KYOTO UNIVERSITY

# Statistical Learning Theory
## - Classification -

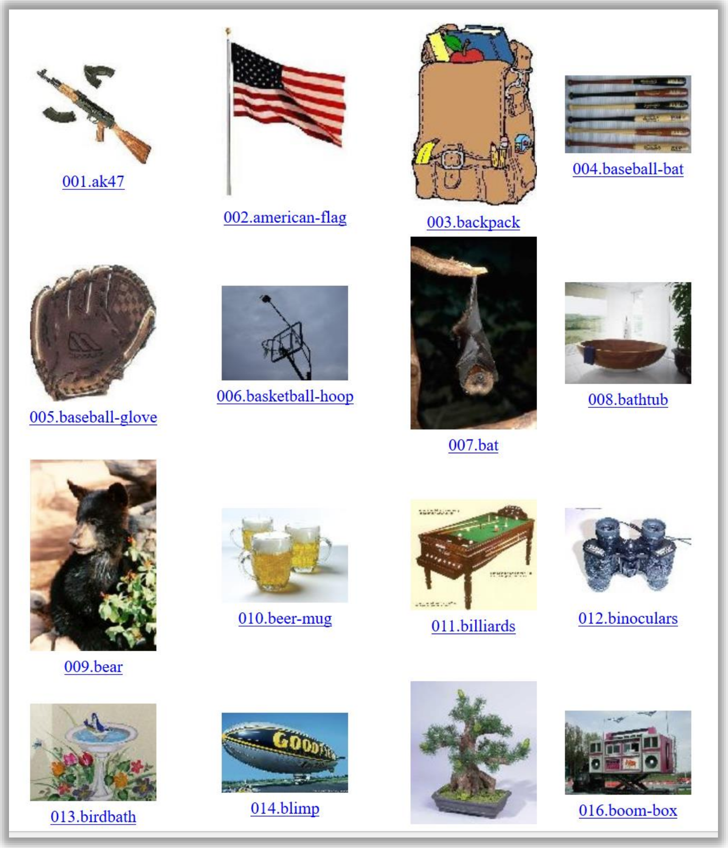
Hisashi Kashima

DEPARTMENT OF INTELLIGENCE SCIENCE AND TECHNOLOGY

# Classification

# Classification: Supervised learning for predicting discrete variable

- Goal: Obtain a function $f: \mathcal{X} \to \mathcal{Y}$ ($\mathcal{Y}$: discrete domain)
  - E.g. $x \in \mathcal{X}$ is an image and $y \in \mathcal{Y}$ is the type of object appearing in the image
  - Two-class classification: $\mathcal{Y} = \{+1, -1\}$
- Training dataset: $N$ pairs of an input and an output $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$




http://www.vision.caltech.edu/Image_Datasets/Caltech256/

# Some applications of classification: From binary to multi-class classification

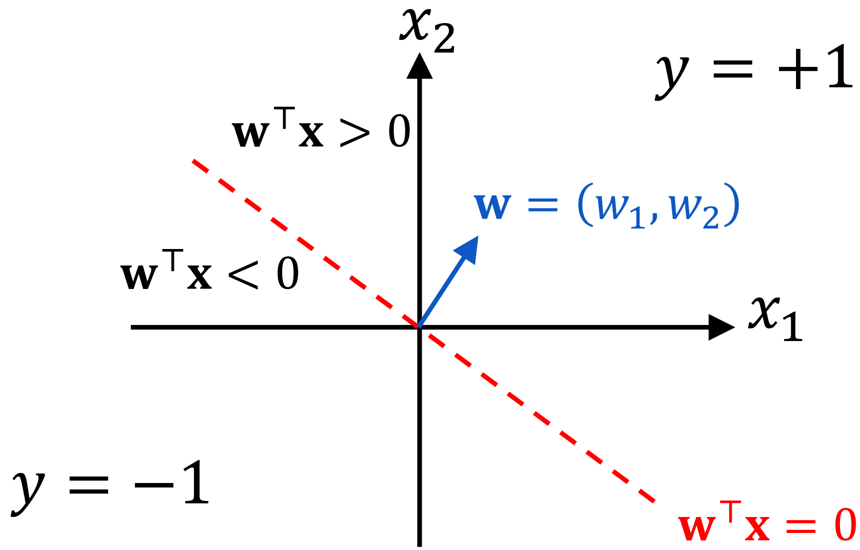
- Binary (two-class)classification:
  - Purchase prediction: Predict if a customer $\mathbf{x}$ will buy a particular product $(+1)$ or not $(-1)$
  - Credit risk prediction: Predict if a obligor $\mathbf{x}$ will pay back a debt $(+1)$ or not $(-1)$
- Multi-class classification ($\neq$ Multi-label classification):
  - Text classification: Categorize a document $\mathbf{x}$ into one of several categories, e.g., $\{\text{politics, economy, sports, ...}\}$
  - Image classification: Categorize the object in an image $\mathbf{x}$ into one of several object names, e.g., $\{\text{AK5, American flag, backpack, ...}\}$
  - Action recognition: Recognize the action type ($\{\text{running, walking, sitting, ...}\}$) that a person is taking from sensor data $\mathbf{x}$

# A simple model for classification: Linear classifier

- Linear (binary) classification model:

$$y = \mathrm{sign}(\mathbf{w}^\top \mathbf{x}) = \mathrm{sign}(w_1 x_1 + w_2 x_2 + \cdots + w_D x_D)$$

  - $|\mathbf{w}^\top \mathbf{x}|$ indicates the intensity of belief
  - $\mathbf{w}^\top \mathbf{x} = 0$ gives a separating hyperplane
    - $\mathbf{w}$: normal vector perpendicular to the separating hyperplane



# Learning framework: Loss minimization and statistical estimation

- Two learning frameworks

  1. Loss minimization: $L(\mathbf{w}) = \sum_{i=1}^{N} \ell\left(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}\right)$
     - Loss function $\ell$: directly handles utility of predictions
     - Regularization term $R(\mathbf{w})$
  2. Statistical estimation (likelihood maximization):
     $L(\mathbf{w}) = \prod_{i=1}^{N} f_{\mathbf{w}}(y^{(i)} | \mathbf{x}^{(i)})$
     - Probabilistic model: generation process of class labels
     - Prior distribution $P(\mathbf{w})$

- They are often equivalent : $\begin{cases} \text{Loss = Probabilistic model} \\ \text{Regularization = Prior} \end{cases}$

# Classification problem in loss minimization framework: Minimize loss function + regularization term

- Minimization problem: $\mathbf{w}^* = \mathrm{argmin}_{\mathbf{w}}\, L(\mathbf{w}) + R(\mathbf{w})$
  - Loss function $L(\mathbf{w})$ : Fitness to training data
  - Regularization term $R(\mathbf{w})$ : Penalty on the model complexity to avoid overfitting to training data (usually norm of $\mathbf{w}$)
- Loss function should reflect the number of misclassifications on training data
  - Zero-one loss seems reasonable:

$$\ell^{(i)}\left(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}\right) = \begin{cases} 0 & \left(y^{(i)} = \mathrm{sign}\left(\mathbf{w}^\top \mathbf{x}^{(i)}\right)\right) \\ 1 & \left(y^{(i)} \neq \mathrm{sign}\left(\mathbf{w}^\top \mathbf{x}^{(i)}\right)\right) \end{cases}$$

Correct classification

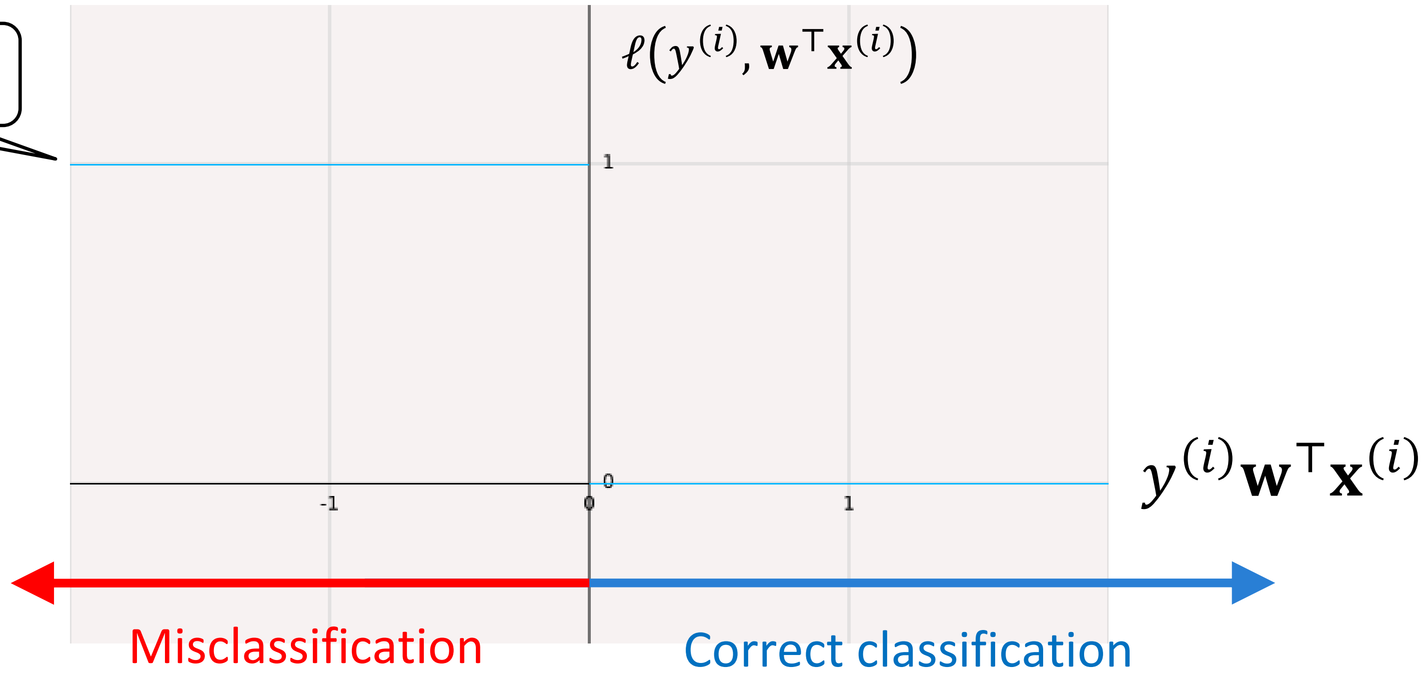
Incorrect classification

# Zero-one loss:
# Number of misclassification is hard to minimize

- Zero-one loss: $\ell\left(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}\right) = \begin{cases} 0 & \left(y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} > 0\right) \\ 1 & \left(y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \leq 0\right) \end{cases}$

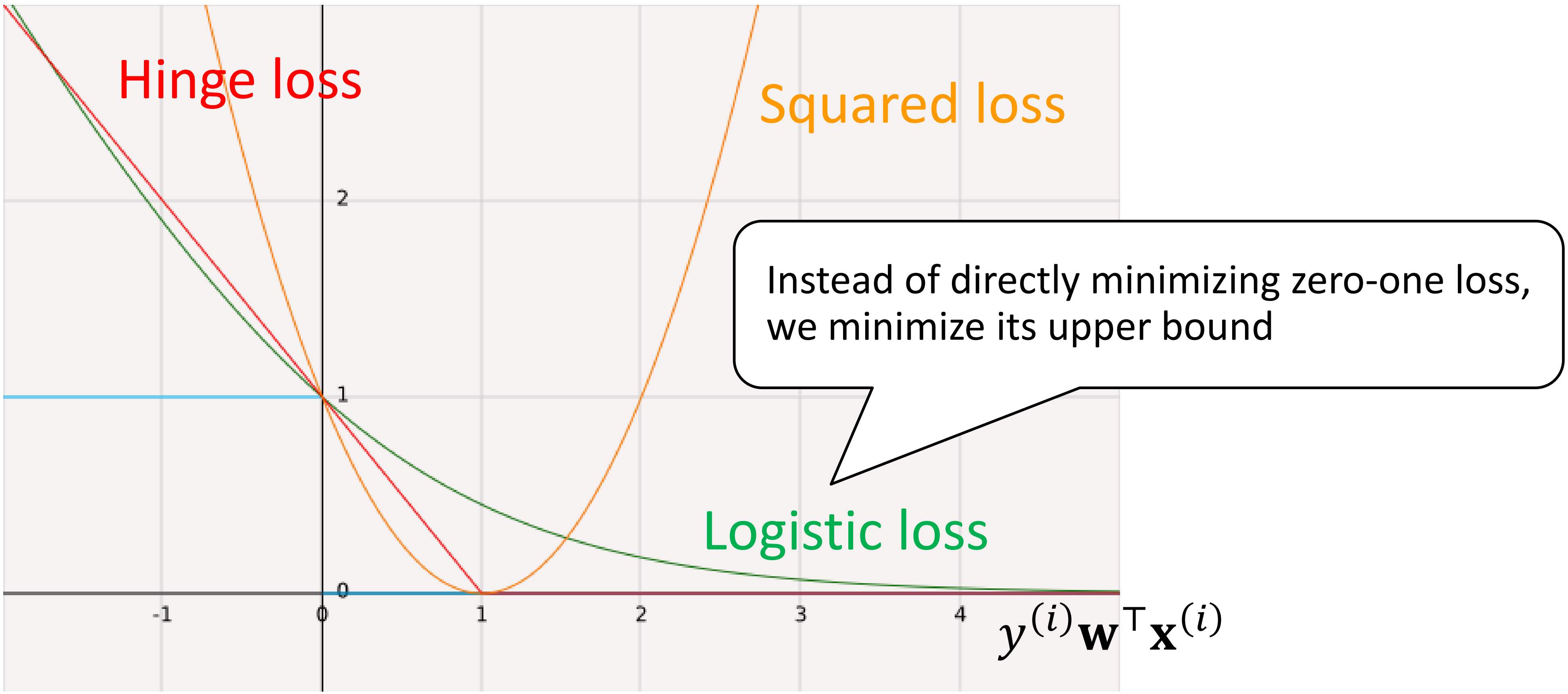
- Non-convex function is hard to optimize directly

Non-convex



# Convex surrogates of zero-one loss: Different functions lead to different learning machines

- Convex surrogates: Upper bounds of zero-one loss
  - Hinge loss → SVM, Logistic loss → logistic regression, ...



# Logistic regression

# Logistic regression: Minimization of logistic loss is a convex optimization

- Logistic loss:

$$\ell\left(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}\right) = \frac{1}{\ln 2} \ln\left(1 + \exp\left(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}\right)\right)$$

- (Regularized) Logistic regression:

Convex

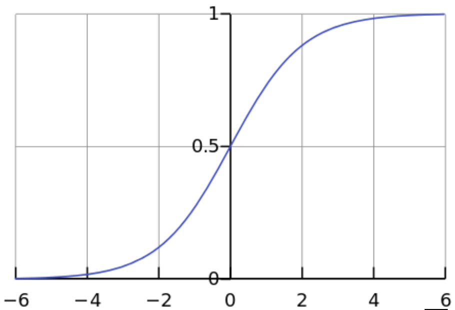
$$\mathbf{w}^* = \mathrm{argmin}_{\mathbf{w}} \sum_{i=1}^{N} \ln\left(1 + \exp\left(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}\right)\right) + \lambda \|\mathbf{w}\|_2^2$$

# Statistical interpretation:
# Logistic loss min. as MLE of logistic regression model

- Minimization of logistic loss is equivalent to maximum likelihood estimation of logistic regression model

- Logistic regression model (conditional probability):

$$f_{\mathbf{w}}(y=1|\mathbf{x}) = \sigma(\mathbf{w}^\top\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^\top\mathbf{x})}$$




  - $\sigma$: Logistic function ($\sigma$: $\Re \to (0,1)$)

- Log likelihood:

$$L(\mathbf{w}) = \sum_{i=1}^{N} \log f_{\mathbf{w}}(y^{(i)}|\mathbf{x}^{(i)}) = -\sum_{i=1}^{N} \log\left(1+\exp\left(-y^{(i)}\mathbf{w}^\top\mathbf{x}\right)\right)$$

$$\left(= \sum_{i=1}^{N} \delta\left(y^{(i)}=1\right)\log\frac{1}{1+\exp(-\mathbf{w}^\top\mathbf{x})} + \delta\left(y^{(i)}=-1\right)\log\left(1-\frac{1}{1+\exp(-\mathbf{w}^\top\mathbf{x})}\right)\right)$$

# Parameter estimation of logistic regression : Numerical nonlinear optimization

- Objective function of (regularized) logistic regression:

$$L(\mathbf{w}) = \sum_{i=1}^{N} \ln\left(1 + \exp\left(-y^{(i)}\mathbf{w}^\top\mathbf{x}^{(i)}\right)\right) + \lambda\|\mathbf{w}\|_2^2$$

- Minimization of logistic loss / MLE of logistic regression model has no closed form solution

- Numerical nonlinear optimization methods are used
  - Iterate parameter updates: $\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} + \mathbf{d}$ (until convergence)

## Parameter update :
## Find the best update minimizing the objective function

- By update $\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} + \mathbf{d}$, the objective function will be:

$$L_{\mathbf{w}}(\mathbf{d}) = \sum_{i=1}^{N} \ln\left(1 + \exp\left(-y^{(i)}(\mathbf{w} + \mathbf{d})^{\top}\mathbf{x}^{(i)}\right)\right) + \lambda\|\mathbf{w} + \mathbf{d}\|_2^2$$

- Find $\mathbf{d}^*$ that minimizes $L_{\mathbf{w}}(\mathbf{d})$:
  - $\mathbf{d}^* = \mathrm{argmin}_{\mathbf{d}}\, L_{\mathbf{w}}(\mathbf{d})$
- … but so far, this problem has not been made easier at all … 😳

# Finding the best parameter update : Approximate the objective with Taylor expansion

- Taylor expansion:

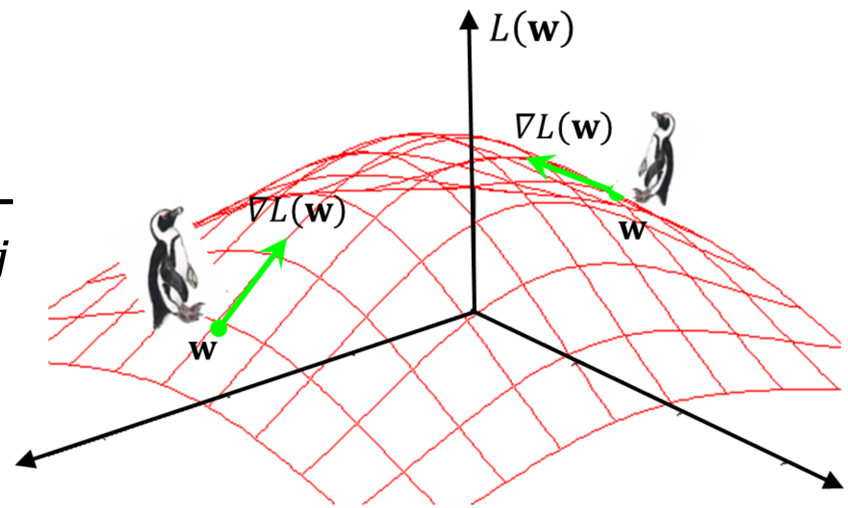
3rd-order term

$$L_{\mathbf{w}}(\mathbf{d}) = L(\mathbf{w}) + \mathbf{d}^\top \nabla L(\mathbf{w}) + \frac{1}{2}\mathbf{d}^\top \boldsymbol{H}(\mathbf{w})\mathbf{d} + O(\mathbf{d}^3)$$

– Gradient vector: $\nabla L(\mathbf{w}) = \left(\frac{\partial L(\mathbf{w})}{\partial w_1}, \frac{\partial L(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial L(\mathbf{w})}{\partial w_D}\right)^\top$

- Steepest direction

– Hessian matrix: $[H(\mathbf{w})]_{i,j} = \frac{\partial^2 L(\mathbf{w})}{\partial w_i \partial w_j}$



# Newton update :
# Minimizes the second order approximation

- Approximated Taylor expansion (neglecting the 3rd order term):

$$L_{\mathbf{w}}(\mathbf{d}) \approx L(\mathbf{w}) + \mathbf{d}^\top \nabla L(\mathbf{w}) + \frac{1}{2}\mathbf{d}^\top \boldsymbol{H}(\mathbf{w})\mathbf{d} \cancel{+ O(\mathbf{d}^3)}$$

- Derivative w.r.t. $\mathbf{d}$: $\frac{\partial L_{\mathbf{w}}(\mathbf{d})}{\partial \mathbf{d}} \approx \nabla L(\mathbf{w}) + \boldsymbol{H}(\mathbf{w})\mathbf{d}$

- Setting it to be $\mathbf{0}$, we obtain $\mathbf{d} = -\boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})$

- Newton update formula:

$$\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} - \boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})$$

$$\mathbf{w} \xrightarrow{-\boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})} \mathbf{w} - \boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})$$

# Modified Newton update:
# Second order approximation + linear search

- The correctness of the update $\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} - \boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})$ depends on the second-order approximation:

$$L_{\mathbf{w}}(\mathbf{d}) \approx L(\mathbf{w}) + \mathbf{d}^\top \nabla L(\mathbf{w}) + \frac{1}{2}\mathbf{d}^\top \boldsymbol{H}(\mathbf{w})\mathbf{d}$$

  – This is not actually true for most cases

- Use only the direction of $\boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})$ and update with $\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} - \eta \boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})$

- Learning rate $\eta > 0$ is determined by linear search:

$$\eta^* = \mathrm{argmax}_\eta\, L\big(\mathbf{w} - \eta \boldsymbol{H}(\mathbf{w})^{-1}\nabla L(\mathbf{w})\big)$$

# (Steepest) gradient descent: Simple update without computing inverse Hessian

- Computing the inverse of Hessian matrix is costly
  - Newton update: $\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} - \eta \boldsymbol{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$
- (Steepest) gradient descent:
  - Replacing $\boldsymbol{H}(\mathbf{w})^{-1}$ with $\boldsymbol{I}$ gives

Gradient of objective function

$$\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$$

    - $\nabla L(\mathbf{w})$ is the steepest direction
    - Learning rate $\eta$ is determined by line search

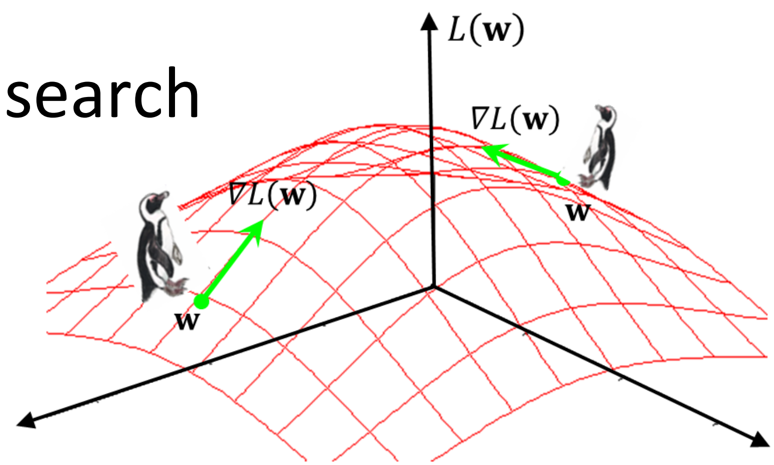
# Summary: Gradient descent

- Steepest gradient descent is the simplest optimization method:
- Update the parameter in the steepest direction of the objective function

$$\mathbf{w}^{\mathrm{NEW}} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$$

 – Gradient: $\nabla L(\mathbf{w}) = \left( \frac{\partial L(\mathbf{w})}{\partial w_1}, \frac{\partial L(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial L(\mathbf{w})}{\partial w_D} \right)^\top$

 – Learning rate $\eta$ is determined by line search

# Example of gradient descent: Gradient of logistic regression

- $L(\mathbf{w}) = \sum_{i=1}^{N} \ln\left(1 + \exp\left(-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\right)\right)$

- $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^{N} \frac{1}{1+\exp\left(-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\right)} \frac{\partial\left(1+\exp\left(-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\right)\right)}{\partial \mathbf{w}}$

$$= -\sum_{i=1}^{N} \frac{1}{1 + \exp\left(-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\right)} \exp\left(-y^{(i)}\mathbf{w}^\top \mathbf{x}^{(i)}\right) y^{(i)}\mathbf{x}^{(i)}$$

$$= -\sum_{i=1}^{N} \left(1 - f_{\mathbf{w}}\left(y^{(i)} | \mathbf{x}^{(i)}\right)\right) y^{(i)}\mathbf{x}^{(i)}$$

Can be easily computed with the current prediction probabilities

# Mini batch optimization:
# Efficient training using data subsets

- Objective function for $N$ instances:
  $L(\mathbf{w}) = \sum_{i=1}^{N} \ell\left(\mathbf{w}^\top \mathbf{x}^{(i)}\right) + \lambda R(\mathbf{w})$

- Its derivative $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^{N} \frac{\partial \ell\left(\mathbf{w}^\top \mathbf{x}^{(i)}\right)}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}}$ needs $O(N)$ computation

- Approximate this with only one instance:
  $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \approx N \frac{\partial \ell\left(\mathbf{w}^\top \mathbf{x}^{(j)}\right)}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}}$ (Stochastic approximation)

- Also we can do this with $1 < M < N$ instances:
  $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \approx \frac{N}{M} \sum_{j \in \mathrm{MiniBatch}} \frac{\partial \ell\left(\mathbf{w}^\top \mathbf{x}^{(j)}\right)}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}}$ (Mini batch)

# Model Evaluation

# Model evaluation:
## How can we know the “real” performance of a model?

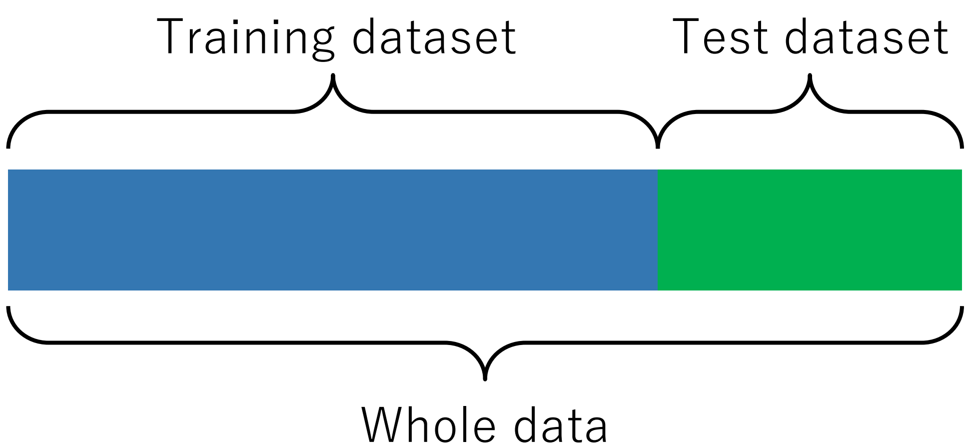
- Once you obtain a trained model, you want to deploy the model in your application
- How well will the model perform? - We are interested in the future performance of the obtained model when it is deployed
  - How many mistakes will the model make in future?
- Even the model performs perfectly on the training data, the same performance is not guaranteed for future data
- “Model evaluation” problem

# The first principle: Evaluation must use a dataset not used in training

- You *must not* evaluate your classifier based on the performance on the dataset you already used for training
- The performance of a model for the training data is not an estimate of its true performance
  - If you memorize all the answers of the training dataset, you will always be correct for them
  - … but there is no guarantee that you will be so for future data

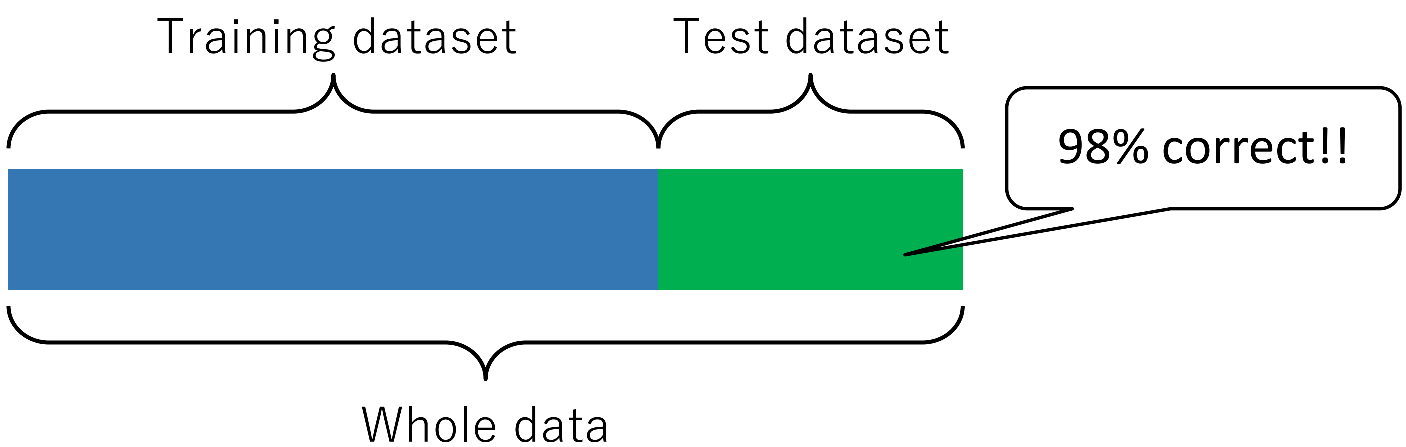
# A simplest solution for model evaluation: Secure some data for performance evaluation

- Divide the dataset into a *training dataset* and a *test dataset*
  1. Train a classifier using the training dataset
  2. Evaluate its performance on the test dataset
- This is simulating a real application scenario using only the dataset at hand (without using real future data)



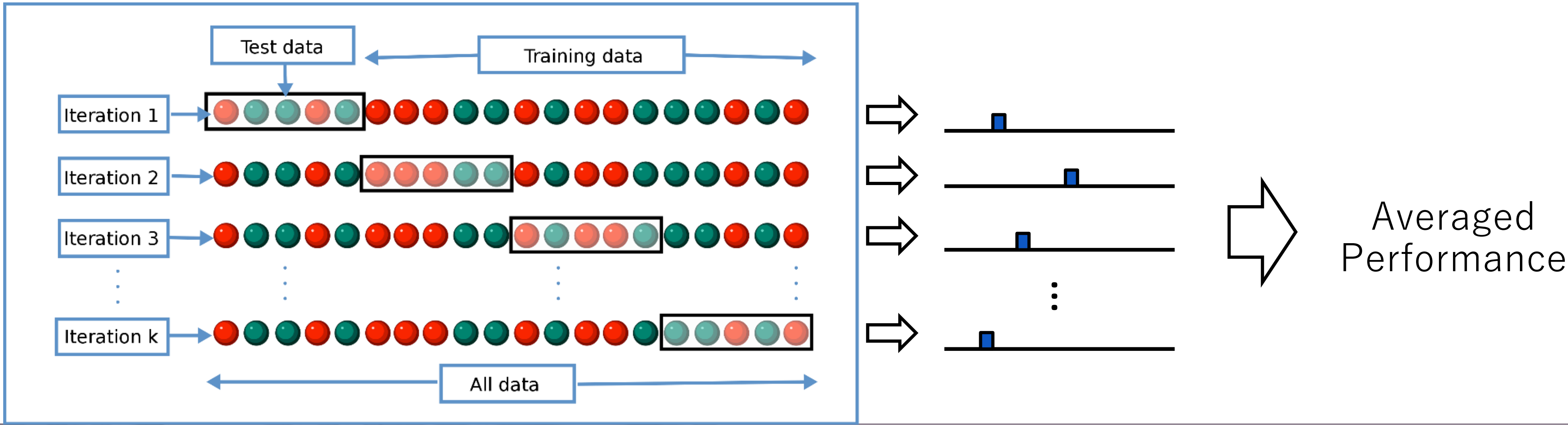
# Reliability of test performance:
# How much can we trust the estimated performance?

- Now you have 98% prediction accuracy on your test data
  … How much can you believe this?
  - Isn't it simply a lucky coincidence?
- Why not just repeat the random separation of training data and test data?



# A statistical framework for performance evaluation: Cross validation

- Divide a given dataset into K non-overlapping sets
  - Use $K - 1$ of them for training
  - Use the remaining one for testing
- Changing the test dataset results in $K$ measurements
  - Take their average to get a final performance estimate



# Model Selection

# Model selection:
# How can we tune the hyperparameters?

- We often have some hyper-parameters to be tuned so that the final performance gets better
  - E.g. Training target of the ridge regression:

Hyperparameter

$$\mathrm{minimize}_{\mathbf{w}}\ \|\mathbf{y} - \boldsymbol{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

  - Hyper-parameters are not optimized in the training
    - Joint optimization just gives a trivial solution $\lambda = 0$

# Statistical framework for tuning hyper-parameters: Cross validation (again)

- ($K$-fold) cross validation can also be used for determining hyper parameters
  - Use $K - 1$ of $K$ sets for training models for various hyper-parameter settings
  - Use the remaining one for testing
  - Choose the hyper-parameter setting with the best averaged performance
    - Note that this is **NOT** the estimate of its final performance

# Double-loop cross validation: Tuning hyper-parameters and performance evaluation at the same time

- Sometimes you want to do *both* hyper-parameter tuning and estimation of future performance
- Doing both with one $K$-fold cross validation is guilty 
  - You saw the test dataset for tuning hyper-parameters
- Double-loop cross validation:
  - Outer loop for performance evaluation
  - Inner loop for hyper-parameter tuning
  - High computational costs...

# A simple alternative of double-loop cross validation: “Development set” approach

- A simple alternative for the double-loop cross validation
- “Development set” approach
  - Use $K-2$ of $K$ sets for training
  - Use one for tuning hyper-parameters
  - Use one for testing