IPTK

Release 0.4.6

Hesham ElAbd

Jul 17, 2021

CONTENTS

Analyzing, Visualizing, Comparing and Integrating Immunopeptidomics data!

CHAPTER

ONE

INTRODUCTION:

IPTK is a Python library specialized in the analysis of HLA-peptidomes identified through an immunopeptidomic(IP) pipeline. The library provides a high level API for analyzing and visualizing the identified peptides, integrating transcriptomics and protein structure information for a rich analysis of the identified immunopeptidomes. It also provides a toolbox for integrating and comparing different experiments and/or different mass-spectrometry runs.

CHAPTER

TWO

INSTALLATION:

The library can be installed using

```
` pip install iptkl --user `
```

CHAPTER

THREE

FUNDING:

The project was funded by the German Research Foundation (DFG) (Research Training Group 1743, ‘Genes, Environment and Inflammation’).

3.1 Guide

3.1.1 License

3.1.2 Contact

for further questions and communication please contact h.elabd@ikmb.uni-kiel.de

3.1.3 Get Started!

To get started with using the library check the interactive tutorials available at https://github.com/ikmb/iptoolkit/tree/master/Tutorials

3.1.4 IPTK

3.1.4.1 IPTK package

Subpackages

IPTK.Analysis package

Submodules

IPTK.Analysis.AnalysisFunction module

Module contents

IPTK.Classes package

Submodules

IPTK.Classes.Annotator module

The class provides methods for visualizing different aspects of the protein biology. This is achieved through three main methods:

1- add_segmented_track: which visualize information about non-overlapping protein substructures, for example, protein domains.

2- add_stacked_track: which visualize information about overlapping protein substructures, for example, splice variants.

3- add_marked_positions_track: which visualize or highlight positions in the protein, for example, sequence variants, or PTM.

The class also provides functions for visualizing the relationship between a protein and its eluted peptide/peptides in an analogous manner to the way NGS reads are aligned to genomic regions. This can be useful to identify regions in the protein with high/low number of eluted peptides, i.e.,Coverage. Also, to link it with other facests of the protein like domain organization,PTM, sequence/splice variants.

Notes

each figure should have a base track this can be done explicitly by calling the function add_base_track or by implicitly by calling the function add_coverage_plot with the parameter coverage_as_base=True.

class IPTK.Classes.Annotator.**Annotator**(*protein_length: int*, *figure_size: Tuple[int, int]*, *figure_dpi: int*, *face_color='white'*)

Bases: `object`

A high level API to plot information about the protein, for example, PTM, Splice variant etc, using matplotlib library

add_base_track(*space_fraction: float = 0.3*, *protein_name_position: float = 0.5*, *track_label: str = 'base_track'*, *track_label_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 8}*, *protein_name: str = 'A protein'*, *protein_name_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 10}*, *rect_dict: Dict[str, Union[int, str]] = {'capstyle': 'butt', 'color': 'olive'}*, *number_ticks: int = 10*, *xticks_font_size: int = 4*)

Adds a base track to the figure.

Parameters

- **space_fraction** (*float, optional*) – A float between 0 and 1 that represent the fraction of space left below and above the track. The default is 0.3 which means that the track will be drown on a 40% while 60% are left as an empty space below and above the track.
- **protein_name_position** (*float, optional*) – A float between 0 and 1 which control the relative position of the protein name on the y-axis. The default is 0.5.
- **track_label** (*string, optional*) – The name on the track, which will be shown on the y-axis. The default is “base_track”.
- **track_label_dict** (*Dict[str,Union[int,str]], optional*) – The parameters that control the printing of the track_label, for example, the font size and the color. These parameters should be provided as dict that will be fed to the function `axes.set_ylabel`.The default is {“fontsize”:8,”color”:”black”}.
- **protein_name** (*string, optional*) – The name of the protein to be printed to the track. The default is “A protein”.
- **protein_name_dict** (*Dict[str,Union[int,str]], optional*) – the parameters that control the printing of the protein name, for example, the font size and the color. These parameters should be provided as dict that will be fed to the function `axes.text()`. The default is {“fontsize”:10,”color”:”black”}.
- **rect_dict** (*Dict[str,Union[int,str]], optional*) – a dictionary that control the character of the track itself, for example, the color and the transparency. this dict will be fed to the function `plt.Rectangle()`. The default is {“color”:”olive”,”capstyle”:”butt”}.
- **number_ticks** (*int*) – The number of ticks on the x-axis. The default is 10.
- **xticks_font_size** (*int*) – The font size of the x-axis ticks. The default is 4.

Returns

Return type None.

Examples

```
>>> example_1=VisTool(250,(3,5),300)
    # create a graph of size 3 inches by 5 inches with a 300 dots per
    # inch (DPI) as a resolution metric for a protein of length 250 amino acids
```

```
>>> example_1.add_base_track()
    # adds a basic track using the default parameters.
```

```
>>> example_1.add_base_track(space_fraction=0.1,
                             track_label="example_1",
                             track_label_dict={"fontsize":5,"color":"blue"}
                             number_ticks=5,
                             xticks_font_size=6)
    # generate a base track with 10% empty space above and below
    #  the track. Track will have the name example_1 and it will be
    # shown in font 5 instead of 8 and in blue color instead of black.
    # five ticks will be shown on the x-axis using a font of size 6.
```

Notes

calling the function more than once will result in an overriding of the previously added base track, for example, in the examples section calling add_base_track for the second time will overrides the graph build by the previous call.

add_coverage_track(*coverage_matrix: numpy.ndarray*, *coverage_as_base: bool = False*, *coverage_dict: Dict[str, Union[int, str]] = {'color': 'blue', 'width': 1.2}*, *xlabel: str = 'positions'*, *xlabel_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 6}*, *ylabel: str = 'coverage'*, *ylabel_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 6}*, *number_ticks: int = 10*, *xticks_font_size: int = 4*, *yticks_font_size: int = 4*)

Adds a coverage plot to the panel. The coverage plot shows the relationship between a peptide and its experimentally detected eluted peptide/peptides.

Parameters

- **coverage_matrix** (*np.ndarray*) – A protein length by one array which summarize information about the protein and the eluted peptides.
- **coverage_as_base** (*bool, optional*) – Whether or not to plot the coverage as a base track for the figure. The default is False which means that the track appended to a figure that have a default base track which can be constructed using the method `add_base_track`. However, if coverage_as_base is set to True, the function will draw the base track using the coverage matrix and calling the function add_base_track should be avoided.
- **coverage_dict** (*Dict[str,Union[int,str]], optional*) – The parameters that control the printing of the coverage matrix, for example, the color. These parameters are fed to the function `axes.bar`. The default is {“color”:”blue”,”width”:1.2}.
- **xlabel** (*str, optional*) – The label of the x-axis of the coverage track. The default is “positions”.
- **xlabel_dict** (*Dict[str,Union[int,str]], optional*) – The parameters that control the x-label printing, for example, the color and/ the font size. these parameters are fed to the function `axes.set_xlabel`. The default is {“fontsize”:6,”color”:”black”}.

- **ylabel** (*str, optional*) – The label of the y-axis of the coverage track. The default is "coverage".
- **ylabel_dict** (*Dict[str,Union[int,str]], optional*) – The parameters that control the x-label printing, for example, the color and/ the font size. these parameters are fed to the function `axes.set_ylabel`. The default is {"fontsize":6,"color":"black"}.
- **number_ticks** (*int, optional*) – The number of ticks on the x-axis. The default is 10.
- **xticks_font_size** (*float, optional*) – The font size of the x-axis ticks. The default is 4.
- **yticks_font_size** (*float, optional*) – The font size of the y-axis ticks. The default is 4.

add_marked_positions_track(*positions: List[int]*, *height_frac: float = 0.5*, *marker_bar_dict: Dict[str, Union[int, str]] = {'color': 'black', 'linestyles': 'solid'}*, *marker_dict: Dict[str, Union[int, str]] = {'color': 'red', 's': 3}*, *track_label: str = 'A marked positions Track'*, *track_label_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 8}*, *base_line_dict: Dict[str, Union[int, str]] = {'color': 'black', 'linewidth': 1}*)

The function adds a marked position to the track which is shown to highlight certain amino acid position within the protein, for example, a sequence variant position, or PTM position.

positions [List[int]] a list that contain the position/positions that should be heighlighted in the protein sequence.

height_frac [float] the relative hight of the marked positions. The default is 0.5 which means that the hight of the marker will be 50% of the y-axis height.

marker_bar_dict [Dict[str,Union[int,str]], optional] The parameters of the marker position bar, for example, line width or color. These parameters are going to be fed to the function `plt.hlines`. The default is {"color":"black","linestyles":"solid"}.

marker_dict [Dict[str,Union[int,str]], optional] These are the parameters for the marker points which sits on top of the marker bar, for example, the color, the shape or the size. The default is {"color":"red","s":3}.

track_label [str, optional] The name of the track, which will be shown on the y-axis. The default is "A marked positions Track".

track_label_dict [Dict[str,Union[int,str]], optional] The parameters that control the printing of the track_label, for example, the font size and the color. These parameters should be provided as dict that will be fed to the function `axes.set_ylabel`.The default is {"fontsize":8,"color":"black"}.

base_line_dict [Dict[str,Union[int,str]], optional]

The parameters that control the shape of the base line, for example, color and/or line width. These parameters are going to be fed to the function `axes.hlines`. The default is {"color":"black","linewidth":1}.

None.

```
>>> test_list=[24,26,75,124,220]
# first define a dict object that define some protein features.
```

```
>>> example_1=Annotator(protein_length=250, figure_size=(5,3), figure_
↪dpi=200)
# creating a VisTool instance
```

```
>>> example_1.add_base_track()
# add a base_track
```

```
>>> example_1.add_marked_positions_track(test_list) # build a marked␣
↪position track using the default parameters
# marked positions track
```

```
>>> example_1.add_marked_positions_track(positions=test_list,height_frac=0.
↪75,
                                    track_label="Post_translational_
↪modifications",
                                    marker_bar_dict={"color":"blue"})
# add a second marked position track with the following parameters:
#track name:  Post_translational_modifications
#hight of the maker bar = 75%
#color of the markerbar= blue
```

Any panel can have zero, one or more than one marked-position track. Thus, in the above examples calling the method `add_marked_positions_track` for the second time does NOT override the previous marked-position track it create a new one and added to the figure.

add_segmented_track(*track_dict: Dict[str, Dict[str, Union[int, str]]]*, *track_label: str = 'A segmented Track'*, *track_label_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 8}*, *track_element_names_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 8}*, *center_line_dict: Dict[str, Union[int, str, float]] = {'alpha': 0.5, 'linewidth': 0.5}*, *track_elements_dict: Dict[str, Union[int, str]] = {'capstyle': 'butt', 'color': 'brown'}*, *show_names: bool = True*) → None

Adds a segmentation track which show non-overlapping features of the protein.

Parameters

- **track_dict** (*Dict[str,Dict[str,Union[int,str]]]*) – A dict that contain the non-overlapping features of the protein. The dict is assumed to have the following structure: a dict with the feature_index as a key and associated features as values. The associated features is a dict with the following three keys:

  1- Name: which contain the feature name

  2- startIdx: which contain the start position of the protein

  3- endIdx: which contain the end position of the protein

- **track_label** (*str, optional*) – The name of the track, which will be shown on the y-axis. The default is "A segmented Track".
- **track_label_dict** (*Dict[str,Union[int,str]], optional*) – the parameters that control the printing of the track_label, for example, the font size and the color. These parameters should be provided as dict that will be fed to the function `axes.set_ylabel`. The default is {"fontsize":8,"color":"black"}.
- **track_element_names_dict** (*Dict[str,Union[int,str]], optional*) – the parameters that control the printing of the feature names on the track, for example, the font

size and the color. These parameters should be provided as a dict that will be fed to the function `axes.text`. The default is {"fontsize":8,"color":"black"}.

- **center_line_dict** (*Dict[str,Union[int,str, float]], optional*) – The parameters that control the printing of the center line of a segmented track object. The default is {"fontsize":8,"color":"black"}.
- **track_elements_dict** (*Dict[str,Union[int,str]], optional*) – the parameters that control the printing of the feature rectangluar representation for example the color, the dict will be fed to the function `plt.Rectangle`. The default is {"color":"brown","capstyle":"butt"}.
- **show_names** (*bool, optional*) – whether or not to show the name of the features. The default is True.

Returns

Return type None.

Examples

```
>>> test_dict={"domain1":{"Name":"domain_one","startIdx":55,"endIdx":150},
               "domain2":{"Name":"domain_Two","startIdx":190,"endIdx":225}}
# first define a dict object that define some protein features.
```

```
>>> example_1=Annotator(protein_length=250, figure_size=(5,3), figure_dpi=200)
# creating a Annotator instance
```

```
>>> example_1.add_base_track()
# add a base_track
```

```
>>> example_1.add_segmented_track(test_dict) # build a segmented track using␣
↪the default parameters
# add the segmented track
```

```
>>> example_1.add_segmented_track(track_dict=test_dict,
                                  track_label="Domains",
                                  track_elements_dict={"color":"brown"})
# add a second segmented track with track name set to Domains and elements
# of the track shown as brown rectangles.
```

Notes

Any panel can have one or more segmented-tracks. Thus, in the above examples calling the method `add_segmented_track` for the second time does NOT override the previous segmented track it create a new one and added to the figure.

add_stacked_track(*track_dict: Dict[str, Dict[str, Union[int, str]]], track_label: str = 'A stacked Track', track_label_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 8}, track_element_names_dict: Dict[str, Union[int, str]] = {'color': 'black', 'fontsize': 8}, track_elements_dict: Dict[str, Union[int, str]] = {'capstyle': 'butt', 'color': 'magenta'}, base_line_dict: Dict[str, Union[int, str]] = {'color': 'black', 'linewidth': 1}, show_names: bool = True*)

The function adds a stacked_track to a visualization panel. The stacked track is used to show overlapping

protein features, for example, different splice variants.

Parameters **track_dict** (*Dict[str,Dict[str,Union[int,str]]]*) – A dict that contain the overlapping features of the protein. The dict is assumed to have the following structure, a dict with the feature_index as a key and associated features as values. The associated features is a dict with the following three keys:

1- Name: which contain the feature's name

2- startIdx: which contain the start position of the feature.

3- endIdx: which contain the end position of the feature.

track_label [str, optional] The name of the track, which will be shown on the y-axis. The default is "A stacked Track".

track_label_dict [Dict[str,Union[int,str]], optional] the parameters that control the printing of the track_label, for example, the font size and the color. These parameters should be provided as dict that will be fed to the function `axes.set_ylabel`.The default is {"fontsize":8,"color":"black"}.

track_element_names_dict [Dict[str,Union[int,str]], optional] the parameters that control the printing of the feature names on the track, for example, the font size and the color. These parameters should be provided as a dict that will be fed to the function `axes.text`. The default is {"fontsize":8,"color":"black"}.

track_elements_dict [Dict[str,Union[int,str]], optional] the parameters that control the printing of the feature rectangluar representation for example the color, the dict will be fed to the function `plt.Rectangle`. The default is {"color":"magenta","capstyle":"butt"}.

base_line_dict [Dict[str,Union[int,str]], optional] the parameters that control the shape of the base line, for example, color and/or line width. These parameters are going to be fed to the function `axes.hlines`. The default is {"color":"black","linewidth":1}.

show_names [bool, optional] whether or not to show the name of the features. The default is True.

Returns

Return type None.

Examples

```
>>> test_dict={"feature_1":{"Name":"X","startIdx":55,"endIdx":150},
               "feature_2":{"Name":"Y","startIdx":85,"endIdx":225},
               "feature_3":{"Name":"Z","startIdx":160,"endIdx":240}}
 # first define a dict object that define some protein features.
```

```
>>> example_1=Annotator(protein_length=250, figure_size=(5,3), figure_dpi=200)
# creating a Annotator instance
```

```
>>> example_1.add_base_track()
# add a base_track
```

```
>>> example_1.add_segmented_track(test_dict) # build a stacked track using the␣
↪default parameters.
# add the stacked track
```

```
>>> example_1.add_segmented_track(track_dict=test_dict,
                                  track_label="OverLappingFeat",
                                  track_elements_dict={"color":"red"})
# add a second segmented track with track name set to OverLappingFeat and␣
↪elements
# of the track shown as red rectangles.
```

Notes

Any panel can have zero, one or more than one stacked-track. Thus, in the above examples calling the method `add_stacked_track` for the second time does NOT override the previous stacked track it creates a new one and added to the figure.

get_figure() → matplotlib.figure.Figure

Returns The figure with all the tracks that have been added to it.

Return type matplotlib.figure.Figure

save_fig(*name: str*, *output_path: str = '.'*, *format_: str = 'png'*, *figure_dpi: str = 'same'*, *figure_saving_dict: Dict[str, Union[int, str]] = {'facecolor': 'white'}*) → None

Write the constructed figure to the disk.

Parameters

- **name** (*str*) – The name of the figure to save the file.
- **output_path** (*str , optional*) – The path to write the output, by default the function write to the current working directory.
- **format** (*str, optional*) – The output format, this parameter will be fed to the method `plt.savefig`. The default is "png".
- **figure_dpi** (*int, optional*) – The dpi of the saved figure. The deafult is same which means the figure will be saved using the same dpi used for creating the figure.
- **figure_saving_dict** (*Dict[str,Union[int,str]],optional*) – The parameters that should be fed to the function `plt.savefig`. The default is figure_saving_dict={"facecolor":"white"}

Returns

Return type None.

IPTK.Classes.Database module

This submodule defines a collection of container classes that are used through the library

class IPTK.Classes.Database.**CellularLocationDB**(*path2data: str = 'https://www.proteinatlas.org/download/subcellular_location.tsv.zip'*, *sep: str = '\t'*)

Bases: object

The class provides an API to access the cellular location information from a database that follows the structure of the Human Proteome Atlas sub-cellular location database. See https://www.proteinatlas.org/about/download for more details.

add_to_database(*genes_to_add:* IPTK.Classes.Database.CellularLocationDB) → None
adds the the location of more proteins to the database.

Parameters **genes_to_add** (CellularLocationDB) – a CellularLocationDB instance containing the genes that shall be added to the database.

Raises

- **ValueError** – if the genes_to_add to the database are already defined in the database
- **RuntimeError** – incase any other error has been encountered while merging the tables.

get_approved_location(*gene_id: Optional[str] = None*, *gene_name=None*) → List[str]
return the location of the provided gene id or gene name

Parameters

- **gene_id** (*str, optional*) – the id of the gene of interest, defaults to None
- **gene_name** (*[type], optional*) – the name of gene of interest, defaults to None

Raises

- **ValueError** – if both gene_id and gene_name are None
- **KeyError** – if gene_id is None and gene_name is not in the database
- **KeyError** – if gene_name is None and gene_id is not in the database
- **RuntimeError** – Incase an error was encountered while retriving the element from the database.

Returns The approved location where the protein that corresponds to the provided name or id is located.

Return type List[str]

get_gene_names() → List[str]
return a list of all gene names in the dataset

Returns the names of all genes in the database

Return type List[str]

get_genes() → List[str]
return a list of all gene ids in the dataset

Returns all genes ids currently defined in the database

Return type List[str]

get_go_names(*gene_id: Optional[str] = None*, *gene_name=None*) → List[str]
return the location of the provided gene id or gene name

Parameters

- **gene_id** (*str, optional*) – the id of the gene of interest , defaults to None
- **gene_name** (*[type], optional*) – the name of the gene of interest , defaults to None

Raises

- **ValueError** – if both gene_id and gene_name are None
- **KeyError** – if gene_id is None and gene_name is not in the database
- **KeyError** – if gene_name is None and gene_id is not in the database

- **RuntimeError** – incase an error was encountered while retriving the element from the database.

Returns The gene ontology, GO, location where the protein that corresponds to the provided name or id is located.

Return type List[str]

get_main_location(*gene_id: Optional[str] = None*, *corresponds=None*) → List[str]

Return the main location(s) of the provided gene id or gene name. If both gene Id and gene name are provided, gene_id has a higher precedence

Parameters

- **gene_id** (*str, optional*) – The id of the gene of interest, defaults to None
- **gene_name** (*[type], optional*) – The name of the gene of interest, defaults to None

Raises

- **ValueError** – if both gene_id and gene_name are None
- **KeyError** – if gene_id is None and gene_name is not in the database
- **KeyError** – if gene_name is None and gene_id is not in the database
- **RuntimeError** – Incase an error was encountered while retriving the element from the database

Returns the main location where the protein that corresponds to the provided name or id is located.

Return type List[str]

get_table() → pandas.core.frame.DataFrame

return the instance table

Returns the location table of the instance.

Return type pd.DataFrame

class IPTK.Classes.Database.**GeneExpressionDB**(*path2data: str = 'https://www.proteinatlas.org/download/rna_tissue_consensus.tsv.zip'*, *sep: str = '\t'*)

Bases: object

The class provides an API to access gene expression data stored in table that follows the same structure as the Human proteome Atlas Normalized RNA Expression see https://www.proteinatlas.org/about/download for more details

get_expression(*gene_name: Optional[str] = None*, *gene_id: Optional[str] = None*) → pandas.core.frame.DataFrame

Return a table summarizing the expression of the provided gene name or gene id accross different tissues.

Parameters

- **gene_id** (*str, optional*) – the id of the gene of interest, defaults to None
- **gene_name** (*[type], optional*) – the name of the gene of interest, defaults to None

Raises

- **ValueError** – if both gene_id and gene_name are None
- **KeyError** – if gene_id is None and gene_name is not in the database
- **KeyError** – if gene_name is None and gene_id is not in the database

- **RuntimeError** – incase an error was encountered while retriving the elements from the database

Returns A table summarizing the expression of the provided gene accross all tissues in the database

Return type pd.DataFrame

get_expression_in_tissue(*tissue_name: str*) → pandas.core.frame.DataFrame
return the expression profile of the provided tissue

Parameters **tissue_name** (*str*) – The name of the tissue

Raises

- **KeyError** – Incase the provided tissue is not defined in the database
- **RuntimeError** – In case an error was encountered while generating the expression profile.

Returns A table summarizing the expression of all genes in the provided tissue.

Return type pd.DataFrame

get_gene_names() → List[str]
returns a list of the UNIQUE gene names currently in the database

Returns A list of the UNIQUE gene names currently in the database

Return type List[str]

get_genes() → List[str]
returns a list of the UNIQUE gene ids currently in the database.

Returns The list of the UNIQUE gene ids currently in the database

Return type List[str]

get_table() → pandas.core.frame.DataFrame
return a table containing the expression value of all the genes accross all tissues in the current instance

Returns The expression of all genes accross all tissues in the database.

Return type pd.DataFrame

get_tissues() → List[str]
return a list of the tissues in the current database

Returns A list containing the names of the UNIQUE tissues in the database.

Return type List[str]

class IPTK.Classes.Database.**OrganismDB**(*path2Fasta: str*)
Bases: object

Extract information about the source organsim of a collection of protein sequencesfrom a fasta file and provides an API to query the results. The function expect the input fasta file to have headers written in the UNIPROT format.

get_number_protein_per_organism() → pandas.core.frame.DataFrame
Provides a table containing the number of proteins per organism.

Returns A table containing the number of proteins per organism

Return type pd.DataFrame

get_org(*prot_id: str*) → str
return the parent organism of the provided protein identifer

Parameters **prot_id** (*str*) – the id of the protein of interest

Raises **KeyError** – incase the provided identifier is not in the database

Returns the name of the parent organism, i.e. the source organism.

Return type str

get_unique_orgs() → List[str]
Get the number of unique organisms in the database

Returns a list of all unique organisms in the current instance

Return type List[str]

class IPTK.Classes.Database.**SeqDB**(*path2fasta: str*)
Bases: object

Load a FASTA file and constructs a lock up dictionary where sequence ids are keys and sequences are values.

get_seq(*protein_id: str*) → str
returns the corresponding sequence if the provided protein-id is defined in the database.

Parameters **protein_id** (*str*) – The protein id to retrive its sequence, CASE SENSITIVE!!.

Raises **KeyError** – If the provided protein does not exist in the database

Returns the protein sequence

Return type str

has_sequence(*sequence_id: str*) → bool
check if the provided sequence id is an element of the database or not

Parameters **sequence_name** (*str*) – The id of the sequence, CASE SENSITIVE!!.

Returns True if the database has this id, False otherwise.

Return type bool

IPTK.Classes.Experiment module

IPTK.Classes.ExperimentalSet module

IPTK.Classes.Features module

Parses the XML scheme of a uniprot protein and provides a python API for quering and accessing the results

class IPTK.Classes.Features.**Features**(*uniprot_id: str*, *temp_dir: Optional[str] = None*)
Bases: object

The class provides a template for the features associated with a protein. The following features are associated with the protein #signal peptide: dict

The range of the signal peptides, if the protein has no signal, for example, a globular cytologic protein. None is used as a default, placeholder value.

#chains:dict the chains making up the mature protein, the protein should at least have one chain.

#domain: dict the known domains in the protein, if no domain is defined, None is used.

#modification sites: nested dict that contains information about the PTM sites, glycosylation site and disulfide bonds.

#sequence variances: dict which contains information about the sequence variants of a protein structure.

#split variance: dict which contain known splice variants

** Notes: Although disulfide bond is not a PTMs, it is being treated as a one here to simplify the workflow.

`get_PTMs()` → Dict[str, Dict[str, Dict[str, Union[int, str]]]]

Returns

a nested dictionary that contains the PTMs found within the protein the PTMs are classified into three main categories:

1- Modifications: which is the generic case and contain information about any sequence modification beside disulfide bonds and glycosylation.

2- glycosylation: contains information about glycosylation sites

3- DisulfideBond: contains information about disulfide bond

Return type Dict[str,Dict[str,Dict[str,Union[str,int]]]]

`get_PTMs_glycosylation()` → Dict[str, Dict[str, Union[int, str]]]

Returns The glycosylation sites found on the protein. If the protein has no glycosylation sites, the function returns None.

Return type [type]

`get_PTMs_modifications()` → Dict[str, Dict[str, Union[int, str]]]

Returns The generic modifications found on the protein. If the protein has no PTM, the function returns None.

Return type Dict[str,Dict[str,Union[str,int]]]

`get_chains()` → Dict[Dict[str, Union[str, int]]]

Returns A dictionary that contains the chains of the protein, if no chain is defined it return None

Return type Dict[Dict[str,Union[str,int]]]

`get_disulfide_bonds()` → Dict[str, Dict[str, Union[int, str]]]

Returns The disulfide sites found on the protein. If the protein has no disulfide sites, the function returns None

Return type [type]

`get_domains()` → Dict[str, Dict[str, int]]

Returns The domains defined in the protein sequence, if no domain is defined it returns None.

Return type Dict[str, Dict[str, int]]

`get_num_transmembrane_regions()` → int

Return the number of transmembrane regions on the protein

Returns Return the number of transmembrane regions on the protein

Return type int

get_number_PTMs() → int

Returns The number of PTMs the sequence has, this include di-sulfide bonds. See Note1 for more details. If the protein has no PTMs the function returns zero

Return type int

get_number_chains() → int

Returns The number of chains in the protein. if no chain is defined it returns zero.

Return type int

get_number_disulfide_bonds() → int

Returns The number of disulfide bonds the protein has, if the protein has no disulfide bonds, the function return zero.

Return type int

get_number_domains() → int

Returns The number of domains a protein has, if no domain is defined it returns zero.

Return type int

get_number_glycosylation_sites() → int

Returns The number of glycosylation_sites the protein has, if the protein has no glycosylation sites, the function returns zero

Return type int

get_number_modifications() → int

Returns Returns the total number of generic modifications found on the protein. if no modification is found it return 0

Return type int

get_number_sequence_variants() → int

Returns The number of sequence variants the protein has, if the protein has no sequence varient, the function returns 0.

Return type int

get_number_splice_variants() → int

Returns The number of slice variants in the protein, if the protein has no splice variants, the function returns zero.

Return type int

get_sequence_variants() → Dict[str, Dict[str, Union[int, str]]]

Returns A dict object that contains all sequence variants within a protein, if the protein has no sequence variants the function returns None.

Return type Dict[str,Dict[str,Union[str,int]]]

get_signal_peptide_index() → Tuple[int, int]

Returns The Index of the signal-peptide in the protein, if not signal peptide is defined, it returns None

Return type Tuple[int,int]

get_splice_variants() → Dict[str, Dict[str, Union[int, str]]]

Returns A dict object that contains the splice variants. If the protein has no splice variants the function returns None.

Return type Dict[str,Dict[str,Union[str,int]]]

get_transmembrane_regions() → List[Tuple[int, int]]
return a list containing the boundaries of transmembrane regions in the protein

Returns a list containing the boundaries of transmembrane regions in the protein

Return type List[Tuple[int,int]]

has_PTMs() → bool
:return:True if the protein has a PTMs and False other wise :rtype: bool

has_chains() → bool

Returns True if the protein has/have chain/chains as feature and False otherwise.

Return type [type]

has_disulfide_bond() → bool

Returns True is the protein has disulfide and False other wise

Return type bool

has_domains() → bool

Returns True if the protein has a defined domain/domains, otherwise it return False.

Return type bool

has_glycosylation_site() → bool

Returns True if the protein has a glycosylation site and False otherwise.

Return type [type]

has_sequence_variants() → bool

Returns True if the protein has a sequence variants, and False otherwise.

Return type bool

has_signal_peptide() → bool

Returns True if the protein has a signal peptide and False other wise.

Return type bool

has_site_modifications() → bool

Returns True if the protein has a modification site and False otherwise

Return type bool

has_splice_variants() → bool

Returns True if the sequence has a splice variants and False otherwise.

Return type bool

has_transmembrane_domains() → bool

Returns True if the protein has transmembrane region and false otherwise

Return type bool

summary() → Dict[str, Union[int, str]]

Returns The function return a dict object that summarizes the features of the protein.

Return type Dict[str,Union[str,int]]

IPTK.Classes.HLAChain module

The implementation of an HLA molecules

class IPTK.Classes.HLAChain.**HLAChain**(*name: str*)

Bases: object

get_allele_group() → str

Returns The allele group

Return type str

get_chain_class(*gene_name: str*) → int

Parameters **gene_name** (*str*) – the name of the gene

Returns 1 if the gene belongs to class one and 2 if it belong to class two

Return type int

get_class() → int

Returns The HLA class

Return type int

get_gene() → str

Returns The gene name

Return type str

get_name() → str

Returns The chain name

Return type str

get_protein_group() → str

Returns The protein name

Return type str

IPTK.Classes.HLAMolecules module

a representation of an HLA molecules

class IPTK.Classes.HLAMolecules.**HLAMolecule**(***hla_chains*)

Bases: object

get_allele_group() → List[str]

Returns The allele group for the instance chain/pair of chains

Return type AlleleGroup

get_class() → int

Returns The class of the HLA molecules

Return type int

get_gene() → List[str]

Returns return gene/pair of genes coding for the current HLA molecules

Return type Genes

get_name(*sep: str = ':'*) → str

Parameters **sep** (*str, optional*) – The name of the allele by concatenating the names of the individual chains using a separator, defaults to ':'

Returns [description]

Return type str

get_protein_group() → List[str]

Returns The protein group for the instance chain/pair of chains

Return type ProteinGroup

IPTK.Classes.HLASet module

An abstraction for a collection of HLA alleles

class IPTK.Classes.HLASet.**HLASet**(*hlas: List[str]*, *gene_sep: str = ':'*)
Bases: object

get_alleles() → List[str]

Returns The current alleles in the set

Return type int

get_class() → int

Returns The class of the HLA-alleles in the current instance

Return type int

get_hla_count() → int

Returns The count of HLA molecules in the set

Return type int

get_names() → List[str]
Return a list of all HLA allele names defined in the set

Returns [description]

Return type List[str]

has_allele(*allele: str*) → bool

Parameters **allele** (*str*) – The name of the alleles to check for its occurrence in the instance.

Returns True, if the provided allele is in the current instance, False otherwise.

Return type bool

has_allele_group(*allele_group: str*) → bool

Parameters **allele_group** (*str*) – The allele group to search the set for

Returns True, if at least one allele in the set belongs to the provided allele group, False otherwise.

Return type bool

has_gene(*gene_name: str*) → bool

Parameters **gene_name** (*str*) – the gene name to search the set against.

Returns True, if at least one of the alleles in the set belongs to the provided gene. False otherwise

Return type bool

has_protein_group(*protein_group: str*) → bool

Parameters **protein_group** – The protein group to search the set for

Returns True, if at least one allele in the set belongs to the provided protein group

Return type bool

IPTK.Classes.Peptide module

IPTK.Classes.Proband module

A description for an IP proband

class IPTK.Classes.Proband.**Proband**(***info*)

Bases: object

get_meta_data() → dict

Returns A dict containing all the meta-data about the proband

Return type dict

get_name() → str

Returns The name of the proband

Return type str

update_info(***info*) → None

Add new or update existing info about the patient using an arbitrary number of key-value pairs to be added to the instance meta-info dict

IPTK.Classes.Protein module

IPTK.Classes.Tissue module

A representation of the Tissue used in an IP Experiment.

class IPTK.Classes.Tissue.**ExpressionProfile**(*name: str*, *expression_table: pandas.core.frame.DataFrame*, *aux_proteins: Optional[pandas.core.frame.DataFrame] = None*)

Bases: object

a representation of tissue reference expression value.

get_gene_id_expression(*gene_id: str*) → float

Parameters **gene_id** (*str*) – the gene id to retrive its expression value from the database

Raises **KeyError** – if the provided id is not defined in the instance table

Returns the expression value of the provided gene id.

Return type float

get_gene_name_expression(*gene_name: str*) → float

Parameters **gene_name** (*str*) – the gene name to retrive its expression value from the database

Raises **KeyError** – if the provided id is not defined in the instance table

Returns the expression value of the provided gene name.

Return type float

get_name() → str

Returns the name of the tissue where the expression profile was obtained

Return type str

get_table() → pandas.core.frame.DataFrame

Returns return a table that contain the expression of all the transcripts in the current profile including core and auxiliary proteins

Return type pd.DataFrame

class IPTK.Classes.Tissue.**Tissue**(*name: str*, *main_exp_value:* IPTK.Classes.Database.GeneExpressionDB, *main_location:* IPTK.Classes.Database.CellularLocationDB, *aux_exp_value: Optional[*IPTK.Classes.Database.GeneExpressionDB*] = None*, *aux_location: Optional[*IPTK.Classes.Database.CellularLocationDB*] = None*)

Bases: object

get_expression_profile() → *IPTK.Classes.Tissue.ExpressionProfile*

Returns the expresion profile of the current tissue

Return type *ExpressionProfile*

get_name() → str

Returns the name of the tissue

Return type str

get_subCellular_locations() → *IPTK.Classes.Database.CellularLocationDB*

Returns the sub-cellular localization of all the proteins stored in current instance resources.

Return type *CellularLocationDB*

Module contents

IPTK.IO package

Submodules

IPTK.IO.InFunctions module

IPTK.IO.MEMEInterface module

The module contains functions to to call meme software via a system call.

IPTK.IO.MEMEInterface.**call_meme**(*input_fasta_file: str*, *output_dir: str*, *verbose: bool = True*, *objfunc: str = 'classic'*, *test: str = 'mhg'*, *use_llr: bool = False*, *shuf: int = 2*, *hsfrac: float = 0.5*, *cefrac: float = 0.25*, *searchsize: int = - 1*, *maxsize: int = - 1*, *norand: bool = False*, *csites: int = - 1*, *seed: int = - 1*, *mod: str = 'oops'*, *nmotifs: int = - 1*, *evt: float = - 1.0*, *time: int = - 1*, *nsite: int = - 1*, *minsites: int = - 1*, *maxsite: int = - 1*, *nsites: int = - 1*, *w: int = - 1*, *minw: int = - 1*, *maxw: int = - 1*, *nomatrim: bool = False*, *wg: int = - 1*, *ws: int = - 1*, *noendgaps: bool = False*, *maxiter: int = - 1*, *prior: str = 'dirichlet'*, *b: int = - 1*, *p: int = - 1*) → None

warper for making a system call to meme software for sequence motif finding for the reset of the function parameters use the function **get_meme_help** defined in the module IO, submodule MEMEInterface.

Parameters

- **input_fasta_file** (*str*) – The path to input FASTA files.
- **output_dir** (*str*) – the output dir to write the results, **IT WILL OVERWRITE EXISTING DIRECTORY**
- **verbose** (*bool*) – whether or not to print the output of calling meme to the screen, default is True.

IPTK.IO.MEMEInterface.**get_meme_help**() → None

Print the command line help interface for the meme tool

Raises **FileNotFoundError** – if meme is not callable

IPTK.IO.MEMEInterface.**is_meme_callable**() → bool

Returns True if meme is callable, False otherwise.

Return type bool

IPTK.IO.OutFunctions module

Module contents

IPTK.Utils package

Submodules

IPTK.Utils.DevFunctions module

IPTK.Utils.Mapping module

A submodule that contain function to map different database keys

IPTK.Utils.Mapping.**map_from_uniprot_gene**(*uniprots: List[str]*) → pandas.core.frame.DataFrame

map from uniprot id to ensemble gene ids

Parameters **uniprots** (*List[str]*) – a list of uniprot IDs

Returns A table that contain the mapping between each uniprot and its corresponding Gene ID/IDs

Return type pd.DataFrame

IPTK.Utils.Mapping.**map_from_uniprot_pdb**(*uniprots: List[str]*) → pandas.core.frame.DataFrame
map from uniprot id to protein data bank identifiers

Parameters **uniprots** (*List[str]*) – a list of uniprot IDs

Returns A table that contain the mapping between each uniprot and its corresponding PDB ID/IDs

Return type pd.DataFrame

IPTK.Utils.Mapping.**map_from_uniprot_to_Entrez_Gene**(*uniprots: List[str]*) → pandas.core.frame.DataFrame
map from uniprot id to ensemble gene ids

Parameters **uniprots** (*List[str]*) – a list of uniprot IDs

Returns A table that contain the mapping between each uniprot and its corresponding Gene ID/IDs

Return type pd.DataFrame

IPTK.Utils.Types module

Contain a definition of commonly used types through the library

IPTK.Utils.UtilityFunction module

Utility functions that are used through the library

IPTK.Utils.UtilityFunction.**append_to_calling_string**(*param: str*, *def_value*, *cur_val*, *calling_string: str*, *is_flag: bool = False*) → str
help function that take a calling string, a parameter, a default value and current value if the parameter does not equal its default value the function append the parameter with its current value to the calling string adding a space before the calling_string.

Parameters

- **param** (*str*) – The name of the parameter that will be append to the calling string
- **def_value** (*[type]*) – The default value for the parameter
- **cur_val** (*[type]*) – The current value for the parameter
- **calling_string** (*str*) – The calling string in which the parameter and the current value might be appended to it
- **is_flag** (*bool, optional*) – If the parameter is a control flag, i.e. a boolean switch, it append the parameter to the calling string without associating a value to it , defaults to False

Returns the updated version of the calling string

Return type str

IPTK.Utils.UtilityFunction.**build_sequence_table**(*sequence_dict: Dict[str, str]*) → pandas.core.frame.DataFrame
construct a sequences database from a sequences dict object

Parameters **sequence_dict** (*Dict[str,str]*) – a dict that contain the protein ids as keys and sequences as values.

Returns pandas dataframe that contain the protein ID and the associated protein sequence

Return type pd.DataFrame

IPTK.Utils.UtilityFunction.**check_peptide_made_of_std_20_aa**(*peptide: str*) → str
Check if the peptide is made of the standard 20 amino acids, if this is the case, it return the peptide sequence, otherwise it return an empty string

Parameters **peptide** (*str*) – a peptide sequence to check its composition

Returns True, if the peptide is made of the standard 20 amino acids, False otherwise.

Return type str

IPTK.Utils.UtilityFunction.**combine_summary**(*child_dfs: List[pandas.core.frame.DataFrame]*, *root_df: Optional[pandas.core.frame.DataFrame] = None*) → pandas.core.frame.DataFrame
combine multiple summaray dataframes into one dataframe

Parameters

- **child_dfs** (*List[pd.DataFrame]*) – a list of summary dataframes to conctinate into one
- **root_df** (*pd.DataFrame, optional*) – a dataframe to append the child dataframe to its tail, defaults to None

Returns a dataframe containing the root and the child dataframes

Return type pd.DataFrame

IPTK.Utils.UtilityFunction.**generate_color_scale**(*color_ranges: int*) → matplotlib.colors.LinearSegmentedColormap
generate a color gradient with number of steps equal to color_ranges -1

Parameters **color_ranges** (*int*) – the number of colors in the range

Returns A color gradient palette

Return type matplotlib.colors.LinearSegmentedColormap

IPTK.Utils.UtilityFunction.**generate_random_name**(*name_length: int*) → str

Parameters **name_length** (*int*) – Generate a random ASCII based string

Returns [description]

Return type str

IPTK.Utils.UtilityFunction.**generate_random_protein_mapping**(*protein_len: int*, *max_coverage: int*) → numpy.ndarray
Generate a NumPy array with shape of 1 by protein_len where the elements in the array is a random integer between zero & max_coverage.

Parameters

- **protein_len** (*int*) – The length of the protein
- **max_coverage** (*int*) – The maximum peptide coverage at each position

Returns a NumPy array containing a simulated protein coverage

Return type np.ndarray

IPTK.Utils.UtilityFunction.**get_experiment_summary**(*ident_table: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame
takes as an input an identification table and return a summary table containing the count of unique peptides, unique proteins, maximum peptide length, minmum peptide length, median and mean peptide length

Parameters **ident_table** (*pd.DataFrame*) – the identification table as returned by one of the parser functions defined in the IO modules

Returns The summary table

Return type pd.DataFrame

IPTK.Utils.UtilityFunction.**get_idx_peptide_in_sequence_table**(*sequence_table: pandas.core.frame.DataFrame*, *peptide: str*) → List[str]

check the sequences table if the provided peptide is locate in one of its sequences and returns a list of protein identifiers containing the identifier of the hit proteins.

Parameters

- **sequence_table** (*pd.DataFrame*) – pandas dataframe that contain the protein ID and the associated protein sequence
- **peptide** (*str*) – The peptide sequence to query the protein with

Returns A list of protein identifiers containing the identifier of the hit proteins

Return type List[str]

IPTK.Utils.UtilityFunction.**load_3d_figure**(*file_path: str*) → matplotlib.figure.Figure

Parameters **file_path** (*str*) – Load a pickled 3D figure from the provided path

Raises **IOError** – The path of the pickled figure.

Returns a matplotlib figure

Return type plt.Figure

IPTK.Utils.UtilityFunction.**pad_mapped_proteins**(*list_array: List[numpy.ndarray]*, *pre_pad: bool = True*, *padding_char: int = - 1*) → numpy.ndarray

Pad the provided list of array into a 2D tensor of shape number of arrays by maxlength.

Parameters

- **list_array** (*List[np.ndarray]*) – A list of NumPy arrays where each array is a mapped_protein array, the expected shape of these arrays is 1 by protein length.
- **pre_pad** (*bool, optional*) – pre or post padding of shorter array in the list_array. Defaults to True, which mean prepadding
- **padding_char** (*int, optional*) – The padding char, defaults to -1

Returns A 2D tensor of shape number of arrays by maxlength.

Return type np.ndarray

IPTK.Utils.UtilityFunction.**save_3d_figure**(*outpath: str*, *fig2save: matplotlib.figure.Figure*) → None

write a pickled version of the a 3D figure so it can be loaded later for more interactive analysis

Parameters

- **outpath** (*str*) – The output path of the writer function
- **fig2save** (*plt.Figure*) – The figure to save to the output file

Raises **IOError** – In case writing the file failed

IPTK.Utils.UtilityFunction.**simulate_protein_binary_represention**(*num_conditions: int*, *protein_length: int*)

Parameters

- **num_conditions** (*int*) – The number of conditions to simulate
- **protein_length** (*int*) – The Length of the protein

Returns A 2D matrix of shape protein_length by number of conditions, where each element can be either zero or 1.

Return type np.ndarray

IPTK.Utils.UtilityFunction.**simulate_protein_representation**(*num_conditions: int*, *protein_len: int*, *protein_coverage: int*) → Dict[str, numpy.ndarray]

Simulate protein peptide coverage under-different conditions

Parameters

- **num_conditions** (*[type]*) – The number of condition to simulate
- **protein_len** (*[type]*) – The length of the protein
- **protein_coverage** (*[type]*) – The maximum protein coverage

Returns a dict of length num_conditions containing the condition index and a simulated protein array

Return type Dict[str, np.ndarray]

Module contents

IPTK.Visualization package

Submodules

IPTK.Visualization.vizTools module

Module contents

Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

H

I

L

M

O

P

S

T

U