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. **v1.1** extends the core feature engine beyond physicochemical scales to PLM embeddings and protein structure.

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

Output types: *classification · regression · ranking · explanation*

Groups: 1: positives · 0: negatives · 0: reliable negatives · 2: unlabeled

*Label 0 is a curated **negative** in labelled data, or a dPULearn-inferred **reliable negative** drawn from the unlabeled (2) pool.*

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).

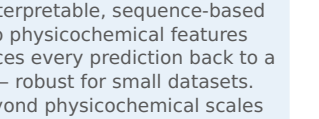
JMD widths set globally: aa.options['jmd_n_len'] · ['jmd_c_len'].

CPP Feature Concept *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



Simplified from Breimann25a (Suppl. Fig. 1C ↗)

```
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`

Sample reference windows (if negatives are missing)
`AAWindowSampler`

Reduce redundant amino acid scales
`AAclust`

Discover discriminative physicochemical features
`CPP`

Train with positives + unlabeled data
`dPULearn`

Train an interpretable classifier
`TreeModel`

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

Data & Preparation *datasets · scales · FASTA*

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`**

Cluster redundant homologs
`filter_seq(df_seq)` → **`df_clust`** pro

Sequence Analysis *logos · windows · motifs*

Position-specific logo
`AAlogo().get_df_logo(df_parts)` → **`df_logo`**

Sample reference windows
`AAWindowSampler().sample_*(df_seq)`

Pairwise sequence similarity
`comp_seq_sim(df_seq)` pro

Scan motifs (FIMO / MEME)
`scan_motif(df_seq, pwm)` → **`df_hits`** 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)` → **`df_feat`**

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

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

Feature Preprocessing *one-hot · PLM · structure · PTM*

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

PLM embeddings v1.1
`EmbeddingPreprocessor().encode(...)` → **`dict_num`**

Structure / DSSP / PAE v1.1
`StructurePreprocessor().encode_dssp(...)` → **`dict_num`** pro

PTM / site annotations v1.1
`AnnotationPreprocessor().encode(...)` → **`dict_num`** pro

Combine sources v1.1
`combine_dict_nums([...])` → **`dict_num`**

Numerical CPP v1.1
`CPP(df_parts).run_num(dict_num_parts, labels)` → **`df_feat`**

Modeling & Explainability *PU · classify · SHAP*

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

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

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

Metrics & Plotting *metrics · plots*

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`

Plot style, fonts & standalone legend
`plot_settings(font_scale) · plot_legend(ax)`

Protein Design (to be extended) *mutations · design*

In-silico point mutations v1.1
`AAMut · AAMutPlot`

Sequence-design libraries v1.1
`SeqMut · SeqMutPlot`

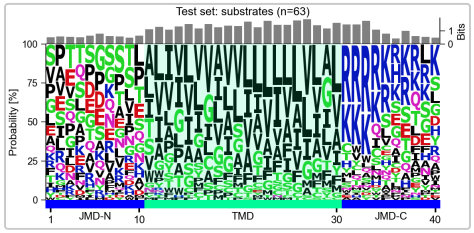
AAanalysis · Worked Examples

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 · test vs ref

```
# default parts + a redundancy-reduced set of 100 scales
df_parts = sf.get_df_parts(df_seq=df_seq)
df_scales = aa.AAclust().select_scales(
    df_scales=df_scales, n_clusters=100)
cpp = aa.CPP(df_parts=df_parts, df_scales=df_scales)
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()
```

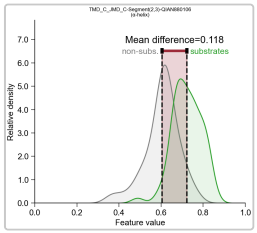
▼ *figure below · left*

CPP — ranking

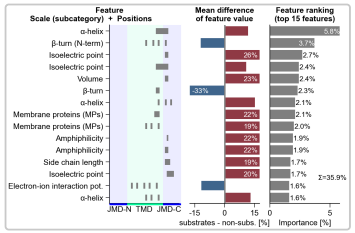
top features · effect + importance

```
# 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

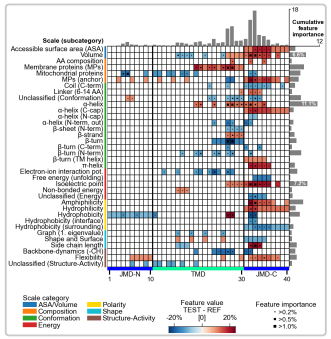


CPPPlot.ranking · top-15 features

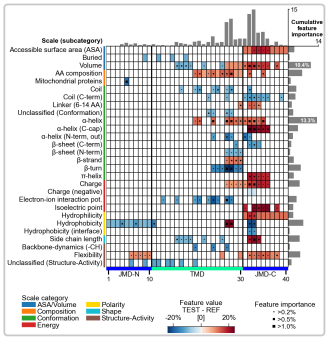
CPP — feature map

group level · importance

```
# 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)
# CPP.simplify → fewer, interpretable correlated scales
df_feat = cpp.simplify(df_feat=df_feat, labels=labels)
cpp_plot.feature_map(df_feat=df_feat)
plt.tight_layout(); plt.show()
```



CPPPlot.feature_map · all scales

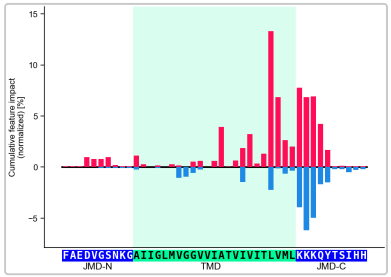


CPPPlot.feature_map · simplified

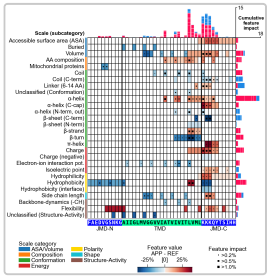
ShapModel — explain a prediction

sample level · [pro]

```
# advanced: per-sample explanation (fuzzy labeling demo)
# 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)
cpp_plot.feature_map(df_feat=df_feat, name_test='APP', **ka)
plt.tight_layout(); plt.show()
```



CPPPlot.profile · SHAP



CPPPlot.feature_map · SHAP

AAclust — clusters

scale reduction · clustering

```
aac = aa.AAclust()
aac.select_scales(df_scales, n_clusters=10)
aac.medoid_names_   # 10 reduced scales (labels_ also set)

aac_plot = aa.AAclustPlot()
aac_plot.centers(np.array(df_scales).T, labels=aac.labels_)
plt.tight_layout(); plt.show()
```

▼ *figure below · left*

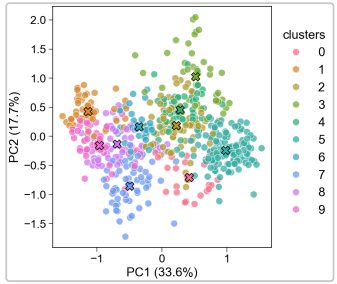
dPULearn — PCA

reliable negatives · PU learning

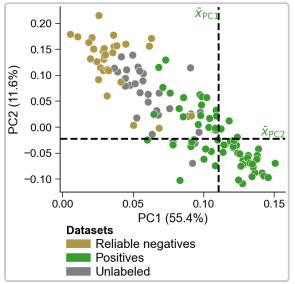
```
# DOM_GSEC ships 1/0 — treat 0 as the unlabeled pool (label_unl=0)
dpul = aa.dPULearn()
dpul.fit(X=X, labels=labels, label_unl=0, n_neg=31)   # n_neg:
reliable negatives to mine
df_pu = dpul.df_pu_   # out: 1 pos · 0 rel-neg · 2 unl

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

▼ *figure below · right*



AAclustPlot.centers · cluster scale profiles



dPULearnPlot.pca · reliable negatives

AAWindowSampler

build reference windows

```
# Reference windows around sites when you lack negatives:
aaws = aa.AAWindowSampler()
# SAME proteins · window 9 (odd) -> PTM / single-residue site
df_same = aaws.sample_same_protein(df_seq, n=100, window_size=9)
# DIFFERENT proteins · window 10 (even) -> cleavage bond
df_diff = aaws.sample_different_protein(df_seq, n=100,
window_size=10)
# SYNTHETIC — AA-frequency priors (null background)
df_syn = aaws.sample_synthetic(df_seq, n=100,
generator='global_freq')
```

AAanalysis · Concepts & Reference

decision guide · gotchas · options · class ↔ plot · glossary · citations

Decision Guide — *pick your setup*

What are you predicting?

per residue / site → AA_* · odd/even window · parts = window
per domain / region → DOM_* · TMD model · parts = jmd_n·tmd·jmd_c
whole protein → SEQ_* · composition · whole chain

What labels do you have?

labeled 0 / 1 → CPP → ML model
positives + unlabeled (1 / 2) → CPP → dPULearn → ML model
no negatives at all → AAWindowSampler → CPP → ML model

What is your learning task?

classify → CPP → classifier (sklearn)
regress → get_labels_quantile / tiered → CPP → regression model
multi-class → get_labels_ovr / ovo → CPP → classifiers
cluster → AAclust

Which explainability do you need?

group level → CPP → TreeModel → CPPPlot → feature importance
per protein → CPP → ShapModel (pro) → CPPPlot → feature impact ↑↓

Gotchas — *things that bite*

- Labels: **1/0** = supervised (pos/neg). **dPULearn** takes 1/0 (**label_unl=0**) or 1/2; **n_neg** = reliable negatives to mine; output **1 · 0 (rel-neg) · 2 (unl)**.
- **load_dataset(name, n=N)** returns **2N** rows (N per class) — count classes via df_seq['label'].
- Compositional vs positional is not a flag — it **emerges from split_kws**.
- Reproducibility: layered seeds — seed= ▸ random_state= ▸ options['random_state'] ▸ default.
- **DOM_*** parts need tmd_start/tmd_stop in df_seq; **[pro]** features need `pip install 'aaanalysis[pro]'`.
- **TMD** = **Target** Middle Domain — generalized from **Transmembrane** domain (Breimann25a) to any central segment; JMD-N / JMD-C are its flanks.

Design Principles — *the AAanalysis way*

- 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)

Key Data Objects — *shapes & columns*

df_seq	entry · sequence · label · tmd_start · tmd_stop
df_parts	one column per part: tmd · jmd_n · jmd_c · …
df_feat	feature · category · subcategory · scale_name · abs_auc · mean_dif · p_val · positions
X	feature matrix (samples × features) from sf.feature_matrix
dict_num	{entry: ndarray (L×D)} — numerical per-residue values (v1.1)

Class · abbr ↔ Plot Class — *mirrored API*

CLASS	ABBR	PLOT CLASS	KIND
SequencePreprocessor	sp	–	
EmbeddingPreprocessor	ep	–	
StructurePreprocessor (pro)	stp	–	
AnnotationPreprocessor (pro)	ap	–	
AAlogo	aal	AAlogoPlot	
AAWindowSampler	aaws	–	
SequenceFeature	sf	–	
CPP	cpp	CPPPlot	
AAclust	aac	AAclustPlot	Wrapper
NumericalFeature	nf	–	
dPULearn	dpul	dPULearnPlot	Wrapper
TreeModel	tm	–	Wrapper
ShapModel (pro)	sm	–	Wrapper
AAMut	aamut	AAMutPlot	
SeqMut	seqmut	SeqMutPlot	

aa.options — *system-level settings*

```
aa.options['random_state'] = 42
aa.options['verbose'] = True
aa.options['n_jobs'] = -1            # all cores (None = auto)
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
aa.options['df_scales'] = my_scales
```

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 ↗

Glossary — *canonical vocabulary · CONTEXT.md*

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

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.

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

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

Test vs reference group — The A-vs-B contrast CPP profiles: a feature's mean_dif is test − reference (name_test / name_ref in CPPPlot).

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

Numerical CPP (pseudo-scale) (v1.1) — CPP generalizes from AA→scale lookup to any per-residue tensor — PLM · structure · PTM — each a pseudo-scale via CPP.run_num.

Feature importance vs impact — Two explainability axes: importance = unsigned, group-level (TreeModel); impact = signed, per-sample (ShapModel, shap_plot).

Reducing features — Four distinct ops: redundancy reduction (AAclust scales) · feature pruning · selection (RFE) · simplification (CPP.simplify → interpretable scales).

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

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

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