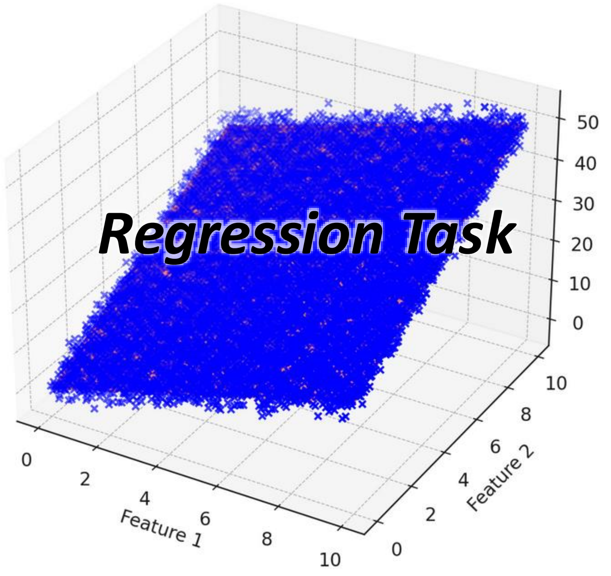
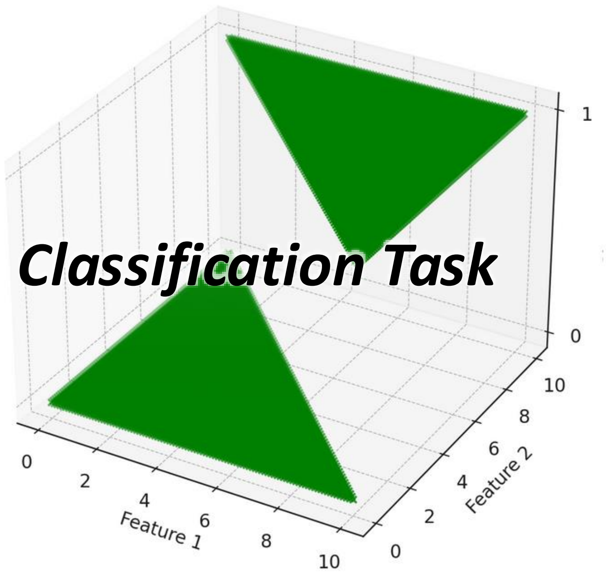
Classification versus Regression

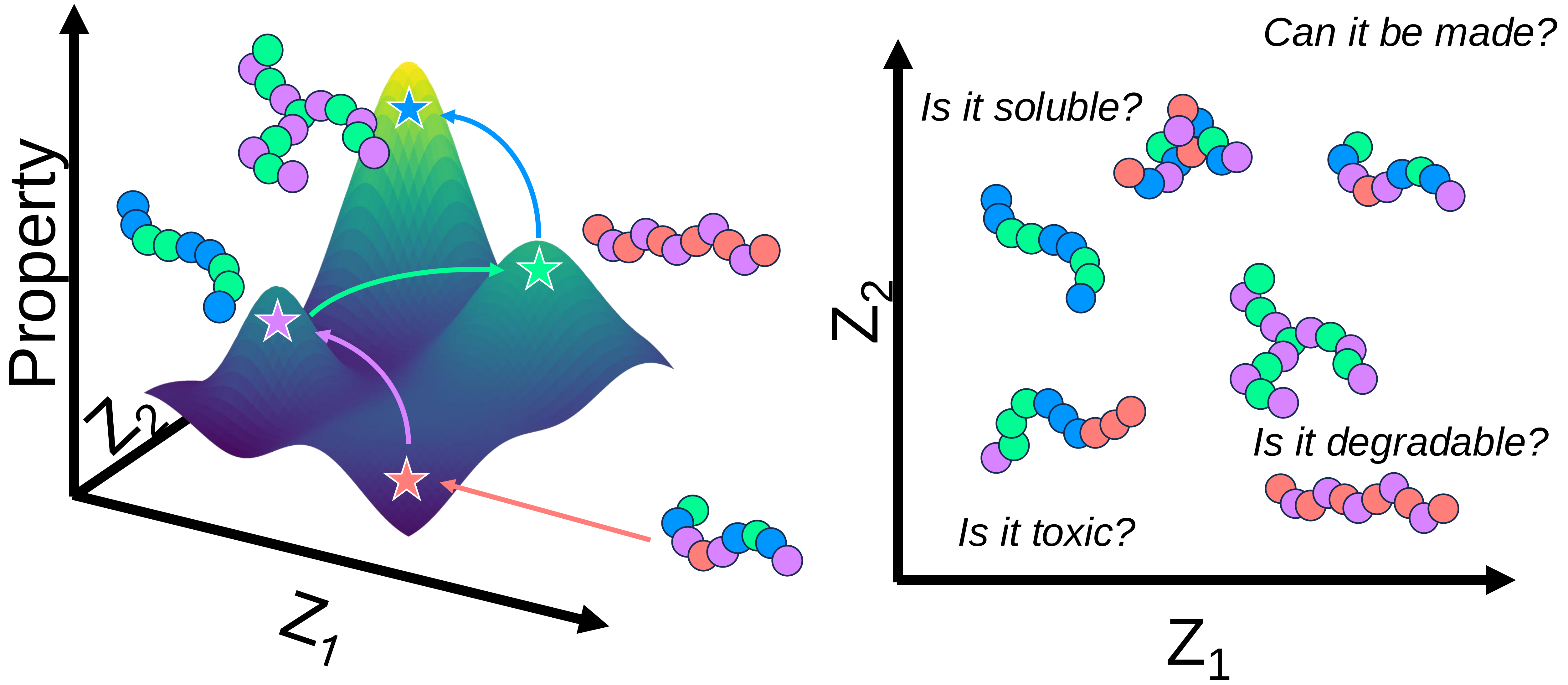


- **Output**: Predicts **continuous values** related to chemical or materials properties.
- **Goal**: Estimate a numerical value based on input features in a chemical or engineering context.
- **Examples**:
  - Predicting the **yield strength** of a new polymer based on its molecular structure.
  - Estimating the **reaction rate** of a chemical process at different temperatures. Predicting the **lifetime** or **degradation rate** of a battery under various operating conditions.

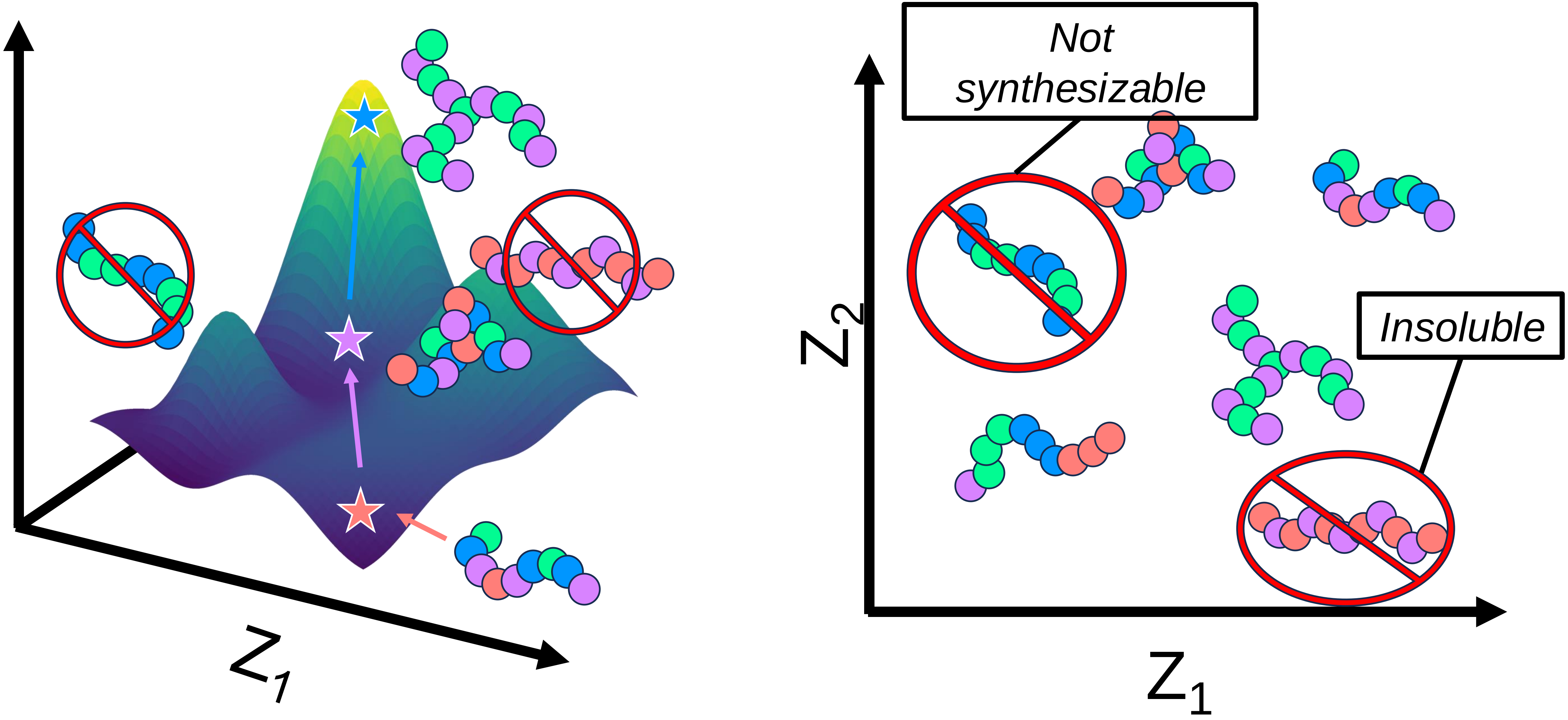


- **Output**: Predicts **discrete labels** or **categories** related to chemical or materials performance.
- **Goal**: Assign input data to one of several predefined categories in the field of chemistry or materials science.
- **Examples**:
  - Determining whether a chemical reaction will be **exothermic** or **endothermic** given certain reactants and conditions.
  - Classifying polymers as **soluble** or **insoluble** in a particular solvent.
  - Predicting whether a material will be **brittle** or **ductile** based on its microstructure.

Classification in Chemical/Materials Optimization

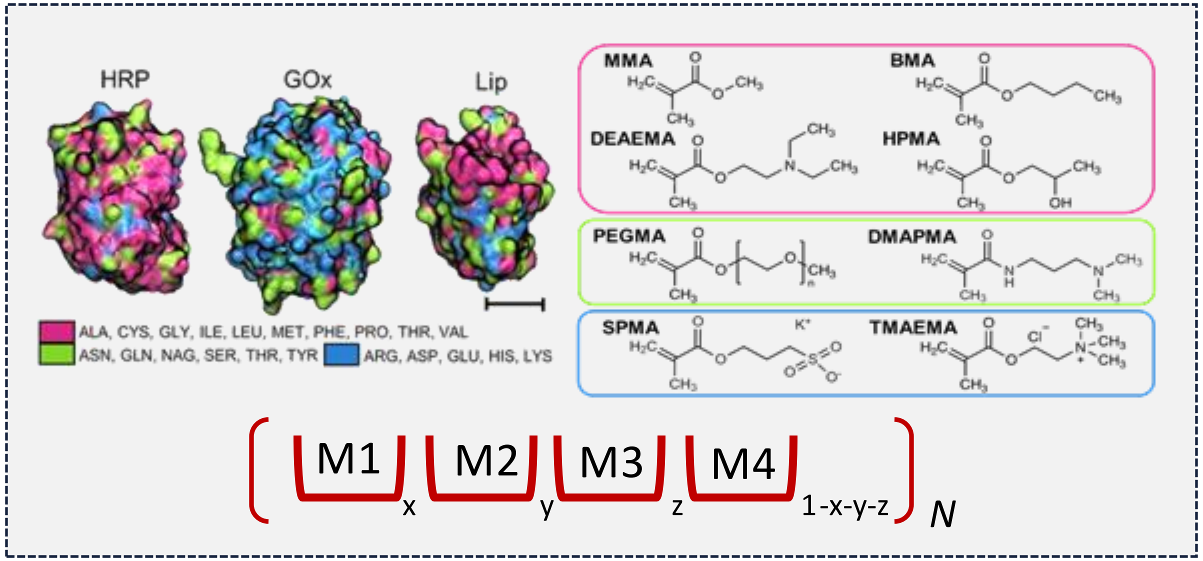


Classification in Chemical/Materials Optimization



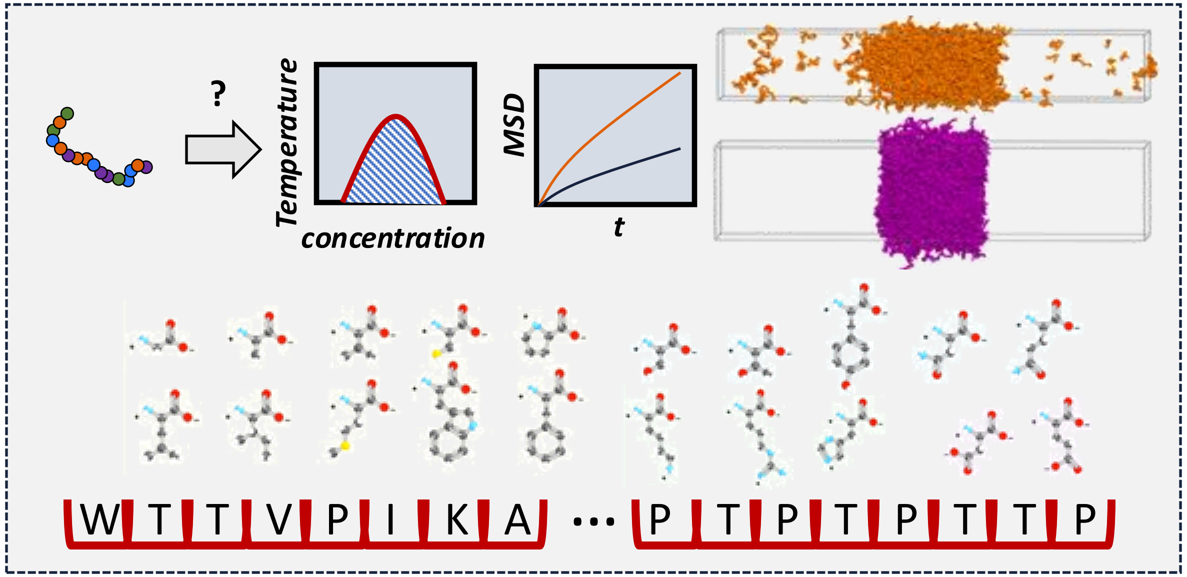
Tangible Motivating Examples

composition design

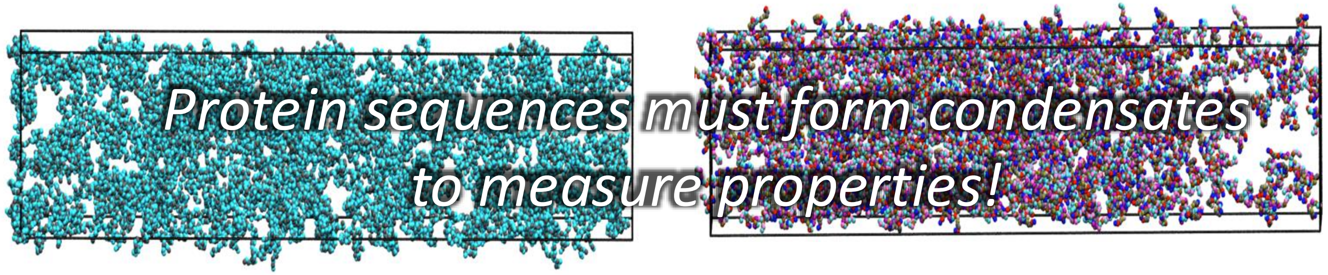


Goal: *Design copolymers that enhance enzyme stability or robustness to stress*

sequence design



Goal: *Explore physical bounds of materials properties of single-component condensates*

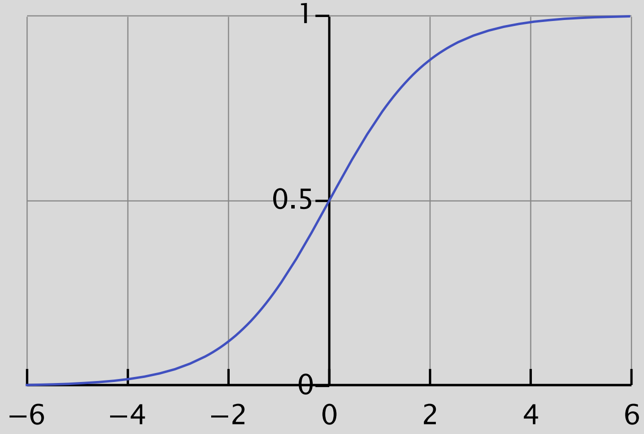


Logistic Regression: Pathway towards Classification

In ***logistic regression***, we want to restrict our predictions to be on the interval [0,1] to represent probabilities of a class

Logistic Function

$$f(x) = \frac{L}{1 + e^{-k(x - x_0)}}$$



$L = 1$, $k = 1$, $x_0 = 0$

other Sigmoid shapes can be used for analogous purpose

The essential premise of a ***logistic model*** is to represent the ***log-odds*** of a label as a linear combination of the features

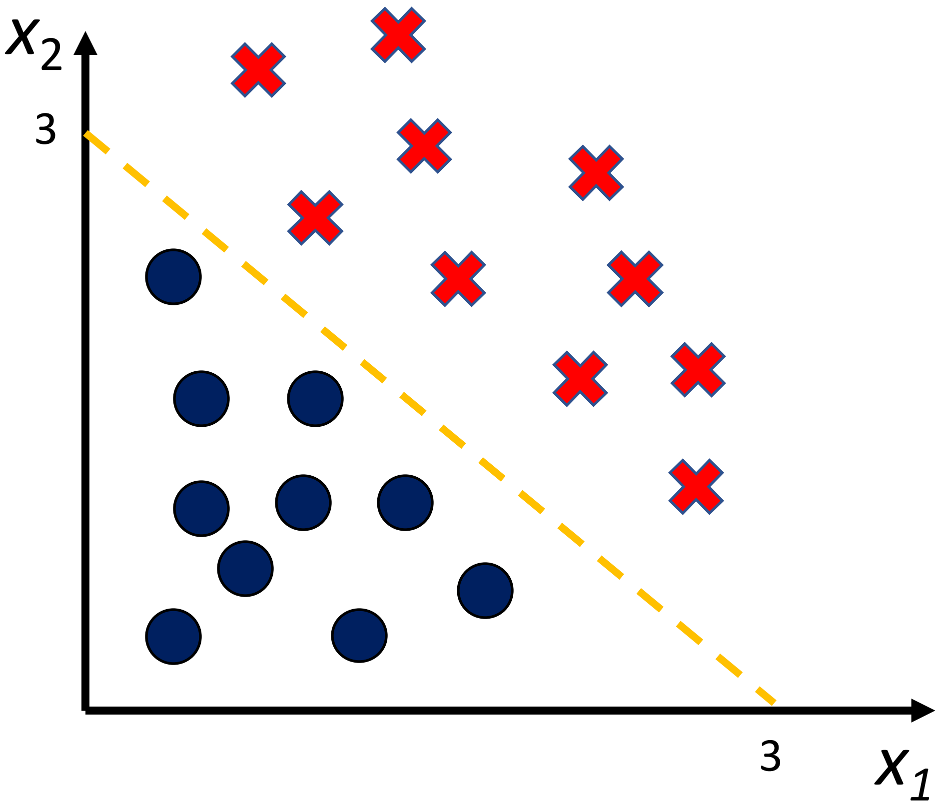
$$\ell = \log_b \frac{p}{1 - p} = \boldsymbol{x}^T \boldsymbol{\theta}$$

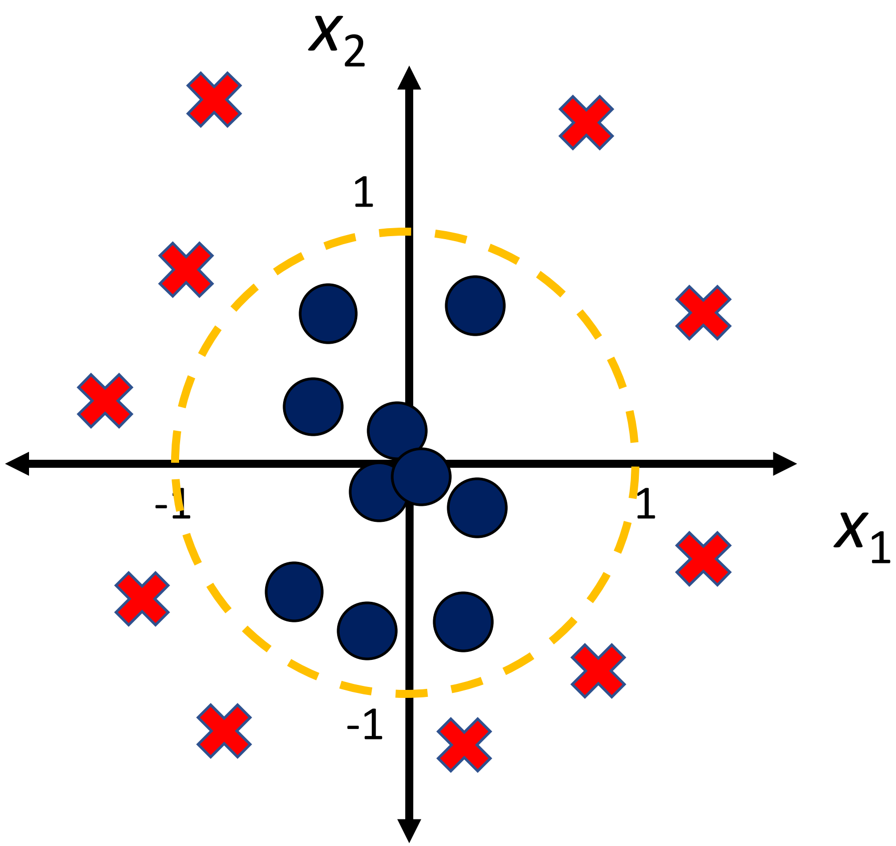
$$\implies p = \frac{1}{1 + b^{-\boldsymbol{x}^T \boldsymbol{\theta}}} \xrightarrow{b=e} \frac{1}{1 + e^{-\boldsymbol{x}^T \boldsymbol{\theta}}}$$

Model predictions: $\hat{y} \leftarrow f(x) = p(y = 1 | \boldsymbol{x}, \boldsymbol{\theta})$

for f(x) = 0.7, we interpret that to mean a 70% chance that y = 1

Parameterizing Decision Boundaries





$$f(x_1, x_2) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}}$$

what would be a good set of thetas?

$$f(x_1, x_2) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)}}$$

what about in this case?

Approaching a Cost Function

As in linear regression, we will identify optimal parameters via minimization of an appropriate cost function; here, we have something to think about

Suppose model predictions are supplied via

$$\hat{y} \leftarrow f(x) = p(y = 1|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}^T\boldsymbol{\theta}}}$$

Can you anticipate any potential issues with our previous mean-squared error metric?

$$\mathcal{E}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

As an alternative, we might consider

$$\mathcal{E}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\varepsilon(\hat{y}_i, y_i)$$

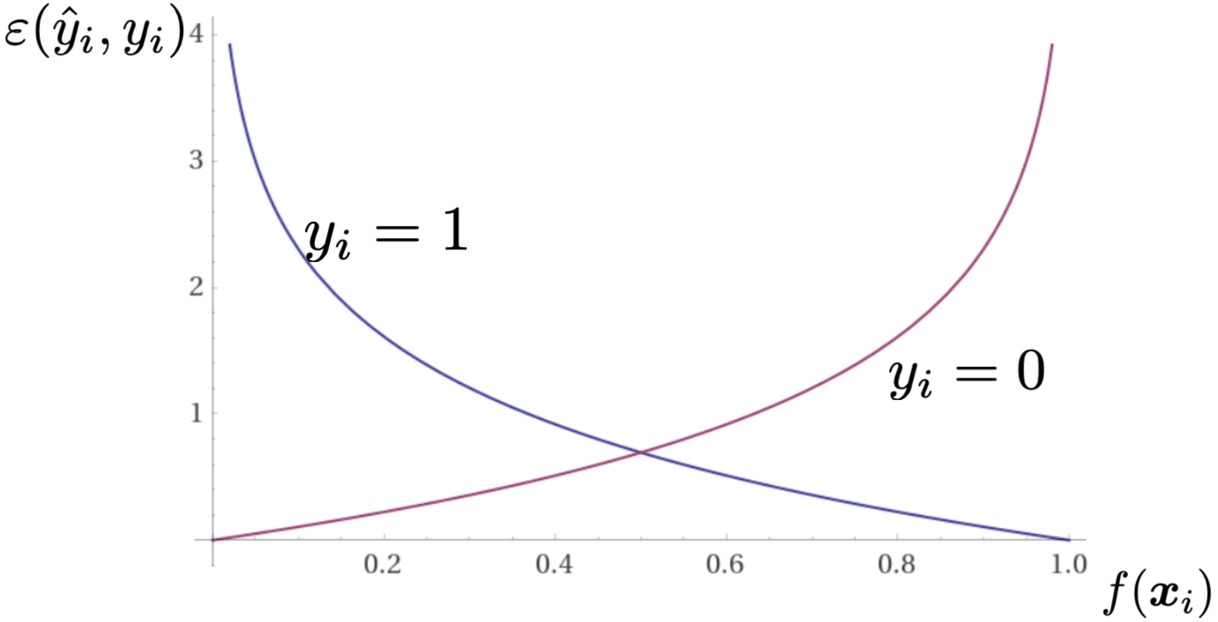
$$\varepsilon(\hat{y}_i, y_i) = \begin{cases} -\log\left[f(\boldsymbol{x}_i)\right], & \text{if } y_i = 1 \\ -\log\left[1 - f(\boldsymbol{x}_i)\right], & \text{if } y_i = 0 \end{cases}$$

Approaching a Cost Function

$$\hat{y} \leftarrow f(x) = p(y = 1|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}^T\boldsymbol{\theta}}}$$

$$\mathcal{E}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \varepsilon(\hat{y}_i, y_i)$$

$$\varepsilon(\hat{y}_i, y_i) = \begin{cases} -\log\left[f(\boldsymbol{x}_i)\right], & \text{if } y_i = 1 \\ -\log\left[1 - f(\boldsymbol{x}_i)\right], & \text{if } y_i = 0 \end{cases}$$



Minimizing the Cost Function

$$\varepsilon(\hat{y}_i, y_i) = -y_i \log f(\boldsymbol{x}_i) - (1 - y_i) \log [1 - f(\boldsymbol{x}_i)]$$

$$\mathcal{E}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \varepsilon(\hat{y}_i, y_i) \qquad \hat{y} \leftarrow f(x) = p(y = 1 | \boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}^T \boldsymbol{\theta}}}$$

Suppose we were to use gradient descent

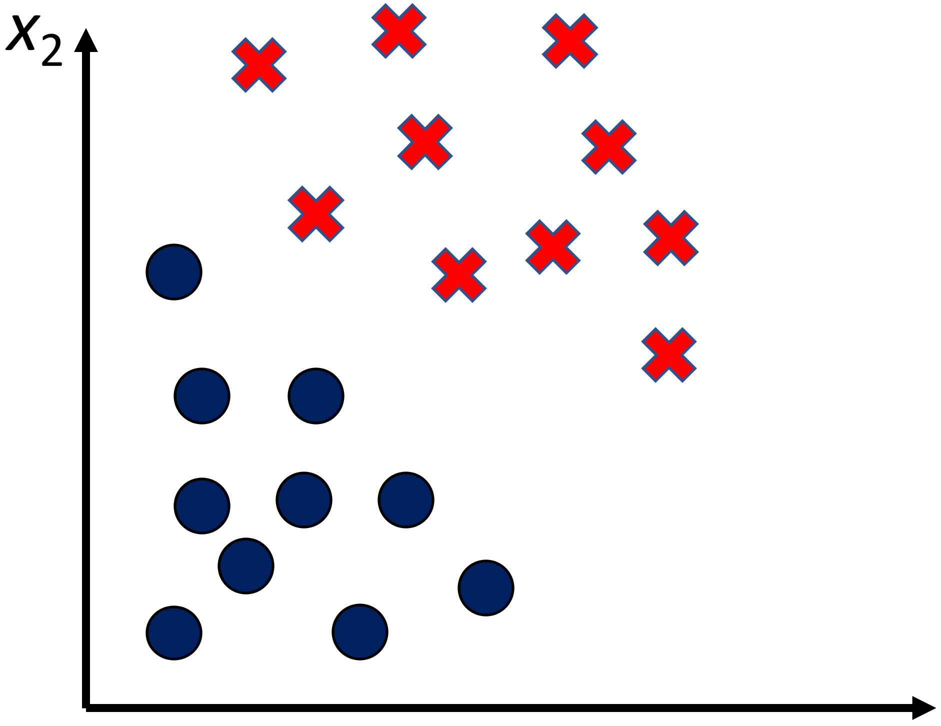
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma_i \left[\nabla_{\boldsymbol{\theta}} \mathcal{E}(\boldsymbol{\theta}) \right]$$

$$\vdots$$

$$\boldsymbol{\theta}_j \leftarrow \boldsymbol{\theta}_j - \gamma_i \sum_{i=1}^{n} \left[f(\boldsymbol{x}_i) - y_i \right] (\boldsymbol{x}_i)_j$$

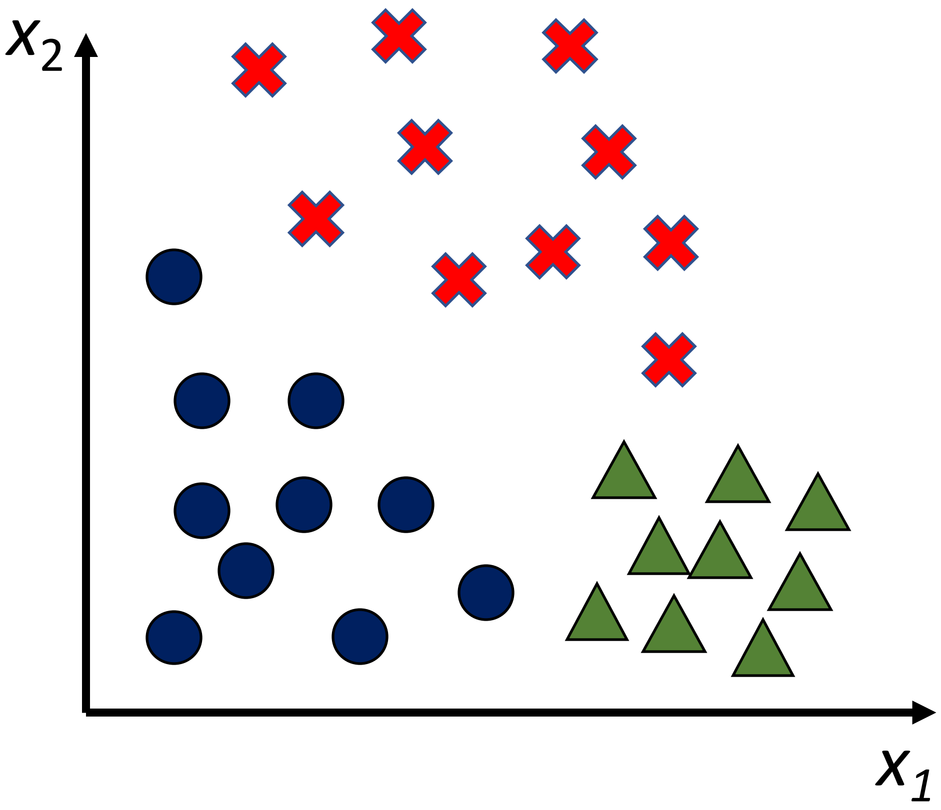
This is the same result as for linear regression!

Application now to classification



- **Binary case:** we just need to find the optimal decision boundary that partitions these classes

Application now to classification: One vs. All



- **Binary case:** we just need to find the optimal decision boundary that partitions these classes

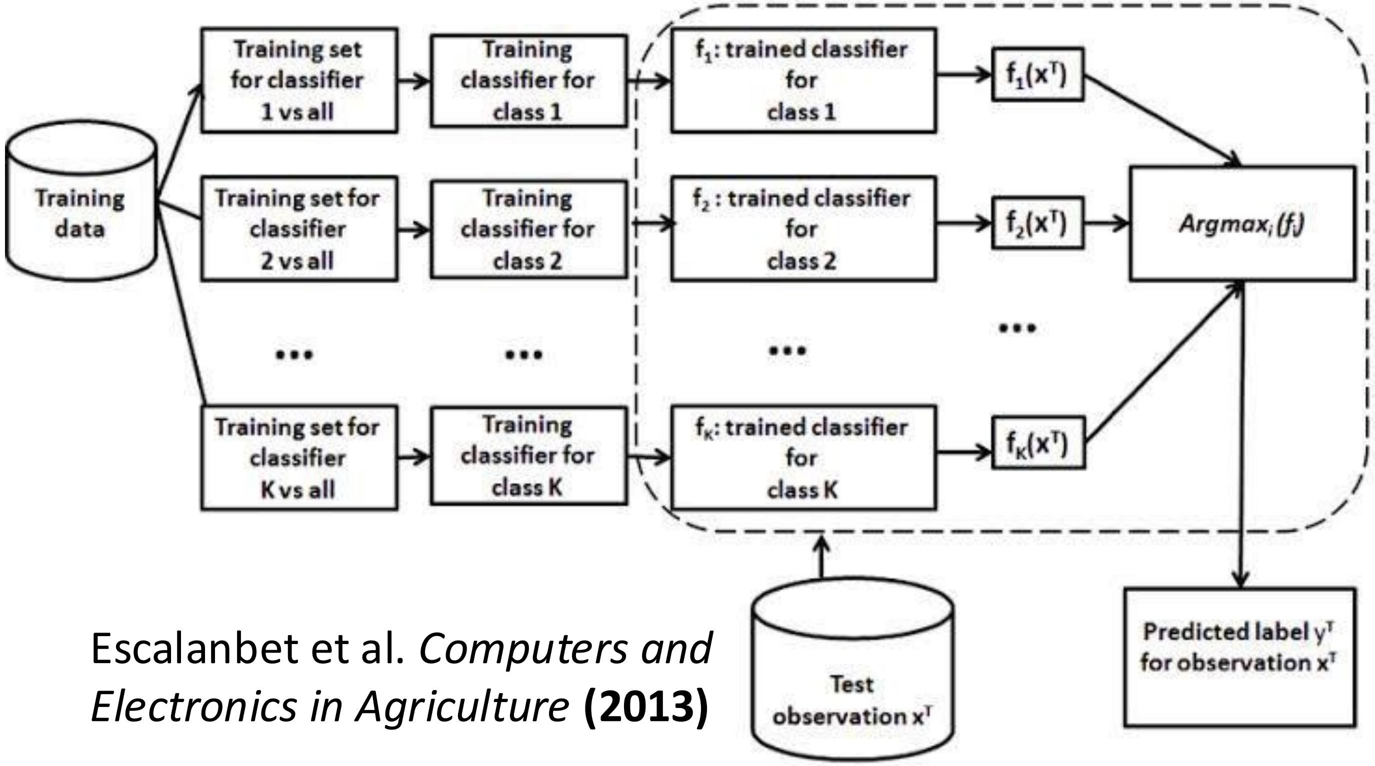
what do we do for multiple classes?

- **Multiclass case:** there are multiple strategies

Consider our data to have N classes…

One vs. All (Rest)

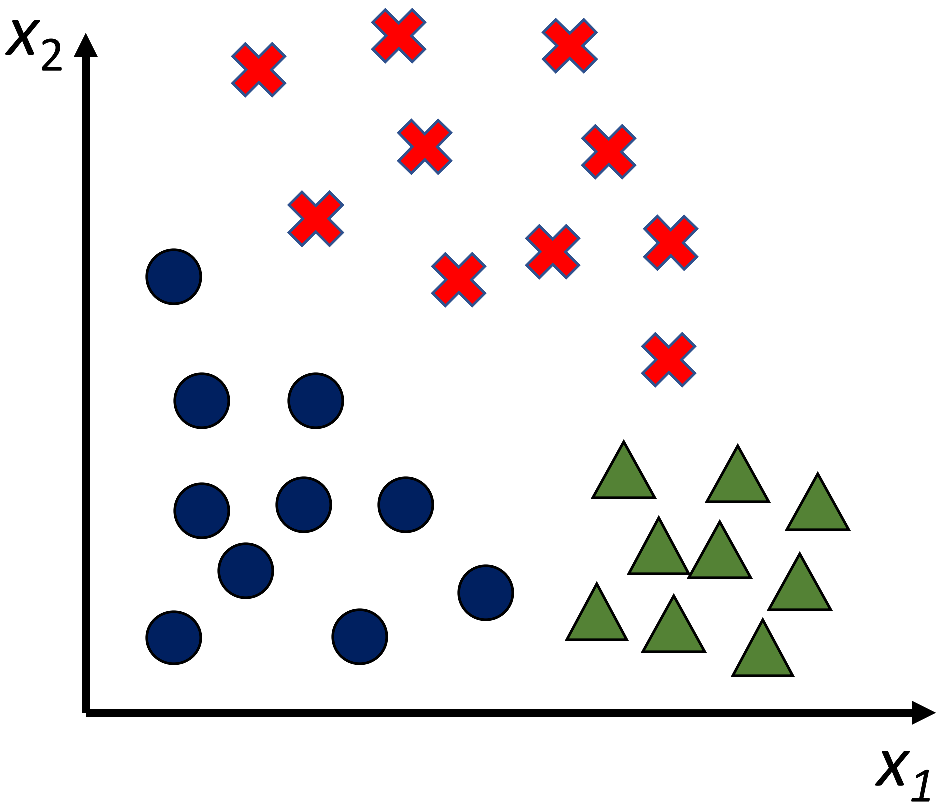
- formulate *N* binary classifier models



Escalanbet et al. *Computers and Electronics in Agriculture* **(2013)**

- pick the class that exhibits the highest score

Application now to classification: One vs. One



- **Binary case:** we just need to find the optimal decision boundary that partitions these classes

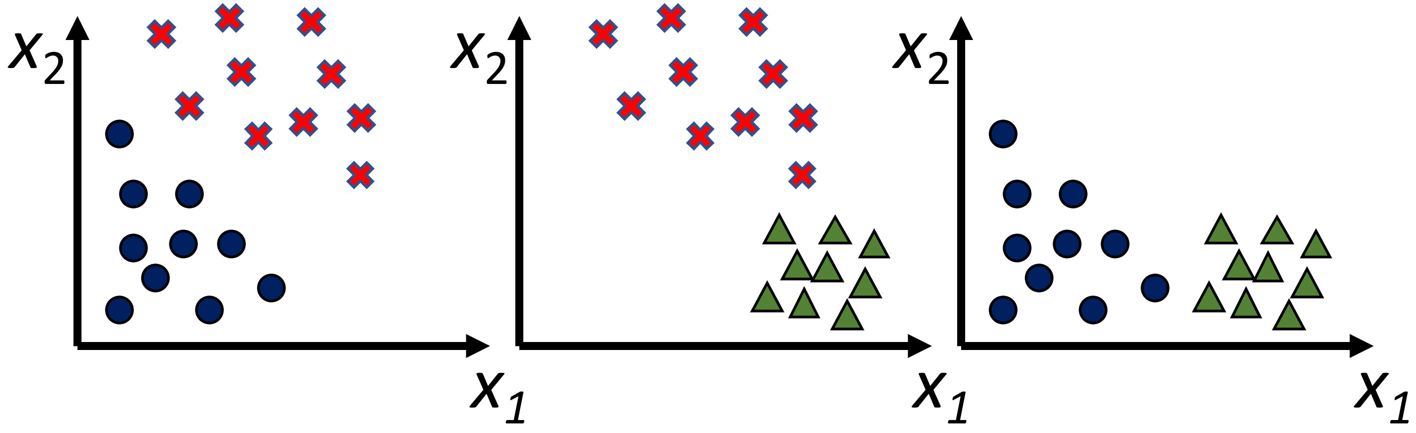
what do we do for multiple classes?

- **Multiclass case:** there are multiple strategies

Consider our data to have N classes…

One vs. One

- formulate *N(N-1)/2* binary classifier models



- pick the class that receives the most positive identifications

Logistic Regression Classification in scikit_learn

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression

# define problem
n      = 500 # number of data points
m      = 10  # size of feature vector
Nclass = 5   # number of classes
model_type = 'ovr' # ovr = one versus rest (examine other options)

# construct dataset
X_train, y_train = make_classification(n_samples=ndata,
        n_features=m,
        n_classes=Nclass,
        n_redundant=m/2,
        n_informative=m/2)

# define a model
myModel = LogisticRegression(multi_class='ovr')

# train the model
myModel.fit(X_train,y_train)

# check outcome of trained model
y_pred = model.predict(X_train)
```