AAanalysis

v1.1 Cheat Sheet

Sequence → physicochemical scales → interpretable features → explainable ML → biological mechanism

pip install aaanalysis · aaanalysis.readthedocs.io

> **AAanalysis** is a Python framework for interpretable, sequence-based protein prediction. It turns sequences into physicochemical features (CPP), trains explainable models, and traces every prediction back to a residue × property × group comparison — robust for small datasets.

Install · Import · Load *core + [pro]*

```
# Python >= 3.11
pip install aaanalysis          # core
pip install 'aaanalysis[pro]'  # SHAP, FIMO, Bio

import numpy as np
import matplotlib.pyplot as plt
import aaanalysis as aa
df_seq = aa.load_dataset(name='DOM_GSEC')   # γ-secretase
labels = df_seq['label'].to_list()
df_scales = aa.load_scales()
```

The Golden Workflow *canonical pipeline*

1. **LOAD** `load_dataset · load_scales → df_seq · df_scales`
2. **PARTS** `SequenceFeature.get_df_parts → df_parts`
3. **FEATURES** `Part × Split × Scale · CPP.run → df_feat`
4. **MODEL** `TreeModel.fit · dPULearn.fit → feat_importance · labels_`
5. **EXPLAIN** `CPPPlot.feature_map · ShapModel → figure · feat_impact`

Prediction Task Levels *task → setup*

Residue `AA_*` — positional
unit: sliding window (aa_window_size)
ref: non-site windows / shuffled background

Domain `DOM_*` — both
unit: part-set jmd_n · tmd · jmd_c (from tmd_start / tmd_stop)
ref: labelled A vs B groups

Protein `SEQ_*` — compositional
unit: whole chain (composition)
ref: labelled groups / composition-matched background

Sequence Anatomy *the TMD model*

TMD Target Middle Domain — the central segment of interest (e.g. transmembrane stretch, binding region).

JMD Juxta Middle Domain — the flanks adjoining the TMD (jmd_n on the N-side, jmd_c on the C-side).

JMD widths set globally: aa.options['jmd_n_len'] · ['jmd_c_len'].

1: positives · 0: rel-negatives · 2: unlabeled

Feature Ontology *Part × Split × Scale*

PART *where on the sequence*
tmd · jmd_n · jmd_c · tmd_jmd · jmd_n_tmd_n · tmd_c_jmd_c

SPLIT *how to read the part*
Segment — contiguous · Pattern — sparse pairs · PeriodicPattern — i, i+3/4

SCALE *which physicochemical property*
AAontology (~600 scales) · hydrophobicity · charge · helix propensity · shape

```
TMD × Segment × hydrophobicity → membrane insertion
JMD × Pattern × net charge → electrostatic recognition
TMD × PeriodicPattern × helix → α-helical interface
```

CPP Strategies *via split_kws*

Compositional ≈ sequence/protein-level

one whole-part average (composition-like, position-agnostic)

```
split_kws = sf.get_split_kws(
    split_types="Segment",
    n_split_max=1)
cpp = aa.CPP(df_parts=df_parts, split_kws=split_kws)
```

Positional ≈ residue-/region-level

sub-segments and/or patterns resolved to positions

```
split_kws = sf.get_split_kws(
    split_types=["Segment", "Pattern", "PeriodicPattern"],
    n_split_max=5,
    steps_pattern=[3, 4],
    steps_periodicpattern=[3, 4])
cpp = aa.CPP(df_parts=df_parts, split_kws=split_kws)
```

Domain level uses both. → CPP strategies: see the CPP tutorial (docs).

Which Module Should I Use? *intent → module*

Explore sequence patterns / composition →
`AAlogo · AAlogoPlot`

Discover discriminative physicochemical features →
`CPP · CPPPlot`

Reduce redundant amino acid scales →
`AAclust`

Train with positives + unlabeled data →
`dPULearn`

Train an interpretable classifier →
`TreeModel`

Explain a prediction (per feature / sample) →
`ShapModel [pro]`

Data & Preparation *load · encode · clean*

Load benchmark sequences
`load_dataset(name) → df_seq`

Load AAontology scales
`load_scales() → df_scales`

Load precomputed features
`load_features(name) → df_feat`

Read / write FASTA
`read_fasta(file) → df_seq`

Encode sequences (one-hot / int)
`SequencePreprocessor().encode_*(seqs)`

Cluster redundant homologs
`filter_seq(df_seq) [pro]`

Feature Engineering *parts · CPP · scales*

SequenceFeature → sf
`sf = aa.SequenceFeature()`

· split sequence into parts
`sf.get_df_parts(df_seq) → df_parts`

· assemble feature matrix X
`sf.feature_matrix(df_feat, df_parts) → X`

Discover discriminative features
`CPP(df_parts).run(labels) → df_feat ⋆`

Sweep CPP configs (grid)
`CPPGrid().run(...) · .eval() → ranked configs`

Simplify → interpretable scales
`CPP.simplify(df_feat, labels)`

Reduce redundant scales
`AAclust().fit(X) [Wrapper]`

Drop correlated features
`NumericalFeature().filter_correlation(X)`

Numerical Feature Sources v1.1 *PLM · structure · PTM*

PLM embeddings
`EmbeddingPreprocessor().encode(...) → dict_num`

Structure / DSSP / PAE
`StructurePreprocessor().encode_dssp(...) [pro]`

PTM / site annotations
`AnnotationPreprocessor().encode(...) [pro]`

Combine sources
`combine_dict_nums([...]) → dict_num`

Slice to parts
`NumericalFeature().get_parts(...) → dict_num_parts`

Numerical CPP
`CPP(df_parts).run_num(dict_num_parts, labels) → df_feat`

Modeling & Explainability

Train + RFE + MC importance
`TreeModel().fit(X, labels) [Wrapper]`

Train with positives + unlabeled
`dPULearn().fit(X, labels) [Wrapper]`

Per-feature / sample SHAP impact
`ShapModel().fit(X, labels) [pro]`

Sequence Analysis *logos · motifs*

Position-specific logo
`AAlogo().get_df_logo(df_parts)`

Sample sequence windows
`AAWindowSampler().sample(df_seq)`

Pairwise sequence similarity
`comp_seq_sim(df_seq) [pro]`

Scan motifs (FIMO / MEME)
`scan_motif(df_seq, pwm) [pro]`

Metrics & Plotting *utilities*

Adjusted AUC (class imbalance)
`comp_auc_adjusted(X, labels)`

BIC score · KL divergence
`comp_bic_score(X, labels) · comp_kld`

Per-protein / detection (v1.1)
`comp_per_protein_ap · comp_detection_metrics`

Global plot style & fonts
`plot_settings(font_scale)`

Colours & standalone legend
`plot_get_clist(n) · plot_legend(ax)`

Protein Design (to be extended) v1.1

In-silico point mutations
`AAMut · AAMutPlot`

Sequence-design libraries
`SeqMut · SeqMutPlot`

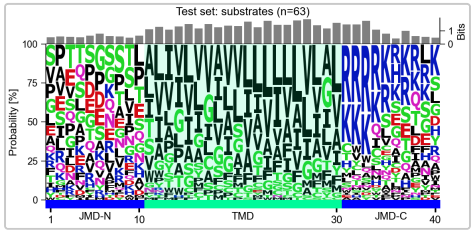
AAanalysis · Worked Examples

tutorial-style code with the figure each call produces

AAlogo — see the data

dataset at a glance

```
import numpy as np, matplotlib.pyplot as plt, aaanalysis as aa
df_seq = aa.load_dataset(name='DOM_GSEC')   # γ-secretase
labels = list(df_seq['label']); df_scales = aa.load_scales()
sf = aa.SequenceFeature()
df_parts = sf.get_df_parts(df_seq=df_seq,
    list_parts=['tmd', 'jmd_n', 'jmd_c'])
aa.plot_settings(font_scale=0.7)
# aal_kws builds df_logo + bits bar for you
aa.AAlogoPlot().single_logo(
    aal_kws=dict(df_parts=df_parts, labels=labels,
        label_test=1, tmd_len=20),
    name_data='Test set: substrates')
plt.tight_layout(); plt.show()
```



AAlogoPlot.single_logo · per-position enrichment

CPP — feature

top feature · REF vs TEST

```
# default parts + a redundancy-reduced set of 100 scales
df_parts = sf.get_df_parts(df_seq=df_seq)
df_scales_sel = aa.AAclust().select_scales(
    df_scales=df_scales, n_clusters=100)
cpp = aa.CPP(df_parts=df_parts, df_scales=df_scales_sel)
df_feat = cpp.run(labels=labels, n_filter=100)
X = sf.feature_matrix(df_feat['feature'], df_parts)
tm = aa.TreeModel(); tm.fit(X, labels=labels)
df_feat = tm.add_feat_importance(df_feat=df_feat, sort=True)
cpp_plot = aa.CPPPlot(); aa.plot_settings()
# distribution of the top feature (feat_rank=1 of the sorted
df_feat)
cpp_plot.feature(feature=df_feat, feat_rank=1, df_seq=df_seq,
    labels=labels, name_test='substrates', name_ref='non-subs.')
plt.tight_layout(); plt.show()
```

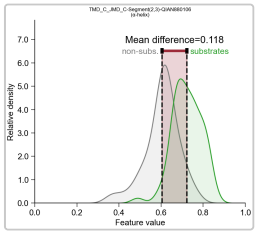
▼ *figure below · left*

CPP — ranking

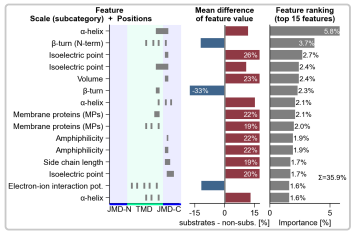
top features · effect + importance

```
# same df_feat — rank the top discriminative features
aa.plot_settings(font_scale=0.6)
cpp_plot.ranking(df_feat=df_feat, n_top=15, rank=True,
    name_test='substrates', name_ref='non-subs.')
plt.tight_layout(); plt.show()
```

▼ *figure below · right*



CPPPlot.feature · top feature, REF vs TEST

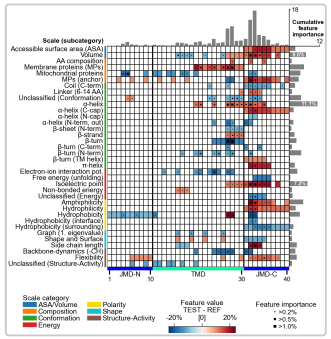


CPPPlot.ranking · top-15 features

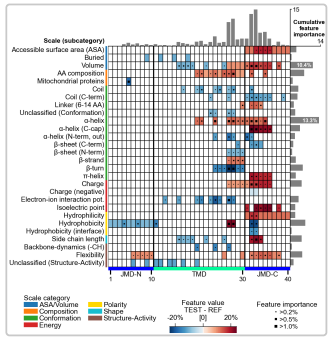
CPP — feature map

group level · full vs simplified

```
# global Part × Split × Scale map — all AAontology scales
cpp_plot = aa.CPPPlot(); aa.plot_settings(font_scale=0.65)
cpp_plot.feature_map(df_feat=df_feat)          # left
# CPP.simplify → fewer, interpretable correlated scales
df_feat = cpp.simplify(df_feat=df_feat, labels=labels)
cpp_plot.feature_map(df_feat=df_feat)          # right
plt.tight_layout(); plt.show()
```



CPPPlot.feature_map · all scales

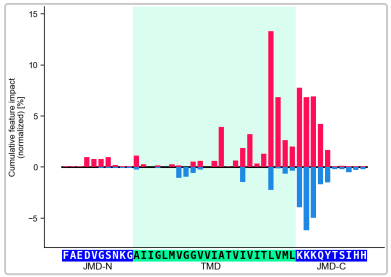


CPPPlot.feature_map · simplified

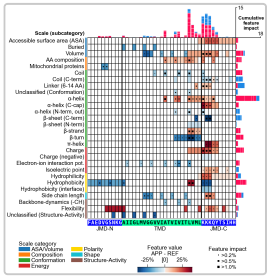
ShapModel — explain a prediction

sample level · [pro]

```
# fuzzy labeling: APP's label is its soft prediction score (0.6,
not 1)
i = list(df_seq['entry']).index('P05067')   # APP
y = [float(v) for v in labels]; y[i] = 0.6
sm = aa.ShapModel(); sm.fit(X, labels=y, fuzzy_labeling=True)
df_feat = sm.add_feat_impact(df_feat=df_feat,
            sample_positions=i, names='APP')
args_seq = {k + '_seq': v for k, v in sf.get_df_parts(
    df_seq=df_seq).loc['P05067'].to_dict().items()}
ka = dict(col_imp='feat_impact_APP', shap_plot=True, **args_seq)
cpp_plot.profile(df_feat=df_feat, **ka)          # left
cpp_plot.feature_map(df_feat=df_feat, name_test='APP', **ka)  #
right
plt.tight_layout(); plt.show()
```



CPPPlot.profile · SHAP



CPPPlot.feature_map · SHAP

AAclust — clusters

scale reduction · Wrapper

```
X = np.array(df_scales).T
aac = aa.AAclust()
aac.fit(X, names=list(df_scales), n_clusters=10)
aac.medoid_names_   # redundancy-reduced scales

aac_plot = aa.AAclustPlot()
aac_plot.centers(X, labels=aac.labels_)
plt.tight_layout(); plt.show()
```

▼ *figure below · left*

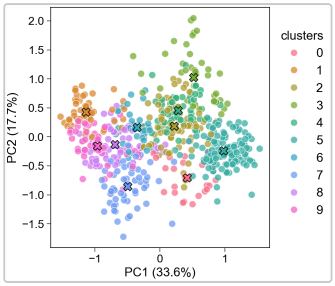
dPULearn — PCA

reliable negatives · Wrapper

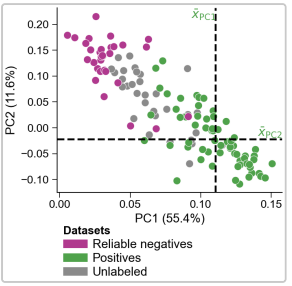
```
# labels: 1 = positive, 2 = unlabeled
n_pos = sum(np.array(labels) == 1)
dpul = aa.dPULearn()
dpul.fit(X=X, labels=labels, n_unl_to_neg=n_pos)
df_pu = dpul.df_pu_   # 1 pos · 0 rel-neg · 2 unl

dpul_plot = aa.dPULearnPlot()
dpul_plot.pca(df_pu=df_pu, labels=labels)
plt.tight_layout(); plt.show()
```

▼ *figure below · right*



AAclustPlot.centers · cluster scale profiles



dPULearnPlot.pca · reliable negatives

AAanalysis · Reference

5-minute example · options · class ↔ plot · glossary · citations

The 5-Minute Example

copy-paste this

```
import matplotlib.pyplot as plt
import aaanalysis as aa
df_seq = aa.load_dataset(name='DOM_GSEC')
labels = df_seq['label'].to_list()
sf = aa.SequenceFeature()
df_parts = sf.get_df_parts(df_seq=df_seq)
cpp = aa.CPP(df_parts=df_parts)
df_feat = cpp.run(labels=labels)
X = sf.feature_matrix(features=df_feat['feature'],
                      df_parts=df_parts)
tm = aa.TreeModel(); tm.fit(X, labels=labels)
df_feat = tm.add_feat_importance(df_feat=df_feat)
cpp_plot = aa.CPPPlot()
aa.plot_settings()
cpp_plot.feature_map(df_feat=df_feat)
plt.tight_layout(); plt.show()
```

aa.options

system-level settings

```
aa.options['random_state'] = 42
aa.options['verbose'] = True
aa.options['allow_multiprocessing'] = True

# TMD model — JMD flank widths
aa.options['jmd_n_len'] = 10
aa.options['jmd_c_len'] = 10

# plot labels & system-level scales
aa.options['name_tmd'] = 'P5-P5´'   # e.g. cleavage-site
prediction
aa.options['df_scales'] = my_scales
```

Class ↔ Plot Class

mirrored API

SequencePreprocessor	–	
EmbeddingPreprocessor	–	v1.1
StructurePreprocessor [pro]	–	v1.1
AnnotationPreprocessor [pro]	–	v1.1
CPP	CPPPlot	
AAclust	AAclustPlot	Wrapper
AAlogo	AAlogoPlot	
dPULearn	dPULearnPlot	Wrapper
TreeModel	–	Wrapper
ShapModel [pro]	–	Wrapper
AAMut	AAMutPlot	to be extended
SeqMut	SeqMutPlot	to be extended
AAWindowSampler	–	
SequenceFeature	–	
NumericalFeature	–	

How to Cite

if you use AAanalysis

AAclust [Breimann24a]

Breimann & Frishman (2024a), AAclust: k-optimized clustering for selecting redundancy-reduced sets of amino acid scales
Bioinformatics Advances

AAontology [Breimann24b]

Breimann et al. (2024b), AAontology: An ontology of amino acid scales for interpretable machine learning
Journal of Molecular Biology

CPP & dPULearn [Breimann25a]

Breimann & Kamp et al. (2025), Charting γ-secretase substrates by explainable AI
Nature Communications

Design Principles

- Explicit over implicit — DataFrames everywhere
- Wrappers (.fit / .predict / .eval) set trailing *_ attributes after fit
- Biological interpretability is first-class
- Small-data robust and reproducible (layered seeds)

Glossary

canonical vocabulary · CONTEXT.md

Prediction level — Residue (AA_*) · Domain (DOM_*) · Protein (SEQ_*) — the unit a task predicts at; sets the dataset, the part, and the reference.

Part — Named segment used as feature input: tmd, jmd_n, jmd_c, tmd_jmd, jmd_n_tmd_n, tmd_c_jmd_c.

Split — How a scale is read across a part: Segment (contiguous), Pattern (sparse), PeriodicPattern (i, i+3/4).

Scale — AA → ℝ mapping. AAontology ships ~600 curated scales in two-level categories.

Feature — (Part × Split × Scale) — the atomic, residue-grounded, interpretable unit of CPP.

Compositional vs positional — How split_kws resolves locality: a whole-part average (compositional) vs sub-region/position-resolved (positional).

df_seq — Sequence table: entry, sequence, label, TMD bounds (tmd_start, tmd_stop).

df_parts — Wide table — one column per part (tmd, jmd_n, jmd_c, …).

df_feat — Ranked features: feature, abs_auc, mean_dif, p_val, positions, scale, category.

Wrapper — sklearn-style class — .fit / .predict / .eval, sets trailing *_ attributes after fit.

Plot class — *Plot mirror of an analytical class — same arguments, visualization only.

PU labels — dPULearn input: 1 = positive, 2 = unlabeled. Output: 1 / 0 (reliable-negative) / 2.

CPP — Comparative Physicochemical Profiling — discovers ranked Part × Split × Scale features.

AAontology — Two-level scale taxonomy; CPP uses its categories to organize and rank features.