PyCUB: Machine exploration of the Codon Usage Bias

Generated by Doxygen 1.8.14

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

PyCUB
Main object of the project that allows the user to access most of the functions . . . . . **??**

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

MutableMapping
- PyCUB.homoset.HomoSet . . . . . **??**

object
- PyCUB.espece.Espece . . . . . **??**
- PyCUB.homology.homology . . . . . **??**
- PyCUB.pyCUB.PyCUB . . . . . **??**

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

PyCUB.espece.Espece
: Docstring for Espece . . . . . **??**

PyCUB.homology.homology
: In homology we store an homology with all its related data, ... **??**

PyCUB.homoset.HomoSet
: HomoSet is the object containing evrey homology as a dictionnary according to thie rhomology code . . . . . **??**

PyCUB.pyCUB.PyCUB . . . . . **??**

Chapter 4

Namespace Documentation

4.1 PyCUB Namespace Reference

is the main object of the project that allows the user to access most of the functions

4.1.1 Detailed Description

is the main object of the project that allows the user to access most of the functions

When using it, please follow the documentation and examples on notebooks thought you can still use it as you please and use some of the nice tricks provided here and in python

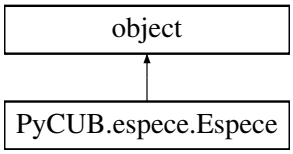
Chapter 5

Class Documentation

5.1 PyCUB.espece.Espece Class Reference

docstring for Espece

Inheritance diagram for PyCUB.espece.Espece:



Public Member Functions

- def __init__ (self, kwargs)

  can intialize the file from kwargs as a raw dictionnary for json format (output of dictify) or from regular args.
- def __str__ (self)

  will present some interesting info about this species.
- def get_tRNAcopy (self, by="entropy", setnans=False, kingdom='fungi', baseCNvalue=2)

  Retrieves tRNA copy numbers from ensembl DB.
- def gettaxons (self, kingdom='fungi')

  Pars the ensemblgenomes REST API to retrieve the taxons id.
- def get_epigenomes (self)

  get from ensembl all the data about the epigenome that could help asking interesting questions about the CUB

Public Attributes

- **name**
- **metadata**
- **is_stored**
- **num_genes**
- **genome_size**

Static Public Attributes

- code = None
  a dict from gene_name to dna_seq string (deprecated)
- int num_genes = 0
  the number of gene of this species
- int genome_size = 0
  the bp size of the coding genome
- link = None
  the link to the ensembl genome
- dictionary metadata
  a dict containing different metadata information that one can gather, preferentially boolean
- bool **is_stored** = False
- string name = ''
  the full scientific name of the species
- taxonid = None
  the number assoxiated to this taxon
- copynumbers = None
  the approx.
- average_entropy = None
  the mean CUB value for each amino acids (CUBD dimension)
- average_size = None
  the mean size of each homologies
- var_entropy = None
  the mean of fullvarentropy
- fullentropy = None
  the array containing all CUB values from the homologies of this species
- fullvarentropy = None
  the variance for each amino acids of the full CUB values of this species
- fullGCcount = None
  the GC content of the full coding genome
- varGCcount = None
  the variance of the GC content of the full coding genome
- **meanGChomo** = None
- tRNAentropy = None
  the entropy values of the copynumbers of the tRNAs if sufficient tRNAs exist
- tot_homologies = None
  the total number of homologies to cerevisiae
- **meanecai** = None

Private Member Functions

- def _dictify (self)
  Used by the saving function.

5.1.1 Detailed Description

docstring for Espece

This is an object that contains all required information of a species for PyCUB and some nice functions to interact for each species

5.1.2 Member Function Documentation

5.1.2.1 _dictify()

```
def PyCUB.espece.Espece._dictify (
              self )  [private]
```

Used by the saving function.

transform the object into a dictionary that can be json serializable

Returns

A dict holding every element to be jsonized

Referenced by PyCUB.pyCUB.PyCUB.loadmore().

5.1.2.2 get_tRNAcopy()

```
def PyCUB.espece.Espece.get_tRNAcopy (
              self,
              by = "entropy",
              setnans = False,
              kingdom = 'fungi',
              baseCNvalue = 2 )
```

Retrieves tRNA copy numbers from ensembl DB.

will print the number of tRNAs and the number of tRNAs with a knwon codons (the usefull ones) will stop and set a trace for the user to inspect the data to do so: please write "dat" in the console. if you see something that should be corrected please do so from the console directly or from the code if there seems to be an error in the code if it is an error in the db that you can't do anything, like a mismatched codon and amino acid, you can't do much. resume the process by typing "c" in the console.

Parameters

species	string, the species from which you want the Trna copy number

Returns

Will populate copynumbers. And tRNAentropy if by="entropy" Or will not do anything if the species is unavailable and will print it

longtabu

Referenced by PyCUB.espece.Espece.__str__().

5.1.2.3 gettaxons()

```
def PyCUB.espece.Espece.gettaxons (
              self,
              kingdom = 'fungi' )
```

Pars the ensemblgenomes REST API to retrieve the taxons id.

for the species from which we would not have any (downloaded via Yun for example)

longtabu

Referenced by PyCUB.espece.Espece.get_tRNAcopy().

5.1.3 Member Data Documentation

5.1.3.1 copynumbers

```
PyCUB.espece.Espece.copynumbers = None  [static]
```

the approx.

copynumbers if any of each tRNA known of this species

Referenced by PyCUB.espece.Espece.__init__(), and PyCUB.espece.Espece.get_tRNAcopy().

5.1.3.2 metadata

```
PyCUB.espece.Espece.metadata  [static]
```

Initial value:

```
=  {
        "isplant_pathogen": False,
        "isanimal_pathogen": False,
        "isplant_symbiotic": False,  # endophyte or mycorrhizal
        "isbrown_rot": False,
        "iswhite_rot": False
    }
```

a dict containing different metadata information that one can gather, preferentially boolean

Referenced by PyCUB.espece.Espece.__init__().

5.1.3.3 name

```
PyCUB.espece.Espece.name = ''  [static]
```

the full scientific name of the species

the name of the species

Referenced by PyCUB.espece.Espece.__init__(), PyCUB.espece.Espece.__str__(), and PyCUB.espece.Espece.↩
get_tRNAcopy().

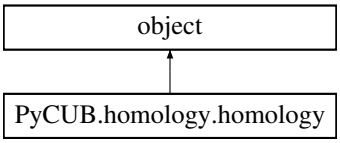
The documentation for this class was generated from the following file:

- PyCUB/espece.py

5.2 PyCUB.homology.homology Class Reference

in homology we store an homology with all its related data,

Inheritance diagram for PyCUB.homology.homology:



Public Member Functions

- def **__init__** (self, kwargs)
- def **__str__** (self)
- def remove (self, species)

  removes the list of species from this homology if it exists there
- def nb_unique_species (self)

  compute the number of unique species in this homologies
- def order (self, withtaxons=False)

  order the names by numerical increasing order
- def compute_averages (self)

  Computes the mean, var and mean of the homology.
- def reduce_dim (self, alg='tsne', n=2, perplexity=40)

  reduce the dimensionality of your gene dataset to a defined dimension using the t-SNE algorithm
- def plot (self, reducer="tsne", per=40, interactive=True, D=2, size=20)

  plot the dimensionality reduced datapoint of the homology matrix
- def clusterize_ (self, clustering='gaussian', eps=0.8, homogroupnb=None, assess=True, verbose=True)

  will clusterize the homology using gaussian mixture clustering or DBSCAN

Public Attributes

- **metrics**
- **homocode**
- **proteinids**
- **protein_abundance**
- **weight**
- **mRNA_abundance**
- **cys_elements**
- **is_secreted**
- **decay_rate**
- **cluster**

Static Public Attributes

- **full** = None
- **var** = None
- **mean** = None
- **clusters** = None
- **centroids** = None
- dictionary **metrics** = {}
- string **homocode** = 'None'
- **nans** = None
- **lenmat** = None
- **names** = None
- **doub** = None
- **GCcount** = None
- **reduced** = None
- **reduced_algo** = None
- **KaKs_Scores** = None
- **similarity_scores** = None
- list **proteinids** = [ ]
- **isrecent** = None
- **ishighpreserved** = None
- **geneids** = None
- **ref** = None
- **refprot** = None
- **refgene** = None
- **ecai** = None
- **meanecai** = None
- **cai** = None
- **meancai** = None
- int **protein_abundance** = 0.
- int **weight** = 0
- **conservation** = None
- int **mRNA_abundance** = 0.
- int **cys_elements** = 0
- bool **is_secreted** = False
- int **decay_rate** = 0.
- **tot_volume** = None
- **mean_hydrophobicity** = None
- **glucose_cost** = None
- **synthesis_steps** = None
- **isoelectricpoint** = None
- **othercods** = None

Private Member Functions

- def _dictify (self)

  Used by the saving function.

5.2.1 Detailed Description

in homology we store an homology with all its related data,

it reduced matrix with dim reduction and its clusters for example it is supposed to be store in a dictionary of homologies an homology is a set of genes from different species related by a common ancester gene and generally a common function. the unique metadatas are generally from the reference species/genome

Parameters

names	list of int corresponding to names in utils.speciestable
full	np.array[float] (species, amino) of one homology with entropy value vector per species
reduced	np.array[float] (species, x∗y) of one homology dimensionality reduced
clusters	list[int] of cluster val for each species
centroids	np.array[float] the position in aminoacid# Dimension of each centroids (number of centroid == number of cluster), dimension is reduced when plotted
metrics	a dict of metrics names and values for the cluster of this homology
nans	np.array[bool] wether or not this position is a nan
lenmat	a np.array[int] species amino of int of length of each amino acids (number)
doub	np.array[bool] of wether or not this position is a doublon (a copy number of the gene for the species)
reduced_algo	str dimensionality reduction algorithm used on this homology
var	np.array of the variance in the CUB value for each datapoint
mean	np.array[float] of the mean CUB for each datapoint
homocode	str the code of the homology
GCcount	np.array[float] GC bias for each datapoint
KaKs_Scores	np.array[float] a form of similarity score between genes/datapoints
similarity_scores	np.array[float] similarity score between genes/datapoints
proteinids	list[str] the protein names for each datapoints
isrecent	float a proxy for wether or not this homology has appeated recently
ishighpreserved	bool a proxy for wether or not this homology is conserved throughout evolution
geneids	list[str] the name of the genes
ref	np.array[float] the reference CUB value of cerevisiae gene
refprot	str the reference protein name of cerevisiae gene
refgene	str the reference gene name of cerevisiae gene
ecai	np.array[float] the codon adaptation index of each gene
meanecai	float the mean ecai of the ecai
protein_abundance	float the average abundance of the protein enoded by this gene in cerevisiae cells
weight	int the molecular weight of this protein
mRNA_abundance	float the average abundance of the messenger RNA of this coding gene in cerevisiae cells
cys_elements	int the number of cys regulatory elements known for cerevisiae on this gene (the value is only referenced for secreted genes)
is_secreted	bool wether or not this protein is secreted out of the cell
decay_rate	float the half life in minute of this protein

Parameters

tot_volume	int a proxy of the molecular volume of this protein
mean_hydrophobicity	a very distant proxy of the hydrophobicity of this protein
glucose_cost	the glucose cost of creating the amino acids required to build up this protein
synthesis_steps	the number of steps required by the cell to build up the amino acids of this protein
isoelectricpoint	float, a proxy of the Pi of this protein.
conservation	float, the total conservation of each amino of the corresponding protein
othercods	float, the average number of codons other than the ones of the 18 amino acids we are looking at per species on the homology

5.2.2 Member Function Documentation

5.2.2.1 _dictify()

```
def PyCUB.homology.homology._dictify (
              self )  [private]
```

Used by the saving function.

transform the object into a dictionary that can be json serializable

Parameters

None	

Returns

A dict holding every element to be jsonized

Referenced by PyCUB.homology.homology.clusterize_(), and PyCUB.pyCUB.PyCUB.loadmore().

5.2.2.2 clusterize_()

```
def PyCUB.homology.homology.clusterize_ (
              self,
              clustering = 'gaussian',
              eps = 0.8,
              homogroupnb = None,
              assess = True,
              verbose = True )
```

will clusterize the homology using gaussian mixture clustering or DBSCAN

and order them according to the density of each cluster (we are interested in the dense ones) and assess the quality using 3 criterion: BIC, AIC ,silhouette, cal_hara, phylodistance.

Parameters

clustering	str flag the clustering algorithm [gaussian, dbscan]
eps	float, hyperparam of the max size of the nsphere of each cluster
homogroupnb	int hyperparam of gaussian for the number of clusters
assess	wether or not to assess the quality of the clustering
verbose	wether or not to show clustering quality information

Returns

The clusters for each datapoint of the homology as a list[int]

longtabu

Referenced by PyCUB.homology.homology.plot().

5.2.2.3 compute_averages()

```
def PyCUB.homology.homology.compute_averages (
              self )
```

Computes the mean, var and mean of the homology.

Parameters

None	

Referenced by PyCUB.homology.homology.order().

5.2.2.4 nb_unique_species()

```
def PyCUB.homology.homology.nb_unique_species (
              self )
```

compute the number of unique species in this homologies

(basically count the number of doub)

Parameters

None	

Returns

The number of unique species in this homology

Referenced by PyCUB.homology.homology.remove().

5.2.2.5 order()

```
def PyCUB.homology.homology.order (
              self,
              withtaxons = False )
```

order the names by numerical increasing order

and sorts every representations as well according to this ordering

Parameters

withtaxons	bool to true if there is taxonomic data (present before preprocessing)

Referenced by PyCUB.homology.homology.nb_unique_species().

5.2.2.6 plot()

```
def PyCUB.homology.homology.plot (
              self,
              reducer = "tsne",
              per = 40,
              interactive = True,
              D = 2,
              size = 20 )
```

plot the dimensionality reduced datapoint of the homology matrix

colors represents the different clusters you can set the interactivity to gain deeper knowledge of the dataset. It can dim reduce the data and will also save the figure in utils/save. Moreover, it will print some interesting data about the homology

Parameters

reducer	str flag the algorithm to reduce the matrix of utils.cubD to D D
per	int he perplexity when using tsne algorithm
size	int the size of the plot
interactive	(recommended) wether to use bokeh or not to represend the data will show name of the species when hovering over datapoint will show a gradient of evolutionary distance of the species of the datapoint currently hovered to each other species
D	int the goal dimension (2-3-4) when using matplotlib

Returns

The plot interactive or not and some informations (as prints)

longtabu

Referenced by PyCUB.homology.homology.reduce_dim().

5.2.2.7 reduce_dim()

```
def PyCUB.homology.homology.reduce_dim (
              self,
              alg = 'tsne',
              n = 2,
              perplexity = 40 )
```

reduce the dimensionality of your gene dataset to a defined dimension using the t-SNE algorithm

Parameters

alg	a matrix of gene codon usage per species
n	the desired dimension
perplexity	an optional value when you know about tsne

Returns

The reduced dataset

Referenced by PyCUB.homology.homology.compute_averages(), and PyCUB.homology.homology.plot().

5.2.2.8 remove()

```
def PyCUB.homology.homology.remove (
              self,
              species )
```

removes the list of species from this homology if it exists there

Parameters

species	list[str] of species to remove

Referenced by PyCUB.homoset.HomoSet.compute_entropyloc().

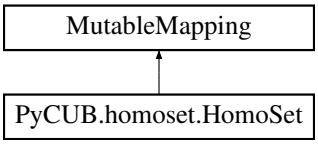
The documentation for this class was generated from the following file:

- PyCUB/homology.py

5.3 PyCUB.homoset.HomoSet Class Reference

HomoSet is the object containing evrey homology as a dictionnary according to thie rhomology code.

Inheritance diagram for PyCUB.homoset.HomoSet:



Public Member Functions

- def __init__ (self, kwargs)

  will initialize the object with the different values you might have from another project use the data dictionnary to add any type of data
- def __getitem__ (self, key)

  get the homology at the corresponding key
- def __setitem__ (self, key, value)

  add an item at the corresponding key
- def __delitem__ (self, key)

  same as in dict()
- def __iter__ (self)

  same as in dict()
- def iteritems (self)

  same as in dict()
- def __len__ (self)

  gives you the length of the homoset (number of homologies)
- def update (self, val)

  a function to update the dictionnary that homoset is
- def plot_all (self, With='tsne', perplexity=60, interactive=False, bins=100, offset=20, iteration=400, redo=False, bypasstsne=False, dotsize=7, inpar=True)

  will plot all the homologies in the full_homo_matrix (and compute it)
- def loadfullhomo (self)

  function to concatenate all the homologies in one big array(practicle for certain computations)
- def loadhashomo (self, withnames=False)

  function to compute the matrix of bool saying wether species X has a gene or more in homology Y
- def size (self)

  the size of the homoset (number of genes)
- def add_random_homology (self)

  a function to populate a homology with random values
- def preprocessing (self, withtaxons=False, withnames=True)

  will compute the full list of names,
- def preprocessing_taxons (self)

  preprocess the data by computing the names as ints
- def preprocessing_names (self)

  same as preprocessing_taxons() but admiting there is no taxon information (Yun's data for example)
- def preprocessing_namelist (self)

same as preprocessing_names() but without preprocessing the names of each homologies only updating the species_namelist

- def compute_ages (self, preserved=True, minpreserv=0.9, minsimi=0.85)

  will compute whether or not a coding gene is highly preserved and if not how recent it is
- def compute_entropyloc (self, using='computejerem')

  called if need entropy location and used ensembl data.
- def remove (self, species)

  remove this list of species from the homologies
- def clean_species (self, thresh=0.3)

  will remove a species from all the homologies of this homoset
- def plot_homoperspecies (self)

  will plot the number of homology per spcies in this homology group
- def plot_speciesperhomo (self)

  will plot the number of species per homologies in this homology group
- def cluster_homologies (self, clustering='kmeans', byspecie=False, order=True, plot_ordering=True, homogroupnb=2, findnb=False)

  Compute an homology group :
- def orderfromclust (self, homogroupnb, clust, byspecie=False, plot_ordering=True)

  creates an ordering of every elements
- def get_clusterstats (self, sort=True, interactive=True, redo=False)

  will find the number of cluster i per homologies and per species
- def compare_clusters (self, cubdistance_matrix=True, plot=True, interactive=True, size=40)

  for each clusters in homologies, will compare them with a similarity matrix and a distance matrix
- def find_clusters (self, clustering='dbscan', homogroupnb=None, assess=True, eps=0.8, best_eps=True, trainingset=30, epoch=20, ranges=[0.2, size=10, redo=False)

  Finds, for each homologies in the working homoset, groups that are part of compact clusters.
- def findbest_eps (self, trainingset=400, clustering="dbscan", epoch=20, ranges=[0.2, size=10, redo=False)

  will find the best eps hyperparameter (the one that minimizes the evolutionary distance within its clusters)
- def plot_hashomo (self, invert=False, size=40, interactive=False, rangeto=None)

  plot the has homo matrix
- def plot_simiclust (self, interactive=True, size=40)

  plot the similarity matrix of each homologies from its clusters
- def plot_distcub (self, interactive=False, size=40)

  plot the distance matrix of each homologies from the average of their CUB distances

Public Attributes

- **homo_namelist**
- **species_namelist**
- **homogroupnb**
- **clusters**
- **datatype**
- **phylo_distances**
- **best_eps**
- **wasclusterized**
- **stats**
- **homodict**
- **cluster_similarity**
- **cub_similarity**

Static Public Attributes

- hashomo_matrix = None

  a np.array[boolean] (species, homologies)
- homo_matrix = None

  np.array[float] (homologies∗species(inthehomologies),aminoacids)
- homo_matrixnames = None

  np.array[int] corresponding to names in PyCUB.utils.speciestable
- fulleng = None

  np.array[int] number of coding for each amino for each gene for each homology
- dictionary homodict = {}

  dictionnary of homology object containing codon usage per species
- list homo_namelist = [ ]

  list[str] of all the homology names
- list species_namelist = [ ]

  list[str] of all the species names
- list clusters = [ ]

  list[int] of clusters for the homoset clustering can be of size of nb of species
- int homogroupnb = 2

  int of group for the homology clustering
- red_homomatrix = None

  np.array[float] (homologies∗species(inthehomologies),x∗y)the reduced 2D version
- bool wasclusterized = False

  bool if the homologies have been clustered or not.
- **homo_clusters** = None
- string datatype = ''

  str of the CUB value type [entropy, A_value(entropyLocation), frequency]
- averagehomo_matrix = None

  np.array[float] of the avg CUB values per homologies
- dictionary stats = {}

  dict of statistics on the clusterings (see get_clusterstats())

Private Member Functions

- def _barplot (self, interactive=True)

  called by statistics function to plot a barplot of the proportion of different cluster per homologies and per species
- def _dictify (self, savehomos=False)

  Used by the saving function.
- def _plot_clust (self, mat, orderedmat)

  will plot the correlation matrix of the has_homomatrix before and after ordering allowing one to show its effect

5.3.1 Detailed Description

HomoSet is the object containing evrey homology as a dictionnary according to thie rhomology code.

from Homoset you can do much computation that requires the set of homologies Object where we store an homology group basically where we do our entire Computation from. Be aware that even if you use str, the keys will be stored as unicode as jsonized dict will output unicode in python < 3

5.3.2 Constructor & Destructor Documentation

5.3.2.1 __init__()

```
def PyCUB.homoset.HomoSet.__init__ (
              self,
              kwargs )
```

will initialize the object with the different values you might have from another project use the data dictionnary to add any type of data

a dictionary to any values present in the homoset

5.3.3 Member Function Documentation

5.3.3.1 __delitem__()

```
def PyCUB.homoset.HomoSet.__delitem__ (
              self,
              key )
```

same as in dict()

Parameters

key	str, unicode the key at which to add

```
        TypeError "the type you should enter is int or unicode or str"
```

longtabu

Referenced by PyCUB.homoset.HomoSet.__init__().

5.3.3.2 __getitem__()

```
def PyCUB.homoset.HomoSet.__getitem__ (
              self,
              key )
```

get the homology at the corresponding key

Parameters

key	str, unicode the key at which to get the homology

```
TypeError "the type you should enter is int or unicode or str"
```

longtabu

Referenced by PyCUB.homoset.HomoSet.__init__().

5.3.3.3 __setitem__()

```
def PyCUB.homoset.HomoSet.__setitem__ (
              self,
              key,
              value )
```

add an item at the corresponding key

Parameters

key	str, unicode the key at which to add
val	PyCUB.homology to add at this key

```
TypeError "the type you should enter is int or unicode or str"
```

longtabu

Referenced by PyCUB.homoset.HomoSet.__init__().

5.3.3.4 _barplot()

```
def PyCUB.homoset.HomoSet._barplot (
              self,
              interactive = True )  [private]
```

called by statistics function to plot a barplot of the proportion of different cluster per homologies and per species

Parameters

interactive	bool to true if you want to use the bokeh interactive version

Returns

the barplot holoviews object (will directly render if in a notebook)

Referenced by PyCUB.homoset.HomoSet.findbest_eps(), and PyCUB.homoset.HomoSet.get_clusterstats().

5.3.3.5 _dictify()

```
def PyCUB.homoset.HomoSet._dictify (
              self,
              savehomos = False )  [private]
```

Used by the saving function.

transform the object into a dictionary that can be json serializable

Parameters

savehomos	bool to true if you consider this homology as containing all the information (the other ones only have references to a subset of the data of this one)

Returns

the dictionnary of all the data in this Object in the correct format to be jsonized

Referenced by PyCUB.homoset.HomoSet.findbest_eps(), and PyCUB.pyCUB.PyCUB.loadmore().

5.3.3.6 _plot_clust()

```
def PyCUB.homoset.HomoSet._plot_clust (
              self,
              mat,
              orderedmat )  [private]
```

will plot the correlation matrix of the has_homomatrix before and after ordering allowing one to show its effect

Parameters

mat	np.array[bool] the current homology matrix
orderedmat	np.array[bool] the ordered homology matrix

Referenced by PyCUB.homoset.HomoSet.cluster_homologies(), and PyCUB.homoset.HomoSet.plot_distcub().

5.3.3.7 add_random_homology()

```
def PyCUB.homoset.HomoSet.add_random_homology (
              self )
```

a function to populate a homology with random values

Parameters

None	

Returns

the name of the random homology (str)

Referenced by PyCUB.homoset.HomoSet.loadfullhomo().

5.3.3.8 clean_species()

```
def PyCUB.homoset.HomoSet.clean_species (
              self,
              thresh = 0.3 )
```

will remove a species from all the homologies of this homoset

Warning, as the all/working homosets share the ref to homologies, deleting some species in working_homoset will result in removing some species in some homologies of all_homoset

Parameters

thresh	float the threshold of avg presence in homologies below which the species are remove

Referenced by PyCUB.homoset.HomoSet.compute_ages().

5.3.3.9 cluster_homologies()

```
def PyCUB.homoset.HomoSet.cluster_homologies (
              self,
              clustering = 'kmeans',
              byspecie = False,
              order = True,
              plot_ordering = True,
              homogroupnb = 2,
              findnb = False )
```

Compute an homology group :

from matrix computation using the homo_matrix (or from network computation in homologize_from_network) Can be computed many times and will updata homoset with the most recent homoset found if homoset exists, it will save it.

Parameters

clustering	str flags to 'kmeans', 'kmodes', 'fast' to use different sk-learn algorithms
plot	bool flags to true for the function to output ploting of the affinity matrix with and without the clusters
homogroupnb	int nb of groups you want to extract
byspecie	bool to true if we cluster by species instead of homologies
order	bool whether or not to order
plot_ordering	bool to true to plot this ordering
findnb	bool to true to find the right number of clusters (homogroupnb)

Returns

if findnb, will return the clusters for each homogroupnb up to 9

longtabu

Referenced by PyCUB.homoset.HomoSet.remove().

5.3.3.10 compare_clusters()

```
def PyCUB.homoset.HomoSet.compare_clusters (
              self,
              cubdistance_matrix = True,
              plot = True,
              interactive = True,
              size = 40 )
```

for each clusters in homologies, will compare them with a similarity matrix and a distance matrix

compare amongst the working homoset homologies, the clusters together, by what species they contains by creating a new vector of species presence in each cluster and plotting the similarity matrix of each of those vectors. –> create a compare function in homoset of homologies clusters similarity matrix and ordering. basically the distance should be nan if it has not the species, -1 if outlier to other and 1 if one cluster to another and zeros if the same to the same

Parameters

cubdistance_matrix	bool to true if want to compute the matrix of the averageCUB value distances summed for each cluster amongst the homologies
plot	bool to true to plot
size	int the size of the plot

Referenced by PyCUB.homoset.HomoSet.get_clusterstats().

5.3.3.11 compute_ages()

```
def PyCUB.homoset.HomoSet.compute_ages (
              self,
```

```
              preserved = True,
              minpreserv = 0.9,
              minsimi = 0.85 )
```

will compute whether or not a coding gene is highly preserved and if not how recent it is

it uses the phylogenetic distances and the similarities amongst homlogies to try to find a good proxy

Parameters

preserved	bool to true if we should find highly preserved genes or not
minpreserv	float minimal percentage of homologous species that have this homology
minsimi	float minimal avg similarity between genes to consider them highly preserved

longtabu

Referenced by PyCUB.homoset.HomoSet.preprocessing_names().

5.3.3.12 compute_entropyloc()

```
def PyCUB.homoset.HomoSet.compute_entropyloc (
              self,
              using = 'computejerem' )
```

called if need entropy location and used ensembl data.

you can always compute entropy location from entropy data.

```
   Will be much faster than doing it directly when calling
   ensembl's data as it computes the partition function
   only one for each lengths
```

Parameters

using	str flags the partition algorithm to use

longtabu

Referenced by PyCUB.homoset.HomoSet.preprocessing_namelist().

5.3.3.13 find_clusters()

```
def PyCUB.homoset.HomoSet.find_clusters (
              self,
              clustering = 'dbscan',
```

```
                homogroupnb = None,
                assess = True,
                eps = 0.8,
                best_eps = True,
                trainingset = 30,
                epoch = 20,
                ranges = [0.2,
                size = 10,
                redo = False )
```

Finds, for each homologies in the working homoset, groups that are part of compact clusters.

it will be using gaussian mixture clustering or DBSCAN and order them according to the density of each cluster (we are interested in the densest ones) and assess the quality using 3 criterion:BIC, number of outliers, also find if we are close to ancestry tree, here we need to represent a comparison of the closeness in a phylogenetic tree to a cluster of species –> given a grouping of phylogenetic tree, what cluster is the most similar to it

Parameters

clustering	method (DBSCAN, gaussian mixture)
homogroupnb	the number of groups can be a number or else will look for the better number of cluster according to assessments.
assess	plot or not the assessments
eps	the hyperparams
best_eps	bool to true wether or not to do an hyperparam greedy search
trainingset	int if best_eps, the size of the training ser
epoch	int the number of increasing trials
ranges	tuple[float] the two min and max values to use
size	int the x size of the plot

Referenced by PyCUB.homoset.HomoSet.compare_clusters().

5.3.3.14 findbest_eps()

```
def PyCUB.homoset.HomoSet.findbest_eps (
                self,
                trainingset = 400,
                clustering = "dbscan",
                epoch = 20,
                ranges = [0.2,
                size = 10,
                redo = False )
```

will find the best eps hyperparameter (the one that minimizes the evolutionary distance within its clusters)

Parameters

trainingset	int the number of homologies in your training set (should be 20% of the total)
clustering	str flags the clustering algorithm from which to find the best hyperparam
epoch	int the number of increasing trials
ranges	tuple[float] the two min and max values to use
size	int the x size of the plot
redo	wether or not to recompute everyhting if it has already been computed once

```
Returns;
```

the best value (often a float)

longtabu

Referenced by PyCUB.homoset.HomoSet.compare_clusters().

5.3.3.15 get_clusterstats()

```
def PyCUB.homoset.HomoSet.get_clusterstats (
              self,
              sort = True,
              interactive = True,
              redo = False )
```

will find the number of cluster i per homologies and per species

plot for each species, how much its genes are outliers, how much are belonging to a secondary cluster and how much are belonging to the principal cluster. −> create a long stacked bar plot with these values

Parameters

sort	bool to true to sort everyhting according to the statistics differences
interactive	bool to true to have an interactive barplot
redo	bool to true not to reused cached data

longtabu

Referenced by PyCUB.homoset.HomoSet.cluster_homologies().

5.3.3.16 loadfullhomo()

```
def PyCUB.homoset.HomoSet.loadfullhomo (
              self )
```

function to concatenate all the homologies in one big array(practicle for certain computations)

Parameters

None	

Referenced by PyCUB.homoset.HomoSet.__len__(), PyCUB.homoset.HomoSet.compute_ages(), and PyCUB.↩homoset.HomoSet.plot_all().

5.3.3.17 loadhashomo()

```
def PyCUB.homoset.HomoSet.loadhashomo (
              self,
              withnames = False )
```

function to compute the matrix of bool saying wether species X has a gene or more in homology Y

Parameters

None	

Referenced by PyCUB.homoset.HomoSet.loadfullhomo(), and PyCUB.homoset.HomoSet.plot_all().

5.3.3.18 orderfromclust()

```
def PyCUB.homoset.HomoSet.orderfromclust (
              self,
              homogroupnb,
              clust,
              byspecie = False,
              plot_ordering = True )
```

creates an ordering of every elements

(names, homologies according to the found clusters) from an ordered cluster of species or of homologies self.↩ hashomomatrix should reflect this orientation as well

Parameters

homogroupnb	int the number of clusters
clust	np.array(int) the clusters
byspecie	bool to true to order by species
plot_ordering	bool to true to plot the ordering

Returns

the ordered hashomomatrix (np.array[bool])

Referenced by PyCUB.homoset.HomoSet.cluster_homologies().

5.3.3.19 plot_all()

```
def PyCUB.homoset.HomoSet.plot_all (
              self,
              With = 'tsne',
              perplexity = 60,
```

```
                interactive = False,
                bins = 100,
                offset = 20,
                iteration = 400,
                redo = False,
                bypasstsne = False,
                dotsize = 7,
                inpar = True )
```

will plot all the homologies in the full_homo_matrix (and compute it)

(sometimes around 800 000 datapoints) to look at any kind of relationships as there is too much datapoints, the plots are density ones.

Parameters

With	flag the dim reduction algorithm to use (tsne: need >16gigs of RAM,now use another version of tsne for large datasets)(PCA: works well)(lsta/hessian:untested)
perplexity	ints of basic tsne hyperparams
interactive	bool if true should use bokeh else matplotlib

Returns

the desired plot if the size is high and we are interactive

longtabu

Referenced by PyCUB.homoset.HomoSet.__delitem__().

5.3.3.20 plot_distcub()

```
def PyCUB.homoset.HomoSet.plot_distcub (
                self,
                interactive = False,
                size = 40 )
```

plot the distance matrix of each homologies from the average of their CUB distances

the interactive version allows you to see each particular datapoint with much more precision

Parameters

interactive	bool to use the bokeh interactive version
size	size of the matrix

Referenced by PyCUB.homoset.HomoSet.compare_clusters(), and PyCUB.homoset.HomoSet.plot_hashomo().

5.3.3.21 plot_hashomo()

```
def PyCUB.homoset.HomoSet.plot_hashomo (
              self,
              invert = False,
              size = 40,
              interactive = False,
              rangeto = None )
```

plot the has homo matrix

the interactive version allows you to see each particular datapoint with much more precision

Parameters

interactive	bool to use the bokeh interactive version
size	int size of the matrix
invert	bool flag to true to invert the plot

Referenced by PyCUB.homoset.HomoSet._barplot().

5.3.3.22 plot_simiclust()

```
def PyCUB.homoset.HomoSet.plot_simiclust (
              self,
              interactive = True,
              size = 40 )
```

plot the similarity matrix of each homologies from its clusters

the interactive version allows you to see each particular datapoint with much more precision

Parameters

interactive	bool to use the bokeh interactive version
size	size of the matrix

Referenced by PyCUB.homoset.HomoSet.get_clusterstats(), and PyCUB.homoset.HomoSet.plot_hashomo().

5.3.3.23 preprocessing()

```
def PyCUB.homoset.HomoSet.preprocessing (
              self,
              withtaxons = False,
              withnames = True )
```

will compute the full list of names,

find doublons, and set the names to ints instead of strings. called after loading from ensembl and associate namelist in each homologies to a number in utils.speciestable

Parameters

withtaxons	bool to true calls preprocessing_taxons() else one of the other two
withnames	bool to true to call preprocessing_names() else calls preprocessing_namelist()

Returns

taxons list, if the names contains an additional list of taxon ids. and the corresponding species in the same order only if withtaxons

Referenced by PyCUB.homoset.HomoSet.loadhashomo().

5.3.3.24 preprocessing_namelist()

```
def PyCUB.homoset.HomoSet.preprocessing_namelist (
              self )
```

same as preprocessing_names() but without preprocessing the names of each homologies only updating the species_namelist

Parameters

None	

Referenced by PyCUB.homoset.HomoSet.add_random_homology(), and PyCUB.homoset.HomoSet.preprocessing↩_taxons().

5.3.3.25 preprocessing_names()

```
def PyCUB.homoset.HomoSet.preprocessing_names (
              self )
```

same as preprocessing_taxons() but admiting there is no taxon information (Yun's data for example)

Parameters

None	

Referenced by PyCUB.homoset.HomoSet.add_random_homology(), and PyCUB.homoset.HomoSet.preprocessing().

5.3.3.26 preprocessing_taxons()

```
def PyCUB.homoset.HomoSet.preprocessing_taxons (
              self )
```

preprocess the data by computing the names as ints

creates a speciestable to find the corresponding names computes the doublons and compute the species_namelist of this homoset will returns the species and the corresponding taxons extracted from the homology.names

Parameters

None	

Referenced by PyCUB.homoset.HomoSet.add_random_homology().

5.3.3.27 remove()

```
def PyCUB.homoset.HomoSet.remove (
              self,
              species )
```

remove this list of species from the homologies

Parameters

species	list[str] the species to remove from all homologies

Referenced by PyCUB.homoset.HomoSet.compute_ages(), and PyCUB.homoset.HomoSet.compute_entropyloc().

5.3.3.28 size()

```
def PyCUB.homoset.HomoSet.size (
              self )
```

the size of the homoset (number of genes)

Parameters

None	

Returns

int the number of genes

Referenced by PyCUB.homoset.HomoSet.loadfullhomo().

5.3.3.29 update()

```
def PyCUB.homoset.HomoSet.update (
              self,
              val )
```

a function to update the dictionnary that homoset is

Parameters

val	the dict to append tot this one

Referenced by PyCUB.homoset.HomoSet.__delitem__(), PyCUB.homoset.HomoSet.__init__(), PyCUB.homoset.HomoSet.__setitem__(), and PyCUB.homoset.HomoSet.loadhashomo().

5.3.4 Member Data Documentation

5.3.4.1 wasclusterized

```
PyCUB.homoset.HomoSet.wasclusterized = False  [static]
```

bool if the homologies have been clustered or not.

usefull for processing requiring clusters

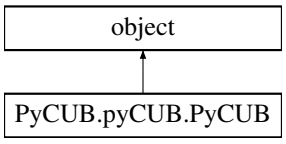
Referenced by PyCUB.homoset.HomoSet.__init__(), PyCUB.homoset.HomoSet._barplot(), PyCUB.homoset.HomoSet.compare_clusters(), and PyCUB.homoset.HomoSet.orderfromclust().

The documentation for this class was generated from the following file:

- PyCUB/homoset.py

5.4 PyCUB.pyCUB.PyCUB Class Reference

Inheritance diagram for PyCUB.pyCUB.PyCUB:



Public Member Functions

- def __init__ (self, species={}, _is_saved=False, _is_loaded=False, working_homoset=None, all_↩ homoset=None, session='session1')

  will initialize the object with the different values you might have from another project
- def getHomologylist (self, species='saccharomyces_cerevisiae', kingdom='fungi')

  A function to retrieve the homologies directly from a given species.
- def get_data (self, From='yun', homonames=None, kingdom='fungi', sequence='cdna', additional='type=orthologues', saveonfiles=False, normalized=True, setnans=False, by="entropy", using="normal", tRNA=True, getC↩ AI=True, first=20, inpar=True)

  Download the data from somewhere on the web (Ensembl, Yun(with links))
- def get_metadata_Ensembl (self, kingdoms)

  download it and put it where it belongs in the Espece object
- def get_mymetadata (self, From='jerem', inpar=True)

  Go ahead and design your own metadata retrieval here.
- def import_metadataTobias (self)

  will import the metadata obtained from tobias for the fungi species affiliated to cerevisiae to each species for further diagnostics.
- def load (self, session=None, All=False, filename='first500', From=None, by='entropy', tRNA=True, inpar=True)

  Get the data that is already present on a filename.
- def loadmore (self, filename='first500', by='entropyLocation')

  Get the data that is already present on a filename when you already have data.
- def save (self, name, save_workspace=True, save_homo=True, add_homosets={}, cmdlinetozip="gzip")

  call to save your work.
- def get_working_homoset (self, clusternb=None, species=None, homologies=None, cleanhomo=None)

  create a subset of all_homoset on which you would like to do further computation
- def get_subset (self, homoset, withcopy=False, clusternb=None, species=None, homologies=None)

  either changes or returns a subset of the provided homoset
- def get_full_genomes (self, kingdom='fungi', seq='cds', avg=True, by="entropy", normalized=False)

  go trought all full genome fasta files in the ftp server of ensemblgenomes and
- def get_taxons (self)

  find the taxons of each referenced species (see PyCUB.Espece.gettaxons())
- def get_evolutionary_distance (self, display_tree=False, size=40)

  uses metadata of the ancestry tree and computes a theoretical evolutionary distance matrix between each species
- def createRefCAI (self, speciestocompare='saccharomyces_cerevisiae', kingdom='fungi', first=20)

  do a compute CAI
- def speciestable (self)

  a copy of the utils.speciestable
- def phylo_distances (self)

  a copy of the phylodistances dataframe see (get_evolutionary_distance())
- def compute_averages (self, homoset)

  compute the average entropy
- def compare_species (self, showvar=True, reducer='tsne', perplexity=40, eps=0.3, size=10)

  compare the species according to their mean CUB,
- def **compute_ages** (self, homoset, preserved=True, minpreserv=0.9, minsimi=0.85)
- def regress_on_species (self, without=[""], full=True, onlyhomo=False, perctrain=0.8, algo="lasso", eps=0.↩ 001, n_alphas=100)

  Will fit a regression curve on the CUB values of the different species according to the metadatas available for each of them.
- def compare_homologies (self, homoset, homosapiens=False, mindistance=10, preserved=True, size=10, minpreserv=0.9, minsimi=0.9, showvar=True, eps=0.28, reducer='tsne', perplexity=40)

finds for species with a common ancester separated by a pseudo phylogenetic distance X,

- def regress_on_genes (self, homoset, full=True, without=['meanecai', meancai, perctrain=0.8, algo="lasso", eps=0.001, n_alphas=100)

  Will fit a regression curve on the CUB values of the different homologies according to the metadatas available for each of them.

- def getRelation2G3DD (self, species_name='saccharomyces_cerevisiae', kingdom='fungi', intrachromosome="utils/meta/3Dmodel/interactions_HindIII_fdr0.01_intra_cerevisiae.csv", interchromose=["utils/meta/3Dmodel/cerevisiae_inter1.csv", utils, meta, Dmodel, cerevisiae_inter2, csv, utils, meta, Dmodel, cerevisiae_inter3, csv, utils, meta, Dmodel, cerevisiae_inter4, csv, utils, meta, Dmodel, cerevisiae_inter5, csv, bins=2000, seq='cds', use='diament2', euclide=False)

  https://www.nature.com/articles/ncomms6876

- def plot_distances (self, size=40)

  plot the phylogenetic distance matrix

- def loadspeciestable (self)

  short function to retrieve the speciestable from Disk

- def savespeciestable (self)

  short function to put the speciestable on Disk

Public Attributes

- **species**
- **homolist**
- pent

  float from the scoring of spearman's rho for entropy

- pcub

  float from the scoring of spearman's rho for CUB

- pcuf

  float from the scoring of spearman's rho for CUF

Static Public Attributes

- dictionary links

  dict of all the links readily available in PyCUB.

- dictionary species = {}

  dictionary of Espece objects from the name of the species.

- working_homoset = None

  PyCUB.homoset object that stores a subset of the homologies.

- **all_homoset** = None
- session = None

  str the session name you want to use (will appear in the savings for example

- coeffgenes = None

  np.array regressing values for each attributes

- scoregenes = None

  the score of the regressor

- scorespecies = None

  the score of the regressor

- coeffspecies = None

  np.array regressing values for each attributes

Private Member Functions

- def _dictify (self, save_workspace, save_homo, add_homosets)

  Used by the saving function.
- def _undictify (self, data)

  same function but to retransform everything

Private Attributes

- **_is_saved**
- **_is_loaded**

Static Private Attributes

- bool _is_saved

  bool trivial system only boolean

5.4.1 Constructor & Destructor Documentation

5.4.1.1 __init__()

```
def PyCUB.pyCUB.PyCUB.__init__ (
              self,
              species = {},
              _is_saved = False,
              _is_loaded = False,
              working_homoset = None,
              all_homoset = None,
              session = 'session1' )
```

will initialize the object with the different values you might have from another project

Parameters

species	dictionary of Espece objects from the name of the species. (see espece.py)
working_homoset	PyCUB.homoset object that stores a subset of the homologies you want to work on all_homoset PyCUB.homoset that stores the all the homologies
session	str the session name you want to use (will appear in the savings for example
_is_saved	bool trivial system only boolean

5.4.2 Member Function Documentation

5.4.2.1 _dictify()

```
def PyCUB.pyCUB.PyCUB._dictify (
              self,
              save_workspace,
              save_homo,
              add_homosets )  [private]
```

Used by the saving function.

transform the workspace object into a dictionary that can be json serializable

```
   adding some params because else the object may be too big
```

Parameters

save_workspace	bool to save working_homoset
save_homo	bool to save all_homoset
add_homosets	PyCUB.homoset instances to add to this dict

Return

A dict holding every element to be jsonized

Referenced by PyCUB.pyCUB.PyCUB.loadmore().

5.4.2.2 _undictify()

```
def PyCUB.pyCUB.PyCUB._undictify (
              self,
              data )  [private]
```

same function but to retransform everything

Here we don't use other classes undictify functions but we just recreate them by passing it to their init methods which is clearer.

Parameters

data	dict to undictify into the workspace object

Returns

Other PyCUB.homosets that would have been saved as add_homosets

Referenced by PyCUB.pyCUB.PyCUB.load().

5.4.2.3 compare_homologies()

```
def PyCUB.pyCUB.PyCUB.compare_homologies (
              self,
              homoset,
              homosapiens = False,
              mindistance = 10,
              preserved = True,
              size = 10,
              minpreserv = 0.9,
              minsimi = 0.9,
              showvar = True,
              eps = 0.28,
              reducer = 'tsne',
              perplexity = 40 )
```

finds for species with a common ancester separated by a pseudo phylogenetic distance X,

genes/functions that are novel to only a subset. plot the two with their differences the differences between the two also considers an homology as highly preserved if it is shared amongst most of the species and if the average similarity score is high amongst this homology. also shows if there is a relationship between the number of amino acids the sequence does not encode for and the codon usage bias We could have used the sequence dating of ensembl but it only works for homo sapiens for now maybe use it for homosapiens later

Parameters

homosapiens	bool to true if we should use homosapiens dataset on gene dates
mindistance	int the minimal phylogenetic distance between in average in this homology to consider it highly conserved
preserved	bool to true if we should find highly preserved genes or not
size	the average size of the datapoints in the pointcloud representation of this dataset
minpreserv	float minimal percentage of homologous species that have this homology
minsimi	float minimal avg similarity between genes to consider them highly preserved
showvar	bool to true, show the mean variance in CUB values accros this homology as a variation in dot sizes
eps	float the hyperparamter of the clustering algorithm applied to this dataset
homoset	PyCUB.homoset the homoset to use
reducer	str the reducer to use 'tsne' or 'PCA'
perplexity	int the perplexity hyperparam for tSNE

longtabu

Referenced by PyCUB.pyCUB.PyCUB.regress_on_species().

5.4.2.4 compare_species()

```
def PyCUB.pyCUB.PyCUB.compare_species (
              self,
              showvar = True,
```

```
                reducer = 'tsne',
                perplexity = 40,
                eps = 0.3,
                size = 10 )
```

compare the species according to their mean CUB,

plot the mean CUB to their full CUB, to their tRNA copy numbers, to the euclidean distance of their CUB to the one of their phylogenetic matrix. in this plot, the mean entropy value is plotted as a regular homology plot but each dot is a species thus we can compare them together, moreover, the size of the dots informs oneself of the variance in entropy per species. the color intensity informs on how much this is close to what is given by the tRNA values. (additional information such as name of the species, number of tRNA values,metadata and the point from its tRNA value is plotted when hovering the dot) then we can also compare the mean of homologies or else, the full entropy of the cdna sequence per species is also computed the euclidean distance amongst the species for the full entropy to see if a difference can be linked to some evolutionary for one codon

Parameters

size	the average size of the datapoints in the pointcloud representation of this dataset
showvar	bool to true, show the mean variance in CUB values accros this homology as a variation in dot sizes
eps	float the hyperparamter of the clustering algorithm applied to this dataset
reducer	str the reducer to use 'tsne' or 'PCA'
perplexity	int the perplexity hyperparam for tSNE

longtabu

Referenced by PyCUB.pyCUB.PyCUB.phylo_distances().

5.4.2.5 compute_averages()

```
def PyCUB.pyCUB.PyCUB.compute_averages (
                self,
                homoset )
```

compute the average entropy

Will add species related averages gotten from this homoset in the species container, and in the homoset everytime you compute averages from a set, they will get erased !

Parameters

homoset	PyCUB.homoset from which to compute the averages

Referenced by PyCUB.pyCUB.PyCUB.createRefCAI().

5.4.2.6 createRefCAI()

```
def PyCUB.pyCUB.PyCUB.createRefCAI (
                self,
```

```
                speciestocompare = 'saccharomyces_cerevisiae',
                kingdom = 'fungi',
                first = 20 )
```

do a compute CAI

where we get Tobias' data to find highly expressed genes and use them to compute codon frequency for the reference set and use it to compute the CAI and mean CAI for each ho mology.

Parameters

speciestocompare	str the name of the species to retrieve the genes from
kingdom	str the kingdom where we can find it
first	the number of highly expressed genes to retrieve

Referenced by PyCUB.pyCUB.PyCUB.get_evolutionary_distance(), and PyCUB.pyCUB.PyCUB.getHomologylist().

5.4.2.7 get_data()

```
def PyCUB.pyCUB.PyCUB.get_data (
                self,
                From = 'yun',
                homonames = None,
                kingdom = 'fungi',
                sequence = 'cdna',
                additional = 'type=orthologues',
                saveonfiles = False,
                normalized = True,
                setnans = False,
                by = "entropy",
                using = "normal",
                tRNA = True,
                getCAI = True,
                first = 20,
                inpar = True )
```

Download the data from somewhere on the web (Ensembl, Yun(with links))

you can provide a lot of different values to scrape Ensembl's datasets it will compute from ensembl to retrieve a similar dataset as what yun's data is.

Parameters

From	str flag 'yun' or 'ensembl':
homonames	list[str] what particular homologies you want to scrap if 'all' and you have used the getHomologylist() function, will get the homologies from there
kingdom	str same for kingdoms
sequence	str the type of sequences you want to use
additional	str additional information about the scrapping
saveonfiles	bool save the unprocessed data before populating working homoset
normalized	bool if you want the values to be normalized by the length of the codons (lengths are always saved)

Parameters

setnans	bool if you want to save the nans as metadata
by	str flag 'entropy', 'entropyLocation' (entropy location), 'frequency'
using	str flag 'random' 'normal' 'permutation' 'full'
inpar	bool or int for parallel computing and number of core
tRNA	bool whether or not to compute tRNA data
getCAI	bool flag to true to retrieve the CAI as well
first	int the first most expressed genes to compute the CAI ref statistics

longtabu

Referenced by PyCUB.pyCUB.PyCUB.__init__().

5.4.2.8 get_evolutionary_distance()

```
def PyCUB.pyCUB.PyCUB.get_evolutionary_distance (
              self,
              display_tree = False,
              size = 40 )
```

uses metadata of the ancestry tree and computes a theoretical evolutionary distance matrix between each species

can optionaly take any hierarchical evolutionary file between a group of species will populate utils.phylo_distances with a pandas.df of the phylodistance and meandist with the average distance amongst species in the df, species are referenced by their taxon ids. you have to have taxons in your species. will also plot the distance matrix

Parameters

display_tree	bool to true to print the phylogenetic tree as a txt (may be quite big)
size	int the x size of the plot

longtabu

Referenced by PyCUB.pyCUB.PyCUB.get_full_genomes().

5.4.2.9 get_full_genomes()

```
def PyCUB.pyCUB.PyCUB.get_full_genomes (
              self,
              kingdom = 'fungi',
              seq = 'cds',
              avg = True,
              by = "entropy",
              normalized = False )
```

go trought all full genome fasta files in the ftp server of ensemblgenomes and

download then parse them to get the full entropy of the genome. usefull for futher comparison steps. will populate the fullentropy, fullvarentropy, fullGCcount, varGCcount of each species where the full sequence is known

Parameters

kingdom	str flags the relevant kingdom of you current session [fungi,plants,bacteria, animals]
seq	str flags the type of sequence you consider the full genome is (coding or non coding or full) [cds, all, cda]
avg	bool to true if we average over each gene or get the full dna in one go.
by	str flags what type of computation should be done [entropy,frequency]
normalized	should we normalize the entorpy by length

longtabu

Referenced by PyCUB.pyCUB.PyCUB.get_working_homoset().

5.4.2.10 get_metadata_Ensembl()

```
def PyCUB.pyCUB.PyCUB.get_metadata_Ensembl (
              self,
              kingdoms )
```

download it and put it where it belongs in the Espece object

parse the server https://fungi.ensembl.org/info/website/ftp/index.html will also get the metadata from the kingdoms that you are analysing

Parameters

kingdoms	str flag the type of kingdoms you wanna have 'fungi' 'bacteria' 'plants' 'animals'

Referenced by PyCUB.pyCUB.PyCUB.get_data().

5.4.2.11 get_mymetadata()

```
def PyCUB.pyCUB.PyCUB.get_mymetadata (
              self,
              From = 'jerem',
              inpar = True )
```

Go ahead and design your own metadata retrieval here.

obviously you woud need to change some other functions. for me it is mean protein abundances in cerevisiae cells.

Parameters

From	str flag designer of the function to load metadatas
inpar	bool for parallel processing

Referenced by PyCUB.pyCUB.PyCUB.get_data().

5.4.2.12 get_subset()

```
def PyCUB.pyCUB.PyCUB.get_subset (
              self,
              homoset,
              withcopy = False,
              clusternb = None,
              species = None,
              homologies = None )
```

either changes or returns a subset of the provided homoset

To use once if you want to further refine a set of homologies

Parameters

homoset	PyCUB.homoset to get a subset from
withcopy	bool to true if we don't want to change the homoset object but create a copy from it
clusternb	int set the cluster of the group you want to get need to be between 1 and homogroupnb
homologies	list[str] the subset as a list or a tuple of int
species	list[the subset as a list, or a list of int

Returns

a HomoSet object (see homoset.py)

Referenced by PyCUB.pyCUB.PyCUB.get_working_homoset().

5.4.2.13 get_taxons()

```
def PyCUB.pyCUB.PyCUB.get_taxons (
              self )
```

find the taxons of each referenced species (see PyCUB.Espece.gettaxons())

Referenced by PyCUB.pyCUB.PyCUB.get_full_genomes().

5.4.2.14 get_working_homoset()

```
def PyCUB.pyCUB.PyCUB.get_working_homoset (
              self,
              clusternb = None,
              species = None,
              homologies = None,
              cleanhomo = None )
```

create a subset of all_homoset on which you would like to do further computation

To use once you have clustered homology groups, else takes everything. Can also be used just to get a subset of the all homosets.

Parameters

clusternb	int set the cluster of the group you want to get need to be between 1 and homogroupnb
homologies	list[str] the subset as a list you want to get from all_homoset (can be additional to a clusternb)
species	list[str] the subset as a list you want to get from all_homoset (can be additional to a clusternb)
cleanhomo	float if the homology is only shared by less than this amount amongst the species present in this homoset, removes them.

Returns

a HomoSet object (see homoset.py)

longtabu

Referenced by PyCUB.pyCUB.PyCUB.loadmore().

5.4.2.15 getHomologylist()

```
def PyCUB.pyCUB.PyCUB.getHomologylist (
              self,
              species = 'saccharomyces_cerevisiae',
              kingdom = 'fungi' )
```

A function to retrieve the homologies directly from a given species.

(it is better to use one of the key species for the different kingdoms (sacharomyces, HS, Arabidopsis..))

Parameters

specie	str the name of the specie to get the homology from
kingdom	str the kingdom where we can find this specie

5.4.2.16 getRelation2G3DD()

```
def PyCUB.pyCUB.PyCUB.getRelation2G3DD (
                self,
                species_name = 'saccharomyces_cerevisiae',
                kingdom = 'fungi',
                intrachromosome = "utils/meta/3Dmodel/interactions_HindIII_fdr0.01_intra_↩
cerevisiae.csv",
                interchromose = ["utils/meta/3Dmodel/cerevisiae_inter1.csv",
                utils,
                meta,
                Dmodel,
                cerevisiae_inter2,
                csv,
                utils,
                meta,
                Dmodel,
                cerevisiae_inter3,
                csv,
                utils,
                meta,
                Dmodel,
                cerevisiae_inter4,
                csv,
                utils,
                meta,
                Dmodel,
                cerevisiae_inter5,
                csv,
                bins = 2000,
                seq = 'cds',
                use = 'diament2',
                euclide = False )
```

https://www.nature.com/articles/ncomms6876

retrieve the data for the species sacharomyces cerevisiae and Schizosaccharomyces pombe and find if similarity distances of CUB using entropy between genes of this species is predictive of closeness of genes in the nucleus.

Used to confirm a work on nature and see if we can have some similar results by only looking at the CUB

Parameters

species_name	str the name of the species to look for
kingdom	str the kingdom in which to find the species
intrachromosome	str the location of the csv interaction data for intrachromosome respecting the format of the default file
interchromose	str the location of the csv interaction data for interchromose respecting the format of the default file
bins	int, the number of bin to use (a power of 2)
seq	the type of sequence to compare to. (to compute the CUB from)
use	str flag different types of algorithm I have made trying to understand the thing
compute	str flag to different computation available
euclidean	bool flag to true to compute euclidean instead of Endres Shcidelin metrics

Referenced by PyCUB.pyCUB.PyCUB.regress_on_genes().

5.4.2.17 import_metadataTobias()

```
def PyCUB.pyCUB.PyCUB.import_metadataTobias (
              self )
```

will import the metadata obtained from tobias for the fungi species affiliated to cerevisiae to each species for further diagnostics.

Populates metadata[num_genes, plant_pathogen, animal_pathogen, genome_size, plant_symbiotic, brown_↩ rot, white_rot] for each species and weight, mRNA_abundance, is_secreted, protein_abundance, cys_elements, decay_rate for each homology

Referenced by PyCUB.pyCUB.PyCUB.get_data().

5.4.2.18 load()

```
def PyCUB.pyCUB.PyCUB.load (
              self,
              session = None,
              All = False,
              filename = 'first500',
              From = None,
              by = 'entropy',
              tRNA = True,
              inpar = True )
```

Get the data that is already present on a filename.

Either load from Yun's datasets or from an already saved session. Is being called by get_data. But you can call it to just use one of Yun's files as well

Parameters

From	str if this flag is set to 'yun' it means that the filename is made of Yundata in which case we will create directly the homology map in the same time as the rest of the PyCUB object.
All	bool set to true if load everything from Yun
by	str same flag as get_data (for Yun's files here).
filename	str the particular filename when not loading them all
session	str if a session name is provided, then will load a zip file from this session's folder
tRNA	bool to true to compute the tRNA values
inpar	int to set the number of processor (as in scikit)

Returns

May return additionals if loading from a session where one decided to save more than the two All/working homologies. to be handled separately

Referenced by PyCUB.pyCUB.PyCUB.getHomologylist(), and PyCUB.pyCUB.PyCUB.import_metadataTobias().

5.4.2.19 loadmore()

```
def PyCUB.pyCUB.PyCUB.loadmore (
              self,
              filename = 'first500',
              by = 'entropyLocation' )
```

Get the data that is already present on a filename when you already have data.

is usefull to load more of Yun's datasets. is called when load is set to All

Parameters

filename	str the filename to additionaly load
by	flag same as before

longtabu

Referenced by PyCUB.pyCUB.PyCUB.load().

5.4.2.20 loadspeciestable()

```
def PyCUB.pyCUB.PyCUB.loadspeciestable (
              self )
```

short function to retrieve the speciestable from Disk

longtabu

Referenced by PyCUB.pyCUB.PyCUB._dictify().

5.4.2.21 phylo_distances()

```
def PyCUB.pyCUB.PyCUB.phylo_distances (
              self )
```

a copy of the phylodistances dataframe see (get_evolutionary_distance())

Returns

a copy of the phylodistances dataframe see (get_evolutionary_distance())

Referenced by PyCUB.pyCUB.PyCUB.compare_species(), PyCUB.pyCUB.PyCUB.compute_averages(), and PyCUB.pyCUB.PyCUB.createRefCAI().

5.4.2.22 plot_distances()

```
def PyCUB.pyCUB.PyCUB.plot_distances (
              self,
              size = 40 )
```

plot the phylogenetic distance matrix

Parameters

size	int the x size of the plot

longtabu

Referenced by PyCUB.pyCUB.PyCUB.get_evolutionary_distance().

5.4.2.23 regress_on_genes()

```
def PyCUB.pyCUB.PyCUB.regress_on_genes (
              self,
              homoset,
              full = True,
              without = ['meanecai',
              meancai,
              perctrain = 0.8,
              algo = "lasso",
              eps = 0.001,
              n_alphas = 100 )
```

Will fit a regression curve on the CUB values of the different homologies according to the metadatas available for each of them.

It will try to see if there is enough information in the metadata to retrieve CUB values. and if there is, how much for each metadata (if we constraint the number of regressors) is it better for entropy values, mean entropy or ECAI values or raw frequency, should we remove some data

Parameters

without	list[str] of flags [similarity_scores, KaKs_Scores, nans, lenmat, GCcount, weight, protein_abundance, mRNA_abundance, decay_rate, cys_elements, tot_volume, mean_hydrophobicity, glucose_cost, synthesis_steps, is_recent, meanecai]
full	bool flags to true to use full CUB values or meanCUB values, as regressee
homoset	PyCUB.homoset the homoset to use
perctrain	the percentage of training set to total set (the rest is used as test set)
algo	str flag to lasso or nn to use either Lasso with Cross Validation, or a 2 layer neural net
eps	the eps value for the Lasso
n_alphas	the number of alphas for the lasso

Returns

scoregenes float, the score of the regression performed
coeffgenes the coefficient applied to each category (for each CUB value if using full)
attrlist the corresponding list[str] of attribute used

longtabu

Referenced by PyCUB.pyCUB.PyCUB.compare_homologies().

5.4.2.24 regress_on_species()

```
def PyCUB.pyCUB.PyCUB.regress_on_species (
              self,
              without = [""],
              full = True,
              onlyhomo = False,
              perctrain = 0.8,
              algo = "lasso",
              eps = 0.001,
              n_alphas = 100 )
```

Will fit a regression curve on the CUB values of the different species according to the metadatas available for each of them.

It will try to see if there is enough information in the metadata to retrieve CUB values. and if there is, how much for each metadata (if we constraint the number of regressors) is it better for mean homology CUB or full genome CUB ? or raw frequency, should we remove some data?

Parameters

without	list[str] of flags [similarity_scores, KaKs_Scores, nans, lenmat, GCcount, weight, protein_abundance, mRNA_abundance, decay_rate, cys_elements, tot_volume, mean_hydrophobicity, glucose_cost, synthesis_steps, is_recent, meanecai]
onlyhomo	bool to true if want to use only CUB from homologies
full	bool flags to true to use full CUB values or meanCUB values, as regressee
perctrain	the percentage of training set to total set (the rest is used as test set)
algo	str flag to lasso or nn to use either Lasso with Cross Validation, or a 2 layer neural net
eps	the eps value for the Lasso
n_alphas	the number of alphas for the lasso

Returns

scoregenes float, the score of the regression performed
coeffgenes the coefficient applied to each category (for each CUB value if using full)
attrlist the corresponding list[str] of attribute used

longtabu

Referenced by PyCUB.pyCUB.PyCUB.compare_species().

5.4.2.25 save()

```
def PyCUB.pyCUB.PyCUB.save (
              self,
              name,
              save_workspace = True,
              save_homo = True,
              add_homosets = {},
              cmdlinetozip = "gzip" )
```

call to save your work.

you should call save on specific data structure if this is what you want to save.

```
Will call other object's save, will transform all the variable into dict and save the dicts
as json files. will save the df also as json files. PyCUB and homoset have
their own json file.
adding some params because else the object may be too big
```

Parameters

name	str the name of the particular save on this session
save_workspace	bool to fale not to save working_homoset
save_homo	bool to false not to save all_homoset add_homosets= PyCUB.homoset homoset to add in addition to the regular ones
cmdlinetozip	str you need to tell the platform how to zip on your system uses gzip by default but it needs to be installed

Referenced by PyCUB.pyCUB.PyCUB.compare_homologies(), PyCUB.pyCUB.PyCUB.compare_species(), and PyCUB.pyCUB.PyCUB.loadmore().

5.4.2.26 savespeciestable()

```
def PyCUB.pyCUB.PyCUB.savespeciestable (
              self )
```

short function to put the speciestable on Disk

This is done since there may be some memory leakage, probably due to some autoreloading behavior of the global data stored on utils.

Referenced by PyCUB.pyCUB.PyCUB._dictify().

5.4.2.27 speciestable()

```
def PyCUB.pyCUB.PyCUB.speciestable (
              self )
```

a copy of the utils.speciestable

Returns

a copy of the utils.speciestable (dict[int,str] of species to their PyCUB coded value

Referenced by PyCUB.pyCUB.PyCUB.get_evolutionary_distance().

5.4.3 Member Data Documentation

5.4.3.1 links

`PyCUB.pyCUB.PyCUB.links [static]`

dict of all the links readily available in PyCUB.

Referenced by PyCUB.pyCUB.PyCUB.get_data(), and PyCUB.pyCUB.PyCUB.getHomologylist().

5.4.3.2 species

`PyCUB.pyCUB.PyCUB.species = {} [static]`

dictionary of Espece objects from the name of the species.

Referenced by PyCUB.pyCUB.PyCUB.compare_species(), PyCUB.pyCUB.PyCUB.compute_averages(), PyCUB.pyCUB.PyCUB.createRefCAI(), PyCUB.pyCUB.PyCUB.get_data(), PyCUB.pyCUB.PyCUB.get_full_genomes(), PyCUB.pyCUB.PyCUB.get_metadata_Ensembl(), PyCUB.pyCUB.PyCUB.get_taxons(), PyCUB.pyCUB.PyCUB.load(), PyCUB.pyCUB.PyCUB.plot_distances(), PyCUB.pyCUB.PyCUB.regress_on_species(), and PyCUB.pyCUB.PyCUB.speciestable().

The documentation for this class was generated from the following file:

- PyCUB/pyCUB.py