RBF Networks and K-Means Clustering

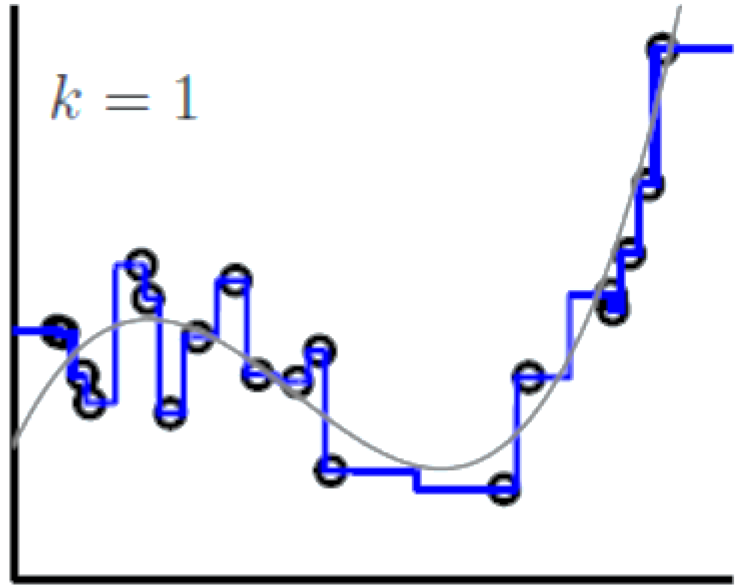
Anahita Zarei, Ph.D.

Outline

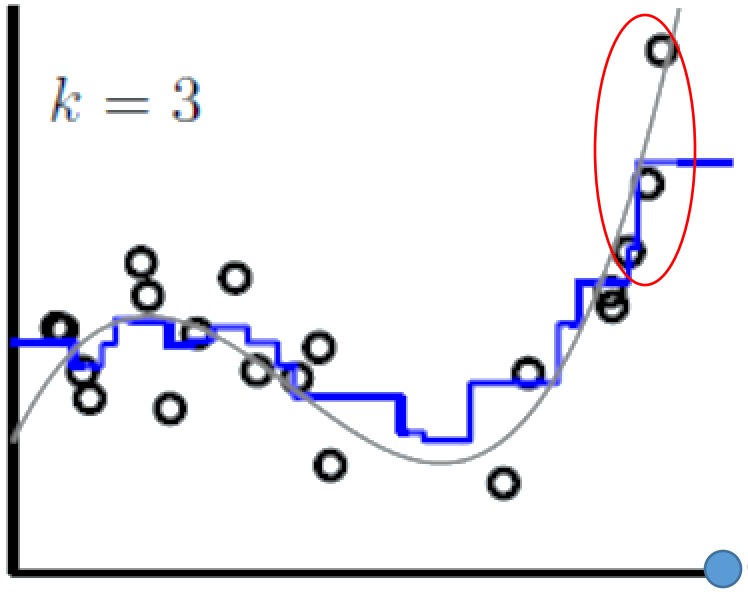
- Distance Weighted KNN -> RBF Nonparametric Regression -> RBF Parametric Regression (i.e. Kernel Regression) -> RBF Networks
- Reading:
  - Abu-Mustafa: 6.3

Revisit K-Nearest Neighbor

- 1-Nearest Neighbor overfits the data
- Increasing K helps with overfitting problem.

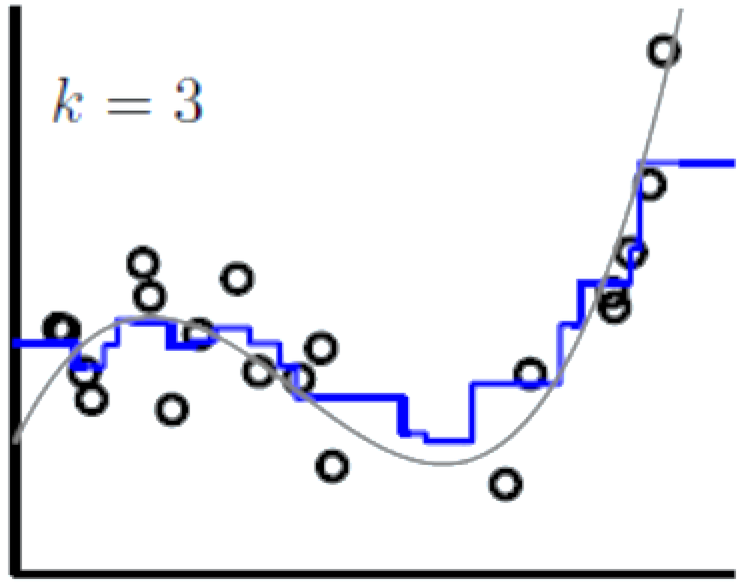


- But there's the boundary and sparse region problem.
- At the boundary, all nearest neighbors are the same k points.



KNN Discontinuity Problem

- A more important issue with KNN is the jumpy nature of the hypothesis.
- This is due to the fact that a nearest neighbor is either completely in or out of the window from one input to another.
- The discontinuity property can be problematic in practice.
    - e.g. price of 2000 sqft-house vs. 2001 sqft-house.
    - e.g. price of a car with 100K miles vs. a car with 101Kn miles



3-NN 1-d regression

A Remedy for KNN Discontinuity Problem

- Make the effect of farther neighbors less significant such that when a neighbor jumps in and out it doesn’t have as great of an impact.
- Weigh each of the K neighbors such that the farther they are the less influence they have.
- **Distance-Weighted KNN:**

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

- How shall we define the weights?
- Choose weights such that as the distance between the test point and its neighbors increases, the corresponding weight drops down to weigh down the impact of that observation:

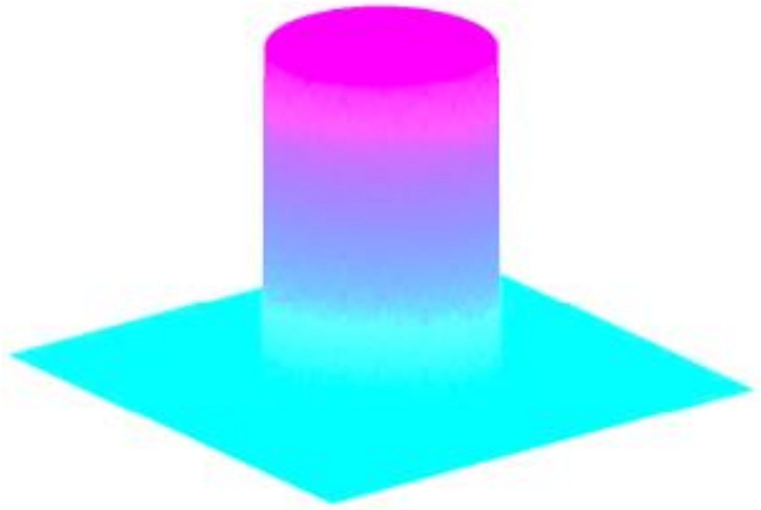
$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Expanding the Locally Weighted KNN to Kernel Regression

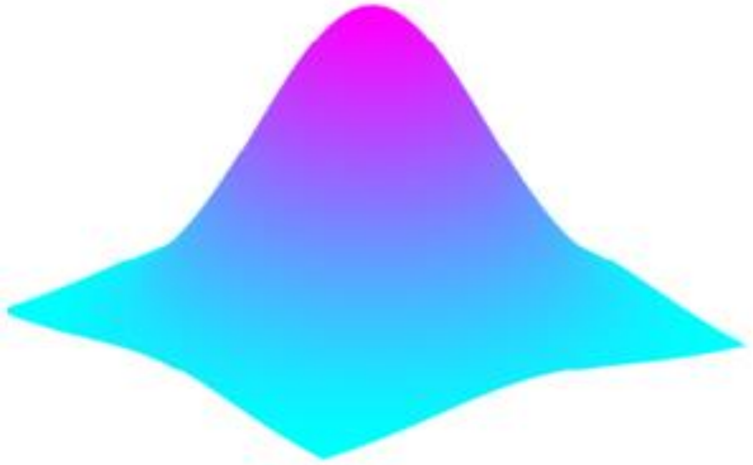
- In weighted KNN, only the K neighbors influence the prediction at a test point x.
- If we extend the impact of the training points beyond only the KNN and expand it to all points then we get kernel regression.

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{\overset{N}{\cancel{k}}} w_i f(x_i)}{\sum_{i=1}^{\overset{N}{\cancel{k}}} w_i}$$

- These weights, in general, are called kernels and are function of the distance between the training points and a query (test) point.



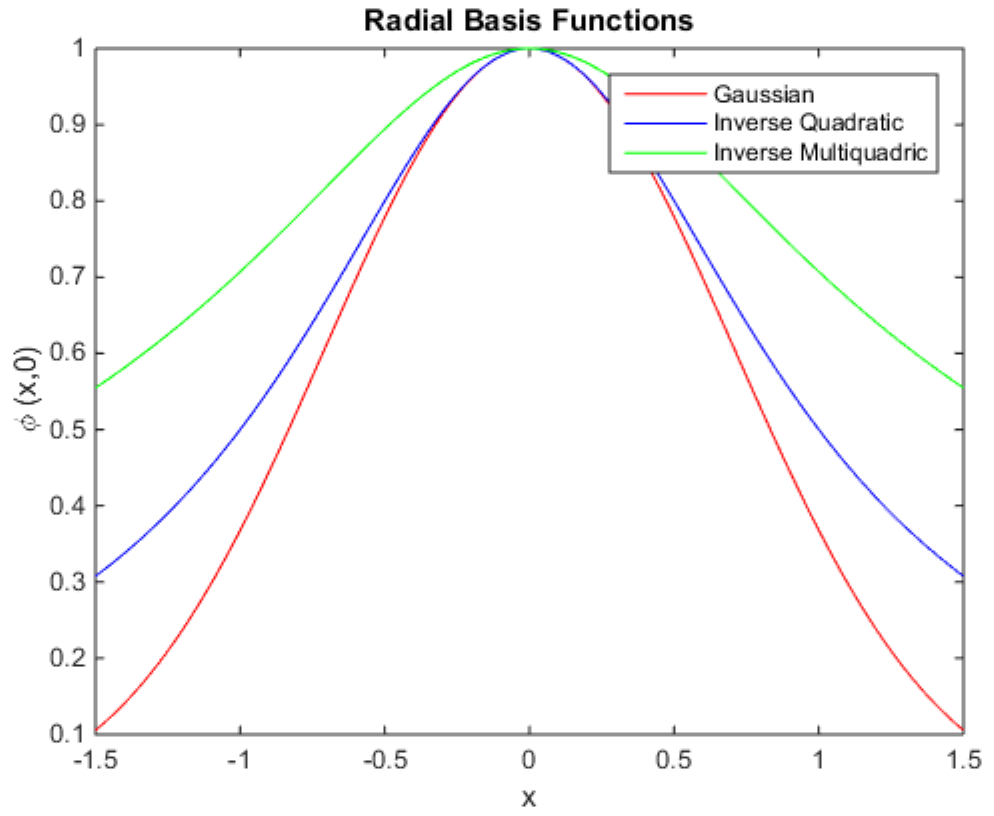
Only K neighbors influence the output.



Every training point influences the output.

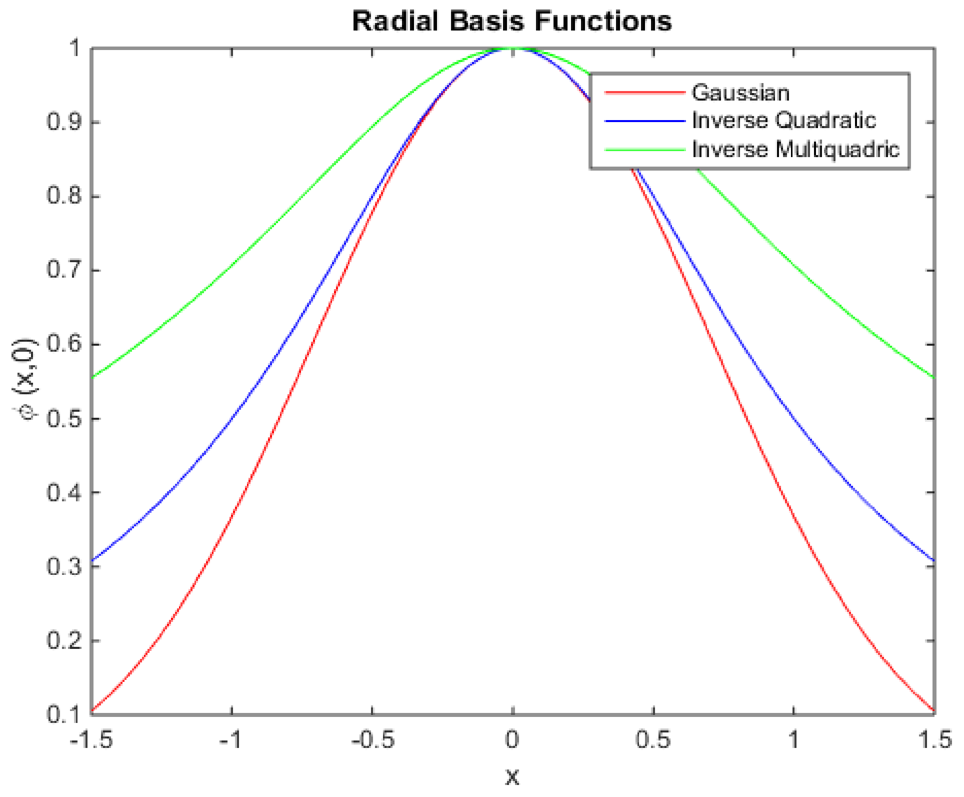
Kernels: Radial Basis Function

- A real-valued function whose value depends only on the distance (radius) from a given point c, called center. (i.e. $\phi(\mathbf{x}, \mathbf{c}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$)
- The norm is usually Euclidean distance, although other distance functions are also possible.
- The 'radial' in RBF reflects the fact that the influence only depends on the radial distance.
- Examples include:
  - Gaussian: $\varphi(x,c) = e^{-\gamma||x-c||^2 = \frac{-||x-c||^2}{r}}$
  - Inverse Quadratic: $\varphi(x,c) = \frac{1}{1+\gamma||x-c||^2}$
  - Inverse Multiquadratic: : $\varphi(x,c) = \frac{1}{\sqrt{1+\gamma||x-c||^2}}$
- The most common RBF is Gaussian.



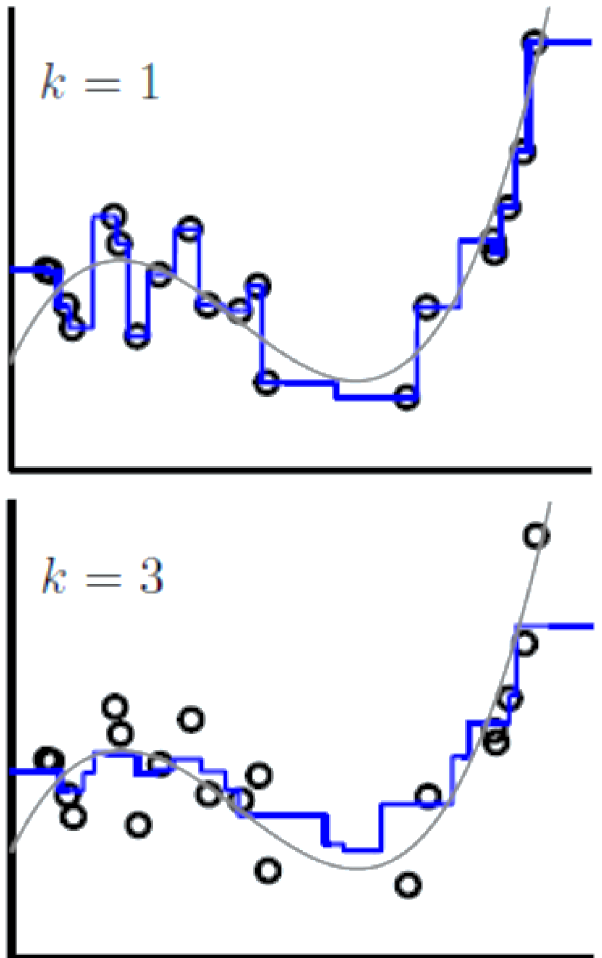
RBF Properties $\varphi(x,c) = e^{\frac{-||x-c||^2}{r}}$

- Positive and non-increasing (i.e. weights never go exactly to zero, they just become very small as the distance increases.)
- r defines the width of the kernel (i.e. how quickly it decays)
- r determines the 'unit' of length against which distances are measured.
- If distance is small relative to r, then the neighbor has a significant influence.

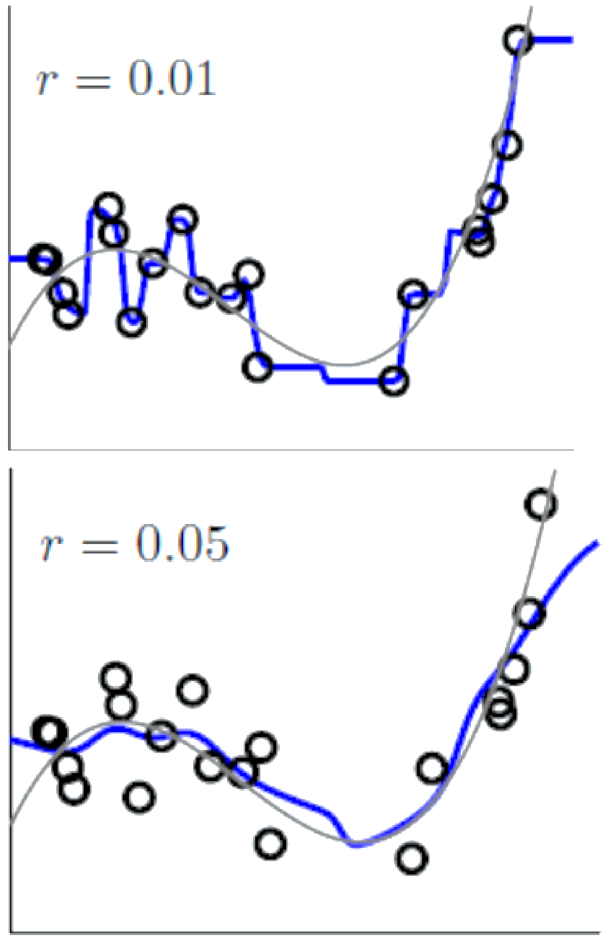


Plots: KNN vs. (nonparametric) RBF

KNN



RBF

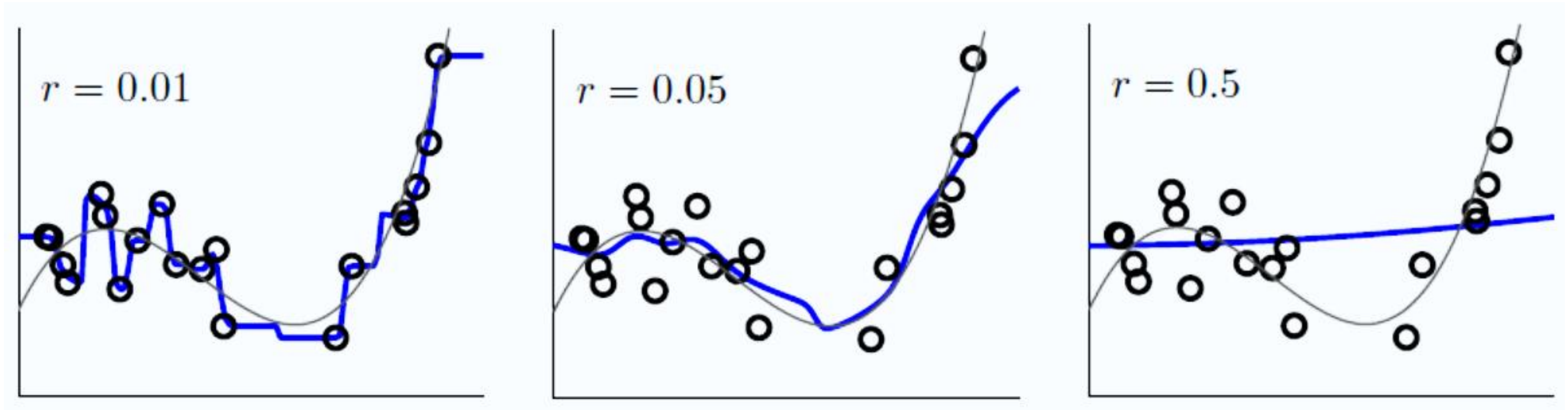


Nonparametric RBF

1. Smoother version of KNN
2. Can be easily interpreted to the client
3. Like KNN, computationally expensive.

Effect of r $(= {}^{1}/_{\gamma})$

- Smaller r places more emphasis on the nearest points.
- As r gets large, the kernel width gets larger and more of the data points contribute. As a result, the final hypothesis gets smoother.
- **Too small an r** results in a complex hypothesis that **overfits**. **Too large an r** results in an excessively smooth hypothesis that **underfits.**



How to Choose r?

- Cross-validation.
- Using heuristics, a good starting value is $\frac{1}{2\sqrt[d]{N}}$
  - Where d is the dimension of the data and N is the number of training points.
- Recall that a good starting value for K in KNN was $\mathrm{K} = \sqrt{N}$
- $\hat{f}(x) = \sum_{n=1}^{N} \frac{\alpha_n(x)}{\sum_{m=1}^{N} \alpha_m(x)} y_n$

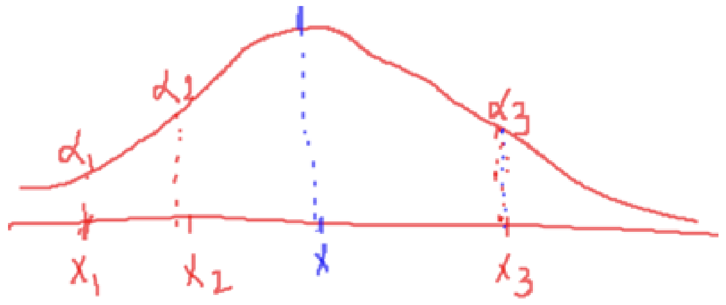
A different interpretation of RBF equation

- $\hat{f}(x) = \frac{\sum_{n=1}^{N} \alpha_n y_n}{\sum_{m=1}^{N} \alpha_m}$

where $\alpha_n = \varphi(\|x - x_n\|)$

$$\hat{f}(x) = \sum_{n=1}^{N} \frac{\alpha_n(x)}{\sum_{m=1}^{N} \alpha_m(x)} y_n$$

- Output is a weighted average of y-values.
- This corresponds to centering a single kernel, or a bump, at x (the query point).
- The value the bump attains at x_n determines the weight.

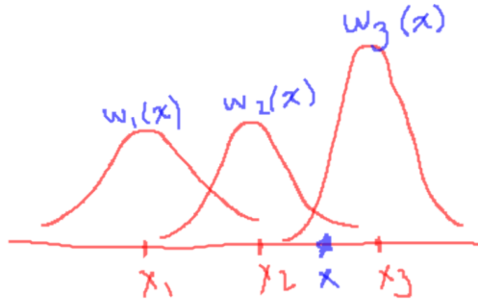


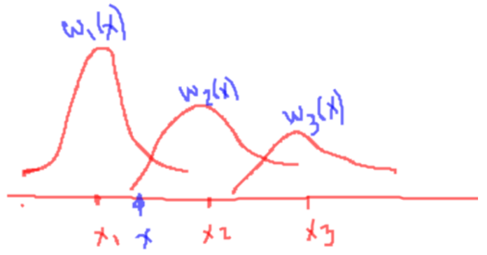
- $\hat{f}(x) = \sum_{n=1}^{N} \frac{y_n}{\sum_{m=1}^{N} \alpha_m(x)} \alpha_n(x)$

- $\hat{f}(x) = \sum_{n=1}^{N} w_n(x) \varphi(\|x - x_n\|)$

Where $w_n(x) = \frac{y_n}{\sum_{m=1}^{N} \varphi(\|x - x_m\|)}$

- This corresponds to centering a bump at every xn.
- The output is the sum of N bumps of different heights, where each bump is centered on a training point.
- The height of the bump centered on xn varies depending on the query point:
- $w_n(x) = \frac{y_n}{\sum_{m=1}^{N} \varphi(\|x - x_m\|)}$



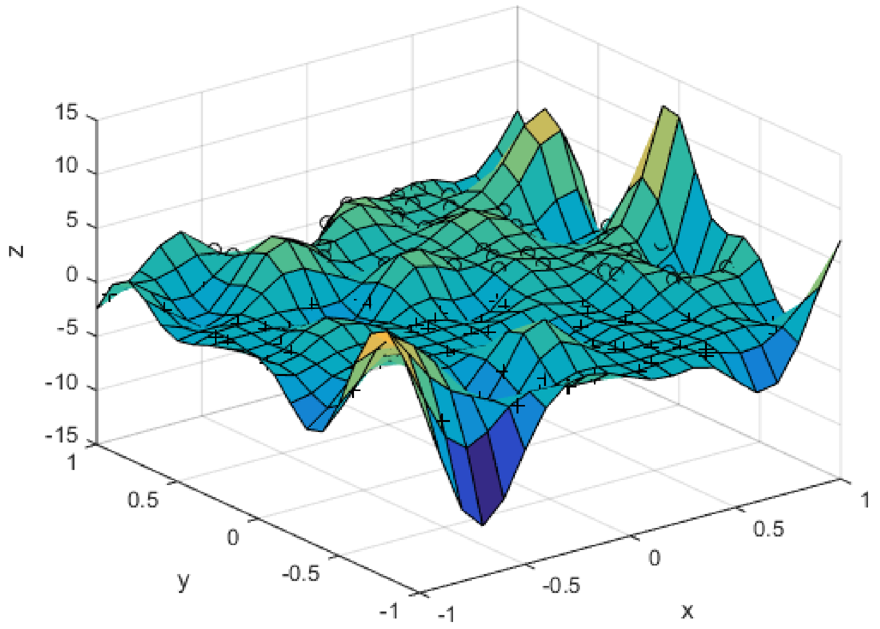


Parametric RBF Kernel Regression Hypothesis

- Fix the bump heights to w_n, independent of the query point.
- This will simplify the functional form to

$$\mathrm{h}(x) = \sum_{n=1}^{N} w_n \varphi(\|x - x_n\|)$$

which is the weighted summation of N Gaussians.
- Note that before making the weights fixed, $w_n(x)$ were specified by the data and there was nothing to learn.
- However in this hypothesis, w_n are the parameters. This model is parametric and parameters need to be learned.

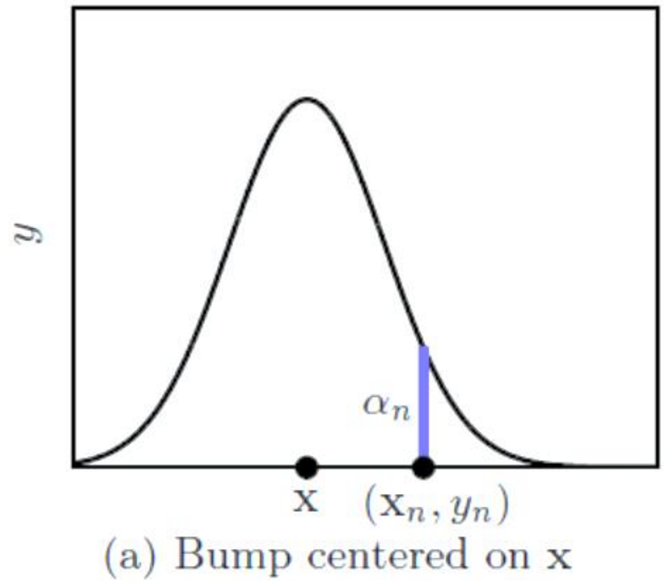


RBF Network Hypothesis Surface in 3 Dimensions

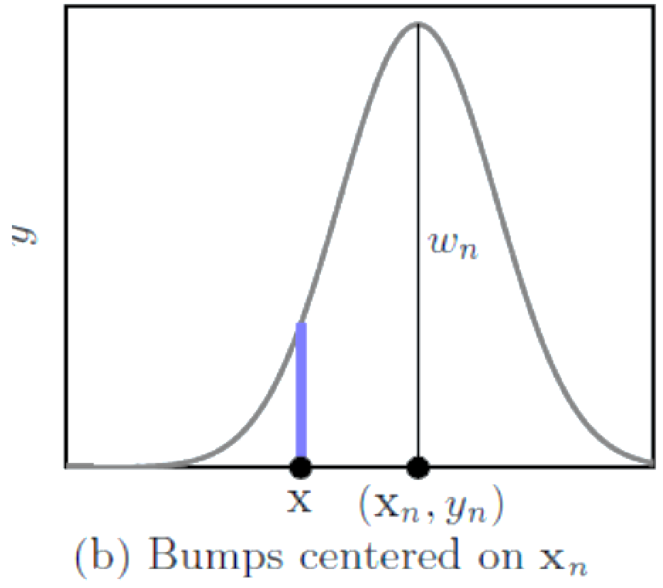
RBF Kernel Regression – cont. Different views of the RBF

(a) **Output** is a weighted sum of $\mathbf{y}_n$ with weights $\boldsymbol{\alpha}_n$ determined by a bump centered on $\mathbf{x}$.

(b) **Output** is a sum of bumps, one on each $\mathbf{x}_n$ having height $\mathbf{w}_n$.



(a) Bump centered on x



(b) Bumps centered on x_n

RBF Kernel Regression – cont.
The Hypothesis and the Learning Algorithm

- Learning algorithm:
  1- What are we learning?
  2- What's the error measure?

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \underbrace{\exp\left(-\gamma \left\|\mathbf{x} - \mathbf{x}_n\right\|^2\right)}_{\text{basis function}}$$

- Goal of the learning algorithm: Learn W's from data points (X_1, y_1),…(X_N, y_N) such that $h(X_n)=y_n$ for n=1 to N.
- We expect E_{in} to be zero. Why?

$$E_{\text{in}} = 0: \quad h(\mathbf{x}_n) = y_n \text{ for } n = 1, \cdots, N:$$

$$\sum_{m=1}^{N} w_m \exp\left(-\gamma \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2\right) = y_n$$

RBF Kernel Regression – cont.
Writing the Equations in Matrix Form

$h(\mathbf{x}_n) = y_n$ for $n = 1, \cdots, N$:

$$\sum_{m=1}^{N} w_m \exp\left(-\gamma \|\mathbf{x}_n - \mathbf{x}_m\|^2\right) = y_n$$

$$w_1 e^{-\gamma||X_1-X_1||^2} + w_2 e^{-\gamma||X_1-X_2||^2} + \ldots + w_N e^{-\gamma||X_1-X_N||^2} = y_1$$

$$w_1 e^{-\gamma||X_2-X_1||^2} + w_2 e^{-\gamma||X_2-X_2||^2} + \ldots + w_N e^{-\gamma||X_2-X_N||^2} = y_2$$

…

…

…

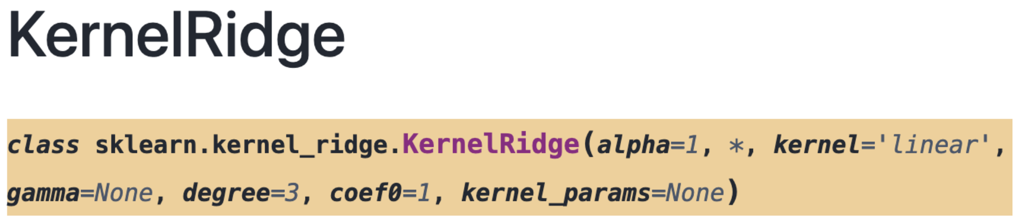
$$w_1 e^{-\gamma||X_N-X_1||^2} + w_2 e^{-\gamma||X_N-X_2||^2} + \ldots + w_N e^{-\gamma||X_N-X_N||^2} = y_N$$

$$\begin{bmatrix} e^{-\gamma||X_1-X_1||^2} & e^{-\gamma||X_1-X_2||^2} & \ldots & e^{-\gamma||X_1-X_N||^2} \\ e^{-\gamma||X_2-X_1||^2} & e^{-\gamma||X_2-X_2||^2} & \ldots & e^{-\gamma||X_2-X_N||^2} \\ & \vdots & \vdots & \vdots \\ e^{-\gamma||X_N-X_1||^2} & e^{-\gamma||X_N-X_2||^2} & \ldots & e^{-\gamma||X_N-X_N||^2} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$\boxed{\mathrm{w} = \Phi^{-1}\mathrm{y}}$ "exact interpolation"

RBF Kernel Regression Summary

- In many real-world scenarios, the relationship between your input features (X) and your target variable (y) isn't linear. Linear regression would struggle to capture these non-linear patterns.
- Kernel regression addresses this by implicitly mapping the data to a high-dimensional feature space where the relationship might be linear.
- A very popular kernel that can capture complex, non-linear relationships. It measures similarity based on the distance between points.



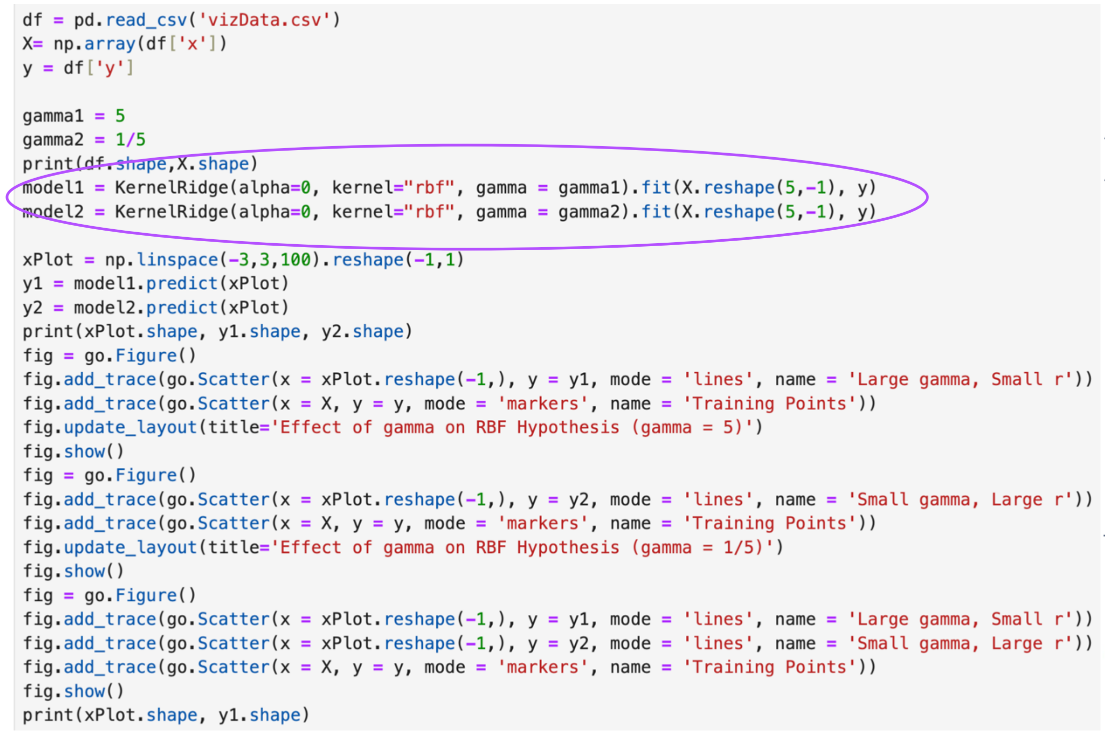
KernelRidge

```
class sklearn.kernel_ridge.KernelRidge(alpha=1, *, kernel='linear',
gamma=None, degree=3, coef0=1, kernel_params=None)
```

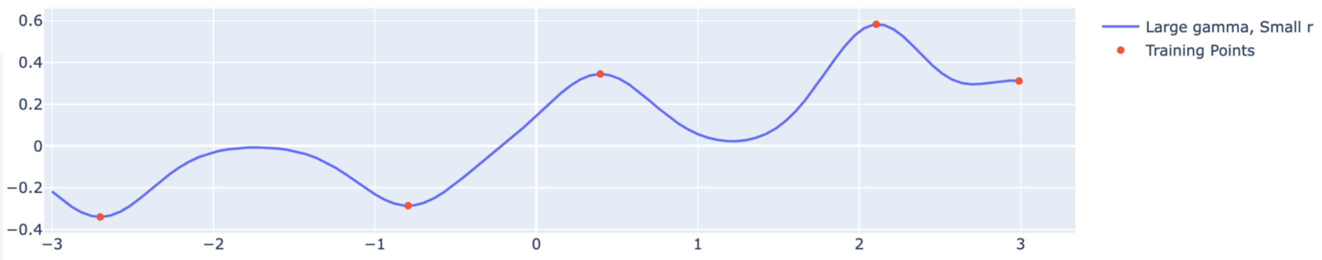
Choose ‘rbf’ for kernel.

- **How it Works:**
  - **Kernel Matrix:** The algorithm first computes a *kernel matrix (phi)*. This matrix contains the similarity scores between all pairs of training data points, as calculated by the chosen kernel function.
  - **Weights:** The algorithm then learns weights associated with each training data point. These weights determine the influence of each training example on the predictions. Data points that are "similar" to a new data point (according to the kernel) will have higher weights.
  - **Prediction:** To make a prediction for a new data point, the algorithm calculates a weighted average of the target values of the training data points. The weights are determined by the kernel function and the learned weights.

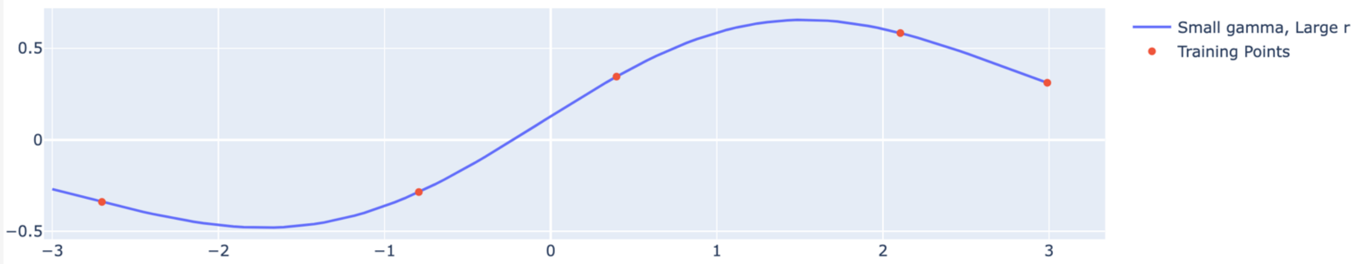
RBF Kernel Regression in Python – Effect of gamma on the final hypothesis

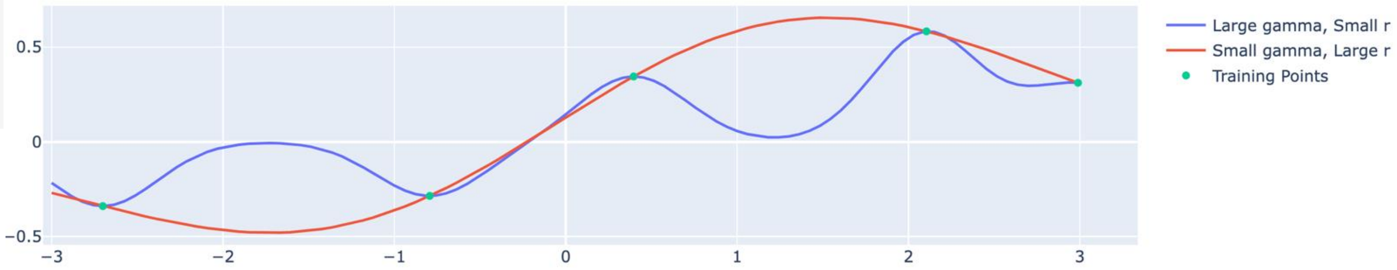

```
df = pd.read_csv('vizData.csv')
X= np.array(df['x'])
y = df['y']

gamma1 = 5
gamma2 = 1/5
print(df.shape,X.shape)
model1 = KernelRidge(alpha=0, kernel="rbf", gamma = gamma1).fit(X.reshape(5,-1), y)
model2 = KernelRidge(alpha=0, kernel="rbf", gamma = gamma2).fit(X.reshape(5,-1), y)

xPlot = np.linspace(-3,3,100).reshape(-1,1)
y1 = model1.predict(xPlot)
y2 = model2.predict(xPlot)
print(xPlot.shape, y1.shape, y2.shape)
fig = go.Figure()
fig.add_trace(go.Scatter(x = xPlot.reshape(-1,), y = y1, mode = 'lines', name = 'Large gamma, Small r'))
fig.add_trace(go.Scatter(x = X, y = y, mode = 'markers', name = 'Training Points'))
fig.update_layout(title='Effect of gamma on RBF Hypothesis (gamma = 5)')
fig.show()
fig = go.Figure()
fig.add_trace(go.Scatter(x = xPlot.reshape(-1,), y = y2, mode = 'lines', name = 'Small gamma, Large r'))
fig.add_trace(go.Scatter(x = X, y = y, mode = 'markers', name = 'Training Points'))
fig.update_layout(title='Effect of gamma on RBF Hypothesis (gamma = 1/5)')
fig.show()
fig = go.Figure()
fig.add_trace(go.Scatter(x = xPlot.reshape(-1,), y = y1, mode = 'lines', name = 'Large gamma, Small r'))
fig.add_trace(go.Scatter(x = xPlot.reshape(-1,), y = y2, mode = 'lines', name = 'Small gamma, Large r'))
fig.add_trace(go.Scatter(x = X, y = y, mode = 'markers', name = 'Training Points'))
fig.show()
print(xPlot.shape, y1.shape)
```

Effect of gamma on RBF Hypothesis (gamma = 5)



Effect of gamma on RBF Hypothesis (gamma = 1/5)





RBF with K centers: RBF Networks

- The parametric RBF has N parameters $w_1, \ldots, w_N$, ensuring we can always fit the data.
- When the data has stochastic or deterministic noise, this means we will overfit.
- The root of the problem is that we have too many parameters, one for each bump.

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2\right)$$

- Solution: **restrict the number of bumps to k ≪ N**. If we restrict the number of bumps to k, then only k weights $w_1, \ldots, w_k$ need to be learned.

$$h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)$$

RBF Networks – cont
Choosing the Centers

$$h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \left\|\mathbf{x} - \boldsymbol{\mu}_k\right\|^2\right)$$

- Where should the bumps be placed?
- Previously, with N bumps, this was not an issue since we placed one bump on each data point.
- Though we cannot place a bump on each data point, we can still try to choose the bump centers so that they represent the data points as closely as possible.
- We appear to have an abundance of parameters in the centers μ_k (each parameter is d-dimension). So there doesn't seem to be a lot of improvement!

RBF Networks – cont. Choosing the Centers

$$h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right) + w_0$$

- We should choose the bump centers to closely represent the **inputs** in the data.
- We can choose the centers by choosing them to represent the inputs $x_1, \ldots, x_N$ **without reference** to the $y_1, \ldots, y_N$.
- We require that no x_n be far away from a bump center. The x_n should cluster around the centers, with each center μ_k representing one cluster.
- Solution: k-means clustering
- Observe that we have added back the bias term w_0. For the parametric RBF with N bumps, we did not need the bias term. However, when you only have a small number of bumps and no bias term, the learning gets distorted if the y-values have non-zero mean

RBF Networks – cont. Determining the Weights

$$\sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2\right) \approx y_n$$

$$w_0 + w_1 e^{-\gamma||X_1-\mu_1||^2} + w_2 e^{-\gamma||X_1-\mu_2||^2} + \ldots + w_K e^{-\gamma||X_1-\mu_K||^2} \approx y_1$$

$$w_0 + w_1 e^{-\gamma||X_2-\mu_1||^2} + w_2 e^{-\gamma||X_2-\mu_2||^2} + \ldots + w_K e^{-\gamma||X_2-\mu_K||^2} \approx y_2$$

…

…

…

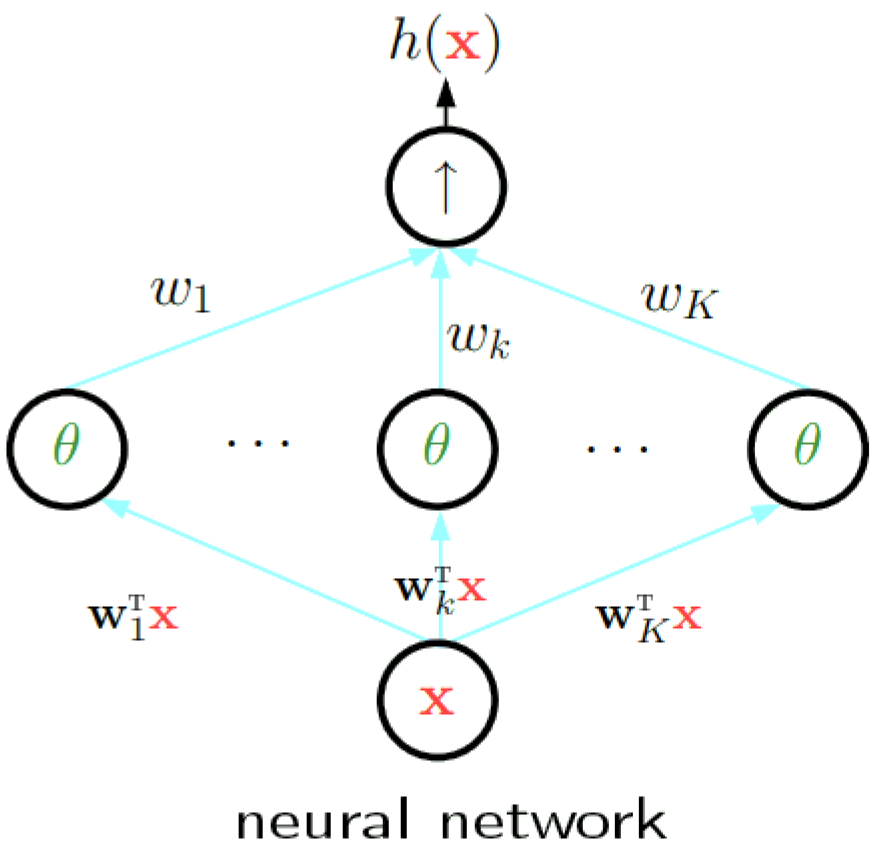
$$w_0 + w_1 e^{-\gamma||X_N-\mu_1||^2} + w_2 e^{-\gamma||X_N-\mu_2||^2} + \ldots + w_K e^{-\gamma||X_N-\mu_K||^2} \approx y_N$$

$$\boxed{\mathbf{w} = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{y}}$$

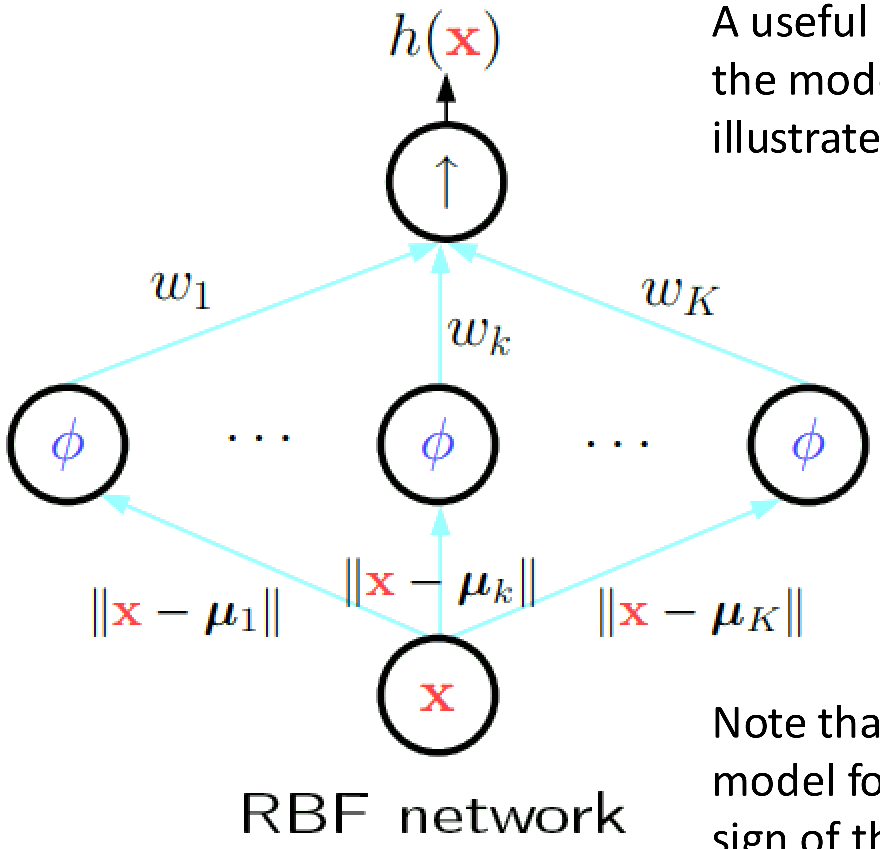
$$\begin{bmatrix} 1 & e^{-\gamma||X_1-\mu_1||^2} & e^{-\gamma||X_1-\mu_2||^2} & \ldots & e^{-\gamma||X_1-\mu_K||^2} \\ 1 & e^{-\gamma||X_2-\mu_1||^2} & e^{-\gamma||X_2-\mu_2||^2} & \ldots & e^{-\gamma||X_2-\mu_K||^2} \\ & & \vdots & \vdots & \vdots \\ 1 & e^{-\gamma||X_N-\mu_1||^2} & e^{-\gamma||X_N-\mu_2||^2} & \ldots & e^{-\gamma||X_N-\mu_K||^2} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

RBF Networks

$$h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \left\|\mathbf{x} - \boldsymbol{\mu}_k\right\|^2\right)$$



$h(\mathbf{x})$
w_1
w_k
w_K
θ
$\mathbf{w}_1^{\mathrm{T}}\mathbf{x}$
$\mathbf{w}_k^{\mathrm{T}}\mathbf{x}$
$\mathbf{w}_K^{\mathrm{T}}\mathbf{x}$
$\mathbf{x}$
neural network



$h(\mathbf{x})$
w_1
w_k
w_K
ϕ
$\|\mathbf{x} - \boldsymbol{\mu}_1\|$
$\|\mathbf{x} - \boldsymbol{\mu}_k\|$
$\|\mathbf{x} - \boldsymbol{\mu}_K\|$
$\mathbf{x}$
RBF network

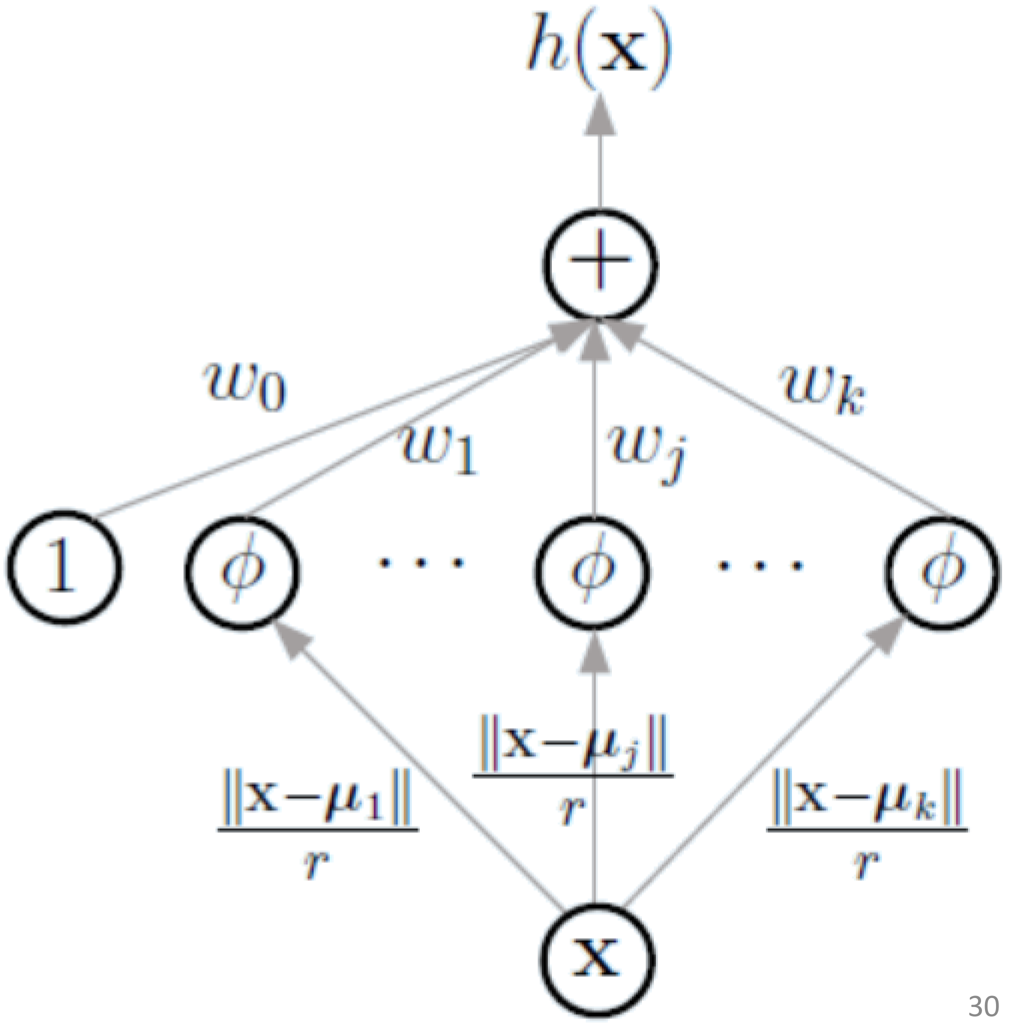
A useful graphical representation of the model as a ‘feed forward’ network is illustrated.

Note that we can use the RBF-network model for classification by taking the sign of the output signal, and for logistic regression we pass the output signal through the sigmoid $\theta(s) = e^s/(1 + e^s)$.

RBF Network

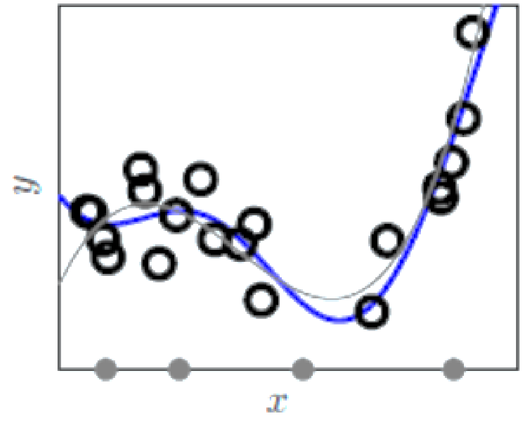
$$h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)$$

- Note that the hypothesis set is very similar to a linear model except that the transformed features
- $\varphi_k(X)$can depend on the data set (through the choice of the μ_k which are chosen to fit the data). But in linear regression, the features were fixed ahead of time.
- Because the μ_k appear inside a nonlinear function, this model is not linear in its parameters. It is linear in the w_k , but nonlinear in the μ_k . It turns out that allowing the basis functions to depend on the data adds significant power to this model over the linear model

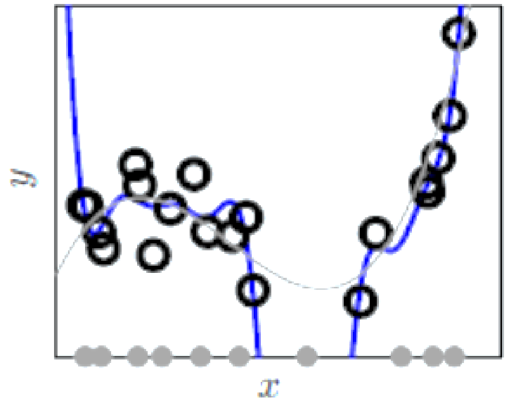

$h(\mathbf{x})$
$+$
w_0
w_1
w_j
w_k
1
ϕ
ϕ
ϕ
$\frac{\|\mathbf{x}-\boldsymbol{\mu}_1\|}{r}$
$\frac{\|\mathbf{x}-\boldsymbol{\mu}_j\|}{r}$
$\frac{\|\mathbf{x}-\boldsymbol{\mu}_k\|}{r}$
$\mathbf{x}$

K and Gamma

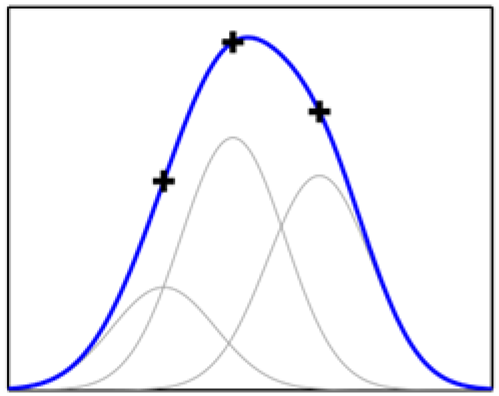
- Other than the parameters w and μ, there are two high-level parameters k and gamma which specify the nature of the hypothesis set.
- These parameters control two aspects of the hypothesis set:
  - size of a hypothesis set (quantified by k, the number of bumps)
  - complexity of a single hypothesis (quantified by gamma, which determines how ‘wiggly’ an individual hypothesis is).
- It is important to choose a good value of k to avoid overfitting (too high a k) or underfitting (too low a k). Same with gamma.
- Use Cross-validation!



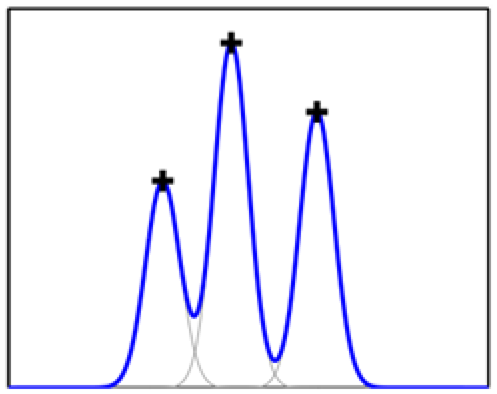
(a) $k = 4$



(b) $k = 10$



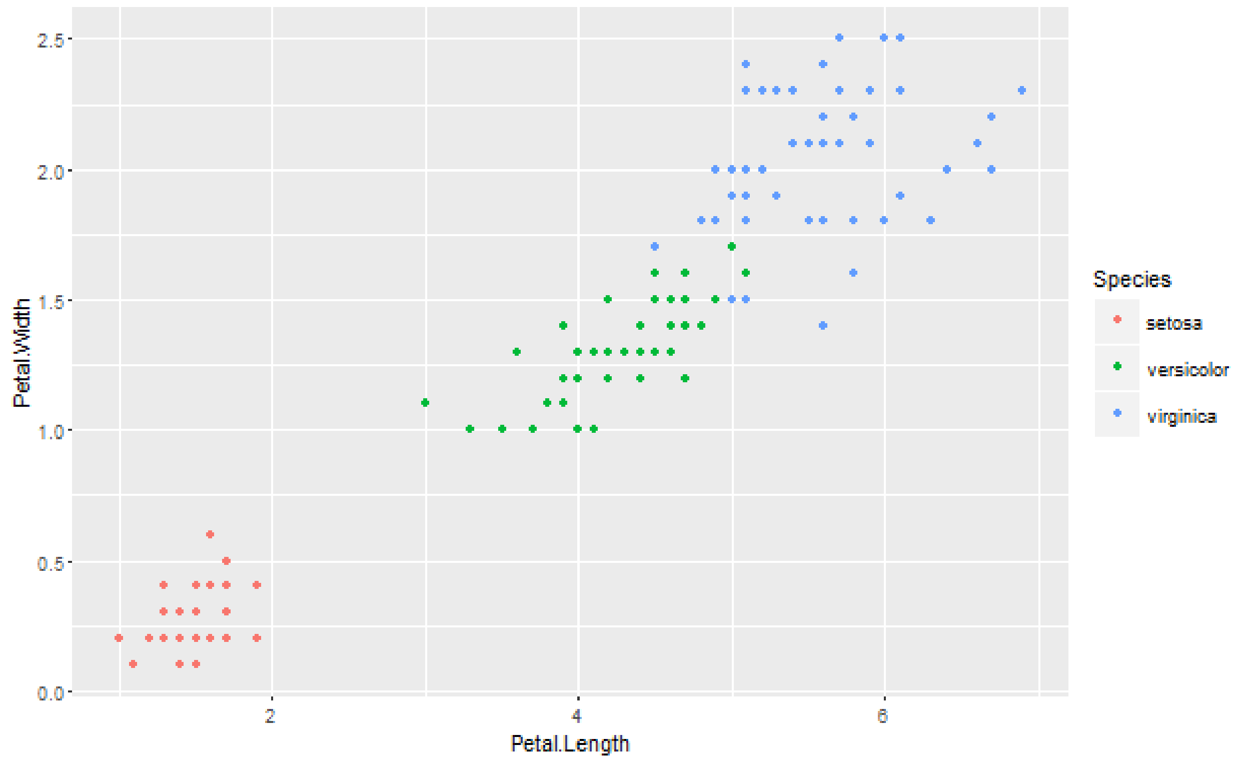
small γ



large γ

Clustering – Unsupervised Classification

- Cluster: a collection of data objects
  - Similar to one another within the same cluster
  - Dissimilar to the objects in other clusters
- Clustering is unsupervised classification, i.e. no predefined classes
- Assessment of clustering quality is application-dependent and to an extent subjective
- Typical applications
  - As a stand-alone tool to get insight into data distribution
  - As a preprocessing step for other algorithms (e.g. Use of K means clustering in RBF models)



Examples of Clustering Applications (as a stand-alone tool)

- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- Urban planning: Identifying groups of houses according to their house type, value, and geographical location
- Politics: Help campaign managers to identify voters with similar interests
- Insurance: Identifying groups of motor insurance policy holders with a high average claim cost

K-means Clustering

- The k-means clustering algorithm is a simple yet powerful tool for obtaining <u>clusters</u> and <u>cluster centers</u>.
- The goal of k-means clustering is to
  1. Select centers $\mu_1,\ldots,\mu_k$ for each cluster
  2. Assign input data points $x_1,\ldots,x_N$ into K cluster sets $S_1,\ldots S_K$
- The centers are representative of the data if every data point in cluster S_k is close to its corresponding center μ_k.
- For cluster S_k with center μ_k, define the squared error measure to quantify the quality of the cluster:

$$\sum_{\mathbf{x}_n \in S_k} \left\|\mathbf{x}_n - \boldsymbol{\mu}_k\right\|^2$$

- The k-means error function just sums this cluster error over all clusters:

$$\sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k} \left\|\mathbf{x}_n - \boldsymbol{\mu}_k\right\|^2$$

K-means Clustering Algorithm

- Objective: Minimize the distance between x_n and the closest enter μ_k by splitting $x_1, \cdots, x_N$ into clusters $S_1, \cdots, S_K$

$$\text{Minimize} \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k} \left\| \mathbf{x}_n - \boldsymbol{\mu}_k \right\|^2$$

- Finding the global minimum of the above cost function is intractable (similar to the NN case).
- We seek an iterative approach to find a local minimum.

Lloyd's algorithm

- Select initial centroids at random.
- Assign each object to the cluster with the nearest centroid:

$$S_k \leftarrow \{\mathbf{x}_n : \|\mathbf{x}_n - \boldsymbol{\mu}_k\| \leq \text{all } \|\mathbf{x}_n - \boldsymbol{\mu}_\ell\|\}$$

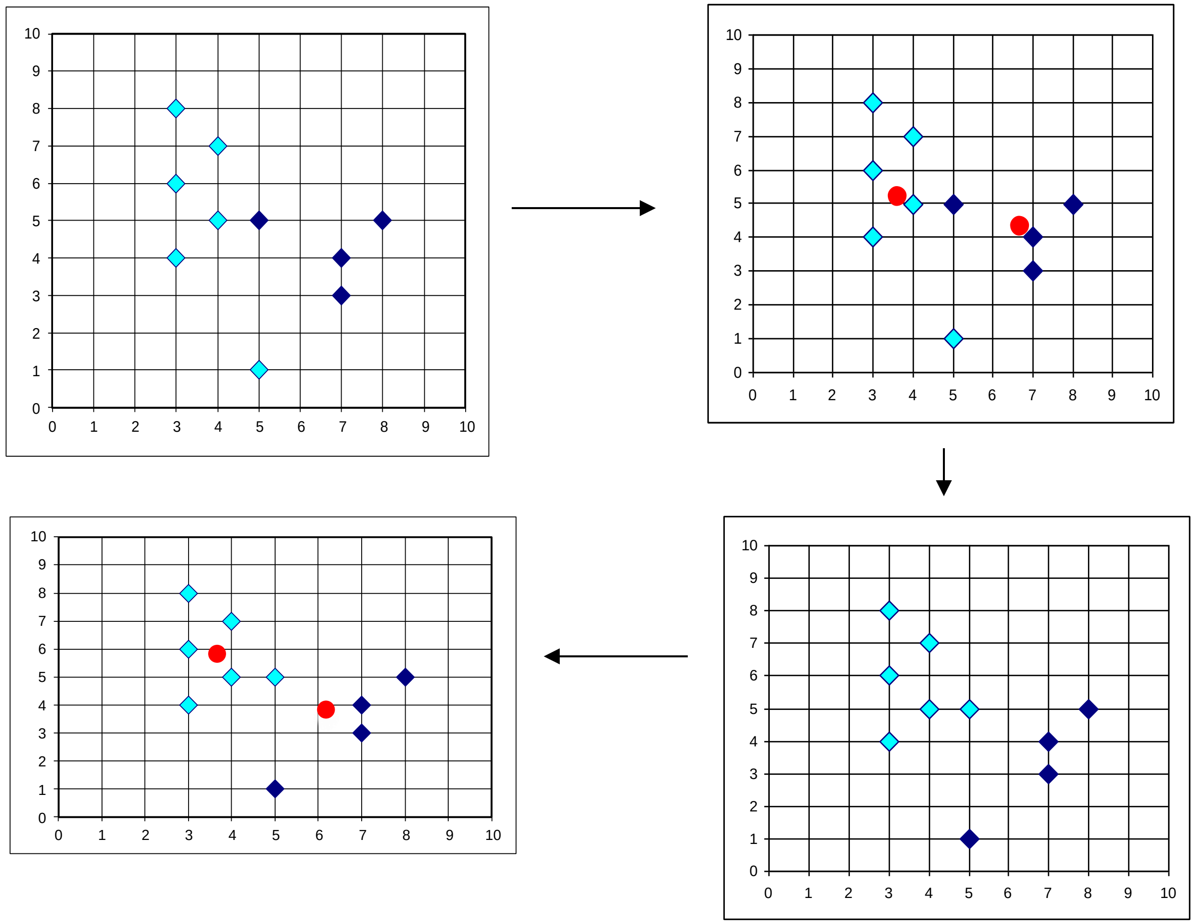
- Compute each centroid as the mean of the objects assigned to it:

$$\boldsymbol{\mu}_k \leftarrow \frac{1}{|S_k|} \sum_{\mathbf{x}_n \in S_k} \mathbf{x}_n$$

- Repeat previous 2 steps until error stops decreasing.

Example

- $x_1 = \frac{3\times3+2\times4+5}{6} = 3.67$
- $y_1 = \frac{8+7+6+5+4+1}{6} = 5.17$
- $x_2 = \frac{5+2\times7+8}{4} = 6.75$
- $y_2 = \frac{3+4+2\times5}{4} = 4.25$



- $x_1 = \frac{3\times3+2\times4+5}{6} = 3.67$
- $y_1 = \frac{8+7+6+2\times5+4}{6} = 5.83$
- $x_2 = \frac{5+2\times7+8}{4} = 6.75$
- $y_2 = \frac{1+3+4+5}{4} = 3.25$

Summary

Nonparametric RBF

Parametric RBF

$$h(\boldsymbol{X}) = \sum_{n=1}^{N} \frac{\alpha_n(X)}{\sum_{m=1}^{N} \alpha_m(X)} y_n$$

$$h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \left\|\mathbf{x} - \boldsymbol{\mu}_k\right\|^2\right)$$

$$\alpha_n(X) = \varphi(\|X - X_n\|) = e^{-\left||X - X_n|\right|^2/r}$$

For classification you take the sign of h(x).

References

- Learning From Data by Abu Mustafa