MHCflurry Documentation

Release 1.6.0

Timothy O'Donnell

Mar 23, 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION AND SETUP

MHCflurry is an open source package for peptide/MHC I binding affinity prediction. It aims to provide competitive accuracy with a fast and documented implementation.

You can download pre-trained MHCflurry models fit to mass spec-identified MHC I ligands and peptide/MHC affinity measurements deposited in IEDB (plus a few other sources) or train a MHCflurry predictor on your own data.

Starting in version 1.6.0, the default MHCflurry binding affinity predictors are "pan-allele" models that support most sequenced MHC I alleles across humans and a few other species (about 14,000 alleles in total). This version also introduces two experimental predictors, an "antigen processing" predictor that attempts to model MHC allele-independent effects such as proteosomal cleavage and a "presentation" predictor that integrates processing predictions with binding affinity predictions to give a composite "presentation score." Both models are trained on mass spec-identified MHC ligands.

MHCflurry supports Python 3.4+. It uses the keras neural network library via either the Tensorflow or Theano backends. GPUs may optionally be used for a modest speed improvement.

If you find MHCflurry useful in your research please cite:

> T. J. O'Donnell, et al., "MHCflurry: Open-Source Class I MHC Binding Affinity Prediction," *Cell Systems*, 2018. https://www.cell.com/cell-systems/fulltext/S2405-4712(18)30232-1.

If you have questions or encounter problems, please file an issue at the MHCflurry github repo: https://github.com/openvax/mhcflurry

1.1 Installation (pip)

Install the package:

```
$ pip install mhcflurry
```

Then download our datasets and trained models:

```
$ mhcflurry-downloads fetch
```

From a checkout you can run the unit tests with:

```
$ pip install nose
$ nosetests .
```

1.2 Using conda

You can alternatively get up and running with a conda environment as follows. Some users have reported that this can avoid problems installing tensorflow.

```
$ conda create -q -n mhcflurry-env python=3.6 'tensorflow<2.0.0'
$ source activate mhcflurry-env
```

Then continue as above:

```
$ pip install mhcflurry
$ mhcflurry-downloads fetch
```

CHAPTER

TWO

COMMAND-LINE TUTORIAL

2.1 Downloading models

Most users will use pre-trained MHCflurry models that we release. These models are distributed separately from the pip package and may be downloaded with the *mhcflurry-downloads* tool:

```
$ mhcflurry-downloads fetch models_class1_presentation
```

Files downloaded with *mhcflurry-downloads* are stored in a platform-specific directory. To get the path to downloaded data, you can use:

```
$ mhcflurry-downloads path models_class1_presentation
/Users/tim/Library/Application Support/mhcflurry/4/1.6.0/models_class1_presentation/
```

We also release a number of other "downloads," such as curated training data and some experimental models. To see what's available and what you have downloaded, run `mhcflurry-downloads info`.

Most users will only need `models_class1_presentation`, however, as the presentation predictor includes a peptide / MHC I binding affinity (BA) predictor as well as an antigen processing (AP) predictor.

Note: The code we use for *generating* the downloads is in the `downloads_generation` directory in the repository (https://github.com/openvax/mhcflurry/tree/master/downloads-generation)

2.2 Generating predictions

The *mhcflurry-predict* command generates predictions for individual peptides (see the next section for how to scan protein sequences for epitopes). By default it will use the pre-trained models you downloaded above. Other models can be used by specifying the `--models` argument.

Running:

```
$ mhcflurry-predict
    --alleles HLA-A0201 HLA-A0301
    --peptides SIINFEKL SIINFEKD SIINFEKQ
    --out /tmp/predictions.csv
Predicting processing.
Predicting affinities.
Wrote: /tmp/predictions.csv
```

results in a file like this:

```
$ cat /tmp/predictions.csv
allele,peptide,mhcflurry_affinity,mhcflurry_affinity_percentile,mhcflurry_processing_
↪score,mhcflurry_presentation_score
HLA-A0201,SIINFEKL,12906.786866783777,6.566374999999997,0.10147304146084934,0.
↪012502709574418581
HLA-A0201,SIINFEKD,34491.778276696794,44.61275000000001,0.015448467253008857,0.
↪00387464531700359
HLA-A0201,SIINFEKQ,31476.142209308568,30.217625,0.0359264743165113,0.
↪004518140004135804
HLA-A0301,SIINFEKL,32253.5874797359,20.37875,0.10147304146084934,0.0055786327707357695
HLA-A0301,SIINFEKD,37075.86269121571,58.8585,0.015448467253008857,0.
↪0036345557828772576
HLA-A0301,SIINFEKQ,33258.4276356094,24.029125,0.0359264743165113,0.004303226784859033
```

The binding affinity predictions are given as affinities (KD) in nM in the `mhcflurry_affinity` column. Lower values indicate stronger binders. A commonly-used threshold for peptides with a reasonable chance of being immunogenic is 500 nM.

The `mhcflurry_affinity_percentile` gives the percentile of the affinity prediction among a large number of random peptides tested on that allele (range 0 - 100). Lower is stronger. Two percent is a commonly-used threshold.

The last two columns give the antigen processing and presentation scores, respectively. These range from 0 to 1 with higher values indicating more favorable processing or presentation.

Note: The processing predictor is experimental. It models allele-independent effects that influence whether a peptide will be detected in a mass spec experiment. The presentation score is a simple logistic regression model that combines the (log) binding affinity prediction with the processing score to give a composite prediction. The resulting prediction may be useful for prioritizing potential epitopes, but no thresholds have been established for what constitutes a "high enough" presentation score.

In most cases you'll want to specify the input as a CSV file instead of passing peptides and alleles as commandline arguments. If you're relying on the processing or presentation scores, you may also want to pass the upstream and downstream sequences of the peptides from their source proteins for potentially more accurate cleavage prediction. See the *mhcflurry-predict* docs.

2.3 Scanning protein sequences for predicted MHC I ligands

Starting in version 1.6.0, MHCflurry supports scanning proteins for MHC-binding peptides using the `mhcflurry-predict-scan` command.

We'll generate predictions across `example.fasta`, a FASTA file with two short sequences:

```
>protein1
MSSSSTPVCPNGPGNCQV
>protein2
MVENKRLLEGMEMIFGQVIPGA
```

Here's the `mhcflurry-predict-scan` invocation to scan the proteins for binders to either of two MHC I genotypes (using a 100 nM threshold):

```
$ mhcflurry-predict-scan
    example.fasta
    --alleles
```

(continues on next page)

(continued from previous page)

```
        HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:02,HLA-C*07:02
        HLA-A*01:01,HLA-A*02:06,HLA-B*44:02,HLA-B*07:02,HLA-C*01:02,HLA-C*03:01
    --results-filtered affinity
    --threshold-affinity 100
Guessed input file format: fasta
Read input fasta with 2 sequences
  sequence_id               sequence
0    protein1      MSSSSTPVCPNGPGNCQV
1    protein2  MVENKRLLEGMEMIFGQVIPGA
Predicting processing.
Predicting affinities.
sequence_name,pos,peptide,n_flank,c_flank,sample_name,affinity,best_allele,affinity_
↪percentile,processing_score,presentation_score
protein2,5,RLLEGMEMI,MVENK,FGQVIPGA,genotype_00,15.506647240532306,HLA-A*02:01,0.
↪06800000000000002,0.9045842811465263,0.9899999181261543
protein2,5,RLLEGMEMI,MVENK,FGQVIPGA,genotype_01,17.618050003789662,HLA-A*02:06,0.
↪09587500000000006,0.9045842811465263,0.9888061128232571
protein2,12,MIFGQVIPGA,MVENKRLLEGME,,genotype_01,27.74054476375278,HLA-A*02:06,0.
↪2527499999999999,0.9025756493210793,0.9831738588268119
protein2,4,KRLLEGMEM,MVEN,IFGQVIPGA,genotype_00,29.368389637130125,HLA-C*07:02,0.
↪09150000000000004,0.49434878677129745,0.9171519649768841
protein2,12,MIFGQVIPGA,MVENKRLLEGME,,genotype_00,35.083004993507345,HLA-A*02:01,0.
↪34662499999999985,0.9025756493210793,0.9793299707138564
protein1,8,CPNGPGNCQV,MSSSSTPV,,genotype_01,44.167261275374614,HLA-B*07:02,0.236125,0.
↪16842720657587051,0.6796971953334389
protein2,10,MEMIFGQVI,MVENKRLLEG,PGA,genotype_00,48.90184726168596,HLA-B*45:01,0.
↪180375,0.40690354630351067,0.8325127154379427
protein2,10,MEMIFGQVI,MVENKRLLEG,PGA,genotype_01,77.43363203601268,HLA-B*44:02,0.
↪3624999999999998,0.40690354630351067,0.7673069170016388
protein2,4,KRLLEGMEMIF,MVEN,GQVIPGA,genotype_00,88.06509237659311,HLA-C*07:02,0.
↪8623749999999999,0.46066924184560776,0.7842639917310846
```

See the *mhcflurry-predict-scan* docs for more options.

2.4 Fitting your own models

If you have your own data and want to fit your own MHCflurry models, you have a few options. If you have data for only one or a few MHC I alleles, the best approach is to use the *mhcflurry-class1-train-allele-specific-models* command to fit an "allele-specific" predictor, in which separate neural networks are used for each allele.

To call *mhcflurry-class1-train-allele-specific-models* you'll need some training data. The data we use for our released predictors can be downloaded with *mhcflurry-downloads*:

```
$ mhcflurry-downloads fetch data_curated
```

It looks like this:

```
$ bzcat "$(mhcflurry-downloads path data_curated)/curated_training_data.csv.bz2" |␣
↪head -n 3
allele,peptide,measurement_value,measurement_inequality,measurement_type,measurement_
↪kind,measurement_source,original_allele
BoLA-1*21:01,AENDTLVVSV,7817.0,=,quantitative,affinity,Barlow - purified MHC/
↪competitive/fluorescence,BoLA-1*02101
BoLA-1*21:01,NQFNGGCLLV,1086.0,=,quantitative,affinity,Barlow - purified MHC/direct/
↪fluorescence,BoLA-1*02101
```

Here's an example invocation to fit a predictor:

```
$ mhcflurry-class1-train-allele-specific-models \
    --data curated_training_data.csv.bz2 \
    --hyperparameters hyperparameters.yaml \
    --min-measurements-per-allele 75 \
    --out-models-dir models
```

The `hyperparameters.yaml` file gives the list of neural network architectures to train models for. Here's an example specifying a single architecture:

```
- activation: tanh
  dense_layer_l1_regularization: 0.0
  dropout_probability: 0.0
  early_stopping: true
  layer_sizes: [8]
  locally_connected_layers: []
  loss: custom:mse_with_inequalities
  max_epochs: 500
  minibatch_size: 128
  n_models: 4
  output_activation: sigmoid
  patience: 20
  peptide_amino_acid_encoding: BLOSUM62
  random_negative_affinity_max: 50000.0
  random_negative_affinity_min: 20000.0
  random_negative_constant: 25
  random_negative_rate: 0.0
  validation_split: 0.1
```

The available hyperparameters for binding predictors are defined in *Class1NeuralNetwork*. To see exactly how these are used you will need to read the source code.

Note: MHCflurry predictors are serialized to disk as many files in a directory. The model training command above will write the models to the output directory specified by the `--out-models-dir` argument. This directory has files like:

```
info.txt
manifest.csv
model_selection.csv.bz2
model_selection_data.csv.bz2
...
weights_PATR-B*24:01-6-da42511a2164a8fe.npz
weights_PATR-B*24:01-7-4e3f5ded9cc1d851.npz
weights_PATR-B*24:01-8-491d1b4d85da0dc4.npz
weights_PATR-B*24:01-9-8f295e814502ffa1.npz
```

The `manifest.csv` file gives metadata for all the models used in the predictor. There will be a `weights_...` file for each model giving its weights (the parameters for the neural network). The `percent_ranks.csv` stores a histogram of model predictions for each allele over a large number of random peptides. It is used for generating the percent ranks at prediction time.

To fit pan-allele models like the ones released with MHCflurry, you can use a similar tool, *mhcflurry-class1-train-pan-allele-models*. You'll probably also want to take a look at the scripts used to generate the production models, which are available in the *downloads-generation* directory in the MHCflurry repository. See the scripts in the *models_class1_pan* subdirectory to see how the fitting and model selection was done for models currently distributed with MHCflurry.

Note: The production MHCflurry models were fit using a cluster with several dozen GPUs over a period of about two days. If you model select over fewer architectures, however, it should be possible to fit a predictor using less resources.

2.5 Environment variables

MHCflurry behavior can be modified using these environment variables:

MHCFLURRY_DEFAULT_CLASS1_MODELS Path to models directory. If you call `Class1AffinityPredictor.load()` with no arguments, the models specified in this environment variable will be used. If this environment variable is undefined, the downloaded models for the current MHCflurry release are used.

MHCFLURRY_OPTIMIZATION_LEVEL The pan-allele models can be somewhat slow. As an optimization, when this variable is greater than 0 (default is 1), we "stitch" the pan-allele models in the ensemble into one large tensorflow graph. In our experiments it gives about a 30% speed improvement. It has no effect on allele-specific models. Set this variable to 0 to disable this behavior. This may be helpful if you are running out of memory using the pan-allele models.

MHCFLURRY_DEFAULT_PREDICT_BATCH_SIZE For large prediction tasks, it can be helpful to increase the prediction batch size, which is set by this environment variable (default is 4096). This affects both allele-specific and pan-allele predictors. It can have large effects on performance. Alternatively, if you are running out of memory, you can try decreasing the batch size.

CHAPTER

THREE

PYTHON LIBRARY TUTORIAL

The MHCflurry Python API exposes additional options and features beyond those supported by the commandline tools and can be more convenient for interactive analyses and bioinformatic pipelines. This tutorial gives a basic overview of the most important functionality. See the *API Documentation* for further details.

3.1 Loading a predictor

Most prediction tasks can be performed using the `Class1PresentationPredictor` class, which provides a programmatic API to the functionality in the *mhcflurry-predict* and *mhcflurry-predict-scan* commands.

Instances of `Class1PresentationPredictor` wrap a `Class1AffinityPredictor` to generate binding affinity predictions and a `Class1ProcessingPredictor` to generate antigen processing predictions. The presentation score is computed using a logistic regression model over binding affinity and processing predictions.

Use the `load` static method to load a trained predictor from disk. With no arguments this method will load the predictor released with MHCflurry (see *Downloading models*). If you pass a path to a models directory, then it will load that predictor instead.

```
>>> from mhcflurry import Class1PresentationPredictor
>>> predictor = Class1PresentationPredictor.load()
>>> predictor.supported_alleles[:5]
['Atbe-B*01:01', 'Atbe-E*03:01', 'Atbe-G*03:01', 'Atbe-G*03:02', 'Atbe-G*06:01']
```

3.2 Predicting for individual peptides

To generate predictions for individual peptides, we can use the `predict` method of the `Class1PresentationPredictor`, loaded above. This method returns a `pandas.DataFrame` with binding affinity, processing, and presentation predictions:

```
>>> predictor.predict(
...     peptides=["SIINFEKL", "NLVPMVATV"],
...     alleles=["HLA-A0201", "HLA-A0301"],
...     verbose=0)
     peptide  peptide_num sample_name      affinity best_allele  processing_score␣
↪presentation_score
0   SIINFEKL            0     sample1  12906.786173   HLA-A0201          0.101473␣
↪       0.012503
1  NLVPMVATV            1     sample1     15.038358   HLA-A0201          0.676289␣
↪       0.975463
```

Here, the list of alleles is taken to be an individual's MHC I genotype (i.e. up to 6 alleles), and the strongest binder across alleles for each peptide is reported.

Note: MHCflurry normalizes allele names using the mhcnames package. Names like `HLA-A0201` or `A*02:01` will be normalized to `HLA-A*02:01`, so most naming conventions can be used with methods such as *`predict`*.

If you have multiple sample genotypes, you can pass a dict, where the keys are arbitrary sample names:

```
>>> predictor.predict(
...     peptides=["KSEYMTSWFY", "NLVPMVATV"],
...     alleles={
...        "sample1": ["A0201", "A0301", "B0702", "B4402", "C0201", "C0702"],
...        "sample2": ["A0101", "A0206", "B5701", "C0202"],
...     },
...     verbose=0)
      peptide  peptide_num sample_name      affinity best_allele  processing_score ␣
↪presentation_score
0  KSEYMTSWFY            0     sample1  16737.745268       A0301          0.381632   ␣
↪        0.026550
1   NLVPMVATV            1     sample1     15.038358       A0201          0.676289   ␣
↪        0.975463
2  KSEYMTSWFY            0     sample2     62.540779       A0101          0.381632   ␣
↪        0.796731
3   NLVPMVATV            1     sample2     15.765500       A0206          0.676289   ␣
↪        0.974439
```

Here the strongest binder for each sample / peptide pair is returned.

Many users will focus on the binding affinity predictions, as the processing and presentation predictions are experimental. If you do use the latter scores, however, when available you should provide the upstream (N-flank) and downstream (C-flank) sequences from the source proteins of the peptides for a small boost in accuracy. To do so, specify the `n_flank` and `c_flank` arguments, which give the flanking sequences for the corresponding peptides:

```
>>> predictor.predict(
...     peptides=["KSEYMTSWFY", "NLVPMVATV"],
...     n_flanks=["NNNNNNN", "SSSSSSSS"],
...     c_flanks=["CCCCCCCC", "YYYAAAA"],
...     alleles={
...        "sample1": ["A0201", "A0301", "B0702", "B4402", "C0201", "C0702"],
...        "sample2": ["A0101", "A0206", "B5701", "C0202"],
...     },
...     verbose=0)
      peptide   n_flank   c_flank  peptide_num sample_name      affinity best_allele ␣
↪processing_score  presentation_score
0  KSEYMTSWFY   NNNNNNN  CCCCCCCC            0     sample1  16737.745268       A0301 ␣
↪        0.605816            0.056190
1   NLVPMVATV  SSSSSSSS   YYYAAAA            1     sample1     15.038358       A0201 ␣
↪        0.824994            0.986719
2  KSEYMTSWFY   NNNNNNN  CCCCCCCC            0     sample2     62.540779       A0101 ␣
↪        0.605816            0.897493
3   NLVPMVATV  SSSSSSSS   YYYAAAA            1     sample2     15.765500       A0206 ␣
↪        0.824994            0.986155
```

3.3 Scanning protein sequences

The *predict_sequences* method supports scanning protein sequences for MHC ligands. Here's an example to identify all peptides with a predicted binding affinity of 500 nM or tighter to any allele across two sample genotypes and two short peptide sequences.

```
>>> predictor.predict_sequences(
...     sequences={
...         'protein1': "MDSKGSSQKGSRLLLLLVVSNLL",
...         'protein2': "SSLPTPEDKEQAQQTHH",
...     },
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     result="filtered",
...     comparison_quantity="affinity",
...     filter_value=500,
...     verbose=0)
  sequence_name  pos     peptide         n_flank     c_flank sample_name    affinity␣
↪best_allele  affinity_percentile  processing_score  presentation_score
0      protein1   13   LLLLVVSNL   MDSKGSSQKGSRL           L     sample1   38.206225 ␣
↪     A0201             0.380125          0.017644            0.571060
1      protein1   14   LLLVVSNLL  MDSKGSSQKGSRLL                sample1   42.243472 ␣
↪     A0201             0.420250          0.090984            0.619213
2      protein1    5   SSQKGSRLL           MDSKG   LLLVVSNLL     sample2   66.749223 ␣
↪     C0202             0.803375          0.383608            0.774468
3      protein1    6   SQKGSRLLL          MDSKGS    LLVVSNLL     sample2  178.033467 ␣
↪     C0202             1.820000          0.275019            0.482206
4      protein1   13  LLLLVVSNLL   MDSKGSSQKGSRL                sample1  202.208167 ␣
↪     A0201             1.112500          0.058782            0.261320
5      protein1   12  LLLLLVVSNL    MDSKGSSQKGSR           L     sample1  202.506582 ␣
↪     A0201             1.112500          0.010025            0.225648
6      protein2    0   SSLPTPEDK                    EQAQQTHH     sample1  335.529377 ␣
↪     A0301             1.011750          0.010443            0.156798
7      protein2    0   SSLPTPEDK                    EQAQQTHH     sample2  353.451759 ␣
↪     C0202             2.674250          0.010443            0.150753
8      protein1    8   KGSRLLLLL        MDSKGSSQ      VVSNLL     sample2  410.327286 ␣
↪     C0202             2.887000          0.121374            0.194081
9      protein1    5    SSQKGSRL           MDSKG  LLLLVVSNLL     sample2  477.285937 ␣
↪     C0202             3.107375          0.111982            0.168572
```

When using `predict_sequences`, the flanking sequences for each peptide are automatically included in the processing and presentation predictions.

See the documentation for *Class1PresentationPredictor* for other useful methods.

3.4 Lower level interfaces

The *Class1PresentationPredictor* delegates to a *Class1AffinityPredictor* instance for binding affinity predictions. If all you need are binding affinities, you can use this instance directly.

Here's an example:

```
>>> from mhcflurry import Class1AffinityPredictor
>>> predictor = Class1AffinityPredictor.load()
>>> predictor.predict_to_dataframe(allele="HLA-A0201", peptides=["SIINFEKL", "SIINFEQL
↪"])
    peptide     allele    prediction  prediction_low  prediction_high  prediction_
↪percentile
0  SIINFEKL  HLA-A0201  12906.786173     8829.460289     18029.923061              6.
↪566375
1  SIINFEQL  HLA-A0201  13025.300796     9050.056312     18338.004869              6.
↪623625
```

The `prediction_low` and `prediction_high` fields give the 5-95 percentile predictions across the models in the ensemble. This detailed information is not available through the higher-level *Class1PresentationPredictor* interface.

Under the hood, `Class1AffinityPredictor` itself delegates to an ensemble of of *Class1NeuralNetwork* instances, which implement the neural network models used for prediction. To fit your own affinity prediction models, call *fit*.

You can similarly use *Class1ProcessingPredictor* directly for antigen processing prediction, and there is a low-level *Class1ProcessingNeuralNetwork* with a *fit* method.

See the API documentation of these classes for details.

CHAPTER

FOUR

COMMAND-LINE REFERENCE

See also the *tutorial*.

4.1 mhcflurry-predict

Run MHCflurry predictor on specified peptides.

By default, the presentation predictor is used, and predictions for MHC I binding affinity, antigen processing, and the composite presentation score are returned. If you just want binding affinity predictions, pass –affinity-only.

Examples:

Write a CSV file containing the contents of INPUT.csv plus additional columns giving MHCflurry predictions:

$ mhcflurry-predict INPUT.csv –out RESULT.csv

The input CSV file is expected to contain columns “allele”, “peptide”, and, optionally, “n_flank”, and “c_flank”.

If `--out` is not specified, results are written to stdout.

You can also run on alleles and peptides specified on the commandline, in which case predictions are written for *all combinations* of alleles and peptides:

$ mhcflurry-predict –alleles HLA-A0201 H-2Kb –peptides SIINFEKL DENDREKLLL

Instead of individual alleles (in a CSV or on the command line), you can also give a comma separated list of alleles giving a sample genotype. In this case, the tightest binding affinity across the alleles for the sample will be returned. For example:

$ mhcflurry-predict –peptides SIINFEKL DENDREKLLL –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:01,HLA-C*07:02 HLA-A*01:01,HLA-A*02:06,HLA-B*44:02,HLA-B*07:02,HLA-C*01:01,HLA-C*03:01

will give the tightest predicted affinities across alleles for each of the two genotypes specified for each peptide.

```
usage: mhcflurry-predict [-h] [--list-supported-alleles]
                         [--list-supported-peptide-lengths] [--version]
                         [--alleles ALLELE [ALLELE ...]]
                         [--peptides PEPTIDE [PEPTIDE ...]]
                         [--allele-column NAME] [--peptide-column NAME]
                         [--n-flank-column NAME] [--c-flank-column NAME]
                         [--no-throw] [--out OUTPUT.csv]
                         [--prediction-column-prefix NAME]
                         [--output-delimiter CHAR] [--no-affinity-percentile]
                         [--always-include-best-allele] [--models DIR]
                         [--affinity-only] [--no-flanking]
                         [INPUT.csv]
```

input.csv
Input CSV

-h, --help
Show this help message and exit

--list-supported-alleles
Prints the list of supported alleles and exits

--list-supported-peptide-lengths
Prints the list of supported peptide lengths and exits

--version
show program's version number and exit

--alleles <allele>
Alleles to predict (exclusive with passing an input CSV)

--peptides <peptide>
Peptides to predict (exclusive with passing an input CSV)

--allele-column <name>
Input column name for alleles. Default: 'allele'

--peptide-column <name>
Input column name for peptides. Default: 'peptide'

--n-flank-column <name>
Column giving N-terminal flanking sequence. Default: 'n_flank'

--c-flank-column <name>
Column giving C-terminal flanking sequence. Default: 'c_flank'

--no-throw
Return NaNs for unsupported alleles or peptides instead of raising

--out <output.csv>
Output CSV

--prediction-column-prefix <name>
Prefix for output column names. Default: '**mhcflurry_**'

--output-delimiter <char>
Delimiter character for results. Default: ','

--no-affinity-percentile
Do not include affinity percentile rank

--always-include-best-allele
Always include the best_allele column even when it is identical to the allele column (i.e. all queries are monoallelic).

--models <dir>
Directory containing models. Either a binding affinity predictor or a presentation predictor can be used. Default: /Users/tim/Library/Application Support/mhcflurry/4/1.6.0/models_class1_presentation/models

--affinity-only
Affinity prediction only (no antigen processing or presentation)

--no-flanking
Do not use flanking sequence information even when available

4.2 mhcflurry-predict-scan

Scan protein sequences using the MHCflurry presentation predictor.

By default, sub-sequences (peptides) with affinity percentile ranks less than 2.0 are returned. You can also specify –results-all to return predictions for all peptides, or –results-best to return the top peptide for each sequence.

Examples:

Scan a set of sequences in a FASTA file for binders to any alleles in a MHC I genotype:

$ mhcflurry-predict-scan test/data/example.fasta –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:01,HLA-C*07:02

Instead of a FASTA, you can also pass a CSV that has "sequence_id" and "sequence" columns.

You can also specify multiple MHC I genotypes to scan as space-separated arguments to the –alleles option:

$ mhcflurry-predict-scan test/data/example.fasta –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:02,HLA-C*07:02 HLA-A*01:01,HLA-A*02:06,HLA-B*44:02,HLA-B*07:02,HLA-C*01:02,HLA-C*03:01

If `--out` is not specified, results are written to standard out.

You can also specify sequences on the commandline:

mhcflurry-predict-scan –sequences MGYINVFAFPFTIYSLLLCRMNSRNYIAQVDVVNFNLT –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:02,HLA-C*07:02

```
usage: mhcflurry-predict-scan [-h] [--list-supported-alleles]
                              [--list-supported-peptide-lengths] [--version]
                              [--input-format {guess,csv,fasta}]
                              [--alleles ALLELE [ALLELE ...]]
                              [--sequences SEQ [SEQ ...]]
                              [--sequence-id-column NAME]
                              [--sequence-column NAME] [--no-throw]
                              [--peptide-lengths PEPTIDE_LENGTHS [PEPTIDE_LENGTHS ...
↪]]
                              [--results-all]
                              [--results-best {presentation_score,processing_score,
↪affinity,affinity_percentile}]
                              [--results-filtered {presentation_score,processing_
↪score,affinity,affinity_percentile}]
                              [--threshold-presentation-score THRESHOLD_PRESENTATION_
↪SCORE]
                              [--threshold-processing-score THRESHOLD_PROCESSING_
↪SCORE]
                              [--threshold-affinity THRESHOLD_AFFINITY]
                              [--threshold-affinity-percentile THRESHOLD_AFFINITY_
↪PERCENTILE]
                              [--out OUTPUT.csv] [--output-delimiter CHAR]
                              [--no-affinity-percentile] [--models DIR]
                              [--no-flanking]
                              [INPUT]
```

input
: Input CSV or FASTA

-h, --help
: Show this help message and exit

--list-supported-alleles
Print the list of supported alleles and exits

--list-supported-peptide-lengths
Print the list of supported peptide lengths and exits

--version
show program's version number and exit

--input-format {guess,csv,fasta}
Format of input file. By default, it is guessed from the file extension.

--alleles <allele>
Alleles to predict

--sequences <seq>
Sequences to predict (exclusive with passing an input file)

--sequence-id-column <name>
Input CSV column name for sequence IDs. Default: 'sequence_id'

--sequence-column <name>
Input CSV column name for sequences. Default: 'sequence'

--no-throw
Return NaNs for unsupported alleles or peptides instead of raising

--peptide-lengths <peptide_lengths>
Peptide lengths to consider. Default: [8, 9, 10, 11].

--results-all
Return results for all peptides regardless of affinity, etc.

--results-best {presentation_score,processing_score,affinity,affinity_percentile}
Take the top result for each sequence according to the specified predicted quantity

--results-filtered {presentation_score,processing_score,affinity,affinity_percentile}
Filter results by the specified quantity.

--threshold-presentation-score <threshold_presentation_score>
Threshold if filtering by presentation score. Default: 0.7

--threshold-processing-score <threshold_processing_score>
Threshold if filtering by processing score. Default: 0.5

--threshold-affinity <threshold_affinity>
Threshold if filtering by affinity. Default: 500

--threshold-affinity-percentile <threshold_affinity_percentile>
Threshold if filtering by affinity percentile. Default: 2.0

--out <output.csv>
Output CSV

--output-delimiter <char>
Delimiter character for results. Default: ','

--no-affinity-percentile
Do not include affinity percentile rank

--models <dir>
Directory containing presentation models.Default: /Users/tim/Library/Application Support/mhcflurry/4/1.6.0/models_class1_presentation/models

--no-flanking
Do not use flanking sequence information in predictions

4.3 mhcflurry-downloads

Download MHCflurry released datasets and trained models.

Examples

Fetch the default downloads: $ mhcflurry-downloads fetch

Fetch a specific download: $ mhcflurry-downloads fetch models_class1_pan

Get the path to a download: $ mhcflurry-downloads path models_class1_pan

Get the URL of a download: $ mhcflurry-downloads url models_class1_pan

Summarize available and fetched downloads: $ mhcflurry-downloads info

```
usage: mhcflurry-downloads [-h] [--quiet] [--verbose]
                           {fetch,info,path,url} ...
```

-h, --help
show this help message and exit

--quiet
Output less

--verbose, -v
Output more

4.3.1 mhcflurry-downloads fetch

```
usage: mhcflurry-downloads fetch [-h] [--keep] [--release RELEASE]
                                 [--already-downloaded-dir DIR]
                                 [DOWNLOAD [DOWNLOAD ...]]
```

download
Items to download

-h, --help
show this help message and exit

--keep
Don't delete archives after they are extracted

--release <release>
Release to download. Default: 1.6.0

--already-downloaded-dir <dir>
Don't download files, get them from DIR

4.3.2 mhcflurry-downloads info

```
usage: mhcflurry-downloads info [-h]
```

-h, --help
show this help message and exit

4.3.3 mhcflurry-downloads path

```
usage: mhcflurry-downloads path [-h] [download_name]
```

download_name

-h, --help
show this help message and exit

4.3.4 mhcflurry-downloads url

```
usage: mhcflurry-downloads url [-h] [download_name]
```

download_name

-h, --help
show this help message and exit

4.4 mhcflurry-class1-train-allele-specific-models

```
usage:
Train Class1 single allele models.
```

-h, --help
show this help message and exit

--data <file.csv>
Training data CSV. Expected columns: allele, peptide, measurement_value

--out-models-dir <dir>
Directory to write models and manifest

--hyperparameters <file.json>
JSON or YAML of hyperparameters

--allele <allele>
Alleles to train models for. If not specified, all alleles with enough measurements will be used.

--min-measurements-per-allele <n>
Train models for alleles with >=N measurements.

--held-out-fraction-reciprocal <n>
Hold out 1/N fraction of data (for e.g. subsequent model selection. For example, specify 5 to hold out 20 percent of the data.

--held-out-fraction-seed <n>
Seed for randomizing which measurements are held out. Only matters when –held-out-fraction is specified. Default: 0.

--ignore-inequalities
Do not use affinity value inequalities even when present in data

--n-models <n>
Ensemble size, i.e. how many models to train for each architecture. If specified here it overrides any 'n_models' specified in the hyperparameters.

--max-epochs <n>
Max training epochs. If specified here it overrides any 'max_epochs' specified in the hyperparameters.

--allele-sequences <file.csv>
Allele sequences file. Used for computing allele similarity matrix.

--save-interval <n>
Write models to disk every N seconds. Only affects parallel runs; serial runs write each model to disk as it is trained.

--verbosity <verbosity>
Keras verbosity. Default: 0

--num-jobs <n>
Number of local processes to parallelize training over. Set to 0 for serial run. Default: 0.

--backend {tensorflow-gpu,tensorflow-cpu,tensorflow-default}
Keras backend. If not specified will use system default.

--gpus <n>
Number of GPUs to attempt to parallelize across. Requires running in parallel.

--max-workers-per-gpu <n>
Maximum number of workers to assign to a GPU. Additional tasks will run on CPU.

--max-tasks-per-worker <n>
Restart workers after N tasks. Workaround for tensorflow memory leaks. Requires Python >=3.2.

--worker-log-dir <worker_log_dir>
Write worker stdout and stderr logs to given directory.

4.5 mhcflurry-class1-select-allele-specific-models

```
usage:
Model select class1 single allele models.
```

-h, --help
show this help message and exit

--data <file.csv>
Model selection data CSV. Expected columns: allele, peptide, measurement_value

--exclude-data <file.csv>
Data to EXCLUDE from model selection. Useful to specify the original training data used

--models-dir <dir>
Directory to read models

--out-models-dir <dir>
Directory to write selected models

--out-unselected-predictions <file.csv>
Write predictions for validation data using unselected predictor to FILE.csv

--unselected-accuracy-scorer <scorer>

--unselected-accuracy-scorer-num-samples <unselected_accuracy_scorer_num_samples>

--unselected-accuracy-percentile-threshold <x>

--allele <allele>
Alleles to select models for. If not specified, all alleles with enough measurements will be used.

--combined-min-models <n>
Min number of models to select per allele when using combined selector

--combined-max-models <n>
Max number of models to select per allele when using combined selector

--combined-min-contribution-percent <x>
Use only model selectors that can contribute at least X % to the total score. Default: 1.0

--mass-spec-min-measurements <n>
Min number of measurements required for an allele to use mass-spec model selection

--mass-spec-min-models <n>
Min number of models to select per allele when using mass-spec selector

--mass-spec-max-models <n>
Max number of models to select per allele when using mass-spec selector

--mse-min-measurements <n>
Min number of measurements required for an allele to use MSE model selection

--mse-min-models <n>
Min number of models to select per allele when using MSE selector

--mse-max-models <n>
Max number of models to select per allele when using MSE selector

--scoring <scoring>
Scoring procedures to use in order

--consensus-min-models <n>
Min number of models to select per allele when using consensus selector

--consensus-max-models <n>
Max number of models to select per allele when using consensus selector

--consensus-num-peptides-per-length <consensus_num_peptides_per_length>
Num peptides per length to use for consensus scoring

--mass-spec-regex <regex>
Regular expression for mass-spec data. Runs on measurement_source col.Default: mass[-]spec.

--verbosity <verbosity>
Keras verbosity. Default: 0

--num-jobs <n>
Number of local processes to parallelize training over. Set to 0 for serial run. Default: 0.

--backend {tensorflow-gpu,tensorflow-cpu,tensorflow-default}
Keras backend. If not specified will use system default.

--gpus <n>
Number of GPUs to attempt to parallelize across. Requires running in parallel.

--max-workers-per-gpu <n>
Maximum number of workers to assign to a GPU. Additional tasks will run on CPU.

--max-tasks-per-worker <n>
Restart workers after N tasks. Workaround for tensorflow memory leaks. Requires Python >=3.2.

--worker-log-dir <worker_log_dir>
Write worker stdout and stderr logs to given directory.

4.6 mhcflurry-class1-train-pan-allele-models

```
usage:
Train Class1 pan-allele models.
```

-h, --help
show this help message and exit

--data <file.csv>
Training data CSV. Expected columns: allele, peptide, measurement_value

--pretrain-data <file.csv>
Pre-training data CSV. Expected columns: allele, peptide, measurement_value

--out-models-dir <dir>
Directory to write models and manifest

--hyperparameters <file.json>
JSON or YAML of hyperparameters

--held-out-measurements-per-allele-fraction-and-max <x>
Fraction of measurements per allele to hold out, and maximum number

--ignore-inequalities
Do not use affinity value inequalities even when present in data

--num-folds <n>
Number of training folds.

--num-replicates <n>
Number of replicates per (architecture, fold) pair to train.

--max-epochs <n>
Max training epochs. If specified here it overrides any 'max_epochs' specified in the hyperparameters.

--allele-sequences <file.csv>
Allele sequences file.

--verbosity <verbosity>
Keras verbosity. Default: 0

--debug
Launch python debugger on error

--continue-incomplete
Continue training models from an incomplete training run. If this is specified then the only required argument is –out-models-dir

--only-initialize
Do not actually train models. The initialized run can be continued later with –continue-incomplete.

--num-jobs <n>
Number of local processes to parallelize training over. Set to 0 for serial run. Default: 0.

--backend {tensorflow-gpu,tensorflow-cpu,tensorflow-default}
Keras backend. If not specified will use system default.

--gpus <n>
Number of GPUs to attempt to parallelize across. Requires running in parallel.

--max-workers-per-gpu <n>
Maximum number of workers to assign to a GPU. Additional tasks will run on CPU.

--max-tasks-per-worker <n>
Restart workers after N tasks. Workaround for tensorflow memory leaks. Requires Python >=3.2.

--worker-log-dir <worker_log_dir>
Write worker stdout and stderr logs to given directory.

--cluster-parallelism

--cluster-submit-command <cluster_submit_command>
Default: sh

--cluster-results-workdir <cluster_results_workdir>
Default: ./cluster-workdir

--additional-complete-file <additional_complete_file>
Additional file to monitor for job completion. Default: STDERR

--cluster-script-prefix-path <cluster_script_prefix_path>

--cluster-max-retries <cluster_max_retries>
How many times to rerun failing jobs. Default: 3

4.7 mhcflurry-class1-select-pan-allele-models

```
usage:
Model select class1 pan-allele models.

APPROACH: For each training fold, we select at least min and at most max models
(where min and max are set by the --{min/max}-models-per-fold argument) using a
step-up (forward) selection procedure. The final ensemble is the union of all
selected models across all folds.
```

-h, --help
show this help message and exit

--data <file.csv>
Model selection data CSV. Expected columns: allele, peptide, measurement_value

--models-dir <dir>
Directory to read models

--out-models-dir <dir>
Directory to write selected models

--min-models-per-fold <n>
Min number of models to select per fold

--max-models-per-fold <n>
Max number of models to select per fold

--mass-spec-regex <regex>
Regular expression for mass-spec data. Runs on measurement_source col.Default: mass[-]spec.

--verbosity <verbosity>
Keras verbosity. Default: 0

--num-jobs <n>
Number of local processes to parallelize training over. Set to 0 for serial run. Default: 0.

--backend {tensorflow-gpu,tensorflow-cpu,tensorflow-default}
Keras backend. If not specified will use system default.

--gpus <n>
Number of GPUs to attempt to parallelize across. Requires running in parallel.

--max-workers-per-gpu <n>
Maximum number of workers to assign to a GPU. Additional tasks will run on CPU.

--max-tasks-per-worker <n>
Restart workers after N tasks. Workaround for tensorflow memory leaks. Requires Python >=3.2.

--worker-log-dir <worker_log_dir>
Write worker stdout and stderr logs to given directory.

--cluster-parallelism

--cluster-submit-command <cluster_submit_command>
Default: sh

--cluster-results-workdir <cluster_results_workdir>
Default: ./cluster-workdir

--additional-complete-file <additional_complete_file>
Additional file to monitor for job completion. Default: STDERR

--cluster-script-prefix-path <cluster_script_prefix_path>

--cluster-max-retries <cluster_max_retries>
How many times to rerun failing jobs. Default: 3

4.8 mhcflurry-class1-train-processing-models

```
usage:
Train Class1 processing models.
```

-h, --help
show this help message and exit

--data <file.csv>
Training data CSV. Expected columns: peptide, n_flank, c_flank, hit

--out-models-dir <dir>
Directory to write models and manifest

--hyperparameters <file.json>
JSON or YAML of hyperparameters

--held-out-samples <n>
Number of experiments to hold out per fold

--num-folds <n>
Number of training folds.

--num-replicates <n>
Number of replicates per (architecture, fold) pair to train.

--max-epochs <n>
Max training epochs. If specified here it overrides any 'max_epochs' specified in the hyperparameters.

--verbosity <verbosity>
Keras verbosity. Default: 0

--debug
Launch python debugger on error

--continue-incomplete
Continue training models from an incomplete training run. If this is specified then the only required argument is –out-models-dir

--only-initialize
Do not actually train models. The initialized run can be continued later with –continue-incomplete.

--num-jobs <n>
Number of local processes to parallelize training over. Set to 0 for serial run. Default: 0.

--backend {tensorflow-gpu,tensorflow-cpu,tensorflow-default}
Keras backend. If not specified will use system default.

--gpus <n>
Number of GPUs to attempt to parallelize across. Requires running in parallel.

--max-workers-per-gpu <n>
Maximum number of workers to assign to a GPU. Additional tasks will run on CPU.

--max-tasks-per-worker <n>
Restart workers after N tasks. Workaround for tensorflow memory leaks. Requires Python >=3.2.

--worker-log-dir <worker_log_dir>
Write worker stdout and stderr logs to given directory.

--cluster-parallelism

--cluster-submit-command <cluster_submit_command>
Default: sh

--cluster-results-workdir <cluster_results_workdir>
Default: ./cluster-workdir

--additional-complete-file <additional_complete_file>
Additional file to monitor for job completion. Default: STDERR

--cluster-script-prefix-path <cluster_script_prefix_path>

--cluster-max-retries <cluster_max_retries>
How many times to rerun failing jobs. Default: 3

4.9 mhcflurry-class1-select-processing-models

```
usage:
Model select antigen processing models.

APPROACH: For each training fold, we select at least min and at most max models
(where min and max are set by the --{min/max}-models-per-fold argument) using a
step-up (forward) selection procedure. The final ensemble is the union of all
selected models across all folds. AUC is used as the metric.
```

-h, --help
show this help message and exit

--data <file.csv>
Model selection data CSV. Expected columns: peptide, hit, fold_0, . . . , fold_N

--models-dir <dir>
Directory to read models

--out-models-dir <dir>
Directory to write selected models

--min-models-per-fold <n>
Min number of models to select per fold

--max-models-per-fold <n>
Max number of models to select per fold

--verbosity <verbosity>
Keras verbosity. Default: 0

--num-jobs <n>
Number of local processes to parallelize training over. Set to 0 for serial run. Default: 0.

--backend {tensorflow-gpu,tensorflow-cpu,tensorflow-default}
Keras backend. If not specified will use system default.

--gpus <n>
Number of GPUs to attempt to parallelize across. Requires running in parallel.

--max-workers-per-gpu <n>
Maximum number of workers to assign to a GPU. Additional tasks will run on CPU.

--max-tasks-per-worker <n>
Restart workers after N tasks. Workaround for tensorflow memory leaks. Requires Python >=3.2.

--worker-log-dir <worker_log_dir>
Write worker stdout and stderr logs to given directory.

--cluster-parallelism

--cluster-submit-command <cluster_submit_command>
Default: sh

--cluster-results-workdir <cluster_results_workdir>
Default: ./cluster-workdir

--additional-complete-file <additional_complete_file>
Additional file to monitor for job completion. Default: STDERR

--cluster-script-prefix-path <cluster_script_prefix_path>

--cluster-max-retries <cluster_max_retries>
How many times to rerun failing jobs. Default: 3

4.10 mhcflurry-class1-train-presentation-models

```
usage:
Train Class1 presentation models.
```

-h, --help
show this help message and exit

--data <file.csv>
Training data CSV. Expected columns: peptide, n_flank, c_flank, hit

--out-models-dir <dir>
Directory to write models and manifest

--affinity-predictor <dir>
Affinity predictor models dir

--processing-predictor-with-flanks <dir>
Processing predictor with flanks

--processing-predictor-without-flanks <dir>
Processing predictor without flanks

--verbosity <verbosity>
Default: 1

--debug
Launch python debugger on error

--hla-column <hla_column>
Column in data giving space-separated MHC I alleles

--target-column <target_column>
Column in data giving hit (1) vs decoy (0)

CHAPTER

FIVE

API DOCUMENTATION

Class I MHC ligand prediction package

class mhcflurry.**Class1AffinityPredictor**(*allele_to_allele_specific_models=None*, *class1_pan_allele_models=None*, *allele_to_sequence=None*, *manifest_df=None*, *allele_to_percent_rank_transform=None*, *metadata_dataframes=None*)

Bases: object

High-level interface for peptide/MHC I binding affinity prediction.

This class manages low-level *Class1NeuralNetwork* instances, each of which wraps a single Keras network. The purpose of *Class1AffinityPredictor* is to implement ensembles, handling of multiple alleles, and predictor loading and saving. It also provides a place to keep track of metadata like prediction histograms for percentile rank calibration.

Parameters

allele_to_allele_specific_models [dict of string -> list of *Class1NeuralNetwork*] Ensemble of single-allele models to use for each allele.

class1_pan_allele_models [list of *Class1NeuralNetwork*] Ensemble of pan-allele models.

allele_to_sequence [dict of string -> string] MHC allele name to fixed-length amino acid sequence (sometimes referred to as the pseudosequence). Required only if class1_pan_allele_models is specified.

manifest_df [pandas.DataFrame, optional] Must have columns: model_name, allele, config_json, model. Only required if you want to update an existing serialization of a Class1AffinityPredictor. Otherwise this dataframe will be generated automatically based on the supplied models.

allele_to_percent_rank_transform [dict of string -> PercentRankTransform, optional] PercentRankTransform instances to use for each allele

metadata_dataframes [dict of string -> pandas.DataFrame, optional] Optional additional dataframes to write to the models dir when save() is called. Useful for tracking provenance.

property manifest_df

A pandas.DataFrame describing the models included in this predictor.

Based on: - self.class1_pan_allele_models - self.allele_to_allele_specific_models

Returns

pandas.DataFrame

clear_cache (*self*)
Clear values cached based on the neural networks in this predictor.

Users should call this after mutating any of the following:

- self.class1_pan_allele_models
- self.allele_to_allele_specific_models
- self.allele_to_sequence

Methods that mutate these instance variables will call this method on their own if needed.

property neural_networks
List of the neural networks in the ensemble.

Returns

list of *Class1NeuralNetwork*

classmethod merge (*predictors*)
Merge the ensembles of two or more *Class1AffinityPredictor* instances.

Note: the resulting merged predictor will NOT have calibrated percentile ranks. Call *calibrate_percentile_ranks* on it if these are needed.

Parameters

predictors [sequence of *Class1AffinityPredictor*]

Returns

***Class1AffinityPredictor* instance**

merge_in_place (*self*, *others*)
Add the models present in other predictors into the current predictor.

Parameters

others [list of Class1AffinityPredictor] Other predictors to merge into the current predictor.

Returns

list of string [names of newly added models]

property supported_alleles
Alleles for which predictions can be made.

Returns

list of string

property supported_peptide_lengths
(minimum, maximum) lengths of peptides supported by *all models*, inclusive.

Returns

(int, int) tuple

check_consistency (*self*)
Verify that self.manifest_df is consistent with: - self.class1_pan_allele_models - self.allele_to_allele_specific_models

Currently only checks for agreement on the total number of models.

Throws AssertionError if inconsistent.

save(*self*, *models_dir*, *model_names_to_write=None*, *write_metadata=True*)
Serialize the predictor to a directory on disk. If the directory does not exist it will be created.

The serialization format consists of a file called "manifest.csv" with the configurations of each Class1NeuralNetwork, along with per-network files giving the model weights. If there are pan-allele predictors in the ensemble, the allele sequences are also stored in the directory. There is also a small file "index.txt" with basic metadata: when the models were trained, by whom, on what host.

Parameters

models_dir [string] Path to directory. It will be created if it doesn't exist.

model_names_to_write [list of string, optional] Only write the weights for the specified models. Useful for incremental updates during training.

write_metadata [boolean, optional] Whether to write optional metadata

static load(*models_dir=None*, *max_models=None*, *optimization_level=None*)
Deserialize a predictor from a directory on disk.

Parameters

models_dir [string] Path to directory. If unspecified the default downloaded models are used.

max_models [int, optional] Maximum number of *Class1NeuralNetwork* instances to load

optimization_level [int] If >0, model optimization will be attempted. Defaults to value of environment variable MHCFLURRY_OPTIMIZATION_LEVEL.

Returns

Class1AffinityPredictor **instance**

optimize(*self*, *warn=True*)
EXPERIMENTAL: Optimize the predictor for faster predictions.

Currently the only optimization implemented is to merge multiple pan- allele predictors at the tensorflow level.

The optimization is performed in-place, mutating the instance.

Returns

bool Whether optimization was performed

static model_name(*allele*, *num*)
Generate a model name

Parameters

allele [string]

num [int]

Returns

string

static weights_path(*models_dir*, *model_name*)
Generate the path to the weights file for a model

Parameters

models_dir [string]

model_name [string]

Returns

string

property master_allele_encoding

An AlleleEncoding containing the universe of alleles specified by self.allele_to_sequence.

Returns

AlleleEncoding

fit_allele_specific_predictors(*self*, *n_models*, *architecture_hyperparameters_list*, *allele*, *peptides*, *affinities*, *inequalities=None*, *train_rounds=None*, *models_dir_for_save=None*, *verbose=0*, *progress_preamble=''*, *progress_print_interval=5.0*)

Fit one or more allele specific predictors for a single allele using one or more neural network architectures.

The new predictors are saved in the Class1AffinityPredictor instance and will be used on subsequent calls to *predict*.

Parameters

n_models [int] Number of neural networks to fit

architecture_hyperparameters_list [list of dict] List of hyperparameter sets.

allele [string]

peptides [`EncodableSequences` or list of string]

affinities [list of float] nM affinities

inequalities [list of string, each element one of ">", "<", or "="] See *Class1NeuralNetwork.fit* for details.

train_rounds [sequence of int] Each training point i will be used on training rounds r for which train_rounds[i] > r, r >= 0.

models_dir_for_save [string, optional] If specified, the Class1AffinityPredictor is (incrementally) written to the given models dir after each neural network is fit.

verbose [int] Keras verbosity

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress. Set to None to disable.

Returns

list of *Class1NeuralNetwork*

fit_class1_pan_allele_models(*self*, *n_models*, *architecture_hyperparameters*, *alleles*, *peptides*, *affinities*, *inequalities*, *models_dir_for_save=None*, *verbose=1*, *progress_preamble=''*, *progress_print_interval=5.0*)

Fit one or more pan-allele predictors using a single neural network architecture.

The new predictors are saved in the Class1AffinityPredictor instance and will be used on subsequent calls to *predict*.

Parameters

n_models [int] Number of neural networks to fit

architecture_hyperparameters [dict]

alleles [list of string] Allele names (not sequences) corresponding to each peptide

peptides [`EncodableSequences` or list of string]

affinities [list of float] nM affinities

inequalities [list of string, each element one of ">", "<", or "="] See Class1NeuralNetwork.fit for details.

models_dir_for_save [string, optional] If specified, the Class1AffinityPredictor is (incrementally) written to the given models dir after each neural network is fit.

verbose [int] Keras verbosity

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress. Set to None to disable.

Returns

list of *`Class1NeuralNetwork`*

`add_pan_allele_model`(*self*, *model*, *models_dir_for_save=None*)

Add a pan-allele model to the ensemble and optionally do an incremental save.

Parameters

model [Class1NeuralNetwork]

models_dir_for_save [string] Directory to save resulting ensemble to

`percentile_ranks`(*self*, *affinities*, *allele=None*, *alleles=None*, *throw=True*)

Return percentile ranks for the given ic50 affinities and alleles.

The 'allele' and 'alleles' argument are as in the *`predict`* method. Specify one of these.

Parameters

affinities [sequence of float] nM affinities

allele [string]

alleles [sequence of string]

throw [boolean] If True, a ValueError will be raised in the case of unsupported alleles. If False, a warning will be logged and NaN will be returned for those percentile ranks.

Returns

numpy.array of float

`predict`(*self*, *peptides*, *alleles=None*, *allele=None*, *throw=True*, *centrality_measure='mean'*, *model_kwargs={}*)

Predict nM binding affinities.

If multiple predictors are available for an allele, the predictions are the geometric means of the individual model (nM) predictions.

One of 'allele' or 'alleles' must be specified. If 'allele' is specified all predictions will be for the given allele. If 'alleles' is specified it must be the same length as 'peptides' and give the allele corresponding to each peptide.

Parameters

peptides [`EncodableSequences` or list of string]

alleles [list of string]

allele [string]

throw [boolean] If True, a ValueError will be raised in the case of unsupported alleles or peptide lengths. If False, a warning will be logged and the predictions for the unsupported alleles or peptides will be NaN.

centrality_measure [string or callable] Measure of central tendency to use to combine predictions in the ensemble. Options include: mean, median, robust_mean.

model_kwargs [dict] Additional keyword arguments to pass to Class1NeuralNetwork.predict

Returns

numpy.array of predictions

predict_to_dataframe(*self*, *peptides*, *alleles=None*, *allele=None*, *throw=True*, *include_individual_model_predictions=False*, *include_percentile_ranks=True*, *include_confidence_intervals=True*, *centrality_measure='mean'*, *model_kwargs={}*)

Predict nM binding affinities. Gives more detailed output than `predict` method, including 5-95% prediction intervals.

If multiple predictors are available for an allele, the predictions are the geometric means of the individual model predictions.

One of 'allele' or 'alleles' must be specified. If 'allele' is specified all predictions will be for the given allele. If 'alleles' is specified it must be the same length as 'peptides' and give the allele corresponding to each peptide.

Parameters

peptides [`EncodableSequences` or list of string]

alleles [list of string]

allele [string]

throw [boolean] If True, a ValueError will be raised in the case of unsupported alleles or peptide lengths. If False, a warning will be logged and the predictions for the unsupported alleles or peptides will be NaN.

include_individual_model_predictions [boolean] If True, the predictions of each individual model are included as columns in the result DataFrame.

include_percentile_ranks [boolean, default True] If True, a "prediction_percentile" column will be included giving the percentile ranks. If no percentile rank info is available, this will be ignored with a warning.

centrality_measure [string or callable] Measure of central tendency to use to combine predictions in the ensemble. Options include: mean, median, robust_mean.

model_kwargs [dict] Additional keyword arguments to pass to Class1NeuralNetwork.predict

Returns

`pandas.DataFrame` of predictions

calibrate_percentile_ranks(*self, peptides=None, num_peptides_per_length=100000, alleles=None, bins=None, motif_summary=False, summary_top_peptide_fractions=[0.001], verbose=False, model_kwargs={}*)

Compute the cumulative distribution of ic50 values for a set of alleles over a large universe of random peptides, to enable taking quantiles of this distribution later.

Parameters

peptides [sequence of string or EncodableSequences, optional] Peptides to use

num_peptides_per_length [int, optional] If peptides argument is not specified, then num_peptides_per_length peptides are randomly sampled from a uniform distribution for each supported length

alleles [sequence of string, optional] Alleles to perform calibration for. If not specified all supported alleles will be calibrated.

bins [object] Anything that can be passed to numpy.histogram's "bins" argument can be used here, i.e. either an integer or a sequence giving bin edges. This is in ic50 space.

motif_summary [bool] If True, the length distribution and per-position amino acid frequencies are also calculated for the top x fraction of tightest- binding peptides, where each value of x is given in the summary_top_peptide_fractions list.

summary_top_peptide_fractions [list of float] Only used if motif_summary is True

verbose [boolean] Whether to print status updates to stdout

model_kwargs [dict] Additional low-level Class1NeuralNetwork.predict() kwargs.

Returns

dict of string -> pandas.DataFrame

If motif_summary is True, this will have keys "frequency_matrices" and

"length_distributions". Otherwise it will be empty.

model_select(*self*, *score_function*, *alleles=None*, *min_models=1*, *max_models=10000*)

Perform model selection using a user-specified scoring function.

This works only with allele-specific models, not pan-allele models.

Model selection is done using a "step up" variable selection procedure, in which models are repeatedly added to an ensemble until the score stops improving.

Parameters

score_function [Class1AffinityPredictor -> float function] Scoring function

alleles [list of string, optional] If not specified, model selection is performed for all alleles.

min_models [int, optional] Min models to select per allele

max_models [int, optional] Max models to select per allele

Returns

Class1AffinityPredictor [predictor containing the selected models]

class mhcflurry.**Class1NeuralNetwork**(***hyperparameters*)

Bases: object

Low level class I predictor consisting of a single neural network.

Both single allele and pan-allele prediction are supported.

Users will generally use Class1AffinityPredictor, which gives a higher-level interface and supports ensembles.

network_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob
Hyperparameters (and their default values) that affect the neural network architecture.

compile_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob
Loss and optimizer hyperparameters.

fit_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object
Hyperparameters for neural network training.

early_stopping_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefa
Hyperparameters for early stopping.

miscelaneous_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaul
Miscelaneous hyperaparameters. These parameters are not used by this class but may be interpreted by other code.

hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object>
Combined set of all supported hyperparameters and their default values.

hyperparameter_renames = {'embedding_init_method': None, 'embedding_input_dim': None

classmethod apply_hyperparameter_renames(*hyperparameters*)
Handle hyperparameter renames.

Parameters

hyperparameters [dict]

Returns

dict [updated hyperparameters]

KERAS_MODELS_CACHE = {}
Process-wide keras model cache, a map from: architecture JSON string to (Keras model, existing network weights)

classmethod clear_model_cache()
Clear the Keras model cache.

classmethod borrow_cached_network(*network_json*, *network_weights*)
Return a keras Model with the specified architecture and weights. As an optimization, when possible this will reuse architectures from a process-wide cache.

The returned object is "borrowed" in the sense that its weights can change later after subsequent calls to this method from other objects.

If you're using this from a parallel implementation you'll need to hold a lock while using the returned object.

Parameters

network_json [string of JSON]

network_weights [list of numpy.array]

Returns

keras.models.Model

network(*self*, *borrow=False*)
Return the keras model associated with this predictor.

Parameters

borrow [bool] Whether to return a cached model if possible. See borrow_cached_network for details

Returns

keras.models.Model

update_network_description (*self*)
Update self.network_json and self.network_weights properties based on this instances's neural network.

static keras_network_cache_key (*network_json*)
Given a Keras JSON description of a neural network, return a key that uniquely defines this network. Networks that share the same key should have compatible weights matrices and give the same prediction outputs when their weights are the same.

Parameters

network_json [string]

Returns

string

get_config (*self*)
serialize to a dict all attributes except model weights

Returns

dict

classmethod from_config (*config*, *weights=None*, *weights_loader=None*)
deserialize from a dict returned by get_config().

Parameters

config [dict]

weights [list of array, optional] Network weights to restore

weights_loader [callable, optional] Function to call (no arguments) to load weights when needed

Returns

Class1NeuralNetwork

load_weights (*self*)
Load weights by evaluating self.network_weights_loader, if needed.

After calling this, self.network_weights_loader will be None and self.network_weights will be the weights list, if available.

get_weights (*self*)
Get the network weights

Returns

list of numpy.array giving weights for each layer or None if there is no

network

peptides_to_network_input (*self*, *peptides*)
Encode peptides to the fixed-length encoding expected by the neural network (which depends on the architecture).

Parameters

peptides [EncodableSequences or list of string]

Returns

numpy.array

property supported_peptide_lengths
(minimum, maximum) lengths of peptides supported, inclusive.

Returns

(int, int) tuple

allele_encoding_to_network_input (*self*, *allele_encoding*)
Encode alleles to the fixed-length encoding expected by the neural network (which depends on the architecture).

Parameters

allele_encoding [AlleleEncoding]

Returns

(numpy.array, numpy.array)

Indices and allele representations.

static data_dependent_weights_initialization (*network*, *x_dict=None*, *method='lsuv'*, *verbose=1*)
Data dependent weights initialization.

Parameters

network [keras.Model]

x_dict [dict of string -> numpy.ndarray] Training data as would be passed keras.Model.fit().

method [string] Initialization method. Currently only "lsuv" is supported.

verbose [int] Status updates printed to stdout if verbose > 0

fit_generator (*self*, *generator*, *validation_peptide_encoding*, *validation_affinities*, *validation_allele_encoding=None*, *validation_inequalities=None*, *validation_output_indices=None*, *steps_per_epoch=10*, *epochs=1000*, *min_epochs=0*, *patience=10*, *min_delta=0.0*, *verbose=1*, *progress_callback=None*, *progress_preamble=''*, *progress_print_interval=5.0*)
Fit using a generator. Does not support many of the features of fit(), such as random negative peptides.

Fitting proceeds until early stopping is hit, using the peptides, affinities, etc. given by the parameters starting with "**validation_**".

This is used for pre-training pan-allele models using data synthesized by the allele-specific models.

Parameters

generator [generator yielding (alleles, peptides, affinities) tuples] where alleles and peptides are lists of strings, and affinities is list of floats.

validation_peptide_encoding [EncodableSequences]

validation_affinities [list of float]

validation_allele_encoding [AlleleEncoding]

validation_inequalities [list of string]

validation_output_indices [list of int]

steps_per_epoch [int]

epochs [int]

min_epochs [int]

patience [int]

min_delta [float]

verbose [int]

progress_callback [thunk]

progress_preamble [string]

progress_print_interval [float]

fit(*self*, *peptides*, *affinities*, *allele_encoding=None*, *inequalities=None*, *output_indices=None*, *sample_weights=None*, *shuffle_permutation=None*, *verbose=1*, *progress_callback=None*, *progress_preamble=''*, *progress_print_interval=5.0*)
Fit the neural network.

Parameters

peptides [EncodableSequences or list of string]

affinities [list of float] nM affinities. Must be same length of as peptides.

allele_encoding [AlleleEncoding] If not specified, the model will be a single-allele predictor.

inequalities [list of string, each element one of ">", "<", or "=".] Inequalities to use for fitting. Same length as affinities. Each element must be one of ">", "<", or "=". For example, a ">" will train on y_pred > y_true for that element in the training set. Requires using a custom losses that support inequalities (e.g. mse_with_ineqalities). If None all inequalities are taken to be "=".

output_indices [list of int] For multi-output models only. Same length as affinities. Indicates the index of the output (starting from 0) for each training example.

sample_weights [list of float] If not specified, all samples (including random negatives added during training) will have equal weight. If specified, the random negatives will be assigned weight=1.0.

shuffle_permutation [list of int] Permutation (integer list) of same length as peptides and affinities If None, then a random permutation will be generated.

verbose [int] Keras verbosity level

progress_callback [function] No-argument function to call after each epoch.

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress update. Set to None to disable.

predict(*self*, *peptides*, *allele_encoding=None*, *batch_size=4096*, *output_index=0*)
Predict affinities.

If peptides are specified as EncodableSequences, then the predictions will be cached for this predictor as long as the EncodableSequences object remains in memory. The cache is keyed in the object identity of the EncodableSequences, not the sequences themselves. The cache is used only for allele-specific models (i.e. when allele_encoding is None).

Parameters

peptides [EncodableSequences or list of string]

allele_encoding [AlleleEncoding, optional] Only required when this model is a pan-allele model

batch_size [int] batch_size passed to Keras

output_index [int or None] For multi-output models. Gives the output index to return. If set to None, then all outputs are returned as a samples x outputs matrix.

Returns

numpy.array of nM affinity predictions

classmethod merge(*models*, *merge_method='average'*)

Merge multiple models at the tensorflow (or other backend) level.

Only certain neural network architectures support merging. Others will result in a NotImplementedError.

Parameters

models [list of Class1NeuralNetwork] instances to merge

merge_method [string, one of "average", "sum", or "concatenate"] How to merge the predictions of the different models

Returns

Class1NeuralNetwork The merged neural network

make_network(*self*, *peptide_encoding*, *allele_amino_acid_encoding*, *allele_dense_layer_sizes*, *peptide_dense_layer_sizes*, *peptide_allele_merge_method*, *peptide_allele_merge_activation*, *layer_sizes*, *dense_layer_l1_regularization*, *dense_layer_l2_regularization*, *activation*, *init*, *output_activation*, *dropout_probability*, *batch_normalization*, *locally_connected_layers*, *topology*, *num_outputs=1*, *allele_representations=None*)

Helper function to make a keras network for class 1 affinity prediction.

clear_allele_representations(*self*)

Set allele representations to an empty array. Useful before saving to save a smaller version of the model.

set_allele_representations(*self*, *allele_representations*, *force_surgery=False*)

Set the allele representations in use by this model. This means mutating the weights for the allele input embedding layer.

Rationale: instead of passing in the allele sequence for each data point during model training or prediction (which is expensive in terms of memory usage), we pass in an allele index between 0 and n-1 where n is the number of alleles in some universe of possible alleles. This index is used in the model to lookup the corresponding allele sequence. This function sets the lookup table.

See also: AlleleEncoding.allele_representations()

Parameters

allele_representations [numpy.ndarray of shape (a, l, m)]

where a is the total number of alleles, l is the allele sequence length, m is the length of the vectors used to represent amino acids

class mhcflurry.**Class1ProcessingPredictor**(*models*, *manifest_df=None*, *metadata_dataframes=None*)

Bases: object

User-facing interface to antigen processing prediction.

Delegates to an ensemble of Class1ProcessingNeuralNetwork instances.

Instantiate a new Class1ProcessingPredictor

Users will generally call load() to restore a saved predictor rather than using this constructor.

Parameters

models [list of Class1ProcessingNeuralNetwork] Neural networks in the ensemble.

manifest_df [pandas.DataFrame] Manifest dataframe. If not specified a new one will be created when needed.

metadata_dataframes [dict of string -> pandas.DataFrame] Arbitrary metadata associated with this predictor

property sequence_lengths

Supported maximum sequence lengths.

Passing a peptide greater than the maximum supported length results in an error.

Passing an N- or C-flank sequence greater than the maximum supported length results in some part of it being ignored.

Returns

dict of string -> int

Keys are "peptide", "n_flank", "c_flank". Values give the maximum

supported sequence length.

add_models(*self*, *models*)

Add models to the ensemble (in-place).

Parameters

models [list of Class1ProcessingNeuralNetwork]

Returns

list of string

Names of the new models.

property manifest_df

A pandas.DataFrame describing the models included in this predictor.

Returns

pandas.DataFrame

static model_name(*num*)

Generate a model name

Returns

string

static weights_path(*models_dir*, *model_name*)

Generate the path to the weights file for a model

Parameters

models_dir [string]

model_name [string]

Returns

string

predict (*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *batch_size=4096*)
Predict antigen processing.

Parameters

peptides [list of string] Peptide sequences

n_flanks [list of string] Upstream sequence before each peptide

c_flanks [list of string] Downstream sequence after each peptide

batch_size [int] Prediction keras batch size.

Returns

numpy.array

Processing scores. Range is 0-1, higher indicates more favorable

processing.

predict_to_dataframe (*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *batch_size=4096*)
Predict antigen processing.

See *predict* method for parameter descriptions.

Returns

pandas.DataFrame

Processing predictions are in the "score" column. Also includes

peptides and flanking sequences.

predict_to_dataframe_encoded (*self*, *sequences*, *batch_size=4096*)
Predict antigen processing.

See *predict* method for more information.

Parameters

sequences [FlankingEncoding]

batch_size [int]

Returns

pandas.DataFrame

check_consistency (*self*)
Verify that self.manifest_df is consistent with instance variables.

Currently only checks for agreement on the total number of models.

Throws AssertionError if inconsistent.

save (*self*, *models_dir*, *model_names_to_write=None*, *write_metadata=True*)
Serialize the predictor to a directory on disk. If the directory does not exist it will be created.

The serialization format consists of a file called "manifest.csv" with the configurations of each Class1ProcessingNeuralNetwork, along with per-network files giving the model weights.

Parameters

models_dir [string] Path to directory. It will be created if it doesn't exist.

classmethod load(*models_dir=None*, *max_models=None*)
Deserialize a predictor from a directory on disk.

Parameters

models_dir [string] Path to directory. If unspecified the default downloaded models are used.

max_models [int, optional] Maximum number of models to load

Returns

***Class1ProcessingPredictor* instance**

class mhcflurry.**Class1ProcessingNeuralNetwork**(***hyperparameters*)
Bases: object

A neural network for antigen processing prediction

network_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob
Hyperparameters (and their default values) that affect the neural network architecture.

fit_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object
Hyperparameters for neural network training.

early_stopping_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefa
Hyperparameters for early stopping.

compile_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob
Loss and optimizer hyperparameters. Any values supported by keras may be used.

auxiliary_input_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDef
Allele feature hyperparameters.

hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object>

property sequence_lengths
Supported maximum sequence lengths

Returns

dict of string -> int

Keys are "peptide", "n_flank", "c_flank". Values give the maximum

supported sequence length.

network(*self*)
Return the keras model associated with this network.

update_network_description(*self*)
Update self.network_json and self.network_weights properties based on this instances's neural network.

fit(*self*, *sequences*, *targets*, *sample_weights=None*, *shuffle_permutation=None*, *verbose=1*, *progress_callback=None*, *progress_preamble=''*, *progress_print_interval=5.0*)
Fit the neural network.

Parameters

sequences [FlankingEncoding] Peptides and upstream/downstream flanking sequences

targets [list of float] 1 indicates hit, 0 indicates decoy

sample_weights [list of float] If not specified all samples have equal weight.

shuffle_permutation [list of int] Permutation (integer list) of same length as peptides and affinities If None, then a random permutation will be generated.

verbose [int] Keras verbosity level

progress_callback [function] No-argument function to call after each epoch.

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress update. Set to None to disable.

predict(*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *batch_size=4096*)
Predict antigen processing.

Parameters

peptides [list of string] Peptide sequences

n_flanks [list of string] Upstream sequence before each peptide

c_flanks [list of string] Downstream sequence after each peptide

batch_size [int] Prediction keras batch size.

Returns

numpy.array

Processing scores. Range is 0-1, higher indicates more favorable

processing.

predict_encoded(*self*, *sequences*, *batch_size=4096*)
Predict antigen processing.

Parameters

sequences [FlankingEncoding] Peptides and flanking sequences

batch_size [int] Prediction keras batch size.

Returns

numpy.array

network_input(*self*, *sequences*)
Encode peptides to the fixed-length encoding expected by the neural network (which depends on the architecture).

Parameters

sequences [FlankingEncoding] Peptides and flanking sequences

Returns

numpy.array

make_network(*self*, *amino_acid_encoding*, *peptide_max_length*, *n_flank_length*, *c_flank_length*, *flanking_averages*, *convolutional_filters*, *convolutional_kernel_size*, *convolutional_activation*, *convolutional_kernel_l1_l2*, *dropout_rate*, *post_convolutional_dense_layer_sizes*)
Helper function to make a keras network given hyperparameters.

get_weights(*self*)
Get the network weights

Returns

list of numpy.array giving weights for each layer or None if there is no

network

get_config(*self*)
serialize to a dict all attributes except model weights

Returns

dict

classmethod from_config(*config*, *weights=None*)
deserialize from a dict returned by get_config().

Parameters

config [dict]

weights [list of array, optional] Network weights to restore

Returns

Class1ProcessingNeuralNetwork

class mhcflurry.**Class1PresentationPredictor**(*affinity_predictor=None*, *processing_predictor_with_flanks=None*, *processing_predictor_without_flanks=None*, *weights_dataframe=None*, *metadata_dataframes=None*)

Bases: object

A logistic regression model over predicted binding affinity (BA) and antigen processing (AP) score.

Instances of this class delegate to Class1AffinityPredictor and Class1ProcessingPredictor instances to generate BA and AP predictions. These predictions are combined using a logistic regression model to give a "presentation score" prediction.

Most users will call the *load* static method to get an instance of this class, then call the *predict* method to generate predictions.

model_inputs = ['affinity_score', 'processing_score']

property supported_alleles
List of alleles supported by the underlying Class1AffinityPredictor

property supported_peptide_lengths
(min, max) of supported peptide lengths, inclusive.

predict_affinity(*self*, *peptides*, *alleles*, *sample_names=None*, *include_affinity_percentile=True*, *verbose=1*, *throw=True*)
Predict binding affinities across samples (each corresponding to up to six MHC I alleles).

Two modes are supported: each peptide can be evaluated for binding to any of the alleles in any sample (this is what happens when sample_names is None), or the i'th peptide can be evaluated for binding the alleles of the sample given by the i'th entry in sample_names.

For example, if we don't specify sample_names, then predictions are taken for all combinations of samples and peptides, for a result size of num peptides * num samples:

```
>>> predictor = Class1PresentationPredictor.load()
>>> predictor.predict_affinity(
...     peptides=["SIINFEKL", "PEPTIDE"],
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
```

(continues on next page)

(continued from previous page)

```
...     verbose=0)
    peptide  peptide_num sample_name      affinity best_allele
0  SIINFEKL            0     sample1  12906.787792       A0201
1   PEPTIDE            1     sample1  36827.681130       B0702
2  SIINFEKL            0     sample2   3588.413748       C0202
3   PEPTIDE            1     sample2  34362.109211       C0202
```

In contrast, here we specify sample_names, so peptide is evaluated for binding the alleles in the corresponding sample, for a result size equal to the number of peptides:

```
>>> predictor.predict_affinity(
...     peptides=["SIINFEKL", "PEPTIDE"],
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     sample_names=["sample2", "sample1"],
...     verbose=0)
    peptide  peptide_num sample_name      affinity best_allele
0  SIINFEKL            0     sample2   3588.412141       C0202
1   PEPTIDE            1     sample1  36827.682779       B0702
```

Parameters

peptides [list of string] Peptide sequences

alleles [dict of string -> list of string] Keys are sample names, values are the alleles (genotype) for that sample

sample_names [list of string [same length as peptides]] Sample names corresponding to each peptide. If None, then predictions are generated for all sample genotypes across all peptides.

include_affinity_percentile [bool] Whether to include affinity percentile ranks

verbose [int] Set to 0 for quiet.

throw [verbose] Whether to throw exception (vs. just log a warning) on invalid peptides, etc.

Returns

pandas.DataFrame [predictions]

predict_processing(*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *verbose=1*)

Predict antigen processing scores for individual peptides, optionally including flanking sequences for better cleavage prediction.

Parameters

peptides [list of string]

n_flanks [list of string [same length as peptides]]

c_flanks [list of string [same length as peptides]]

verbose [int]

Returns

numpy.array [Antigen processing scores for each peptide]

fit (*self*, *targets*, *peptides*, *sample_names*, *alleles*, *n_flanks=None*, *c_flanks=None*, *verbose=1*)
Fit the presentation score logistic regression model.

Parameters

targets [list of int/float] 1 indicates hit, 0 indicates decoy

peptides [list of string [same length as targets]]

sample_names [list of string [same length as targets]]

alleles [dict of string -> list of string] Keys are sample names, values are the alleles for that sample

n_flanks [list of string [same length as targets]]

c_flanks [list of string [same length as targets]]

verbose [int]

get_model (*self*, *name=None*)
Load or instantiate a new logistic regression model. Private helper method.

Parameters

name [string] If None (the default), an un-fit LR model is returned. Otherwise the weights are loaded for the specified model.

Returns

sklearn.linear_model.LogisticRegression

predict (*self*, *peptides*, *alleles*, *sample_names=None*, *n_flanks=None*, *c_flanks=None*, *include_affinity_percentile=False*, *verbose=1*, *throw=True*)
Predict presentation scores across a set of peptides.

Presentation scores combine predictions for MHC I binding affinity and antigen processing.

This method returns a pandas.DataFrame giving presentation scores plus the binding affinity and processing predictions and other intermediate results.

Example:

```
>>> predictor = Class1PresentationPredictor.load()
>>> predictor.predict(
...     peptides=["SIINFEKL", "PEPTIDE"],
...     n_flanks=["NNN", "SNS"],
...     c_flanks=["CCC", "CNC"],
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     verbose=0)
    peptide n_flank c_flank  peptide_num sample_name      affinity best_
↪allele  processing_score  presentation_score
0  SIINFEKL     NNN     CCC            0     sample1  12906.787792
↪A0201          0.802466           0.140365
1   PEPTIDE     SNS     CNC            1     sample1  36827.681130
↪B0702          0.105260           0.004059
2  SIINFEKL     NNN     CCC            0     sample2   3588.413748
↪C0202          0.802466           0.338647
3   PEPTIDE     SNS     CNC            1     sample2  34362.109211
↪C0202          0.105260           0.004317
```

You can also specify sample_names, in which case peptide is evaluated for binding the alleles in the corresponding sample only. See *predict_affinity* for an examples.

Parameters

peptides [list of string] Peptide sequences

alleles [list of string or dict of string -> list of string] If you are predicting for a single sample, pass a list of strings (up to 6) indicating the genotype. If you are predicting across multiple samples, pass a dict where the keys are (arbitrary) sample names and the values are the alleles to predict for that sample.

sample_names [list of string [same length as peptides]] If you are passing a dict for 'alleles', you can use this argument to specify which peptides go with which samples. If it is None, then predictions will be performed for each peptide across all samples.

n_flanks [list of string [same length as peptides]] Upstream sequences before the peptide. Sequences of any length can be given and a suffix of the size supported by the model will be used.

c_flanks [list of string [same length as peptides]] Downstream sequences after the peptide. Sequences of any length can be given and a prefix of the size supported by the model will be used.

include_affinity_percentile [bool] Whether to include affinity percentile ranks

verbose [int] Set to 0 for quiet.

throw [verbose] Whether to throw exception (vs. just log a warning) on invalid peptides, etc.

Returns

pandas.DataFrame

Presentation scores and intermediate results.

predict_sequences(*self*, *sequences*, *alleles*, *result='best'*, *comparison_quantity='presentation_score'*, *filter_value=None*, *peptide_lengths=(8, 9, 10, 11)*, *use_flanks=True*, *include_affinity_percentile=True*, *verbose=1*, *throw=True*)

Predict presentation across protein sequences.

Example:

```
>>> predictor = Class1PresentationPredictor.load()
>>> predictor.predict_sequences(
...     sequences={
...         'protein1': "MDSKGSSQKGSRLLLLLVVSNLL",
...         'protein2': "SSLPTPEDKEQAQQTHH",
...     },
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     result="filtered",
...     comparison_quantity="affinity",
...     filter_value=500,
...     verbose=0)
  sequence_name  pos     peptide         n_flank     c_flank sample_name
↪affinity best_allele  affinity_percentile  processing_score  presentation_
↪score
```

(continues on next page)

(continued from previous page)

```
0      protein1   13   LLLLVVSNL   MDSKGSSQKGSRL          L     sample1   38.
↪206225       A0201             0.380125          0.017644            0.
↪571060
1      protein1   14   LLLVVSNLL  MDSKGSSQKGSRLL                sample1   42.
↪243472       A0201             0.420250          0.090984            0.
↪619213
2      protein1    5   SSQKGSRLL           MDSKG   LLLVVSNLL     sample2   66.
↪749223       C0202             0.803375          0.383608            0.
↪774468
3      protein1    6   SQKGSRLLL          MDSKGS    LLVVSNLL     sample2  178.
↪033474       C0202             1.820000          0.275019            0.
↪482206
4      protein1   13  LLLLVVSNLL   MDSKGSSQKGSRL                sample1  202.
↪208167       A0201             1.112500          0.058782            0.
↪261320
5      protein1   12  LLLLLVVSNL    MDSKGSSQKGSR          L     sample1  202.
↪506582       A0201             1.112500          0.010025            0.
↪225648
6      protein2    0   SSLPTPEDK                    EQAQQTHH     sample1  335.
↪529377       A0301             1.011750          0.010443            0.
↪156798
7      protein2    0   SSLPTPEDK                    EQAQQTHH     sample2  353.
↪451759       C0202             2.674250          0.010443            0.
↪150753
8      protein1    8   KGSRLLLLL        MDSKGSSQ      VVSNLL     sample2  410.
↪327286       C0202             2.887000          0.121374            0.
↪194081
9      protein1    5    SSQKGSRL           MDSKG  LLLLVVSNLL     sample2  477.
↪285954       C0202             3.107375          0.111982            0.
↪168572
```

Parameters

sequences [str, list of string, or string -> string dict] Protein sequences. If a dict is given, the keys are arbitrary (e.g. protein names), and the values are the amino acid sequences.

alleles [list of string, list of list of string, or dict of string -> list of string] MHC I alleles. Can be: (1) a string (a single allele), (2) a list of strings (a single genotype), (3) a list of list of strings (multiple genotypes, where the total number of genotypes must equal the number of sequences), or (4) a dict giving multiple genotypes, which will each be run over the sequences.

result [string] Specify 'best' to return the strongest peptide for each sequence, 'all' to return predictions for all peptides, or 'filtered' to return predictions where the comparison_quantity is stronger (i.e (<) for affinity, (>) for scores) than filter_value.

comparison_quantity [string] One of "presentation_score", "processing_score", "affinity", or "affinity_percentile". Prediction to use to rank (if result is "best") or filter (if result is "filtered") results.

filter_value [float] Threshold value to use, only relevant when result is "filtered". If comparison_quantity is "affinity", then all results less than (i.e. tighter than) the specified nM affinity are retained. If it's "presentation_score" or "processing_score" then results greater than the indicated filter_value are retained.

peptide_lengths [list of int] Peptide lengths to predict for.

use_flanks [bool] Whether to include flanking sequences when running the AP predictor

(for better cleavage prediction).

include_affinity_percentile [bool] Whether to include affinity percentile ranks in output.

verbose [int] Set to 0 for quiet mode.

throw [boolean] Whether to throw exceptions (vs. log warnings) on invalid inputs.

Returns

pandas.DataFrame with columns: peptide, n_flank, c_flank, sequence_name, affinity, best_allele, processing_score, presentation_score

save(*self*, *models_dir*)

Save the predictor to a directory on disk. If the directory does not exist it will be created.

The wrapped Class1AffinityPredictor and Class1ProcessingPredictor instances are included in the saved data.

Parameters

models_dir [string] Path to directory. It will be created if it doesn't exist.

classmethod load(*models_dir=None*, *max_models=None*)

Deserialize a predictor from a directory on disk.

This will also load the wrapped Class1AffinityPredictor and Class1ProcessingPredictor instances.

Parameters

models_dir [string] Path to directory. If unspecified the default downloaded models are used.

max_models [int, optional] Maximum number of affinity and processing (counted separately) models to load

Returns

Class1PresentationPredictor **instance**

5.1 Submodules

5.2 mhcflurry.allele_encoding module

class mhcflurry.allele_encoding.**AlleleEncoding**(*alleles=None*, *allele_to_sequence=None*, *borrow_from=None*)

Bases: `object`

A place to cache encodings for a sequence of alleles.

We frequently work with alleles by integer indices, for example as inputs to neural networks. This class is used to map allele names to integer indices in a consistent way by keeping track of the universe of alleles under use, i.e. a distinction is made between the universe of supported alleles (what's in `allele_to_sequence`) and the actual set of alleles used for some task (what's in `alleles`).

Parameters

alleles [list of string] Allele names. If any allele is None instead of string, it will be mapped to the special index value -1.

allele_to_sequence [dict of str -> str] Allele name to amino acid sequence

borrow_from [AlleleEncoding, optional] If specified, do not specify allele_to_sequence. The sequences from the provided instance are used. This guarantees that the mappings from allele to index and from allele to sequence are the same between the instances.

compact(*self*)
Return a new AlleleEncoding in which the universe of supported alleles is only the alleles actually used.

Returns

AlleleEncoding

allele_representations(*self*, *encoding_name*)
Encode the universe of supported allele sequences to a matrix.

Parameters

encoding_name [string] How to represent amino acids. Valid names are "BLOSUM62" or "one-hot". See `amino_acid.ENCODING_DATA_FRAMES`.

Returns

numpy.array of shape (num alleles in universe, sequence length, vector size)

where vector size is usually 21 (20 amino acids + X character)

fixed_length_vector_encoded_sequences(*self*, *encoding_name*)
Encode allele sequences (not the universe of alleles) to a matrix.

Parameters

encoding_name [string] How to represent amino acids. Valid names are "BLOSUM62" or "one-hot". See `amino_acid.ENCODING_DATA_FRAMES`.

Returns

numpy.array with shape: (num alleles, sequence length, vector size)

where vector size is usually 21 (20 amino acids + X character)

5.3 mhcflurry.amino_acid module

Functions for encoding fixed length sequences of amino acids into various vector representations, such as one-hot and BLOSUM62.

mhcflurry.amino_acid.**available_vector_encodings**()
Return list of supported amino acid vector encodings.

Returns

list of string

mhcflurry.amino_acid.**vector_encoding_length**(*name*)
Return the length of the given vector encoding.

Parameters

name [string]

Returns

int

mhcflurry.amino_acid.**index_encoding**(*sequences*, *letter_to_index_dict*)

Encode a sequence of same-length strings to a matrix of integers of the same shape. The map from characters to integers is given by `letter_to_index_dict`.

Given a sequence of `n` strings all of length `k`, return a `k * n` array where the (`i`, `j`)th element is `letter_to_index_dict[sequence[i][j]]`.

Parameters

sequences [list of length n of strings of length k]

letter_to_index_dict [dict]

Returns

numpy.array of integers with shape (k, n)

mhcflurry.amino_acid.**fixed_vectors_encoding**(*index_encoded_sequences*, *letter_to_vector_df*)

Given a `n x k` matrix of integers such as that returned by *index_encoding()* and a dataframe mapping each index to an arbitrary vector, return a `n * k * m` array where the (`i`, `j`)'th element is `letter_to_vector_df.iloc[sequence[i][j]]`.

The dataframe index and columns names are ignored here; the indexing is done entirely by integer position in the dataframe.

Parameters

index_encoded_sequences [`n` x `k` array of integers]

letter_to_vector_df [pandas.DataFrame of shape (`alphabet size`, `m`)]

Returns

numpy.array of integers with shape (n, k, m)

5.4 mhcflurry.calibrate_percentile_ranks_command module

Calibrate percentile ranks for models. Runs in-place.

mhcflurry.calibrate_percentile_ranks_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.calibrate_percentile_ranks_command.**do_calibrate_percentile_ranks**(*alleles*, *constant_data={}*)

mhcflurry.calibrate_percentile_ranks_command.**calibrate_percentile_ranks**(*allele*, *predictor*, *peptides=None*, *motif_summary=False*, *summary_top_peptide_fraction*, *verbose=False*, *model_kwargs={}*)

5.5 mhcflurry.class1_affinity_predictor module

class mhcflurry.class1_affinity_predictor.**Class1AffinityPredictor**(*allele_to_allele_specific_models=None*, *class1_pan_allele_models=None*, *allele_to_sequence=None*, *manifest_df=None*, *allele_to_percent_rank_transform=None*, *metadata_dataframes=None*)

Bases: `object`

High-level interface for peptide/MHC I binding affinity prediction.

This class manages low-level `Class1NeuralNetwork` instances, each of which wraps a single Keras network. The purpose of *`Class1AffinityPredictor`* is to implement ensembles, handling of multiple alleles, and predictor loading and saving. It also provides a place to keep track of metadata like prediction histograms for percentile rank calibration.

Parameters

allele_to_allele_specific_models [dict of string -> list of `Class1NeuralNetwork`] Ensemble of single-allele models to use for each allele.

class1_pan_allele_models [list of `Class1NeuralNetwork`] Ensemble of pan-allele models.

allele_to_sequence [dict of string -> string] MHC allele name to fixed-length amino acid sequence (sometimes referred to as the pseudosequence). Required only if class1_pan_allele_models is specified.

manifest_df [`pandas.DataFrame`, optional] Must have columns: model_name, allele, config_json, model. Only required if you want to update an existing serialization of a Class1AffinityPredictor. Otherwise this dataframe will be generated automatically based on the supplied models.

allele_to_percent_rank_transform [dict of string -> `PercentRankTransform`, optional] `PercentRankTransform` instances to use for each allele

metadata_dataframes [dict of string -> pandas.DataFrame, optional] Optional additional dataframes to write to the models dir when save() is called. Useful for tracking provenance.

property manifest_df

A pandas.DataFrame describing the models included in this predictor.

Based on: - self.class1_pan_allele_models - self.allele_to_allele_specific_models

Returns

pandas.DataFrame

clear_cache(*self*)

Clear values cached based on the neural networks in this predictor.

Users should call this after mutating any of the following:

- self.class1_pan_allele_models
- self.allele_to_allele_specific_models

- self.allele_to_sequence

Methods that mutate these instance variables will call this method on their own if needed.

property neural_networks
List of the neural networks in the ensemble.

Returns

list of Class1NeuralNetwork

classmethod merge(*predictors*)
Merge the ensembles of two or more *Class1AffinityPredictor* instances.

Note: the resulting merged predictor will NOT have calibrated percentile ranks. Call *calibrate_percentile_ranks* on it if these are needed.

Parameters

predictors [sequence of *Class1AffinityPredictor*]

Returns

***Class1AffinityPredictor* instance**

merge_in_place(*self*, *others*)
Add the models present in other predictors into the current predictor.

Parameters

others [list of Class1AffinityPredictor] Other predictors to merge into the current predictor.

Returns

list of string [names of newly added models]

property supported_alleles
Alleles for which predictions can be made.

Returns

list of string

property supported_peptide_lengths
(minimum, maximum) lengths of peptides supported by *all models*, inclusive.

Returns

(int, int) tuple

check_consistency(*self*)
Verify that self.manifest_df is consistent with: - self.class1_pan_allele_models - self.allele_to_allele_specific_models

Currently only checks for agreement on the total number of models.

Throws AssertionError if inconsistent.

save(*self*, *models_dir*, *model_names_to_write=None*, *write_metadata=True*)
Serialize the predictor to a directory on disk. If the directory does not exist it will be created.

The serialization format consists of a file called "manifest.csv" with the configurations of each Class1NeuralNetwork, along with per-network files giving the model weights. If there are pan-allele predictors in the ensemble, the allele sequences are also stored in the directory. There is also a small file "index.txt" with basic metadata: when the models were trained, by whom, on what host.

Parameters

models_dir [string] Path to directory. It will be created if it doesn't exist.

model_names_to_write [list of string, optional] Only write the weights for the specified models. Useful for incremental updates during training.

write_metadata [boolean, optional] Whether to write optional metadata

static load(*models_dir=None*, *max_models=None*, *optimization_level=None*)

Deserialize a predictor from a directory on disk.

Parameters

models_dir [string] Path to directory. If unspecified the default downloaded models are used.

max_models [int, optional] Maximum number of `Class1NeuralNetwork` instances to load

optimization_level [int] If >0, model optimization will be attempted. Defaults to value of environment variable MHCFLURRY_OPTIMIZATION_LEVEL.

Returns

Class1AffinityPredictor **instance**

optimize(*self*, *warn=True*)

EXPERIMENTAL: Optimize the predictor for faster predictions.

Currently the only optimization implemented is to merge multiple pan- allele predictors at the tensorflow level.

The optimization is performed in-place, mutating the instance.

Returns

bool Whether optimization was performed

static model_name(*allele*, *num*)

Generate a model name

Parameters

allele [string]

num [int]

Returns

string

static weights_path(*models_dir*, *model_name*)

Generate the path to the weights file for a model

Parameters

models_dir [string]

model_name [string]

Returns

string

property master_allele_encoding

An AlleleEncoding containing the universe of alleles specified by self.allele_to_sequence.

Returns

AlleleEncoding

fit_allele_specific_predictors(*self*, *n_models*, *architecture_hyperparameters_list*, *allele*, *peptides*, *affinities*, *inequalities=None*, *train_rounds=None*, *models_dir_for_save=None*, *verbose=0*, *progress_preamble=''*, *progress_print_interval=5.0*)

Fit one or more allele specific predictors for a single allele using one or more neural network architectures.

The new predictors are saved in the Class1AffinityPredictor instance and will be used on subsequent calls to *predict*.

Parameters

n_models [int] Number of neural networks to fit

architecture_hyperparameters_list [list of dict] List of hyperparameter sets.

allele [string]

peptides [`EncodableSequences` or list of string]

affinities [list of float] nM affinities

inequalities [list of string, each element one of ">", "<", or "="] See `Class1NeuralNetwork.fit` for details.

train_rounds [sequence of int] Each training point i will be used on training rounds r for which train_rounds[i] > r, r >= 0.

models_dir_for_save [string, optional] If specified, the Class1AffinityPredictor is (incrementally) written to the given models dir after each neural network is fit.

verbose [int] Keras verbosity

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress. Set to None to disable.

Returns

list of `Class1NeuralNetwork`

fit_class1_pan_allele_models(*self*, *n_models*, *architecture_hyperparameters*, *alleles*, *peptides*, *affinities*, *inequalities*, *models_dir_for_save=None*, *verbose=1*, *progress_preamble=''*, *progress_print_interval=5.0*)

Fit one or more pan-allele predictors using a single neural network architecture.

The new predictors are saved in the Class1AffinityPredictor instance and will be used on subsequent calls to *predict*.

Parameters

n_models [int] Number of neural networks to fit

architecture_hyperparameters [dict]

alleles [list of string] Allele names (not sequences) corresponding to each peptide

peptides [`EncodableSequences` or list of string]

affinities [list of float] nM affinities

inequalities [list of string, each element one of ">", "<", or "="] See Class1NeuralNetwork.fit for details.

models_dir_for_save [string, optional] If specified, the Class1AffinityPredictor is (incrementally) written to the given models dir after each neural network is fit.

verbose [int] Keras verbosity

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress. Set to None to disable.

Returns

list of `Class1NeuralNetwork`

`add_pan_allele_model`(*self*, *model*, *models_dir_for_save=None*)

Add a pan-allele model to the ensemble and optionally do an incremental save.

Parameters

model [Class1NeuralNetwork]

models_dir_for_save [string] Directory to save resulting ensemble to

`percentile_ranks`(*self*, *affinities*, *allele=None*, *alleles=None*, *throw=True*)

Return percentile ranks for the given ic50 affinities and alleles.

The 'allele' and 'alleles' argument are as in the *`predict`* method. Specify one of these.

Parameters

affinities [sequence of float] nM affinities

allele [string]

alleles [sequence of string]

throw [boolean] If True, a ValueError will be raised in the case of unsupported alleles. If False, a warning will be logged and NaN will be returned for those percentile ranks.

Returns

numpy.array of float

`predict`(*self*, *peptides*, *alleles=None*, *allele=None*, *throw=True*, *centrality_measure='mean'*, *model_kwargs={}*)

Predict nM binding affinities.

If multiple predictors are available for an allele, the predictions are the geometric means of the individual model (nM) predictions.

One of 'allele' or 'alleles' must be specified. If 'allele' is specified all predictions will be for the given allele. If 'alleles' is specified it must be the same length as 'peptides' and give the allele corresponding to each peptide.

Parameters

peptides [`EncodableSequences` or list of string]

alleles [list of string]

allele [string]

throw [boolean] If True, a ValueError will be raised in the case of unsupported alleles or peptide lengths. If False, a warning will be logged and the predictions for the unsupported alleles or peptides will be NaN.

centrality_measure [string or callable] Measure of central tendency to use to combine predictions in the ensemble. Options include: mean, median, robust_mean.

model_kwargs [dict] Additional keyword arguments to pass to Class1NeuralNetwork.predict

Returns

numpy.array of predictions

predict_to_dataframe(*self*, *peptides*, *alleles=None*, *allele=None*, *throw=True*, *include_individual_model_predictions=False*, *include_percentile_ranks=True*, *include_confidence_intervals=True*, *centrality_measure='mean'*, *model_kwargs={}*)

Predict nM binding affinities. Gives more detailed output than `predict` method, including 5-95% prediction intervals.

If multiple predictors are available for an allele, the predictions are the geometric means of the individual model predictions.

One of 'allele' or 'alleles' must be specified. If 'allele' is specified all predictions will be for the given allele. If 'alleles' is specified it must be the same length as 'peptides' and give the allele corresponding to each peptide.

Parameters

peptides [`EncodableSequences` or list of string]

alleles [list of string]

allele [string]

throw [boolean] If True, a ValueError will be raised in the case of unsupported alleles or peptide lengths. If False, a warning will be logged and the predictions for the unsupported alleles or peptides will be NaN.

include_individual_model_predictions [boolean] If True, the predictions of each individual model are included as columns in the result DataFrame.

include_percentile_ranks [boolean, default True] If True, a "prediction_percentile" column will be included giving the percentile ranks. If no percentile rank info is available, this will be ignored with a warning.

centrality_measure [string or callable] Measure of central tendency to use to combine predictions in the ensemble. Options include: mean, median, robust_mean.

model_kwargs [dict] Additional keyword arguments to pass to Class1NeuralNetwork.predict

Returns

`pandas.DataFrame` of predictions

calibrate_percentile_ranks(*self, peptides=None, num_peptides_per_length=100000, alleles=None, bins=None, motif_summary=False, summary_top_peptide_fractions=[0.001], verbose=False, model_kwargs={}*)

Compute the cumulative distribution of ic50 values for a set of alleles over a large universe of random peptides, to enable taking quantiles of this distribution later.

Parameters

peptides [sequence of string or EncodableSequences, optional] Peptides to use

num_peptides_per_length [int, optional] If peptides argument is not specified, then num_peptides_per_length peptides are randomly sampled from a uniform distribution for each supported length

alleles [sequence of string, optional] Alleles to perform calibration for. If not specified all supported alleles will be calibrated.

bins [object] Anything that can be passed to numpy.histogram's "bins" argument can be used here, i.e. either an integer or a sequence giving bin edges. This is in ic50 space.

motif_summary [bool] If True, the length distribution and per-position amino acid frequencies are also calculated for the top x fraction of tightest- binding peptides, where each value of x is given in the summary_top_peptide_fractions list.

summary_top_peptide_fractions [list of float] Only used if motif_summary is True

verbose [boolean] Whether to print status updates to stdout

model_kwargs [dict] Additional low-level Class1NeuralNetwork.predict() kwargs.

Returns

dict of string -> pandas.DataFrame

If motif_summary is True, this will have keys "frequency_matrices" and

"length_distributions". Otherwise it will be empty.

model_select(*self*, *score_function*, *alleles=None*, *min_models=1*, *max_models=10000*)

Perform model selection using a user-specified scoring function.

This works only with allele-specific models, not pan-allele models.

Model selection is done using a "step up" variable selection procedure, in which models are repeatedly added to an ensemble until the score stops improving.

Parameters

score_function [Class1AffinityPredictor -> float function] Scoring function

alleles [list of string, optional] If not specified, model selection is performed for all alleles.

min_models [int, optional] Min models to select per allele

max_models [int, optional] Max models to select per allele

Returns

Class1AffinityPredictor [predictor containing the selected models]

5.6 mhcflurry.class1_neural_network module

class mhcflurry.class1_neural_network.**Class1NeuralNetwork**(***hyperparameters*)

Bases: object

Low level class I predictor consisting of a single neural network.

Both single allele and pan-allele prediction are supported.

Users will generally use Class1AffinityPredictor, which gives a higher-level interface and supports ensembles.

network_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob

Hyperparameters (and their default values) that affect the neural network architecture.

compile_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob
Loss and optimizer hyperparameters.

fit_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object
Hyperparameters for neural network training.

early_stopping_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefa
Hyperparameters for early stopping.

miscelaneous_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaul
Miscelaneous hyperaparameters. These parameters are not used by this class but may be interpreted by other code.

hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object>
Combined set of all supported hyperparameters and their default values.

hyperparameter_renames = {'embedding_init_method': None, 'embedding_input_dim': None

classmethod apply_hyperparameter_renames(*hyperparameters*)
Handle hyperparameter renames.

Parameters

hyperparameters [dict]

Returns

dict [updated hyperparameters]

KERAS_MODELS_CACHE = {}
Process-wide keras model cache, a map from: architecture JSON string to (Keras model, existing network weights)

classmethod clear_model_cache()
Clear the Keras model cache.

classmethod borrow_cached_network(*network_json*, *network_weights*)
Return a keras Model with the specified architecture and weights. As an optimization, when possible this will reuse architectures from a process-wide cache.

The returned object is "borrowed" in the sense that its weights can change later after subsequent calls to this method from other objects.

If you're using this from a parallel implementation you'll need to hold a lock while using the returned object.

Parameters

network_json [string of JSON]

network_weights [list of numpy.array]

Returns

keras.models.Model

network(*self*, *borrow=False*)
Return the keras model associated with this predictor.

Parameters

borrow [bool] Whether to return a cached model if possible. See borrow_cached_network for details

Returns

keras.models.Model

update_network_description (*self*)
Update self.network_json and self.network_weights properties based on this instances's neural network.

static keras_network_cache_key (*network_json*)
Given a Keras JSON description of a neural network, return a key that uniquely defines this network. Networks that share the same key should have compatible weights matrices and give the same prediction outputs when their weights are the same.

Parameters

network_json [string]

Returns

string

get_config (*self*)
serialize to a dict all attributes except model weights

Returns

dict

classmethod from_config (*config*, *weights=None*, *weights_loader=None*)
deserialize from a dict returned by get_config().

Parameters

config [dict]

weights [list of array, optional] Network weights to restore

weights_loader [callable, optional] Function to call (no arguments) to load weights when needed

Returns

Class1NeuralNetwork

load_weights (*self*)
Load weights by evaluating self.network_weights_loader, if needed.

After calling this, self.network_weights_loader will be None and self.network_weights will be the weights list, if available.

get_weights (*self*)
Get the network weights

Returns

list of numpy.array giving weights for each layer or None if there is no

network

peptides_to_network_input (*self*, *peptides*)
Encode peptides to the fixed-length encoding expected by the neural network (which depends on the architecture).

Parameters

peptides [EncodableSequences or list of string]

Returns

numpy.array

property supported_peptide_lengths
(minimum, maximum) lengths of peptides supported, inclusive.

Returns

(int, int) tuple

allele_encoding_to_network_input(*self*, *allele_encoding*)
Encode alleles to the fixed-length encoding expected by the neural network (which depends on the architecture).

Parameters

allele_encoding [AlleleEncoding]

Returns

(numpy.array, numpy.array)

Indices and allele representations.

static data_dependent_weights_initialization(*network*, *x_dict=None*, *method='lsuv'*, *verbose=1*)
Data dependent weights initialization.

Parameters

network [keras.Model]

x_dict [dict of string -> numpy.ndarray] Training data as would be passed keras.Model.fit().

method [string] Initialization method. Currently only "lsuv" is supported.

verbose [int] Status updates printed to stdout if verbose > 0

fit_generator(*self*, *generator*, *validation_peptide_encoding*, *validation_affinities*, *validation_allele_encoding=None*, *validation_inequalities=None*, *validation_output_indices=None*, *steps_per_epoch=10*, *epochs=1000*, *min_epochs=0*, *patience=10*, *min_delta=0.0*, *verbose=1*, *progress_callback=None*, *progress_preamble=''*, *progress_print_interval=5.0*)
Fit using a generator. Does not support many of the features of fit(), such as random negative peptides.

Fitting proceeds until early stopping is hit, using the peptides, affinities, etc. given by the parameters starting with "**validation_**".

This is used for pre-training pan-allele models using data synthesized by the allele-specific models.

Parameters

generator [generator yielding (alleles, peptides, affinities) tuples] where alleles and peptides are lists of strings, and affinities is list of floats.

validation_peptide_encoding [EncodableSequences]

validation_affinities [list of float]

validation_allele_encoding [AlleleEncoding]

validation_inequalities [list of string]

validation_output_indices [list of int]

steps_per_epoch [int]

epochs [int]

min_epochs [int]

patience [int]

min_delta [float]

verbose [int]

progress_callback [thunk]

progress_preamble [string]

progress_print_interval [float]

fit(*self*, *peptides*, *affinities*, *allele_encoding=None*, *inequalities=None*, *output_indices=None*, *sample_weights=None*, *shuffle_permutation=None*, *verbose=1*, *progress_callback=None*, *progress_preamble=''*, *progress_print_interval=5.0*)
Fit the neural network.

Parameters

peptides [EncodableSequences or list of string]

affinities [list of float] nM affinities. Must be same length of as peptides.

allele_encoding [AlleleEncoding] If not specified, the model will be a single-allele predictor.

inequalities [list of string, each element one of ">", "<", or "=".] Inequalities to use for fitting. Same length as affinities. Each element must be one of ">", "<", or "=". For example, a ">" will train on y_pred > y_true for that element in the training set. Requires using a custom losses that support inequalities (e.g. mse_with_ineqalities). If None all inequalities are taken to be "=".

output_indices [list of int] For multi-output models only. Same length as affinities. Indicates the index of the output (starting from 0) for each training example.

sample_weights [list of float] If not specified, all samples (including random negatives added during training) will have equal weight. If specified, the random negatives will be assigned weight=1.0.

shuffle_permutation [list of int] Permutation (integer list) of same length as peptides and affinities If None, then a random permutation will be generated.

verbose [int] Keras verbosity level

progress_callback [function] No-argument function to call after each epoch.

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress update. Set to None to disable.

predict(*self*, *peptides*, *allele_encoding=None*, *batch_size=4096*, *output_index=0*)
Predict affinities.

If peptides are specified as EncodableSequences, then the predictions will be cached for this predictor as long as the EncodableSequences object remains in memory. The cache is keyed in the object identity of the EncodableSequences, not the sequences themselves. The cache is used only for allele-specific models (i.e. when allele_encoding is None).

Parameters

peptides [EncodableSequences or list of string]

allele_encoding [AlleleEncoding, optional] Only required when this model is a pan-allele model

batch_size [int] batch_size passed to Keras

output_index [int or None] For multi-output models. Gives the output index to return. If set to None, then all outputs are returned as a samples x outputs matrix.

Returns

numpy.array of nM affinity predictions

classmethod merge(*models*, *merge_method='average'*)

Merge multiple models at the tensorflow (or other backend) level.

Only certain neural network architectures support merging. Others will result in a NotImplementedError.

Parameters

models [list of Class1NeuralNetwork] instances to merge

merge_method [string, one of "average", "sum", or "concatenate"] How to merge the predictions of the different models

Returns

Class1NeuralNetwork The merged neural network

make_network(*self*, *peptide_encoding*, *allele_amino_acid_encoding*, *allele_dense_layer_sizes*, *peptide_dense_layer_sizes*, *peptide_allele_merge_method*, *peptide_allele_merge_activation*, *layer_sizes*, *dense_layer_l1_regularization*, *dense_layer_l2_regularization*, *activation*, *init*, *output_activation*, *dropout_probability*, *batch_normalization*, *locally_connected_layers*, *topology*, *num_outputs=1*, *allele_representations=None*)

Helper function to make a keras network for class 1 affinity prediction.

clear_allele_representations(*self*)

Set allele representations to an empty array. Useful before saving to save a smaller version of the model.

set_allele_representations(*self*, *allele_representations*, *force_surgery=False*)

Set the allele representations in use by this model. This means mutating the weights for the allele input embedding layer.

Rationale: instead of passing in the allele sequence for each data point during model training or prediction (which is expensive in terms of memory usage), we pass in an allele index between 0 and n-1 where n is the number of alleles in some universe of possible alleles. This index is used in the model to lookup the corresponding allele sequence. This function sets the lookup table.

See also: AlleleEncoding.allele_representations()

Parameters

allele_representations [numpy.ndarray of shape (a, l, m)]

where a is the total number of alleles, l is the allele sequence length, m is the length of the vectors used to represent amino acids

5.7 mhcflurry.class1_presentation_predictor module

class mhcflurry.class1_presentation_predictor.**Class1PresentationPredictor**(*affinity_predictor=None*, *processing_predictor_with_flanking=None*, *processing_predictor_without_flanking=None*, *weights_dataframe=None*, *metadata_dataframes=None*)

Bases: `object`

A logistic regression model over predicted binding affinity (BA) and antigen processing (AP) score.

Instances of this class delegate to Class1AffinityPredictor and Class1ProcessingPredictor instances to generate BA and AP predictions. These predictions are combined using a logistic regression model to give a "presentation score" prediction.

Most users will call the *load* static method to get an instance of this class, then call the *predict* method to generate predictions.

model_inputs = ['affinity_score', 'processing_score']

property supported_alleles
List of alleles supported by the underlying Class1AffinityPredictor

property supported_peptide_lengths
(min, max) of supported peptide lengths, inclusive.

predict_affinity(*self*, *peptides*, *alleles*, *sample_names=None*, *include_affinity_percentile=True*, *verbose=1*, *throw=True*)
Predict binding affinities across samples (each corresponding to up to six MHC I alleles).

Two modes are supported: each peptide can be evaluated for binding to any of the alleles in any sample (this is what happens when sample_names is None), or the i'th peptide can be evaluated for binding the alleles of the sample given by the i'th entry in sample_names.

For example, if we don't specify sample_names, then predictions are taken for all combinations of samples and peptides, for a result size of num peptides * num samples:

```
>>> predictor = Class1PresentationPredictor.load()
>>> predictor.predict_affinity(
...     peptides=["SIINFEKL", "PEPTIDE"],
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     verbose=0)
    peptide  peptide_num sample_name      affinity best_allele
0  SIINFEKL            0     sample1  12906.787792       A0201
1   PEPTIDE            1     sample1  36827.681130       B0702
2  SIINFEKL            0     sample2   3588.413748       C0202
3   PEPTIDE            1     sample2  34362.109211       C0202
```

In contrast, here we specify sample_names, so peptide is evaluated for binding the alleles in the corresponding sample, for a result size equal to the number of peptides:

```
>>> predictor.predict_affinity(
...     peptides=["SIINFEKL", "PEPTIDE"],
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     sample_names=["sample2", "sample1"],
...     verbose=0)
    peptide  peptide_num sample_name      affinity best_allele
0  SIINFEKL            0     sample2   3588.412141       C0202
1   PEPTIDE            1     sample1  36827.682779       B0702
```

Parameters

peptides [list of string] Peptide sequences

alleles [dict of string -> list of string] Keys are sample names, values are the alleles (genotype) for that sample

sample_names [list of string [same length as peptides]] Sample names corresponding to each peptide. If None, then predictions are generated for all sample genotypes across all peptides.

include_affinity_percentile [bool] Whether to include affinity percentile ranks

verbose [int] Set to 0 for quiet.

throw [verbose] Whether to throw exception (vs. just log a warning) on invalid peptides, etc.

Returns

pandas.DataFrame [predictions]

predict_processing(*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *verbose=1*)

Predict antigen processing scores for individual peptides, optionally including flanking sequences for better cleavage prediction.

Parameters

peptides [list of string]

n_flanks [list of string [same length as peptides]]

c_flanks [list of string [same length as peptides]]

verbose [int]

Returns

numpy.array [Antigen processing scores for each peptide]

fit(*self*, *targets*, *peptides*, *sample_names*, *alleles*, *n_flanks=None*, *c_flanks=None*, *verbose=1*)

Fit the presentation score logistic regression model.

Parameters

targets [list of int/float] 1 indicates hit, 0 indicates decoy

peptides [list of string [same length as targets]]

sample_names [list of string [same length as targets]]

alleles [dict of string -> list of string] Keys are sample names, values are the alleles for that sample

n_flanks [list of string [same length as targets]]

c_flanks [list of string [same length as targets]]

verbose [int]

get_model(*self*, *name=None*)

Load or instantiate a new logistic regression model. Private helper method.

Parameters

name [string] If None (the default), an un-fit LR model is returned. Otherwise the weights are loaded for the specified model.

Returns

sklearn.linear_model.LogisticRegression

predict(*self*, *peptides*, *alleles*, *sample_names=None*, *n_flanks=None*, *c_flanks=None*, *include_affinity_percentile=False*, *verbose=1*, *throw=True*)

Predict presentation scores across a set of peptides.

Presentation scores combine predictions for MHC I binding affinity and antigen processing.

This method returns a pandas.DataFrame giving presentation scores plus the binding affinity and processing predictions and other intermediate results.

Example:

```
>>> predictor = Class1PresentationPredictor.load()
>>> predictor.predict(
...     peptides=["SIINFEKL", "PEPTIDE"],
...     n_flanks=["NNN", "SNS"],
...     c_flanks=["CCC", "CNC"],
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     verbose=0)
    peptide n_flank c_flank  peptide_num sample_name      affinity best_
↪allele  processing_score  presentation_score
0  SIINFEKL     NNN     CCC            0     sample1  12906.787792
↪A0201          0.802466           0.140365
1   PEPTIDE     SNS     CNC            1     sample1  36827.681130
↪B0702          0.105260           0.004059
2  SIINFEKL     NNN     CCC            0     sample2   3588.413748
↪C0202          0.802466           0.338647
3   PEPTIDE     SNS     CNC            1     sample2  34362.109211
↪C0202          0.105260           0.004317
```

You can also specify sample_names, in which case peptide is evaluated for binding the alleles in the corresponding sample only. See *predict_affinity* for an examples.

Parameters

peptides [list of string] Peptide sequences

alleles [list of string or dict of string -> list of string] If you are predicting for a single sample, pass a list of strings (up to 6) indicating the genotype. If you are predicting across multiple

samples, pass a dict where the keys are (arbitrary) sample names and the values are the alleles to predict for that sample.

sample_names [list of string [same length as peptides]] If you are passing a dict for 'alleles', you can use this argument to specify which peptides go with which samples. If it is None, then predictions will be performed for each peptide across all samples.

n_flanks [list of string [same length as peptides]] Upstream sequences before the peptide. Sequences of any length can be given and a suffix of the size supported by the model will be used.

c_flanks [list of string [same length as peptides]] Downstream sequences after the peptide. Sequences of any length can be given and a prefix of the size supported by the model will be used.

include_affinity_percentile [bool] Whether to include affinity percentile ranks

verbose [int] Set to 0 for quiet.

throw [verbose] Whether to throw exception (vs. just log a warning) on invalid peptides, etc.

Returns

pandas.DataFrame

Presentation scores and intermediate results.

predict_sequences(*self*, *sequences*, *alleles*, *result='best'*, *comparison_quantity='presentation_score'*, *filter_value=None*, *peptide_lengths=(8, 9, 10, 11)*, *use_flanks=True*, *include_affinity_percentile=True*, *verbose=1*, *throw=True*)

Predict presentation across protein sequences.

Example:

```
>>> predictor = Class1PresentationPredictor.load()
>>> predictor.predict_sequences(
...     sequences={
...         'protein1': "MDSKGSSQKGSRLLLLLVVSNLL",
...         'protein2': "SSLPTPEDKEQAQQTHH",
...     },
...     alleles={
...         "sample1": ["A0201", "A0301", "B0702"],
...         "sample2": ["A0101", "C0202"],
...     },
...     result="filtered",
...     comparison_quantity="affinity",
...     filter_value=500,
...     verbose=0)
  sequence_name  pos     peptide         n_flank     c_flank sample_name    ␣
↪affinity best_allele  affinity_percentile  processing_score  presentation_
↪score
0      protein1   13   LLLLVVSNL   MDSKGSSQKGSRL           L     sample1   38.
↪206225       A0201             0.380125          0.017644            0.
↪571060
1      protein1   14   LLLVVSNLL  MDSKGSSQKGSRLL                 sample1   42.
↪243472       A0201             0.420250          0.090984            0.
↪619213
2      protein1    5   SSQKGSRLL           MDSKG   LLLVVSNLL     sample2   66.
↪749223       C0202             0.803375          0.383608            0.
↪774468
```

(continues on next page)

(continued from previous page)

```
3      protein1    6   SQKGSRLLL          MDSKGS    LLVVSNLL     sample2  178.
↪033474       C0202             1.820000          0.275019            0.
↪482206
4      protein1   13  LLLLVVSNLL   MDSKGSSQKGSRL                  sample1  202.
↪208167       A0201             1.112500          0.058782            0.
↪261320
5      protein1   12  LLLLLVVSNL    MDSKGSSQKGSR          L      sample1  202.
↪506582       A0201             1.112500          0.010025            0.
↪225648
6      protein2    0   SSLPTPEDK                    EQAQQTHH     sample1  335.
↪529377       A0301             1.011750          0.010443            0.
↪156798
7      protein2    0   SSLPTPEDK                    EQAQQTHH     sample2  353.
↪451759       C0202             2.674250          0.010443            0.
↪150753
8      protein1    8   KGSRLLLLL        MDSKGSSQ      VVSNLL     sample2  410.
↪327286       C0202             2.887000          0.121374            0.
↪194081
9      protein1    5    SSQKGSRL           MDSKG  LLLLVVSNLL     sample2  477.
↪285954       C0202             3.107375          0.111982            0.
↪168572
```

Parameters

sequences [str, list of string, or string -> string dict] Protein sequences. If a dict is given, the keys are arbitrary (e.g. protein names), and the values are the amino acid sequences.

alleles [list of string, list of list of string, or dict of string -> list of string] MHC I alleles. Can be: (1) a string (a single allele), (2) a list of strings (a single genotype), (3) a list of list of strings (multiple genotypes, where the total number of genotypes must equal the number of sequences), or (4) a dict giving multiple genotypes, which will each be run over the sequences.

result [string] Specify 'best' to return the strongest peptide for each sequence, 'all' to return predictions for all peptides, or 'filtered' to return predictions where the comparison_quantity is stronger (i.e (<) for affinity, (>) for scores) than filter_value.

comparison_quantity [string] One of "presentation_score", "processing_score", "affinity", or "affinity_percentile". Prediction to use to rank (if result is "best") or filter (if result is "filtered") results.

filter_value [float] Threshold value to use, only relevant when result is "filtered". If comparison_quantity is "affinity", then all results less than (i.e. tighter than) the specified nM affinity are retained. If it's "presentation_score" or "processing_score" then results greater than the indicated filter_value are retained.

peptide_lengths [list of int] Peptide lengths to predict for.

use_flanks [bool] Whether to include flanking sequences when running the AP predictor (for better cleavage prediction).

include_affinity_percentile [bool] Whether to include affinity percentile ranks in output.

verbose [int] Set to 0 for quiet mode.

throw [boolean] Whether to throw exceptions (vs. log warnings) on invalid inputs.

Returns

pandas.DataFrame with columns: peptide, n_flank, c_flank, sequence_name, affinity, best_allele, processing_score, presentation_score

save (*self*, *models_dir*)
Save the predictor to a directory on disk. If the directory does not exist it will be created.

The wrapped Class1AffinityPredictor and Class1ProcessingPredictor instances are included in the saved data.

Parameters

models_dir [string] Path to directory. It will be created if it doesn't exist.

classmethod load (*models_dir=None*, *max_models=None*)
Deserialize a predictor from a directory on disk.

This will also load the wrapped Class1AffinityPredictor and Class1ProcessingPredictor instances.

Parameters

models_dir [string] Path to directory. If unspecified the default downloaded models are used.

max_models [int, optional] Maximum number of affinity and processing (counted separately) models to load

Returns

Class1PresentationPredictor **instance**

5.8 mhcflurry.class1_processing_neural_network module

Antigen processing neural network implementation

class mhcflurry.class1_processing_neural_network.**Class1ProcessingNeuralNetwork** (***hyperparamete*
Bases: object

A neural network for antigen processing prediction

network_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob
Hyperparameters (and their default values) that affect the neural network architecture.

fit_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object
Hyperparameters for neural network training.

early_stopping_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefa
Hyperparameters for early stopping.

compile_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults ob
Loss and optimizer hyperparameters. Any values supported by keras may be used.

auxiliary_input_hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDef
Allele feature hyperparameters.

hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object>

property sequence_lengths
Supported maximum sequence lengths

Returns

dict of string -> int

Keys are "peptide", "n_flank", "c_flank". Values give the maximum

supported sequence length.

network (*self*)
Return the keras model associated with this network.

update_network_description (*self*)
Update self.network_json and self.network_weights properties based on this instances's neural network.

fit (*self*, *sequences*, *targets*, *sample_weights=None*, *shuffle_permutation=None*, *verbose=1*, *progress_callback=None*, *progress_preamble=''*, *progress_print_interval=5.0*)
Fit the neural network.

Parameters

sequences [FlankingEncoding] Peptides and upstream/downstream flanking sequences

targets [list of float] 1 indicates hit, 0 indicates decoy

sample_weights [list of float] If not specified all samples have equal weight.

shuffle_permutation [list of int] Permutation (integer list) of same length as peptides and affinities If None, then a random permutation will be generated.

verbose [int] Keras verbosity level

progress_callback [function] No-argument function to call after each epoch.

progress_preamble [string] Optional string of information to include in each progress update

progress_print_interval [float] How often (in seconds) to print progress update. Set to None to disable.

predict (*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *batch_size=4096*)
Predict antigen processing.

Parameters

peptides [list of string] Peptide sequences

n_flanks [list of string] Upstream sequence before each peptide

c_flanks [list of string] Downstream sequence after each peptide

batch_size [int] Prediction keras batch size.

Returns

numpy.array

Processing scores. Range is 0-1, higher indicates more favorable

processing.

predict_encoded (*self*, *sequences*, *batch_size=4096*)
Predict antigen processing.

Parameters

sequences [FlankingEncoding] Peptides and flanking sequences

batch_size [int] Prediction keras batch size.

Returns

numpy.array

network_input (*self*, *sequences*)
Encode peptides to the fixed-length encoding expected by the neural network (which depends on the architecture).

Parameters

sequences [FlankingEncoding] Peptides and flanking sequences

Returns

numpy.array

make_network (*self*, *amino_acid_encoding*, *peptide_max_length*, *n_flank_length*, *c_flank_length*, *flanking_averages*, *convolutional_filters*, *convolutional_kernel_size*, *convolutional_activation*, *convolutional_kernel_l1_l2*, *dropout_rate*, *post_convolutional_dense_layer_sizes*)
Helper function to make a keras network given hyperparameters.

get_weights (*self*)
Get the network weights

Returns

list of numpy.array giving weights for each layer or None if there is no

network

get_config (*self*)
serialize to a dict all attributes except model weights

Returns

dict

classmethod from_config (*config*, *weights=None*)
deserialize from a dict returned by get_config().

Parameters

config [dict]

weights [list of array, optional] Network weights to restore

Returns

Class1ProcessingNeuralNetwork

5.9 mhcflurry.class1_processing_predictor module

class mhcflurry.class1_processing_predictor.**Class1ProcessingPredictor** (*models*, *manifest_df=None*, *metadata_dataframes=None*)

Bases: object

User-facing interface to antigen processing prediction.

Delegates to an ensemble of Class1ProcessingNeuralNetwork instances.

Instantiate a new Class1ProcessingPredictor

Users will generally call load() to restore a saved predictor rather than using this constructor.

Parameters

models [list of Class1ProcessingNeuralNetwork] Neural networks in the ensemble.

manifest_df [pandas.DataFrame] Manifest dataframe. If not specified a new one will be created when needed.

metadata_dataframes [dict of string -> pandas.DataFrame] Arbitrary metadata associated with this predictor

property sequence_lengths

Supported maximum sequence lengths.

Passing a peptide greater than the maximum supported length results in an error.

Passing an N- or C-flank sequence greater than the maximum supported length results in some part of it being ignored.

Returns

dict of string -> int

Keys are "peptide", "n_flank", "c_flank". Values give the maximum

supported sequence length.

add_models(*self*, *models*)

Add models to the ensemble (in-place).

Parameters

models [list of Class1ProcessingNeuralNetwork]

Returns

list of string

Names of the new models.

property manifest_df

A pandas.DataFrame describing the models included in this predictor.

Returns

pandas.DataFrame

static model_name(*num*)

Generate a model name

Returns

string

static weights_path(*models_dir*, *model_name*)

Generate the path to the weights file for a model

Parameters

models_dir [string]

model_name [string]

Returns

string

predict(*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *batch_size=4096*)

Predict antigen processing.

Parameters

peptides [list of string] Peptide sequences

n_flanks [list of string] Upstream sequence before each peptide

c_flanks [list of string] Downstream sequence after each peptide

batch_size [int] Prediction keras batch size.

Returns

numpy.array

Processing scores. Range is 0-1, higher indicates more favorable

processing.

predict_to_dataframe(*self*, *peptides*, *n_flanks=None*, *c_flanks=None*, *batch_size=4096*)
Predict antigen processing.

See *predict* method for parameter descriptions.

Returns

pandas.DataFrame

Processing predictions are in the "score" column. Also includes

peptides and flanking sequences.

predict_to_dataframe_encoded(*self*, *sequences*, *batch_size=4096*)
Predict antigen processing.

See *predict* method for more information.

Parameters

sequences [FlankingEncoding]

batch_size [int]

Returns

pandas.DataFrame

check_consistency(*self*)
Verify that self.manifest_df is consistent with instance variables.

Currently only checks for agreement on the total number of models.

Throws AssertionError if inconsistent.

save(*self*, *models_dir*, *model_names_to_write=None*, *write_metadata=True*)
Serialize the predictor to a directory on disk. If the directory does not exist it will be created.

The serialization format consists of a file called "manifest.csv" with the configurations of each Class1ProcessingNeuralNetwork, along with per-network files giving the model weights.

Parameters

models_dir [string] Path to directory. It will be created if it doesn't exist.

classmethod load(*models_dir=None*, *max_models=None*)
Deserialize a predictor from a directory on disk.

Parameters

models_dir [string] Path to directory. If unspecified the default downloaded models are used.

max_models [int, optional] Maximum number of models to load

Returns

Class1ProcessingPredictor **instance**

5.10 mhcflurry.cluster_parallelism module

Simple, relatively naive parallel map implementation for HPC clusters.

Used for training MHCflurry models.

mhcflurry.cluster_parallelism.**add_cluster_parallelism_args**(*parser*)

Add commandline arguments controlling cluster parallelism to an argparse ArgumentParser.

Parameters

parser [argparse.ArgumentParser]

mhcflurry.cluster_parallelism.**cluster_results_from_args**(*args*, *work_function*, *work_items*, *constant_data=None*, *input_serialization_method='pickle'*, *result_serialization_method='pickle'*, *clear_constant_data=False*)

Parallel map configurable using commandline arguments. See the cluster_results() function for docs.

The `args` parameter should be an argparse.Namespace from an argparse parser generated using the add_cluster_parallelism_args() function.

Parameters

args

work_function

work_items

constant_data

result_serialization_method

clear_constant_data

Returns

generator

mhcflurry.cluster_parallelism.**cluster_results**(*work_function*, *work_items*, *constant_data=None*, *submit_command='sh'*, *results_workdir='./cluster-workdir'*, *additional_complete_file=None*, *script_prefix_path=None*, *input_serialization_method='pickle'*, *result_serialization_method='pickle'*, *max_retries=3*, *clear_constant_data=False*)

Parallel map on an HPC cluster.

Returns [work_function(item) for item in work_items] where each invocation of work_function is performed as a separate HPC cluster job. Order is preserved.

Optionally, "constant data" can be specified, which will be passed to each work_function() invocation as a keyword argument called constant_data. This data is serialized once and all workers read it from the same source, which is more efficient than serializing it separately for each worker.

Each worker's input is serialized to a shared NFS directory and the submit_command is used to launch a job to process that input. The shared filesystem is polled occasionally to watch for results, which are fed back to the user.

Parameters

work_function [A -> B]

work_items [list of A]

constant_data [object]

submit_command [string] For running on LSF, we use "bsub" here.

results_workdir [string] Path to NFS shared directory where inputs and results can be written

script_prefix_path [string] Path to script that will be invoked to run each worker. A line calling the _mhcflurry-cluster-worker-entry-point command will be appended to the contents of this file.

result_serialization_method [string, one of "pickle" or "save_predictor"] The "save_predictor" works only when the return type of work_function is Class1AffinityPredictor

max_retries [int] How many times to attempt to re-launch a failed worker

clear_constant_data [bool] If True, the constant data dict is cleared on the launching host after it is serialized to disk.

Returns

generator of B

mhcflurry.cluster_parallelism.**worker_entry_point**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

Entry point for the worker command.

Parameters

argv [list of string]

5.11 mhcflurry.common module

mhcflurry.common.**set_keras_backend**(*backend=None*, *gpu_device_nums=None*, *num_threads=None*)

Configure Keras backend to use GPU or CPU. Only tensorflow is supported.

Parameters

backend [string, optional] one of 'tensorflow-default', 'tensorflow-cpu', 'tensorflow-gpu'

gpu_device_nums [list of int, optional] GPU devices to potentially use

num_threads [int, optional] Tensorflow threads to use

mhcflurry.common.**configure_logging**(*verbose=False*)

Configure logging module using defaults.

Parameters

verbose [boolean] If true, output will be at level DEBUG, otherwise, INFO.

mhcflurry.common.**amino_acid_distribution**(*peptides*, *smoothing=0.0*)
Compute the fraction of each amino acid across a collection of peptides.

Parameters

peptides [list of string]

smoothing [float, optional] Small number (e.g. 0.01) to add to all amino acid fractions. The higher the number the more uniform the distribution.

Returns

pandas.Series indexed by amino acids

mhcflurry.common.**random_peptides**(*num*, *length=9*, *distribution=None*)
Generate random peptides (kmers).

Parameters

num [int] Number of peptides to return

length [int] Length of each peptide

distribution [pandas.Series] Maps 1-letter amino acid abbreviations to probabilities. If not specified a uniform distribution is used.

Returns

list of string

mhcflurry.common.**positional_frequency_matrix**(*peptides*)
Given a set of peptides, calculate a length x amino acids frequency matrix.

Parameters

peptides [list of string] All of same length

Returns

pandas.DataFrame Index is position, columns are amino acids

mhcflurry.common.**save_weights**(*weights_list*, *filename*)
Save model weights to the given filename using numpy's ".npz" format.

Parameters

weights_list [list of numpy array]

filename [string]

mhcflurry.common.**load_weights**(*filename*)
Restore model weights from the given filename, which should have been created with *save_weights*.

Parameters

filename [string]

Returns

list of array

class mhcflurry.common.**NumpyJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

JSON encoder (used with json module) that can handle numpy arrays.

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, float or None. If skipkeys is True, such items are simply skipped.

If ensure_ascii is true, the output is guaranteed to be str objects with all incoming non-ASCII characters escaped. If ensure_ascii is false, the output can contain non-ASCII characters.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation.

If specified, separators should be an (item_separator, key_separator) tuple. The default is (', ', ': ') if *indent* is None and (',', ': ') otherwise. To get the most compact JSON representation, you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a TypeError.

default(*self*, *obj*)

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

5.12 mhcflurry.custom_loss module

Custom loss functions.

For losses supporting inequalities, each training data point is associated with one of (=), (<), or (>). For e.g. (>) inequalities, penalization is applied only if the prediction is less than the given value.

mhcflurry.custom_loss.**get_loss**(*name*)

Get a custom_loss.Loss instance by name.

Parameters

name [string]

Returns

custom_loss.Loss

class mhcflurry.custom_loss.**Loss**(*name=None*)

Bases: object

Thin wrapper to keep track of neural network loss functions, which could be custom or baked into Keras.

Each subclass or instance should define these properties/methods: - name : string - loss : string or function

This is what gets passed to keras.fit()

- **encode_y** [numpy.ndarray -> numpy.ndarray] Transformation to apply to regression target before fitting

loss(*self*, *y_true*, *y_pred*)

get_keras_loss(*self*, *reduction='sum_over_batch_size'*)

class mhcflurry.custom_loss.**StandardKerasLoss**(*loss_name='mse'*)

Bases: *mhcflurry.custom_loss.Loss*

A loss function supported by Keras, such as MSE.

supports_inequalities = False

supports_multiple_outputs = False

static encode_y(*y*)

class mhcflurry.custom_loss.**TransformPredictionsLossWrapper**(*loss*, *y_pred_transform=None*)

Bases: *mhcflurry.custom_loss.Loss*

Wrapper that applies an arbitrary transform to y_pred before calling an underlying loss function.

The y_pred_transform function should be a tensor -> tensor function.

encode_y(*self*, **args*, ***kwargs*)

loss(*self*, *y_true*, *y_pred*)

class mhcflurry.custom_loss.**MSEWithInequalities**(*name=None*)

Bases: *mhcflurry.custom_loss.Loss*

Supports training a regression model on data that includes inequalities (e.g. x < 100). Mean square error is used as the loss for elements with an (=) inequality. For elements with e.g. a (> 0.5) inequality, then the loss for that element is (y - 0.5)^2 (standard MSE) if y < 500 and 0 otherwise.

This loss assumes that the normal range for y_true and y_pred is 0 - 1. As a hack, the implementation uses other intervals for y_pred to encode the inequality information.

y_true is interpreted as follows:

between 0 - 1 Regular MSE loss is used. Penalty (y_pred - y_true)**2 is applied if y_pred is greater or less than y_true.

between 2 - 3: Treated as a ">" inequality. Penalty (y_pred - (y_true - 2))**2 is applied only if y_pred is less than y_true - 2.

between 4 - 5: Treated as a "<" inequality. Penalty (y_pred - (y_true - 4))**2 is applied only if y_pred is greater than y_true - 4.

name = 'mse_with_inequalities'

supports_inequalities = True

supports_multiple_outputs = False

static encode_y(*y*, *inequalities=None*)

loss(*self*, *y_true*, *y_pred*)

class mhcflurry.custom_loss.**MSEWithInequalitiesAndMultipleOutputs**(*name=None*)
Bases: *mhcflurry.custom_loss.Loss*

Loss supporting inequalities and multiple outputs.

This loss assumes that the normal range for y_true and y_pred is 0 - 1. As a hack, the implementation uses other intervals for y_pred to encode the inequality and output-index information.

Inequalities are encoded into the regression target as in the MSEWithInequalities loss.

Multiple outputs are encoded by mapping each regression target x (after transforming for inequalities) using the rule x -> x + i * 10 where i is the output index.

The reason for explicitly encoding multiple outputs this way (rather than just making the regression target a matrix instead of a vector) is that in our use cases we frequently have missing data in the regression target. This encoding gives a simple way to penalize only on (data point, output index) pairs that have labels.

name = 'mse_with_inequalities_and_multiple_outputs'

supports_inequalities = True

supports_multiple_outputs = True

static encode_y(*y*, *inequalities=None*, *output_indices=None*)

loss(*self*, *y_true*, *y_pred*)

class mhcflurry.custom_loss.**MultiallelicMassSpecLoss**(*delta=0.2*, *multiplier=1.0*)
Bases: *mhcflurry.custom_loss.Loss*

name = 'multiallelic_mass_spec_loss'

supports_inequalities = True

supports_multiple_outputs = False

static encode_y(*y*)

loss(*self*, *y_true*, *y_pred*)

mhcflurry.custom_loss.**check_shape**(*name*, *arr*, *expected_shape*)
Raise ValueError if arr.shape != expected_shape.

Parameters

name [string] Included in error message to aid debugging

arr [numpy.ndarray]

expected_shape [tuple of int]

mhcflurry.custom_loss.**cls**
alias of *mhcflurry.custom_loss.MultiallelicMassSpecLoss*

5.13 mhcflurry.data_dependent_weights_initialization module

Layer-sequential unit-variance initialization for neural networks.

See: Mishkin and Matas, "All you need is a good init". 2016. https://arxiv.org/abs/1511.06422

mhcflurry.data_dependent_weights_initialization.**svd_orthonormal**(*shape*)

mhcflurry.data_dependent_weights_initialization.**get_activations**(*model*, *layer*, *X_batch*)

mhcflurry.data_dependent_weights_initialization.**lsuv_init**(*model*, *batch*, *verbose=True*, *margin=0.1*, *max_iter=100*)
Initialize neural network weights using layer-sequential unit-variance initialization.

See: Mishkin and Matas, "All you need is a good init". 2016. https://arxiv.org/abs/1511.06422

Parameters

model [keras.Model]

batch [dict] Training data, as would be passed keras.Model.fit()

verbose [boolean] Whether to print progress to stdout

margin [float]

max_iter [int]

Returns

keras.Model Same as what was passed in.

5.14 mhcflurry.downloads module

Manage local downloaded data.

mhcflurry.downloads.**get_downloads_dir**()
Return the path to local downloaded data

mhcflurry.downloads.**get_current_release**()
Return the current downloaded data release

mhcflurry.downloads.**get_downloads_metadata**()
Return the contents of downloads.yml as a dict

mhcflurry.downloads.**get_default_class1_models_dir**(*test_exists=True*)
Return the absolute path to the default class1 models dir.

If environment variable MHCFLURRY_DEFAULT_CLASS1_MODELS is set to an absolute path, return that path. If it's set to a relative path (i.e. does not start with /) then return that path taken to be relative to the mhcflurry downloads dir.

If environment variable MHCFLURRY_DEFAULT_CLASS1_MODELS is NOT set, then return the path to downloaded models in the "models_class1" download.

Parameters

test_exists [boolean, optional] Whether to raise an exception of the path does not exist

Returns

string [absolute path]

mhcflurry.downloads.**get_default_class1_presentation_models_dir**(*test_exists=True*)
Return the absolute path to the default class1 presentation models dir.

See *get_default_class1_models_dir*.

If environment variable MHCFLURRY_DEFAULT_CLASS1_PRESENTATION_MODELS is set to an absolute path, return that path. If it's set to a relative path (does not start with /) then return that path taken to be relative to the mhcflurry downloads dir.

Parameters

test_exists [boolean, optional] Whether to raise an exception of the path does not exist

Returns

string [absolute path]

mhcflurry.downloads.**get_default_class1_processing_models_dir**(*test_exists=True*)
Return the absolute path to the default class1 processing models dir.

See *get_default_class1_models_dir*.

If environment variable MHCFLURRY_DEFAULT_CLASS1_PROCESSING_MODELS is set to an absolute path, return that path. If it's set to a relative path (does not start with /) then return that path taken to be relative to the mhcflurry downloads dir.

Parameters

test_exists [boolean, optional] Whether to raise an exception of the path does not exist

Returns

string [absolute path]

mhcflurry.downloads.**get_current_release_downloads**()
Return a dict of all available downloads in the current release.

The dict keys are the names of the downloads. The values are a dict with two entries:

downloaded [bool] Whether the download is currently available locally

metadata [dict] Info about the download from downloads.yml such as URL

up_to_date [bool or None] Whether the download URL(s) match what was used to download the current data. This is None if it cannot be determined.

mhcflurry.downloads.**get_path**(*download_name*, *filename=''*, *test_exists=True*)
Get the local path to a file in a MHCflurry download

Parameters

download_name [string]

filename [string] Relative path within the download to the file of interest

test_exists [boolean] If True (default) throw an error telling the user how to download the data if the file does not exist

Returns

string giving local absolute path

mhcflurry.downloads.**configure**()
Setup various global variables based on environment variables.

5.15 mhcflurry.downloads_command module

Download MHCflurry released datasets and trained models.

Examples

Fetch the default downloads: $ mhcflurry-downloads fetch

Fetch a specific download: $ mhcflurry-downloads fetch models_class1_pan

Get the path to a download: $ mhcflurry-downloads path models_class1_pan

Get the URL of a download: $ mhcflurry-downloads url models_class1_pan

Summarize available and fetched downloads: $ mhcflurry-downloads info

mhcflurry.downloads_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.downloads_command.**mkdir_p**(*path*)
Make directories as needed, similar to mkdir -p in a shell.

From: http://stackoverflow.com/questions/600268/mkdir-p-functionality-in-python

mhcflurry.downloads_command.**yes_no**(*boolean*)

class mhcflurry.downloads_command.**TqdmUpTo**(*iterable=None*, *desc=None*, *total=None*, *leave=True*, *file=None*, *ncols=None*, *mininterval=0.1*, *maxinterval=10.0*, *miniters=None*, *ascii=None*, *disable=False*, *unit='it'*, *unit_scale=False*, *dynamic_ncols=False*, *smoothing=0.3*, *bar_format=None*, *initial=0*, *position=None*, *postfix=None*, *unit_divisor=1000*, *gui=False*, ***kwargs*)
Bases: tqdm._tqdm.tqdm

Provides update_to(n) which uses tqdm.update(delta_n).

Parameters

iterable [iterable, optional] Iterable to decorate with a progressbar. Leave blank to manually manage the updates.

desc [str, optional] Prefix for the progressbar.

total [int, optional] The number of expected iterations. If unspecified, len(iterable) is used if possible. As a last resort, only basic progress statistics are displayed (no ETA, no progressbar). If gui is True and this parameter needs subsequent updating, specify an initial arbitrary large positive integer, e.g. int(9e9).

leave [bool, optional] If [default: True], keeps all traces of the progressbar upon termination of iteration.

file [io.TextIOWrapper or io.StringIO, optional] Specifies where to output the progress messages (default: sys.stderr). Uses file.write(str) and file.flush() methods.

ncols [int, optional] The width of the entire output message. If specified, dynamically resizes the progressbar to stay within this bound. If unspecified, attempts to use environment width. The fallback is a meter width of 10 and no limit for the counter and statistics. If 0, will not print any meter (only stats).

mininterval [float, optional] Minimum progress display update interval, in seconds [default: 0.1].

maxinterval [float, optional] Maximum progress display update interval, in seconds [default: 10]. Automatically adjusts `miniters` to correspond to `mininterval` after long display update lag. Only works if `dynamic_miniters` or monitor thread is enabled.

miniters [int, optional] Minimum progress display update interval, in iterations. If 0 and `dynamic_miniters`, will automatically adjust to equal `mininterval` (more CPU efficient, good for tight loops). If > 0, will skip display of specified number of iterations. Tweak this and `mininterval` to get very efficient loops. If your progress is erratic with both fast and slow iterations (network, skipping items, etc) you should set miniters=1.

ascii [bool, optional] If unspecified or False, use unicode (smooth blocks) to fill the meter. The fallback is to use ASCII characters `1-9 #`.

disable [bool, optional] Whether to disable the entire progressbar wrapper [default: False]. If set to None, disable on non-TTY.

unit [str, optional] String that will be used to define the unit of each iteration [default: it].

unit_scale [bool or int or float, optional] If 1 or True, the number of iterations will be reduced/scaled automatically and a metric prefix following the International System of Units standard will be added (kilo, mega, etc.) [default: False]. If any other non-zero number, will scale `total` and `n`.

dynamic_ncols [bool, optional] If set, constantly alters `ncols` to the environment (allowing for window resizes) [default: False].

smoothing [float, optional] Exponential moving average smoothing factor for speed estimates (ignored in GUI mode). Ranges from 0 (average speed) to 1 (current/instantaneous speed) [default: 0.3].

bar_format [str, optional] Specify a custom bar string formatting. May impact performance. [default: '{l_bar}{bar}{r_bar}'], where l_bar='{desc}: {percentage:3.0f}%|' and r_bar='| {n_fmt}/{total_fmt} [{elapsed}<{remaining}, '

'{rate_fmt}{postfix}]'

Possible vars: l_bar, bar, r_bar, n, n_fmt, total, total_fmt, percentage, rate, rate_fmt, rate_noinv, rate_noinv_fmt, rate_inv, rate_inv_fmt, elapsed, remaining, desc, postfix.

Note that a trailing ": " is automatically removed after {desc} if the latter is empty.

initial [int, optional] The initial counter value. Useful when restarting a progress bar [default: 0].

position [int, optional] Specify the line offset to print this bar (starting from 0) Automatic if unspecified. Useful to manage multiple bars at once (eg, from threads).

postfix [dict or *, optional] Specify additional stats to display at the end of the bar. Calls `set_postfix(**postfix)` if possible (dict).

unit_divisor [float, optional] [default: 1000], ignored unless `unit_scale` is True.

gui [bool, optional] WARNING: internal parameter - do not use. Use tqdm_gui(...) instead. If set, will attempt to use matplotlib animations for a graphical output [default: False].

Returns

out [decorated iterator.]

update_to(*self*, *b=1*, *bsize=1*, *tsize=None*)

b [int, optional] Number of blocks transferred so far [default: 1].

bsize [int, optional] Size of each block (in tqdm units) [default: 1].

tsize [int, optional] Total size (in tqdm units). If [default: None] remains unchanged.

mhcflurry.downloads_command.**fetch_subcommand**(*args*)

mhcflurry.downloads_command.**info_subcommand**(*args*)

mhcflurry.downloads_command.**path_subcommand**(*args*)
Print the local path to a download

mhcflurry.downloads_command.**url_subcommand**(*args*)
Print the URL(s) for a download

5.16 mhcflurry.encodable_sequences module

Class for encoding variable-length peptides to fixed-size numerical matrices

exception mhcflurry.encodable_sequences.**EncodingError**(*message*, *supported_peptide_lengths*)
Bases: ValueError

Exception raised when peptides cannot be encoded

class mhcflurry.encodable_sequences.**EncodableSequences**(*sequences*)
Bases: object

Class for encoding variable-length peptides to fixed-size numerical matrices

This class caches various encodings of a list of sequences.

In practice this is used only for peptides. To encode MHC allele sequences, see AlleleEncoding.

unknown_character = 'X'

classmethod create(*sequences*)
Factory that returns an EncodableSequences given a list of strings. As a convenience, you can also pass it an EncodableSequences instance, in which case the object is returned unchanged.

variable_length_to_fixed_length_categorical(*self*, *alignment_method='pad_middle'*, *left_edge=4*, *right_edge=4*, *max_length=15*)
Encode variable-length sequences to a fixed-size index-encoded (integer) matrix.

See *sequences_to_fixed_length_index_encoded_array* for details.

Parameters

alignment_method [string] One of "pad_middle" or "left_pad_right_pad"

left_edge [int, size of fixed-position left side] Only relevant for pad_middle alignment method

right_edge [int, size of the fixed-position right side] Only relevant for pad_middle alignment method

max_length [maximum supported peptide length]

Returns

numpy.array of integers with shape (num sequences, encoded length)

For pad_middle, the encoded length is max_length. For left_pad_right_pad,

it's 3 * max_length.

variable_length_to_fixed_length_vector_encoding(*self*, *vector_encoding_name*, *alignment_method='pad_middle'*, *left_edge=4*, *right_edge=4*, *max_length=15*, *trim=False*, *allow_unsupported_amino_acids=False*)

Encode variable-length sequences to a fixed-size matrix. Amino acids are encoded as specified by the vector_encoding_name argument.

See *sequences_to_fixed_length_index_encoded_array* for details.

See also: variable_length_to_fixed_length_categorical.

Parameters

vector_encoding_name [string] How to represent amino acids. One of "BLOSUM62", "one-hot", etc. Full list of supported vector encodings is given by available_vector_encodings().

alignment_method [string] One of "pad_middle" or "left_pad_right_pad"

left_edge [int] Size of fixed-position left side. Only relevant for pad_middle alignment method

right_edge [int] Size of the fixed-position right side. Only relevant for pad_middle alignment method

max_length [int] Maximum supported peptide length

trim [bool] If True, longer sequences will be trimmed to fit the maximum supported length. Not supported for all alignment methods.

allow_unsupported_amino_acids [bool] If True, non-canonical amino acids will be replaced with the X character before encoding.

Returns

numpy.array with shape (num sequences, encoded length, m)

where

- m is the vector encoding length (usually 21).
- encoded length is max_length if alignment_method is pad_middle; 3 * max_length if it's left_pad_right_pad.

classmethod sequences_to_fixed_length_index_encoded_array(*sequences*, *alignment_method='pad_middle'*, *left_edge=4*, *right_edge=4*, *max_length=15*, *trim=False*, *allow_unsupported_amino_acids=False*)

Encode variable-length sequences to a fixed-size index-encoded (integer) matrix.

How variable length sequences get mapped to fixed length is set by the "alignment_method" argument. Supported alignment methods are:

pad_middle Encoding designed for preserving the anchor positions of class I peptides. This is what is used in allele-specific models.

Each string must be of length at least left_edge + right_edge and at most max_length. The first left_edge characters in the input always map to the first left_edge characters in the output. Similarly for the last right_edge characters. The middle characters are filled in based on the length, with the X character filling in the blanks.

Example:

AAAACDDDD -> AAAAXXXCXXXDDDD

left_pad_centered_right_pad Encoding that makes no assumptions on anchor positions but is 3x larger than pad_middle, since it duplicates the peptide (left aligned + centered + right aligned). This is what is used for the pan-allele models.

Example:

AAAACDDDD -> AAAACDDDDXXXXXXXXXAAAACDDDDXXXXXXXXXAAAACDDDD

left_pad_right_pad Same as left_pad_centered_right_pad but only includes left- and right-padded peptide.

Example:

AAAACDDDD -> AAAACDDDDXXXXXXXXXXXXAAAACDDDD

Parameters

sequences [list of string]

alignment_method [string] One of "pad_middle" or "left_pad_right_pad"

left_edge [int] Size of fixed-position left side. Only relevant for pad_middle alignment method

right_edge [int] Size of the fixed-position right side. Only relevant for pad_middle alignment method

max_length [int] maximum supported peptide length

trim [bool] If True, longer sequences will be trimmed to fit the maximum supported length. Not supported for all alignment methods.

allow_unsupported_amino_acids [bool] If True, non-canonical amino acids will be replaced with the X character before encoding.

Returns

numpy.array of integers with shape (num sequences, encoded length)

For pad_middle, the encoded length is max_length. For left_pad_right_pad,

it's 2 * max_length. For left_pad_centered_right_pad, it's

3 * max_length.

5.17 mhcflurry.ensemble_centrality module

Measures of centrality (e.g. mean) used to combine predictions across an ensemble. The input to these functions are log affinities, and they are expected to return a centrality measure also in log-space.

mhcflurry.ensemble_centrality.**robust_mean**(*log_values*)

Mean of values falling within the 25-75 percentiles.

Parameters

log_values [2-d numpy.array] Center is computed along the second axis (i.e. per row).

Returns

center [numpy.array of length log_values.shape[1]]

5.18 mhcflurry.fasta module

Adapted from pyensembl, github.com/openvax/pyensembl Original implementation by Alex Rubinsteyn.

The worse sin in bioinformatics is to write your own FASTA parser. We're doing this to avoid adding another dependency to MHCflurry, however.

mhcflurry.fasta.**read_fasta_to_dataframe**(*filename*)

class mhcflurry.fasta.**FastaParser**

Bases: object

FastaParser object consumes lines of a FASTA file incrementally.

iterate_over_file(*self*, *fasta_path*)

Generator that yields identifiers paired with sequences.

static open_file(*fasta_path*)

Open either a text file or compressed gzip file as a stream of bytes.

5.19 mhcflurry.flanking_encoding module

Class for encoding variable-length flanking and peptides to fixed-size numerical matrices

class mhcflurry.flanking_encoding.**EncodingResult**(*array*, *peptide_lengths*)

Bases: tuple

Create new instance of EncodingResult(array, peptide_lengths)

property array

Alias for field number 0

property peptide_lengths

Alias for field number 1

class mhcflurry.flanking_encoding.**FlankingEncoding**(*peptides*, *n_flanks*, *c_flanks*)

Bases: object

Encode peptides and optionally their N- and C-flanking sequences into fixed size numerical matrices. Similar to EncodableSequences but with support for flanking sequences and the encoding scheme used by the processing predictor.

Instances of this class have an immutable list of peptides with flanking sequences. Encodings are cached in the instances for faster performance when the same set of peptides needs to encoded more than once.

Constructor. Sequences of any lengths can be passed.

Parameters

peptides [list of string] Peptide sequences

n_flanks [list of string [same length as peptides]] Upstream sequences

c_flanks [list of string [same length as peptides]] Downstream sequences

unknown_character = 'X'

vector_encode(*self*, *vector_encoding_name*, *peptide_max_length*, *n_flank_length*, *c_flank_length*, *allow_unsupported_amino_acids=True*)

Encode variable-length sequences to a fixed-size matrix.

Parameters

vector_encoding_name [string] How to represent amino acids. One of "BLOSUM62", "one-hot", etc. See `amino_acid.available_vector_encodings()`.

peptide_max_length [int] Maximum supported peptide length.

n_flank_length [int] Maximum supported N-flank length

c_flank_length [int] Maximum supported C-flank length

allow_unsupported_amino_acids [bool] If True, non-canonical amino acids will be replaced with the X character before encoding.

Returns

numpy.array with shape (num sequences, length, m)

where

- num sequences is number of peptides, i.e. len(self)
- length is peptide_max_length + n_flank_length + c_flank_length
- m is the vector encoding length (usually 21).

static encode(*vector_encoding_name*, *df*, *peptide_max_length*, *n_flank_length*, *c_flank_length*, *allow_unsupported_amino_acids=False*)

Encode variable-length sequences to a fixed-size matrix.

Helper function. Users should use *vector_encode*.

Parameters

vector_encoding_name [string]

df [pandas.DataFrame]

peptide_max_length [int]

n_flank_length [int]

c_flank_length [int]

allow_unsupported_amino_acids [bool]

Returns

numpy.array

5.20 mhcflurry.hyperparameters module

Hyperparameter (neural network options) management

class mhcflurry.hyperparameters.**HyperparameterDefaults**(***defaults*)
Bases: object

Class for managing hyperparameters. Thin wrapper around a dict.

Instances of this class are a specification of the hyperparameters *supported* by a model and their defaults. The particular hyperparameter settings to be used, for example, to train a model are kept in plain dicts.

extend(*self*, *other*)
Return a new HyperparameterDefaults instance containing the hyperparameters from the current instance combined with those from other.

It is an error if self and other have any hyperparameters in common.

with_defaults(*self*, *obj*)
Given a dict of hyperparameter settings, return a dict containing those settings augmented by the defaults for any keys missing from the dict.

subselect(*self*, *obj*)
Filter a dict of hyperparameter settings to only those keys defined in this HyperparameterDefaults .

check_valid_keys(*self*, *obj*)
Given a dict of hyperparameter settings, throw an exception if any keys are not defined in this HyperparameterDefaults instance.

models_grid(*self*, ***kwargs*)
Make a grid of models by taking the cartesian product of all specified model parameter lists.

Parameters

The valid kwarg parameters are the entries of this

HyperparameterDefaults instance. Each parameter must be a list

giving the values to search across.

Returns

list of dict giving the parameters for each model. The length of the

list is the product of the lengths of the input lists.

5.21 mhcflurry.local_parallelism module

Infrastructure for "local" parallelism, i.e. multiprocess parallelism on one compute node.

mhcflurry.local_parallelism.**add_local_parallelism_args**(*parser*)
Add local parallelism arguments to the given argparse.ArgumentParser.

Parameters

parser [argparse.ArgumentParser]

mhcflurry.local_parallelism.**worker_pool_with_gpu_assignments_from_args**(*args*)
Create a multiprocessing.Pool where each worker uses its own GPU.

Uses commandline arguments. See *worker_pool_with_gpu_assignments*.

Parameters

args [argparse.ArgumentParser]

Returns

multiprocessing.Pool

mhcflurry.local_parallelism.**worker_pool_with_gpu_assignments**(*num_jobs*, *num_gpus=0*, *backend=None*, *max_workers_per_gpu=1*, *max_tasks_per_worker=None*, *worker_log_dir=None*)

Create a multiprocessing.Pool where each worker uses its own GPU.

Parameters

num_jobs [int] Number of worker processes.

num_gpus [int]

backend [string]

max_workers_per_gpu [int]

max_tasks_per_worker [int]

worker_log_dir [string]

Returns

multiprocessing.Pool

mhcflurry.local_parallelism.**make_worker_pool**(*processes=None*, *initializer=None*, *initializer_kwargs_per_process=None*, *max_tasks_per_worker=None*)

Convenience wrapper to create a multiprocessing.Pool.

This function adds support for per-worker initializer arguments, which are not natively supported by the multiprocessing module. The motivation for this feature is to support allocating each worker to a (different) GPU.

IMPLEMENTATION NOTE: The per-worker initializer arguments are implemented using a Queue. Each worker reads its arguments from this queue when it starts. When it terminates, it adds its initializer arguments back to the queue, so a future process can initialize itself using these arguments.

There is one issue with this approach, however. If a worker crashes, it never repopulates the queue of initializer arguments. This will prevent any future worker from re-using those arguments. To deal with this issue we add a second 'backup queue'. This queue always contains the full set of initializer arguments: whenever a worker reads from it, it always pushes the pop'd args back to the end of the queue immediately. If the primary arg queue is ever empty, then workers will read from this backup queue.

Parameters

processes [int] Number of workers. Default: num CPUs.

initializer [function, optional] Init function to call in each worker

initializer_kwargs_per_process [list of dict, optional] Arguments to pass to initializer function for each worker. Length of list must equal the number of workers.

max_tasks_per_worker [int, optional] Restart workers after this many tasks. Requires Python >=3.2.

Returns

multiprocessing.Pool

mhcflurry.local_parallelism.**worker_init_entry_point**(*init_function*, *arg_queue=None*, *backup_arg_queue=None*)

mhcflurry.local_parallelism.**worker_init**(*keras_backend=None*, *gpu_device_nums=None*, *worker_log_dir=None*)

exception mhcflurry.local_parallelism.**WrapException**
Bases: Exception

Add traceback info to exception so exceptions raised in worker processes can still show traceback info when re-raised in the parent.

mhcflurry.local_parallelism.**call_wrapped**(*function*, **args*, ***kwargs*)
Run function on args and kwargs and return result, wrapping any exception raised in a WrapException.

Parameters

function [arbitrary function]

Any other arguments provided are passed to the function.

Returns

object

mhcflurry.local_parallelism.**call_wrapped_kwargs**(*function*, *kwargs*)
Invoke function on given kwargs and return result, wrapping any exception raised in a WrapException.

Parameters

function [arbitrary function]

kwargs [dict]

Returns

object

result of calling function(kwargs)**

5.22 mhcflurry.percent_rank_transform module

Class for transforming arbitrary values into percent ranks given a distribution.

class mhcflurry.percent_rank_transform.**PercentRankTransform**
Bases: object

Transform arbitrary values into percent ranks.

fit(*self*, *values*, *bins*)
Fit the transform using the given values (in our case ic50s).

Parameters

values [ic50 values]

bins [bins for the cumulative distribution function] Anything that can be passed to numpy.histogram's "bins" argument can be used here.

transform(*self*, *values*)
Return percent ranks (range [0, 100]) for the given values.

to_series(*self*)
Serialize the fit to a pandas.Series.

The index on the series gives the bin edges and the valeus give the CDF.

Returns

pandas.Series

static from_series(*series*)
Deseralize a PercentRankTransform the given pandas.Series, as returned by *to_series()*.

Parameters

series [pandas.Series]

Returns

PercentRankTransform

5.23 mhcflurry.predict_command module

Run MHCflurry predictor on specified peptides.

By default, the presentation predictor is used, and predictions for MHC I binding affinity, antigen processing, and the composite presentation score are returned. If you just want binding affinity predictions, pass –affinity-only.

Examples:

Write a CSV file containing the contents of INPUT.csv plus additional columns giving MHCflurry predictions:

$ mhcflurry-predict INPUT.csv –out RESULT.csv

The input CSV file is expected to contain columns “allele”, “peptide”, and, optionally, “n_flank”, and “c_flank”.

If `--out` is not specified, results are written to stdout.

You can also run on alleles and peptides specified on the commandline, in which case predictions are written for *all combinations* of alleles and peptides:

$ mhcflurry-predict –alleles HLA-A0201 H-2Kb –peptides SIINFEKL DENDREKLLL

Instead of individual alleles (in a CSV or on the command line), you can also give a comma separated list of alleles giving a sample genotype. In this case, the tightest binding affinity across the alleles for the sample will be returned. For example:

$ mhcflurry-predict –peptides SIINFEKL DENDREKLLL –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:01,HLA-C*07:02 HLA-A*01:01,HLA-A*02:06,HLA-B*44:02,HLA-B*07:02,HLA-C*01:01,HLA-C*03:01

will give the tightest predicted affinities across alleles for each of the two genotypes specified for each peptide.

mhcflurry.predict_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

5.24 mhcflurry.predict_scan_command module

Scan protein sequences using the MHCflurry presentation predictor.

By default, sub-sequences (peptides) with affinity percentile ranks less than 2.0 are returned. You can also specify –results-all to return predictions for all peptides, or –results-best to return the top peptide for each sequence.

Examples:

Scan a set of sequences in a FASTA file for binders to any alleles in a MHC I genotype:

$ mhcflurry-predict-scan test/data/example.fasta –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:01,HLA-C*07:02

Instead of a FASTA, you can also pass a CSV that has "sequence_id" and "sequence" columns.

You can also specify multiple MHC I genotypes to scan as space-separated arguments to the –alleles option:

$ mhcflurry-predict-scan test/data/example.fasta –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:02,HLA-C*07:02 HLA-A*01:01,HLA-A*02:06,HLA-B*44:02,HLA-B*07:02,HLA-C*01:02,HLA-C*03:01

If `--out` is not specified, results are written to standard out.

You can also specify sequences on the commandline:

mhcflurry-predict-scan –sequences MGYINVFAFPFTIYSLLLCRMNSRNYIAQVDVVNFNLT –alleles HLA-A*02:01,HLA-A*03:01,HLA-B*57:01,HLA-B*45:01,HLA-C*02:02,HLA-C*07:02

`mhcflurry.predict_scan_command.`**`run`**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

5.25 mhcflurry.random_negative_peptides module

class `mhcflurry.random_negative_peptides.`**`RandomNegativePeptides`**(***hyperparameters*)

Bases: `object`

Generate random negative (peptide, allele) pairs. These are used during model training, where they are resampled at each epoch.

`hyperparameter_defaults = <mhcflurry.hyperparameters.HyperparameterDefaults object>`

Hyperperameters for random negative peptides.

Number of random negatives will be: random_negative_rate * (num measurements) + random_negative_constant

where the exact meaning of (num measurements) depends on the particular random_negative_method in use.

If random_negative_match_distribution is True, then the amino acid frequencies of the training data peptides are used to generate the random peptides.

Valid values for random_negative_method are:

"by_length": used for allele-specific prediction. See description in *RandomNegativePeptides.plan_by_length* method.

"by_allele": used for pan-allele prediction. See *RandomNegativePeptides.plan_by_allele* method.

"by_allele_equalize_nonbinders": used for pan-allele prediction. See *RandomNegativePeptides.plan_by_allele_equalize_nonbinders* method.

"recommended": the default. Use by_length if the predictor is allele- specific and by_allele if it's pan-allele.

plan (*self*, *peptides*, *affinities*, *alleles=None*, *inequalities=None*)
Calculate the number of random negatives for each allele and peptide length. Call this once after instantiating the object.

Parameters

peptides [list of string]

affinities [list of float]

alleles [list of string, optional]

inequalities [list of string (">", "<", or "="), optional]

Returns

pandas.DataFrame indicating number of random negatives for each length

and allele.

plan_by_length (*self*, *df_all*, *df_binders=None*, *df_nonbinders=None*)
Generate a random negative plan using the "by_length" policy.

Parameters are as in the *plan* method. No return value.

Used for allele-specific predictors. Does not work well for pan-allele.

Different numbers of random negatives per length. Alleles are sampled proportionally to the number of times they are used in the training data.

plan_by_allele (*self*, *df_all*, *df_binders=None*, *df_nonbinders=None*)
Generate a random negative plan using the "by_allele" policy.

Parameters are as in the *plan* method. No return value.

For each allele, a particular number of random negatives are used for all lengths. Across alleles, the number of random negatives varies; within an allele, the number of random negatives for each length is a constant

plan_by_allele_equalize_nonbinders (*self*, *df_all*, *df_binders*, *df_nonbinders*)
Generate a random negative plan using the "by_allele_equalize_nonbinders" policy.

Parameters are as in the *plan* method. No return value.

Requires that the random_negative_binder_threshold hyperparameter is set.

In a first step, the number of random negatives selected by the "by_allele" method are added (see *plan_by_allele*). Then, the total number of non-binders are calculated for each allele and length. This total includes non-binder measurements in the training data plus the random negative peptides added in the first step. In a second step, additional random negative peptides are added so that for each allele, all peptide lengths have the same total number of non-binders.

get_alleles (*self*)
Get the list of alleles corresponding to each random negative peptide as returned by *get_peptides*. This does NOT change and can be safely called once and reused.

Returns

list of string

get_peptides (*self*)
Get the list of random negative peptides. This will be different each time the method is called.

Returns

list of string

`get_total_count` (*self*)

Total number of planned random negative peptides.

Returns

int

5.26 mhcflurry.regression_target module

`mhcflurry.regression_target.from_ic50` (*ic50*, *max_ic50=50000.0*)

Convert ic50s to regression targets in the range [0.0, 1.0].

Parameters

ic50 [numpy.array of float]

Returns

numpy.array of float

`mhcflurry.regression_target.to_ic50` (*x*, *max_ic50=50000.0*)

Convert regression targets in the range [0.0, 1.0] to ic50s in the range [0, 50000.0].

Parameters

x [numpy.array of float]

Returns

numpy.array of float

5.27 mhcflurry.scoring module

Measures of prediction accuracy

`mhcflurry.scoring.make_scores` (*ic50_y*, *ic50_y_pred*, *sample_weight=None*, *threshold_nm=500*, *max_ic50=50000*)

Calculate AUC, F1, and Kendall Tau scores.

Parameters

ic50_y [float list] true IC50s (i.e. affinities)

ic50_y_pred [float list] predicted IC50s

sample_weight [float list [optional]]

threshold_nm [float [optional]]

max_ic50 [float [optional]]

Returns

dict with entries "auc", "f1", "tau"

5.28 mhcflurry.select_allele_specific_models_command module

Model select class1 single allele models.

mhcflurry.select_allele_specific_models_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

class mhcflurry.select_allele_specific_models_command.**ScrambledPredictor**(*predictor*)

Bases: object

predict(*self*, *peptides*, *allele*)

mhcflurry.select_allele_specific_models_command.**model_select**(*allele*)

mhcflurry.select_allele_specific_models_command.**cache_encoding**(*predictor*, *peptides*)

class mhcflurry.select_allele_specific_models_command.**ScoreFunction**(*function*, *summary=None*)

Bases: object

Thin wrapper over a score function (Class1AffinityPredictor -> float). Used to keep a summary string associated with the function.

class mhcflurry.select_allele_specific_models_command.**CombinedModelSelector**(*model_selectors*, *weights=None*, *min_contribution_per*

Bases: object

Model selector that computes a weighted average over other model selectors.

usable_for_allele(*self*, *allele*)

plan_summary(*self*, *allele*)

score_function(*self*, *allele*, *dry_run=False*)

class mhcflurry.select_allele_specific_models_command.**ConsensusModelSelector**(*predictor*, *num_peptides_per_* *multiply_score_by_value*

Bases: object

Model selector that scores sub-ensembles based on their Kendall tau consistency with the full ensemble over a set of random peptides.

usable_for_allele(*self*, *allele*)

max_absolute_value(*self*, *allele*)

plan_summary(*self*, *allele*)

score_function(*self*, *allele*)

class mhcflurry.select_allele_specific_models_command.**MSEModelSelector**(*df*, *predictor*, *min_measurements=1*, *multiply_score_by_data_size=Tru*

Bases: object

Model selector that uses mean-squared error to score models. Inequalities are supported.

usable_for_allele(*self*, *allele*)

max_absolute_value(*self*, *allele*)

plan_summary(*self*, *allele*)

score_function(*self*, *allele*)

class mhcflurry.select_allele_specific_models_command.**MassSpecModelSelector**(*df*, *predictor*, *decoys_per_length=0*, *min_measurements=1*, *multiply_score_by_data_s*

Bases: object

Model selector that uses PPV of differentiating decoys from hits from mass-spec experiments.

static ppv(*y_true*, *predictions*)

usable_for_allele(*self*, *allele*)

max_absolute_value(*self*, *allele*)

plan_summary(*self*, *allele*)

score_function(*self*, *allele*)

5.29 mhcflurry.select_pan_allele_models_command module

Model select class1 pan-allele models.

APPROACH: For each training fold, we select at least min and at most max models (where min and max are set by the –{min/max}-models-per-fold argument) using a step-up (forward) selection procedure. The final ensemble is the union of all selected models across all folds.

mhcflurry.select_pan_allele_models_command.**mse**(*predictions*, *actual*, *inequalities=None*, *affinities_are_already_01_transformed=False*)

Mean squared error of predictions vs. actual

Parameters

predictions [list of float]

actual [list of float]

inequalities [list of string (“>”, “<”, or “=”)]

affinities_are_already_01_transformed [boolean] Predictions and actual are taken to be nanomolar affinities if affinities_are_already_01_transformed is False, otherwise 0-1 values.

Returns

float

mhcflurry.select_pan_allele_models_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.select_pan_allele_models_command.**do_model_select_task**(*item*, *constant_data={}*)

mhcflurry.select_pan_allele_models_command.**model_select**(*fold_num*, *models*, *min_models*, *max_models*, *constant_data={}*)

Model select for a fold.

Parameters

fold_num [int]

models [list of Class1NeuralNetwork]

min_models [int]

max_models [int]

constant_data [dict]

Returns

dict with keys ‘fold_num’, ‘selected_indices’, ‘summary’

5.30 mhcflurry.select_processing_models_command module

Model select antigen processing models.

APPROACH: For each training fold, we select at least min and at most max models (where min and max are set by the –{min/max}-models-per-fold argument) using a step-up (forward) selection procedure. The final ensemble is the union of all selected models across all folds. AUC is used as the metric.

mhcflurry.select_processing_models_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.select_processing_models_command.**do_model_select_task**(*item*, *constant_data={}*)

mhcflurry.select_processing_models_command.**model_select**(*fold_num*, *models*, *min_models*, *max_models*, *constant_data={}*)

Model select for a fold.

Parameters

fold_num [int]

models [list of Class1NeuralNetwork]

min_models [int]

max_models [int]

constant_data [dict]

Returns

dict with keys 'fold_num', 'selected_indices', 'summary'

5.31 mhcflurry.testing_utils module

Utilities used in MHCflurry unit tests.

mhcflurry.testing_utils.**startup**()
Configure Keras backend for running unit tests.

mhcflurry.testing_utils.**cleanup**()
Clear tensorflow session and other process-wide resources.

5.32 mhcflurry.train_allele_specific_models_command module

Train Class1 single allele models.

mhcflurry.train_allele_specific_models_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.train_allele_specific_models_command.**alleles_by_similarity**(*allele*)

mhcflurry.train_allele_specific_models_command.**train_model**(*n_models*, *allele_num*, *n_alleles*, *hyperparameter_set_num*, *num_hyperparameter_sets*, *allele*, *hyperparameters*, *verbose*, *progress_print_interval*, *predictor*, *save_to*)

mhcflurry.train_allele_specific_models_command.**subselect_df_held_out**(*df*, *reciprocal_held_out_fraction=10*, *seed=0*)

5.33 mhcflurry.train_pan_allele_models_command module

Train Class1 pan-allele models.

mhcflurry.train_pan_allele_models_command.**assign_folds**(*df*, *num_folds*, *held_out_fraction*, *held_out_max*)
Split training data into multple test/train pairs, which we refer to as folds. Note that a given data point may be assigned to multiple test or train sets; these folds are NOT a non-overlapping partition as used in cross validation.

A fold is defined by a boolean value for each data point, indicating whether it is included in the training data for that fold. If it's not in the training data, then it's in the test data.

Folds are balanced in terms of allele content.

Parameters

df [pandas.DataFrame] training data

num_folds [int]

held_out_fraction [float] Fraction of data to hold out as test data in each fold

held_out_max For a given allele, do not hold out more than held_out_max number of data points in any fold.

Returns

pandas.DataFrame index is same as df.index, columns are "fold_0", ... "fold_N" giving whether the data point is in the training data for the fold

mhcflurry.train_pan_allele_models_command.**pretrain_data_iterator**(*filename*, *master_allele_encoding*, *peptides_per_chunk=1024*)

Step through a CSV file giving predictions for a large number of peptides (rows) and alleles (columns).

Parameters

filename [string]

master_allele_encoding [AlleleEncoding]

peptides_per_chunk [int]

Returns

Generator of (AlleleEncoding, EncodableSequences, float affinities) tuples

mhcflurry.train_pan_allele_models_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.train_pan_allele_models_command.**main**(*args*)

mhcflurry.train_pan_allele_models_command.**initialize_training**(*args*)

mhcflurry.train_pan_allele_models_command.**train_models**(*args*)

mhcflurry.train_pan_allele_models_command.**train_model**(*work_item_name*, *work_item_num*, *num_work_items*, *architecture_num*, *num_architectures*, *fold_num*, *num_folds*, *replicate_num*, *num_replicates*, *hyperparameters*, *pretrain_data_filename*, *verbose*, *progress_print_interval*, *predictor*, *save_to*, *constant_data={}*)

5.34 mhcflurry.train_presentation_models_command module

Train Class1 presentation models.

mhcflurry.train_presentation_models_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.train_presentation_models_command.**main**(*args*)

5.35 mhcflurry.train_processing_models_command module

Train Class1 processing models.

mhcflurry.train_processing_models_command.**assign_folds**(*df*, *num_folds*, *held_out_samples*)

Split training data into mulitple test/train pairs, which we refer to as folds. Note that a given data point may be assigned to multiple test or train sets; these folds are NOT a non-overlapping partition as used in cross validation.

A fold is defined by a boolean value for each data point, indicating whether it is included in the training data for that fold. If it's not in the training data, then it's in the test data.

Parameters

df [pandas.DataFrame] training data

num_folds [int]

held_out_samples [int]

Returns

pandas.DataFrame index is same as df.index, columns are "fold_0", ... "fold_N" giving whether the data point is in the training data for the fold

mhcflurry.train_processing_models_command.**run**(*argv=['-b', 'latex', '-v', '-d', '_build/doctrees', '.', '_build/latex']*)

mhcflurry.train_processing_models_command.**main**(*args*)

mhcflurry.train_processing_models_command.**initialize_training**(*args*)

mhcflurry.train_processing_models_command.**train_models**(*args*)

mhcflurry.train_processing_models_command.**train_model**(*work_item_name*, *work_item_num*, *num_work_items*, *architecture_num*, *num_architectures*, *fold_num*, *num_folds*, *replicate_num*, *num_replicates*, *hyperparameters*, *verbose*, *progress_print_interval*, *predictor*, *save_to*, *constant_data={}*)

5.36 mhcflurry.version module

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y