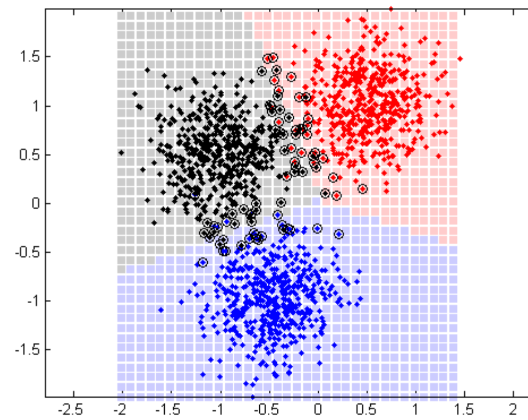
Documentation of the OPENTURNS SVM module



Abstract

The purpose of this document is to present the OpenTURNS SVM module, which enables to realize support vector regression and classification with Libsvm and to manipualte results with OpenTURNS.
It offers to the user the ability to:

- define a regression or a classification problem and solve it with Libsvm
- build OpenTURNS objects from Libsvm models
- extract metamodels of Libsvm as OpenTURNS metamodels

This document is organised according to the Open TURNS documentation :

- a *Reference Guide which gives some theoretical basis on support vector regression.*
- *a Use cases Guide which details scripts in python (the Textual Interface langage of Open TURNS) and helps the User to learn as quickly as possible the manipulation of the otsvm module.*
- *the User Manual which details the otsvm objects and give the list of their methods.*

Contents

1 Reference Guide **3**
- 1.1 Introduction to support vector machine . . . 3
- 1.2 Linear SVM . . . 3
  - 1.2.1 Primal form . . . 4
  - 1.2.2 Dual form . . . 4
- 1.3 Soft margin . . . 5
- 1.4 Nonlinear SVM . . . 5
- 1.5 Classification . . . 6
- 1.6 Regression . . . 6

2 Use Cases Guide **8**
- 2.1 Which python modules to import ? . . . 8
- 2.2 UC : creation of a SVM regression algorithm . . . 8
- 2.3 UC : Creation of a Classification algorithm . . . 11

3 User Manual **13**
- 3.1 LibSVMRegression . . . 13
- 3.2 LibSVMClassification . . . 14
- 3.3 Various . . . 16

©2005-2012 Phimeca Engineering

1 Reference Guide

The Libsvm library provides efficient algorithms to produce a model by Support Vector Machine.

1.1 Introduction to support vector machine

A support vector machine is a concept in statistics and computer science for a set of related supervised learning methods that analyze data and recognize patterns, used for classification and regression analysis. The standard SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the input. Given a set of training examples, each marked as belonging to one of two categories, a SVM training algorithm builds a model that assigns new examples into one category or the other.

More formally, a SVM constructs a hyperplane in a high or infinite-dimensional space, which can be used for classification or regression. A good separation is acheived by the hyperplane that has the largest distance to the nearest training data point of any class.

Whereas the original problem may be stated in a finite dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function $K(x, y)$ selected to suit the problem.

1.2 Linear SVM

Given some training data D, a set of n points of the form

$$D \;=\; \{(x_i, y_i) \mid x_i \in \mathbb{R}^p,\; y_i \in \{-1, 1\}\}_{i=1}^{n}$$

where the y_i is either 1 or -1, indicating the class to which the point x_i belongs. Each x_i is a p-dimensional real vector. We want to find the maximum-margin hyperplane that divides the points having $y_i = 1$ from those having $y_i = -1$.
Any hyperplane can be written as the set of points x satisfying

$$w \cdot x - b = 0$$

where $\cdot$ denotes the dot product and w the normal vector to the hyperplane. We want to choose the w and b to maximize the margin, or distance between the parallel hyperplanes that are as far apart as possible while still separating the data. These hyperplanes can be described by the equations

$$w \cdot x - b = 1$$

$$and$$

$$w \cdot x - b = -1$$

The distance between these two hyperplanes is $\frac{2}{||w||}$, so we want to minimize $||w||$. As we also have to prevent data points from falling into the margin, we add the following constraint : for each i either

$$y_i(w \cdot x_i - b) \geq 1 \quad for\ all\ 1 \leq i \leq n \tag{1}$$

©2005-2012 Phimeca Engineering

So we get the optimization problem :

$$Min\ ||w||$$
$$subject\ to\ (for\ any\ i = 1, ..., n)$$
$$y_i(w \cdot x_i - b) \geq 1$$

1.2.1 Primal form

The optimization problem presented in the preceding section is difficult to solve because it depends on $||w||$, which involves a square root. It is possible to alter the equation by substituing $||w||$ with $\frac{1}{2}||w||^2$ without changing the solution. This is a quadratic programming optimization problem :

$$Min\ \frac{1}{2}||w||^2$$
$$subject\ to\ (for\ any\ i = 1, ..., n)$$
$$y_i(w \cdot x_i - b) \geq 1$$

By introducing Lagrange multipliers α, the previous constrained problem can be expressed as

$$\min_{w,b} \max_{\alpha \geq 0} \{\frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i - b) - 1]\}$$

This problem can now be solved by standard quadratic programming techniques and programs. The stationary Karush-Kuhn-Tucker condition implies that the solution can be expressed as a linear combination of the training vectors

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Only a few α_i will be greater than zero. The corresponding x_i are exactly the support vectors, which lie on the margin and satisfy $y_i(w \cdot x_i - b) = 1$.

1.2.2 Dual form

Using the fact that $||w||^2 = w \cdot w$ and substituing $w = \sum_{i=1}^{n} \alpha_i x_i y_i$, one can show that the dual of the SVM reduces to the following optimization problem :

$$Max\ L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$
$$subject\ to\ (for\ any\ i = 1, ..., n)$$
$$\alpha_i \geq 0$$

$$and\ to\ the\ constraint\ from\ the\ minimization\ in\ b$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

w can be computed thanks to the α terms :

$$w = \sum_{i} \alpha_i y_i x_i$$

©2005-2012 Phimeca Engineering

1.3 Soft margin

If there exists no hyperplane that can split the "yes" and "no" examples, the soft margin method will choose a hyperplane that splits the examples as cleanly as possible, while still maximizing the distance to the nearest cleanly split examples. The method introduces slack variables, ξ_i, which measure the degree of misclassification of the data x_i

$$y_i(w \cdot x_i - b) \geq 1 - \xi_i \quad 1 \leq i \leq n$$

The objective function is then increased by a function which penalizes non-zero ξ_i and the optimization becomes a trade off between a large margin and a small error penalty. If the penalty function is linear, the optimization problem becomes :

$$\min_{w,b,\xi}\{\frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i\}$$

subject to (for any $i = 1, ..., n$)

$$y_i(w \cdot x_i - b) \geq 1 - \xi_i \quad \xi_i \geq 0$$

This constaint with the objective of minimizing $||w||$ can be solved using Lagrange multipliers as done above. One has then to solve the problem :

$$\min_{w,b,\xi}\max_{\alpha,\beta}\{\frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(w \cdot x_i - b) - 1 + \xi_i] - \sum_{i=1}^{n}\beta_i\xi_i\}$$

with $\alpha_i, \beta_i \geq 0$

The dual form becomes :

$$\max_{\alpha_i}\{L(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j k(x_i, x_j)\}$$

subject to (for any $i = 1, ..., n$)

$$0 \leq \alpha_i \leq C$$

and

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

1.4 Nonlinear SVM

The algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high dimensional, thus though the classifier is a hyperplane in the high-dimensional feature space, it may be nonlinear in the original input space.
Some common kernels include :

- Polynomial : $k(x_i, x_j) = (x_i \cdot x_j + c)^d$
- Gaussian Radial Basis Function : $k(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$
- Hyperbolic tangent : $k(x_i, x_j) = tanh(\gamma x_i \cdot x_j + c)$

©2005-2012 Phimeca Engineering

The kernel is related to the transform $\varphi(x_i)$ by the equation $k(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$. The value w is also in the transformed space, with $w = \sum_i \alpha_i y_i \varphi(x_i)$.

The effectiveness of SVM depends on the selection of kernel, the kernel's parameters, and soft margin parameter C. A common choice is a Gaussian kernel, which has a single parameter γ. Best combination of C and γ is selected by a grid search with exponentially growing sequences of C and γ. Each combination of parameter choices is checked using cross validation, and the parameters with best cross-validation accuracy are picked. The final model, which is used for testing and for classifying data, is then trained on the whole training set using the selected parameters.

1.5 Classification

Given training vectors x_i in two classes and a vector $y \in -1, 1,$ C-SVC (the algorithm that we use for classification) solves the following dual problem :

$$min_\alpha \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

$$0 \leq \alpha_i \leq C$$

$$y^T \alpha = 0$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an l by l positive semidefinite matrix $Q_{ij} = y_i y_j K(x_i, x_j)$ and $K(x_i, x_j)$ is the kernel. The decision function is

$$sign(\sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b)$$

For some classification problems, numbers of data in different classes are unbalanced. We can use different penalty parameters in the SVM formulation, the dual of C-SVC becomes :

$$min_\alpha \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

$$0 \leq \alpha_i \leq C_+,\ if\ y_i = 1$$

$$0 \leq \alpha_i \leq C_-,\ if\ y_i = -1$$

$$y^T \alpha = 0$$

where C_+ and C_- depending on y_i and y_j and of weights that we can fix for each class.

1.6 Regression

Up to now, we presented SVM for classification. We can use too for Regression. This is similar to the nonlinear case. We replace the soft margin by a ε-insensitive loss function which is defined like:

$$|y - f(x)|_\varepsilon = max(0, |y - f(x)| - \varepsilon)$$

where f(x) is the loss function and ε a precision parameter.
We get this optimization problem if we introduce the slack variables $\xi\ and\ \xi_i$:

$$min_w \{ \frac{||w||^2}{2} + C \sum_{i=1}^{n} (\xi_i + \xi_i^*) \}$$

©2005-2012 Phimeca Engineering

$$\text{subject to (for any } i = 1, ..., n)$$

$$l_i - wx_i + b \leq \varepsilon + \xi_i$$

$$wx_i - b - l_i \leq \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

with C which is a control parameter like in soft margin.
To solve this problem, we introduce a new time Lagrange multipliers and we will get this regression function :

$$f(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) - b$$

©2005-2012 Phimeca Engineering

2 Use Cases Guide

This section presents the main functionalities of the module *otsvm* in their context.

2.1 Which python modules to import ?

In order to use the functionalities described in this documentation, it is necessary to import :

- the openturns python module which gives access to the OpenTURNS functionalities.
- the *otsvm* module which links the *openturns* functionalities.

Python script for this use case :

```
# Load OpenTURNS
from openturns import *
#Load the svm module
from otsvm import *
```

2.2 UC : creation of a SVM regression algorithm

The objective of this Use Case is to create a SVM Regression algorithm in order to create a metamodel. Otsvm enables :

- to set lists of tradeoff factors and kernel parameter with the methods setTradeoffFactor, setKernelParameter.
- to select the kernel type in this list : Linear Kernel, Polynomial Kernel, Sigmoid Kernel, RBF kernel.
- to compute the algorithm on an input and output samples.
- to compute the algorithm on an experiment plane and a function.
- to compute the algorithm on an input and output samples and an isoprobabilistic distribution.

We recommend for users to use the RBF Kernel (the gaussian kernel). Moreover, it is important to understand that the selection of parameters (kernel parameter and tradeoff factor) is primary. If you don't know what to take as parameters, you must take a wide range values, for example $tradeoff \in \{10^-5, 10^-3, 10^-1 ... 10^3\}$ $kernel\ parameter \in \{10^-15, 10^-13 ..., 10^3\}$. Normally, the algorithm always converges, but this can take a long while, mostly if you have a lot of parameters to test.

Python script for this UseCase :

```
#create a function , here we create the Sobol function
dimension = 3
meanTh = 1.0
a = NumericalPoint(dimension)
inputVariables = Description(dimension)
outputVariables = Description(1)
outputVariables[0] = "y"
formula = Description(1)
formula[0] = "1.0"
```

©2005-2012 Phimeca Engineering

```
covTh = 1.0
for i in range(dimension):
  a[i] = 0.5*i
  covTh = covTh * (1.0 + 1.0 / (3.0 * (1.0 + a[i])**2))
  inputVariables[i] = "xi" + str(i)
  formula[0] = formula[0] + " * ((abs(4.0 * xi" +str(i) + " -2.0) + " +
  str(a[i]) + ") / (1.0 + " + str(a[i]) + "))"
covTh = covTh -1.0
model = NumericalMathFunction(inputVariables, outputVariables, formula)

#create the input distribution
RandomGenerator.SetSeed(0)
marginals = DistributionCollection(dimension)
for i in range(dimension):
  marginals[i] = Uniform(0.0, 1.0)
distribution = ComposedDistribution(marginals)

#create lists of kernel parameters and tradeoff factors
tradeoff = NumericalPoint([0.01,0.1,1,10,100,1000])
kernel = NumericalPoint([0.001,0.01,0.1,1,10,100])

#first example : create the problem with an input and output samples:
#first, we create samples
dataIn = distribution.getNumericalSample(250)
dataOut = model(dataIn)
#second, we create our svm regression object, we must select the third parameter
#in an enumerate in the list { NormalRBF, Linear, Sigmoid, Polynomial }
Regression = SVMRegression(dataIn, dataOut, LibSVM.NormalRBF)
#third, we set kernel parameter and tradeoff factor
Regression.setTradeoffFactor(tradeoff)
Regression.setKernelParameter(kernel)
# Perform the algorithm
Regression.run()
# Stream out the results
SVMRegressionResult = Regression.getResult()
# get the residual error
residual = result.getResiduals()
# get the relative error
relativeError = result.getRelativeErrors()

#second example : create the problem with an experiment plane:
#first, we create the plane
myPlane = MonteCarloExperiment(distribution, 250)
myExperiment = Experiment(myPlane, "example")
#second, we create our svm regression object, the first parameter is the function
Regression2 = SVMRegression(model, myExperiment,
LibSVM.Linear)
#third, we set kernel parameter and tradeoff factor
```

©2005-2012 Phimeca Engineering

```
Regression2.setTradeoffFactor(tradeoff)
Regression2.setKernelParameter(kernel)
# Perform the algorithm
Regression2.run()
# Stream out the results
SVMRegressionResult = Regression2.getResult()
# get the residual error
residual = result.getResiduals()
# get the relative error
relativeError = result.getRelativeErrors()

#third example : create the problem with an isoprobabilistic distribution
#first , we create our distribution
marginals = DistributionCollection(dimension)
for i in range(dimension):
  marginals[i] = Uniform(0.0, 1.0)
distribution = ComposedDistribution(marginals)
#second, we create input and output samples
dataIn = distribution.getNumericalSample(250)
dataOut = model(dataIn)
#third, we create our svm regression
Regression3 = SVMRegression(dataIn,dataOut,distribution,
LibSVM.Polynomial)
#and to finish, we set kernel parameter and tradeoff factor
Regression3.setTradeoffFactor(tradeoff)
Regression3.setKernelParameter(kernel)
# Perform the algorithm
Regression3.run()
# Stream out the results
SVMRegressionResult = Regression3.getResult()
# get the residual error
residual = result.getResiduals()
# get the relative error
relativeError = result.getRelativeErrors()

#fourth example is here to present you the SVMResourceMap class.
#Users can fix others parameters like the degree and the constant of the
#Polynomial Kernel,the cacheSize, the number of folds or the epsilon
#first , we create samples
dataIn = distribution.getNumericalSample(250)
dataOut = model(dataIn)
#second, we create our svm regression object
#here, we select the Polynomial Kernel but by default his degree is 3. We want a
#degree of 2
ResourceMap.Set("LibSVM-DegreePolynomialKernel","2")
#now the degree of the Polynomial kernel is 2
Regression = SVMRegression(dataIn, dataOut, LibSVM.Polynomial)
#third, we set kernel parameter and tradeoff factor
```

©2005-2012 Phimeca Engineering

```
Regression.setTradeoffFactor(tradeoff)
Regression.setKernelParameter(kernel)
# Perform the algorithm
Regression.run()
# Stream out the results
SVMRegressionResult = Regression.getResult()
# get the residual error
residual = result.getResiduals()
# get the relative error
relativeError = result.getRelativeErrors()
```

2.3 UC : Creation of a Classification algorithm

The objective of this Use Case is to create a SVM Classification algorithm in order to build a metamodel that separates datas in 2 classes.
Otsvm enables to :

- to set lists of tradeoff factors and kernel parameter with the methods setTradeoffFactor, setKernelParameter.
- to select the kernel type in this list : Linear Kernel, Polynomial Kernel, Sigmoid Kernel, RBF kernel.
- to compute the algorithm on an input and output samples.

Python script for this UseCase :

```
#this example uses a csv file with the datas for the classification
#we retreive the sample from the file sample.csv
path = os.path.abspath(os.path.dirname(__file__))
dataInOut = NumericalSample().ImportFromCSVFile(path + "/sample.csv")

#we create dataIn and dataOut
dataIn=NumericalSample(861,2)
dataOut=NumericalSample(861,1)

#we build the input Sample and the output Sample because we must separate dataInOut
for i in range(861):
  a=dataInOut[i]
  b=NumericalPoint(2)
  b[0]=a[1]
  b[1]=a[2]
  dataIn[i]=b
  dataOut[i]=int(a[0])

#list of C parameter
cp=NumericalPoint([0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100])
#list of gamma parameter in kernel function
gamma=NumericalPoint([0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100])

#create the Classification Problem
```

©2005-2012 Phimeca Engineering

```
Regression=LibSVMClassification(dataIn,dataOut)
Regression.setKernelType(LibSVM.NormalRbf)
Regression.setTradeoffFactor(cp)
Regression.setKernelParameter(gamma)

#compute the classification
Regression.run()
print "#######################"
print "Results with Samples I/O"
print "Accuracy(p.c.)=",Regression.getAccuracy()
```

©2005-2012 Phimeca Engineering

3 User Manual

This section gives an exhaustive presentation of the objects and functions provided by the *otsvm* module, in the alphabetic order.

3.1 LibSVMRegression

A *LibSVMRegression* is an *otsvm* object.

Usage :

$LibSVMRegression(dataIn, dataOut, kerneltype)$

$LibSVMRegression(inputFunction, experience, kerneltype)$

$LibSVMRegression(dataIn, dataOut, distribution, kerneltype)$

Arguments :

dataIn : a NumericalSample, an OpenTURNS object which is the input sample.

dataOut : a NumericalSample, an OpenTURNS object which is the output sample.

kerneltype : a Libsvm enum which is the selection of the kernel. We can select (Linear, Polynomial, NormalRbf, Sigmoid).

experience : an Experiment, an OpenTURNS object which is an experiment plane.

distribution : a Distribution, an OpenTURNS object which is the isoprobalistic distribution

Value : a LibSVMRegression which is the object manipulated to make the regression.

Some methods :

run

Usage : $run()$

Argument : none.

Value : none. It makes the svm regression and builds the metamodel. To understand the algorithm, First, we make a cross validation to determinate the best parameters. Second, we train the problem and retreive some results (support vectors, support vectors coefficients, kernel parameters). Third, we build the model with OpenTurns and save results in the MetaModelResult.

©2005-2012 Phimeca Engineering

3.2 LibSVMClassification

A $LibSVMClassification$ is an *otsvm* object.

Usage :

$LibSVMClassification(dataIn, outClasses)$

Arguments :

$dataIn$: a NumericalSample, an OpenTURNS object which is the input sample.

$outClasses$: an Indices, an OpenTURNS object which is the output labels (labels must be positive).

Value : a LibSVMClassification which is the object manipulated to make the classification.

Some methods :

run

Usage : $run()$

Argument : none.

Value : none. It makes the svm classification and builds the metamodel.

$classify$

Usage :

$getLabel(vector)$

$getLabel(sample)$

Argument :

vector, an OpenTURNS object which is a NumericalPoint. This is the input vector to classify.

sample, an OpenTURNS object which is a NumericalSample. This is an input sample to classify.

Value :

an UnsignedLong for the vector, an openturns type. It predicts for an input vector , the output label.

a Indices for the sample, an openturns object. It predicts for a sample, the output label for each vector.

$setKernelType$

Usage : $setKernelType(kerneltype)$

Argument : *kerneltype*: an enum from LibSVM. We can select (Linear, Polynomial, NormalRbf, Sigmoid).

Value : none. It set the type of the kernel.

$setTradeoffFactor$

Usage : $setTradeoffFactor(trade)$

Argument : $trade$: a NumericalPoint, an OpenTURNS object.

Value : none. It set the tradeoff factor.

$setKernelParameter$

Usage : $setKernelParameter(kernel)$

Argument : $kernel$: a NumericalPoint, an OpenTURNS object.

Value : none. It set the kernel parameter.

©2005-2012 Phimeca Engineering

grade

Usage :

$grade(vector, outClasse)$

$grade(sample, indices)$

Argument :

vector, an OpenTURNS object which is a NumericalPoint. This is the input vector.

sample, an OpenTURNS object which is a NumericalSample. This is the input sample.

outClasse, an UnsignedLong, an OpenTURNS type. This is a potential label.

indices, a Indices, an OpenTURNS object. This is a list of potentials labels.

Value :

an UnsignedLong for the vector, an openturns type. It gives a positive integer. More the value is high and more the given label in parameter is a good label for the vector. The maximum of this value is number of classes - 1.

a Indices for the sample, an openturns object. It gives a list of positives integers. More the value is high and more the given label in parameter is a good label for each vector. The maximum of this value is number of classes - 1.

setWeight

Usage : $setWeight(weigt)$

Argument : *weight* : a NumericalPoint, an OpenTURNS object.

Value : none. For each class, we associate a penalty. It is useful for unbalances data or assymetric misclassification cost.

©2005-2012 Phimeca Engineering

3.3 Various

The module adds some keys to the $ResourceMap$ in OpenTURNS.

There is five new keys :

- $LibSVM - DegreePolynomialKernel$: the degree of the polynomial kernel d. By default, it's 3.
- $LibSVM - ConstantPolynomialKernel$: the constant of the polynomial kernel c. By default it's 0.
- $LibSVM - CacheSize$: the cache size of the calculation for kernel trick in MB. By default, it's 100.
- $LibSVM - Epsilon$: the parameter ε in the regression. By default it's 0.001.
- $LibSVMRegression - NumberOfFolds$: the number of folds for the cross validation. By default it's 3.
- $LibSVM - Shrinking$: a boolean. We can use this to to eliminate outlying vectors. By default it's False.

If we want to change the value , we must use the method $ResourceMap.Set(key, value)$. If we want to retrieve the value , we must use the method $ResourceMap.Get(key)$.

©2005-2012 Phimeca Engineering