

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 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

- LOAD** load\_dataset · load\_scales → df\_seq · df\_scales
- PARTS** SequenceFeature.get\_df\_parts → df\_parts
- FEATURES** Part × Split × Scale · CPP.run → df\_feat
- MODEL** TreeModel.fit · dPULearn.fit → feat\_importance · labels\_
- 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 domain); *variable length*.

**JMD** Juxta Middle Domain — the fixed-width flanks adjoining the TMD (jmd\_n on the N-side, jmd\_c on the C-side).



## 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

```
TMD          A I I G L M V G G V V I
Segment(1,4) ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Pattern(N,1,4,8) ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
PeriodicPattern ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
```

Splitting maps parts (of various length) to fixed relative positions.  
 Simplified from Breimann25 (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).  
 See details in Breimann25 · Suppl. Fig. 1 ↗

## 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)**

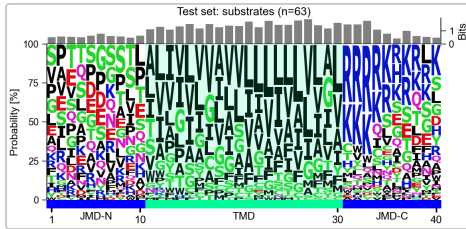
## Protein Design (to be extended)

mutations · design

In-silico point mutations v1.1  
**AAMut · AAMutPlot**  
 Sequence-design libraries v1.1  
**SeqMut · SeqMutPlot**

**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'])
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*

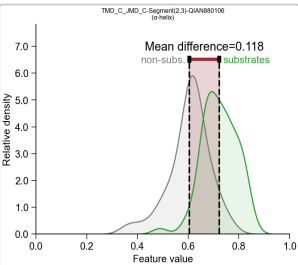
```
# default parts + a redundancy-reduced set of 100 scales
df_parts = sf.get_df_parts(df_seq=df_seq)
df_scales = aa.AAclust().select_scales(
    df_scales=df_scales, n_clusters=100)
cpp = aa.CPP(df_parts=df_parts, df_scales=df_scales)
df_feat = cpp.run(labels=labels, n_filter=100)
X = sf.feature_matrix(df_feat='df_parts',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

figure below · left

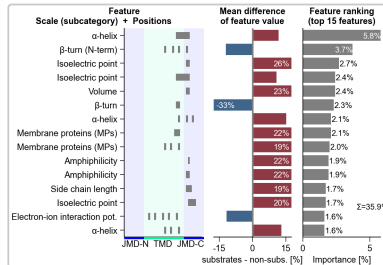
**CPP — ranking** *top features · effect + importance*

```
# 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.')
```

figure below · right



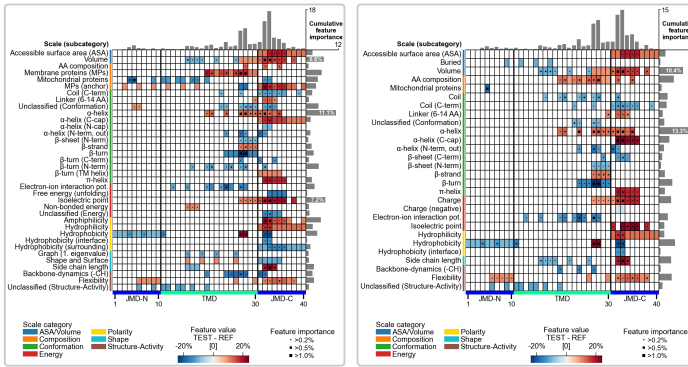
CPPPlot.feature · top feature



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()
```

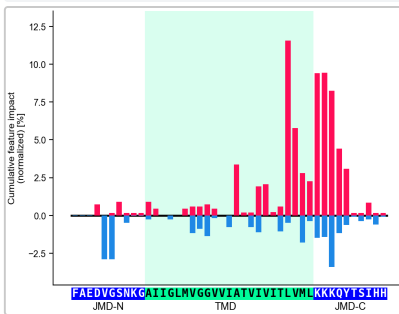


CPPPlot.feature\_map · all scales

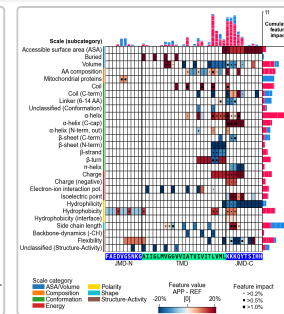
CPPPlot.feature\_map · simplified

**ShapModel — explain a prediction** *sample level · [pro]*

```
# per-sample SHAP — APP's soft label (0.6), keyed by entry
sm = aa.ShapModel()
sm.fit(X, labels=labels, df_seq=df_seq,
    fuzzy_labels={'P05067': 0.6})
df_feat = sm.add_feat_impact(df_feat=df_feat, df_seq=df_seq,
    samples='P05067', names='APP')
seq_kws = sf.get_seq_kws(df_seq=df_seq, df_parts=df_parts,
    sample='P05067')
ka = dict(col_imp='feat_impact APP', shap_plot=True, **seq_kws)
cpp_plot.profile(df_feat=df_feat, **ka)
# vmin/vmax=±21% → same color scale as the global feature map
(comparable)
cpp_plot.feature_map(df_feat=df_feat, name_test='APP',
    vmin=-21, vmax=21, **ka)
plt.tight_layout(); plt.show()
```



CPPPlot.profile · SHAP  
 See details in Breimann25 · Suppl. Fig. 10



CPPPlot.feature\_map · SHAP

**AAclust — clusters** *scale reduction · clustering*

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

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

# AAclust also reduces redundant proteins (not just scales)
df_seq = aac.select_proteins(df_seq=df_seq, X=X)
```

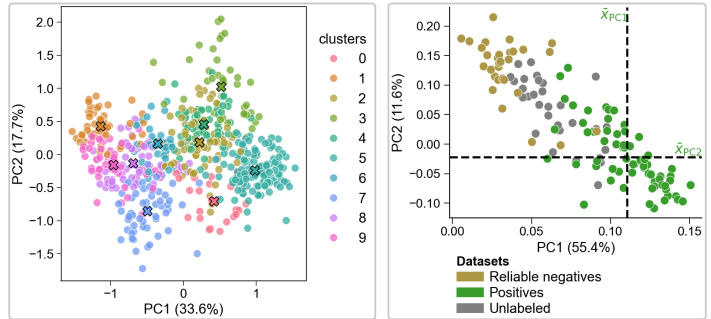
figure below · left

**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  
 See details in Breimann25 · Suppl. Fig. 3  
 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')
```

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]'.

Glossary

**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**.

**TMD (Target Middle Domain)** — The central *variable-length* sequence segment of interest, made comparable across samples by splitting. Called “transmembrane domain” in [Breimann25](#), but generalized here to any target segment.

**JMD (Juxta Middle Domain)** — The *fixed-length* flanking regions adjacent to the TMD: **jmd\_n** (N-terminal side) and **jmd\_c** (C-terminal side); called “juxtamembrane domain” in [Breimann25](#).

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

**Scale** — AA (amino acid) → ℝ mapping. AAontology ships ~600 curated scales in two-level categories.

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*

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

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	–	
NumericalFeature	nf	–	
AAclust	aac	AAclustPlot	Wrapper
CPP	cpp	CPPPlot	
dPULearn	dpul	dPULearnPlot	Wrapper
TreeModel	tm	–	Wrapper
ShapModel (pro)	sm	–	Wrapper
AAMut	aamut	AAMutPlot	
SeqMut	seqmut	SeqMutPlot	

**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) (v1.2)** — CPP generalizes from AA→scale lookup to any per-residue tensor — PLM (protein language model) · structure (PDB file) · functional annotations (e.g. PTMs) — each a pseudo-scale via CPP.run\_num.

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 and 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

Breimann & Frishman (2024a), AAclust: k-optimized clustering for selecting redundancy-reduced sets of amino acid scales [Bioinformatics Advances](#) ▶ [\[Breimann24a\]](#)

AAontology

Breimann et al. (2024b), AAontology: An ontology of amino acid scales for interpretable machine learning [Journal of Molecular Biology](#) ▶ [\[Breimann24b\]](#)

CPP & dPULearn

Breimann & Kamp et al. (2025), Charting γ-secretase substrates by explainable AI [Nature Communications](#) ▶ [\[Breimann25\]](#)

*canonical vocabulary · CONTEXT.md*

**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, recursive feature elimination) · simplification (CPP.simplify → interpretable scales).

**PU learning (Positive-Unlabeled learning)** — Training from labeled positives and unlabeled data only (no curated negatives); dPULearn infers reliable negatives from the unlabeled pool.

**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.