7) Aim: Build Support Vector Machine (SVM) model

In [43]:

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.metrics import accuracy_score , confusion_matrix , roc_auc_score ,
```

In [2]:

```
%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()
```

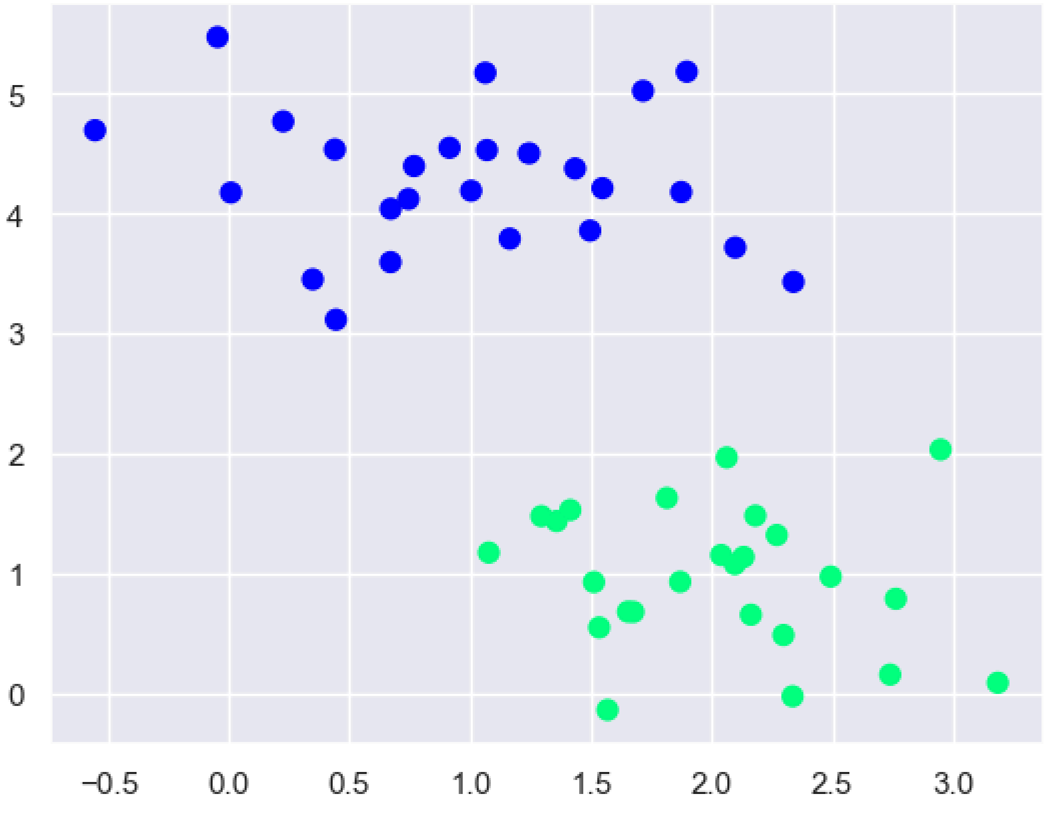
Working with Perfectly Linear Dataset

In [3]:

```
from sklearn.datasets import make_blobs
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='winter')
```

Out[3]:

```
<matplotlib.collections.PathCollection at 0x1d333e5b290>
```



In [4]:

```
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear', C=1)
```

```
model.fit(X, y)
```

Out[4]:

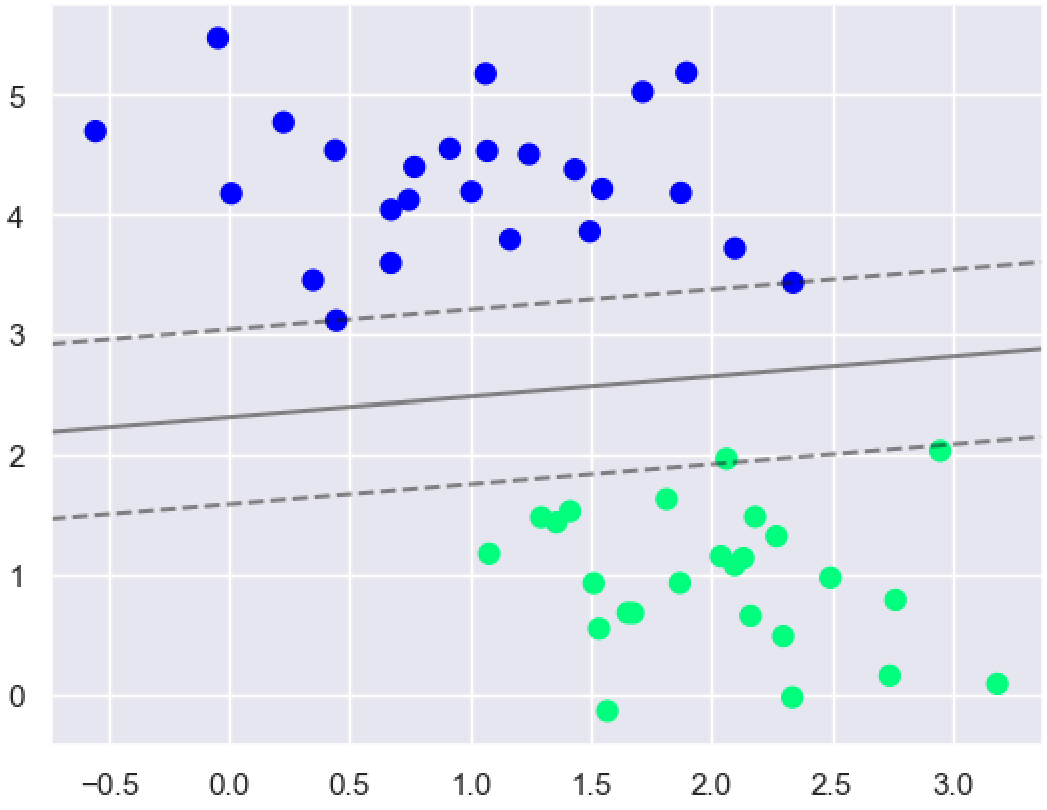
```
▾ SVC
SVC(C=1, kernel='linear')
```

In [5]:

```
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)
```

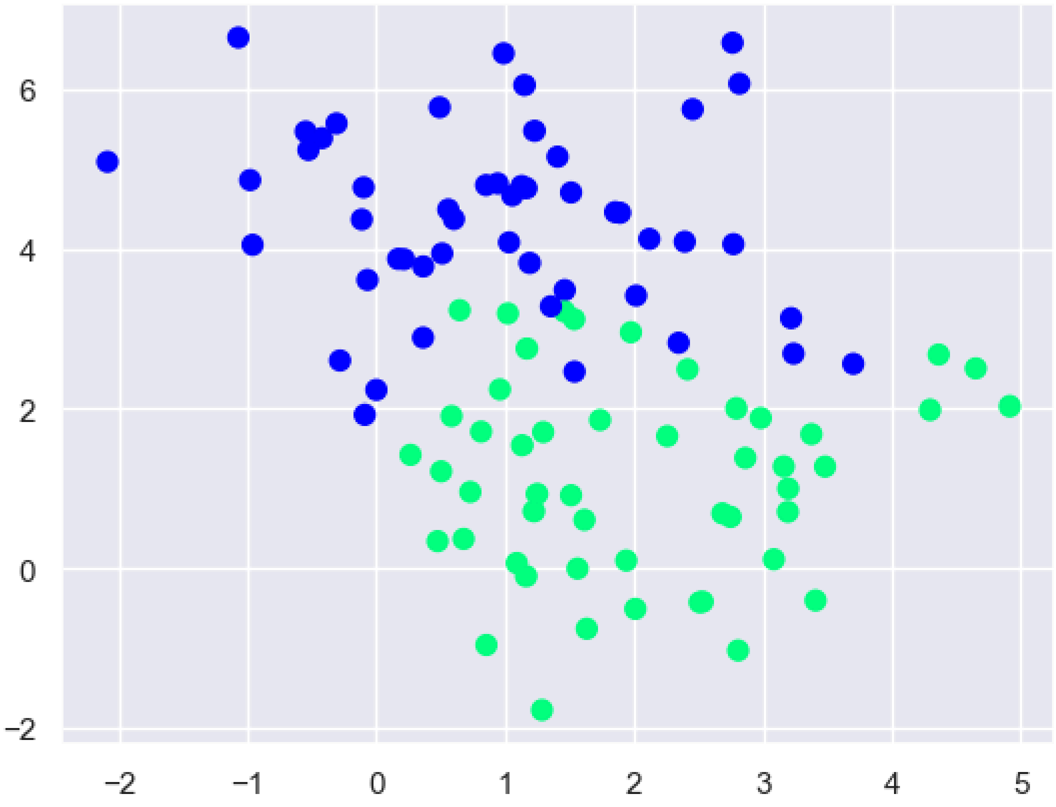
In [6]:

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



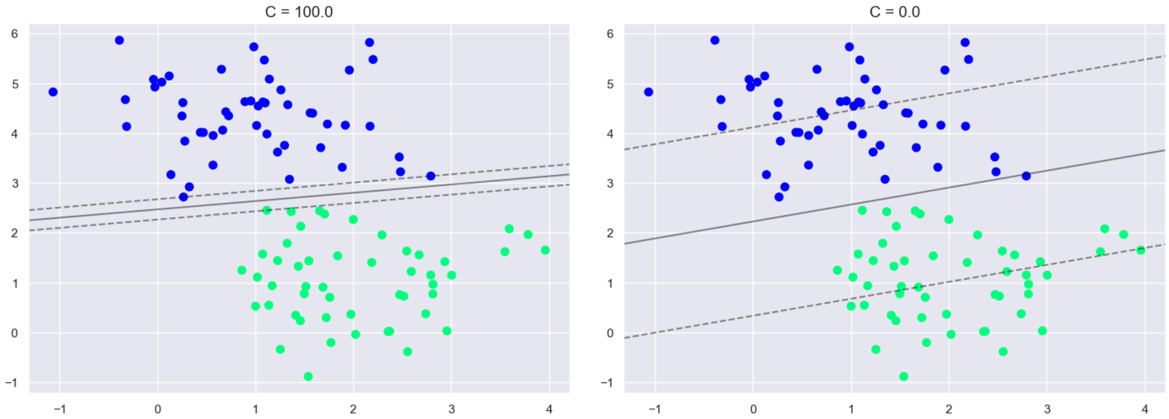
Almost Linearly Separable Dataset

In [7]:
```
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='winter');
```



In [8]:
```
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)

for axi, C in zip(ax, [100.0, 0.01]):
    model = SVC(kernel='linear', C=C).fit(X, y)
    axi.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='winter')
    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)
```



Working with Perfectly Linear Dataset

In [9]:
```
data = pd.read_csv('mushrooms.csv')
```

In [10]:
```
data.shape
```

Out[10]:
```
(8124, 23)
```

In [11]:
```
X = data.drop(columns='class')
```

In [12]:
```
y = data['class']
```

In [13]:
```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
```

In [14]:
```
df2 = pd.DataFrame()
df2['cap-shape'] = encoder.fit_transform(data['cap-shape'])
df2['cap-surface'] = encoder.fit_transform(data['cap-surface'])
df2['cap-color'] = encoder.fit_transform(data['cap-color'])
df2['bruises'] = encoder.fit_transform(data['bruises'])
df2['odor'] = encoder.fit_transform(data['odor'])
df2['gill-attachment'] = encoder.fit_transform(data['gill-attachment'])
df2['gill-spacing'] = encoder.fit_transform(data['gill-spacing'])
df2['gill-size'] = encoder.fit_transform(data['gill-size'])
df2['gill-color'] = encoder.fit_transform(data['gill-color'])
df2['stalk-shape'] = encoder.fit_transform(data['stalk-shape'])
df2['stalk-root'] = encoder.fit_transform(data['stalk-root'])
df2['stalk-surface-above-ring'] = encoder.fit_transform(data['stalk-surface-abov
df2['stalk-surface-below-ring'] = encoder.fit_transform(data['stalk-surface-belo
df2['stalk-color-above-ring'] = encoder.fit_transform(data['stalk-color-above-ri
df2['stalk-color-below-ring'] = encoder.fit_transform(data['stalk-color-below-ri
df2['veil-type'] = encoder.fit_transform(data['veil-type'])
df2['veil-color'] = encoder.fit_transform(data['veil-color'])
df2['ring-number'] = encoder.fit_transform(data['ring-number'])
df2['ring-type'] = encoder.fit_transform(data['ring-type'])
df2['spore-print-color'] = encoder.fit_transform(data['spore-print-color'])
df2['population'] = encoder.fit_transform(data['population'])
df2['habitat'] = encoder.fit_transform(data['habitat'])
df2['class'] = data['class']
```

In [15]:

```
X = df2.drop(columns='class')
y = df2['class']
```

In [16]:

```
x_train , x_test, y_train,y_test = train_test_split(X,y,random_state = 20)
```

In [17]:

```
from sklearn.svm import SVC
```

In [18]:

```
svc= SVC()
svc.fit(x_train,y_train)
```

Out[18]:

```
▾ SVC
SVC()
```

In [19]:

```
svc.score(x_test,y_test)
```

Out[19]:

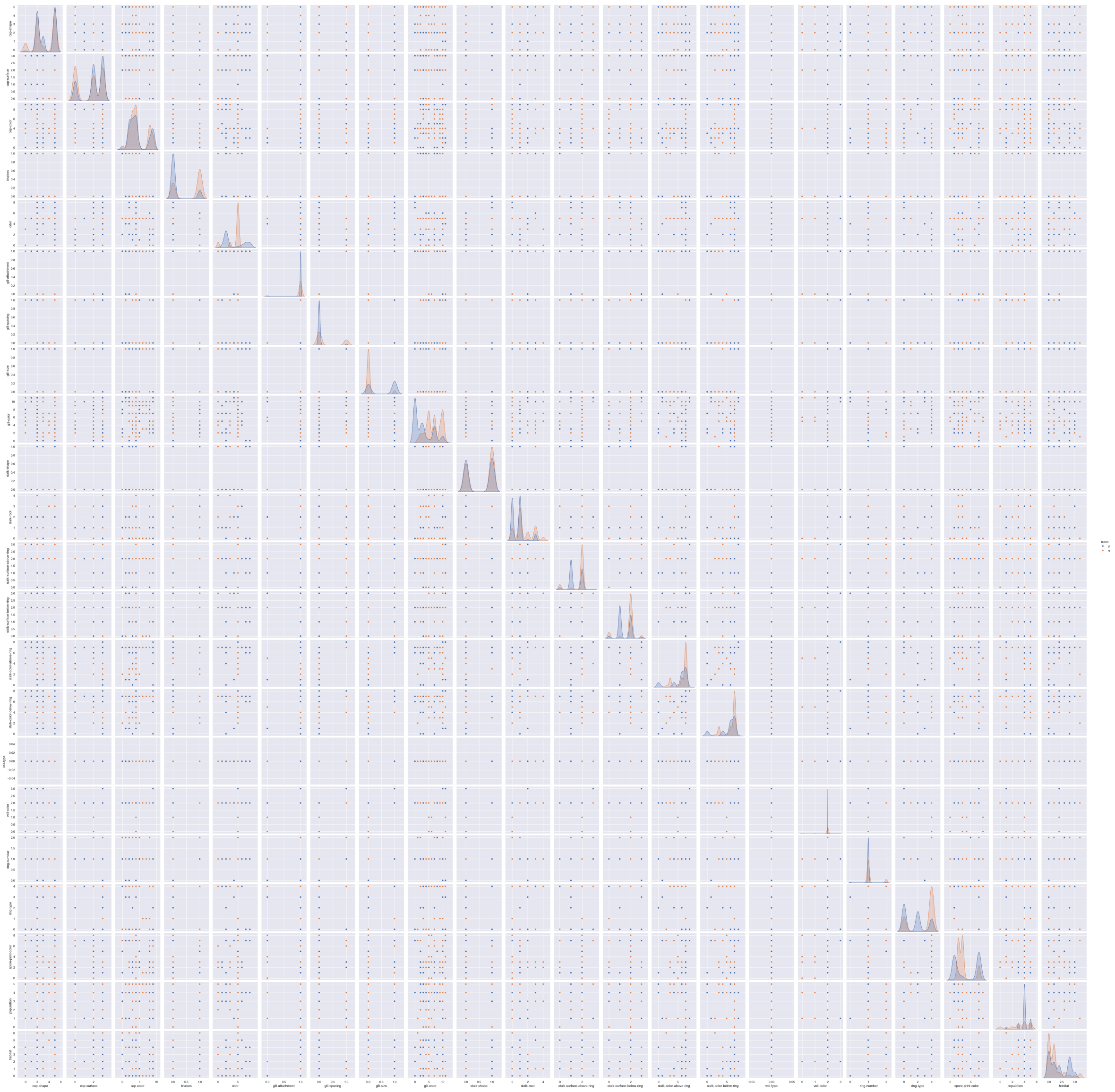
```
0.9911373707533235
```

In [44]:

```
sns.pairplot(df2, hue='class')
```

Out[44]:

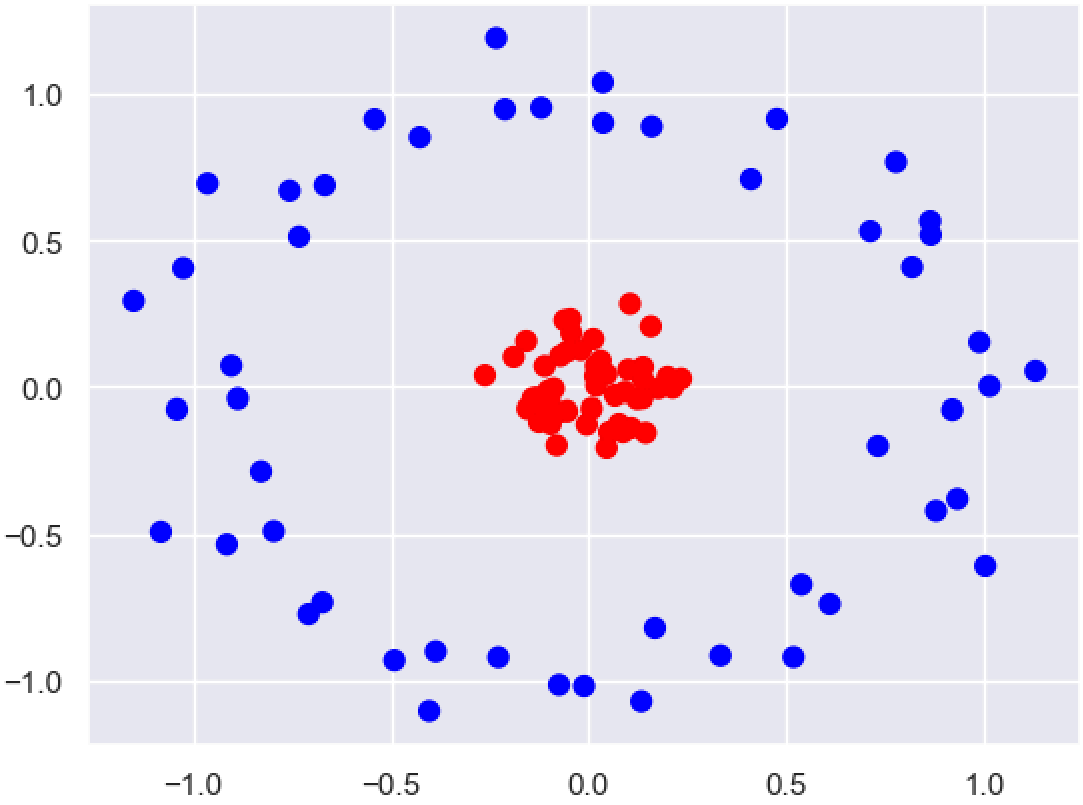
```
<seaborn.axisgrid.PairGrid at 0x1d33a2818d0>
```



```
In [21]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         from matplotlib.axes._axes import _log as matplotlib_axes_logger
         from mpl_toolkits import mplot3d
         from sklearn.model_selection import train_test_split
         from sklearn.svm import SVC
         from matplotlib.colors import ListedColormap
```

```
In [22]: from sklearn.datasets import make_circles
         X, y = make_circles(100, factor=.1, noise=.1)

         plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='bwr')
```

```
Out[22]: <matplotlib.collections.PathCollection at 0x1d3364ed210>
```



In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

In [24]:
```
classifier = SVC(kernel="linear")
classifier.fit(X_train, y_train.ravel())
y_pred = classifier.predict(X_test)
```

In [25]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

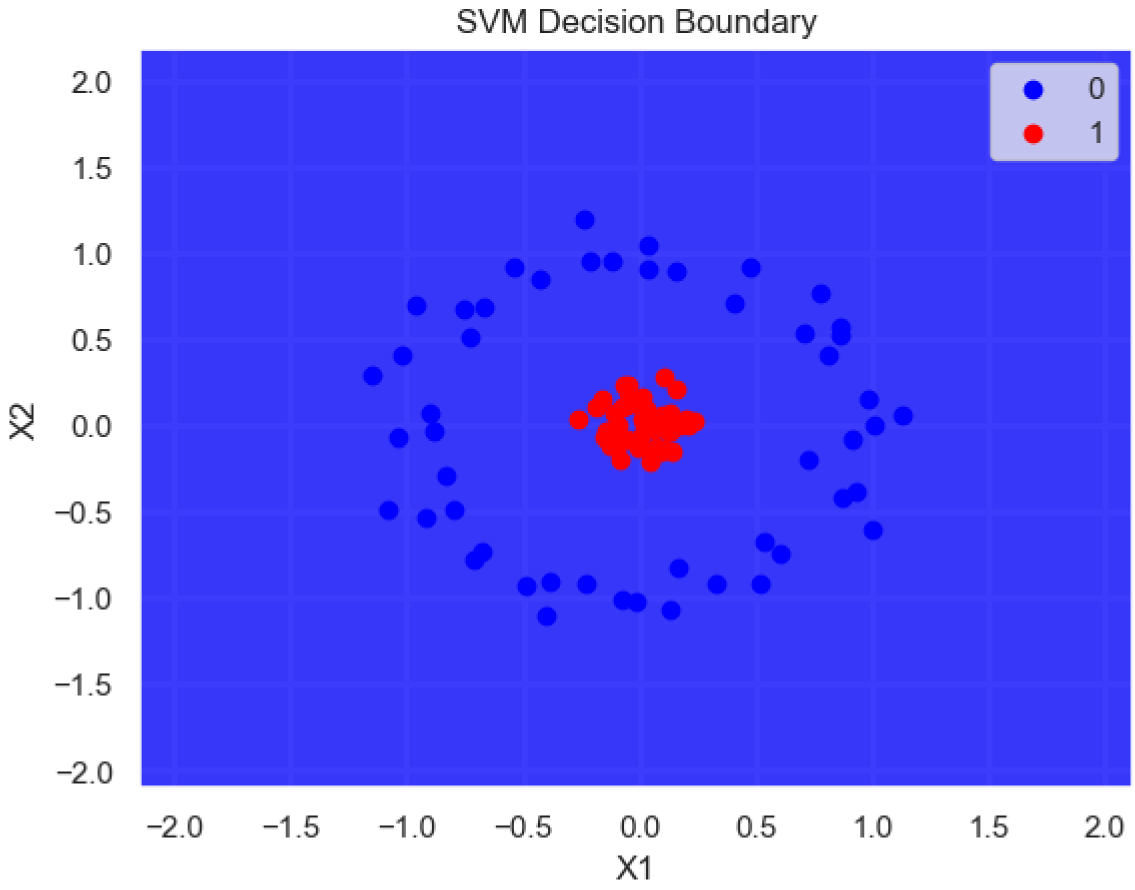
Out[25]:
```
0.4
```

In [26]:
```
zero_one_colourmap = ListedColormap(('blue', 'red'))
def plot_decision_boundary(X, y, clf):
    X_set, y_set = X, y
    X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                                   stop = X_set[:, 0].max() + 1,
                                   step = 0.01),
                         np.arange(start = X_set[:, 1].min() - 1,
                                   stop = X_set[:, 1].max() + 1,
                                   step = 0.01))

    plt.contourf(X1, X2, clf.predict(np.array([X1.ravel(),
                                               X2.ravel()]).T).reshape(X1.shape),
                 alpha = 0.75,
                 cmap = zero_one_colourmap)
    plt.xlim(X1.min(), X1.max())
    plt.ylim(X2.min(), X2.max())
    for i, j in enumerate(np.unique(y_set)):
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                    c = (zero_one_colourmap)(i), label = j)
    plt.title('SVM Decision Boundary')
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.legend()
    return plt.show()
```

In [27]:

```
plot_decision_boundary(X, y, classifier)
```

```
C:\Users\nayan\AppData\Local\Temp\ipykernel_19720\3603277588.py:18: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be av
oided as value-mapping will have precedence in case its length matches with *x* &
*y*.  Please use the *color* keyword-argument or provide a 2D array with a single
row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



In [31]:

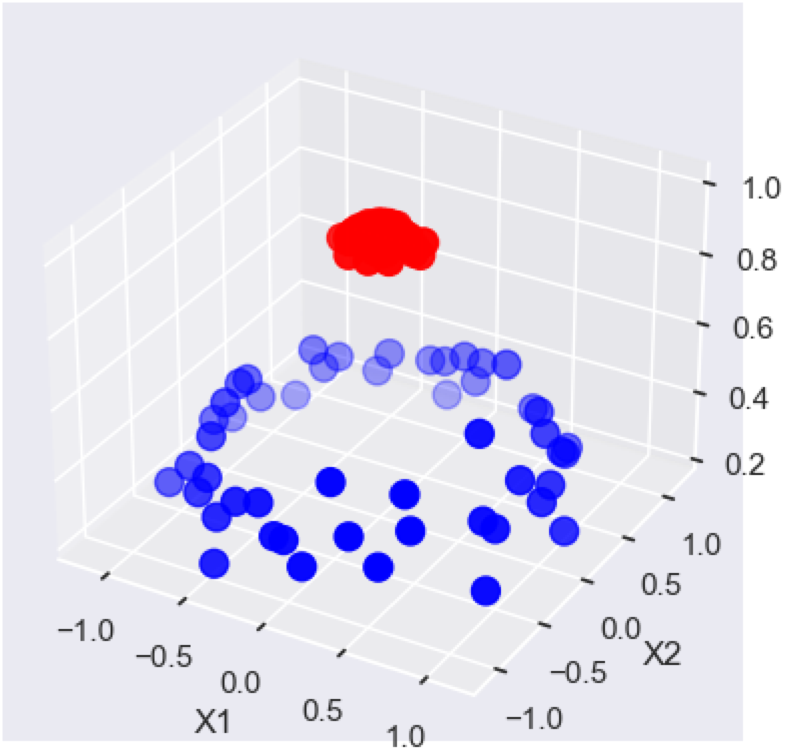
```
def plot_3d_plot(X, y):
    r = np.exp(-(X ** 2).sum(1))
    ax = plt.subplot(projection='3d')
    ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=100, cmap='bwr')
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('y')
    rbf_classifier = SVC(kernel="rbf")
    rbf_classifier.fit(X_train, y_train)
    y_pred = rbf_classifier.predict(X_test)
    return ax, rbf_classifier
```

In [32]:

```
plot_3d_plot(X,y)
```

Out[32]:

```
(<Axes3D: xlabel='X1', ylabel='X2', zlabel='y'>, SVC())
```



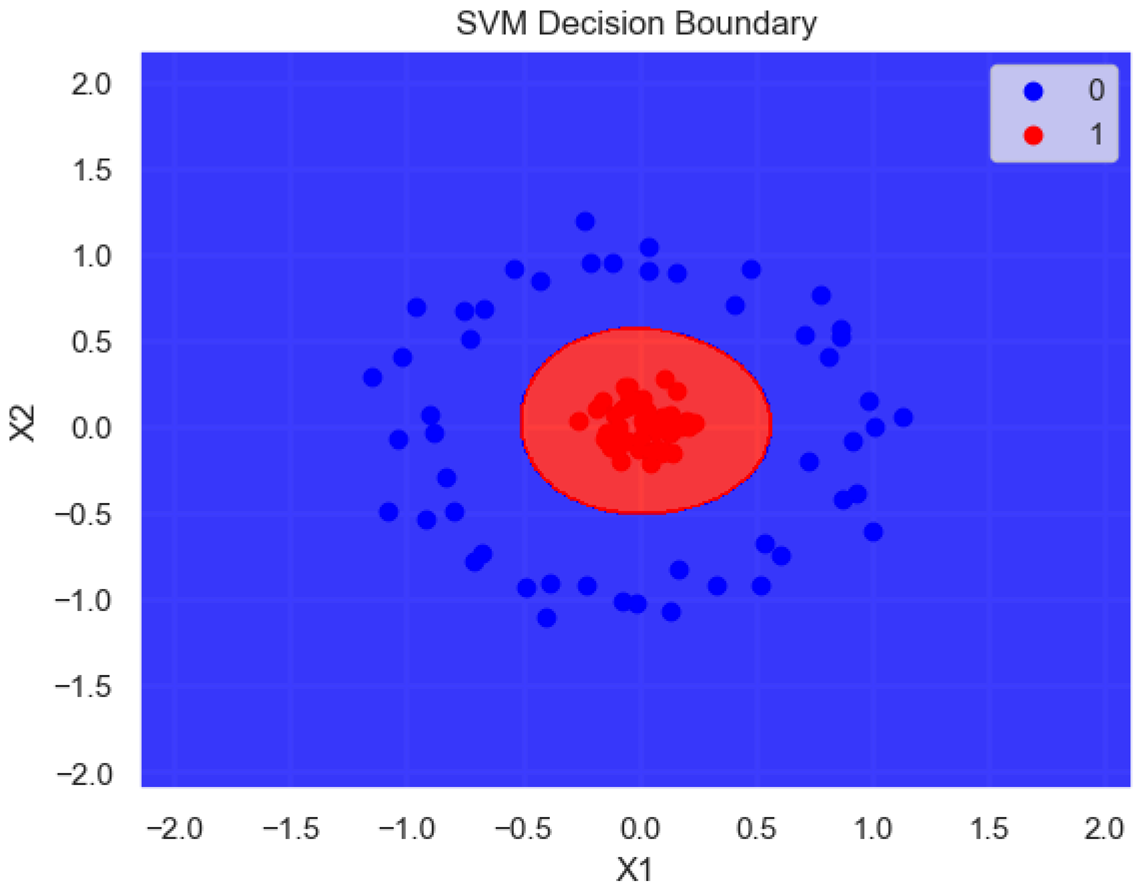
```
In [33]: rbf_classifier = SVC(kernel="rbf")
         rbf_classifier.fit(X_train, y_train)
         y_pred = rbf_classifier.predict(X_test)
```

```
In [34]: accuracy_score(y_test, y_pred)
```

```
Out[34]: 1.0
```

```
In [35]: plot_decision_boundary(X, y, rbf_classifier)
```

```
C:\Users\nayan\AppData\Local\Temp\ipykernel_19720\3603277588.py:18: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be av
oided as value-mapping will have precedence in case its length matches with *x* &
*y*.  Please use the *color* keyword-argument or provide a 2D array with a single
row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



In [36]:

```
poly_classifier = SVC(kernel="poly",degree=2)
poly_classifier.fit(X_train, y_train)
y_pred = poly_classifier.predict(X_test)
```

In [37]:

```
accuracy_score(y_test, y_pred)
```

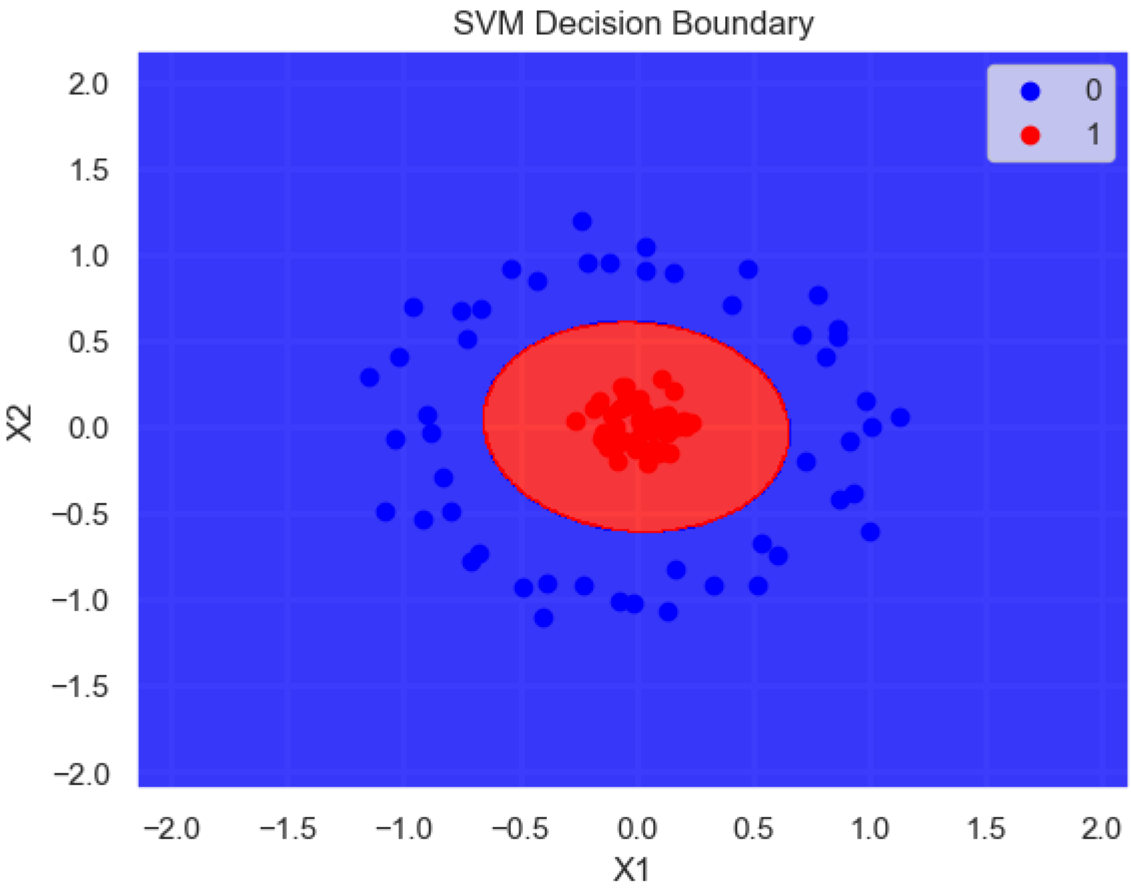
Out[37]:

```
1.0
```

In [38]:

```
plot_decision_boundary(X, y, poly_classifier)
```

```
C:\Users\nayan\AppData\Local\Temp\ipykernel_19720\3603277588.py:18: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be av
oided as value-mapping will have precedence in case its length matches with *x* &
*y*.  Please use the *color* keyword-argument or provide a 2D array with a single
row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```


SVM Decision Boundary
0
1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
X2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X1

In [39]:

```
X
```

```
Out[39]:  array([[-0.26270179,  0.03970227],
                 [ 0.15851424,  0.88928043],
                 [ 0.10676569, -0.13669366],
                 [ 0.98615457,  0.15272451],
                 [ 0.04411998,  0.0434028 ],
                 [ 0.77574929,  0.76813534],
                 [-0.10263278, -0.01275311],
                 [ 0.12174538, -0.03841417],
                 [ 1.01186311,  0.00311822],
                 [ 0.23277584,  0.02826321],
                 [-0.12619366, -0.116463  ],
                 [-0.05515501, -0.08094575],
                 [ 0.01862415,  0.00461327],
                 [ 0.05100537, -0.15087725],
                 [-0.07242467,  0.10586939],
                 [-0.90334036,  0.07285372],
                 [-0.21215085,  0.94799041],
                 [-0.12043496,  0.95370894],
                 [ 0.86206195,  0.56606022],
                 [ 0.01108062,  0.16356915],
                 [-0.54147001,  0.91410441],
                 [-0.22890293, -0.92006256],
                 [-0.82831043, -0.28564595],
                 [ 0.00673765, -0.07159252],
                 [ 0.01535918,  0.0370715 ],
                 [-0.08879698, -0.00505482],
                 [ 0.33226009, -0.91409394],
                 [-0.9146175 , -0.53459254],
                 [ 0.15632662,  0.20664411],
                 [-0.09970809, -0.04435902],
                 [-0.70824629, -0.77255713],
                 [-0.13198269, -0.03628261],
                 [-0.15837101,  0.15664499],
                 [ 0.14336058, -0.15329341],
                 [-0.42729486,  0.85272225],
                 [ 0.06563228, -0.02720204],
                 [-0.88699994, -0.03833978],
                 [ 0.40927999,  0.70949701],
                 [ 0.53666816, -0.67152132],
                 [-0.01221414, -1.01824442],
                 [ 1.12805737,  0.05457372],
                 [-0.10584043, -0.08117653],
                 [-0.73222533,  0.51293125],
                 [ 0.91780385, -0.07609075],
                 [ 0.10374702,  0.28398464],
                 [ 0.03512168,  1.04001848],
                 [ 0.01717682,  0.07160302],
                 [ 0.13011817,  0.04984664],
                 [-0.19005556,  0.10251981],
                 [-0.11194241,  0.07142231],
                 [ 0.09110143, -0.01651711],
                 [-0.23404367,  1.19153535],
                 [-0.66724402,  0.68880684],
                 [-0.9634286 ,  0.69470227],
                 [-0.00552213, -0.12520917],
                 [-1.08129253, -0.49179551],
                 [-0.04460026,  0.18399554],
                 [-0.49140285, -0.93025292],
                 [ 0.19882735,  0.03413714],
                 [ 0.93062065, -0.37909995],
                 [ 0.08607156, -0.15092002],
                 [-0.15528478, -0.07180977],
                 [ 0.21137088, -0.00226574],
                 [-0.07810824, -0.08538256],
                 [ 0.13227709, -1.07061114],
                 [ 0.03585299,  0.90177966],
                 [ 0.17518971, -0.00473621],
                 [ 0.4749785 ,  0.91542188],
                 [ 0.863474  ,  0.51942117],
                 [-1.0247355 ,  0.40564335],
                 [-0.40351659, -1.10355181],
                 [ 0.81635167,  0.40905497],
                 [-0.67341072, -0.73230941],
                 [ 0.13665099,  0.06748889],
                 [-0.38699417, -0.90006501],
                 [ 0.13713203,  0.02354001],
                 [-0.06169993,  0.22771108],
                 [-1.0407244 , -0.07450696],
                 [ 0.72988111, -0.19842039],
                 [-0.08143947, -0.19598535],
                 [-1.14988424,  0.29374998],
                 [ 0.87683523, -0.41966637],
                 [-0.14203945, -0.03828265],
                 [ 0.02955128,  0.08965659],
                 [ 0.1667348 , -0.81984713],
                 [-0.05448835,  0.12136071],
                 [-0.79617972, -0.48938396],
                 [-0.09679756, -0.12275809],
                 [ 0.044721  , -0.20477344],
                 [-0.02059797,  0.12585555],
                 [-0.07550248, -1.014038  ],
                 [ 0.0993392 ,  0.05906298],
                 [ 0.51671625, -0.91938228],
                 [ 0.60850238, -0.73795593],
                 [ 1.00124657, -0.60809698],
                 [ 0.07587995, -0.12341192],
                 [-0.0467429 ,  0.23199883],
                 [ 0.71070918,  0.53187104],
                 [ 0.13526671, -0.03597758],
                 [-0.75606671,  0.66991993]])
```

In [40]:

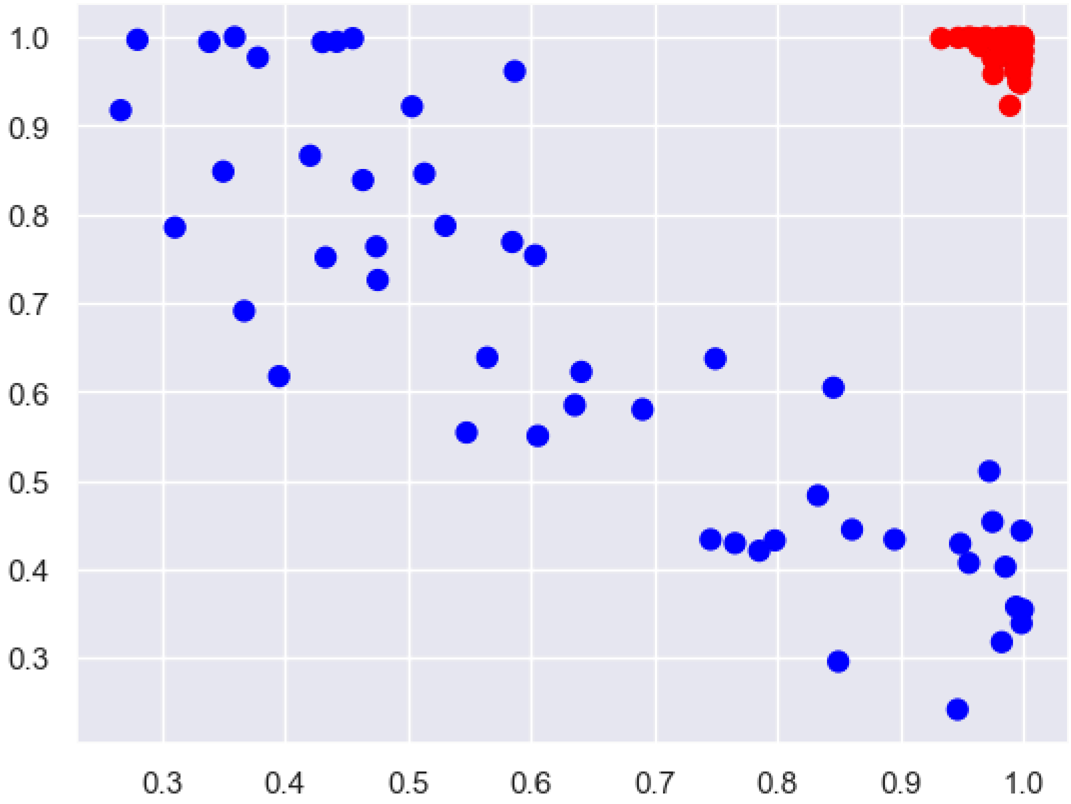
```
np.exp(-(X**2)).sum(1)
```

```
Out[40]:  array([1.93174024, 1.42865923, 1.97015414, 1.35508134, 1.99617329,
                 1.1021414 , 1.98935917, 1.98381281, 1.35919351, 1.94645874,
                 1.97072923, 1.99043175, 1.99963192, 1.97489503, 1.98362265,
                 1.43689278, 1.36309417, 1.38830118, 1.20145408, 1.9734771 ,
                 1.1795005 , 1.37785951, 1.42518338, 1.99484223, 1.99839077,
                 1.99212055, 1.329107  , 1.18463427, 1.93405527, 1.98814176,
                 1.15609936, 1.98141585, 1.9509915 , 1.95643258, 1.31640877,
                 1.99496199, 1.45384473, 1.45024848, 1.38678056, 1.35443068,
                 1.2771528 , 1.98229235, 1.35366007, 1.42491813, 1.9118133 ,
                 1.3378068 , 1.99459113, 1.98073018, 1.9540682 , 1.98245893,
                 1.9914621 , 1.18846965, 1.26291094, 1.01243636, 1.98441442,
                 1.09578122, 1.96472508, 1.20636264, 1.96007423, 1.28674069,
                 1.97009964, 1.97103167, 1.95630056, 1.986654  , 1.30049463,
                 1.44214927, 1.96975233, 1.23060961, 1.23798899, 1.19818705,
                 1.14561186, 1.35946244, 1.22033362, 1.97695538, 1.30571672,
                 1.98081654, 1.94566941, 1.33300604, 1.54839666, 1.95570762,
                 1.18386829, 1.30206793, 1.97856246, 1.99112102, 1.48319232,
                 1.98241493, 1.3175415 , 1.97571742, 1.95693686, 1.98386098,
                 1.35193961, 1.9866979 , 1.19512044, 1.27063027, 1.05784781,
                 1.97914367, 1.94541687, 1.35704576, 1.98057574, 1.20299891])
```

```
In [41]:  X_new=np.exp(-(X**2))
```

```
In [42]:  plt.scatter(X_new[:, 0], X_new[:, 1], c=y, s=50, cmap='bwr')
```

```
Out[42]:  <matplotlib.collections.PathCollection at 0x1d33a2d63d0>
```



```
In [ ]:
```