API Documentation

API Documentation

October 7, 2009

Contents

1 Module ClustersTools

This module provides tools to represent and manage clusters.

1.1 Variables

Name	Description
__package__	**Value:** `None`
verdana	**Value:** `<ImageFont.ImageFont instance at 0x00000000030C3E88>`

1.2 Class ClusterSet

The class ClusterSet defines a set of clusters, each clusters containing vectors from a specified dictionnary.

1.2.1 Methods

__init__(*self*, *vectors*)

An instance is created containing no clusters, only a dictionnary of vectors defined by "vectors".

attributeVector(*self*, *vectID*, *clusterID*)

This method attributes the vector which key in the dictionnary is "vectID" to the cluster that has the ID "clusterID". If the cluster doesn't exist it is created.

boxDimensions(*self*)

This method returns the width and the hight of the box representation of the ClusterSet.

computeCentroids(*self*)

This method computes the centers of each cluster.

drawDiagram(*self*, *im*)

This method takes a PIL image instance "im" and draws in it the ClusterSet's box representation.

getCenter(*self*, *clusterID*)

This method returns the lastly computed center of the cluster which ID is "clusterID".

hasChanged(*self*)

This method returns True if the cluster attribution has changed since the last time it was executed or the creation of the object.

setFullLabelWriter(*self*, *fullLabelWriter*)

This method defines the full label writer (as "fullLabelWriter") : this is a function that from a vector ID returns a text description of the vector.

setThumbnailAccessor(*self*, *thumbnailAccessor*)

This method defines the thumbnail accessor (as "thumbnailAccessor") : this is a function that from a vector ID returns a picture description of the vector.

1.3 Class Tree

This class represents a very general tree. A Tree can be:

- empty
- a leaf
- a list of other non empty trees (its branches)

A leaf has a label (which can be any object). If a Tree isn't empty and has branches, it has a label which is obtained by merging the labels of its branches. A Tree can be used to represent a hierarchical clustering of word vectors. In that case, a label will be composed of an id and a WordVector.

1.3.1 Methods

__init__(*self*, *labelMerger*=`None`)

A Tree is created empty by the constructor. The optional argument "labelMerger" is the Tree's label merger : a function that can merge two labels.

addBranch(*self*, *branch*)

This method adds the tree "branch" to the branches of the Tree.

boxDimensions(*self*)

This method returns the width and the hight of the box representation of the Tree.

countBranches(*self*)

This method returns the number of branches a Tree has.

drawDiagram(*self*, *im*, *topLeft*=`(0, 0)`, *gray*=`True`)

This method takes a PIL image instance "im" and draws in it the Tree's box representation.

fullLabel(*self*, *label*)

This method takes a label "label" and returns its text description. It uses the full label writer if it is defined and otherwise, if the Tree is a branch, it uses its parent Tree's "fullLabel" method.

getBranchLabel(*self*, *i*)

This method returns the i-th branch's label.

getLabel(*self*)

This method returns the Tree's label.

getThumbnail(*self*, *label*)

This method takes a label "label" and returns its picture description. It uses the thumbnail accessor if it is defined and otherwise, if the Tree is a branch, it uses its parent Tree's "getThumbnail" method.

mergeBranches(*self*, *i*, *j*)

This method takes the i-th and the j-th branches of the Tree and makes a new Tree containing only these branches. It replaces these two branches with the new Tree.

mergeLabels(*self*, *label1*, *label2*)

This method takes two labels ("label1" and "label2") and returns the merger of these. It uses the label merger if it is defined and otherwise, if the Tree is a branch, it uses its parent Tree's "mergeLabels" method.

setFullLabelWriter(*self*, *fullLabelWriter*)

This method defines the full label writer (as "fullLabelWriter") : this is a function that from a label returns a text description of the label.

setLeaf(*self*, *label*)

This method transforms the Tree into a leaf with the label "label".

setParent(*self*, *parent*)

This method gives to a Tree that is a branch a pointer to its parent Tree ("parent"), allowing it to access its parent public methods.

setThumbnailAccessor(*self*, *thumbnailAccessor*)

This method defines the thumbnail accessor (as "thumbnailAccessor") : this is a function that from a label returns a picture description of the label.

2 Module IMDbTools

This module provides specific tools to get content from IMDb

2.1 Functions

getIMDbTop250()

This functions the IMDb IDs of the movies from the Top 250. IMDbPy doesn't provide a way to do this so this function accesses the webpage displaying the Top 250 and retreives the IDs from the HTML source.

openCachedURL(*url*)

This function creates a file object containing the data from the given URL "url". It downloads the content to a file which name is the URL (with the special characters encoded). If the file exists it only opens it, without downloading again.

2.2 Variables

Name	Description
__package__	**Value:** `None`

2.3 Class IMDbInterface

This is some kind of a offline capable version of the IMDb access provided by IMDbPy. It download data and saves it to a local file. If the requested data is already in the file, it doesn't download it again.

2.3.1 Methods

__**init**__(*self*, *filename*)

"filename" is the name of the file in which the data is (or will be) stored. If the file doesn't exist, it is created.

getMovie(*self*, *movieID*)

This method return a IMDbPy movie object from its ID "movieID".

getThumbnail(*self*, *movieID*, *size*=`100`)

This method returns a thumbnail corresponding to the movie which ID is "movieID". The optional parameter "size" sets the bigest dimension of the thumbnail in pixels. Its default value is 100 pixels. If the thumbnail cannot be found, it returns "False".

3 Module WordsTools

This module provides tools to represent and manage texts as word vectors.

3.1 Functions

wordHistogramFromTextList(*textList*)

This function returns a word histogram from a list of texts "textList".

3.2 Variables

Name	Description
__package__	**Value:** `None`

3.3 Class WordVector

The WordVector object behaves as a dictionnary : the keys are words and the values are positive (or zero) numbers. However, a word is present among the keys only if it is associated with a value different from 0 in more than 10% and less than 50% of all the WordVector instances. Thus creating a new WordVector can change the words contained in all the WordVector instances. Arithmetic operations between two WordVector instances or between a WordVector and a number are possible but they return a "pseudo" WordVector : its coeficients might be negative and its creation doesn't affect the words present in the other instances.

3.3.1 Methods

__**add**__(*self*, *other*)

This method allows addition between two WordVector instancesaor a WordVector and a number to return a "pseudo" WordVector.

__**div**__(*self*, *other*)

This method allows (pointwise) division between two WordVector instances or a WordVector and a number to return a "pseudo" WordVector.

__**getitem**__(*self*, *word*)

This method returns the coeficient for the key "word" of the WordVector when it is considered as a dictionary.

__**init**__(*self*, *wordHistogram*, *updateVectorType*=`True`)

A WordVector is obtained from a word histogram "wordHistogram" which is a dictionnary associating positive umbers to words. One can create a "pseudo" WordVector by setting the optional parameter "updateVectorType" to False.

__iter__(*self*)

This method associates a WordVector with a WordVectorIterator so it can be considered as a dictionary.

__len__(*self*)

This method returns the dimension of the WordVector when it is considered as a dictionnary.

__mul__(*self*, *other*)

This method allows (pointwise) multiplication between two WordVector instances or a WordVector and a number to return a "pseudo" WordVector.

__sub__(*self*, *other*)

This method allows substraction between two WordVector instances or a WordVector and a number to return a "pseudo" WordVector.

__truediv__(*self*, *other*)

This method allows (pointwise) division between two WordVector instances or a WordVector and a number to return a "pseudo" WordVector.

copy(*self*)

This method returns a copy as a "pseudo" WordVector.

words()

This method returns the words that are (currently) present in all the WordVector instances.

3.4 Class WordVectorIterator

This class is used internally to iterate through WordVector objects so they can be considered has dictionarries.

3.4.1 Methods

__init__(*self*, *vect*)

__iter__(*self*)

next(*self*)

4 Module clustering

This module defines the actual algorithms used for clustering.

4.1 Functions

hierarchicalCluster(*vectors*, *simMeasure*)

This function computes the hierarchical clustering from the WordVector instances in the "vectors" list using the similarity measure "simMeasure". It returns a Tree with WordVector labels.

kMeansClustering(*vectors*, *simMeasure*, *k*)

This function computes the k-mean clustering from the WordVector instances in the "vectors" list using the similarity measure "simMeasure" and "k". It returns a ClusterSet containing WordVector instances.

mergeWordVectorsLabels(*label1*, *label2*)

This function returns the merger of two WordVector labels "label1" and "label2". A WordVector label is a couple of an ID and a WordVector. The merged label is a couple of a list containing the input IDs or a concatenation if one of these IDs was already a list, and the "pseudo" WordVector obtained as the arithmetic average of the two input WordVector instances.

similarity(*vect1*, *vect2*, *measure*=`'pearson'`)

This function returns the similarity between the two vectors "vect1 and "vect2". The optional third argument, which is by default "pearson" specifies the kind of similarity. The other possibilies are "tanimoto" and "inverse euclidian"

4.2 Variables

Name	Description
__package__	**Value:** `None`

Index