

Machine Learning

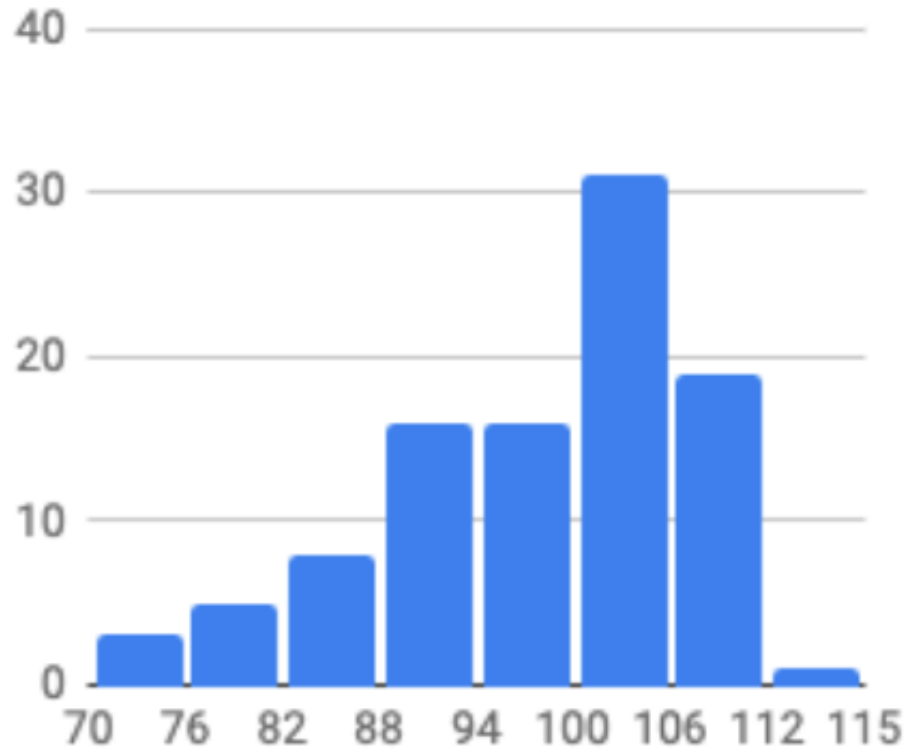
Feature expansion, kernels and neural networks

DSC 240

Feb 25, 2025

Instructor: Prof. Yu-Xiang Wang

Good job on the midterm quiz!



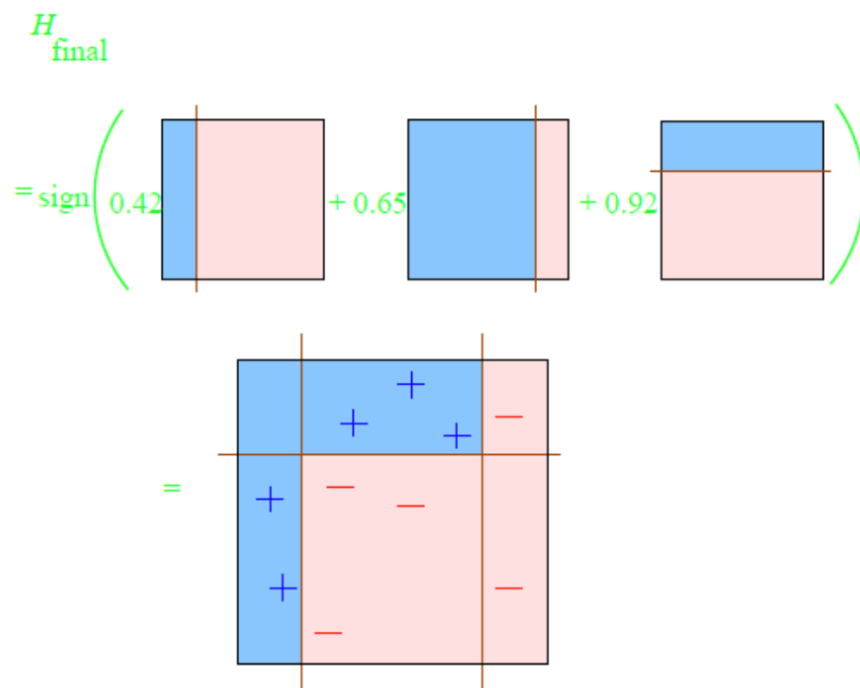
Class average 97, median 100

Last lecture

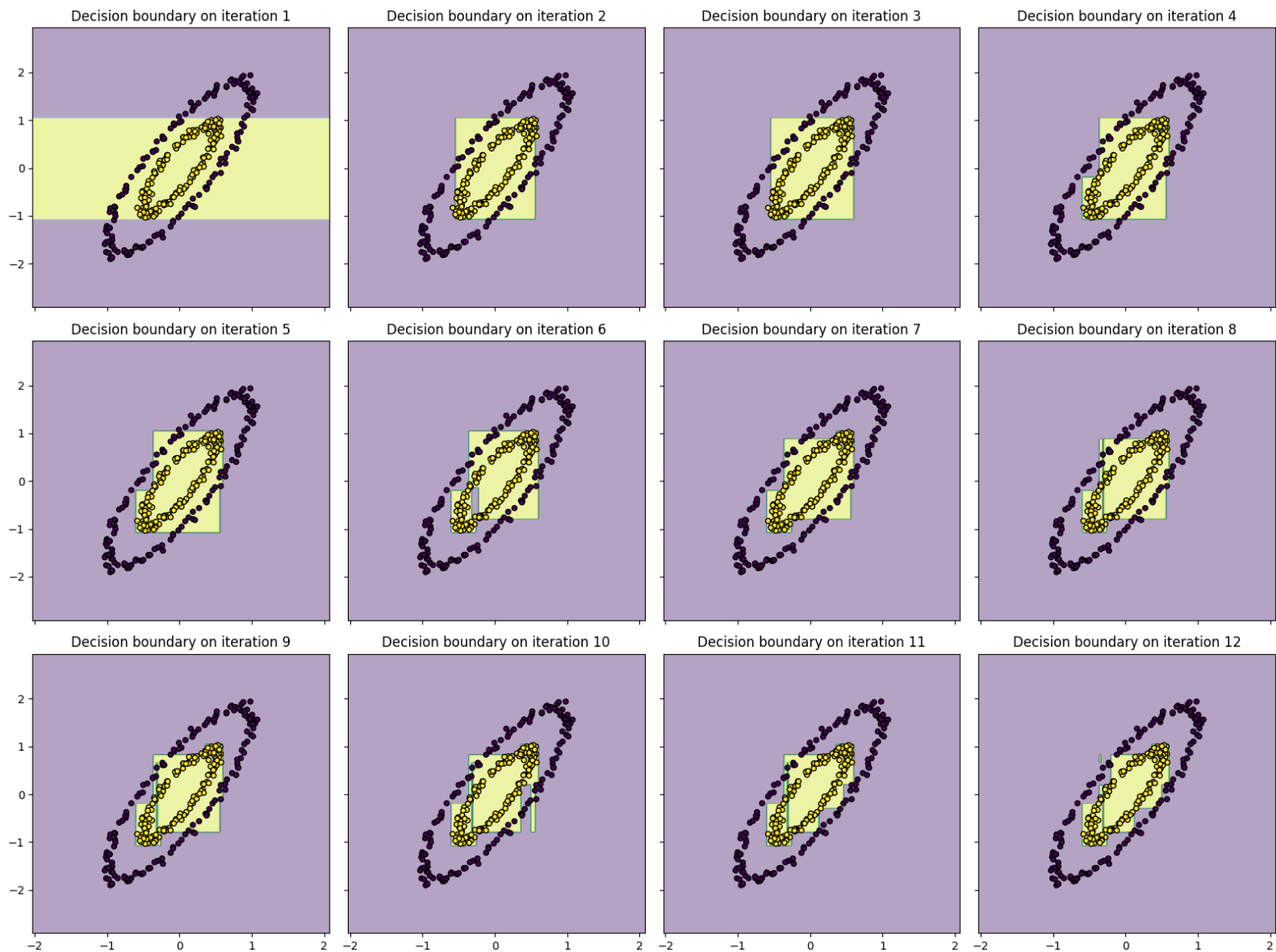
- Risk-Decomposition
 - “Optimization error”, “generalization error” and “approximation error”
- Ensemble Learning methods
 - Bagging and Random Forest
 - Boosting
- Take-home-message: “Weak Learner” → “Strong learner”
 - You can convert a “simple” ML algorithm (e.g., a Decision Stump) into a much stronger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

The main idea in bagging / boosting etc... is to construct “strong learner” by getting “Weak learners” to vote.

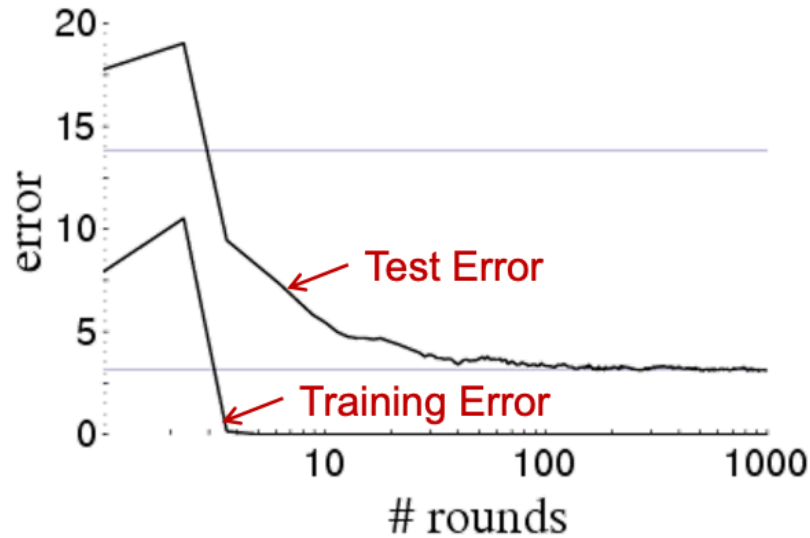
$$h_{\text{vote}}(x) = \arg \max_{y \in \mathcal{Y}} \frac{1}{N} \sum_{i=1}^N \alpha_i \mathbf{1}(h_i(x) = y)$$



Recap: Illustration of the iterations of AdaBoost (Base learner: Depth 2 Decision tree)



Recap: Errors as number of boosting iteration increases.

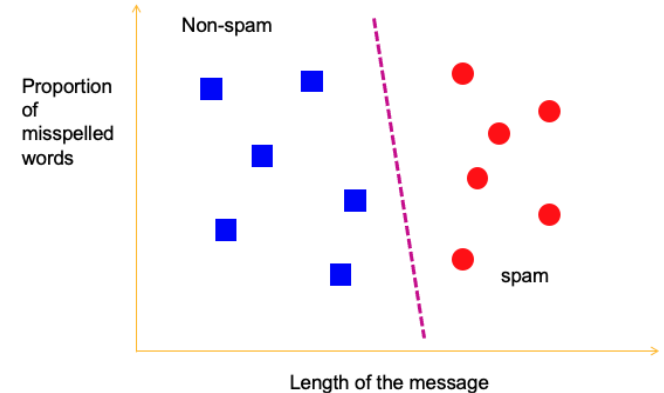


- How does the “optimization error”, “generalization error” and “approximation error” changes?

Today

- Feature expansion
- Kernel methods
- Neural Networks (if time permits)
- **Punchline: There is a lot more mileage in your linear learners.**

Recall: Linear classifiers



- How does it make prediction?

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

- Shape of the decision boundary: a hyperplane

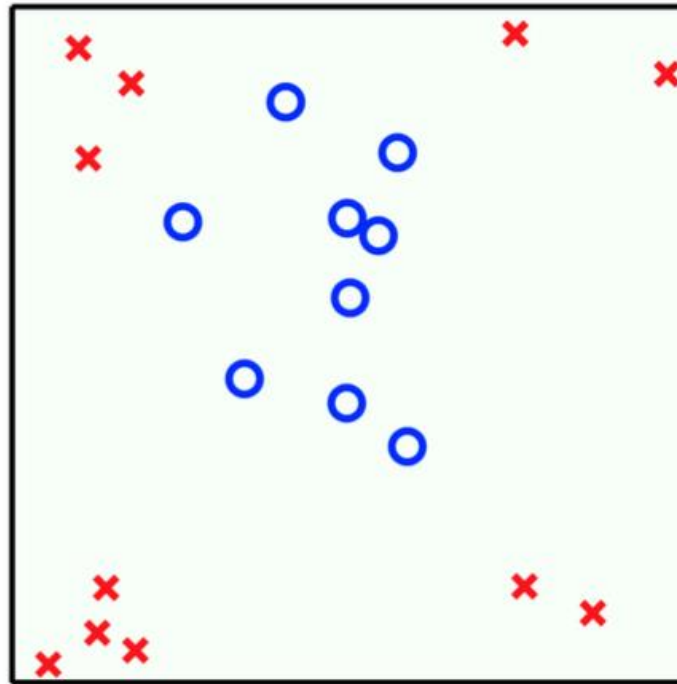
- Shape of **the set classified as “+”**, i.e., the **“discrimination function”**, i.e., the **classifier**

- Parameters: the weight vector

- How to train a linear classifier?

- Perceptron
- GD / SGD with Logistic loss, hinge loss or other surrogate losses
- Regularization

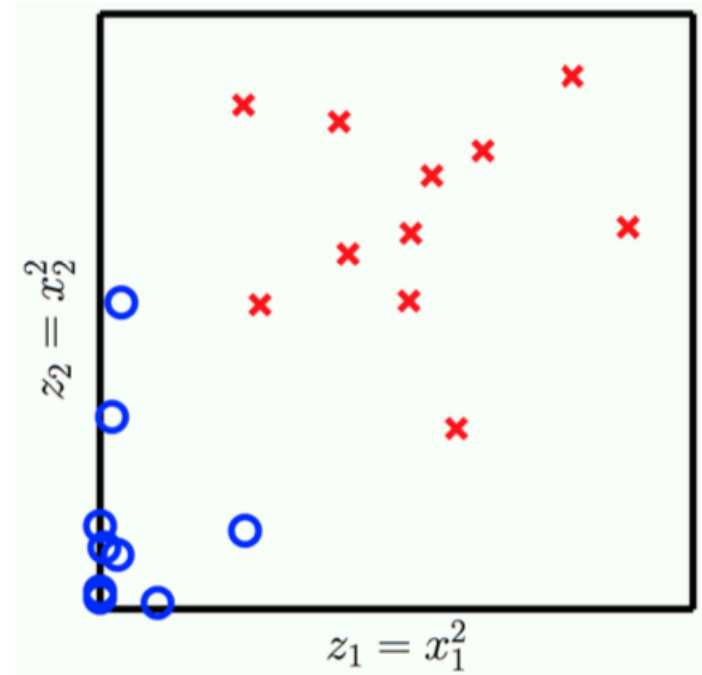
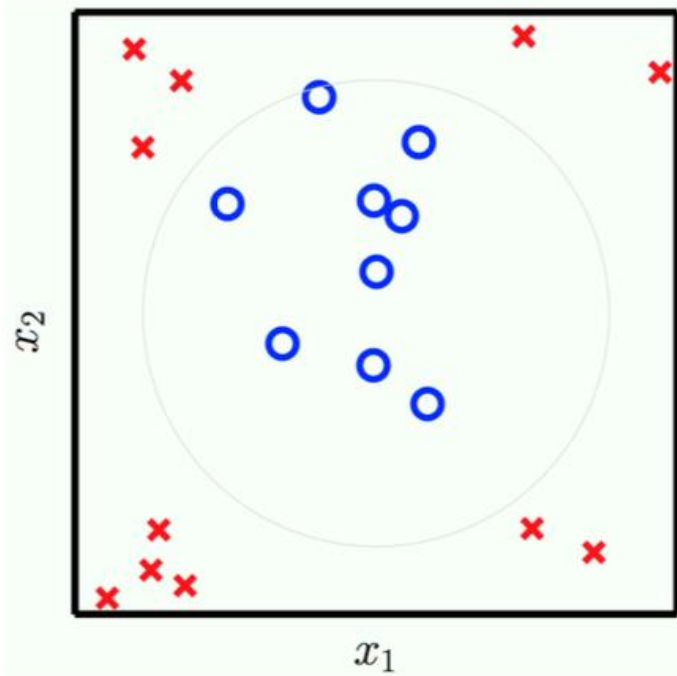
Linear classifier is limited. The best linear classifier might not be good.



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

- (A) Large optimization error
- (B) Large generalization error
- (C) Large Approximation Error
- (D) Overfitting
- (E) Underfitting

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

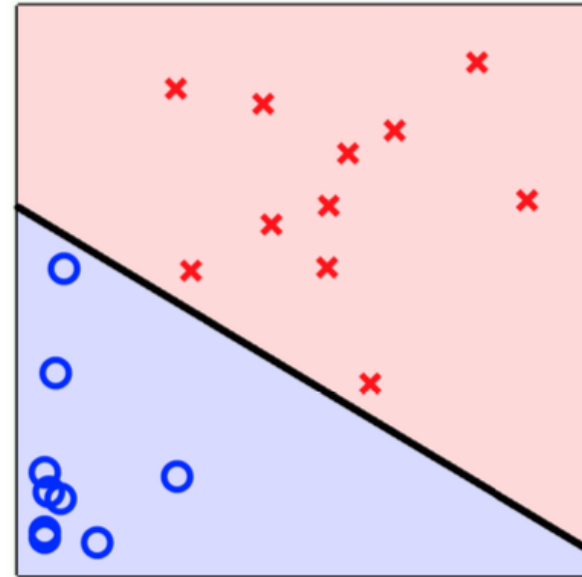
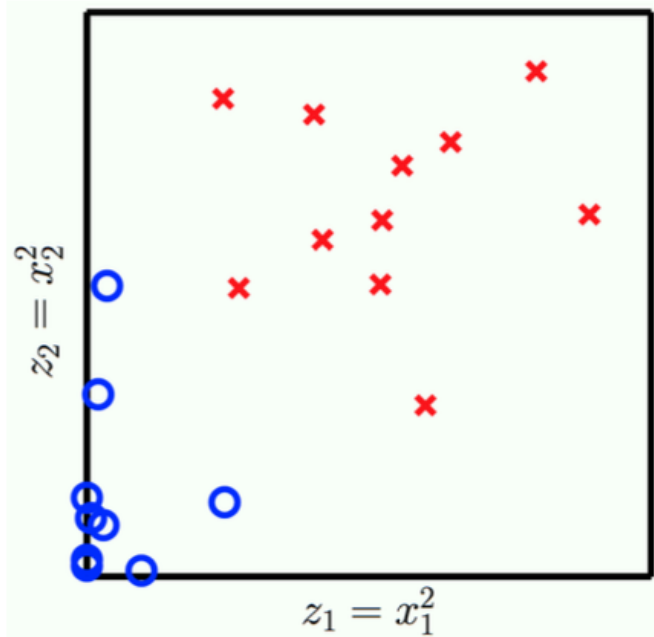


$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

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

The data is now linearly separable in the transformed space!

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

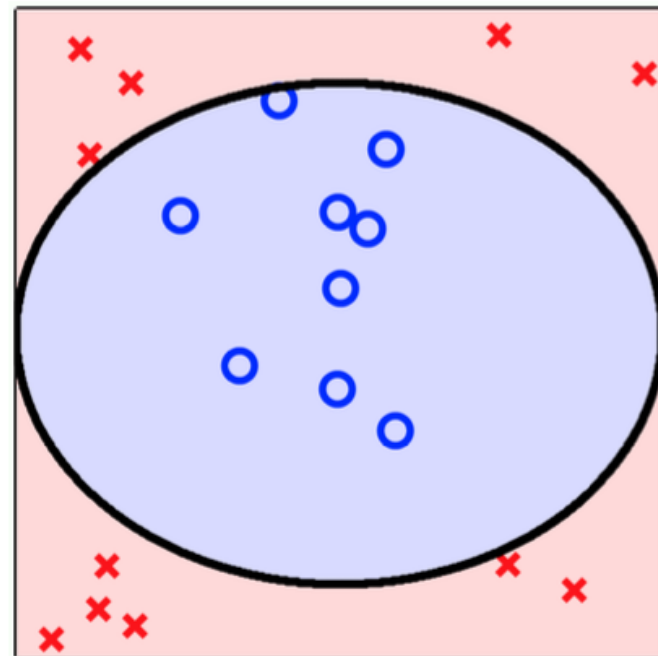
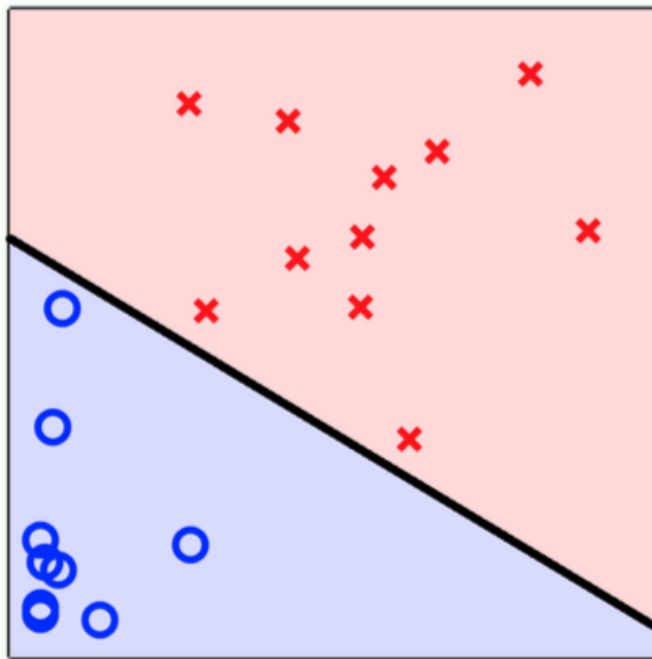


How to make a prediction for a new point x ?

- First transform x to $\phi(x)$ then apply the linear classifier in the transformed space!

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

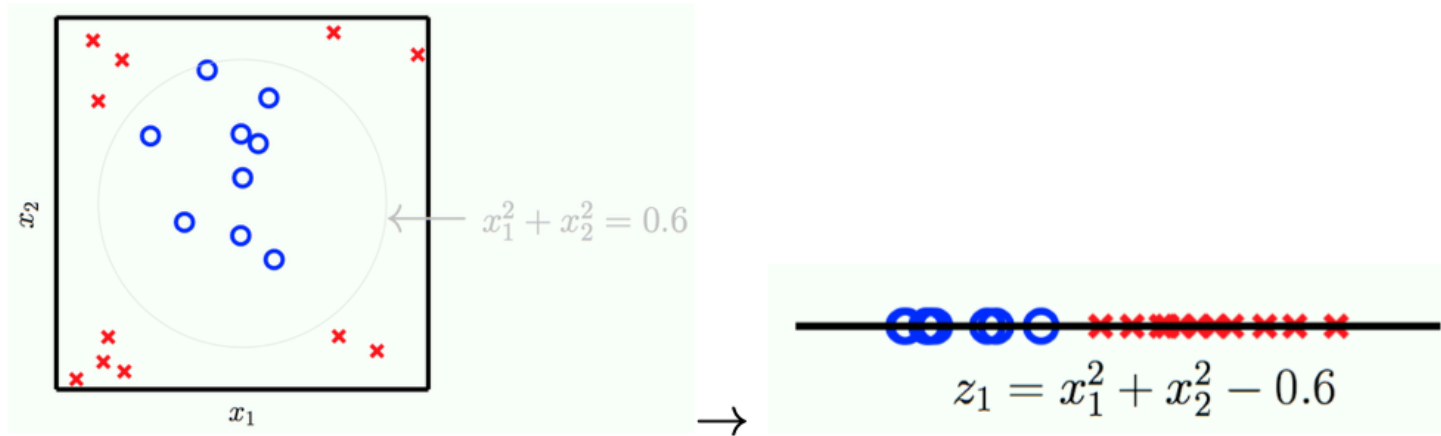
$$g(\mathbf{x}) = \tilde{g}(\Phi(\mathbf{x})) = \text{sign}(\tilde{\mathbf{w}}^T \Phi(\mathbf{x}))$$



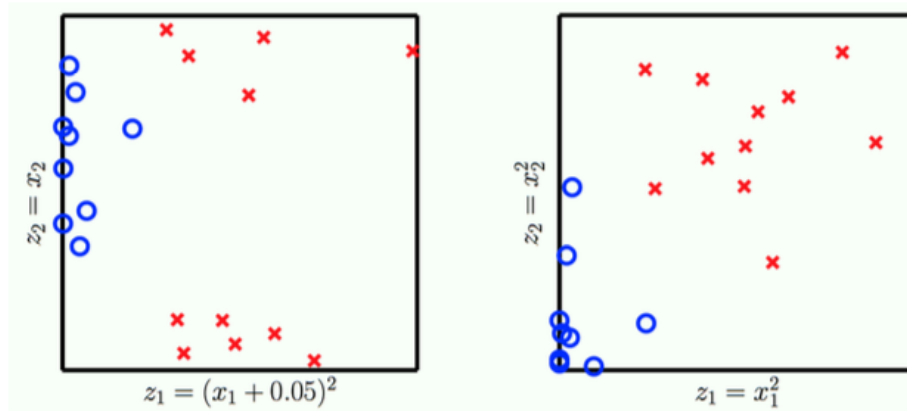
The idea of feature expansion --- “lifting” a feature vector to a higher-dimensional space.

	\mathcal{X} -space is \mathbb{R}^d	\mathcal{Z} -space is $\mathbb{R}^{\tilde{d}}$
Features	$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$	$\mathbf{z} = \Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \Phi_1(\mathbf{x}) \\ \vdots \\ \Phi_{\tilde{d}}(\mathbf{x}) \end{bmatrix}$
Data	$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$	$\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N$
Label	y_1, y_2, \dots, y_N	y_1, y_2, \dots, y_N
Weight	no weights	$\tilde{\mathbf{w}} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{\tilde{d}} \end{bmatrix}$
	Model $g(\mathbf{z}) = \text{sign}(\tilde{\mathbf{w}}^T \Phi(\mathbf{x}))$	

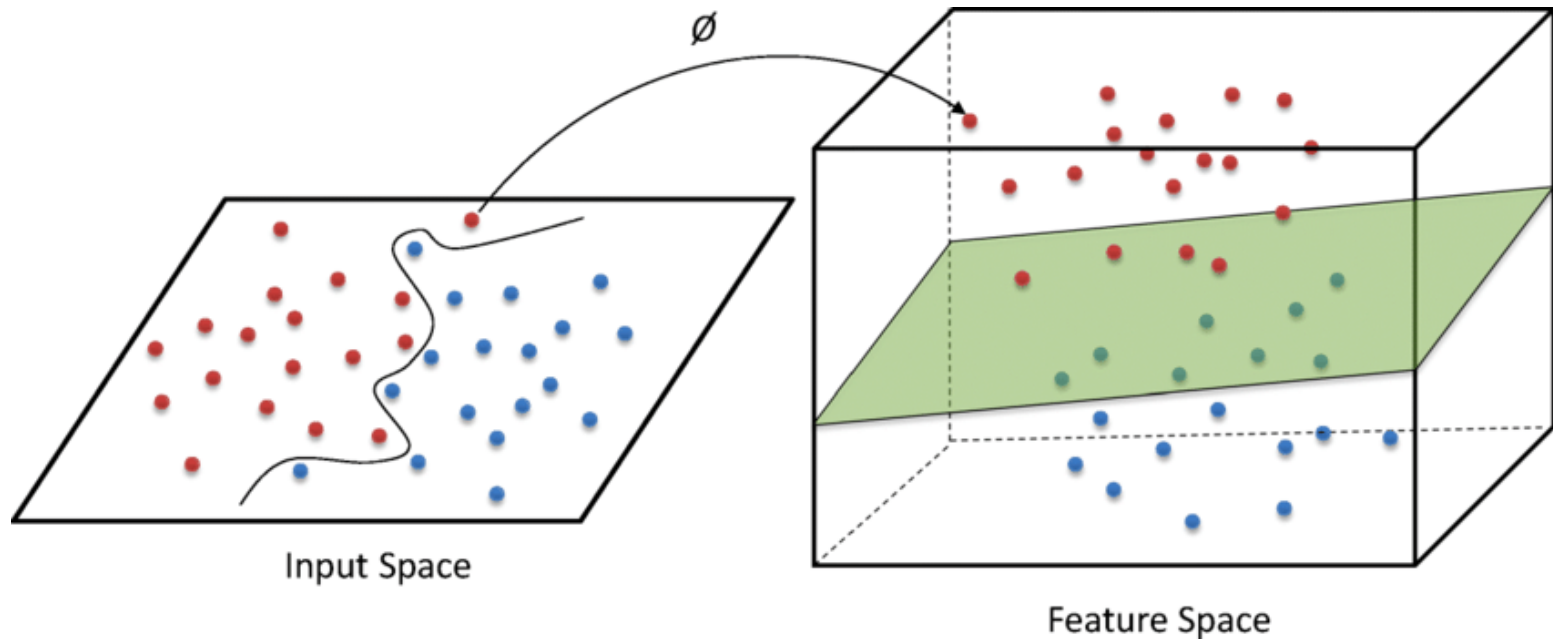
Many “feature transformation/expansion” would work!



And many other would work ...



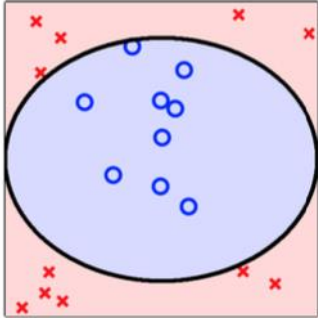
The general idea is that in higher dimensions it is easier for the data to be linearly separable



3 min Exercise: Constructing a feature map to linearly separate the following points

How do we do that systematically?

- Example: Quadratic expansion

$$\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = \mathbf{x} \rightarrow \Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \Phi_1(\mathbf{x}) \\ \Phi_2(\mathbf{x}) \\ \Phi_3(\mathbf{x}) \\ \Phi_4(\mathbf{x}) \\ \Phi_5(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \end{bmatrix}$$


- More generally: kth order polynomial expansion

$$\Phi_1(\mathbf{x}) = (1, \mathbf{x}_1, x_2)$$

$$\Phi_2(\mathbf{x}) = (1, \mathbf{x}_1, x_2, \mathbf{x}_1^2, x_1x_2, x_2^2)$$

$$\Phi_3(\mathbf{x}) = (1, \mathbf{x}_1, x_2, \mathbf{x}_1^2, x_1x_2, x_2^2, \mathbf{x}_1^3, x_1^2x_2, x_1x_2^2, x_2^3)$$

Any issue with this for learning?

Recap: As feature dimension increases, the model is prone to overfitting --- see the “curve fitting” example we saw before.

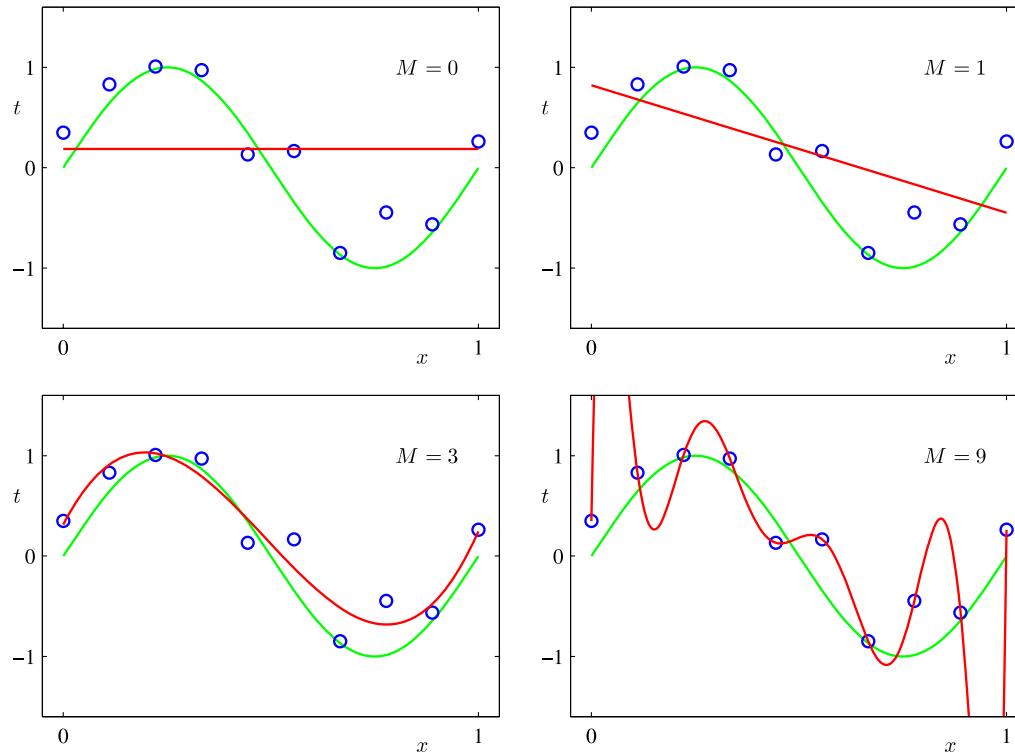


Figure 1.4 Plots of polynomials having various orders M , shown as red curves, fitted to the data set shown in Figure 1.2.

- What prevents overfitting? (1) Regularization; (2) Large-Margin; and (3) “effectively low-dimensionality”

“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 $O((1/\Delta)^d)$

***Minor observation: These features are outputs of decision trees!**

- 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.

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

(Mercer) Kernel and Reproducing Kernel Hilbert Space (RKHS)

- Let $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$ 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](#)”
- Which distance function “*qualifies*”?
 - **Positive semi definite** $k(\cdot, \cdot) \succeq 0$ by theorems of Mercer and Aronszajn.

- Define a feature map $\phi(x) = k(x, \cdot)$
 - For every such kernel, there exists a unique **Hilbert space \mathcal{H}** , that satisfies “**reproducing property**”, which implies:

$$\langle \phi(x), \phi(x') \rangle_{\mathcal{H}} = k(x, x')$$

- Conceptually, this 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

$$\begin{aligned}\hat{\theta} &= (X^T X + \lambda I_d)^{-1} X^T \mathbf{y} && \text{(See a cute proof [here](#))} \\ &= X^T (X X^T + \lambda I_n)^{-1} \mathbf{y}\end{aligned}$$

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

- Kernel ridge regression

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

- Observe that we never need to represent the infinite dimensional feature map! Only calls to $k(.,.)$ are needed!

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)
```

- How a fitted kernel SVM is making predictions?

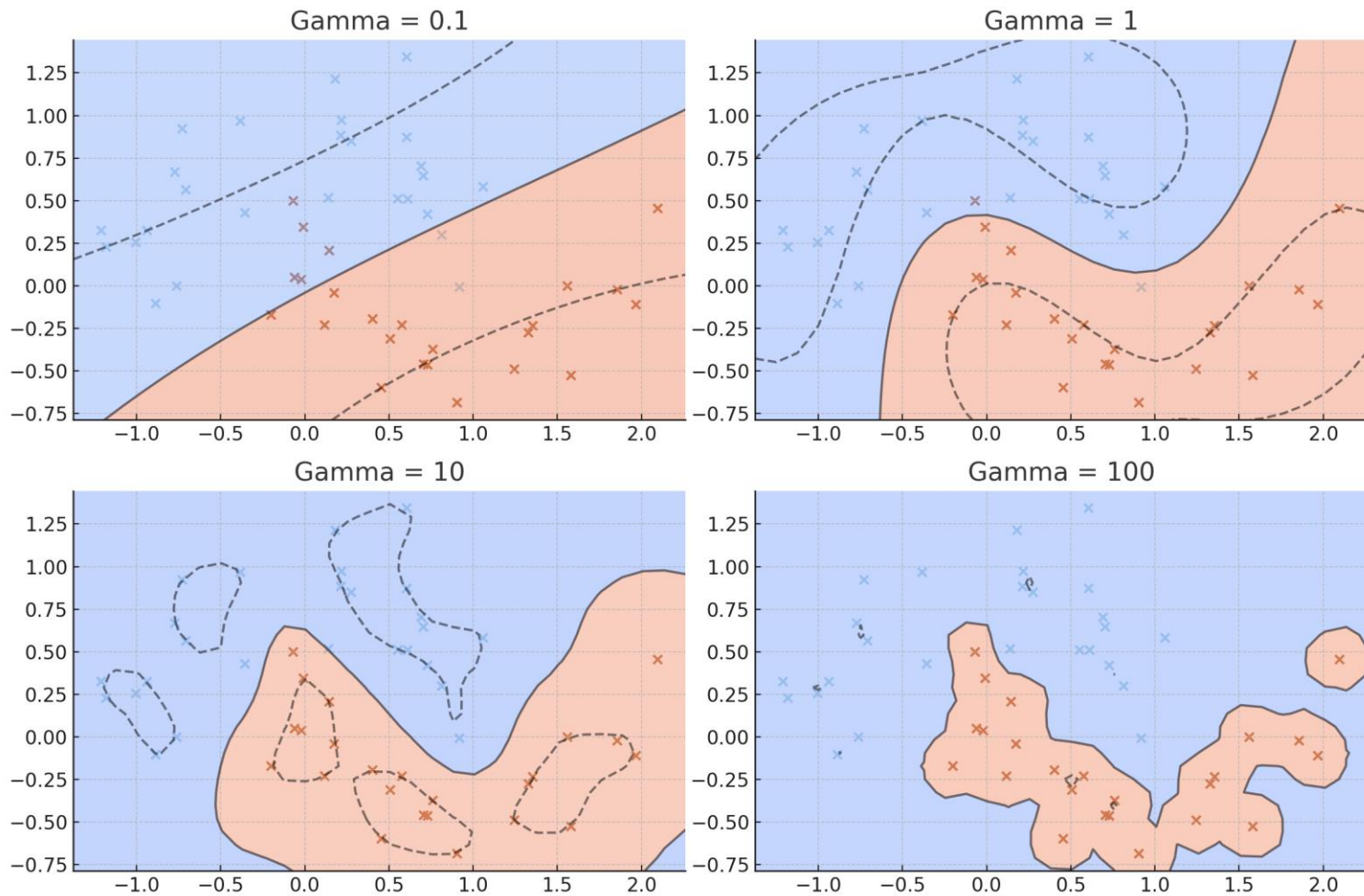
$$\hat{y}(x) = \text{sign} \left(\sum_{i=1}^n \alpha_i y_i k(x, x_i) + b \right) = \text{sign} \left(\left\langle \underbrace{\sum_{i=1}^n \alpha_i y_i k(x_i, \cdot)}_{\text{RKHS}_k}, k(x, \cdot) \right\rangle_{\text{RKHS}_k} + b \right)$$

Linear combination of your neighbors!

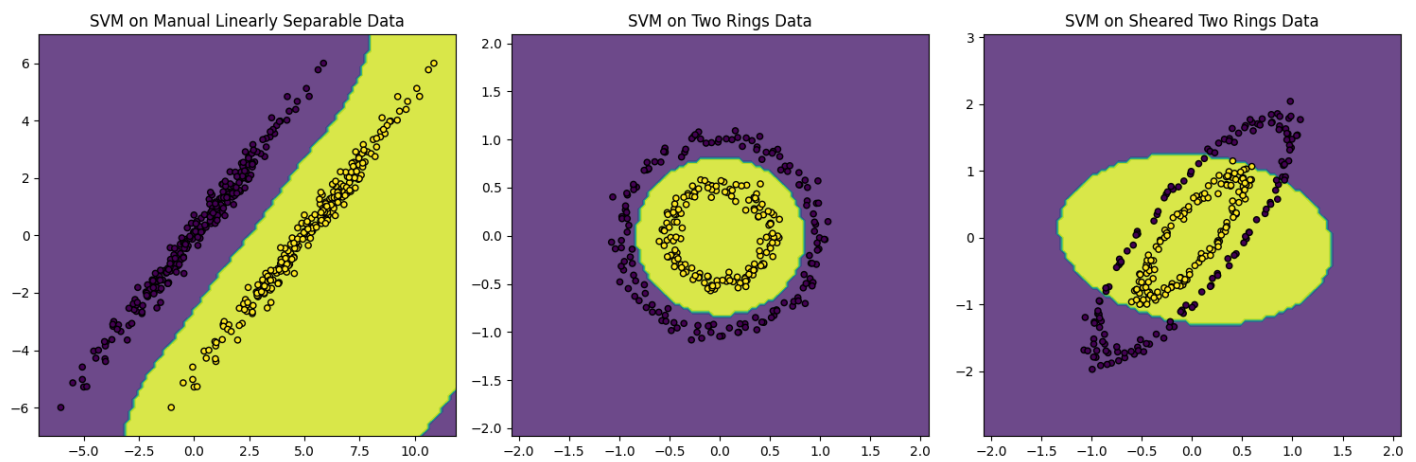
A linear classifier in an infinite dimensional space.
Here are the weights.

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

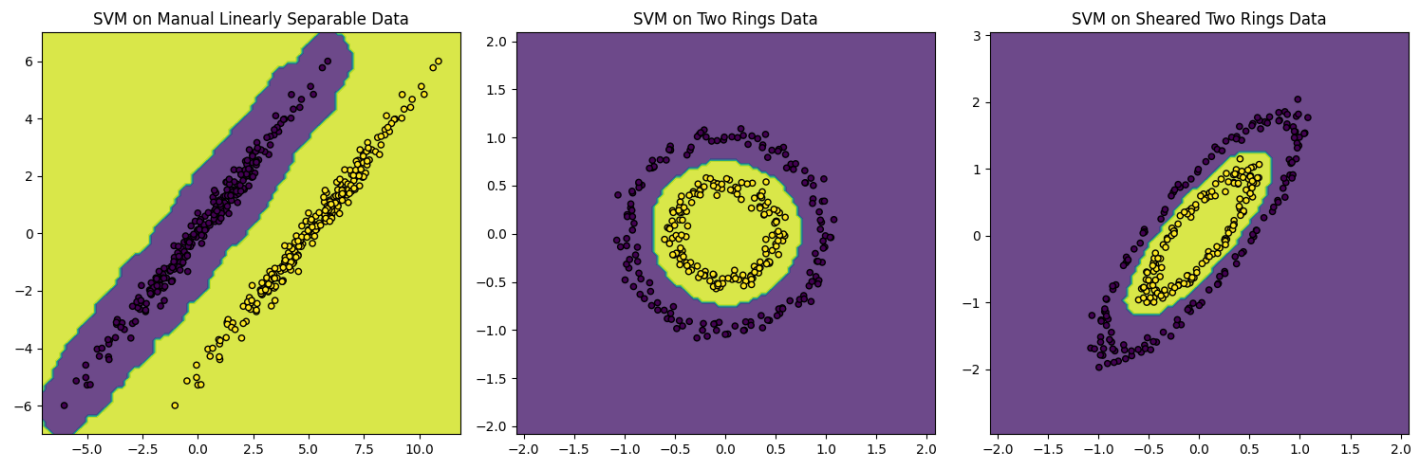
Kernel: $k(x, x') = e^{-\gamma \|x - x'\|^2}$ Feature map: $\phi(x) = e^{-\gamma \|x - \cdot\|^2}$



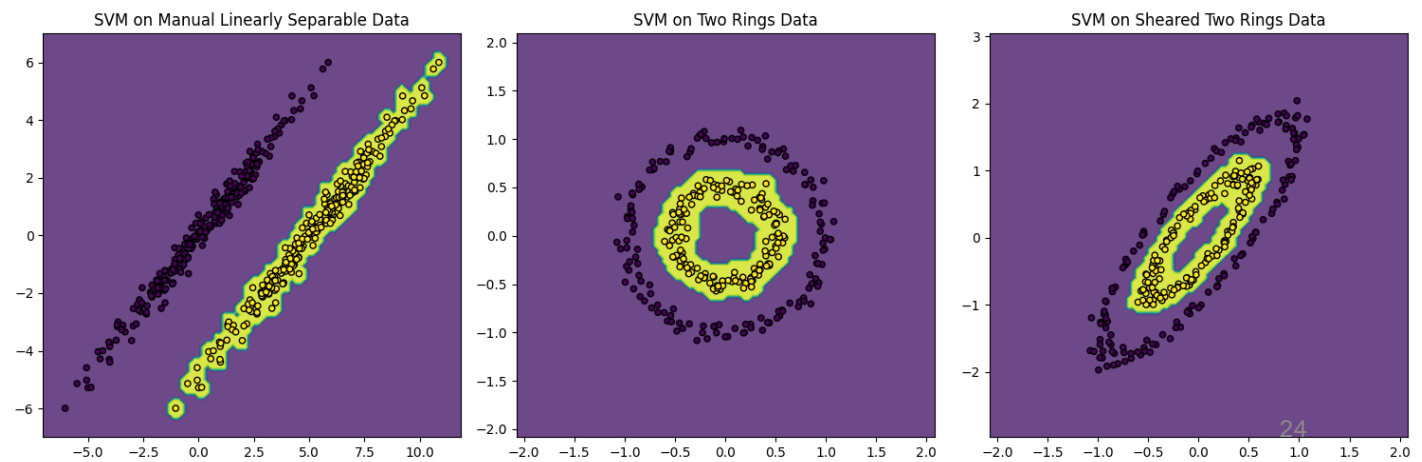
$\gamma = 0.1$



$\gamma = 10$

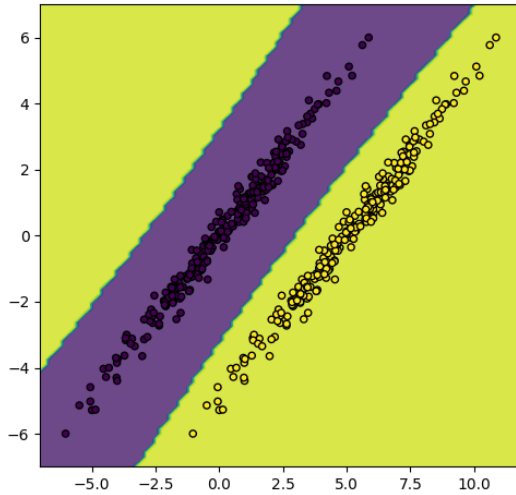


$\gamma = 100$

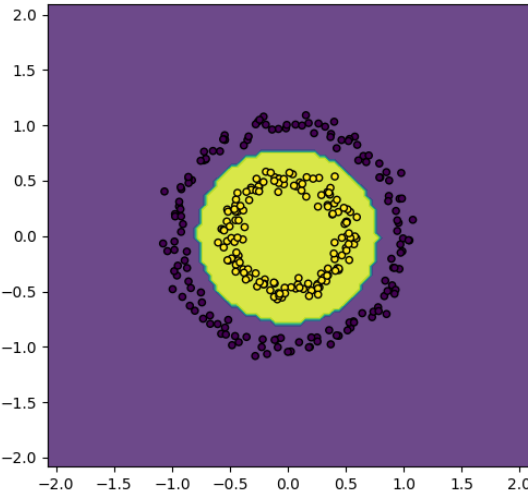


```
svm_clf = SVC(kernel='poly', gamma='auto', degree=2)
```

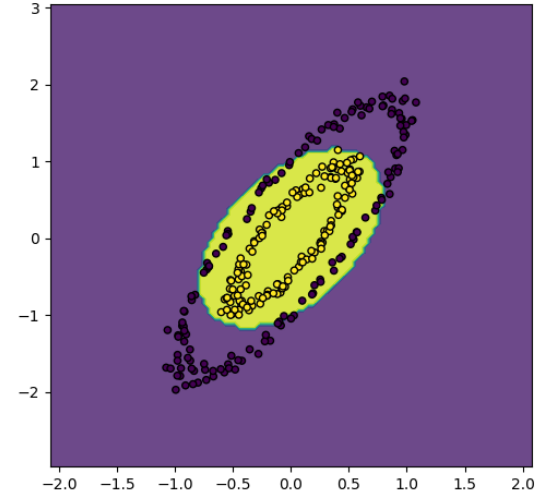
SVM on Manual Linearly Separable Data



SVM on Two Rings Data

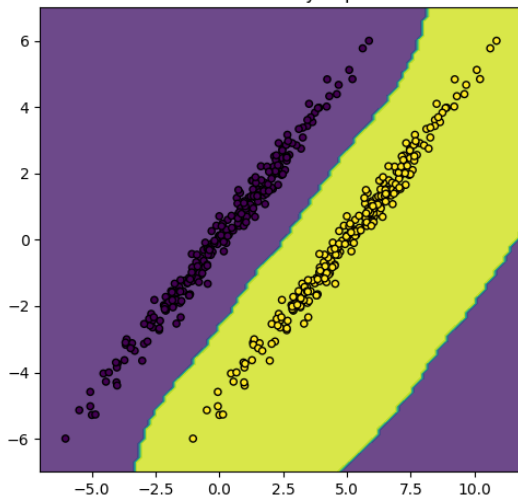


SVM on Sheared Two Rings Data

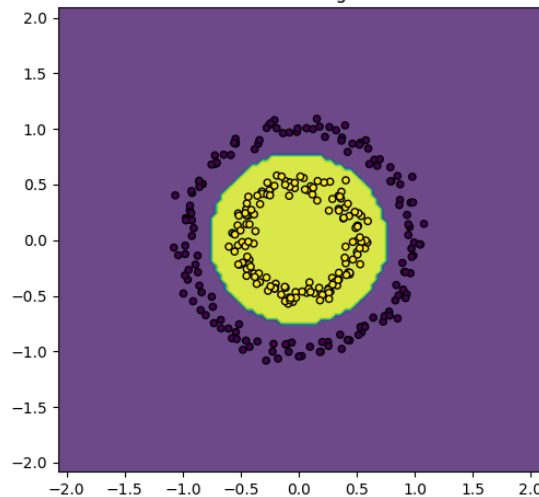


```
svm_clf = SVC(kernel='rbf', gamma='auto')
```

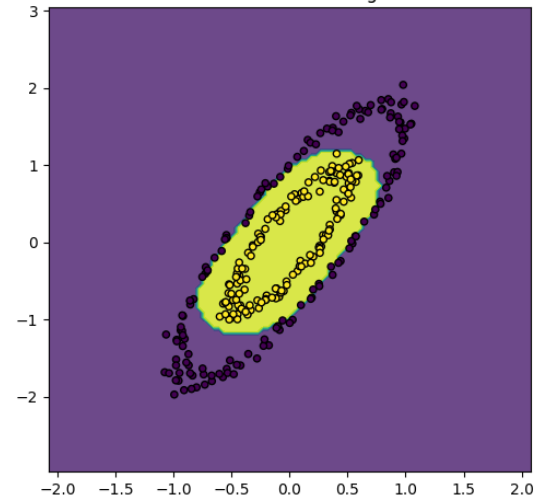
SVM on Manual Linearly Separable Data



SVM on Two Rings Data

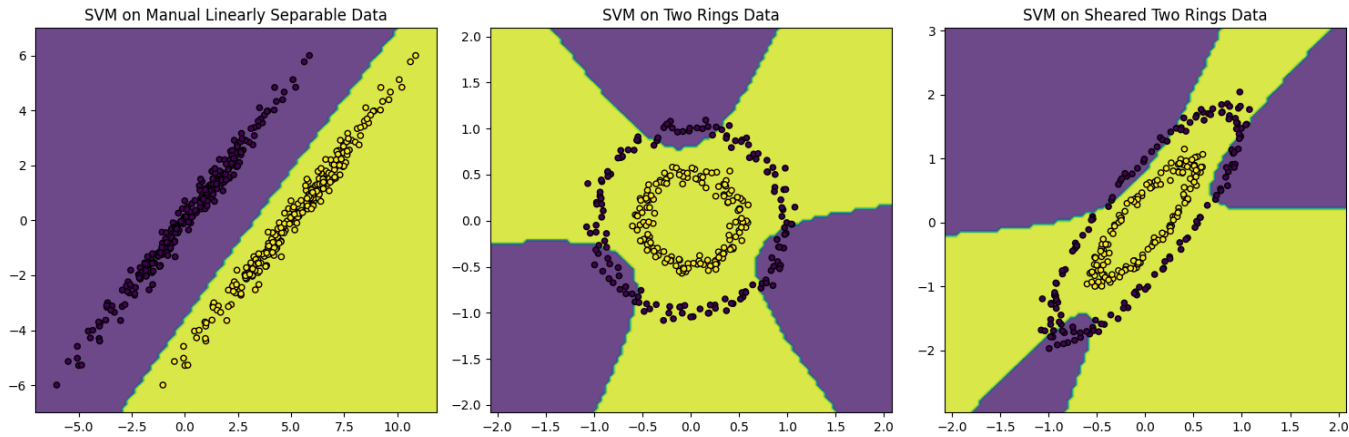


SVM on Sheared Two Rings Data

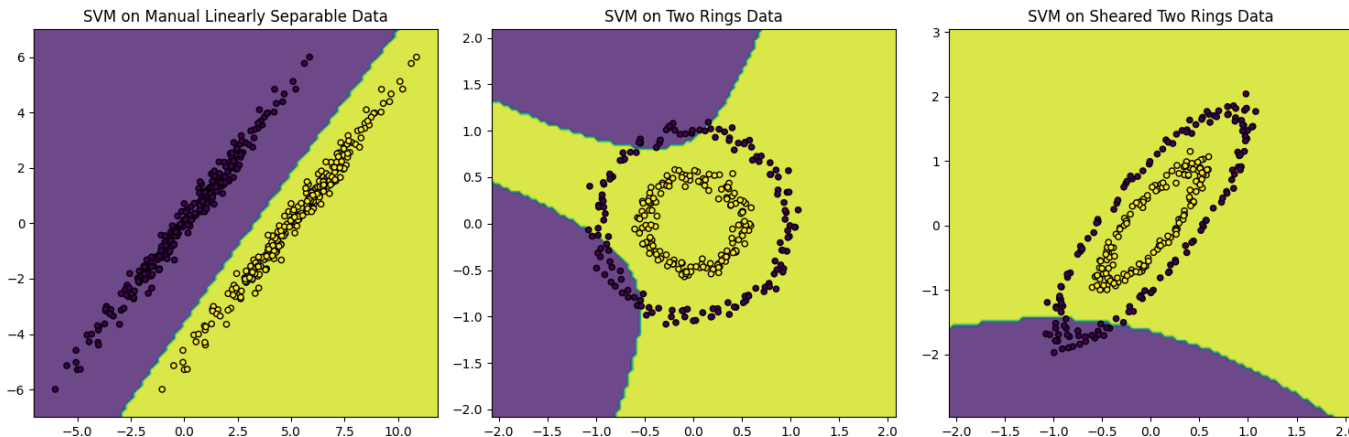


The choice of hyperparameters of these kernels can be delicate!

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



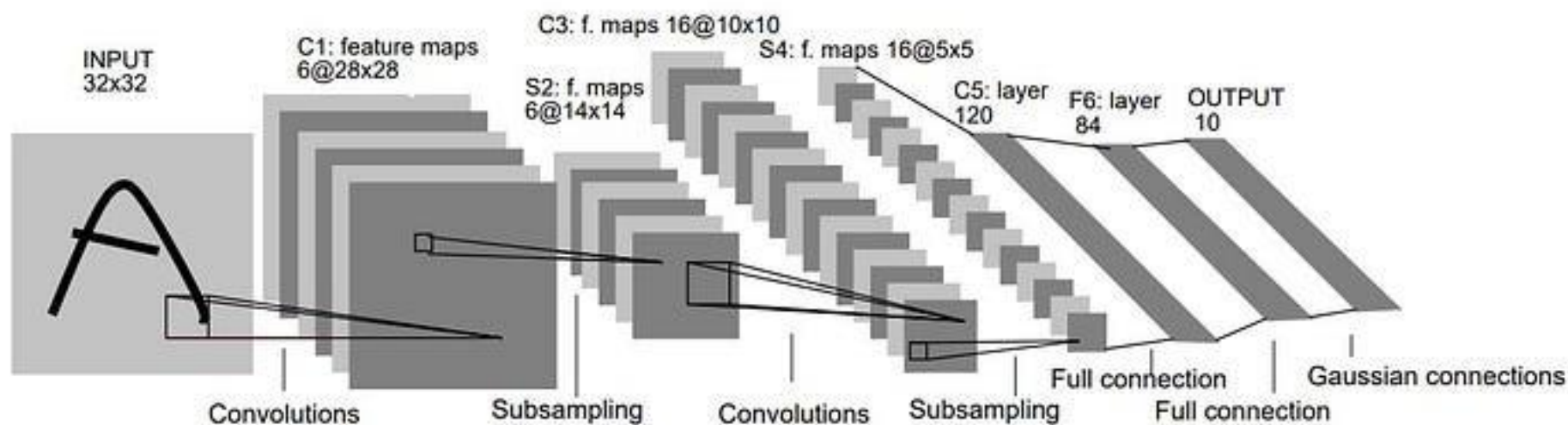
```
svm_clf = SVC(kernel='poly', gamma='auto', degree=3)
```



Checkpoint: 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 ($O(n^3)$)!

LeNet (1998)



Gradient-based learning applied to document recognition

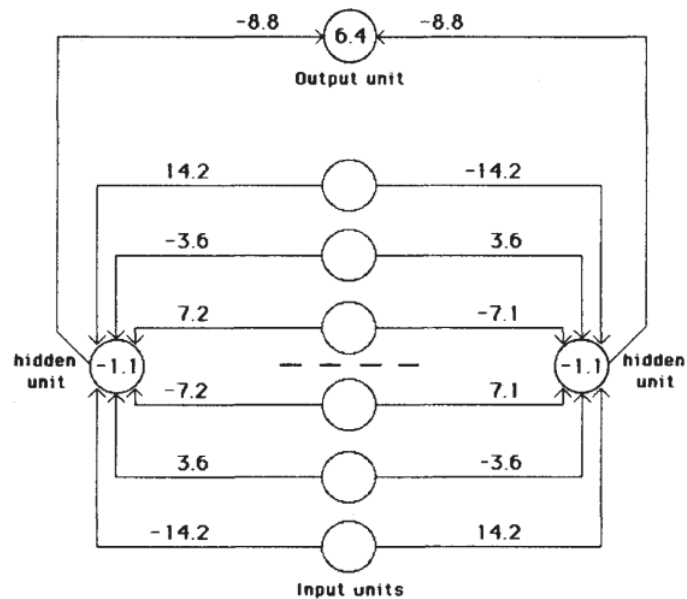
[Y LeCun, L Bottou, Y Bengio...](#) - Proceedings of the ..., 1998 - [ieeexplore.ieee.org](#)

... **gradientbased learning** technique. Given an appropriate network architecture, **gradient-based learning** algorithms can be **used** to ... methods **applied** to handwritten character **recognition** ...

☆ Save [Cite](#) Cited by 59964 [Related articles](#) [All 41 versions](#) [Web of Science: 27156](#) [Import into BibTeX](#)

Rumelhart, Hinton, Williams (1986)

- One layer of a feedforward neural networks



Learning representations by back-propagating errors

DE Rumelhart, [GE Hinton](#), [RJ Williams](#) - nature, 1986 - nature.com

... their states are completely determined by the input vector: they do not **learn representations**.)

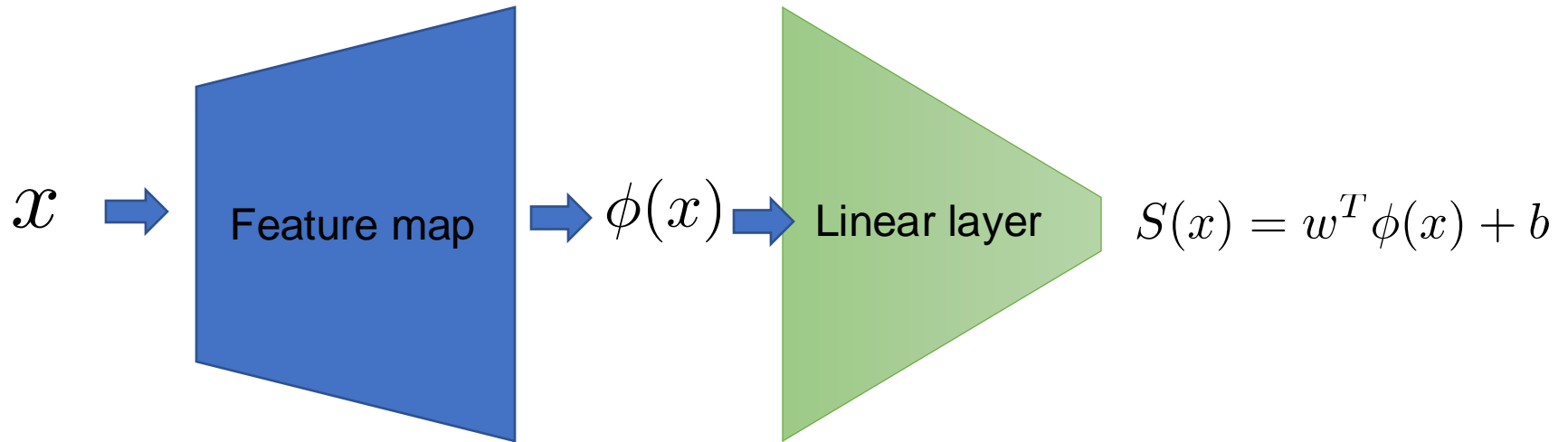
The **learning** procedure must decide under what circumstances the hidden units should be ...

☆ Save [Cite](#) Cited by 35698 [Related articles](#) [All 29 versions](#) [Import into BibTeX](#)

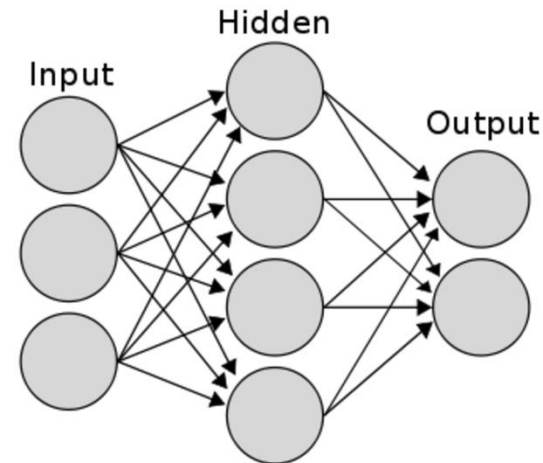
It goes back even further...

- 1943 Pitts and McCulloch: Perceptron model to mimic the brain
- 1956: Rosenblatt's Perceptron Implementation
- 1960s:
 - Ivakhnenko and Lapa: Multi-layer Perceptron (going deeper)
 - Dreyfus: Backpropagation for training (not yet the same as SGD)
 - Amari: Use SGD for training MLPs (separating non-linearly separable patterns)
- 1970s:
 - Fukushima: Convolutional Neural Networks for images
- 1982:
 - Werbos: Modern day backpropagation / SGD

From kernels to neural networks



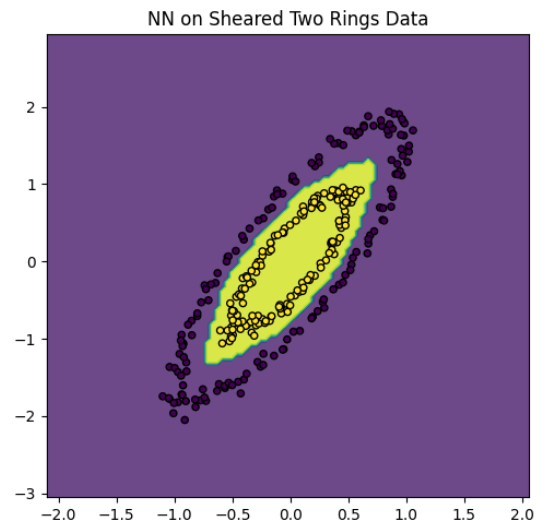
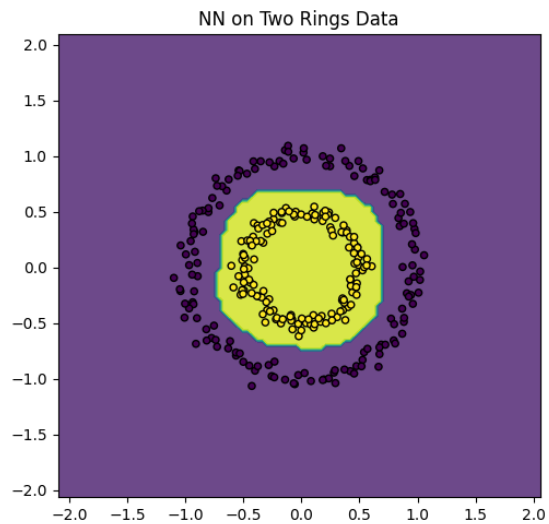
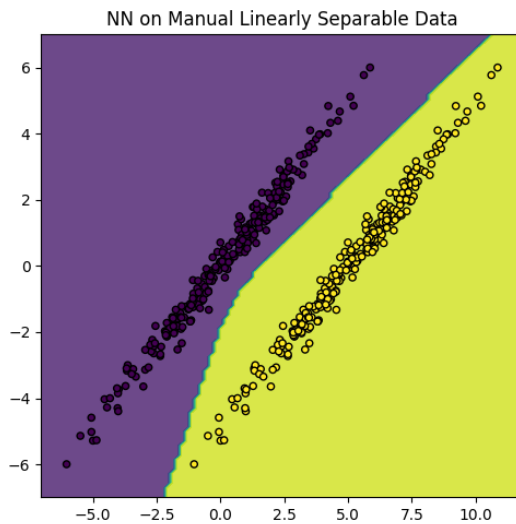
Two-layer neural networks



- Linear neural network: $S(x) = w_2^T (W_1 x + \mathbf{b}_1) + b_2$
 - Still a linear model at the end of the day, so let's add a nonlinearity σ !
- Two-layer MLP: $S(x) = w_2^T \sigma(W_1 x + \mathbf{b}_1) + b_2$
 - Linear model w.r.t. to a learnable feature map
- RBF-kernel: $S(x) = w_2^T \exp(i (W_1 x + \mathbf{b}_1)) + b_2$
 - Kernels are infinite width neural networks with fixed weights
 - By Bochner's theorem, see, e.g., [\[Rahimi and Recht, 2007\]](#)

Results of fitting MLPs on our three examples

```
from sklearn.neural_network import MLPClassifier
hidden = 100
# Neural network classifier
nn_clf = MLPClassifier(hidden_layer_sizes=(hidden,), activation='relu', max_iter=1000)
```



Next lecture

- Finish neural networks / deep learning
- Start unsupervised learning