Linear Models for Regression Tasks

Ryan Miller

Grinnell College
Statistics

Outline

1. Basic review of linear regression
2. Parameters and optimization via gradient descent

Linear Regression

Linear regression is a supervised learning approach that models a numeric outcome as a linear combination of predictors:

$$Y = w_o + w_1 X_1 + w_2 X_2 + \ldots + w_p X_p + \epsilon$$

Using matrix notation:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon$$

- In statistics, the vector of *weights*, **w**, are called the *slopes* (with with w_0 being the *intercept*)
  - In machine learning, the intercept w_0, is often called the *bias* (or the *y-axis offset*)

Grinnell College
Statistics

Parameters and Cost Functions

For linear regression (and many other methods) the data are used to estimate $\mathbf{w}$. This is done using a **cost function**:

$$Cost = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

This function, *squared error cost*, expresses how close the model's predictions are to their corresponding observed values. In matrix notation:

$$Cost = \frac{1}{n}(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$$

Optimizing the Weights

- The optimal set of weights are those that minimize the cost function
  - As you might expect, we can use calculus to help us perform this minimization
- Before jumping in, let's first do some algebraic rearrangement:

$$
\begin{aligned}
Cost &= \frac{1}{n}(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) \\
&= \frac{1}{n}\mathbf{y}^T\mathbf{y} + \frac{1}{n}(-2\mathbf{y}^T\mathbf{X}\hat{\mathbf{w}} + (\mathbf{X}\hat{\mathbf{w}})^T\mathbf{X}\hat{\mathbf{w}}) \\
&= \frac{1}{n}\mathbf{y}^T\mathbf{y} + \frac{1}{n}(-2\mathbf{y}^T\mathbf{X}\hat{\mathbf{w}} + \hat{\mathbf{w}}^T\mathbf{X}^T\mathbf{X}\hat{\mathbf{w}})
\end{aligned}
$$

Optimizing the Weights

- In calculus, the **gradient** is the vector of partial derivatives with respect to each unknown variable in a function
  - For linear regression, these unknowns are the model's weights (coefficients)
- While it might not be immediately obvious, we can solve for a closed-form expression that minimizes the squared error cost function (the least squares solution)

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

- For educational purposes, we will ignore this closed-form solution and explore alternative methods for estimating an optimal set of weights

Gradient Descent

- The derivative is the slope of a function at a particular location, so we can use the gradient to gradually move towards the minimum of any (convex) cost function
  - The **gradient descent algorithm** works to minimize the cost function using sequential updates:

$$\hat{\mathbf{w}}^{(j)} = \hat{\mathbf{w}}^{(j-1)} - \alpha \frac{\partial \text{Cost}}{\partial \mathbf{w}}\left(\hat{\mathbf{w}}^{(j-1)}\right)$$

- α is a tuning parameter that controls the *learning rate*, or how quickly to update the weight vector at each iteration
  - A small α requires many iterations for the algorithm to converge (reach the minimum)
  - A large α can overshoot the minimum, which can also cause convergence issues

Some Math

Recall that we can express the linear regression cost function as:

$$Cost = \frac{1}{n}\mathbf{y}^T\mathbf{y} + \frac{1}{n}(-2\mathbf{y}^T\mathbf{X}\hat{\mathbf{w}} + \hat{\mathbf{w}}^T\mathbf{X}^T\mathbf{X}\hat{\mathbf{w}})$$

Thus, the gradient is:

$$Gradient = \frac{-2}{n}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$$

And our gradient descent updates look like:

$$\hat{\mathbf{w}}^{(j)} = \hat{\mathbf{w}}^{(j-1)} + \frac{2}{n}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}^{(j-1)})$$

Illustration

- To illustrate gradient descent, let's look at a very simple special case of linear regression involving no bias term (intercept) and a single weight parameter:

$$Y = 2.5X_1 + \epsilon$$

- For this model, the squared error cost function is:

$$Cost = \frac{1}{n}(\mathbf{y} - \mathbf{x}_1^T \hat{w}_1)^T(\mathbf{y} - \mathbf{x}_1^T \hat{w}_1)$$

Learning Rates

The graphs below illustrate 10 iterations of gradient descent for our simple, one-parameter regression example (starting at $w_1^{(0)} = 0$):

Generally, gradient descent algorithms are programmed to end once the estimated parameters change by no more than an acceptable tolerance (a small predetermined constant)

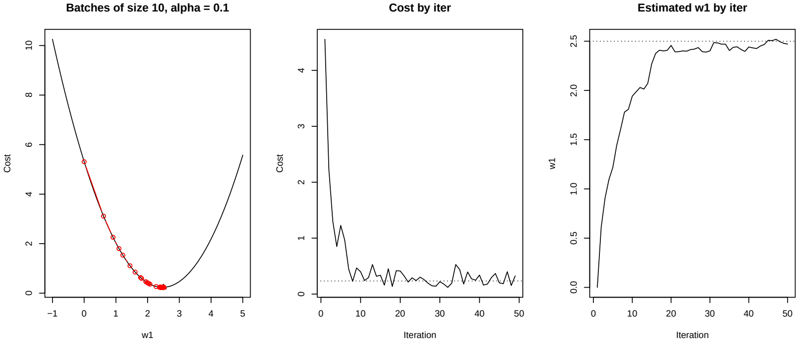
Grinnell College
Statistics

Stochastic Gradient Descent

- In our simple example, computing the gradient at each iteration required two vector-product calculations: $\mathbf{y}^T\mathbf{x}_1$ and $\mathbf{x}_1^T\mathbf{x}_1$
  - Fortunately, these can both be computed ahead of time (rather than at each iteration) which makes algorithm very computationally efficient
- For other models, the parameter vector is involved in vector-product calculations within the gradient, so these vector-product calculations must be redone at each iteration
  - In big-data settings, this computational challenge (among others) has led to the popularity of **stochastic gradient descent**

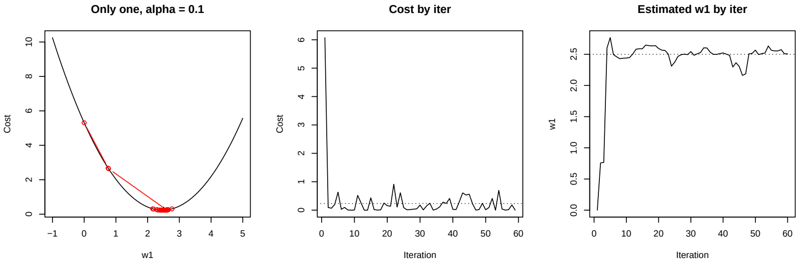
Stochastic Gradient Descent

Stochastic Gradient Descent uses the same framework as gradient descent (updating parameters using the gradient to improve the cost function) but it does so using a *subset of training data* (or even just one data-point) at each iteration:



Stochastic Gradient Descent

Even using *only one* data-point each iteration, stochastic gradient descent converges to the optimal value of w_1 (at least on average):



- This can be a huge computational benefit in big-data settings for models with complex gradients
- The algorithm's "noisy" behavior can help avoid local minima

Conclusion

This presentation briefly introduced *linear regression*, a modeling framework I'm assuming you're already familiar with:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon$$

Compared with other models we've discussed:

- Linear regression involves a structured set of parameters that must be learned from the data
  - Gradient descent (or stochastic gradient descent) can be used to learn these parameters