

Statistical Learning Theory

- Classification -

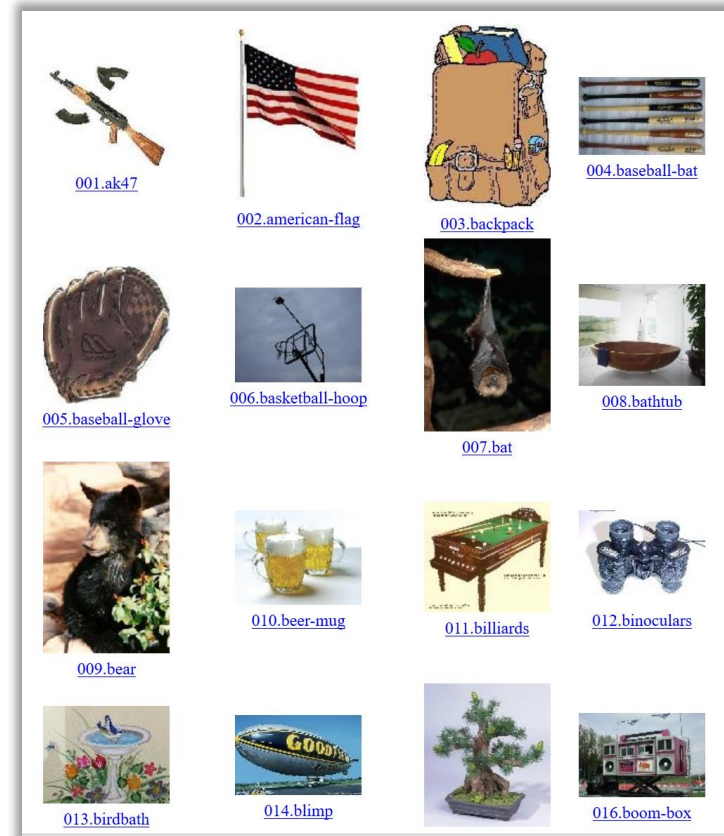
Hisashi Kashima

Classification

Classification:

Supervised learning for predicting discrete variable

- Goal: Obtain a function $f: \mathcal{X} \rightarrow \mathcal{Y}$ (\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
 $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$



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., {politics, economy, sports, ...}
 - Image classification: Categorize the object in an image \mathbf{x} into one of several object names, e.g., {AK5, American flag, backpack, ...}
 - Action recognition: Recognize the action type ({running, walking, sitting, ...}) that a person is taking from sensor data \mathbf{x}

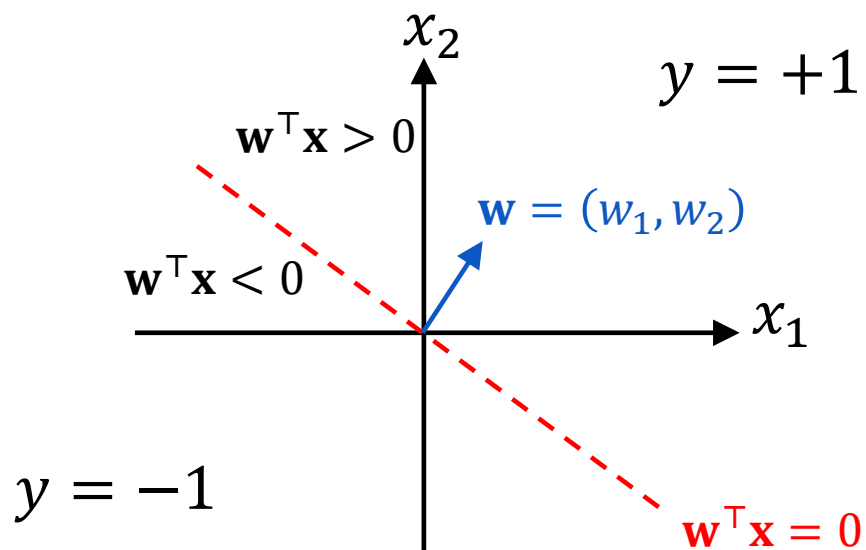
Model for classification:

Linear classifier

- Linear (binary) classification model:

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

- $|\mathbf{w}^\top \mathbf{x}|$ 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: $L(\mathbf{w}) = \sum_{i=1}^N \ell(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)})$

- Loss function ℓ : 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 : $\begin{cases} \text{Loss} = \text{Probabilistic model} \\ \text{Regularization} = \text{Prior} \end{cases}$

Classification problem in loss minimization framework:

Minimize loss function + regularization term

- Minimization problem: $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} L(\mathbf{w}) + R(\mathbf{w})$
 - 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:

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

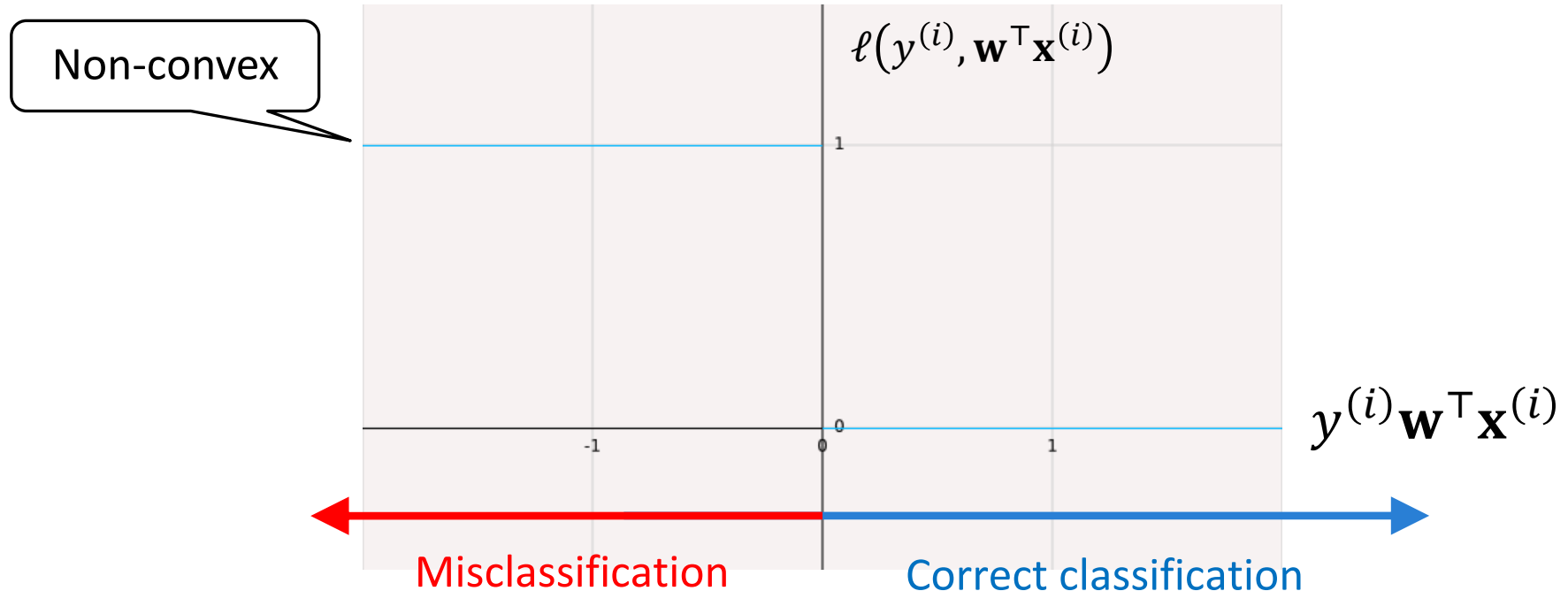
Correct classification

Incorrect classification

Zero-one loss:

Number of misclassification is hard to minimize

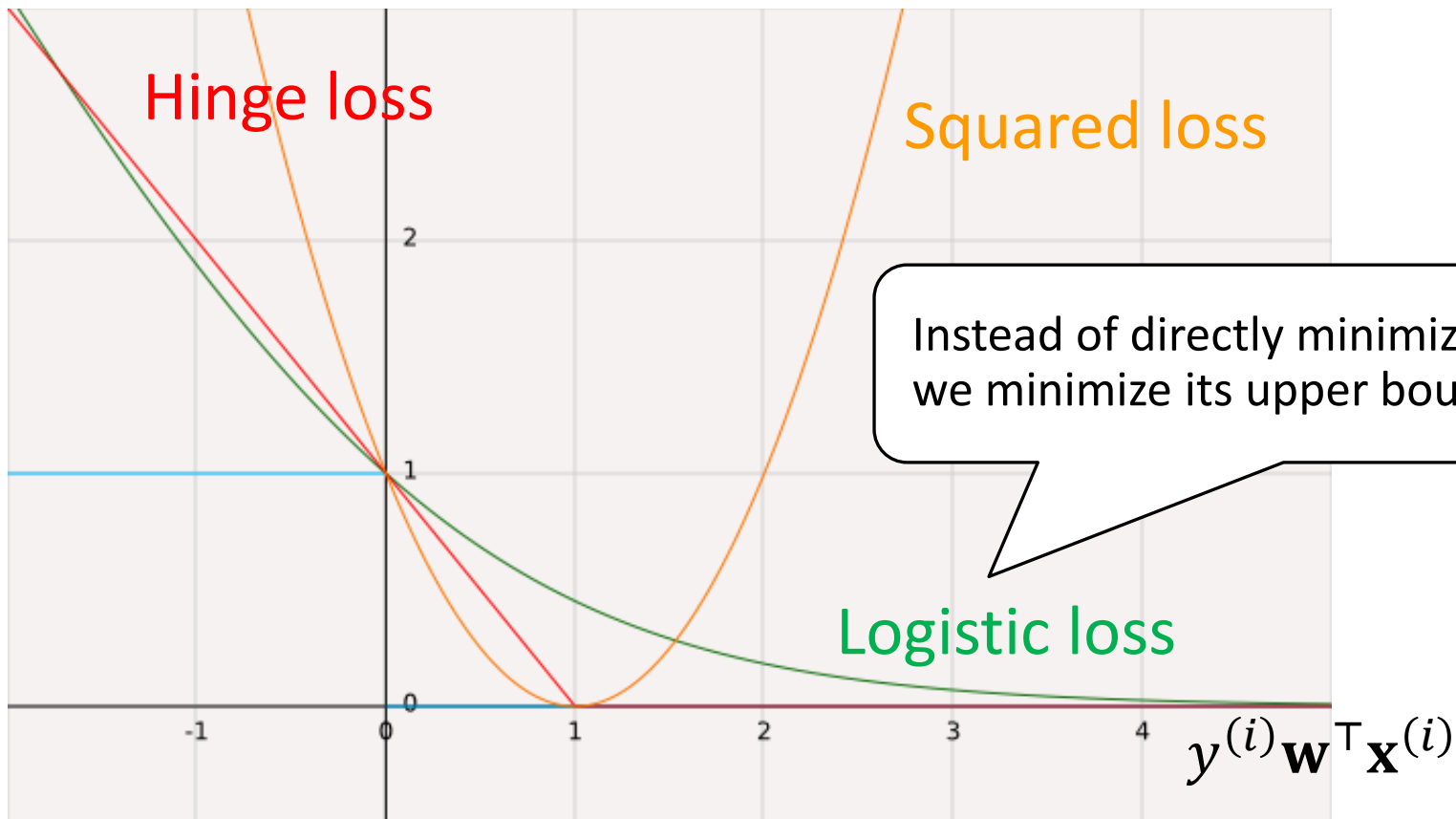
- Zero-one loss: $\ell(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}) = \begin{cases} 0 & (y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} > 0) \\ 1 & (y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \leq 0) \end{cases}$
- Non-convex function is hard to optimize directly



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:

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

- (Regularized) Logistic regression:

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

Convex



Statistical interpretation:

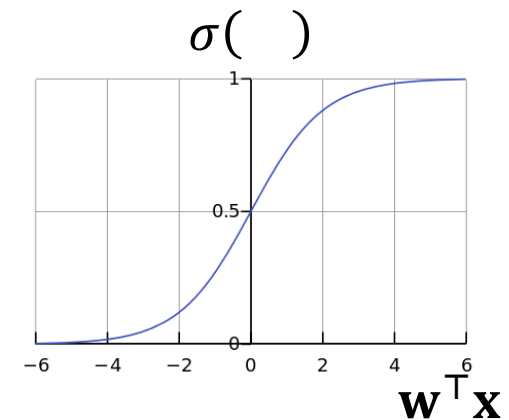
Logistic loss min. as MLE of logistic regression model

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

- Logistic regression model (conditional probability):

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

- σ : Logistic function ($\sigma: \mathbb{R} \rightarrow (0,1)$)



- Log likelihood:

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

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

Parameter estimation of logistic regression :

Numerical nonlinear optimization

- Objective function of (regularized) logistic regression:

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

- Minimization of logistic loss / MLE of logistic regression model has no closed form solution
- Numerical nonlinear optimization methods are used
 - Iterate parameter updates: $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} + \mathbf{d}$



Parameter update :

Find the best update minimizing the objective function

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

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

- Find \mathbf{d}^* that minimizes $L_{\mathbf{w}}(\mathbf{d})$:
 $-\mathbf{d}^* = \operatorname{argmin}_{\mathbf{d}} L_{\mathbf{w}}(\mathbf{d})$

Finding the best parameter update :

Approximate the objective with Taylor expansion

- Taylor expansion:

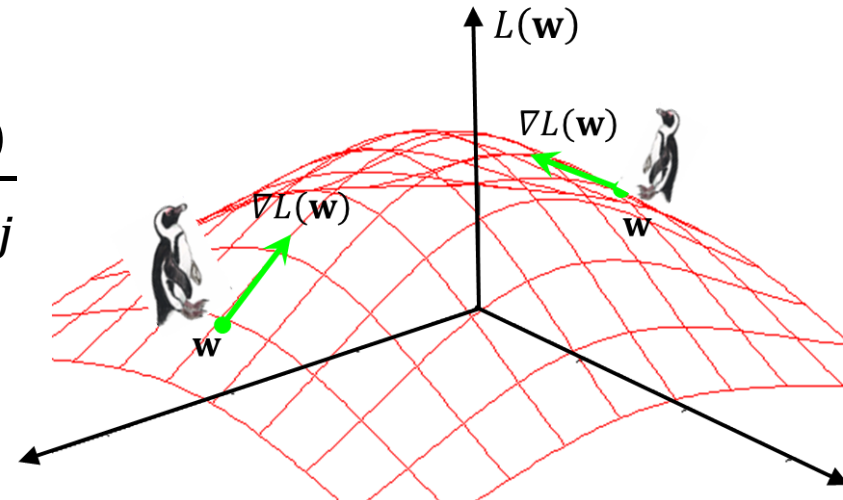
$$L_{\mathbf{w}}(\mathbf{d}) = L(\mathbf{w}) + \mathbf{d}^\top \nabla L(\mathbf{w}) + \frac{1}{2} \mathbf{d}^\top \mathbf{H}(\mathbf{w}) \mathbf{d} + O(\mathbf{d}^3)$$

3rd-order term

– 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):

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

- Derivative w.r.t. \mathbf{d} : $\frac{\partial L_{\mathbf{w}}(\mathbf{d})}{\partial \mathbf{d}} \approx \nabla L(\mathbf{w}) + \mathbf{H}(\mathbf{w}) \mathbf{d}$
- Setting it to be $\mathbf{0}$, we obtain $\mathbf{d} = -\mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$
- Newton update formula:

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



Modified Newton update:

Second order approximation + linear search

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

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

—This is not actually true for most cases

- Use only the direction of $\mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$ and update with $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$
- Learning rate $\eta > 0$ is determined by linear search:
$$\eta^* = \operatorname{argmax}_{\eta} L(\mathbf{w} - \eta \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w}))$$

(Steepest) gradient descent:

Simple update without computing inverse Hessian

- Computing **the inverse of Hessian matrix** is costly

- Newton update: $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$

- (Steepest) gradient descent:

- Replacing $\mathbf{H}(\mathbf{w})^{-1}$ with \mathbf{I} gives

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

Gradient of
objective function

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



Summary:

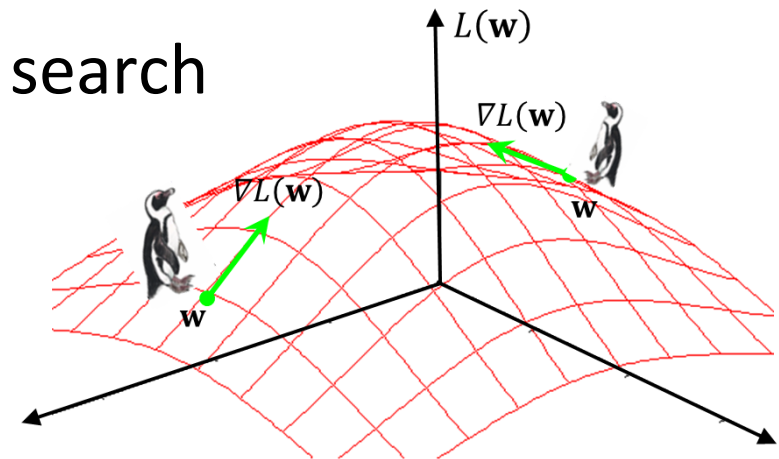
Gradient descent

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

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

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

– Learning rate η is determined by line search



Example of gradient descent: Gradient of logistic regression

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

- $$\begin{aligned} \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} &= \sum_{i=1}^N \frac{1}{1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})} \frac{\partial (1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}))}{\partial \mathbf{w}} \\ &= - \sum_{i=1}^N \frac{1}{1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})} \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}) y^{(i)} \mathbf{x}^{(i)} \\ &= - \sum_{i=1}^N (1 - f_{\mathbf{w}}(y^{(i)} | \mathbf{x}^{(i)})) y^{(i)} \mathbf{x}^{(i)} \end{aligned}$$

Can be easily computed with the current prediction probabilities

Mini batch optimization:

Efficient training using data subsets

- Objective function for N instances:

$$L(\mathbf{w}) = \sum_{i=1}^N \ell(\mathbf{w}^\top \mathbf{x}^{(i)}) + \lambda R(\mathbf{w})$$

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

- Approximate this with only one instance:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \approx N \frac{\partial \ell(\mathbf{w}^\top \mathbf{x}^{(j)})}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}} \quad (\text{Stochastic approximation})$$

- Also we can do this with $1 < M < N$ instances:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \approx \frac{N}{M} \sum_{j \in \text{MiniBatch}} \frac{\partial \ell(\mathbf{w}^\top \mathbf{x}^{(j)})}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}} \quad (\text{Mini batch})$$

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:

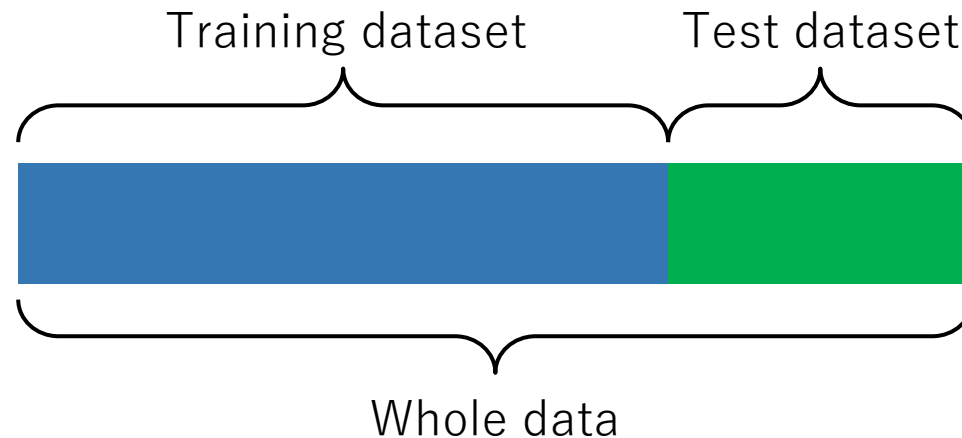
Evaluation must use a dataset not used in training

- 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

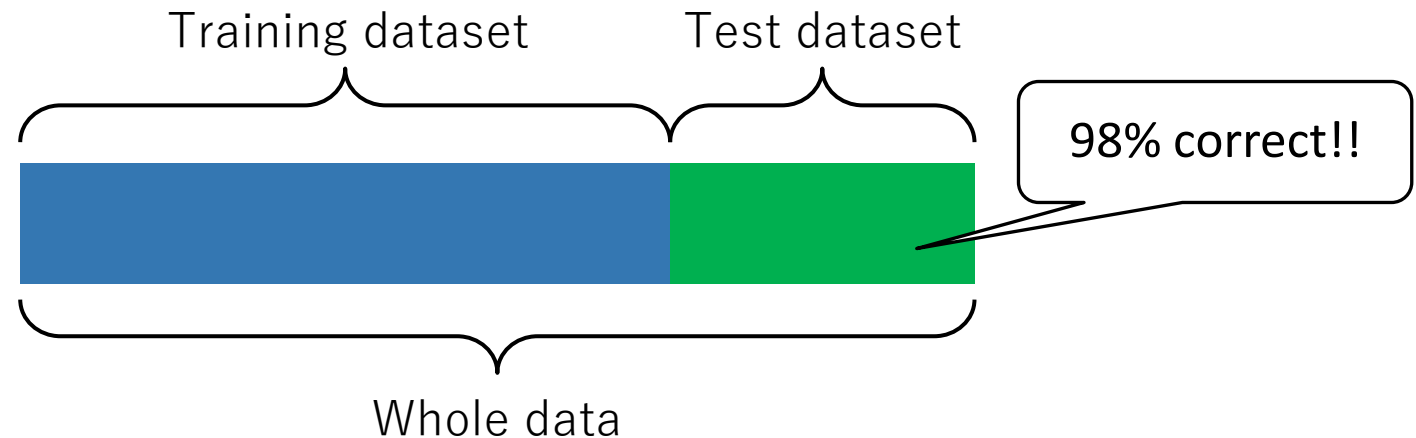
- Divide the dataset into a *training dataset* and a *test dataset*
 1. Train a classifier using the training dataset
 2. Evaluate its performance on the test dataset
- This is simulating a real application scenario using only the dataset at hand (without using real future data)



Reliability of test performance:

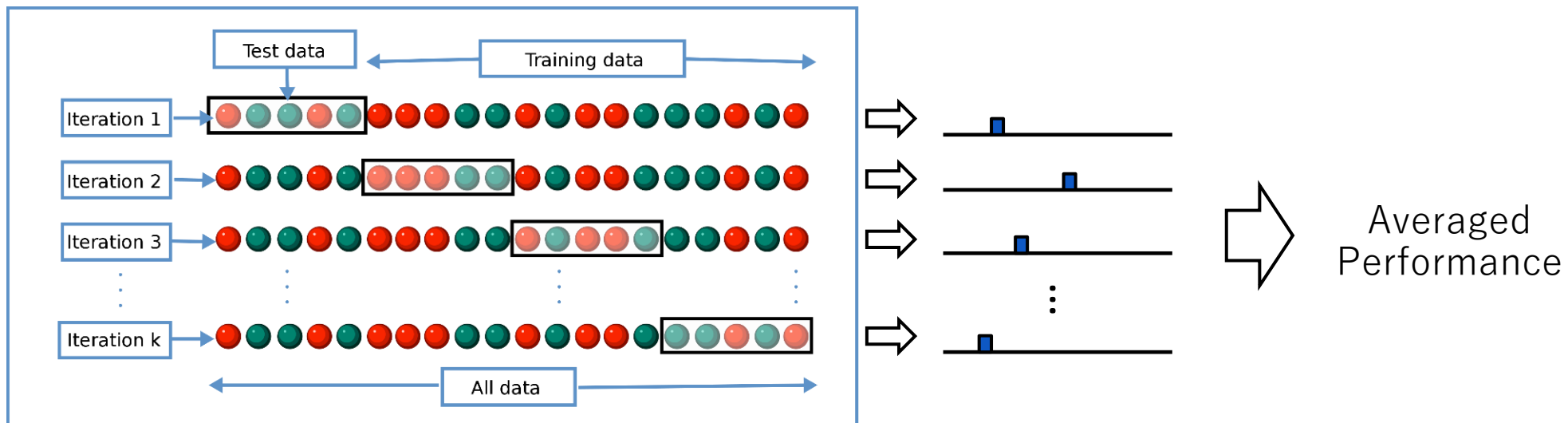
How much can we trust the estimated performance?

- 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

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


- 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

