UVA CS 4501 - 001 / 6501 – 007 Introduction to Machine Learning and Data Mining

Lecture 4: More optimization for Linear Regression

Yanjun Qi / Jane

University of Virginia
Department of
Computer Science

Last Lecture Recap

- ❑ Linear regression (aka **least squares**)
- ❑ Learn to derive the least squares estimate by optimization
- ❑ Evaluation with Cross-validation

e.g. SUPERVISED LEARNING

$$f : X \longrightarrow Y$$

- Find function to map input space X to output space Y

- Generalisation: learn function / hypothesis from past data in order to "explain", "predict", "model" or "control" new data examples

KEY

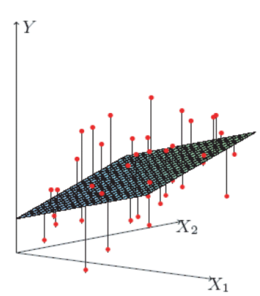
Linear Regression Models

$$f : X \longrightarrow Y$$

➔ e.g. Linear Regression Models

$$\hat{y} = f(x) = \theta_0 + \theta_1 x^1 + \theta_2 x^2$$

- Features:
  Living area, distance to campus, # bedroom …
- Target y:
  Rent ➔ Continuous



training / learning goal

- Using matrix form, we get the following general representation of the linear function on train set:

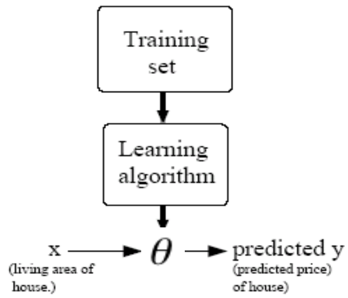
$$\hat{Y} = X\theta$$

n×1   n×p   p×1

- Our goal is to pick the optimal θ that minimize the following cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i(\vec{x}_i) - y_i)^2$$

Our goal:



Method I: normal equations

- Write the cost function in matrix form:

$$\begin{aligned} J(\theta) &= \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2 \\ &= \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\ &= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \end{aligned}$$

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

To minimize $J(\theta)$, take its gradient and set to zero:

$$\Rightarrow X^T X\theta = X^T\vec{y}$$

The normal equations

$$\Downarrow$$

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

e.g. 10 fold Cross Validation

- Divide data into 10 equal pieces
- 9 pieces as training set, the rest 1 as test set
- Collect the scores from the diagonal

model	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	train	train	train	train	train	train	train	train	train	test
2	train	train	train	train	train	train	train	train	test	train
3	train	train	train	train	train	train	train	test	train	train
4	train	train	train	train	train	train	test	train	train	train
5	train	train	train	train	train	test	train	train	train	train
6	train	train	train	train	test	train	train	train	train	train
7	train	train	train	test	train	train	train	train	train	train
8	train	train	test	train	train	train	train	train	train	train
9	train	test	train	train	train	train	train	train	train	train
10	test	train	train	train	train	train	train	train	train	train

Today

- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD for LR

Review: Definitions of gradient (from Stanford handout)

Suppose that $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ is a function that takes as input a matrix A of size $m \times n$ and returns a real value. Then the ***gradient*** of f (with respect to $A \in \mathbb{R}^{m\times n}$) is the matrix of

$$\nabla_A f(A) \in \mathbb{R}^{m\times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

Review: Definitions of gradient (from Stanford handout)

- Size of gradient is always the same as the size of

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \quad \text{if } x \in \mathbb{R}^n$$

Review: Definitions of gradient

(from http://en.wikipedia.org/wiki/Matrix_calculus#Scalar-by-vector)

The derivative of a scalar *y* function of a matrix **X** of independent variables, with respect to the matrix **X**, is given as

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{p1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{p2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1q}} & \frac{\partial y}{\partial x_{2q}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix}$$

Notice that the indexing of the gradient with respect to **X** is transposed as compared with the indexing of **X**.

(from http://en.wikipedia.org/wiki/Matrix_calculus#Scalar-by-vector)

The derivative of a scalar *y* by a vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, is

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

This gradient is a 1×n row vector whose entries respectively contain the n partial derivatives

Review: Derivative of a Function

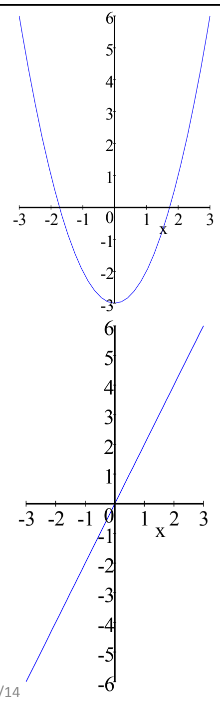
$\lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$ is called the derivative of f at a.

We write: $f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$

"The derivative of f with respect to x is …"

There are many ways to write the derivative of $y = f(x)$

➔ e.g. define the slope of the curve y=f(x) at the point x



Review: Derivative of a Quadratic Function

$$y = x^2 - 3$$

$$y' = \lim_{h \to 0} \frac{(x+h)^2 - 3 - (x^2 - 3)}{h}$$

$$y' = \lim_{h \to 0} \frac{x^2 + 2xh + h^2 - x^2}{h}$$

$$y' = \lim_{h \to 0} 2x + h \quad (h \to 0)$$

$$y' = 2x$$

Today

- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD for LR

A little bit more about [Optimization]

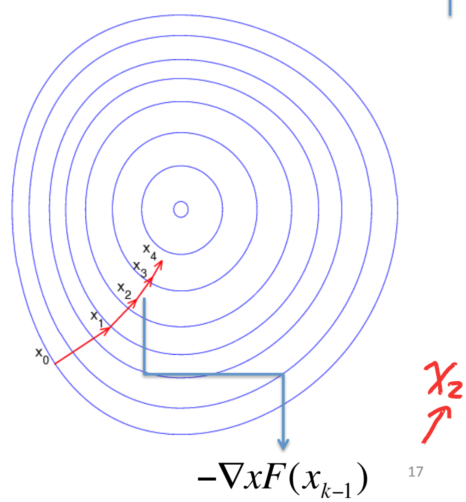
- Objective function $F(x)$
- Variables x
- Constraints

To find values of the variables
that minimize or maximize the objective function
while satisfying the constraints

e.g. Gradient Descent (Steepest Descent)

A first-order optimization algorithm.

To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient of the function at the current point.



Gradient Descent (GD)

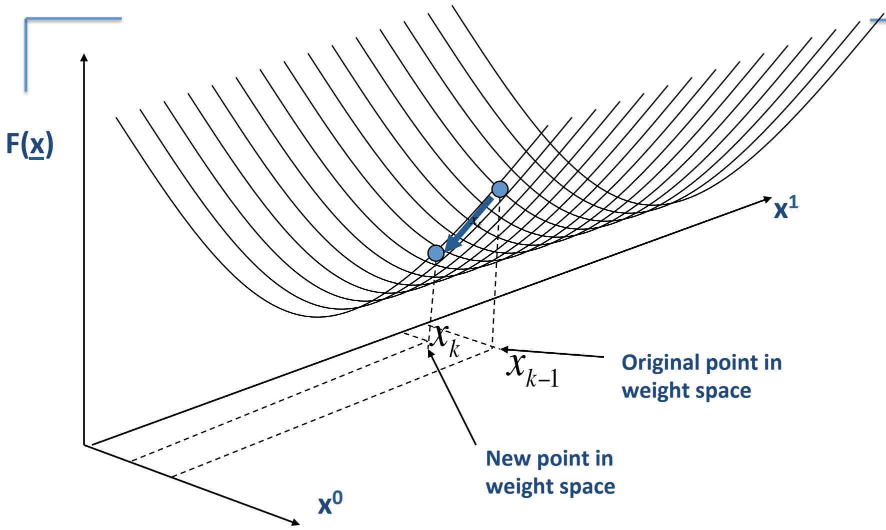
- Initialize k=0, choose x_0
- While $k<k_{max}$

For the k-th epoch

$$x_k = x_{k-1} - \alpha \nabla x F(x_{k-1})$$

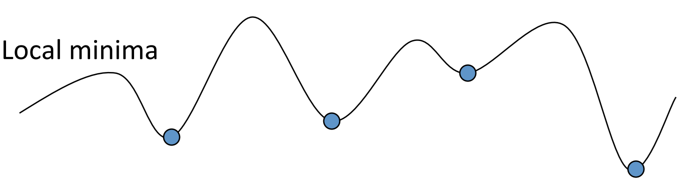
Please READ this note to clarify the confusion : http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gradientDescent.pdf

Illustration of Gradient Descent (2D case)



Comments on Gradient Descent Algorithm

- Works on any objective function F(w)
  - as long as we can evaluate the gradient
  - this can be very useful for minimizing complex functions E
- Local minima



  - Can have multiple local minima
  - (note: for LR, its cost function only has a single global minimum, so this is not a problem)
  - If gradient descent goes to the closest local minimum:
    - solution: random restarts from multiple places in weight space

Method III: LR with batch GD

- The Cost Function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$

- Consider a **gradient descent** algorithm:

$$\theta_j^{t+1} = \theta_j^t - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\Big|_t$$

For the (t+1)-th epoch

$$J(\theta) = (X\theta - y)^T(X\theta - y)$$

$$= ((X\theta)^T - y^T)(X\theta - y)$$

$$= (\theta^T X^T - y^T)(X\theta - y)$$

$$= \theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y$$

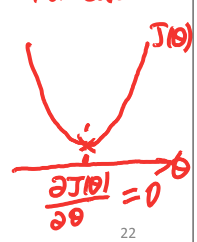
Since $\theta^T X^T y = y^T X\theta$

$\langle X\theta, y\rangle \quad \langle y, X\theta\rangle$

$$= \theta^T X^T X\theta - 2\theta^T X^T y + y^T y$$

$\Rightarrow J(\theta)$ quadratic func of θ;

1d case



See handout 4.1 + 4.3 ⇒ matrix calculus, partial deri ⇒ Gradient

$$\nabla_\theta (\theta^T X^T X \theta) = 2X^T X \theta \quad (P24)$$

$$\nabla_\theta (-2\theta^T X^T y) = -2X^T Y \quad (P24)$$

$$\nabla_\theta (y^T y) = 0$$

$$\Rightarrow \nabla_\theta J(\theta) = X^T X \theta - X^T Y$$

Illustration of Gradient Descent (2D case)

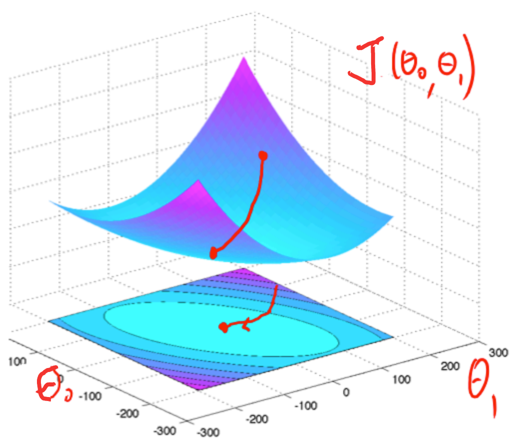
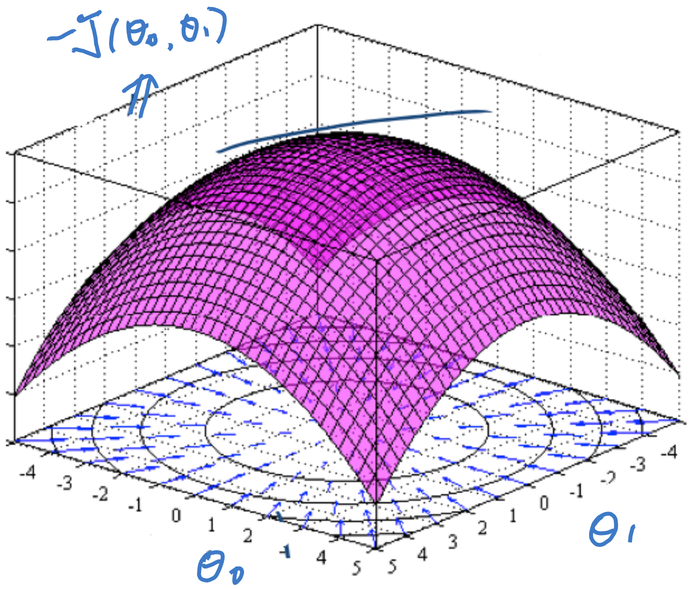


Illustration of Gradient Descent (2D case)



$$\nabla_\theta J(\theta) = X^T X\theta - X^T Y$$

$$= X^T (X\theta - Y)$$

$$= X^T \left(\begin{bmatrix} - x_1^T - \\ - x_2^T - \\ \vdots \\ - x_n^T - \end{bmatrix}_{n\times p} \theta_{p\times 1} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right)$$

$$= X^T_{p\times n} \begin{bmatrix} x_1^T\theta - y_1 \\ x_2^T\theta - y_2 \\ \cdots \\ x_n^T\theta - y_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} x_1^T\theta - y_1 \\ \vdots \\ x_n^T\theta - y_n \end{bmatrix}$$

$$= \sum_{i=1}^{n} x_i (x_i^T\theta - y_i)$$

LR with batch GD

- Steepest descent / GD
  - Note that:

$$\theta_j^{t+1} = \theta_j^{t} + \alpha \sum_{i=1}^{n} (y_i - \vec{\mathbf{x}}_i^{T} \theta^t) x_i^j$$

Update Rule Per Feature Variable-Wise

$$\nabla_\theta J = \left[\frac{\partial}{\partial \theta_1} J, \ldots, \frac{\partial}{\partial \theta_k} J\right]^T = -\sum_{i=1}^{n} (y_n - \mathbf{x}_n^T \theta) \mathbf{x}_n$$

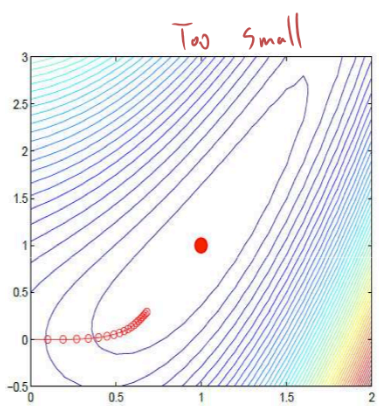
Based on Stanford Handout's Definition of Gradient

$$\theta^{t+1} = \theta^{t} + \alpha \sum_{i=1}^{n} (y_n - \mathbf{x}_n^{T} \theta^t) \mathbf{x}_n$$

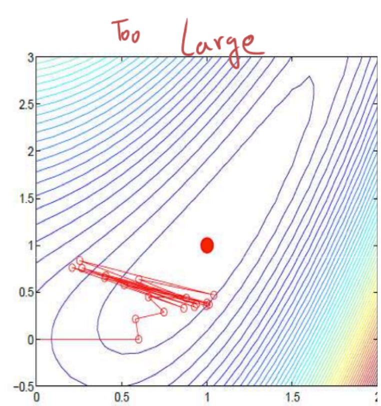
- This is as a **batch** gradient descent algorithm

Please READ this note to clarify the confusion : http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gradientDescent.pdf

Choosing the Right Step-Size / Learning-Rate is critical



$\alpha = 0.1.$



$\alpha = 0.6.$

Method III: LR with Stochastic GD ➔

- Now we have the following descent rule:

- For a single training point, we have:

$$\theta^{t+1} = \theta^t + \alpha(y_i - \vec{\mathbf{x}}_i^{\ T}\theta^t)\vec{\mathbf{x}}_i$$

  - This is known as the Least-Mean-Square update rule, or the Widrow-Hoff learning rule
  - This is actually a "**stochastic**", "**coordinate**" descent algorithm
  - This can be used as a **on-line** algorithm

Summary: three ways to learn LR

- Normal equations $\theta^* = \left(X^T X\right)^{-1} X^T \vec{y}$
  - Pros: a single-shot algorithm! Easiest to implement.
  - Cons: need to compute pseudo-inverse $(X^TX)^{-1}$, expensive, numerical issues (e.g., matrix is singular ..), although there are ways to get around this …
- GD or Steepest descent $\theta^{t+1} = \theta^t + \alpha\sum_{i=1}^{n}(y_n - \mathbf{x}_n^{\ T}\theta^t)\mathbf{x}_n$
  - Pros: easy to implement, conceptually clean, guaranteed convergence
  - Cons: batch, often slow converging
- Stochastic LMS update rule $\theta_j^{t+1} = \theta_j^t + \alpha(y_n - \mathbf{x}_n^T\theta^t)x_{n,j}$
  - Pros: on-line, low per-step cost, fast convergence and perhaps less prone to local optimum
  - Cons: convergence to optimum not always guaranteed

Direct (normal equation) vs. Iterative (GD) methods

- Direct methods: we can achieve the solution in a single step by solving the normal equation
  - Using Gaussian elimination or QR decomposition, we converge in a finite number of steps
  - It can be infeasible when data are streaming in in real time, or of very large amount
- Iterative methods: stochastic or steepest gradient
  - Converging in a limiting sense
  - But more attractive in large practical problems
  - Caution is needed for deciding the learning rate α

Convergence rate

- **Theorem**: the steepest descent equation algorithm converge to the minimum of the cost characterized by normal equation:

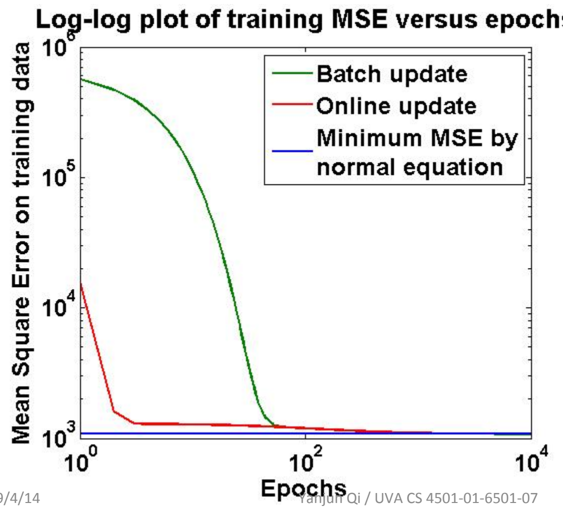
$$\theta^{(\infty)} = (X^T X)^{-1} X^T y$$

If the learning rate parameter satisfy ➔

$$0 < \alpha < 2/\lambda_{\max}[X^T X]$$

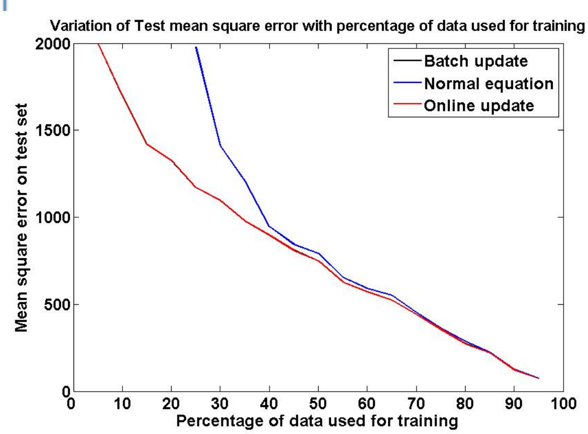
- A formal analysis of LMS need more math; in practice, one can use a small α, or gradually decrease α.

Convergence Curves, for an example



- For the batch method, the training MSE is initially large due to uninformed initialization
- In the online update, N updates for every epoch reduces MSE to a much smaller value.

Performance vs. Training Size for an example



- The results from B and O update are almost identical. So the plots coincide.
- The test MSE from the normal equation is more than that of B and O during small training. This is probably due to overfitting.
- In B and O, since only 2000 (for example) iterations are allowed at most. This roughly acts as a mechanism that avoids overfitting.

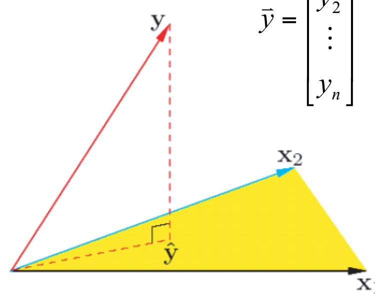
Geometric Interpretation of Least Mean Square Solution

- The predictions on the training data are:
- Note that $\hat{\vec{y}} = X\theta^* = X\left(X^T X\right)^{-1} X^T \vec{y}$

and $\hat{\vec{y}} - \vec{y} = \left(X\left(X^T X\right)^{-1} X^T - I\right)\vec{y}$

$$X^T\left(\hat{\vec{y}} - \vec{y}\right) = X^T\left(X\left(X^T X\right)^{-1} X^T - I\right)\vec{y}$$
$$= \left(X^T X\left(X^T X\right)^{-1} X^T - X^T\right)\vec{y}$$
$$= 0 \quad !!$$

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$



$\hat{\vec{y}}$ → the orthogonal projection of the true y vector into the space spanned by the columns of X

Today Recap

- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD for LR

References

- Big thanks to Prof. Eric Xing @ CMU for allowing me to reuse some of his slides

- **Notes about Gradient Descent from Toussaint: (please read) http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gradientDescent.pdf**
- http://en.wikipedia.org/wiki/Matrix_calculus#Scalar-by-vector