Linear SVM

Import some necessary libraries:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
```

Generate data and preprocessing:

```
# Generate a simple dataset
X, y = make_blobs(n_samples=200, centers=2, random_state=0, cluster_std=0.6)
# split dataset into random train and test subsets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

Fit the model:

```
from sklearn.svm import SVC # "Support Vector Classifier"
clf = SVC(kernel='linear')
#Train the model using the training sets
clf.fit(X_train, y_train)
```

Make predictions on testing data and evaluate the predations

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)

from sklearn import metrics
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Precision: 1.0
Recall: 1.0
Accuracy: 1.0
```

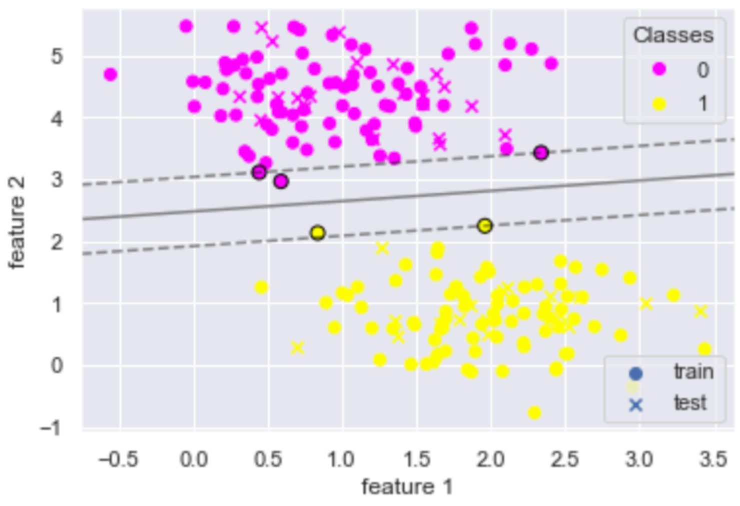
Visualization

Use the following function to plot hyperplane and support vectors:

```
def plot_svc_decision_function(clf, ax=None):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    x = np.linspace(plt.xlim()[0], plt.xlim()[1], 30)
    y = np.linspace(plt.ylim()[0], plt.ylim()[1], 30)
    Y, X = np.meshgrid(y, x)
    P = np.zeros_like(X)
    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            P[i, j] = clf.decision_function([[xi, yj]])
    # plot the margins
    ax.contour(X, Y, P, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

To visualize the data, hyperplane and support vectors using the flowing script:

```
fig, ax = plt.subplots()
#draw the train and test data
train = ax.scatter(X_train[:,0], X_train[:,1], c=y_train, marker='o',cmap='spring')
test = ax.scatter(X_test[:,0], X_test[:,1], c=y_test, marker='x',cmap='spring')

# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*train.legend_elements(), title="Classes",loc="upper right")
ax.add_artist(legend1)

# produce a legend for different marks
legend2 = ax.legend((train, test),('train', 'test'), loc="lower right")
plt.xlabel('feature 1')
plt.ylabel('feature 2')

# plot hyperplane and support vectors.
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],s=50, edgecolors='black', facecolors='none')
plt.show()
```

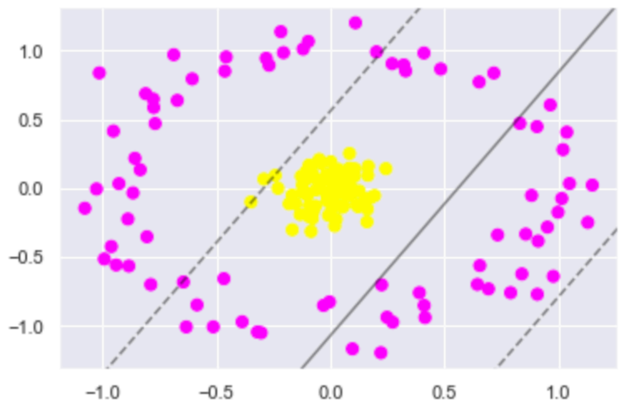


Kernel Methods:

Generate a linear inseparable data and apply linear SVM to it:

```
from sklearn.datasets import make_circles
X, y = make_circles(200, factor=.1, noise=.1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# fit a linear SVM
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=50, cmap='spring')
plot_svc_decision_function(clf);
```

```
Precision: 0.5277777777777778
Recall: 1.0
Accuracy: 0.575
```

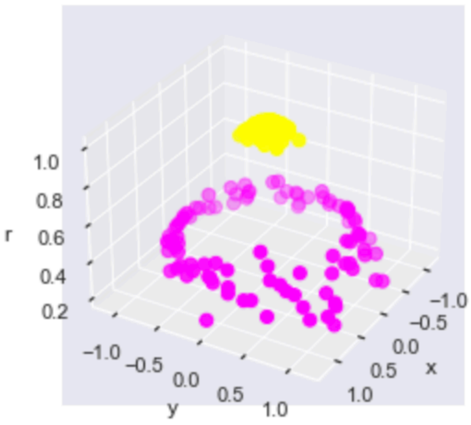


Now we use the radial basis function:

```
# radial basis function
r = np.exp(-(X[:, 0] ** 2 + X[:, 1] ** 2))
```

Mapping to higher dimension, the data becomes trivially linearly separable:

```python
from mpl_toolkits import mplot3d
def plot_3D(elev=30, azim=30):
    ax = plt.subplot(projection='3d')
    ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='spring')
    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('r')
plot_3D()
```

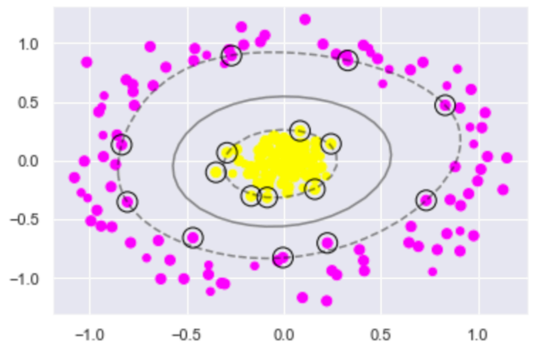


Fit a SVM with rbf kernel:

```python
clf = SVC(kernel='rbf')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=50, cmap='spring')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, s=30, cmap='spring')
plot_svc_decision_function(clf);

plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=200, edgecolors='black', facecolors='none')
```

```
Precision: 1.0
Recall: 1.0
Accuracy: 1.0
```

```
<matplotlib.collections.PathCollection at 0x11c104610>
```



The possible value for kernel function can be any of the following:

- 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 γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma\langle x, x'\rangle + r)$, where r is specified by `coef0`.

Multi-class SVM:

Import the Iris dataset:

```
from sklearn import svm, datasets
# 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
```

Create a of SVM with polynomial kernel with and fit out data:

```
C = 1.0  # SVM regularization parameter
clf = svm.SVC(kernel='poly', degree=3, gamma='auto', C=C)
models = clf.fit(X, y)
```

Where the parameter gamma is the kernel coefficient for 'rbf', 'poly' and 'sigmoid'. C *is the* regularization parameter.

Use the following function to plot the decision boundaries for a classifier:

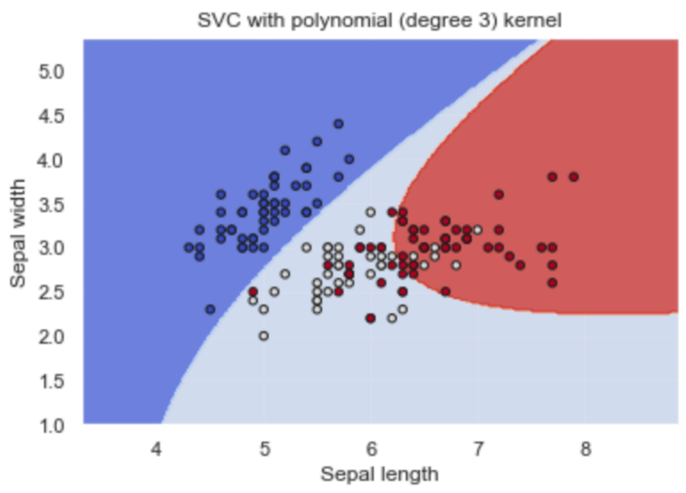
```
def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    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
```

Use the following function to create a mesh of point to plot in:

```
def make_meshgrid(x, y, h=.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
```

To plot the classification result use the following script:

```
# title for the plots
title = 'SVC with polynomial (degree 3) kernel'
fig, ax = plt.subplots()
X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)
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_title(title)
plt.show()
```


SVC with polynomial (degree 3) kernel
Sepal width
Sepal length