IPTK Reference

version

Hesham ElAbd

September 30, 2020

Contents

Welcome to IPTK's documentation!

Analysis, Visualize, Compare and Integrate experimentally generated or in-silico predicted Immunopeptidomics data, !

Introduction:

IPTK is a Pythonic library specialized in the analysis of HLA-peptidomes identified through an Immunopeptiomics pipeline. The library provides a high level API for analyzing and visualizing the identified peptides, Integrating transcritomics and protein structure information for a rich analysis and for comparing different experiments and different runs.

Installation:

The library can installed using pip as follow

Funding:

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

Guide

License

Contact

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

Get Started!

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

IPTK

IPTK package

Subpackages

IPTK.Analysis package

Submodules

IPTK.Analysis.AnalysisFunction module

The module contain a collection of analysis function that can be used by the methods of the classes defined in the classes module.

`IPTK.Analysis.AnalysisFunction.`**`compute_binary_distance`** (peptides: List[str], dist_func: Callable) → numpy.ndarray

compare the distance between every pair of peptides in a collection of peptides. @param: peptides: a collection of peptides sequences. @param: dist_func: function to compute the distance between each pair of peptides. @note:

Parameters:

- **peptides** (*List[str]*) – a collection of peptides sequences.
- **dist_func** (*Callable*) – a function to compute the distance between each pair of peptides.

Raises: **RuntimeError** – make sure that the dist_function is suitable with the peptides which might have different lengths.

Returns: the distance between each pair of peptides in the provided list of peptides

Return type: np.ndarray

`IPTK.Analysis.AnalysisFunction.`**`compute_change_in_protein_representation`** (mapped_prot_cond1: numpy.ndarray, mapped_prot_cond2: numpy.ndarray) → float

Compute the change in the protein representation between two conditions, by computing the difference in the area under the curve, AUC.

Parameters:

- **mapped_prot_cond1** (*np.ndarray*) – a mapped protein instance containing the protein coverage in the first condition
- **mapped_prot_cond2** (*np.ndarray*) – a mapped protein instance containing the protein coverage in the second condition

Raises: **ValueError** – if the provided pair of proteins is of different length

Returns: the difference in the area under the coverage curve between the two experiments.

Return type: float

`IPTK.Analysis.AnalysisFunction.`**`compute_difference_in_representation`** (mapped_prot_cond1: numpy.ndarray, mapped_prot_cond2: numpy.ndarray) → numpy.ndarray

return the difference in the representation of a protein between two conditions by substracting the coverage of the first protein from the second proteins.
@param: mapped_prot_cond1: a mapped protein instance containing the protein coverage in the first condition @param: mapped_prot_cond2: a mapped protein instance containing the protein coverage in the second condition

Parameters:
- **mapped_prot_cond1** (*np.ndarray*) – a mapped protein instance containing the protein coverage in the first condition
- **mapped_prot_cond2** (*np.ndarray*) – a mapped protein instance containing the protein coverage in the second condition

Returns: an array that shows the difference in coverage between the two proteins at each amino acid position.

Return type: np.ndarray

`IPTK.Analysis.AnalysisFunction.`**`compute_expression_correlation`** (exp1: IPTK.Classes.Experiment.Experiment, exp2: IPTK.Classes.Experiment.Experiment) → float

compute the correlation in the gene expression between two experiments by constructing a union of all the proteins expressed in the first and second experiments, extract the gene expression of these genes and then compute the correlation using SciPy stat module.

Parameters:
- **exp1** (*Experiment*) – The first experimental object
- **exp2** (*Experiment*) – he second experimental object

Returns: the correlation in gene expression of the proteins inferred in the provided pair of experiment

Return type: float

`IPTK.Analysis.AnalysisFunction.`**`download_structure_file`** (pdb_id: str) → None

Download PDB/mmCIF file containing the pbd_id from PDB using BioPython library

Parameters: **pdb_id** (*str*) – the protein id in protein databank

`IPTK.Analysis.AnalysisFunction.`**`get_PTMs_disuldfide_bonds`** (protein_feature) → List[int]

Parameters: **protein_feature** (*Features*) – a protein feature instance containing all protein features

Returns: A list that contains the position of the known disulfide bonds in the protein. If no disulfide bond(s) is/are known in the protein the function returns an empty list.

Return type: List[int]

`IPTK.Analysis.AnalysisFunction.`**`get_PTMs_glycosylation_positions`** (protein_feature: IPTK.Classes.Features.Features) → List[int]

Parameters: **protein_feature** (*Features*) – a protein feature instance containing all protein features

Returns: a list that contains the position of the generic glycosylation in the protein. If no glycosylation site(s) are/is known in the protein the function returns an empty list.

Return type: List[int]

`IPTK.Analysis.AnalysisFunction.`**`get_PTMs_modifications_positions`** (protein_feature: IPTK.Classes.Features.Features) → List[int]

Parameters: **protein_feature** (*Features*) – a protein feature instance containing all protein features

Returns: A list that contains the position of the generic modifications in the protein. If no modifications is known the function returns an empty list

Return type: List[int]

`IPTK.Analysis.AnalysisFunction.`**`get_binnary_peptide_overlap`** (exp1: IPTK.Classes.Experiment.Experiment, exp2: IPTK.Classes.Experiment.Experiment) → List[str]

compare the peptide overlap between two experimental objects.

Parameters:
- **exp1** (*Experiment*) – an instance of class Experiment.
- **exp2** (*Experiment*) – an instance of class Experiment.

Returns: a list of peptides that have been identified in both experiments.

Return type: Peptides

`IPTK.Analysis.AnalysisFunction.`**`get_binnary_protein_overlap`** (exp1: IPTK.Classes.Experiment.Experiment, exp2: IPTK.Classes.Experiment.Experiment) → List[str]

compare the protein overlap between two experimental objects.

Parameters:

- **exp1** (*Experiment*) – an instance of class Experiment.
- **exp2** (*Experiment*) – an instance of class Experiment.

Returns: a list of proteins that have been identified in both experiments.

Return type: Proteins

`IPTK.Analysis.AnalysisFunction.`**`get_chain_positions`** (protein_feature) → List[List[int]]

Parameters: **protein_feature** (*Features*) – a protein feature instance containing all protein features

Returns: a list of list that contains the position of the known chain/chains in the protein. each list has two elements which are the start and the end position of the position lists. If no sequence variant/variants that is/are known in the protein, the function returns an empty list. :rtype: List[List[int]]

`IPTK.Analysis.AnalysisFunction.`**`get_domains_positions`** (protein_feature) → List[List[int]]

Parameters: **protein_feature** (*Features*) – a protein feature instance containing all protein features

Returns: a list of list that contains the position of the known domain/domains in the protein. each list has two elements which are the start and the end position of the domain. If no sequence domain/domains that is/are known in the protein, the function returns an empty list. :rtype: List[List[int]]

`IPTK.Analysis.AnalysisFunction.`**`get_sequence_motif`** (peptides: List[str], temp_dir: str = './TEMP_DIR', verbose: bool = False, meme_params: Dict[str, str] = {}) → None

compute the sequences motif from a collection of peptide sequences using meme software.

Parameters:

- **peptides** (*Peptides*) – a list of string containing the peptide sequences
- **temp_dir** (*str, optional*) – he temp directory to write temp-files to it, defaults to "./TEMP_DIR"
- **verbose** (*bool, optional*) – whether or not to print the output of the motif discovery tool to the stdout, defaults to False
- **meme_params** (*Dict[str,str], optional*) – a dict object that contain meme controlling parameters, defaults to {}

Raises:

- **FileNotFoundError** – incase meme is not installed or could not be found in the path!
- **ValueError** – incase the peptides have different length!

`IPTK.Analysis.AnalysisFunction.`**`get_sequence_variants_positions`** (protein_feature) → List[int]

Parameters: **protein_feature** (*Features*) – a protein feature instance containing all protein features

Returns: A list that contains the position of the known sequence variants in the protein. If no sequence variant/variants that is/are known in the protein, the function returns an empty list.

Return type: List[int]

`IPTK.Analysis.AnalysisFunction.`**`get_splice_variants_positions`** (protein_feature) → List[List[int]]

Parameters: **protein_feature** (*Features*) – a protein feature instance containing all protein features

Returns: a list of list that contains the position of the known splice variant/variants in the protein. each list has two elements which are the start and the end position of the splice variants. If no splice variant/variants that is/are known in the protein, the function returns an empty list. :rtype: List[List[int]]

Module contents

IPTK.Classes package

Submodules

IPTK.Classes.Annotator module

The class provided 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 information about positions in the protein, for example, sequence variance, or PTM.

The class also provided 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, also, to visualize this aspect of the protein along 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`**

higher 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**)**

add 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** (*dDict[str,Union[int,str]]ict, 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**)**

Add 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 the protein and the eluted peptides. The coverage matrix can be computed using the function `AntigenicProtein.compute_protein_coverage_by_peptide`.
- **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 function `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*) – he 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 parameters of marker point 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 : *string, 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
axes.hlines. The default is {“color”:”black”,”linewidth”:1}.
e.

```
> 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
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:
rack name:  Post_translational_modifications
ight of the maker bar = 75%
olor of the markerbar= blue
```

panel can have zero, one or more than one marked-position track. Thus, in the above examples calling
_marked_positions_track for the second time does NOT override the previous marked-position track it create a new one and
re.

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

Add 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
# 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 add 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 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 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 pa
# 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 create 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 construct fig to the hard-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 mean 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 define 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 the follow 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

add 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 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 some error was encountered while running retriving the elements from the database

Returns: the approved location where the protein the 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 some error was encountered while running retriving the elements from the database

Returns: the gene ontology, GO, location where the protein the 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, both 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 some error was encountered while running retriving the elements from the database

Returns: the main location where the protein the 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`**

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 some error was encountered while running 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 provided in the database
- **RuntimeError** – in case any 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]

return 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]

return a list of the UNIQUE gene ids currently in the database

Returns: a 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 sequences from a fasta file and provides an API to query the results. The function expect the input fasta file to have header 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 proteins

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.

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

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

Return type: bool

IPTK.Classes.Experiment module

This module provides an abstraction for an IP experiment.

class `IPTK.Classes.Experiment.`**`Experiment`** (proband: IPTK.Classes.Proband.Proband, hla_set: IPTK.Classes.HLASet.HLASet, tissue: IPTK.Classes.Tissue.Tissue, database: IPTK.Classes.Database.SeqDB, ident_table: pandas.core.frame.DataFrame)

Bases: **`object`**

A representation of an immunopeptidomic experiment.

`add_org_info` **(**prot2org: Dict[str, str]**)** → None

annotated the inferred proteins with their source organism

Parameters: **prot2org** (*ProteinSource*) – a dict that contain the protein id as keys and its source organism as values and add this info to each protein inferred in the current experiment.

Raises: **RuntimeWarning** – If the provided dictionary does cover all proteins in the experimental object.

`annotate_proteins` **(**organisms_db: IPTK.Classes.Database.OrganismDB**)** → None

Extract the parent organisms of each protein in the experiment from an organism database instance.

Parameters: **organisms_db** (*OrganismDB*) – an OrgansimDB instance that will be used to annotate the proteins identified in the experiment.

`drop_peptide_belong_to_org` **(**org: str**)** → None

Drop the all the peptides that belong to a user provided organism. Note that, this function will IRREVERSIBLY remove the peptide from the experimental object.

Parameters: **org** (*str*) – the organims name

`get_binarized_results` **()** → List[numpy.ndarray]

Return a list of NumPy arrays where each array represents a child peptide, parent protein mapped pair. Please note that, The function treat each peptide-protein pair individually, that is if two peptides originating from the same protein, it treat them independently and the same protein will be represented twice with the two different peptides. Incase an integrative mapping is needed, the function @get_integrated_binarized_results@ shall be used.

Returns: a list of NumPy arrays containing the mapping between each peptide protein pair.

Return type: MappedProtein

`get_c_terminal_flanked_seqs` **(**flank_length: int**)** → List[IPTK.Classes.Peptide.Peptide]

return the c-terminal flanking sequences

Parameters: **flank_length** (*int*) – the length of the peptide downstream of the C-terminal of the peptide

Returns: a list sequences contain the N-terminal flanking sequence for each peptide in the instance.

Return type: Peptides

`get_experiment_reference_tissue_expression` **()** → pandas.core.frame.DataFrame

return the reference gene expression for the current tissue

Returns: A table that contain the expression value for ALL the genes in the instance Tissue

Return type: pd.DataFrame

`get_expression_of_parent_proteins` **(**non_mapped_dval: float = - 1**)** → pandas.core.frame.DataFrame

return a table containing the expression of the proteins inferred in the current experiment from the current tissue. This method need internet connection as it need to access uniprot mapping API to map uniprot IDs to gene IDs.

Parameters: **non_mapped_dval** (*float, optional*) – A default value to be added incase the parent protein is not define in the expression database, defaults to -1

Returns: a table that contain the expression of the protein inferred in the database

Return type: pd.DataFrame

`get_flanked_peptides` **(**flank_length: int**)** → List[str]

returns a list of sequences containing the peptides identified in the experiment padded with the flanking regions from all the parents of each peptide.

Parameters: **flank_length** (*int*) – the length of the flanking region

Returns: a list of the peptides + the flanking region.

Return type: Sequences

`get_go_location_id_parent_proteins` **(**not_mapped_val: str = 'UNK'**)** → pandas.core.frame.DataFrame

retrun the gene ontology,GO, location terms for all the identified proteins.

@brief: @param: not_mapped_val: the default value to return incase the GO term of the protein can not be extracted. @note: This method need internet connection as it need to access uniprot mapping API to map uniprot IDs to gene IDs.

Parameters: **not_mapped_val** (*str, optional*) – The default value to return incase the GO term of the protein can not be extracted, defaults to 'UNK'

Returns: A table that contain the GO-location term for each protein in the current instance.

Return type: pd.DataFrame

`get_hla_allele` () → List[str]

Returns: the set of HLA alleles from which the instance peptides have been eluted

Return type: List[str]

`get_hla_class` () → int

Returns: the HLA class

Return type: int

`get_main_sub_cellular_location_of_parent_proteins` (not_mapped_val: str = 'UNK'**)** → pandas.core.frame.DataFrame

retrun the main cellular location for the identified proteins. This method need internet connection as it need to access uniprot mapping API to map uniprot IDs to gene IDs.

Parameters: **not_mapped_val** (*str, optional*) – The default value to return incase the location of a protein can not be extracted, defaults to 'UNK'

Returns: A table that contain the main cellular compartment for each protein in the current instance.

Return type: pd.DataFrame

`get_mapped_protein` (pro_id: str**)** → numpy.ndarray

return an NumPy array of shape 1 x protein length where each number in the array represents the total number of peptides identified in the experiment that have originated from the said position in the protein.

Parameters: **pro_id** (*str*) – the protein id

Raises: **KeyError** – if the provided protein id was inferred from the current experiment

Returns: a NumPy array that contain the coverage of the protein.

Return type: np.ndarray

`get_mapped_proteins` () → Dict[str, List[numpy.ndarray]]

return a dictionary of all the proteins identified in the current experiment with all inferred peptides mapped to them.

Returns: a dictionary that contain the mapped proteins for all the proteins in the current instance.

Return type: MappedProteins

`get_mono_parent_peptides` () → List[IPTK.Classes.Peptide.Peptide]

return a list of peptides that have only one parent protein

Returns: list of peptide instance

Return type: Peptides

`get_n_terminal_flanked_seqs` (flank_length: int**)** → List[IPTK.Classes.Peptide.Peptide]

return the n-terminal flanking sequences

Parameters: **flank_length** (*int*) – the length of the flanking region upstream of the N-terminal of the peptide

Returns: a list sequences contain the N-terminal flanking sequence for each peptide in the instance.

Return type: Peptides

`get_negative_example` **(**fold: int = 2**)** → List[str]
generate negative examples, i.e., non-bounding peptides from the proteins identified in the current experiment.

Parameters:	**fold** (*int, optional*) – the number of negative example to generate relative to the number of unique identified peptides, defaults to 2
Returns:	list of non-presented peptides from all inferred proteins.
Return type:	Sequences

`get_num_peptide_expression_table` **()** → pandas.core.frame.DataFrame

Get a table that contain the id of all parent proteins, number of peptide per-proteins and the expression value

of these parent transcripts. Please note, this method need internet connection as it need to access uniprot mapping API to map uniprot IDs to gene IDs.

Returns:	the number of peptides per protein table
Return type:	pd.DataFrame

`get_num_peptide_per_go_term` **()** → pandas.core.frame.DataFrame
retrun the number of peptides per each GO-Term :return: A table that has two columns, namely, GO-Terms and Counts. :rtype: pd.DataFrame

`get_num_peptide_per_location` **()** → pandas.core.frame.DataFrame
retrun the number of peptides obtained from proteins localized to different sub-cellular compartments

Returns:	A table that has two columns, namely, Compartment and Counts.
Return type:	pd.DataFrame

`get_number_of_children` **(**pro_id: str**)** → int
return the number of children, i.e. number of peptides belonging to a parent protein

Parameters:	**pro_id** (*str*) – the id of the parent protein
Returns:	the number of peptides
Return type:	int

`get_number_of_proteins_per_compartment` **()** → pandas.core.frame.DataFrame
get the number of proteins from each compartment

Returns:	A table that has two columns, namely, Compartment and Counts.
Return type:	pd.DataFrame

`get_number_of_proteins_per_go_term` **()** → pandas.core.frame.DataFrame
get the number of proteins from each GO-Term

Returns:	A table that has two columns, namely, GO-Terms and Counts.
Return type:	pd.DataFrame

`get_orgs` **()** → List[str]
return a list containing the UNIQUE organisms identified in the current experiment

Returns:	list of all UNIQUE organisms inferred from the inferred proteins.
Return type:	List[str]

`get_peptide` **(**pep_seq: str**)** → IPTK.Classes.Peptide.Peptide
return a peptide instance corresponding to the user provided peptide sequence.

Parameters:	**pep_seq** (*str*) – the peptide sequence
Raises:	**KeyError** – if the peptide sequence has not been inferred from the current database.
Returns:	the peptide instance with the coresponding sequence
Return type:	Peptide

`get_peptide_number_parent` **(**ascending: bool = False**)** → pandas.core.frame.DataFrame

return a pandas dataframe with the peptide sequence in the first columns and the number of parent proteins in the second column.

Parameters: **ascending** (*bool, optional*) – ascending sort the peptide by their number of parent proteins, defaults to False

Returns: the number of parents for each peptide

Return type: pd.DataFrame

`get_peptides` **()** → List[IPTK.Classes.Peptide.Peptide]

Returns: a set of all the peptide stored in the experimental object

Return type: Peptides

`get_peptides_length` **()** → List[int]

return a list containing the length of each unique peptide in the database.

Returns: peptides' lengths

Return type: List[int]

`get_peptides_per_organism` **()** → pandas.core.frame.DataFrame

return a pandas dataframe that contain the count of peptides belonging to each organism in the database

Returns: a table with two columns, namely, Organisms and Counts

Return type: pd.DataFrame

`get_peptides_per_protein` **(**ascending: bool = False**)** → pandas.core.frame.DataFrame

return a pandas dataframe that contain the number of peptides belonging to each protein inferred in the experiment

Parameters: **ascending** (*bool, optional*) – ascending sort the proteins by their number of parent number of child peptides, defaults to False

Returns: a table with the following columns, Proteins and Number_of_Peptides

Return type: pd.DataFrame

`get_poly_parental_peptides` **()** → List[IPTK.Classes.Peptide.Peptide]

return a list of peptides that have more than one parent proteins :return: [list of peptide instance :rtype: Peptides

`get_proband_name` **()** → str

Returns: the proband name

Return type: str

`get_proteins` **()** → List[IPTK.Classes.Protein.Protein]

Returns: a set of all the proteins in the experimental object

Return type: Proteins

`get_tissue` **()** → IPTK.Classes.Tissue.Tissue

Returns: the tissue of the current experiment.

Return type: Tissue

`get_tissue_name` **()** → str

Returns: the tissue name

Return type: str

`has_allele_group` **(**gene_group: str**)** → bool return whether or not the experiment contain peptides eluted from an HLA-alleles belonging to the provided allele group or not

Parameters: **gene_group** (*str*) – the gene group to query the hla_set against

Returns: True if the gene group has a member that is a member of the instance HLASet and False otherwise

Return type: bool

`has_gene` (locus: str) → bool

return whether or not the experiment contain peptides eluted from an HLA-alleles belonging to the provided locus or not

Parameters: **locus** (*str*) – the locus of the allele to query the hla_set against

Returns: True if the locus has a member that is a member of the instance HLASet and False otherwise

Return type: bool

`has_hla_allele` (individual: str) → bool

return whether or not the experiment contain an eluted peptides from the provided alleles

Parameters: **individual** (*str*) – is the name of the allele as a string

Returns: True if the allele is a member of the instance HLASet and False otherwise.

Return type: bool

`has_protein_group` (protein_group: str) → bool

return whether or not the experiment contain peptides eluted from an HLA-alleles belonging to the provided protein group or not

Parameters: **protein_group** (*str*) – The protein group to query the hla_set against

Returns: True if the locus has a member that is a member of the instance HLASet and False otherwise

Return type: bool

`is_a_parent_protein` (protein: str) → bool

Parameters: **protein** – check if the protein is a member of the instance proteins or not.

Returns: True if the protein has been identified in the current instance, False otherwise.

Return type: bool

`is_member` (peptide: str) → bool

Parameters: **peptide** (*str*) – check if the peptide is a member of the instance peptides or not.

Returns: True if the peptide has been identified in the current instance, False otherwise.

Return type: bool

IPTK.Classes.ExperimentalSet module

An Experimental set which is a collection of experiments. The class provides an API for integrating and comparing different experimental instances.

class `IPTK.Classes.ExperimentalSet.`**`ExperimentSet`** (**exp_id_pair)

Bases: **`object`**

an API for integrating and comparing different experimental instances

`add_experiment` (**exp_id_pair) → None

add an arbitrary number of experiments to the set

`compare_org_count_among_exps` (org: str, abs_count: bool = False) → pandas.core.frame.DataFrame

Parameters:
- **org** (*str*) – The name of the organism to query the database for it.
- **abs_count** (*bool, optional*) – The absolute count, defaults to False

Returns: the count of the peptides that belong to a specific organism in the database.

Return type: pd.DataFrame

`compare_peptide_counts` **()** → pandas.core.frame.DataFrame

Returns: A table that contain the total number of peptides and per-organism peptide counts among all experiments in the set

Return type: pd.DataFrame

`compute_average_distance_between_exps` **()** → pandas.core.frame.DataFrame

compute the average distance between experiments by taking the average over the z-axis of the 3D tensor computed by the function compute_change_in_protein_representation.

Returns: A 2D tensor with shape of (num-experiments, num-experiments)

Return type: pd.DataFrame

`compute_change_in_protein_representation` **()** → numpy.ndarray

Compute the change in protein representation among the proteins that are presented/ detect in all of the instance's experiments.

Returns: a 3D tensor, T, with shape of (num-experiments, num-experiments, num-proteins), where T[i,j,k] is a the difference between experiment i & j with respect to the k th protein :rtype: np.ndarray

`compute_correlation_in_experssion` **()** → pandas.core.frame.DataFrame

compute the correlation in parent protein gene-expression across all the experiments in the set. See the function **compute_binary_correlation** in the analysis module for information about the computational logic.

Returns: return a 2D matrix containing the coorelation in gene expression between each pair of experiments inside the current instance collection of experiments.

Return type: pd.DataFrame

`compute_peptide_length_table` **()** → pandas.core.frame.DataFrame

Returns: A table that contain the length of each peptide in the experiment

Return type: pd.DataFrame

`compute_peptide_overlap_matrix` **()** → numpy.ndarray

Returns: a 2D matrix containing the number of peptide overlapping between each pair of experiments inside the current instance collection of experiment.

Return type: np.ndarray

`compute_peptide_representation_count` **()** → Dict[str, int]

Returns: The number of times a peptide was observed accross experiments in the set

Return type: Counts

`compute_protein_coverage_over_the_set` **()** → Dict[str, numpy.ndarray]

Returns: the mapped representation for each protein in accross the entire set

Return type: Dict[str, np.ndarray]

`compute_protein_overlap_matrix` **()** → numpy.ndarray

Returns: return a 2D matrix containing the number of proteins overlapping between each pair of experiments inside the current instance collection of experiment.

Return type: np.ndarray

`compute_protein_representation_count` **()** → Dict[str, int]

Returns: The number of times a protein was observed accross the experiment in the set

Return type: Counts

`drop_peptides_belong_to_org` **(**org_name: str**)** → None

drop all the peptides that belong to the provided organisms from all experiments in the set.

Parameters: **org_name** (*str*) – the name of the organism to drop

`get_allele_count` **()** → Dict[str, int]

Returns: the number of experiments obtained from each allele in the instance.

Return type: Counts

`get_experiment` **(**exp_name: str**)** → IPTK.Classes.Experiment.Experiment

return the experiment pointed to by the provided experimental name

Parameters: **exp_name** (*str*) – the name of the experiment

Raises: **KeyError** – if the provided experimental name is not in the dataset.

Returns: the experiment corresponds to the provided name

Return type: Experiment

`get_experimental_names` **()** → List[str]

Returns: a list with all the identifiers of the experiments in the set

Return type: Names

`get_experiments` **()** → Dict[Experiment]

Returns: return a dict with all the experiments stored in the instance as value of ids as keys.

Return type: Dict[Experiment]

`get_num_experiments_in_the_set` **()** → int

Returns: The number of experiments currently in the set

Return type: int

`get_peptides_present_in_all` **()** → List[IPTK.Classes.Peptide.Peptide]

Returns: the peptides that are observed in every experiments in the set.

Return type: Peptides

`get_proband_count` **()** → Dict[str, int]

Returns: The number of experiments obtained from each proband in the ExperimentalSet.

Return type: Counts

`get_proteins_present_in_all` **()** → List[str]

Returns: the proteins that are inferred in all experiments of the set

Return type: Proteins

`get_tissue_counts` **()** → Dict[str, int]

Returns: The number of experiments obtained from each tissue in the current instance

Return type: Counts

`get_total_peptide_per_org_count` **()** → pandas.core.frame.DataFrame

Returns: The total count of peptides per organism accross the all experiments in the set.

Return type: pd.DataFrame

`get_unique_orgs` **()** → List[str]

Returns: a list of the unique organisms in the set

Return type: List[str]

`get_unique_peptides` **()** → List[IPTK.Classes.Peptide.Peptide]

Returns: The set of unique peptides in the experimentalSet

Return type: Peptides

`get_unique_proteins` **()** → List[str]

Returns: the set of unique proteins in the experimentalset

Return type: Proteins

`group_by_proband` **()** → Dict[str, IPTK.Classes.ExperimentalSet.ExperimentSet]

Returns: a map between each proband and an Experimentalset object represent all the experiments objects belonging to this proband.

Return type: Dict[str,ExperimentSet]

`group_by_tissue` **()** → Dict[str, IPTK.Classes.ExperimentalSet.ExperimentSet]

Returns: A map between each tissue and an ExperimentalSet object representing all experiments belonging to that tissue.

Return type: Dict[str,ExperimentSet]

`is_peptide_present_in_all` **(**peptide: str**)** → bool

Parameters: **peptide** (*str*) – The peptide sequence to search its occurrences in every experiment contained in the set

Returns: True if peptide is present in all experiments inside the instance, False otherwise

Return type: bool

`is_protein_present_in_all` **(**protein: str**)** → bool

Parameters: **protein** (*str*) – the protein id to search its occurrences in every experimental in the set

Returns: True if peptide is present in all experiments inside the instance, False otherwise

Return type: bool

IPTK.Classes.Features module

Parse the XML scheme of a uniprot protein and represent and provide 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 provide 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 peptide,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 site, glycosylation site and disulfide bonds.

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

#split variance: dict
which contain known split variance

** 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 dictionary the contains the PTMs found within the protein the PTMs are classified into the main categories: 1- Modifications: which is the generic case and contain information about any sequence modification beside disulfide bonds and glycosylation. 2- glycosylation: contain information about glycosylation sites 3- DisulfideBond: contain 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 modification 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 the 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_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 chain/chains in the protein features. 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 modification 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 contain the splice variants. if the protein has no splice variants the function returns None.

Return type: Dict[str,Dict[str,Union[str,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

`summary` **()** → Dict[str, Union[int, str]]
:return:The function return a dict object that summarizes the features of the protein. :rtype: 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: 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 class of the HLA-alleles in the current instance

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

`has_allele` **(**allele: str**)** → bool

Parameters: **allele** (*str*) – the name of the allele to check the instance for

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

A representation of the eluted peptides and its identified proteins.

class `IPTK.Classes.Peptide.`**`Peptide`** (pep_seq: str)

Bases: **`object`**

An representation of an eluted peptide.

`add_org_2_parent` **(**prot_name: str**,** org: str**)** → None

add the source organism of one of the instance parent protein

Parameters:

- **prot_name** (*str*) – The name of the protein, i.e. the identifier of the protein
- **org** (*str*) – the name of the organism

Raises: **ValueError** – incase the provided protein is not a parent of the provided peptide

`add_parent_protein` **(**parent_protein**,** start_index: int**,** end_index: int**)** → None

add a protein instance as a parent to the current peptide. The library use Python-based indexing where its 0-indexed and ranges are treated as [start, end). :param parent_protein: a Protein instance that act as a parent to the peptide. :type parent_protein: Protein :param start_index: the position in the parent protein where the peptide starts :type start_index: int :param end_index: the index of the amino acid that occurs after the last amino acid in the peptide, :type start_index: int

`get_c_terminal_flank_seq` **(**flank_len: int**)** → List[str]

:param flank_len:the length of the flanking regions :type flank_len: int :return: a list of string containing the sequences located downstream of the peptide in the parent protein. :rtype: [type]

`get_flanked_peptide` (flank_len: int) → List[str]

Parameters: **flank_len** (*int*) – the length of the flanking regions

Returns: A list of string containing the length of the peptide + the flanking region from both the N and C terminal of the instance peptide, from all proteins.

Return type: Sequences

`get_length` () → int

Returns: the length of the peptides

Return type: int

`get_n_terminal_flank_seq` (flank_len: int) → List[str]

Parameters: **flank_len** (*int*) – the length of the flanking regions

Returns: a list of string containing the sequences located upstream of the peptide in the parent protein.

Return type: List[str]

`get_non_presented_peptides` (length: int) → List[str]

Parameters: **length** (*int*) – The length, i.e. number of amino acids, for the non-presented peptide

Returns: non-presented peptide from all the parent protein of the current peptide instance.

Return type: Sequences

`get_number_of_parents` () → int

Returns: the number of instance parent proteins

Return type: int

`get_number_parent_protein` () → int

Returns: the number of parent proteins this instance has

Return type: int

`get_parent` (pro_id: str)

Parameters: **pro_id** (*str*) – The protein identifer

Returns: the parent protein that has an id matching the user defined pro_id

Return type: Protein

`get_parent_proteins` () → List[str]

`get_parents_org` () → List[str]

Returns: a list containing the name of each parent protein source organisms

Return type: Organisms

`get_peptide_seq` () → str

Returns: the sequence of the peptide.

Return type: str

`get_pos_in_parent` (pro_id: str) → Tuple[int, int]

Parameters: **pro_id** (*str*) – the id of the parent protein

Raises: **ValueError** – If the identifier is not a parent of the instance

Returns: the start and end position of the instance peptide in the parent pointed out by the provided identifier

Return type: Range

`is_child_of` (pro_id: str) → bool

Parameters: **pro_id** (*str*) – is the protein id

Returns: True if a user provided protein-id is a parent for the instance peptide, False otherwise

Return type: bool

`map_to_parent_protein` () → List[numpy.ndarray]

Mapped the instance peptide to the parent protein and returned a list of numpy arrays where each array has a size of 1 by protein length. within the protein the range representing the peptide is encoded as one while the rest is zero.

Returns: A list of binary encoded arrays represent this mapping.

Return type: MappedProtein

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 that contain all the meta-data about the patient

Return type: dict

`get_name` () → str

Returns: the name of the patient

Return type: str

`update_info` (**info) → None

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

IPTK.Classes.Protein module

A representation of a protein that has been inferred from an IP experiment.

class `IPTK.Classes.Protein.`**`Protein`** (prot_id: str, seq: str, org: Optional[str] = None)

Bases: **`object`**

representation of a protein that has been infered from an IP experiment.

`get_id` () → str

Returns: return the protein identifier.

Return type: str

`get_non_presented_peptide` (exc_reg_s_idx: int, exc_reg_e_idx: int, length: int) → str

sample a peptide from the protein sequences where the sampled peptides is not part of the experimentally identified regions.

Parameters:

- **exc_reg_s_idx** (*int*) – the start point in the reference protein sequence of the experimentally identified peptide.
- **exc_reg_e_idx** (*int*) – the end point in the reference protein sequence of the experimentally identified peptide.
- **length** (*int*) – length the non-presented peptides.

Raises:
- **ValueError** – if the length of the peptide is bigger than the protein length
- **ValueError** – if the length of the peptide is smaller than or equal to zero

Returns: a substring of the instance sequence

Return type: str

`get_org` **()** → str

Returns: the organism in which this instance protein belong.

Return type: str

`get_peptides_map` **(**start_idxs: List[int]**,** end_idxs: List[int]**)** → numpy.ndarray

compute a coverage over the protein sequence

Parameters:
- **start_idxs** (*Index*) – a list of integers representing the start positions
- **end_idxs** – a list of integers representing the end positions

Raises: **ValueError** – if start_indxs and end_idxs MUST be of equal length are not of equal length

Returns: A numpy array with shape of 1 by the length of the protein where every element in the array donates the number of times, It has been observed in the experiment.

Return type: np.ndarray

`get_seq` **()** → str

Returns: the protein sequence.

Return type: str

`set_org` **(**org: str**)** → None

a post-instantitation mechanism to set the organism for which the protein belong.

Parameters: **org** (*str*) – the name of the organism

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 which the instance profile its gene expression

Return type: str

`get_table` **()** → pandas.core.frame.DataFrame

Returns: return a table that contain the expression of all the transcript 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

Parse different user inputs into a standard format/tables used by the library.

`IPTK.IO.InFunctions.`**`download_pdb_entry`** (prot_id: str) → str

Download the structure of a protein from protein databank form as mmCIF file.

Parameters: **prot_id** (*str*) – the protein id

Raises: **IOError** – incase downloading and accessing the data failed

Returns: the path to the downloaded file

Return type: str

`IPTK.IO.InFunctions.`**`fasta2dict`** (path2fasta: str, filter_decoy: bool = True, decoy_string: str = 'DECOY') → Dict[str, str]

loads a fasta file and construct a dict object where ids are keys and sequences are the value

Parameters:

- **path2fasta** (*str*) – The path to load the fasta file
- **filter_decoy** (*bool, optional*) – A boolean of whether or not to filter the decoy sequences from the database, defaults to True
- **decoy_string** (*str, optional*) – The decoy database prefix, only valid incase filter_decoy is set to true, defaults to 'DECOY'

Raises: **IOError** – [description]

Returns: a dict where the protein ids are the keys and the protein sequences are the value

Return type: Dict[str,str]

`IPTK.IO.InFunctions.`**`load_identification_table`** (input_path: str, sep: str) → pandas.core.frame.DataFrame

load & process an identification table

Parameters:

- **input_path** (*str*) – the path two the identification table. with the following columns: peptides which hold the identified peptide sequence, protein which hold the identified protein sequence, start_index, and end_index where the last two columns define the position of the peptide in the parent protein.
- **sep** (*str*) – The separator to parse the provided table.

Raises:

- **IOError** – [description]
- **ValueError** – [description]

Returns: [description]

Return type: pd.DataFrame

`IPTK.IO.InFunctions.`**`parse_mzTab_to_identification_table`** (path2mzTab: str, path2fastaDB: str, fasta_reader_param: Dict[str, str] = {'decoy_string': 'DECOY', 'filter_decoy': True}) → pandas.core.frame.DataFrame

parse a user provided mzTab to an identification table

Parameters:

- **path2mzTab** (*str*) – the path to the input mzTab file
- **path2fastaDB** (*str*) – the path to a fasta sequence database to obtain the protein sequences
- **fasta_reader_param** (*Dict[str,str], optional*) – a dict of parameters for controlling the behavior of the fasta reader , defaults to {'filter_decoy':True, 'decoy_string':'DECOY' }

Raises:

- **IOError** – if the mztab file could not be open and loaded or if the fasta database could not be read
- **KeyError** – if a protein id defined in the mzTab file could not be extracted from a matched sequence database
- **ValueError** – if the peptide can not be mapped to the identified protein

Returns: the identification table

Return type: pd.DataFrame

`IPTK.IO.InFunctions.`**`parse_text_table`** (path2file: str, path2fastaDB: str, sep=',', fasta_reader_param: Dict[str, str] = {'decoy_string': 'DECOY', 'filter_decoy': True}, seq_column: str = 'Sequence', accession_column: str = 'Protein Accessions', protein_group_sep: str = ';', remove_protein_version: bool = True, remove_if_not_matched: bool = True) → pandas.core.frame.DataFrame

Parse a user defined table to extract the columns containing the identification table

Parameters:

- **path2file** (*str*) – The path to load the CSV file holding the results
- **path2fastaDB** (*str*) – The path to a fasta sequence database to obtain the protein sequences
- **sep** (*str, optional*) – The table separators, defaults to ‘,’
- **fasta_reader_param** (*Dict[str,str], optional*) – a dict of parameters for controlling the behavior of the fasta reader, defaults to {‘filter_decoy’:True, ‘decoy_string’:’DECOY’ }
- **seq_column** (*str, optional*) – The name of the columns containing the peptide sequence, defaults to ‘Sequence’
- **accession_column** (*str, optional*) – The name of the column containing the protein accession , defaults to ‘Protein Accessions’
- **protein_group_sep** (*str, optional*) – The separator for the protein group,, defaults to ‘;’
- **remove_protein_version** (*bool, optional*) – A bool if true strip the version number from the protein , defaults to True
- **remove_if_not_matched** (*bool, optional*) – remove the peptide if it could not be matched to the parent protein, defaults to True

Raises:

- **IOError** – Incase either the sequences database or the identification table can not be open and loaded
- **KeyError** – In case the provided column names not in the provided identification table.
- **KeyError** – Incase the protein sequence can not be extract from the sequence database
- **ValueError** – incase the peptide could not be located in the protein sequence

Returns: an identification table

Return type: pd.DataFrame

`IPTK.IO.InFunctions.`**`parse_xml_based_format_to_identification_table`**(path2XML_file: str, path2fastaDB: str, decoy_prefix: str = 'DECOY', is_idXML: bool = False, fasta_reader_param: Dict[str, str] = {'decoy_string': 'DECOY', 'filter_decoy': True}, remove_if_not_matched: bool = True) → pandas.core.frame.DataFrame

parse either a pepXML or an idXML file to generate an identification table ,

Parameters:

- **path2XML_file** (*str*) – The path to the input pepXML files
- **path2fastaDB** (*str*) – The path to a fasta sequence database to obtain the protein sequences
- **decoy_prefix** (*str, optional*) – the prefix of the decoy sequences, default is DECOY, defaults to ‘DECOY’
- **is_idXML** (*bool, optional*) – Whether or not the provided file is an idXML, default is false which assume the provided file is a pepXML file, defaults to False
- **fasta_reader_param** (*Dict[str,str], optional*) – a dict of parameters for controlling the behavior of the fasta reader , defaults to {‘filter_decoy’:True, ‘decoy_string’:’DECOY’ }
- **remove_if_not_matched** (*bool, optional*) – remove the peptide if it could not be matched to the parent protein,, defaults to True

Raises:

- **IOError** – if the fasta database could not be open
- **ValueError** – if the XML file can not be open
- **KeyError** – if a protein id defined in the mzTab file could not be extracted from a matched sequence database
- **ValueError** – if the peptide can not be mapped to the identified protein

Returns: the identification table

Return type: pd.DataFrame

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

Write the results generated by the library into a wide variety of formats.

`IPTK.IO.OutFunctions.`**`write_annotated_sequences`** (peptides: List[str], labels: List[int], path2write: str, sep: str = ',', shuffle: bool = True) → None

take a list of peptides along with it sequences and write the results to a CSV file.

Parameters:

- **peptides** (*List[str]*) – a list of peptide sequences
- **labels** (*List[int]*) – a list of numerical labels associated with the peptides
- **path2write** (*str*) – the path to write the generated file
- **sep** (*str, optional*) – The separator in the resulting table, defaults to ','
- **shuffle** (*bool, optional*) – Whether or not to shuffle the table , defaults to True

Raises:

- **ValueError** – incase the length of the tables and labels is not matching
- **IOError** – In case writing the output table failed

`IPTK.IO.OutFunctions.`**`write_auto_named_peptide_to_fasta`** (peptides: List[IPTK.Classes.Peptide.Peptide], output_file: str) → None

Takes a list of peptides, generate automatic names for the peptides and write the results to the disk as FASTA files

Parameters:

- **peptides** (*Peptides*) – a list of peptide sequences
- **output_file** (*str*) – the name of the output file to write the results to, it will OVERWRITE existing files

`IPTK.IO.OutFunctions.`**`write_mapped_tensor_to_h5py`** (tensor: numpy.ndarray, path2write: str, dataSet_name: str = 'MAPPED_TENSOR') → None

Write a mapped tensor to an hdf5 file

Parameters:

- **tensor** (*np.ndarray*) – The provided tensor to write it to the hdf5 file.
- **path2write** (*str*) – The path of the output file
- **dataSet_name** (*str, optional*) – The name of the dataset inside the mapped tensor, defaults to 'MAPPED_TENSOR'

Raises: **IOError** – In case opening the file for writing failed

`IPTK.IO.OutFunctions.`**`write_named_peptides_to_fasta`** (names: List[str], peptides: List[IPTK.Classes.Peptide.Peptide], output_file: str)

Takes a list of names and peptide sequences and writes them as an output file to the disk as fasta files

Parameters:

- **names** (*Names*) – A list of sequences names
- **peptides** (*Peptides*) – A list of peptide sequences
- **output_file** (*str*) – The name of the output file to write the results to, it will OVERWRITE existing files

Raises:

- **ValueError** – Incase the length of the tables and labels is not matching
- **IOError** – In case writing the output file failed

`IPTK.IO.OutFunctions.`**`write_pep_file`** (peptides: List[IPTK.Classes.Peptide.Peptide], output_file: str)

Takes a file and write the peptides to .pep file which is a text file that contain one peptide per line

Parameters:

- **peptides** (*Peptides*) – a list of peptide sequences
- **output_file** (*str*) – the name of the output file to write the results to, it will OVERWRITE existing files

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

Module contents

IPTK.Utils package

Submodules

IPTK.Utils.DevFunctions module

The module contain functions that can be used for developing & testing other functions of the library

`IPTK.Utils.DevFunctions.`**`generate_random_peptide_seq`** (peptide_length: int, num_peptides: int) → List[str]

generate a list of random peptides for testing and developing purposes.

Parameters:

- **peptide_length** (*int*) – The peptide length
- **num_peptides** (*int*) – The number of peptides in the generate list

Returns: a list of random peptides

Return type: List[str]

`IPTK.Utils.DevFunctions.`**`simulate_an_experimental_ident_table_from_fasta`** (path2load: str, num_pep: int, num_prot: int) → pandas.core.frame.DataFrame

simulate an IP identification table from a fasta file. Please Note, if the reminder of num_pep over num_prot does not equal to zero, the floor of this ratio will be used to sample peptides from each proteins

Parameters:

- **path2load** (*str*) – The path to load the Fasta files
- **num_pep** (*int*) – The number of peptides in the tables
- **num_prot** (*int*) – The number of UNIQUE proteins in the table

Raises: **ValueError** – if number of proteins or number of peptide is zero

Returns: an identification table

Return type: pd.DataFrame

`IPTK.Utils.DevFunctions.`**`simulate_an_expression_table`** (num_transcripts: int = 100) → pandas.core.frame.DataFrame

create a dummy expression table to be used for testing and developing Tissue based classes

Parameters: **num_transcripts** (*int, optional*) – The number of transcripts that shall be contained in the transcript , defaults to 100

Raises: **ValueError** – incase number of transcripts is 0

Returns: [description]

Return type: pd.DataFrame

`IPTK.Utils.DevFunctions.`**`simulate_mapped_array_list`** (min_len: int = 20, max_len: int = 100, num_elem: int = 100) → List[numpy.ndarray]

Simulate a list of mapped arrays proteins to be used for developing purposes

Parameters:

- **min_len** (*int, optional*) – the minmum length of the protein , defaults to 20
- **max_len** (*int, optional*) – the maximum length for the protein , defaults to 100
- **num_elem** (*int, optional*) – the number of arrays in the protein , defaults to 100

Returns: a list of simulated NumPy array that represent protein peptide coverage

Return type: List[np.ndarray]

`IPTK.Utils.DevFunctions.`**`simulate_random_experiment`** (alleles: List[str], path2fasta: str, tissue_name: str = 'TEST_TISSUE', num_pep: int = 10, num_prot: int = 5, proband_name: str = None) → IPTK.Classes.Experiment.Experiment

Simulate a random experiment objects

Parameters:

- **alleles** (*List[str]*) – a list of alleles names.
- **path2fasta** (*str*) – The path to load the database objects
- **tissue_name** (*str, optional*) – the name of the tissue, defaults to 'TEST_TISSUE'
- **num_pep** (*int, optional*) – the number of peptides in the table, defaults to 10
- **num_prot** (*int, optional*) – number of proteins, defaults to 5
- **proband_name** (*str, optional*) – The name of the Proband, defaults to None

Returns: a simulated experimental object

Return type: Experiment

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.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 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.`**`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 contain a simulate protein coverage

Return type: np.ndarray

`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 thr 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 library.Default is pre-padding, defaults to True
- **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.

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 contains that the condition index and a simulated protein array

Return type: Dict[str, np.ndarray]

Module contents

IPTK.Visualization package

Submodules

IPTK.Visualization.vizTools module

The module contain visualization functions the can be used to plot the results obtained from the methods of the classes defined in the Class module or from the analysis functions defined in the Analysis Module.

`IPTK.Visualization.vizTools.`**`imposed_coverage_on_3D_structure`** (path2mmCIF: str, mapped_protein: numpy.ndarray, background_color: str = 'black', low: str = 'red', high: str = 'blue') → None

A function to impose the peptide coverage on top of a protein 3D structure where the color of each position is marked by a color gradient that reflect the number of peptides aligned to this position. Note: Use the function with Jupyter-note book as it return an NGLWidget object that your can explore and navigate from the browser.

Parameters:

- **path2mmCIF** (*str*) – The path to load the mmCIF file containing the protein structure
- **mapped_protein** (*np.ndarray*) – a Numpy array of containg the number of peptides originated from each position in the array
- **background_color** (*str, optional*) – the color of the background, default is black , defaults to 'black'
- **low** (*str, optional*) – the color of low covered position, default is red. , defaults to 'red'
- **high** (*str, optional*) – the color of high covered position, default is violet., defaults to 'blue'

Raises:

- **ValueError** – incase the provided path to the structure file does not exists
- **IOError** – if the structure file can not be loaded or if more than one file are located in the provided path

`IPTK.Visualization.vizTools.`**`plot_change_in_presentation_between_experiment`** (change_in_presentation_array: numpy.ndarray, index_first: int, index_second: int, plotting_kwargs: Dict[str, str] = {}, title='Change in protein presentation', xlabel='Proteins', ylabel='magnitude of change in protein count') → matplotlib.figure.Figure

plot the change in protein presentation between two experiment

Parameters:

- **change_in_presentation_array** (*np.ndarray*) – a 3D tensor of shape number of experiments by number of experiment by number of identified proteins.
- **index_first** (*int*) – [description]
- **index_second** (*int*) – the index of the first experiment in the tensor
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.scatterplot function, defaults to {}
- **title** (*str, optional*) – The title of the figure, defaults to 'Change in protein presentation'
- **xlabel** (*str, optional*) – The axis on the x-axis , defaults to "Proteins"
- **ylabel** (*str, optional*) – The axis on the y-axis, defaults to "magnitude of change in protein count"

Raises:
- **ValueError** – if the provided tensor is not of rank 3
- **IndexError** – if the provided indices are out of bound

`IPTK.Visualization.vizTools.`**`plot_coverage_and_annotation`** (protein_coverage: Dict[str, numpy.ndarray], temp_dir: str = '.', figure_size: Tuple[int] = 7, 8, figure_dpi: int = 600, coverage_track_dict: Dict[str, Dict[str, Union[str, dict]]] = {'coverage_dict': {'color': 'grey', 'width': 1.2}}, chains_track: bool = True, chains_track_dict: Dict[str, Dict[str, Union[str, dict]]] = {'track_elements_dict': {'capstyle': 'butt', 'color': 'magenta'}, 'track_label_dict': {'color': 'black', 'fontsize': 6}}, domains_track: bool = True, domain_track_dict: Dict[str, Dict[str, Union[str, dict]]] = {'track_element_names_dict': {'color': 'black', 'fontsize': 4}, 'track_elements_dict': {'capstyle': 'butt', 'color': 'blue'}, 'track_label_dict': {'color': 'black', 'fontsize': 6}}, modifications_track: bool = True, modifications_track_dict: Dict[str, Dict[str, Union[str, dict]]] = {'height_frac': 0.5, 'track_label_dict': {'color': 'black', 'fontsize': 6}}, glyco_track: bool = True, glyco_track_dict: Dict[str, Dict[str, Union[str, dict]]] = {'height_frac': 0.5, 'track_label_dict': {'color': 'black', 'fontsize': 6}}, disulfide_track: bool = True, disulfide_track_dict={'height_frac': 0.5, 'track_label_dict': {'color': 'black', 'fontsize': 6}}, sequence_variants_track: bool = True, sequence_variants_track_dict: Dict[str, Dict[str, Union[str, dict]]] = {'height_frac': 0.5, 'track_label_dict': {'color': 'black', 'fontsize': 6}}, splice_variants_track: bool = True, splice_variants_track_dict: bool = {'track_elements_dict': {'capstyle': 'butt', 'color': 'green'}, 'track_label_dict': {'color': 'black', 'fontsize': 6}}) → matplotlib.figure.Figure

The function plot the annotation track which summarizes all the known information about the protein and its associated peptides.

protein_coverage: Dict[str,np.ndarray]
A dict that contain the uniprot accession of the protein as key and the protein coverage as value

temp_dir : *str*
The name of the temp directory to download the protein XML scheme to it.

figure_size: list/tuple, optional
The figure size in inches. The default is 7 by 8 in which is (17.8*20.32) cm.

figure_dpi: int, optional
The resolution of the figure in dots-per-inch. The default is 600.

coverage_track : *bool, optional*
whether or not to plot the coverage track. The default is True.

coverage_track_dict: dict, optional
The parameters of the function `Annotator.add_coverage_track`. it is only used if coverage_track is set to True. The default is {"coverage_dict":{"color":"grey","width":1.2}}.

chains_track : *bool, optional*
whether or not to plot the chain track. The default is True.

chains_track_dict: dict, optional
The parameters of the function `Annotator.add_stacked_track`. it is only used if chains_track is set to True. The default is {"track_label_dict":{"fontsize":6,"color":"black"}, "track_elements_dict":{"color":"magenta","capstyle":"butt"}}.

domains_track : *bool, optional*
whether or not to plot the domains track. The default is True.

domains_track_dict: dict, optional
The parameters of the function `Annotator.add_segmented_track`. it is only used if domains_track is set to True. The default is {"track_label_dict":{"fontsize":6,"color":"black"}, "track_element_names_dict":{"fontsize":4,"color":"black"}, "track_elements_dict":{"color":"blue","capstyle":"butt"}}

modifications_track : *bool, optional*
whether or not to plot the generic modification track. The default is True.

modifications_track_dict: dict, optional
The parameters of the function `Annotator.add_marked_positions_track`. it is only used if modifications_track is set to True. The default is {"height_frac":0.5, "track_label_dict":{"fontsize":6,"color":"black"}}

glyco_track : *bool, optional*

whether or not to plot the generic glycosylation track. The default is True.

glyco_track_dict: dict, optional

The parameters of the function `Annotator.add_marked_positions_track`. it is only used if glyco_track is set to True. The default is {“height_frac”:0.5, “track_label_dict”:{“fontsize”:6,”color”:”black”}}

disulfide_track : *bool, optional*

whether or not to plot the generic disulfide bond track. The default is True.

disulfide_track_dict: dict, optional

The parameters of the function `Annotator.add_marked_positions_track`. it is only used if disulfide_track is set to True. The default is {“height_frac”:0.5, “track_label_dict”:{“fontsize”:6,”color”:”black”}}

sequence_varients_track : *bool, optional*

whether or not to plot the sequence varients track. The default is True.

sequence_varients_track_dict: dict, optional

The parameters of the function `Annotator.add_marked_positions_track`. it is only used if sequence_varients_track is set to True. The default is {“height_frac”:0.5, “track_label_dict”:{“fontsize”:6,”color”:”black”}}.

splice_varient_track : *bool, optional*

whether or not to plot the solice varients track. The default is True.

splice_varients_track_dict:

The parameters of the function `Annotator.add_stacked_track`. it is only used if chains_track is set to True. The default is {“track_label_dict”:{“fontsize”:6,”color”:”black”}, “track_elements_dict”:{“color”:”magenta”,”capstyle”:”butt”}}.

In the following series of examples we are going to optimize the plotting of an annotator track for the Syntenin-1 protein, protein accession O00560 which has 298 amino acids

## first, let’s simulate some coverage data >>> sim_coverage: np.ndarray = np.concatenate([np.zeros(50),np.ones(50),np.zeros(198)]) >>> panel_trial1: plt.Figure = plot_coverage_and_annotation({‘O00560’:sim_coverage})

Ass seen from panel_trial1 the figure looks overcrowded an un-balanced. To polish the figure, let’s first start by adjusting the size of the axes’ axis

```
>>> panel_trial2: plt.Figure = plot_coverage_and_annotation(
    {'O00560':sim_coverage},
    coverage_track_dict={
        "xlabel_dict":{
                        "fontsize":2
                        },
        "ylabel_dict":{
                        "fontsize":2
                        }
                    },
    chains_track_dict={
        "track_label_dict":{
                        "fontsize":2
                            }
                    },
    domain_track_dict={
        "track_label_dict":{
                        "fontsize":2
                            }
                    },
```

modifications_track_dict={

“track_label_dict”:{

"fontsize":2 }

}

)

Okay, 2nd trial looks much better than the first trial, but we can polish it further by adjusting the font size and the size of domains and chains

```
>>> panel_trial3: plt.Figure = plot_coverage_and_annotation(
    {'O00560':sim_coverage},
    coverage_track_dict={
        "xlabel_dict":{
                        "fontsize":2
                        },
        "ylabel_dict":{
                        "fontsize":2
                        }
                    },
    chains_track_dict={
        "track_label_dict":{
                        "fontsize":2
                          },
        "track_element_names_dict":{
                                "fontsize":2
                                    }
                    },
    domain_track_dict={
        "track_label_dict":{
                        "fontsize":2
                          },
```

"track_element_names_dict":{

"fontsize":2

}

},

modifications_track_dict={

"track_label_dict":{

"fontsize":2

}

}

)

Okay, trial 3 looks much better than the previous tow trials, let's polish the graph further by adjust the size of elments in the track as well as font-size

```
>>> panel_trial4: plt.Figure = plot_coverage_and_annotation(
    {'O00560':sim_coverage},
    coverage_track_dict={
        "xlabel_dict":{
                        "fontsize":3
                        },
```

"ylabel_dict":{

"fontsize":3 },

"coverage_dict":{

"color":"grey", "width":1,

}, "number_ticks":25,

},

chains_track_dict={

"track_label_dict":{

"fontsize":3

},

"track_element_names_dict":{

"fontsize":3

},

"track_elements_dict":{

"color":"blue"

}

},

domain_track_dict={

"track_label_dict":{

"fontsize":3

},

"track_element_names_dict":{

"fontsize":3 },

"track_elements_dict":{

"color":"lime"

}

},

modifications_track_dict={

"track_label_dict":{

"fontsize":3

},

"marker_bar_dict": {

"color":"black", "linestyles":"solid", "linewidth":0.5, "alpha":0.4 },

"marker_dict":{

"s":2, "color":"red"

}

}

)

vispanel: matplotlib.figure.Figure

The resulting panel.

`IPTK.Visualization.vizTools.`**`plot_experiment_set_counts`** (counts_table: pandas.core.frame.DataFrame, log_scale: bool = False, plotting_kwargs: Dict[str, str] = {}) → matplotlib.pyplot.figure

visualize the number of peptides and number of peptides-per-organism per experiment.

Parameters:
- **counts_table** (*pd.DataFrame*) – a pandas dataframe that contain the count, organism name and the count
- **log_scale** (*bool, optional*) – Normalize the peptide counts one log 10, defaults to False
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.catplot function, defaults to {}

`IPTK.Visualization.vizTools.`**`plot_gene_expression_vs_num_peptides`** (exp_count_table: pandas.core.frame.DataFrame, tissue_name: str, def_value: float = - 1, plotting_kwargs: Dict[str, str] = {}, xlabel: str = 'Number of peptides', ylabel: str = 'Expression value', title: str = 'Peptides per protein Vs. Expression Level') → matplotlib.figure.Figure

Plot the correlation between the gene expression and the num of peptids per protein using seaborn library

Parameters:
- **exp_count_table** (*pd.DataFrame*) – A table that contain the number of peptides and the expresion value for each protein in the database
- **tissue_name** (*str*) – The name of the tissue
- **def_value** (*float, optional*) – The default value for proteins that could not be mapped to the expression database, defaults to -1
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.scatter function, defaults to {}
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of peptides'
- **ylabel** (*str, optional*) – the label on the y-axis, defaults to 'Expression value'
- **title** (*str, optional*) – The title of the figure, defaults to 'Peptides per protein Vs. Expression Level'

`IPTK.Visualization.vizTools.`**`plot_motif`** (pwm_df: pandas.core.frame.DataFrame, plotting_kwargs: Dict[str, str] = {'fade_probabilities': True}) → matplotlib.figure.Figure

A generic motif plotter that forward its argument to logomaker for making plots

Parameters:
- **pwm_df** (*pd.DataFrame*) – A pandas dataframe containing the position weighted matrix
- **plotting_kwargs** (*PlottingKeywards, optional*) – A dictionary of parameter to control the behavior of the logo plotter, defaults to {'fade_probabilities':True}

Returns: a matplotlib figure instance containing the ploted motif

Return type: plt.Figure

`IPTK.Visualization.vizTools.`**`plot_num_peptide_per_go_term`** (pep2goTerm: pandas.core.frame.DataFrame, plotting_kwargs: Dict[str, str] = {}, drop_unknown: bool = False, xlabel: str = 'Number of peptides', ylabel: str = 'GO-Term', title: str = 'Number of peptides per GO Term') → matplotlib.figure.Figure

plot the number of peptides obtained per Go-Term using matplotlib library.

Parameters:
- **pep2goTerm** (*pd.DataFrame*) – A table that contain the count of peptides from each GO-Term
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.barplot function, defaults to {}
- **drop_unknown** (*bool, optional*) – whether or not to drop peptide with unknown GO-term, defaults to False
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of peptides'
- **ylabel** (*str, optional*) – the label on the y-axis, defaults to 'GO-Term'
- **title** (*str, optional*) – The title of the figure, defaults to 'Number of peptides per GO Term'

Returns: [description]

Return type: plt.Figure

`IPTK.Visualization.vizTools.`**`plot_num_peptides_per_location`** (pep2loc: pandas.core.frame.DataFrame, plotting_kwargs: Dict[str, str] = {}, drop_unknown: bool = False, xlabel: str = 'Number of peptides', ylabel: str = 'Compartment', title: str = 'Number of peptides per sub-cellular compartment') → matplotlib.figure.Figure

plot the number of peptides obtained from each compartment using seaborn library.

Parameters:

- **pep2loc** (*pd.DataFrame*) – A table that contain the count of peptides from each location
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.barplot function, defaults to {}
- **drop_unknown** (*bool, optional*) – whether or not to drop protein with unknown location, defaults to False
- **xlabel** (*str, optional*) – The label on the x-axis, defaults to 'Number of peptides'
- **ylabel** (*str, optional*) – The label on the y-axis, defaults to 'Compartment'
- **title** (*str, optional*) – The title of the figure, defaults to 'Number of peptides per sub-cellular compartment'

`IPTK.Visualization.vizTools.`**`plot_num_peptides_per_organism`** (pep_per_org: pandas.core.frame.DataFrame, log_scale: bool = False, plotting_kwargs: Dict[str, str] = {}, xlabel: str = 'Number of peptides', ylabel: str = 'Organism', title: str = 'Number of peptides per organism') → matplotlib.figure.Figure

plot the number of peptides per each organism inferred from the experiment using seaborn and matlotlib.

Parameters:

- **pep_per_org** (*pd.DataFrame*) – A table that contain the number of peptides belonging to each organism
- **log_scale** (*bool, optional*) – Whether or not to scale the number of peptides using a log scale, default is False, defaults to False
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.barplot function, defaults to {}
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of peptides'
- **ylabel** (*str, optional*) – The label on the y-axis, defaults to 'Organism'
- **title** (*str, optional*) – The title of the figure, defaults to 'Number of peptides per organism'

`IPTK.Visualization.vizTools.`**`plot_num_peptides_per_parent`** (nums_table: pandas.core.frame.DataFrame, num_prot: int = - 1, plotting_kwargs: Dict[str, str] = {}, x_label: str = 'Number of peptides', y_label: str = 'Protein ID', title: str = 'Number of peptides per protein')

Visualize a histogram of the eluted peptide length.

Parameters:

- **nums_table** (*pd.DataFrame*) – a pandas dataframe containing number of peptides identified from each protein.
- **num_prot** (*int, optional*) – the number of protein to show relative to the first element, for example, the first 10, 20 etc. If the default value of -1 is used then all protein will be plotted, however, this might lead to a crowded figure, defaults to -1.
- **plotting_kwargs** – a dict object containing parameters for the function seaborn::distplot, defaults to {}
- **x_label** (*str, optional*) – the label of the x-axis, defaults to 'Number of peptides'
- **y_label** (*str, optional*) – the label of the y-axis, defaults to 'Protein ID'
- **title** (*str, optional*) – The title of the figure, defaults to 'Number of peptides per protein'

Raises: **ValueError** – if the num_prot is bigger than the number of elements in the provided table

`IPTK.Visualization.vizTools.`**`plot_num_protein_per_go_term`** (protein2goTerm: pandas.core.frame.DataFrame, tissue_name: str, plotting_kwargs: Dict[str, str] = {}, drop_unknown: bool =

False, xlabel: str = 'Number of Proteins', ylabel: str = 'Compartment', title: str = 'Number of proteins per sub-cellular compartment') → matplotlib.figure.Figure

plot the number of proteins per each GO Term

Parameters:

- **protein2goTerm** (*pd.DataFrame*) – A table that contain the count of proteins from each GO-Term
- **tissue_name** (*str*) – a dict object containing parameters for the sns.barplot function.
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.barplot function, defaults to {}
- **drop_unknown** (*bool, optional*) – whether or not to drop protein with unknown location, defaults to False
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of Proteins'
- **ylabel** (*str, optional*) – the label on the y-axis, defaults to 'Compartment'
- **title** (*str, optional*) – the title of the figure, defaults to 'Number of proteins per sub-cellular compartment'

`IPTK.Visualization.vizTools.`**`plot_num_protein_per_location`** (protein_loc: pandas.core.frame.DataFrame, plotting_kwargs: Dict[str, str] = {}, drop_unknown: bool = False, xlabel: str = 'Number of Proteins', ylabel: str = 'Compartment', title: str = 'Number of proteins per sub-cellular compartment') → matplotlib.figure.Figure

plot the number of proteins per each sub-cellular compartment

Parameters:

- **protein_loc** (*pd.DataFrame*) – A table that contain the count of protein from each location.
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.barplot function, defaults to {}
- **drop_unknown** (*bool, optional*) – whether or not to drop protein with unknown location, defaults to False
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of Proteins'
- **ylabel** (*str, optional*) – the label on the y-axis, defaults to 'Compartment'
- **title** (*str, optional*) – the title of the figure, defaults to 'Number of proteins per sub-cellular compartment'

`IPTK.Visualization.vizTools.`**`plot_overlap_heatmap`** (results_df: pandas.core.frame.DataFrame, plotting_kwargs: Dict[str, str] = {}) → seaborn.matrix.ClusterGrid

Plot a user provided dataframe as a cluster heatmap using seaborn library

Parameters:

- **results_df** (*pd.DataFrame*) – A pandas dataframe table that hold the overlapping number
- **plotting_kwargs** (*PlottingKeywards, optional*) – forward parameter to the clustermap function, defaults to {}

Returns: the generated clustermap

Return type: sns.matrix.ClusterGrid

`IPTK.Visualization.vizTools.`**`plot_overlay_representation`** (proteins: Dict[str, Dict[str, numpy.ndarray]], alpha: float = 0.5, title: str = None, legend_pos: int = 2) → matplotlib.figure.Figure

plot an overlayed representation for the SAME protein or proteins OF EQUAL LENGTH in different conditions in which the mapped represention of each protein are stacked on top of each other to generate a representation for the protein representability under different conditions.

Parameters:

- **proteins** (*Dict[str,np.ndarray]]*) – a nested dict object containing for each protein a child dict that contain the mapping array and the color in the figure.
- **alpha** (*float, optional*) – the transparency between proteins , defaults to 0.5
- **title** (*str, optional*) – The title of the figure, defaults to None
- **legend_pos** (*int, optional*) – the position of the legend , defaults to 2

Raises: **ValueError** – if the provided protein have different lengths

Returns: a matplotlib figure containing the mapped representation

Return type: plt.Figure

`IPTK.Visualization.vizTools.`**`plot_paired_represention`** (protein_one_repr: Dict[str, numpy.ndarray], protein_two_repr: Dict[str, numpy.ndarray], color_first: str = 'red', color_second: str = 'blue', alpha: float = 0.9, title=' Parired protein representation') → matplotlib.figure.Figure

visualize the difference in representation for the same protein between two experiments using matplotlib library.

Parameters:

- **protein_one_repr** (*Dict[str, np.ndarray]*) – a dict object containing the legand of the first condition along with its mapped array
- **protein_two_repr** (*Dict[str, np.ndarray]*) – a dict object containing the legand of the second condition along with its mapped array
- **alpha** (*the transparency of the figure.*) – the transparency of the figure.

Param: color_first: the color of representation for the first condition

Param: color_second: the color of the second condition

Param: title: the title of the figure.

Returns: A matplotlib Figure containing the representation

Return type: plt.Figure

`IPTK.Visualization.vizTools.`**`plot_parent_protein_expression_in_tissue`** (expression_table: pandas.core.frame.DataFrame, ref_expression: pandas.core.frame.DataFrame, tissue_name: str, sampling_num: int = 10, plotting_kwargs: Dict[str, str] = {'orient': 'v'}, def_value: float = - 1, ylabel: str = 'Normalized Expression') → matplotlib.figure.Figure

Plot the parent protein expression in tissue relative a sampled collection of non-presented data using seaborn library.

Parameters:

- **expression_table** (*pd.DataFrame*) – The protein expression table which contains the expresion value for each parent protein
- **ref_expression** (*pd.DataFrame*) – The reference expression of the tissue under investigation.
- **tissue_name** (*str*) – The name of the tissue .
- **sampling_num** (*int, optional*) – The number of times to sample from the non-prsenter. , defaults to 10
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.violinplot function., defaults to {'orient':'v'}
- **def_value** (*float, optional*) – The default value for proteins that could not be mapped to the expression database , defaults to -1
- **ylabel** (*str, optional*) – the label on the y-axis. , defaults to 'Normalized Expression'

Raises: **ValueError** – if the reference gene expression table is smaller than the number of parents

`IPTK.Visualization.vizTools.`**`plot_peptide_length_dist`** (pep_length: List[int], plotting_kwargs: Dict[str, str] = {}, x_label: str = 'Peptide Length', y_label: str = 'Frequency', title: str = 'Peptide Length distribution')

Visualize a histogram of the eluted peptide length using seaborn library.

Parameters:

- **pep_length** (*List[int]*) – [description]
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the function seaborn::distplot, defaults to {}
- **x_label** (*str, optional*) – the label of the x-axis , defaults to 'Peptide Length'
- **y_label** (*str, optional*) – the label of the y-axis , defaults to 'Frequency'
- **title** (*str, optional*) – the title of the figure, defaults to 'Peptide Length distribution'

`IPTK.Visualization.vizTools.`**`plot_peptide_length_per_experiment`** (counts_table: pandas.core.frame.DataFrame, plotting_kwargs: Dict[str, str] = {}) → matplotlib.pyplot.figure

visualize the peptide length distribution among the experiments defined in the set

Parameters:

- **counts_table** (*pd.DataFrame*) – a pandas dataframe that contain the length of each peptide defined in the experiment along with the experiment name
- **plotting_kwargs** (*Dict[str,str], optional*) – a dict object containing parameters for the sns.catplot function, defaults to {}

`IPTK.Visualization.vizTools.`**`plot_protein_coverage`** (mapped_protein: numpy.ndarray, col: str = 'r', prot_name: str = None) → matplotlib.figure.Figure

plot the peptide coverage for a given protein.

Parameters:

- **mapped_protein** (*np.ndarray*) – a NumPy array with shape of 1 by protein length or shape protein-length
- **col** (*str, optional*) – the color of the coverage respresentation, defaults to 'r'
- **prot_name** (*str, optional*) – The default protein name, defaults to None

Return type: plt.Figure

`IPTK.Visualization.vizTools.`**`plot_protein_presentation_3D`** (proteins: Dict[str, Dict[str, numpy.ndarray]], plotting_args={'color': 'red'}, title: str = None) → matplotlib.figure.Figure

plot a 3D surface representation for the SAME protein or proteins OF EQAUL LENGTH in different conditions.

Parameters:

- **proteins** (*[type]*) – a nested dict object containing for each protein a child dict that contain the mapping array and the color in the figure.
- **plotting_args** (*dict, optional*) – a dict that contain further parameter to the plot_surface functions, defaults to {'color':'red'}
- **title** (*str, optional*) – the title of the figure, defaults to None

Raises: **ValueError** – if the provided proteins are of different length

Return type: plt.Figure

`IPTK.Visualization.vizTools.`**`plotly_gene_expression_vs_num_peptides`** (exp_count_table: pandas.core.frame.DataFrame, tissue_name: str, def_value: float = - 1, xlabel: str = 'Number of peptides', ylabel: str = 'Expression value', title: str = 'Peptides per protein Vs. Protein Expression Level') → matplotlib.figure.Figure

Plot the correlation between the gene expression and the number of peptids per protein using plotly library.

Parameters:

- **exp_count_table** (*pd.DataFrame*) – A table that contain the number of peptides and the expresion value for each protein in the database
- **tissue_name** (*str*) – The name of the tissue
- **def_value** (*float, optional*) – The default value for proteins that could not be mapped to the expression database , defaults to -1
- **xlabel** (*str, optional*) – The label on the x-axis, defaults to 'Number of peptides'
- **ylabel** (*str, optional*) – The label on the y-axis., defaults to 'Expression value'
- **title** (*str, optional*) – The title of the figure, defaults to 'Peptides per protein Vs. Protein Expression Level'

`IPTK.Visualization.vizTools.`**`plotly_multi_traced_coverage_representation`** (proteins: Dict[str, Dict[str, numpy.ndarray]], title: str = 'Protein Coverage Across ') → plotly.graph_objs._figure.Figure

plot a multi-traced representation for the same protein accross using plotly library

Parameters:

- **proteins** (*[type]*) – A dict object containing for each protein the corresponding mapped array.
- **title** (*str, optional*) – the title of the figure, defaults to "Protein Coverage Across "

Returns: a multitraced traced figure showing the coverage of proteins accross different conditions

Return type: Figure

`IPTK.Visualization.vizTools.`**`plotly_num_peptide_per_go_term`** (pep2goTerm: pandas.core.frame.DataFrame, drop_unknown: bool = False, xlabel: str = 'Number of peptides', ylabel: str = 'GO-Term', title: str = 'Number of peptides per GO Term') → plotly.graph_objs._figure.Figure

plot the number of peptides obtained per Go-Term using plotly library.

Parameters:

- **pep2goTerm** (*pd.DataFrame*) – A table that contain the count of peptides from each GO-Term
- **drop_unknown** (*bool, optional*) – whether or not to drop peptide with unknown GO-term, defaults to False
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of peptides'
- **ylabel** (*str, optional*) – the label on the y-axis, defaults to 'GO-Term'
- **title** (*str, optional*) – the title of the figure, defaults to 'Number of peptides per GO Term'

`IPTK.Visualization.vizTools.`**`plotly_num_peptides_per_location`** (pep2loc: pandas.core.frame.DataFrame, drop_unknown: bool = False, xlabel: str = 'Number of peptides', ylabel: str = 'Compartment', title: str = 'Number of peptides per sub-cellular compartment') → matplotlib.figure.Figure

plot the number of peptides obtained from each compartment using plotly library

Parameters:

- **pep2loc** (*pd.DataFrame*) – A table that contain the count of peptides from each location
- **drop_unknown** (*bool, optional*) – whether or not to drop protein with unknown location, defaults to False
- **xlabel** (*str, optional*) – The label on the x-axis, defaults to 'Number of peptides'
- **ylabel** (*str, optional*) – The label on the y-axis, defaults to 'Compartment'
- **title** (*str, optional*) – The title of the figure, defaults to 'Number of peptides per sub-cellular compartment'

`IPTK.Visualization.vizTools.`**`plotly_num_peptides_per_organism`** (pep_per_org: pandas.core.frame.DataFrame, log_scale: bool = False, xlabel: str = 'Number of Peptides', ylabel: str = 'Organism', title: str = 'Number of peptides per organism') → plotly.graph_objs._figure.Figure

plot the number of peptides per each organism inferred from the experiment using plotly library.

Parameters:

- **pep_per_org** (*pd.DataFrame*) – A table that contain the count of peptides from each organism
- **log_scale** (*bool, optional*) – Whether or not to scale the number of peptide using a log scale, defaults to False
- **xlabel** (*str, optional*) – The label on the x-axis, defaults to 'Number of Peptides'
- **ylabel** (*str, optional*) – The label on the y-axis, defaults to 'Organism'
- **title** (*str, optional*) – the title of the figure , defaults to 'Number of peptides per organism'

`IPTK.Visualization.vizTools.`**`plotly_num_peptides_per_parent`** (nums_table: pandas.core.frame.DataFrame, num_prot: int = - 1, x_label: str = 'Number of peptides', y_label: str = 'Protein ID', title: str = 'Number of peptides per protein')

Visualize a histogram of the the number of peptides per each inferred protein.

Parameters:

- **nums_table** (*pd.DataFrame*) – a pandas dataframe containing number of peptides identified from each protein.
- **num_prot** – the number of protein to show relative to the first element, for example, the first 10, 20 etc. If the default value of -1 is used then all protein will be plotted, however, this might lead to a crowded figure., defaults to -1 :type num_prot: int, optional
- **x_label** (*str, optional*) – the label of the x-axis , defaults to 'Number of peptides'
- **y_label** (*str, optional*) – the label of the y-axis , defaults to 'Protein ID'
- **title** (*str, optional*) – title, defaults to 'Number of peptides per protein'

Raises: **ValueError** – if the num_prot is bigger than the number of elements in the provided table

`IPTK.Visualization.vizTools.`**`plotly_num_protein_per_go_term`** (protein2goTerm: pandas.core.frame.DataFrame, drop_unknown: bool = False, xlabel: str = 'Number of Proteins', ylabel: str = 'GO-Term', title: str = 'Number of proteins per GO-Term') → plotly.graph_objs._figure.Figure

plot the number of proteins per each GO Term using plotly library

Parameters:

- **protein2goTerm** (*pd.DataFrame*) – A table that contain the count of proteins from each GO-Term
- **drop_unknown** (*bool, optional*) – whether or not to drop protein with unknown location, defaults to False
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of Proteins'
- **ylabel** (*str, optional*) – the label on the y-axis, defaults to 'GO-Term'
- **title** (*str, optional*) – the title of the figure, defaults to 'Number of proteins per GO-Term'

Returns: [description]

Return type: Figure

`IPTK.Visualization.vizTools.`**`plotly_num_protein_per_location`** (protein_loc: pandas.core.frame.DataFrame, drop_unknown: bool = False, xlabel: str = 'Number of Proteins', ylabel: str = 'Compartment', title: str = 'Number of proteins per sub-cellular compartment') → plotly.graph_objs._figure.Figure

plot the number of proteins per each sub-cellular compartment

Parameters:

- **protein_loc** (*pd.DataFrame*) – A table that contain the count of protein from each location
- **drop_unknown** (*bool, optional*) – whether or not to drop protein with unknown location, defaults to False
- **xlabel** (*str, optional*) – the label on the x-axis, defaults to 'Number of Proteins'
- **ylabel** (*str, optional*) – the label on the y-axis, defaults to 'Compartment'
- **title** (*str, optional*) – [description], defaults to 'Number of proteins per sub-cellular compartment'

`IPTK.Visualization.vizTools.`**`plotly_overlap_heatmap`** (results_df: pandas.core.frame.DataFrame) → plotly.graph_objs._figure.Figure

Plot a user provided dataframe as a heatmap using plotly library

Parameters: **results_df** (*pd.DataFrame*) – A pandas dataframe table that hold the overlapping number.

Returns: a plotly Figure containing the heatmap

Return type: Figure

`IPTK.Visualization.vizTools.`**`plotly_paired_representation`** (protein_one_repr: Dict[str, numpy.ndarray], protein_two_repr: Dict[str, numpy.ndarray], title: str = ' Parired protein representation') → plotly.graph_objs._figure.Figure

compare the peptide coverage for the same protein under different conditions using the same protein using plotly library.

Parameters:

- **protein_one_repr** (*Dict[str, np.ndarray]*) – a dict object containing the legand of the first condition along with its mapped array
- **protein_two_repr** (*Dict[str, np.ndarray]*) – a dict object containing the legand of the second condition along with its mapped array

Returns: A plotly Figure containing the representation

Return type: Figure

`IPTK.Visualization.vizTools.`**`plotly_parent_protein_expression_in_tissue`** (expression_table: pandas.core.frame.DataFrame, ref_expression: pandas.core.frame.DataFrame, tissue_name: str, sampling_num: int = 10, def_value: float = - 1, ylabel: str = 'Normalized Expression') → plotly.graph_objs._figure.Figure

plot the parent protein expression in tissue relative a sampled collection of non-presented genes using plotly library.

Parameters:

- **expression_table** (*pd.DataFrame*) – The protein expression table which contains the expresion value for each parent proteins.
- **ref_expression** (*pd.DataFrame*) – The reference expression of the tissue under investigation.
- **tissue_name** (*str*) – The name of the tissue.
- **sampling_num** (*int, optional*) – the number of times to sample from the non-prsenter, defaults to 10
- **def_value** (*float, optional*) – The default value for proteins that could not be mapped to the expression database, defaults to -1
- **ylabel** (*The label on the y-axis, optional*) – [description], defaults to 'Normalized Expression'

Raises: **ValueError** – If the reference gene expression table is smaller than the number of parents

`IPTK.Visualization.vizTools.`**`plotly_peptide_length_dist`** (pep_length: List[int], x_label: str = 'Peptide Length', y_label: str = 'Counts', title: str = 'Peptide Length distribution')

visualize a histogram of the eluted peptide length using plotly library

Parameters:

- **pep_length** (*List[int]*) – a list of integer containing the peptides' lengths
- **x_label** (*str, optional*) – the label of the x-axis , defaults to 'Peptide Length'
- **y_label** (*str, optional*) – the label of the y-axis, defaults to 'Counts'
- **title** (*str, optional*) – the figure title, defaults to 'Peptide Length distribution'

`IPTK.Visualization.vizTools.`**`plotly_protein_coverage`** (mapped_protein: numpy.ndarray, prot_name: str = None) → plotly.graph_objs._figure.Figure

Plot the peptide coverage for a given protein

Parameters:

- **mapped_protein** (*np.ndarray*) – A NumPy array with shape of 1 by protein length or shape protein-length
- **prot_name** (*str, optional*) – The default protein name, defaults to None

Return type: Figure

Module contents

Module contents

Indices and tables

- **`genindex`**
- **`modindex`**

- **`search`**

Index

Python Module Index