Introduction to Machine Learning
Gradient Descent and Normal Equations

Nikolay Manchev

July 30, 2016

London Machine Learning Study Group

Housekeeping

Next events

http://www.meetup.com/London-Machine-Learning-Study-Group

Follow me

https://twitter.com/nikolaymanchev

Slides and code

Available at https://github.com/nmanchev/MachineLearningStudyGroup

Outline

Linear Regression and Gradient Descent

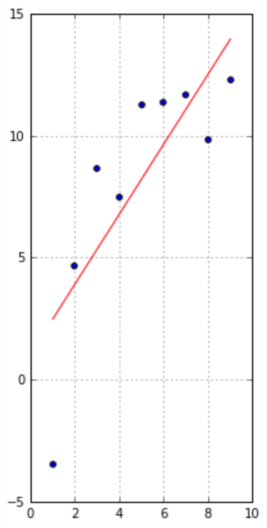
Normal Equations

Linear Regression and Gradient Descent

Univariate Linear Regression

Fitting a linear regression model

$$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}^T$$
$$\boldsymbol{y} = \{y_1, y_2, \ldots, y_N\}^T$$

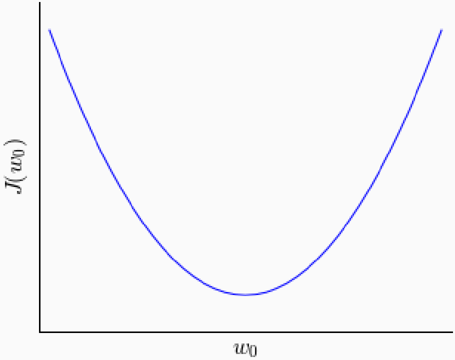


Cost function minimisation

Minimising the cost function leads us to the coefficients of the best fitting line.

$$J(\boldsymbol{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Gradient Descent



Update rule

The derivative $\frac{d}{dw_0}J(w_0)$ provides the slope of the tangent line to the graph of the function at w_0

repeat until convergence $\{$

$w_0 := w_0 - \alpha\frac{d}{dw_0}J(w_0)$

$\}$

- A positive $\alpha\frac{d}{dw_0}J(w_0)$ moves w_0 to the left
- A negative $\alpha\frac{d}{dw_0}J(w_0)$ moves w_0 to the right

Matrix Notation

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{bmatrix}$$

Hypothesis: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$

Cost function: $J(\boldsymbol{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$

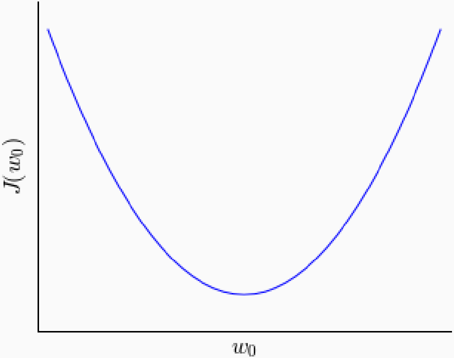
Update rule:

repeat until convergence $\{$

$$\boldsymbol{w} := \boldsymbol{w} - \alpha\frac{\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})}{N}$$

$\}$

Choice of alpha



Not an easy choice

$$w_0 := w_0 - \alpha \frac{d}{dw_0} J(w_0)$$

- Small alpha – slow convergence
- Large alpha – risk of overshooting

- Lipschitz continuity ($\alpha = \frac{1}{L}$)
- L not readily available – optimise manually
- Backtracking line search

Backtracking line search

Determine the maximum move along a given direction

Start with a large α and iteratively shrink it until an adequate decrease of the objective function.

Given a starting position $\boldsymbol{w}$ and a search direction $\boldsymbol{p}$, we want to find a value of α that reduces $J(\boldsymbol{w} + \alpha\, \boldsymbol{p})$ relative to $J(\boldsymbol{w})$.

Backtracking Line Search

1. Select α_0 and control parameters $c \in (0, 1)$ and $\tau \in (0, 1)$
2. Set $j = 0$ and $t = -c\, \boldsymbol{p}$
3. **repeat** $\{$

$$j := j + 1$$
$$\alpha_j = \tau \alpha_{j-1}$$

$\}$ **until** $\{\ J(\boldsymbol{w}) - J(\boldsymbol{w} + \alpha_j\, \boldsymbol{p}) \geq \alpha_j t\ \}$

Generic Line Search

1. pick an initial $\boldsymbol{w}$
2. **repeat until convergence** $\{$

    2.1 calculate a search direction $\boldsymbol{p}$ (descent direction)

    2.2 find an optimal α

    2.3 $\boldsymbol{w} := \boldsymbol{w} + \alpha \boldsymbol{p}$

    $\}$

Normal Equations

Closed Form Solution

$$\frac{\partial}{\partial w} J(\boldsymbol{w}) = \frac{\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})}{N}$$

$$0 = \frac{\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})}{N}$$
$$0 = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$
$$0 = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y}$$
$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}$$

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Gradient Descent vs. Normal Equations

Normal Equations

- No need to choose α
- No need to iterate to reach convergence
- Computing $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ is expensive – $O(n^3)$

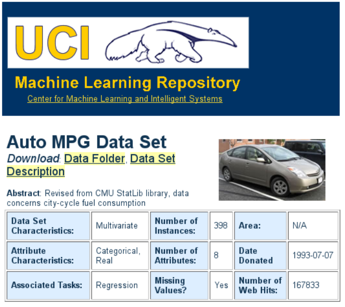
Gradient Descent

- Works well for large n
- Can produce a "good enough" solution with a run-time that's order of magnitude smaller [BB08]
- General optimisation algorithm

Data Set Example

UCI Machine Learning Repository – `archive.ics.uci.edu/ml`

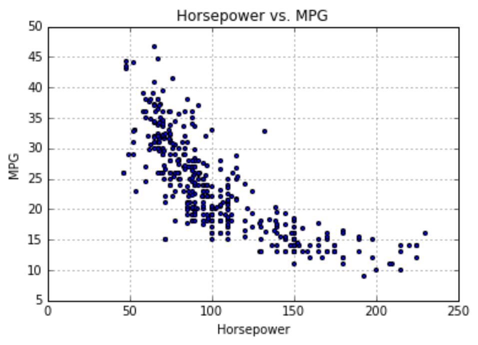
- Great resource for Machine Learning data sets
- Over 330 freely available sets
- Auto MPG Data Set
    - Fuel consumption in MPG
    - Attributes: mpg, cylinders, displacement, horsepower, weight, acceleration etc.



MPG vs Horsepower

Simple use-case

- Auto MPG Data Set
- Predicting *MPG* based on *Horsepower*



Z-score Normalization

- Rescale the features to give them properties of a standard normal distribution ($\mu = 0$, $\sigma = 1$).
- Z-score normalization is calculated as

$$z = \frac{x - \mu}{\sigma}$$

  where

  μ is the mean of the population

  σ is the standard deviation

- General requirement for many machine learning algorithms
- Helps Gradient Descent converge faster

Z-score Normalization

Why normalizing the inputs works

- Gradient descent is curvature ignorant
- Brings all features to the same scale
- Gives the error surface a spherical shape. Look at [Hin14]

References I

- Olivier Bousquet and Léon Bottou, *The tradeoffs of large scale learning*, Advances in Neural Information Processing Systems 20 (J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, eds.), Curran Associates, Inc., 2008, pp. 161–168.
- Geoffrey Hinton, *Neural Networks for Machine Learning, Lecture 6b*, Video Lecture, 2014.