Upcoming lecture schedule:
Advanced topics and hands-on practices

- Jun. 19 (today): Classification performance measure & non-linear models [Kashima]
- Jun. 26: Hands-on practice on regression and classification [Takeuchi]
- Jul. 3: Hands-on (cont'd) & Neural networks [Takeuchi]
- Jun. 10: Graph neural networks [Yamada]
- Jun. 24: Hands-on practice on neural networks and graph NN [Takeuchi]

* It is preferable (but not mandatory) that you bring your own PC with an Internet connection for the hands-on practices.

Kyoto University

Statistical Learning Theory
- Nonlinear Models -

Hisashi Kashima

Department of Intelligence Science and Technology

Contents:
Classification performance measures & nonlinear models

- Performance measures for classification:
  - Precision / Recall (depending on decision thresholds)
  - AUC (independent of decision thresholds)
- Nonlinear models:
  - Simple nonlinear transformation / cross-terms
  - Kernel methods:
    - kernel ridge regression
    - Kernel function: polynomial kernel, Gaussian kernel, …

Performance Measures for Classification

Various performance measures of classifiers: Accuracy, precision, recall, and AUC

- In supervised classification, we use various surrogate functions of 0/1-loss
  - Such as logistic loss, hinge loss, …
- In evaluation of a classifier, several performance measure are used
  - Performance measures depending on decision thresholds:
    - Accuracy, precision and recall, …
  - Performance measures independent of decision thresholds:
    - AUC

Confusion matrix:
Set of predictions on a dataset gives a confusion matrix

- A classifier makes positive ($+1$) or negative (-1) predictions
  - Linear classifier: $y = \mathrm{sign}\big(f(\mathbf{x})\big), f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$
  - The larger $f(\mathbf{x})$ is, the more strongly the classifier believes that $\mathbf{x}$ belongs to class $y = +1$
- Once we have a set of predictions on a dataset, we have a confusion matrix:

		predicted label	
		positive	negative
true label	positive	#true positives 😀	#false negatives
	negative	#false positives	#true negatives 😀

Basic performance measures : Accuracy, precision, recall

		predicted label	
		positive	negative
true label	positive	#true positives ☺	#false negatives
	negative	#false positives	#true negatives ☺

- Accuracy: percentage of $\frac{\#\text{true positives} + \#\text{true negatives}}{\#\text{all predictions}}$
  - In other words, averaged 0-1 loss
- Precision & Recall:

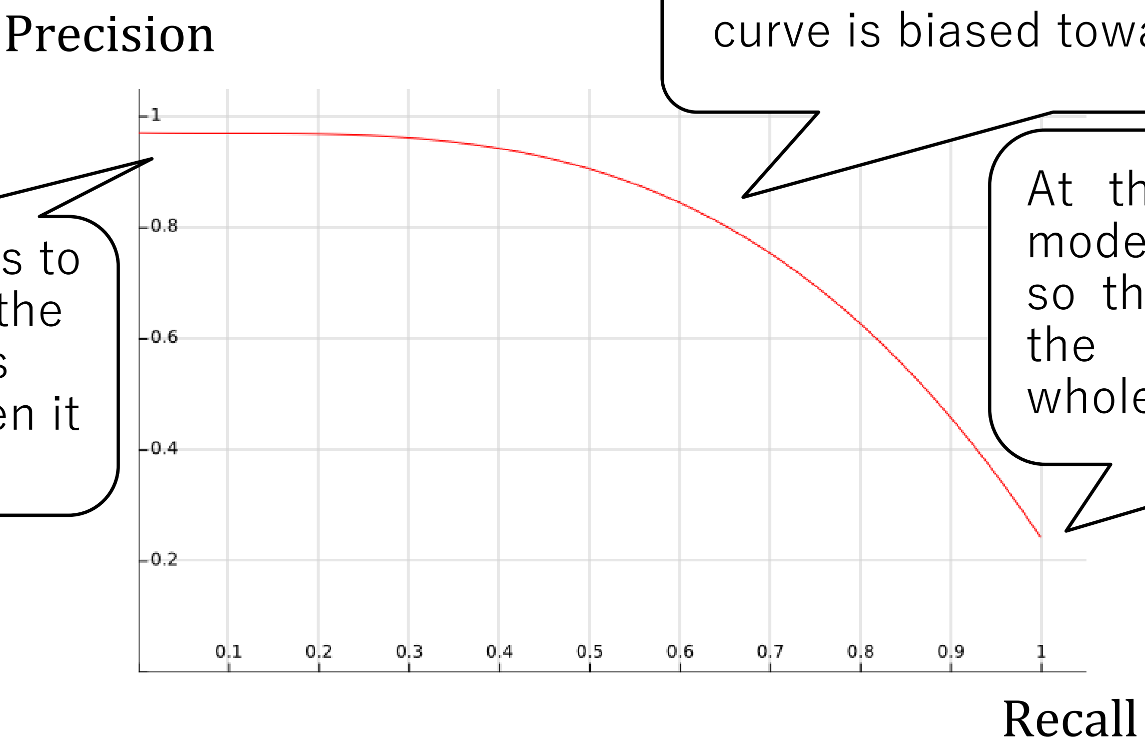
Wherever he goes, there's always a murder.

  - $\text{Precision} = \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false positives}}$
  - $\text{Recall} = \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false negatives}}$
  - $\text{F-measure} = \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$
    - Harmonic mean of precision and recall

Wherever there's a murder, he’s always there.

Precision-recall curve: View changes in precision & recall with different thresholds

- Changing the threshold gives different precision and recall



Precision tends to be high when the model predicts $Y = 1$ only when it is confident

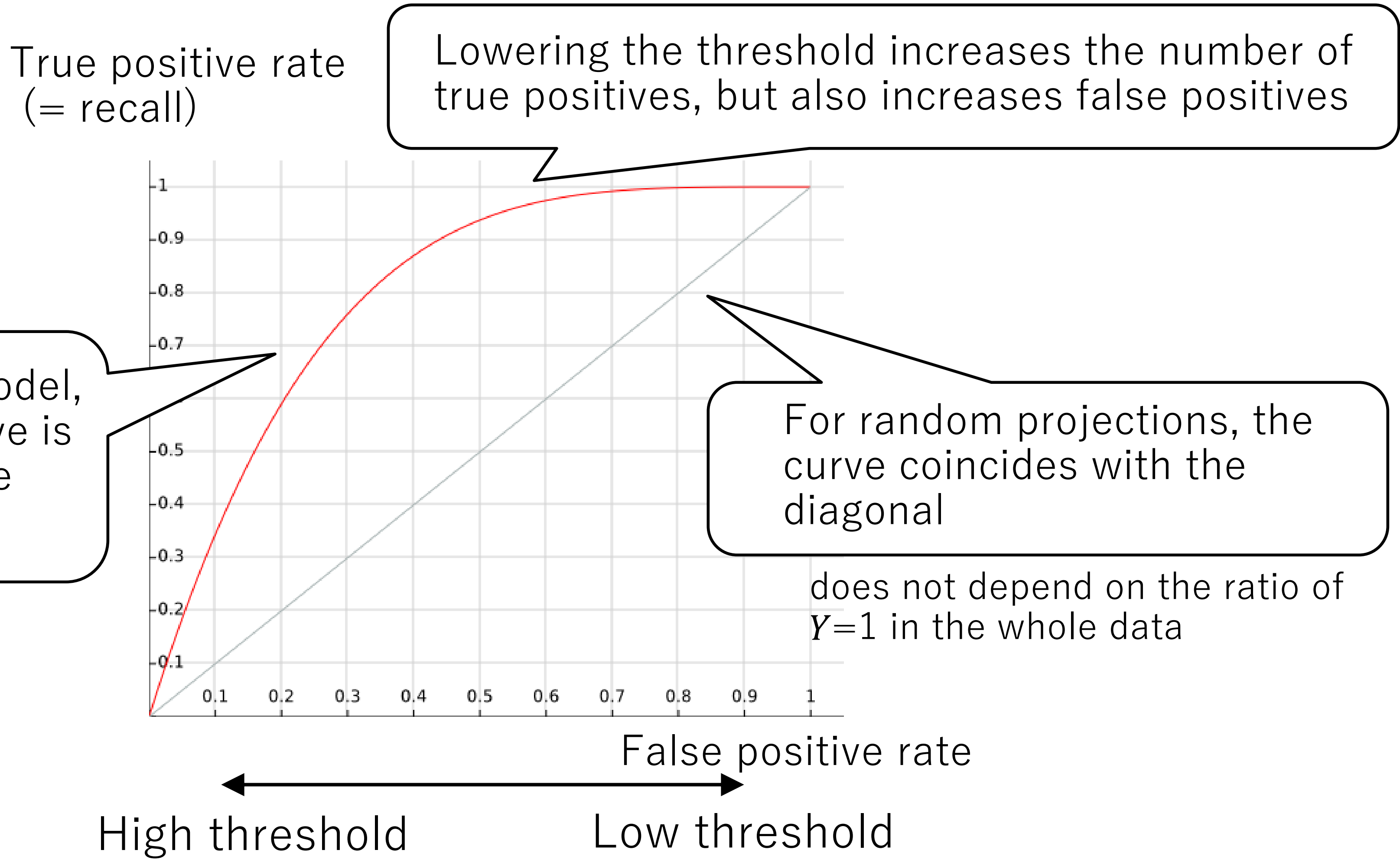
The better the model, the more the curve is biased toward the upper right

At the least threshold, the model always predicts $Y = 1$, so the precision approaches the ratio of $Y = 1$ in the whole data

High threshold ⟷ Low threshold

ROC-curve: View changes in true-positives & false-positives with different thresholds

- Yet another (and popular) performance curve

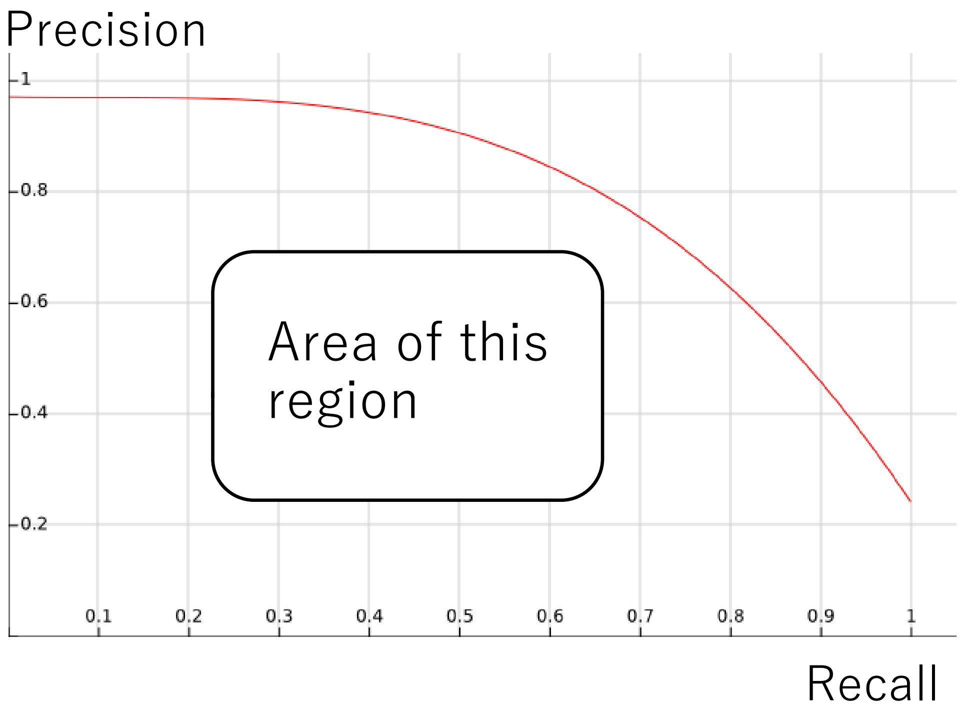


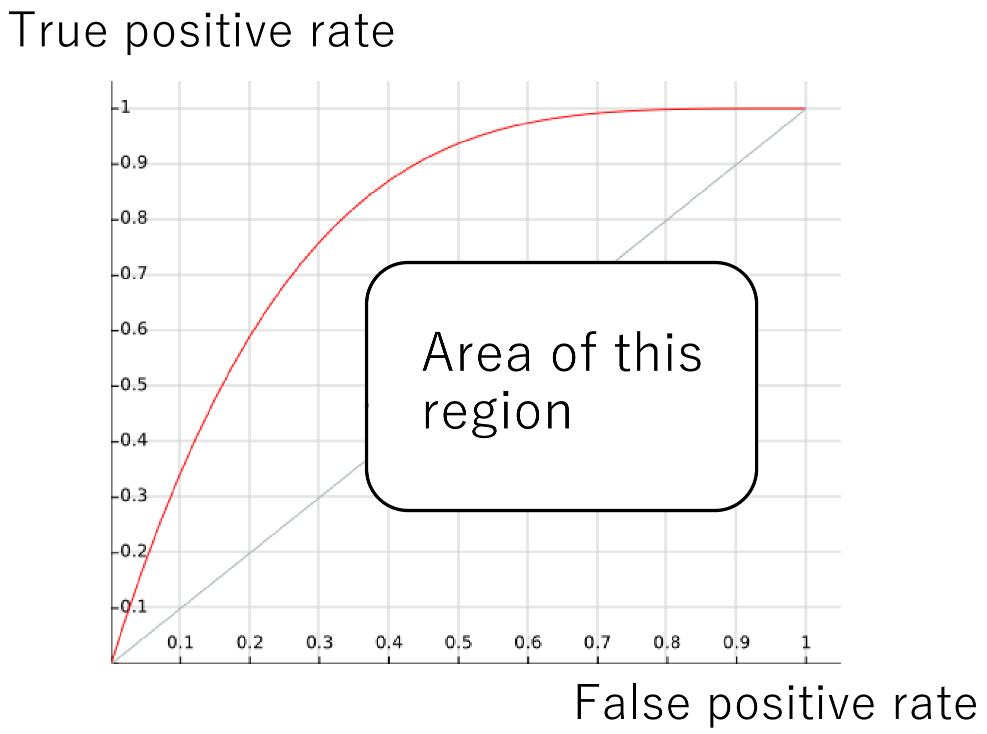
does not depend on the ratio of Y=1 in the whole data

Area under the curve: Performance measures *independent* of thresholds

- The area under the PR-curve (PR-AUC)
- The area under the ROC-curve (ROC-AUC)
  - ROC-AUC is not affected by class (im)balance

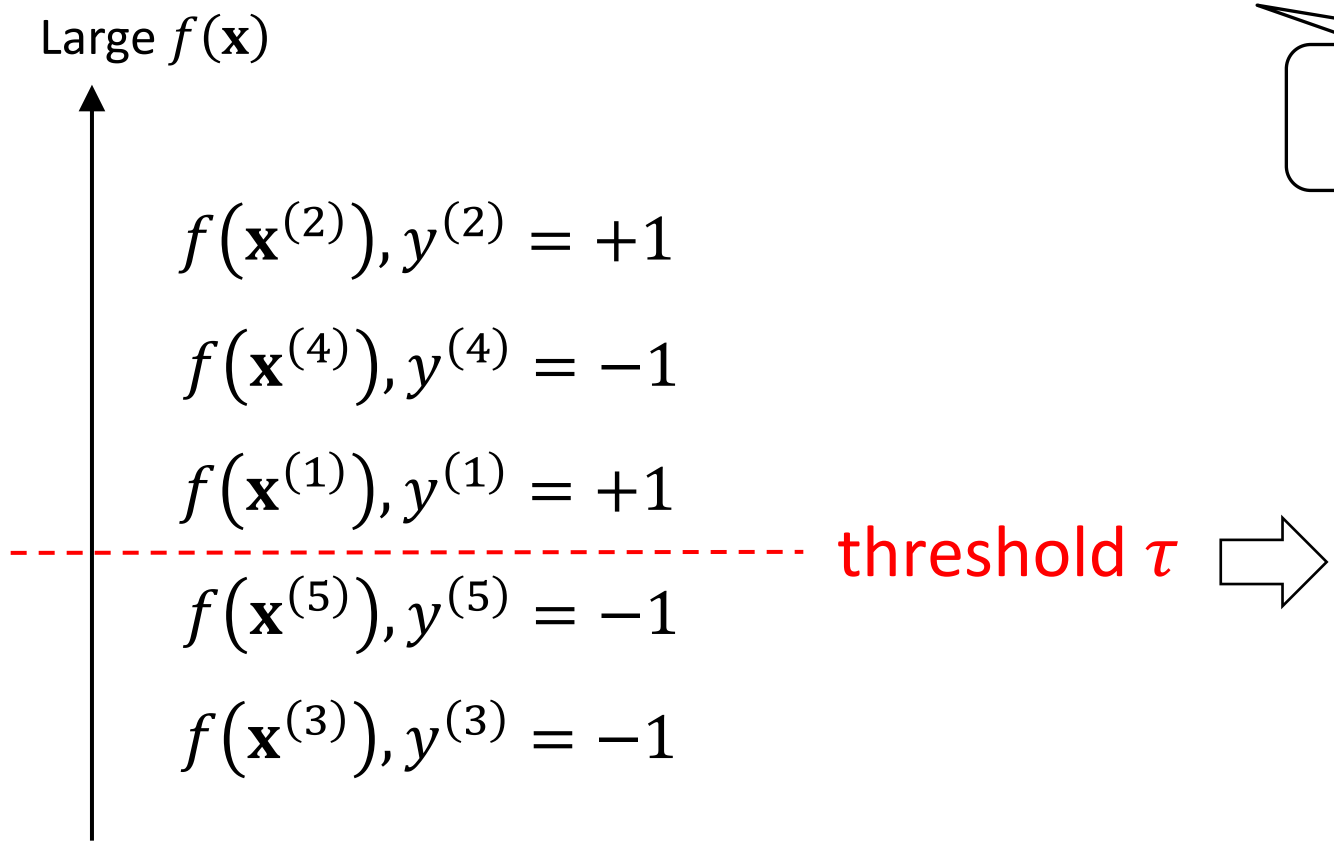
When we simply say "AUC", we usually mean this





Computational complexity of the performance measures: Sorting the model predictions

- Complexity of drawing {PR, ROC}-{curve, AUC} is equivalent to that of sorting the prediction scores $f(\mathbf{x})$ in descending order


Large $f(\mathbf{x})$
$f(\mathbf{x}^{(2)}), y^{(2)} = +1$
$f(\mathbf{x}^{(4)}), y^{(4)} = -1$
$f(\mathbf{x}^{(1)}), y^{(1)} = +1$
threshold τ
$f(\mathbf{x}^{(5)}), y^{(5)} = -1$
$f(\mathbf{x}^{(3)}), y^{(3)} = -1$

$\mathrm{O}(n \log n)$

Precision=2/3

Recall=2/2

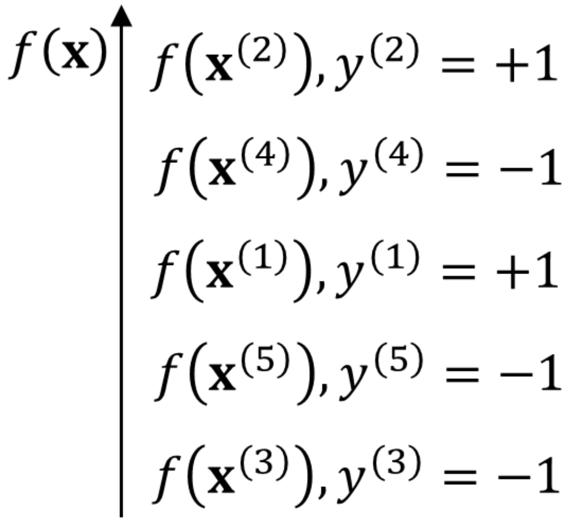
True positive rate=2/2

(=recall)

False positive rate=1/3

Another implication of ROC-AUC: ROC-AUC measures ordering correctness

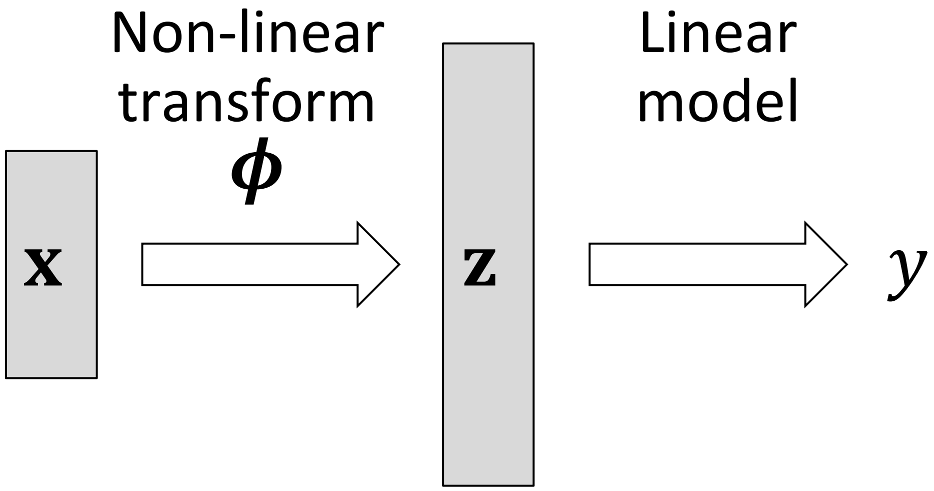
- ROC-AUC: Proportion of (i, j) pairs satisfying $y^{(i)} = +1, y^{(j)} = -1$, and $f\left(\mathbf{x}^{(i)}\right) > f\left(\mathbf{x}^{(j)}\right)$
- It checks that the test data are ranked in the correct order by f
  - AUC=1: Perfect ranking
  - AUC=0.5: Completely random ranking
  - AUC=0: Perfectly reversed ranking
- Example: AUC=5/6
  - Among $2 \times 3 = 6$ pos-neg pairs
  - 5 pairs are in correct order


$f(\mathbf{x})$
$f\left(\mathbf{x}^{(2)}\right), y^{(2)} = +1$
$f\left(\mathbf{x}^{(4)}\right), y^{(4)} = -1$
$f\left(\mathbf{x}^{(1)}\right), y^{(1)} = +1$
$f\left(\mathbf{x}^{(5)}\right), y^{(5)} = -1$
$f\left(\mathbf{x}^{(3)}\right), y^{(3)} = -1$

Nonlinear Models

Transformation-based nonlinear models: Apply nonlinear transform before applying linear model

- Input vector $\mathbf{x} \in \mathbb{R}^D$ is transformed to a new vector $\mathbf{z} \in \mathbb{R}^{D'}$ using some nonlinear transformation function $\boldsymbol{\phi}: \mathbb{R}^D \to \mathbb{R}^{D'}$
- Linear model is applied to $\mathbf{z}$:
  - Linear regression $y = \mathbf{v}^\top \mathbf{z}$, logistic regression $y = \sigma(\mathbf{v}^\top \mathbf{z})$



Nonlinear regression: Introducing nonlinearity in linear models

- So far we have considered only linear models:
  - Linear regression $y = \mathbf{w}^\top \mathbf{x}$, logistic regression $y = \sigma(\mathbf{w}^\top \mathbf{x})$
- How to introduce non-linearity in the models?

1. Use of inherently nonlinear models:
   - Decision/regression tree, random forest, boosting trees
2. Transformation-based approaches:
   - Nonlinear feature transformation
   - Kernel methods
   - Neural networks

Nonlinear transformation of features: Simplest way to introduce nonlinearity in linear models

- Apply non-linear transformation:
  - $\mathbf{x} = (x) \Rightarrow \mathbf{z} = \left(\log x\,, e^{x}, x^{2}, \frac{1}{x}, \ldots\right)^{\top}$
  - $y = wx \Rightarrow y = w_1 \log x + w_2 e^{x}, +w_3\ x^{2} + w_4 \frac{1}{x} + \cdots$
- It is up to the user to decide which transformations to use.

Cross terms & factorization machine: Can include synergetic effects among different features

- Use cross terms products $\{x_d x_{d'}\}_{d,d'}$ of $x_1, x_2, \ldots, x_D$
- Model has a matrix parameter $\boldsymbol{W}$:

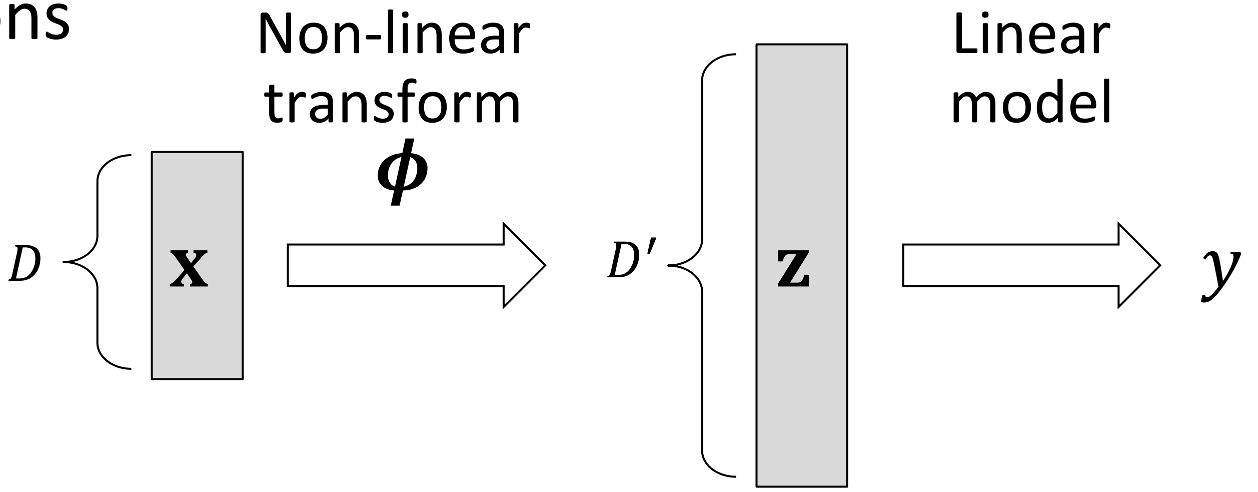
$$y = \mathrm{Trace}\left(\begin{bmatrix} w_{1,1} & \cdots & w_{1,D} \\ \vdots & \ddots & \vdots \\ w_{D,1} & \cdots & w_{D,D} \end{bmatrix}^\top \begin{bmatrix} x_1^2 & x_1 x_2 & \cdots & x_1 x_D \\ x_2 x_1 & x_2^2 & & x_2 x_D \\ \vdots & & \ddots & \vdots \\ x_D x_1 & x_D x_2 & \cdots & x_D^2 \end{bmatrix} \right) = \mathbf{x}^\top \boldsymbol{W}^\top \mathbf{x}$$

- Loss function: $L(\boldsymbol{W}) = \sum_{i=1}^{N} \left(y^{(i)} - \mathbf{x}^{(i)\top} \boldsymbol{W}^\top \mathbf{x}^{(i)} \right)^2$
- Assuming low rankness of $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{U}^\top$ leads to factorization machine ($O(DK)$ parameters instead of $O(D^2)$)

Kernel Methods

Kernels: Linear model in a high-dimensional feature space

- Kernel method is a general framework to convert a linear machine to non-linear machine
- High dimensional non-linear mapping: $\mathbf{x} \rightarrow \mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$
- Consider a linear model $y = \mathbf{v}^\top \mathbf{z}$ in the high dim. space
- Resolves computational difficulties caused by high dimensionality through kernel functions



“Dual form” of linear regression model: Representation using only inner products of input vectors

- Let us construct a “kernel version” of linear regression
- Linear regression: $y = \mathbf{w}^\top \mathbf{x}$

  D dimensional model

  - Training data: $\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$
  - Objective function: $L(\mathbf{w}) = \|\mathbf{y} - \boldsymbol{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$
  - Solution: $\mathbf{w}^* = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\top\mathbf{y}$

  Dealing with $D \times D$ matrix

- The computational costs are governed by D

"Dual form" of linear regression model: Representation using only inner products of input vectors

- Now we assume $\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}^{(j)}$ (weighted sum of inputs)
  - (For the time being, we accept this without reason)
  - $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)^\top$: a new N-dimensional parameter
- We have "kernel ridge regression":
  - Model: $y = \sum_{i=1}^{N} \alpha_i \langle \mathbf{x}^{(j)}, \mathbf{x} \rangle$

    N dimensional model
  - Objective function: $L(\mathbf{w}) = \|\mathbf{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}$
  - Solution: $\mathbf{w}^* = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \mathbf{y}$

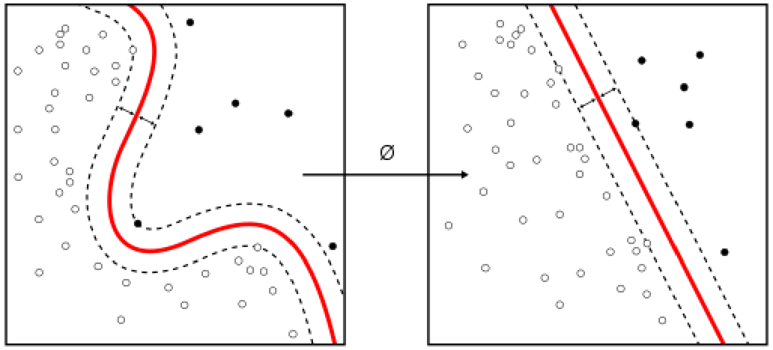
    Dealing with $N \times N$ matrix
  - $\boldsymbol{K} = [\boldsymbol{K}_{i,j}] = [\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle]$ (Kernel matrix)

Advantage of kernel methods: Computational costs depending on the number of training data

- Now we have a “dual form” of the ridge regression
- What is nice about the kernel ridge regression?
  - Model/problem size depend on the size of the training data N instead of the number of dimensions D
  - Computational advantage when $D > N$
- Note: Kernel machines access data only through kernel functions (= inner products between data)

Kernel functions: Introducing non-linearity in linear models

- Now we consider non-linear regression
- Introduce a (nonlinear) mapping $\boldsymbol{\phi}: \mathbb{R}^D \to \mathbb{R}^{D'}$
  - D-dimensional space to $D'(\gg D)$-dimensional space
  - Vector $\mathbf{x}$ is mapped to a high-dimensional vector $\boldsymbol{\phi}(\mathbf{x})$
- Define kernel function $K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \Rightarrow \left\langle \boldsymbol{\phi}\left(\mathbf{x}^{(i)}\right), \boldsymbol{\phi}\left(\mathbf{x}^{(j)}\right) \right\rangle$ in the D'-dimensional space



https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Kernel_Machine.svg

Advantage of kernel methods: Computationally efficient (when D' is large)

- Advantage of using kernel function:

$$K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \langle \boldsymbol{\phi}(\mathbf{x}^{(i)}), \boldsymbol{\phi}(\mathbf{x}^{(j)}) \rangle$$

- Usually we expect the computation cost of K depends on D'
  - D' can be high-dimensional (possibly infinite dimensional)
- If we can somehow compute $\langle \boldsymbol{\phi}(\mathbf{x}^{(i)}), \boldsymbol{\phi}(\mathbf{x}^{(j)}) \rangle$ in time depending on D, the dimension of $\boldsymbol{\phi}$ does not matter
- Problem size:

$$D'(\text{number of dimensions}) \rightarrow N(\text{number of data})$$

  - Advantageous when D' is very large or infinite

Example of kernel functions: Polynomial kernel can consider high-order cross terms

- Combinatorial features: Not only the original features $x_1, x_2, \dots, x_D$, we use their cross terms (e.g. $x_1 x_2$)
  - If we consider M-th order cross terms, we have $\mathrm{O}(D^M)$ terms
- Polynomial kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left({\mathbf{x}^{(i)}}^\top \mathbf{x}^{(j)} + c\right)^M$
  - E.g. when $c = 0, M = 2, D = 2$,

$$\mathbf{x}^{(i)} = \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix}$$

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left(x_1^{(i)} x_1^{(j)} + x_2^{(i)} x_2^{(j)}\right)^2$$

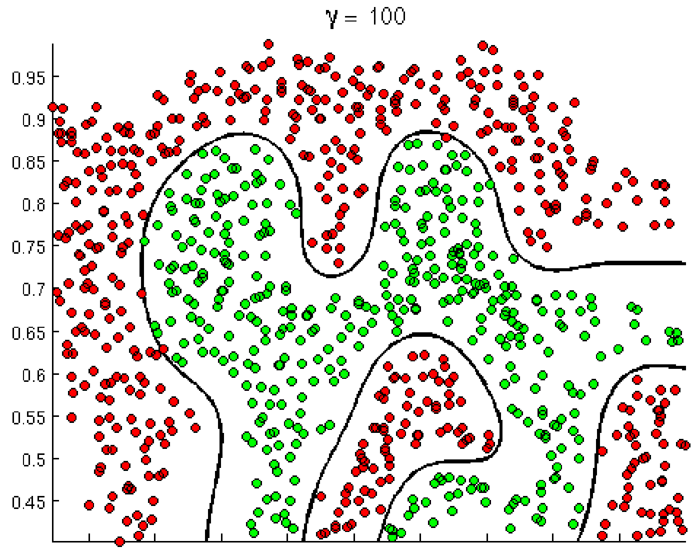
$$= \left({x_1^{(i)}}^2, {x_2^{(i)}}^2, \sqrt{2} x_1^{(i)} x_2^{(i)}\right)\left({x_1^{(j)}}^2, {x_2^{(j)}}^2, \sqrt{2} x_1^{(j)} x_2^{(j)}\right)$$

  - Can be computed in $\mathrm{O}(D)$!!

Example of kernel functions: Gaussian kernel with infinite feature space

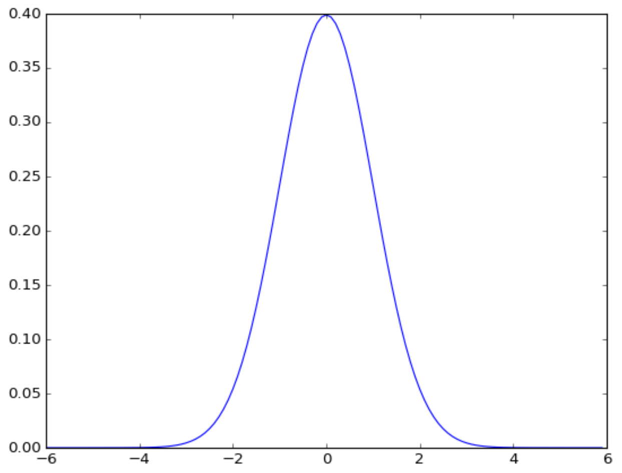
- Gaussian kernel: $K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|_2^2}{\sigma}\right)$
  - Can be interpreted as an inner product in an infinite-dimensional space

Discrimination surface with Gaussian kernel



http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html

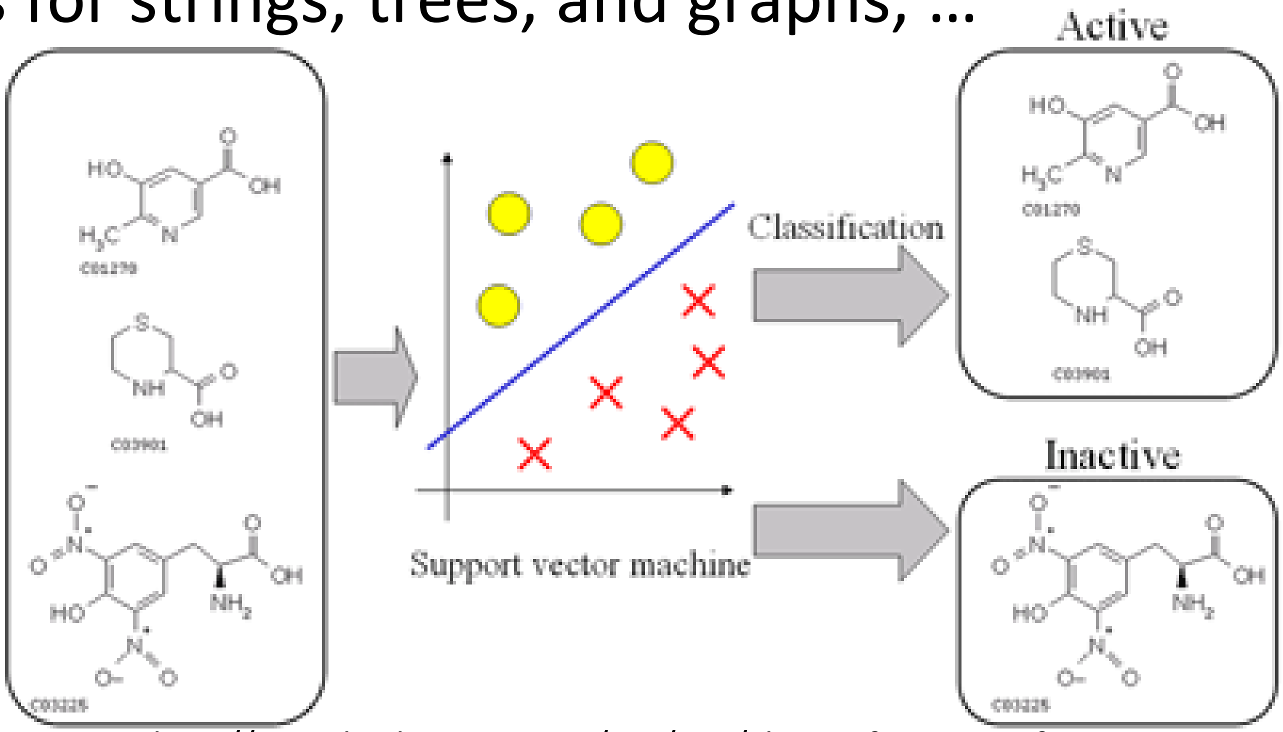
Gaussian kernel (RBF kernel)



$\| \mathbf{x}_i - \mathbf{x}_j \|_2^2$

Kernel methods for non-vectorial data: Kernels for sequences, trees, and graphs

- Kernel methods can handle any kinds of objects (even non-vectorial objects) as long as efficiently computable kernel functions are available
  - Kernels for strings, trees, and graphs, …



http://www.bic.kyoto-u.ac.jp/coe/img/akutsu_fig_e_02.gif

Representer theorem: Theoretical underpinning of kernel methods

- Can I use a similarity function that I created by myself as a kernel function?
  - Yes (under *certain conditions*)
- Kernel methods are derived from the assumption that the optimal parameter is represented as a linear combination of input vectors:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}^{(i)}$$

When does it hold?

- Representer theorem guarantees this (if we use L2-regularizer)

(Simple) proof of representer theorem: Obj. func. depends only on linear combination of inputs

- Assumption: Loss ℓ for i-th data depends only on $\mathbf{w}^\top \mathbf{x}^{(i)}$
  - Objective function: $L(\mathbf{w}) = \sum_{i=1}^{N} \ell\left(\mathbf{w}^\top \mathbf{x}^{(i)}\right) + \lambda \|\mathbf{w}\|_2^2$
- Divide the optimal parameter $\mathbf{w}^*$ into two parts $\mathbf{w} + \mathbf{w}^\perp$:
  - $\mathbf{w}$: Linear combination of input data $\left\{\mathbf{x}^{(i)}\right\}_i$
  - $\mathbf{w}^\perp$: Other parts (orthogonal to all input data $\left\{\mathbf{x}^{(i)}\right\}$)
- $L(\mathbf{w}^*)$ depends only on $\mathbf{w}$: $\sum_{i=1}^{N} \ell\left(\mathbf{w}^{*\top} \mathbf{x}^{(i)}\right) + \lambda \|\mathbf{w}^*\|_2^2$

$$= \sum_{i=1}^{N} \ell\left(\mathbf{w}^\top \mathbf{x}^{(i)} + \underbrace{\mathbf{w}^{\perp\top} \mathbf{x}^{(i)}}_{=0}\right) + \lambda\left(\|\mathbf{w}\|_2^2 + \underbrace{2\mathbf{w}^\top \mathbf{w}^\perp}_{=0} + \underbrace{\|\mathbf{w}^\perp\|_2^2)}_{\text{Minimized to } = 0}$$