edu.udo.cs.yale.example

Class Example

java.lang.Object
└ **edu.udo.cs.yale.example.Example**

public class **Example**
extends java.lang.Object

An example consists of regular attributes used for learning purposes and a set of special attributes like labels, clusters, or ids. The data is backed by a DataRow. Hence, all values are actually doubles, symbolic values are mapped to integers stored in doubles.
Since ExampleSets are only a view on ExampleTables, Examples are generated on the fly by ExampleReaders. Since they only contain the currently selected attributes operators need not to consider attribute selections or example subsets (samplings).
A given weight applier scales the attribute values if this example is part of an weighted example set.

Version:
$Id: Example.java,v 2.39 2006/08/03 14:39:27 ingomierswa Exp $
Author:
Ingo Mierswa, Simon Fischer

Field Summary

private Attribute[]	**attributes** The attributes used by this example.
private AttributeWeights	**attributeWeights** The weights of the attributes.
private DataRow	**data** The data for this example.
static java.lang.String	**SEPARATOR** Separator used in the getAttributesAsString() method (tab).
static java.lang.String	**SPARSE_SEPARATOR** Separates indices from values in sparse format (colon).
private java.util.Map<java.lang.String,Attribute>	**specialAttributes** The map with all special attributes.
private WeightApplier	**weightApplier** The value of all attributes is calculated as a function f(original value, weight) by an instance of WeightApplier.

Constructor Summary

Example(DataRow data, Attribute[] attributes, java.util.Map<java.lang.String,Attribute> specialAttributes) Creates a simple example without attribute weights and weight applier (null).
Example(DataRow data, Attribute[] attributes, java.util.Map<java.lang.String,Attribute> specialAttributes, AttributeWeights attributeWeights, WeightApplier weightApplier) Creates a new Example that uses the data stored in a DataRow.

Method Summary

void	**copySpecialAttributesTo**(Example other) Copies the values of the special attributes of this example into the given example.
Attribute	**getAttribute**(int i) Returns the i-th regular attribute.
Attribute	**getAttribute**(java.lang.String name) Returns the special attribute with the given name.
java.lang.String	**getAttributesAsSparseString**() Calls getAttributesAsSparseString(String,String) using default separator characters.
java.lang.String	**getAttributesAsSparseString**(java.lang.String separator, java.lang.String indexValueSeparator) Returns the attribute values in the format index:value index:value Index starts with 1.
java.lang.String	**getAttributesAsString**() This string output can be used for file output.
java.lang.String	**getAttributesAsString**(java.lang.String sep) This string output can be used for file output.
double	**getConfidence**(java.lang.String value) Returns the confidence for the given value.
DataRow	**getDataRow**() Returns the data row which backs up the example in the example table.
double	**getLabel**() Returns the double value of the label attribute.
int	**getNumberOfAttributes**() Returns the number of regular attributes.
double	**getPredictedLabel**() Returns the double value of the predicted label attribute.
java.util.Collection<java.lang.String>	**getSpecialAttributeNames**() Returns a collection of the names of all special attributes defined for this example.
double	**getUnweightedValue**(Attribute a) Returns the original, unweighted value of the given attribute.
double	**getValue**(Attribute a) Returns the value of attribute a.
double	**getValue**(int index) Invokes the method getValue(Attribute) for the i-th regular attribute.
java.lang.String	**getValueAsString**(Attribute attribute) Returns the value of this attribute as string representation, i.e. the number as string for numerical attributes and the correctly mapped categorical value for nominal values.
java.lang.String	**getValueAsString**(Attribute attribute, int fractionDigits) Returns the value of this attribute as string representation, i.e. the number as string for numerical attributes and the correctly mapped categorical value for nominal values.
double	**getWeight**() Returns the double value of the weight attribute.

void	**setConfidence**(java.lang.String value, double confidence) Sets the confidence for the given nominal value.
void	**setLabel**(double value) Sets the value for the label attribute.
void	**setPredictedLabel**(double value) Sets the value for the label attribute.
void	**setValue**(Attribute a, double value) Sets the value of attribute a.
void	**setValue**(Attribute a, java.lang.String str) Sets the value of attribute a which must be a nominal attribute.
void	**setWeight**(double value) Sets the value for the weight attribute.
java.lang.String	**toSparseString**(int format) Returns regular and some special attributes (label, id, and example weight) in sparse format.
java.lang.String	**toString**() This method returns a dense string representation of the example.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

SEPARATOR

public static final java.lang.String **SEPARATOR**

Separator used in the getAttributesAsString() method (tab).

See Also:
Constant Field Values

SPARSE_SEPARATOR

public static final java.lang.String **SPARSE_SEPARATOR**

Separates indices from values in sparse format (colon).

See Also:
Constant Field Values

---

data

private DataRow **data**

The data for this example.

---

attributes

private Attribute[] **attributes**

The attributes used by this example.

---

specialAttributes

private java.util.Map<java.lang.String, Attribute> **specialAttributes**

The map with all special attributes.

---

attributeWeights

private AttributeWeights **attributeWeights**

The weights of the attributes.

---

weightApplier

private WeightApplier **weightApplier**

The value of all attributes is calculated as a function f(original value, weight) by an instance of WeightApplier.

Constructor Detail

Example

public **Example**(DataRow data,
Attribute[] attributes,
java.util.Map<java.lang.String, Attribute> specialAttributes)

Creates a simple example without attribute weights and weight applier (null). Using this constructor will speed up example construction in cases where feature weights are not given / needed.

Example

public **Example**(DataRow data,
Attribute[] attributes,
java.util.Map<java.lang.String, Attribute> specialAttributes,
AttributeWeights attributeWeights,
WeightApplier weightApplier)

Creates a new Example that uses the data stored in a DataRow. The array of attributes corresponds to the regular attributes which should be used for learning. Like ExampleSets each example knows the special attributes like labels. The given attribute weights are used in combination with the weight applier to generate scaled attribute values on the fly.

Method Detail

getDataRow

public DataRow **getDataRow**()

Returns the data row which backs up the example in the example table.

---

getAttribute

public Attribute **getAttribute**(int i)

Returns the i-th regular attribute. These are the non-special attributes used for learning.

---

getAttribute

public Attribute **getAttribute**(java.lang.String name)

Returns the special attribute with the given name. If no special attribute with the given name is found, this method tries to return the first regular attribute with the given name. If neither a special attribute nor a regular attribute was found, this method returns null.

---

getNumberOfAttributes

public int **getNumberOfAttributes**()

Returns the number of regular attributes.

---

getSpecialAttributeNames

public java.util.Collection<java.lang.String> **getSpecialAttributeNames**()

Returns a collection of the names of all special attributes defined for this example.

---

copySpecialAttributesTo

public void **copySpecialAttributesTo**(Example other)

Copies the values of the special attributes of this example into the given example.

---

getValue

public double **getValue**(int index)

Invokes the method getValue(Attribute) for the i-th regular attribute.

---

getValue

public double **getValue**(Attribute a)

Returns the value of attribute a. The attribute a need not necessarily be part of the example set the example is taken from, although this is no good style. If no weight is given for the attribute or the attribute is nominal the unweighted value is returned.

---

getUnweightedValue

public double **getUnweightedValue**(Attribute a)

Returns the original, unweighted value of the given attribute.

---

setValue

public void **setValue**(Attribute a,
double value)

Sets the value of attribute a. The attribute a need not necessarily be part of the example set the example is taken from, although this is no good style.

---

setValue

public void **setValue**(Attribute a,
java.lang.String str)

Sets the value of attribute a which must be a nominal attribute. The attribute a need not necessarily be part of the example set the example is taken from, although this is no good style.

---

getValueAsString

public java.lang.String **getValueAsString**(Attribute attribute)

Returns the value of this attribute as string representation, i.e. the number as string for numerical attributes and the correctly mapped categorical value for nominal values.

---

getValueAsString

public java.lang.String **getValueAsString**(Attribute attribute, int fractionDigits)

Returns the value of this attribute as string representation, i.e. the number as string for numerical attributes and the correctly mapped categorical value for nominal values. If the value is numerical the given number of fraction digits is used.

---

getLabel

public double **getLabel**()

Returns the double value of the label attribute. This must be cast to integer and mapped with help of the label attribute to get the correct nominal value. Therefore, for classification tasks the usage of the getValueAsString(Attribute label) method is suggested.

---

setLabel

public void **setLabel**(double value)

Sets the value for the label attribute. For classification tasks this value must be mapped to an integer using the mapString(String) method of attribute.

---

getPredictedLabel

public double **getPredictedLabel**()

Returns the double value of the predicted label attribute. This must be cast to integer and mapped with help of the label attribute to get the correct nominal value. Therefore, for classification tasks the usage of the getValueAsString(Attribute label) method is suggested.

---

setPredictedLabel

public void **setPredictedLabel**(double value)

Sets the value for the label attribute. For classification tasks this value must be mapped to an integer using the mapString(String) method of attribute.

---

getConfidence

public double **getConfidence**(java.lang.String value)

Returns the confidence for the given value. Will throw a NullPointerException if the example did not have an predicted label attribute. It will throw an ClassCastException if the predicted label attribute is not of type ConfidencesAttribute.

---

setConfidence

public void **setConfidence**(java.lang.String value,
double confidence)

Sets the confidence for the given nominal value. Will throw a NullPointerException if the example did not have an predicted label attribute. It will throw an ClassCastException if the predicted label attribute is not of type ConfidencesAttribute.

---

getWeight

public double **getWeight**()

Returns the double value of the weight attribute.

---

setWeight

public void **setWeight**(double value)

Sets the value for the weight attribute.

---

getAttributesAsString

public java.lang.String **getAttributesAsString**()

This string output can be used for file output.

---

getAttributesAsString

public java.lang.String **getAttributesAsString**(java.lang.String sep)

This string output can be used for file output.

---

getAttributesAsSparseString

public java.lang.String **getAttributesAsSparseString**()

Calls getAttributesAsSparseString(String,String) using default separator characters.

---

getAttributesAsSparseString

public java.lang.String **getAttributesAsSparseString**(java.lang.String separator,
java.lang.String indexValueSeparator)

Returns the attribute values in the format
index:value index:value

Index starts with 1.

Parameters:

separator - separates attributes
indexValueSeparator - separates index and value.

---

toString

public java.lang.String **toString**()

This method returns a dense string representation of the example. It first returns the values of all special attributes and then the values of all regular attributes.

Overrides:

toString in class java.lang.Object

---

toSparseString

public java.lang.String **toSparseString**(int format)

Returns regular and some special attributes (label, id, and example weight) in sparse format.

Parameters:

format - one of the formats specified in SparseFormatDataRowReader

---

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
