# Data Normalization

Constrain data to <0;1> or <-1;1> range

*IDLmLinearNormalizer*

*IDLmlRangeNormalizer*

*IDLmlTanHNormalizer*

*IDLmlUnitNormalizer*

*IDLmlVarianceNormalizer*

# Data Separation

Two groups of data: 1) training data set, 2) Testing data set

*IDLmlShuffle, IDLmlPartition*

**Example 1 (data preparation):**

labels = array[210] of string

features = array[7,210] of double

Normalizer = [**IDLmlVarianceNormalizer**](https://www.harrisgeospatial.com/docs/idlmlvariancenormalizer.html)(features)

Normalizer.Normalize, features

[**IDLmlShuffle**](https://www.harrisgeospatial.com/docs/idlmlshuffle.html), features, labels

part = [**IDLmlPartition**](https://www.harrisgeospatial.com/docs/idlmlpartition.html)({train:80, test:20}, features, labels)

part.train.labels = array[168] of string ;80% of original data

part.train.features = array[7,168] of double

part.test.labels = array[42] of string ;20% of original data

part.test.features = array[7,42] of double

# Classification

*IDLmlFeedForwardNeuralNetwork*

*IDLmlSoftmax*

*IDLmlSupportVectorMachineClassification*

**Example 2 (data classification using SVM):**

Following example 1, there are 7 number attributes (inputs) and 3 possible string outputs (‘Kama’, ‘Rosa’, ‘Canadian’)

Classifier = [**IDLmlSupportVectorMachineClassification**](https://www.harrisgeospatial.com/docs/idlmlsupportvectormachineclassification.html)(7, ['Kama', 'Rosa', 'Canadian'])

loss = Classifier.Train(part.train.features, labels=part.train.labels)

the goal is to minimize loss

For SVM it is enough to train only once. Other classifier require iterative training in a loop.

**Example 3 (data classification using Softmax):**

SoftmaxClassifier = [**IDLmlSoftmax**](https://www.harrisgeospatial.com/docs/idlmlsoftmax.html)(7, ['Kama', 'Rosa', 'Canadian'])

p = [**Plot**](https://www.harrisgeospatial.com/docs/plot.html)([**Fltarr**](https://www.harrisgeospatial.com/docs/fltarr.html)(2), title='Loss')

Loss = [**List**](https://www.harrisgeospatial.com/docs/list.html)()

[**for**](https://www.harrisgeospatial.com/docs/for.html) i=1, 200 **do** [**begin**](https://www.harrisgeospatial.com/docs/begin___end.html)

Loss.**Add**, SoftmaxClassifier.Train(part.train.features, $

LABELS=part.train.labels)

p.**SetData**, Loss.**ToArray**()

[**endfor**](https://www.harrisgeospatial.com/docs/begin___end.html)

**Example 4 (assessing the quality of a model):**

confMatrix = [**IDLmlTestClassifier**](https://www.harrisgeospatial.com/docs/idlmltestclassifier.html)(Classifier, $

part.test.features, part.test.labels, $

accuracy=accuracy)

[**print**](https://www.harrisgeospatial.com/docs/print.html), accuracy

**Example 5 (saving for further use):**

Normalizer.**Save**, 'mynormalizer.sav'

Classifier.**Save**, 'myclassifier.sav'

**Example 6 (restoring and actual classification):**

Normalizer = IDLmlNormalizer.**Restore**('mynormalizer.sav')

Classifier = IDLmlModel.**Restore**('myclassifier.sav')

[**print**](https://www.harrisgeospatial.com/docs/print.html), Classifier.Classify(part.test.features[\*,0])

# Clustering

*IDLmlAutoEncoder*

*IDLmlKMeans*

Input: 2D array [n x m], where n number of attributes, m number of examples

**Example 7 (clustering):**

features = array[3,10000] of double ;3 attributes, 10000 samples

Normalizer = [**IDLmlVarianceNormalizer**](https://www.harrisgeospatial.com/docs/idlmlvariancenormalizer.html)(features)

Normalizer.Normalize, features

Dedicated neural network, let’s use 2 layers.

We have 3 attributes => the size of the first & last layer must be 3.

Will cluster into 5 categories => middle layer should be 5.

Classifier = [**IDLmlAutoEncoder**](https://www.harrisgeospatial.com/docs/idlmlautoencoder.html)([3, 5, 3])

Training:

An instance of an optimizer should not be reused to train a different model.

Optimizer = [**IDLmloptGradientDescent**](https://www.harrisgeospatial.com/docs/idlmloptgradientdescent.html)(0.01)

p = [**Plot**](https://www.harrisgeospatial.com/docs/plot.html)([**Fltarr**](https://www.harrisgeospatial.com/docs/fltarr.html)(2), title='Loss')

Loss = [**List**](https://www.harrisgeospatial.com/docs/list.html)()

[**for**](https://www.harrisgeospatial.com/docs/for.html) i=1, 300 **do** [**begin**](https://www.harrisgeospatial.com/docs/begin___end.html)

Loss.**Add**, Classifier.Train(features, optimizer=Optimizer)

p.**SetData**, Loss.**ToArray**()

[**endfor**](https://www.harrisgeospatial.com/docs/begin___end.html)

Having trained it, the clustered data can be obtained as:

result = Classifier.Classify(features)

# Regression

*IDLmlFeedForwardNeuralNetwork*

*IDLmlSoftmax*

*IDLmlSupportVectorMachineRegression*

**Example 8 (Feed Forward Neural Network Regression):**

Input: 2D array [n x m], where n number of attributes, m number of samples

features = array[2,10000] of double ;2 attributes, 10000 samples

scores = array[10000] of double ;corresponding values

[**IDLmlShuffle**](https://www.harrisgeospatial.com/docs/idlmlshuffle.html), features, scores

part = [**IDLmlPartition**](https://www.harrisgeospatial.com/docs/idlmlpartition.html)({train:50, test:50}, features, scores)

Need to normalize both features and scores:

Normalizer1 = [**IDLmlVarianceNormalizer**](https://www.harrisgeospatial.com/docs/idlmlvariancenormalizer.html)(features)

Normalizer1.Normalize, features

Normalizer2 = [**IDLmlVarianceNormalizer**](https://www.harrisgeospatial.com/docs/idlmlvariancenormalizer.html)(scores)

Normalizer2.Normalize, scores

Define the neural network & how many layers to use.

The first layer must correspond to number of attributes, the last layer will be 1.

Model = [**IDLmlFeedForwardNeuralNetwork**](https://www.harrisgeospatial.com/docs/idlmlfeedforwardneuralnetwork.html)([2, 7, 7, 1], $

ACTIVATION\_FUNCTIONS=[[**IDLmlafArcTan**](https://www.harrisgeospatial.com/docs/idlmlafarctan.html)(), [**IDLmlafArcTan**](https://www.harrisgeospatial.com/docs/idlmlafarctan.html)(), $

[**IDLmlafArcTan**](https://www.harrisgeospatial.com/docs/idlmlafarctan.html)()])

Then need to be trained:

Optimizer = [**IDLmloptAdam**](https://www.harrisgeospatial.com/docs/idlmloptadam.html)(0.1)

Loss = [**List**](https://www.harrisgeospatial.com/docs/list.html)()

p = [**Plot**](https://www.harrisgeospatial.com/docs/plot.html)([**Fltarr**](https://www.harrisgeospatial.com/docs/fltarr.html)(10), title='Loss')

[**for**](https://www.harrisgeospatial.com/docs/for.html) i=1, 300 **do** [**begin**](https://www.harrisgeospatial.com/docs/begin___end.html)

Loss.**Add**, Model.Train(part.train.features, $

scores=part.train.scores, optimizer=Optimizer)

p.**SetData**, Loss.**ToArray**()

[**endfor**](https://www.harrisgeospatial.com/docs/begin___end.html)

An instance of an optimizer should not be reused to train a different model.

test\_data = array[2,10] of double ;where we want to evaluate the model

Normalizer1.Normalize, test\_data

result = Model.Evaluate(test\_data)

Evaluate the accuracy against the test data:

!null = Model.Evaluate(part.test.features, scores=part.test.scores, $

loss=loss)

[**print**](https://www.harrisgeospatial.com/docs/print.html), loss

Saving and loading normalizer & model:

Normalizer1.**Save**, normalizer.sav'

Model.**Save**, model.sav'

Normalizer = IDLmlNormalizer.**Restore**('normalizer.sav')

Model = IDLmlModel.**Restore**('model.sav')