
```

1.6 Creación de conjunto de datos

[51]:
```
# Conjunto de datos
X = datos.iloc[:,:-1]
y = datos.iloc[:,2]
```

1.7 Creación de subconjutos CP y CE

[52]:
```
# Se elige una semilla para la selección pseudo-aleatoria
semilla = 123456
```

[53]:
```
X_ce, X_cp, y_ce, y_cp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=semilla)
```

1.8 Creación y ajuste del clasificador SVM

[54]:
```
# Entrenamiento y ajuste
clasificador = svm.SVC(kernel="poly", degree=3, gamma="auto", C=1.0)
clasificador = clasificador.fit(X_ce, y_ce)
```

1.9 Predicción

[55]:
```
y_pred = clasificador.predict(X_cp)
```

[56]:
```
print(y_cp)
```

```
16       rojo
30    naranja
0        rojo
22    naranja
35    naranja
24    naranja
3        rojo
5        rojo
9        rojo
17       rojo
34    naranja
Name: clase, dtype: object
```

[57]:
```
print(y_pred)
```

```
['rojo' 'naranja' 'rojo' 'naranja' 'rojo' 'naranja' 'rojo' 'rojo'
 'naranja' 'naranja' 'naranja']
```

1.10 Creación de los resultados estadísticos de la clasificación

1.10.1 Importación de método para la matriz de confusión desde paquete *sklearn*

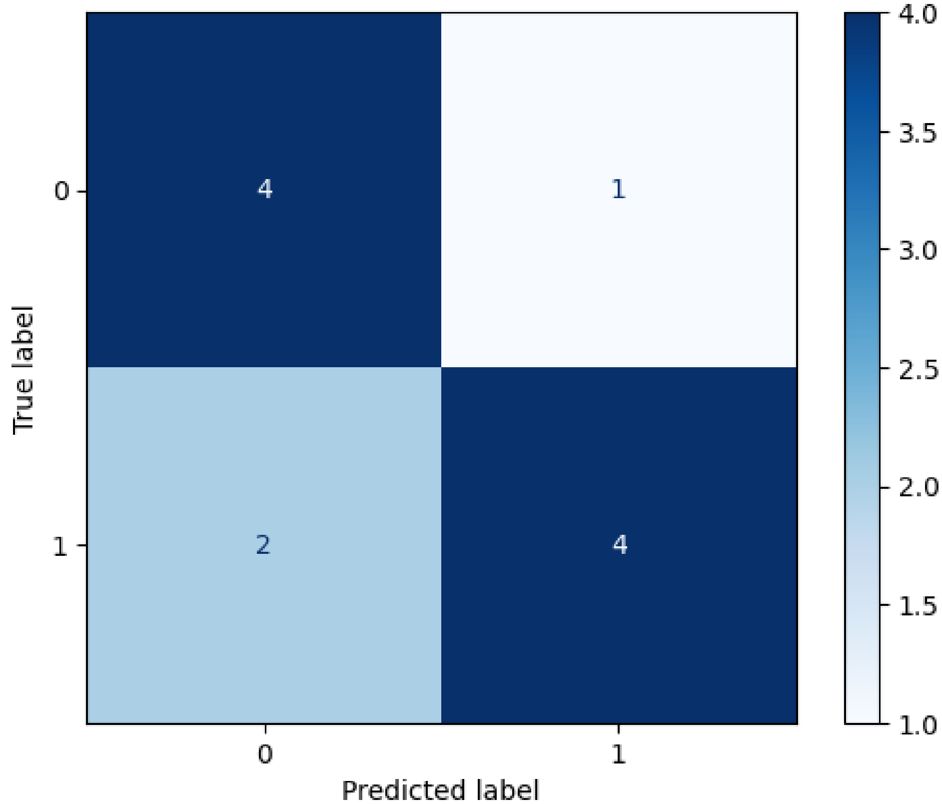
[58]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

1.10.2 Cálculo de la matriz de confusión

[59]:
```
mconf = confusion_matrix(y_cp, y_pred)
```

1.10.3 Impresión de la matriz de confusión

[60]:
```
mconfg = ConfusionMatrixDisplay(mconf).plot(cmap='Blues')
```



1.10.4 Importación de método para la puntuación de precisión desde paquete *sklearn*

[61]:
```
from sklearn.metrics import accuracy_score
```

1.10.5 Cálculo de la puntuación de precisión

[62]:
```
cc = accuracy_score(y_cp, y_pred)
```

1.10.6 Impresión de la puntuación

[63]:
```
print(f'Accuracy Score = {cc}')
```

```
Accuracy Score = 0.7272727272727273
```

1.11 Importación de métodos para el gráfico

[64]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

1.12 Ajuste del etiquetado de la variable y

[65]:
```
y_ce
```

[65]:
```
7        rojo
19    naranja
13       rojo
31    naranja
33    naranja
25    naranja
28    naranja
21    naranja
14       rojo
2        rojo
6        rojo
18    naranja
15       rojo
26    naranja
20    naranja
11       rojo
12       rojo
29    naranja
10       rojo
8        rojo
4        rojo
23    naranja
32    naranja
27    naranja
1        rojo
Name: clase, dtype: object
```

[66]:
```
y_cp
```

[66]:
```
16       rojo
30    naranja
0        rojo
22    naranja
35    naranja
24    naranja
3        rojo
5        rojo
9        rojo
17       rojo
34    naranja
Name: clase, dtype: object
```

[67]:
```python
# Importación del etiquetador
from sklearn.preprocessing import LabelEncoder
# Creación del etiquetador
labelencoder_y = LabelEncoder()
# Etiquetado y ajuste
y_ce = labelencoder_y.fit_transform(y_ce)
```

[68]:
```python
y_ce
```

[68]:
```
array([1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0,
       0, 0, 1])
```

1.12.1 Nota: Es necesario realizar el ajuste de nuevo dado que cambió la variable y debido al proceso de etiquetado

[69]:
```python
clasificador.fit(X_ce, y_ce)
```

[69]:
```
SVC(gamma='auto', kernel='poly')
```

2 Se grafica todo el conjunto de datos empleando el clasificador DT para cada dato

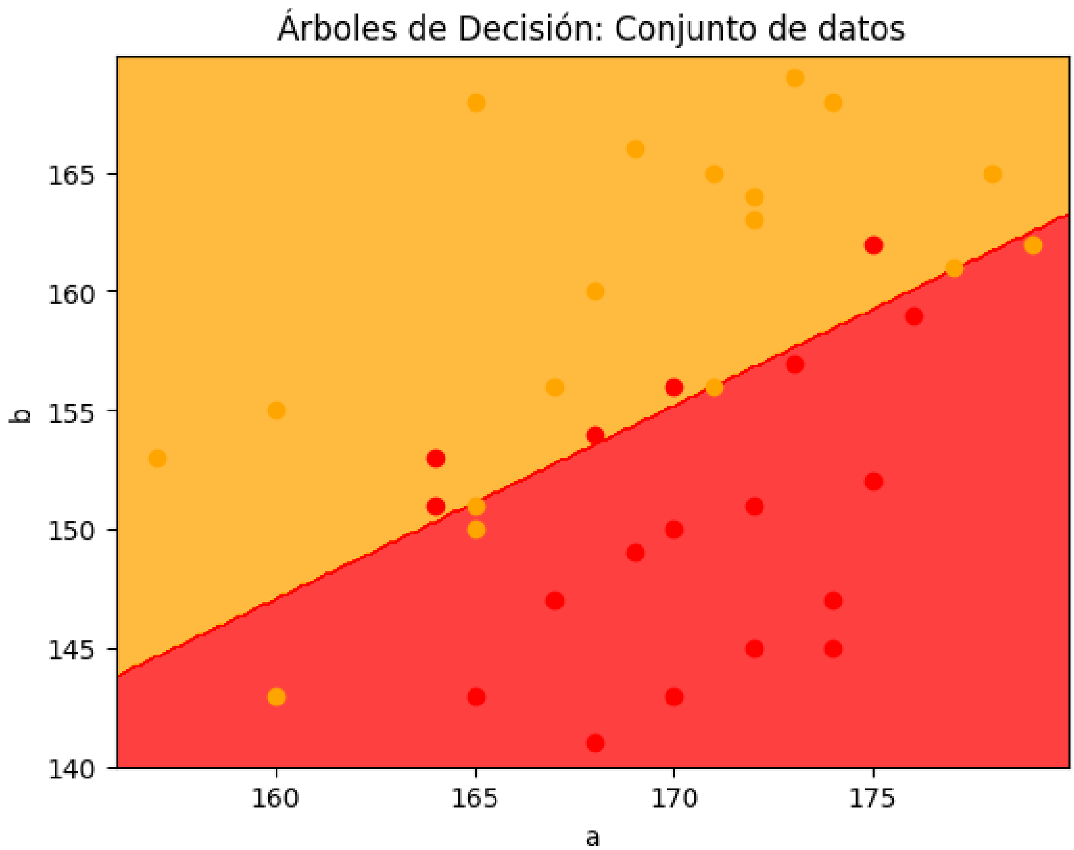
[70]:
```python
# Etiquetado y ajuste del conjunto de datos original
X_set, y_set = X, labelencoder_y.fit_transform(y)
```

2.1 Creación de la malla (plano cartesiano)

[71]:
```python
X1, X2 = np.meshgrid(
    np.arange(start = X_set.iloc[:,0].min()-1, stop = X_set.iloc[:,0].max()+1,␣
 ↪step=0.1),
    np.arange(start = X_set.iloc[:,1].min()-1, stop = X_set.iloc[:,1].max()+1,␣
 ↪step=0.1)
)
```

2.2 Creación del gráfico

[72]:
```python
# Al construir la malla, se colorea la región de naranja o rojo
# de acuerdo al clasificador DT obtenido
plt.contourf(X1, X2,
    clasificador.predict(
        np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
        alpha = 0.75, cmap = ListedColormap(('orange', 'red'))
)

# Se establecen los límites de los ejes x,y en el gráfico
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# Se grafica cada dato en el plano cartesiano, la clase de cada dato determina␣
↪el color.
# Debido al proceso de etiquetado, 'n' fue sustituido por 0 y 'r' sustituido␣
↪por 1
# 0 -> Naranja
# 1 -> Rojo
j=0
for i in y_set:
    if i==0:
        color = "orange"
    else:
        color = "red"
    plt.scatter(
        X_set.iloc[j,0], # a
        X_set.iloc[j,1], # b
        c = color,
        label = i
    )
    j=j+1

# Etiqueta del gráfico y sus ejes
plt.title('Árboles de Decisión: Conjunto de datos')
plt.xlabel('a')
plt.ylabel('b')

# Creación del gráfico
plt.show()
```



3 Clasificar nuevos datos con DT

3.1 Se clasifica un dato con el clasificador construido con DT

dato = (160, 145)

[73]:
```python
# Predicción del dato = (160, 145)
x = clasificador.predict([[160, 145]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
rojo
```

3.2 Se clasifica otro dato con el clasificador construido con DT

dato = (160, 165)

[74]:
```python
# Predicción del dato = (160, 165)
x = clasificador.predict([[160, 165]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
naranja
```

3.3 Ahora, a manera de prueba, se clasifica el promedio de los datos

[75]:
```python
# X_set es un DataFrame de pandas
X_set.mean(0)
```

[75]:
```
a    169.694444
b    155.000000
dtype: float64
```

[76]:
```python
# Predicción del dato promedio = (169.6944, 155)
x = clasificador.predict([[169.6944, 155]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
naranja
```

4 Clasificación multiclase

Se utiliza el conjuto de datos iris para la clasificación mediante SVC

[77]:
```python
iris = datasets.load_iris()
print(iris)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
```

```
[5.7, 4.4, 1.5, 0.4],
[5.4, 3.9, 1.3, 0.4],
[5.1, 3.5, 1.4, 0.3],
[5.7, 3.8, 1.7, 0.3],
[5.1, 3.8, 1.5, 0.3],
[5.4, 3.4, 1.7, 0.2],
[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
```

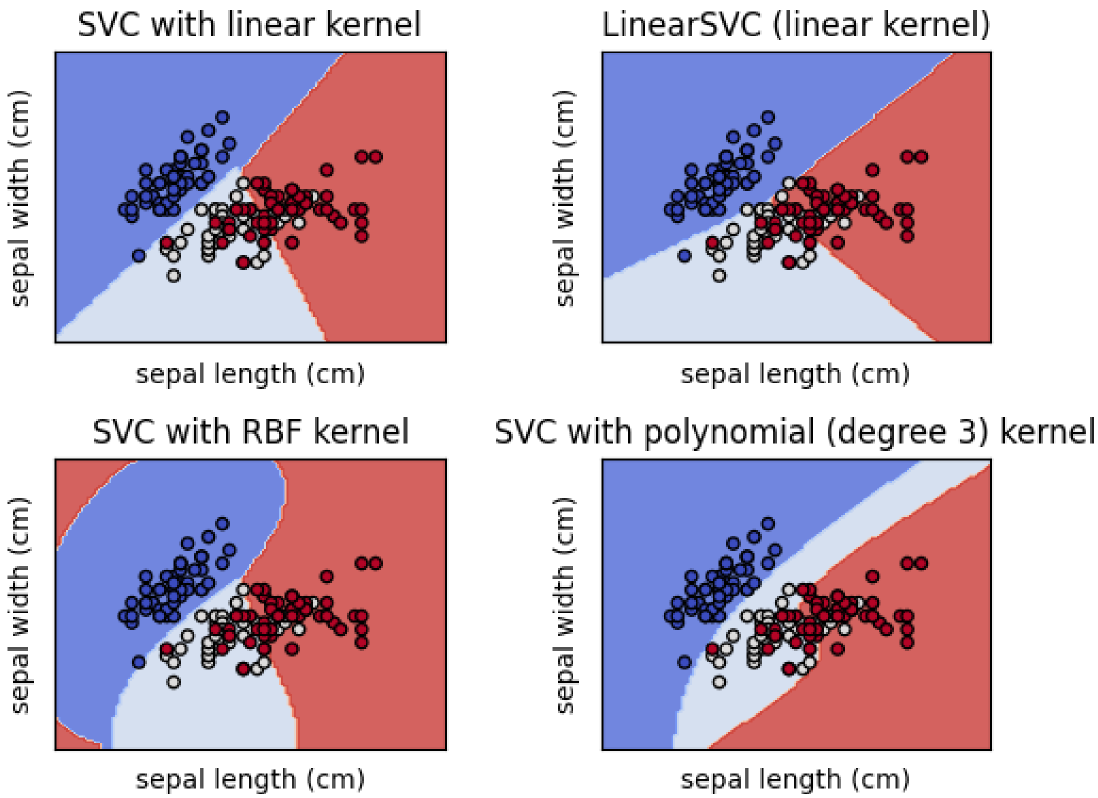
```
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
```

```
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 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, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'frame': None,
'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
'DESCR': '.. _iris_dataset:\n\nIris plants
```

dataset\n--------------------\n\n**Data Set Characteristics:**\n\n   :Number of Instances: 150 (50 in each of three classes)\n   :Number of Attributes: 4 numeric, predictive attributes and the class\n   :Attribute Information:\n   - sepal length in cm\n   - sepal width in cm\n   - petal length in cm\n   - petal width in cm\n   - class:\n   - Iris-Setosa\n   - Iris-Versicolour\n   - Iris-Virginica\n   \n   :Summary Statistics:\n\n   ============== ==== ==== ======= ===== ====================\n   Min Max   Mean   SD   Class Correlation\n   ============== ==== ==== ======= ===== ====================\n   sepal length: 4.3 7.9   5.84   0.83   0.7826\n   sepal width: 2.0 4.4   3.05   0.43   -0.4194\n   petal length: 1.0 6.9   3.76   1.76   0.9490 (high!)\n   petal width: 0.1 2.5   1.20   0.76   0.9565 (high!)\n   ============== ==== ==== ======= ===== ====================\n\n   :Missing Attribute Values: None\n   :Class Distribution: 33.3% for each of 3 classes.\n   :Creator: R.A. Fisher\n   :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n   :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature. Fisher\'s paper is a classic in the field and\nis referenced frequently to this day. (See Duda & Hart, for example.) The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant. One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n   Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n   Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n   (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n   Structure and Classification Rule for Recognition in Partially Exposed\n   Environments". IEEE Transactions on Pattern Analysis and Machine\n   Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions\n   on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II\n   conceptual clustering system finds 3 classes in the data.\n   - Many, many more …', 'feature_names': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'filename': 'iris.csv', 'data_module': 'sklearn.datasets.data'}

[79]:
```
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.inspection import DecisionBoundaryDisplay


# import some data to play with
iris = datasets.load_iris()
```

```python
# Take the first two features. We could avoid this by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
models = (
    svm.SVC(kernel="linear", C=C),
    svm.LinearSVC(C=C, max_iter=10000),
    svm.SVC(kernel="rbf", gamma=0.7, C=C),
    svm.SVC(kernel="poly", degree=5, gamma="auto", C=C),
)
models = (clf.fit(X, y) for clf in models)

# title for the plots
titles = (
    "SVC with linear kernel",
    "LinearSVC (linear kernel)",
    "SVC with RBF kernel",
    "SVC with polynomial (degree 3) kernel",
)

# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1 = X[:, 0], X[:, 1]

for clf, title, ax in zip(models, titles, sub.flatten()):
    disp = DecisionBoundaryDisplay.from_estimator(
        clf,
        X,
        response_method="predict",
        cmap=plt.cm.coolwarm,
        alpha=0.8,
        ax=ax,
        xlabel=iris.feature_names[0],
        ylabel=iris.feature_names[1],
    )
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)

plt.show()
```



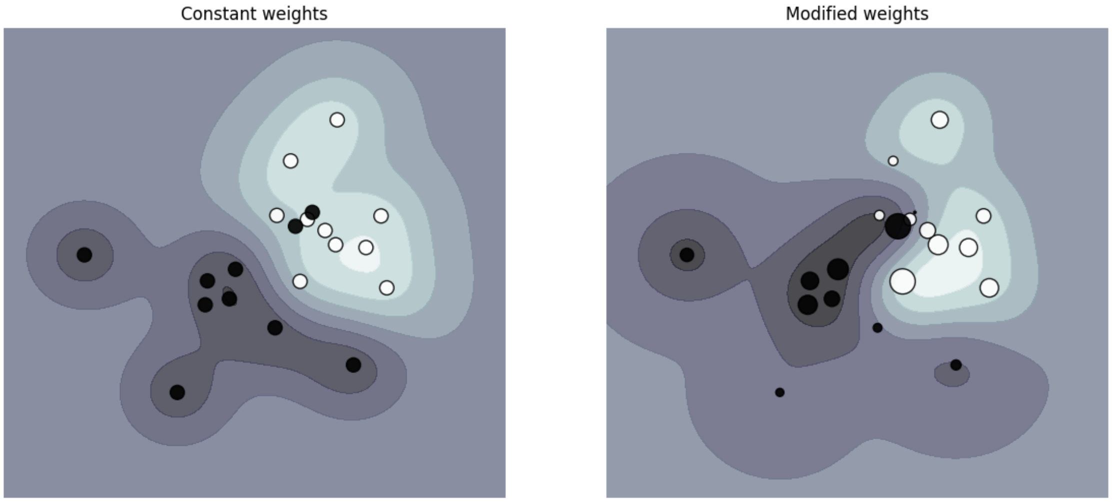
5 Clasificación desbalanceada

Muestras con peso

5.1 Ejemplo 1

[87]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm


def plot_decision_function(classifier, sample_weight, axis, title):
    # plot the decision function
    xx, yy = np.meshgrid(np.linspace(-4, 5, 500), np.linspace(-4, 5, 500))

    Z = classifier.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # plot the line, the points, and the nearest vectors to the plane
    axis.contourf(xx, yy, Z, alpha=0.75, cmap=plt.cm.bone)
    axis.scatter(
```

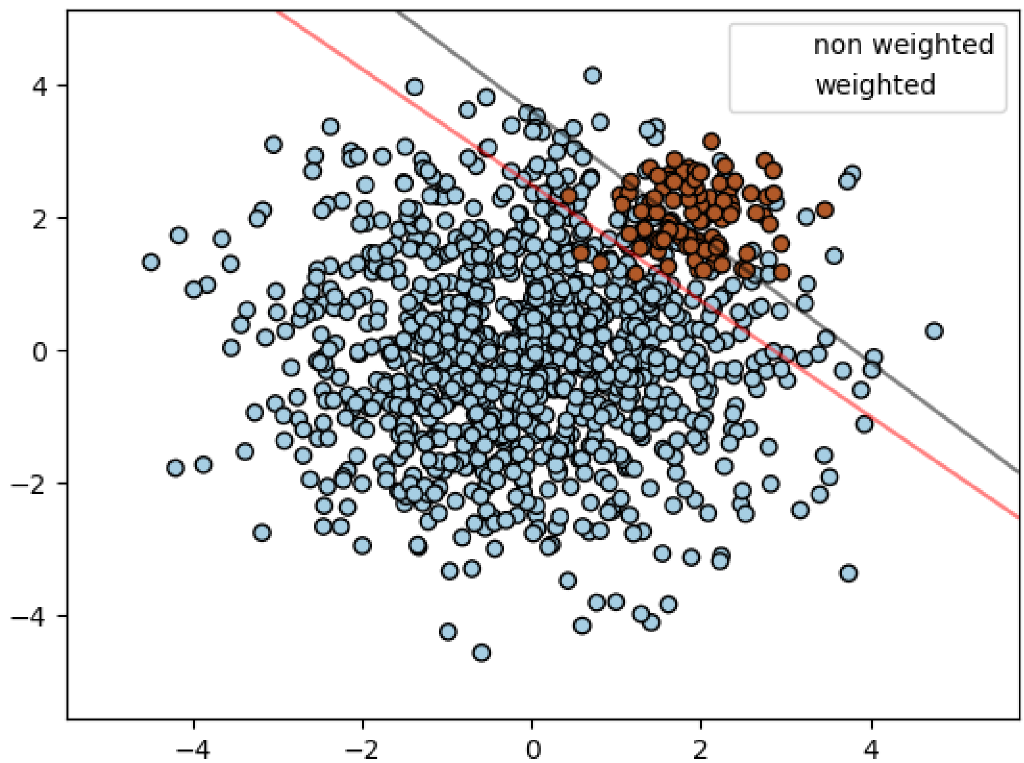
```
        X[:, 0],
        X[:, 1],
        c=y,
        s=100 * sample_weight,
        alpha=0.9,
        cmap=plt.cm.bone,
        edgecolors="black",
    )

    axis.axis("off")
    axis.set_title(title)


# we create 20 points
np.random.seed(0)
X = np.r_[np.random.randn(10, 2) + [1, 1], np.random.randn(10, 2)]
y = [1] * 10 + [-1] * 10
sample_weight_last_ten = abs(np.random.randn(len(X)))
sample_weight_constant = np.ones(len(X))
# and bigger weights to some outliers
sample_weight_last_ten[15:] *= 5
sample_weight_last_ten[9] *= 15

# Fit the models.

# This model does not take into account sample weights.
clf_no_weights = svm.SVC(gamma=1)
clf_no_weights.fit(X, y)

# This other model takes into account some dedicated sample weights.
clf_weights = svm.SVC(gamma=1)
clf_weights.fit(X, y, sample_weight=sample_weight_last_ten)

fig, axes = plt.subplots(1, 2,  figsize=(14, 6))
plot_decision_function(
    clf_no_weights, sample_weight_constant, axes[0], "Constant weights"
)
plot_decision_function(clf_weights, sample_weight_last_ten, axes[1], "Modified␣
 ↪weights")

plt.show()
```



5.2 Ejemplo 2

[88]:
```
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs
from sklearn.inspection import DecisionBoundaryDisplay

# we create two clusters of random points
n_samples_1 = 1000
n_samples_2 = 100
centers = [[0.0, 0.0], [2.0, 2.0]]
clusters_std = [1.5, 0.5]
X, y = make_blobs(
    n_samples=[n_samples_1, n_samples_2],
    centers=centers,
    cluster_std=clusters_std,
    random_state=0,
    shuffle=False,
)

# fit the model and get the separating hyperplane
clf = svm.SVC(kernel="linear", C=1.0)
clf.fit(X, y)

# fit the model and get the separating hyperplane using weighted classes
wclf = svm.SVC(kernel="linear", class_weight={1: 10})
wclf.fit(X, y)

# plot the samples
```

```python
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolors="k")

# plot the decision functions for both classifiers
ax = plt.gca()
disp = DecisionBoundaryDisplay.from_estimator(
    clf,
    X,
    plot_method="contour",
    colors="k",
    levels=[0],
    alpha=0.5,
    linestyles=["-"],
    ax=ax,
)

# plot decision boundary and margins for weighted classes
wdisp = DecisionBoundaryDisplay.from_estimator(
    wclf,
    X,
    plot_method="contour",
    colors="r",
    levels=[0],
    alpha=0.5,
    linestyles=["-"],
    ax=ax,
)

plt.legend(
    [disp.surface_.collections[0], wdisp.surface_.collections[0]],
    ["non weighted", "weighted"],
    loc="upper right",
)
plt.show()
```


non weighted
weighted
4
2
0
−2
−4
−4
−2
0
2
4

```
[ ]:
```