Machine Learning Algorithms From Scratch

With Python

Jason Brownlee

MACHINE LEARNING MASTERY

Machine Learning Algorithms From Scratch

Edition: v1.0

This is Just a Sample

Thank-you for your interest in **Machine Learning Algorithms From Scratch**.
This is just a sample of the full text. You can purchase the complete book online from:
https://machinelearningmastery.com/machine-learning-algorithms-from-scratch/

Contents

Welcome

Welcome to Machine Learning Algorithms From Scratch. This is your guide to learning the details of machine learning algorithms by implementing them from scratch in Python. You will discover how to load data, evaluate models and implement a suite of top machine learning algorithms using step-by-step tutorials and sample code.

Machine learning algorithms do have a lot of math and theory under the covers, but you do not need to know why algorithms work to be able to implement then and apply them to achieve real and valuable results. From an applied perspective, machine learning is a shallow field and a motivated developer can quickly pick it up and start making very real and impactful contributions. This is my goal for you and this book is your ticket to that outcome.

Implement Machine Learning Algorithms

Most developers that I know (myself included) learn best by implementing. It is our preferred learning style and it is the reason that I created this book. This section lists the benefits of implementing machine learning algorithms from scratch, some benefits from extending your own implementations as well as some limitations to this approach to learning.

Implementation Benefits

Machine learning is all about algorithms and there are so many algorithms that it can feel overwhelming. Really, there are probably only 10 algorithms that if understood will unlock the field for you. An approach that you can use to get a handle on machine learning algorithms is to implement them from scratch. This will give you a deep understanding of how the algorithm works and all of the micro-decision points within the method that can be parameterized or modified to tune it to a specific problem. The benefits of implementing algorithms from scratch are:

- **Understanding**: You will gain a deep appreciation for how the algorithm works. You begin to understand how the mathematical description of the method relates to vectors and matrices of numbers that you code operates on. You will also know how all of the parameters are used, their effects and even have insights into how it could be further parameterized to specialize it for a problem.
- **Starting Point**: Your implementation will provide the basis for more advanced extensions and even an operational system that uses the algorithm. Your deep knowledge of the algorithm and your implementation can give you advantages of knowing the space and time complexity of your own code over using an opaque off-the-shelf library.

- **Ownership**: The implementation is your own giving you confidence with the method and ownership over how it is realized as a system. It is no longer just a machine learning algorithm, but a method that is now in your toolbox.

Implementation Extensions

Once you have implemented an algorithm you can explore making improvements to the implementation. Some examples of improvements you could explore include:

- **Experimentation**: You can expose many of the micro-decisions you made in the algorithms implementation as parameters and perform studies on variations of those parameters. This can lead to new insights and disambiguation of algorithm implementations that you can share and promote.
- **Optimization**: You can explore opportunities to make the implementation more efficient by using tools, libraries, different languages, different data structures, patterns and internal algorithms. Knowledge you have of algorithms and data structures for classical computer science can be very beneficial in this type of work.
- **Specialization**: You may explore ways of making the algorithm more specific to a problem. This can be required when creating production systems and is a valuable skill. Making an algorithm more problem specific can also lead to increases in efficiency (such as running time) and efficacy (such as accuracy or other performance measures).
- **Generalization**: Opportunities can be created by making a specific algorithm more general. Programmers (like mathematicians) are uniquely skilled in abstraction and you may be able to see how the algorithm could be applied to more general cases of a class of problem or other problems entirely.

Implementation Limitations

Developers and engineers often learn best by implementing, but implementing machine learning algorithms is not the place to start for everyone.

- **Slow for Beginners**. Often, practitioners will make more progress and progress faster by learning how to apply machine learning algorithms to predictive modeling problems. Implementing algorithms is a second step for learning how to get more out of each algorithm by discovering how they work and how the parameters affect the algorithm's behavior.
- **Speed and Correctness**. Algorithms that you use to solve business problems need to be fast and correct. And this can be very hard to do for beginners. The implementations developed for learning purposes are almost certainly going to be too slow or too fragile for use in operations (that includes all examples in this book). Use implementations for learning and efficient code libraries for production systems.

Book Organization

This book is divided into 6 main parts:

1. **Introduction**. Welcomes you to the book and clearly lays out what to expect and your learning outcomes (you are here).
2. **Data Preparation**. Tutorials for loading and preparing data, evaluating model predictions, estimating model skill and developing a baseline for model performance.
3. **Linear Algorithms**. Tutorials on linear machine learning algorithms such as linear regression, multivariate linear regression, logistic regression and the Perceptron algorithm.
4. **Nonlinear Algorithms**. Tutorials on nonlinear machine learning algorithms such as Naive Bayes, k-Nearest Neighbors, Learning Vector Quantization, Back-propagation and Decision Trees.
5. **Ensemble Algorithms**. Tutorials on ensemble machine learning algorithms such as Bootstrap Aggregation, Random Forest and Stacked Generalization.
6. **Conclusions**. A review of how far you have come and resources for getting help and further reading.

There are a few ways you can read this book. You can dip into the tutorials as your need or interests motivate you. Alternatively, you can work through the book end-to-end and take advantage of how the tutorials build in complexity and range. I recommend the latter approach. To get the very most from this book, I recommend taking each tutorial and build upon them. Attempt to improve the results, apply the method to a similar but different problem, and so on. I share a number of extension ideas for each tutorial for you to consider in each tutorial.

Write up what you tried or learned and share it on your blog, social media or send me an email at `jason@MachineLearningMastery.com`. This book is really what you make of it and by putting in a little extra, you can quickly become a true force in machine learning algorithms.

Tutorial Structure

A tutorial-based approach is used throughout the book. It is conversational rather than formal and focuses on the ideas, the code needed to implement those ideas, and the results to expect. All tutorials in this book follow a carefully designed 6-part structure. This structure can be summarized as follows:

1. Overview.
2. Description.
3. Tutorial.
4. Case Study.
5. Extensions.
6. Review.

Overview

This is a short section that summarizes the tutorial. You will discover exactly what you will know after completing the tutorial.

Description

This section describes both the technique and problem that you will be applying it to. It will not describe the theory behind why the technique works. Instead, it focuses on the salient details for how the technique works. This includes points relevant to implementing it from scratch.

Also included in this section is a summary of the problem that the technique will be evaluated on, if relevant. Not all tutorials will have a problem description, only those tutorials on a given machine learning algorithm. Only standard well known machine learning problems are used as they are freely accessible and known best results are available for comparison.

Tutorial

The tutorial are the steps to complete to go from just an idea to a fully working implementation. Each tutorial was designed with three principles in mind:

1. **Procedural**: Tutorials are procedural, meaning that they are presented as a recipe of discrete steps intended to be completed in order.

2. **Standalone**: Tutorials are standalone, meaning that all code needed to run the example is available within the tutorial, even if this involves repetition.

3. **Consistent**: Tutorials are consistent, meaning that the same routines developed earlier in the book are used again and again to load data and evaluate algorithms.

A summary of the steps in the tutorial is provided followed by the numbered steps of each part of the procedure. Code was designed to be modular and broken down into many small functions that can be understood and tested as standalone units. A procedural rather than object-oriented approach was used in all code examples. Clever Python tricks and advanced use of lambdas and list comprehensions were kept to a minimum in favor of `for` loops. This was done intentionally for 3 reasons:

- To be kind to Python novices.
- To be understandable as almost pseudocode, a large benefit of Python.
- To be readily adaptable for use in other other languages and environments.

If you have advanced Python skills you can see more efficient ways to structure the code, please share your ideas with me and we can put them on the ML Mastery blog. I would love to see what you come up with.

Disclaimer: All the code in this book is for education and demonstration purposes only. It is not intended for use in production systems or operational environments.

Case Study

Each algorithm tutorial ends with a complete code listing of a fully working case study on a real world predictive modeling problem. This is to show you how to use the technique in practice, often leveraging techniques introduced earlier in the book such as data loading, data preparation and algorithm evaluation. This is to ensure that even if small copy-paste errors were made during the execution of the tutorial or steps were skipped, that you always have a reference version of the tutorial to run and use as a template for your own work. A sample output is also provided from executing the example, again so that you have reference for comparison.

Extensions

This section lists ideas to extend the example in the tutorial. This may include additional implementation concerns to make the technique more robust or generally applicable. It may also include usage heuristics for the technique to ensure you can get more out of its application to new problems. Please try some of the extension ideas. Even email me and share your experiences, we can put them to the ML Mastery blog.

Review

Tutorials end with a summary of the principles and skills that you learned. This helps to reinforce and remind you of your progress through the book and keep you highly motivated.

Requirements For This Book

Python

You do not need to be a Python expert, but it would be helpful if you have or know how to install and setup a Python environment. You are expected to know some basic Python syntax. If you are a programmer from another language like Java or C#, there is a Python crash course in the appendix to bring you up to speed quickly.

The tutorials assume that you have a working Python 2.7 installed and working. I have not tested the code with a Python 3 environment, but my students tell me that the code works with little or no modification. You may have a Python environment on your workstation or laptop, it may be in a VM or a Docker instance that you run, or it may be a server instance that you can configure in the cloud.

Machine Learning

You do not need to be a machine learning expert, but it would be helpful if you knew how to navigate a small machine learning problem. Concepts and techniques are described at the beginning of each tutorial, but only briefly enough to give you context. Additional resources are listed for you to learn more on each concept introduced. There are resources to go into these topics in more detail at the end of the book, but some knowledge of these areas might make things easier for you.

Your Outcomes From Reading This Book

This book will lead you from being a developer who is interested in machine learning algorithms to a machine learning developer that knows how to implement a suite of machine learning algorithms and techniques from scratch in Python. Specifically, you will know:

- How to load from CSV files and prepare data for modeling.
- How to select algorithm evaluation metrics and resampling techniques for a test harness.
- How to develop a baseline expectation of performance for a given problem.
- How to implement and apply a suite of linear machine learning algorithms.
- How to implement and apply a suite of advanced nonlinear machine learning algorithms.
- How to implement and apply ensemble machine learning algorithms to improve performance.

From this outcome you will:

- Know how top machine learning algorithms work internally.
- Know how to better configure machine learning algorithms in order to get the most out of them.
- Know the myriad of micro-decisions that a machine learning library has hidden from you in practice.
- Know how you might begin to develop your own custom machine learning algorithm implementations.

What This Book is Not

This book solves a specific problem of getting you, a developer, up to speed on how to implement top machine learning algorithms from scratch in Python. This book was not intended to be everything to everyone and it is very important to calibrate your expectations. Specifically:

- **This is not a machine learning textbook**. We will not be getting into the theory or mathematical description of machine learning algorithms as this is not required to implement algorithms from scratch. You are also expected to have some familiarity with machine learning basics, or be able to pick them up yourself.
- **This is not an applications book**. We will be using real world problems as case studies, but we will not linger on the best practices for working through machine learning problems end-to-end.
- **This is not a Python programming book**. We will not be spending a lot of time on Python syntax and programming (e.g. basic programming tasks in Python). You are expected to already be familiar with Python or a developer who can pick up a new C-like language relatively quickly.

Resources are provided in the final chapter if you are interested in focusing on one of these related areas.

Summary

It is a special time right now. The interest and information available for applied machine learning is so great. The pace of change with machine learning feels like it has never been so fast, spurred by the amazing results that the methods are showing in such a broad range of fields.

This is the start of your journey into expanding your understanding of machine learning algorithms and I am excited for you. Take your time, have fun and I'm so excited to see where you can take this amazing new technology.

Next

In the next section you will start with your first tutorial on how to load machine learning data.

Chapter 1

Simple Linear Regression

Linear regression is a prediction method that is more than 200 years old. Simple linear regression is a great first machine learning algorithm to implement as it requires you to estimate properties from your training dataset, but is simple enough for beginners to understand. In this tutorial, you will discover how to implement the simple linear regression algorithm from scratch in Python.

After completing this tutorial you will know:

- How to estimate statistical quantities from training data.
- How to estimate linear regression coefficients from data.
- How to make predictions using linear regression for new data.

Let's get started.

1.1 Description

This section is divided into two parts, a description of the simple linear regression technique and a description of the dataset to which we will later apply it.

1.1.1 Simple Linear Regression

Linear regression assumes a linear or straight line relationship between the input variables (`X`) and the single output variable (`y`). More specifically, that output (`y`) can be calculated from a linear combination of the input variables (`X`). When there is a single input variable, the method is referred to as a simple linear regression.

In simple linear regression we can use statistics on the training data to estimate the coefficients required by the model to make predictions on new data. The line for a simple linear regression model can be written as:

$$y = b0 + b1 \times x \tag{1.1}$$

Where `b0` and `b1` are the coefficients we must estimate from the training data. Once the coefficients are known, we can use this equation to estimate output values for `y` given new input

examples of `x`. It requires that you calculate statistical properties from the data such as mean, variance and covariance.

All the algebra has been taken care of and we are left with some arithmetic to implement to estimate the simple linear regression coefficients. Briefly, we can estimate the coefficients as follows:

$$B1 = \frac{\sum_{i=1}^{n}((x_i - mean(x)) \times (y_i - mean(y)))}{\sum_{i=1}^{n}(x_i - mean(x))^2}$$
$$B0 = mean(y) - B1 \times mean(x) \tag{1.2}$$

Where the `i` refers to the value of the `i`th value of the input `x` or output `y`. Don't worry if this is not clear right now, these are the functions will implement in the tutorial.

1.1.2 Swedish Auto Insurance Dataset

In this tutorial we will use the Swedish Auto Insurance Dataset. This dataset involves the prediction of total claim payments. The baseline RMSE on the problem is approximately 72.251 thousand Kronor. You can learn more about it in Appendix A, Section **??**. Download the dataset and save it into your current working directory with the filename `insurance.csv`. Note, you may need to convert the European comma (`,`) to the decimal dot (`.`). You will also need change the file from white-space-separated variables to CSV format.

1.2 Tutorial

This tutorial is broken down into five parts:

1. Calculate Mean and Variance.
2. Calculate Covariance.
3. Estimate Coefficients.
4. Make Predictions.
5. Swedish Auto Insurance Case Study.

These steps will give you the foundation you need to implement and train simple linear regression models for your own prediction problems.

1.2.1 Calculate Mean and Variance

The first step is to estimate the mean and the variance of both the input and output variables from the training data. The mean of a list of numbers can be calculated as:

$$mean(x) = \frac{\sum_{i=1} x_i}{count(x)} \tag{1.3}$$

Below is a function named `mean()` that implements this behavior for a list of numbers.

```
# Calculate the mean value of a list of numbers
def mean(values):
  return sum(values) / float(len(values))
```

Listing 1.1: Function To Calculate the Mean of a List of Numbers.

The variance is the sum squared difference for each value from the mean value. Variance for a list of numbers can be calculated as:

$$variance = \sum_{i=1}^{n}(x_i - mean(x))^2 \tag{1.4}$$

Below is a function named `variance()` that calculates the variance of a list of numbers. It requires the mean of the list to be provided as an argument, just so we don't have to calculate it more than once.

```
# Calculate the variance of a list of numbers
def variance(values, mean):
  return sum([(x-mean)**2 for x in values])
```

Listing 1.2: Function To Calculate the Variance of a List of Numbers.

We can put these two functions together and test them on a small contrived dataset. Below is a small dataset of `x` and `y` values.

```
x,  y
1,  1
2,  3
4,  3
3,  2
5,  5
```

Listing 1.3: Small Contrived Dataset For Testing.

We can plot this dataset on a scatter plot graph as follows:

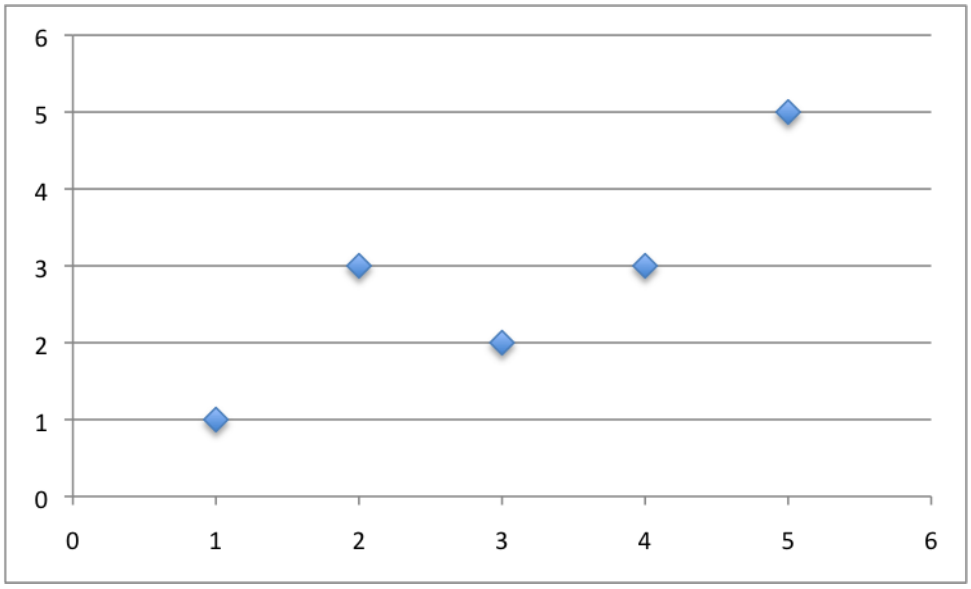

Figure 1.1: Plot of the Small Contrived Dataset.

We can calculate the mean and variance for both the x and y values in the example below.

```
# Example of Estimating Mean and Variance

# Calculate the mean value of a list of numbers
def mean(values):
  return sum(values) / float(len(values))

# Calculate the variance of a list of numbers
def variance(values, mean):
  return sum([(x-mean)**2 for x in values])

# calculate mean and variance
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
mean_x, mean_y = mean(x), mean(y)
var_x, var_y = variance(x, mean_x), variance(y, mean_y)
print('x stats: mean=%.3f variance=%.3f' % (mean_x, var_x))
print('y stats: mean=%.3f variance=%.3f' % (mean_y, var_y))
```

Listing 1.4: Example to Calculate Mean and Variance on the Contrived Dataset.

Running this example prints out the mean and variance for both columns.

```
x stats: mean=3.000 variance=10.000
y stats: mean=2.800 variance=8.800
```

Listing 1.5: Example Output of Mean and Variance on the Contrived Dataset.

This is our first step, next we need to put these values to use in calculating the covariance.

1.2.2 Calculate Covariance

The covariance of two groups of numbers describes how those numbers change together. Covariance is a generalization of correlation. Correlation describes the relationship between two groups of numbers, whereas covariance can describe the relationship between two or more groups of numbers. Additionally, covariance can be normalized to produce a correlation value. Nevertheless, we can calculate the covariance between two variables as follows:

$$covariance = \sum_{i=1}^{n}(x_i - mean(x)) \times (y_i - mean(y)) \tag{1.5}$$

Below is a function named `covariance()` that implements this statistic. It builds upon the previous step and takes the lists of `x` and `y` values as well as the mean of these values as arguments.

```
# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
  covar = 0.0
  for i in range(len(x)):
    covar += (x[i] - mean_x) * (y[i] - mean_y)
  return covar
```

Listing 1.6: Function To Calculate the Covariance.

We can test the calculation of the covariance on the same small contrived dataset as in the previous section. Putting it all together we get the example below.

```
# Example of Calculating Covariance

# Calculate the mean value of a list of numbers
def mean(values):
  return sum(values) / float(len(values))

# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
  covar = 0.0
  for i in range(len(x)):
    covar += (x[i] - mean_x) * (y[i] - mean_y)
  return covar

# calculate covariance
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
mean_x, mean_y = mean(x), mean(y)
covar = covariance(x, mean_x, y, mean_y)
print('Covariance: %.3f' % (covar))
```

Listing 1.7: Example to Calculate Covariance on the Contrived Dataset.

Running this example prints the covariance for the `x` and `y` variables.

```
Covariance: 8.000
```

Listing 1.8: Example Output of Calculating Covariance on the Contrived Dataset.

We now have all the pieces in place to calculate the coefficients for our model.

1.2.3 Estimate Coefficients

We must estimate the values for two coefficients in simple linear regression. The first is `B1` which can be estimated as:

$$B1 = \frac{\sum_{i=1}^{n}(x_i - mean(x) \times (y_i - mean(y))}{\sum_{i=1}^{n}(x_i - mean(x))^2} \tag{1.6}$$

We have learned some things above and can simplify this arithmetic to:

$$B1 = \frac{covariance(x, y)}{variance(x)} \tag{1.7}$$

We already have functions to calculate `covariance()` and `variance()`. Next, we need to estimate a value for `B0`, also called the intercept as it controls the starting point of the line where it intersects the y-axis.

$$B0 = mean(y) - B1 \times mean(x) \tag{1.8}$$

Again, we know how to estimate `B1` and we have a function to estimate `mean()`. We can put all of this together into a function named `coefficients()` that takes the dataset as an argument and returns the coefficients.

```
# Calculate coefficients
def coefficients(dataset):
  x = [row[0] for row in dataset]
  y = [row[1] for row in dataset]
  x_mean, y_mean = mean(x), mean(y)
  b1 = covariance(x, x_mean, y, y_mean) / variance(x, x_mean)
  b0 = y_mean - b1 * x_mean
  return [b0, b1]
```

Listing 1.9: Function To Calculate the Coefficients.

We can put this together with all of the functions from the previous two steps and test out the calculation of coefficients.

```
# Example of Calculating Coefficients

# Calculate the mean value of a list of numbers
def mean(values):
  return sum(values) / float(len(values))

# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
  covar = 0.0
  for i in range(len(x)):
    covar += (x[i] - mean_x) * (y[i] - mean_y)
  return covar

# Calculate the variance of a list of numbers
def variance(values, mean):
  return sum([(x-mean)**2 for x in values])

# Calculate coefficients
def coefficients(dataset):
  x = [row[0] for row in dataset]
  y = [row[1] for row in dataset]
  x_mean, y_mean = mean(x), mean(y)
  b1 = covariance(x, x_mean, y, y_mean) / variance(x, x_mean)
  b0 = y_mean - b1 * x_mean
  return [b0, b1]

# calculate coefficients
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
b0, b1 = coefficients(dataset)
print('Coefficients: B0=%.3f, B1=%.3f' % (b0, b1))
```

Listing 1.10: Example to Calculate Coefficients on the Contrived Dataset.

Running this example calculates and prints the coefficients.

```
Coefficients: B0=0.400, B1=0.800
```

Listing 1.11: Example Output of Calculating Coefficients on the Contrived Dataset.

Now that we know how to estimate the coefficients, the next step is to use them.

1.2.4 Make Predictions

The simple linear regression model is a line defined by coefficients estimated from training data. Once the coefficients are estimated, we can use them to make predictions. The equation to make predictions with a simple linear regression model is as follows:

$$y = b0 + b1 \times x \tag{1.9}$$

Below is a function named `simple_linear_regression()` that implements the prediction equation to make predictions on a test dataset. It also ties together the estimation of the coefficients on training data from the steps above. The coefficients prepared from the training data are used to make predictions on the test data, which are then returned.

```
def simple_linear_regression(train, test):
  predictions = list()
  b0, b1 = coefficients(train)
  for row in test:
    yhat = b0 + b1 * row[0]
    predictions.append(yhat)
  return predictions
```

Listing 1.12: Function To Run Simple Linear Regression.

Let's pull together everything we have learned and make predictions for our simple contrived dataset. As part of this example, we will also add in a function to manage the evaluation of the predictions called `evaluate_algorithm()` and another function to estimate the Root Mean Squared Error of the predictions called `rmse_metric()`. The full example is listed below.

```
# Example of Standalone Simple Linear Regression
from math import sqrt

# Calculate root mean squared error
def rmse_metric(actual, predicted):
  sum_error = 0.0
  for i in range(len(actual)):
    prediction_error = predicted[i] - actual[i]
    sum_error += (prediction_error ** 2)
  mean_error = sum_error / float(len(actual))
  return sqrt(mean_error)

# Evaluate regression algorithm on training dataset
def evaluate_algorithm(dataset, algorithm):
  test_set = list()
  for row in dataset:
    row_copy = list(row)
    row_copy[-1] = None
    test_set.append(row_copy)
  predicted = algorithm(dataset, test_set)
  print(predicted)
  actual = [row[-1] for row in dataset]
  rmse = rmse_metric(actual, predicted)
  return rmse

# Calculate the mean value of a list of numbers
def mean(values):
  return sum(values) / float(len(values))
```

```
# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
  covar = 0.0
  for i in range(len(x)):
    covar += (x[i] - mean_x) * (y[i] - mean_y)
  return covar

# Calculate the variance of a list of numbers
def variance(values, mean):
  return sum([(x-mean)**2 for x in values])

# Calculate coefficients
def coefficients(dataset):
  x = [row[0] for row in dataset]
  y = [row[1] for row in dataset]
  x_mean, y_mean = mean(x), mean(y)
  b1 = covariance(x, x_mean, y, y_mean) / variance(x, x_mean)
  b0 = y_mean - b1 * x_mean
  return [b0, b1]

# Simple linear regression algorithm
def simple_linear_regression(train, test):
  predictions = list()
  b0, b1 = coefficients(train)
  for row in test:
    yhat = b0 + b1 * row[0]
    predictions.append(yhat)
  return predictions

# Test simple linear regression
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
rmse = evaluate_algorithm(dataset, simple_linear_regression)
print('RMSE: %.3f' % (rmse))
```

Listing 1.13: Example of Simple Linear Regression on the Contrived Dataset.

Running this example displays the following output that first lists the predictions and the RMSE of these predictions.

```
[1.1999999999999995, 1.9999999999999996, 3.5999999999999996, 2.8, 4.399999999999999]
RMSE: 0.693
```

Listing 1.14: Example Output Simple Linear Regression on the Contrived Dataset.

Finally, we can plot the predictions as a line and compare it to the original dataset.

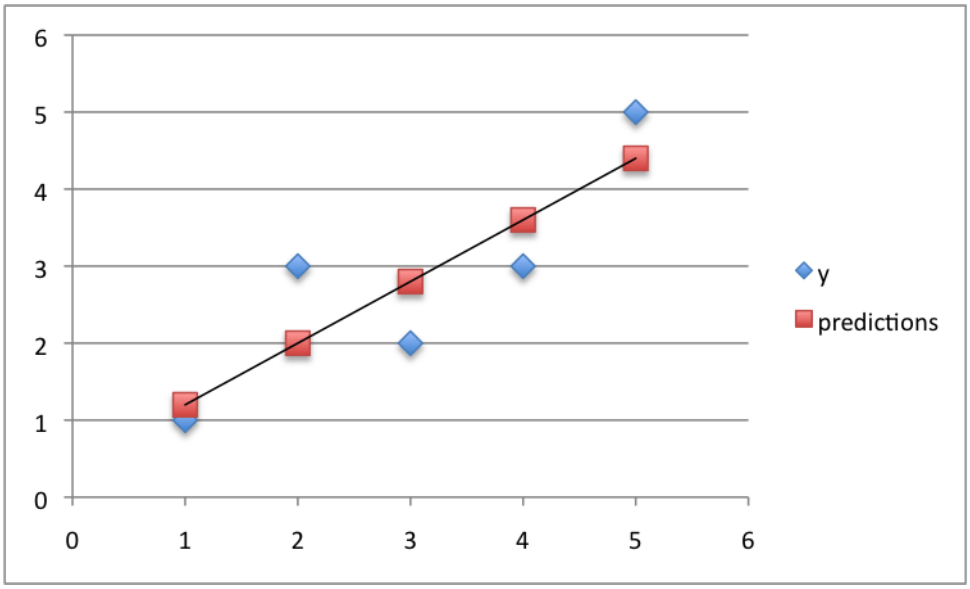

Figure 1.2: Plot of the Simple Linear Regression Predictions on the Contrived Dataset.

1.2.5 Swedish Auto Insurance Case Study

We now know how to implement a simple linear regression model. Let's apply it to the Swedish insurance dataset. This section assumes that you have downloaded the dataset to the file `insurance.csv` and it is available in the current working directory. We will add some convenience functions to the simple linear regression from the previous steps.

Specifically a function to load the CSV file called `load_csv()`, a function to convert a loaded dataset to numbers called `str_column_to_float()`, a function to evaluate an algorithm using a train and test set called `train_test_split()` a function to calculate RMSE called `rmse_metric()` and a function to evaluate an algorithm called `evaluate_algorithm()`.

The complete example is listed below. A training dataset of 60% of the data is used to prepare the model and predictions are made on the remaining 40%.

```
# Example of Simple Linear Regression on the Swedish Insurance Dataset
from random import seed
from random import randrange
from csv import reader
from math import sqrt

# Load a CSV file
def load_csv(filename):
  dataset = list()
  with open(filename, 'r') as file:
    csv_reader = reader(file)
    for row in csv_reader:
      if not row:
        continue
      dataset.append(row)
  return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
```

```
  for row in dataset:
    row[column] = float(row[column].strip())

# Split a dataset into a train and test set
def train_test_split(dataset, split):
  train = list()
  train_size = split * len(dataset)
  dataset_copy = list(dataset)
  while len(train) < train_size:
    index = randrange(len(dataset_copy))
    train.append(dataset_copy.pop(index))
  return train, dataset_copy

# Calculate root mean squared error
def rmse_metric(actual, predicted):
  sum_error = 0.0
  for i in range(len(actual)):
    prediction_error = predicted[i] - actual[i]
    sum_error += (prediction_error ** 2)
  mean_error = sum_error / float(len(actual))
  return sqrt(mean_error)

# Evaluate an algorithm using a train/test split
def evaluate_algorithm(dataset, algorithm, split, *args):
  train, test = train_test_split(dataset, split)
  test_set = list()
  for row in test:
    row_copy = list(row)
    row_copy[-1] = None
    test_set.append(row_copy)
  predicted = algorithm(train, test_set, *args)
  actual = [row[-1] for row in test]
  rmse = rmse_metric(actual, predicted)
  return rmse

# Calculate the mean value of a list of numbers
def mean(values):
  return sum(values) / float(len(values))

# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
  covar = 0.0
  for i in range(len(x)):
    covar += (x[i] - mean_x) * (y[i] - mean_y)
  return covar

# Calculate the variance of a list of numbers
def variance(values, mean):
  return sum([(x-mean)**2 for x in values])

# Calculate coefficients
def coefficients(dataset):
  x = [row[0] for row in dataset]
  y = [row[1] for row in dataset]
  x_mean, y_mean = mean(x), mean(y)
  b1 = covariance(x, x_mean, y, y_mean) / variance(x, x_mean)
```

```
  b0 = y_mean - b1 * x_mean
  return [b0, b1]

# Simple linear regression algorithm
def simple_linear_regression(train, test):
  predictions = list()
  b0, b1 = coefficients(train)
  for row in test:
    yhat = b0 + b1 * row[0]
    predictions.append(yhat)
  return predictions

# Simple linear regression on insurance dataset
seed(1)
# load and prepare data
filename = 'insurance.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])):
  str_column_to_float(dataset, i)
# evaluate algorithm
split = 0.6
rmse = evaluate_algorithm(dataset, simple_linear_regression, split)
print('RMSE: %.3f' % (rmse))
```

Listing 1.15: Example of Simple Linear Regression on the Insurance Dataset.

Running the algorithm prints the RMSE for the trained model on the training dataset. A score of about 38 (thousands of Kronor) was achieved, which is much better than the baseline performance of 72 (thousands of Kronor) on the same problem.

```
RMSE: 38.339
```

Listing 1.16: Example Output Simple Linear Regression on the Insurance Dataset.

1.3 Extensions

The best extension to this tutorial is to try out the algorithm on more problems. Small datasets with just an input (`x`) and output (`y`) columns are popular for demonstration in statistical books and courses. Many of these datasets are available online. Seek out some more small datasets and make predictions using simple linear regression.

1.4 Review

In this tutorial, you discovered how to implement the simple linear regression algorithm from scratch in Python. Specifically, you learned:

- How to estimate statistics from a training dataset like mean, variance and covariance.
- How to estimate model coefficients and use them to make predictions.
- How to use simple linear regression to make predictions on a real dataset.

1.4.1 Further Reading

- Section 3.1 Simple Linear Regression, page 61, *An Introduction to Statistical Learning*, 2014.
  http://amzn.to/2eeTyQX
- Section 18.6. Regression and Classification with Linear Models, page 717, *Artificial Intelligence: A Modern Approach*, 2010.
  http://amzn.to/2e3lFqP

1.4.2 Next

In the next tutorial, you will discover how to implement and apply the multivariate linear regression algorithm.

This is Just a Sample

Thank-you for your interest in **Machine Learning Algorithms From Scratch**.
This is just a sample of the full text. You can purchase the complete book online from:
https://machinelearningmastery.com/machine-learning-algorithms-from-scratch/