

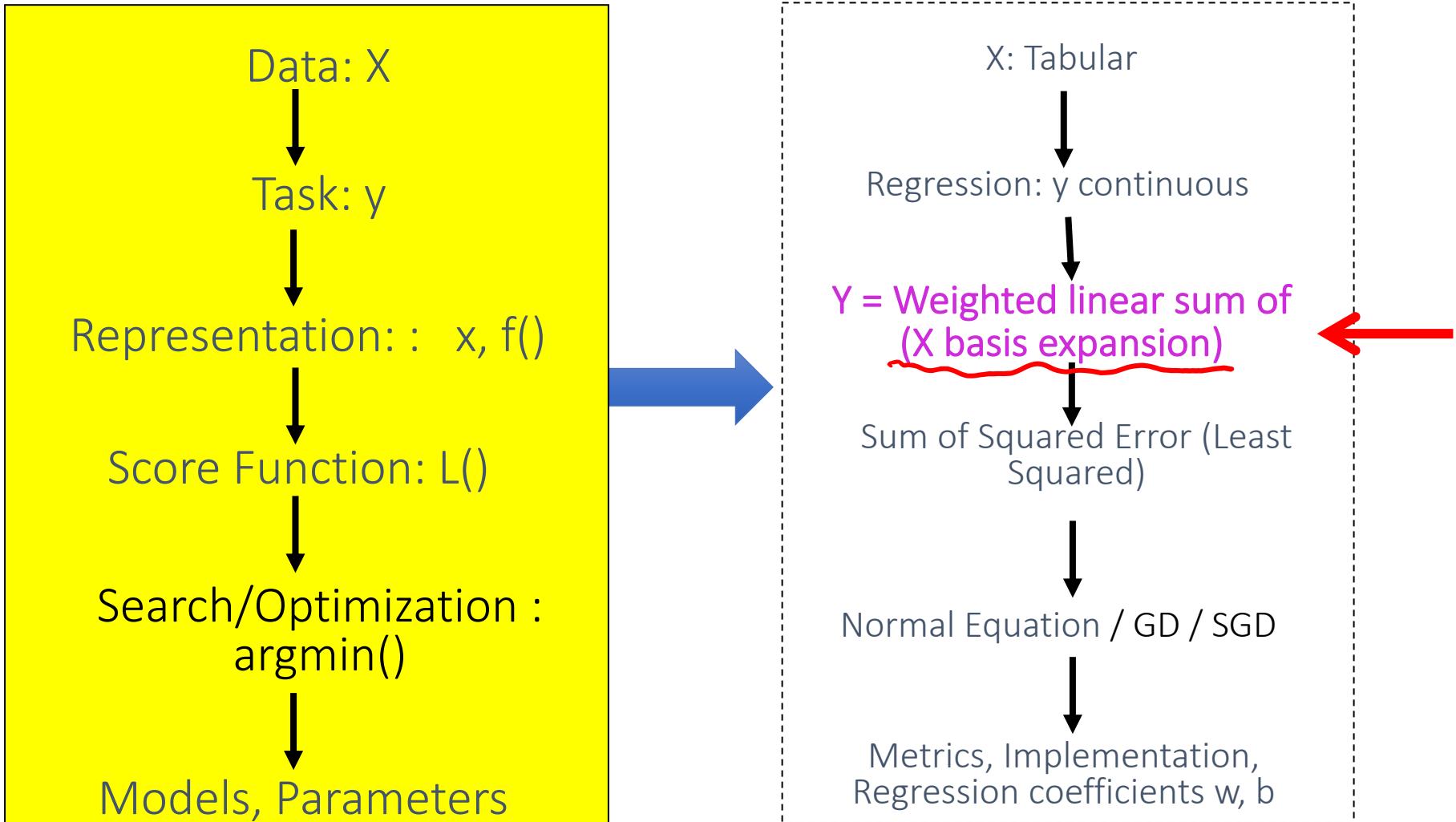
UVA CS 4774: Machine Learning

Lecture 5: Linear Regression with Basis Functions Expansion

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Today : Multivariate (non-) Linear Regression with Basis Expansion



$$\hat{y} = \theta_0 + \sum_{j=1}^m \theta_j \varphi_j(x) = \underline{\varphi(x)^T \theta}$$

Linear Regression with non-linear basis functions

- LR can deal with nonlinear relationships

$$\hat{y} = \theta^T \mathbf{x} \quad \longrightarrow \quad \hat{y} = \theta_0 + \sum_{j=1}^m \theta_j \varphi_j(\mathbf{x}) = \theta^T \varphi(\mathbf{x})$$

$$\vec{\theta} = [\theta_0, \theta_1, \dots, \theta_m]^T$$

$$\vec{\varphi} = [1, \varphi_1(x), \dots, \varphi_m(x)]^T$$

LR with non-linear basis functions

- Free to design basis functions (e.g., non-linear features):

Here $\varphi_j(x)$ are [predefined] basis functions (also $\varphi_0(x)=1$)

- E.g.: polynomial regression with degree up-to two ($d=2$):

$$\varphi(x) := [1, x, x^2]^T$$

Linear $\vec{x} : [1, x]^T$

Polynomial basis based Regression

<https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.06-Linear-Regression.ipynb>

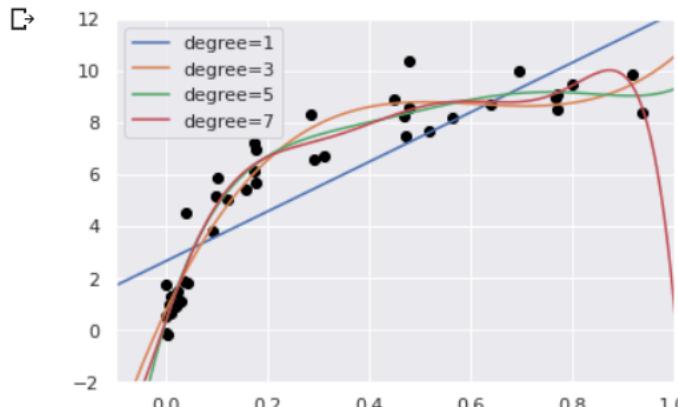
My google-colab modified version: (I will cell-run it during class)

https://colab.research.google.com/drive/1gKEk0BuVORnpw_5jQ10wY1ZkA2mSXEAG?usp=sharing

```
▶ %matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set() # plot formatting

X_test = np.linspace(-0.1, 1.1, 500)[:, None]

plt.scatter(X.ravel(), y, color='black')
axis = plt.axis()
for degree in [1, 3, 5, 7]:
    y_test = PolynomialRegression(degree).fit(X, y).predict(X_test)
    plt.plot(X_test.ravel(), y_test, label='degree={0}'.format(degree))
plt.xlim(-0.1, 1.0)
plt.ylim(-2, 12)
plt.legend(loc='best');
```

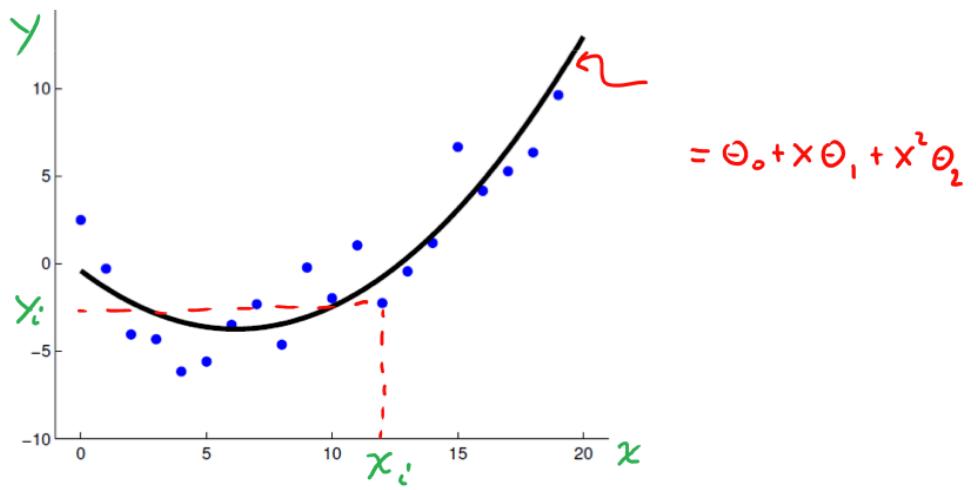
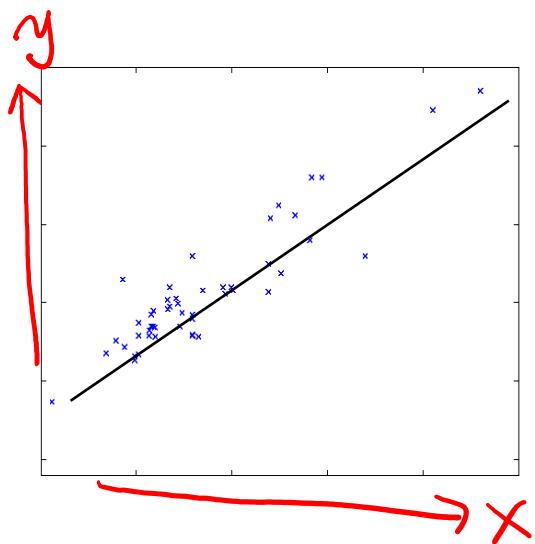


e.g. (1) polynomial regression

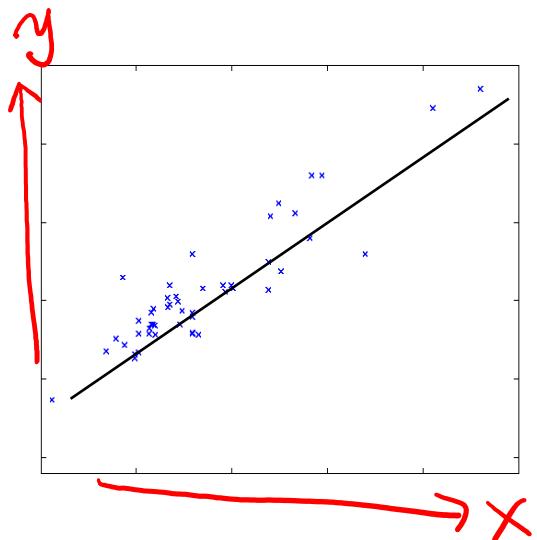
$$\hat{y} = \theta^T \mathbf{x}$$



$$\hat{y} = \theta^T \varphi(\mathbf{x})$$



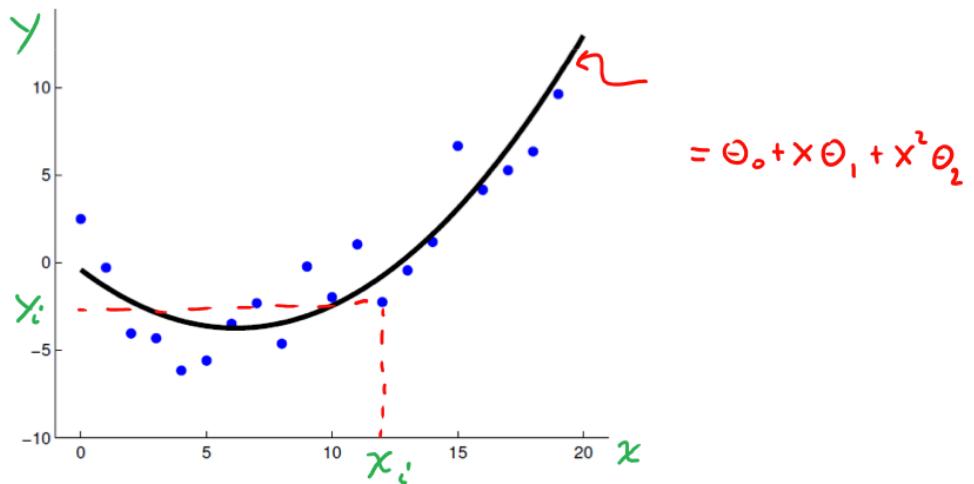
$$\hat{y} = \theta^T \mathbf{x}$$



$$\theta^* = (X^T X)^{-1} X^T \vec{y}$$

9/10/20

$$\hat{y} = \theta^T \varphi(\mathbf{x})$$



$$\theta^* = (\varphi^T \varphi)^{-1} \varphi^T \vec{y}$$

$$\varphi(x) := [1, x, x^2]^T$$

feature engineering

Linear $\vec{X} : [1, x]^T$

$$\theta^* = (X^T X)^{-1} X^T \vec{y}$$

$$X_{n \times p} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ n \end{bmatrix} \xrightarrow{(p+1)}$$

$$\varphi(x) := \begin{bmatrix} 1, x, x^2 \end{bmatrix}^T$$

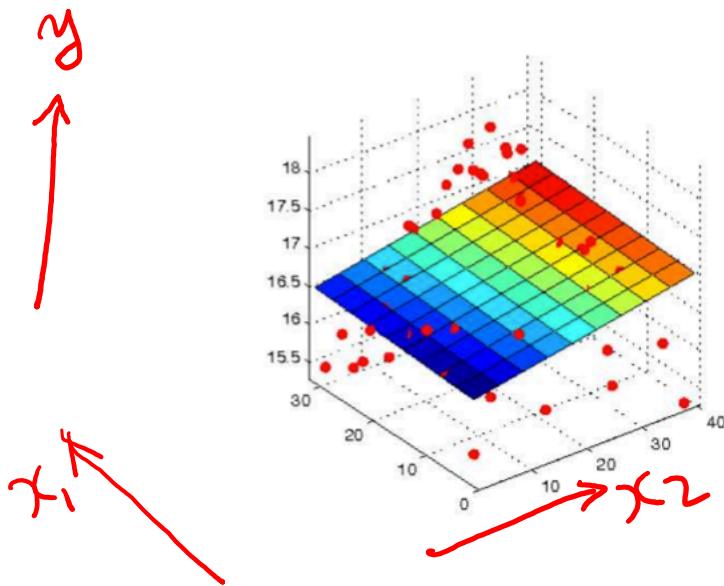
$$\theta^* = (\varphi^T \varphi)^{-1} \varphi^T \vec{y}$$

$$\varphi(x) = \begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n \\ 1 & \dots & m \end{bmatrix}$$

$$\vec{\theta} \in \mathbb{R}^m$$

$$\hat{y} = \theta^T \mathbf{x}$$

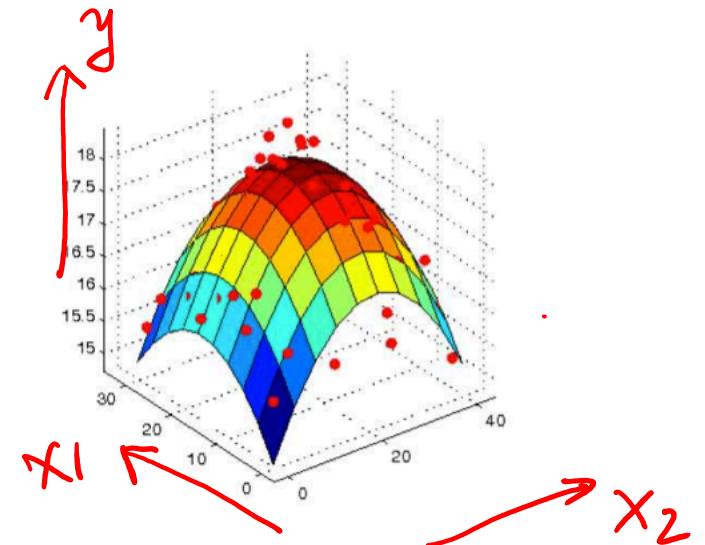
$$\phi(\mathbf{x}) = [1, x_1, x_2]$$



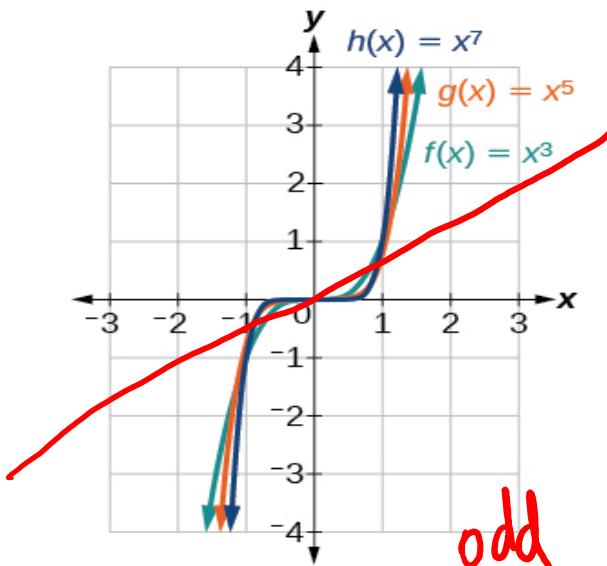
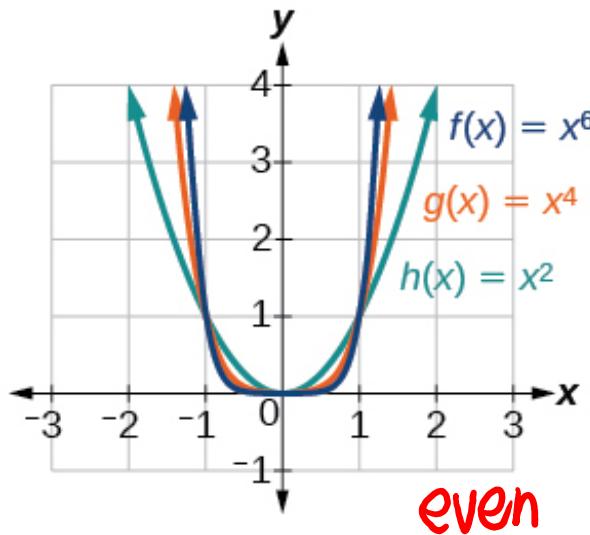
KEY: when the basis func are given, the problem of learning the parameters from data is still LR.

$$\hat{y} = \theta^T \varphi(\mathbf{x})$$

$$\phi(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_2^2] , x_1 x_2$$



$$\varphi_j(x) = x^{j-1}$$



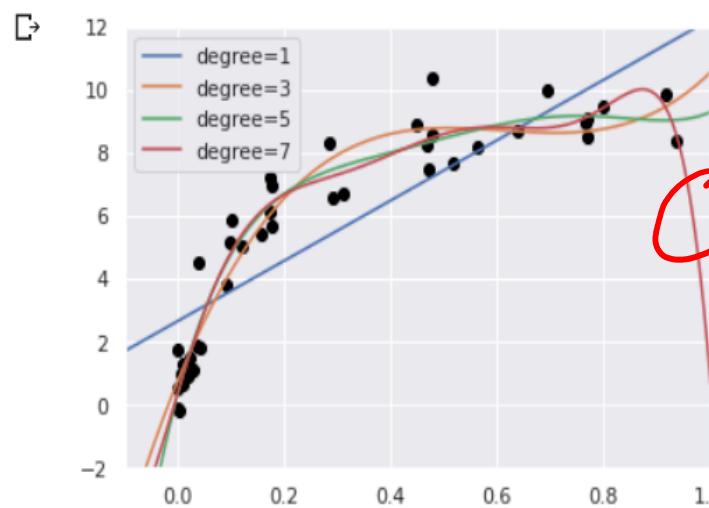
9/10/20

My google-colab modified version: (I will cell-run it)
https://colab.research.google.com/drive/1gKEk0BuVO_Rnpw_5jQ10wY1ZkA2mSXEAG?usp=sharing

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set() # plot formatting

X_test = np.linspace(-0.1, 1.1, 500)[:, None]

plt.scatter(X.ravel(), y, color='black')
axis = plt.axis()
for degree in [1, 3, 5, 7]:
    y_test = PolynomialRegression(degree).fit(X, y).predict(X_test)
    plt.plot(X_test.ravel(), y_test, label='degree={0}'.format(degree))
plt.xlim(-0.1, 1.0)
plt.ylim(-2, 12)
plt.legend(loc='best');
```



Dr. Yanjun Qi / UVA CS

UVA CS 4774: Machine Learning

Lecture 5: Linear Regression with Basis Functions Expansion

Module 2

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

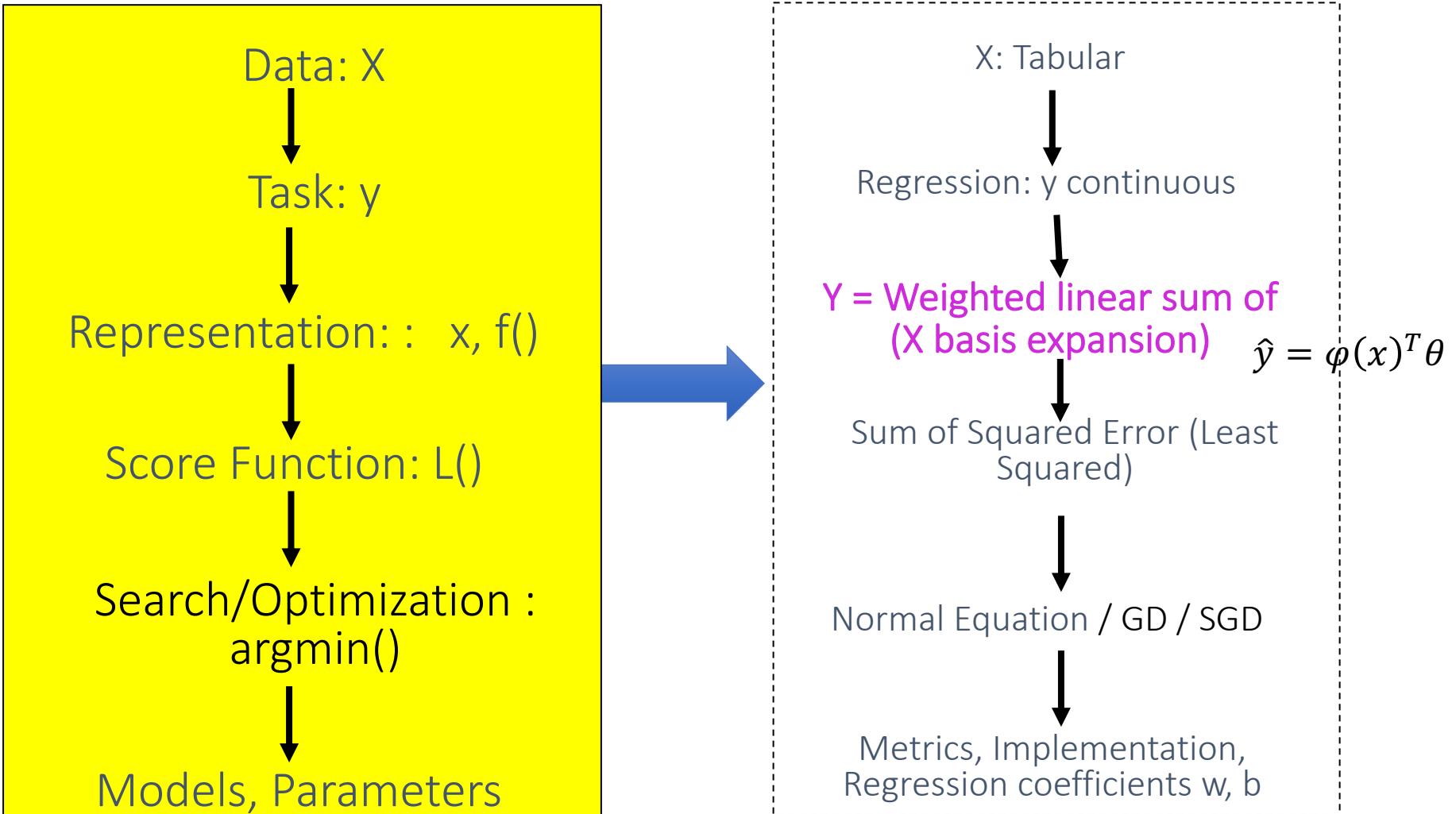
$$\hat{y} = \theta^T \mathbf{x}$$



$$\hat{y} = \theta^T \varphi(\mathbf{x})$$

Many Possible Basis functions

Recap : Multivariate (non-) Linear Regression with Basis Expansion



φ : Which and what type?

Many Possible Basis functions

- There are many basis functions, e.g.:

- Polynomial

$$\varphi_j(x) = x^{j-1}$$

$$[1, x, x^2, x^3, \dots, x^d]$$

- Radial basis functions

$$\varphi_j(x) = \exp\left(-\frac{(x - \mu_j)^2}{2\lambda^2}\right)$$



- Sigmoidal

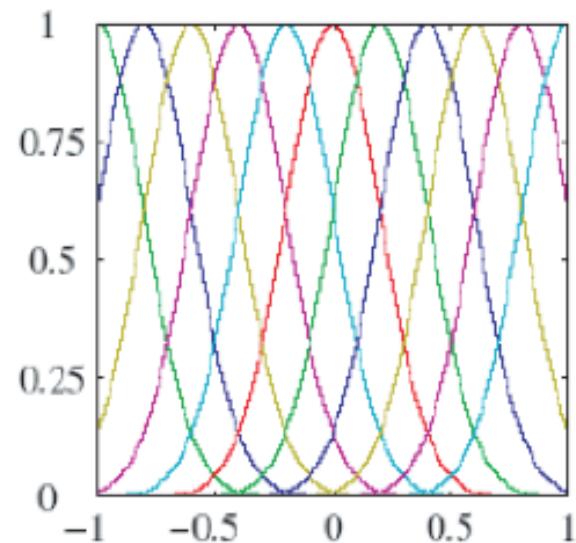
$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

- Splines,

- Fourier,

- Wavelets, etc

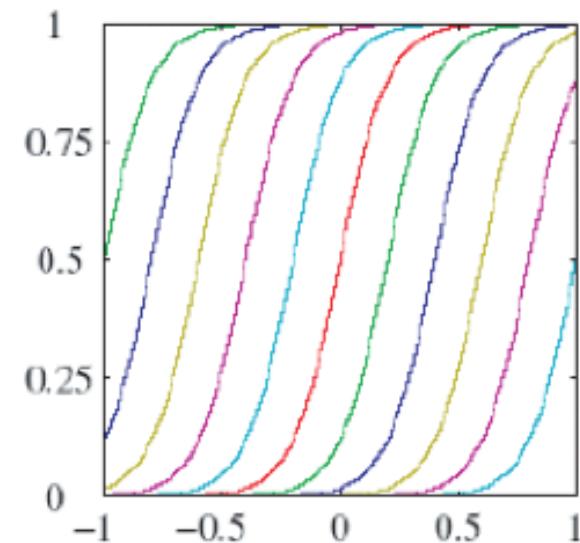
$$\varphi_j(x) = \exp\left(-\frac{(x - \mu_j)^2}{2\lambda^2}\right)$$



RBF

"bell"-Shaped

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$



Sigmoid

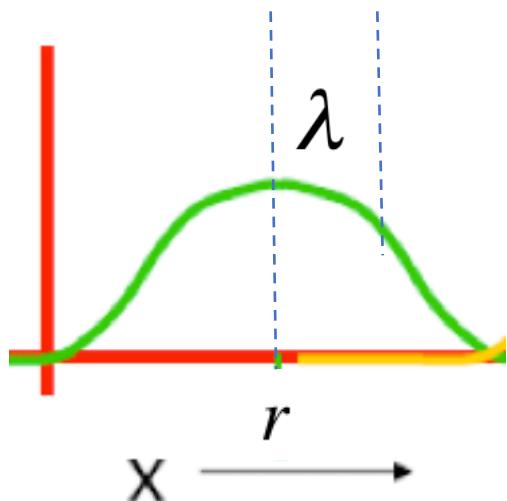
"S"-Shaped

RBF = radial-basis function: a function which depends on the radial distance from a Centre point

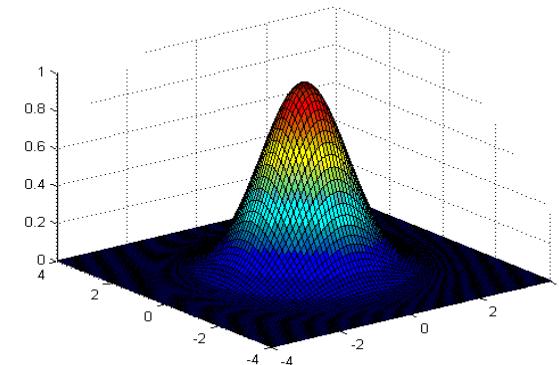
Gaussian RBF →

$$K_{\lambda}(x,r) = \exp\left(-\frac{(x-r)^2}{2\lambda^2}\right)$$

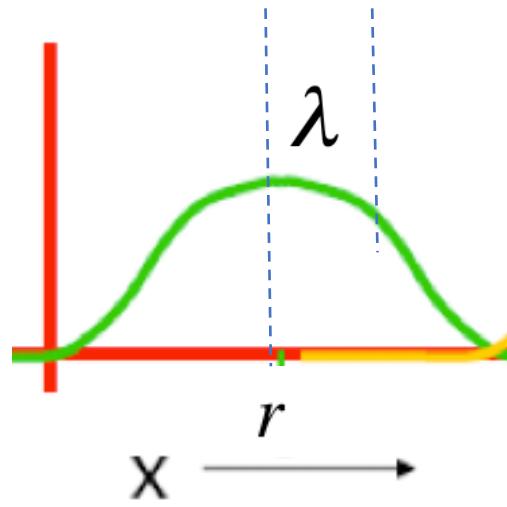
as distance from the center r increases, the output of the RBF decreases



1D case



2D case



$$K_\lambda(x, r) = \exp\left(-\frac{(x-r)^2}{2\lambda^2}\right)$$

$X =$	$K_\lambda(x, r) =$
r	1
$r + \lambda$	0.6065307
$r + 2\lambda$	0.1353353
$r + 3\lambda$	0.0001234098

LR with radial-basis functions

- E.g.: LR with RBF regression:

$$\hat{y} = \theta_0 + \sum_{j=1}^m \theta_j \varphi_j(x) = \varphi(x)^T \theta$$
$$\varphi_j(x) := K_{\lambda_j}(x, r_j) = \exp\left(-\frac{(x - \mu_j)^2}{2\lambda_j^2}\right)$$

hyperparameters of RBF basis functions
(the predefined Centers and Width)

LR with radial-basis functions

- E.g.: LR with RBF regression:

$$\hat{y} = \theta_0 + \sum_{j=1}^m \theta_j \varphi_j(x) = \varphi(x)^T \theta$$

$$\varphi_j(x) := K_{\lambda_j}(x, r_j) = \exp\left(-\frac{(x - \mu_j)^2}{2\lambda_j^2}\right)$$

$\varphi(x)$: E.g. with four predefined RBF kernels

$$= [1, K_{\lambda_1}(x, r_1), K_{\lambda_2}(x, r_2), K_{\lambda_3}(x, r_3), K_{\lambda_4}(x, r_4)]^T$$

e.g. (2) LR with radial-basis functions

- E.g.: LR with RBF regression:

$$\hat{y} = \theta_0 + \sum_{j=1}^m \theta_j \varphi_j(x) = \varphi(x)^T \theta$$

$$\varphi(x) := \left[1, K_{\lambda_1}(x, r_1), K_{\lambda_2}(x, r_2), K_{\lambda_3}(x, r_3), K_{\lambda_4}(x, r_4) \right]^T$$

$$\vec{\theta} = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4]^T$$

$$\theta^* = (\varphi^T \varphi)^{-1} \varphi^T \bar{y}$$

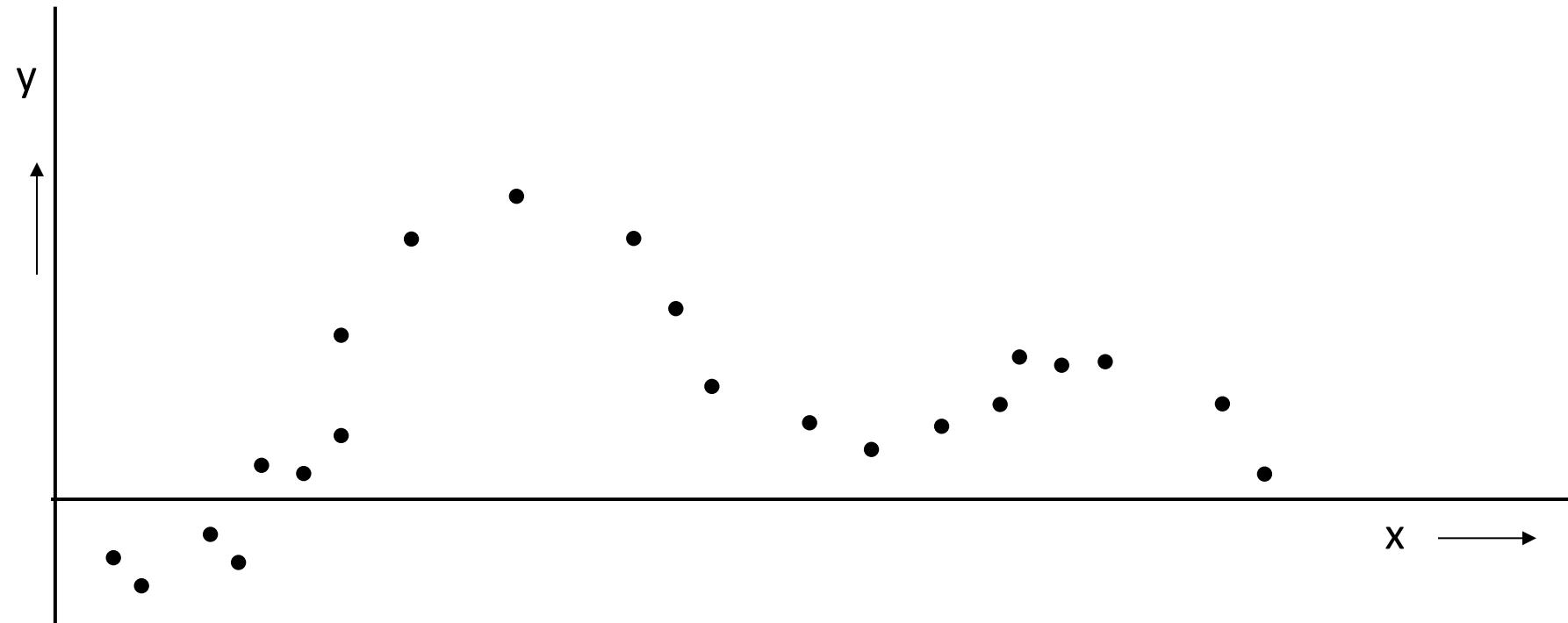
Users need to define the **hyperparameters** of RBF basis functions (the **predefined Centers and Width**)

$\varphi(x)$: E.g. with four predefined RBF kernels

$$= [1, K_{\lambda_1}(x, r_1), K_{\lambda_2}(x, r_2), K_{\lambda_3}(x, r_3), K_{\lambda_4}(x, r_4)]^T$$

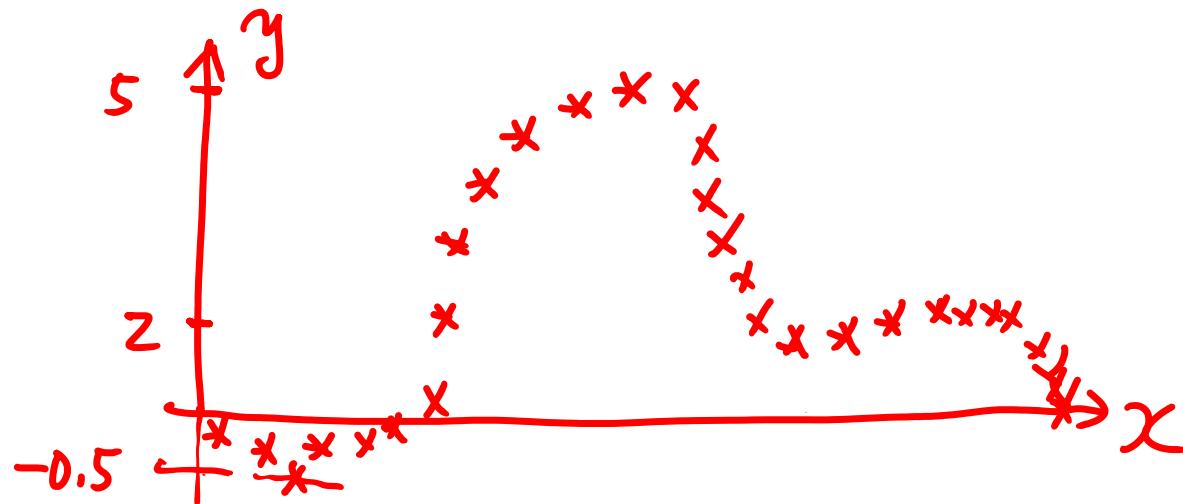
$$\theta^* = (\varphi^T \varphi)^{-1} \varphi^T \bar{y}$$

For example: I want to use 3 RBF basis functions to fit the following data points
(I need to assume 3 predefined centres and width)



Given Training Data's scatter plot:

$$(x_i, y_i) \quad i=1, \dots, n$$



After my 3 RBF fit:

$$f(x) = -0.5 k_{\lambda_1}(x, y_1) + 5 k_{\lambda_2}(x, y_2) + 2 k_{\lambda_3}(x, y_3)$$

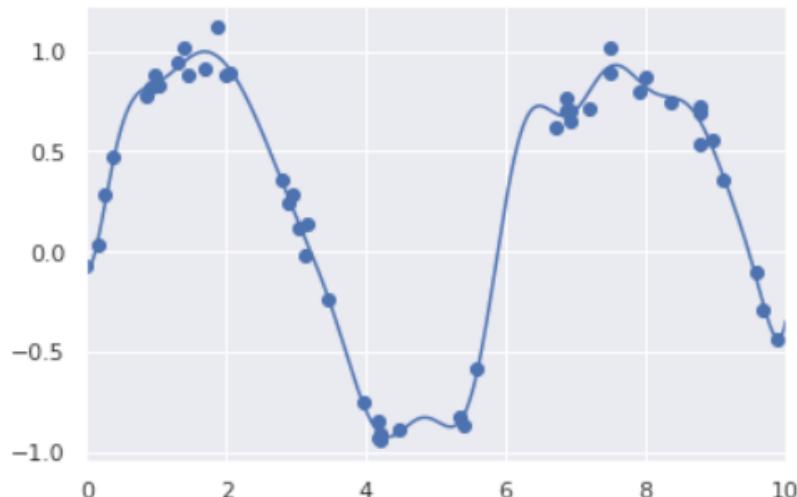
<https://colab.research.google.com/drive/1BVcHUBYDO4AlwldcKmpmj5blAbf7ISJb?usp=sharing>

Modified from:

<https://jakevdp.github.io/PythonDataScienceHandbook/05.06-linear-regression.html>

```
gauss_model = make_pipeline(GaussianFeatures(20),
                             LinearRegression())
gauss_model.fit(x[:, np.newaxis], y)
yfit = gauss_model.predict(xfit[:, np.newaxis])

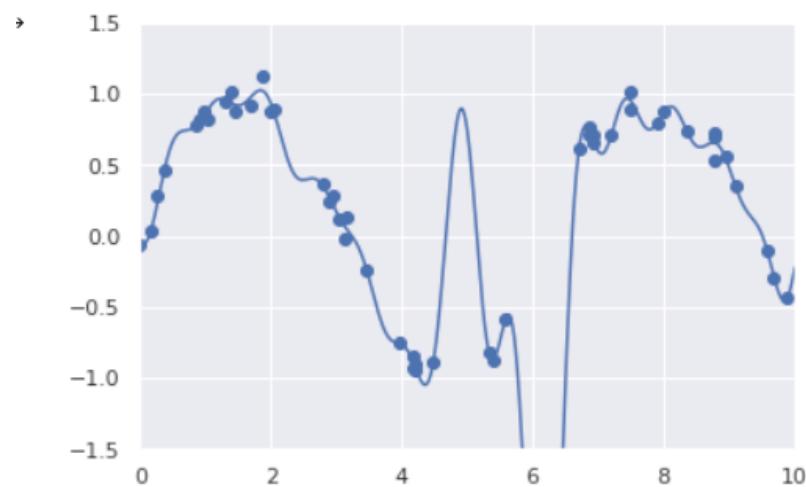
plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.xlim(0, 10);
```



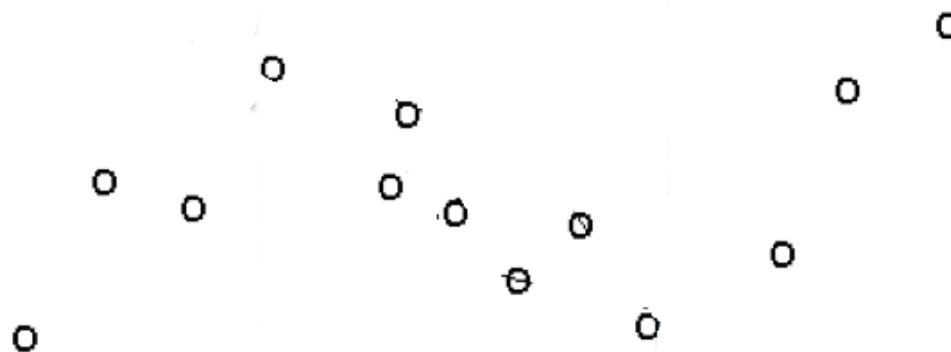
```
model = make_pipeline(GaussianFeatures(30),
                      LinearRegression())
model.fit(x[:, np.newaxis], y)

plt.scatter(x, y)
plt.plot(xfit, model.predict(xfit[:, np.newaxis]))

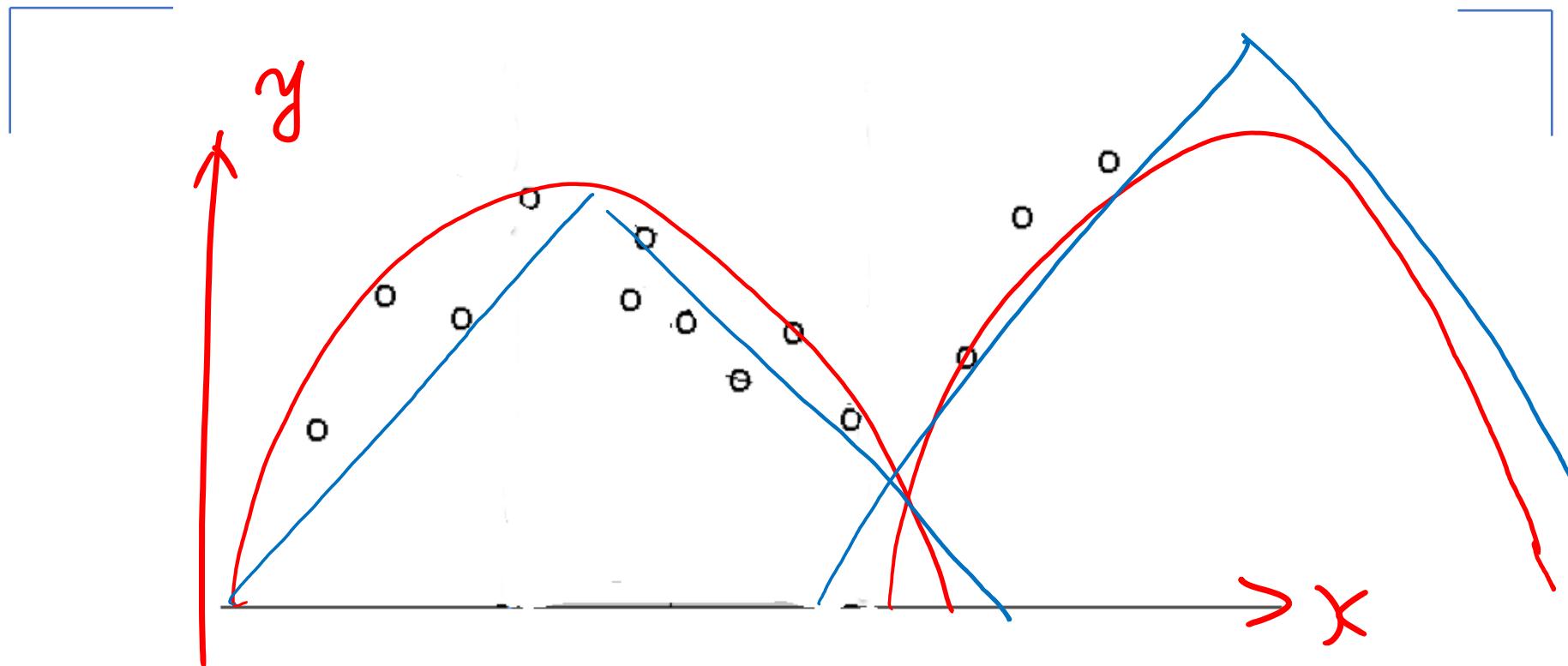
plt.xlim(0, 10)
plt.ylim(-1.5, 1.5);
```



Extra: even more possible Basis Function:
RBF, or Piecewise Linear based?



e.g. Even more possible Basis Func?



Thank You

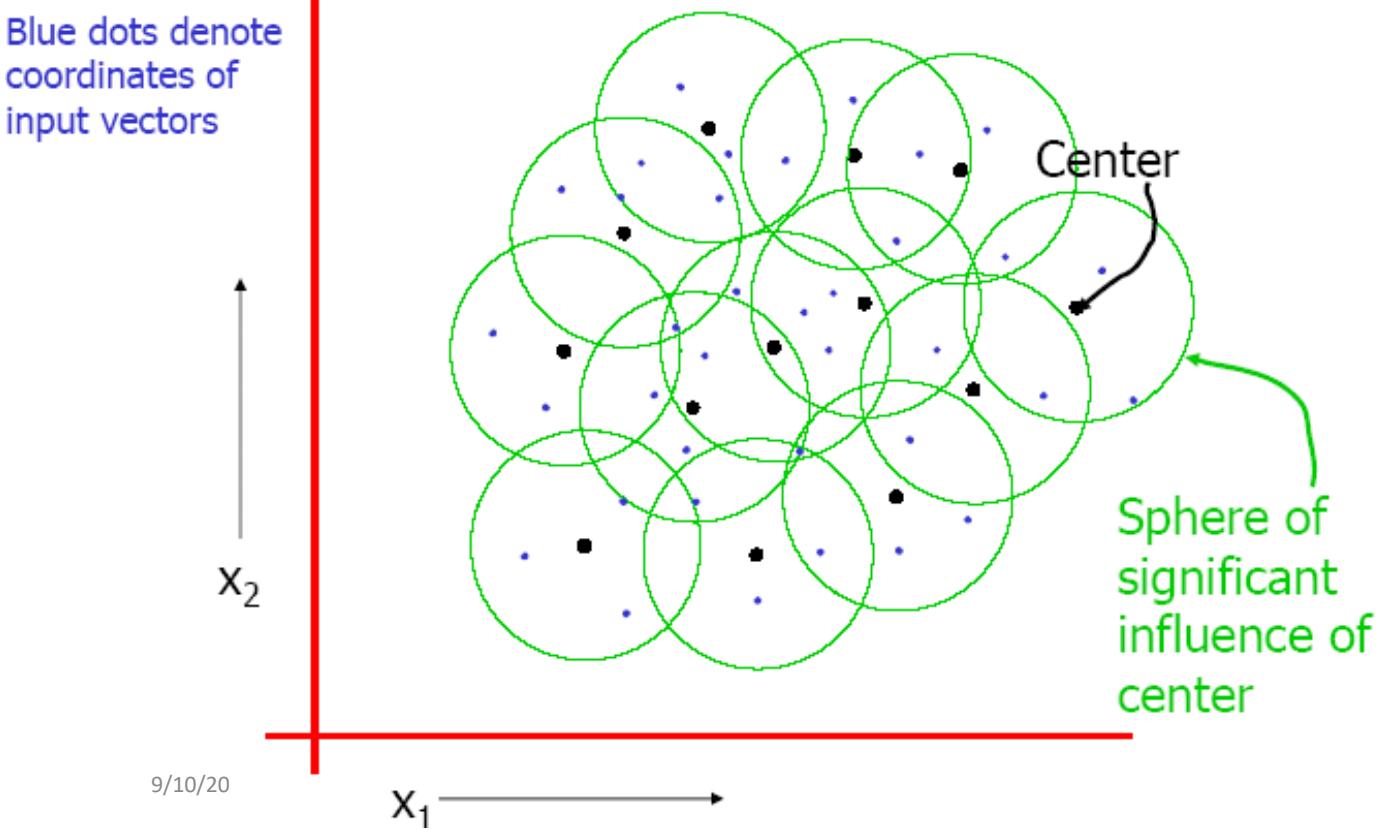


Extra

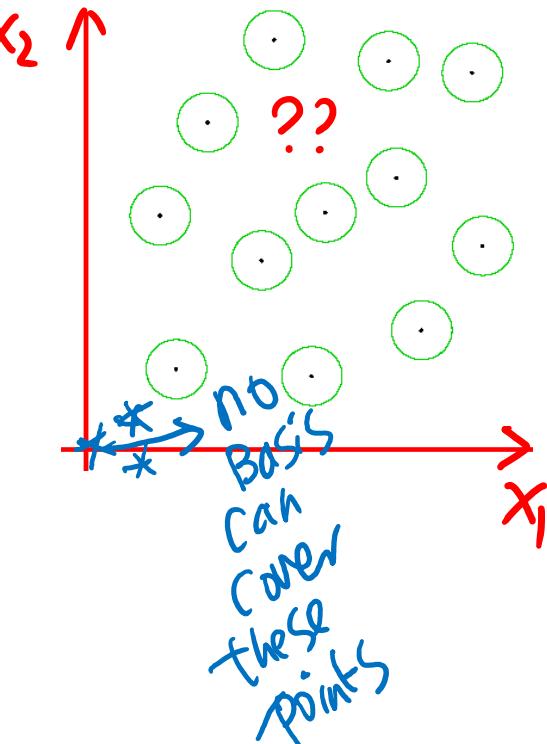
Extra: e.g. 2D Good and Bad RBF Basis

Contour view

- A good set of 2D predefined RBF basis



- A Bad set of predefined 2D RBFs



Extra: Nonparametric Regression Models

- K-Nearest Neighbor (KNN) and Locally weighted linear regression are **non-parametric** algorithms.
- The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm
 - because it has a fixed, finite number of parameters which are fit to the data;
 - Once we've fit the θ and stored them away, we no longer need to keep the training data around to make future predictions.
 - In contrast, to make predictions using KNN or locally weighted linear regression, we need to keep the entire training set around.
- The term "**non-parametric**" (roughly) refers to the fact that the **amount of knowledge we need to keep**, in order to represent the hypothesis grows with **linearly the size of the training set**.