PairProphet

Release 1.0.1

Logan Roberts, Humood Alanzi, Chau Vuong, Ryan Francis, Amin

Jun 28, 2023

CONTENTS:

1 Overview **3**

2 Requirements and Installation **5**

3 Instructions and Use Case **7**

4 PairProphet package **9**
- 4.1 Submodules . . . 10
- 4.2 pairpro.dev_tools module . . . 10
- 4.3 pairpro.evaluate_input_cleaning module . . . 10
- 4.4 pairpro.evaluate_model module . . . 11
- 4.5 pairpro.hmmer module . . . 11
- 4.6 pairpro.preprocessing module . . . 19
- 4.7 pairpro.structures module . . . 20
- 4.8 pairpro.train_val_classification module . . . 22
- 4.9 pairpro.train_val_featuregen module . . . 23
- 4.10 pairpro.train_val_input_cleaning module . . . 24
- 4.11 pairpro.train_val_wrapper module . . . 25
- 4.12 pairpro.user_blast module . . . 25
- 4.13 pairpro.utils module . . . 26

5 Acknowledgements **29**

6 References **31**

7 Our Pledge **33**

Python Module Index **37**

Index **39**

PairProphet is a software pipeline developed at the University of Washington. This package implements a "first-pass" model that allows researchers to draw on existing proteome data and generate a pairwise screening that will narrow down their search space and vastly reduce screening time. Our pipeline leverages machine learning to train a model based on alignment metrics from multiple programs, centralizing the searching process to one software pipeline that can be executed with a single command.

For more information, please visit our Github. For contributions, see here.

CHAPTER

ONE

OVERVIEW

Protein pair validation is time consuming and resource intensive, given that proteins can be related through many unique functions, both direct and inferred. Many unique softwares specialize in characterizing protein based on a few of these functions. Our pipeline aims to combine different softwares, spanning sequence alignment, structure and folding prediction, and residue conservation into a single pipeline to improve prediction quality and streamline the characterization process.

CHAPTER

TWO

REQUIREMENTS AND INSTALLATION

To create and activate the environment specified in *environment.yml* in our Github:

To install the PairProphet package, use the following commands:

```
conda env create --file environment.yml
```

```
conda activate pairpro
```

```
pip install .
```

PairProphet is dependent on Python 3.11.

CHAPTER

THREE

INSTRUCTIONS AND USE CASE

Training

PairProphet is trained on the Learn2Therm database, developed by the University of Washington's Beck Research Group.

Use Case

Assess protein pair functionality with model developed during training with **user_input.py**.

Input:
CSV file with protein pairs.

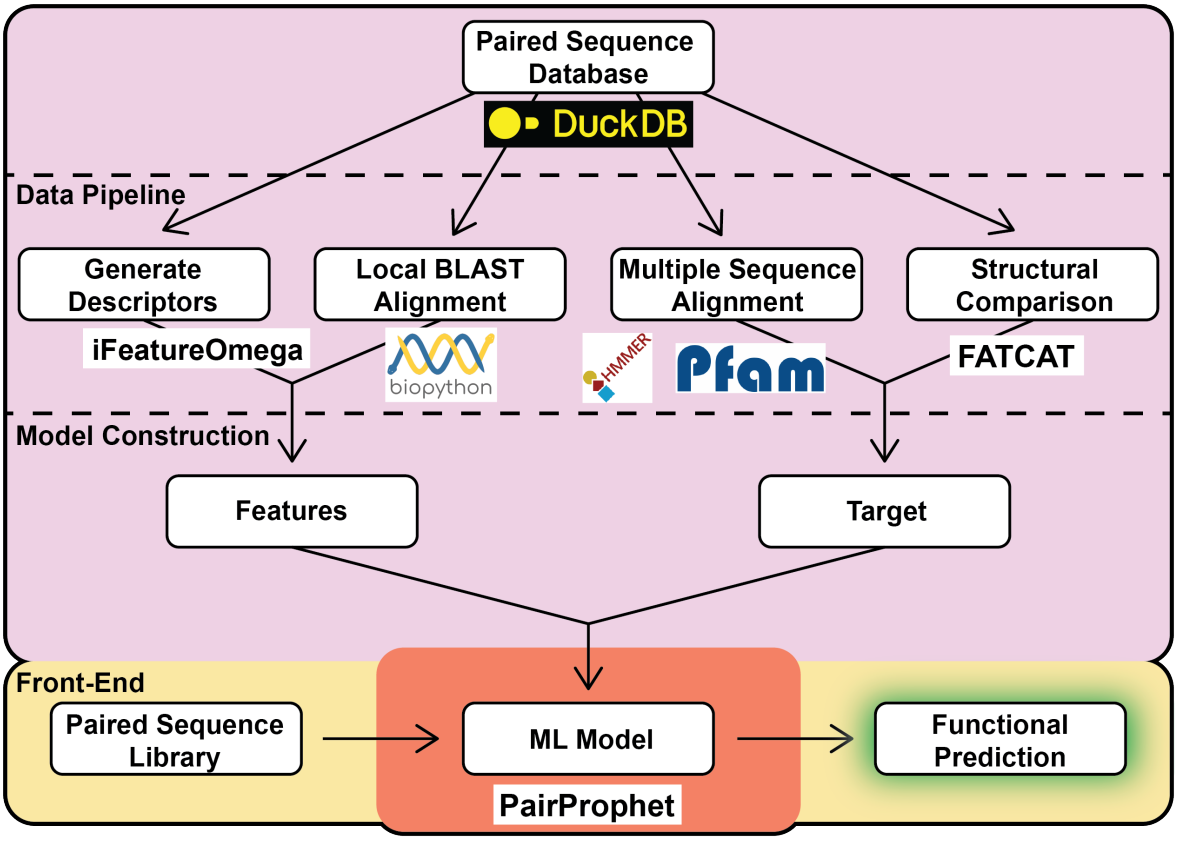
Output:
CSV with:
- protein pairs
- hmmer functionality (Boolean (1=True, 0=False))
- structure functionality (Boolean (1=True, 0=False))

.txt file with:
- accuracy
- mean precision
- mean F1 score
- mean recall

CHAPTER

FOUR

PAIRPROPHET PACKAGE



The following submodules contain source code for developing our machine learning model.
For a detailed description of the pipeline, see here.

4.1 Submodules

4.2 pairpro.dev_tools module

pairpro.dev_tools.**build_sample_l2t**(*db_in*, *db_out*, *size*)

Generates a sample l2t relational database of given size. Note that the final size will about 30% of 'size' due to pair filtering.

Parameters

- **db_in** (*str*) – Path to full size l2t database
- **db_out** (*int*) – Path to sample l2t database to be created
- **size** (*str*) – Number of pairs to sample for test database. Final size will be about 30% of this.

Returns

None. Database file is saved at db_out.

Raises

None. –

4.3 pairpro.evaluate_input_cleaning module

This module cleans dataframe from user input for upstream classifier. Keeps both protein sequences for reporting results.

pairpro.evaluate_input_cleaning.**check_input_nans**(*dataframe*)

Checks for NaN values in input dataframe. Removes rows with NaN values present. :param pandas dataframe:

Returns

pandas dataframe

pairpro.evaluate_input_cleaning.**check_input_type**(*dataframe*)

Takes in input dataframe and asserts that it is the correct data type.

Parameters

dataframe (*pandas*) –

Returns

pandas dataframe

pairpro.evaluate_input_cleaning.**clean_input_columns**(*dataframe*)

Cleans out columns that are not in a predefined list of features.

Parameters

dataframe (*pandas*) –

Returns

pandas dataframe

pairpro.evaluate_input_cleaning.**input_cleaning_wrapper**(*dataframe*, *structure*)

Takes in a pandas dataframe and runs it through each of the cleaning and verification steps. :param pandas dataframe:

Returns

pandas dataframe

pairpro.evaluate_input_cleaning.**normalize_bit_scores**(*dataframe*)

Creates two new columns of bit score normalized by the protein length.

Parameters
dataframe (*pandas*) –

Returns
pandas dataframe

pairpro.evaluate_input_cleaning.**verify_input_columns**(*dataframe*)

Asserts that columns we want to keep remain in the dataframe.

Parameters
dataframe (*pandas*) –

Returns
pandas dataframe

pairpro.evaluate_input_cleaning.**verify_protein_pairs**(*dataframe*)

Checks that input data has two protein sequences with simple assert statements. :param pandas dataframe:

Returns
pandas dataframe

4.4 pairpro.evaluate_model module

pairpro.evaluate_model.**evaluate_model**(*model*, *target: list*, *dataframe*)

Takes a trained model and test data and tests the model. Runs a single or multi-class Classifier depending on input.

Parameters

- **path** (*output*) – File path: str
- **model** – sklearn.neighbors.KNeighborsClassifier
- **target** – target for classifier (list)
- **dataframe** – pandas dataframe

Returns
Vector of predictions (numpy arrray) precision score (numpy array) results (csv)

4.5 pairpro.hmmer module

The following are various importable code for running HMMER either locally via pyhmmer or via Interpro's API

The local version runs in parrallel using joblib. The API version runs in parrallel using asyncio.

The local version is faster, but the API version is more accessible.

pairpro.hmmer.**calculate_jaccard_similarity**(*meso_accession_set*, *thermo_accession_set*)

Calculates the Jaccard similarity between meso_pid and thermo_pid pairs based on their accessions.

Jaccard similarity is defined as the size of the intersection divided by the size of the union of two sets.

Parameters

- **meso_accession_set** (*set*) – Set of meso_pid accessions.

- **thermo_accession_set** (*set*) – Set of thermo_pid accessions.

Returns
Jaccard similarity between the two sets of accessions. Returns 0 if the union is empty.

Return type
float

pairpro.hmmer.**calculate_similarity_API**(*file1: str*, *file2: str*, *threshold: float*) → Dict[str, Tuple[str, float]]

Calculates the Jaccard similarity score between each protein in file1 and file2, and returns a dictionary with query IDs as keys and a tuple indicating whether the score threshold was met and the Jaccard similarity score.

pairpro.hmmer.**calculate_similarity_user**(*file1: str*, *file2: str*, *threshold: float*) → Dict[str, Tuple[str, float]]

Calculates the Jaccard similarity score between each protein in file1 and file2, and returns a dictionary with query IDs as keys and a tuple indicating whether the score threshold was met and the Jaccard similarity score.

pairpro.hmmer.**find_jaccard_similarity_API**(*set1: set*, *set2: set*) → float

Calculates the Jaccard similarity score between two sets.

pairpro.hmmer.**get_file_pairs_API**(*directory_path*)

A quick silly function to get pairs

pairpro.hmmer.**get_file_pairs_user**(*directory_path*)

A quick silly function to get pairs

async pairpro.hmmer.**hmmerscanner**(*df: pandas.DataFrame*, *which: str*, *k: int*, *max_concurrent_requests: int*, *output_path: str*)

Scans multiple protein sequences using the HMMER API, asynchronously submitting and processing each request.

Parameters

- **df** (*pd.DataFrame*) – A DataFrame containing protein sequences.
- **which** (*str*) – The column name of the protein sequences.
- **k** (*int*) – The number of protein sequences to search.
- **max_concurrent_requests** (*int*) – The maximum number of concurrent requests to the HMMER API.
- **output_path** (*str*) – The output directory where the data will be stored.

Returns
A DataFrame containing the search results for all protein sequences.

Return type
pd.DataFrame

Raises
ValueError – If the number of sequences exceeds the limit of 1000.

pairpro.hmmer.**hmmpress_hmms**(*hmms_path*, *pfam_data_folder*)

Presses the HMMs in the given HMM database and stores the resulting files in a specified directory.

Parameters

- **hmms_path** (*str*) – Path to the HMM database.
- **pfam_data_folder** (*str*) – Path to the directory where the HMMs should be stored.

Returns
None

Notes

This function uses HMMER's hmmpress program to compress the HMMs in the given HMM database and stores the resulting files in the specified directory for faster access during future HMMER runs. If the specified directory does not exist, it will be created.

pairpro.hmmer.**local_hmmer_wrapper**(*chunk_index*, *chunked_inputs*, *press_path*, *hmm_path*, *out_dir*, *e_value: float = 1e-06*, *prefetch=True*, *cpu=1*, *wakeup=None*, *scan=True*, ***kwargs*)

A wrapping function that runs and parses pyhmmer in chunks.

Parameters

- **chunk_index** (*int*) – Number of sequence chunks.
- **chunked_inputs** (*pandas.DataFrame*) – DataFrame containing chunked PID inputs
- **press_path** (*str*) – Path to the pressed HMMs.
- **hmm_path** (*str*) – Path to the HMMs.
- **out_dir** (*str*) – Path to the output directory.
- **e_value** (*float, optional*) – E-value threshold. Defaults to 1e-6.
- **prefetch** (*bool, optional*) – Specifies whether to prefetch the HMMs. Defaults to True.
- **cpu** (*int, optional*) – Number of CPUs to use. Defaults to 1.
- **wakeup** (*int or None, optional*) – Delay in seconds before starting the execution. Default is None.
- **scan** (*bool, optional*) – Specifies whether to run hmmscan or hmmsearch. Defaults to True.

Returns
None

Notes

This function performs the following steps: 1. Converts string sequences to pyhmmer digital blocks. 2. Runs HMMER via pyhmmer with the provided sequences. 3. Parses the pyhmmer output and saves it to a CSV file.

The parsed pyhmmer output is saved in the directory specified by OUTPUT_DIR, with each chunk having its own separate output file named '{chunk_index}_output.csv'.

If the wakeup parameter is specified, the function will wait for the specified number of seconds before starting the execution.

pairpro.hmmer.**local_hmmer_wrapper_example**(*chunk_index*, *dbpath*, *chunked_pid_inputs*, *press_path*, *out_dir*, *wakeup=None*)

A wrapping function that runs and parses pyhmmer in chunks.

Parameters

- **chunk_index** (*int*) – Number of sequence chunks.
- **dbpath** (*stf*) – Path to the database.
- **chunked_pid_inputs** (*pandas.DataFrame*) – DataFrame containing chunked PID inputs.
- **press_path** (*str*) – Path to the pressed HMM database.

- **out_path** (*str*) – Path to directory where output will be saved.
- **wakeup** (*int or None, optional*) – Delay in seconds before starting the execution. Default is None.

Returns
None

Notes

This function performs the following steps: 1. Queries the database to get sequences only from chunked_pid_inputs. 2. Converts the query result to a DataFrame. 3. Converts string sequences to pyhmmer digital blocks. 4. Runs HMMER via pyhmmer with the provided sequences. 5. Parses the pyhmmer output and saves it to a CSV file.

The parsed pyhmmer output is saved in the directory specified by OUTPUT_DIR, with each chunk having its own separate output file named '{chunk_index}_output.csv'.

If the wakeup parameter is specified, the function will wait for the specified number of seconds before starting the execution.

pairpro.hmmer.**parse_function_csv_API**(*file_path: str*) → Dict[str, List[str]]

Parses the CSV file with protein IDs and their corresponding accession IDs and returns a dictionary with protein IDs as keys and accession IDs as values.

pairpro.hmmer.**parse_function_csv_user**(*file_path: str*) → Dict[str, List[str]]

Parses the CSV file with protein IDs and their corresponding accession IDs and returns a dictionary with protein IDs as keys and accession IDs as values.

pairpro.hmmer.**parse_pyhmmer**(*all_hits*, *chunk_query_ids*, *scanned: bool = True*)

Parses the TopHit pyhmmer objects, extracting query and accession IDs, and saves them to a DataFrame.

Parameters

- **all_hits** (*list*) – A list of TopHit objects from pyhmmer.
- **chunk_query_ids** (*list*) – A list of query IDs from the chunk.
- **scanned** (*bool, optional*) – Specifies whether the sequences were scanned or searched. Defaults to True.

Returns
A DataFrame containing the query and accession IDs.

Return type
pandas.DataFrame

Notes

This function iterates over each protein hit in the provided list of TopHit objects and extracts the query and accession IDs. The resulting query and accession IDs are then saved to a DataFrame. Any query IDs that are missing from the parsed hits will be added to the DataFrame with a placeholder value indicating no accession information.

pairpro.hmmer.**parse_pyhmmer_user**(*all_hits*, *chunk_pair_ids*)

Parses the TopHit pyhmmer objects, extracting query and accession IDs, and saves them to a DataFrame.

Parameters

- **all_hits** (*list*) – A list of TopHit objects from pyhmmer.

- **chunk_pair_ids** (*list*) – A list of query IDs from the chunk.

Returns
A DataFrame containing the pair and accession IDs.

Return type
pandas.DataFrame

Notes

This function iterates over each protein hit in the provided list of TopHit objects and extracts the query and accession IDs. The resulting query and accession IDs are then saved to a DataFrame. Any pair IDs that are missing from the parsed hits will be added to the DataFrame with a placeholder value indicating no accession information.

pairpro.hmmer.**prefetch_targets**(*hmms_path: str*)

Prefetch HMM profiles from a given HMM database.

Parameters
hmms_path (*str*) – Path to the pressed HMM database.

Returns
The HMM profiles loaded from the database.

Return type
targets (pyhmmer.plan7.OptimizedProfileBlock)

pairpro.hmmer.**preprocess_accessions**(*meso_accession: str*, *thermo_accession: str*)

Preprocesses meso_accession and thermo_accession by converting them to sets.

Parameters

- **meso_accession** (*str*) – Meso accession string separated by ';'.
- **thermo_accession** (*str*) – Thermo accession string separated by ';'.

Returns
A tuple containing the preprocessed meso_accession and thermo_accession sets.

Return type
tuple

pairpro.hmmer.**process_pairs_table**(*conn*, *dbname*, *chunk_size: int*, *output_directory*, *jaccard_threshold*)

Processes the pairs table, calculates Jaccard similarity, and generates output CSV.

Parameters

- **conn** – Path to the database file.
- **dbname** (*str*) – Name of the database.
- **chunk_size** (*int*) – Size of each query chunk to fetch from the database.
- **output_directory** (*str*) – Directory path to save the output CSV files.
- **jaccard_threshold** (*float*) – Threshold value for Jaccard similarity.

Returns
None

async pairpro.hmmer.**process_response**(*semaphore*, *sequence*, *response*, *client*, *pair_id*, *max_retries=3*)

Processes the response received from the HMMER API, including retrying requests that have failed.

Parameters

- **semaphore** (*asyncio.Semaphore*) – An object that controls concurrent request submission, helping to avoid server overload.
- **sequence** (*str*) – The protein sequence associated with the response.
- **response** (*httpx.Response*) – The response received from the HMMER API.
- **client** (*httpx.AsyncClient*) – An HTTP client for sending subsequent requests.
- **pair_id** (*int*) – The protein ID associated with the sequence.
- **max_retries** (*int, optional*) – The maximum number of retries for failed requests. Defaults to 3.

Returns

A DataFrame containing the search results for the protein sequence, or None if an error occurred.

Return type

pd.DataFrame or None

Raises

- **KeyError** – If expected key is not found in the response.
- **json.JSONDecodeError** – If JSON decoding fails.

pairpro.hmmer.**run_hmmerscanner**(*df: pandas.DataFrame*, *which: str*, *k: int*, *max_concurrent_requests: int*, *output_path: str*)

Runs the asynchronous HMMER scanning operation in a new event loop.

Parameters

- **df** (*pd.DataFrame*) – A DataFrame containing protein sequences.
- **which** (*str*) – The column name of the protein sequences.
- **k** (*int*) – The number of protein sequences to search.
- **max_concurrent_requests** (*int*) – The maximum number of concurrent requests to the HMMER API.
- **output_path** (*str*) – The output directory where the data will be stored. (like '/Users/amin/ValidProt/data/')

Returns

A DataFrame containing the search results for all protein sequences.

Return type

pd.DataFrame

Raises

- **nest_asyncio.NestingError** – If the event loop is already running.
- **Any exceptions raised by hmmerscanner function.** –

pairpro.hmmer.**run_pyhmmer**(*seqs: pyhmmer.easel.DigitalSequenceBlock | str*, *hmms_path: str = None*, *pressed_path: str = None*, *prefetch: bool | pyhmmer.plan7.OptimizedProfileBlock = False*, *output_file: str = None*, *cpu: int = 4*, *scan: bool = True*, *eval_con: float = 1e-10*, ***kwargs*)

Run HMMER's hmmscan program on a set of input sequences using HMMs from a database.

Parameters

- **seqs** (*pyhmmer.easel.DigitalSequenceBlock*) – Digital sequence block of input sequences.
- **hmms_path** (*str*) – Path to the HMM database.
- **pressed_path** (*str*) – Path to the pressed HMM database.
- **prefetch** (*bool, optional*) – Specifies whether to use prefetching mode for HMM storage. Defaults to False.
- **output_file** (*str, optional*) – Path to the output file if the user wants to write the file. Defaults to None.
- **cpu** (*int, optional*) – The number of CPUs to use. Defaults to 4.
- **scan** (*bool, optional*) – Specifies whether to run hmmscan or hmmsearch. Defaults to True.
- **eval_con** (*float, optional*) – E-value threshold for domain reporting. Defaults to 1e-10.

Returns

If output_file is specified, the function writes the results to a domtblout file and returns the file path. Otherwise, it returns a list of pyhmmer.plan7.TopHits objects.

Return type

Union[pyhmmer.plan7.TopHits, str]

Notes

This function runs HMMER's hmmscan program on a set of input sequences using HMMs from a given database. The function supports two modes: normal mode and prefetching mode. In normal mode, the HMMs are pressed and stored in a directory before execution. In prefetching mode, the HMMs are kept in memory for faster search.

pairpro.hmmer.**save_to_digital_sequences**(*dataframe: pandas.DataFrame*)

Save protein sequences from a DataFrame to a digital sequence block.

Parameters

dataframe (*pd.DataFrame*) – DataFrame containing PIDs (Protein IDs) and sequences.

Returns

A digital sequence block containing the converted sequences.

Return type

pyhmmer.easel.DigitalSequenceBlock

pairpro.hmmer.**save_to_digital_sequences_user_query**(*dataframe: pandas.DataFrame*)

Save protein sequences from a DataFrame to a digital sequence block.

Parameters

dataframe (*pd.DataFrame*) – DataFrame containing pair_id (Protein pair IDs) and sequences.

Returns

A digital sequence block containing the converted sequences.

Return type

pyhmmer.easel.DigitalSequenceBlock

pairpro.hmmer.**save_to_digital_sequences_user_subject**(*dataframe: pandas.DataFrame*)

Save protein sequences from a DataFrame to a digital sequence block.

Parameters
dataframe (*pd.DataFrame*) – DataFrame containing pair_id (Protein pair IDs) and sequences.

Returns
A digital sequence block containing the converted sequences.

Return type
pyhmmer.easel.DigitalSequenceBlock

async pairpro.hmmer.**send_request**(*semaphore*, *sequence*, *client*)

Asynchronously sends a POST request to the HMMER API, submitting a protein sequence for analysis.

Parameters

- **semaphore** (*asyncio.Semaphore*) – An object that controls concurrent request submission, helping to avoid server overload.
- **sequence** (*str*) – The protein sequence that is to be analyzed and included in the body of the POST request.
- **client** (*httpx.AsyncClient*) – An HTTP client for sending the request.

Returns
Response received from the HMMER API.

Return type
httpx.Response

Raises

- **httpx.HTTPStatusError** – If the HTTP request returned a status code that denotes an error.
- **httpx.TimeoutException** – If the request times out.

pairpro.hmmer.**user_local_hmmer_wrapper_query**(*chunk_index*, *press_path*, *sequences*, *out_dir*)

TODO

pairpro.hmmer.**user_local_hmmer_wrapper_subject**(*chunk_index*, *press_path*, *sequences*, *out_dir*)

TODO

pairpro.hmmer.**write_function_output_API**(*output_dict: Dict[str, Tuple[str, float]]*, *output_file: str*)

Writes a dictionary of protein pair IDs and functional tuple values to a CSV file.

Args: output_dict : Dict[str, Tuple[str, float]]

A dictionary of protein pair IDs and functional tuple values

output_file
[str] File path to write the output CSV file

4.6 pairpro.preprocessing module

This package builds the PairProphet database from learn2thermDB.

Functions:

connect_df: Establishes connection to DuckDB database using local or
remote input path. Reports time to connection.

build_pairpro: Constructs pairprophet database from learn2therm database.

pairpro.preprocessing.**build_pairpro**(*con*, *out_db_path*, *min_ogt_diff: int = 20*, *min_16s: int = 1300*)

Converts learn2therm DuckDB database into a DuckDB database for PairProphet by adding filtered and constructed tables. Ensure at lease 20 GB of free disk space and 30 GB of system memory are available before running on the full database.

Parameters

- **con** (*duckdb.DuckDBPyConnection*) – DuckDB connection object. Links script to DuckDB SQL database.
- **out_db_path** (*str*) – Path to PairProphet output database file.
- **min_ogt_diff** (*int*) – Cutoff for minimum difference in optimal growth temperature between thermophile and mesophile pairs. Default 20 deg C.
- **min_16s** (*int*) – Cutoff for minimum 16S read length for taxa. Default 1300 bp. Filters out organisms with poor or incomplete 16S sequencing.

Returns
None. Database object is modified in place.

Raises

- **ValueError** – Optimal growth temperature difference must be positive.
- **ValueError** – Minimum 16S sequence read is 1 bp.
- **AttributeError** – Database must be in the learn2therm format.

pairpro.preprocessing.**connect_db**(*path: str*, *empty=False*)

Runs duckdb.connect() function on database path. Returns a duckdb.DuckDBPyConnection object and prints execution time.

Parameters
path (*str*) – Path to DuckDB database file containing learn2therm.

Returns

A DuckDB connection object linking
script to learn2therm database.

Return type
con (duckdb.DuckDBPyConnection)

Raises
AttributeError – Input database contains no tables.

4.7 pairpro.structures module

This module takes in a pandas dataframe containing Uniprot IDs and PDB IDs, download the pdb files and run FATCAT for structural alignment purpose. Returns a Boolean for structure similarity. If no structures were found for proteins, that pair is dropped in the output file.

pairpro.structures.**compare_fatcat**(*p1_file*, *p2_file*, *pdb_dir*, *pair_id*)

Compares two protein structures using FATCAT.

Parameters

- **p1_file** (*str*) – The path to the first protein structure file.
- **p2_file** (*str*) – The path to the second protein structure file.
- **pdb_dir** (*str*) – The directory containing the structure files.
- **pair_id** (*str*) – The ID of the protein pair.

Returns
A dictionary containing the pair ID and the p-value.

Return type
dict

async pairpro.structures.**download_af**(*row*, *u_column*, *pdb_dir*)

Downloads AlphaFold files for a given row asynchronously.

Parameters

- **row** (*pd.Series*) – The row containing the data for the download.
- **u_column** (*str*) – The column name for the UniProt ID.
- **pdb_dir** (*str*) – The directory to save the downloaded files.

Returns
True if the download is successful, False otherwise.

Return type
bool

async pairpro.structures.**download_aff**(*session*, *url*, *filename*)

Downloads a file asynchronously using an HTTP session.

Parameters

- **session** (*httpx.AsyncClient*) – An HTTP session for making requests.
- **url** (*str*) – The URL of the file to download.
- **filename** (*str*) – The name of the file to save.

Returns
True if the file is successfully downloaded, False otherwise.

Return type
bool

pairpro.structures.**download_pdb**(*df*, *pdb_column*, *pdb_dir*)

Downloads PDB files for the given DataFrame based on PDB IDs.

Parameters

- **df** (*pd.DataFrame*) – The DataFrame containing the PDB IDs.

- **pdb_column** (*str*) – The column name for the PDB ID.
- **pdb_dir** (*str*) – The directory to save the downloaded files.

Returns: pdb files containing structural information.

pairpro.structures.**download_structure**(*df*, *pdb_column*, *u_column*, *pdb_dir*)

Downloads structure files for a DataFrame using AlphaFold and PDB.

Parameters

- **df** (*pd.DataFrame*) – The DataFrame containing the data for the downloads.
- **pdb_column** (*str*) – The column name for the PDB ID.
- **u_column** (*str*) – The column name for the UniProt ID.
- **pdb_dir** (*str*) – The directory to save the downloaded files.

Returns: pdb files containing structural information.

pairpro.structures.**process_row**(*row*, *pdb_dir*)

Processes a row of a DataFrame to compare protein structures using FATCAT.

Parameters

- **row** (*pd.Series*) – The row containing the data for the comparison.
- **pdb_dir** (*str*) – The directory containing the structure files.

Returns
A dictionary containing the pair ID and the p-value.

Return type
dict

pairpro.structures.**run_download_af_all**(*df*, *pdb_column*, *u_column*, *pdb_dir*)

Runs the asynchronous download of AlphaFold files for all rows in a DataFrame.

Parameters

- **df** (*pd.DataFrame*) – The DataFrame containing the data for the downloads.
- **pdb_column** (*str*) – The column name for the PDB ID.
- **u_column** (*str*) – The column name for the UniProt ID.
- **pdb_dir** (*str*) – The directory to save the downloaded files.

Returns
pdb files containing structural information.

Return type
files

pairpro.structures.**run_fatcat_dict_job**(*df*, *pdb_dir*, *file*)

Runs the FATCAT comparison job on a DataFrame and saves the results to a file.

Parameters

- **df** (*pd.DataFrame*) – The DataFrame containing the data for the comparison.
- **pdb_dir** (*str*) – The directory containing the structure files.
- **file** (*str*) – The path to the output file.

Returns
a csv file containing the pair ID and the p-value.

Return type
csv

4.8 pairpro.train_val_classification module

This module takes in a pandas dataframe from c5_input_cleaning and runs it through a RandomForestClassifier model from scitkit learn. Returns a Boolean prediction for protein pair functionality.

pairpro.train_val_classification.**plot_model**(*model*, *val_X*, *val_y*)

Takes a test classifier model and plots the confusion matrix.

Parameters

- **model** – sklearn.neighbors.RandomForestClassifier
- **test_X** – numpy array
- **test_y** – numpy array

Returns
Confusion predictions vs. observations Model score

pairpro.train_val_classification.**rf_wrapper**(*dataframe*, *target*)

Takes a test classifier model and plots the confusion matrix.

Parameters
dataframe – Pandas dataframe

Returns
Target feature predictions Parity plot

pairpro.train_val_classification.**train_model**(*dataframe*, *columns=[]*, *target=[]*)

Takes dataframe and splits it into a training and testing set. Trains a RF Classifier with data.

Parameters

- **dataframe** – Pandas dataframe
- **columns** – list of strings, representing input features
- **target** – list of strings, representing target feature(s)

Returns
Sk-learn model object train data (features) train data (target) validation data (features) validation data (target)

pairpro.train_val_classification.**train_model_structure**(*dataframe*, *columns=[]*, *target=[]*)

Takes dataframe and splits it into a training and testing set. Trains a RF Classifier with data.

Parameters

- **dataframe** – Pandas dataframe
- **columns** – list of strings, representing input features
- **target** – list of strings, representing target feature(s)

Returns
Sk-learn model object train data (features) train data (target) validation data (features) validation data (target)

pairpro.train_val_classification.**validate_model**(*model*, *val_X*, *val_y*)

Takes a trained model and test data and tests the model.

Parameters

- **model** – sklearn.neighbors.KNeighborsClassifier
- **test_X** – numpy array
- **test_y** – numpy array

Returns

Vector of predictions based on the model (numpy array)
Precision score of model

4.9 pairpro.train_val_featuregen module

This module utilizes iFeatureOmega, a feature generation package for proteins and nucleic acids.

pairpro.train_val_featuregen.**clean_new_dataframe**(*dataframe*)

Asserts that artifact columns generated from iFeatureOmega such as "index" are removed.

Parameters
dataframe (*Pandas*) –

Returns
Pandas dataframe

pairpro.train_val_featuregen.**create_new_dataframe**(*dataframe*, *output_files: list*, *descriptors=[]*)

Creates new dataframe with descriptors added.

Parameters

- **dataframe** (*Pandas*) –
- **strings** (*list of descriptors as*) –
- **name.** (*output file*) –

Returns
Dataframe including vector(s) of descriptors (pandas dataframe)

pairpro.train_val_featuregen.**get_fasta_from_dataframe**(*dataframe*, *output_file_a: str*, *output_file_b: str*)

Generates fasta file type from pandas dataframe.

Parameters

- **Dataframe** (*pandas dataframe*) –
- **files** (*Names of output fasta*) –

Returns
Two fasta files with protein sequences and pair_id

pairpro.train_val_featuregen.**get_protein_descriptors**(*fasta_file: str*, *descriptors=[]*)

Generates features from a protein sequence.

Parameters
sequences (*Fasta file with amino acid*) –

Returns
Vector of descriptors (numpy array)

4.10 pairpro.train_val_input_cleaning module

This module takes a dataframe from the data scraping component and cleans it so that it can be passed through a machine learning algorithm.

pairpro.train_val_input_cleaning.**check_input_nans**(*dataframe*)

Checks for NaN values in input dataframe. Removes rows with NaN values present. :param pandas dataframe:

Returns
pandas dataframe

pairpro.train_val_input_cleaning.**check_input_type**(*dataframe*)

Takes in input dataframe and asserts that it is the correct data type.

Parameters
dataframe (*pandas*) –

Returns
pandas dataframe

pairpro.train_val_input_cleaning.**clean_input_columns**(*dataframe*)

Cleans out columns that are not in a predefined list of features.

Parameters
dataframe (*pandas*) –

Returns
pandas dataframe

pairpro.train_val_input_cleaning.**input_cleaning_wrapper**(*dataframe*, *structure*)

Takes in a pandas dataframe and runs it through each of the cleaning and verification steps. :param pandas dataframe:

Returns
pandas dataframe

pairpro.train_val_input_cleaning.**normalize_bit_scores**(*dataframe*)

Creates two new columns of bit score normalized by the protein length.

Parameters
dataframe (*pandas*) –

Returns
pandas dataframe

pairpro.train_val_input_cleaning.**verify_input_columns**(*dataframe*)

Asserts that columns we want to keep remain in the dataframe.

Parameters
dataframe (*pandas*) –

Returns
pandas dataframe

pairpro.train_val_input_cleaning.**verify_protein_pairs**(*dataframe*)

Checks that input data has two protein sequences with simple assert statements. :param pandas dataframe:

Returns
pandas dataframe

4.11 pairpro.train_val_wrapper module

Wrapper functions for all of the machine learning component.

pairpro.train_val_wrapper.**train_val_wrapper**(*dataframe*, *target*, *structure=False*, *features=False*)

Takes dataframe and runs it through cleaning script. Generates features with iFeatureOmegaCLI. Passes result through RF Classifier model.

Parameters

- **Dataframe** (*pandas dataframe*) –
- **iFeatureOmega** (*Features from*) –

Returns
Vector of predictions (numpy arrray) Parity plot Model score

4.12 pairpro.user_blast module

To do: Raise exception for invalid inputs, try capitalization before removing rows

pairpro.user_blast.**gap_compressed_percent_id**(*n_matches*, *n_gaps*, *n_columns*, *n_comp_gaps*)

Calculates the percent id with compressed gaps.

Parameters

- **n_matches** (*int*) – Number of matches in match columns
- **n_gaps** (*int*) – Number of gaps in match columns
- **n_columns** (*int*) – Total number of alignment match columns
- **n_compressed_gaps** (*int*) – Number of compressed gaps in match columns

Returns
n_matches / (n_columns - n_gaps + n_comp_gaps)

pairpro.user_blast.**get_matches_gaps**(*query*, *subject*)

Parses sequence alignment text to calculate the number of matches, gaps, compressed gaps, and total columns.

Parameters

- **query** (*str*) – Query aligned sequence.
- **subject** (*str*) – Subject aligned sequence.

Returns

Number of matching amino acids in the sequence
alignment.

n_gaps (int): Total number of gaps across both aligned sequences. n_columns (int): Length of the aligned query sequence. n_comp_gaps (int): Number of compressed gaps.

Return type
n_matches (int)

pairpro.user_blast.**make_blast_df**(*df*, *mode='local'*, *path='./data/blast_db.db'*)

This function generates pairwise alignment scores for a set of protein sequences.

Parameters

- **df** (*pandas.core.DataFrame*) – A 2-column DataFrame containing the query and subject sequences for alignment.
- **mode** (*str*) – Alignment type is 'local' or 'global'. Default: 'local'.

Returns

A dataframe with the input sequence
pairs, associated id values, and alignment scores.

Return type
blast_df (pandas.core.DataFrame)

pairpro.user_blast.**sequence_validate**(*seq*, *alph*)

Makes sure sequence complies with alphabet.

Parameters

- **seq** (*int*) – Number of matches in match columns
- **alph** (*int*) – Number of gaps in match columns

Returns
True if sequence is valid, False if not

Return type
(bool)

4.13 pairpro.utils module

The following is importable random utilites. You will find: - logger function - pairwise sequence builder

pairpro.utils.**make_pairs**(*seq1_list*, *seq2_list*, *seq1_name='seq1'*, *seq2_name='seq2'*, *csv_path='./paired_seqs.csv'*, *save=True*)

Function for building a combinatorial set of sequences from two lists.

Parameters

- **seq1_list** (*list*) – List of protein sequence strings
- **seq2_list** (*list*) – List of protein sequence strings
- **seq1_name** (*str*) – Column name for first sequence column
- **seq2_name** (*str*) – Column name for second sequence column
- **csv_path** (*str*) – Path for saved .csv file
- **save** (*bool*) – Saves paired sequences as .csv when True

Returns
A dataframe with rows as all possible sequence pairs.

Return type
combined_df (pd.DataFrame)

pairpro.utils.**start_logger_if_necessary**(*logger_name: str*, *log_file: str*, *log_level*, *filemode: str = 'a'*, *worker: bool = False*)

Quickly configure and return a logger that respects parallel processes.

Parameters

- **logger_name** (*str*) – name of logger to start or retrieve
- **log_file** (*str*) – path to file to log to
- **log_level** – log level to respect
- **worker** (*str*) – name of worker using this logger
- **filemode** (*str*) – mode to apply to log file eg "a" for append

CHAPTER

FIVE

ACKNOWLEDGEMENTS

We would like to thank Prof. David Beck and Evan Komp for sponsoring and mentoring our research project. PairProphet was developed by the following team of University of Washington graduate students:

Humood Alanzi is a second year M.S. student in the Beck Group, Department of Chemical Engineering

Ryan Francis is a third year Ph.D student in the Deforest Group, Department of Chemical Engineering

Logan Roberts is a first year M.S. student, Department of Chemical Engineering

Chau Vuong is a third year Ph.D student in the Ailion Group, Department of Biochemistry

Amin Mosallenejad is a first year Ph.D student in the Ratner Group, Department of Chemical Engineering

CHAPTER

SIX

REFERENCES

Literature

1) PyHMMER (Larralde et al., 2022), a Python library binding to HMMER (Eddy, 2011).

2) Cock PA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, Friedberg I, Hamelryck T, Kauff F, Wilczynski B and de Hoon MJL (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics.

3) Korendovych, Ivan V., and William F. DeGrado. "De Novo Protein Design, a Retrospective." Quarterly Reviews of Biophysics 53 (2020): e3. https://doi.org/10.1017/S0033583519000131.

4) Sonnhammer, Erik L.L., Sean R. Eddy, and Richard Durbin. "Pfam: A Comprehensive Database of Protein Domain Families Based on Seed Alignments." Proteins: Structure, Function, and Genetics 28, no. 3 (July 1997): 405–20. https://doi.org/10.1002/(SICI)1097-0134(199707)28:3<405::AID-PROT10>3.0.CO;2-L.

5) Gromiha, M. Michael, Raju Nagarajan, and Samuel Selvaraj. "Protein Structural Bioinformatics: An Overview." In Encyclopedia of Bioinformatics and Computational Biology, 445–59. Elsevier, 2019. https://doi.org/10.1016/B978-0-12-809633-8.20278-1.

6) Baker, David, and Andrej Sali. "Protein Structure Prediction and Structural Genomics." Science 294, no. 5540 (October 5, 2001): 93–96. https://doi.org/10.1126/science.1065659.

7) Zhen Chen, Xuhan Liu, Pei Zhao, Chen Li, Yanan Wang, Fuyi Li, Tatsuya Akutsu, Chris Bain, Robin B. Gasser, Zuoren Yang*, Lukasz Kurgan*, Jiangning Song*, iFeatureOmega – an integrative platform for the feature engineering, visualization and analysis of features from molecular sequence, structural and ligand data sets. Nucleic Acids Research , 2022.

8) Pearson, William R. "An Introduction to Sequence Similarity ('Homology') Searching." Current Protocols in Bioinformatics 42, no. 1 (June 2013). https://doi.org/10.1002/0471250953.bi0301s42.

9) Zhanwen Li, Lukasz Jaroszewski, Mallika Iyer, Mayya Sedova and Adam Godzik. “FATCAT 2.0: towards a better understanding of the structural diversity of proteins” Nucleic Acids Res., May 2020. gkaa443, https://doi.org/10.1093/nar/gkaa443

Python Packages

Pyhmmer

FatCat

Biopython

iFeatureOmega

CHAPTER

SEVEN

OUR PLEDGE

We, as members, contributors, and leaders, pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes and learning from the experience
- Focusing on what is best for the overall community, not just for individuals

Examples of unacceptable behavior include:

- The use of sexualized language or imagery and sexual attention or advances of any kind
- Trolling, insulting, or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as physical or email addresses, without explicit permission
- Other conduct that could reasonably be considered inappropriate in a professional setting

Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior. They will take appropriate and fair corrective action in response to any behavior they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that do not align with this Code of Conduct. They will communicate reasons for moderation decisions when appropriate.

Scope

This Code of Conduct applies within all community spaces and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official email address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at halanzi@uw.edu. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

Enforcement Guidelines

Community leaders will follow these enforcement guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. **Correction**
    - Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.
    - Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.
2. **Warning**
    - Community Impact: A violation through a single incident or series of actions.
    - Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.
3. **Temporary Ban**
    - Community Impact: A serious violation of community standards, including sustained inappropriate behavior.
    - Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. **Permanent Ban** - Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals. - Consequence: A permanent ban from any sort of public interaction within the community.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available here

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ here. Translations are available here

PYTHON MODULE INDEX

p

pairpro.dev_tools, 10
pairpro.evaluate_input_cleaning, 10
pairpro.evaluate_model, 11
pairpro.hmmer, 11
pairpro.preprocessing, 19
pairpro.structures, 20
pairpro.train_val_classification, 22
pairpro.train_val_featuregen, 23
pairpro.train_val_input_cleaning, 24
pairpro.train_val_wrapper, 25
pairpro.user_blast, 25
pairpro.utils, 26

INDEX

B

build_pairpro() (*in module pairpro.preprocessing*), 19
build_sample_l2t() (*in module pairpro.dev_tools*), 10

C

calculate_jaccard_similarity() (*in module pairpro.hmmer*), 11
calculate_similarity_API() (*in module pairpro.hmmer*), 12
calculate_similarity_user() (*in module pairpro.hmmer*), 12
check_input_nans() (*in module pairpro.evaluate_input_cleaning*), 10
check_input_nans() (*in module pairpro.train_val_input_cleaning*), 24
check_input_type() (*in module pairpro.evaluate_input_cleaning*), 10
check_input_type() (*in module pairpro.train_val_input_cleaning*), 24
clean_input_columns() (*in module pairpro.evaluate_input_cleaning*), 10
clean_input_columns() (*in module pairpro.train_val_input_cleaning*), 24
clean_new_dataframe() (*in module pairpro.train_val_featuregen*), 23
compare_fatcat() (*in module pairpro.structures*), 20
connect_db() (*in module pairpro.preprocessing*), 19
create_new_dataframe() (*in module pairpro.train_val_featuregen*), 23

D

download_af() (*in module pairpro.structures*), 20
download_aff() (*in module pairpro.structures*), 20
download_pdb() (*in module pairpro.structures*), 20
download_structure() (*in module pairpro.structures*), 21

E

evaluate_model() (*in module pairpro.evaluate_model*), 11

F

find_jaccard_similarity_API() (*in module pairpro.hmmer*), 12

G

gap_compressed_percent_id() (*in module pairpro.user_blast*), 25
get_fasta_from_dataframe() (*in module pairpro.train_val_featuregen*), 23
get_file_pairs_API() (*in module pairpro.hmmer*), 12
get_file_pairs_user() (*in module pairpro.hmmer*), 12
get_matches_gaps() (*in module pairpro.user_blast*), 25
get_protein_descriptors() (*in module pairpro.train_val_featuregen*), 23

H

hmmerscanner() (*in module pairpro.hmmer*), 12
hmmpress_hmms() (*in module pairpro.hmmer*), 12

I

input_cleaning_wrapper() (*in module pairpro.evaluate_input_cleaning*), 10
input_cleaning_wrapper() (*in module pairpro.train_val_input_cleaning*), 24

L

local_hmmer_wrapper() (*in module pairpro.hmmer*), 13
local_hmmer_wrapper_example() (*in module pairpro.hmmer*), 13

M

make_blast_df() (*in module pairpro.user_blast*), 26
make_pairs() (*in module pairpro.utils*), 26
module
    pairpro.dev_tools, 10
    pairpro.evaluate_input_cleaning, 10
    pairpro.evaluate_model, 11
    pairpro.hmmer, 11

pairpro.preprocessing, 19
pairpro.structures, 20
pairpro.train_val_classification, 22
pairpro.train_val_featuregen, 23
pairpro.train_val_input_cleaning, 24
pairpro.train_val_wrapper, 25
pairpro.user_blast, 25
pairpro.utils, 26

N

normalize_bit_scores() (*in module pairpro.evaluate_input_cleaning*), 11
normalize_bit_scores() (*in module pairpro.train_val_input_cleaning*), 24

P

pairpro.dev_tools
module, 10
pairpro.evaluate_input_cleaning
module, 10
pairpro.evaluate_model
module, 11
pairpro.hmmer
module, 11
pairpro.preprocessing
module, 19
pairpro.structures
module, 20
pairpro.train_val_classification
module, 22
pairpro.train_val_featuregen
module, 23
pairpro.train_val_input_cleaning
module, 24
pairpro.train_val_wrapper
module, 25
pairpro.user_blast
module, 25
pairpro.utils
module, 26
parse_function_csv_API() (*in module pairpro.hmmer*), 14
parse_function_csv_user() (*in module pairpro.hmmer*), 14
parse_pyhmmer() (*in module pairpro.hmmer*), 14
parse_pyhmmer_user() (*in module pairpro.hmmer*), 14
plot_model() (*in module pairpro.train_val_classification*), 22
prefetch_targets() (*in module pairpro.hmmer*), 15
preprocess_accessions() (*in module pairpro.hmmer*), 15
process_pairs_table() (*in module pairpro.hmmer*), 15
process_response() (*in module pairpro.hmmer*), 15
process_row() (*in module pairpro.structures*), 21

R

rf_wrapper() (*in module pairpro.train_val_classification*), 22
run_download_af_all() (*in module pairpro.structures*), 21
run_fatcat_dict_job() (*in module pairpro.structures*), 21
run_hmmerscanner() (*in module pairpro.hmmer*), 16
run_pyhmmer() (*in module pairpro.hmmer*), 16

S

save_to_digital_sequences() (*in module pairpro.hmmer*), 17
save_to_digital_sequences_user_query() (*in module pairpro.hmmer*), 17
save_to_digital_sequences_user_subject() (*in module pairpro.hmmer*), 17
send_request() (*in module pairpro.hmmer*), 18
sequence_validate() (*in module pairpro.user_blast*), 26
start_logger_if_necessary() (*in module pairpro.utils*), 26

T

train_model() (*in module pairpro.train_val_classification*), 22
train_model_structure() (*in module pairpro.train_val_classification*), 22
train_val_wrapper() (*in module pairpro.train_val_wrapper*), 25

U

user_local_hmmer_wrapper_query() (*in module pairpro.hmmer*), 18
user_local_hmmer_wrapper_subject() (*in module pairpro.hmmer*), 18

V

validate_model() (*in module pairpro.train_val_classification*), 22
verify_input_columns() (*in module pairpro.evaluate_input_cleaning*), 11
verify_input_columns() (*in module pairpro.train_val_input_cleaning*), 24
verify_protein_pairs() (*in module pairpro.evaluate_input_cleaning*), 11
verify_protein_pairs() (*in module pairpro.train_val_input_cleaning*), 24

W

write_function_output_API() (*in module pairpro.hmmer*), 18