# Machine Learning

2021. 8

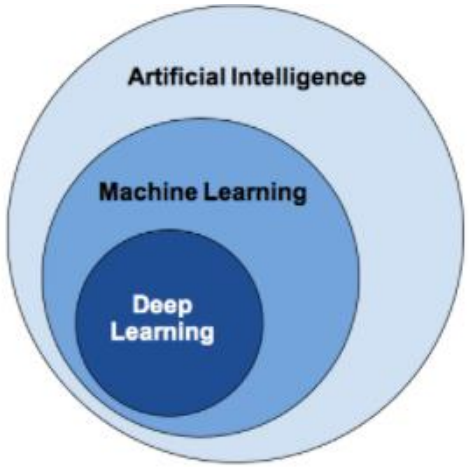
Yongjin Jeong, KwangWoon University

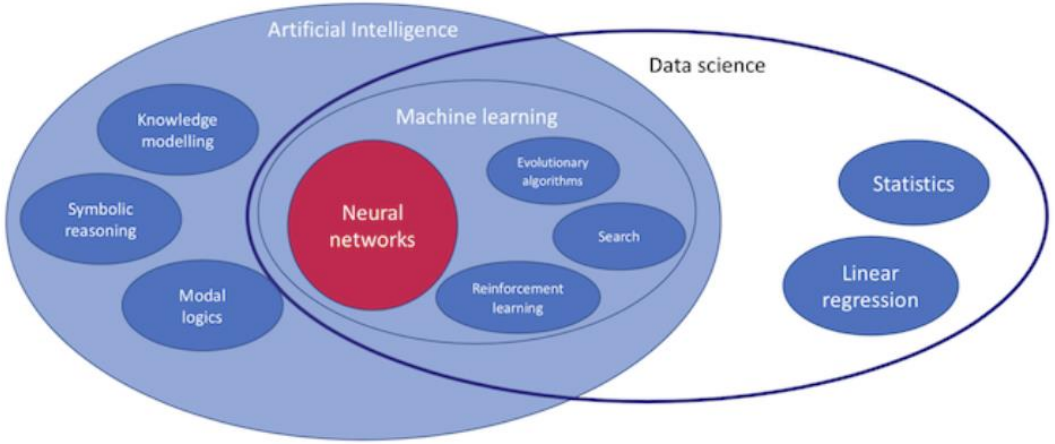
# Contents

- What is machine Learning
- Supervised Learning
  - ✓ Regression
  - ✓ Classification
- Unsupervised Learning
  - Clustering
  - Dimension reduction (SelectPercentile, PCA, t-SNE)
- Anomaly Detection
- Problems in Machine Learning
  - Overfitting and Underfitting
  - Regularization
- Validation and Testing
  - Hyper-parameter Tuning
  - K-Fold Cross Validation

# Machine Learning

- **What is ML**
  - the study of computer algorithms that improve automatically **<u>through experience and by the use of data</u>**. It is seen as a part of artificial intelligence. [wikipedia]
  - ML algorithms build a model based on sample data (training data) in order to make **predictions** or **decisions** without being explicitly programmed to do so.




[ref] https://ictinstitute.nl/ai-machine-learning-and-neural-networks-explained/

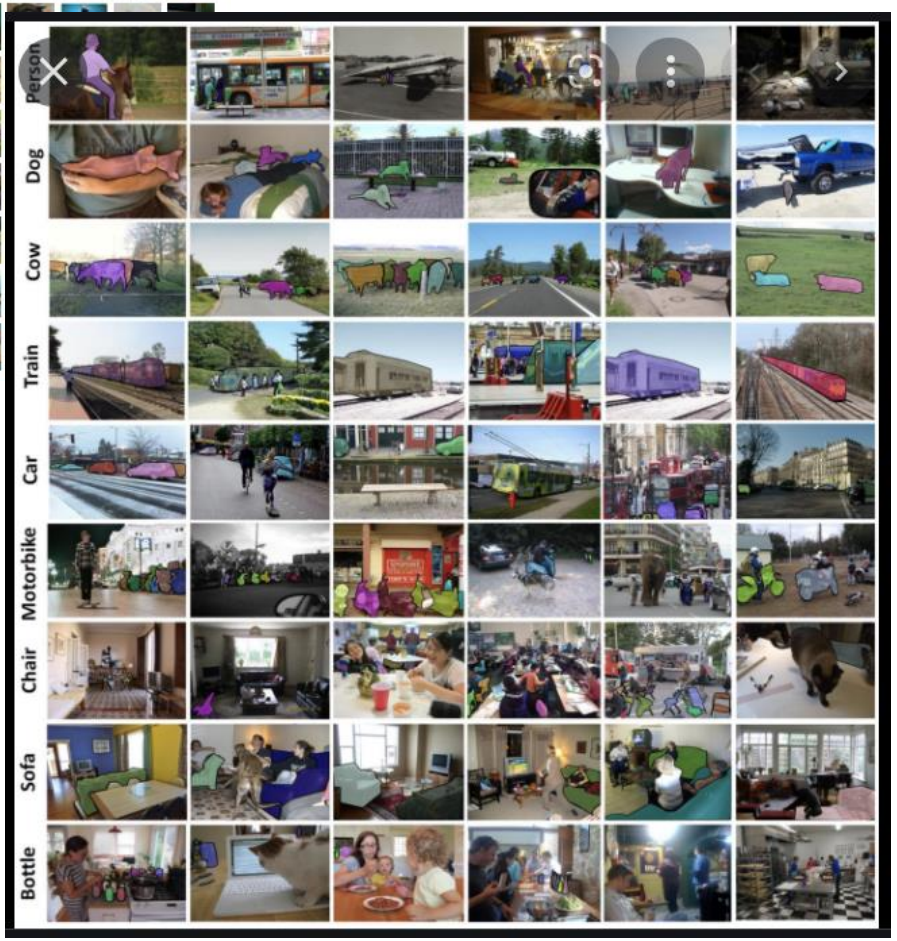
# Datasets for Machine Learning

- **Iris** dataset
  - 4-column features, 150 samples
  - Target: 3 classes
- **MNIST**(Modified National Institute of Standards and Technology dataset)
  - 28*28 grayscale pixels, 60,000 train, 10,000 test set
  - Target: single digits 0 ~ 9
- **Fashion MNIST**
  - 28*28 pixels, 60,000 train set, 10,000 test set
  - Target:10 categories
- Many more in Kaggle.com


Fashion MNIST

# Datasets for Machine Learning

- **Cifar-10 (Canadian institute for advanced research)**
  - 32*32 color images in 10 classes, with 6,000 per class
  - 50,000 training, and 10,000 test images
  - Bigger dataset (Cifar-100)
- **Microsoft COCO** (common objects in Context)
  - Large image recognition/classification, object detection, segmentation, and captioning dataset
  - 330K images (200K+ annotated), more than 2M instances in 80 object categories
  - 5 captions per image, and 250,000 people with key points

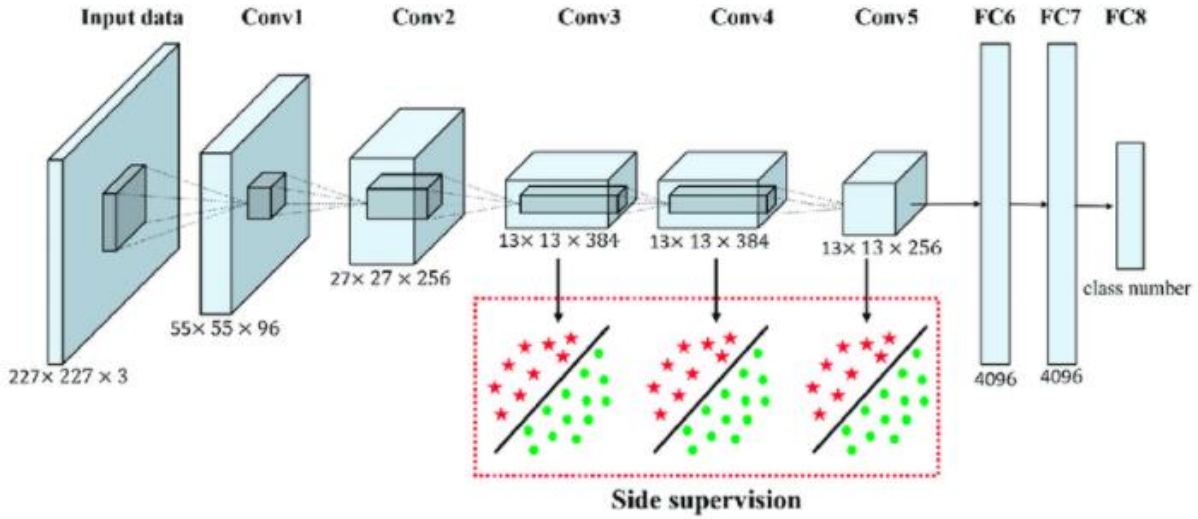



# Datasets for Machine Learning

- **ImageNet**
  - largest image dataset for computer vision, used in ILSVRC(ImageNet Large Scale Visual Recognition Challenge) for image classification and object detection (150 GB)
  - 469*387 resolution in average (usually cropped to 256*256 or 224*224 before usage)
  - More than 14 million images with more than 21,000 groups(classes)
  - More than 1 million images have bounding box annotations
- **ILSVRC**
  - To evaluate algorithms for object detection and image classification (and localization) at large scale
  - To measure the progress of computer vision for large scale image indexing for retrieval and annotation
  - Uses smaller portion of the ImageNet (1,000 categories with 1.3 million train images, 50,000 validation images, and 1,00,000 test images
  - Available in Kaggle
  - 2010 ~ 2017

# ILSVRC Winners

- **ALexNet – 2012 Winner (top-5 error rate 15.3%)**
  - Use CNN (prior to 2012, the classification model error was 25%)
  - Regarded as a Pioneer of CNN and starting point of the Deep learning
  - 60 million parameters
  - Used ReLU activation function, data augmentation, dropout, and overlapped pooling layers

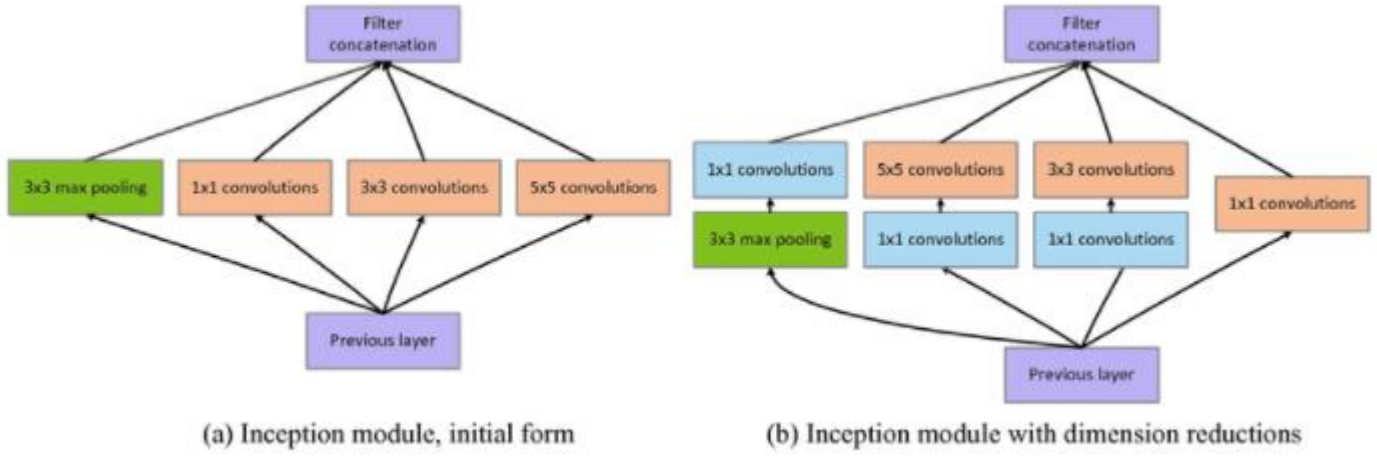



*ImageNet Challenge (2012) – AlexNet (Source)*

(*) top-5 error: The model is considered to have classified a given image correctly if the target label is one of the model's top 5 predictions. (image classification)

# ILSVRC Winners

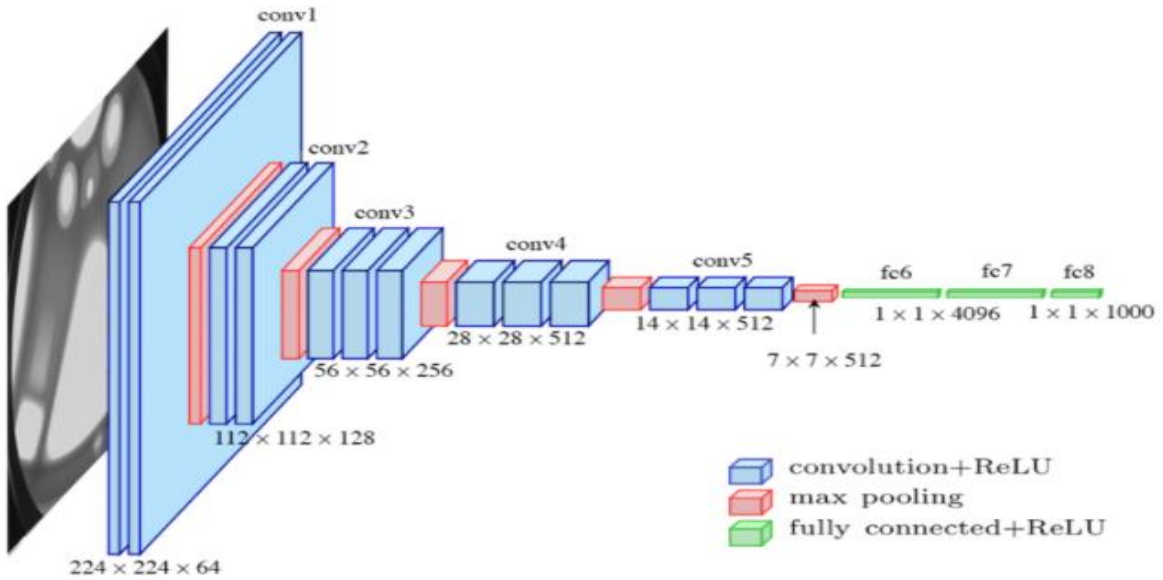
- **Inception V1 (GoogleNet) – 2014 Winner (top-5 error rate 6.67%)**
  - The v1 stands for 1st version and later there were further versions v2, v3, etc.  It is also popularly known as GoogLeNet.
  - Deep with 22 layers.
  - Used multiple types of filter size, instead of being restricted to a single filter size, in a single image block, which we then concatenate and pass onto the next layer.
  - Used 1x1 convolutional with ReLU to reduce dimensions and number of operations.




*ImageNet Challenge (2014)- Inception-V1 (GoogLeNet)* Source

# ILSVRC Winners

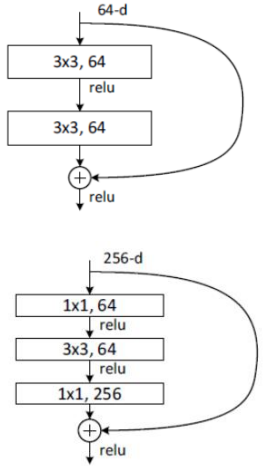
- **VGG-16 (University of Oxford) – 2014 Runners-Up (top-5 error rate 7.3%)**
  - Despite not winning the competition, VGG-16 architecture was appreciated and went on to become one of the most popular image classification models.
  - instead of using large-sized filters like AlexNet, it uses several 3×3 kernel-sized filters consecutively. The hidden layers of the network leverage ReLU activation functions.
  - VGG-16 is however very slow to train and the network weights, when saved on disk, occupy a large space.



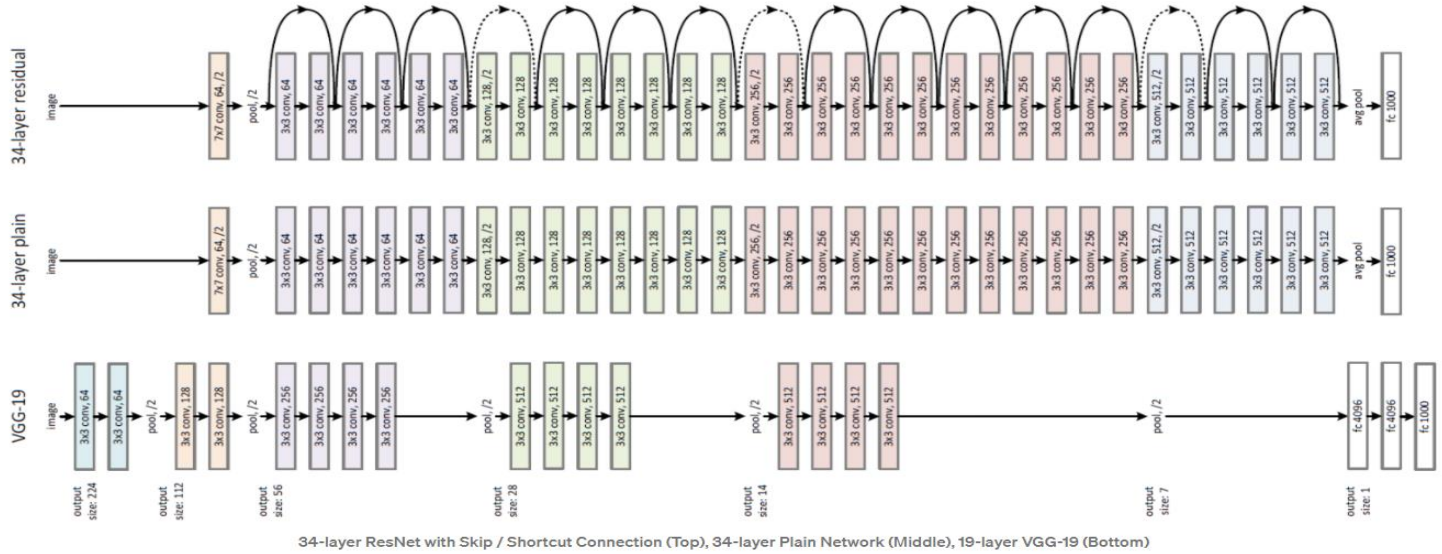

*ImageNet Challenge (2014) – VGG-16 (Source)*

# ILSVRC Winners

- **ResNet – 2015 Winner (top-5 error rate 3.57%)**
  - ResNet (Residual Network) was created by the Microsoft Research team.
  - To solve the problem of vanishing/exploding gradients, a skip/shortcut connection is added to add the input $x$ to the output after few weight layers as below:
  - 1×1 Conv can reduce the number of connections (parameters) while not degrading the performance of the network so much. (as in Inception-V1)
  - ResNet-18/34/50/101/152 has 1.8/3.6/3.8/7.6/11.3 GFLOPs (lower than VGG-16/19 with 15.3/19.6 GFLOPS)




The Basic block (top) and the Proposed Bottleneck design (bottom)

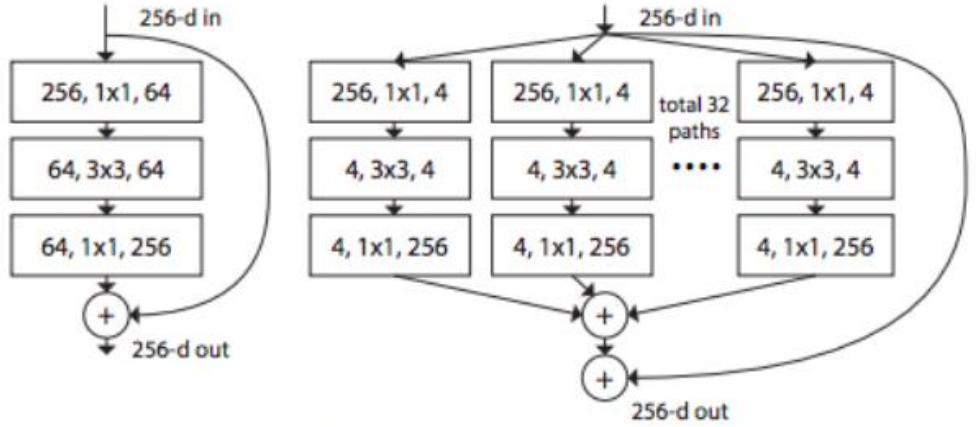



34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom)

(*) VGG-19 (bottom): state-of-the-art approach in 2014
middle: deeper network of VGG-19 (i.e. more Conv layers)

# ILSVRC Winners

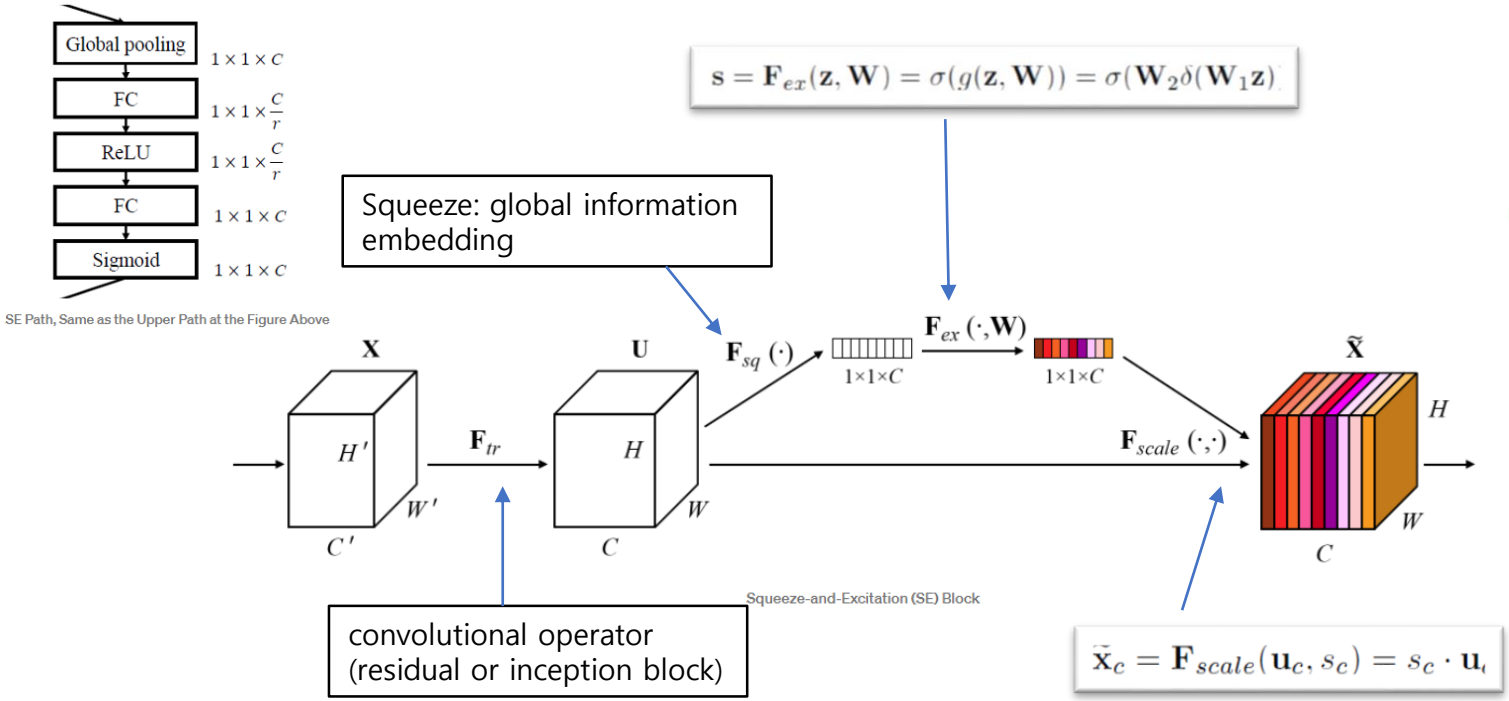
- **ResNeXt – 2016 Runners-Up (top-5 error rate 4.1%)**
  - Developed in the collaboration of the Researchers from UC San Diego and Facebook AI Research.
  - Inspired by (ResNet + VGG + Inception).
  - Still it became a popular model.
  - stacks the blocks and then use the ResNet approach of residual blocks. Here the hyper-parameters such as width and filters were also shared.

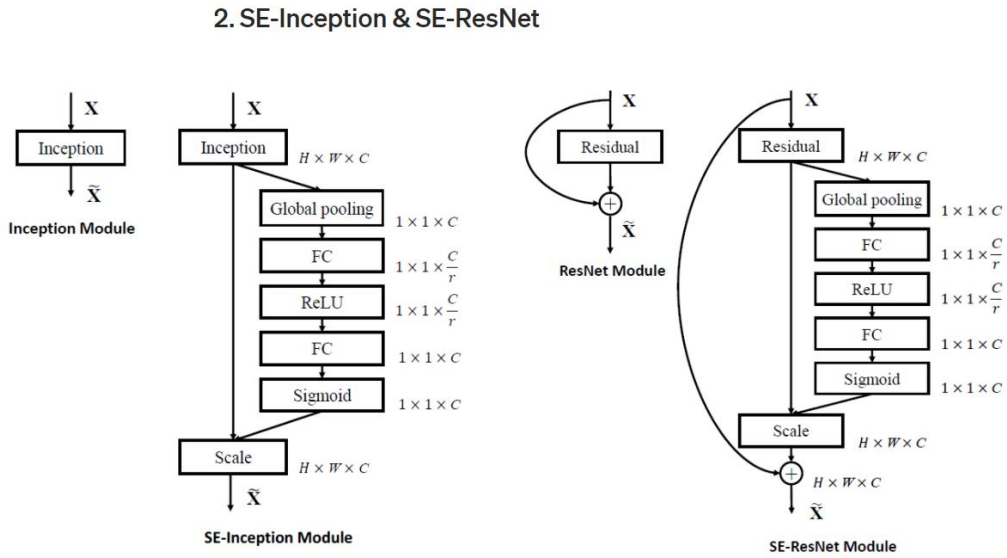



*ImageNet Challenge (2016)- ResNeXt (Source)*

# ILSVRC Winners

- **SENet(Squeeze-and-Excitation Network) – 2017 Winner (top-5 error rate 2.251%)**
  - Developed in University of Oxford.
  - With "Squeeze-and-Excitation" (SE) block that **adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels**, SENet is constructed.
  - SE block can be added to both Inception and ResNet block easily as **SE-Inception** and **SE-ResNet**. (achieved the best 2.25%).



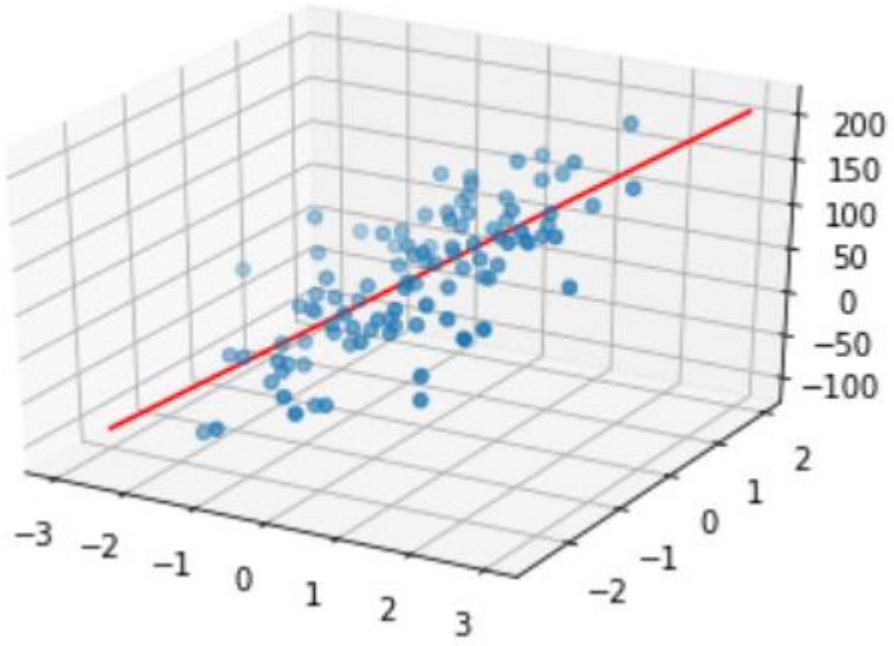

(*) Human Beings – Top-5 Error Rate – 5.1%

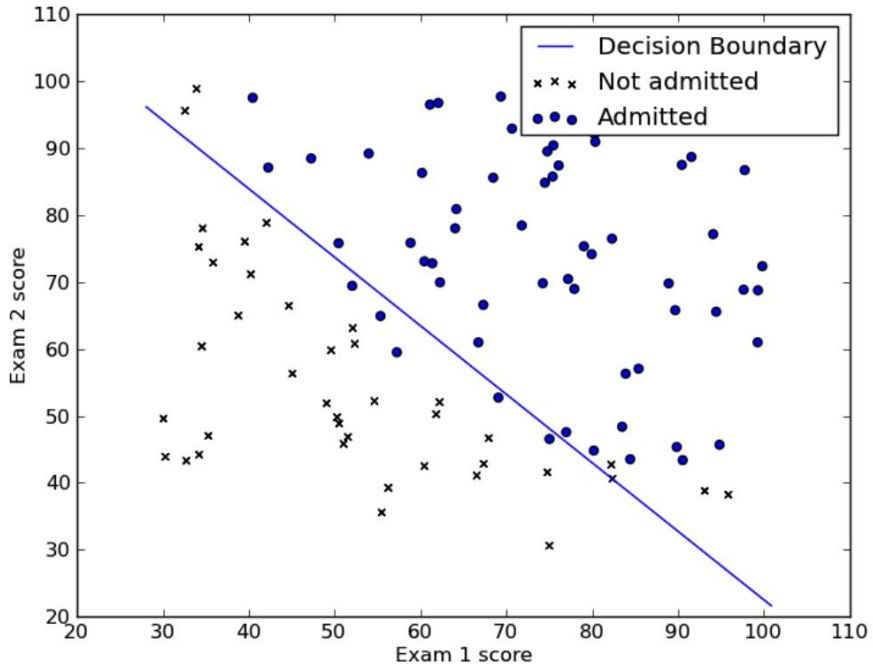
# Machine Learning

- **Machine Learning approaches**
  - Supervised learning: inputs(features) and outputs(labels) are both given (by a teacher) to learn a general rule to map features to labels
    - **Regression** : labels are continuous values
    - **Classification**: labels are categorical values
  - Unsupervised learning: no labels are given, and to discover hidden patterns or features in data
    - **Dimension reduction**: PCA(Principal Component Analysis)
    - **Clustering** (or grouping)
  - Semi-supervised learning: large unlabeled data with small labeled data
  - Reinforcement learning: interacts with environment by producing actions and discovers errors or rewards (trial and error search, delayed reward)
    - Model-based RL (like control theory)
    - Model-free RL: Policy-iteration (Policy gradient, A3C) and value-iterations (Q-learning)

# Supervised Learning

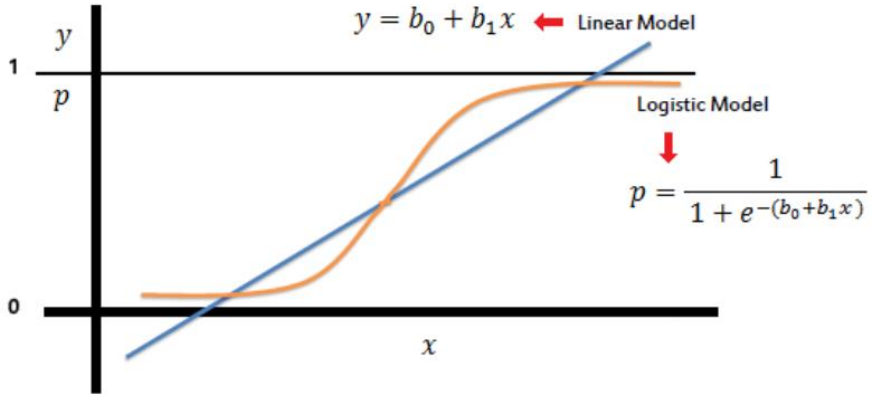
- **Linear Regression and Linear Classification**
  - (regression) $y = w1 \cdot x1 + w2 \cdot x2 + b$
  - (classification) $w1 \cdot x1 + w2 \cdot x2 + b = 0$

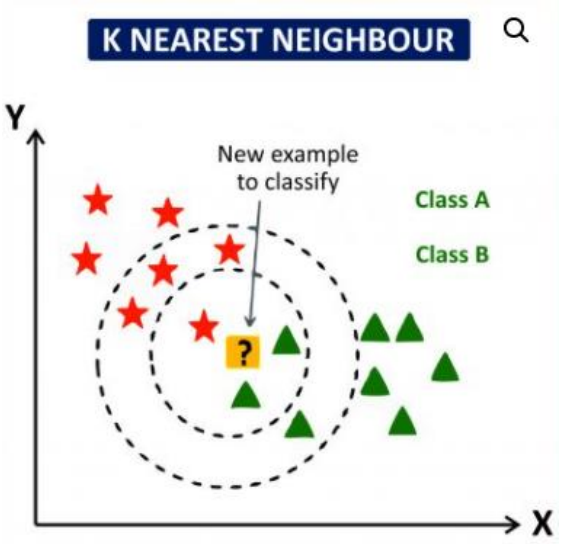


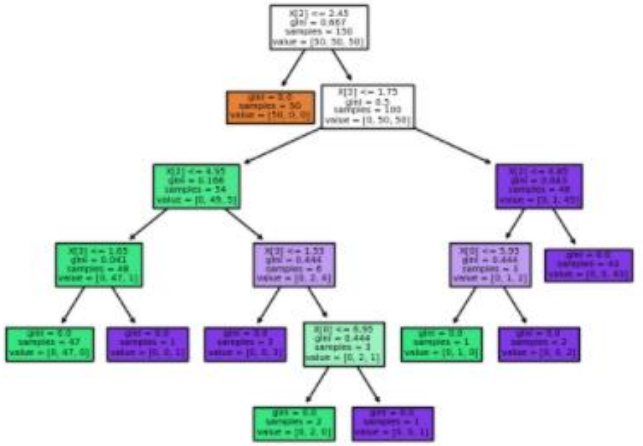



# Supervised Learning

- **Other algorithms (linear and nonlinear)**
  - Logistic Regression Classifier
  - Knn (k-nearest neighbor)
  - Decision Tree

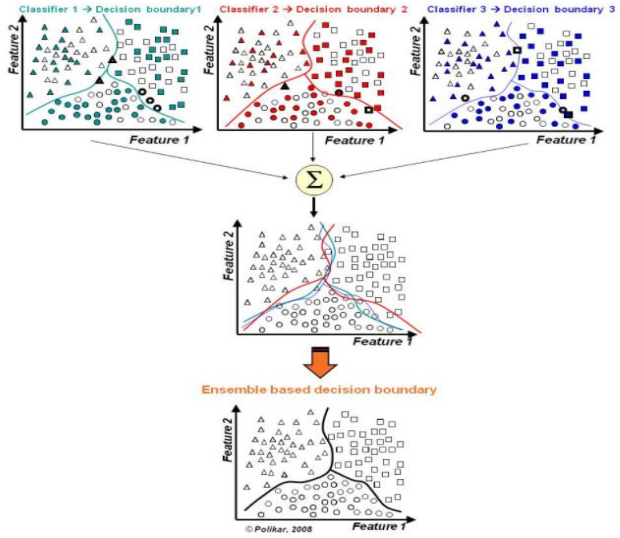


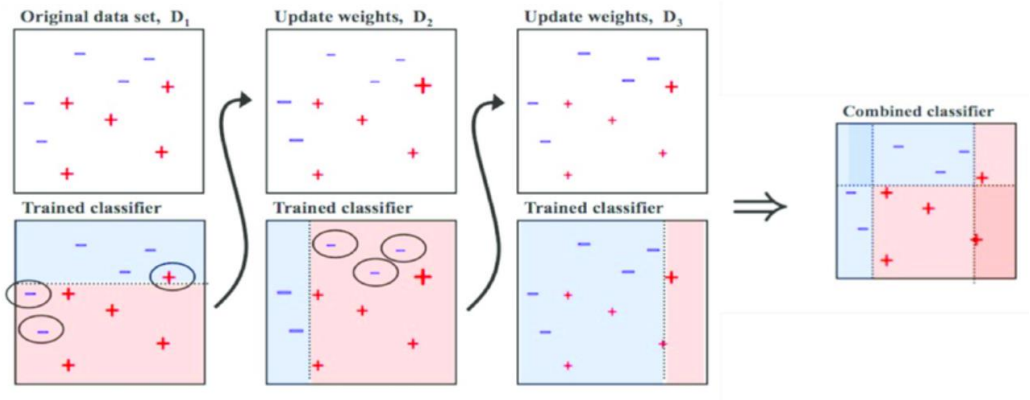



# Supervised Learning

- **Ensemble method**
  - Combine many weak learners
  - Bagging: learns them independently in parallel and average them (ex: Random Forest)
  - Boosting: learns them sequentially in adaptive way and combines them (ex: Gradient Boost)



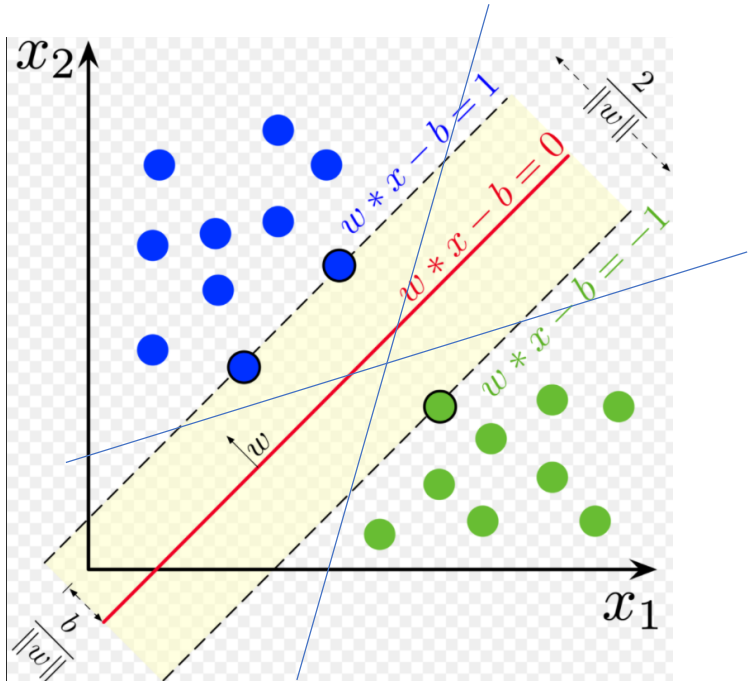

출처: https://swalloow.github.io/bagging-boosting

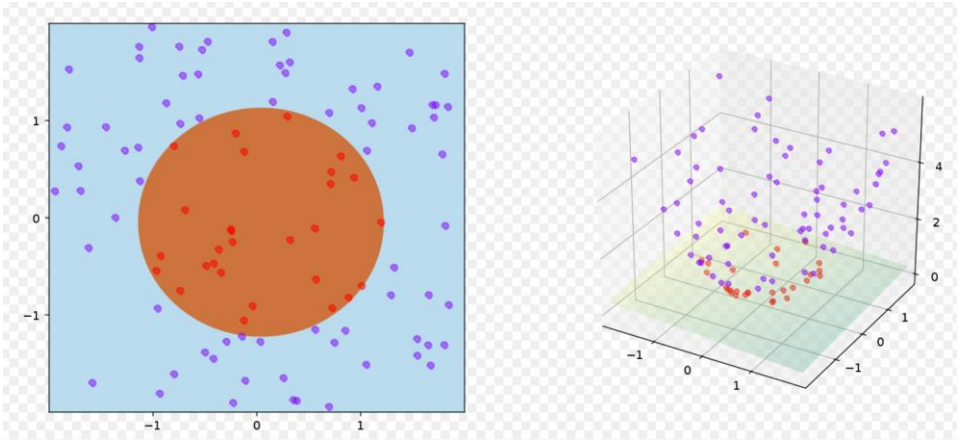



출처: Medium (Boosting and Bagging explained with examples)

# Supervised Learning

- **SVM (Support Vector Machine)**
  - Finds a maximum-margin hyperplane
  - Linear SVM
  - Nonlinear SVM: use kernel (polynomial, sigmoid, rbf)

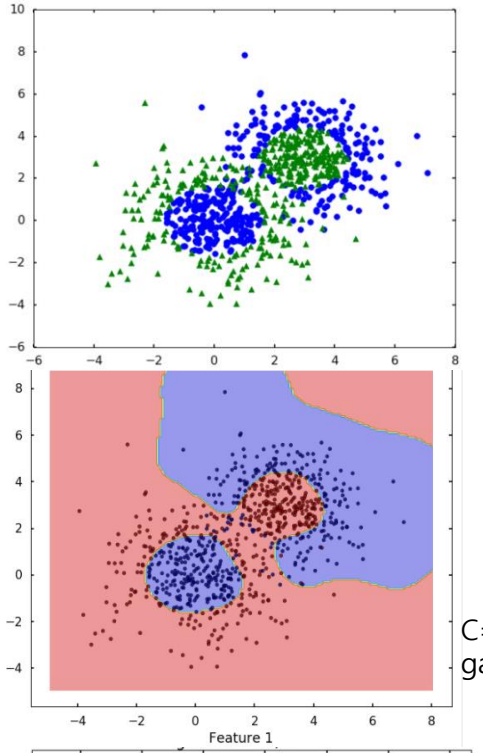


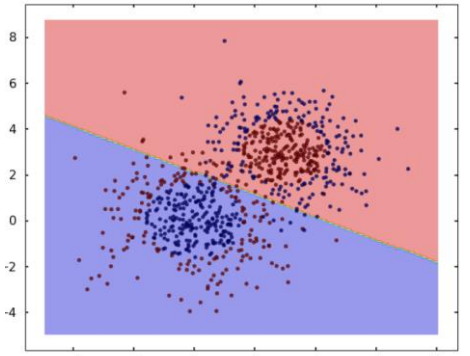



SVM with kernel given by $\varphi((a, b)) = (a, b, a^2 + b^2)$

# Supervised Learning

- **SVM (continued)**
  - '**rbf**' kernel (hyperparameters: C and gamma)
  - C: tradeoff between classification error and simplicity of the boundary
  - gamma: defines how far the influence of a single example reaches (high value is 'close')
  - class_weight: imbalance cases

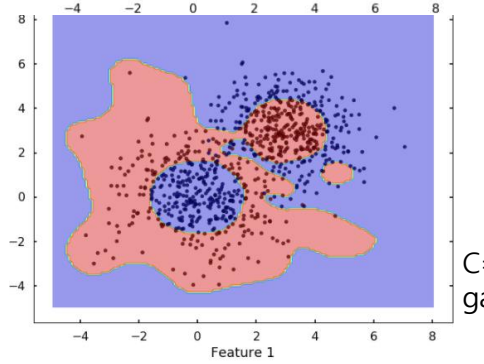




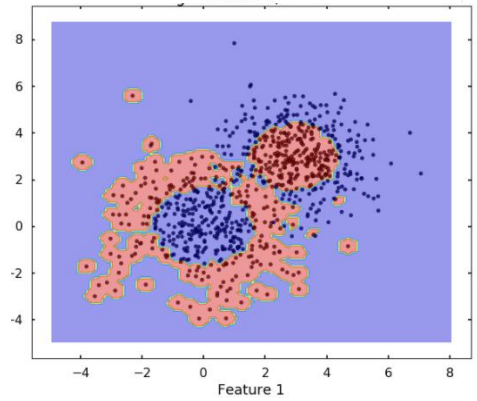
linear



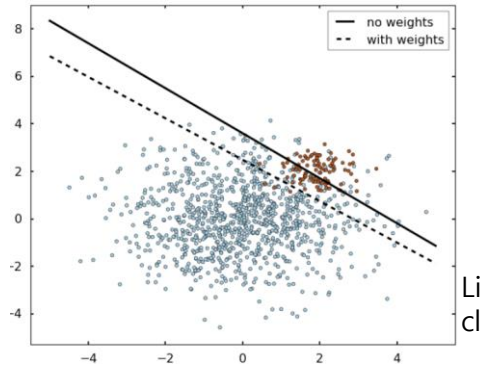

C=1
gamma=0.5




C=10000
gamma=0.5



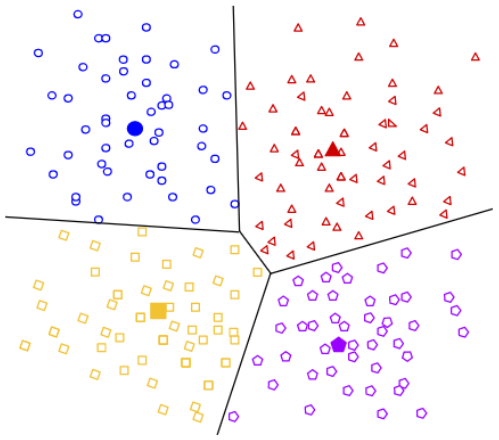

C=1
gamma=50

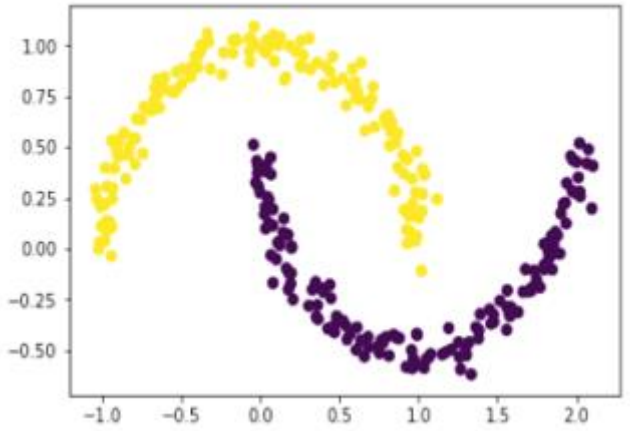


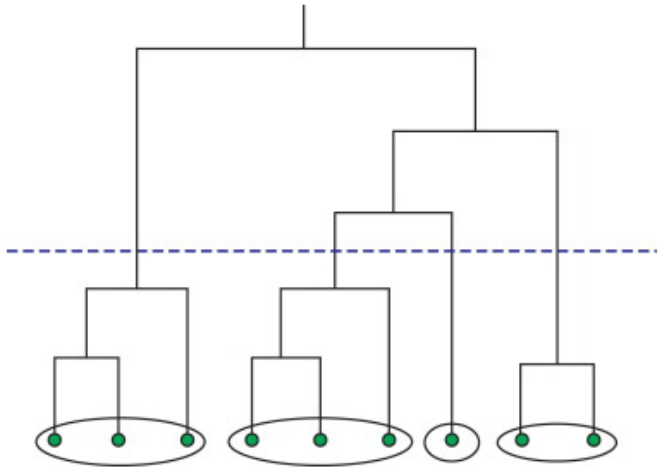

Linear with
class_weight=1:10

# Unsupervised Learning

- **Clustering (or Grouping)**
  - Divide the data points into several clusters based on similarity (유사도)
  - Need scaling as a preprocessing step
  - Applications: detect hackers and criminal activity, identifying fake news, etc.
  - Centroid-based (K-means)
  - Density-based (DBSCAN)
  - Hierarchical (ex: dendrogram)

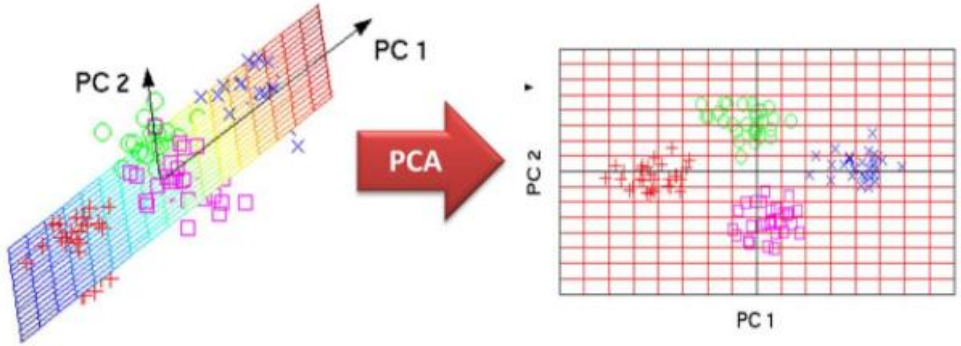




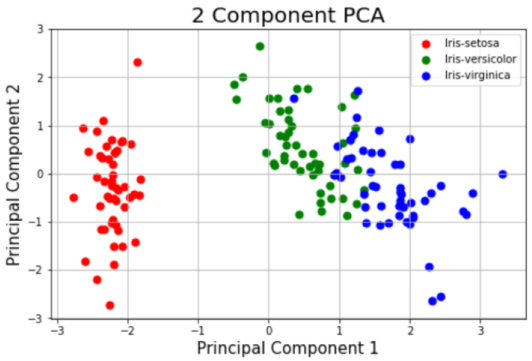
# Unsupervised Learning

- **Dimension reduction by PCA**
  - Standard scaling
  - Calculate covariance (or correlation) matrix
  - Eigen-decomposition ($A = P\Lambda P^{-1}$)
  - Select k eigenvectors
  - pca_result = PCA(n_components=2).fit_transform(X_all)
  - Also, SelectPercentile(), tSNE()



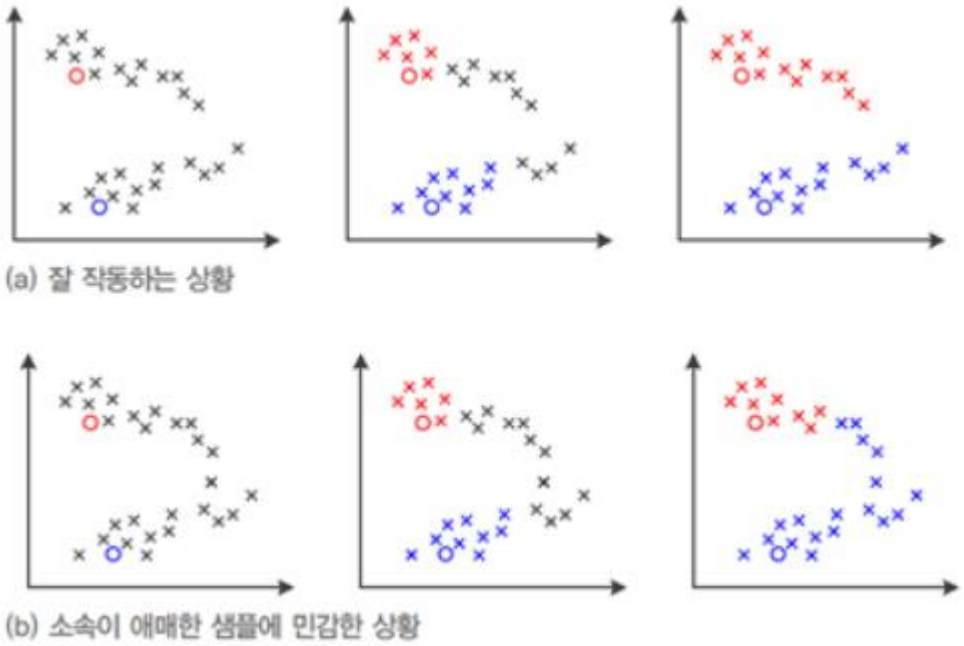

| | sepal_len | sepal_wid | petal_len | petal_wid | class |
|---|---|---|---|---|---|
| 145 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virgi |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virgi |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virgi |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virgi |

| | principal component 1 | principal component 2 | species |
|---|---|---|---|
| 0 | -2.264542 | -0.505704 | Iris-setosa |
| 1 | -2.086426 | 0.655405 | Iris-setosa |
| 2 | -2.367950 | 0.318477 | Iris-setosa |
| 3 | -2.304197 | 0.575368 | Iris-setosa |
| 4 | -2.388777 | -0.674767 | Iris-setosa |



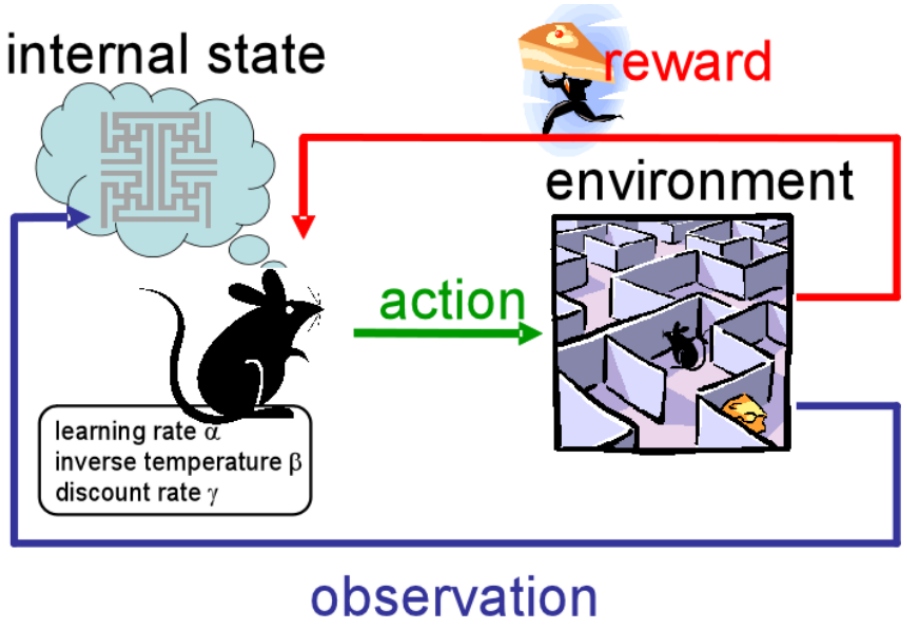
# Semi-supervised Learning

- **Semi-supervised**
  - large unlabeled data with small labeled data
  - Will unlabeled data be helpful? Yes or No
  - Self-training:
    - Train with labeled data
    - Classify unlabeled data
    - Select samples with high confidence and include them in labeled data
    - Retrain the classifier
    - Repeat
  - Using GAN
  - Many others

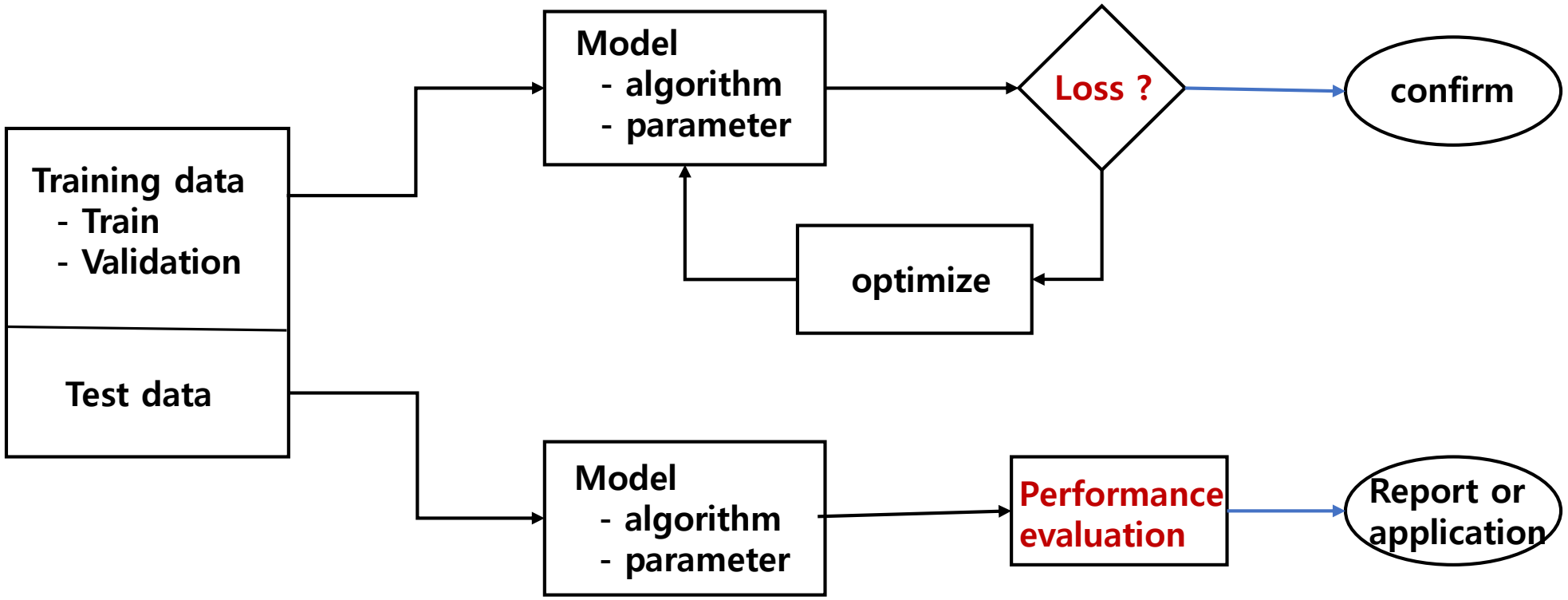


# Reinforcement Learning

- **Reinforcement learning**
  - A computer learns to perform a task through repeated trial-and-error interactions with a dynamic environment.
  - It enables the computer to make a series of decisions that maximize a reward (with human intervention and explicit program).



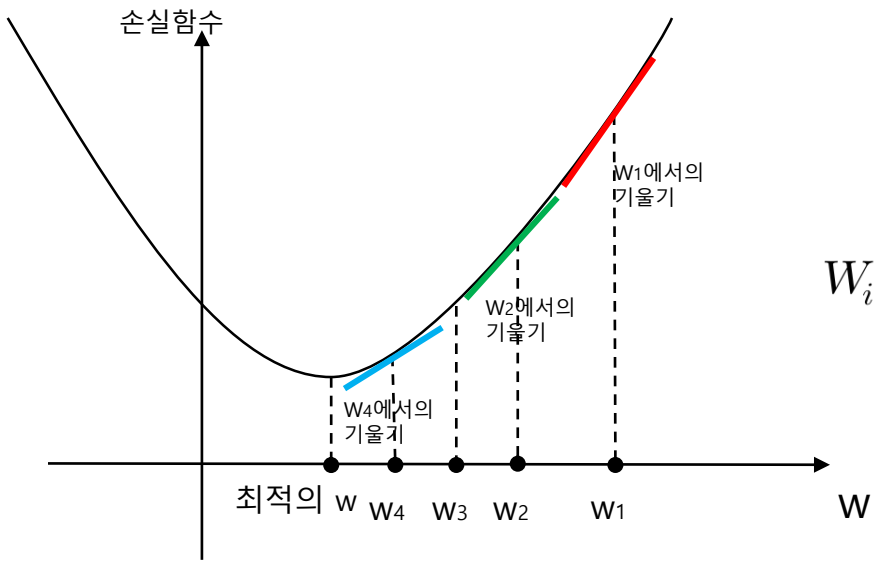
# Machine Learning Model (supervised)



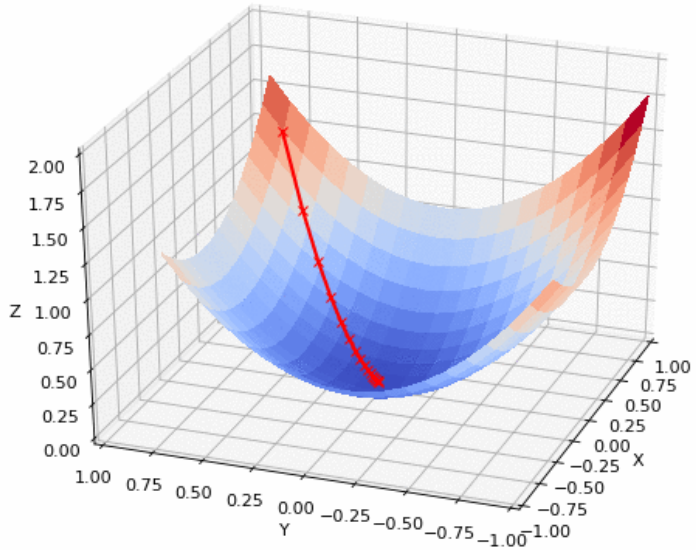

- (model) Parameter: estimated from the dataset (learned during training from the historical data sets)
- Hyper-parameter: external to the model (defined manually before the model training by trial-error)

# Gradient Descent (GD) algorithm

- **Gradient Descent (경사하강법)**
  - General optimization algorithm
  - take repeated steps in the opposite direction of the **gradient** (or approximate **gradient**) of the function at the current point

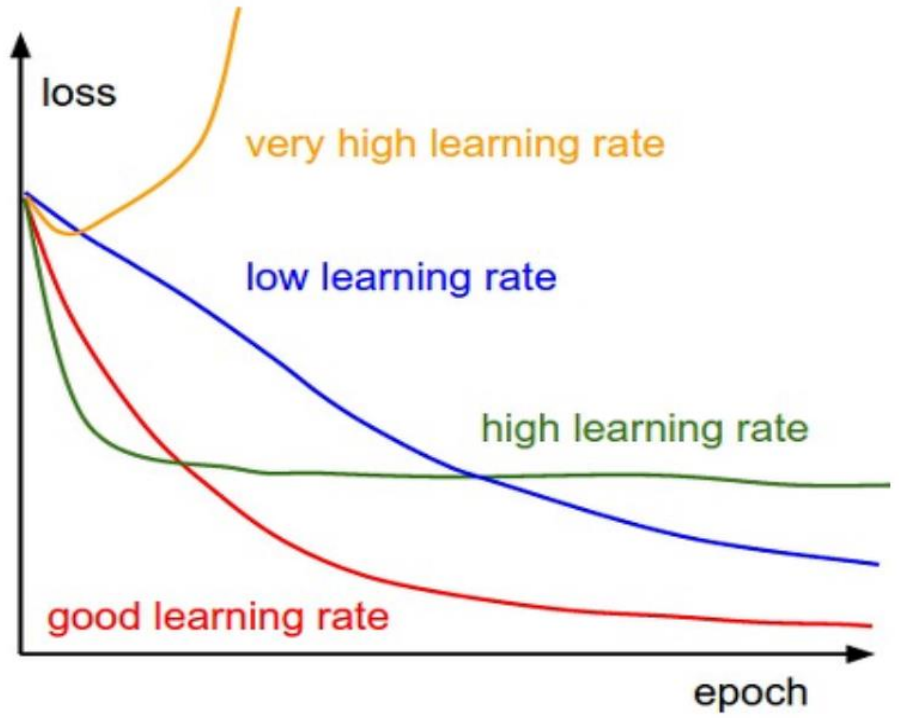



$$W_i = W_{i-1} - \eta Grad(i)$$



# Gradient Descent (GD) algorithm

- **Learning rate: η (eta)**
  - low: takes time to converge, and may get stuck in an undesirable local minimum
  - high: may jump over minima
  - too high: may diverge
  - Need adaptive adjustment



# Loss Function

- **What to reduce? (Loss or Error or Cost: 손실함수)**
  - Regression (회귀): MSE (Mean Square Error)

$$MSE = \sum_{k=1}^{N} (y - \hat{y})^2$$

  - Classification (분류): Cross Entropy (CE), Gini Coefficient

$$CE = \sum_{i} p_i \log(\frac{1}{p_i}) \qquad Gini = 1 - \sum_{k=1}^{m} p_k^2$$

  - Binary case:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

# Loss Function

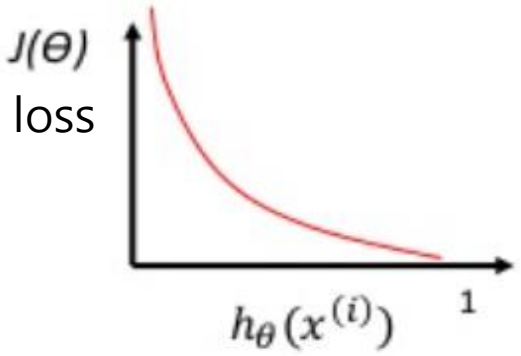
- **Binary Cross Entropy**

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$


Sigmoid ft'n

For $y^{(i)} = 1$ *case*

$J(\Theta) = -\log h_\theta(x^{(i)})$




predition

For $y^{(i)} = 0$ *case*

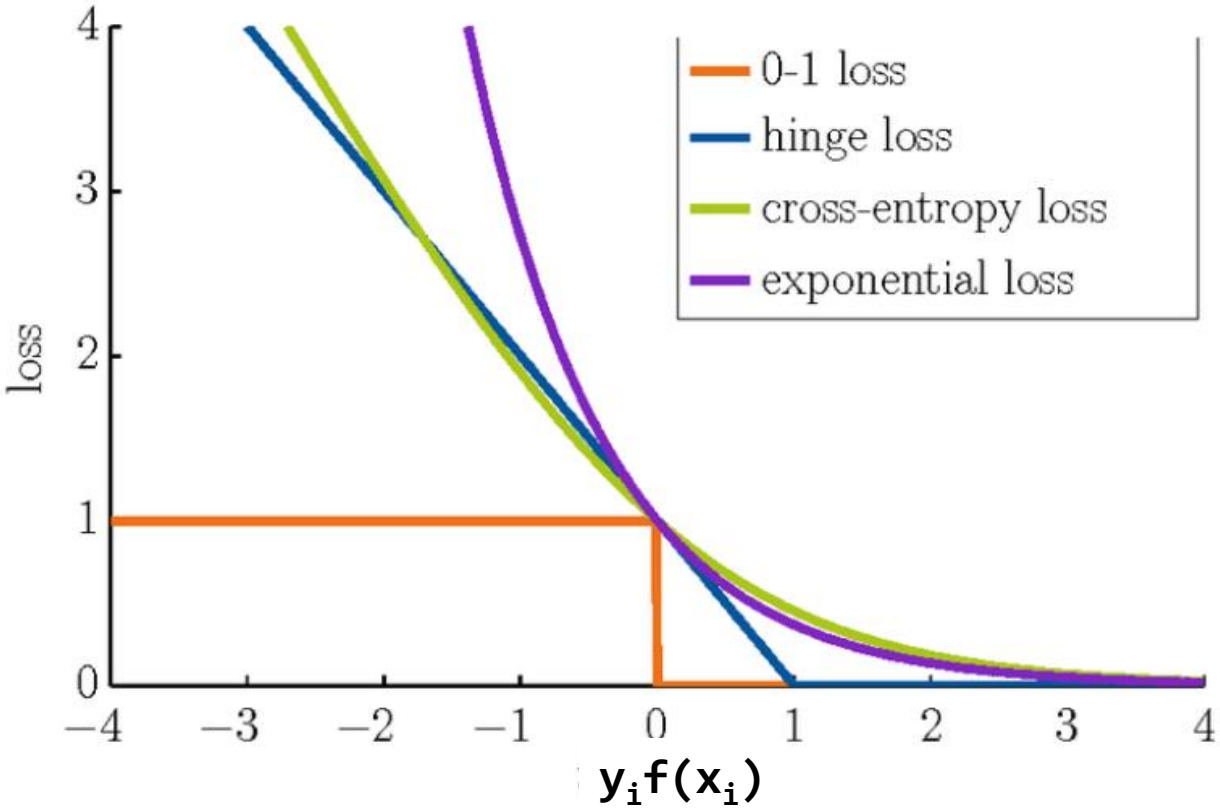
$J(\Theta) = -\log(1 - h_\theta(x^{(i)}))$

# Classification Loss Functions

- Truth value, $y_i$: [1, -1, 1, -1, 1]
- Output score: $f(x_i)$-> + (likely +1)
  -> - (likely -1)
- $y_i f(x_i)$ 가 negative 일 때 얼마나 penalty 를 줄 것인지.
- **0/1 Loss**

  **Li = 0 , if $y_i f(x_i) \geq 0$**
  **1 , if $y_i f(x_i) < 0$**

- **Exponential loss**
  - $e^{-y_i f(x_i)}$
- **Hinge Loss (SVM Loss)**
  - Max [0, 1 - $y_i f(x_i)$ ]
- **Cross Entropy (Log loss)**

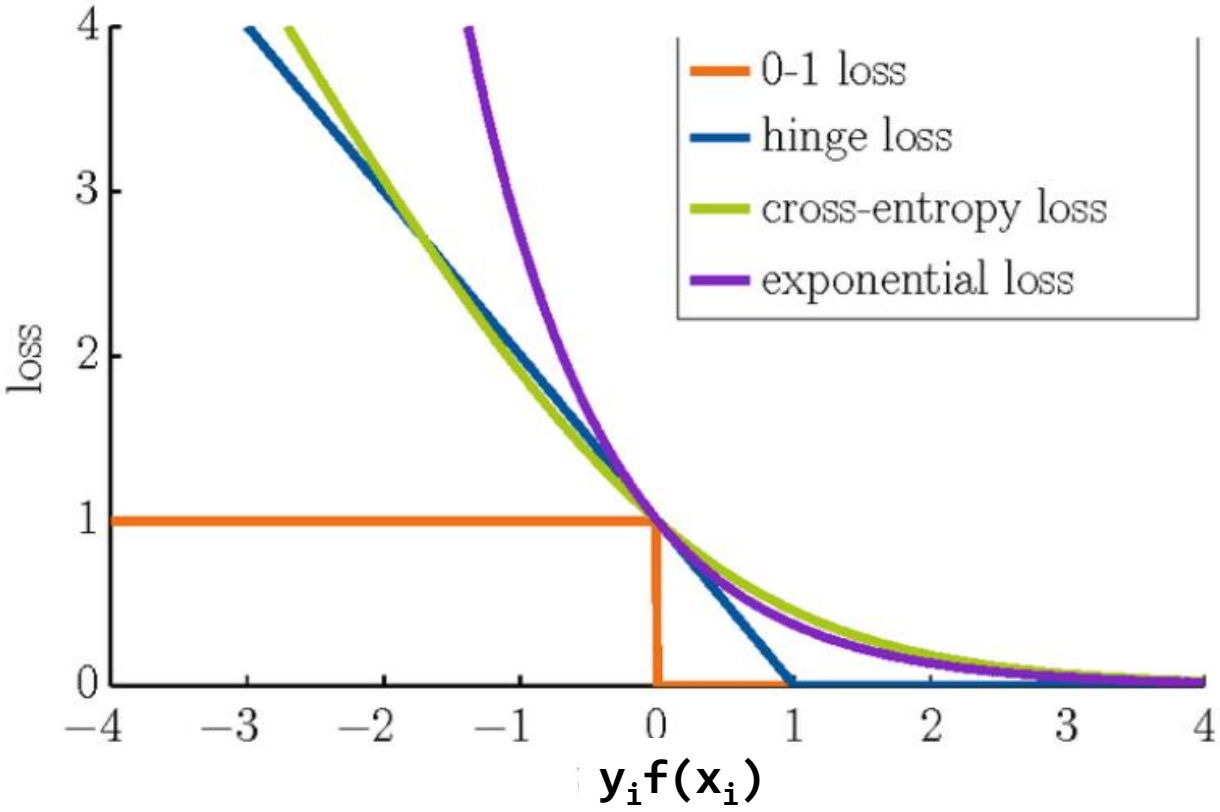


# Classification Loss Functions

- Truth value, $y_i$: [1, -1, 1, -1, 1]
- Output score: $f(x_i)$-> + (likely +1)
  -> - (likely -1)
- $y_i f(x_i)$ 가 negative 일 때 얼마나 penalty 를 줄 것인지.
- **0/1 Loss**

  **Li = 0 , if $y_i f(x_i) \geq 0$**
  **1 , if $y_i f(x_i) < 0$**

- **Exponential loss**
  - $e^{-y_i f(x_i)}$
- **Hinge Loss (SVM Loss)**
  - Max [0, 1 - $y_i f(x_i)$ ]
- **Cross Entropy (Log loss)**



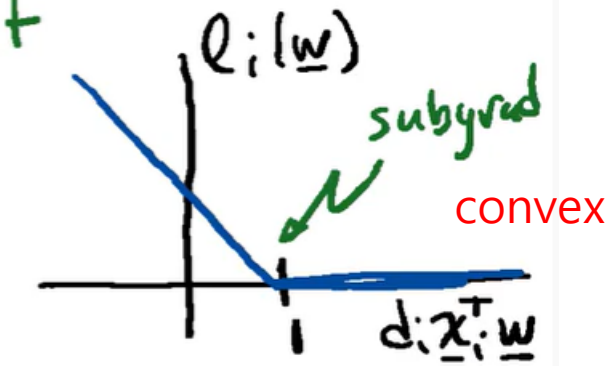
# Sub-gradient Descent

## Gradient descent for SVMs



$$\ell(\underline{w}) = \sum_{i=1}^{N} (1 - d_i \underline{x}_i^T \underline{w})_+ \rightarrow \text{subgradient}$$

$$\ell_i(\underline{w}) = (1 - d_i \underline{x}_i^T \underline{w})_+ = \begin{cases} 1 - d_i \underline{x}_i^T \underline{w} & d_i \underline{x}_i^T \underline{w} < 1 \\ 0 & d_i \underline{x}_i^T \underline{w} > 1 \end{cases}$$




Subgradient

$$\underline{v}_i(\underline{w}) = \begin{cases} -d_i \underline{x}_i & d_i \underline{x}_i^T \underline{w} < 1 \\ 0 & d_i \underline{x}_i^T \underline{w} \geq 1 \end{cases} = -d_i \underline{x}_i I_{\{d_i \underline{x}_i^T \underline{w} < 1\}}$$

indicator function

Cost $f(\underline{w}) = \ell(\underline{w}) + \lambda \|\underline{w}\|_2^2$

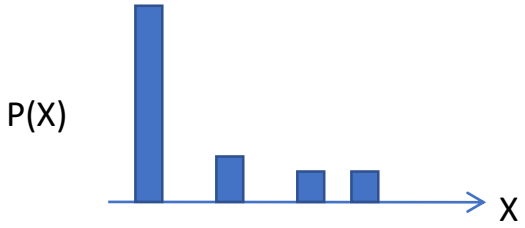
$$\Rightarrow \nabla f(\underline{w})\big|_{\underline{w}^{(k)}} = \sum_{i=1}^{N} \left(-d_i \underline{x}_i I_{\{d_i \underline{x}_i^T \underline{w}^{(k)} < 1\}}\right) + 2\lambda \underline{w}^{(k)}$$

Gradient descent $\underline{w}^{(k+1)} = \underline{w}^{(k)} - \tau \nabla f(\underline{w})\big|_{\underline{w}^{(k)}}$

https://www.youtube.com/watch?v=vi7VhPzF7YY

# Entropy

- P(X) encodes our uncertainty about X
  - Some variables are more uncertain than others




  - How can we quantify this intuition?
    - Information: $\log \frac{1}{p(x)}$
    - Entropy: average number of bits required to encode discrete R.V. X

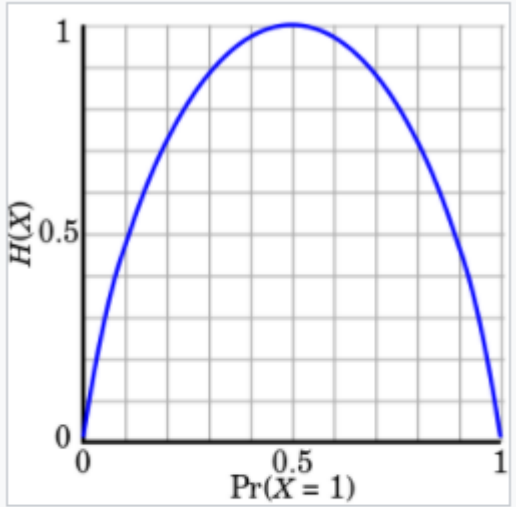
$$H_P(X) = E\left[\log \frac{1}{p(x)}\right] = \sum_x P(x) \log \frac{1}{P(x)} = -\sum_x P(x) \log P(x)$$

# Binary Entropy

$$H_P(X) = E\left[\log \frac{1}{p(x)}\right] = \sum_x P(x) \log \frac{1}{P(x)} = -\sum_x P(x) \log P(x)$$

If $\Pr(X = 1) = p$, then $\Pr(X = 0) = 1 - p$ and the entropy of $X$ (in shannons) is given by

$$\mathrm{H}(X) = \mathrm{H_b}(p) = -p \log_2 p - (1 - p) \log_2 (1 - p),$$




Entropy of a Bernoulli trial as a function of binary outcome probability, called the **binary entropy function**.

Ref: en.wikipedia.org

# Cross Entropy, KL Divergence

- Cross Entropy (CE)
  - For discrete distribution p and q with the same support X,

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

- KL Divergence
  - Measure how one probability distribution P(x) is different from a second (reference distribution) Q(x)

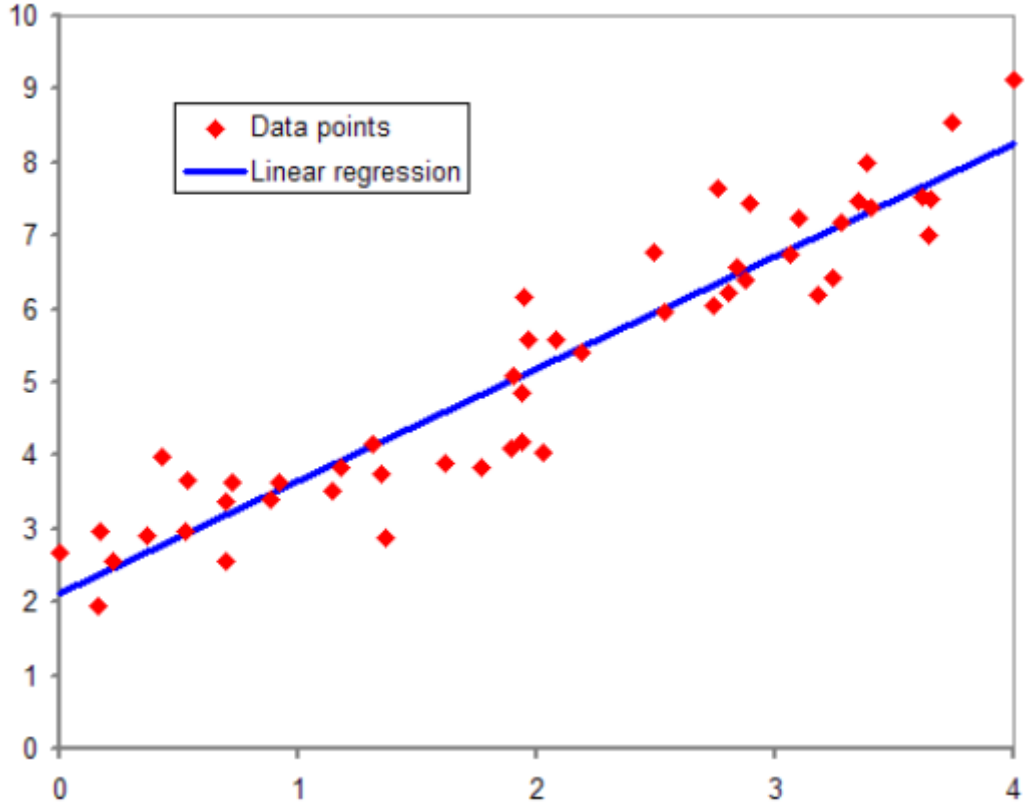
$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

$$\begin{aligned} H(p, q) &= H(p) + D_{KL}(p \parallel q) \\ &= -\sum_{i=0}^{n} p(x_i) log(p(x_i)) + \sum_{i=0}^{n} p(x_i) log(p(x_i)) - \sum_{i=0}^{n} p(x_i) log(q(x_i)) \\ &= -\sum_{i=0}^{n} p(x_i) log(q(x_i)) \end{aligned}$$

(*) If p, q perfectly match, $D_{KL}(p||q) = 0$, and the lower $D_{KL}$ is, the better match.

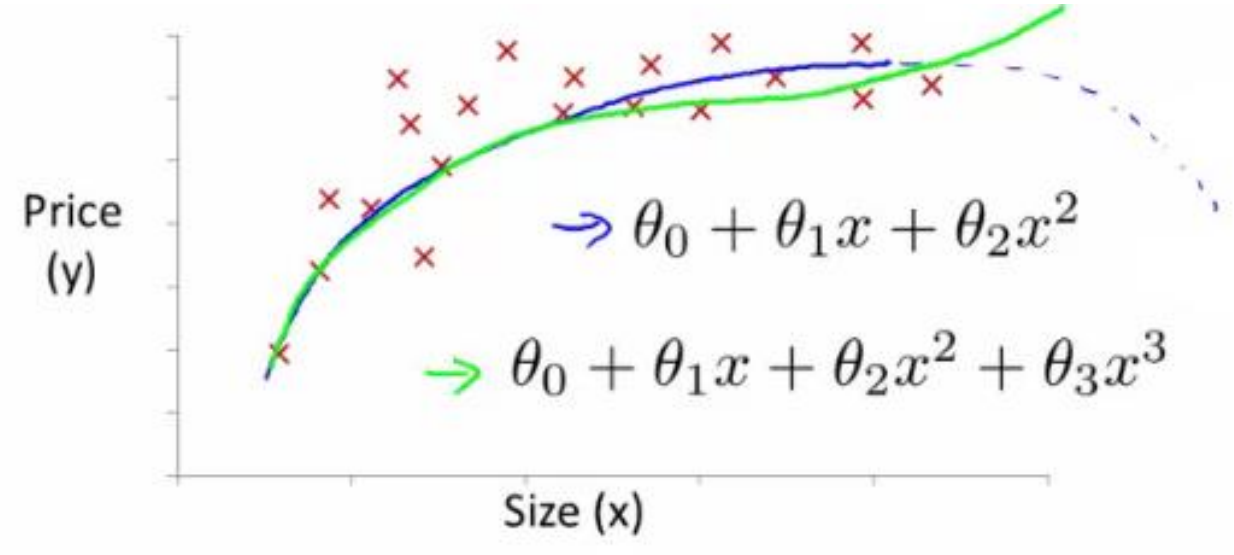
# Linear Regression

- **Linear regression**



# Linear Regression

- **Polynomial Regression**

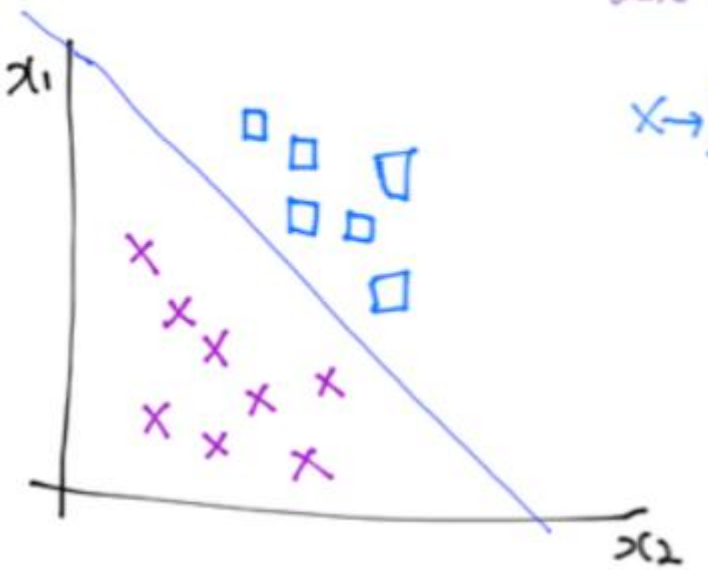


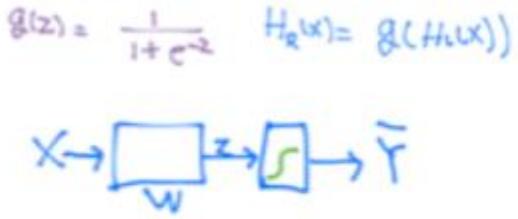

- To map our old linear hypothesis and cost functions to these polynomial descriptions the easy thing to do is set
  - $x_1 = x$
  - $x_2 = x^2$
  - $x_3 = x^3$
- By selecting the features like this and applying the linear regression algorithms you can do polynomial linear regression

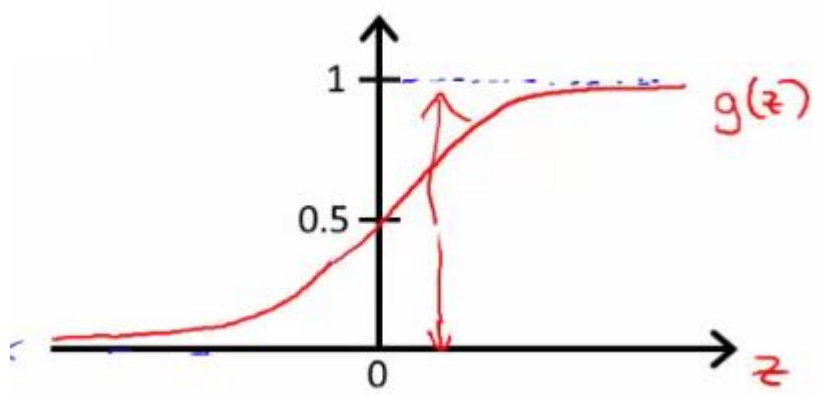
Ref: http://holehouse.org/mlclass/06_Logistic_Regression.html

# Linear Classification

- **Logistic Regression classifier (binary)**



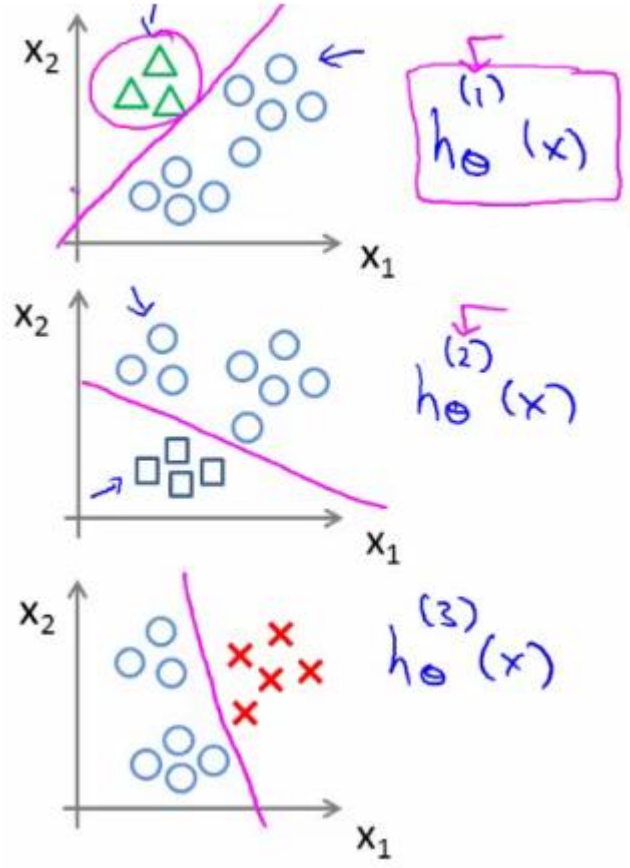



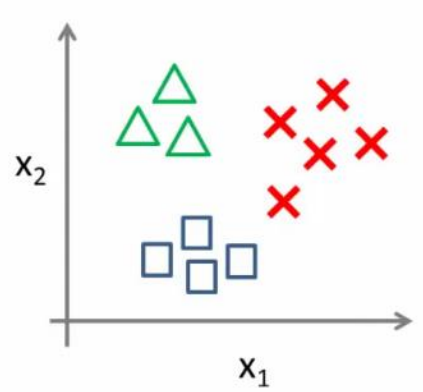

- Since this is a binary classification task we know y = 0 or 1
  - So the following must be true
    - $P(y=1|x;\theta) + P(y=0|x;\theta) = 1$
    - $P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$

Ref: http://holehouse.org/mlclass/06_Logistic_Regression.html

# Linear Classification

- **Multinomial (multi-class) classifier (One vs. Rest)**




- Split the training set into three separate binary classification problems
  - i.e. create a new fake training set
    - Triangle (1) vs crosses and squares (0) $h_\theta^1(x)$
      - $P(y=1 \mid x_1; \theta)$
    - Crosses (1) vs triangle and square (0) $h_\theta^2(x)$
      - $P(y=1 \mid x_2; \theta)$
    - Square (1) vs crosses and square (0) $h_\theta^3(x)$
      - $P(y=1 \mid x_3; \theta)$

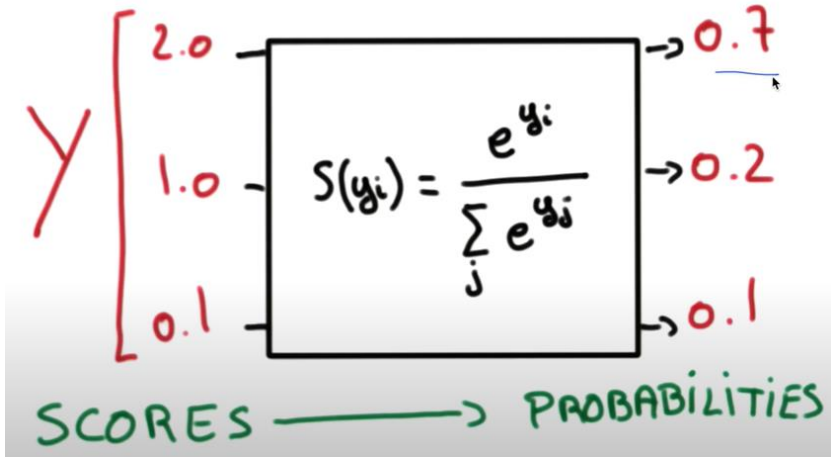
**Overall**

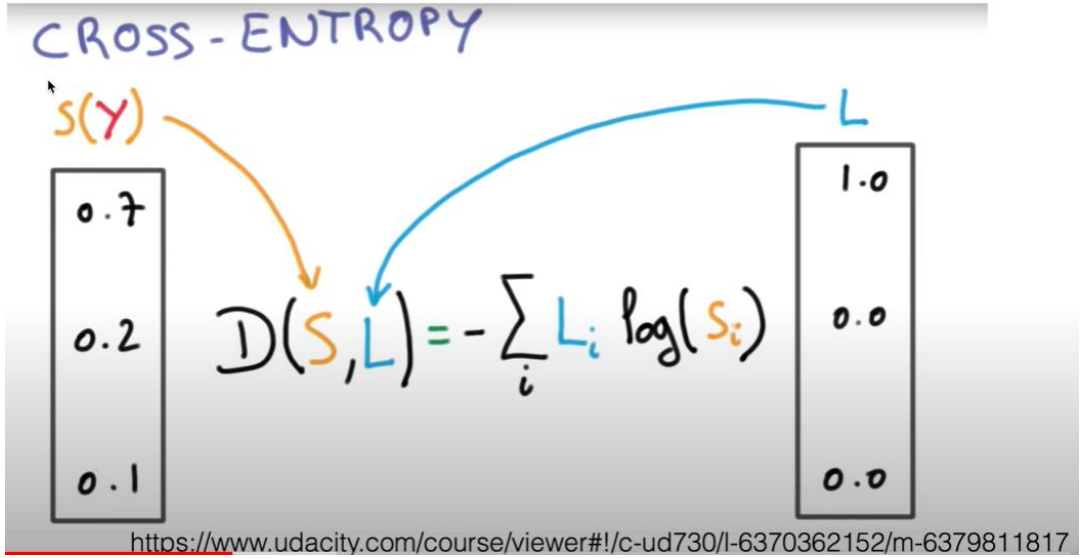
- Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that y = i
- On a new input, *x* to make a prediction, pick the class *i* that maximizes the probability that $h_\theta^{(i)}(x) = 1$

Ref: http://holehouse.org/mlclass/06_Logistic_Regression.html

# Linear Classification

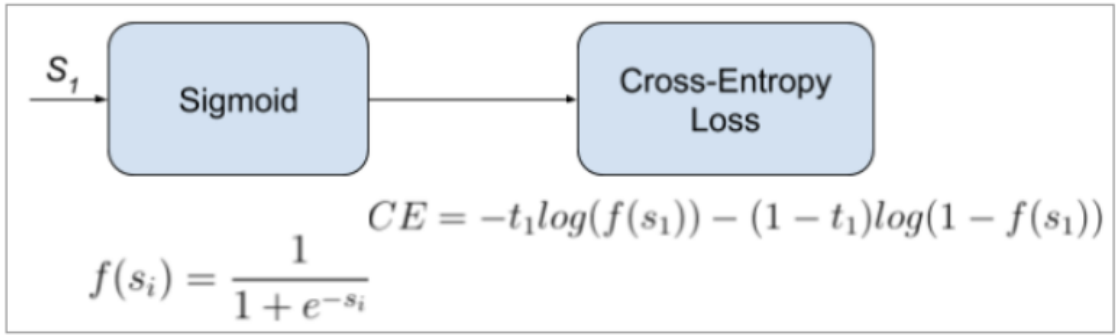
- **Softmax (regression) Classifier**
  - Also called: Multinomial Logistic, Maximum Entropy Classifier, Multi-class Logistic Regression
  - Generalization of logistic regression (assuming that the classes are mutually exclusive)
  - Categorical cross entropy



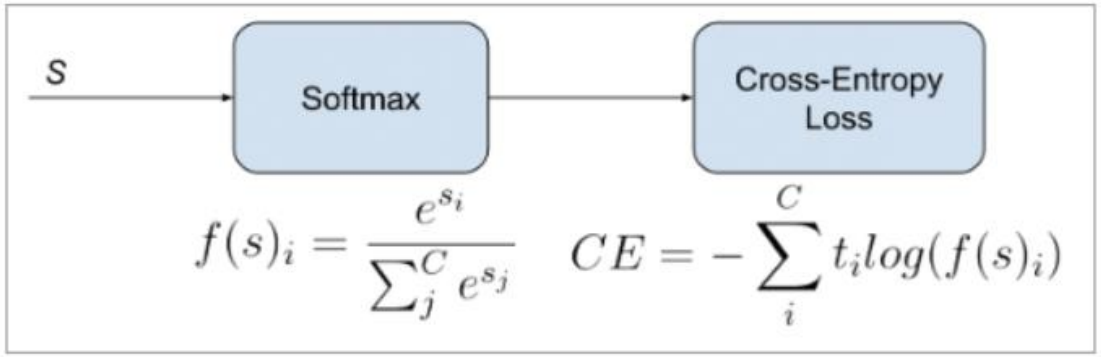



# Binary and multi-class classification

- **Binary CE**

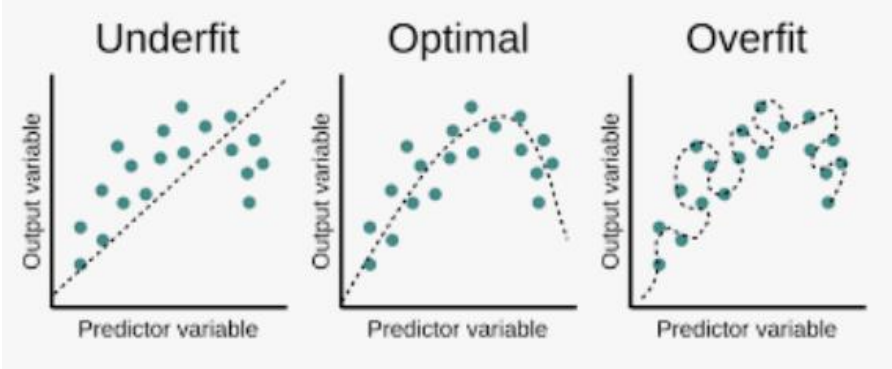


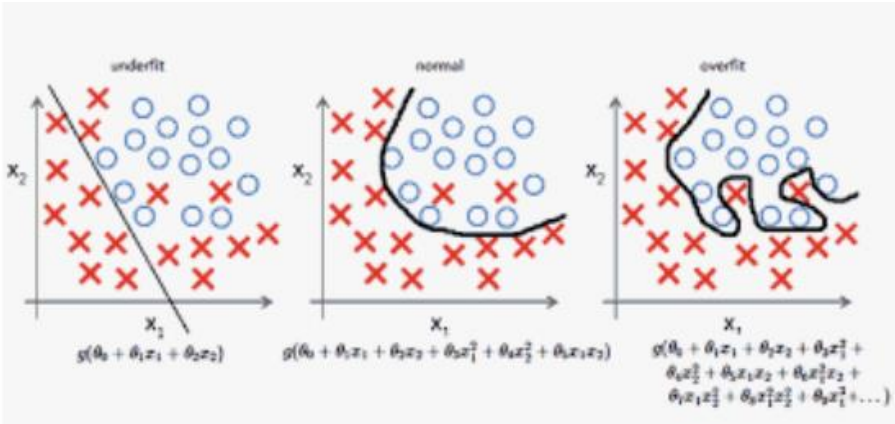

- **Categorical CE**



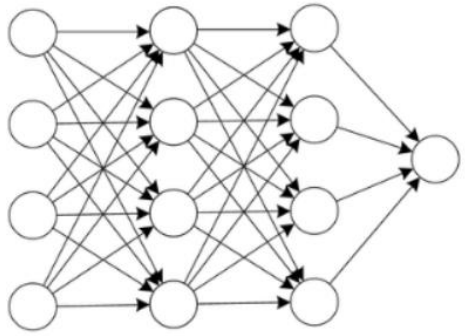
# Overfitting and Underfitting

- **Overfitting** and **Underfitting**

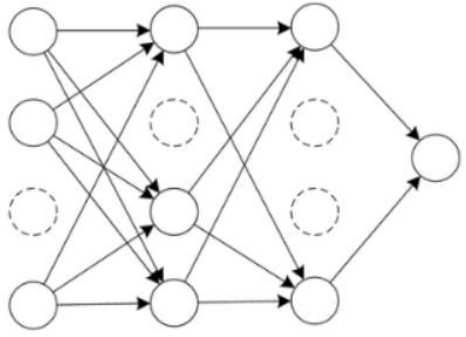



- **How to reduce overfitting?**
  - **More data**
  - **Simplify the model**
  - **Feature selection**
  - **Data augmentation**
  - **Regularization**
  - **Early stopping**
  - **Dropouts**



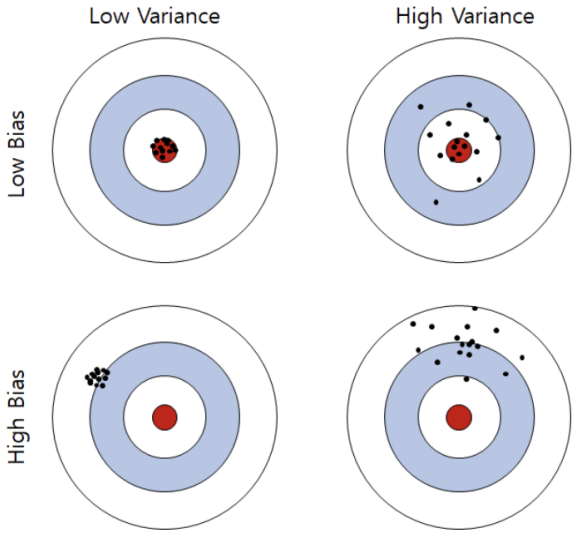
(a) Standard Neural Network

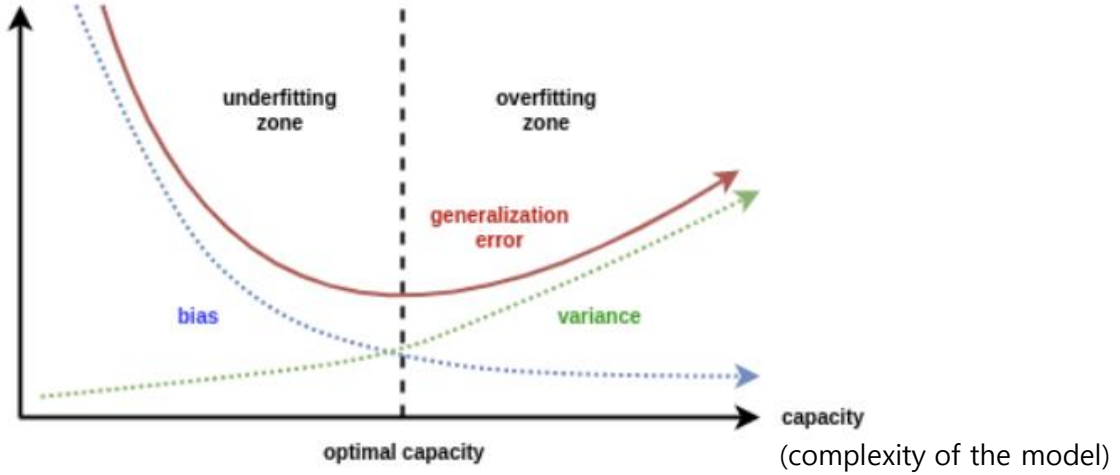


(b) Network after Dropout

# Bias and Variance

- **Bias-Variance tradeoff (편향과 분산)**
  - Bias: difference between the average prediction of the model and the correct value (wrong model -> high bias -> underfitting)
  - Variance: variability of model prediction (noisy dataset -> high variance -> overfitting)
  - total error = Bias + Variance + noise




Bias-Variance tradeoff

# Regularization

- **Regularization (규제화)**
  - Add information in objective function to reduce model complexity for reducing overfitting (regression and classification)
  - **Ridge** (L2): give more penalties on large-valued coefficients
  - **Lasso** (L1) (Least Absolute Shrinkage and Selection Operator): shrinks the less important features' coefficient to zero (good for **feature selection**)
  - **Elastic net**: use the both


$$J(W) = MSE(W) + \alpha \frac{1}{2} \sum_{i=1}^{n} W_i^2$$

$$J(W) = MSE(W) + \alpha \sum_{i=1}^{n} |W_i|$$

$$J(\theta) = MSE(\theta) + \gamma\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-\gamma}{2} \alpha \sum_{i=1}^{n} \theta_i^2$$

# Performance Metrics - Regression

- **MSE and MAE**
  - MSE(Mean squared error)
  - MAE(Mean absolute error)
  - MAE 가 이상치(outlier)에 대해서는 MSE 보다 robust 하다고 알려짐.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad \mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- **R Squared ($R^2$)**
  - 실제 출력 값에 대한 예측 출력 값 세트의 우수성 또는 적합성 표시
  - Numerator(MSE), denominator(variance)
  - What if all Yi is correctly predicted?
  - What if all Yi is predicted as the mean?

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{\hat{Y}}_i)^2}$$

# Performance Metrics - classification

- **Classification – static, dynamic**
- **Static: confusion matrix (a.k.a. Error matrix)**
  - Tabular visualization of the model prediction vs. ground-truth labels
  - Row: the instances in a predicted class
  - Column: the instances in an actual class

| n=165 | **Predicted: NO** | **Predicted: YES** | |
|---|---|---|---|
| **Actual: NO** | TN = 50 | FP = 10 | 60 |
| **Actual: YES** | FN = 5 | TP = 100 | 105 |
| | 55 | 110 | |

  - True-positive (100), false-negative (5)
  - true-negative (50), false-positive (10)

# Performance Metrics - classification

- **Accuracy**
  - Number of correct predictions divided by the total number of predictions
  - (ex) accuracy = (100+50)/165
- **Precision**
  - In some cases, accuracy is not a good indicator of your model performance, for example, when your class distribution is imbalanced.
  - Precision_yes = 100/110
  - Precision_noncat = 50/55
  - Says "**How reliable your prediction is…**" or "how much I can trust…"
- **Recall**
  - Recall_yes = 100/105
  - Recall_no = 50/60
  - Says "**how many actual class samples are correctly predicted**…"
- **F1-Score**
  - Combine the above two metrics (precision and recall) as harmonic mean:
    F1-score = 2*Precision*Recall / (Precision+Recall)

# Performance Metrics - classification

- **Dynamic: ROC Curve (Receiver Operating Characteristic curve)**
  - Shows the performance of a binary classifier as function of its cut-off threshold
  - It essentially shows the true positive rate(tpr) against the false positive rate(fpr) for various threshold values.
- **As an example,**
  - Suppose your model predicts 4 sample images with probabilities [0.45, 0.6, 0.7, 0.3].
  - Then, depending on the threshold, you will get different labels:

    cut_off=0.5: predicted_value=[0,1,1,0] (default threshold)
    cut_off=0.2: predicted_value=[1,1,1,1]
    cut_off=0.8: predicted_value=[0,0,0,0]

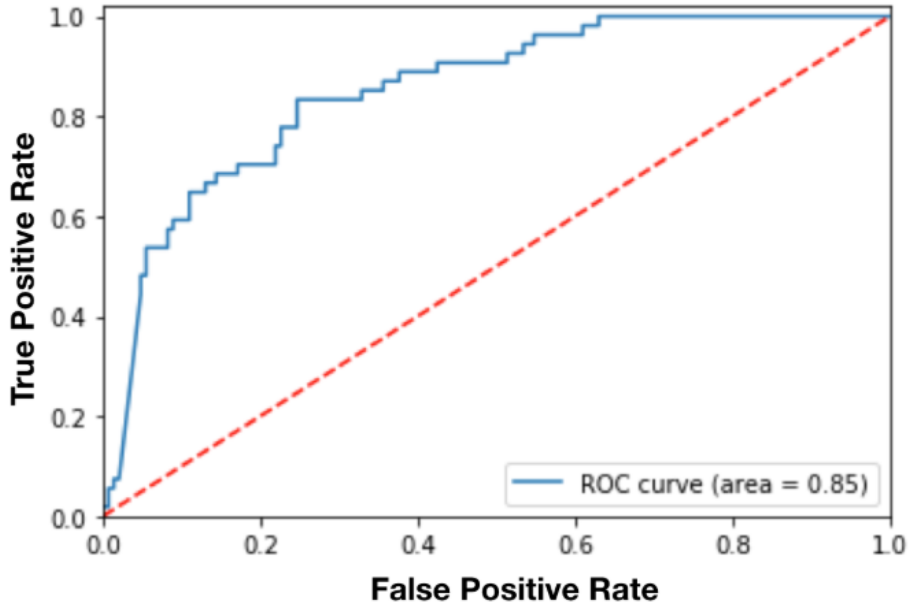
  - Making different confusion matrix depending on the threshold.

# Performance Metrics (example)

| 환자번호 | 성별 | 점수 | 순위 | 실제 값 | | |
|---|---|---|---|---|---|---|
| 7 | F | 0.98 | 1 | N (0) | FP | |
| 125 | M | 0.96 | 2 | C (1) | TP | |
| 4 | F | 0.95 | 3 | N | FP | |
| 199 | M | 0.86 | 4 | C | TP | |
| 2 | F | 0.84 | 5 | N | FP | |
| 200 | M | 0.82 | 6 | C | TP | |
| 176 | M | 0.81 | 7 | C | TP | |
| 73 | M | 0.80 | 8 | N | FP | |
| 82 | M | 0.79 | 9 | C | FN | TP |
| 3 | F | 0.77 | 10 | N | TN | FP |
| 123 | F | 0.76 | 11 | N | TN | FP |
| | | ... | | C | FN | TP |
| 43 | F | 0.48 | 198 | N | TN | FP |
| 93 | M | 0.42 | 199 | N | TN | |
| 120 | F | 0.40 | 200 | N | TN | |

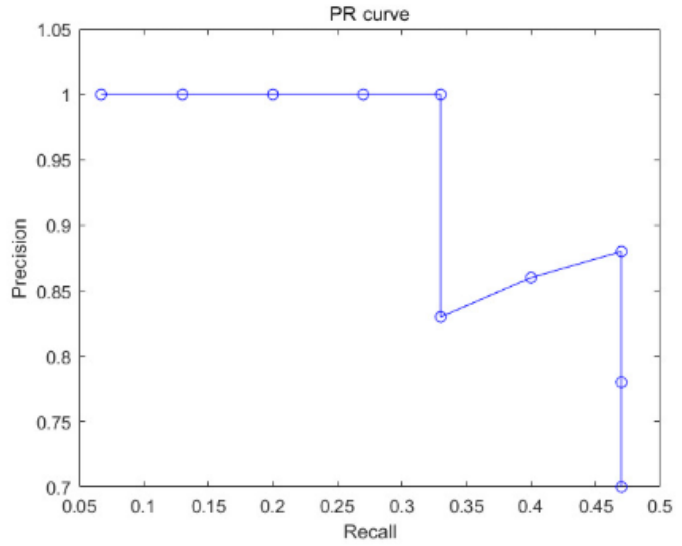
conservative

(*) score : probability of cancer
(*) depending on the threshold, you will get different confusion matrix

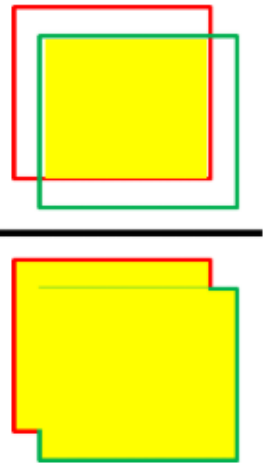


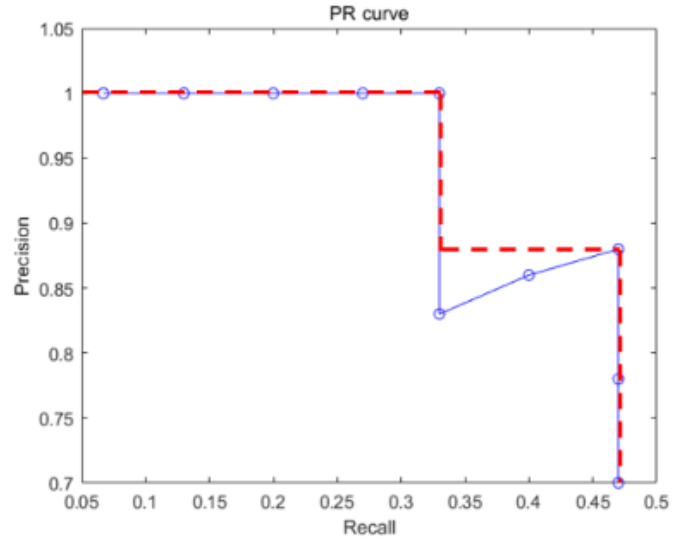
# Performance Metrics (IoU and mAP)

- **Object detection and Image Classification**
  - IOU (Intersection over Union)
  - Average Precision (AP): average of precisions for Recall=0.1,0.2,…,1.0 (빨간선 아래 면적)
  - mAP (mean Average Precision) : mean of AP's of multiple objects

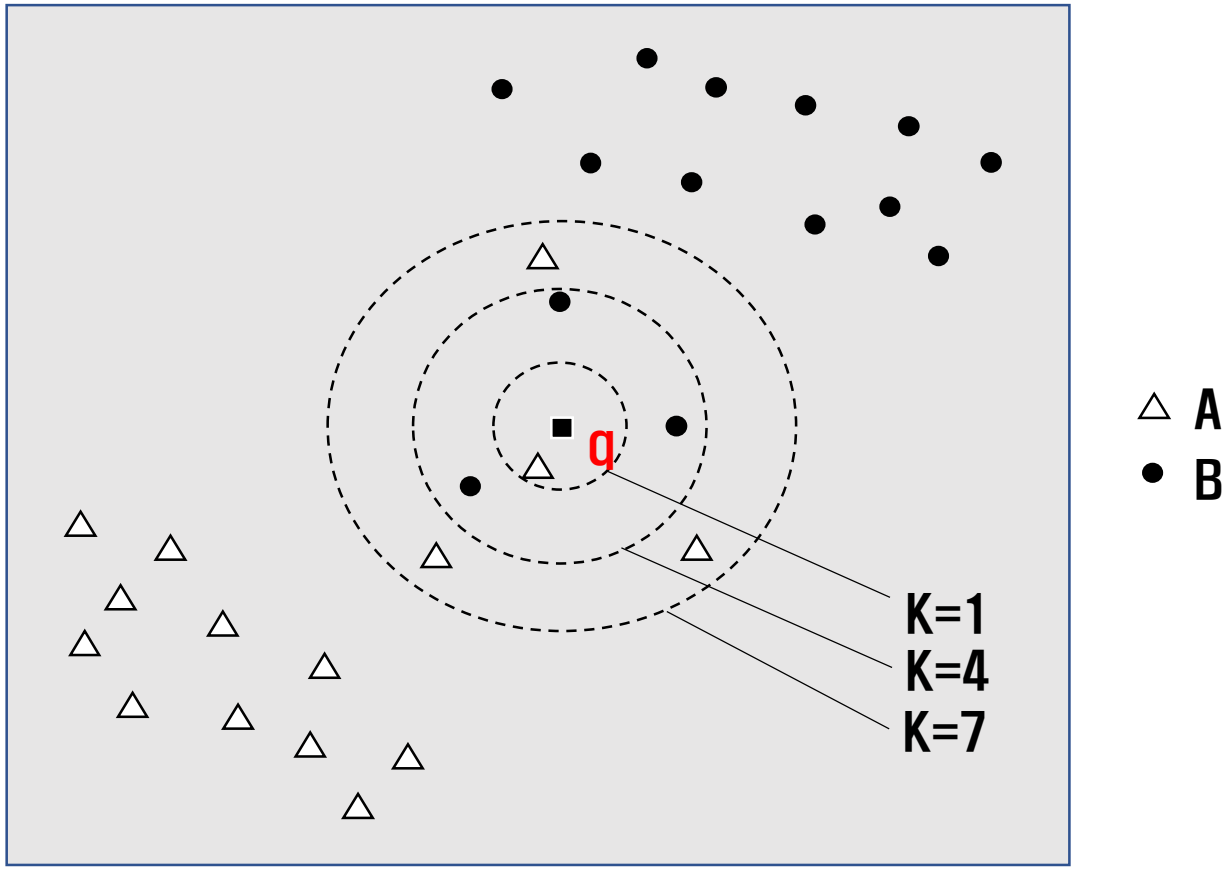



$$IoU = \frac{area(B_{gt} \cap B_p)}{area(B_{gt} \cup B_p)} =$$





# Knn (K-Nearest Neighbor)



# Knn (K-Nearest Neighbor)

- kNN의 장점
  - 훈련시간이 거의 없다
  - 알고리즘의 개념이 명확
  - 모델링에 필요한 하이퍼 파라미터 : k 값 하나뿐
  - 특성 변수만 잘 선정하면 예측 성능도 좋다
- kNN의 단점
  - 분류를 처리하는 시간, 즉 알고리즘을 수행하는 시간이 길다
  - 확보한 샘플들을 모두 비교해서 어떤 그룹에 가까운 지 새로 계산해야
  - 또한 새로운 샘플이 계속 추가될 때마다 가까운 이웃이 달라진다
  - Lazy 알고리즘
    - 나중에 계산량이 많은 알고리즘
- 샘플들간의 거리에 대한 가중치를 고려
  - 가까이 있는 이웃에 대해서는 가중치를 크게

# Decision Trees

- **Decision Tree**
  - Handles each feature independently
  - Both Regression and Classification
  - Use Classification And Regression Tree (CART) algorithm to train Decision Trees
- **CART algorithm**
  - first splits the training set in two subsets using a single feature $k$ and a threshold $t_k$
  - How to choose it? It searches for the pair ($k$, $t_k$) for purest subsets (greedy)
  - It splits the subsets using the same logic recursively.

*Equation 6-2. CART cost function for classification*

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

where $\begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$

# Decision Trees

- 결정트리는 나누려는 그룹의 순도가 가장 높아지도록 그룹을 나누어야 한다.
- 불순도(Impurity): 해당 범주 안에 서로 다른 데이터가 얼마나 섞여 있는지의 정도
- 그룹의 순도를 표현
  - 지니(Gini) 계수

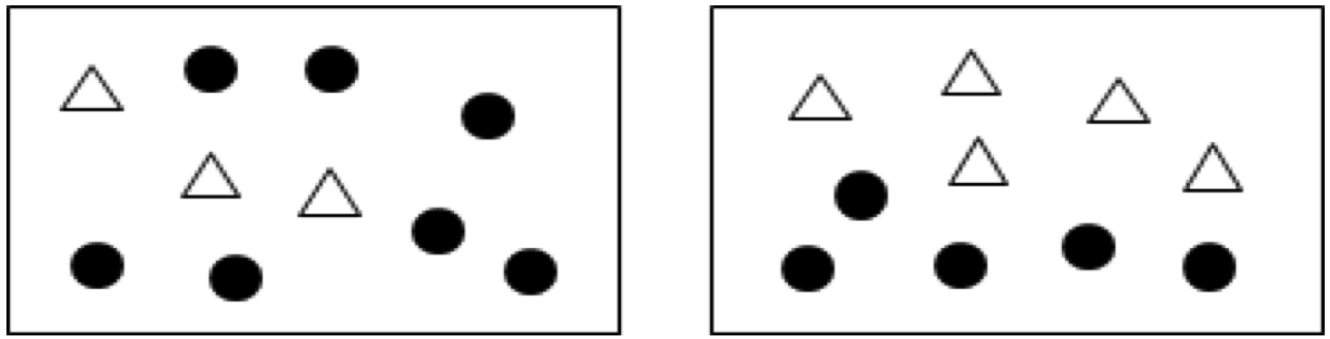
$$Gini = 1 - \sum_{k=1}^{m} p_k^2$$

  - 엔트로피(entrophy)

$$Entropy = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

  - 성능은 비슷하지만 연산 속도는 Gini 가 빠름 (log 연산 없음)

# Decision Trees



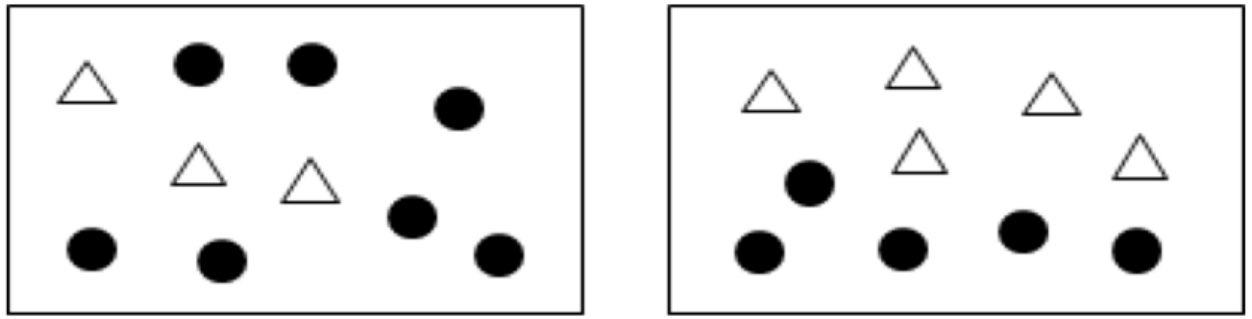
좌측 박스: $\text{지니}(7:3) = 1 - \left[\left(\frac{7}{10}\right)^2 + \left(\frac{3}{10}\right)^2\right] = 1 - (0.49 + 0.09) = 0.42$

---

우측 박스: $\text{지니}(5:5) = 1 - \left[\left(\frac{5}{10}\right)^2 + \left(\frac{5}{10}\right)^2\right] = 1 - (0.25 + 0.25) = 0.5$

(*) worst: 0.5
Best: Gini=0 (all in one class)

# Decision Trees

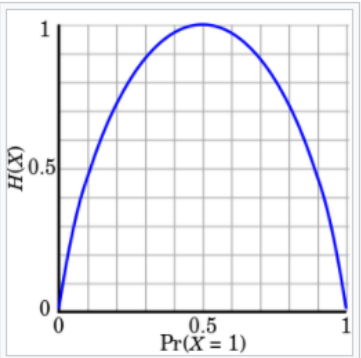


$$Entropy = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

$$\text{좌측}(7:3) = -\left[0.7^*\log_2(0.7) + 0.3^*\log_2(0.3)\right]$$
$$= -\left[(-0.36) + (-0.52)\right] = -(-0.88) = 0.88$$

$$\text{우측}(5:5) = -\left[0.5^*\log_2(0.5) + 0.5^*\log_2(0.5)\right]$$
$$= -\left[(-0.5) + (-0.5)\right] = -(-1) = 1$$

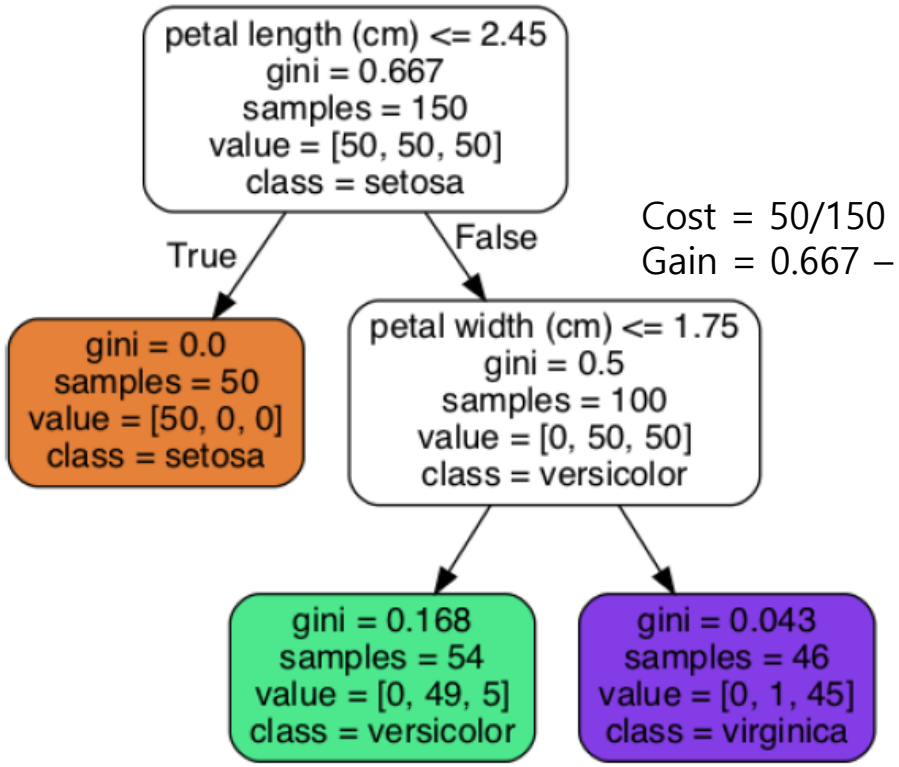
(*) worst: 1




Entropy of a Bernoulli trial as a function of binary outcome probability, called the **binary entropy function**.

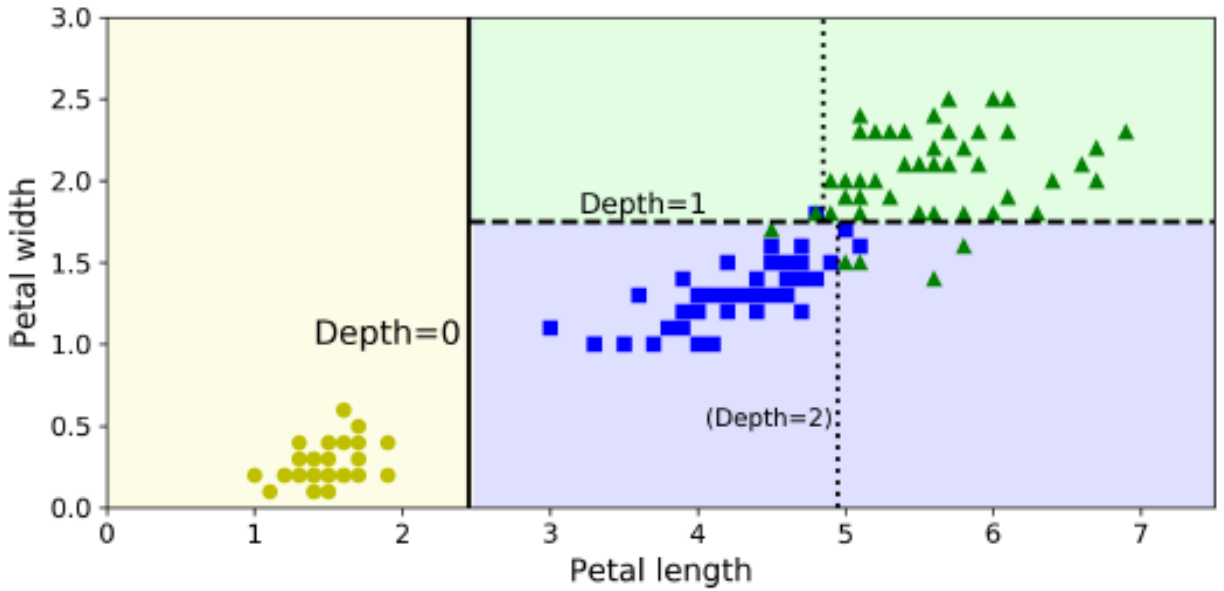
# Decision Trees

Gini = 1 – $[(1/3)^2 + (1/3)^2 + (1/3)^2]$
= 0.667




Cost = 50/150 * 0 + **100/150 * 0.5** = 0.333
Gain = 0.667 – 0.333 = 0.334

Cost = 54/100 * 0.168 + 46/100 * 0.043 = 0.11
Gain = 0.5 – 0.11 = 0.39



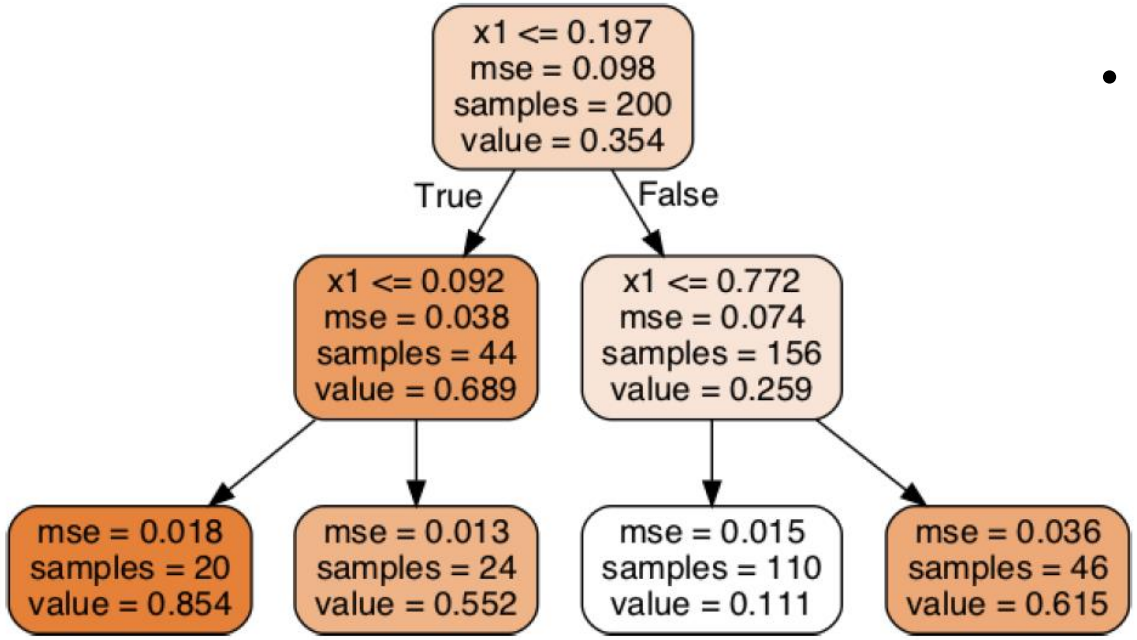
# Decision Trees

- Hyper-Paramters
  - max_depth: 트리의 최대 깊이 (이보다 깊은 트리를 만들지 않는다)
  - max_leaf_nodes: 리프 노드의 최대 수 (리프 노드를 이보다 많이 만들지 않는다)
  - max_samples_split: 분할하기 위한 최소 샘플수 (이보다 작으면 분할하지 않는다)
  - min_samples_leaf: 리프 노드에 포함될 최소 샘플수 (이보다 작은 노드는 만들지 않는다)
  - max_features: 최대 특성수 (분할할 때 이보다 적은 수의 특성만 사용한다)
- Internal Parameters
  - 결정 트리 모델을 만든 후에, 어떤 특성이 결정 트리를 생성할 때 중요한 역할을 했는지 비중을 파악 가능
  - 이 결과를 보고 중요하지 않은 특성은 향후에 제외하기도 함
  - 내부 변수로 확인
    - feature_importances_

# Decision Trees

- **Computational complexity**
  - Prediction: traversing the Decision Tree from the root to a leaf
  - Training: compares all features on all samples at each node
  - Finding the optimal tree is known to be NP-complete (O(exp(m)).

# Decision Tree Regression

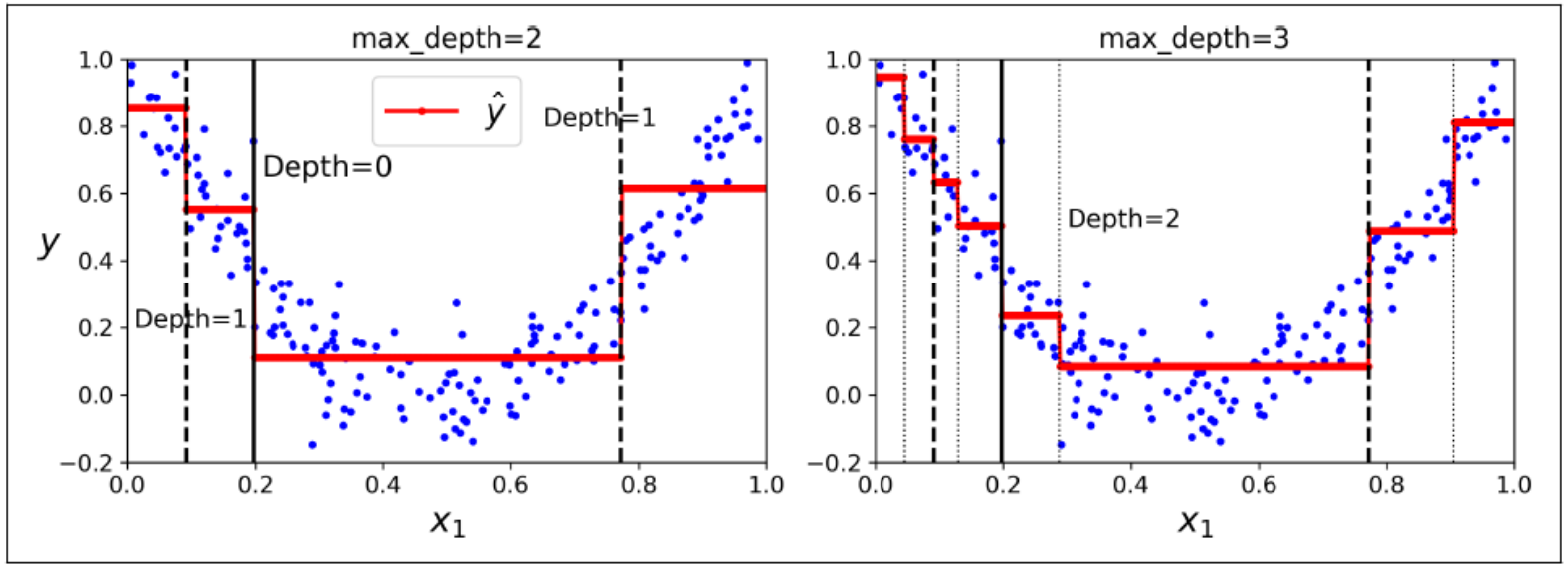



- It now tries to split the training set in a way that minimizes the MSE. (instead of minimizing impurity)

*Equation 6-4. CART cost function for regression*

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

# Decision Tree Regression

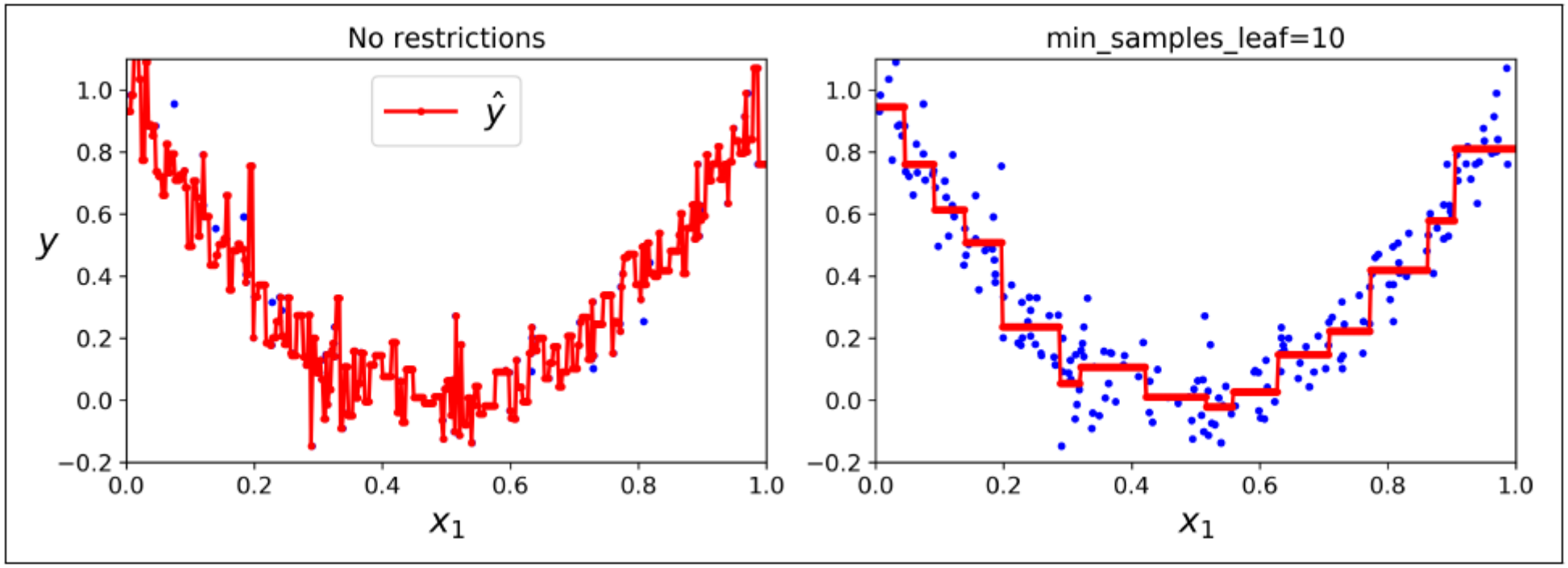



*Equation 6-4. CART cost function for regression*

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

# Decision Tree Regression

- **Regularizing the Model**



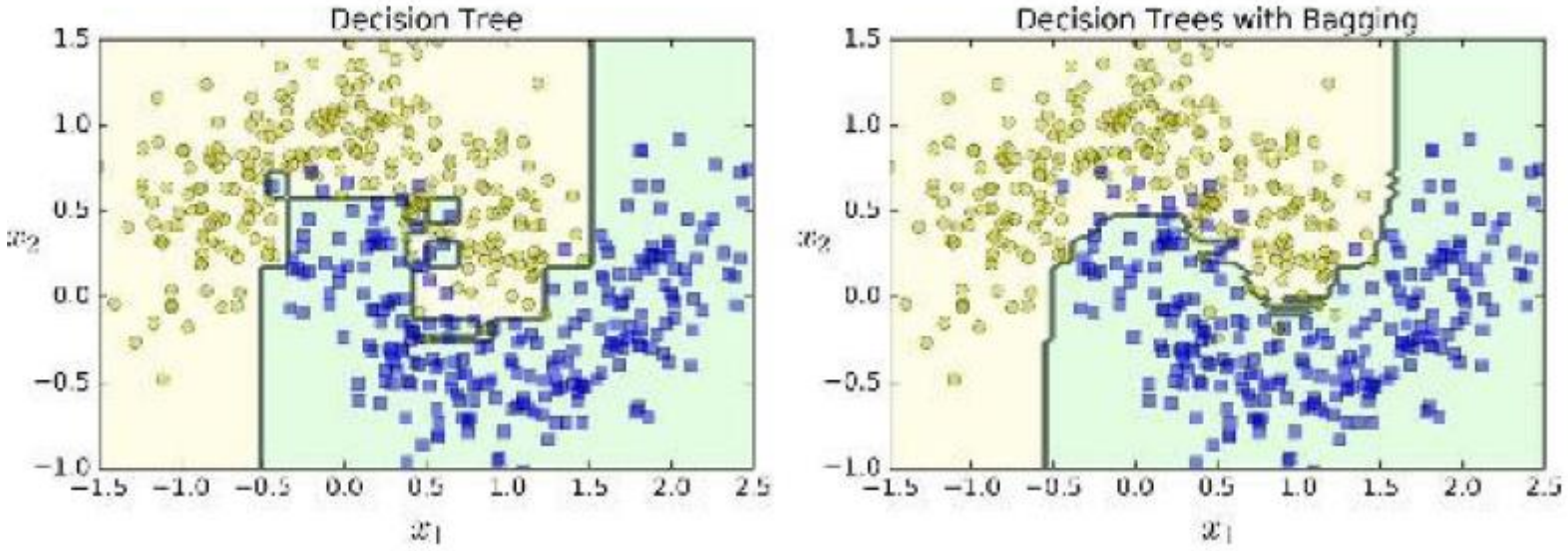
# Ensemble Models

- **Ensemble method:**
  - to aggregate the predictions of each classifier and predict the class that gets the most votes (Hard Voting: 직접투표)
  - to predict the class with the highest class probability, averaged over all the individual classifiers (Soft Voting: 간접투표 )

| | P일 확률 | Q일 확률 | 판정결과 (Hard Voting) |
|---|---|---|---|
| 세부 모델 A | 0.9 | 0.1 | P |
| 세부 모델 B | 0.4 | 0.6 | Q |
| 세부 모델 C | 0.3 | 0.7 | Q |
| 확률의 평균 (Soft Voting) | (1.6)/3 = 0.533 | (1.4)/3 = 0.456 | P or Q |

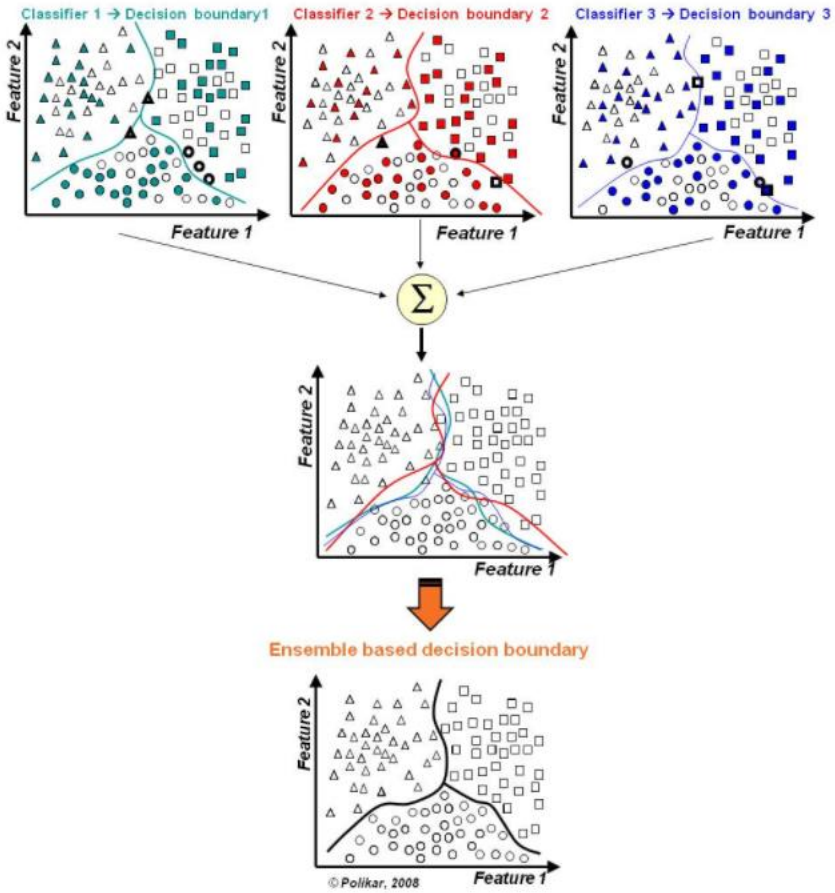
# Ensemble Models

- **Decision Tree and Decision Tree with Bagging**
  - Moon dataset with 500 decision trees
  - The ensemble has a comparable bias, but a smaller variance. (roughly the same number of errors on the training set, but the decision boundary is less irregular)



# Ensemble Models

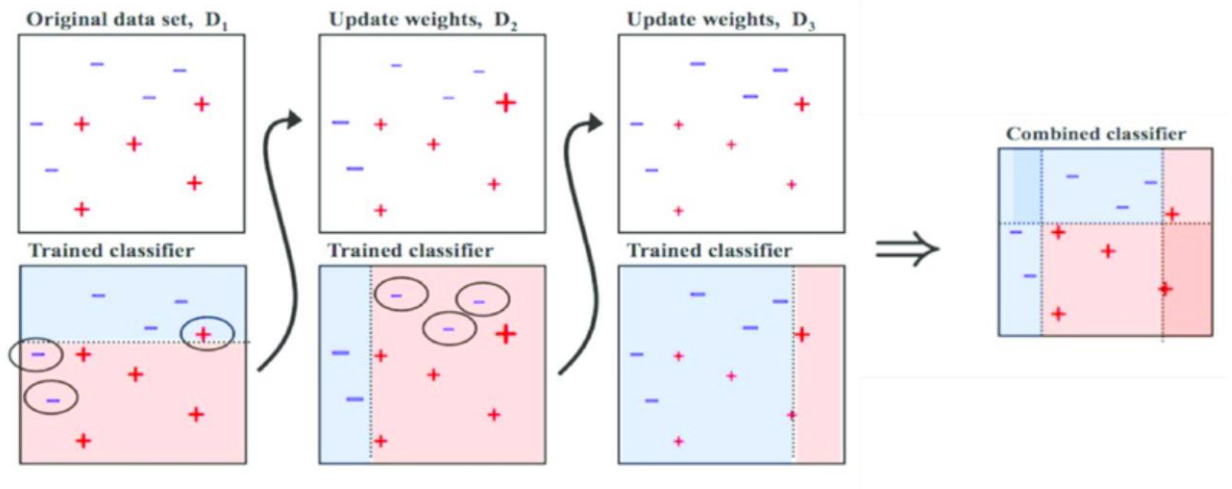
- **Bagging: 중복을 허용**
  - 중복을 허용한 n 개의 샘플을 추출하여 평균을 구하는 작업을 m 번 반복.
  - Overfitting 을 효율적으로 줄이고 일반적인 모델을 만드는데 집중.
  - (ex) RandomForest




출처: https://swalloow.github.io/bagging-boosting

# Ensemble Models

- **Boosting:**
  - Bagging은 독립적인 input data를 가지고(복원 추출) 독립적으로 예측하지만, 부스팅은 **이전 모델이 다음 모델에 영향을 준다. (틀린 부분에 더 큰 가중치)**
  - Bagging과 다르게 일반적인 모델에 집중되어 있지 않고, 맞추기 어려운 문제를 맞추는데 초점
  - (ex) XGBoost, AdaBoost, GradientBoost




출처: Medium (Boosting and Bagging explained with examples)

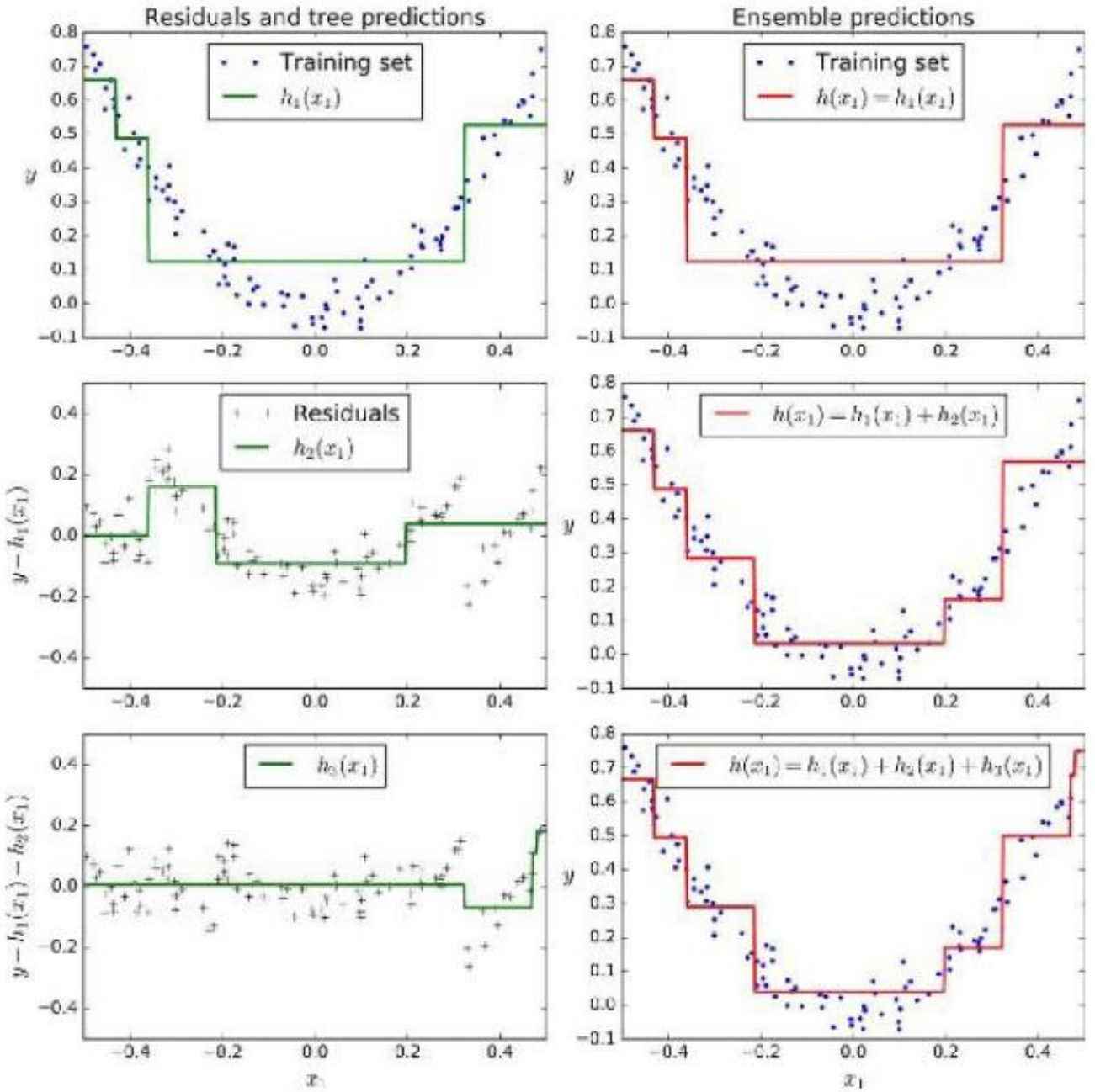
# Ensemble Models

- **Adaboost classifier**
  - pays a bit more attention to the training instances that the predecessor underfitted.
  - This results in new predictors focusing more and more on the hard cases.
  - the new predictor's weight is computed, the instance weights are updated, then another predictor is trained, and so on

1. Initialize the obervation weights $w_i = 1/N$
2. For $m = 1, 2, \ldots, M$
   - Compute the weighted error $Err_m = \frac{\sum_{i-1}^{N} w_i \mathcal{I}(y^{(i)} \neq G_m(x^{(i)}))}{\sum_{i=1}^{N} w_i}$ ← Weights on the error (loss)
   - Compute coefficient $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$ ← Contribution of each G(x) (the more accurate, the higher influence)
   - Set data weights $w_i \leftarrow w_i \exp[\alpha_m \mathcal{I}(y^{(i)} \neq G_m(x^{(i)}))]$ ← Higher weight on misclassified
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$ ← Weighted majority vote

# Ensemble Models

- **Gradient Boosting Regression Trees (GBRT)**
  - Just like Adaboost, it sequentially adds predictors to an ensemble, each one correcting its predecessor
  - (instead of tweaking the instance weights) It tries to fit the new predictor to the *residual errors* made by the previous predictor.
  - Different residual calculation for classification



# Ensemble Models

- **배깅**(bootstrap aggregation)
  - **전체 훈련 데이터에서 "중복을 허용" 하여 데이터를 샘플링을 하는 방법**
  - bootstrap resampling**의 줄임말로 부트 스트래핑이라고도 함**
  - **목적 : 부족한 훈련 데이터를 효과적으로 늘리기 위함**
- **페이스팅**(pasting)
  - **배깅과 달리 주어진 원래 데이터에서 중복을 허용하지 않고, 즉, 한 번 샘플링 된 것은 다음 샘플링에서 제외하는 방식**
- **배깅을 수행하면 학습에 선택되지 않는 샘플은 평균 37% 정도**
  - **이 샘플을** oob(out of bag) **샘플이라고 함**
  - **이** oob **데이터는 훈련에 사용되지 않았으므로 검증에 사용하기에 좋다**
- **RandomForest:**
  - **결정 트리 구조에 배깅을 적용한 방식**

# Clustering

- 유사도가 큰 항목끼리 묶음
  - 비정상 패턴 (Outlier) 식별에도 사용 (컴퓨터시스템 해커 등)
- Similarity (유사도)
- Need Scaling as a preprocessing step
- K-Means()
- Agglomerative Clustering
- DBSCAN
- Many more …

# Clustering

- **Similarity** (유사도) and **Distance** (거리)
  - 항목 간의 유사한 정도를 수치로 표현
  - 유사도 결과에 따라 데이터 분석 결과가 달라짐
  - 분석 경험과 도메인에 대한 이해 필요함
  - 최적의 분석 결과가 나오도록 유사도를 변경해 가면서 반복 수행 필요함
  - 유사도 s(similarity)는 $0 \leq s \leq 1$ (1에 가까울수록 유사도 높음)
  - 유사도의 상대 개념으로 거리(distance) 사용
    - 유사도와 거리의 관계: $d = 1 - s$
  - (ex) A,B,C 중 누가 서로 가까운지?
    - 보통 상대적인 차이 사용 (Z-변환 or standard Scaling)

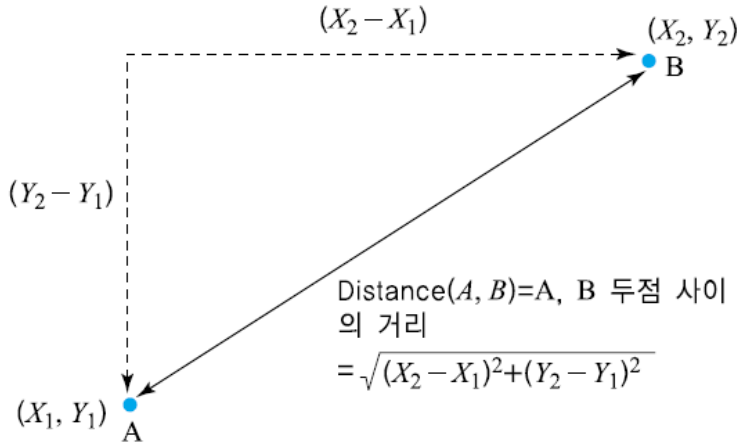
| 구분 | 키 | 몸무게 | 나이 |
|---|---|---|---|
| A | 174cm | 70kg | 21세 |
| B | 170cm | 61kg | 27세 |
| C | 162cm | 73kg | 29세 |

# Clustering

- **Similarity (유사도) and Distance (거리)**
  - **Euclidian distance**

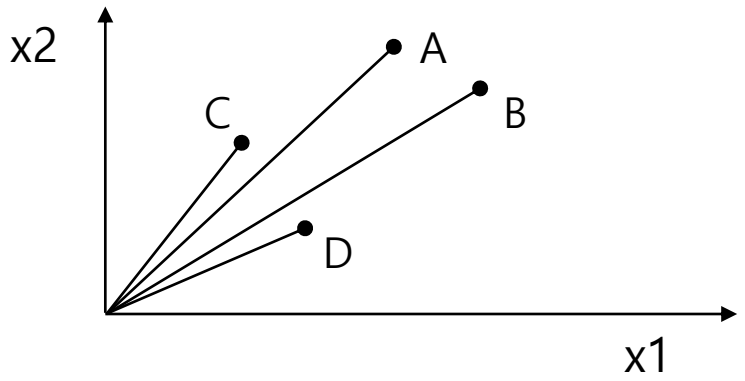
**N-차원**

$$\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2}=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}$$




  - **Cosine distance – 방향성, 취향**

$$s_{\cos}(x,y)=\frac{X \cdot Y}{|X||Y|}$$



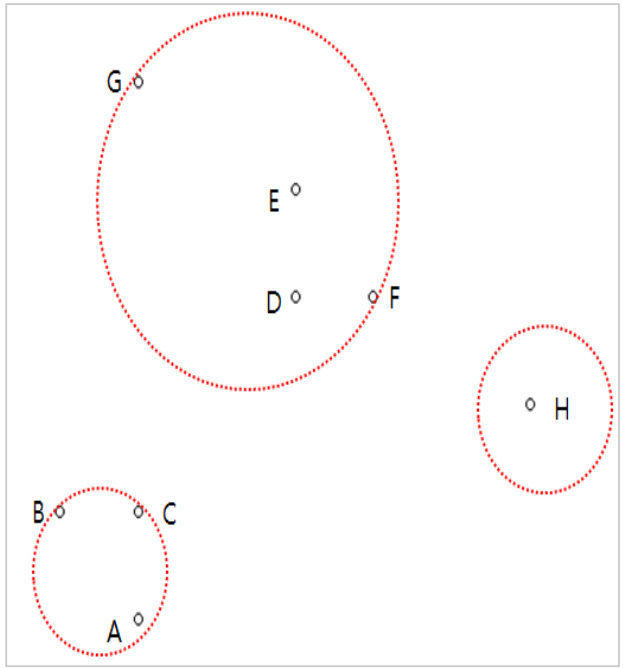
# Clustering

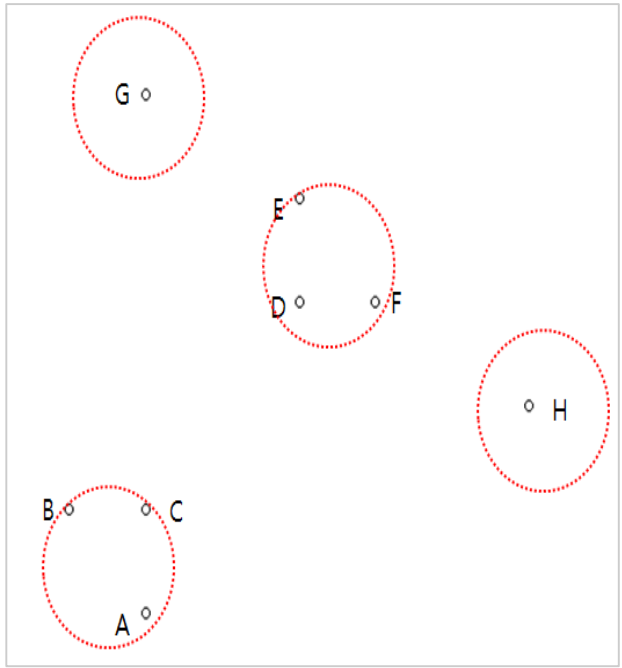
- Jaccard similarity (자카드 유사도)
  - 비슷한 취향의 사람을 찾을 때 사용 - 영화, 도서, 음악 추천 등
  - 영화 보는 취향에 따른 유사도 측정
  - (ex) 지난 1년 동안 국내에 개봉된 영화가 500편
    - A와 B가 본 영화 중 겹치는 영화가 5편, 5/500 = 0.01
    - A와 C가 본 영화 중 겹치는 영화가 10편, 10/500 = 0.02
    - 즉, 0.01 < 0.02이므로 A와 C가 더 가깝다고 할 수 있음
    - 위와 같은 계산 방법이 적절한가?
  - A, B, C가 각각 지난해 본 영화의 총 개수가 20편, 50편, 200편이라면?
    - J(A,B) = 5 / (20+50-5) = 0.076
    - J(A,C) = 10 / (20+200-10) = 0.047
    - 즉, 0.076 > 0.047이므로 A와 B가 더 가깝다고 할 수 있음

$$S_{Jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

# Clustering (군집화)

- **Number of clusters, *k* (important hyper-parameter)**
  - **적당한 군집의 수 (k) 를 먼저 찾아야 함**

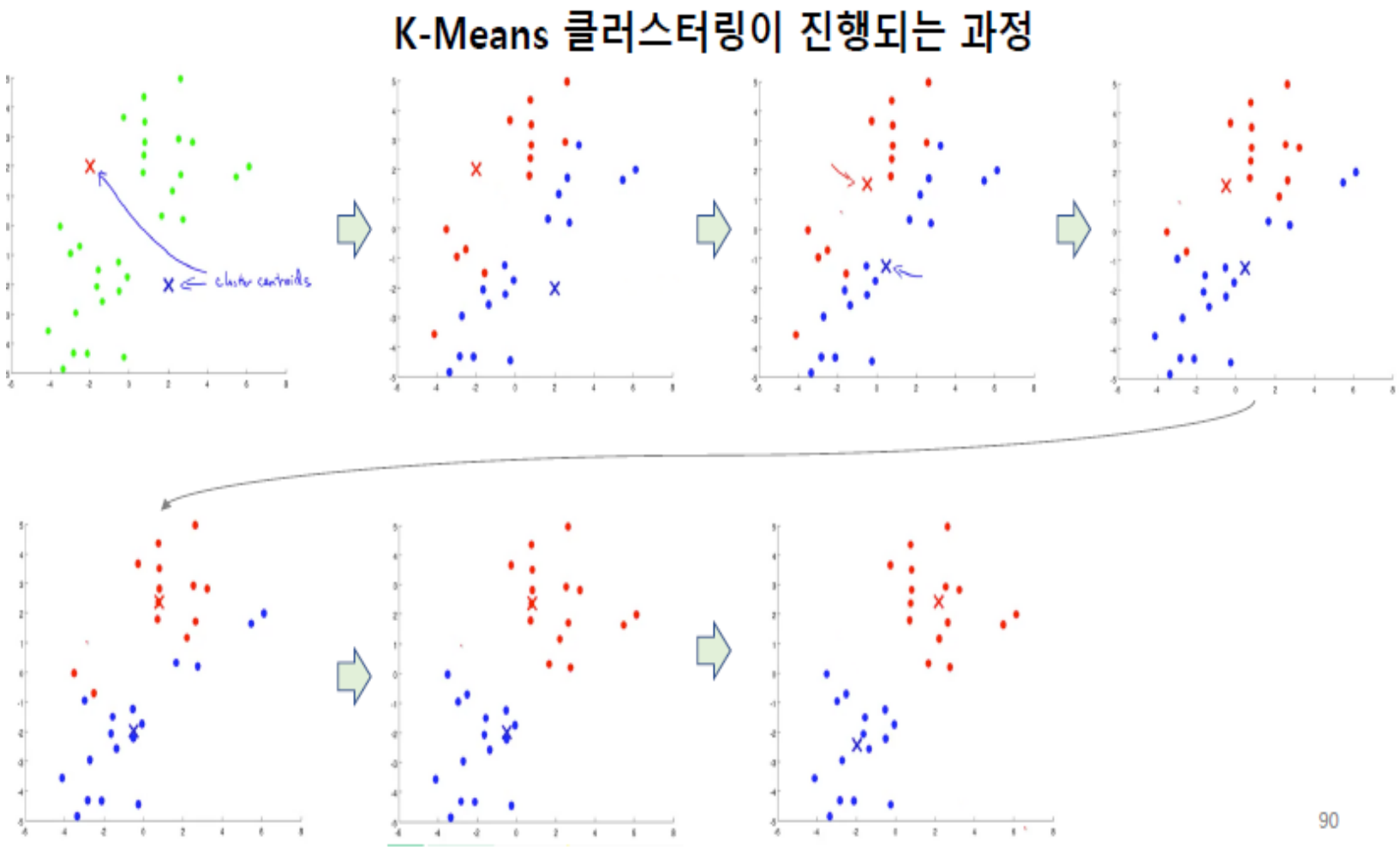


# Clustering (군집화)

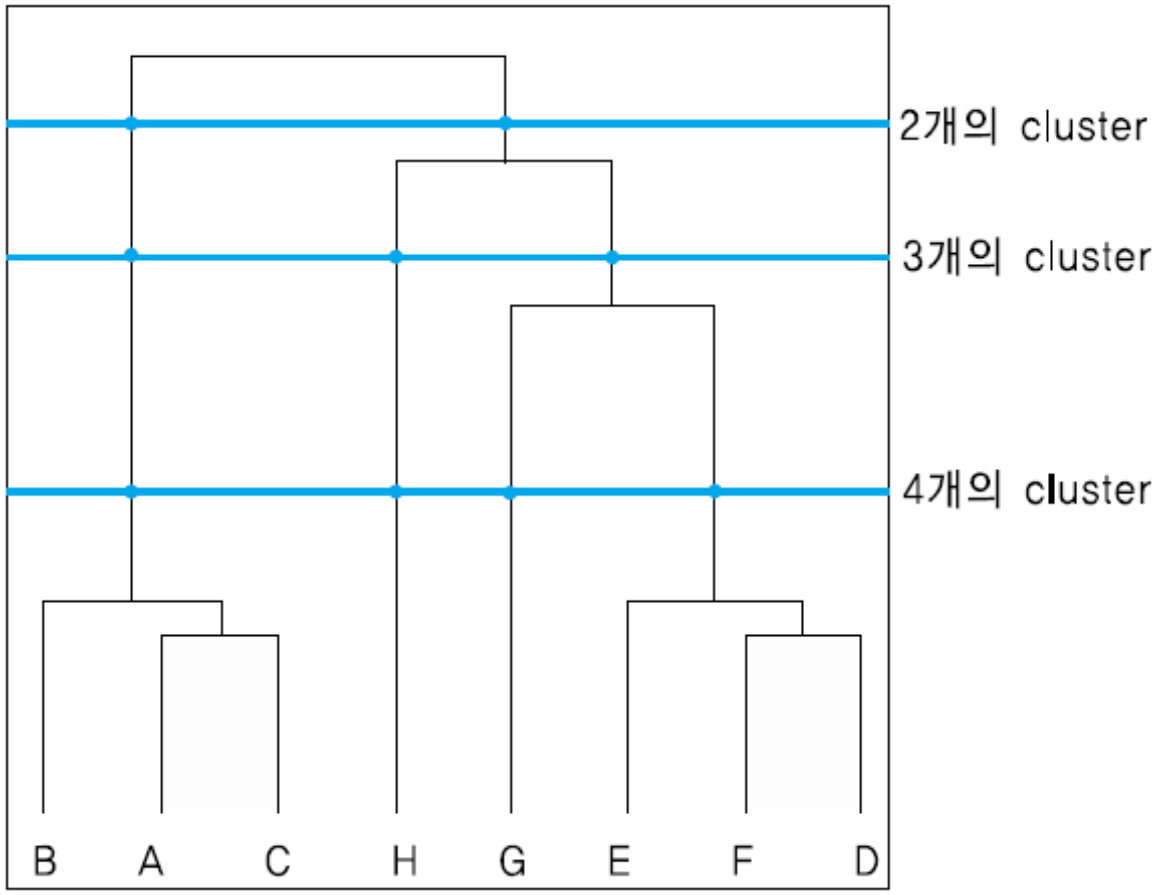
- K-means()
  - 공간상에 임의의 k 개의 임의의 초기 지점을 클러스터 중점으로(cluster center) 정함
  - 클러스터 중점을 중심으로 거리가 가까운 항목을 선택하여 클러스터 공간을 나눔
  - 각 클러스터에 포함된 항목들의 평균 위치를 구해 이를 새로운 클러스터 중점(centroid)으로 변경
  - 새로 설정된 센트로이드를 중심으로 경계를 다시 그림 (각 항목들이 소속된 클러스터가 바뀔 수 있음)
  - 변경된 항목들을 가지고 클러스터 중심을 다시 계산
  - 더 이상 클러스터의 모양이 바뀌지 않을 때까지 반복 수행
    - KMeans() 사용

# Clustering (군집화)



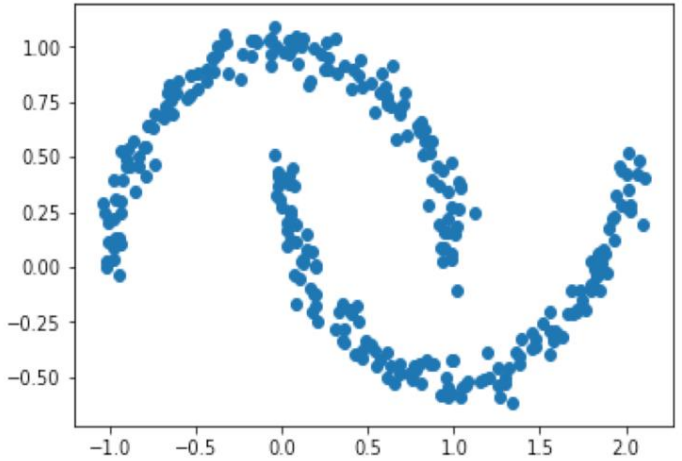
# Clustering (군집화)

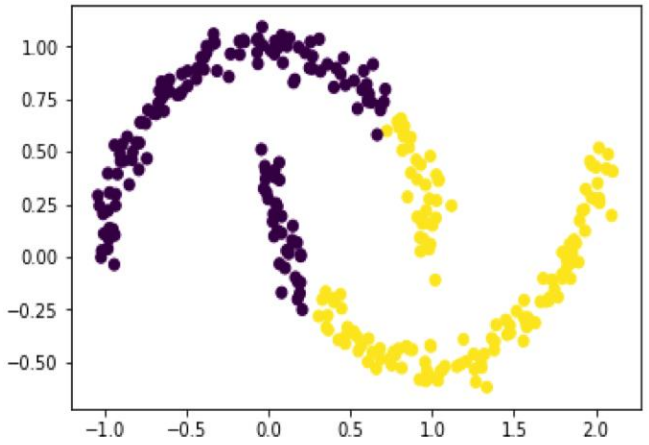
- **Agglomerative hierarchical clustering (Dendrogram) (계층적 병합 군집화)**

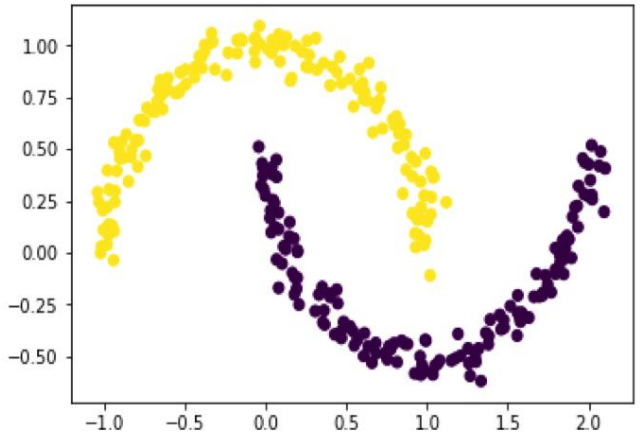


# Clustering (군집화)

- **Two Moon dataset**

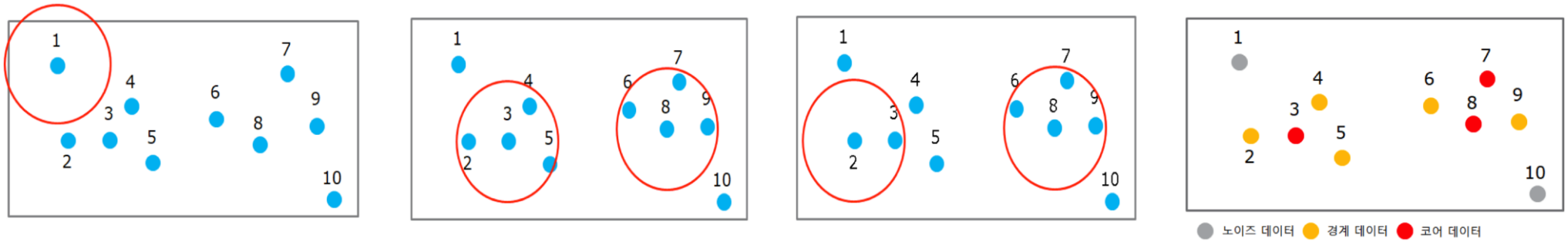


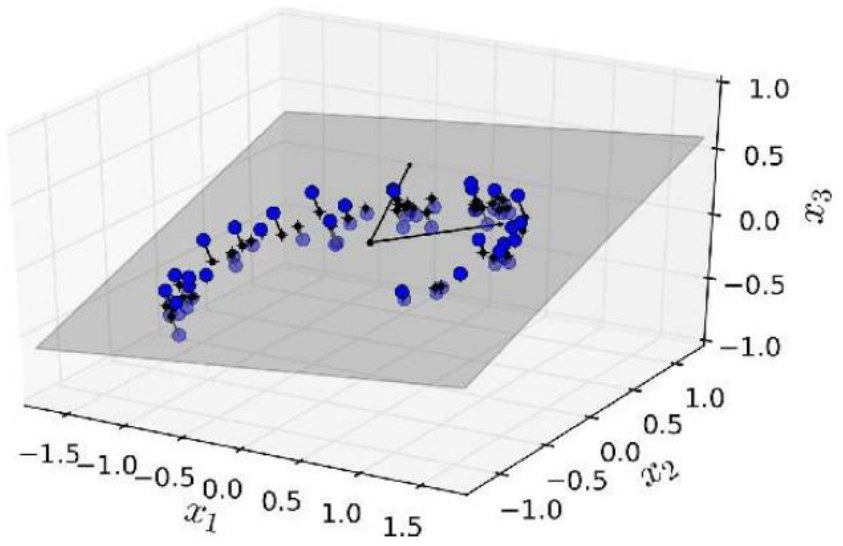
# Clustering (군집화)

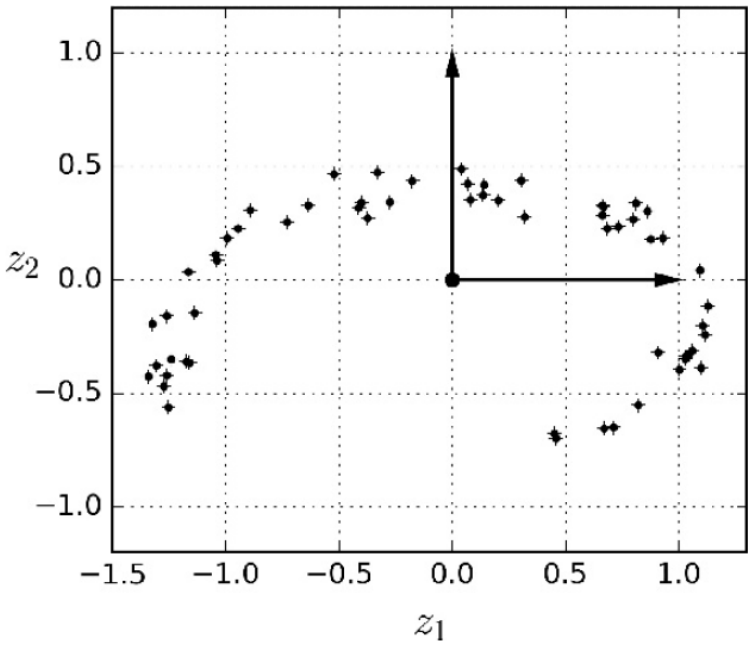
- **DBSCAN(**Density Based Spatial Clustering of Applications with Noise )
  - One of the most common clustering algorithm
  - Density-based (밀도기반): "가까이 있는 샘플들은 같은 군집에 속한다" 는 원칙
    - **Core point**:거리 e (epsilon) 내에 m (minPts) 개의 점이 있을 때 – 스스로 Cluster 형성
    - **Border point**: Core point 는 아니지만 Core point 의 군집에 속할 때
    - **Noise point**: 어느 군집에도 포함되지 않은 점
    - **Core Point 가 다른 Core 의 군집 일부가 되면 하나의 군집으로 연결**
    - (ex) m = 4



# Dimension Reduction

- **Dimension Reduction approach**
  - Projection
  - Manifold Learning

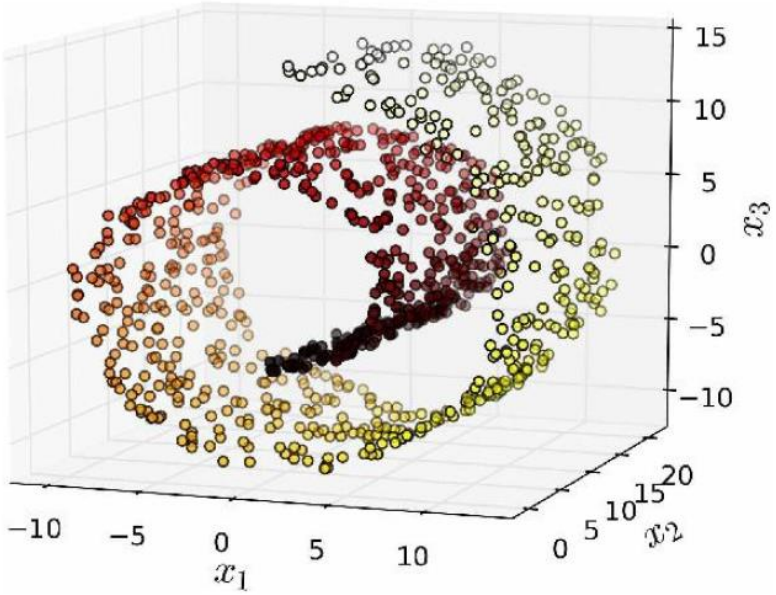


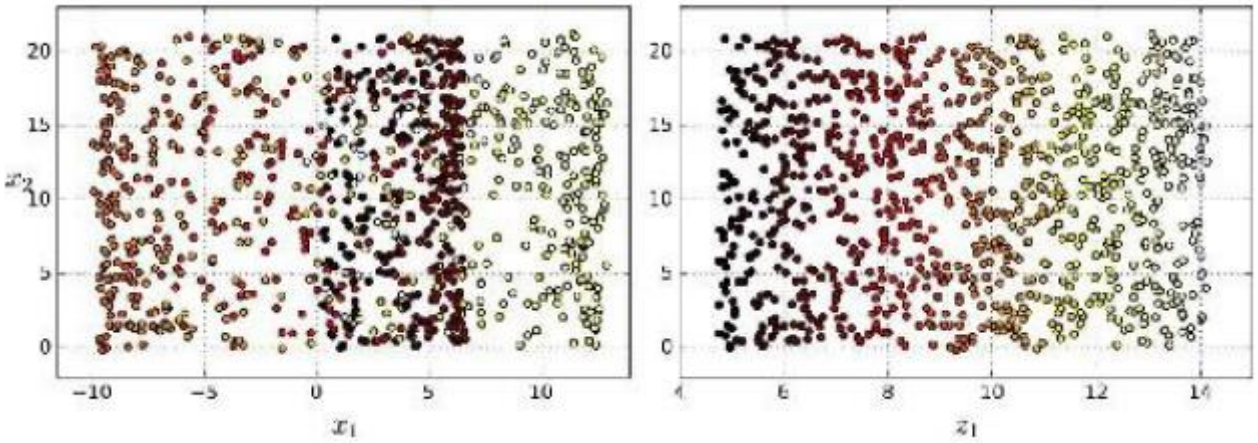




A 3D dataset lying close to a 2D subspace

# Dimension Reduction

- **Dimension Reduction approach**
  - Projection
  - Manifold Learning

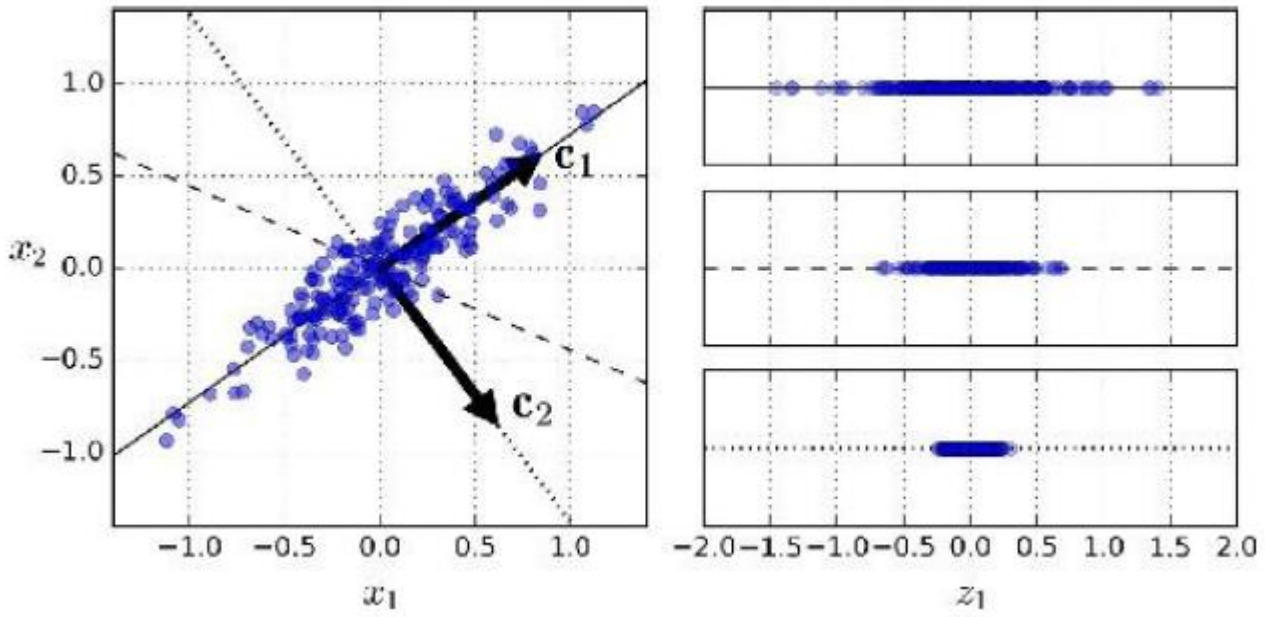



Swiss roll: Projection is not always Good. – Projection and Unrolling.

# Dimension Reduction

- **Preserving the Variance**
  - Select the axis that preserves the **maximum amount of Variance** (lose less information)
  - It minimizes the mean squared distance between the original dataset and projected ones.



# Dimension Reduction

- **Principal Component Analysis – linear**
  - Summarize the data consisting of *p* features into *k* new (uncorrelated) features
  - New *k* features are linear combination of original *p* features
  - project original data on the new axes in a way that preserves the variance of the original data as much as possible.
  - Main purposes:
    - Data Dimension Reduction (n x p -> n x k, where k << p)
    - Data analysis and visualization

# Dimension Reduction

- **PCA calculation**

**Step 1.** 데이터 정규화 (mean centering)

**Step 2.** 기존 변수의 covariance (correlation) matrix 계산

$$Cov(x,y) = \frac{\Sigma (x_i - \bar{x}) * (y_i - \bar{y})}{N}$$

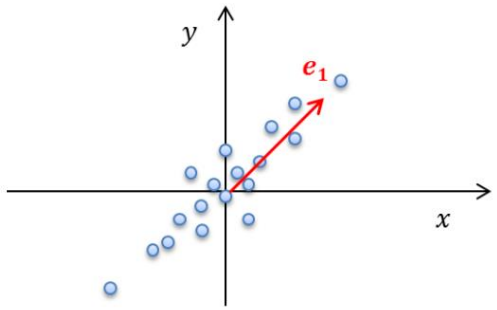
**Step 3.** Covariance (correlation) matrix로부터 eigenvalue 및 이에 해당되는 eigenvector를 계산

Eigen vector of Cov(x,y): 분산의 방향
Eigen value: 분산의 크기

**Step 4.** Eigenvalue 및 해당되는 eigenvectors 를 순서대로 나열

$\lambda(1) > \lambda(2) > \lambda(3) > \lambda(4) > \lambda(5)$

$e(1) > e(2) > e(3) > e(4) > e(5)$, $e(i)$, $i=1,\ldots,5$ is a vector

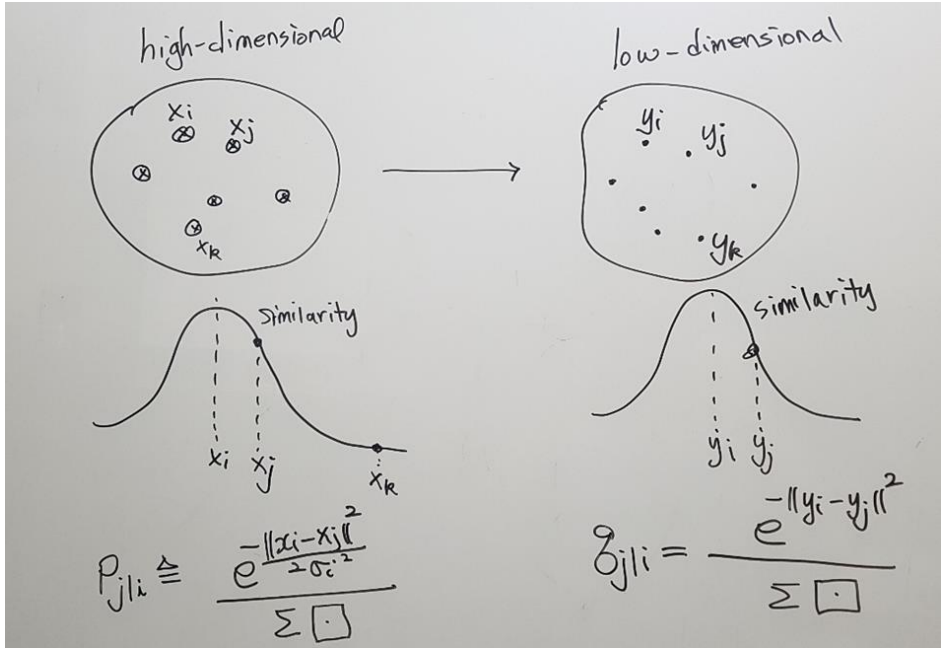



**Step 5.** 정렬된 eigenvector를 토대로 기존 변수를 변환

$$Z_1 = e(1)\mathbf{X} = e_{11} \cdot X_1 + e_{12} \cdot X_2 + \ldots + e_{15} \cdot X_5$$
$$Z_2 = e(2)\mathbf{X} = e_{21} \cdot X_1 + e_{22} \cdot X_2 + \ldots + e_{25} \cdot X_5$$
$$\ldots \quad = \quad \ldots$$
$$Z_5 = e(5)\mathbf{X} = e_{51} \cdot X_1 + e_{52} \cdot X_2 + \ldots + e_{55} \cdot X_5$$

# Dimension Reduction

- **t-SNE (T-distributed Stochastic Neighbor Embedding) - nonlinear**
  - Converts similarities between data points to joint probabilities (normalized)
  - Tries to minimize the KL-divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data
  - Very high time-complexity (recommended to use PCA (dense matrix) or TruncatedSVD (sparse matrix) for high dimensional features)



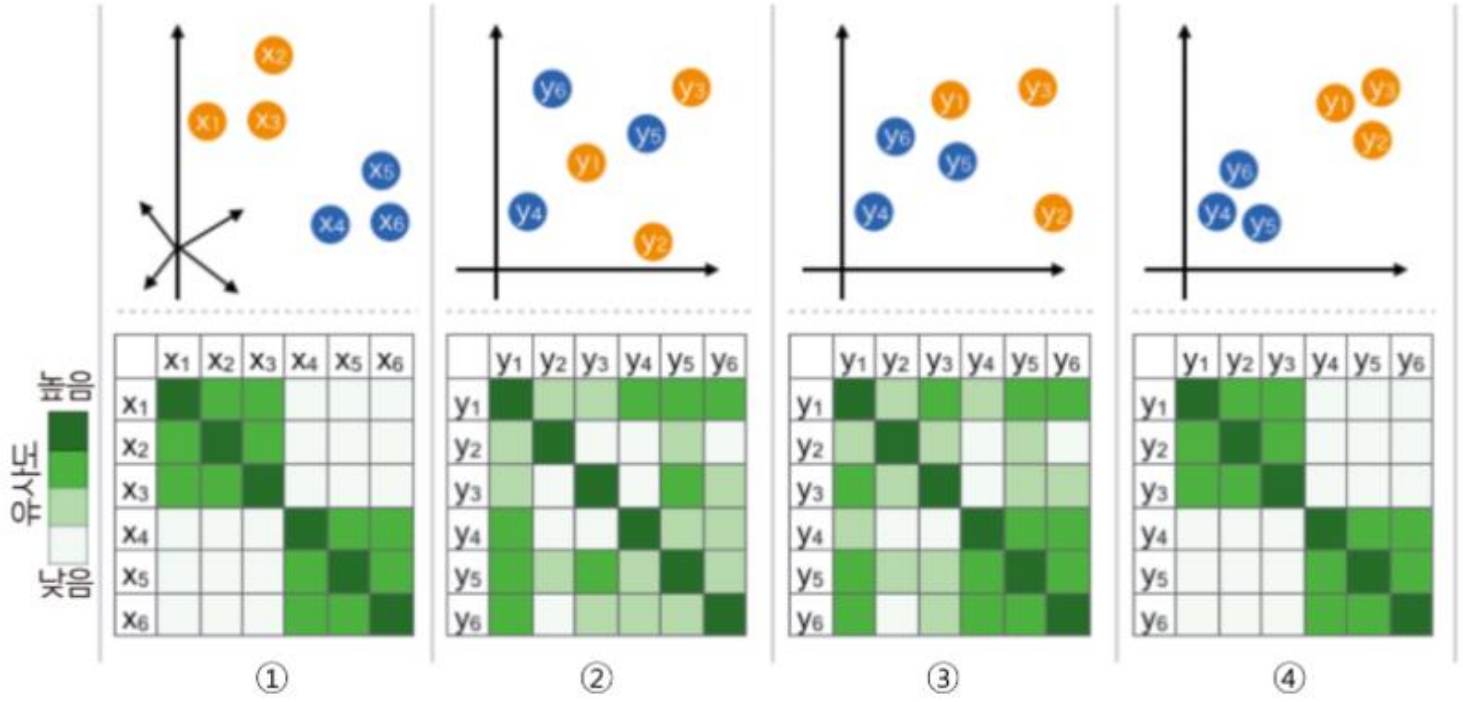

- Hyper–parameter 'perplexity' controls $\sigma_i$, normally 5 ~ 50
- We want to make $q_{j|I}$ -> $p_{j|I,}$ the cost function C is

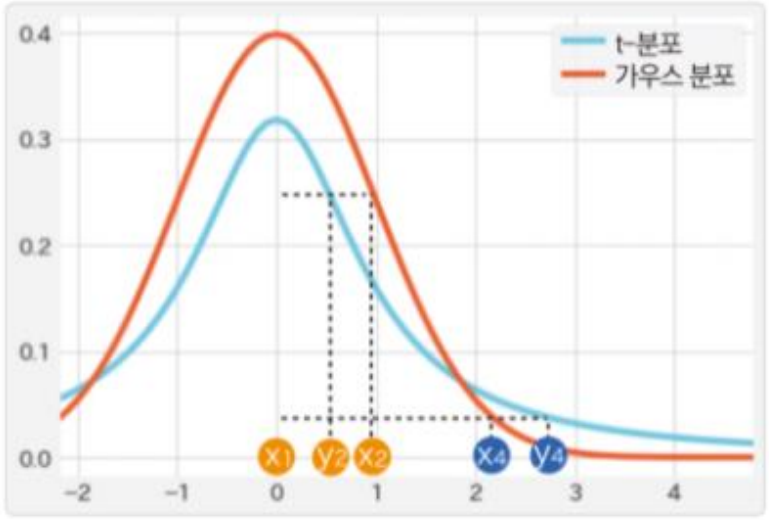
$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

# Dimension Reduction

- **t-SNE (T-distributed Stochastic Neighbor Embedding) - nonlinear**
  - Example showing the t-SNE operation



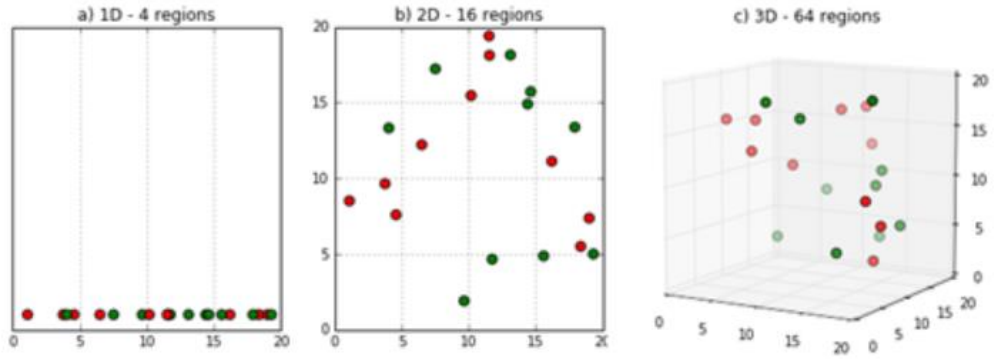

(low dimensional space상에서)
high similarity 는 더 가깝게, low similarity 는 더 멀게 배치

# Practical issues in machine learning

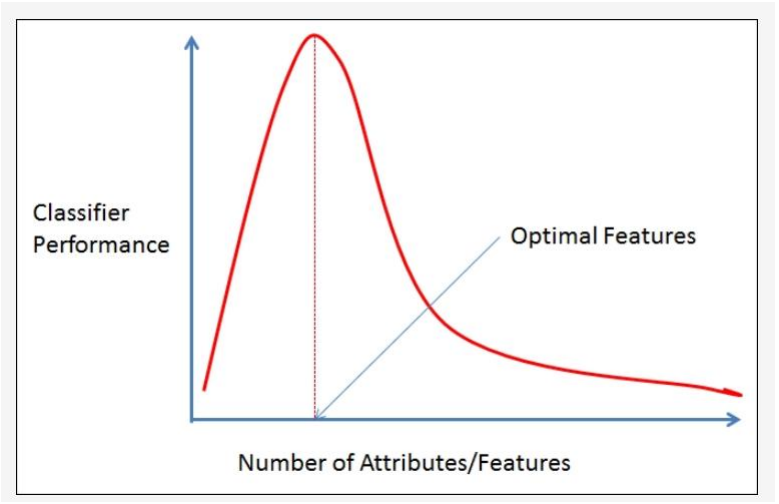
| issue | result | method |
|---|---|---|
| Data volume, velocity, and scalability | • May be infeasible due to constraints in algorithms and hardware | • Data sampling (stratified sampling, varying sample sizes) |
| Data quality and noise (missing values, duplicate values, incorrect values, Incorrect formatting) | • Incorrect or incomplete model | • data cleaning |
| Imbalanced data | • affects the choice of learning, the process of selecting algorithms, model evaluation and verification<br>• May suffer large biases | • cost-sensitive learning, ensemble learning, outlier detection |
| Overfitting | • Poor performance (to unseen data) | • More data, simple model, regularization, early stopping, dropout |
| Curse of dimensionality (too many features) | • Introduce **sparsity** (fewer data points on average per unit volume of feature space), hence poor performance (in distance-based models) | • Dimension reduction<br>• Feature selection<br>• Feature engineering |

# Practical issues in machine learning

- **Curse of dimensionality**
  - **Sparsity** of data occurs when moving to higher dimensions. (risk of massively overfitting)
  - **Distance concentration**: as dimensionality increases, distance may converge to the same value between different samples (critical in distance-based model like knn, clustering, etc.)
- Each features increases the data set requirement **exponentially**.
- **How to mitigate it**
  - **Dimensionality reduction techniques** (feature selection or feature extraction)




sparsity




Curse of dimensionality illustrated in classification

# How to deploy ML model?

- Deploy your machine learning model into production
- Some free deployment platforms:
  - Algorithmia: create code snippets and you can call your code as an API
  - PythonAnywhere: easy to run Python in cloud (do not support GPU)
  - Heroku: flexible and easy to use
  - GCP(Google Could Platform): offers AI platform, App engine, Could functions
  - Microsoft Azure Functions: serverless cloud service as a FaaS (Functions-as-a-service)
  - AWS Lambda from Amazon: serverless computing service as part of Amazon Web Services
  - Use the **mc2gen** Python library: convert the model into 14 different programming languages
  - More commercial platforms
  - See https://www.freecodecamp.org/news/deploy-your-machine-learning-models-for-free/

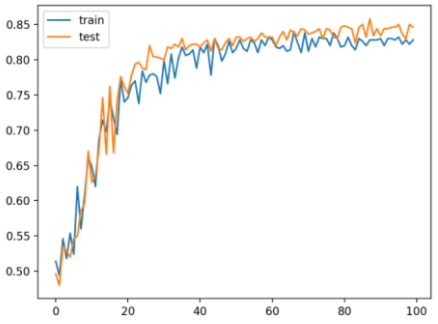
# More on Machine Learning

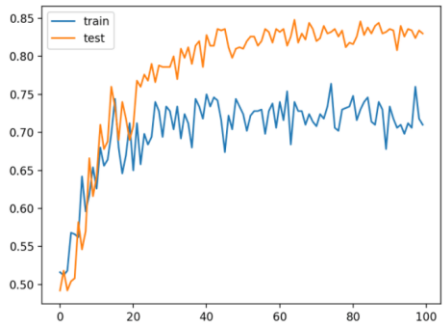
- **Batch normalization**
  - To reduce gradient vanishing and exploding problem
  - Automatically standardize the inputs to layers in Deep Learning network (over mini-batch)
  - Dramatically accelerating the training process, and in some cases improves the performance
  - Can be used **before** or **after** the activation function (in ANN, CNN, RNN)

```
model = Sequential
model.add(Dense(32))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

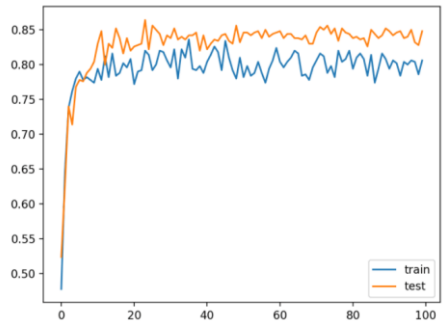
```
...
model = Sequential
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
```

- (example) a simple 2-layer MLP classification (https://machinelearningmastery.com/)




Batch normalize before ReLU




Batch normalize after ReLU

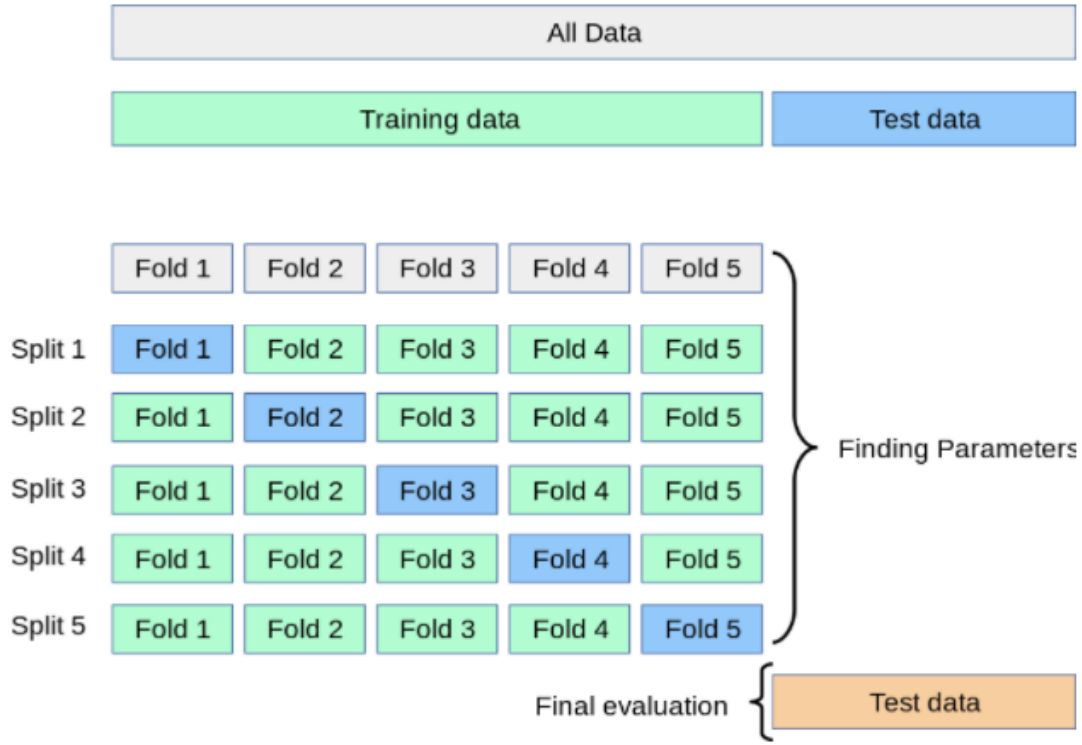
In practice, run the model a few times and compare the average outcome !

# More on Machine Learning

- **Initialization of neural networks**
  - Initialization is critical to performance, and any constant initialization will perform poorly.
  - Too small or too large initial weights will lead slow learning and divergence.
  - visual demo at https://www.deeplearning.ai/ai-notes/initialization/
- **How to find appropriate values (rule of thumb)**
  - The mean of the activations should be zero
  - The variance of activations should stay the same across every layer
- **Xavier** and **He** initialization,
  - For every layer:
    - Weights are initialized with: mean=0, variance = 1 / (fan_in + fan_out) or 2 / (fan_in)
    - Biases are initialized with zeros
  - Xavier works better for layers with sigmoid activation, and He initialization works better for layers with ReLu.
- Keras supports both.

# More on Machine Learning

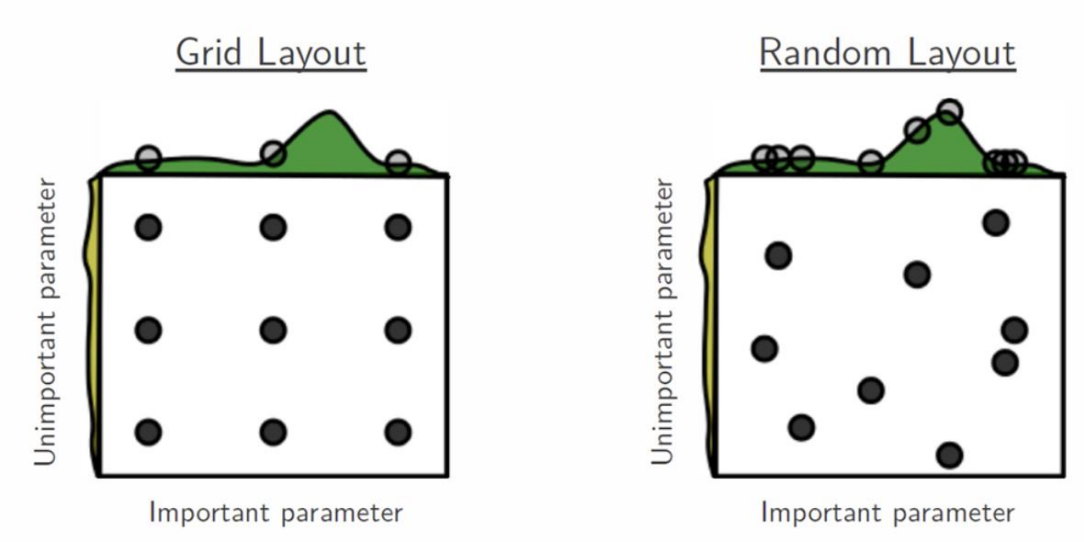
- **(stratified) K-Fold Cross Validation**
  - Helps to generalize the model (average)
  - Can see how the model is robust, and data is reliable




```
>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1, random_state=42)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores
array([0.96..., 1. , 0.96..., 0.96..., 1. ])
```

# More on Machine Learning

- **Hyperparameter tuning**
  - Grid Search: select the best among possible combination of hyperparameters
  - Random search: provide a statistical distribution or ranges for search
  - Fine-tuning: do the fine-tuning around the candidate points

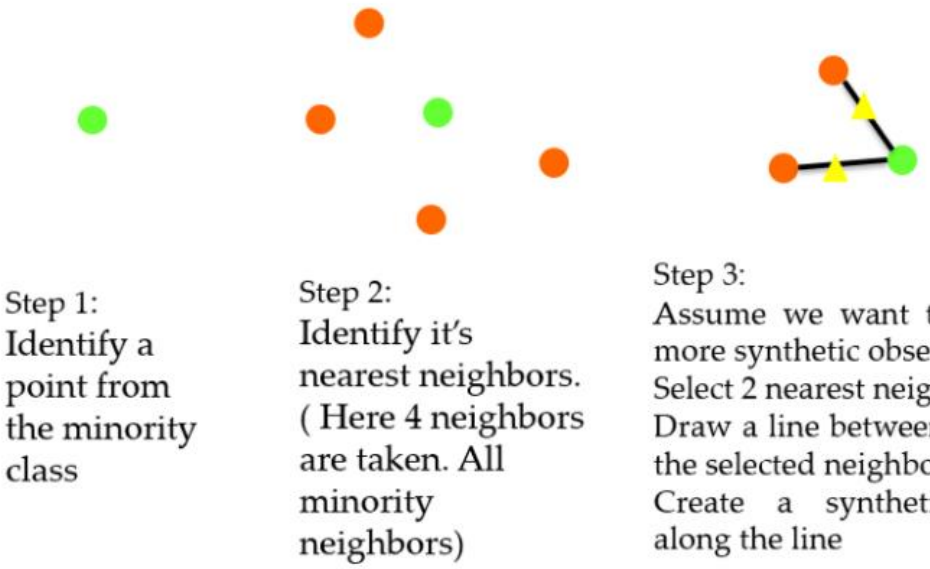


# More on Machine Learning

- **Class imbalance problem**
  - Class distributions are highly imbalanced in the training dataset
  - Tend to have low prediction accuracy for the infrequent (minority) class
  - Exists in many real word classification problems, such as fraud detection, spam detection, threat-object detection, anomaly detection, etc.
  - Causes: properties of the domain, biased sampling, measurement error
- **How to reduce the imbalance problem?**
  - Artificial resampling: over-sampling (replicating minority class), under-sampling of majority class
  - SMOTE(Synthetic Minority Oversampling TEchnique): make synthetic data points by finding the nearest neighbors to each minority sample
  - Augmentation or Use generative model for synthetic data
  - More weights on minority samples
  - Majority sample selection based on RL
  - Still a hot research topic

# Imbalance Problem

- **SMOTE (**Synthetic Minority Oversampling Technique**)**

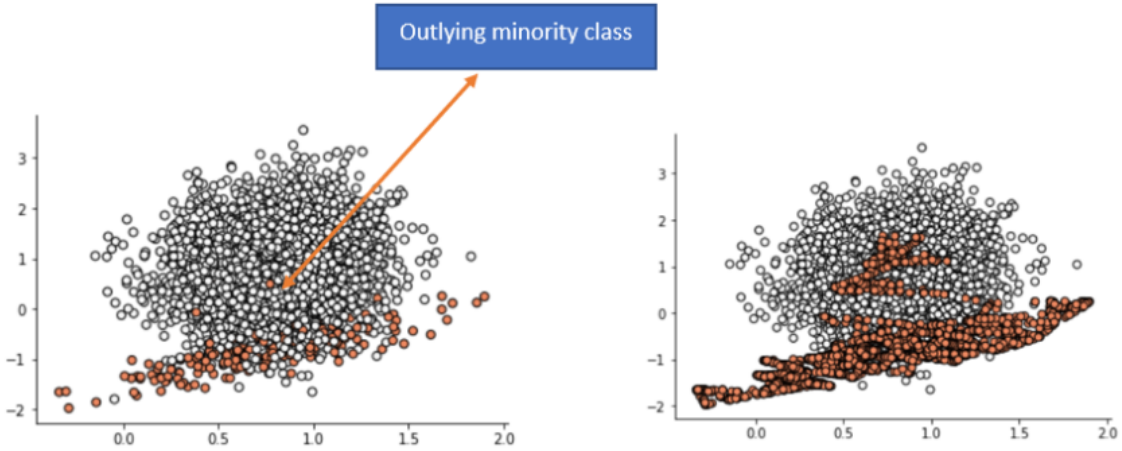


Step 1:
Identify a point from the minority class

Step 2:
Identify it's nearest neighbors. ( Here 4 neighbors are taken. All minority neighbors)

Step 3:
Assume we want to add 2 times more synthetic observations
Select 2 nearest neighbor randomly
Draw a line between the point and the selected neighbors
Create a synthetic observation along the line

SMOTE, Synthetic Minority Observation Generation Process (Source: Author)

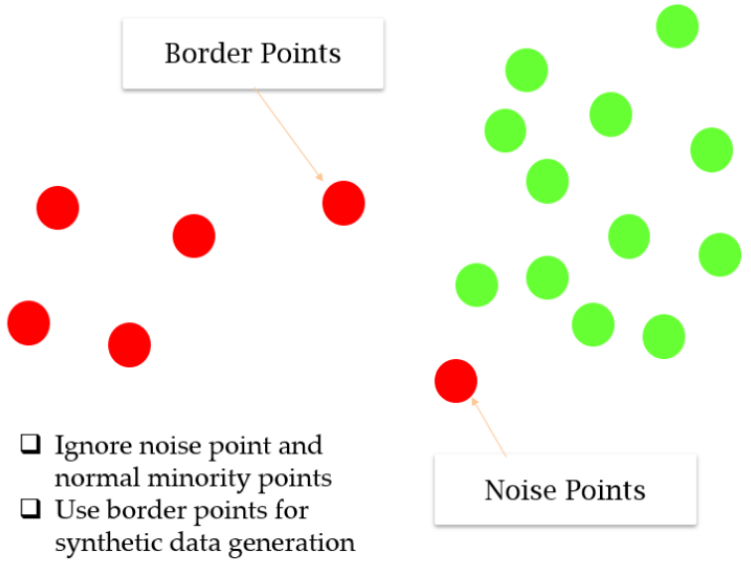



Left Side: Original Data Right Side: Data after SMOTE is Applied ( Image Source: Author)

- If there are outlying minority classes and appear in the majority class, it creates a line bridge with the majority class. (problem!)

# Imbalance Problem

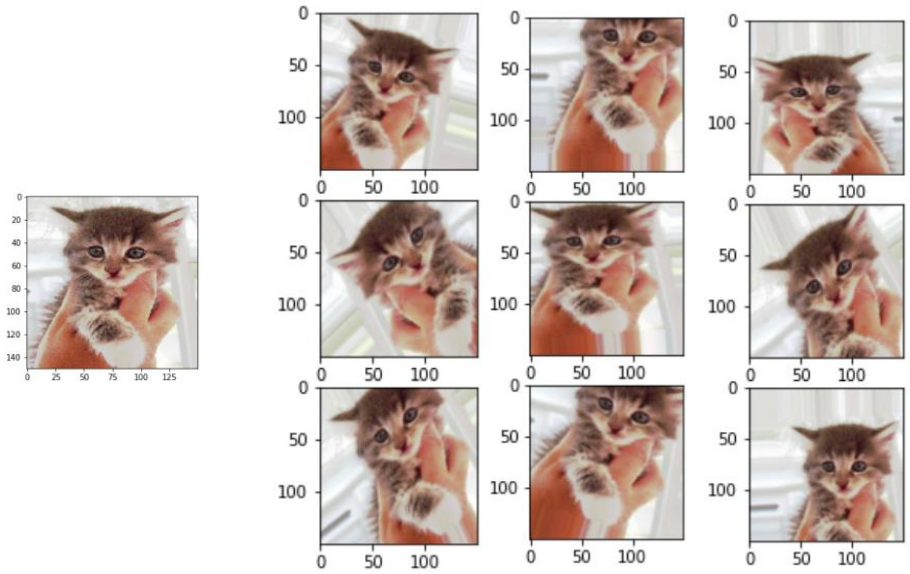
- **Borderline SMOTE**
  - Classify minority classes as noise point if all neighbors are the majority class – ignored.
  - Classifies a few points as border points that have both majority and minority class.
  - Resample only from border points.
  - **End up giving more attention to extreme observations.**
- More variants
  - ADASYN
  - and more …




Border Line SMOTE : (Image Source Author)

# More on Machine Learning

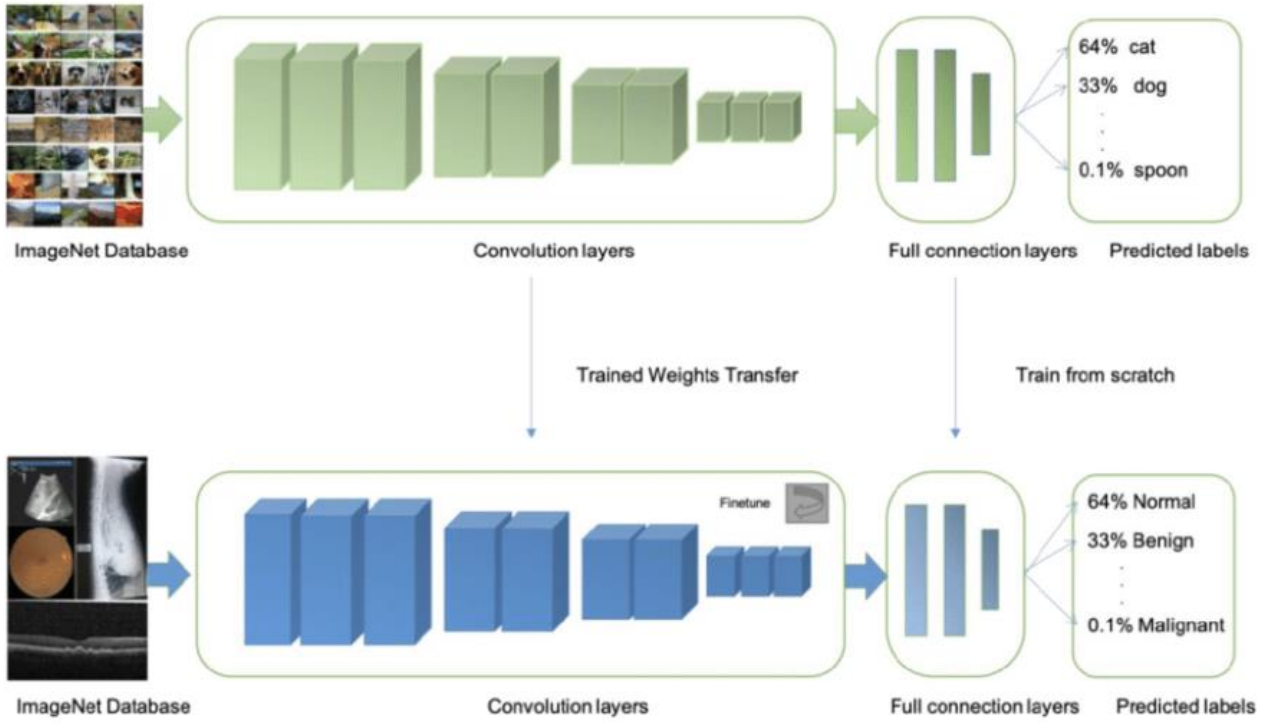
- **Augmentation (증강 or 확장)**
  - Increase the amount of data by adding slightly modified copies of existing data or newly created synthetic data from existing data
  - Introducing new synthetic images: transformation, GAN, image synthesis
  - Data augmentation for speech recognition based on RNN



```
train_datagen = ImageDataGenerator(
    rescale= 1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip = True)
```

# More on Machine Learning

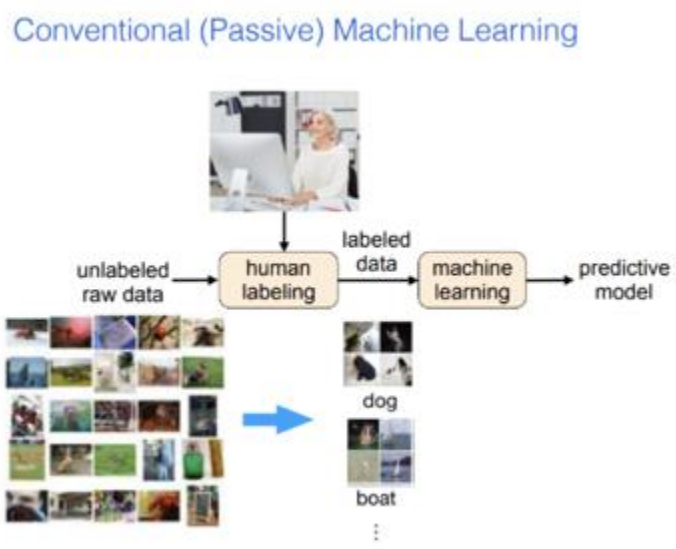
- **Transfer Learning (전이학습)**
  - A model trained on one task is reused on a second related task
  - Examples for image: Oxford VGG model, Google Inception model, Microsoft ResNet model
  - Examples for language data: Google's word2vec model, Stanford's Glove model

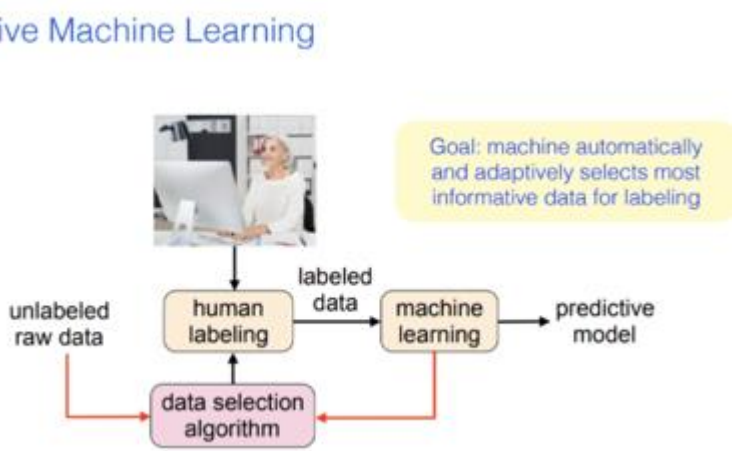


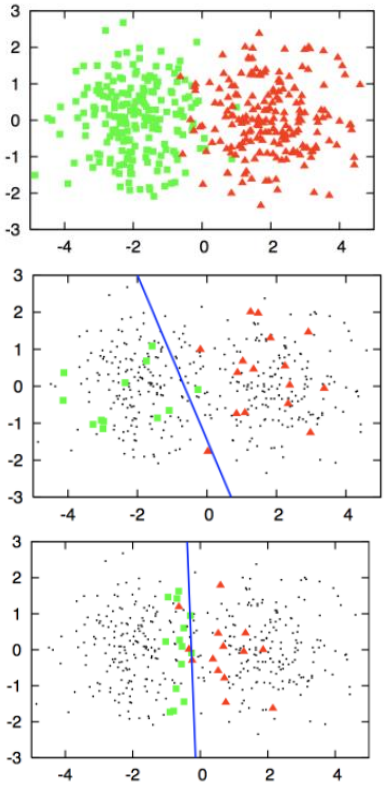

[ref: https://www.researchgate.net/figure/]

# More on Machine Learning

- **Active learning**
  - Machine may do better labeling than human, or we may have many unlabeled samples.
  - Machine automatically and adaptively selects most informative data for labeling.
  - Many query (selection) strategies



poor selection

better selection

[ref: https://www.datacamp.com/community/tutorials/active-learning]

# Scikit-Learn design principle

- **Consistency**
  - **Estimators**: **estimate some parameters based on dataset**
    - `fit(X [,y])` method with some hyper-parameters
  - **Transformers: some estimators can transform a dataset**
    - `transform(X)` method
    - `fit_transform(X [,y])`: equivalent to calling `fit()` and `transform()`
  - **Predictors: making predictions**
    - `predict()` method : returns a dataset of corresponding predictions
    - `score()` method : measure the quality of the predictions
- **Inspection**
  - **Hyper-parameters** are accessed via instance variable (e.g. `imputer.strategy`)
  - Estimator's **learned parameters** are accessed via instance variable with an underscore suffix (e.g. `imputer.statistics_`)

# Scikit-Learn design principle

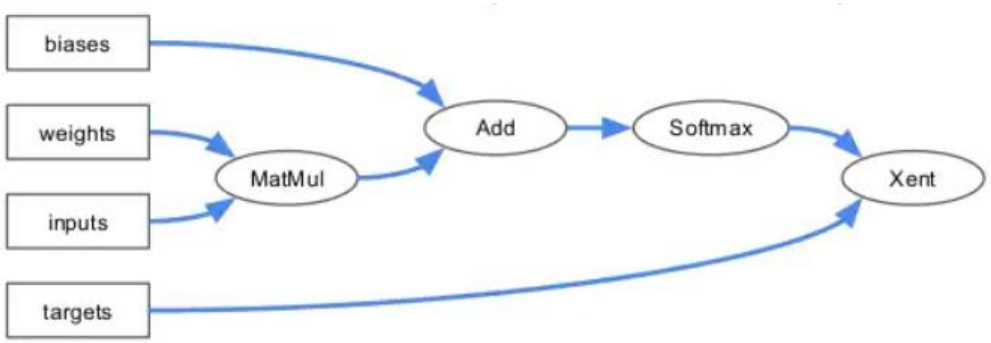
- **Nonproliferation of classes**
  - Datasets are represented as Numpy arrays or Scipy sparse matrices.
  - Hyper-parameters are just regular Python strings or numbers
- **Composition**
  - Existing building blocks are reused as much as possible.
  - For example, it is easy to create a `Pipeline` estimator from an arbitrary sequence of transformers followed by a final estimator.
- **Sensible defaults**
  - Provides reasonable default values for most parameters

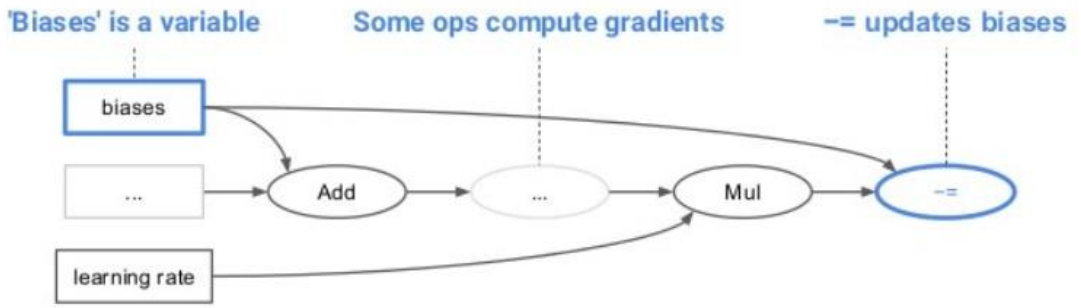
# Tensorflow

- Google's open source library for machine learning
- Tensor: N-dimensional array
- Flow: data flow computation framework
- http://tensorflow.org
- Machine learning Framework based on data flow graph, automatic differentiation
- Tensorflow 0.5 Release (2015.11)
- C++ Core, Python API
- Several High-level API including Keras
- Define and Run (Graph and Session)

# Tensorflow

- **Computation as a Dataflow Graph**
  - Graph of Nodes, also called operations (ops)




**Forward**:
- Edges are N-dimensional arrays: Tensors

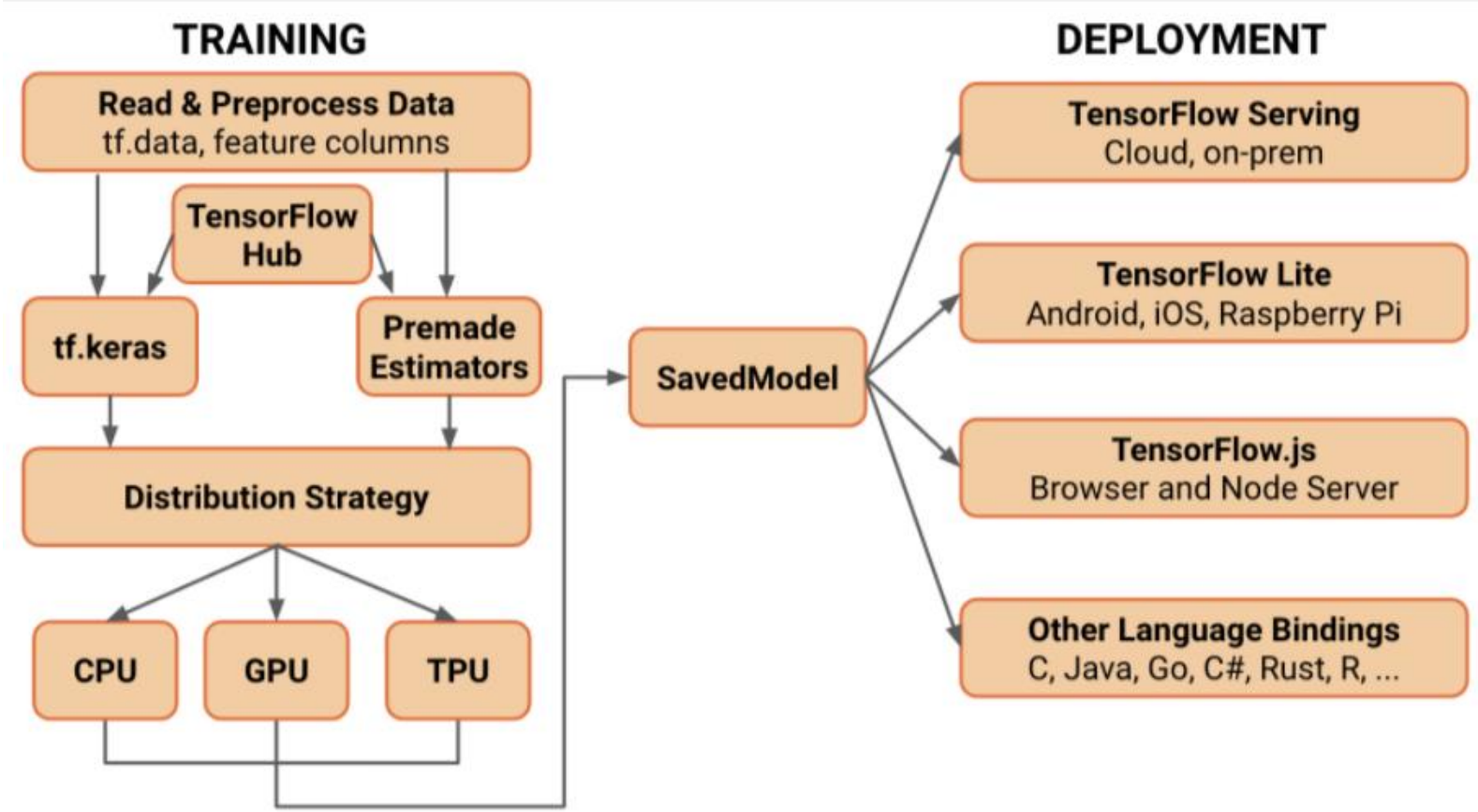
**Backward graph and updates:**
- Backward graph and update are added automatically to graph

# Tensorflow 2.0

- Eager Execution (Define by Run):
  - easy debug
  - and rapid development
  - Autograph to boost performance
- Functions, not Session
- Keras Python API
- API Cleanup
- No more Globals
- Default since 2019
- Standard SavedModel file format
  - TensorFlow Serving (ML Servers)
  - Tensorflow Lite (for Mobiles or IoT)
  - TensorFlow.js (in JavaScript, and for browser)

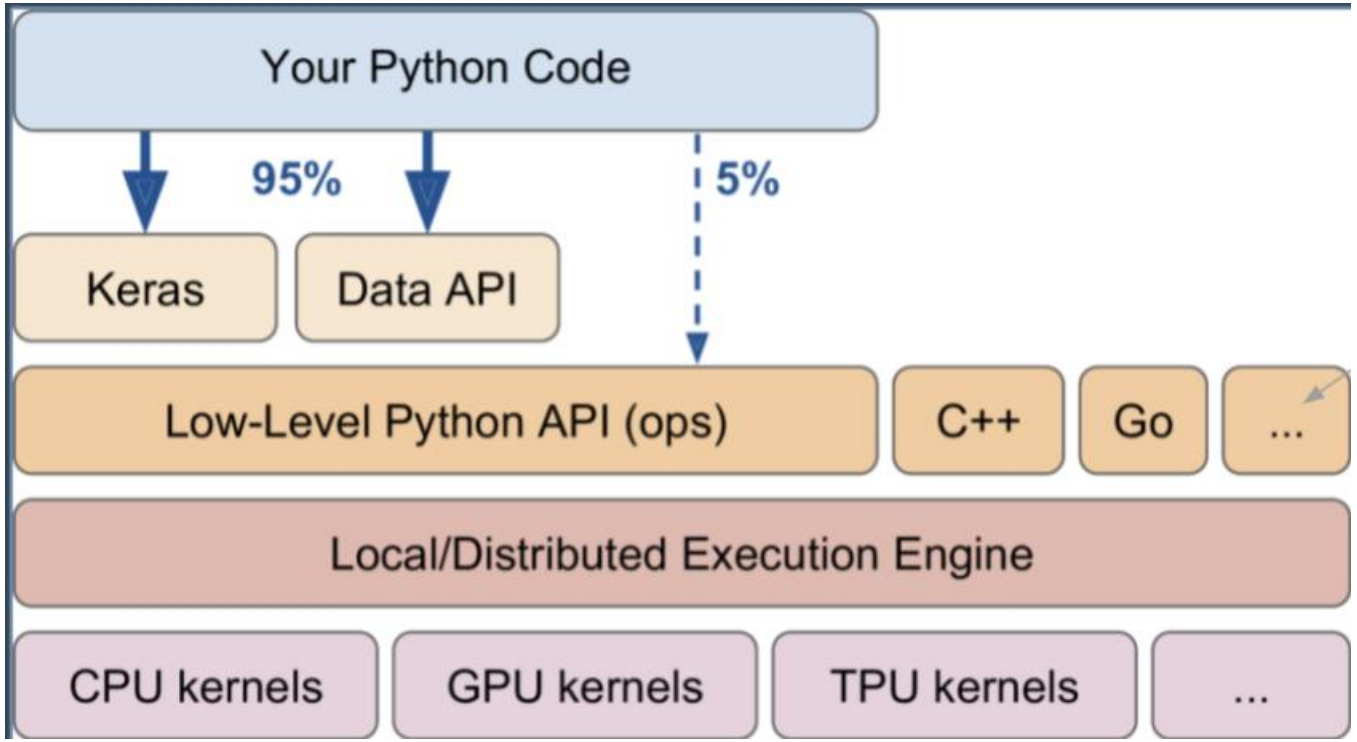
# Tensorflow 2.0

- **TF2.0 Concept Diagram**



# Keras

- High-level Neural Networks Specification (https://keras.io) (2015)
- Python API for TensorFlow 2.0 (2020)
- Deprecated tf.layer, tf.contrib.layers(Slim)



# Tensorflow Keras

- **Sequential API**
  - Plain stack of layers (one input and one output tensor)

```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(4),
    ]
)
```

```
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

  - (ex) Transfer learning

```
# Load a convolutional base with pre-trained weights
base_model = keras.applications.Xception(
    weights='imagenet',
    include_top=False,
    pooling='avg')

# Freeze the base model
base_model.trainable = False

# Use a Sequential model to add a trainable classifier on top
model = keras.Sequential([
    base_model,
    layers.Dense(1000),
])

# Compile & train
model.compile(...)
model.fit(...)
```

# Tensorflow Keras

- **Keras model**

```
from tensorflow import tf

model = tf.keras.Sequential()
# 64개의 유닛을 가진 완전 연결 층을 모델에 추가합니다:
model.add(tf.keras.layers.Dense(64, activation='relu'))
# 또 하나를 추가합니다:
model.add(tf.keras.layers.Dense(64, activation='relu'))
# 10개의 출력 유닛을 가진 소프트맥스 층을 추가합니다:
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# 컴파일
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 모델 훈련
model.fit(train_data, labels, epochs=10, batch_size=32)
# 모델 평가
model.evaluate(test_data, labels)
# 샘플 예측
model.predict(new_sample)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(10),
])
```

# Tensorflow Keras

- **Functional API**
  - to create models in more flexible way
  - Can handle non-linear topology, shared layers, and multiple inputs or outputs
  - All Layers and Models are callable
  - Manipulate complex graph topologies
- **Built-in layers in Keras:**
  - Convolutional layers: Conv1D, Conv2D, Conv3D, Conv2DTranspose
  - Pooling layers: MaxPooling1D, maxPooling2D, AveragePooling1D
  - SimpleRNN, GRU, LSTM, ConvLSTM2D
  - BatchNormalization, Dropout, Embedding, etc.

```
encoder_input = keras.Input(shape=(28, 28, 1), name="original_img")
x = layers.Conv2D(16, 3, activation="relu")(encoder_input)
x = layers.Conv2D(32, 3, activation="relu")(x)
x = layers.MaxPooling2D(3)(x)
x = layers.Conv2D(32, 3, activation="relu")(x)
x = layers.Conv2D(16, 3, activation="relu")(x)
encoder_output = layers.GlobalMaxPooling2D()(x)

encoder = keras.Model(encoder_input, encoder_output, name="encoder")
encoder.summary()

decoder_input = keras.Input(shape=(16,), name="encoded_img")
x = layers.Reshape((4, 4, 1))(decoder_input)
x = layers.Conv2DTranspose(16, 3, activation="relu")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu")(x)
x = layers.UpSampling2D(3)(x)
x = layers.Conv2DTranspose(16, 3, activation="relu")(x)
decoder_output = layers.Conv2DTranspose(1, 3, activation="relu")(x)

decoder = keras.Model(decoder_input, decoder_output, name="decoder")
decoder.summary()

autoencoder_input = keras.Input(shape=(28, 28, 1), name="img")
encoded_img = encoder(autoencoder_input)
decoded_img = decoder(encoded_img)
autoencoder = keras.Model(autoencoder_input, decoded_img, name="autoencoder")
autoencoder.summary()
```