Classification

Injung Kim
Handong Global University

Agenda

- Bayesian Theorem
- k-Nearest Neighbor
- Linear Classifiers
- Support Vector Machines
- Q&A

Regression

- Regression: estimating the relationships among variables.
  - Techniques for modeling and analyzing several variables focusing on the relationship between a dependent variable and independent variables (or 'predictors').

  Ex) Predicting variables from other variables

  (bankrupt prediction, sales prediction, object coordinate estimation, …)

Classification

- The act of taking in raw data and taking an action based on the category of the data.
  - Input: a data (event, object, observation, …) that belongs to one of predefined categories
  - Output: category of the input event

Classification

- Classes (hidden)

$$\omega_i \in \{\omega_1, \omega_2, \dots, \omega_n\}$$

- Input
  - Observation, feature vector (or sequence)

$$X = (x_1, x_2, \dots, x_d)$$

- Goal: Finding ω_i to which X belongs
  - Without any information: $\omega^* = argmax_{\omega} P(\omega)$
    - $P(\omega)$ is called prior probability
  - Given an observation X : $\omega^* = argmax_{\omega} P(\omega|X)$
    - $P(\omega|X)$ is called posterior probability

Bayesian Classifier

- How to find the class that is likely to have generated the input pattern?
  - Select ω_i that maximizes $P(\omega_i|X)$
    - ω_i: class or hidden var, X: observation
  - Theoretically optimum classifier

- However, it's difficult to know $P(\omega_i|X)$ for each class

➔ Bayes' theorem

posterior prob. → $P(\omega|X) = \frac{P(\omega \cap X)}{P(X)}$

likelihood → $= \boxed{\frac{P(X|\omega)P(\omega)}{P(X)}}$ ← prior prob. ← evidence

Bayesian Classifier

- A classifiers that selects ω^* such that

$$\omega^* = \arg\max_{\omega} P(\omega \mid X)$$

posterior prob.

$$= \arg\max_{\omega} \frac{P(X \mid \omega)P(\omega)}{P(X)}$$

P(X) is independent from ω

$$= \arg\max_{\omega} \underbrace{P(X \mid \omega)}_{\text{likelihood}} \underbrace{P(\omega)}_{\text{prior prob.}}$$

likelihood

prior prob.

Bayesian Classifier

- Prior probability $P(\omega)$
  - Measure the frequencies of classes
    - Estimate the distributions of classes by models
  - Assume $P(\omega)$'s are same for all ω's. ➔ ignore
  - Heuristically designed constraints

    Ex) Smoothness assumption

- Likelihood $P(X|\omega)$
  - Estimated by a generative model (e.g. Gaussian)

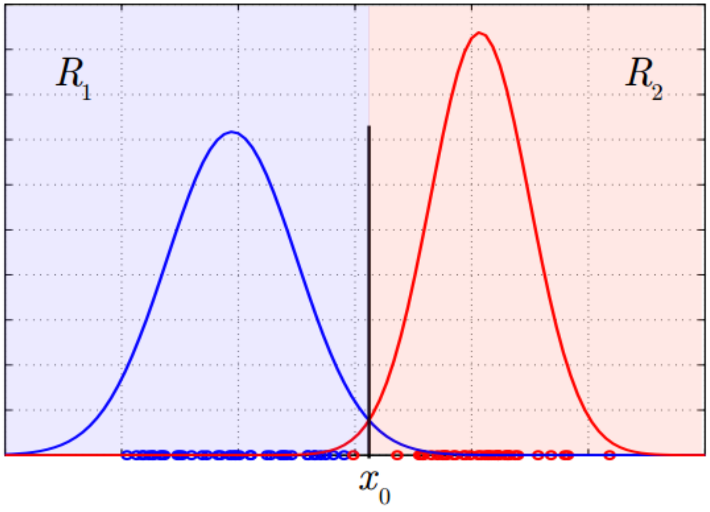
Cf. Discriminative models estimate $P(\omega|X)$ directly

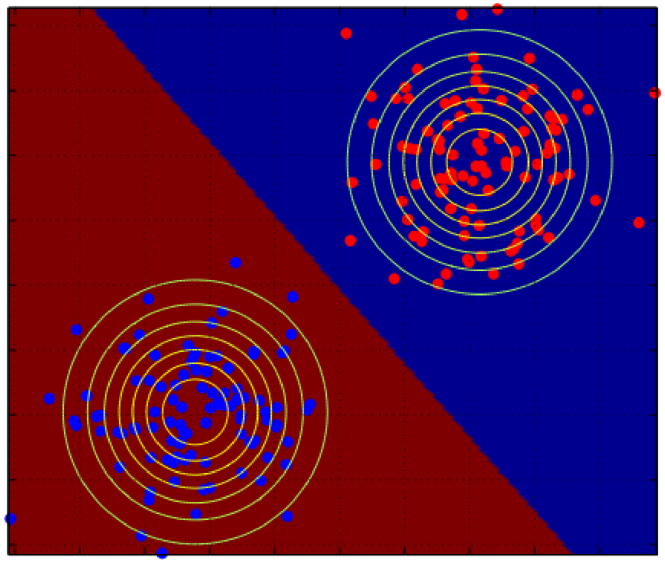
Gaussian Model

- Assume each class has Gaussian distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$P(X \mid \omega) = \frac{1}{(2\pi)^{d/2} \left|\Sigma_\omega\right|^{1/2}} \exp\left[-\frac{(X-\mu_\omega)^T \Sigma_\omega^{-1} (X-\mu_\omega)}{2} \right]$$





Discriminant Function

- Discriminant function of a class ω
  - Classification rule: select ω whose $g_\omega(X)$ is the maximum

$$g_\omega(X) \equiv \ln P(X \mid \omega) P(\omega)$$

$$= -\frac{\boxed{(X-\mu_\omega)^T \Sigma_\omega^{-1} (X-\mu_\omega)}}{2} - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln\left|\Sigma_\omega\right| + \ln P(\omega)$$

  - Usually, $P(\ )$ are assumed to be same

- Three cases
  - Case 1: $\Sigma_\omega = \sigma^2 I$ → linear classifier
  - Case 2: Σ_ω's are same for all classes → linear classifier
  - Case 3: each class has an arbitrary Σ_ω → quadratic classifier

Linear Discriminant Function

- Case 1: $\Sigma_\omega = \sigma^2 I$

$$g_\omega(X) = -\frac{(X-\mu_\omega)^T \Sigma_\omega^{-1}(X-\mu_\omega)}{2} \underbrace{-\frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_\omega|}_{\text{same for all classes}} + \ln P(\omega)$$

$$g_\omega(X) = -\frac{(X-\mu_\omega)^2}{2\sigma} + \ln P(\omega)$$

$$= -\frac{1}{2\sigma^2}\left[X^T X - 2\mu_\omega^T X + \mu_\omega^T \mu_\omega\right] + \ln P(\omega)$$

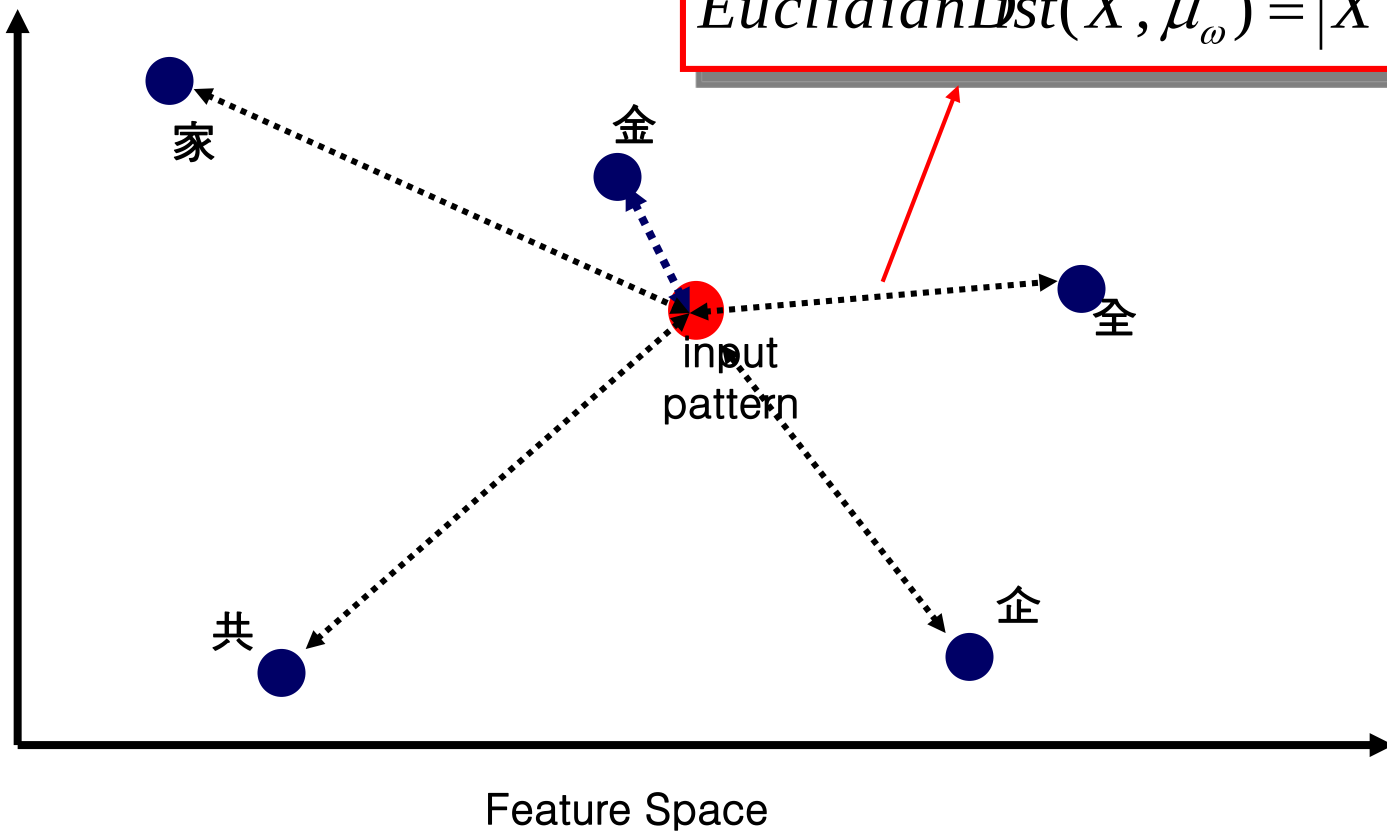
$$g_\omega(X) = \frac{1}{\sigma^2}\mu_\omega^T X - \frac{1}{2\sigma^2}\mu_\omega^T \mu_\omega + \ln P(\omega)$$

Linear Discriminant Function

LDF with Euclidian Distance

- Select the class with the nearest mean from the input pattern

$$EuclidianDist(X, \mu_\omega) = |X - \mu_\omega|^2$$



Linear Discriminant Function

- Case 2: Σ_ω's are same for all classes

$$g_\omega(X) = -\boxed{\frac{(X-\mu_\omega)^T \Sigma_\omega^{-1}(X-\mu_\omega)}{2}} \underbrace{- \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_\omega|}_{\text{same for all classes}} + \ln P(\omega)$$

$$g_\omega(X) = -\frac{(X-\mu_\omega)^T \Sigma^{-1}(X-\mu_\omega)}{2} + \ln P(\omega)$$

$$g_\omega(X) = \Sigma^{-1}\mu_\omega X - \frac{1}{2}\mu_\omega^T \Sigma^{-1}\mu_\omega + \ln P(\omega)$$

Linear Discriminant Function

* $(X-\mu_\omega)^T\Sigma^{-1}(X-\mu\omega)$ is called Mahalanobis Distance

Quadratic Discriminant Function

- Case 3: each class has an arbitrary Σ_ω

$$g_\omega(X) = -\frac{(X-\mu_\omega)^T \Sigma_\omega^{-1}(X-\mu_\omega)}{2} - \underline{\frac{d}{2}\ln 2\pi} - \frac{1}{2}\ln|\Sigma_\omega| + \ln P(\omega)$$

same for all classes

$$g_\omega(X) = -\frac{X \Sigma_\omega^{-1} X}{2} + \Sigma_\omega^{-1} X - \frac{1}{2}\mu_\omega^T \Sigma_\omega^{-1} \mu_\omega - \frac{1}{2}\ln|\Sigma_\omega| + \ln P(\omega)$$

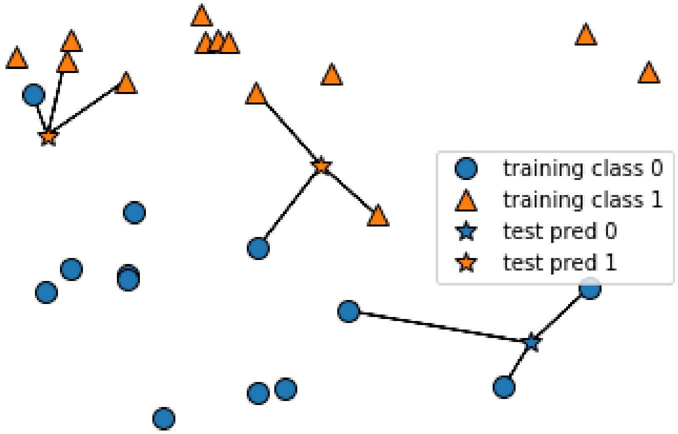
Quadratic Discriminant Function

Agenda

- Bayesian Theorem
- <u>k-Nearest Neighbor</u>
- Linear Classifiers
- Support Vector Machines
- Q&A

k-Nearest Neighbor Algorithm

- A non-parametric method used for classification and regression.
  - The input consists of the ***k* closest training examples** in the feature space.
  - **Classification**: the input object is classified by a plurality vote of its neighbors
  - **Regression**: the output is the average of the values of k nearest neighbors.

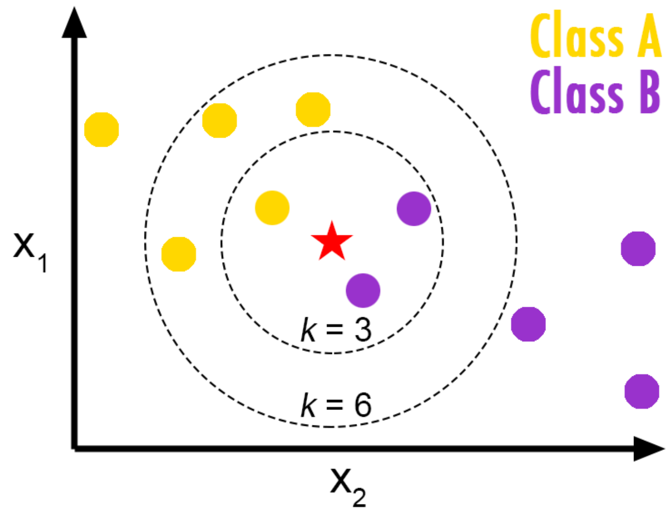


K–NN Classifier

- Approximate $P(\omega_j|X)$ by distribution of classes around X

$$P(\omega_j|X) \approx \frac{N_j}{\sum_j N_j} = \frac{N_j}{k}$$

  - ω_j: classes, N_j: # of class–j samples among neighbors
  - k controls the size of neighborhood (typically, 1, 3 or 5)



k-Nearest Neighbor Algorithm

- Upper-bound of error rate

$$R^* \le R_{k\mathrm{NN}} \le R^* \left(2 - \frac{MR^*}{M-1}\right)$$

  - R^*: Bayesian error rate (optimum)
  - M: # of classes

- Limitations
  - Relies on distance metric
  - Not applicable to a large volume of data

- Combined with metric learning
  - Deep learning + k-NN ➔ one-shot learning

K-NN using scikit-learn

- Import packages

```
import numpy as np
import sklearn as sk
```

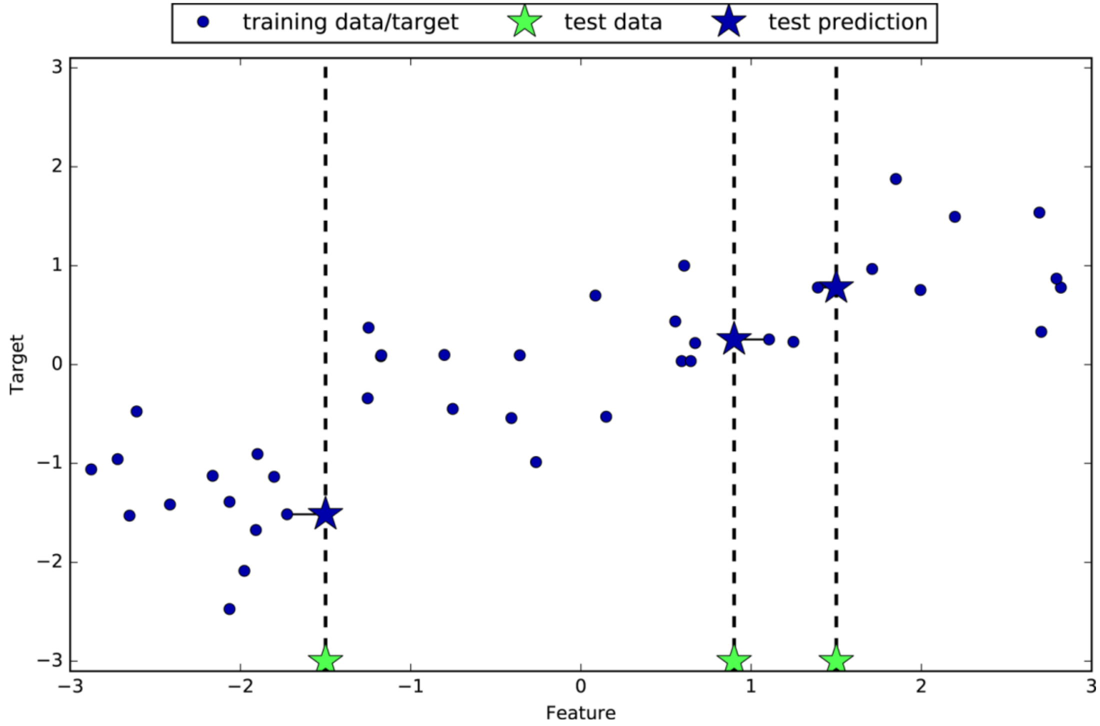
- Load dataset

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
iris_dataset.data., iris_dataset.target, random_state=0)
```

k-Nearest Neighbor Regression

- k-NN regression
  - Find k-nearest neighbors
  - Prediction = average value of the neighbors



K-NN using scikit-learn

- Create and train kNN classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
```

- Apply to new samples

```
X_new = np.array([[5,2.9,1,0.2]])    # 2D array
prediction = knn.predict(X_new)
print("prediction = {} ({})".format(prediction.squeeze(),
                iris_dataset.target_names[prediction].squeeze()))
```

- Evaluate
  - print("Accuracy = {}".format(knn.score(X_test, y_test)))

Numpy Functions: reshape()

- Creating a 1D array
  - a = np.arange(0, 20)
  - print("a.shape = ", a.shape)

    a.shape = (20,)
  - print(a)

    [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

- reshape()
  - b = np.reshape(a, (4, 5))
  - print("b.shape = ", b.shape)

    b.shape = (4, 5)
  - print(b)

    [[0 1 2 3 4] [5 6 7 8 9] [10 11 12 13 14] [15 16 17 18 19]]

Numpy Functions: expand_dims()

- expand_dims()
  - parameter axis specifies the new axis

  - # the new dimension becomes axis 0
  - c = np.expand_dims(a, axis = 0)
  - print("c.shape = ", c.shape)
    c.shape = (1, 20)
  - print(c)
    [[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]]

  - # the new dimension becomes axis 1
  - d = np.expand_dims(a, axis = 1)
  - print("d.shape = ", d.shape)
    d.shape = (20, 1)
  - print(d)
    [[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]
    [14] [15] [16] [17] [18] [19]]

Numpy Functions: squeeze()

- squeeze() removes single dimensional entries
  - print("c.shape = ", c.shape)
    c.shape = (1, 20)
  - f = np.squeeze(c)
  - print("f.shape = ", f.shape)
    f.shape = (20,)
  - print(f)
    [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

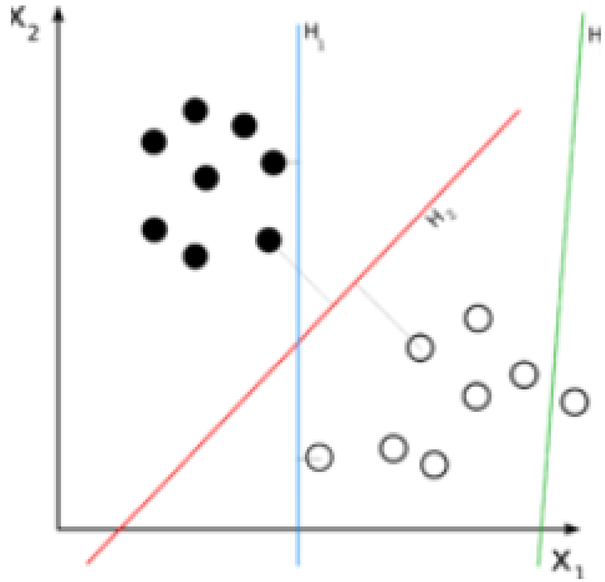
  - print("d.shape = ", d.shape)
    d.shape = (20, 1)
  - g = np.squeeze(d)
  - print("g.shape = ", g.shape)
    g.shape = (20,)
  - print(g)
    [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

Agenda

- Bayesian Theorem
- k-Nearest Neighbor
- <u>Linear Classifiers</u>
- Support Vector Machines
- Q&A

Linear Classifier

- Binary classification
  - class +1 if $f(x) = \sum_i w_i x_i + b = w_1 x_1 + \cdots + w_n x_n + b > 0$
  - class −1 if $f(x) = \sum_i w_i x_i + b = w_1 x_1 + \cdots + w_n x_n + b < 0$
    - n: feature dim.
    - $W = (w_i | 1 \le i \le n)$: model parameters
  - Class boundary ($f(x) = \sum_i w_i x_i + b = 0$) is hyperplane

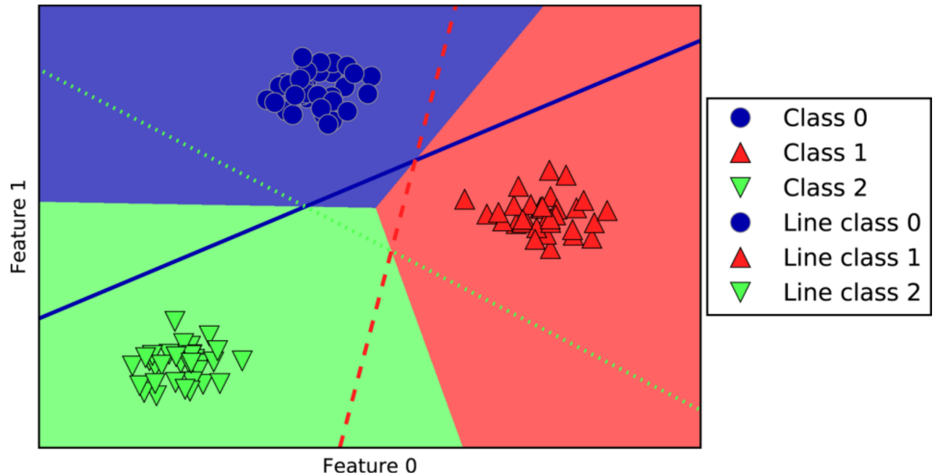


Linear Classifier

- Multi-class classification
  - Each class j has a discriminant function $f_j(x)$

$$\hat{y} = argmax_j[f_j(x)] = argmax_j\left[\sum_i w_{ij}x_i + b_j\right]$$

    - N: # of features, C: # of classes
    - $W = (w_{ij}|1 \le i \le N, 1 \le j \le C)$: model parameters
  - Class boundaries $(f_j(x) = f_k(x))$ are hyperplanes.

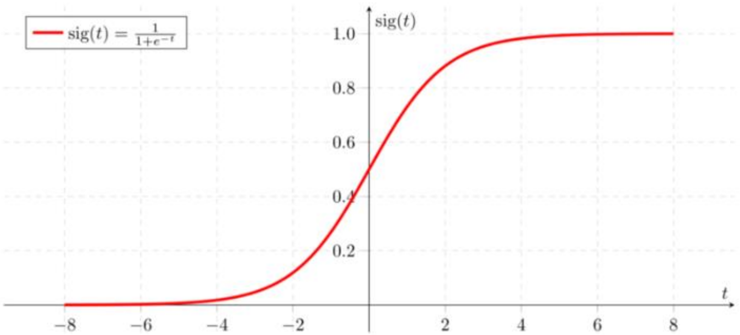


Parameter Estimation

- Logistic regression
- Linear Discriminant Function
- Linear SVM

Logistic Regression

- Logistic regression = Linear regression + logisistic function
  - $Logistic(x) = Sigmoid(x) = \frac{1}{1+\exp(-x)}$
- Discrimination function
  - $f_j(x) = logistic(\sum_i w_{ji}x_i + b_j)$
    - $f_j(x)$ has range [0,1]



- Parameter estimation
  - $W^* = argmin_W L(\hat{Y}, Y; W)$
  - $L(\cdot)$: a loss function such as cross entropy or MSE

Loss Functions

- Mean squared error

$$E_{MSE} = \frac{1}{2}\frac{\sum_N (\hat{y}_t - y_t)^2}{N}$$

- Cross entropy (with softmax activation)
  - Softmax activation: $\hat{y}_t = \frac{\exp(net_t)}{\sum_t \exp(net_t)}$

$$E_{CE} = -\sum_N y_t \log(\hat{y}_t)$$

    - $y_i \in \{0,1\}$: label (only $y_{true} = 1$)

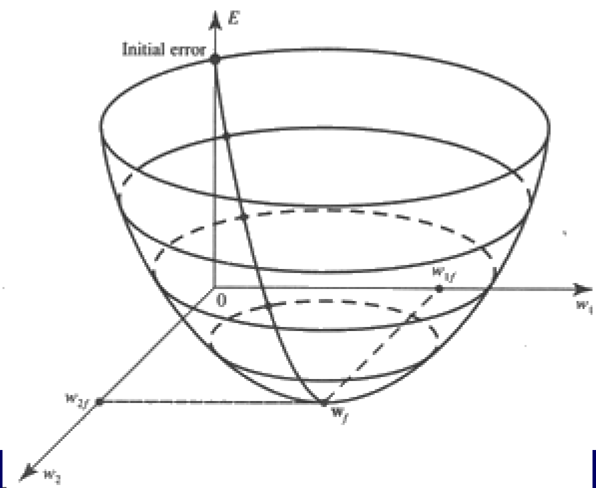
Gradient-based Learning

- Given current weights *W*, the gradient gives a direction in which increases the error most rapidly
  - Gradient of E(W) with respect to weight

$$\frac{\nabla \boldsymbol{E}(\boldsymbol{W})}{\nabla \boldsymbol{W}} = \left(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}_1}, \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}_2}, \dots, \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}_i}, \dots, \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}_M}\right)$$

  - Update rule

$$W^{t+1} = W^t - \eta \cdot \frac{\nabla \boldsymbol{E}(\boldsymbol{W})}{\nabla \boldsymbol{W}^t}$$

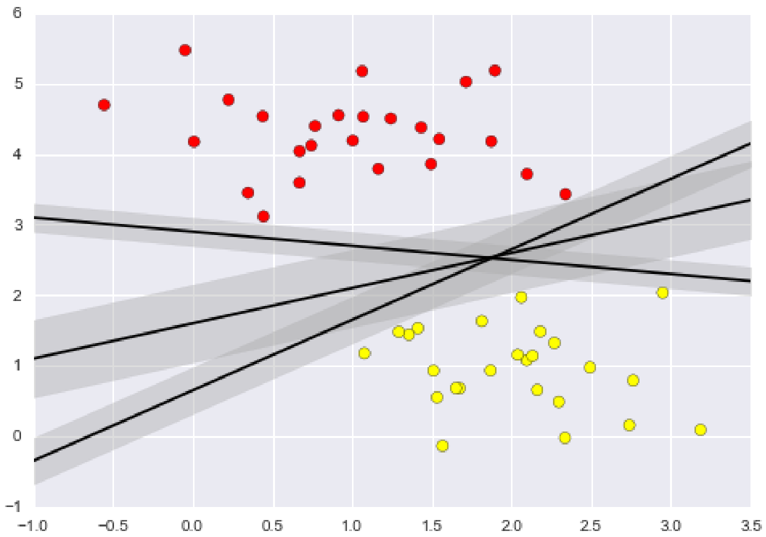


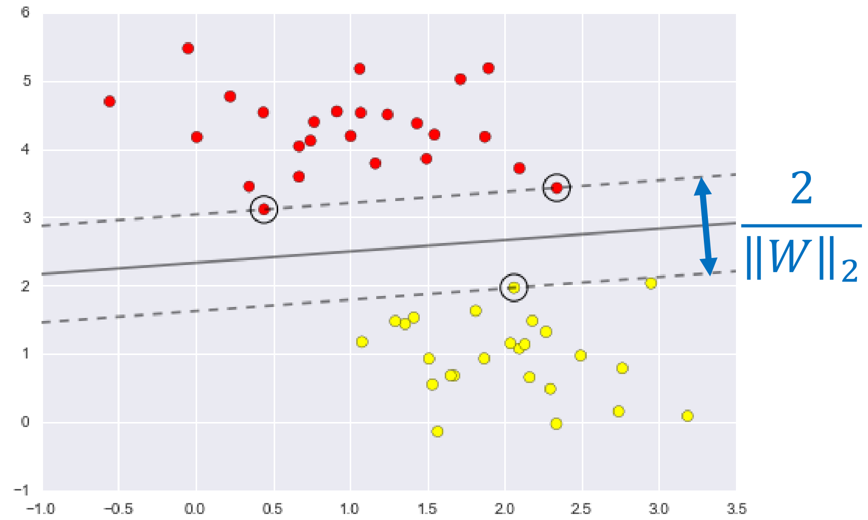
LogisticRegression in scikit-learn

- Import LogisticRegression
  - from sklearn.linear_model import LogisticRegression
- Instance creation and training
  - logi_reg = LogisticRegression().fit(X_train,y_train)
- Checking coefficients and intercept
  - print("logi_reg.coef_: ", logi_reg.coef_)   # W
  - print("logi_reg.intercept_:", logi_reg.intercept_)   # b
- Applying to new data
  - y_pred = logi_reg.predict(X_test)

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)



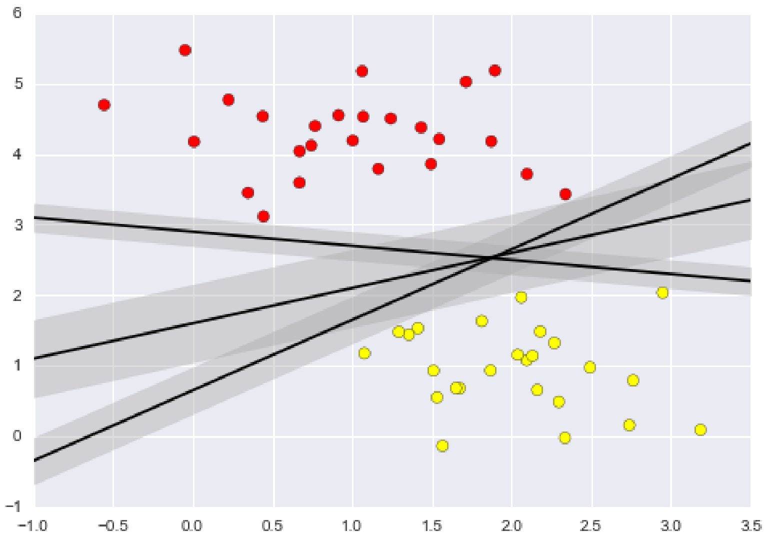


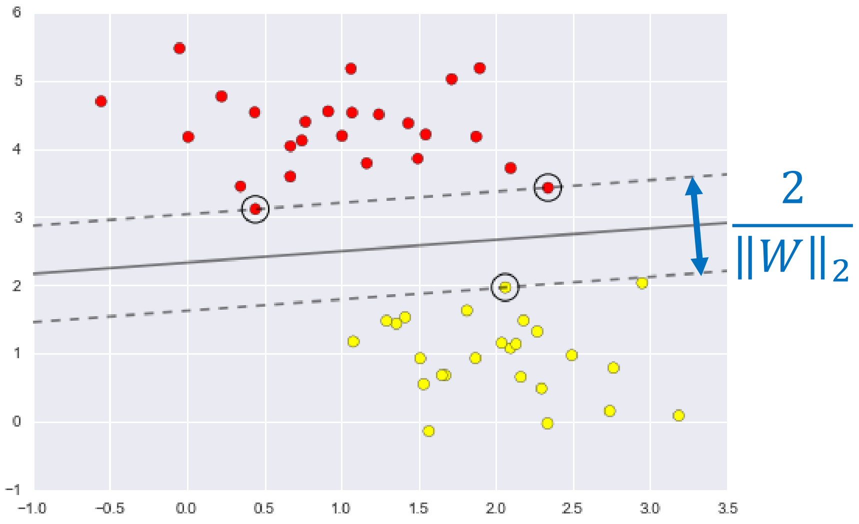
Agenda

- Bayesian Theorem
- k-Nearest Neighbor
- Linear Classifiers
- <u>Support Vector Machines</u>
- Q&A

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

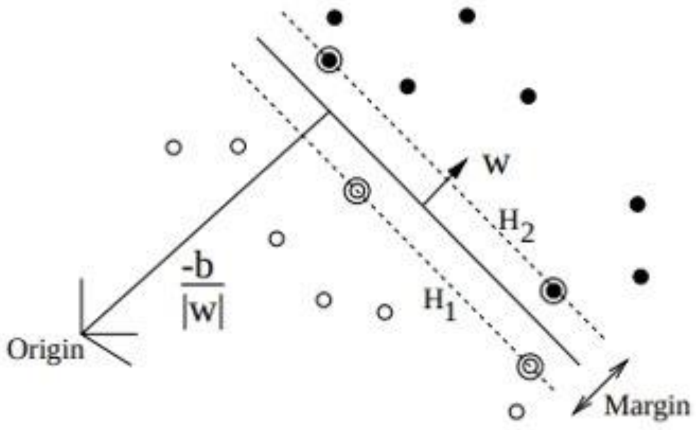
$$L_p = \frac{1}{2}|W|_2^2 - \sum_{i=1}^{l} \alpha_i y_i (WX_i + b) + \sum_{i=1}^{l} \alpha_i$$

  - $\alpha_i \geq 0$'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 $\frac{\partial L_p}{\partial W} = 0,\ \frac{\partial L_p}{\partial b} = 0,\ \alpha_i \geq 0$

Linear SVM

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



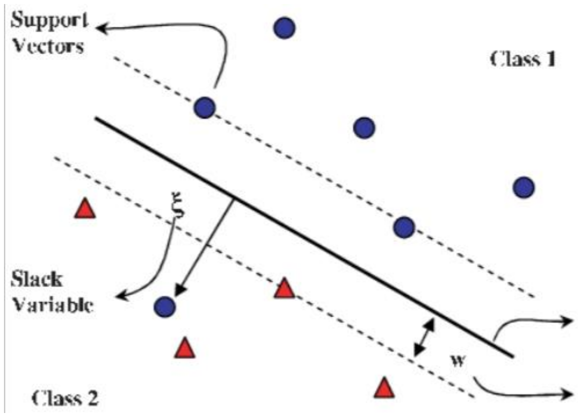
- Substitute $\frac{\partial L_p}{\partial W} = 0,\ \frac{\partial L_p}{\partial b} = 0$ into L_p

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

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

Linear SVM with Soft Margin (C-SVM)

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

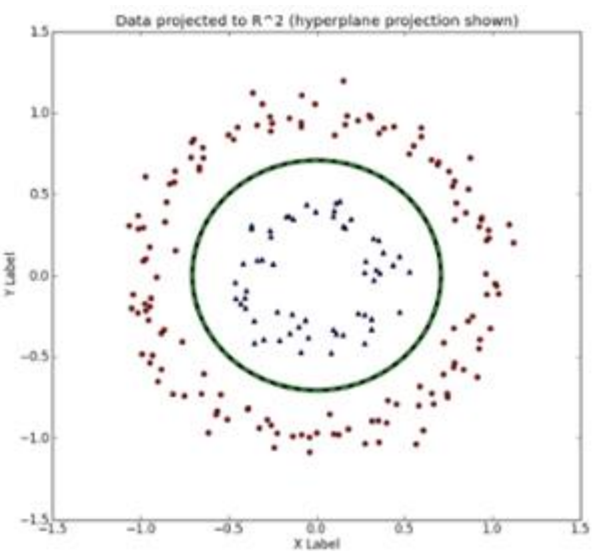


- Primal formulation

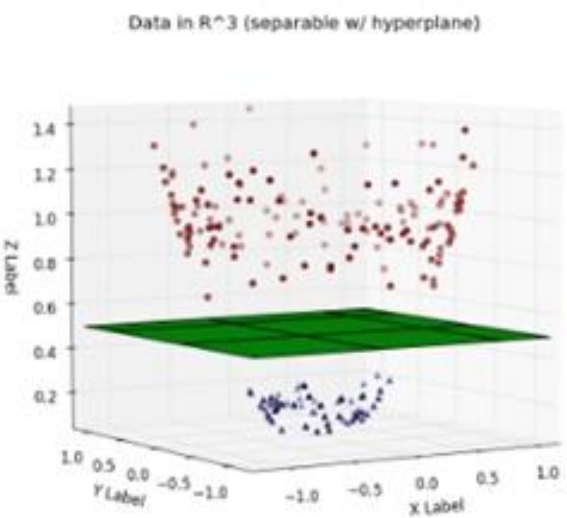
$$L_p = \frac{1}{2}|W|^2 + \boldsymbol{C}\sum_{\boldsymbol{C}}^{\boldsymbol{l}} \boldsymbol{\xi_i} - \sum_{\{i=1\}}^{l} \alpha_i \{y_i (X_iW + b) - 1 + \boldsymbol{\xi_i}\} - \sum_{\boldsymbol{i}} \boldsymbol{\mu_i}\, \boldsymbol{\xi_i}$$

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

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

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

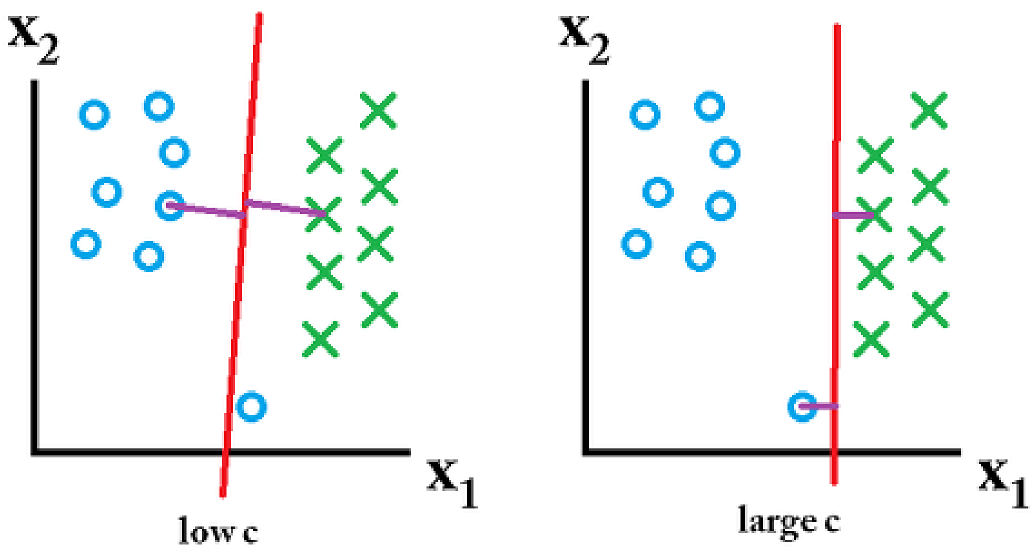
  - Example

```
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

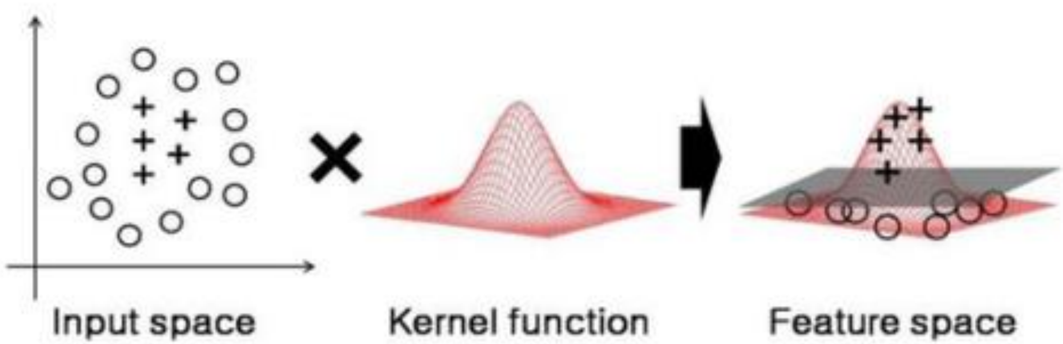
SVM Hyper-Parameters

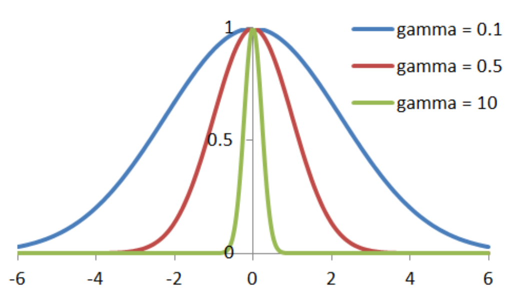
- Penalty parameter C of the error term (default: 1)
  - Too small C: underfitting
  - Too high C: overfitting



SVM Hyper-Parameters

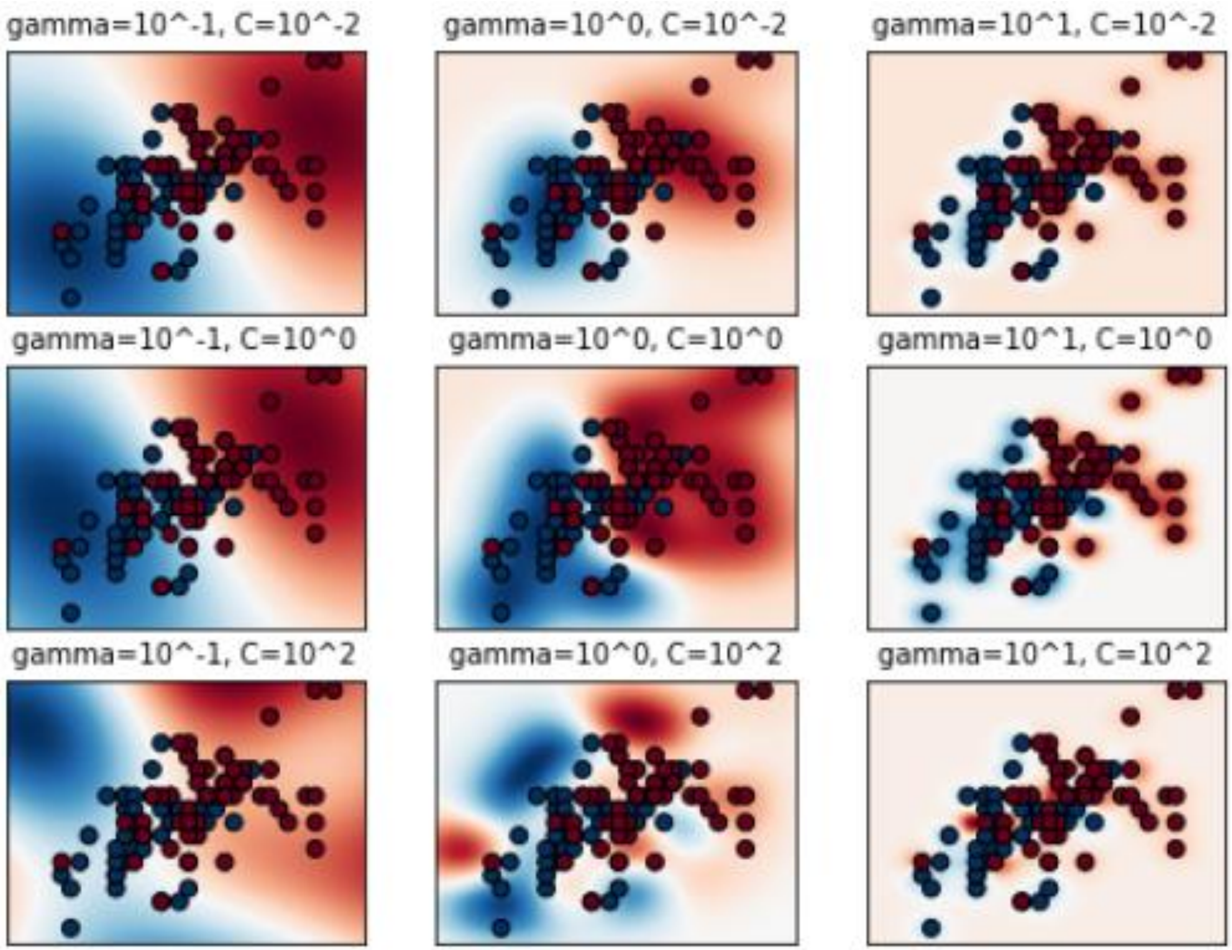
- Kernel coefficient gamma for ‘rbf’, ‘poly’ and ‘sigmoid’.
  - Too small: underfitting
  - Too large: overfitting





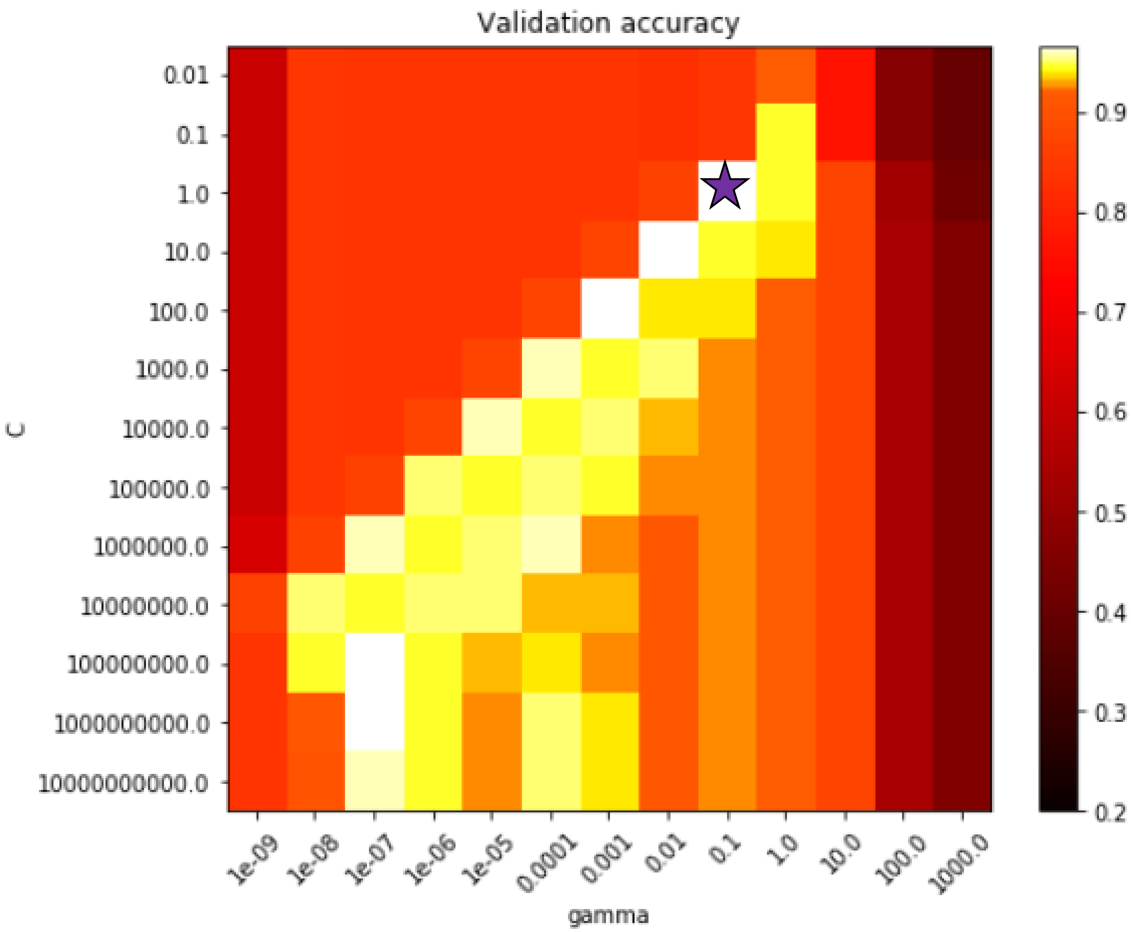
scikit-learning **gamma** options
- ‘**auto**’: 1/n_features
- ‘**scale**’: 1/(n_feature * X.var())
  X.var(): variance of X

SVM Hyper-Parameters



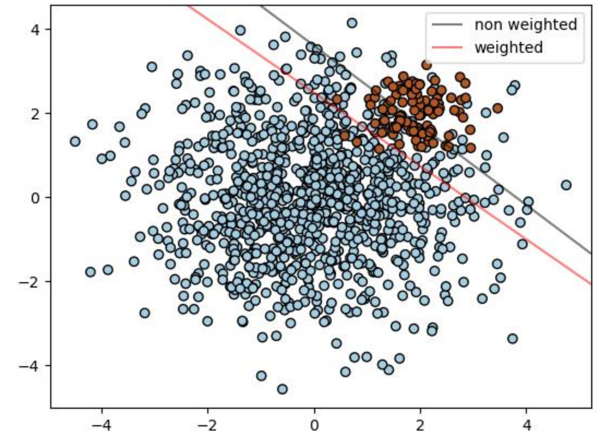
SVM Hyper-Parameters

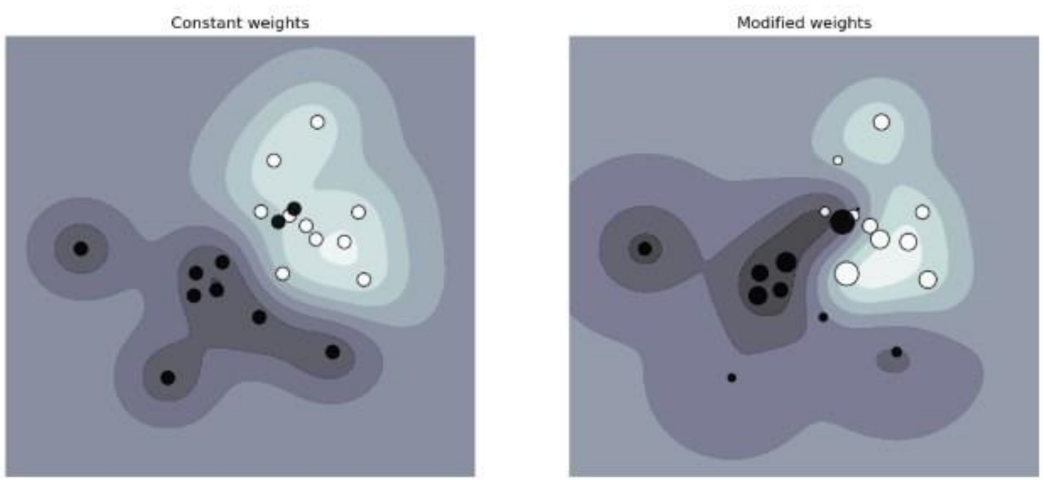
- C – gamma grid



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!