Chapter 2: Scale Machine Learning Data

Many machine learning algorithms expect data to be scaled consistently. There are two popular methods that you should consider when scaling your data for machine learning. In this tutorial, you will discover how you can rescale your data for machine learning. After reading this tutorial you will know:

- How to normalize your data from scratch.
- How to standardize your data from scratch.
- When to normalize as opposed to standardize data.

Let's get started.

2.1 Description

Many machine learning algorithms expect the scale of the input and even the output data to be equivalent. It can help in methods that weight inputs in order to make a prediction, such as in linear regression and logistic regression. It is practically required in methods that combine weighted inputs in complex ways such as in artificial neural networks and deep learning.

2.1.1 Pima Indians Diabetes Dataset

In this tutorial we will use the Pima Indians Diabetes Dataset. This dataset involves the predic- tion of the onset of diabetes within 5 years. The baseline performance on the problem is approx- imately 65%. You can learn more about it in Appendix A, Section A.4. Download the dataset and save it into your current working directory with the filename pima-indians-diabetes.csv.

2.2 Tutorial

This tutorial is divided into 3 parts:

1. Normalize Data.
2. Standardize Data.
3. When to Normalize and Standardize.

These steps will provide the foundations you need to handle scaling your own data.

2.2.1 Normalize Data

Normalization can refer to different techniques depending on context. Here, we use normalization to refer to rescaling an input variable to the range between 0 and 1. Normalization requires that you know the minimum and maximum values for each attribute. This can be estimated from training data or specified directly if you have deep knowledge of the problem domain. You can easily estimate the minimum and maximum values for each attribute in a dataset by enumerating through the values. The snippet of code below defines the dataset minmax() function that calculates the min and max value for each attribute in a dataset, then returns an array of these minimum and maximum values.

In [ ]:
```
use strict;
use warnings;
use Data::Dump qw(dump);
use List::Util qw(zip min max sum);
use sml; # Statistical Machine Learning Library
```

In [1]:
```
# Function To Calculate the Min and Max Values For a Dataset.
# Find the min and max values for each column
sub dataset_minmax{
  my ($self, $dataset) = @_;

  my @minmax;
  for my $i (0 .. $#{$dataset->[0]}){ # Be careful not to include the Y labels
    my $col_values = [map {$_->[$i]}  @$dataset];
    my $value_min = min(@$col_values);
    my $value_max = max(@$col_values);
    push @minmax, [$value_min, $value_max];
  }
  return \@minmax;
}

sml->add_to_class('dataset_minmax', \&{'dataset_minmax'});
```

Out[1]:
```
*sml::dataset_minmax
```

```
Subroutine sml::dataset_minmax redefined at /usr/local/share/perl5/5.30/x86_64-linux-thread-multi/sml.pm line 22.
```

With this contrived dataset, we can test our function for calculating the min and max for each column.

In [2]:
```
# Contrive small dataset
my $dataset = [[50, 30], [20, 90]];
printf "%s\n", dump $dataset;
# Calculate min and max for each column
my $minmax = sml->dataset_minmax($dataset);
printf "%s\n", dump $minmax;
# Output of Example Calculating the Min and Max Values.
# [[50, 30], [20, 90]]
# [[20, 50], [30, 90]]
```

```
[[50, 30], [20, 90]]
[[20, 50], [30, 90]]
```

Out[2]: 1

Once we have estimates of the maximum and minimum allowed values for each column, we can now normalize the raw data to the range 0 and 1. The calculation to normalize a single value for a column is:

$$scaled\ value = (value - min) \ / \ (max - min)$$

(2.1)

Below is an implementation of this in a function called normalize dataset() that normalizes values in each column of a provided dataset.

In [7]:
```
# Function To Normalize a Dataset.
# Rescale dataset columns to the range 0-1
sub normalize_dataset{
  my ($self, $dataset, $minmax) = @_;
  for my $row (@$dataset){
    for my $i (0 .. $#{$row}){
      $row->[$i] = ($row->[$i] - $minmax->[$i][0]) / ($minmax->[$i][1] - $minmax->[$i][0]);
    }
  }
}

sml->add_to_class('normalize_dataset', \&{'normalize_dataset'});
```

Out[7]: *sml::normalize_dataset

```
Subroutine normalize_dataset redefined at reply input line 3.

Subroutine sml::normalize_dataset redefined at /usr/local/share/perl5/5.30/x86_64-linux-thread-multi/sml.pm line 22.
```

We can tie this function together with the dataset minmax() function and normalize the contrived dataset.

In [4]:
```
# Contrive small dataset
$dataset = [[50, 30], [20, 90]];
print dump $dataset;
# Calculate min and max for each column
$minmax = sml->dataset_minmax($dataset);
print "\n", dump $minmax;
# Normalize columns
sml->normalize_dataset($dataset, $minmax);
print "\n", dump $dataset;

# Example Output of Normalizing the Contrived Dataset.
# [[50, 30], [20, 90]]
```

```
# [[20, 50], [30, 90]]
# [[1, 0], [0, 1]]
```

```
[[50, 30], [20, 90]]
[[20, 50], [30, 90]]
[[1, 0], [0, 1]]
```

Out[4]: 1

We can combine this code with code for loading a CSV dataset and load and normalize the Pima Indians Diabetes dataset. The example first loads the dataset and converts the values for each column from string to floating point values. The minimum and maximum values for each column are estimated from the dataset, and finally, the values in the dataset are normalized.

In [5]:
```
# Load pima-indians-diabetes dataset
my $filename = '../data/pima-indians-diabetes.csv';
$dataset = sml->load_csv($filename);
printf "Loaded data file %s with %d rows and %d columns.\n", $filename, scalar @$dataset, scalar @{$dataset->[0]};
print "[@{$dataset->[0]}]";

# convert string columns to float
for my $i (0 .. $#{$dataset->[0]}){
  sml->str_column_to_float($dataset, $i);
}
print "\n[@{$dataset->[0]}]";

# Calculate min and max for each column
$minmax = sml->dataset_minmax($dataset);
sml->normalize_dataset($dataset, $minmax);
print "\n[@{$dataset->[0]}]";

# Example Output of Normalizing the Diabetes Dataset.
# Loaded data file pima-indians-diabetes.csv with 768 rows and 9 columns
# [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]
# [0.35294117647058826, 0.7437185929648241, 0.5901639344262295, 0.35353535353535354, 0.0,
# 0.5007451564828614, 0.23441502988898377, 0.48333333333333334, 1.0]
```

```
Loaded data file ../data/pima-indians-diabetes.csv with 768 rows and 9 columns.
[6 148 72 35 0 33.6 0.627 50 1]
[6.0 148.0 72.0 35.0 0.0 33.6 0.6 50.0 1.0]
[0.352941176470588 0.743718592964824 0.590163934426229 0.353535353535354 0 0.500745156482861 0.217391304347826 0.4833333
33333333 1]
```

Out[5]: 1

2.2.2 Standardize Data

Standardization is a rescaling technique that refers to centering the distribution of the data on the value 0 and the standard deviation to the value 1. Together, the mean and the standard deviation can be used to summarize a normal distribution, also called the Gaussian distribution or bell curve. It requires that the mean and standard deviation of the values for each column be known prior to scaling. As with normalizing above, we can estimate these values from training data, or use domain knowledge to specify their values. Let's start with creating functions to estimate the mean and standard deviation statistics for each column from a dataset. The mean describes the middle or central tendency for a collection of numbers. The mean for a column is calculated as the sum of all values for a column divided by the total number of values.

$$\sum_{i=1}^{n} values_i / count(values)$$

(2.2)

The function below named column_means() calculates the mean values for each column in the dataset.

In [6]:
```
# Function To Calculate Means For Each Column in a Dataset.
# Calculate column means
my $column_means = sub{
  my ($self, $dataset) = @_;
  my $means = [0, map {$_} 0 .. $#{$dataset->[0]} -1];
  for my $i (0 .. $#{$dataset->[0]}){
    my $col_values = [map {$_->[$i]} @$dataset];
    $means->[$i] = sum(@$col_values) / scalar(@$dataset);
  }
  return $means;
};

sml->add_to_class('column_means', \&{'column_means'});
```

Out[6]:
```
*sml::column_means
```

```
Subroutine sml::column_means redefined at /usr/local/lib/perl5/site_perl/5.32.1/x86_64-linux/sml.pm line 22.
```

The standard deviation describes the average spread of values from the mean. It can be calculated as the square root of the sum of the squared difference between each value and the mean and dividing by the number of values minus 1.

$$standard\ deviation = \sqrt{\sum_{i=1}^{n}(values_i - mean)^2 / count(values) - 1}$$

(2.3)

The function below named column stdevs() calculates the standard deviation of values for each column in the dataset and assumes the means have already been calculated.

In [8]:
```
# Function To Calculate Standard Deviations For Each Column in a Dataset.
# Calculate column standard deviations
my $column_stdevs = sub{
```

```
  my ($self, $dataset, $means) = @_;
  my $stdevs = [0, map {$_} 0 .. $#{$dataset->[0]} -1];
  for my $i (0 .. $#{$dataset->[0]}){
    my $variance = [map {($_->[$i] - $means->[$i]) ** 2} @$dataset];
    $stdevs->[$i] = sum(@$variance);
  }
  $stdevs = [map {sqrt($_ / (scalar(@$dataset) -1))} @$stdevs];
  return $stdevs;
};

sml->add_to_class('column_stdevs', \&{'column_stdevs'});
```

Out[8]:
```
*sml::column_stdevs
```

```
Subroutine sml::column_stdevs redefined at /usr/local/share/perl5/5.30/x86_64-linux-thread-multi/sml.pm line 22.
```

Using the contrived dataset, we can estimate the summary statistics.

In [8]:
```
# Standardize dataset
$dataset = [[50, 30], [20, 90], [30, 50]];
print dump $dataset;
# Estimate mean and standard deviation
my $means  = sml->column_means($dataset);
my $stdevs = sml->column_stdevs($dataset, $means);
printf "%s\n", dump $means;
printf "%s\n", dump $stdevs;

# Example Output From Calculating Statistics from the Contrived Dataset.
# [[50, 30], [20, 90], [30, 50]]
# [33.333333333333336, 56.666666666666664]
# [15.275252316519467, 30.550504633038933]
```

```
[[50, 30], [20, 90], [30, 50]][33.3333333333333, 56.6666666666667]
[15.2752523165195, 30.5505046330389]
```

Out[8]:
```
1
```

Once the summary statistics are calculated, we can easily standardize the values in each column. The calculation to standardize a given value is as follows:

$$standardized\_value_i = (value_i - mean) \ / \ stdev$$

(2.4)

Below is a function named standardize dataset() that implements this equation

In [9]:
```
# Function To Standardize a Dataset.
# Standardize dataset
```

```
sub standardize_dataset{
  my ($self, $dataset, $means, $stdevs) = @_;
  for my $row (@$dataset){
    for my $i (0 .. $#$row){
      $row->[$i] = ($row->[$i] - $means->[$i]) / $stdevs->[$i];
    }
  }
}

sml->add_to_class('standardize_dataset', \&{'standardize_dataset'});
```

Out[9]:
```
*sml::standardize_dataset
```

```
Subroutine sml::standardize_dataset redefined at /usr/local/share/perl5/5.30/x86_64-linux-thread-multi/sml.pm line 22.
```

Combining this with the functions to estimate the mean and standard deviation summary statistics, we can standardize our contrived dataset.

In [10]:
```
printf "%s\n", dump $means;
printf "%s\n", dump $stdevs;
# Standardize dataset
sml->standardize_dataset($dataset, $means, $stdevs);
printf "%s\n", dump $dataset;

# Example Output From Standardizing the Contrived Dataset.
# [[1.0910894511799618, -0.8728715609439694],
#  [-0.8728715609439697, 1.091089451179962],
#  [-0.21821789023599253, -0.2182178902359923]]
```

```
[33.3333333333333, 56.6666666666667]
[15.2752523165195, 30.5505046330389]
[
  [1.09108945117996, -0.872871560943969],
  [-0.87287156094397, 1.09108945117996],
  [-0.218217890235993, -0.218217890235992],
]
```

Out[10]:
```
1
```

Again, we can demonstrate the standardization of a machine learning dataset. The example below demonstrates how to load and standardize the Pima Indians diabetes dataset, assumed to be in the current working directory as in the previous normalization example.

In [12]:
```
# Load pima-indians-diabetes dataset
$filename = '../data/pima-indians-diabetes.csv';
$dataset = sml->load_csv($filename);
printf "Loaded data file %s with %d rows and %d columns.\n", $filename, scalar @$dataset, scalar @{$dataset->[0]};

# convert string columns to float
for my $i (0 .. $#{$dataset->[0]}){
```

```
  sml->str_column_to_float($dataset, $i);
}
printf "%s\n", dump $dataset->[0];

# Calculate min and max for each column
$minmax = sml->dataset_minmax($dataset);
sml->normalize_dataset($dataset, $minmax);
# Estimate mean and standard deviation
$means  = sml->column_means($dataset);
$stdevs = sml->column_stdevs($dataset, $means);
# standardize dataset
sml->standardize_dataset($dataset, $means, $stdevs);
printf "%s\n", dump $dataset->[0];

# Example Output From Standardizing the Diabetes Dataset.
# Loaded data file pima-indians-diabetes.csv with 768 rows and 9 columns
# [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]
# [0.6395304921176576, 0.8477713205896718, 0.14954329852954296, 0.9066790623472505,
# -0.692439324724129, 0.2038799072674717, 0.468186870229798, 1.4250667195933604,
# 1.3650063669598067]
```

```
Loaded data file ../data/pima-indians-diabetes.csv with 768 rows and 9 columns.
["6.0", "148.0", "72.0", "35.0", "0.0", 33.6, 0.6, "50.0", "1.0"]
[
  0.639530492117648,
  0.847771320589669,
  0.149543298529545,
  0.906679062347249,
  -0.69243932472413,
  0.203879907267472,
  0.384829971238835,
  1.42506671959336,
  1.36500636695981,
]
```

Out[12]: 1

2.2.3 When to Normalize and Standardize

Standardization is a scaling technique that assumes your data conforms to a normal distribution. If a given data attribute is normal or close to normal, this is probably the scaling method to use. It is good practice to record the summary statistics used in the standardization process so that you can apply them when standardizing data in the future that you may want to use with your model. Normalization is a scaling technique that does not assume any specific distribution.

If your data is not normally distributed, consider normalizing it prior to applying your machine learning algorithm. It is good practice to record the minimum and maximum values for each column used in the normalization process, again, in case you need to normalize new data in the future to

be used with your model.

2.3 Extensions

There are many other data transforms you could apply. The idea of data transforms is to best expose the structure of your problem in your data to the learning algorithm. It may not be clear what transforms are required upfront. A combination of trial and error and exploratory data analysis (plots and stats) can help tease out what may work. Below are some additional transforms you may want to consider researching and implementing:

- Normalization that permits a configurable range, such as -1 to 1 and more.
- Standardization that permits a configurable spread, such as 1, 2 or more standard deviations from the mean.
- Exponential transforms such as logarithm, square root and exponents.
- Power transforms such as Box-Cox for fixing the skew in normally distributed data.

2.4 Review

In this tutorial, you discovered how to rescale your data for machine learning from scratch. Specifically, you learned:

- How to normalize data from scratch.
- How to standardize data from scratch.
- When to use normalization or standardization on your data.