

22 Function learning algorithms

Important general learning problem:

Learning a function from examples

- Given a finite set of training pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_t, y_t)$
- Attempt to learn a rule for predicting y given \mathbf{x}

$$\begin{array}{c} \mathbf{x}_1, y_1 \\ \mathbf{x}_2, y_2 \\ \vdots \\ \mathbf{x}_t, y_t \end{array} \Rightarrow \boxed{\text{Learner}} \Rightarrow h : X \rightarrow Y$$

- Want to minimize prediction error, $\text{err}(h(x), y)$, on test examples (x, y)
- The function $\text{err}(\hat{y}, y)$ depends on the problem

For *classification* problems, use

$$\text{err}(\hat{y}, y) = \begin{cases} 0 & \hat{y} = y \\ 1 & \hat{y} \neq y \end{cases}$$

For *real-valued* prediction (regression) problems, use

$$\text{err}(\hat{y}, y) = (\hat{y} - y)^2$$

Generic learning algorithm

1. Fix a hypothesis space $H = \{\text{set of } h\text{'s}\}$
2. Given $(x_1, y_1) \dots (x_t, y_t)$, calculate $h^* \in H$ that minimizes

$$\frac{1}{t} \sum_{i=1}^t \text{err}(h(x_i), y_i)$$

(average training error)

22.1 Special case: real valued prediction

Learn $h : \mathbb{R}^n \rightarrow \mathbb{R}$ to minimize the prediction error

$$\text{err}(\hat{y}, y) = (\hat{y} - y)^2$$

Assume

$$\begin{aligned} H &= \{\text{linear functions}\} \\ &= \{h_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^n\} \\ \text{where } h_{\mathbf{w}}(\mathbf{x}) &= \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^n w_i x_i \end{aligned}$$

Given training data

$$\left[\begin{array}{cccc|c} x_{11} & x_{12} & \dots & x_{1n} & y_1 \\ x_{21} & x_{22} & \dots & x_{2n} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{t1} & x_{t2} & \dots & x_{tn} & y_t \end{array} \right]$$

Calculate \mathbf{w} that minimizes

$$SSE = \sum_{i=1}^t (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2$$

$$\text{where } \mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in})$$

Can solve for this \mathbf{w} by taking derivatives of SSE with respect to each w_j and setting them to zero

$$\begin{aligned} \frac{\partial}{\partial w_j} SSE &= \sum_{i=1}^t 2(\mathbf{w} \cdot \mathbf{x}_i - y_i)x_{ij} \\ &= 2 \left(\sum_{i=1}^t \sum_{k=1}^n w_k x_{ik} x_{ij} - \sum_{i=1}^t y_i x_{ij} \right) \end{aligned}$$

Thus, consider whole vector of derivatives

$$\begin{aligned}
\nabla_{\mathbf{w}} SSE &= \begin{bmatrix} \frac{\partial SSE}{\partial w_1} \\ \frac{\partial SSE}{\partial w_2} \\ \vdots \\ \frac{\partial SSE}{\partial w_n} \end{bmatrix} = \begin{bmatrix} 2 \left(\sum_{i=1}^t \sum_{k=1}^n w_k x_{ik} x_{i1} - \sum_{i=1}^t y_i x_{i1} \right) \\ 2 \left(\sum_{i=1}^t \sum_{k=1}^n w_k x_{ik} x_{i2} - \sum_{i=1}^t y_i x_{i2} \right) \\ \vdots \\ 2 \left(\sum_{i=1}^t \sum_{k=1}^n w_k x_{ik} x_{in} - \sum_{i=1}^t y_i x_{in} \right) \end{bmatrix} \\
&= 2 \begin{bmatrix} \sum_{i=1}^t \sum_{k=1}^n w_k x_{ik} x_{i1} \\ \sum_{i=1}^t \sum_{k=1}^n w_k x_{ik} x_{i2} \\ \vdots \\ \sum_{i=1}^t \sum_{k=1}^n w_k x_{ik} x_{in} \end{bmatrix} - 2 \begin{bmatrix} \sum_{i=1}^t y_i x_{i1} \\ \sum_{i=1}^t y_i x_{i2} \\ \vdots \\ \sum_{i=1}^t y_i x_{in} \end{bmatrix} \\
&= 2X^T \begin{bmatrix} \sum_{k=1}^n w_k x_{1k} \\ \sum_{k=1}^n w_k x_{2k} \\ \vdots \\ \sum_{k=1}^n w_k x_{tk} \end{bmatrix} - 2X^T \mathbf{y} \\
&= 2X^T X \mathbf{w} - 2X^T \mathbf{y}
\end{aligned}$$

Now solve for \mathbf{w} that makes this vector of derivatives equal to zero

$$\begin{aligned}
2X^T X \mathbf{w} - 2X^T \mathbf{y} &= 0 \\
X^T X \mathbf{w} &= X^T \mathbf{y} \\
\mathbf{w} &= (X^T X)^{-1} X^T \mathbf{y}
\end{aligned}$$

Or in Matlab

$$\mathbf{w} = (X^T X) \setminus (X^T \mathbf{y})$$

This \mathbf{w} is unique if the columns of X are independent

22.2 Geometric derivation

Think of columns of X as vectors

$$\left[\begin{bmatrix} \mathbf{v}_1 \end{bmatrix} \quad \begin{bmatrix} \mathbf{v}_2 \end{bmatrix} \quad \dots \quad \begin{bmatrix} \mathbf{v}_n \end{bmatrix} \right]$$

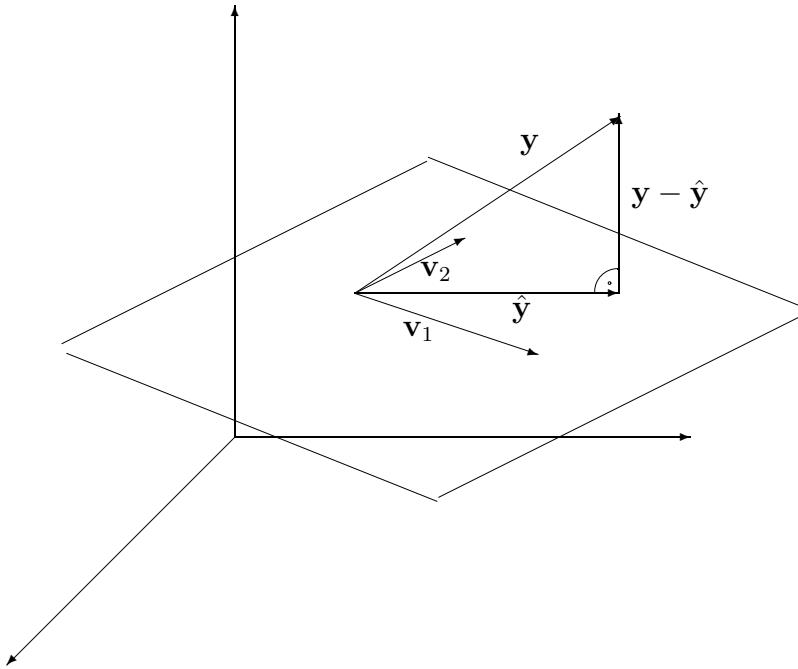
Looking for linear combination of columns that best approximates \mathbf{y}

For any weight vector \mathbf{w} get

$$\hat{\mathbf{y}} = w_1 \mathbf{v}_1 + w_2 \mathbf{v}_2 + \dots + w_n \mathbf{v}_n$$

Any such $\hat{\mathbf{y}}$ lies in $\text{span}(\mathbf{v}_1, \dots, \mathbf{v}_n)$

The element $\hat{\mathbf{y}}$ of $\text{span}(\mathbf{v}_1, \dots, \mathbf{v}_n)$ that minimizes $\|\hat{\mathbf{y}} - \mathbf{y}\|^2$ is given by the orthogonal projection of \mathbf{y} onto $\text{span}(\mathbf{v}_1, \dots, \mathbf{v}_n)$



Since the vector $\mathbf{y} - \hat{\mathbf{y}}$ is orthogonal to $\text{span}(\mathbf{v}_1, \dots, \mathbf{v}_n)$, the inner products $\mathbf{v}_i(\mathbf{y} - \hat{\mathbf{y}})$ are zero. That is, $\mathbf{v}_i \cdot \hat{\mathbf{y}} = \mathbf{v}_i \cdot \mathbf{y}$ for all $i = 1 \dots n$

The vectors \mathbf{v}_i are columns of the matrix X , so the equalities can be rewritten

$$X^T \hat{\mathbf{y}} = X^T \mathbf{y}$$

Remembering that $\hat{\mathbf{y}} = X\mathbf{w}$, we finally get

$$X^T X \mathbf{w} = X^T \mathbf{y}$$

Same as before

22.3 Other error functions

Sum of squared errors is not always the best error function

For example, could use

$$\begin{aligned} \text{err}(\hat{\mathbf{y}}, \mathbf{y}) &= \sum_i (\hat{y}_i - y_i)^2 && L_2 \text{ error} \\ &\quad \sum_i |\hat{y}_i - y_i| && L_1 \text{ error} \\ &\quad \sum_i |\hat{y}_i - y_i|^p && L_p \text{ error} \\ &\quad \max_i |\hat{y}_i - y_i| && L_\infty \text{ error} \end{aligned}$$

Example: Minimize maximum training error (L_∞)

- Choose \mathbf{w} to minimize $\max_i |\mathbf{w} \cdot \mathbf{x}_i - y_i|$
- Linear programming: find \mathbf{w}, δ to minimize δ , subject to constraints

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_1 &\leq y_1 + \delta \\ \mathbf{w} \cdot \mathbf{x}_1 &\geq y_1 - \delta \\ &\vdots \quad \vdots \quad \vdots \\ \mathbf{w} \cdot \mathbf{x}_t &\leq y_t + \delta \\ \mathbf{w} \cdot \mathbf{x}_t &\geq y_t - \delta \end{aligned}$$

- Can be done in polynomial time

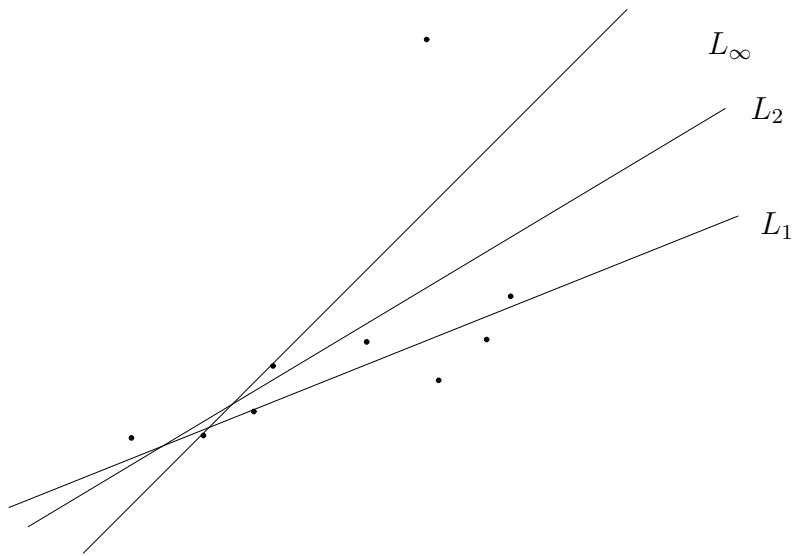
Example: Minimize sum of absolute error (L_1)

- Choose w to minimize $\sum_{i=1}^t |\mathbf{w} \cdot \mathbf{x}_i - y_i|$
- Linear programming: choose $\mathbf{w}, \boldsymbol{\delta}$ to minimize $\sum_{i=1}^t |\delta_i| = \sum_{i=1}^t \delta_i^+ + \delta_i^-$ subject to

$$\begin{aligned}
 \mathbf{w} \cdot \mathbf{x}_1 + \delta_1 &= \mathbf{w} \cdot \mathbf{x}_1 + \delta_1^+ - \delta_1^- = y_1 \\
 \mathbf{w} \cdot \mathbf{x}_2 + \delta_2 &= \mathbf{w} \cdot \mathbf{x}_2 + \delta_2^+ - \delta_2^- = y_2 \\
 &\vdots &&\vdots \\
 \mathbf{w} \cdot \mathbf{x}_t + \delta_t &= \mathbf{w} \cdot \mathbf{x}_t + \delta_t^+ - \delta_t^- = y_t
 \end{aligned}$$

Which objective is best?

- L_1 is the most robust against outliers
- L_2 is cheapest computationally
- L_∞ gives the best upper bound on training set error

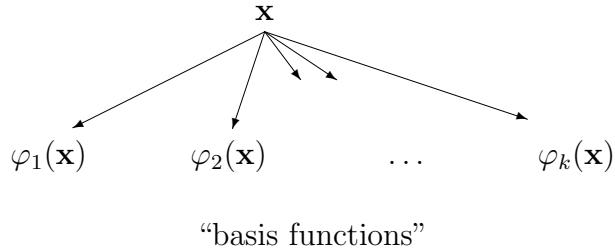


22.4 Generalized linear functions

Linear functions may not be expressive enough

Trick expand representation $\mathbf{x} \mapsto \varphi(\mathbf{x})$

Define new attributes which are non-linear functions of original attributes



Expand training set

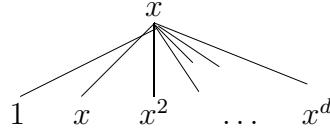
$$\begin{array}{c}
 \left[\begin{array}{ccc|c} x_{11} & \dots & x_{1n} & y_1 \\ x_{21} & \dots & x_{2n} & y_2 \\ \vdots & \ddots & \vdots & \vdots \\ x_{t1} & \dots & x_{tn} & y_t \end{array} \right] \\
 \Downarrow \\
 \left[\begin{array}{ccc|c} \varphi_1(\mathbf{x}_1) & \dots & \varphi_k(\mathbf{x}_1) & y_1 \\ \varphi_1(\mathbf{x}_2) & \dots & \varphi_k(\mathbf{x}_2) & y_2 \\ \vdots & \ddots & \vdots & \vdots \\ \varphi_1(\mathbf{x}_t) & \dots & \varphi_k(\mathbf{x}_t) & y_t \end{array} \right] \\
 \Phi \qquad \qquad \qquad \mathbf{y}
 \end{array}$$

Learn a linear function over expanded representation, which is *non-linear* in the original representation

E.g. learn non-linear function to minimize L_2 error by solving for \mathbf{w} such that $\Phi^T \Phi \mathbf{w} = \Phi^T \mathbf{y}$

Example: Polynomial regression

Let $X = \mathbb{R}$, and use the expanded set of all powers of x , up to d



Expand training set

$$\begin{array}{l} (x_1, y_1) \\ (x_2, y_2) \\ \vdots \\ (x_t, y_t) \end{array} \Rightarrow \left[\begin{array}{c|c} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \dots & x_t^d \end{array} \right] \begin{array}{l} y_1 \\ y_2 \\ \vdots \\ y_t \end{array}$$

To find the best polynomial, solve for \mathbf{w} in $\Phi^T \Phi \mathbf{w} = \Phi^T \mathbf{y}$

Obtain coefficients of minimal squared error polynomial of degree d

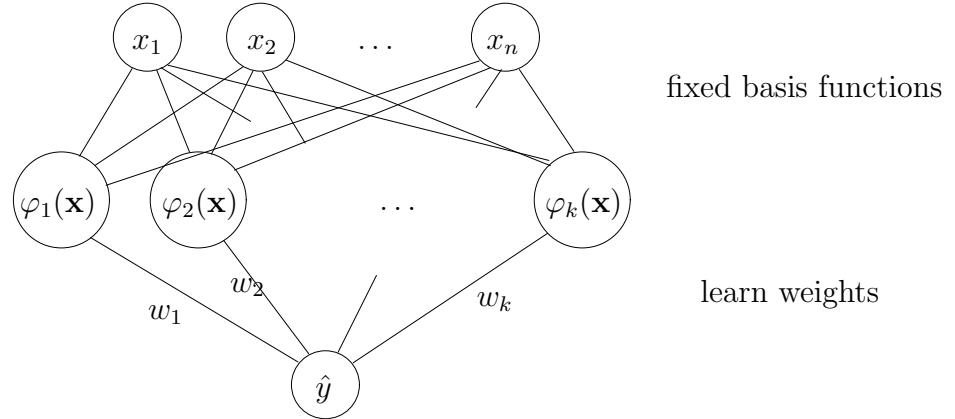
(Can also easily solve for min absolute error, and min maximum error polynomials, by exploiting the same basis expansion and using the linear programming formulations shown earlier to calculate \mathbf{w})

Other basis functions

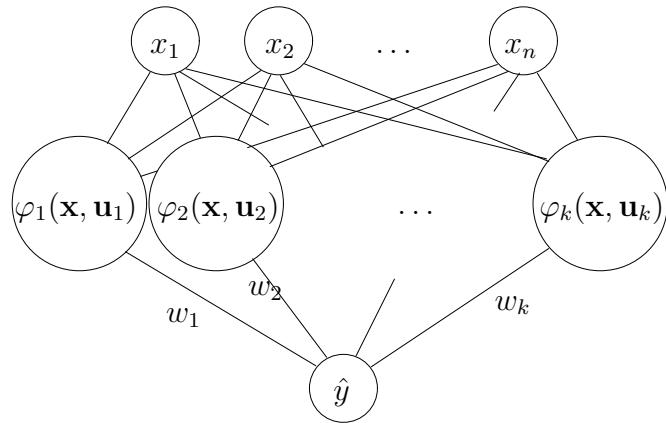
- spline fitting (basis splines)
- radial basis functions
- Fourier analysis

22.5 Neural networks

Learning a generalized linear function could be depicted as



However, in addition to the weights \mathbf{w} at the final level, we could also try to learn the basis functions φ_i themselves. So add parameters \mathbf{u}_i to φ_i , and attempt to learn \mathbf{u}_i parameters in addition to \mathbf{w} parameters



In total, have to learn weights $\mathbf{w}, \mathbf{U} = [\mathbf{u}_1; \dots; \mathbf{u}_k]$ to minimize SSE

- Unfortunately, the problem is *NP-hard*. There is no general polynomial time training procedure.

- Can use gradient descent optimization in \mathbf{w}, U to heuristically find a local minimum

“Backpropagation algorithm”

- Uses efficient scheme for calculating weight gradients using chain rule of differentiation

Readings

Russell and Norvig: Chapter 20

Dean, Allen, Aloimonos: Sections 5.5, 5.6, 5.8

Hastie, Tibshirani, Friedman: Chapters 2, 3, 11