UNIVERSITY AT ALBANY
STATE UNIVERSITY OF NEW YORK

CSI 436/536 (Spring 2025)

Machine Learning

Lecture 15: Kernel Methods

Chong Liu

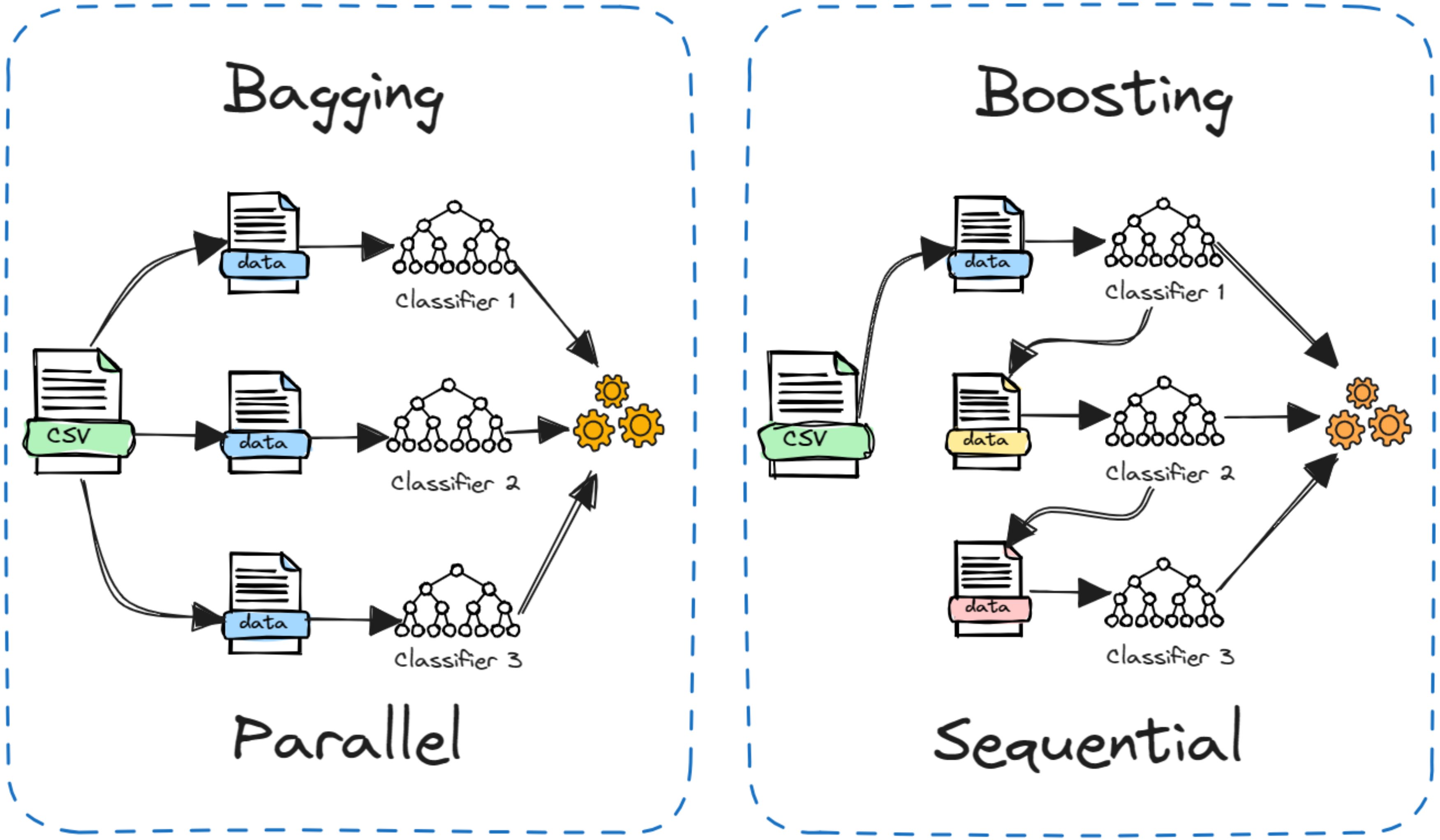
Department of Computer Science

Apr 7, 2025

Recap: Last lecture

- Risk-Decomposition
  - "Optimization error", "generalization error" and "approximation error"
- Ensemble Learning methods
  - Bagging and Random Forest
  - Boosting
- Key message: "Weak Learner" ➔ "Strong learner"
  - You can convert a "simple" ML algorithm (e.g., a Decision Stump) into a much stronger ML algorithm.

Recap: Bagging and Boosting



https://www.datacamp.com/tutorial/what-bagging-in-machine-learning-a-guide-with-examples

Recap: Three main approaches for expanding the hypothesis class (systematically minimizing the approx. error)

- Boosting and Bagging (Ensemble learning)
  - Combine many weak learners (e.g., decision trees with depth 3) into a strong learner

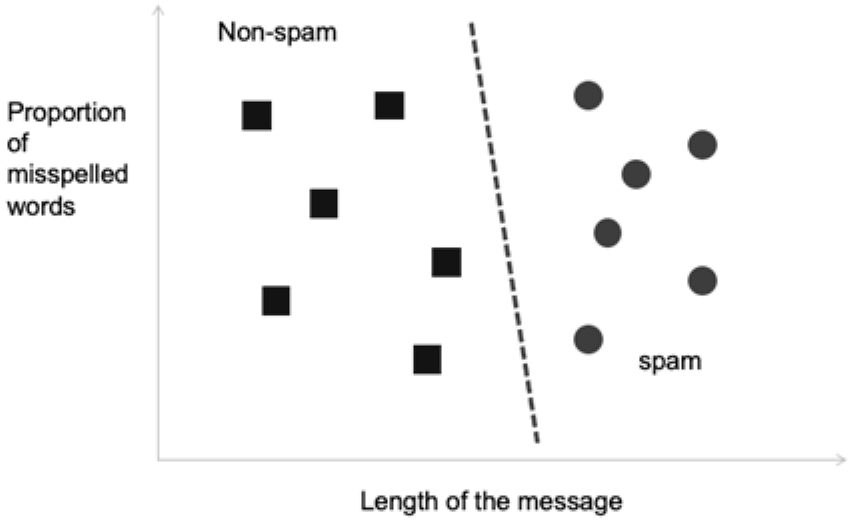
- Kernel methods (lift features to higher-dimensional space)
  - e.g., adding polynomial expansion, add interaction terms
  - Other nonlinear transformation of the original features

- Deep Learning
  - Train large neural networks using SGD
  - Learn feature representation and classification jointly.

Today

- Feature expansion
- Kernel methods

Recap: Linear classifiers



- How does it make prediction?

$$h_{w,b}(x) = \mathrm{sign}(x^T w + b)$$

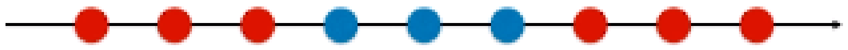
- Shape of the decision boundary: a decision line

- Parameters: the weight vector

- How to train a linear classifier?
  - Perceptron
  - GD / SGD with Logistic loss, hinge loss or other surrogate losses
  - Regularization

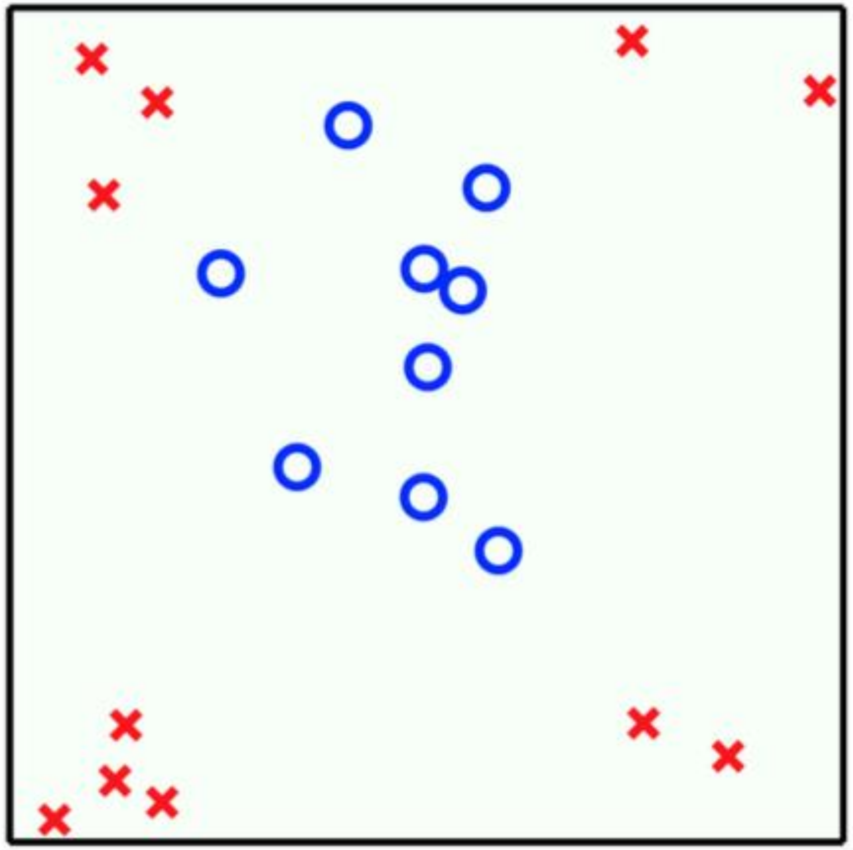
Linear classifier is limited. The best linear classifier might not be good.

1-dimensional example

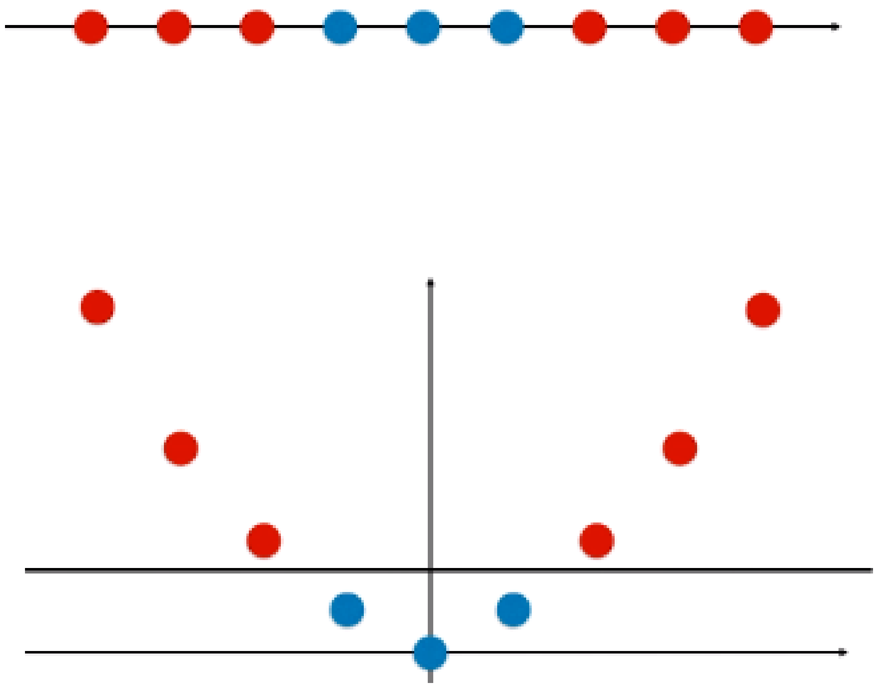


Linear classifier is limited. The best linear classifier might not be good.

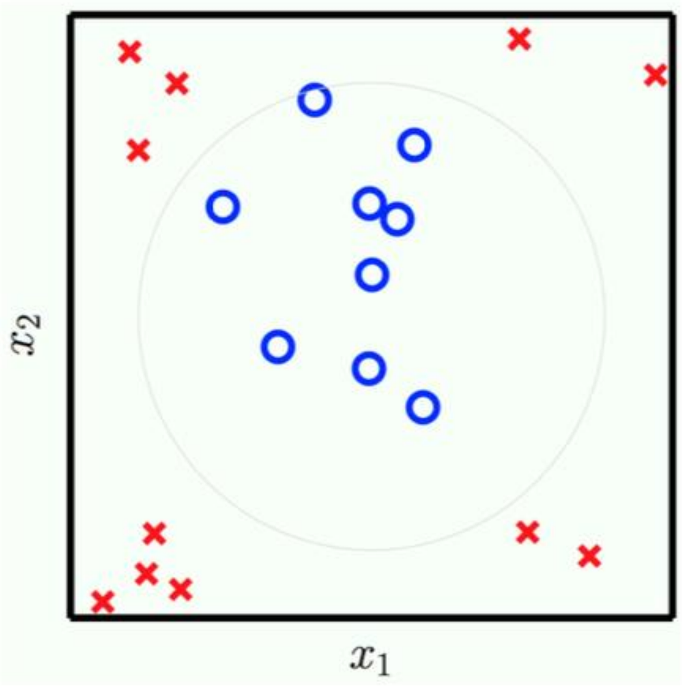
What is the prominent issue for linear classifiers in this example?

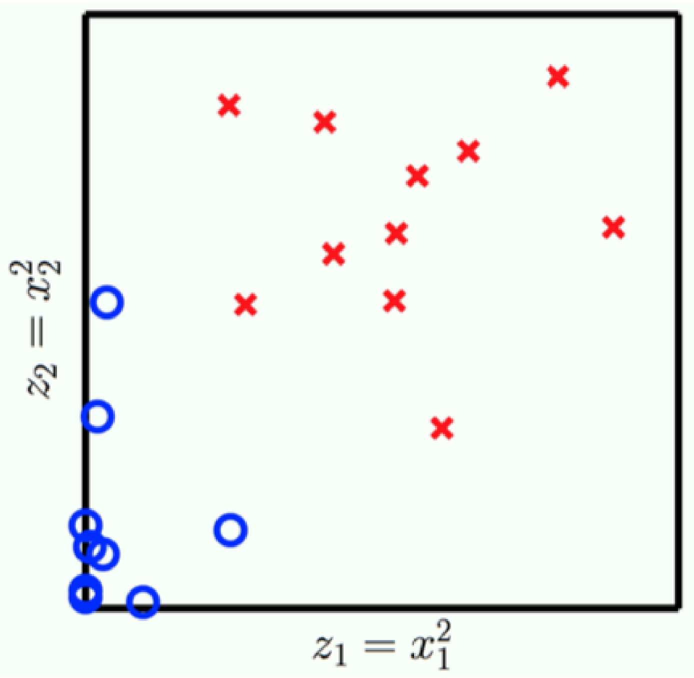


Idea: Transform the feature so that it becomes linearly separable!



Idea: Transform the feature so that it becomes linearly separable!

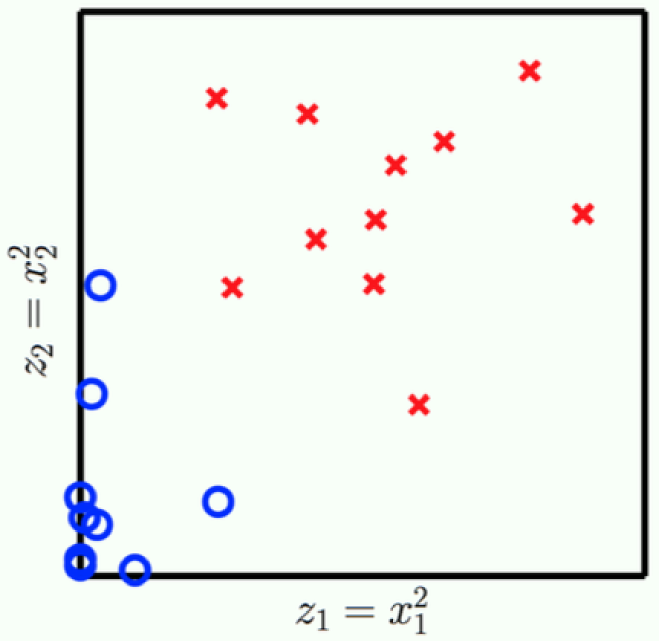




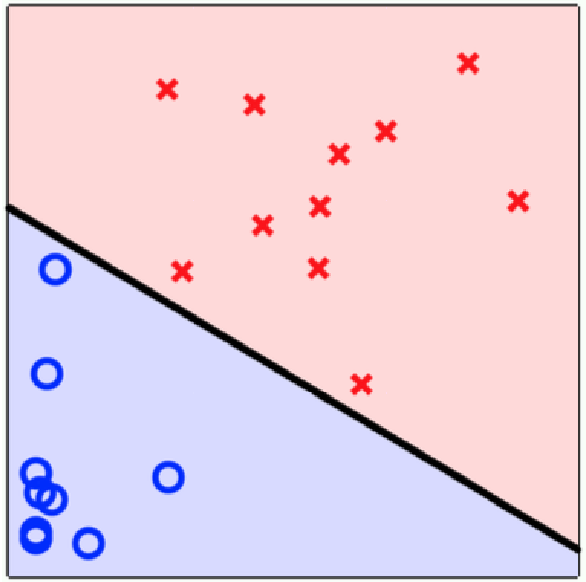
$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

$$\mathbf{z} = \mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix} = \begin{bmatrix} 1 \\ \Phi_1(\mathbf{x}) \\ \Phi_2(\mathbf{x}) \end{bmatrix}$$

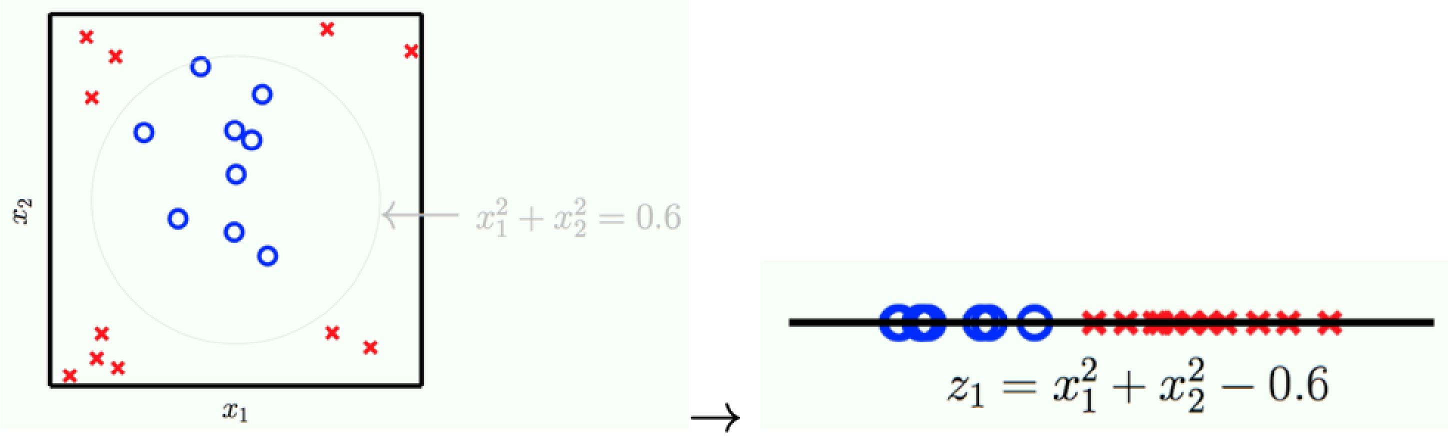
The data is now linearly separable in the **transformed space**!



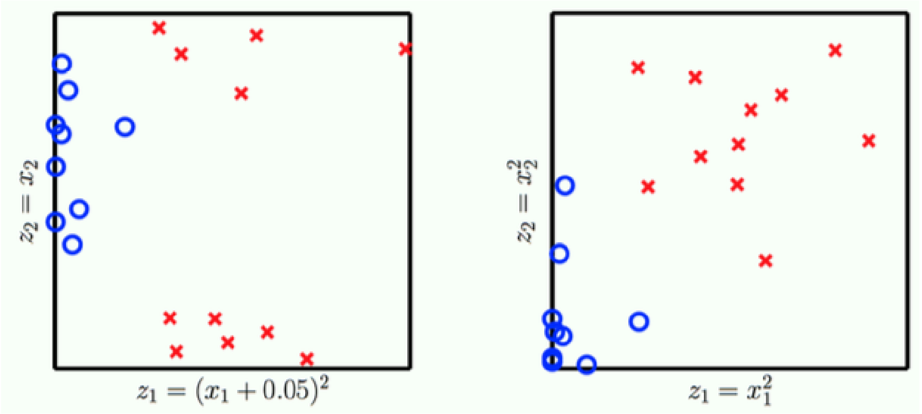
$$\tilde{g}(\mathbf{z}) = \text{sign}(\tilde{\mathbf{w}}^T \mathbf{z})$$



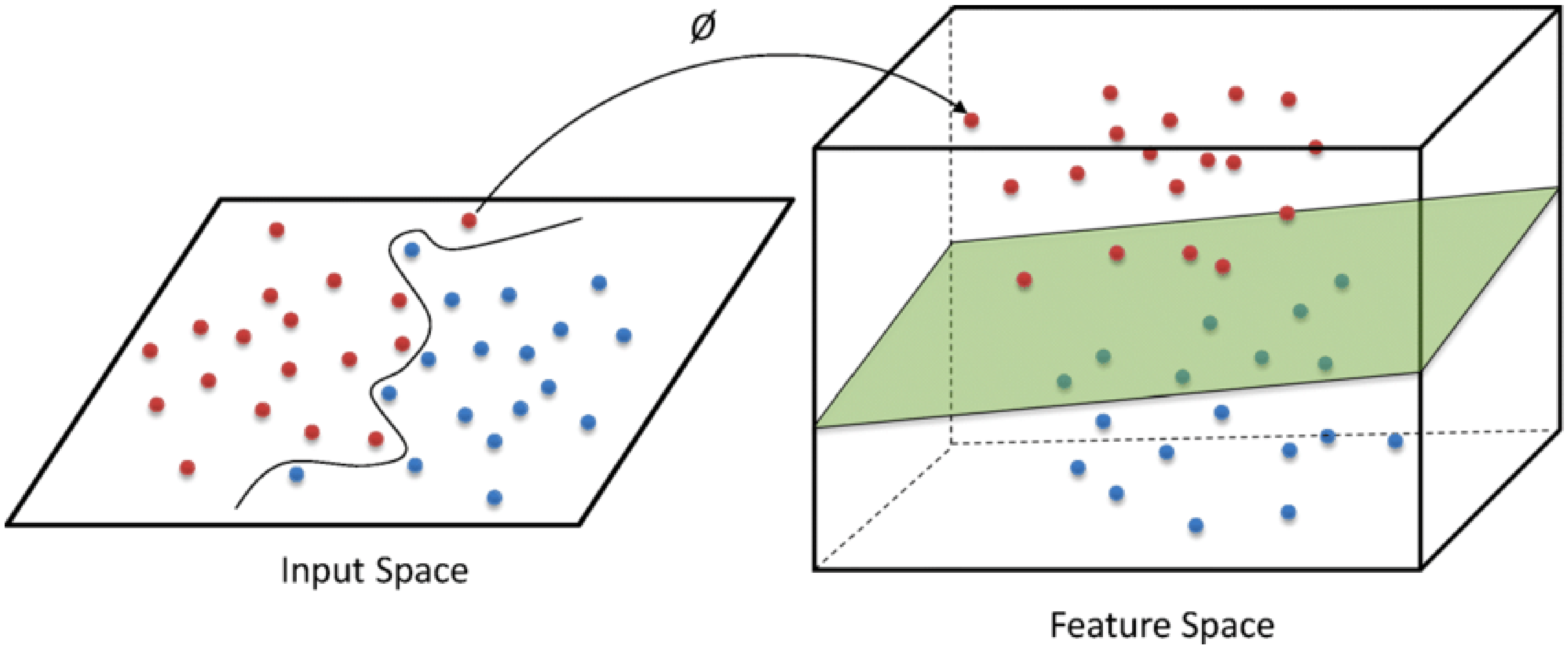
Many "feature transformation/expansion" would work!



And many other would work ...



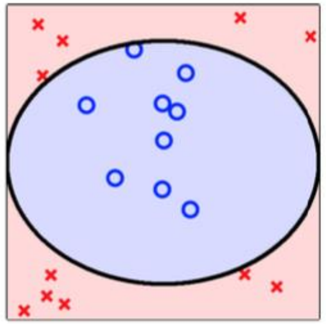
The general idea is that in higher dimensions it is easier for the data to be linearly separable



How do we do that systematically?

- Example: Quadratic expansion

$$\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = \mathbf{x} \rightarrow \mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ \Phi_1(\mathbf{x}) \\ \Phi_2(\mathbf{x}) \\ \Phi_3(\mathbf{x}) \\ \Phi_4(\mathbf{x}) \\ \Phi_5(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}$$

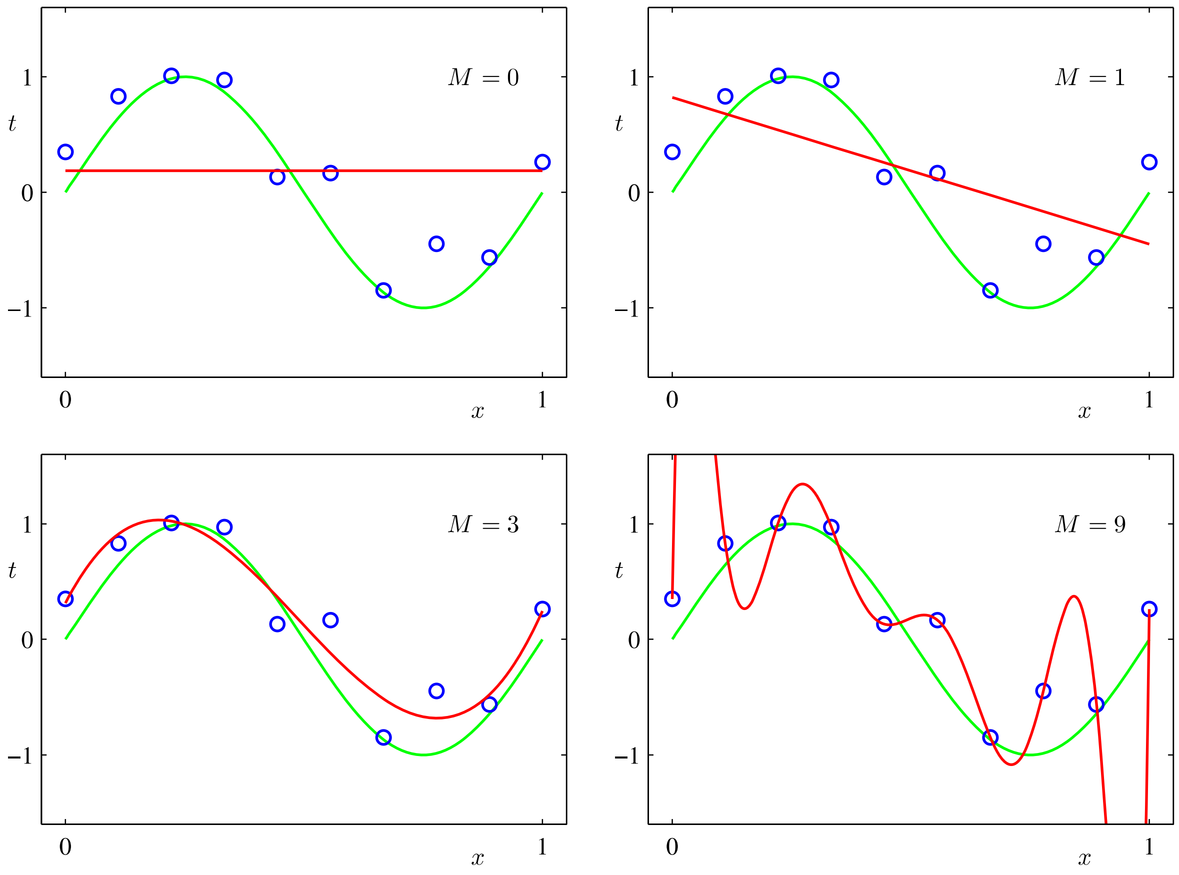


- More generally: kth order polynomial expansion

$$\mathbf{\Phi}_1(\mathbf{x}) = (1, x_1, x_2)$$
$$\mathbf{\Phi}_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$$
$$\mathbf{\Phi}_3(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3)$$

Any issue with this for learning?

Recap: As feature dimension increases, the model is prone to overfitting --- see "curve fitting"



"Kernel methods": systematically constructing feature expansions to a very high-dimension!

- Example: Discretization, assume $x \in \mathcal{X} = [0,1]$

$$\phi(x) = \begin{bmatrix} \mathbf{1}(x \in [0, \Delta]) \\ \mathbf{1}(x \in [\Delta, 2\Delta]) \\ \mathbf{1}(x \in [2\Delta, 3\Delta]) \\ \vdots \\ \mathbf{1}(x \in [1-\Delta, 1]) \end{bmatrix}$$

We can take Δ to be arbitrarily small.
It can fit any function.

But the dimension $O\left((1/\Delta)^d\right)$

- Example: Gaussian RBF kernel Expansion

$$\phi(x) = \left[e^{-\frac{\|x-t\|^2}{2\sigma^2}} \right]_{\text{for } t \in [0,1]}$$

"Kernel Trick"

It suffices to work with a finite n-dimension.

$$\phi(x) = \begin{bmatrix} e^{-\frac{\|x-x_1\|^2}{2\sigma^2}} \\ e^{-\frac{\|x-x_2\|^2}{2\sigma^2}} \\ \vdots \\ e^{-\frac{\|x-x_n\|^2}{2\sigma^2}} \end{bmatrix}$$

Kernel

- Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ be a *qualifying* "distance" function.
  - Example 1: Dot product, i.e., $k(x, x') = x \cdot x' = x^T x'$
  - Example 2: Gaussian RBF-kernel: $k(x, x') = e^{-\gamma||x - x'||^2}$
  - Example 3: x can be a string, a graph, or a protein structure! Check "String kernel" and "Graph kernels"

- It allows us to generalize all linear methods into kernel methods
  - linear in a high-dimensional/function space
  - Kernel Ridge Regression, Kernel Logistic Regression, Kernel SVM

Kernel ridge regression

- Ridge regression

$$\hat{\theta} = (X^T X + \lambda I_d)^{-1} X^T \mathbf{y}$$
$$= X^T (X X^T + \lambda I_n)^{-1} \mathbf{y}$$

- Prediction:

$$\langle x, \hat{\theta} \rangle = x^T X^T (X X^T + \lambda I_n)^{-1} \mathbf{y} = \sum_{i=1}^{n} \langle x, x_i \rangle \left[(X X^T + \lambda I_n)^{-1} \mathbf{y} \right]_i$$

- Kernel ridge regression

$$\langle x, \hat{\theta} \rangle = \left[k(x, x_1), \ldots, k(x, x_n) \right] (K + \lambda I_n)^{-1} \mathbf{y}$$

  - *K* is the matrix of kernelized features

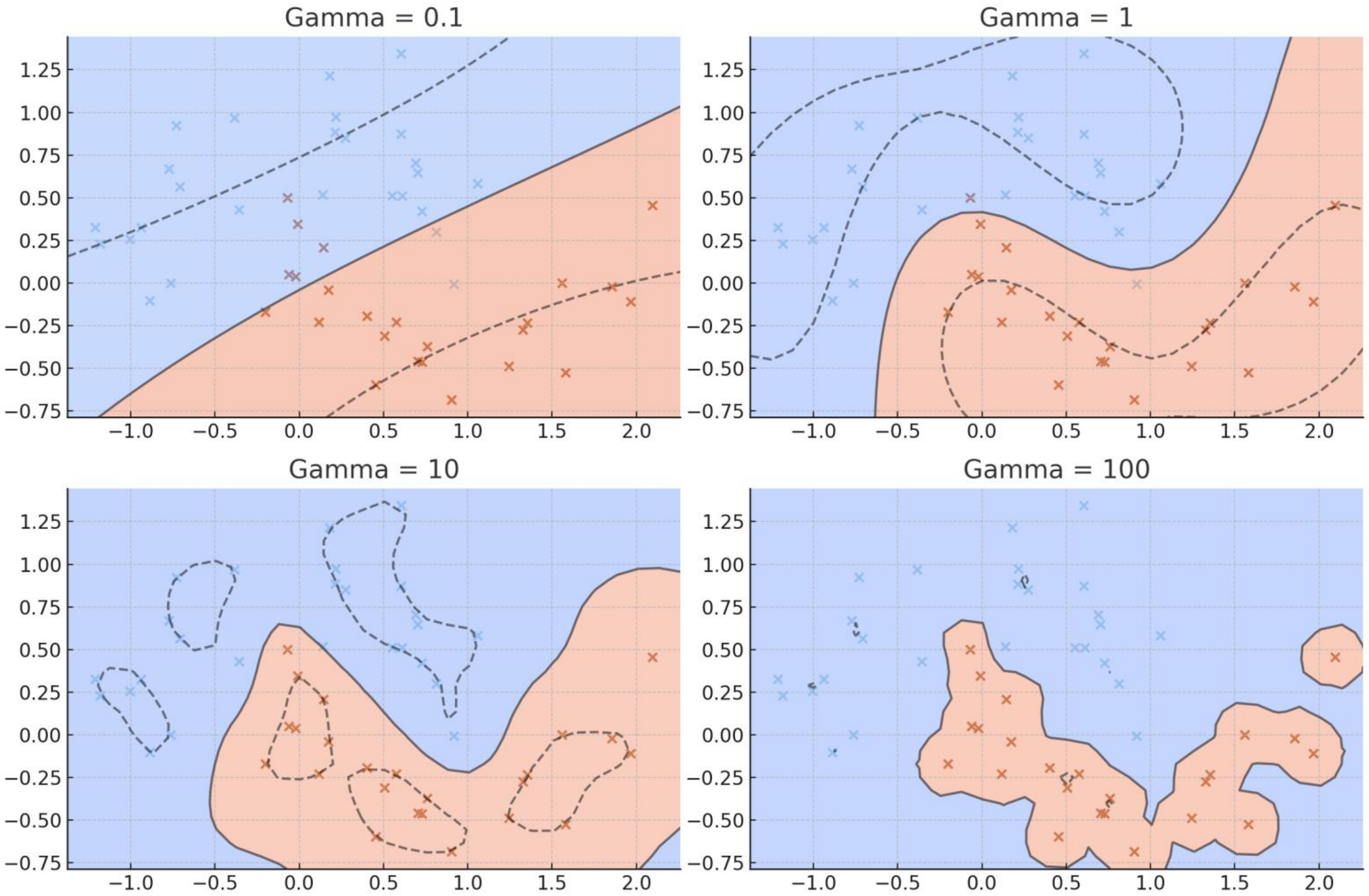
Implementing kernel SVM in just a few lines with *sklearn* (also on *libsvm* and *liblinear*)

```
from sklearn import svm

clf = svm.SVC(kernel='rbf', gamma=gamma)
clf.fit(X_train, y_train)
ypred = clf.predict(x_new)
```

Illustration of how a kernel-SVM works as we adjust the kernel bandwidth

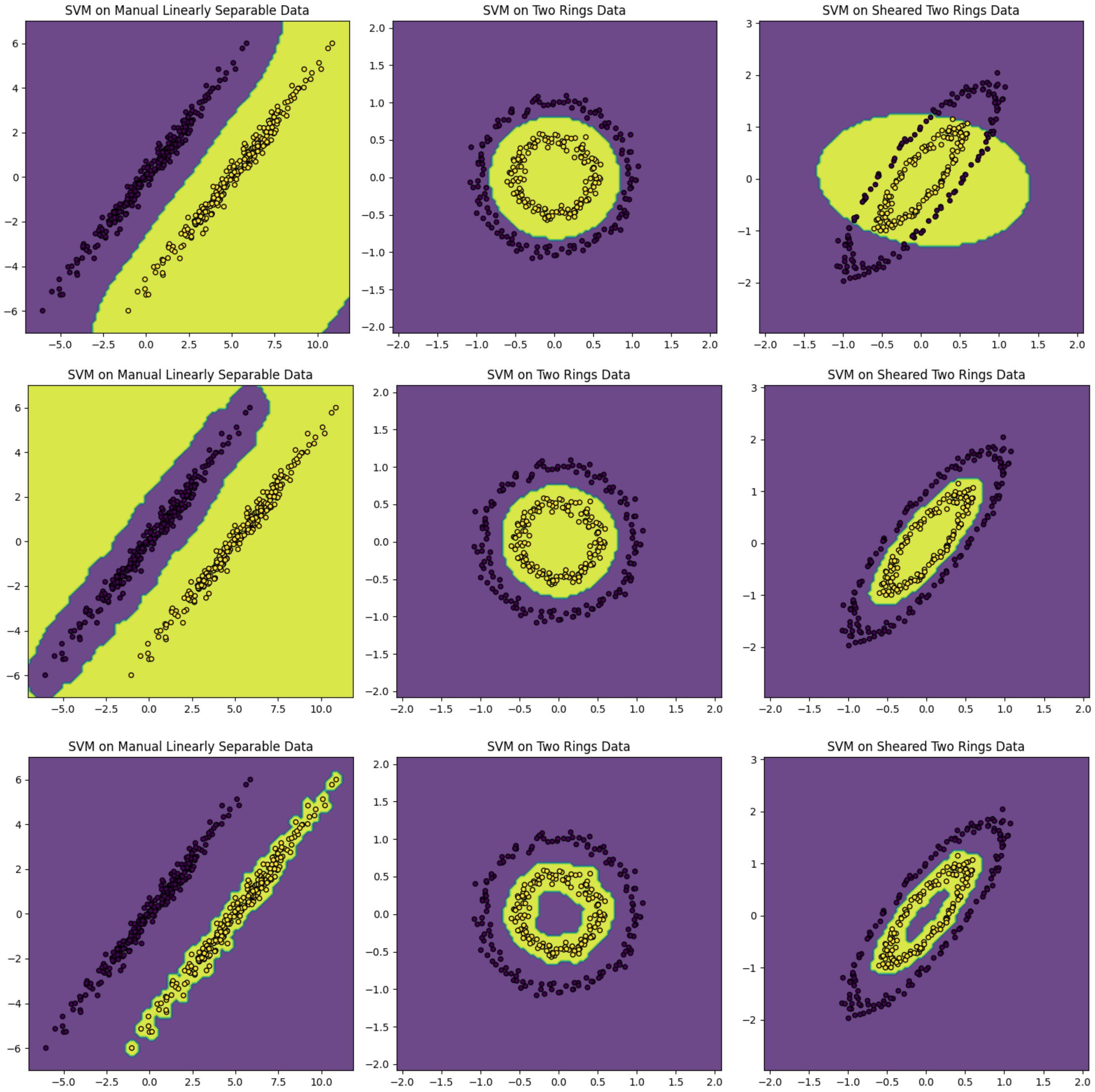
Kernel: $k(x, x') = e^{-\gamma||x - x'||^2}$ Feature map: $\phi(x) = e^{-\gamma||x - \cdot||^2}$



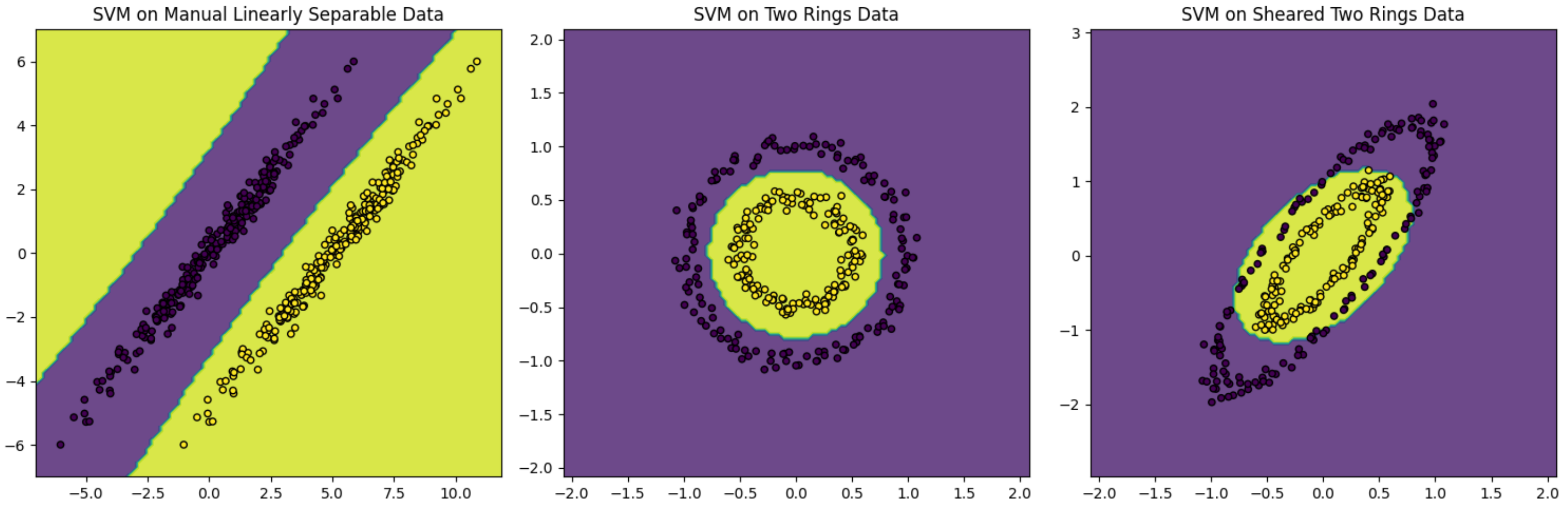
$\gamma = 0.1$

$\gamma = 10$

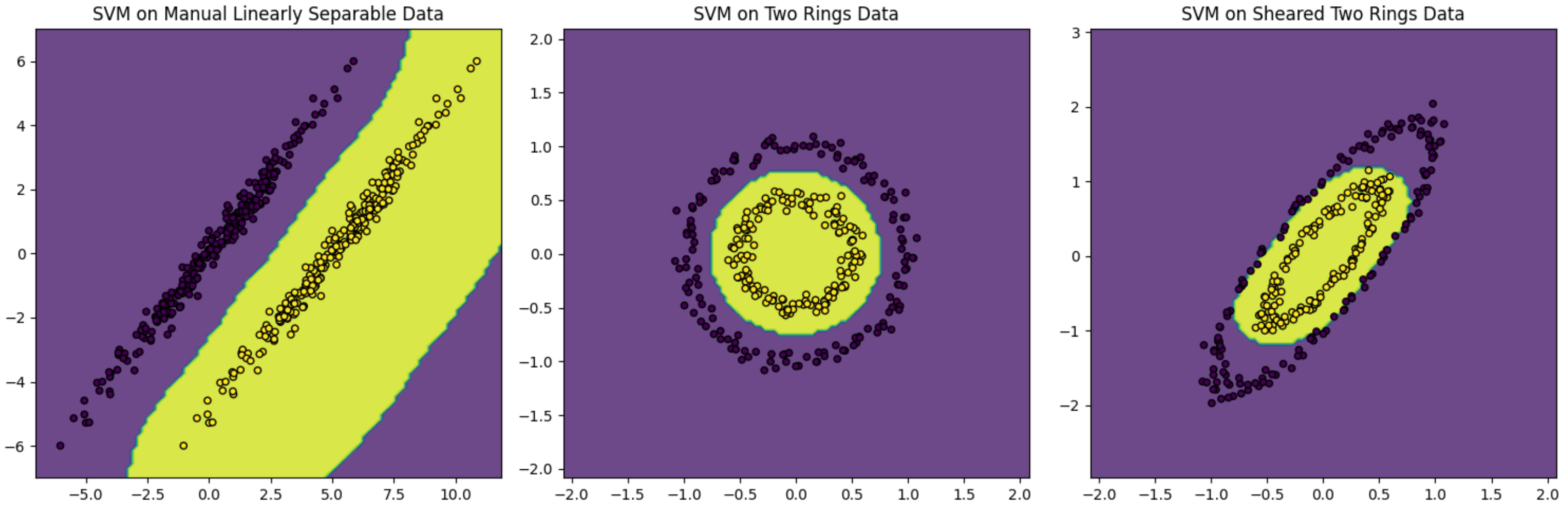
$\gamma = 100$



```python
svm_clf = SVC(kernel='poly', gamma='auto',degree=2)
```

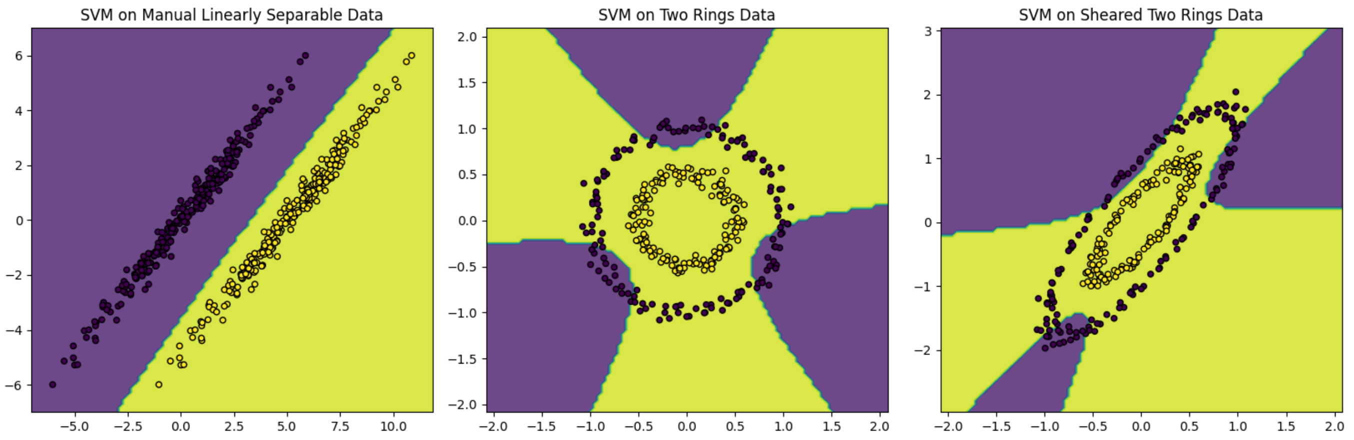



```python
svm_clf = SVC(kernel='rbf', gamma='auto')
```

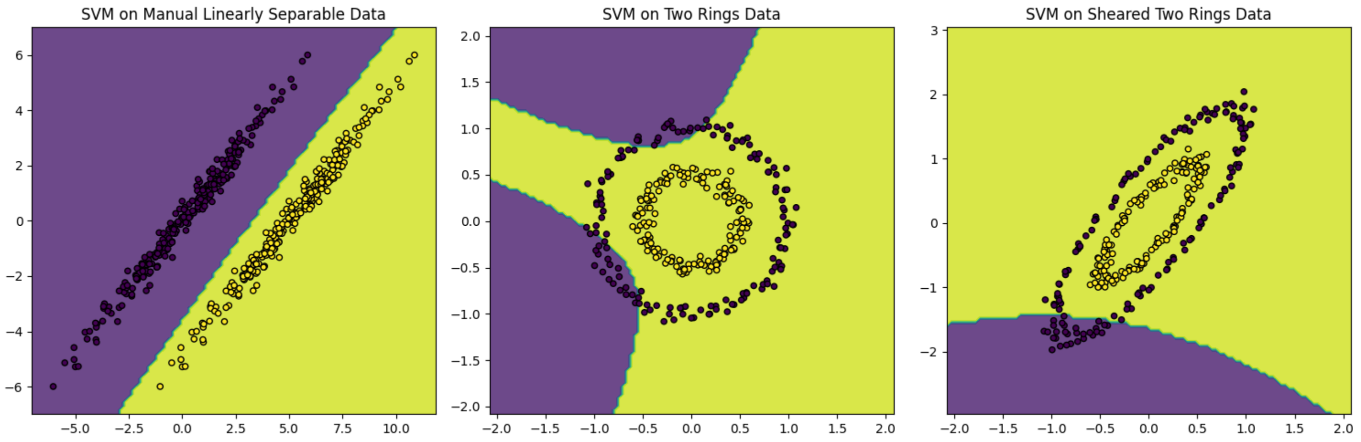


The choice of hyperparameters of these kernels can be delicate!

```python
svm_clf = SVC(kernel='poly', gamma=10,degree=3)
```




```python
svm_clf = SVC(kernel='poly', gamma='auto',degree=3)
```



Summary: Kernel methods

- They are essentially linear models --- linear in the expanded feature space
- Systematic way to tune the kernel-bandwidth, polynomial order, allows us to reduce “approximation error” and its tradeoff with “generalization error”.
- Drawbacks:
  - Need to specify the kernel
  - Computationally efficient but not scalable!