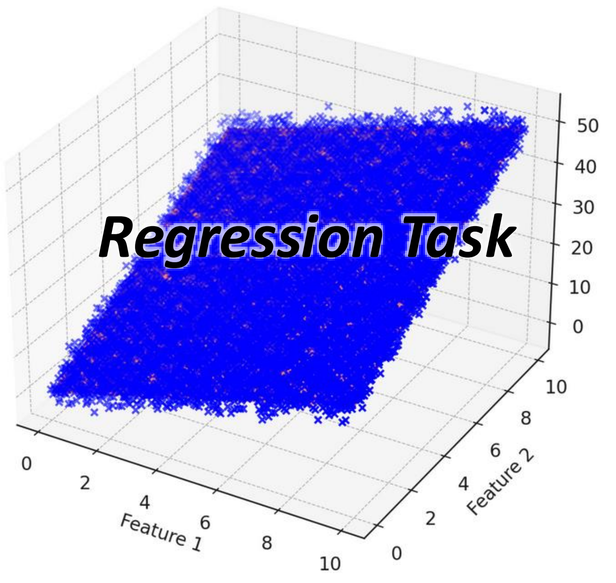
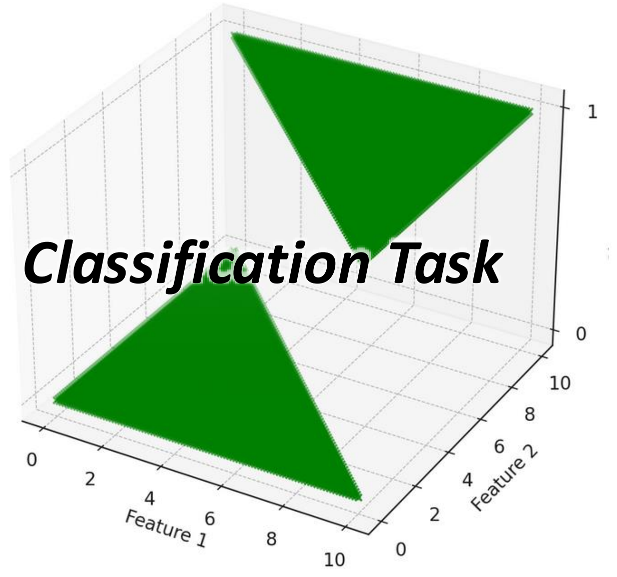
Classification versus Regression

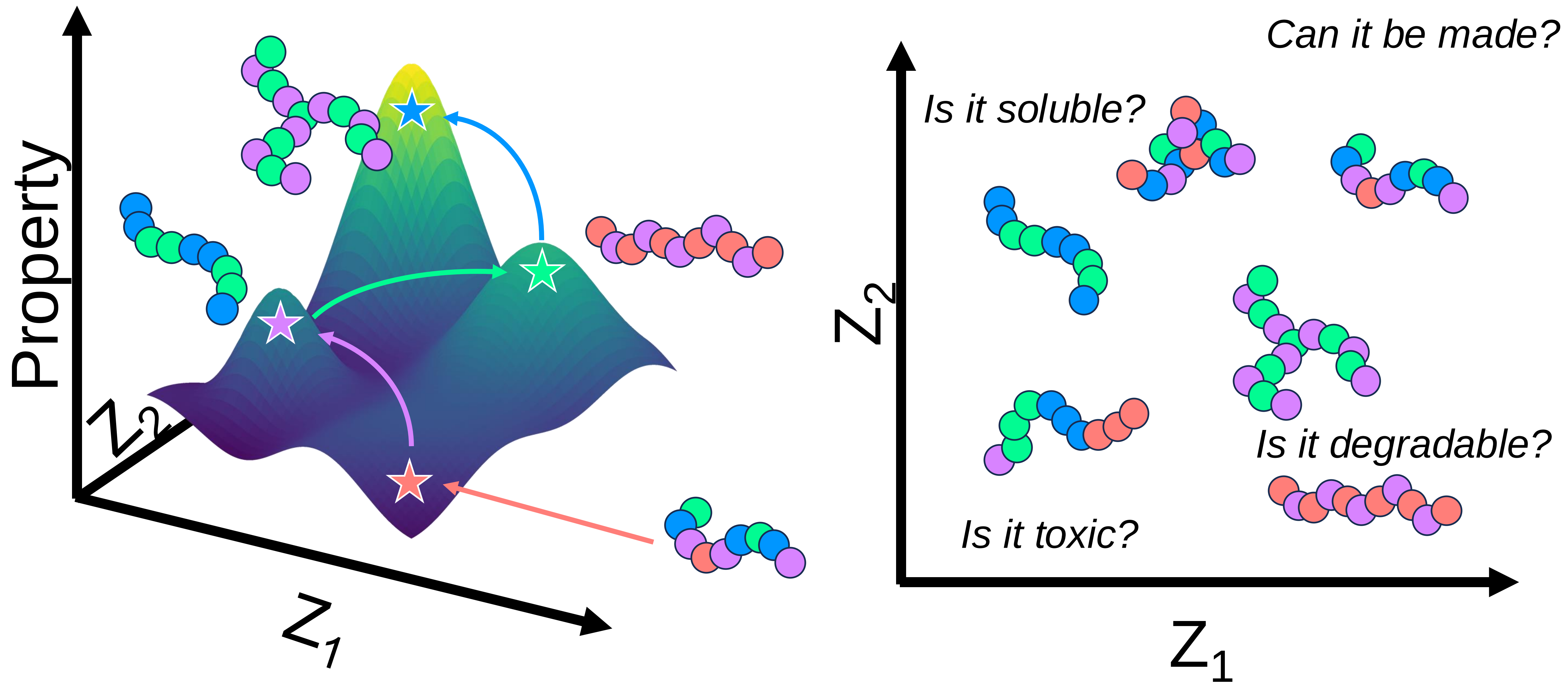


- **Output**: Predicts **continuous values** related to chemical or materials properties.
- **Goal**: Estimate a numerical value based on input features in a chemical or engineering context.
- **Examples**:
  - Predicting the **yield strength** of a new polymer based on its molecular structure.
  - Estimating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

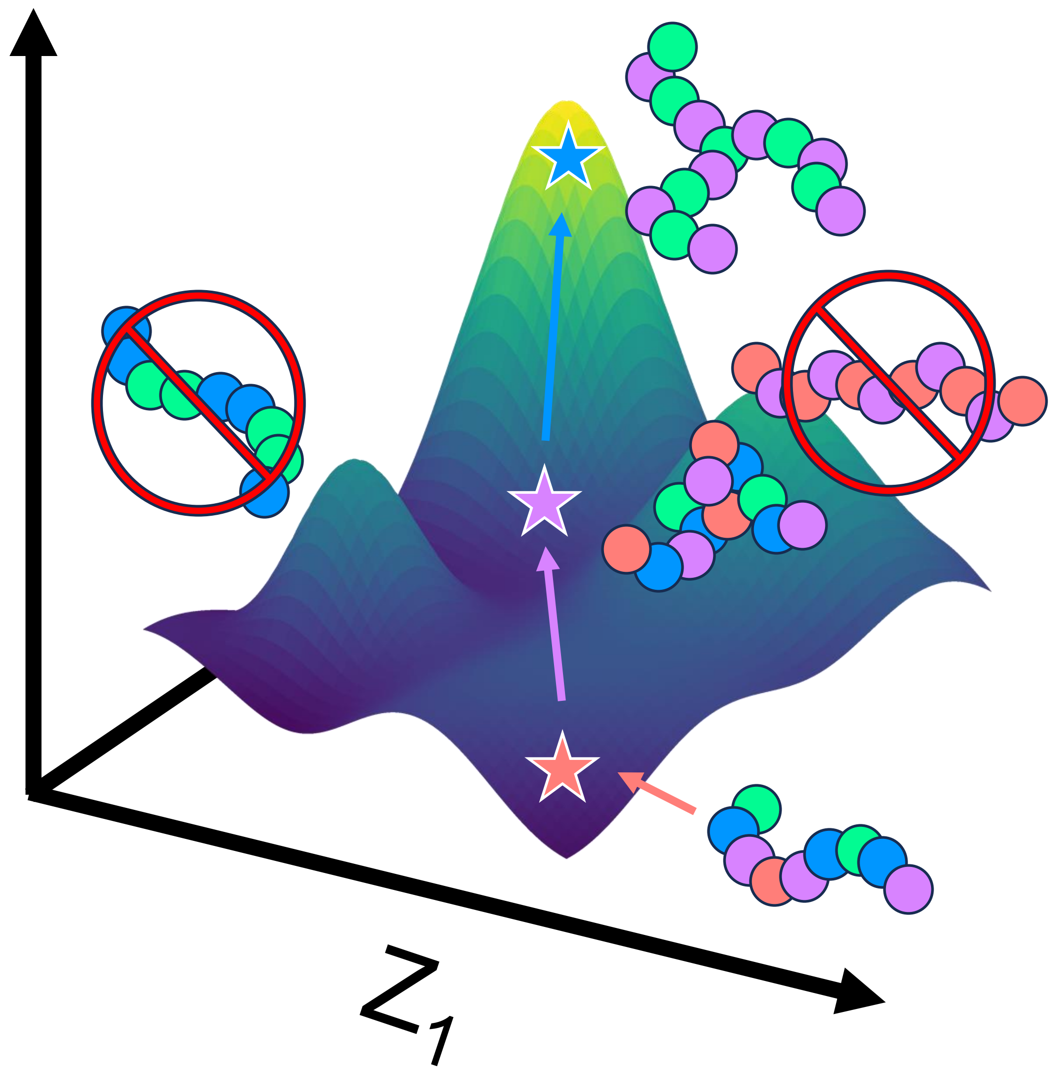


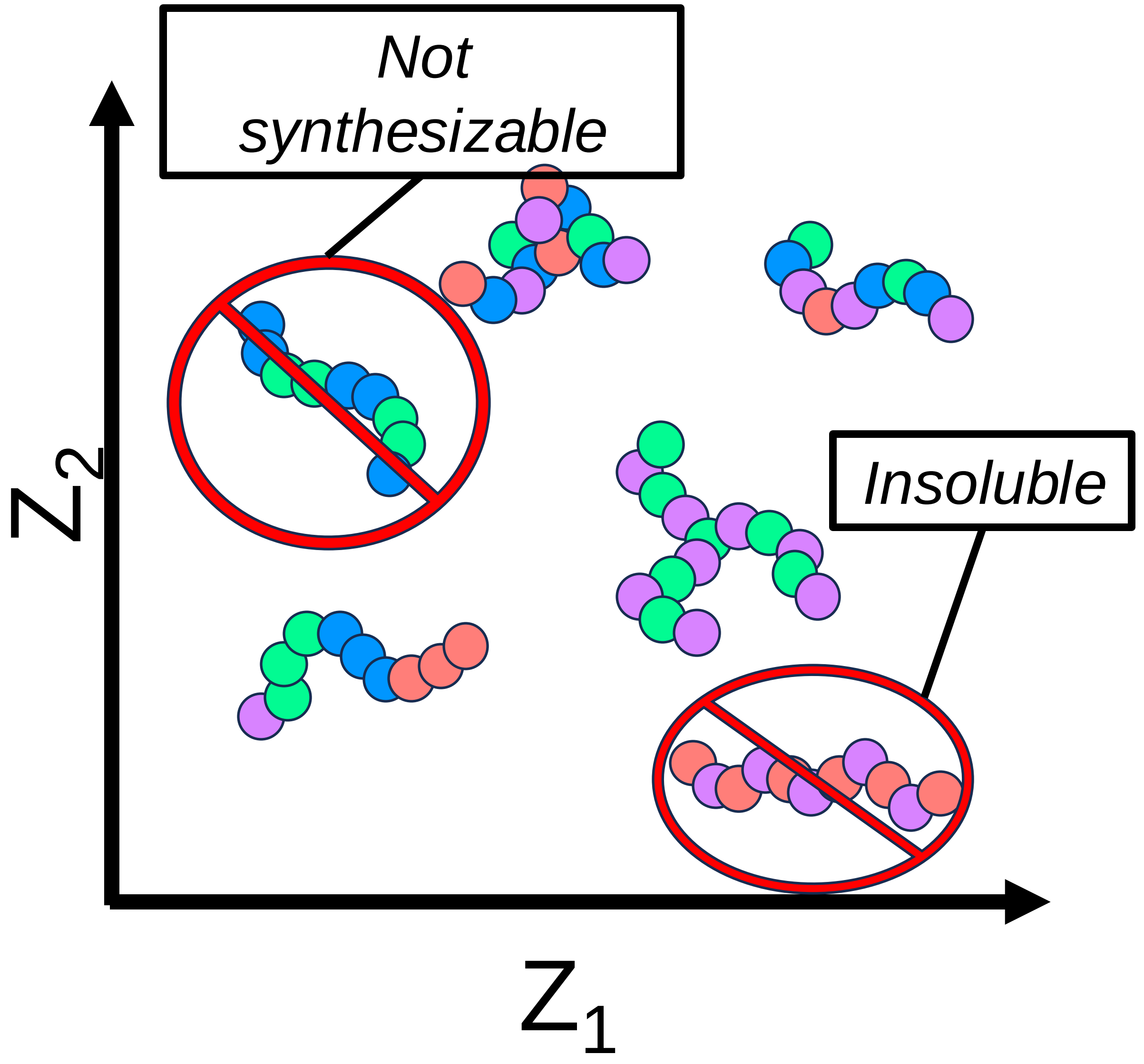
- **Output**: Predicts **discrete labels** or **categories** related to chemical or materials performance.
- **Goal**: Assign input data to one of several predefined categories in the field of chemistry or materials science.
- **Examples**:
  - Determining whether a chemical reaction will be **exothermic** 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

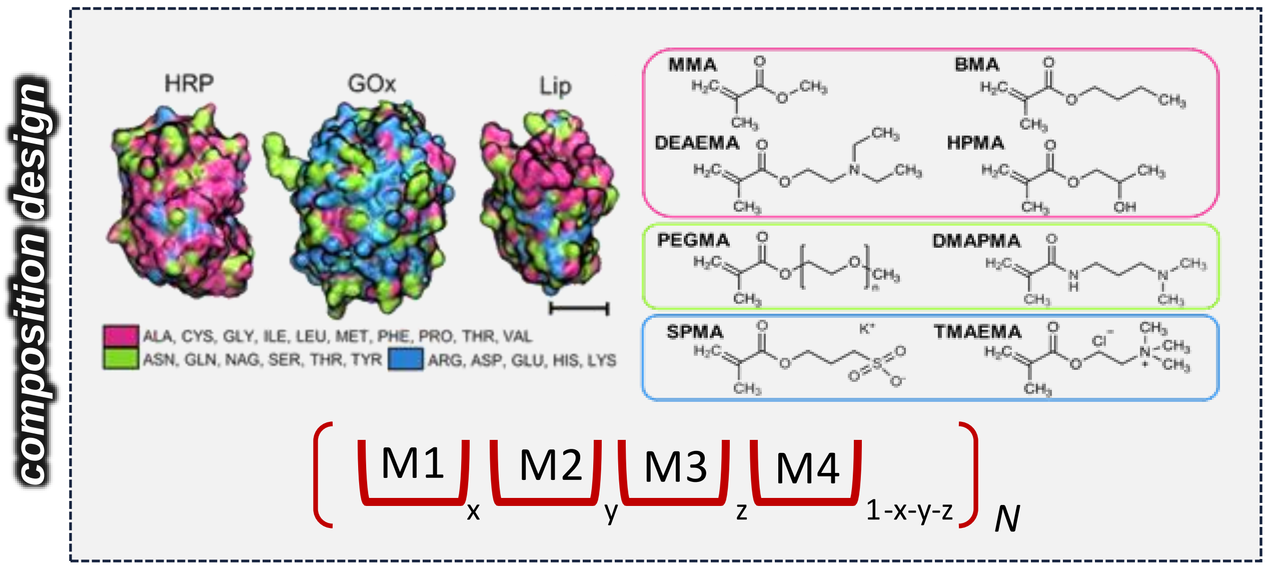


Classification in Chemical/Materials Optimization

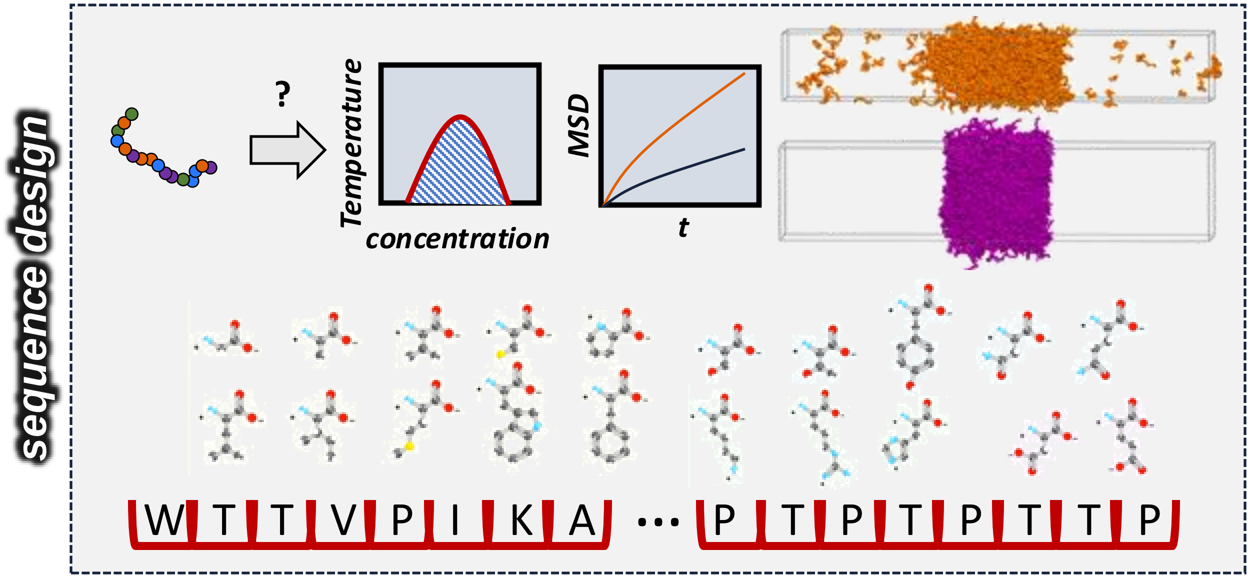




Tangible Motivating Examples

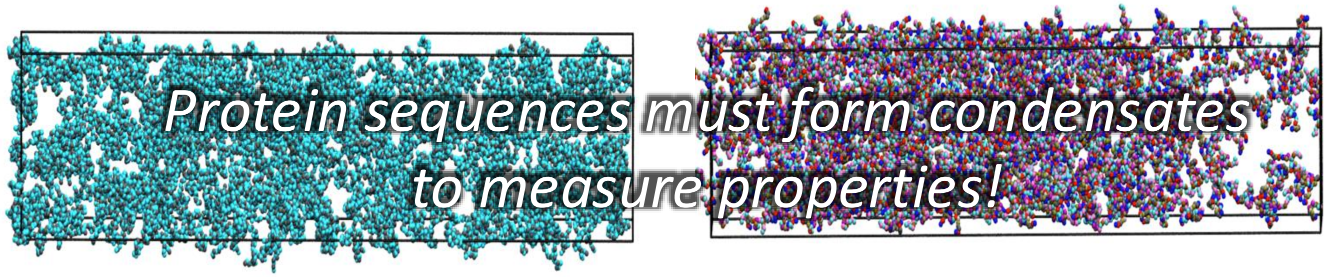

composition design
HRP
GOx
Lip
🟥 ALA, CYS, GLY, ILE, LEU, MET, PHE, PRO, THR, VAL
🟩 ASN, GLN, NAG, SER, THR, TYR
🟦 ARG, ASP, GLU, HIS, LYS
MMA
BMA
DEAEMA
HPMA
PEGMA
DMAPMA
SPMA
TMAEMA
M1 x M2 y M3 z M4 1-x-y-z N

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



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

Polymers must remain soluble for enzyme assay and characterization



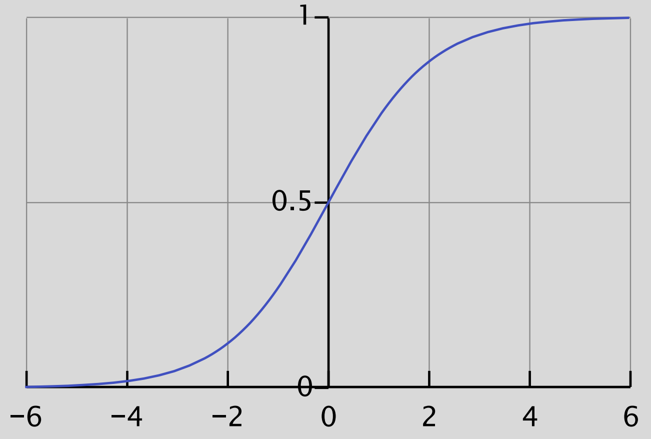
Protein sequences must form condensates to measure properties!

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

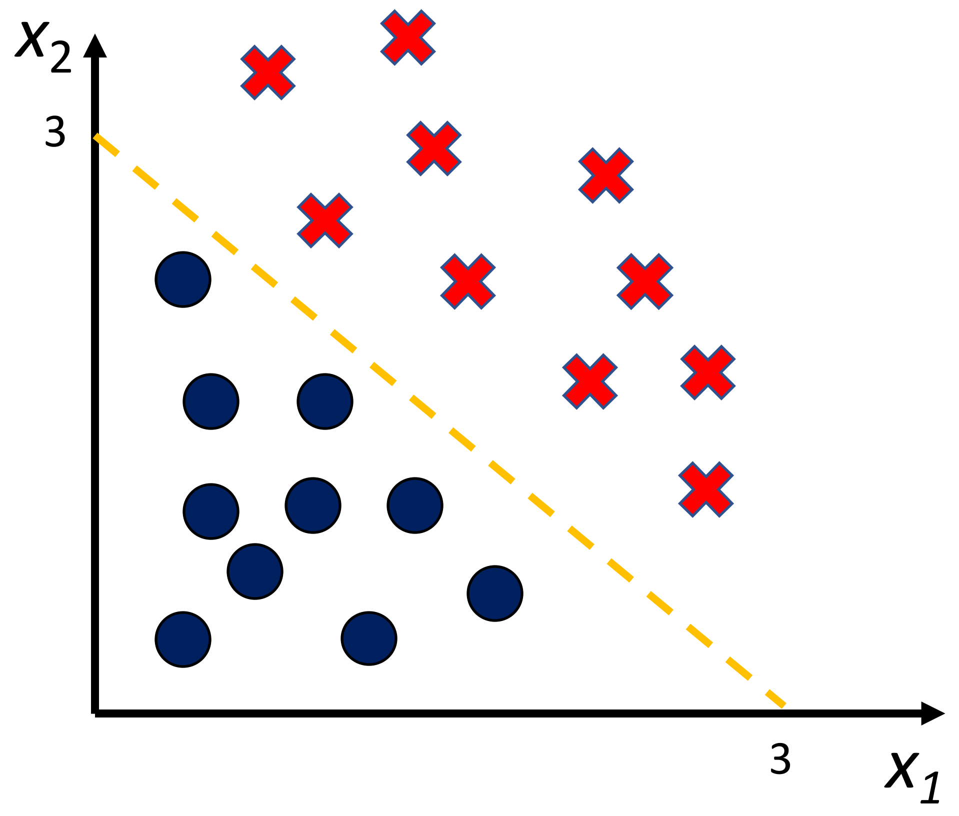
$$\ell = \log_b \frac{p}{1-p} = \boldsymbol{x}^T\boldsymbol{\theta}$$

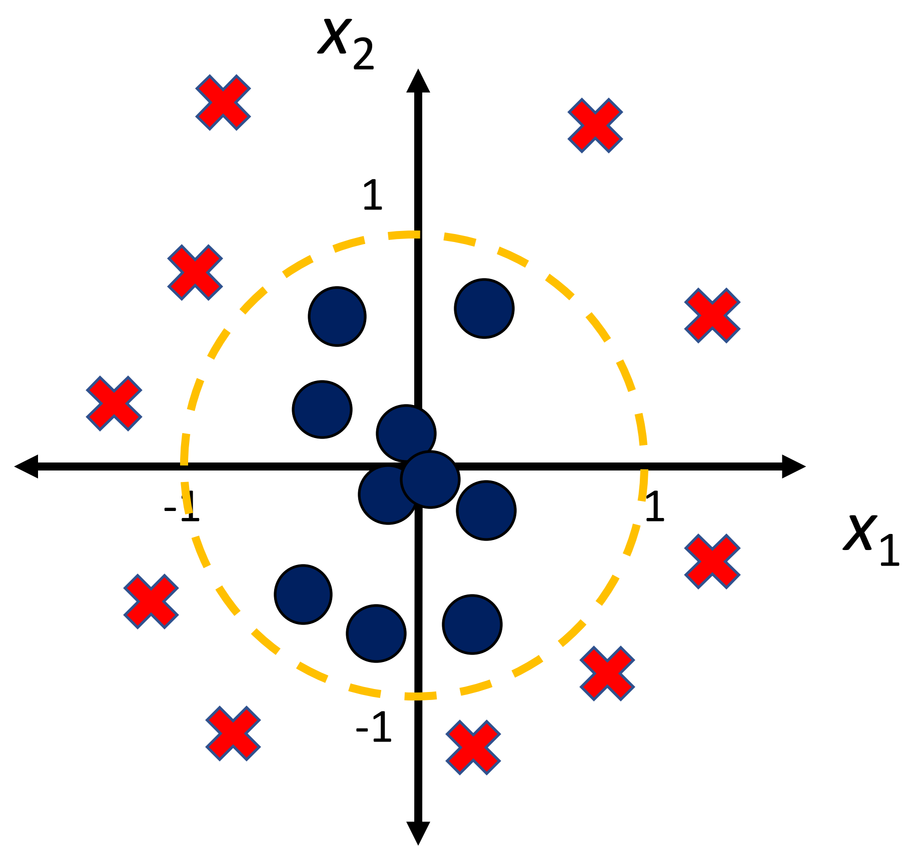
$$\Longrightarrow \quad p = \frac{1}{1 + b^{-\boldsymbol{x}^T\boldsymbol{\theta}}} \xrightarrow{b=e} \frac{1}{1 + e^{-\boldsymbol{x}^T\boldsymbol{\theta}}}$$

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

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

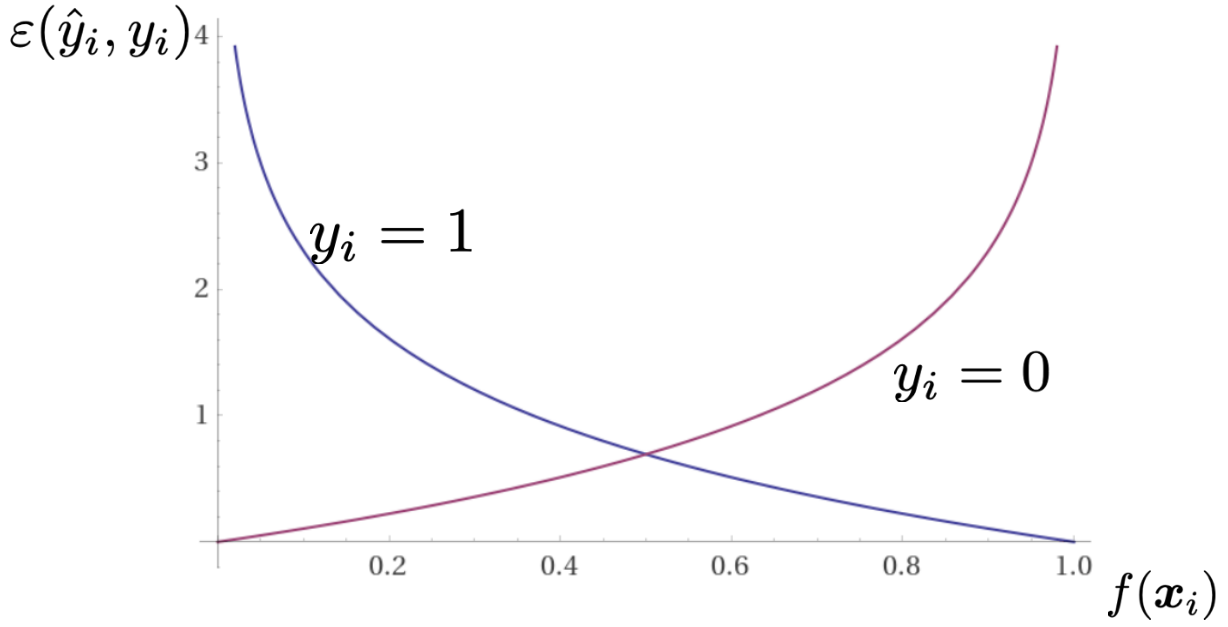
$$\varepsilon(\hat{y}_i, y_i) = \begin{cases} -\log\left[f(\boldsymbol{x}_i)\right], & \text{if } y_i = 1 \\ -\log\left[1 - f(\boldsymbol{x}_i)\right], & \text{if } y_i = 0 \end{cases}$$

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

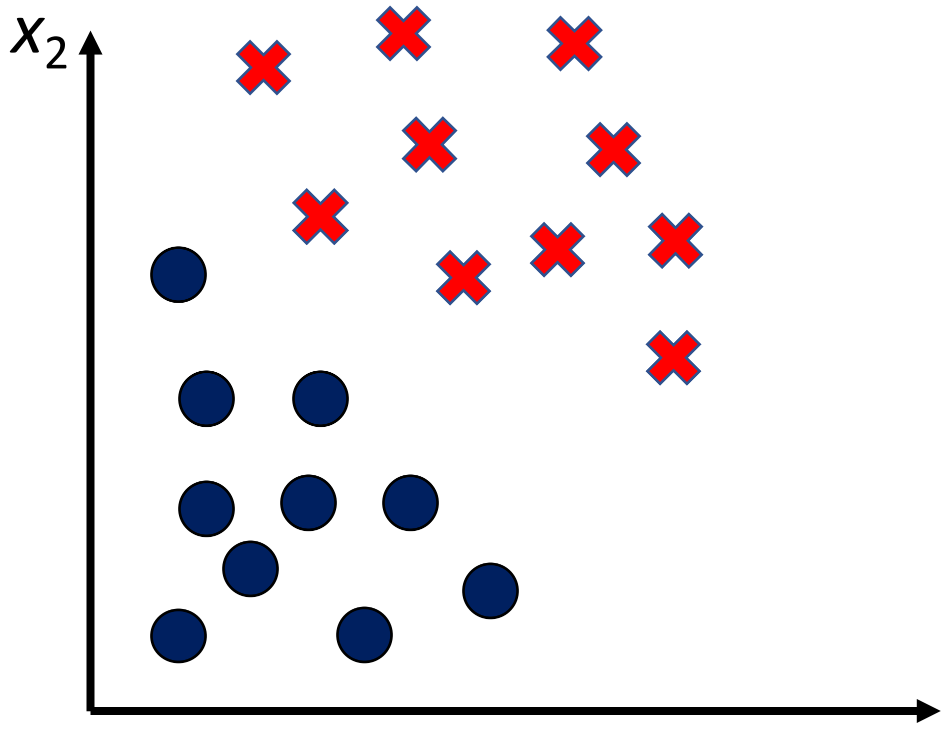
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma_i \left[\nabla_{\boldsymbol{\theta}} \mathcal{E}(\boldsymbol{\theta}) \right]$$

$$\vdots$$

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

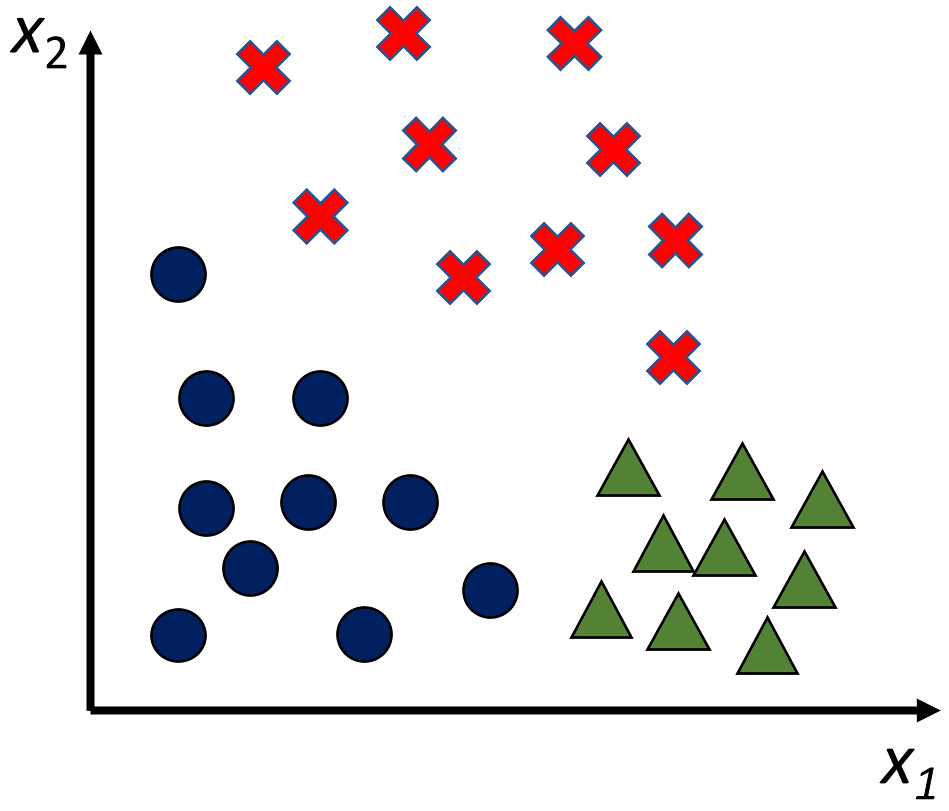
This is the same result as for linear regression!

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

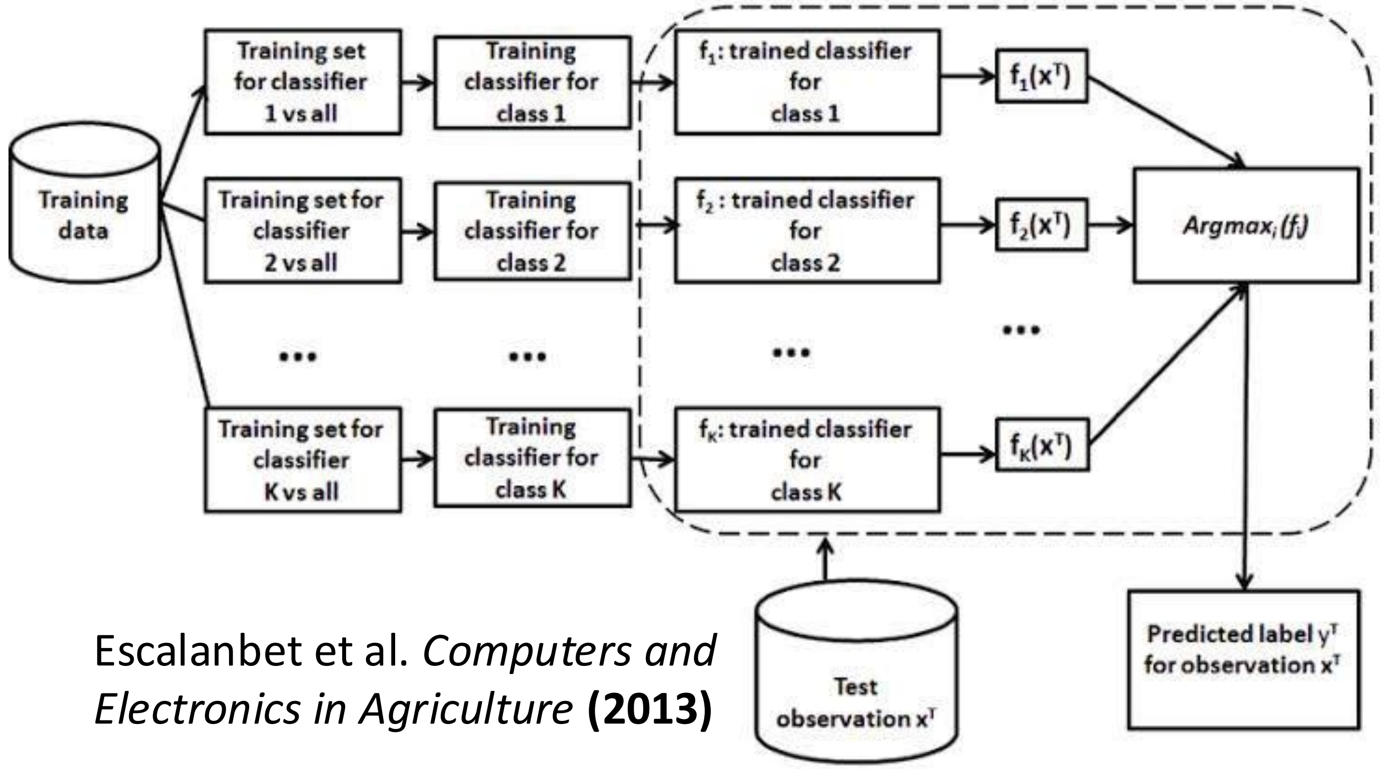
what do we do for multiple classes?

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

Consider our data to have N classes…

One vs. All (Rest)

- formulate *N* binary classifier models

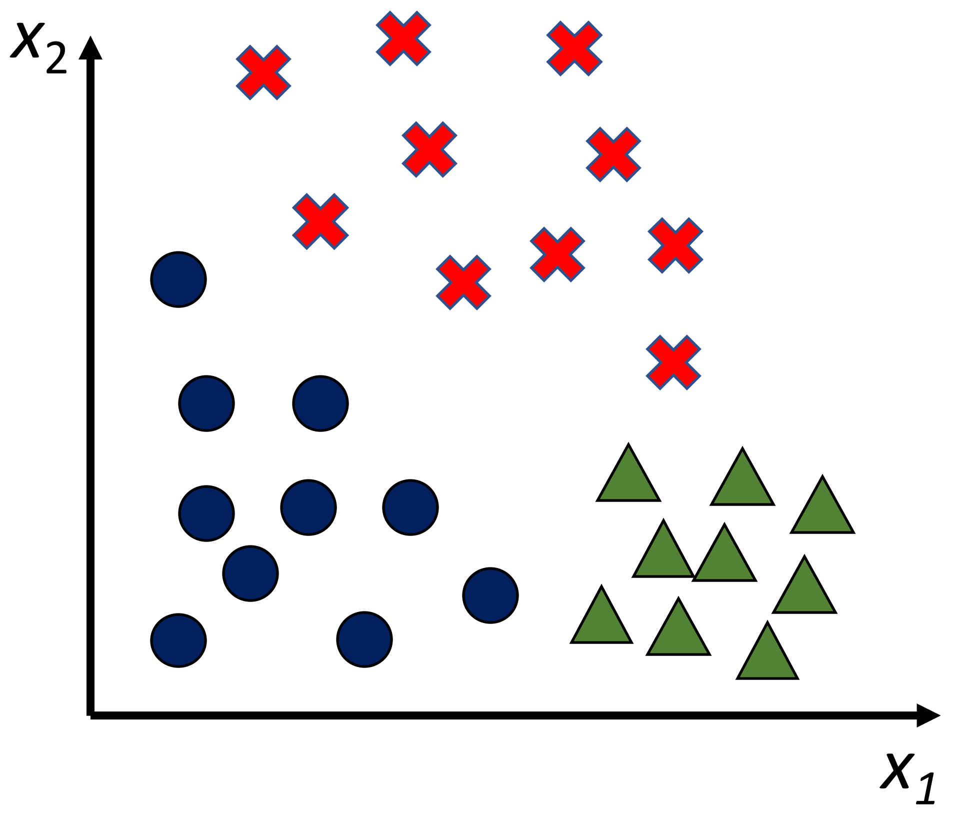


Training data
Training set for classifier 1 vs all
Training set for classifier 2 vs all
Training set for classifier K vs all
Training classifier for class 1
Training classifier for class 2
Training classifier for class K
f_1: trained classifier for class 1
f_2: trained classifier for class 2
f_K: trained classifier for class K
$f_1(x^T)$
$f_2(x^T)$
$f_K(x^T)$
$Argmax_i(f_i)$
Test observation x^T
Predicted label y^T for observation x^T

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

- pick the class that exhibits the highest score

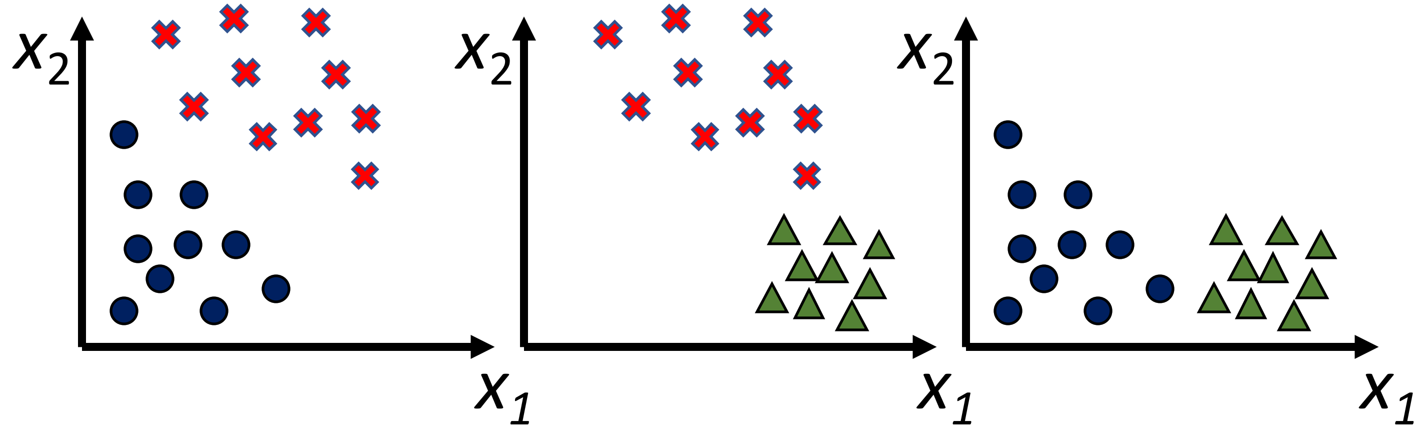
Application now to classification: One vs. One



Consider our data to have N classes…

<u>One vs. One</u>

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



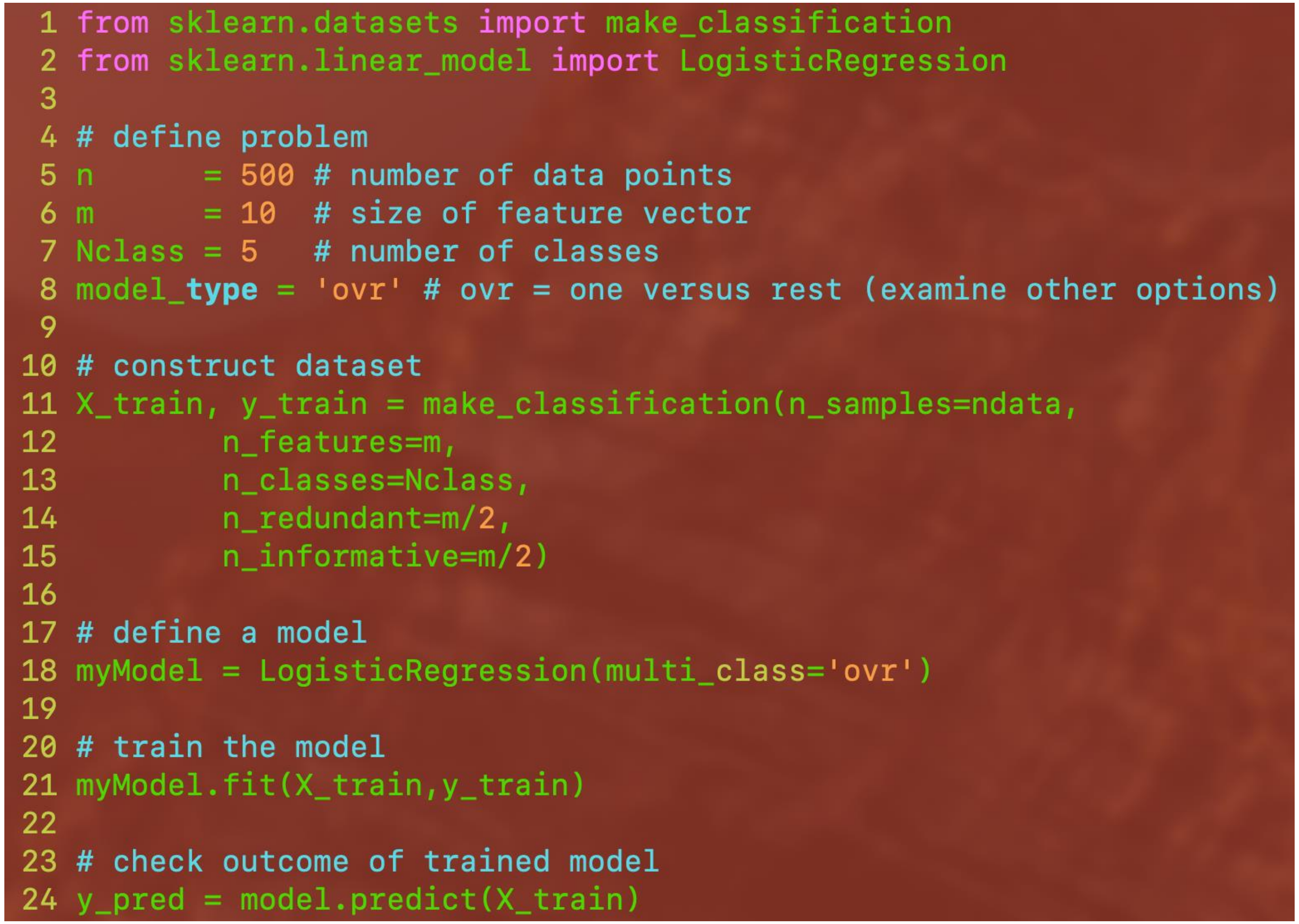
- pick the class that receives the most positive identifications

- **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

Logistic Regression Classification in scikit_learn



```python
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)
```

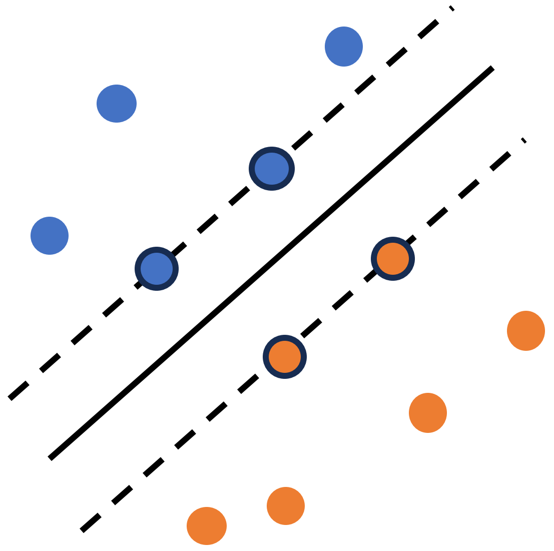
Support Vector Machine (SVM)

“Support-vector networks”, Corinna Cortes & Vladimir Vapnik, 1995

Four Key SVM Ideas:

1. Margin
2. Support Vectors
3. Kernel
4. Regularization parameter

Input: 2D array (feature matrix)
Output: 1D array (vector of classes)

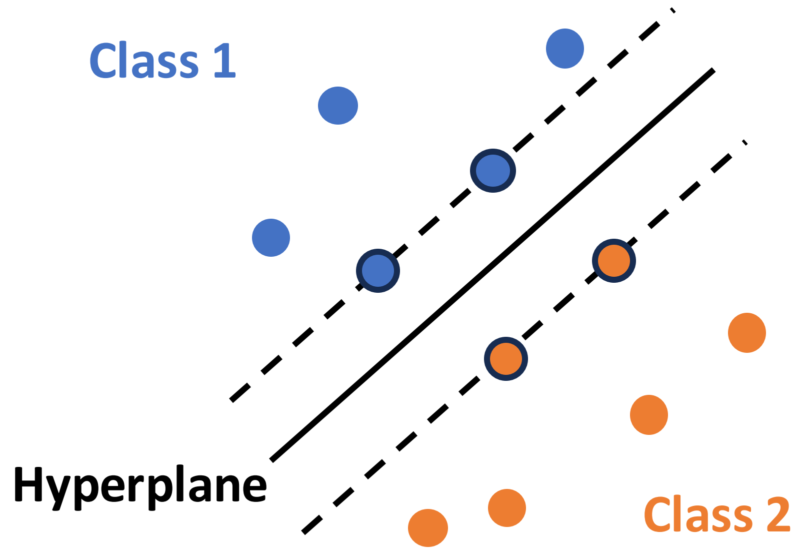


When to use SVM?

1. **Number of features > number of samples**: gene expression, process system monitoring.
2. **Small to medium-size data.**
3. **Features are all numerical.**

Linear Separator Learning

Definition: Linear separator learning is a supervised learning task in which the goal is to find a **hyperplane** (or **a linear decision boundary**) in a feature space that can separate **two classes** of data points.



The **decision function** is

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

- $f(x) > 0$ for all x belong to class 1 (e.g., positive).
- $f(x) < 0$ for all x belong to class 2 (e.g., negative).
- $f(x) = 0$ for all x on the hyperplane.

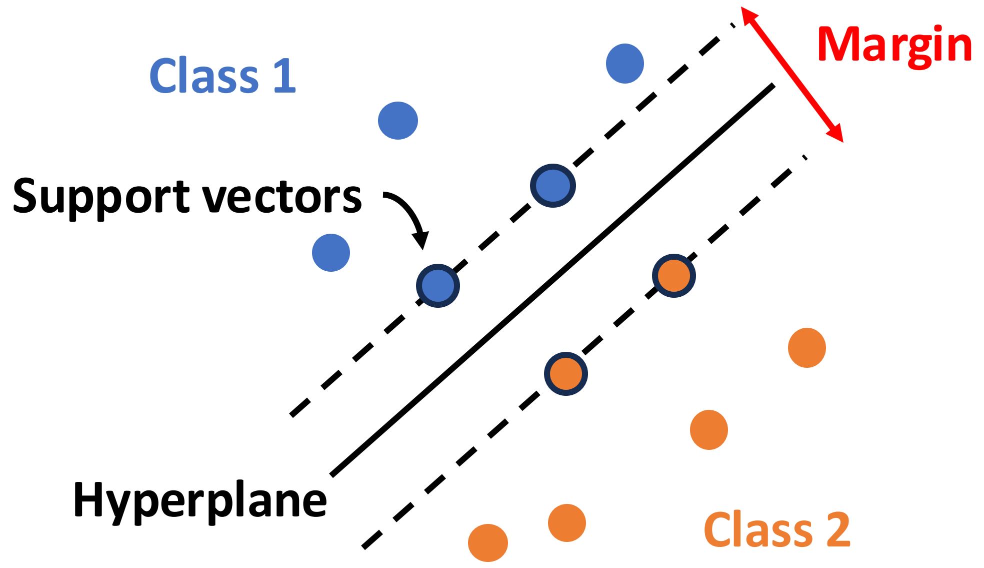
A data point (x, y) is classified **correctly** if

$$y \cdot (\mathbf{w}^T\mathbf{x} + b) > 0$$

Large Margin Classification

Definition: Large margin classification is a supervised learning method where the objective is to find a hyperplane that:

1. separates the classes of data, and
2. **maximizes the margin** between the hyperplane and the closest data points from either class.



Why?

Given a data set that is linearly separable, there are **infinitely many hyperplanes** that can separate the two classes.

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0, \ i = 1, 2, \cdots, m$$

$$\alpha\mathbf{w}, \alpha b \text{ for any } \alpha > 0$$

In SVM, we aim to find the hyperplane that maximizes the **margin**.

Definition: Margin is the distance between the hyperplane and the closest data points from either class.
Definition: Support vectors are the closest data points.

Large Margin Classification

What is the geometric interpretation of *w*?

$$\mathbf{w}^T(\mathbf{x}_2 - \mathbf{x}_1) = 0$$

w is **orthogonal** to any vector that lies on the hyperplane.

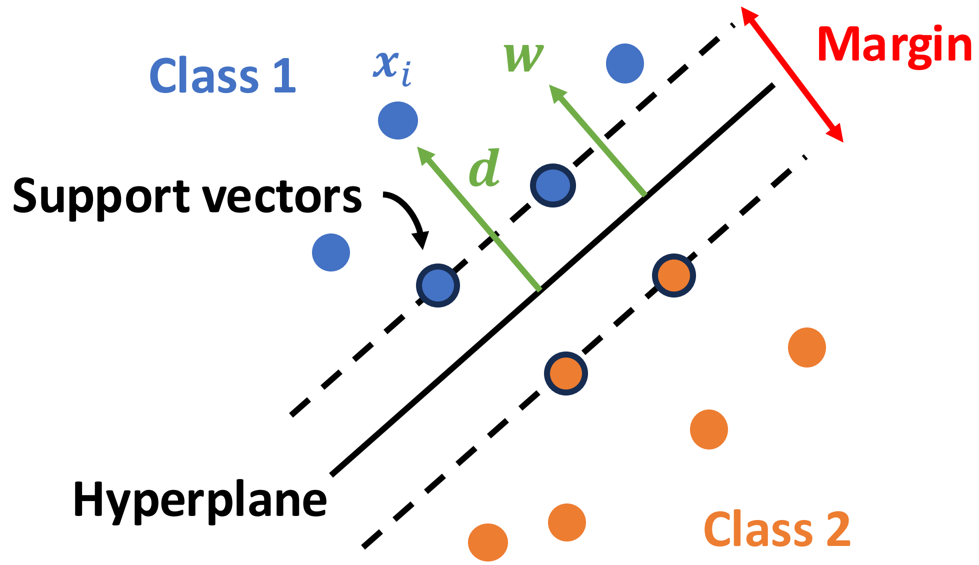
The distance ***d*** of a point $\boldsymbol{x}_i$ to the hyperplane is

$$\mathbf{w}^T(\mathbf{x}_i - \mathbf{d}) + b = 0$$

$$\mathbf{w}^T(\mathbf{x}_i - \alpha\mathbf{w}) + b = 0$$

$$\alpha = \frac{\mathbf{w}^T\mathbf{x}_i + b}{\mathbf{w}^T\mathbf{w}}$$

$$||\mathbf{d}||_2 = \sqrt{\alpha^2\mathbf{w}^T\mathbf{w}} = \frac{|\mathbf{w}^T\mathbf{x}_i + b|}{||\mathbf{w}||_2}$$



The hyperplane must lie right in the middle of the two classes, with an equal distance from the positive and negative support vectors.

Because the hyperplane is **scale invariant**

$$\text{Margin} = \frac{2}{||\mathbf{w}||_2}$$

Why?

The Hard-Margin SVM

Definition: Hard-margin SVM is a type of SVM used for binary classification when the data is linearly separable.

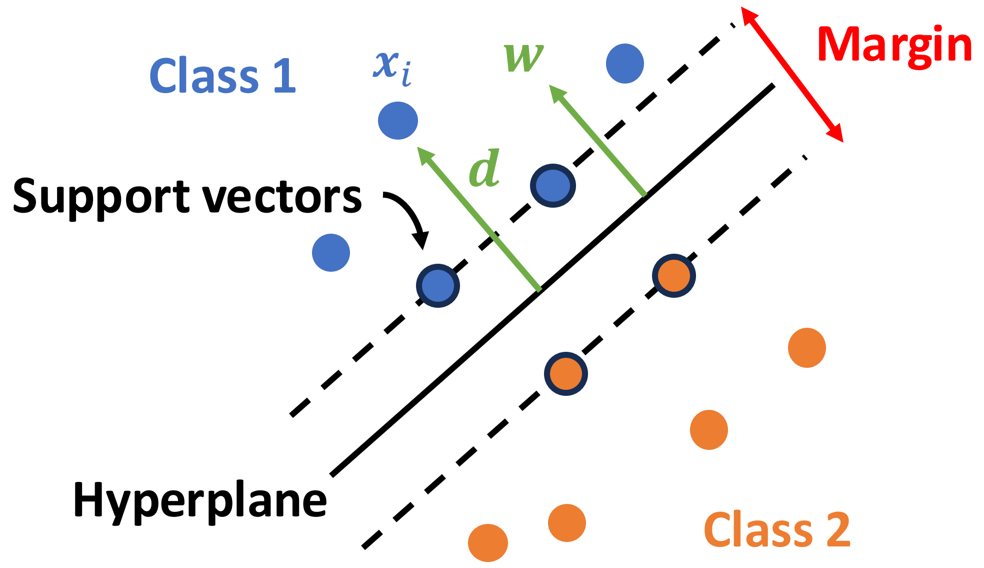
Optimization:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2$$

$$\text{subject to :}$$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \ \forall i = 1, 2, \cdots, m$$

- The constraint ensures that all points are correctly classified and lie on the correct side of the margin.
- This is a **convex quadratic programming problem**, which can be efficiently solved using standard optimization techniques.



What if the data points are not linearly separable?

The Soft-Margin SVM

Definition: Soft-margin SVM is a type of SVM that allows some points to violate the margin constraints by introducing **slack variables** that permit misclassification or margin violations, while still attempting to maximize the margin.

Optimization:

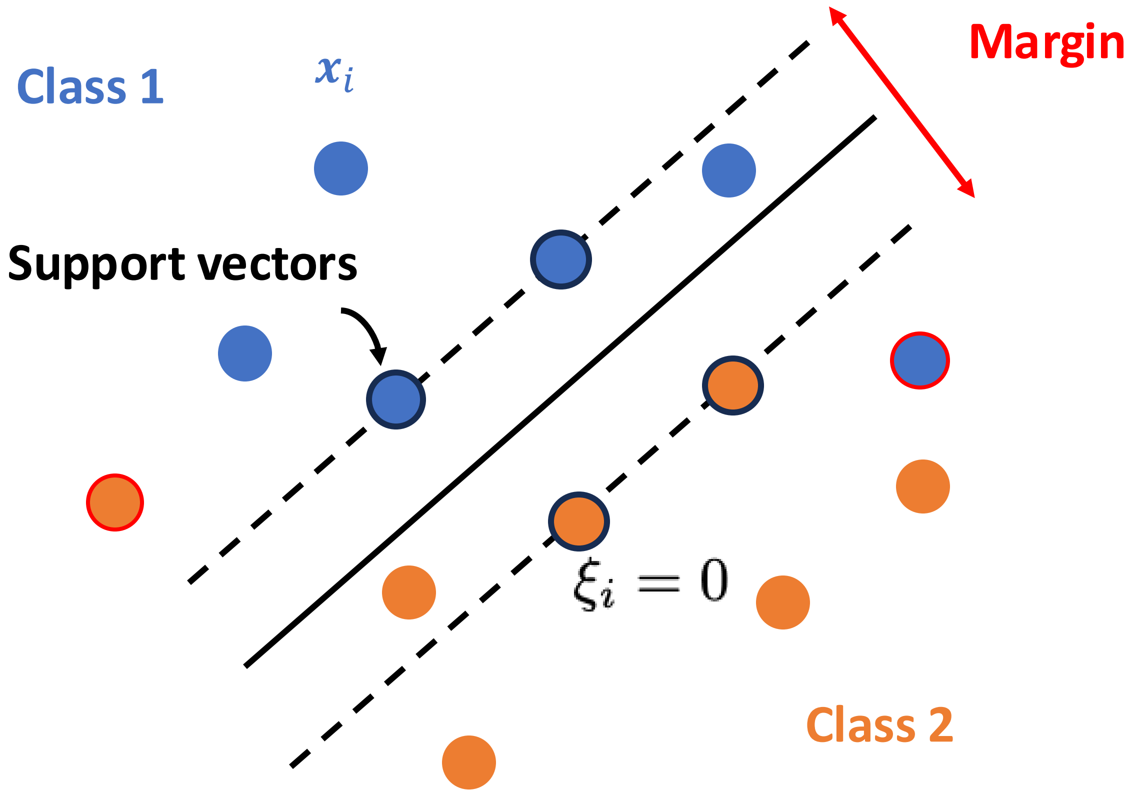
$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to :}$$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \; \forall i = 1, 2, \cdots, m$$

$$\xi_i \geq 0, \; \forall i = 1, 2, \cdots, m$$

- ξ_i: slack variable.
- C: regularization parameter.



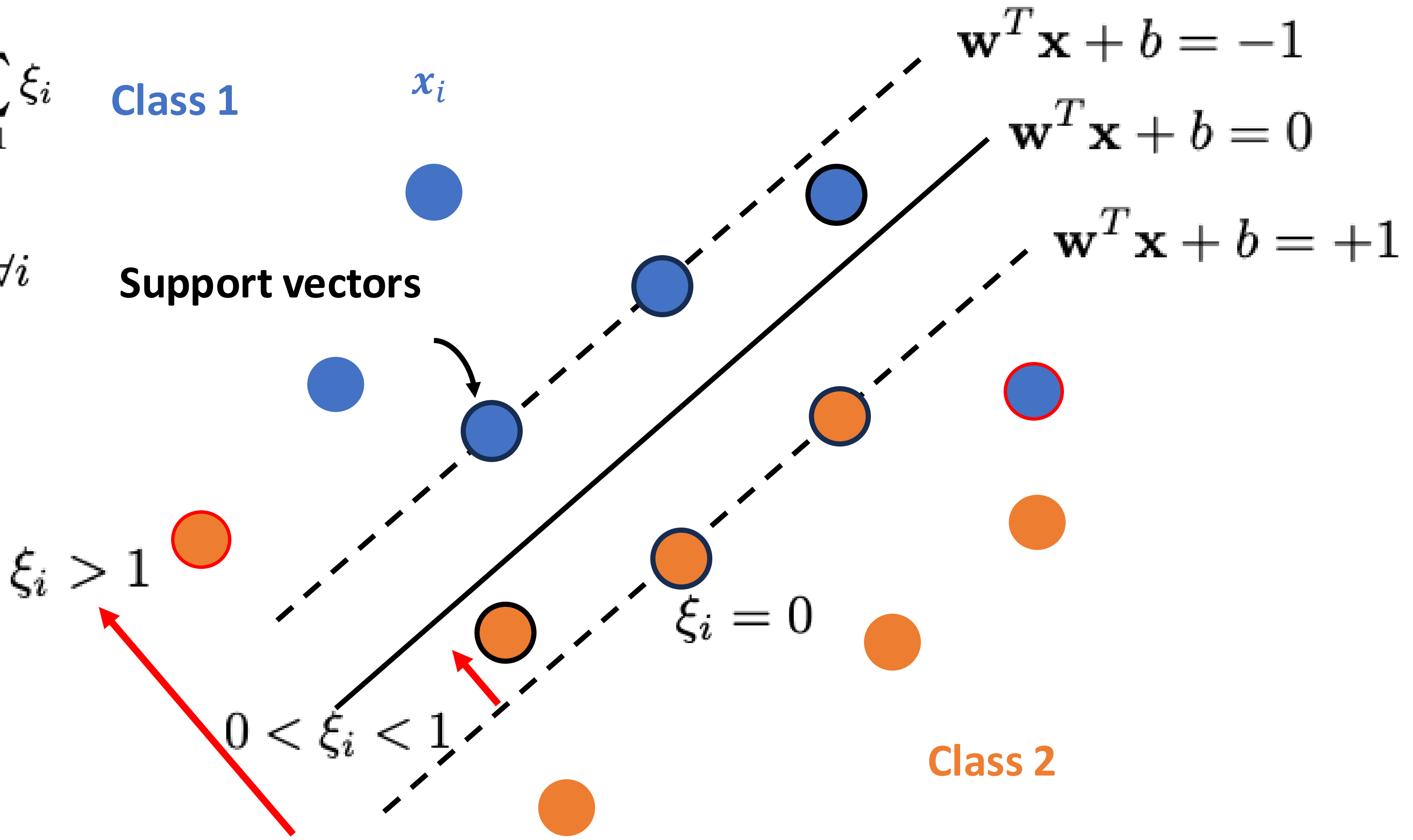
The Soft-Margin SVM

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2 + C\sum_{i=1}^{n}\xi_i$$

subject to :

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \ \forall i$$

$$\xi_i \geq 0, \ \forall i$$


Class 1
x_i
$\mathbf{w}^T\mathbf{x} + b = -1$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = +1$
Support vectors
$\xi_i > 1$
$\xi_i = 0$
$0 < \xi_i < 1$
Class 2

- Smaller C: Allows more margin violations but tries to maximize the margin.
- Larger C: A smaller margin but fewer violations

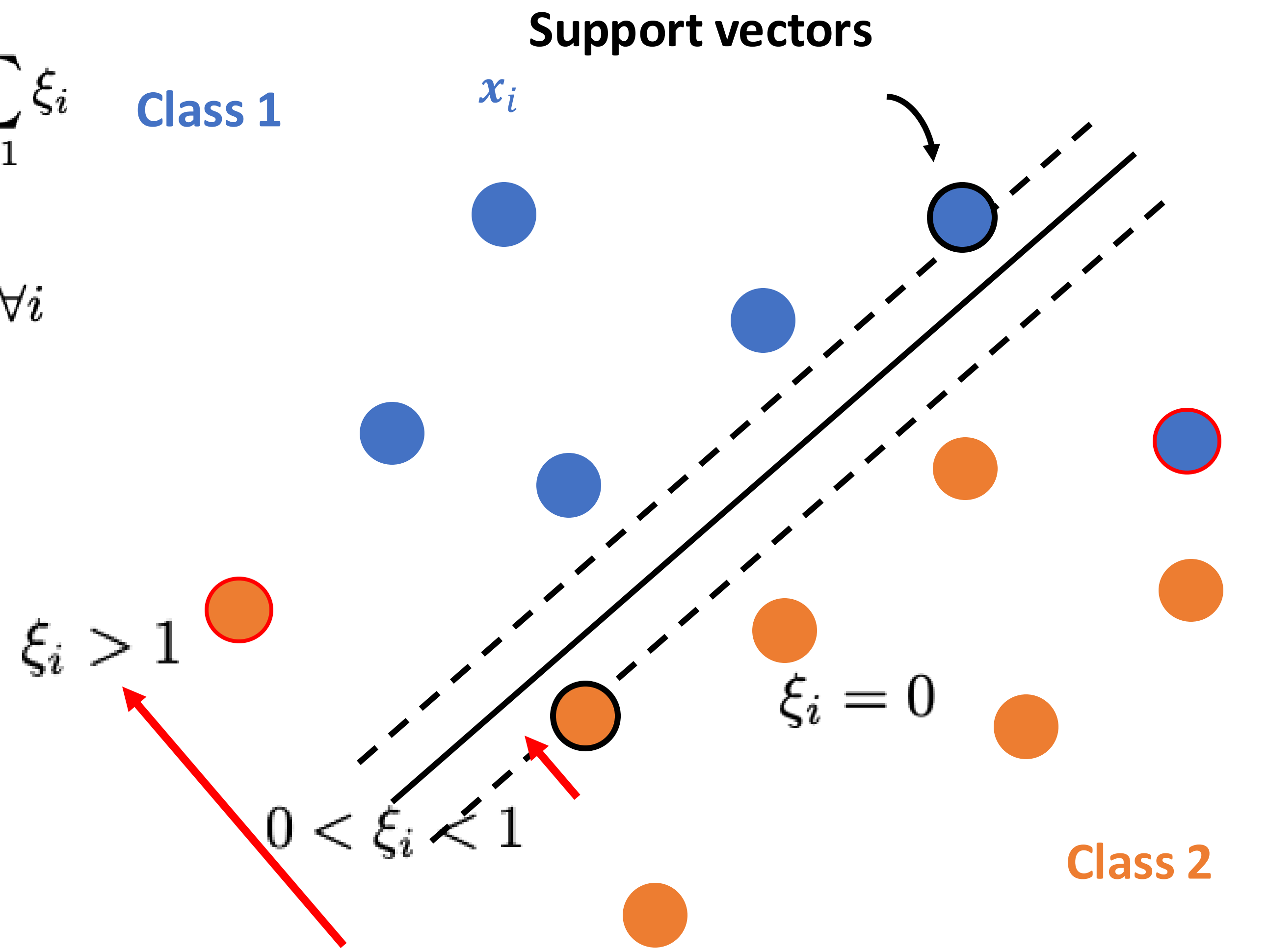
The Soft-Margin SVM

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2 + C\sum_{i=1}^{n}\xi_i$$

subject to :

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \forall i$$

$$\xi_i \geq 0,\ \forall i$$



- Smaller C: Allows more margin violations but tries to maximize the margin.
- Larger C: A smaller margin but fewer violations

Non-Linear SVM

Definition: Non-linear SVM extends linear SVM by using a kernel function to transform data into a higher-dimensional space, making it separable.

Non-linear SVM **hyperplane** is

$$\mathbf{w}^T \phi(\mathbf{x}) + b = 0$$

$$\phi : \mathbb{R}^n \rightarrow \mathbb{R}^d,\ d > n$$

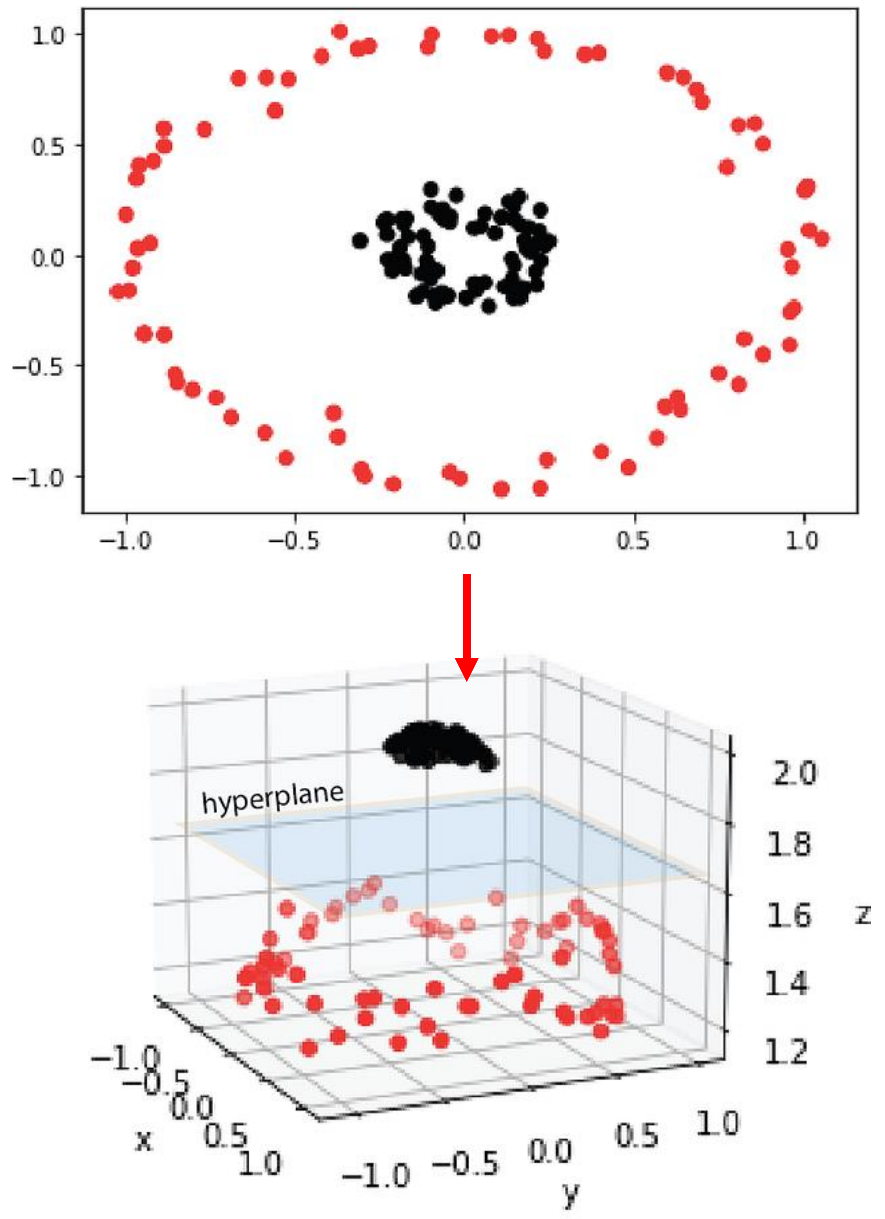
- $\phi(\boldsymbol{x})$ is a non-linear mapping function.

- Radial basis function (RBF):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2\right)$$

- Polynomial:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^d$$



Kernel Trick

The **kernel trick** allows us to avoid explicitly computing $\phi(\boldsymbol{x})$ while still achieving the benefits of mapping the data into a higher-dimensional space.

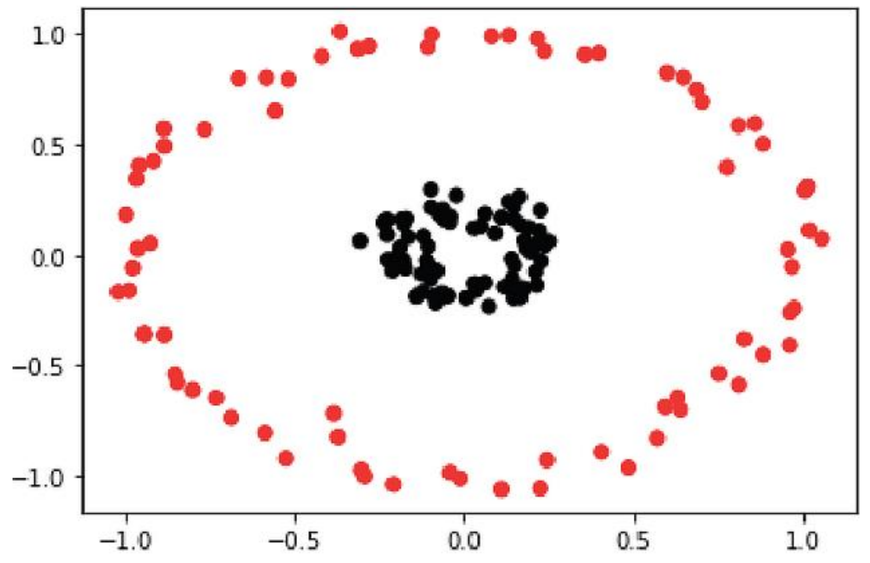
A kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ computes the **dot product** of two transformed data points $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$:

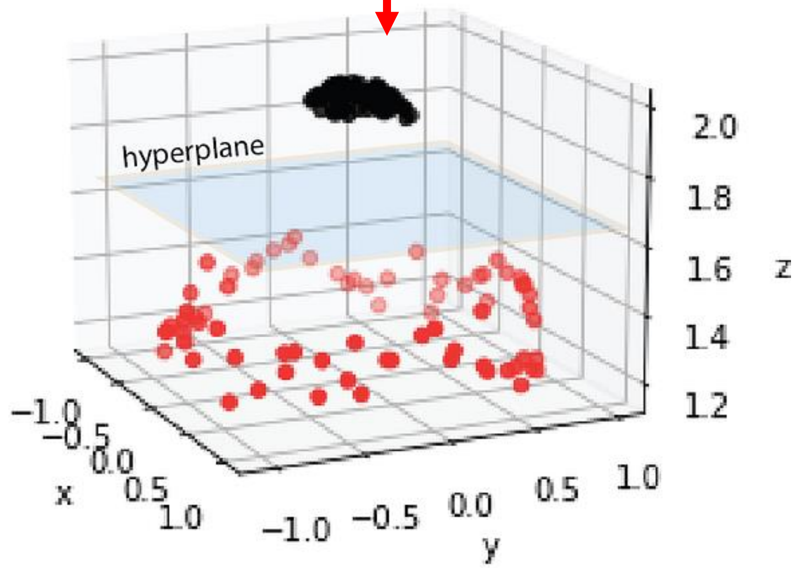
$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2\right)$$

$\downarrow$ Taylor expansion

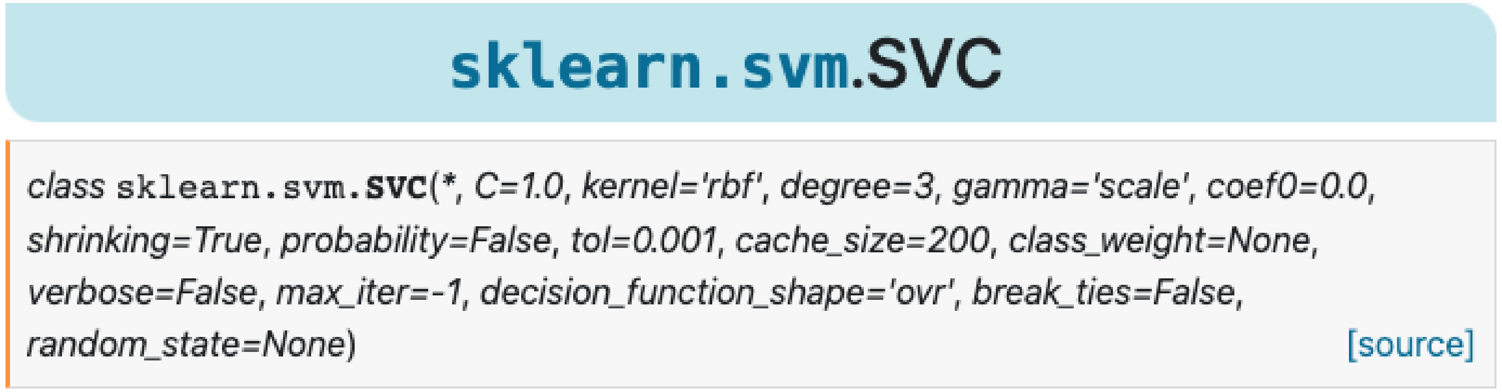
$$k(\mathbf{x}_i, \mathbf{x}_j) = C\left(\sum_{n=0}^{\infty} \frac{(\mathbf{x}_i^T \mathbf{x}_j)^n}{n!}\right)$$

- Infinite dimension dot product for RBF kernel.
- The kernel measures the **similarity or distance** between points, helping the SVM model understand how they relate to each other in the new space.



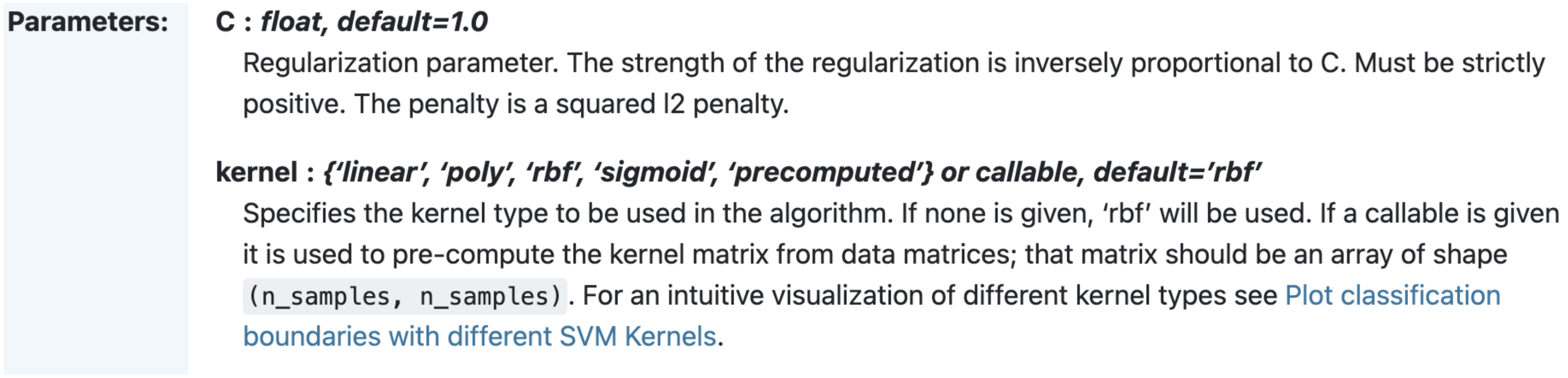


SVM Classification in scikit_learn



sklearn.svm.SVC

class `sklearn.svm.SVC`(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]



Parameters:

C : ***float, default=1.0***

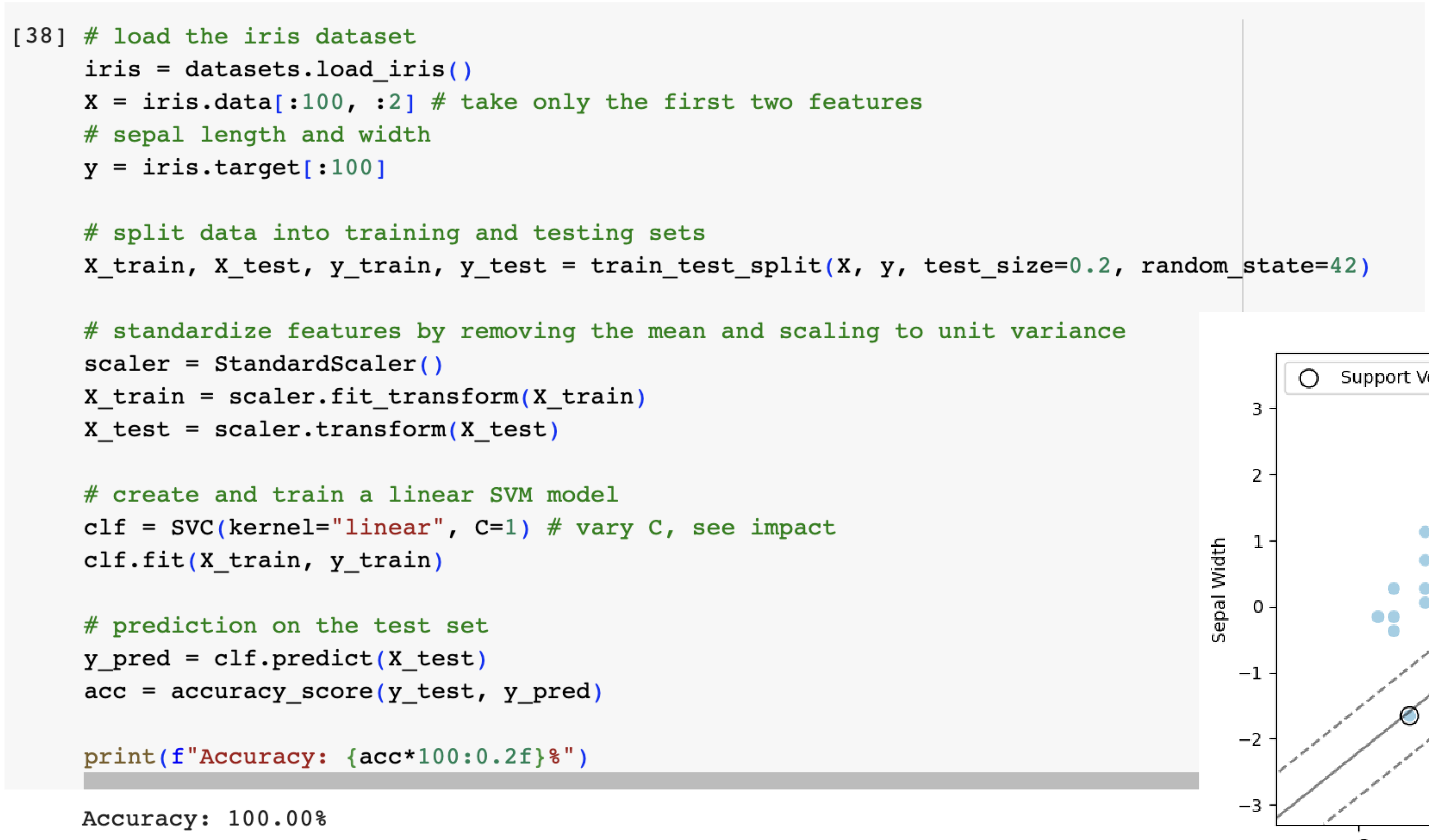
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : ***{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'***

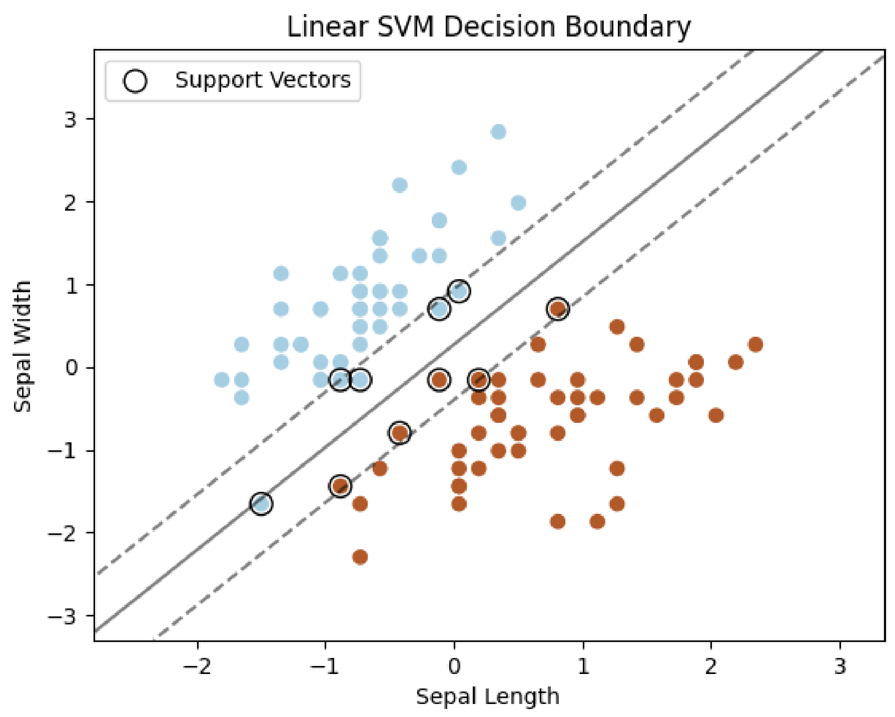
Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`. For an intuitive visualization of different kernel types see Plot classification boundaries with different SVM Kernels.

SVM Classification in scikit_learn

1. Simple linear SVC and decision boundary

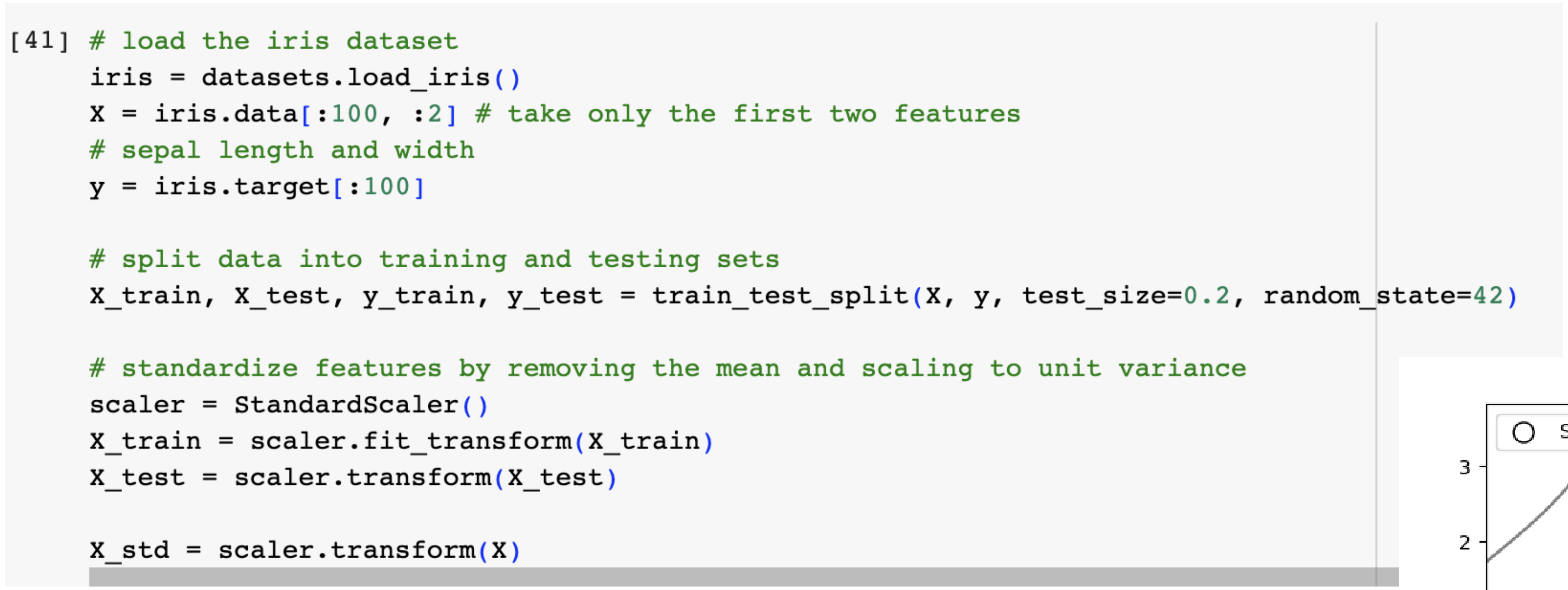


```
[38] # load the iris dataset
     iris = datasets.load_iris()
     X = iris.data[:100, :2] # take only the first two features
     # sepal length and width
     y = iris.target[:100]

     # split data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # standardize features by removing the mean and scaling to unit variance
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)

     # create and train a linear SVM model
     clf = SVC(kernel="linear", C=1) # vary C, see impact
     clf.fit(X_train, y_train)

     # prediction on the test set
     y_pred = clf.predict(X_test)
     acc = accuracy_score(y_test, y_pred)

     print(f"Accuracy: {acc*100:0.2f}%")

     Accuracy: 100.00%
```

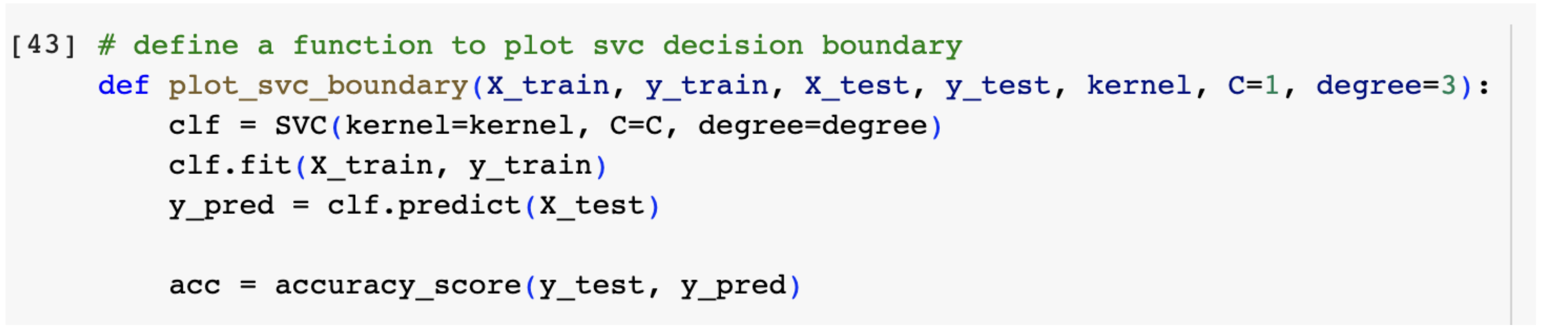


SVM Classification in scikit_learn

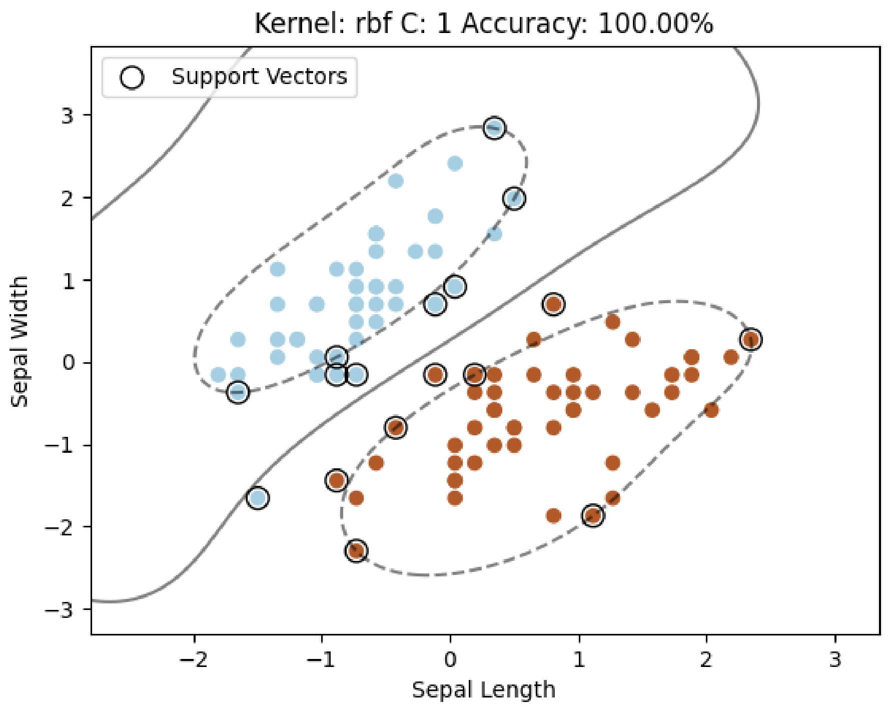
2. SVC with non-linear kernels



```
[41] # load the iris dataset
     iris = datasets.load_iris()
     X = iris.data[:100, :2] # take only the first two features
     # sepal length and width
     y = iris.target[:100]

     # split data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # standardize features by removing the mean and scaling to unit variance
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)

     X_std = scaler.transform(X)
```

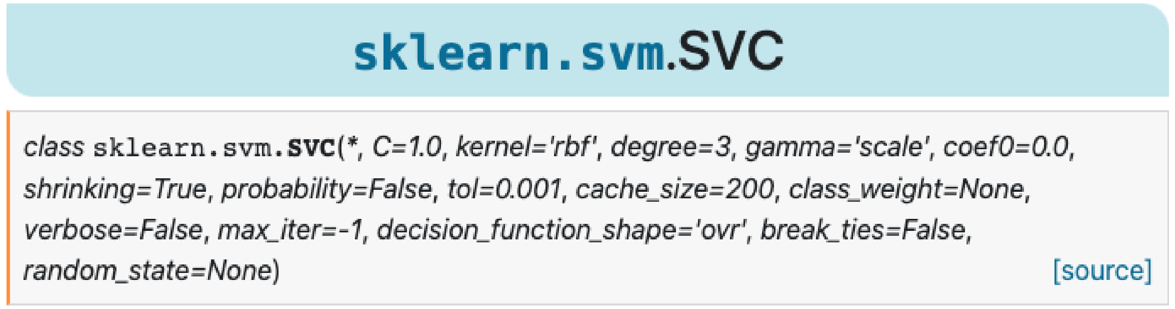


```
[43] # define a function to plot svc decision boundary
     def plot_svc_boundary(X_train, y_train, X_test, y_test, kernel, C=1, degree=3):
         clf = SVC(kernel=kernel, C=C, degree=degree)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)

         acc = accuracy_score(y_test, y_pred)
```



Hyperparameter Tuning

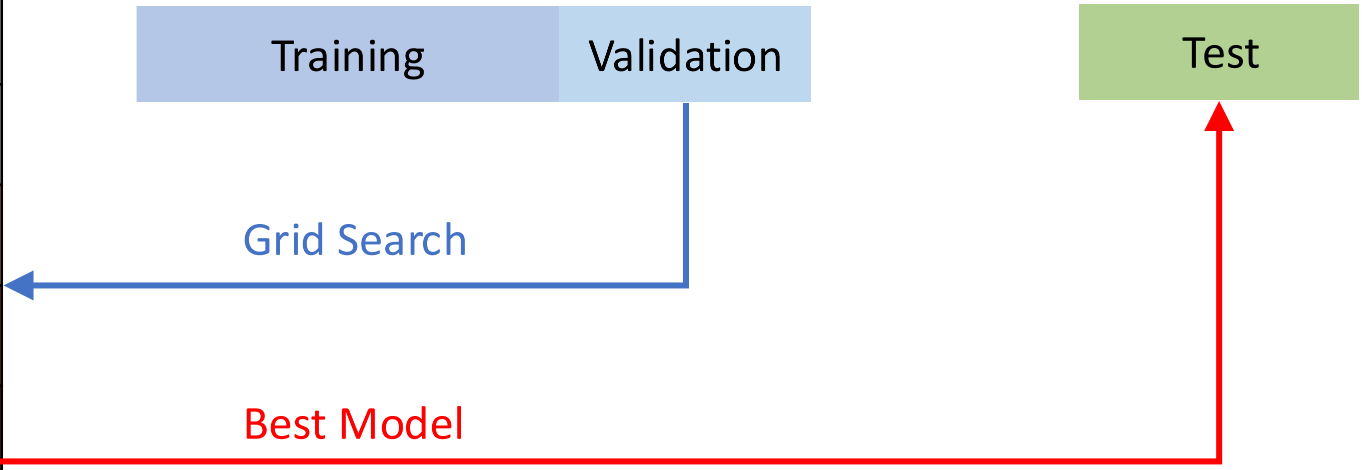
<u>Definition</u>: Grid search is an **exhaustive** search algorithm used to find the optimal hyperparameters for a given model by evaluating all possible combinations of a predefined set of hyperparameter values.



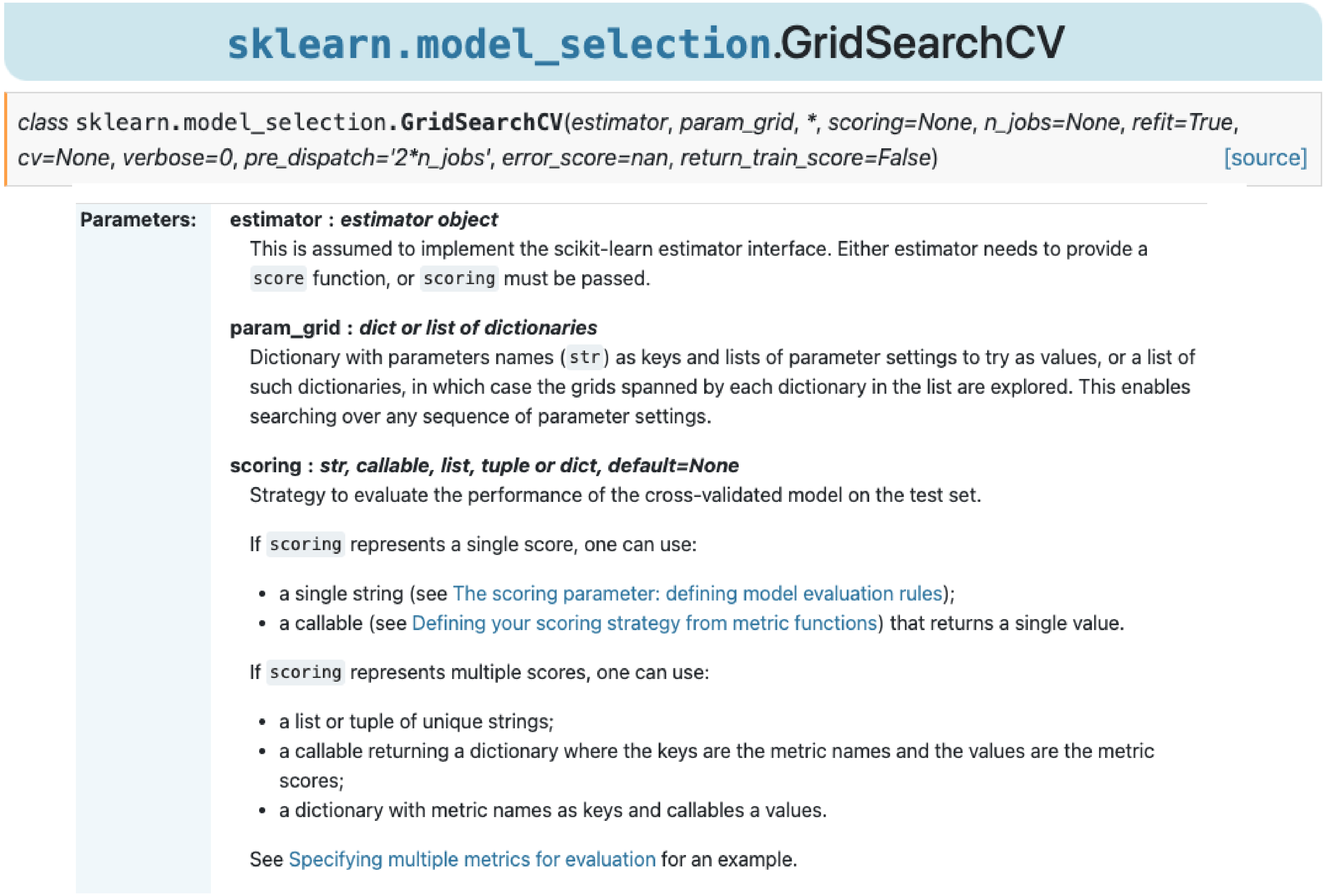
sklearn.svm.SVC

class `sklearn.svm.`**`SVC`**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

Validation Accuracy		C			
		0.1	1	10	100
kernel	linear	0.5	0.6	0.7	0.6
	rbf	0.6	0.7	0.8	0.7
	sigmoid	0.7	0.8	**<u>0.9</u>**	0.8
	poly	0.6	0.7	0.8	0.7



Grid Search in scikit_learn



sklearn.model_selection.GridSearchCV

class `sklearn.model_selection.`**`GridSearchCV`**(*estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False*) [source]

Parameters:

estimator : *estimator object*

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : *dict or list of dictionaries*

Dictionary with parameters names (`str`) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

scoring : *str, callable, list, tuple or dict, default=None*

Strategy to evaluate the performance of the cross-validated model on the test set.

If `scoring` represents a single score, one can use:

- a single string (see The scoring parameter: defining model evaluation rules);
- a callable (see Defining your scoring strategy from metric functions) that returns a single value.

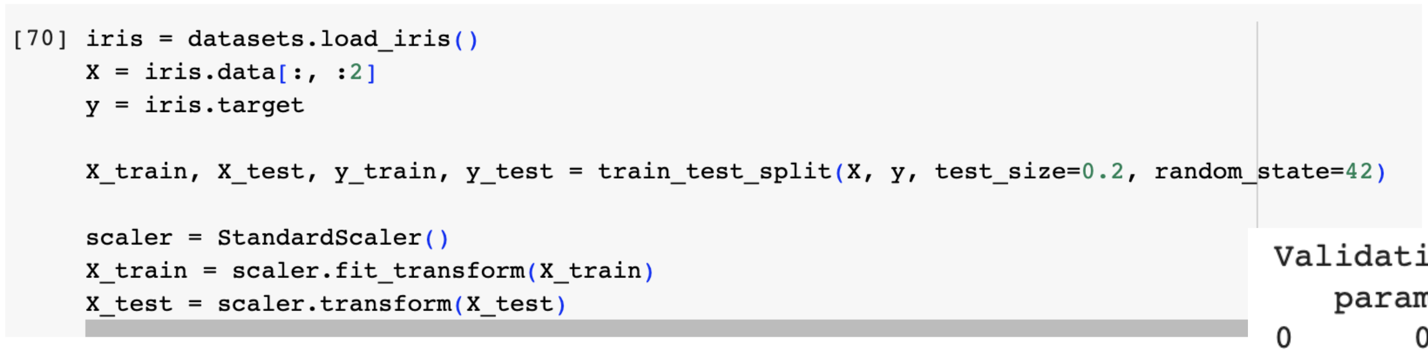
If `scoring` represents multiple scores, one can use:

- a list or tuple of unique strings;
- a callable returning a dictionary where the keys are the metric names and the values are the metric scores;
- a dictionary with metric names as keys and callables a values.

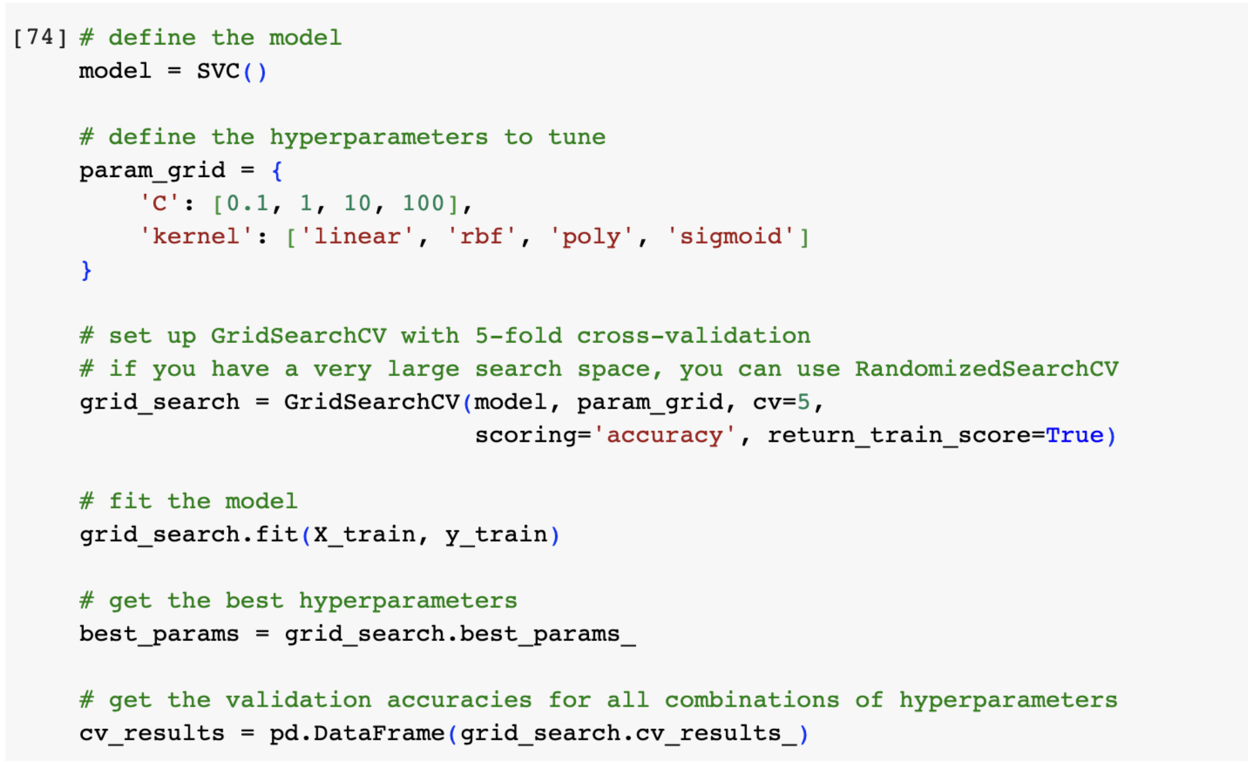
See Specifying multiple metrics for evaluation for an example.

Grid Search in scikit_learn

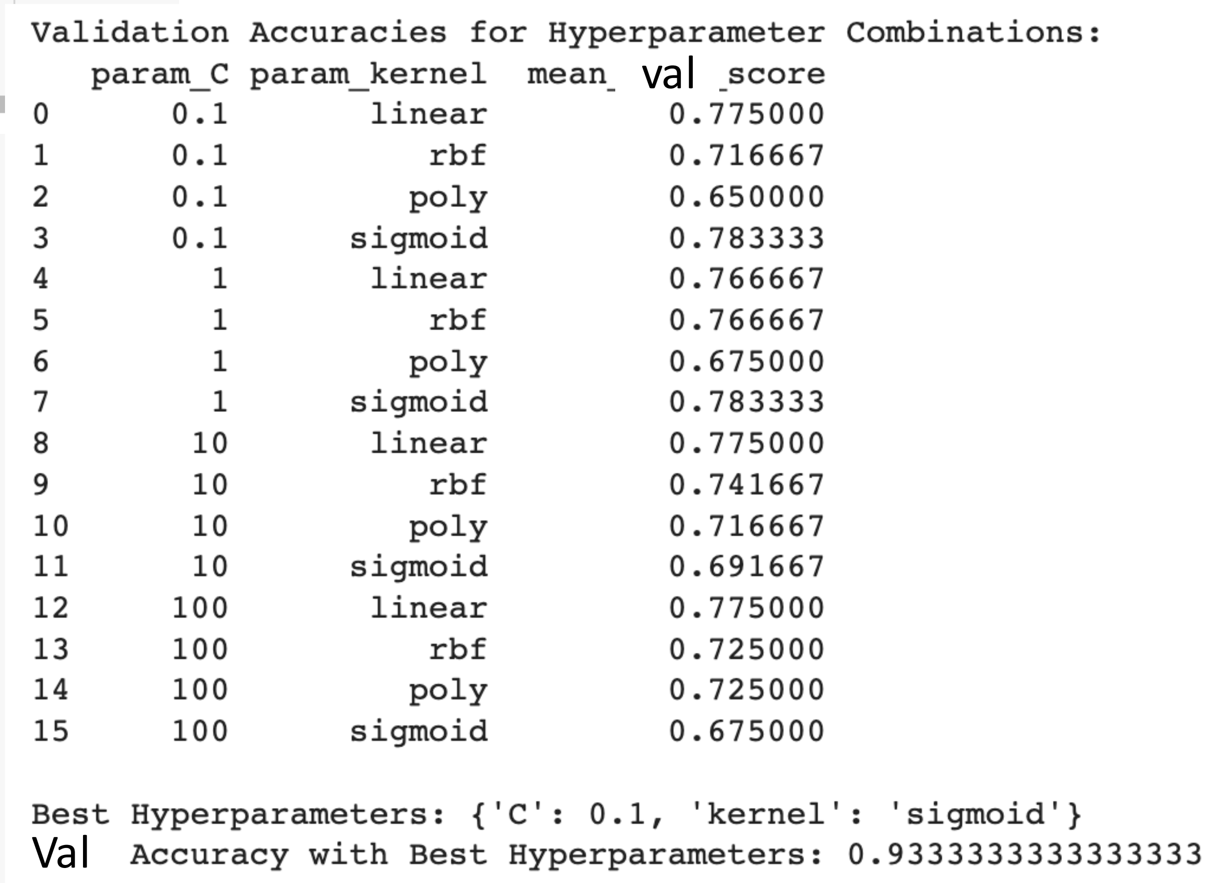
4. SVC hyperparameter tuning



```
[70] iris = datasets.load_iris()
     X = iris.data[:, :2]
     y = iris.target

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```



```
[74] # define the model
     model = SVC()

     # define the hyperparameters to tune
     param_grid = {
         'C': [0.1, 1, 10, 100],
         'kernel': ['linear', 'rbf', 'poly', 'sigmoid']
     }

     # set up GridSearchCV with 5-fold cross-validation
     # if you have a very large search space, you can use RandomizedSearchCV
     grid_search = GridSearchCV(model, param_grid, cv=5,
                                scoring='accuracy', return_train_score=True)

     # fit the model
     grid_search.fit(X_train, y_train)

     # get the best hyperparameters
     best_params = grid_search.best_params_

     # get the validation accuracies for all combinations of hyperparameters
     cv_results = pd.DataFrame(grid_search.cv_results_)
```



```
Validation Accuracies for Hyperparameter Combinations:
   param_C param_kernel  mean_ val _score
0      0.1       linear         0.775000
1      0.1          rbf         0.716667
2      0.1         poly         0.650000
3      0.1      sigmoid         0.783333
4        1       linear         0.766667
5        1          rbf         0.766667
6        1         poly         0.675000
7        1      sigmoid         0.783333
8       10       linear         0.775000
9       10          rbf         0.741667
10      10         poly         0.716667
11      10      sigmoid         0.691667
12     100       linear         0.775000
13     100          rbf         0.725000
14     100         poly         0.725000
15     100      sigmoid         0.675000

Best Hyperparameters: {'C': 0.1, 'kernel': 'sigmoid'}
Val  Accuracy with Best Hyperparameters: 0.9333333333333333
```