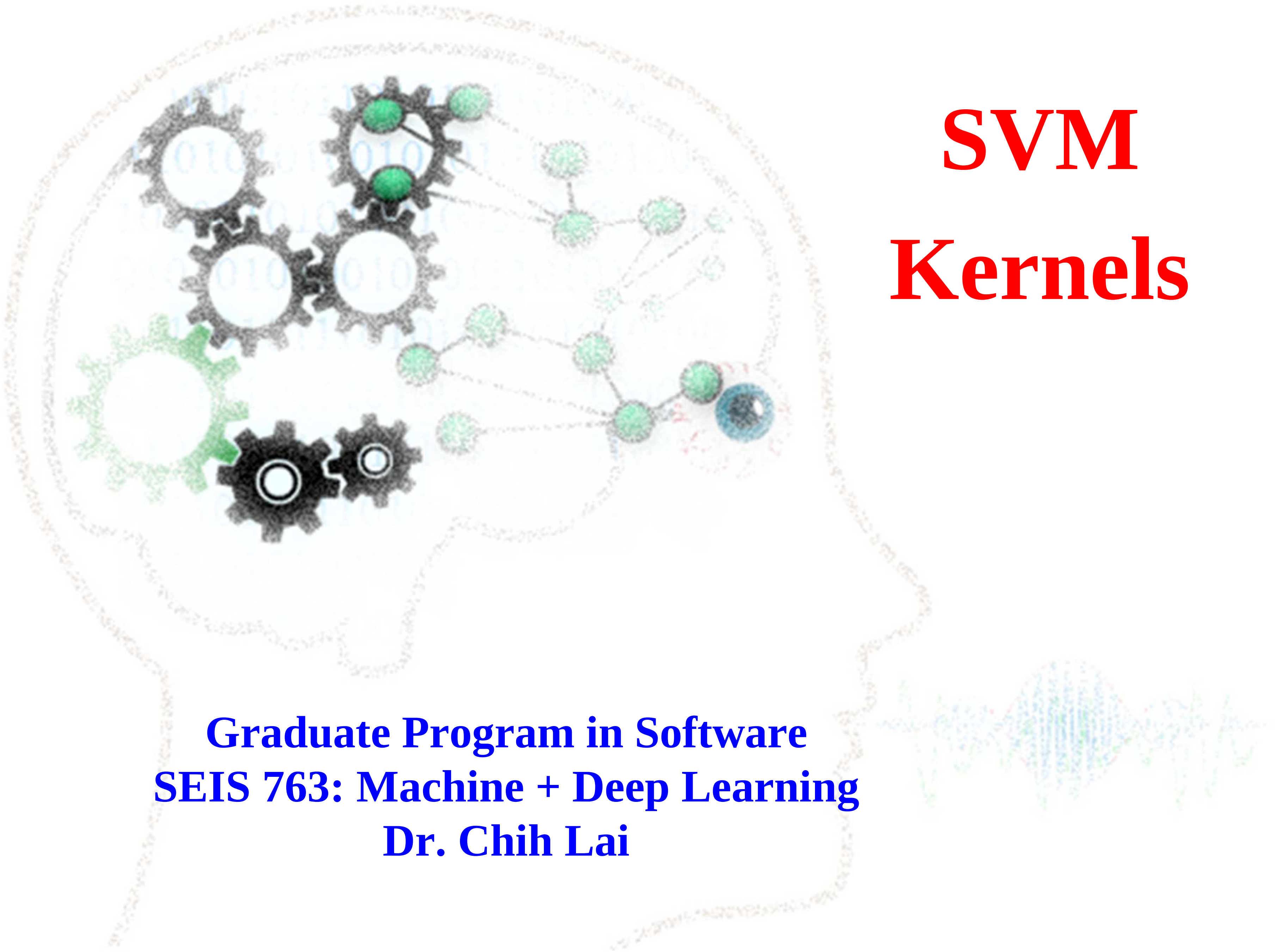
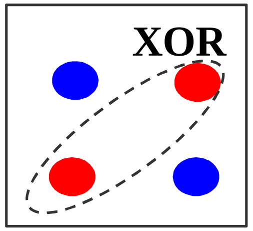


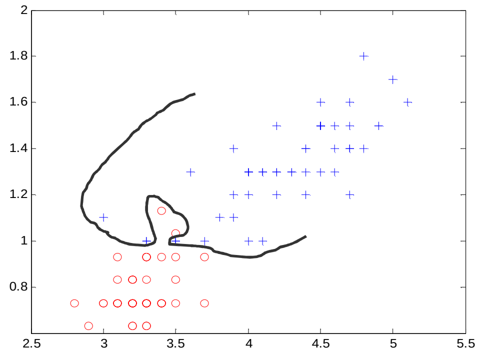
SVM Kernels

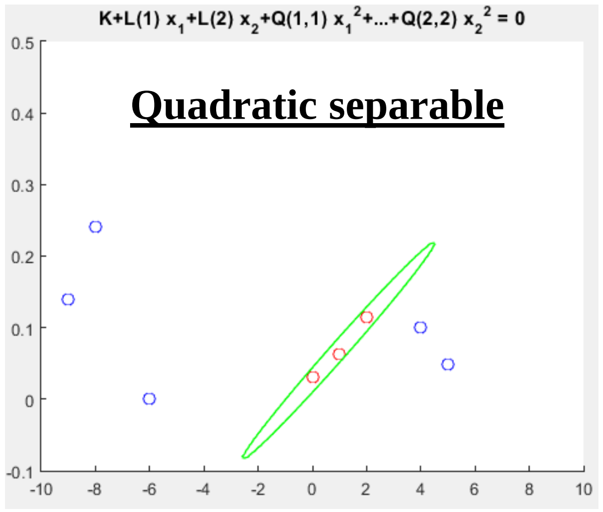
Graduate Program in Software
SEIS 763: Machine + Deep Learning
Dr. Chih Lai

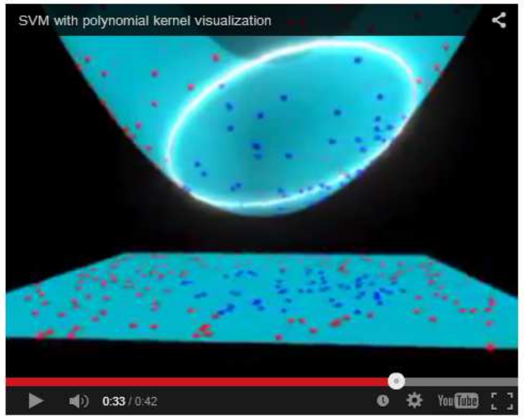
Separability

- Linear separability
  - Data can be separated into classes w/ a linear straight line in the original space.
- Linear non-separable data. (Famous XOR problem.)
  - Separable if use <u>more complex models</u> w/ non-linear decision boundary.
- Xform features so **dimension↑** &or **degree↑** ➔ more linear separable in H-D!!
  - **Visualizing "separability"** in high-D space??









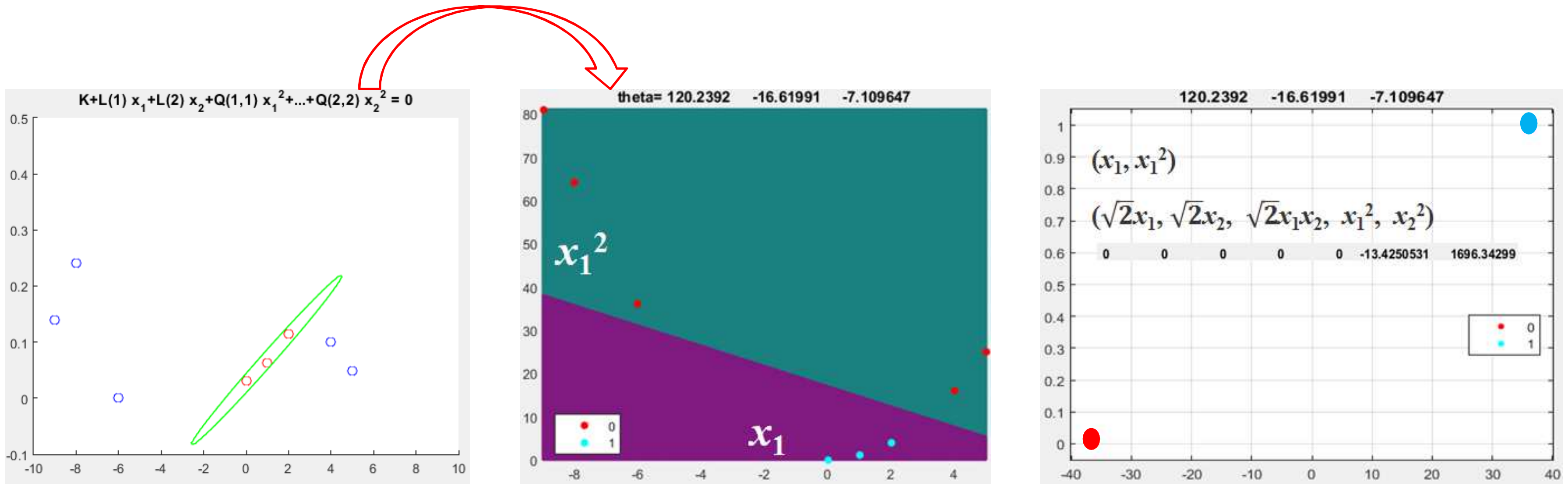
https://www.youtube.com/watch?v=3liCbRZPrZA&feature=youtu.be

Non-Linearly Separable in Low-D ≠ Non-Separable in High-D

- Transform features to H-D so you can apply simple linear model…
  - $\boldsymbol{D} = \{(\boldsymbol{x}, \boldsymbol{y})\}$ ➔ $\hat{y}(\boldsymbol{x}) = \theta_0 + \theta_1\mathbf{x} + \theta_2\mathbf{x}^2 + \ldots$ (~linear solution low-D)
  - $\boldsymbol{\Phi}(\boldsymbol{D}) = \{([\boldsymbol{x}, \boldsymbol{x}^2, \ldots], \boldsymbol{y})\}$ ➔ $\hat{y}(\boldsymbol{x}) = \theta \times \Phi(\boldsymbol{D}) = \theta_0 + \theta_1\hat{\boldsymbol{x}}_1 + \theta_2\hat{\boldsymbol{x}}_2 + \ldots$ (linear in H-D)

➢ The linear separation in $\Phi(\boldsymbol{x})$ space = a quadratic separation in original space.

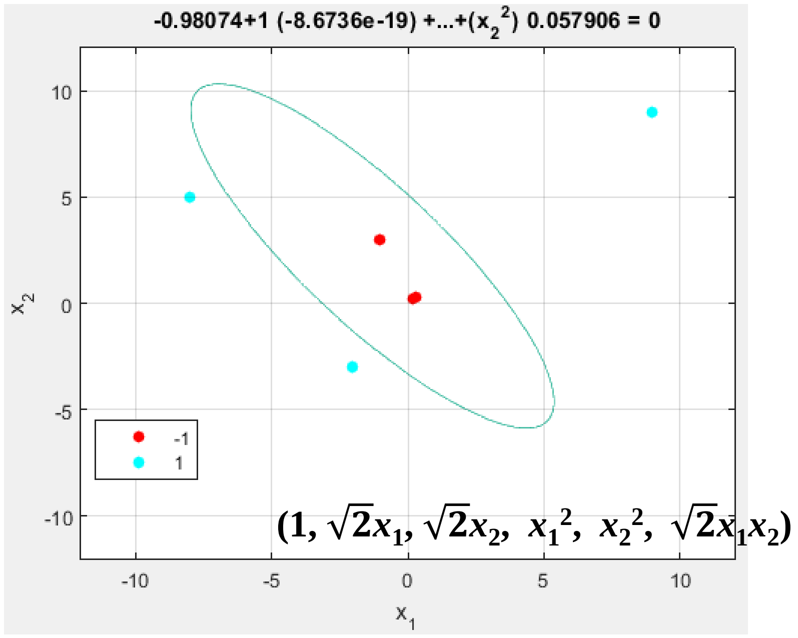
- (x_1, x_2) ➔ non-linearly separable.
- (x_1, x_2, x_1x_2) or (x_1, x_2, x_2^2) ➔ non-linearly separable.
- (x_1, x_1^2) ➔ **linearly separable**.
- $(\sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$ ➔ **linearly separable in H-D**. **(i.e. use simple linear model in H-D)**

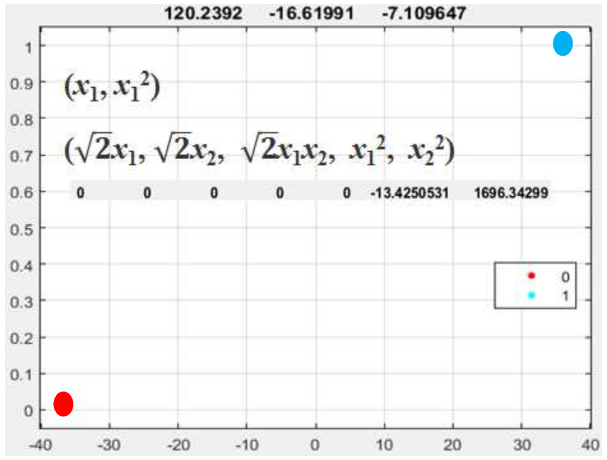


Solution in 6-D = Solution in 2-D

- $A = x^{(i)} = (x_1, x_2)$ ➔2-D, $\Phi(x^{(i)}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$. ➔6-D

```
X = [0.2 0.2; 0.3 0.3;   -1 3; -2 -3; -8 5; 9 9];
Y = [-1 -1  -1 1  1  1]';
XP = [ones(length(X(:,1)), 1),   1.414.*X(:,1), ...
    1.414.*X(:,2),  1.414.*X(:,1).*X(:,2),   X(:,1).^2, ...
    X(:,2).^2];                 %% xfer data to 6-D
ConvStr = {'1', '1.414*x1', '1.414*x2', ...
         '1.414*x1*x2', 'x1^2', 'x2^2'};

svm = fitcsvm(XP, Y);    %% ← **Linear**  SVM
fstr = num2str(svm.Bias);
for i = 1 : length(svm.Beta),
   fstr = [fstr '+' ConvStr{i} '*' num2str(svm.Beta(i))];
end
figure, ezplot(fstr, [-12 12 -12 12])   % plot 6-D solution in 2D
hold on, gscatter(X(:,1), X(:, 2), Y, '', '', 20),
hold off, grid on
```



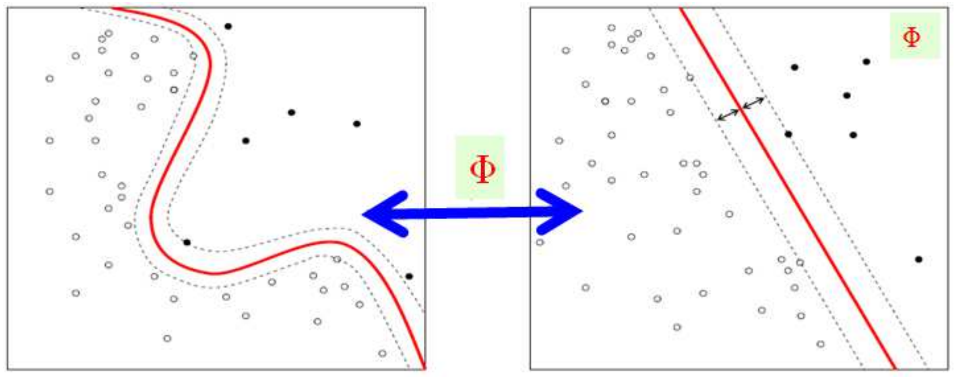


Transformed Features into ∞-**space**???

- SVM a very good linear classifier, but, many nonlinear datasets. Solutions?

  1. Collect more "**real**" features. 2. Add more "**transformed**" features.

- Transform function $x \rightarrow \Phi(x)$. /ˈfaɪ/
  - Add more dimensions, we may find a linear separation in high-D space.
  - Map data to a HD space → **linear** methods operate in HD space will behave **non-linearly** in the original input space.



https://en.wikipedia.org/wiki/Support_vector_machine

- What transformation is good enough? What dimension is good enough? **∞??**
  - Too much computation??
  - Where exactly is the **"too much" computation?**

Dot Product between **EACH Pair** of Training Data

- Objective function:
  - $min_{w,b}\ C \times \sum_{i=1}^{m}[max(0, (1 - y^{(i)}(wx^{(i)} + b))] + \frac{1}{2}||w||^2 = min_w[CE + L].$
- Solve by using Lagrange multiplier α_i for every point *i*:
  - Min $L_P = \frac{1}{2}||w||^2 - \sum_{i=1}^{m}[\alpha_i(1 - y^{(i)}(wx^{(i)} + b))]$ **(Primary form)**
  - $\frac{\partial L_P}{\partial w} = 0 \rightarrow\ w = \sum_{i=1}^{m}\alpha_i y_i x_i$ $\frac{\partial L_P}{\partial b} = 0 \rightarrow\ \sum_{i=1}^{m}\alpha_i y_i = 0$ [F1]
  - Karush-Kuhn-Tucker (*KKT*) conditions: $\alpha_i \geq 0$ and $[\alpha_i(1 - y^{(i)}(wx^{(i)} + b))] = 0$
  - Substitute [F1] to primary form we get dual form as:

- Min $L_D = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x_i x_j} - \sum_{i=1}^{m}\alpha_i$ **(Dual form)**
- Min $L_D = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \Phi(\boldsymbol{x_i})\Phi(\boldsymbol{x_j}) - \sum_{i=1}^{m}\alpha_i$
  - Decision boundary: $\boldsymbol{wX} + \boldsymbol{b} = 0 \rightarrow (\sum_{i=1}^{m}\alpha_i y_i x_i X) + \boldsymbol{b} = 0.$

- **dot products**

$X1 = \begin{bmatrix}1\\2\\3\end{bmatrix} \quad X2 = \begin{bmatrix}1\\0\\-1\end{bmatrix}$

$\Longrightarrow$ **Do we really need to compute $\Phi(\boldsymbol{x_i})\Phi(\boldsymbol{x_j})$?**

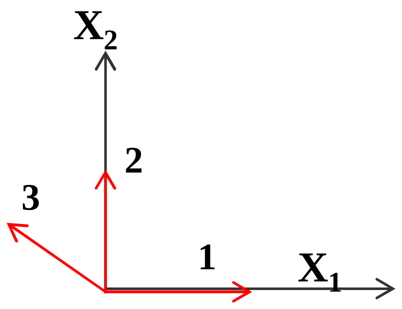
$\Longrightarrow$ **Do we really need to "physically" reach the ∞-space?**

Dot Product = Similarity

- A^TA = similarity (dot product) between column vectors of (**normalized**) A.
  - Dot products between every pair of **column** vectors in two matrices A and B = A^TB.
  - Dot product↓ ➔ angle↑, similarity↓.   dot product↑ ➔ angle↓, similarity↑.

$$\theta = \begin{bmatrix} 0 \\ 4 \end{bmatrix}, \quad X = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 2 & 3 \end{bmatrix},$$

$$\hat{y} = \theta^T \boldsymbol{X} = [0 \ \ 4] \times \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 2 & 3 \end{bmatrix} = [4 \ \ 8 \ \ 12].$$



- **dot products.** (between data pts)

$$X1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad X2 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

```
% assuming column vectors

a = [[1; 0]  [0; 1] [-1; 1]]

norms = sqrt(diag(a'*a))'

na = a ./ repmat(norms, 2, 1)

b = na' * na

acosd(b)
```

```
a
     1     0    -1
     0     1     1

na
  1.0000         0   -0.7071
       0    1.0000    0.7071

b = na' * na

    1.0000         0   -0.7071

         0    1.0000    0.7071

   -0.7071    0.7071    1.0000

acosd(b) =

        0     90.0000  135.0000
   90.0000         0    45.0000
  135.0000   45.0000         0
```

Use Kernel Functions

- Define kernel function $\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to **avoid** real mapping $\Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j)$ into high-D.
  - Define a kernel function as $\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x}_j)$ or = $<\Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j)>$
    - Define a kernel function $\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ **as** doing dot-product in high-D space.
    - **But**, doing only dot products in the original space **w/ other simple ops**.
  - Kernel functions are also referred to as ***similarity functions***.

- Again, what do we mean $\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x}_j)$?
  1. During training (i.e. learning)…
  2. Whenever we need dot product $\Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x}_j)$ in **H-D**…
  3. Just compute $\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in the **original space**.
     - Doing only dot products in the original space **w/ other simple ops**.
     - **NO** need to actually perform Φ to H-D ➔ Xformation exists only "***implicitly***" (conceptually)
  - **Examples???**

Kernel Tricks– A Simple Example

- Transform a D-dimension original data $x^{(i)}$ into high-dimensional data.
  - Let $A = (a_1, a_2)$ ➔**2-D**, $\Phi(A) = (a_1^2, \ a_2^2, \ \sqrt{2}a_1a_2)$. ➔**3-D**
  - Let $B = (b_1, b_2)$, $\Phi(A) \bullet \Phi(B)$ requires intensive computation & memory.
  - But, we know $\Phi(A) \bullet \Phi(B) = a_1^2b_1^2 + a_2^2b_2^2 + 2a_1b_1a_2b_2 = (a_1b_1 + a_2b_2)^2 = (A \bullet B)^2$.
  - **In other words, $\Phi(A) \bullet \Phi(B) = (A \bullet B)^2 = K(A, B) = (0 + A \bullet B)^2$.** ➔**2-D**
  - Dot product in the ***feature space*** = dot product in the original space **w/ other simple ops**.
  - **NO** need to actually transform data into high-D and do dot product in high-D.
  - Only need to compute a kernel matrix (Gram matrix) that contains dot products of original vectors.
- So now we can **substantially** increase the # of features for our classifier.
  - How about $\Phi(A) \bullet \Phi(B) = K(A, B) = (\mathbf{C} + A \bullet B)^{\mathbf{100}}$ ➔ **Computations still in 2-D.**

http://www.wolframalpha.com/input/?i=(a1*b1%2Ba2*b2)%5E2

Kernel Example, 2-D

- Original data points $A = (x_1, x_2)$ ➔**2-D.**

P1	6	3
P2	7	3
P3	4	3
P4	5	3
P5	6	2

- dot(***A***, ***B***)

	P1	P2	P3	P4	P5
P1	45	51	33	39	42
P2	51	58	37	44	48
P3	33	37	25	29	30
P4	39	44	29	34	36
P5	42	48	30	36	40

- $\Phi(A) = (x_1^2,\ \ x_2^2,\ \ \sqrt{2}x_1x_2)$.➔**3-D**

P1	36	9	25.4558
P2	49	9	29.6985
P3	16	9	16.9706
P4	25	9	21.2132
P5	36	4	16.9706

- $\mathbf{\Phi(\mathit{A}) \bullet \Phi(\mathit{B}) = (0 + \mathit{A}^T\mathit{B})^2 = (0 + dot(\mathit{A}, \mathit{B}))^2}$
  - **Gram Matrix**

	P1	P2	P3	P4	P5
P1	2025	2601	1089	1521	1764
P2	2601	3364	1369	1936	2304
P3	1089	1369	625	841	900
P4	1521	1936	841	1156	1296
P5	1764	2304	900	1296	1600

http://www.wolframalpha.com/input/?i=(a1*b1%2Ba2*b2)%5E2

Kernel Tricks– Another More Complex Example

- Transform a D-dimension original data x_i into high-dimensional data.
  - For example, let $A = x_i = (x_1, x_2)$, $\Phi(x^{(i)}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$. $\rightarrow$**6-D**
  - After transformation, x has H-D [i.e. $O(D^2)$–dim], intensive computation and memory.
  - $\Phi(x_i) \bullet \Phi(x_j)$ again requires intensive computation and memory.
  - But, we know $\Phi(x_i) \bullet \Phi(x_j) =$

    $$1 + 2\sum_{d=1}^{D} x_{di}x_{dj} + \sum_{d=1}^{D} x_{di}^2 x_{dj}^2 + 2\sum_{d1=1,d2=1,d1<d2}^{D} x_{d1}^{(i)} x_{d2}^{(i)} x_{d1}^{(j)} x_{d2}^{(j)} = (1 + x_i \bullet x_j)^2.$$

  - **In other words, $\Phi(x^{(i)}) \bullet \Phi(x^{(j)}) = (1 + x_i \bullet x_j)^2 = K(x^{(i)}, x^{(j)}) = (1 + A \bullet B)^2$.** $\rightarrow$**2-D**
  - Dot product in the feature space = dot product in the original space **w/ other ops**.
  - No need to actually transform data into high-D and do dot product in high-D.
  - Only need to compute a kernel matrix (Gram matrix) that contains dot products of original vectors.
- So now you can **substantially** increase the # of features for your classifier.
  - How about $\Phi(A) \bullet \Phi(B) = K(A, B) = (C + A \bullet B)^{100}$ $\rightarrow$ **Computations still in 2-D.**

http://www.wolframalpha.com/input/?i=(a1*b1%2Ba2*b2)%5E2

Kernel Example, 6-D

- Original data points $A = (x_1, x_2)$ ➔**2-D.**

P1	0	1
P2	1	1
P3	6	1
P4	9	1

- dot(***A***, ***B***)

	P1	P2	P3	P4
P1	1	1	1	1
P2	1	2	7	10
P3	1	7	37	55
P4	1	10	55	82

- $\Phi(\boldsymbol{A})=(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$➔**6-D**

P1	1	0	1.4142	0	1	0
P2	1	1.4142	1.4142	1	1	1.4142
P3	1	8.4853	1.4142	36	1	8.4853
P4	1	12.7279	1.4142	81	1	12.7279

- $\boldsymbol{\Phi(A) \bullet \Phi(B) = (1 + A^TB)^2 = (1 + \text{dot}(A, B))^2}$
  - **Gram Matrix**

	P1	P2	P3	P4
P1	4	4	4	4
P2	4	9	64	121
P3	4	64	1444	3136
P4	4	121	3136	6889

http://www.wolframalpha.com/input/?i=(a1*b1%2Ba2*b2)%5E2

Visiting The ∞-Space

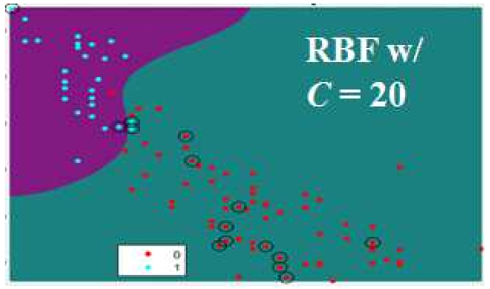
- How about $\Phi(x_i) \bullet \Phi(x_j) = K(x_i, x_j) = (C + A^TB)^{100}$ ➔ **2-D to ???**
  - Computations only happen in 2-D.
  - How does that ***Z-space*** look like? **Do we care?** Only exist conceptually.

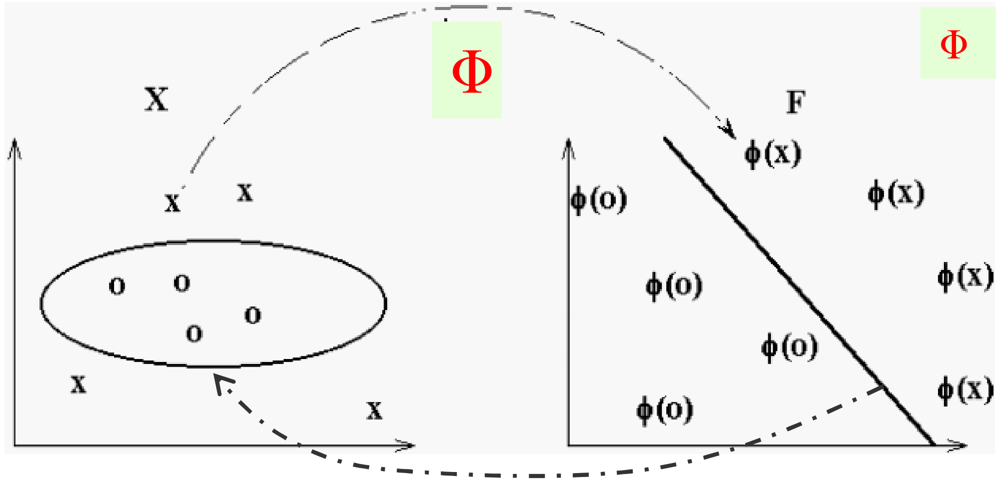
- How about $\Phi(x_i) \bullet \Phi(x_j) = K(x_i, x_j) = (C + A^TB)^{\infty}$ ➔ **2-D to ???**
  - Computations only happen in 2-D.
  - How does that ***Z-space*** look like? **Do we care?** Only exist conceptually.

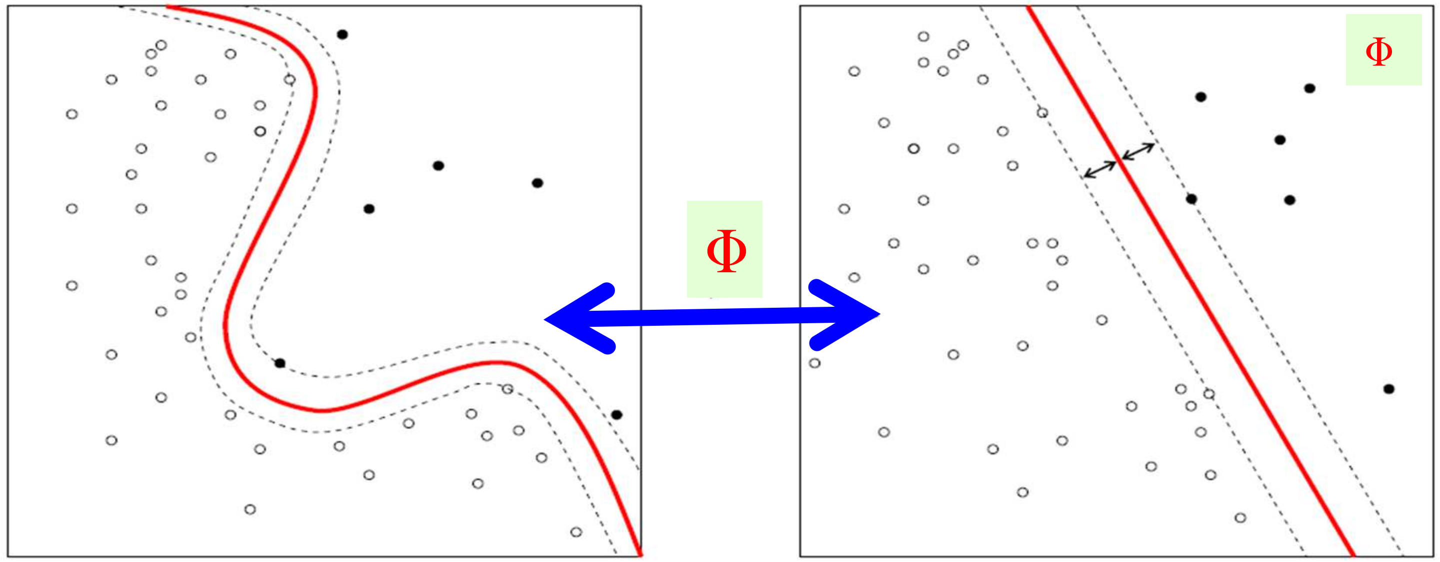
- Transform a D-dimension original data $x^{(i)}$ into ∞-dimensional data.
  - **In other words, $\Phi(x_i) \bullet \Phi(x_j) = (C + x_i^Tx_j)^? = K(x_i, x_j) = (C + A \bullet B)^?$.**
  - Dot product in the ***feature space*** **=** dot product in the original space **w/ other ops**.
  - No need to actually transform data into high-D and do dot product in high-D.
  - Only need to compute a kernel matrix (**Gram matrix**) that contains dot products of original vectors.

Basic Kernel Idea

- Using a simple <u>**linear SVM w/ Kernel**</u> in H-D, nonlinear separations can be learned **efficiently**.







https://en.wikipedia.org/wiki/Support_vector_machine

Kernel in A Short & Plain English

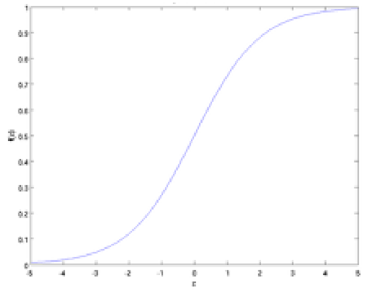
- In machine learning "kernel" is usually used to refer to the kernel trick.
  - A method of using a **linear** classifier to solve a non-linear problem by …
  - Mapping the original points into a higher-dimensional space s.t. …
  - Data points can be separated by a “linear hyperplane” in the H-D space.
  - Kernel maps data to H-D (or ∞-D), hoping data becomes more easily separated.
  - ➢ The mapping, **however**, hardly needs to be computed because of ***kernel trick***.
  - Kernel method provides a simple bridge from linearity to non-linearity for algorithms which can be expressed in terms of dot products. **(similarity)**

What Kind of “*Similarity*” Functions?   Various Kernels

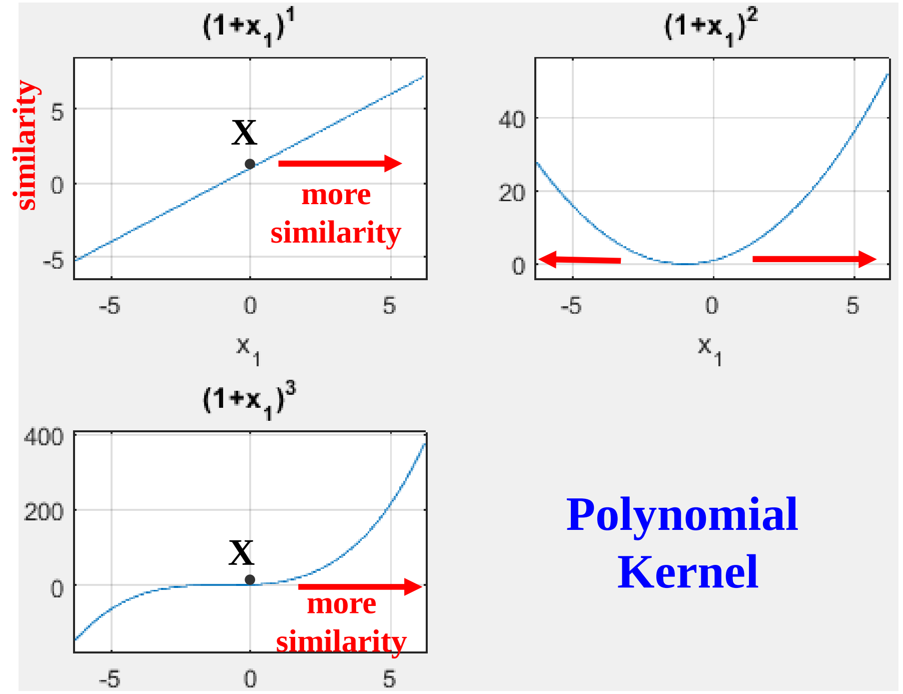
- **Polynomial Kernel:** $K(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{c} + \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b})^{d}$.

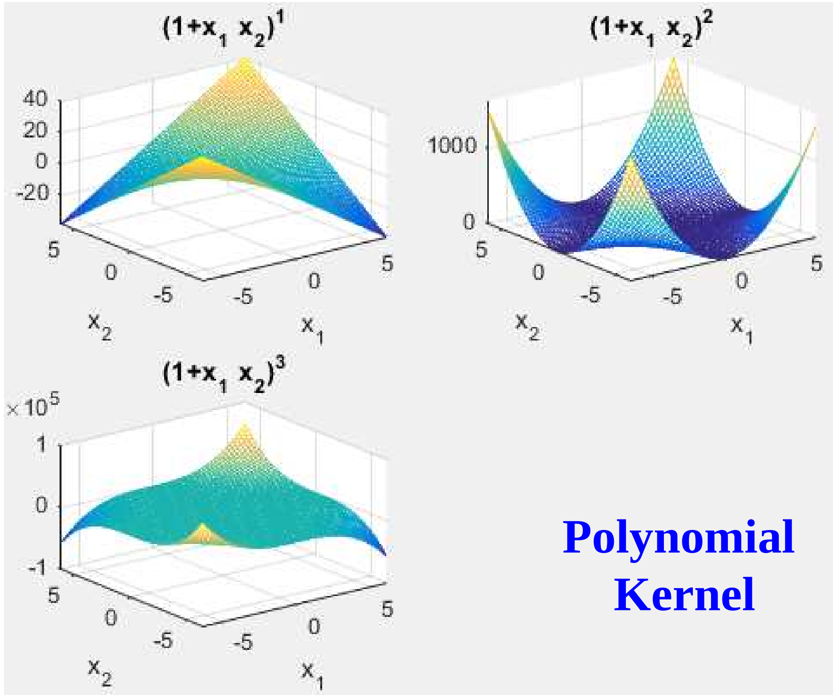
$a1^2\, b1^2 + 2\, a1\, a2\, b1\, b2 + 2\, a1\, b1 + a2^2\, b2^2 + 2\, a2\, b2 + 1$

- ***RBF Kernel***: $K(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\|\boldsymbol{a} - \boldsymbol{b}\|^2) / 2\sigma^2 = e^{\frac{-\|a-b\|^2}{2\sigma^2}}$.
  - Radial Basis Function (next slide).



- Sigmoid-like: $K(\boldsymbol{a}, \boldsymbol{b}) = \tanh(\boldsymbol{c}\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{h})$





for i = 1 : 3, subplot(2,2,i), ezplot(['(1+x1)^' num2str(i)]); grid on, end

for i = 1 : 3, subplot(2,2,i), ezmesh(['(1+x1*x2)^' num2str(i)]); grid on, end

RBF Similarity (Radial Basis Function)

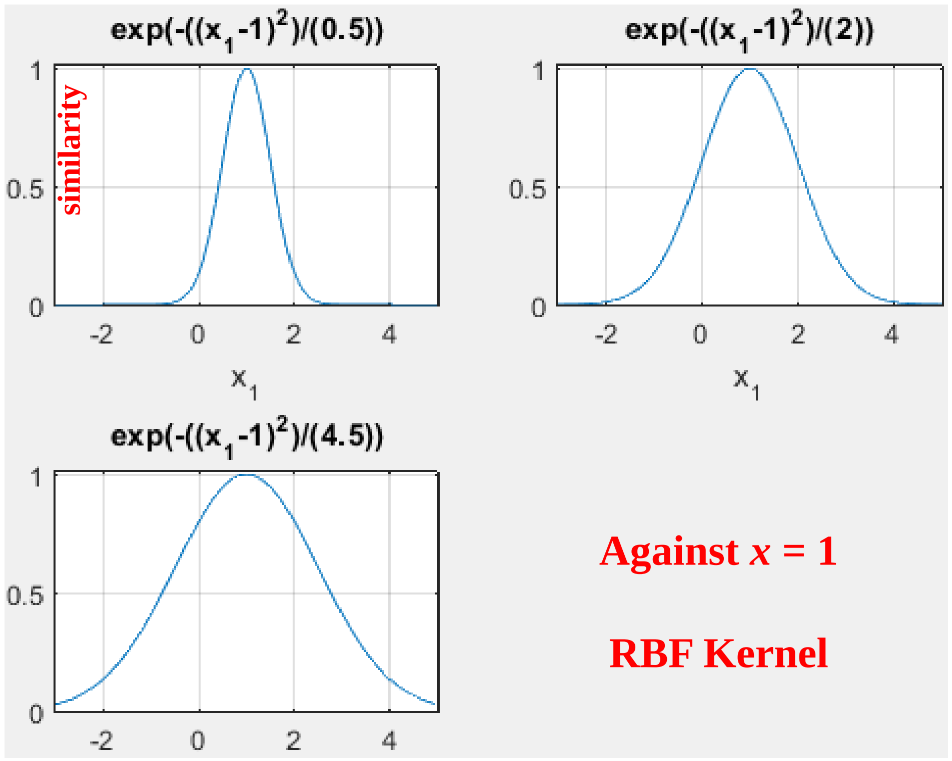
Why RBF is in ∞-D? See Appendix

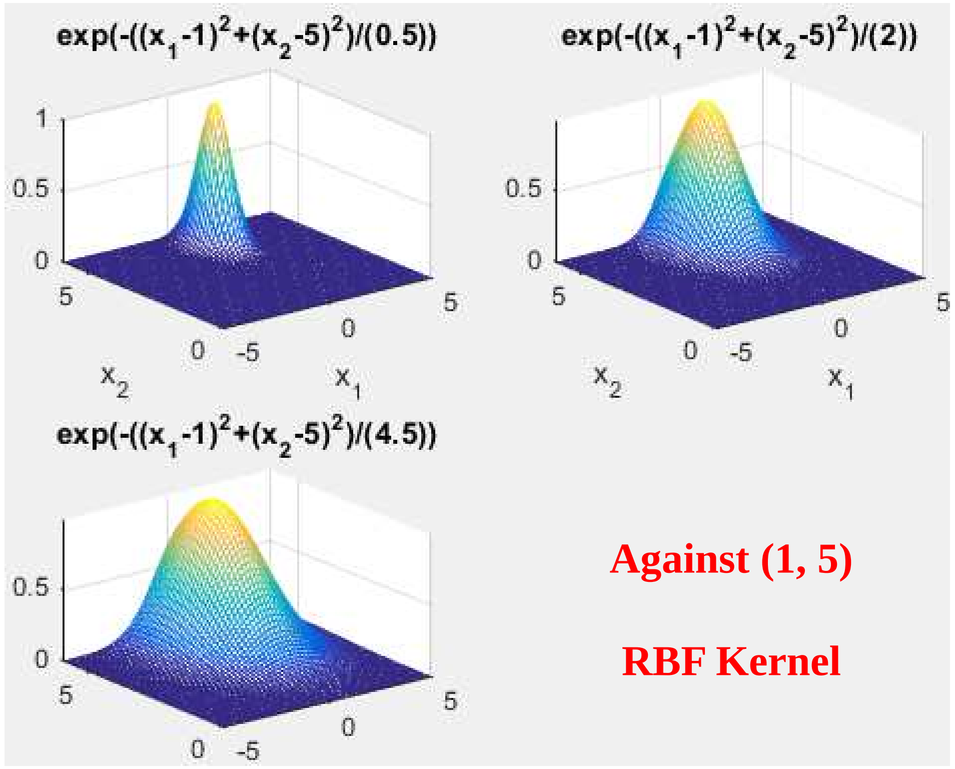
- $K(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\|\boldsymbol{a} - \boldsymbol{b}\|^2) / 2\sigma^2 = e^{\frac{-\|a-b\|^2}{2\sigma^2}}$.
  - σ (SD) can be used as **another** way of ***regularization***.
  - $\sigma\uparrow$, data similarity$\uparrow$ ➔ regularization$\uparrow$, error (bias)$\uparrow$, variance$\downarrow$.
  - $\sigma\downarrow$, data similarity$\downarrow$ ➔ regularization$\downarrow$, error (bias)$\downarrow$, variance$\uparrow$.

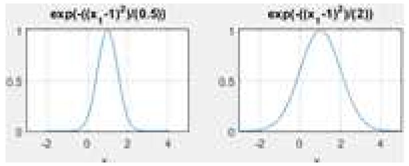
PDF $\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

($\sigma\uparrow$, 1/exp($\downarrow$) ≈ 1)

($\sigma\downarrow$, 1/exp($\uparrow$) ≈ 0)







Sigma = 1

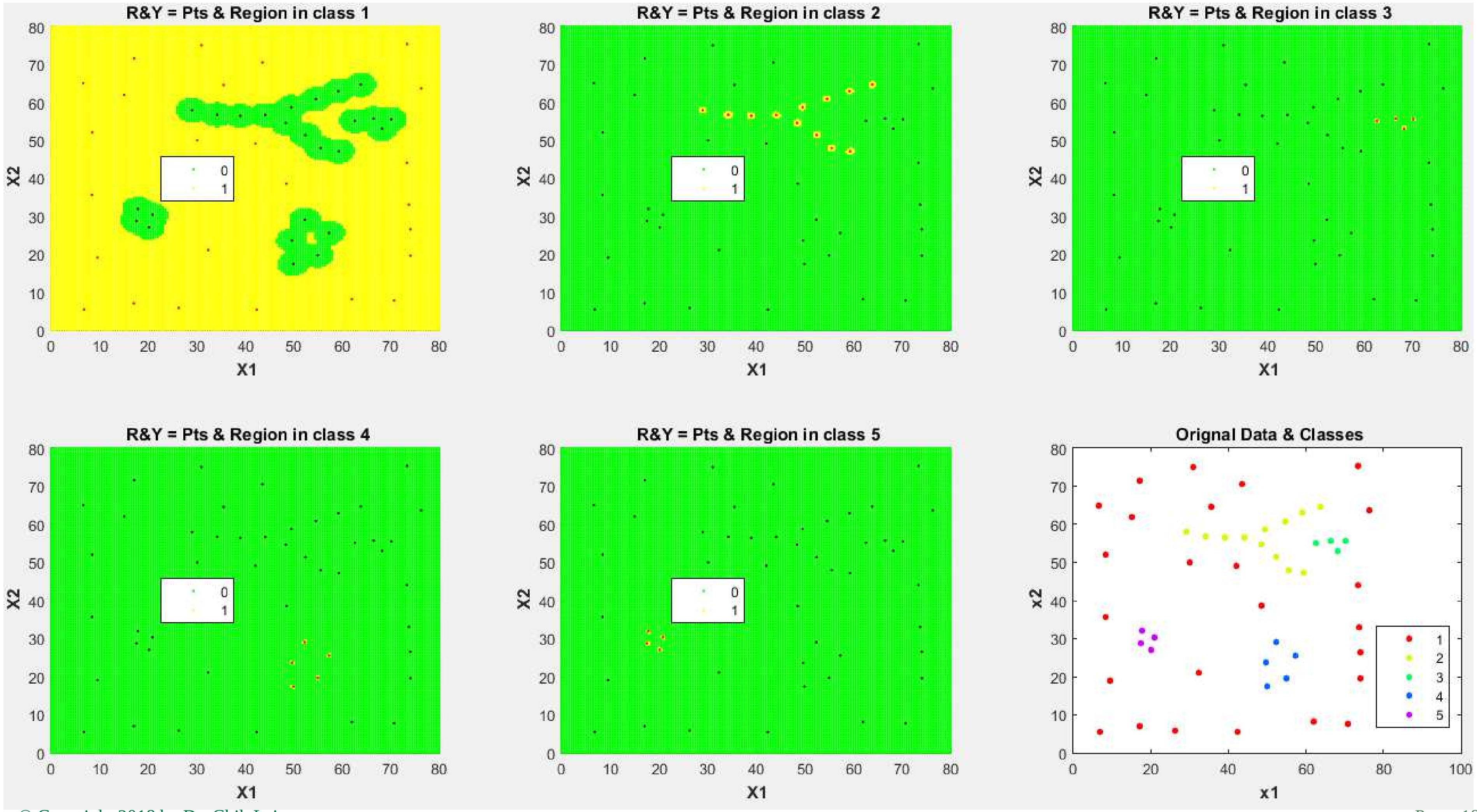
σ↑, similarity↑ ➔ regularization↑, error (bias)↑.

σ↓, similarity↓ ➔ regularization↓, error (bias)↓.

- $K(\boldsymbol{a}, \boldsymbol{b}) = e^{\frac{-||a-b||^2}{2\sigma^2}}$.

fitcsvm(X, Y, 'KernelFunction', 'rbf', 'KernelScale', 1, 'BoxConstraint', 1)

SVC(C=1.0, kernel = ’rbf’, gamma = ’auto’)



$K(a,b) = e^{\frac{-||a-b||^2}{2\sigma^2}}$.

Sigma = 2

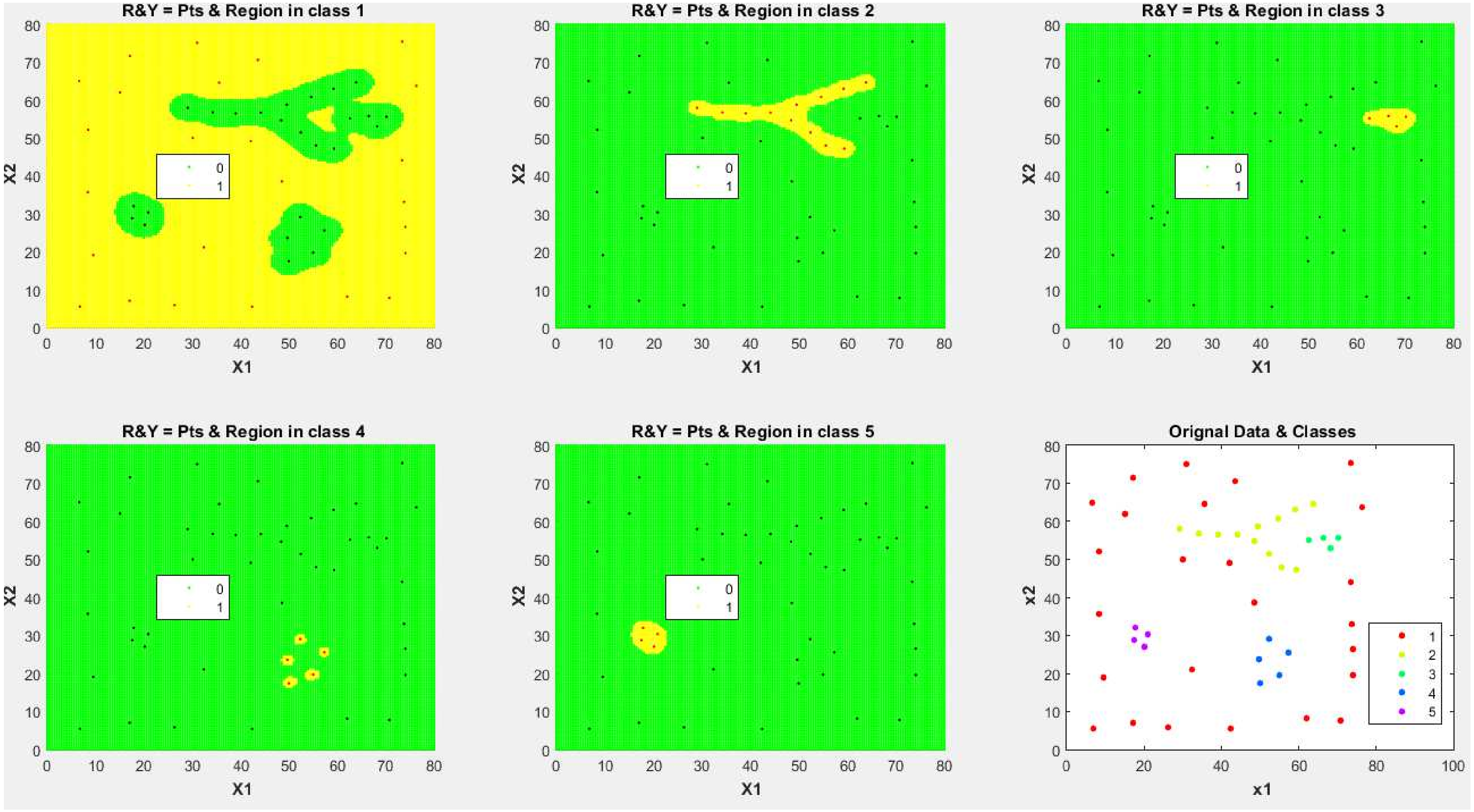
σ↑, similarity↑ ➔ regularization↑, error (bias)↑.

σ↓, similarity↓ ➔ regularization↓, error (bias)↓.

- $\boldsymbol{K}(\boldsymbol{a}, \boldsymbol{b}) = e^{\frac{-||a-b||^2}{2\sigma^2}}$.

fitcsvm(X, Y, 'KernelFunction', 'rbf', 'KernelScale', 1, 'BoxConstraint', 1)

SVC(C=1.0, kernel = ’rbf’, gamma = ’auto’)



$K(a, b) = e^{\frac{-||a-b||^2}{2\sigma^2}}$.

Sigma = 3

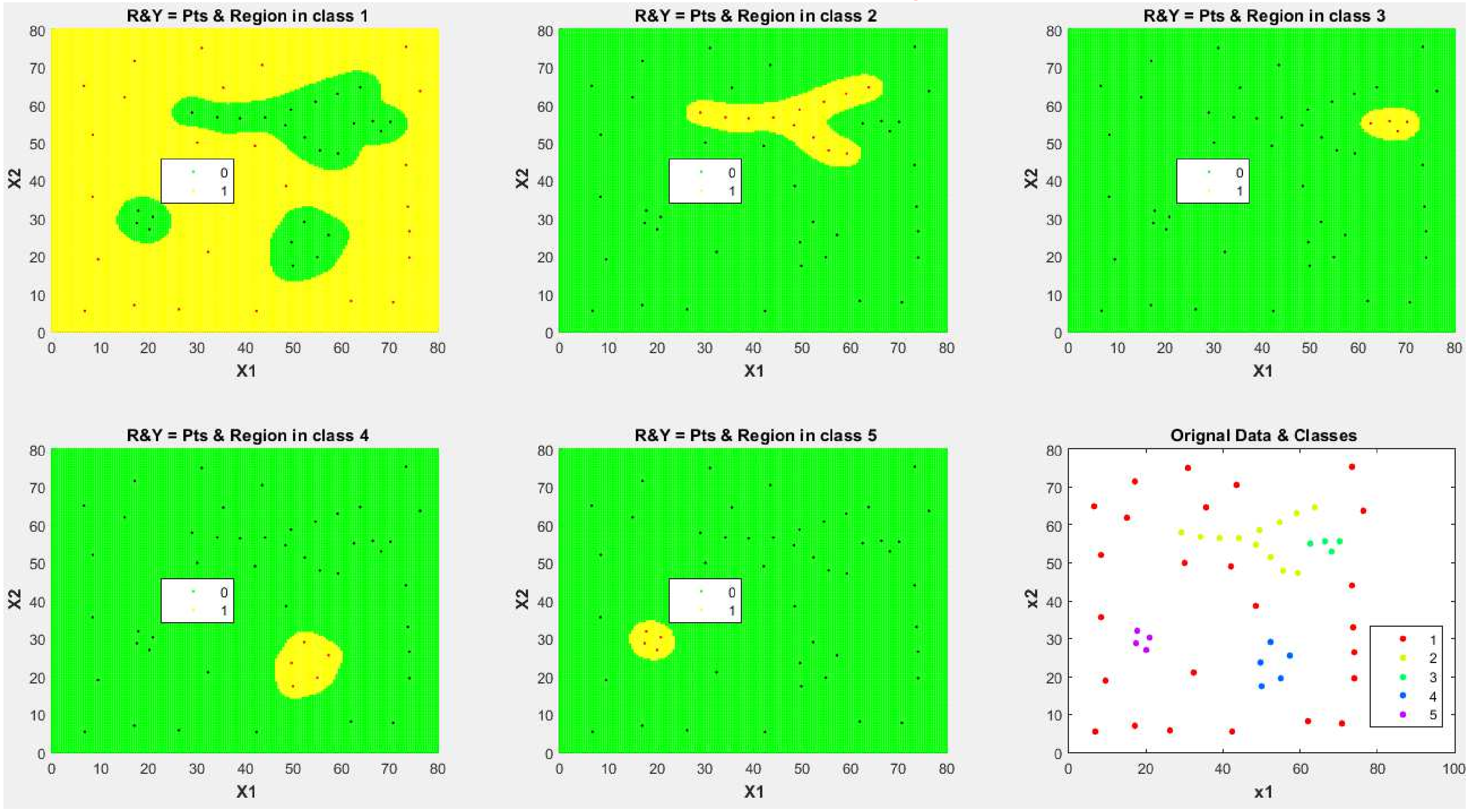
σ↑, similarity↑ ➔ regularization↑, error (bias)↑.

σ↓, similarity↓ ➔ regularization↓, error (bias)↓.

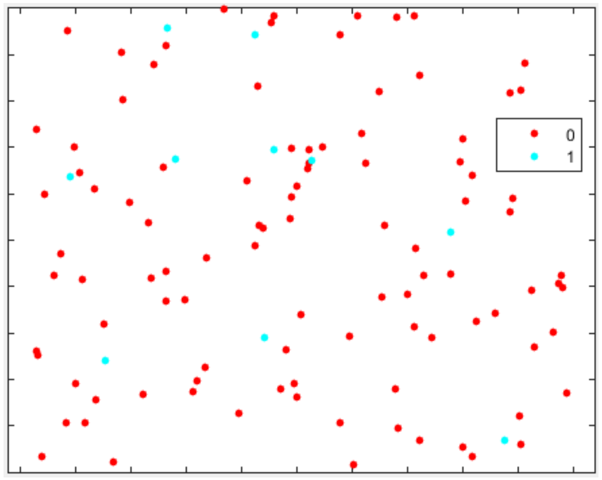
- $K(a, b) = e^{\frac{-||a-b||^2}{2\sigma^2}}$.

fitcsvm(X, Y, 'KernelFunction', 'rbf', 'KernelScale', 1, 'BoxConstraint', 1)

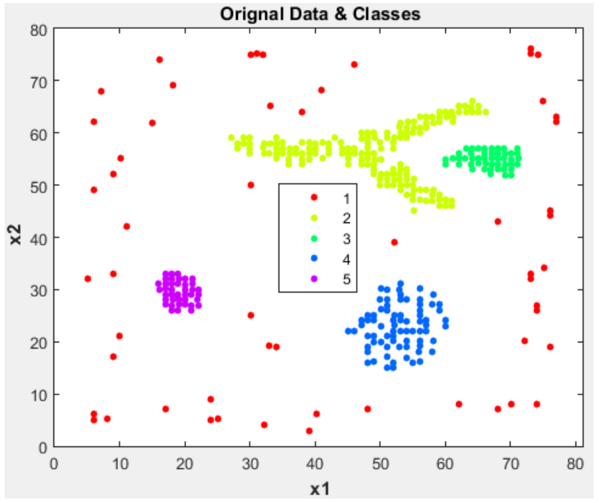
SVC(C=1.0, kernel = ’rbf’, gamma = ’auto’)

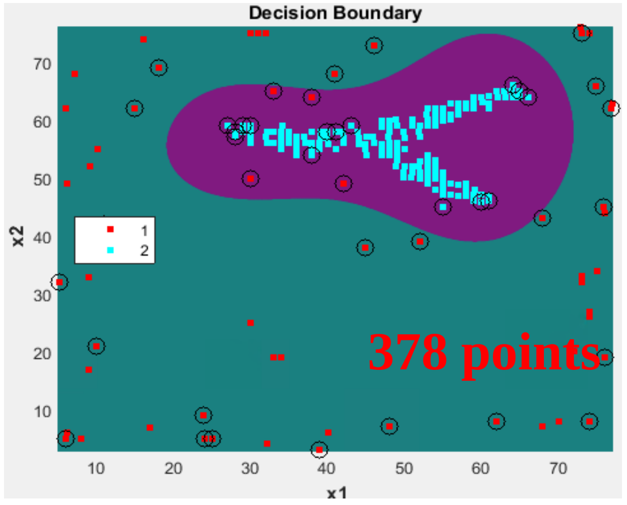


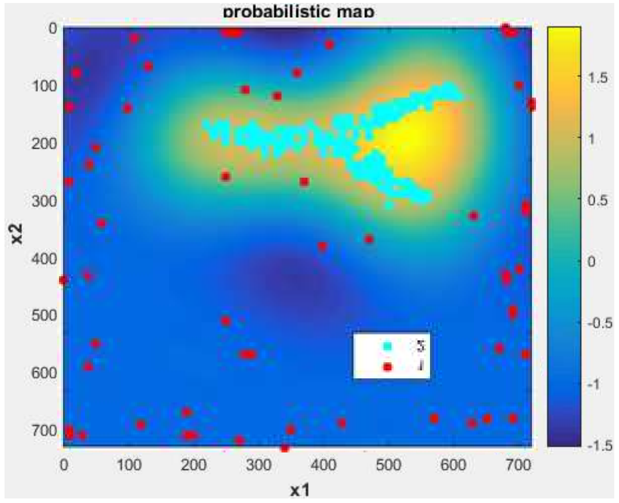
RBF with Y-Shape Data



- Build an SVM over class 1 and 2.
  - Points in Class 1 are outliers.
  - Decision boundary ≈ contour of Y-shape.
  - Note that SVs are in the ***Z***-space.
  - SVs concentrate in the protruding part of Y-shape.
  - Can we narrow the decision boundary to be closer to the Y-shape?
  - Do we want do achieve that?
  - Real-world problem?



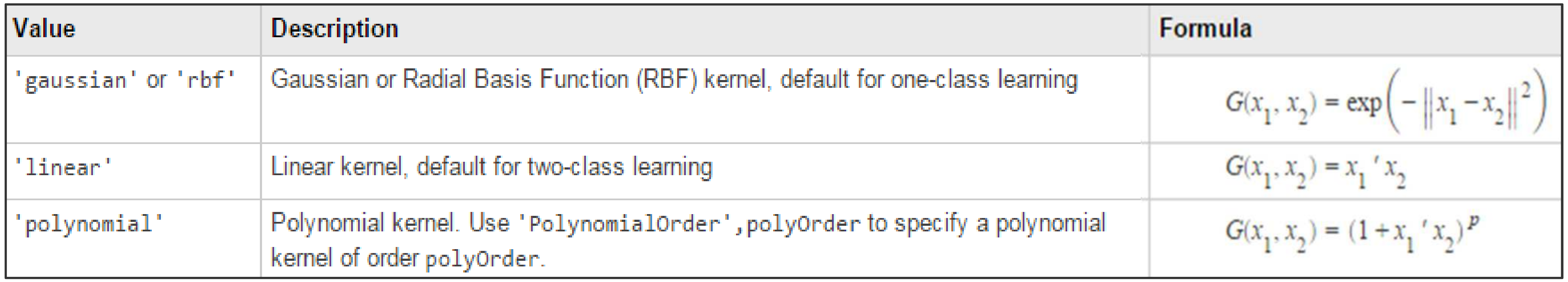




Why RBF is in ∞-D? See Appendix

Matlab Kernels

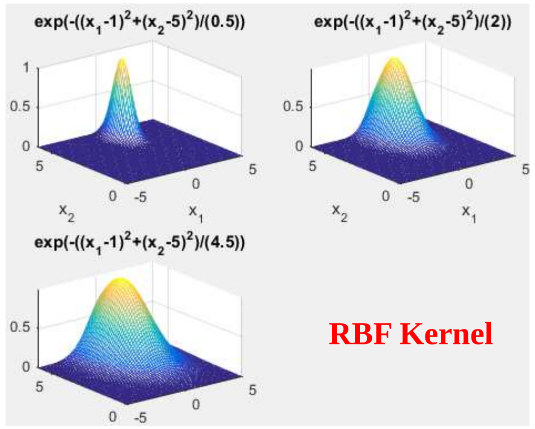
- ‘**KernelFunction**’ under http://www.mathworks.com/help/stats/fitcsvm.html



Value	Description	Formula
`'gaussian'` or `'rbf'`	Gaussian or Radial Basis Function (RBF) kernel, default for one-class learning	$G(x_1, x_2) = \exp\left(-\lVert x_1 - x_2 \rVert^2\right)$
`'linear'`	Linear kernel, default for two-class learning	$G(x_1, x_2) = x_1{}' x_2$
`'polynomial'`	Polynomial kernel. Use `'PolynomialOrder',polyOrder` to specify a polynomial kernel of order `polyOrder`.	$G(x_1, x_2) = (1 + x_1{}' x_2)^p$

- A polynomial kernel can model **feature conjunctions** up to any order polynomial.
  - i.e. **interactive effects** of features.

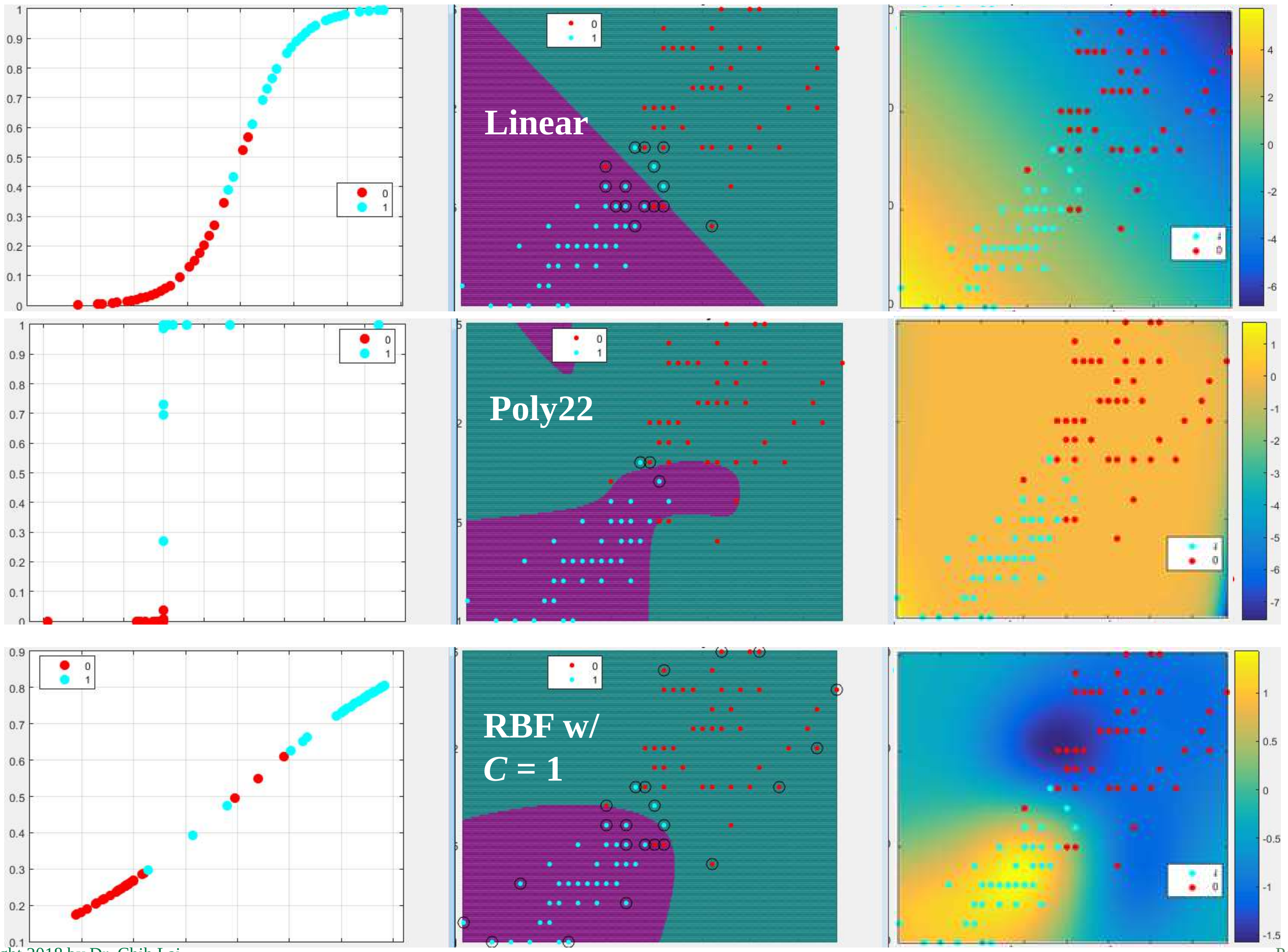
- Radial basis functions allows to pick out hyperspheres.
  - In contrast w/ linear kernel



More RBF Examples

- Examples of SVM with RBF kernel in the following slides.

Fisher Iris (2-Class)

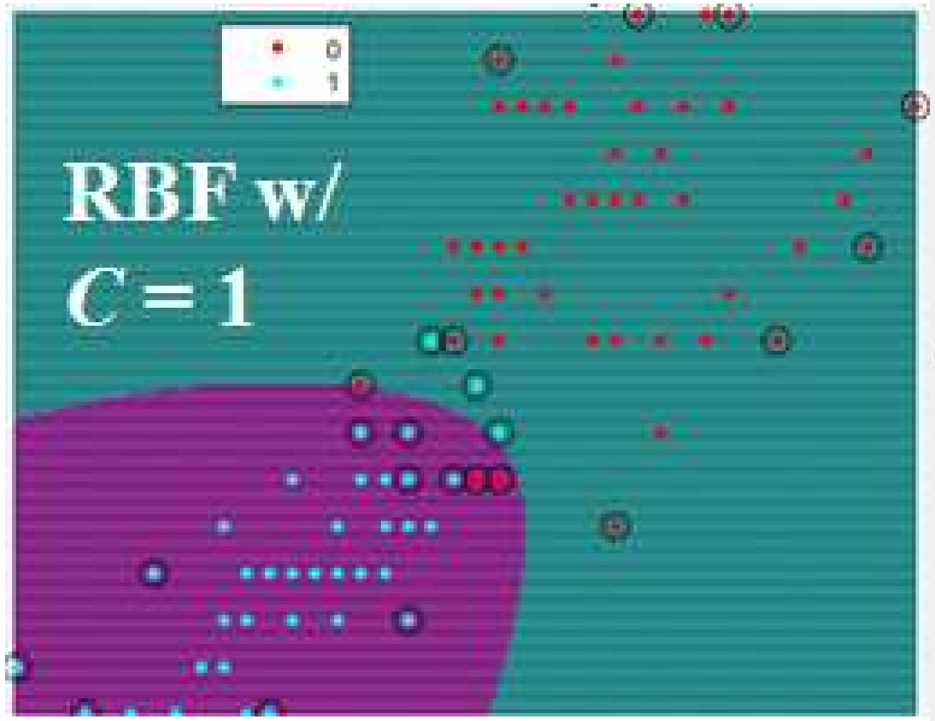


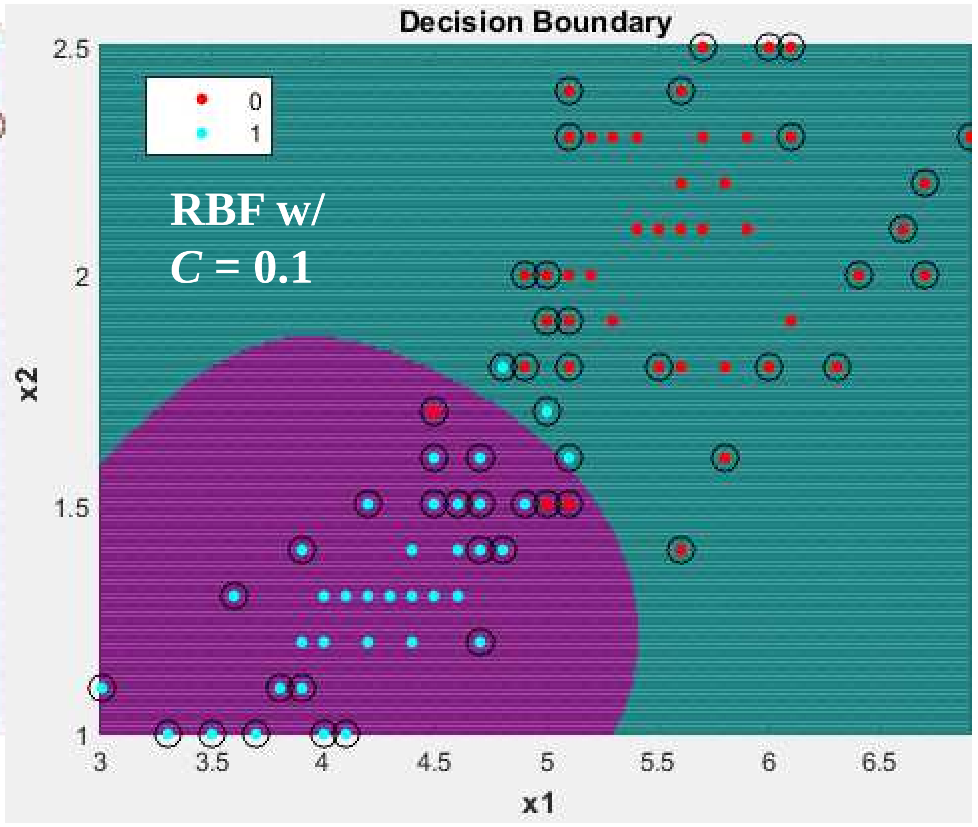
Fisher Iris (2-Class) RBF with different ***C***s

- **WHY** do we have such weird support vectors?

$$min_{w,b}[\boldsymbol{CE} + \boldsymbol{L}]$$

- **NOTE** that they are support vectors in the imaginary **Z-space**.





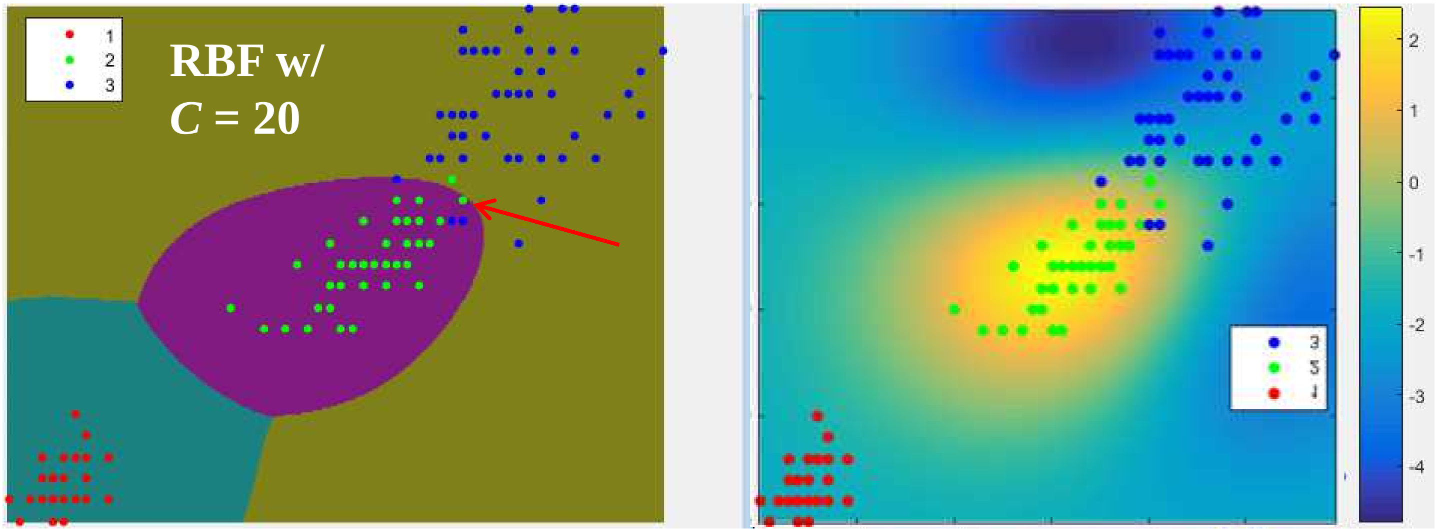
Fisher Iris (3-Class) RBF

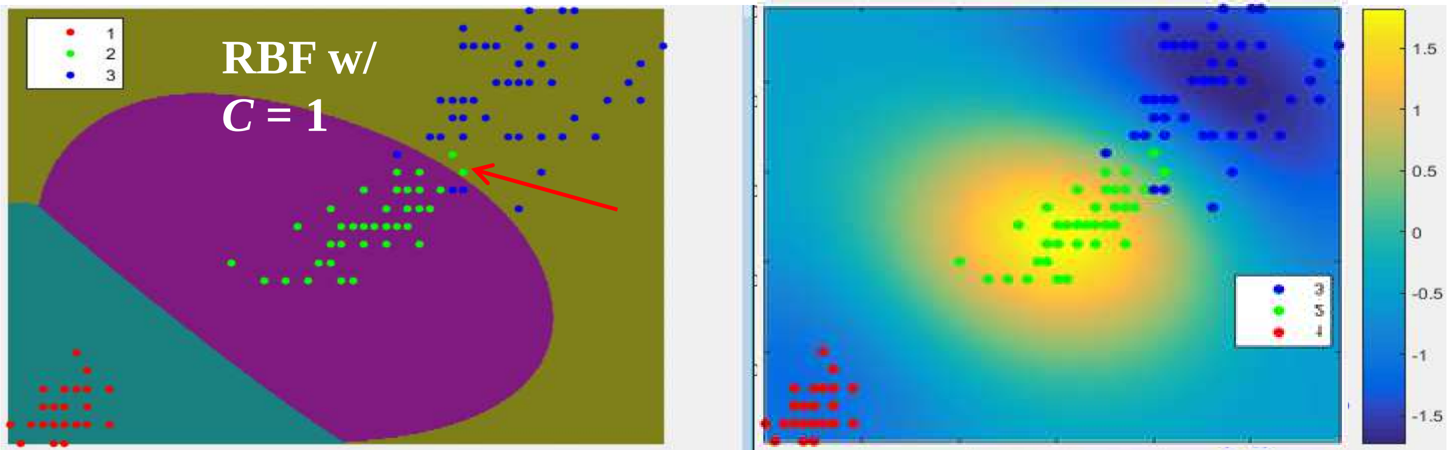
$min_{w,b}[\boldsymbol{C}\boldsymbol{E} + \boldsymbol{L}]$

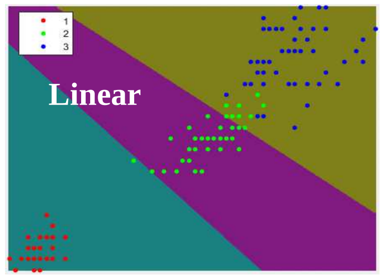
$C\uparrow$ ➔ Regularization↓

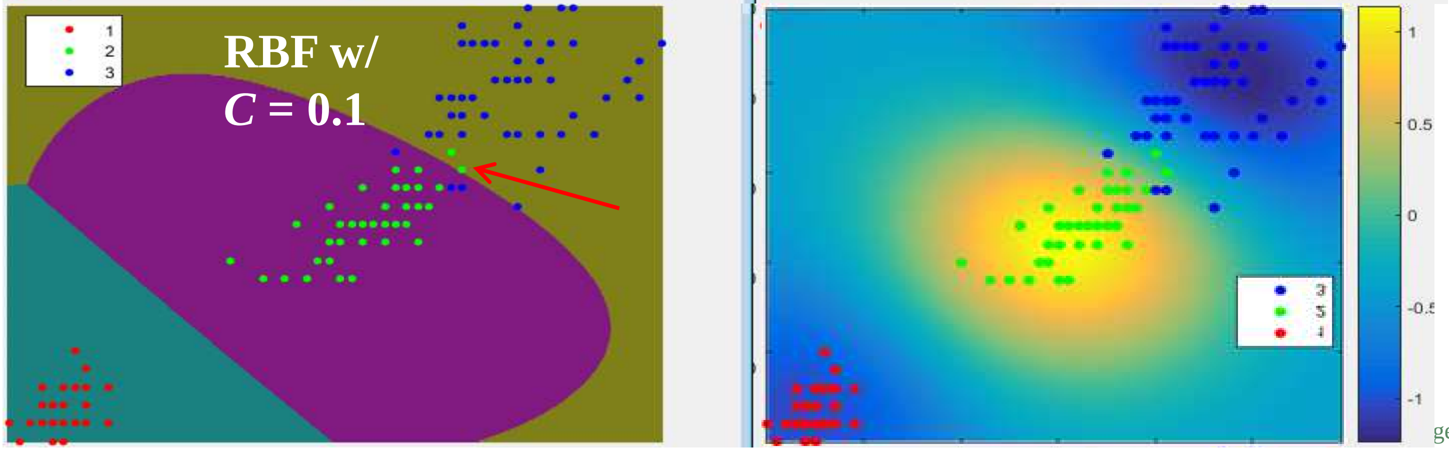
➔ $w\uparrow$ ➔ M↓ ➔ Err↓

➔ #SV↓









MPG + Weight, RBF

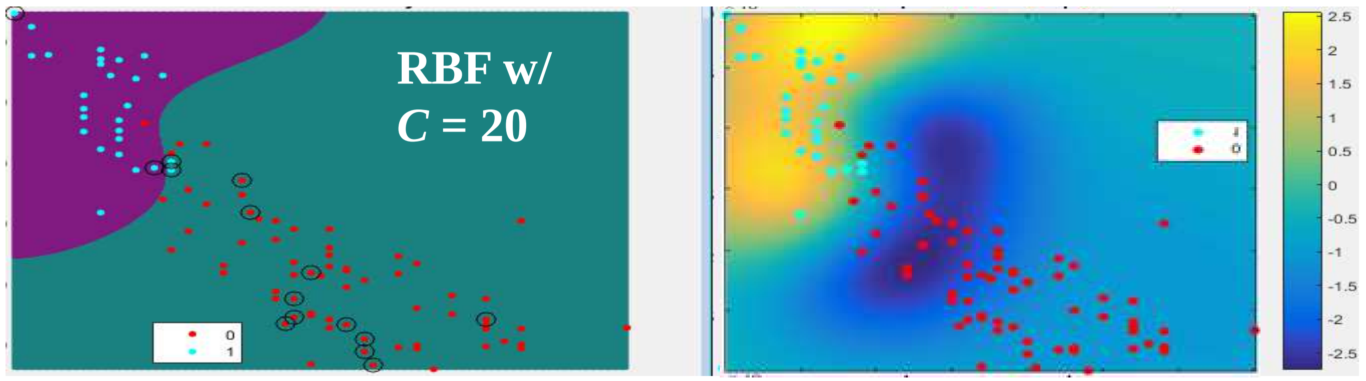
- Polynomial method not working well.

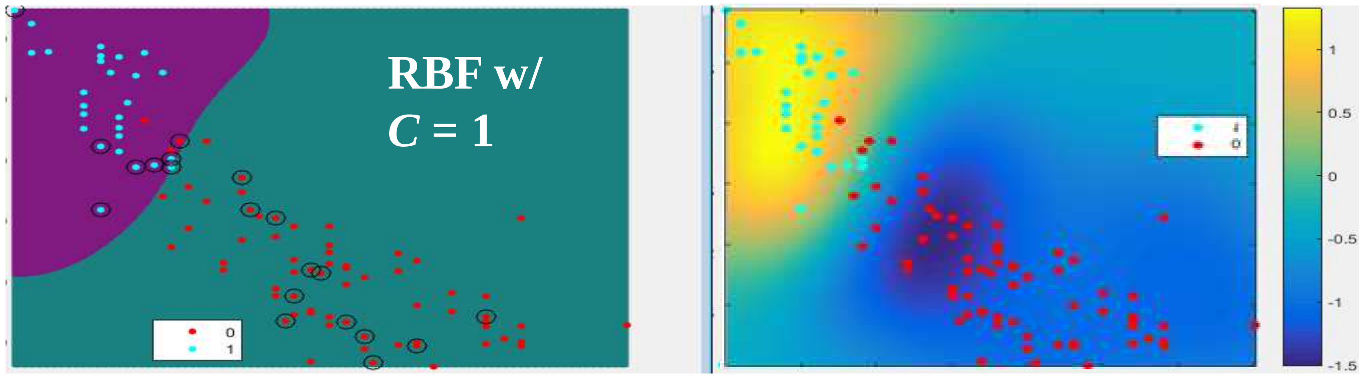
$$min_{w,b}[\boldsymbol{C}\boldsymbol{E} + \boldsymbol{L}]$$

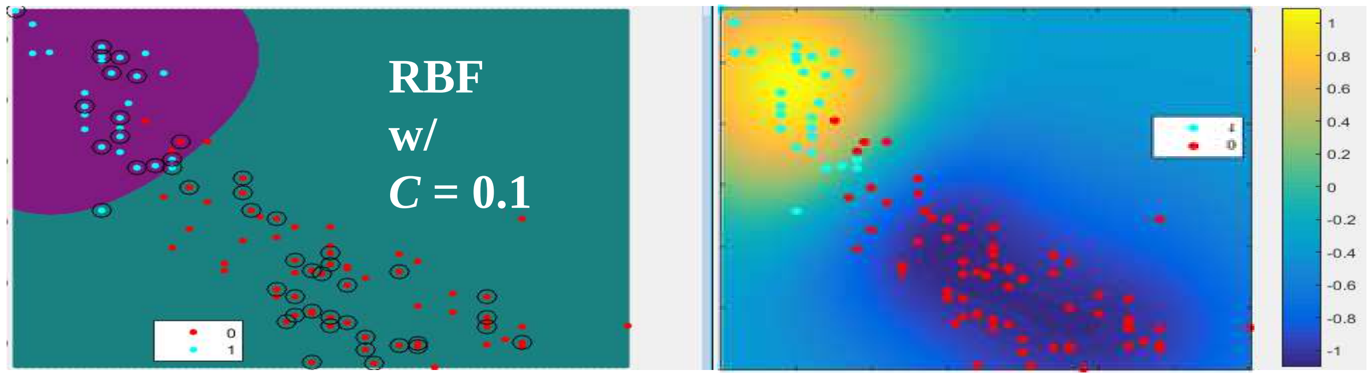
$C\uparrow$ ➔ Regularization$\downarrow$

➔ $w\uparrow$ ➔ M$\downarrow$ ➔ Err$\downarrow$

➔ #SV$\downarrow$







Feature Scaling before Using Kernels

- Feature scaling becomes more important when using kernels.
  - **Otherwise, kernel function will make greater values even bigger.**
  - New features (i.e. $\boldsymbol{x}^3$) grow fast from the original features.
  - (age, \$\$), (age^2, \$\$^{2}), (age^3, \$\$^{3}), … (age^{3000}, \$\$^{3000}), … …
  - $\Phi(x^{(i)}) \bullet \Phi(x^{(j)}) = K(x^{(i)}, x^{(j)}) = (\boldsymbol{C} + A^T B)^{\infty}$ $\Phi(\boldsymbol{A})=(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^{2}, x_2^{2}, \sqrt{2}x_1x_2)$
  - $\boldsymbol{K}(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\|\boldsymbol{a} - \boldsymbol{b}\|^{2}) / 2\sigma^2 = e^{\frac{-\|a-b\|^2}{2\sigma^2}} =$

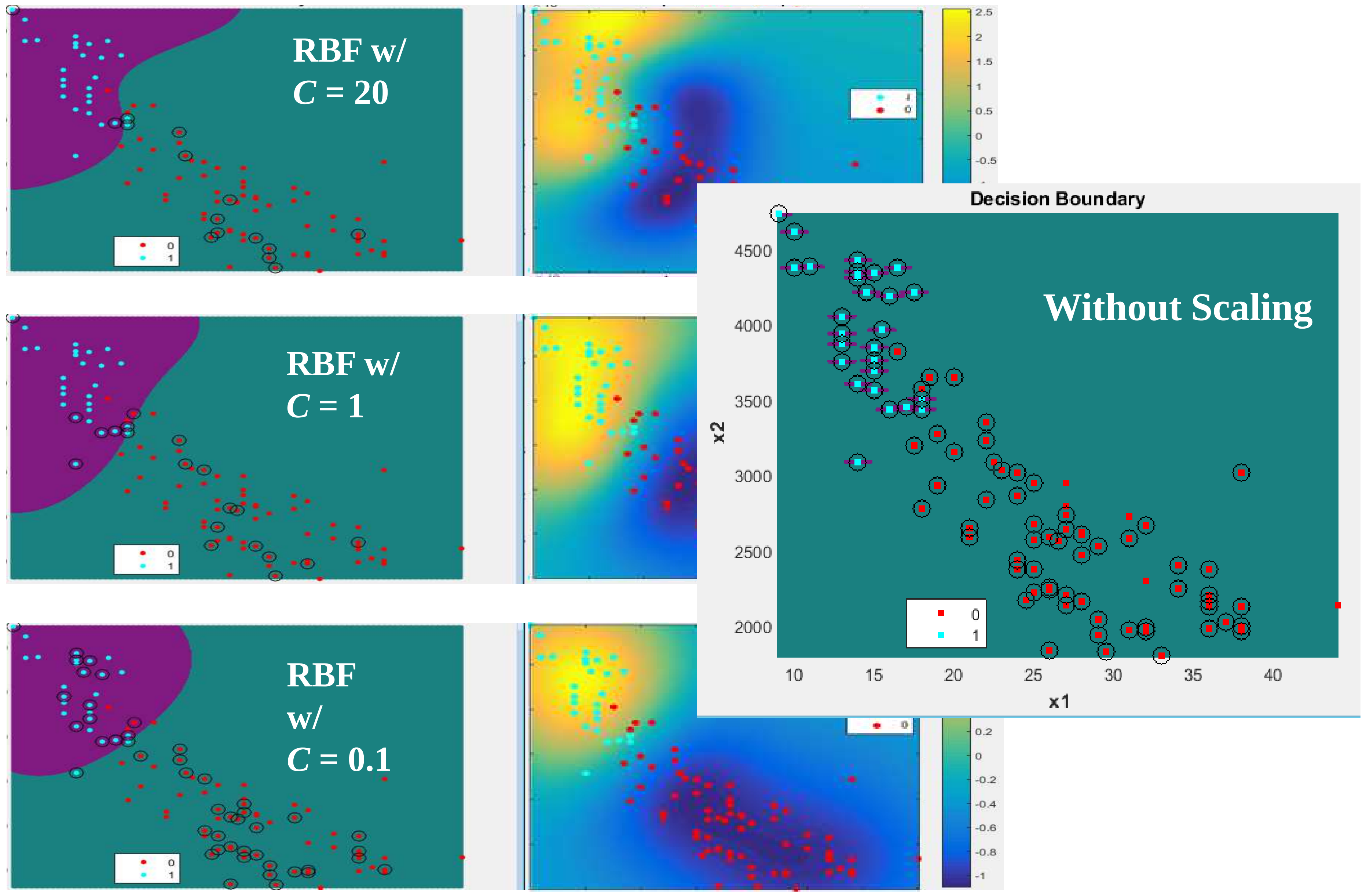
$$K(x, y) = \exp(-\|x - y\|^2) = \exp(-(x_1 - y_1)^2 - (x_2 - y_2)^2)$$

$$= \exp(-x_1^2 + 2x_1y_1 - y_1^2 - x_2^2 + 2x_2y_2 - y_2^2)$$

$$= \exp(-\|x\|^{2}) \times \exp(-\|y\|^{2}) \times \mathbf{exp(2x^Ty)}$$

$$k(x, y) = \exp(-\|x\|^2)\exp(-\|y\|^2)\boxed{\sum_{n=0}^{\infty}\frac{(2x^Ty)^n}{n!}}$$

MPG + Weight, RBF, Same Dataset, Feature Scaling Effects



Kernel Example, 6-D… Problems???

- Original data points $A = (x_1, x_2)$ ➔ **2-D.**

P1	0	1
P2	1	1
P3	6	1
P4	9	1

- dot(***A***, ***B***)

	P1	P2	P3	P4
P1	1	1	1	1
P2	1	2	7	10
P3	1	7	37	55
P4	1	10	55	82

Dot products between <u>EVERY</u> data pair??

- $\Phi(\boldsymbol{A})=(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$ ➔ **6-D**

P1	1	0	1.4142	0	1	0
P2	1	1.4142	1.4142	1	1	1.4142
P3	1	8.4853	1.4142	36	1	8.4853
P4	1	12.7279	1.4142	81	1	12.7279

- $\boldsymbol{\Phi(A) \bullet \Phi(B) = (1 + A^TB)^2 = (1 + \text{dot}(A, B))^2}$
  - **Gram Matrix**

	P1	P2	P3	P4
P1	4	4	4	4
P2	4	9	64	121
P3	4	64	1444	3136
P4	4	121	3136	6889

Dot Products Between **EVERY** Pair (although in low-D)?

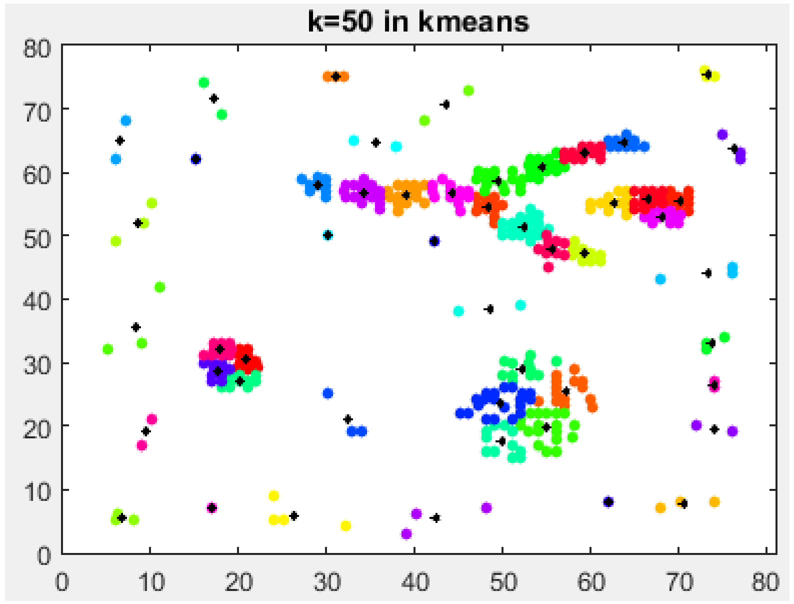
$$L_D = \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_j) - \sum_{i=1}^{m} \alpha_i$$

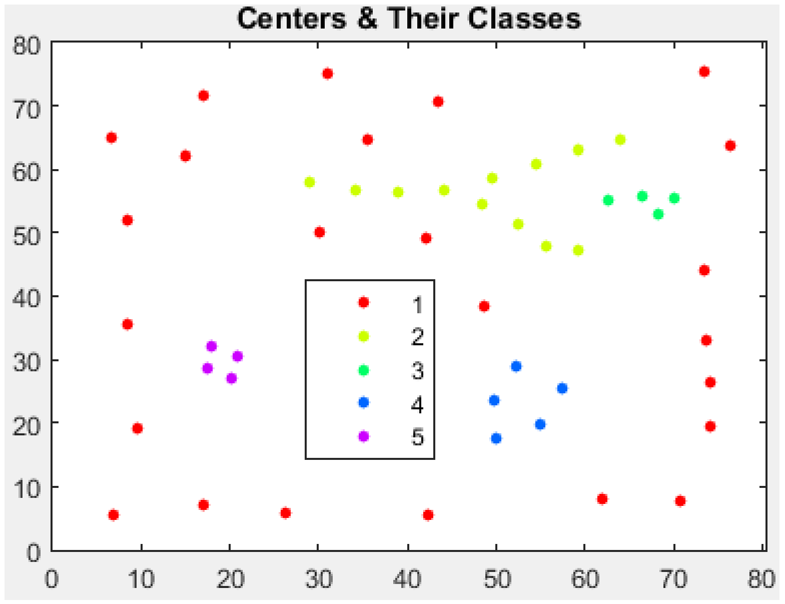
- In other words, when you call SVM with “rbf” or “poly” kernel…
  - Matlab syntax… mdl = fitcsvm(X, Y, '**KernelFunction**', '**rbf**');
  - A ***Gram Matrix*** ***G*** between **every** pair of points in original space is created.
    - ***G*** is the matrix of inner products of **all pairs** of vectors, i.e. $g_{ij} = \boldsymbol{v}_i^T \boldsymbol{v}_j$.
  - **What if** you have BIG data w/ huge records, i.e. 100-million. So, **$(100\text{-million})^2$ = ?**
- Compute Gram Matrix ***G*** that has similarity / kernel btwn **every** pair of points?
  - Pass this ***G*** to a **linear** SVM and build a **linear** model, **conceptually** in **∞-D**.
- How about build a much smaller Gram Matrix on only ***landmark points***. **??!!**

Generating Micro-Clusters First

- Using faster/cheaper (??) algorithm to generate micro clusters first.
- Then compute G-Mat = Build L-SVM on the G-Mat = kernel trick
- Elapsed time of 50 points is 0.522405 seconds.
- Elapsed time of 378 points is 0.740561 seconds.
- 50 / 378 = 0.13% data, 0.522405 / 0.740561 = 71% execution time.

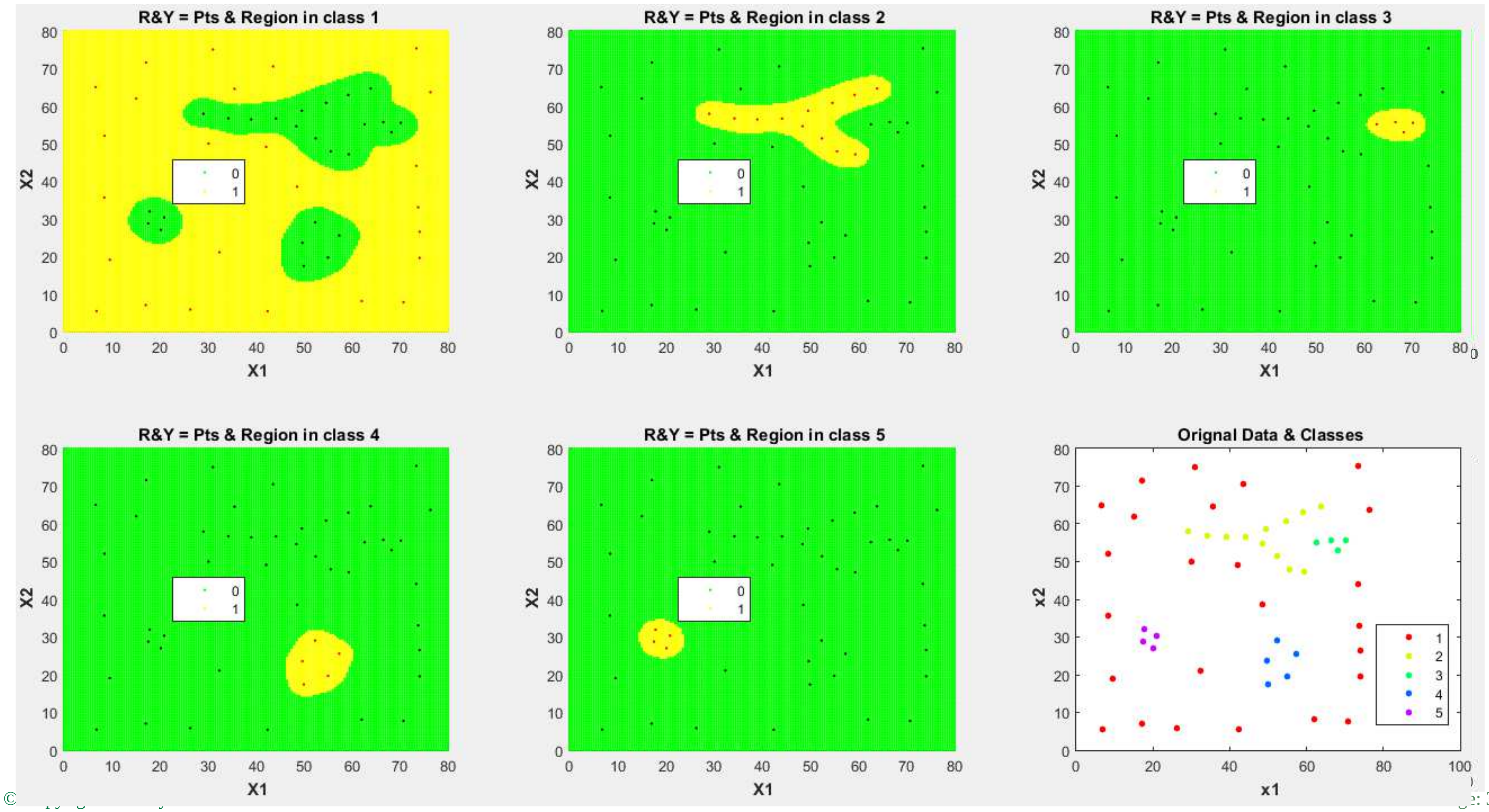
speedup = 1.42





Y-Shape Data with **HUGE** Gram Matrix, **Solution???**

1. Compute an RBF G-Matrix for all pairs between landmark points ***X***.
2. Build a linear SVM ***S*** on the G-Matrix.
3. Convert each new test point ***X***’ to ***X’’*** by computing a kernel between ***X’*** and ***X***.
4. Predict the class of ***X’’*** using linear SVM ***S***.

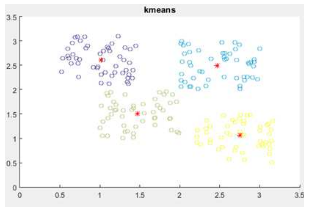


Summary of **Kernel Tricks**

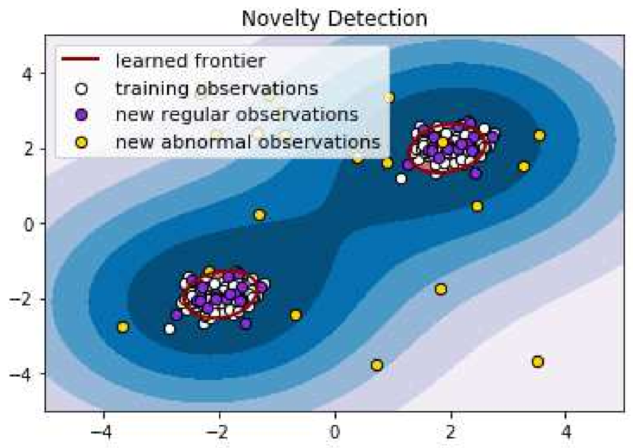
- You can have $\Phi(x^{(i)})$ in **VERY** high dimension.
  - The dot product in that **VERY** H-D can be done by a ***kernel function*** in the original dimension.
  - Build a Gram matrix that contains similarity (kernel) between pair-wise data points in the original space.
  - The (conceptual) dimension of $\Phi(x^{(i)})$ can be >> **(much larger than)** # of data points.
  - Have SVM learns a linear separation from the Gram matrix.
- So now you can **substantially** increases the # of features for your classifier.
- Kernel can be thought as an "**instance-based**" method.
  - Remember data (or ***landmarks***) rather than remember parameters θ (or $\boldsymbol{w}$).

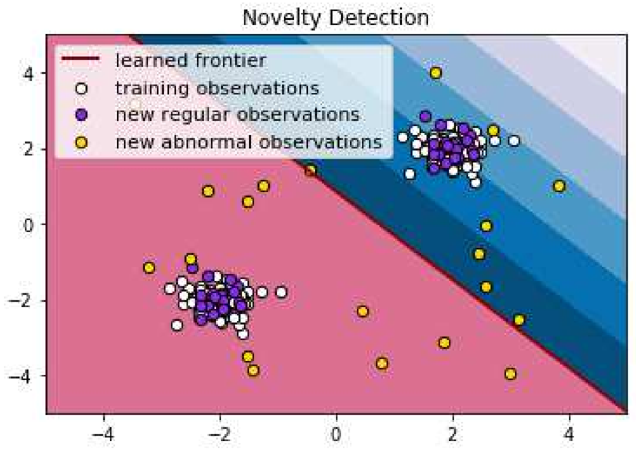
$\boldsymbol{wX} + \boldsymbol{b} = 0 \Rightarrow (\sum_{i=1}^{m} \alpha_i y_i x_i \mathrm{X}) + \boldsymbol{b} = 0$

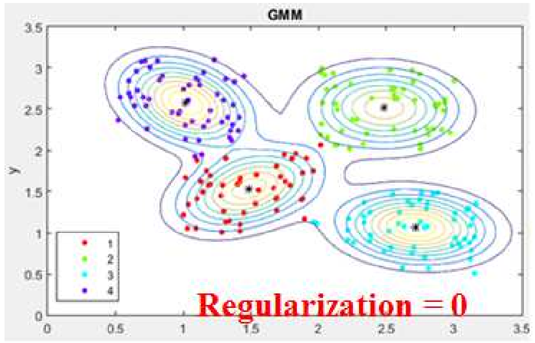
SVM One Class Classification

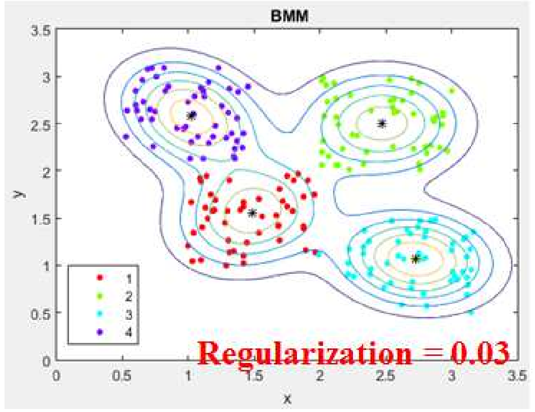


- **Matlab fitcsvm( )** **or** **Sklearn svm.OneClassSVM( )**
  - An unsupervised method to learn a decision function for novelty detection.
  - Classify new data as how similar or how different to the training set. ***k*NN**??
  - **Like clustering???** Compare either to *k*-means or ***GMM*** (Gaussian Mixture Model).
  - http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html#sklearn.svm.OneClassSVM
  - http://scikit-learn.org/stable/auto_examples/covariance/plot_outlier_detection.html#sphx-glr-auto-examples-covariance-plot-outlier-detection-py
  - http://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html





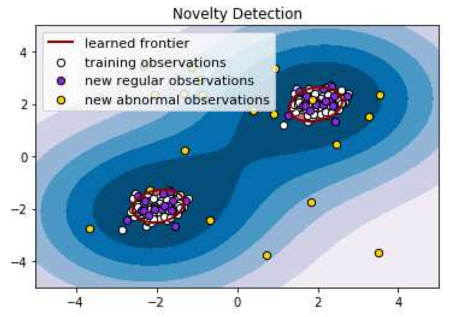




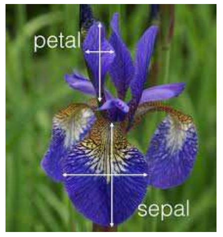
```
# kernel must be one of 'linear', 'poly', 'rbf', 'sigmoid'
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf.fit(X_train);        y_pred_test = clf.predict(X_test)
```

One-Class SVM Experiments

- sklearn dataset
  - Build **only** **1** **one**-class SVM from …
  - Data of (**one** class (majority) + outliers (minority).

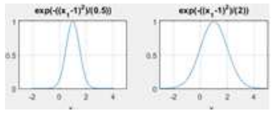


- Iris dataset
  - Build **2** **one**-class SVMs from …
  - Unbalanced data of (**two** classes).
  - Compare to **1** **two**-class SVM.

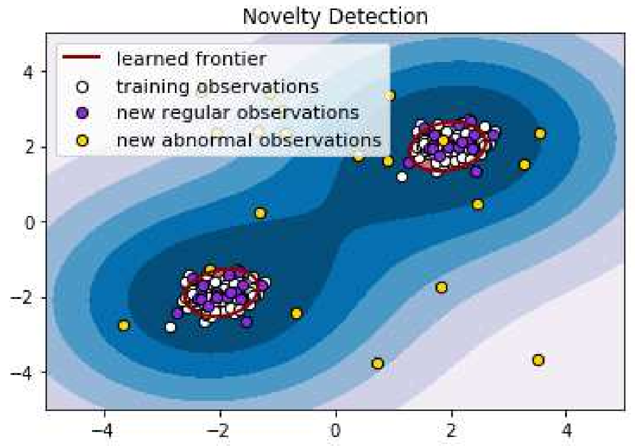


- Ovarian cancer dataset
  - Build **2** one-class SVMs from …
  - **Super** unbalanced data of (**two** classes).
  - **4,000**-dimension data.

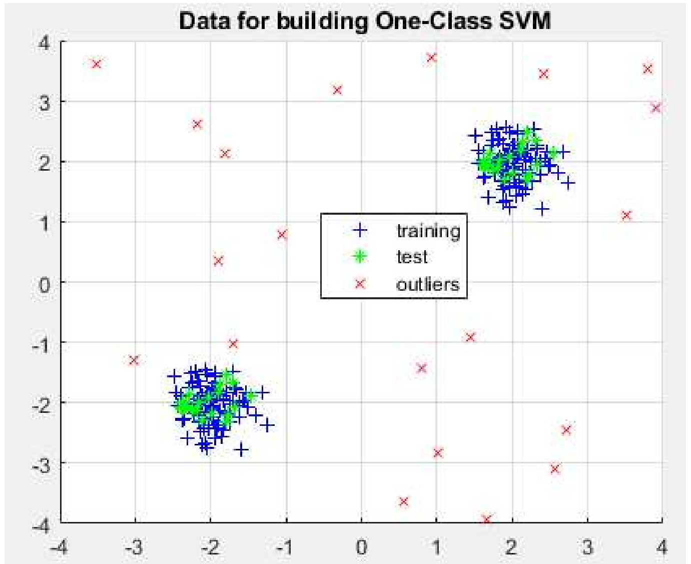
SVM RBF One Class, NO Kernel Scale

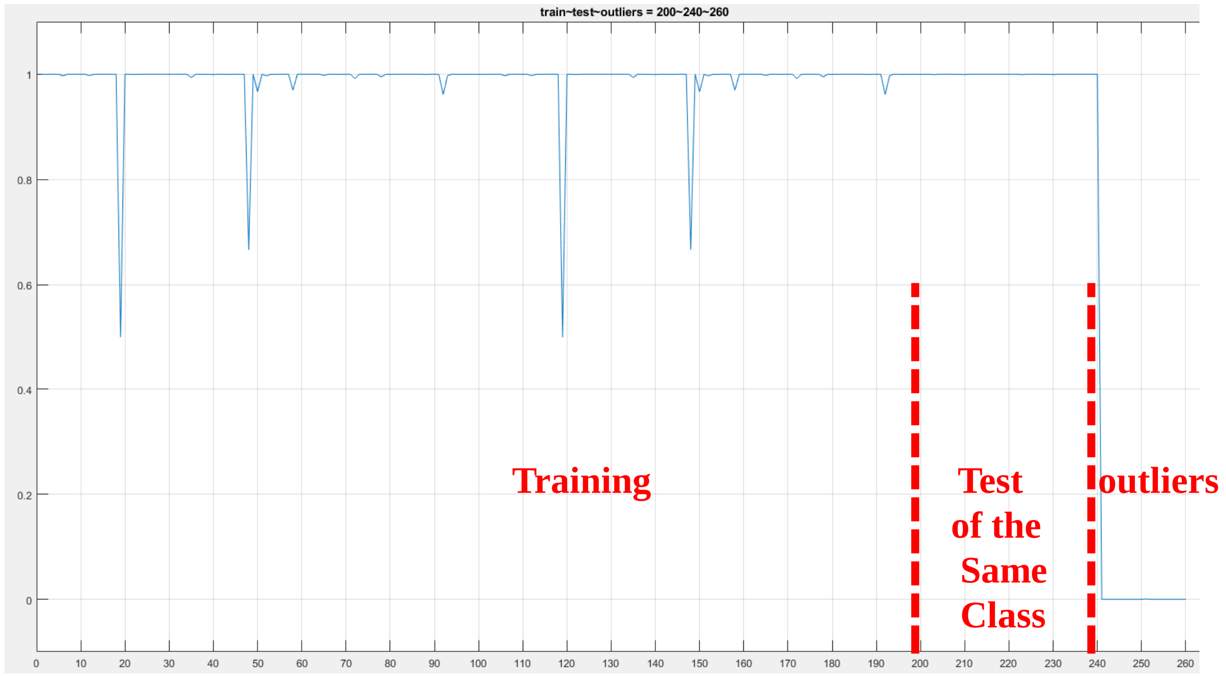


- http://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html#sphx-glr-auto-examples-svm-plot-oneclass-py

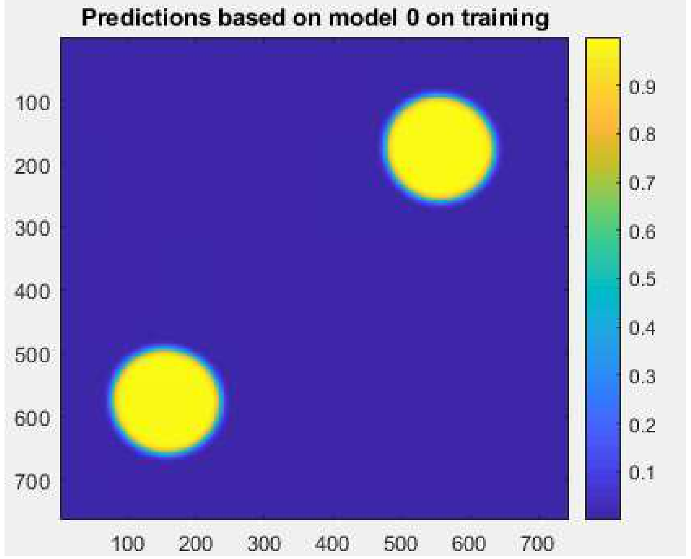


```
# data
rng(10)

XX = 0.3 * randn(100, 2);
X = [XX + 2; XX - 2];
Y = zeros(size(X, 1), 1);

XX = 0.3 * randn(20, 2);
X_test = [XX + 2; XX - 2];

% Generate outliers
a = -4;    b = 4;
X_outliers = (b-a).*rand(20,2) + a;
```



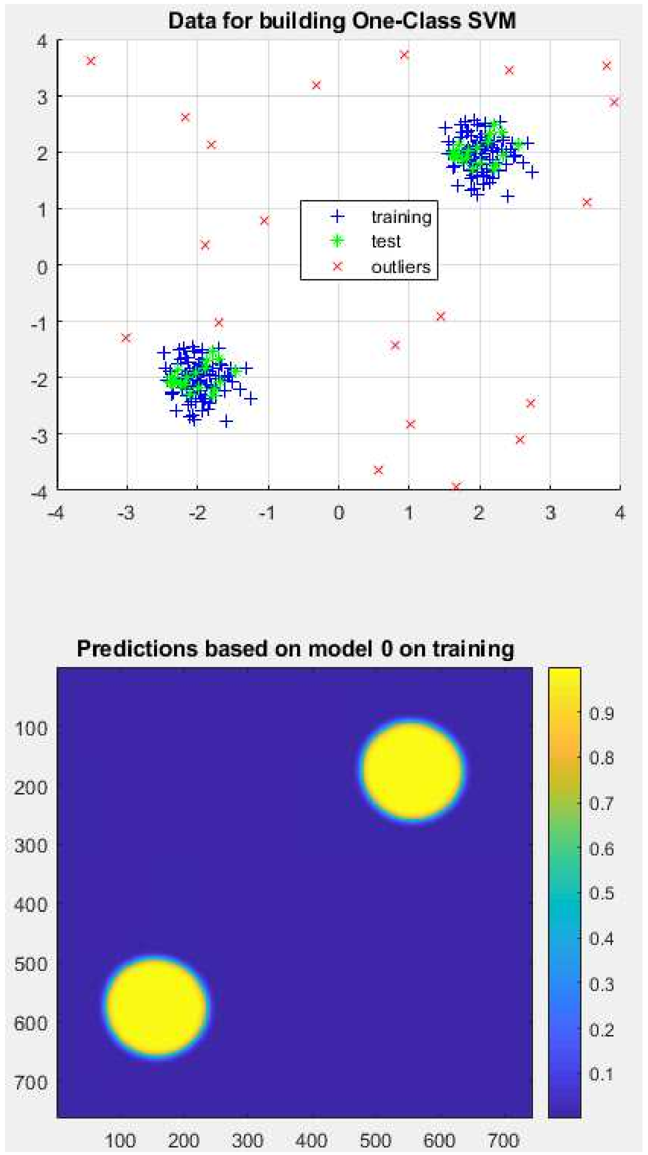


Probability

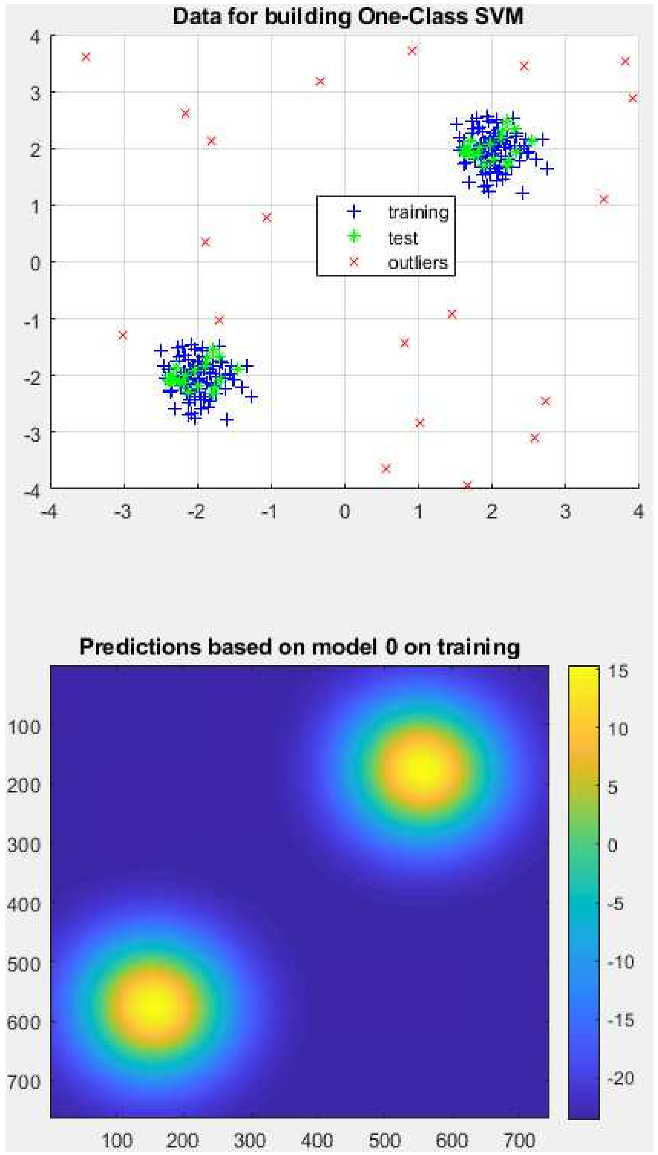


SVM RBF One Class, **NO** Kernel Scale

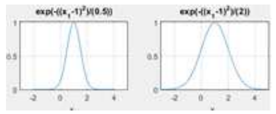
Probability



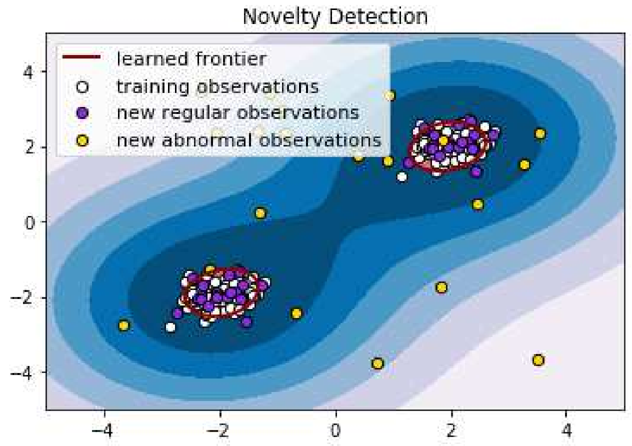
W*^{T}*X

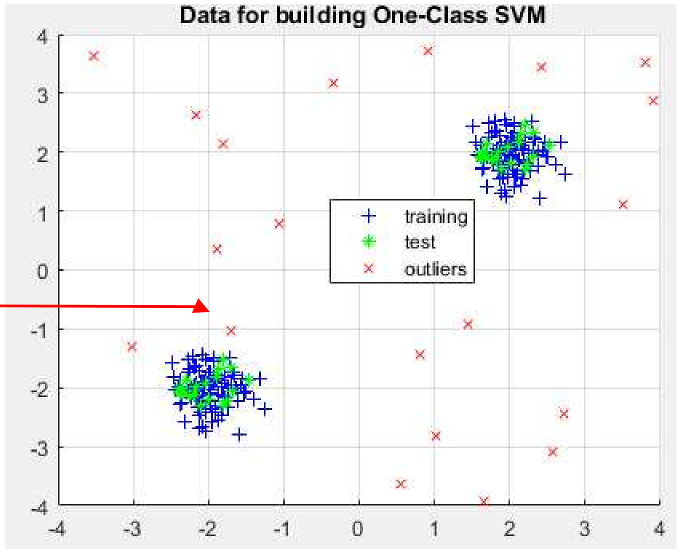


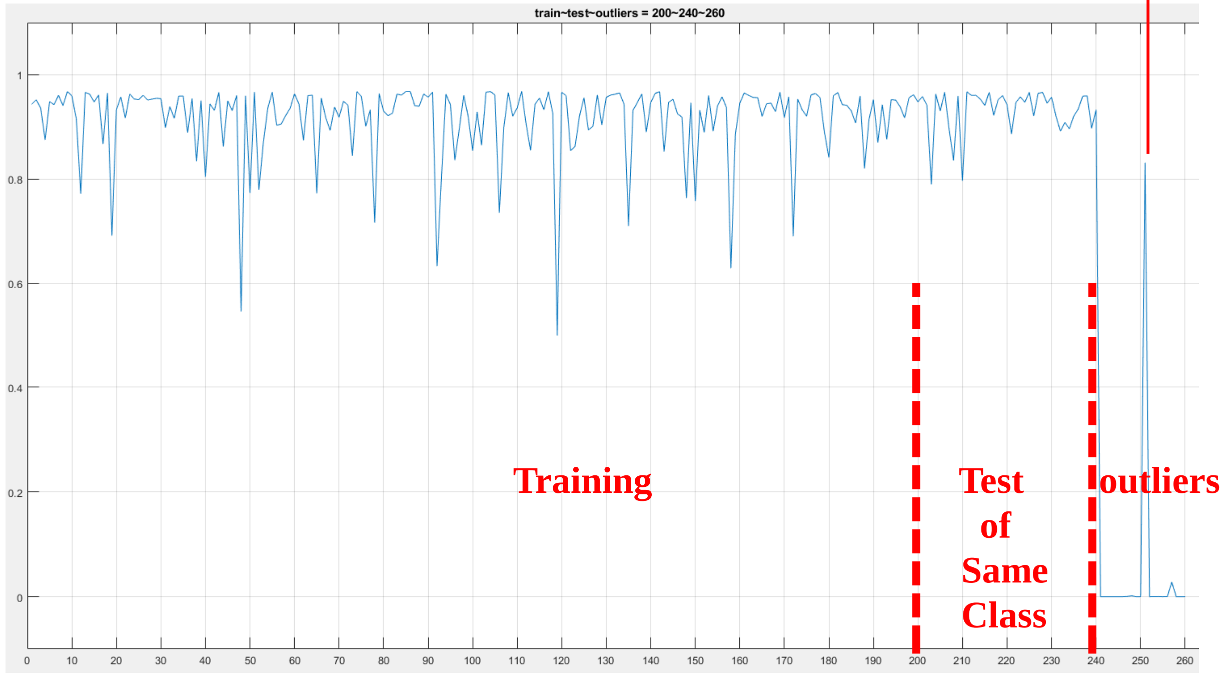
SVM RBF One Class, **Kernel Scale = 3.5**



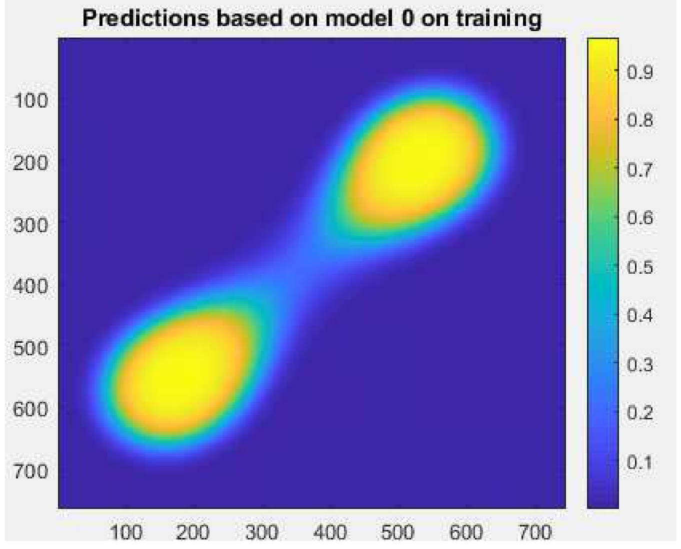
- http://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html#sphx-glr-auto-examples-svm-plot-oneclass-py



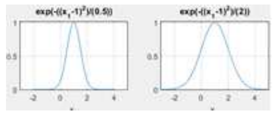




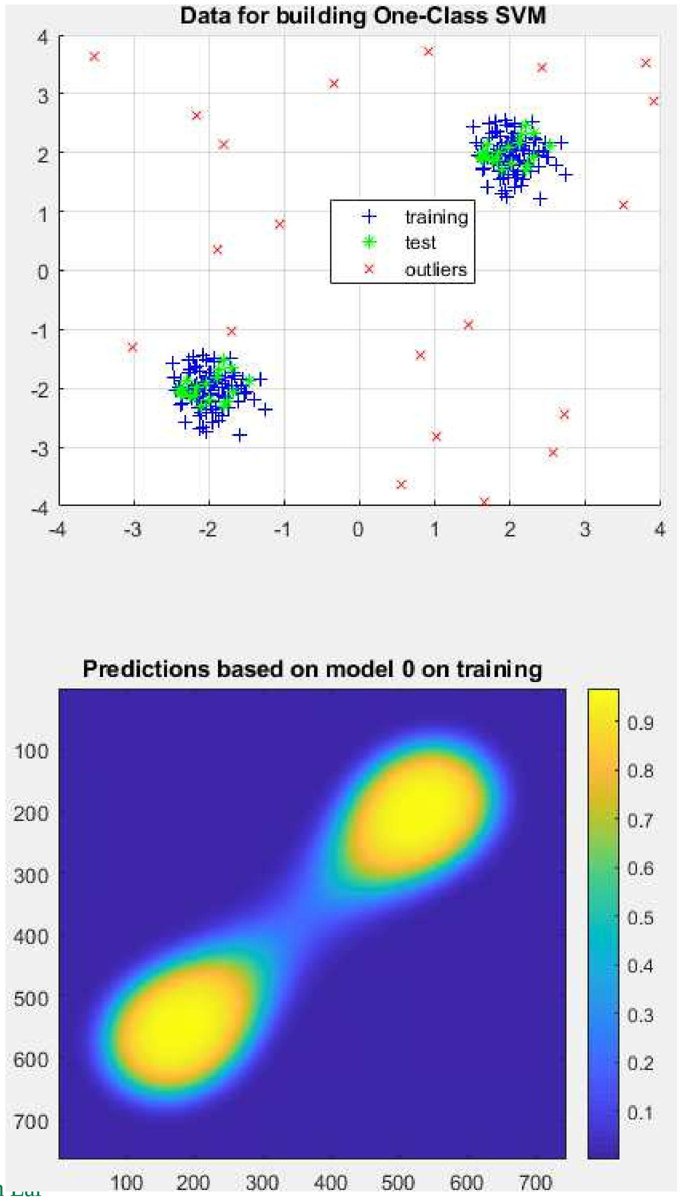
Probability



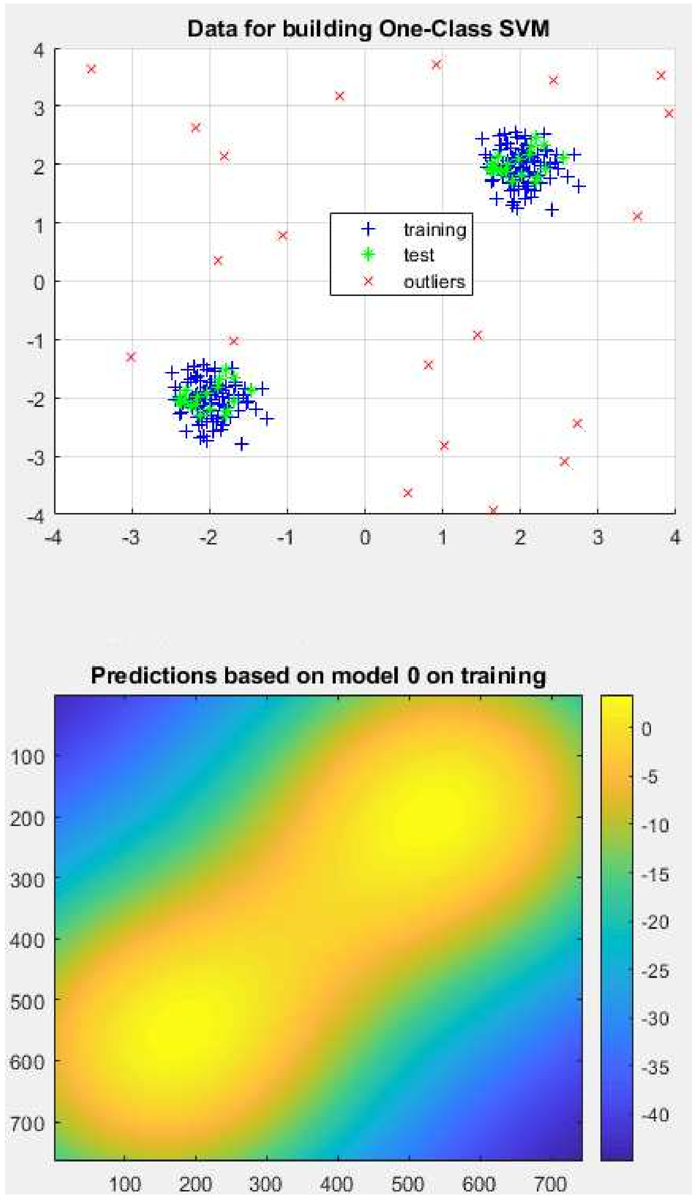
SVM RBF One Class, **Kernel Scale = 3.5**



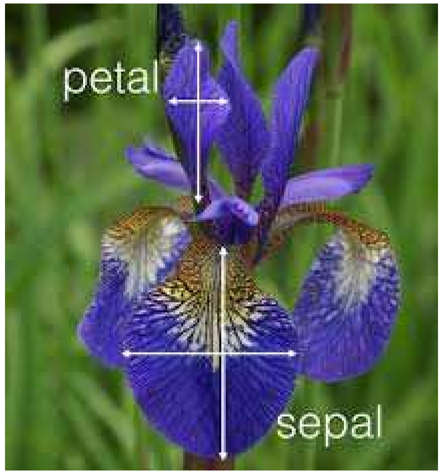
Probability



W^TX

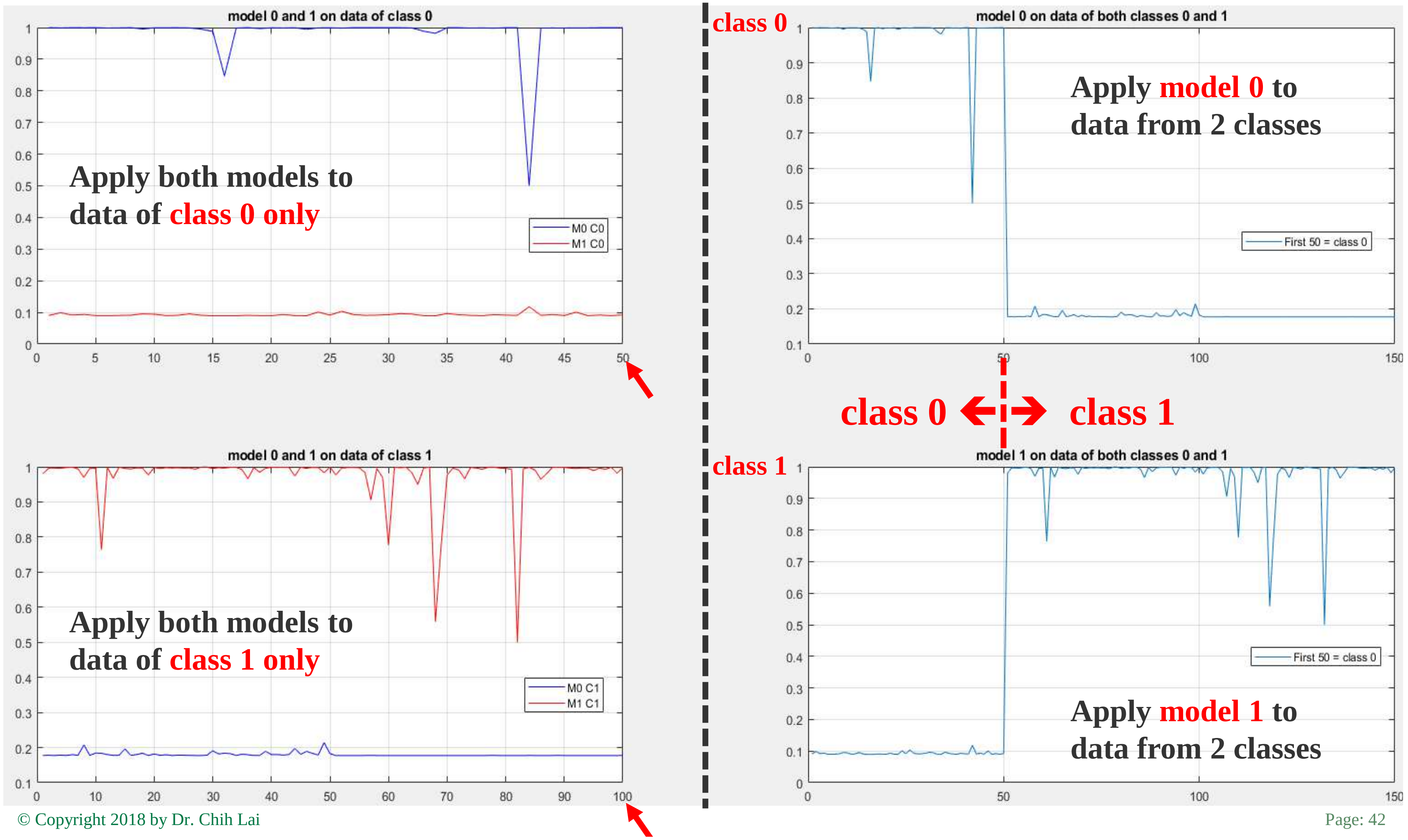


Iris Dataset

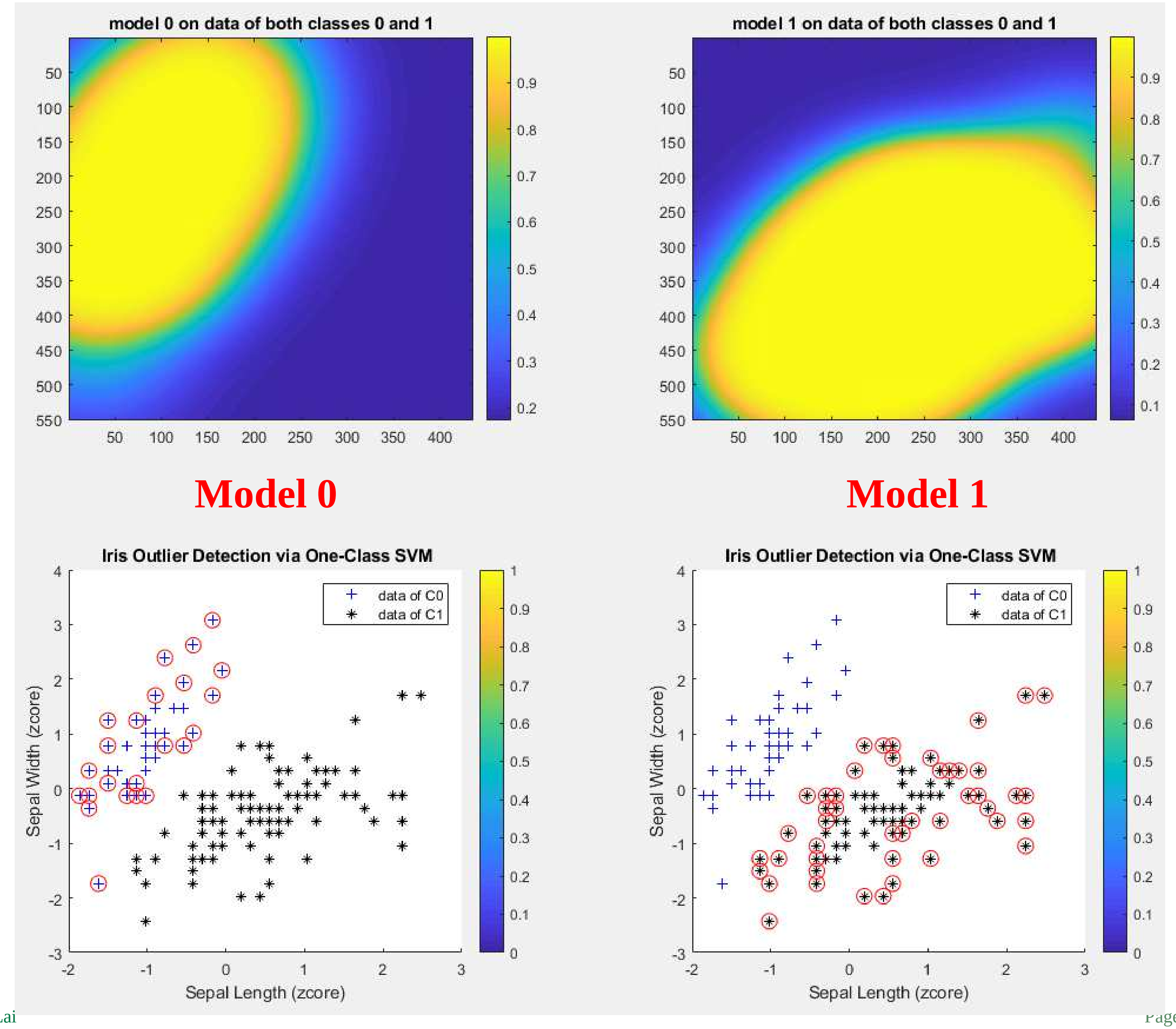


SVM / RBF One Class for **Unbalanced** Iris (50 vs. 100) Dataset

- Build model 0 (M0) from data of class 0 (first 50 rows) only.
- Build model 1 (M1) from data of class 1 (100 rows) only.

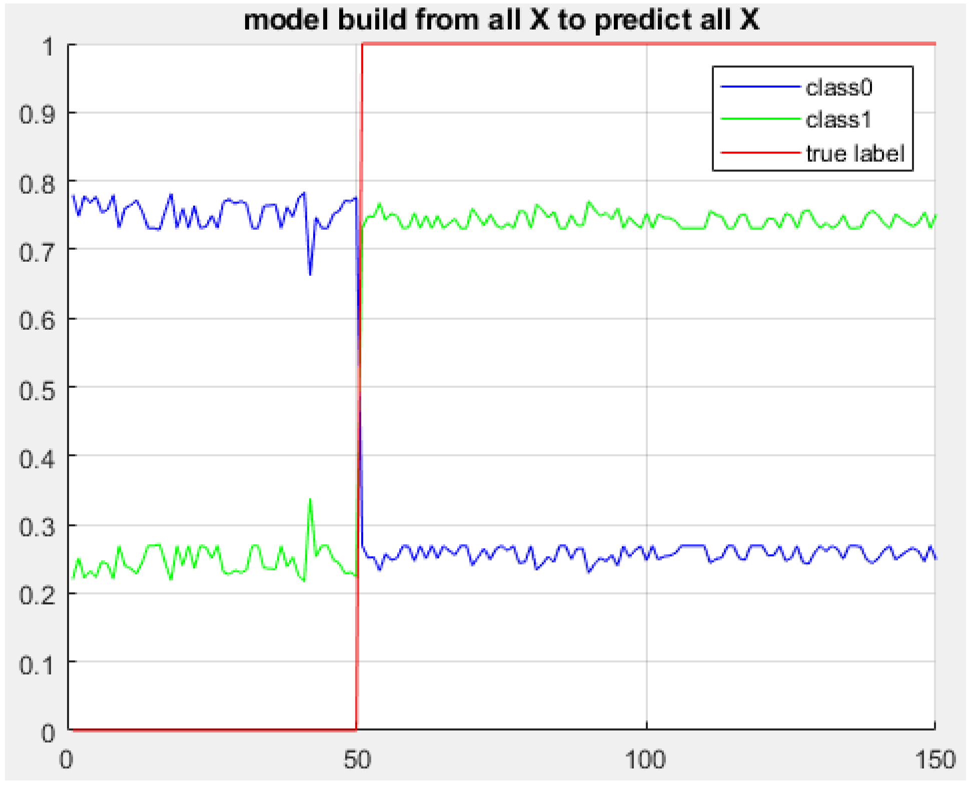


SVM / RBF One Class for **Unbalanced** Iris (50 vs. 100) Dataset– Cont’d



SVM / RBF for **Unbalanced** Iris (50 vs. 100) Data– One 2-Class SVM

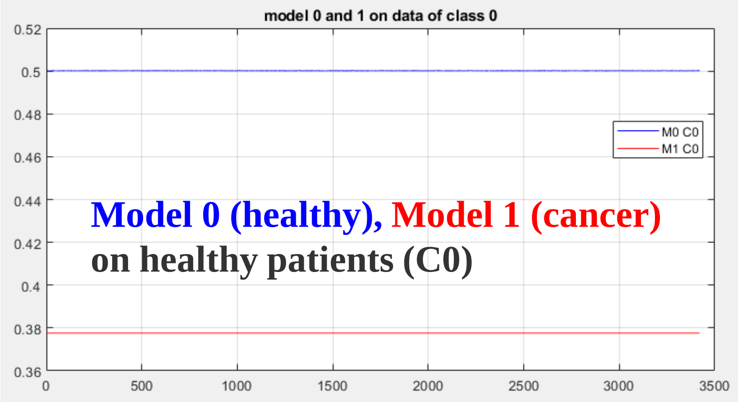
- Build **ONE** 2-Class model from both classes.

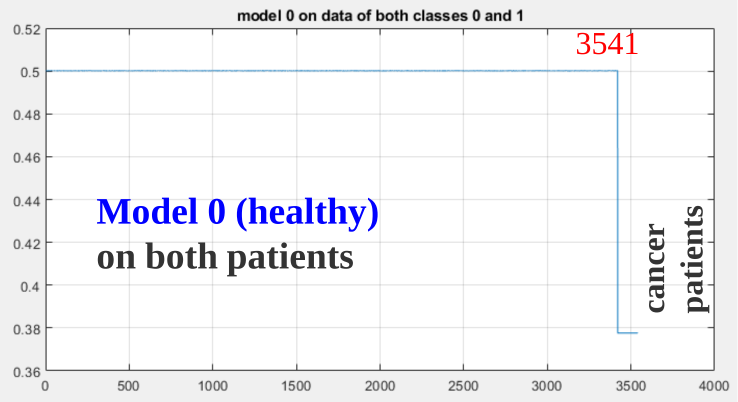


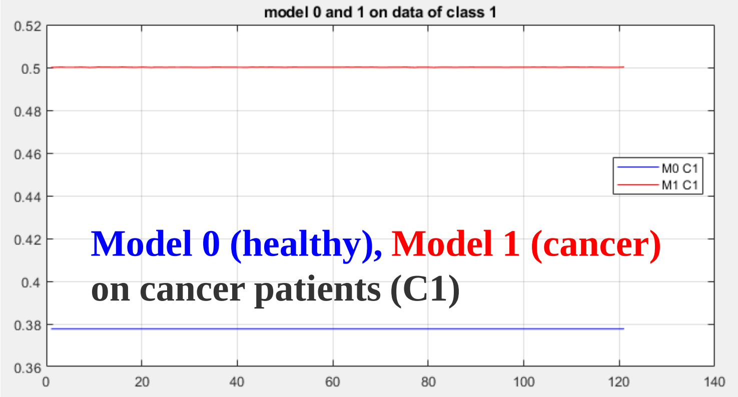
Ovarian Cancer Dataset

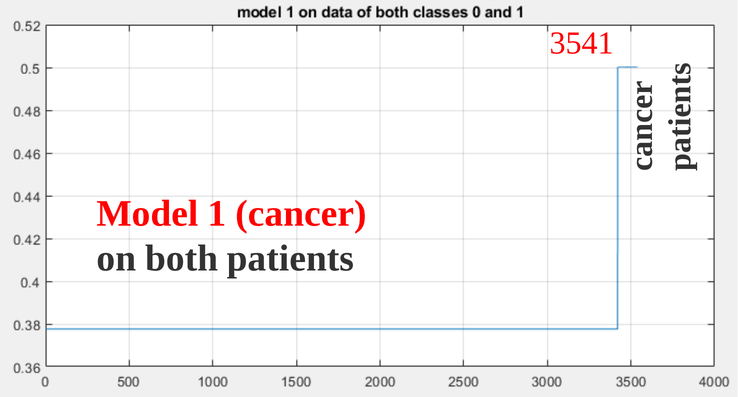
Detecting Super Unbalanced Ovarian Cancer Patients (One-Class)

- M0 was trained on healthy patients (C0).
- M1 was trained on ovarian cancer patients (C1).
- Duplicate healthy records couple times to create a skewed dataset
  - Original data, total 261, Cancer = 121 (46%). Duplicate data, total 3662, Cancer = 121 (3.4%)



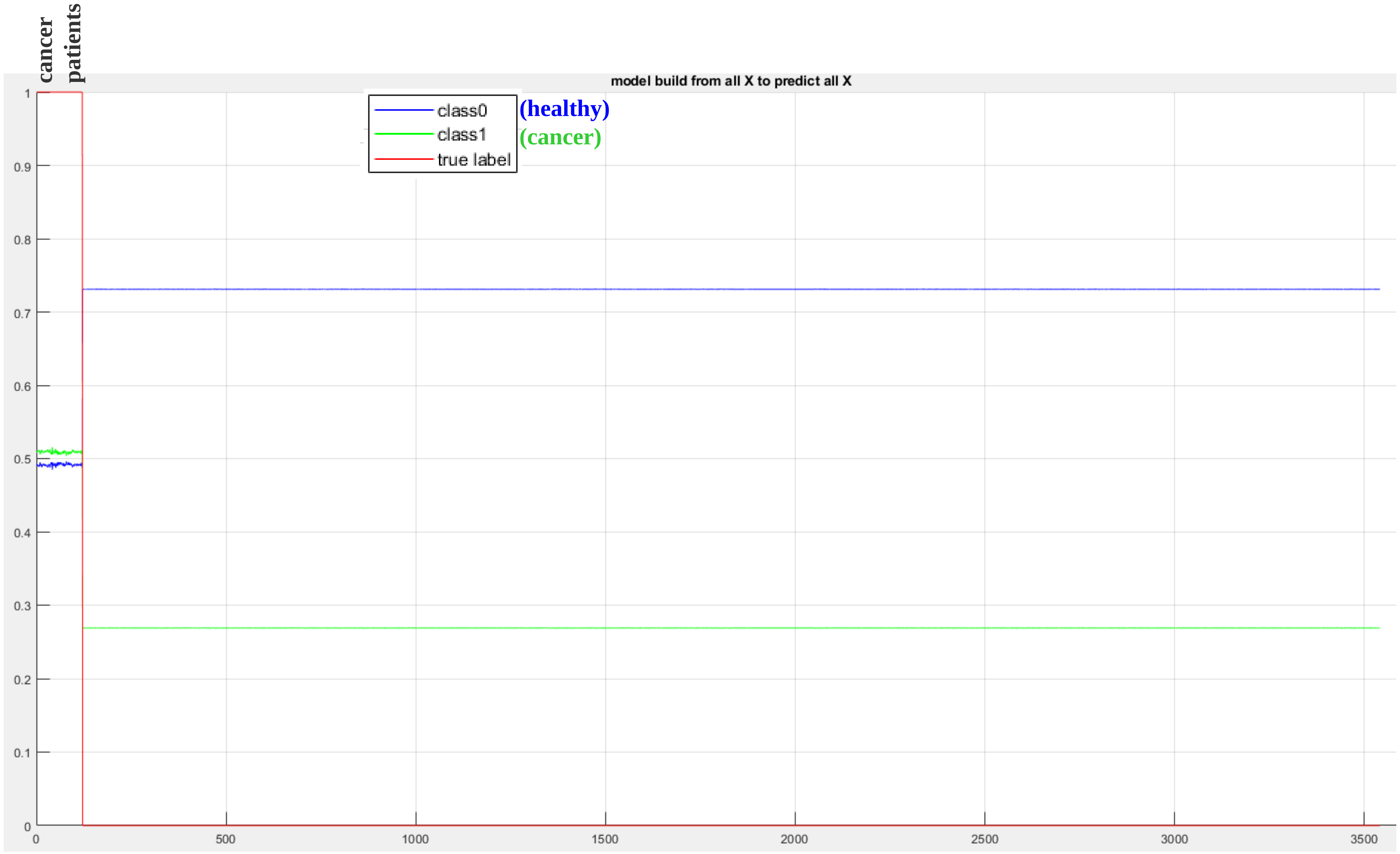






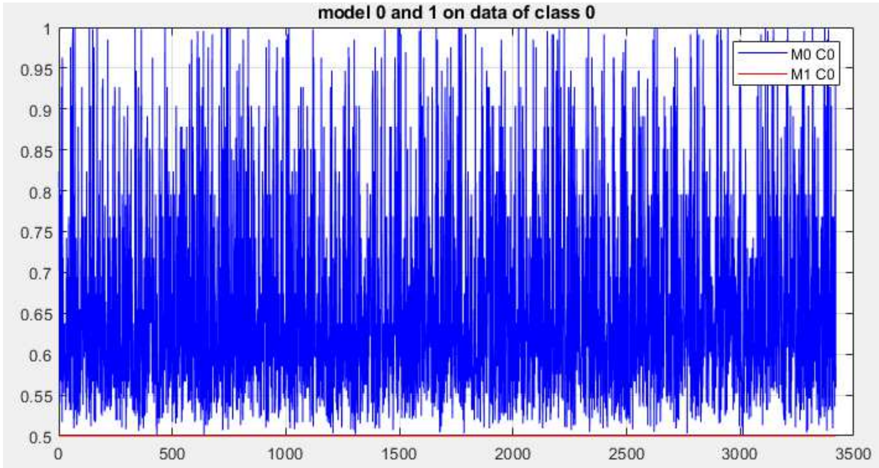
Build One 2-Class SVM

- Build **one** **2-class** SVM model from **ALL** (unbalanced) patients.

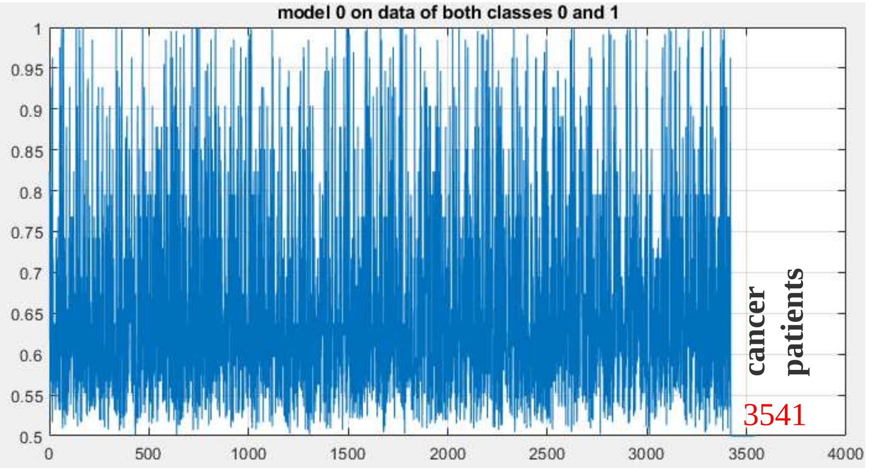


Super Unbalanced Ovarian Cancer Patients (1-Class), **Kernel Scale = 0.1**

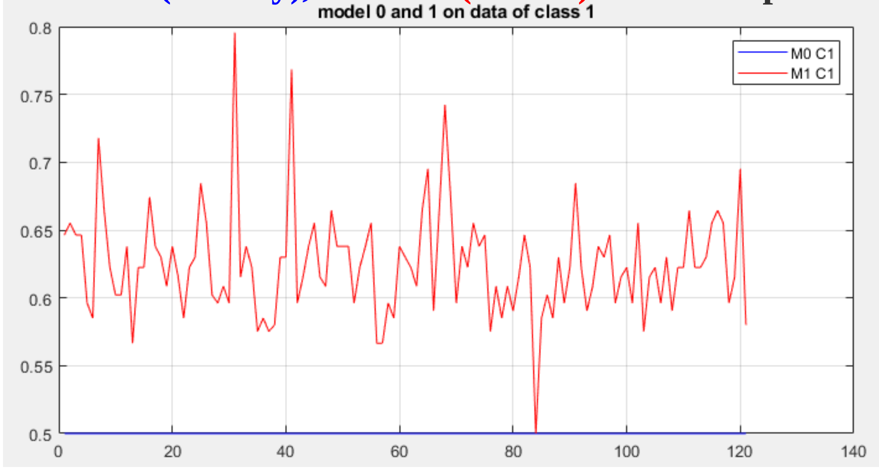
Model 0 (healthy), Model 1 (cancer) on healthy patients



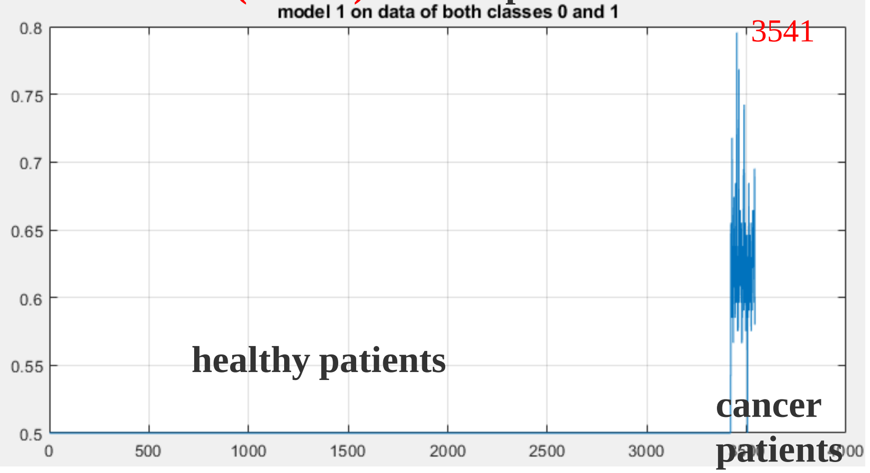
Model 0 (healthy) on both patients



Model 0 (healthy), Model 1 (cancer) on cancer patients



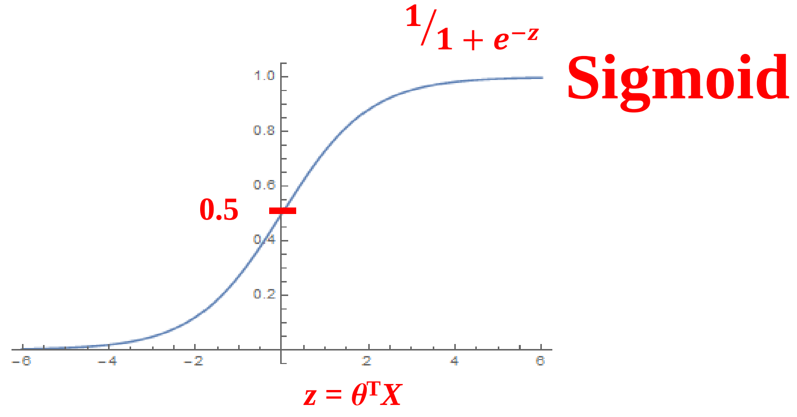
Model 1 (cancer) on both patients



Apply One-Class Classification Using Logit??

- Objective Function for Logistic Regression
  - **minimize** ***negative log likelihood***

$$\frac{-1}{m}\sum_{i=1}^{m}[Y_i log(P_i) + (1 - Y_i) log(1 - P_i)]$$

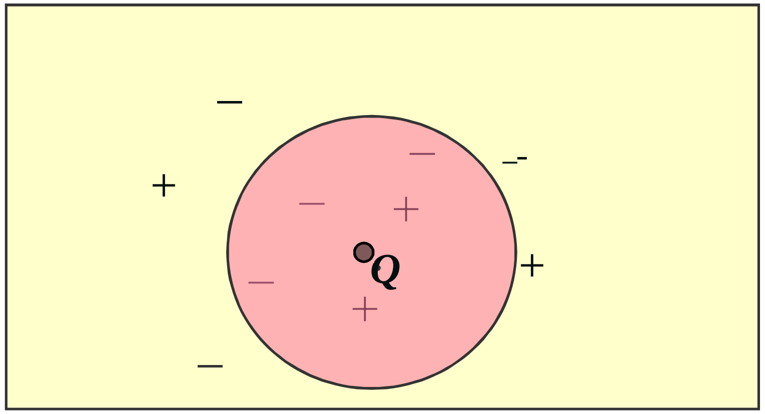


- Object function for SVM Kernel
  - Min $L_D = \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i x_j} - \sum_{i=1}^{m} \alpha_i$ **(Dual form)**
  - Min $L_D = \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x_i})\Phi(\boldsymbol{x_j}) - \sum_{i=1}^{m} \alpha_i$

Instance-Based Methods

- **Eager** instance-based learning, like SVM-RBF.
- **Lazy** instance-based learning. Store past data, **NO** model construction.
  - Approach 1– ***k*-nearest neighbor (*k*NN)**
    - All instances (records) are represented as points in the *n*-D Euclidean space.
    - Assign the majority class of the nearest neighbors to the new (unseen) data.
    - For each query, finding *k*NN can be very time consuming.
    - *k*-nearest neighbors can be far away (very dissimilar) from *Q*.
  - Approach 2– *range query*

+ *k*=11

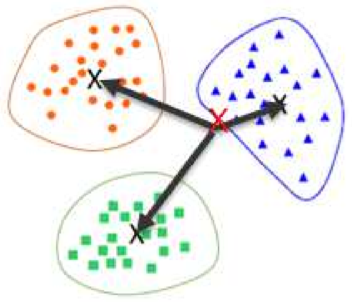


Outlook	Temp.	Humid	Windy	Play
S	W	H	F	N
S	W	H	T	N
O	W	H	F	Y
R	M	H	F	Y
R	C	L	F	Y
R	C	L	T	N
O	C	L	T	Y
S	M	H	F	N
S	C	L	F	Y
R	M	L	F	Y
S	M	L	T	Y
O	M	H	T	Y
O	W	L	F	Y
R	M	H	T	N

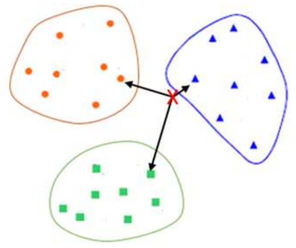
*k*NN for Digit Recognition

- Compute distance between each test instance against **ALL** training data
  - Predict query image based on majority of ***k***NN digits.   Slow to run.

- Improvements?
  - **Idea 1**: classifying based on distance to the **center** of each class.
    - **What is the center of each class?**

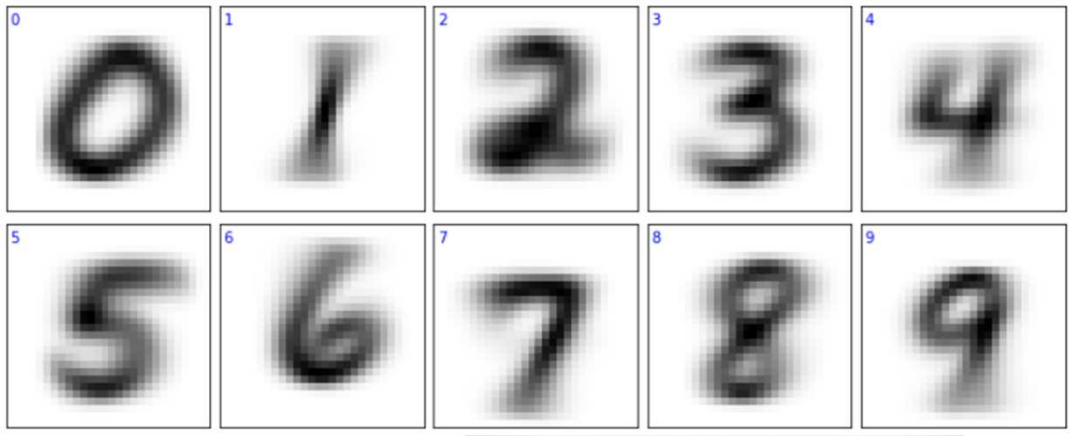


  - **Idea 2**: using smaller samples of the dataset.



- Data Source: https://www.kaggle.com/c/digit-recognizer/data

center of each class



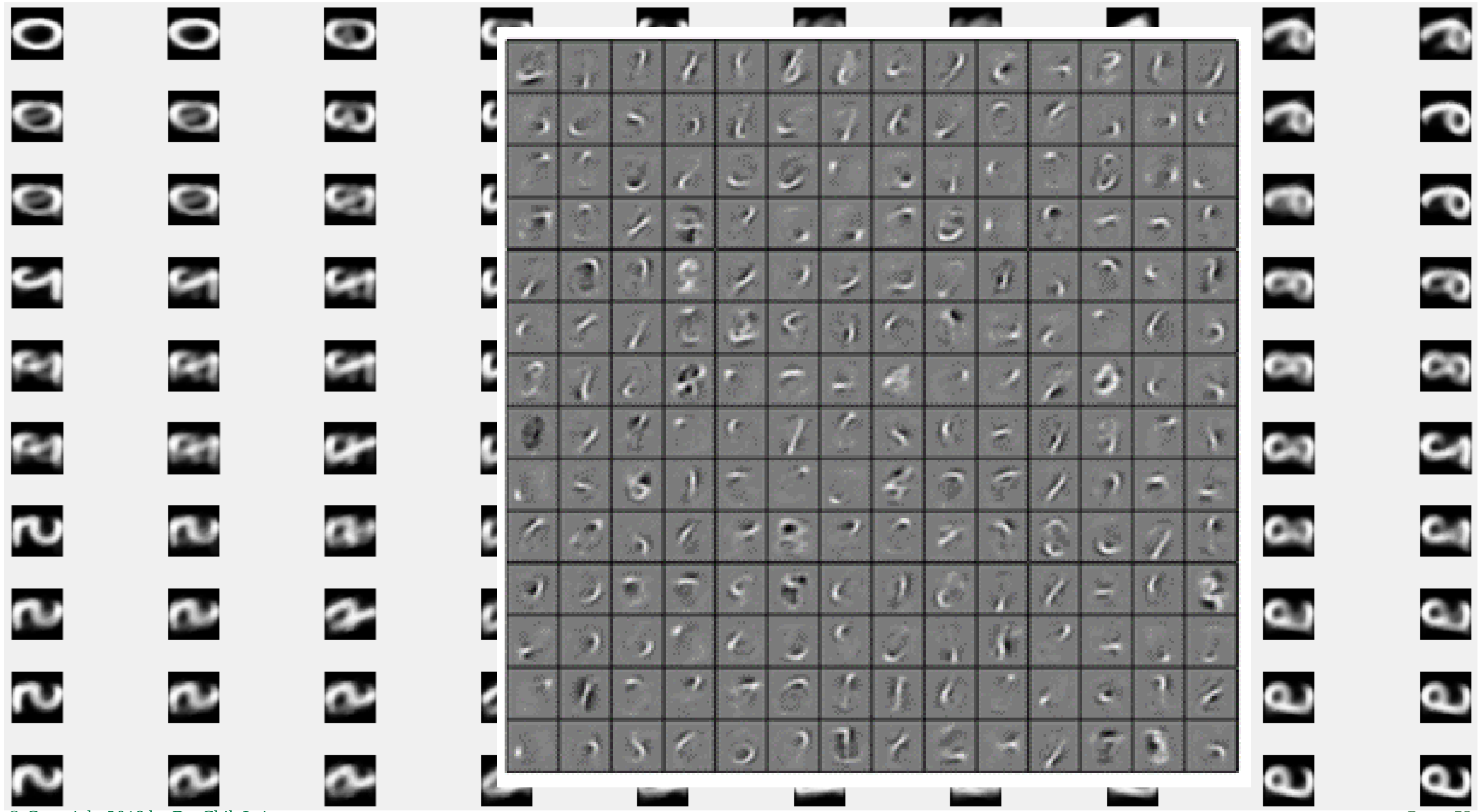
Digit Recognition, DM-02-18S

Alreshidi Abdulaziz, Yogita Singh
Bader Albulayhis, Sidi Mohamed,

Self Organizing Map (SOM)

- Another "better" alternative➔ **Convolutional Neural Network** (CNN).
  - Less recognizable patterns. They are no longer centers. They are ***features***.

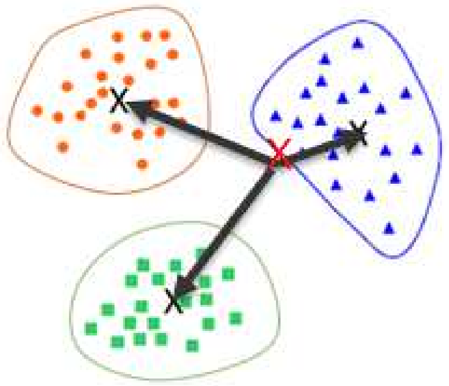
10×10 SOM neurons

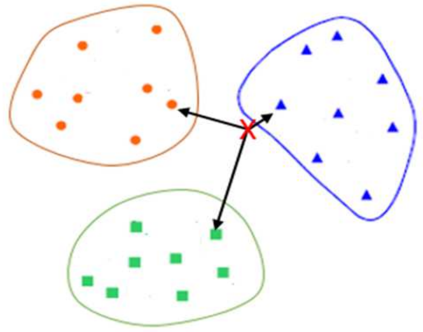


Idea 2: Using Smaller Samples from Training Data

- Much faster with accuracy tradeoff.

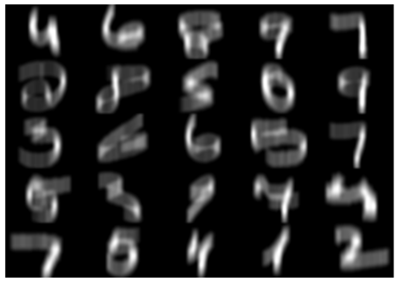
data size used in kNN	Accuracy	Exe Time
400 (1% of the data)	84%	1sec
2000 (5% of the data)	91%	20 sec
5000 (12.5% of the data)	93%	50 sec
10000 (25% of the data)	95%	2min

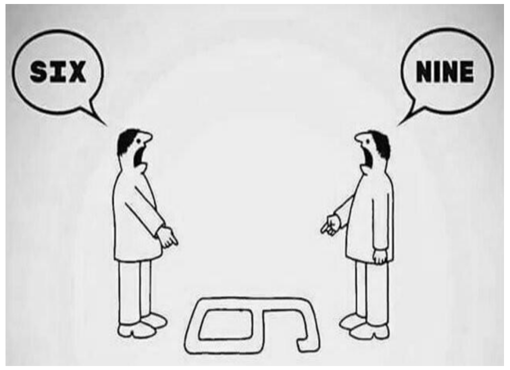




Digit Recognition, DM-02-18S

Alreshidi Abdulaziz, Yogita Singh
Bader Albulayhis, Sidi Mohamed,





Compare kNN Classification Quality to Other Methods

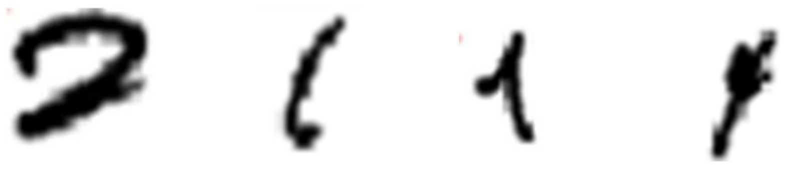
- Computing time is bit longer. But, kNN produce not bad result.
- How about executing time and quality of SVM or SVM+RBF?
- **How about advanced NN (i.e. CNN)???**

Algorithm	Accuracy (%)
Decision Tree	85
Naïve Bayes	82.55
KNN	96.0
Random Forest	96
MLP ← **Regular NN**	94

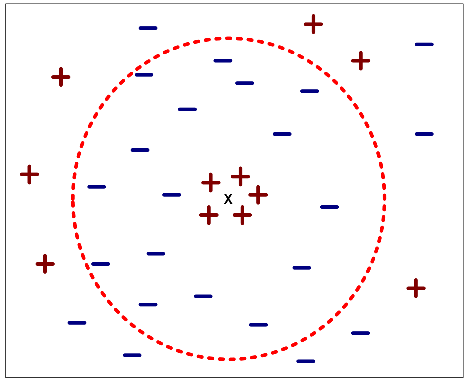
Digit Recognition, DM-02-18S

Alreshidi Abdulaziz, Yogita Singh
Bader Albulayhis, Sidi Mohamed,

- Some writings are difficult to classify…



Issues in ***k***NN, or Instance-Based Learning



- Difficult choosing right ***k*** value
  - If ***k*** is too small, sensitive to noise points
  - If ***k*** is too large, neighborhood may include points from other classes

- Numeric attributes with different scales.
  - Distance measures may be dominated by one of the attributes
    - Heights of persons vary from 1.5m to 1.8m. $$ of persons vary from $100K to $100B.

- Binary attributes.

```
R1 =  (0   0   0)
R2 = ( 1   1   0)
R3 = ( 1   0   1)
```

- Categorical attributes. (e.g. diseases, states…)
  - Convert them to dummy variables…
  - **That's it??!!**
  - Before RBF, we **never** compare distance btwn records.
    - We only derive θ to compute $\theta^T X$.

```
1 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1
```

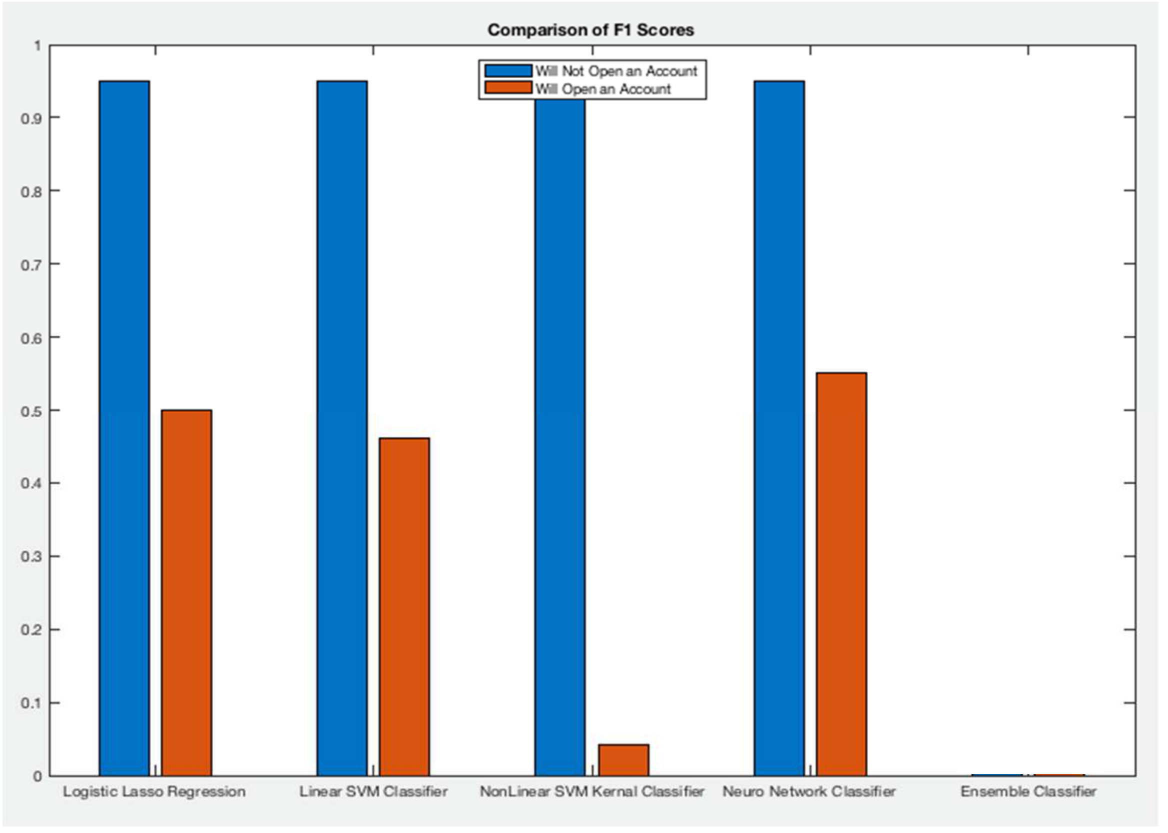
Lazy vs. Eager Learning

- Lazy evaluation
  - kNN or Naïve Bayes (**instance-based learner**).
  - **Less time training but more time predicting** **need to carry all instances**
  - Generalize **beyond** the **current** training data.

- Eager evaluation
  - Decision-tree, logistic, LDA, SVM, **SVM RBF** (**instance-based learner**)
  - **More time in training but less time in predicting**
  - **Commit** to a fixed / static model.

Reminder

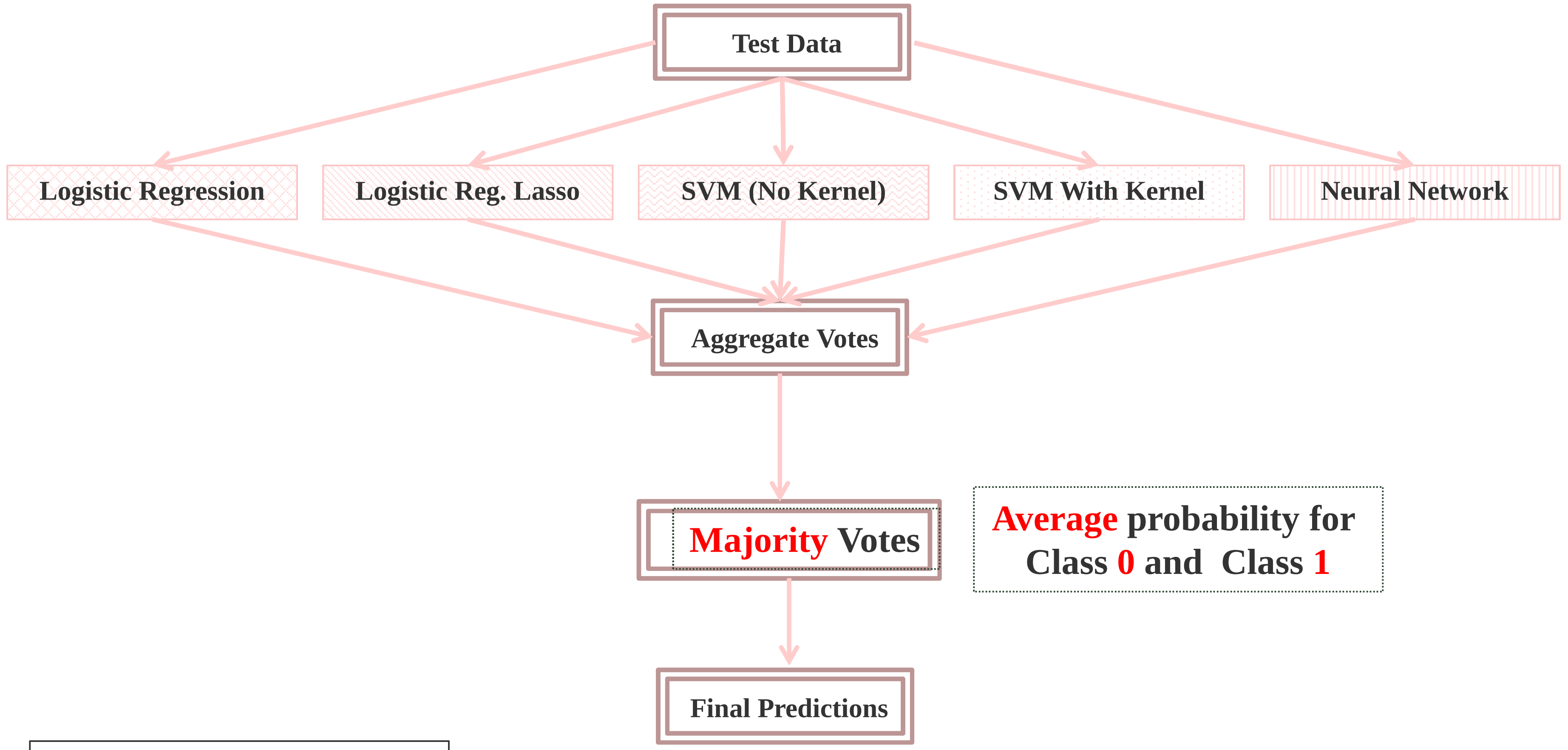
- We discussed many machine learning techniques.
- Try to use all or multiple methods.
- Not only you can compare their performance, but also… WHAT???



MDL-01 2018S, Bank Marketing

Terrence White
Ronald E Twite
Leela Sowjanya Chippada
Ahmad K Lubnani
Mowlid Abdillahi
Nathan Adams

Ensemble Hard / Soft Voting

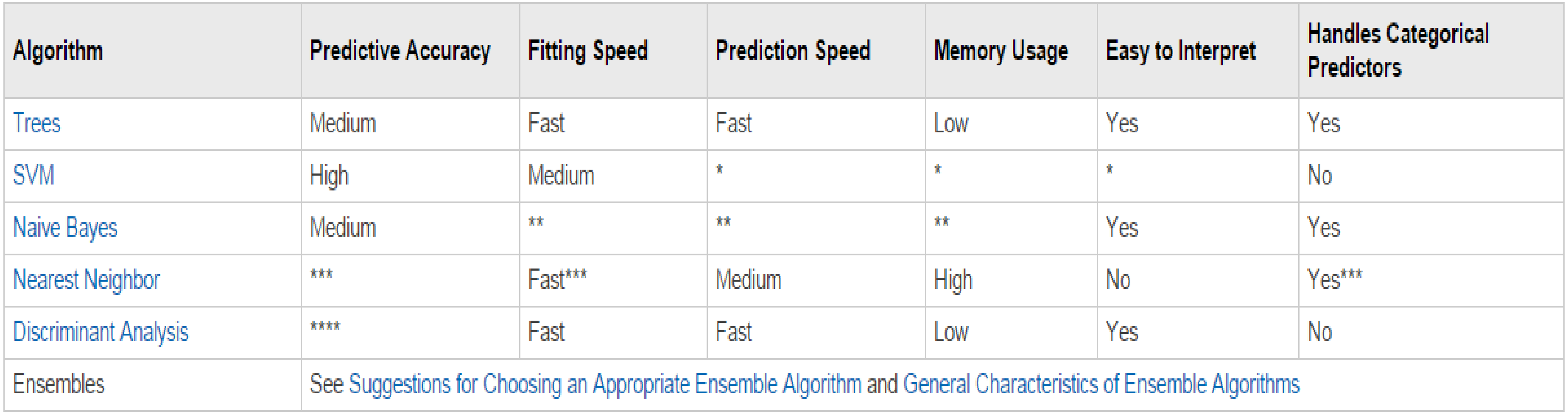


MDL-01 2018S, Bank Marketing

Terrence White
Ronald E Twite
Leela Sowjanya Chippada
Ahmad K Lubnani
Mowlid Abdillahi
Nathan Adams

Characteristics of Classification Algorithms

- SVM
  - Speed & memory usage are good w/ few support vectors, poor if too many. Difficult to interpret how SVM classifies data w/ kernels. Easy for linear SVM.
- Naive Bayes
  - Speed & memory usage are good for simple distributions, but poor for kernel distributions and large data sets.
- Nearest Neighbor
  - Good predictions in low D, but poor predictions in high D. Need kd-trees for speed. Vars can be either continuous or categorical, not both.
- Discriminant Analysis
  - Accurate when **normal dist**. Otherwise, accuracy varies.



Algorithm	Predictive Accuracy	Fitting Speed	Prediction Speed	Memory Usage	Easy to Interpret	Handles Categorical Predictors
Trees	Medium	Fast	Fast	Low	Yes	Yes
SVM	High	Medium	*	*	*	No
Naive Bayes	Medium	**	**	**	Yes	Yes
Nearest Neighbor	***	Fast***	Medium	High	No	Yes***
Discriminant Analysis	****	Fast	Fast	Low	Yes	No
Ensembles	See Suggestions for Choosing an Appropriate Ensemble Algorithm and General Characteristics of Ensemble Algorithms					

Appendix

Why RBF $\in \infty$-Space?

- $\boldsymbol{K}(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\|\boldsymbol{a} - \boldsymbol{b}\|^2) / 2\sigma^2 = e^{\frac{-||a-b||^2}{2\sigma^2}}$.

- $K(x, y) = \exp(-\|x - y\|^2) = \exp(-(x_1 - y_1)^2 - (x_2 - y_2)^2)$

  $= \exp(-x_1^2 + 2x_1y_1 - y_1^2 - x_2^2 + 2x_2y_2 - y_2^2)$

  $= \exp(-\|x\|^2) \times \exp(-\|y\|^2) \times \mathbf{exp(2x^Ty)}$

$$k(x,y) = \exp(-\|x\|^2)\exp(-\|y\|^2)\boxed{\sum_{n=0}^{\infty}\frac{(2x^Ty)^n}{n!}}$$

- **Tylor series for *e*.**
  - Raise ***x*** & ***y*** to ***n***-dimension, divide it by ***n***-factorial, and sum to infinity.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$