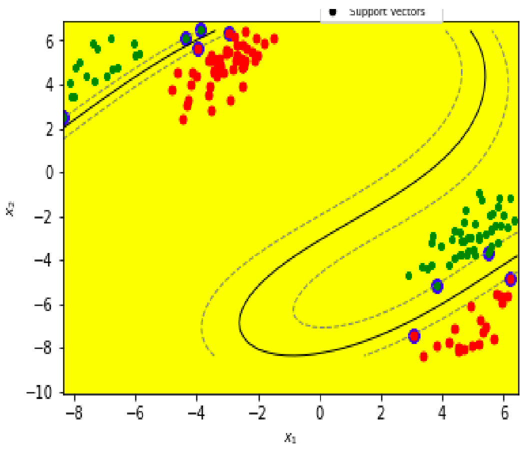
SVM with SMO Algorithm

SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.



We will be considering a linear classifier for a binary classification problem with labels y and features x.

Dataset:

$$X = (x_i, y_i)_{i=1}^{N}$$

Class labels :

$$y \in \{-1, 1\}$$

Hypothesis Function:

$$h_{w,b}(x) = g(w^T x + b).$$

Functional margin of (w, b):

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b).$$

The **goal of SVM** is to identify an optimal separating hyperplane which maximizes the margin between different classes of the training data. i.e., we want to maximize γ, subject to each training example having functional margin at least γ,

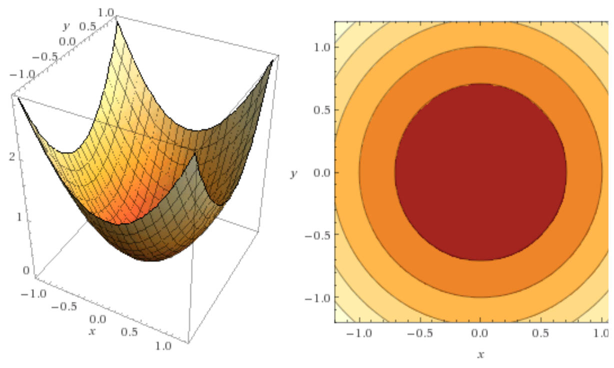
$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \end{aligned}$$

which is equivalent to minimizing,

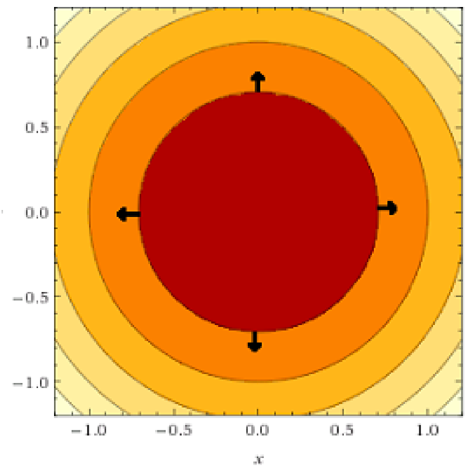
$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

Optimization:

We can find the maximum or minimum of a multivariable function with some constraint using the **Lagrange Multiplier.** It's important to understand Lagrange Multiplier to solve constraint optimization problems, like we have in SVM. A contour plot is a way to visualize a three-dimensional function into two dimensions. for each point on a line, the function returns the same value the darker the area is, the smaller the value of the function is. You can see the last point illustrated in the figure below



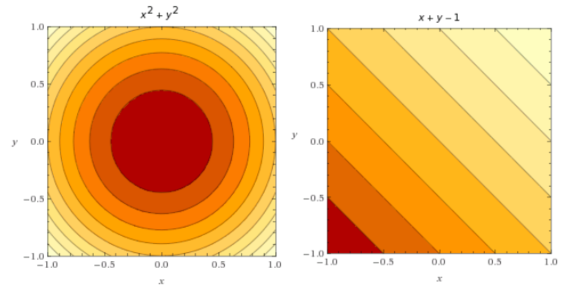
gradient of a function can be visualized as a vector field, with the arrow pointing in the direction where the function is increasing: they are perpendicular to a contour line they should point in the direction where the function is increasing.



Let’s Say we want to minimize,

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & f(x,y) = x^2 + y^2 \\ \text{subject to} & g_i(x,y) = x + y - 1 = 0 \end{array}$$

The objective function f(x,y) and the constraint function g(x,y) can be visualized as contour in the figure below:

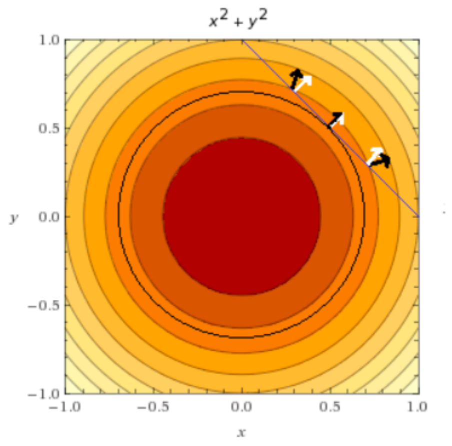


given constraint g(x,y) ,

x+y-1 = 0

y = 1 - x

draw some gradient vectors of the objective function (in black) and some gradient vectors of the constraint function (in white).



We can see that there is only one point where two vectors point in the same direction: it is the minimum of the objective function, under the constraint.

In order to find the minimum va
lue of the function, we need to find points where,

$\nabla f(x,y)=\lambda \nabla g(x,y)$

λ: is called lagrange multiplier

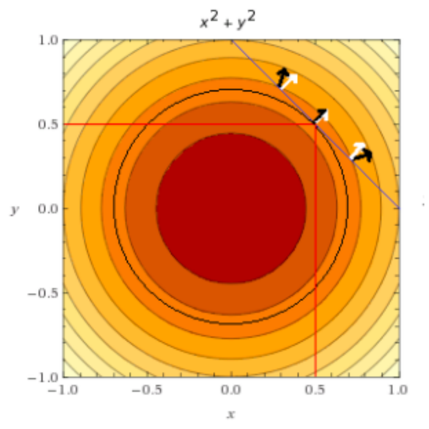
$\nabla f(x,y)-\lambda \nabla g(x,y)=0$ --------(1)
$L(x,y,\lambda)=f(x,y)-\lambda g(x,y)$ -------(2)
then its gradient is:
$\nabla L(x,y,\lambda)=\nabla f(x,y)-\lambda \nabla g(x,y)$ --(3)
This function L is called the Lagrangian, and solving for the gradient of the Lagrangian (solving $\nabla L(x,y,\lambda)=0$) means finding the points where the gradient of f and g are parallels.

Solving, previous optimization problem,

$$\begin{cases} \frac{\partial L}{\partial x} = 0 \\ \frac{\partial L}{\partial y} = 0 \\ \frac{\partial L}{\partial \lambda} = 0 \end{cases}$$

$$\begin{cases} 2x - \lambda = 0 \\ 2y - \lambda = 0 \\ -x - y + 1 = 0 \end{cases}$$

We have found a solution :
x=1/2, y=1/2 and λ=1



The minimum of f is at x=1/2 and y=1/2

Coming back to our SVM problem,

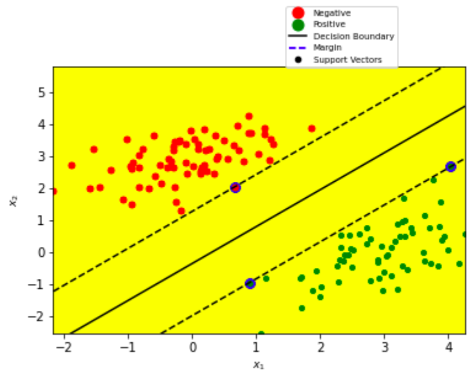
Lagrangian multipliers for inequalities is α_i and equalities h_i.The Lagrangian function is

$$L(x,\lambda,\alpha) = f(x) + \sum_i \lambda_i h_i(x) + \sum_i \alpha_i g_i(x)$$

KKT conditions are:

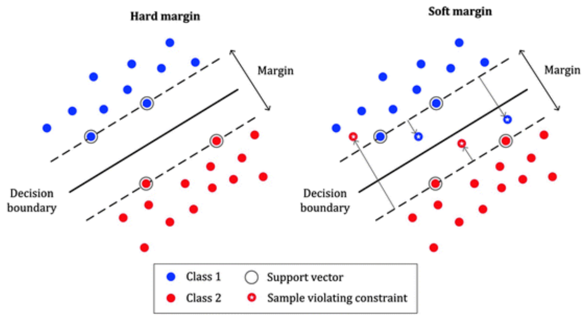
$$\Delta L(x^*,\lambda^*,\alpha^*) = 0$$
$$\alpha_i \geq 0$$
$$g_i(x^*) \leq 0$$
$$\alpha_i g_i(x^*) = 0$$

The **hard margin** is the oldest and simplest formulation of SVM. It assumes that the dataset is **linearly separable** by class. Let's see the case when the dataset is not linearly separable using **Soft margin** and **kernel.**



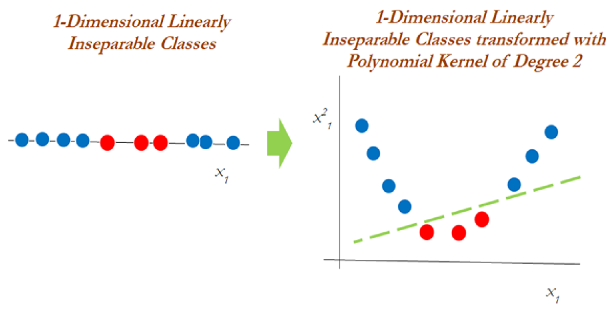
Soft Margin:

it tolerates a few dots to get misclassified.it tries to balance the trade-off between finding a line that maximizes the margin and minimizes the misclassification.



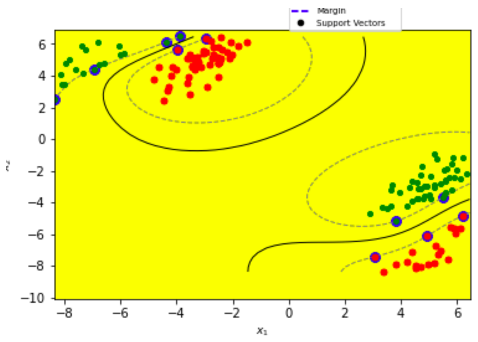
Kernel:

it utilizes existing features, applies some transformations, and creates new features. Those new features are the key for SVM to find the nonlinear decision boundary.


1-Dimensional Linearly Inseparable Classes
1-Dimensional Linearly Inseparable Classes transformed with Polynomial Kernel of Degree 2
x_1
x_1^2
x_1

Gaussian/RBF Kernel:

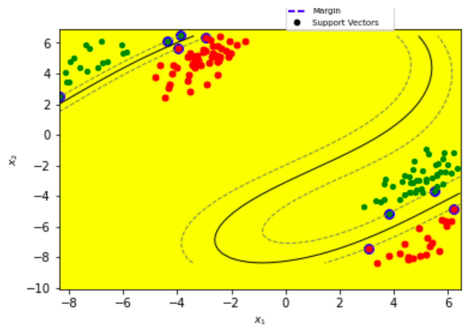
Gaussian kernel is simply a weighted linear combination of the kernel function computed between a data point and each of the support vectors.



```
def gaussian_kernel(self, x, y,
gamma=0.5):
        # Inputs:
        #    x    : input var
        #    y    : support vectors
        #    gamma    : param
        # K(x,xi) = exp(-gamma *
sum((x — xi²)).
return np.exp(-gamma*linalg.norm(x
- y) ** 2 )
```

Polynomial Kernel:

The polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that represents the similarity of vectors in a feature space over polynomials of the original variables, allowing learning of non-linear models.



```
def polynomial_kernel(self, x,
y,C=1, d=3):
        # Inputs:
        #    x    : input var
        #    y    : support vectors
        #    c    : param svm
        #    d    : degree of
polynomial.
        #    K(x,xi) = C + sum(x *
xi)^d
    return (np.dot(x, y) + C) ** d
```

SMO Algorithm:

Sequential minimal optimization (**SMO**) is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support-vector machines (SVM).
The SMO algorithm selects two α parameters, αi and αj and optimizes the objective value jointly for both these α's. Finally it adjusts the b parameter based on the new α's. This process is repeated until the α's converge.

1. Selecting α parameters:

Step1: iterate over all αi , i = 1, . . . m. If αi does not fulfill the KKT conditions within some numerical tolerance, we select αj at random from the remaining m − 1 α's and attempt to jointly optimize αi and αj.
Step2: If none of the α's are changed after a few iterations over all the αi 's, then the algorithm terminates.

2. Optimizing αi and αj

Step1: find bounds L and H such that L ≤ αj ≤ H must hold in order for αj to satisfy the constraint that 0 ≤ αj ≤ C. It can be shown that these are given by :

- If $y^{(i)} \neq y^{(j)}$, $\quad L = \max(0, \alpha_j - \alpha_i), \quad H = \min(C, C + \alpha_j - \alpha_i)$
- If $y^{(i)} = y^{(j)}$, $\quad L = \max(0, \alpha_i + \alpha_j - C), \quad H = \min(C, \alpha_i + \alpha_j)$

optimal αj is given by:

$$\alpha_j := \alpha_j - \frac{y^{(j)}(E_i - E_j)}{\eta}$$

$$\begin{aligned} E_k &= f(x^{(k)}) - y^{(k)} \\ \eta &= 2\langle x^{(i)}, x^{(j)} \rangle - \langle x^{(i)}, x^{(i)} \rangle - \langle x^{(j)}, x^{(j)} \rangle. \end{aligned}$$

Step2: Next we clip αj to lie within the range [L, H]

$$\alpha_j := \begin{cases} H & \text{if } \alpha_j > H \\ \alpha_j & \text{if } L \leq \alpha_j \leq H \\ L & \text{if } \alpha_j < L. \end{cases}$$

Step3: find the value for αi . This is given by

$$\alpha_i := \alpha_i + y^{(i)} y^{(j)} (\alpha_j^{(\text{old})} - \alpha_j)$$

3. Calculate threshold b:

After optimizing αi and αj , we select the threshold b such that the KKT conditions are satisfied for the ith and jth examples. If, after optimization, αi is not at the bounds (i.e., 0 < αi < C), then the following threshold b1 is valid, since it forces the SVM to output y (i) when the input is x (i):

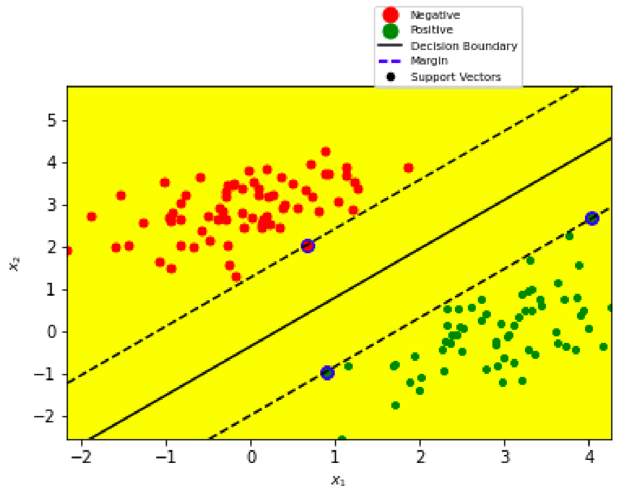
$$b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i^{(old)})\langle x^{(i)}, x^{(i)} \rangle - y^{(j)}(\alpha_j - \alpha_j^{(old)})\langle x^{(i)}, x^{(j)} \rangle$$

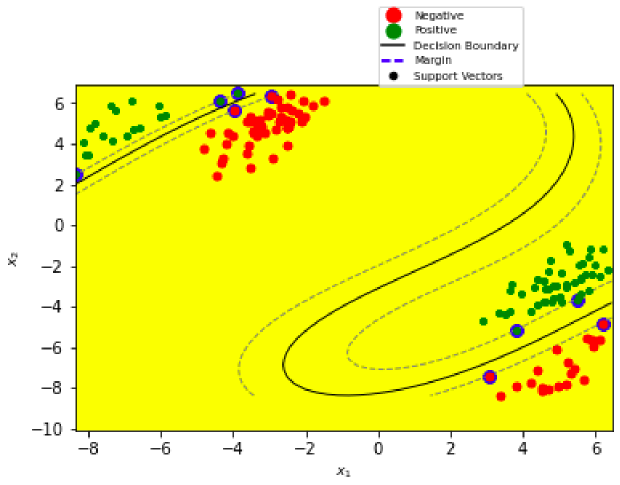
$$b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i^{(old)})\langle x^{(i)}, x^{(j)} \rangle - y^{(j)}(\alpha_j - \alpha_j^{(old)})\langle x^{(j)}, x^{(j)} \rangle$$

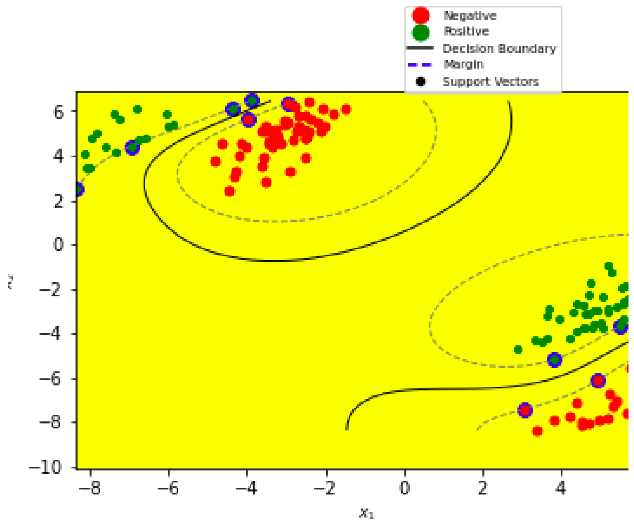
Now computing the threshold b:

$$b = \begin{cases} \mathrm{b}_1, & \text{if } 0 < \alpha_i < C \\ \mathrm{b}_2 & \text{if } 0 < \alpha_j < C \\ (\mathrm{b}_1 + b_2)/2, & \text{otherwise} \end{cases} \qquad (9)$$

Results:


Negative
Positive
Decision Boundary
Margin
Support Vectors
x_1
x_2


Negative
Positive
Decision Boundary
Margin
Support Vectors
x_1
x_2


Negative
Positive
Decision Boundary
Margin
Support Vectors
x_1
x_2