SUMMARY SESSION

WEEK – 07 (PART-B)

By:
Jimmi Kumar Bharti

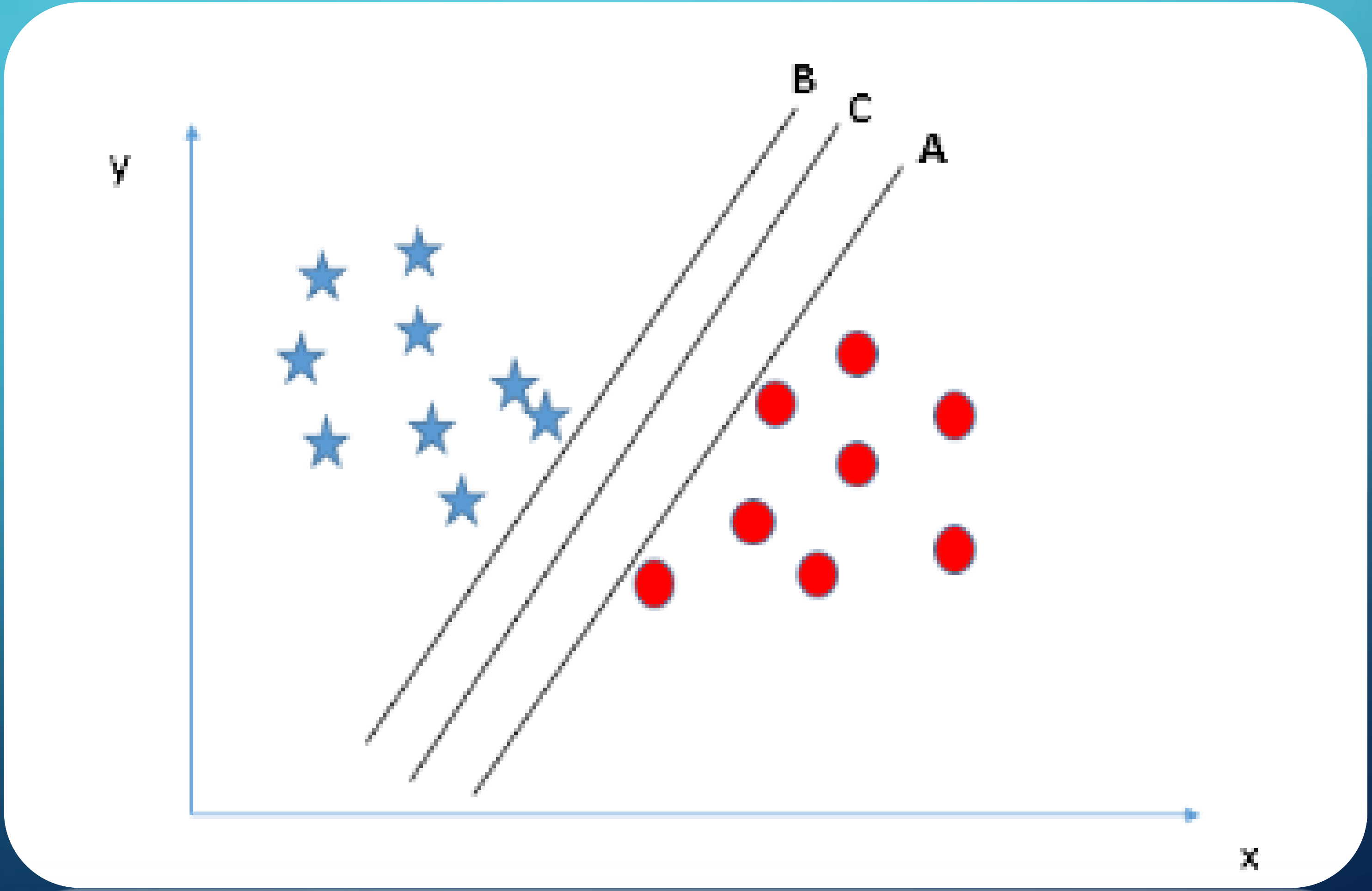
- SVM is a supervised machine learning algorithm that can be used for both classification and regression problems.
- We have focused on classification aspect of SVM in our course.
- SVM works in both binary and multiclass classification set ups.

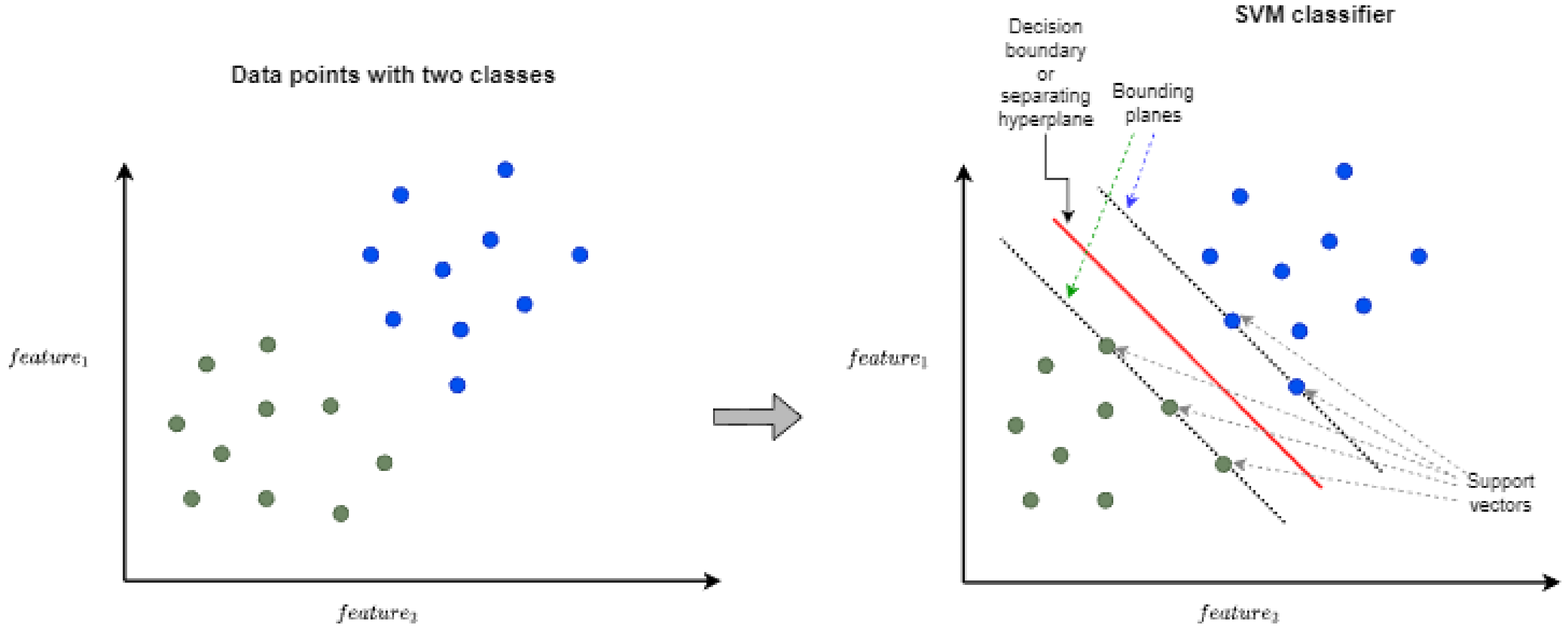
How an SVM works

- A simple linear SVM classifier works by making a straight line between two classes. That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category. This means there can be an infinite number of lines to choose from.
- What makes the linear SVM algorithm better than some of the other algorithms, like k-nearest neighbors, is that it chooses the best line to classify your data points. It chooses the line that separates the data and is the furthest away from the closest data points as possible.

How SVM works:

Increase the distance of decision boundary to classes (or support vectors).
Maximize the number of points that are correctly classified in the training set.





It selects the hyperplane that maximizes the distance to the closest data points from both classes.

In other words, it is the hyperplane with maximum margin between two classes.

Bounding planes

The bounding planes are defined as follows:

The bounding plane on the side of the positive class has the following equation:

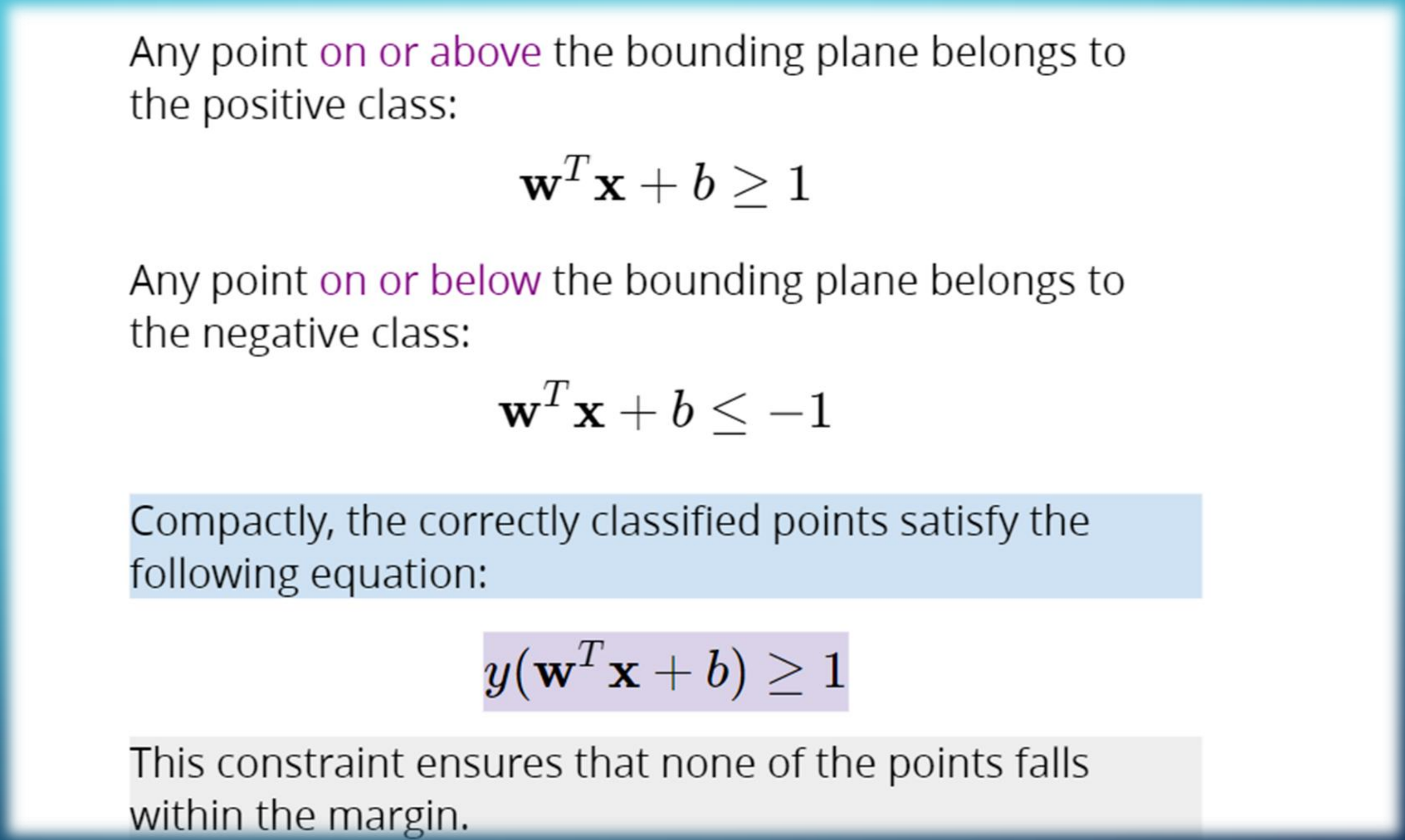
$$\mathbf{w}^T\mathbf{x} + b = 1$$

The bounding plane on the side of the negative class has the following equation:

$$\mathbf{w}^T\mathbf{x} + b = -1$$

We can write this in one equation as follows using the label of an example.

$$y(\mathbf{w}^T\mathbf{x} + b) = 1$$



Any point on or above the bounding plane belongs to the positive class:

$$\mathbf{w}^T\mathbf{x} + b \geq 1$$

Any point on or below the bounding plane belongs to the negative class:

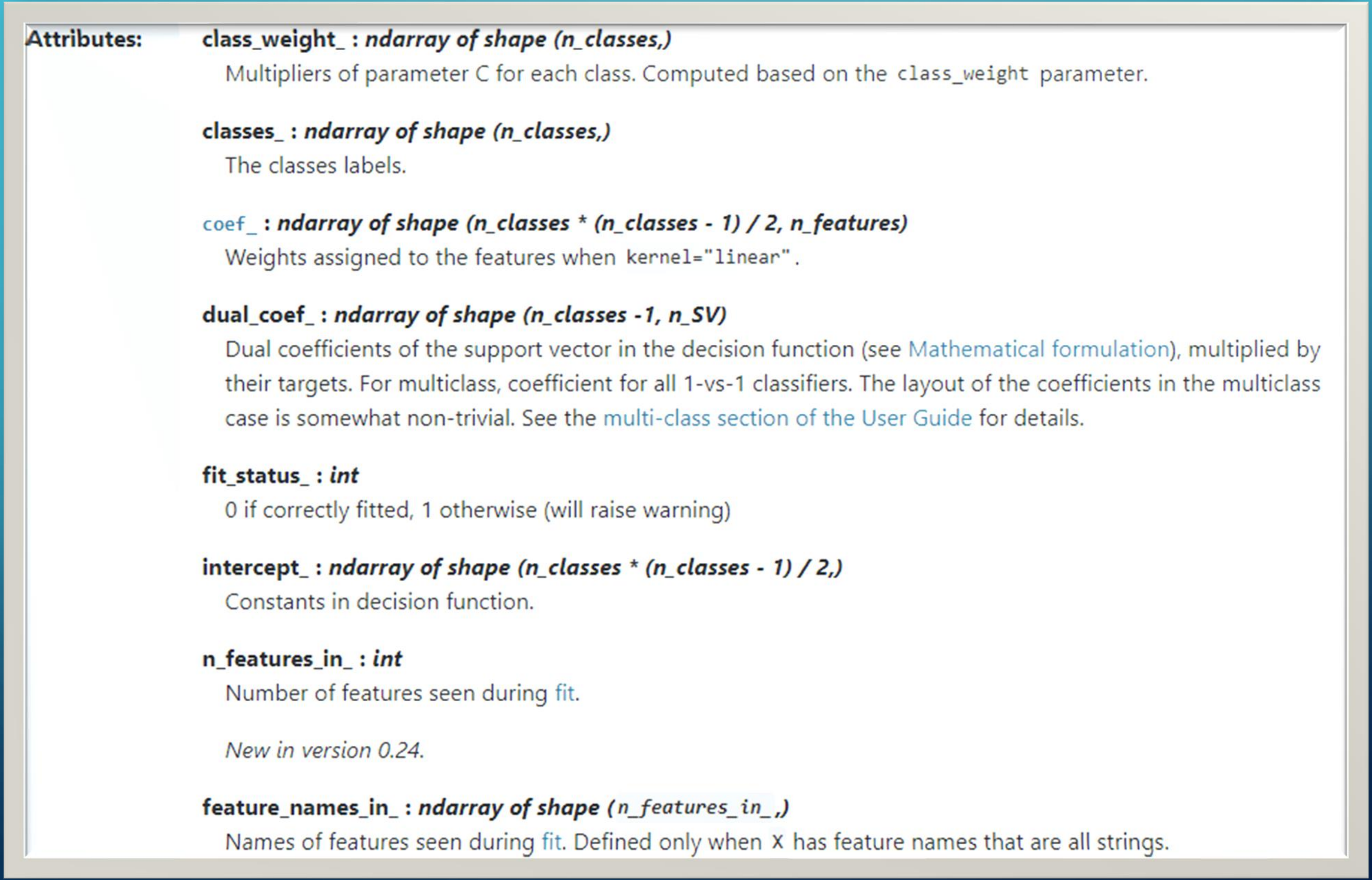
$$\mathbf{w}^T\mathbf{x} + b \leq -1$$

Compactly, the correctly classified points satisfy the following equation:

$$y(\mathbf{w}^T\mathbf{x} + b) \geq 1$$

This constraint ensures that none of the points falls within the margin.

Some attributes of SVM Classifier:-



Attributes:

class_weight_ : *ndarray of shape (n_classes,)*

Multipliers of parameter C for each class. Computed based on the `class_weight` parameter.

classes_ : *ndarray of shape (n_classes,)*

The classes labels.

`coef_` **: *ndarray of shape (n_classes * (n_classes - 1) / 2, n_features)***

Weights assigned to the features when `kernel="linear"`.

dual_coef_ : *ndarray of shape (n_classes -1, n_SV)*

Dual coefficients of the support vector in the decision function (see Mathematical formulation), multiplied by their targets. For multiclass, coefficient for all 1-vs-1 classifiers. The layout of the coefficients in the multiclass case is somewhat non-trivial. See the multi-class section of the User Guide for details.

fit_status_ : *int*

0 if correctly fitted, 1 otherwise (will raise warning)

intercept_ : *ndarray of shape (n_classes * (n_classes - 1) / 2,)*

Constants in decision function.

n_features_in_ : *int*

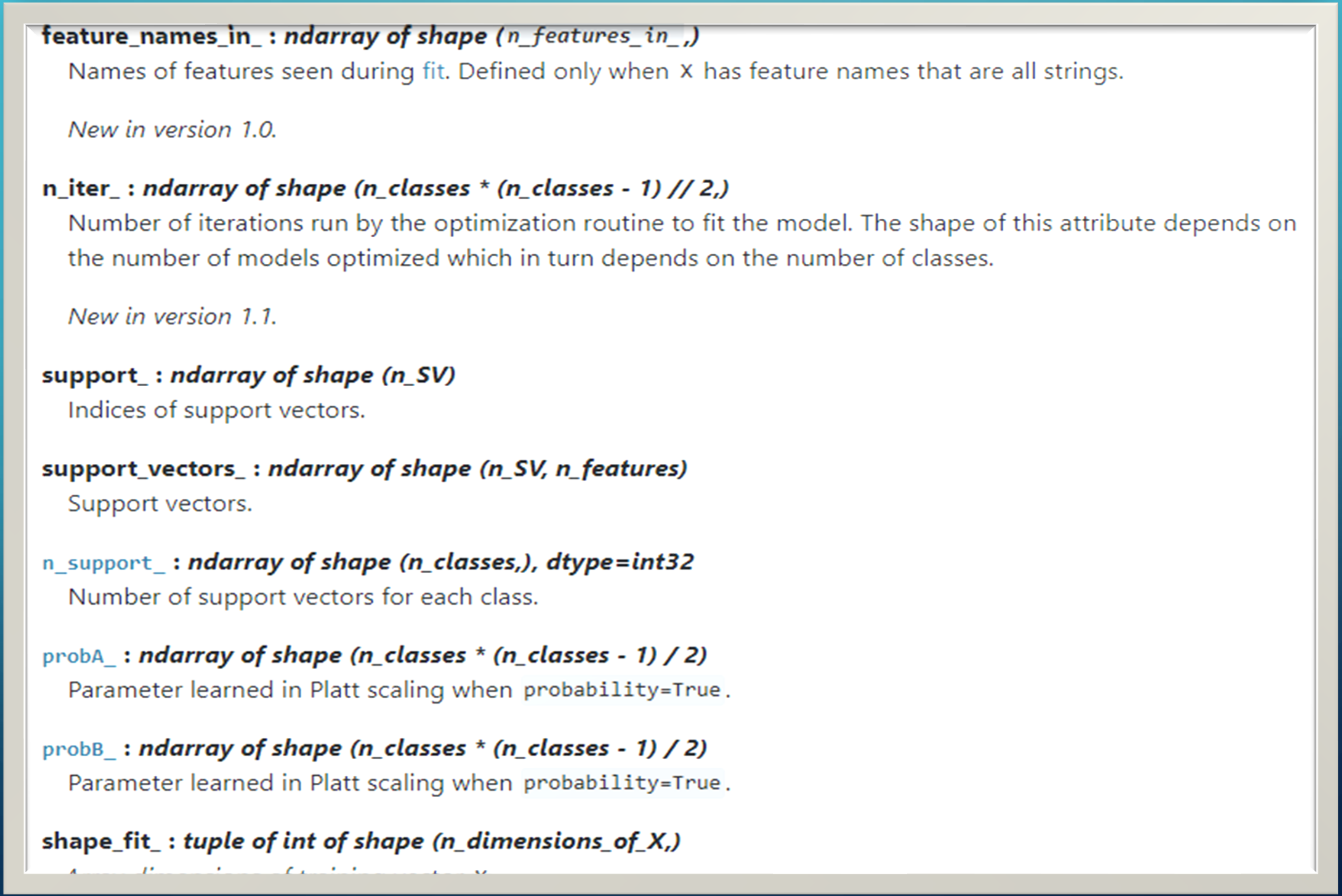
Number of features seen during fit.

New in version 0.24.

feature_names_in_ : *ndarray of shape (n_features_in_,)*

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

Some attributes of SVM Classifier:-



feature_names_in_ : *ndarray of shape (`n_features_in_`,)*

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

New in version 1.0.

n_iter_ : *ndarray of shape (n_classes * (n_classes - 1) // 2,)*

Number of iterations run by the optimization routine to fit the model. The shape of this attribute depends on the number of models optimized which in turn depends on the number of classes.

New in version 1.1.

support_ : *ndarray of shape (n_SV)*

Indices of support vectors.

support_vectors_ : *ndarray of shape (n_SV, n_features)*

Support vectors.

`n_support_` **: *ndarray of shape (n_classes,), dtype=int32***

Number of support vectors for each class.

`probA_` **: *ndarray of shape (n_classes * (n_classes - 1) / 2)***

Parameter learned in Platt scaling when `probability=True`.

`probB_` **: *ndarray of shape (n_classes * (n_classes - 1) / 2)***

Parameter learned in Platt scaling when `probability=True`.

shape_fit_ : *tuple of int of shape (n_dimensions_of_X,)*

All The Best….!!!