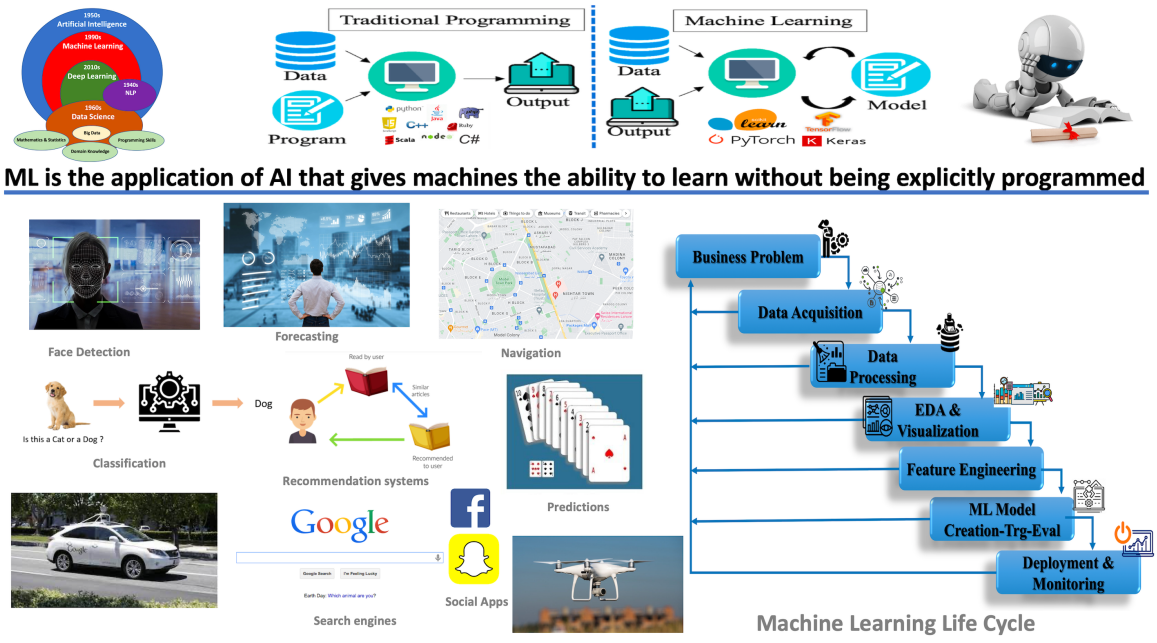
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.29 (Support Vector Machines Part-II)

Open in Colab (https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)



Learning agenda of this notebook

Section I: (Soft Margin SVM)

- Recap of Hard Margin SVM
- Issues with Hard Margin SVM
- Soft Margin SVM
  - Math Behind Soft Margin SVM
  - Python Implementation: (Soft Margin SVM)

- Perfectly Linearly Separable Dataset
- Almost Linearly Separable Dataset
- What about Not Linearly Separable Dataset?

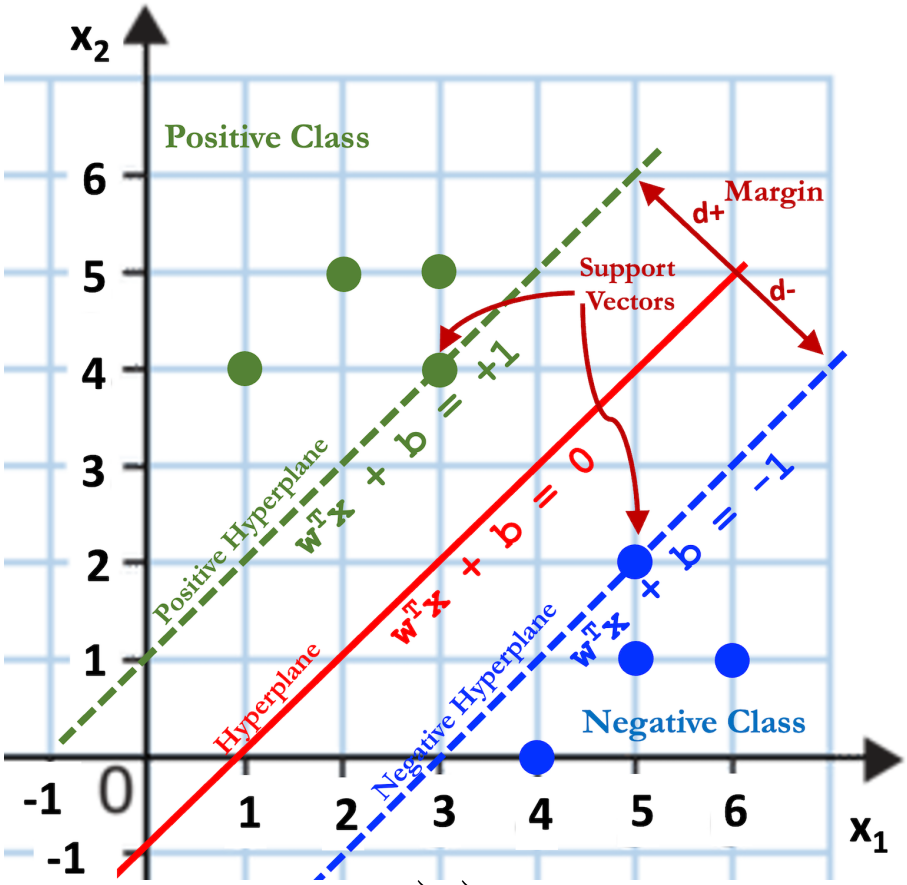
Section II: (Kernel Trick in SVM)

- Handling Non Linear Data
- Mapping Non-Linear data to higher dimensions
- SVM Kernel Functions and its Types
  - Linear Kernel
  - Polynomial Kernel
  - Radial Basis Function (RBF) Kernel
  - Sigmoid Kernel
  - Laplace RBF Kernel
  - Hyperbolic Tangent Kernel
  - Bessel Function Kernel
  - Anova RBF Kernel
  - Linear Spline Kernel
- Python Implementation
  - Linear Kernel
  - Polynomial Kernel
  - RBF Kernel
- Tasks to Do

Section I: (Soft Margin SVM)

1. Recap of Hard Margin SVM

- General Rule to Classify a New Data Point:



$$\hat{y}_i = \begin{cases} +1, & \text{if} \quad \vec{w}\vec{x_i} + b \geq 0 \\ -1, & \text{if} \quad \vec{w}\vec{x_i} + b < 0 \end{cases}$$

- **Constraint for Hard SVM:**
  - We need to maximize the margin, such that
    - NO training point is `below the Positive Hyperplane` :
      $$\vec{w}\,\vec{x} + b \geq 1$$
    - NO training point is `above the Negative Hyperplane` :
      $$\vec{w}\,\vec{x} + b < -1$$
  - Simplify the above two equation to one, assuming that the label of all the positive training points is $+1$ and the label of all the negative training points is -1:

$$y_i(\vec{w}\,\vec{x_i} + b) \geq 1$$

$$argmax_{(w^*,b^*)} \quad \frac{2}{\|w\|}$$

$$argmax_{(w^*,b^*)} \quad \frac{1}{\|w\|}$$

$$argmin_{(w^*,b^*)} \quad \|w\|$$

$$argmin_{(w^*,b^*)} \quad \frac{1}{2}\|w\|^2, \qquad \text{such tha}$$

Langrange Multipliers:

$$L = \frac{1}{2}\|w\|^2 - \sum \alpha_i [y_i(\vec{w}\,\vec{x_i} + b) - 1]$$

$$\frac{\partial L}{\partial w} = \vec{w} - \sum \alpha_i y_i \vec{x_i} \qquad \frac{\partial L}{\partial b} = -\sum \alpha_i y_i$$

$$\vec{w} - \sum \alpha_i y_i \vec{x_i} = 0 \qquad -\sum \alpha_i y_i = 0$$

$$\vec{w} = \sum \alpha_i y_i \vec{x_i} \qquad \sum \alpha_i y_i = 0$$

- Plug in the value of $\vec{w}$ in above Loss function

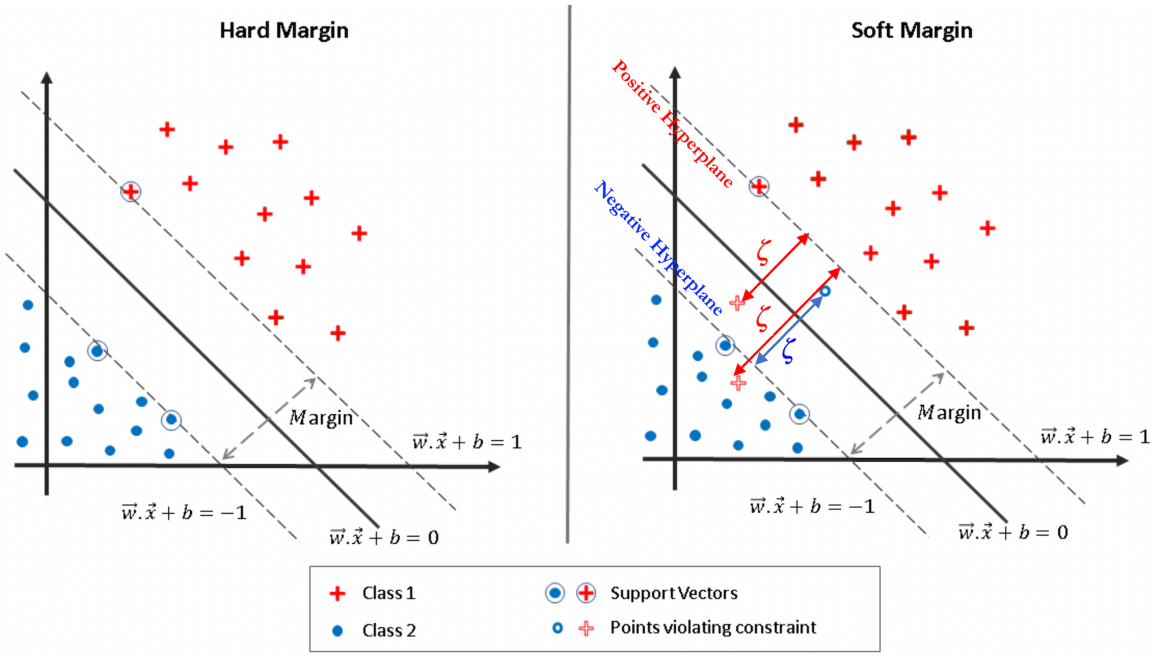
$$L = \frac{1}{2}(\sum \alpha_i y_i x_i).(\sum \alpha_i y_i x_i) - (\sum \alpha_i y_i x_i).(\sum \alpha_j y$$

$$L = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j x_i . x_j$$

2. Soft Margin SVM

- The **hard margin** is the decision boundary that maximizes the margin and separates the classes perfectly. This works well when the data is linearly separable, but it is sensitive to outliers and noise.
- The **soft margin** allows for some misclassification of the data points in order to maximize the margin. This is useful when the data is not linearly separable and there are some outliers or noise in the data.

> In **Soft Margin** we allow SVM to make a certain number of mistakes and keep margin as wide as possible so that other points can still be classified correctly.



$$argmin_{(w^*,b^*)} \quad \frac{1}{2}\|w\|^2, \qquad \text{such}$$

$$argmin_{(w^*,b^*)} \quad \frac{1}{2}\|w\|^2 + C\sum \zeta_i, \qquad \text{such}$$

Langrange Multipliers:

$$L = \frac{1}{2}\|w\|^2 - \sum \alpha_i [y_i(\vec{w}\vec{x_i} + b) - 1]$$

ζ_i:

- For every data point x_i, we introduce a slack variable ζ_i. The value of ζ_i is the distance of x_i from the corresponding class's margin if x_i is on the wrong side of the margin, otherwise zero.
- For all `positive` points `above` the `positive hyperplane` , the ζ value will be zero.
- For all `negative` points `below` the `negative hyperplane` , the ζ value will be zero.
- For a positive data point below the decision boundry the corresponding ζ value will be greater than 1, while for a positive data point that is on the correct side of the decision boundry, but within the margin will have a ζ value between 0 and 1.
- For a negative data point abive the decision boundry the corresponding ζ value will be greater than 1, while for a negative data point that is on the correct side of the decision boundry, but within the margin will have a ζ value between 0 and 1.
- Thus the points that are far away from the margin on the wrong side would get more penalty.

C:

- The regularization hyperparameter `C` controls the trade-off between maximizing the margin and minimizing the misclassification.
- A very small value of C, makes the second term of above expression small, thus the model will ignore classification errors and will focus on increasing the margin.
- A large value of C will not concentrate on maximizing the margin, rather will focus on rducing the number of miss-classified points.

3. Implementation of Soft Margin Support Vector Classifier

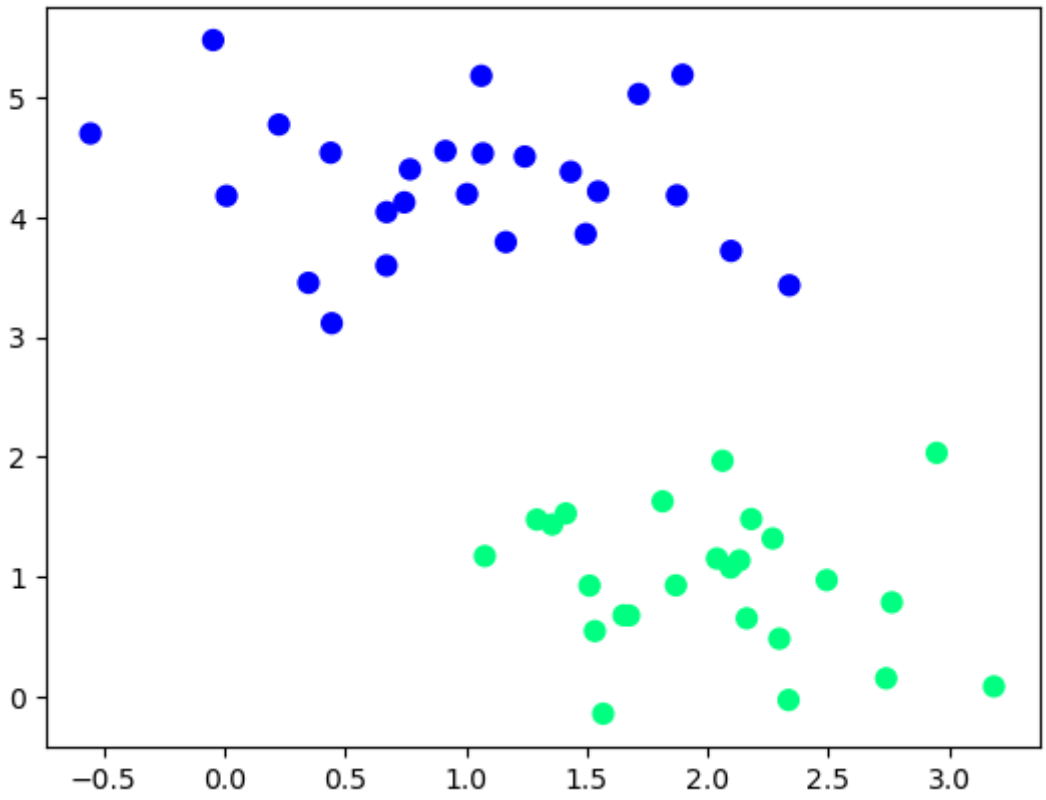
In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# User defined plotting function
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)
```

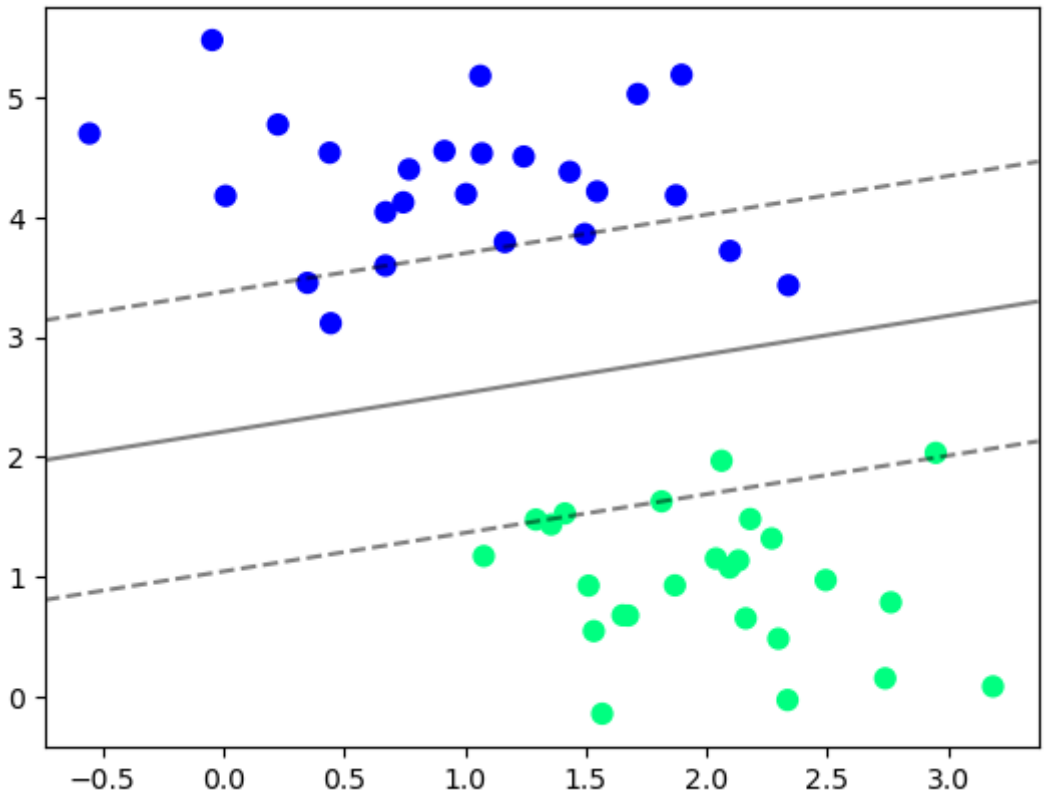
a. Perfectly Linearly Separable Dataset

In [2]:
```
from sklearn import datasets

X, y = datasets.make_blobs(n_samples=50, n_features=2, centers=2, clus
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='winter');
print("Shape of X array: ",X.shape)
print("Shape of y array: ",y.shape)
```

```
Shape of X array:  (50, 2)
Shape of y array:  (50,)
```

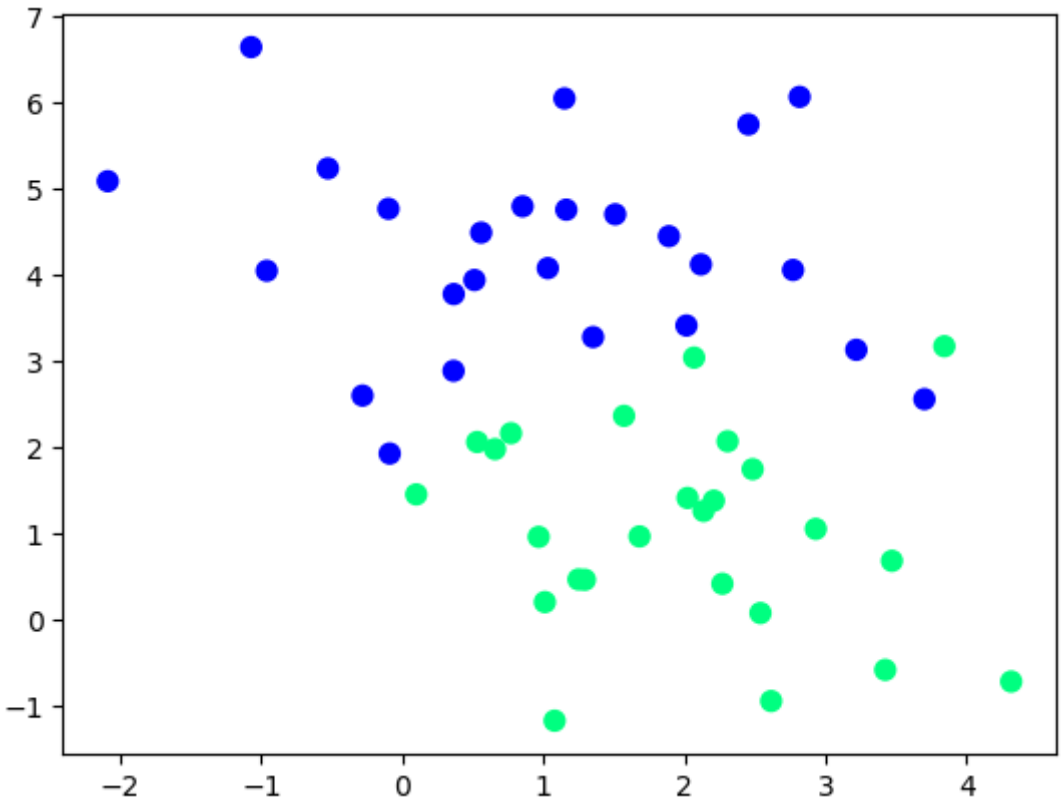


In [3]:
```
from sklearn.svm import SVC

model = SVC(kernel='linear', C=0.1)
model.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='winter')
plot_svc_decision_function(model);
```

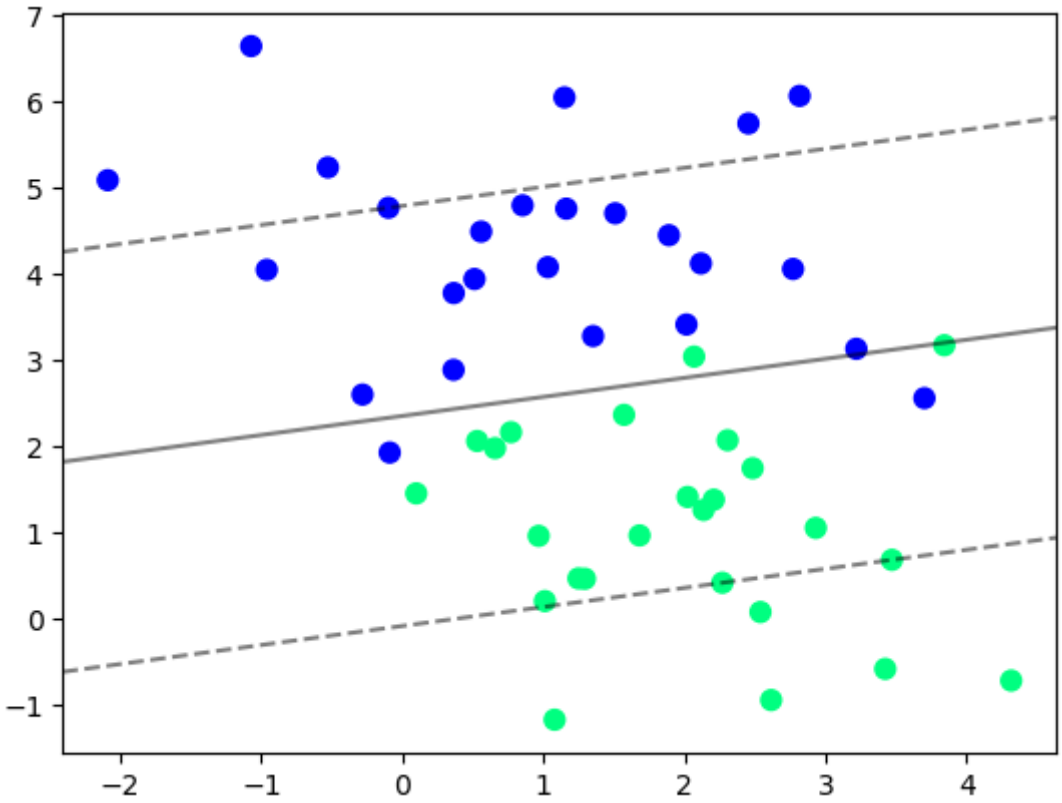


b. Almost Linearly Separable Dataset

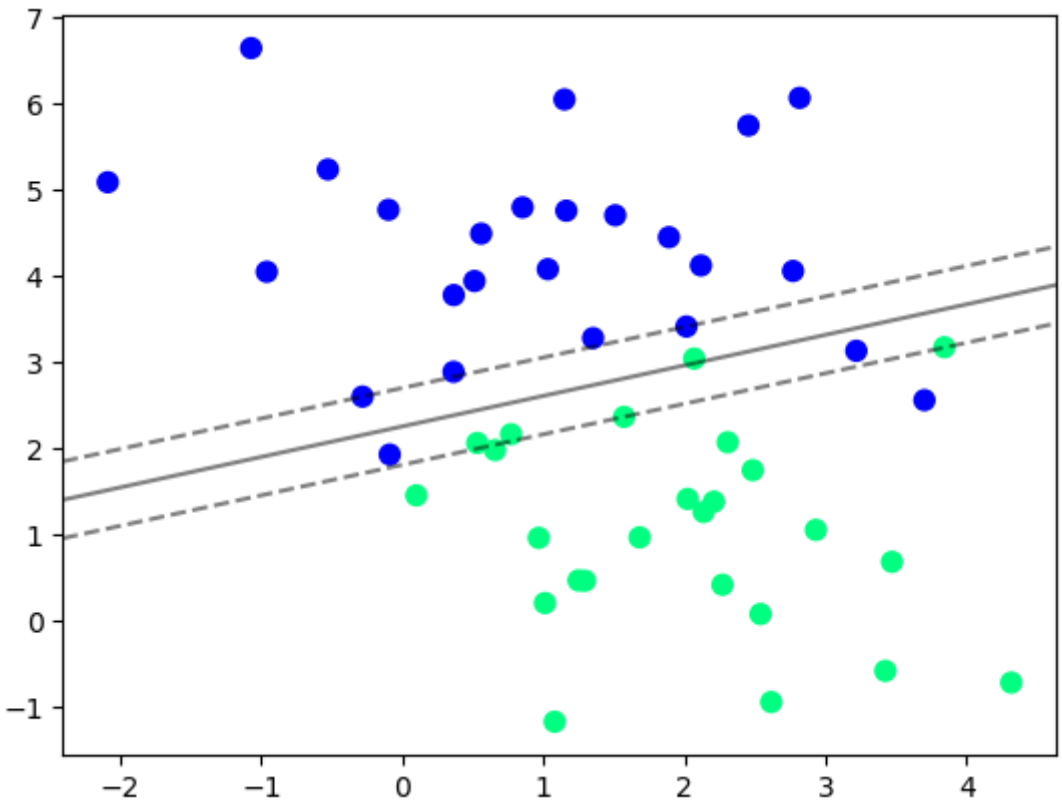
In [4]:
```
X, y = datasets.make_blobs(n_samples=50, n_features=2, centers=2, clus
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='winter');
```



In [5]:
```
# With a small value of C, the model will ignore classification errors
model = SVC(kernel='linear', C=0.01)
model.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='winter')
plot_svc_decision_function(model);
```

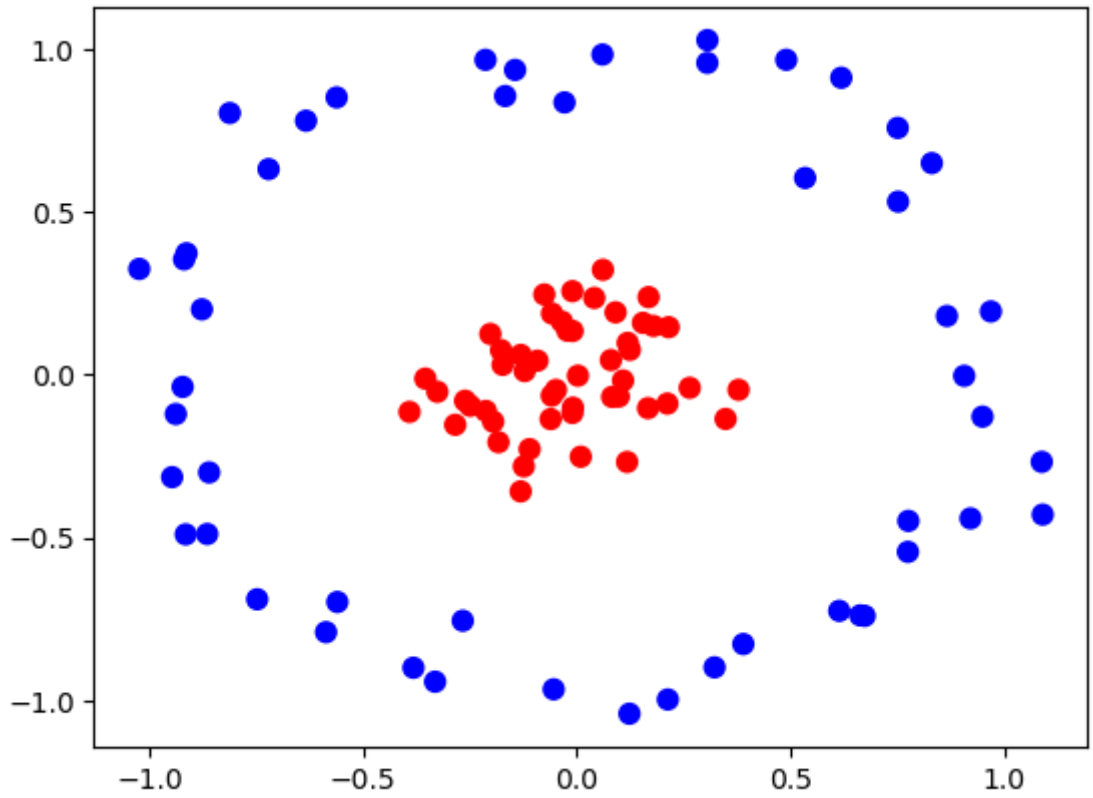


In [6]:
```
# With a large value of C, the model will not concentrate on maximizir
# rather will focus on rducing the number of miss-classified points.
model = SVC(kernel='linear', C=100)
model.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='winter')
plot_svc_decision_function(model);
```



c. What about Not Linearly Separable Dataset?

In [7]:
```
from sklearn import datasets
X, y = datasets.make_circles(100, factor=.2, noise=.1)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='bwr');
```

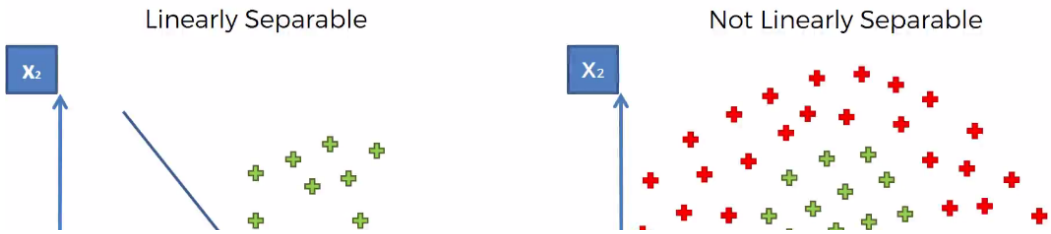


Section II: (Kernel Trick in SVM)

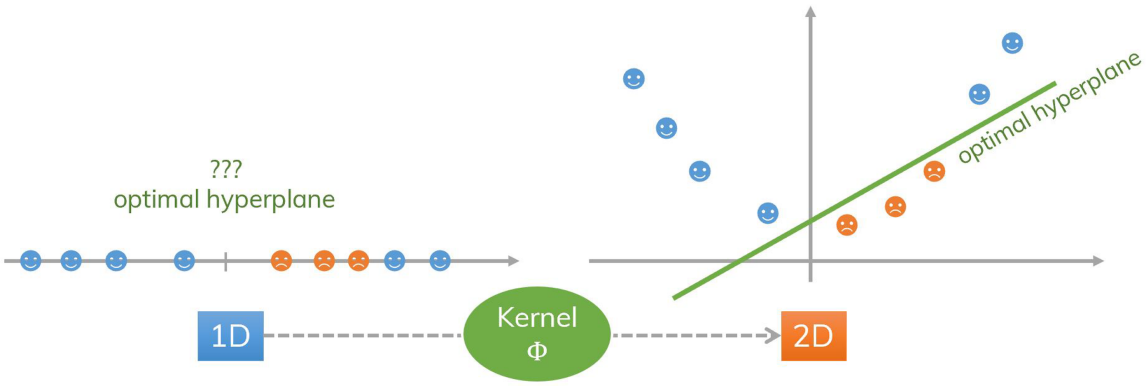
1. Classifying Non-Linear Data using SVM

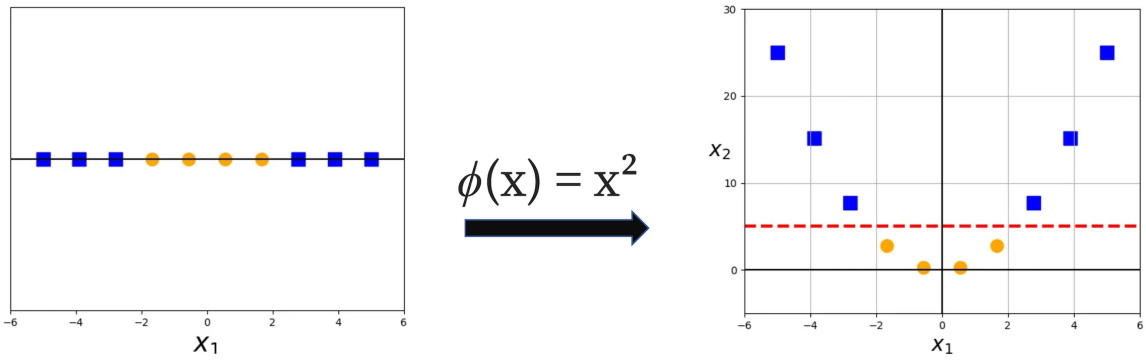
a. Linearly vs Non-Linearly Separable Data

- **Linear SVM:** When the data is perfectly linearly separable only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D).
- **Non-Linear SVM:** When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.

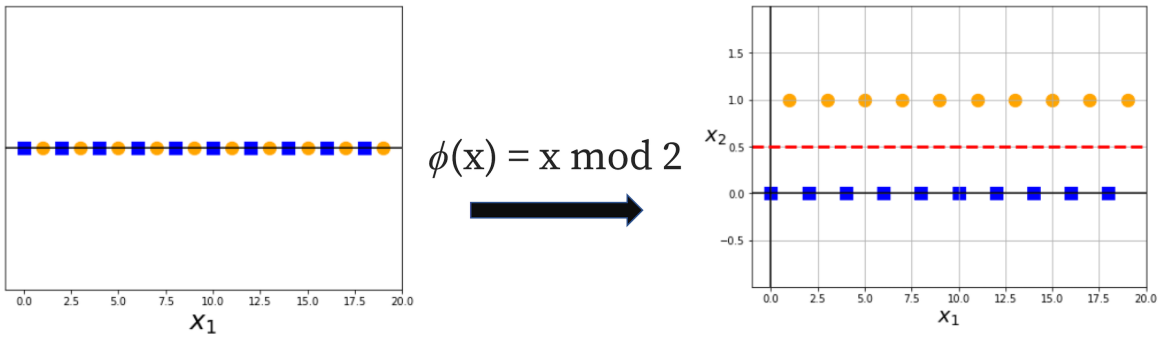


b. Mapping a 1D Dataset to 2D to make it Linearly Separable



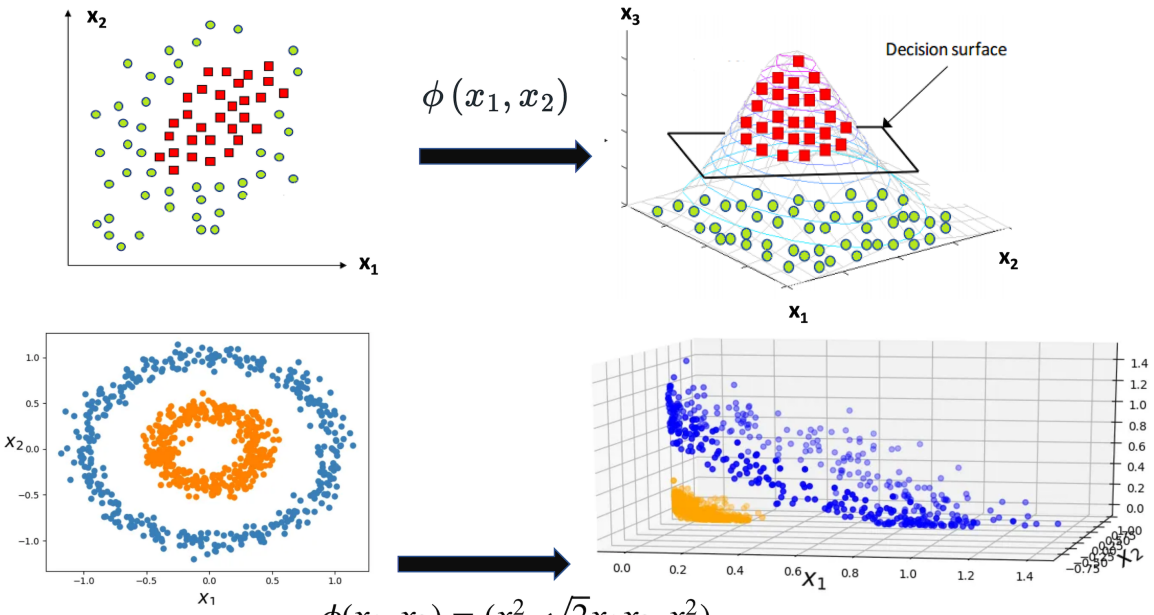


- In 1-dimension, this data is not linearly separable, but after applying the transformation ϕ(x) = x² and adding this second dimension to our feature space, the classes become linearly separable.



c. Mapping a 2D dataset to 3D to make it Linearly

Separable



$$\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$(1, 2)$ will be mapped to $(1, 2\sqrt{2}, 4)$ and $(3, 4)$ will be mapped to $(9, 12\sqrt{2}, 16)$

- We need to choose what non-linear transformation should be applied to a data set?

- If we want a sophisticated decision boundry, we may need to increase the dimensions of the transformations, which in turn increases computational requirements specially for large

2. SVM Kernel Trick

A kernel function is a mathematical function that takes as its inputs, vectors in the original (lower dimensional) space, and return the dot product of the transformed vectors in the feature (higher dimensional) space.

- Linear Kernel
- Polynomial Kernel
- Radial Basis Function (RBF) Kernel
- Sigmoid Kernel
- Laplace RBF Kernel
- Hyperbolic Tangent Kernel
- Bessel Function Kernel
- Anova RBF Kernel
- Linear Spline Kernel

a. Linear Kernel

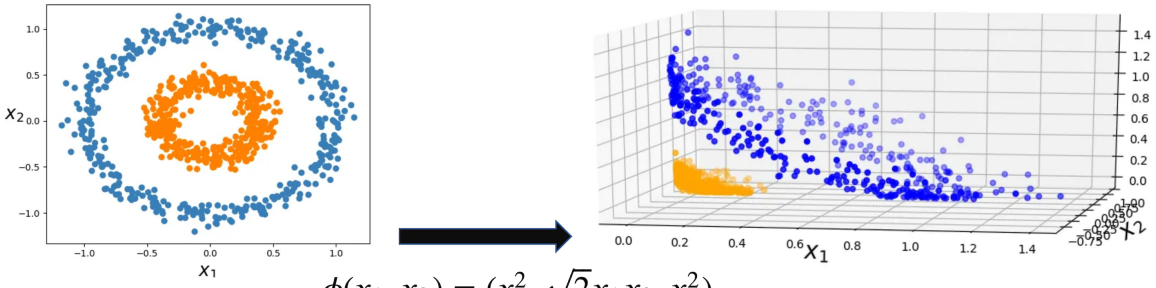
- The most simplest of all is Linear Kernel, usually one dimensional in nature.

$$K(\overrightarrow{x_1}, \overrightarrow{x_2}) = \phi(\overrightarrow{x_1}).\phi(\overrightarrow{x_2}) = \overrightarrow{x_1}$$

- It proves to be the best function when there are lots of features. The linear kernel is mostly preferred for text-classification problems as most of these kinds of classification problems can be linearly separated.

b. Polynomial Kernel

$$K(\overrightarrow{x_1}, \overrightarrow{x_2}) = (\overrightarrow{x_1^T}.\overrightarrow{x_2} + c)^d$$



$$\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Use Mapping Function:

- Apply mapping function of 1^{st} data point: $\phi(1,2) = (1, 2\sqrt{2}, 4)$
- Apply mapping function of 2^{nd} data point: $\phi(3,4) = (9, 12\sqrt{2}, 16)$

- Take dot product:

$$\phi(1,2).\phi(3,4) = (1, 2\sqrt{2}, 4).(9, 12\sqrt{2}, 16) = (1*9 + 2\sqrt{2}*12\sqrt{2} + 4*1($$

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\,a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\,b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b}\right)^2$$

Use Kernel Trick:

- $K(a,b) = (a^T.b)^2 = \left(\begin{bmatrix}1\\2\end{bmatrix}.\begin{bmatrix}3 & 4\end{bmatrix}\right)^2 = (1*3$

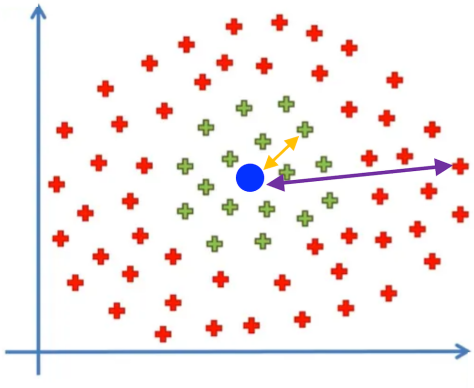
Sample Problem: Consider two data points $x_1 = (1, 2)$ and $x_2 = (2, 3)$ with $c = 1$. Calculate the value using the following kernels:

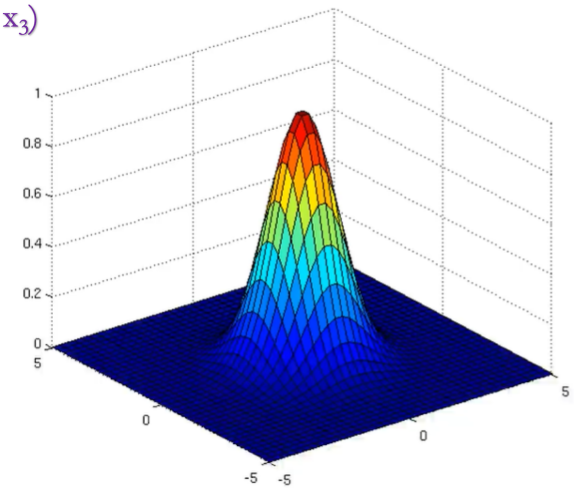
- Linear Kernel
- Quadratic Homogeneous Kernel
- Quadratic In-homogeneous Kernel

c. Radial Basis Function (RBF) Kernel

$$K(\vec{x}, \vec{l^i}) = e^{-\gamma\|x - l^i\|^2} =$$

$d_{12} < d_{13} \Rightarrow K(x_1, x_2) > K(x_1, x_3)$





- **Example 1:** Consider two data points $x_1 = (1, 2)$ and $l = (1, 2)$ with $\sigma = 1$. Calculate the value of RBF kernel

$$K(x_1, x_2) = e^{-\frac{(1-1)^2+(2-2)^2)}{2*1^2}} = e^0 = 1.0$$

- **Example 2:** Consider two data points $x_1 = (1, 2)$ and $l = (2, 3)$ with $\sigma = 1$. Calculate the value of RBF kernel

$$K(x_1, x_2) = e^{-\frac{(2-1)^2+(3-2)^2)}{2*1^2}} = e^{-1} = 0.368$$

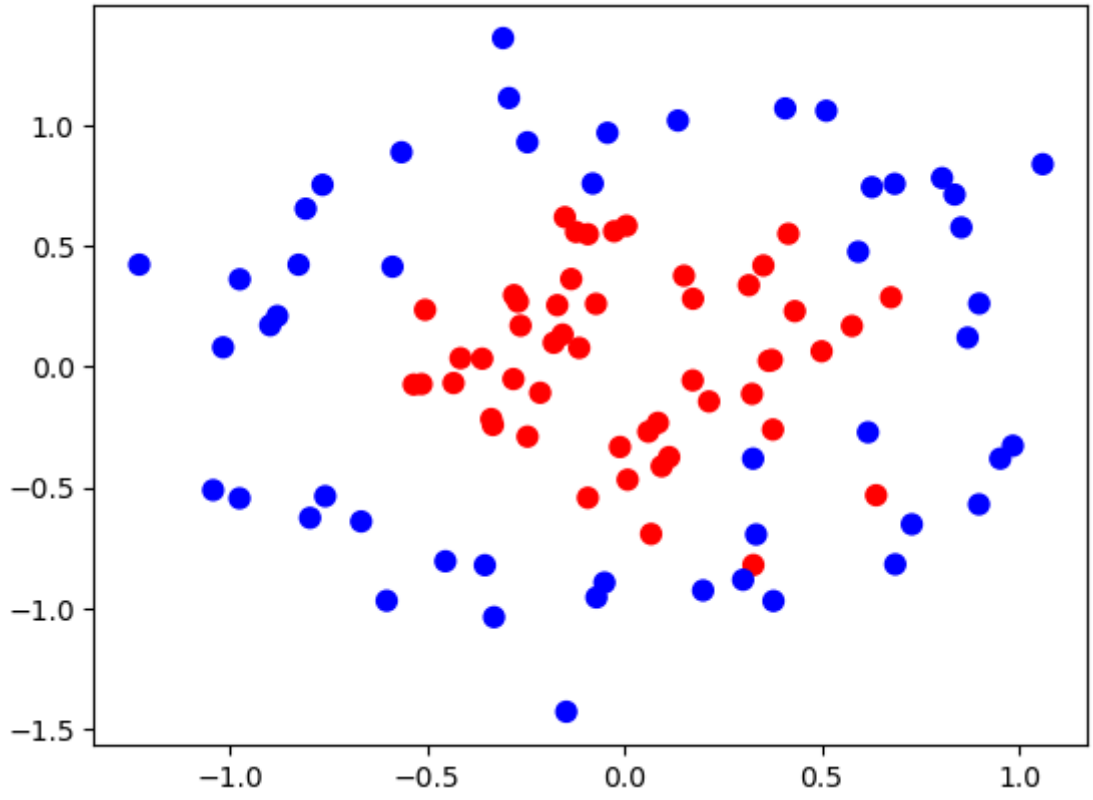
3. Implementation of Kernel Trick in SVM

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_sc

X, y = datasets.make_circles(n_samples=100, factor=0.4, noise=.2, ranc
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='bwr');
df = pd.DataFrame(dict(x1=X[:, 0], x2=X[:, 1], y=y))
df
```

Out[8]:

	x1	x2	y
0	-0.070628	-0.955298	0
1	-0.114435	0.077069	1
2	0.952296	-0.381005	0
3	0.008264	-0.468202	1
4	-0.281060	-0.051566	1
...	...	...	...
95	-0.026076	0.559787	1
96	-0.879248	0.209111	0
97	-0.292361	1.111181	0
98	0.853666	0.575727	0
99	-0.169994	0.253907	1

100 rows × 3 columns



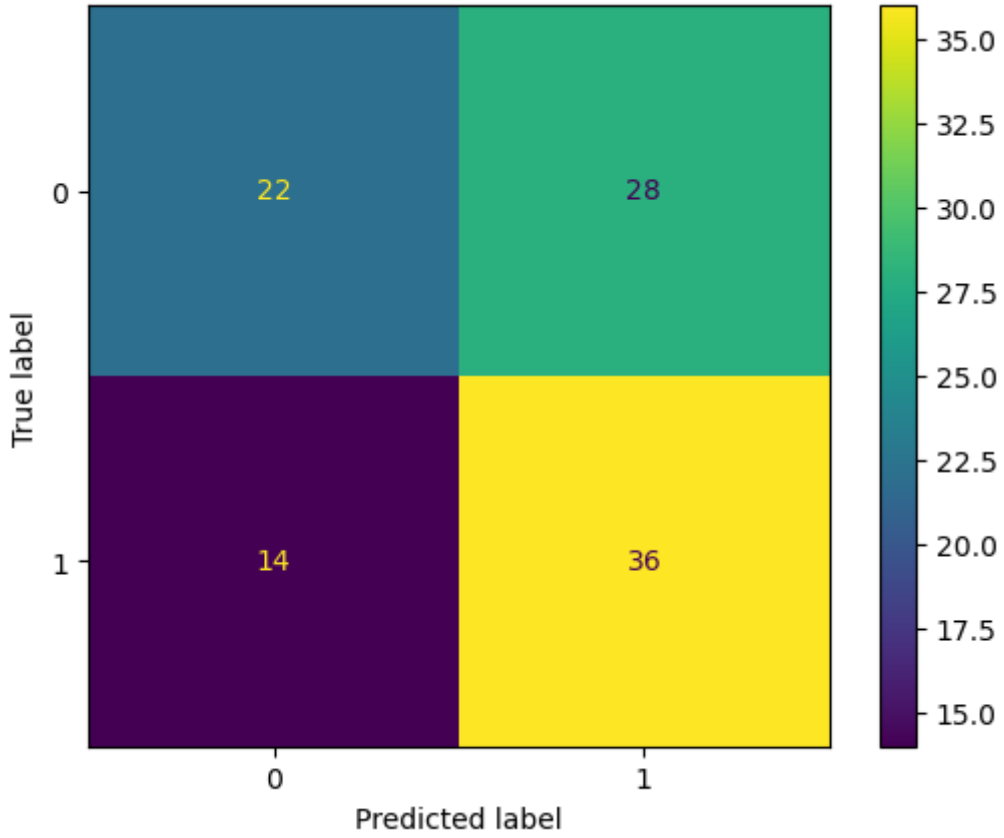
a. SVC Linear Kernel

$$K(\overrightarrow{x_1}, \overrightarrow{x_2}) = \phi(\overrightarrow{x_1}).\phi(\overrightarrow{x_2}) = \overrightarrow{x_1}^T.\overrightarrow{x_2}$$

Train the Model, Predict and Evaluate

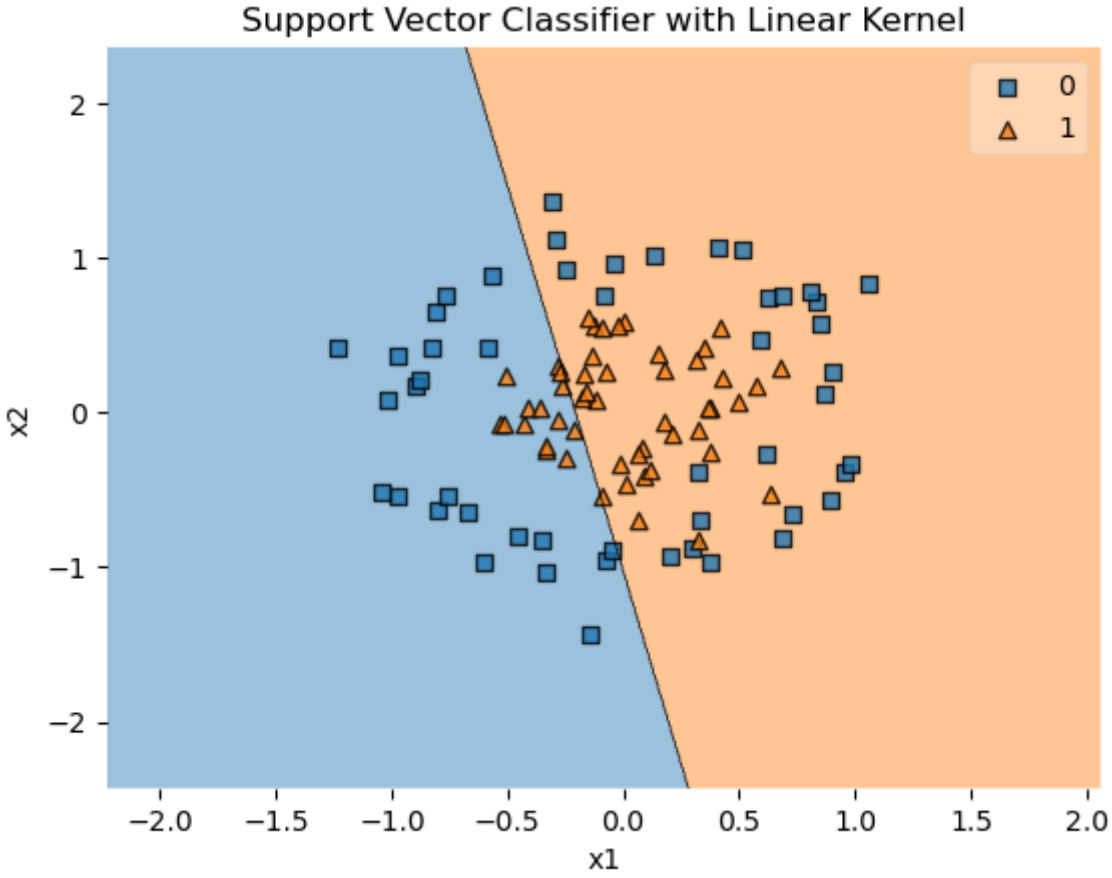
In [9]:
```
model = SVC(kernel='linear')
model.fit(X, y)
y_pred = model.predict(X)
print("Accuracy score: ", accuracy_score(y, y_pred))
print("F1 score: ", f1_score(y, y_pred))
ConfusionMatrixDisplay.from_predictions(y, y_pred);
```

```
Accuracy score:  0.58
F1 score:  0.631578947368421
```



Plot Decision Boundry for Visualizing Results

In [10]:
```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X=X, y=y, clf=model)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Support Vector Classifier with Linear Kernel')
plt.show();
```



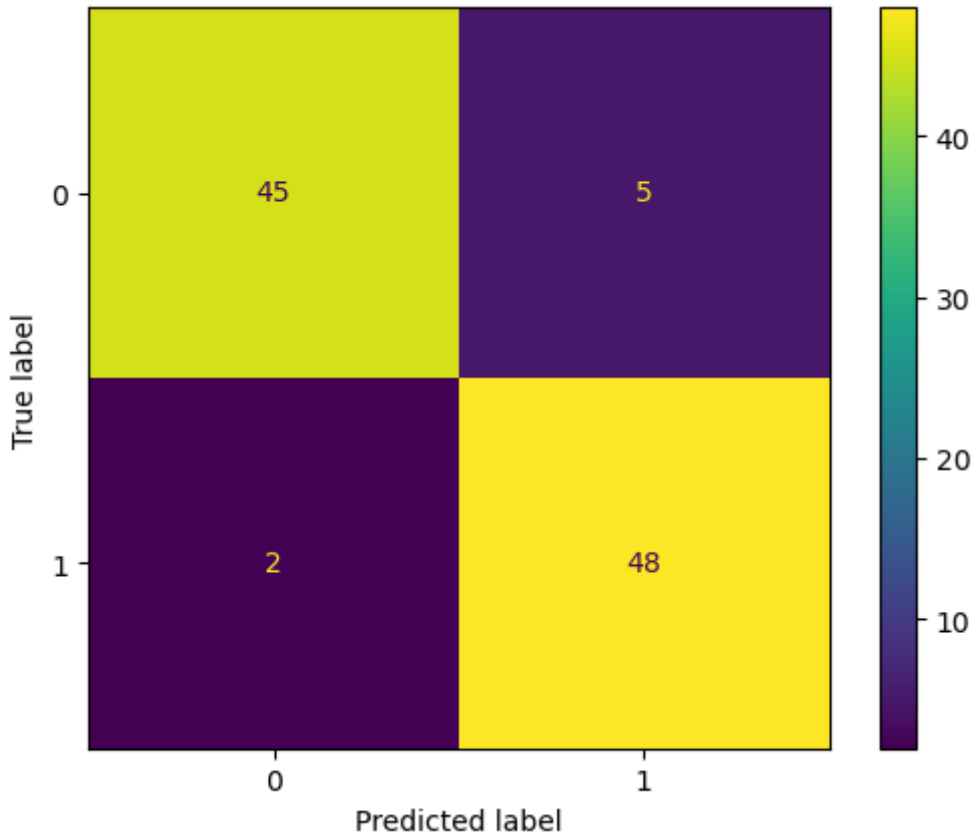
b. SVC Polynomial Kernel

$$K(\overrightarrow{x_1}, \overrightarrow{x_2}) \quad = \quad (\overrightarrow{x_1^T} . \overrightarrow{x_2} + c)^d$$

Train the Model, Predict and Evaluate

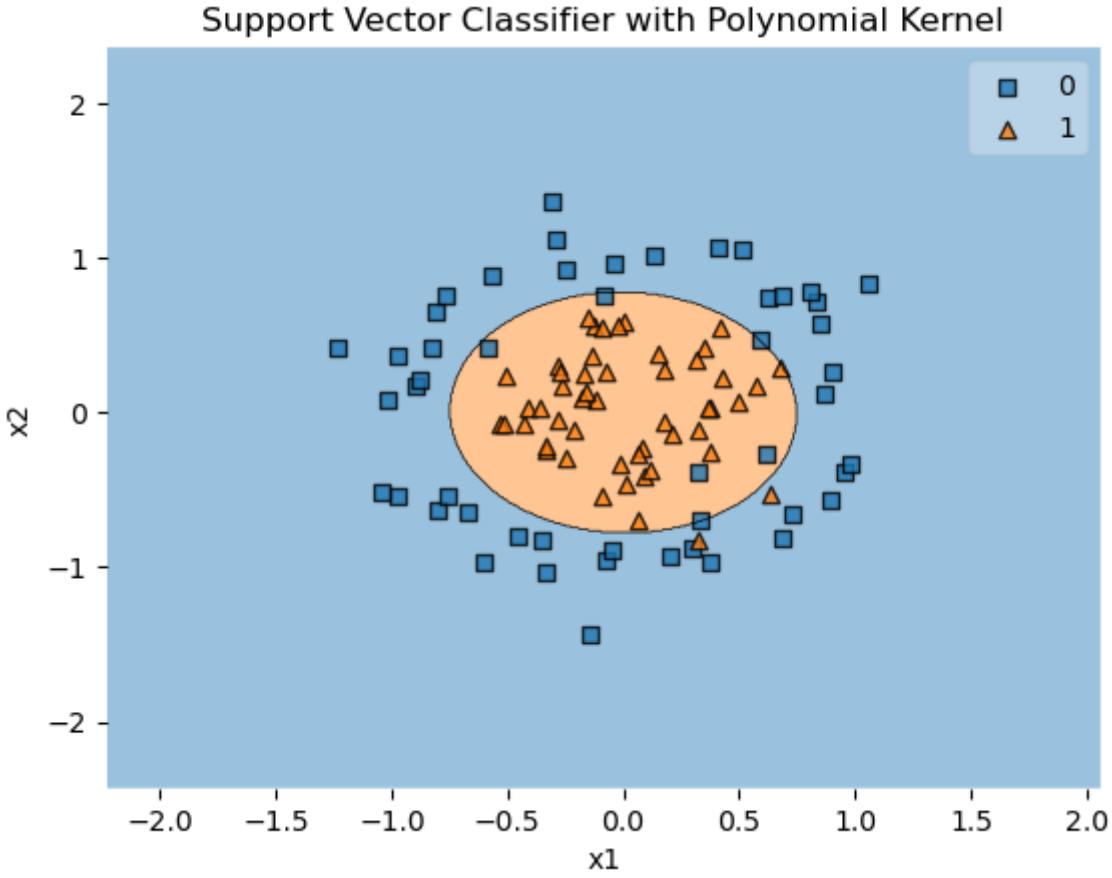
In [11]:
```
model = SVC(kernel='poly', degree=2) # Try other values of degree
model.fit(X, y)
y_pred = model.predict(X)
print("Accuracy score: ", accuracy_score(y, y_pred))
print("F1 score: ", f1_score(y, y_pred))
ConfusionMatrixDisplay.from_predictions(y, y_pred);
```

```
Accuracy score:  0.93
F1 score:  0.9320388349514563
```



Plot Decision Boundry for Visualizing Results

In [12]:
```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X=X, y=y, clf=model)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Support Vector Classifier with Polynomial Kernel')
plt.show();
```



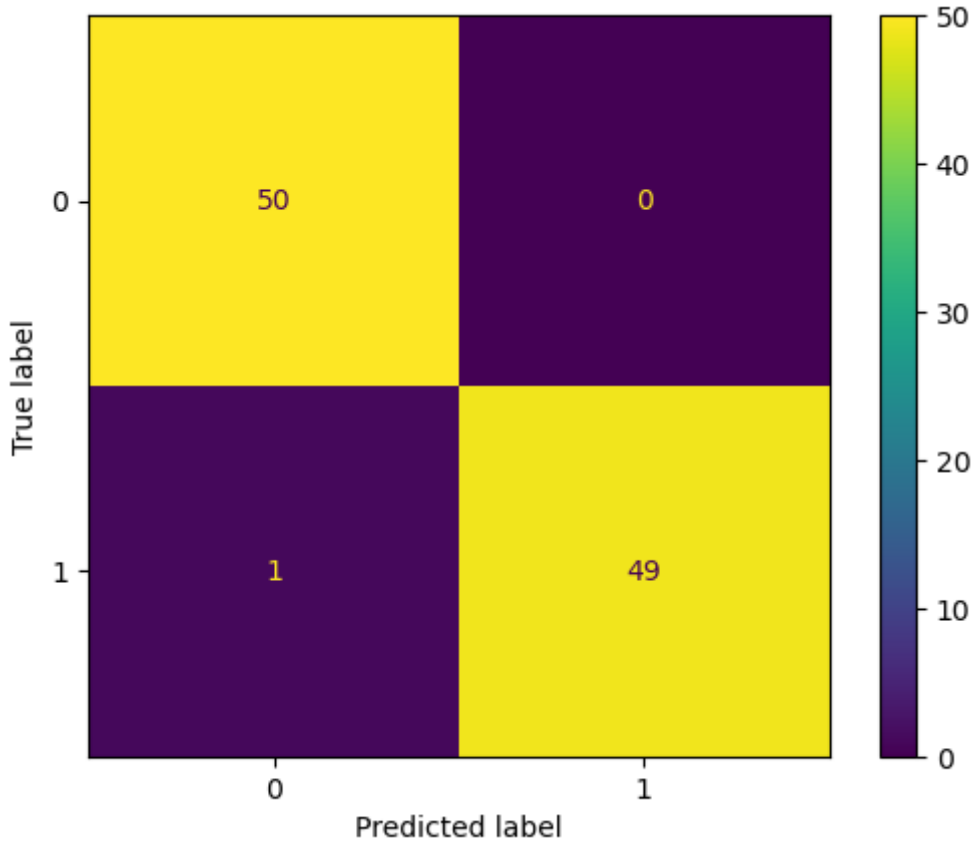
c. SVC Radio Basis Function Kernel

$$K(\vec{x}, \vec{l^i}) = e^{-\gamma\|x-l^i\|^2} = e^{-\frac{\|x-}{2\sigma}}$$

Train the Model, Predict and Evaluate

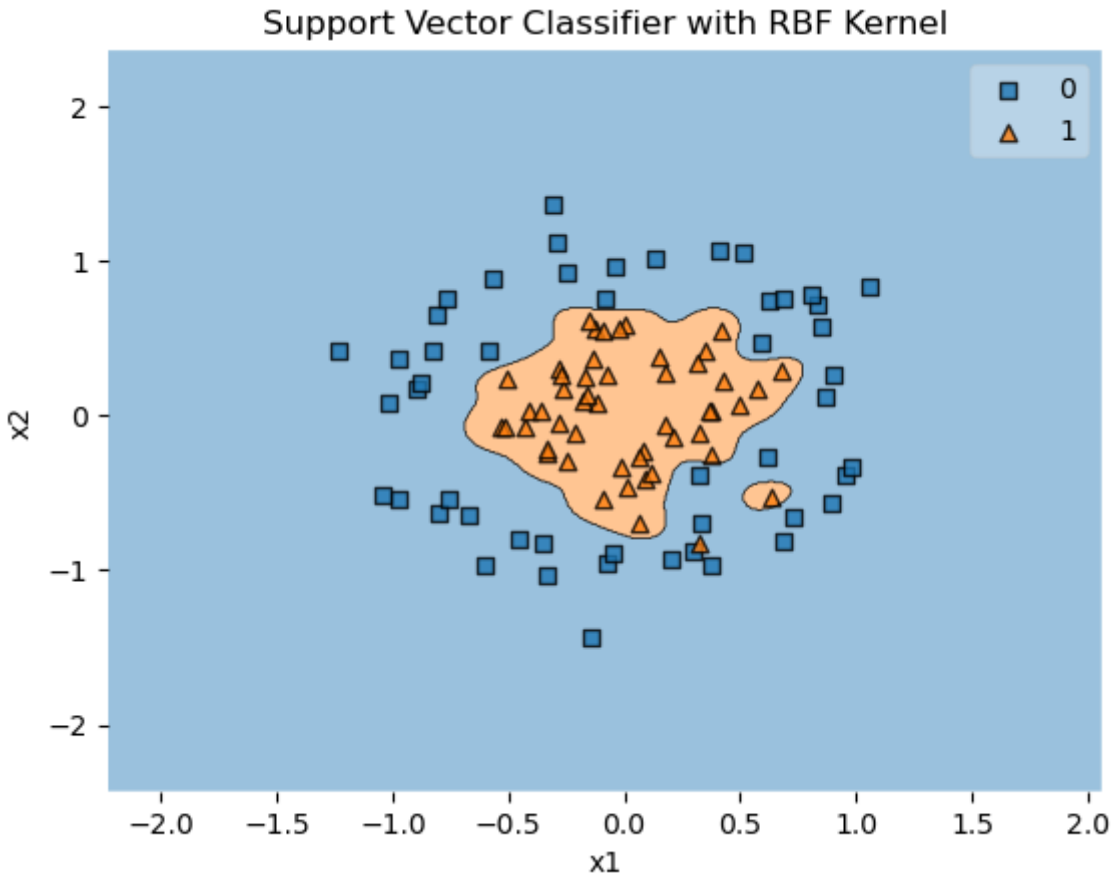
In [13]:
```
model = SVC(kernel='rbf', gamma=50)
model.fit(X, y)
y_pred = model.predict(X)
print("Accuracy score: ", accuracy_score(y, y_pred))
print("F1 score: ", f1_score(y, y_pred))
ConfusionMatrixDisplay.from_predictions(y, y_pred);
```

```
Accuracy score:  0.99
F1 score:  0.98989898989899
```



Plot Decision Boundry for Visualizing Results

In [14]:
```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X=X, y=y, clf=model)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Support Vector Classifier with RBF Kernel')
plt.show();
```



Tasks To Do

a. Purchase User-Data

In [15]:
```
df=pd.read_csv('datasets/User_Data.csv')
df
```

Out[15]:

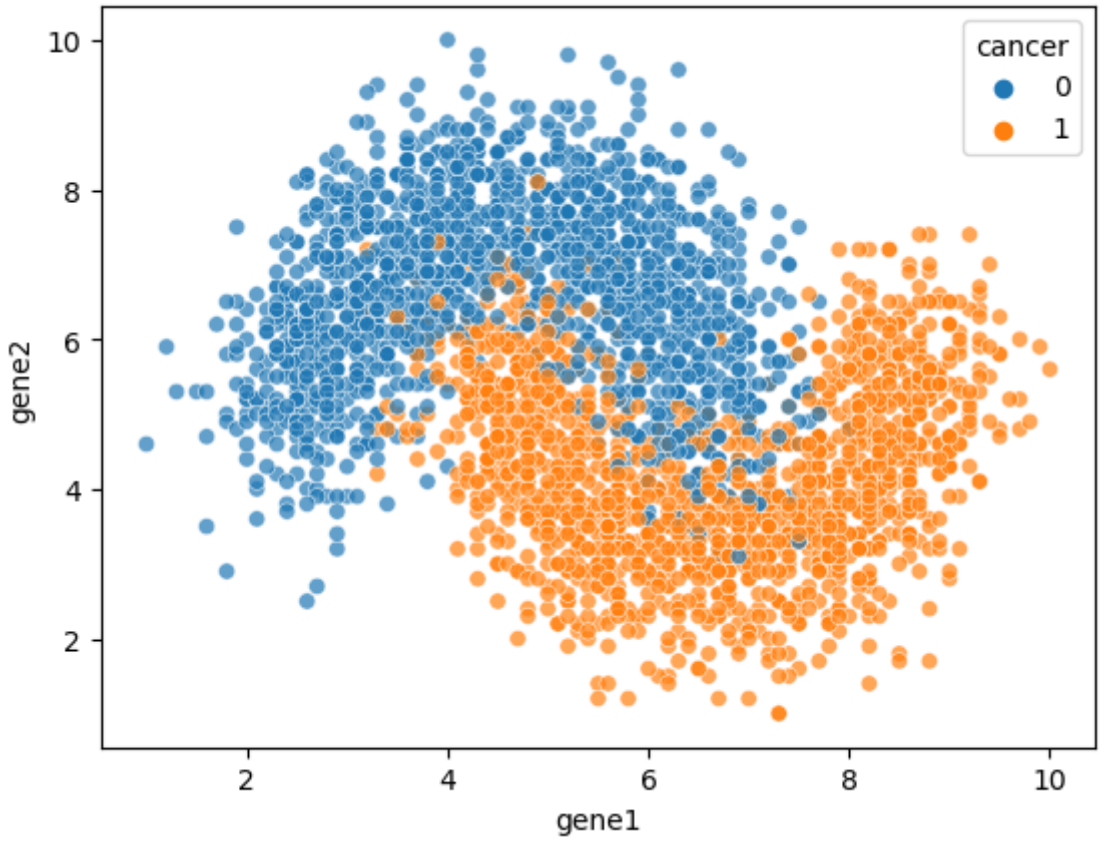
	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0
...	...	...	...	...
395	Female	46	41000	1
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

400 rows × 4 columns

b. Cancer-Genes

In [16]:
```
df = pd.read_csv('datasets/gene_expression.csv')
print(df.head())
sns.scatterplot(x='gene1',y='gene2',hue='cancer',data=df,alpha=0.7);
```

```
   gene1  gene2  cancer
0    4.3    3.9       1
1    2.5    6.3       0
2    5.7    3.9       1
3    6.1    6.2       0
4    7.4    3.4       1
```



c. Digit Classification

In [17]:
```
digits = datasets.load_digits()
df = pd.DataFrame(digits.data, columns=digits.feature_names)
df['target'] = digits.target
df.head()
```

Out[17]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_(
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.(
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.(
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.(
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.(
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.(

5 rows × 65 columns