# Support Vector Machines

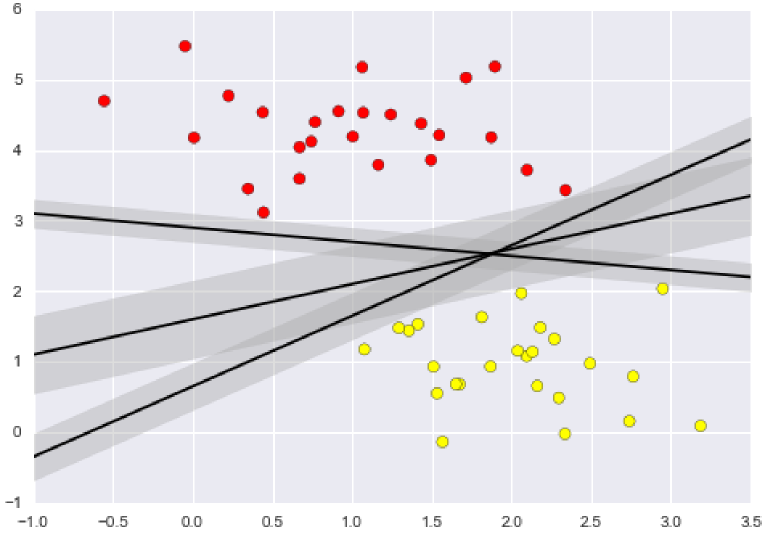
Injung Kim
Handong Global University

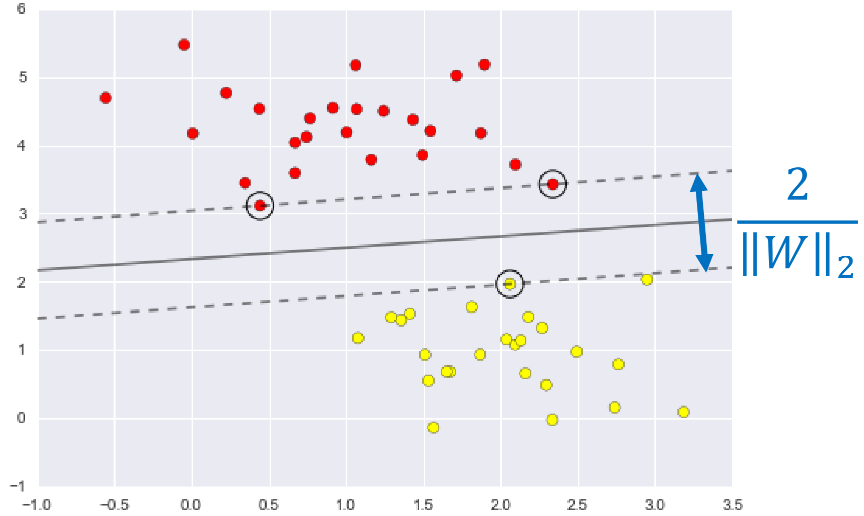
# Agenda

- Linear SVM
- Kernel SVM
- SVM in scikit-learn

# Support Vector Machines

- Search for the boundary that separates classes with maximum margin
  - Linear SVM
  - C-SVM (SVM with soft margin)
  - Nonlinear SVM (SVM with kernel)



# Linear SVM

- Search for $W^* = argmin_W |W|_2^2$ s.t.
  - $WX_i + b \geq +1$ for $y_i = +1$
  - $WX_i + b \leq -1$ for $y_i = -1$
  - ➔ $\boldsymbol{y_i(WX_i + b) - 1 \geq 0}$ combines the two conditions

- Loss function

$$L_p = \frac{1}{2}|W|_2^2 - \sum_{i=1}^{l} \alpha_i y_i (WX_i + b) + \sum_{i=1}^{l} \alpha_i$$

  - $\alpha_i \geq 0$'s are Lagrange multipliers

- Dual formulation
  - Minimizing $L_P$ subject to $\frac{\partial L_p}{\partial \alpha_i} = 0,\ \alpha_i \geq 0$
  - Maximizing $L_P$ subject to $\frac{\partial L_p}{\partial W} = 0,\ \frac{\partial L_p}{\partial b} = 0,\ \alpha_i \geq 0$

# Lagrange Multiplier

- A strategy for finding the local maxima and minima of a function subject to equality constraints
  - Maximize $f(X) = f(x_1, \dots, x_n)$
  - Subject to $g_j(X) = g_j(x_1, \dots, x_n) = 0,\ 1 \le j \le m$

  * Assume $f(\cdot)$ and $g_j(\cdot)$ are continuous and have 1st partial derivatives

- Lagrange function
  - $L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_m) = f(x_1, \dots, x_n) - \sum_j \lambda_j g_j(x_1, \dots, x_n)$
    - $\lambda_i$'s are called Lagrange multiplier

- We can find local optimum $X^*$ by jointly optimize $L(\cdot)$ w.r.t. $X$ and all $\lambda_j$'s (assume $g_j'(\cdot) \neq 0$)

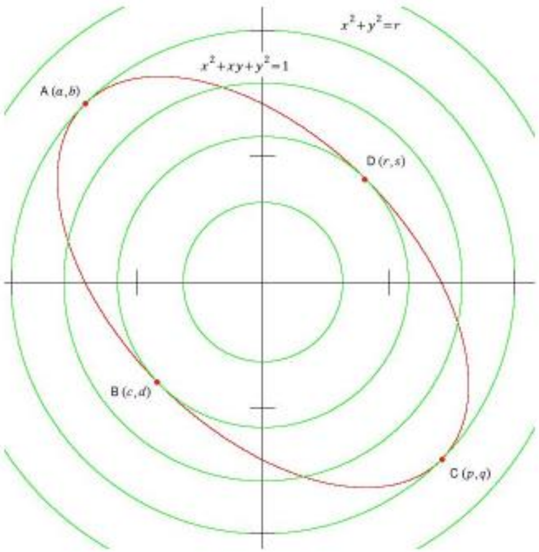
# Lagrange Multiplier

Example)

- Objective: $f(x,y) = x^2 + y^2$
- Subject to $g(x,y) = x^2 + xy + y^2 = 0$

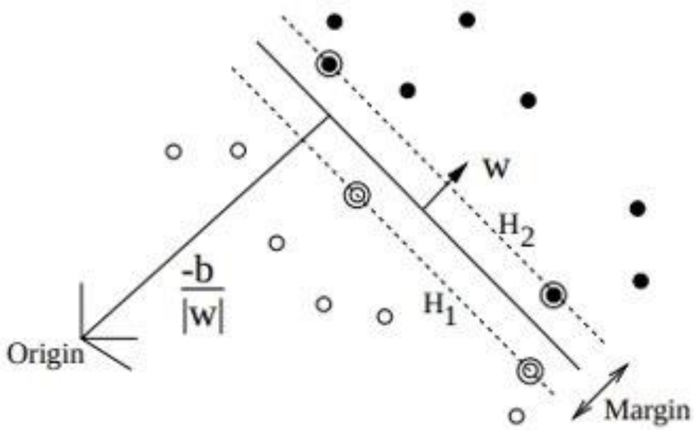
- The optimum solution is found at the tangent points of the two curves.

- $\nabla f(x,y)$ and $\nabla g(x,y)$ are parallel
- $\nabla f(x,y) = \lambda \nabla g(x,y)$



# Linear SVM

- Solution from $\frac{\partial L_p}{\partial W} = 0, \frac{\partial L_p}{\partial b} = 0$
  - $W = \sum_{i=1}^{l} \alpha_i y_i X_i$
  - $\sum_{i=1}^{l} \alpha_i y_i = 0$



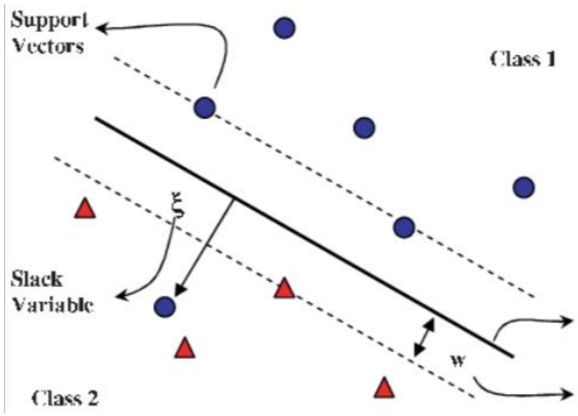

- Substitute $\frac{\partial L_p}{\partial W} = 0, \frac{\partial L_p}{\partial b} = 0$ into $L_p$

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j X_i \cdot X_j$$

  - Find $\alpha_i$'s that maximize $L_D$.
    - $X_i$'s with $\alpha_i > 0$ are called support vectors
  - Discriminant function: $WX_j + b = \sum_{i=1}^{l} \alpha_i y_i X_i \cdot X_j + b$

# Linear SVM with Soft Margin (C-SVM)

- Allows soft margin for non-separable tasks
  - $W^* = argmin_W \left\{ \left[ \frac{|W|^2}{2} + C(\sum_i \xi_i) \right] \right\}$ s.t.
    - $WX_i + b \geq +1 - \xi_i$ for $y_i = +1$
    - $WX_i + b \leq -1 + \xi_i$ for $y_i = -1$
  - Equivalent to
    - $y_i(X_i W + b) - 1 + \xi_i \geq 0$




- Primal formulation

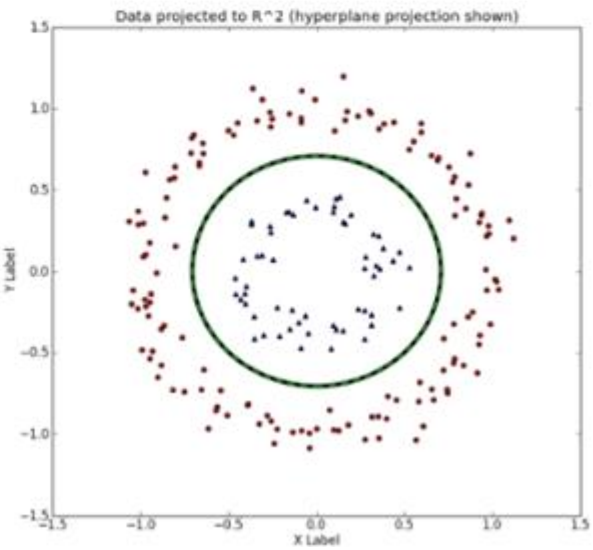
$$L_p = \frac{1}{2}|W|^2 + C\sum_{C}^{l} \xi_i - \sum_{\{i=1\}}^{l} \alpha_i \{y_i (X_i W + b) - 1 + \xi_i\} - \sum_{i} \mu_i \xi_i$$

# Agenda

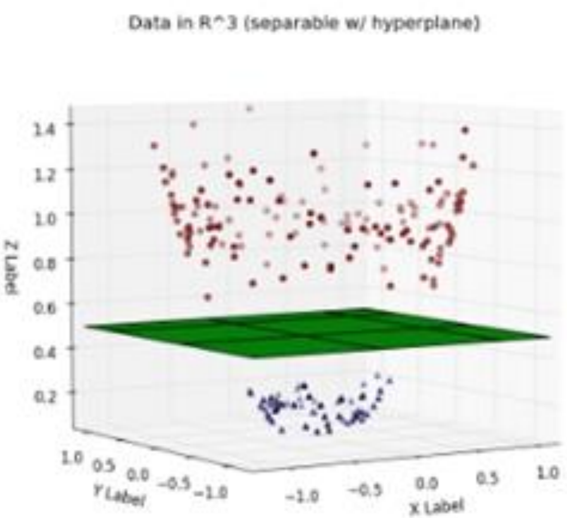
- Linear SVM
- <u>Kernel SVM</u>
- SVM in scikit-learn

# SVM with Kernel

- Nonlinear SVM
  - Transforms input feature to higher dimensional space using nonlinear kernel functions
  - Samples are better separated in higher-dimensional space




$(x, y)$




$(x, y, x^2 + y^2)$

# Nonlinear SVM

- Transform $X$ to high dimension space through a nonlinear transform $\phi(\cdot)$

| | Loss function | Discriminant function |
|---|---|---|
| Linear SVM | $L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j X_i \cdot Xj$ | $WX + b$ <br> $= \sum_{i=1}^{l} \alpha_i y_i X_i \cdot X + b$ |
| Nonlinear SVM | $L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{\phi}(\boldsymbol{X}_i)\boldsymbol{\phi}(\boldsymbol{X}_j)$ | $W\phi(X) + b$ <br> $= \sum_i \alpha_i y_i \boldsymbol{\phi}(\boldsymbol{X}_i)\boldsymbol{\phi}(\boldsymbol{X}) + b$ |

  - It is hard to find appropriate high-dimensional transform $\phi(\cdot)$

# Nonlinear SVM – Kernel Trick

- Replace $\phi(X_i)\phi(X_j)$ by $K(X_i, X_j)$
  - We don't need to know $\phi(\cdot)$
  - $K(X_i, X_j)$ eliminates computation in high-dimensional space.

| | Loss function | Discriminant function |
|---|---|---|
| Linear SVM | $L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j X_i \cdot Xj$ | $WX + b$ <br> $= \sum_{i=1}^{l} \alpha_i y_i X_i \cdot X + b$ |
| Nonlinear SVM | $L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \boldsymbol{\phi(X_i)\phi(X_j)}$ | $W\phi(X) + b$ <br> $= \sum_i \alpha_i y_i \boldsymbol{\phi(X_i)\phi(X)} + b$ |
| Nonlinear SVM (kernel) | $L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \boldsymbol{K(X_i, X_j)}$ | $W\phi(X) + b$ <br> $= \sum_i \alpha_i y_i \boldsymbol{K(X_i, X)} + b$ |

# Popular Kernels for SVM

- Polynomial kernel

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

- Gaussian kernel

$$k(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right)$$

- RBF kernel

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma ||\mathbf{x_i} - \mathbf{x_j}||^2)$$

# SVM in scikit-learn

- SVM classifiers in scikit-learn
  - SVC
  - NuSVC ($\nu - SVC$)
  - LinearSVC

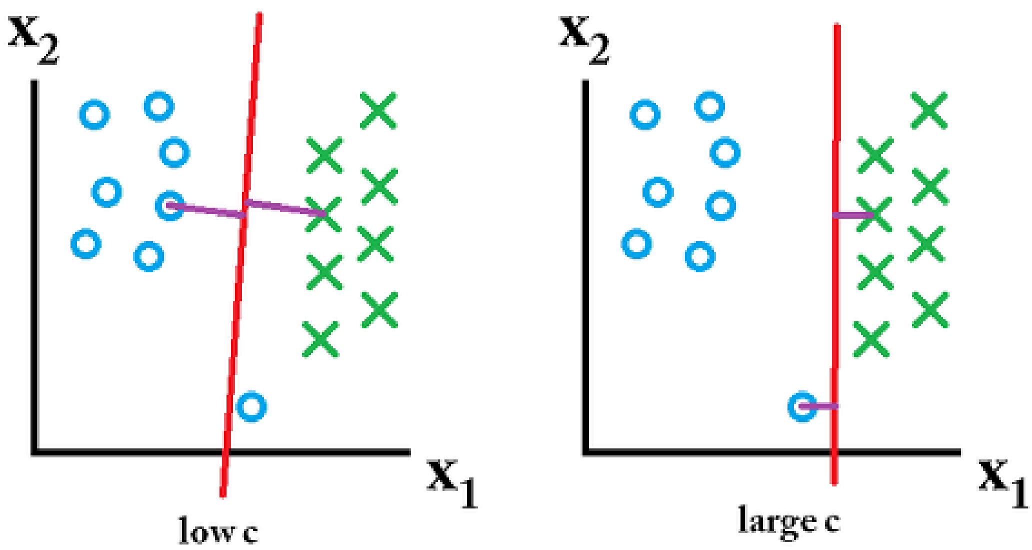
  - Example

```
from sklearn import svm                # import
X = [[0, 0], [1, 1]]                   # input data
y = [0, 1]                             # target label
clf = svm.SVC(gamma='scale')           # create SVC
clf.fit(X, y)                          # train
clf.predict([[2., 2.]])                # predict
```

- Reference
  - https://scikit-learn.org/stable/modules/svm.html

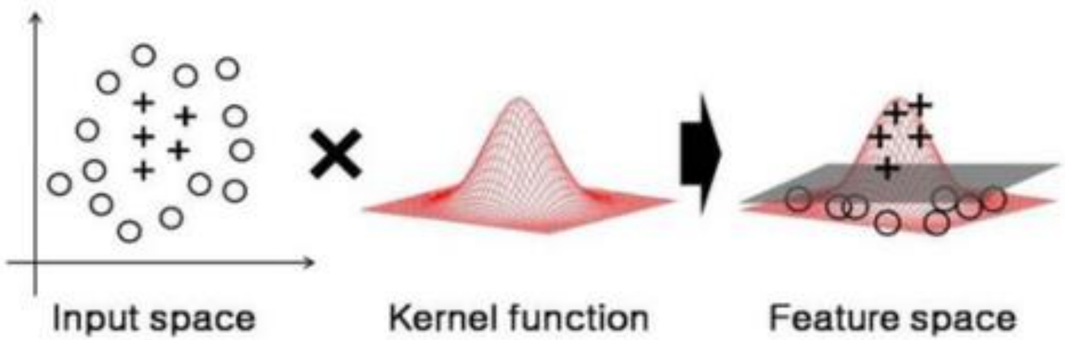
# SVM Hyper-Parameters

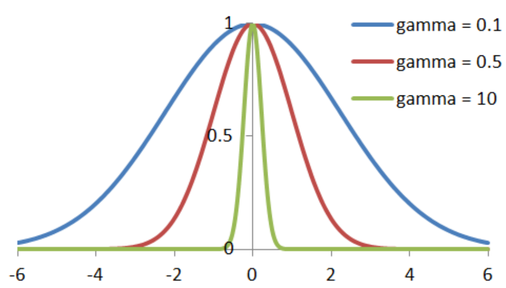
- Penalty parameter C of the error term (default: 1)
  - Too small C: underfitting
  - Too high C: overfitting



# SVM Hyper-Parameters

- Kernel coefficient gamma for 'rbf', 'poly' and 'sigmoid'.
  - Too small: underfitting
  - Too large: overfitting

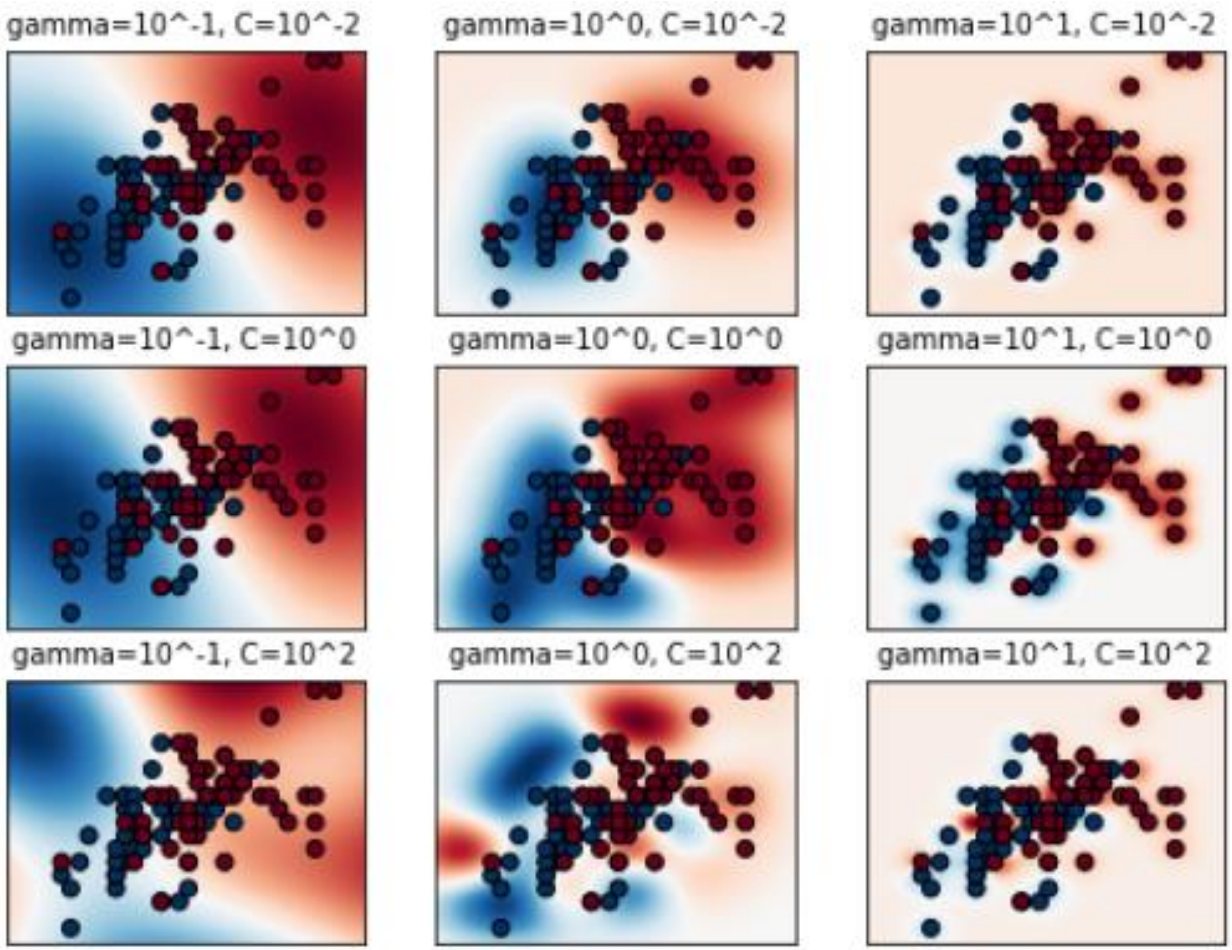



scikit-learning **gamma** options

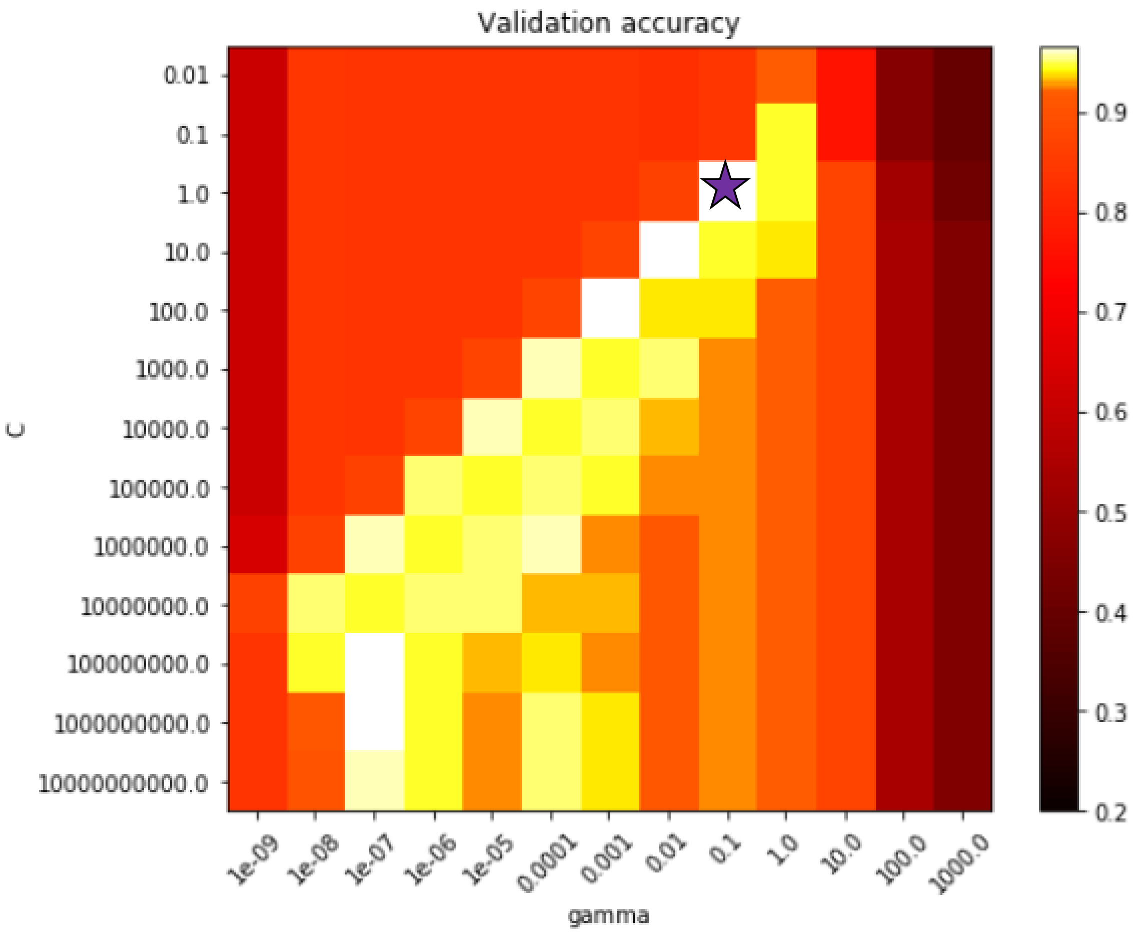
- **'auto'**: 1/n_features
- **'scale'**: 1/(n_feature * X.var())
  X.var(): variance of X

# SVM Hyper-Parameters



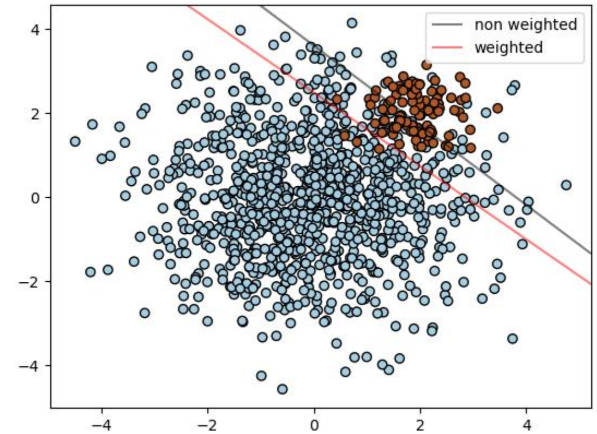
# SVM Hyper-Parameters

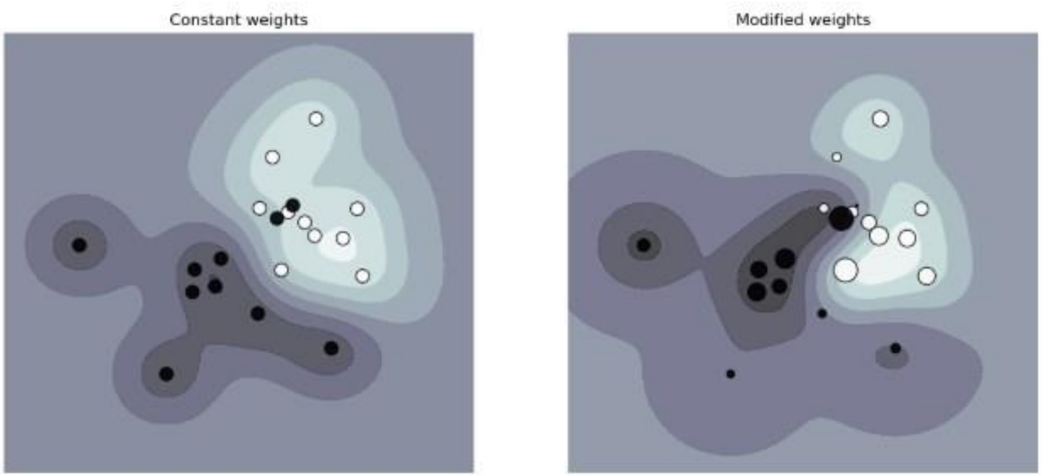
- C – gamma grid



# Handling Unbalanced Datasets

- Assign heavier weights to the class with less sample



# References

- Handing unbalanced data
  - https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
  - https://shiring.github.io/machine_learning/2017/04/02/unbalanced
  - https://www.datascience.com/blog/imbalanced-data
- Support Vector Regression
  - https://www.saedsayad.com/support_vector_machine_reg.htm

# Q&A

## Thank you for your attention!