Linear Prediction (Continued) and Generalizations

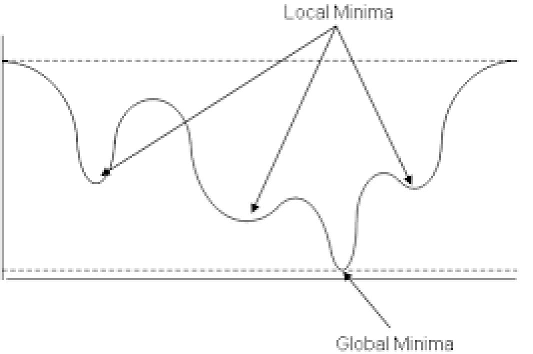
Thanks to Professor Dale Schuurmans
University of Alberta and Google Brain

Instructor: Farzaneh Mirzazadeh
Department of Computer Science, UCSC, Winter 2017

Math background: Reminder

Global and Local Min

- A point x^* is a global minimizer of a function f, if $f(x^*) \leq f(x)\ \forall x$.
- A point x^* is a local minimizer of a a function f if there is a neighborhood N of x^*, such that if $f(x^*) \leq f(x)\ \forall x \in N$.

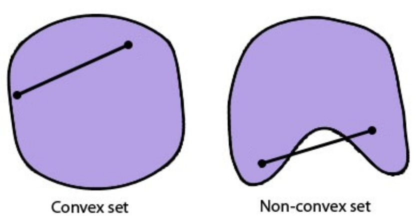


Math background: Convexity

- A fundamental property in optimization
- Many practical problems, possess this property, which generally makes them easier to solve.
- Applies to both sets and functions

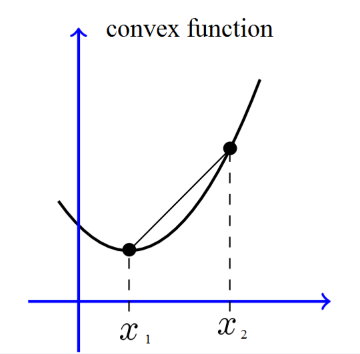
Convex Set

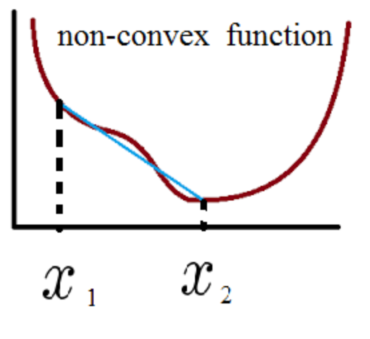
- A set $S \in \mathbb{R}^n$ is a convex set, if the straight line segment connecting any two points in S lies entirely inside S.
- Formally, for any two points $x_1 \in S$ and $x_2 \in S$ we have $\alpha x_1 + (1 - \alpha)x_2 \in S, \forall \alpha, \alpha \in [0, 1]$.



Math background: Convexity

Convex Function

- A function f is convex if its domain is a convex set, and for any two points in the domain, the straight line segment connecting them together lies above or on the function.
- $f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2), \forall \alpha \in [0, 1]$
- 

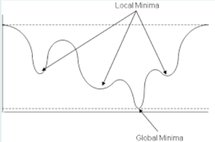


Optimization in general

Convex

- Local min = global min
- Efficient guaranteed algorithms

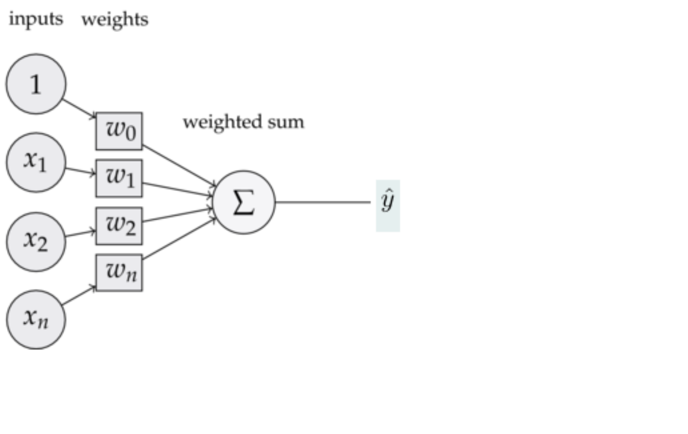
Non-Convex

- Local min $\neq$ global min 

- Possibly many local minina
- Usually finding global min is NP-hard
- Usually no efficient guaranteed algorithm
- Use **local descent** algorithms (like gradient descent) + possible **heuristic restart method** (to restart search after getting stock in a local min)

Questions?

Back to linear prediction



Reminder: linear prediction

Setting:

1. Targets
   $y \in \mathbb{R}$, i.e. **real** valued **scalar** targets
2. Inputs
   $\mathbf{x} \in \mathbb{R}^n$, i.e. real valued vector inputs. (In other words each training example n features.)
3. Hypotheses
   $h(.) : h_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$, i.e linear hypotheses
   h is a linear function of $\mathbf{x}$.
4. Hypothesis class
   $H = \{h_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^n\}$, i.e. set of all linear functions. (Note: Changing $\mathbf{w}$ changes the linear function)
5. Error function
   Options: sum of squares error (L_2 based error), sum of absolute error (L_1 error), max absolute error(L_∞ error), etc.

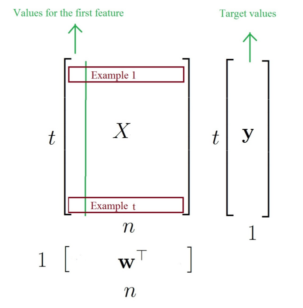
Reminder: linear prediction

I. Training phase

Given X, $\mathbf{y}$,
compute $\mathbf{w}^*$ such that

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{t} err(X_{i:}\mathbf{w}, \mathbf{y}_i)$$

In other words, find parameter $\mathbf{w}$ of the hypothesis whose predictions over training data has smallest total error.



II. Testing phase (prediction phase)

Make prediction for each future test example $\mathbf{x}_\circ$ using

$$\hat{y} = h_{\mathbf{w}^*}(\mathbf{x}_\circ) = \mathbf{x}_\circ^\top \mathbf{w}^*$$

(In other words, find the value of learned function at $\mathbf{x}_\circ$. Return it as the prediction.)

Reminder: Training procedure

- If err is L_1 norm of residual (absolute error)

$$err(\hat{\mathbf{y}}, \mathbf{y}_) = \sum_{i=1}^{t} |\hat{\mathbf{y}}_i - \mathbf{y}_i| \implies \quad \text{Linear program}$$

- Iif $err()$ based on L_2 norm of residual (squared error)

$$err(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{t} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \implies \quad \text{Analytical solution}$$

- If $err()$ is L_∞ norm of residual (max absolute error)

$$err(\hat{\mathbf{y}}, \mathbf{y}) = \max_i |\hat{\mathbf{y}}_i - \mathbf{y}_i| \implies \quad \text{Linear program}$$

Minimize L_2 based training error

Training problem (Least Squares)

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{t} (X_{i:}\mathbf{w} - y_i)^2$$

How to solve?

- Note: Objective function is convex quadratic $\implies$ Any local min is a global min.

$$\begin{aligned} J(\mathbf{w}) &= \sum_{i=1}^{t} (X_{i:}\mathbf{w} - y_i)^2 \\ &= (X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y}) \end{aligned}$$

- Find a local min, since for a continuous function it is easy to find. (How?)
- Set the gradient to zero. Find $\mathbf{w}^*$.

Minimize L_2 based training error

Training problem (Least Squares)

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} (X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y})$$

How to solve?

$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{w}) &= 2X^\top (X\mathbf{w} - \mathbf{y}) = 0 \\ &= 2(X^\top X\mathbf{w} - X^\top \mathbf{y}) = 0 \end{aligned}$$

- Set the gradient wrt to $\mathbf{w}$ to zero.

$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{w}) &= 2X^\top (X\mathbf{w} - \mathbf{y}) = 0 \\ &= 2(X^\top X\mathbf{w} - X^\top \mathbf{y}) = 0 \end{aligned}$$

- To find $\mathbf{w}$, solve for $\mathbf{w}$ the system of equations

$$X^\top X\mathbf{w} - X^\top \mathbf{y} = 0.$$

- $\mathbf{w}^*$ unique if the columns of X linearly independent, i.e. if features are linearly independent.

Geometry of minimum L_2 based error

Think of columns of X as vectors.

$$X = \begin{array}{c} \left[\begin{array}{c|c|c|c} X_{:1} & X_{:2} & \dots & X_{:n} \end{array} \right] \\ \begin{array}{cccc} w_1 & w_2 & \dots & w_n \end{array} \end{array}$$

Linear combination of vectors

$$\begin{bmatrix} X_{:1} \end{bmatrix} w_1 + \begin{bmatrix} X_{:2} \end{bmatrix} w_2 + \ldots + \begin{bmatrix} X_{:n} \end{bmatrix} w_n = \begin{bmatrix} \hat{y} \end{bmatrix} \approx \begin{bmatrix} y \end{bmatrix}$$

Looking for linear combination of columns of X that is closest to $\mathbf{y}$

$$(X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y}) = (\hat{\mathbf{y}} - \mathbf{y})^\top (\hat{\mathbf{y}} - \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$$

But notice that any $\hat{\mathbf{y}} \in ColSpan(X)$. Therefor $\hat{\mathbf{y}}$ that minimizes $\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$ is given by (by definition of projection onto a subspace)

$$\hat{\mathbf{y}} = \text{orthogonal projection of } \hat{\mathbf{y}} \text{ onto column span}(X)$$

Euclidean projection

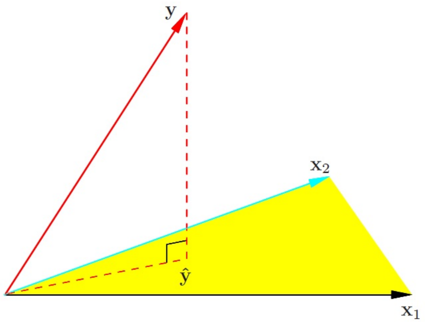


FIGURE 3.2. *The N-dimensional geometry of least squares regression with two predictors. The outcome vector* $\mathbf{y}$ *is orthogonally projected onto the hyperplane spanned by the input vectors* $\mathbf{x}_1$ *and* $\mathbf{x}_2$*. The projection* $\hat{\mathbf{y}}$ *represents the vector of the least squares predictions*

Special cases: co-linear features, orthonormal features

Co-linear features

- Becomes numerically unstable if two columns $X_{:j}$, $X_{:k}$ are nearly co-linear.
- If nearly co-linear need large weights to represent small $\hat{\mathbf{y}}$ very bad

Orthonormal features

- An interesting special case when features (columns of X) are orthonormal.

$$X^\top X = I$$

implies $\mathbf{w}^* = X^\top \mathbf{y}$ minimizes L_2 error

$$\mathbf{w}_j^* = X_{:j}^\top \mathbf{y}$$

can compute wights just by taking dot product.

Choice of prediction error function

Which prediction error function is best to use?

- L_1 linear program $\approx O(n^3)$
- L_2 solving linear system $O(n^3)$
- L_∞ linear program $\approx O(n^3)$

Comparison:

- L_2 is easiest to solve computationally. But the others are not so bad computationally.
- L_1 is the most robust to outliers.

Where to read?

- Linear Prediction
  - Linear prediction: Hasie et al., 2nd ed(2009), Sec 2.3.1, Sec 3.1
  - Bishop (2006), Sec 3.1
  - Shaw-Tayloe and Cristinani (2004), Sec 2.2.1
  - Cherkassky and Mulier (1998) Sec 7.2.1
- Linear Algebra: Book by G. Strang
  `https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/`
- Optimization
  - Nocedal et al (chapters 1 and 2)
- Convex optimization, minimizing different norms
  - Boyd and Vandenberghe (3 parts: Theory, Application, Algorithms)
- Coursera Machine Learning Course, Andrew Ng

A generalization to linear prediction: feature expansion

Linear model on an expended set of features

Feature expansion

- Simple linear functions often not expressive enough
- It is possible and useful to expand representation
- New features could be nonlinear function of old features
- Mapping each example to a new representation: $\mathbf{x} \to \phi(\mathbf{x})$
- Example

	new feature	example	meaning
1	$\phi_1(\mathbf{x})$	$\phi_1(\mathbf{x}) = \mathbf{x}_1^2$	Square of first feature
2	$\phi_2(\mathbf{x})$	$\phi_2(\mathbf{x}) = \mathbf{x}_1 * \mathbf{x}2$	Product of first and second features
	$\vdots$		
L	$\phi_L(\mathbf{x})$	$\phi_L(\mathbf{x}) = \exp(\mathbf{x}_3)$	Exponential of third feature

- New features are new basis functions.
- Expand training set. (with L features), then new training matrix Φ

$$X \to \Phi, \quad X \in \mathbb{R}^{t \times n}, \Phi \in \mathbb{R}^{t \times L}$$

Linear model on the expanded feature set

Step 1: Forming new representation

- Expand training set

$$X \to \Phi$$

Step 2: train phase. Learn a linear model for the new training matrix

- Learn linear functions over extended features (a nonlinear function of the original features)
- I.e. Learn an $L \times 1$ vector $\mathbf{w}$ via, $\min_{\mathbf{w}} \sum_{i=1}^{t} Err(\Phi_{i:}\mathbf{w}, \mathbf{y}_i)$

Step 3: test phase

- Learned predictor: given test example $\mathbf{x}_\circ$, re-represent it with new features, then predict via:

$$\hat{y} = \sum_{j=1}^{L} \mathbf{w}_j \phi_j(\mathbf{x}_\circ) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_\circ)$$

Example

Polynomial basis

Assume a data X set with only one scalar feature $x \in \mathbb{R}$. Training matrix:

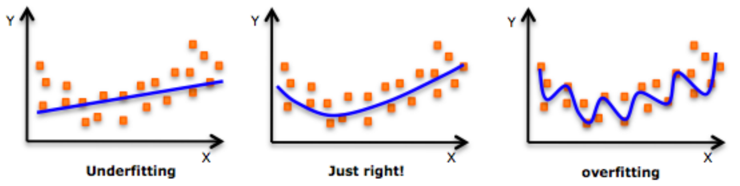
$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_t \end{bmatrix}$$

Size of original training matrix with one feature: $t \times 1$
New training features: the kth power of the feature, $0 \leq k \leq d$

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ \vdots & & & & \\ 1 & x_t & x_t^2 & \dots & x_t^d \end{bmatrix}$$

Size of new training matrix, with $d+1$ features: $t \times (d+1)$

How many basis functions (features) to use?



Overfitting vs. Underfitting

- Too many basis functions (too many features)
- Too few basis functions (too few features)

Generalization behavior

- Training error
- Test error

How to prevent overfitting?

- One way is to penalize large number of features by adding a relevant term to the objective function
- If not possible a proxy to that
- Called: regularization (next class)