CDS503: Machine Learning

LAB 5: Support Vector Machine (SVM)

Support vector machines (SVMs) are a particularly powerful and flexible class of supervised algorithms for both classification and regression. In this section, we will develop the intuition behind support vector machines and their use in classification problems.

We begin with the standard imports:

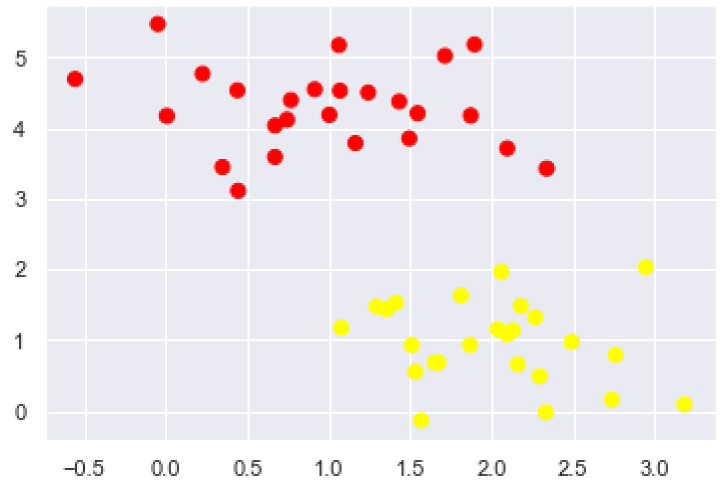
In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()
```

Motivating Support Vector Machines

We learned a simple model describing the distribution of each underlying class, and used these generative models to probabilistically determine labels for new points. That was an example of *generative classification*; here we will consider instead *discriminative classification*: rather than modeling each class, we simply find a line or curve (in two dimensions) or manifold (in multiple dimensions) that divides the classes from each other.

As an example of this, consider the simple case of a classification task, in which the two classes of points are well separated:

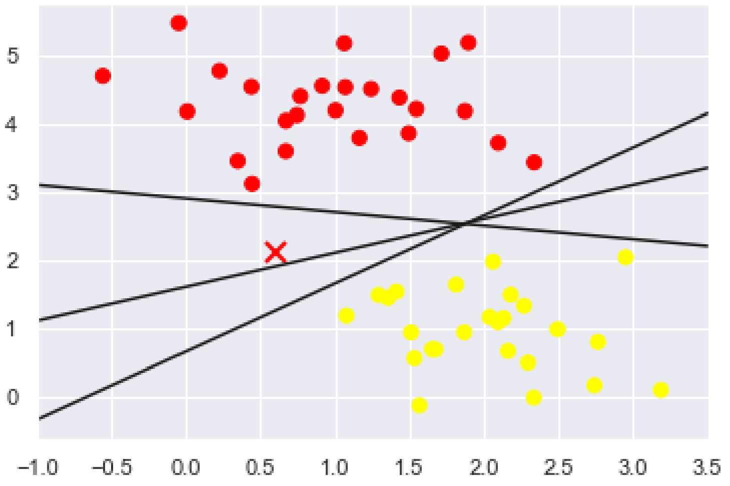
In [2]:
```
from sklearn.datasets import make_blobs # library to make random data blobs

# define the data that clearly seperated
X, y = make_blobs(n_samples=50, centers=2, random_state=0, cluster_std=0.60)
# plot the data using scatter chart
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



A *linear discriminative* classifier would attempt to draw a straight line separating the two sets of data, and thereby create a **model** for *classification*. For two dimensional data like that shown here, this is a task we could do by hand. But immediately we see a problem: there is more than one possible dividing line that can perfectly discriminate between the two classes!

We can draw them as follows:

In [3]:
```
# draw a best-fit line
xfit = np.linspace(-1, 3.5)

# plot the data using scatter chart
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
# mark the possible line point
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

# try three different data on the best-fit line
for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    # plot y = mx + b
    plt.plot(xfit, m * xfit + b, '-k')

# display within this x limits
plt.xlim(-1, 3.5);
```

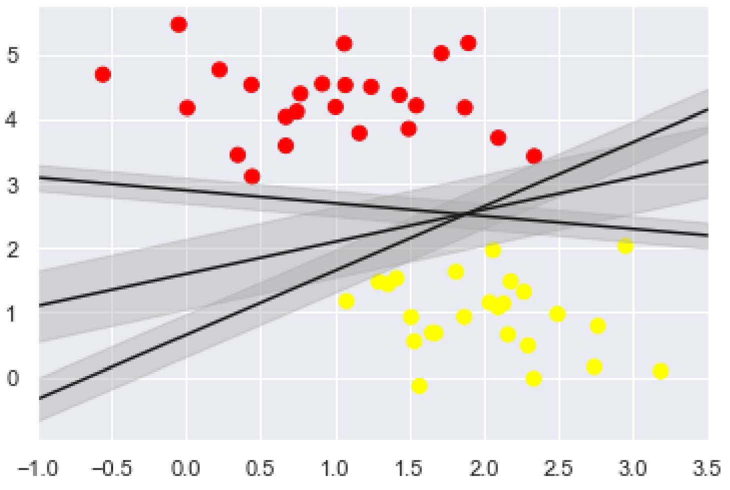


These are three *very* different separators which, nevertheless, perfectly discriminate between these samples. Depending on which you choose, a new data point (e.g., the one marked by the "X" in this plot) will be assigned a different label! Evidently our simple intuition of "drawing a line between classes" is not enough, and we need to think a bit deeper.

Support Vector Machines: Maximizing the *Margin*

Support vector machines offer one way to improve on this. The intuition is this: rather than simply drawing a zero-width line between the classes, we can draw around each line a *margin* of some width, up to the nearest point. Here is an example of how this might look:

In [4]:
```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

# use three data points on the best-fit line with some margin
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')
    # fill the margin with color
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)

# display within this x limits
plt.xlim(-1, 3.5);
```



In support vector machines, the line that maximizes this margin is the one we will choose as the optimal model. **Support vector machines** are an example of such a *maximum margin* estimator.

Fitting a support vector machine

Let's see the result of an actual fit to this data: we will use `sklearn` support vector classifier to train an SVM model on this data. For the time being, we will use a `linear` kernel and set the `C` parameter to a very large number (we'll discuss the meaning of these in more depth momentarily).

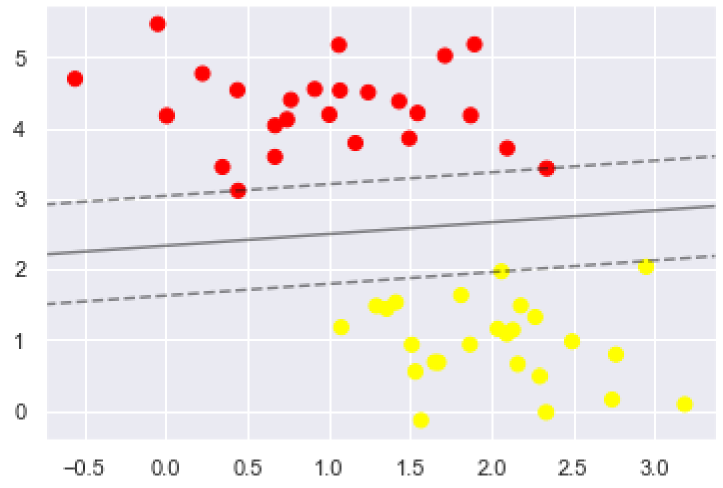
In [5]:
```
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

Out[5]:
```
▾ SVC

SVC(C=10000000000.0, kernel='linear')
```

To better *visualize* what's happening here, let's create a *quick* convenience function that will plot SVM decision boundaries for us:

In [6]:
```python
"""
Plot the decision function for a 2D SVC
"""
def plot_svc_decision_function(model, ax=None, plot_support=True):
    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 [7]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```



This is the dividing line that maximizes the margin between the two sets of points. Notice that a few of the training points just touch the margin: they are indicated by the black circles in this figure. These points are the pivotal elements of this fit, and are known as the *support vectors*, and give the algorithm its name. In Scikit-Learn, the identity of these points are stored in the `support_vectors_` attribute of the classifier:

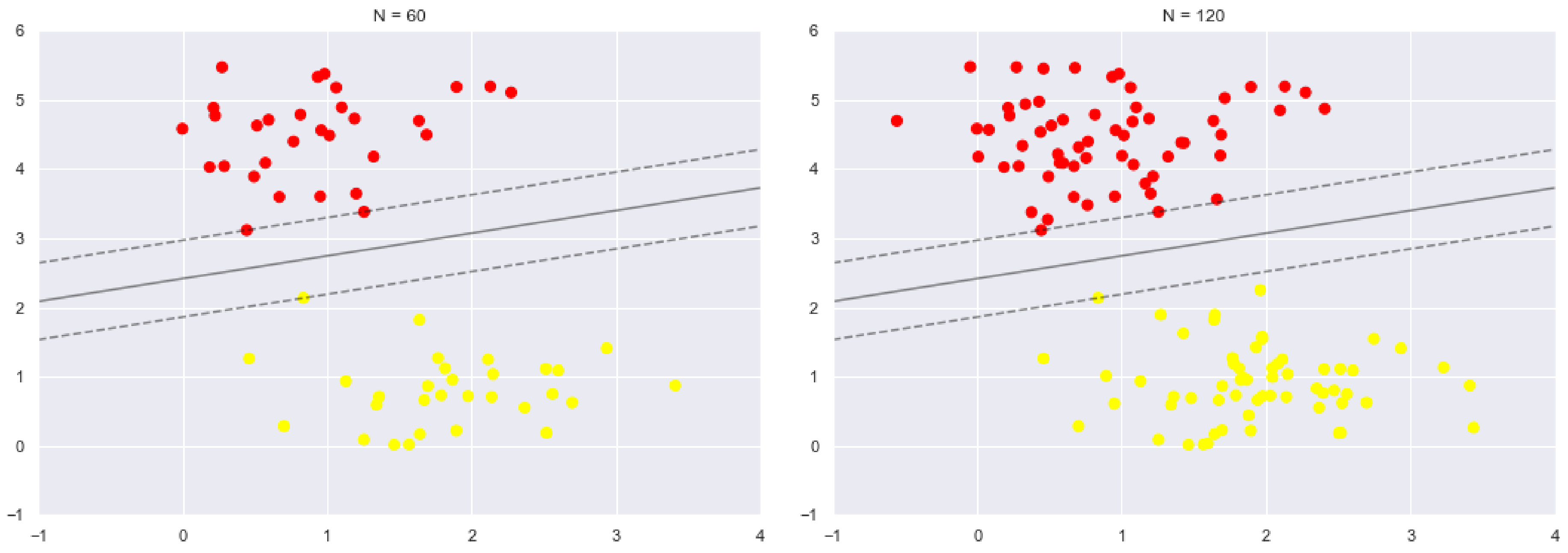
In [8]:
```
model.support_vectors_
```

Out[8]:
```
array([[0.44359863, 3.11530945],
       [2.33812285, 3.43116792],
       [2.06156753, 1.96918596]])
```

A key to this classifier's success is that for the fit, only the position of the support vectors matter; any points further from the margin which are on the correct side do not modify the fit! Technically, this is because these points do not contribute to the loss function used to fit the model, so their position and number do not matter so long as they do not cross the margin.

We can see this, for example, if we plot the model learned from the first 60 points and first 120 points of this dataset:

In [9]:
```
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))
```

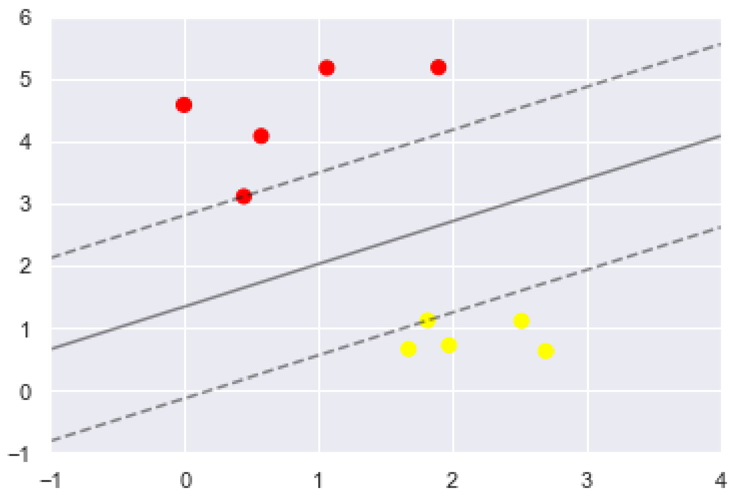


In the left panel, we see the model and the support vectors for 60 training points. In the right panel, we have doubled the number of training points, but the model has not changed: the three support vectors from the left panel are still the support vectors from the right panel. This insensitivity to the exact behavior of distant points is one of the strengths of the SVM model.

If you are running this notebook live, you can use IPython's interactive widgets to view this feature of the SVM model interactively:

In [10]:
```
# library to make an interactive python notebook widget
from ipywidgets import interact, fixed

# plot the interactive chart
interact(plot_svm, N=[10, 30, 50, 70, 100, 120, 150, 200], ax=fixed(None));
```

N 10

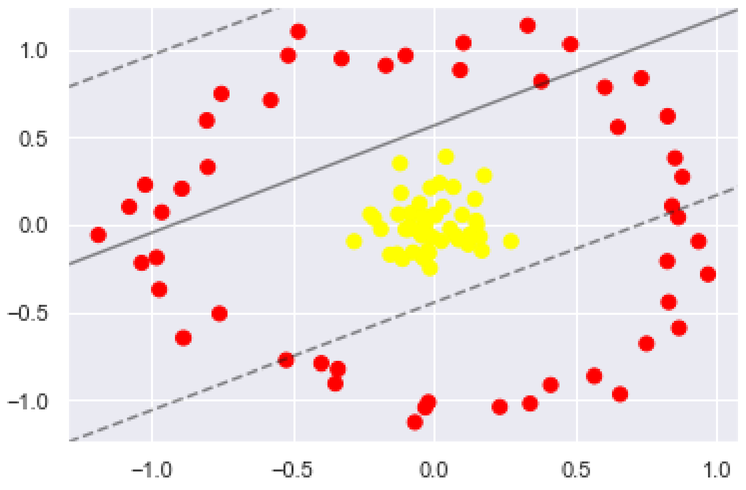


Beyond linear boundaries: Kernel SVM

Where **SVM** becomes *extremely* **powerful** is when it is combined with **kernels**. There we projected our data into higher-dimensional space defined by polynomials and Gaussian basis functions, and thereby were able to *fit* for **nonlinear** relationships with a linear classifier.

In SVM models, we can use a version of the same idea. To motivate the need for kernels, let's look at some data that is not linearly separable:

In [11]:
```python
# library to generate circle data
from sklearn.datasets import make_circles

# generate the data
X, y = make_circles(100, factor=.1, noise=.1)

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

# plot the chart
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

# try to make the best-fit line
plot_svc_decision_function(clf, plot_support=False);
```



It is clear that no linear discrimination will **ever** be able to *separate* this data. But we can think about how we might *project* the data into a higher dimension such that a linear separator *would* be sufficient. For example, one simple projection we could use would be to compute a *radial basis function* centered on the middle clump:

In [12]:
```python
# gaussian radial basis function
r = np.exp(-(X ** 2).sum(1))
```

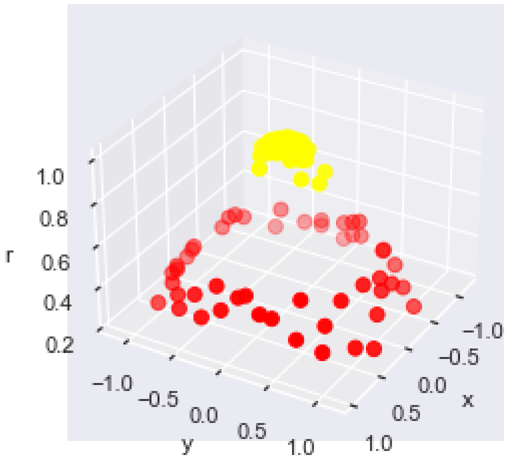
We can visualize this extra data dimension using a **three-dimensional** plot.

If you are running this notebook live, you will be able to use the sliders to rotate the plot:

In [13]:
```
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=[90, 70, 50, 30, 10, -10, -30, -50, -70, -90], azip=(-180, 180),
         X=fixed(X), y=fixed(y));
```

elev 30

azim 30



We can see that with this additional dimension, the data becomes *trivially* linearly separable, by **drawing** a separating plane at, say, *r*=0.7.

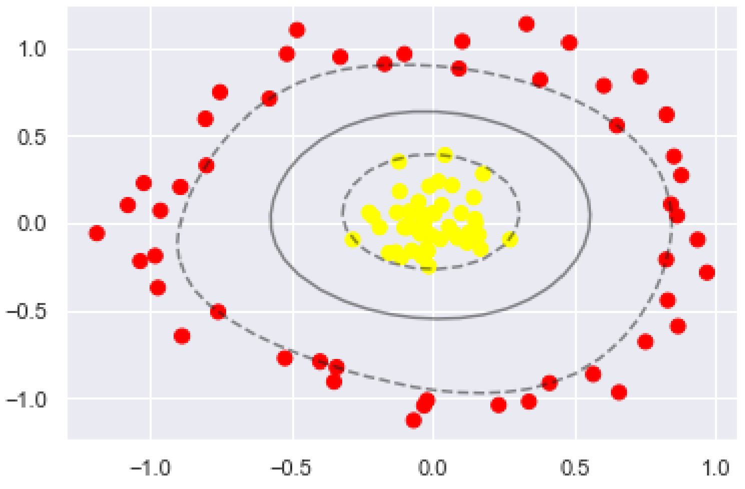
Here we had to choose and carefully tune our projection: if we had not centered our radial basis function in the right location, we would not have seen such clean, linearly separable results. In general, the need to make such a choice is a problem: we would like to somehow automatically find the best basis functions to use.

One strategy to this end is to compute a basis function centered at *every* point in the dataset, and let the SVM algorithm sift through the results. This type of basis function transformation is known as a *kernel transformation*, as it is based on a similarity relationship (or kernel) between each pair of points.

A potential problem with this strategy—projecting N points into N dimensions—is that it might become very computationally intensive as N grows large. However, because of a neat little procedure known as the *kernel trick* (https://en.wikipedia.org/wiki/Kernel_trick), a fit on kernel-transformed data can be done implicitly—that is, without ever building the full N-dimensional representation of the kernel projection! This kernel trick is built into the SVM, and is one of the reasons the method is so powerful.

In `sklearn`, we can apply kernelized SVM simply by changing our linear kernel to an RBF (radial basis function) kernel, using the `kernel` model hyperparameter:

In [14]:
```
# change the kernel from "linear" to "rbf"
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)

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');
```

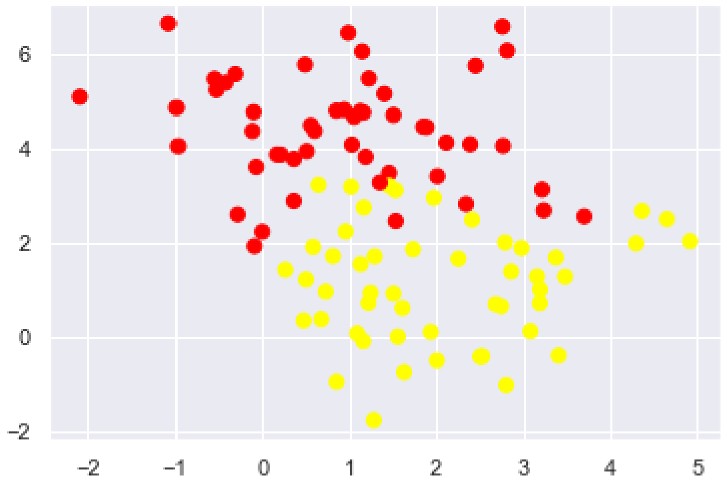


Using this **kernelized** support vector machine, we learn a *suitable* nonlinear decision boundary. This kernel transformation strategy is used often in machine learning to turn fast linear methods into fast nonlinear methods, especially for models in which the *kernel trick* can be used.

Tuning the SVM: Softening Margins

Our discussion thus far has *centered* around very **clean datasets**, in which a perfect decision boundary exists. But what if your data has some amount of overlap? For example, you may have data like this:

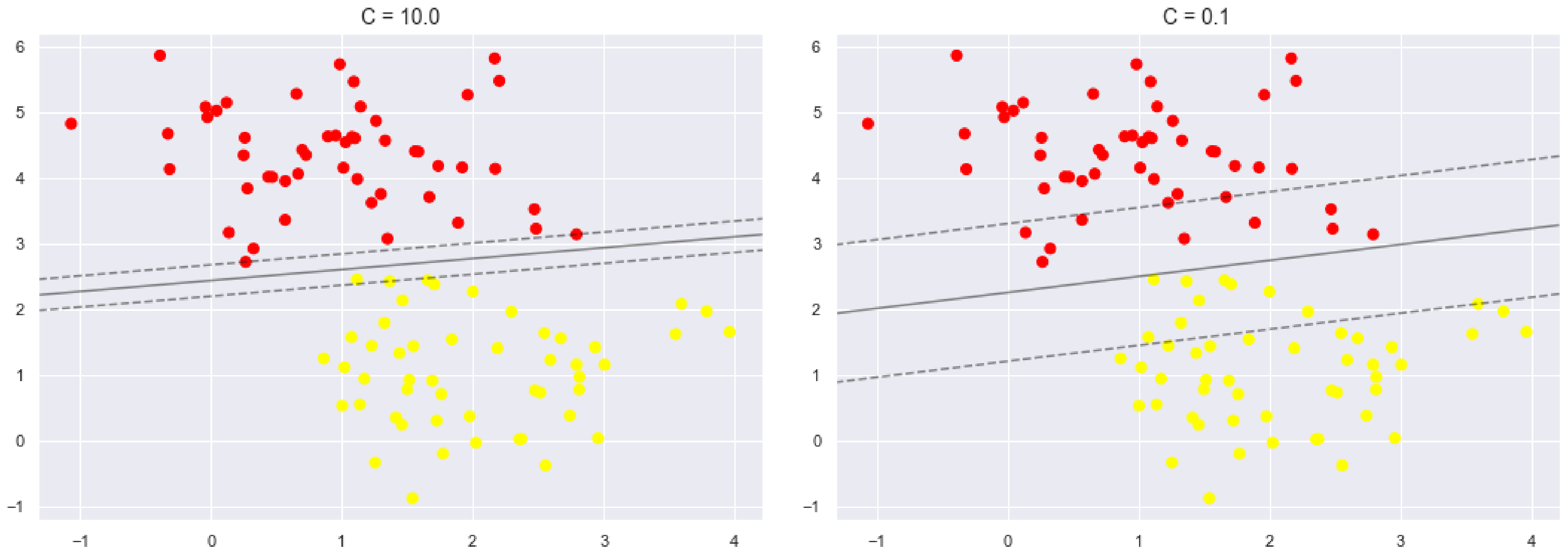
In [15]:
```
X, y = make_blobs(n_samples=100, centers=2,
                  random_state=0, cluster_std=1.2)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



To handle this case, the SVM implementation has a bit of a fudge-factor which "softens" the margin: that is, it allows some of the points to creep into the margin if that allows a better fit. The hardness of the margin is controlled by a tuning parameter, most often known as C. For very large C, the margin is hard, and points cannot lie in it. For smaller C, the margin is softer, and can grow to encompass some points.

The plot shown below gives a visual picture of how a **changing** C parameter affects the *final fit*, via the **softening** of the *margin*:

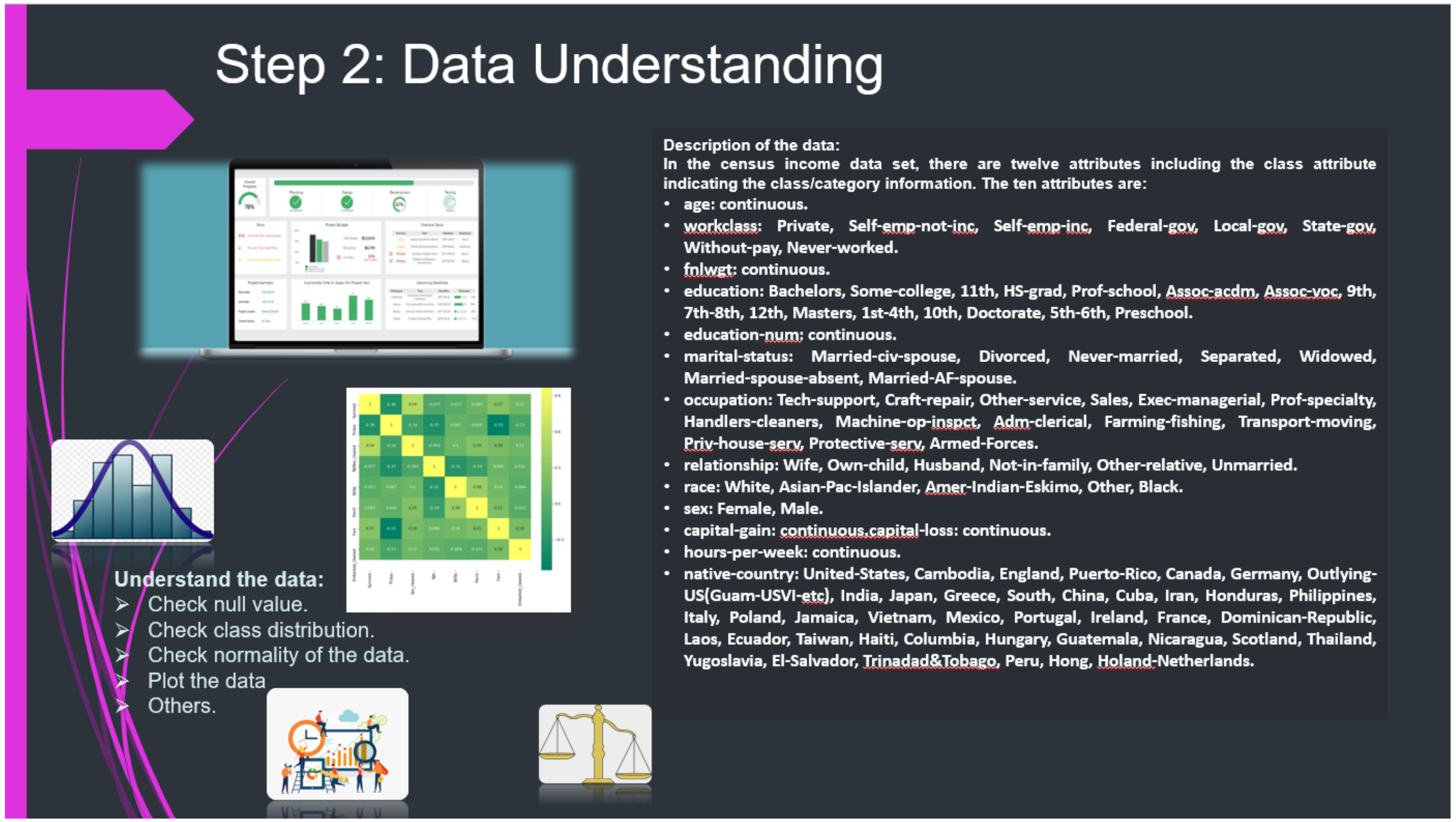
In [16]:
```
X, y = make_blobs(n_samples=100, centers=2, random_state=0, cluster_std=0.8)

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

# test two different 'c' values (10 and 0.1) and plot the results
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)
```



The **optimal** value of the C parameter will depend on your dataset, and should be **tuned** using cross-validation or a similar procedure.

Step1: Business Understanding

This data was extracted from the census bureau database found at: [http://www.census.gov/ftp/pub/DES/www/welcome.html (http://www.census.gov/ftp/pub/DES/www/welcome.html)](http://www.census.gov/ftp/pub/DES/www/welcome.html). It contains the cencus income of the people. They are trying to see the income of more than 50k and less than 50k. | Class label '>50K' : 23.93% / 24.78% (without unknowns) | Class label '<=50K' : 76.07% / 75.22% (without unknowns)



Step 2: Data Understanding

Understand the data:
- ➢ Check null value.
- ➢ Check class distribution.
- ➢ Check normality of the data.
- ➢ Plot the data
- ➢ Others.

Description of the data:
In the census income data set, there are twelve attributes including the class attribute indicating the class/category information. The ten attributes are:
- **age: continuous.**
- **workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.**
- **fnlwgt: continuous.**
- **education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.**
- **education-num: continuous.**
- **marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.**
- **occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.**
- **relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.**
- **race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.**
- **sex: Female, Male.**
- **capital-gain: continuous.capital-loss: continuous.**
- **hours-per-week: continuous.**
- **native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.**

Step 2: Data Understanding

Description of the data:

In the censuc income data set, there are fifteen attributes including the class attribute indicating the class/category information. The 15 attributes are:

- age: continuous.
- workclass:
    - Private
    - Self-emp-not-inc
    - Self-emp-inc
    - Federal-gov
    - Local-gov
    - State-gov
    - Without-pay
    - Never-worked
- fnlwgt: continuous.
- education:
    - Bachelors
    - Some-college
    - 11th
    - HS-grad
    - Prof-school
    - Assoc-acdm
    - Assoc-voc
    - 9th
    - 7th-8th
    - 12th
    - Masters
    - 1st-4th
    - 10th
    - Doctorate
    - 5th-6th
    - Preschool
- education-num: continuous.
- marital-status:
    - Married-civ-spouse

    - Divorced
    - Never-married
    - Separated
    - Widowed
    - Married-spouse-absent
    - Married-AF-spouse
- occupation:
    - Tech-support
    - Craft-repair
    - Other-service
    - Sales
    - Exec-managerial
    - Prof-specialty
    - Handlers-cleaners
    - Machine-op-inspct
    - Adm-clerical
    - Farming-fishing
    - Transport-moving
    - Priv-house-serv
    - Protective-serv
    - Armed-Forces
- relationship:
    - Wife
    - Own-child
    - Husband
    - Not-in-family
    - Other-relative
    - Unmarried
- race:
    - White
    - Asian-Pac-Islander
    - Amer-Indian-Eskimo
    - Other
    - Black
- sex:

    - Female
    - Male
- capital-gain: continuous
- capital-loss: continuous
- hours-per-week: continuous.
- native-country:
    - United-States
    - Cambodia
    - England
    - Puerto-Rico
    - Canada
    - Germany
    - Outlying-US(Guam-USVI-etc)
    - India
    - Japan
    - Greece
    - South
    - China
    - Cuba
    - Iran
    - Honduras
    - Philippines
    - Italy
    - Poland
    - Jamaica
    - Vietnam
    - Mexico
    - Portugal
    - Ireland
    - France
    - Dominican-Republic
    - Laos
    - Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

In [17]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import  accuracy_score, confusion_matrix, precision_recall_fscore_support
import warnings # to hide unnecesary warning
warnings.filterwarnings('ignore')

%matplotlib inline
```

In [18]:
```python
# import library to display multiple outputs
from IPython.display import display

# Importing dataset
train = pd.read_csv("input/adult_train_modified.csv")
test = pd.read_csv("input/adult_test_modified.csv")

# see some of it, their overall statistics and dimensions
display(train.head(5))
display(train.describe())
display(train.shape)
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	Class
0	22	5	2491	9	12	4	0	1	4	1	24	0	39	38	0
1	33	4	2727	9	12	2	3	0	4	1	0	0	12	38	0
2	21	2	13188	11	8	0	5	1	4	1	0	0	39	38	0
3	36	2	14354	1	6	2	5	0	2	1	0	0	39	38	0
4	11	2	18120	9	12	2	9	5	2	0	0	0	39	4	0

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	se
count	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.00000
mean	21.435482	2.199324	9825.221504	10.333764	9.121312	2.580134	5.959850	1.418341	3.678602	0.67568
std	13.125355	0.953925	5671.017927	3.812292	2.549995	1.498016	4.029566	1.601338	0.834709	0.46812
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	11.000000	2.000000	5025.250000	9.000000	8.000000	2.000000	2.000000	0.000000	4.000000	0.00000
50%	20.000000	2.000000	9689.500000	11.000000	9.000000	2.000000	6.000000	1.000000	4.000000	1.00000
75%	30.000000	2.000000	14520.750000	12.000000	12.000000	4.000000	9.000000	3.000000	4.000000	1.00000
max	71.000000	6.000000	20262.000000	15.000000	15.000000	6.000000	13.000000	5.000000	4.000000	1.00000

```
(30162, 15)
```

In [19]:
```
#describe the data
train.describe()
```

Out[19]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	se
count	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.000000	30162.00000
mean	21.435482	2.199324	9825.221504	10.333764	9.121312	2.580134	5.959850	1.418341	3.678602	0.67568
std	13.125355	0.953925	5671.017927	3.812292	2.549995	1.498016	4.029566	1.601338	0.834709	0.46812
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	11.000000	2.000000	5025.250000	9.000000	8.000000	2.000000	2.000000	0.000000	4.000000	0.00000
50%	20.000000	2.000000	9689.500000	11.000000	9.000000	2.000000	6.000000	1.000000	4.000000	1.00000
75%	30.000000	2.000000	14520.750000	12.000000	12.000000	4.000000	9.000000	3.000000	4.000000	1.00000
max	71.000000	6.000000	20262.000000	15.000000	15.000000	6.000000	13.000000	5.000000	4.000000	1.00000

In [20]:
```
#Get the shape/dimension of data
train.shape
```

Out[20]:
```
(30162, 15)
```

In [21]:
```
# Count observations based on attribute
train['Class'].value_counts()
```

Out[21]:
```
0    22654
1     7508
Name: Class, dtype: int64
```

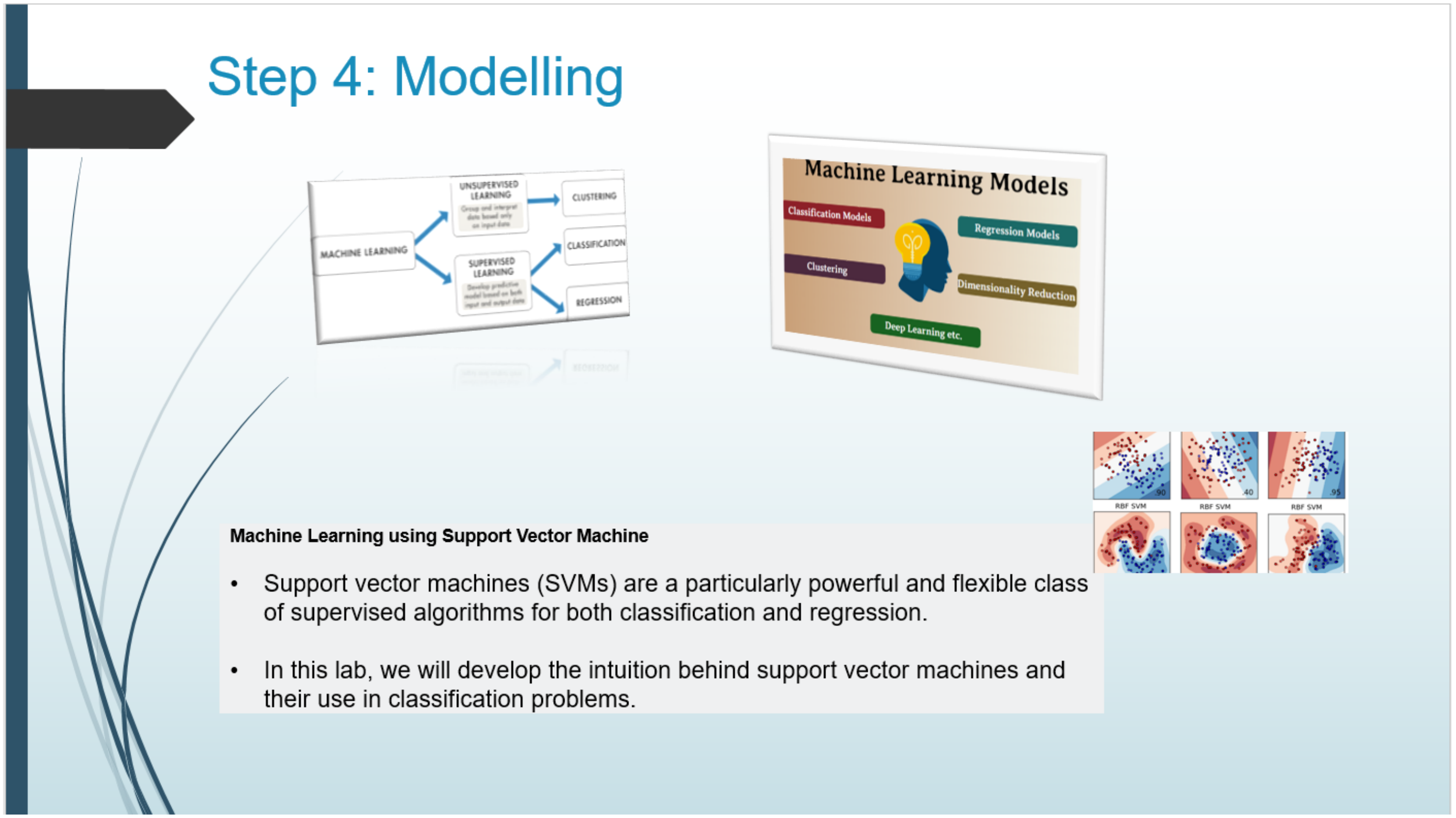
In [22]:
```
# select rows from dataframe
x=train.iloc[:,:-1]

# sum of null data based on attributes
x.isnull().sum()
```

Out[22]:
```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
dtype: int64
```

Step 4: Modelling

Classify using Support Vector Machine

Here, the next **question** directly arises:

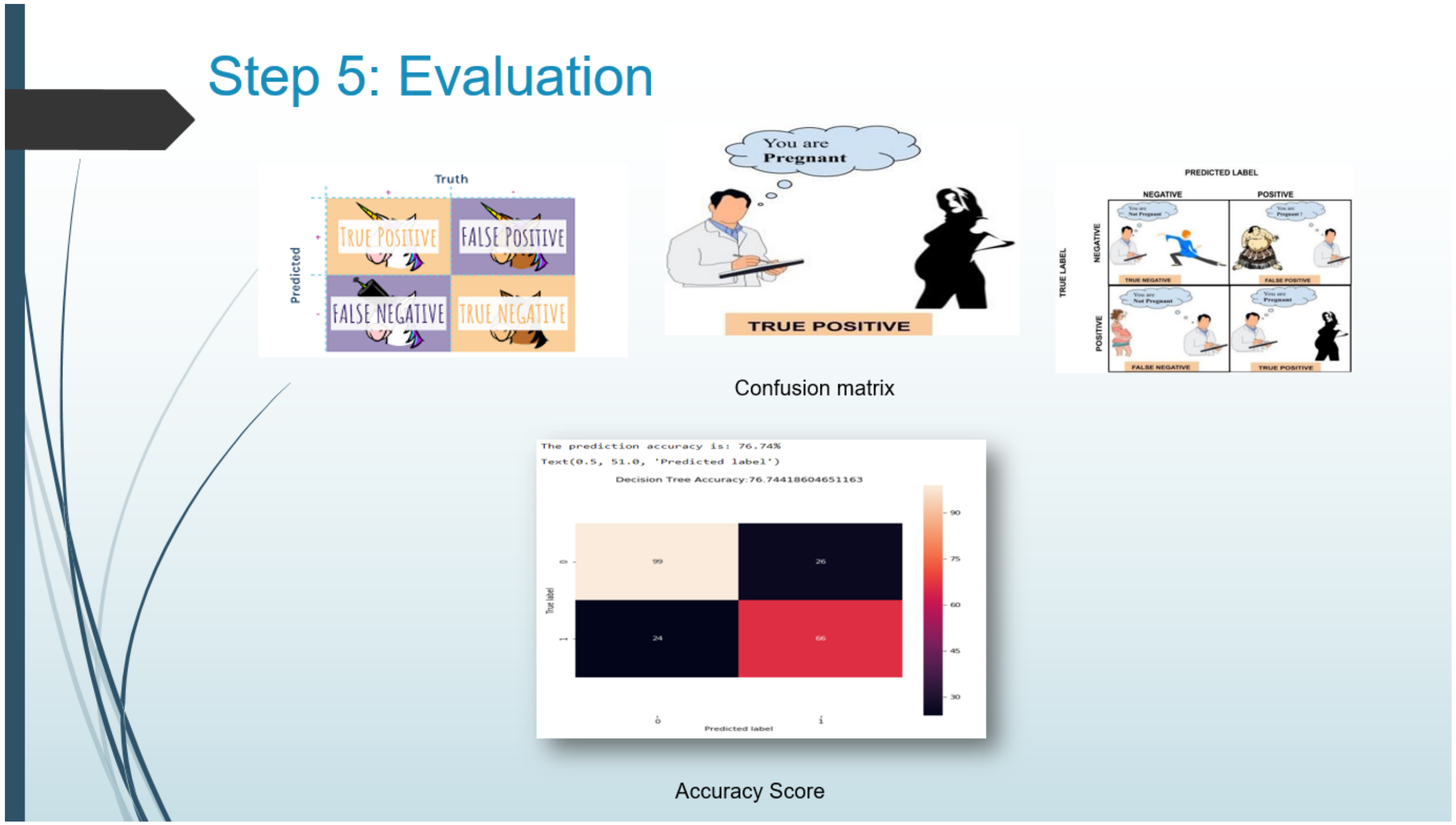

MACHINE LEARNING
UNSUPERVISED LEARNING
SUPERVISED LEARNING
CLUSTERING
CLASSIFICATION
REGRESSION
Machine Learning Models
Classification Models
Regression Models
Clustering
Dimensionality Reduction
Deep Learning etc.
RBF SVM

Step 4: Modelling

Machine Learning using Support Vector Machine

- Support vector machines (SVMs) are a particularly powerful and flexible class of supervised algorithms for both classification and regression.
- In this lab, we will develop the intuition behind support vector machines and their use in classification problems.

Step 5: Evaluation



Step 5: Evaluation

```
The prediction accuracy is: 76.74%
Text(0.5, 51.0, 'Predicted label')
```

Confusion matrix

Accuracy Score

In [23]:
```
# select all columns except the last one (the target label)
x_train=train.iloc[:,:-1]
# set target categorical data label (sixth attribute)
y_train=train.iloc[:,14]

# select all columns except the last one (the target label)
x_test=test.iloc[:,:-1]
# set target categorical data label (sixth attribute)
y_test=test.iloc[:,14]

#Use line below if want to split data into training and testing
#x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4,random_state=0)
```

In [24]:
```
print('-------- x axis test ----------')
print(x_test)
print('-------- x axis train ---------')
print(x_train)
print('-------- y axis test ----------')
print(y_test)
print('-------- y axis train ---------')
print(y_train)
print('******************************')
```

```
-------- x axis test ----------
       age  workclass  fnlwgt  education  education-num  marital-status  \
0        8          2    8315          1              6               4   
1       21          2    1754         11              8               2   
2       11          1   10750          7             11               2   
3       27          2    4780         15              9               2   
4       17          2    7091          0              5               4   
...    ...        ...     ...        ...            ...             ...   
15055   16          2    8927          9             12               4   
15056   22          2    7893          9             12               0   
15057   21          2   11193          9             12               2   
15058   27          2    1593          9             12               0   
15059   18          3    6062          9             12               2   

       occupation  relationship  race  sex  capital-gain  capital-loss  \
0               6             3     2    1             0             0   
1               4             0     4    1             0             0   
2              10             0     4    1             0             0   
3               6             0     2    1            87             0   
```

Min-Max scaler

Transform features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g., between zero and one. This Scaler shrinks the data within the range of -1 to 1 if there are negative values. We can set the range like [0,1] or [0,5] or [-1,1].

In [25]:
```
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(-1,1)).fit(x_train)
x_train = scaling.transform(x_train)
x_test = scaling.transform(x_test)
```

In [26]:
```
print(x_train)
```

```
[[-0.38028169  0.66666667 -0.75412101 ... -1.         -0.16129032
   0.9       ]
 [-0.07042254  0.33333333 -0.73082618 ... -1.         -0.74193548
   0.9       ]
 [-0.4084507  -0.33333333  0.30174711 ... -1.         -0.16129032
   0.9       ]
 ...
 [ 0.15492958 -0.33333333 -0.27183891 ... -1.         -0.16129032
   0.9       ]
 [-0.85915493 -0.33333333  0.19040569 ... -1.         -0.59139785
   0.9       ]
 [-0.01408451  0.          0.64732011 ... -1.         -0.16129032
   0.9       ]]
```

Linear Kernel

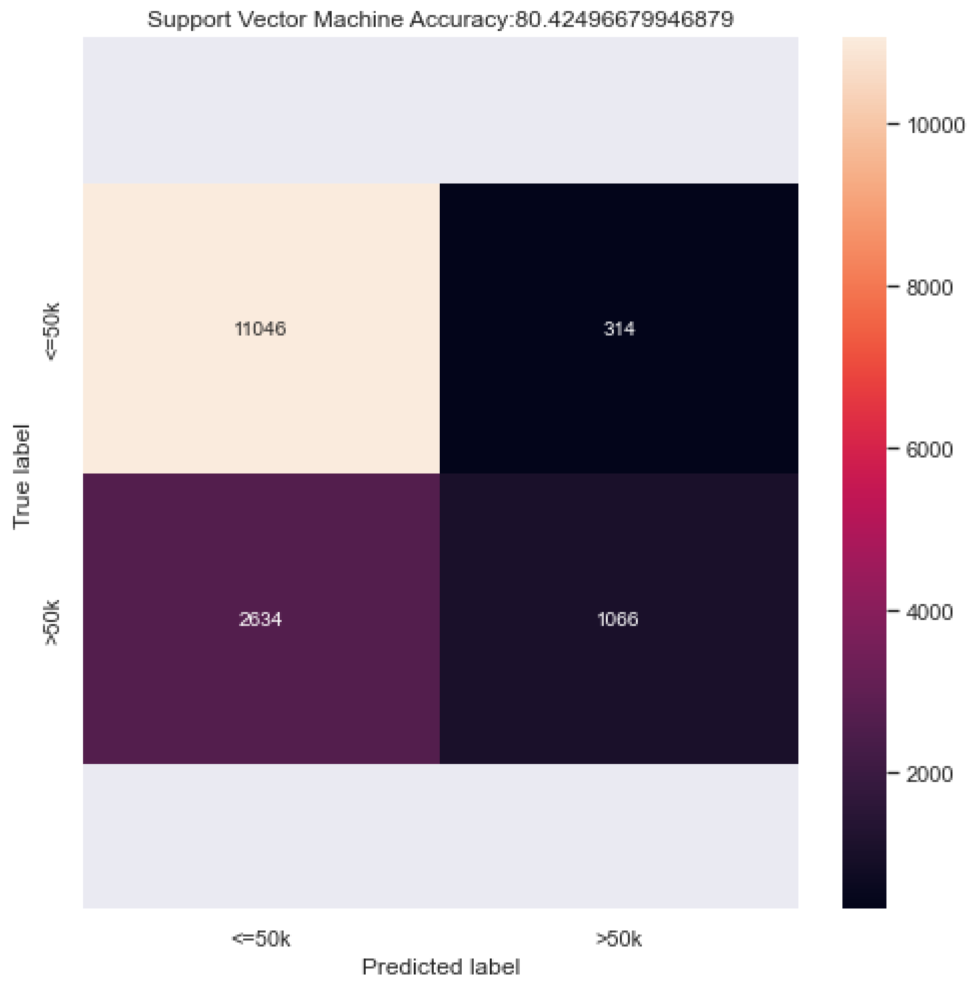
In [27]:
```
from sklearn.svm import SVC
svc_linear= SVC(C=1, kernel='linear', gamma = 0.001)
svc_linear.fit(x_train, y_train)
yfit = svc_linear.predict(x_test)
```

In [28]:
```
# import classification report metrices
from sklearn.metrics import classification_report

print(classification_report(y_test, yfit))
```

```
              precision    recall  f1-score   support

           0       0.81      0.97      0.88     11360
           1       0.77      0.29      0.42      3700

    accuracy                           0.80     15060
   macro avg       0.79      0.63      0.65     15060
weighted avg       0.80      0.80      0.77     15060
```

In [29]:
```
# import the confusion matrix
from sklearn.metrics import confusion_matrix

# compute the confusion matrix
cm = confusion_matrix(y_test, yfit)
# Transform to dataframe for easier plotting
cm_df = pd.DataFrame(cm, index = ['<=50k','>50k'],
                     columns = ['<=50k','>50k'])

# plot the confusion matrix
plt.figure(figsize=(8,8))
ax= sns.heatmap(cm_df, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.title("Support Vector Machine Accuracy:" + str(svc_linear.score(x_test,y_test)*100))
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[29]:
```
Text(0.5, 48.5, 'Predicted label')
```



Next, we can use a `grid search` cross-validation to explore **combinations** of parameters. Here we will adjust `C` (which controls the margin hardness) and `gamma` , and determine the best model:

Other parameters in SVM: class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)

In [30]:
```
from sklearn.model_selection import GridSearchCV
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
simplefilter(action='ignore', category=DeprecationWarning)

param_grid= {'C': [1, 5, 10, 50],
              'gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(svc_linear, param_grid, verbose=1, n_jobs=-1)
grid.fit(x_train, y_train)

gridSVM=grid.best_params_
print(gridSVM)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
{'C': 5, 'gamma': 0.0001}
```

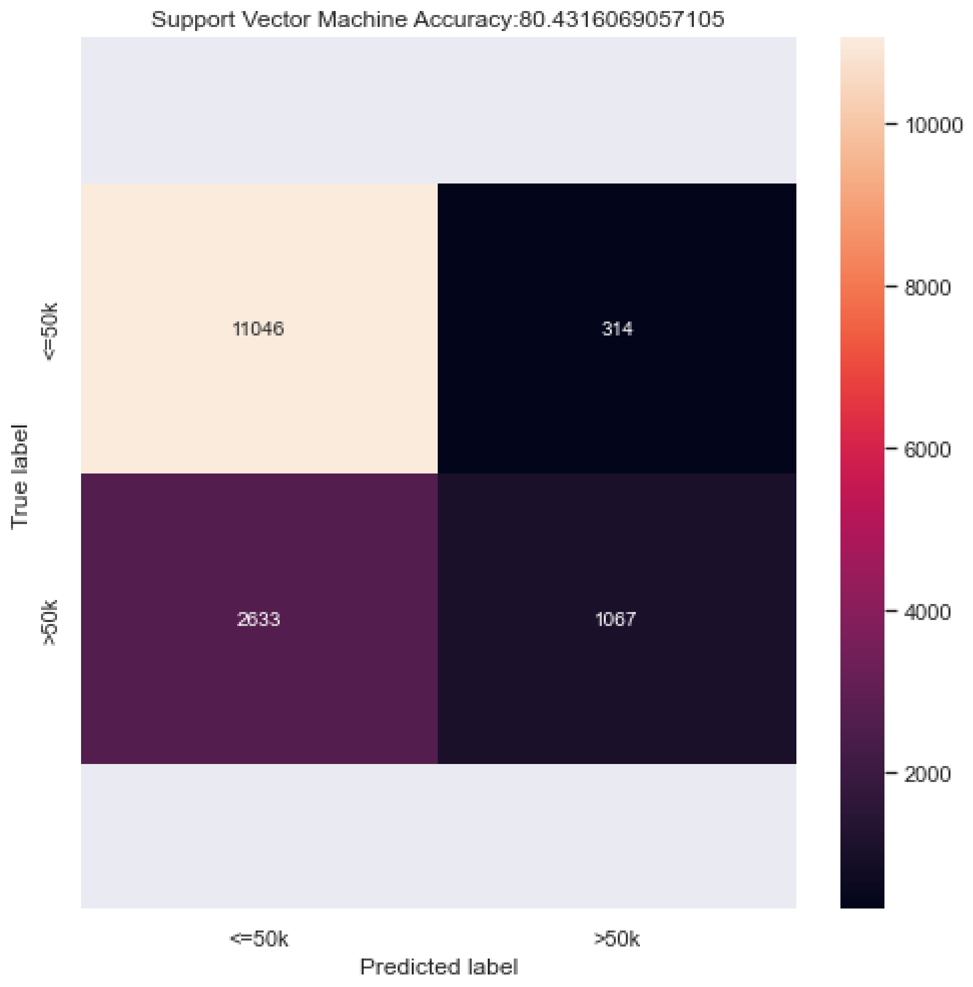
In [31]:
```
#model = grid.best_estimator_
svcGrid= SVC(C=5, kernel='linear', gamma = 0.0001)
svcGrid.fit(x_train, y_train)
yfitGrid = svcGrid.predict(x_test)
```

In [32]:
```
# import classification report metrices
from sklearn.metrics import classification_report

print(classification_report(y_test, yfitGrid))
```

```
              precision    recall  f1-score   support

           0       0.81      0.97      0.88     11360
           1       0.77      0.29      0.42      3700

    accuracy                           0.80     15060
   macro avg       0.79      0.63      0.65     15060
weighted avg       0.80      0.80      0.77     15060
```

In [33]:
```
# import the confusion matrix
from sklearn.metrics import confusion_matrix

# compute the confusion matrix
cm = confusion_matrix(y_test, yfitGrid)
# Transform to dataframe for easier plotting
cm_df = pd.DataFrame(cm, index = ['<=50k','>50k'],
                     columns = ['<=50k','>50k'])

# plot the confusion matrix
plt.figure(figsize=(8,8))
ax= sns.heatmap(cm_df, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.title("Support Vector Machine Accuracy:" + str(svcGrid.score(x_test,y_test)*100))
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[33]:
```
Text(0.5, 48.5, 'Predicted label')
```



RBF Kernel

Let's change our kernel to rbf now and observed the results.

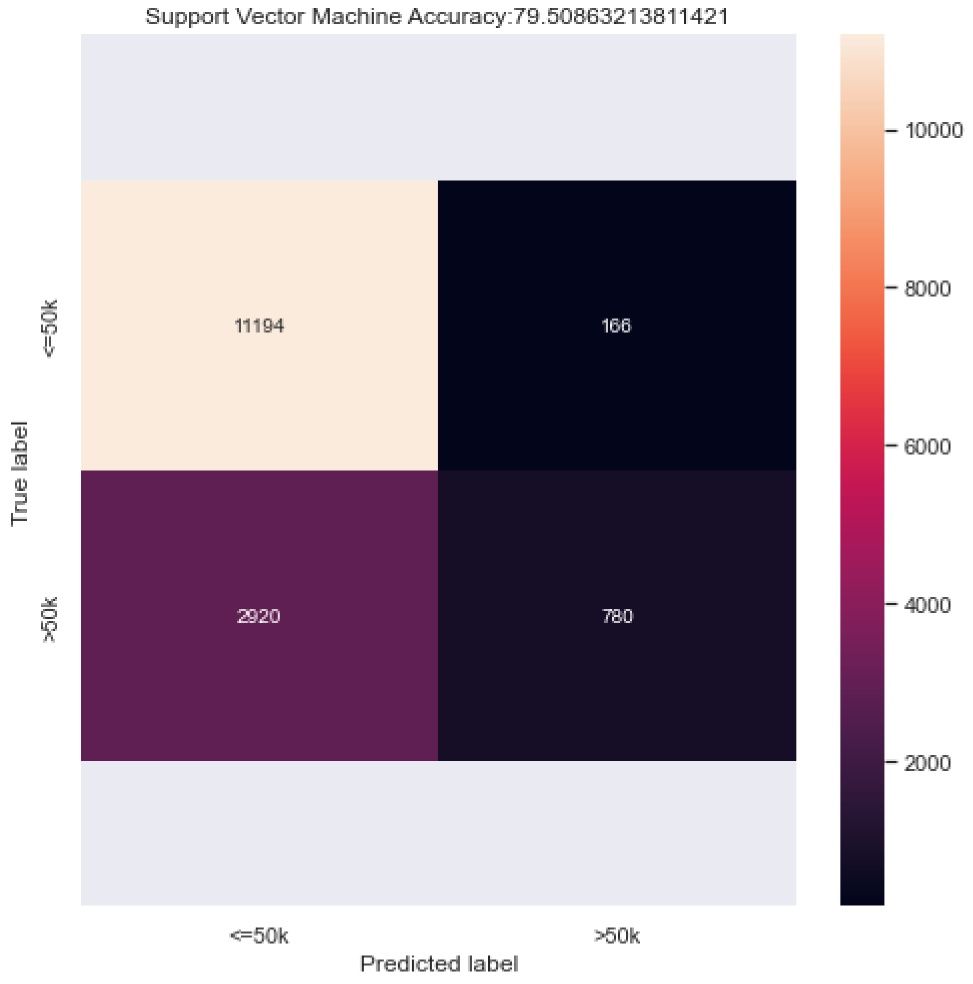
In [34]:
```
from sklearn.svm import SVC
svc_rbf= SVC(C=1, kernel='rbf', gamma = 0.001)
svc_rbf.fit(x_train, y_train)
yfit = svc_rbf.predict(x_test)
```

In [35]:
```
# import classification report metrices
from sklearn.metrics import classification_report

print(classification_report(y_test, yfit))
```

```
              precision    recall  f1-score   support

           0       0.79      0.99      0.88     11360
           1       0.82      0.21      0.34      3700

    accuracy                           0.80     15060
   macro avg       0.81      0.60      0.61     15060
weighted avg       0.80      0.80      0.75     15060
```

In [36]:
```
# import the confusion matrix
from sklearn.metrics import confusion_matrix

# compute the confusion matrix
cm = confusion_matrix(y_test, yfit)
# Transform to dataframe for easier plotting
cm_df = pd.DataFrame(cm, index = ['<=50k','>50k'],
                     columns = ['<=50k','>50k'])

# plot the confusion matrix
plt.figure(figsize=(8,8))
ax= sns.heatmap(cm_df, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.title("Support Vector Machine Accuracy:" + str(svc_rbf.score(x_test,y_test)*100))
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[36]:
```
Text(0.5, 48.5, 'Predicted label')
```


Support Vector Machine Accuracy:79.50863213811421
True label
<=50k
>50k
11194
166
2920
780
Predicted label
<=50k
>50k
10000
8000
6000
4000
2000

In [44]:
```
from sklearn.model_selection import GridSearchCV
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
simplefilter(action='ignore', category=DeprecationWarning)

param_grid= {'C': [1, 5, 10, 50],
              'gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(svc_rbf, param_grid, verbose=1, n_jobs=-1)
grid.fit(x_train, y_train)

gridSVM=grid.best_params_
print(gridSVM)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
{'C': 50, 'gamma': 0.005}
```

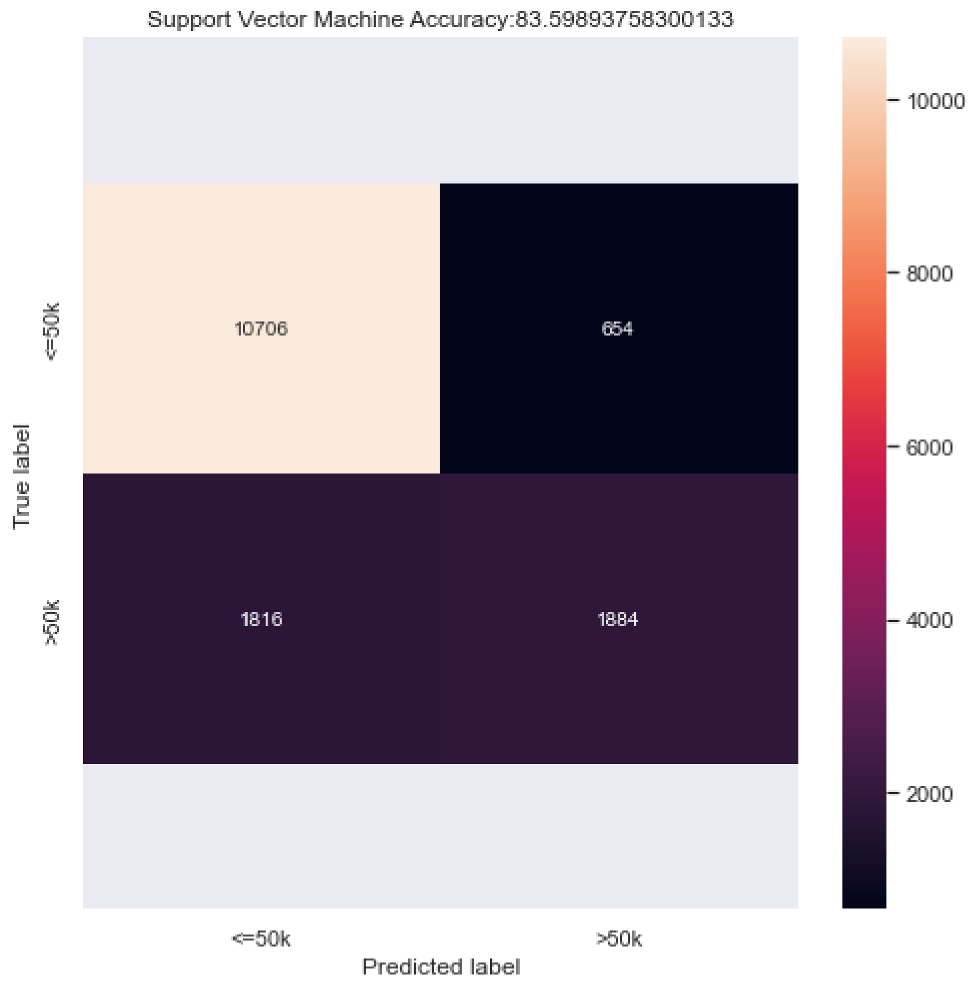
In [41]:
```
#model = grid.best_estimator_
svcGridRBF= SVC(C=50, kernel='rbf', gamma = 0.005)
svcGridRBF.fit(x_train, y_train)
yfitGridrbf = svcGridRBF.predict(x_test)
```

In [42]:
```
# import classification report metrices
from sklearn.metrics import classification_report

print(classification_report(y_test, yfitGridrbf))
```

```
              precision    recall  f1-score   support

           0       0.85      0.94      0.90     11360
           1       0.74      0.51      0.60      3700

    accuracy                           0.84     15060
   macro avg       0.80      0.73      0.75     15060
weighted avg       0.83      0.84      0.82     15060
```

In [43]:
```
# import the confusion matrix
from sklearn.metrics import confusion_matrix

# compute the confusion matrix
cm = confusion_matrix(y_test, yfitGridrbf)
# Transform to dataframe for easier plotting
cm_df = pd.DataFrame(cm, index = ['<=50k','>50k'],
                     columns = ['<=50k','>50k'])

# plot the confusion matrix
plt.figure(figsize=(8,8))
ax= sns.heatmap(cm_df, annot=True, fmt='g')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.title("Support Vector Machine Accuracy:" + str(svcGridRBF.score(x_test,y_test)*100))
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[43]:
```
Text(0.5, 48.5, 'Predicted label')
```



Support Vector Machine Summary

We have seen here a brief intuitive introduction to the principals behind support vector machines. These methods are a powerful classification method for a number of reasons:

- Their dependence on **relatively few support vectors** means that they are **very compact** models, and take up very little memory.
- Once the model is **trained**, the prediction phase is *very fast*.
- Because they are affected only by points near the margin, they **work well with high-dimensional** data; even data with more dimensions than samples, which is a challenging regime for other algorithms.

- Their *integration* with **kernel methods** makes them very *versatile*, able to **adapt** to many types of data.

However, SVMs have several disadvantages as well:

- The scaling with the number of samples N is $\mathcal{O}[N^3]$ at worst, or $\mathcal{O}[N^2]$ for efficient implementations. This mean that for **large numbers** of training samples, the computational cost can be prohibitive (very long, even can took days or weeks!).
- The results are **strongly dependent** on a *suitable choice* for the softening parameter C. This must be **carefully chosen** via cross-validation, which can be expensive as datasets grow in size.

With those traits in mind, SVMs generally used when other simpler, faster, and less tuning-intensive methods have been shown to be **insufficient** for your needs. Nevertheless, if you have the CPU cycles to **commit** to *training* and *cross-validating* an SVM on your data, the method can lead to **excellent** results.

Additional link for SVM parameter exploration: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)

In [ ]: