MLBA Assignment 2
Documentation

Group 18: Meetakshi Setiya (2019253), Shelly Gupta (2019108), Garmit Pant (2019240)

This program classifies peptides with variable lengths into binary class. We have used amino acid composition to generate features for the peptide strings to feed the dataset to a machine learning model. The final predictions file contains class probabilities for the test dataset. Below is a more detailed description of the code and how to run it.

❖ Inside the zipped folder

Please find the following files in the zipped folder

- ➔ The README as a pdf file - Group_18_README.pdf
- ➔ The python source code - Group_18_Code.py
- ➔ The training dataset from kaggle - train.csv
- ➔ The testing dataset from kaggle - test.csv
- ➔ The final predictions file submitted on kaggle renamed to Group_18_Predictions_Output.csv

❖ Description

Given below are the steps we used to arrive at the final version of the machine learning model we used in the assignment.

1. Feature Engineering

We tried using one-hot encoding for feature generation. We also tested a neural network and fed the peptide sequences as strings as the input. Lastly, we tried the amino acid composition technique for feature engineering and it gave the best results.
In the composition of a sequence, we calculated the frequency of each of the 20 amino acids present in the sequence divided by the length of the sequence.

We stored all the 20 amino acids in the variable 'amino_acids'. The method `composition()` takes a peptide sequence as input and calculates the ratio of the frequency of each amino acid in the inputted sequence and the length of the sequence and appends the calculated value in the list of that particular amino acid.

Composition is applied on train_data_seq and test_data_seq (containing the respective training and testing peptide sequences) separately.

`train_data_comp`: a variable that stores the amino acid composition of training sequences as a dataframe

`test_data_comp`: a variable that stores the amino acid composition of testing sequences as a dataframe

2. Model Selection

We experimented with Support Vector Classifier, Extra Tree Classifier, Random Forest Classifier, Convolutional Neural Network, XGBoost classifier and Catboost Classifier. The Hypertuned Random Forest Classifier gave the best accuracy. Ultimately we decided to stack the Random Forest Classifier and Logistic Regression Classifier, which gave us our best score. The same has been implemented in the attached source file. Given below are the parameters for the classifiers:

Best model: StackingClassifier

For Stacking Classifier
Base Estimator (estimators): RandomForestClassifier, LogisticRegression
Final estimator (final_estimator): LogisticRegression
Cross-validation splitting strategy to train final estimator (cv): 5-fold Cross Validation
Rest of the parameters were left on default settings.

For RandomForestClassifier (Base Estimator)
Number of trees in the forest (n_estimators): 700
Use out-of-bag samples to estimate the generalization score (oob_score): True
Use all processors (n_jobs): -1,
number of features to consider when looking for the best split (max_features): 'sqrt'
Rest of the parameters were left on default settings.

For LogisticRegression (Final Estimator)
All parameters were left on default settings.

3. Scoring and evaluating the model

Repeated stratified k-fold cross validation method was used on the training dataset to cross validate the model. The value of k was set to 10 and the number of repeats was set to 5. The motivation to use cross validation was to ensure that the model is not overfitted. Given below the parameters used for the cross validator:

Number of folds (n_splits): 10
Number of time Cross Validation is repeated (n_repeats): 5

The cross validator was scored based on 'Accuracy'.
We have displayed the accuracy scores obtained from cross validation, the mean of the scores and their standard deviation.

4. Fitting the model on the entire training dataset

Once the model was tuned to our satisfaction and gave good cross validation scores, we fitted it on the entire training dataset (`train_data_comp`) to get it ready to give predictions on the testing dataset (`test_data_comp`).

5. Making predictions

After training the model on the entire training dataset, we used `predict_proba()` to predict class probabilities for the test data (`test_data_comp`). The class probabilities and corresponding indices were made into a dataframe and exported as a CSV. The CSV will be present in the same directory the program is in and will be named Group_18_Predictions_Output.csv. Prediction file submitted on Kaggle is called predictionsrflroptimized.csv

6. Final Results

Public leaderboard score: 0.78027
Private leaderboard score: 0.76301

About the Python Program:

❖ Input files

The program takes paths to the training data as the user input. We have used the peptide sequences' files provided on kaggle as our training and testing data to find our predictions file Group_18_Predictions_Output.csv.
For training data: train_data.csv
For testing data: test_data.csv

❖ Output file

The program outputs a csv file Group_18_Predictions_Output.csv containing the predicted class probabilities for the binary classes(0,1) for the testing data.
The name of the prediction file submitted on kaggle was different though.
Prediction file submitted on Kaggle: predictionsrflroptimized.csv.
We have also included the submitted prediction file renamed to Group_18_Predictions_Output.csv in the zip folder.

❖ Steps to run the Python program

Make sure that all dependencies and libraries used by this program are installed on your system. A list of the required libraries are:

➔ Numpy
➔ Pandas
➔ Re

➔ Sklearn
➔ Collections

Use the following steps to run the python file Group_18_Code.py containing the code used for training and prediction from the terminal:

1. Run the command- `python3 Group_18_Code.py`
2. The program will ask to input the train data file path. Please provide a path to the csv training dataset containing peptide sequences and their binary class labels.
3. The program will ask to input the test data file path. Please provide a path to the csv testing dataset.
4. After the program is completed running, it will create a CSV output file *Group_18_Predictions_Output.csv* containing the predictions corresponding to the inputted testing dataset csv file. The program will also display the cross validation scores and their mean and standard deviation.

Sample output on the Terminal:

```
Accuracy scores for the model from k-fold cross validation:
 [0.746875   0.7390625  0.7421875  0.7421875  0.73239437 0.74021909
 0.72926448 0.74491393 0.72143975 0.73865415 0.71875    0.7328125
 0.75625    0.71875    0.73239437 0.73239437 0.71205008 0.73865415
 0.75117371 0.74647887 0.690625   0.725      0.75       0.74375
 0.72926448 0.74178404 0.7543036  0.72143975 0.75117371 0.72769953
 0.7703125  0.7359375  0.7171875  0.7234375  0.72613459 0.74491393
 0.72300469 0.70422535 0.76525822 0.74960876 0.725      0.6765625
 0.746875   0.7578125  0.7370892  0.73865415 0.74491393 0.72300469
 0.75743349 0.75899844]

Mean accuracy score: 0.736

Standard deviation of accuracy score: 0.018
```

❖ Miscellaneous

Submission scores from other models:

1. RandomForestClassifier:
   Public leaderboard score: 0.77856
   Private leaderboard score: 0.76473
2. Stacking Random Forest Classifier, CatBoost Classifier and LogisticRegression:
   Public leaderboard score: 0.78000
   Private leaderboard score: 0.76301