metapredict Documentation

metapredict

Sep 07, 2021
# Contents

1 **Getting Started with metapredict**  
1.1 What is metapredict? ................................. 1  
1.2 How does metapredict work? ........................... 1  
1.3 What might the predicted confidence scores from AlphaFold2 be used for? .................. 2  
1.4 Why is metapredict useful? ........................... 2  
1.5 How to cite metapredict ............................... 2  
1.6 Installation ........................................... 2  
1.7 Known installation/execution issues ................. 3  
1.8 Testing .............................................. 3  
1.9 Example datasets .................................... 4  

2 **metapredict from the command-line**  
2.1 Predicting Disorder from Fasta Files .................. 5  
2.2 Predicting Disorder from a Sequence ................... 5  
2.3 Predicting AlphaFold2 Confidence Scores from a Fasta File .......................... 6  
2.4 Graphing Disorder from a Fasta file ................... 6  
2.5 Quick Graphing ...................................... 7  
2.6 Graphing using Uniprot ID ............................ 8  
2.7 Graphing Predicted AlphaFold2 pLDDT Scores from a fasta file .......................... 8  

3 **metapredict in Python**  
3.1 Predicting Disorder ................................... 11  
3.2 Predicting AlphaFold2 Confidence Scores ................. 12  
3.3 Graphing Disorder .................................... 12  
3.4 Graphing AlphaFold2 Confidence Scores ................. 14  
3.5 Calculating Percent Disorder: .......................... 14  
3.6 Predicting Disorder From a .fasta File: .................. 15  
3.7 Predicting AlphaFold2 confidence scores From a .fasta File .......................... 15  
3.8 Predict Disorder Using Uniprot ID ........................ 16  
3.9 Predicting AlphaFold2 Confidence Scores Using Uniprot ID .......................... 16  
3.10 Generating Disorder Graphs From a .fasta File: .................. 16  
3.11 Generating AlphaFold2 Confidence Score Graphs from fasta files .......................... 17  
3.12 Generating Graphs Using Uniprot ID ..................... 17  
3.13 Generating AlphaFold2 Confidence Score Graphs Using Uniprot ID .......................... 18  
3.14 Predicting Disorder Domains: .......................... 18  
3.15 Predicting Disorder Domains using a Uniprot ID: .................. 19
1.1 What is metapredict?

metapredict is a software tool to predict intrinsically disordered regions in protein sequences. It is provided as a downloadable Python tool that includes a Python application programming interface (API) and a set of command-line tools for working with FASTA files.

Our goal in building metapredict was to develop a robust, accurate, and high-performance predictor of intrinsic disorder that is also easy to install and use. As such, metapredict is implemented in Python and can be installed directly via pip (see below).

1.1.1 This already seems complicated…

As well as providing a set of high-performance software tools, metapredict is provided as a stand-alone webserver which can predict disorder profiles, scores, and contiguous IDRs for single sequences.

To access the webserver go to metapredict.net.

1.2 How does metapredict work?

metapredict is a bit different than your typical protein disorder predictor. Instead of predicting the percent chance that a residue within a sequence might be disordered, metapredict tries to predict the consensus disorder score for the residue. Consensus disorder reports on the fraction of independent disorder predictors that would predict a given residue as disordered.

metapredict is a deep-learning-based predictor trained on consensus disorder data from 8 different predictors, as pre-computed and provided by MobiDB. Functionally, this means each residue is assigned a score between 0 and 1 which reflects the confidence we have that the residue is disordered (or not). If the score was 0.5, this means half of the predictors predict that residue to be disordered. In this way, metapredict can help you quickly determine the likelihood that residues are disordered by giving you an approximation of what other predictors would predict (things got pretty ‘meta’ there, hence the name metapredict).
In addition, metapredict offers predicted confidence scores from AlphaFold2. These predicted scores use a bidirectional recurrent neural network (BRNN) trained on the per residue pLDDT (predicted IDDT-Ca) confidence scores generated by AlphaFold2 (AF2). The confidence scores (pLDDT) from the proteomes of Danio rerio, Candida albicans, Mus musculus, Escherichia coli, Drosophila melanogaster, Methanococcus jannaschii, Plasmodium falciparum, Caenorhabditis elegans, Dictyostelium discoideum, Trypanosoma cruzi, Saccharomyces cerevisiae, Schizosaccharomyces pombe, Rattus norvegicus, Homo sapiens, Arabidopsis thaliana, Zea mays, Leishmania infantum, Staphylococcus aureus, Glycine max, Oryza sativa were used to generate the BRNN. These confidence scores measure the local confidence that AlphaFold2 has in its predicted structure. The scores go from 0-100 where 0 represents low confidence and 100 represents high confidence. For more information, please see: Highly accurate protein structure prediction with AlphaFold https://doi.org/10.1038/s41586-021-03819-2. In describing these scores, the team states that regions with pLDDT scores of less than 50 should not be interpreted except as possible disordered regions.

1.3 What might the predicted confidence scores from AlphaFold2 be used for?

These scores can be used for many applications such as generating a quick preview of which regions of your protein of interest AF2 might be able to predict with high confidence, or which regions of your protein might be disordered. AF2 is not (strictly speaking) a disorder predictor, and the confidence scores are not directly representative of protein disorder. Therefore, any conclusions drawn with regards to disorder from predicted AF2 confidence scores should be interpreted with care, but they may be able to provide an additional metric to assess the likelihood that any given protein region may be disordered.

1.4 Why is metapredict useful?

Consensus disorder scores are really useful as they distribute the biases and uncertainty associated with any specific predictor. However, a drawback of consensus disorder databases (like MobiDB) is that they can only give you values of previously predicted protein sequences. metapredict provides a way around this, allowing arbitrary sequences to be analyzed!

The major advantages that metapredict offers over existing predictors is performance, ease of use, and ease of installation. Given metapredict uses a pre-trained bidirectional recurrent neural network, on hardware we’ve tested metapredict gives ~10,000 residues per second prediction power. This means that predicting disorder across entire proteomes is accessible in minutes - for example it takes ~20 minutes to predict disorder for every human protein in the reviewed human proteome (~23000 sequences). We provide metapredict as a simple-to-use Python library to integrate into existing Python workflows, and as a set of command-line tools for the stand-alone prediction of data from direct input or from FASTA files.

1.5 How to cite metapredict

If you use metapredict for your work, please cite the metapredict paper -


1.6 Installation

metapredict is available through GitHub or the Python Package Index (PyPI). To install through PyPI, run
$ pip install metapredict

To clone the GitHub repository and gain the ability to modify a local copy of the code, run

$ git clone https://github.com/idptools/metapredict.git
$ cd metapredict
$ pip install .

This will install metapredict locally. If you modify the source code in the local repository, be sure to re-install with pip.

1.7 Known installation/execution issues

Below we include documentation on known issues.

PyTorch currently ships with its own version of the OpenMP library (libiomp.dylib). Unfortunately when numpy is installed from conda (although not from pip) this leads to a collision because the conda-derived numpy library also includes a local copy of the libiomp5.dylib library. This leads to the following error message (included here for google-ability).

OMP: Error #15: Initializing libiomp5.dylib, but found libomp.dylib already initialized.
OMP: Hint This means that multiple copies of the OpenMP runtime have been linked into the program.
That is dangerous, since it can degrade performance or cause incorrect results. The best thing to do is to ensure that only a single OpenMP runtime is linked into the process, e.g. by avoiding static linking of the OpenMP runtime in any library. As an unsafe, unsupported, undocumented workaround you can set the environment variable KMP_DUPLICATE_LIB_OK=TRUE to allow the program to continue to execute, but that may cause crashes or silently produce incorrect results. For more information, please see http://www.intel.com/software/products/support/.

To avoid this error we make the executive decision to ignore this clash. This has largely not appeared to have any deleterious issues on performance or accuracy across the tests run. If you are uncomfortable with this then the code in metapredict/__init__.py can be edited with IGNORE_LIBOMP_ERROR set to False and metapredict re-installed from the source directory.

1.8 Testing

To see if your installation of metapredict is working properly, you can run the unit test included in the package by navigating to the metapredict/tests folder within the installation directory and running:

$ pytest -v
1.9 Example datasets

Example data that can be used with metapredict can be found in the metapredict/data folder on GitHub. The example data set is just a .fasta file containing 5 protein sequences.
2.1 Predicting Disorder from Fasta Files

The `metapredict-predict-disorder` command from the command line takes a .fasta file as input and returns disorder scores for the sequences in the FASTA file.

Once metapredict is installed, the user can run `metapredict-predict-disorder` from the command line:

```
$ metapredict-predict-disorder <Path to .fasta file>
```

Example:

```
$ metapredict-predict-disorder /Users/thisUser/Desktop/interestingProteins.fasta
```

Additional Usage:

Specifying where to save the output - If you would like to specify where to save the output, simply use the `-o` or `--output-file` flag and then specify the file path and file name. By default this command will save the output file as `disorder_scores.csv` to your current working directory. However, you can specify the file name in the output path.

Example:

```
$ metapredict-predict-disorder /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/thisUser/Desktop/disorder_predictions/my_disorder_predictions.csv
```

2.2 Predicting Disorder from a Sequence

`metapredict-quick-predict` is a command that will let you input a sequence and get disorder values immediately printed to the terminal. The only argument that can be input is the sequence.

Example:
2.3 Predicting AlphaFold2 Confidence Scores from a Fasta File

The metapredict-predict-pLDDT command from the command line takes a .fasta file as input and returns predicted AlphaFold2 pLDDT confidence scores for the sequences in the FASTA file.

$ metapredict-predict-pLDDT <Path to .fasta file>

Example

$ metapredict-predict-pLDDT /Users/thisUser/Desktop/interestingProteins.fasta

Additional Usage

Specify where to save the output - If you would like to specify where to save the output, simply use the -o or --output-file flag and then specify the file path. By default this command will save the output file as pLDDT_scores.csv to your current working directory. However, you can specify the file name in the output path.

Example

$ metapredict-predict-pLDDT /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/thisUser/Desktop/disorder_predictions/my_pLDDT_predictions.csv

2.4 Graphing Disorder from a Fasta file

The metapredict-graph-disorder command from the command line takes a .fasta file as input and returns a graph for every sequence within the .fasta file. Warning This will return a graph for every sequence in the FASTA file.

$ metapredict-graph-disorder <Path to .fasta file>

Example

$ metapredict-graph-disorder /Users/thisUser/Desktop/interestingProteins.fasta

If no output directory is specified, this function will make an output directory in the current working directory called disorder_out. This directory will hold all generated graphs.

Additional Usage

Adding AlphaFold2 Confidence Scores - To add predicted AlphaFold2 pLDDT confidence scores, simply use the -p or --pLDDT flag.

Example

$ metapredict-graph-disorder /Users/thisUser/Desktop/interestingProteins.fasta p

Specifying where to save the output - To specify where to save the output, simply use the -o or --output-directory flag.

Example
$ metapredict-graph-disorder /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/thisUser/Desktop/FolderForCoolPredictions

**Changing resolution of saved graphs** - By default, the output graphs have a DPI of 150. However, the user can change the DPI of the output (higher values have greater resolution but take up more space). To change the DPI simply add the flag `-D` or `--dpi` followed by the wanted DPI value.

**Example**

$ metapredict-graph-disorder /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/thisUser/Desktop/DisorderGraphsFolder/ -D 300

**Changing the file type** - By default the graphs will save as .png files. However, you can specify the file type by calling `--filetype` and then specifying the file type. Any matplotlib compatible file extension should work (for example, pdf).

**Example**


**Indexing file names** - If you would like to index the file names with a leading unique integer starting at 1, use the `--indexed-filenames` flag.

**Example**

$ metapredict-graph-disorder /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/thisUser/Desktop/DisorderGraphsFolder/ --indexed-filenames

**Changing the disorder threshold line on the graph** - If you would like to change the disorder threshold line plotted on the graph, use the `--disorder-threshold` flag followed by some value between 0 and 1. Default is 0.3.

**Example**

$ metapredict-graph-disorder /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/thisUser/Desktop/DisorderGraphsFolder/ --disorder-threshold 0.5

### 2.5 Quick Graphing

`metapredict-quick-graph` is a command that will let you input a sequence and get a plot of the disorder back immediately. You cannot input fasta files for this command. The command only takes three arguments, 1. the sequence 2. *optional* DPI `-D` or `--dpi` of the output graph which defaults to 150 DPI, and 3. *optional* to include predicted AlphaFold2 confidence scores, use the `p` or `--pLDDT` flag.

**Example:**

$ metapredict-quick-graph ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQQGIYNGTIAVAN

**Example:**

$ metapredict-quick-graph
  --ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQQGIYNGTIAVAN -p
2.6 Graphing using Uniprot ID

metapredict-uniprot is a command that will let you input any Uniprot ID and get a plot of the disorder for the corresponding protein. The default behavior is to have a plot automatically appear. Apart from the Uniprot ID which is required for this command, the command has four possible additional optional arguments, 1. To include predicted AlphaFold2 pLDDT confidence scores, use the -p or --pLDDT flag. DPI can be changed with the -D or --dpi flags, default is 150 DPI, 3. Using -o or --output-file will save the plot to a specified directory (default is current directory) - filenames and file extensions (pdf, jpg, png, etc) can be specified here. If there is no file name specified, it will save as the Uniprot ID and as a .png, 4. -t or --title will let you specify the title of the plot. By default the title will be Disorder for followed by the Uniprot ID.

Example:

$ metapredict-uniprot Q8RYC8

Example:

$ metapredict-uniprot Q8RYC8 -p

Example:

$ metapredict-uniprot Q8RYC8 -D 300

Example:

$ metapredict-uniprot Q8RYC8 -o /Users/ThisUser/Desktop/MyFolder/DisorderGraphs

Example:

$ metapredict-uniprot Q8RYC8 -o /Users/ThisUser/Desktop/MyFolder/DisorderGraphs/my_\_graph.png

Example:

$ metapredict-uniprot Q8RYC8 -t ARF19

2.7 Graphing Predicted AlphaFold2 pLDDT Scores from a fasta file

The metapredict-graph-pLDDT command from the command line takes a .fasta file as input and returns a graph of the predicted AlphaFold2 pLDDT confidence score for every sequence within the .fasta file. Warning This will return a graph for every sequence in the FASTA file.

Example:

$ metapredict-graph-pLDDT <Path to .fasta file>
If no output directory is specified, this function will make an output directory in the current working directory called *pLDDT_out*. This directory will hold all generated graphs.

**Additional Usage**

**Specifying where to save the output** - To specify where to save the output, simply use the `-o` or `--output-directory` flag.

**Example**

```
$ metapredict-graph-pLDDT /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
    →thisUser/Desktop/FolderForCoolPredictions
```

**Changing resolution of saved graphs** - By default, the output graphs have a DPI of 150. However, the user can change the DPI of the output (higher values have greater resolution but take up more space). To change the DPI simply add the flag `-D` or `--dpi` followed by the wanted DPI value.

**Example**

```
$ metapredict-graph-pLDDT /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
    →thisUser/Desktop/pLDDTGraphsFolder/ -D 300
```

**Changing the file type** - By default the graphs will save as .png files. However, you can specify the file type by calling `--filetype` and then specifying the file type. Any matplotlib compatible file extension should work (for example, pdf).

**Example**

```
$ metapredict-graph-pLDDT /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
    →thisUser/Desktop/pLDDTGraphsFolder/ --filetype pdf
```

**Indexing file names** - If you would like to index the file names with a leading unique integer starting at 1, use the `--indexed-filenames` flag.

**Example**

```
$ metapredict-graph-pLDDT /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
    →thisUser/Desktop/pLDDTGraphsFolder/ --indexed-filenames
```
In addition to using metapredict from the command line, you can also use it directly in Python.

First import metapredict -

```python
import metapredict as meta
```

Once metapredict is imported, you can work with individual sequences or .fasta files.

### 3.1 Predicting Disorder

The `predict_disorder()` function will return a list of predicted disorder consensus values for the residues of the input sequence. The input sequence should be a string made of valid amino acids. Running -

```python
meta.predict_disorder("DSSPEAPAEPKPVDWLYSYYFLTHHPADFLR")
```

would output -

```python
[1, 1, 1, 1, 0.957, 0.934, 0.964, 0.891, 0.863, 0.855, 0.793, 0.719, 0.665, 0.638, 0.576, 0.536, 0.496, 0.482, 0.306, 0.152, 0.096, 0.088, 0.049, 0.097, 0.235, 0.317, 0.341, 0.377, 0.388, 0.412, 0.46, 0.47, 0.545, 0.428]
```

**Additional Usage:**

**Disabling prediction value normalization** - By default, output prediction values are normalized between 0 and 1. However, some of the raw values from the predictor are slightly less than 0 or slightly greater than 1. The negative values are simply replaced with 0 and the values greater than 1 are replaced with 1 by default. However, the user can get the raw prediction values by specifying `normalized=False` as a second argument in `meta.predict_disorder`. There is not a very good reason to do this, and it is generally not recommended. However, we wanted to give users the maximum amount of flexibility when using metapredict, so we made it an option.

```python
meta.predict_disorder("DSSPEAPAEPKPVDWLYSYYFLTHHPADFLR", normalized=False)
```
3.2 Predicting AlphaFold2 Confidence Scores

The `predict_pLDDT` function will return a list of predicted AlphaFold2 pLDDT confidence scores for each residue of the input sequence. The input sequence should be a string. Running:

```python
meta.predict_pLDDT("DAPPTSQEHTQAEDKERD")
```

would output:

```python
[35.7925, 40.4579, 46.3753, 46.2976, 40.7481, 40.1676, 41.9618, 43.3977, 43.938, 41.8352, 44.0462, 44.5382, 46.3081, 49.2345, 46.0671]
```

3.3 Graphing Disorder

The `graph_disorder()` function will show a plot of the predicted disorder consensus values across the input amino acid sequence. Running:

```python
meta.graph_disorder("GHPGKQRNPGEHHSSRNVKRNWNNSPGPNEGRESQEEKTPPRRGQQSGESHNQDETNPSPDNHHEEEDDNAAHRGNDSPEAPEPPKDVPH...
```

would output:

![Graph of Predicted Protein Disorder](image)

Additional Usage

**Adding Predicted AlphaFold2 Confidence Scores** - To add predicted AlphaFold2 pLDDT confidence scores, simply specify `pLDDT_scores=True`.

**Example**

```python
seq = "GHPGKQRNPGEHHSSRNVKRNWNNSPGPNEGRESQEEKTPPRRGQQSGESHNQDETNPSPDNHHEEEDDNAAHRGNDSPEAPEPPKDVPH..."
meta.graph_disorder(seq, pLDDT_scores=True)
```

would output:
Changing title of generated graph - There are two parameters that the user can change for graph_disorder(). The first is the name of the title for the generated graph. The name by default is blank and the title of the graph is simply Predicted protein disorder. However, the title can be specified by specifying title = “my cool title” would result in a title of my cool title. Running -

```python
meta.graph_disorder(
    →"GHPGKQRNPGEHSSRNVTWNSPSGPNEGRESQEREERKTPRRGGQSGESHQDETNKPNPSDNHHEEKEADDNAHRGDSPAPEAPKDVPH...
    →", title = "MadeUpProtein")
```

would output -

![Graph example](image)

Changing the resolution of the generated graph - By default, the output graph has a DPI of 150. However, the user can change the DPI of the generated graph (higher values have greater resolution). To do so, simply specify DPI=Number where the number is an integer.

Example:

```python
meta.graph_disorder("DAPPTSQEHTQAEDKERD", DPI=300)
```

Changing the disorder threshold line - The disorder threshold line for graphs defaults to 0.3. However, if you want to change where the line designating the disorder cutoff is, simply specify disorder_threshold = Float where Float is some decimal value between 0 and 1.

Example

```python
meta.graph_disorder("DAPPTSQEHTQAEDKERD", disorder_threshold=0.5)
```

Adding shaded regions to the graph - If you would like to shade specific regions of your generated graph (perhaps shade the disordered regions), you can specify shaded_regions=[[list of regions]] where the list of regions is a list of lists that defines the regions to shade.

Example

```python
meta.graph_disorder("DAPPTSQEHTQAEDKERDAPPTSQEHTQAEDKERDDAPPTSQEHTQAEDKERD", shaded_regions=[[1, 20], [30, 40]])
```

In addition, you can specify the color of the shaded regions by specifying shaded_region_color. The default for this is red. You can specify any matplotlib color or a hex color string.

Example

```python
```
meta.graph_disorder("DAPPTSQHTQAEDKERDDAPPTSQHTQAEDKRDAPPTSQHTQAEDKERD", shaded_regions=[[1, 20], [30, 40]], shaded_region_color="blue")

Saving the graph - By default, the graph will automatically appear. However, you can also save the graph if you’d like. To do this, simply specify output_file = path_where_to_save/filename.file_extension. For example, output_file=/Users/thisUser/Desktop/cool_graphs/myCoolGraph.png. You can save the file with any valid matplotlib extension (.png, .pdf, etc.).

Example

meta.graph_disorder("DAPPTSQHTQAEDKER", output_file=/Users/thisUser/Desktop/cool__graphs/myCoolGraph.png)

3.4 Graphing AlphaFold2 Confidence Scores

The graph_pLDDT function will show a plot of the predicted AlphaFold2 pLDDT confidence scores across the input amino acid sequence.

Example

meta.graph_pLDDT("DAPTSQEHTQAEDKERDSKTHPQKKQPS")

This function has all of the same functionality as graph_disorder.

3.5 Calculating Percent Disorder:

The percent_disorder() function will return the percent of residues in a sequence that have predicted consensus disorder values of 0.3 or more. Running -

meta.percent_disorder("DSSPEAPAEPKDVPHDWLYSYVFTHHPADFLR")

would output -

82.39999

By default, this uses a cutoff predicted value of equal to or greater than 0.3 for a residue to be considered disordered.

Additional Usage:

Changing the cutoff value - If you want to be more strict in what you consider to be disordered for calculating percent disorder of an input sequence, you can simply specify the cutoff value by adding the argument cutoff=decimal where the decimal corresponds to the percent you would like to use as the cutoff (for example, 0.8 would be 80%).

Example:

meta.percent_disorder("DSSPEAPAEPKDVPHDWLYSYVFTHHPADFLR", cutoff = 0.8)

would output

29.4

The higher the cutoff value, the higher the value any given predicted residue must be greater than or equal to in order to be considered disordered when calculating the final percent disorder for the input sequence.
3.6 Predicting Disorder From a .fasta File:

By using the `predict_disorder_fasta()` function, you can predict disorder values for the amino acid sequences in a .fasta file. By default, this function will return a dictionary where the keys in the dictionary are the fasta headers and the values are the consensus disorder predictions of the amino acid sequence associated with each fasta header in the original .fasta file.

**Example:**

```python
meta.predict_disorder_fasta("file path to .fasta file/fileName.fasta")
```

An actual filepath would look something like:

```python
meta.predict_disorder_fasta("/Users/thisUser/Desktop/coolSequences.fasta")
```

**Additional Usage:**

**Save the output values** - By default the `predict_disorder_fasta` function will immediately return a dictionary. However, you can also save the output to a .csv file by specifying `output_file = "location you want to save the file to"`. When specifying the file path, you also want to specify the file name. The first cell of each row will contain a fasta header and the subsequent cells in that row will contain predicted consensus disorder values for the protein associated with the fasta header.

**Example:**

```python
meta.predict_disorder_fasta("file path to .fasta file/fileName.fasta", output_file="file path where the output .csv should be saved")
```

An actual filepath would look something like:

```python
meta.predict_disorder_fasta("/Users/thisUser/Desktop/coolSequences.fasta", output_file="/Users/thisUser/Desktop/cool_predictions.csv")
```

**Get raw prediction values** - By default, this function will output prediction values that are normalized between 0 and 1. However, some of the raw values from the predictor are slightly less than 0 or slightly greater than 1. The negative values are simply replaced with 0 and the values greater than 1 are replaced with 1 by default. If you want the raw values simply specify `normalized=False`. There is not a very good reason to do this, and it is generally not recommended. However, we wanted to give users the maximum amount of flexibility when using metapredict, so we made it an option.

**Example:**

```python
meta.predict_disorder_fasta("/Users/thisUser/Desktop/coolSequences.fasta", normalized=False)
```

3.7 Predicting AlphaFold2 confidence scores From a .fasta File

Just like with `predict_disorder_fasta`, you can use `predict_pLDDT_fasta` to get predicted AlphaFold2 pLDDT confidence scores from a fasta file. All the same functionality in `predict_disorder_fasta` is in `predict_pLDDT_fasta`.

**Example**

```python
meta.predict_pLDDT_fasta("/Users/thisUser/Desktop/coolSequences.fasta")
```
3.8 Predict Disorder Using Uniprot ID

By using the `predict_disorder_uniprot()` function, you can return predicted consensus disorder values for the amino acid sequence of a protein by specifying the Uniprot ID.

Example

```python
meta.predict_disorder_uniprot("Q8N6T3")
```

3.9 Predicting AlphaFold2 Confidence Scores Using Uniprot ID

By using the `predict_pLDDT_uniprot` function, you can generate predicted AlphaFold2 pLDDT confidence scores by inputting a Uniprot ID.

Example

```python
meta.predict_pLDDT_uniprot('P16892')
```

3.10 Generating Disorder Graphs From a .fasta File:

By using the `graph_disorder_fasta()` function, you can graph predicted consensus disorder values for the amino acid sequences in a .fasta file. The `graph_disorder_fasta` function takes a .fasta file as input and by default will return the graphs immediately. However, you can specify `output_dir=path_to_save_files` which result in a .png file saved to that directory for every sequence within the .fasta file. You cannot specify the output file name here! By default, the file name will be the first 14 characters of the FASTA header followed by the filetype as specified by filetype. If you wish for the files to include a unique leading number (i.e. X_rest_of_name where X starts at 1 and increments) then set `indexed_filenames = True`. This can be useful if you have sequences where the 1st 14 characters may be identical, which would otherwise overwrite an output file. By default this will return a single graph for every sequence in the FASTA file.

**WARNING** - This command will generate a graph for *every* sequence in the .fasta file. If you have 1,000 sequences in a .fasta file and you do not specify the `output_dir`, it will generate 1,000 graphs that you will have to close sequentially. Therefore, I recommend specifying the `output_dir` such that the output is saved to a dedicated folder.

Example:

```python
meta.graph_disorder_fasta("file path to .fasta file/fileName.fasta", output_dir="file...
->path of where to save output graphs")
```

An actual filepath would look something like:

```python
meta.graph_disorder_fasta("/Users/thisUser/Desktop/coolSequences.fasta", output_dir="/...
->Users/thisUser/Desktop/folderForGraphs")
```

Additional Usage

**Adding Predicted AlphaFold2 Confidence Scores** - To add predicted AlphaFold2 pLDDT confidence scores, simply specify `pLDDT_scores=True`.

Example

```python
meta.graph_disorder_fasta("/Users/thisUser/Desktop/coolSequences.fasta", pLDDT_...
->scores=True)
```
**Changing resolution of saved graphs** - By default, the output files have a DPI of 150. However, the user can change the DPI of the output files (higher values have greater resolution but take up more space). To change the DPI, specify `DPI=Number` where Number is an integer.

**Example:**

```python
meta.graph_disorder.fasta("/Users/thisUser/Desktop/coolSequences.fasta", DPI=300, output_dir="/Users/thisUser/Desktop/folderForGraphs")
```

**Changing the output File Type** - By default the output file is a .png. However, you can specify the output file type by using `output_filetype="file_type"` where file_type is some matplotlib compatible file type (such as .pdf).

**Example**

```python
meta.graph_disorder.fasta("/Users/thisUser/Desktop/coolSequences.fasta", output_dir="/Users/thisUser/Desktop/folderForGraphs", output_filetype = "pdf")
```

**Indexing generated files** - If you would like to index the file names with a leading unique integer starting at 1, set `indexed_filenames=True`.

**Example**

```python
meta.graph_disorder.fasta("/Users/thisUser/Desktop/coolSequences.fasta", output_dir="/Users/thisUser/Desktop/folderForGraphs", indexed_filenames=True)
```

### 3.11 Generating AlphaFold2 Confidence Score Graphs from fasta files

By using the `graph_pLDDT.fasta` function, you can graph predicted AlphaFold2 pLDDT confidence scores for the amino acid sequences in a .fasta file. This works the same as `graph_disorder.fasta` but instead returns graphs with just the predicted AlphaFold2 pLDDT scores.

```python
meta.graph_pLDDT.fasta("/Users/thisUser/Desktop/coolSequences.fasta", output_dir="/Users/thisUser/Desktop/folderForGraphs")
```

### 3.12 Generating Graphs Using Uniprot ID

By using the `graph_disorder_uniprot()` function, you can graph predicted consensus disorder values for the amino acid sequence of a protein by specifying the Uniprot ID.

**Example**

```python
meta.graph_disorder_uniprot("Q8N6T3")
```

This function carries all of the same functionality as `graph_disorder()` including specifying disorder_threshold, title of the graph, the DPI, and whether or not to save the output.

**Example**

```python
meta.graph_disorder_uniprot("Q8N6T3", disorder_threshold=0.5, title="my protein", DPI=300, output_file="/Users/thisUser/Desktop/my_cool_graph.png")
```
Additional usage

Adding Predicted AlphaFold2 Confidence Scores - To add predicted AlphaFold2 pLDDT confidence scores, simply specify `pLDDT_scores=True`.

Example

```python
meta.graph_disorder_uniprot("Q8N6T3", pLDDT_scores=True)
```

3.13 Generating AlphaFold2 Confidence Score Graphs Using Uniprot ID

Just like with disorder predictions, you can also get AlphaFold2 pLDDT confidence score graphs using the Uniprot ID. This will only display the pLDDT confidence scores and not the predicted disorder scores.

Example

```python
meta.graph_pLDDT_uniprot("Q8N6T3")
```

3.14 Predicting Disorder Domains:

The `predict_disorder_domains()` function takes in an amino acid function and returns a 4-position tuple with: 0. the raw disorder scores from 0 to 1 where 1 is the highest probability that a residue is disordered, 1. the smoothed disorder score used for boundary identification, 2. a list of elements where each element is a list where 0 and 1 define the IDR location and 2 gives the actual sequence, and 3. a list of elements where each element is a list where 0 and 1 define the folded domain location and 2 gives the actual sequence

```python
meta.predict_disorder_domains("MKAPSNGFLPSNEGEKKPINSQLWHACAGPLVSLPPVGSLVVYFPQGHSEQVAASMQKQTDFIPNYPNLPSKLICLLHS")
```

would output -

```plaintext
[(0.828, 0.891, 0.885, 0.859, 0.815, 0.795, 0.773, 0.677, 0.66, 0.736, 0.733, 0.708,...
 0.66, 0.631, 0.601, 0.564, 0.532, 0.508, 0.495, 0.458, 0.383, 0.373, 0.398, 0.36, 0.
 0.205, 0.158, 0.135, 0.091, 0.09, 0.102, 0.126, 0.129, 0.114, 0.106, 0.097, 0.085, 0.
 0.099, 0.114, 0.093, 0.119, 0.117, 0.043, 0.015, 0.05, 0.139, 0.172, 0.144, 0.121, 0.
 0.124, 0.128, 0.147, 0.173, 0.129, 0.152, 0.169, 0.2, 0.172, 0.22, 0.216, 0.25, 0.272,
 0.308, 0.248, 0.255, 0.301, 0.274, 0.264, 0.28, 0.25, 0.235, 0.221, 0.211, 0.235,
 0.185, 0.14, 0.168, 0.307, 0.509, 0.544, 0.402], array([0.87596856, 0.86139124, 0.
 0.84559624, 0.82968293, 0.81255466,
 0.79457882, 0.77575767, 0.75605988, 0.73557951, 0.71422703,
 0.69203382, 0.66900124, 0.63956894, 0.62124099, 0.60188696,
 0.57893168, 0.55214615, 0.52131925, 0.4859528, 0.44109689,
 0.3953789, 0.35264348, 0.31495776, 0.28 , 0.24661615,
 0.21469814, 0.18500621, 0.15963478, 0.13604845, 0.1172087,
 0.10798882, 0.1026882 , 0.09419503, 0.08462484, 0.08256398,
 0.08832671, 0.0908559 , 0.09263851, 0.09438758, 0.09309938,
 0.09102733, 0.09338137, 0.09665342, 0.10073913, 0.10392671,
 0.11010311, 0.11402981, 0.11898634, 0.12430683, 0.13169441,
 0.1381764 , 0.15245093, 0.16746957, 0.17518385, 0.18167578,
 0.18893043, 0.20013416, 0.21581491, 0.23015652, 0.2420559 ,
 0.25209814, 0.25817391, 0.26588944, 0.27456894, 0.27429068,
 0.26411925, 0.24452671, 0.23076894, 0.22834783, 0.21689842,
```

(continues on next page)
0.20887549, 0.20564427, 0.20856996, 0.21901779, 0.23835296, 0.26794071, 0.30914625, 0.36333478, 0.43187154, 0.51612174), [[0, 20, →'MKAPSNGFLPSSNEGEKKPI'], [[20, 80, →'NSQLWHACAGPLVSLPPVGLVYFPQGSEQVAASMQKQTDFIPNYPNLPSKICLLHS']]]

Additional Usage

Altering the disorder threshold - To alter the disorder threshold, simply set `disorder_threshold=my_value` where `my_value` is a float. The higher the threshold value, the more conservative metapredict will be for designating a region as disordered. Default = 0.42

Example

```python
meta.predict_disorder_domains("MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLV", disorder_threshold=0.3)
```

Altering minimum IDR size - The minimum IDR size will define the smallest possible region that could be considered an IDR. In other words, you will not be able to get back an IDR smaller than the defined size. Default is 12.

Example

```python
meta.predict_disorder_domains("MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLV", minimum_IDR_size=10)
```

Altering the minimum folded domain size - The minimum folded domain size defines where we expect the limit of small folded domains to be. *NOTE* this is not a hard limit and functions more to modulate the removal of large gaps. In other words, gaps less than this size are treated less strictly. *Note* that, in addition, gaps < 35 are evaluated with a threshold of 0.35 x disorder_threshold and gaps < 20 are evaluated with a threshold of 0.25 x disorder_threshold. These two lengthscales were decided based on the fact that coiled-coiled regions (which are IDRs in isolation) often show up with reduced apparent disorder within IDRs but can be as short as 20-30 residues. The folded_domain_threshold is used based on the idea that it allows a ‘shortest reasonable’ folded domain to be identified. Default=50.

Example

```python
meta.predict_disorder_domains("MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLV", minimum_folded_domain=60)
```

Altering gap_closure - The gap closure defines the largest gap that would be closed. Gaps here refer to a scenario in which you have two groups of disordered residues separated by a ‘gap’ of not disordered residues. In general large gap sizes will favour larger contiguous IDRs. It’s worth noting that gap_closure becomes relevant only when minimum_region_size becomes very small (i.e. < 5) because really gaps emerge when the smoothed disorder fit is “noisy”, but when smoothed gaps are increasingly rare. Default=10.

Example

```python
meta.predict_disorder_domains("MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLV", gap_closure=5)
```

3.15 Predicting Disorder Domains using a Uniprot ID:

In addition to inputting a sequence, you can predict disorder domains by inputting a Uniprot ID by using the `predict_disorder_domains_uniprot` function. This function has the exact same functionality as `predict_disorder_domains` except you can now input a Uniprot ID.

Example

```python
3.15. Predicting Disorder Domains using a Uniprot ID: 19
```
meta.predict_disorder_domains_uniprot('Q8N6T3')
4.1 Recommended usage

In general, we recommend using metapredict in Python by first importing metapredict as meta:

```python
import metapredict as meta
```

The `meta` module can be used to call all the user-facing functions. Documentation for these functions is included below.

4.2 metapredict functions

**metapredict.print_metapredict_network_version()**

Function that returns a string with the current trained network version used in disorder prediction. This is useful to know if updated versions of the network are provided, which will always accompany a version bump so prior versions of the code will always be available.

- **Returns** Returns a string in the format v<version information>
- **Return type** str

**metapredict.print_performance**(seg_len=500, num_segs=100, verbose=True)

Function that lets you test metapredict's performance on your local hardware.

- **Parameters**
  - `seq_len (int)`: Length of each random sequence to be tested. Default = 500.
  - `num_segs (int)`: Number of sequences to compute over. Default = 100.
  - `verbose (bool)`: Flag which, if true, means the function prints a summary when finished. If false simply returns an integer
- **Returns** Returns the nearest number of sequences-per-second metapredict is currently predicting. For ref, on a spring 2020 MBP this value was ~10,000 sequences per second.
Return type  int

metapredict.meta.predict_disorder_domains(sequence, disorder_threshold=0.42, minimum_IDR_size=12, minimum_folded_domain=50, gap_closure=10, normalized=True)

This function takes an amino acid sequence, a disorder score, and returns a 4-position tuple with the following information:

[0] - ‘Raw’ disorder score; i.e. disorder propensity as predicted by metapredict

[1] - Smoothed disorder score used to aid in domain boundary identification. This can be useful for understanding how IDRs/folded domains were identified, and will vary depending on minimum_region_size.

[2] - a list of elements, where each element is itself a list where position 0 and 1 define the IDR location and position 2 gives the actual IDR sequence

[3] - a list of elements, where each element is itself a list where position 0 and 1 define the folded domain location and position 2 gives the actual folded domain sequence.

Parameters

-sequence (str) – Amino acid sequence

-disorder_threshold (float) – Value that defines what ‘disordered’ is based on the metapredict disorder score. The higher the value the more stringent the cutoff. Default = 0.42

-minimum_IDR_size (int) – Defines the smallest possible IDR. This is a hard limit - i.e. we CANNOT get IDRs smaller than this. Default = 12.

-minimum_folded_domain (int) – Defines where we expect the limit of small folded domains to be. This is NOT a hard limit and functions to modulate the removal of large gaps (i.e. gaps less than this size are treated less strictly). Note that, in addition, gaps < 35 are evaluated with a threshold of 0.35*disorder_threshold and gaps < 20 are evaluated with a threshold of 0.25*disorder_threshold. These two length scales were decided based on the fact that coiled-coiled regions (which are IDRs in isolation) often show up with reduced apparent disorder within IDRs, and but can be as short as 20-30 residues. The folded_domain_threshold is used based on the idea that it allows a ‘shortest reasonable’ folded domain to be identified. Default=50.

-gap_closure (int) – Defines the largest gap that would be ‘closed’. Gaps here refer to a scenario in which you have two groups of disordered residues separated by a ‘gap’ of undisordered residues. In general large gap sizes will favour larger contiguous IDRs. It’s worth noting that gap_closure becomes relevant only when minimum_region_size becomes very small (i.e. < 5) because really gaps emerge when the smoothed disorder fit is “noisy”, but when smoothed gaps are increasingly rare. Default=10.

Returns

Always returns a list with 4 elements, as outlined below

[0] - List of floats - this is the ‘raw’ disorder score; i.e. disorder propensity as predicted by metapredict

[1] - List of floats - this is the smoothed disorder score used to aid in domain boundary identification. This can be useful for understanding how IDRs/folded domains were identified, and will vary depending on minimum_region_size.

[2] - a list of elements, where each element is itself a list where position 0 and 1 define the IDR location and position 2 gives the actual IDR sequence
[3] - a list of elements, where each element is itself a list where position 0 and 1 define the folded domain location and position 2 gives the actual folded domain sequence.

**Return type** list

```python
metapredict.meta.predict_disorder(sequence, normalized=True)
```

Function to return disorder of a single input sequence. Returns the predicted values as a list.

**Parameters**

- `sequence` *(str)* – Input amino acid sequence (as string) to be predicted.
- `normalized` *(bool)* – Flag which defines in the predictor should control and normalize such that all values fall between 0 and 1. The underlying learning model can, in fact output some negative values and some values greater than 1. Normalization controls for this. Default = True

**Returns** Returns a list of floats that corresponds to the per-residue disorder score.

**Return type** list

```python
metapredict.meta.graph_disorder(sequence, title='Predicted protein disorder', disorder_threshold=0.3, pLDDT_scores=False, shaded_regions=None, shaded_region_color='red', DPI=150, output_file=None)
```

Function to plot the disorder of an input sequence. Displays immediately.

**Parameters**

- `sequence` *(str)* – Input amino acid sequence (as string) to be predicted.
- `title` *(str)* – Sets the title of the generated figure. Default = “Predicted protein disorder”
- `disorder_threshold` *(float)* – Sets a threshold which draws a horizontal black line as a visual guide along the length of the figure. Must be a value between 0 and 1. Default = 0.3
- `pLDDT_scores` *(bool)* – Sets whether to include the predicted confidence scores from AlphaFold2
- `shaded_regions` *(list of lists)* – A list of lists, where sub-elements are of length 2 and contain start and end values for regions to be shaded. Assumes that sanity checking on positions has already been done. Default is None, but if there were specific regions you wanted to highlight this might, for example, look like shaded_regions=[[1,10],[40,50]], which would shade between 1 and 10 and then between 40 and 50. This can be useful to either highlight specific IDRs or specific folded domains
- `shaded_region_color` *(str)* – String that defines the color of the shaded region. The shaded region is always set with an alpha of 0.3 but the color can be any valid matplotlib color name or a hex color string (i.e. “#ff0000” is red).
- `DPI` *(int)* – Dots-per-inch. Defines the resolution of the generated figure. Passed to the dpi argument in matplotlib.pyplot.savefig().

**Returns** No return object, but, the graph is saved to disk or displayed locally.

**Return type** None
metapredict.meta.\texttt{percent\_disorder}(sequence, cutoff=0.3)

function to return the percent disorder for any given protein. By default, uses 0.3 as a cutoff (values greater than or equal to 0.3 will be considered disordered).

This function rounds to a single decimal place.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{sequence} (str) – Input amino acid sequence (as string) to be predicted.
  \item \texttt{disorder\_threshold} (float) – Sets a threshold which defines if a residue is considered disordered or not. Default = 0.3.
\end{itemize}

\textbf{Returns} Returns a floating point value between 0 and 100 that defines what percentage of the sequence is considered disordered.

\textbf{Return type} float

metapredict.meta.\texttt{predict\_disorder\_fasta}(filepath, output\_file=None, normalized=True, invalid\_sequence\_action=’convert’)

Function to read in a .fasta file from a specified filepath. Returns a dictionary of disorder values where the key is the fasta header and the values are the predicted disorder values.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{filepath} (str) – The path to where the .fasta file is located. The filepath should end in the file name. For example (on MacOS):filepath=’/Users/thisUser/Desktop/folder_of_seqs/interesting_proteins.fasta’
  \item \texttt{output\_file} (str) – By default, a dictionary of predicted values is returned immediately. However, you can specify an output filename and path and a .csv file will be saved. This should include any file extensions. Default = None.
  \item \texttt{normalized} (bool) – Flag which defines in the predictor should control and normalize such that all values fall between 0 and 1. The underlying learning model can, in fact output some negative values and some values greater than 1. Normalization controls for this. Default = True
  \item \texttt{invalid\_sequence\_action} (str) – Tells the function how to deal with sequences that lack standard amino acids. Default is convert, which as the name implies converts via standard rules. See https://protfasta.readthedocs.io/en/latest/read\_fasta.html for more information.
\end{itemize}

\textbf{Returns} If output\_file is set to None (as default) then this fiction returns a dictionary of sequence ID to disorder vector. If output\_file is set to a filename then a .csv file will instead be written and no return data will be provided.

\textbf{Return type} dict or None

metapredict.meta.\texttt{graph\_disorder\_fasta}(filepath, pLDDT\_scores=False, disorder\_threshold=0.3, DPI=150, output\_dir=None, output\_filetype=’png’, invalid\_sequence\_action=’convert’, indexed\_filenames=False)

Function to make graphs of predicted disorder from the sequences in a specified .fasta file. By default will save the generated graphs to the location output\_path specified in filepath.

\textbf{WARNING}: It is unadvisable to not include an output directory if you are reading in a .fasta file with many sequences! This is because each graph must be closed individually before the next will appear. Therefore, you will spend a bunch of time closing each graph.

\textbf{NB}: You cannot specify the output file name here! By default, the file name will be the first 14 characters of the FASTA header followed by the filetype as specified by filetype. If you wish for the files to include a unique leading number (i.e. X\_rest\_of\_name where X starts at 1 and increments) then set indexed\_filenames to True.
This can be useful if you have sequences where the 1st 14 characters may be identical, which would otherwise overwrite an output file.

Parameters

• **filepath** *(str)* – The path to where the .fasta file is located. The filepath should end in the file name. For example (on MacOS): `filepath="/Users/thisUser/Desktop/folder_of_seqs/interesting_proteins.fasta"`

• **pLDDT_scores** *(Bool)* – Sets whether to include the predicted pLDDT scores from AlphaFold2

• **disorder_threshold** *(float)* – Sets a threshold which draws a horizontal black line as a visual guide along the length of the figure. Must be a value between 0 and 1.

• **DPI** *(int)* – Dots-per-inch. Defines the resolution of the generated figure. Passed to the dpi argument in `matplotlib.pyplot.savefig()`.

• **output_dir** *(str)* – If provided, the output_dir variable defines the directory where file should be saved. This should be a writeable filepath. Default is None. Output files are saved with filename as first 14 chars of fasta header (minus bad characters) plus the appropriate file extension, as defined by filetype.

• **output_filetype** *(str)* – String that defines the output filetype to be used. Must be one of pdf, png, jpg.

• **invalid_sequence_action** *(str)* – Tells the function how to deal with sequences that lack standard amino acids. Default is convert, which as the name implies converts via standard rules. See [https://protfasta.readthedocs.io/en/latest/read_fasta.html](https://protfasta.readthedocs.io/en/latest/read_fasta.html) for more information.

• **indexed_filenames** *(bool)* – Bool which, if set to true, means filenames start with an unique integer.

Returns No return object, but, the graph is saved to disk or displayed locally.

Return type None

```python
metapredict.meta.predict_disorder_uniprot(uniprot_id, normalized=True)
```

Function to return disorder of a single input sequence. Uses a Uniprot ID to get the sequence.

Parameters

• **uniprot_ID** *(str)* – The uniprot ID of the sequence to predict

• **no_ID** *(str)* – The uniprot ID of the sequence to predict

Returns No return object, but, the graph is saved to disk or displayed locally.

Return type None

```python
metapredict.meta.graph_disorder_uniprot(uniprot_id, title='Predicted protein disorder', pLDDT_scores=False, disorder_threshold=0.3, shaded_regions=None, shaded_region_color='red', DPI=150, output_file=None)
```

Function to plot the disorder of an input sequence. Displays immediately.

Parameters

• **sequence** *(str)* – Input amino acid sequence (as string) to be predicted.

• **title** *(str)* – Sets the title of the generated figure. Default = “Predicted protein disorder”

• **pLDDT_scores** *(Bool)* – Sets whether to include the predicted pLDDT scores from AlphaFold2

4.2. metapredict functions 25
• **disorder_threshold** (*float*) – Sets a threshold which draws a horizontal black line as a visual guide along the length of the figure. Must be a value between 0 and 1.

• **shaded_regions** (*list of lists*) – A list of lists, where sub-elements are of length 2 and contain start and end values for regions to be shaded. Assumes that sanity checking on positions has already been done. Default is None, but if there were specific regions you wanted to highlight this might, for example, look like shaded_regions=[[[1,10],[40,50]]], which would shade between 1 and 10 and then between 40 and 50. This can be useful to either highlight specific IDRs or specific folded domains

• **shaded_region_color** (*str*) – String that defines the color of the shaded region. The shaded region is always set with an alpha of 0.3 but the color can be any valid matplotlib color name or a hex color string (i.e. “#ff0000” is red).

• **DPI** (*int*) – Dots-per-inch. Defines the resolution of the generated figure. Passed to the dpi argument in matplotlib.pyplot.savefig().

• **output_file** (*str*) – If provided, the output_file variable defines the location and type of the file to be saved. This should be a file location and filename with a valid matplotlib extension (such as .png, or .pdf) and, if provided, this value is passed directly to the matplotlib.pyplot.savefig() function as the fname parameter. Default = None.

**Returns** No return object, but, the graph is saved to disk or displayed locally.

**Return type** None

```python
def predict_disorder_domains_uniprot(uniprot_id, disorder_threshold=0.42, minimum_IDR_size=12, minimum_folded_domain=50, gap_closure=10, normalized=True):
```

This function takes an amino acid sequence, a disorder score, and returns a 4-position tuple with the following information:

[0] - ‘Raw’ disorder score; i.e. disorder propensity as predicted by metapredict

[1] - Smoothed disorder score used to aid in domain boundary identification. This can be useful for understanding how IDRs/folded domains were identified, and will vary depending on minimum_region_size.

[2] - a list of elements, where each element is itself a list where position 0 and 1 define the IDR location and position 2 gives the actual IDR sequence

[3] - a list of elements, where each element is itself a list where position 0 and 1 define the folded domain location and position 2 gives the actual folded domain sequence.

**Parameters**

• **uniprot_ID** (*String*) – The uniprot ID of the sequence to predict

• **sequence** (*str*) – Amino acid sequence

• **disorder_threshold** (*float*) – Value that defines what ‘disordered’ is based on the metapredict disorder score. The higher the value the more stringent the cutoff. Default = 0.42

• **minimum_IDR_size** (*int*) – Defines the smallest possible IDR. This is a hard limit - i.e. we CANNOT get IDRs smaller than this. Default = 12.

• **minimum_folded_domain** (*int*) – Defines where we expect the limit of small folded domains to be. This is NOT a hard limit and functions to modulate the removal of large gaps (i.e. gaps less than this size are treated less strictly). Note that, in addition, gaps < 35 are evaluated with a threshold of 0.35*disorder_threshold and gaps < 20 are evaluated
with a threshold of 0.25*disorder_threshold. These two lengthscales were decided based on the fact that coiled-coiled regions (which are IDRs in isolation) often show up with reduced apparent disorder within IDRs, and but can be as short as 20-30 residues. The folded_domain_threshold is used based on the idea that it allows a ‘shortest reasonable’ folded domain to be identified. Default=50.

- **gap_closure** *(int)* – Defines the largest gap that would be ‘closed’. Gaps here refer to a scenario in which you have two groups of disordered residues separated by a ‘gap’ of unordered residues. In general large gap sizes will favour larger contiguous IDRs. It’s worth noting that gap_closure becomes relevant only when minimum_region_size becomes very small (i.e. < 5) because really gaps emerge when the smoothed disorder fit is “noisy”, but when smoothed gaps are increasingly rare. Default=10.

**Returns**

Always returns a list with 4 elements, as outlined below

[0] - List of floats - this is the ‘raw’ disorder score; i.e. disorder propensity as predicted by metapredict

[1] - List of floats - this is the smoothed disorder score used to aid in domain boundary identification. This can be useful for understanding how IDRs/folded domains were identified, and will vary depending on minimum_region_size.

[2] - a list of elements, where each element is itself a list where position 0 and 1 define the IDR location and position 2 gives the actual IDR sequence

[3] - a list of elements, where each element is itself a list where position 0 and 1 define the folded domain location and position 2 gives the actual folded domain sequence.

**Return type** list

`metapredict.meta.predict_disorder_domains_from_external_scores` *(disorder, sequence=None, disorder_threshold=0.5, minimum_IDR_size=12, minimum_folded_domain=50, gap_closure=10, override_folded_domain_minsize=False)*

This function takes in disorder scores generated from another predictor and applies the same domain-decomposition algorithm as predict_disorder_domains() does to extract contiguous IDRs. For example, if one were to predict disorder using the (excellent) ODINPred, download the resulting scores, and read the scores into a list, that list could be passed as the $disorder argument to this function.

Note that the settings used here may be inapplicable to another disorder predictor, so you may need to play around with the parameters including disorder_threshold, minimum_IDR_size, minimum_folded_domain and gap_closure.

the following information:

[0] - Smoothed disorder score used to aid in domain boundary identification. This can be useful for understanding how IDRs/folded domains were identified, and will vary depending on the settings provided.

[1] - a list of elements, where each element defines the start and end position of each IDR

[2] - a list of elements, where each element defines the start and end position of each folded region
Parameters

- **disorder** *(list)* – A list of per-residue disorder scores.
- **sequence** *(str)* – An optional argument which, if provided, is assumed to reflect the the amino acid sequence from which the disorder scores were computed. Note if these do not match one another in length then the function raises an exception. Default = None
- **disorder_threshold** *(float)* – Value that defines what ‘disordered’ is based on the input predictor score. The higher the value the more stringent the cutoff. Default = 0.5.
- **minimum_IDR_size** *(int)* – Defines the smallest possible IDR. This is a hard limit - i.e. we CANNOT get IDRs smaller than this. Default = 12.
- **minimum_folded_domain** *(int)* – Defines where we expect the limit of small folded domains to be. This is NOT a hard limit and functions to modulate the removal of large gaps (i.e. gaps less than this size are treated less strictly). Note that, in addition, gaps < 35 are evaluated with a threshold of 0.35*disorder_threshold and gaps < 20 are evaluated with a threshold of 0.25*disorder_threshold. These two length scales were decided based on the fact that coiled-coiled regions (which are IDRs in isolation) often show up with reduced apparent disorder within IDRs, and but can be as short as 20-30 residues. The folded_domain_threshold is used based on the idea that it allows a ‘shortest reasonable’ folded domain to be identified. Default=50.
- **gap_closure** *(int)* – Defines the largest gap that would be ‘closed’. Gaps here refer to a scenario in which you have two groups of disordered residues separated by a ‘gap’ of un-disordered residues. In general large gap sizes will favour larger contiguous IDRs. It’s worth noting that gap_closure becomes relevant only when minimum_region_size becomes very small (i.e. < 5) because really gaps emerge when the smoothed disorder fit is “noisy”, but when smoothed gaps are increasingly rare. Default=10.
- **override_folded_domain_minsize** *(bool)* – By default this function includes a fail-safe check that assumes folded domains really shouldn’t be less than 35 or 20 residues. However, for some approaches we may wish to over-ride these thresholds to match the passed minimum_folded_domain value. If this flag is set to True this override occurs. This is generally not recommended unless you expect there to be well-defined sharp boundaries which could define small (20-30) residue folded domains. This is not provided as an option in the normal predict_disorder_domains for metapredict. Default = False.

Returns

Always returns a list with three elements, as outlined below.

0 - Smoothed disorder score used to aid in domain boundary identification. This can be useful for understanding how IDRs/folded domains were identified, and will vary depending on the settings provided

1 - a list of elements, where each element defines the start and end position of each IDR. If a sequence was provided the third element in each sub-element is the IDR sequence. If no sequence was provided, then each sub-element is simply len=2.

2 - a list of elements, where each element defines the start and end position of each folded region. If a sequence was provided the third element in each sub-element is the folded domain sequence. If no sequence was provided, then each sub-element is simply len=2.

Return type  list

metapredict.meta.graph_pLDDT_uniprot(uniprot_id, title='Predicted AF2 pLDDT Scores', shaded_regions=None, shaded_region_color='red', DPI=150, output_file=None)

Function to plot the disorder of an input sequence. Displays immediately.
Parameters

- **sequence** *(str)* – Input amino acid sequence (as string) to be predicted.
- **title** *(str)* – Sets the title of the generated figure. Default = “Predicted protein disorder”
- **shaded_regions** *(list of lists)* – A list of lists, where sub-elements are of length 2 and contain start and end values for regions to be shaded. Assumes that sanity checking on positions has already been done. Default is None, but if there were specific regions you wanted to highlight this might, for example, look like shaded_regions=[[1,10],[40,50]], which would shade between 1 and 10 and then between 40 and 50. This can be useful to either highlight specific IDRs or specific folded domains
- **shaded_region_color** *(str)* – String that defines the color of the shaded region. The shaded region is always set with an alpha of 0.3 but the color can be any valid matplotlib color name or a hex color string (i.e. “#ff0000” is red).
- **DPI** *(int)* – Dots-per-inch. Defines the resolution of the generated figure. Passed to the dpi argument in matplotlib.pyplot.savefig().
- **output_file** *(str)* – If provided, the output_file variable defines the location and type of the file to be saved. This should be a file location and filename with a valid matplotlib extension (such as .png, or .pdf) and, if provided, this value is passed directly to the matplotlib.pyplot.savefig() function as the fname parameter. Default = None.

Returns
No return object, but, the graph is saved to disk or displayed locally.

Return type
None

metapredict.meta.predict_pLDDT_uniprot *(uniprot_id)*
Function to return pLDDT score of a single input sequence. Uses a Uniprot ID to get the sequence.

Parameters
- **uniprot_ID** *(str)* – The uniprot ID of the sequence to predict

Returns
No return object, but, the graph is saved to disk or displayed locally.

Return type
None

metapredict.meta.graph_pLDDT_fasta *(filepath, DPI=150, output_dir=None, output_filetype='png', invalid_sequence_action='convert', indexed_filenames=False)*
Function to make graphs of predicted pLDDT from the sequences in a specified .fasta file. By default will save the generated graphs to the location output_path specified in filepath.

WARNING: It is unadvisable to not include an output directory if you are reading in a .fasta file with many sequences! This is because each graph must be closed individually before the next will appear. Therefore, you will spend a bunch of time closing each graph.

NB: You cannot specify the output file name here! By default, the file name will be the first 14 characters of the FASTA header followed by the filetype as specified by filetype. If you wish for the files to include a unique leading number (i.e. X_rest_of_name where X starts at 1 and increments) then set indexed_filenames to True. This can be useful if you have sequences where the 1st 14 characters may be identical, which would otherwise overwrite an output file.

Parameters
- **filepath** *(str)* – The path to where the .fasta file is located. The filepath should end in the file name. For example (on MacOS):filepath="/Users/thisUser/Desktop/folder_of_seq/interesting_proteins.fasta"
- **DPI** *(int)* – Dots-per-inch. Defines the resolution of the generated figure. Passed to the dpi argument in matplotlib.pyplot.savefig().
**output_dir** (str) – If provided, the output_dir variable defines the directory where file should be saved. This should be a writeable filepath. Default is None. Output files are saved with filename as first 14 chars of fasta header (minus bad characters) plus the appropriate file extension, as defined by filetype.

**output_filetype** (str) – String that defines the output filetype to be used. Must be one of pdf, png, jpg.

**invalid_sequence_action** (str) – Tells the function how to deal with sequences that lack standard amino acids. Default is convert, which as the name implies converts via standard rules. See https://protfasta.readthedocs.io/en/latest/read_fasta.html for more information.

**indexed_filenames** (bool) – Bool which, if set to true, means filenames start with an unique integer.

Returns No return object, but, the graph is saved to disk or displayed locally.

Return type None

```
metapredict.meta.predict_pLDDT_fasta(filepath, output_file=None, invalid_sequence_action='convert')
```

Function to read in a .fasta file from a specified filepath. Returns a dictionary of pLDDT values where the key is the fasta header and the values are the predicted pLDDT values.

Parameters

- **filepath** (str) – The path to where the .fasta file is located. The filepath should end in the file name. For example (on MacOS): filepath="/Users/thisUser/Desktop/folder_of_seqs/interesting_proteins.fasta"

- **output_file** (str) – By default, a dictionary of predicted values is returned immediately. However, you can specify an output filename and path and a .csv file will be saved. This should include any file extensions. Default = None.

- **invalid_sequence_action** (str) – Tells the function how to deal with sequences that lack standard amino acids. Default is convert, which as the name implies converts via standard rules. See https://protfasta.readthedocs.io/en/latest/read_fasta.html for more information.

Returns If output_file is set to None (as default) then this function returns a dictionary of sequence ID to pLDDT vector. If output_file is set to a filename then a .csv file will instead be written and no return data will be provided.

Return type dict or None

```
metapredict.meta.graph_pLDDT(sequence, title='Predicted AF2 pLDDT Confidence Score', pLDDT_scores=True, disorder_scores=False, shaded_regions=None, shaded_region_color='red', DPI=150, output_file=None)
```

Function to plot the AF2 pLDDT scores of an input sequence. Displays immediately.

Parameters

- **sequence** (str) – Input amino acid sequence (as string) to be predicted.

- **title** (str) – Sets the title of the generated figure. Default = “Predicted protein disorder”

- **pLDDT_scores** (Bool) – Sets whether to include the predicted confidence scores from AlphaFold2

- **disorder_scores** (Bool) – Whether to include disorder scores. Can set to False if you just want the AF2 confidence scores.

- **shaded_regions** (list of lists) – A list of lists, where sub-elements are of length 2 and contain start and end values for regions to be shaded. Assumes that sanity checking on positions...
has already been done. Default is None, but if there were specific regions you wanted to highlight this might, for example, look like shaded_regions=[[1,10],[40,50]], which would shade between 1 and 10 and then between 40 and 50. This can be useful to either highlight specific IDRs or specific folded domains.

- **shaded_region_color** *(str)* – String that defines the color of the shaded region. The shaded region is always set with an alpha of 0.3 but the color can be any valid matplotlib color name or a hex color string (i.e. “#ff0000” is red).

- **DPI** *(int)* – Dots-per-inch. Defines the resolution of the generated figure. Passed to the dpi argument in matplotlib.pyplot.savefig().

- **output_file** *(str)* – If provided, the output_file variable defines the location and type of the file to be saved. This should be a file location and filename with a valid matplotlib extension (such as .png, or .pdf) and, if provided, this value is passed directly to the matplotlib.pyplot.savefig() function as the fname parameter. Default = None.

Returns  No return object, but, the graph is saved to disk or displayed locally.

Return type  None

**metapredict.meta.predict_pLDDT (sequence)**

Function to return predicted pLDDT scores from AlphaFold2 for an input sequence.

Parameters  sequence *(str)* – Input amino acid sequence (as string) to be predicted.

Returns  Returns a list of floats that corresponds to the per-residue pLDDT score.

Return type  list
IDP-Parrot, created by Dan Griffith, was used to generate the network used for metapredict. See https://pypi.org/project/idptools-parrot/ for some very cool machine learning stuff from Dan.

In addition to using Dan Griffith’s tool for creating metapredict, the code for brnn_architecture.py and encode_sequence.py was written by Dan (originally for idptools-Parrot).

We would also like to thank the team at MobiDB for creating the database that was used to train this predictor. Check out their awesome stuff at https://mobidb.bio.unipd.it

We would like to thank the DeepMind team for developing AlphaFold.

Project based on the [Computational Molecular Science Python Cookiecutter](https://github.com/molssi/cookiecutter-cms) version 1.3.
HELP! Metapredict isn’t working!

6.1 Python Version Issues

I have received occasional feedback that metapredict is not working for a user. A common problem is that the user is using a different version of Python than metapredict was made on. Metapredict was made using Python version 3.7, but works on 3.8 as well. I recommend using one of these versions to avoid problems (I haven’t done extensive testing using other versions of Python, so if you’re not using 3.7 or 3.8, do so at your own risk). A convenient workaround is to use a conda environment that has Python 3.7 set as the default version of Python. For more info on conda, please see https://docs.conda.io/projects/conda/en/latest/index.html

Once you have conda installed, simply use the command

```
conda create --name my_env python=3.7
conda activate my_env
```

and once activate install metapredict from PyPI

```
pip install metapredict
```

You can, then use metapredict from within this conda environment. In all our testing, this setup leads to a working version of metapredict. However, in principle metapredict should work automatically when installed from pip.

6.2 Reporting Issues

If you are having other problems, please report them to the issues section on the metapredict Github page at https://github.com/idptools/metapredict/issues
7.1 About

This section is a log of recent changes with metapredict. My hope is that as I change things, this section can help you figure out why a change was made and if it will break any of your current work flows. The first major changes were made for the 0.56 release, so tracking will start there.

7.2 V1.51

Changes: Updated to require V1.0 of alphaPredict for pLDDT scores. This improves accuracy from over 9% per residue to about 8% per residue for pLDDT score predictions. Documentation was updated for this change.

7.3 V1.5

Changes: Fixed bug causing some functions to fail when getting sequences from Uniprot. Added information on citing metapredict because the final publication went live.

7.4 V1.4

Change: For clarity, previous functions that used the term ‘confidence’ such as graph_confidence_uniprot() were changed to use the term pLDDT rather than confidence. This is to clarify that the confidence scores are AlphaFold2 pLDDT confidence scores and not scores to reflect the confidence that the user should have in the metapredict disorder prediction. For command-line usage where confidence scores are optional (such as metapredict-graph-disorder), when a -c or --confidence flag used to be used, now a -p or --pLDDT flag is used to graph confidence scores. This is similarly reflected in Python where now you must use pLDDT_scores=True instead of confidence_scores=True.
7.5 V1.3
Change: Added functionality to generate predicted AlphaFold2 confidence scores. Can get scores or generate graphs from Python or command-line. Can also generate graphs with both predicted disorder and predicted confidence scores. Also added functionality to predict disorder domains using scores from a different disorder predictor.

7.6 V1.2
Change: Major update. Changed some basic functionality. Made it such that you don’t need to specify to save (for disorder prediction values or graphs). Rather, if a file path is specified, the files will be saved. Updated graphing functionality to allow for specifying the disorder cutoff line and to allow users to highlight various regions of the graph. Changed import such that you can now just use import metapredict as meta in Python (as opposed to import metapredict and then from metapredict import meta). Lots of backend changes to make metapredict more stable. Added additional testing. Updated documentation. Standardized file reading/writing. Made it so user can specify file type of saved graphs. Added backend meta_tools.py to handle the busywork. Changed version numbering for networks. Updated code to avoid OMPLIB issue (known bug in previous versions). Updated all command-line tools to match backend code.

7.7 V1.1
Change: Fixed some bugs.

7.8 V1.0
Change: Added functionality to generate graphs using a Uniprot ID as the input. Added functionality to predict disorder domains.

7.9 V.061
Change: Added functionality to predict or graph a disordered sequence from the command line by directly inputting the sequence. This can only do one sequence at a time and does not save the disorder values or graph. It is meant to provide a very quick and easy way to check something out.

7.10 V.060
Change: Added functionality to specify the horizontal lines that appear across the graphs rather than only having the option of having the dashed lines appear at intervals of 0.2. This functionality is in both Python and the command line.

7.11 V0.58
Change: Updated the network with a newly trained network (using the same dataset as the original) that is slightly more accurate.
Reason: I am always trying to find ways to make metapredict more accurate. When I manage to make the predictor better, I will update it.

7.12 V0.57

Change: Bug fix that could result in prediction values to six decimal places in some scenarios

Change: Changed titles for graphs generated by `metapredict-graph-disorder` to be 14 characters instead of 10. This is reflected in the title graph and the saved files.

Reason: The 10 character save file was occasionally the same for multiple proteins. This resulted in the inability to discern which protein corresponded to which graph and could result in overwriting previously generated graphs. The 14 characters should be long enough to keep unique names for all proteins being analyzed.

Change: Fixed bug that could result in crashing due to short fasta headers.

7.13 V0.56

Change: Number of decimals in predictions was reduced from 6 to 3. Reason: It is not necessary to have accuracy out to 6 decimal places.

Change: Added functionality to use `.` to specify current directory from command line. Reason: Improve functionality.

Change: -DPI flag changed to -dpi in command line graphing function Reason: It was annoying to have to do all caps for this flag.

Change: The `predict-disorder` command is now `metapredict-predict-disorder` and the `graph-disorder` command is now `metapredict-graph-disorder` Reason: This will help users be able to use auto complete functionality from the command line using tab to pull up the graph or predict disorder commands while only having to remember metapredict.

Change: The output for `.csv` files will now have a comma space between each value instead of just a comma. Reason: Improve readability.
How to cite metapredict

If you use metapredict for your work, please cite the metapredict paper -

## Python Module Index

<table>
<thead>
<tr>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>metapredict, 21</td>
</tr>
<tr>
<td>metapredict.meta, 22</td>
</tr>
</tbody>
</table>
Index

G

graph_disorder() (in module metapredict.meta), 23
graph_disorder_fasta() (in module metapredict.meta), 24
graph_disorder_uniprot() (in module metapredict.meta), 25
graph_pLDDT() (in module metapredict.meta), 30
graph_pLDDT_fasta() (in module metapredict.meta), 29
graph_pLDDT_uniprot() (in module metapredict.meta), 28

M

metapredict (module), 21
metapredict.meta (module), 22

P

percent_disorder() (in module metapredict.meta), 23
predict_disorder() (in module metapredict.meta), 23
predict_disorder_domains() (in module metapredict.meta), 22
predict_disorderDomains_fromExternal_scores() (in module metapredict.meta), 27
predict_disorder_domains_uniprot() (in module metapredict.meta), 26
predict_disorder_fasta() (in module metapredict.meta), 24
predict_disorder_uniprot() (in module metapredict.meta), 25
predict_pLDDT() (in module metapredict.meta), 31
predict_pLDDT_fasta() (in module metapredict.meta), 30
predict_pLDDT_uniprot() (in module metapredict.meta), 29
print_metapredict_network_version() (in module metapredict), 21
print_performance() (in module metapredict), 21