ML Hands-on Workshop @ Elec, SFIT

Instructor: Santosh Chapaneri

Jan 2022

In [25]:
```
import numpy as np
import pandas as pd
import sklearn.svm as svm
import matplotlib.pyplot as plt
# %matplotlib inline
```

Support Vector Machines (SVM)

- We generate 2D points and assign a binary label according to a linear operation on the coordinates.

In [26]:
```
X = np.random.randn(200, 2)
y = (X[:, 0] + X[:, 1]) > 1 # Imagine an AND function

X[:10], y[:10]
```

Out[26]:
```
(array([[ 1.10182890e+00,  1.63997803e-01],
        [ 5.54692088e-01,  5.12114836e-01],
        [-1.90611261e-02, -9.13936739e-01],
        [ 5.40503603e-01, -1.44962686e-01],
        [-9.66361265e-01, -4.05920547e-01],
        [ 1.02819928e-01,  5.06715832e-03],
        [-8.09612152e-01,  2.68775885e+00],
        [-7.71194000e-01, -9.14522368e-01],
        [-8.14323032e-01,  2.11823106e-03],
        [-2.31353109e-01, -4.90825246e-01]]),
 array([ True,  True, False, False, False, False,  True, False, False,
        False]))
```

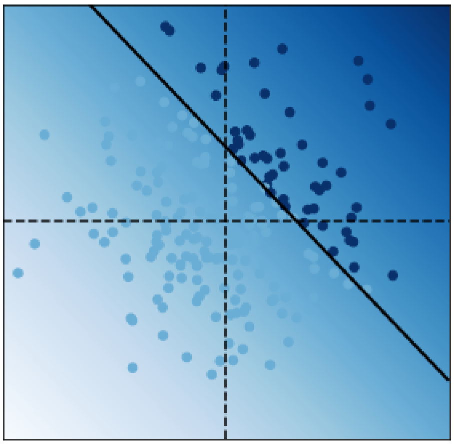
- Fit a linear Support Vector Classifier (SVC)

In [27]:
```
est = svm.LinearSVC()
est.fit(X, y)
```

Out[27]:
```
LinearSVC()
```

In [28]:
```
# Generate a grid in the square [-3,3 ]^2
xx, yy = np.meshgrid(np.linspace(-3, 3, 500),
                     np.linspace(-3, 3, 500))

# This function takes a SVM estimator as input
def plot_decision_function(est, X, y):
    # We evaluate the decision function on the grid.
    Z = est.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    cmap = plt.cm.Blues

    # Display the decision function on the grid
    plt.figure(figsize=(5,5));
    plt.imshow(Z,
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               aspect='auto', origin='lower', cmap=cmap)

    # Display the boundaries
    plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='k')

    # Display the points with their true labels
    plt.scatter(X[:, 0], X[:, 1], s=30, c=.5+.5*y, lw=1,
               cmap=cmap, vmin=0, vmax=1);
    plt.axhline(0, color='k', ls='--')
    plt.axvline(0, color='k', ls='--')
    plt.xticks(())
    plt.yticks(())
    plt.axis([-3, 3, -3, 3])
```

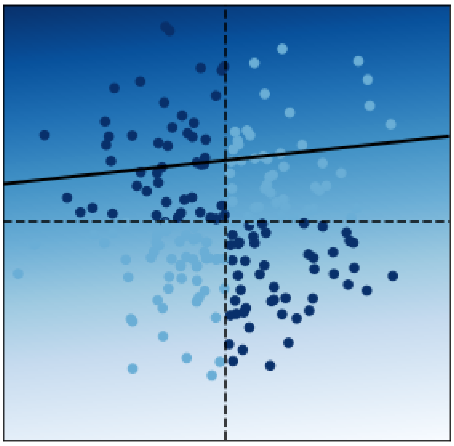
In [29]:
```
plot_decision_function(est, X, y);
#plt.title("Linearly separable, linear SVC")
```



The linear SVC tried to separate the points with a line and it did a good job.

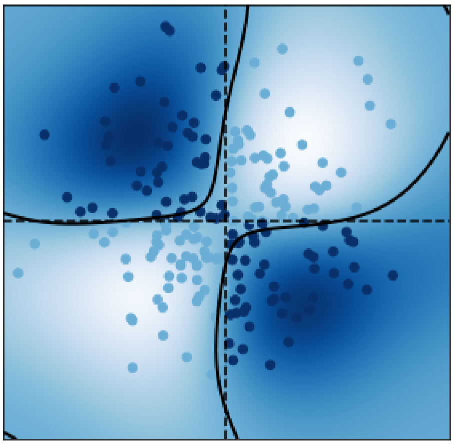
- We now modify the labels with a XOR function.
- A point's label is 1 if the coordinates have different signs. This classification is not linearly separable. Therefore, a linear SVC fails completely.

In [30]:
```
y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)

est2 = svm.LinearSVC()
est2.fit(X, y)

plot_decision_function(est2, X, y)
#plt.title("XOR, linear SVC")
```

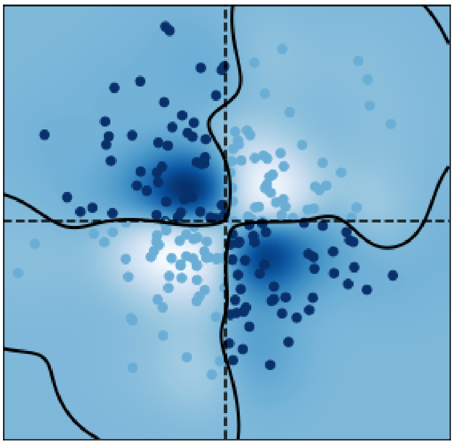


- It is possible to use non-linear SVCs by using non-linear kernels.
- Kernels specify a non-linear transformation of the points into a higher-dimensional space. Transformed points in this space are assumed to be more linearly separable, although they are not necessarily in the original space.
- By default, the SVC classifier in scikit-learn uses the Radial Basis Function (RBF) kernel.

In [31]:
```
est3 = svm.SVC()
est3.fit(X, y)

plot_decision_function(est3, X, y)
```



In [32]:
```
from sklearn.model_selection import GridSearchCV

est4 = GridSearchCV(svm.SVC(),
                    {'C': np.logspace(-3., 3., 10),
                     'gamma': np.logspace(-3., 3., 10)}, cv=5);
est4.fit(X, y)
#print("Score: {0:.3f}".format(cv.cross_val_score(est4, X, y).mean()))

plot_decision_function(est4.best_estimator_, X, y)
```



Applying SVM on IRIS dataset

In [33]:
```
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
y = iris.target
```

- Let's do **hyper-parameter tuning**
- Use **5-fold cross validation to perform grid search to calculate optimal hyper-parameters**

In [34]:
```
from sklearn.model_selection import GridSearchCV
import sklearn.model_selection as cv
from sklearn.metrics import classification_report

# Split the dataset
X_train, X_test, y_train, y_test = cv.train_test_split(X, y, test_size=0.25)
```

In [35]:
```
# Set the parameters by cross-validation
parameters = [{'kernel': ['rbf'],
               'gamma': [1e-4, 1e-3, 0.01, 0.1, 0.2, 0.5],
               'C': [1, 10, 100, 1000]},
              {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

print("# Tuning hyper-parameters")
print()

clf = GridSearchCV(svm.SVC(decision_function_shape='ovr'), parameters, cv=5)
clf.fit(X_train, y_train)
```

```
# Tuning hyper-parameters
```

Out[35]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid=[{'C': [1, 10, 100, 1000],
                          'gamma': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.5],
                          'kernel': ['rbf']},
                         {'C': [1, 10, 100, 1000], 'kernel': ['linear']}])
```

In [36]:
```
print("Best parameters set found on training set:")
print()
print(clf.best_params_)
```

```
Best parameters set found on training set:

{'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}
```

In [37]:
```
y_true, y_pred = y_test, clf.predict(X_test)
print(classification_report(y_true, y_pred))
# The support is the number of occurrences of each class in y_true
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.77      1.00      0.87        10
           2       1.00      0.75      0.86        12

    accuracy                           0.92        38
   macro avg       0.92      0.92      0.91        38
weighted avg       0.94      0.92      0.92        38
```

In [38]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
```

Out[38]: 0.9210526315789473