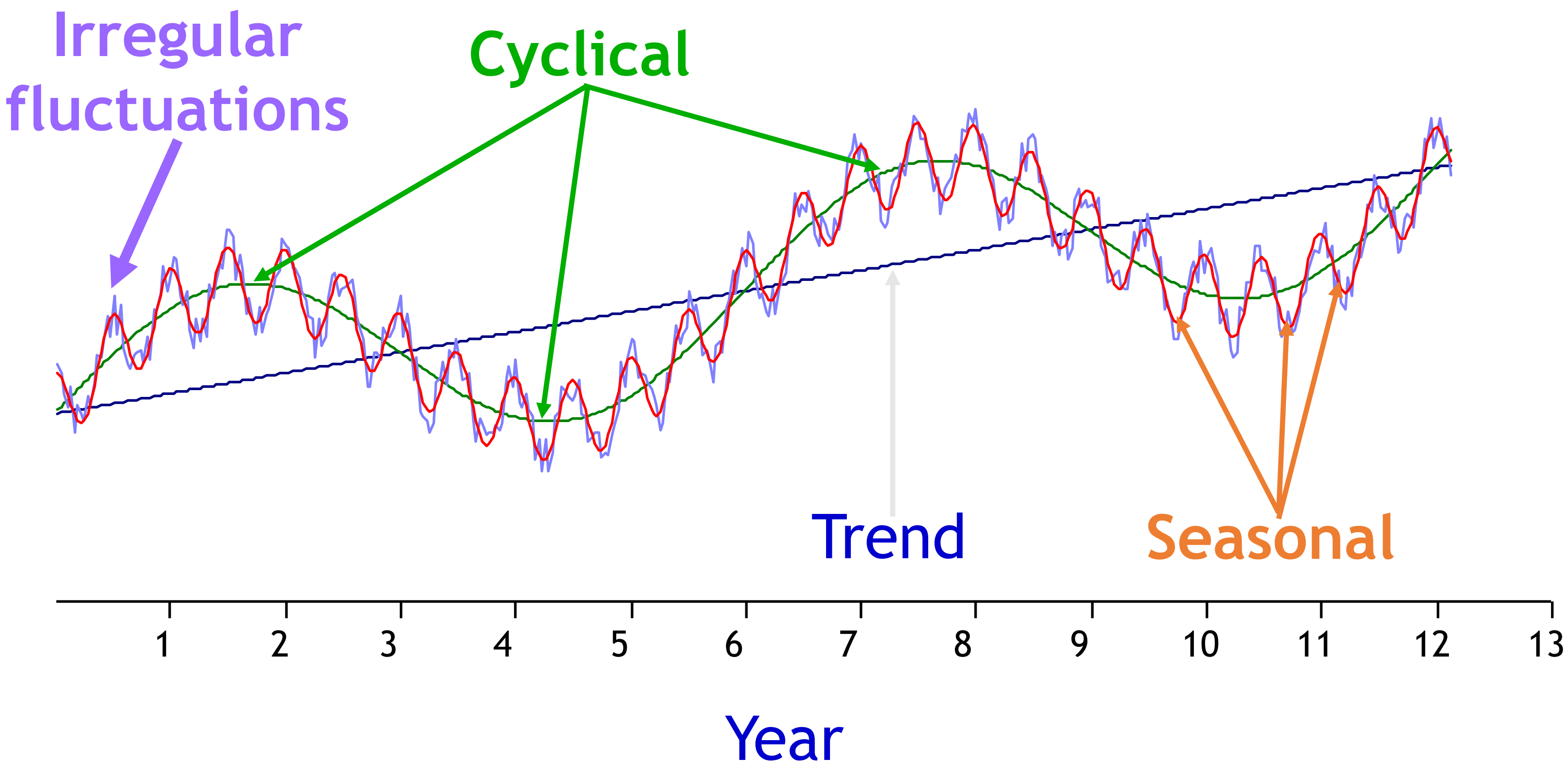
Lecture 9: Support Vector Machines

Olexandr Isayev

Department of Chemistry, CMU

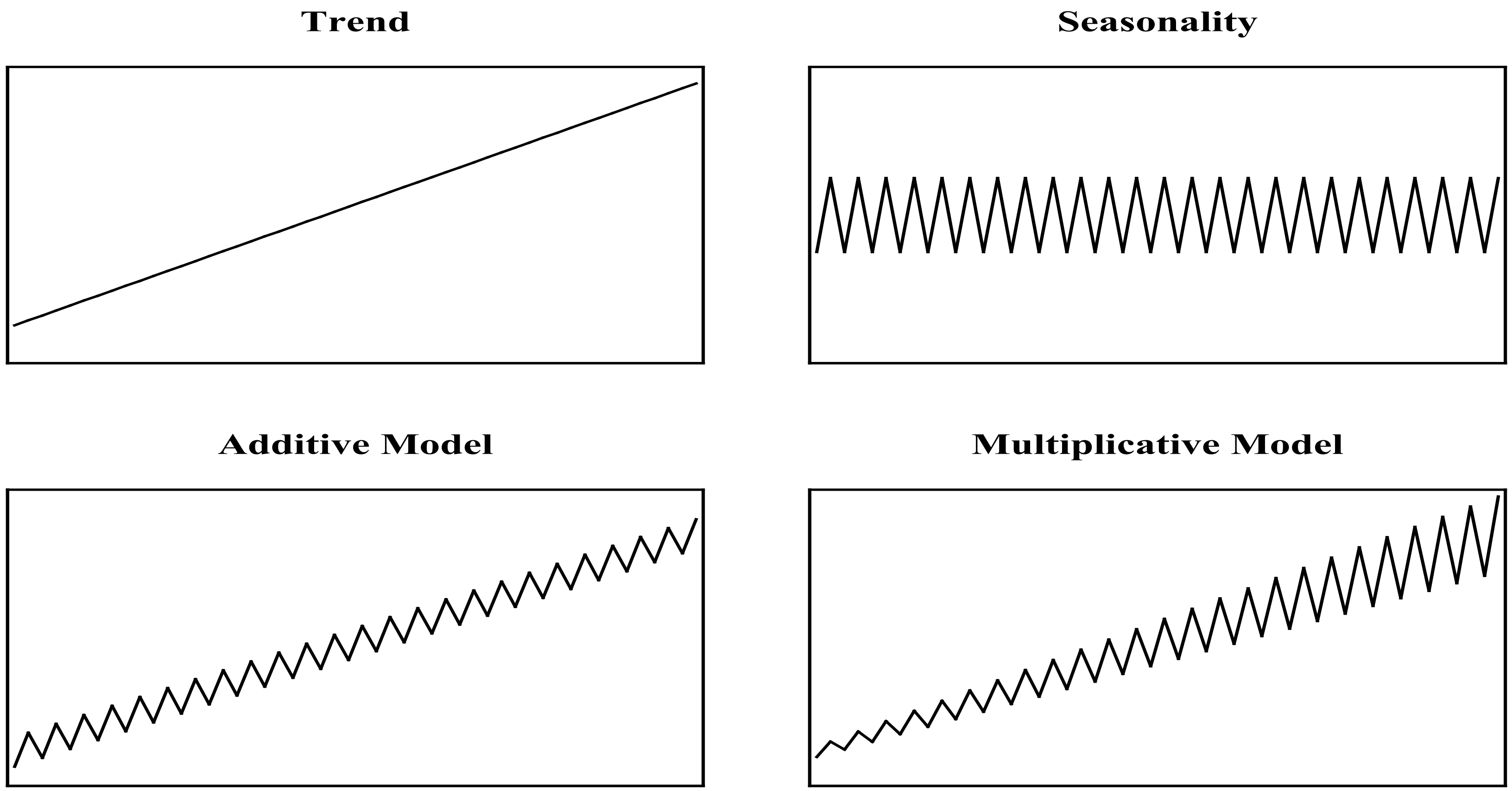
olexandr@cmu.edu

Components of Time-Series Data



Decomposition Models

Some usual shapes



Common Models

- White Noise
- AR
- MA
- ARMA
- ARIMA
- SARIMA
- ARMAX
- Kalman Filter
- Exponential Smoothing, trend, seasons

Autoregressive Models (AR)

An **autoregressive model** is when a value from a time series is regressed on previous values from that same time series. For example, y_t on y_{t-1}:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \epsilon_t.$$

In this regression model, the response variable in the previous time period has become the predictor and the errors have our usual assumptions about errors in a simple linear regression model. The **order** of an autoregression is the number of immediately preceding values in the series that are used to predict the value at the present time. So, the preceding model is a first-order autoregression, written as AR(1).

If we want to predict y using measurements of the previous two observations (y_{t-1}, y_{t-2}), then the autoregressive model for doing so would be:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \epsilon_t.$$

This model is a second-order autoregression, written as AR(2), since the value at time t is predicted from the values at times $t-1$ and $t-2$.

Moving Average (MA)

Moving average model (MA) model generates the current values based on the ***ERRORS*** from the past forecasts instead of using the past values like AR. Past errors are analyzed to produce the current value. Perfecting a baking recipe will be like a moving average model. You will do adjustments for needed sugars or butter for today's baking depending on the previous days' amount to perfect the recipe.

$$MR(X_t, q) = e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q}$$

MR(q)

Moving average model, MR(q) adjusts the model based on the average predictions errors from previous q observations, which can be stated as below, where *e* represents the error terms and *ϑ* represents the weights. q value determines the number of error terms to include in the moving average window.

Time Series Analysis

Time Series is a unique type of machine learning where time plays a critical role in model predictions.

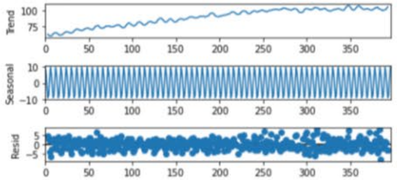
Time Series Components:

1. Trend
2. Seasonality
3. Residual

How to Describe Time Series?

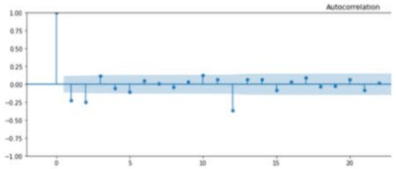
Decomposition

seasonal_decompose(df)



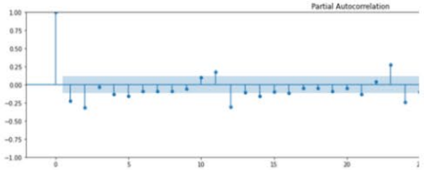
ACF

sm.graphics.tsa.plot_acf()



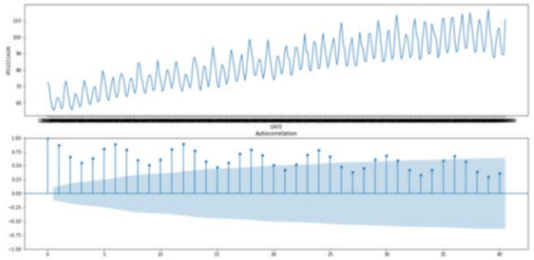
PACF

sm.graphics.tsa.plot_pacf()

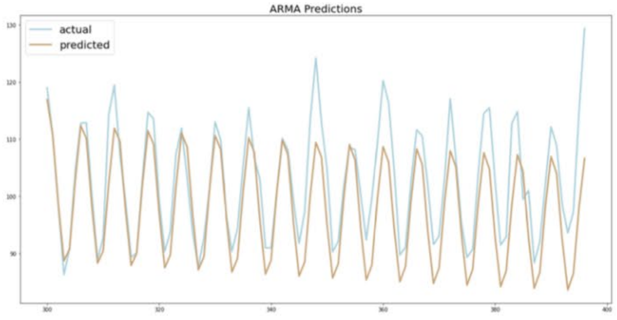


ARMA

Autoregressive + Moving Average

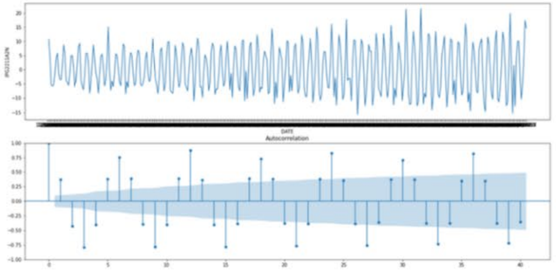


ARIMA(p, 0, q)

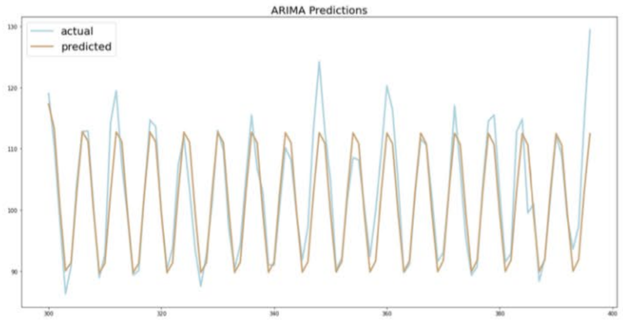


ARIMA

Autoregressive + Moving Average + Trend Differencing

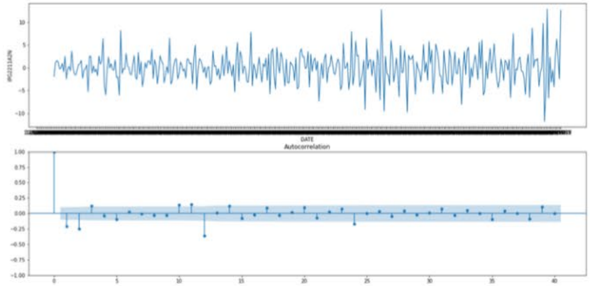


ARIMA(p, d, q)

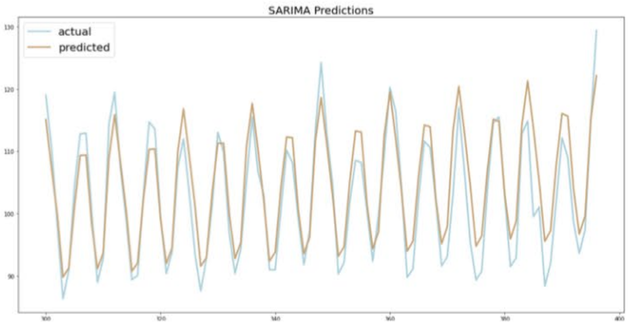


SARIMA

Autoregressive + Moving Average + Trend Differencing + Seasonality Differencing

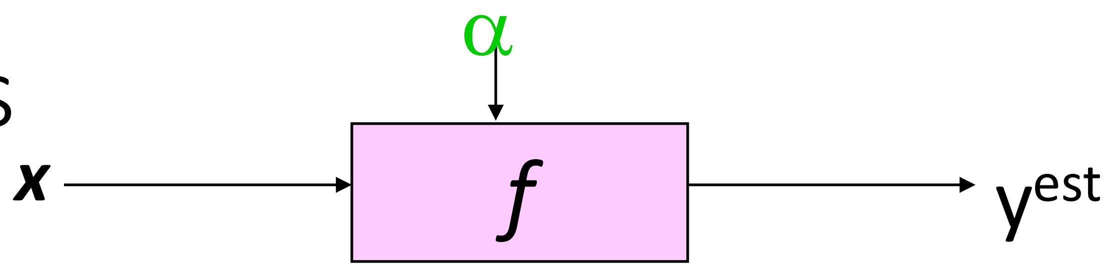


SARIMAX(p, d, q) x (P, D, Q, s)

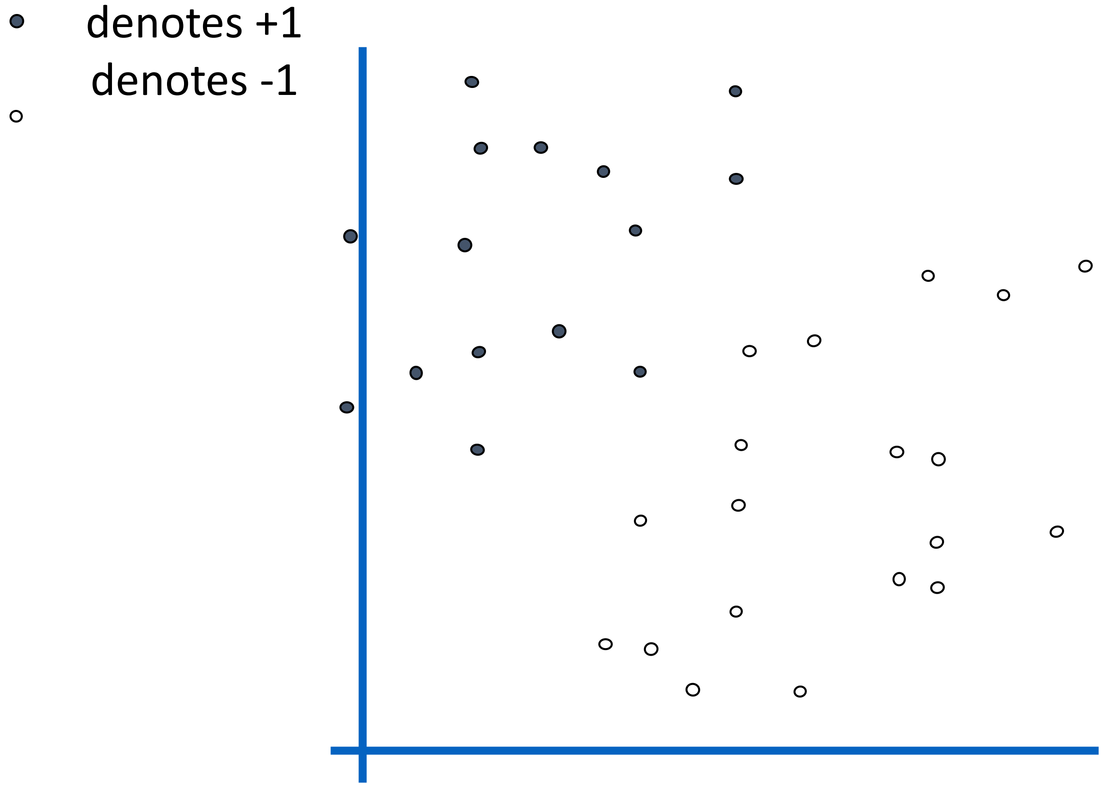


Classification

Linear Classifiers

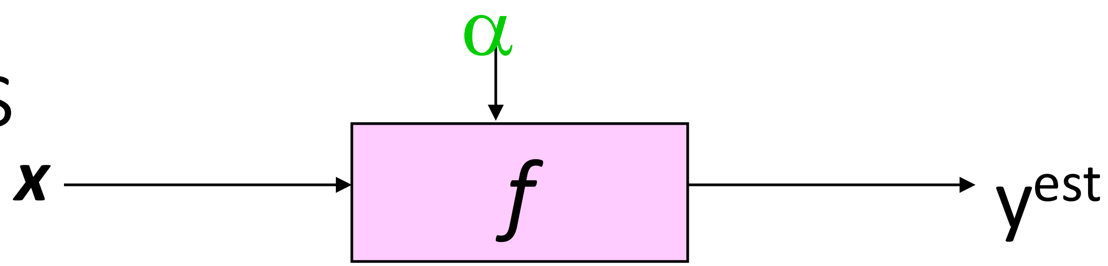


$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}.\ \boldsymbol{x} - b)$

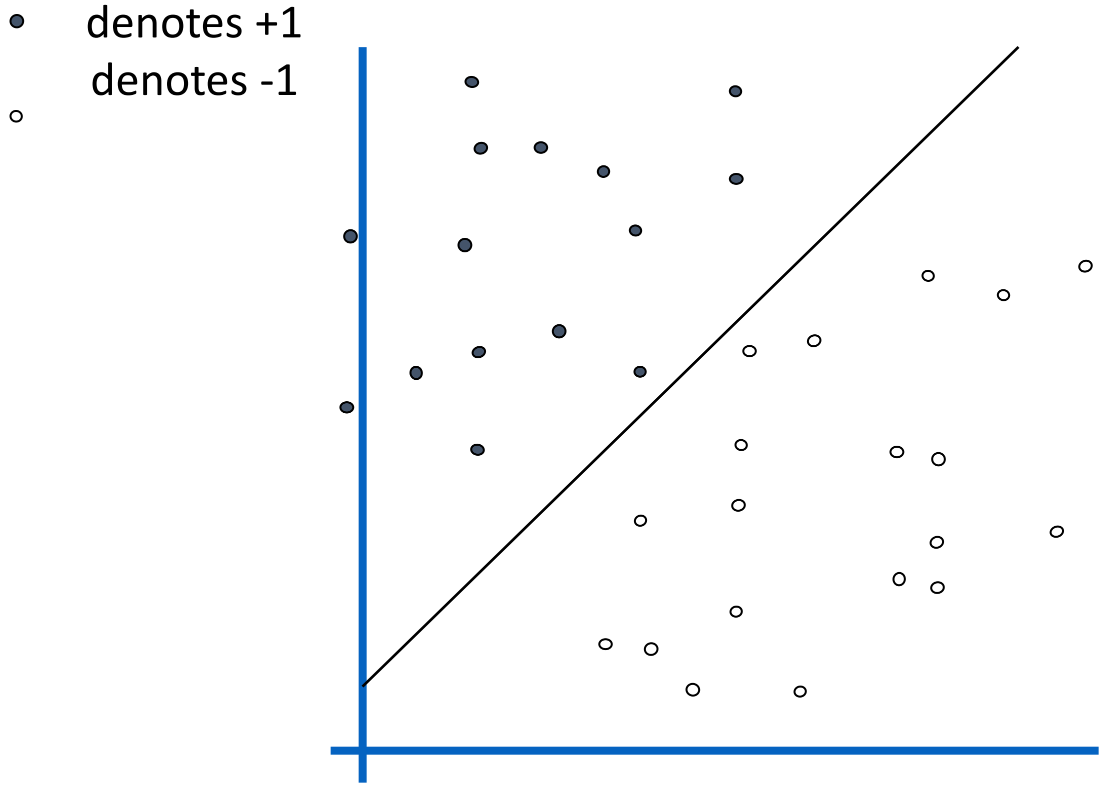


How would you classify this data?

Linear Classifiers

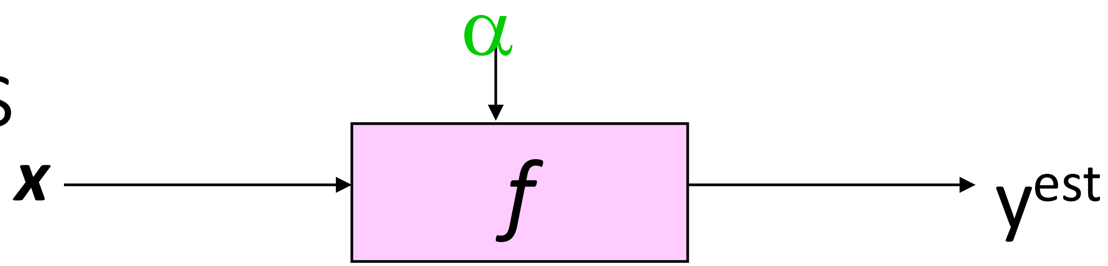


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

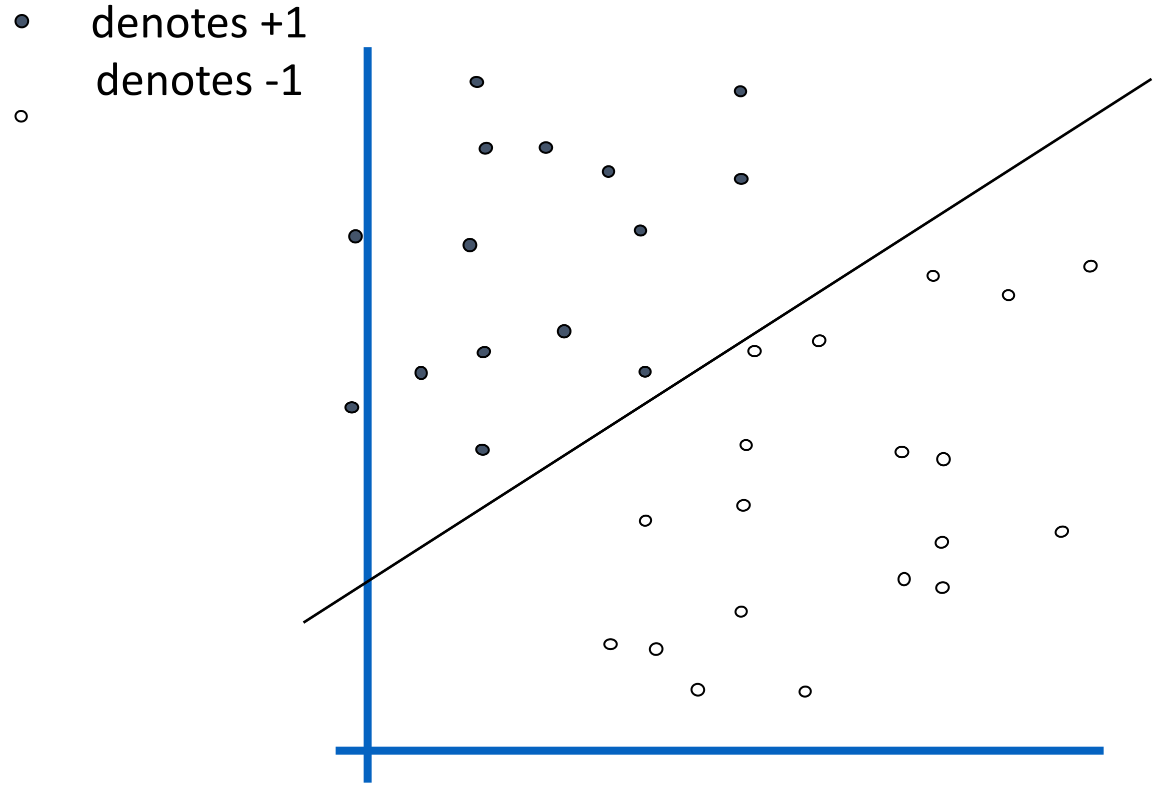


How would you classify this data?

Linear Classifiers

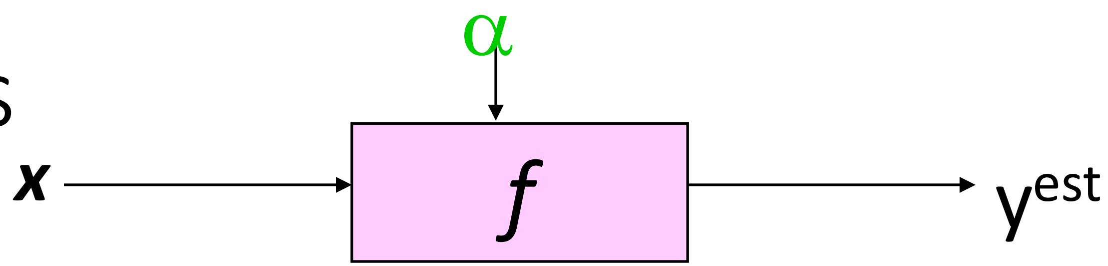


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

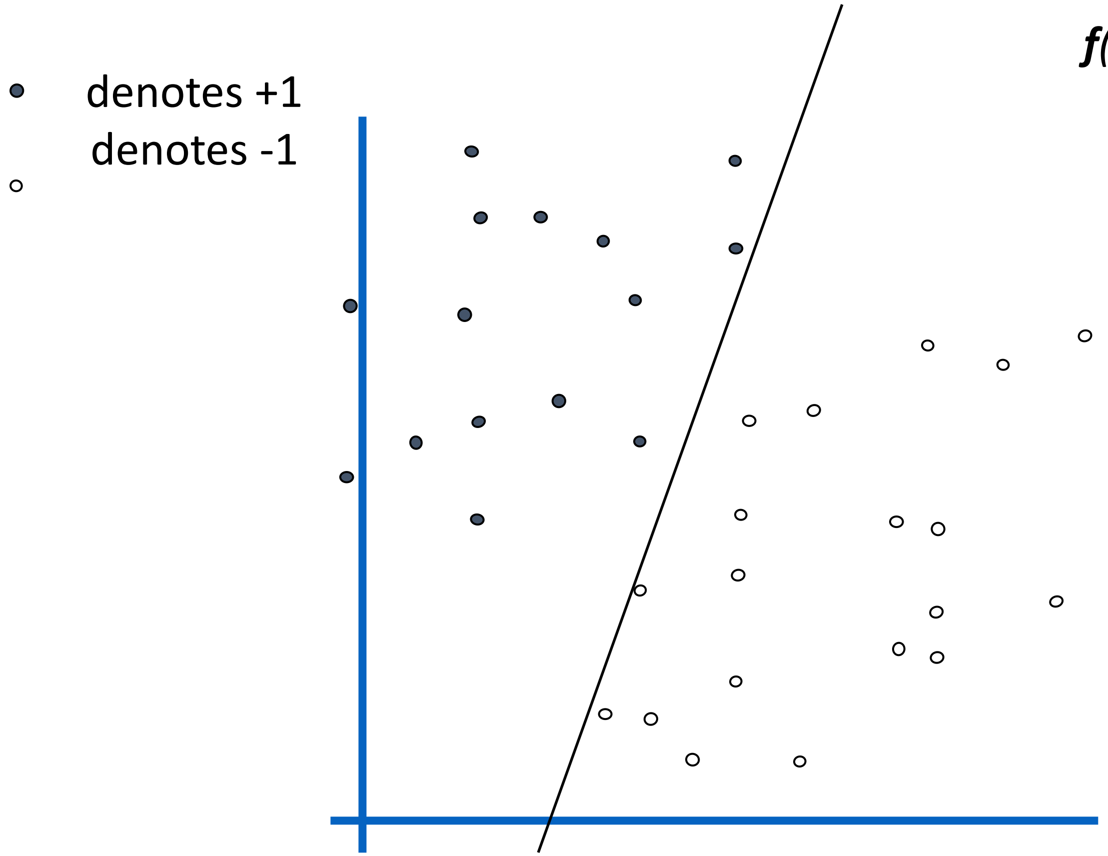


How would you classify this data?

Linear Classifiers

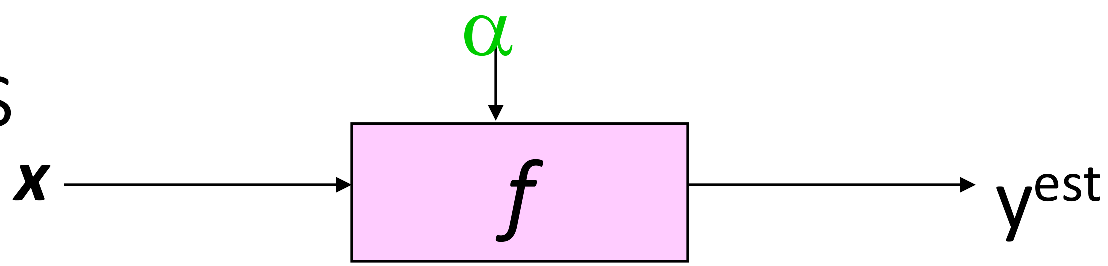


$f(x,w,b) = sign(w.\ x - b)$

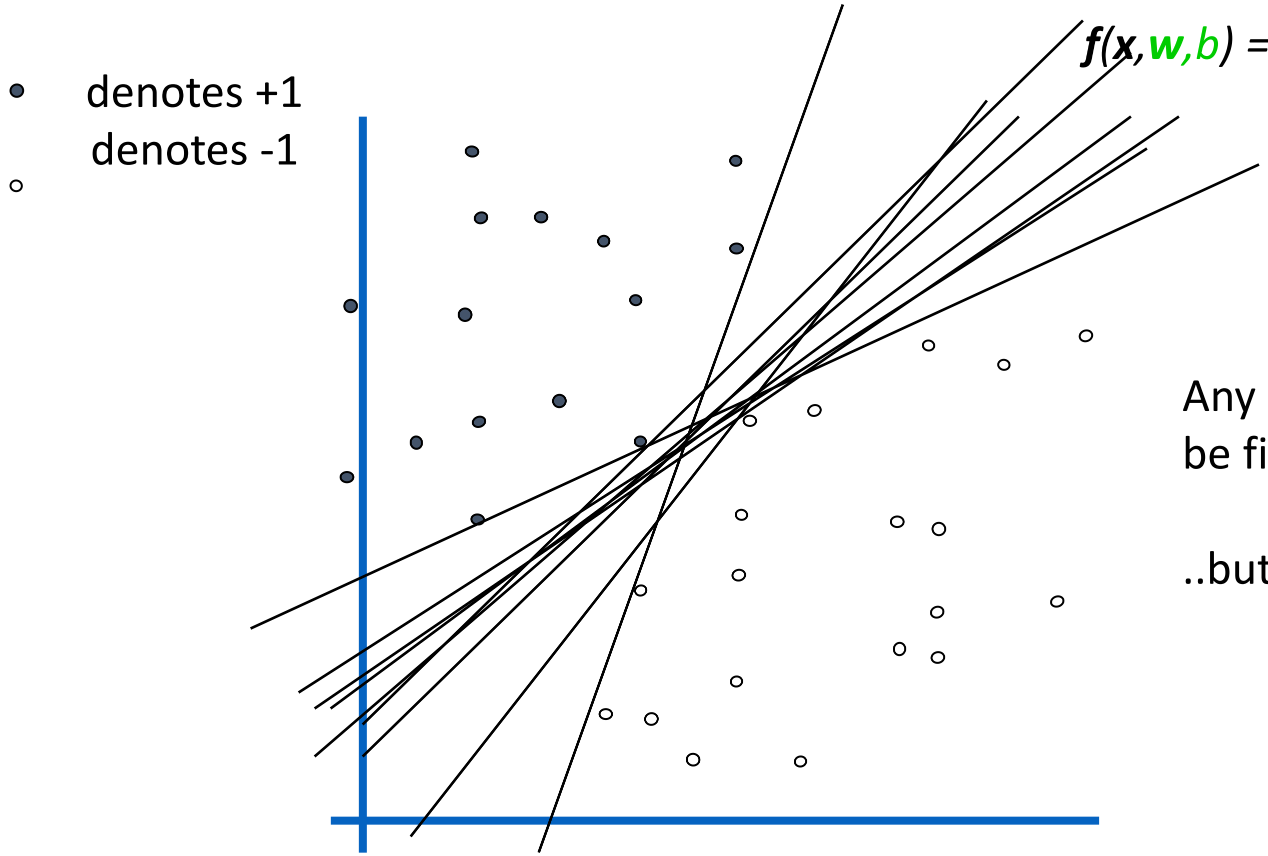


How would you classify this data?

Linear Classifiers



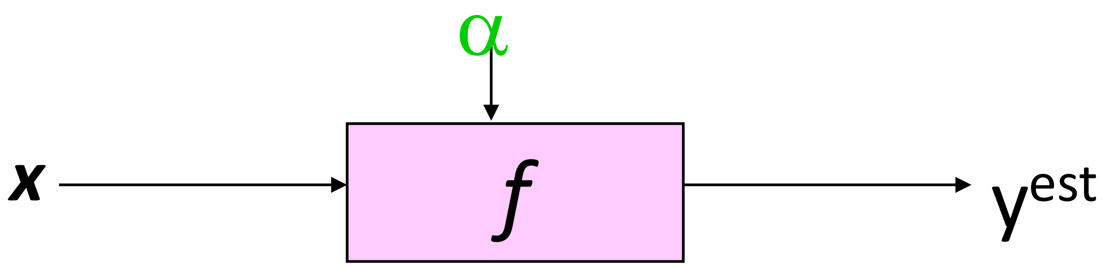
$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}.\ \boldsymbol{x} - b)$



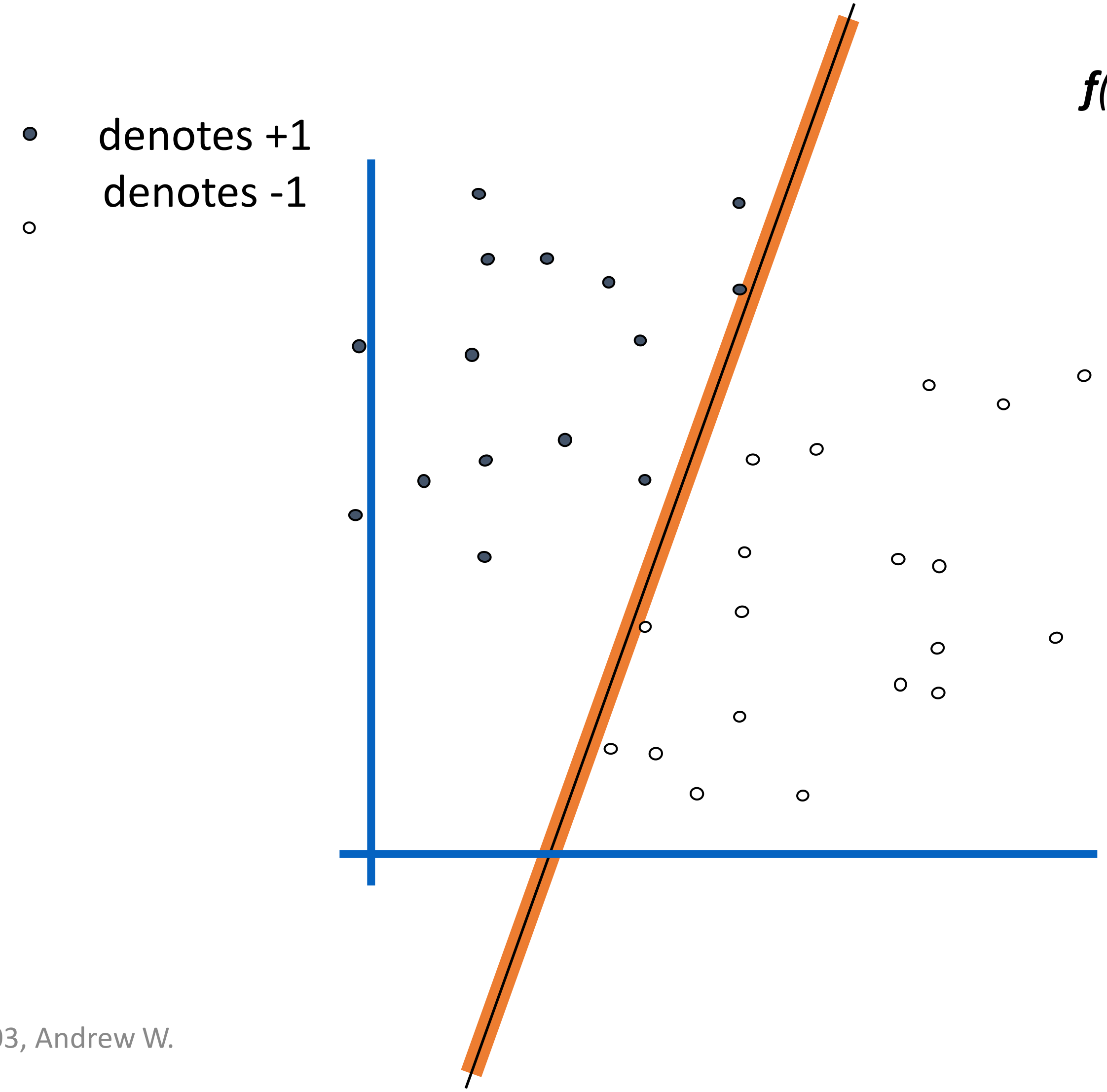
Any of these would be fine..

..but which is best?

Classifier Margin

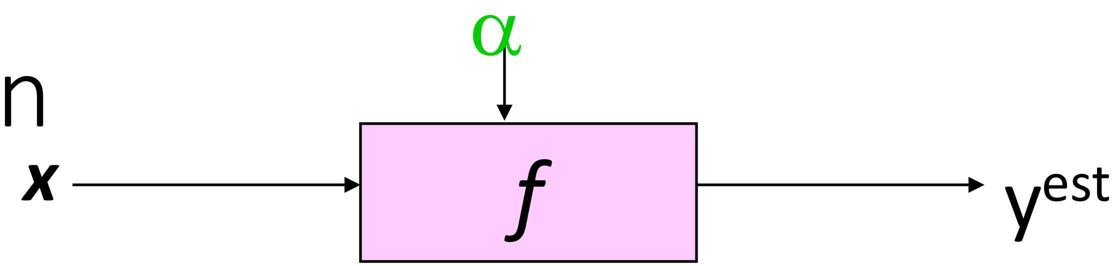


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

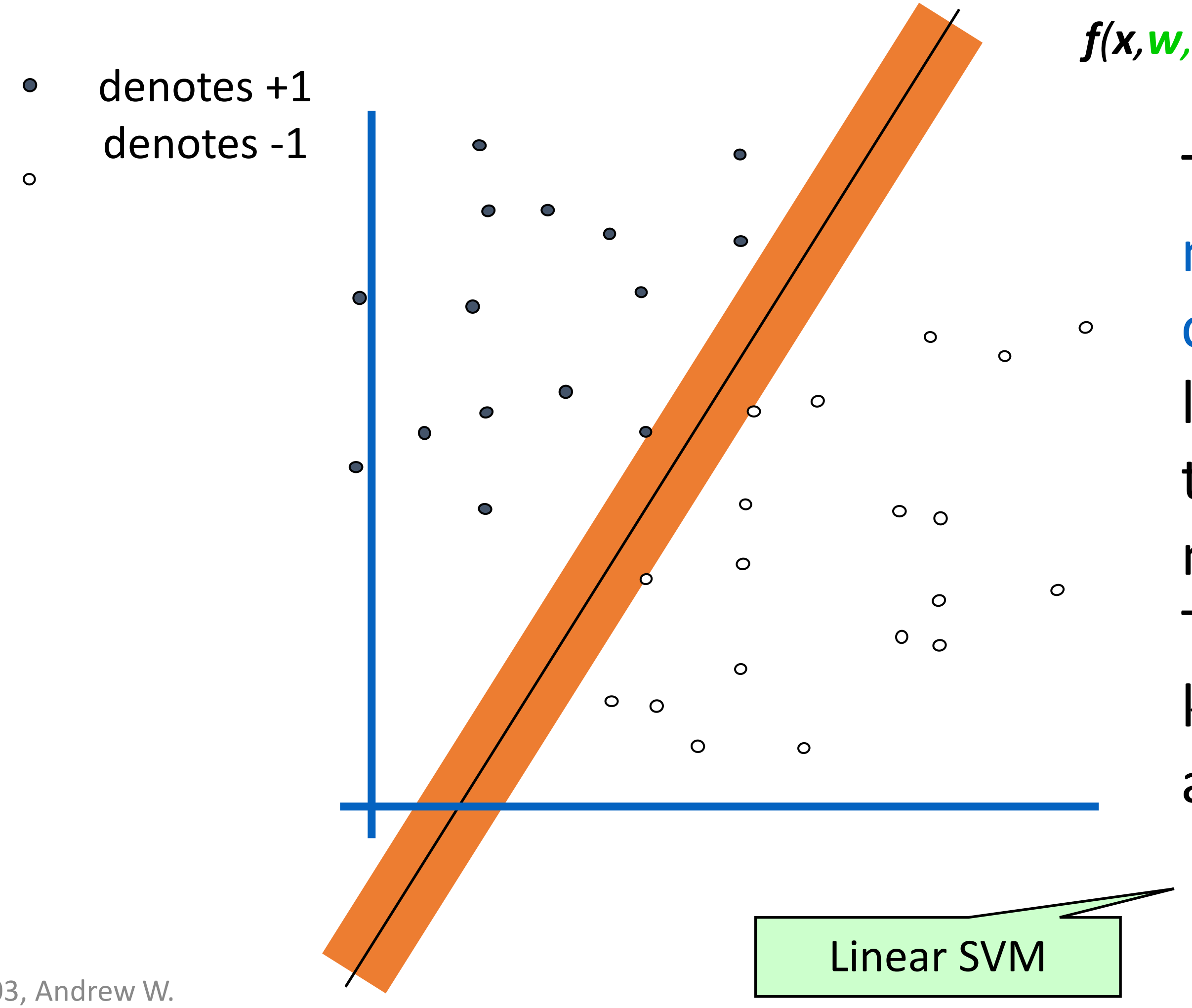


Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



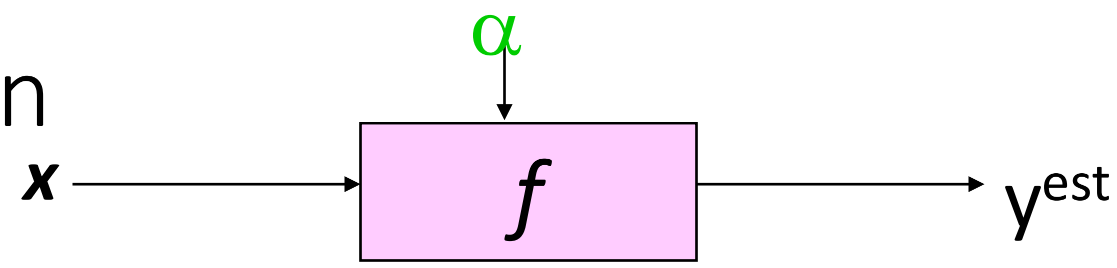
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$



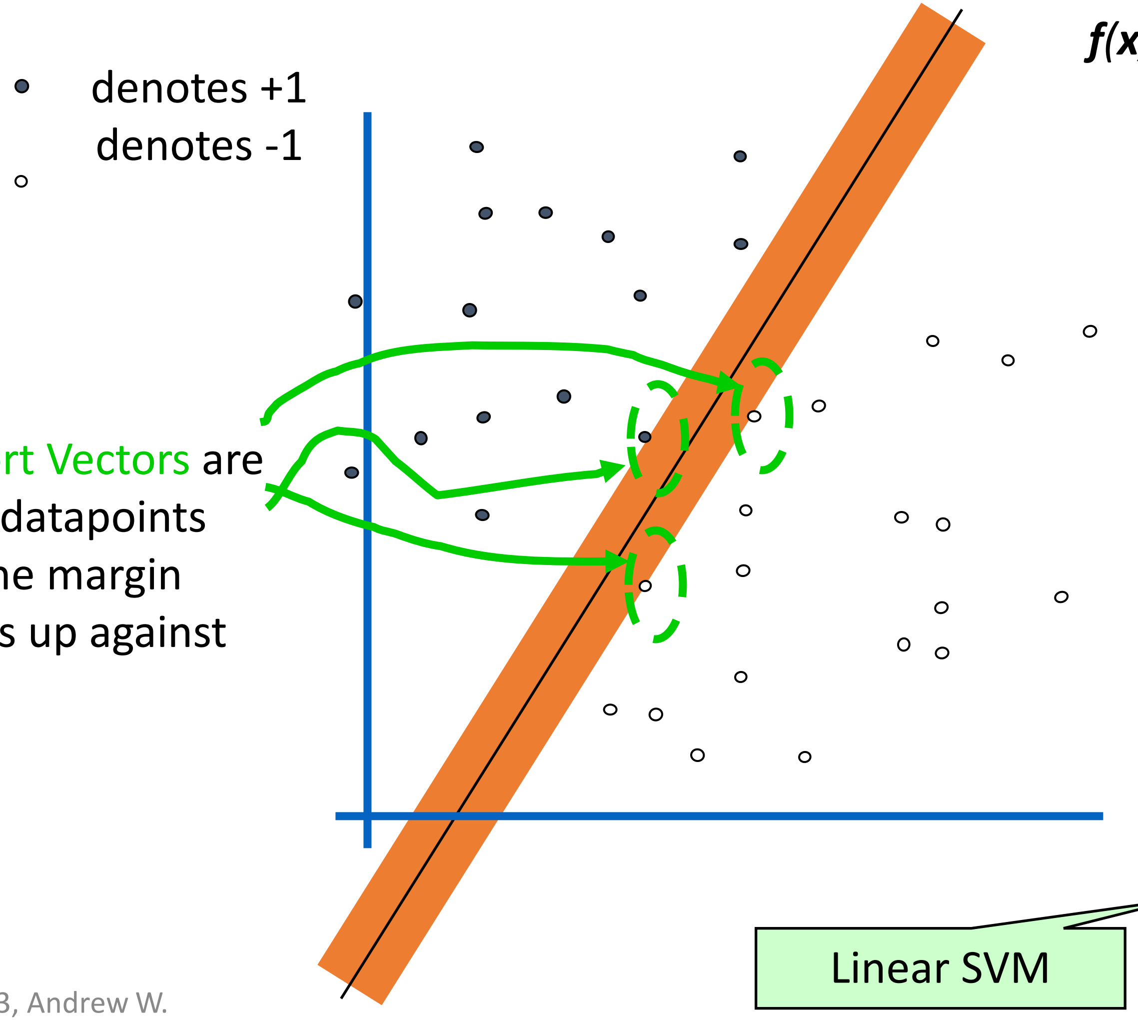
The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$



Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

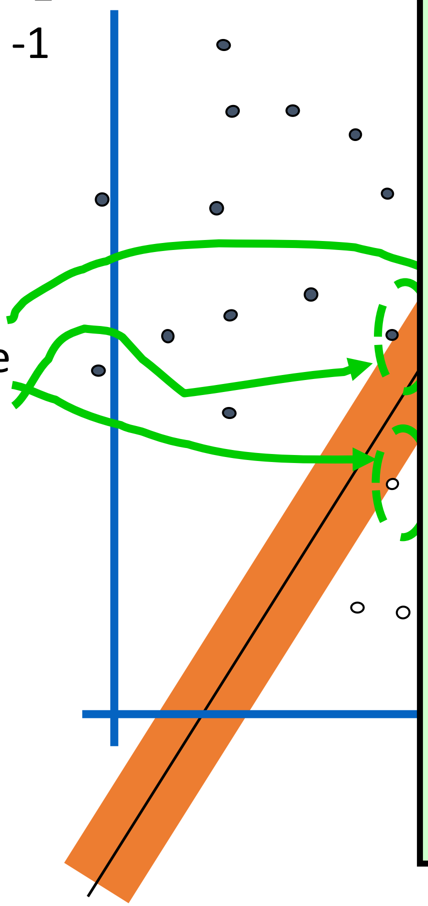
This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Why Maximum Margin?

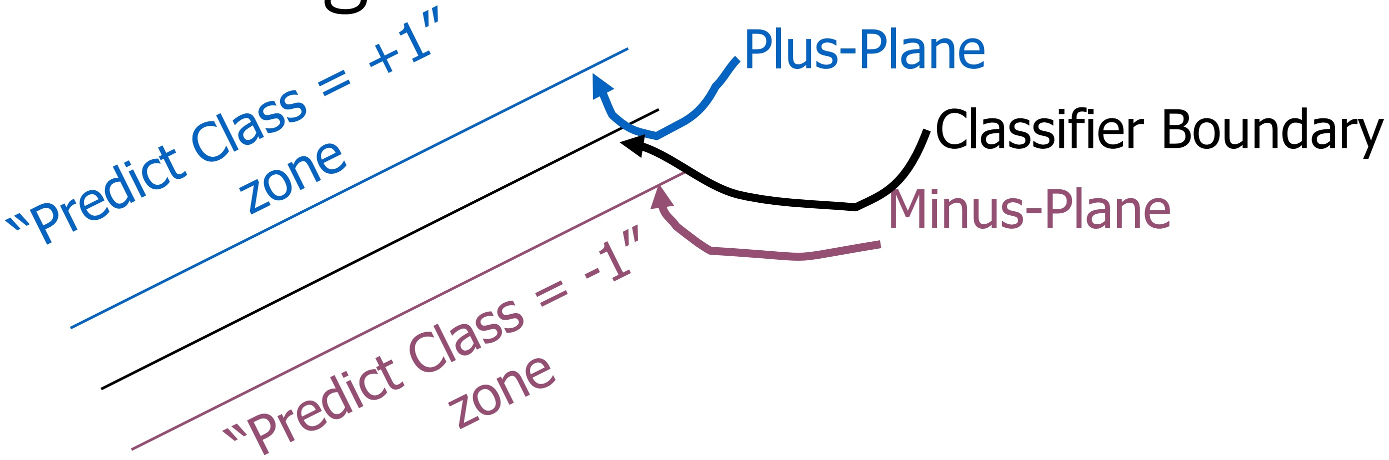
● denotes +1
○ denotes -1

Support Vectors are those datapoints that the margin pushes up against



1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
4. There's some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.
5. Empirically it works very very well.

Specifying a line and margin



- How do we represent this mathematically?
- …in *m* input dimensions?

Distances to the Boundary

- The decision boundary consists of all points $\boldsymbol{x}$ that are solutions to equation: $\boldsymbol{w}^T\boldsymbol{x} + b = 0$.
  - $\boldsymbol{w}$ is a column vector of parameters (weights).
  - $\boldsymbol{x}$ is an input vector.
  - b is a scalar value (a real number).
- If $\boldsymbol{x}_n$ is a training point, its distance to the boundary is computed using this equation:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \left| \frac{\boldsymbol{w}^T\boldsymbol{x} + b}{||\boldsymbol{w}||} \right|$$

Distances to the Boundary

- If $\boldsymbol{x}_n$ is a training point, its distance to the boundary is computed using this equation:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \left| \frac{\boldsymbol{w}^T \boldsymbol{x}_n + b}{\|\boldsymbol{w}\|} \right|$$

- Since the training data are linearly separable, the data from each class should fall on opposite sides of the boundary.
- Suppose that $t_n = -1$ for points of one class, and $t_n = +1$ for points of the other class.
- Then, we can rewrite the distance as:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|}$$

Distances to the Boundary

- So, given a decision boundary defined $\boldsymbol{w}$ and $\boldsymbol{b}$, and given a training input $\boldsymbol{x}_n$, the distance of $\boldsymbol{x}_n$ to the boundary is:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{\|\boldsymbol{w}\|}$$

- If $t_n = -1$, then:
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b < 0$.
  - $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) > 0$.
- If $t_n = 1$, then:
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b > 0$.
  - $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) > 0$.
- So, in all cases, $t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$ is positive.

Optimization Criterion

- If $\boldsymbol{x}_n$ is a training point, its distance to the boundary is computed using this equation:

$$D(\boldsymbol{x}_n, \boldsymbol{w}) = \frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{||\boldsymbol{w}||}$$

- Therefore, the optimal boundary $\boldsymbol{w}_{\text{opt}}$ is defined as:

$$(\boldsymbol{w}_{\text{opt}}, b_{\text{opt}}) = \text{argmax}_{\boldsymbol{w},b} \left\{ \min_n \left[\frac{t_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)}{||\boldsymbol{w}||} \right] \right\}$$

  - In words: find the $\boldsymbol{w}$ and b that maximize the minimum distance of any training input from the boundary.

Constrained Optimization

- Summarizing the previous slides, we want to find:

$$\boldsymbol{w}_{\text{opt}} = \text{argmin}_{\boldsymbol{w}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^{\mathbf{2}} \right\}$$

subject to the following constraints:

$$\forall n \in \{1, \ldots, N\}, t_n\left(\boldsymbol{w}^T \boldsymbol{x}_n + b\right) \geq 1$$

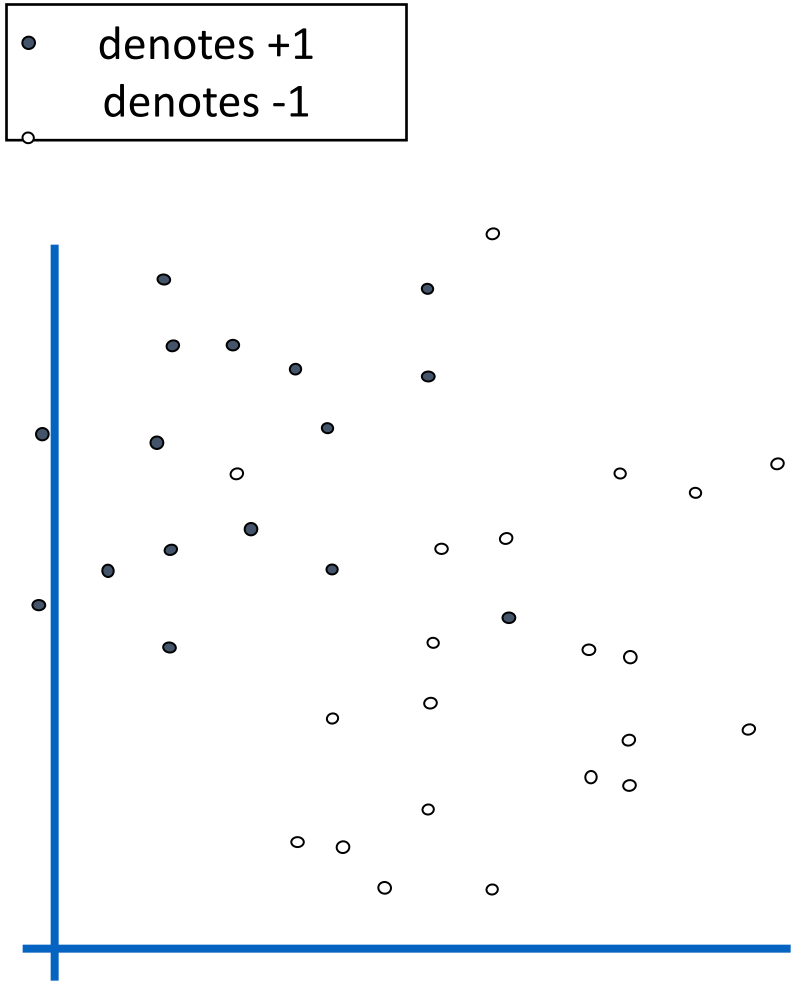
- This is a different optimization problem than what we have seen before.
- We need to minimize a quantity **while satisfying a set of inequalities**.
- This type of problem is a **<u>constrained optimization problem</u>**.

Quadratic Programming

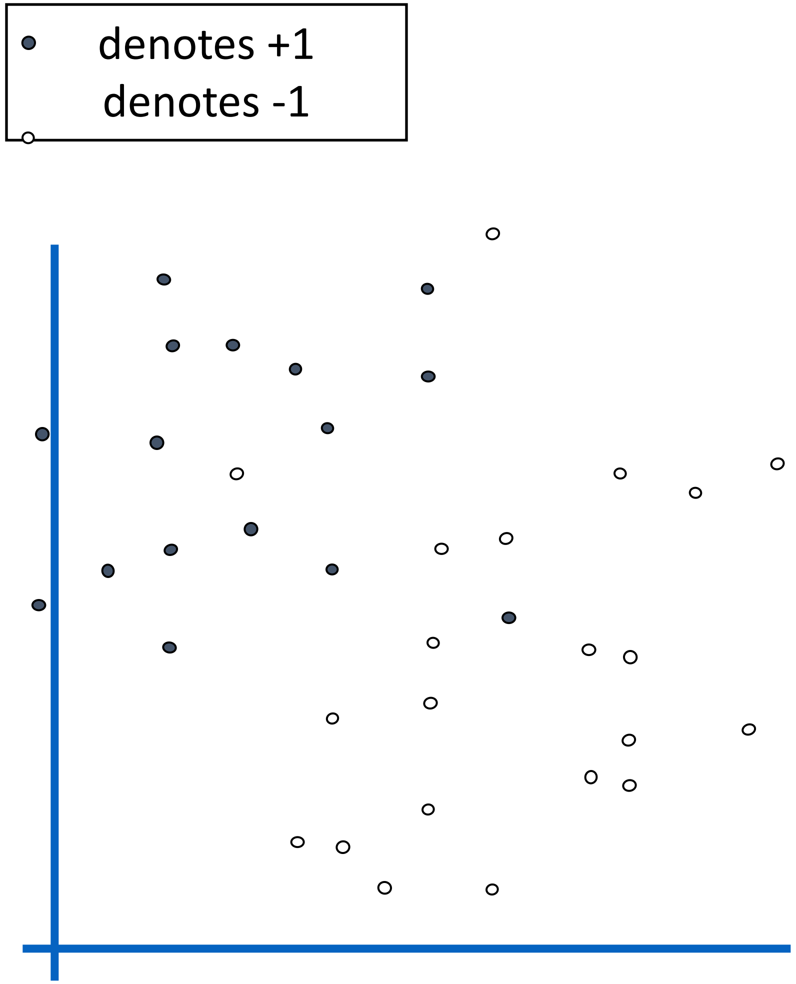
- Our constrained optimization problem can be solved using a method called **quadratic programming**.
- Describing **quadratic programming** in depth is outside the scope of this course.
- Our goal is simply to understand how to use quadratic programming as a black box, to solve our optimization problem.
  - This way, you can use any quadratic programming toolkit (Python has one).

Uh-oh!

This is going to be a problem!
What should we do?



Uh-oh!



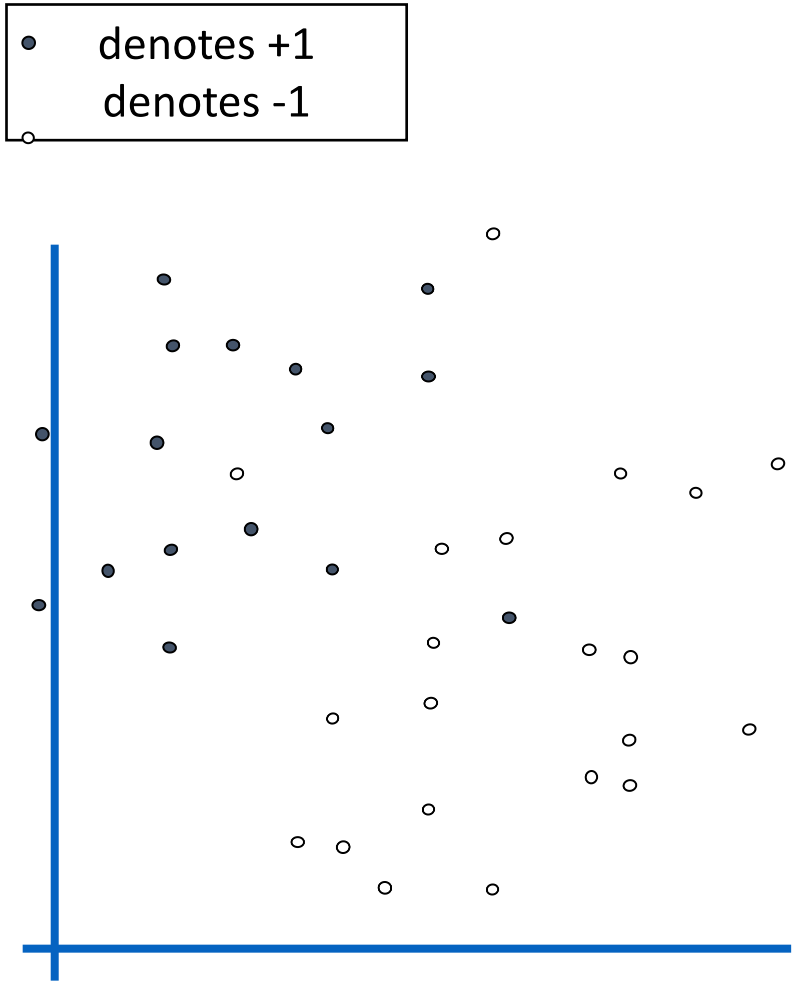
This is going to be a problem!
What should we do?

Idea 1:

Find minimum ***w.w***, while minimizing number of training set errors.

Problemette: Two things to minimize makes for an ill-defined optimization

Uh-oh!



This is going to be a problem!
What should we do?

Idea 1.1:

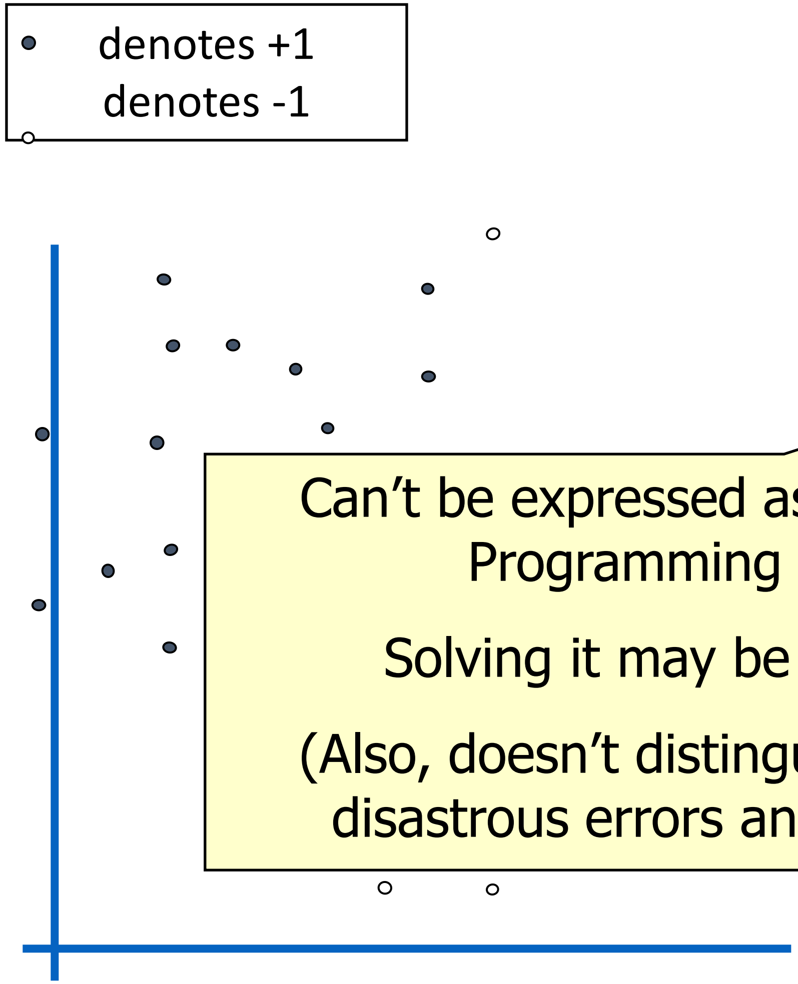
Minimize

w.w *+ C (#train errors)*

Tradeoff parameter

There’s a serious practical problem that’s about to make us reject this approach. Can you guess what it is?

Uh-oh!



This is going to be a problem!
What should we do?

Idea 1.1:

Minimize

***w.w** + C (#train errors)*

Tradeoff parameter

There's a serious practical
t's about to make us
pproach. Can you
t is?

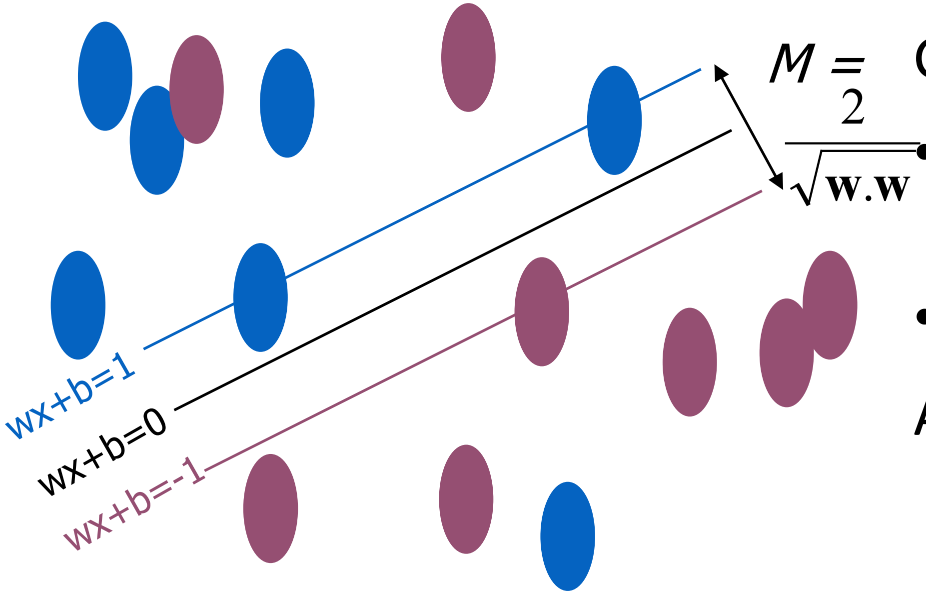
Can't be expressed as a Quadratic Programming problem.

Solving it may be too slow.

(Also, doesn't distinguish between disastrous errors and near misses)

So… any other ideas?

Learning Maximum Margin with Noise



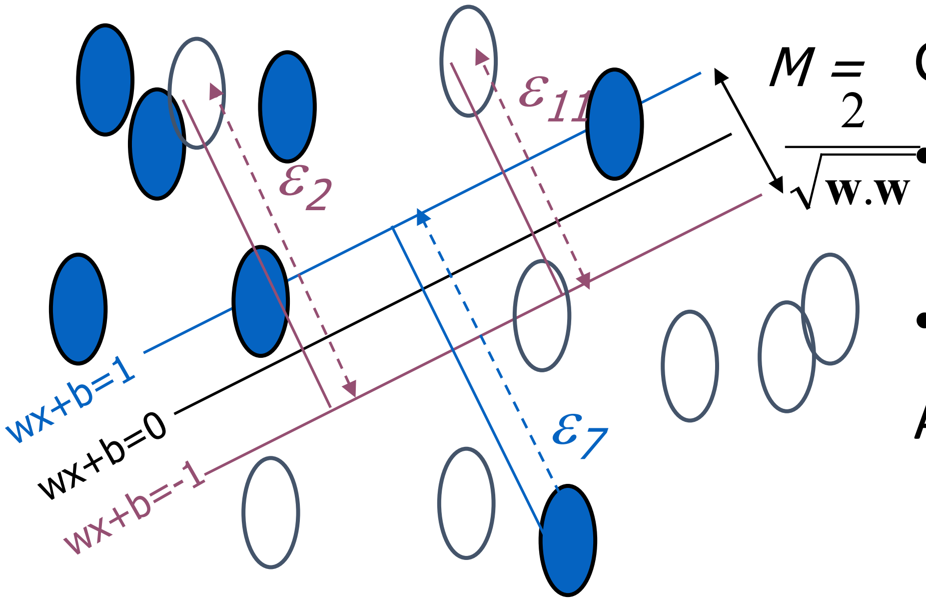
Given guess of $\mathbf{w}$, b we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume R datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Learning Maximum Margin with Noise



Given guess of $\mathbf{w}$, b we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume R datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize $$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

How many constraints will we have? *2R*

What should they be?

$\mathbf{w} \,.\, \mathbf{x}_k + b >= 1-\varepsilon_k$ if $y_k = 1$

$\mathbf{w} \,.\, \mathbf{x}_k + b <= -1+\varepsilon_k$ if $y_k = -1$

SVM Objective Function

- The problem we solved is:

$$Min\ ½\ ||\boldsymbol{w}||^2\ +\ c\sum \xi_i$$

- Where $\xi_i > 0$ is called a slack variable, and is defined by:
  - $\xi_i = \max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i)$
  - Equivalently, we can say that: $y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 1 - \xi_i$; $\xi_i \geq 0$
- And this can be written as:

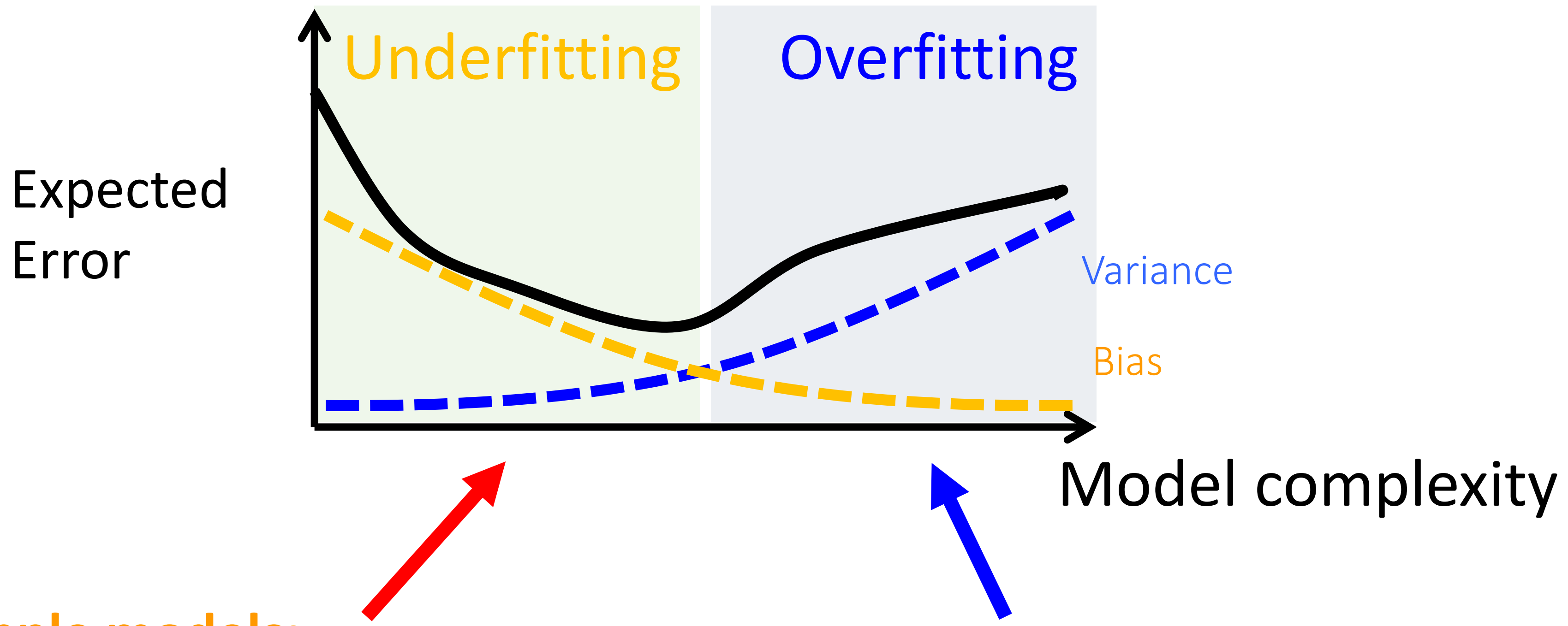
$$Min\ \underbrace{½\ ||\boldsymbol{w}||^2}_{\text{Regularization term}}\ +\ \underbrace{c\sum\xi_i}_{\text{Empirical loss}}$$

Can be replaced by other **regularization functions**

Can be replaced by other **loss functions**

- General Form of a learning algorithm:
  - Minimize empirical loss, and Regularize (to avoid over fitting)

Underfitting and Overfitting



Simple models:
High bias and low variance

Smaller C

Complex models:
High variance and low bias

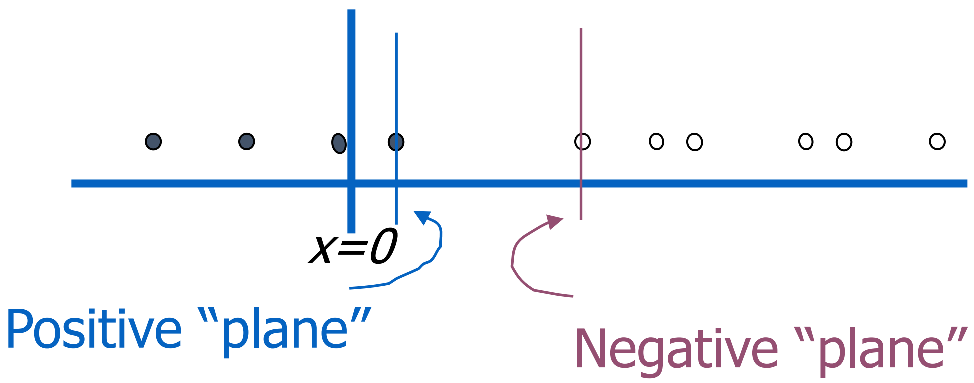
Larger C

Suppose we're in 1-dimension

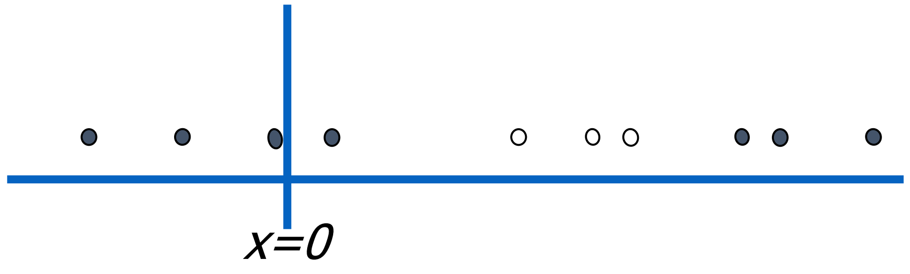
What would SVMs do with this data?

Suppose we're in 1-dimension

Not a big surprise



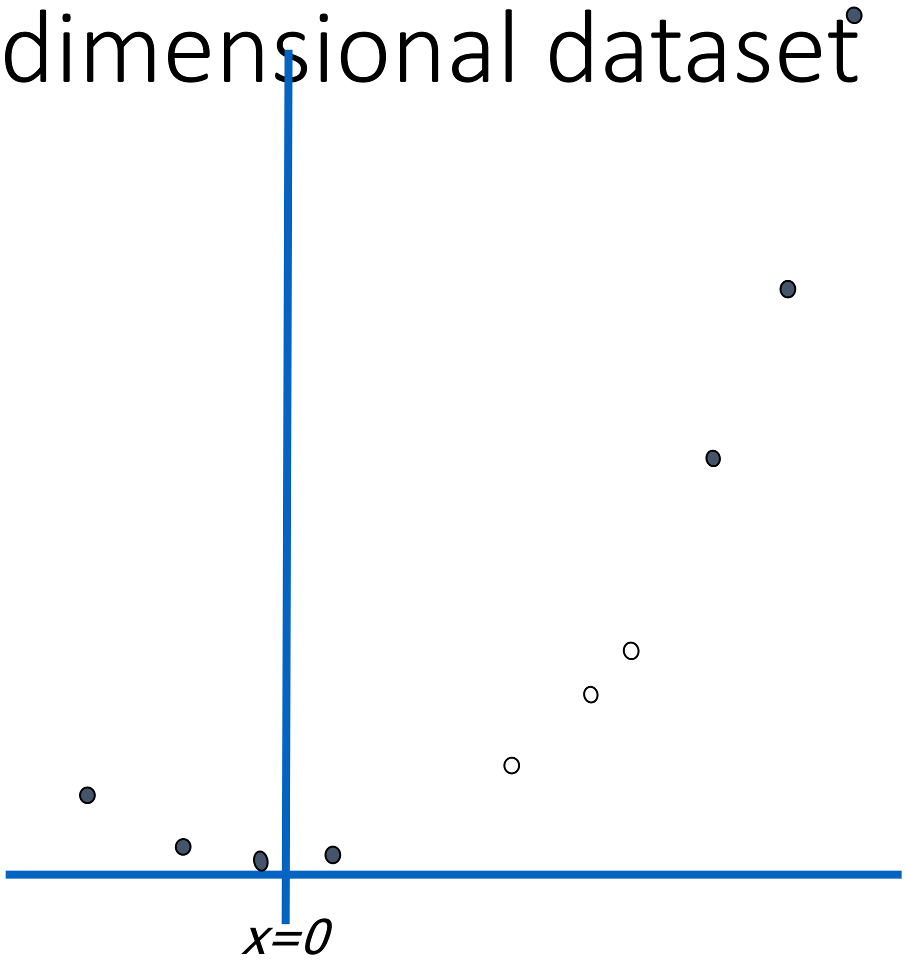
Harder 1-dimensional dataset



That's wiped the smirk off SVM's face.

What can be done about this?

Harder 1-dimensional dataset

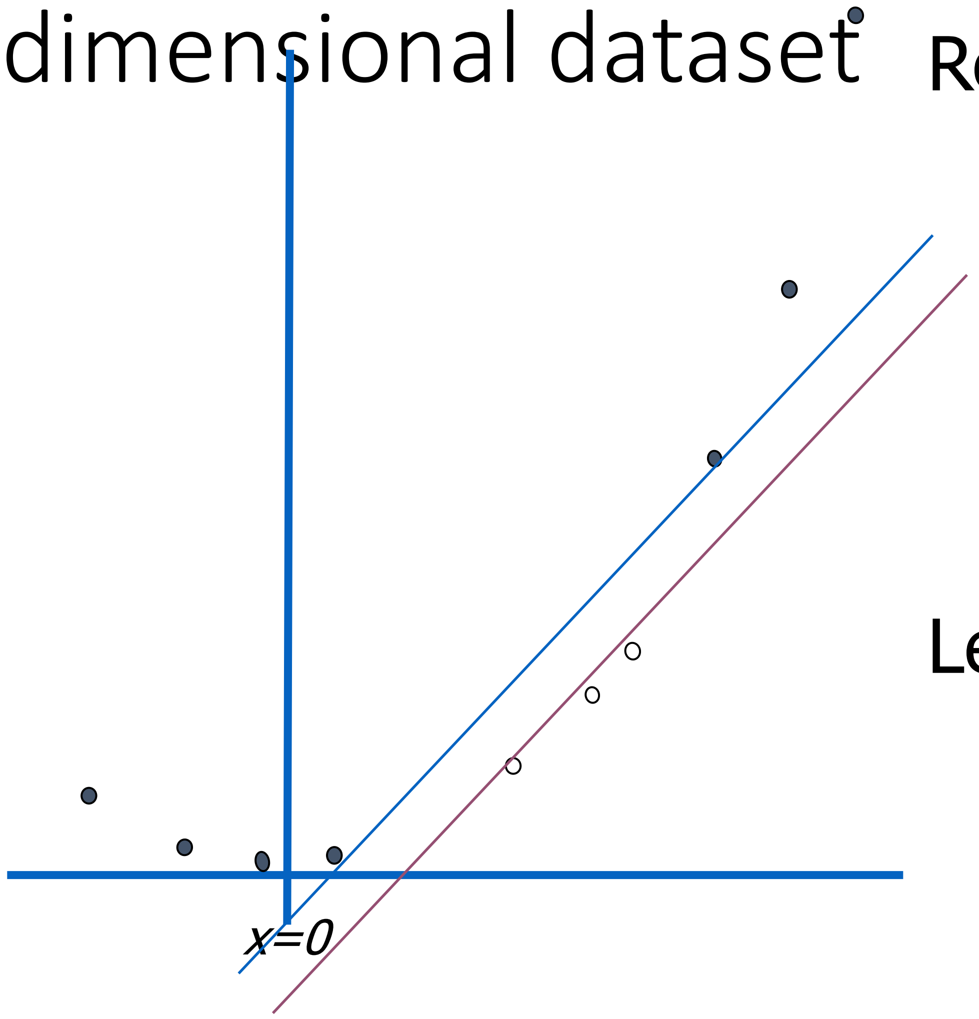


Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Harder 1-dimensional dataset



Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Kernels and Basis Functions

- In general, kernels make it easy to incorporate basis functions into SVMs:
  - Define $\varphi(\boldsymbol{x})$ any way you like.
  - Define $k(\boldsymbol{x}, \boldsymbol{z}) = \varphi(\boldsymbol{x})^{\boldsymbol{T}} \varphi(\boldsymbol{z})$.

- The kernel function represents a dot product, but in a (typically) higher-dimensional feature space compared to the original space of $\boldsymbol{x}$ and $\boldsymbol{z}$.

Common SVM basis functions

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 1 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \mathrm{KernelFn}\left(\frac{|\mathbf{x}_k - \mathbf{c}_j|}{\mathrm{KW}}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

Polynomial Kernels

- Let $\boldsymbol{x}$ and $\boldsymbol{z}$ be D-dimensional vectors.
- A polynomial kernel of degree d is defined as:

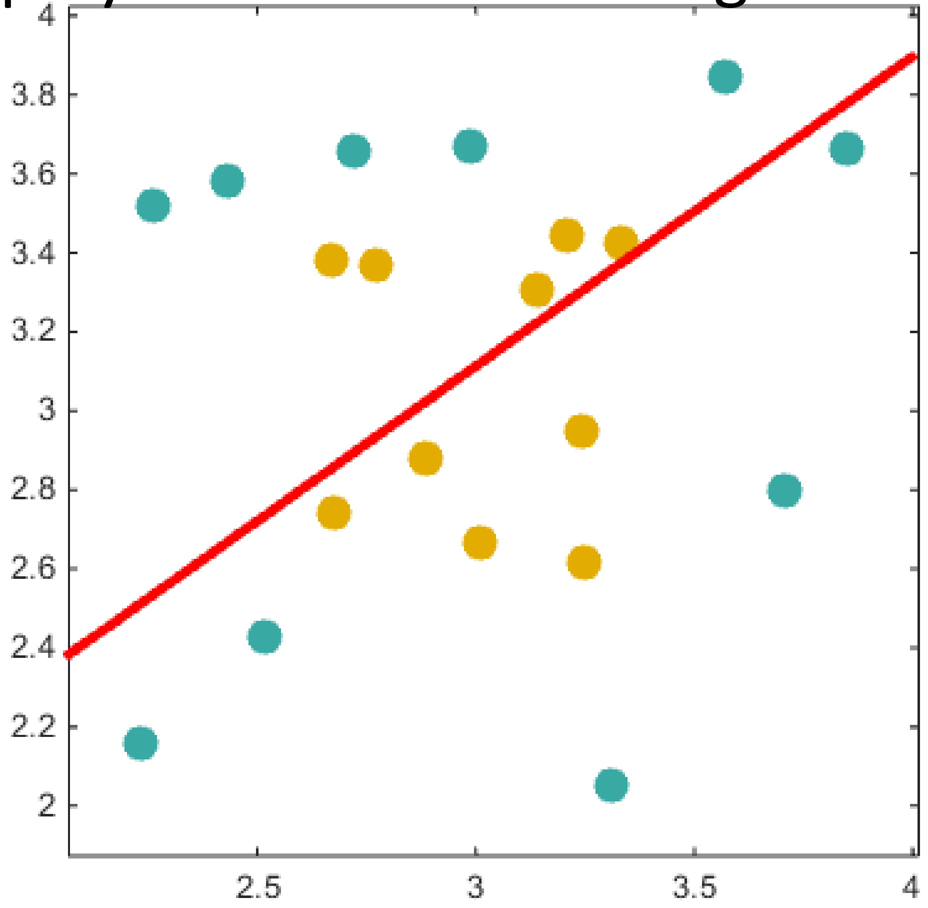
$$k(\boldsymbol{x}, \boldsymbol{z}) = \left(c + \boldsymbol{x}^T \boldsymbol{z}\right)^d$$

- The kernel $k(\boldsymbol{x}, \boldsymbol{z}) = \left(1 + \boldsymbol{x}^T \boldsymbol{z}\right)^2$ that we saw a couple of slides back was a quadratic kernel.
- Parameter c controls the trade-off between influence higher-order and lower-order terms.
  - Increasing values of c give increasing influence to lower-order terms.

Polynomial Kernels – An Easy Case

Decision boundary with polynomial kernel of degree 1.

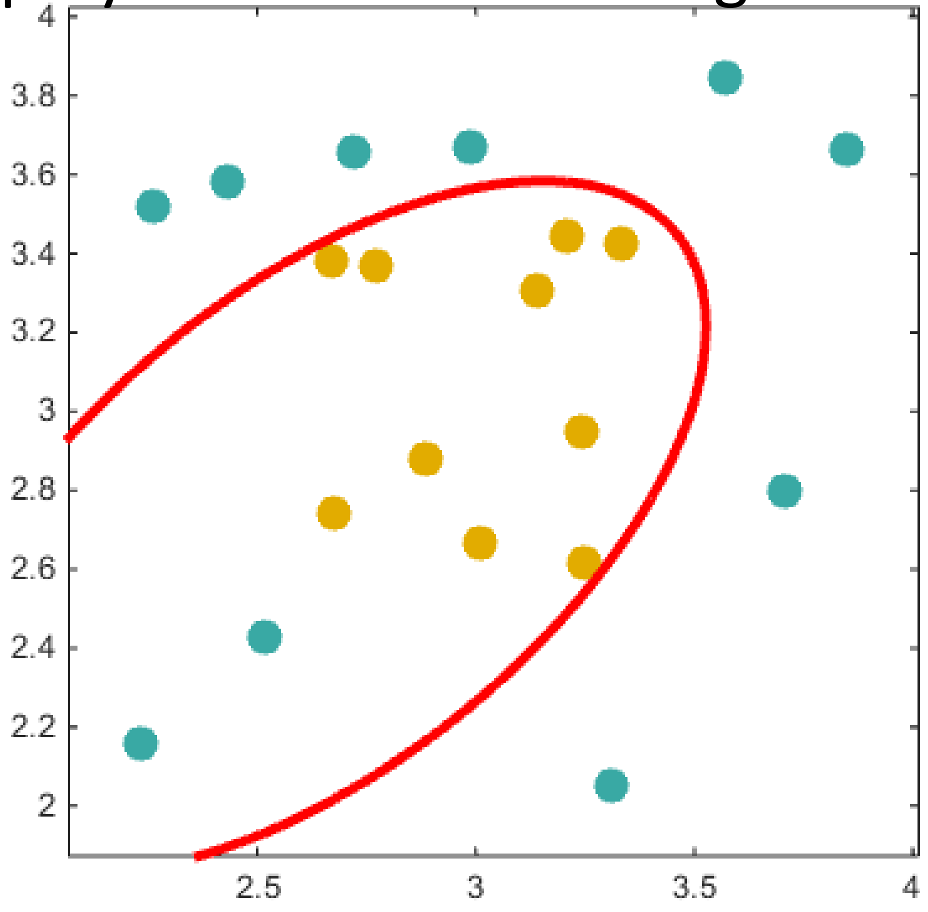
This is identical to the result using the standard dot product as kernel.



Polynomial Kernels – An Easy Case

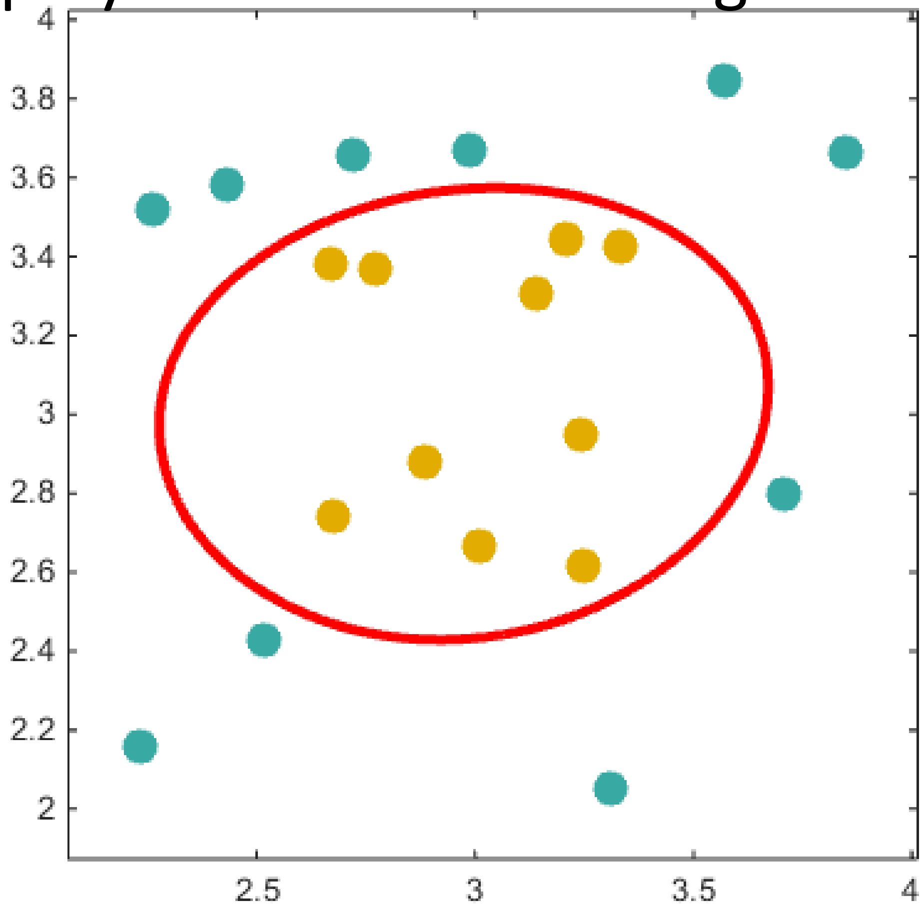
Decision boundary with polynomial kernel of degree 2.

The decision boundary is not linear anymore.



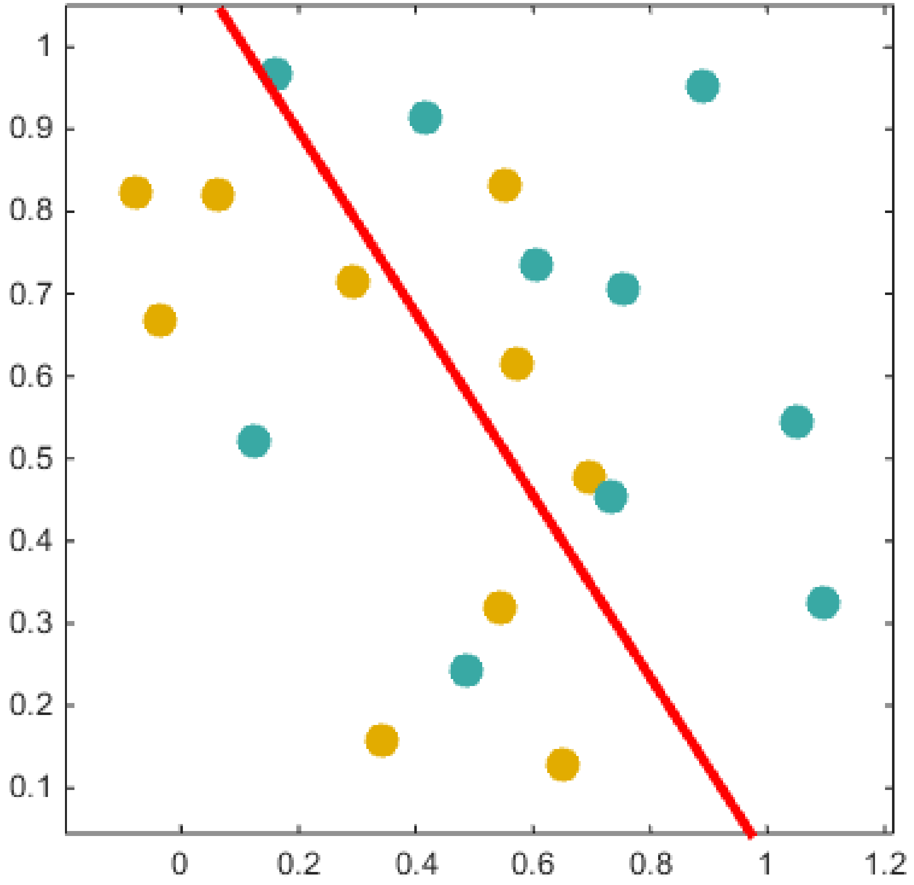
Polynomial Kernels – An Easy Case

Decision boundary with polynomial kernel of degree 3.



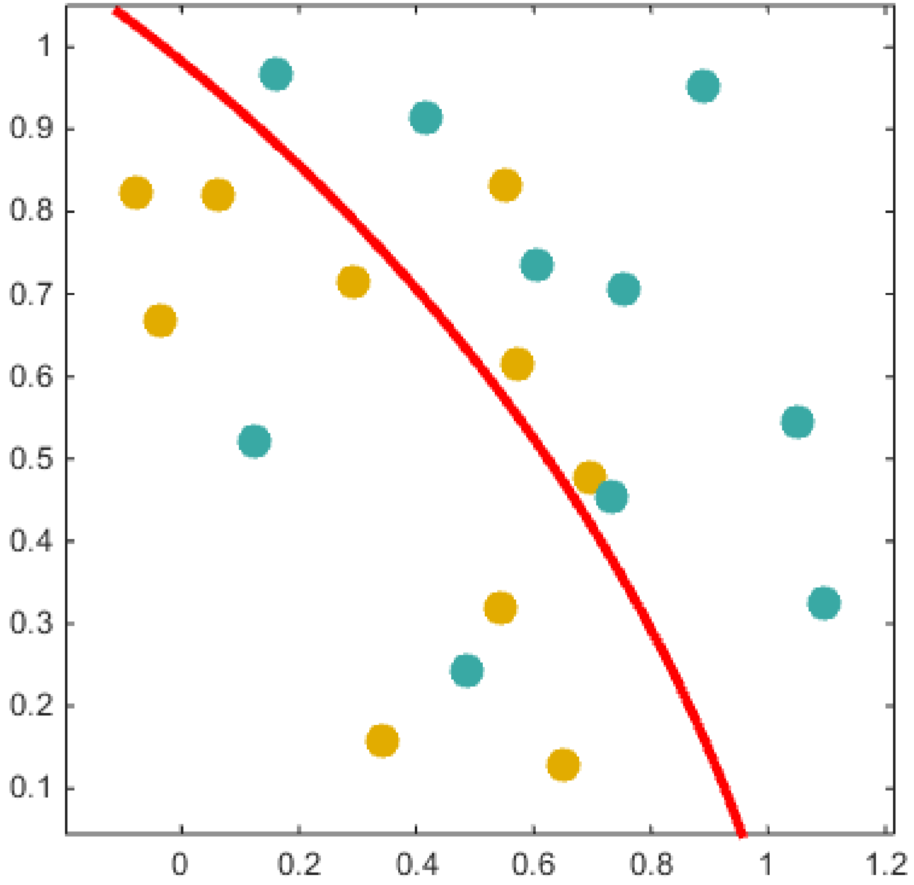
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 1.



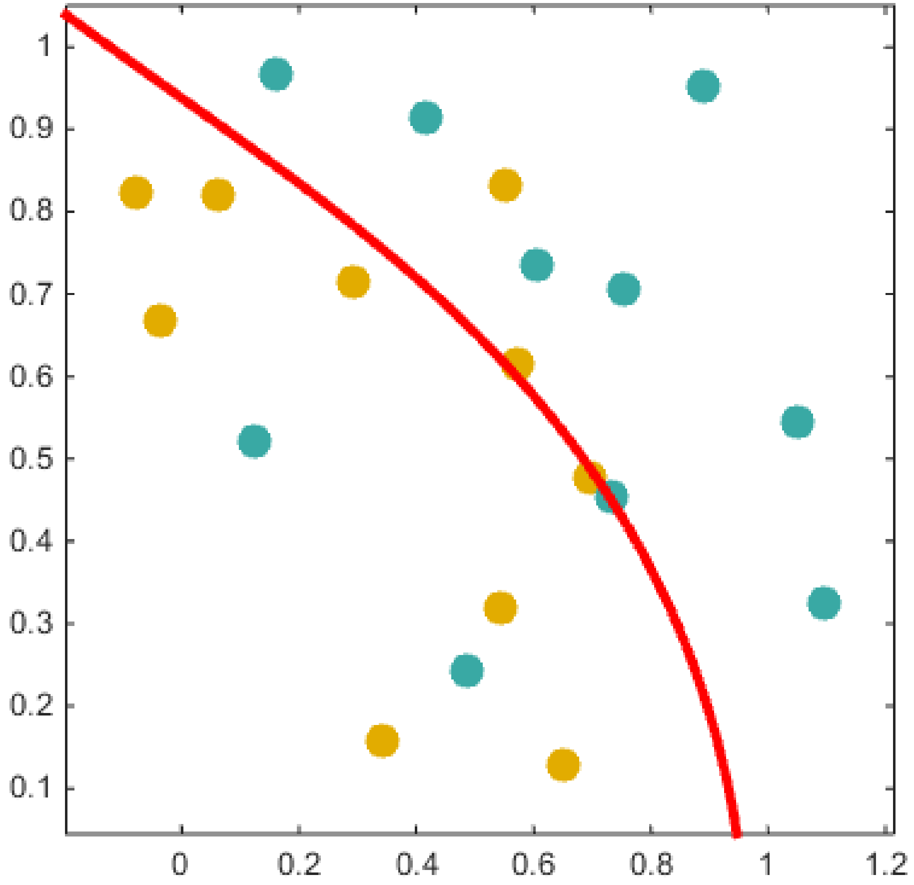
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 2.



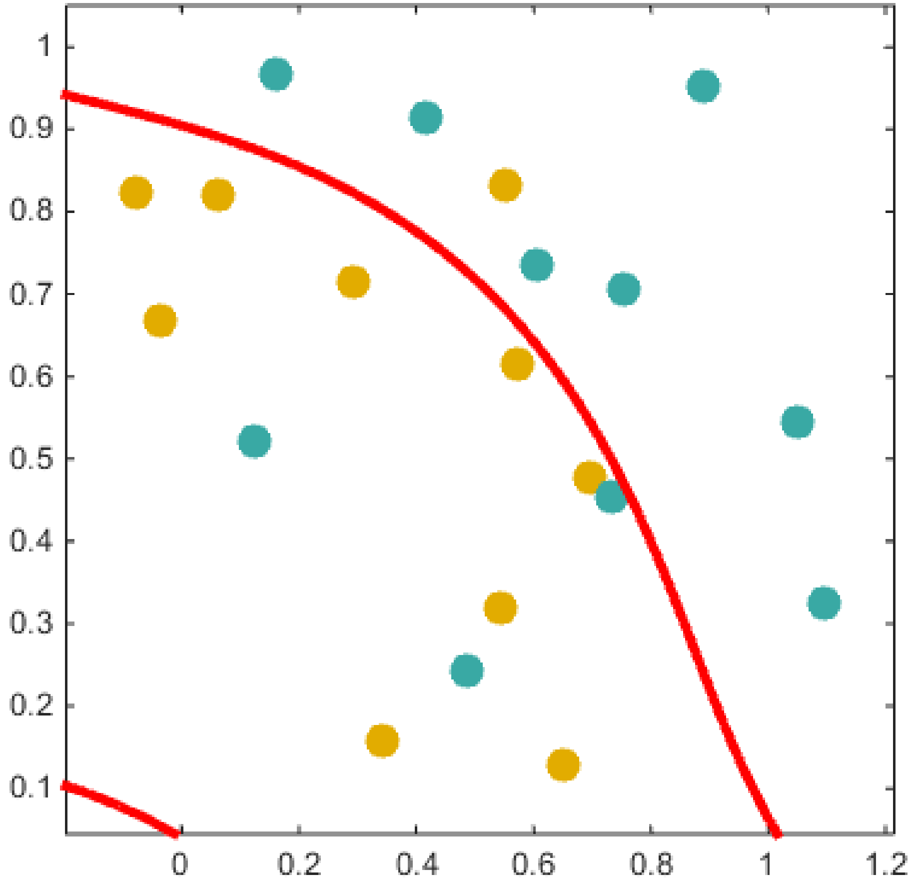
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 3.



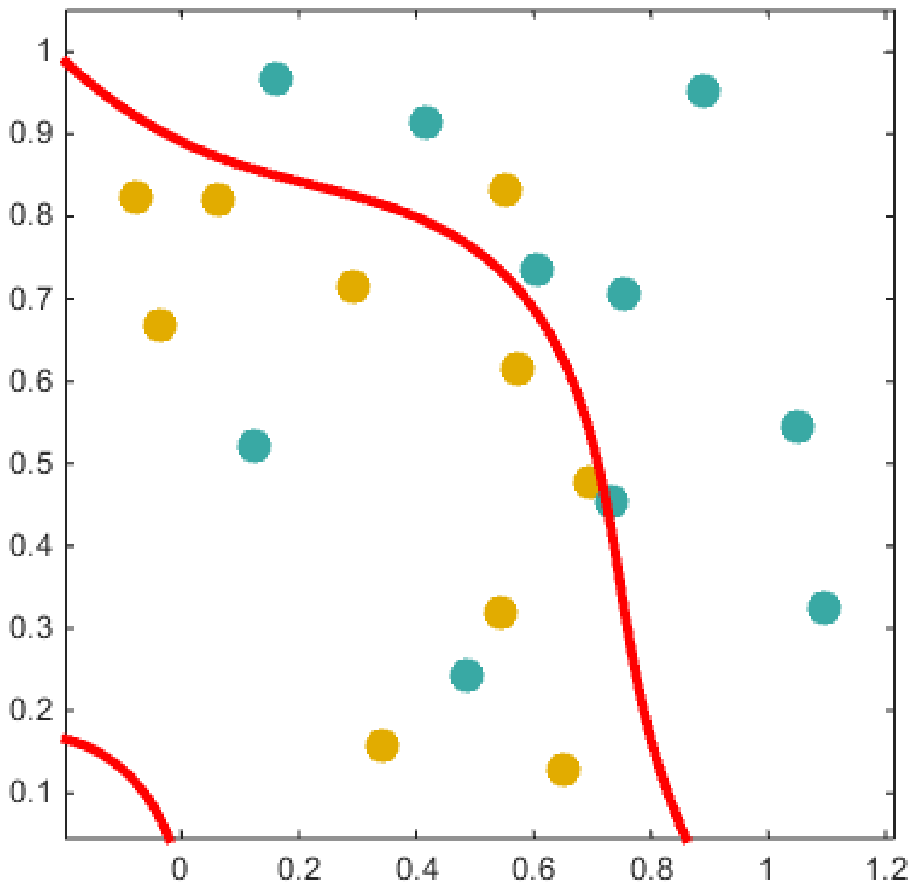
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 4.



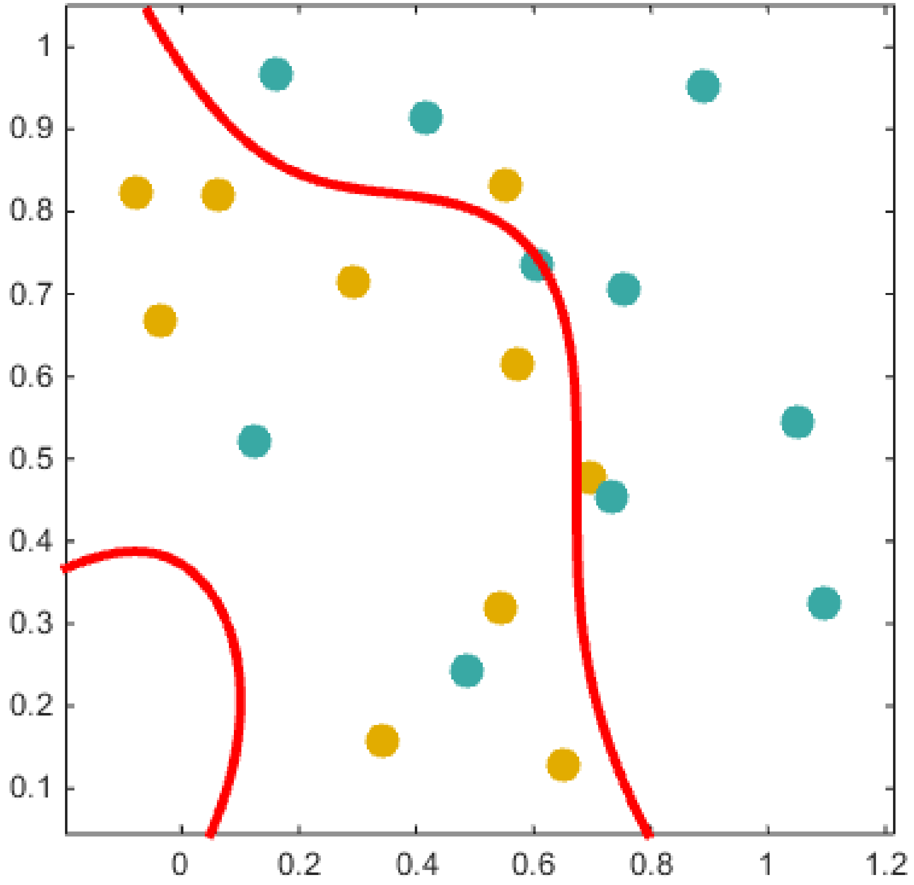
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 5.



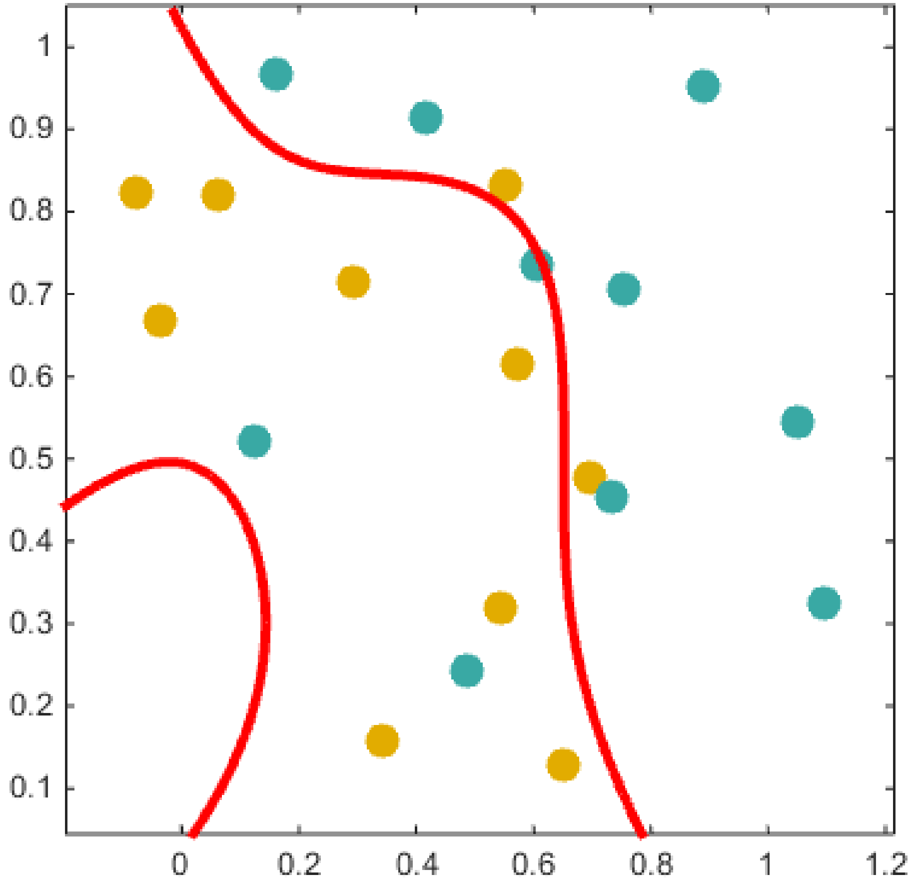
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 6.



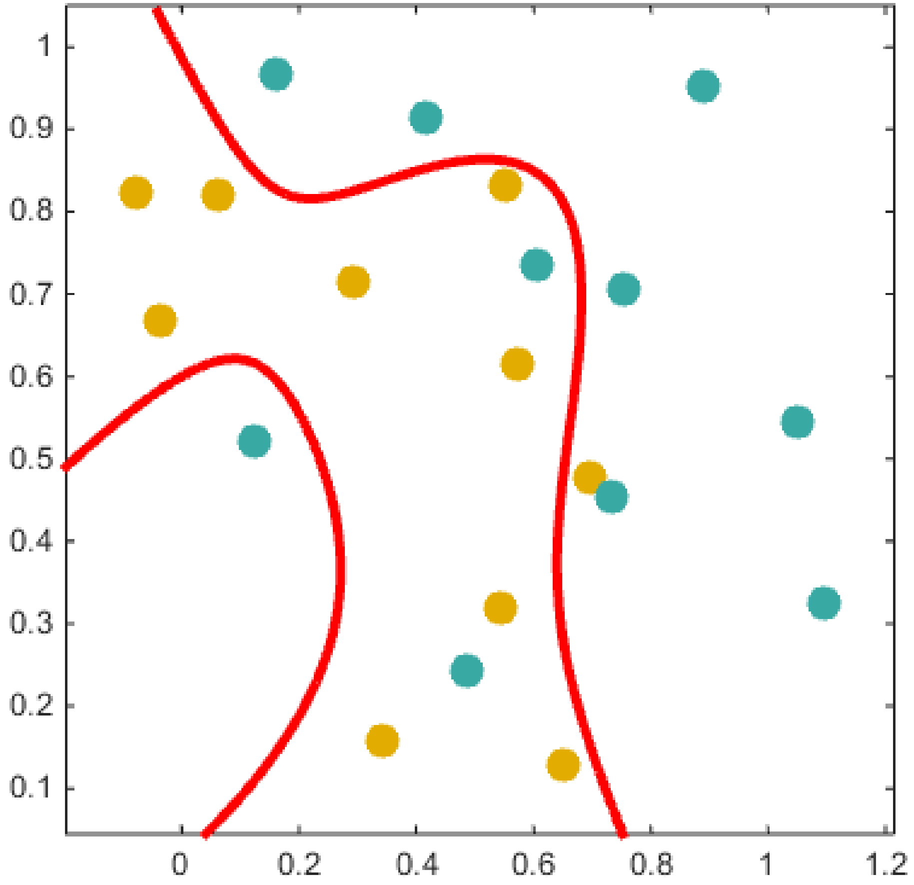
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 7.



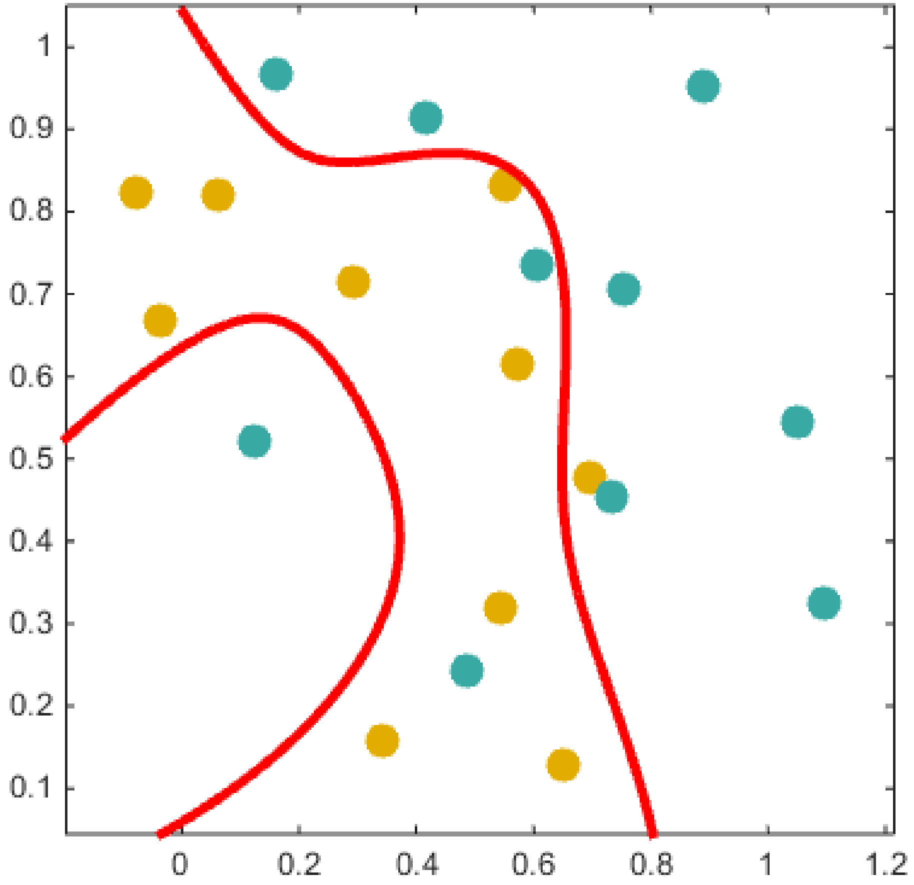
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 8.



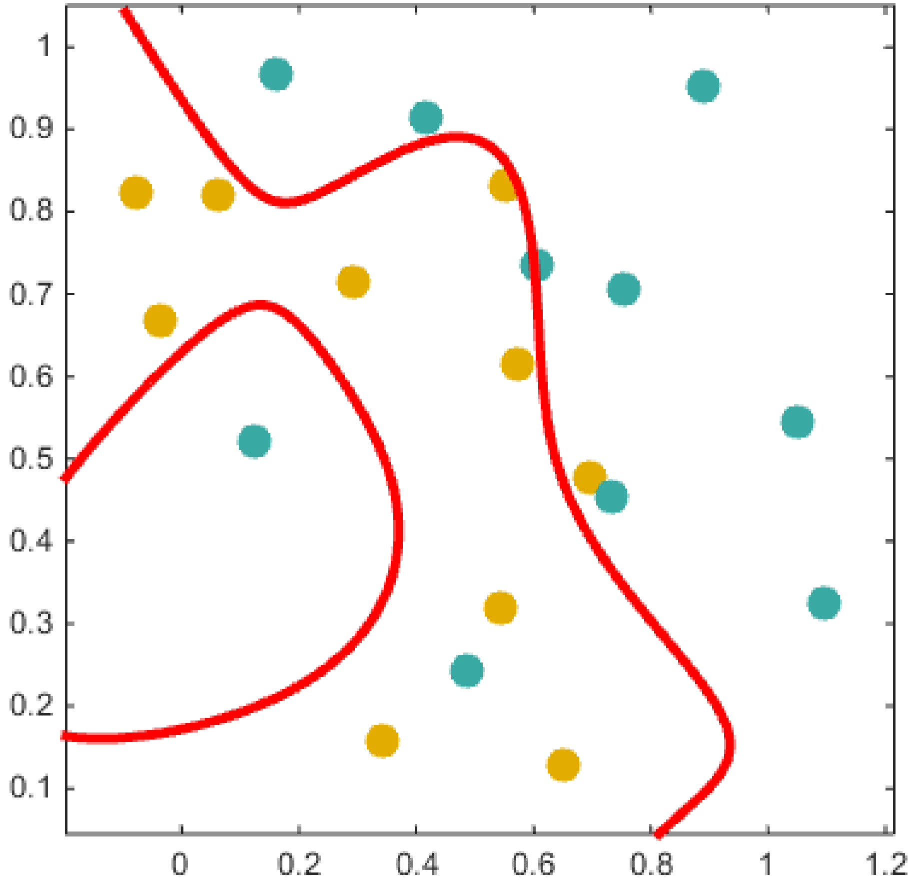
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 9.



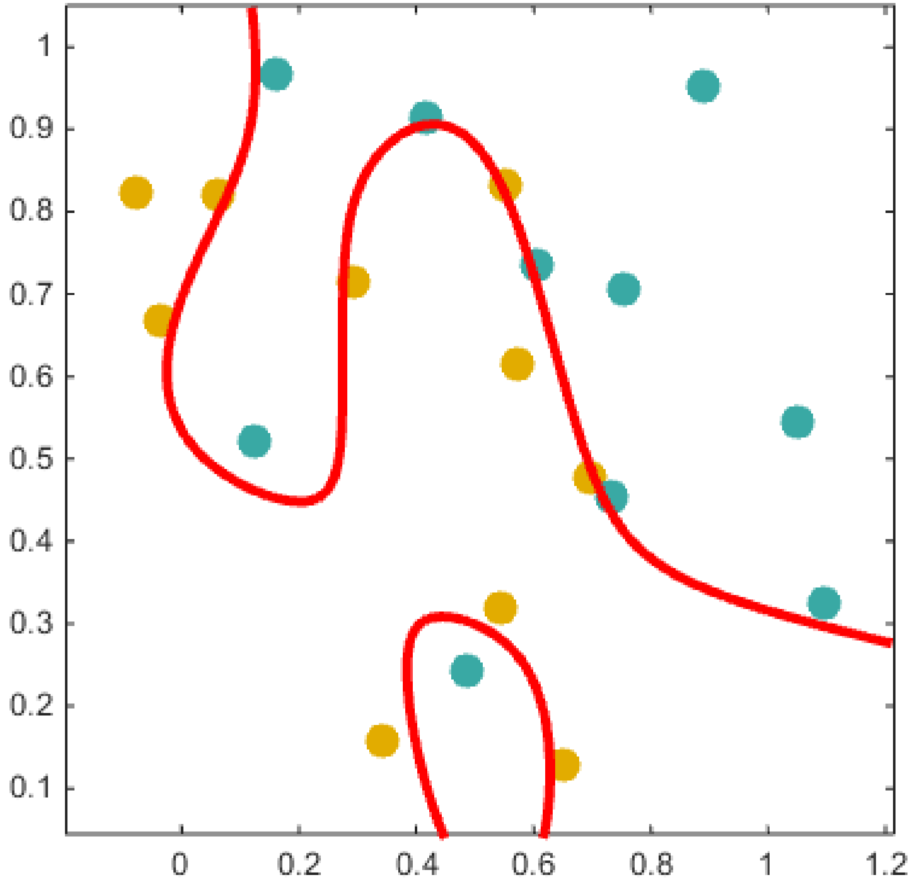
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 10.



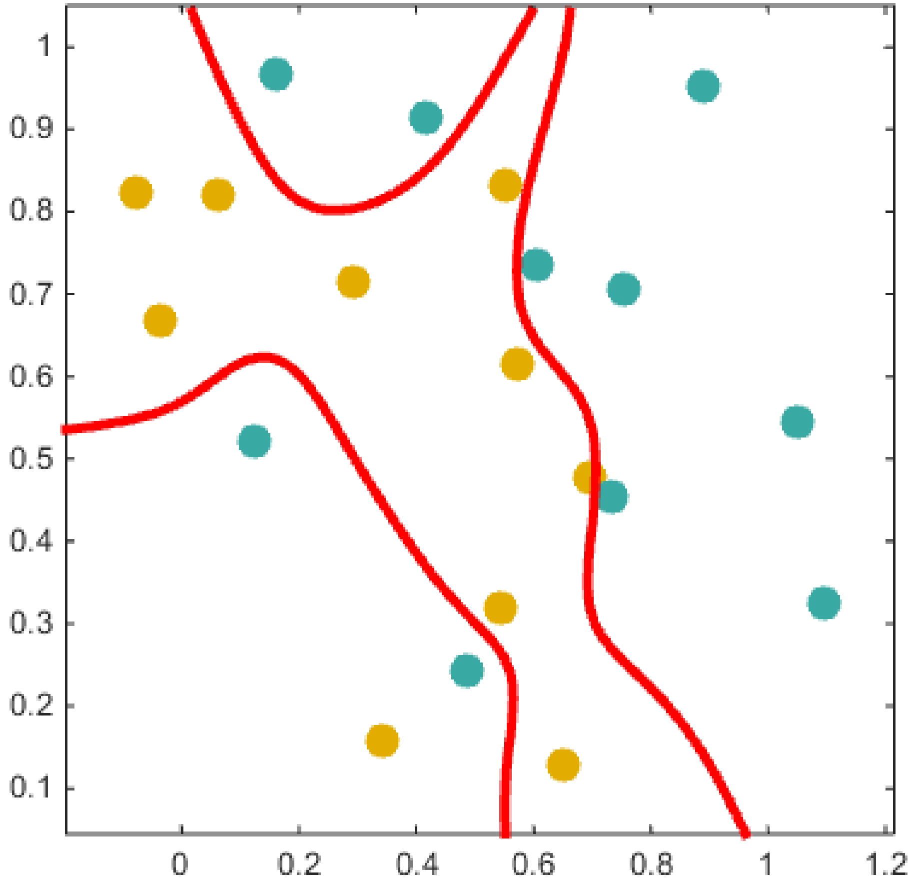
Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 20.



Polynomial Kernels – A Harder Case

Decision boundary with polynomial kernel of degree 100.



RBF/Gaussian Kernels

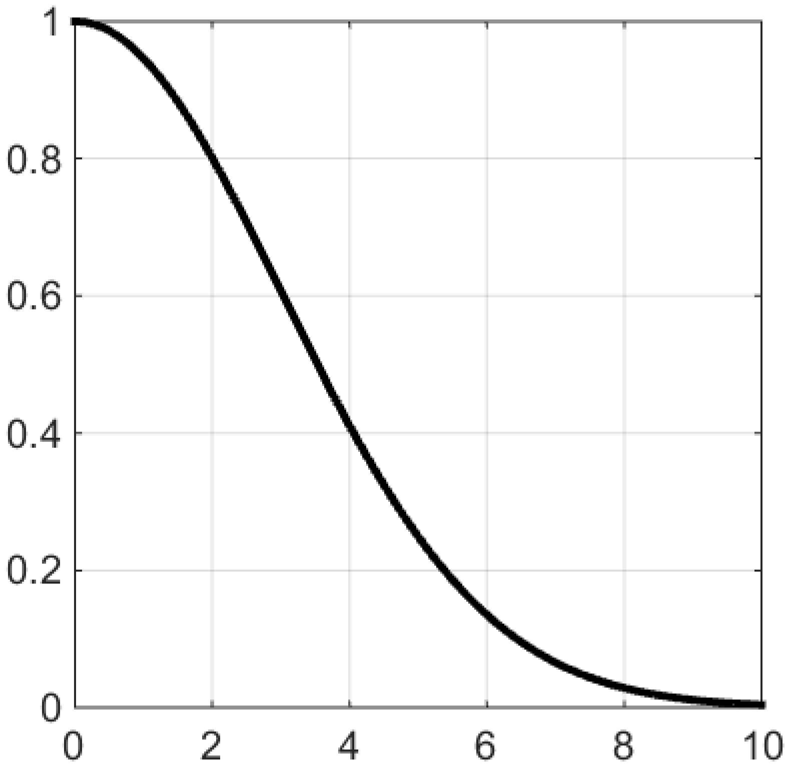
- The **Radial Basis Function (RBF) kernel**, also known as **Gaussian kernel**, is defined as:

$$k_\sigma(\boldsymbol{x}, \boldsymbol{z}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2}}$$

- Given σ, the value of $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$ only depends on the distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
  - $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$ decreases exponentially to the distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Parameter σ is chosen manually.
  - Parameter σ specifies how fast $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$ decreases as $\boldsymbol{x}$ moves away from $\boldsymbol{z}$.

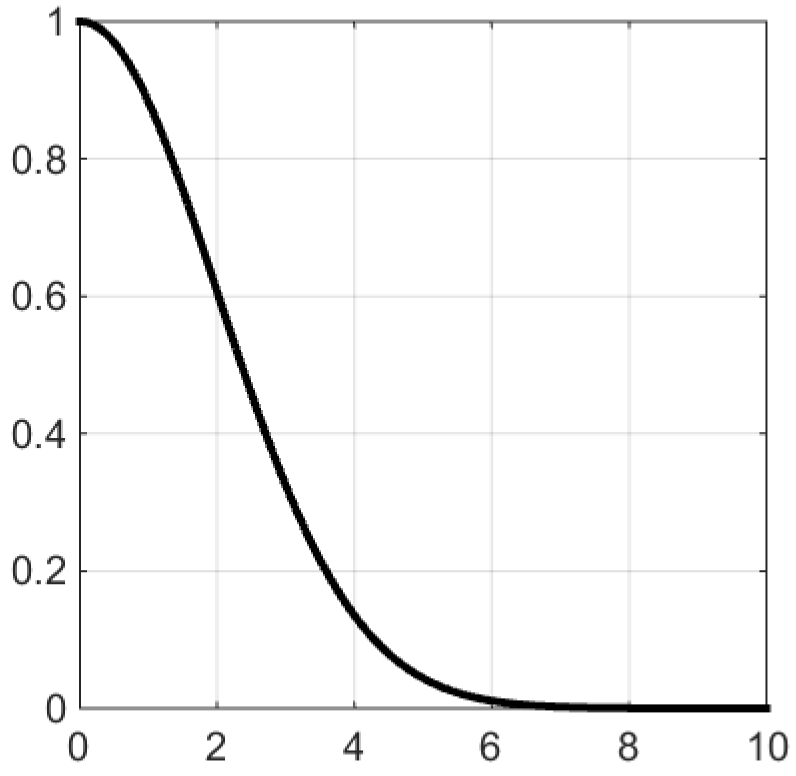
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$, with $\sigma = 3$.



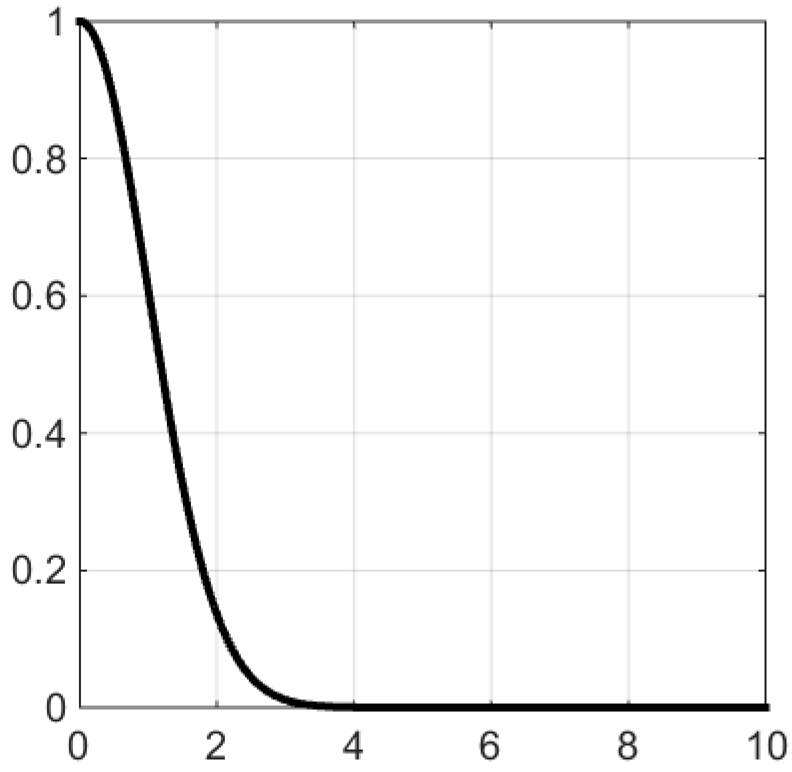
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$, with $\sigma = 2$.



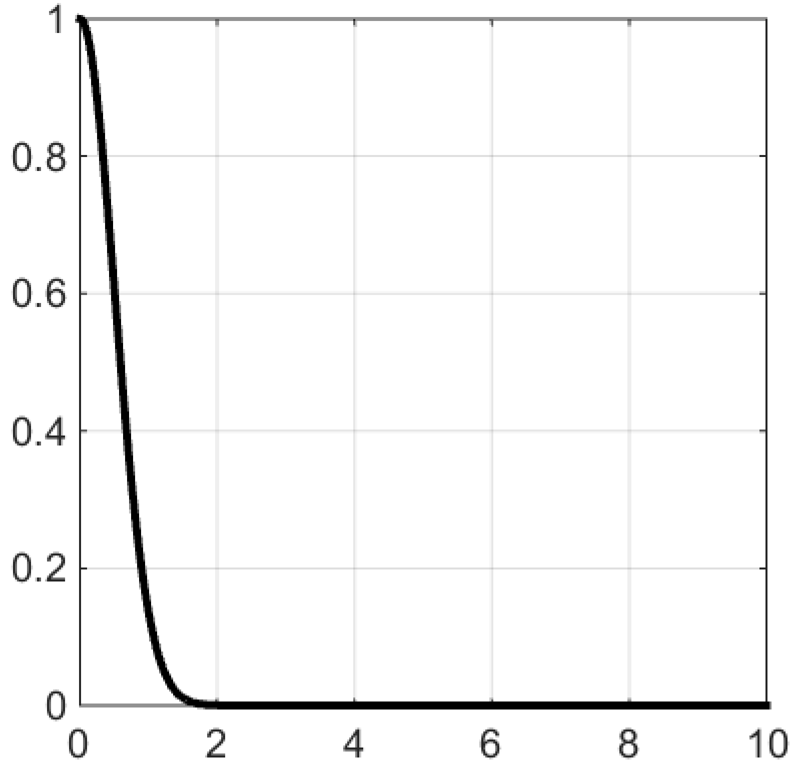
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$, with σ =1.



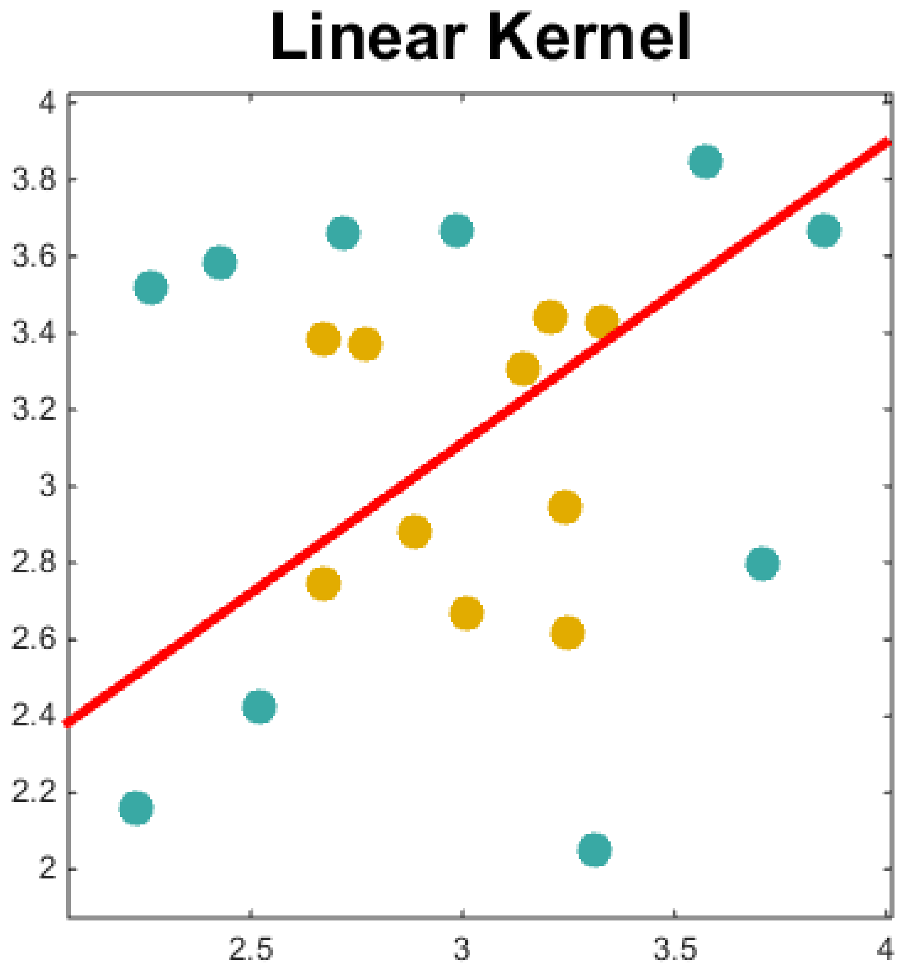
RBF Output Vs. Distance

- X axis: distance between $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Y axis: $k_\sigma(\boldsymbol{x}, \boldsymbol{z})$, with $\sigma = 0.5$.



RBF Kernels – An Easier Dataset

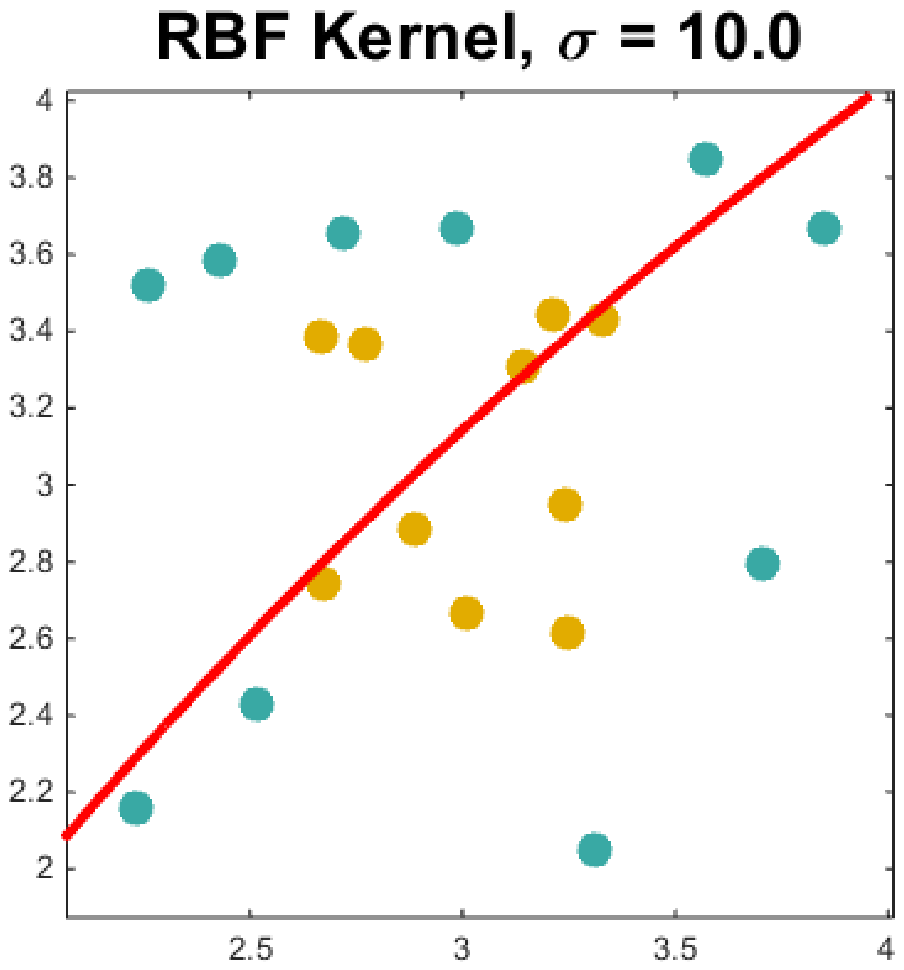
Decision boundary with a linear kernel.



RBF Kernels – An Easier Dataset

Decision boundary
with an RBF kernel.

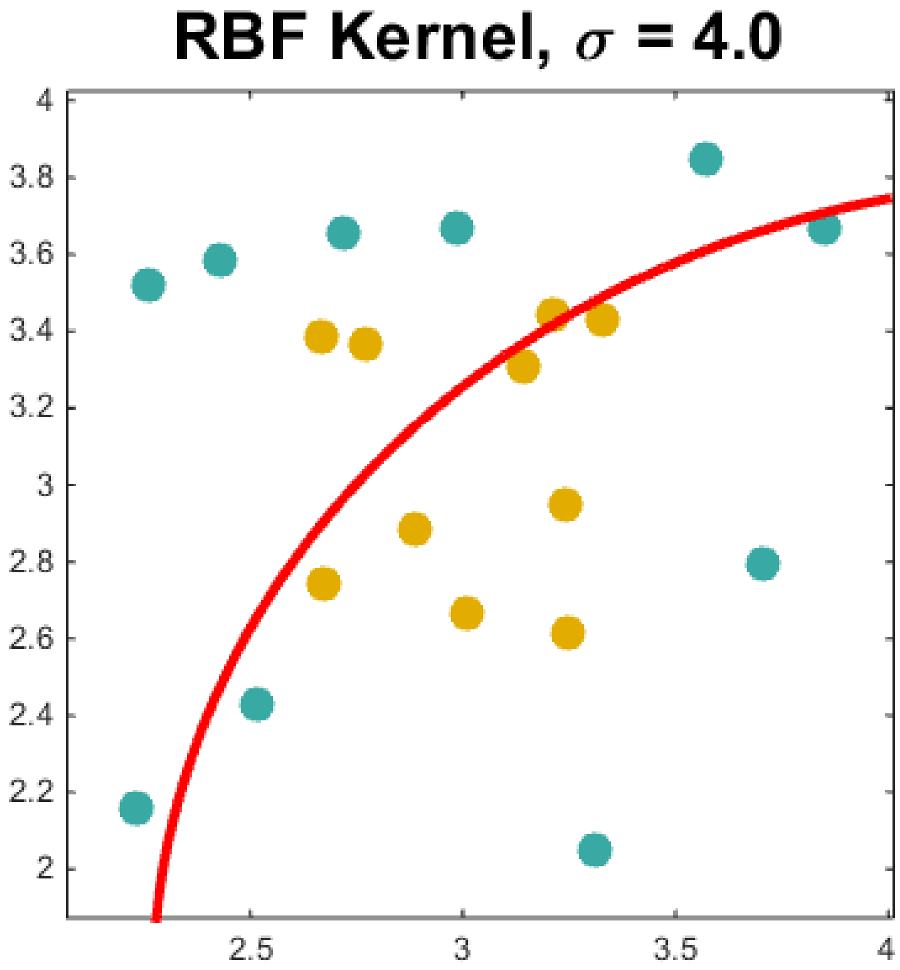
For this dataset, this is a relatively large value for σ, and it produces a boundary that is almost linear.



RBF Kernels – An Easier Dataset

Decision boundary
with an RBF kernel.

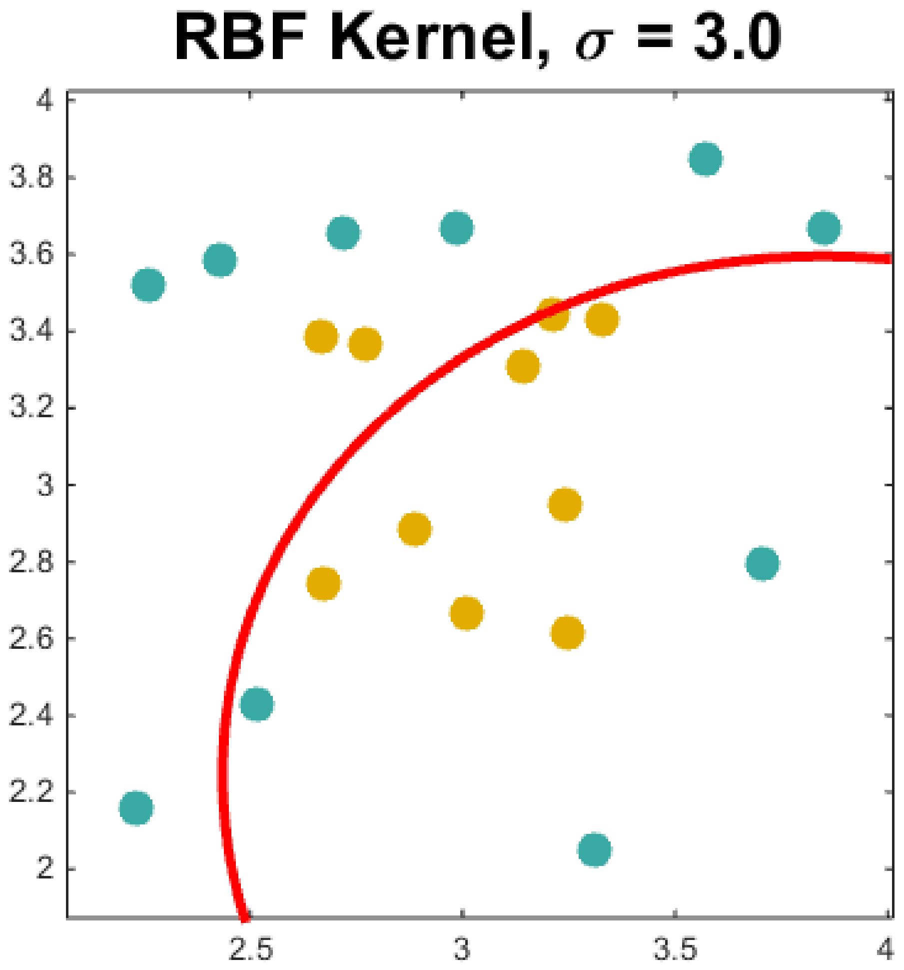
Decreasing the value of σ leads
to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary
with an RBF kernel.

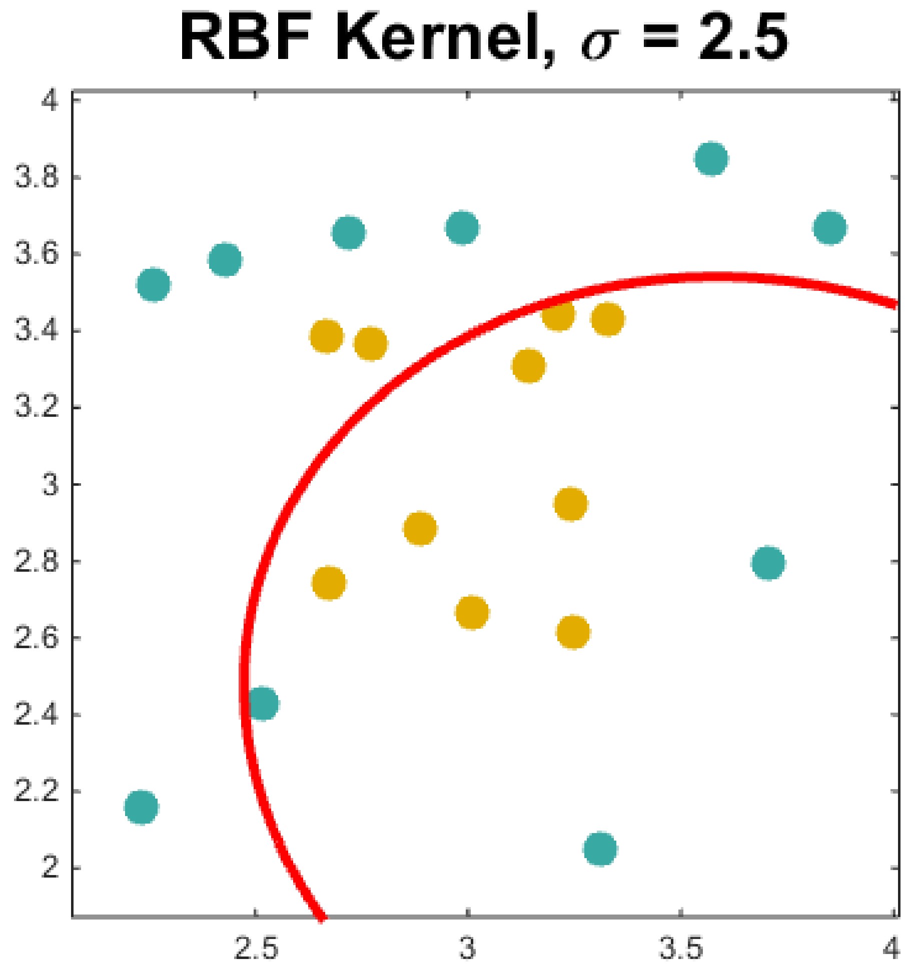
Decreasing the value of σ leads
to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary with an RBF kernel.

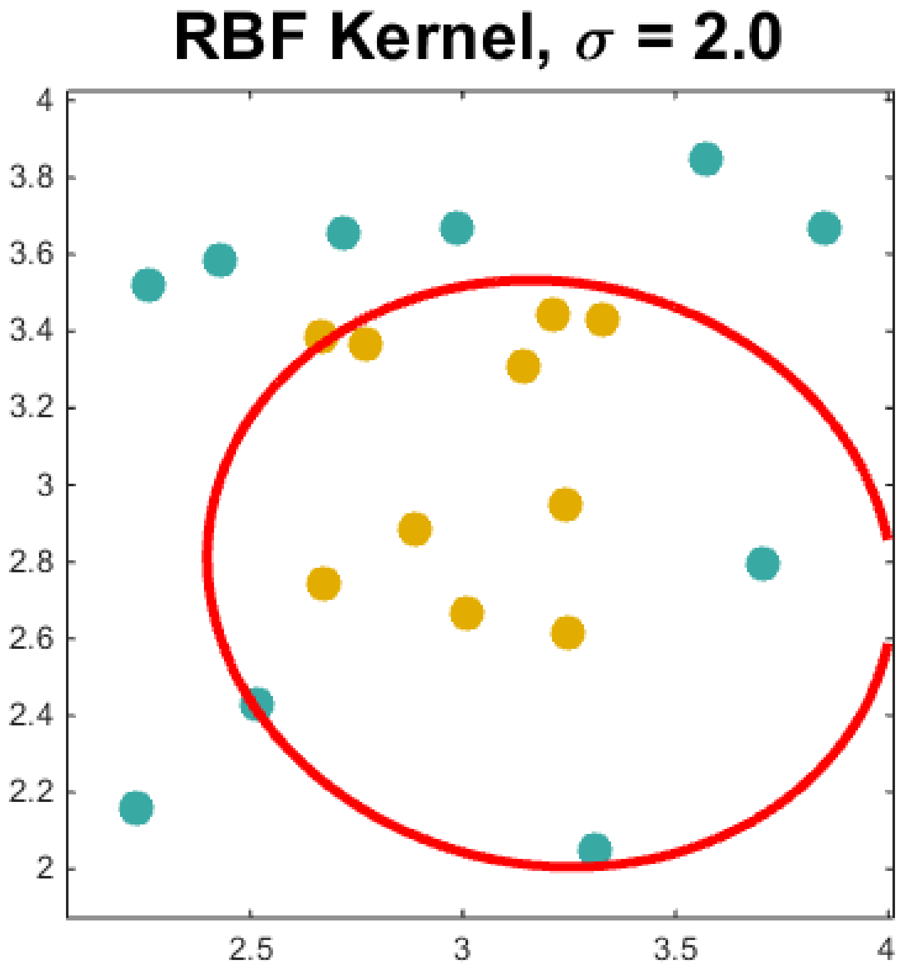
Decreasing the value of σ leads to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary
with an RBF kernel.

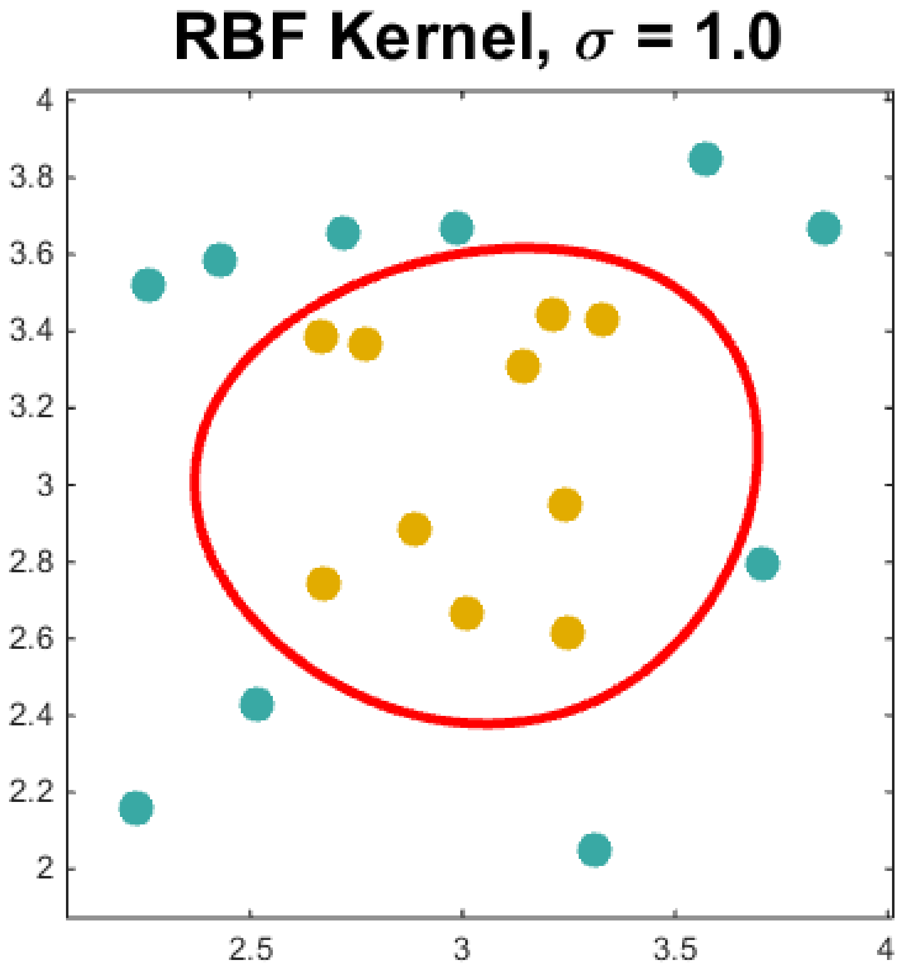
Decreasing the value of σ leads
to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary
with an RBF kernel.

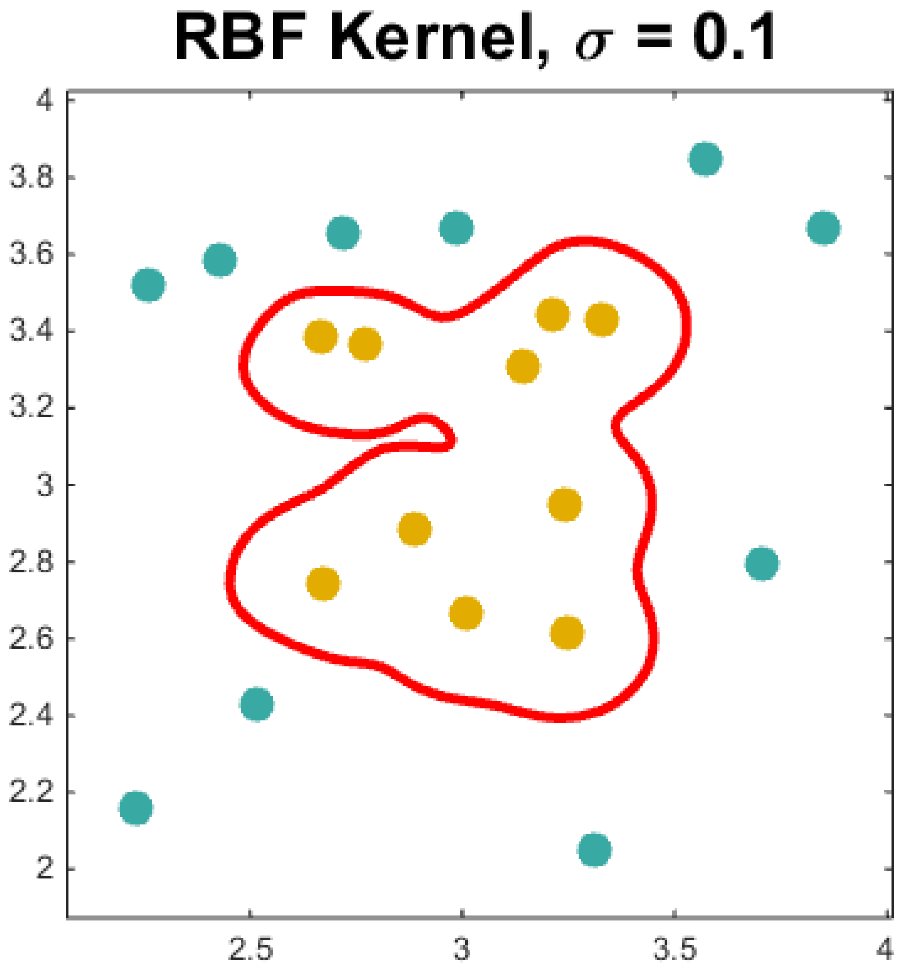
Decreasing the value of σ leads
to less linear boundaries.



RBF Kernels – An Easier Dataset

Decision boundary
with an RBF kernel.

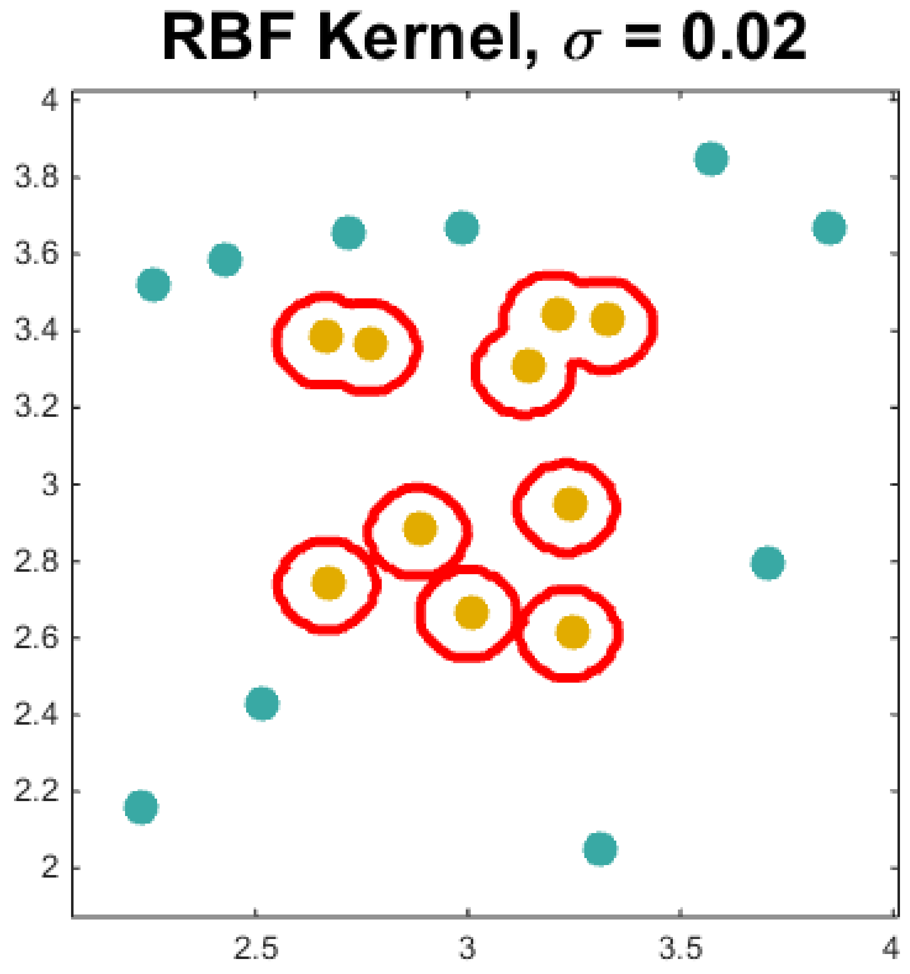
Note that smaller
values of σ increase dangers of
overfitting.



RBF Kernels – An Easier Dataset

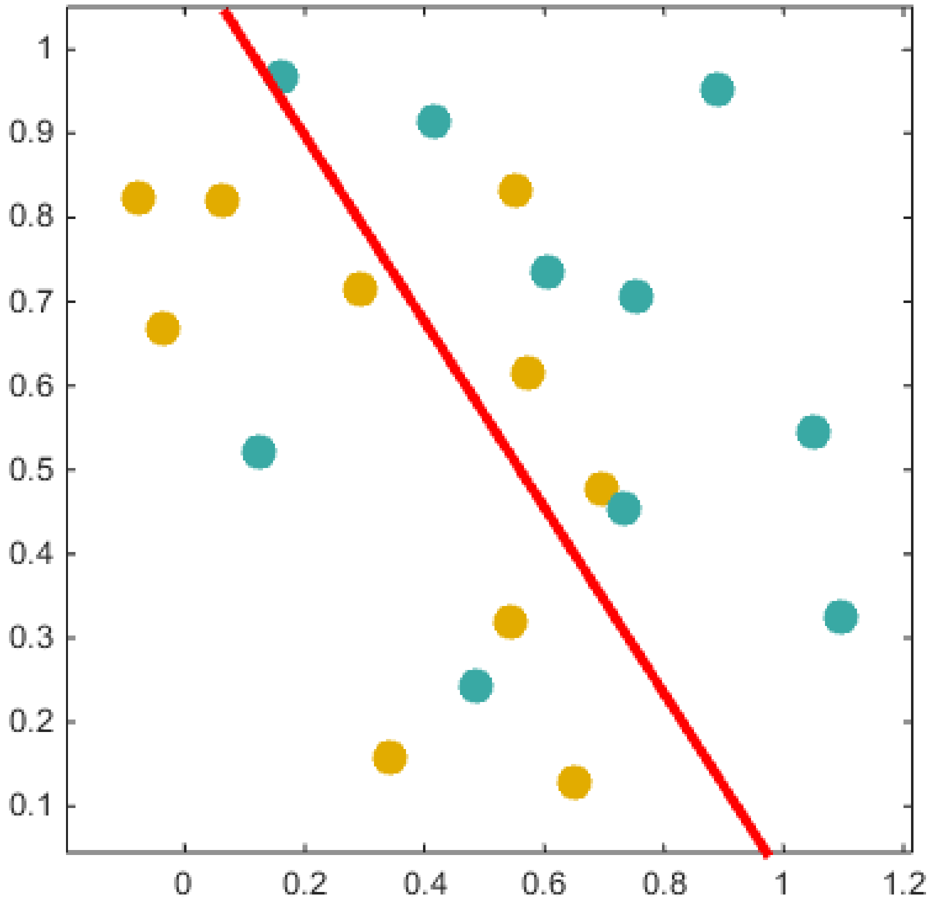
Decision boundary
with an RBF kernel.

Note that smaller
values of σ increase dangers of
overfitting.



RBF Kernels – A Harder Dataset

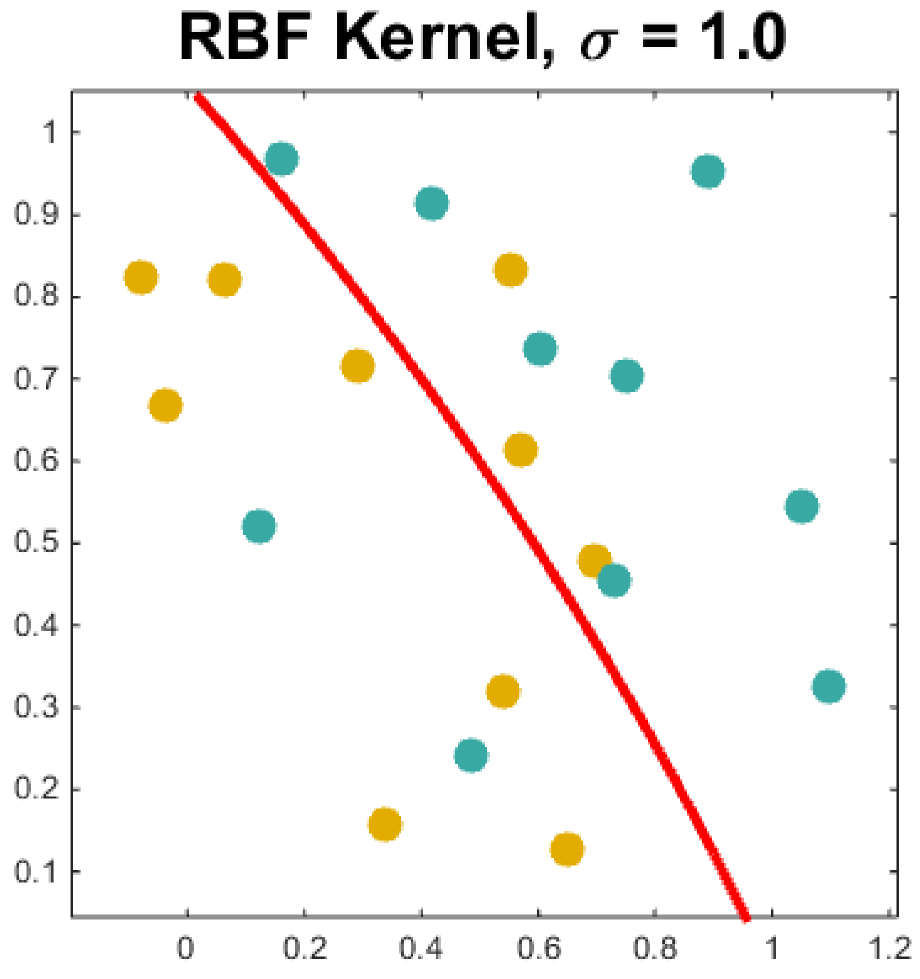
Decision boundary with a linear kernel.



RBF Kernels – A Harder Dataset

Decision boundary with an RBF kernel.

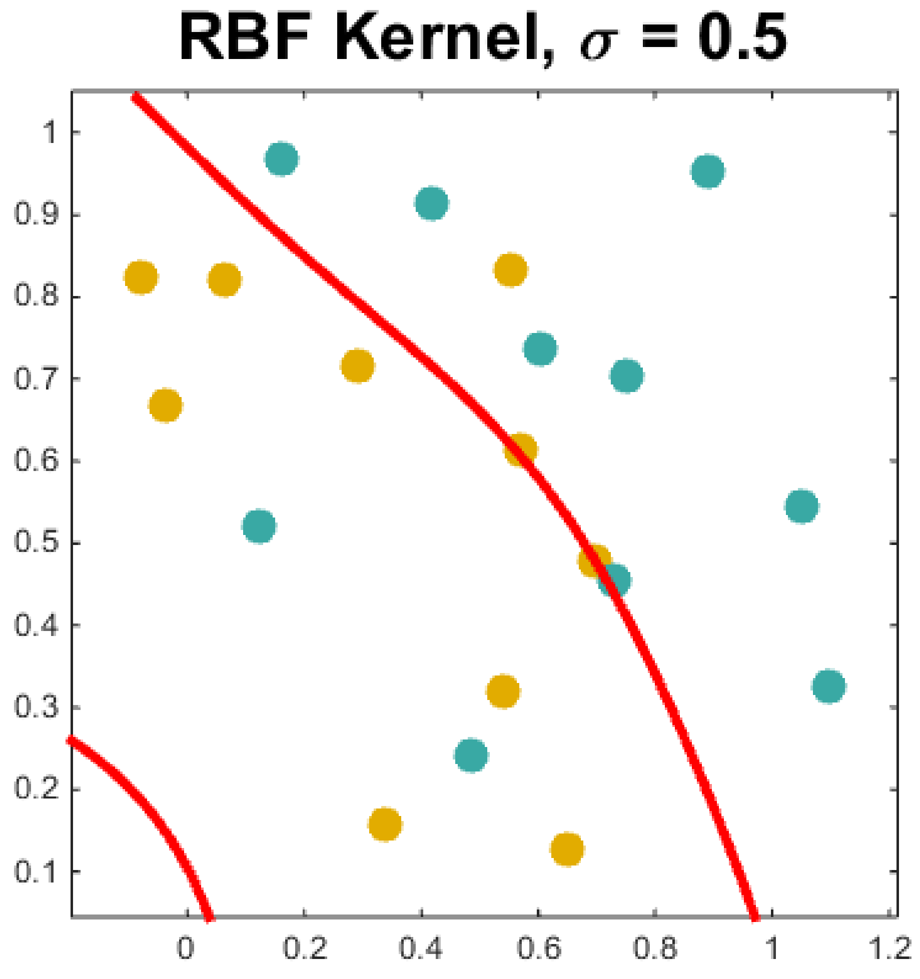
The boundary is almost linear.



RBF Kernels – A Harder Dataset

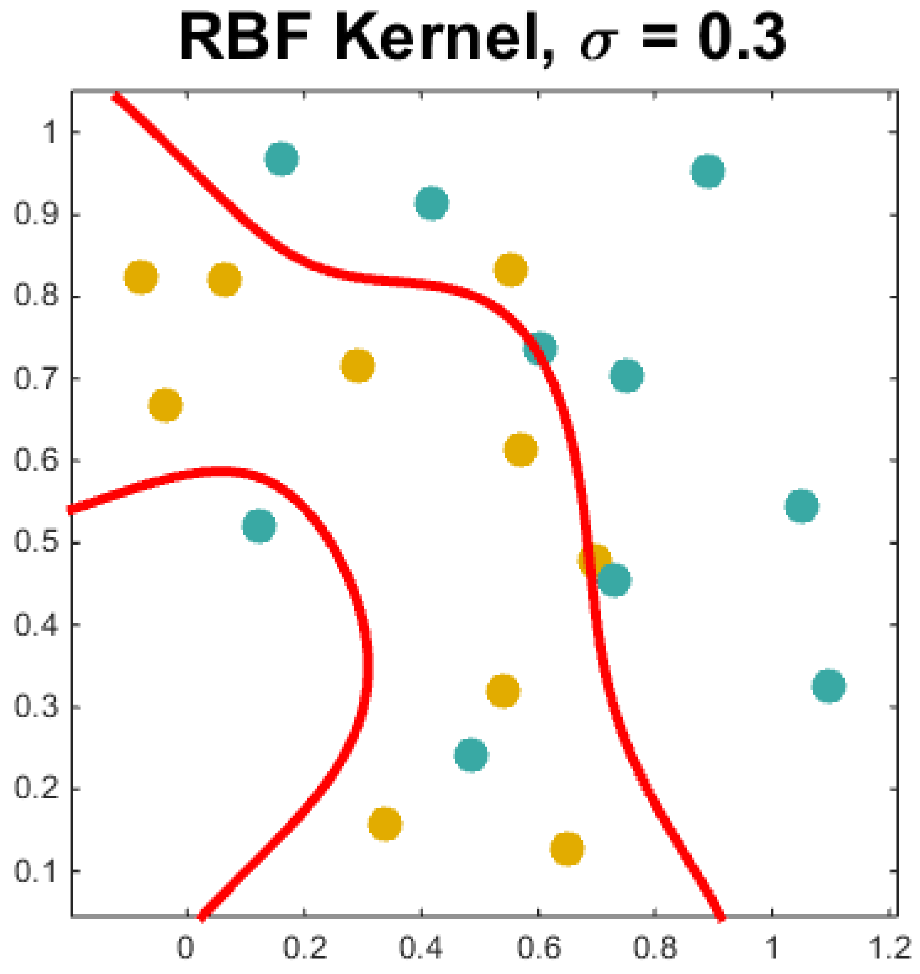
Decision boundary
with an RBF kernel.

The boundary now
is clearly nonlinear.



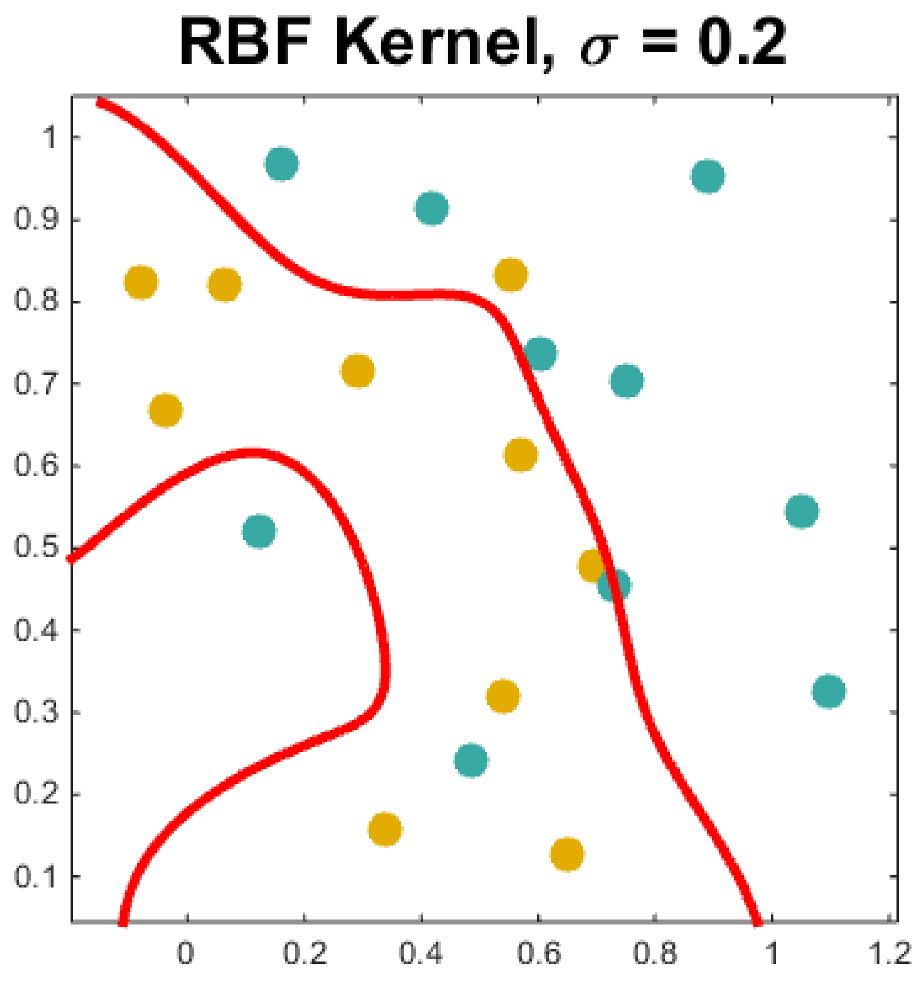
RBF Kernels – A Harder Dataset

Decision boundary with an RBF kernel.



RBF Kernels – A Harder Dataset

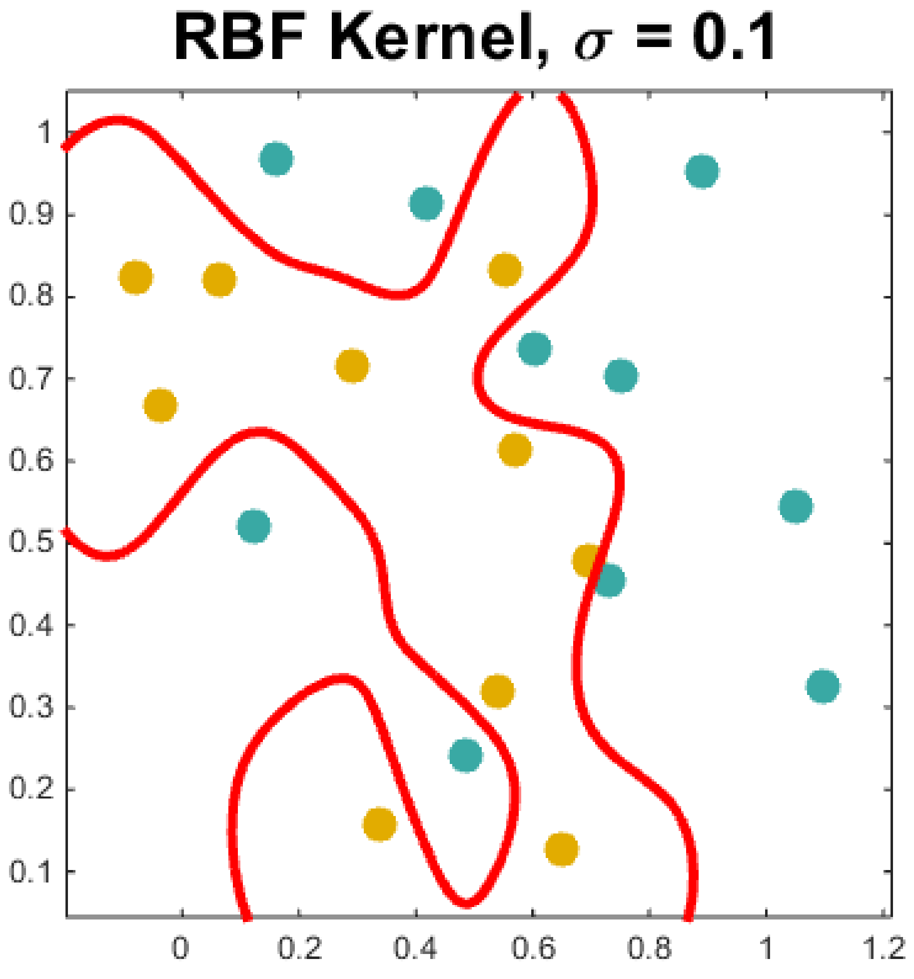
Decision boundary with an RBF kernel.



RBF Kernels – A Harder Dataset

Decision boundary
with an RBF kernel.

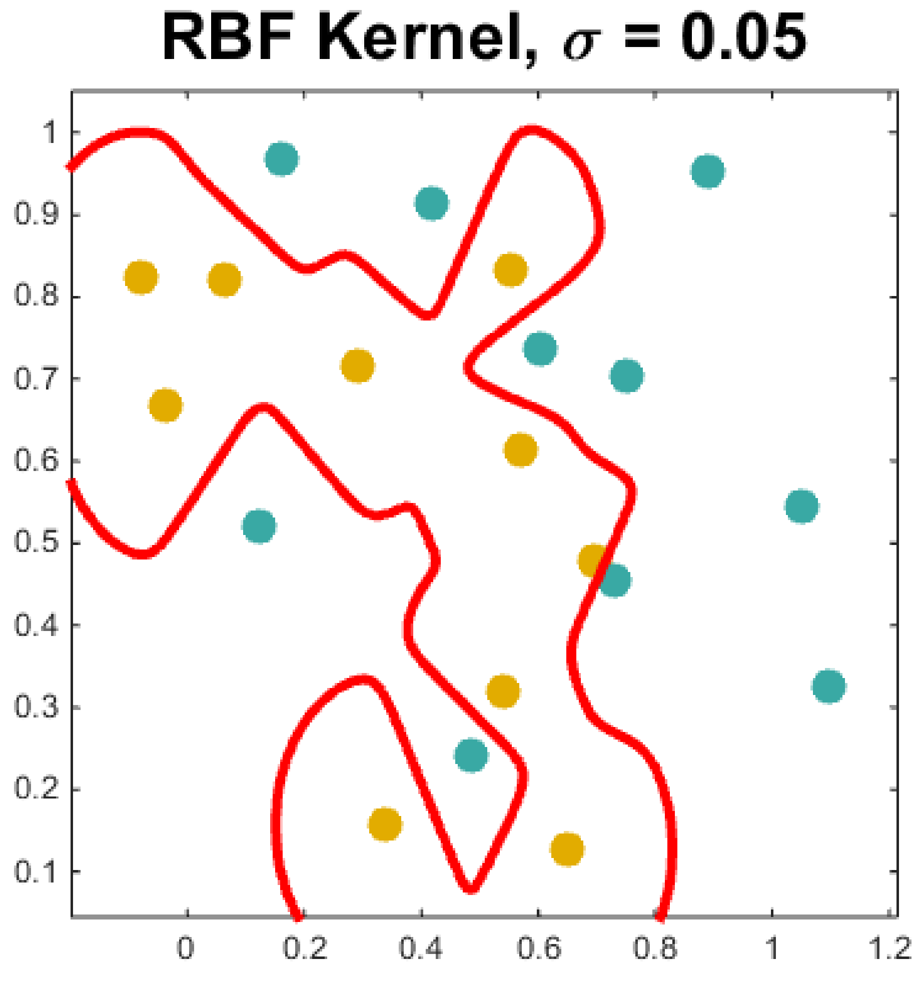
Again, smaller values
of σ increase dangers of overfitting.



RBF Kernels – A Harder Dataset

Decision boundary
with an RBF kernel.

Again, smaller values
of σ increase dangers of overfitting.



SVM parameters

SVM has another set of parameters called <u>hyperparameters</u>.

- The soft margin constant *C*.
- Any parameters the kernel function depends on
  - linear kernel – no hyperparameter (except for *C*)
  - polynomial – degree
  - Gaussian – width of Gaussian

- So which kernel and which parameters should I use?
- The answer is data-dependent.
- Several kernels should be tried.
- Try linear kernel first and then see, if the classification can be improved with nonlinear kernels (tradeoff between quality of the kernel and the number of dimensions).
- Select kernel + parameters + *C* by crossvalidation.

Practical Aspects

Many beginners use the following procedure:

- Transform data to the format of an SVM software (often obsolete)
- Randomly try a few kernels and parameters
- Test

Instead try

- Conduct simple data scaling/normalization
- Consider the RBF kernel first
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and γ to test model

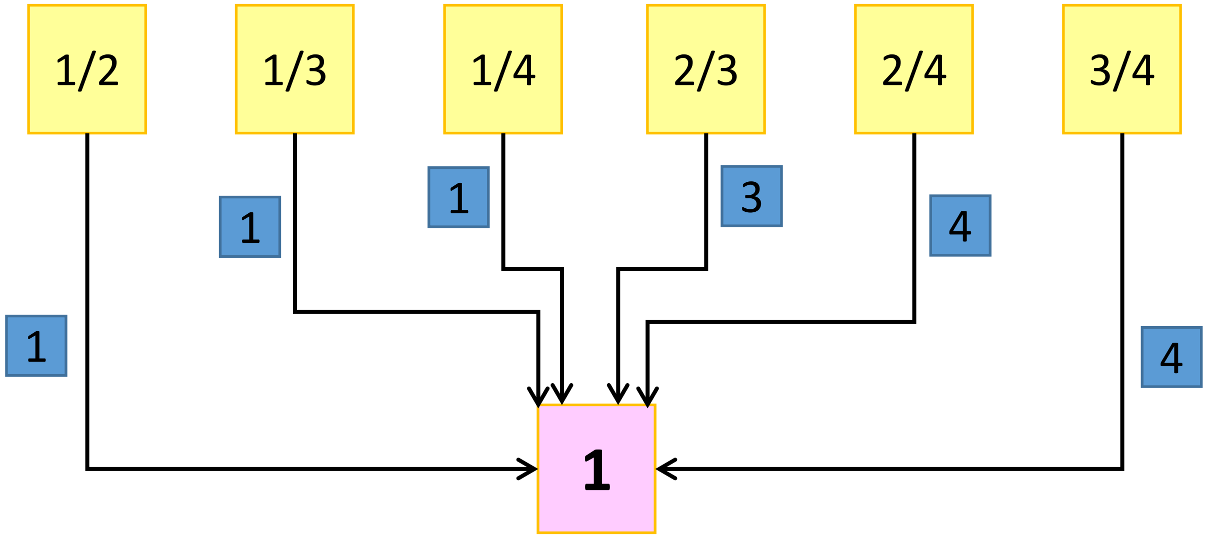
Computational aspects

- Classification of new samples is very quick, training is longer (reasonably fast for thousands of samples).
- Linear kernel – scales linearly.
- Nonlinear kernels – scale quadratically.

Multiclass SVM

- SVM is defined for binary classification.
- How to predict more than two classes (multiclass)?
- Simplest approach: decompose the multiclass problem into several binary problems and train several binary SVM's.

- one-versus-one approach
  - Train a binary SVM for any two classes from the training set
  - For k-class problem create $\frac{k(k-1)}{2}$ SVM models
  - Prediction: voting procedure assigns the class to be the class with the maximum votes



- one-versus-all approach
  - For *k*-class problem train only *k* SVM models.
  - Each will be trained to predict one class (+1) vs. the rest of classes (-1)
  - Prediction:
    - Winner takes all strategy
    - Assign new example to the class with the largest output value $f(\boldsymbol{x})$.

1/rest 2/rest 3/rest 4/rest

SVMs: Recap

- Advantages:
  - Training finds globally best solution.
    - No need to worry about local optima, or iterations.
  - SVMs can define complex decision boundaries.
- Disadvantages:
  - Training time is cubic to the number of training data. This makes it hard to apply SVMs to large datasets.
  - High-dimensional kernels increase the risk of overfitting.
    - Usually larger training sets help reduce overfitting, but SVMs cannot be applied to large training sets due to cubic time complexity.
  - Some choices must still be made manually.
    - Choice of kernel function.
    - Choice of parameter C in formula $C\left(\sum_{n=1}^{N} \xi_n\right) + \frac{1}{2}\|\boldsymbol{w}\|^{\mathbf{2}}$.

References

- An excellent tutorial on VC-dimension and Support Vector Machines:

C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998. http://citeseer.nj.nec.com/burges98tutorial.html

- The VC/SRM/SVM Bible:

Statistical Learning Theory by Vladimir Vapnik, Wiley-Interscience; 1998