Chapter 7
Error-based Learning

Prof. Chang-Chieh Cheng

Dept. Computer Science

National Chiao Tung University, Taiwan

Overview

- The main concept of error-based models
  - Given a set of data instances $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$ and each data instance $\mathbf{x}_i$ has a target y_i
  - Design an error-based model *M* with a set of parameters **w**.
  - *w* should be satisfied the following equation

$$\mathbf{w} = \arg\min_{\mathbf{w}'} \sum_{i=1}^{m} D(y_i, M_{\mathbf{w}'}(\mathbf{x}_i))$$

    - where *D(x, y)* is the distance between *x* and *y*.

  - Therefore, we can predict a query **q** by the model M with *w*.

$$y_q = M_{\mathbf{w}}(\mathbf{q})$$

Overview

- How to choose a model to fit the training data?
  - Line
  - Polynomial
  - Nonlinear model
- How to decide *w*?
  - Gradient descent

Linear Model

- Single-variable linear equation
  - $\mathbf{x} = \{x\}$

$$y = w_0 + w_1 x$$

- Multi-variable linear equation
  - $\mathbf{x} = \{x_0, x_1, x_2, \ldots, x_n\}$, where $x_0 = 1$

$$y = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

$$= \sum_{j=0}^{n} w_j x_j$$

$$= \mathbf{w} \cdot \mathbf{x}$$

Linear Model

- Example
  - A dataset that includes office rental prices and a number of descriptive features for 10 Dublin city-centre offices.
    - Size(x) → Rental price(y)

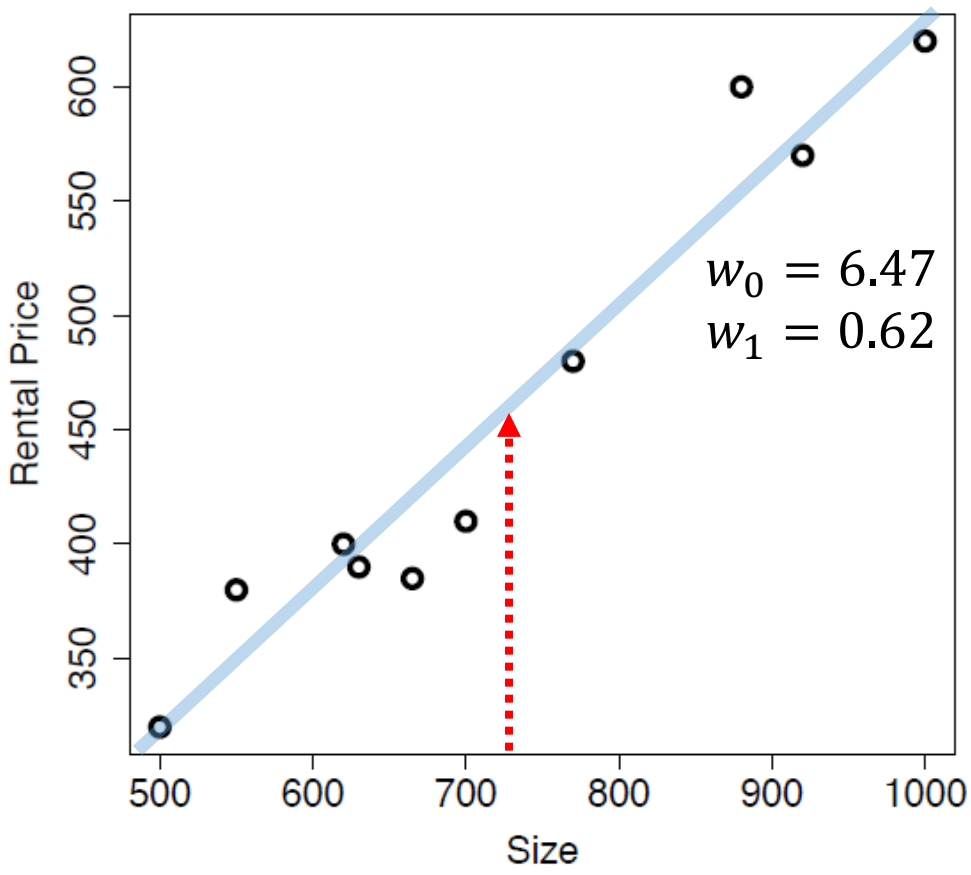
ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

Linear Model

- Example
  - Size(x) - Rental price(y)

$$y = w_0 + w_1 x$$
$$w_0 = ?$$
$$w_1 = ?$$

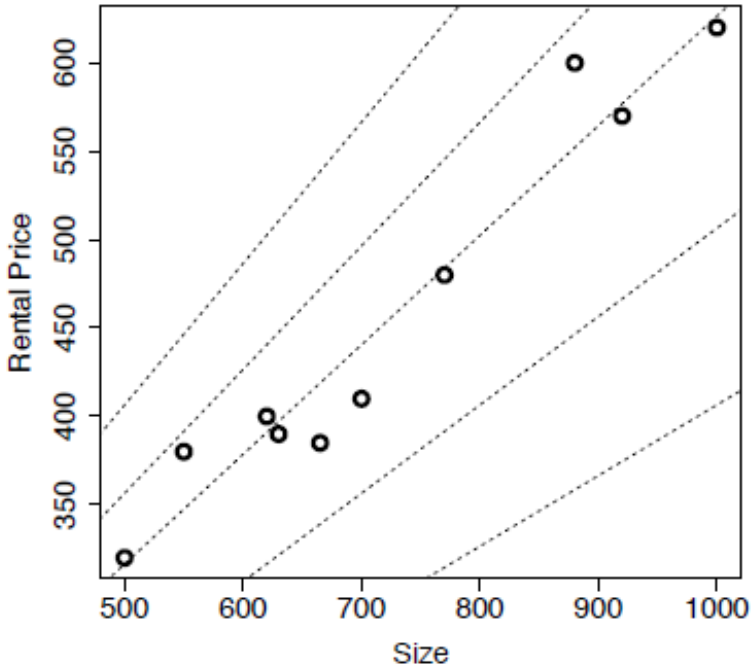
  - Query: *x* = 730, *y* = ?



Linear Model

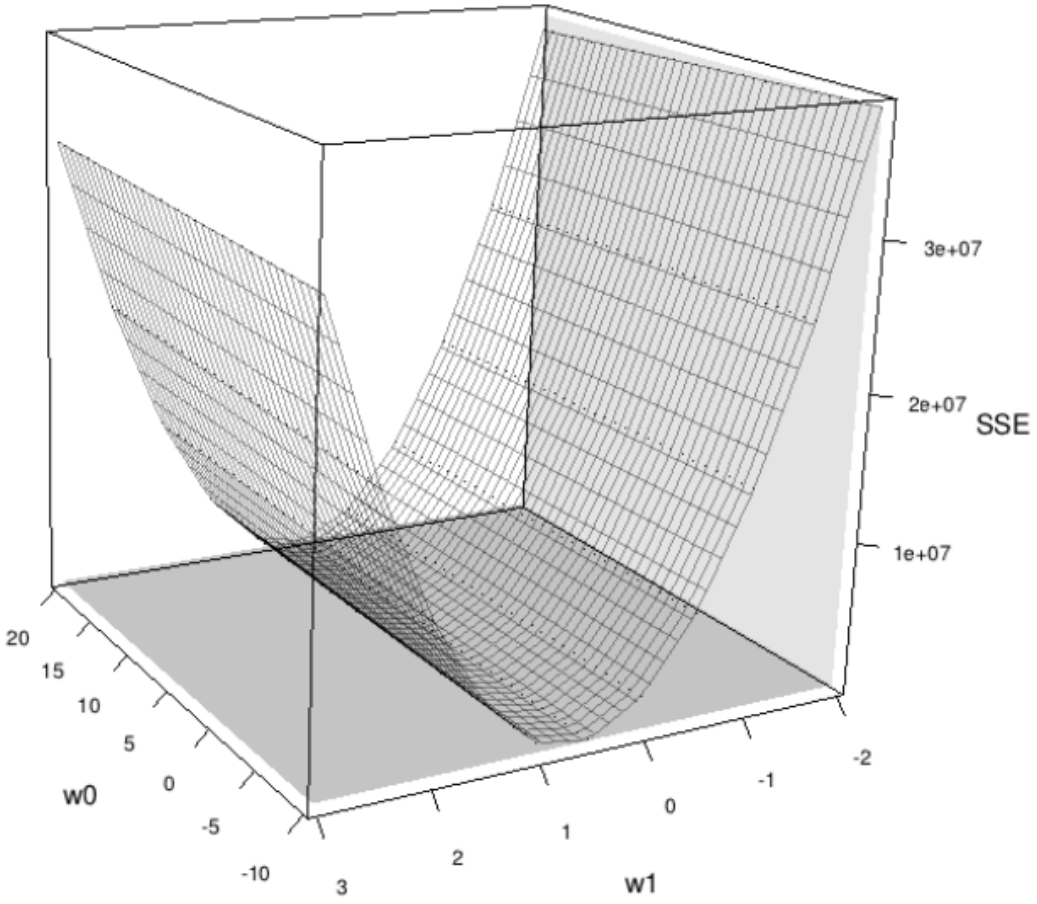
- L_2-norm minimization
- Let $\mathbf{w} = [w_0, w_1]$

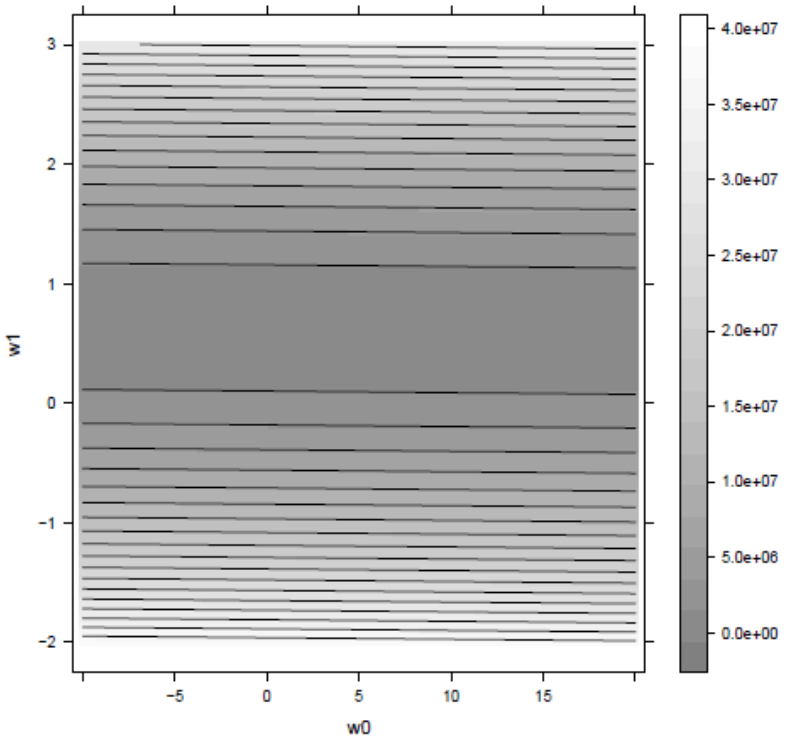
$$\mathbf{w} = \arg\min_{\mathbf{w}'} \frac{1}{2} \sum_{i=1}^{m} (y_i - (w'_0 + w'_1 x_i))^2$$



Linear Model

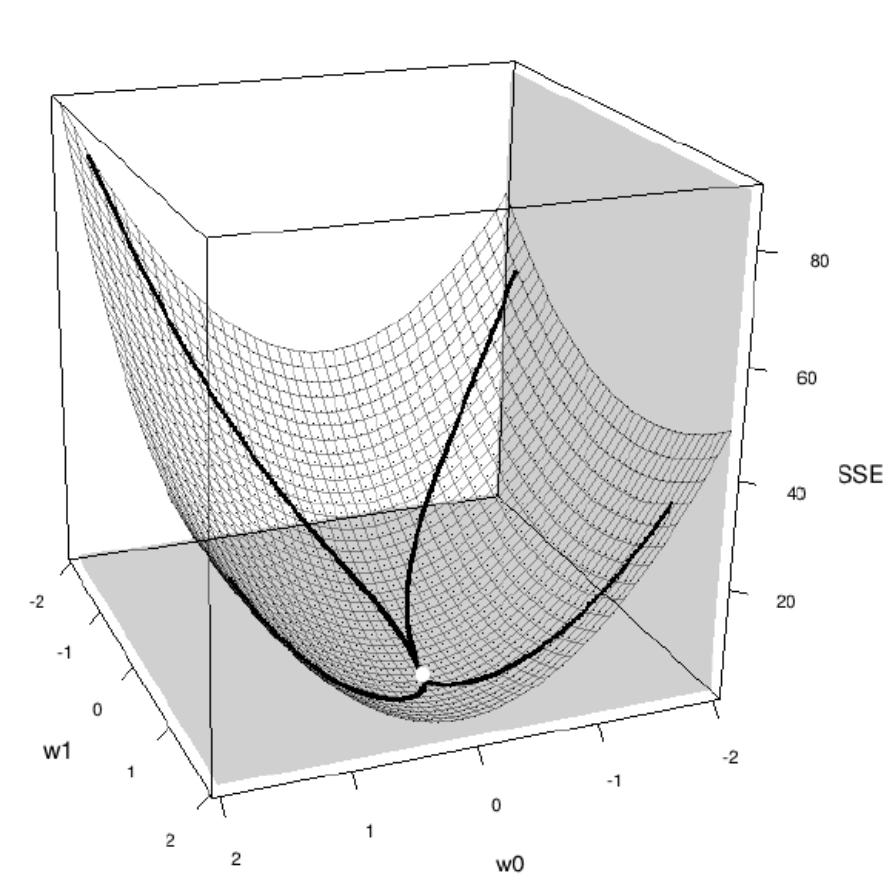
- Error surface

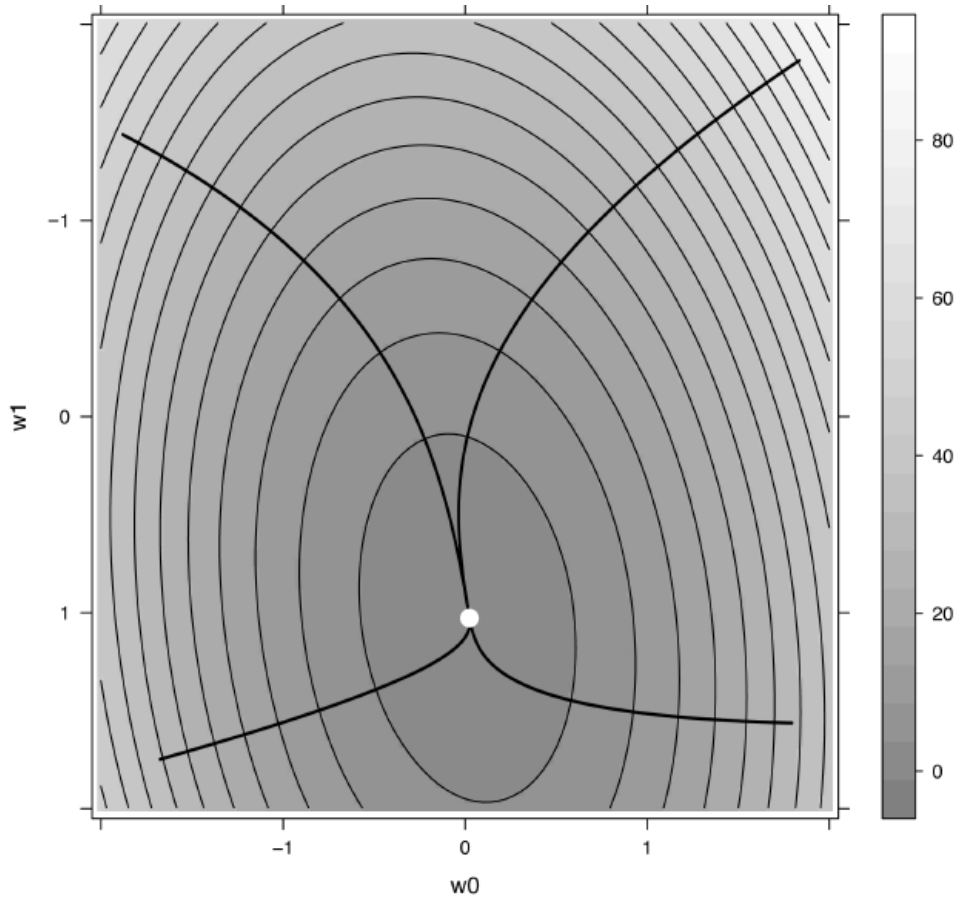




Linear Model

- Error surface





Finding a **w** such that its error is minimum

Linear Model

- If the minimum is existed in the error surface

$$\mathbf{w} = \arg\min_{\mathbf{w}'} \frac{1}{2} \sum_{i=1}^{m} (y_i - (w'_0 + w'_1 x_i))^2$$

$$\rightarrow \frac{d \frac{1}{2} \sum_{i=1}^{m} (y_i - (w_0 + w_1 x_i))^2}{d\mathbf{w}} = \mathbf{0}$$

Linear Model

- Multi-variable linear model
  - $\mathbf{x} = \{x_0, x_1, x_2, \ldots, x_n\}$, where $x_0 = 1$
  - $\mathbf{w} = \{w_0, w_1, w_2, \ldots, w_n\}$

$$\mathbf{w} = \arg\min_{\mathbf{w}'} \frac{1}{2} \sum_{i=1}^{m} (y_i - \mathbf{w}' \cdot \mathbf{x})^2$$

$$\rightarrow \frac{d\frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\cdot\mathbf{x})^2}{d\mathbf{w}} = \mathbf{0}$$

Gradient Descent

$$\frac{\partial \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\cdot\mathbf{x})^2}{\partial w_j}$$

$$\frac{\partial f(x,y)g(x,y)}{\partial x}$$

$$= \frac{\partial f(x,y)}{\partial x} g(x,y) + f(x,y)\frac{\partial g(x,y)}{\partial x}$$

- If $m = 1$

$$\frac{\partial \frac{1}{2}(y - \mathbf{w}\cdot\mathbf{x})^2}{\partial w_j} = (y - \mathbf{w}\cdot\mathbf{x})\frac{\partial(y - \mathbf{w}\cdot\mathbf{x})}{\partial w_j}$$

$$= (y - \mathbf{w}\cdot\mathbf{x})\frac{\partial\left(y - w_0x_0 - w_1x_1 - \cdots - w_jx_j - \cdots - w_nx_n\right)}{\partial w_j}$$

$$= -x_j(y - \mathbf{w}\cdot\mathbf{x})$$

Gradient Descent

- For $m > 1$

$$\frac{\partial \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\cdot\mathbf{x})^2}{\partial w_j}$$

$$= \sum_{i=1}^{m} \frac{\partial \frac{1}{2}(y_i - \mathbf{w}\cdot\mathbf{x})^2}{\partial w_j}$$

$$= \sum_{i=1}^{m} -x_{ij}(y_i - \mathbf{w}\cdot\mathbf{x}_i)$$

Gradient Descent

- To minimize to error

$$w_j' = -\frac{\partial \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}\cdot\mathbf{x})^2}{\partial w_j}$$

- ➔ $$w_j' = \sum_{i=1}^{m} x_{ij}(y_i - \mathbf{w}\cdot\mathbf{x}_i)$$

  - Therefore, increasing w_j by w'_j can let total error to approach zero

    - where α is an adjustment factor called **learning rate**

$$w_j = w_j + \alpha w_j'$$

Gradient Descent

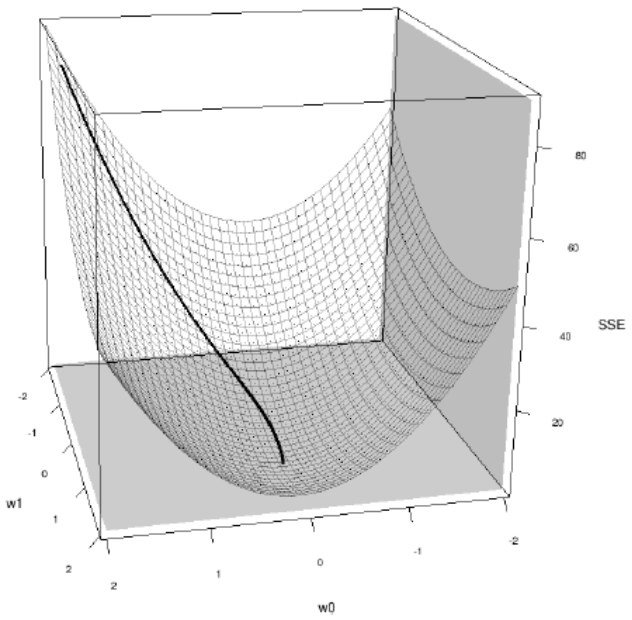
- Algorithm
  - **Data set:** $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$
    - each $\mathbf{x}_i = \{x_{i0}, x_{i1}, x_{i2}, \ldots, x_{in}\}$, where $x_{i0} = 1$
  - $\mathbf{w} = \{w_0, w_1, w_2, \ldots, w_n\}$

1. Initializing **w**
2. *k* = 0
3. do
4. for *j* = 0 to *n*
5. $$w_j' = \sum_{i=1}^{m} x_{ij}(y_i - \mathbf{w} \cdot \mathbf{x}_i) = 0$$
6. $$w_j = w_j + \alpha w_j'$$
7. *k* = *k* + 1
8. until all w_j' < a small value or *k* ≥ the maximum iterations

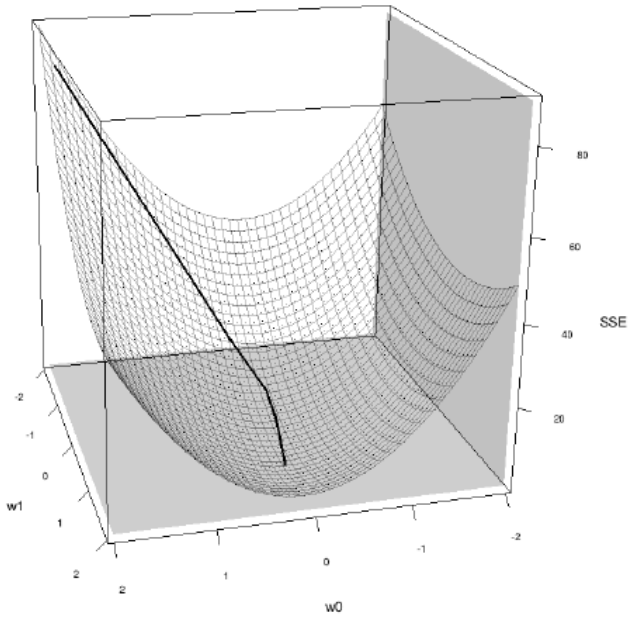
Gradient Descent

- Learning rate α is difficult to choose

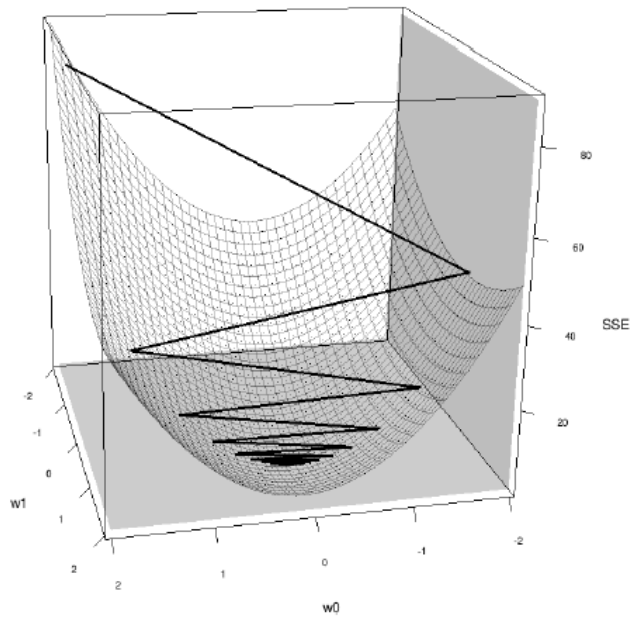
α is too small



Appropriate α



α is too large



- A typical range of α is [0.00001, 10]

Gradient Descent

- Initial value of $\mathbf{w}$
  - choosing random value uniformly from the range [-0.2, 0.2]

Gradient Descent

- Example
  - A dataset that includes office rental prices and a number of descriptive features for 10 Dublin city-centre offices.
    - x_1: Size, x_2: Floor, x_3: Broadband rate,
    - $w_0 + w_1x_1 + w_2x_2 + w_2x_2$ = Rental price

ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

Gradient Descent

- Example
  - α = 0.00000002
  - $\mathbf{w} = [-0.146 \quad 0.185 \quad -0.044 \quad 0.119]$
  - SSE = 1067571.59

$$w_1' = \sum_{i=1}^{m} x_{i1}(y_i - \mathbf{w} \cdot \mathbf{x}_i) = 2412074$$

$$\alpha w_1' = 0.23324148$$

$$w_1 = w_1 + \alpha w_j' = 0.185 + 0.23324148 = 0.233$$

Gradient Descent

- Example
  - α = 0.00000002
  - $\mathbf{w} = [-0.146 \quad \mathbf{0.233} \quad -0.044 \quad 0.119]$
  - SSE = 886723.04

$$w_1' = \sum_{i=1}^{m} x_{i1}(y_i - \mathbf{w} \cdot \mathbf{x}_i) = 2195616.08$$

$$w_1 = w_1 + \alpha w_j' = 0.27691232$$

  - ….
  - After 100 iterations
    - $\mathbf{w} = [-0.1513 \quad 0.6270 \quad -0.1781 \quad 0.0714]$
    - SSE = 2913.5

Gradient Descent

- Which one is good?
  - Each iteration only updates w_j
  - Each iteration updates **w**

1. Initializing **w**
2. $k = 0$, $\mathbf{w}' = [0]$
3. do
4. for $j = 0$ to n
5. $$w_j' = \sum_{i=1}^{m} x_{ij}(y_i - \mathbf{w} \cdot \mathbf{x}_i) = 0$$
6. $$\mathbf{w} = \mathbf{w} + \alpha \mathbf{w}'$$
7. $k = k + 1$
8. until $|\mathbf{w}'|$ < a small value or $k \geq$ the maximum iterations

Gradient Descent

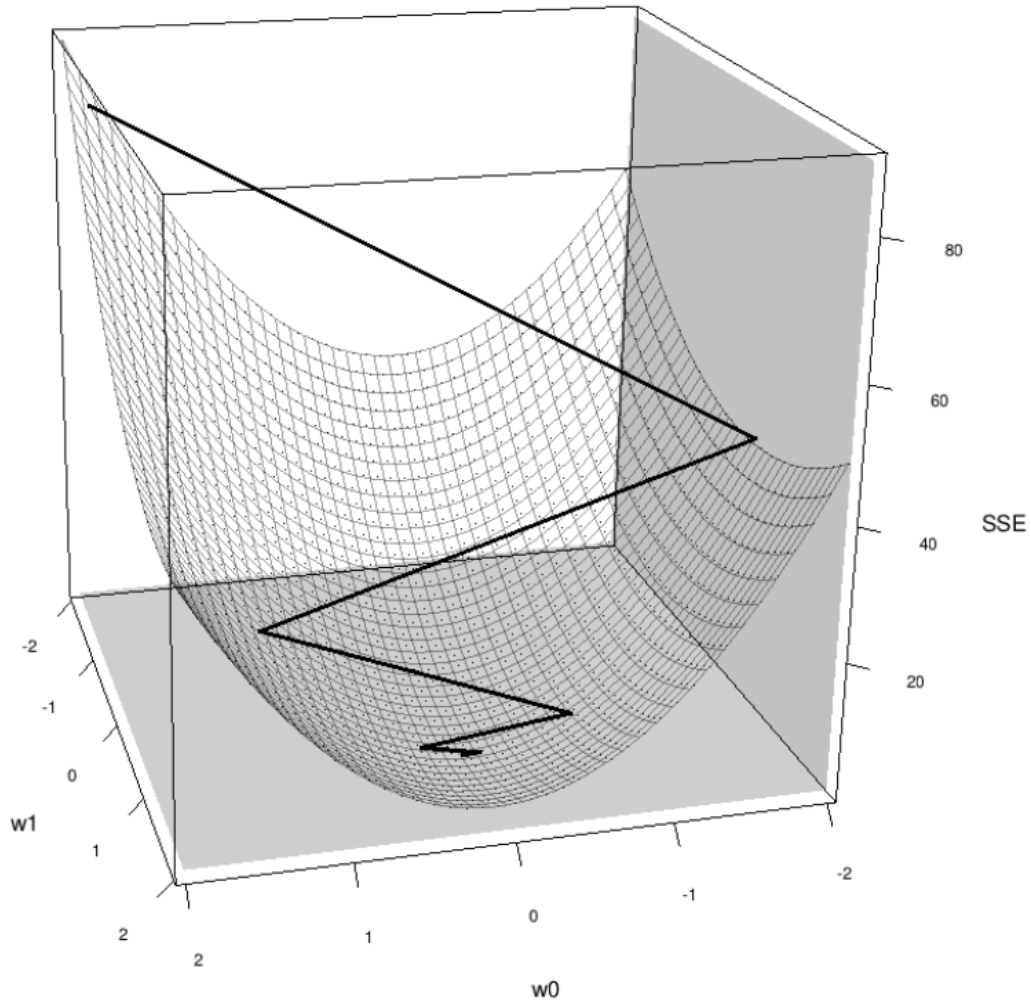
- Learning rate decay

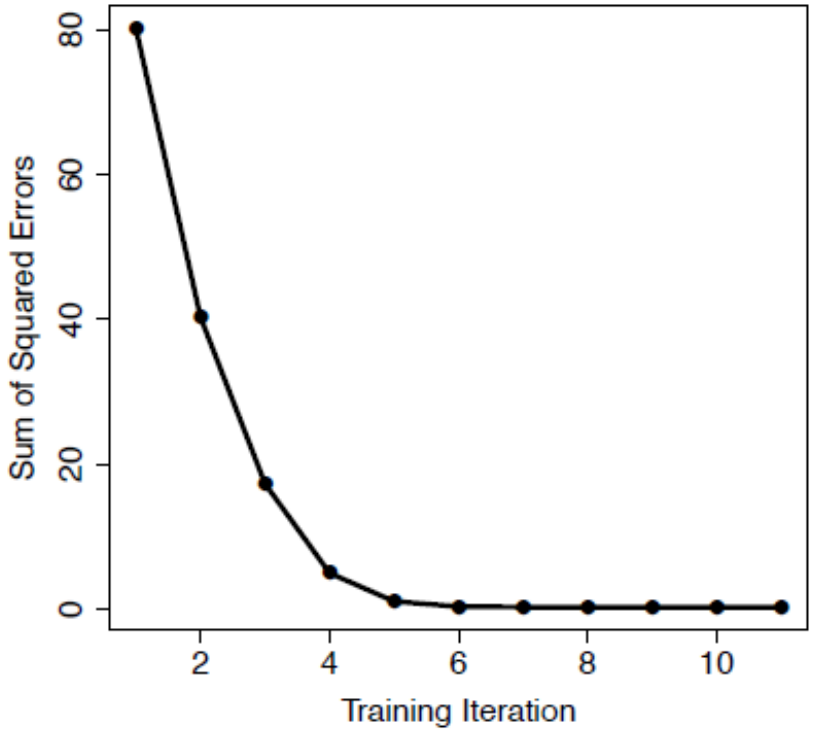
$$\alpha_k = \alpha_0 \frac{c}{c + k}$$

  - where *k* is the number of iteration and *c* is a constant number

Gradient Descent

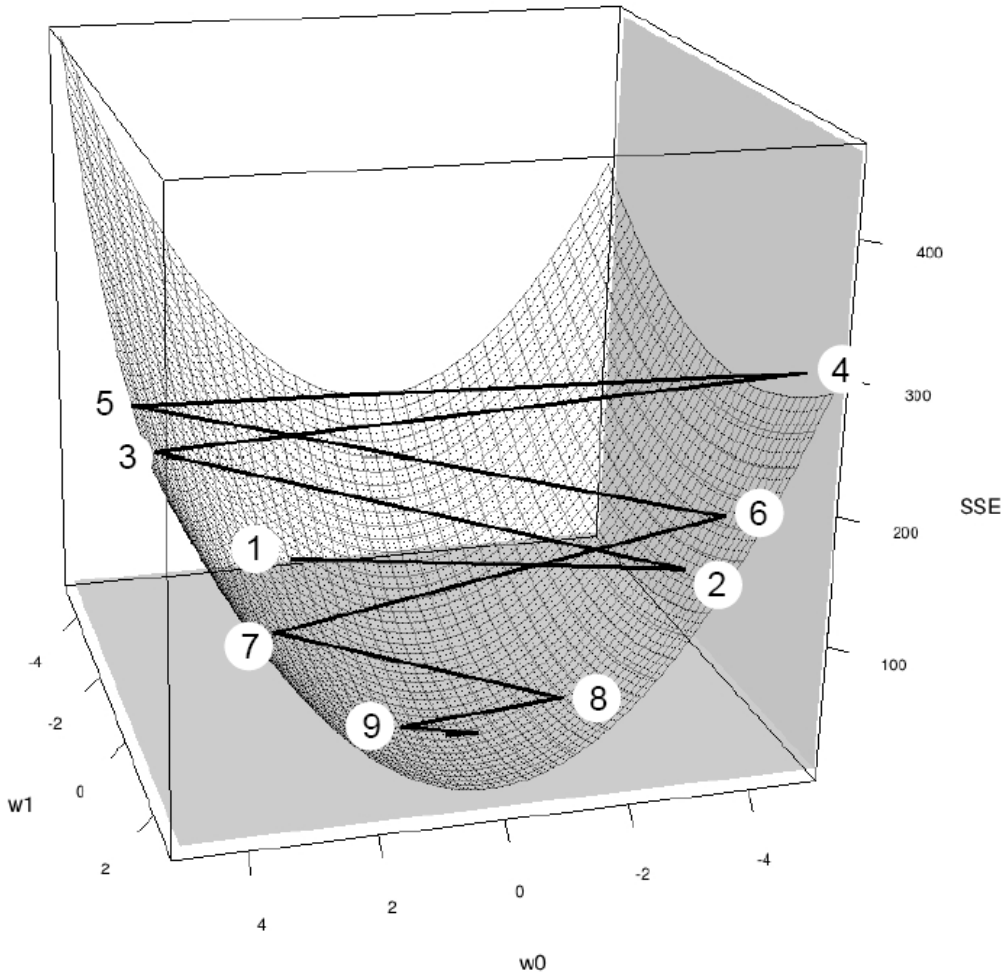
- Learning rate decay
  - $\alpha_0 = 0.18$, $C = 10$

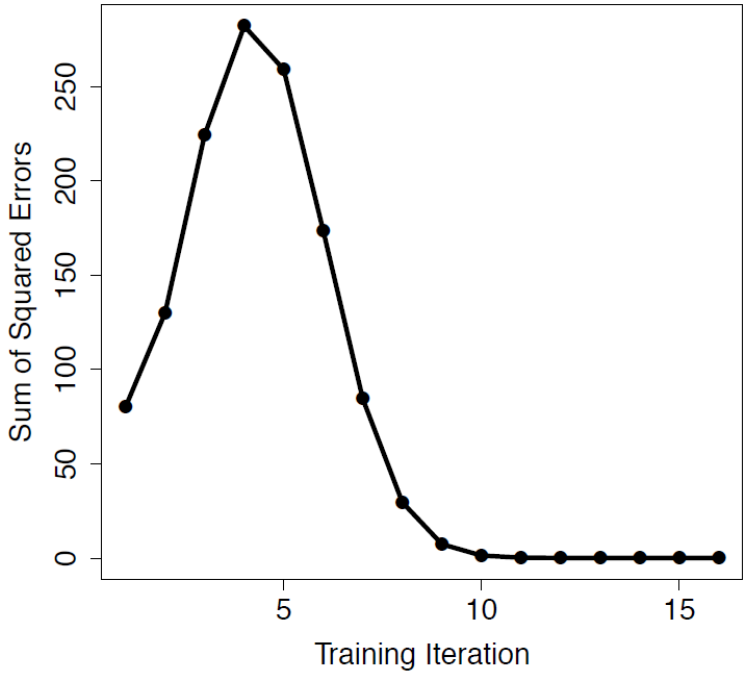




Gradient Descent

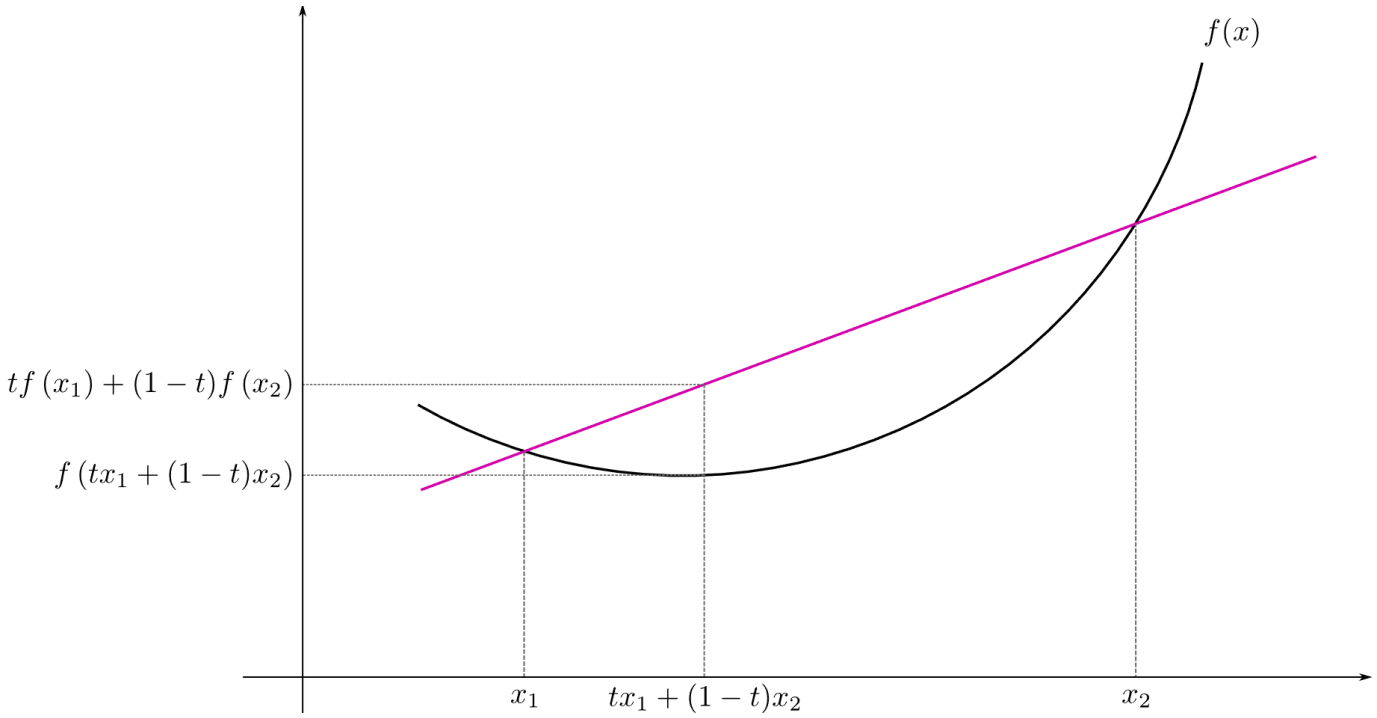
- Learning rate decay
  - $\alpha_0 = 0.25$, C = 100





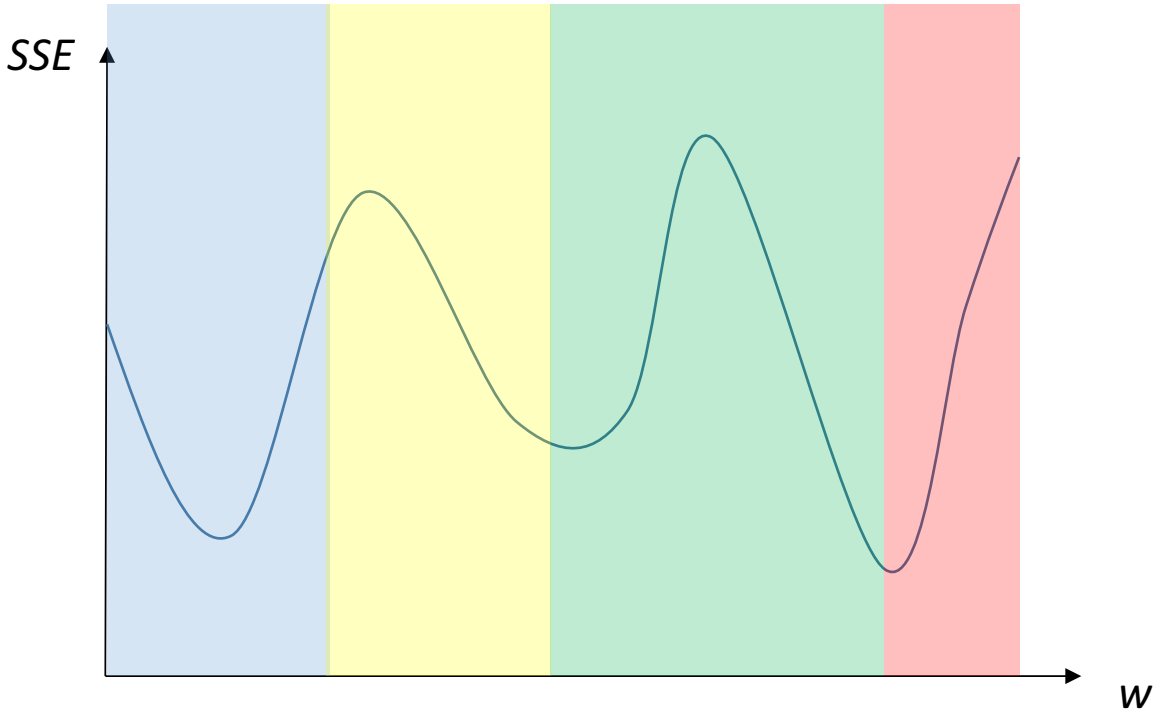
Gradient Descent

- Convex
  - A real-valued function defined on an n-dimensional interval is called convex if the line segment between any two points on the graph of the function lies above or on the graph, in a Euclidean space of at least two dimensions.



Gradient Descent

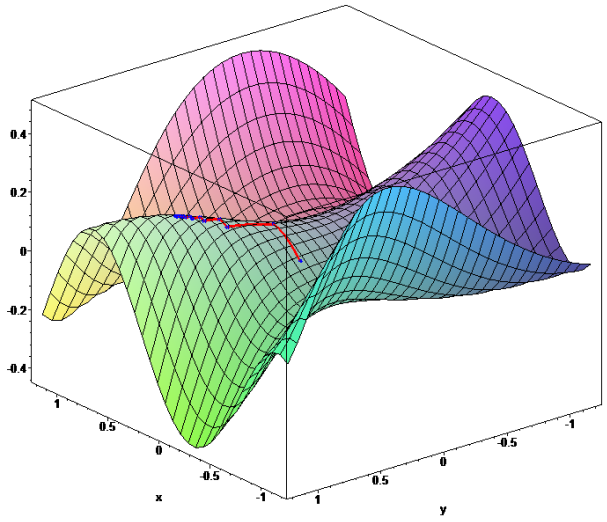
- If the error surface is convex, the gradient descent can find the unique answer
- Otherwise, the answer may not be the best.

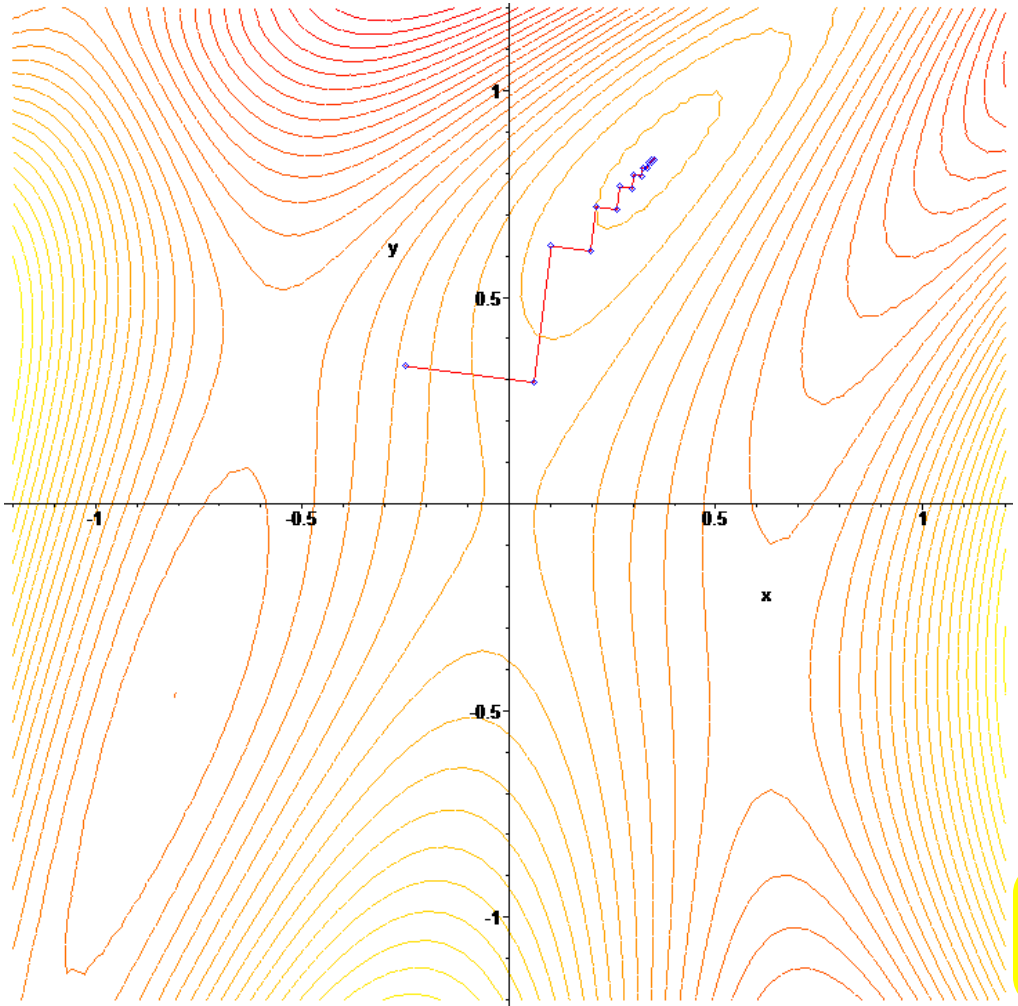


Gradient Descent

Example:

$$f(x,y) = \sin\left(\frac{x^2}{2} - \frac{y^2}{4} + 3\right)\cos(2x + 1 - e^y)$$





Gradient Descent

- Categorical Descriptive Features
  - Gradient descent requires that all value must be continuous
    - ➔Differentiable
  - For example, ENERGY RATING has three levels: A, B, and C
    - ➔ Create three binary features to stand for three levels

ID	Size	Floor	Broadband Rate	Energy Rating A	Energy Rating B	Energy Rating C	Rental Price
1	500	4	8	0	0	1	320
2	550	7	50	1	0	0	380
3	620	9	7	1	0	0	400
4	630	5	24	0	1	0	390
5	665	8	100	0	0	1	385
6	700	4	8	0	1	0	410
7	770	10	7	0	1	0	480
8	880	12	50	1	0	0	600
9	920	14	8	0	0	1	570
10	1 000	9	24	0	1	0	620

Gauss-Newton Method

- Taylor series
  - For any infinitely differentiable function

$$f(\mathbf{u}) \in \mathbf{R}, \mathbf{u} = [u_1, u_2, \cdots, u_n]^{\mathrm{T}} \in \mathbf{R}^{n\times 1}, n \geq 1$$

  - *f* can be expanded as a Taylor series

$$f(\mathbf{u}) = f(\mathbf{u}_s) + (\mathbf{u} - \mathbf{u}_s)^{\mathrm{T}} \frac{f'(\mathbf{u}_s)}{1!} + (\mathbf{u} - \mathbf{u}_s)^{\mathrm{T}} \frac{f''(\mathbf{u}_s)}{2!} (\mathbf{u} - \mathbf{u}_s) + \cdots$$

    - where $\mathbf{u}_s$ is any vector $\in \mathbf{R}^{n\times 1}$

Gauss-Newton Method

- If $\mathbf{u} = \mathbf{u}_s + \Delta\mathbf{u}$, the Taylor series of $f(\mathbf{u})$ can be written as follows

$$f(\mathbf{u}) = f(\mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} \frac{f'(\mathbf{u}_s)}{1!} + \Delta\mathbf{u}^{\mathrm{T}} \frac{f''(\mathbf{u}_s)}{2!} \Delta\mathbf{u} + \cdots$$

- Take the first order of the Taylor series of $f(\mathbf{u})$

$$f(\mathbf{u}) \approx f(\mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} f'(\mathbf{u}_s)$$

Gauss-Newton Method

- Jacobian matrix

$$f(\mathbf{u}) \approx f(\mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} f'(\mathbf{u}_s)$$
$$= f(\mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} \frac{\partial f(\mathbf{u}_s)}{\partial \mathbf{u}_s}$$
$$= f(\mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} \mathbf{J}(\mathbf{u}_s)$$

- where $\mathbf{J}(\mathbf{u}) = \left[\frac{\partial f(\mathbf{u})}{\partial u_1}, \frac{\partial f(\mathbf{u})}{\partial u_2}, \cdots, \frac{\partial f(\mathbf{u})}{\partial u_n} \right]^{\mathrm{T}}$

$\mathbf{J}$ is a form of the Jacobian matrix

Gauss-Newton Method

- Given two functions, $f(\mathbf{x}, \mathbf{u})$ and $g(\mathbf{x})$, where $\mathbf{x} \in \mathbf{R}^m, m \geq 1$
- Assume that we have r instances of $\mathbf{x}$
  - $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r$
- Define

$$E(\mathbf{u}) = \sum_{i=1}^{r} \left(f(\mathbf{x}_i, \mathbf{u}) - g(\mathbf{x}_i) \right)^2$$

- The Gauss-Newton method can find $\Delta\mathbf{u}$ with $\mathbf{u}_s$ such that $E(\mathbf{u}_s + \Delta\mathbf{u})$ is minimum

$$E(\mathbf{u}_s + \Delta\mathbf{u}) = \sum_{i=1}^{r} \left(f(\mathbf{x}_i, \mathbf{u}_s + \Delta\mathbf{u}) - g(\mathbf{x}_i) \right)^2$$

Gauss-Newton Method

- Applying the first order of Taylor series to expand $f(\mathbf{x}, \mathbf{u})$

$$E(\mathbf{u}_s + \Delta\mathbf{u}) \approx \sum_{i=1}^{r} \left(f(\mathbf{x}_i, \mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s) - g(\mathbf{x}_i) \right)^2$$

  - where

$$\mathbf{J}_{\mathbf{x}}(\mathbf{u}) = \left[\frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial u_1}, \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial u_2}, \cdots, \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial u_n} \right]^{\mathrm{T}}$$

Gauss-Newton Method

- Applying the first order of Taylor series to expand $f(\mathbf{x}, \mathbf{u})$

$$E(\mathbf{u}_s + \Delta\mathbf{u}) \approx \sum_{i=1}^{r} \left(f(\mathbf{x}_i, \mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s) - g(\mathbf{x}_i) \right)^2$$

- The differential of E is zero if E is minimum

$$\frac{\partial E(\mathbf{u}_s + \Delta\mathbf{u})}{\partial \Delta\mathbf{u}} = \frac{\partial \sum_{i=1}^{r} \left(f(\mathbf{x}_i, \mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s) - g(\mathbf{x}_i) \right)^2}{\partial \Delta\mathbf{u}} = 0$$

$$\Rightarrow 2 \sum_{i=1}^{r} \left(f(\mathbf{x}_i, \mathbf{u}_s) + \Delta\mathbf{u}^{\mathrm{T}} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s) - g(\mathbf{x}_i) \right) \mathbf{J}_{\mathbf{x}_i}{}^{\mathrm{T}}(\mathbf{u}_s) = 0$$

$$\Rightarrow \sum_{i=1}^{r} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s) \mathbf{J}_{\mathbf{x}_i}{}^{\mathrm{T}}(\mathbf{u}_s) \Delta\mathbf{u} = -\sum_{i=1}^{r} \left(f(\mathbf{x}_i, \mathbf{u}_s) - g(\mathbf{x}_i) \right) \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s)$$

Gauss-Newton Method

- Define the matrix H, which is similar to a Hessian matrix

$$\mathbf{H}(\mathbf{u}) = \sum_{i=1}^{r} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}) \mathbf{J}_{\mathbf{x}_i}{}^{T}(\mathbf{u})$$

$$= \begin{bmatrix} \sum_{i=1}^{r} \left(\frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_1} \right)^2 & \sum_{i=1}^{r} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_1} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_2} & \cdots & \sum_{i=1}^{r} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_1} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_n} \\ \sum_{i=1}^{r} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_2} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_1} & \sum_{i=1}^{r} \left(\frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_2} \right)^2 & \cdots & \sum_{i=1}^{r} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_2} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{r} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_n} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_2} & \sum_{i=1}^{r} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_n} \frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_2} & \cdots & \sum_{i=1}^{r} \left(\frac{\partial f(\mathbf{x}_i, \mathbf{u})}{\partial u_n} \right)^2 \end{bmatrix}$$

Gauss-Newton Method

- Let

$$\mathbf{G}(\mathbf{u}) = -\sum_{i=1}^{r} \left(f\left(\mathbf{x}_i, \mathbf{u}\right) - g(\mathbf{x}_i) \right) \mathbf{J}_{\mathbf{x}_i}(\mathbf{u})$$

- We then can find the $\Delta\mathbf{u}$

$$\sum_{i=1}^{r} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s) \mathbf{J}_{\mathbf{x}_i}{}^{T}(\mathbf{u}_s) \Delta\mathbf{u} = -\sum_{i=1}^{r} \left(f\left(\mathbf{x}_i, \mathbf{u}_s\right) - g(\mathbf{x}_i) \right) \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}_s)$$

$$\Rightarrow \Delta\mathbf{u} = \mathbf{H}(\mathbf{u}_s)^{-1} \mathbf{G}(\mathbf{u}_s)$$

Gauss-Newton Method

- Although the first order of Taylor series is an approximation, $\Delta\mathbf{u}$ can be determined after several iterations (k).

$$\mathbf{u}_s^{k+1} = \mathbf{u}_s^k + \Delta\mathbf{u},$$

  - where is $\mathbf{u}_s^0$ an initial guess

- Therefore,

$$\Delta\mathbf{u} = \mathbf{H}(\mathbf{u}_s^k)^{-1}\mathbf{G}(\mathbf{u}_s^k)$$

Gauss-Newton Method

- Levenberg–Marquardt algorithm
  - K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares". *Quarterly of Applied Mathematics*. 2: 164–168, 1944.
  - D. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". *SIAM Journal on Applied Mathematics*. 11 (2): 431–441, 1963.

$$(\mathbf{H}(\mathbf{u}_s) + \lambda\mathbf{I})\,\Delta\mathbf{u} = \mathbf{G}(\mathbf{u}_s)$$

  - where λ is the damping factor
    - It is large in the 1st iteration.
    - It may be decayed in the next iteration

Gauss-Newton Method

- Image Alignment by Gauss-Newton Method
  - B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence, IJCAI 1981*, Vancouver, Canada, 1981, pp. 674-679.
  - R. Szeliski and H. Y. Shum, "Creating full view panoramic image mosaics and environment maps," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 251-258
  - S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, vol. 56, pp. 221-255, Feb. 2004.
  - R. Szeliski, “Image alignment and stitching: a tutorial,” *Found. Trends. Comput. Graph. Vis.*, vol. 2, no. 1, pp. 1-104, Dec. 2006.

Gauss-Newton Method

- Image Alignment by Gauss-Newton Method
  - Given two images $I_A(\mathbf{x})$ and $I_B(\mathbf{x})$ of r pixels, where $\mathbf{x} \in R^2$ is a 2D coordinate of a pixel
  - And given a transformation function $T(\mathbf{x}, \mathbf{u}) \in R^2$, where $\mathbf{u}$ is the transformation parameter, $\mathbf{u} \in R^n$, $n \geq 1$
  - Define an error function E

$$E(\mathbf{u}) = \sum_{i=1}^{r} \left[I_A\left(T(\mathbf{x}_i, \mathbf{u})\right) - I_B(\mathbf{x}_i) \right]^2$$

  - To find the $\mathbf{u}$ such that $E(\mathbf{u})$ is minimum, we can use Gauss-Newton method with an initial guess $\mathbf{u}_s^0$.

$$E(\mathbf{u}_s^0 + \Delta\mathbf{u}) = \sum_{i=1}^{r} \left[I_A\left(T(\mathbf{x}_i, \mathbf{u}_s^0 + \Delta\mathbf{u})\right) - I_B(\mathbf{x}_i) \right]^2$$

$$\Rightarrow \Delta\mathbf{u} = \mathbf{H}(\mathbf{u}_s^0)^{-1}\mathbf{G}(\mathbf{u}_s^0)$$

Gauss-Newton Method

- Image Alignment by Gauss-Newton Method
  - H and G are defined as follows

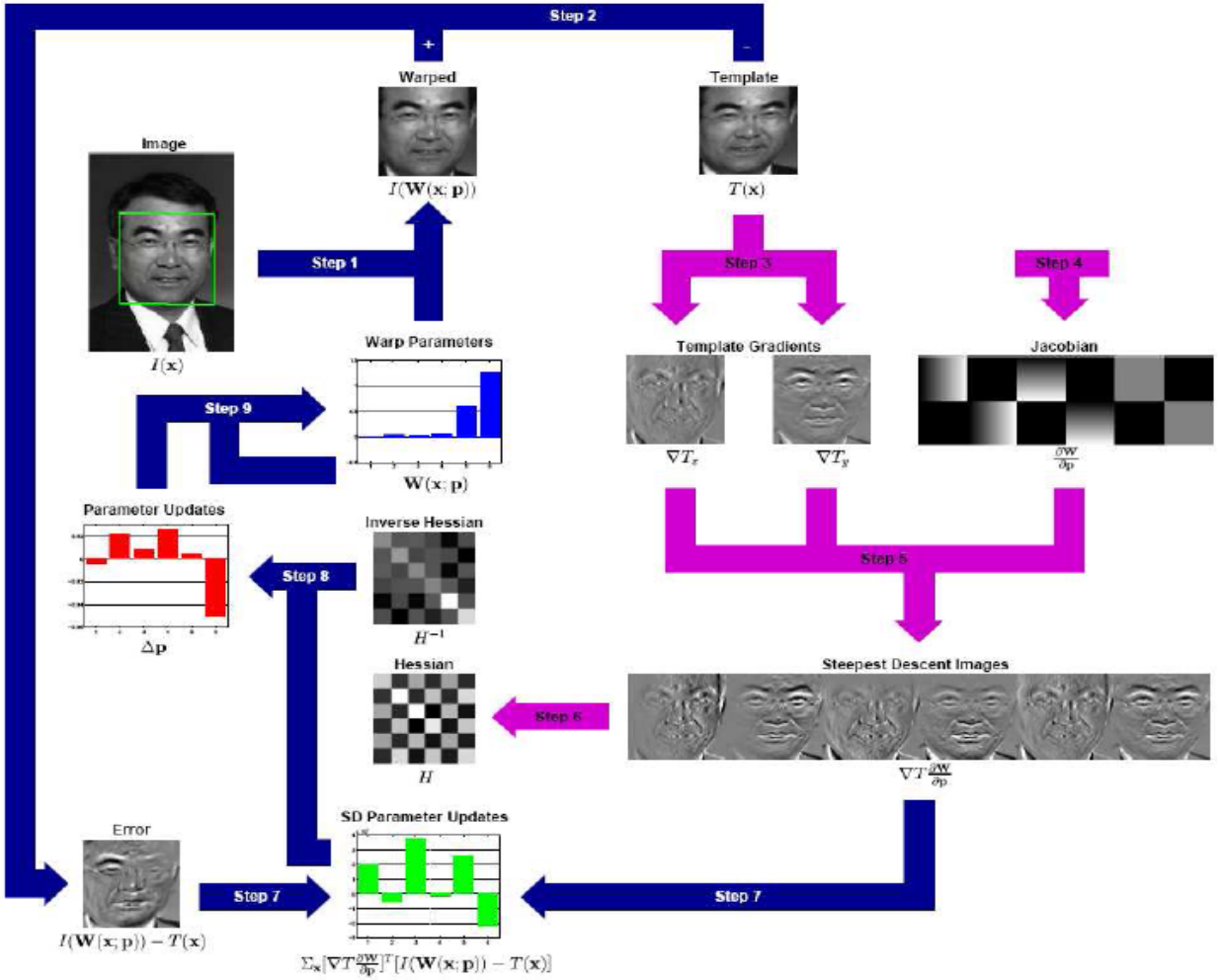
$$\mathbf{H}(\mathbf{u}) = \sum_{i=1}^{r} \mathbf{J}_{\mathbf{x}_i}(\mathbf{u}) \mathbf{J}_{\mathbf{x}_i}{}^{T}(\mathbf{u}) = \begin{bmatrix} \sum_{i=1}^{r} \left(\frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_1} \right)^2 & \sum_{i=1}^{r} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_1} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_2} & \cdots & \sum_{i=1}^{r} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_1} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_n} \\ \sum_{i=1}^{r} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_2} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_1} & \sum_{i=1}^{r} \left(\frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_2} \right)^2 & \cdots & \sum_{i=1}^{r} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_2} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{r} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_n} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_1} & \sum_{i=1}^{r} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_n} \frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_2} & \cdots & \sum_{i=1}^{r} \left(\frac{\partial I_A(T(\mathbf{x}_i,\mathbf{u}))}{\partial u_n} \right)^2 \end{bmatrix}$$

$$\mathbf{G}(\mathbf{u}) = -\sum_{i=1}^{r} \left(I_A(\mathbf{x}_i,\mathbf{u}) - I_B(\mathbf{x}_i) \right) \mathbf{J}_{\mathbf{x}_i}(\mathbf{u})$$

$$= -\sum_{i=1}^{r} \left(I_A(\mathbf{x}_i,\mathbf{u}) - I_B(\mathbf{x}_i) \right) \left[\frac{\partial I_A(\mathbf{x}_i,\mathbf{u})}{\partial u_1}, \frac{\partial I_A(\mathbf{x}_i,\mathbf{u})}{\partial u_2}, \cdots \frac{\partial I_A(\mathbf{x}_i,\mathbf{u})}{\partial u_n} \right]^{\mathrm{T}}$$

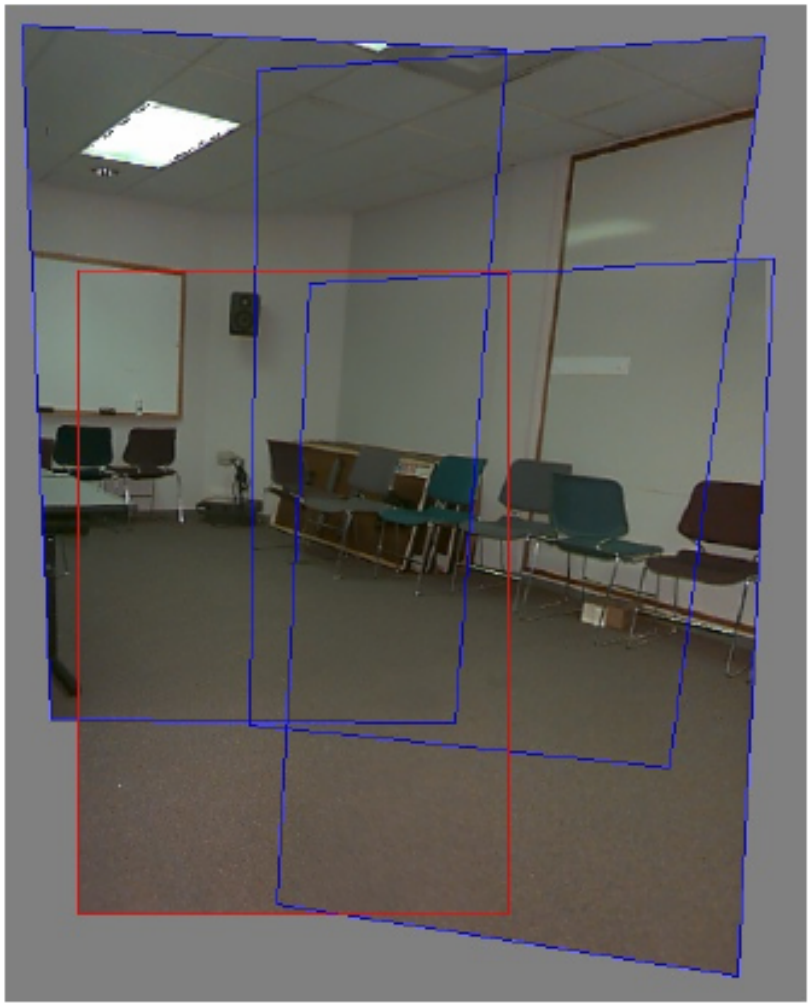
Gauss-Newton Method

- Image Alignment by Gauss-Newton Method
  - Lucas-Kanade algorithm



Gauss-Newton Method

- Image Alignment by Gauss-Newton Method



Gauss-Newton Method

- What if the error function is not L2-norm?
  - Second order of Taylor expansion
    - High computational time
  - Quasi-Newton method

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T B \, \Delta x,$$

    - where *B* is an approximation to the Hessian matrix

  - Broyden–Fletcher–Goldfarb–Shanno algorithm, BFGS

$$B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

$$B_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

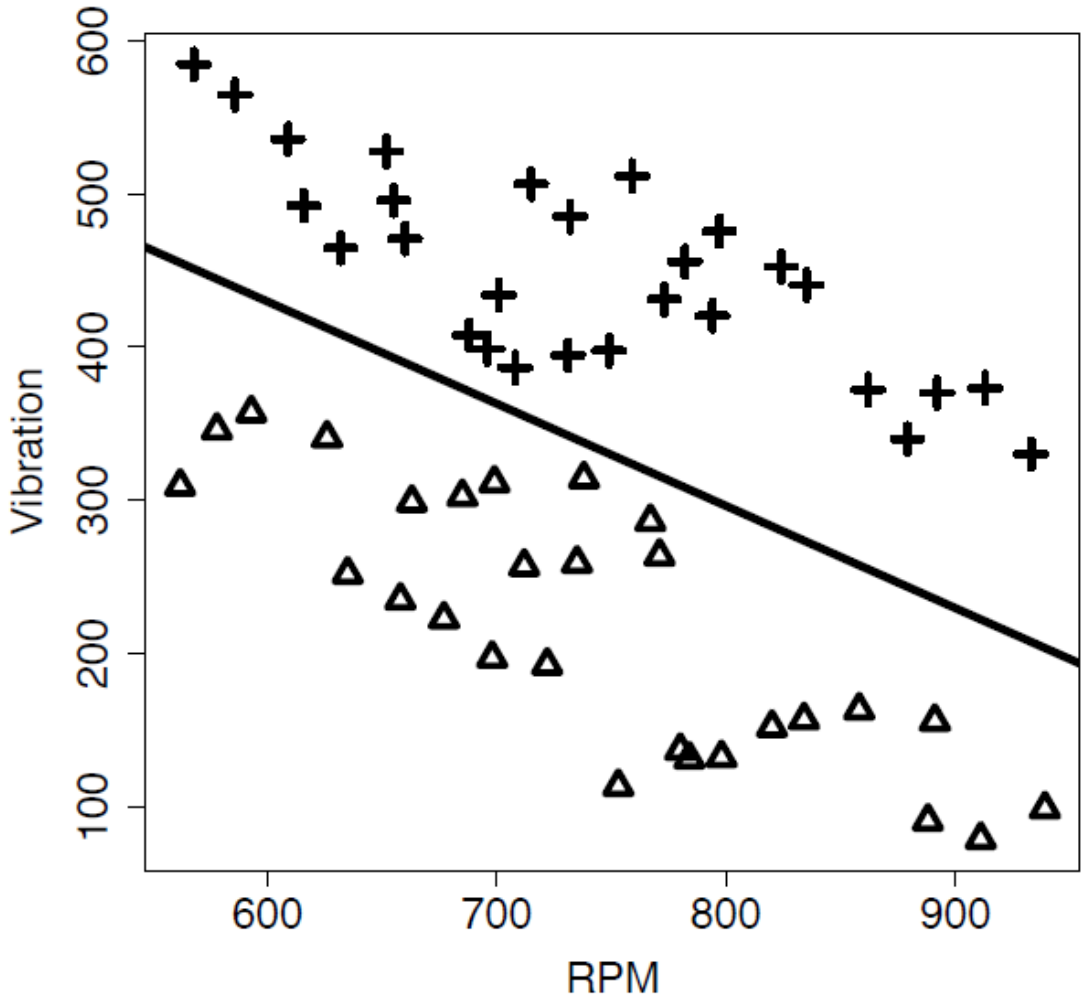
Decision surface

- Example: A dataset listing features for a number of generators

ID	RPM	VIBRATION	STATUS
1	568	585	good
2	586	565	good
3	609	536	good
4	616	492	good
5	632	465	good
6	652	528	good
7	655	496	good
8	660	471	good
9	688	408	good
10	696	399	good
11	708	387	good
12	701	434	good
13	715	506	good
14	732	485	good
15	731	395	good
16	749	398	good
17	759	512	good
18	773	431	good
19	782	456	good
20	797	476	good
21	794	421	good
22	824	452	good
23	835	441	good
24	862	372	good
25	879	340	good
26	892	370	good
27	913	373	good
28	933	330	good

ID	RPM	VIBRATION	STATUS
29	562	309	faulty
30	578	346	faulty
31	593	357	faulty
32	626	341	faulty
33	635	252	faulty
34	658	235	faulty
35	663	299	faulty
36	677	223	faulty
37	685	303	faulty
38	698	197	faulty
39	699	311	faulty
40	712	257	faulty
41	722	193	faulty
42	735	259	faulty
43	738	314	faulty
44	753	113	faulty
45	767	286	faulty
46	771	264	faulty
47	780	137	faulty
48	784	131	faulty
49	798	132	faulty
50	820	152	faulty
51	834	157	faulty
52	858	163	faulty
53	888	91	faulty
54	891	156	faulty
55	911	79	faulty
56	939	99	faulty

Decision surface



decision boundary

- +: Good
- Δ: Faulty

Decision surface

- Affine hyperplane
  - For a n-dimensional point $\mathbf{x} = [x_0\ x_1\ x_2\ \ldots\ x_n]$ in Euclidean space, where $x_0 = 1$
  - Let $\mathbf{w} = [w_0\ w_1\ \ldots\ w_n]$, at least one of the $w_1\ \ldots\ w_n$ is non-zero

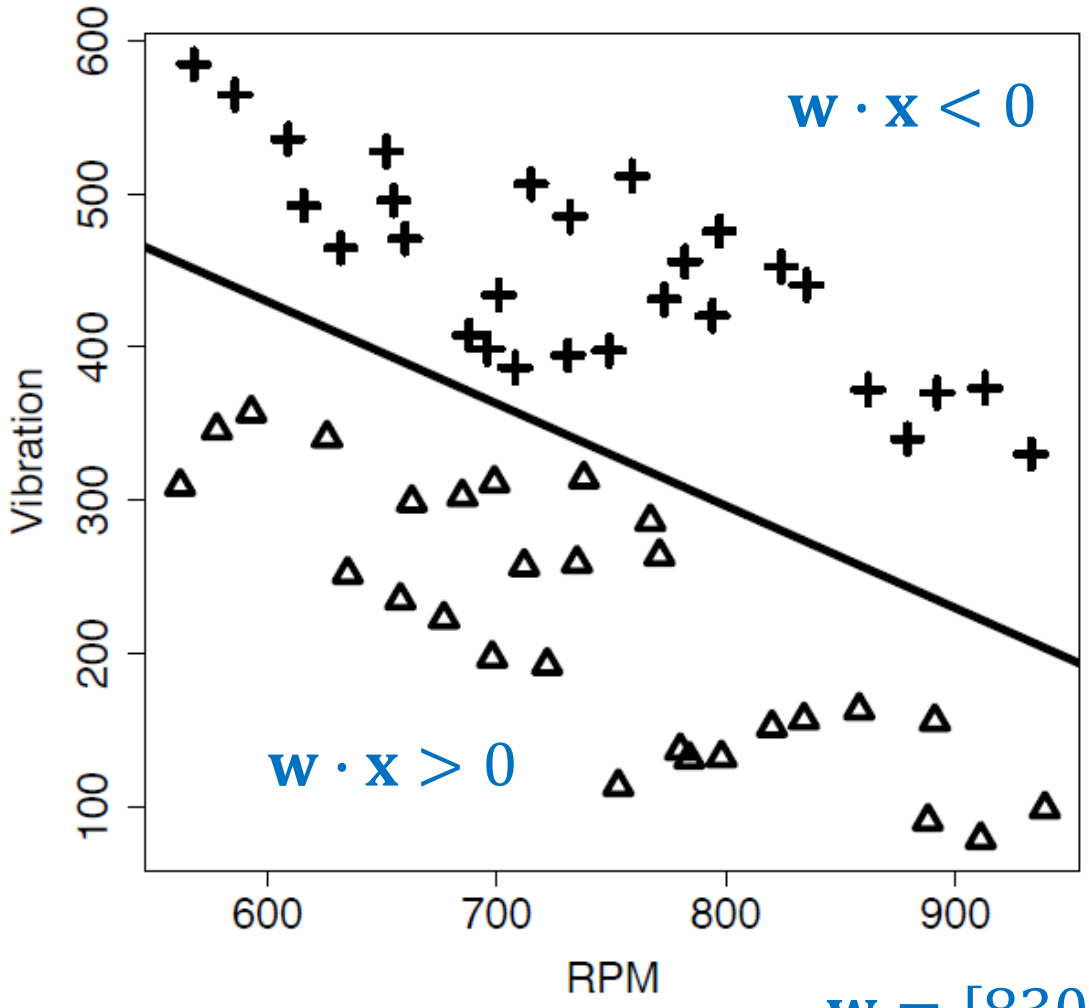
$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = -w_0 x_0$$

$$\sum_{j=0}^{n} w_j x_j = \mathbf{w} \cdot \mathbf{x} = 0$$

  - A hyperplane can separate the space into two half-spaces

$$\mathbf{w} \cdot \mathbf{x} < 0 \quad \text{and} \quad \mathbf{w} \cdot \mathbf{x} > 0$$

Decision surface



- +: Good
- Δ: Faulty

$$\mathbf{w} = [830 \quad -0.667 \quad -1]$$
$$\mathbf{x} = [1 \quad RPM \quad Vibration]$$

Decision surface

- The learning model with decision surface

$$M_{\mathbf{w}}(\mathbf{q}) = \begin{cases} 1 & if\ \mathbf{w} \cdot \mathbf{x} \geq 0 \quad \text{(Negative)} \\ 0 & Otherwise \quad \text{(positive)} \end{cases}$$

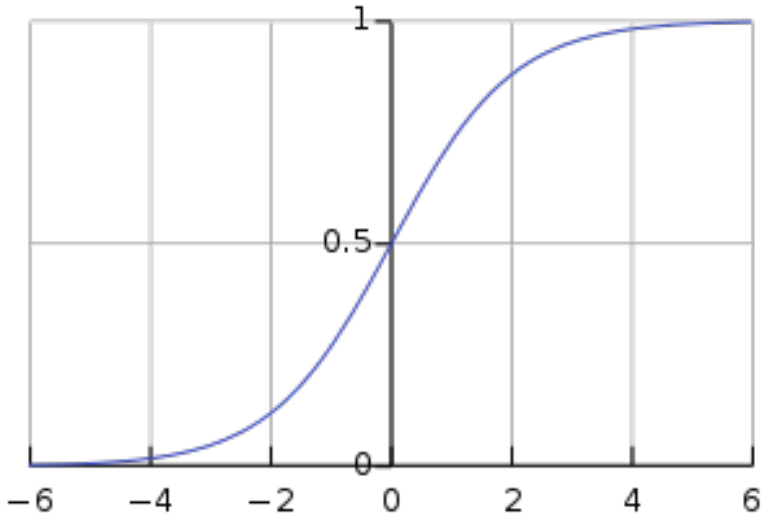
  - How to find the decision surface?
    - $M_{\mathbf{w}}(\mathbf{q})$ is not a continuous function ➔ not differentiable
    - Gradient descent cannot be used!

Logistic Regression

- Logistic function or sigmoid function

$$L(x) = \frac{1}{1 + e^{-x}}$$

- The standard logistic function for *x* over a small range of real numbers such as a range contained in [−6, +6].



Logistic Regression

- Hyperbolic tangent

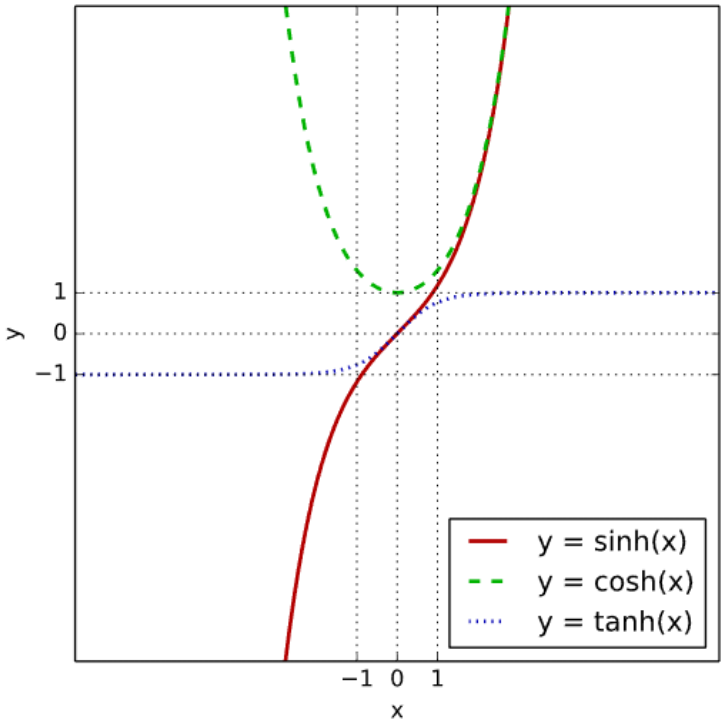
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

$$= \frac{e^x(1 - e^{-2x})}{e^x(1 + e^{-2x})} = \frac{1}{1 + e^{-2x}} - \frac{e^{-x}}{e^x(1 + e^{-2x})}$$

$$= L(2x) - \frac{e^{-2x}}{1 + e^{-2x}} = L(2x) - \frac{e^{-2x} + 1 - 1}{1 + e^{-2x}}$$

$$= 2L(2x) - 1$$

$$L(x) = \frac{1}{2} + \frac{1}{2}\tanh\frac{x}{2}$$



Logistic Regression

- The logistic function has the symmetry property

$$L(-x) = \frac{1}{1+e^{x}} = \frac{e^{-x}}{1+e^{-x}} = \frac{1+e^{-x}-1}{1+e^{-x}}$$

$$= 1 - \frac{1}{1+e^{-x}}$$

$$= 1 - L(x)$$

Logistic Regression

- Rotational symmetry

$$L(x) + L(-x) = \frac{1}{1+e^{-x}} + \frac{1}{1+e^{x}}$$

$$= \frac{(1+e^{x}) + (1+e^{-x})}{(1+e^{-x})(1+e^{x})} = 1$$

Logistic Regression

- Derivative of logistic function

$$\frac{dL(x)}{dx} = \frac{e^x}{(1+e^x)^2} = L(x)(1-L(x))$$

$$\frac{dL(-x)}{dx} = L(-x)(1-L(-x))$$

- Since $L(-x) = 1 - L(x)$

$$\frac{dL(-x)}{dx} = \frac{dL(x)}{dx}$$

Logistic Regression

- Indefinite integral of logistic function

$$\int \mathrm{L}(x)dx = x + \ln(1 + e^{-x})$$

$$= x + \ln(e^{-x}e^{x} + e^{-x})$$

$$= x + \ln(e^{-x}) + \ln(e^{x} + 1)$$

$$= \ln(e^{x} + 1)$$

Logistic Regression

- The learning model with logistic function

$$M_{\mathbf{w}}(\mathbf{x}) = L(\mathbf{w} \cdot \mathbf{x})$$

$$= \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

$\mathbf{x} = [x_0\ x_1\ x_2\ \ldots\ x_n]$ and $x_0 = 1$

$$P(\text{target} = \text{negative}|\mathbf{q}) = M_{\mathbf{w}}(\mathbf{q})$$

it should be closed to 1.0

$$P(\text{target} = \text{positive}|\mathbf{q}) = 1 - M_{\mathbf{w}}(\mathbf{q})$$

it should be closed to 0.0

Logistic Regression

- Error function
  - Let *y* be the labeled target for each training data instance **x**
    - *y* = 1.0 if the target is negative
    - Otherwise *y* = 0.0
  - L2-norm error =

$$\sum_{i=1}^{m} \frac{1}{2}\left(y - M_{\mathbf{w}}(\mathbf{x}_i)\right)^2$$

Logistic Regression

$$\frac{\partial \sum_{i=1}^{m} \frac{1}{2}\left(y_i - M_{\mathbf{w}}(\mathbf{x}_i)\right)^2}{\partial w_j}$$

Chain rule

$$F(x) = f\big(g(x)\big)$$
$$F'(x) = f'\big(g(x)\big)g'(x)$$

- Let $m = 1$

$$\frac{\partial \frac{1}{2}\left(y - M_{\mathbf{w}}(\mathbf{x})\right)^2}{\partial w_j} = \left(y - M_{\mathbf{w}}(\mathbf{x})\right)\frac{\partial\left(y - M_{\mathbf{w}}(\mathbf{x})\right)}{\partial w_j} = (y - L(\mathbf{w}\cdot\mathbf{x}))\frac{\partial\left(y - L(\mathbf{w}\cdot\mathbf{x})\right)}{\partial w_j}$$

$$= (y - M_{\mathbf{w}}(\mathbf{x}))\frac{\partial\left(-L(\mathbf{w}\cdot\mathbf{x})\right)}{\partial w_j}\frac{\partial(\mathbf{w}\cdot\mathbf{x})}{\partial w_j}$$

$$= (y - M_{\mathbf{w}}(\mathbf{x}))\frac{\partial\left(-L(\mathbf{w}\cdot\mathbf{x})\right)}{\partial w_j}x_j$$

Logistic Regression

- Since $\frac{dL(x)}{dx} = L(x)(1 - L(x))$

$$(y - L(\mathbf{w} \cdot \mathbf{x}))\frac{\partial(-L(\mathbf{w} \cdot \mathbf{x}))}{\partial w_j}x_j = -(y - M_{\mathbf{w}}(\mathbf{x}))L(\mathbf{w} \cdot \mathbf{x})(1 - L(\mathbf{w} \cdot \mathbf{x}))x_j$$

$$= -(y - M_{\mathbf{w}}(\mathbf{x}))M_{\mathbf{w}}(\mathbf{x})(1 - M_{\mathbf{w}}(\mathbf{x}))x_j$$

- For $m > 1$

$$\frac{\partial \sum_{i=1}^{m} \frac{1}{2}\left(y_i - M_{\mathbf{w}}(\mathbf{x}_i)\right)^2}{\partial w_j}$$

$$= \sum_{i=1}^{m} -(y_i - M_{\mathbf{w}}(\mathbf{x}_i))M_{\mathbf{w}}(\mathbf{x}_i)(1 - M_{\mathbf{w}}(\mathbf{x}_i))x_{ij}$$

Logistic Regression

- To minimize to error

$$w_j' = -\frac{\partial \sum_{i=1}^{m} \frac{1}{2}\left(y_i - M_{\mathbf{w}}(\mathbf{x}_i)\right)^2}{\partial w_j}$$

- ➔ $$w_j' = \sum_{i=1}^{m} (y_i - M_{\mathbf{w}}(\mathbf{x}_i)) M_{\mathbf{w}}(\mathbf{x}_i)(1 - M_{\mathbf{w}}(\mathbf{x}_i)) x_{ij}$$

- Therefore, increasing w_j by w'_j can let total error to approach zero

$$w_j = w_j + \alpha w_j'$$

- α: learning rate

Logistic Regression

- All feature values should be normalized.
- Normalized to [-1.0, 1.0] or [0.0. 1.0]

Logistic Regression

- Example
  - α = 0.02
  - Initial Weights $\mathbf{w} = [-2.9456 \quad -1.0147 \quad -2.161]$

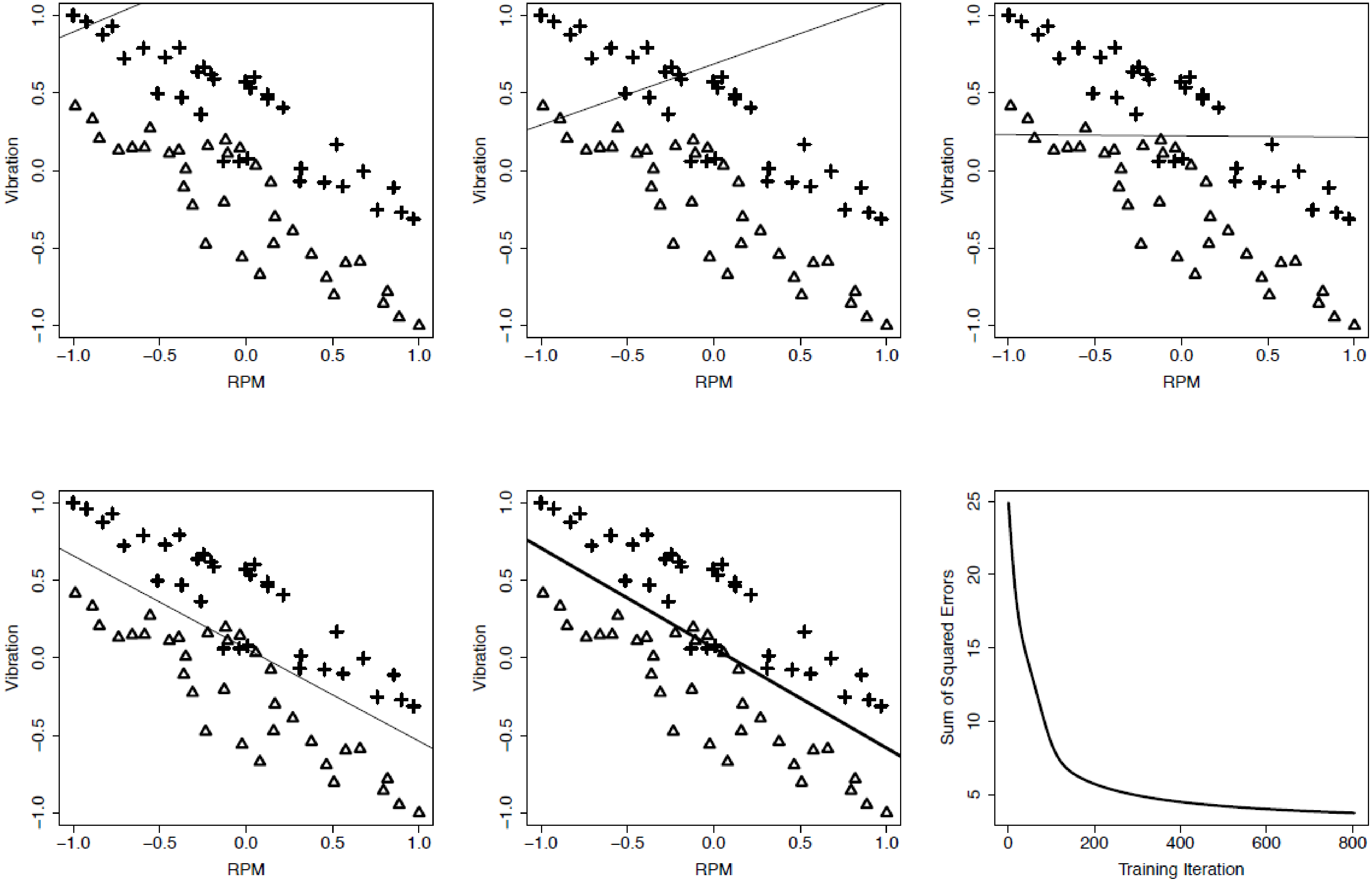
ID	Target Level	Pred.	Error	Squared Error	w_0'	w_1'	w_2'
1	1	0.5570	0.4430	0.1963	0.1093	-0.1093	0.1093
2	1	0.5168	0.4832	0.2335	0.1207	-0.1116	0.1159
3	1	0.4469	0.5531	0.3059	0.1367	-0.1134	0.1197
4	1	0.4629	0.5371	0.2885	0.1335	-0.1033	0.1244
				. . .			
65	0	0.0037	-0.0037	0.0000	0.0000	0.0000	0.0000
66	0	0.0042	-0.0042	0.0000	0.0000	0.0000	0.0000
67	0	0.0028	-0.0028	0.0000	0.0000	0.0000	0.0000
68	0	0.0022	-0.0022	0.0000	0.0000	0.0000	0.0000
			Sum	24.4738	2.7031	-0.7015	1.6493
Sum of squared errors (Sum/2)				12.2369			

Logistic Regression

- Example
  - After the 1st iteration, all *w* are updated once,
  - $\mathbf{w} = [-2.8924 \quad -1.0287 \quad -2.194]$

ID	TARGET LEVEL	Pred.	Error	Squared Error	w_0'	w_1'	w_2'
1	1	0.5817	0.4183	0.1749	0.1018	-0.1018	0.1018
2	1	0.5414	0.4586	0.2103	0.1139	-0.1053	0.1094
3	1	0.4704	0.5296	0.2805	0.1319	-0.1094	0.1155
4	1	0.4867	0.5133	0.2635	0.1282	-0.0992	0.1194
				. . .			
65	0	0.0037	-0.0037	0.0000	0.0000	0.0000	0.0000
66	0	0.0043	-0.0043	0.0000	0.0000	0.0000	0.0000
67	0	0.0028	-0.0028	0.0000	0.0000	0.0000	0.0000
68	0	0.0022	-0.0022	0.0000	0.0000	0.0000	0.0000
			Sum	24.0524	2.7236	-0.6646	1.6484
Sum of squared errors (Sum/2)				12.0262			

Logistic Regression



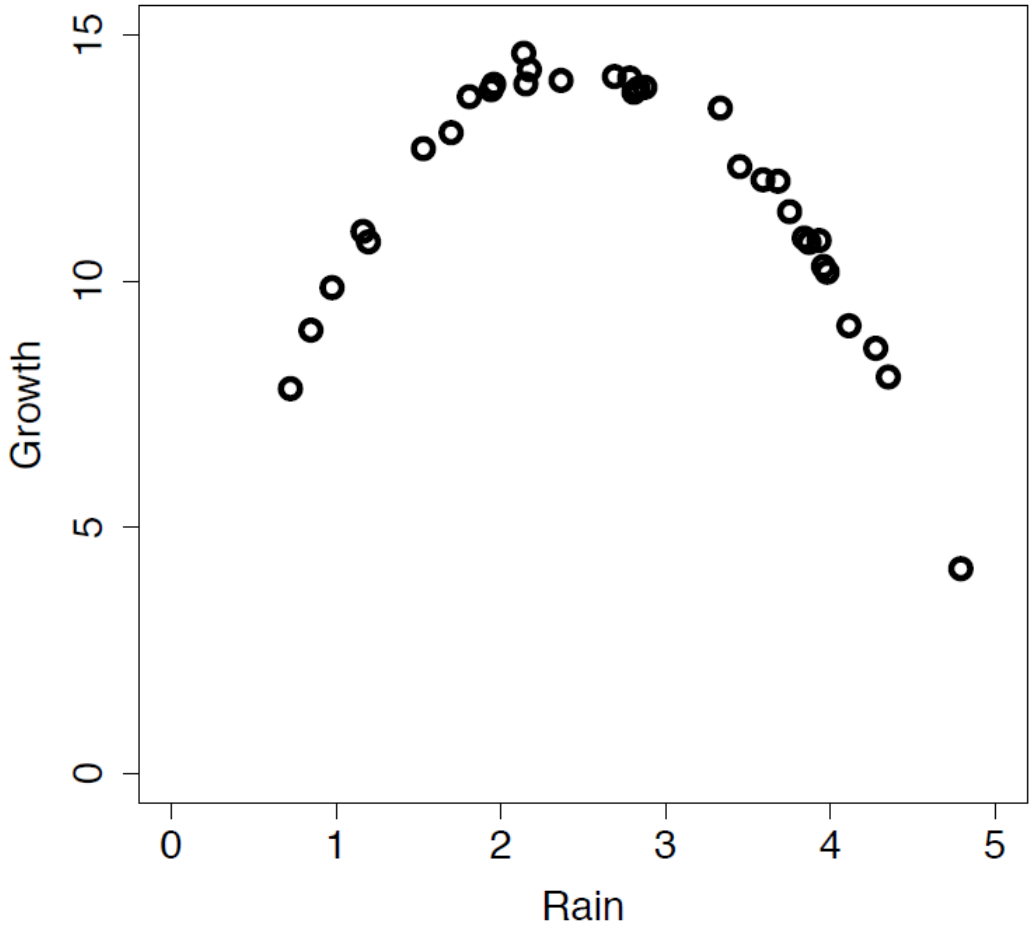
Logistic Regression

- Modeling non-linear relationships
- Example: Grass growth on Irish farms during July 2012.

ID	RAIN	GROWTH	ID	RAIN	GROWTH	ID	RAIN	GROWTH
1	2.153	14.016	12	3.754	11.420	23	3.960	10.307
2	3.933	10.834	13	2.809	13.847	24	3.592	12.069
3	1.699	13.026	14	1.809	13.757	25	3.451	12.335
4	1.164	11.019	15	4.114	9.101	26	1.197	10.806
5	4.793	4.162	16	2.834	13.923	27	0.723	7.822
6	2.690	14.167	17	3.872	10.795	28	1.958	14.010
7	3.982	10.190	18	2.174	14.307	29	2.366	14.088
8	3.333	13.525	19	4.353	8.059	30	1.530	12.701
9	1.942	13.899	20	3.684	12.041	31	0.847	9.012
10	2.876	13.949	21	2.140	14.641	32	3.843	10.885
11	4.277	8.643	22	2.783	14.138	33	0.976	9.876

Logistic Regression

- The data scattering cannot be separated into two spaces
- Even the linear regression cannot be used



Regression with basis functions

- $\mathbf{x} = [x_0\ x_1\ x_2\ \ldots\ x_n]$, where $x_0 = 1$
- $\mathbf{w} = [w_0\ w_1\ \ldots\ w_b]$, $k > 0$

$$y = w_0\phi_0(\mathbf{x}) + w_1\phi_1(\mathbf{x}) + \cdots + w_k\phi_b(\mathbf{x})$$

$$= \sum_{j=0}^{b} w_j\phi_j(\mathbf{x})$$

$$= \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x})$$

- where $\mathbf{\Phi}$ is a series of basis functions to transform $\mathbf{x}$

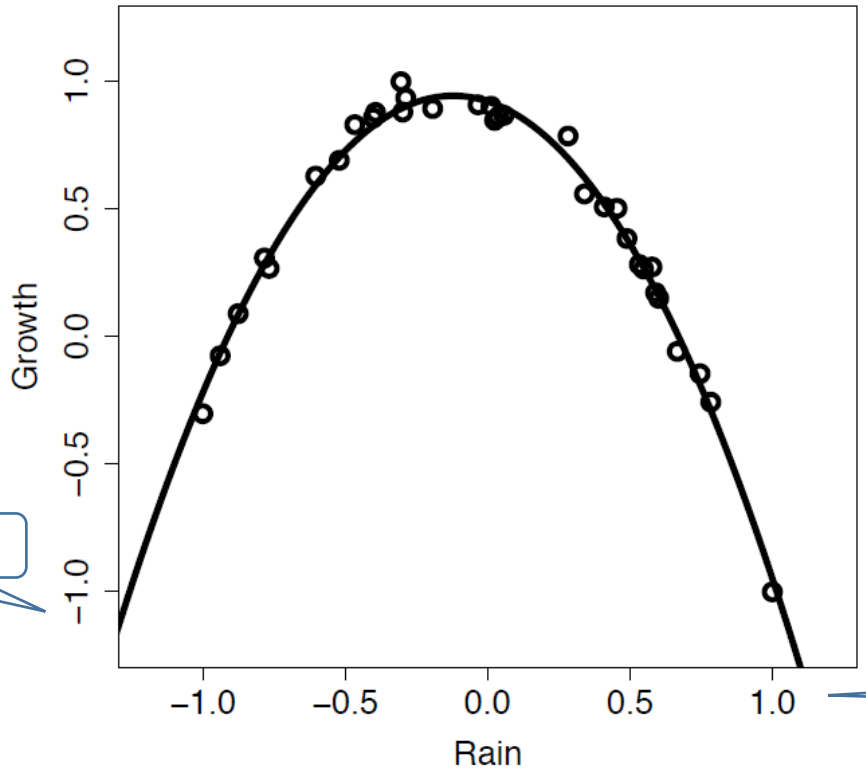
Regression with basis functions

- Example:

$$\phi_0(x) = 1$$
$$\phi_1(x) = x$$
$$\phi_2(x) = x^2$$

Using gradient descent, we have

$$\mathbf{w} = [0.3707 \quad 0.8475 \quad 1.717]$$



Normalized

Normalized

Regression with basis functions

- Logistic regression model using basis functions

$$M_{\mathbf{w}}(\mathbf{x}) = L(\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}))$$

$$= \frac{1}{1 + e^{-\sum_{j=0}^{k} w_j \phi_j(\mathbf{x})}}$$

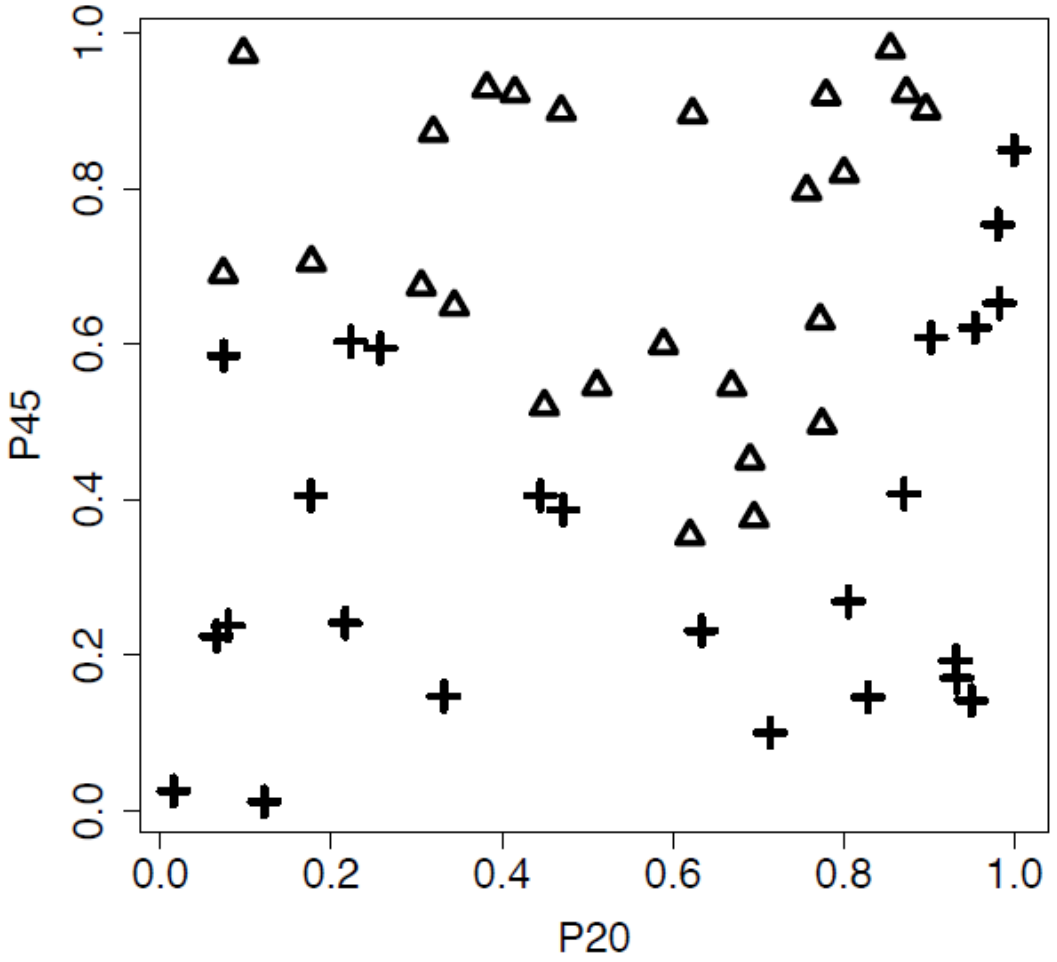
Regression with basis functions

- Example: A dataset showing participants' responses to viewing 'positive' and 'negative' images measured on the EEG P20 and P45 potentials.

ID	P20	P45	TYPE	ID	P20	P45	TYPE
1	0.4497	0.4499	negative	26	0.0656	0.2244	positive
2	0.8964	0.9006	negative	27	0.6336	0.2312	positive
3	0.6952	0.3760	negative	28	0.4453	0.4052	positive
4	0.1769	0.7050	negative	29	0.9998	0.8493	positive
5	0.6904	0.4505	negative	30	0.9027	0.6080	positive
6	0.7794	0.9190	negative	31	0.3319	0.1473	positive
		⋮				⋮	

Regression with basis functions

- Example:



+: Positive

Δ: Negative

Regression with basis functions

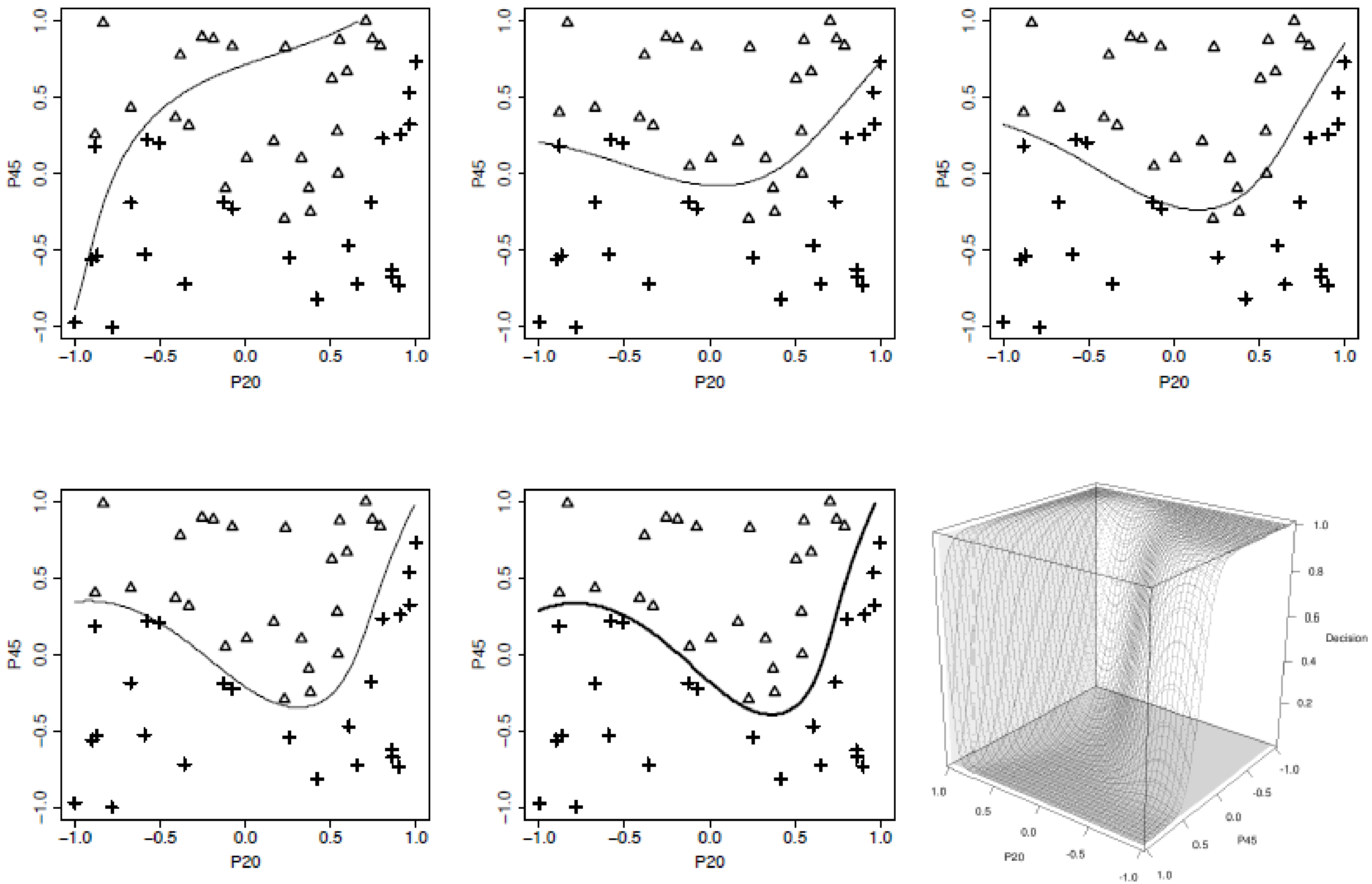
- Basis function:

$$\phi_0(x_0, x_1) = 1$$
$$\phi_1(x_0, x_1) = x_0$$
$$\phi_2(x_0, x_1) = x_1$$
$$\phi_3(x_0, x_1) = {x_0}^2$$
$$\phi_4(x_0, x_1) = {x_1}^2$$
$$\phi_5(x_0, x_1) = {x_0}^3$$
$$\phi_6(x_0, x_1) = {x_1}^3$$
$$\phi_7(x_0, x_1) = x_0 x_1$$

These are not standard 3rd-order polynomial surface

Regression with basis functions

- Gradient descent



Regression with basis functions

- Quadratic basis functions

$$\mathbf{x} = [x_0\ x_1\ x_2\ \dots\ x_n], \text{ where } x_0 = 1$$

$$\mathbf{w} = [w_0\ w_1\ \dots w_b], \text{where } b + 1 = \frac{(n+2)(n+1)}{2}$$

$$y = w_0 x_0 x_0 + w_1 x_0 x_1 + \cdots + w_n x_0 x_n +$$
$$w_{n+1} x_1 x_1 + w_{n+2} x_1 x_2 + \cdots + w_{2n-1} x_1 x_n +$$
$$w_{2n} x_2 x_2 + w_{2n+1} x_2 x_3 + \cdots + w_{3n-2} x_2 x_n +$$
$$\dots$$
$$w_b x_n x_n$$

$$y = \sum_{i=0}^{n} \sum_{j=i}^{n} w_{ij} x_i x_j$$

Regression with basis functions

- Cubic basis functions

$$y = \sum_{i=0}^{n} \sum_{j=i}^{n} \sum_{k=j}^{n} w_{ijk} x_i x_j x_k$$

$$\mathbf{w} = [w_0\ w_1\ \ldots w_b], \text{where } b + 1 = \sum_{k=0}^{n} \frac{(k+2)(k+1)}{2}$$

n = 1, b = 4 - 1
n = 2, b = 10 - 1
n = 3, b = 20 - 1

Multinomial Logistic Regression

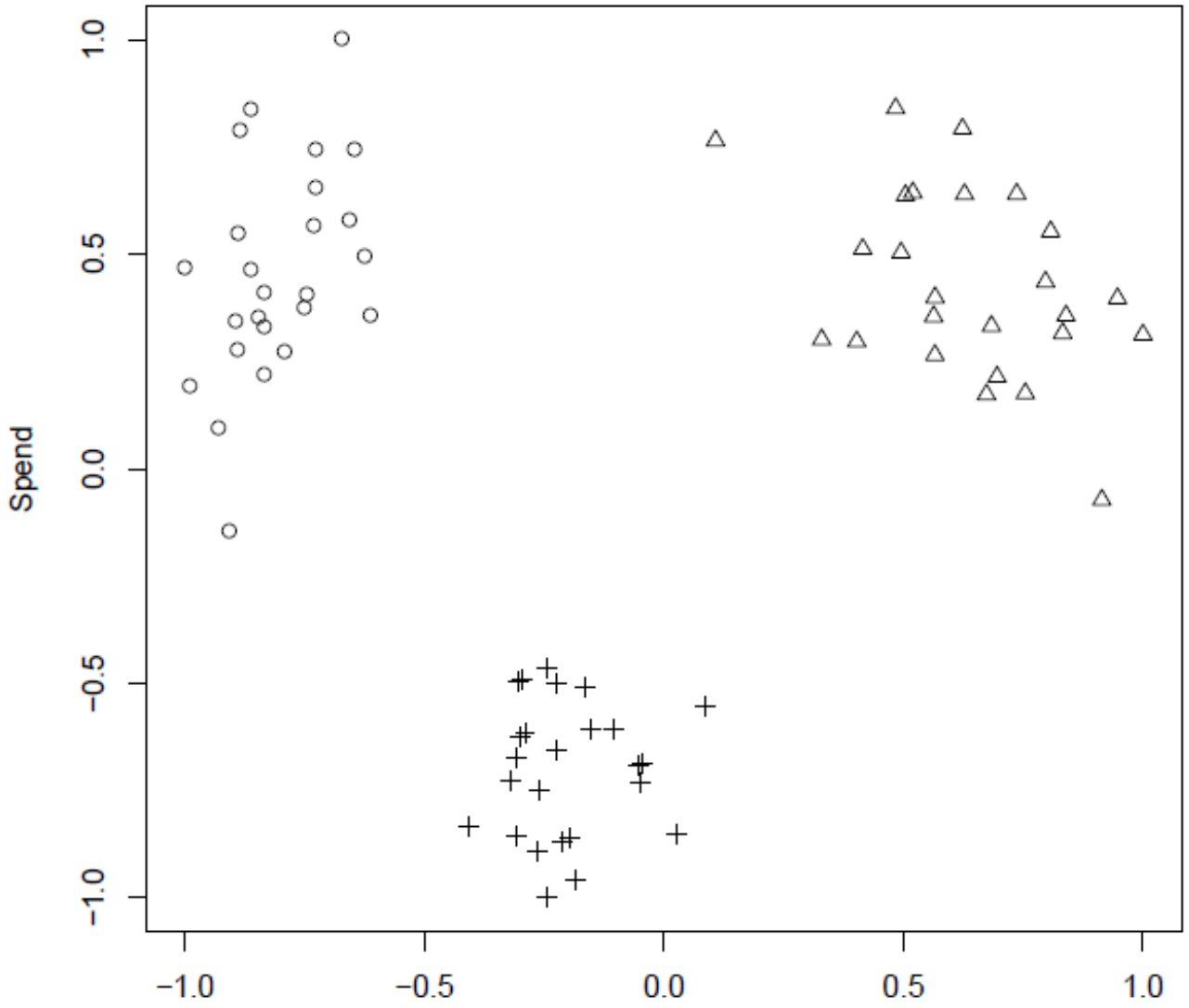
- Example: A dataset of customers of a large national retail chain

ID	SPEND	FREQ	TYPE
1	21.6	5.4	single
2	25.7	7.1	single
3	18.9	5.6	single
4	25.7	6.8	single
		⋮	
26	107.9	5.8	business
27	92.9	5.5	business

ID	SPEND	FREQ	TYPE
28	122.6	6.0	business
29	107.7	5.7	business
		⋮	
47	53.2	2.6	family
48	52.4	2.0	family
49	46.1	1.4	family
50	65.3	2.2	family

Multinomial Logistic Regression

- Example: A dataset of customers of a large national retail chain



O: Single
+: Family
Δ: Business

Multinomial Logistic Regression

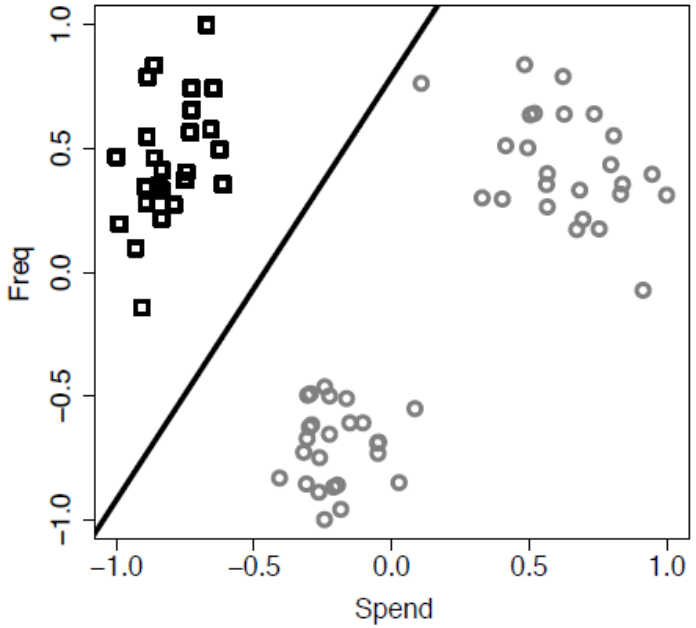
- Given the *r* target levels $T = \{t_1 \quad t_2 \quad \ldots \quad t_r\}$
- *r* logistic regression models:
  - ➔Each of them is a one-versus-all separation

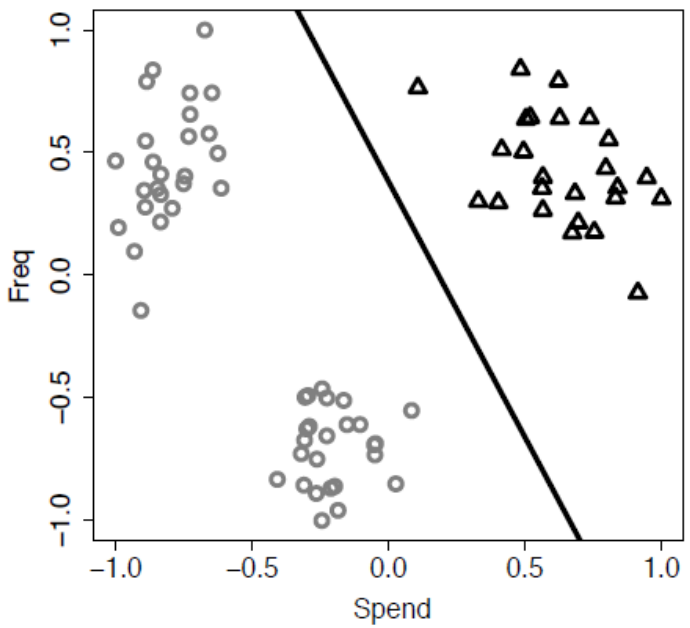
$$M_{\mathbf{w}_1}(\mathbf{x}) = L(\mathbf{w}_1 \cdot \mathbf{x})$$
$$M_{\mathbf{w}_2}(\mathbf{x}) = L(\mathbf{w}_2 \cdot \mathbf{x})$$

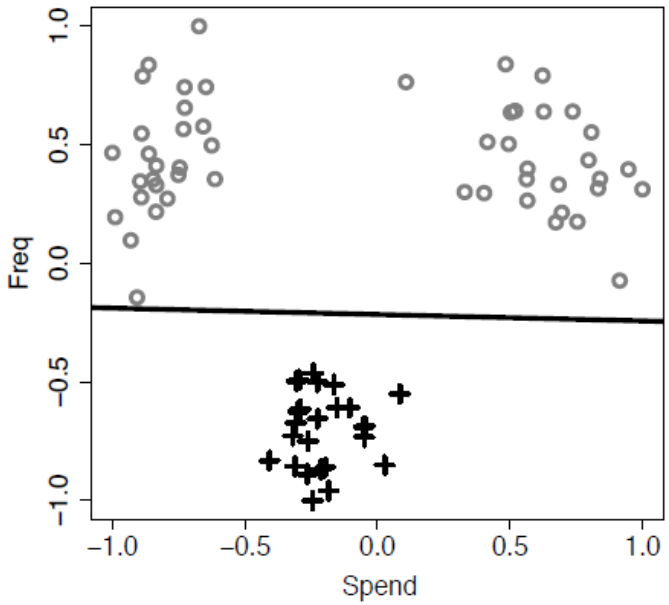
...

$$M_{\mathbf{w}_r}(\mathbf{x}) = L(\mathbf{w}_r \cdot \mathbf{x})$$

Multinomial Logistic Regression







Multinomial Logistic Regression

- To combine the outputs of these different models, we normalize the result of each logistic model:

$$M'_{\mathbf{w}_k}(\mathbf{x}) = \frac{M_{\mathbf{w}_k}(\mathbf{x})}{\sum_{s=1}^{r} M_{\mathbf{w}_s}(\mathbf{x})}$$

- To verify each step of gradient descent, we can compute the L2 error of all training data instances

$$E(M_{\mathbf{w}_k}, \mathbf{X}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - M'_{\mathbf{w}_k}(\mathbf{x}_i)\right)^2$$

  - where y_i is 0 if the target of $\mathbf{x}_i$ is t_k; otherwise y_i is 1.

Multinomial Logistic Regression

- The multinomial logistic model then is

$$M(\mathbf{x}) = \arg\max_{t_k \in T} M'_{\mathbf{w}_k}(\mathbf{x})$$

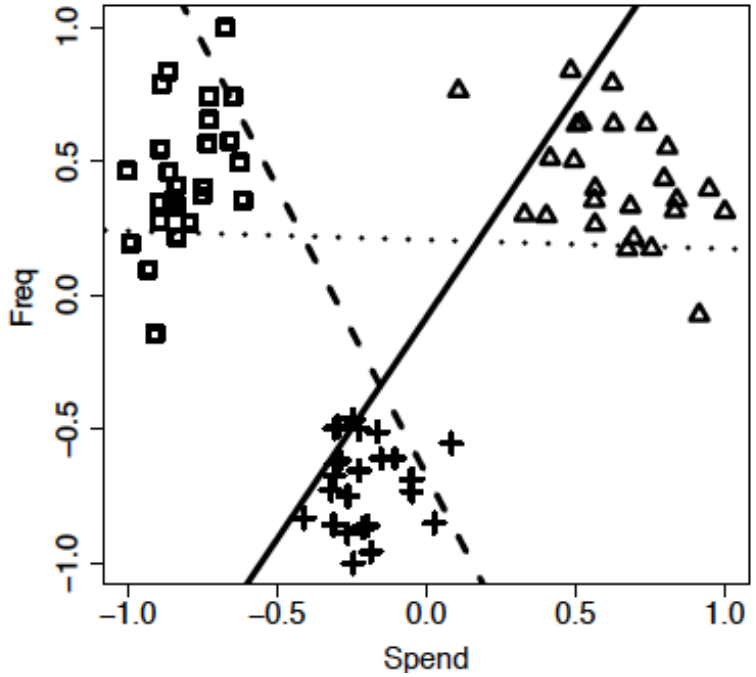
Multinomial Logistic Regression

- Example

$$Single(O):\ M_{\mathbf{w}_1}(\mathbf{x}) = L([0.7993 \quad -15.9 \quad 9.5974] \cdot \mathbf{x})$$

$$Family(+):\ M_{\mathbf{w}_2}(\mathbf{x}) = L([3.6526 \quad -0.58 \quad 17.5886] \cdot \mathbf{x})$$

$$Business(\Delta):\ M_{\mathbf{w}_3}(\mathbf{x}) = L([4.6419 \quad 14.94 \quad 6.9457] \cdot \mathbf{x})$$



Multinomial Logistic Regression

- Example
  - **q** = [1 25.67 6.12] ➔ normalize ➔ **q** = [1 -0.7279 0.4789]

$$Single(O)\colon M_{\mathbf{w}_1}(\mathbf{q}) = 0.9999$$

$$Family(+)\colon M_{\mathbf{w}_2}(\mathbf{q}) = 0.01278$$

$$Business(\Delta)\colon M_{\mathbf{w}_3}(\mathbf{q}) = 0.0518$$

$$Single(O)\colon M'_{\mathbf{w}_1}(\mathbf{q}) = 0.9393$$

$$Family(+)\colon M_{\mathbf{w}_2}(\mathbf{q}) = 0.0120$$

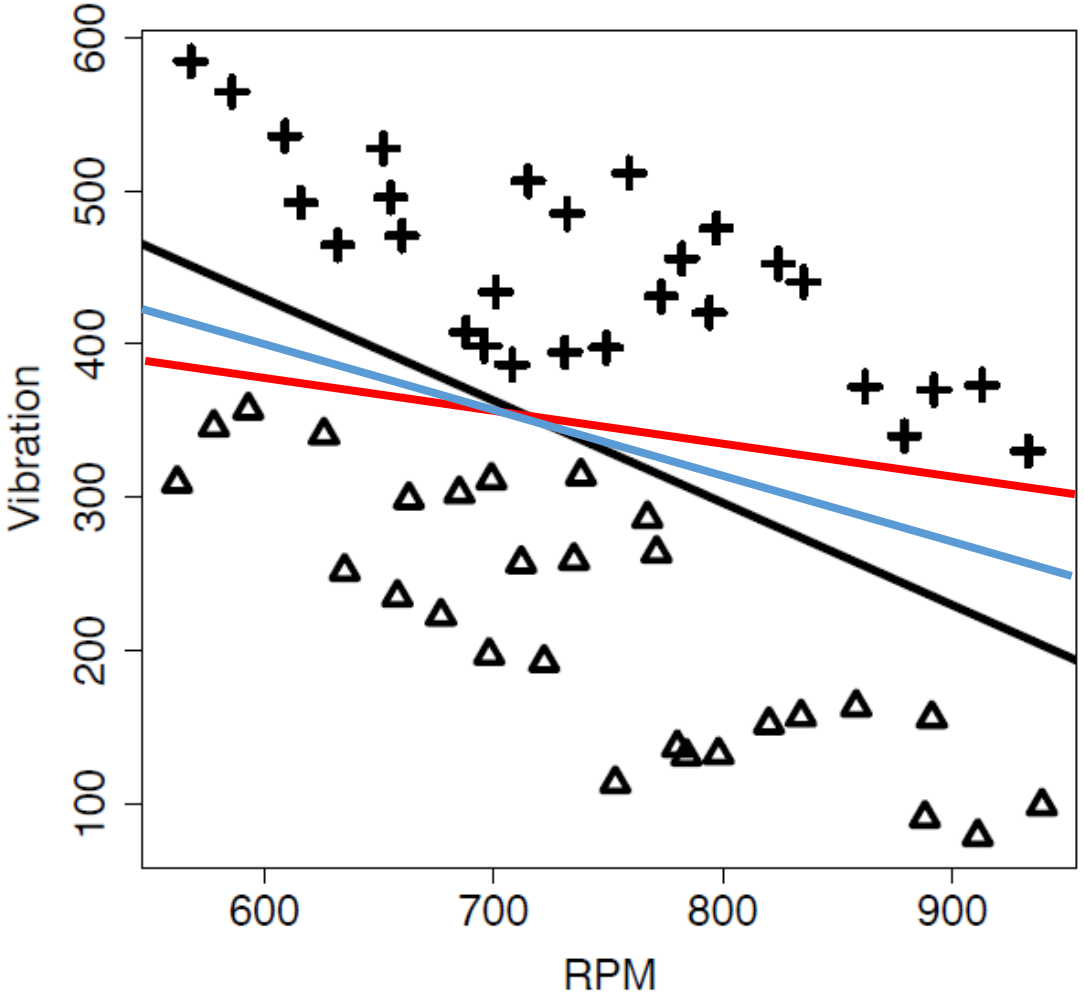
$$Business(\Delta)\colon M_{\mathbf{w}_3}(\mathbf{q}) = 0.0487$$

Support Vector Machines

- SVM
- Binary linear classification by a hyperplane
- The hyperplane is decided by some data instances from the training dataset
- These selected data instances are called **support vectors**
- Such that the distance (margin) between the support vectors and hyperplane is maximum.

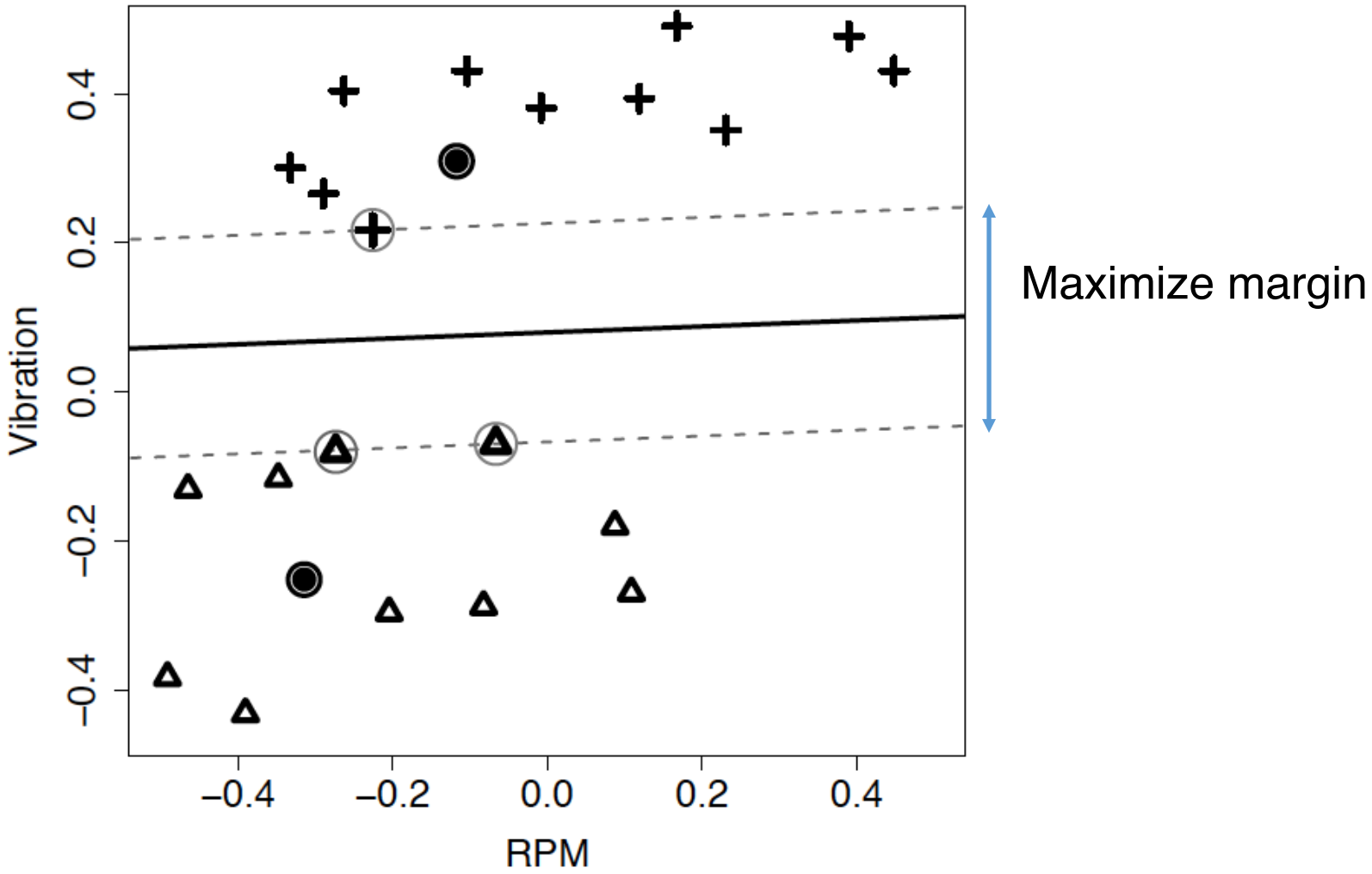
Support Vector Machines

- Example: Generators status
  - Linear regression: infinite solutions!



Support Vector Machines

- Example: Generators status
  - SVM: Optimal solution



Support Vector Machines

- Training data set:
  - $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$
  - and each data instance $\mathbf{x}_i$ has a target y_i
- Each data instance consists of *n* features:

$$\mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]^T$$

- A set of n weights: $\mathbf{w} = [w_1 \quad w_2 \quad \cdots \quad w_n]^T$
- The hyperplane:
  - $\mathbf{w} \cdot \mathbf{x} + b = 0$
- Decision surface
  - $\mathbf{w} \cdot \mathbf{x} + b > 0$, for $y > 0$
  - $\mathbf{w} \cdot \mathbf{x} + b < 0$, for $y < 0$

Support Vector Machines

- Decision surface with margin
  - $\mathbf{w} \cdot \mathbf{x} + b \geq 1$, for y = 1
  - $\mathbf{w} \cdot \mathbf{x} + b \leq -1$, for y = -1
- Why 1?
  - Given any number c
    - $\mathbf{w} \cdot \mathbf{x} + b \geq c$, for y = 1
    - $\mathbf{w} \cdot \mathbf{x} + b \leq -c$, for y = -1
  - You always can find a number b' such that
    - $\mathbf{w} \cdot \mathbf{x} + b' \geq 1$, for y = 1
    - $\mathbf{w} \cdot \mathbf{x} + b' \leq -1$, for y = -1

Support Vector Machines

- Perpendicular distance between a point to a plane

$$d = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|}$$

  - where $\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$

- Perpendicular distance between two **parallel** planes

$$\boldsymbol{H_1} \equiv \mathbf{w} \cdot \mathbf{x} + b_1 = 0$$

$$\boldsymbol{H_2} \equiv \mathbf{w} \cdot \mathbf{x} + b_2 = 0$$

$$d = \frac{|b_1 - b_2|}{\|\mathbf{w}\|}$$

Support Vector Machines

- Let

$$\boldsymbol{H_0} \equiv \mathbf{w} \cdot \mathbf{x} + b = 0$$

$$\boldsymbol{H_1} \equiv \mathbf{w} \cdot \mathbf{x} + b = 1$$

$$\boldsymbol{H_2} \equiv \mathbf{w} \cdot \mathbf{x} + b = -1$$

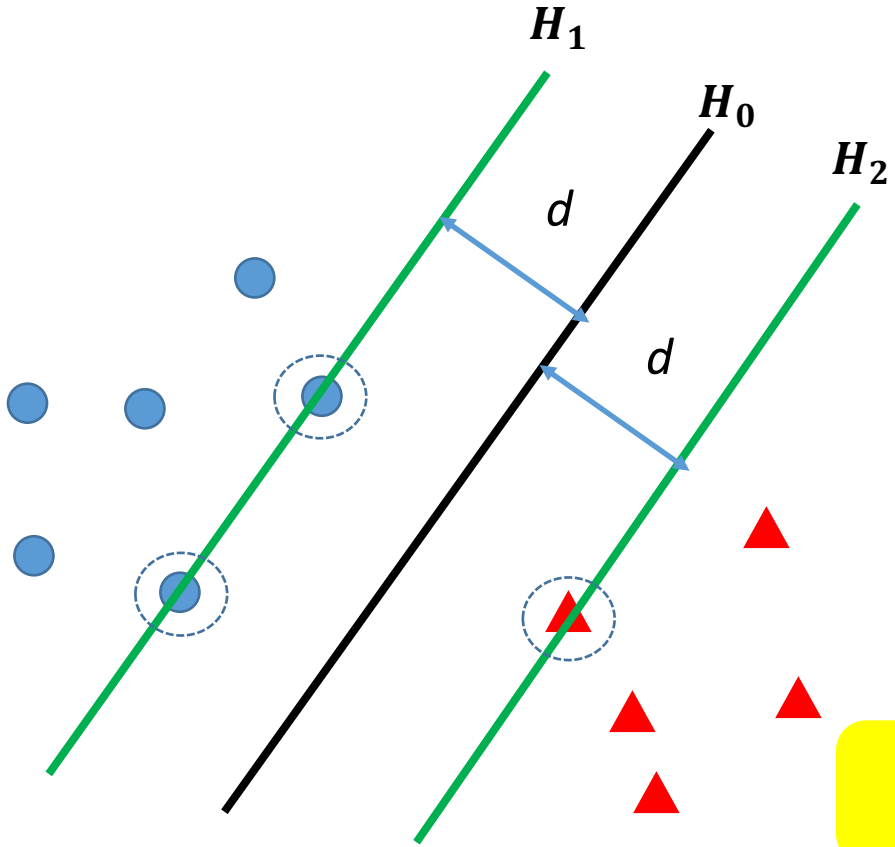
- The distance *d* between $\boldsymbol{H_0}$ and $\boldsymbol{H_1}$ (or $\boldsymbol{H_0}$ and $\boldsymbol{H_2}$)

$$d = \frac{1}{\|\mathbf{w}\|}$$

- Therefore, maximize d ←→minimize $\|\mathbf{w}\|$

Support Vector Machines

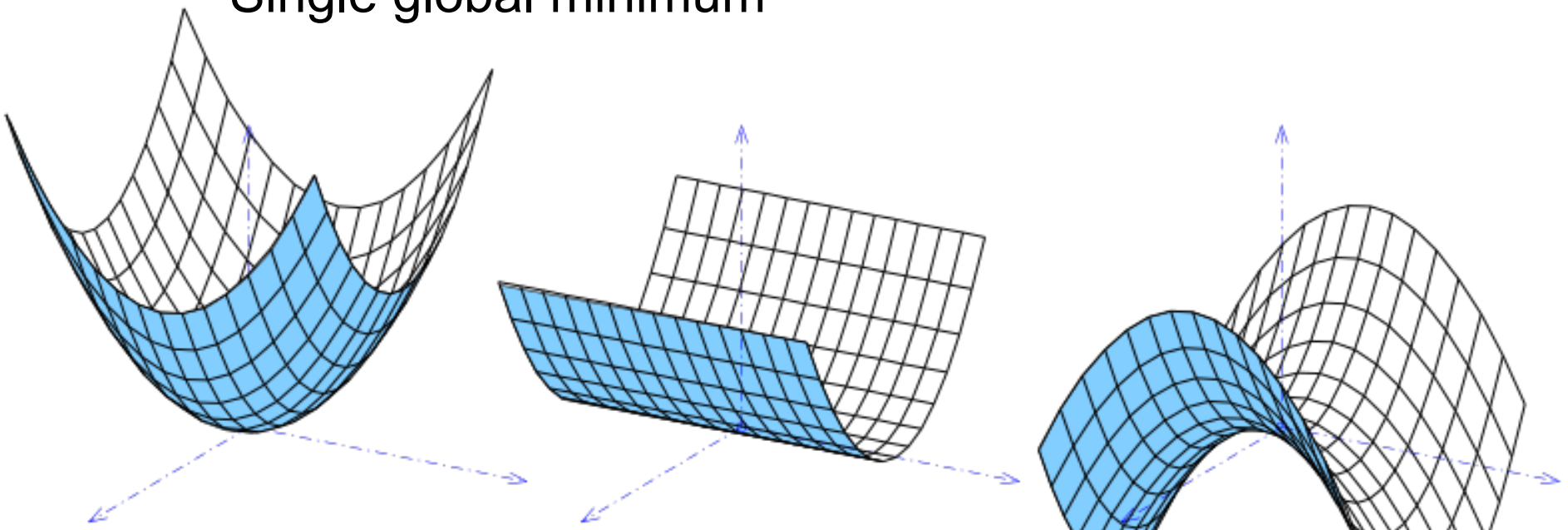
- Finding a set of support vectors:
  - $X' \subseteq X = \{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_k\}$
  - $\boldsymbol{H_1} \equiv \mathbf{w} \cdot \mathbf{x}'_i + b = 1$, where $\mathbf{x}'_i \in X'$ and $\boldsymbol{y}'_i = 1$
  - $\boldsymbol{H_2} \equiv \mathbf{w} \cdot \mathbf{x}'_j + b = -1$, where $\mathbf{x}'_j \in X'$ and $\boldsymbol{y}'_j = -1$
- Such that *d* is maximum



Support Vector Machines

- Minimizing $\|\mathbf{w}\|$➔Minimizing a quadratic function $f(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$

- The surface of a quadratic function is a **paraboloid**.
  - Single global minimum



$z = x^2 + y^2$ $z = x^2$ $z = x^2 - y^2$

Support Vector Machines

- Recall that

$$\mathbf{w} \cdot \mathbf{x} + b \geq 1, \text{ for } y = 1$$

$$\mathbf{w} \cdot \mathbf{x} + b \leq -1, \text{ for } y = -1$$

- It can be rewritten as

$$\mathbf{1}(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0, \text{ for } y = 1$$

$$-\mathbf{1}(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0, \text{ for } y = -1$$

- ➔

$$y(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0$$

Support Vector Machines

- Minimizing $$f(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$$
- Subject to $$g(\mathbf{w}) = y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \forall \mathbf{x}_i$$

Support Vector Machines

- Lagrange Multiplier Method (LMM)
  - Minimize or maximize a function of N variables
    $$f(x_1, x_2, \ldots, x_N)$$
  - Subject to *M* constraints
    $$g_k(x_1, x_2, \ldots, x_N), k = 1,2, \ldots, M$$
  - **Lagrangian expression** defined by

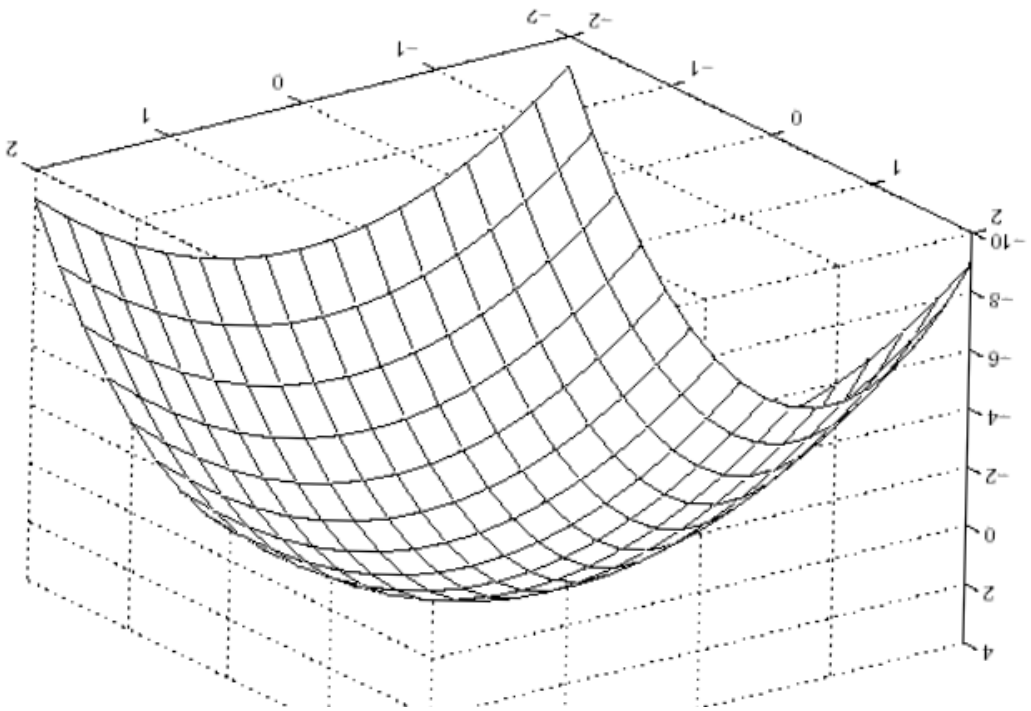
$$\mathcal{L}(x_1, x_2, \ldots, x_N) = f(x_1, x_2, \ldots, x_N) - \sum_{k=1}^{M} \alpha_k g_k(x_1, x_2, \ldots, x_N)$$

  - α_k is called the **Lagrange multiplier**
  - You can give a boundary for each α_k
  - In SVM, $C \geq \alpha_k \geq 0$, for a constant *C*

Support Vector Machines

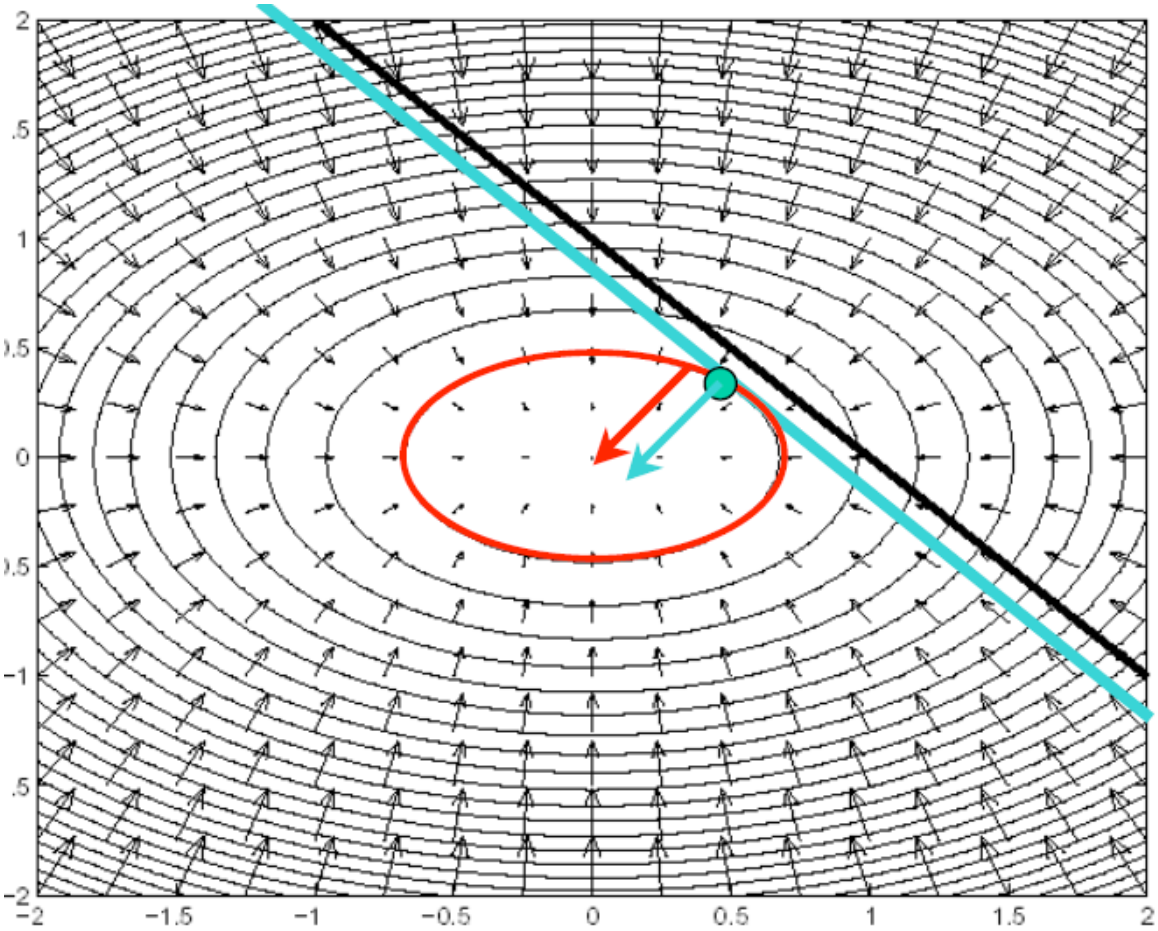
- Example of LMM
  - From: "An Idiot's guide to Support vector machines (SVMs)" -- R. Berwick, Village Idiot.

  - $f(x, y) = 2 + x^2 + 2y^2$
  - $g(x, y) = x + y = 1$



Support Vector Machines

- Example of LMM
  - find intersection of *f* and *g* at a tangent point (intersection = both constraints satisfied; tangent = derivative is 0); this will be a min (or max) for *f* s.t. the contraint *g* is satisfied.



Support Vector Machines

- Lagrange Multiplier Method (LMM)
  - the solution is $\mathbf{x}'$ when

$$\frac{df(\mathbf{x}')}{d\mathbf{x}} = \alpha \frac{dg(\mathbf{x}')}{d\mathbf{x}} \qquad \mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]$$

  - That means ➔

$$\frac{d\mathcal{L}(\mathbf{x})}{d\mathbf{x}} = 0$$

Support Vector Machines

- LMM + SVM

$$f(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$$

$$g(\mathbf{w}, b) = y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \qquad i = 1,2, \ldots, m$$

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1)$$

$$= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i y_i(\mathbf{w} \cdot \mathbf{x}_i + b) + \sum_{i=1}^{m} \alpha_i$$

where $C \geq \alpha_i \geq 0$, for a constant $C > 0$

Support Vector Machines

- Our goals
  - Minimizing $\mathcal{L}(\mathbf{w}, b)$
  - Maximizing $\boldsymbol{\alpha} = [\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_m]^T$

➔ Maximizing $\sum_{i=1}^{m} \alpha_i$

Support Vector Machines

- Minimizing $\mathcal{L}$ ➔ $d\mathcal{L} = 0$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{m} \alpha_i y_i = 0$$

- We get:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \,, \qquad \sum_{i=1}^{m} \alpha_i y_i = 0$$

Support Vector Machines

- Maximizing $\boldsymbol{\alpha}$

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1)$$

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \,, \qquad \sum_{i=1}^{m} \alpha_i y_i = 0$$

- ➔ Maximizing

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Support Vector Machines

- How to find $\boldsymbol{\alpha}$?
  - Let $z_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$

$$\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j z_{ij} = \begin{bmatrix}\alpha_1 & \alpha_2 & \cdots & \alpha_m\end{bmatrix} \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1m} \\ z_{21} & z_{22} & \cdots & z_{2m} \\ \vdots & & \ddots & \\ z_{m1} & z_{m2} & \cdots & z_{mm} \end{bmatrix} \begin{bmatrix}\alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m\end{bmatrix}$$

$$= \boldsymbol{\alpha}^{\mathrm{T}} Q \boldsymbol{\alpha}$$

Q is a symmetric matrix

- Let $\mathbf{u} = [1\ 1\ \ldots\ 1]^{\mathrm{T}}$, we have

$$\mathcal{L}(\boldsymbol{\alpha}) = \mathbf{u}^{\mathrm{T}}\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}} Q \boldsymbol{\alpha}$$

Support Vector Machines

- Quadratic programming
  - Find $\mathbf{x}$ to minimize or maximize

$$\mathbf{u}^{\mathrm{T}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathrm{T}}Q\mathbf{x}$$

    - for a certain vector $\mathbf{u}$ and a symmetric matrix Q

  - Subject to

$$A\mathbf{x} \leq \mathbf{v}$$

    - for a boundary vector $\mathbf{v}$ and a certain matrix A

Support Vector Machines

- Quadratic programming
  - Find $\boldsymbol{\alpha}$ to minimize or maximize

$$\mathcal{L}(\boldsymbol{\alpha}) = \mathbf{u}^{\mathrm{T}}\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}} Q \boldsymbol{\alpha}$$

    - for a certain vector $\mathbf{u}$ and a symmetric matrix Q

  - Subject to

$$C \geq \alpha_i \geq 0$$

Support Vector Machines

- Algorithms of Quadratic programming
  - Interior point
  - Active set
  - Augmented Lagrangian
  - Conjugate gradient
  - Gradient projection
  - Extensions of the simplex algorithm
- All of them are iterative methods.

Support Vector Machines

- We then can get **w** and *b* after we found **α**.

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

- Since $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \forall \mathbf{x}_i$

$y_k(\mathbf{w} \cdot \mathbf{x}_k + b) - 1 = 0$, only for support vectors ($\alpha_k > 0$)

$y_k y_k(\mathbf{w} \cdot \mathbf{x}_k + b) = y_k$

$(\mathbf{w} \cdot \mathbf{x}_k + b) = y_k$ because $y_k \in \{1, -1\}$

$b = y_k - \mathbf{w} \cdot \mathbf{x}_k$ for any support vector ($\alpha_k > 0$)

Support Vector Machines

- What if the training data is not linearly separable?
- Applying a series of basis function →High computational cost
- Applying a **kernel function** φ

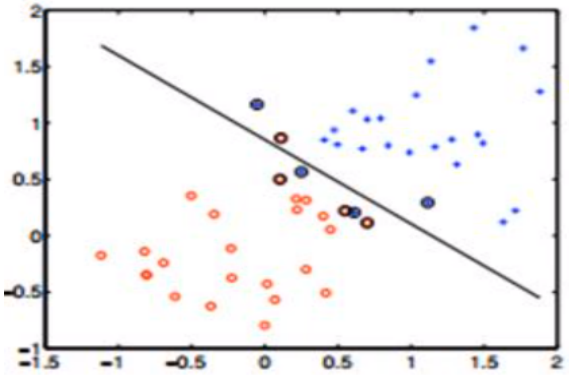
$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \varphi(\mathbf{x}_i \cdot \mathbf{x}_j)$$

Support Vector Machines

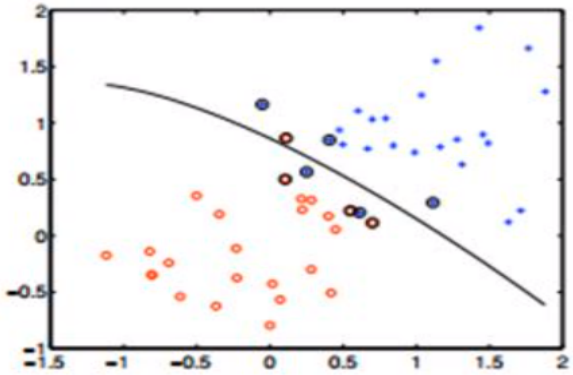
- Polynomial kernel

$$\varphi(\mathbf{x}_i \cdot \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^p$$

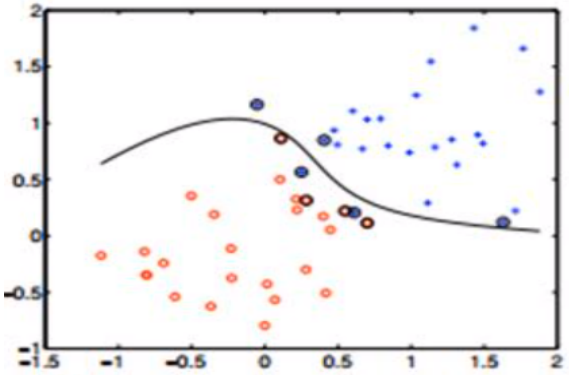
  - where *p* is the degree of polynomial and *c* is a constant.



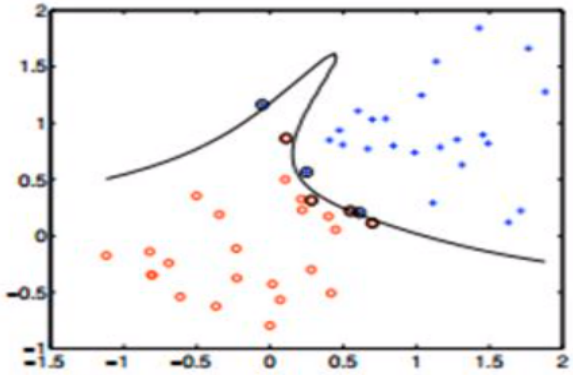
linear



2^{nd} order polynomial



4^{th} order polynomial



8^{th} order polynomial

Fig. created by *Tommi S. Jaakkola. MIT*.

Support Vector Machines

- Gaussian kernel
- Radial basis function kernel $\varphi(\mathbf{x}_i \cdot \mathbf{x}_j) = e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$

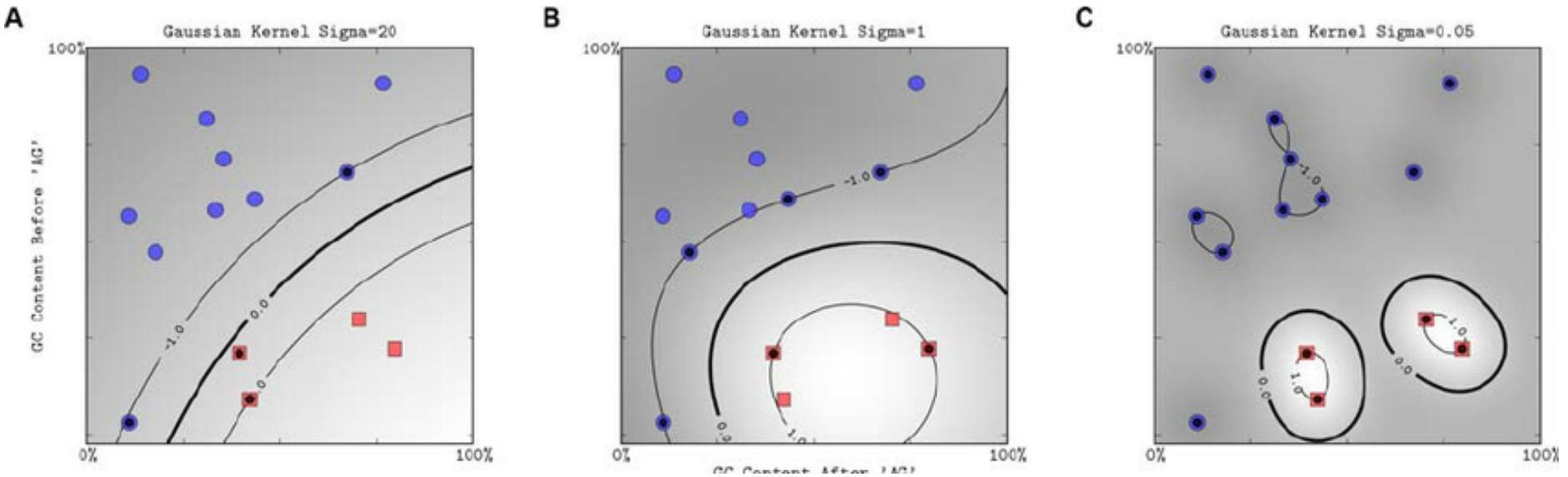
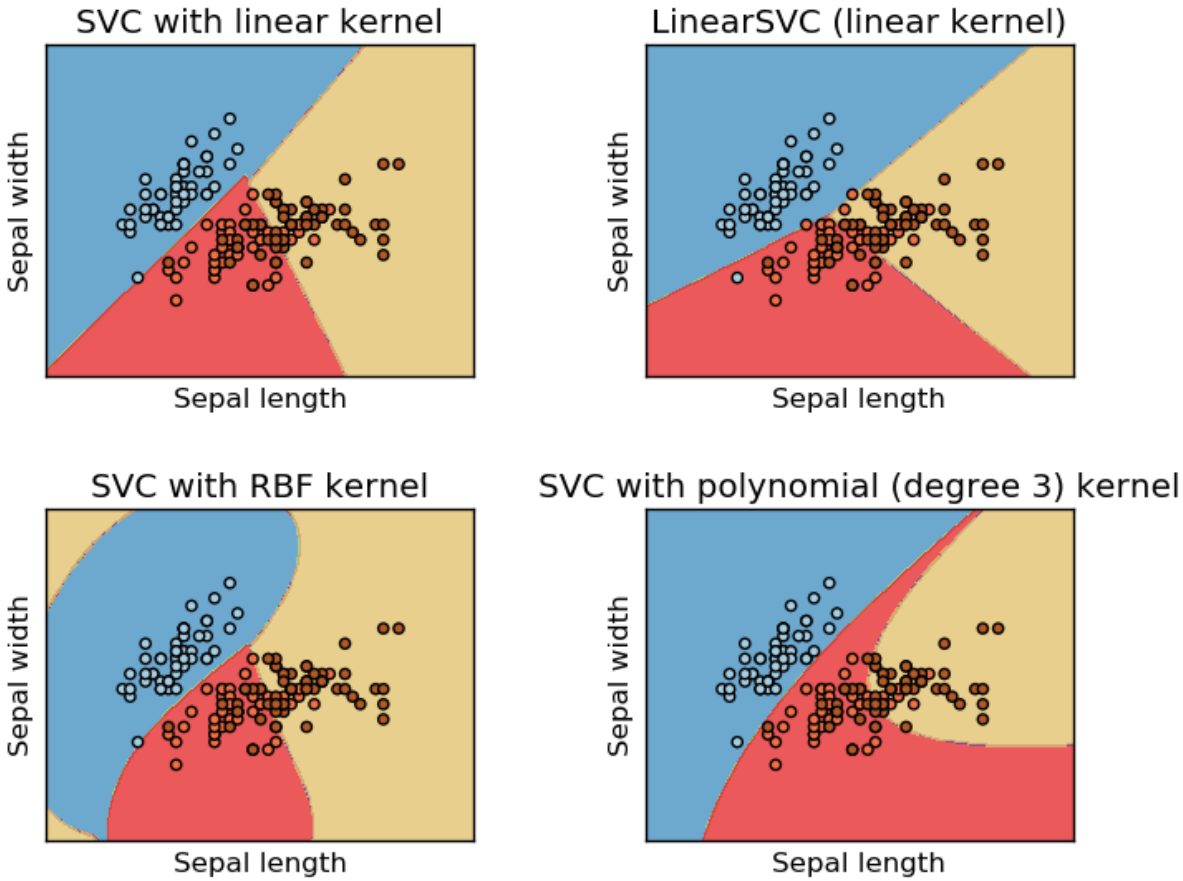


Fig. from:
Ben-Hur, Asa & Ong, Cheng Soon & Sonnenburg, Sören & Schölkopf, Bernhard & Rätsch, Gunnar. (2008). **"Support Vector Machines and Kernels for Computational Biology."** *PLoS computational biology.* 4. e1000173. 10.1371/journal.pcbi.1000173.

Support Vector Machines

- Multinomial SVM
  - Similar to multinomial logistic regression
  - One-versus-all separation



Advanced reading

- Jan A. Snyman, "**Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms**," *Springer*, 2005.
- Nello Cristianini, "**An Introduction to Support Vector Machines and Other Kernel-based Learning Methods**," *Cambridge University Press*, 2000.