Linear Models for Regression Tasks

Ryan Miller

Outline

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

Linear Regression

Linear regression is a supervised learning approach that assumes the dependence of a numeric outcome on a set of predictors is linear:

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

Using matrix notation:

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

- In statistics, the vector of *weights*, $\mathbf{w}$, 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*)

Parameters and Cost Functions

For linear regression, $\mathbf{w}$, is estimated from the data. This is done using a *cost function*:

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

This function measures 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 multivariate 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 minimizer of our 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 pursue alternative methods for estimating an optimal set of weights

Gradient Descent

- A derivative describes 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** suggests we can find the set of parameters that minimize our 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
  - A large α can overshoot the minimum, potentially causing similar 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:

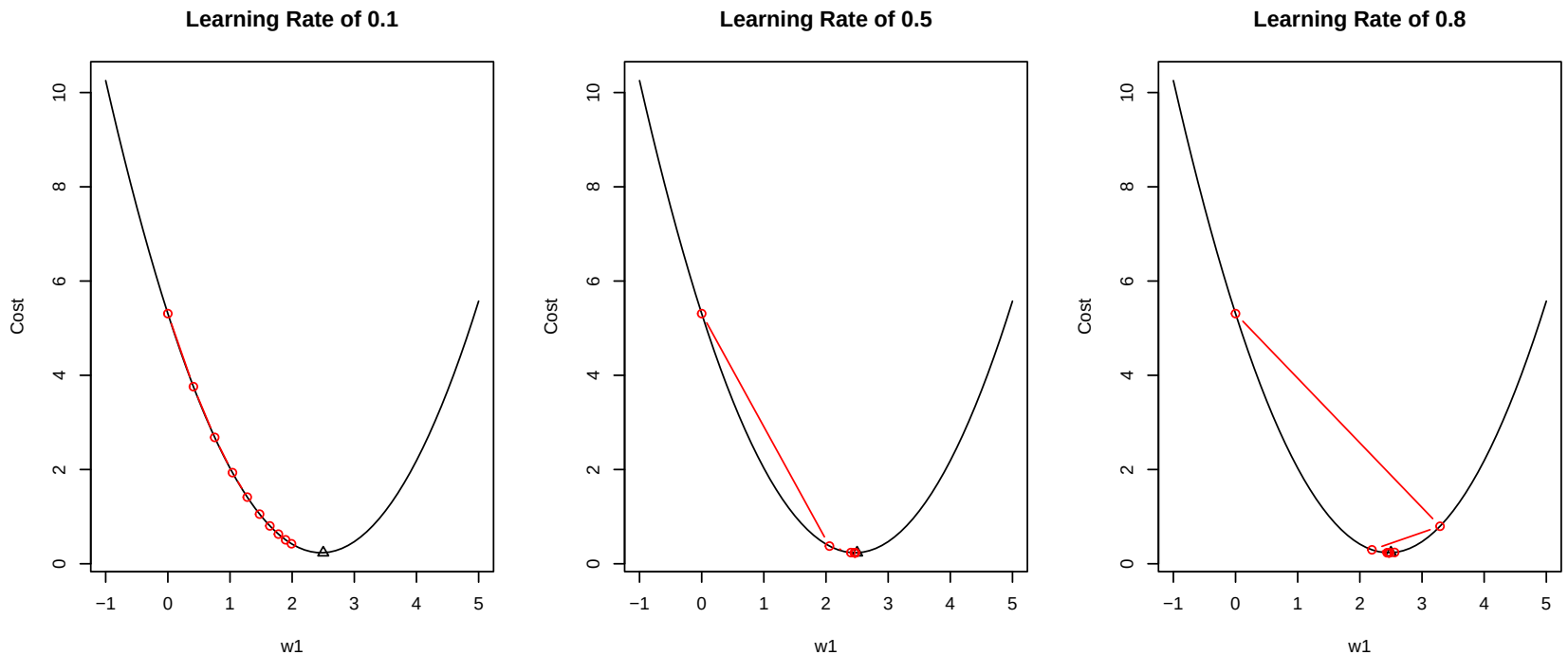
$$Y = 2.5X_1 + \epsilon$$

- For this model, the cost function looks like:

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

Learning Rates

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



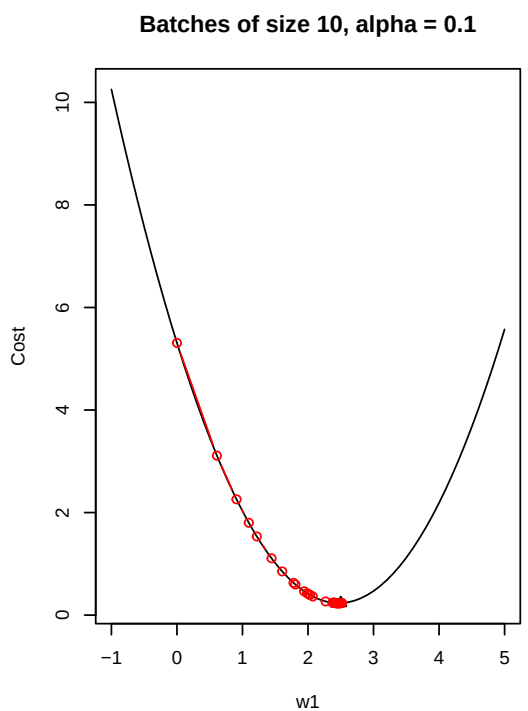
Generally, gradient descent algorithms are programmed to end when the estimated parameters change by no more than an acceptable tolerance (a predetermined small constant)

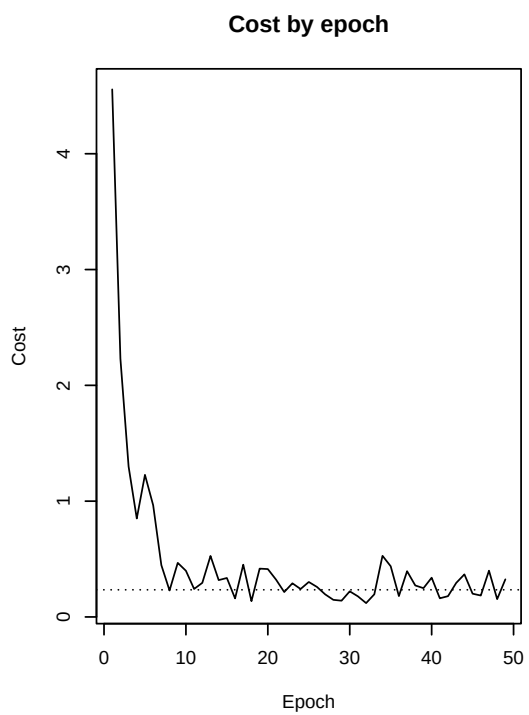
Stochastic Gradient Descent

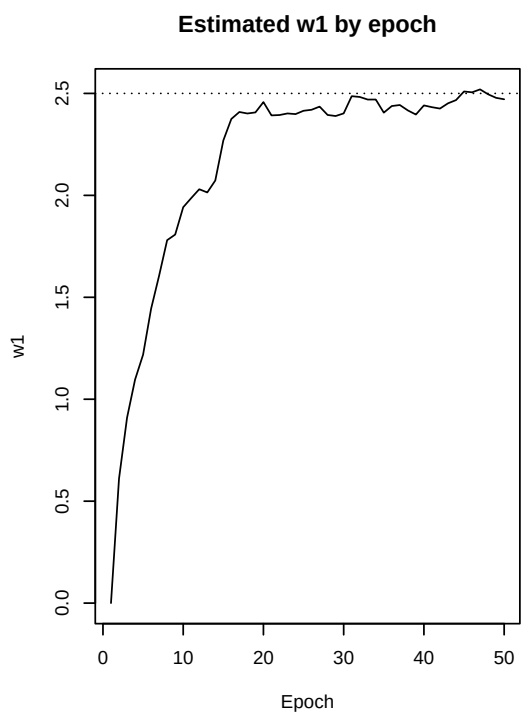
- For our simple example, computing the gradient at each epoch required two vector-product calculations: $\mathbf{y}^T\mathbf{x}_1$ and $\mathbf{x}_1^T\mathbf{x}_1$
  - Fortunately, these could be computed ahead of time (rather than at each iteration) and the algorithm is extremely computationally efficient
- For many other models, the parameter vector is involved in vector-product calculations within the gradient, so these calculations must be redone at every epoch
  - In big-data settings, this computational challenge 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 epoch:

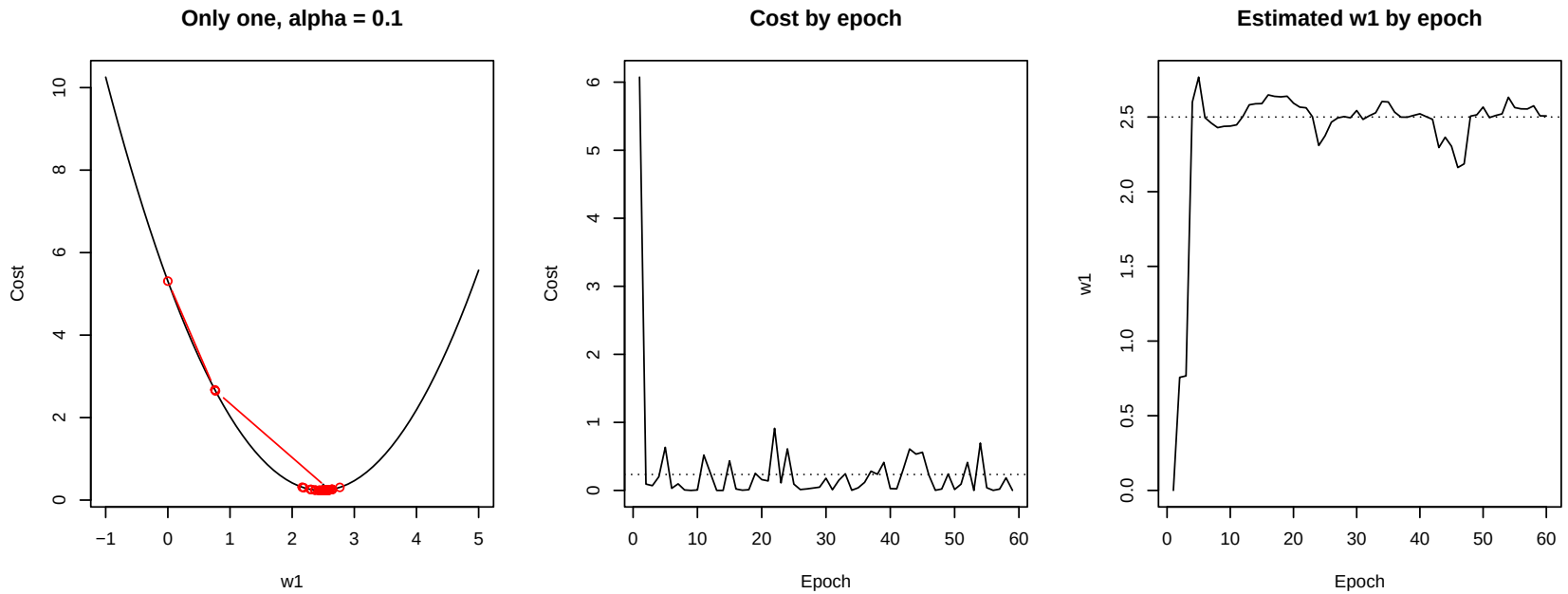






Stochastic Gradient Descent

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



- This can be a huge computational benefit in big-data settings for models with complex gradients
- An interesting additional benefit is that the algorithm's "noisy" behavior can help it avoid local minima if the cost function is not convex

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 versus k-nearest neighbors:

- Linear regression involves parameters that must be learned from the data
  - Gradient descent (or stochastic gradient descent) are methods for learning such parameters
- k-nearest neighbors is "lazy", as its only "learning" comes from storing the training data for later use
  - In contrast, linear regression only requires the learned weights to make predictions on new data