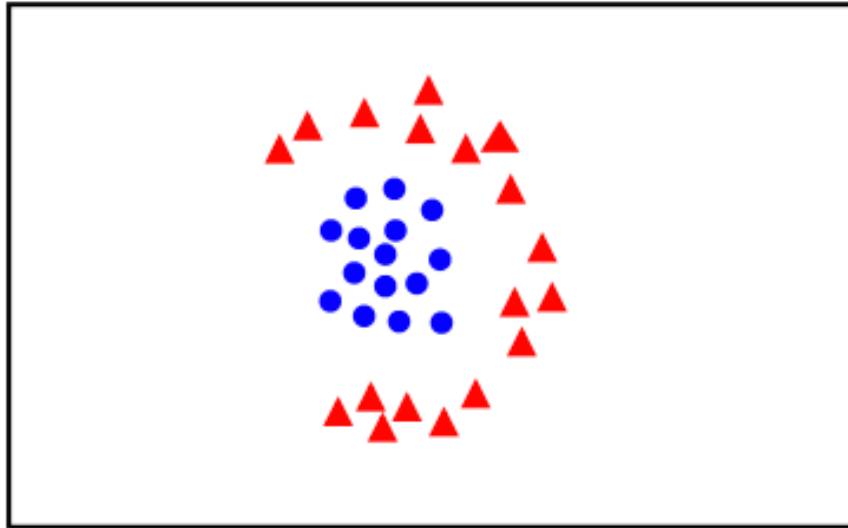


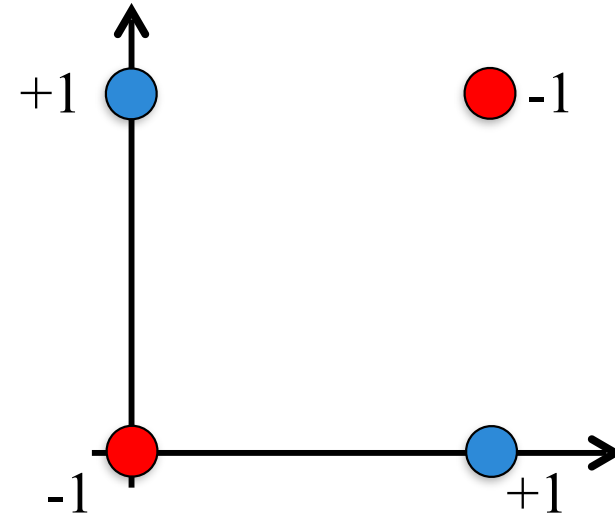
Support Vector Machines

Pascal Fua
IC-CVLab

Non-Linearly Separable Data



Adaboost can handle this using linear classifiers.



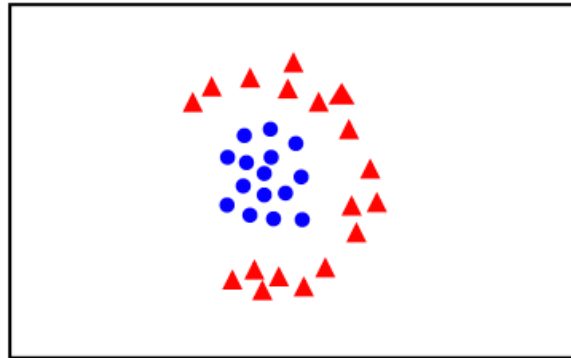
But not this.

—> Map the data to a higher dimension.

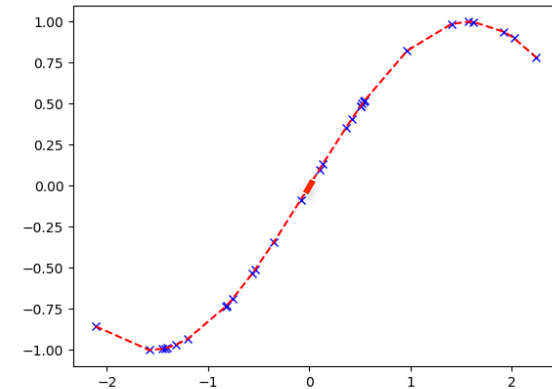
Mapping to a Higher Dimension: Three Examples



1D classification.



2D classification.



Polynomial approximation.

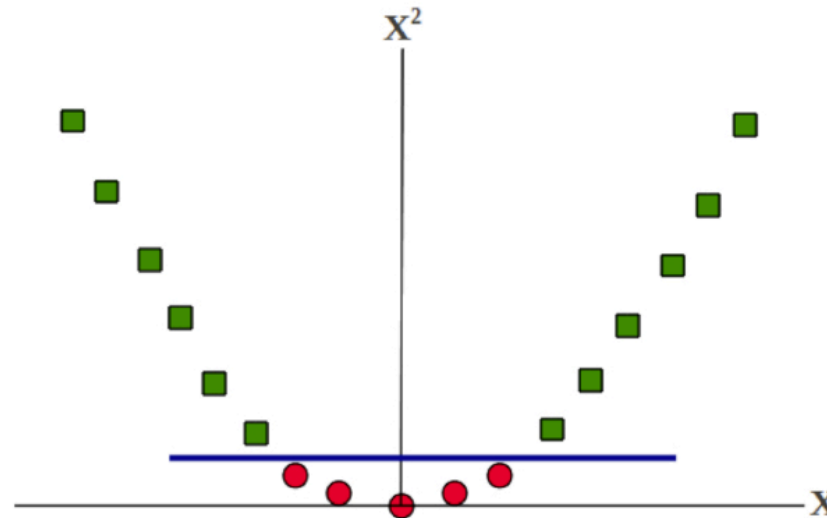
1D Classification Example

How can we handle this 1D/2-class data?



We can map it to 2D:

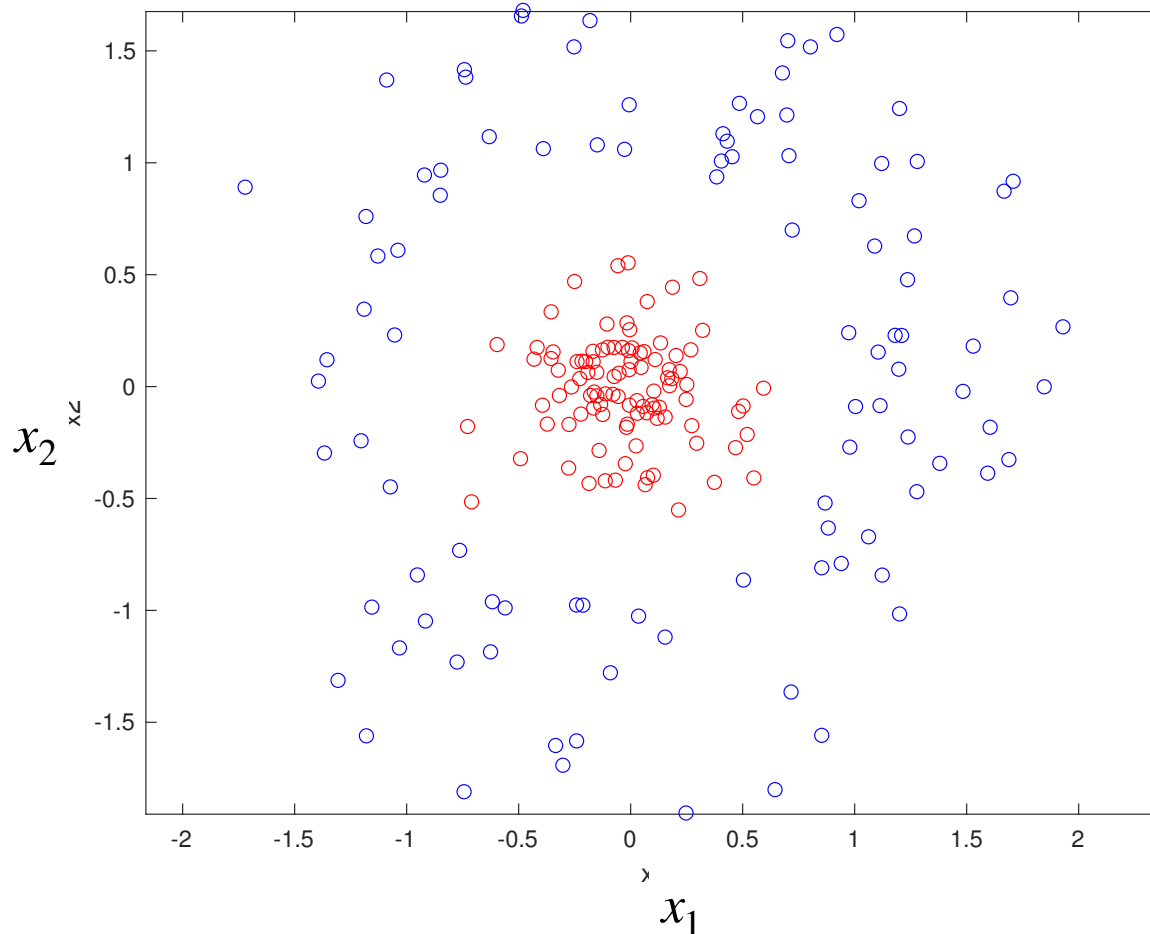
$$x \rightarrow \phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$



—> We can now use a linear classifier.

2D Classification Example

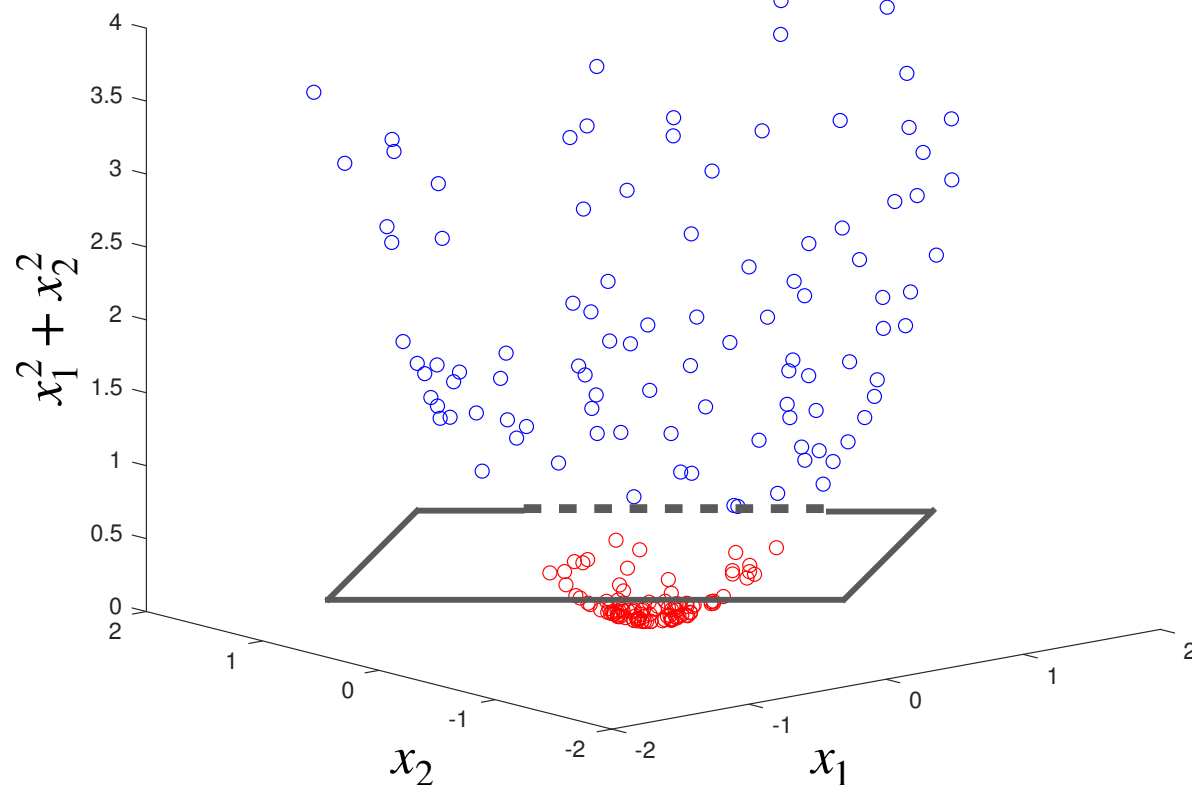
How about this 2D/2-class data?



2D Classification Example

We can map the 2D data to 3D:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \phi(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix}$$



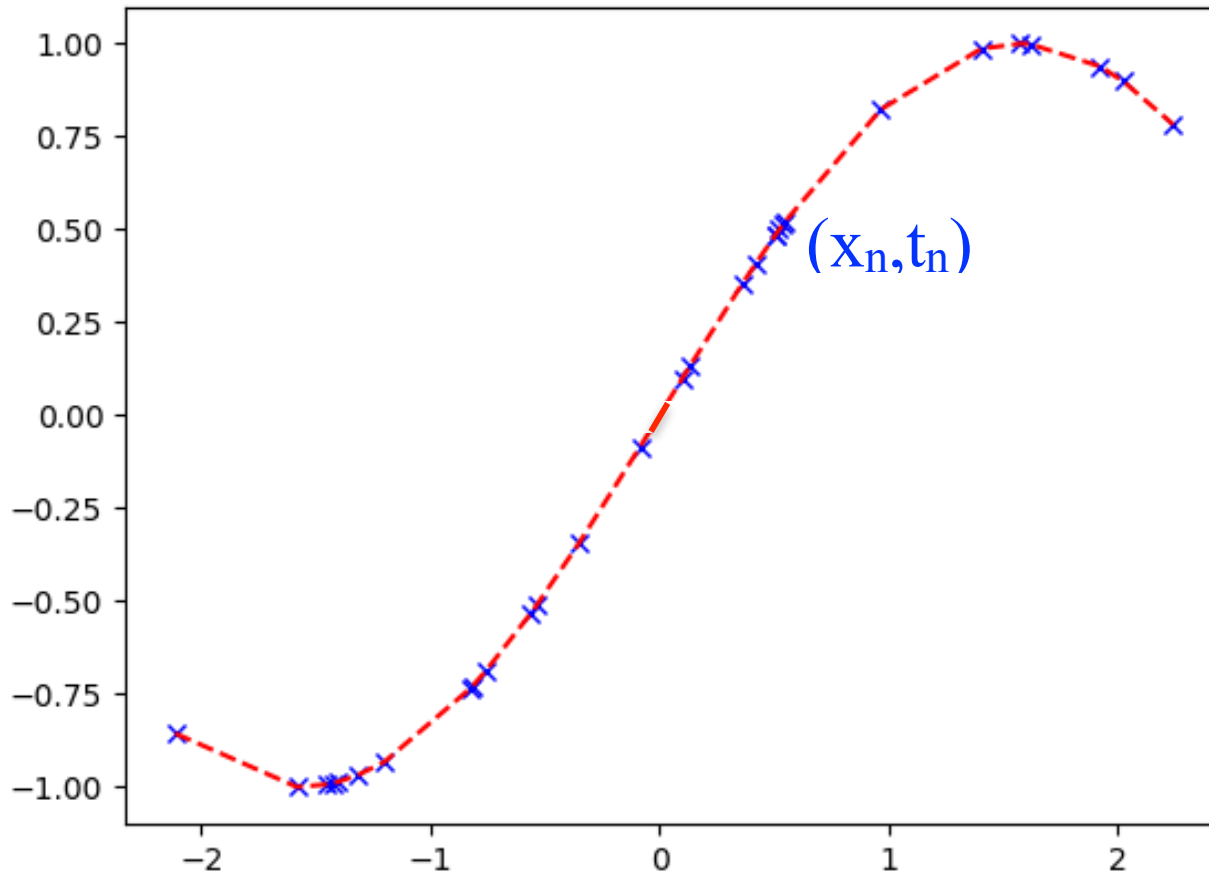
→ We can now use a linear classifier.

Lifting from 2D to 3D

*SVM with a polynomial
Kernel visualization*

*Created by:
Udi Aharoni*

Polynomial Approximation



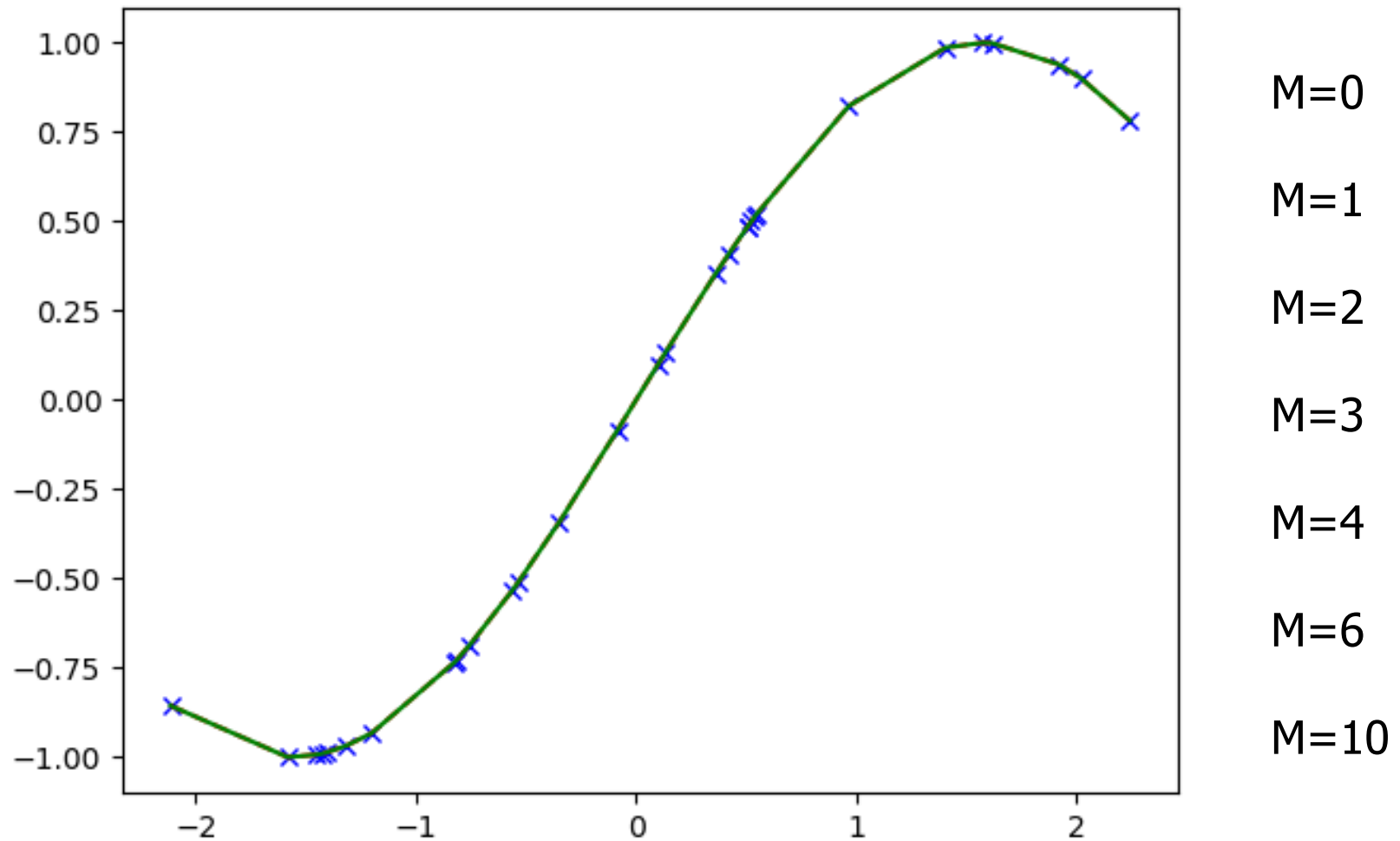
For $1 \leq n \leq N$:

$$t_n = f(x_n) + \varepsilon$$

- The (x_i, t_i) are given.
- f is unknown.

- Find $\mathbf{w} = [w_0, w_1, \dots, w_M]$ such that: $\forall x, f(x) \approx \sum_{i=0}^M w_i x^i$
- Least squares solution: $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \sum_n (t_n - \sum_{i=0}^M w_i x_n^i)^2$
- For $M=1$, reduces to linear regression.

Polynomial Approximation



For a given M, we plot in green:

$$f_M(x) = \sum_{i=0}^M w_i * x^i$$

Polynomial Feature Expansion

$$x \rightarrow \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix}$$

The polynomial can be rewritten as:

$$\sum_{i=0}^M w_i x^i = \mathbf{w} \cdot \phi(x) = \mathbf{w}^T \phi(x) \text{ with } \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix}$$

The least squares solution becomes:

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \sum_n (t_n - \mathbf{w}^T \phi(x_n))^2$$

Least-Squares Formulation

$$\begin{aligned}\mathbf{w}^* &= \arg \min_{\mathbf{w}} \sum_{n=1}^N (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))^2 \\ &= \arg \min_{\mathbf{w}} \|\Phi \mathbf{w} - \mathbf{t}\|^2\end{aligned}$$

with

$$\Phi = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^M \\ 1 & x_2 & x_2^2 & \dots & x_2^M \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_N & x_N^2 & \dots & x_N^M \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_M \end{bmatrix}, \quad \text{and } \mathbf{t} = \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ \dots \\ t_N \end{bmatrix}.$$

$$\text{Intuitively:} \quad \Rightarrow \Phi \mathbf{w}^* \approx \mathbf{t}$$

$N \times \tilde{M}$ $\tilde{M} \times 1$ $N \times 1$

$$\text{Formally:} \quad \Rightarrow (\Phi^T \Phi) \mathbf{w}^* = \Phi^T \mathbf{t}$$

$\tilde{M} \times \tilde{M}$ $\tilde{M} \times 1$ $\tilde{M} \times 1$

Optional: Proof Sketch

We want to minimize:

$$\begin{aligned} R &= \frac{1}{2} \|\Phi \mathbf{w} - \mathbf{t}\|^2 \\ &= \frac{1}{2} (\Phi \mathbf{w} - \mathbf{t})^T (\Phi \mathbf{w} - \mathbf{t}) \end{aligned}$$

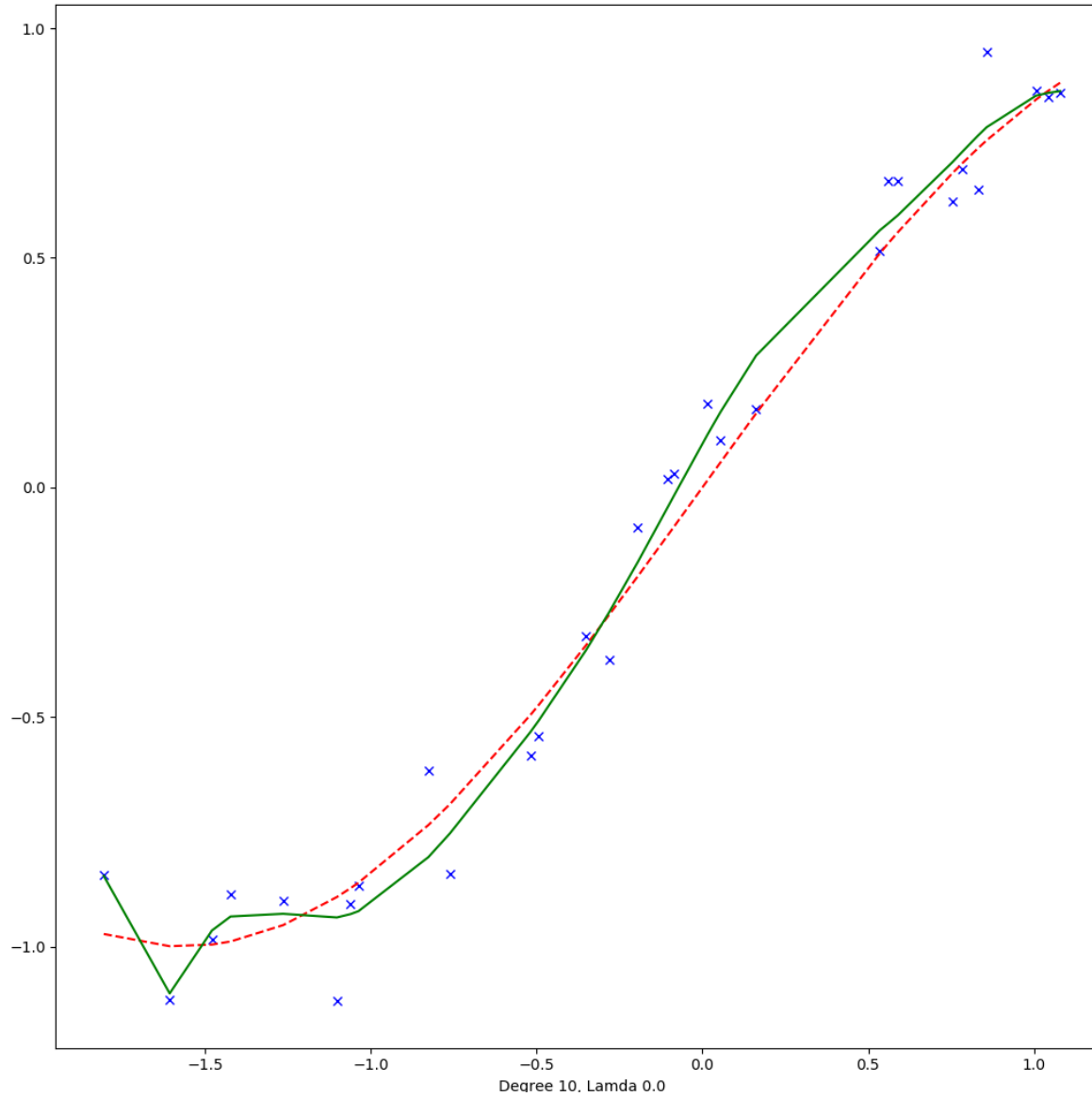
The gradient of R w.r.t \mathbf{w} is:

$$\nabla R = \Phi^T (\Phi \mathbf{w} - \mathbf{t})$$

At the minimum:

$$\begin{aligned} 0 &= \nabla R = \Phi^T (\Phi \mathbf{w} - \mathbf{t}) \\ \Rightarrow \Phi^T \Phi \mathbf{w} &= \Phi^T \mathbf{t} \end{aligned}$$

Adding Noise



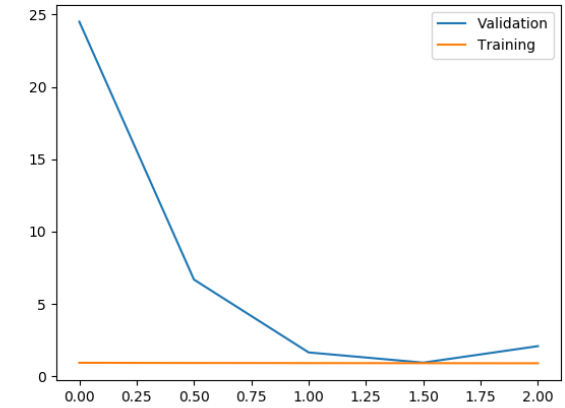
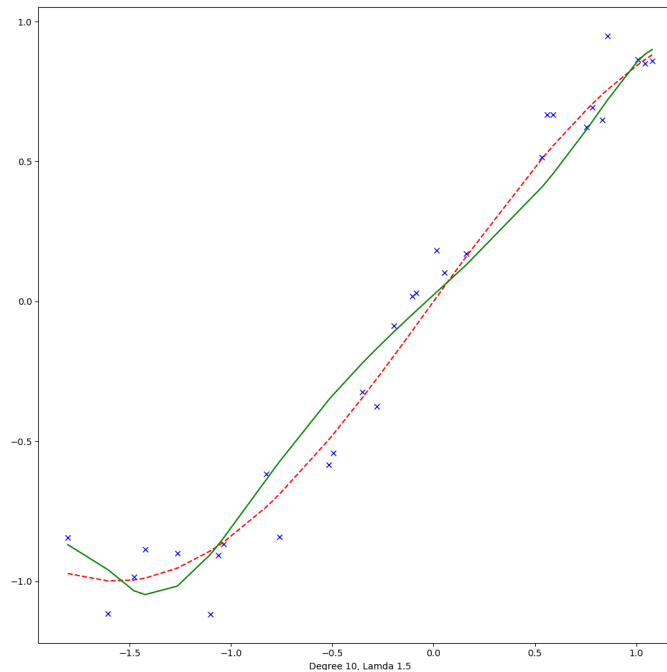
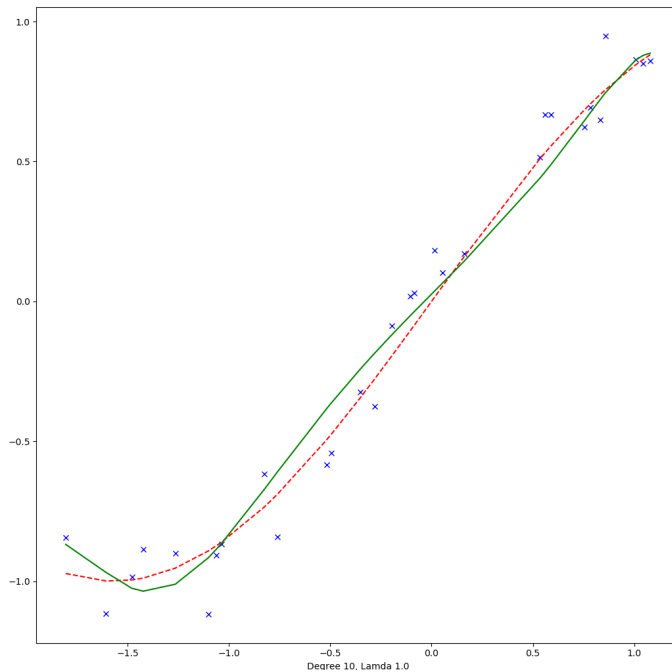
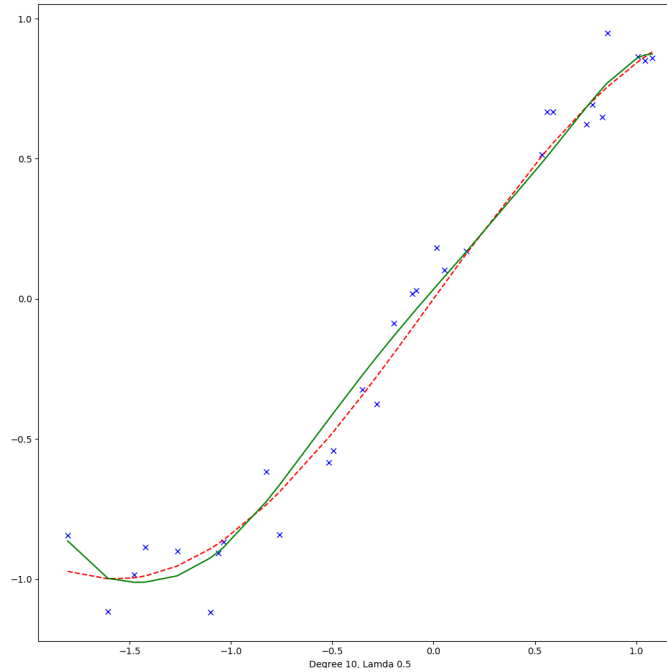
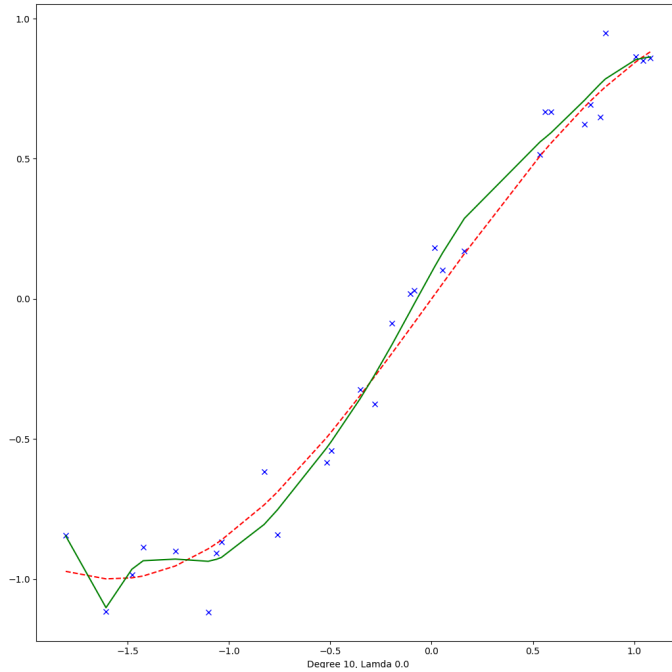
Regularization

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \|\Phi \mathbf{w} - \mathbf{t}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

$$\Rightarrow \text{Solve: } (\Phi^T \Phi + \lambda \mathbf{I}) \mathbf{w} = \Phi^T \mathbf{t}$$

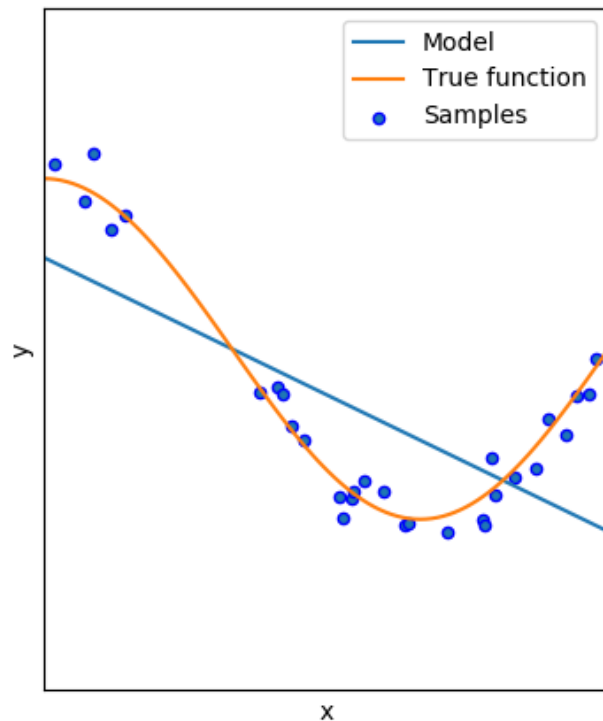
- This is known as weight decay because in iterative algorithms it encourages the weight values to decay to zero, unless supported by the data.
- It discourages large weights and therefore quick variations.

Without and with Regularization

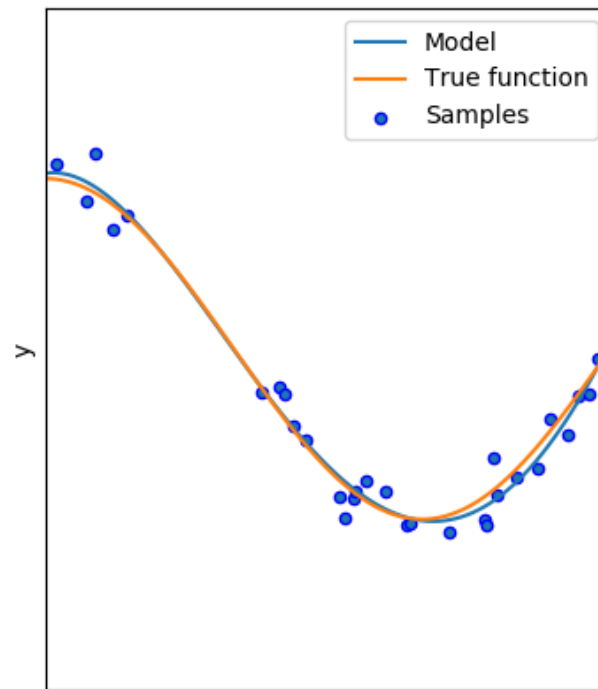


Use cross-validation data to select the value of λ .

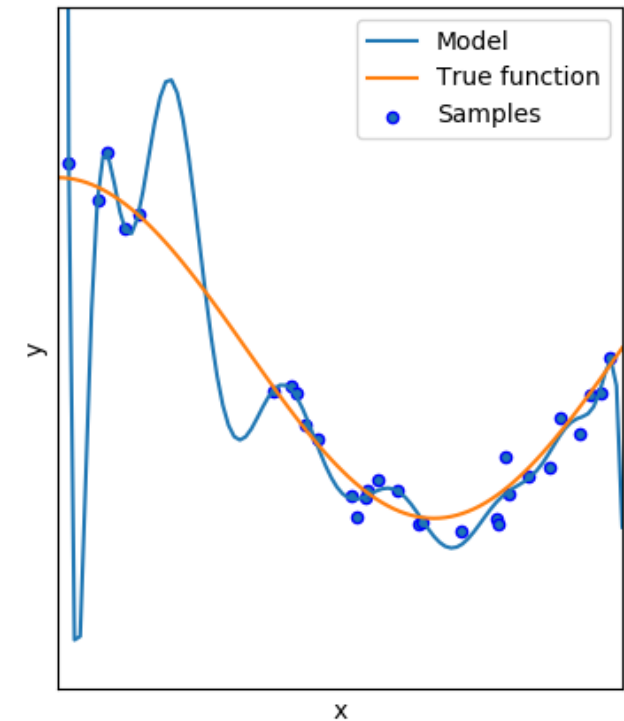
Linear and Non-Linear Regression



Order 1



Order 4

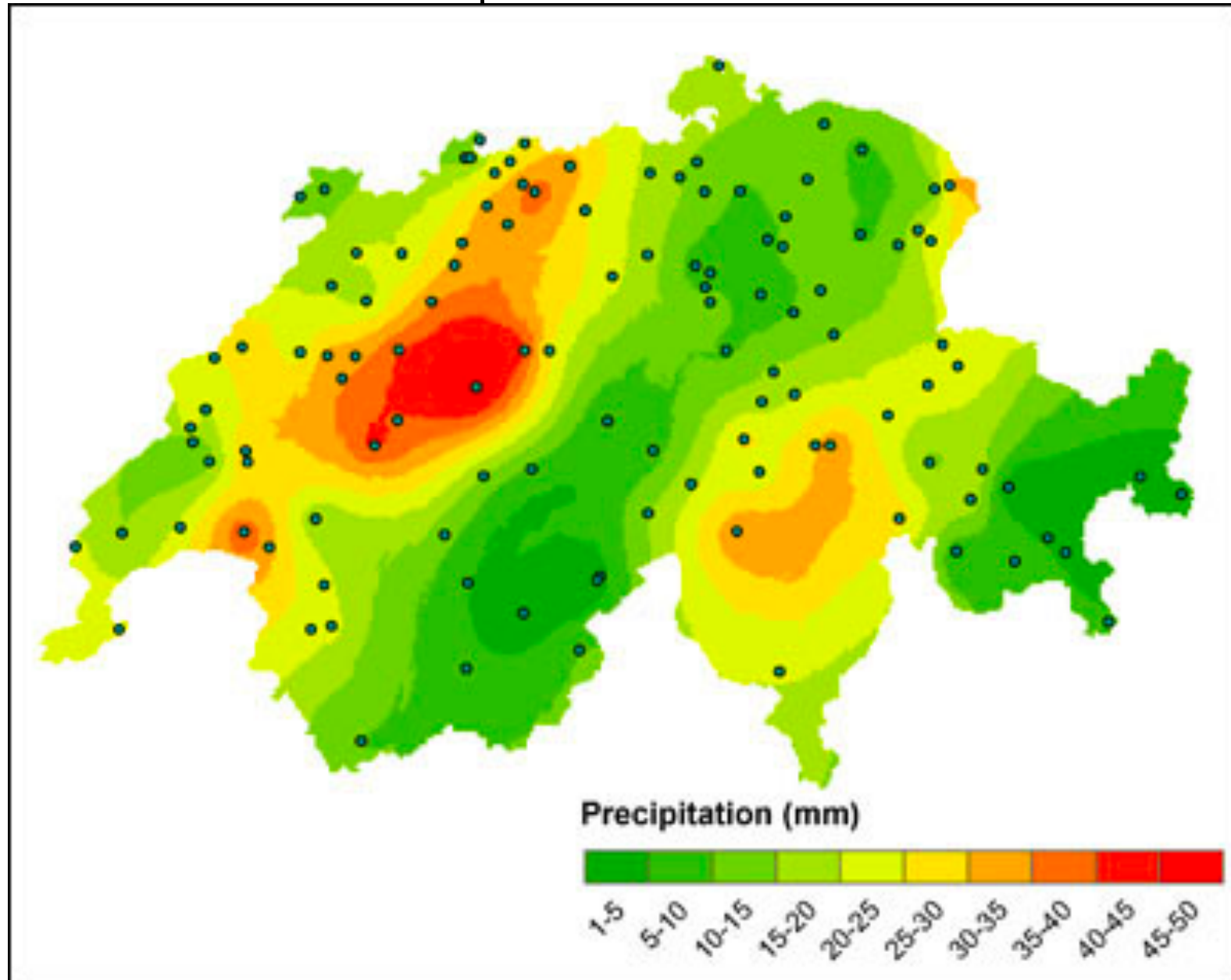


Order 15

For both kind of regressions, the trick is to find the best compromise between simplicity and goodness of fit.

Application: Rainfall in Switzerland

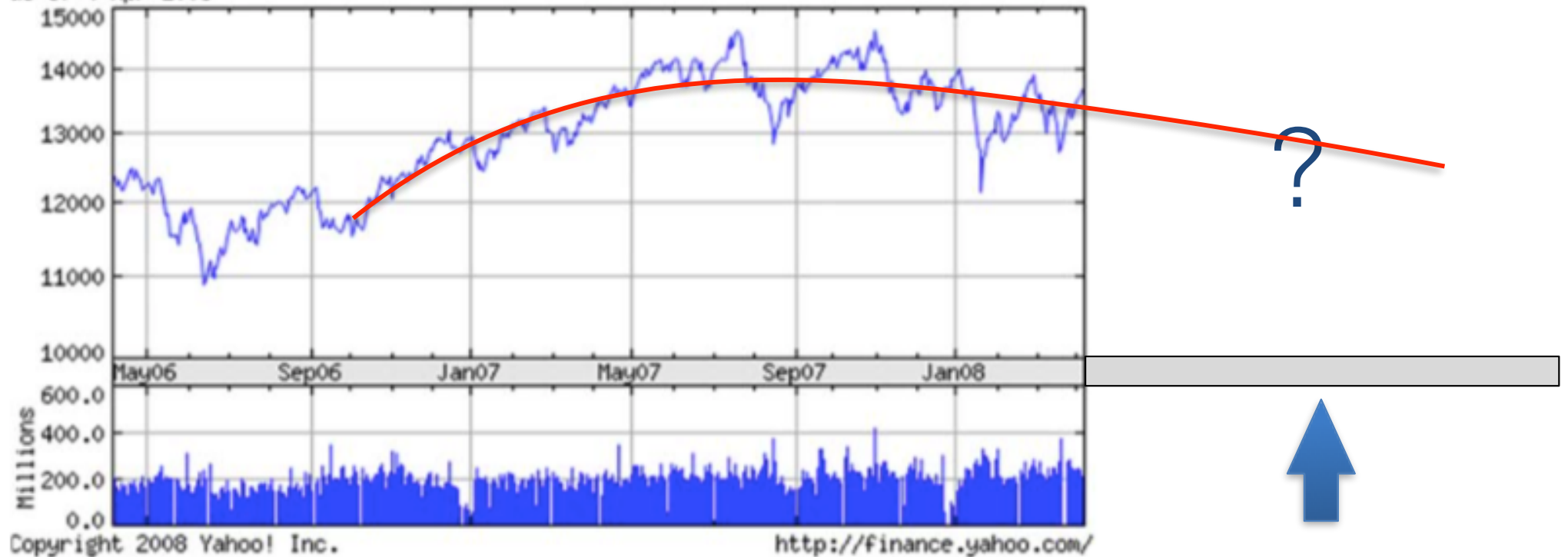
The circles represent actual measurements



—> Extends to Higher Dimensions.

Application: Stock Price Prediction

S&P/TSX COMPOSITE
as of 4-Apr-2008



\mathbf{x}_t

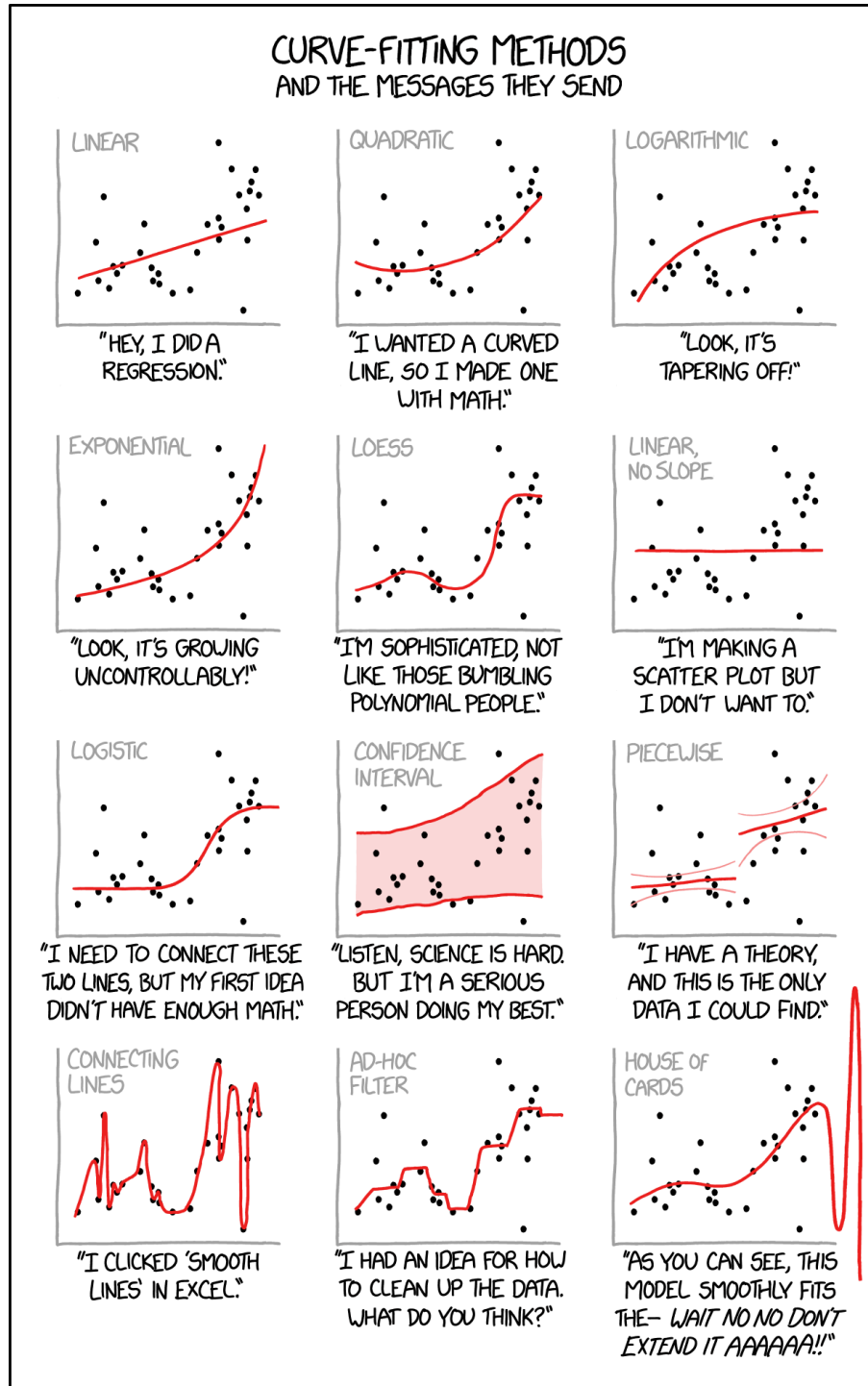
$x_{t+\Delta t}$

$$\mathbf{x}_t = [x_{t-T+1}, \dots, x_{t-1}, x_t]$$

$$y(\mathbf{x}_t; \mathbf{w}) = x_{t+\Delta t}$$

—> Regression problem

But Be Careful!



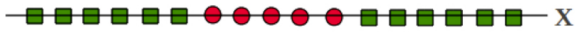
Never trust a
statistic you have
not faked yourself!

But Be VERY Careful!

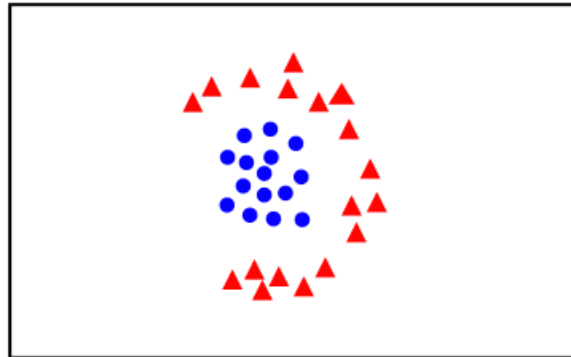


March 2019 to March 2020

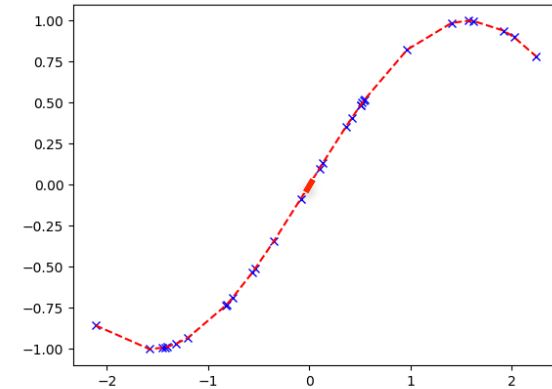
Recap: Mapping to a Higher Dimension



1D classification.



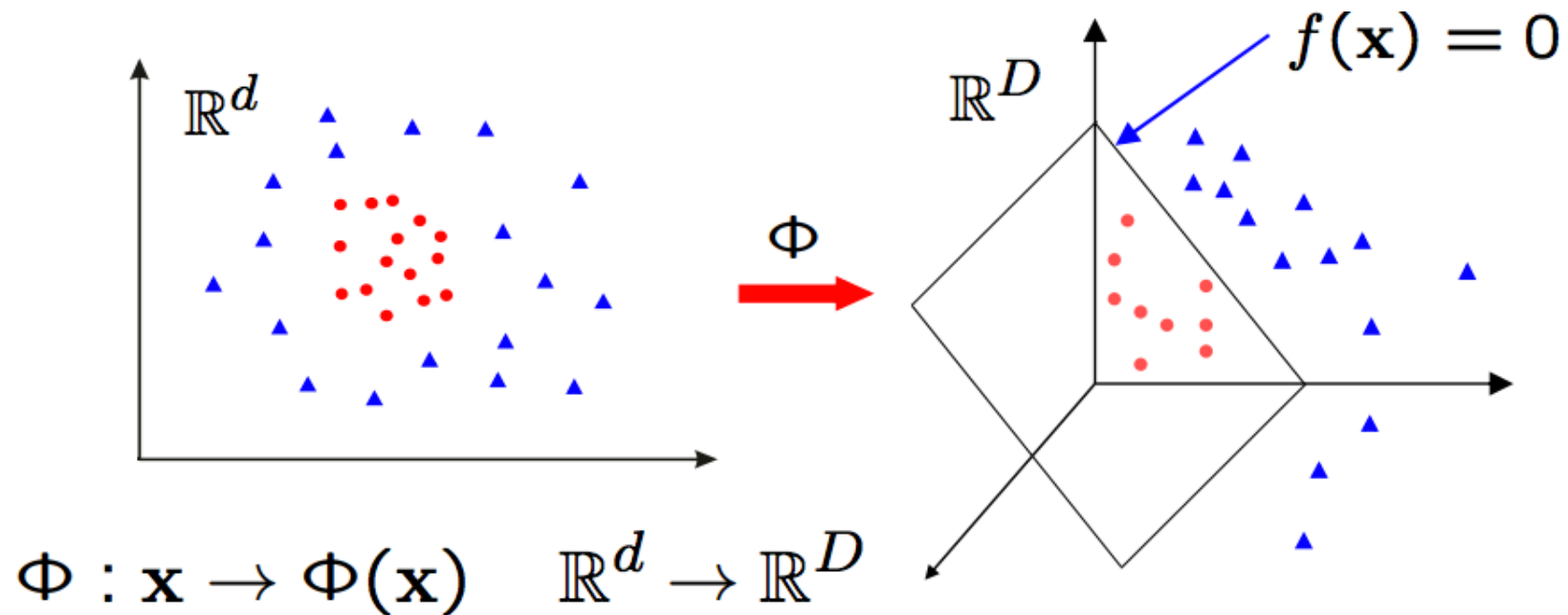
2D classification.



Polynomial approximation.

- We have seen three examples in which mapping to a higher dimension makes the problem linear.
- This idea can be generalized.

Classification in Feature Space



- Map from \mathbb{R}^d to \mathbb{R}^D
- Learn a linear classifier in \mathbb{R}^D

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T \phi(\mathbf{x}) + w_0)$$

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$$

Polynomial Feature Expansion

1-Dimensional Input

$$x \rightarrow \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix}$$

d-Dimensional Input

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1^2 \\ \vdots \\ x_d^2 \\ \vdots \\ x_1^M \\ \vdots \\ x_d^M \\ x_1x_2 \\ \vdots \\ x_1x_d \\ \vdots \\ x_{d-1}x_d \\ x_1^2x_2 \\ \vdots \end{bmatrix}$$

constant

linear

quadratic

M^{th} power

Products of 2 variables

All other possible monomials

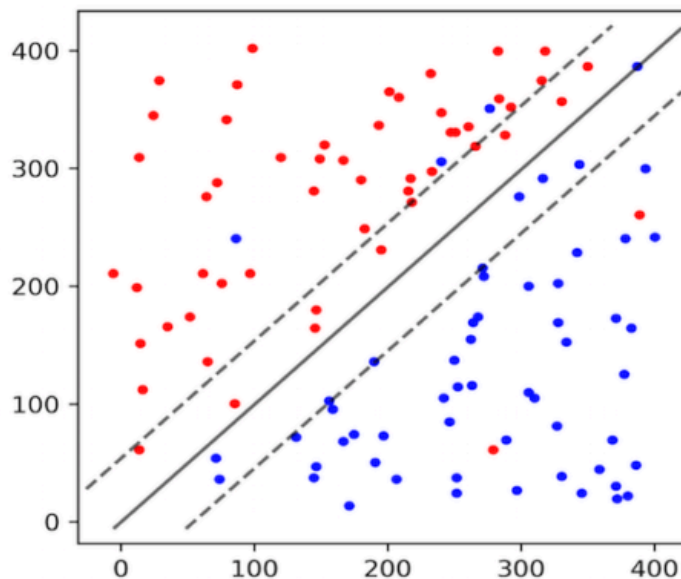
- The dimension of $\phi(\mathbf{x})$ grows quickly with the degree M of the polynomial.
- $\phi(\mathbf{x})$ can be used in any algorithm that we have seen so far.

Reminder: Linear SVM

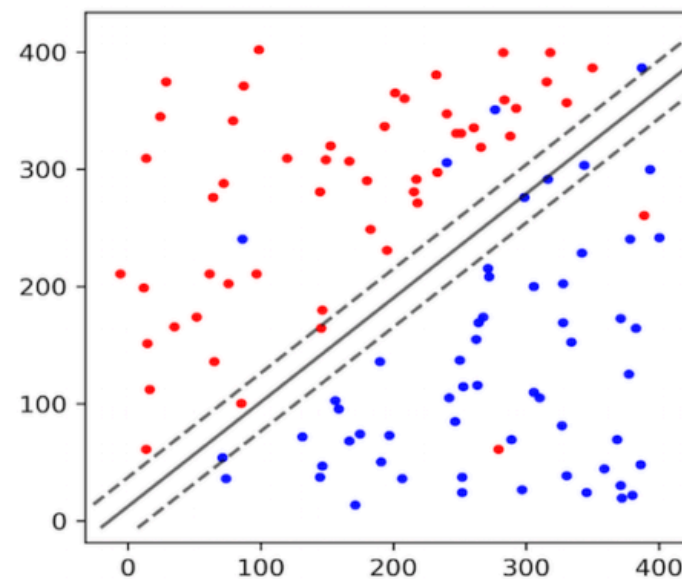
$$\mathbf{w}^* = \min_{(\mathbf{w}, \{\xi_n\})} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n,$$

subject to $\forall n, \quad t_n \cdot (\tilde{\mathbf{w}} \cdot \mathbf{x}_n) \geq 1 - \xi_n$ and $\xi_n \geq 0$.

- C is constant that controls how costly constraint violations are.
- The problem is still convex.



$C=1$



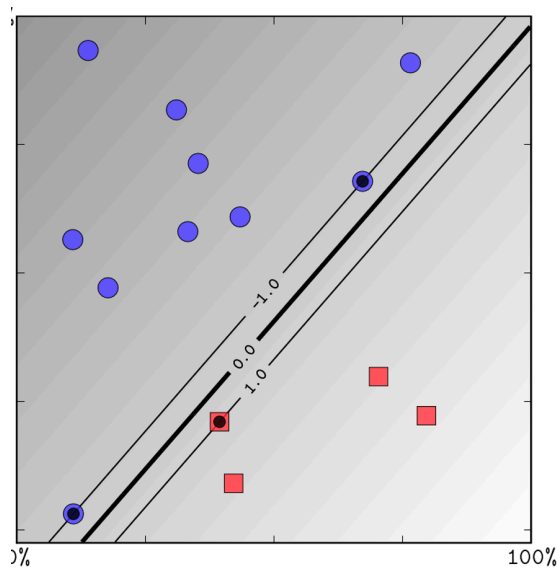
$C=100$

Polynomial SVM

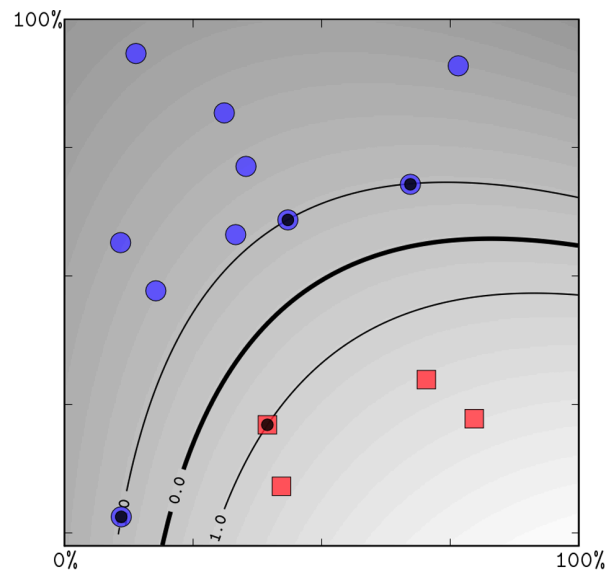
$$\mathbf{w}^* = \min_{(\mathbf{w}, \{\xi_n\})} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n,$$

subject to $\forall n, \quad t_n \cdot (\tilde{\mathbf{w}} \cdot \phi(\mathbf{x}_n)) \geq 1 - \xi_n$ and $\xi_n \geq 0$.

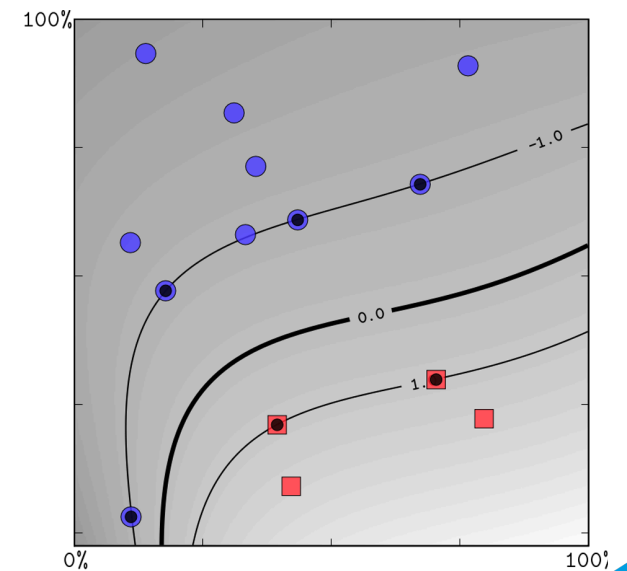
- C is constant that controls how costly constraint violations are.
- The problem is still convex.



M = 1

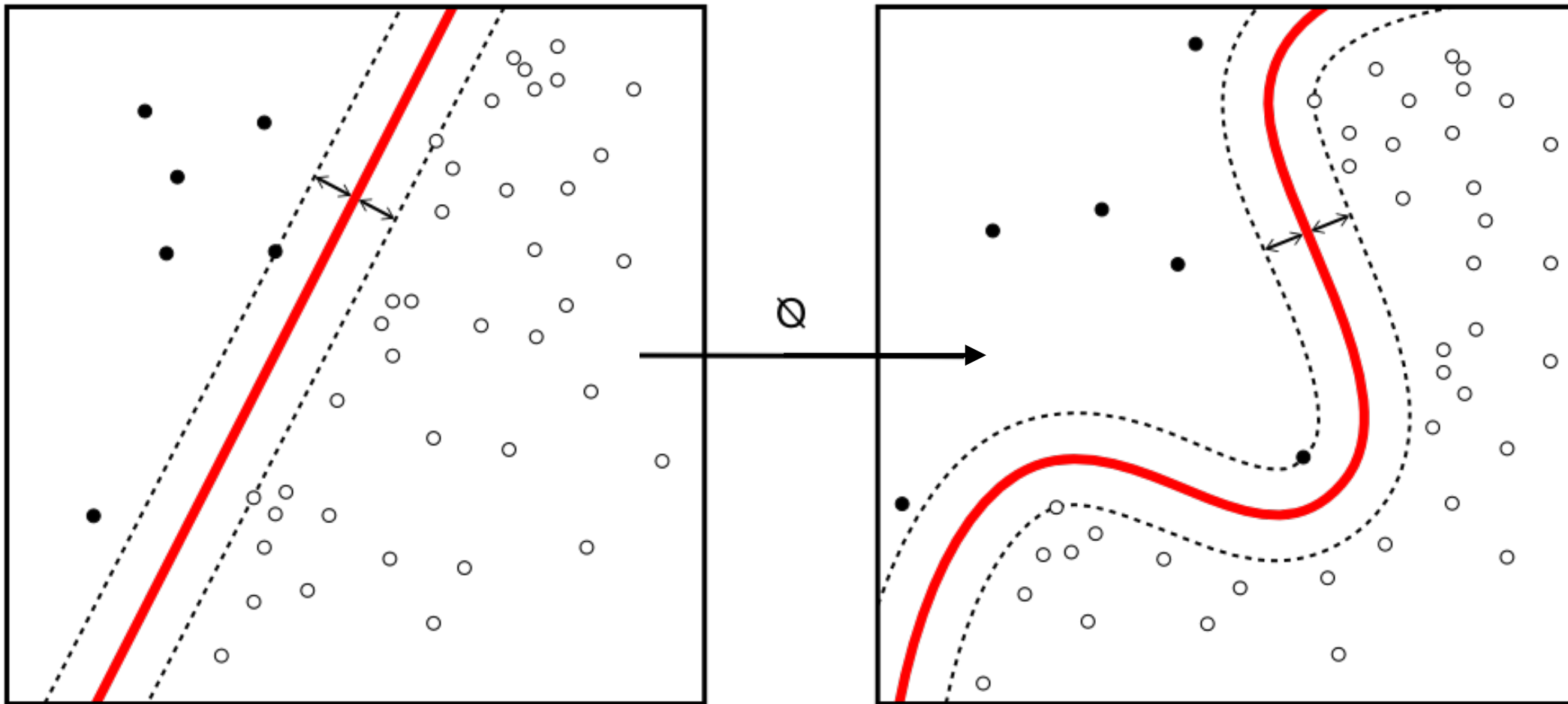


M = 2

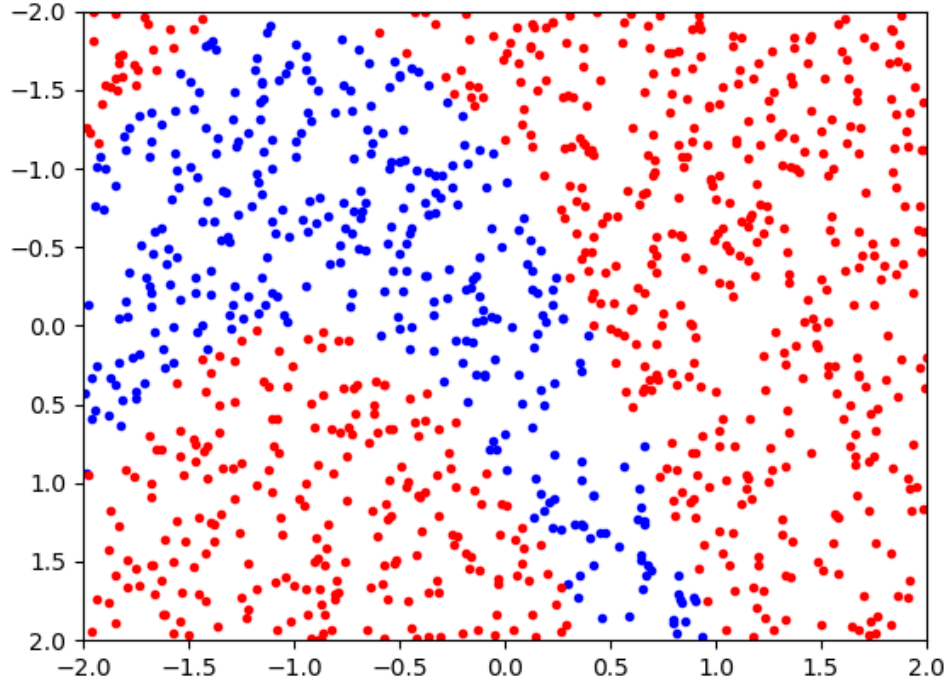


M = 5

Interpretation

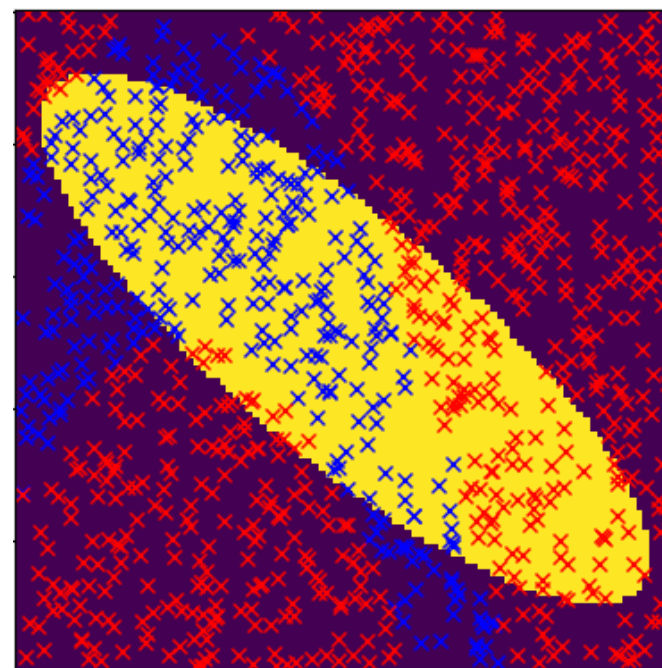


The linear decision boundary in a high dimensional space becomes a curvy one in the original low dimensional space.

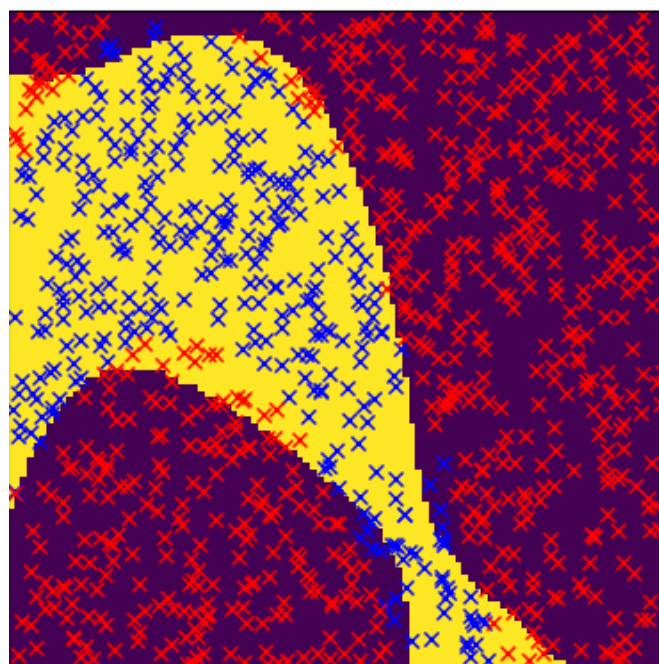


$$r(x, y) = 100 * (y - x^2)^2 + (1 - x)^2$$

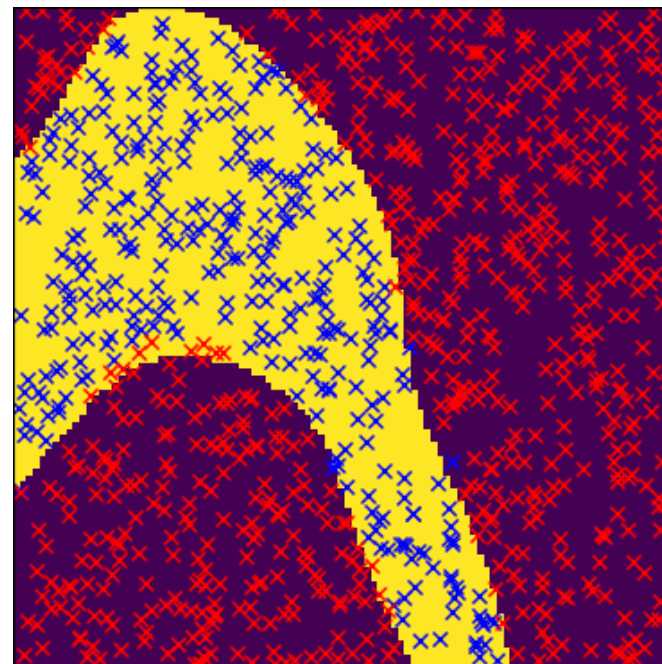
Rosenbrock:
$$f(x, y) = \begin{cases} -1 & \text{if } r(x, y) < T \\ 1 & \text{otherwise} \end{cases}$$



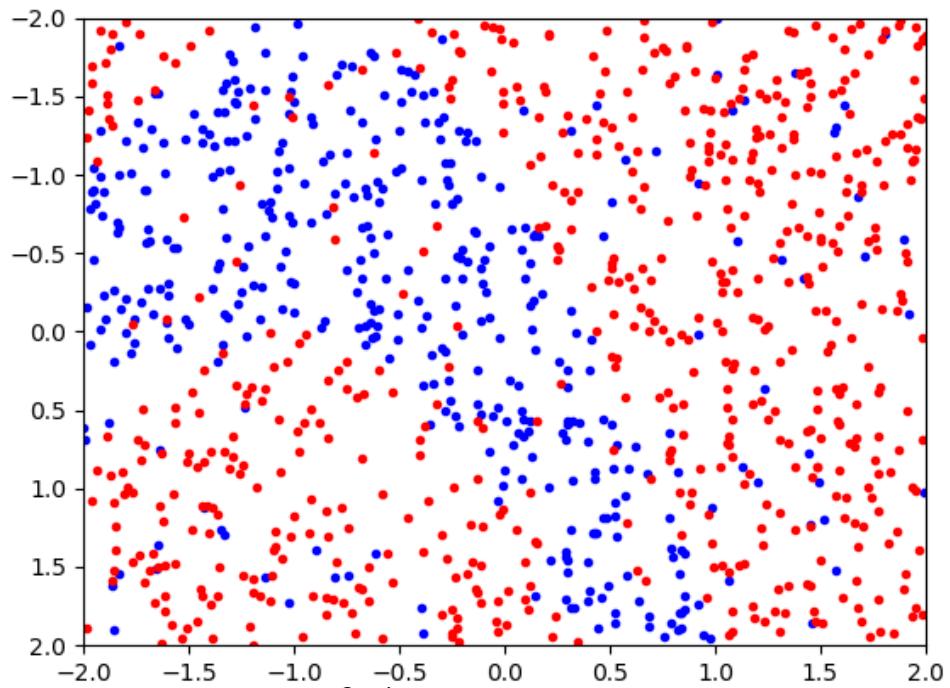
$$[x, y, xy]$$



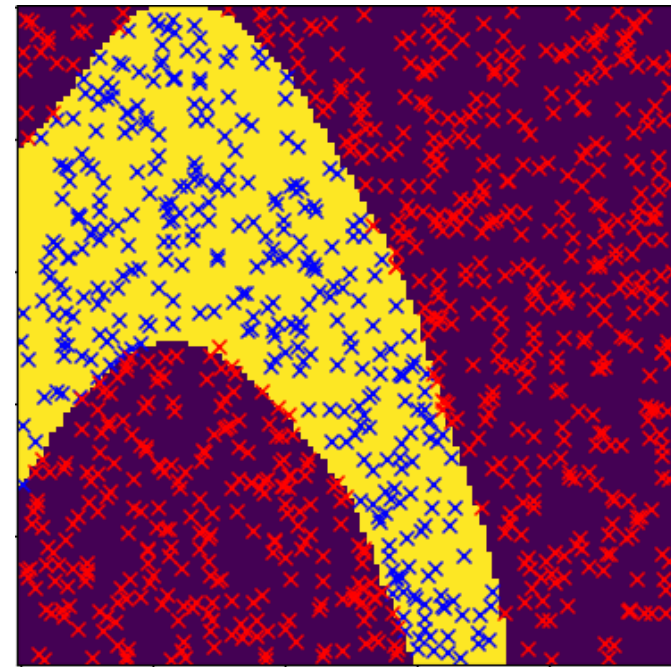
$$[x, y, x^2, \dots, xy^2, y^3]$$



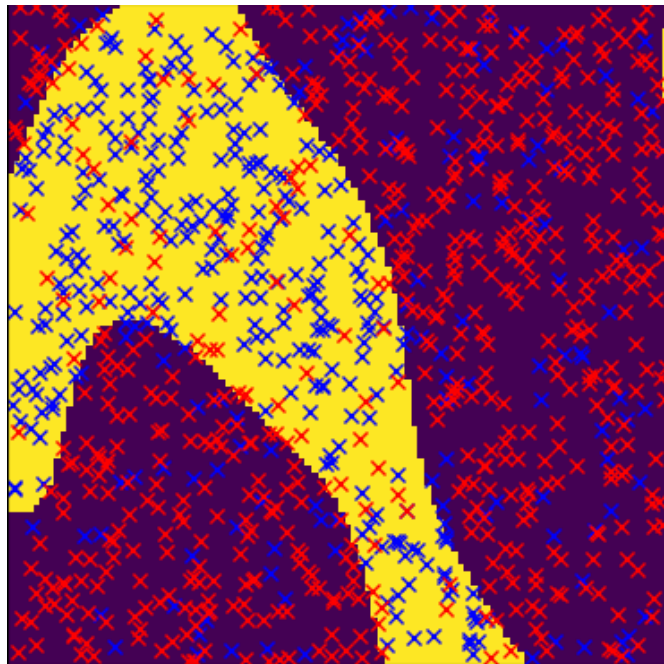
$$[x, y, x^2, \dots, xy^3, y^4]$$



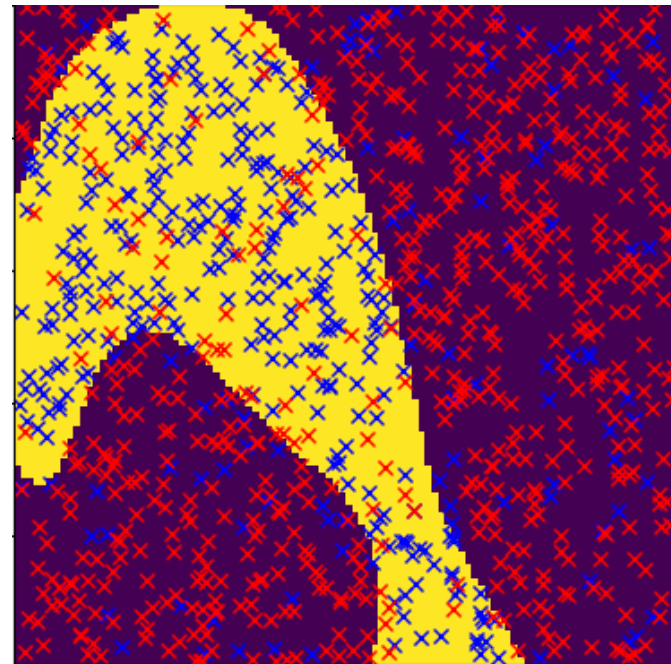
10% noise



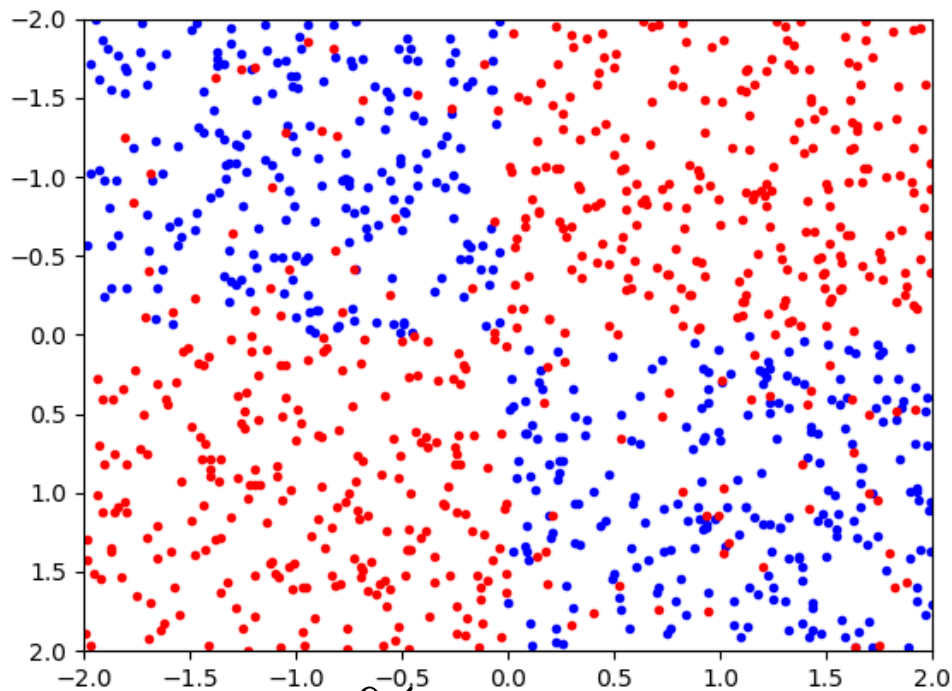
$[x, y, x^2, \dots, xy^7, y^8]$



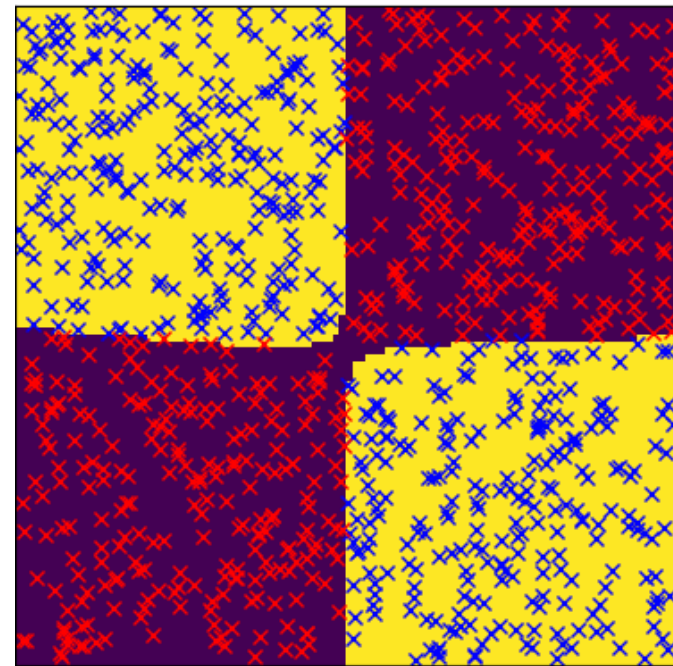
5% noise



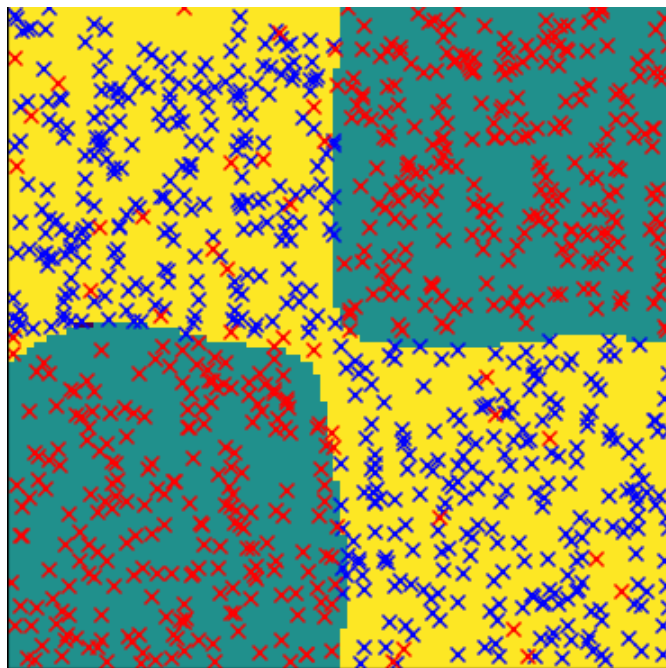
10% noise



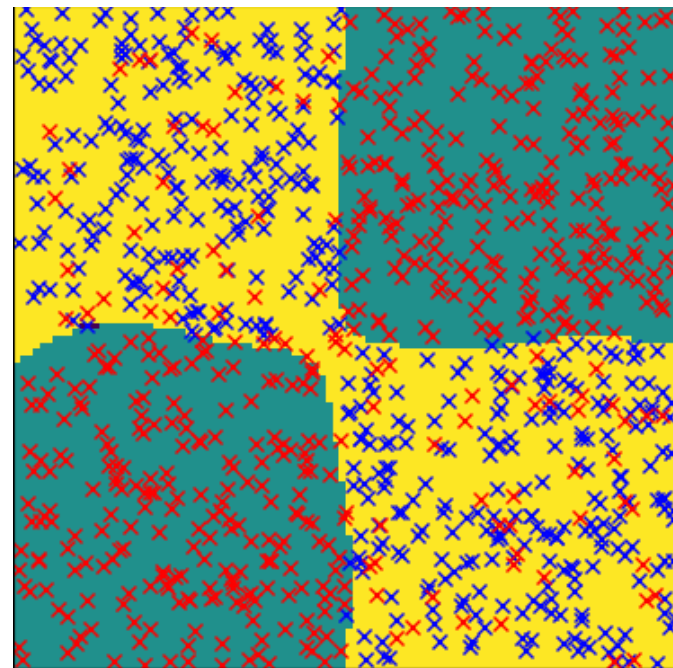
15%noise



$[x, y, x^2, \dots, xy^7, y^8]$



5%noise



10%noise

Polynomial SVM

- A higher-degree polynomial expansion yields a more flexible boundary.
- It also increases the dimensionality of the problem.
- The computational complexity of training SVMs grows like the cube of the dimension.

—> Inherent limitation of polynomial SVMs.

Another Way to Map to a Higher Dimension



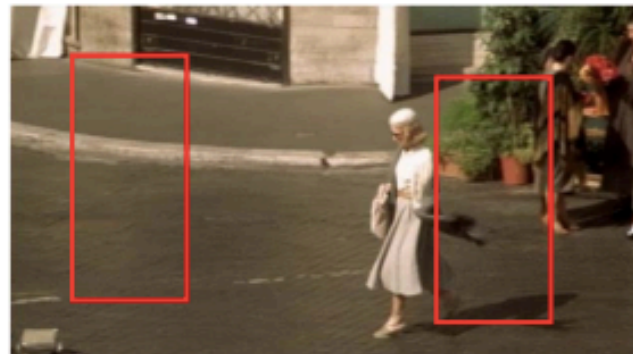
People Detection in Images

Training Data

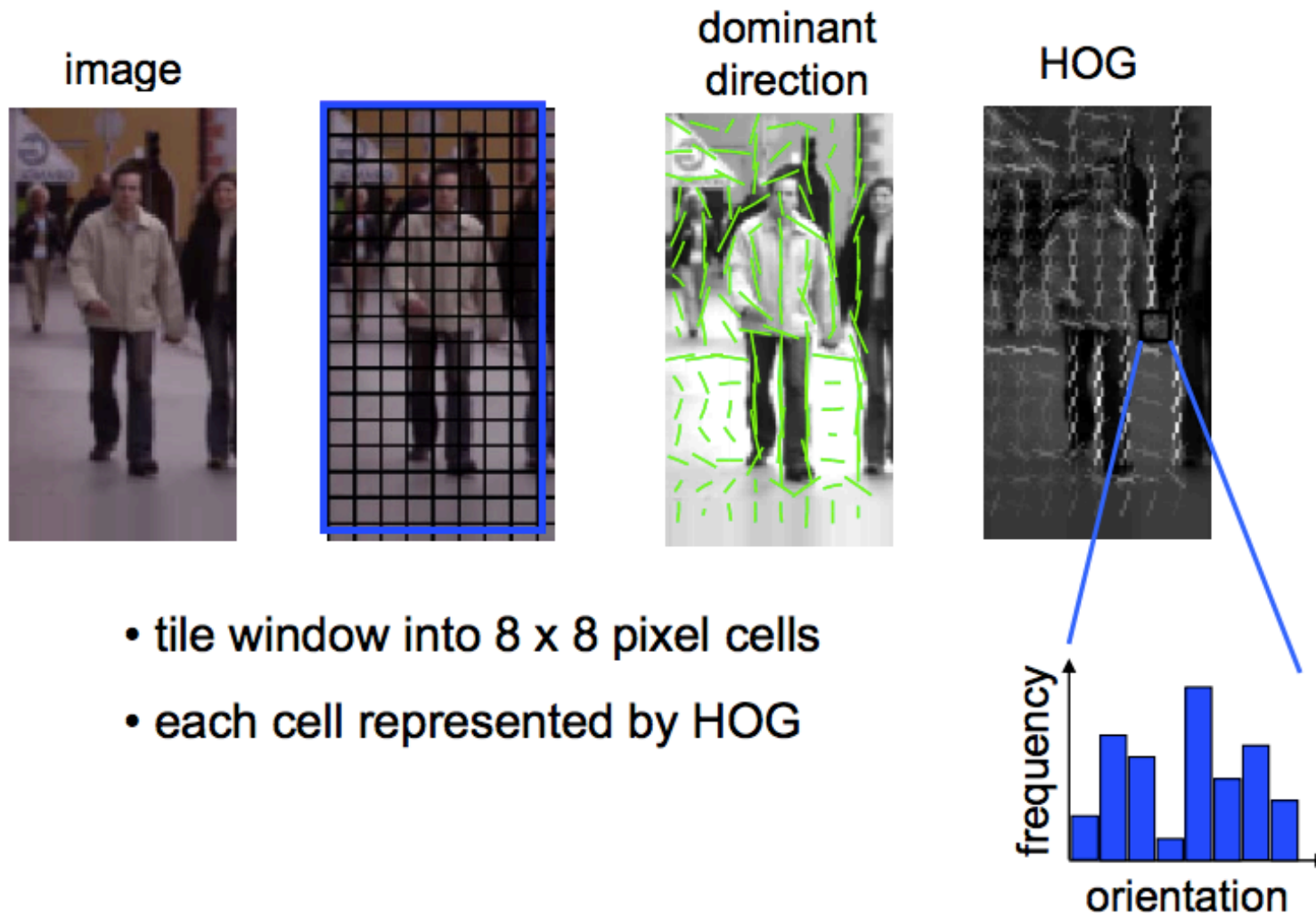
- Positive data – 1208 positive window examples



- Negative data – 1218 negative window examples (initially)



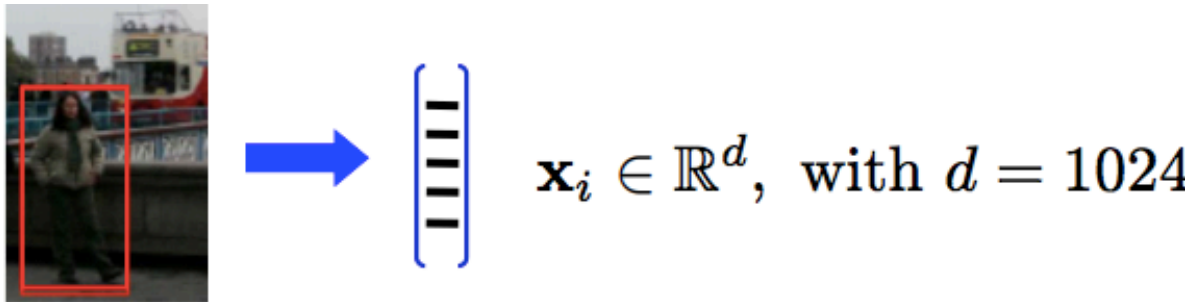
Histogram of Oriented Gradients



Feature vector dimension = 16×8 (for tiling) $\times 8$ (orientations) = 1024

Training:

- Represent each example window by a HOG classifier.



- Train a linear classifier.

Testing:

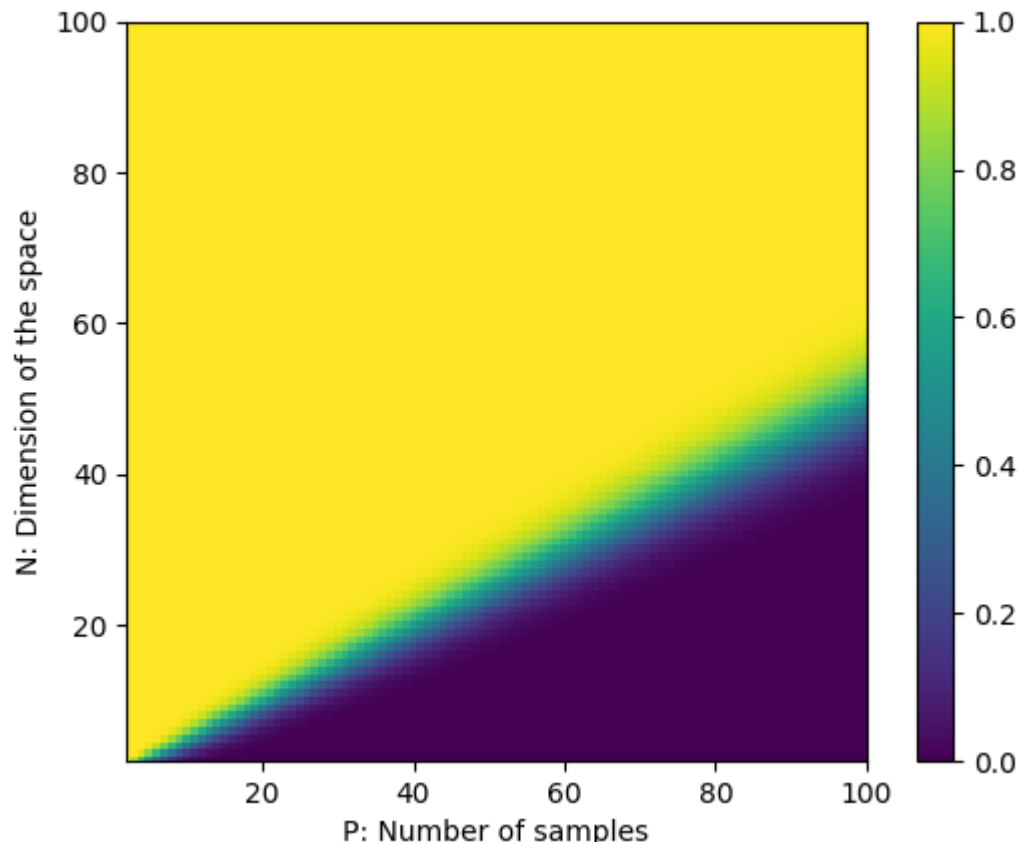
- Sliding window classifier.

$$y(\mathbf{x}; \mathbf{w}, w_0) = \sigma(\mathbf{w}^T \mathbf{x} + w_0)$$

Cover's Theorem

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.

Geometrical and Statistical properties of systems of linear inequalities with applications, 1965



N : Dimension of space

p : Number of samples

$\frac{C(p, N)}{2^p}$: Percentage of separable partitions

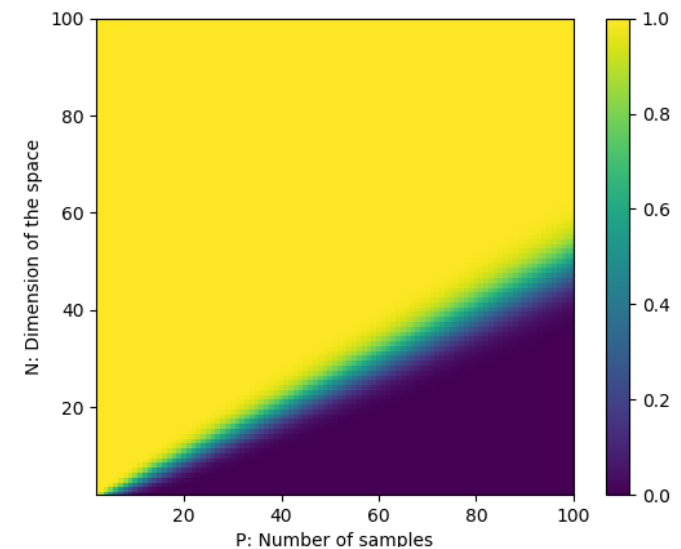
Optional: Recursive Computation

$p \setminus n$	N=1	N=2	N=3
p=1	2	2	2
p=2	3	4	4
p=3	4	7	8
p=4	5	11	15

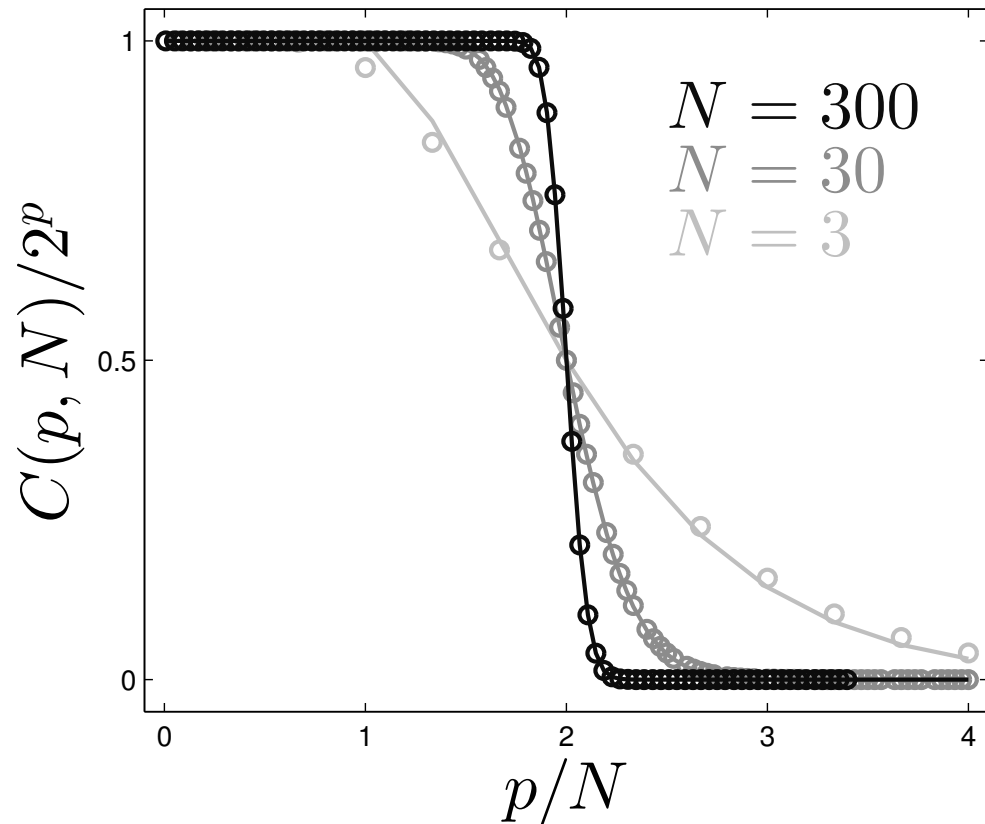
$$\forall n, C(1, n) = 2$$

$$\forall p, C(p, 1) = p + 1$$

$$C(p, N) = C(p - 1, N) + C(p - 1, N - 1)$$

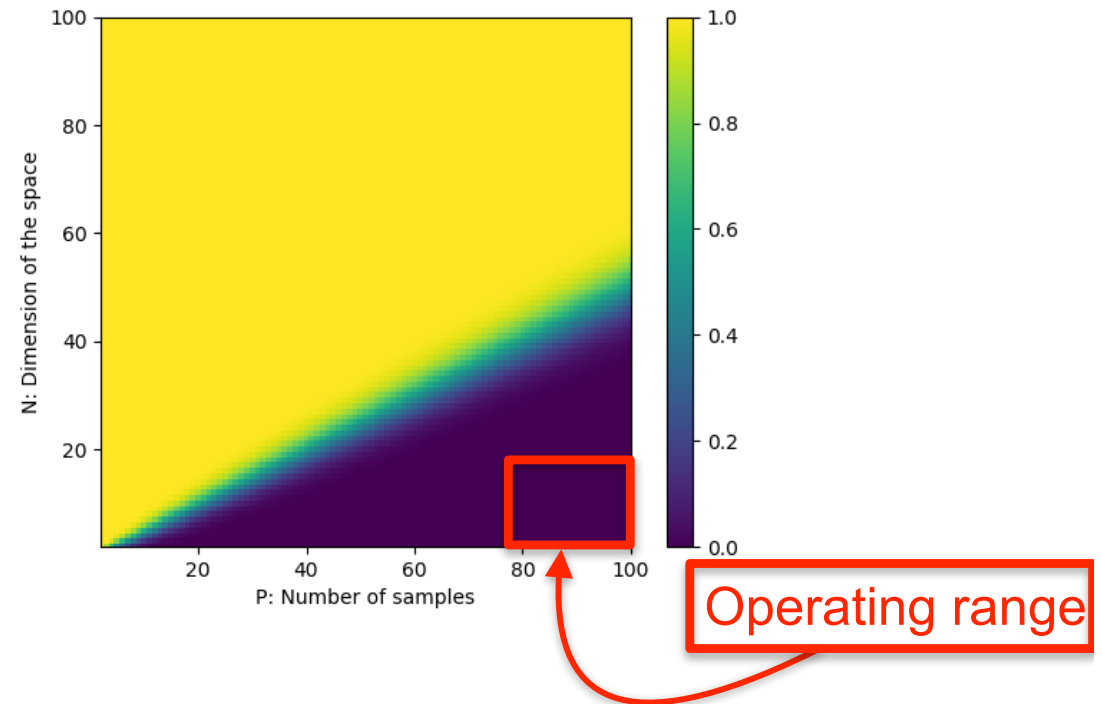
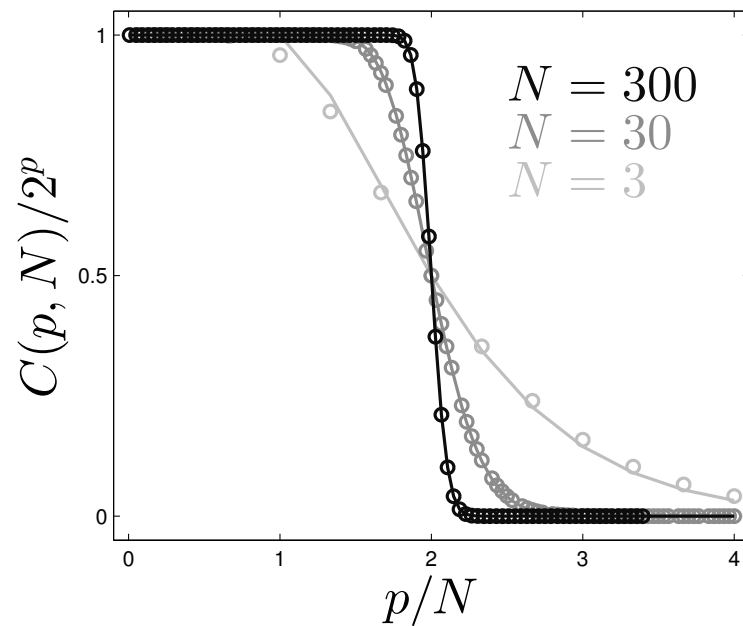


High Dimension is Good



When N is large, almost all partitions are separable if the number p of samples is less than $2N$.

Problem Solved?



- Facebook or Google deal with BILLIONS of images.
- p and therefore N should be of that magnitude.
- Dealing with matrices of dimension $N \times N$ is impractical.



Neither Solved nor Hopeless

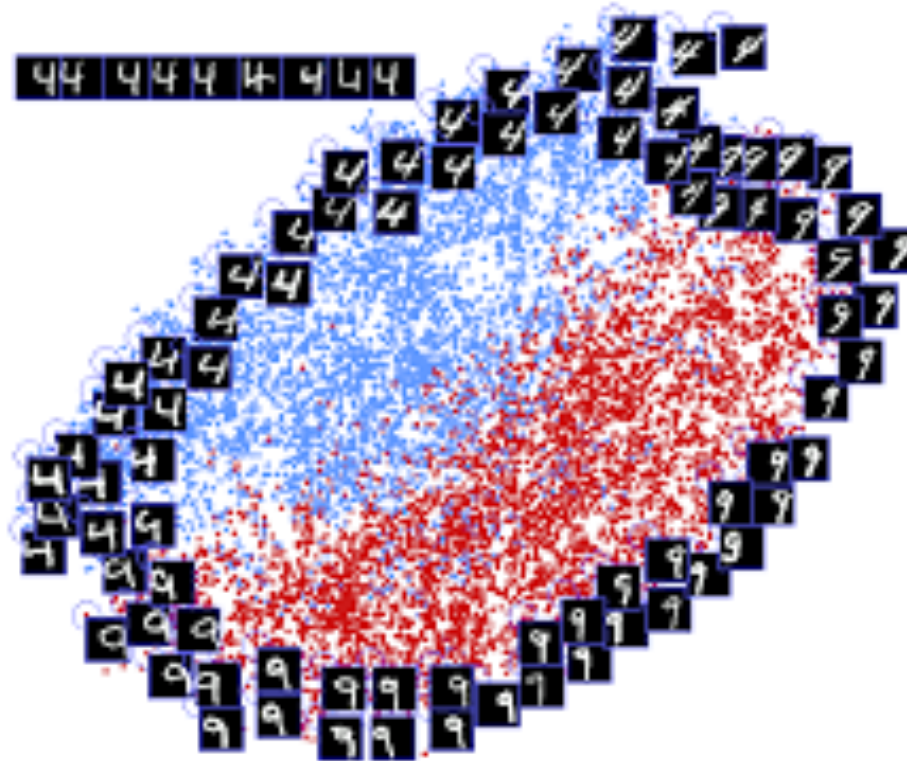
Bad news:

- The ratio of the number of points to the dimension must be less than 2.
- The dimension must be huge for large databases.
- As the dimension increases, the boundaries become increasingly irregular and sensitive to noise.

Good news:

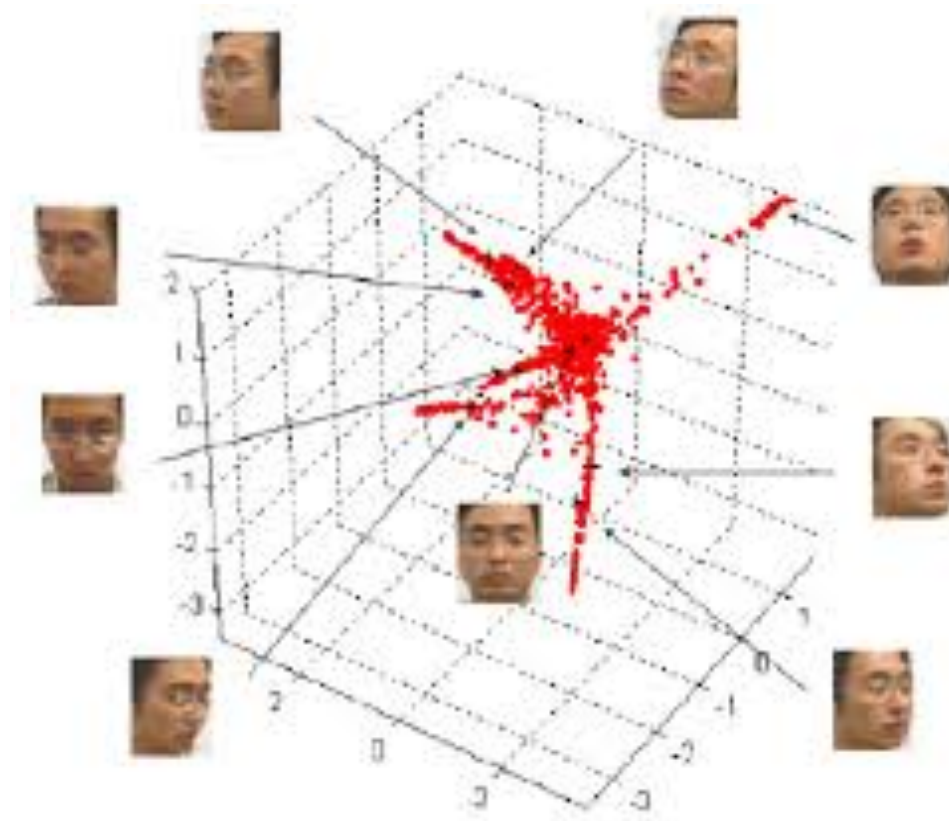
- The world is structured and the points we want to classify are NOT randomly distributed.
- We can compute feature vectors that are “close” for objects that belong to the same class.

Dimensionality Reduction



- The MNIST images are 28x28 arrays.
- They are **not** uniformly distributed in \mathbb{R}^{784} .
- In fact they exist on a low dimensional manifold.

Face Images



- The same can be said about face images.
- And of many other things.
- > Non linear classification is a practical proposition.

Increasing the Dimension Further

Can we increase the dimension massively:

- in a principled way,
- while keeping the computational burden down?

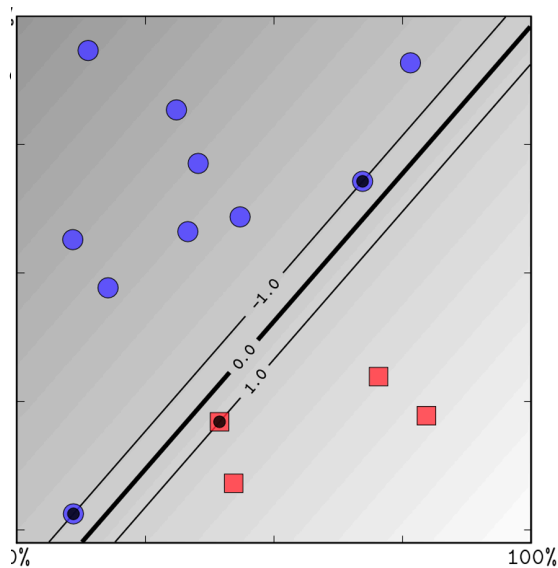
—> Non-linear support vector machines that use the so-called kernel trick.

Reminder: Polynomial SVM

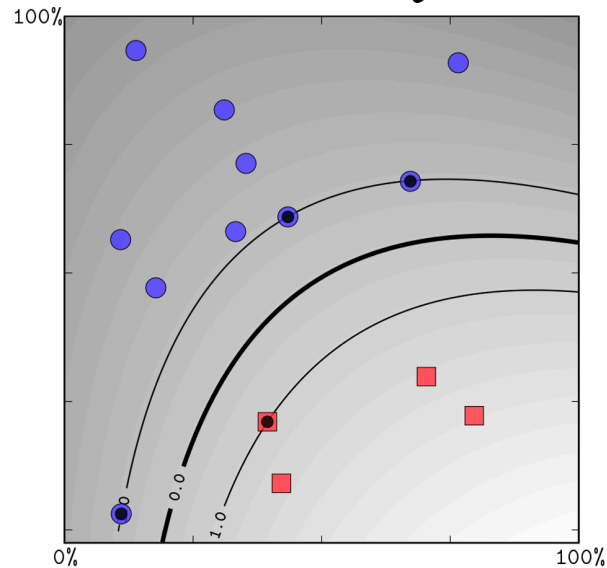
$$\mathbf{w}^* = \min_{(\mathbf{w}, \{\xi_n\})} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^N \xi_n,$$

subject to $\forall n, \quad t_n \cdot (\tilde{\mathbf{w}} \cdot \phi(\mathbf{x})_n) \geq 1 - \xi_n$ and $\xi_n \geq 0$.

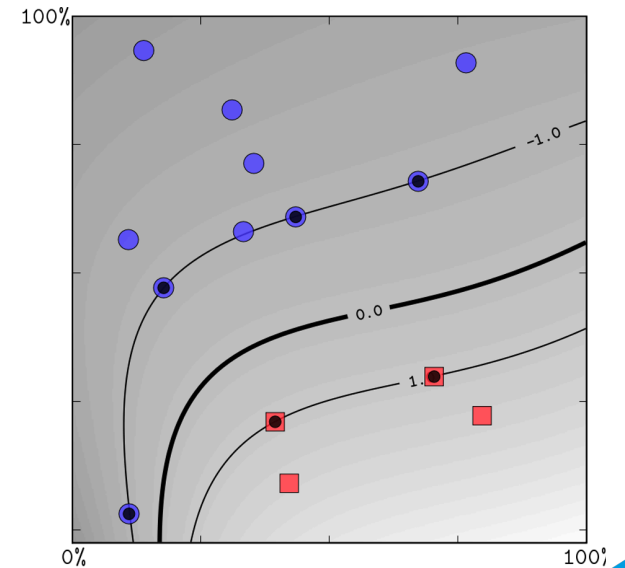
- C is constant that controls how costly constraint violations are.



M = 1



M = 2



M = 5

Polynomial SVM w/o Slack Variables

- For simplicity

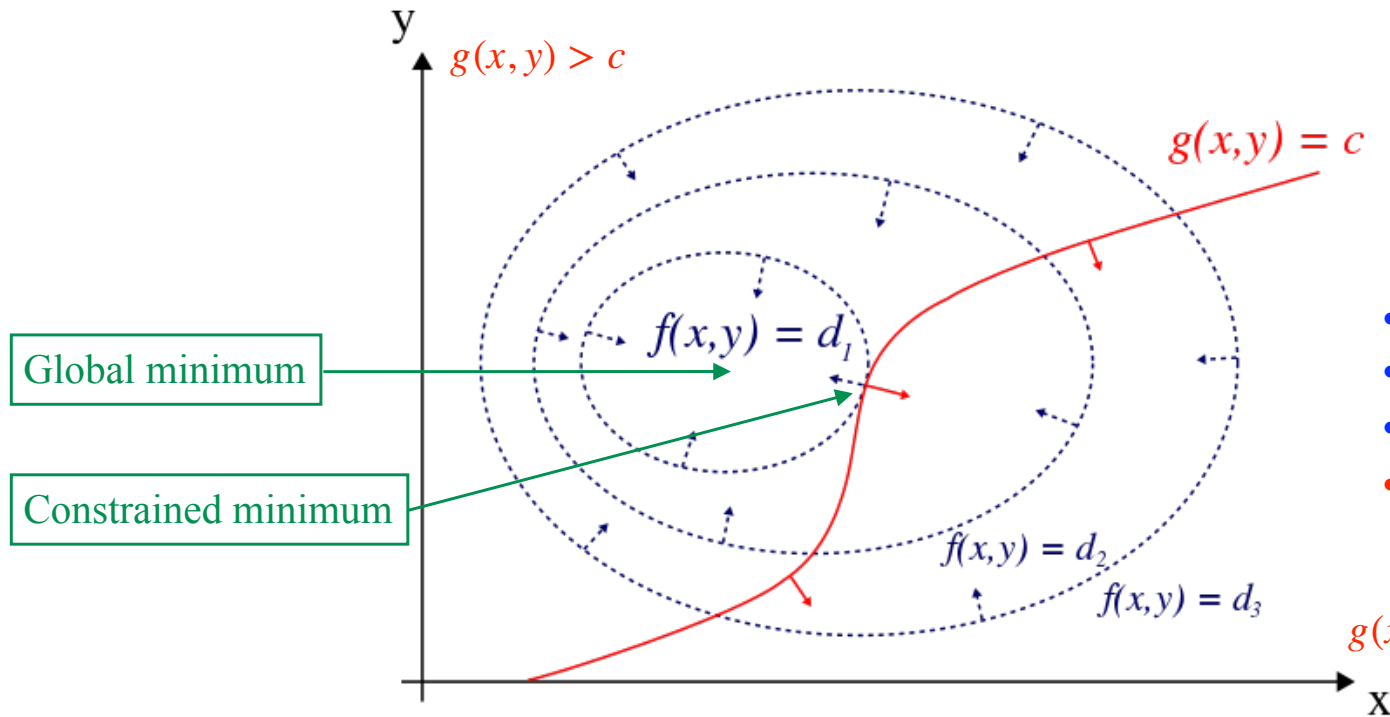
$$\mathbf{w}^* = \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2,$$

$$\text{subject to } \forall n, \quad t_n \cdot (\tilde{\mathbf{w}} \cdot \phi(\mathbf{x})_n) \geq 1.$$

- We will re-introduce the slack variables later.

—> Constrained optimization.

Reminder: Constrained Optimization



- Blue dotted lines are “level” lines.
- In this example, $d_1 < d_2 < d_3$.
- The blue arrows represent ∇f .
- The red arrows represent ∇g .

- Minimize $f(x, y)$ subject to $g(x, y) \leq c$.

- At the constrained minimum

$$\exists \lambda \in R, \nabla f = \lambda \nabla g$$

- λ is known as a Lagrange multiplier.

Reminder: Lagrangian Formulation

Lagrangian:

$$L(\mathbf{w}, \Lambda) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N \lambda_n (t_n \tilde{\mathbf{w}} \cdot \phi(\mathbf{x}_n) - 1)$$


$$\Lambda = [\lambda_1, \dots, \lambda_n]$$

Lagrange multipliers.
One per constraint.

Theorem:

A solution of the constrained minimization problem must be such that L is minimized with respect to the components of vector \mathbf{w} and maximized with respect to the Lagrange multipliers, which must remain greater or equal to zero.

Optional: Minimizing the Lagrangian

$$L(\mathbf{w}, \Lambda) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^N \lambda_n (t_n \tilde{\mathbf{w}} \cdot \phi(\mathbf{x}_n) - 1)$$


$\mathbf{w} = \sum_n \lambda_n t_n \phi(\mathbf{x}_n)$

Setting the derivatives of $L(\mathbf{w}, \Lambda)$ to zero with respects to the elements of \mathbf{w} and Λ yields

$$\mathbf{w} = \sum_{n=1}^N \lambda_n t_n \phi(\mathbf{x}_n) ,$$
$$0 = \sum_{n=1}^N \lambda_n t_n .$$

Optional: Dual Problem

Therefore, we minimize

$$\tilde{L}(\Lambda) = L(\mathbf{w}, \Lambda) = \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to

$$\lambda_n \geq 0 \quad \forall n ,$$

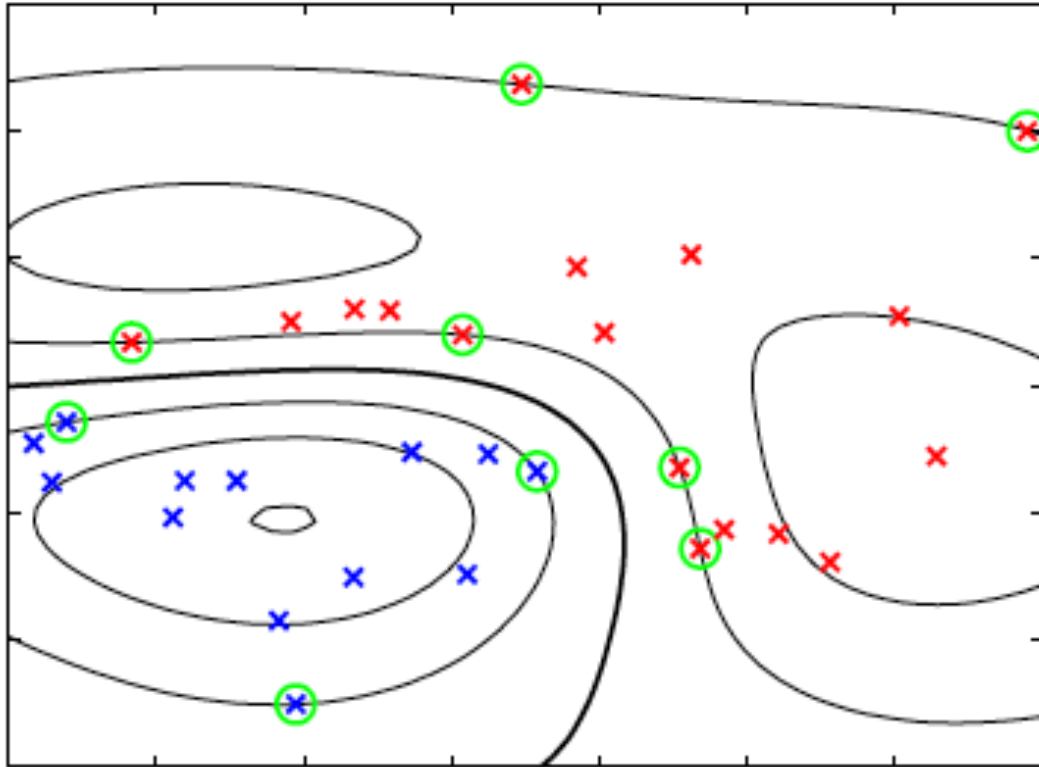
$$\sum_{n=1}^N \lambda_n t_n = 0$$

and with

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') .$$

- > Quadratic programming problem with N variables.
- > Complexity in $O(N^3)$ instead of $O(D^3)$.

Support Vectors



$$\mathbf{w} = \sum_{n=1}^N \lambda_n t_n \phi(\mathbf{x}_n) .$$

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b ,$$

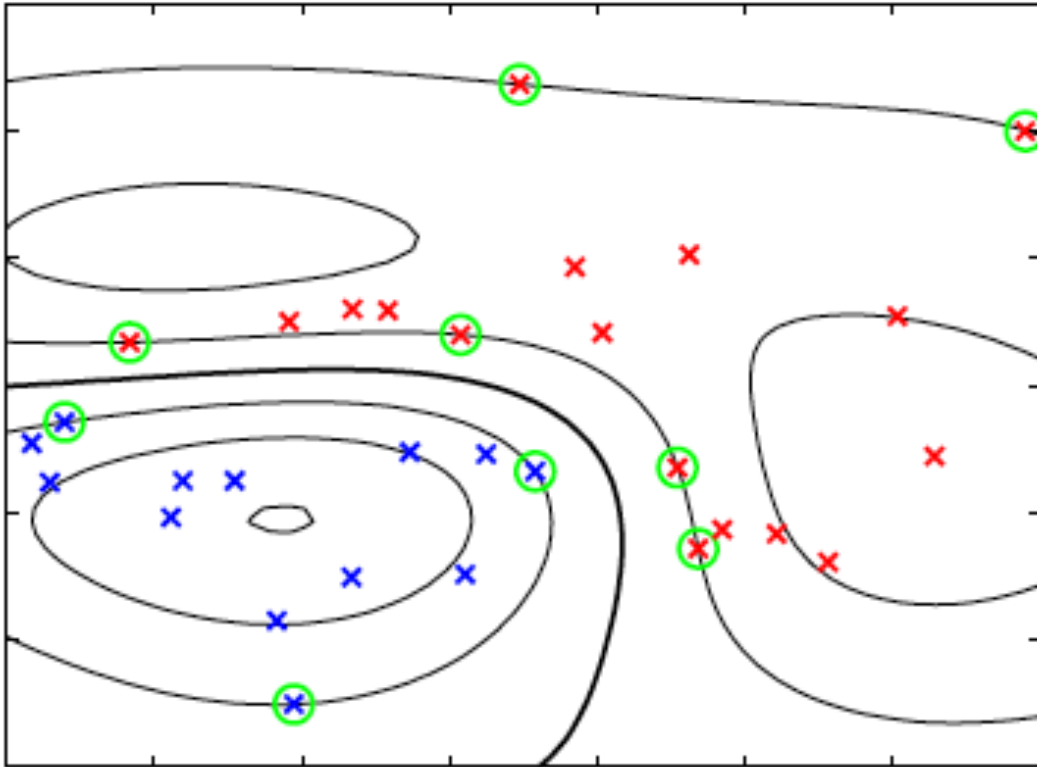
$$= \sum_{n=1}^N \lambda_n t_n k(\mathbf{x}, \mathbf{x}_n) + b ,$$

with $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$.

- Only for a subset of the data points is λ_n is non zero.
- They are denoted by green circles.
- The corresponding \mathbf{x}_n are the support vectors and satisfy $t_n y(\mathbf{x}_n) = 1$.
- They are the only ones that need to be considered as test time.

—> That is what makes SVMs practical!

At Inference Time



$$\begin{aligned}y(\mathbf{x}) &= \sum_{n=1}^N \lambda_n t_n k(\mathbf{x}, \mathbf{x}_n) + b \\ &= \sum_{n \in \mathcal{S}} \lambda_n t_n k(\mathbf{x}, \mathbf{x}_n) + b\end{aligned}$$

- Only for a subset of the data points is a_n is non zero.
- The feature vector $\phi(\mathbf{x})$ does **not** appear explicitly anymore.
- The kernel function $k(.,.)$ can be understood as a similarity measure.

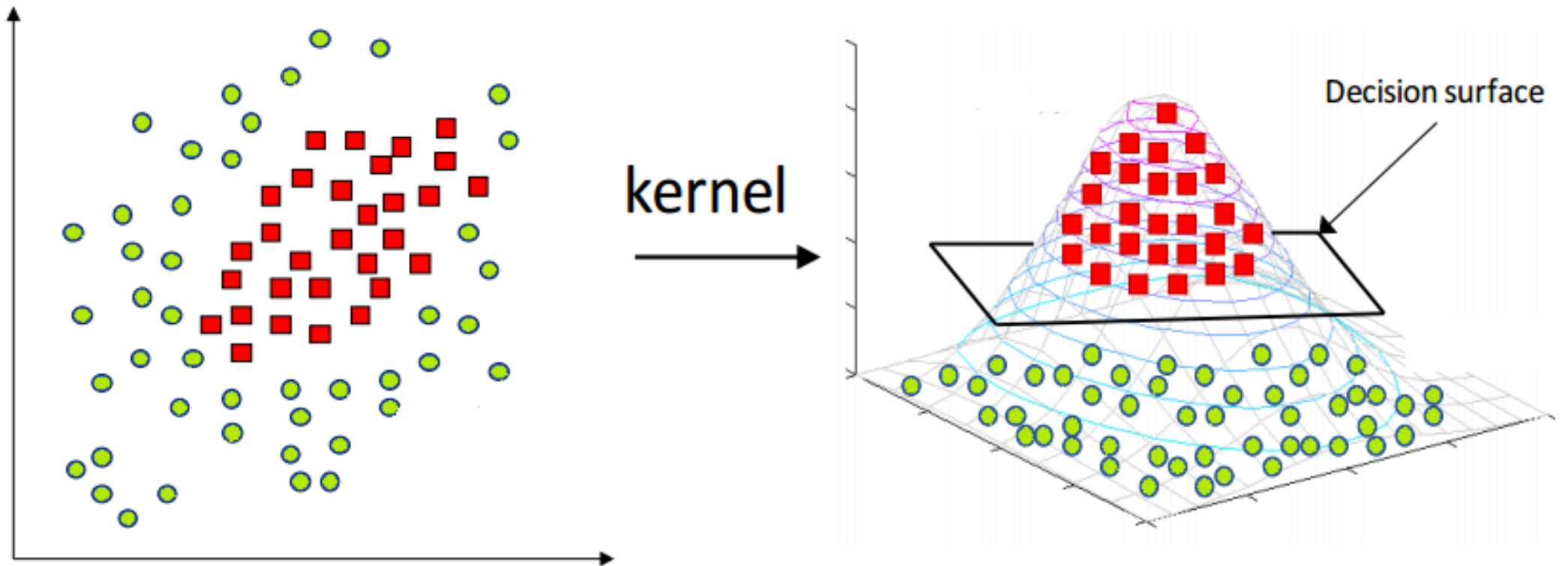
The Kernel Trick

$$y(\mathbf{x}) = \sum_{n \in \mathcal{S}} \lambda_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

- ϕ is implicit: In practice, we never compute it.
- We only need to compute k .
- This is known as the **kernel trick** and is used in many different algorithms besides SVMs.

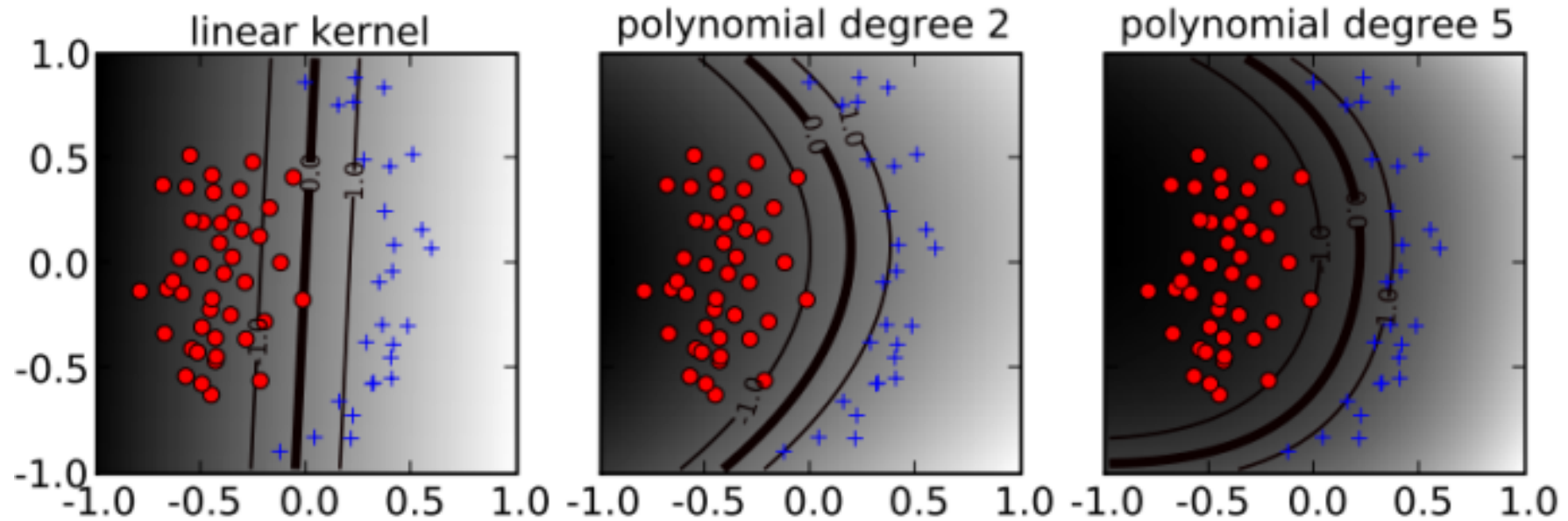
Role of the Kernel



Polynomial kernels: From small to high dimension.

Gaussian kernels: From small to infinite dimension.

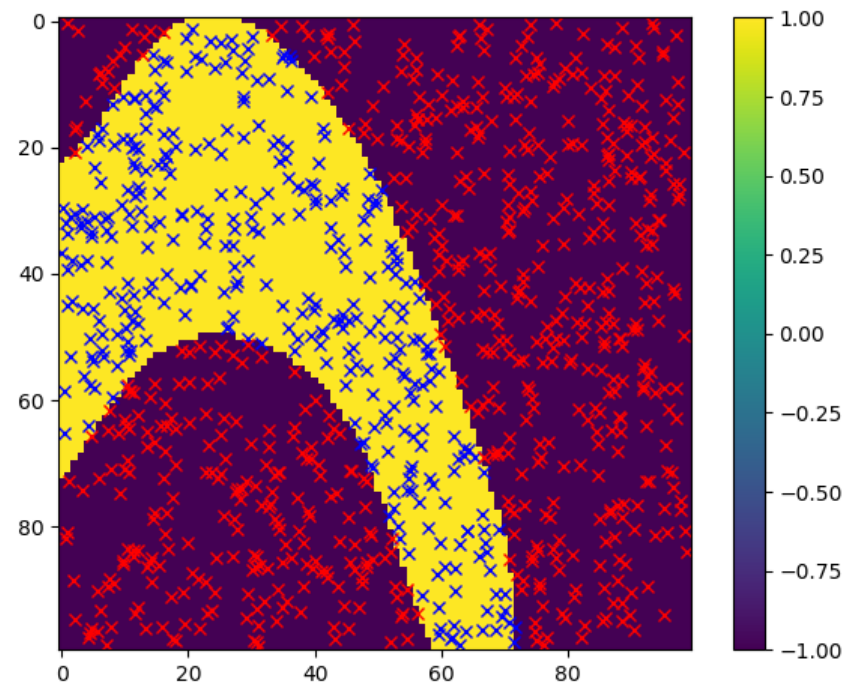
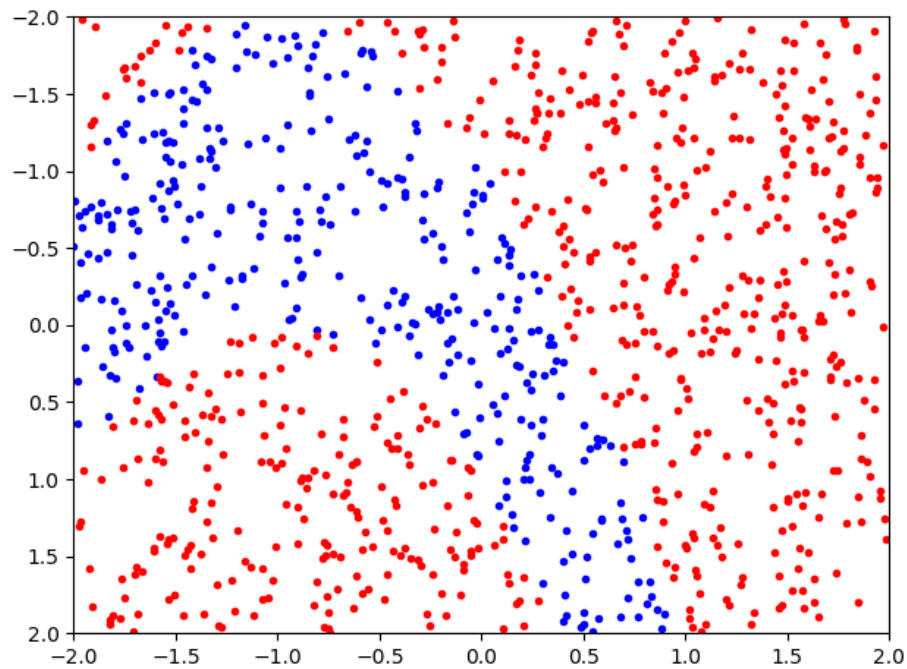
Influence of the Kernel



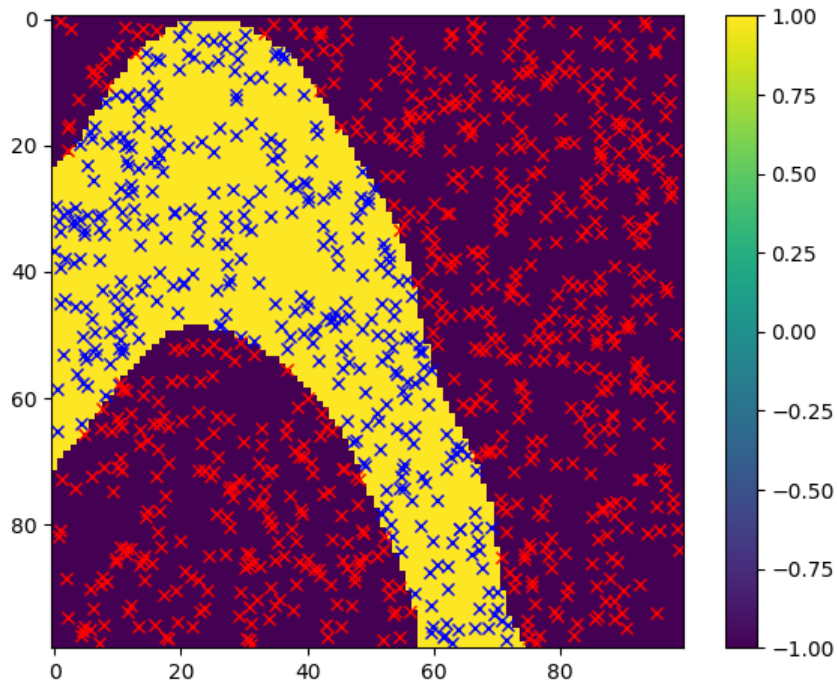
$$y(\mathbf{x}) = \sum_{n=1}^N \lambda_n t_n k(\mathbf{x}, \mathbf{x}_n) + b ,$$

$$k(\mathbf{x}, \mathbf{x}') = 1 + (\mathbf{x}^T \mathbf{x}')^d \quad (\text{Polynomial terms up to degree } d).$$

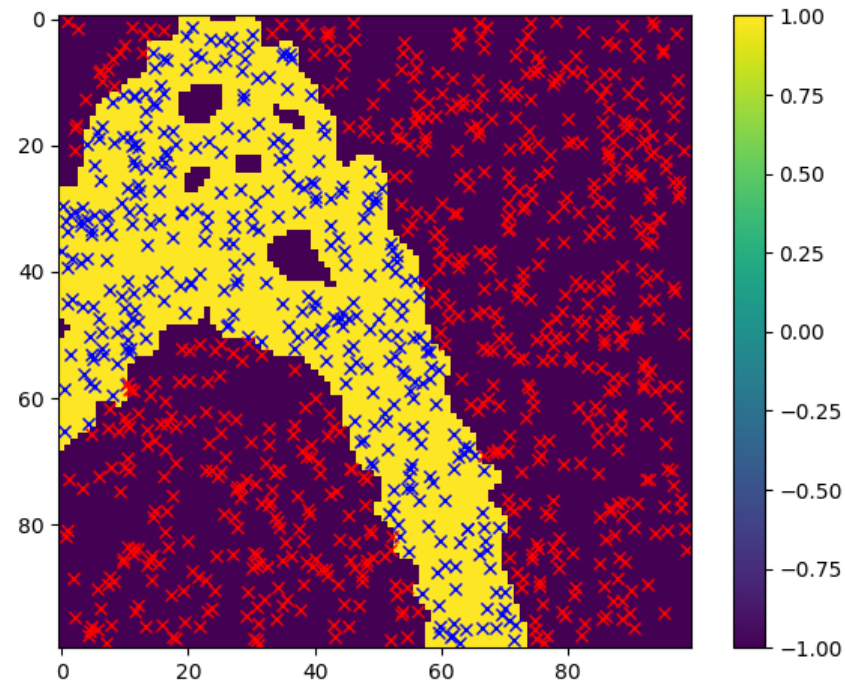
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma^2}\right) \quad (\text{Gaussian, feature space of infinite dimension}).$$



$|x, y, \text{Rbf}, \sigma = 1.0, \tilde{y}|$

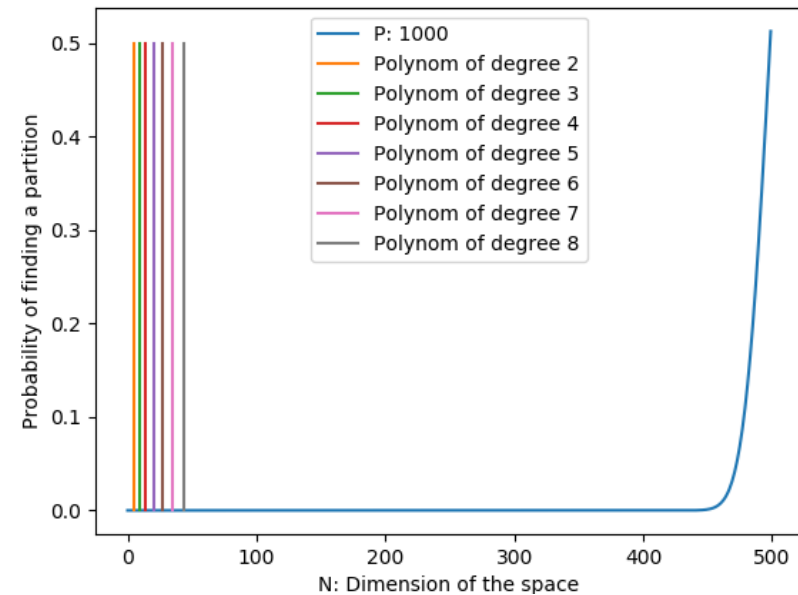
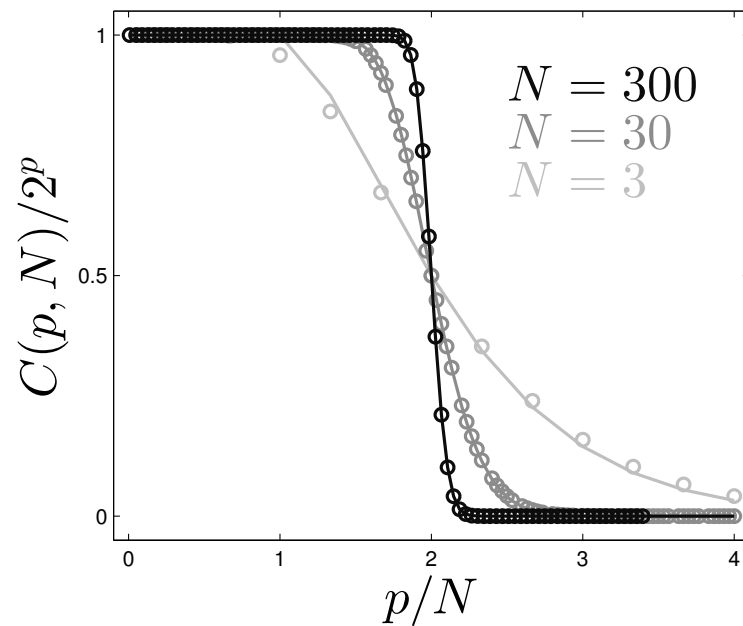


Rbf, $\sigma = 1.0$



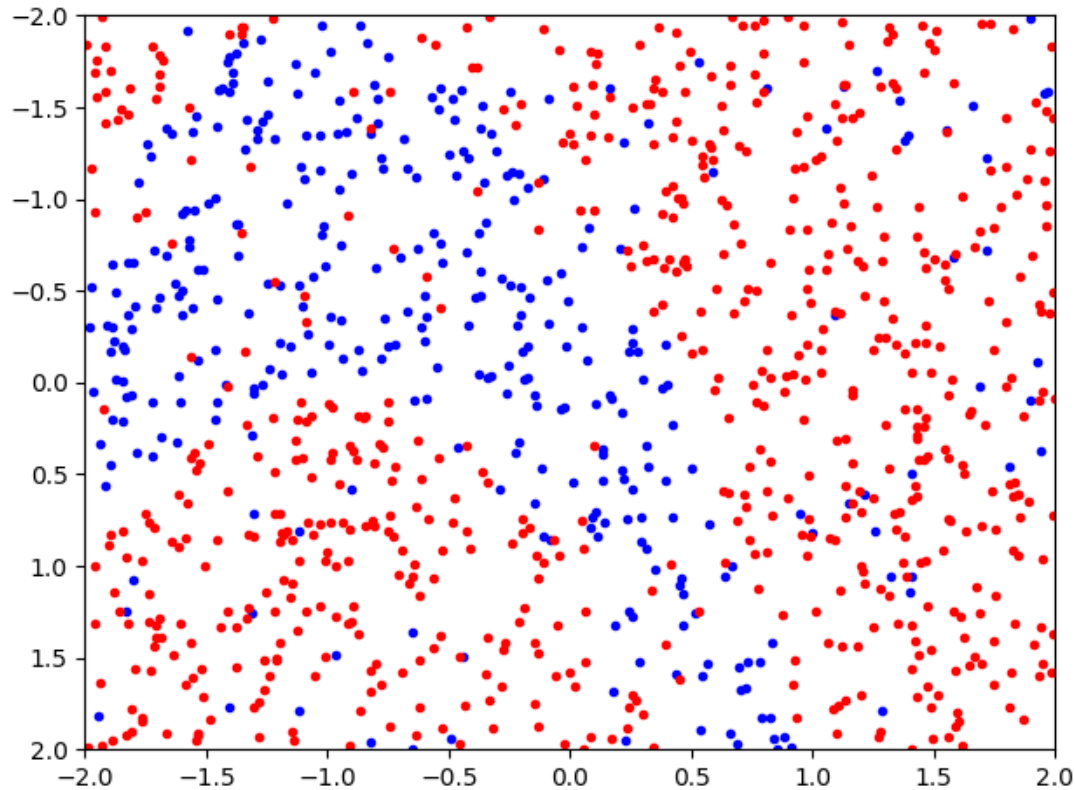
Rbf, $\sigma = 0.01$

Back to Cover's Theorem



- Good news: Working with a Gaussian kernel virtually makes the dimension as large as the number of samples.
- Bad news: It is still not ideal for very large values of the number of points N due to the $O(N^3)$ computational complexity.

Overlapping Class Distributions



Some blues among the reds and some reds among the blues.

- Some training examples must be allowed to be misclassified.
- Cannot satisfy all the hard constraints.
- For linear SVMs, we used slack variables to achieve this.
- For kernel SVMs, we can do so by bounding the Lagrange multipliers.

Optional: Dual Problem with Slack Variables

We now minimize

$$\tilde{L}(\Lambda) = L(\mathbf{w}, \Lambda) = \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to

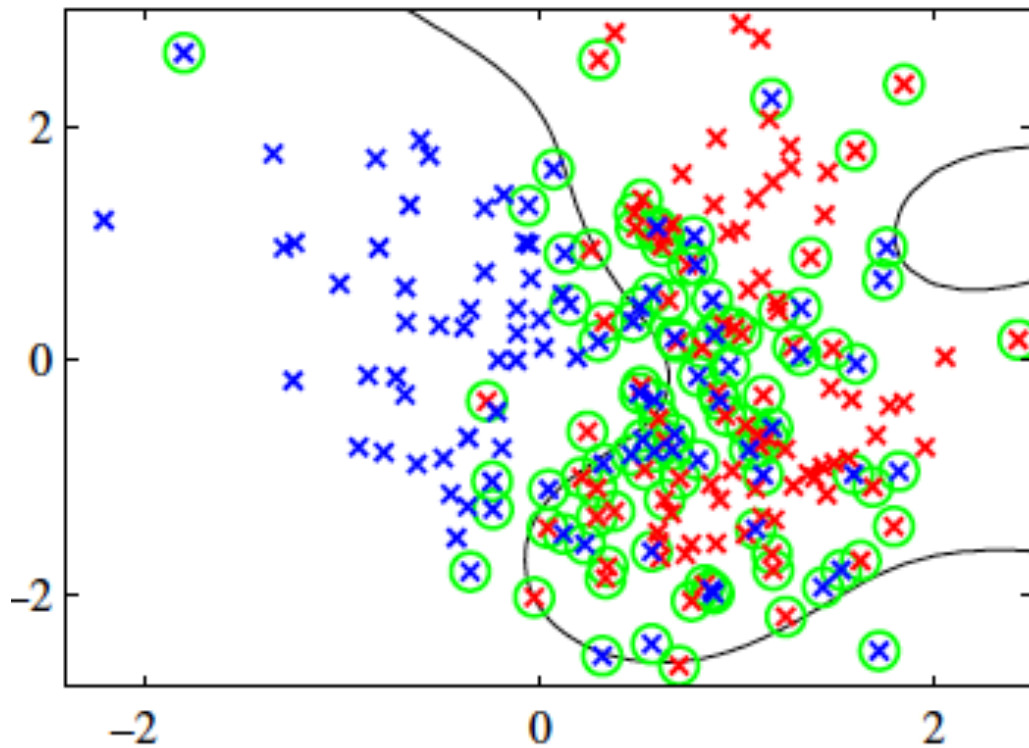
$$\forall n, \quad 0 \leq \lambda_n \leq C, \quad \lambda_n \text{ cannot become infinite and therefore some of the constraints can be violated.}$$

$$\sum_{n=1}^N \lambda_n t_n = 0$$

and with

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}').$$

Finally a Simple Usable Formula

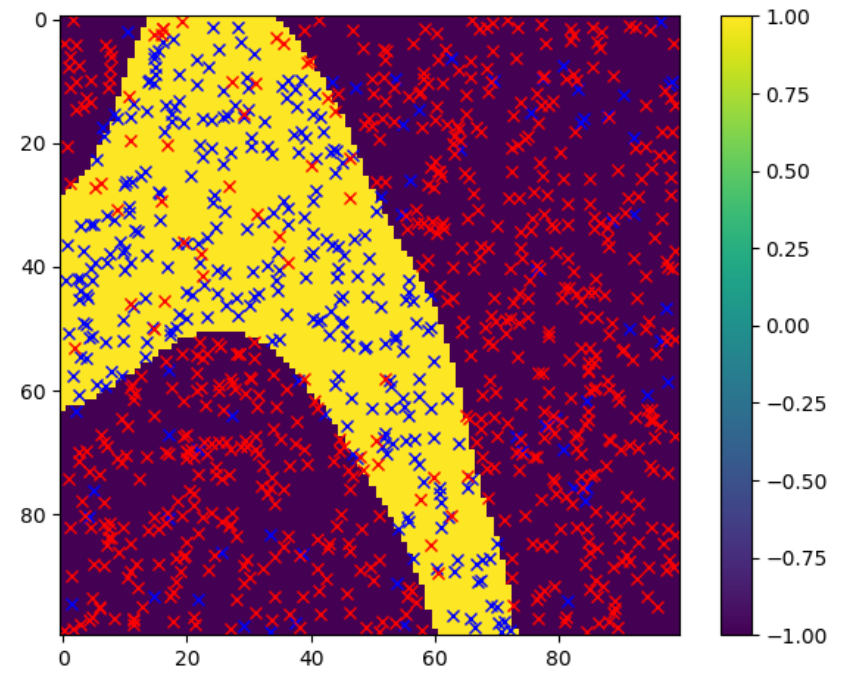
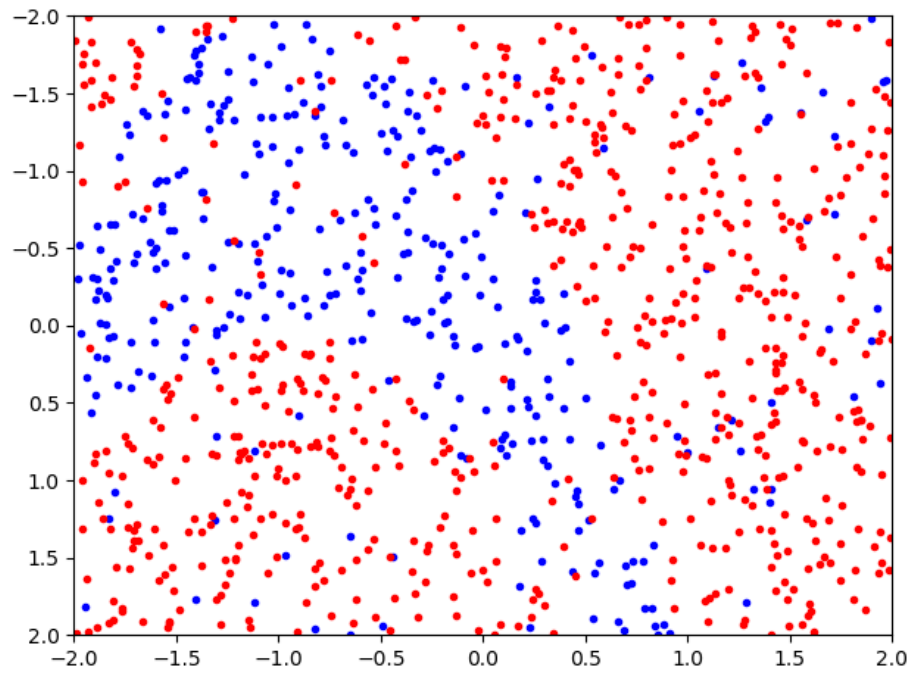


The green circles denote the support vectors.

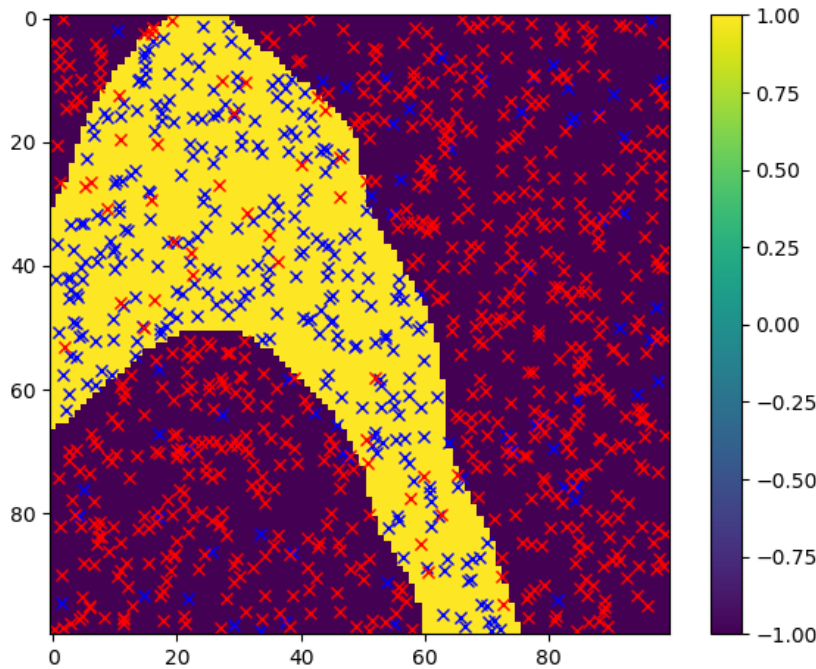
$$y(\mathbf{x}) = \sum_{n \in \mathcal{S}} \lambda_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

where \mathcal{S} is the set of support vectors.

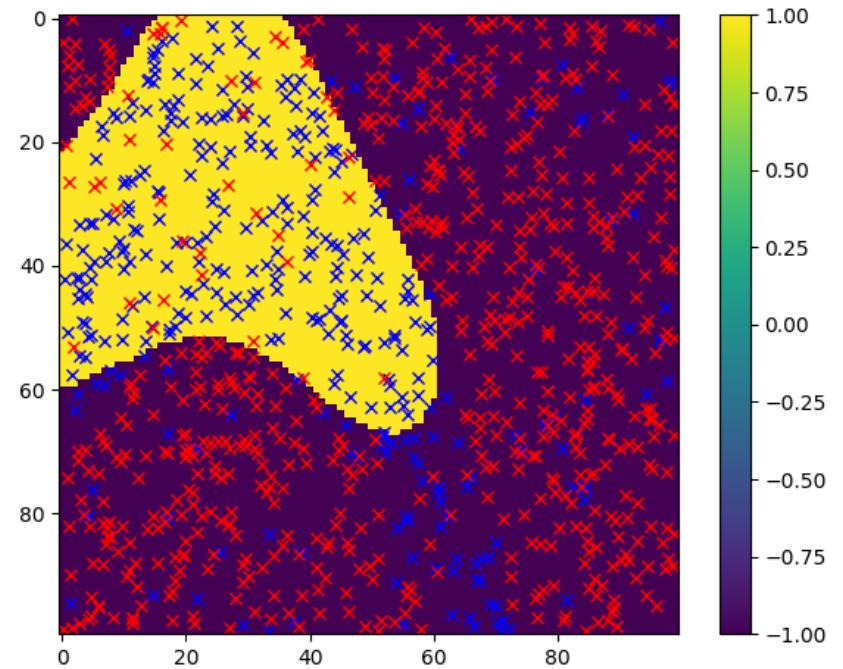
- $\lambda_n < C$: \mathbf{x}_n lies on the margin.
- $\lambda_n = C$: \mathbf{x}_n lies inside the margin.



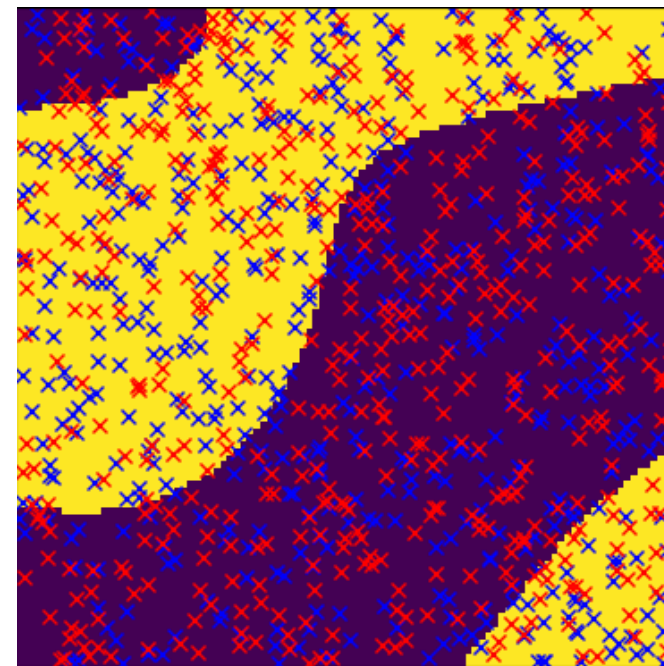
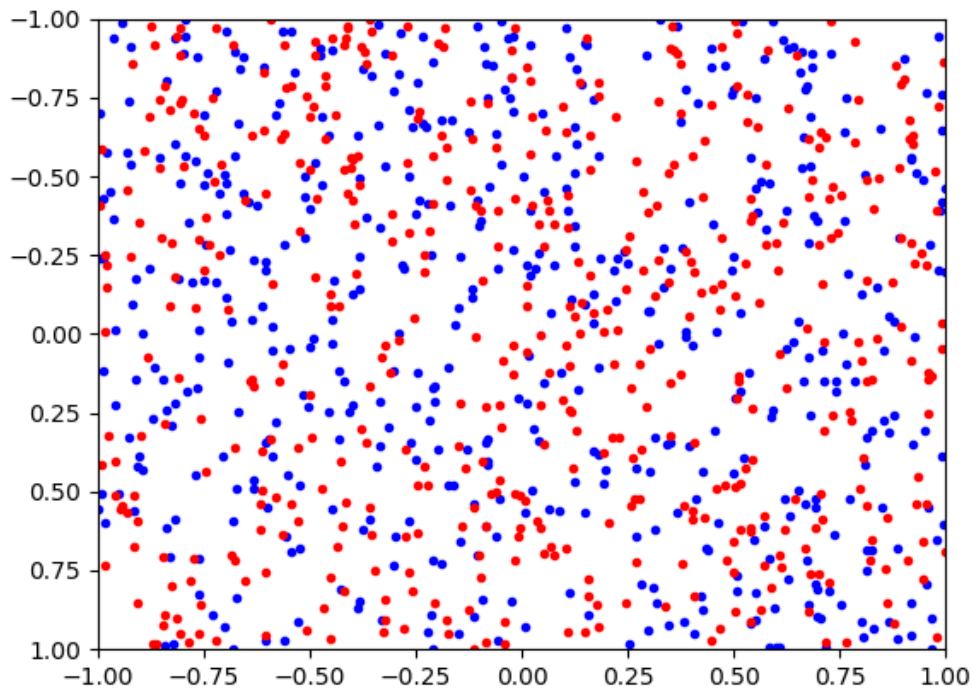
Rbf, $\sigma = 1.0$, $C=1.0$



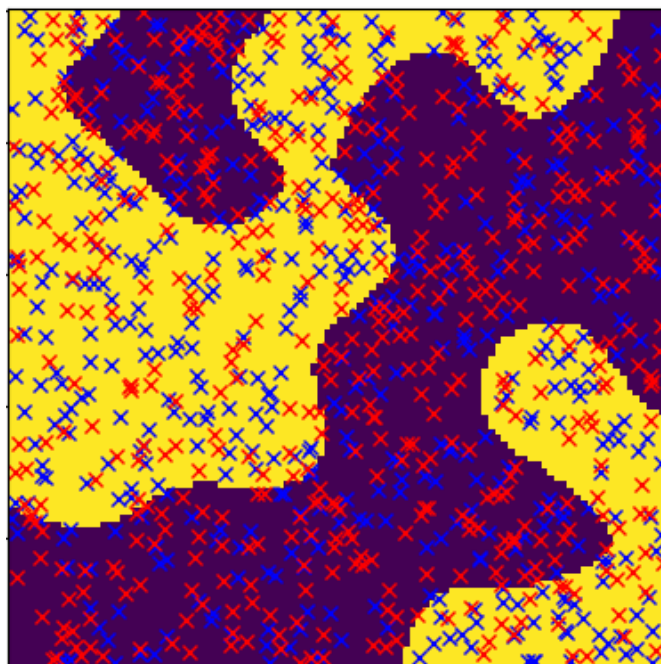
Rbf, $\sigma = 1.0$, $C= 100.0$



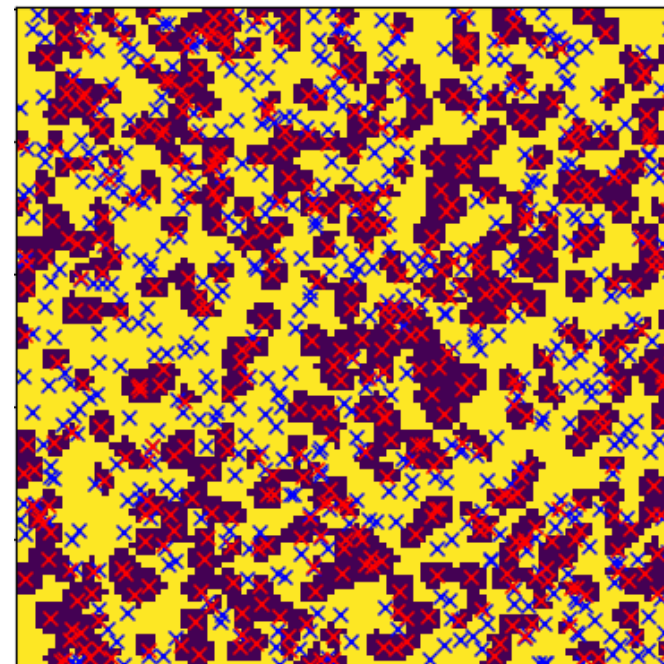
Rbf, $\sigma = 1.0$, $C= 0.1$



Rbf, $\sigma = 1.0$, $C = 1.0$

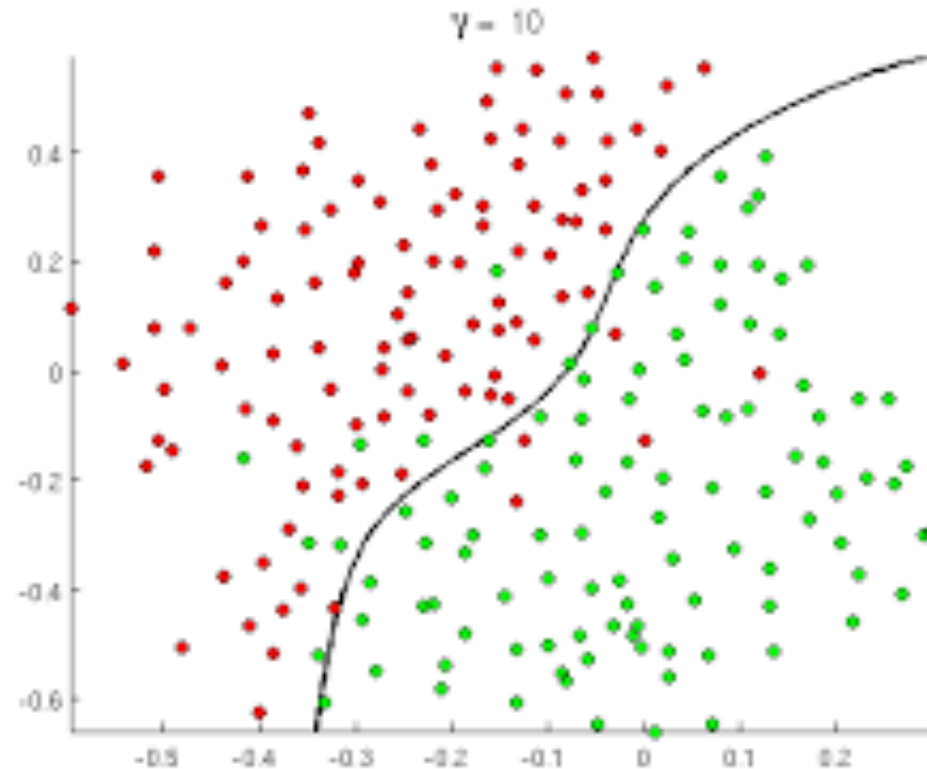


Rbf, $\sigma = 0.1$, $C = 1.0$



Rbf, $\sigma = 0.05$, $C = 1.0$

Non-Separable Distributions



The slack variables allow some training points to be misclassified.

- A large σ sigma tends to smooth the decision boundary.
- A large C tends to minimize the number of misclassified training points.

—> Validation data is required to choose them properly.

Recognizing Hand-Written Digits

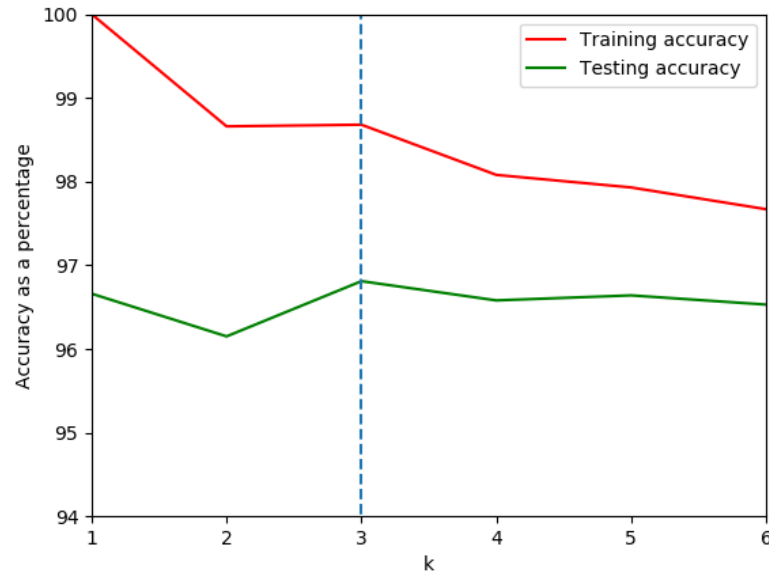
Test sample

Nearest neighbors

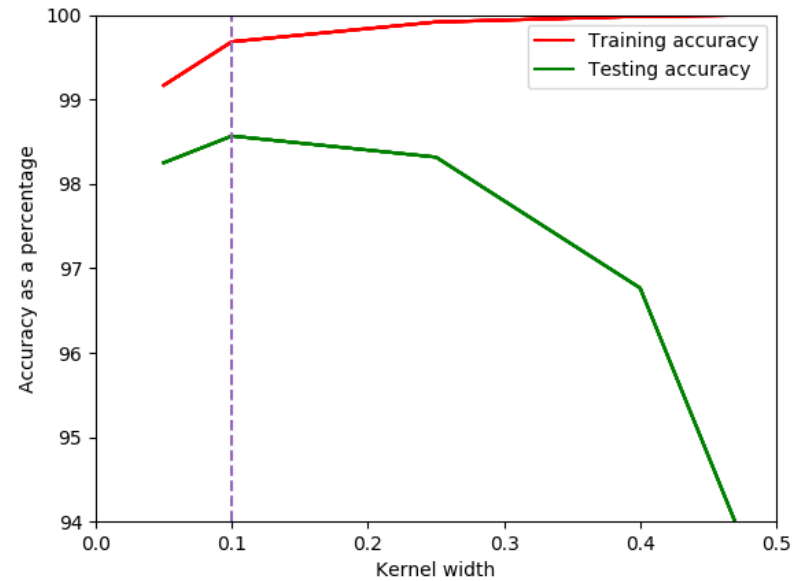
0
2
4
y
9

0000006
2228887
4444444
9494949
9777777

k-Nearest Neighbors vs SVM on MNIST



Knn: 96.8%



Rbf-SVM: 98.6%

- Better accuracy.
- But the kernel and its parameters must be well chosen.

SVMs in Short

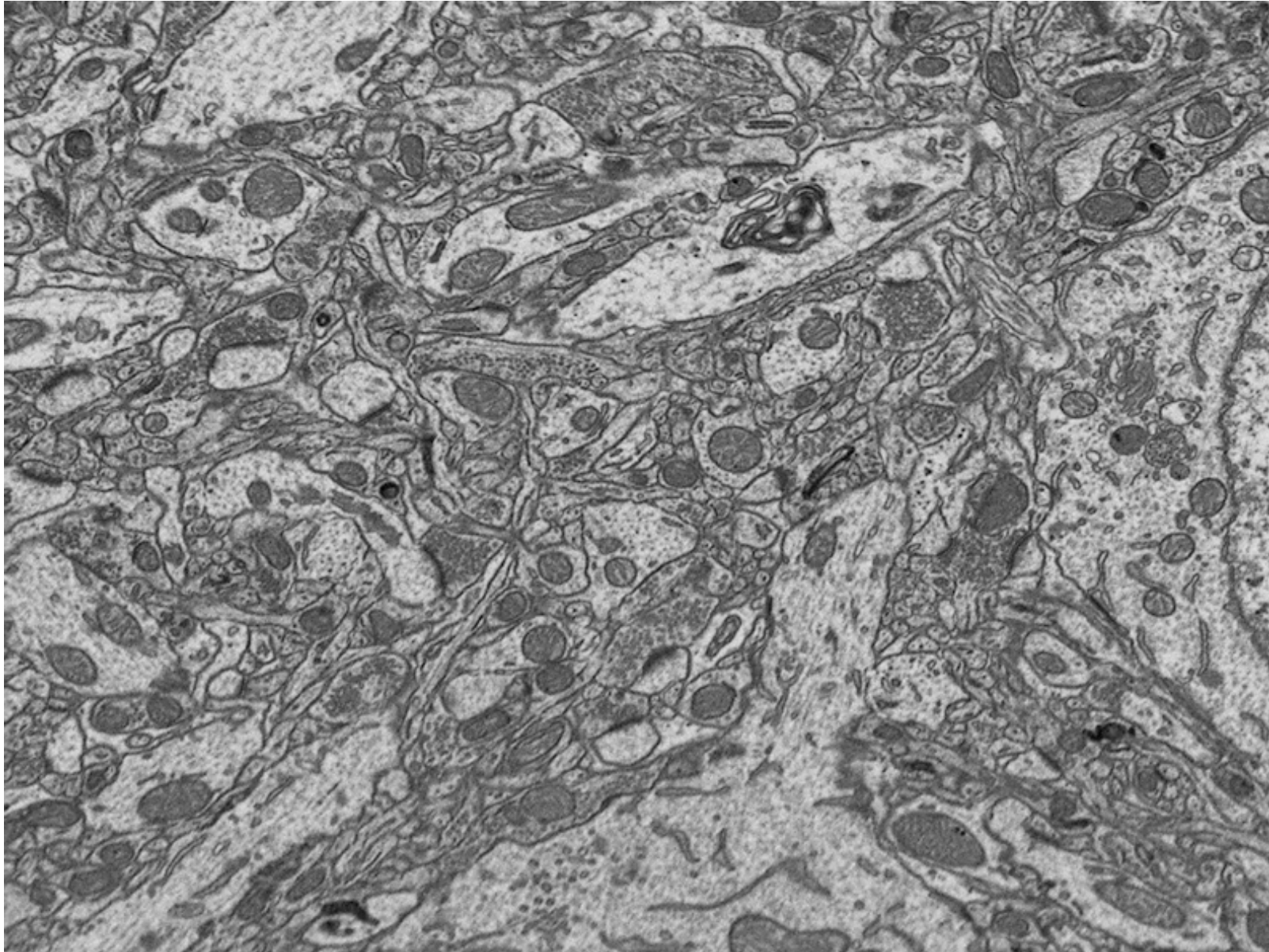
- The data can be separable in a high-dimensional feature space without being separable in the input space.
- Classifiers can be learned in the feature space without having to actually perform the mapping.
- However the $O(D^3)$ or $O(N^3)$ complexity at training time makes it hard to exploit large training sets.

Reminder: SLIC Superpixels



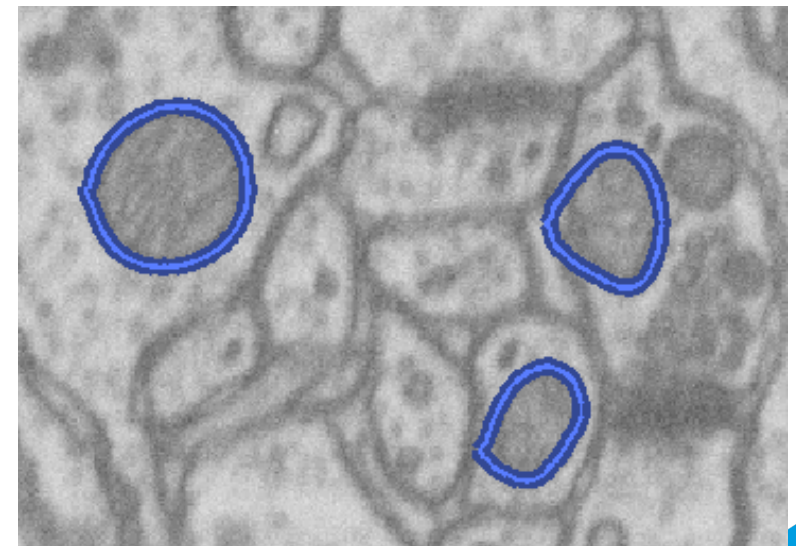
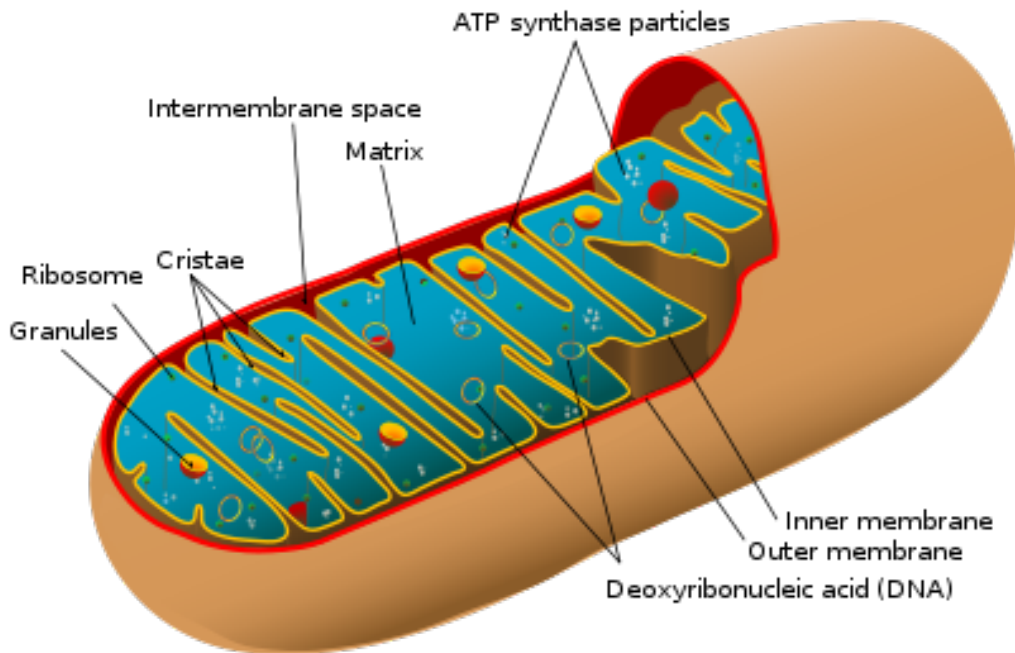
- Superpixel segmentations with centers on a 64x64, 256x256, and 1024x1024 grid.
- Can be used to describe the image in terms of a set of small regions.

Optional: Electron Microscopy

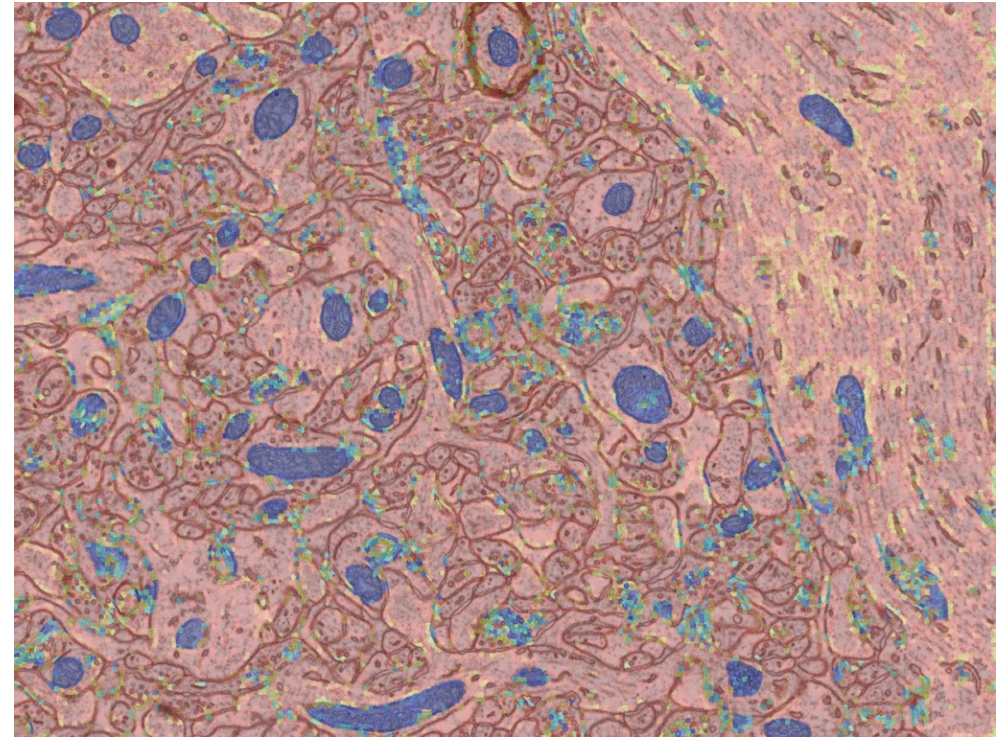
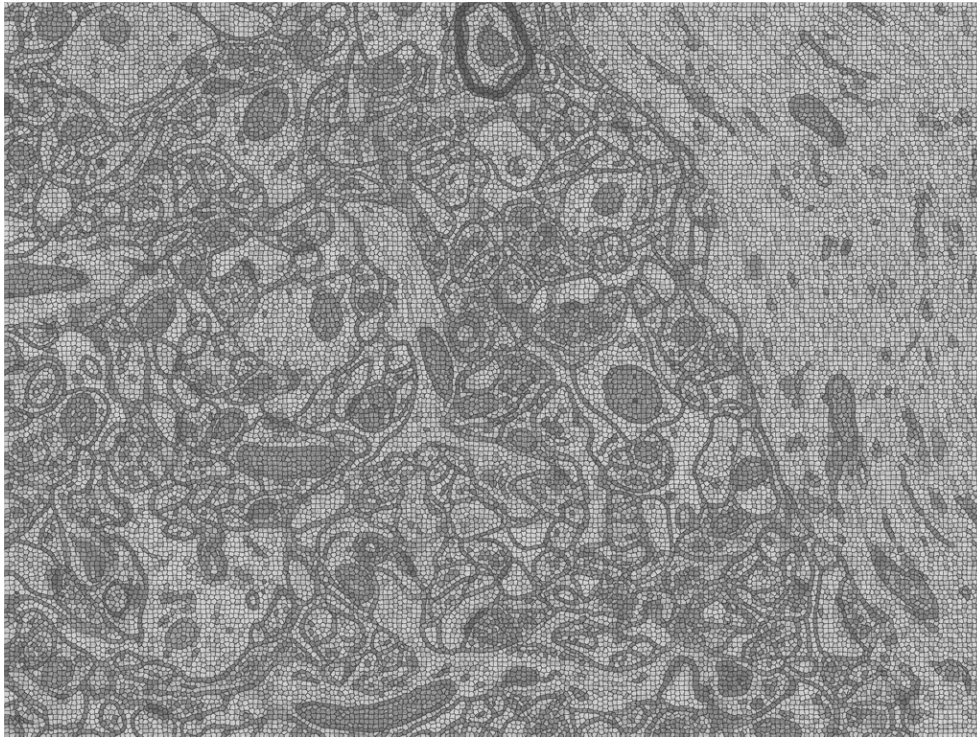


Let us SVMs to model structures of interest!

Optional: Mitochondria Segmentation

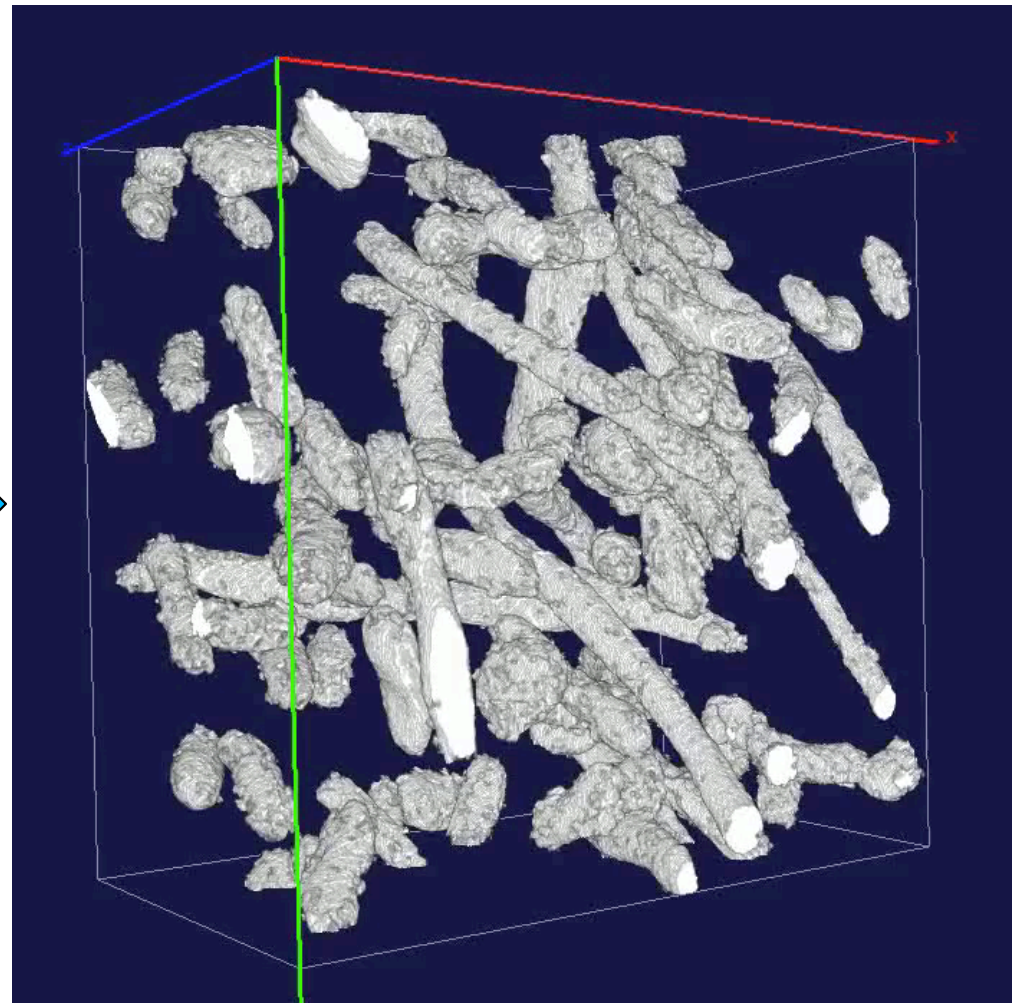
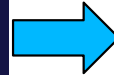
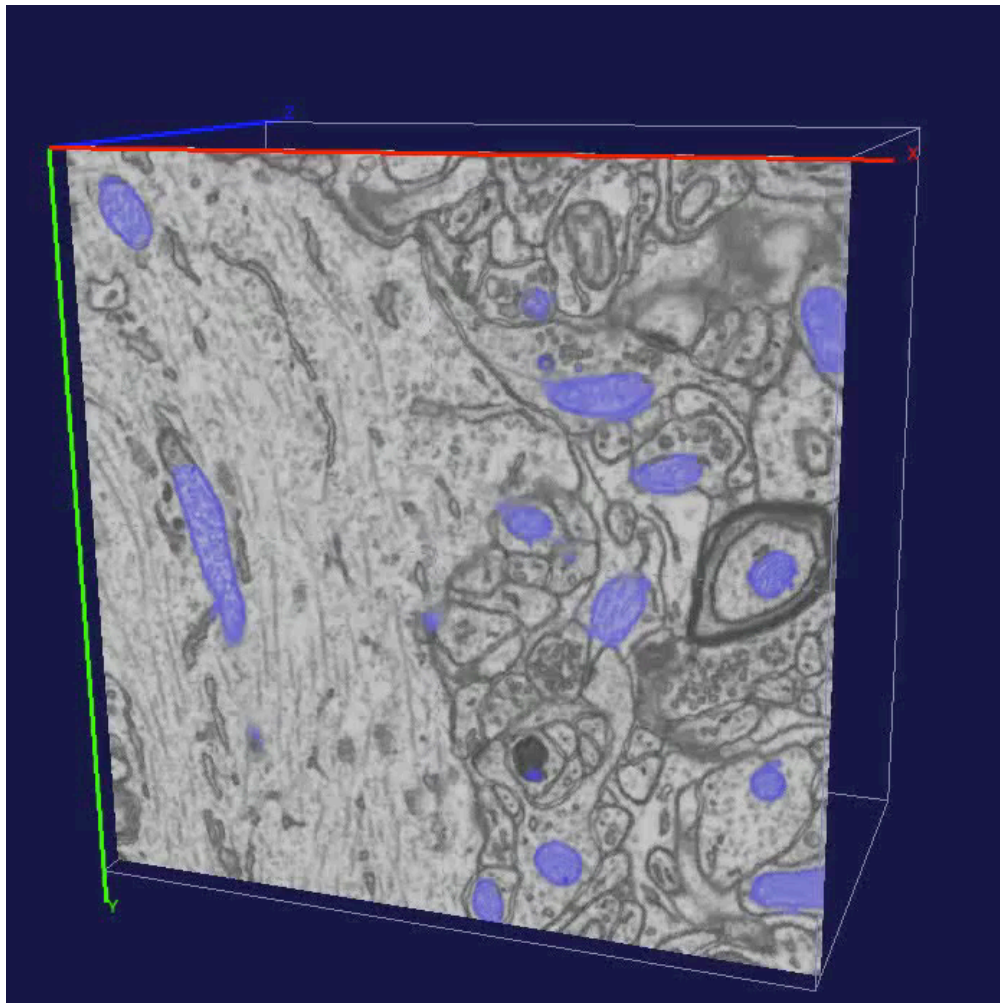


Optional: Assigning Probabilities

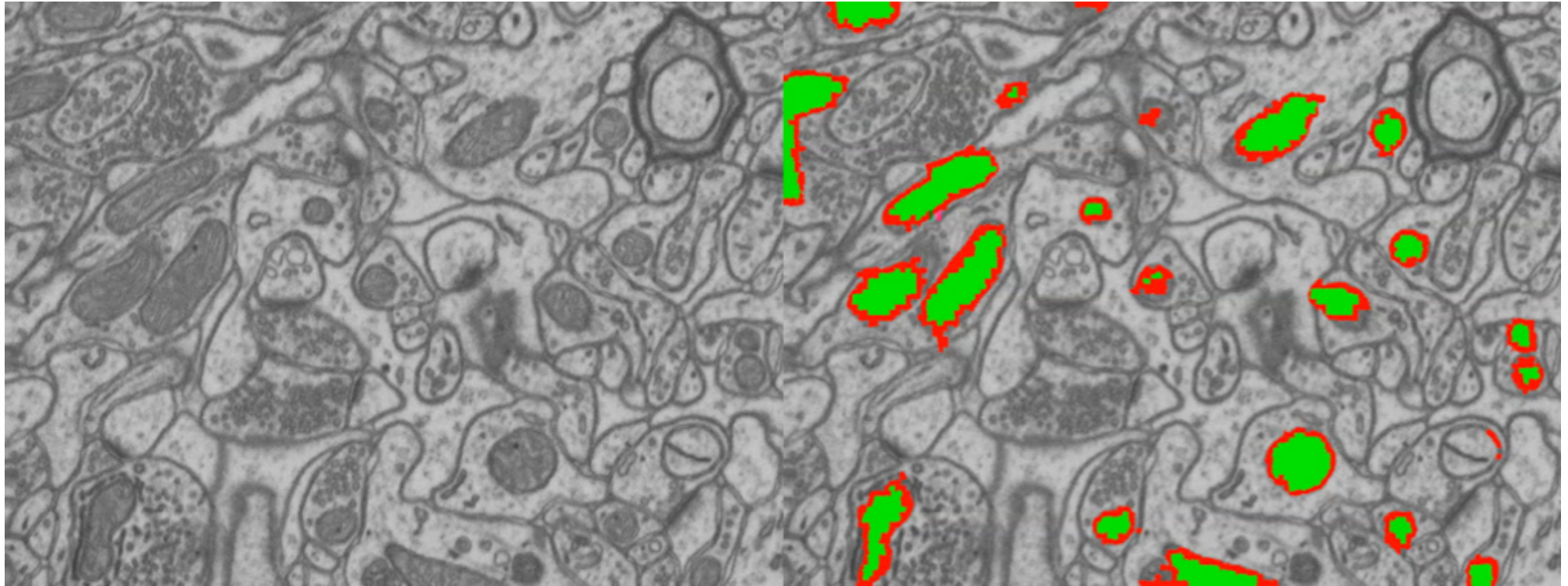


- Compute image features for each superpixel.
- Train an SVM classifier to assign a probability to be within a mitochondria.
- Can be used to produce segmentations using graph-based techniques.

Optional: 3D Mitochondria



Optional: Modeling Membranes



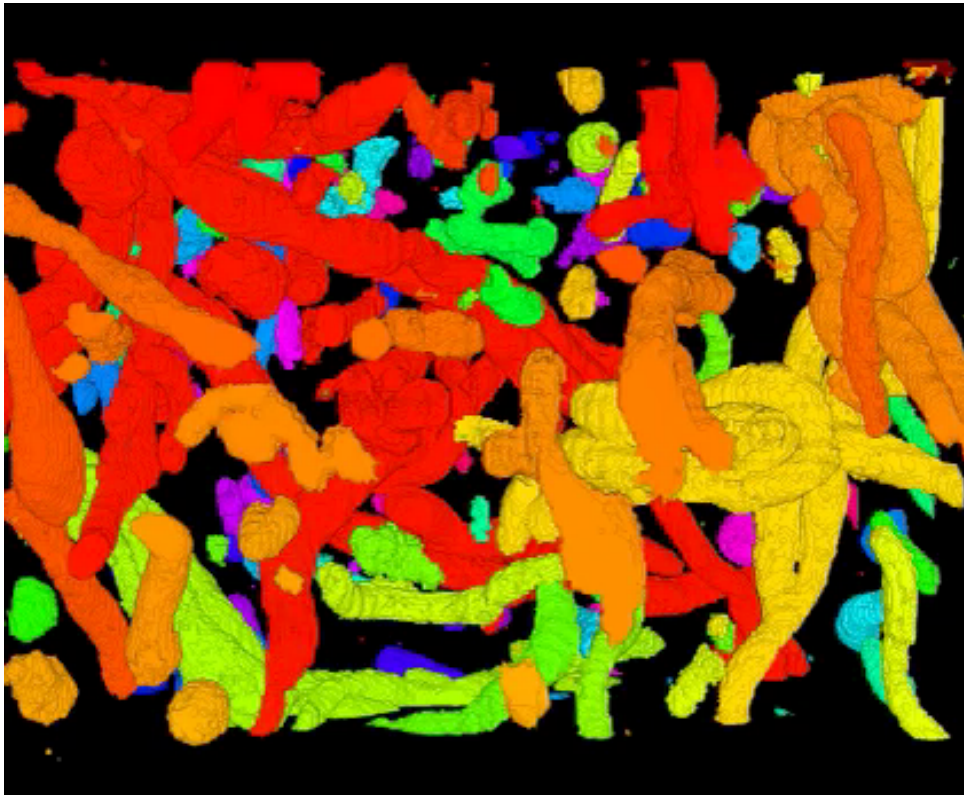
Here we use three classes instead of two:

- Inside
- Membrane
- Everything else

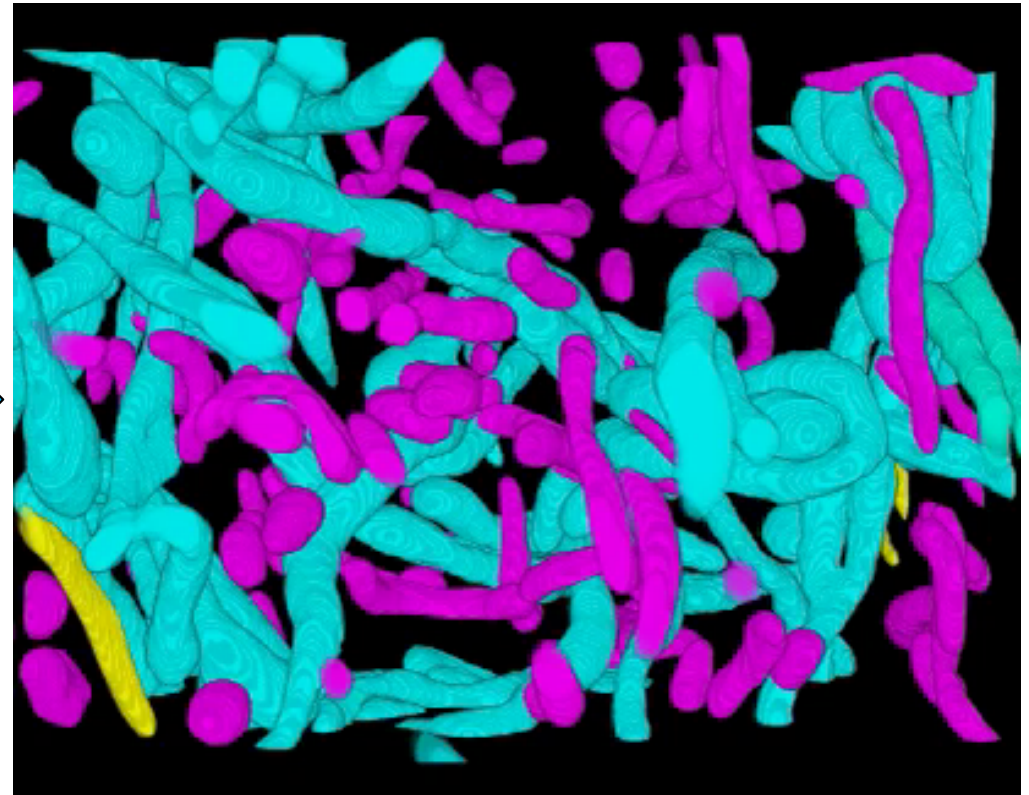
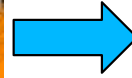
—> Because the inside is fully enclosed by the membranes, we can still find a global optimum.

Optional: Speeding up the Analysis

3.21 μm \times 3.21 μm \times 1.08 μm : 53 mitochondria



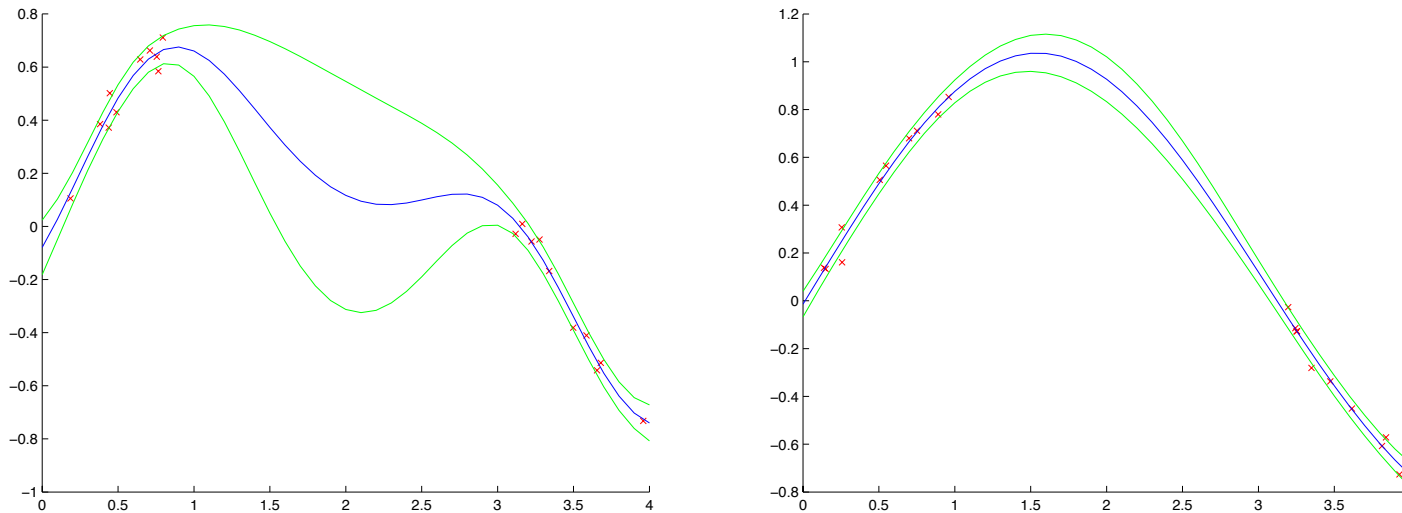
Automated result



Interactively cleaned-up result

- By hand: 6 hours.
 - Semi-automatically: 1.5 hours
- > Substantial time saving for the neuroscientists.

Optional: Non-Linear Regression



- Let $(\mathbf{x}_n, \mathbf{t}_n)_{1 \leq n \leq N}$ be N training pairs where the \mathbf{x}_n are d dimensional and the \mathbf{t}_n D dimensional.
- Let y be the function defined as $y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$, where \mathbf{w} is a $M \times D$ matrix, and ϕ denotes a function from R^d to R^M .

Our goal is to minimize

$$J(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \underbrace{\|\mathbf{w}^T \phi(\mathbf{x}_n) - \mathbf{t}_n\|^2}_{\text{Data term}} + \frac{\lambda}{2} \underbrace{\|\mathbf{w}\|^2}_{\text{Regularization term}},$$

with respect to \mathbf{w} .

Data term

Regularization term

Optional: Kernel Trick Again

Let $\mathbf{a}_n = \frac{1}{\lambda} \{ \mathbf{w}^T \phi(x_n) - \mathbf{t}_n \}$. It can be shown that the minimum is achieved for

$$\begin{bmatrix} \mathbf{a}_1^T \\ \dots \\ \mathbf{a}_N^T \end{bmatrix} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \begin{bmatrix} \mathbf{t}_1^T \\ \dots \\ \mathbf{t}_N^T \end{bmatrix},$$

where \mathbf{K} is the $N \times N$ symmetric Gram matrix with elements

$$\mathbf{K}_{n,m} = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m).$$

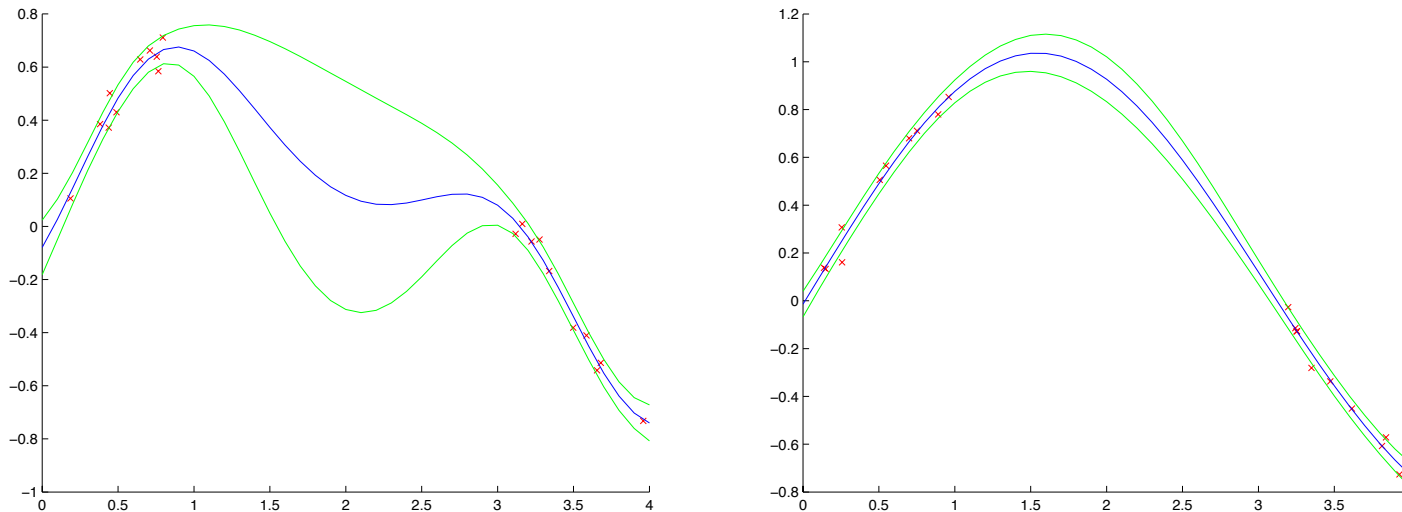
Therefore

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \mathbf{k}(\mathbf{x}) (\mathbf{K} + \lambda \mathbf{I})^{-1} \begin{bmatrix} \mathbf{t}_1^T \\ \dots \\ \mathbf{t}_N^T \end{bmatrix},$$

where $\mathbf{k}(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1), \dots, k(\mathbf{x}, \mathbf{x}_N)]^T$.

ϕ does not appear explicitly anymore!

Optional: Gaussian Processes



$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})(\mathbf{K} + \lambda\mathbf{I})^{-1} \begin{bmatrix} \mathbf{t}_1^T \\ \dots \\ \mathbf{t}_N^T \end{bmatrix},$$

$$\mathbf{K}_{n,m} = k(\mathbf{x}_n, \mathbf{x}_m),$$

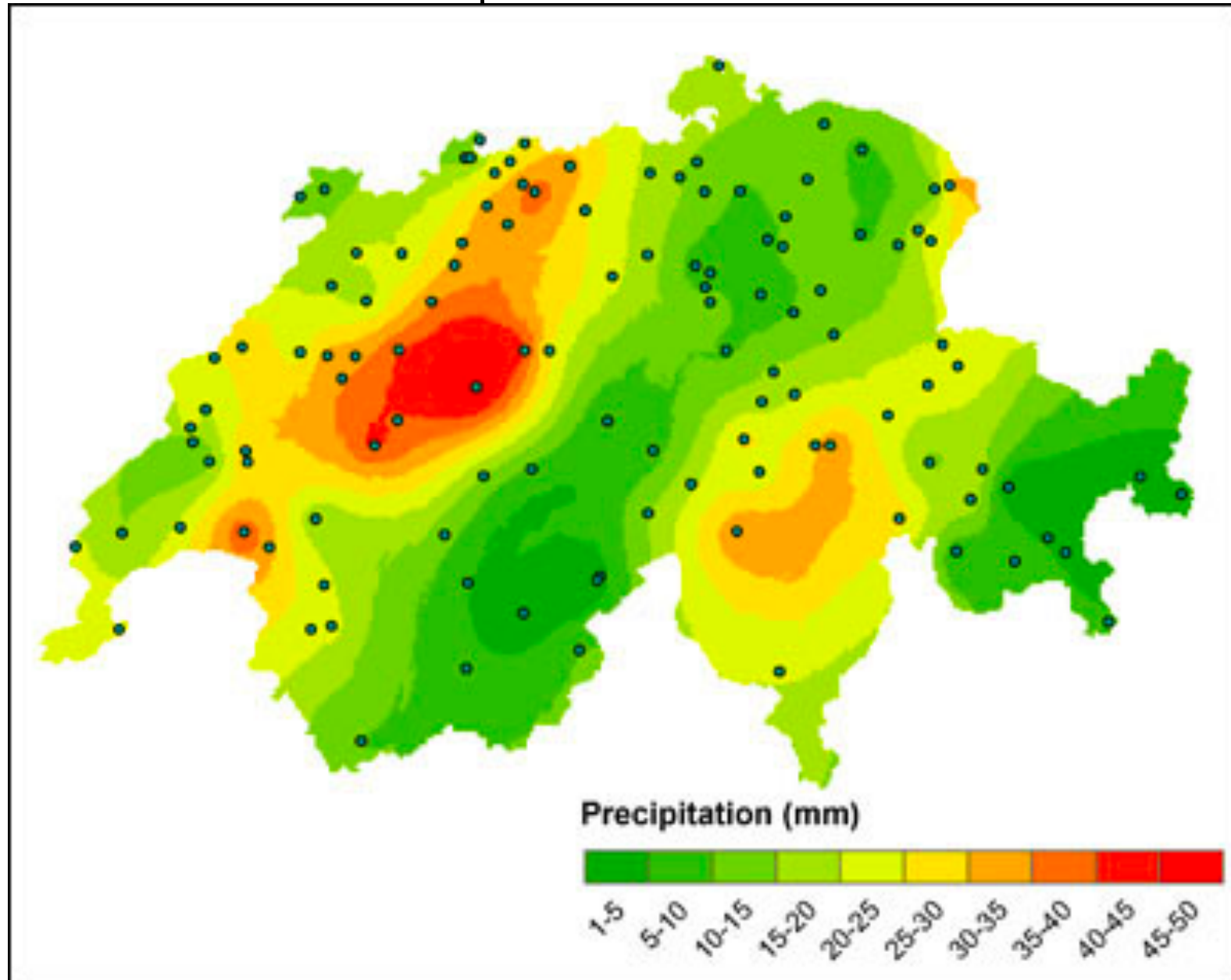
$$\mathbf{k}(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1), \dots, k(\mathbf{x}, \mathbf{x}_N)]^T,$$

$$k(\mathbf{x}, \mathbf{x}') = \theta_0 \left(\exp\left(-\frac{\theta_1}{2} \|\mathbf{x} - \mathbf{x}'\|^2\right) + \theta_2 + \theta_3 \mathbf{x}^T \mathbf{x}' \right).$$

$\Theta = [\theta_0, \theta_1, \theta_2, \theta_3]$ is the vector of hyper-parameters.

Application: Rainfall in Switzerland

The circles represent actual measurements



This is how the rainfall map was generated.