Gradient Descent and Related Methods

Jack Bennetto

September 26, 2016

Objectives

- Explain how **gradient descent** works
- Use gradient descent to optimize the cost function for logistic regression
- Explain the advantage of **stochastic gradient descent**
- Implement stochastic gradient descent
- Implement **Newton's method**

Agenda

- Morning
  1. What is gradient descent and why do we need it?
  2. Examples of gradient descent
  3. What can go wrong?
  4. Using gradient descent to solve logistic regression
- Afternoon
  1. Stochastic Gradient Descent
  2. Newton's Method

Cost Functions

- Machine learning often involves fitting a model to test data
- The best fit is often determined using a *cost function* or *likelihood function*
  - Linear Regression:
$$\sum (y_i - \beta^T \mathbf{x}_i)^2$$
  - Logistic Regression:
$$\sum y_i \log g(\beta^T \mathbf{x}_i) + (1 - y_i) \log(1 - g(\beta^T \mathbf{x}_i))$$
$$\left(g(z) = \frac{1}{1+e^{-z}} \right)$$

Linear Regression

The cost function $\sum(y_i - \beta^T \mathbf{x}_i)^2$ can be represented in matrix format:

$$||\mathbf{y} - X\beta||^2$$

Has a closed-form solution for the minimum

$$\beta = (X^T X)^{-1} X^T \mathbf{y}$$

Why is this infeasible sometimes?

Logistic Regression

The log-likelihood function

$$\sum y_i \log g(\beta^T \mathbf{x}_i) + (1 - y_i) \log(1 - g(\beta^T \mathbf{x}_i))$$

has no such closed form for its maximum.

How will you find the maximum?

Gradient Descent

Basic gradient-descent algorithm to find a minimum

- Choose a point and
- Calculate the gradient (direction of fastest ascent)
- Step in the opposite direction
- Repeat

How would you find a maximum?

Recall

- The *gradient* of a multivariate function $f(x_1, \ldots, x_n)$ is

$$\nabla f(\mathbf{a}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{a}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{a}) \right)$$

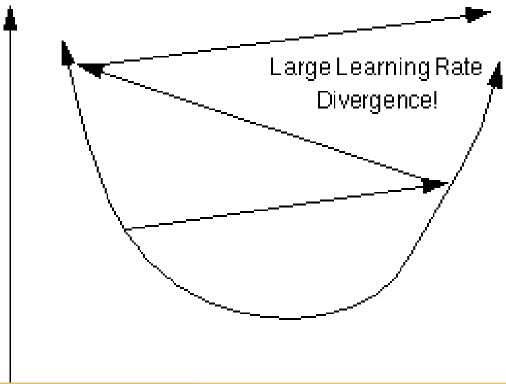
- $\nabla f(\mathbf{a})$ points in the direction of greatest increase of f at $\mathbf{a}$

Gradient Descent

To minimize f

- Choose:
  - a starting point $\mathbf{x}_0$
  - *learning rate* α
  - threshold ϵ
- Move in the direction of $-\nabla f(\mathbf{x})$:
  - Update $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$
- If $\frac{|f(\mathbf{x}_i) - f(\mathbf{x}_{i+1})|}{|f(\mathbf{x}_i)|} < \epsilon$, return $f(\mathbf{x}_{i+1})$ as the min, and $\mathbf{x}_{i+1}$ as the argmin

- `alpha` is called the *step-size* or *learning rate*
  - If 'alpha' is too small, convergence takes a long time
  - If 'alpha' is too big, can overshoot the minimum



Choosing Alpha

- If the value of

$$\frac{|\nabla f(\mathbf{x}_i) - \nabla f(\mathbf{x}_{i+1})|}{|\mathbf{x}_i - \mathbf{x}_{i+1}|}$$

is bounded above by some number $L(\nabla f)$ then

$$\alpha \leq \frac{1}{L(\nabla f)}$$

will converge.

- For example:
  - $f(x) = x^2$
  - $L(\nabla f) = 2$
  - $\alpha = 1/2$ will be the best value

Adaptive Step Size

- Change α at each iteration
- Barzilai and Borwein, 1998
  - Suppose $\mathbf{x}_i$ is the value of $\mathbf{x}$ at the iteration i
  - $\Delta\mathbf{x} = \mathbf{x}_i - \mathbf{x}_{i-1}$
  - $\Delta g(\mathbf{x}) = \nabla f(\mathbf{x}_i) - \nabla f(\mathbf{x}_{i-1})$
  - At each step

$$\alpha = \frac{\Delta g(\mathbf{x})^T \Delta\mathbf{x}}{||\Delta g(\mathbf{x})||^2}$$

  is a good choice of α

Convergence Criteria

Choices:

- $\frac{|f(\mathbf{x})-f(\mathbf{y})|}{|f(\mathbf{x})|} < \epsilon$
- Max number of iterations
- Magnitude of gradient $|\nabla f| < \epsilon$

Gradient Ascent

- To maximize f, we can minimize $-f$
- Still use almost the same algorithm
  - Just replace

$$\mathbf{x} = \mathbf{x} - \alpha \nabla f(\mathbf{x})$$

with

$$\mathbf{x} = \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

Some Examples

- Examples

What Can Go Wrong

- Where do you think gradient descent fails?

Example

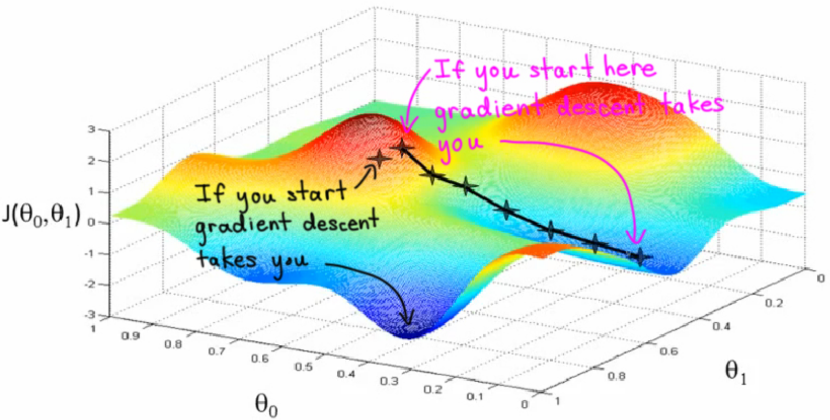


Figure 2:Non-convex function

More Bad Things

Gradient descent has limitations.

- Need differentiable and convex cost/likelihood function
- Only finds local extrema
- Poor performance without feature scaling

Back to Logistic Regression

Trying to maximize the log-likelihood function

$$\ell(\beta) = \sum_i y_i \log g(\beta^T \mathbf{x}_i) + (1 - y_i) \log(1 - g(\beta^T \mathbf{x}_i))$$

To use gradient ascent: need to compute $\nabla \ell(\beta)$

More Logistic Regression

First, let's compute the derivative of the sigmoid function g:

$$\begin{aligned}
\frac{d}{dz}g(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{d}{dz}(1+e^{-z})^{-1} \\
&= (-1)(-e^{-z})(1+e^{-z})^{-2} \\
&= g(z)\frac{1+e^{-z}-1}{1+e^{-z}} \\
&= g(z)(1-g(z))
\end{aligned}$$

More Logistic Regression

Using this and the chain rule, and writing $g = g(\beta^T \mathbf{x}_i)$ compute $\frac{\partial \ell}{\partial \beta_i}$,

$$\begin{aligned}
\frac{\partial \ell}{\partial \beta_j} &= \frac{\partial}{\partial \beta_i} \sum_i y_i \log g(\beta^T \mathbf{x}_i) + (1 - y_i) \log(1 - g(\beta^T \mathbf{x}_i)) \\
&= \frac{\partial}{\partial \beta_j} \sum_i y_i \log g + (1 - y_i) \log(1 - g) \\
&= \sum_i y_i \frac{1}{g} g(1 - g) \mathbf{x}_i - (1 - y_i) \frac{1}{1 - g} g(1 - g) x_{ij} \\
&= \sum_i (y_i - y_i g - g + y_i g) x_{ij} \\
&= \sum_i (y_i - g(\beta^T \mathbf{x}_i)) x_{ij}
\end{aligned}$$

This is what you'll use to update the value of β in each iteration of gradient descent

Stochastic Gradient Descent

Why Not Regular Gradient Descent?

What are the problems with gradient descent?

Why Not Regular Gradient Descent?

What are the problems with gradient descent?

- Need differentiable and convex cost/likelihood function
- Only finds local extrema
- Poor performance without feature scaling

Why Not Regular Gradient Descent?

What are the problems with gradient descent?

- Need differentiable and convex cost/likelihood function
- Only finds local extrema
- Poor performance without feature scaling

- Memory constrained
  - Need to store all data in memory
- CPU constrained
  - Cost function is a function of *all* data
- What if you are getting new data continuously?

Solution

Only use a single data point, or a small subset of your data, at in each step!

Algorithm

Same as gradient descent except

- at each step compute the cost function by using **just one observation**
- For example in linear regression, instead of computing the gradient of

$$\sum_i (y_i - \beta^T \mathbf{x}_i)^2$$

randomly select some $\mathbf{x}_i, y_i$ and compute the gradient of

$$(y_i - \beta^T \mathbf{x}_i)^2$$

Properties

- Faster than *batch* (regular) Gradient Descent on average
- Prone to oscillation around an optimum
- Only requires one observation in memory at once

Variants

There are a couple variants.

- "Minibatch" SGD: use a small subset of your data instead of a single observations
- "Online" SGD: update the model by performing a gradient descent step each time a new observation is collected

Newton's Method

What Is It?

- Optimization technique similar to gradient descent
- Uses a root-finding method applied to $f'(x)$

Algorithm in One Dimension

Algorithm

- Choose initial x_0
- While $f'(x) > \epsilon$:

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

Higher Dimensions

For higher dimentions, change

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

to

$$\mathbf{y}_{i+1} = \mathbf{y}_i - H(\mathbf{y}_i)^{-1} \nabla f(\mathbf{y}_i)$$

where $H(\mathbf{a}) = \left[\frac{\partial f}{\partial x_i \partial x_j}(\mathbf{a}) \right]$ is the *Hessian* matrix, the matrix of second partial derivatives at $\mathbf{a}$

Problems

- Hessian might be singular, or computation can be slow
- Can diverge with a bad starting guess