metapredict Documentation

metapredict

Mar 30, 2024

CONTENTS:

1 Getting Started with metapredict . . . 1
- 1.1 What is metapredict? . . . 1
- 1.2 metapredict updates and news . . . 1
- 1.3 Installation . . . 2
- 1.4 About . . . 2
- 1.5 Why is metapredict useful? . . . 4
- 1.6 How to cite . . . 4
- 1.7 Known installation issues . . . 4
- 1.8 Testing . . . 5
- 1.9 Example datasets . . . 5

2 metapredict from the command-line . . . 7
- 2.1 Using the original metapredict network . . . 7
- 2.2 Predicting disorder scores from fasta files . . . 7
- 2.3 Predicting IDRs from a fasta file . . . 8
- 2.4 Predicting disorder scores from sequence . . . 8
- 2.5 Predicting AlphaFold2 confidence scores from a fasta file . . . 9
- 2.6 Plotting disorder profiles from a fasta file . . . 9
- 2.7 Quick graphing of disorder scores . . . 10
- 2.8 Graphing disorder scores using UniProt ID . . . 11
- 2.9 Graphing disorder using the common name of a protein . . . 12
- 2.10 Graphing predicted AlphaFold2 pLDDT scores from a fasta file . . . 13

3 metapredict in Python . . . 15
- 3.1 Important updates . . . 15
- 3.2 Predicting Disorder . . . 16
- 3.3 Predicting AlphaFold2 Confidence Scores . . . 16
- 3.4 Predicting Disorder Domains: . . . 17
- 3.5 Calculating Percent Disorder: . . . 19
- 3.6 Graphing Disorder . . . 20
- 3.7 Graphing AlphaFold2 Confidence Scores . . . 22
- 3.8 Predicting Disorder From a .fasta File: . . . 23
- 3.9 Predicting AlphaFold2 confidence scores From a .fasta File . . . 24
- 3.10 Predict Disorder Using Uniprot ID . . . 24
- 3.11 Predicting AlphaFold2 Confidence Scores Using Uniprot ID . . . 24
- 3.12 Generating Disorder Graphs From a .fasta File: . . . 24
- 3.13 Generating AlphaFold2 Confidence Score Graphs from fasta files . . . 26
- 3.14 Generating Graphs Using UniProt ID . . . 26
- 3.15 Generating AlphaFold2 Confidence Score Graphs Using UniProt ID . . . 26
- 3.16 Predicting Disorder Domains using a Uniprot ID: . . . 27

3.17 Batch prediction of disorder scores or disordered domains 27
3.18 Predicting Disorder Domains from external scores: 29

4 Python Module Documentation **33**
4.1 Recommended usage 33
4.2 metapredict functions 33

5 Acknowledgements **49**
5.1 Contributors 49

6 HELP! Metapredict isn't working! **51**
6.1 Python Version Issues 51
6.2 Running tests 51
6.3 Reporting Issues 52

7 Recent changes **53**

8 How to cite metapredict **55**

Python Module Index **57**

Index **59**

CHAPTER

ONE

GETTING STARTED WITH METAPREDICT

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).

metapredict is ALSO available via a webserver for single sequence prediction and a Google Colab notebook for batch prediction. However, this documentation here focuses on the Python package which provides both a set of Python library functions and a set of command-line tools.

1.2 metapredict updates and news

1.2.1 May 2023: Update to default version (metapredict V2-FF)

As of May 2023, we have pushed our improved version metapredict V2-FF. metapredict V2-FF does not change any of the predictions, but does implement substantial performance improvements. Notably these are realized by using the `predict_disorder_batch()` function.

1.2.2 February 2022: Update to default version (metapredict V2)

As of February 15, 2022 we have updated metapredict to V2. This comes with important changes that improve the accuracy of metapredict. Please see the section on the update *Major update to metapredict predictions to increase overall accuracy* below. In addition, this update changes the functionality of the *predict_disorder_domains()* function, so please read the documentation on that function if you were using it previously.

These changes are detailed in a permanent preprint that lives on bioRxiv. We ask you still cite the original metapredict article rather than this preprint.

1.2.3 July 2021: Initial version (metapredict v1)

The initial version of metapredict was released in July 2021 with the corresponding paper published shortly thereafter in Biophysical Journal:

Emenecker, R. J., Griffith, D. & Holehouse, A. S. Metapredict: a fast, accurate, and easy-to-use predictor of consensus disorder and structure. Biophys. J. 120, 4312–4319 (2021).

1.3 Installation

The current stable version of **metapredict** is available through GitHub or the Python Package Index (PyPI).

To install through PyPI, run:

```
$ pip install metapredict
```

You can also install the current development version from

```
$ pip install git+https://git@github.com/idptools/parrot
```

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 -e .
```

Note you will need the -e flag to ensure the *cython* code compiles correctly, but this also means the installed version is linked to the local version of the code.

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

1.4 About

It's important to understand how tools were built and developed. Below we provide a quick overview of how metapredict works and was trained.

1.4.1 How does metapredict work?

metapredict V2 (the current default version) works by combining consensus disorder predictions (which is how V1 was developed) with structural predictions to assign a residue as being disordered or folded.

The original metapredict (V1) was purely a consensus 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.

As of metapredict V2 (including V2-FF) the overall disorder prediction combines the V1 consensus disorder score with inverted AlphaFold2 predictions in a single deep learning network that provides robust, accurate, and fast assignment of individual residues as being either disordered or folded.

1.4.2 How was metapredict V1 trained?

metapredict V1 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 V1** can 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**).

Note that metapredict V1 predictions are available via the `legacy=True` flag.

1.4.3 How was metapredict V2 trained?

V2 was trained by generating an initial hybrid score that combined AlphaFold2 predicted pLDDT scores with consensus disorder along with some signal process algorithms to make a new structure/disorder consensus prediction. Finally, we trained a new deep learning network to predict our hybrid network (meta meta), substantially improving accuracy with very little loss in performance.

These changes and new assessment of performance are available in our preprint: An update to metapredict, a fast, accurate, and easy-to-use predictor of consensus disorder and structure. In bioRxiv (p. 2022.06.06.494887). https://doi.org/10.1101/2022.06.06.494887

As per the 2023 Critical Assessment of Intrinsic Disorder (CAID) competition, metapredict V2 is ranked the 9th most accurate disorder predictor available. However, importantly, it is among the fastest regardless of accuracy, and is accessible across multiple platforms, via a web server, and with very few software dependencies. Among the top 10, the difference in accuracy is 0.95 to 0.93 AUC, suggesting to us that all top 10 predictors are highly accurate. In short, we believe metapredict V2 hits a sweet spot of accuracy and performance.

1.4.4 How does metapredict V2 differ from V2-FF

metapredict V2 and V2-FF are identical in terms of predictions and features, with the major difference being that metapredict V2-FF offers batched predictions. Batched predictions are automatically parallelized on either the CPU or GPU. In addition, we rewrote the metapredict domain decomposition algorithm in C to provide a 10-20x improvement in performance for this step.

We note that V2-FF was released after CAID, so the performance reported there is the V2 network performance. Because metapredict V2-FF is implemented in a Google Colab notebook for batch prediction you don't have to take our word for it that it's fast; just upload a proteome and see for yourself!

Generating predicted pLDDT (AlphaFold2 confidence) scores in metapredict

In addition to predicting disorder scores, 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*, *Methanocaldococcus jannaschii*, *Plasmodium falciparum*, *Mycobacterium tuberculosis*, *Caenorhabditis elegans*, *Dictyostelium discoideum*, *Trypanosoma cruzi*, *Saccharomyces cerevisiae*, *Schizosaccharomyces pombe*, *Rattus norvegicus*, *Homo sapiens*, *Arabidopsis thaliana*, *Zea mays*, *Leishmania infantum*, *Staphylococcus aureus*, *Glycine max*, and *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.

1.4.5 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.

1.5 Why is metapredict useful?

We think **metapredict** is useful for three main reasons.

1. It's highly accurate, provide strong boundaries between disordered and folded regions.
2. It's incredibly fast; on CPUs one can predict every IDR in the human proteome in ~5 minutes. On modest GPUs one can predict every IDR in the human proteome in 40 seconds. This stands in stark contrast to other predictors which place length caps on sequences and can take hours per sequence.
3. It is easy to use and distributed via a wide range of channels. In addition to this Python package, metapredict is distributed as a stand-alone webserver, colab notebooks for large-scale predictions, and as an API for SHEP-HARD, our general-purpose toolkit for working with an annotating large protein datasets. This Python package further implements metapredict as both Python modules and as a set of command-line tools.

In summary, we believe metapredict provides the three key ingredients of a useful disorder predictor: it's extremely accurate, it's incredibly fast, and it's very easy to use.

1.6 How to cite

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

Emenecker, R. J., Griffith, D. & Holehouse, A. S. Metapredict: a fast, accurate, and easy-to-use predictor of consensus disorder and structure. Biophys. J. 120, 4312–4319 (2021).

Additionally, if you are using V2 (which is now the default) please make this clear in methods section. You should not feel obliged to cite the V2 preprint, and this pre-print exists solely so we could fully document the changes and test some edge cases in an accessible and clear way.

1.7 Known installation issues

Below we include documentation on known issues.

1.7.1 macOS libiomp clash

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.

CHAPTER TWO

METAPREDICT FROM THE COMMAND-LINE

2.1 Using the original metapredict network

We have recently updated the network that makes predictions for metapredict to massively improve accuracy. However, if you need to use the original metapredict predictor as opposed to our new, updated predictor, use the `-l` or `--legacy` flag!

2.2 Predicting disorder scores 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
```

Note that as of metapredict V2-FF this will automatically parallelize on a GPU or CPU if available. A progress bar will also be generated in the terminal.

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
```

Using the original metapredict network- To use the original metapredict network, simply use the `-l` or `--legacy` flag.

Example:

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

2.3 Predicting IDRs from a fasta file

The `metapredict-predict-idrs` command from the command line takes a .fasta file as input and returns a .fasta file containing the IDRs for every sequence from the input .fasta file.

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

Example

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

Note that as of metapredict V2-FF this will automatically parallelize on a GPU or CPU if available. A progress bar will also be generated in the terminal.

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.

Example

```
$ metapredict-predict-idrs /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
↪thisUser/Desktop/disorder_predictions/my_idrs.fasta
```

Using the original metapredict predictor To use the original metapredict predictor as opposed to our new, updated predictor, use the `-l` or `--legacy` flag! Note that if legacy mode is requested no parallelization is possible.

Example

```
$ metapredict-predict-idrs /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
↪thisUser/Desktop/disorder_predictions/my_idrs.fasta -l
```

Changing output threshold for disorder- To change the cutoff value for something to be considered disordered, simply use the `--threshold` flag and then specify your value. For legacy, the default is 0.42. For the new version of metapredict, the value is 0.5.

Example

```
$ metapredict-predict-idrs /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
↪thisUser/Desktop/disorder_predictions/my_idrs.fasta --threshold 0.3
```

2.4 Predicting disorder scores from 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:

```
$ metapredict-quick-predict ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN
```

Using the original metapredict network- To use the original metapredict network, simply use the `-l` or `--legacy` flag.

Example:

```
$ metapredict-quick-predict ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVA -
↪l
```

2.5 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
```

2.6 Plotting disorder profiles 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 dave 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

```
$ metapredict-graph-disorder /Users/thisUser/Desktop/interestingProteins.fasta -o /Users/
↪thisUser/Desktop/DisorderGraphsFolder/ --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-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
```

Using the original metapredict network- To use the original metapredict network, simply use the `-l` or `--legacy` flag.

Example:

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

2.7 Quick graphing of disorder scores

`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 ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN
```

Example:

```
$ metapredict-quick-graph ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN -p
```

Example:

```
$ metapredict-quick-graph ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN -
↪D 200
```

Using the original metapredict network- To use the original metapredict network, simply use the `-l` or `--legacy` flag.

Example:

```
$ metapredict-quick-graph ISQQMQAQPAMVKSQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN -l
```

2.8 Graphing disorder scores 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
```

Using the original metapredict network- To use the original metapredict network, simply use the `-l` or `--legacy` flag.

Example:

```
$ metapredict-uniprot Q8RYC8 -l
```

2.9 Graphing disorder using the common name of a protein

Sometimes you just don't know the UniProt ID for your favorite protein, and looking it up can be a pain. With the `metapredict-name` command, you can input the common name of your favorite protein and get a graph in return. Metapredict will also print the name of the organisms and the UniProt ID it found so you know you're looking at the correct protein. This is because this functionality queries your input protein name on UniProt and takes the top hit. Sometimes this is the protein you're looking for, but not always. To increase the likelihood of success, use your protein name and the organism name for this command.

Example

```
$ metapredict-name p53
```

will graph the metapredict disorder scores for the Homo sapiens p53 protein. This is because Homo sapiens p53 is the top hit on UniProt when you search p53. However...

```
$ metapredict-name p53 chicken
```

will graph the p53 from Gallus gallus!

Additional Usage

Changing the DPI

Changing the DPI will adjust the resolution of the graph. To change the DPI, use the `-D` or `--dpi` flag.

Example

```
$ metapredict-name p53 -D 300
```

Graphing predicted pLDDT scores

To add predicted pLDDT scores to the graph, use the `-p` or `--pLDDT` flag.

Example

```
$ metapredict-name p53 -p
```

Changing the title

To change the title, use the `-t` or `--title` flag.

Example

```
$ metapredict-name p53 -t my_cool_graph_of_p53
```

Using the legacy version of metapredict

To use the legacy version of metapredict for your disorder scores, use the `-l` or `--legacy` flag.

Example

```
$ metapredict-name p53 -l
```

Printing the full UniProt ID to your terminal

To have your terminal print the entire UniProt ID as well as the full protein sequence from your specified protein upon graphing, use the `-v` or `--verbose` flag.

Example

```
$ metapredict-name p53 -v
```

Turning off all printing to the terminal

By default, the *metapredict-name* command prints the UniProt ID as well as other information related to your protein to the terminal. The purpose of this is to make it explicitly clear which protein was graphed because grabbing the top hit from UniProt *does not guarantee* that it is the protein you want or expected. However, this behavior can be turned off by using the `-s` or `--silent` flag.

Example

```
$ metapredict-name p53 -s
```

2.10 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.

```
$ metapredict-graph-pLDDT <Path to .fasta file>
```

Example

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

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 dave 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
```

CHAPTER

THREE

METAPREDICT IN PYTHON

In addition to using metapredict from the command line, you can also use it directly in Python. This enables metapredict to be incorporated into your bioinformatic workflows with ease

First import metapredict:

```
import metapredict as meta
```

Once metapredict is imported, you can work with individual sequences or .fasta files. *For a list of all metapredict's public-facing functions and their documentation click here*

3.1 Important updates

3.1.1 Update to metapredict V2-FF (May 2023)

In May 2023 the default version of metapredict updated to be V2-FF. V2-FF introduces one primary difference from the user's perspective; disorder scores and disordered domains can now be predicted in parallel batches using:

```
import metapredict as meta

# batch disorder in batch
meta.predict_disorder_batch(...)
```

If GPU are available, batch prediction will automatically use GPUs. If not, batch prediction will distribute predictions across the CPUs. While all the original functionality is preserved, `predict_disorder_batch()`, offers a 5-10x speedup on CPUs and 30-40x speedup on GPUs.

`predict_disorder_batch()` can take in a list of sequences or a dictionary of sequences, and returns a list or dictionary that maps input index back to a two-position list of sequence and disorder scores or, if `return_disorder_domains` is set to True, a list of `DisorderDomain` objects.

This functionality is described in detail in the function documentation under the Python Module Documentation entry for `predict_disorder_batch()`.

3.1.2 Update to metapredict V2 (Feb 2022)

As of February 15, 2022 we have updated metapredict to V2. V2 provides a major improvement in accuracy and interpretability and works by incorporating in predictions made from AlphaFold2 to provide a new underlying prediction network. The original metapredict network is still available using the `legacy=True` flag. For more information, please see the section on the update *Major update to metapredict predictions to increase overall accuracy* below. In addition, this update changes the functionality of the `predict_disorder_domains()` function, so please read the documentation on that function if you were using it previously!

We released a preprint documenting all these changes and more!

3.2 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 -

```
meta.predict_disorder("DSSPEAPAEPPKDVPHDWLYSYVFLTHHPADFLR")
```

would output -

```
[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.

```
meta.predict_disorder("DSSPEAPAEPPKDVPHDWLYSYVFLTHHPADFLR", normalized=False)
```

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
meta.predict_disorder("DSSPEAPAEPPKDVPHDWLYSYVFLTHHPADFLR", legacy=True)
```

3.3 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 made of valid amino acids. Running -

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

would output -

```
[35.7925, 40.4579, 46.3753, 46.2976, 42.3189, 42.0248, 43.5976, 40.7481, 40.1676, 41.
↪9618, 43.3977, 43.938, 41.8352, 44.0462, 44.5382, 46.3081, 49.2345, 46.0671]
```

3.4 Predicting Disorder Domains:

The `predict_disorder_domains()` function takes in an amino acid sequence and returns a DisorderObject. The DisorderObject has 6 dot variables that can be called to get information about your input sequence. They are as follows:

.sequence
[str] Amino acid sequence

.disorder
[list or np.ndaarray] Hybrid disorder score

.disordered_domain_boundaries
[list] List of domain boundaries for IDRs using Python indexing

.folded_domain_boundaries
[list] List of domain boundaries for folded domains using Python indexing

.disordered_domains
[list] List of the actual sequences for IDRs

.folded_domains
[list] List of the actual sequences for folded domains

Examples

```
seq = meta.predict_disorder_domains(
↪"MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLVSLPPVGSLVVYFPQGHSEQVAASMQKQTDFIPNYPNLPSKLICLLHS")
```

Now we can call the various dot values for **seq**.

Getting the sequence

```
print(seq.sequence)
```

returns

```
MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLVSLPPVGSLVVYFPQGHSEQVAASMQKQTDFIPNYPNLPSKLICLLHS
```

Getting the disorder scores

```
print(seq.disorder)
```

returns

```
[0.922  0.9223 0.9246 0.9047 0.8916 0.8956 0.8931 0.883  0.8613 0.8573
0.852  0.8582 0.8614 0.8455 0.826  0.7974 0.7616 0.7248 0.6782 0.6375
0.5886 0.5476 0.5094 0.4774 0.4472 0.4318 0.4266 0.4222 0.3953 0.3993
0.3904 0.4004 0.3962 0.3721 0.3855 0.3582 0.3456 0.3682 0.3488 0.3274
0.3258 0.2937 0.2864 0.3004 0.3358 0.3815 0.4397 0.4594 0.4673 0.4535
0.4446 0.4481 0.4546 0.4454 0.4549 0.4564 0.4677 0.4539 0.4713 0.49
0.4934 0.4835 0.4815 0.4692 0.4548 0.4856 0.495  0.4809 0.502  0.4944
0.4612 0.4561 0.436  0.4203 0.3784 0.3624 0.3739 0.3983 0.4348 0.4369]
```

Getting the disorder domain boundaries

```
print(seq.disordered_domain_boundaries)
```

returns

```
[[0, 23]]
```

Where each nested list is the boundaries for a specific disordered region and the first element in each list is the start of that region and the second element is the end of that region.

Getting the folded domain boundaries

```
print(seq.folded_domain_boundaries)
```

returns

```
[[23, 80]]
```

Where each nested list is the boundaries for a specific folded region and the first element in each list is the start of that region and the second element is the end of that region.

Getting the disordered domain sequences

```
print(seq.disordered_domains)
```

returns

```
['MKAPSNGFLPSSNEGEKKPINSQ']
```

Where each element in the list is a specific disordered region identified in the sequence.

Getting the folded domain sequences

```
print(seq.folded_domains)
```

returns

```
['LWHACAGPLVSLPPVGSLVVYFPQGHSEQVAASMQKQTDFIPNYPNLPSKLICLLHS']
```

Where each element in the list is a specific folded region identified in the sequence.

Additional Usage

Altering the disorder theshhold - 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 = 0.5 (V2) and 0.42 (legacy).

Example

```
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

```
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 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 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

```
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 favor 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

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

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
predict_disorder_domains("MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLV", legacy=True)
```

3.5 Calculating Percent Disorder:

The `percent_disorder()` function will return the percent of residues in a sequence that are predicted to be disordered.

Running -

```
meta.percent_disorder("DSSPEAPAEPPKDVPHDWLPYSYVFGLGTPHGHPPADFGLR")
```

would output -

```
58.537
```

`Percent_disorder()` has two modes defined by the `mode` keyword: `threshold` and `disorder_domains`.

The default usage is with the `threshold` mode. In this case, each residue is evaluated against a threshold value, where disorder scores above that threshold count towards disordered residues. This mode uses a threshold value of 0.5 (for V2) or 0.3 (for legacy), although the threshold can be changed (see below).

The alternative mode, `disorder_domains`, makes use of metapredict's `predict_disorder_domains()` functionality. Now, the sequence is divided up into IDRs and folded domains, and then the percentage disordered is based on what fraction of residues fall into IDRs. The underlying disorder domain prediction uses the default disorder thresholds as per the `` predict_disorder_domains()` function, but this can be over-ridden if a ``disorder_threshold `` keyword is passed. For example:

```
meta.percent_disorder("DSSPEAPAEPPKDVPHDWLPYSYVFGLGTPHGHPPADFGLR", mode='disorder_domains
↪')
```

would output -

```
100.0
```

because the short 'folded' region where residue have a disorder score below the threshold are incorporated into the IDR in the `predict_disorder_domains()` function.

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=<value>` where the `<value>` corresponds to the percent (expressed as a fraction) you would like to use as the cutoff (for example, 0.8 would be 80%).

Example:

```
meta.percent_disorder("DSSPEAPAEPPKDVPHDWLYSYVFLTHHPADFLR", disorder_threshold= 0.8)
```

would output

```
26.471
```

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.

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
meta.percent_disorder("DSSPEAPAEPPKDVPHDWLYSYVFLTHHPADFLR", disorder_threshold= 0.8,␣
↪legacy=True)
```

would output

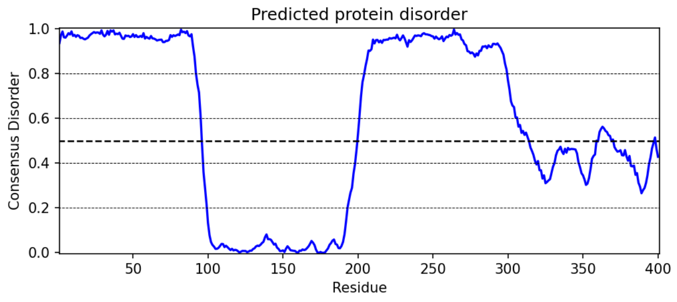
```
29.412
```

3.6 Graphing Disorder

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

```
meta.graph_disorder(
↪"GHPGKQRNPGEHHSSRNVKRNWNNSPSGPNEGRESQEERKTPPRRGGQQSGESHNQDETNKPNPSDNHHEEEKADDNAHRGNDSSPEAPAEPPKDVPHDW
↪")
```

would output -



Additional Usage

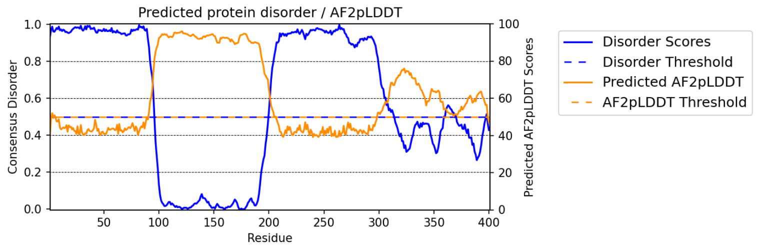
Adding Predicted AlphaFold2 Confidence Scores - To add predicted AlphaFold2 pLDDT confidence scores, simply specify `pLDDT_scores=True`.

Example

```
seq =
↪'GHPGKQRNPGEHHSSRNVKRNWNNSPSGPNEGRESQEERKTPPRRGGQQSGESHNQDETNKPNPSDNHHEEEKADDNAHRGNDSSPEAPAEPPKDVPHDWL
↪'

meta.graph_disorder(seq, pLDDT_scores=True)
```

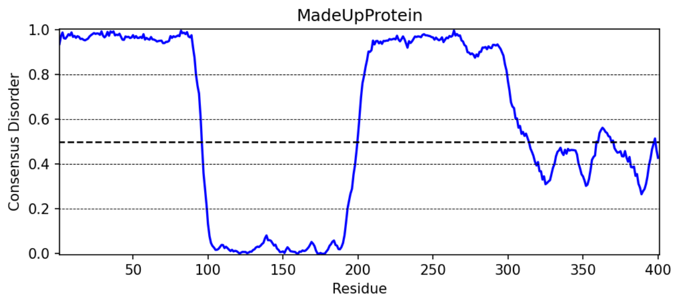
would output -



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 -

```
meta.graph_disorder(
↪"GHPGKQRNPGEHHSSRNVKRNWNNSPSGPNEGRESQEERKTPPRRGGQQSGESHNQDETNKPNPSDNHHEEEKADDNAHRGNDSSPEAPAEPPKDVPHDWL
↪", title = "MadeUpProtein")
```

would output -



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 `<number` is an integer.

Example:

```
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 a value between 0 and 1.

Example

```
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

```
meta.graph_disorder("DAPPTSQEHTQAEDKERDDAPPTSQEHTQAEDKERDDAPPTSQEHTQAEDKERD", 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

```
meta.graph_disorder("DAPPTSQEHTQAEDKERDDAPPTSQEHTQAEDKERDDAPPTSQEHTQAEDKERD", 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("DAPPTSQEHTQAEDKER", output_file=/Users/thisUser/Desktop/cool_graphs/
↪myCoolGraph.png)
```

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
meta.graph_disorder("DAPPTSQEHTQAEDKER", legacy=True)
```

3.7 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("DAPTSQEHTQAEDKERDSKTHPQKKQSPS")
```

This function has all of the same functionality as `graph_disorder`.

3.8 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:

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

An actual file path would look something like:

```
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:

```
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:

```
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:

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

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
meta.predict_disorder_fasta("/Users/thisUser/Desktop/coolSequences.fasta", legacy=True)
```

3.9 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

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

3.10 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

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

Using the original metapredict network- To use the original metapredict network, simply set legacy=True.

Example:

```
meta.predict_disorder_uniprot("Q8N6T3", legacy=True)
```

3.11 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

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

3.12 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:

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

An actual file path would look something like:

```
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

```
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:

```
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

```
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

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

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
meta.graph_disorder_fasta("/Users/thisUser/Desktop/coolSequences.fasta", output_dir="/
↪Users/thisUser/Desktop/folderForGraphs", legacy=True)
```

3.13 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.

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

3.14 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

```
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

```
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

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

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
meta.graph_disorder_uniprot("Q8N6T3", legacy=True)
```

3.15 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

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

3.16 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. This also returns a DisorderedObject. The DisorderObject has 6 dot variables that can be called to get information about your input sequence. They are as follows:

.sequence
[str] Amino acid sequence

.disorder
[list or np.ndaarray] Hybrid disorder score

.disordered_domain_boundaries
[list] List of domain boundaries for IDRs using Python indexing

.folded_domain_boundaries
[list] List of domain boundaries for folded domains using Python indexing

.disordered_domains
[list] List of the actual sequences for IDRs

.folded_domains
[list] List of the actual sequences for folded domains

Example

```
seq = meta.predict_disorder_domains_uniprot('Q8N6T3')
```

```
print(seq.disorder)
```

Using the original metapredict network- To use the original metapredict network, simply set `legacy=True`.

Example:

```
meta.predict_disorder_domains_uniprot('Q8N6T3' legacy=True)
```

3.17 Batch prediction of disorder scores or disordered domains

As of metapredict V2-FF (V2.6), metapredict enables GPU or CPU enabled batch prediction.

3.17.1 Predicting disorder scores in batch mode

The simplest usage is to pass a list of sequences to `predict_disorder_batch()` e.g.:

```
seqs = ['APSPASPPASPSA','PQPQPQPWQPWPQPW','ASDASFPAPSDPASDPA']

return_data = meta.predict_disorder_batch(seqs)
```

In this scenario, `return_data` is a list of three elements, where each element is itself a list that has two elements; the sequence and the per-residue disorder scores as an `np.ndarray`:

```
[['APSPASPPASPSA',
  array([0.8983, 0.9628, 0.9682, 0.9767, 0.9798, 0.9904, 0.9774, 0.9711,
         0.9656, 0.969 , 0.9361, 0.8879, 0.7606], dtype=float32)],
 ['PQPQPQPWQPWPQPW',
  array([0.9251, 0.9448, 0.949 , 0.9393, 0.9276, 0.9132, 0.8923, 0.8575,
         0.8385, 0.8138, 0.7777, 0.7366, 0.7164, 0.6184, 0.4999],
        dtype=float32)],
 ['ASDASFPAPSDPASDPA',
  array([0.8881, 0.9427, 0.95  , 0.9415, 0.9431, 0.9336, 0.9295, 0.9304,
         0.9299, 0.9377, 0.9351, 0.9235, 0.9137, 0.9203, 0.8864, 0.83  ,
         0.7037], dtype=float32)]]
```

Note also that by default this function will print a progress bar to report on how quickly predictions are running. If this is not desired, the progress bar can be turned off using `show_progress_bar=False` option in the function signature.

In addition to passing in a list of sequences, you can also pass in a dictionary of sequences with protein_id:sequence mapping. In this case, the function will return a dictionary that has the same key-value pairing as the input dictionary, but instead of key-value (protein_id:[sequence, disorder prediction]). In this way, predicting disorder scores for large sets of sequences becomes straight forward.

3.17.2 Predicting disordered domains in batch mode

For disordered domains, the same function can be used with `return_domains=True` set. If this is the case, the same input/output behavior (lists or dictionaries as inputs) can be used, but rather than returning a two-position list of sequence and disorder score, the return type is a single DisorderDomain object.

DisorderDomain objects are data structures that present a set of information about a protein. Each object has six so-called "dot variables" (object variables) that provide distinct information:

- *sequence* - reports on the sequence of the full protein
- *disorder* - reports on the per-residue disorder score for the whole protein (i.e. the same information that would be reported if `return_domains=False`
- *disordered_domain_boundaries* - is a list with 0 or more sublists, where those sublists define the start and end positions of the IDRs within the protein sequence. These domain boundaries follow Python notation, i.e. if a disordered region ran between residue 1 and 10 in a protein, the boundaries would be [0,9].
- *folded_domain_boundaries* - same conceptual idea as described for the *disordered_domain_boundaries*, except here the reciprocal folded domain boundaries are reported.
- *disordered_domains* - the actual amino acid sequence of the IDRs - i.e. the length of *disordered_domains* is the same as the length of *disordered_domain_boundaries*.
- *folded_domains* - the actual amino acid sequence of the folded domains - i.e. the length of *folded_domains* is the same as the length of *folded_domain_boundaries*.

As an example:

```
seqs = ['APSPASPPASPSA','PQPQPQPWQPWPQPW','ASDASFPAPSDPASDPA']

return_data = meta.predict_disorder_batch(seqs, return_domains=True)

# if we then examined one of the return objects
tmp = return_data[0]

```

(continues on next page)

(continued from previous page)

```
print(tmp)

        DisorderObject for sequence with 13 residues, 1 IDRs, and 0 folded domains
        Available dot variables are:
          .sequence
          .disorder
          .disordered_domain_boundaries
          .folded_domain_boundaries
          .disordered_domains
          .folded_domains

print(tmp.disordered_domains)
        ['APSPASPPASPSA']

print(disorder)
        [0.8983 0.9628 0.9682 0.9767 0.9798 0.9904 0.9774 0.9711 0.9656                ␣
↪0.969 0.9361 0.8879 0.7606]
```

The various options for changing the definition of a disordered domain are also available to be passed to `meta.predict_disorder_batch()`. For a complete list of possible input variables we recommend checking out the corresponding Python module documentation.

3.18 Predicting Disorder Domains from external scores:

The `predict_disorder_domains_from_external_scores()` function takes in an disorder scores, an amino acid sequence (optinally), and returns a DisorderObject. This function lets you use other disorder predictor scores and still use the predict_disorder_domains() functionality. The DisorderObject has 6 dot variables that can be called to get information about your input sequence. They are as follows:

.sequence
    [str] Amino acid sequence

.disorder
    [list or np.ndaarray] Hybrid disorder score

.disordered_domain_boundaries
    [list] List of domain boundaries for IDRs using Python indexing

.folded_domain_boundaries
    [list] List of domain boundaries for folded domains using Python indexing

.disordered_domains
    [list] List of the actual sequences for IDRs

.folded_domains
    [list] List of the actual sequences for folded domains

Examples

```
seq = meta.predict_disorder_domains_from_external_scores(disorder=[0.8577, 0.9313, 0.
↪9313, 0.9158, 0.8985, 0.8903, 0.8895, 0.869, 0.8444, 0.8594, 0.8643, 0.8605, 0.8697, 0.
↪8627, 0.8641, 0.8633, 0.8487, 0.8512, 0.8236, 0.8079, 0.8047, 0.8021, 0.7954, 0.7867,␣
↪0.7797, 0.7982, 0.7842, 0.7614, 0.7931, 0.8166, 0.8298, 0.8222, 0.8227, 0.8183, 0.8279,
```

(continues on next page)

(continued from previous page)

```
↪ 0.838, 0.8535, 0.8512, 0.8464, 0.8469, 0.8322, 0.8265, 0.794, 0.7827, 0.7699, 0.7575,␣
↪0.7178, 0.5988], sequence = 'MKAPSNGFLPSSNEGEKKPINSQLMKAPSNGFLPSSNEGEKKPINSQL')
```

Now we can call the various dot values for **seq**.

Getting the sequence

```
print(seq.sequence)
```

returns

```
MKAPSNGFLPSSNEGEKKPINSQLMKAPSNGFLPSSNEGEKKPINSQL
```

Getting the disorder scores

```
print(seq.disorder)
```

Getting the disorder domain boundaries

```
print(seq.disordered_domain_boundaries)
```

Getting the folded domain boundaries

```
print(seq.folded_domain_boundaries)
```

Getting the disordered domain sequences

```
print(seq.disordered_domains)
```

Getting the folded domain sequences

```
print(seq.folded_domains)
```

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

```
meta.predict_disorder_domains_from_external_scores("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

```
meta.predict_disorder_domains_from_external_scores("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

```
meta.predict_disorder_domains_from_external_scores("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 seprated 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

```
meta.predict_disorder_domains_from_external_scores("MKAPSNGFLPSSNEGEKKPINSQLWHACAGPLV",␣
↪gap_closure = 5)
```

CHAPTER

FOUR

PYTHON MODULE DOCUMENTATION

4.1 Recommended usage

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

```
import metapredict as meta
```

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

4.2 metapredict functions

`metapredict.`**`print_metapredict_legacy_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_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`**`(`*seq_len=500*, *num_seqs=100*, *verbose=True*, *batch=True*, *legacy=False*, *batch_mode=None*, *variable_length=False*`)`

Function that lets you test metapredicts performance on your local hardware.

Parameters

- **seqlen** (*int*) – Length of each random sequence to be tested. Default = 500.
- **num_seqs** (*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
- **batch** (*bool*) – Flag which, if set to true, means we use batch mode, else we use serial mode.
- **legacy** (*bool*) – Flag which determines if legacy (v1) or updated (v2) metapredict networks are used.
- **batch_mode** (*int*) – Flag which defines which batch_mode algorithm to use for batched predictions. Default = None which means the mode is dynamically picked. Can also be 1 or 2.
- **variable_length** (*bool*) – Flag which, if provided, means sequences vary between 20 and seq_len length.

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.`**`graph_disorder`**(*sequence*, *title='Predicted protein disorder'*, *disorder_threshold=None*, *pLDDT_scores=False*, *shaded_regions=None*, *shaded_region_color='red'*, *DPI=150*, *output_file=None*, *legacy=False*)

Function to plot the disorder of an input sequece. 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*) – Set to None by default such that if the user chooses to set legacy=True, the threshhold line will be at 0.3 and if legacy is set to false (default) then the threshold line will be at 0.5.

  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 for legacy and 0.5 for new version of metapredict.
- **pLDDT_scores** (*Bool*) – Sets whether to include the predicted pLDDT scores in the figure
- **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 or list of sts*) – 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). Alternatively a list where number of elements matches number in shaded_regions, assigning a color-per-shaded regions.
- **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.

- **legacy** (*bool*) – whether to use the legacy metapredict predictions

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

Return type
None

`metapredict.meta.graph_disorder_fasta`(*filepath*, *pLDDT_scores=False*, *disorder_threshold=None*, *DPI=150*, *output_dir=None*, *output_filetype='png'*, *invalid_sequence_action='convert'*, *indexed_filenames=False*, *legacy=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.

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_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 besaved to 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.
- **legacy** (*bool*) – Whether to use the legacy metapredict predictor.

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

Return type
None

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

Function to plot the disorder of an input sequece. 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
- **disorder_threshold** (*float*) – Set to None by default such that it will change depending of if the user sets legacy to True of if legacy remains = False. Can still be set manually.

  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.
- **legacy** (*bool*) – whether to use the legacy metapredict predictor

Returns

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

Return type

None

`metapredict.meta.`**`graph_pLDDT`**`(`*sequence*, *title='Predicted AF2 pLDDT Confidence Score'*, *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 sequece. 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 AF2 pLDDT Confidence Score"
- **disorder_scores** (*Bool*) – Whether to include disorder scores. Can set to False if you just want the AF2 confidence scores. Default = False

- **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. Default = None
- **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.`**`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_seqs/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 besaved to 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.`**`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 sequece. 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.`**`percent_disorder`**`(`*sequence*, *disorder_threshold=None*, *mode='threshold'*, *legacy=False*`)`

Function that returns the percent disorder for any given protein. By default, uses 0.5 as a cutoff for the new version of metapredict and 0.3 for the legacy version of metapredict (values greater than or equal to 0.5 will be considered disordered). If a value for cutoff is specified, that value will be used.

Mode lets you toggle between 'threshold' and 'disorder_domains'. If threshold is used a simple per-residue logic operation is applied and the fraction of residues above the disorder_threshold is used. If 'disorder_domains' is used then the sequence is divided into IDRs and folded domains using the predict_disordered_domains() function.

Parameters

- **sequence** (*str*) – Input amino acid sequence (as string) to be predicted.
- **disorder_threshold** (*float*) – Set to None by default such that it will change depending on whether legacy is set to True or False.

Sets a threshold which defines if a residue is considered disordered or not. Default for new metapredict = 0.5. Default for legacy metapredict is 0.3.

- **mode** (*str*) – Selector which lets you choose which mode to calculate percent disorder with. Default is 'threshold', meaning the percentage of disorder is calculated as what fraction of residues are above the disorder_threshold. Alternatively, 'disorder_domains' means we use the predict_disorder_domains() function and then calculate what fraction of the protein's residues are in the predicted IDRs.
- **legacy** (*bool*) – Whether or not to use the legacy metapredict.

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

Return type
float

`metapredict.meta.`**`predict_all`**`(`*sequence*`)`

Function to return all three types of predictions (legacy_metapredict, metapredict, and ppLDDT). Returns as a tuple of numpy arrays, with ppLDDT returned as normalized between 0 and 1 (rather than 0 and 100) so can be plotted on same axis easily.

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

Returns
[0] - metapredict disorder scores (updated metapredict disorder) [1] - legacy metapredict disorder (original metapredict disorder) [2] - normalized ppLDDT scores

Return type
tuple with three np.ndarrays

`metapredict.meta.`**`predict_disorder`**`(`*sequence*, *normalized=True*, *return_numpy=False*, *legacy=False*`)`

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
- **return_numpy** (*bool*) – Flag which if set to true means the function returns a np.array.
- **legacy** (*bool*) – Whether to use the original metapredict disorder predictor.

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

Return type
list or np.ndarray

`metapredict.meta.`**`predict_disorder_batch`**`(`*input_sequences*, *gpuid=0*, *return_domains=False*, *disorder_threshold=0.5*, *minimum_IDR_size=12*, *minimum_folded_domain=50*, *gap_closure=10*, *show_progress_bar=True*, *batch_mode=None*`)`

Batch mode predictor which takes advantage of PyTorch parallelization such that whether it's on a GPU or a CPU, predictions for a set of sequences are performed rapidly.

Batch mode was implemented in metapredict V2-FF, as is optimized for the hybrid network first released in V2. As such a few options are not available for batch mode which include:

- legacy - you cannot predict legacy metapredict scores with batch_mode
- normalize - all predictions are automatically normalized to fall between 0 and 1
- return_numpy - all disorder scores are returned as numpy arrays.

Note also that batch mode uses 32-bit float vectors whereas non-batch uses 64-bit float vectors, so the precise values in batch vs. non-batch may differ slighly, however this is a numerical precision difference, such that values by both methods are always within 1e-3 of one another.

Parameters

- **input_sequences** (*list or dictionary*) – A collection of sequences that are presented either as a list of sequences or a dictionary of key-value pairs where values are sequences.
- **gpuid** (*int*) – Identifier for the GPU being requested. Note that if this is left unset the code will use the first GPU available and if none is available will default back to CPU; in general it is recommended to not try and set this unless there's a specific reason why a specific GPU should be used
- **return_domains** (*bool*) – Flag which, if set to true, means we return DisorderDomain objects instead of simply the disorder scores. These domain objects include the boundaries between IDRs and folded domains, the disorder scores, and the individual sequences for IDRs and folded domains. This adds a small amount of overhead to the prediction, but typically only increase prediction time by 10-15%.
- **disorder_threshold** (*float*) – Used only if return_domains = True.

  Threshold used to deliniate between folded and disordered regions. We use a value of 0.5 because predict_disorder_batch does not support legacy.
- **minimum_IDR_size** (*int*) – Used only if return_domains = True.

  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*) – Used only if return_domains = True.

  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*) – Used only if return_domains = True.

  Defines the largest gap that would be 'closed'. Gaps here refer to a scenario in which you have two groups of disordered residues seprated by a 'gap' of un-disordered residues. In general large gap sizes will favour larger contigous 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.
- **show_progress_bar** (*bool*) – Flag which, if set to True, means a progress bar is printed as predictions are made, while if False no progress bar is printed. Default = True

- **batch_mode** (*string*) – Indictor which, if set to 'pack-n-pad', 'size-collect' will FORCE the batch algorithm to use mode 'pack-n-pad', 'size-collect' for batch decomposition.

  Mode 'size-collect' means we pre-filter sequences into groups where they're all the same length, avoiding padding/packing. This works in all versions of torch, and will be faster if you have very large datasets or have many copies of the same sequence.

  Mode 'pack-n-pad' involves padding/packing the sequences so that all sequences can be passed in a batchsize of 32. This is only available if pytorch 1.11 or higher is available. In testing, we found that pack-n-pad is about 2x faster than size-collect if running on CPU with variable length sequence if fewer 5000 sequences. On GPU, size-collect was consistently faster.

  Default = None, which means dynamic selection occurs. Default is to use size-collect because it is faster.

Returns

IF RETURN DOMAINS == FALSE: this function returns either a list or a dictionary.

If a list was provided as input, the function returns a list of the same length as the input list, where each element is itself a sublist where element 0 = sequence and element 1 is a numpy array of disorder scores. The order of the return list matches the order of the input list.

If a dictionary was provided as input, the function returns a dictionary, where the same input keys map to values which are lists of 2 elements, where element 0 = sequence and element 1 is a numpy array of disorder scores.

IF RETURN DOMAINS == TRUE: this function returns either a list or a dictionary.

If a list was provided as input, the function returns a list of the same length as the input list, where each element is a DisorderDomain object. The order of the return list matches the order of the input list.

If a dictionary was provided as input, the function returns a dictionary, where the same input keys map to a DisorderDomain object that corresponds to the input dictionary sequence.

Return type

dict or list

`metapredict.meta.`**`predict_disorder_caid`**(*input_fasta*, *output_file*)

executing script for generating a caid-compliant output file for disorder predictions using a .fasta file as the input.

Parameters

- **input_fasta** (*str*) – the input file as a string that includes the file path preceeding the file name if the file is not in the curdir
- **output_file** (*str*) – the output file name as a string. This can include a file path to a specific save location or by default saves to the curdir

Returns

Does not return anything, saves a file to the destination output file

Return type

None

`metapredict.meta.`**`predict_disorder_domains`**(*sequence*, *disorder_threshold=None*, *minimum_IDR_size=12*, *minimum_folded_domain=50*, *gap_closure=10*, *normalized=True*, *return_numpy=True*, *legacy=False*, *return_list=False*)

This function takes an amino acid sequence and one or more variable options and returns a data structure called a *DisorderObject*. The object parameters associated with this object are defined below.

The previous version of metapredict returned a list of values, which can be obtained instead of the DisorderedObject if return_list is set to True.

Parameters

- **sequence** (*str*) – Amino acid sequence
- **disorder_threshold** (*float*) – Set to None such that it will change to 0.42 for legacy and 0.5 for metapredict. Can still manually set value.

  Value that defines what 'disordered' is based on the metapredict disorder score. The higher the value the more stringent the cutoff. Default = 0.5 for new version and 0.42 for legacy metapredict.
- **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 seprated by a 'gap' of un-disordered residues. In general large gap sizes will favour larger contigous 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.
- **normalized** (*bool*) – whether the disorder scores are normalized between zero and one, default is true
- **return_numpy** (*bool*) – Flag which if set to true means all numerical types are returned as numpy.ndlist. Default is True
- **legacy** (*bool*) – Whether to use the original metapredict network
- **return_list** (*bool*) – Flag that determines i to return the old format where a tuple is returned. This is retained for backwards compatibility

Returns

By default, the function returns a DisorderObject. A DisorderObject has 7 dot variables:

.sequence
[str] Amino acid sequence

.disorder
[list or np.ndaarray] disorder scores

.disordered_domain_boundaries
[list] List of domain boundaries for IDRs using Python indexing

.folded_domain_boundaries
[list] List of domain boundaries for folded domains using Python indexing

.disordered_domains
[list] List of the actual sequences for IDRs

.folded_domains
[list] List of the actual sequences for folded domains

Return type
DisorderObject

Returns

However, if `return_list` == True. Then, the function 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.`**`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*, *return_numpy=True*)

This function takes in disorder scores generated from another predictor and applies the same domain-decomposition algorithm as predict_disorder_domains() does to extract out congigous 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.

Parameters

- **disorder** (*list*) – A list of per-residue disorder scores.
- **sequence** (*str*) – The protein sequence as a string. If no sequence is passed, calling DisorderObject.sequence will return an fake sequence.
- **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 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 seprated by a 'gap' of undisordered residues. In general large gap sizes will favour larger contigous 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.
- **return_numpy** (*bool*) – Flag which if set to true means all numerical types are returned as numpy.ndlist. Default is True

Returns

Returns a DisorderObject. DisorderObject has 7 dot variables:

.sequence
[str] Amino acid sequence

.disorder
[list or np.ndaarray] Hybrid disorder score

.disordered_domain_boundaries
[list] List of domain boundaries for IDRs using Python indexing

.folded_domain_boundaries
[list] List of domain boundaries for folded domains using Python indexing

.disordered_domains
[list] List of the actual sequences for IDRs

.folded_domains
[list] List of the actual sequences for folded domains

Return type
DisorderObject

`metapredict.meta.`**`predict_disorder_domains_uniprot`**`(`*uniprot_id*, *disorder_threshold=None*, *minimum_IDR_size=12*, *minimum_folded_domain=50*, *gap_closure=10*, *normalized=True*, *return_numpy=True*, *legacy=False*`)`

This function takes an amino acid sequence, a disorder score, and returns either a DisorderObjec 4-position tuple with the information listed below.

Parameters

- **uniprot_ID** (*String*) – The uniprot ID of the sequence to predict
- **sequence** (*str*) – Amino acid sequence
- **disorder_threshold** (*float*) – Set to None by default such that the threshold value is is dependent on whether legacy is set to True. The default for legacy is 0.42, the default for the new metapredict is 0.5.

  Value that defines what 'disordered' is based on the metapredict disorder score.
- **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 seprated by a 'gap' of un-disordered residues. In general large gap sizes will favour larger contigous 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.
- **return_numpy** (*bool*) – Flag which if set to true means all numerical types are returned as numpy.ndlist. Default is True

Returns

Returns a DisorderObject. DisorderObject has 7 dot variables:

.sequence
[str] Amino acid sequence

.disorder
[list or np.ndaarray] Hybrid disorder score

.disordered_domain_boundaries
[list] List of domain boundaries for IDRs using Python indexing

.folded_domain_boundaries
[list] List of domain boundaries for folded domains using Python indexing

.disordered_domains
[list] List of the actual sequences for IDRs

.folded_domains
[list] List of the actual sequences for folded domains

Return type

DisorderObject

`metapredict.meta.`**`predict_disorder_fasta`**(*filepath*, *output_file=None*, *normalized=True*, *invalid_sequence_action='convert'*, *legacy=False*)

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.

Parameters

- **filepath** (*str*) – The path to where the .fasta file is located. The filepath should end in the file name, and can be an absolute or relative path
- **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.
- **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
- **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.
- **legacy** (*bool*) – Whether to use the legacy metapredict predictor. Default = False.

Returns

If output_file is set to None (as default) then this fiction returns a dictionary of sequence ID to disorder np.ndarrays(dtype=np.float32).

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.`**`predict_disorder_uniprot`**`(`*uniprot_id*, *normalized=True*, *legacy=False*`)`

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

`metapredict.meta.`**`predict_pLDDT`**`(`*sequence*, *return_numpy=False*, *normalized=False*`)`

Function to return predicted pLDDT scores. pLDDT scores are the scores reported by AlphaFold2 (AF2) that provide a measure of the confidence with which AF2 has on the local structure prediction. predicted_pLDDT (ppLDDT for short) is a prediction of this confidence score generated using a LSTM-BRNN network trained on ~360,000 protein structures.

In effect, this value should be considered a prediction of how confident we are that AF2 would be able to predict the structure. This is a reasonably good proxy for the prediction that a region will be structured but is not perfect.

Parameters

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

- **return_numpy** (*bool*) – Flag which, if set to true, means the function returns a numpy array instead of a list.
- **normalized** (*bool*) – Flag which, if set to true, means the function returns values scaled between 0 and 1 (rather than 0 and 100).

Returns
Returns a list (or np.ndarray) of floats that corresponds to the per-residue pLDDT score. Return type depends on the flag return_numpy

Return type
list or np.ndarray

`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, and can be an absolute or relative path

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 fiction 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.`**`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

CHAPTER FIVE

ACKNOWLEDGEMENTS

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 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 and EBI/UniProt for making these data so readily available.

5.1 Contributors

We'd also like to thank the following folks who have contribute code, reported errors, and suggested changes.

- The Fried lab (broadly defined)
- Broder Schmidt
- Sean Cascarina
- Keith Cheveralls

CHAPTER

SIX

HELP! METAPREDICT ISN'T WORKING!

6.1 Python Version Issues

We 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 developed using Python version 3.7, but has been tested on 3.8, 3.9 and 3.10 as well. However, metapredict was developed for macOS and Linux, and while we expect it to work for Windows this has been far less rigorously tested.

If you commonly use a Python version outside of the 3.7 - 3.10 window, a convenient workaround is to use a conda environment that has Python 3.8 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=3.8
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 Running tests

If you would like to check if metapredict is working, you can also run the test suite found in the source directory (*metapredict/tests*).

To run all tests simply run:

```
pytest --versbose
```

From within the directory. Note you may need to install pytest first:

```
pip install pytest
```

6.3 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

CHAPTER

SEVEN

RECENT CHANGES

As of version 2.6, a single changelog file is kept in the main metapredict GitHub repo in the main README.md file. We removed the changelog from the documentation to remove redundancy.

CHAPTER EIGHT

HOW TO CITE METAPREDICT

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

Emenecker RJ, Griffith D, Holehouse AS, metapredict: a fast, accurate, and easy-to-use predictor of consensus disorder and structure, Biophysical Journal (2021), doi: https:// doi.org/10.1016/j.bpj.2021.08.039.

Note that in addition to the original paper, there's a V2 preprint; HOWEVER, we ask you only cite the original paper and describe the version being used (V2 or V2).

Emenecker, R. J., Griffith, D. & Holehouse, A. S. Metapredict V2: An update to metapredict, a fast, accurate, and easy-to-use predictor of consensus disorder and structure. bioRxiv 2022.06.06.494887 (2022). doi:10.1101/2022.06.06.494887

PYTHON MODULE INDEX

m

metapredict, 33
metapredict.meta, 34

INDEX

G

`graph_disorder()` (*in module metapredict.meta*), 34
`graph_disorder_fasta()` (*in module metapredict.meta*), 35
`graph_disorder_uniprot()` (*in module metapredict.meta*), 35
`graph_pLDDT()` (*in module metapredict.meta*), 36
`graph_pLDDT_fasta()` (*in module metapredict.meta*), 37
`graph_pLDDT_uniprot()` (*in module metapredict.meta*), 38

M

`metapredict`
    `module`, 33
`metapredict.meta`
    `module`, 34
`module`
    `metapredict`, 33
    `metapredict.meta`, 34

P

`percent_disorder()` (*in module metapredict.meta*), 38
`predict_all()` (*in module metapredict.meta*), 39
`predict_disorder()` (*in module metapredict.meta*), 39
`predict_disorder_batch()` (*in module metapredict.meta*), 39
`predict_disorder_caid()` (*in module metapredict.meta*), 41
`predict_disorder_domains()` (*in module metapredict.meta*), 41
`predict_disorder_domains_from_external_scores()` (*in module metapredict.meta*), 43
`predict_disorder_domains_uniprot()` (*in module metapredict.meta*), 44
`predict_disorder_fasta()` (*in module metapredict.meta*), 45
`predict_disorder_uniprot()` (*in module metapredict.meta*), 46
`predict_pLDDT()` (*in module metapredict.meta*), 46
`predict_pLDDT_fasta()` (*in module metapredict.meta*), 47
`predict_pLDDT_uniprot()` (*in module metapredict.meta*), 47
`print_metapredict_legacy_network_version()` (*in module metapredict*), 33
`print_metapredict_network_version()` (*in module metapredict*), 33
`print_performance()` (*in module metapredict*), 33