Support Vector Machines

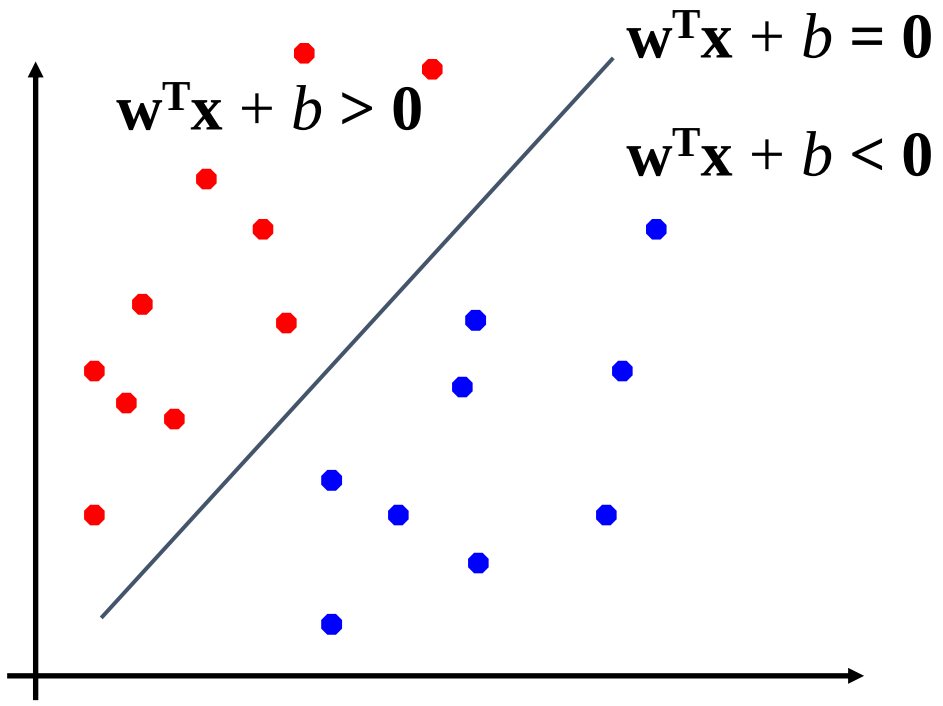
Support Vector Machines

1. **Linear SVM**:
   In its simplest form, an SVM is a linear classifier. It tries to find a hyperplane that best separates two classes of data. The decision boundary is a straight line (in 2D), a plane (in 3D), or a hyperplane in higher dimensions, which is a linear function of the input features. This linear separation works well when the data is linearly separable, meaning that a straight line (or hyperplane in higher dimensions) can cleanly separate the classes.

2. **Non-Linear SVM**:
   Many real-world datasets are not linearly separable. In such cases, SVM can still be used as a non-linear classifier by applying a technique known as the "kernel trick." A kernel function is used to map the original input data into a higher-dimensional feature space where it might become linearly separable. Common kernel functions include the polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. The choice of the kernel function allows SVM to model complex, non-linear decision boundaries.

Linear Separators

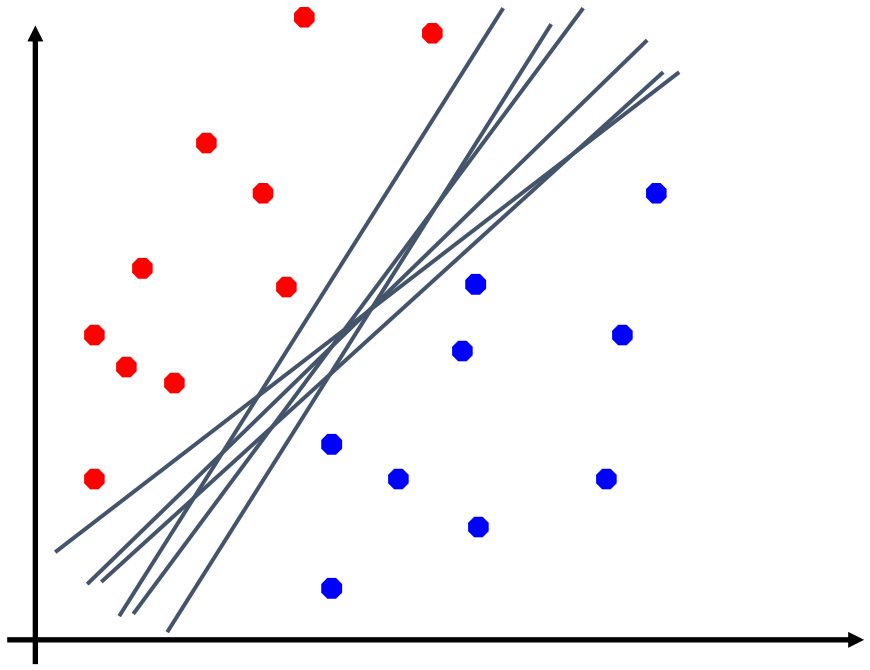
- Binary classification can be viewed as the task of separating classes in feature space:



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

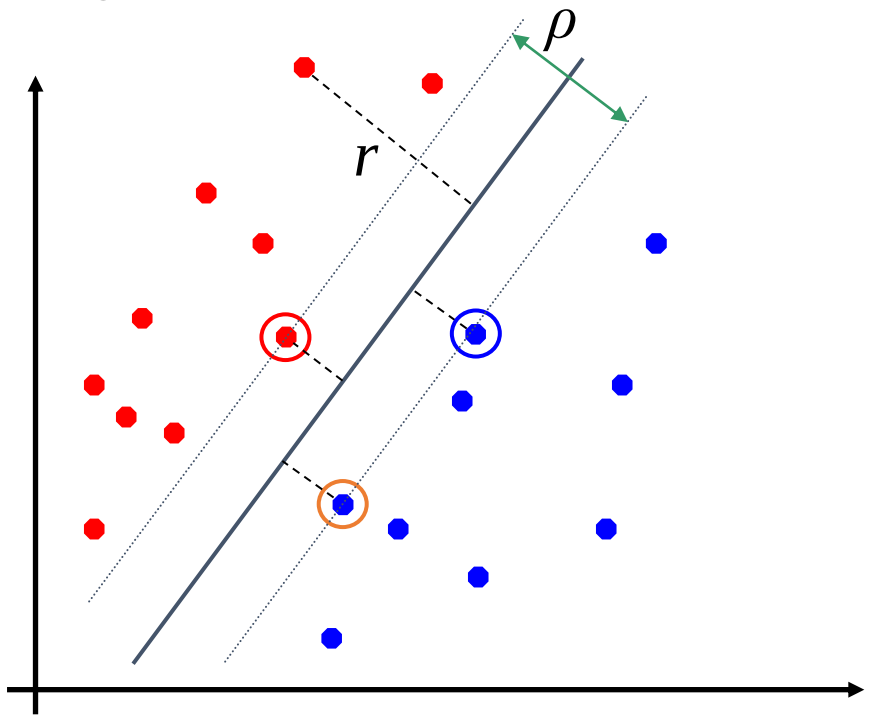
Linear Separators

- Which of the linear separators is optimal?



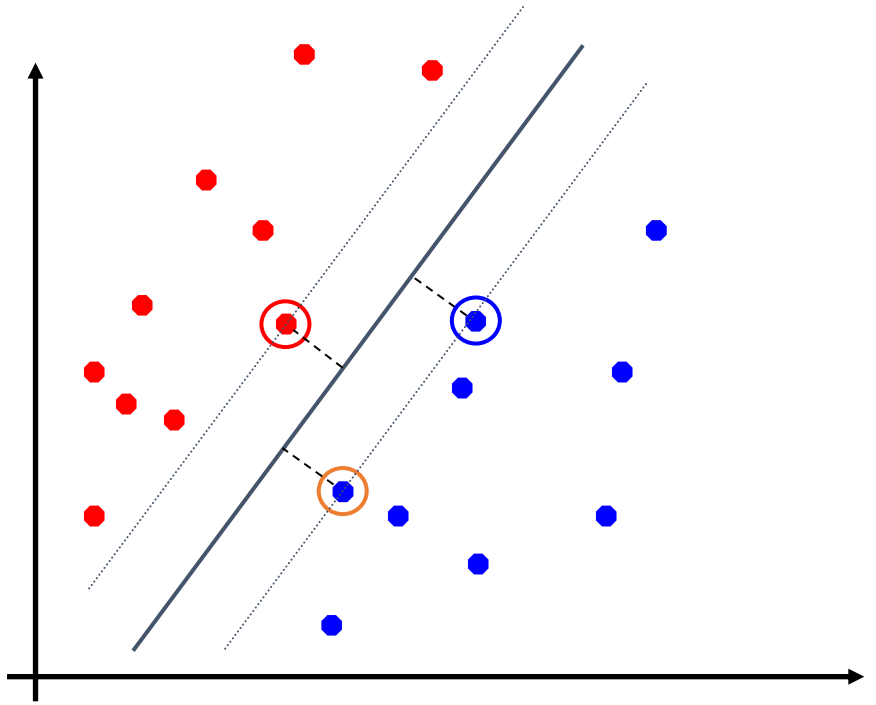
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{|W^T x_i + b|}{\|W\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors.



Maximum Margin Classification

- Maximizing the margin is good according to intuition.
- Implies that only support vectors matter; other training examples are ignorable.



Linear SVM Mathematically

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbf{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ. Then for each training example $(\mathbf{x}_i, y_i)$:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b \leq -\rho/2 \quad \text{if } y_i = -1 \\ \mathbf{w}^T\mathbf{x}_i + b \geq \rho/2 \quad \text{if } y_i = 1 \end{aligned} \quad \Leftrightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq \rho/2$$

- For every support vector $\mathbf{x}_s$ the above inequality is an equality. After rescaling $\mathbf{w}$ and b by $\rho/2$ in the equality, we obtain that distance between each $\mathbf{x}_s$ and the hyperplane is

$$r = \frac{|\boldsymbol{W}^T \boldsymbol{x}_s + b|}{\|\boldsymbol{W}\|}$$

$$r = \frac{y_s(\mathbf{w}^T\mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

- Then the margin can be expressed through (rescaled) $\mathbf{w}$ and b as: $\rho = 2r = \frac{2}{\|\mathbf{w}\|}$

Linear SVMs Mathematically (cont.)

- Then we can formulate *the quadratic optimization problem:*

Find $\mathbf{w}$ and b such that

$\rho = \frac{2}{\|\mathbf{w}\|}$ is maximized

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Which can be reformulated as:

Find $\mathbf{w}$ and b such that

$\mathbf{\Phi}(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$ is minimized

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Solving the Optimization Problem

Find $\mathbf{w}$ and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every inequality constraint in the primal (original) problem:

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

Why Dual Representation?

- The dual representation in SVMs is beneficial for handling non-linear data, improving computational efficiency, and providing insights into the support vectors and decision boundary. It offers flexibility and interpretability, making it a valuable tool in machine learning and pattern recognition applications.

The Optimization Problem Solution

- Given a solution $\alpha_1 \ldots \alpha_n$ to the dual problem, solution to the primal is:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \qquad b = y_k - \sum \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_k \quad \text{for any } \alpha_k > 0$$

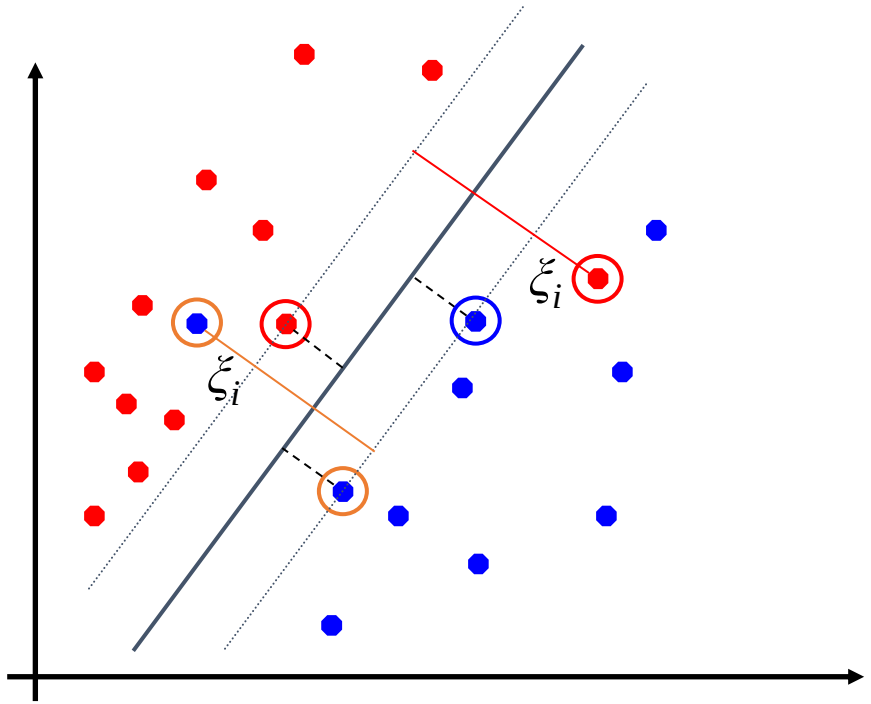
- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a support vector.
- Then the classifying function is (note that we don't need **w** explicitly):

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x} + b$$

- Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x}_i$ – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$ between all training points.

Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



Soft Margin Classification Mathematically

- The old formulation:

Find $\mathbf{w}$ and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find $\mathbf{w}$ and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\sum\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, , $\xi_i \geq 0$

- Parameter *C* can be viewed as a way to control overfitting: it controls the trade-off between maximizing the margin and minimizing the sum of slack variables. A smaller value of C places more emphasis on maximizing the margin and tolerates fewer misclassifications. A larger value of C allows more misclassifications to be tolerated to achieve a more accurate separation.

Soft Margin Classification – Solution

- Dual problem is identical to separable case :

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

- Again, $\mathbf{x}_i$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

$\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x}_i$
$b = y_k(1 - \xi_k) - \Sigma\alpha_i y_i \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_k$ for any k s.t. $\alpha_k > 0$

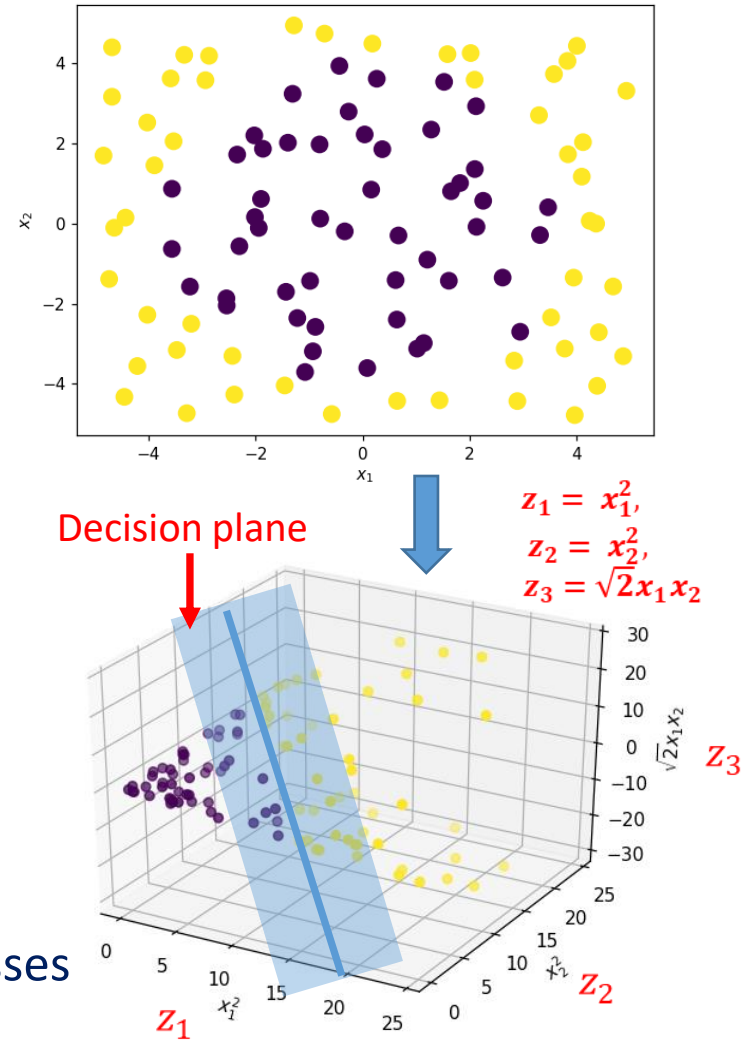
Again, we don't need to compute **w** explicitly for classification:

$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^{\mathrm{T}}\mathbf{x} + b$

Non-Linear SVM

- For SVM to provide non-linear decision boundaries, features are transformed into a higher dimensional space so the data would be linearly separable in that space.
- For example, for data with two features x_1 and x_2 that represent two classes problem shown in the upper figure, no linear decision boundary is available using SVM in the 2D feature space of x_1 and x_2. To solve that, the data is transformed into 3-D feature space z_1, z_2, and z_3 where:

$$z_1 = x_1^2,$$
$$z_2 = x_2^2,$$
$$z_3 = \sqrt{2}x_1x_2$$



In this case we can find a 2-D hyperplane that can separate the 2 classes in the 3-D space as shown in the figure at the bottom.

Non-Linear SVM

- From the dual representation of the SVM problem we need to find:

$$\max_{\alpha^{(n)}} \quad \sum_{n=1}^{N} \alpha^{(n)} - \frac{1}{2} \sum_{n,m}^{N} \alpha^{(n)} \alpha^{(m)} y^{(n)} y^{(m)} x^{(n)^T} x^{(m)} \, , n, m = 1,2, \dots, N$$

In order to support nonlinear decision boundaries, SVM uses $x \xrightarrow{mapping} \Phi(x)$, where $\Phi(x)$ is nonlinear function of x, then need to solve

$$\max_{\alpha^{(n)}} \quad \sum_{n=1}^{N} \alpha^{(n)} - \frac{1}{2} \sum_{n,m}^{N} \alpha^{(n)} \alpha^{(m)} y^{(n)} y^{(m)} \Phi(x)^{(n)^T} \Phi(x)^{(m)} \, , n, m = 1,2$$

However, finding the dot product $\Phi(x)^T \Phi(x)$ in the higher dimensional space is computationally expensive. But fortunately Mercer's Theorem provide a method to find this dot product without involving calculations in the higher dimensional space. This offers the solution that is known as the kernel trick using the kernel function K. So the SVM problem is reduced to:

$$\max_{\alpha^{(n)}} \quad \sum_{n=1}^{N} \alpha^{(n)} - \frac{1}{2} \sum_{n,m}^{N} \alpha^{(n)} \alpha^{(m)} y^{(n)} y^{(m)} K(x^{(n)}, x^{(m)}) \, , n, m = 1,2, \dots,$$

Non-Linear SVM

- For example, for

$$\mathrm{x} = [x_1 \;\; x_2]^T, \text{ and } \; \Phi(\mathrm{x}) = [x_1^2 \;\; \sqrt{2}x_1x_2 \;\; x_2^2]^T$$

the dot product $\Phi(x)^T\Phi(x) = x_1^4 + 2x_1^2x_2^2 + x_2^4$ can be calculated directedly using

$$\mathrm{K}(\mathrm{x},\mathrm{x}) = (\mathrm{x}^T\mathrm{x})^2 = \; (x_1^2 + x_2^2)^2 = \; x_1^4 + 2x_1^2x_2^2 + x_2^4 = \Phi(x)^T\Phi(x)$$

- There are some valid kernel functions K(a, b) that can be used (should follow Mercer's Theorem to be considered as valid kernels), where $\gamma, coef0$, and d are parameters to specify for certain kernels (in sklearn)
  - Polynomials: $\mathrm{K}(\mathrm{a},\mathrm{b}) = (\gamma \mathrm{a}^T\mathrm{b} + coef0)^d, \; where \; d > 0 \; is \; the \; degree \; of \; the \; polynomial$
  - Hyperbolic Tangent: $\mathrm{K}(\mathrm{a},\mathrm{b}) = \tanh(\gamma \mathrm{a}^T\mathrm{b} + coef0)$
  - Radial Basis Functions (Gaussian Kernel): $\mathrm{K}(\mathrm{a},\mathrm{b}) = \exp\left(-\frac{||a-b||^2}{\sigma^2}\right) = \exp(-\gamma||a-b||^2)$
- To assign input x to the +ve class C_1, then $\sum_{n=1}^{N_s} \alpha^{(n)} y^{(n)} \mathrm{K}\left(x^{(n)}, x\right) + b > 0$

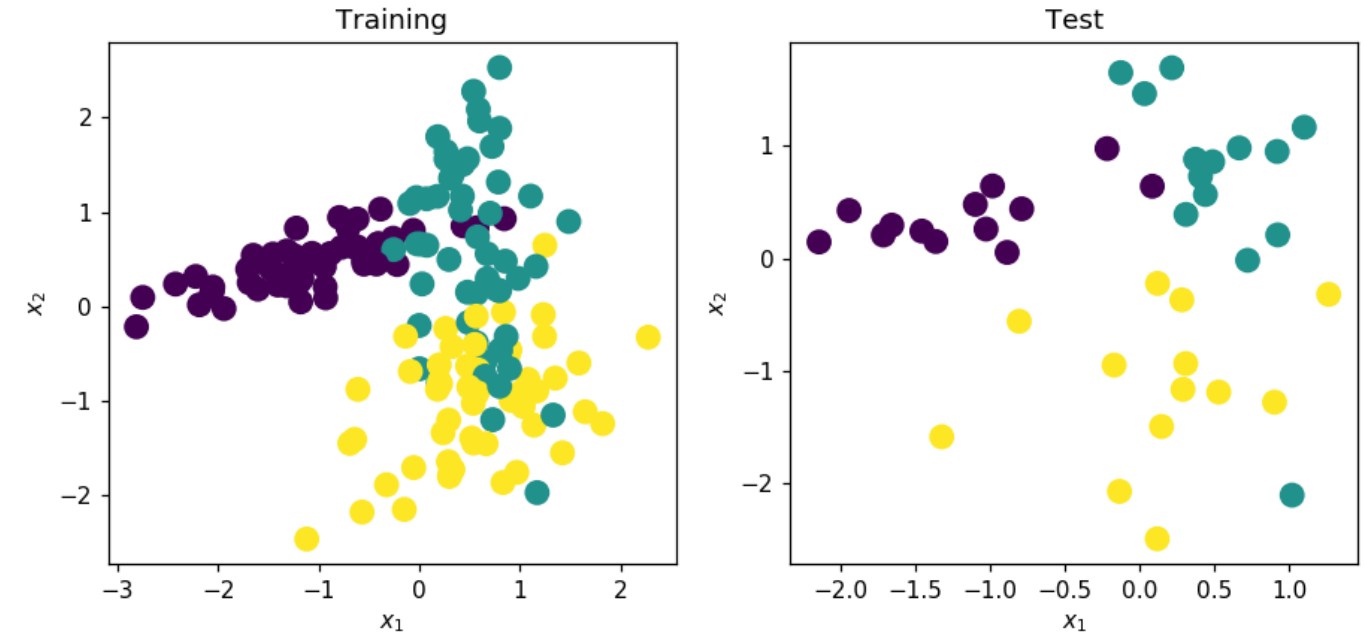
  to assign input x to the -ve class C_2, then $\sum_{n=1}^{N_s} \alpha^{(n)} y^{(n)} \mathrm{K}\left(x^{(n)}, x\right) + b < 0$

  where, $\mathrm{x}^{(\mathrm{n})}$ are the support vectors, Ns is the number of support vectors.

SVM Classifiers in sklearn, Example 1

LinearSVC:

class sklearn.svm.**LinearSVC**(*penalty='l2', loss='squared_hinge', *, dual=True, 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*)




```
from sklearn.svm import LinearSVC
C1 = LinearSVC()
C1.fit(X_train, y_train)
y_p1 = C1.predict(X_test)
```

```
print(accuracy_score(y_test,y_p1))
```

```
0.9
```

SVM Classifiers in sklearn, Example 1

```
grid_sample_dist = 0.01 # sampling period
x1_min, x1_max = X_train[:, 0].min() - .5, X_train[:, 0].max() + .5
x2_min, x2_max = X_train[:, 1].min() - .5, X_train[:, 1].max() + .5
x1grid, x2grid = np.meshgrid(np.arange(x1_min, x1_max, grid_sample_dist),
                             np.arange(x2_min, x2_max, grid_sample_dist))
Z = C1.predict(np.c_[x1grid.ravel(), x2grid.ravel()])
Z = Z.reshape(x1grid.shape)
```

```
plt.figure(4)
plt.pcolormesh(x1grid, x2grid, Z, alpha = 0.4)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.scatter(X_test[:, 0], X_test[:, 1], marker='o', edgecolor = 'k',
            c=y_test,s=100)
plt.show()
```

```
print(C1.classes_) # class label
```

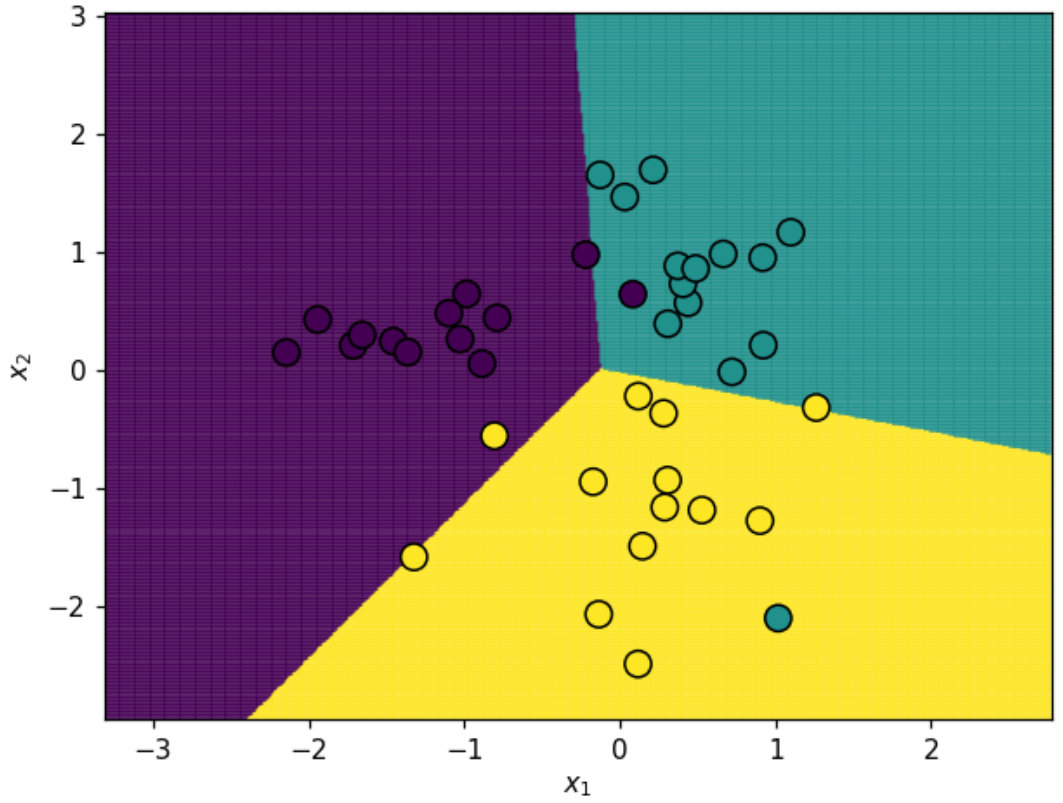
```
[0 1 2]
```

```
#One versus Rest (ovr) multi-class approach
print(C1.coef_) # weights, number of classes X number of features (3x2)
```

```
[[-1.41683256  0.45481615]
 [ 0.58016913  0.56742815]
 [ 0.23303045 -0.80505455]]
```

```
print(C1.intercept_) # intercept b with dimensions number of classes (one for each class)
```

```
[-0.65440001 -0.40093139 -0.42995281]
```



SVM Classifiers in sklearn, Example 1

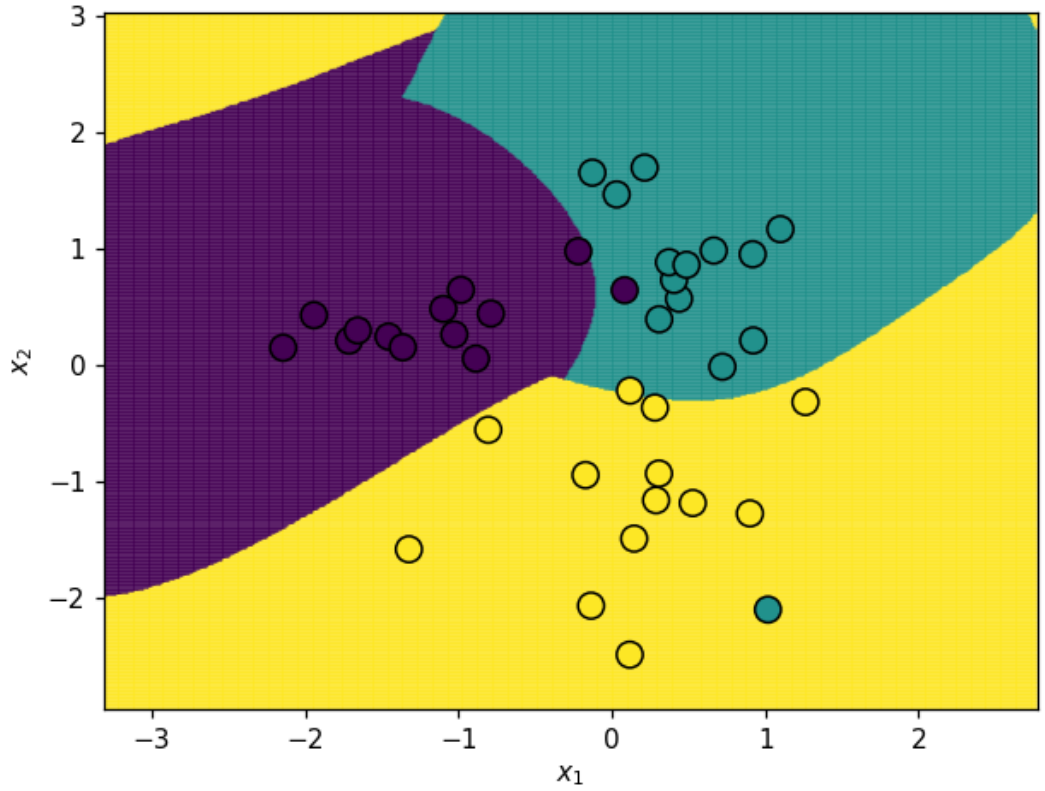
SVC (non-linear SVM with kernel selection)

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*)

```
from sklearn.svm import SVC
C2 = SVC(kernel = 'rbf', gamma = 0.1)
C2.fit(X_train, y_train)
y_p2 = C2.predict(X_test)
```

```
print(accuracy_score(y_test,y_p2))
```

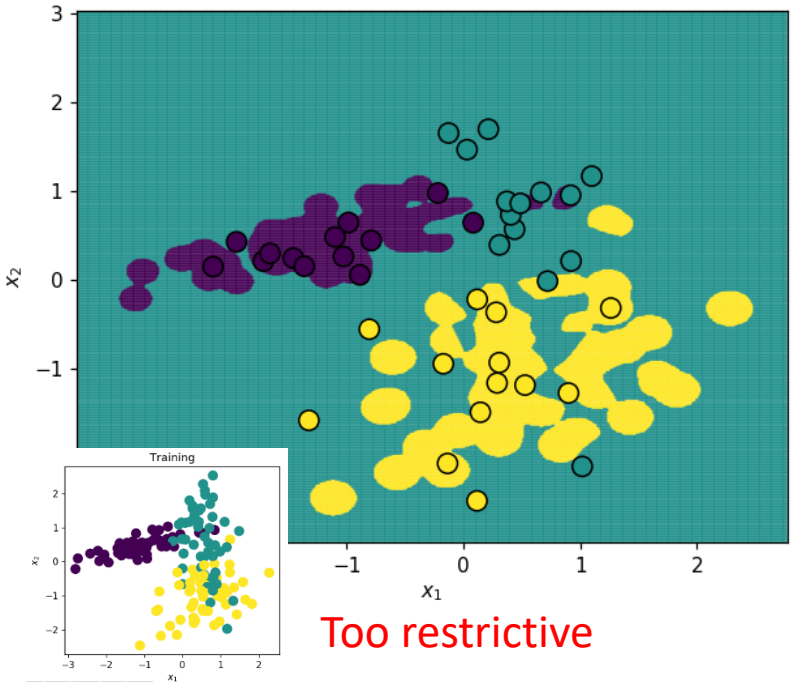
```
0.925
```



SVM Classifiers in sklearn, Example 1

rbf, gamma = 100

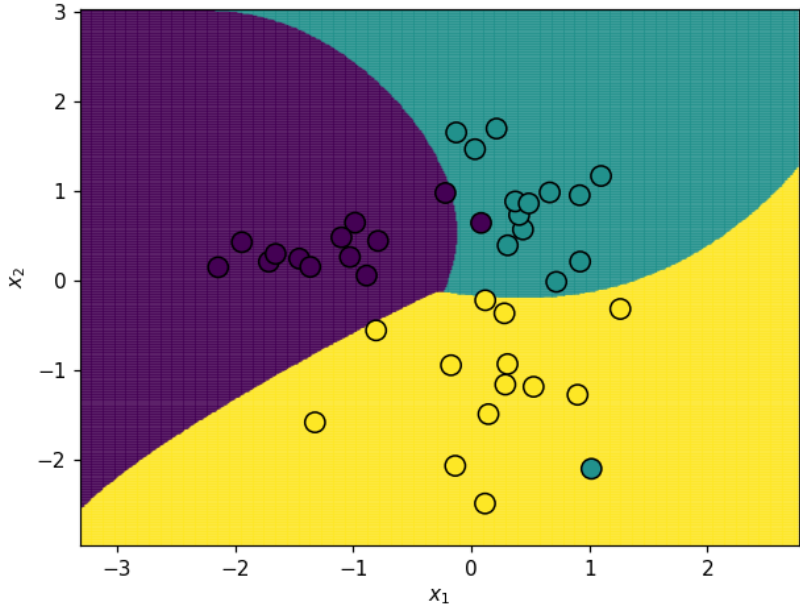
Accuracy = 0.65



Too restrictive

rbf, gamma = 0.1

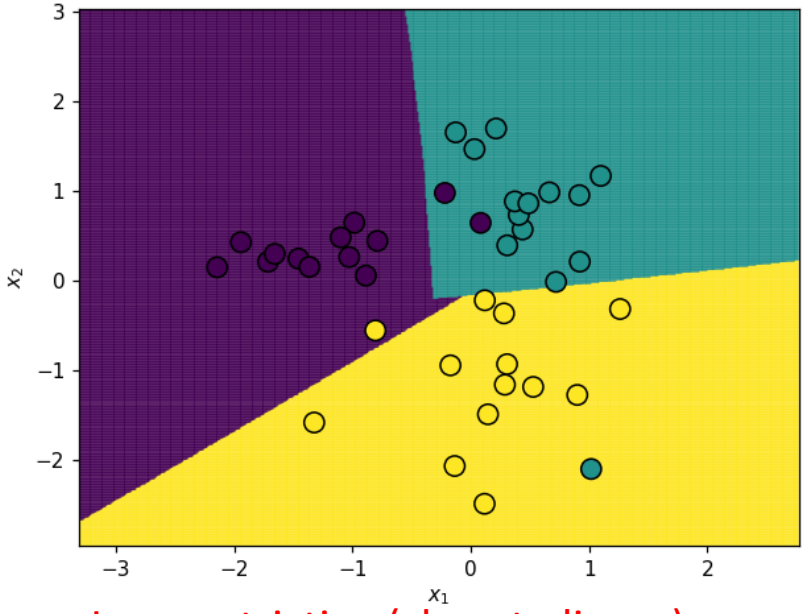
Accuracy = 0.95



Acceptable

rbf, gamma = 0.01

Accuracy = 0.90

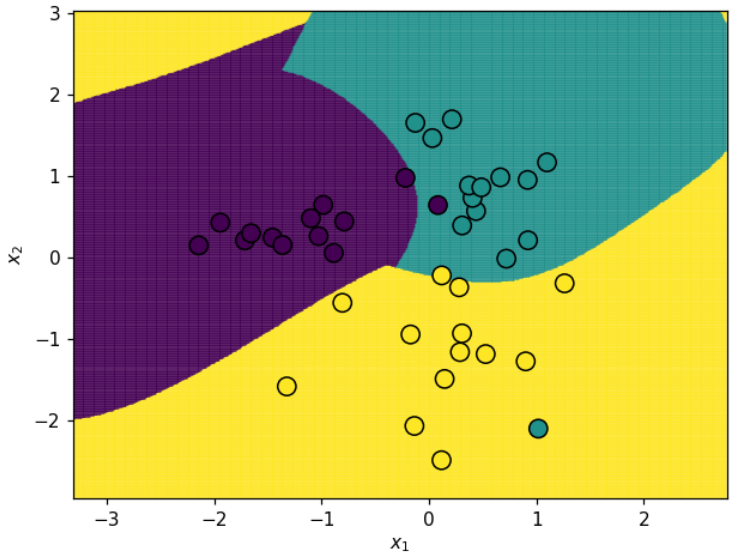


Less restrictive (close to linear)

SVM Classifiers in sklearn, Example 1

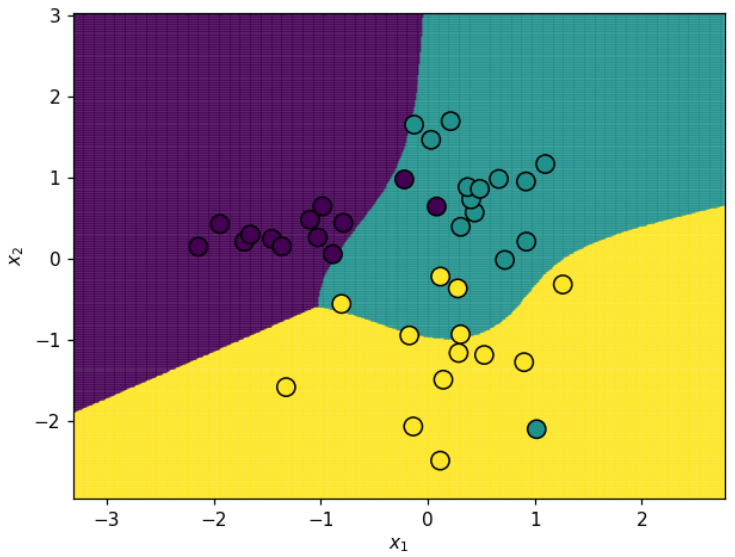
```
from sklearn.svm import SVC
C2 = SVC(kernel = 'rbf')
C2.fit(X_train, y_train)
y_p2 = C2.predict(X_test)
```

Accuracy = 0.93



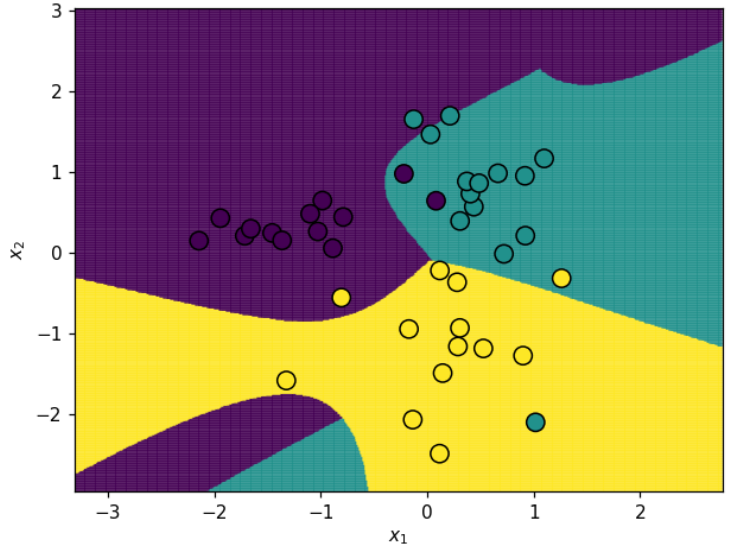

```
from sklearn.svm import SVC
C2 = SVC(kernel = 'poly')
C2.fit(X_train, y_train)
y_p2 = C2.predict(X_test)
```

Accuracy = 0.82

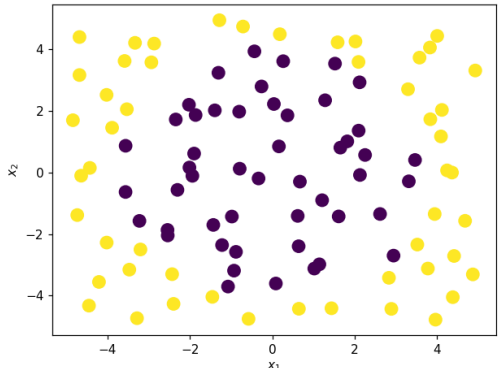


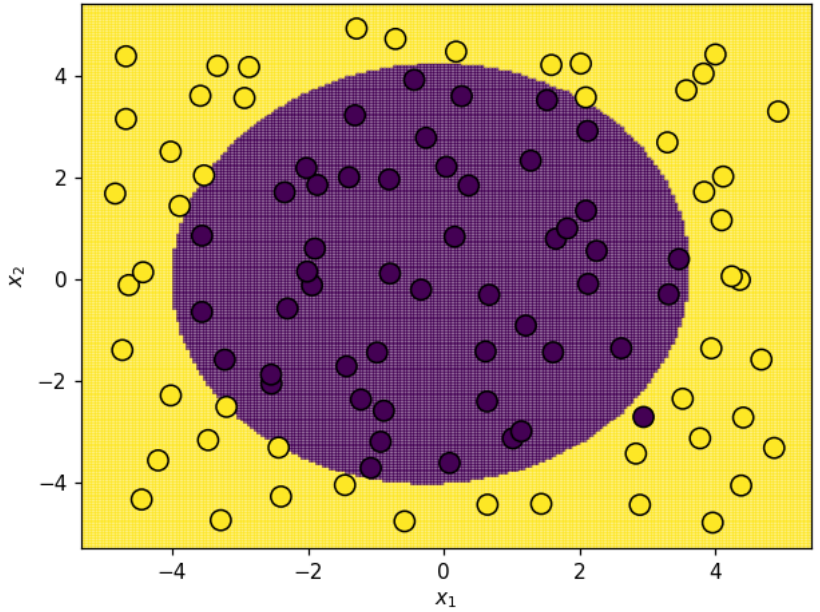

```
from sklearn.svm import SVC
C2 = SVC(kernel = 'sigmoid')
C2.fit(X_train, y_train)
y_p2 = C2.predict(X_test)
```

Accuracy = 0.82

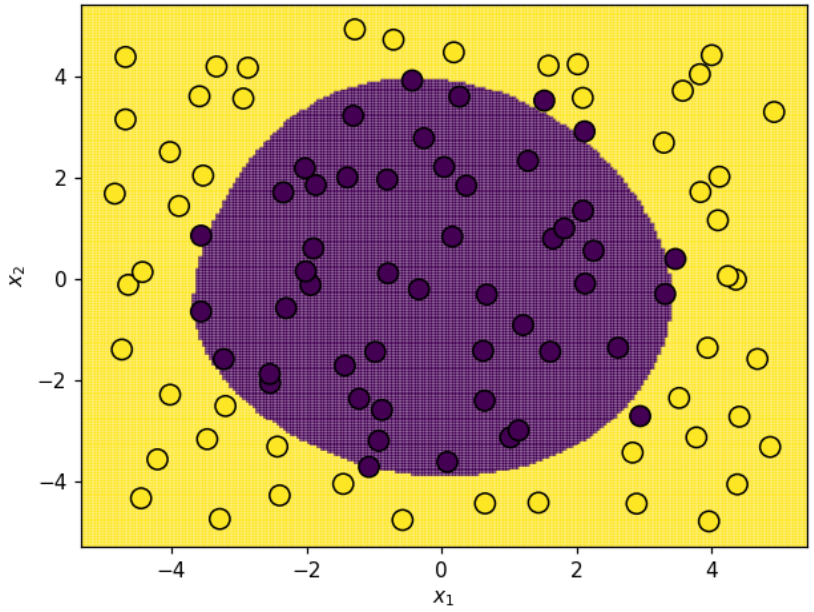


SVM Classifiers in sklearn, Example 2

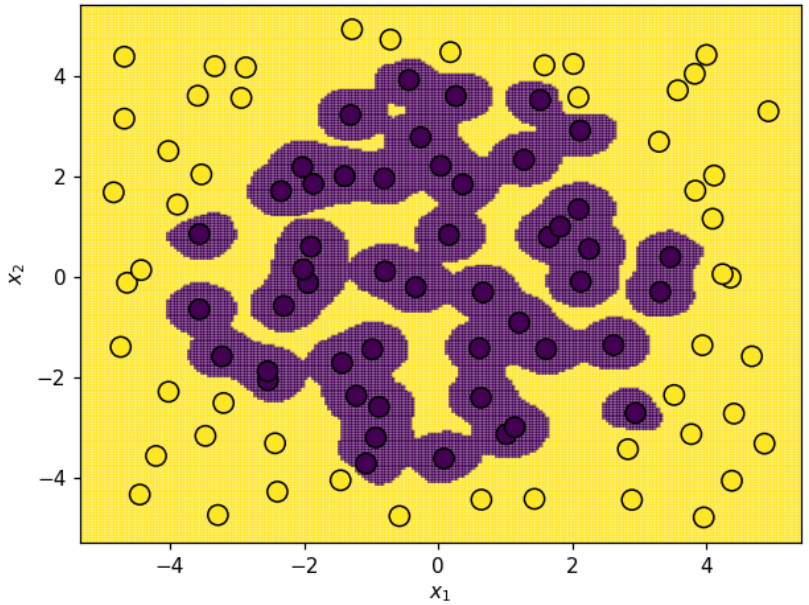




rbf, gamma = 0.01



rbf, gamma = 0.1



rbf, gamma = 10