CSEN342 - Program 1 (Peptide Classification)

Aditya Puranik
aspuranik@scu.edu
7700005906 - Rank 7, MCC Score 87.93
Professor: Dr. David Anastasiu

I. Introduction

THE task involves categorizing **peptide sequences into two groups, labeled as 1 and -1**, via a binary classification approach. The dataset comprises two distinct sets of peptide sequences: a training set (train.dat) and a test set (test.dat). Each sequence is presented on an individual line within these files. The sequences are depicted as strings, consisting of 20 characters that correspond to amino acid residues. In the training set, each sequence is accompanied by an associated label, which can be **either 1 (signifying antibiofilm) or -1 (indicating non-antibiofilm)**. These labels are placed at the beginning of each line in the training file, and are separated from the sequence by a tab character.

II. Methodology

To establish a robust foundation for our neural network, we first need to undertake data pre-processing, transforming peptide sequences into a format conducive to computational analysis. This is followed by addressing the prevalent issue of class imbalance, employing targeted techniques to counteract its potential effects on the model's predictive efficacy and overall accuracy.

II-A. Feature Extraction

During the first stage of feature extraction and pre-processing, we investigated **the use of k-mers**, which have a range of 1 to 3, to represent the peptide sequences. However, this approach **did not adequately capture** the full spectrum of characteristics inherent in the peptide sequences, leading to a limited representation of their properties. Recognizing this limitation, we transitioned to **Amino Acid Composition Analysis**. This method involved constructing a vector for each peptide sequence, where the frequency of occurrence of each amino acid was quantified. This method worked better at encapsulating the peptides' underlying patterns and tendencies, providing a richer and more meaningful representation for later classification tasks.

This document corresponds to the final submission of the CSEN342 Program 1 submitted at Santa Clara Univeristy, CA, Winter'24

II-B. Dealing with Class Imbalance

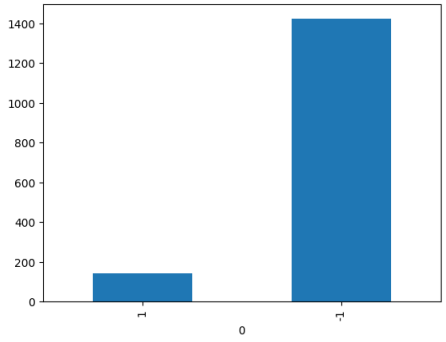

Figura 1: Class imbalance: anti-biofilm vs not anti-biofilm sequences

In our dataset, addressing class imbalance was critical due to the disproportionate distribution between the classes, **with 1424 instances in class -1 and only 142 in class 1**. To rectify this imbalance, various strategies were evaluated, including oversampling techniques, which are effective in augmenting the minority class to match the majority class's prevalence. We experimented with several oversampling methods, such as Synthetic Minority Over-sampling Technique (**SMOTE**), **Borderline-SMOTE**, and Adaptive Synthetic Sampling (**ADASYN**), each offering a nuanced approach to generating synthetic samples for the minority class. Ultimately, SMOTE emerged as the most effective method in our context, as it significantly improved the representativeness of the minority class and enhanced the overall performance of our classification model.

III. Proposed Solution

III-A. Custom Neural Network Class

Our custom-designed Neural Network class implements a **two-layer architecture** specifically for peptide sequence classification comprising an input layer, a hidden layer, and an output layer, forming the foundational structure of the feedforward neural network. The network's input layer is *dimensioned to match the size of the peptide vector generated* through our feature extraction process.

The neural network operates by taking in the input data (peptide sequences), processing it through the hidden layer with a **ReLU** activation function, and producing output using a **sigmoid** activation function in the output layer. To evaluate

the model's performance and avoid overfitting, the dataset was split into a **training set (70 %) and a validation set (30 %)**. The class implements a training method that applies **backpropagation** and **stochastic gradient descent** for weight updates, incorporating the learning rate and regularization parameter to optimize performance.

Tabla I: Hyperparameter Iterations and Best Scenario

Hyperparameter	Values Tried	Best Scenario
Hidden Neurons	12, 15, 16, 18	16
Learning Rate	0.07, 0.08, 0.09, 0.1	0.08
Lambda_Reg	0.001, 0.0	0.0
Threshold	0.5, 0.8, 0.9	0.9
Class Weights	[1.0, 1.3], [1.0, 1.5], [1.1, 1.4]	[1.1, 1.4]

In terms of hyperparameters, the network was iteratively fine-tuned. We experimented with varying numbers of hidden nodes, learning rates, and regularization strengths to identify the most effective combination. The **learning rate was crucial for convergence speed and overall accuracy**, while the class weights helped us undo some oversampling issues of class 1. This process of iterative refinement and testing was key to enhancing the model's ability to accurately classify peptide sequences based on their respective characteristics. Our results demonstrated excellent proficiency of the model, with high accuracy on both the training and validation datasets. The model's precision was particularly noteworthy, as evidenced by the low proportion of false positives and negatives relative to true positives and negatives.

III-B. PyTorch Sequential NN Implementation

In parallel to our custom class, we implemented a **PyTorch model** using *nn.Sequential*, with 16 hidden nodes and ReLU and Sigmoid activations. We employed **Stochastic Gradient Descent** for optimization and **Binary Cross-Entropy** for loss calculation, maintaining identical hyperparameters. The same data pre-processing operations were applied to the data.

IV. Simulations and tests

IV-A. Loss Function

Effective learning and optimization were demonstrated in our simulations, where the training loss for the **Custom Neural Network class neared 0.0 over 100 epochs**. On the other hand, the **PyTorch implementation** showed a steady decline in training loss, which reached **0.005014**. This suggests a slightly different convergence behavior, but it nevertheless demonstrated a notable improvement in the model's learning capacity.

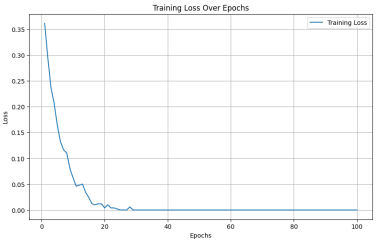

Figura 2: *Loss Function Plot - Custom Neural Network*

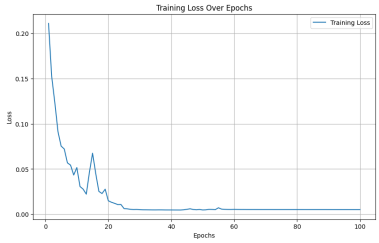

Figura 3: *Loss Function Plot - PyTorch Sequential NN*

IV-B. Confusion Matrix

The custom neural network's confusion matrix illustrates near-perfect class prediction, while the PyTorch model shows slightly more false positives, yet both display strong performance in correctly classifying the majority class.

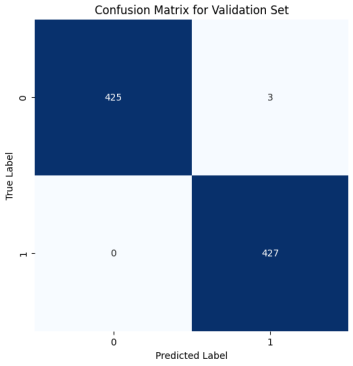

Figura 4: *Confusion Matrix - Custom Neural Network*

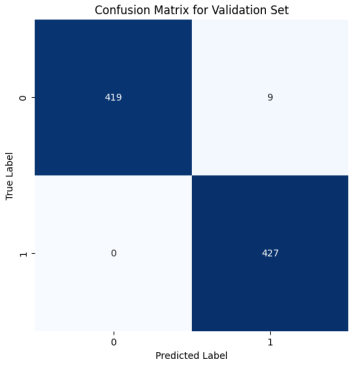

Figura 5: *Confusion Matrix - PyTorch Sequential NN*

V. Results & Conclusion

V-A. Results

The MCC on validation set were as follows:

Tabla II: MCC Scores on Validation Set

Model	MCC Score
Custom Neural Network	0.9930
PyTorch Neural Network	0.9792

The Custom Neural Network achieved an impressive **MCC** of *0.9930*, while the PyTorch implementation also performed well, with an **MCC** of *0.9792* on the validation set.

V-B. Conclusion

The **MCC score** (on 50 % data) is **87.93 %** on the test data using our developed custom neural network. Our investigation concludes that custom neural networks can be tailored to achieve **high classification accuracy**, as evidenced by the strong MCC scores. The robust MCC scores support our investigation's conclusion that custom neural networks can be tuned to attain high classification accuracy. Nevertheless, **the ease of use and advantages of utilizing pre-built frameworks like PyTorch** suggest a preference for their adoption in practice, balancing performance with implementation efficiency.