**README for LipocalinPred: A CNN-LSTM-based Method for Predicting Lipocalin Using Hybrid Features.**

**Author**

Gang Xue1,3, Quan Zou1,4, Wen Zhu1,3\*, Xinyi Liao2, Shaoyou Yu1,3

1School of Mathematics and Statistics, Hainan Normal University; Hainan Haikou 571158, China.

²China Unicom (Hainan) Industrial Internet Co., Ltd, Hainan Haikou 571924, China.

3Key Laboratory of Data Science and Intelligence Education, Hainan Normal University, Ministry of Education, Hainan Haikou 571158, China.

4Yangtze Delta Region Institute (Quzhou), University of Electronic Science and Technology of China, Quzhou 324000, China

\* Corresponding author: Wen Zhu, E-mail: syzhuwen@163.com

**Abstract**

This paper presents a deep-learning algorithm technique based on CNN-LSTM for the identification of lipocalins, called Lipo-CNN\_LSTM. Firstly, the protein sequences underwent feature extraction using three methodologies: K-mer, Composition of K-spaced Amino Acid Pairs (CKSAAP), and Correlation Coefficient Position-Specific Scoring Matrix (CC-PSSM). Then, feature integration techniques are used to fusion features, and then these features are further optimized by Principal Component Analysis (PCA) to obtain important features. In order to leverage the local feature information and the interdependencies of these features fully, we employ an effective approach that integrates the Convolutional Neural Networks (CNN) with a Long Short-Term Memory (LSTM) to detect lipocalins. An accuracy of 0.976 and 0.935 is achieved on the validation set and independent test dataset.

**Data Preprocessing**

Source: This experiment used the dataset from the open-source database UniProt.

1. we firstly collected 307 positive and 307 negative samples from the open-source database UniProt.

2. Use CD-HIT with the cutoff of 40%.

3. We exclude the sequences containing "X," "B," "Z," "J," and "O" non-amino acid characters

Finally, we obtain a processed data sample consisting of 209 lipocalins and 210 non-lipocalins. Furthermore, we further establish an independent dataset of 42 lipocalins and 51 non-lipocalins to validate the model.

**Code and Reproduction Guide**

Individual Data Analysis - Deep Learning.py：This file includes the processes of reading individual data, splitting into training and testing sets, training and running the CNN-LSTM model, and saving the model results.

Combined Data Analysis (Optimal Model).py：This file includes the processes of reading different combination data, splitting into training and testing sets, training and running the CNN-LSTM model, and saving the model results.

Dimensionality Reduction Analysis (Optimal Data and Model).py：This file includes the processes of reading the optimal combination data, performing dimensionality reduction, splitting into training and testing sets, training and running the CNN-LSTM model along with other different classification models, and saving the model results.

Optimizer Comparison (Optimal Data and Model).py: This file includes the processes of reading the optimal combination data, dimensionality reduction, splitting into training and testing sets, training and running the model using different optimizers based on the optimal CNN-LSTM model, and saving the model results.

PCA.py: This Python file is used in the code program for dimensionality reduction.

evaluation.py: This file is a function file used in the code program, designed to evaluate the proposed technique using evaluation methods.

data.fasta： This file contains the raw data, including positive and negative samples of lipocalins.

Independent data in.fasta：This file contains the positive samples of lipocalins.

Independent data no.fasta：This file contains the negative samples of lipocalins.

The runtime environment for the above code is : The training protocol for our model adopts a comprehensive configuration with 50 epochs and a batch size of 8 across parallel processes, using input data dimensions of 1 × 180. For optimization, we employ the SGD optimizer coupled with a weight decay factor of 5e-4 to mitigate overfitting. A momentum of 0.9 is set to accelerate convergence, while the learning rate is configured at 0.01 to enhance computational efficiency. Additionally, we implement Early Stopping—a mechanism that automatically halts training when validation loss plateaus—to ensure our model achieves essential convergence. Experimental implementation was conducted on a Windows system utilizing TensorFlow 2.9.0, Keras 2.9.0, Python 3.8.19, and CUDA 10.1.0. The hardware setup features an Intel® Core™ E5-2680 v3 CPU operating at 2.50 GHz, complemented by an NVIDIA GeForce RTX 2070 GPU.