Birla Institute of Technology and Science Pilani, Hyderabad Campus

22.11.2024

BITS F464: Machine Learning (1st Sem 2024-25)

INSTANCE AND KERNEL BASED LEARNING:k-NN, SVM

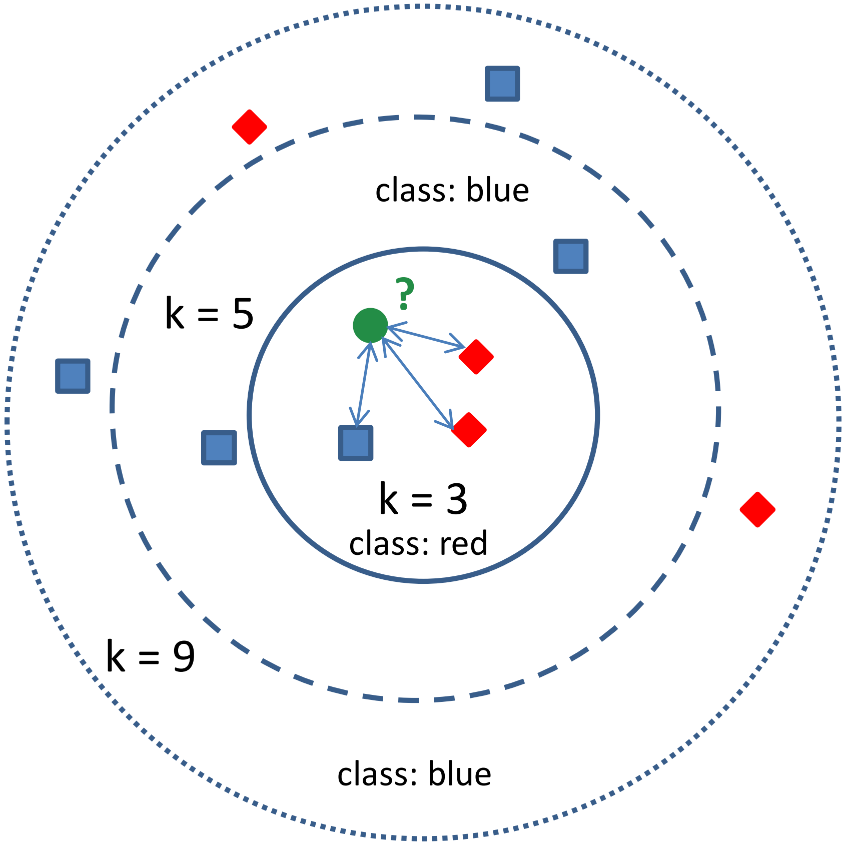
Chittaranjan Hota, Sr. Professor
Dept. of Computer Sc. and Information Systems
hota@hyderabad.bits-pilani.ac.in

Instance-based learning: k-NN

- Why is it called Instance-based?
  - Predictions are made based on specific instances or examples from the training data.
  - Instead of learning explicit relationships between features, it learns from memorization of training data.
- Called Lazy learning. Why?
  - Because it postpones generalization until prediction/ classification time.
- Where is it useful over Symbolic or Connectionist learning you have read?
  - Where the underlying relationships between features and labels are complex OR where the dataset is dynamic and constantly evolving.
- They are robust to Concept drift. Why?
  - As they directly adapt to new examples, they are not affected by data distribution over time or change in the characteristics of the target variable.
- Based on **Similarity metrics** (Euclidian, Manhattan, Cosine, etc…)

k-NN: k-Nearest Neighbor Algorithm

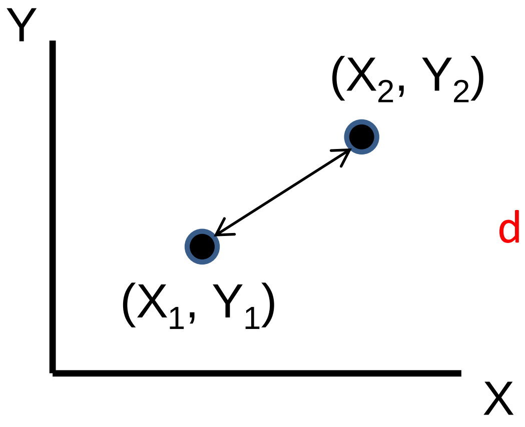
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data                load dataset
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)                        k=5

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```



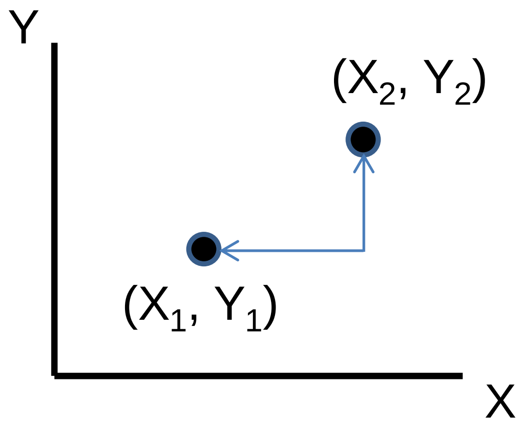
Discrete valued target fun: voting, and
Real-valued target fun: taking mean

Applications: Optical Character Recognition (OCR), Credit Scoring, Loan Approval.

k-NN Distance Metrics: Common Choices



distance: hyper-parameter



Euclidean: $\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

Manhattan: $| X_2 - X_1 | + | Y_2 - Y_1 |$

```
from sklearn.metrics.pairwise import
euclidean_distances
point1 = [[1, 2]]
point2 = [[4, 6]]
distance = euclidean_distances(point1, point2)
print("Euclidean Distance:", distance[0][0])
```

```
import numpy as np
point1 = np.array([1, 2])
point2 = np.array([4, 6])

distance = np.sum(np.abs
                  (point1 - point2))

print("Manhattan Dist:", distance)
```

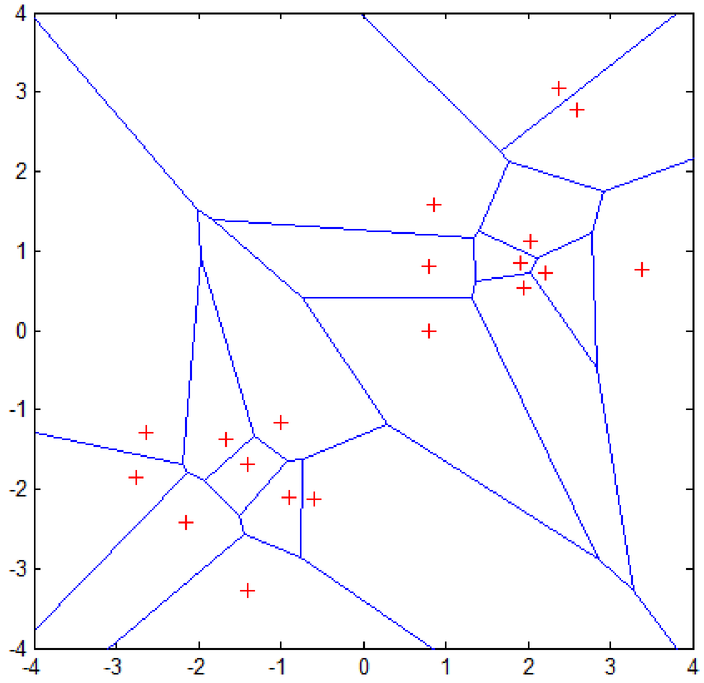
manually

Decision boundaries: Voronoi-like dia.

$\hat{f}(x_q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$ Where, δ(a, b) = 1 if a==b, zero (0) otherwise.

Properties:

- All possible points within a sample's Voronoi cell are the nearest neighboring points for that sample.
- For any sample, the nearest sample is determined by the closest Voronoi cell edge
- When you perform a KNN search for a given point, you're effectively partitioning the space around each data point into regions based on distance.



(1-Nearest Neighbor Algorithm)

Distance-weighted k-NN: Refinement

- Weight the contribution of each of the k-neighbors according to their distance to the query point, x_q, giving greater weight to closer neighbors.

$$\hat{f}(x_q) = \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \frac{1}{d(x_i, x_q)^2} \delta(v, f(x_i))$$

Where, 'd' is the distance between x_i and x_q.

- To accommodate the case where the query point x_q, exactly matches one of the training instances 'x_i' and the denominator $d(x_i, x_q)^2$ will therefore be zero, and hence we assign:

$$\hat{f}(x_q) = f(x_i)$$

- If there are several such training examples, we assign the majority classification among them.
- Closer neighbors have a greater influence on the decision, while farther neighbors have less influence. Sensitive to outliers.

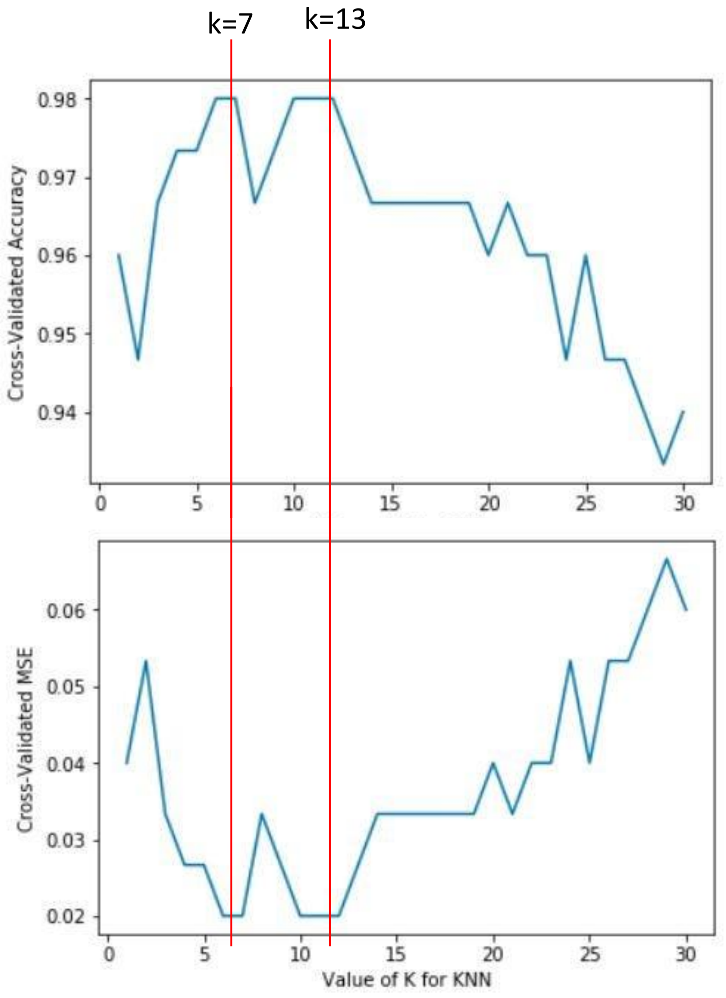
How to choose a right value of 'k'?

If 'k' = small value:

- Can lead to sensitive decision boundaries when data-distribution is Non-linear
- Can be sensitive to noise and outliers, hence leading to Overfitting
- Hence, Low bias and High variance.

If 'k' = large value:

- Can lead to smoother decision boundary as influence of individual neighbor is diluted.
- Reduce the impact of noise and outliers.
- Can lead to Under-fitting in complex cases.
- Hence, High bias and Low variance.

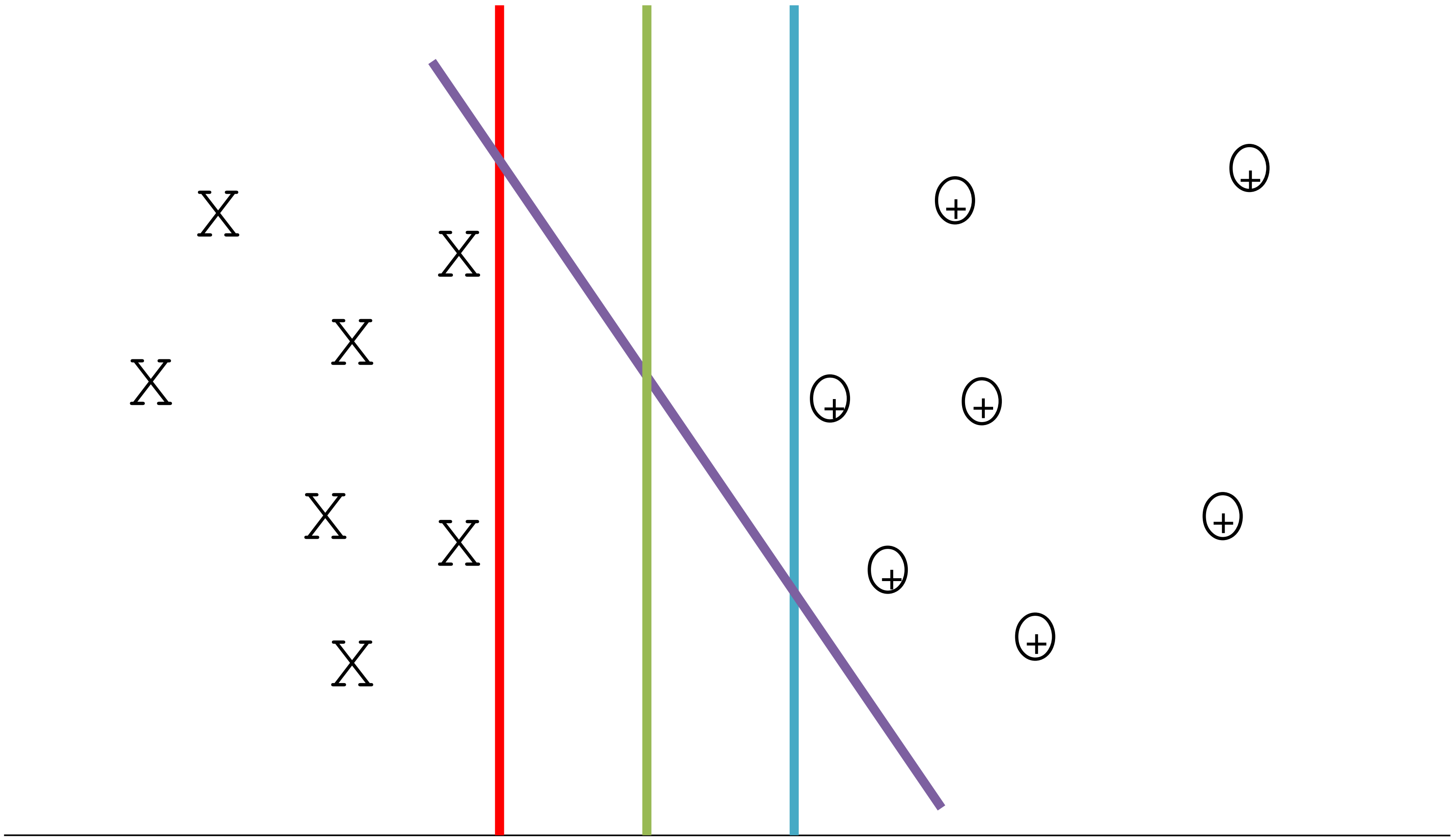


Curse of Dimensionality: Distance computation complexity, Sparse data, Curse of Proximity

Kernel-based Learning: Support Vector Machine

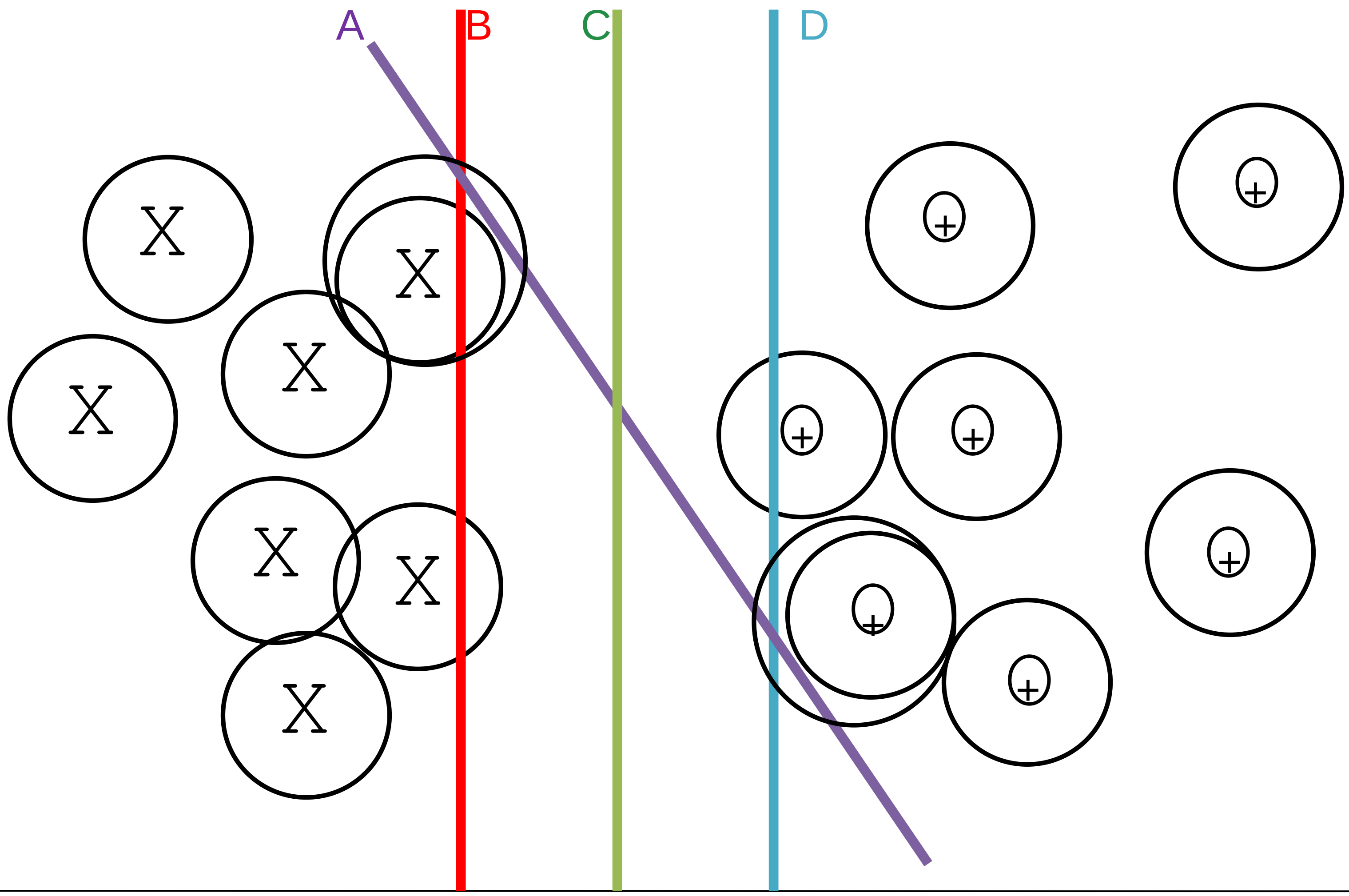
- Transform the input data into a higher-dimensional feature space using a kernel function.
- In this higher-dimensional space, the data may become more linearly separable, allowing linear algorithms to do the classification well.
- SVM finds an optimal hyperplane that separates the classes in this transformed feature space.
- The optimal hyperplane is the one that Maximizes the Margin between the two classes of data points.
- Model complexity depends on the number of training samples, not on the dimensionality of the kernel space.
- As it can handle high dimensional vector spaces with ease, it makes feature selection less critical.

SVM: Intuition behind choice of surface



Which one is the best separator out of these 4?

Some Noise in the Input Samples

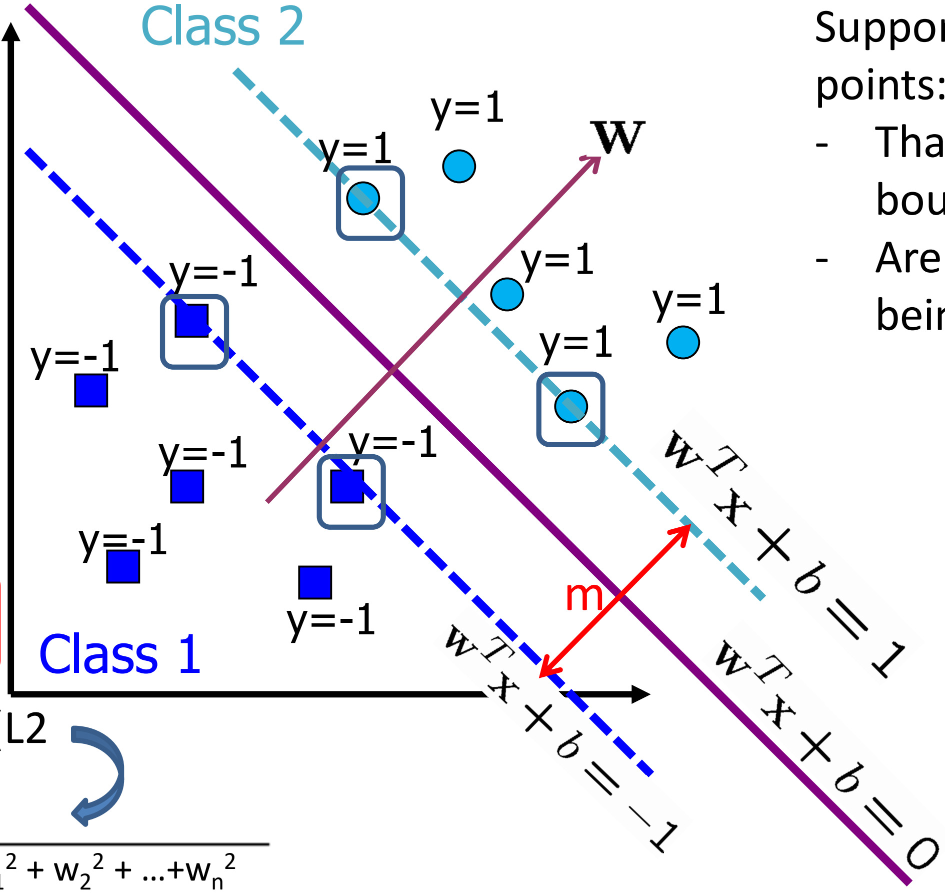


Which ones are better now?

Which one is good ow?

Finding the Decision Boundary: Another Ex.

- Let $\{x_1, ..., x_n\}$ be the data set and let $y_i \in \{1,-1\}$ be the class label of x_i



Support Vectors are the data points:

- That lie on or within the margin boundary.
- Are misclassified or are close to being misclassified.

$$\boxed{m = \frac{2}{||\mathbf{w}||}}$$

Euclidean (L2 Norm)

$||w|| = \sqrt{w_1^2 + w_2^2 + ... + w_n^2}$

Hence, the decision boundary should be as far away as possible from the data of both the classes → Maximum Margin Classifier (m)

Support vectors are the data points that lie closest to the decision boundary (hyperplane) and have the largest influence on determining the position and orientation of the boundary.

Maximum Margin Classifier (m)

For the marginal plane, we can write the equation as:

$$w^T x + b = 0$$

For the positive hyperplane the equation will be:

$$w^T x + b \geq 0 \text{ when } y_n = +1$$

And for negative hyperplane:

$$w^T x + b < 0 \text{ when } y_n = -1$$

Marginal Distance?

$$w^T x_1 + b - w^T x_2 + b = 1 - -1$$

$$w^T (x_1 - x_2) = 2 \quad \text{———} \quad \text{(Eq.1)}$$

As w^T is a vector which has a direction, divide the equation (1) by ||w||:

$$\frac{w^T}{\|w\|}(x_1 - x_2) = \frac{2}{\|w\|}$$

$$\text{ie,} (x_1 - x_2) = \frac{2}{\|w\|}$$

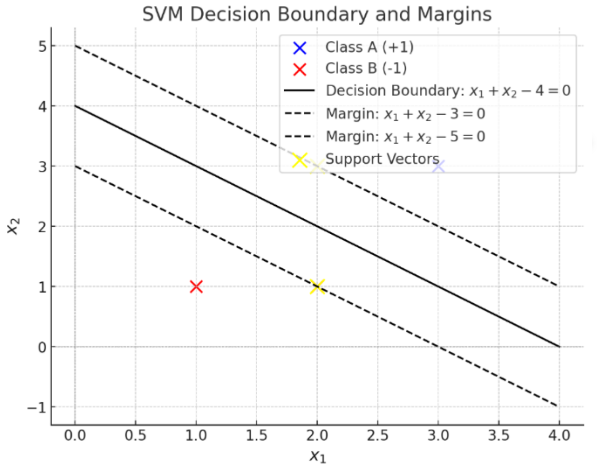
Hence, the goal of SVM is:

$$\boxed{\max \frac{2}{\|w\|}} \longrightarrow \text{Regularizer}$$

$$\text{subject to} \qquad y_n(w^T x + b) \geq 1$$

$$\boxed{y_n \begin{cases} +1 & w^T x + b \geq 1 \\ -1 & w^T x + b \leq -1 \end{cases}}$$

Convex Optimization Prob



- Let there be, Class A (+1): (2,3), (3,3) and Class
- We want to find the linear decision boundary t between these two classes.
- Margin=$\frac{2}{\|w\|}$
- To maximize the margin, we minimize the $\|w\|^2$ subject to $y_i(w \cdot x_i + b)$ >= 1, where y is the label of i-th point.
- We solve using Lagrange multiplier:
  - Define the Lagrangian: $L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i$
  - Derivate with respect to w: $\frac{\partial L}{\partial w} = \mathrm{w} - \sum_{i}^{n} \alpha_i y_i x_i =$
  - Derivate with respect to b: $\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0$

2(1)+3(1)+b >= 1 → b >= -4
3(1)+3(1)+b >= 1 → b >= -5
1(1)+1(1)+b <= -1→ b <= -3
2(1)+1(1)+b <= -1→ b <= -4
Assume normalized direction vector w = (1,1). Combine the results for b.

- Compute the solution: using the constraints and solving the optimization problem: w = (1, 1), b = -4.
- The decision boundary becomes: wx + b = 0 → $x_1 + x_2 - 4 = 0$
- The margin: $\frac{2}{\|w\|} = \frac{2}{\sqrt{1^2+1^2}} = \sqrt{2}$
- **Support Vectors**: (2,3) and (2,1).

Soft margin SVM

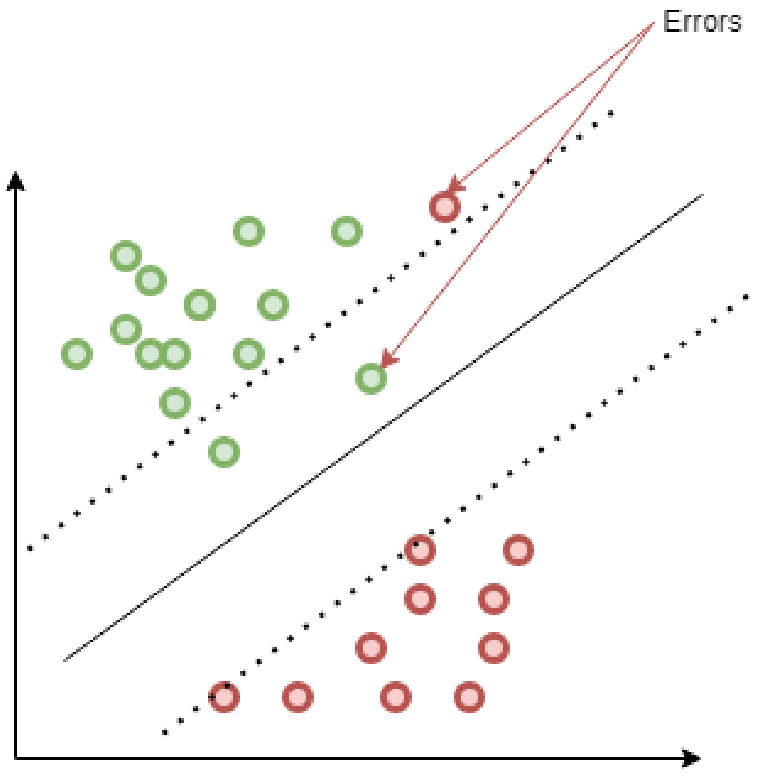
- For non-linearly separable data, slack variables (ξ_i) are introduced to allow some misclassification:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

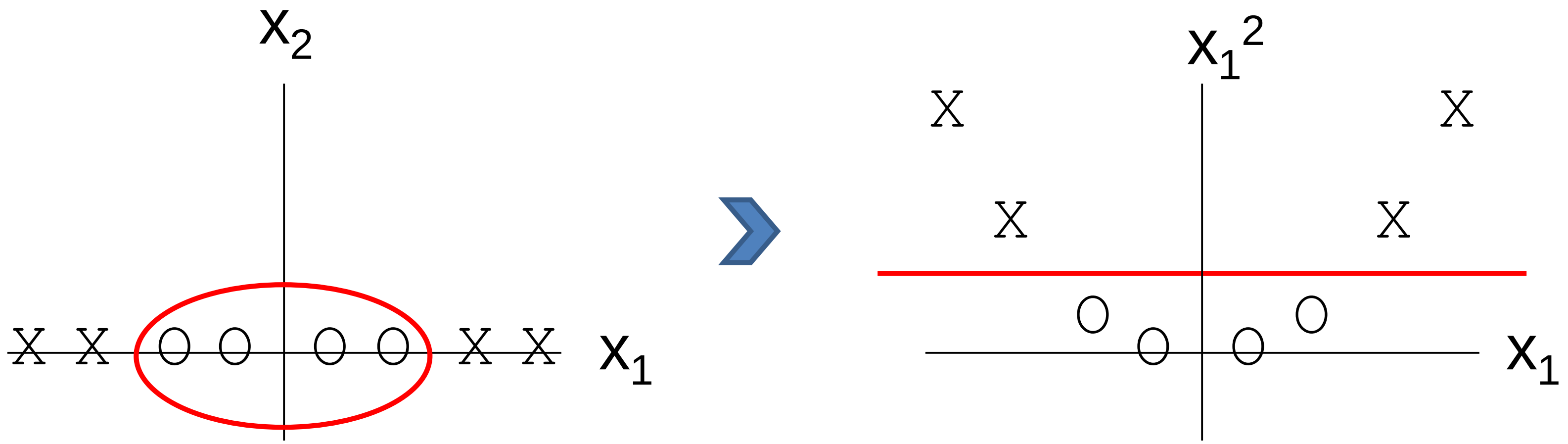
- Including the number of errors in the training and the sum of the value of error, the optimization term will be:

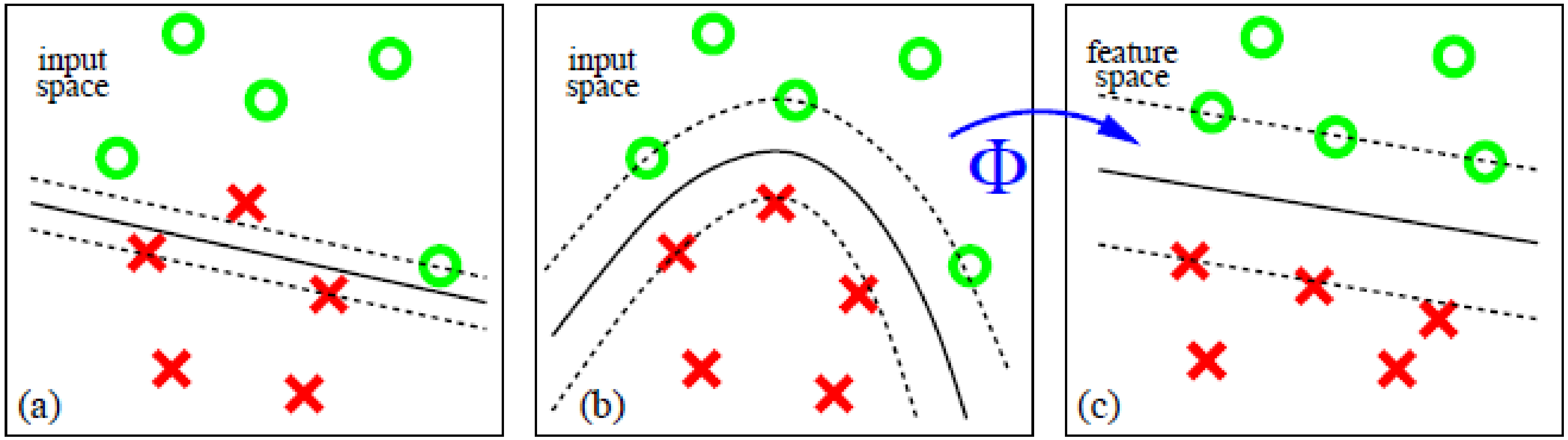
$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

where ‘C’ is a regularization parameter controlling the trade-off between margin width and classification error.



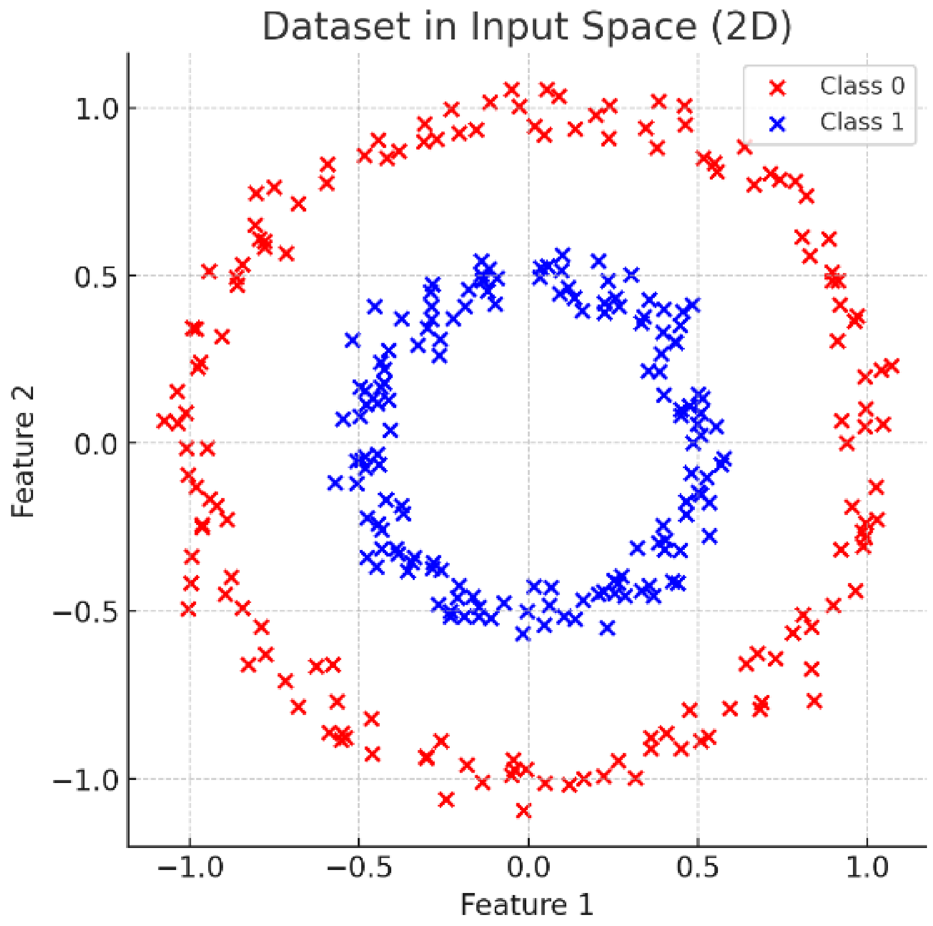
Problems with linear SVM: How to Solve?

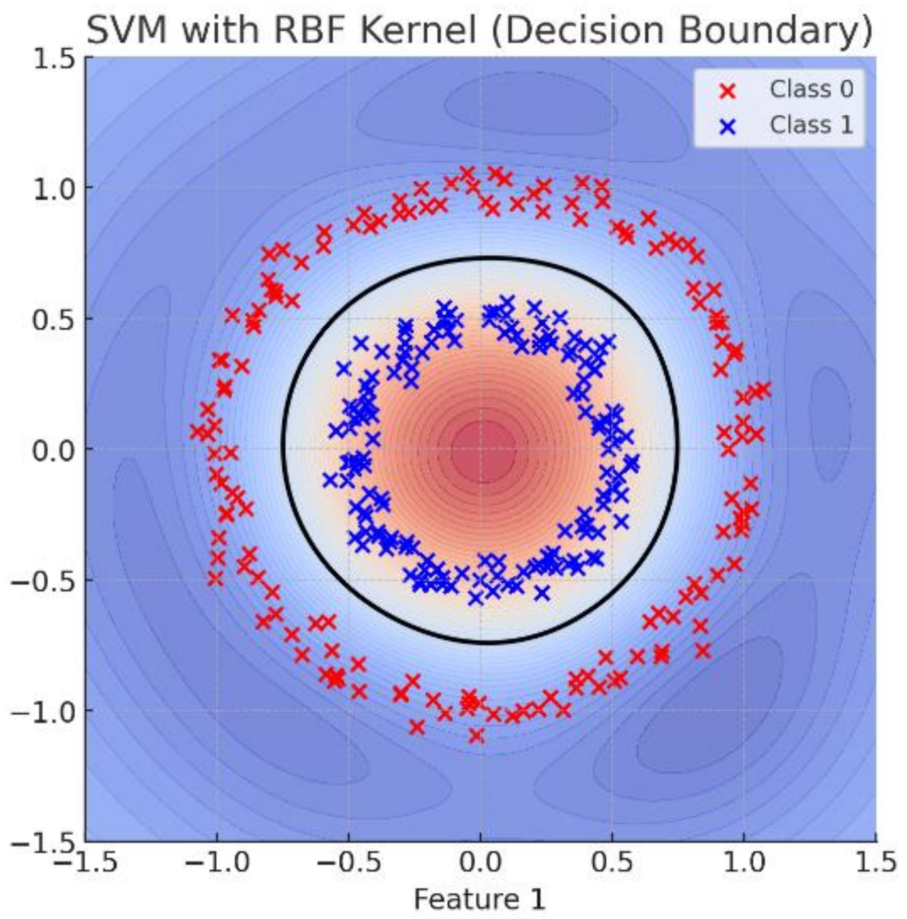




What if the decision function is not a linear?

Kernel Trick



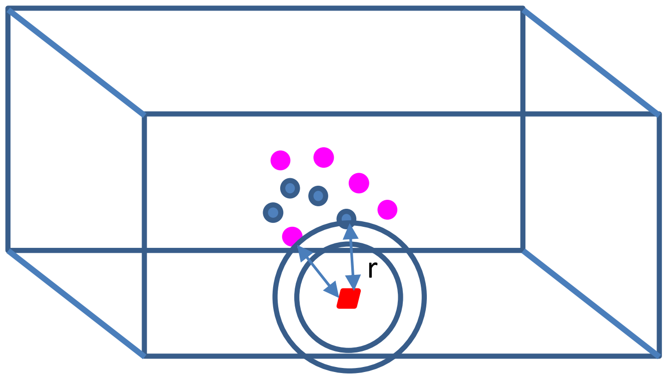


A mathematical technique that allows SVMs to classify data that is not linearly separable in the original input space by implicitly mapping it to a higher-dimensional feature space, without explicitly performing the transformation.

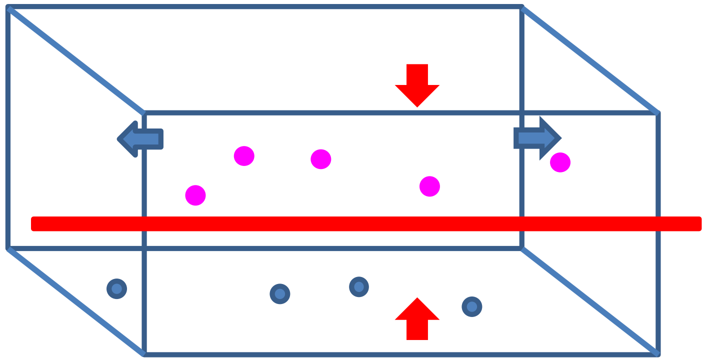
What is a Radial Basis Function?

- The radial basis function is a mathematical function that takes a real-valued input and gives a real-valued output based on the distance between the input value projected in space from an imaginary fixed point placed elsewhere.

The distance between the center and any data point positioned in the boundary of the circle is called the radius.



(Imaginary point)



After calculating the radius, we need to pass this value inside a mathematical function (RBF) that will return a real value. The returned value will be the transformed magnitude of a particular data point.

Radial Basis Kernel: Similar to Gaussian

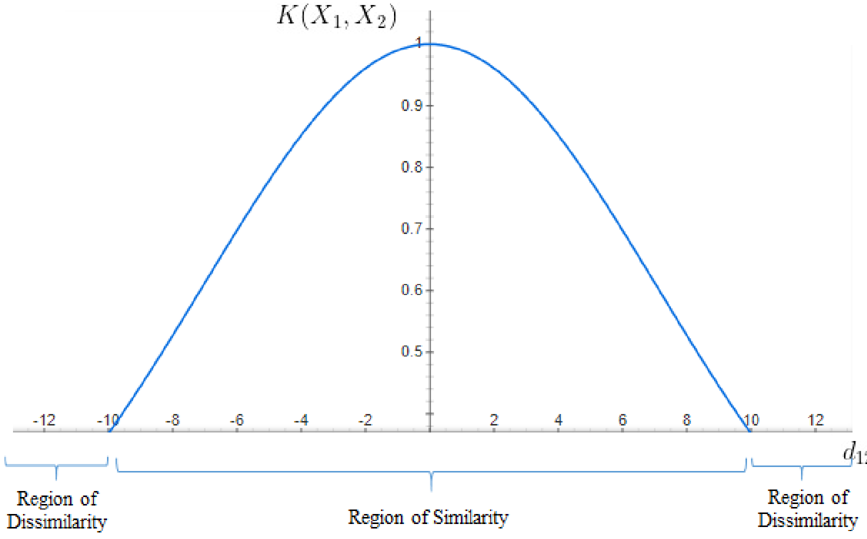
- The RBF kernel function for two points X_1 and X_2 computes the similarity or how close they are to each other. This kernel can be mathematically represented as follows:

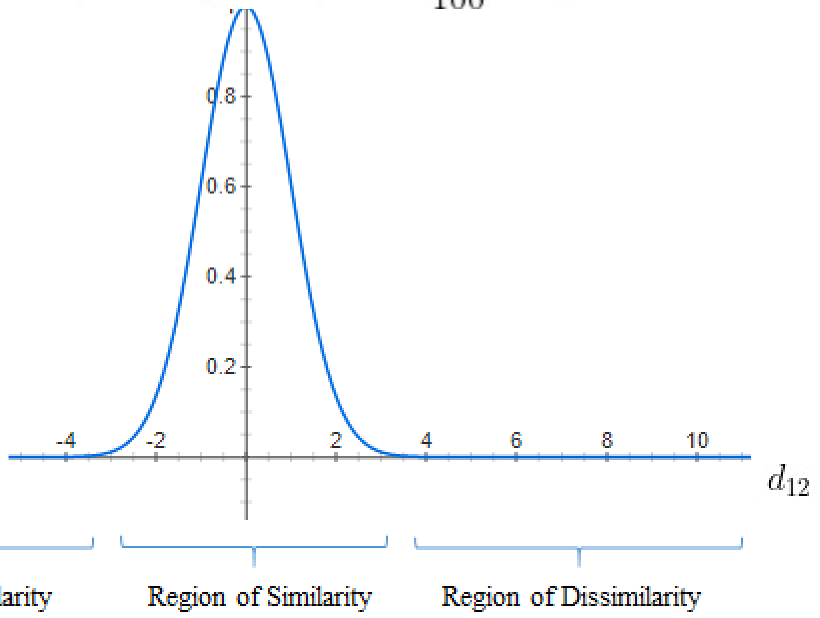
$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

1. ‘σ’ is the variance and our hyper-parameter
2. $||X_1 - X_2||$ is the Euclidean (L_2-norm) Distance between two points X_1 and X_2

$\sigma = 10$

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{100})$$





$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2})$$

$\sigma = 1$

Image source: https://towardsdatascience.com/

SVM with RBF Kernel Example

Let there be points:
Class A: (1,1), (2,2)
Class B: (1,2), (2,3)

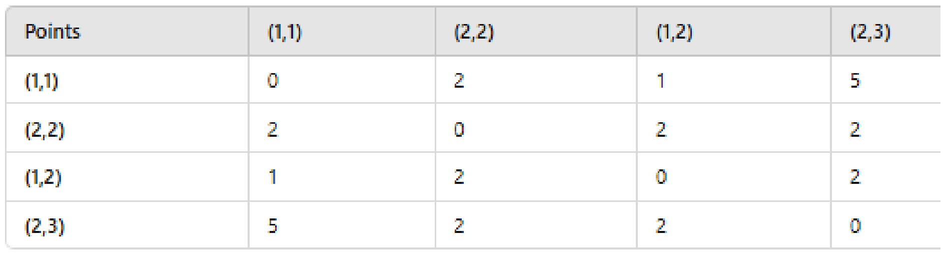
And RBF kernel is:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \qquad \gamma = 0.5.$$

Compute kernel matrix K that captures pairwise similarities between all points:

Step 1: Calculate squared Euclidean distances between all points:

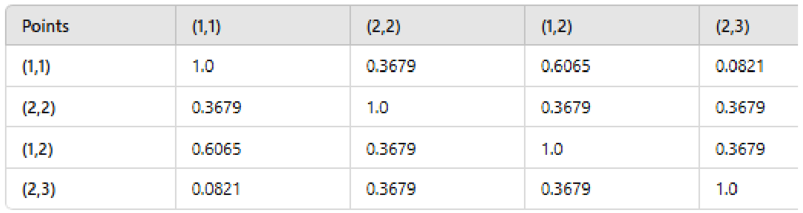
$$\|x_i - x_j\|^2 = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2$$



Points	(1,1)	(2,2)	(1,2)	(2,3)
(1,1)	0	2	1	5
(2,2)	2	0	2	2
(1,2)	1	2	0	2
(2,3)	5	2	2	0

Kernel Matrix ($K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$):

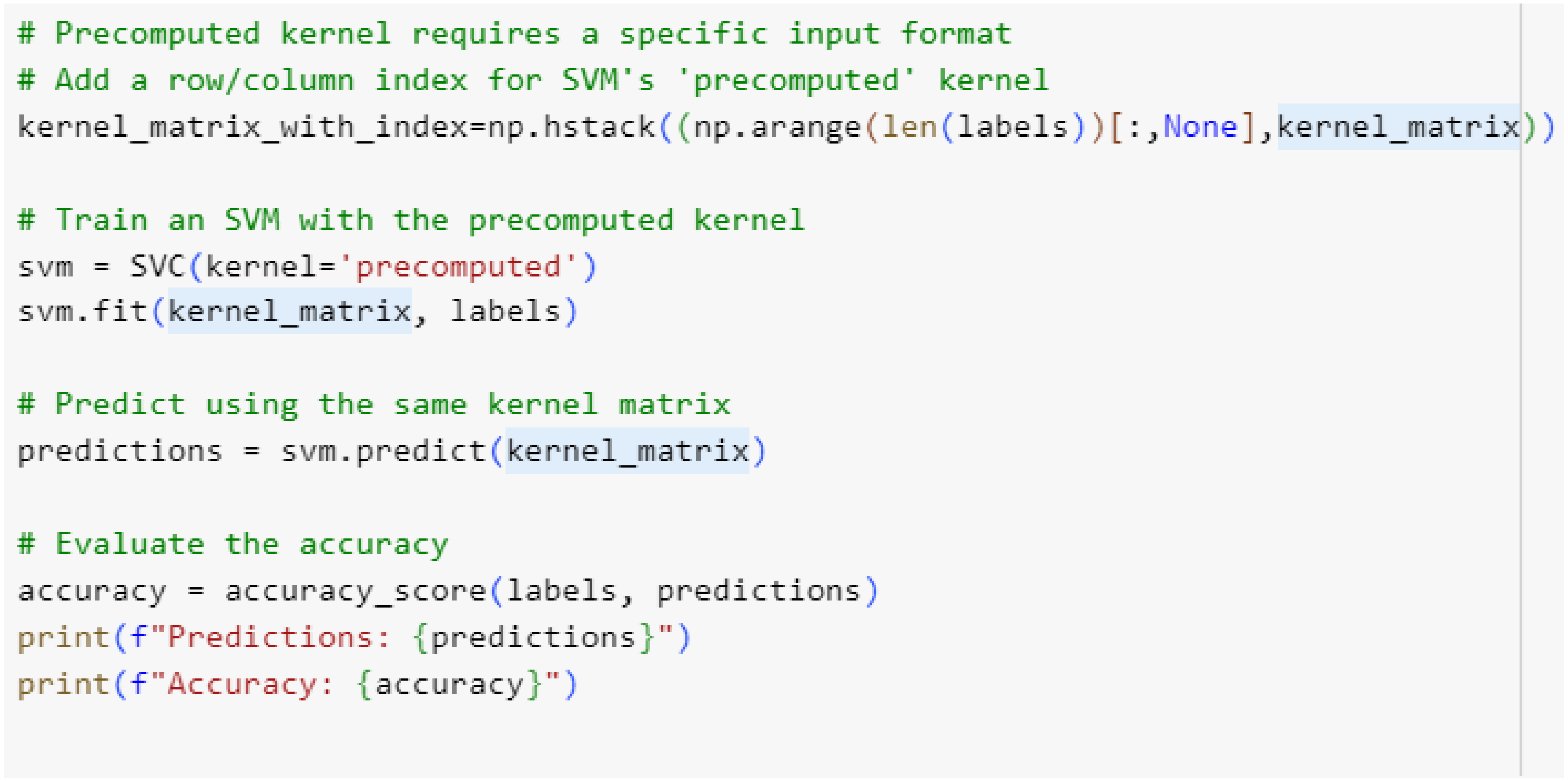
Use $\gamma = 0.5$:



Points	(1,1)	(2,2)	(1,2)	(2,3)
(1,1)	1.0	0.3679	0.6065	0.0821
(2,2)	0.3679	1.0	0.3679	0.3679
(1,2)	0.6065	0.3679	1.0	0.3679
(2,3)	0.0821	0.3679	0.3679	1.0

The kernel matrix implicitly represents the data in a higher-dimensional space, where we can now use a linear classifier like an SVM.

SVM Classification with Sklearn



```python
# Precomputed kernel requires a specific input format
# Add a row/column index for SVM's 'precomputed' kernel
kernel_matrix_with_index=np.hstack((np.arange(len(labels))[:,None],kernel_matrix))

# Train an SVM with the precomputed kernel
svm = SVC(kernel='precomputed')
svm.fit(kernel_matrix, labels)

# Predict using the same kernel matrix
predictions = svm.predict(kernel_matrix)

# Evaluate the accuracy
accuracy = accuracy_score(labels, predictions)
print(f"Predictions: {predictions}")
print(f"Accuracy: {accuracy}")
```

```
Predictions: [ 1  1 -1 -1]
Accuracy: 1.0
```

Thank You!