Convex Optimization

Problem 1: Basic Exercises of SVM in Scikit-Learn

Team Members

- Oscar Nava
- Cesar Contreras
- Yared Flores

Support Vector Machines (SVM)

SVM is a **supervised machine learning algorithm** that helps in both **classification** and **regression** problem statements.

It tries to find an optimal boundary (known as hyperplane) between different classes. In simple words, SVM does complex data transformations depending on the selected kernel function, and based on those transformations, it aims to maximize the separation boundaries between your data points.

SVM Kernel Functions

$$- \textbf{linear} : \left\langle x, x' \right\rangle$$
$$- \textbf{polynomial} : \left(\gamma \left\langle x, x' \right\rangle + r\right)^d, \text{ where d is specified by parameter degree, r by coefo.}$$
$$- \ \textbf{rbf} : \exp\left(-\gamma \|x - x'\|^2\right), where \gamma \text{is specified by parameter gamma, must be greater tha}$$
$$- \textbf{sigmoid} \tanh\left(\gamma \left\langle x, x' \right\rangle + r\right), \text{where r is specified by coefo.}$$

RFB Kernel is popular **because of its similarity to K-Nearest Neighborhood Algorithm**. It has the advantages of K-NN and overcomes the space complexity problem as RBF Kernel Support Vector Machines just needs to store the support vectors during training and not the entire dataset.

In sci-kit SVM RBF has to parameters to consider: *C* and *gamma*.

C : common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface

A low ***C*** makes the decision surface smooth, while a high `C` aims at classifying all training examples correctly.

gamma defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected.

The math behind SVM

The Optimization Problem

$$\begin{aligned}
&\min_{w,b,\xi} \mathcal{P}(w,\xi) = \tfrac{1}{2} w^T w + c\left(\nu\varepsilon + \tfrac{1}{N}\textstyle\sum_{k=1}^{N}\left(\xi_k + \xi_k^*\right)\right) \\
&\text{ s. t.} \quad y_k - w^T \varphi\left(x_k\right) - b \leq \varepsilon + \xi_k, \quad k = 1, \ldots, N \\
&w^T \varphi\left(x_k\right) + b - y_k \leq \varepsilon + \xi_k^*, \quad k = 1, \ldots, N \\
&\xi_k, \xi_k^* \geq 0, \quad k = 1, \ldots, N
\end{aligned}$$

Decision Function

$$\text{s. t.} \quad y_k - w^T \varphi\left(x_k\right) - b \leq \varepsilon + \xi_k, \quad k = 1, \ldots, N$$

Loss Function

$$\mathcal{P}(w,\xi) = \frac{1}{2} w^T w + c\left(\nu\varepsilon + \frac{1}{N}\sum_{k=1}^{N}\left(\xi_k + \xi_k^*\right)\right)$$

SVC (Classification)

$$\begin{aligned}
\min_{w,b,\zeta} \tfrac{1}{2} w^T w + C \textstyle\sum_{i=1}^{n} \zeta_i \\
\text{subject to } y_i\left(w^T \phi\left(x_i\right) + b\right) \geq 1 - \zeta_i \\
\zeta_i \geq 0, i = 1, \ldots, n
\end{aligned}$$

The dual problem to the primal is:

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

The output decision function for a given x becomes:

$$\sum_{i \in SV} y_i \alpha_i K\left(x_i, x\right) + b$$

NuSVC

$$\min_{w,b,\xi} \mathcal{P}(w,\xi) = \tfrac{1}{2}w^T w + c\left(\nu\varepsilon + \tfrac{1}{N}\sum_{k=1}^{N}\left(\xi_k + \xi_k^*\right)\right)$$
$$\text{s. t.} \quad y_k - w^T\varphi\left(x_k\right) - b \leq \varepsilon + \xi_k, \quad k = 1,\ldots,N$$
$$w^T\varphi\left(x_k\right) + b - y_k \leq \varepsilon + \xi_k^*, \quad k = 1,\ldots,N$$
$$\xi_k, \xi_k^* \geq 0, \quad k = 1,\ldots,N$$

SVR

$$\min_{w,b,\zeta,\zeta^*} \tfrac{1}{2}w^T w + C\sum_{i=1}^{n}\left(\zeta_i + \zeta_i^*\right)$$
$$\text{subject to } y_i - w^T\phi\left(x_i\right) - b \leq \varepsilon + \zeta_i$$
$$w^T\phi\left(x_i\right) + b - y_i \leq \varepsilon + \zeta_i^*$$
$$\zeta_i, \zeta_i^* \geq 0, i = 1,\ldots,n$$

Linear SVC

$$\min_{w,b} \frac{1}{2}w^T w + C\sum_{i=1}\max\left(0, 1 - y_i\left(w^T\phi\left(x_i\right) + b\right)\right)$$

SVM on sci-kit learn

Types of SVMs related analysis supported by Sci-kit Learn

1. `SVC` ,
2. `NuSVC` ,
3. `SVR` ,
4. `NuSVR` ,
5. `LinearSVC`
6. `LinearSVR`
7. `OneClassSVM`

According to sci-kit documentation for sci-py sparce sparse array, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`

SVM basic code implementation is:

Jus then we are ready for prediction:

In [4]:
```
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
```

Out[4]: SVC()

In [6]:
```
clf.predict([[2., 2.]])
```

Out[6]:
```
array([1])
```

How to query the support vectors using sci-kit api?

In [10]:
```
clf.support_vectors_
# get indices of support vectors
clf.support_
```

Out[10]:
```
array([0, 1])
```

In [11]:
```
clf.n_support_
```

Out[11]:
```
array([1, 1])
```

In [12]:
```
clf.n_support_
```

Out[12]:
```
array([1, 1])
```

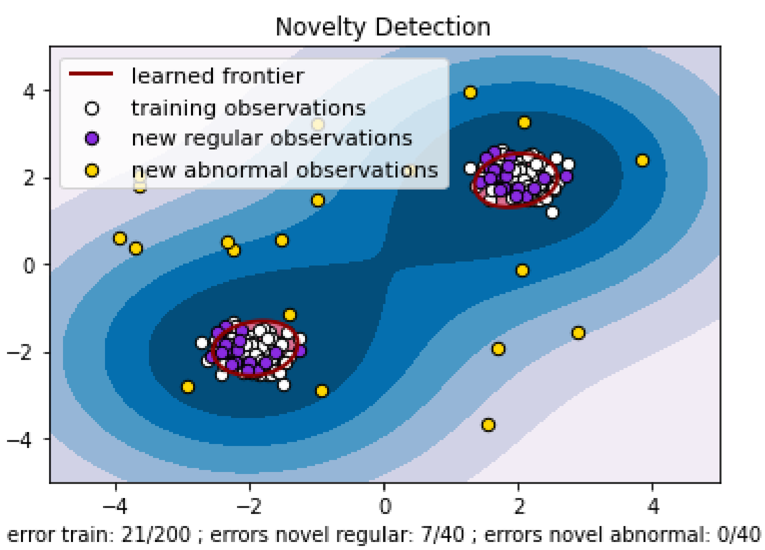
One-Class SVM

It is an unnsupervised algorithm. Used for outliers detection. Supported by sci kit library through this way:

```
from sklearn import svm
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
```

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn import svm

xx, yy = np.meshgrid(np.linspace(-5, 5, 500), np.linspace(-5, 5, 500))
# Generate train data
X = 0.3 * np.random.randn(100, 2)
X_train = np.r_[X + 2, X - 2]
# Generate some regular novel observations
X = 0.3 * np.random.randn(20, 2)
X_test = np.r_[X + 2, X - 2]
# Generate some abnormal novel observations
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))

# fit the model
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf.fit(X_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)
n_error_train = y_pred_train[y_pred_train == -1].size
n_error_test = y_pred_test[y_pred_test == -1].size
n_error_outliers = y_pred_outliers[y_pred_outliers == 1].size

# plot the line, the points, and the nearest vectors to the plane
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
```

```
Z = Z.reshape(xx.shape)

plt.title("Novelty Detection")
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), 0, 7), cmap=plt.cm.PuBu)
a = plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors="darkred")
plt.contourf(xx, yy, Z, levels=[0, Z.max()], colors="palevioletred")

s = 40
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c="white", s=s, edgecolors="k")
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c="blueviolet", s=s, edgecolors="k")
c = plt.scatter(X_outliers[:, 0], X_outliers[:, 1], c="gold", s=s, edgecolors="k")
plt.axis("tight")
plt.xlim((-5, 5))
plt.ylim((-5, 5))
plt.legend(
    [a.collections[0], b1, b2, c],
    [
        "learned frontier",
        "training observations",
        "new regular observations",
        "new abnormal observations",
    ],
    loc="upper left",
    prop=matplotlib.font_manager.FontProperties(size=11),
)
plt.xlabel(
    "error train: %d/200 ; errors novel regular: %d/40 ; errors novel abnormal: %d/40"
    % (n_error_train, n_error_test, n_error_outliers)
)
plt.show()
```



SVM Margins Example (SVC)

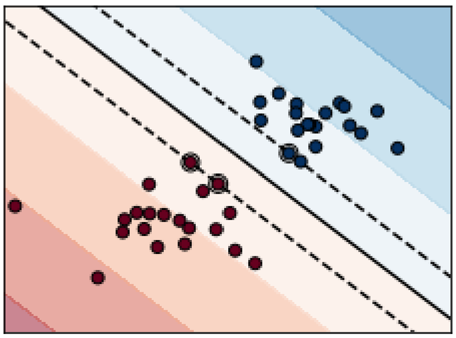
This is the traditional example of SVN, that you will find in every web site and technical blog about the subject. Used for classification Supported by sci kit library through this way:

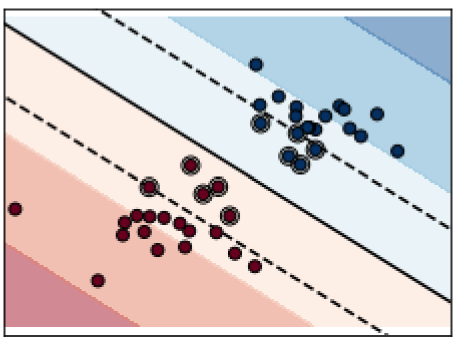
```
from sklearn import svm
clf = svm.SVC(kernel="linear", C=penalty)
```

In [16]:
```
# Code source: Gaël Varoquaux
```

```python
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from sklearn import svm

# we create 40 separable points
np.random.seed(0)
X = np.r_[np.random.randn(20, 2) - [2, 2], np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20

# figure number
fignum = 1

# fit the model
for name, penalty in (("unreg", 1), ("reg", 0.05)):

    clf = svm.SVC(kernel="linear", C=penalty)
    clf.fit(X, Y)

    # get the separating hyperplane
    w = clf.coef_[0]
    a = -w[0] / w[1]
    xx = np.linspace(-5, 5)
    yy = a * xx - (clf.intercept_[0]) / w[1]

    # plot the parallels to the separating hyperplane that pass through the
    # support vectors (margin away from hyperplane in direction
    # perpendicular to hyperplane). This is sqrt(1+a^2) away vertically in
    # 2-d.
    margin = 1 / np.sqrt(np.sum(clf.coef_ ** 2))
    yy_down = yy - np.sqrt(1 + a ** 2) * margin
    yy_up = yy + np.sqrt(1 + a ** 2) * margin

    # plot the line, the points, and the nearest vectors to the plane
    plt.figure(fignum, figsize=(4, 3))
    plt.clf()
    plt.plot(xx, yy, "k-")
    plt.plot(xx, yy_down, "k--")
    plt.plot(xx, yy_up, "k--")

    plt.scatter(
        clf.support_vectors_[:, 0],
        clf.support_vectors_[:, 1],
        s=80,
        facecolors="none",
        zorder=10,
        edgecolors="k",
        cmap=cm.get_cmap("RdBu"),
    )
    plt.scatter(
        X[:, 0], X[:, 1], c=Y, zorder=10, cmap=cm.get_cmap("RdBu"), edgecolors="k"
    )

    plt.axis("tight")
    x_min = -4.8
    x_max = 4.2
    y_min = -6
```

```
    y_max = 6

    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = clf.decision_function(xy).reshape(XX.shape)

    # Put the result into a contour plot
    plt.contourf(XX, YY, Z, cmap=cm.get_cmap("RdBu"), alpha=0.5, linestyles=["-"])

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)

    plt.xticks(())
    plt.yticks(())
    fignum = fignum + 1

plt.show()
```





Non Linear SVM (NuSVR)

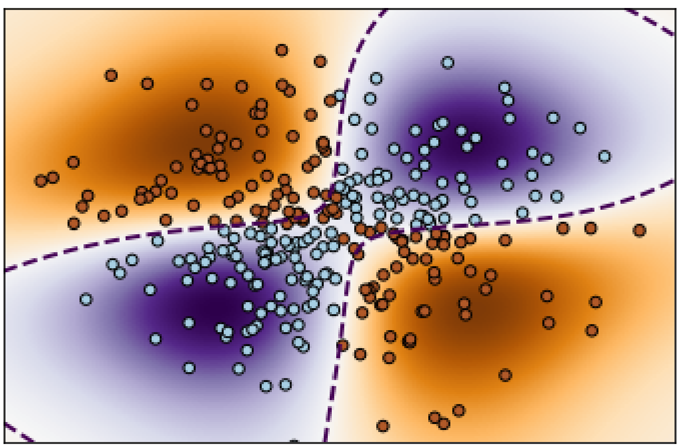
This is the NuSVR implementation in sci kit learn, as expected you will need something like the following:

```
from sklearn import svm
clf = svm.NuSVC(gamma="auto")
```

In [17]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

xx, yy = np.meshgrid(np.linspace(-3, 3, 500), np.linspace(-3, 3, 500))
np.random.seed(0)
X = np.random.randn(300, 2)
Y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)
```

```
# fit the model
clf = svm.NuSVC(gamma="auto")
clf.fit(X, Y)

# plot the decision function for each datapoint on the grid
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    aspect="auto",
    origin="lower",
    cmap=plt.cm.PuOr_r,
)
contours = plt.contour(xx, yy, Z, levels=[0], linewidths=2, linestyles="dashed")
plt.scatter(X[:, 0], X[:, 1], s=30, c=Y, cmap=plt.cm.Paired, edgecolors="k")
plt.xticks(())
plt.yticks(())
plt.axis([-3, 3, -3, 3])
plt.show()
```



Tie-Breaking (Multi-class classification)

This exemplifies OVR (One-vs-Rest) method as the multiclass classification (also known as OVA, One-vs-All)

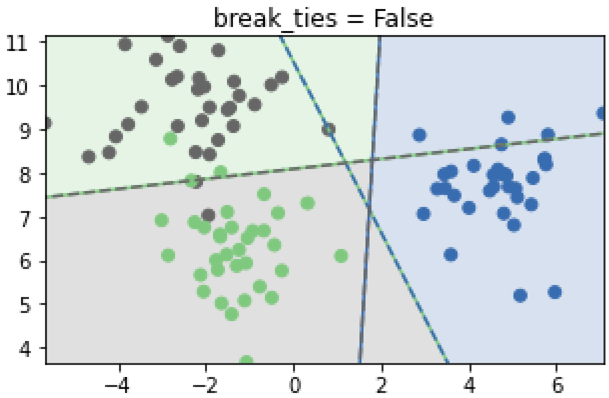
In its most basic type, SVM doesn't support multiclass classification. For multiclass classification, the same principle is utilized after breaking down the multi-classification problem into smaller subproblems, all of which are binary classification problems.

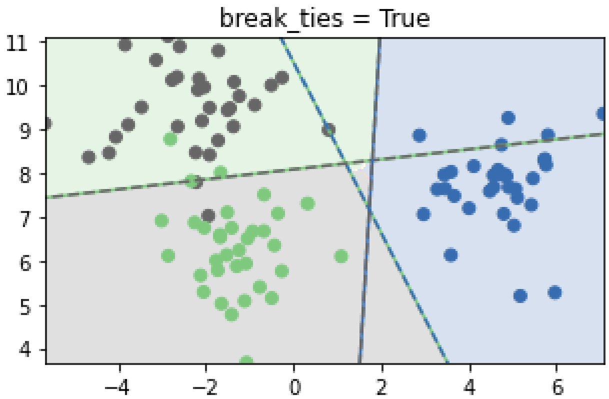
Basic multiclass code declaration for OVR:

```
svm = SVC(
       kernel="linear", C=1, break_ties=break_ties, decision_function_shape="ovr"
   ).fit(X, y)
```

In [18]:
```
# Code source: Andreas Mueller, Adrin Jalali
# License: BSD 3 clause
```

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=27)

fig, sub = plt.subplots(2, 1, figsize=(5, 8))
titles = ("break_ties = False", "break_ties = True")

for break_ties, title, ax in zip((False, True), titles, sub.flatten()):

    svm = SVC(
        kernel="linear", C=1, break_ties=break_ties, decision_function_shape="ovr"
    ).fit(X, y)

    xlim = [X[:, 0].min(), X[:, 0].max()]
    ylim = [X[:, 1].min(), X[:, 1].max()]

    xs = np.linspace(xlim[0], xlim[1], 1000)
    ys = np.linspace(ylim[0], ylim[1], 1000)
    xx, yy = np.meshgrid(xs, ys)

    pred = svm.predict(np.c_[xx.ravel(), yy.ravel()])

    colors = [plt.cm.Accent(i) for i in [0, 4, 7]]

    points = ax.scatter(X[:, 0], X[:, 1], c=y, cmap="Accent")
    classes = [(0, 1), (0, 2), (1, 2)]
    line = np.linspace(X[:, 1].min() - 5, X[:, 1].max() + 5)
    ax.imshow(
        -pred.reshape(xx.shape),
        cmap="Accent",
        alpha=0.2,
        extent=(xlim[0], xlim[1], ylim[1], ylim[0]),
    )

    for coef, intercept, col in zip(svm.coef_, svm.intercept_, classes):
        line2 = -(line * coef[1] + intercept) / coef[0]
        ax.plot(line2, line, "-", c=colors[col[0]])
        ax.plot(line2, line, "--", c=colors[col[1]])
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    ax.set_title(title)
    ax.set_aspect("equal")

plt.show()
```





SVM: Separating hyperplane for unbalanced classes¶

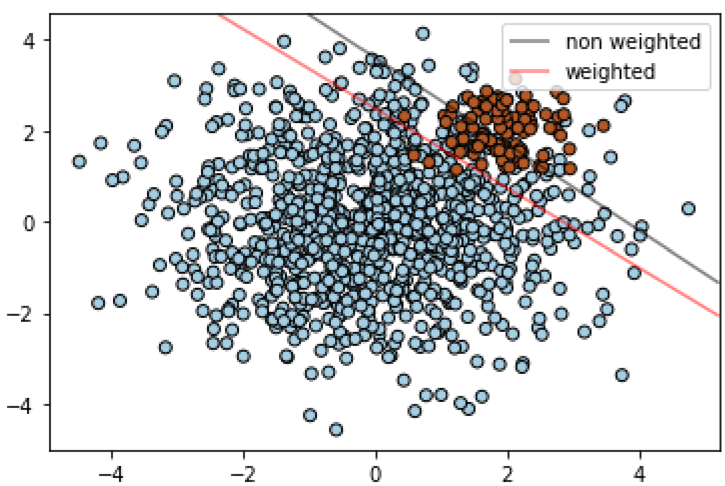
This technique is useful for unbalanced datasets:

In [ ]:
```
Looks like for this case of analysis, this is the key line:
```

```
wclf = svm.SVC(kernel="linear", class_weight={1: 10})
wclf.fit(X, y)
```

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs

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

# plot the decision functions for both classifiers
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T

# get the separating hyperplane
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
a = ax.contour(XX, YY, Z, colors="k", levels=[0], alpha=0.5, linestyles=["-"])

# get the separating hyperplane for weighted classes
Z = wclf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins for weighted classes
b = ax.contour(XX, YY, Z, colors="r", levels=[0], alpha=0.5, linestyles=["-"])

plt.legend(
    [a.collections[0], b.collections[0]],
    ["non weighted", "weighted"],
    loc="upper right",
)
plt.show()
```



SVM: Weighted samples

This seemingly is more a technique to "emphasize" dataset points based on pre-defined weight.

The effects of this technique is to perceive those points with greater weight bigger in the employed plot.

This part of the plot increase the weight of some outliers:

```
# and bigger weights to some outliers
sample_weight_last_ten[15:] *= 5
sample_weight_last_ten[9] *= 15
```

This is the way SVM class is initialized for the weighted points:

```
# fit the model
clf_weights = svm.SVC(gamma=1)
clf_weights.fit(X, y, sample_weight=sample_weight_last_ten)
```

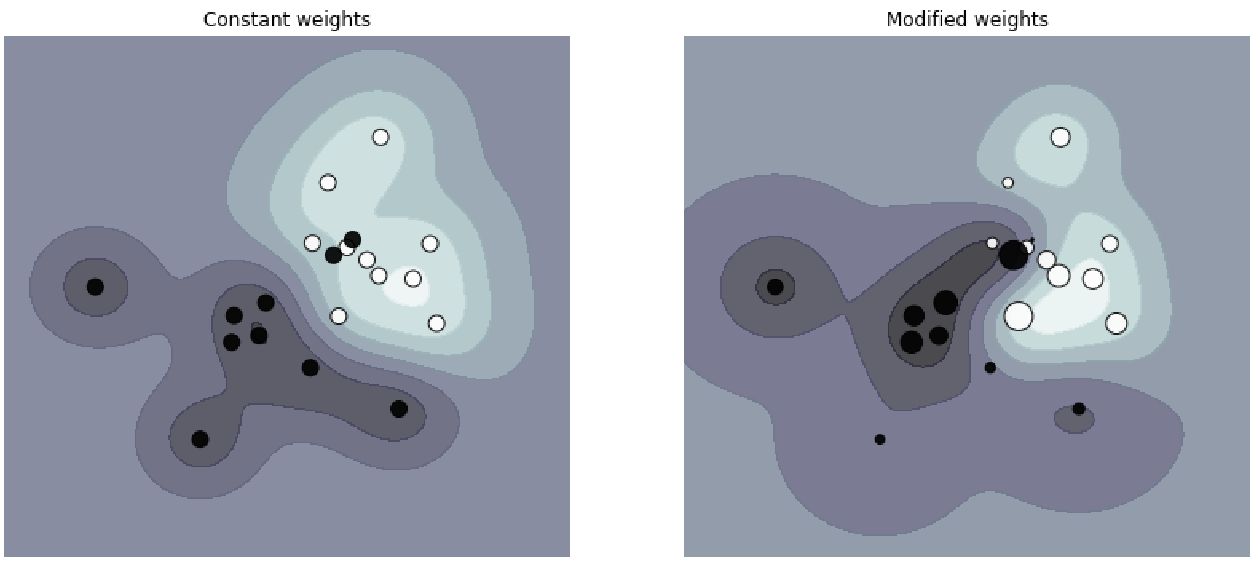
no different for the no weighted ones:

```
clf_no_weights = svm.SVC(gamma=1)
clf_no_weights.fit(X, y)
```

In [21]:

```
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
```

```
# for reference, first fit without sample weights

# fit the model
clf_weights = svm.SVC(gamma=1)
clf_weights.fit(X, y, sample_weight=sample_weight_last_ten)

clf_no_weights = svm.SVC(gamma=1)
clf_no_weights.fit(X, y)

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()
```



Outlier detection on a real data set

This technique points out the importance a robust covariance estimation has for outlier detection. The One-Class SVM is in some way better since it does does assume any parametric form of the data distribution and can therefore model the complex shape of the data much better.

Important to mention the use of *OnClassSVM* commonly used to run outlier detection.

Another important piece of the code is: *sklearn.covariance.EllipticEnvelope*¶

EllipiticEnvelope class

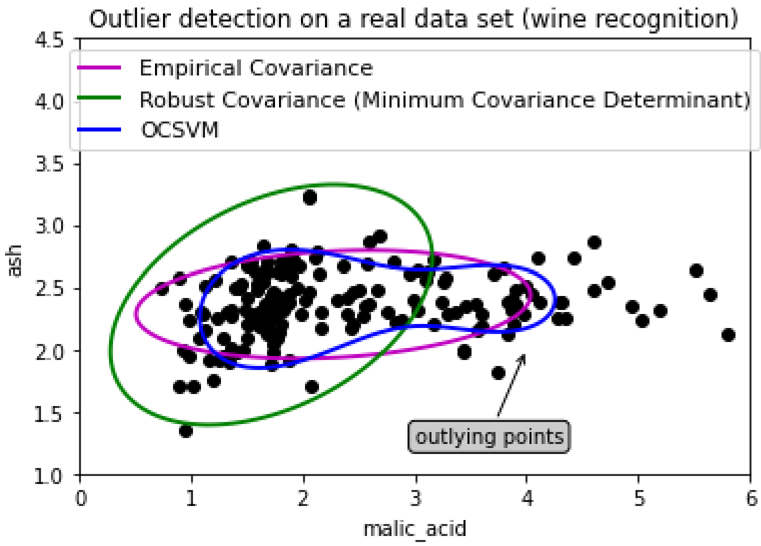
class sklearn.covariance.EllipticEnvelope(, store_precision=True, assume_centered=False, support_fraction=None, contamination=0.1, random_state=None)¶

An object for detecting outliers in a Gaussian distributed dataset.

In [22]:
```
# Author: Virgile Fritsch <virgile.fritsch@inria.fr>
# License: BSD 3 clause

import numpy as np
```

```
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn.datasets import load_wine

# Define "classifiers" to be used
classifiers = {
    "Empirical Covariance": EllipticEnvelope(support_fraction=1.0, contamination=0.25),
    "Robust Covariance (Minimum Covariance Determinant)": EllipticEnvelope(
        contamination=0.25
    ),
    "OCSVM": OneClassSVM(nu=0.25, gamma=0.35),
}
colors = ["m", "g", "b"]
legend1 = {}
legend2 = {}

# Get data
X1 = load_wine()["data"][:, [1, 2]]  # two clusters

# Learn a frontier for outlier detection with several classifiers
xx1, yy1 = np.meshgrid(np.linspace(0, 6, 500), np.linspace(1, 4.5, 500))
for i, (clf_name, clf) in enumerate(classifiers.items()):
    plt.figure(1)
    clf.fit(X1)
    Z1 = clf.decision_function(np.c_[xx1.ravel(), yy1.ravel()])
    Z1 = Z1.reshape(xx1.shape)
    legend1[clf_name] = plt.contour(
        xx1, yy1, Z1, levels=[0], linewidths=2, colors=colors[i]
    )

legend1_values_list = list(legend1.values())
legend1_keys_list = list(legend1.keys())

# Plot the results (= shape of the data points cloud)
plt.figure(1)  # two clusters
plt.title("Outlier detection on a real data set (wine recognition)")
plt.scatter(X1[:, 0], X1[:, 1], color="black")
bbox_args = dict(boxstyle="round", fc="0.8")
arrow_args = dict(arrowstyle="->")
plt.annotate(
    "outlying points",
    xy=(4, 2),
    xycoords="data",
    textcoords="data",
    xytext=(3, 1.25),
    bbox=bbox_args,
    arrowprops=arrow_args,
)
plt.xlim((xx1.min(), xx1.max()))
plt.ylim((yy1.min(), yy1.max()))
plt.legend(
    (
        legend1_values_list[0].collections[0],
        legend1_values_list[1].collections[0],
        legend1_values_list[2].collections[0],
    ),
    (legend1_keys_list[0], legend1_keys_list[1], legend1_keys_list[2]),
    loc="upper center",
```

```
    prop=matplotlib.font_manager.FontProperties(size=11),
)
plt.ylabel("ash")
plt.xlabel("malic_acid")

plt.show()
```



Plot Different SVM Classifiers in the iris dataset

This article highlights the difference and effects of using the different kernels. Key code segment is the one:

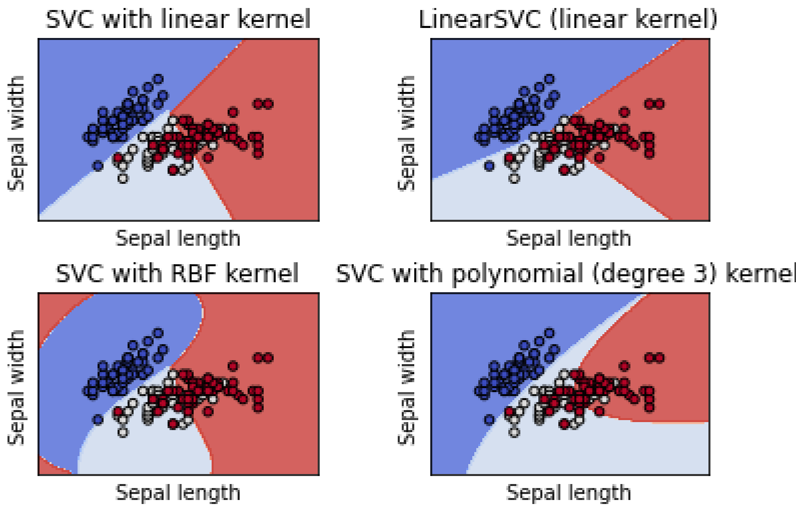
```
# we create an instance of SVM and fit out data. We do not scale ou
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
models = (
    svm.SVC(kernel="linear", C=C),
    svm.LinearSVC(C=C, max_iter=10000),
    svm.SVC(kernel="rbf", gamma=0.7, C=C),
    svm.SVC(kernel="poly", degree=3, gamma="auto", C=C),
)
```

In [24]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets


def make_meshgrid(x, y, h=0.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
```

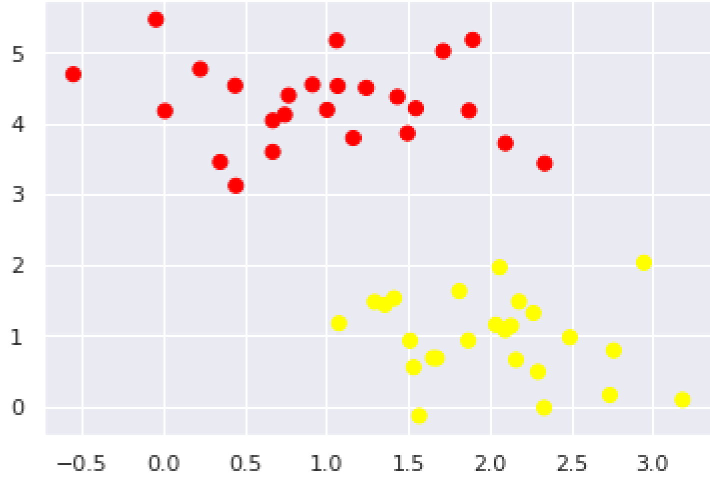
```
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


# import some data to play with
iris = datasets.load_iris()
# 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=3, 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]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
```

```
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel("Sepal length")
    ax.set_ylabel("Sepal width")
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)

plt.show()
```



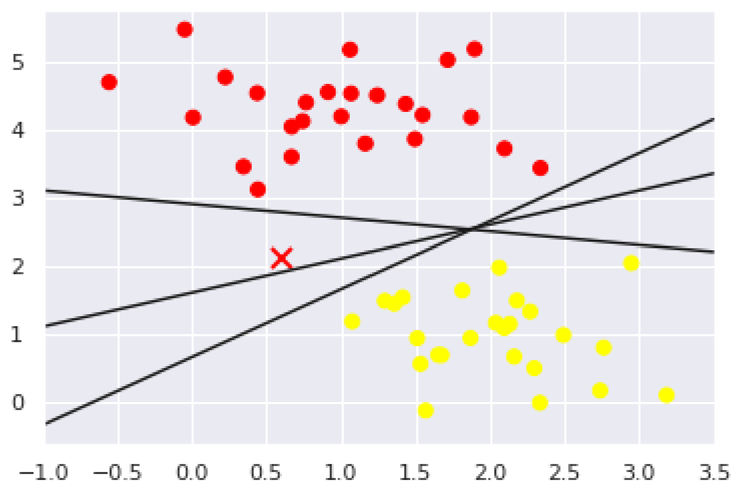
Problem 2: Application Case

In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



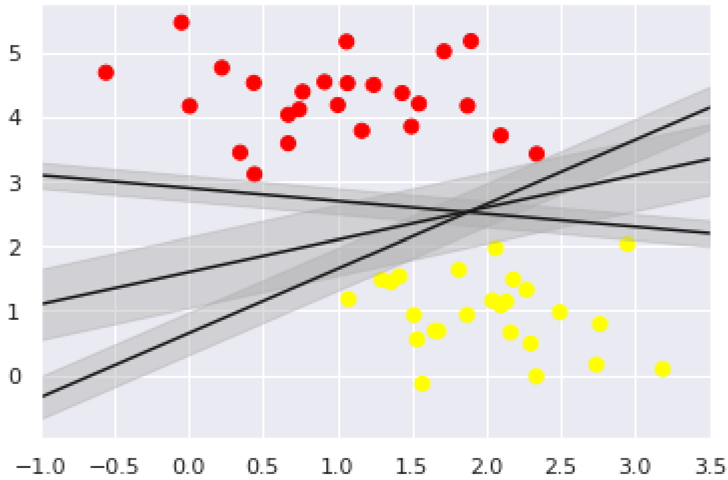
Solo estamos generando los datos que pueden ser separados linealmente.

In [ ]:
```
import numpy as np
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(xfit, m * xfit + b, '-k')

plt.xlim(-1, 3.5);
```



Se dibujaron lineas en el diagrama de puntos para mostrar que son linealmente separables los datos

In [ ]:
```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
```



A las lineas divisoras se les agregó márgenes hasta el punto más cercano (los support vectors)

Fitting SVM

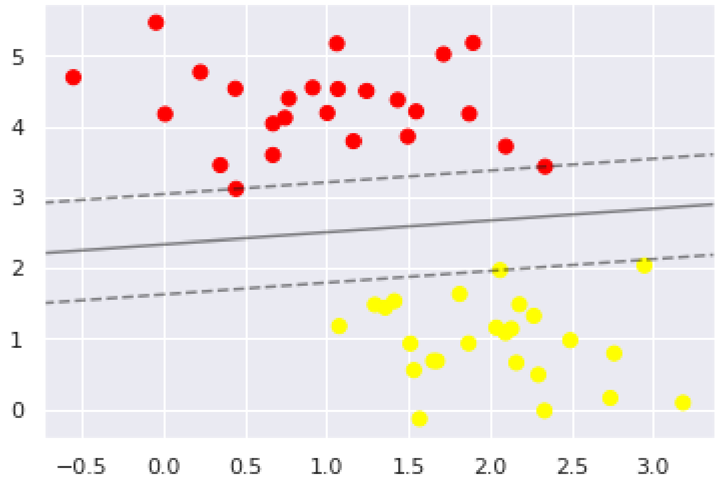
In [ ]:
```
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

Out[ ]:
```
SVC(C=10000000000.0, kernel='linear')
```

Creamos el modelito svm

In [ ]:
```
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

In [ ]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```

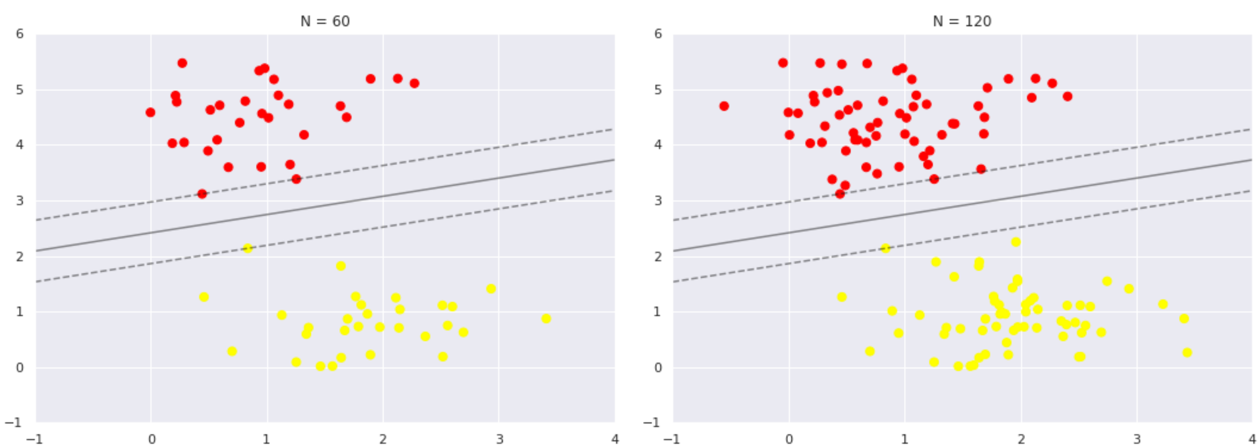


Definimos una función para graficar que tome en cuenta los support vectors y que marque los límites según el modelo usado.

In [ ]:
```
model.support_vectors_
```

Out[ ]:
```
array([[0.44359863, 3.11530945],
       [2.33812285, 3.43116792],
       [2.06156753, 1.96918596]])
```

In [ ]:
```
def plot_svm(N=10, ax=None):
    X, y = make_blobs(n_samples=200, centers=2,
                      random_state=0, cluster_std=0.60)
    X = X[:N]
    y = y[:N]
    model = SVC(kernel='linear', C=1E10)
    model.fit(X, y)

    ax = ax or plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    ax.set_xlim(-1, 4)
    ax.set_ylim(-1, 6)
    plot_svc_decision_function(model, ax)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)
for axi, N in zip(ax, [60, 120]):
    plot_svm(N, axi)
    axi.set_title('N = {0}'.format(N))
```

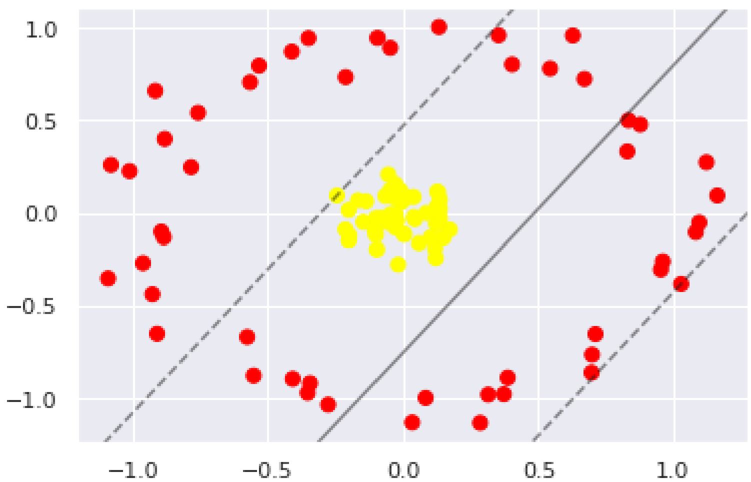


En estas gráficas se demuestran que el modelo se crea con base en los vectores de soporte y al agregar nuevos datos no se modifica el fit del modelo (donde podría variar la cantidad de support vectors o el tamaño de w).

In [ ]:
```
from ipywidgets import interact, fixed
interact(plot_svm, N=[10, 200], ax=fixed(None));
```

Beyond linear boundaries: Kernel SVM

In [ ]:
```
from sklearn.datasets import make_circles
X, y = make_circles(100, factor=.1, noise=.1)

clf = SVC(kernel='linear').fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf, plot_support=False);
```



Se definen el dataset para el caso de una forma no linealmente separable, en la gráfica se muestra lo mencionado previamente.

In [ ]:
```
r = np.exp(-(X ** 2).sum(1))
```

Se crea una base radial que se centra en el medio de los datos para poder crear una proyección de los datos en una nueva dimensión.

In [ ]:
```
from mpl_toolkits import mplot3d

def plot_3D(elev=30, azim=30, X=X, y=y):
    ax = plt.subplot(projection='3d')
    ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='autumn')
    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('r')

interact(plot_3D, elev=[-150, 150], azip=(-180, 180),
         X=fixed(X), y=fixed(y));
```

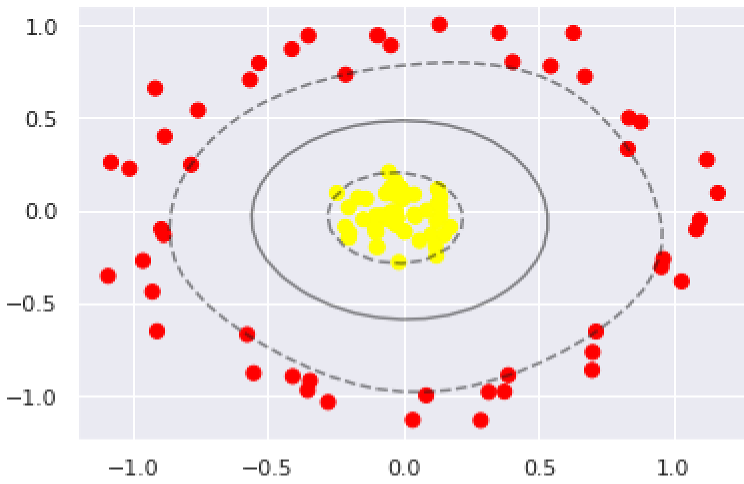
Gráfica de los datos incluída la nueva dimensión.

In [ ]:
```
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

Out[ ]: SVC(C=1000000.0)

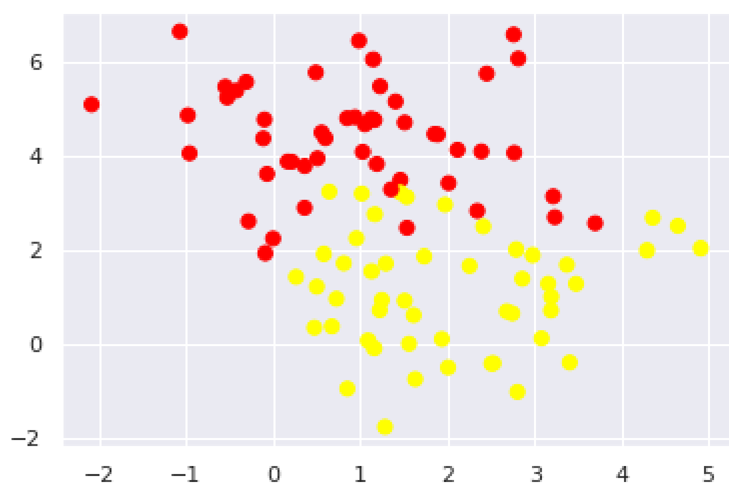
Ahora pasamos a crear un kernel no lineal

In [ ]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=300, lw=1, facecolors='none');
```



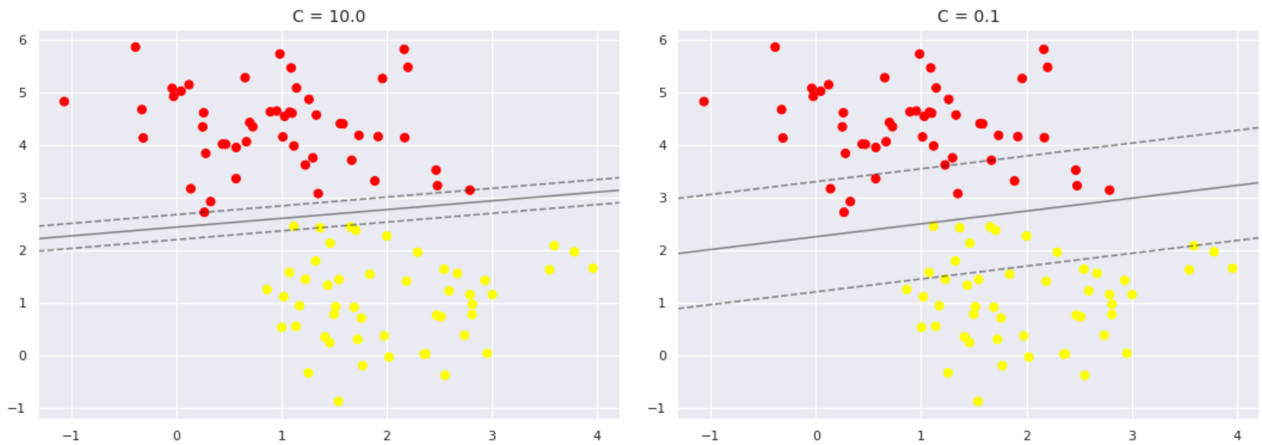
In [ ]:
```
Esta máquina transfomra los métodos lienales en nolineales, donde se puede aplicar el t
```

In [ ]:
```
X, y = make_blobs(n_samples=100, centers=2,
                  random_state=0, cluster_std=1.2)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



In [ ]:
```
X, y = make_blobs(n_samples=100, centers=2,
                  random_state=0, cluster_std=0.8)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)
```

```python
for axi, C in zip(ax, [10.0, 0.1]):
    model = SVC(kernel='linear', C=C).fit(X, y)
    axi.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    plot_svc_decision_function(model, axi)
    axi.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=300, lw=1, facecolors='none');
    axi.set_title('C = {0:.1f}'.format(C), size=14)
```



En este modelo, el hiperparámetro C es el que controla los márgenes, donde una C grande, haría que los márgenes fueran restrictivos o muy fijos. Si la C es pequeña, quedan suaves y permiten más puntos.