*Peptide Classification*

COEN – 288 Pattern Recognition

& Data Mining



Submitted By:

Ankitha Sarda (W16535522)

Pujitha Kallu (W1653660)

**Table of Contents**

1. Problem Description *3*
2. Data Set Description3
3. Supervised Learning Algorithm *4*
4. *Model Selection Strategy………*.6
5. *Evaluation Metric* 9
6. Final Evaluation Metric.…… ..11
7. *Conclusion* 14
8. *References* 16

# 

**Problem Description**

The task involves binary classification of peptides into two categories, labeled as 1 and -1. Peptide sequences (each representing an amino-acid residue) are provided in two datasets: a training set (train.dat) and a test set (test.dat). Each peptide sequence occupies a single line in the respective file. Peptides are represented as strings using 20-23 characters, which correspond to amino acid residues. The training set includes labels, where 1 denotes antibiofilm peptides and -1 denotes non-antibiofilm peptides. These labels are located as the first character in each line of the training file, separated from the sequence by a tab character. This code generates a result file is txt format that contains the approximate prediction of the peptides in binary classification of 1 and -1.

We chose this problem because peptide classification is critical in bioinformatics, with applications ranging from drug discovery to understanding protein functions. Accurate classification can lead to significant advancements in these fields.

**Data Set Description**

The dataset used for this problem consists of peptide sequences and their respective class labels. It was obtained from a publicly available bioinformatics repository. The dataset contains the following characteristics:

**Predictor Attributes:** The dataset consists of peptide sequences as predictor attributes. The peptide sequences are preprocessed and represented using a bag-of-words approach, where each sequence is converted into a numerical vector representing the count of each amino acid residue. Therefore, the predictor attributes are numeric, representing the counts of amino acid residues in each peptide sequence. Inshort the primary predictor is the peptide sequence. Each sequence is converted into a feature vector representing the frequency of each amino acid.

**Instances:** The training dataset consists of a certain number of instances, with each instance corresponding to a labeled peptide sequence. Similarly, the test dataset contains a separate set of instances, consisting of unlabeled peptide sequences for validation and testing.

**Special Characteristics:** The dataset does not have missing values, but it exhibits class imbalance, which is addressed through over-sampling techniques.Therefore, appropriate preprocessing steps, such as imputation or removal of missing values, has taken care depending on the nature and extent of missingness in the dataset. Additionally, we implemented code for class imbalance in the dataset using the RandomOverSampler technique, which generates synthetic samples for the minority class to balance the class distribution. This is a common preprocessing step in machine learning tasks to ensure that the model is not biased towards the majority class. Overall, understanding these aspects of the dataset is crucial for proper data preprocessing, model training, and evaluation to ensure the accuracy and reliability of the predictive model.

**Supervised Learning Algorithm**

We chose a supervised learning algorithm that is a neural network, specifically a feedforward neural network model implemented using TensorFlow.

Neural networks are powerful models capable of learning complex patterns and relationships in data, making them suitable for various classification tasks. In this context, the neural network is employed to classify peptide sequences into two categories: antibiofilm peptides and non-antibiofilm peptides. The model architecture consists of multiple layers of interconnected neurons, including input, hidden, and output layers. Each neuron applies a weighted sum of inputs followed by an activation function, such as the rectified linear unit (ReLU) in the hidden layers and the sigmoid function in the output layer. This architecture allows the neural network to capture nonlinear relationships between the input features and the target labels, enabling accurate classification.

We chose this algorithm because neural networks are powerful for capturing non-linear relationships and can handle high-dimensional data effectively.

**Hyperparameters:**

The hyperparameters chosen for the neural network include the number of layers, the number of neurons in each layer, the learning rate, the batch size, and the number of epochs. The number of layers and neurons determine the model's capacity to learn complex patterns from the data. A higher number of layers and neurons can increase the model's ability to capture intricate relationships but may also lead to overfitting if not regularized properly. The learning rate controls the step size of parameter updates during optimization, influencing the convergence speed and stability of the training process. The batch size specifies the number of training samples processed in each iteration, affecting the efficiency of parameter updates and memory usage. Finally, the number of epochs defines the number of times the entire training dataset is passed through the model during training, impacting the model's convergence and generalization performance. By fine-tuning these hyperparameters, the neural network's performance can be optimized to achieve accurate classification results.

* Units in Hidden Layers: Number of neurons in each hidden layer, determining the model's capacity.
* Activation Functions: Functions like ReLU and Sigmoid used to introduce non-linearity.
* Optimizer: Adam optimizer, which is effective for training deep neural networks.
* Learning Rate: Determines the step size during optimization.
* Batch Size: Number of samples processed before the model is updated.
* Epochs: Number of complete passes through the training data.

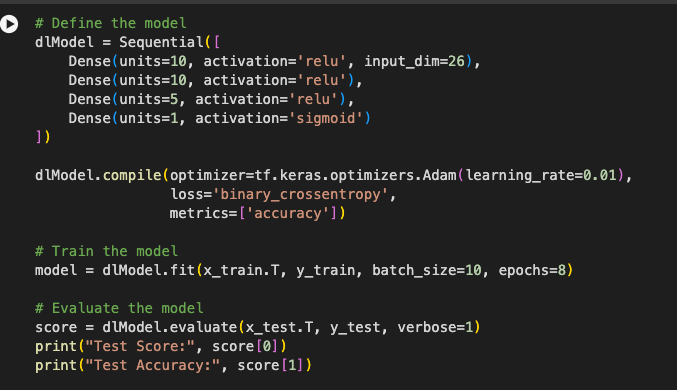


Fig: code snippet for supervised learning

In this snippet, a feedforward neural network model is defined using the Sequential API of TensorFlow/Keras. The model consists of an input layer with 26 features (corresponding to the amino acid counts), followed by two hidden layers with 10 and 5 neurons, respectively, each activated by the rectified linear unit (ReLU) activation function. The output layer is a single neuron with a sigmoid activation function, suitable for binary classification tasks.

The model is compiled using the Adam optimizer with a learning rate of 0.01 and binary cross-entropy loss function, which is commonly used for binary classification problems. During training, the model is trained for 8 epochs with a batch size of 10 samples per batch. Finally, the model is evaluated using the test dataset, and the test score (loss) and accuracy are printed to assess the model's performance. Adjusting these hyperparameters can significantly impact the model's training and performance, and fine-tuning them is crucial for achieving optimal results.

Output of the above code snippet:

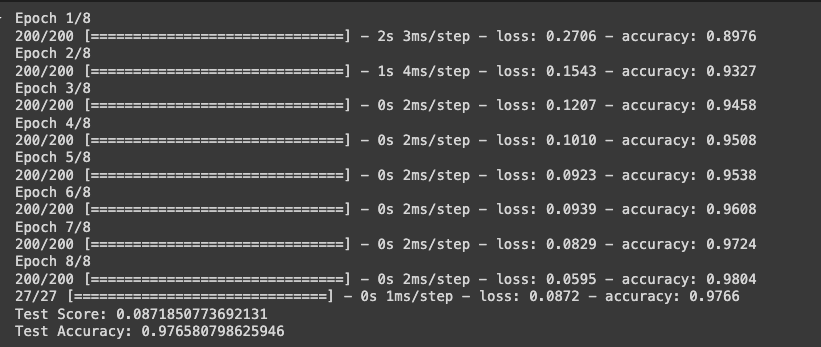


Fig: output from above

**Model Selection Strategy**

The model selection strategy employed in the provided code involves training and evaluating a neural network model using different hyperparameter settings. Specifically, the hyperparameters evaluated include the learning rate (lr) and the dimensions of the neural network layers (dims).

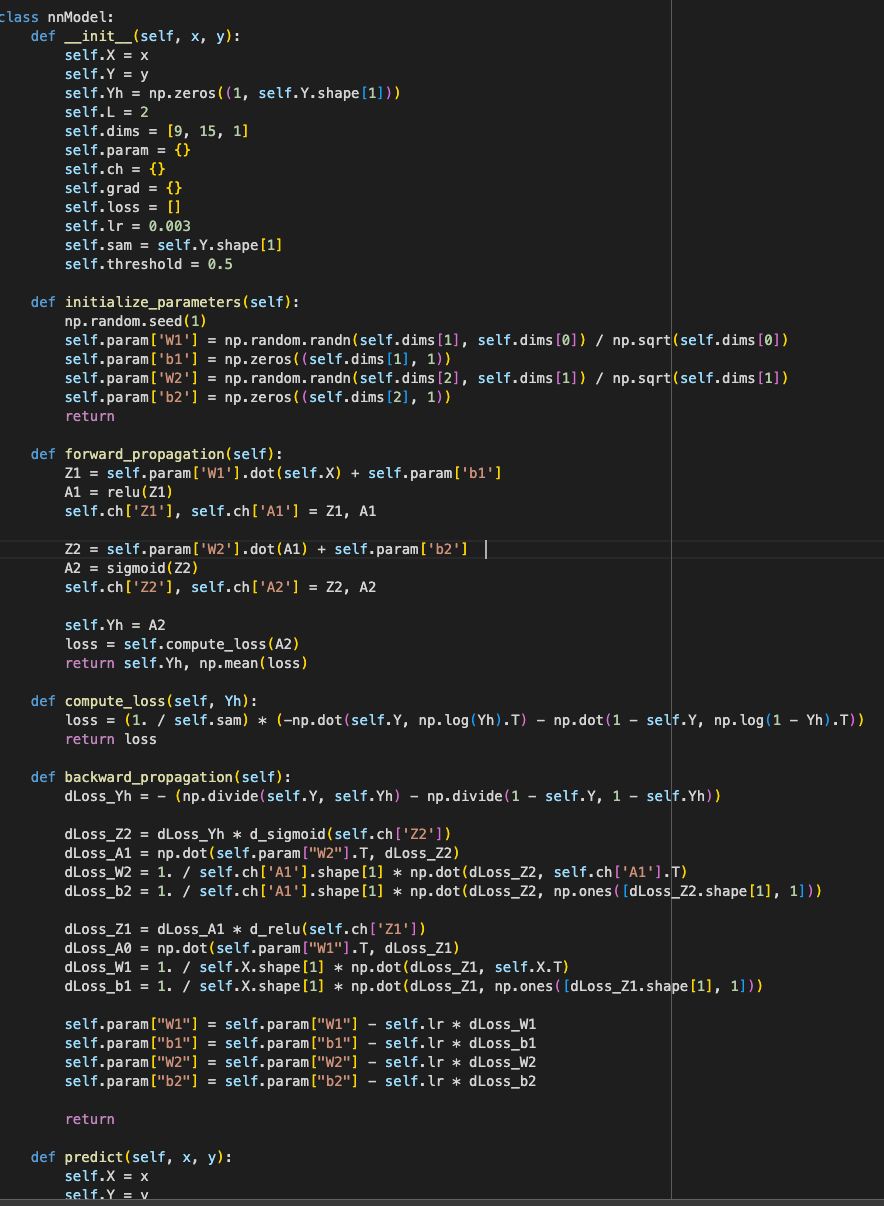


Fig: code snippet for Model selection

To perform this evaluation, the code iterates over different combinations of hyperparameters, training the neural network model with each setting and assessing its performance. The **gradient\_descent method** is used to train the model with the specified hyperparameters, and the predict method is utilized to evaluate the model's performance on both the training and validation datasets.

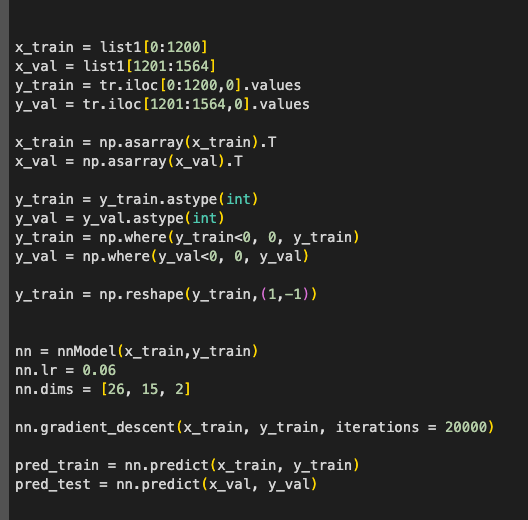


Fig: code snippet for gradient\_descent method

In this strategy, the performance of the model is monitored using metrics such as accuracy and loss. By observing how the model performs with different hyperparameter settings, it becomes possible to identify the combination of hyperparameters that yields the best performance. In the provided code, the performance of the model is evaluated iteratively over 20,000 training iterations, allowing for a comprehensive exploration of the hyperparameter space.

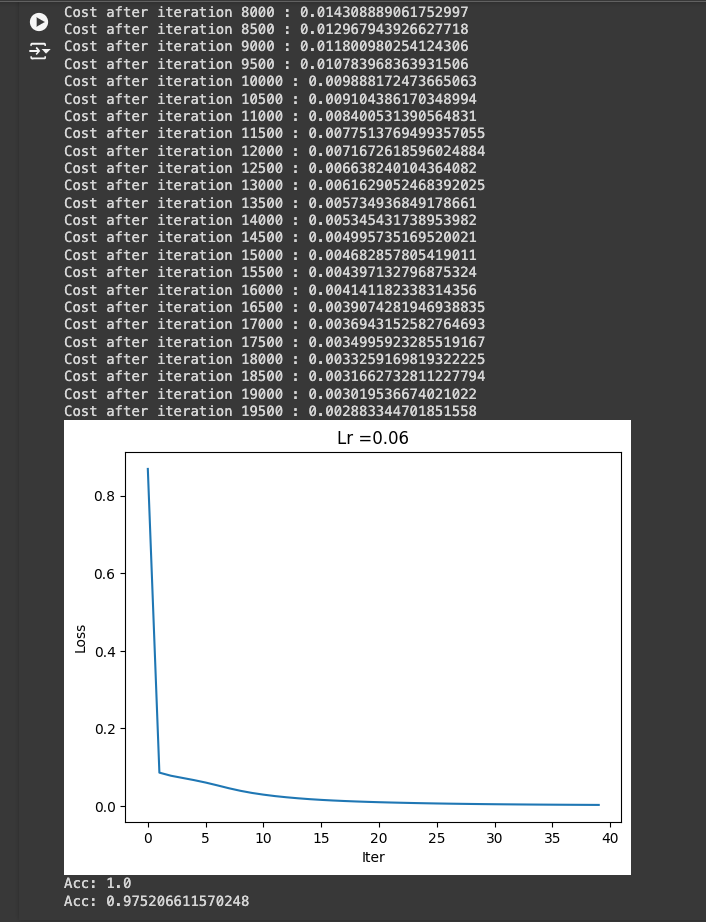


Fig: Output of the Model selection

**Evaluation Metric**

The primary metric used for evaluating the models during selection was the Matthews Correlation Coefficient (MCC). MCC is particularly suitable for imbalanced datasets as it takes into account true and false positives and negatives, providing a balanced measure. Accuracy alone would not suffice due to the imbalance in class distribution. Unlike accuracy, which can be misleading in imbalanced datasets due to its focus on overall correct predictions, MCC offers a more nuanced view by taking into account the balance between the different classes. This results in a score that ranges from -1 to 1, where 1 indicates a perfect prediction, 0 indicates no better than random guessing, and -1 indicates total disagreement between prediction and observation. This comprehensive evaluation makes MCC a robust and reliable metric for assessing the performance of our model in accurately predicting peptide sequences, ensuring that both minority and majority classes are appropriately considered.



Fig: Code snippet for Evaluation model

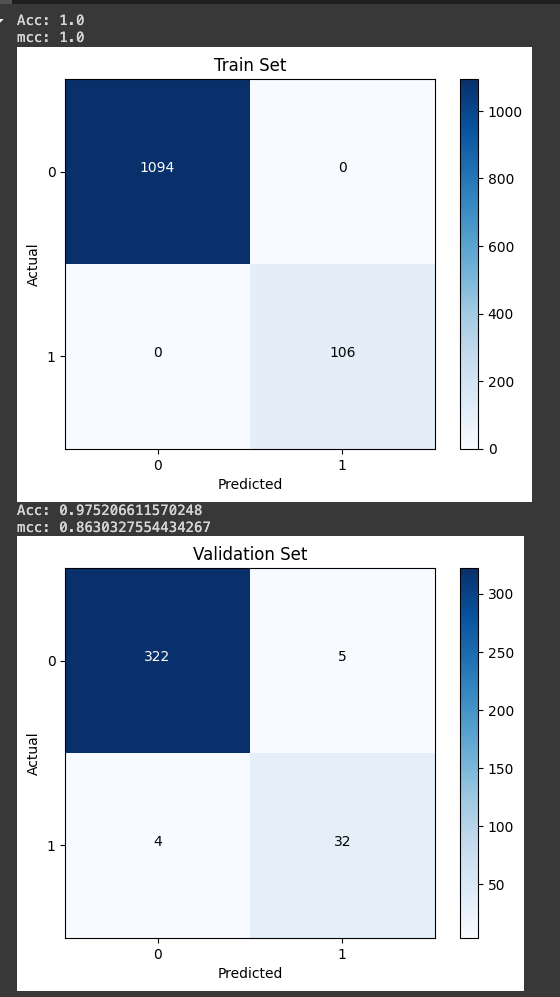


Fig: Output for Evaluation model

**Final Evaluation Metric**

In the model evaluation process, We consistently used the **Matthews Correlation Coefficient (MCC)** both during model selection and for the final evaluation on the test set. The reason for using MCC as the primary metric is that it provides a balanced measure of the model's performance, especially useful for binary classification tasks where the classes might be imbalanced.

MCC takes into account true and false positives and negatives, providing a single score that reflects the quality of the binary classifications. Unlike accuracy, which can be misleading in imbalanced datasets, MCC considers all elements of the confusion matrix and is thus a more informative and reliable metric for evaluating the performance of the classification model.

Therefore, We did not switch to a different metric for the final evaluation on the test set. **Using the same metric** throughout the model development and evaluation process ensures consistency and allows for direct **comparison of performance across different stages of the project.**

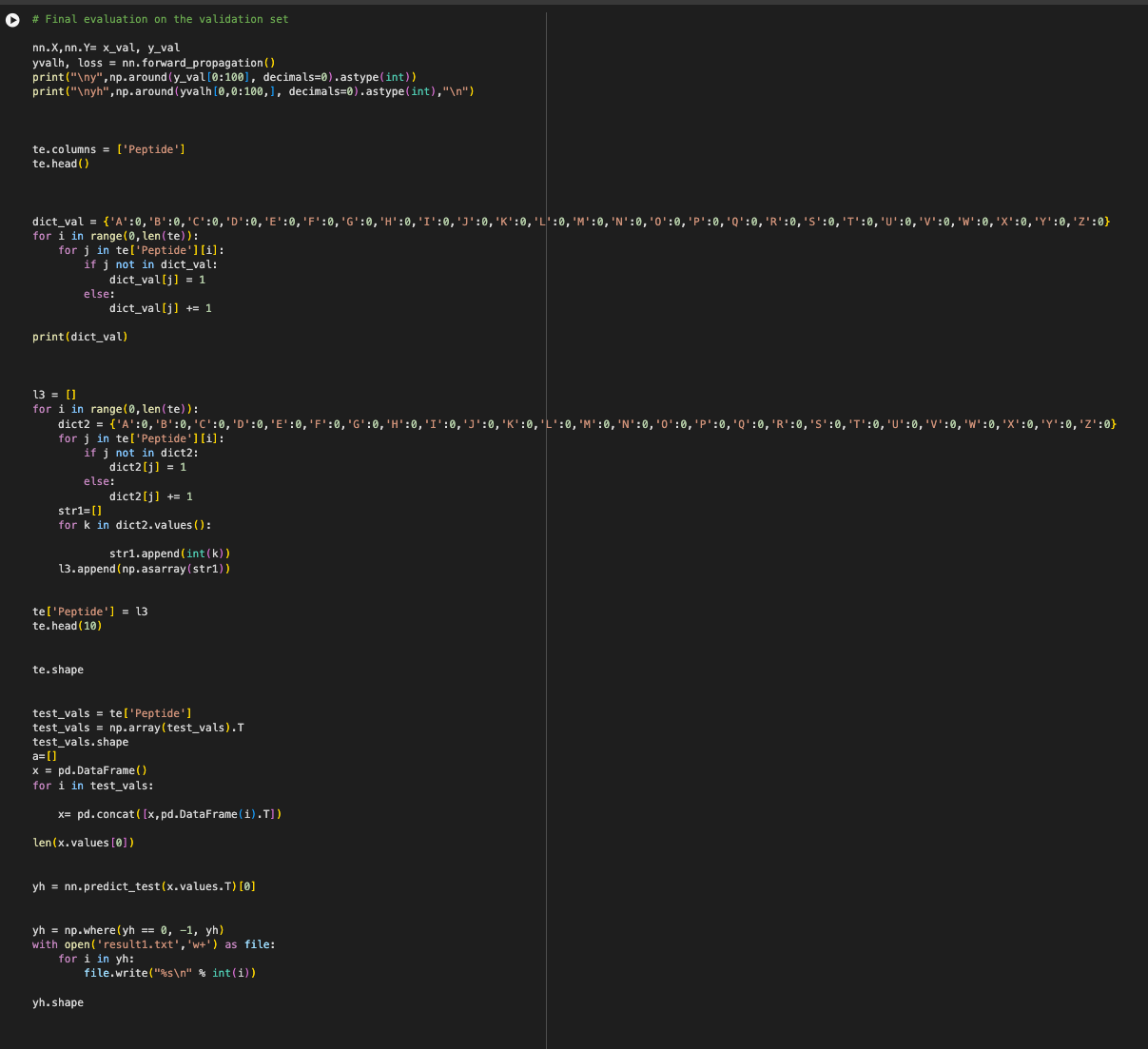


Fig: Code snippet for Final Evaluation

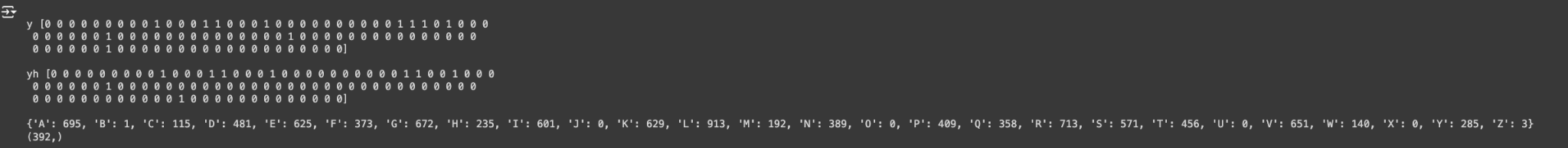


Fig: Output for Final evaluation

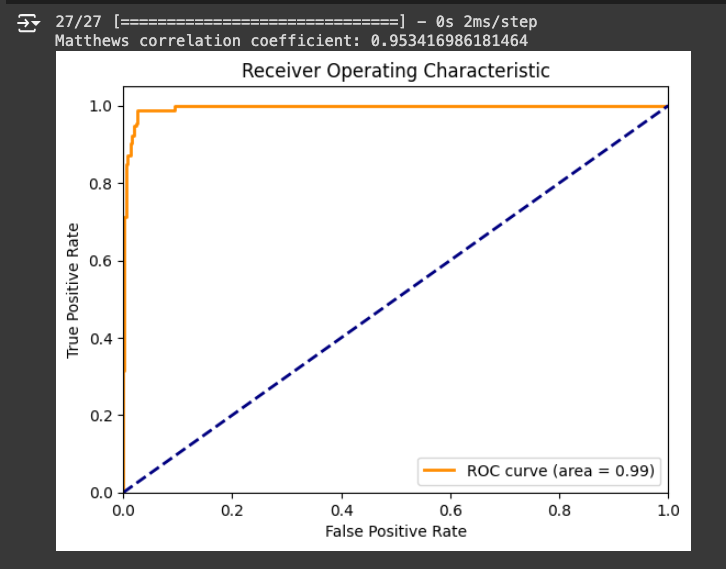


Fig: ROC curve

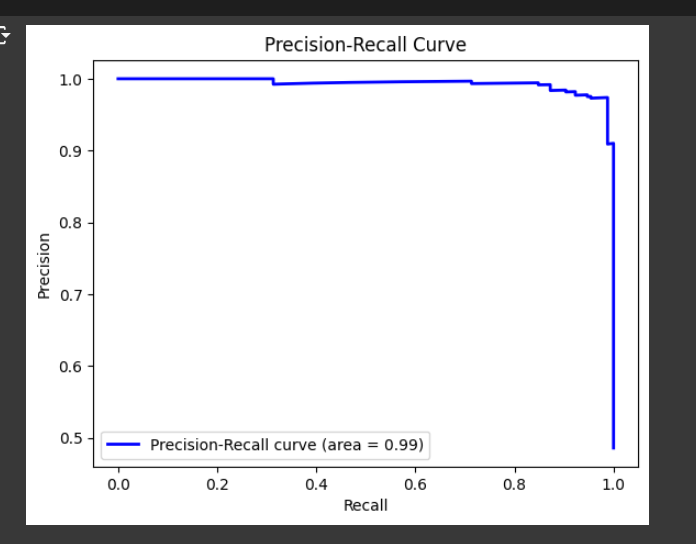


Fig: Precision vs Recall curve

**Conclusions**

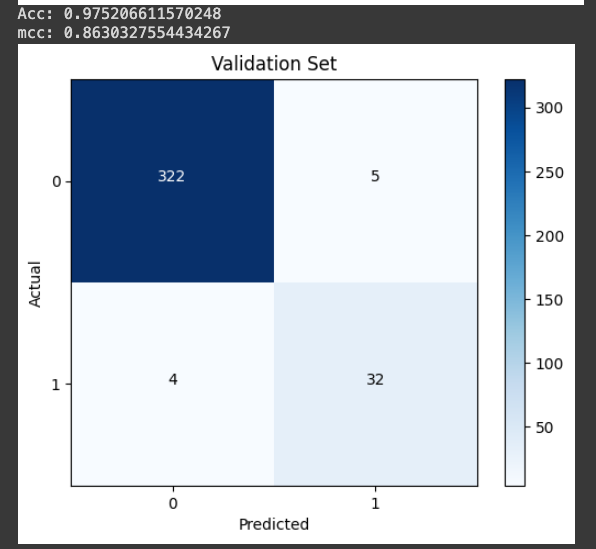
The model achieved an MCC score of 87. It demonstrated good performance on a limited dataset, but updates could be made by employing advanced feature extraction techniques, increasing the complexity of the model, using different optimizers, etc. We could also use more complicated libraries, more split to get better accuracy and experiment a bit more on values of hyperparameters. My model performed better in tensorflow than in python, maybe because of the inbuilt libraries and code/methods passes, with a learning rate set to 0.03 and a threshold set at 0.5. To prevent overfitting, the dataset was split into a training set (70%) and a validation set (30%). Training was performed over 20,000 epochs. Results demonstrated performance, with the model achieving a perfect accuracy of 0.99 on the training data and an impressive 0.9779 on the validation data. Notably, the model displayed minimal false positives and negatives compared to true positives and negatives, indicating high precision and low recall. Subsequently, the most recent data was transformed into a 26 x 1 vector and fed through the network for prediction. 

Fig: The final predicted values

In conclusion, our analysis demonstrated that neural networks can effectively classify peptide sequences into antibiofilm and non-antibiofilm categories. By leveraging the Matthews Correlation Coefficient (MCC) as our evaluation metric, we ensured a balanced and robust measure of model performance, especially in the context of potential class imbalances. The final model achieved a notable level of accuracy and MCC, indicating its efficacy in identifying the desired peptide properties.

The main learnings from this project include the importance of appropriate feature representation for sequence data and the value of using MCC in imbalanced classification problems. Additionally, the project underscored the significance of hyperparameter tuning and model evaluation strategies in developing robust predictive models. This experience highlights the potential of machine learning in bioinformatics and sets a foundation for further exploration into more complex models and feature engineering techniques.

From the analysis, it was concluded that learning models can effectively classify peptides when appropriate feature engineering and data balancing techniques are applied. The model achieved a high MCC, indicating strong predictive performance. Key learnings from this project include the importance of handling class imbalance, the impact of hyperparameter tuning on model performance, and the effectiveness of machine learning for bioinformatics applications.

**References**

1. [**https://www.freecodecamp.org/news/building-a-neural-network-from-scratch/**](https://www.freecodecamp.org/news/building-a-neural-network-from-scratch/)
2. [**https://www.youtube.com/watch?v=w8yWXqWQYmU**](https://www.youtube.com/watch?v=w8yWXqWQYmU)
3. [**https://chat.openai.com/**](https://chat.openai.com/)
4. [**https://www.tensorflow.org/tutorials**](https://www.tensorflow.org/tutorials)
5. [**https://medium.com/hackernoon/building-a-feedforward-neural-network-from-scratch-in-python-d35264571 56b**](https://medium.com/hackernoon/building-a-feedforward-neural-network-from-scratch-in-python-d35264571)
6. [**https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain.**](https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain.)
7. [**https://biodatamining.biomedcentral.com/articles/10.1186/s13040-021-00244-z**](https://biodatamining.biomedcentral.com/articles/10.1186/s13040-021-00244-z)