How to Solve Linear Regression Using Linear Algebra

by **Jason Brownlee** on March 5, 2018 in **Linear Algebra**

Linear regression is a method for modeling the relationship between one or more independent variables and a dependent variable.

It is a staple of statistics and is often considered a good introductory machine learning method. It is also a method that can be reformulated using matrix notation and solved using matrix operations.

In this tutorial, you will discover the matrix formulation of linear regression and how to solve it using direct and matrix factorization methods.

After completing this tutorial, you will know:

- Linear regression and the matrix reformulation with the normal equations.
- How to solve linear regression using a QR matrix decomposition.
- How to solve linear regression using SVD and the pseudoinverse.

Let’s get started.

How to Solve Linear Regression Using Linear Algebra
Photo by likeaduck, some rights reserved.

Tutorial Overview

This tutorial is divided into 6 parts; they are:

1. Linear Regression
2. Matrix Formulation of Linear Regression
3. Linear Regression Dataset
4. Solve Directly
5. Solve via QR Decomposition
6. Solve via Singular-Value Decomposition

Linear Regression

Linear regression is a method for modeling the relationship between two scalar values: the input variable x and the output variable y.

The model assumes that y is a linear function or a weighted sum of the input variable.

```
y = f(x)
```

Or, stated with the coefficients.

```
y = b0 + b1 . x1
```

The model can also be used to model an output variable given multiple input variables called multivariate linear regression (below, brackets were added for readability).

```
y = b0 + (b1 . x1) + (b2 . x2) + ...
```

The objective of creating a linear regression model is to find the values for the coefficient values (b) that minimize the error in the prediction of the output variable y.

Matrix Formulation of Linear Regression

Linear regression can be stated using Matrix notation; for example:

```
y = X . b
```

Or, without the dot notation.

```
y = Xb
```

Where X is the input data and each column is a data feature, b is a vector of coefficients and y is a vector of output variables for each row in X.

```
      x11, x12, x13
X = (x21, x22, x23)
      x31, x32, x33
      x41, x42, x43

      b1
b = (b2)
      b3

      y1
y = (y2)
      y3
      y4
```

Reformulated, the problem becomes a system of linear equations where the b vector values are unknown. This type of system is referred to as overdetermined because there are more equations than there are unknowns, i.e. each coefficient is used on each row of data.

It is a challenging problem to solve analytically because there are multiple inconsistent solutions, e.g. multiple possible values for the coefficients. Further, all solutions will have some error because there is no line that will pass nearly through all points, therefore the approach to solving the equations must be able to handle that.

The way this is typically achieved is by finding a solution where the values for b in the model minimize the squared error. This is called linear least squares.

```
||X . b - y||^2 = sum i=1 to m ( sum j=1 to n Xij . bj - yi)^2
```

This formulation has a unique solution as long as the input columns are independent (e.g. uncorrelated).

> *We cannot always get the error e = b – Ax down to zero. When e is zero, x is an exact solution to Ax = b. When the length of e is as small as possible, xhat is a least squares solution.*

— Page 219, Introduction to Linear Algebra, Fifth Edition, 2016.

In matrix notation, this problem is formulated using the so-named normal equation:

```
X^T . X . b = X^T . y
```

This can be re-arranged in order to specify the solution for b as:

```
b = (X^T . X)^-1 . X^T . y
```

This can be solved directly, although given the presence of the matrix inverse can be numerically challenging or unstable.

Linear Regression Dataset

In order to explore the matrix formulation of linear regression, let's first define a dataset as a context.

We will use a simple 2D dataset where the data is easy to visualize as a scatter plot and models are easy to visualize as a line that attempts to fit the data points.

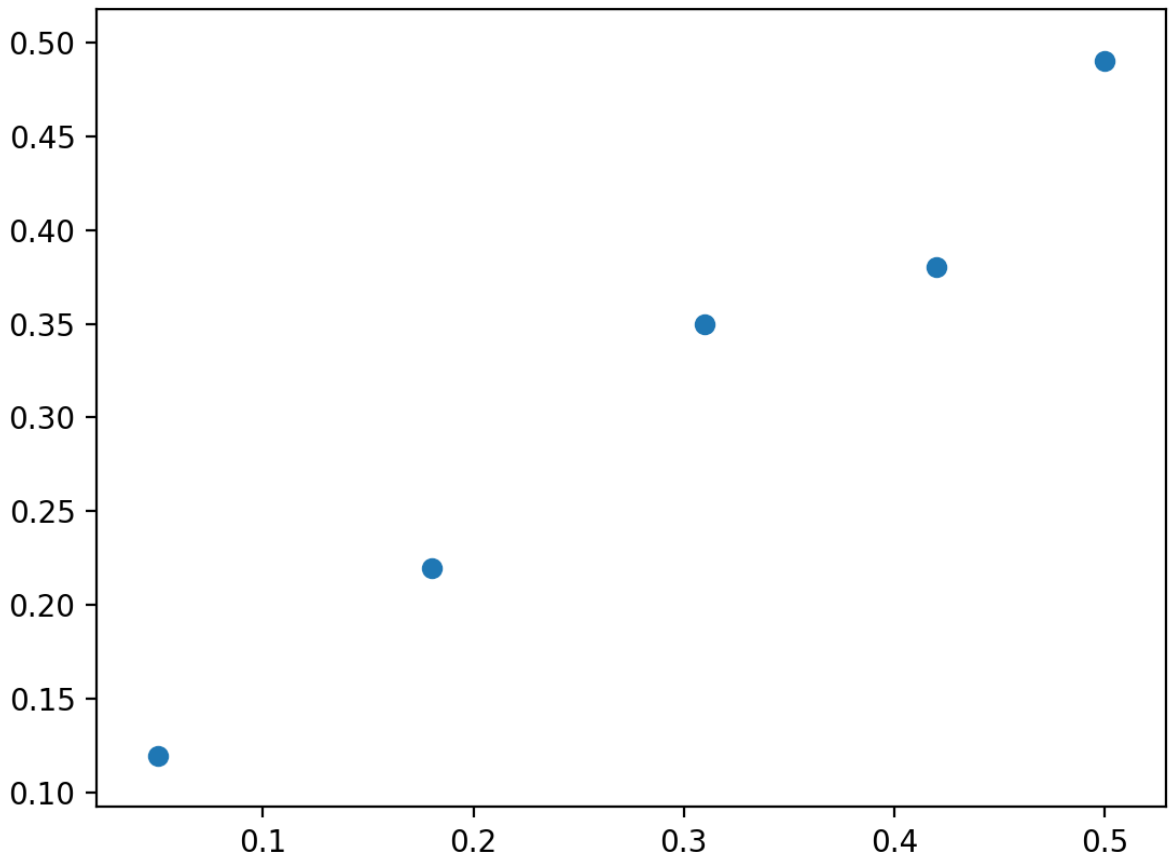
The example below defines a 5×2 matrix dataset, splits it into X and y components, and plots the dataset as a scatter plot.

```
from numpy import array
from matplotlib import pyplot
data = array([
    [0.05, 0.12],
    [0.18, 0.22],
    [0.31, 0.35],
    [0.42, 0.38],
    [0.5, 0.49],
    ])
print(data)
X, y = data[:,0], data[:,1]
X = X.reshape((len(X), 1))
# plot dataset
pyplot.scatter(X, y)
pyplot.show()
```

Running the example first prints the defined dataset.

```
[[ 0.05  0.12]
 [ 0.18  0.22]
 [ 0.31  0.35]
 [ 0.42  0.38]
 [ 0.5   0.49]]
```

A scatter plot of the dataset is then created showing that a straight line cannot fit this data exactly.



Scatter Plot of Linear Regression Dataset

Solve Directly

The first approach is to attempt to solve the regression problem directly.

That is, given X, what are the set of coefficients b that when multiplied by X will give y. As we saw in a previous section, the normal equations define how to calculate b directly.

```
b = (X^T . X)^-1 . X^T . y
```

This can be calculated directly in NumPy using the inv() function for calculating the matrix inverse.

```
b = inv(X.T.dot(X)).dot(X.T).dot(y)
```

Once the coefficients are calculated, we can use them to predict outcomes given X.

```
yhat = X.dot(b)
```

Putting this together with the dataset defined in the previous section, the complete example is listed below.

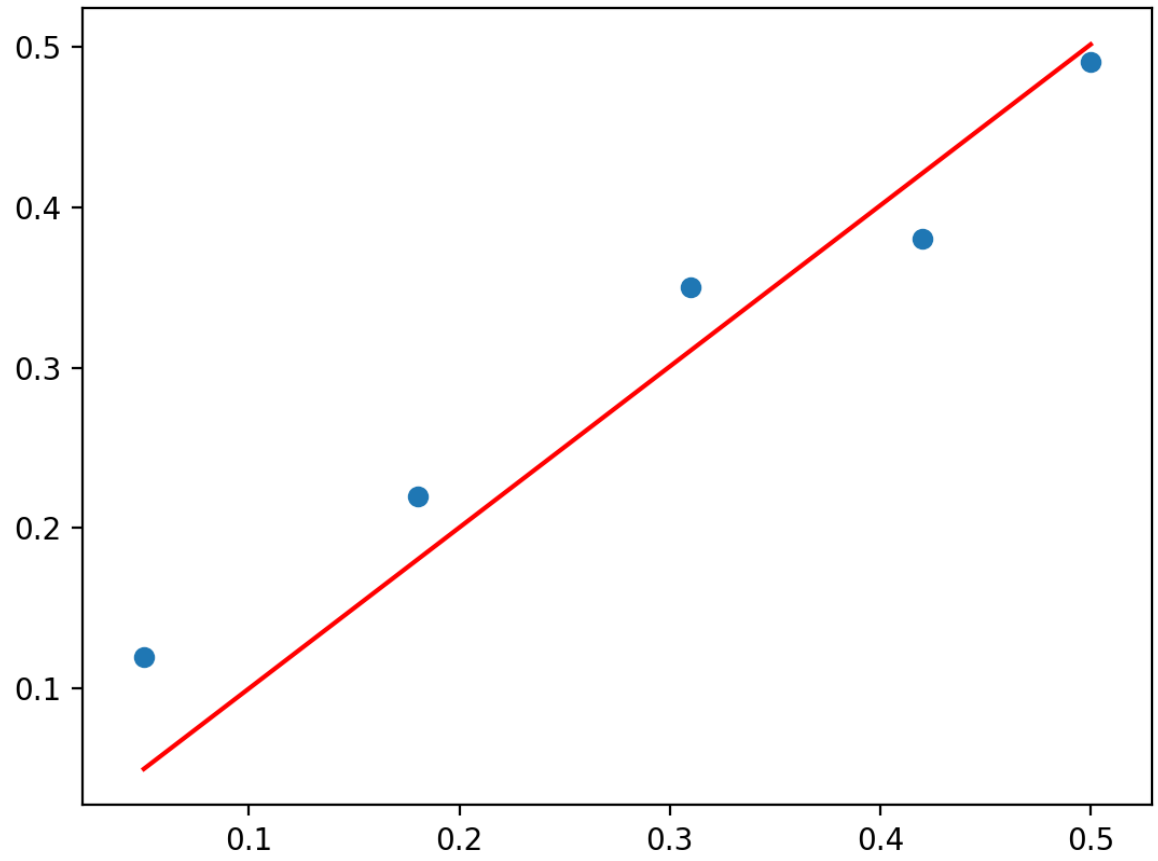
```
# solve directly
from numpy import array
from numpy.linalg import inv
from matplotlib import pyplot
data = array([
    [0.05, 0.12],
```

```
    [0.18, 0.22],
    [0.31, 0.35],
    [0.42, 0.38],
    [0.5, 0.49],
    ])
X, y = data[:,0], data[:,1]
X = X.reshape((len(X), 1))
# linear least squares
b = inv(X.T.dot(X)).dot(X.T).dot(y)
print(b)
# predict using coefficients
yhat = X.dot(b)
# plot data and predictions
pyplot.scatter(X, y)
pyplot.plot(X, yhat, color='red')
pyplot.show()
```

Running the example performs the calculation and prints the coefficient vector b.

```
[ 1.00233226]
```

A scatter plot of the dataset is then created with a line plot for the model, showing a reasonable fit to the data.



Scatter Plot of Direct Solution to the Linear Regression Problem

A problem with this approach is the matrix inverse that is both computationally expensive and numerically unstable. An alternative approach is to use a matrix decomposition to avoid this operation. We will look at two examples in the following sections.

Solve via QR Decomposition

The QR decomposition is an approach of breaking a matrix down into its constituent elements.

```
A = Q . R
```

Where A is the matrix that we wish to decompose, Q a matrix with the size m x m, and R is an upper triangle matrix with the size m x n.

The QR decomposition is a popular approach for solving the linear least squares equation.

Stepping over all of the derivation, the coefficients can be found using the Q and R elements as follows:

```
b = R^-1 . Q.T . y
```

The approach still involves a matrix inversion, but in this case only on the simpler R matrix.

The QR decomposition can be found using the qr() function in NumPy. The calculation of the coefficients in NumPy looks as follows:

```
# QR decomposition
Q, R = qr(X)
b = inv(R).dot(Q.T).dot(y)
```

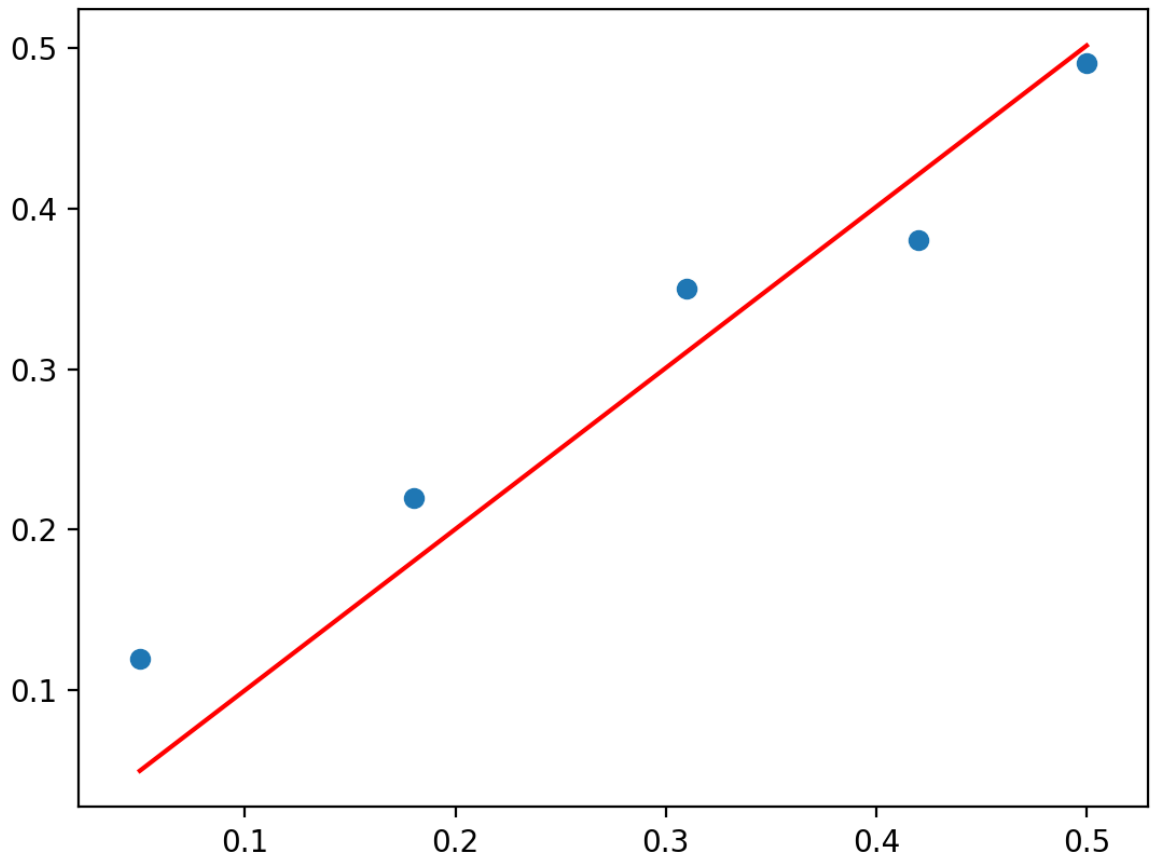
Tying this together with the dataset, the complete example is listed below.

```
# least squares via QR decomposition
from numpy import array
from numpy.linalg import inv
from numpy.linalg import qr
from matplotlib import pyplot
data = array([
[0.05, 0.12],
[0.18, 0.22],
[0.31, 0.35],
[0.42, 0.38],
[0.5, 0.49],
])
X, y = data[:,0], data[:,1]
X = X.reshape((len(X), 1))
# QR decomposition
Q, R = qr(X)
b = inv(R).dot(Q.T).dot(y)
print(b)
# predict using coefficients
yhat = X.dot(b)
# plot data and predictions
pyplot.scatter(X, y)
pyplot.plot(X, yhat, color='red')
pyplot.show()
```

Running the example first prints the coefficient solution and plots the data with the model.

```
[ 1.00233226]
```

The QR decomposition approach is more computationally efficient and more numerically stable than calculating the normal equation directly, but does not work for all data matrices.



Scatter Plot of QR Decomposition Solution to the Linear Regression Problem

Solve via Singular-Value Decomposition

The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method like the QR decomposition.

```
1 X = U . Sigma . V^*
```

Where A is the real n x m matrix that we wish to decompose, U is a m x m matrix, Sigma (often represented by the uppercase Greek letter Sigma) is an m x n diagonal matrix, and V^* is the conjugate transpose of an n x n matrix where * is a superscript.

Unlike the QR decomposition, all matrices have an SVD decomposition. As a basis for solving the system of linear equations for linear regression, SVD is more stable and the preferred approach.

Once decomposed, the coefficients can be found by calculating the pseudoinverse of the input matrix X and multiplying that by the output vector y.

```
1 b = X^+ . y
```

Where the pseudoinverse is calculated as following:

```
1 X^+ = U . D^+ . V^T
```

Where X^+ is the pseudoinverse of X and the + is a superscript, D^+ is the pseudoinverse of the diagonal matrix Sigma and V^T is the transpose of V^*.

> *Matrix inversion is not defined for matrices that are not square. [...] When A has more columns than rows, then solving a linear equation using the pseudoinverse provides one of the many possible solutions.*

— Page 46, Deep Learning, 2016.

We can get U and V from the SVD operation. D^+ can be calculated by creating a diagonal matrix from Sigma and calculating the reciprocal of each non-zero element in Sigma.

```
          s11,   0,   0
Sigma = (  0, s22,   0)
           0,   0, s33

      1/s11,     0,     0
D = (    0, 1/s22,     0)
           0,     0, 1/s33
```

We can calculate the SVD, then the pseudoinverse manually. Instead, NumPy provides the function pinv() that we can use directly.

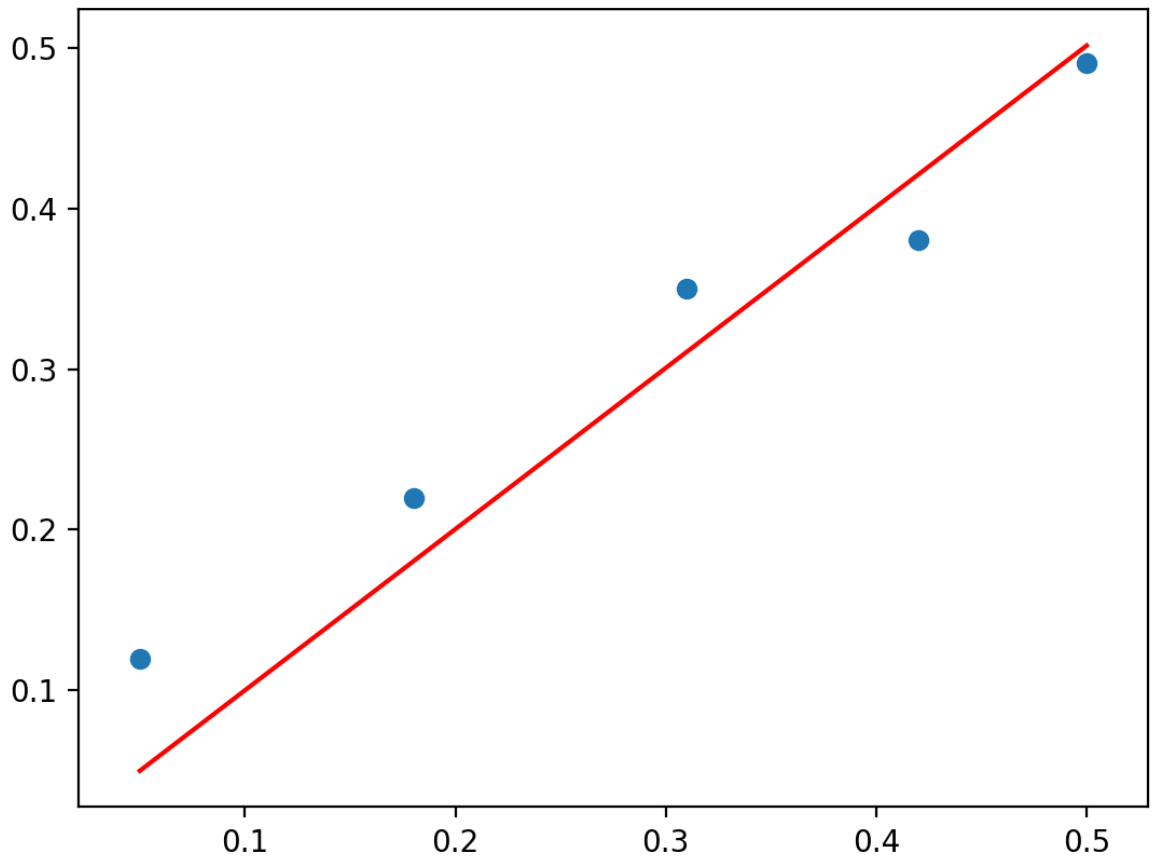
The complete example is listed below.

```
# least squares via SVD with pseudoinverse
from numpy import array
from numpy.linalg import pinv
from matplotlib import pyplot
data = array([
    [0.05, 0.12],
    [0.18, 0.22],
    [0.31, 0.35],
    [0.42, 0.38],
    [0.5, 0.49],
    ])
X, y = data[:,0], data[:,1]
X = X.reshape((len(X), 1))
# calculate coefficients
b = pinv(X).dot(y)
print(b)
# predict using coefficients
yhat = X.dot(b)
# plot data and predictions
pyplot.scatter(X, y)
pyplot.plot(X, yhat, color='red')
pyplot.show()
```

Running the example prints the coefficient and plots the data with a red line showing the predictions from the model.

```
[ 1.00233226]
```

In fact, NumPy provides a function to replace these two steps in the lstsq() function that you can use directly.



Scatter Plot of SVD Solution to the Linear Regression Problem

Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Implement linear regression using the built-in lstsq() NumPy function
- Test each linear regression on your own small contrived dataset.
- Load a tabular dataset and test each linear regression method and compare the results.

If you explore any of these extensions, I'd love to know.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Books

- Section 7.7 Least squares approximate solutions. No Bullshit Guide To Linear Algebra, 2017.
- Section 4.3 Least Squares Approximations, Introduction to Linear Algebra, Fifth Edition, 2016.
- Lecture 11, Least Squares Problems, Numerical Linear Algebra, 1997.
- Chapter 5, Orthogonalization and Least Squares, Matrix Computations, 2012.
- Chapter 12, Singular-Value and Jordan Decompositions, Linear Algebra and Matrix Analysis for Statistics, 2014.

- Section 2.9 The Moore-Penrose Pseudoinverse, Deep Learning, 2016.
- Section 15.4 General Linear Least Squares, Numerical Recipes: The Art of Scientific Computing, Third Edition, 2007.

API

- numpy.linalg.inv() API
- numpy.linalg.qr() API
- numpy.linalg.svd() API
- numpy.diag() API
- numpy.linalg.pinv() API
- numpy.linalg.lstsq() API

Articles

- Linear regression on Wikipedia
- Least squares on Wikipedia
- Linear least squares (mathematics) on Wikipedia
- Overdetermined system on Wikipedia
- QR decomposition on Wikipedia
- Singular-value decomposition on Wikipedia
- Moore–Penrose inverse

Tutorials

- The Linear Algebra View of Least-Squares Regression
  Linear Algebra with Python and NumPy

Summary

In this tutorial, you discovered the matrix formulation of linear regression and how to solve it using direct and matrix factorization methods.

Specifically, you learned:

- Linear regression and the matrix reformulation with the normal equations.
- How to solve linear regression using a QR matrix decomposition.
- How to solve linear regression using SVD and the pseudoinverse.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ How to Calculate the Principal Component Analysis from Scratch in Python No Bullshit Guide To Linear Algebra Review ›

4 Responses to *How to Solve Linear Regression Using Linear Algebra*

John Menzies March 9, 2018 at 10:05 pm #

REPLY

In your introduction you refer to the univariate problem, y=b0+b1*x, or Y=X.b in matrix notation. It is clear that b is a 2×1 matrix, so X has to be a nx2 matrix.
However, in your implementation of the various methods of solution you implicitly assume that b0 = 0 and then X is a nx1 matrix and b a 1×1 matrix.
For the example dataset you have chosen, this leads to a plausible solution of b1=1.00233, with a sum of squared deviations of 0.00979 and the fitted line looks more or less OK. But it is definitely not a least squares solution for the data set.
If you fit for b0 as well, you get a slope of b1= 0.78715 and b0=0.08215, with the sum of squared deviations of 0.00186. To do this, the X matrix has to be augmented with a column of ones.
If the data set had been
data = array([

[5.05, 0.12],
[5.18, 0.22],
[5.31, 0.35],
[5.42, 0.38],
[5.5, 0.49],
])
then fitting for b0 and b1 would give b1=0.78715 as before but using your formalism b1 becomes 0.05963 with sums of squared deviations of 0.00186 as before for the 2-parameter fit, but 0.07130 in your case.

Your presentation is generally quite clear, but I think it is misleading nevertheless in suggesting that it leads to a least squares solution.

Jason Brownlee March 10, 2018 at 6:28 am #

REPLY

Thanks John.

Alex November 10, 2018 at 9:00 am #

REPLY

Hi Jason,

Thank you very much for this great and very helpful article!!!
I wonder how this method can work for quadratic curve fitting, such as parabola?

Jason Brownlee November 11, 2018 at 5:56 am #

REPLY

You can use a linear model with inputs raised to exponents, e.g, x^2, x^3, etc. E.g. polynomial regression.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

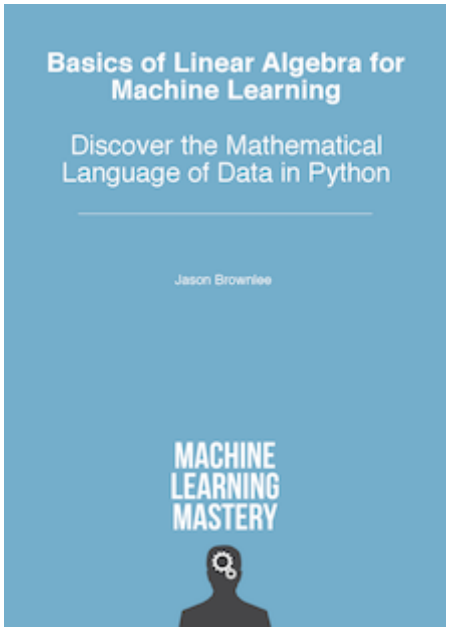
Welcome to Machine Learning Mastery!

Hi, I'm Jason Brownlee, PhD
I write tutorials to help developers (*like you*) get results with machine learning.

Read More

Linear Algebra for Machine Learning
Understand linear algebra by writing code in Python.

Click to Get Started Now!



POPULAR

So, You are Working on a Machine Learning Problem…
APRIL 4, 2018

How to Make Predictions with Keras
APRIL 9, 2018

11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)
AUGUST 6, 2018

How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption
OCTOBER 10, 2018

You're Doing it Wrong. Why Machine Learning Does Not Have to Be So Hard
MARCH 28, 2018

How to Make Predictions with scikit-learn
APRIL 6, 2018

How to Develop LSTM Models for Time Series Forecasting
NOVEMBER 14, 2018

You might also like…

- How to Install Python for Machine Learning
- Your First Machine Learning Project in Python
- Your First Neural Network in Python
- Your First Classifier in Weka
- Your First Time Series Forecasting Project