Introduction to Data Science: Solving Linear Problems

Héctor Corrada Bravo

University of Maryland, College Park, USA

2020-04-12

Solving linear problems

How to fit the type of analysis methods we've seen so far?

We will use linear regression as a case study of how this insight would work.

Solving linear problems

Case Study

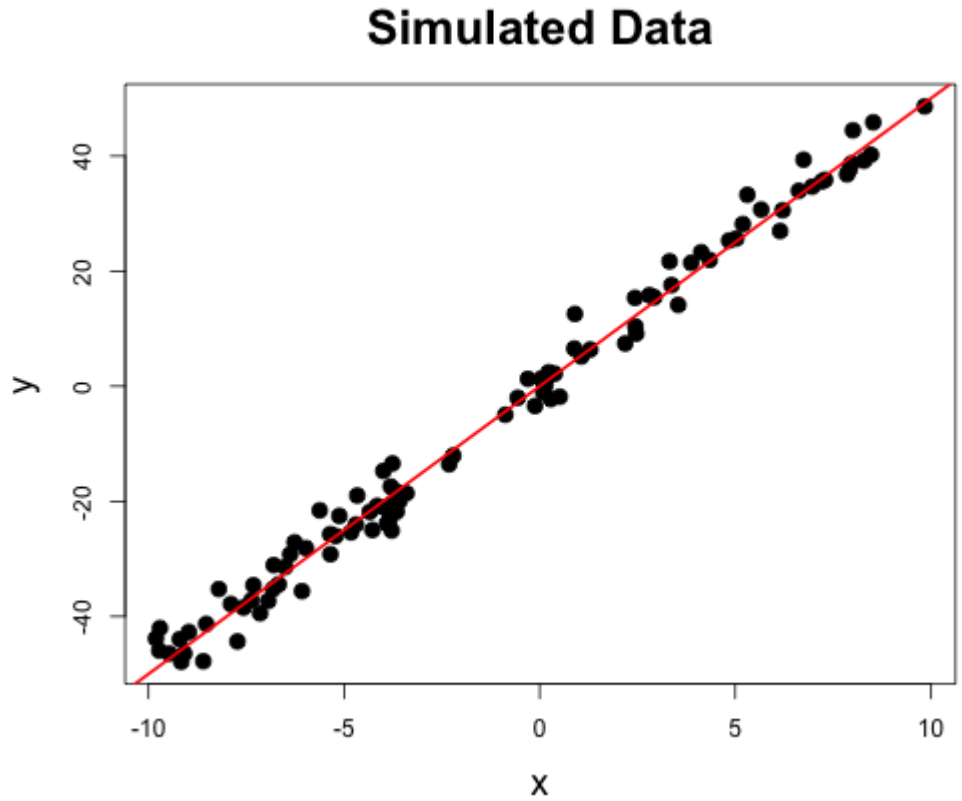
Given: Training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, with continuous response y_i and single predictor x_i for the i-th observation.

Do: Estimate parameter β_1 in model $y = \beta_1 x$ to solve

$$\min_{\beta_1} L(\beta_1) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_1 x_i)^2$$

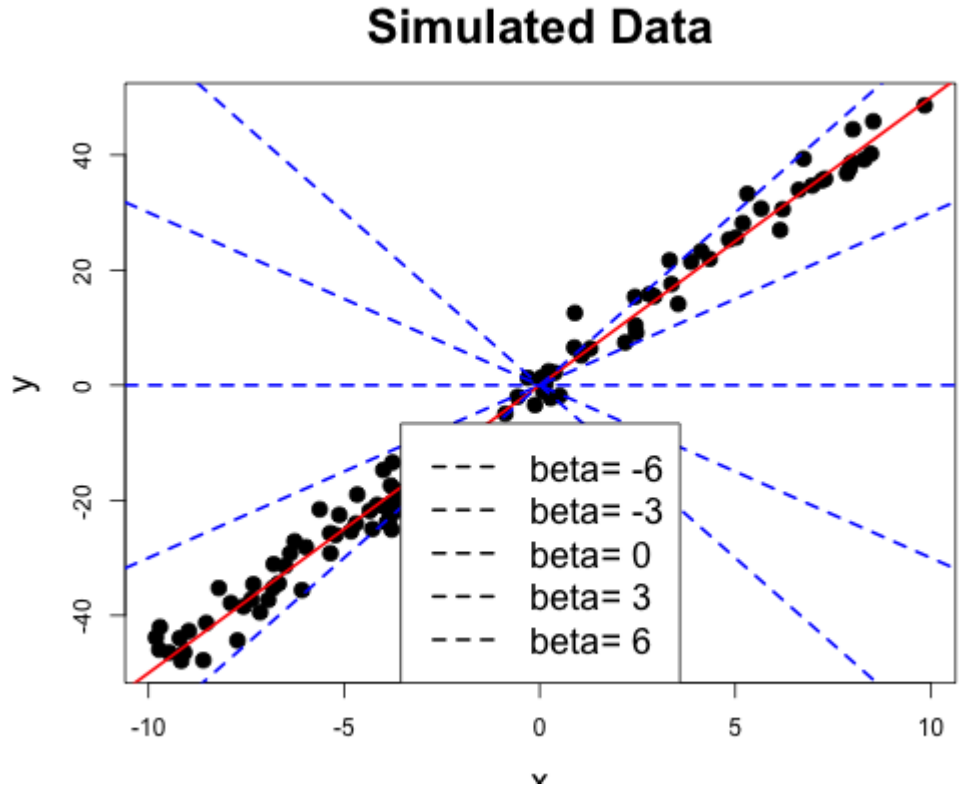
Solving linear problems

And suppose we want to fit this model to the following (simulated) data:



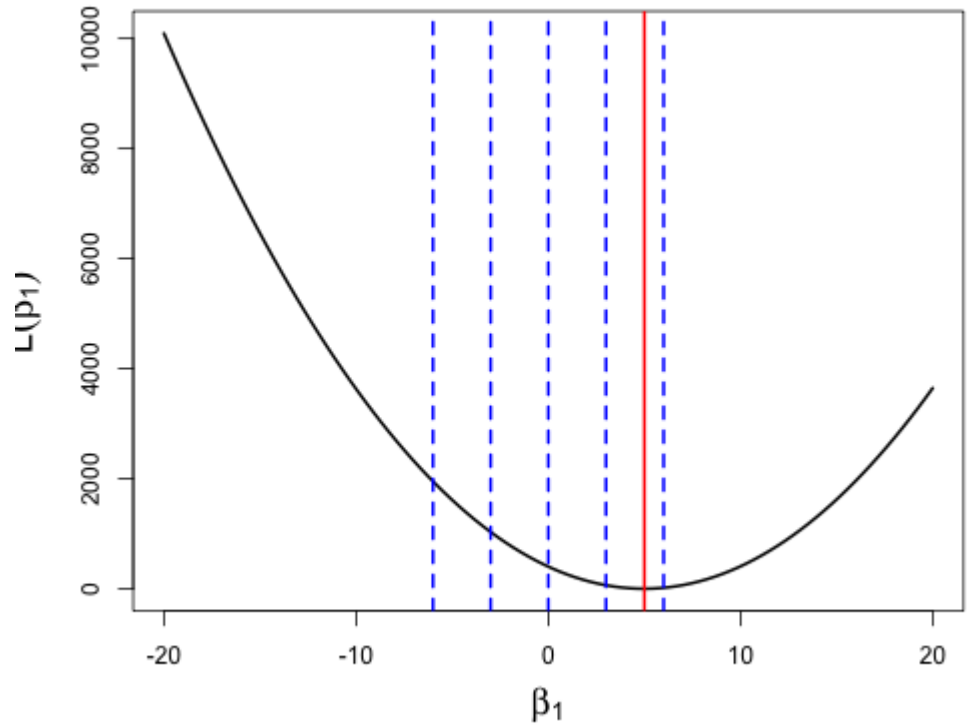
Solving linear problems

Our goal is then to find the value of β_1 that minimizes mean squared error. This corresponds to finding one of these many possible lines:



Solving linear problems

Each of which has a specific error for this dataset:



Solving linear problems

1) As we saw before in class, loss is minimized when the derivative of the loss function is 0

2) and, the derivative of the loss (with respect to β_1) at a given estimate β_1 suggests new values of β_1 with smaller loss!

Solving linear problems

Let's take a look at the derivative:

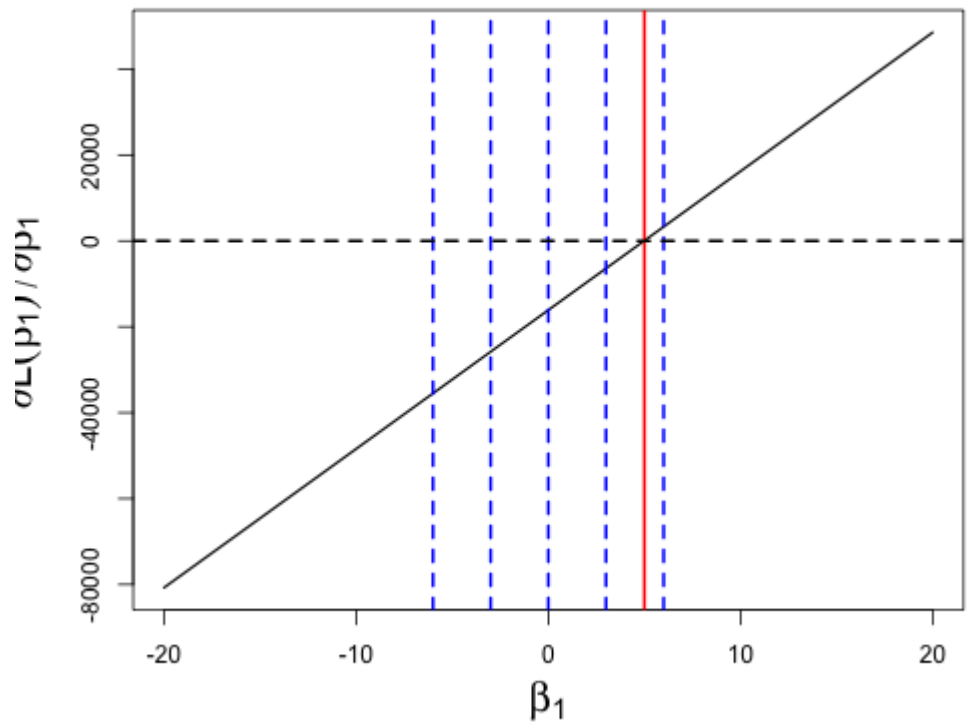
$$\frac{\partial}{\partial \beta_1} L(\beta_1) = \frac{\partial}{\partial \beta_1} \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_1 x_i)^2$$

$$= \sum_{i=1}^{n} (y_i - \beta_1 x_i) \frac{\partial}{\partial \beta_1} (y_i - \beta_1 x_i)$$

$$= \sum_{i=1}^{n} (y_i - \beta_1 x_i)(-x_i)$$

Solving linear problems

and plot it for our case study data:



Solving linear problems

Gradient Descent

This plot suggests an algorithm:

1. Initialize $\beta_1^0 = 0$
2. Repeat for $k = 1, 2, \ldots$ until convergence
    - Set $\beta_1^k = \beta_1^{k-1} + \alpha \sum_{i=1}^{n} (y_i - f(x_i; \beta_1^{k-1}))x_i$

Note: $f(x_i; \beta_1) = \beta_1 x_i$

Solving linear problems

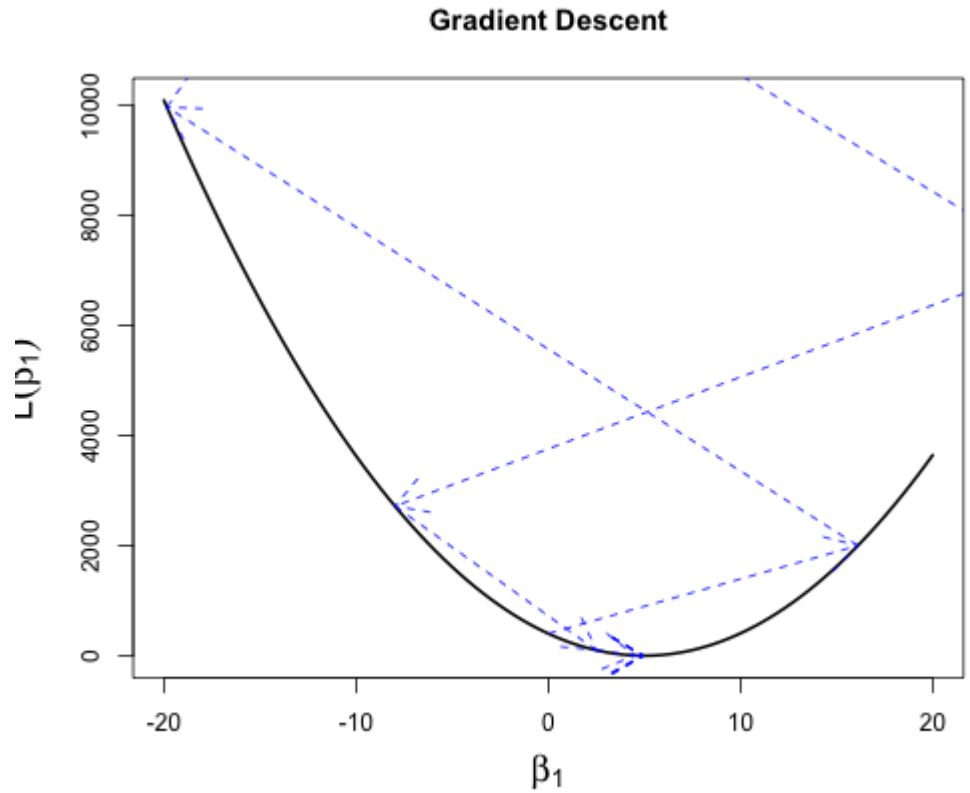
This algorithm is called **gradient descent** in the general case.

The basic idea is to move the current estimate of β_1 in the direction that minimizes loss the *fastest*.

Another way of calling this algorithm is **Steepest Descent**.

Solving linear problems

Let's run this algorithm and track what it does:



Solving linear problems

This algorithm is referred to as "Batch" gradient descent,

we take a step (update β_1) by calculating derivative with respect to *all* n observations in our dataset.

Solving linear problems

For clarity, let's write out the update equation again:

$$\beta_1^k = \beta_1^{k-1} + \alpha \sum_{i=1}^{n} (y_i - f(x_i; \beta_1^{k-1}))x_i$$

where $f(x_i; \beta_1) = \beta_1 x_i$.

Solving linear problems

For multiple predictors (e.g., adding an intercept), this generalizes to the *gradient* i.e., the vector of first derivatives of *loss* with respect to parameters.

In this case, the model sets

$$\begin{aligned} f(\mathbf{x}_i; \beta) &= \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} \\ &= \sum_{j=0}^{p} \beta_j x_{ij} \\ &= \beta' x \end{aligned}$$

Note: we take $x_{i0} = 1$

Solving linear problems

The gradient given by partial derivatives for each parameter

$$\nabla_\beta L(\beta) = \begin{bmatrix} \frac{\partial L(\beta)}{\partial \beta_0} \\ \frac{\partial L(\beta)}{\partial \beta_1} \\ \vdots \\ \frac{\partial L(\beta)}{\partial \beta_p} \end{bmatrix}$$

Solving linear problems

The update equation is exactly the same for least squares regression

$$\beta^k = \beta^{k-1} + \alpha \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \beta^{k-1}))\mathbf{x}_i$$

where $f(\mathbf{x}_i; \beta) = \beta' \mathbf{x}_i$

Note: $x_{i0} = 1$

Solving linear problems

Gradient descent falls within a family of optimization methods called *first-order methods*

These methods have properties amenable to use with very large datasets:

1. Inexpensive updates
2. "Stochastic" version can converge with few sweeps of the data
3. "Stochastic" version easily extended to streams
4. Easily parallelizable

Drawback: Can take many steps before converging

Solving linear problems

Logistic Regression

Gradient descent is also used to solve the logistic regression problem.
The same procedure follows

(1) define a loss function;
(2) derive the update equation;
(3) run the iterative gradient descent algorithm.

Solving linear problems

Let's take a look at the first two steps in this case.

For logistic regression, we turn to *maximum likelihood* to formulate a loss function.

For the logistic regression problem we are given dataset $\{\langle \mathbf{x}_1, y_1 \rangle, \ldots, \langle \mathbf{x}_n, y_n \rangle\}$, where outcomes $y_i \in \{0, 1\}$ since we are learning a binary classification problem.

Solving linear problems

The goal is to estimate parameters β in model

$$\begin{aligned} \log \frac{p(Y=1|\mathbf{X}=\mathbf{x})}{1-p(Y=1|\mathbf{X}=\mathbf{x})} &= \beta_0 + \beta_0 x_1 + \cdots + \beta_p x_p \\ &= \beta' \mathbf{x} \end{aligned}$$

Note: $x_{i0} = 1$

Solving linear problems

To establish a loss function we first assume a model for data generation. The assumption we make here is if an entity has attribute values $\mathbf{x}$, then the outcome $Y = 1$ with probability given by

$$p(\mathbf{x}; \beta) = \frac{e^{f(\mathbf{x};\beta)}}{1 + e^{f(\mathbf{x};\beta)}}$$

Note that we use the same notation $f(\mathbf{x}; \beta) = \beta' \mathbf{x}$ as we did in linear regression.

Solving linear problems

Now, we can ask, what is the probability of the data we observe for entity i under this model? We can write this probability in this form:

$$p(\mathbf{x}_i; \beta)^{y_i} (1 - p(\mathbf{x}_i; \beta))^{(1-y_i)}$$

Solving linear problems

Now, we can put these together for all observed entities since we assume that these are generated independently to get a *likelihood* function:

$$\mathcal{L}(\beta) = \prod_{i=1}^{n} p_i(\mathbf{x}_i; \beta)^{y_i} (1 - p_i(\mathbf{x}_i; \beta))^{(1-y_i)}$$

Solving linear problems

Now, we need to turn this into a loss function we can *minimize*.

The likelihood function we wrote down is one we would *maximize*.

Also, it is usually more convenient to work with the logarithm of likelihoods.

Solving linear problems

The loss function we use for gradient descent is the *negative log likelihood*

$$L(\beta) = \sum_{i=1}^{n} -y_i f(\mathbf{x}_i; \beta) + \log(1 + e^{f(\mathbf{x}_i;\beta)})$$

Solving linear problems

So, now that we have a loss function, we need to derive it's gradient to use the gradient descent algorithm. Check the lecture notes.

$$\nabla_\beta L(\beta) = \sum_{i=1}^{n} (p(\mathbf{x}_i; \beta) - y_i)\mathbf{x}_i$$

Solving linear problems

Note the nice similarity to the gradient for linear regression.

It multiplies each data (expanded) data vector $\mathbf{x}_i$ by the difference between a prediction, in this case the probability that the outcome $y_i = 1$ and the observed outcome y_i.

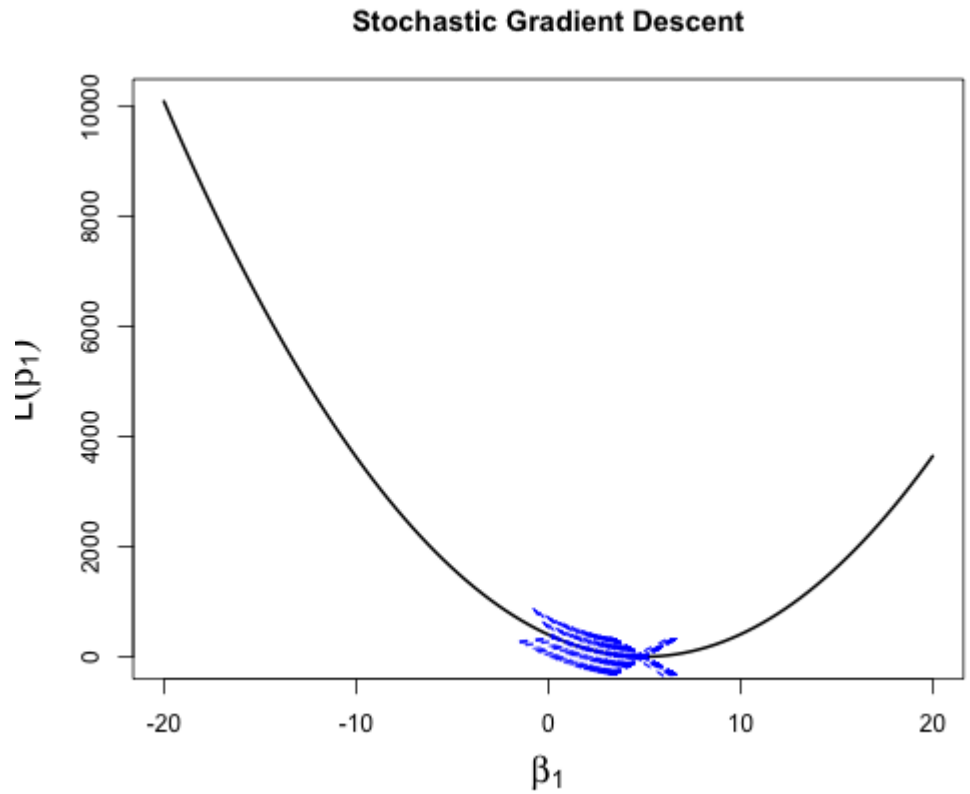
Stochastic gradient descent

Key Idea: Update parameters using update equation *one observation at a time*:

1. Initialize $\beta = \mathbf{0}$, $i = 1$
2. Repeat until convergence
   - For $i = 1$ to n
   - Set $\beta = \beta + \alpha(y_i - f(\mathbf{x}_i, \beta))\mathbf{x}_i$

Stochastic gradient descent

Let's run this and see what it does:



Stochastic gradient descent

The stochastic gradient descent algorithm can easily adapt to *data streams* where we receive observations one at a time and *assume* they are not stored.

This setting falls in the general category of *online* learning.

Stochastic gradient descent

Parallelizing gradient descent

Gradient descent algorithms are easily parallelizable:

- Split observations across computing units
- For each step, compute partial sum for each partition (map), compute final update (reduce)

Stochastic gradient descent

$$\beta^k = \beta^{k-1} + \alpha * \sum_{\text{partition } P} \sum_{i \in P} (y_i - f(\mathbf{x_i}; \beta^{k-1}))\mathbf{x}_i$$

Stochastic gradient descent

This observation has resulted in their implementation if systems for large-scale learning:

1. Vowpal Wabbit
   - Implements general framework of (sparse) stochastic gradient descent for many optimization problems
   - R interface: [http://cran.r-project.org/web/packages/RVowpalWabbit/index.html]

Stochastic gradient descent

This observation has resulted in their implementation if systems for large-scale learning:

1. Spark MLlib
    - Implements many learning algorithms using Spark framework we saw previously
    - Some access to the MLlib API via R, but built on primitives accessible through `SparkR` library we saw previously