metapredict Documentation

metapredict

Aug 27, 2022

Contents:

CHAPTER 1

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).

Important update - 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.

We recently released a preprint documenting all these changes and more!

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

To access the web server go to metapredict.net.

1.2 How does metapredict work?

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

1.2.1 This already seems complicated...

metapredict is a deep-learning-based predictor trained on consensus disorder data from 8 different predictors, as pre-computed and provided by MobiDB. Functionally, this means each residue is assigned a score between 0 and 1 which reflects the confidence we have that the residue is disordered (or not). If the score was 0.5, this means half of the predictors predict that residue to be disordered. In this way, **metapredict** can help you quickly determine the likelihood that residues are disordered by giving you an approximation of what other predictors would predict (things got pretty 'meta' there, hence the name **metapredict**).

1.2.2 Major update to metapredict predictions to increase overall accuracy

We are always working to make metapredict better, and we have recently managed just that. More details will be below, but the short story is that we have made significant improvements in the accuracy of disorder predictions using metapredict. By analyzing our new network using the Disprot-PDB dataset predictions, we found that the MCC (which is a measurement accounting for false positives, false negatives, true positives, and true negatives) for metapredict increased from 0.588 for the old (original) network to 0.7 for our new network. To put this in perspective, our original network was ranked 12th most accurate when analyzing the Disprot-PDB dataset, and by our own estimates V2 is now ranked as the 2nd most accurate available predictor.

For more information on the changes made please see our recent preprint:

Emenecker, R. J., Griffith, D., & Holehouse, A. S. (2022). Metapredict V2: 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

Any questions, please don't hesitate to reach out!

1.3 But wait! I need the old metapredict predictions!!!

No worries! We left users access to the old network. The *default network is now our new, more accurate network*. However, by calling `-l` or `--legacy` from the command line or by specifying `legacy=True` from Python, you will be able to use the original metapredict network. We wanted to keep making metapredict better, but we also wanted to minimize disruptions to anyone currently relying on the original metapredict predictions for whatever reason.

1.4 So... how exactly was this more accurate metapredict network made?

We didn't think it was possible, but metapredict has somehow become *even more meta*. Get ready, because things are about to get a little weird. When we implemented the AlphaFold2 pLDDT prediction feature (see section below), we noticed that there were occasional discrepancies between metapredict and the predicted pLDDT (ppLDDT) scores. When the ppLDDT scores get high enough, it is unlikely that a given region is actually disordered. So, we developed a version of metapredict that we originally called 'metapredict-hybrid' that essentially combined aspects of the ppLDDT scores and the original metapredict scores. We found that this 'hybrid predictor' was **much better** than the original metapredict disorder predictor at predicting disordered regions. **But we didn't stop there.**

We think one of metapredicts best features is *it is really really fast*. This 'hybrid-predictor' was a little on the slow side, coming in at about 1/3 the speed of the original metapredict predictor. This is still VERY fast, but we thought we could do better. So, we took a little over 300,000 protein sequences and generated metapredict-hybrid scores for those sequences. We then fed those sequences and the corresponding metapredict-hybrid scores and generated a new bidirectional recurrent neural network (BRNN) using PARROT. We then tested this new network against the original metapredict-hybrid predictions and the original metapredict network. The new network that was trained

on metapredict-hybrid scores *actually outperformed the metapredict-hybrid predictions when benchmarking against Disprot-PDB*. Importantly, this new (super accurate) network was only 30% slower than the original metapredict network, which is substantially better than the 70% hit that metapredict-hybrid took.

TL;DR We made the original metapredict predictor using a network trained on consensus scores from MobiDB. We then trained a network on AlphaFold2 pLDDT scores. Next, we made a predictor that combined prediction values from the original metapredict predictor and the AlphaFold2 pLDDT predictor to make very accurate disorder predictions. Finally, we took hundreds of thousands of proteins, generated disorder prediction scores using the aforementioned combination of the original metapredict predictions and the AlphaFold2 predictions, and then trained our final network on those scores. **That's pretty dang meta.**

1.5 Generating AlphaFold2 pLDDT scores in metapredict

In addition, metapredict offers predicted confidence scores from AlphaFold2. These predicted scores use a bidirectional recurrent neural network (BRNN) trained on the per residue pLDDT (predicted IDDT-Ca) confidence scores generated by AlphaFold2 (AF2). The confidence scores (pLDDT) from the proteomes of *Danio rerio*, *Candida albicans*, *Mus musculus*, *Escherichia coli*, *Drosophila melanogaster*, *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.

1.6 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.

1.7 Why is metapredict useful?

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

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

1.8 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.

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 soley so we could fully document the changes and test some edge cases in an accessible and clear way.

1.9 Installation

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

```
$ pip install metapredict
```

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

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

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

1.10 Known installation/execution issues

Below we include documentation on known issues.

1.11 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,
```

(continues on next page)

(continued from previous page)

```
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.

1.12 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
```

1.13 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 2

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 from Fasta Files

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

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

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

Example:

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

Additional Usage:

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

Example:

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

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 Disorder from a Sequence

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

Example:

```
$ 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
```

2.4 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
```

2.5 Graphing Disorder from a Fasta file

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

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

Example

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

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

Additional Usage

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

Example

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

Specifying where to save the output - To specify where to 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
```

2.6 Quick Graphing

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

Example:

```
$ metapredict-quick-graph ISQQMQAQPAMVKSQQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN
```

Example:

```
$ metapredict-quick-graph␣
↪ISQQMQAQPAMVKSQQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN -p
```

Example:

```
$ metapredict-quick-graph␣
↪ISQQMQAQPAMVKSQQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN -D 200
```

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

Example:

```
$ metapredict-quick-graph␣
↪ISQQMQAQPAMVKSQQQQQQQQQQQHQHQQQQLQQQQQLQMSQQQVQQQGIYNNGTIAVAN -l
```

2.7 Graphing using Uniprot ID

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

Example:

```
$ metapredict-uniprot Q8RYC8
```

Example:

```
$ metapredict-uniprot Q8RYC8 -p
```

Example:

```
$ metapredict-uniprot Q8RYC8 -D 300
```

Example:

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

Example:

```
$ metapredict-uniprot Q8RYC8 -o /Users/ThisUser/Desktop/MyFolder/DisorderGraphs/my_
↪graph.png
```

Example:

```
$ metapredict-uniprot Q8RYC8 -t ARF19
```

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

Example:

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

2.8 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
```

2.9 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
```

2.10 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
```

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!

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
```

CHAPTER 3

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 update to predict_disorder_domains() function for V2.0 and above

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 recently 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 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("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 metapredictis `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

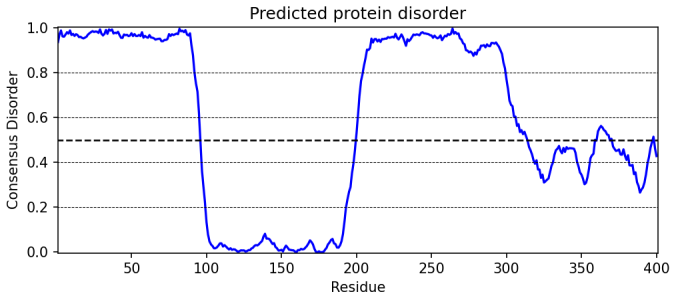
```
29.412
```

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(
↪"GHPGKQRNPGEHHSSRNVKRNWNNSPSGPNEGRESQEERKTPPRRGGQQSGESHNQDETNKPNPSDNHHEEEKADDNAHRGNDSSPEAPAEPPKDVPP
↪")
```

would output -



Additional Usage

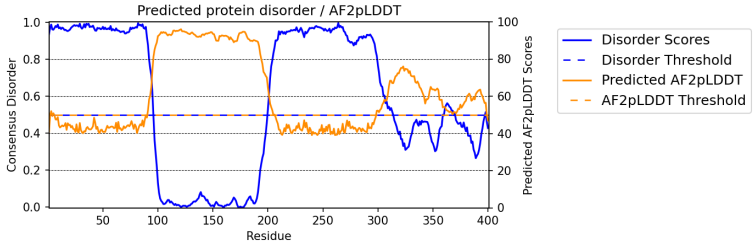
Adding Predicted AlphaFold2 Confidence Scores - To add predicted AlphaFold2 pLDDT confidence scores, simply specify `pLDDT_scores=True`.

Example

```
seq =
↪'GHPGKQRNPGEHHSSRNVKRNWNNSPSGPNEGRESQEERKTPPRRGGQQSGESHNQDETNKPNPSDNHHEEEKADDNAHRGNDSSPEAPAEPPKDVPP
↪'

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

would output -



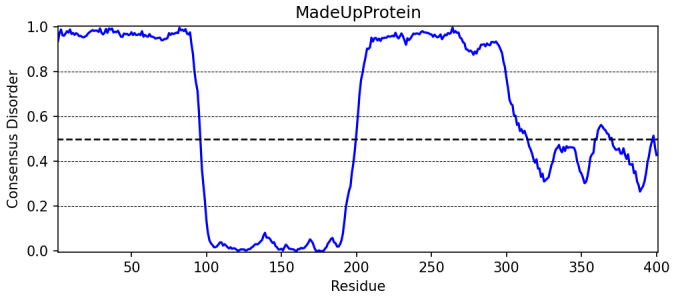
Changing title of generated graph - There are two parameters that the user can change for graph_disorder(). The first is the name of the title for the generated graph. The name by default is blank and the title of the graph is simply *Predicted protein disorder*. However, the title can be specified by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

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

(continues on next page)

(continued from previous page)

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 Confidnce 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 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, 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 4

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*, *legacy=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
- **legacy** (*bool*) – Flag which determines if legacy (v1) or updated (v2) metapredict networks are used.

Returns Returns the nearest number of sequences-per-second metapredict is currently predicting. For ref, on a spring 2020 MBP this value was ~10,000 sequences per second.

Return type int

metapredict.meta.**predict_disorder_domains**(*sequence*, *disorder_threshold=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 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.
- **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`**(*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.**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.**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.**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_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 vector. If output_file is set to a filename then a .csv file will instead be written and no return data will be provided.

Return type dict or None

metapredict.meta.**graph_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.**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.**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.**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 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.
- **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_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.**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.**predict_pLDDT_uniprot**(*uniprot_id*)

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

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

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

Return type None

metapredict.meta.**graph_pLDDT_fasta**(*filepath*, *DPI=150*, *output_dir=None*, *output_filetype='png'*, *invalid_sequence_action='convert'*, *indexed_filenames=False*)

Function to make graphs of predicted pLDDT from the sequences in a specified .fasta file. By default will save the generated graphs to the location output_path specified in filepath.

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

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

Parameters

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

CHAPTER 5

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 6

HELP! Metapredict isn't working!

6.1 Python Version Issues

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

Once you have conda installed, simply use the command

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

and once activate install metapredict from PyPI

```
pip install metapredict
```

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

6.2 Reporting Issues

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

CHAPTER 7

Recent changes

7.1 About

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

7.2 V2.4.2

Changes:

- Integrated in changes from @FriedLabJHU to make f-strings more Pythonic
- Changed `return_normalized` keyword to `normalized` in `meta.predict_pLDDT()` for consistency with other functions
- Added sanity check in case a passed sequence is an empty string (h/t Broder Schmidt)
- Added docs for the mode keyword in *meta.percent_disorder()*, so this is actually obvious to understand
- Added several additional tests and updated the docs.

7.3 V2.4.1

Changes:

- Some minor bug fixes and updates to code

7.4 V2.3

Changes:

- Merged pull request from @FriedLabJHU to fix keyword name *cutoff* to *disorder_threshold* in *meta.percent_disorder()*. Thanks!!
- Added the *mode* keyword into *meta.percent_disorder()* so disorder can be predicted in terms of what percentage of residues fall within IDRs, as well as what percent are above some fixed threshold.

7.5 V2.2

Changes: Fixed bug in metapredict-name command that could result in the organism name being named twice in the title of the graph.

7.6 V2.1

Changes: Added functionality to graph the disorder of a protein by specifying its common name using the *metapredict-name* command.

7.7 V2.0

Changes: Massive update to the network behind metapredict to improve accuracy. Implementation of code to keep the original network accessible to users. Changes to predict_disorder_domain functions where a DisorderObject is no returned and access to values are used by calling properties from the generated object. Graphing functionality updated to accomadate new cutoff value for the new network at 0.5. If the original metapredict network is used, then the cutoff value automatically resets to the original value of 0.3. Tests updated.

7.8 V1.51

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

7.9 V1.5

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

7.10 V1.4

Change: For clarity, previous functions that used the term 'confidence' such as *graph_confidence_uniprot()* were changed to use the term pLDDT rather than confidence. This is to clarify that the confidence scores are AlphaFold2 pLDDT confidence scores and not scores to reflect the confidence that the user should have in the metapredict disorder prediction. For command-line usage where confidence scores are optional (such as metapredict-graph-disorder), when

a *-c* or *–confidence* flag used to be used, now a *-p* or *–pLDDT* flag is used to graph confidence scores. This is similarly reflected in Python where now you must use *pLDDT_scores=True* instead of *confidence_scores=True*.

7.11 V1.3

Change: Added functionality to generate predicted AlphaFold2 confidence scores. Can get scores or generate graphs from Python or command-line. Can also generate graphs with both predicted disorder and predicted confidence scores. Also added functionality to predict disorder domains using scores from a different disorder predictor.

7.12 V.1.2

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

7.13 V.1.1

Change: Fixed some bugs.

7.14 V.1.0

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

7.15 V.061

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

7.16 V.060

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

7.17 V0.58

Change: Updated the network with a newly trained network (using the same dataset as the original) that is slightly more accurate.

Reason: I am always trying to find ways to make metapredict more accurate. When I manage to make the predictor better, I will update it.

7.18 V0.57

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

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

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

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

7.19 V0.56

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

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

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

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

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

CHAPTER 8

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.

Python Module Index

m

Index