SVM (Support Vector Machine):

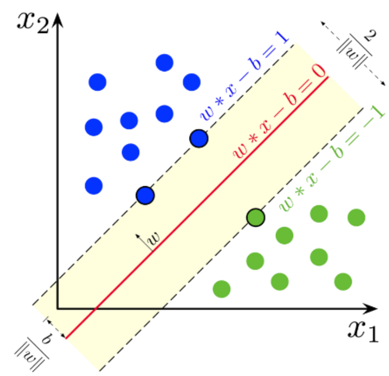
- used for classification

Goal:

- the goal is to **maximize the margin** between the two classes
- we basically minimize the **objective function** $\frac{1}{2}\|w\|^2$ using **QP** (**Q**uadratic **P**rograming) $\Longrightarrow$ we are maximizing the **margin** from the data point to the hyperplane given by $\frac{1}{\|w\|}$, while applying the **constraint** that each classified point must be ≥ 1

Input data:

i	x_{1i}	x_{2i}	label y_i
0	1	1	a
1	2	0.5	a
2	40	40	b
3	50	35	b



Constraint:

- $\forall i$ for all i
- w is the weight

$$\boldsymbol{y_i(w^T.x + b) \geq 1} \quad \forall i \tag{1}$$

- verify the constraint is true

$$\begin{gathered} y_i = 1: \\ 1(w^T.x + b) \geq 1 \;\; \rightarrow \;\; (w^T.x + b) \geq 1 \\ y_i = -1: \\ -1(w^T.x + b) \geq 1 \;\; \rightarrow \;\; (w^T.x + b) \leq -1 \end{gathered} \tag{2}$$

Primal problem - objective function:

$$min \frac{1}{2}\|w\|^2 \tag{3}$$

QP (Quadratic Programming)-equation:

- it is used to solve optimization problems where the objective function is quadratic, and the constraints (if any) are typically linear

$$\min \frac{1}{2} x^T . P . x + q^T . x \qquad (5)$$

- in the code we can skip (set to zero) the linear term $q^T . x$ because we try to find the maximum margin between classes without any offset or linear bias:

QP (Quadratic Programming)-code:

- relabel the values and ad bias to the matrix

```python
import numpy as np
from scipy.optimize import minimize

# Input data
X = np.array([[1, 1], [2, 0.5], [40, 40], [50, 35]])
y = np.array(['a', 'a', 'b', 'b'])
```
✓ 0.0s

```python
# Re-label data to {-1, 1} for SVM compatibility
y = np.where(y == 'a', -1, 1)
y
```
✓ 0.0s Open 'y' in Data Wrangler

```
array([-1, -1,  1,  1])
```

```python
# Add a bias term to the feature matrix (append a column of ones)
X_with_bias = np.hstack((X, np.ones((X.shape[0], 1))))
X_with_bias
```
✓ 0.0s Open 'X_with_bias' in Data Wrangler

```
array([[ 1. ,  1. ,  1. ],
       [ 2. ,  0.5,  1. ],
       [40. , 40. ,  1. ],
       [50. , 35. ,  1. ]])
```

- Define matrices P and q

```python
# Define identity matrix (P = I for quadratic term)
P = np.eye(X_with_bias.shape[1])
P
```
✓ 0.0s Open 'P' in Data Wrangler

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```python
# Define quadratic term-setting it into zero we neglect the impact (we are interested in maximazing the margin)
q = np.zeros(X_with_bias.shape[1])
q
```
✓ 0.0s Open 'q' in Data Wrangler

```
array([0., 0., 0.])
```

- **define objective function** and **constraint** with the initial guess and run the optimization

```
# Objective function: 1/2 * w^T * P * w + q^T * w
def objective(w):
    return 0.5 * np.dot(w.T, np.dot(P, w)) + np.dot(q.T, w)

# Constraints: y_i * (w^T x_i + b) - 1 >= 0 for each data point
def constraints(w):
    return y * (np.dot(X_with_bias, w)) - 1
✓ 0.0s
```

```
# Initial guess for the variables: weights + bias
w0 = np.zeros(X_with_bias.shape[1])

# Solve the optimization problem using SLSQP
result = minimize(objective, w0, constraints={'type': 'ineq', 'fun': constraints}, method='SLSQP')
✓ 0.0s
```

```
# Print the result
if result.success:
    print("Optimal weights (w1, w2):", result.x[:-1])
    print("Optimal bias (b):", result.x[-1])
else:
    print("Optimization failed:", result.message)
✓ 0.0s
```

```
Optimal weights (w1, w2): [0.01709402 0.03418803]
Optimal bias (b): -1.0512820512820509
```

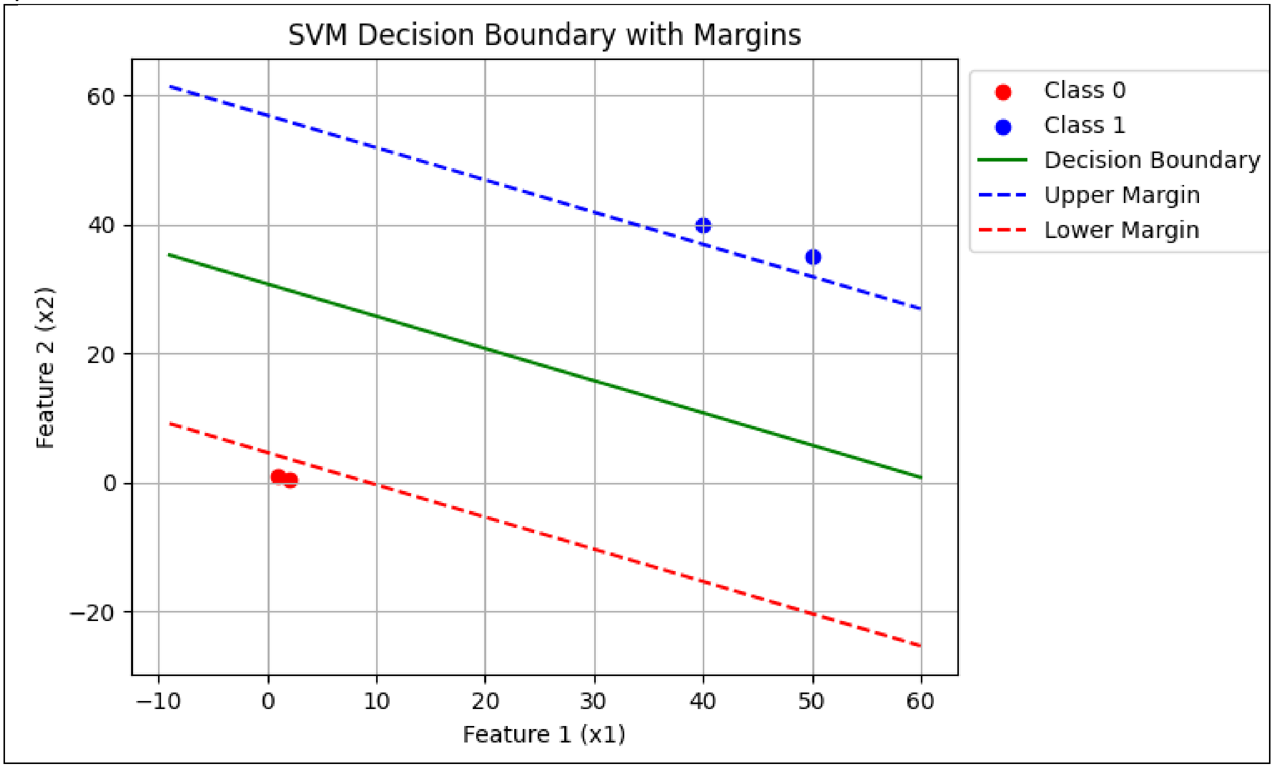
- predict the values and calculate the accuracy

$$y_i = sign(w^T.x + b) \quad (6)$$

```
# Predict function to evaluate the decision function on the training data
def predict(X, w, b):
    return np.sign(np.dot(X, w) + b)

# Extract the weights and bias from the result
optimal_weights = result.x[:-1]
optimal_bias = result.x[-1]

# Make predictions on the training data
predictions = predict(X, optimal_weights, optimal_bias)

# Print predictions and true labels
print("Predictions:", predictions)
print("True Labels:", y)

# Calculate and print accuracy
accuracy = np.mean(predictions == y)
print("Accuracy:", accuracy)
✓ 0.0s
```

```
Predictions: [-1. -1.  1.  1.]
True Labels: [-1 -1  1  1]
Accuracy: 1.0
```

-print results



- test model for new data

```
# Make predictions on the training data
new_point1 = np.array([0,20])
predictions = predict(new_point1, optimal_weights, optimal_bias)
predictions
```

✓ 0.0s

```
-1.0
```

```
# Make predictions on the training data
new_point2 = np.array([0,40])
predictions = predict(new_point2, optimal_weights, optimal_bias)
predictions
```

✓ 0.0s

```
1.0
```

Dual Problem

Lagrangian for the primal problem:

$$L(w, b, \lambda) = objective\ function - constraint$$

$$L(w, b, \lambda) = min\frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\lambda_i[y_i(w^T.x + b) - 1]$$

$$\frac{\partial L}{\partial w} = 0\ \rightarrow w \tag{4}$$

$$\frac{\partial L}{\partial b} = 0\ \rightarrow b$$

Dual objective function:

$$max_{(\lambda)}\left(\sum_{i=1}^{n}\lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j\left(x_i^T x_j\right)\right) \tag{4}$$