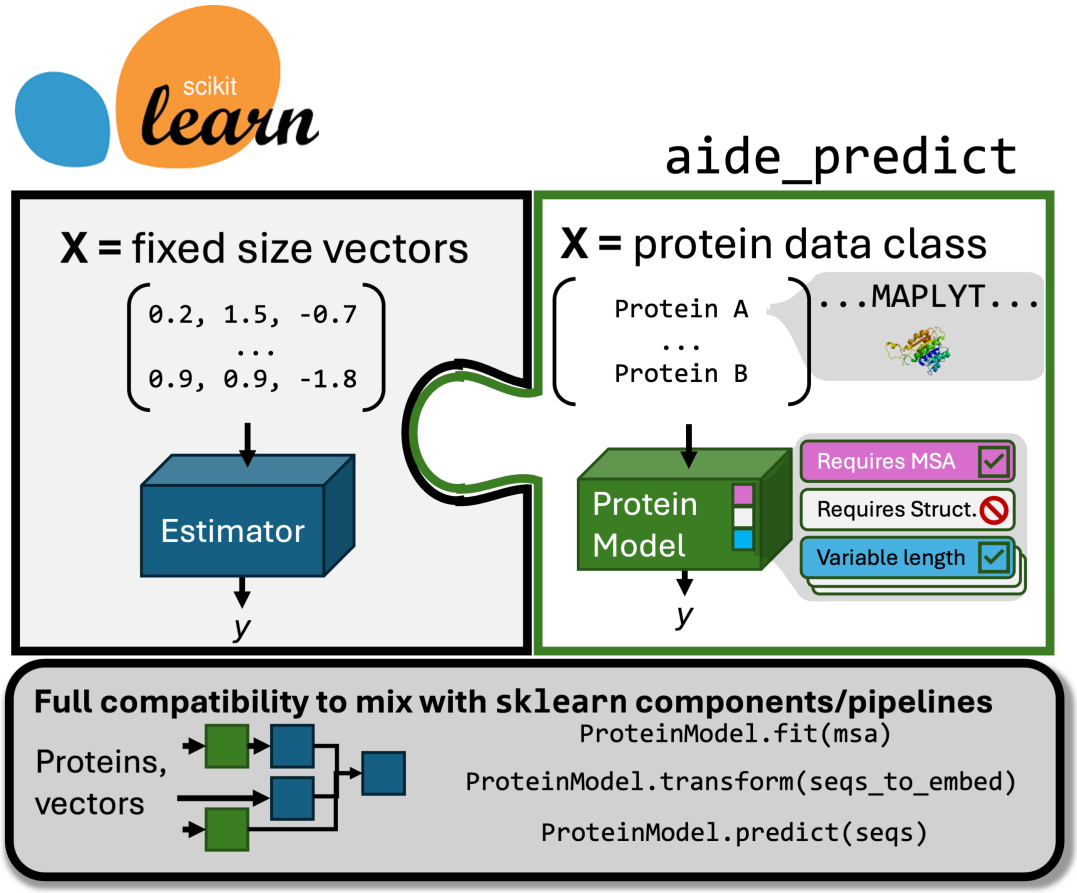
aide_predict

Release v0

Evan Komp, Gregg T. Beckham

Oct 23, 2024

CONTENTS:


scikit
learn
aide_predict
X = fixed size vectors
0.2, 1.5, -0.7
...
0.9, 0.9, -1.8
Estimator
y
X = protein data class
Protein A
...
Protein B
...MAPLYT...
Protein
Model
Requires MSA
Requires Struct.
Variable length
y
Full compatibility to mix with sklearn components/pipelines
Proteins,
vectors
ProteinModel.fit(msa)
ProteinModel.transform(seqs_to_embed)
ProteinModel.predict(seqs)

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

This repository serves fundementally to increase the accessibility of protein engineering tasks that fall into the following catagory:

$$\hat{y} = f(X)$$

Here, X is a set of proteins, eg. their sequence and optionaly structure. y is a property of the protein that is difficult to measure, such as binding affinity, stability, or catalytic activity. $\hat{y}$ is the predicted value of y given X.

Existing models f in the literature are varied, and a huge amount of work has gone into designing clever algorithms that leverage labeled and unlabeled data. For example, models differ in the following ways (non exhaustive): - Some require supervised labels y, while others do not - Unsupervised models can be trained on vast sets of sequences, or MSAs of the related proteins - Models exist to predict the effect of mutations on a wild type sequence, or to globally predict protein properties - Some models incorporate structural information - Some models are pretrained - Some models are capable of position specific predictions, which can be useful for some tasks

The variety an nuance of each of these means that each application is a bespoke, independent codebase, and are generally inaccessible to those with little or no coding exprience. Some applications alleviate the second problem by hosting web servers. Add to this problem is a lack of standardization in API across applications, where individual code bases can be extremely poorly documented or hard to use due to hasty development to minimize time to publication.

The goals of this project are succinctly as follows: - [X] **Create a generalizable, unittested, API for protein prediction tasks that is compatible with scikit learn**. This API will allow those who are familiar with the gold standard of ML libraries to conduct protein prediction tasks in much the same way you'd see on an intro to ML Medium article. Further, it makes it much easier for bespoke strategies to be accessed and compared; any new method whose authors wrap their code in the API are easily accessed by the community without spending hours studying the codebase. - [ ] **Use API components to create a DVC tracked pipeline for protein prediction tasks**. This pipeline will allow for those with zero software experience to conduct protein prediction tasks with a few simple commands. After (optionally) editting a config file, inputing their training data and their putative proteins, they can train and get predictions as simply as executing `dvc repro`.

1.1 API examples:

The following should look and feel like canonical sklearn tasks/code. See the `demo` folder for more details and executable examples. Also see the colab notebook to play with some if its capabilities in the cloud. Finally, checkout the notebooks in `showcase` where we conduct two full protein predictions optimization and scoring tasks on real data that are greater than small example sets.

1.1.1 In silico mutagenesis using MSATransformer

```
# data preparation
wt = ProteinSequence(
    "LADDRTLLMAGVSHDLRTPLTRIRLATEMMSEQDGYLAESINKDIEECNAIIEQFIDYLR",
)
msa = ProteinSequences.from_fasta("data/msa.fasta")
library = wt.saturation_mutagenesis()
mutations = library.ids
print(mutations[0])
>>> 'L1A'

# model fitting
model = MSATransformerLikelihoodWrapper(
   wt=wt,
   marginal_method="masked_marginal"
)
model.fit(msa)

# make predictions for each mutated sequence
predictions = model.predict(library)

results = pd.DataFrame({'mutation': mutations, 'seqeunce': library,'prediction':␣
↪predictions})
```

1.1.2 Compare a couple of zero shot predictors against experimental data

```
# data preparation
data = pd.read_csv("data/experimental_data.csv")
X = ProteinSequences.from_list(data['sequence'])
y = data['experimental_value']
wt = X['my_id_for_WT']
msa = ProteinSequences.from_fasta("data/msa.fasta")

# model defenitions
evmut = EVMutation(wt=wt, metadata_folder='./tmp/evm')
evmut.fit(msa)
esm2 = ESM2LikelihoodWrapper(wt=wt, model_checkpoint='esm2_t33_650M_UR50S')
esm2.fit([])
models = {'evmut': evmut, 'esm2': esm2}

# model fitting and scoring
for name, model in models.items():
```

(continues on next page)

(continued from previous page)

```
    score = model.score(X, y)
    print(f"{name} score: {score}")
```

1.1.3 Train a supervised model to predict activity on an experimental combinatorial library, test on sequences with greater mutational depth than training

```
# data preparation
data = pd.read_csv("data/experimental_data.csv")
sequences = ProteinSequences.from_list(data['sequence'])
sequences.aligned
>>> True
sequences.fixed_length
>>> True

wt = sequences['my_id_for_WT']
data['sequence'] = sequences
data['mutational_depth'] = data['sequence'].apply(lambda x: len(x.mutated_positions(wt)))
test = data[data['mutational_depth'] > 5]
train = data[data['mutational_depth'] <= 5]
train_X, train_y = train['sequence'], train['experimental_value']
test_X, test_y = test['sequence'], test['experimental_value']

# embeddings protein sequences
# use mean pool embeddings of esm2
embedder = ESM2Embedding(pool=True)
train_X = embedder.fit_transform(train_X)
test_X = embedder.transform(test_X)

# model fitting
model = RandomForestRegressor()
model.fit(train_X, train_y)

# model scoring
train_score = model.score(train_X, train_y)
test_score = model.score(test_X, test_y)
print(f"Train score: {train_score}, Test score: {test_score}")
```

1.1.4 Train a supervised predictor on a set of homologs, focusing only on positions of known importance, wrap the entire process into an sklearn pipeline including some standard sklearn transormers, and make predictions for a new set of homologs

```
# data preparation
data = pd.read_csv("data/experimental_data.csv")
data.set_index('id', inplace=True)
sequences = ProteinSequences.from_dict(data['sequence'].to_dict())
y_train = data['experimental_value']
```

(continues on next page)

(continued from previous page)

```
wt = sequences['my_id_for_WT']
wt_important_positions = np.array([20, 21, 22, 33, 45]) # zero indexed, known from␣
↪analysis elsewhere
sequences.aligned
>>> False
sequences.fixed_length
>>> False

# align the training sequences and get the important positions
msa = sequences.align_all()
msa.fixed_length
>>> False
msa.aligned
>>> True

wt_alignment_mapping = msa.get_alignment_mapping()['my_id_for_WT']
aligned_important_positions = wt_alignment_mapping[wt_important_positions]

# model defenitions
embedder = OneHotAlignedEmbedding(important_positions=aligned_important_positions).
↪fit(msa)
scaler = StandardScaler()
feature_selector = VarianceThreshold(threshold=0.2)
predictor = RandomForestRegressor()
pipeline = Pipeline([
    ('embedder', embedder),
    ('scaler', scaler),
    ('feature_selector', feature_selector),
    ('predictor', predictor)
])

# model fitting
pipeline.fit(sequences, y_train)

# score new analigned homologs
new_homologs = ProteinSequences.from_fasta("data/new_homologs.fasta")
y_pred = pipeline.predict(new_homologs)
```

1.2 Supported tools

Import `aide_predict.utils.checks.get_supported_tools()` to see the tools that are available based on your environment. The base package has few dependencies and concurrently few tools. Additional tools can be accessed with additional dependency steps. This choice was made to reduce dependency clashes for the codebase. For example, the base package does not include `pytorch`, but the environment can be extended with "requirements-transformers.txt" to access ESM2 embeddings and log likelihood predictors.

1.3 Available Tools

- Protein Sequence and Structure data structures
- `StructureMapper` - A utility for mapping a folder of PDB structures to sequences

1. HMM (Hidden Markov Model)
    - Computes statistics over matching columns in an MSA, treating each column independantly but allowing for alignment of query sequences before scoring
    - Requires MSA for fitting
    - Can handle aligned sequences during inference
2. EVMutation
    - Computes pairwise couplings between AAs in an MSA for select positions well represented in the MSA, variants are scored by the change in coupling energy.
    - Requires MSA for fitting
    - Requires wild-type sequence for inference
    - Requires fixed-length sequences
3. ESM2 Likelihood Wrapper
    - Pretrained PLM (BERT style) model for protein sequences, scores variants according to masked, mutant, or wild type marginal likelihoods. Mutant marginal computes likelihoods in the context of the mutant sequence, while masked and wild type marginal compute likelihoods in the context of the wild type sequence. These methods are apprximations of the joint likelihood.
    - Can handle aligned sequences
    - Requires additional dependencies (see `requirements-transformers.txt`)
4. SaProt Likelihood Wrapper
    - ESM except using a size 400 vocabulary including local structure tokens from Foldseek's VAE. The authors only used Masked marginal, but we've made Wild type, Mutant, and masked marginals avialable.
    - Requires fixed-length sequences
    - Uses WT structure if structures of sequences are not passed
    - Requires additional dependencies:
        - `requirements-transformers.txt`
5. MSA Transformer Likelihood Wrapper
    - Like ESM but with a transformer model that is trained on MSAs. The variants are placed at the top position in the MSA and scores are computed along that row. Wild type, Mutant, and masked marginals avialable.
    - Requires MSA for fitting
    - Requires wild-type sequence during inference
    - Requires additional dependencies (see `requirements-fair-esm.txt`)
6. VESPA
    - Conservation head model trained on PLM embeddings and logistic regression used to predict if mutation is detrimental.
    - Requires wild type, only works for single point mutations

- Requires fixed-length sequences
- Requires additional dependencies (see `requirements-vespa.txt`)

1. One Hot Protein Embedding
   - Columnwise one hot encoding of amino acids for a fixed length set of sequences
   - Requires fixed-length sequences
   - Position specific
2. One Hot Aligned Embedding
   - Columnwise one hot encoding including gaps for sequences aligned to an MSA.
   - Requires MSA for fitting
   - Position specific
3. Kmer Embedding
   - Counts of observed amino acid kmers in the sequences
   - Allows for variable length sequences
4. ESM2 Embedding
   - Pretrained PLM (BERT style) model for protein sequences, outputs embeddings for each amino acid in the sequece from the last transformer layer.
   - Position specific
   - Requires additional dependencies (see `requirements-transformers.txt`)
5. SaProt Embedding
   - ESM except using a size 400 vocabulary including local structure tokens from Foldseek's VAE. AA embeddings from the last layer of the transformer are used.
   - Position specific
   - Requires additional dependencies:
     - `requirements-transformers.txt`
     - `foldseek` executable must be available in the PATH
6. MSA Transformer Embedding
   - Like ESM but with a transformer model that is trained on MSAs. The embeddings are computed for each amino acid in the query sequence in the context of an existing MSA
   - Requires MSA for fitting
   - Requires fixed-length sequences
   - Requires additional dependencies (see `requirements-fair-esm.txt`)

Each model in this package is implemented as a subclass of `ProteinModelWrapper`, which provides a consistent interface for all models. The specific behaviors (e.g., requiring MSA, fixed-length sequences, etc.) are implemented using mixins, making it easy to understand and extend the functionality of each model.

1.4 Installation

```
conda env create -f environment.yaml
pip install .
```

1.5 Installation of additional modules

Tools that require additional dependancies can be installed with the corresponding requirements file. See above for those files. For example, to access VESPA:

```
pip install -r requirements-vespa.txt
```

1.6 Tests

Continuous integration only runs base module tests, eg. `pytest -v -m "not slow and not optional"`

Additional tests are availabe to check the scientific output of wrapped models, that they meet the expected values, such as: - Score of ESM2 log liklihood, MSATransformer, SaProt, VESPA againstENVZ_ECOLI_Ghose benchmark of ProteinGym - run with `pytest -v -m tests/not_base_models`

1.7 TODO:

- Write Tranception wrapper * (low priority, PN did not provide a clear entry point so will require some finagling)
- DVC pipeline of common tasks

1.8 Third party software

1. EVCouplings is a dependancy and their software is used to run jackhmmer searches and available as a position specific predictor.
2. Of course, many of the tools here are just wrapping of the work of others - see above.

1.9 Citations

No software or code with viral licenses was used in the creation of this project.

1.10 License

This project is licensed under the MIT License.

CHAPTER

TWO

MODULES

2.1 aide_predict

2.1.1 aide_predict package

Subpackages

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.kmer module

- Author: Evan Komp
- Created: 8/9/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

class aide_predict.bespoke_models.embedders.kmer.**KmerEmbedding**
metadata_folder: str | None = None, *k: int = 3*, *normalize: bool = True*, *wt:* ProteinSequence *| None = None*

Bases: *CanHandleAlignedSequencesMixin*, *ProteinModelWrapper*

A fast K-mer embedding class for protein sequences.

This class generates K-mer embeddings for protein sequences, handling both aligned and unaligned sequences efficiently.

k

The size of the K-mers.

Type
int

normalize

Whether to normalize the K-mer counts.

Type
bool

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 = '$UNCHANGED$'* → *KmerEmbedding*

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

aide_predict.bespoke_models.embedders.saprot module

- Author: Evan Komp
- Created: 7/16/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

class aide_predict.bespoke_models.embedders.saprot.**SaProtEmbedding**
metadata_folder: str | None = None, *model_checkpoint: str = 'westlake-repl/SaProt_650M_AF2'*, *layer: int = -1*, *positions: List[int] | None = None*, *flatten: bool = False*, *pool: bool = False*, *batch_size: int = 32*, *device: str = 'cpu'*, *foldseek_path: str = 'foldseek'*, *wt: str | ProteinSequence | None = None*, ***kwargs*

Bases: *CacheMixin*, *RequiresStructureMixin*, *PositionSpecificMixin*, *ProteinModelWrapper*

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

This class wraps the SaProt 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 SaProt 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

`foldseek_path`

Path to the FoldSeek executable.

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 = '$UNCHANGED$'* → *SaProtEmbedding*

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.evmutation module

- Author: Evan Komp
- Created: 7/12/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Wrapper around EVmutation model from the EVCouplings repositorty: https://github.com/debbiemarkslab/EVcouplings/tree/develop

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

class aide_predict.bespoke_models.predictors.evmutation.**EVMutationWrapper**
metadata_folder: str | None = None, *wt: str | ProteinSequence | None = None*, *protocol: str = 'standard'*, *theta: float = 0.8*, *iterations: int = 100*, *lambda_h: float = 0.01*, *lambda_J: float = 0.01*, *lambda_group: float | None = None*, *min_sequence_distance: int = 6*, *cpu: int = 1*, *use_cache: bool = False*

Bases: *CacheMixin*, *RequiresWTToFunctionMixin*, *RequiresFixedLengthMixin*, *RequiresMSAMixin*, *CanRegressMixin*, *AcceptsLowerCaseMixin*, *ProteinModelWrapper*

A wrapper for EVCouplings that implements the ProteinModelWrapper interface.

check_metadata
→ None

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

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
A list containing a single feature name.

Return type
List[str]

set_fit_request
**, force: bool | None | str = '$UNCHANGED$'* → *EVMutationWrapper*

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 = '$UNCHANGED$'* → *EVMutationWrapper*

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_folder: str | None = None, *marginal_method:* MarginalMethod *= MarginalMethod.WILDTYPE*, *positions: List[int] | None = None*, *flatten: bool = False*, *pool: bool = True*, *batch_size: int = 32*, *device: str = 'cpu'*, *n_msa_seqs: int = 360*, *wt: str |* ProteinSequence *| 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

aide_predict.bespoke_models.predictors.pretrained_transformers module

- Author: Evan Komp
- Created: 7/11/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Base class for log likelihood based transformer models. Supports wildtype, mutant, and masked marginal methods.

See: 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).

class aide_predict.bespoke_models.predictors.pretrained_transformers.**LikelihoodTransformerBase**(*metadata_folder: str | None = None*, *marginal_method: MarginalMethod = 'wildtype_marginal'*, *positions: List[int] | None = None*, *flatten: bool = False*, *pool: bool = True*, *batch_size: int = 2*, *device: str = 'cpu'*, *wt: str | ProteinSequence | None = None*)

Bases: *PositionSpecificMixin*, *CanRegressMixin*, *RequiresWTDuringInferenceMixin*, *ProteinModelWrapper*, ABC

Base class for likelihood transformer models.

This abstract base class provides a framework for implementing likelihood transformer models that can compute various types of marginal likelihoods for protein sequences.

marginal_method

The method used to compute marginal likelihoods.

Type
MarginalMethod

batch_size

The number of sequences to process in each batch.

Type
int

device

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

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]]*) – 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 = '$UNCHANGED$'*) → *LikelihoodTransformerBase*

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 = '$UNCHANGED$'* → *LikelihoodTransformerBase*

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

class aide_predict.bespoke_models.predictors.pretrained_transformers.**MarginalMethod**
value

Bases: Enum

An enumeration.

MASKED = 'masked_marginal'

MUTANT = 'mutant_marginal'

WILDTYPE = 'wildtype_marginal'

class aide_predict.bespoke_models.predictors.pretrained_transformers.**ModelDeviceManager**
model_instance: Any, *device: str = 'cpu'*

Bases: object

classmethod get_instance
model_instance: Any, *device: str*

model_on_device
load_func: Callable[[], None], *cleanup_func: Callable[[], None]*

aide_predict.bespoke_models.predictors.pretrained_transformers.**model_device_context**
model_instance: Any, *load_func: Callable[[], None]*, *cleanup_func: Callable[[], None]*, *device: str = 'cpu'*

Context manager used to load and clean up a model on a specific device.

This ensures model weights are not sitting on the GPU when not being accessed, unless the KEEP_MODEL_ON_DEVICE environment variable is set to True. If set to True, the model is loaded only once and kept on the device across multiple calls.

aide_predict.bespoke_models.predictors.saprot module

- Author: Evan Komp
- Created: 7/16/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Wrapper around SaProt model. Please see here and all credit to the oroginal authors for their method and model: https://www.biorxiv.org/content/10.1101/2023.10.01.560349v2

class aide_predict.bespoke_models.predictors.saprot.**SaProtLikelihoodWrapper**
metadata_folder: str | None = None, *model_checkpoint: str = 'westlake-repl/SaProt_650M_AF2'*, *marginal_method:* MarginalMethod *= MarginalMethod.WILDTYPE*, *positions: list | None = None*, *pool: bool = True*, *flatten: bool = True*, *wt: str | None = None*, *batch_size: int = 2*, *device: str = 'cpu'*, *foldseek_path: str = 'foldseek'*

Bases: *RequiresStructureMixin*, *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 = '$UNCHANGED$'* → *SaProtLikelihoodWrapper*

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 = '$UNCHANGED$'* → *SaProtLikelihoodWrapper*

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.saprot.**get_structure_tokens**(*structure: ProteinStructure*, *foldseek_path: str*, *process_id: int = 0*, *plddt_threshold: float = 70.0*, *return_seq_tokens: bool = False*) → str

Extract structure tokens from a ProteinStructure using FoldSeek.

Parameters

- **structure** (*ProteinStructure*) – The protein structure to process.
- **foldseek_path** (*str*) – Path to the FoldSeek executable.
- **process_id** (*int*) – Process ID for temporary files. Used for parallel processing.
- **plddt_threshold** (*float*) – Threshold for pLDDT scores. Regions below this are masked.

Returns

A string of structure tokens.

Return type

str

aide_predict.bespoke_models.predictors.vespa module

- Author: Evan Komp
- Created: 8/1/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Wrapper of VESPA: Marquet, C. et al. Embeddings from protein language models predict conservation and variant effects. Hum Genet 141, 1629–1647 (2022).

This model embeds the sequences with a PLM, then uses the embeddings for a pretrained logistic regression model for conservation. These are input into a model to predict single mutation effects.

class aide_predict.bespoke_models.predictors.vespa.**VESPAWrapper**
metadata_folder: str | None = None, *wt: str | ProteinSequence | None = None*, *light: bool = True*

Bases: *CanRegressMixin*, *RequiresWTDuringInferenceMixin*, *RequiresWTToFunctionMixin*, *ProteinModelWrapper*

A wrapper class for the VESPA (Variant Effect Score Prediction using Attention) model.

This class provides an interface to use VESPA within the AIDE framework, allowing for prediction of variant effects on protein sequences.

light
If True, uses the lighter VESPAl model. If False, uses the full VESPA model.

Type
bool

get_feature_names_out
input_features: List[str] | None = None → List[str]

Get the names of the output features.

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

Returns
A list containing the name of the output feature.

Return type
List[str]

set_fit_request
**, force: bool | None | str = '$UNCHANGED$'* → *VESPAWrapper*

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 = '$UNCHANGED$'* → *VESPAWrapper*

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.

Module contents

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

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

Subpackages

aide_predict.utils.data_structures package

Submodules

aide_predict.utils.data_structures.sequences module

- Author: Evan Komp
- Created: 6/21/2024
- Company: Bottle Institute @ National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Base data structures for the AIDE Predict package Where they do not exist in sklearn.

class aide_predict.utils.data_structures.sequences.**ProteinCharacter**
seq: str

Bases: `str`

Represents a single character in a protein sequence.

This class inherits from UserString and provides additional properties to check the nature of the amino acid character.

property is_gap: bool

Check if the character represents a gap in the sequence.

property is_non_canonical: bool

Check if the character represents a non-canonical amino acid.

property is_not_focus: bool

Check if the character is not in focus.

A character is considered not in focus if it's a gap or a lowercase letter.

class aide_predict.utils.data_structures.sequences.**ProteinSequence**
seq: str, *id: str | None = None*, *structure: str | ProteinStructure | None = None*

Bases: `str`

Represents a protein sequence.

This class inherits from UserString and provides additional methods and properties for analyzing and manipulating protein sequences.

align
other: ProteinSequence → *ProteinSequence*

Align this sequence with another using global pairwise alignment.

Parameters
other (ProteinSequence) – The sequence to align with.

Returns
The aligned sequence.

Return type
ProteinSequence

`property as_array: ndarray`

Convert the sequence to a numpy array of characters.

`property base_length: int`

Get the length of the sequence excluding gaps.

`get_mutations`

other: str | ProteinSequence → List[str]

Find mutations between this sequence and another.

Parameters
other (*Union[str, ProteinSequence]*) – The sequence to compare against.

Returns
A list of mutations in the format 'A123B' where A is the original character, 123 is the position, and B is the new character.

Return type
List[str]

`get_protein_character`

position: int → *ProteinCharacter*

Get the ProteinCharacter at the specified position.

Parameters
position (*int*) – The position to get the character from.

Returns
The character at the specified position.

Return type
ProteinCharacter

`property has_gaps: bool`

Check if the sequence contains any gaps.

`property has_non_canonical: bool`

Check if the sequence contains any non-canonical amino acids.

`property id: str | None`

Get the identifier of the sequence.

`iter_protein_characters`

→ Iterator[*ProteinCharacter*]

Iterate over the ProteinCharacters in the sequence.

Returns
An iterator over the ProteinCharacters.

Return type
Iterator[*ProteinCharacter*]

`mutate`

mutations: str | List[str], *one_indexed: bool = True*

Create a new ProteinSequence with mutations applied.

Params

mutations: Union[str, List[str]]
A single mutation in the format 'A123B' or a list of mutations.

one_indexed: bool
If True, positions are one-indexed. If False, positions are zero-indexed.

`mutated_positions`
other: str | ProteinSequence → List[int]

Find positions where this sequence differs from another.

Parameters
other (*Union[str, ProteinSequence]*) – The sequence to compare against.

Returns
A list of positions where the sequences differ.

Return type
List[int]

`property num_gaps: int`
Get the number of gaps in the sequence.

`saturation_mutagenesis`
positions: List[int] | None = None → List[*ProteinSequence*]

Perform saturation mutagenesis at the specified positions.

Parameters
positions (*List[int]*) – The positions to mutate.

Returns
A list of mutated sequences.

Return type
ProteinSequences

`slice_as_protein_sequence`
start: int, *end: int* → *ProteinSequence*

Create a new ProteinSequence from a slice of this sequence.

Parameters

- **start** (*int*) – The start position of the slice.
- **end** (*int*) – The end position of the slice.

Returns
A new ProteinSequence containing the specified slice.

Return type
ProteinSequence

`property structure: str | None`
Get the structure of the sequence.

`upper`
→ *ProteinSequence*

Return a new ProteinSequence with all characters converted to uppercase.

`with_no_gaps`
→ *ProteinSequence*

Return a new ProteinSequence with all gaps removed.

class aide_predict.utils.data_structures.sequences.**ProteinSequences**
sequences: List[ProteinSequence], *weights: ndarray | None = None*

Bases: UserList

A collection of ProteinSequence objects with additional functionality.

aligned

True if all sequences have the same length, False otherwise.

Type
bool

fixed_length

True if all sequences have the same base length, False otherwise.

Type
bool

width

The length of the sequences if aligned, None otherwise.

Type
Optional[int]

has_gaps

True if any sequence has gaps, False otherwise.

Type
bool

mutated_positions

List of mutated positions if aligned, None otherwise.

Type
Optional[List[int]]

to_dict

Convert ProteinSequences to a dictionary.

to_fasta

Write sequences to a FASTA file.

from_fasta

Create a ProteinSequences object from a FASTA file.

align_all
output_fasta: str | None = None → *ProteinSequences* | *ProteinSequencesOnFile*

Align the sequences within this ProteinSequences object using MAFFT.

Parameters
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
Union[*ProteinSequences*, *ProteinSequencesOnFile*]

Raises

- **ValueError** – If the sequences already contain gaps.
- **RuntimeError** – If MAFFT alignment fails.
- **FileNotFoundError** – If MAFFT is not installed or not in PATH.

`align_to`

existing_alignment: ProteinSequences | ProteinSequencesOnFile, *realign: bool = False*, *return_only_new: bool = False*, *output_fasta: str | None = None* → *ProteinSequences* | *ProteinSequencesOnFile*

Align this ProteinSequences object to an existing alignment using MAFFT.

Parameters

- **existing_alignment** (*Union[ProteinSequences, ProteinSequencesOnFile]*) – The existing alignment to align to.
- **realign** (*bool*) – If True, realign all sequences from scratch. If False, add new sequences to existing alignment. return_only_new (bool): If True, return only the newly aligned sequences. If False, return all sequences.
- **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

Union[*ProteinSequences*, *ProteinSequencesOnFile*]

Raises

- **ValueError** – If the sequences already contain gaps or if the existing alignment is not aligned.
- **RuntimeError** – If MAFFT alignment fails.
- **FileNotFoundError** – If MAFFT is not installed or not in PATH.

`property aligned: bool`

Check if all sequences are of equal length (including gaps).

Returns

True if all sequences have the same length, False otherwise.

Return type

bool

`apply_alignment_mapping`

mapping: Dict[str, List[int | None]] → *ProteinSequences*

Apply an alignment mapping to the current sequences.

Parameters

mapping (*Dict[str, List[Optional[int]]]*) – The alignment mapping to apply.

Returns

A new ProteinSequences object with aligned sequences.

Return type

ProteinSequences

Raises
ValueError – If a sequence ID or hash is not found in the mapping or if the mapping is invalid.

`as_array`
→ ndarray

Convert the sequence to a numpy array of characters.

`property fixed_length: bool`

Check if all contained sequences have the same base length (excluding gaps).

Returns
True if all sequences have the same base length, False otherwise.

Return type
bool

`classmethod from_dict`
sequences: Dict[str, str] → *ProteinSequences*

Create a ProteinSequences object from a dictionary.

Parameters
sequences (*Dict[str, str]*) – A dictionary with sequence IDs as keys and sequences as values.

Returns
A new ProteinSequences object containing the sequences from the dictionary.

Return type
ProteinSequences

`classmethod from_fasta`
input_path: str → *ProteinSequences*

Create a ProteinSequences object from a FASTA file.

Parameters
input_path (*str*) – The path to the input FASTA file.

Returns
A new ProteinSequences object containing the sequences from the FASTA file.

Return type
ProteinSequences

`classmethod from_list`
sequences: List[str] → *ProteinSequences*

Create a ProteinSequences object from a list of sequences.

Parameters
sequences (*List[str]*) – A list of protein sequences.

Returns
A new ProteinSequences object containing the sequences from the list.

Return type
ProteinSequences

`get_alignment_mapping`
→ Dict[str, List[int | None]]

Create a mapping of original sequence positions to aligned positions for each sequence.

Returns
A dictionary where keys are sequence IDs or hashes and values are lists of integers. Each integer represents the position in the aligned sequence corresponding to the original sequence position. E.g., [0,1,2,5,6,7] indicates that there is a gap between amino acid 2 and 3, and 3 is in position 5 in the aligned sequence.

Return type
Dict[str, List[Optional[int]]]

Raises
ValueError – If the sequences are not aligned.

`get_id_mapping`
→ Dict[str, int]

Create a mapping of sequence IDs to indices.

Returns
A dictionary where keys are sequence IDs and values are indices.

Return type
Dict[str, int]

`property has_gaps: bool`
Check if any sequences have gaps.

Returns
True if any sequence has gaps, False otherwise.

Return type
bool

`has_lower`
→ bool

Check if any sequence contains lowercase characters.

`property id_mapping: Dict[str, int]`

`property ids: List[str]`
Get a list of sequence IDs.

`iter_batches`
batch_size: int → Iterable[*ProteinSequences*]

Iterate over batches of sequences.

Parameters
batch_size (*int*) – The size of each batch.

Yields
ProteinSequences – A batch of sequences.

`msa_process`
focus_seq_id: str | None = None, ***kwargs* → *ProteinSequence*

Align this sequence with another using global pairwise alignment.

Kwargs:
**kwargs: Additional arguments to pass to MSAprocessing

Returns
The aligned sequence.

Return type
ProteinSequence

`property mutated_positions: List[int] | None`

List columns that have more than one character, assuming sequences are aligned.

Returns
List of mutated positions if aligned, None otherwise.

Return type
Optional[List[int]]

`sample`

n: int, *replace: bool = False* → *ProteinSequences*

Sample n sequences from the ProteinSequences object.

Parameters

- **n** (*int*) – Number of sequences to sample.
- **replace** (*bool*) – Whether to sample with replacement. Default is False.

Returns
A new ProteinSequences object containing the sampled sequences.

Return type
ProteinSequences

Raises
ValueError – If n is greater than the number of sequences and replace is False.

`to_dict`

→ Dict[str, str]

Convert ProteinSequences to a dictionary.

Returns
A dictionary with sequence IDs as keys and sequences as values.

Return type
Dict[str, str]

`to_fasta`

output_path: str

Write sequences to a FASTA file.

Parameters
output_path (*str*) – The path to the output FASTA file.

`to_on_file`

output_path: str → None

Write sequences to a FASTA file.

Parameters
output_path (*str*) – The path to the output FASTA file.

`upper`

→ *ProteinSequences*

Return a new ProteinSequences with all sequences converted to uppercase.

`property weights: ndarray`

Get the weights for each sequence.

property width: int | None

Get the length of the sequences if aligned.

Returns
The length of the sequences if aligned, None otherwise.

Return type
Optional[int]

with_no_gaps

→ *ProteinSequences*

Return a new ProteinSequences with all gaps removed.

class aide_predict.utils.data_structures.sequences.**ProteinSequencesOnFile**

file_path: str, *weights: ndarray | None = None*

Bases: *ProteinSequences*

A memory-efficient representation of protein sequences stored in a FASTA file.

This class maintains the same API as ProteinSequences but avoids loading all sequences into memory at once. It creates an index of the FASTA file for efficient access to individual sequences and precomputes some global properties for quick access.

aligned

True if all sequences have the same length, False otherwise.

Type
bool

fixed_length

True if all sequences have the same base length, False otherwise.

Type
bool

width

The length of the sequences if aligned, None otherwise.

Type
Optional[int]

has_gaps

True if any sequence has gaps, False otherwise.

Type
bool

mutated_positions

List of mutated positions if aligned, None otherwise.

Type
Optional[List[int]]

to_dict

Convert ProteinSequences to a dictionary.

to_fasta

Write sequences to a FASTA file.

`from_fasta`

Create a ProteinSequences object from a FASTA file.

`property aligned: bool`

Check if all sequences are of equal length (including gaps).

Returns
True if all sequences have the same length, False otherwise.

Return type
bool

`property fixed_length: bool`

Check if all contained sequences have the same base length (excluding gaps).

Returns
True if all sequences have the same base length, False otherwise.

Return type
bool

`classmethod from_dict`

sequences: Dict[str, str] → *ProteinSequences*

Create a ProteinSequences object from a dictionary.

Parameters
sequences (*Dict[str, str]*) – A dictionary with sequence IDs as keys and sequences as values.

Returns
A new ProteinSequences object containing the sequences from the dictionary.

Return type
ProteinSequences

`classmethod from_fasta`

input_path: str → *ProteinSequencesOnFile*

Create a ProteinSequencesOnFile object from a FASTA file.

Parameters
input_path (*str*) – The path to the input FASTA file.

Returns
A new ProteinSequencesOnFile object.

Return type
ProteinSequencesOnFile

`property has_gaps: bool`

Check if any sequences have gaps.

Returns
True if any sequence has gaps, False otherwise.

Return type
bool

`property ids: List[str]`

Get a list of sequence IDs.

`iter_batches`

batch_size: int → Iterable[*ProteinSequences*]

Iterate over batches of sequences.

Parameters
batch_size (*int*) – The size of each batch.

Yields
ProteinSequences – A batch of sequences.

property mutated_positions: List[int] | None

List columns that have more than one character, assuming sequences are aligned.

Returns
List of mutated positions if aligned, None otherwise.

Return type
Optional[List[int]]

to_dict

→ Dict[str, str]

Convert sequences to a dictionary.

Returns
A dictionary with sequence IDs as keys and sequences as values.

Return type
Dict[str, str]

to_fasta

output_path: str → None

Write sequences to a FASTA file.

Parameters
output_path (*str*) – The path to the output FASTA file.

to_memory

→ *ProteinSequences*

Load all sequences into memory as a ProteinSequences object.

Returns
A new ProteinSequences object containing all sequences.

Return type
ProteinSequences

property width: int | None

Get the length of the sequences if aligned.

Returns
The length of the sequences if aligned, None otherwise.

Return type
Optional[int]

aide_predict.utils.data_structures.structures module

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

class aide_predict.utils.data_structures.structures.**ProteinStructure**
pdb_file: str, *chain: str = 'A'*, *plddt_file: str | None = None*

Bases: `object`

chain: str = 'A'

classmethod from_af2_folder
folder_path: str, *chain: str = 'A'* → *ProteinStructure*

Create a ProteinStructure object from an AlphaFold2 prediction folder.

This method prioritizes the top-ranked relaxed structure. If no relaxed structures are available, it selects the top-ranked unrelaxed structure.

Parameters

- **folder_path** (*str*) – Path to the folder containing AlphaFold2 predictions.
- **chain** (*str*) – Chain identifier (default is 'A').

Returns
A new ProteinStructure object.

Return type
ProteinStructure

Raises
FileNotFoundError – If no suitable PDB file is found in the folder.

get_chain
→ <module 'Bio.PDB.Chain' from '/Users/ekomp/miniconda3/envs/aidep/lib/python3.9/site-packages/Bio/PDB/Chain.py'>

Load and return the specified chain.

Returns
The specified protein chain.

Return type
Chain

get_dssp
→ Dict[str, str]

Get the DSSP secondary structure assignments.

Returns
Dictionary of DSSP assignments.

Return type
Dict[str, str]

get_plddt
→ ndarray | None

Get the pLDDT scores if available.

Returns
Array of pLDDT scores or None if not available.

Return type
Optional[np.ndarray]

`get_residue_positions`
→ List[int]

Get the residue positions present in the structure.

Returns
List of residue positions.

Return type
List[int]

`get_sequence`
→ str

Get the amino acid sequence from the PDB file.

Returns
The amino acid sequence.

Return type
str

`get_structure`
→ <module 'Bio.PDB.Structure' from '/Users/ekomp/miniconda3/envs/aidep/lib/python3.9/site-packages/Bio/PDB/Structure.py'>

Load and return the complete structure.

Returns
The complete protein structure.

Return type
Structure

`pdb_file: str`

`plddt_file: str | None = None`

`validate_sequence`
protein_sequence: str → bool

Validate if the given sequence matches the structure's sequence.

Parameters
protein_sequence (*str*) – The sequence to validate.

Returns
True if the sequences match, False otherwise.

Return type
bool

`class aide_predict.utils.data_structures.structures.StructureMapper`
structure_folder: str

Bases: `object`

A class for mapping protein structures to sequences based on files in a given folder.

This class scans a specified folder for PDB files and AlphaFold2 prediction folders, creates ProteinStructure objects, and can assign these structures to ProteinSequence or ProteinSequences objects based on their IDs.

structure_folder

The path to the folder containing structure files.

Type
str

structure_map

A dictionary mapping protein IDs to ProteinStructure objects.

Type
Dict[str, *ProteinStructure*]

assign_structures

sequences: ProteinSequence | ProteinSequences → *ProteinSequence* | *ProteinSequences*

Assign structures to the given protein sequence(s).

This method attempts to assign a structure to each protein sequence based on its ID. If a matching structure is found in the structure_map, it is assigned to the sequence.

Parameters
sequences (*Union['ProteinSequence', 'ProteinSequences']*) – The protein sequence(s) to assign structures to.

Returns
The input sequence(s) with structures assigned where possible.

Return type
Union['ProteinSequence', 'ProteinSequences']

Raises
ValueError – If the input is neither a ProteinSequence nor a ProteinSequences object.

get_available_structures

→ List[str]

Get a list of all available structure IDs.

Returns
A list of structure IDs available in the structure_map.

Return type
List[str]

get_protein_sequences

→ *ProteinSequences*

Get a ProteinSequences object containing all available protein sequences.

Returns
A ProteinSequences object containing all available protein sequences.

Return type
ProteinSequences

Module contents

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

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.conservation module

- Author: Evan Komp
- Created: 9/9/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

class aide_predict.utils.conservation.**ConservationAnalysis**
protein_sequences: ProteinSequences, *ignore_gaps: bool = True*

Bases: object

A class for analyzing amino acid property conservation in protein sequence alignments.

This class provides methods to compute conservation scores and their statistical significance for various amino acid properties across aligned protein sequences. It can also compare conservation between two alignments.

PROPERTIES
A dictionary mapping property names to sets of amino acids that possess that property.

Type
Dict[str, set]

EXPECTED_FREQUENCIES

A dictionary mapping property names to their expected frequencies based on the 20 standard amino acids.

Type
Dict[str, float]

Parameters

- **protein_sequences** (ProteinSequences) – An aligned set of protein sequences.
- **ignore_gaps** (*bool*) – Whether to ignore gaps in conservation calculations. Default is True.

Raises
ValueError – If the input ProteinSequences object is not aligned.

```
EXPECTED_FREQUENCIES = {'Aliphatic': 0.13636363636363635, 'Aromatic':
0.1818181818181818, 'Charged': 0.22727272727272727, 'Hydrophobic':
0.5909090909090909, 'Negative': 0.09090909090909091, 'Polar': 0.5909090909090909,
'Positive': 0.13636363636363635, 'Proline': 0.045454545454545456, 'Small':
0.4090909090909091, 'Tiny': 0.13636363636363635, 'not_Aliphatic':
0.7272727272727273, 'not_Aromatic': 0.8181818181818182, 'not_Charged':
0.7727272727272727, 'not_Hydrophobic': 0.4090909090909091, 'not_Negative':
0.9090909090909091, 'not_Polar': 0.4090909090909091, 'not_Positive':
0.8636363636363636, 'not_Proline': 0.9545454545454546, 'not_Small':
0.5909090909090909, 'not_Tiny': 0.8636363636363636}
```

```
PROPERTIES = {'Aliphatic': {'I', 'L', 'V'}, 'Aromatic': {'F', 'H', 'W', 'Y'},
'Charged': {'D', 'E', 'H', 'K', 'R'}, 'Hydrophobic': {'A', 'C', 'F', 'G', 'H',
'I', 'K', 'L', 'M', 'T', 'V', 'W', 'Y'}, 'Negative': {'D', 'E'}, 'Polar': {'B',
'D', 'E', 'H', 'K', 'N', 'Q', 'R', 'S', 'T', 'W', 'Y', 'Z'}, 'Positive': {'H', 'K',
'R'}, 'Proline': {'P'}, 'Small': {'A', 'C', 'D', 'G', 'N', 'P', 'S', 'T', 'V'},
'Tiny': {'A', 'G', 'S'}, 'not_Aliphatic': {'A', 'C', 'D', 'E', 'F', 'G', 'H', 'K',
'M', 'N', 'Q', 'R', 'S', 'T', 'W', 'Y'}, 'not_Aromatic': {'A', 'B', 'C', 'D', 'E',
'G', 'I', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'Z'}, 'not_Charged':
{'A', 'B', 'C', 'F', 'G', 'I', 'L', 'M', 'N', 'P', 'Q', 'S', 'T', 'V', 'W', 'Y',
'Z'}, 'not_Hydrophobic': {'B', 'D', 'E', 'N', 'P', 'Q', 'R', 'S', 'Z'},
'not_Negative': {'A', 'B', 'C', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q',
'R', 'S', 'T', 'V', 'W', 'Y', 'Z'}, 'not_Polar': {'A', 'C', 'F', 'G', 'I', 'L',
'M', 'P', 'V'}, 'not_Positive': {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'I', 'L', 'M',
'N', 'P', 'Q', 'S', 'T', 'V', 'W', 'Y', 'Z'}, 'not_Proline': {'A', 'B', 'C', 'D',
'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'Q', 'R', 'S', 'T', 'V', 'W', 'Y',
'Z'}, 'not_Small': {'B', 'E', 'F', 'H', 'I', 'K', 'L', 'M', 'Q', 'R', 'W', 'Y',
'Z'}, 'not_Tiny': {'B', 'C', 'D', 'E', 'F', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q',
'R', 'T', 'V', 'W', 'Y', 'Z'}}
```

static compare_alignments

alignment1: ProteinSequences, *alignment2:* ProteinSequences, *ignore_gaps: bool = True*, *alpha: float = 0.01* → Tuple[Dict[str, ndarray], Dict[str, ndarray]]

Compare conservation scores between two alignments and compute statistical significance.

Parameters

- **alignment1** (ProteinSequences) – The first aligned set of protein sequences.
- **alignment2** (ProteinSequences) – The second aligned set of protein sequences.

- **ignore_gaps** (*bool*) – Whether to ignore gaps in conservation calculations. Default is True.
- **alpha** (*float*) – The significance level for the binomial test. Default is 0.01.

Returns

A tuple containing:

1. A dictionary mapping property names to arrays of conservation score differences.
2. A dictionary mapping property names to arrays of p-values for the differences.

Return type
Tuple[Dict[str, np.ndarray], Dict[str, np.ndarray]]

Raises
ValueError – If the two alignments have different lengths.

compute_conservation
→ Dict[str, ndarray]

Compute conservation scores for each amino acid property across all alignment positions.

Returns

A dictionary mapping property names to arrays of conservation
scores. Each array has a length equal to the alignment width, with values between 0 and 1 representing the fraction of sequences that have the property at each position.

Return type
Dict[str, np.ndarray]

compute_significance
alpha: float = 0.01 → Dict[str, ndarray]

Compute the statistical significance of conservation for each property and position.

This method uses a binomial test to compare the observed frequency of each property to its expected frequency based on amino acid composition.

Parameters
alpha (*float, optional*) – The significance level for the binomial test. Defaults to 0.01.

Returns

A tuple containing:

1. A boolean array indicating significant positions (True if any property is significant).
2. A dictionary mapping property names to arrays of p-values for each position.

Return type
Tuple[np.ndarray, Dict[str, np.ndarray]]

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.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

aide_predict.utils.plotting module

- Author: Evan Komp
- Created: 7/26/2024
- Company: National Renewable Energy Lab, Bioeneergy Science and Technology
- License: MIT

Common plotting calls.

aide_predict.utils.plotting.**plot_conservation**
conservation_scores: Dict[str, ndarray], p_values: Dict[str, ndarray] | None = None, alpha: float = 1e-10, stacked: bool = False, figsize: tuple = (20, 6), title: str = 'Conservation Scores Across Alignment Positions' → Figure

Create a bar plot of conservation scores across alignment positions.

Parameters

- **conservation_scores** (*Dict[str, np.ndarray]*) – Dictionary of conservation scores for each property.
- **p_values** (*Optional[Dict[str, np.ndarray]]*) – Dictionary of p-values for each property. If provided, insignificant bars will be colored grey.
- **alpha** (*float*) – Significance level for p-values. Default is 0.05.
- **stacked** (*bool*) – If True, create a stacked bar plot with colors for different properties. If False, create a single bar plot with height determined by sum of conservation scores.
- **figsize** (*tuple*) – Figure size (width, height) in inches. Default is (12, 6).
- **title** (*str*) – Title of the plot. Default is "Conservation Scores Across Alignment Positions".

Returns
The matplotlib Figure object containing the plot.

Return type
plt.Figure

aide_predict.utils.plotting.**plot_mutation_heatmap**
mutations, *scores*

Plot a heatmap of single point mutation scores.

Parameters: mutations (list): List of mutation strings (e.g., [“L1V”, “A2G”, ...]) scores (list): List of corresponding scores

Returns: None (displays the plot)

aide_predict.utils.plotting.**plot_protein_sequence_heatmap**
sequences: ProteinSequences, *figsize: tuple = (20, 5)*, *cmap: str = 'viridis'*, *title: str = 'Protein Sequence Heatmap'* → Figure

Create a heatmap visualization of protein sequences with additional sequence properties.

Parameters

- **sequences** (ProteinSequences) – A ProteinSequences object containing the protein sequences.
- **figsize** (*tuple*) – Figure size (width, height) in inches.
- **cmap** (*str*) – Colormap to use for the heatmap.
- **title** (*str*) – Title of the plot.

Returns
The matplotlib Figure object containing the heatmap.

Return type
plt.Figure

Module contents

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

Submodules

aide_predict.patches_ module

aide_predict.patches_.**patch_pandas_append**

aide_predict.patches_.**patched_parse_plmc_log**
log

A patched version of parse_plmc_log that handles the new output format.

Module contents

- Author: Evan Komp
- Created: 5/7/2024
-

PYTHON MODULE INDEX

a

INDEX

Symbols

A

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W