aide_predict

Release v0

Evan Komp

Oct 23, 2024

CONTENTS:

CHAPTER

ONE

AIDE_PREDICT

1.1 aide_predict package

1.1.1 Subpackages

aide_predict.aquisition_functions package

Submodules

aide_predict.aquisition_functions.aquisition_function module

Module contents

aide_predict.bespoke_models package

Subpackages

aide_predict.bespoke_models.embedders package

Submodules

aide_predict.bespoke_models.embedders.esm2 module

- Author: Evan Komp
- Created: 7/5/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

ESM2 language model self supervised embeddings.

class aide_predict.bespoke_models.embedders.esm2.**ESM2Embedding**(*metadata_folder: str | None = None*, *model_checkpoint: str = 'esm2_t6_8M_UR50D'*, *layer: int = -1*, *positions: List[int] | None = None*, *flatten: bool = False*, *pool: bool | None = None*, *batch_size: int = 32*, *device: str = 'cpu'*, *wt: str | ProteinSequence | None = None*, ***kwargs*)

Bases: *CacheMixin*, *PositionSpecificMixin*, *CanHandleAlignedSequencesMixin*, *ProteinModelWrapper*

A protein sequence embedder that uses the ESM2 model to generate embeddings.

This class wraps the ESM2 model to provide embeddings for protein sequences. It can handle both aligned and unaligned sequences and allows for retrieving embeddings from a specific layer of the model.

model_checkpoint

The name of the ESM2 model checkpoint to use.

Type
str

layer

The layer from which to extract embeddings (-1 for last layer).

Type
int

positions

Specific positions to encode. If None, all positions are encoded.

Type
Optional[List[int]]

pool

Whether to pool the encoded vectors across positions.

Type
bool

flatten

Whether to flatten the output array.

Type
bool

batch_size

The batch size for processing sequences.

Type
int

device

The device to use for computations ('cuda' or 'cpu').

Type
str

get_feature_names_out(*input_features: List[str] | None = None*) → List[str]

Get output feature names for transformation.

Parameters
input_features (*Optional[List[str]]*) – Ignored. Present for API consistency.

Returns
Output feature names.

Return type
List[str]

set_fit_request(*, *force: bool | None | str = '$UNCHANGED$'*) → *ESM2Embedding*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `force` parameter in `fit`.

Returns
self – The updated object.

Return type
object

aide_predict.bespoke_models.embedders.msa_transformer module

- Author: Evan Komp
- Created: 7/8/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

class aide_predict.bespoke_models.embedders.msa_transformer.**MSATransformerEmbedding**(*metadata_folder: str | None = None, layer: int = -1, positions: List[int] | None = None, flatten: bool = False, pool: bool = False, batch_size: int = 32, n_msa_seqs: int = 360, device: str = 'cpu', wt: str | ProteinSequence | None = None*)

Bases: *CacheMixin*, *PositionSpecificMixin*, *RequiresMSAMixin*, *ProteinModelWrapper*

A protein sequence embedder that uses the MSA Transformer model to generate embeddings.

This class wraps the MSA Transformer model to provide embeddings for protein sequences. It requires fixed-length sequences and an MSA for fitting. At prediction time, it can handle sequences of the same length as the MSA used for fitting.

layer

The layer from which to extract embeddings (-1 for last layer).

Type
int

positions

Specific positions to encode. If None, all positions are encoded.

Type
Optional[List[int]]

pool

Whether to pool the encoded vectors across positions.

Type
bool

flatten

Whether to flatten the output array.

Type
bool

batch_size

The batch size for processing sequences.

Type
int

device

The device to use for computations ('cuda' or 'cpu').

Type
str

get_feature_names_out(*input_features: List[str] | None = None*) → List[str]

Get output feature names for transformation.

Parameters
input_features (*Optional[List[str]]*) – Ignored. Present for API consistency.

Returns
Output feature names.

Return type
List[str]

set_fit_request(*, *force: bool | None | str = '$UNCHANGED$'*) → *MSATransformerEmbedding*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `force` parameter in `fit`.

Returns
self – The updated object.

Return type
object

aide_predict.bespoke_models.embedders.ohe module

- Author: Evan Komp
- Created: 7/5/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Two classes: OneHotProteinEmbedding for fixed length sequences and OneHotAlignmentEmbedding which will dynamically align sequences to reference alignment before encoding.

class aide_predict.bespoke_models.embedders.ohe.**OneHotAlignedEmbedding**(*metadata_folder: str*, *wt: str | ProteinSequence | None = None*, *positions: List[int] | None = None*, *flatten: bool = True*, *pool: bool = False*)

Bases: *ShouldRefitOnSequencesMixin*, *PositionSpecificMixin*, *RequiresMSAMixin*, *CanHandleAlignedSequencesMixin*, *ProteinModelWrapper*

A protein sequence embedder that performs one-hot encoding for aligned sequences.

This class allows for variable-length sequences and requires an MSA for fitting. It creates an encoding on the alignment including gaps. At prediction time, it can handle both aligned and unaligned sequences.

vocab

The vocabulary of amino acids and gap characters used for encoding.

Type
List[str]

encoder

The underlying sklearn OneHotEncoder.

Type
OneHotEncoder

positions

Specific positions to encode. If None, all positions are encoded.

Type
Optional[List[int]]

pool

Whether to pool the encoded vectors across positions.

Type
bool

`flatten`

Whether to flatten the output array.

Type
bool

`alignment_width`

The width of the original alignment.

Type
int

`original_alignment`

The original alignment used for fitting.

Type
ProteinSequences

`get_feature_names_out`(*input_features: List[str] | None = None*) → List[str]

Get output feature names for transformation.

Parameters
input_features (*Optional[List[str]]*) – Ignored. Present for API consistency.

Returns
Output feature names.

Return type
List[str]

`set_fit_request`(*, *force: bool | None | str = '$UNCHANGED$'*) → *OneHotAlignedEmbedding*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `force` parameter in `fit`.

Returns
self – The updated object.

Return type
object

class aide_predict.bespoke_models.embedders.ohe.**OneHotProteinEmbedding**(*metadata_folder: str | None = None*, *wt: str | ProteinSequence | None = None*, *positions: List[int] | None = None*, *flatten: bool = True*, *pool: bool = False*)

Bases: *PositionSpecificMixin*, *RequiresFixedLengthMixin*, *ProteinModelWrapper*

A protein sequence embedder that performs one-hot encoding with position-specific capabilities.

This class wraps sklearn's OneHotEncoder to provide one-hot encoding specifically for protein sequences. It expects fixed-length sequences without gaps and uses a 20 amino acid vocabulary. It also allows for position-specific encoding.

vocab
The vocabulary of amino acids used for encoding.

Type
List[str]

encoder
The underlying sklearn OneHotEncoder.

Type
OneHotEncoder

positions
Specific positions to encode. If None, all positions are encoded.

Type
Optional[List[int]]

pool
Ignored

Type
bool

flatten
Whether to flatten the output array.

Type
bool

seq_length
The length of the sequences, determined during fitting.

Type
Optional[int]

get_feature_names_out(*input_features: List[str] | None = None*) → List[str]

Get output feature names for transformation.

Parameters
input_features (*Optional[List[str]]*) – Ignored. Present for API consistency.

Returns
Output feature names.

Return type
List[str]

inverse_transform(*X: ndarray*) → ProteinSequences

Convert one-hot encoded vectors back into protein sequences.

Parameters
X (*np.ndarray*) – The one-hot encoded sequences to inverse transform.

Returns
The reconstructed protein sequences.

Return type
ProteinSequences

Raises
ValueError – If the input shape is incompatible with the encoder's expectations.

set_fit_request(**, force: bool | None | str = '$UNCHANGED$'*) → *OneHotProteinEmbedding*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `force` parameter in `fit`.

Returns
self – The updated object.

Return type
object

Module contents

- Author: Evan Komp
- Created: 7/5/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

aide_predict.bespoke_models.predictors package

Submodules

aide_predict.bespoke_models.predictors.esm2 module

- Author: Evan Komp
- Created: 6/14/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Using ESM as a zero shot evaluator.

ESM has a few methods for which to evaluate likelihoods, see the paper: Meier, J. et al. Language models enable zero-shot prediction of the effects of mutations on protein function. Preprint at https://doi.org/10.1101/2021.07.09.450648 (2021).

The paper explored the following methods: 1. Masked Marginal Likelihood (masked_marginal) (Not yet implemented)

   Pass the wild type sequence L times, where L is the length of the sequence. Compute the likelihood of each AA at each position. Compare mutant vs wildtype AA at each position.

2. **Mutant Marginal Likelihood (mutant_marginal) (Not yet implemented)**
   Pass each variant sequence. N forward passes, where N is the count of variants. Compute the likelihood of mutated vs wildtype AA on each variant.
3. **Wildtype Marginal Likelihood (wildtype_marginal)**
   Pass the wild type sequence. 1 forward pass, regardless of count of variants Compute the likelihood of mutated vs wildtype AA.
4. **Psuedo-Likelihood (pseudo_likelihood) (Not implmented)**
   No plans to implement, proved poor performance in the paper.

Since ESM is a transformer, it can output position specific scores. Recall that such a model must adhere to the following rules: Inherits from PositionSpecificMixin, which enforces that *positions* is a parameter. We can use those positions to extract likelihoods at specific positions. If *positions* is None, we will return all positions.

There is a lot of here. Let's lay out a logic table to determine how to be most efficient here.

WT | Fixed Length | Positions passed | Pool | Method | N passes | Description

|----|————|------------------|——|--------|—-| —- | Y | Y | N | Y | masked | M unique mutated positions in whole set, < L | Traditional masked marginal as described in the paper. Take WT, mask each mutated position, compare to WT, pool | Y | Y | N | N | masked | L | Can no longer only mask mutated positions since we are not pooling. Must mask all positions. This is L forward passes. Return comparison of mut to wt for each position individually. Many will be zero if they are not mutated anywhere. | Y | Y | Y | Y | masked | Positions passed | Mask each position, compare to WT, pool | Y | Y | Y | N | masked | Positions passed | Mask each position, compare to WT, no pooling output positions | Y | Y | N | Y | wild_type | 1 | Traditional wild type marginal as described in the paper. Take WT and pass. Compare mutant likelihood to WT and pool only the mutated positions | Y | Y | N | N | wild_type | 1 | Take WT and pass. Compare mutant likelihood to WT on WT probability vector. Many positions will be zero since they are unmutated | Y | Y | Y | Y | wild_type | 1 | Take WT and pass. Compare mutant likelihood to WT on WT probability vector for only chosen positions. Pool. | Y | Y | Y | N | wild_type | 1 | Take WT and pass. Compare mutant likelihood to WT on WT probability vector for only chosen positions. No pooling. | Y | Y | N | Y | mutant | N | Traditional mutant marginal as described in the paper. Take each mutant and pass. Compare mutant likelihood to WT for only mutate positions on the mutant probability vector. Pool. | Y | Y | N | N | mutant | N | Take each mutant and pass. Compare mutant likelihood to WT for all positions on the mutant probability vector many will be zero. No pooling. | Y | Y | Y | Y | mutant | N | Take each mutant and pass. Compare mutant likelihood to WT on the mutant vector for positions specified. | Y | Y | Y | N | mutant | N | Take each mutant and pass. Compare mutant likelihood to WT on the mutant vector for positions specified. No pooling. | N | Y | N | Y | masked | L*N | Mask each position of each mutant, check probability of true AA at each position. Pool. | N | Y | N | N | masked | L*N | Mask each position of each mutant, check probability of true AA at each position. No pooling. | N | Y | Y | Y | masked | N * positions passed | Mask mutants on each position passed, check probability of true AA at each position. Pool. | N | Y | Y | N | masked | N * positions passed | Mask mutants on each position passed, check probability of true AA at each position. No pooling. | N | Any | Any | Any | wild_type | 0 | Not avialable. No wild type to compare to | N | Y | N | Y | mutant | N | Pass each mutant, check probability of true AA at each position. Pool. | N | Y | N | N | mutant | N | Pass each mutant, check probability of true AA at each position. No pooling. | N | Y | Y | Y | mutant | N | Pass each mutant, check probability of true AA at only passed positions. Pool. | N | Y | Y | N | mutant | N | Pass each mutant, check probability of true AA at only passed positions. No pooling. | N | N | N | Y | masked | ~L*N | Mask each position of each mutant, check probability of true AA at each position. Pool. | N | N | N | N | masked | 0 | Not available. Not pooling results in variabel length outputs. | N | N | Y | Y | masked | 0 | Not available. Cannot specify positions with variable length sequences. | N | N | Y | N | masked | 0 | Not available. Cannot specify positions with variable length sequences. | N | N | N | Y | mutant | N | Pass each mutant, check probability of true AA at each position. Pool. | N | N | N | N | mutant | 0 | Not available. Not pooling results in variabel length outputs. | N | N | Y | Y | mutant | 0 | Not available. Cannot specify positions with variable length sequences. | N | N | Y | N | mutant | 0 | Not available. Cannot specify positions with variable length sequences. | Y | N | N | Y | masked | ~L*(N+1) | Mask each position of each mutant, check probability of true AA at each position. Pool. Repeat for WT and noramlize. | Y | N | N | N | masked | 0 | Not available. Not pooling results in variabel length outputs. | Y | N | Y | Y | masked | 0 | Not available. Cannot specify positions with variable length sequences. | Y | N | Y | N | masked | 0 | Not available. Cannot specify positions with variable length sequences. | Y | N | N | Y | wild_type | 0 | Not available. Wild type not same length as mutants, so you cannit look at mutant likelihood from wt pass. | Y | N | N | N | wild_type | 0 | Not available. Wild type not same length as mutants, so you cannit look at mutant likelihood from wt pass. | Y | N | Y | Y | wild_type | 0 | Not available. Wild type not same length as mutants, so you cannit look at mutant likelihood from wt pass. | Y | N | Y | N | wild_type | 0 | Not available. Wild type not same length as mutants, so you cannit look at mutant likelihood from wt pass. | Y | N | N | Y | mutant | N+1 | Pass each mutant, check probability of true AA at each position on its own probability vector. Pool. Normalize by WT value | Y | N | N | N | mutant | 0 | Not available. Not pooling results in variabel length outputs. | Y | N | Y | Y | mutant | 0 | Not available. Cannot specify positions with variable length sequences. | Y | N | Y | N | mutant | 0 | Not available. Cannot specify positions with variable length sequences.

Conclusions:

1. If Variable length sequences, must pool. Cannot pass positions. wild_type marginal not available
2. If no wild type is given, only mutant or masked marginal is available.
3. Masked marginal removed for the case where wt is not given or sequences are variable length. For these cases, masks will have to be applied to all sequences not just the WT, vastly increasing cost.

Oh boy.

class aide_predict.bespoke_models.predictors.esm2.**ESM2LikelihoodWrapper**(*metadata_folder: str | None = None*, *model_checkpoint: str = 'esm2_t6_8M_UR50D'*, *marginal_method: MarginalMethod = 'mutant_marginal'*, *positions: list | None = None*, *pool: bool = True*, *flatten: bool = True*, *wt: str | None = None*, *batch_size: int = 2*, *device: str = 'cpu'*, *use_cache: bool = True*)

Bases: *CacheMixin*, *RequiresFixedLengthMixin*, LikelihoodTransformerBase

get_feature_names_out(*input_features: List[str] | None = None*) → List[str]

Get output feature names for transformation.

Parameters
input_features (*Optional[List[str]]*) – Input feature names (not used in this method).

Returns
Output feature names.

Return type
List[str]

Raises
ValueError – If the model hasn't been fitted or if feature names can't be generated.

set_fit_request(**, force: bool | None | str = '$UNCHANGED$'*) → *ESM2LikelihoodWrapper*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `force` parameter in `fit`.

Returns
self – The updated object.

Return type
object

`set_score_request`(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *ESM2LikelihoodWrapper*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `score`.

Returns
self – The updated object.

Return type
object

aide_predict.bespoke_models.predictors.hmm module

- Author: Evan Komp
- Created: 6/11/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Wrapper of HMMs into an sklearn transformer for use in the AIDE pipeline. Uses HMMsearch against the HMM

Here are the docs for HMMSearch:

Usage: hmmsearch [options] <hmmfile> <seqdb>

Basic options:
-h : show brief help on version and usage

Options directing output:

-o <f> : direct output to file <f>, not stdout

-A <f> : save multiple alignment of all hits to file <f>

--tblout <f> : save parseable table of per-sequence hits to file <f>

--domtblout <f> : save parseable table of per-domain hits to file <f>

–pfamtblout <f> : save table of hits and domains to file, in Pfam format <f> –acc : prefer accessions over names in output –noali : don't output alignments, so output is smaller –notextw : unlimit ASCII text output line width –textw <n> : set max width of ASCII text output lines [120] (n>=120)

Options controlling reporting thresholds:

-E <x> : report sequences <= this E-value threshold in output [10.0] (x>0)

-T <x> : report sequences >= this score threshold in output

–domE <x> : report domains <= this E-value threshold in output [10.0] (x>0) –domT <x> : report domains >= this score cutoff in output

Options controlling inclusion (significance) thresholds:

--incE <x> : consider sequences <= this E-value threshold as significant

--incT <x> : consider sequences >= this score threshold as significant

–incdomE <x> : consider domains <= this E-value threshold as significant –incdomT <x> : consider domains >= this score threshold as significant

Options controlling model-specific thresholding:
–cut_ga : use profile's GA gathering cutoffs to set all thresholding –cut_nc : use profile's NC noise cutoffs to set all thresholding –cut_tc : use profile's TC trusted cutoffs to set all thresholding

Options controlling acceleration heuristics:

--max : Turn all heuristic filters off (less speed, more power)

–F1 <x> : Stage 1 (MSV) threshold: promote hits w/ P <= F1 [0.02] –F2 <x> : Stage 2 (Vit) threshold: promote hits w/ P <= F2 [1e-3] –F3 <x> : Stage 3 (Fwd) threshold: promote hits w/ P <= F3 [1e-5] –nobias : turn off composition bias filter

Other expert options:

--nonull2 : turn off biased composition score corrections

-Z <x> : set # of comparisons done, for E-value calculation

--domZ <x> : set # of significant seqs, for domain E-value calculation

--seed <n> : set RNG seed to <n> (if 0: one-time arbitrary seed) [42]

–tformat <s> : assert target <seqfile> is in format <s>: no autodetection –cpu <n> : number of parallel CPU workers to use for multithreads [2]

Some of these need to be user parameterizable, and some need to be fixed.

class aide_predict.bespoke_models.predictors.hmm.**HMMWrapper**(*threshold: float = 100*, *metadata_folder: str | None = None*, *wt: str | ProteinSequence | None = None*)

Bases: *CanRegressMixin*, *RequiresMSAMixin*, *ProteinModelWrapper*

Wrapper for Hidden Markov Models (HMMs) using HMMsearch to score sequences.

This wrapper builds an HMM from an input alignment and uses HMMsearch to get scores for new sequences. Bit scores are used to compare to the HMM as opposed to E values. Tune the threshold parameter accordingly.

threshold

Threshold for HMMsearch.

Type
float

metadata_folder

Folder to store metadata.

Type
str

wt

Wild-type sequence.

Type
Optional[ProteinSequence]

set_fit_request(**, force: bool | None | str = '$UNCHANGED$'*) → *HMMWrapper*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters

force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `force` parameter in `fit`.

Returns
self – The updated object.

Return type
object

`set_score_request`(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *HMMWrapper*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `sample_weight` parameter in `score`.

Returns
self – The updated object.

Return type
object

aide_predict.bespoke_models.predictors.msa_transformer module

- Author: Evan Komp
- Created: 7/8/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

class aide_predict.bespoke_models.predictors.msa_transformer.**MSATransformerLikelihoodWrapper**(*metadata_fold*
str
|
None
=
None,
marginal_me
Marginal-
Method
=
Marginal-
Method.WILD
po-
si-
tions:
List[int]
|
None
=
None,
flat-
ten:
bool
=
False,
pool:
bool
=
True,
batch_size:
int
=
32,
de-
vice:
str
=
'cpu',
n_msa_seqs:
int
=
360,
wt:
str
|
Pro-
tein-
Se-
quence
|
None
=
None)

Bases: *CacheMixin*, *RequiresMSAMixin*, *RequiresFixedLengthMixin*, LikelihoodTransformerBase

A wrapper for the MSA Transformer model to compute log likelihoods for protein sequences.

This class uses the MSA Transformer model to calculate log likelihoods for protein sequences based on multiple sequence alignments (MSAs). It supports various marginal likelihood calculation methods and can handle masked positions.

_available

Indicates whether the MSA Transformer model is available.

Type
MessageBool

set_fit_request(*, *force: bool | None | str = '$UNCHANGED$'*) → *MSATransformerLikelihoodWrapper*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters
force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `force` parameter in `fit`.

Returns
self – The updated object.

Return type
object

set_score_request(*, *sample_weight: bool | None | str = '$UNCHANGED$'*) → *MSATransformerLikelihoodWrapper*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.

- None: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- str: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (sklearn.utils.metadata_routing.UNCHANGED) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a Pipeline. Otherwise it has no effect.

Parameters

sample_weight (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for sample_weight parameter in score.

Returns

self – The updated object.

Return type

object

Module contents

- Author: Evan Komp
- Created: 6/26/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Submodules

aide_predict.bespoke_models.base module

- Author: Evan Komp
- Created: 5/7/2024
- (c) Copyright by Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology

Base classes for models to be wrapped into the API as sklearn estimators

class aide_predict.bespoke_models.base.**AcceptsLowerCaseMixin**

Bases: object

Mixin to indicate that a model can accept lower case sequences.

This mixin overrides the accepts_lower_case attribute to be True.

class aide_predict.bespoke_models.base.**CacheMixin**(**args*, *use_cache: bool = True*, ***kwargs*)

Bases: object

Mixin to provide per-protein caching functionality for ProteinModelWrapper subclasses. Uses SQLite for metadata indexing and HDF5 for efficient embedding storage. Optimized for batch operations and improved file handling.

get_fitted_attributes() → List[str]

Get a list of attributes that are set during fitting.

transform(*X: ProteinSequences | List[str]*) → ndarray

Override transform to use cache when possible on a per-protein basis.

class aide_predict.bespoke_models.base.**CanHandleAlignedSequencesMixin**

Bases: object

Mixin to indicate that a model can handle aligned sequences (with gaps) during prediction.

This mixin overrides the can_handle_aligned_sequences attribute to be True.

class aide_predict.bespoke_models.base.**CanRegressMixin**

Bases: RegressorMixin

Mixin to ensure model can regress.

This mixin overrides the can_regress attribute to be True. It also overrides the score method to use spearman correlation isntead of R2, such that it can be used out of the mox with zero shot predicors.

score(*X*, *y*, *sample_weight=None*)

Return the Spearman correlation

class aide_predict.bespoke_models.base.**PositionSpecificMixin**(*positions: bool | None = None*, *pool: bool = True*, *flatten: bool = True*, **args*, ***kwargs*)

Bases: object

Mixin for protein models that can output per position scores.

This mixin: 1. Overrides the per_position_capable attribute to be True. 2. Checks that positions, pool, and flatten are attributes. 3. Wraps the predict and transform methods to check that if positions were passed and not pooling, the output is the same length as the positions. 4. Flattens the output if flatten is True.

Note that you are responsible for selecting positions and pooling. This mixing only provides checks that the output is consistent with the specified positions. You DO NOT need to implement flattening, as this mixin will handle it for you.

positions

The positions to output scores for.

Type
Optional[List[int]]

pool

Whether to pool the scores across positions.

Type
bool

flatten

Whether to flatten dimensions beyond the second dimension.

Type
bool

`get_feature_names_out`(*input_features: List[str] | None = None*) → List[str]

Get output feature names for transformation, considering position-specific output and flattening.

Parameters
input_features (*Optional[List[str]]*) – Input feature names.

Returns
Output feature names.

Return type
List[str]

`transform`(*X: ProteinSequences | List[str]*) → ndarray

Transform the sequences, ensuring correct output dimensions for position-specific models. If flatten is True, flatten dimensions beyond the second dimension.

Parameters
X (*Union[ProteinSequences, List[str]]*) – Input sequences.

Returns
Transformed sequences.

Return type
np.ndarray

Raises
ValueError – If the output dimensions do not match the specified positions.

`class` aide_predict.bespoke_models.base.`ProteinModelWrapper`(*metadata_folder: str | None = None*, *wt: str | ProteinSequence | None = None*)

Bases: `TransformerMixin`, `BaseEstimator`

Base class for bespoke models that take proteins as input.

This class serves as a foundation for creating protein-based models that can be used in machine learning pipelines, particularly those compatible with scikit-learn. It provides a standard interface for fitting, transforming, and predicting protein sequences, as well as handling metadata and wild-type sequences.

All models that take proteins as input should inherit from this class. They are considered transformers and can be used natively to produce features in the AIDE pipeline. Models can additionally be made regressors by inheriting from RegressorMixin.

X values for fit, transform, and predict are expected to be ProteinSequences objects.

`metadata_folder`

The folder where the metadata is stored.

Type
str

`wt`

The wild type sequence if present.

Type
Optional[ProteinSequence]

Class Attributes:

requires_msa_for_fit (bool): Whether the model requires an MSA as input for fitting. requires_wt_to_function (bool): Whether the model requires the wild type sequence to function. requires_wt_during_inference (bool): Whether the model requires the wild type sequence during inference. per_position_capable (bool): Whether the model can output per position scores. requires_fixed_length (bool): Whether the model requires a fixed length input. can_regress (bool): Whether the model outputs from transform can also be considered estimates of activity label. can_handle_aligned_sequences (bool): Whether the model can handle unaligned sequences at predict time. should_refit_on_sequences (bool): Whether the model should refit on new sequences when given. requires_structure (bool): Whether the model requires structure information. _available (bool): Flag to indicate whether the model is available for use.

To subclass ProteinModelWrapper: 1. Implement the abstract methods:

- _fit(self, X: ProteinSequences, y: Optional[np.ndarray] = None) -> None
- _transform(self, X: ProteinSequences) -> np.ndarray

2. If your model supports partial fitting, implement: - _partial_fit(self, X: ProteinSequences, y: Optional[np.ndarray] = None) -> None
3. If your model requires specific metadata, override: - check_metadata(self) -> None - _construct_necessary_metadata(cls, model_directory: str, necessary_metadata: dict) -> None
4. If your model has additional parameters, implement __init__ and call super().__init__ with the metadata_folder and wt arguments.
5. If your model requires specific behavior, consider inheriting from the provided mixins. See the mixins for the provided behaviors: - RequiresMSAMixin - if the model requires an MSA for fitting - RequiresFixedLengthMixin - if the model requires fixed length sequences at predict time - CanRegressMixin - if the model can regress, otherwise it is assumed to be a transformer only eg. embedding - RequiresWTToFunctionMixin - if the model requires the wild type sequence to function - RequiresWTDuringInferenceMixin - if the model requires the wild type sequence duing inference in order to normalize by wt - PositionSpecificMixin - if the model can output per position scores - RequiresStructureMixin - if the model requires structure information - AcceptsLowerCaseMixin - if the model can accept lower case sequences - ShouldRefitOnSequencesMixin - if the model should refit on new sequences when given. Often, we are calling fit on NOT raw sequences, eg. MSAs.

   We still want to be able to use the model in the context of sklearn pipelines which will attempt to clone and refit the model on X data. We want the models to return themselves already fitted when cloned, unless this is mixex in
6. If the model requires more than the base package, set the _available attribute to be dynamic based on a check in the module.

Example

ESM2 using WT marginal can be used as a "regressor".

try:

import transformers AVALABLE = MessageBool(True, "This model is available.")

except ImportError:

AVALABLE = MessageBool(False, "This model is not available, make sure transformers is installed.")

class ESM2Model(CanRegressMixin, PositionSpecificMixin, ProteinModelWrapper):

_available = AVAILABLE

def __init__(self, model_checkpoint: str, metadata_folder: str, wt: Optional[Union[str, ProteinSequence]] = None):
super().__init__(metadata_folder, wt) self.model_checkpoint = model_checkpoint

def _fit(self, X: ProteinSequences, y: Optional[np.ndarray] = None) -> None:
Fit the model … return self

def _transform(self, X: ProteinSequences) -> np.ndarray:
Transform the sequences … return outputs

property accepts_lower_case: bool

Whether the model can accept lower case sequences.

property can_handle_aligned_sequences: bool

Whether the model can handle aligned sequences (with gaps) at predict time.

property can_regress: bool

Whether the model can perform regression.

check_metadata() → None

Ensures that everything this model class needs is in the metadata folder.

fit(*X: ProteinSequences | List[str]*, *y: ndarray | None = None*, *force: bool = False*) → *ProteinModelWrapper*

Fit the model.

Parameters

- **X** (*Union[ProteinSequences, List[str]]*) – Input sequences.
- **y** (*Optional[np.ndarray]*) – Target values.

Returns
The fitted model.

Return type
ProteinModelWrapper

get_feature_names_out(*input_features: List[str] | None = None*) → List[str]

Get output feature names for transformation.

Parameters
input_features (*Optional[List[str]]*) – Input feature names.

Returns
Output feature names.

Return type
List[str]

get_params(*deep: bool = True*) → Dict[str, Any]

Get parameters for this estimator.

Parameters
deep (*bool*) – If True, will return the parameters for this estimator and contained subobjects.

Returns
Parameter names mapped to their values.

Return type
Dict[str, Any]

property metadata_folder

partial_fit(*X: ProteinSequences | List[str]*, *y: ndarray | None = None*) → *ProteinModelWrapper*

Partially fit the model to the given sequences.

This method can be called multiple times to incrementally fit the model.

Parameters

- **X** (*Union[ProteinSequences, List[str]]*) – The input sequences to partially fit the model on.
- **y** (*Optional[np.ndarray]*) – The target values, if applicable.

Returns
The partially fitted model.

Return type
ProteinModelWrapper

property per_position_capable: bool

Whether the model can output per position scores.

predict(*X: ProteinSequences | List[str]*) → ndarray

Predict the sequences.

Parameters
X (*Union[ProteinSequences, List[str]]*) – Input sequences.

Returns
Predicted values.

Return type
np.ndarray

Raises
ValueError – If the model is not capable of regression.

property requires_fixed_length: bool

Whether the model requires fixed length input.

property requires_msa_for_fit: bool

Whether the model requires an MSA for fitting.

property requires_structure: bool

Whether the model requires structure information.

property requires_wt_during_inference: bool

Whether the model requires the wild type sequence during inference.

property requires_wt_to_function: bool

Whether the model requires the wild type sequence to function.

set_fit_request(**, force: bool | None | str = '$UNCHANGED$'*) → *ProteinModelWrapper*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.

- None: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- str: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (sklearn.utils.metadata_routing.UNCHANGED) retains the existing request. This allows you to change the request for some parameters and not others.

Added in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a Pipeline. Otherwise it has no effect.

Parameters
force (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for force parameter in fit.

Returns
self – The updated object.

Return type
object

set_params(***params: Any*) → *ProteinModelWrapper*

Set the parameters of this estimator.

Parameters
****params** – Estimator parameters.

Returns
Estimator instance.

Return type
ProteinModelWrapper

property should_refit_on_sequences: bool

Whether the model should refit on new sequences when given.

transform(*X: ProteinSequences | List[str]*) → ndarray

Transform the sequences.

Parameters
X (*Union[ProteinSequences, List[str]]*) – Input sequences.

Returns
Transformed sequences.

Return type
np.ndarray

property wt

class aide_predict.bespoke_models.base.**RequiresFixedLengthMixin**

Bases: object

Mixin to ensure model receives fixed length sequences at transform.

This mixin overrides the requires_fixed_length attribute to be True.

class aide_predict.bespoke_models.base.**RequiresMSAMixin**

Bases: object

Mixin to ensure model receives aligned sequences at fit.

This mixin overrides the requires_msa_for_fit attribute to be True.

class aide_predict.bespoke_models.base.**RequiresStructureMixin**

Bases: object

Mixin to ensure model requires structure information.

This mixin overrides the requires_structure attribute to be True.

class aide_predict.bespoke_models.base.**RequiresWTDuringInferenceMixin**

Bases: object

Mixin to ensure model requires wild type during inference.

This mixin overrides the requires_wt_during_inference attribute to be True.

class aide_predict.bespoke_models.base.**RequiresWTToFunctionMixin**

Bases: object

Mixin to ensure model requires wild type to function.

This mixin overrides the requires_wt_to_function attribute to be True.

class aide_predict.bespoke_models.base.**ShouldRefitOnSequencesMixin**

Bases: object

Mixin to indicate that a model should refit on new sequences when given.

This mixin overrides the should_refit_on_sequences attribute to be True.

aide_predict.bespoke_models.base.**is_jsonable**(*x*)

Checks if an object is JSON serializable.

aide_predict.bespoke_models.eve module

aide_predict.bespoke_models.tranception module

Module contents

- Author: Evan Komp
- Created: 5/7/2024
- (c) Copyright by Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology

aide_predict.embeddings package

Submodules

aide_predict.embeddings.base module

Module contents

aide_predict.io package

Submodules

aide_predict.io.bio_files module

- Author: Evan Komp
- Created: 5/22/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Importing EVcouplings alignment IO into the namespace. All credit goes to the EVcouples team:

Hopf T. A., Green A. G., Schubert B., et al. The EVcouplings Python framework for coevolutionary sequence analysis. Bioinformatics 35, 1582–1584 (2019)

Module contents

- Author: Evan Komp
- Created: 5/7/2024
- (c) Copyright by Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology

aide_predict.utils package

Submodules

aide_predict.utils.alignment_calls module

- Author: Evan Komp
- Created: 6/12/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Wrapper of EVCouplings alignment functions. All credit goes to the EVcouplings team: Hopf T. A., Green A. G., Schubert B., et al. The EVcouplings Python framework for coevolutionary sequence analysis. Bioinformatics 35, 1582–1584 (2019)

aide_predict.utils.alignment_calls.**mafft_align**(*sequences: ProteinSequences*, *existing_alignment: ProteinSequences | None = None*, *realign: bool = False*, *output_fasta: str | None = None*) → ProteinSequences

Perform multiple sequence alignment using MAFFT.

Parameters

- **sequences** (*ProteinSequences*) – The sequences to align.
- **existing_alignment** (*Optional[ProteinSequences]*) – An existing alignment to add sequences to.
- **realign** (*bool*) – If True, realign all sequences from scratch. If False, add new sequences to existing alignment.
- **output_fasta** (*Optional[str]*) – Path to save the alignment. If None, a temporary file is used.

Returns
The aligned sequences, either in memory or on file depending on output_fasta.

Return type
ProteinSequences

Raises

- **subprocess.CalledProcessError** – If MAFFT execution fails.
- **FileNotFoundError** – If MAFFT is not installed or not in PATH.

aide_predict.utils.alignment_calls.**sw_global_pairwise**(*seq1: ProteinSequence*, *seq2: ProteinSequence*, *matrix: str = 'BLOSUM62'*, *gap_open: float = -10*, *gap_extend: float = -0.5*) → tuple[ProteinSequence, ProteinSequence]

Align two ProteinSequence objects using global alignment with a specified substitution matrix.

Parameters

- **seq1** (*ProteinSequence*) – The first protein sequence to align.
- **seq2** (*ProteinSequence*) – The second protein sequence to align.
- **matrix** (*str, optional*) – The substitution matrix to use. Defaults to 'BLOSUM62'.
- **gap_open** (*float, optional*) – The gap opening penalty. Defaults to -10.
- **gap_extend** (*float, optional*) – The gap extension penalty. Defaults to -0.5.

Returns
A tuple containing the aligned sequences as ProteinSequence objects.

Return type
tuple[ProteinSequence, ProteinSequence]

aide_predict.utils.checks module

- Author: Evan Komp
- Created: 6/13/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Common checks to ensure that different pipeline components are compatable.

aide_predict.utils.checks.**check_dvc_params**()

Ensures the user provided the necessary data and did not ask for incompatible steps.

Returns pre-run metrics about the pipeline.

Checks done: - If the user asks for a model that requires an MSA, ensure the MSA step is on. - If the user is using specifying specific positions to score, ensure that any

sequences to be evaluated are fixed length and all models are capable of position specific scoring. Currently incompatable with supervised models.

- If the user gives either training or test data, and any have different legnths, ensure that models are capable of handling variable length sequences.
- If the user asks for jackhmmer search, ensure that wt.fasta was provided
- If the user asks for supervised and or/msa mode that adds training sequences, ensure that the training sequences are provided.
- If the user asks for CV, ensure training data is provided.

aide_predict.utils.checks.**check_model_compatibility**(*training_sequences: ProteinSequences | None = None*, *testing_sequences: ProteinSequences | None = None*, *training_msa: ProteinSequences | None = None*, *wt: ProteinSequence | None = None*) → Dict[str, List[str]]

Check which models are compatible with the given data.

Parameters

- **training_sequences** (*Optional[ProteinSequences]*) – Training protein sequences.
- **testing_sequences** (*Optional[ProteinSequences]*) – Testing protein sequences.
- **training_msa** (*Optional[ProteinSequences]*) – Training multiple sequence alignment.
- **wt** (*Optional[ProteinSequence]*) – Wild-type protein sequence.

Returns

A dictionary with two keys: 'compatible' and 'incompatible', each containing a list of compatible and incompatible model names respectively.

Return type

Dict[str, List[str]]

aide_predict.utils.checks.**get_supported_tools**()

aide_predict.utils.common module

- Author: Evan Komp
- Created: 6/11/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Common utility functions

class aide_predict.utils.common.**MessageBool**(*value*, *message*)

Bases: object

aide_predict.utils.common.**convert_dvc_params**(*dvc_params_dict: dict*)

DVC Creates a nested dict with the parameters.

We want an object that has nested attributes so that we can access parameters with dot notation.

aide_predict.utils.constants module

- Author: Evan Komp
- Created: 6/11/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

aide_predict.utils.data_structures module

- Author: Evan Komp
- Created: 7/10/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

aide_predict.utils.msa module

- MSAProcessing class Refactored from Frazer et al.

@article{Frazer2021DiseaseVP,

title={Disease variant prediction with deep generative models of evolutionary data.}, author={Jonathan Frazer and Pascal Notin and Mafalda Dias and Aidan Gomez and Joseph K Min and Kelly P. Brock and Yarin Gal and Debora S. Marks}, journal={Nature}, year={2021}

}

- Author: Evan Komp
- Created: 5/8/2024
- (c) Copyright by Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology

Peocessing of MSAs for preparation of input data for the zero-shot model. Note that The MSAProcessing class IS A REFACTORING of the MSA processing class from The marks Lab https://github.com/OATML-Markslab/EVE/blob/master/utils/data_utils.py Credit is given to them for the original implementation and the methodology of sequence weighting. Here, we make it more pythonic and readbale, as well as an order of magnitude speed up.

In addition to refactoring, we add some additional functionality: - A focus seq need not be present, in which case all columns are considered focus columns and contribute to weight computation - One Hot encoding is reworked to use sklearn's OneHotEncoder instead of a loop of loops, with about an order of magnitude speedup - Weight computation leverages numpy array indexing instead of a loop, and if torch is available

and advanced hardware is present, GPU is used.

Tested on 10000 protein sequences sequences of length 55:

- original: 8.9 seconds
- cpu array operations: 1.2 seconds (7.4x speedup)
- gpu array operations: 0.2 seconds (44.5x speedup)

- other minor speedups with array operations

class aide_predict.utils.msa.**MSAProcessing**(*theta: float = 0.2*, *use_weights: bool = True*, *preprocess_msa: bool = True*, *threshold_sequence_frac_gaps: float = 0.5*, *threshold_focus_cols_frac_gaps: float = 0.3*, *remove_sequences_with_indeterminate_aa_in_focus_cols: bool = True*, *weight_computation_batch_size: int = 10000*)

Bases: `object`

get_most_populated_chunk(*msa: ProteinSequences*, *chunk_size: int*) → ProteinSequences

Get the most populated chunk of contiguous columns from the MSA.

Parameters

- **msa** (*ProteinSequences*) – The input MSA.
- **chunk_size** (*int*) – The size of the chunk.

Returns
The chunk of contiguous columns.

Return type
ProteinSequences

process(*msa: ProteinSequences*, *focus_seq_id: str | None = None*) → ProteinSequences

Process the input MSA.

Parameters

- **msa** (*ProteinSequences*) – The input multiple sequence alignment.
- **focus_seq_id** (*Optional[str]*) – The ID of the focus sequence. If None, no focus sequence is used.

Returns
The processed MSA with computed weights.

Return type
ProteinSequences

Module contents

- Author: Evan Komp
- Created: 5/7/2024
- (c) Copyright by Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology

1.1.2 Module contents

- Author: Evan Komp
- Created: 5/7/2024
- (c) Copyright by Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

O

P

R

S

T

V

W