A Machine Learning Classification Model for Gold-Binding Peptides

Ali Ahmadi Esfidi

May 2025

1 Gold-Binding Peptides

Gold-binding peptides are short chains of amino acids (typically 5–20 residues) that have a natural affinity for gold surfaces or nanoparticles. They are identified or designed so that specific residues (often cysteine, histidine, or aromatic amino acids) coordinate with gold atoms, allowing the peptide to stick strongly and specifically to gold.[1]

1.1 Usage

1. **Nano-templating & Nanofabrication.** Peptides guide the formation of gold nanowires, rods, or particles with controlled size and shape, serving as a "molecular mold" for electronic or optical devices.[2]

2. **Biosensing & Diagnostics.** When immobilized on electrodes or sensor surfaces, gold-binding peptides can capture target biomolecules (e.g., antibodies) in a precise orientation, improving sensitivity for medical assays.[3]

3. **Targeted Drug Delivery & Imaging.** Conjugating drugs or imaging agents to gold nanoparticles via these peptides allows for targeted delivery and enhanced imaging contrast in cancer or inflammatory disease models.[4]

4. **Surface Functionalization.** They enable simple, one-step coating of gold surfaces with proteins or other functional polymers, useful in creating antifouling coatings or bioactive interfaces.

1.2 Intensity

In spot-array binding assays, each 10-residue peptide is immobilized on a solid support and exposed to gold nanoparticles. The *intensity* is defined as the

[1] https://pmc.ncbi.nlm.nih.gov/articles/PMC10337651/
[2] https://pubs.rsc.org/en/content/articlepdf/2023/ra/d3ra04269c
[3] https://pmc.ncbi.nlm.nih.gov/articles/PMC9918321/
[4] https://www.sciencedirect.com/science/article/abs/pii/S0378517324011542

optical (colorimetric) signal measured at each peptide spot, proportional to the amount of bound nanoparticles. Peptides with stronger binding produce darker spots (higher intensity), whereas weak or non-binding sequences yield lighter spots (lower intensity).

In the dataset of Janairo *et al.*, each of the 1 720 unique 10-mer peptides was assigned an intensity value based on the median image-analysis readout from Tanaka *et al.*'s screen. To classify peptides into binders and non-binders, the median intensity of the entire set, denoted I_{med}, was used as the threshold:

$$I_{\text{med}} = 207{,}500 \quad \text{(arbitrary units)}.$$

Peptides were then dichotomized into two classes:

$$\begin{aligned} \text{Class A (strong binders)} &: I > I_{\text{med}}, \\ \text{Class B (weak/non-binders)} &: I \leq I_{\text{med}}. \end{aligned}$$

Thus, the intensity values (in arbitrary units) provide a relative measure of each peptide's adsorption of gold nanoparticles under the standardized assay conditions.

2 Formulation of Problem

- **Input:**
  1. **Peptide sequence:** A string of ten amino-acid letters.
  2. **Derived features:** Each sequence is converted into a fixed-length numeric vector, so that the model can process it.
- **Output:** A binary prediction for each input sequence:
  - "Strong binder" (class A) if the model believes the peptide's binding intensity would exceed the threshold set by the median of all measured intensities.
  - "Weak/non-binder" (class B) otherwise.

2.1 Mathematical Presentation

1. Peptide Sequences:
$$S = \{s_i\}, \quad s_i \in \mathcal{A}^{10}$$
2. Feature Mapping:
$$\forall s_i \in S, \quad \Phi(s_i) = x_i \in R^d$$
3. Binding Intensities:
$$I = \{I_i\}_{i=1}^{N}, \quad I_i \in R_{\geq 0},$$

4. Classification Labels:

$$y_i = \begin{cases} A, & I_i > T, \\ B, & I_i \leq T, \end{cases} \quad T = \text{median}\big(\{I_i\}\big)$$

5. Prediction Function:

$$\forall s_i \in S, \quad f(\Phi(s_i)) = \hat{y}_i \in \{A, B\}$$

2.2 Assumptions

1. All peptides have fixed length 10.
2. Median intensity threshold T meaningfully separates strong vs. weak binders.
3. Samples are independently and identically distributed.

3 Dataset

Dataset comprises 1,720 peptide sequences, which we classify as A or B based on their intensity values.[5]

3.1 Distribution

It's important to confirm class balance before training classifiers, as imbalanced datasets can introduce bias.

The class sizes are nearly identical; exact count: **861** for Class A and **859** for Class B.

However, as illustrated in the violin plot and histogram in Figure 1, Class A peptides are high-intensity binders with intensity values concentrated around 250,000. Class B peptides, conversely, exhibit lower intensity values, ranging widely from nearly 0 to 200,000, and display a prominent left tail.

4 Related Work

As part of his lecture series, Jose Isagani B. Janairo presents a practical workflow for classifying gold-binding peptides that elegantly demonstrates end-to-end machine-learning in R: starting with raw amino-acid sequences, deriving ten physicochemical descriptors via Kidera factors, and organizing those into a labeled dataset; next, stratified train–validation splitting and 10-fold cross-validation to tune and compare multiple classifiers (logistic regression, decision trees, k-nearest neighbors, SVMs with various kernels, and a neural network); then refining the best performer—a radial-basis SVM—by selecting only the

[5] `https://pubs.acs.org/doi/suppl/10.1021/acsomega.2c00640/suppl_file/ao2c00640_si_001.pdf`

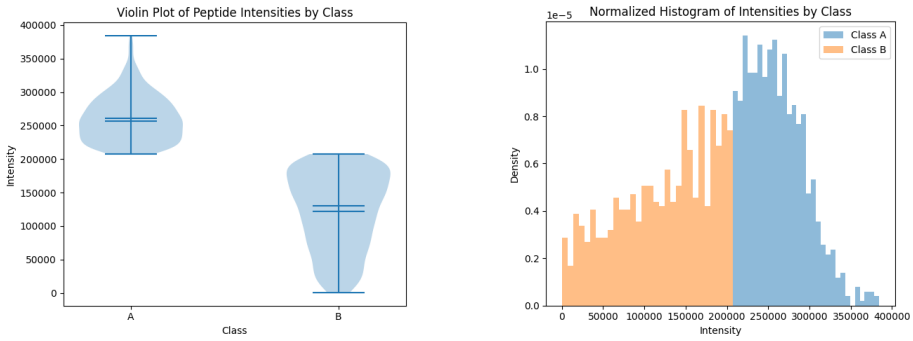

Figure 1: Overall caption for both images.

most informative descriptor subset; and finally, assessing generalization on hold-out data and employing permutation-based feature importance to reveal which peptide properties most strongly drive gold-binding predictions, all accompanied by clear visualizations of model performance and variable rankings.

On a macOS environment, the code was executed and produced results (Table 1) consistent with the related lecture (Accuracy: 0.8019).

	Reference	
	A	**B**
Prediction A	175	45
Prediction B	40	169

Table 1: Confusion Matrix

Also Table 2 reports the permutation-based importance of each Kidera factor (KF1–KF10) for the final radial-basis SVM, together with uncertainty bounds and the baseline increase in classification error when that feature is shuffled.

In practice, this tells you that the model heavily relies on KF4, KF2, and KF3—disrupting these degrades performance the most—whereas KF1 and KF8 contribute almost nothing to gold-binding predictions in the final SVM.

5 Alternative Models and Embeddings

To further improve classification performance, we can explore alternative modelling techniques and sequence embedding methods.

5.1 Modelling Techniques

- **XGBoost:** A highly optimized implementation of gradient-boosted decision trees. It handles missing values natively, regularizes to prevent

feature	importance	permutation.error
KF4	1.446512	0.2408985
KF2	1.395349	0.2323780
KF3	1.395349	0.2323780
KF9	1.306977	0.2176607
KF10	1.158140	0.1928737
KF5	1.148837	0.1913246
KF7	1.144186	0.1905500
KF6	1.130233	0.1882262
KF1	1.000000	0.1665376
KF8	1.000000	0.1665376

Table 2: Feature Importance

overfitting, and often achieves state-of-the-art results on tabular data.

- **Support Vector Machines (SVM):** Effective in high-dimensional spaces and robust to overfitting when properly regularized. Kernel functions (e.g., RBF, polynomial) allow SVMs to capture non-linear relationships.
- **Neural Networks:** Neural networks are composed of interconnected layers of artificial neurons that learn hierarchical feature representations directly from input embeddings. By stacking multiple dense or specialized layers and employing non-linear activation functions (e.g., ReLU, GELU), they can capture complex, non-linear relationships in the data. Key design considerations include:

5.2 Sequence Embedding Methods

- **Amino Acid Composition (AAC):** A simple, interpretable feature vector of length 20 that counts the frequency of each amino acid in a peptide or protein.
- **Kidera Factors:** A set of 10 physicochemical descriptors derived via multivariate analysis, summarizing properties like hydrophobicity, bulkiness, and electronic characteristics.
- **BLOSUM62 Encoding:** Using the BLOSUM62 substitution matrix rows as 20-dimensional vectors for each residue, capturing evolutionary substitution preferences.

We concatenate these embeddings to construct a more informative representation of each peptide sequence. The resulting combined embedding is then used as input to the chosen classification model (e.g., XGBoost, SVM, or a neural network). This approach may also reveal the presence of additional discriminative features.

6 Support Vector Machines (SVM)

This section details the application of a Support Vector Machine (SVM) classifier. The dataset underwent a rigorous preprocessing sequence to prepare it for model training. First, it was stratified into training and testing sets, ensuring the preservation of the original class distribution. Feature magnitudes were subsequently normalized using `StandardScaler` to bring them to a common scale. A crucial step involved **feature selection**, applied to the scaled training data using `SelectKBest` with the `mutual_info_classif` scoring function. This process identified and retained the most informative features, which were then used to transform both the training and test sets, reducing dimensionality and potential noise. Finally, a Support Vector Machine classifier was trained on these selected features. Its hyperparameters were optimized using `GridSearchCV` in conjunction with `StratifiedKFold` cross-validation. The model optimization employed the **Receiver Operating Characteristic Area Under the Curve (ROC AUC)** as the primary scoring metric, and `class_weight='balanced'` was set to effectively address potential class imbalance within the dataset.

7 XGBoost Classifier

The combined feature set was subjected to a standard preprocessing workflow for the XGBoost model. Initially, the data was stratified into training and testing sets to preserve the intrinsic class proportions. **Feature normalization** was then performed using `StandardScaler`, fitted exclusively on the training data to prevent data leakage. A critical step in this pipeline was **feature selection**, implemented via `SelectKBest` with `mutual_info_classif`. This aimed to distill the most relevant features, thereby reducing both dimensionality and potential noise. The core classification model, an XGBoost classifier, was optimized using `GridSearchCV` with 5-fold cross-validation. The optimization process prioritized the **ROC AUC** as the primary scoring metric and incorporated the `scale_pos_weight` parameter to effectively address any existing class imbalance.

8 Residual Attention Classifier

The data for the Residual Attention Classifier underwent a meticulous preprocessing pipeline. To ensure the preservation of original class distributions, the dataset was initially split into stratified training, validation, and test sets. **Feature scaling** was subsequently performed using `StandardScaler`, fitted exclusively on the training data. A key enhancement involved **feature selection** via `SelectKBest` with `mutual_info_classif`, which reduced dimensionality and emphasized highly informative features.

A custom `FeatureDataset` class was developed for PyTorch, enabling on-the-fly **data augmentation** through the application of Gaussian noise and Mixup. This augmentation strategy was implemented to improve model generalization and robustness. The `ClassifierNN` architecture integrates several ad-

vanced components: **Residual Blocks** for facilitating deeper network training and mitigating vanishing gradients, **Attention Pooling** for adaptively weighing feature importance, and **Exponential Moving Average (EMA)** for achieving more stable model inference. During training, the loss was computed using `BCEWithLogitsLoss` with **class weighting** to effectively mitigate class imbalance. The model was optimized using `AdamW`, a robust optimizer, with a configurable learning rate scheduler (either `CyclicLR` for cyclical learning rates or `ReduceLROnPlateau` for adaptive learning rate reduction). Additionally, **early stopping** was employed to prevent overfitting, and **gradient clipping** was applied to maintain training stability.

Algorithm 1 Residual Attention Architecture

function ResidualBlock(x, in_dim, out_dim, dropout_rate)
    id $\leftarrow$ x
    $h \leftarrow \text{Swish}(\text{BatchNorm}(\text{Linear}_1(x)))$
    $h \leftarrow \text{Dropout}(h)$
    $h \leftarrow \text{BatchNorm}(\text{Linear}_2(h))$
    return $\text{Swish}(h + \text{id})$
end function

function AttentionPooling(x)
    $s \leftarrow \text{Linear}_2(\text{Tanh}(\text{Linear}_1(x)))$ ▷ Attention scores
    $\alpha \leftarrow \text{softmax}(s)$ ▷ Feature weights
    return $x \odot \alpha$ ▷ Element-wise multiplication
end function

function ClassifierNN(x)
    $h_0 \leftarrow \text{Linear}_{\text{proj}}(x)$ ▷ Initial projection
    $h_{i+1} \leftarrow \text{ResidualBlock}(h_i)$
    $y \leftarrow \text{AttentionPooling}(h_a)$
    return $\text{Linear}_{\text{out}}(y)$ ▷ Binary classification output
end function

9 Siamese-like Classifier

The Siamese-like Classifier pipeline commenced by independently loading and processing three distinct types of amino acid sequence embeddings: `iFeature AAC`, `Kidera`, and `BLOSUM62`. Consistency of the target variable was ensured across these diverse datasets. The data for each embedding type then underwent a meticulous preprocessing pipeline, involving **stratified splitting** into training, validation, and test sets to maintain class proportions. This was followed by **feature scaling** using `StandardScaler`, fitted exclusively on the respective training data for each embedding. A critical step was the application

of `SelectKBest` with `mutual_info_classif` for **feature selection**, aiming to reduce dimensionality and focus on highly informative features specific to each embedding type.

The implementation leveraged a custom `FeatureDataset` for PyTorch, which supported on-the-fly **data augmentation** techniques such as Gaussian noise and Mixup to further enhance model generalization. The core `SiameseClassifier` architecture employed separate `EmbeddingBranch` networks for each embedding type. This design allowed each branch to learn distinct, specialized representations from its respective input. The outputs of these branches were then concatenated into a shared classification head. Each `EmbeddingBranch` incorporated **SiLU activation functions**, **BatchNorm1d** for improved training stability, and **Dropout layers** for regularization. The overall model was optimized using `AdamW`, with a configurable learning rate scheduler (`CyclicLR` or `ReduceLROnPlateau`).

During training, `BCEWithLogitsLoss` with **class weighting** was utilized to address inherent class imbalance. Furthermore, **early stopping** was implemented to prevent overfitting, and **gradient clipping** was applied to ensure training stability. The weights of the best-performing model, as determined by the validation metric, were saved using an `EarlyStopping` mechanism.

Algorithm 2 Siamese-like Architecture

function EMBEDDINGBRANCH(x, input_dim, out_dim, dropout_rate)
    $x_{i+1} \leftarrow \text{Dropout}(\text{SiLU}(\text{BatchNorm}(\text{Linear}(x_i))))$
    $x_a \leftarrow \text{Linear}(x_a)$
    $x_a \leftarrow \text{BatchNorm}(x_a)$
    return $\text{SiLU}(x_a)$
end function

function SIAMESECLASSIFIER($x_{\text{aac}}, x_{\text{kidera}}, x_{\text{blosum}}$)
    $\text{emb}_{\text{aac}} \leftarrow \text{EmbeddingBranch}(x_{\text{aac}})$
    $\text{emb}_{\text{kidera}} \leftarrow \text{EmbeddingBranch}(x_{\text{kidera}})$
    $\text{emb}_{\text{blosum}} \leftarrow \text{EmbeddingBranch}(x_{\text{blosum}})$
    $c \leftarrow \text{Concat}(\text{emb}_{\text{aac}}, \text{emb}_{\text{kidera}}, \text{emb}_{\text{blosum}})$
    $h \leftarrow \text{Linear}(c, \dim_c/2)$
    $h \leftarrow \text{BatchNorm}(h)$
    $h \leftarrow \text{SiLU}(h)$
    $h \leftarrow \text{Dropout}(h)$
    return $\text{Linear}(h)$
end function

10 Results

This section presents the performance of the various machine learning models employed for classifying gold-binding peptides. The evaluation metrics include Accuracy, F1-score (for both classes and macro average), Precision, Recall, and ROC AUC, alongside their respective confusion matrices.

Table 3 provides a concise summary of the key performance metrics for all evaluated models.

Model	**Accuracy**	**F1-score (Macro)**	**Precision**	**Recall**	**True Positives**
Related Work	0.802	0.811	0.796	0.847	344
XGBoost	0.819	0.818	0.799	0.851	352
SVM	0.828	0.828	**0.822**	0.837	356
Residual Attention NN	0.833	0.832	0.810	0.870	358
Siamese-like NN	**0.840**	**0.839**	0.809	**0.888**	**361**

Table 3: Comparative Summary of Model Performance on Test Set

The Siamese-like Neural Network, leveraging multiple embedding types, achieved the best overall performance in terms of accuracy and F1-score.