D3APL – Aplicações em Ciência de Dados

Implementing a Linear Classifier from Scratch (Part 2)

Prof. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas

L2 Regularization (Ridge)

L2 Regularization (Ridge)

L2 regularization adds the following ***penalization*** to our cost function:

number of features

$$R(\mathbf{w}) = \alpha \sum_{j=1}^{n} w_j^2$$

weight vector

regularization parameter

⚠ The **bias term** (w_0) **is not regularized.**

L2 Regularization (Ridge)

Some authors adds some **scale factors** to L2 regularization only to make future mathematical operations (*e.g.*, derivative) easier.

Example 1:

$$R(\mathbf{w}) = \alpha \frac{1}{2} \sum_{j=1}^{n} w_j^2$$

Example 2:

$$R(\mathbf{w}) = \alpha \frac{1}{2m} \sum_{j=1}^{n} w_j^2$$

number of training samples

L2 Regularization (Ridge)

Some authors adds some **scale factors** to L2 regularization only to make future mathematical operations (*e.g.*, derivative) easier.

Example 1:

$$R(\mathbf{w}) = \alpha \frac{1}{2} \sum_{j=1}^{n} w_j^2$$

Example 2:

$$R(\mathbf{w}) = \alpha \frac{1}{2m} \sum_{j=1}^{n} w_j^2$$

number of training samples

Regularized Objective Cost Function

$J(\mathbf{w}, b) = J(\mathbf{w}, b) + R(\mathbf{w})$

Regularized Objective Cost Function

$$J(\mathbf{w}, b) = J(\mathbf{w}, b) + R(\mathbf{w})$$

$$J(\mathbf{w}, b) = \left(\frac{1}{m}\sum_{i=1}^{m} L(\hat{p}^{(i)}, y^{(i)})\right) + \alpha\,\frac{1}{2}\sum_{j=1}^{n} w_j^2$$

Regularized Objective Cost Function

$$J(\mathbf{w}, b) = J(\mathbf{w}, b) + R(\mathbf{w})$$

$$J(\mathbf{w}, b) = \left(\frac{1}{m}\sum_{i=1}^{m} L(\hat{p}^{(i)}, y^{(i)})\right) + \alpha\frac{1}{2}\sum_{j=1}^{n} w_j^2$$

$$= \left(\frac{1}{m}\sum_{i=1}^{m} -\left[y^{(i)} * \ln\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) * \ln(1 - \hat{p}^{(i)})\right]\right) + \alpha\frac{1}{2}\sum_{j=1}^{n} w_j^2$$

$$= \left(-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} * \ln\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) * \ln(1 - \hat{p}^{(i)})\right]\right) + \alpha\frac{1}{2}\sum_{j=1}^{n} w_j^2$$

$$= \left(-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} * \ln\left(\sigma\left(\mathbf{w}\cdot\mathbf{x}^{(\boldsymbol{i})^{T}} + b\right)\right) + \left(1 - y^{(i)}\right) * \ln(1 - \sigma\left(\mathbf{w}\cdot\mathbf{x}^{(\boldsymbol{i})^{T}} + b\right))\right]\right) + \alpha\frac{1}{2}\sum_{j=1}^{n} w_j^2$$

Deriving the regularized cost function

Deriving the regularized cost function

$$\frac{\partial}{\partial b}J(\mathbf{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left[\left(\hat{p}^{(i)} - y^{(i)}\right) * 1\right] = \frac{1}{m}\sum_{i=1}^{m}\left[\left(\sigma\left(\mathbf{w}\cdot \mathrm{x}^{(i)^{\mathrm{T}}} + b\right) - y^{(i)}\right) * 1\right]$$

$$\frac{\partial}{\partial w_j}J(\mathbf{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left[\left(\hat{p}^{(i)} - y^{(i)}\right) * x_j^{(i)}\right] + \alpha w_j = \frac{1}{m}\sum_{i=1}^{m}\left[\left(\sigma\left(\mathbf{w}\cdot \mathrm{x}^{(i)^{\mathrm{T}}} + b\right) - y^{(i)}\right) * x_j^{(i)}\right] + \alpha w_j$$

Deriving the regularized cost function

$$\frac{\partial}{\partial b} J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \left[\left(\hat{p}^{(i)} - y^{(i)} \right) * 1 \right] = \frac{1}{m} \sum_{i=1}^{m} \left[\left(\sigma \left(\mathbf{w} \cdot \mathrm{x}^{(i)^{\mathrm{T}}} + b \right) - y^{(i)} \right) * 1 \right]$$

$$\frac{\partial}{\partial w_j} J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \left[\left(\hat{p}^{(i)} - y^{(i)} \right) * x_j^{(i)} \right] + \alpha w_j = \frac{1}{m} \sum_{i=1}^{m} \left[\left(\sigma \left(\mathbf{w} \cdot \mathrm{x}^{(i)^{\mathrm{T}}} + b \right) - y^{(i)} \right) * x_j^{(i)} \right] + \alpha w_j$$

$$w_j = w_j - \eta \frac{\partial}{\partial w_j} J(w_1, w_2, \cdots, w_n, b)$$

$$w_j = w_j - \eta \left(\frac{1}{m} \sum_{i=1}^{m} \left[\left(\sigma \left(\mathbf{w} \cdot \mathrm{x}^{(i)^{\mathrm{T}}} + b \right) - y^{(i)} \right) * x_j^{(i)} \right] + \alpha w_j \right)$$

Gradient Descent Variants

Batch Gradient Descent

"Batch": each iteration of the gradient descent uses **all samples** of the **training set $\mathbf{X}$**:

Given some cost function: $J(\mathbf{w}, b) = J(w_1, w_2, \cdots, w_n, b)$

Algorithm parameter: learning rate $\eta \in (0, 1]$, training set $\mathbf{X}$, target labels $\mathbf{y}$

Goal: $\min_{w_1, w_2, \cdots, w_n, b} J(w_1, w_2, \cdots, w_n, b)$

Algorithm:

1. Initialize $w_1, w_2, \cdots, w_n, b$ arbitrarily (*e.g., following a standard normal distribution*)
2. **repeat until convergence** (*e.g., for a given number of epochs/iterations*):
3. $\hat{y} = h_{\mathbf{w},b}(\mathbf{X})$ // *predict all samples in* $\mathbf{X}$
4. Compute $J(w_1, w_2, \cdots, w_n, b)$ // *based on* $\mathbf{y}$ and $\hat{y}$
5. Change $w_1, w_2, \cdots, w_n, b$ **to reduce** $J(w_1, w_2, \cdots, w_n, b)$

Mini-batch Gradient Descent

Given some cost function: $J(\mathbf{w}, b) = J(w_1, w_2, \cdots, w_n, b)$

Algorithm parameter: learning rate $\eta \in (0, 1]$, **mini-batch size k**, training set $\mathbf{X}$, target labels $\mathbf{y}$

Goal: $\min\limits_{w_1, w_2, \cdots, w_n, b} J(w_1, w_2, \cdots, w_n, b)$

Algorithm:

1. Initialize $w_1, w_2, \cdots, w_n, b$ arbitrarily (*e.g., following a standard normal distribution*)
2. **repeat until convergence** (*e.g., for a given number of epochs/iterations*):
3. *Shuffle* $\langle \mathbf{X}, \mathbf{y} \rangle$
4. for each batch of size **k** in $\mathbf{X}, \mathbf{y}$:
5. $\mathbf{X_b}, \mathbf{y_b}$ = get the next batch of size **k** from $\langle \mathbf{X}, \mathbf{y} \rangle$
6. $\hat{y}_b = h_{\mathbf{w},b}(\mathbf{X_b})$ // *predict all samples in $\boldsymbol{X_b}$*
7. Compute $J(w_1, w_2, \cdots, w_n, b)$ // *based on* $\mathbf{y_b}$ and $\hat{y}_b$
8. Change $w_1, w_2, \cdots, w_n, b$ **to reduce** $J(w_1, w_2, \cdots, w_n, b)$

Alternatively, one could **draw k samples randomly** with replacement.

Stochastic Gradient Descent

Given some cost function: $J(\mathbf{w}, b) = J(w_1, w_2, \cdots, w_n, b)$

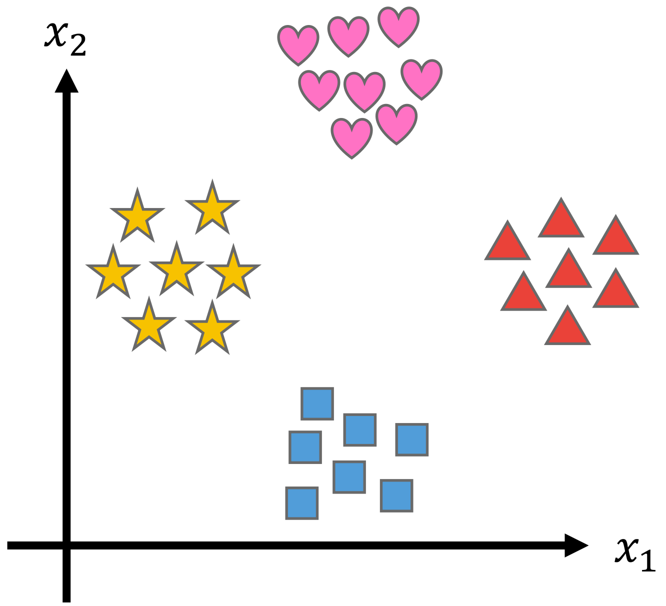
Algorithm parameter: learning rate $\eta \in (0, 1]$, training set $\mathbf{X}$, target labels $\mathbf{y}$

Goal: $\min_{w_1, w_2, \cdots, w_n, b} J(w_1, w_2, \cdots, w_n, b)$

Algorithm:

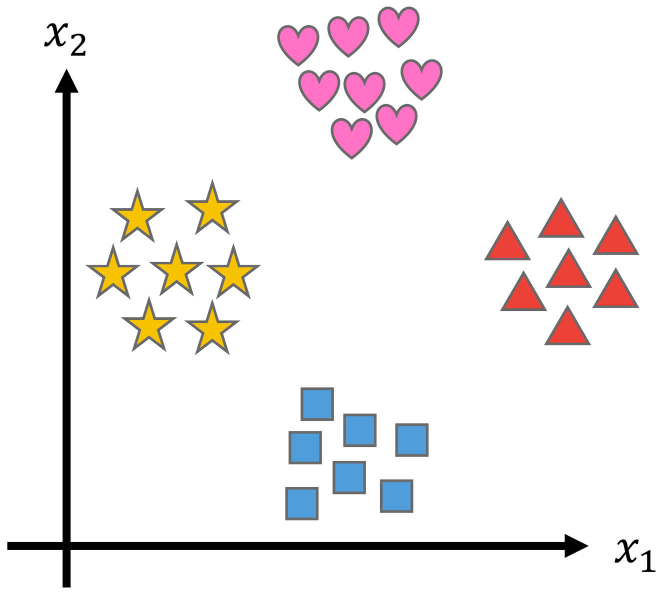
1. Initialize $w_1, w_2, \cdots, w_n, b$ arbitrarily (*e.g., following a standard normal distribution*)
2. **repeat until convergence** (*e.g., for a given number of epochs/iterations*):
3. *Shuffle* $\langle \mathbf{X}, \mathbf{y} \rangle$
4. for each sample $\mathrm{x}^{(\mathrm{i})}$ in shuffled $\mathbf{X}$:
5. $\hat{y}^{(i)} = h_{\mathbf{w},b}\left(\mathrm{x}^{(\mathrm{i})}\right)$ *// predict* $\mathrm{x}^{(\mathrm{i})}$
6. Compute $J(w_1, w_2, \cdots, w_n, b)$
7. Change $w_1, w_2, \cdots, w_n, b$ **to reduce** $J(w_1, w_2, \cdots, w_n, b)$

Extending a Binary Classifier to Multiclass Classification



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

One vs Rest (One vs All)



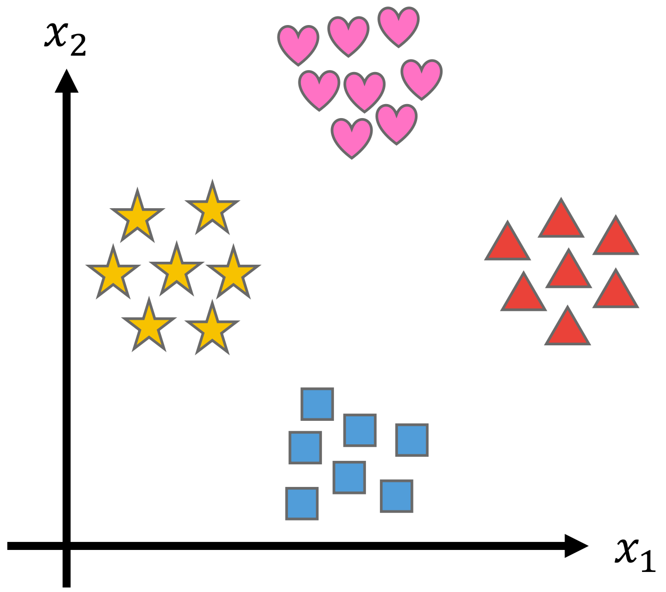
Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

Training

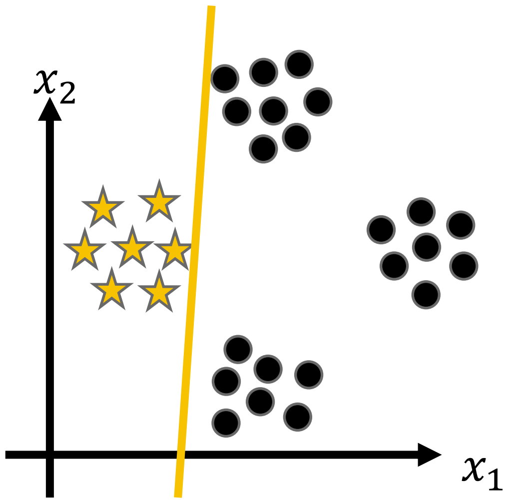
For each **class c**, train a **binary classifier considering c as the positive class** and all other instances (from other classes) as **negatives**.

Total of classifiers = number of classes.

One vs Rest (One vs All)



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

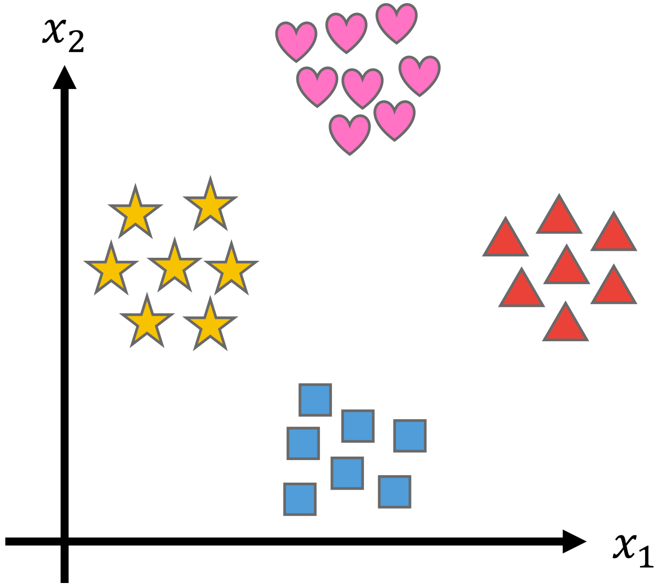


Training

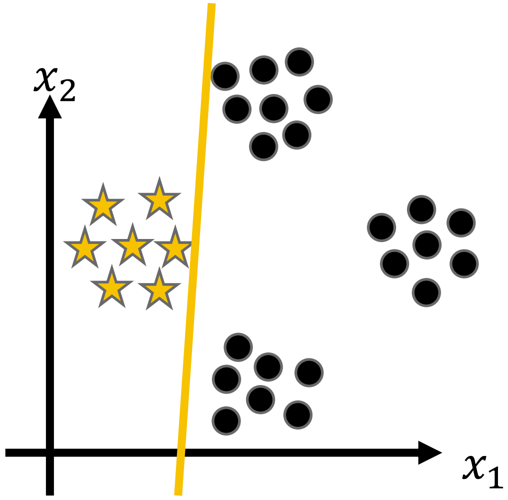
For each **class c**, train a **binary classifier considering c as the positive class** and all other instances (from other classes) as **negatives**.

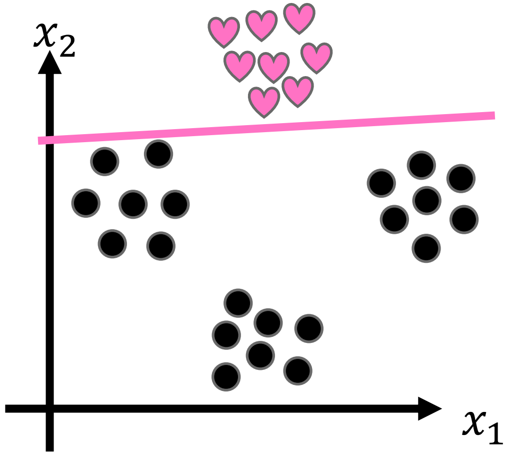
Total of classifiers = number of classes.

One vs Rest (One vs All)



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue



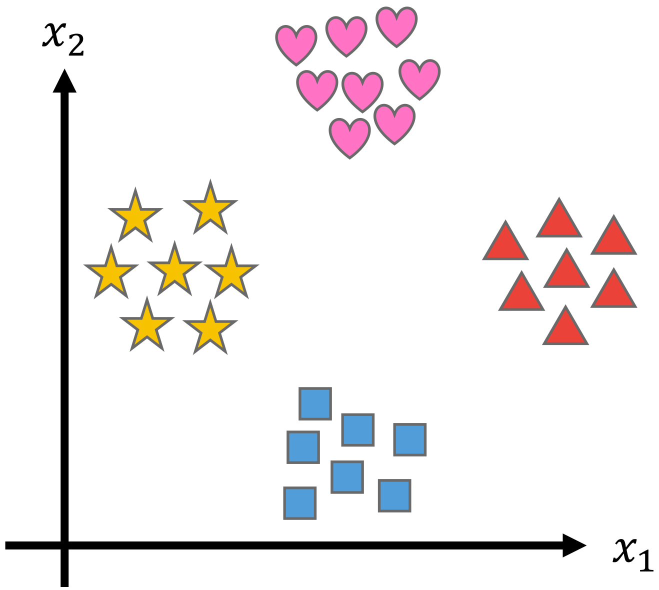


Training

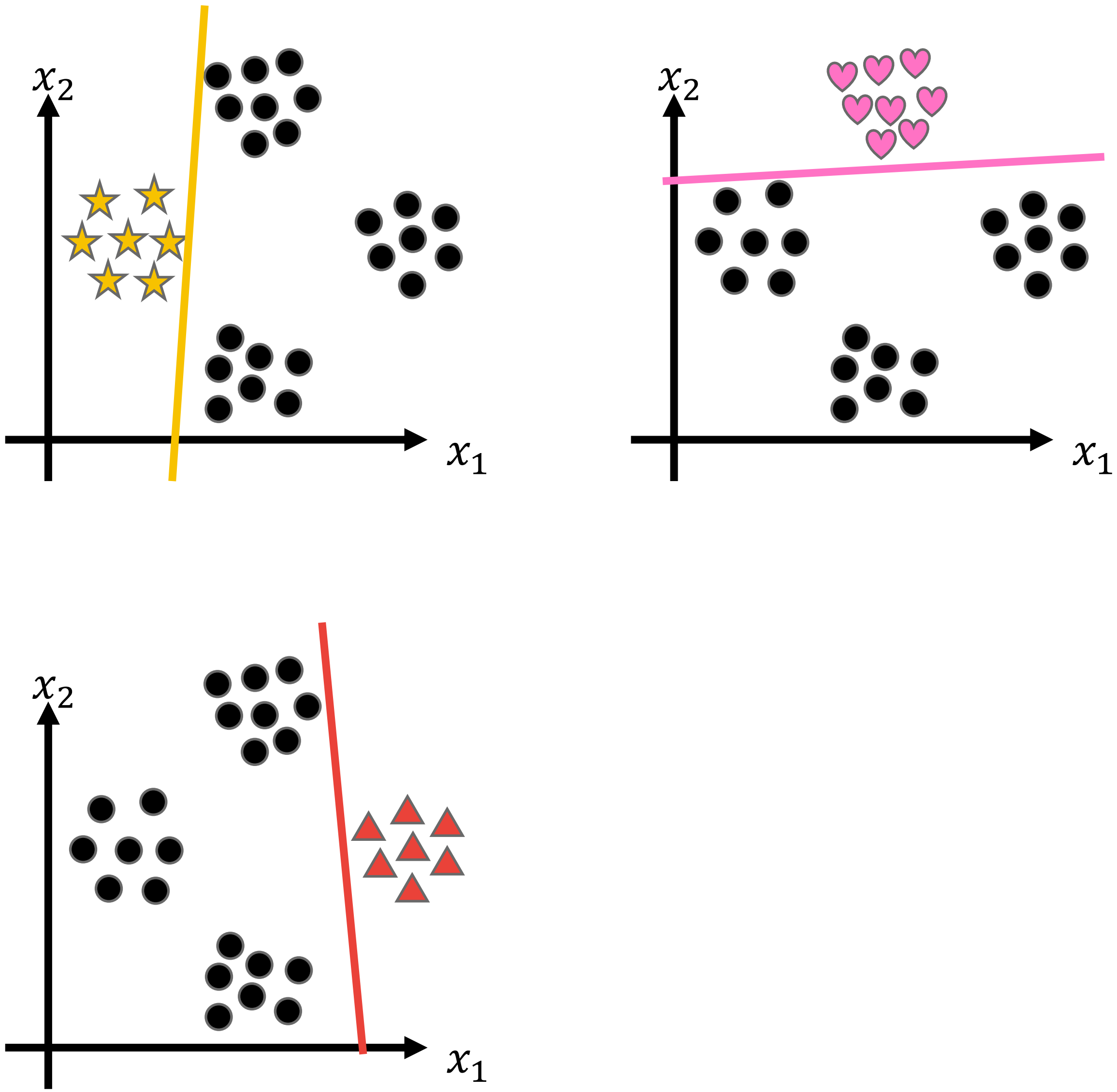
For each class c, train a **binary classifier** **considering** c as the positive class and all other instances (from other classes) as **negatives**.

Total of classifiers = number of classes.

One vs Rest (One vs All)



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

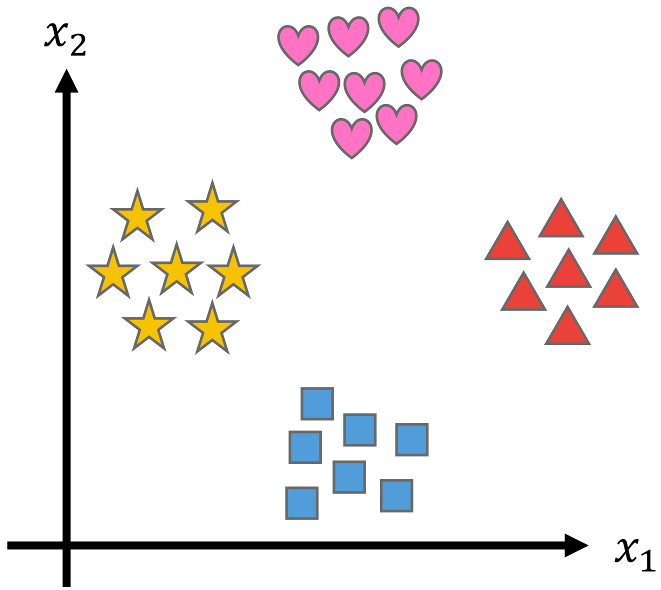


Training

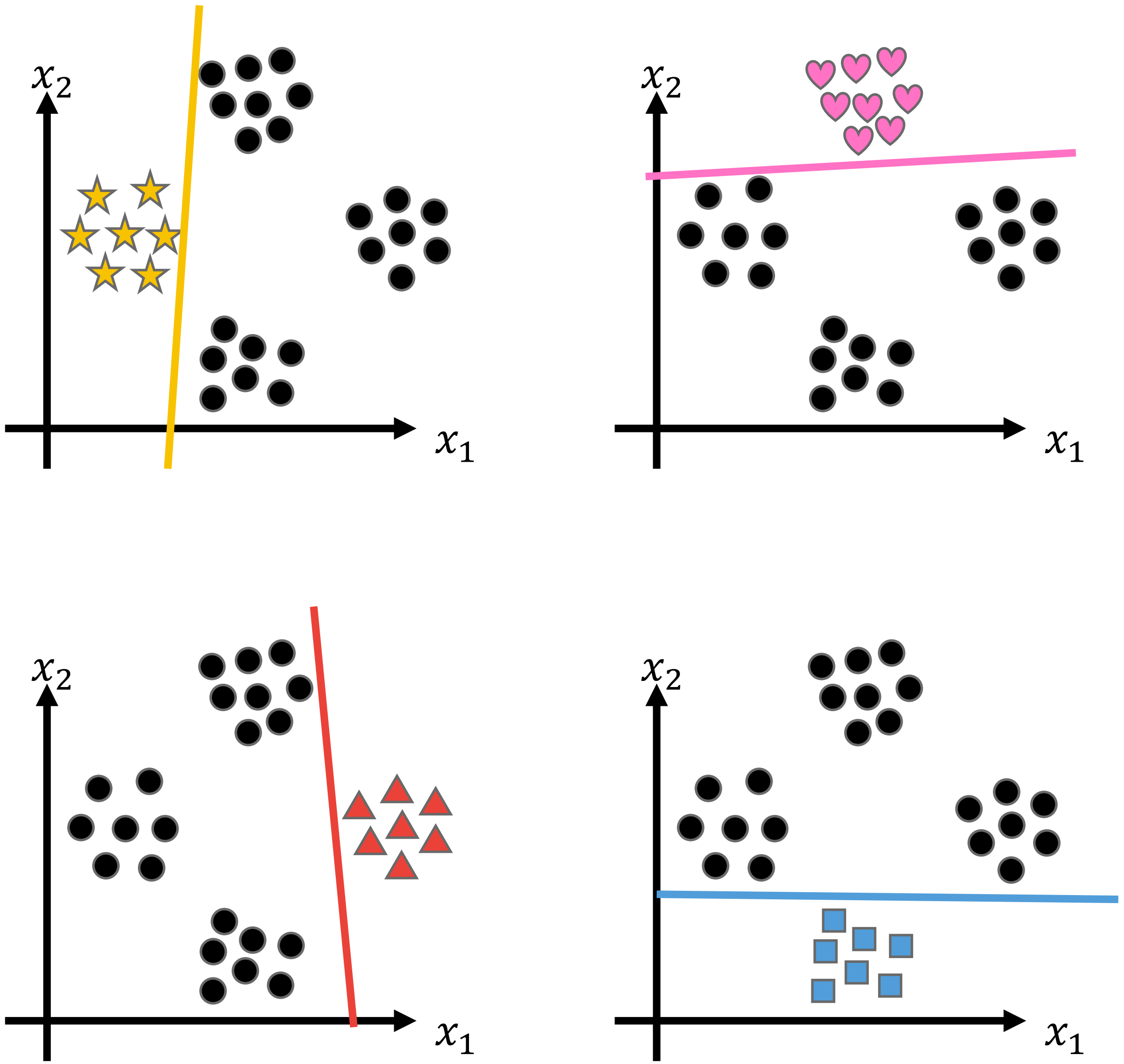
For each class c, train a **binary classifier** **considering** c as the positive class and all other instances (from other classes) as **negatives**.

Total of classifiers = number of classes.

One vs Rest (One vs All)



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue



Training

For each class c, train a **binary classifier** **considering** c as the positive class and all other instances (from other classes) as **negatives**.

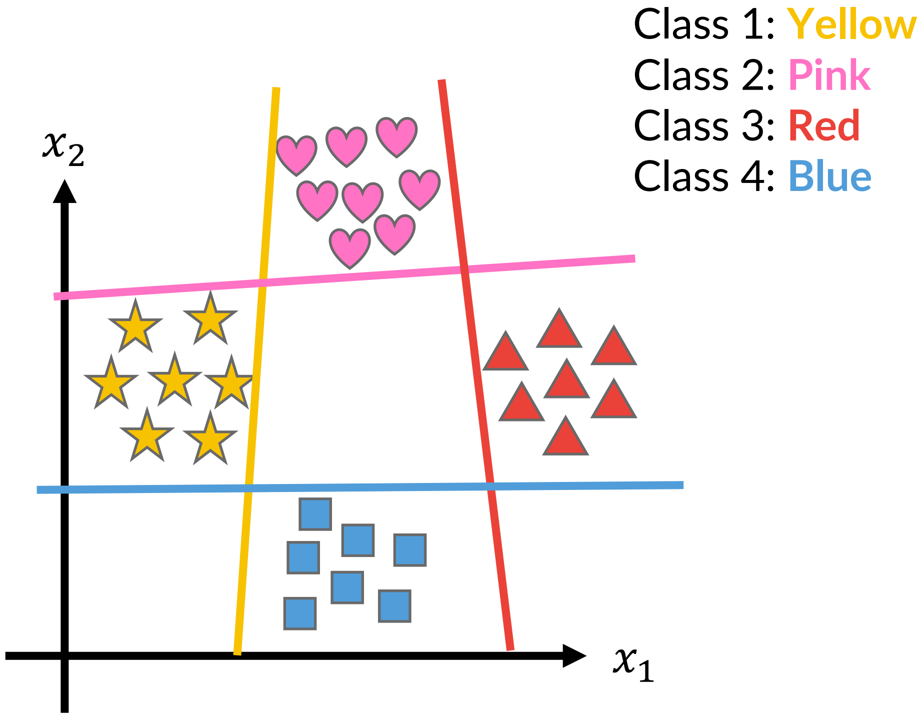
Total of classifiers = number of classes.

One vs Rest (One vs All)

Training

For each **class c**, train a **binary classifier considering c as the positive class** and **all other instances** (from other classes) as **negatives**.

Total of classifiers = number of classes.



One vs Rest (One vs All)

Training

For each class c, train a **binary classifier** **considering** c as the positive class and all other instances (from other classes) as **negatives**.

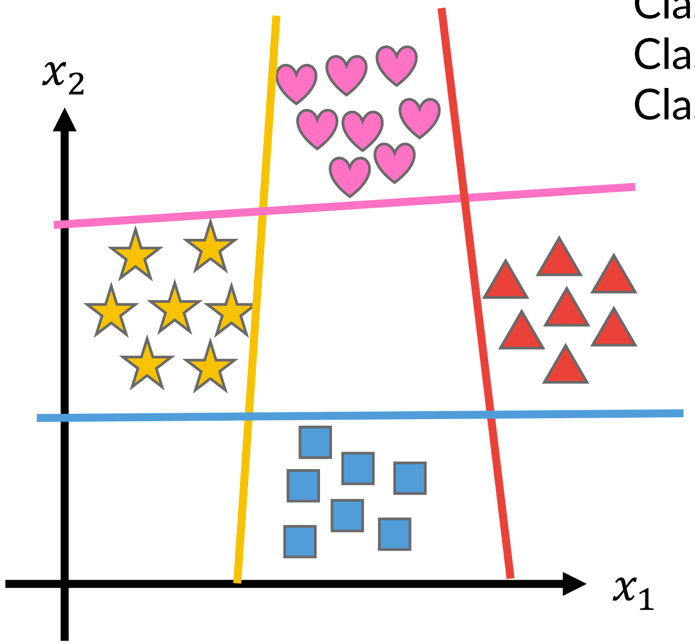
Total of classifiers = number of classes.

Classification/prediction

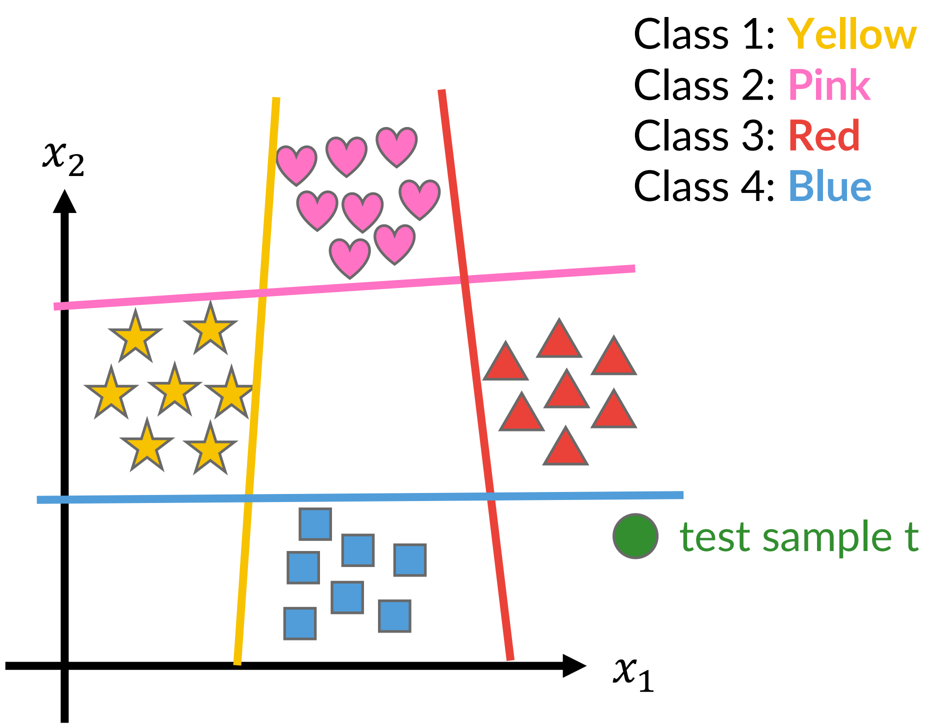
Given a test sample t:

- Classify t w.r.t. **all binary classifiers**;
- **Assign the label** from the binary classifier that **provided the highest score** (e.g., probability of belonging to its class): **argmax**

Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue



One vs Rest (One vs All)



Training

For each **class c**, train a **binary classifier considering c as the positive class** and **all other instances (from other classes) as negatives**.

Total of classifiers = number of classes.

Classification/prediction

Given a test sample t:

- Classify t w.r.t. **all binary classifiers**;
- **Assign the label** from the binary classifier that **provided the highest score** (e.g., probability of belonging to its class): **argmax**

One vs Rest (One vs All)

Training

For each **class c**, train a **binary classifier considering c as the positive class** and **all other instances** (from other classes) as **negatives**.

Total of classifiers = number of classes.

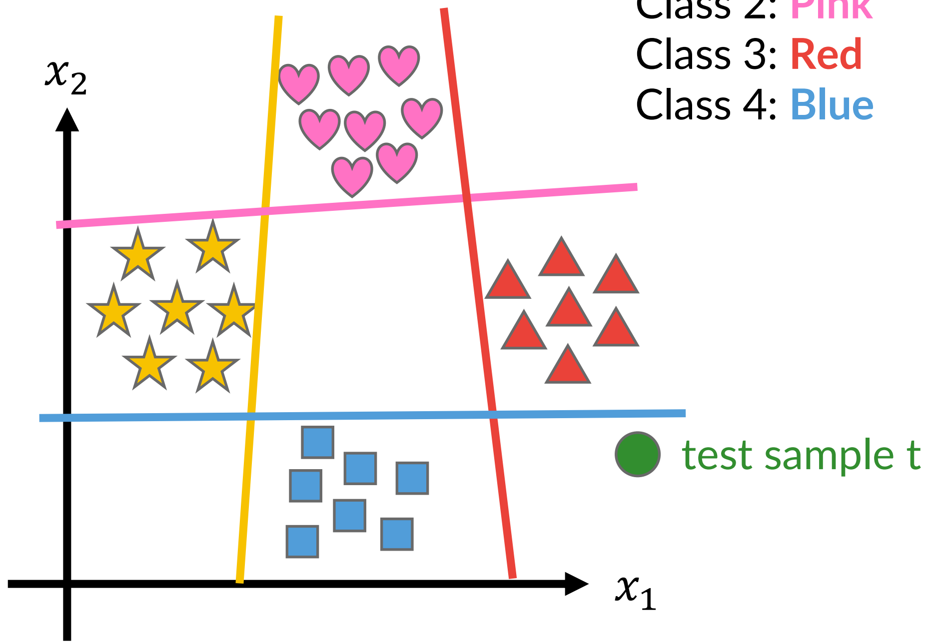
Classification/prediction

Given a test sample t:

- Classify t w.r.t. **all binary classifiers**;

- **Assign the label** from the binary classifier that **provided the highest score** (e.g., probability of belonging to its class): **argmax**

Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue



Estimated probabilities

$$\hat{p}_1^{(t)} = 0.05$$
$$\hat{p}_2^{(t)} = 0.03$$
$$\hat{p}_3^{(t)} = 0.80$$
$$\hat{p}_4^{(t)} = 0.60$$

One vs Rest (One vs All)

Training

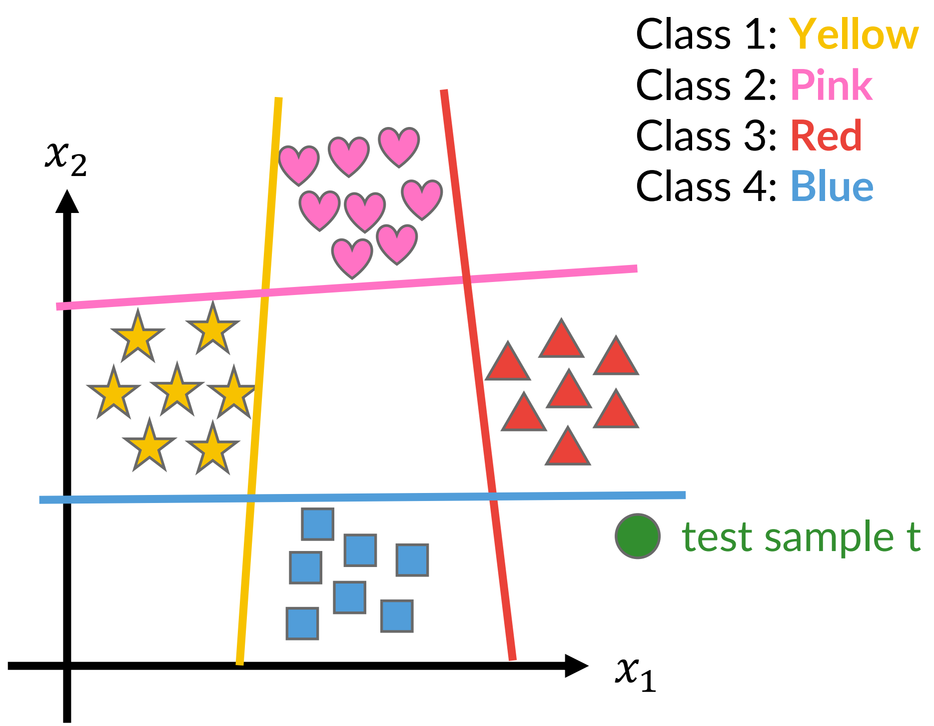
For each class c, train a **binary classifier** **considering** c as the positive class and all other instances (from other classes) as **negatives**.

Total of classifiers = number of classes.

Classification/prediction

Given a test sample t:

- Classify t w.r.t. **all binary classifiers**;
- **Assign the label** from the binary classifier that **provided the highest score** (e.g., probability of belonging to its class): **argmax**



Estimated probabilities

$$\hat{p}_1^{(t)} = 0.05$$
$$\hat{p}_2^{(t)} = 0.03$$
$$\boxed{\hat{p}_3^{(t)} = 0.80}$$
$$\hat{p}_4^{(t)} = 0.60$$

Predicted label

$$\hat{y}^{(t)} = 3$$

One vs Rest (One vs All)

Training

For each **class c**, train a **binary classifier** **considering c as the positive class** and **all other instances** (from other classes) as **negatives**.

Total of classifiers = number of classes.

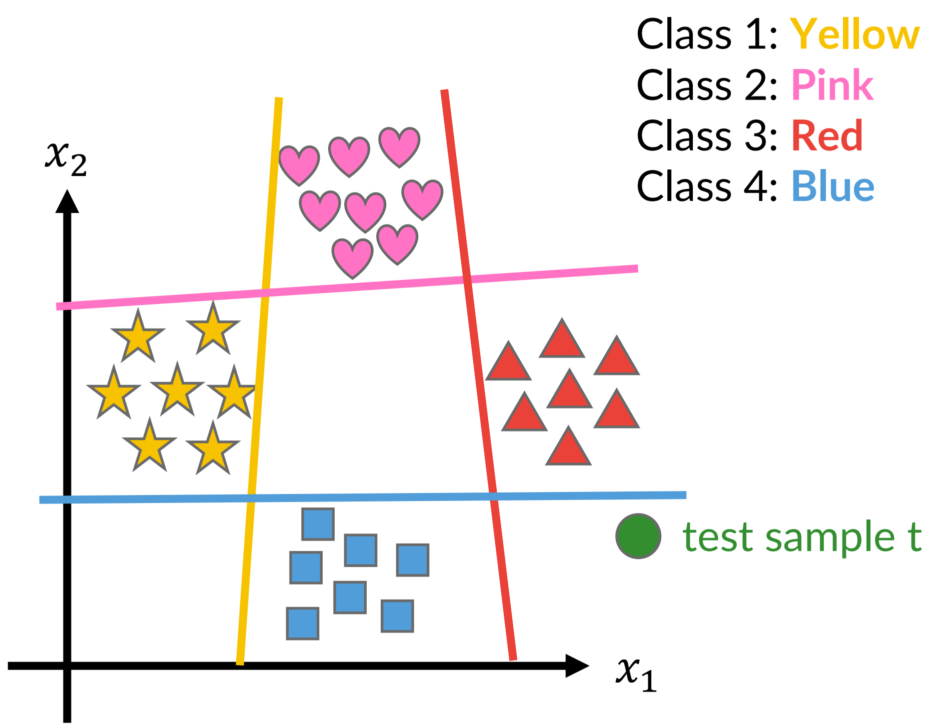
Classification/prediction

Given a test sample t:

- Classify t w.r.t. **all binary classifiers**;
- **Assign the label** from the binary classifier that **provided the highest score** (e.g., probability of belonging to its class): **argmax**

`sklearn.multiclass.OneVsRestClassifier`

https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html



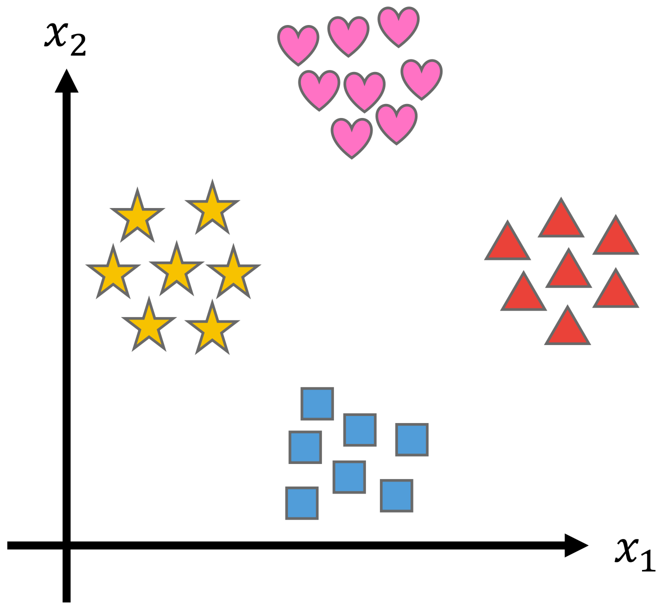
Estimated probabilities

$$\hat{p}_1^{(t)} = 0.05$$
$$\hat{p}_2^{(t)} = 0.03$$
$$\boxed{\hat{p}_3^{(t)} = 0.80}$$
$$\hat{p}_4^{(t)} = 0.60$$

Predicted label

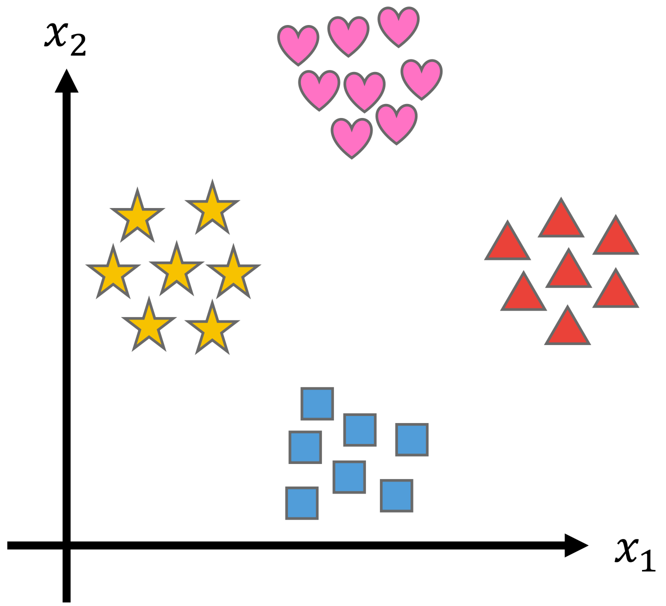
$$\hat{y}^{(t)} = 3$$

One vs One



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

One vs One



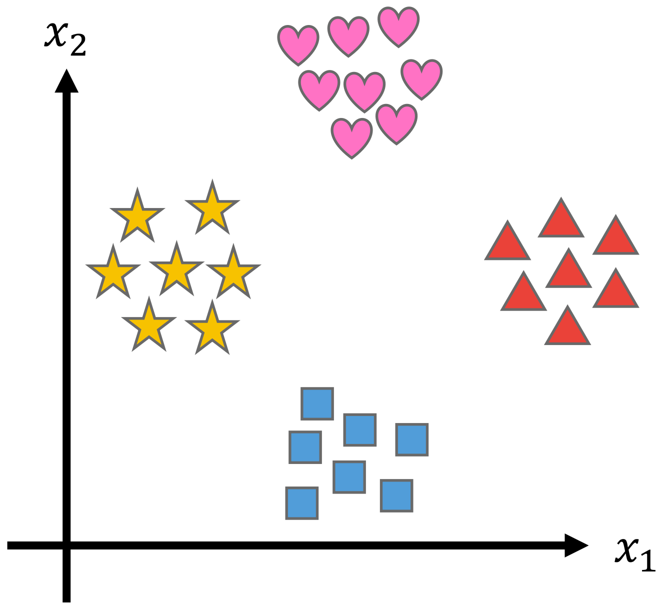
Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

Training

Train a **binary classifier** for *each pair of classes* from your dataset (total of k classes).

Total of classifiers = $C(k,2) = \frac{k!}{(k-2)!2!} = \frac{k(k-1)}{2}$

One vs One



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

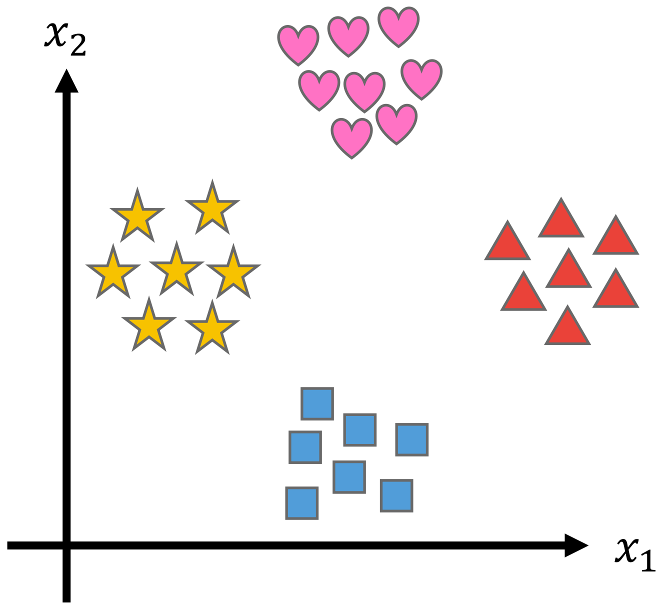
Training

Train a **binary classifier** for *each pair of classes* from your dataset (total of k classes).

Total of classifiers = $C(k, 2) = \frac{k!}{(k-2)!2!} = \frac{k(k-1)}{2}$

Total of classifiers = $C(4, 2) = \frac{4(4-1)}{2} = 6$

One vs One



Class 1: Yellow
Class 2: Pink
Class 3: Red
Class 4: Blue

Training

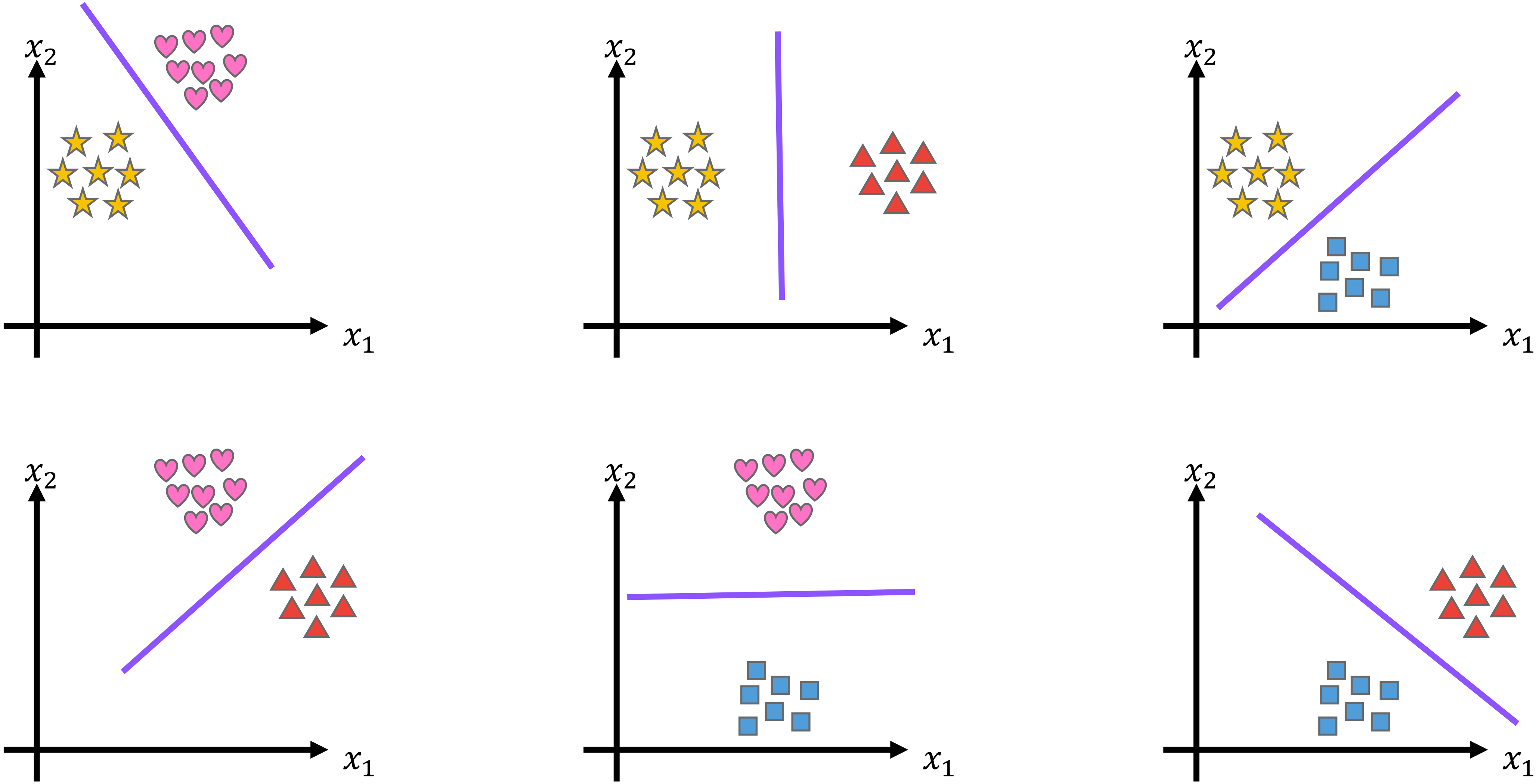
Train a **binary classifier** for *each pair of classes* from your dataset (total of k classes).

Total of classifiers = $C(k,2) = \frac{k!}{(k-2)!2!} = \frac{k(k-1)}{2}$

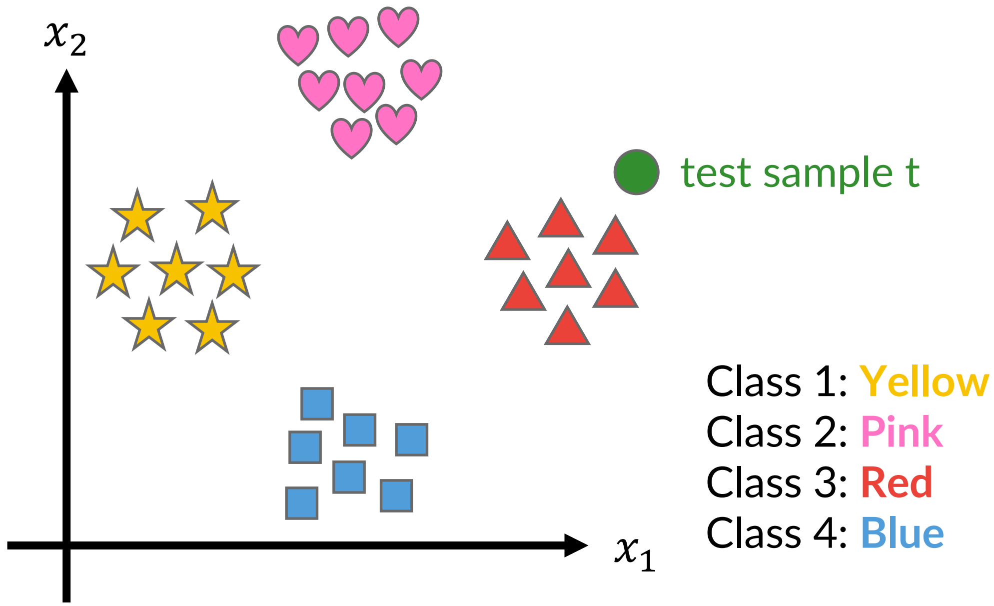
Total of classifiers = $C(4,2) = \frac{4(4-1)}{2} = 6$

Classifier 1: Yellow (1) vs Pink (2)
Classifier 2: Yellow (1) vs Red (3)
Classifier 3: Yellow (1) vs Blue (4)
Classifier 4: Pink (2) vs Red (3)
Classifier 5: Pink (2) vs Blue (4)
Classifier 6: Red (3) vs Blue (4)

One vs One



One vs One



Training

Train a **binary classifier** for *each pair of classes* from your dataset (total of k classes).

Total of classifiers = $C(k,2) = \frac{k!}{(k-2)!2!} = \frac{k(k-1)}{2}$

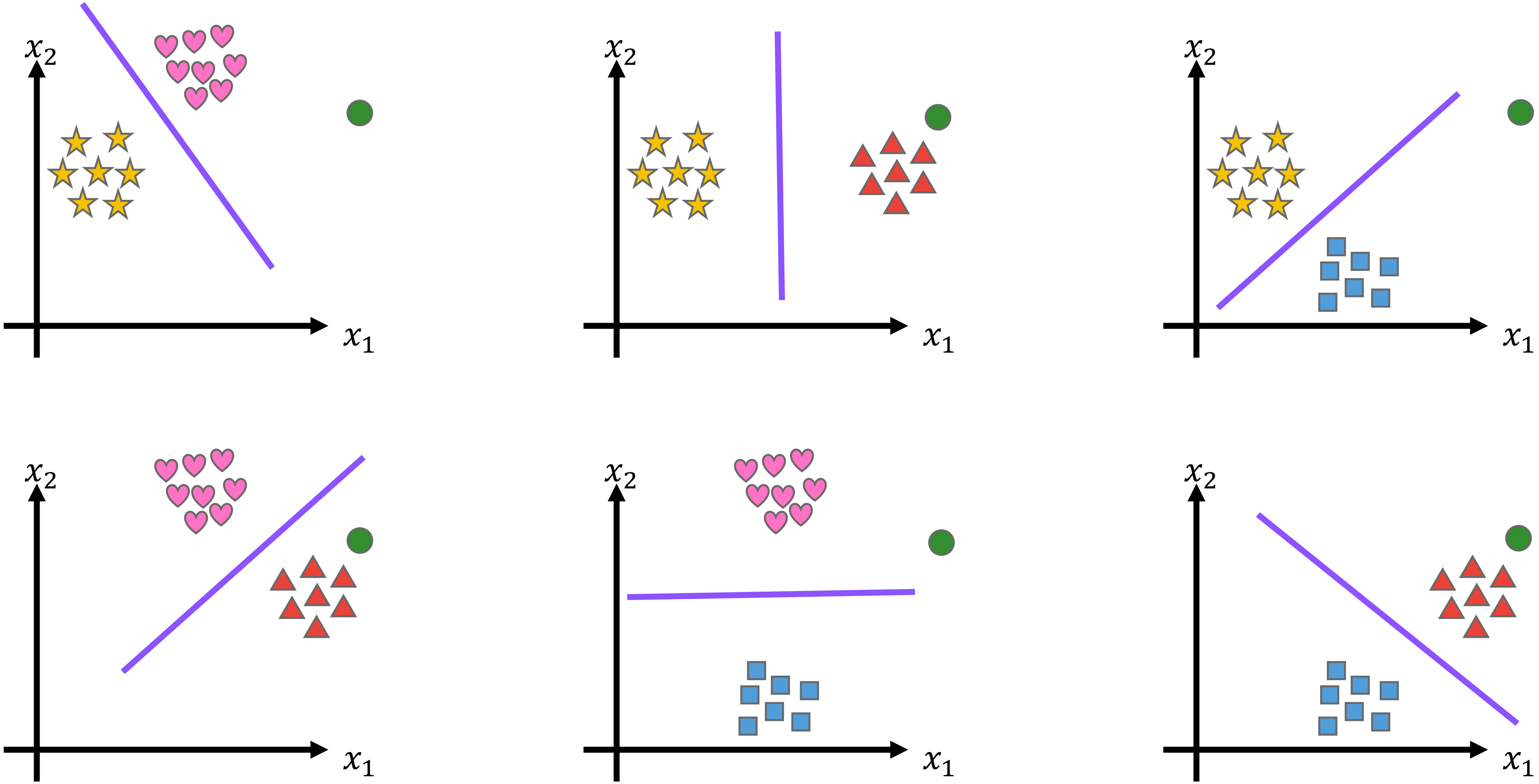
Classification/prediction

Given a test sample t:

- Classify t w.r.t. **all binary classifiers**;
- **Assign the label** from the binary classifier that **provided the highest score** (e.g., probability of belonging to its class): **argmax**

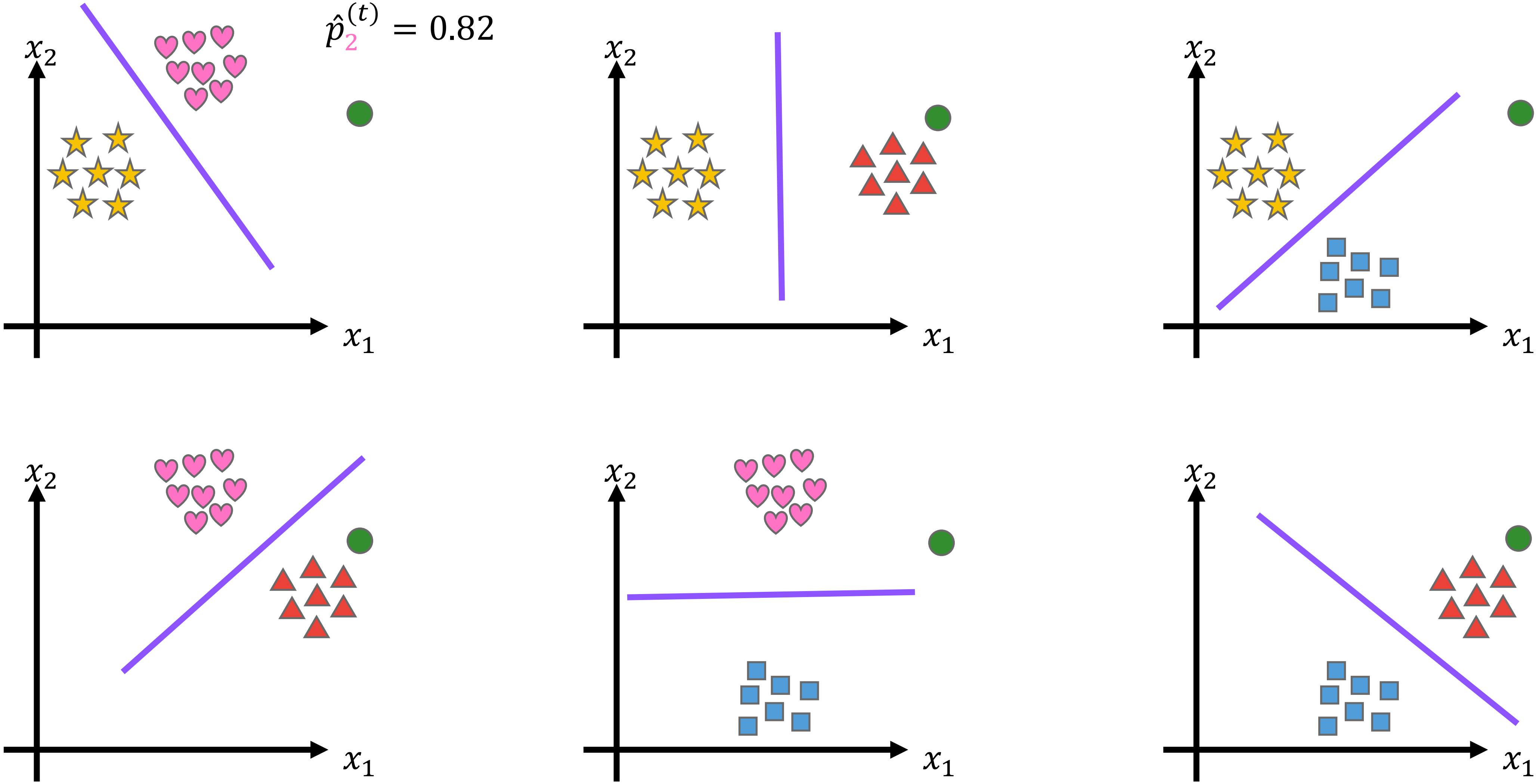
> We must also consider the **probability of** belonging to the **negative class.**

One vs One

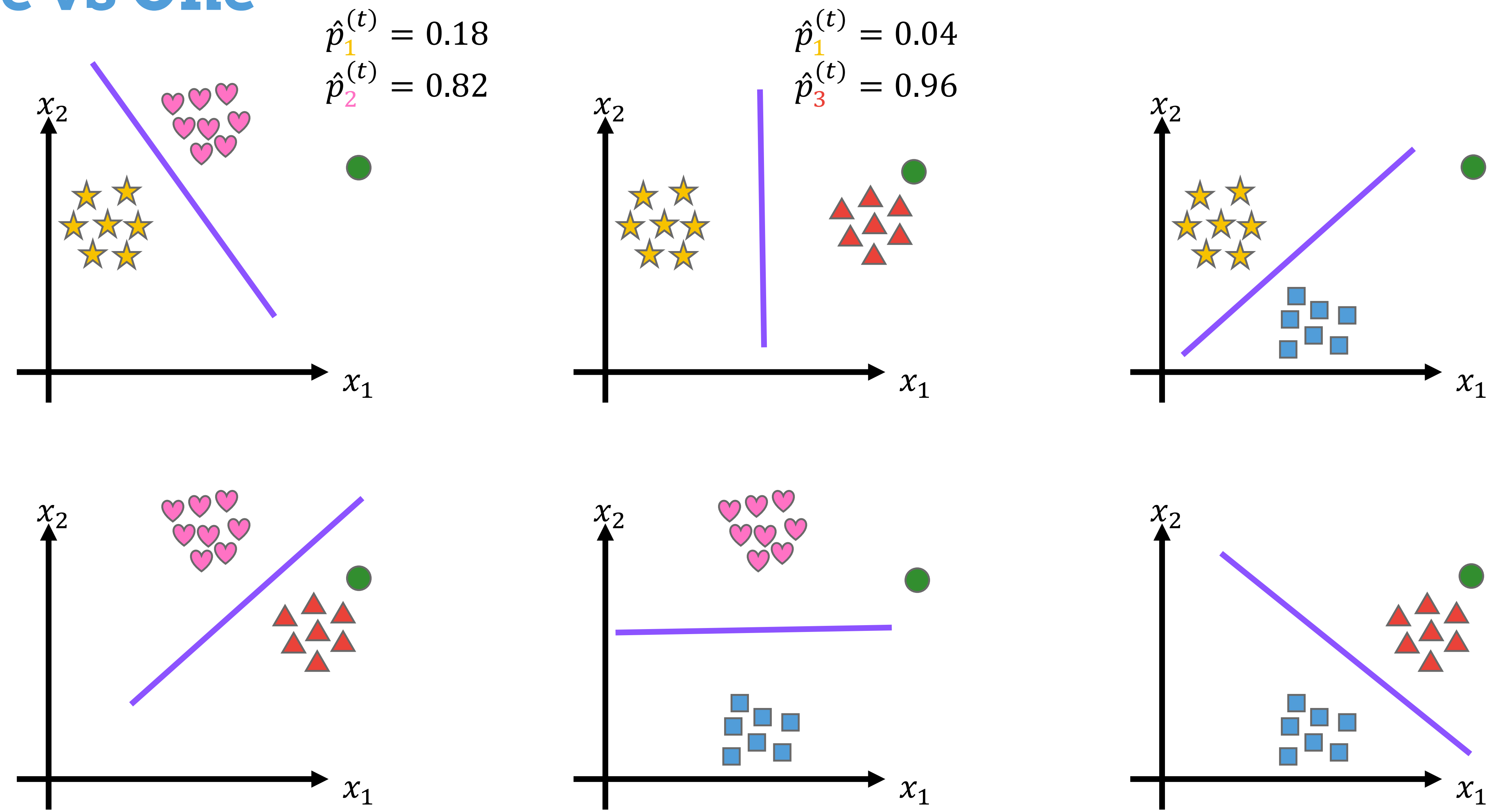


One vs One

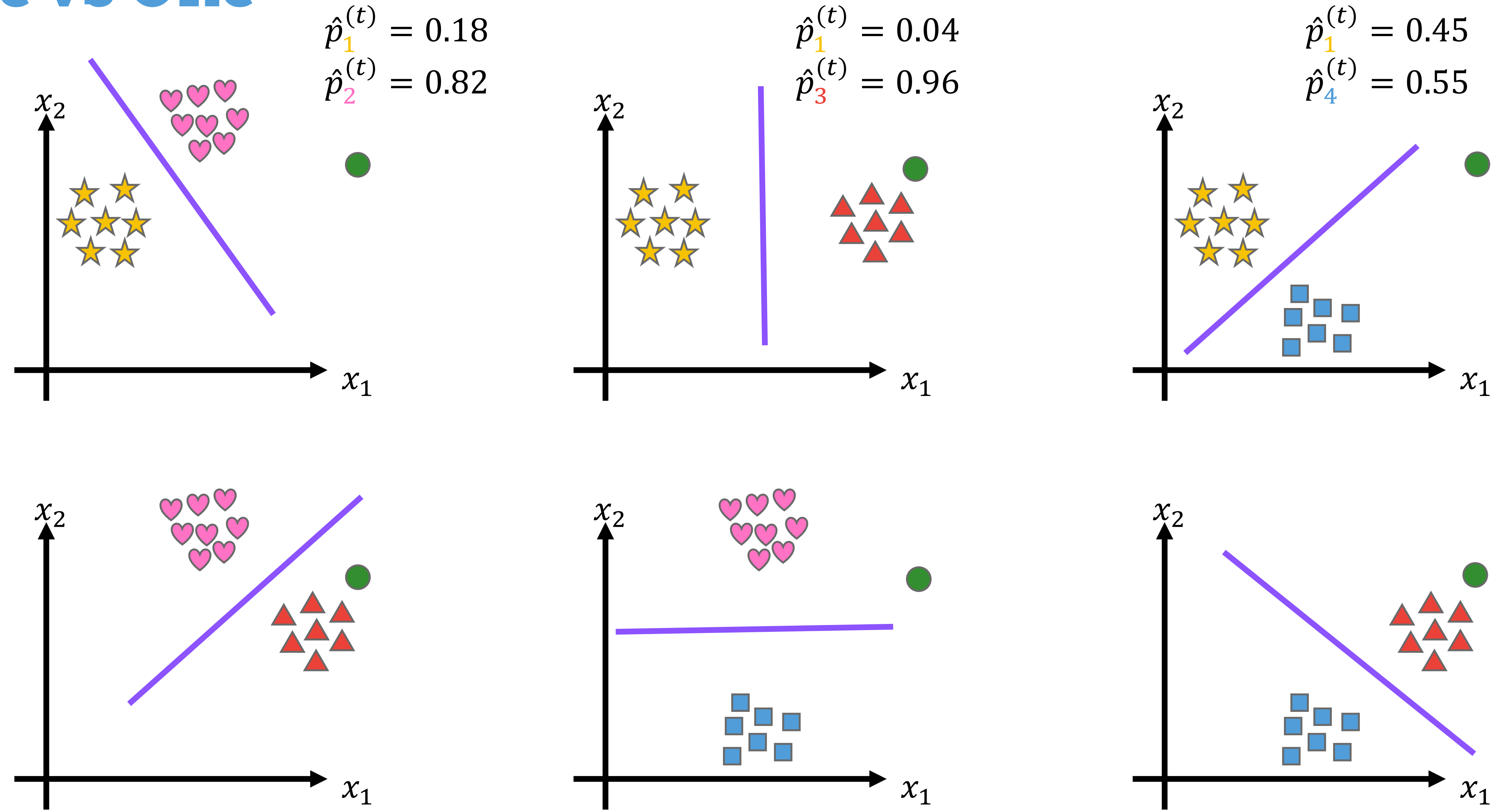
$\hat{p}_1^{(t)} = 0.18$
$\hat{p}_2^{(t)} = 0.82$



One vs One



One vs One



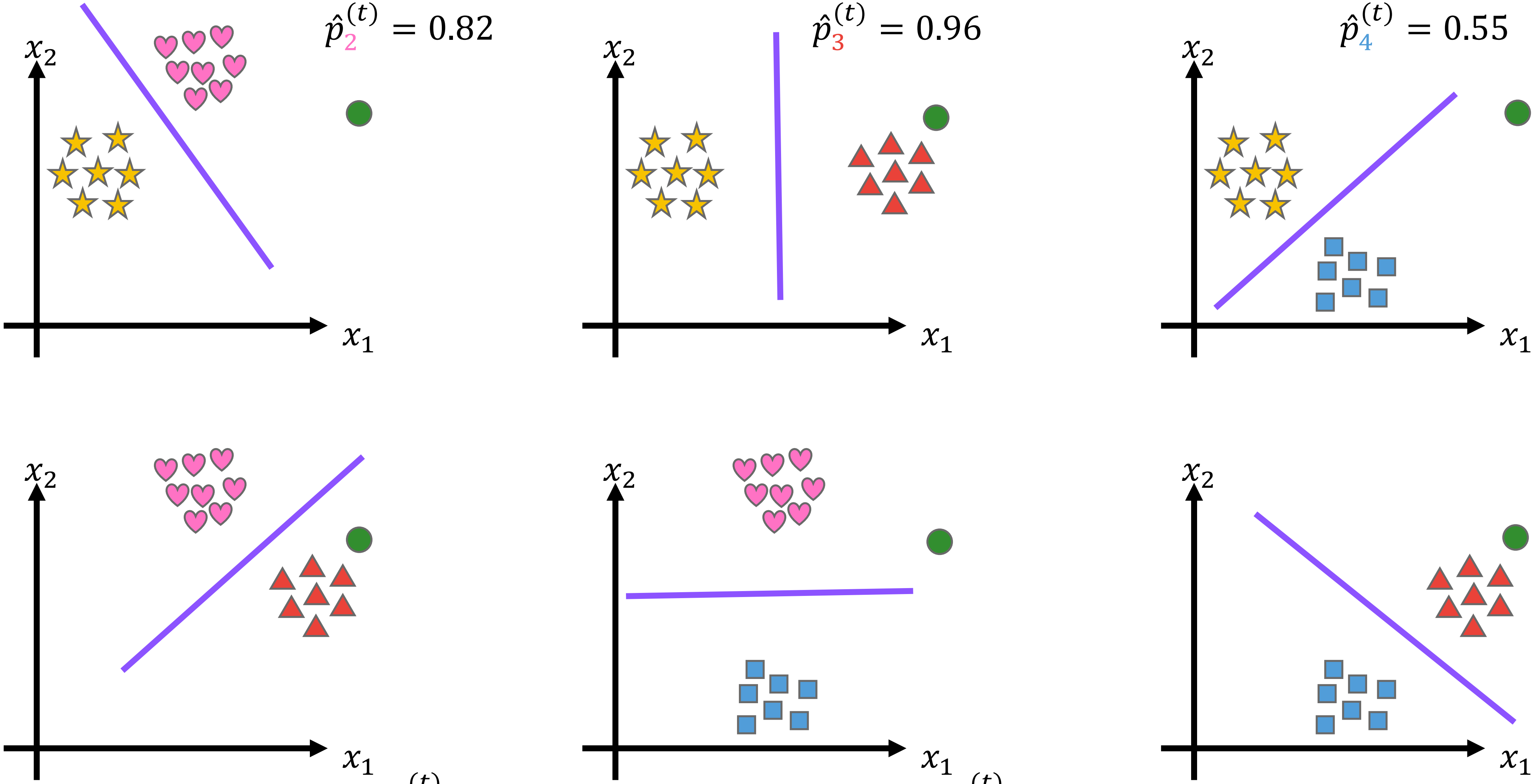
$\hat{p}_1^{(t)} = 0.18$
$\hat{p}_2^{(t)} = 0.82$
$\hat{p}_1^{(t)} = 0.04$
$\hat{p}_3^{(t)} = 0.96$
$\hat{p}_1^{(t)} = 0.45$
$\hat{p}_4^{(t)} = 0.55$
x_2
x_1

One vs One

$\hat{p}_1^{(t)} = 0.18$
$\hat{p}_2^{(t)} = 0.82$

$\hat{p}_1^{(t)} = 0.04$
$\hat{p}_3^{(t)} = 0.96$

$\hat{p}_1^{(t)} = 0.45$
$\hat{p}_4^{(t)} = 0.55$

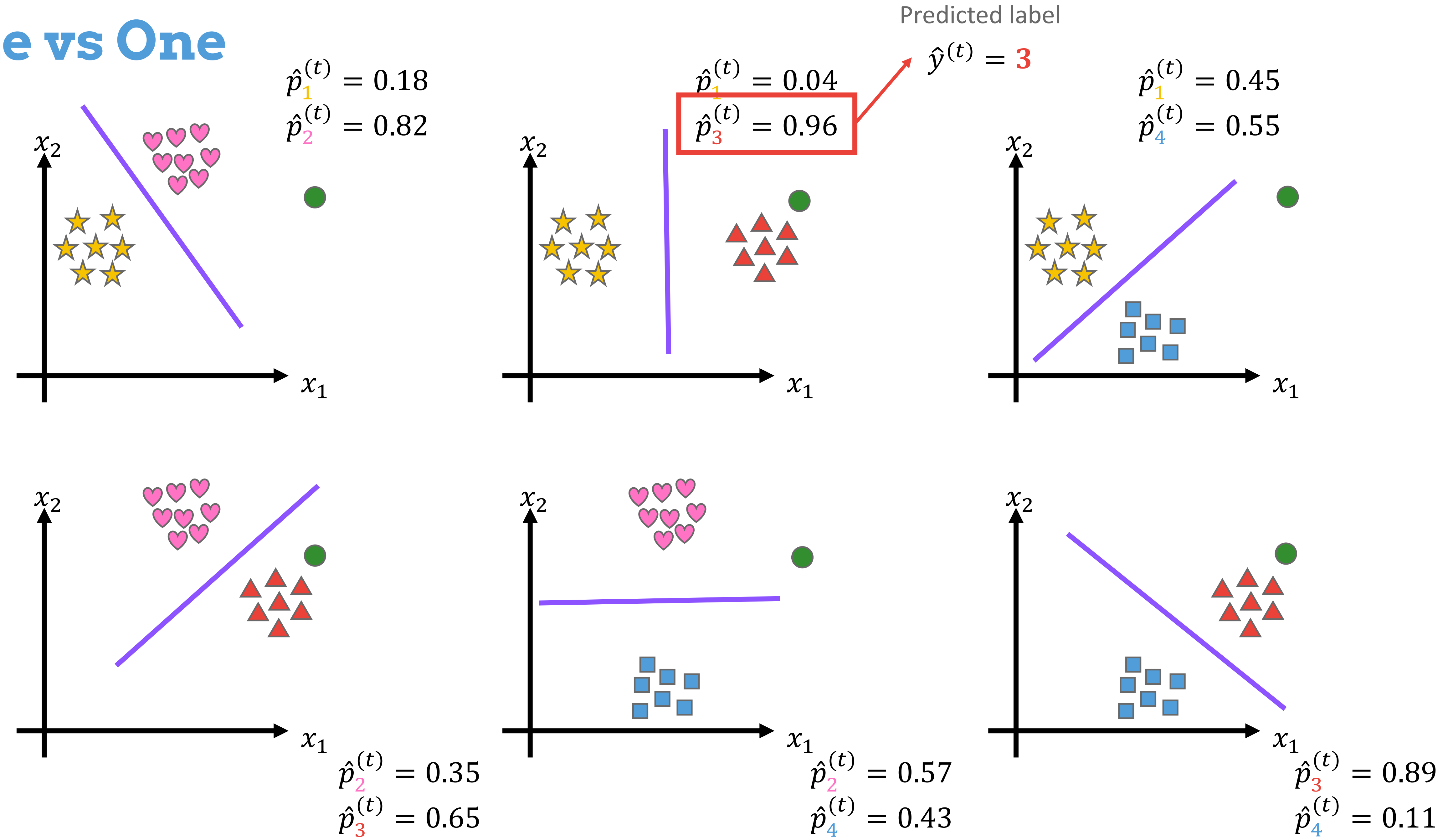


$\hat{p}_2^{(t)} = 0.35$
$\hat{p}_3^{(t)} = 0.65$

$\hat{p}_2^{(t)} = 0.57$
$\hat{p}_4^{(t)} = 0.43$

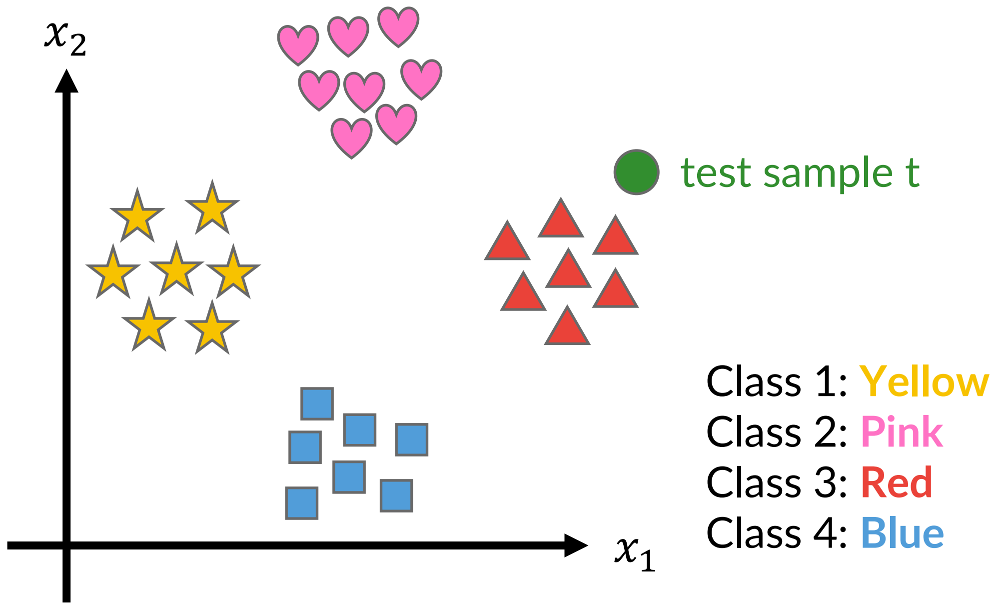
$\hat{p}_3^{(t)} = 0.89$
$\hat{p}_4^{(t)} = 0.11$

One vs One



Predicted label
$\hat{y}^{(t)} = 3$
$\hat{p}_1^{(t)} = 0.18$
$\hat{p}_2^{(t)} = 0.82$
$\hat{p}_1^{(t)} = 0.04$
$\hat{p}_3^{(t)} = 0.96$
$\hat{p}_1^{(t)} = 0.45$
$\hat{p}_4^{(t)} = 0.55$
$\hat{p}_2^{(t)} = 0.35$
$\hat{p}_3^{(t)} = 0.65$
$\hat{p}_2^{(t)} = 0.57$
$\hat{p}_4^{(t)} = 0.43$
$\hat{p}_3^{(t)} = 0.89$
$\hat{p}_4^{(t)} = 0.11$
x_1
x_2

One vs One



Training

Train a **binary classifier** for *each pair of classes* from your dataset (total of k classes).

Total of classifiers = $C(k, 2) = \frac{k!}{(k-2)!2!} = \frac{k(k-1)}{2}$

Classification/prediction

Given a test sample t:

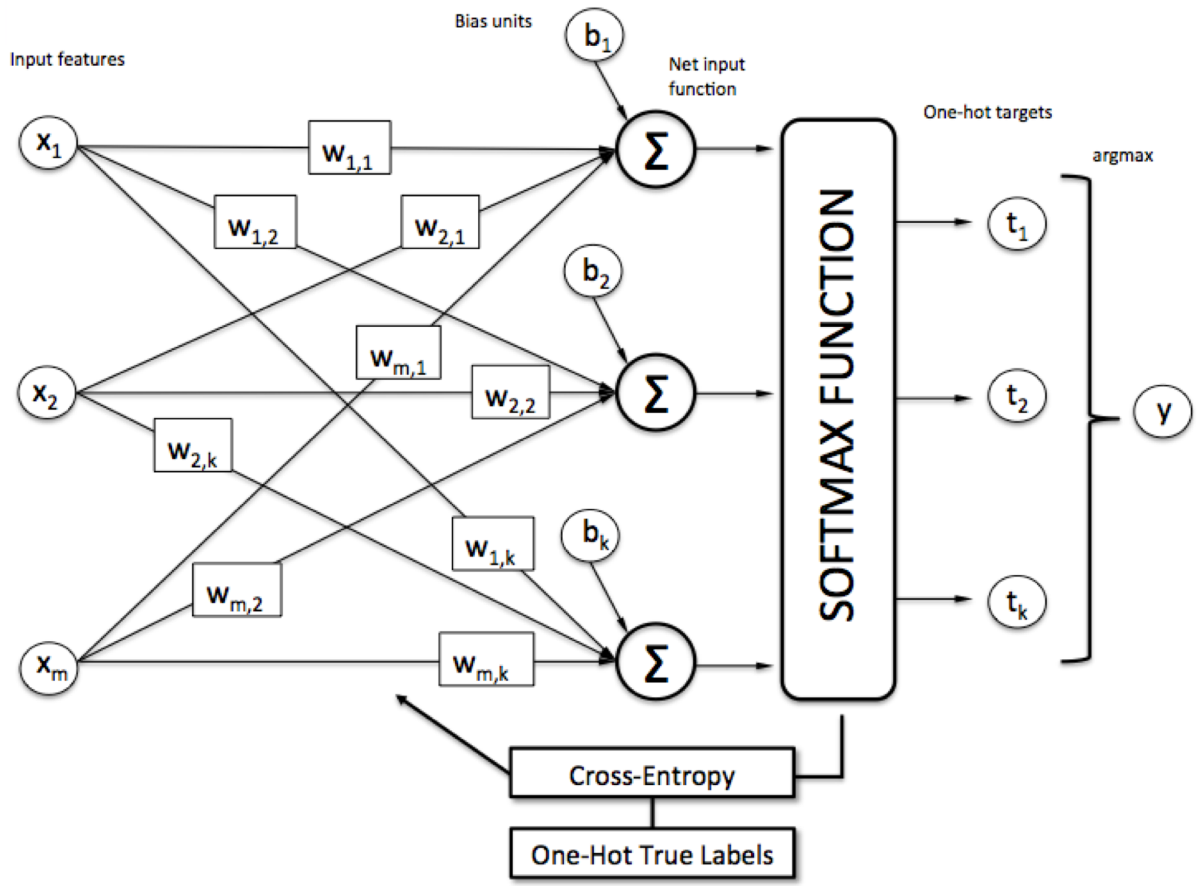
- Classify t w.r.t. **all binary classifiers**;
- **Assign the label** from the binary classifier that **provided the highest score** (e.g., probability of belonging to its class): **argmax**

We must also consider the **probability of** belonging to the **negative class**.

`sklearn.multiclass`.OneVsOneClassifier

https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html

Softmax Regression



Softmax Regression

https://www.kdnuggets.com/2016/07/softmax-regression-related-logistic-regression.html

D3APL – Aplicações em Ciência de Dados

Implementing a Linear Classifier from Scratch (Part 2)

Prof. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas