Encog 3.3: Quick Start Guide

Encog 3.3: Quick Start Guide

Jeff Heaton

Heaton Research, Inc.
St. Louis, MO, USA

Title	Encog 3.3: Quick Start Guide
Author	Jeff Heaton
Published	October 01, 2014
Copyright	Copyright 2014 by Heaton Research, Inc., All Rights Reserved.
ISBN	
Price	FREE
File Created	Sat Oct 04 17:25:39 CDT 2014

Contents

Chapter 1

Using Encog for Java & C#

- Encog Java Examples
- Encog C# Examples
- Using an IDE

Encog is available for both Java and .Net. The next sections will show you how to make use of the Encog examples, as well as create your own Encog projects.

1.1 Using Encog with Java

Encog 3.3 requires Java 1.7 or higher. If you do not already have Java installed, you will need to install Java. It is important that you properly install Java and ensure that Java is both in your path and the **JAVA_HOME** environmental variable is defined.

1.1.1 Installing Java

The exact procedure to install Java varies greatly across Windows, Macintosh and Linux. Installing Java is beyond the scope of this document. For complete installation instructions for Java, refer to the following URL:

http://www.java.com/en/download/help/download_options.xml

You can easily verify if Java is installed properly by running **java -version** and echoing **JAVA_HOME**. Here I perform this test on Windows.

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\Jeff>java -version
java version "1.7.0_45"
Java(TM) SE Runtime Environment (build 1.7.0_45-b18)
Java HotSpot(TM) 64-Bit Server VM (build 24.45-b08, mixed mode)
C:\Users\Jeff>echo %java_home%
C:\java\jdk1.7.0_45
C:\Users\Jeff>
```

The process for Linux is very similar, as can be seen here:

```
[jheaton@jeffdev java]$ java -version
java version "1.7.0_13"
Java(TM) SE Runtime Environment (build 1.7.0_13-b20)
Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)
[jheaton@jeffdev java]$ echo $JAVA_HOME
/usr/java/default
[jheaton@jeffdev java]$ ^C
[jheaton@jeffdev java]$
```

Now that you are sure Java is installed, you are ready to download Encog.

1.1.2 Downloading Encog

All of the important Encog links can be found at the following URL.

http://www.encog.org

At the above link you will find instructions for downloading the latest version of Encog.

It is also possible to obtain the Encog examples directly from GitHub. The following command will pull the latest Encog examples

```
git clone https://github.com/encog/encog-java-examples.git
```

Once you've obtained the Encog examples, you are ready to run them.

1.1.3 Encog Java from the Command Line

All Encog examples can be run from the command line using the Gradle build management system. It is not necessary to have Gradle installed to run the examples. However, Gradle can be very useful when you choose to create your own Encog projects. Gradle allows you to specify Encog as a dependency to your project and download the correct version of Encog automatically. The examples contain the Gralde wrapper. If you simply use the Gradle wrapper you do not need to download and install Gradle. The following instructions assume that you are using the Gradle wrapper.

If you are using a Linux/UNIX operating system, it may be necessary to grant **gradlew** permission to execute. To do this, execute the following command from the Encog examples directory.

```
chmod +x ./gradlew
```

You can use the following Gradle command to determine what examples you can run.

```
gradlew tasks
```

If you are using a UNIX operating system, it might be necessary to prefix gradlew with a "./", as seen here.

```
./gradlew tasks
```

This will list all of the Encog examples and the tasks to run them. For example, to run the XOR neural network "Hello World" example, use the following command in Windows:

```
gradlew runHelloWorld
```

In Linux, you might have to use:

```
./gradlew runHelloWorld
```

This should result in the following output.

```
[jheaton@jeffdev encog-java-examples]$ ./gradlew runHelloWorld
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
```

```
:classes UP-TO-DATE
:runHelloWorld
Epoch #1 Error:0.32169908465997293
Epoch #2 Error:0.3001583911638903
Epoch #3 Error:0.27814800047830207
Epoch #4 Error:0.2591350408848929
Epoch #5 Error:0.24807257611353625
Epoch #6 Error:0.24623233964519337
Epoch #7 Error:0.2448993459247424
Epoch #8 Error:0.24054454230164823
Epoch #9 Error:0.2368200193886572
Epoch #10 Error:0.23219970754041114
...
Epoch #96 Error:0.017080335499927907
Epoch #97 Error:0.012487031230186490
Epoch #98 Error:0.00918572008572443
Neural Network Results:
0.0,0.0, actual=0.037434516460193114,ideal=0.0
1.0,0.0, actual=0.8642455025347225,ideal=1.0
0.0,1.0, actual=0.8950073477748369,ideal=1.0
1.0,1.0, actual=0.0844306876871185,ideal=0.0
BUILD SUCCESSFUL
Total time: 4.401 secs
[jheaton@jeffdev encog-java-examples]$
```

The XOR "Hello World" application shows how to train a neural network to learn the XOR function.

1.1.4 Creating a Stand Alone Encog Project in Java

The easiest way to setup an Encog standalone application is to use Gradle. This approach automatically downloads the latest version of Encog from the Maven central repository. If you need instructions for installing Gradle, they can be found here.

`http://www.gradle.org/installation`

Listing 1.1 shows a very simple Encog project in Gradle.

Listing 1.1: Sample Gradle Project (build.gradle)

```
apply plugin: 'java'
apply plugin: 'application'
```

```
targetCompatibility = 1.7
sourceCompatibility = 1.7
repositories {
    mavenCentral()
}
dependencies {
    compile 'org.encog:encog-core:3.3.0'
}
task(runExample, group: 'examples',
   description: 'Run the sample stand-alone project.',
        dependsOn: 'classes', type: JavaExec) {
    main = 'HelloWorld'
    classpath = sourceSets.main.runtimeClasspath
}
```

The same thing can also be accomplished with Maven.

Listing 1.2: Sample Maven Project (pom.xml)

```
<project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
   maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.jeffheaton</groupId>
  <artifactId>encog-sample-java</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Encog standalone sample</name>
  <url>http://www.encog.org</url>

  <build>
  <plugins>
  <plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.2.1</version>
  <executions>
    <execution>
      <goals>
        <goal>java</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <mainClass>HelloWorld</mainClass>
  </configuration>
</plugin>
</plugins>
</build>
  <dependencies>
    <dependency>
    <groupId>org.encog</groupId>
    <artifactId>encog-core</artifactId>
    <version>3.3.0</version>
</dependency>
  </dependencies>
</project>
```

The Gradle and Maven project files both make use of Listing 1.3,

Listing 1.3: Sample Encog Application (HelloWorld.java)

```
import org.encog.Encog;
import org.encog.engine.network.activation.ActivationSigmoid;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.basic.BasicMLDataSet;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
    ResilientPropagation;

public class HelloWorld {

/**
 * The input necessary for XOR.
 */
public static double XOR_INPUT[][] = { { 0.0, 0.0 },
        { 1.0, 0.0 }, { 0.0, 1.0 }, { 1.0, 1.0 } };

/**
 * The ideal data necessary for XOR.
 */
public static double XOR_IDEAL[][] = { { 0.0 },
         { 1.0 }, { 1.0 }, { 0.0 } };

/**
 * The main method.
 * @param args No arguments are used.
 */
public static void main(final String args[]) {

// create a neural network, without using a factory
BasicNetwork network = new BasicNetwork();
network.addLayer(
              new BasicLayer(null,true,2));
network.addLayer(
              new BasicLayer(new ActivationSigmoid(),true,3));
network.addLayer(
              new BasicLayer(new ActivationSigmoid(),false,1));
network.getStructure().finalizeStructure();
network.reset();

// create training data
MLDataSet trainingSet = new BasicMLDataSet(
              XOR_INPUT, XOR_IDEAL);

// train the neural network
final ResilientPropagation train =
              new ResilientPropagation(network, trainingSet);

int epoch = 1;

do {
train.iteration();
System.out.println(
          "Epoch #" + epoch + " Error:" + train.getError());
epoch++;
} while(train.getError() > 0.01);
train.finishTraining();

// test the neural network
System.out.println("Neural Network Results:");
for(MLDataPair pair: trainingSet ) {
final MLData output = network.compute(pair.getInput());
System.out.println(pair.getInput().getData(0)
             + "," + pair.getInput().getData(1)
```

```
    + ", actual=" + output.getData(0) + ",ideal="
+ pair.getIdeal().getData(0));
}

Encog.getInstance().shutdown();
}
}
```

You can find this complete example on GitHub at the following URL.

`https://github.com/encog/encog-sample-java`

To run the project under Gradle, use the following command:

```
gradle runExample
```

To run the project under Maven, use the following command:

```
Mvn exec:java
```

1.1.5 Encog Java from an IDE

There are a number of different IDE's for the Java programming language. Additionally, there are a number of different ways to make use of a third party library, such as Encog, in each IDE. I make use of IntelliJ and simply import the Gradle project. This allows my project to easily be used from either an IDE or the command line. You might also be able to instruct your IDE to pull the Encog JAR from Maven central:

`http://search.maven.org/#search%7Cga%7C1%7Ca%3A%22encog-core%22`

1.2 Using Encog with .Net

Encog 3.3 requires Microsoft .Net 3.5 or higher. This is normally installed with Visual Studio. For more information about .Net visit the following URL:

`http://www.microsoft.com/net`

Encog can be used with any .Net programming language. The instructions in this guide pertain to using Encog with C#. With some adaptation, these instructions are also be useful for other .Net languages.

1.2.1 Downloading Encog

All of the important Encog links can be found at the following URL.

`http://www.encog.org`

At the above link you will find instructions for downloading the latest version of Encog.

It is also possible to obtain the Encog examples directly from GitHub. The following command will pull the latest Encog examples and core:

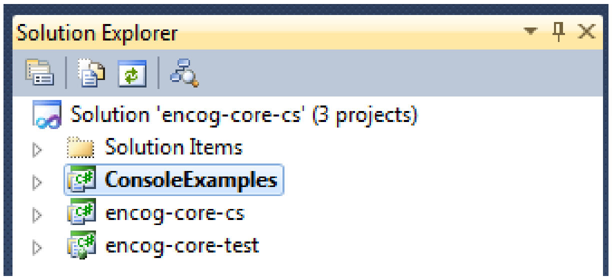
```
git clone https://github.com/encog/encog-dotnet-core.git
```

Once you've obtained the Encog examples, you are ready to run them.

1.2.2 Running Encog .Net Examples

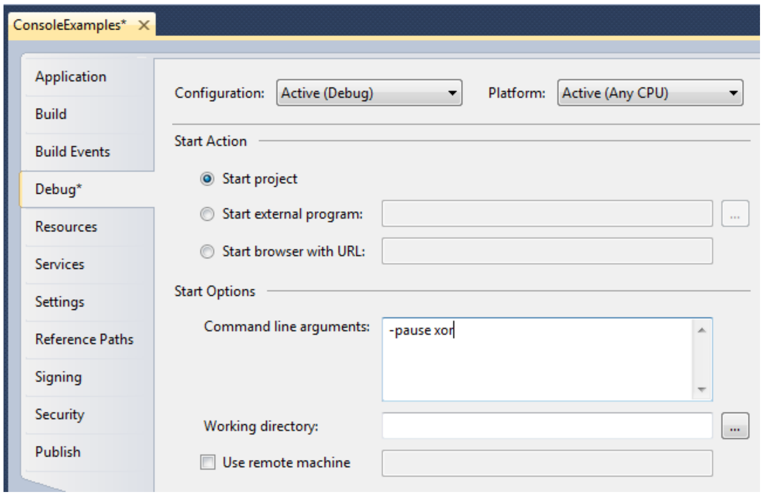
The Encog C# examples and core are both contained in the **encog-core-cs.sln** solution file, as seen in Figure 1.1.

Figure 1.1: Encog C# Examples and Core



Running the **ConsoleExamples** project allows you to run the individual examples. The arguments provided to this application determine what example will run. Figure 1.2 shows the arguments needed to run an XOR example.

Figure 1.2: Example Arguments



Configuration: Active (Debug)
Platform: Active (Any CPU)
Start Action
Start Options
Command line arguments: -pause xor
Working directory:

As you can see, we specified the **xor** example and requested that a pause occur before the program exited. You can also specify a **?** to see all available examples. This will produce output similar to the following.

```
adalinedigits        : ADALINE Digits
analyst              : Encog Analyst
art1-classify        : Classify Patterns with ART1
bam                  : Bidirectional Associative Memory
bayesian-taxi        : The taxicab problem with Bayesian networks.
benchmark            : Perform an Encog benchmark.
benchmark-elliott    : Perform a benchmark of the Elliott
   activation function.
benchmark-simple     : Perform a simple Encog benchmark.
cpn                  : Counter Propagation Neural Network (CPN)
csvmarket            : Simple Market Prediction
CSVPredict           : CSVPredict
encoder              : A Fahlman encoder.
epl-simple           : Simple EPL equation solve.
forest               : Forest Cover
Forex                : Predict Forex rates via CSV.
freeform-convert     : Freeform Network: convert flat network to
   freeform
freeform-elman       : Freeform Network: Elman SRN
freeform-online-xor  : Freeform Network: Online XOR
freeform-skip        : Freeform Network: Skip network
```

```
freeform-xor          : Freeform Network: XOR
guide-auto-mpg        : Encog Guide: Regression: Predict an auto's
   MPG.
guide-iris            : Encog Guide: Classification: The Iris Data
   Set.
guide-sunspots        : Encog Guide: Time Series Regression: Predict
    Sunspots.
hopfield-associate    : Hopfield Associates Patterns
hopfield-simple       : Hopfield Recognize Simple Patterns
image                 : Image Neural Networks
indicator-download    : Download data from Ninjatrader.
indicator-ema         : Provide a EMA indicator.
indicator-sma         : Provide an example indicator.
lander                : Train a neural network to land a space ship.
livesimul             : A quick example that simulates neuronal
   activty in live si
tuation and shows how to process data and compute into the network
    in 2 commands
..
market                : Simple Market Prediction
multibench            : Multithreading Benchmark
normalize-file        : Normalize a file.
normalize-simple      : Simple normalize example.
persist-encog         : Persist using .Net Serialization
persist-serial        : Persist using .Net Serialization
playground            : Not an actual example. Do not run.
Prunate               : Prunes network
radial-multi          : A RBF network example.
Range                 : Analyzes ranges and predicts them.
SP500                 : Predicts the SNP500
sunspot               : Predict sunspots.
sunwindow             : Predict sunspots w/Window.
SVM                   : Random Makes a small SVM random trainer.
SVMCSV                : SVMCSV
svmreg                : This example shows how to preform simple
   regression with a
 SVM.
threadcount           : Evaluate Thread Count Performance
tsp-anneal            : Annealing Traveling Salesman
tsp-boltzmann         : Boltzmann Machine for the Traveling Salesman
    (TSP)
tsp-genetic           : Genetic Algorithm Traveling Salesman
weight-init           : weight Initializers
```

```
xor                    : Simple XOR with backprop, no factories or
    helper functions
.
xor-elman              : Elman Temporal XOR
xor-factory            : Use XOR with many different training and
    network types.
xor-jordan             : Jordan Temporal XOR
xor-neat               : Simple XOR with NEAT.
xor-online             : Simple XOR with backprop, use online
    training.
xor-pso                : Simple XOR with PSO (particle stream)
    training.
```

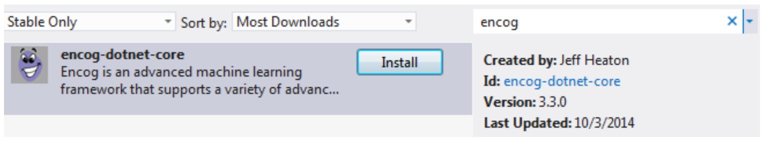
1.2.3 Creating a Stand Alone Encog Project in .Net

The easiest method for adding Encog to a C# application is NuGet. Third party DLL's can be browsed and installed using NuGet. You can see the Encog NuGet page at the following URL:

`https://www.nuget.org/packages/encog-dotnet-core/`

To see how to create a simple Encog XOR application, start Visual Studio and create a new console application. Right-click the references tab and choose "Manage NuGet PackagesĚ". Enter "Encog" into the search box and perform a search. You should see Encog listed, as in Figure 1.3.

Figure 1.3: Encog in NuGet



encog-dotnet-core
Encog is an advanced machine learning framework that supports a variety of advanc...

Created by: Jeff Heaton
Id: encog-dotnet-core
Version: 3.3.0
Last Updated: 10/3/2014

Click the "Install" button, and Encog will be added to your project. You should now modify your **Program.cs** file to look similar to the below example in Listing 1.4. Note, that I named my project **encog_sample_csharp**, your namespace line will match your project name.

Listing 1.4: Simple C# XOR Example

```
using System;
using Encog.Neural.Networks;
using Encog.Neural.Networks.Layers;
using Encog.Engine.Network.Activation;
using Encog.ML.Data;
using Encog.Neural.Networks.Training.Propagation.Resilient;
using Encog.ML.Train;
using Encog.ML.Data.Basic;
using Encog;
namespace encog_sample_csharp
{
  internal class Program
  {
    /// <summary>
    /// Input for the XOR function.
    /// </summary>
    public static double[][] XORInput =
    {
      new[] {0.0, 0.0},
      new[] {1.0, 0.0},
      new[] {0.0, 1.0},
      new[] {1.0, 1.0}
    };
    /// <summary>
    /// Ideal output for the XOR function.
    /// </summary>
    public static double[][] XORIdeal =
    {
      new[] {0.0},
      new[] {1.0},
      new[] {1.0},
      new[] {0.0}
    };
    private static void Main(string[] args)
    {
      // create a neural network, without using a factory
      var network = new BasicNetwork();
      network.AddLayer(new BasicLayer(null, true, 2));
      network.AddLayer(new BasicLayer(new ActivationSigmoid(),
          true, 3));
      network.AddLayer(new BasicLayer(new ActivationSigmoid(),
          false, 1));
      network.Structure.FinalizeStructure();
      network.Reset();
      // create training data
      IMLDataSet trainingSet = new BasicMLDataSet(XORInput,
         XORIdeal);
      // train the neural network
      IMLTrain train = new ResilientPropagation(network,
         trainingSet);
      int epoch = 1;
      do
      {
        train.Iteration();
        Console.WriteLine(@"Epoch #" + epoch + @" Error:" + train.
           Error);
        epoch++;
      } while (train.Error > 0.01);
      train.FinishTraining();
      // test the neural network
      Console.WriteLine(@"Neural Network Results:");
      foreach (IMLDataPair pair in trainingSet)
      {
        IMLData output = network.Compute(pair.Input);
        Console.WriteLine(pair.Input[0] + @"," + pair.Input[1]
        + @", actual=" + output[0] + @",ideal=" + pair.Ideal[0]);
      }
      EncogFramework.Instance.Shutdown();
    }
  }
}
```

You can find this complete example at the following GitHub URL:

`https://github.com/encog/encog-sample-csharp`

Chapter 2

Encog Quick Start Examples

- Using Encog for Classification
- Using Encog for Regression
- Using Encog for Time Series

This chapter will take you through three non-trivial Encog examples. These examples are designed to be starting points for your own projects. These examples demonstrate classification, regression and time-series.

2.1 Using Encog for Classification

Classification problems seek to place data set elements into predefined classes. The dataset that will be used for this example is Fisher's Iris dataset. This is a classic dataset that contains measurements for 150 different Iris flowers. Each of the 150 flowers contains four measurements. The species of Iris is also provided. For this example we would like to train a machine-learning model to classify the species of iris given the four measurements. This dataset can be found at the following URL:

`https://archive.ics.uci.edu/ml/datasets/Iris`

A sampling of the dataset is shown here.

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
```

This dataset has no column headers and is comma delineated. Each additional line provides the measurements and species of a particular flower.

We will create a program that generates a model to predict the type of iris, based on the four measurements. This program will allow us to easily change the model type to any of the following:

- Feedforward Neural Network
- NEAT Neural Network
- Probabilistic Neural Network
- RBF Neural Network
- Support Vector Machine

When you change the model type, Encog will automatically change the way that the data are normalized.

This program will split the training data into a training and validation set. The validation set will be held until the end to see how well we can predict data that the model was not trained on. Training will be performed using a 5-fold cross-validation.

This complete example can be found with the Encog examples. The Java version contains this example here:

```
org.encog.examples.guide.classification.IrisClassification
```

The C# version can be executed with the argument **guide-iris**, and can be found at the following location:

```
Encog.Examples.Guide.Classification.IrisClassification
```

2.1.1 Mapping the Input File

We begin by defining a **VersatileMLDataSet** object that will load from a CSV file. We define the five columns of the Iris data set. The file, downloaded from the UCI site, does not contain column headers. Because of this we must name, and specify the index of, each column. We define the four measurements as continuous. A continuous value is similar to a Java or C# double. We define the Iris species as a nominal value. A nominal value species a class, but there is no implied order, as one type of species is not "greater than" another. The following Java code is used to define the Iris file.

```
VersatileDataSource source = new CSVDataSource(
  irisFile, false,
  CSVFormat.DECIMAL_POINT);
  VersatileMLDataSet data = new VersatileMLDataSet(source);
data.defineSourceColumn(
  "sepal-length", 0, ColumnType.continuous);
data.defineSourceColumn(
  "sepal-width", 1, ColumnType.continuous);
data.defineSourceColumn(
  "petal-length", 2, ColumnType.continuous);
data.defineSourceColumn(
  "petal-width", 3, ColumnType.continuous);
ColumnDefinition outputColumn = data.defineSourceColumn(
    "species", 4, ColumnType.nominal);
data.analyze();
```

The following C# code accomplishes the same thing.

```
IVersatileDataSource source = new CSVDataSource(
  irisFile, false,
  CSVFormat.DecimalPoint);

var data = new VersatileMLDataSet(source);
data.DefineSourceColumn(
    "sepal-length", 0, ColumnType.Continuous);
data.DefineSourceColumn(
    "sepal-width", 1, ColumnType.Continuous);
data.DefineSourceColumn(
    "petal-length", 2, ColumnType.Continuous);
data.DefineSourceColumn(
    "petal-width", 3, ColumnType.Continuous);
ColumnDefinition outputColumn = data.DefineSourceColumn(
    "species", 4,
    ColumnType.Nominal);
data.Analyze();
```

The final step is to call the **Analyze** method. This reads the entire file and determines the minimum, maximum, mean and standard deviations for each column. These statistics will be useful for both normalization and interpolation of missing values. Fortunately, the iris data set has no missing values.

2.1.2 Specifying the Model & Normalizing

Before we can normalize the data, we must choose our desired model type. The model type often dictates how the data should be normalized. For this example, I will use a feedforward neural network. We must also specify the column that we are going to predict. In this case, we are predicting the iris species. Because the iris species is non-numeric, this is a classification problem. Performing a regression problem is simply a matter of choosing to predict a numeric column.

We also choose to send all output to the console. Now that everything is set, we can normalize. The normalization process will load the CSV file into memory and normalize the data as it is loaded.

The following Java code accomplishes this.

```
// Map the prediction column to the output of the model, and all
// other columns to the input.
data.defineSingleOutputOthersInput(outputColumn);
EncogModel model = new EncogModel(data);
model.selectMethod(data, MLMethodFactory.TYPE_FEEDFORWARD);
model.setReport(new ConsoleStatusReportable());
data.normalize();
```

The following C# code accomplishes the same thing.

```
// Map the prediction column to the output of the model, and all
// other columns to the input.
data.DefineSingleOutputOthersInput(outputColumn);
var model = new EncogModel(data);
              model.SelectMethod(data, MLMethodFactory.
                  TypeFeedforward);
```

```
model.Report = new ConsoleStatusReportable();
data.Normalize();
```

The data are now in memory and ready for use.

2.1.3 Fitting the Model

Before we fit the model we hold back part of the data for a validation set. We choose to hold back 30%. We chose to randomize the data set with a fixed seed value. This fixed seed ensures that we get the same training and validation sets each time. This is a matter of preference. If you want a random sample each time then pass in the current time for the seed. Finally, we fit the model with a k-fold cross-validation of size 5.

The following Java code accomplishes this.

```
model.holdBackValidation(0.3, true, 1001);
model.selectTrainingType(data);
MLRegression bestMethod = (MLRegression)model.crossvalidate(5,
    true);
```

The following C# code accomplishes the same.

```
model.HoldBackValidation(0.3, true, 1001);
model.SelectTrainingType(data);
var bestMethod = (IMLRegression) model.Crossvalidate(5, true);
```

Cross-validation breaks the training dataset into 5 different combinations of training and validation data. Do not confuse the cross-validation validation data with the ultimate validation data that we set aside previously. The cross-validation process does not use the validation data that we previously set aside. Those data are for a final validation, after training has occurred.

At the end of the cross-validation training you will obtain the best model of the 5 folds. You will also see the cross-validated error. This error is an average of the validation errors of the five folds. The cross-validation error is an estimate how your model might perform on data that it was not trained on.

2.1.4 Displaying the Results

We can now display several of the errors. We can check the training error and validation errors. We can also display the stats gathered on the data.

The following Java code accomplishes this.

```
System.out.println( "Training error: "
    + EncogUtility.calculateRegressionError(bestMethod,
    model.getTrainingDataset()));
System.out.println( "Validation error: "
    + EncogUtility.calculateRegressionError(bestMethod,
    model.getValidationDataset()));
NormalizationHelper helper = data.getNormHelper();
System.out.println(helper.toString());
System.out.println("Final model: " + bestMethod);
```

The following C# code accomplishes the same.

```
Console.WriteLine(@"Training error: "
    + EncogUtility.CalculateRegressionError(bestMethod,
    model.TrainingDataset));
Console.WriteLine(@"Validation error: "
    + EncogUtility.CalculateRegressionError(bestMethod,
    model.ValidationDataset));
NormalizationHelper helper = data.NormHelper;
Console.WriteLine(helper.ToString());
Console.WriteLine(@"Final model: " + bestMethod);
```

2.1.5 Using the Model & Denormalizing

Once you've trained a model you will likely want to use this mode. The best model can be saved using normal serialization. However, you will need a way to normalize data going into the model, and denormalize data coming out of the mode. The normalization helper object, obtained in the previous section, can do this for you. You can also serialize the normalization helper.

The following Java code opens the CSV file and predicts each iris using the best model and normalization helper.

```
ReadCSV csv = new ReadCSV(
    irisFile, false, CSVFormat.DECIMAL_POINT);
```

```
String[] line = new String[4];
MLData input = helper.allocateInputVector();
while(csv.next()) {
    StringBuilder result = new StringBuilder();
    line[0] = csv.get(0);
    line[1] = csv.get(1);
    line[2] = csv.get(2);
    line[3] = csv.get(3);
    String correct = csv.get(4);
    helper.normalizeInputVector(line,input.getData(),false);
    MLData output = bestMethod.compute(input);
    String irisChosen =
        helper.denormalizeOutputVectorToString(output)[0];
        result.append(Arrays.toString(line));
        result.append(" -> predicted: ");
        result.append(irisChosen);
        result.append("(correct: ");
        result.append(correct);
        result.append(")");
        System.out.println(result.toString());
}
// Delete data file ande shut down.
irisFile.delete();
Encog.getInstance().shutdown();
```

The following C# code accomplishes similar.

```
var csv = new ReadCSV(irisFile, false, CSVFormat.DecimalPoint);
var line = new String[4];
IMLData input = helper.AllocateInputVector();
while (csv.Next())
{
    var result = new StringBuilder();
    line[0] = csv.Get(0);
    line[1] = csv.Get(1);
    line[2] = csv.Get(2);
    line[3] = csv.Get(3);
    String correct = csv.Get(4);
    helper.NormalizeInputVector(
        line, ((BasicMLData) input).Data, false);
    IMLData output = bestMethod.Compute(input);
    String irisChosen =
        helper.DenormalizeOutputVectorToString(output)[0];
    result.Append(line);
    result.Append(" -> predicted: ");
    result.Append(irisChosen);
    result.Append("(correct: ");
    result.Append(correct);
    result.Append(")");
    Console.WriteLine(result.ToString());
}
```

The output from this program will look similar to the following. First the program downloads the data set and begins training. Training occurs over 5 folds. Each fold uses a separate portion of the training data as validation. The remaining portion of the training data is used to train the model for that fold. Each fold gives us a different model; we choose the model with the best validation score. We train until the validation score ceases to improve. This helps to prevent over-fitting. The first fold trains for 48 iterations before it stops:

```
Downloading Iris dataset to: /var/folders/m5/
   gbcvpwzj7gjdb41z1_x9rzch0000gn/T/iris.csv
1/5 : Fold #1
1/5 : Fold #1/5: Iteration #1, Training Error: 1.34751708,
   Validation Error: 1.42040606
1/5 : Fold #1/5: Iteration #2, Training Error: 0.99412971,
   Validation Error: 1.42040606
...
1/5 : Fold #1/5: Iteration #47, Training Error: 0.03025748,
   Validation Error: 0.00397662
1/5 : Fold #1/5: Iteration #48, Training Error: 0.03007620,
   Validation Error: 0.00558196
```

The first fold had a very good validation error, and we move on to the second fold.

```
2/5 : Fold #2
2/5 : Fold #2/5: Iteration #1, Training Error: 1.10153372,
   Validation Error: 1.22069520
2/5 : Fold #2/5: Iteration #2, Training Error: 0.58543151,
   Validation Error: 1.22069520
...
2/5 : Fold #2/5: Iteration #28, Training Error: 0.04351376,
   Validation Error: 0.15599265
```

```
2/5 : Fold #2/5: Iteration #29, Training Error: 0.04061504,
   Validation Error: 0.15599265
2/5 : Fold #2/5: Iteration #30, Training Error: 0.03745747,
   Validation Error: 0.15844284
```

The second fold did not have a very good validation error. It is important to note that that the folds are independent of each other. Each fold starts with a new model.

```
3/5 : Fold #3
3/5 : Fold #3/5: Iteration #1, Training Error: 1.13685270,
   Validation Error: 1.09062392
3/5 : Fold #3/5: Iteration #2, Training Error: 0.78567165,
   Validation Error: 1.09062392
...
3/5 : Fold #3/5: Iteration #47, Training Error: 0.01850279,
   Validation Error: 0.04417794
3/5 : Fold #3/5: Iteration #48, Training Error: 0.01889085,
   Validation Error: 0.05261448
```

Fold 3 did somewhat better than fold 2, but not as good as fold 1. We now begin fold 4.

```
4/5 : Fold #4
4/5 : Fold #4/5: Iteration #1, Training Error: 1.15492772,
   Validation Error: 1.17098262
4/5 : Fold #4/5: Iteration #2, Training Error: 0.56095813,
   Validation Error: 1.17098262
...
4/5 : Fold #4/5: Iteration #41, Training Error: 0.01982776,
   Validation Error: 0.08958218
4/5 : Fold #4/5: Iteration #42, Training Error: 0.02225716,
   Validation Error: 0.09186468
```

Fold 4 finished with a validation of 0.09.

```
5/5 : Fold #5
5/5 : Fold #5/5: Iteration #1, Training Error: 1.13078723,
   Validation Error: 1.31180090
5/5 : Fold #5/5: Iteration #2, Training Error: 0.73188602,
   Validation Error: 1.31180090
...
5/5 : Fold #5/5: Iteration #47, Training Error: 0.02974431,
   Validation Error: 0.00218783
```

```
5/5 : Fold #5/5: Iteration #48, Training Error: 0.03006581,
    Validation Error: 0.00270633
5/5 : Cross-validated score:0.06224205725984283
```

After fold 5 is complete, we report the cross-validated score that is the average of all 5 validation scores. This should give us a reasonable estimate of how well the model might perform on data that it was not trained with. Using the best model, from the 5 folds, we now evaluate it with the training data and the true validation data that we set aside earlier.

```
Training error: 0.023942862952610295
Validation error: 0.061413317688009464
```

As you can see, the training error is lower than the validation error. This is normal, as models always tend to perform better on data that they were trained with. However, it is important to note that the validation error is close to the cross-validated error. The cross-validated error will often give us a good estimate of how our model will perform on untrained data.

Finally, we display the normalization data. This shows us the min, max, mean and standard deviation for each column.

```
[NormalizationHelper:
[ColumnDefinition:sepal-length(continuous);low=4.300000,high
    =7.900000,mean=5.843333,sd=0.825301]
[ColumnDefinition:sepal-width(continuous);low=2.000000,high
    =4.400000,mean=3.054000,sd=0.432147]
[ColumnDefinition:petal-length(continuous);low=1.000000,high
    =6.900000,mean=3.758667,sd=1.758529]
[ColumnDefinition:petal-width(continuous);low=0.100000,high
    =2.500000,mean=1.198667,sd=0.760613]
[ColumnDefinition:species(nominal);[Iris-setosa, Iris-versicolor,
    Iris-virginica]]
]
```

Finally, we loop over the entire dataset and display predictions. This part of the example shows you how to use the model with new data you might acquire. However, for new data, you might not have the correct outcome, as that is what you seek to predict.

```
Final model: [BasicNetwork: Layers=3]
[5.1, 3.5, 1.4, 0.2] -> predicted: Iris-setosa(correct: Iris-
    setosa)
```

```
[4.9, 3.0, 1.4, 0.2] -> predicted: Iris-setosa(correct: Iris-
   setosa)
...
[7.0, 3.2, 4.7, 1.4] -> predicted: Iris-versicolor(correct: Iris-
   versicolor)
[6.4, 3.2, 4.5, 1.5] -> predicted: Iris-versicolor(correct: Iris-
   versicolor)
...
[6.3, 3.3, 6.0, 2.5] -> predicted: Iris-virginica(correct: Iris-
   virginica)
...
```

2.2 Using Encog for Regression

Regression problems seek to produce a numeric outcome from the input data. In this section we will create a model that attempts to predict the miles-per-gallon that a particular car will achieve. This example makes use of the UCI auto MPG dataset that can be found at the following URL:

`https://archive.ics.uci.edu/ml/datasets/Auto+MPG`

A sampling of the dataset is shown here.

```
18.0   8   307.0      130.0      3504.      12.0   70   1"chevrolet
    chevelle malibu"
15.0   8   350.0      165.0      3693.      11.5   70   1"buick
   skylark 320"
18.0   8   318.0      150.0      3436.      11.0   70   1"plymouth
   satellite"
16.0   8   304.0      150.0      3433.      12.0   70   1"amc rebel
    sst"
17.0   8   302.0      140.0      3449.      10.5   70   1"ford
   torino"
```

As you can see, from the data, there are no column headings and the data is space-separated. This must be considered when mapping the file to a dataset. The UCI database tells us that the columns represent the following:

```
    1. mpg:            continuous
    2. cylinders:      multi-valued discrete
    3. displacement:   continuous
    4. horsepower:     continuous
    5. weight:         continuous
    6. acceleration:   continuous
    7. model year:     multi-valued discrete
    8. origin:         multi-valued discrete
    9. car name:       string (unique for each instance)
```

We will create a program that generates a model to predict the MPG for the car, based on some of the other values. This program will allow us to easily change the model type to any of the following:

- Feedforward Neural Network
- NEAT Neural Network
- Probabilistic Neural Network
- RBF Neural Network
- Support Vector Machine

When you change the model type, Encog will automatically change the way that the data are normalized.

This program will split the training data into a training and validation set. The validation set will be held until the end to see how well we can predict data that the model was not trained on. Training will be performed using a 5-fold cross-validation.

This complete example can be found with the Encog examples. The Java version contains this example here.

```
org.encog.examples.guide.regression.AutoMPGRegression
```

The C# version can be executed with the argument **guide-auto-mpg**, and can be found at the following location.

```
Encog.Examples.Guide.Regression.AutoMPGRegression
```

2.2.1 Mapping the Input File

We begin by defining a **VersatileMLDataSet** object that will read from the UCI provided data file. We define the columns of the auto MPG dataset. The file, downloaded from the UCI site, does not contain column headers. Because of this we must name, and specify the index of, each column. We define the cylinder count and model year as ordinal. Even though both of these values appear numeric, we treat them as classes. However, they do have an implied ordering, so we must define that ordering. You can see below that the code lists every value for cylinders and model years. If we missed a value, an error will be raised with an unknown ordinal value is encountered during normalization.

Other values, such as MPG, horsepower, weight, and acceleration are simply mapped as continuous. The automobile's name is not considered. Sometimes useful information can be extracted from string fields; however, they must always be converted to numeric in some clever way.

Finally, it is important to define how to handle missing values. First, we define that the question mark (?) is used to represent an unknown field. Next, we define how to handle the missing values on a column-by-column basis. Horsepower is the only column with missing values, and we simply substitute each missing horsepower value with the mean of all horsepower values. There are certainly more sophisticated methods to determine a missing horsepower value; however, this example it is sufficient.

The following Java code sets up the mappings to the data file.

```java
// decimal point and space separated
CSVFormat format = new CSVFormat('.',' ');
VersatileDataSource source = new CSVDataSource(
    filename, false, format);
VersatileMLDataSet data = new VersatileMLDataSet(source);
data.getNormHelper().setFormat(format);
ColumnDefinition columnMPG = data.defineSourceColumn("mpg", 0,
   ColumnType.continuous);
ColumnDefinition columnCylinders = data.defineSourceColumn("
   cylinders", 1, ColumnType.ordinal);
// It is very important to predefine ordinals,
// so that the order is known.
columnCylinders.defineClass(new String[] {"3","4","5","6","8"});
```

```
data.defineSourceColumn("displacement", 2,ColumnType.continuous);
ColumnDefinition columnHorsePower =
    data.defineSourceColumn(
    "horsepower", 3, ColumnType.continuous);
data.defineSourceColumn("weight", 4, ColumnType.continuous);
data.defineSourceColumn(
    "acceleration", 5,ColumnType.continuous);
ColumnDefinition columnModelYear =
data.defineSourceColumn("model_year", 6, ColumnType.ordinal);
columnModelYear.defineClass(
new String[] {"70","71","72","73","74","75","76",
"77","78","79","80","81","82"});
data.defineSourceColumn("origin", 7, ColumnType.nominal);
// Define how missing values are represented.
data.getNormHelper().defineUnknownValue("?");
data.getNormHelper().defineMissingHandler(
    columnHorsePower, new MeanMissingHandler());
// Analyze the data, determine the min/max/mean/sd
// of every column.
data.analyze();
```

The following C# code accomplishes the same thing.

```
// Download the data that we will attempt to model.
string filename = DownloadData(app.Args);
// Define the format of the data file.
// This area will change, depending on the columns and
// format of the file that you are trying to model.
var format = new CSVFormat('.', ' ');
// decimal point and space separated
IVersatileDataSource source = new CSVDataSource(
    filename, false, format);
var data = new VersatileMLDataSet(source);
data.NormHelper.Format = format;
ColumnDefinition columnMPG = data.DefineSourceColumn(
    "mpg", 0, ColumnType.Continuous);

ColumnDefinition columnCylinders = data.DefineSourceColumn(
    "cylinders", 1, ColumnType.Ordinal);
// It is very important to predefine ordinals,
// so that the order is known.
columnCylinders.DefineClass(new[] {"3", "4", "5", "6", "8"});
data.DefineSourceColumn(
    "displacement", 2, ColumnType.Continuous);
```

```
ColumnDefinition columnHorsePower =
    data.DefineSourceColumn(
    "horsepower", 3, ColumnType.Continuous);
data.DefineSourceColumn(
    "weight", 4, ColumnType.Continuous);
data.DefineSourceColumn(
    "acceleration", 5, ColumnType.Continuous);
ColumnDefinition columnModelYear =
    data.DefineSourceColumn(
      "model_year", 6, ColumnType.Ordinal);
      columnModelYear.DefineClass(new[]
            {"70", "71", "72", "73", "74", "75", "76",
             "77", "78", "79", "80", "81", "82"});
      data.DefineSourceColumn("origin", 7, ColumnType.Nominal);
// Define how missing values are represented.
data.NormHelper.DefineUnknownValue("?");
            data.NormHelper.DefineMissingHandler(columnHorsePower,
                new MeanMissingHandler());
// Analyze the data, determine the min/max/mean/sd
// of every column.
data.Analyze();
```

The final step is to call the **Analyze** method. This reads the entire file and determines the minimum, maximum, mean and standard deviations for each column. These statistics will be useful for both normalization and interpolation of missing values. Fortunately, the iris data set has no missing values.

2.2.2 Specifying the Model & Normalizing

Before we can normalize the data, we must choose our desired model type. The model type often dictates how the data should be normalized. For this example, I will use a feedforward neural network. We must also specify the column that we are going to predict. In this case, we are predicting the mpg value. Because the MPG value is numeric, this is a regression problem. Performing a classification problem is simply a matter of choosing to predict a non-numeric column, as we did in the last section.

We also choose to send all output to the console. Now that everything is set we can normalize. The normalization process will load the CSV file into memory and normalize the data as it is loaded.

The following Java code accomplishes this.

```
// Map the prediction column to the output of the model, and all
// other columns to the input.
data.defineSingleOutputOthersInput(columnMPG);
EncogModel model = new EncogModel(data);
model.selectMethod(data, MLMethodFactory.TYPE_FEEDFORWARD);
// Send any output to the console.
model.setReport(new ConsoleStatusReportable());
// Now normalize the data. Encog will automatically
// determine the correct normalization type based
// on the model you chose in the last step.
data.normalize();
```

The following C# code accomplishes the same thing.

```
// Map the prediction column to the output of the model, and all
// other columns to the input.
data.DefineSingleOutputOthersInput(columnMPG);
var model = new EncogModel(data);
model.SelectMethod(data, MLMethodFactory.TypeFeedforward);
// Send any output to the console.
model.Report = new ConsoleStatusReportable();
// Now normalize the data. Encog will automatically
// determine the correct normalization
// type based on the model you chose in the last step.
data.Normalize();
```

2.2.3 Fitting the Model

Before we fit the model we hold back part of the data for a validation set. We choose to hold back 30%. We chose to randomize the data set with a fixed seed value. This fixed seed ensures that we get the same training and validation sets each time. This is a matter of preference. If you want a random sample each time then pass in the current time for the seed. Finally, we fit the model with a k-fold cross-validation of size 5.

The following Java code accomplishes this.

```
model.holdBackValidation(0.3, true, 1001);
model.selectTrainingType(data);
```

```
MLRegression bestMethod = (MLRegression)model.crossvalidate(5,
   true);
```

The following C# code accomplishes the same.

```
model.HoldBackValidation(0.3, true, 1001);
model.SelectTrainingType(data);
var bestMethod = (IMLRegression) model.Crossvalidate(5, true);
```

Cross-validation breaks the training dataset into 5 different combinations of training and validation data. Do not confuse the cross-validation validation data with the ultimate validation data that we set aside previously. The cross-validation process does not use the validation data that we previously set aside. Those data are for a final validation, after training has occurred.

At the end of the cross-validation training you will obtain the best model of the 5 folds. You will also see the cross-validated error. This error is an average of the validation errors of the five folds. The cross-validation error is an estimate how your model might perform on data that it was not trained on.

2.2.4 Displaying the Results

We can now display several of the errors. We can check the training error and validation errors. We can also display the stats gathered on the data.

The following Java code accomplishes this.

```
// Display the training and validation errors.
System.out.println( "Training error: "
    + model.calculateError(bestMethod, model.getTrainingDataset())
        );
System.out.println( "Validation error: "
    + model.calculateError(bestMethod, model.getValidationDataset
        ()));
// Display our normalization parameters.
NormalizationHelper helper = data.getNormHelper();
System.out.println(helper.toString());
// Display the final model.
System.out.println("Final model: " + bestMethod);
```

The following C# code accomplishes the same.

```
// Display the training and validation errors.
Console.WriteLine(@"Training error: "
    + model.CalculateError(bestMethod, model.TrainingDataset));
Console.WriteLine(@"Validation error: "
    + model.CalculateError(bestMethod, model.ValidationDataset));
// Display our normalization parameters.
NormalizationHelper helper = data.NormHelper;
Console.WriteLine(helper.ToString());
// Display the final model.
Console.WriteLine("Final model: " + bestMethod);
```

2.2.5 Using the Model & Denormalizing

Once you've trained a model you will likely want to use the model. The best model can be saved using normal serialization. However, you will need a way to normalize data going into the model, and denormalize data coming out of the mode. The normalization helper object, obtained in the previous section, can do this for you. You can also serialize the normalization helper.

The following Java code opens the CSV file and predicts each car's MPG using the best model and normalization helper.

```
ReadCSV csv = new ReadCSV(filename, false, format);
String[] line = new String[7];
MLData input = helper.allocateInputVector();
while(csv.next()) {
    StringBuilder result = new StringBuilder();
    line[0] = csv.get(1);
    line[1] = csv.get(2);
    line[2] = csv.get(3);
    line[3] = csv.get(4);
    line[4] = csv.get(5);
    line[5] = csv.get(6);
    line[6] = csv.get(7);
    String correct = csv.get(0);

    helper.normalizeInputVector(line,input.getData(),false);
    MLData output = bestMethod.compute(input);
    String irisChosen =
       helper.denormalizeOutputVectorToString(output)[0];
    result.append(Arrays.toString(line));
    result.append(" -> predicted: ");
    result.append(irisChosen);
    result.append("(correct: ");
    result.append(correct);
    result.append(")");
    System.out.println(result.toString());
}
```

The following C# code accomplishes the same thing.

```
var csv = new ReadCSV(filename, false, format);
var line = new String[7];
IMLData input = helper.AllocateInputVector();
while (csv.Next())
{
    var result = new StringBuilder();
    line[0] = csv.Get(1);
    line[1] = csv.Get(2);
    line[2] = csv.Get(3);
    line[3] = csv.Get(4);
    line[4] = csv.Get(5);
    line[5] = csv.Get(6);
    line[6] = csv.Get(7);
    String correct = csv.Get(0);
    helper.NormalizeInputVector(line,
        ((BasicMLData) input).Data, false);
    IMLData output = bestMethod.Compute(input);
    String irisChosen =
         helper.DenormalizeOutputVectorToString(output)[0];
    result.Append(line);
    result.Append(" -> predicted: ");
    result.Append(irisChosen);
    result.Append("(correct: ");
    result.Append(correct);
    result.Append(")");
    Console.WriteLine(result.ToString());
}
csv.Close();
```

The output from this program will look similar to the following. First the program downloads the data set and begins training. Training occurs over 5 folds. Each fold uses a separate portion of the training data as validation. The remaining portion of the training data is used to train the model for that fold. Each fold gives us a different model; we choose the model with the best validation score. We train until the validation score ceases to improve. This helps to prevent over-fitting. The first fold trains for 60 iterations before it stops:

```
Downloading auto-mpg dataset to: /var/folders/m5/
   gbcvpwzj7gjdb41z1_x9rzch0000gn/T/auto-mpg.data
1/5 : Fold #1
1/5 : Fold #1/5: Iteration #1, Training Error: 1.58741311,
   Validation Error: 1.38996414
1/5 : Fold #1/5: Iteration #2, Training Error: 1.48792340,
   Validation Error: 1.38996414
1/5 : Fold #1/5: Iteration #3, Training Error: 1.45292108,
   Validation Error: 1.38996414
1/5 : Fold #1/5: Iteration #4, Training Error: 1.25876413,
   Validation Error: 1.38996414
1/5 : Fold #1/5: Iteration #5, Training Error: 1.10317339,
   Validation Error: 1.38996414
...
1/5 : Fold #1/5: Iteration #60, Training Error: 0.01503148,
   Validation Error: 0.02394547
```

The first fold stopped with a validation error of 0.02. The second fold continues.

```
2/5 : Fold #2
2/5 : Fold #2/5: Iteration #1, Training Error: 0.41743768,
   Validation Error: 0.38868284
2/5 : Fold #2/5: Iteration #2, Training Error: 0.29303614,
   Validation Error: 0.38868284
2/5 : Fold #2/5: Iteration #3, Training Error: 0.23245726,
   Validation Error: 0.38868284
2/5 : Fold #2/5: Iteration #4, Training Error: 0.23780972,
   Validation Error: 0.38868284
2/5 : Fold #2/5: Iteration #5, Training Error: 0.12788026,
   Validation Error: 0.38868284
2/5 : Fold #2/5: Iteration #6, Training Error: 0.10327476,
   Validation Error: 0.06406355
2/5 : Fold #2/5: Iteration #7, Training Error: 0.06530528,
   Validation Error: 0.06406355
2/5 : Fold #2/5: Iteration #8, Training Error: 0.07534470,
   Validation Error: 0.06406355
...
```

```
2/5 : Fold #2/5: Iteration #60, Training Error: 0.01872822,
    Validation Error: 0.02699839
```

The second fold stops with a validation error of 0.02. It is important to note that that the folds are independent of each other. Each fold starts with a new model.

```
3/5 : Fold #3
3/5 : Fold #3/5: Iteration #1, Training Error: 0.51587682,
    Validation Error: 0.62952953
3/5 : Fold #3/5: Iteration #2, Training Error: 0.40655151,
    Validation Error: 0.62952953
3/5 : Fold #3/5: Iteration #3, Training Error: 0.39780736,
    Validation Error: 0.62952953
3/5 : Fold #3/5: Iteration #4, Training Error: 0.29733447,
    Validation Error: 0.62952953
3/5 : Fold #3/5: Iteration #5, Training Error: 0.29933895,
    Validation Error: 0.62952953
...
3/5 : Fold #3/5: Iteration #90, Training Error: 0.01364865,
    Validation Error: 0.02184541
4/5 : Fold #4
4/5 : Fold #4/5: Iteration #1, Training Error: 0.66926738,
    Validation Error: 0.71307852
4/5 : Fold #4/5: Iteration #2, Training Error: 0.44893095,
    Validation Error: 0.71307852
4/5 : Fold #4/5: Iteration #3, Training Error: 0.55186651,
    Validation Error: 0.71307852
4/5 : Fold #4/5: Iteration #4, Training Error: 0.53754145,
    Validation Error: 0.71307852
4/5 : Fold #4/5: Iteration #5, Training Error: 0.23648463,
    Validation Error: 0.71307852
...
4/5 : Fold #4/5: Iteration #108, Training Error: 0.01597952,
    Validation Error: 0.01835486
5/5 : Fold #5
5/5 : Fold #5/5: Iteration #1, Training Error: 1.43940573,
    Validation Error: 1.36648367
5/5 : Fold #5/5: Iteration #2, Training Error: 0.57334529,
    Validation Error: 1.36648367
5/5 : Fold #5/5: Iteration #3, Training Error: 0.65765025,
    Validation Error: 1.36648367
5/5 : Fold #5/5: Iteration #4, Training Error: 0.42384536,
    Validation Error: 1.36648367
5/5 : Fold #5/5: Iteration #5, Training Error: 0.40821277,
    Validation Error: 1.36648367
...
5/5 : Fold #5/5: Iteration #54, Training Error: 0.01579053,
    Validation Error: 0.02912145
```

After fold 5 is complete, we report the cross-validated score that is the average of all 5 validation scores. This should give us a reasonable estimate of how well the model might perform on data that it was not trained with. Using the best model, from the 5 folds, we now evaluate it with the training data and the true validation data that we set aside earlier.

```
5/5 : Cross-validated score:0.02405311775325248
Training error: 0.016437770234365972
Validation error: 0.022529531723353303
```

As you can see, the training error is lower than the validation error. This is normal, as models always tend to perform better on data that they were trained with. However, it is important to note that the validation error is close to the cross-validated error. The cross-validated error will often give us a good estimate of how our model will perform on untrained data.

Finally, we display the normalization data. This shows us the min, max, mean and standard deviation for each column.

```
[NormalizationHelper:
[ColumnDefinition:mpg(continuous);low=9.000000,high=46.600000,mean
    =23.514573,sd=7.806159]
[ColumnDefinition:cylinders(ordinal);[3, 4, 5, 6, 8]]
[ColumnDefinition:displacement(continuous);low=68.000000,high
    =455.000000,mean=193.425879,sd=104.138764]
[ColumnDefinition:horsepower(continuous);low=?,high=?,mean=?,sd=?]
[ColumnDefinition:weight(continuous);low=1,613.000000,high
    =5,140.000000,mean=2,970.424623,sd=845.777234]
[ColumnDefinition:acceleration(continuous);low=8.000000,high
    =24.800000,mean=15.568090,sd=2.754222]
[ColumnDefinition:model_year(ordinal);[70, 71, 72, 73, 74, 75, 76,
     77, 78, 79, 80, 81, 82]]
[ColumnDefinition:origin(nominal);[1, 3, 2]]
]
Final model: [BasicNetwork: Layers=3]
```

Finally, we loop over the entire dataset and display predictions. This part of the example shows you how to use the model with new data you might acquire. However, for new data, you might not have the correct outcome, as that is what you seek to predict.

```
[8, 307.0, 130.0, 3504., 12.0, 70, 1] -> predicted:
    14.435441733777008(correct: 18.0)
[8, 350.0, 165.0, 3693., 11.5, 70, 1] -> predicted:
    13.454496578812098(correct: 15.0)
[8, 318.0, 150.0, 3436., 11.0, 70, 1] -> predicted:
    14.388722851782898(correct: 18.0)
[8, 304.0, 150.0, 3433., 12.0, 70, 1] -> predicted:
    14.72605875261915(correct: 16.0)
[8, 302.0, 140.0, 3449., 10.5, 70, 1] -> predicted:
    14.418818543779944(correct: 17.0)
[8, 429.0, 198.0, 4341., 10.0, 70, 1] -> predicted:
    12.399521136402008(correct: 15.0)
[8, 454.0, 220.0, 4354., 9.0, 70, 1] -> predicted:
    12.518569151158149(correct: 14.0)
[8, 440.0, 215.0, 4312., 8.5, 70, 1] -> predicted:
    12.555365172162254(correct: 14.0)
[8, 455.0, 225.0, 4425., 10.0, 70, 1] -> predicted:
    12.388570799526281(correct: 14.0)
[8, 390.0, 190.0, 3850., 8.5, 70, 1] -> predicted:
    12.969680895760376(correct: 15.0)
[8, 383.0, 170.0, 3563., 10.0, 70, 1] -> predicted:
    13.504299010941919(correct: 15.0)
[8, 340.0, 160.0, 3609., 8.0, 70, 1] -> predicted:
    13.47743472814497(correct: 14.0)
[8, 400.0, 150.0, 3761., 9.5, 70, 1] -> predicted:
    13.076737534131402(correct: 15.0)
[8, 455.0, 225.0, 3086., 10.0, 70, 1] -> predicted:
    14.54484159281664(correct: 14.0)
[4, 113.0, 95.00, 2372., 15.0, 70, 3] -> predicted:
    24.169018638449415(correct: 24.0)
...
```

2.3 Using Encog for Time Series

Time series problems can be either regression or classification. The difference is that we will now use several rows of data to make a prediction. The example that we will see here will use regression to predict the number of sunspots present using past data. The data set that we will use comes from NASA, and can be found at the following URL:

`http://solarscience.msfc.nasa.gov/greenwch/spot_num.txt`

A sampling of the data are shown here.

```
YEAR MON   SSN    DEV
1749   1   58.0   24.1
1749   2   62.6   25.1
1749   3   70.0   26.6
1749   4   55.7   23.6
1749   5   85.0   29.4
1749   6   83.5   29.2
1749   7   94.8   31.1
1749   8   66.3   25.9
1749   9   75.9   27.7
1749  10   75.5   27.7
1749  11  158.6   40.6
1749  12   85.2   29.5
1750   1   73.3   27.3
1750   2   75.9   27.7
1750   3   89.2   30.2
1750   4   88.3   30.0
```

We will create a program that generates a model to predict the sunspots for a month, based on previous values. This program will allow us to easily change the model type to any of the following:

- Feedforward Neural Network
- NEAT Neural Network
- Probabilistic Neural Network
- RBF Neural Network
- Support Vector Machine

When you change the model type, Encog will automatically change the way that the data are normalized.

This program will split the training data into a training and validation set. The validation set will be held until the end to see how well we can predict data that the model was not trained on. Training will be performed using a 5-fold cross-validation.

This complete example can be found with the Encog examples. The Java version contains this example here.

```
org.encog.examples.guide.timeseries.SunSpotTimeseries
```

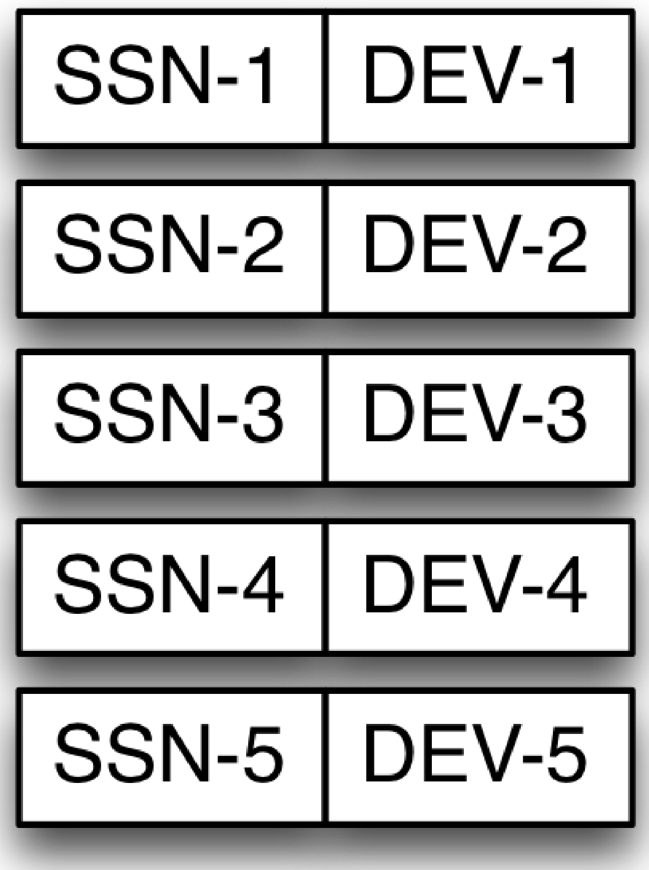
The C# version can be executed with the argument **guide-sunspots**, and can be found at the following location.

```
Encog.Examples.Guide.Timeseries.SunSpotTimeseries
```

2.3.1 Mapping the Input File

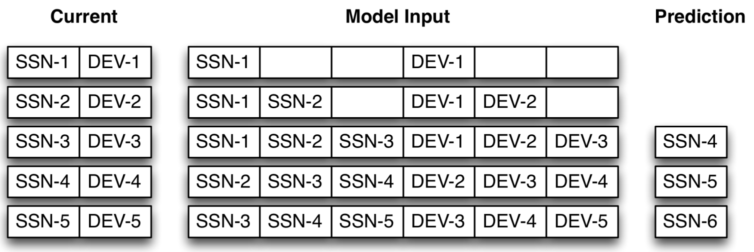
We begin by defining a **VersatileMLDataSet** object that will read from a CSV file. We define the sunspot number (**SSN**) and **DEV** columns columns of the data file. We will ignore the **YEAR** and **MON** fields. The file, downloaded from NASA contains column headers. Because of this we can refer to the columns by their names. We must also account for the fact that this is a time series. We do not want to simply use **SSN** and **DEV** for a given day to determine the **SSN** for the next day. Rather, we must use a series of days. For this example we use the last three days. There are many different ways to consider a number of days. This example uses a sliding window. To see how this works, consider Figure 2.1 that shows the sunspot data.

Figure 2.1: Sunspot Data for 5 Months



The **VersatileMLDataSet** allows you to specify a lead and a lag for time-boxing. We are using a lag of 3, and a lead of 1. This means that we will use the last three **SSN** and **DEV** values to predict the next one. It takes a few months to build up the lag. Because of this we are cannot use the first two months to generate a prediction. Figure 2.2 shows how the time-box is built up.

Figure 2.2: Sunspot Data Timeboxed (lead:1, Lag:3)



You can also specify a lead value and predict further into the future than just one unit. Not all model types support this. A model type must support multiple outputs to predict further into the future than one unit. Neural networks are a good choice for multiple outputs; however, models such as support vector machines do not.

If you would like to predict further into the future than one unit, there are ways of doing this without multiple outputs. You can use your predicted value as part of the lag values and extrapolate as far into the future as you wish. The following shows you how the numbers 1 through 10 would look with different lead and lag values.

```
Lag 0; Lead 0 [10 rows] 1->1 2->2 3->3 4->4 5->5 6->6 7->7 8->8
    9->9 10->10
Lag 0; Lead 1 [9 rows] 1->2 2->3 3->4 4->5 5->6 6->7 7->8 8->9
    9->10
Lag 1; Lead 0 [9 rows, not useful] 1,2->1 2,3->2 3,4->3 4,5->4
    5,6->5 6,7->6
7,8->7 8,9->8 9,10->9

Lag 1; Lead 1 [8 rows] 1,2->3 2,3->4 3,4->5 4,5->6 5,6->7 6,7->8
    7,8->9
8,9->10
Lag 1; Lead 2 [7 rows] 1,2->3,4 2,3->4,5 3,4->5,6 4,5->6,7
    5,6->7,8 6,7->8,9
7,8->9,10

Lag 2; Lead 1 [7 rows] 1,2,3->4 2,3,4->5 3,4,5->6 4,5,6->7
    5,6,7->8 6,7,8->9
7,8,9->10
```

We will now look at how to map the sunspot data to a **VersatileMLDataSet**. The following Java code accomplishes this data mapping.

```
File filename = downloadData(args);
// Define the format of the data file.
// This area will change, depending on the columns and
// format of the file that you are trying to model.
CSVFormat format = new CSVFormat('.', ' '); // decimal point and
// space separated
VersatileDataSource source = new CSVDataSource(
    filename, true, format);
VersatileMLDataSet data = new VersatileMLDataSet(source);
data.getNormHelper().setFormat(format);
ColumnDefinition columnSSN = data.defineSourceColumn("SSN",
    ColumnType.continuous);
ColumnDefinition columnDEV = data.defineSourceColumn("DEV",
    ColumnType.continuous);
// Analyze the data, determine the min/max/mean/sd
// of every column.
data.analyze();
```

The following C# code accomplishes the same thing.

```
string filename = DownloadData(app.Args);
// Define the format of the data file.
// This area will change, depending on the columns and
// format of the file that you are trying to model.
var format = new CSVFormat('.', ' '); // decimal point and
// space separated
IVersatileDataSource source = new CSVDataSource(filename, true,
    format);
var data = new VersatileMLDataSet(source);
data.NormHelper.Format = format;
ColumnDefinition columnSSN = data.DefineSourceColumn("SSN",
    ColumnType.Continuous);
ColumnDefinition columnDEV = data.DefineSourceColumn("DEV",
    ColumnType.Continuous);
```

2.3.2 Specifying the Model & Normalizing

Before we can normalize the data, we must choose our desired model type. The model type often dictates how the data should be normalized. For this example, I will use a feedforward neural network. We must also specify the column that we are going to predict. In this case, we are predicting the **SSN** value. Because the **SSN** value is numeric, this is a regression problem. Performing a classification problem is simply a matter of choosing to predict a non-numeric column, as we did in the last section.

We also choose to send all output to the console. Now that everything is set we can normalize. The normalization process will load the CSV file into memory and normalize the data as it is loaded.

The following Java code accomplishes this.

```
// Map the prediction column to the output of the model, and all
// other columns to the input.
data.defineSingleOutputOthersInput(columnMPG);
EncogModel model = new EncogModel(data);
model.selectMethod(data, MLMethodFactory.TYPE_FEEDFORWARD);
// Send any output to the console.
model.setReport(new ConsoleStatusReportable());
// Now normalize the data. Encog will automatically
// determine the correct normalization type based
// on the model you chose in the last step.
data.normalize();
```

The following C# code accomplishes the same thing.

```
// Map the prediction column to the output of the model, and all
// other columns to the input.
data.DefineSingleOutputOthersInput(columnMPG);
var model = new EncogModel(data);
model.SelectMethod(data, MLMethodFactory.TypeFeedforward);
// Send any output to the console.
model.Report = new ConsoleStatusReportable();
// Now normalize the data. Encog will automatically
// determine the correct normalization
// type based on the model you chose in the last step.
data.Normalize();
```

2.3.3 Fitting the Model

Before we fit the model we hold back part of the data for a validation set. We choose to hold back 30%. We chose to randomize the data set with a fixed seed value. This fixed seed ensures that we get the same training and validation sets each time. This is a matter of preference. If you want a random sample each time then pass in the current time for the seed. We also establish the lead and lag window sizes. Finally, we fit the model with a k-fold cross-validation of size 5.

The following Java code accomplishes this.

```
// Set time series.
data.setLeadWindowSize(1);
data.setLagWindowSize(WINDOW_SIZE);
// Hold back some data for a final validation.
// Do not shuffle the data into a random ordering.
// (never shuffle time series)
// Use a seed of 1001 so that we always use the same
// holdback and will get more consistent results.
model.holdBackValidation(0.3, false, 1001);
// Choose whatever is the default training type for this model.
model.selectTrainingType(data);
// Use a 5-fold cross-validated train. Return the
// best method found.
// (never shuffle time series)
MLRegression bestMethod =
  (MLRegression) model.crossvalidate(5, false);
model.holdBackValidation(0.3, true, 1001);
model.selectTrainingType(data);
MLRegression bestMethod =
  (MLRegression)model.crossvalidate(5, true);
```

The following C# code accomplishes the same.

```
// Set time series.
data.LeadWindowSize = 1;
data.LagWindowSize = WindowSize;
// Hold back some data for a final validation.
// Do not shuffle the data into a random ordering. (never shuffle
// time series)
// Use a seed of 1001 so that we always use the same holdback and
// will get more consistent results.
model.HoldBackValidation(0.3, false, 1001);
// Choose whatever is the default training type for this model.
model.SelectTrainingType(data);
// Use a 5-fold cross-validated train. Return the best
// method found. (never shuffle time series)
```

```
var bestMethod = (IMLRegression) model.Crossvalidate(
    5, false);
model.HoldBackValidation(0.3, true, 1001);
model.SelectTrainingType(data);
var bestMethod = (IMLRegression) model.Crossvalidate(5, true);
```

Cross-validation breaks the training dataset into 5 different combinations of training and validation data. Do not confuse the cross-validation validation data with the ultimate validation data that we set aside previously. The cross-validation process does not use the validation data that we previously set aside. Those data are for a final validation, after training has occurred.

At the end of the cross-validation training you will obtain the best model of the 5 folds. You will also see the cross-validated error. This error is an average of the validation errors of the five folds. The cross-validation error is an estimate how your model might perform on data that it was not trained on.

2.3.4 Displaying the Results

We can now display several of the errors. We can check the training error and validation errors. We can also display the stats gathered on the data.

The following Java code accomplishes this.

```
// Display the training and validation errors.
System.out.println( "Training error: "
    + model.calculateError(bestMethod, model.getTrainingDataset())
        );
System.out.println( "Validation error: "
    + model.calculateError(bestMethod, model.getValidationDataset
        ()));
// Display our normalization parameters.
NormalizationHelper helper = data.getNormHelper();
System.out.println(helper.toString());
// Display the final model.
System.out.println("Final model: " + bestMethod);
```

The following C# code accomplishes the same.

```
// Hold back some data for a final validation.
// Display the training and validation errors.
Console.WriteLine(@"Training error: "
    + model.CalculateError(bestMethod, model.TrainingDataset));
Console.WriteLine(@"Validation error: "
    + model.CalculateError(bestMethod, model.ValidationDataset));
// Display our normalization parameters.
NormalizationHelper helper = data.NormHelper;
Console.WriteLine(helper.ToString());
// Display the final model.
Console.WriteLine("Final model: " + bestMethod);
```

2.3.5 Using the Model & Denormalizing

Once you've trained a model you will likely want to use the model. The best model can be saved using normal serialization. However, you will need a way to normalize data going into the model, and denormalize data coming out of the mode. The normalization helper object, obtained in the previous section, can do this for you. You can also serialize the normalization helper.

The following Java code opens the CSV file and predicts each month's sunspot number (SSN) using the best model and normalization helper.

```
ReadCSV csv = new ReadCSV(filename, true, format);
String[] line = new String[2];
// Create a vector to hold each time-slice, as we build them.
// These will be grouped together into windows.
double[] slice = new double[2];
VectorWindow window = new VectorWindow(WINDOW_SIZE + 1);
MLData input = helper.allocateInputVector(WINDOW_SIZE + 1);
// Only display the first 100
int stopAfter = 100;
while (csv.next() && stopAfter > 0) {
    StringBuilder result = new StringBuilder();
    line[0] = csv.get(2);// ssn
    line[1] = csv.get(3);// dev
    helper.normalizeInputVector(line, slice, false);
    // enough data to build a full window?
    if (window.isReady()) {
        window.copyWindow(input.getData(), 0);
        String correct = csv.get(2);
```

```
            // trying to predict SSN.
            MLData output = bestMethod.compute(input);
            String predicted = helper
                .denormalizeOutputVectorToString(output)[0];
            result.append(Arrays.toString(line));
            result.append(" -> predicted: ");
            result.append(predicted);
            result.append("(correct: ");
            result.append(correct);
            result.append(")");
            System.out.println(result.toString());
        }
// Add the normalized slice to the window. We do this just after
// the after checking to see if the window is ready so that the
// window is always one behind the current row. This is because
// we are trying to predict next row.
    window.add(slice);
    stopAfter--;
}
```

The following C# code accomplishes the same thing.

```
var csv = new ReadCSV(filename, true, format);
var line = new String[2];
// Create a vector to hold each time-slice, as we build them.
// These will be grouped together into windows.
var slice = new double[2];
var window = new VectorWindow(WindowSize + 1);
IMLData input = helper.AllocateInputVector(WindowSize + 1);
// Only display the first 100
int stopAfter = 100;
while (csv.Next() && stopAfter > 0)
{
    var result = new StringBuilder();
    line[0] = csv.Get(2); // ssn
    line[1] = csv.Get(3); // dev
    helper.NormalizeInputVector(line, slice, false);
    // enough data to build a full window?
    if (window.IsReady())
    {
       window.CopyWindow(((BasicMLData) input).Data, 0);
       String correct = csv.Get(2); // trying to predict SSN.
       IMLData output = bestMethod.Compute(input);
       String predicted = helper
```

```
          .DenormalizeOutputVectorToString(output)[0];
        result.Append(line);
        result.Append(" -> predicted: ");
        result.Append(predicted);
        result.Append("(correct: ");
        result.Append(correct);
        result.Append(")");
        Console.WriteLine(result.ToString());
                    }
// Add the normalized slice to the window. We do this just after
// the after checking to see if the window is ready so that the
// window is always one behind the current row. This is because
// we are trying to predict next row.
        window.Add(slice);
        stopAfter--;
}
```

The output from this program will look similar to the following. First the program downloads the data set and begins training. Training occurs over 5 folds. Each fold uses a separate portion of the training data as validation. The remaining portion of the training data is used to train the model for that fold. Each fold gives us a different model; we choose the model with the best validation score. We train until the validation score ceases to improve. This helps to prevent over-fitting. The first fold trains for 24 iterations before it stops:

```
Downloading sunspot dataset to: /var/folders/m5/
   gbcvpwzj7gjdb41z1_x9rzch0000gn/T/auto-mpg.data
1/5 : Fold #1
1/5 : Fold #1/5: Iteration #1, Training Error: 1.09902944,
   Validation Error: 1.02673263
1/5 : Fold #1/5: Iteration #2, Training Error: 0.64352979,
   Validation Error: 1.02673263
1/5 : Fold #1/5: Iteration #3, Training Error: 0.22823721,
   Validation Error: 1.02673263
1/5 : Fold #1/5: Iteration #4, Training Error: 0.27106762,
   Validation Error: 1.02673263
...
1/5 : Fold #1/5: Iteration #24, Training Error: 0.08642049,
   Validation Error: 0.06355912
```

The first fold gets a validation error of 0.06 and continues into the second fold.

```
2/5 : Fold #2
2/5 : Fold #2/5: Iteration #1, Training Error: 0.81229781,
   Validation Error: 0.91492569
2/5 : Fold #2/5: Iteration #2, Training Error: 0.31978710,
   Validation Error: 0.91492569
...
2/5 : Fold #2/5: Iteration #30, Training Error: 0.11828392,
   Validation Error: 0.13355361
```

The second fold gets a validation error of 0.13 and continues on to the third fold. It is important to note that that the folds are independent of each other. Each fold starts with a new model.

```
3/5 : Fold #3
3/5 : Fold #3/5: Iteration #1, Training Error: 1.42311914,
   Validation Error: 1.36189059
3/5 : Fold #3/5: Iteration #2, Training Error: 0.97598935,
   Validation Error: 1.36189059
3/5 : Fold #3/5: Iteration #3, Training Error: 0.26472233,
   Validation Error: 1.36189059
3/5 : Fold #3/5: Iteration #4, Training Error: 0.26861918,
   Validation Error: 1.36189059
3/5 : Fold #3/5: Iteration #5, Training Error: 0.26472233,
   Validation Error: 1.36189059
...
3/5 : Fold #3/5: Iteration #126, Training Error: 0.04777174,
   Validation Error: 0.04556459
```

The third fold gets a validation error of 0.045 and continues on to the fourth fold.

```
4/5 : Fold #4
4/5 : Fold #4/5: Iteration #1, Training Error: 0.43642221,
   Validation Error: 0.41741128
4/5 : Fold #4/5: Iteration #2, Training Error: 0.26367259,
   Validation Error: 0.41741128
4/5 : Fold #4/5: Iteration #3, Training Error: 0.25940789,
   Validation Error: 0.41741128
4/5 : Fold #4/5: Iteration #4, Training Error: 0.20787347,
   Validation Error: 0.41741128
4/5 : Fold #4/5: Iteration #5, Training Error: 0.18484274,
   Validation Error: 0.41741128
...
```

```
4/5 : Fold #4/5: Iteration #330, Training Error: 0.03478583,
   Validation Error: 0.03441984
```

The fourth fold gets a validation error of 0.03 and continues on to the fifth fold.

```
5/5 : Fold #5
5/5 : Fold #5/5: Iteration #1, Training Error: 1.03537886,
   Validation Error: 1.13457447
5/5 : Fold #5/5: Iteration #2, Training Error: 0.61248351,
   Validation Error: 1.13457447
5/5 : Fold #5/5: Iteration #3, Training Error: 0.35799763,
   Validation Error: 1.13457447
5/5 : Fold #5/5: Iteration #4, Training Error: 0.34937204,
   Validation Error: 1.13457447
5/5 : Fold #5/5: Iteration #5, Training Error: 0.32800730,
   Validation Error: 1.13457447
...
5/5 : Fold #5/5: Iteration #30, Training Error: 0.06560991,
   Validation Error: 0.07119405
5/5 : Cross-validated score:0.0696582424947976
Training error: 0.1342019169847873
Validation error: 0.15649156756982546
```

We now display the normalization stats on each column.

```
[NormalizationHelper:
[ColumnDefinition:SSN(continuous);low=0.000000,high=253.800000,
   mean=52.067252,sd=1,830.873430]
[ColumnDefinition:DEV(continuous);low=0.000000,high=90.200000,mean
   =20.231399,sd=14.163302]
]
Final model: [BasicNetwork: Layers=3]
```

Finally, we attempt some prediction with the new model.

```
[85.0, 29.4] -> predicted: 58.52699993534398(correct: 85.0)
[83.5, 29.2] -> predicted: 64.45005584765465(correct: 83.5)
[94.8, 31.1] -> predicted: 73.24597015866078(correct: 94.8)
[66.3, 25.9] -> predicted: 55.5113451251101(correct: 66.3)
...
```

Index