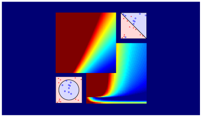
Machine Learning Foundations
(機器學習基石)



Lecture 11: Linear Models for Classification

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. When Can Machines Learn?
2. Why Can Machines Learn?
3. **How** Can Machines Learn?

Lecture 10: Logistic Regression

gradient descent on **cross-entropy error**
to get good **logistic hypothesis**

Lecture 11: Linear Models for Classification

- Linear Models for Binary Classification
- Stochastic Gradient Descent
- Multiclass via Logistic Regression
- Multiclass via Binary Classification

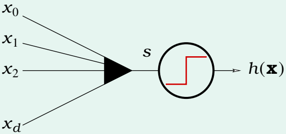
4. How Can Machines Learn Better?

Linear Models Revisited

linear scoring function: $s = \mathbf{w}^T\mathbf{x}$

linear classification

$h(\mathbf{x}) = \text{sign}(s)$

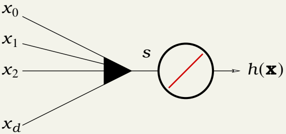


plausible err = 0/1

discrete $E_{\text{in}}(\mathbf{w})$:
NP-hard to solve

linear regression

$h(\mathbf{x}) = s$

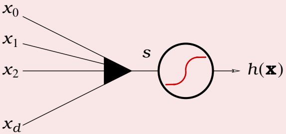


friendly err = squared

quadratic convex $E_{\text{in}}(\mathbf{w})$:
closed-form solution

logistic regression

$h(\mathbf{x}) = \theta(s)$



plausible err = cross-entropy

smooth convex $E_{\text{in}}(\mathbf{w})$:
gradient descent

can linear regression or logistic regression **help linear classification**?

Error Functions Revisited

linear scoring function: $s = \mathbf{w}^T\mathbf{x}$

for binary classification $y \in \{-1, +1\}$

linear classification

$$\begin{aligned} h(\mathbf{x}) &= \text{sign}(s) \\ \text{err}(h, \mathbf{x}, y) &= [\![h(\mathbf{x}) \neq y]\!] \end{aligned}$$

$$\begin{aligned} &\text{err}_{0/1}(s, y) \\ =\;& [\![\text{sign}(s) \neq y]\!] \\ =\;& [\![\text{sign}(ys) \neq 1]\!] \end{aligned}$$

linear regression

$$\begin{aligned} h(\mathbf{x}) &= s \\ \text{err}(h, \mathbf{x}, y) &= (h(\mathbf{x}) - y)^2 \end{aligned}$$

$$\begin{aligned} &\text{err}_{\text{SQR}}(s, y) \\ =\;& (s - y)^2 \\ =\;& (ys - 1)^2 \end{aligned}$$

logistic regression

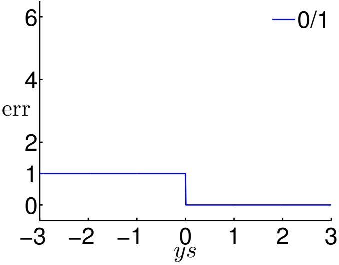
$$\begin{aligned} h(\mathbf{x}) &= \theta(s) \\ \text{err}(h, \mathbf{x}, y) &= -\ln h(y\mathbf{x}) \end{aligned}$$

$$\begin{aligned} &\text{err}_{\text{CE}}(s, y) \\ =\;& \ln(1 + \exp(-ys)) \end{aligned}$$

(ys): classification correctness score

Visualizing Error Functions

$$
\begin{array}{rrcl}
0/1 & \mathrm{err}_{0/1}(s, y) & = & [\![\mathrm{sign}(ys) \neq 1]\!] \\
\mathrm{sqr} & \mathrm{err}_{\mathrm{SQR}}(s, y) & = & (ys - 1)^2 \\
\mathrm{ce} & \mathrm{err}_{\mathrm{CE}}(s, y) & = & \ln(1 + \exp(-ys)) \\
\mathrm{scaled\ ce} & \mathrm{err}_{\mathrm{SCE}}(s, y) & = & \log_2(1 + \exp(-ys))
\end{array}
$$

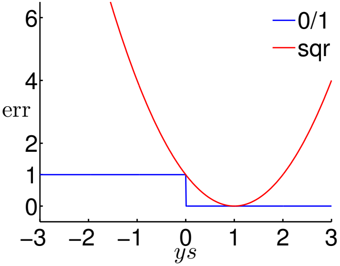


- 0/1: 1 iff $ys \leq 0$
- sqr: large if $ys \ll 1$
  but over-charge $ys \gg 1$
  small $\mathrm{err}_{\mathrm{SQR}} \rightarrow$ small $\mathrm{err}_{0/1}$
- ce: monotonic of ys
  small $\mathrm{err}_{\mathrm{CE}} \leftrightarrow$ small $\mathrm{err}_{0/1}$
- scaled ce: a proper upper bound of 0/1
  small $\mathrm{err}_{\mathrm{SCE}} \leftrightarrow$ small $\mathrm{err}_{0/1}$

upper bound:
useful for designing algorithmic error $\widehat{\mathrm{err}}$

Visualizing Error Functions

$$
\begin{array}{rrcl}
\text{0/1} & \mathrm{err}_{0/1}(s,y) & = & [\![\mathrm{sign}(ys) \neq 1]\!] \\
\text{sqr} & \mathrm{err}_{\mathrm{SQR}}(s,y) & = & (ys-1)^2 \\
\text{ce} & \mathrm{err}_{\mathrm{CE}}(s,y) & = & \ln(1+\exp(-ys)) \\
\text{scaled ce} & \mathrm{err}_{\mathrm{SCE}}(s,y) & = & \log_2(1+\exp(-ys))
\end{array}
$$

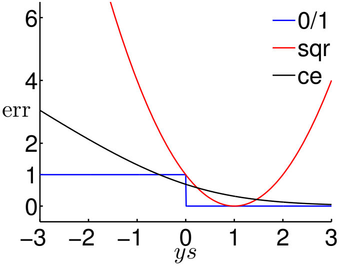


- 0/1: 1 iff $ys \leq 0$
- sqr: large if $ys \ll 1$
  but over-charge $ys \gg 1$
  small $\mathrm{err}_{\mathrm{SQR}} \to$ small $\mathrm{err}_{0/1}$
- ce: monotonic of ys
  small $\mathrm{err}_{\mathrm{CE}} \leftrightarrow$ small $\mathrm{err}_{0/1}$
- scaled ce: a proper upper bound of 0/1
  small $\mathrm{err}_{\mathrm{SCE}} \leftrightarrow$ small $\mathrm{err}_{0/1}$

upper bound:
useful for designing algorithmic error $\widehat{\mathrm{err}}$

Visualizing Error Functions

$$
\begin{array}{rrcl}
0/1 & \mathrm{err}_{0/1}(s, y) & = & [\![\mathrm{sign}(ys) \neq 1]\!] \\
\mathrm{sqr} & \mathrm{err}_{\mathrm{SQR}}(s, y) & = & (ys - 1)^2 \\
\mathrm{ce} & \mathrm{err}_{\mathrm{CE}}(s, y) & = & \ln(1 + \exp(-ys)) \\
\text{scaled ce} & \mathrm{err}_{\mathrm{SCE}}(s, y) & = & \log_2(1 + \exp(-ys))
\end{array}
$$

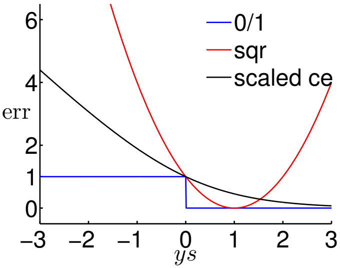


- 0/1: 1 iff $ys \leq 0$
- sqr: large if $ys \ll 1$
  but over-charge $ys \gg 1$
  small $\mathrm{err}_{\mathrm{SQR}}$ → small $\mathrm{err}_{0/1}$
- ce: monotonic of ys
  small $\mathrm{err}_{\mathrm{CE}}$ ↔ small $\mathrm{err}_{0/1}$
- scaled ce: a proper upper bound of 0/1
  small $\mathrm{err}_{\mathrm{SCE}}$ ↔ small $\mathrm{err}_{0/1}$

upper bound:
useful for designing algorithmic error $\widehat{\mathrm{err}}$

Visualizing Error Functions

$$\begin{array}{rrcl}
\text{0/1} & \mathrm{err}_{0/1}(s,y) & = & [\![\mathrm{sign}(ys) \neq 1]\!] \\
\text{sqr} & \mathrm{err}_{\mathrm{SQR}}(s,y) & = & (ys-1)^2 \\
\text{ce} & \mathrm{err}_{\mathrm{CE}}(s,y) & = & \ln(1+\exp(-ys)) \\
\text{scaled ce} & \mathrm{err}_{\mathrm{SCE}}(s,y) & = & \log_2(1+\exp(-ys))
\end{array}$$



- 0/1: 1 iff $ys \le 0$
- sqr: large if $ys \ll 1$
  but over-charge $ys \gg 1$
  small $\mathrm{err}_{\mathrm{SQR}} \rightarrow$ small $\mathrm{err}_{0/1}$
- ce: monotonic of ys
  small $\mathrm{err}_{\mathrm{CE}} \leftrightarrow$ small $\mathrm{err}_{0/1}$
- scaled ce: a proper upper bound of 0/1
  small $\mathrm{err}_{\mathrm{SCE}} \leftrightarrow$ small $\mathrm{err}_{0/1}$

upper bound:
useful for designing algorithmic error $\widehat{\mathrm{err}}$

Theoretical Implication of Upper Bound

For any ys where $s = \mathbf{w}^T\mathbf{x}$

$$\text{err}_{0/1}(s, y) \quad \le \quad \text{err}_{\text{SCE}}(s, y) = \frac{1}{\ln 2}\text{err}_{\text{CE}}(s, y).$$

$$\Longrightarrow \qquad \begin{aligned} E_{\text{in}}^{0/1}(\mathbf{w}) &\le E_{\text{in}}^{\text{SCE}}(\mathbf{w}) = \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w}) \\ E_{\text{out}}^{0/1}(\mathbf{w}) &\le E_{\text{out}}^{\text{SCE}}(\mathbf{w}) = \frac{1}{\ln 2} E_{\text{out}}^{\text{CE}}(\mathbf{w}) \end{aligned}$$

VC on $0/1$:

$$\begin{aligned} E_{\text{out}}^{0/1}(\mathbf{w}) &\le E_{\text{in}}^{0/1}(\mathbf{w}) + \Omega^{0/1} \\ &\le \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w}) + \Omega^{0/1} \end{aligned}$$

VC-Reg on CE :

$$\begin{aligned} E_{\text{out}}^{0/1}(\mathbf{w}) &\le \frac{1}{\ln 2} E_{\text{out}}^{\text{CE}}(\mathbf{w}) \\ &\le \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w}) + \frac{1}{\ln 2}\Omega^{\text{CE}} \end{aligned}$$

small $E_{\text{in}}^{\text{CE}}(\mathbf{w}) \Longrightarrow$ small $E_{\text{out}}^{0/1}(\mathbf{w})$:
logistic/linear reg. for linear classification

Regression for Classification

1. run **logistic**/**linear** **reg.** on $\mathcal{D}$ with $y_n \in \{-1, +1\}$ to get $\mathbf{w}_{\text{REG}}$
2. return $g(\mathbf{x}) = \text{sign}(\mathbf{w}_{\text{REG}}^T \mathbf{x})$

PLA

- pros: **efficient + strong guarantee if lin. separable**
- cons: works only if lin. separable, otherwise needing **pocket** heuristic

linear regression

- pros: **'easiest' optimization**
- cons: loose bound of $\text{err}_{0/1}$ for large $|ys|$

logistic regression

- pros: **'easy' optimization**
- cons: loose bound of $\text{err}_{0/1}$ for very negative ys

---

- **linear regression** sometimes used to set $\mathbf{w}_0$ for **PLA**/**pocket**/**logistic regression**
- **logistic regression** often preferred over **pocket**

Fun Time

Following the definition in the lecture, which of the following is not always $\geq \mathrm{err}_{0/1}(y, s)$ when $y \in \{-1, +1\}$?

1. $\mathrm{err}_{0/1}(y, s)$
2. $\mathrm{err}_{\mathrm{SQR}}(y, s)$
3. $\mathrm{err}_{\mathrm{CE}}(y, s)$
4. $\mathrm{err}_{\mathrm{SCE}}(y, s)$

Reference Answer: 3

Too simple, uh? :-) Anyway, note that $\mathrm{err}_{0/1}$ is surely an upper bound of itself.

Two Iterative Optimization Schemes

For $t = 0, 1, \ldots$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \mathbf{v}$$

when stop, return last $\mathbf{w}$ as g

PLA

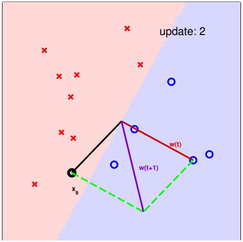
pick $(\mathbf{x}_n, y_n)$ and decide $\mathbf{w}_{t+1}$ by **the one example**

$O(1)$ time per iteration **:-)**

logistic regression (pocket)

check $\mathcal{D}$ and decide $\mathbf{w}_{t+1}$ (or new $\hat{\mathbf{w}}$) by **all examples**

$O(N)$ time per iteration **:-(**



logistic regression with
$O(1)$ **time per iteration?**

Logistic Regression Revisited

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \underbrace{\frac{1}{N}\sum_{n=1}^{N}\theta\left(-y_n\mathbf{w}_t^T\mathbf{x}_n\right)\left(y_n\mathbf{x}_n\right)}_{-\nabla E_{\text{in}}(\mathbf{w}_t)}$$

- want: update direction $\mathbf{v} \approx -\nabla E_{\text{in}}(\mathbf{w}_t)$
  while computing $\mathbf{v}$ by one single $(\mathbf{x}_n, y_n)$
- technique on removing $\frac{1}{N}\sum_{n=1}^{N}$:
  view as expectation $\mathcal{E}$ over uniform choice of n!

stochastic gradient:

$$\nabla_{\mathbf{w}}\ \mathrm{err}(\mathbf{w}, \mathbf{x}_n, y_n) \text{ with random } n$$

true gradient:

$$\nabla_{\mathbf{w}}\ E_{\text{in}}(\mathbf{w}) = \underset{\text{random } n}{\mathcal{E}}\ \nabla_{\mathbf{w}}\ \mathrm{err}(\mathbf{w}, \mathbf{x}_n, y_n)$$

Stochastic Gradient Descent (SGD)

stochastic gradient = true gradient + zero-mean 'noise' directions

Stochastic Gradient Descent

- idea: replace true gradient by stochastic gradient
- after enough steps,
  average true gradient $\approx$ average stochastic gradient
- pros: **simple & cheaper computation :-)**
  —useful for **big data** or **online learning**
- cons: less stable in nature

SGD logistic regression, **looks familiar? :-)**:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \underbrace{\theta\left(-y_n \mathbf{w}_t^T \mathbf{x}_n\right)\left(y_n \mathbf{x}_n\right)}_{-\nabla \mathrm{err}(\mathbf{w}_t, \mathbf{x}_n, y_n)}$$

PLA Revisited

SGD logistic regression:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \cdot \theta\left(-y_n \mathbf{w}_t^T \mathbf{x}_n\right)\left(y_n \mathbf{x}_n\right)$$

PLA:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + 1 \cdot \left[\!\left[y_n \neq \operatorname{sign}(\mathbf{w}_t^T \mathbf{x}_n) \right]\!\right]\left(y_n \mathbf{x}_n\right)$$

- SGD logistic regression $\approx$ 'soft' PLA
- PLA $\approx$ SGD logistic regression with $\eta = 1$ when $\mathbf{w}_t^T \mathbf{x}_n$ large

two practical rule-of-thumb:

- stopping condition? t **large enough**
- η? 0.1 when $\mathbf{x}$ in proper range

Fun Time

Consider applying SGD on linear regression for big data. What is the update direction when using the negative stochastic gradient?

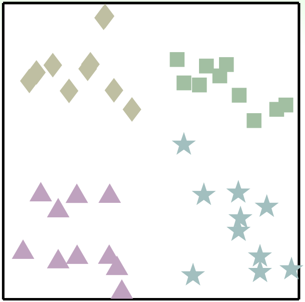
1. $\mathbf{x}_n$
2. $y_n \mathbf{x}_n$
3. $2(\mathbf{w}_t^T \mathbf{x}_n - y_n)\mathbf{x}_n$
4. $2(y_n - \mathbf{w}_t^T \mathbf{x}_n)\mathbf{x}_n$

Reference Answer: 4

Go check lecture 9 if you have forgotten about the gradient of squared error. :-)

Anyway, the update rule has a nice physical interpretation: improve $\mathbf{w}_t$ by 'correcting' proportional to the residual $(y_n - \mathbf{w}_t^T \mathbf{x}_n)$.

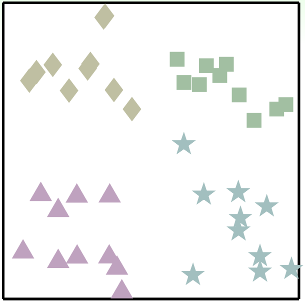
Multiclass Classification

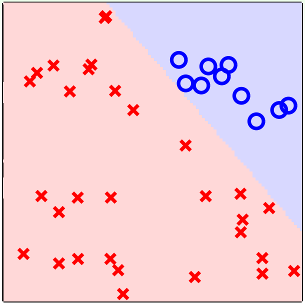


- $\mathcal{Y} = \{\square, \Diamond, \triangle, \star\}$ (4-class classification)
- **many applications** in practice, especially for 'recognition'

next: use **tools for $\{\times, \circ\}$ classification** to $\{\square, \Diamond, \triangle, \star\}$ classification

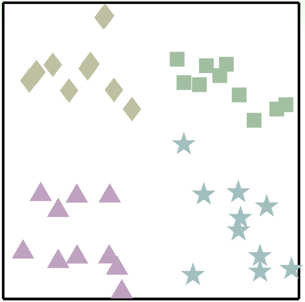
One Class at a Time

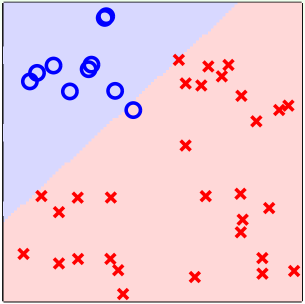




$\square$ or not? $\{\square = \circ, \lozenge = \times, \triangle = \times, \star = \times\}$

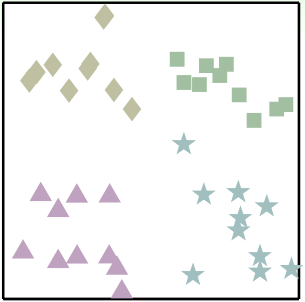
One Class at a Time

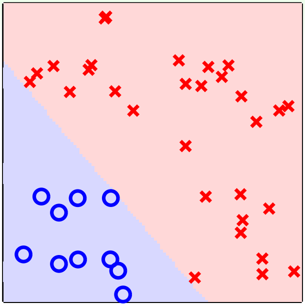




$\diamond$ or not? $\{\square = \times, \diamond = \circ, \triangle = \times, \star = \times\}$

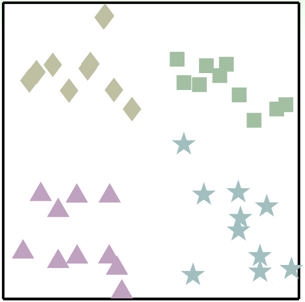
One Class at a Time

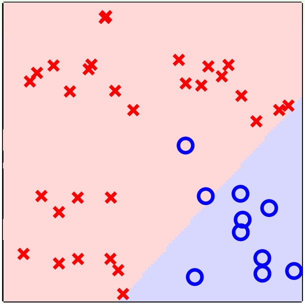




$\triangle$ or not? $\{\square = \times, \lozenge = \times, \triangle = \circ, \star = \times\}$

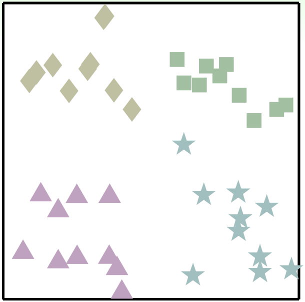
One Class at a Time

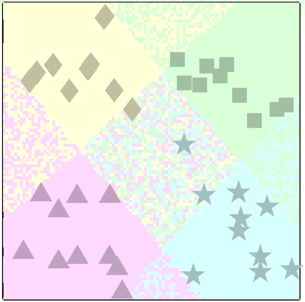


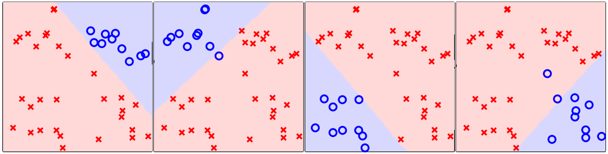


$\star$ or not? $\{\square = \times, \lozenge = \times, \triangle = \times, \star = \circ\}$

Multiclass Prediction: Combine Binary Classifiers

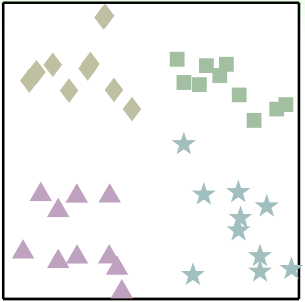


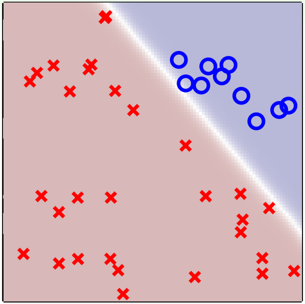


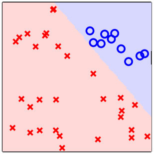


but **ties? :-)**

One Class at a Time **Softly**

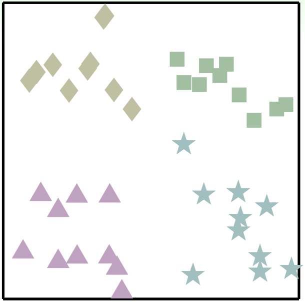


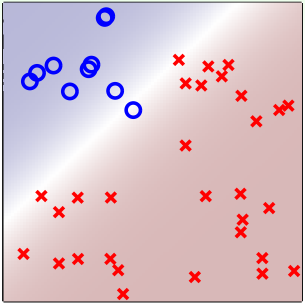


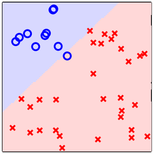


$$P(\square|\mathbf{x})? \ \{\square = \circ, \Diamond = \times, \triangle = \times, \star = \times\}$$

One Class at a Time **Softly**

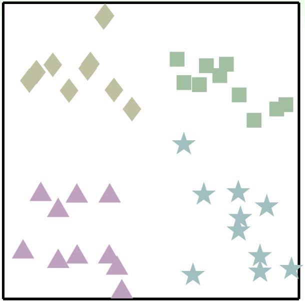


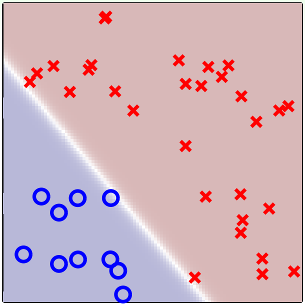


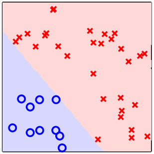


$P(\Diamond|\mathbf{x})$? $\{\square = \times, \Diamond = \circ, \triangle = \times, \star = \times\}$

One Class at a Time **Softly**

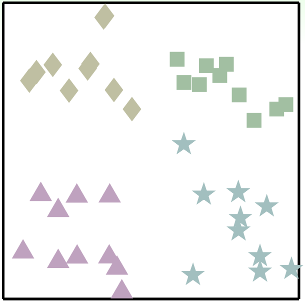


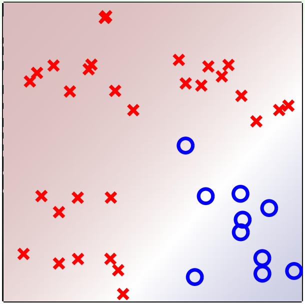


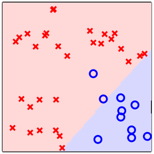


$$P(\triangle|\mathbf{x})?\ \{\square = \times, \Diamond = \times, \triangle = \circ, \star = \times\}$$

One Class at a Time **Softly**

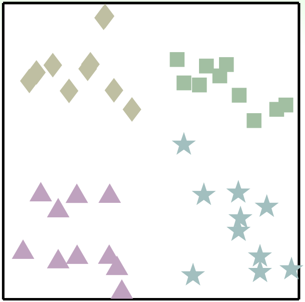


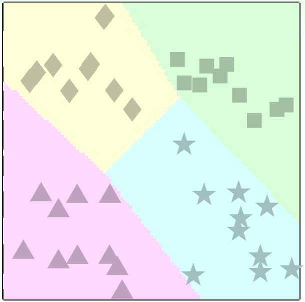


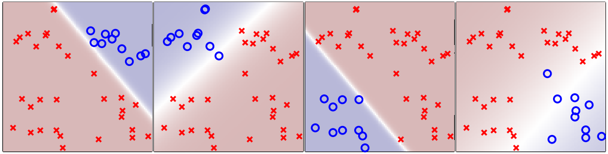


$P(\star|\mathbf{x})$? $\{\square = \times, \lozenge = \times, \triangle = \times, \star = \circ\}$

Multiclass Prediction: Combine **Soft** Classifiers







$$g(\mathbf{x}) = \operatorname{argmax}_{k \in \mathcal{Y}} \, \theta\left(\mathbf{w}_{[k]}^T \mathbf{x}\right)$$

One-Versus-All (OVA) Decomposition

1. for $k \in \mathcal{Y}$

   obtain $\mathbf{w}_{[k]}$ by running logistic regression on

$$\mathcal{D}_{[k]} = \{(\mathbf{x}_n, y'_n = 2\,[\![y_n = k]\!] - 1)\}_{n=1}^{N}$$

2. return $g(\mathbf{x}) = \text{argmax}_{k \in \mathcal{Y}} \left(\mathbf{w}_{[k]}^T \mathbf{x}\right)$

- pros: efficient,
  can be coupled with any logistic regression-like approaches
- cons: often **unbalanced** $\mathcal{D}_{[k]}$ when K large
- extension: **multinomial ('coupled') logistic regression**

OVA: a simple multiclass **meta-algorithm**
to keep in your toolbox

Fun Time

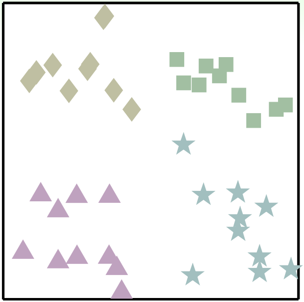
Which of the following best describes the training effort of OVA decomposition based on logistic regression on some K-class classification data of size N?

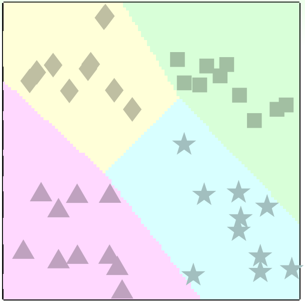
1. learn K logistic regression hypotheses, each from data of size N/K
2. learn K logistic regression hypotheses, each from data of size $N \ln K$
3. learn K logistic regression hypotheses, each from data of size N
4. learn K logistic regression hypotheses, each from data of size NK

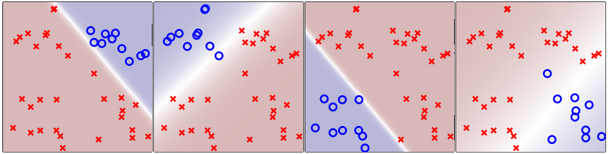
Reference Answer: 3

Note that the **learning part can be easily done in parallel**, while the data is essentially of the same size as the original data.

Source of **Unbalance**: One versus **All**

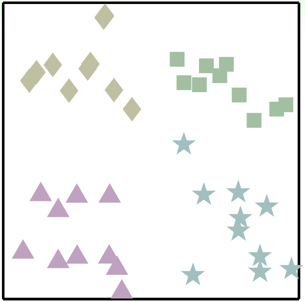


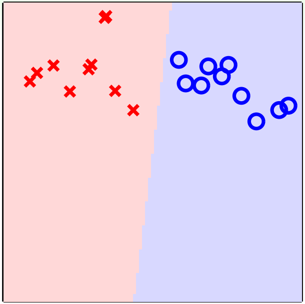




idea: make binary classification problems
more **balanced** by one versus **one**

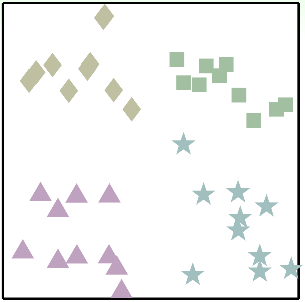
One versus One at a Time

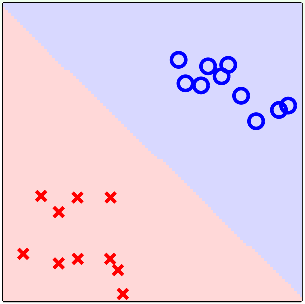




$\square$ or $\lozenge$? $\{\square = \circ, \lozenge = \times, \triangle = \text{nil}, \star = \text{nil}\}$

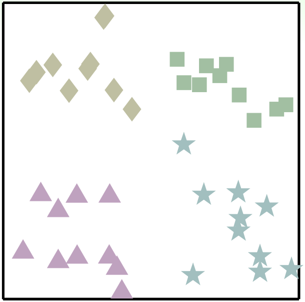
One versus One at a Time

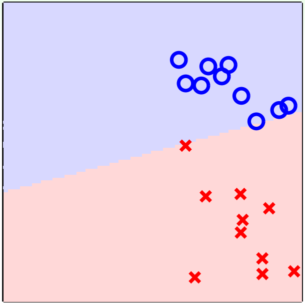




$\square$ or $\triangle$? $\{\square = \circ, \Diamond = \text{nil}, \triangle = \times, \star = \text{nil}\}$

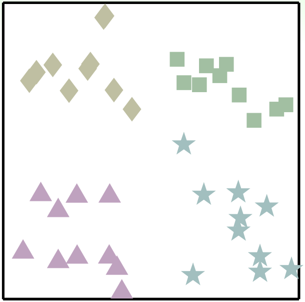
One versus One at a Time

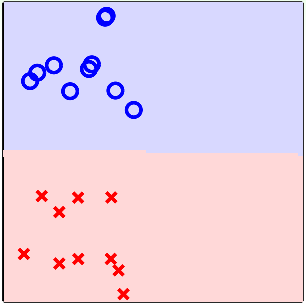




$\square$ or $\star$? $\{\square = \circ, \Diamond = \text{nil}, \triangle = \text{nil}, \star = \times\}$

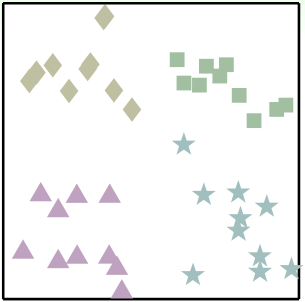
One versus One at a Time

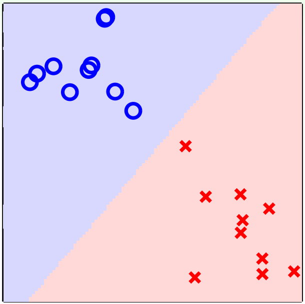




$\Diamond$ or $\triangle$? $\{\square = \text{nil}, \Diamond = \circ, \triangle = \times, \star = \text{nil}\}$

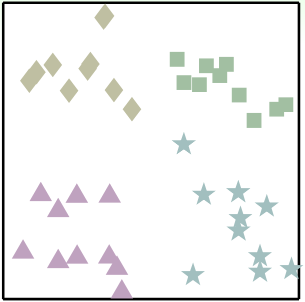
One versus One at a Time

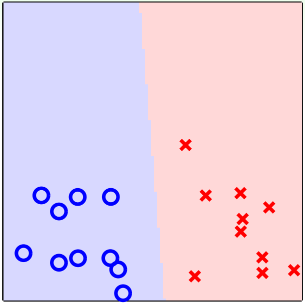




$\Diamond$ or $\star$? $\{\Box = \text{nil}, \Diamond = \circ, \triangle = \text{nil}, \star = \times\}$

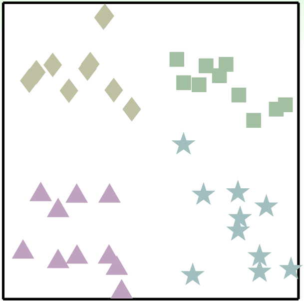
One versus One at a Time

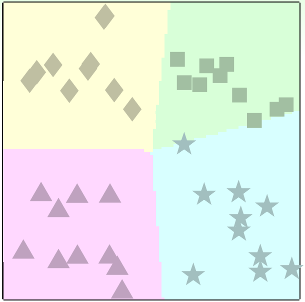


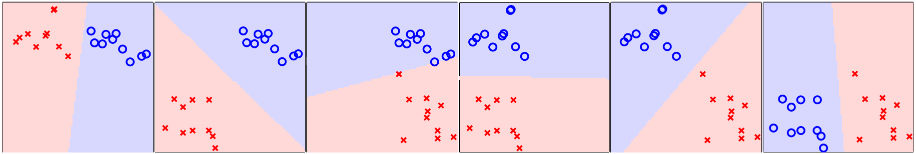


$\triangle$ or $\star$? $\{\square = \text{nil}, \lozenge = \text{nil}, \triangle = \circ, \star = \times\}$

Multiclass Prediction: Combine **Pairwise** Classifiers







$$g(\mathbf{x}) = \text{tournament champion} \left\{ \mathbf{w}_{[k,\ell]}^T \mathbf{x} \right\}$$

(voting of classifiers)

One-versus-one (OVO) Decomposition

1. for $(k, \ell) \in \mathcal{Y} \times \mathcal{Y}$
   obtain $\mathbf{w}_{[k,\ell]}$ by running linear binary classification on

$$\mathcal{D}_{[k,\ell]} = \{(\mathbf{x}_n, y_n' = 2\,[\![y_n = k]\!] - 1) : y_n = k \text{ or } y_n = \ell\}$$

2. return $g(\mathbf{x}) =$ tournament champion $\left\{\mathbf{w}_{[k,\ell]}^T \mathbf{x}\right\}$

- pros: efficient ('smaller' training problems), stable,
  can be coupled with any binary classification approaches
- cons: use $O(K^2)$ $\mathbf{w}_{[k,\ell]}$
  —**more space, slower prediction, more training**

OVO: another simple multiclass
meta-algorithm to keep in your toolbox

Fun Time

Assume that some binary classification algorithm takes exactly N^3 CPU-seconds for data of size N. Also, for some 10-class multiclass classification problem, assume that there are $N/10$ examples for each class. Which of the following is total CPU-seconds needed for OVO decomposition based on the binary classification algorithm?

1. $\frac{9}{200} N^3$
2. $\frac{9}{25} N^3$
3. $\frac{4}{5} N^3$
4. N^3

Reference Answer: 2

There are 45 binary classifiers, each trained with data of size $(2N)/10$. Note that OVA decomposition with the same algorithm would take $10N^3$ time, much worse than OVO.

Summary

1. When Can Machines Learn?
2. Why Can Machines Learn?
3. **How** Can Machines Learn?

Lecture 10: Logistic Regression

Lecture 11: Linear Models for Classification

- Linear Models for Binary Classification
  three models useful in different ways
- Stochastic Gradient Descent
  follow negative stochastic gradient
- Multiclass via Logistic Regression
  predict with maximum estimated $P(k|\mathbf{x})$
- Multiclass via Binary Classification
  predict the tournament champion

- **next: from linear to nonlinear**

4. How Can Machines Learn Better?