Support Vector Machines

Peerapon S.

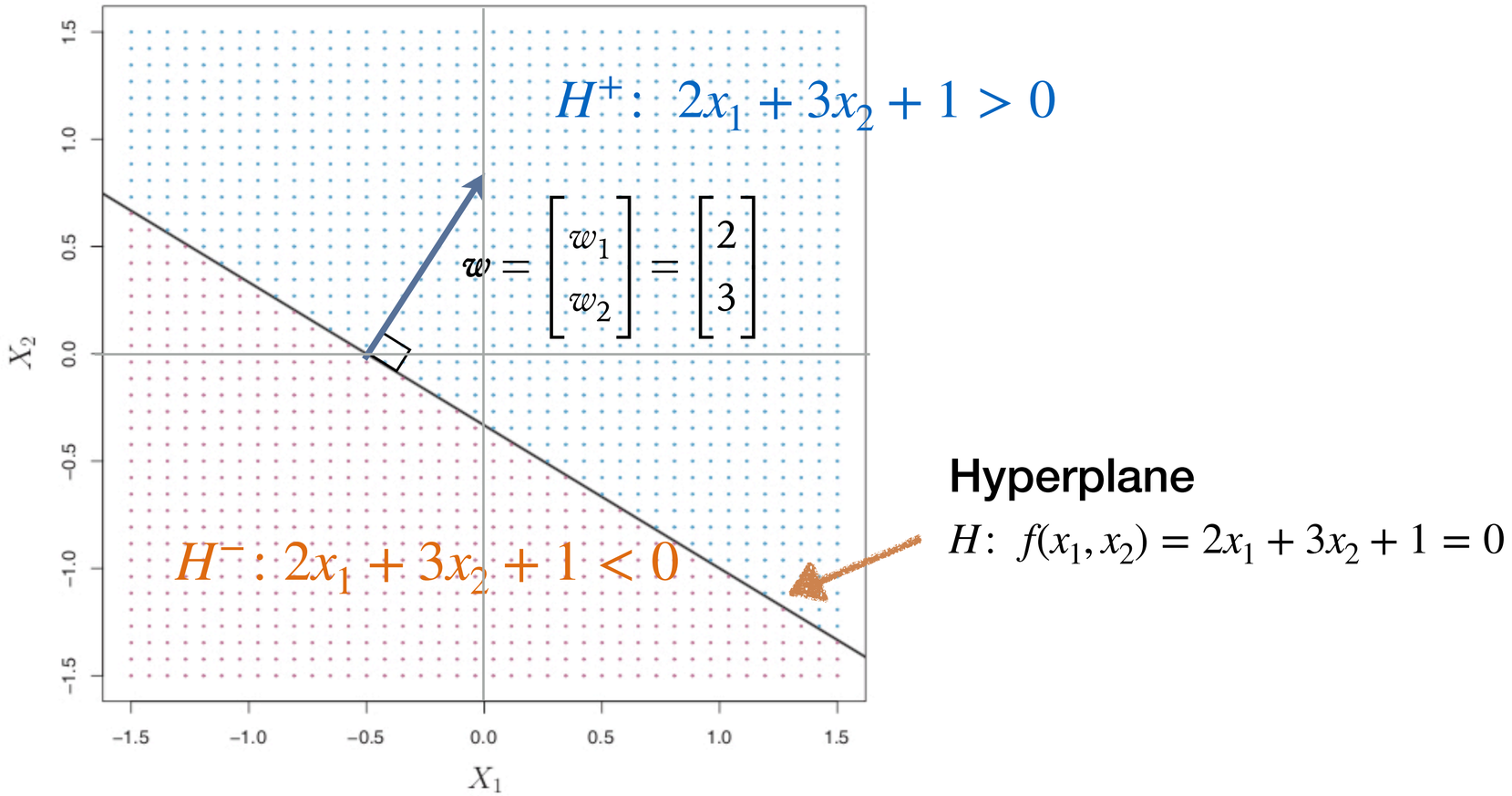
`Codes & Dataset: https://kmutt.me/cpe-ml`

Topics

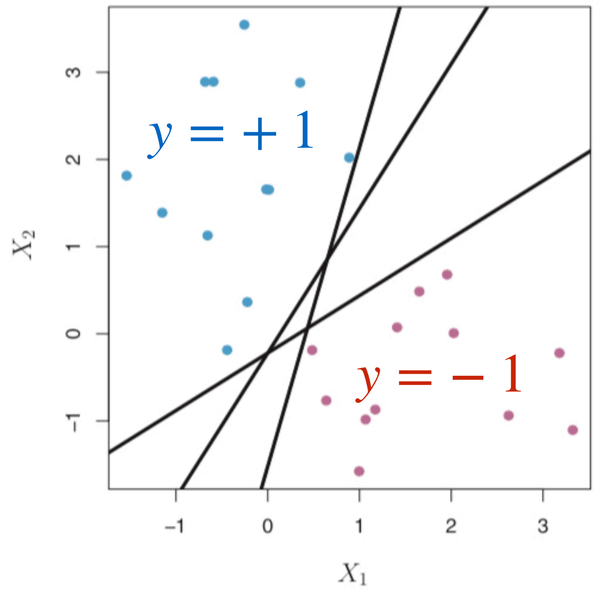
- ☐ Hyperplane
- ☐ Hard-margin classifier
- ☐ Linear support vector classifier
- ☐ Non-linear support vector classifier
- ☐ Support vector regression

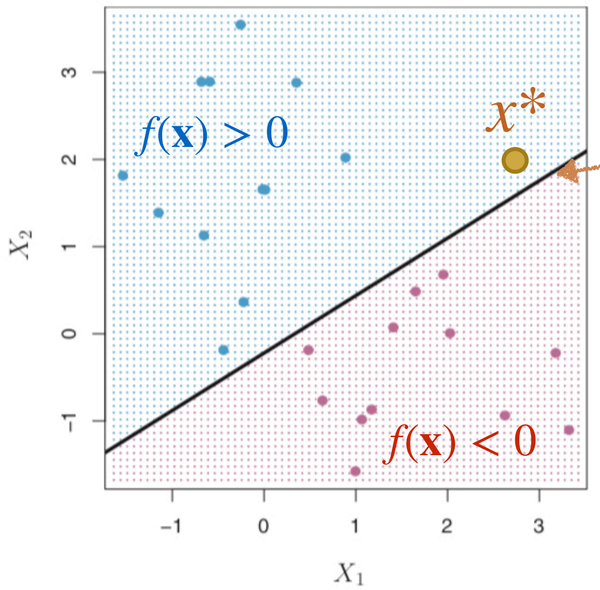
Hyperplane

- Surface dividing a p-dimension space into two halves: $H:\ f(x_1, x_2, \ldots, x_p) = w_1 x_2 + w_2 x_2 + \cdots + w_p x_p + b = 0$
- **Invariant to scaling** : What happens if you multiply a constant to f ?



Linearly Separable Data





$$f(\mathbf{x}) = w_1 x_1 + w_2 x_2 + b = 0$$
$$= \boldsymbol{w} \cdot \boldsymbol{x} + b = 0$$
$$= \boldsymbol{w}^T \boldsymbol{x} + b = 0$$

$w_1 x_1 + w_2 x_2 + b > 0$ if $y = 1$

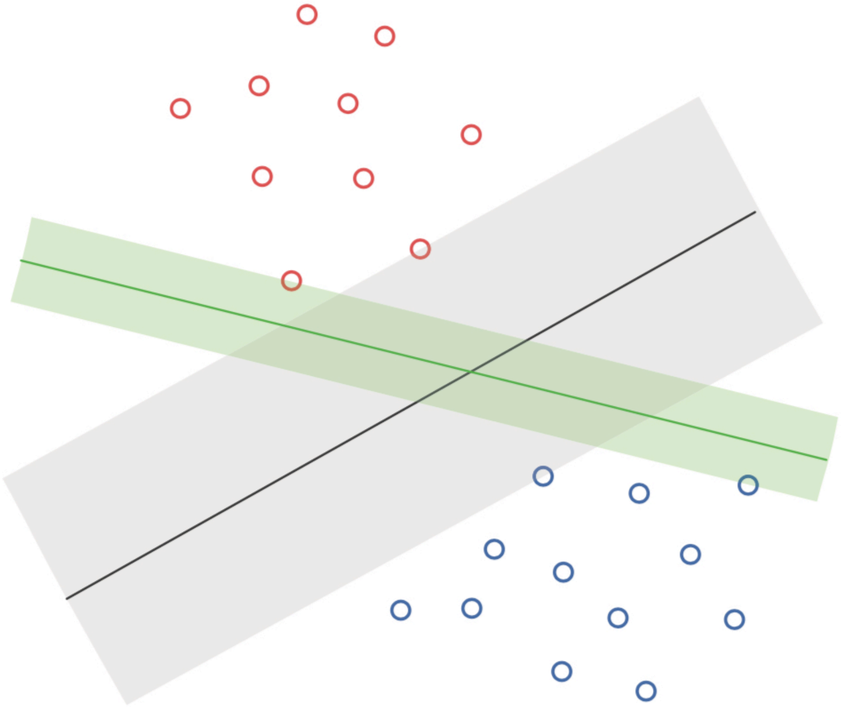
$w_1 x_1 + w_2 x_2 + b < 0$ if $y = -1$

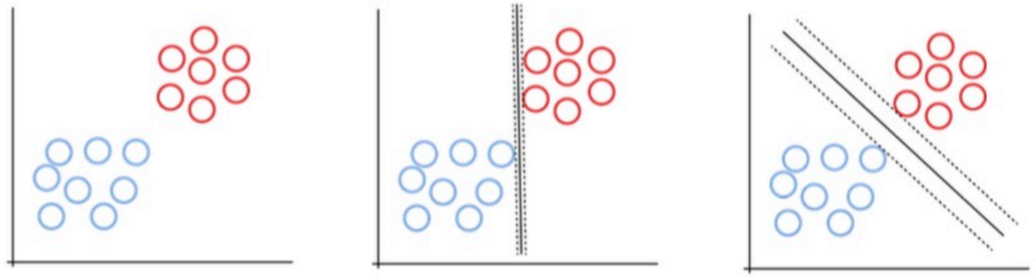
or

$y(w_1 x_1 + w_2 x_2 + b) > 0$

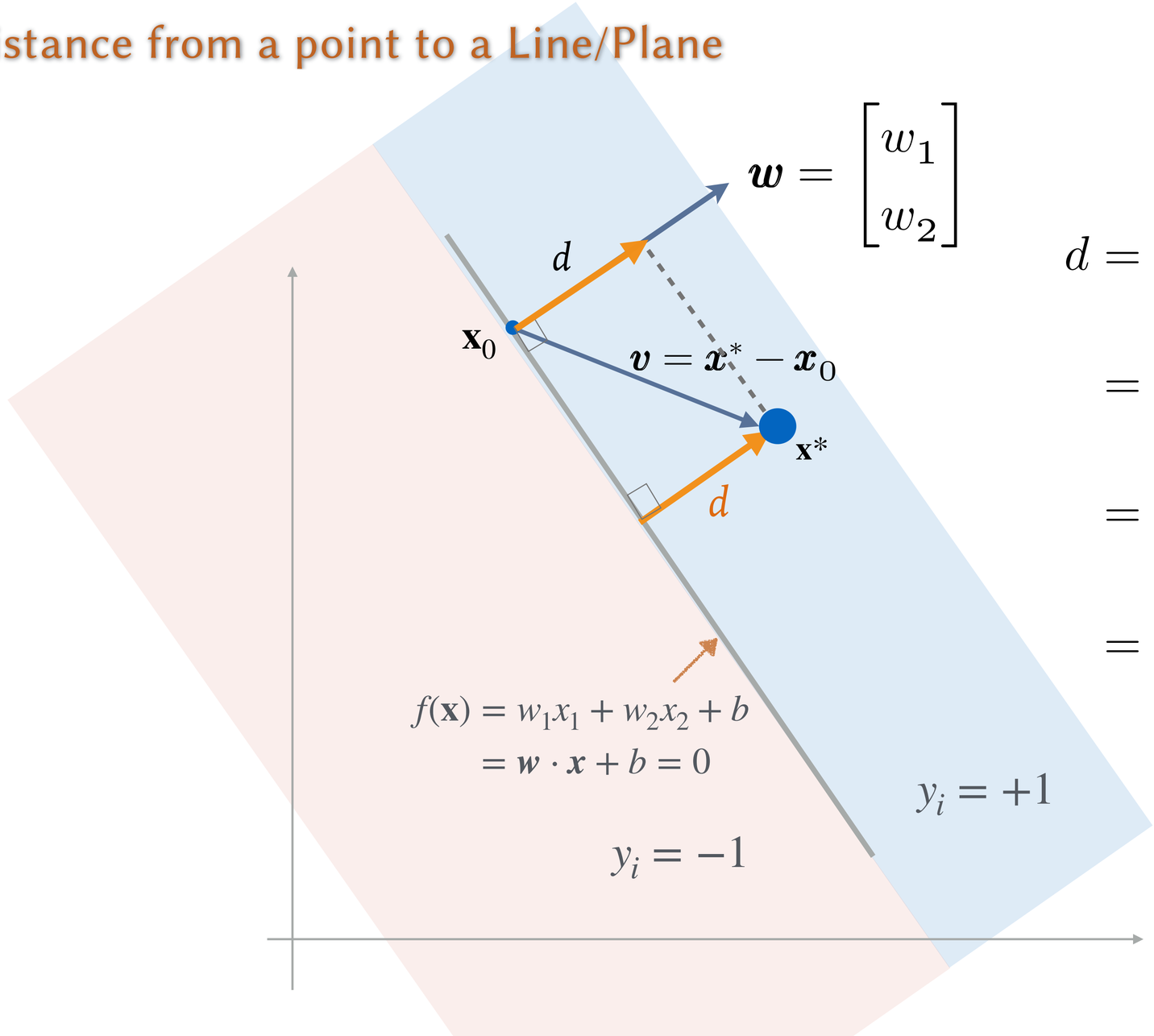
$y f(\boldsymbol{x}) > 0$

Knowing $f(\mathbf{x})$, How would you classify a new point $\mathbf{x}^*$?





Distance from a point to a Line/Plane

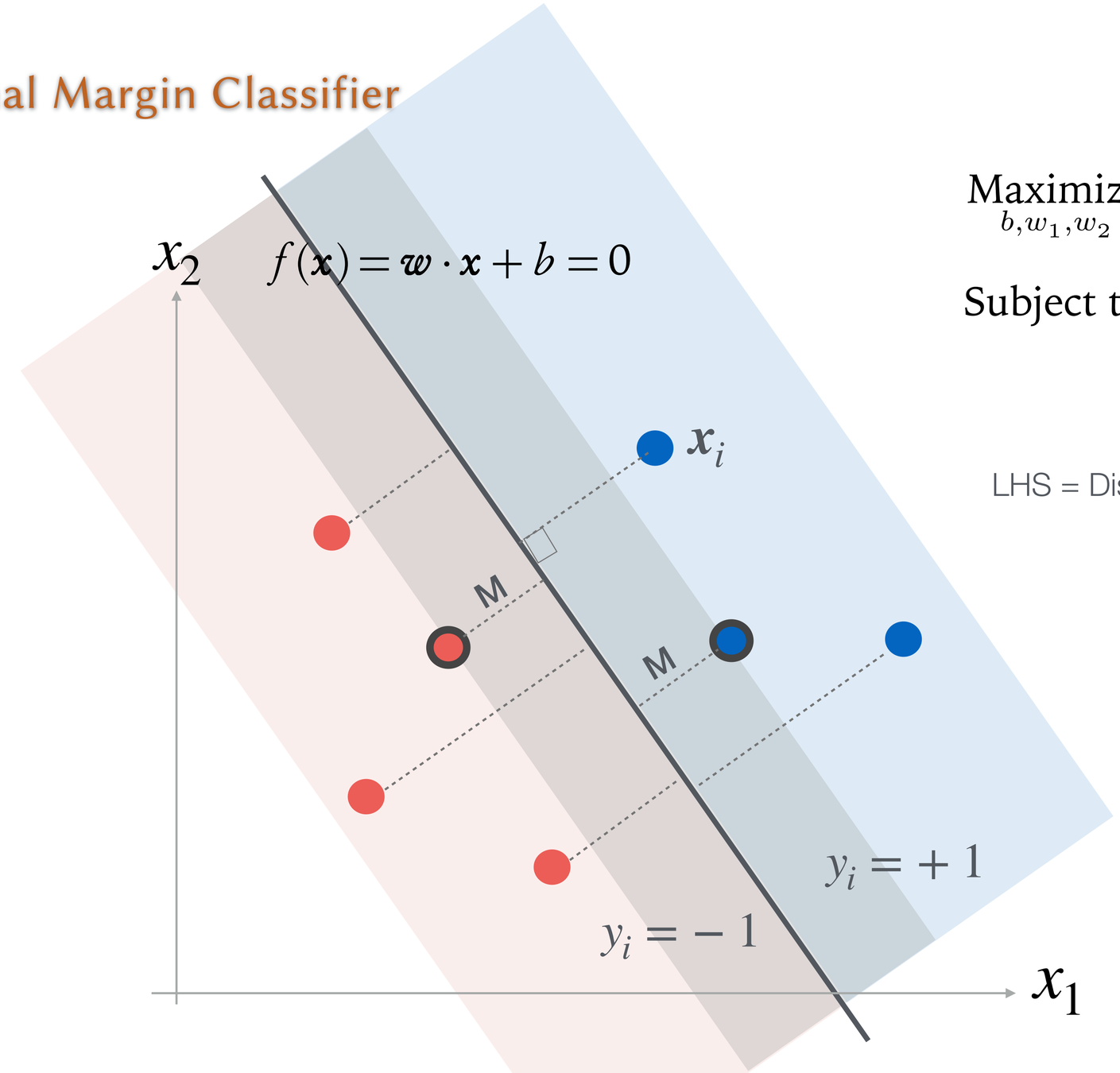


$$
\begin{aligned}
d &= \frac{\boldsymbol{w} \cdot (\boldsymbol{x}^* - \boldsymbol{x}_0)}{\|\boldsymbol{w}\|} \\
&= \frac{\boldsymbol{w} \cdot \boldsymbol{x}^* - \boldsymbol{w} \cdot \boldsymbol{x}_0}{\|\boldsymbol{w}\|} \\
&= \frac{\boldsymbol{w} \cdot \boldsymbol{x}^* + b}{\|\boldsymbol{w}\|} \\
&= \frac{f(\boldsymbol{x}^*)}{\|\boldsymbol{w}\|}
\end{aligned}
$$

Negative sign possible !

Weight vector magnitude

Maximal Margin Classifier



$$\underset{b, w_1, w_2}{\text{Maximize}} \quad M$$

$$\text{Subject to} \quad \frac{y_i f(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|} \geq M, \quad i = 1, 2, \ldots, n$$

LHS = Distance magnitude from point x_i to the hyperplane.

$$\underset{b,w_1,w_2}{\text{Maximize}} \quad M$$

$$\text{Subject to} \quad y_i f(\boldsymbol{x}_i) \geq \|\boldsymbol{w}\| M, \quad i = 1, 2, \ldots, n$$

Setting $\|\boldsymbol{w}\| = \frac{1}{M}$ (Invariant to scaling)

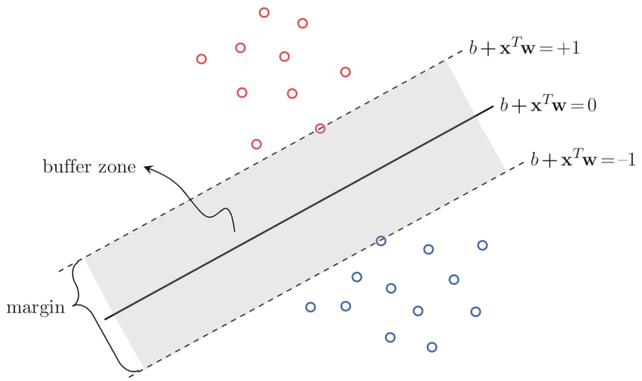
Common representation of SVM optimization problem (Quadratic programming)

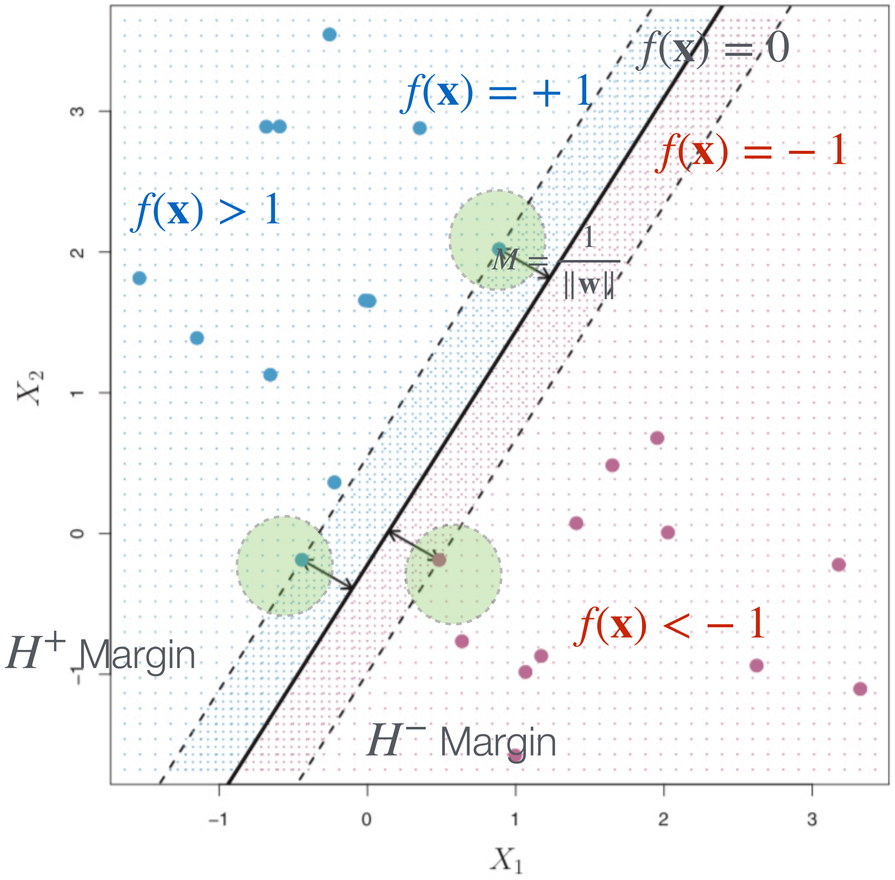
$$\underset{b,w_1,w_2}{\text{Minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\text{Subject to} \quad y_i f(\boldsymbol{x}_i) \geq 1, \quad i = 1, 2, \ldots, n$$

Support Vectors

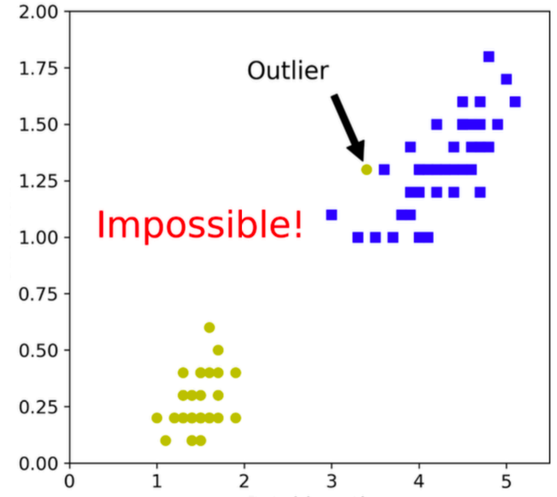
- Only a set of ***support vectors*** $S_v = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ constitutes the maximum margin classifier, with the margin $M = 1/\|\boldsymbol{w}\|$.
- Note that SV's have $y_i f(\boldsymbol{x}_i) = 1$ (Unit distance from hyperplane).

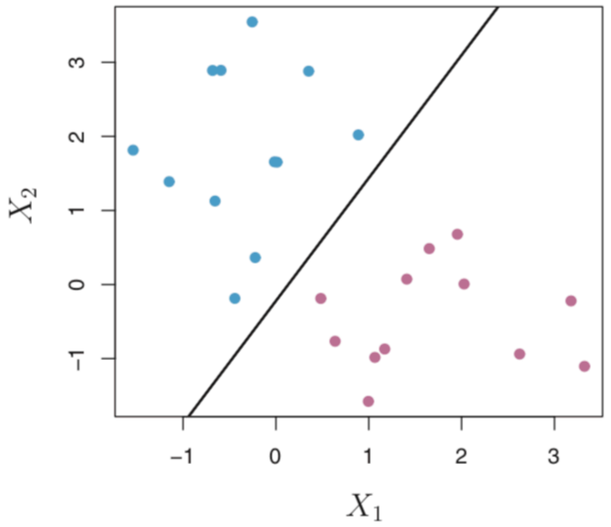


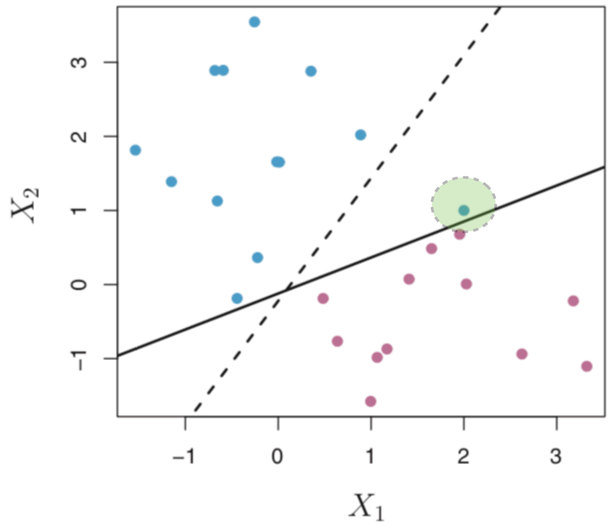


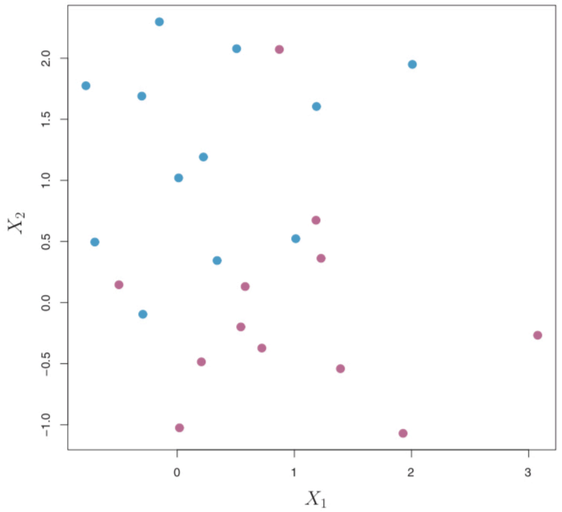
Knowing $f(\mathbf{x})$, How would you classify a new point $\mathbf{x}^*$?

Non-Separable Data: Support Vector Classifier



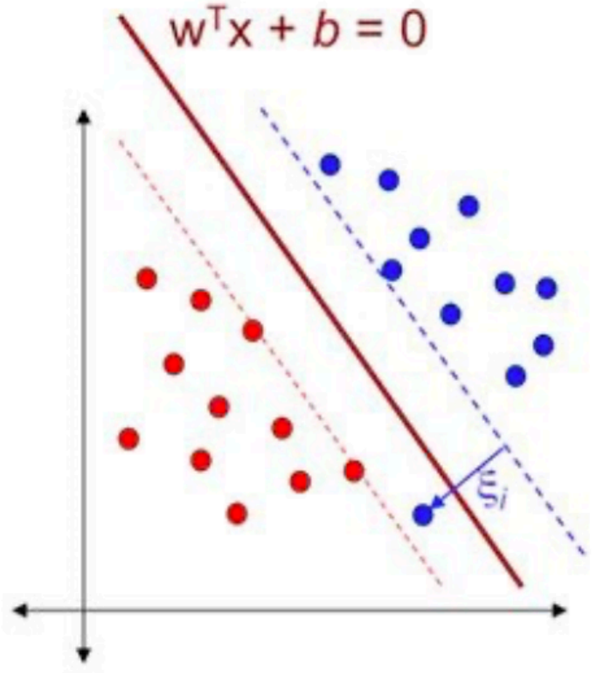




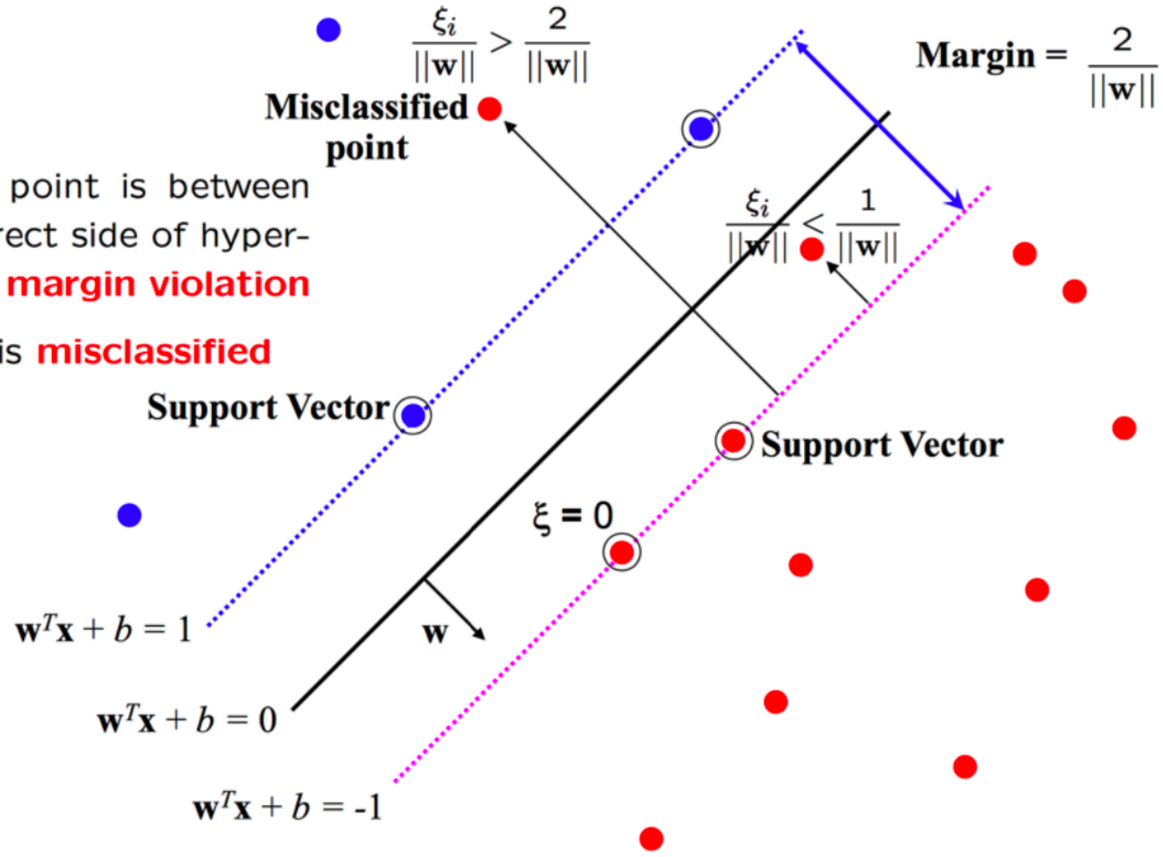


Soft-Margin Classification: Support Vector Classifier (SVC)

☐ Each sample i is permitted to have the distance $\xi_i \geq 0$ from its (correct-side) margin.



- for $0 < \xi \leq 1$ point is between margin and correct side of hyper-plane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**



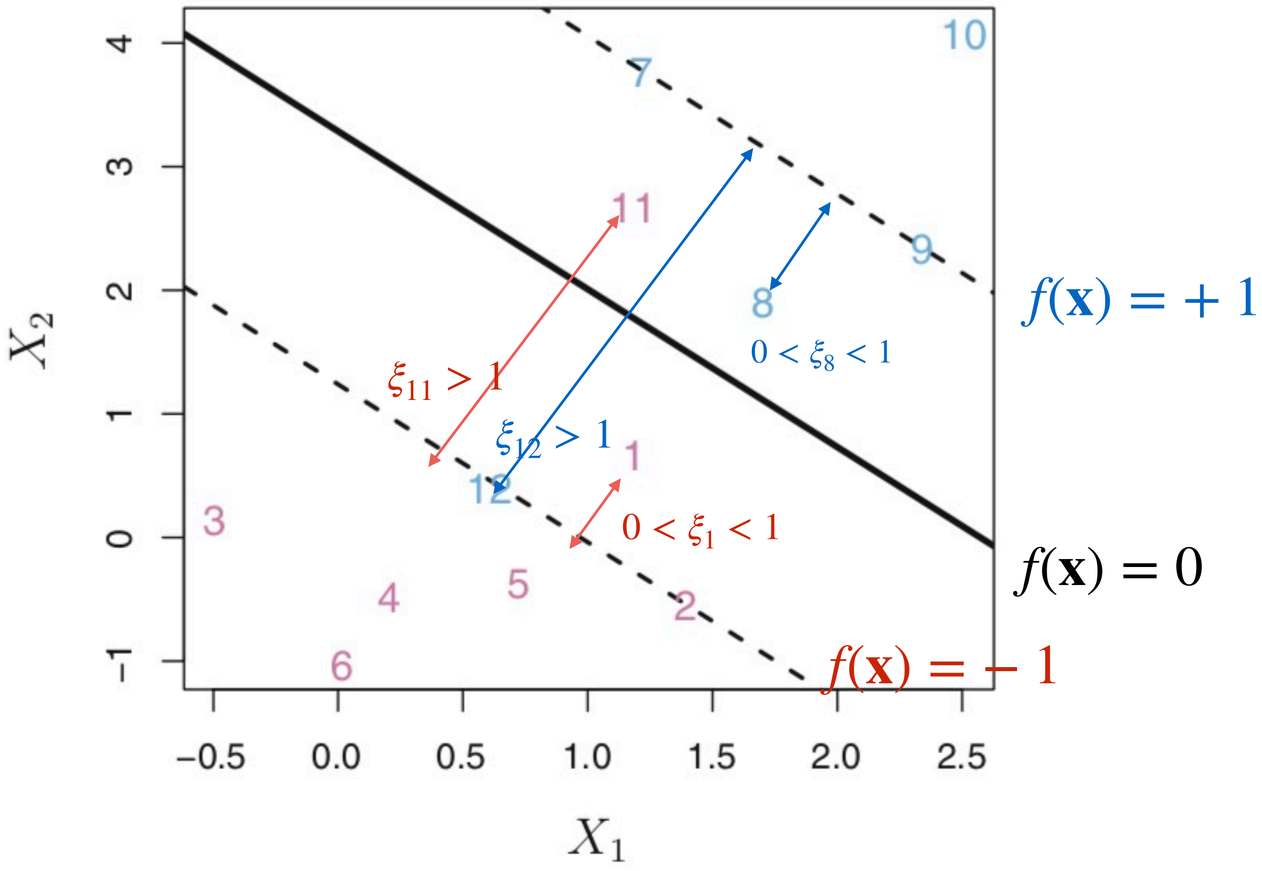
☐ Reformulate the maximal margin classifier to account for permissible errors $\{\xi_i\}$ and the ***penalty parameter*** C:

$$\underset{b,w_i,\xi_i}{\text{Minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{Subject to} \quad y_i f(\boldsymbol{x}_i) \geq 1 - \xi_i, \quad i = 1, 2, \ldots, n$$

$$\xi_i \geq 0$$

Seven support vectors
(those affecting the solution)



$$\underset{b,w_i,\xi_i}{\text{Minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{Subject to} \quad y_i f(\boldsymbol{x}_i) \geq 1 - \xi_i, \quad i = 1, 2, \ldots, n$$

$$\xi_i \geq 0$$

$$\underset{b,w_i,\xi_i}{\text{Minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{Subject to} \quad 1 - y_i f(\boldsymbol{x}_i) \leq \xi_i, \quad i = 1, 2, \ldots, n$$

$$\xi_i \geq 0$$

Penalty function

L1-penalty: $\|\boldsymbol{w}\|$

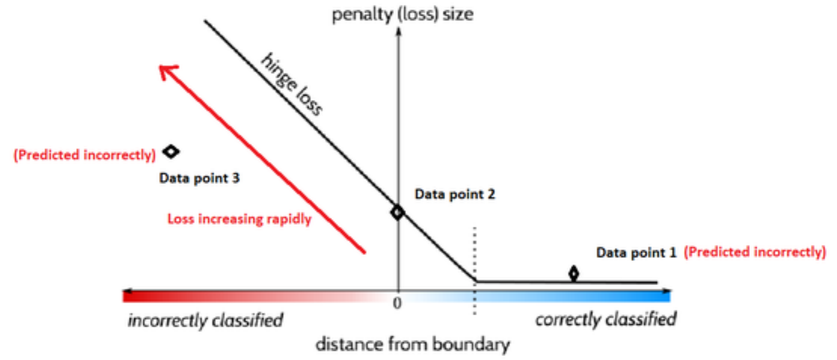
L2-penalty: $\|\boldsymbol{w}\|^2$

Loss functions (Cost of points being in wrong side)
- Hinge loss, Squared hinge loss

Primal Optimization Problem

$$\underset{b,w_i}{\text{Minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}[1 - y_i f(\boldsymbol{x}_i)]^+$$

$$\underset{b,w_i}{\text{Minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(1 - y_i f(\boldsymbol{x}_i))^2$$



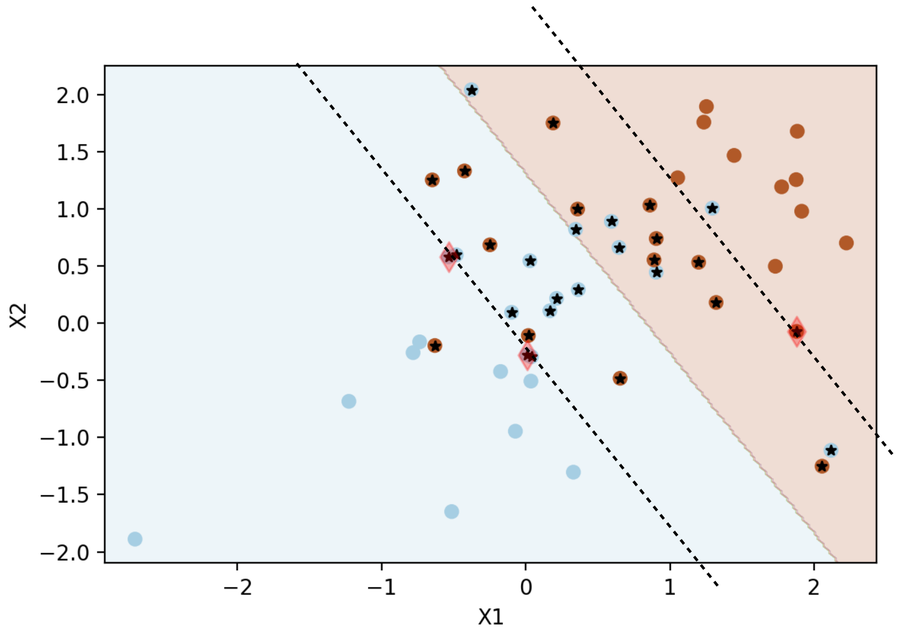
Solving SVM Optimization using Dual Problem

☐ By using the method of Lagrange multiplier, the earlier SVM formulation has the *dual problem*:

$$\underset{\lambda_i}{\text{Minimize}} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j(\boldsymbol{x}_i \cdot \boldsymbol{x}_j) - \sum_{i=1}^{n}\lambda_i$$

$$\text{Subject to} \quad 0 \le \lambda_i \le C, \quad i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n}\lambda_i y_i = 0$$



☐ Apply optimization algorithms to solve for optimal $\{\lambda_i\}$.

☐ The solution contains one of three cases for $\{\lambda_i\}$:

- Correctly classified points (on correct sides of the margins): $\lambda_i = 0$
- Support vectors on the margins: $0 < \lambda_i < C$
- Support vectors that are misclassified points or those inside the margin: $\lambda_i = C$

☐ Used when applying kernel tricks to the features (More on this later).

SVM Key Results

☐ From $\{\lambda_i\}$ and $S_v = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|S_v|}\}$, the decision function f is obtained from:

$$\boldsymbol{w} = [w_1, \ldots, w_p]^T = \sum_{i \in S_v} \lambda_i y_i \boldsymbol{x}_i, \text{ and } \quad b = \frac{1}{|S_v|} \sum_{i \in S_v} (y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)$$

$$\begin{aligned} f(\boldsymbol{x}) &= b + \boldsymbol{w} \cdot \boldsymbol{x} \\ &= b + \sum_{i \in S_v} \lambda_i y_i \, \boldsymbol{x} \cdot \boldsymbol{x_i} \\ &= b + \sum_{i \in S_v} \alpha_i (\boldsymbol{x} \cdot \boldsymbol{x}_i), \quad \alpha_i = \lambda_i y_i \text{ (Dual coefficients)} \end{aligned}$$

Linear SVC using Sklearn

```
class sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', *,
dual='auto', tol=0.0001, C=1.0, multi_class='ovr',
fit_intercept=True, intercept_scaling=1, class_weight=None,
verbose=0, random_state=None, max_iter=1000)
```

[source]

Parameters:

penalty : *{'l1', 'l2'}, default='l2'*

Specifies the norm used in the penalization. The 'l2' penalty is the standard used in SVC. The 'l1' leads to `coef_` vectors that are sparse.

loss : *{'hinge', 'squared_hinge'}, default='squared_hinge'*

Specifies the loss function. 'hinge' is the standard SVM loss (used e.g. by the SVC class) while 'squared_hinge' is the square of the hinge loss. The combination of `penalty='l1'` and `loss='hinge'` is not supported.

dual : *"auto" or bool, default="auto"*

Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features. `dual="auto"` will choose the value of the parameter automatically, based on the values of `n_samples`, `n_features`, `loss`, `multi_class` and `penalty`. If `n_samples` < `n_features` and optimizer supports chosen `loss`, `multi_class` and `penalty`, then dual will be set to True, otherwise it will be set to False.

> ***Changed in version 1.3:*** The `"auto"` option is added in version 1.3 and will be the default in version 1.5.

tol : *float, default=1e-4*

Tolerance for stopping criteria.

C : *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. For an intuitive visualization of the effects of scaling the regularization parameter C, see Scaling the regularization parameter for SVCs.

More on this later.

Attributes:

coef_ : *ndarray of shape (1, n_features) if n_classes == 2 else (n_classes, n_features)*

Weights assigned to the features (coefficients in the primal problem).

`coef_` is a readonly property derived from `raw_coef_` that follows the internal memory layout of liblinear.

intercept_ : *ndarray of shape (1,) if n_classes == 2 else (n_classes,)*

Constants in decision function.

classes_ : *ndarray of shape (n_classes,)*

The unique classes labels.

n_features_in_ : *int*

Number of features seen during fit.

> *Added in version 0.24.*

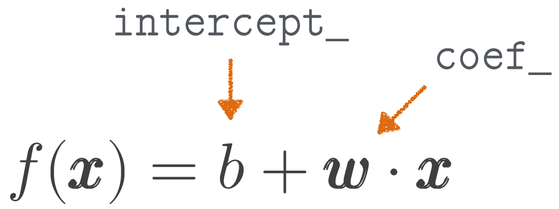
feature_names_in_ : *ndarray of shape (`n_features_in_`,)*

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

> *Added in version 1.0.*

n_iter_ : *int*

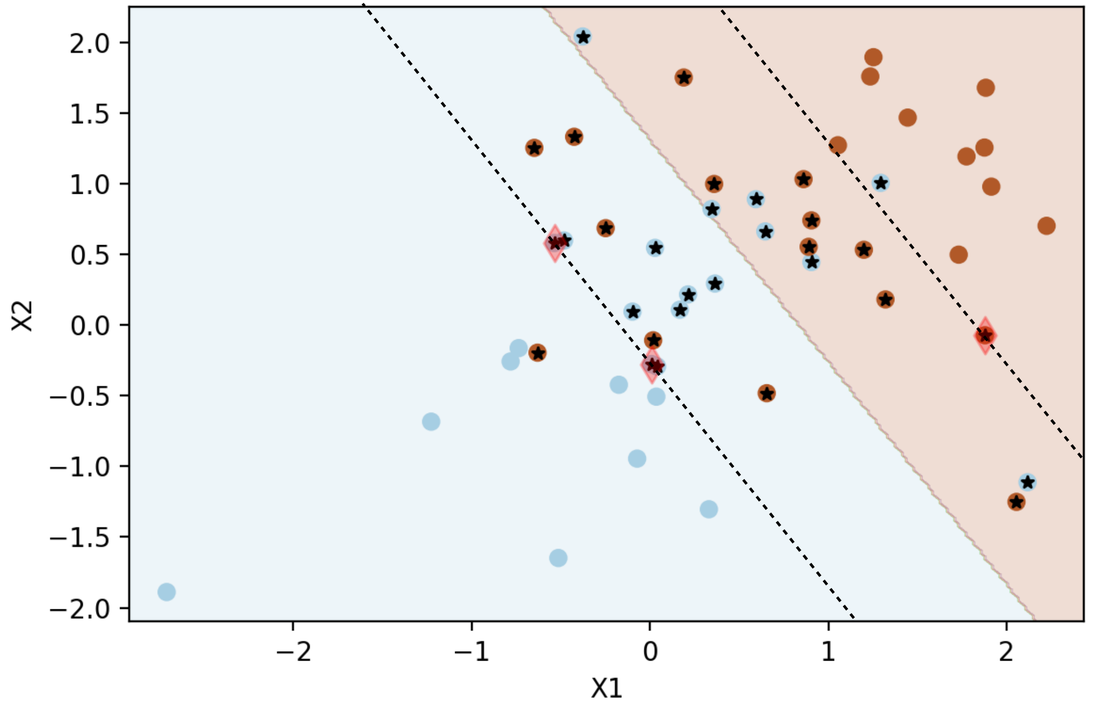
Maximum number of iterations run across all classes.



```python
# Load financial data
df = pd.read_excel('data/supervised-learning.xlsx', sheet_name='SYNDATA')

df.head()

# SVM model fitting
from sklearn.svm import LinearSVC

y = df['y']
X = df.drop('y', axis=1)

svc_linear = LinearSVC(C=1)
svc_linear.fit(X, y)

# Evaluate model performance on train data
from sklearn.metrics import confusion_matrix, classification_report

y_pred = svc_linear.predict(X)
print(confusion_matrix(y, y_pred))
print(classification_report(y, y_pred))
```

```
[[18  7]
 [ 6 19]]
              precision    recall  f1-score   support

          -1       0.75      0.72      0.73        25
           1       0.73      0.76      0.75        25

    accuracy                           0.74        50
   macro avg       0.74      0.74      0.74        50
weighted avg       0.74      0.74      0.74        50
```

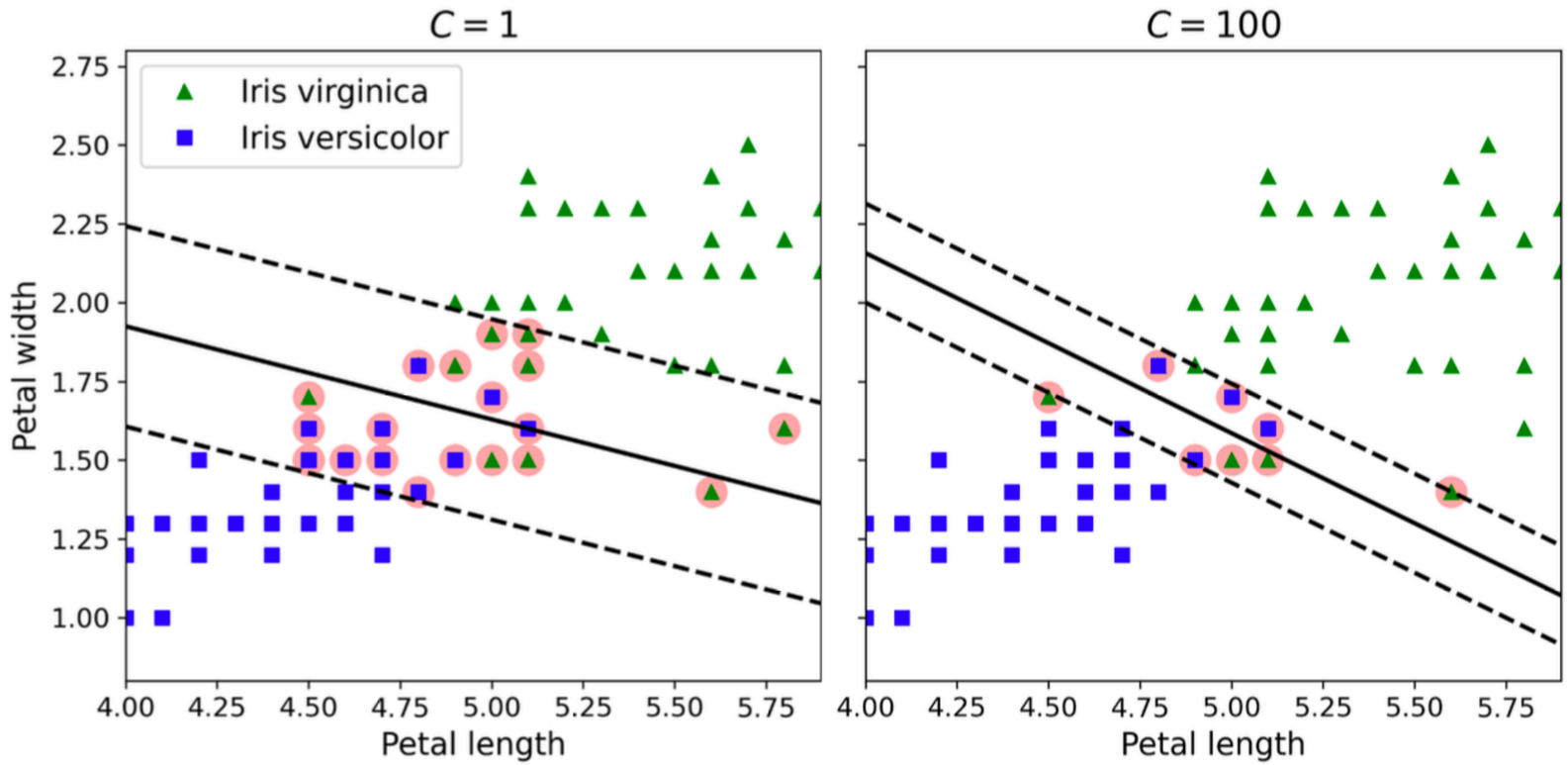
```python
# Look at the fitted coefficients
svc_linear.intercept_
svc_linear.coef_
```



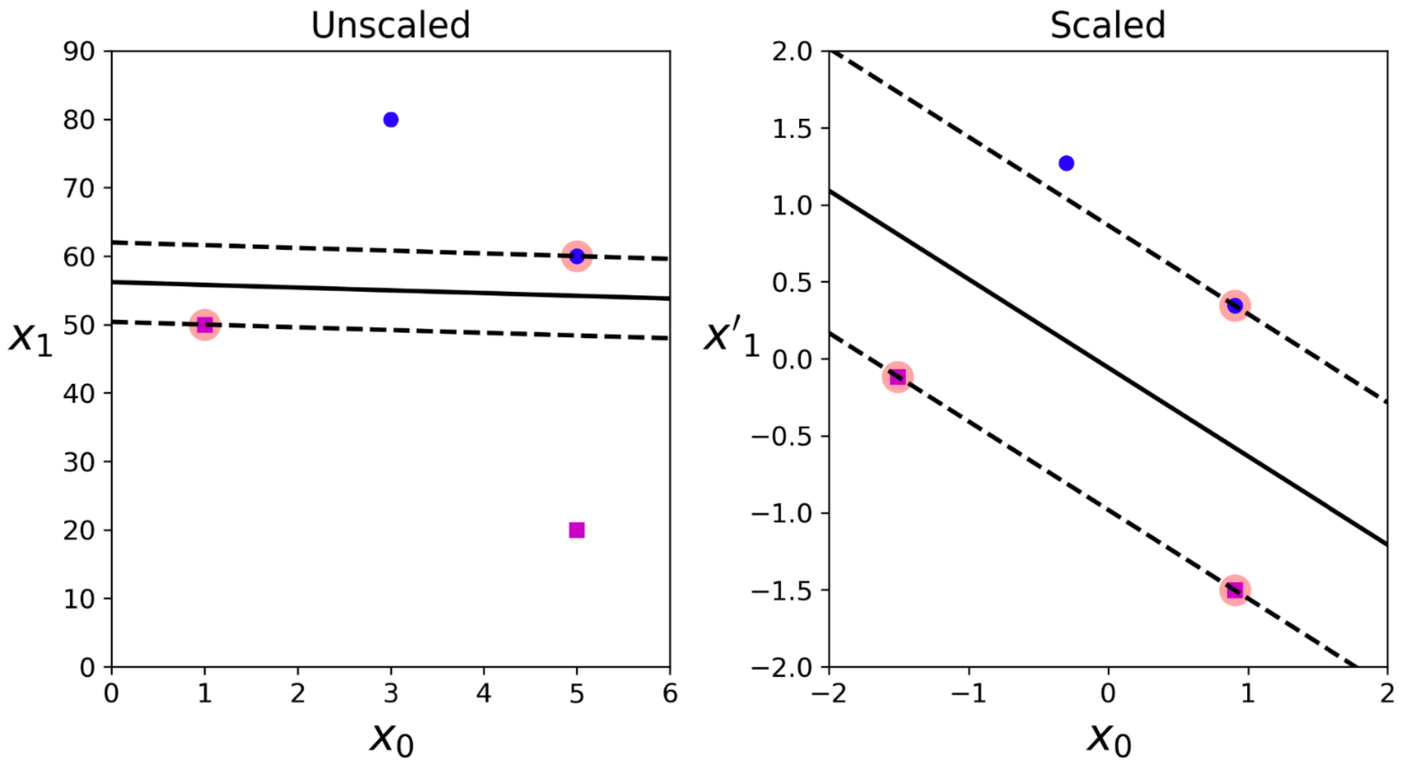

```
array([-0.83264801])
array([[0.99938331, 0.63504364]])
array([[-1.        , -1.        , -0.10582245, -1.        , -1.        ,
        -1.        , -1.        , -0.00125233, -1.        , -1.        ,
        -1.        , -1.        , -1.        , -1.        , -1.        ,
        -1.        ,  1.        ,  0.10707478,  1.        ,  1.        ,
         1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
         1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
         1.        ]])
array([[ 0.34558419,  0.82161814],
       [ 0.90535587,  0.44637457],
       [-0.53695324,  0.5811181 ],
       [ 0.3645724 ,  0.2941325 ],
       [ 0.02842224,  0.54671299],
       [-0.48211931,  0.59884621],
       [ 0.03972211, -0.29245675],
       [ 0.00814218, -0.27560291],
       [ 1.29406381,  1.00672432],
       [ 0.213643  ,  0.21732193],
       [ 2.11783876, -1.11202076],
       [-0.37760501,  2.04277161],
       [ 0.646703  ,  0.66306337],
       [ 0.16746474,  0.10901409],
       [-0.09826997,  0.09548303],
       [ 0.59374807,  0.89116695],
       [ 1.3208483 ,  0.18176977],
       [ 1.87916062, -0.07178742],
       [-0.24874889,  0.68610053],
       [ 0.01781188, -0.10737305],
       [ 1.19958453,  0.53325038],
       [-0.64878737,  1.25438812],
       [ 0.18918542,  1.75224383],
       [ 0.65478429, -0.48181827],
       [ 0.88998924,  0.55417185],
       [-0.63084923, -0.19516308],
       [ 0.35975663,  0.9989512 ],
       [ 0.90517166,  0.74115194],
       [ 2.0557428 , -1.25085428],
       [ 0.86134467,  1.0330001 ],
       [-0.42534896,  1.33281361]])
```

Regularization/Penalty Parameter C

- Inversely proportional to the "budget" of permissible errors.



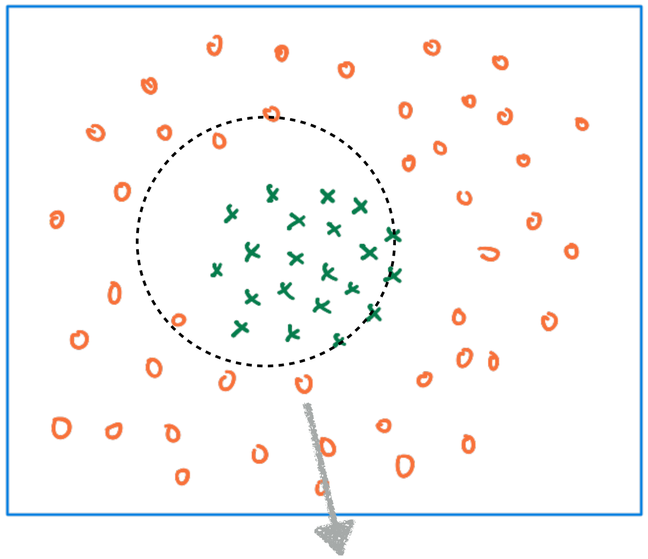
Effect of Data Scaling



Penalty parameter $C = 100$

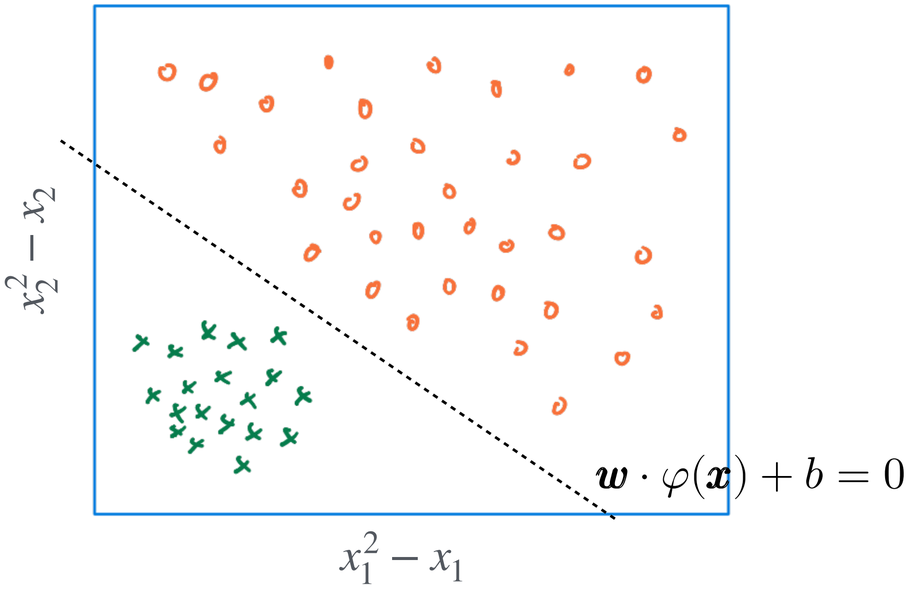
Non-Linear Decision Boundary

- Non-linear boundary can be treated by enlarging the input space with some non-linear function such as quadratic or cubic polynomial functions.
- Extend the flexibility of SVM to handle more complex decision boundary.



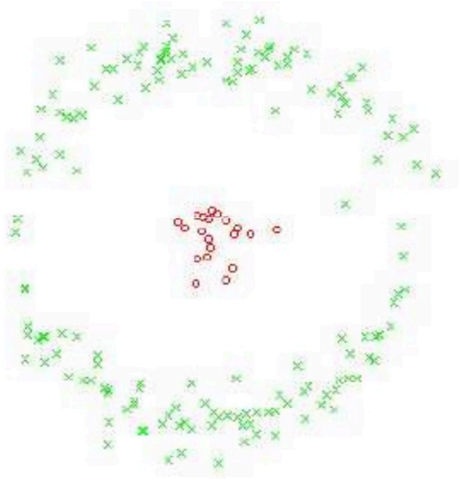
$$(x_1 - 0.5)^2 + (x_2 - 0.5)^2 = 0.2^2$$

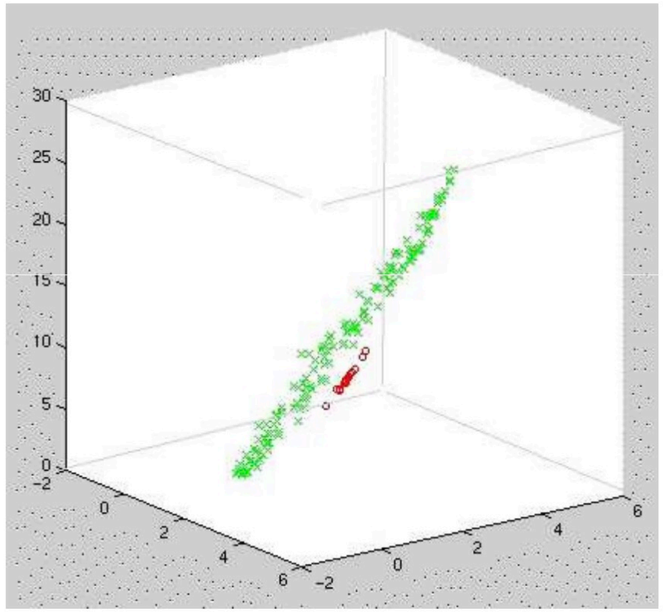
$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46$$

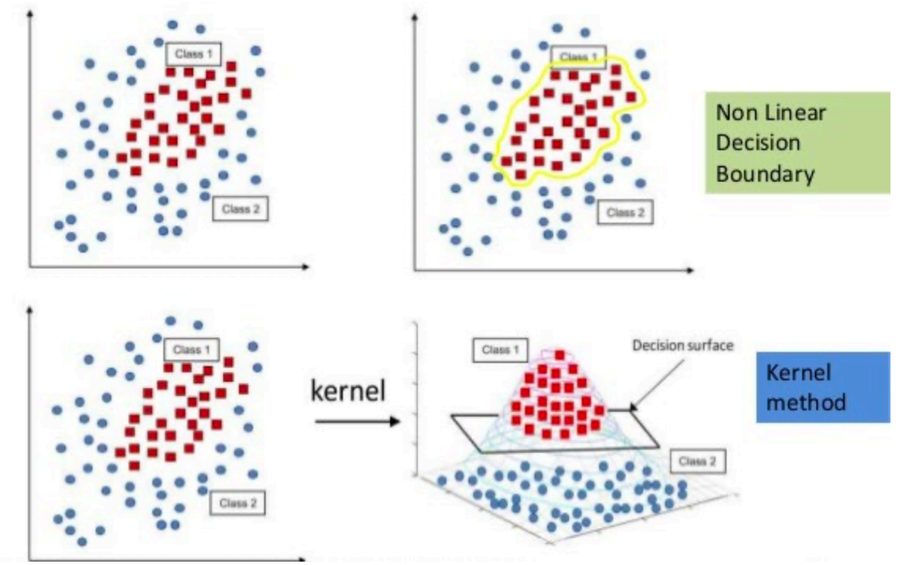


Non-linear transformation: $\varphi(\boldsymbol{x}) : (x_1, x_2) \Longrightarrow (x_1^2 - x_1, x_2^2 - x_2)$

Enhancing Input Space



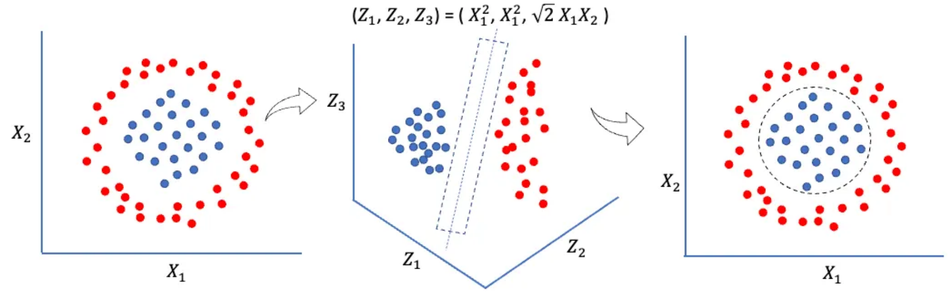




$$\Phi : \mathbf{R}^2 \rightarrow \mathbf{R}^3$$

$$(x_1, x_2) \mapsto (x_1, x_2, x_1^2 + x_2^2)$$

Input space Feature space



☐ From the SVM dual optimization problem,

$$\underset{\lambda_i}{\text{Minimize}} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j(\boldsymbol{x}_i \cdot \boldsymbol{x}_j) - \sum_{i=1}^{n}\lambda_i$$

$$\text{Subject to} \quad 0 \leq \lambda_i \leq C, \quad i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n}\lambda_i y_i = 0$$

☐ Involves all pair-wise dot/inner products of data points.

☐ If the feature transformation $\varphi(\cdot)$ is applied, the objective becomes

$$\underset{\lambda_i}{\text{Minimize}} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j(\varphi(\boldsymbol{x}_i) \cdot \varphi(\boldsymbol{x}_j)) - \sum_{i=1}^{n}\lambda_i$$

☐ Suppose the original input space has two columns (x_1, x_2) and the feature space is $(x_1, x_2, x_1x_2, x_1^2, x_2^2)$.

☐ Then,

All pairwise dot product calculation

Input space

-3	-2
5	1
7	2
...	...
...	...
...	...
...	...
4	1

Feature space

-3	-2	6	9	4
5	1	5	25	1
7	2	14	49	4
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
4	3	12	16	9

Kernel Function

- A kernel is any function that takes two vectors from the same space as inputs and returns a scalar.
- Ex:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x} \cdot \boldsymbol{z}$$

$$K(\boldsymbol{x}, \boldsymbol{z}) = \varphi(\boldsymbol{x}) \cdot \varphi(\boldsymbol{z})$$

- Consider the degree-2 polynomial kernel corresponding to the ***feature transformation/mapping*** $\varphi_{poly}(\boldsymbol{x})$:

$$\varphi_{poly}(\boldsymbol{x}) : (x_1, x_2) \Longrightarrow (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$

dim = 2

dim = 5 (linear+interaction+quadratic terms)

- For two observations $\mathbf{x} = (2,3)$ and $\mathbf{z} = (4,5)$, the dot product after the transformation is given by

$$\varphi(\boldsymbol{x}) = (1, 2\sqrt{2}, 3\sqrt{2}, 6\sqrt{2}, 4, 9)$$
$$\varphi(\boldsymbol{z}) = (1, 4\sqrt{2}, 5\sqrt{2}, 20\sqrt{2}, 16, 25)$$
$$K(\boldsymbol{x}, \boldsymbol{z}) = \varphi_{poly}(\boldsymbol{x}) \cdot \varphi_{poly}(\boldsymbol{z}) = 1 + 16 + 30 + 240 + 64 + 225 = 576$$

Input space

2	3
4	5
...	...
...	...
...	...
...	...
...	...
...	...

Feature space

1	2.83	4.24	8.48	4	9
1	5.66	7.07	28.28	16	25
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...

Kernel Trick

- Consider the feature transformation $\varphi(\cdot)$:

$$\varphi_{poly}(\boldsymbol{x}) : (x_1, x_2) \Longrightarrow (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$

- Given two vectors $\boldsymbol{x}, \boldsymbol{z}$ from the input space, the dot product of two transformed vectors are given by

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{z}) &= \varphi_{poly}(\boldsymbol{x}) \cdot \varphi_{poly}(\boldsymbol{z}) \\ &= 1 + 2x_1z_1 + 2x_2z_2 + (x_1z_1)^2 + 2x_1x_2z_1z_2 + (x_2z_2)^2 \\ &= (1 + \boldsymbol{x} \cdot \boldsymbol{z})^2 \end{aligned}$$

- For some non-linear feature transformation/mapping $\varphi(\cdot)$, the kernel function can be evaluated directly from the input space without computing $\varphi(\cdot)$.
  - Allow for efficient computations of fitting the SVM model.
  - Can also represent infinite-dimensional feature spaces.

SVC with Kernel Tricks

☐ With non-linear feature transformation, the SVM dual problem becomes

$$\begin{aligned} \underset{\lambda_i}{\text{Minimize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) - \sum_{i=1}^{n}\lambda_i \\ \text{Subject to} \quad & 0 \le \lambda_i \le C, \quad i = 1, 2, \ldots, n \\ & \sum_{i=1}^{n}\lambda_i y_i = 0 \end{aligned}$$

☐ Given the set of support vectors $\boldsymbol{S}_v$, we can compute the decision (classifier) function from

$$\begin{aligned} f(\boldsymbol{x}) &= b + \sum_{i\in S_v}\alpha_i(\boldsymbol{x}\cdot\boldsymbol{x}_i), \quad \alpha_i = \lambda_i y_i \\ &= b + \sum_{i\in S_v}\alpha_i K(\boldsymbol{x},\boldsymbol{x}_i) \end{aligned}$$

Dual coefficients
Solved from SVM dual problem by computing $K(\cdot)$ for all pairs of data points.

Common Kernels

Linear kernel: $K(\boldsymbol{x}, \boldsymbol{z}) = \varphi_{linear}(\boldsymbol{x}) \cdot \varphi_{linear}(\boldsymbol{z}) = \boldsymbol{x} \cdot \boldsymbol{z}$

Polynomial kernel: $K(\boldsymbol{x}, \boldsymbol{z}) = \varphi_{poly}(\boldsymbol{x}) \cdot \varphi_{poly}(\boldsymbol{z}) = (c_0 + \gamma \boldsymbol{x} \cdot \boldsymbol{z})^d$

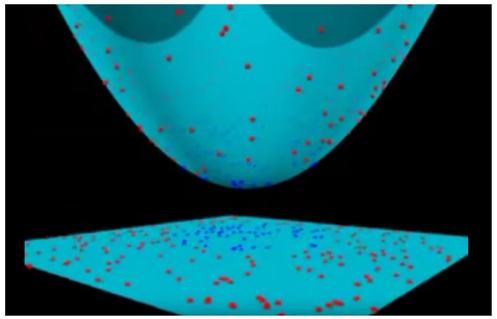
c_0 (***coef0***), γ (***gamma***), d (***degree***)

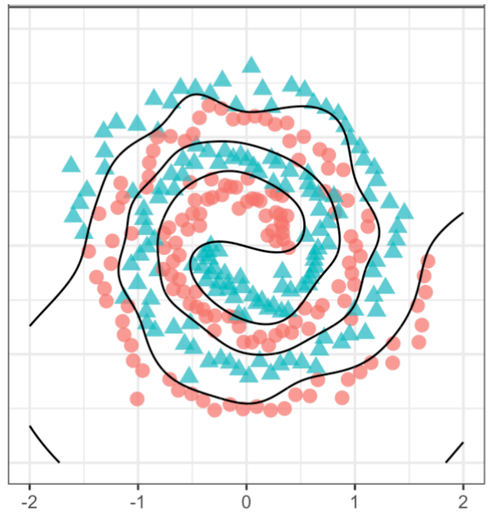
Radial Basis Function (RBF) kernel: $K(\boldsymbol{x}, \boldsymbol{z}) = \varphi_{rbf}(\boldsymbol{x}) \cdot \varphi_{rbf}(\boldsymbol{z}) = e^{-\gamma \|\boldsymbol{x} - \boldsymbol{z}\|^2} = e^{-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2}}$

Sigmoid kernel: $K(\boldsymbol{x}, \boldsymbol{z}) = \varphi_{sgm}(\boldsymbol{x}) \cdot \varphi_{sgm}(\boldsymbol{z}) = \tanh(\gamma \boldsymbol{x} \cdot \boldsymbol{z} + c_0)$

γ (***gamma***) called ***slope***,
c_0 (***coef0***) called ***Intercept***

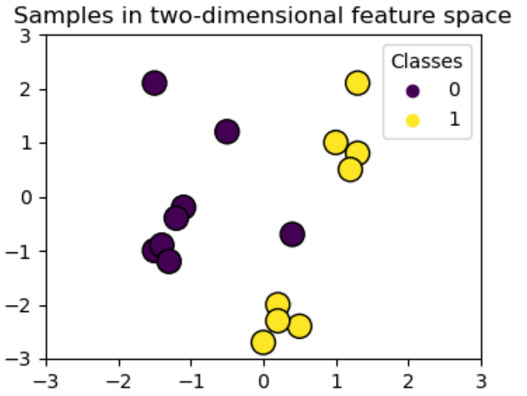
Polynomial kernel

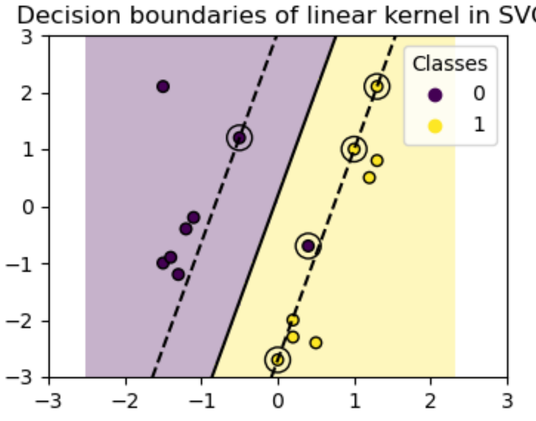


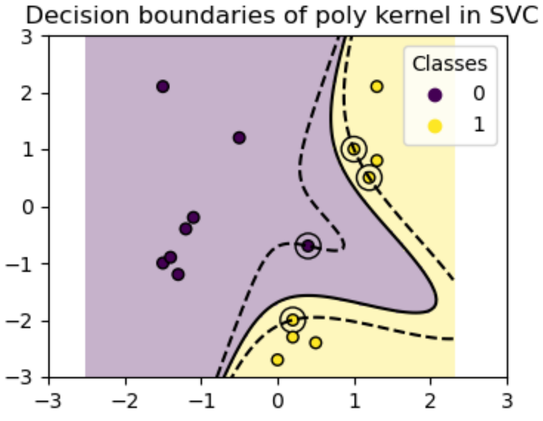


SVM with RBF kernel

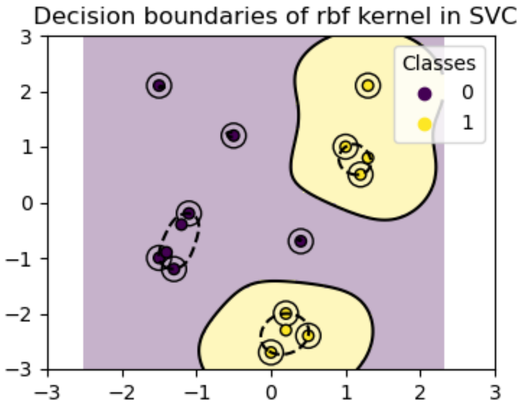
https://scikit-learn.org/stable/modules/metrics.html#metrics



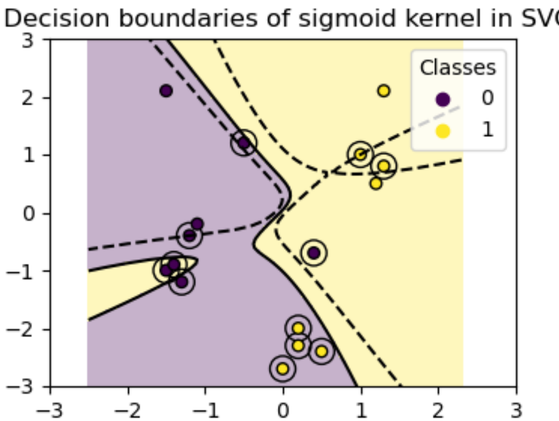




Curved decision boundary



closer points are grouped together



Curved and irregular decision boundary that may not generalized well.

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)*

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.

degree : *int, default=3*

Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

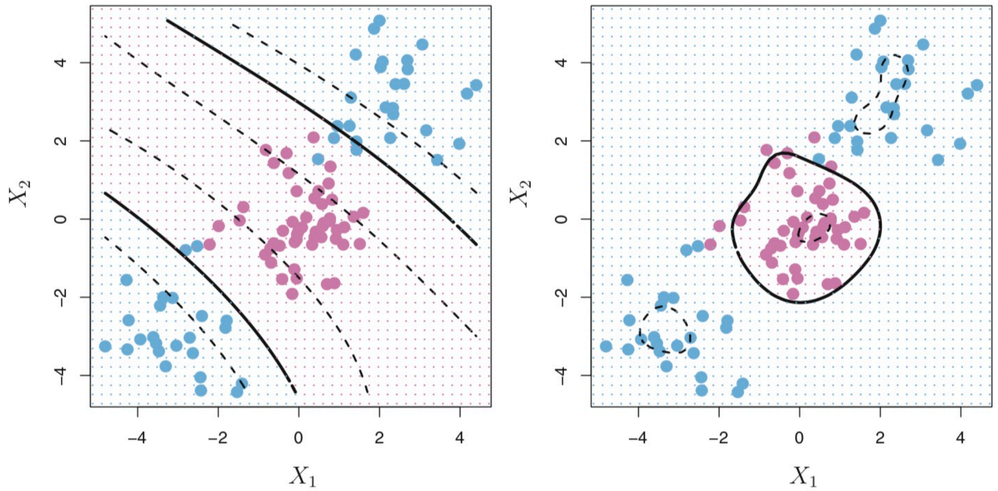
coef0 : *float, default=0.0*

Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

class_weight : *dict or 'balanced', default=None*

Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

Ex: SVC with Polynomial kernel p = 3



Attributes:

class_weight_ : *ndarray of shape (n_classes,)*
Multipliers of parameter C for each class. Computed based on the `class_weight` parameter.

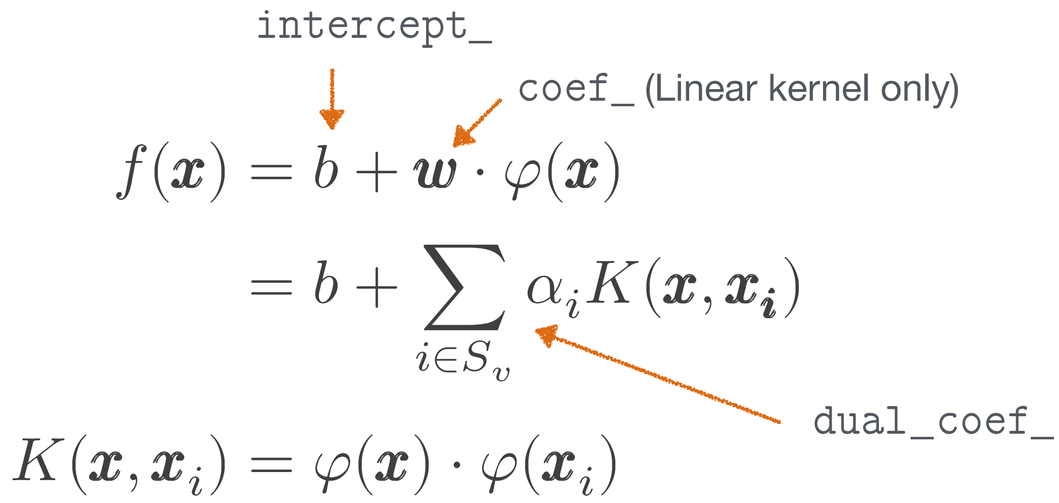
classes_ : *ndarray of shape (n_classes,)*
The classes labels.

`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)

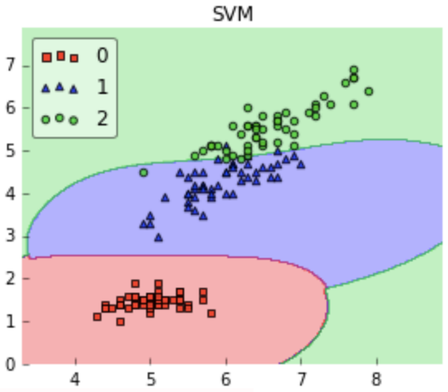
intercept_ : *ndarray of shape (n_classes * (n_classes - 1) / 2,)*
Constants in decision function.



Multiclass SVM

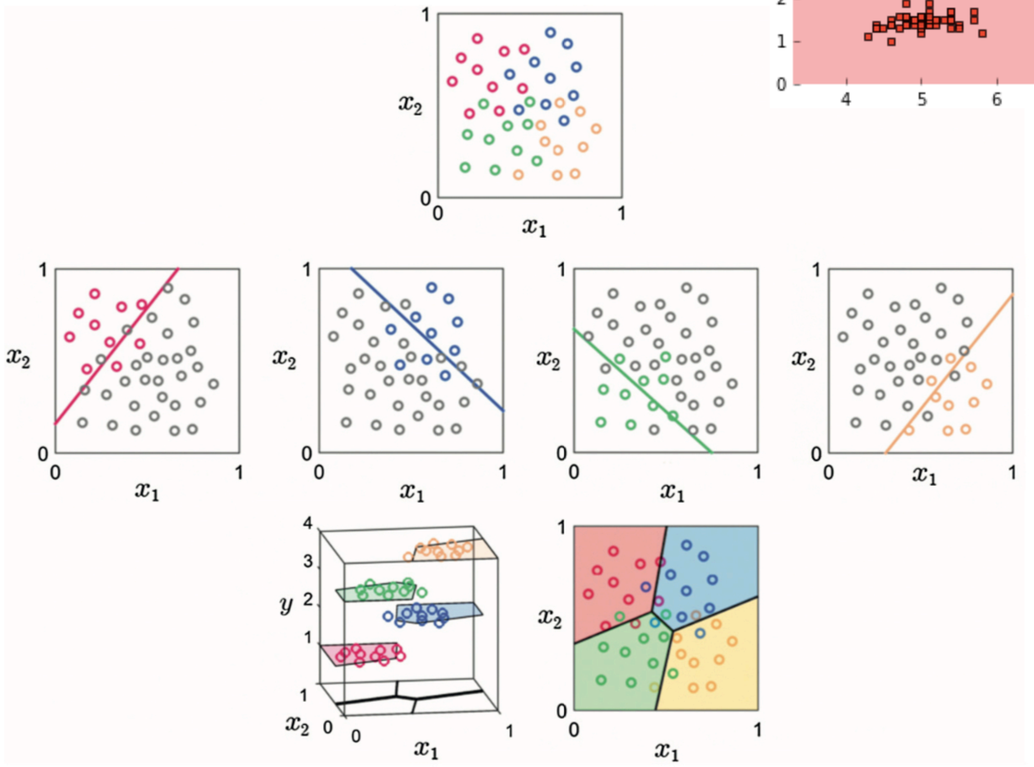
☐ One-vs-One

- Fit $\binom{K}{2}$ classifiers.
- Tally # times that $\boldsymbol{x}^*$ is assigned to each class and take the majority vote.

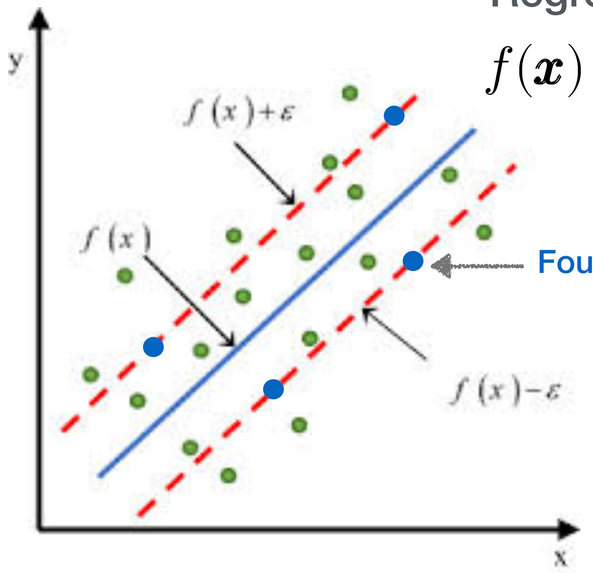


☐ One-vs-Rest (OVR) or One-vs-All

- Fit K classifiers, one with remaining $K-1$ classes.
- Assign $\boldsymbol{x}^*$ to classifier whose $f(\boldsymbol{x}^*)$ is largest.



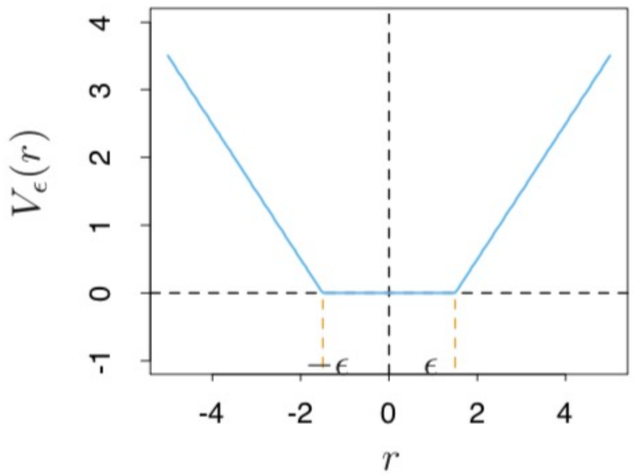
Support Vector Regression (Epsilon-SVR)



Regression line

$$f(\boldsymbol{x}) = b + \boldsymbol{w} \cdot \boldsymbol{x}$$

Four support vectors



$$\text{Minimize} \quad L(b, \boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} V_\varepsilon(y_i - f(\boldsymbol{x}_i))$$

$$V_\varepsilon(r) = \max(0, |r| - \varepsilon)$$

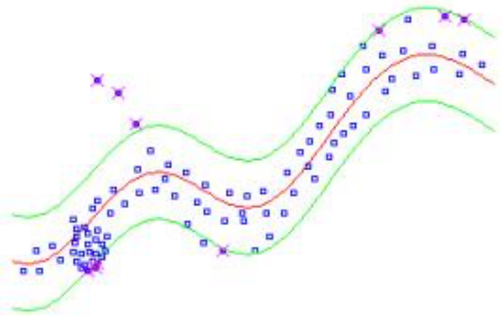
Size of **epsilon tube** where no penalty is applied.

☐ Ex: For non-linear (polynomial) feature transformation,

$$\varphi(\boldsymbol{x}) : (x_1, x_2) \Longrightarrow (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$

☐ The regression function becomes

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{w} \cdot \varphi(\boldsymbol{x}) + b \\ &= w_0 + \sqrt{2}w_1x_1 + \sqrt{2}w_2x_2 + \sqrt{2}w_{12}x_1x_2 \\ &\quad + w_{11}x_1^2 + w_{22}x_2^2 + b \end{aligned}$$



Credit: http://kernelsvm.tripod.com

☐ Given a set of support vectors S_v, the fitted SVR function has the form

$$f(\boldsymbol{x}) = b + \sum_{i \in S_v} \alpha_i K(\boldsymbol{x}, \boldsymbol{x_i})$$

$$K(\boldsymbol{x}, \boldsymbol{x}_i) = \varphi(\boldsymbol{x}) \cdot \varphi(\boldsymbol{x}_i)$$

class sklearn.svm.SVR(, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)*

Parameters:

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 precompute the kernel matrix.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

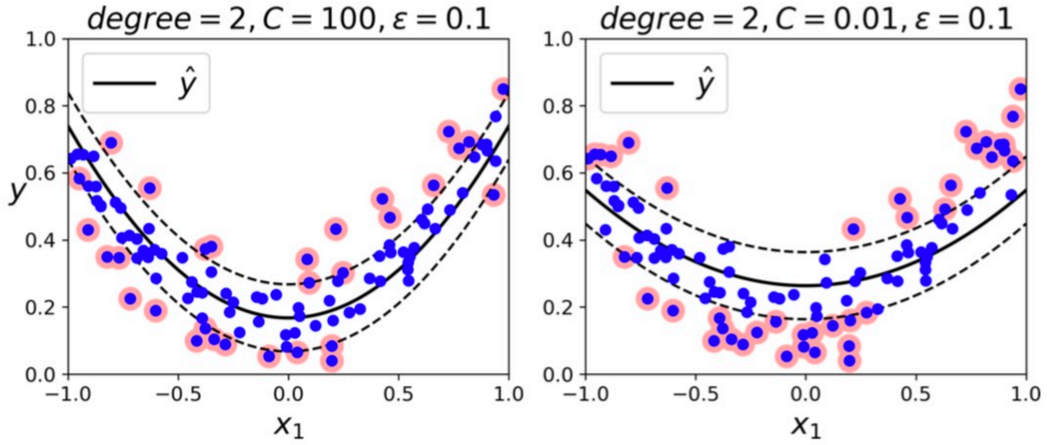
Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

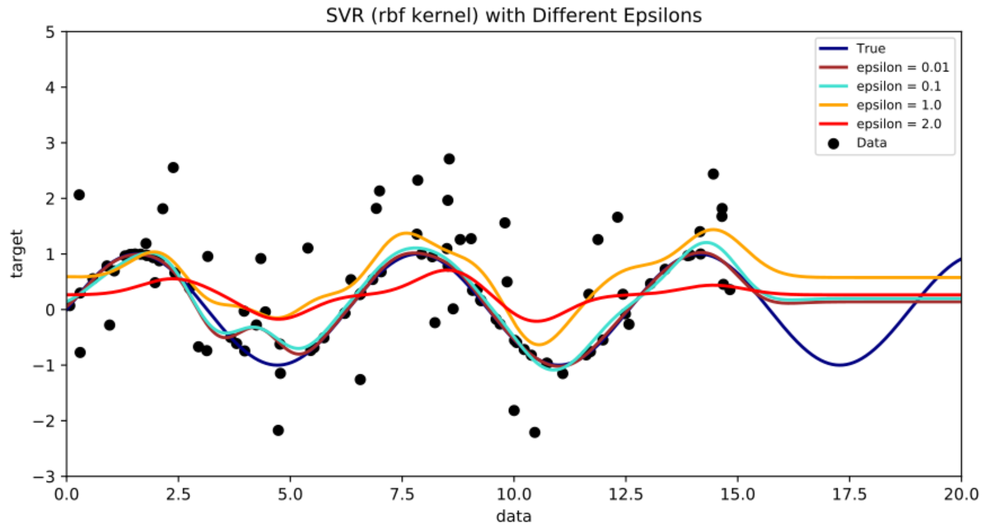
tol : *float, default=1e-3*
Tolerance for stopping criterion.

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.

epsilon : *float, default=0.1*
Epsilon in the epsilon-SVR model. It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. Must be non-negative.



Credit: A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019



Credit: https://www.wikiwand.com/en/Support-vector_machine