Instructions

Introduction

This project focuses on DNA Binding Protein classification using various machine learning models. Our goal is to explore different feature extraction methods and classifiers to accurately distinguish between binding and non-binding proteins. By both biological knowledge (amino acid distribution, term frequency inverse document term, physicochemical properties) and natural language processing techniques (embedding, integer encoding) with both traditional MLs (Logistic Regression, Naive Bayes, KNN, Support Vector Machine, Decision Tree, Random Forest) and Deep Learning models (CNN, ProtCNN, Large Language Model), we aim to identify the most effective approach for protein classification.

Dataset and Files

We are using the dataset titled `DNA Binding Protein`. In the Code Folder, here is the order of the code files we intend for you to access:

1. `Exploratory Data Analysis.ipynb`
2. `AAD.ipynb` (Amino Acid Distribution & TF-IDF)
3. `TF-IDF.ipynb` (Term Frequency-Inverse Document Frequency)
4. `Physicochemical Properties.ipynb` (Physicochemical Properties)

5. `pseAAC.ipynb` (Pseudo Amino-Acid Composition)
6. `CNN.ipynb` (Convolutional Neural Network)
7. `ProtBert.ipynb` (Large Language Model)
8. `Comparison Plots between Models`

In the Model & DataSet folder, below are 4 datasets we include:

1. `Train.fasta`: original training dataset
2. `Test.fasta`: original testing dataset
3. `Train_valid.fasta`: cleaned training dataset containing only sequences with common amino acids
4. `Test_valid.fasta`: cleaned testing dataset containing only sequences with common amino acids
5. `DNA_Test.csv`: cleaned testing dataset containing physicochemical properties
6. `DNA_Train.csv`: cleaned testing dataset containing physicochemical properties

Below are the saved file for different models:

1. `LR_Amino Acid Distribution_Full.joblib`: Logistic Regression classifier with full 20 amino acid distribution as features
2. `LR_Amino Acid Distribution_Selected.joblib`: Logistic Regression classifier with selected 20 amino acid distribution as features
3. `NB_Amino Acid Distribution_Full.joblib`: Naive Bayes classifier with full 20 amino acid distribution as features
4. `NB_Amino Acid Distribution_Selected.joblib`: Logistic Regression classifier with selected 20 amino acid distribution as features
5. `KNN_Amino Acid Distribution_Full.joblib`: K-Nearest Neighbour classifier with full 20 amino acid distribution as features
6. `KNN_Amino Acid Distribution_Selected.joblib`: K-Nearest Neighbour classifier with selected 20 amino acid distribution as features
7. `DT_Amino Acid Distribution_Full.joblib`: Decision Tree classifier with full 20 amino acid distribution as features
8. `DT_Amino Acid Distribution_Selected.joblib`: Decision Tree classifier with selected 20 amino acid distribution as features

9. `RF_Amino Acid Distribution_Full.joblib`: Random Forest classifier with full 20 amino acid distribution as features

10. `RF_Amino Acid Distribution_Selected.joblib`: Random Forest classifier with selected 20 amino acid distribution as features

11. `SVM_Amino Acid Distribution_Full.joblib`: Support Vector Machine classifier with full 20 amino acid distribution as features

12. `SVM_Amino Acid Distribution_Selected.joblib`: Support Vector Machine classifier with selected 20 amino acid distribution as features

13. `LR_Physicochemical_Properties.joblib`: Logistic Regression classifier with physicochemical properties as features

14. `NB_Physicochemical_Properties.joblib`: Naive Bayes classifier with physicochemical properties as features

15. `KNN_Physicochemical_Properties.joblib`: K-Nearest Neighbour classifier with physicochemical properties as features

16. `DT_Physicochemical_Properties.joblib`: Decision Tree classifier with physicochemical properties as features

17. `RF_Physicochemical_Properties.joblib`: Random Forest classifier with physicochemical properties as features

18. `SVM_Physicochemical_Properties.joblib`: Support Vector Machine classifier with physicochemical properties as features

19. `LR_pseAAC.joblib`: Logistic Regression classifier with pseudo-Amino Acid Composition as features

20. `NB_pseAAC.joblib`: Naive Bayes classifier with pseudo-Amino Acid Composition as features

21. `KNN_pseAAC.joblib`: K-Nearest Neighbour classifier with pseudo-Amino Acid Composition as features

22. `DT_pseAAC.joblib`: Decision Tree classifier with pseudo-Amino Acid Composition as features

23. `RF_pseAAC.joblib`: Random Forest classifier with pseudo-Amino Acid Composition as features

24. `SVM_pseAAC.joblib`: Support Vector Machine classifier with pseudo-Amino Acid Composition as features

25. `CNN1.h5`: CNN model using Embedding technique

26. `CNN2.h5`: CNN model using Amino acid integer encoding

27. `ProtCNN1.h5`: ProtCNN model using Embedding technique
28. `ProtCNN2.h5`: ProtCNN model using Amino acid integer encoding
29. `trained_model`: This folder includes the trained model's weights and architecture configuration. It contains files like `pytorch_model.bin` (the model weights) and `config.json` (model configuration). These files are necessary for loading and using the trained model later without retraining.
30. `trained_tokenizer`: This folder holds the tokenizer files required to preprocess your input data the same way as during training. It includes `vocab.txt` (the vocabulary) and `tokenizer_config.json` (tokenizer settings).

Ensure that the training and testing data, `Train.fasta` and `Test.fasta` are present in the same root directory as the aforementioned files. Moreover, the filtered `Train_valid.fasta` and `Test_valid.fasta` are also in the same folder. Those contain filtered datasets for protein sequences that contain common amino acids.

In each subsequent section, we outline the required libraries and any additional requirements to run the code.

Installing Libraries

Ensure that the file `requirements.txt` is present in the root directory. Run the following command in your terminal to install all required libraries:

```
pip install -r requirements.txt
```

It should install the following libraries not included in a fresh installation of Python:

- biopython
- imbalanced-learn
- propy3
- scikit-learn
- tensorflow
- tqdm
- pandas
- numpy
- keras
- seaborn

- kat
- statsmodels

Exploratory Data Analysis

In this file, we attempted to make sense of the data. We did so by reading and processing textual data, then plotting simple graphs to understand what we are working with. In particular, we explored the composition of class 1 and class 0 amino acids, explored amino acid distributions, analysed protein sequence length distribution.

Running Code

Code may be run in a Jupyter notebook environment without issues. If run in a Google Colab environment, ensure training and testing data is uploaded and file path is correctly adjusted.

Amino Acid Frequency Distribution and TD-IDF

In this file, we used biological knowledge to extract biological features of amino acid frequency distribution and TF-IDF, and attempted to classify the proteins using traditional Machine Learning models.

Running Code

Code may be run in a Jupyter notebook environment without issues. If run in a Google Colab environment, ensure training and testing data is uploaded.

Saving/Loading Model

To load the saved model, you need to import the load function from the joblib library and call the load function with the file name. `from sklearn.externals import joblib`
`model = joblib.load(filename)`

Physicochemical Properties

In this file, we explored how different forms of physicochemical properties of an amino acid, which includes hydrophobicity, net charge, etc. may affect the classification of a protein using traditional Machine Learning models.

Running Code

Code may be run in a Jupyter notebook environment without issues. If run in a Google Colab environment, ensure training and testing data is uploaded.

Saving/Loading Model

To load the saved model, you need to import the load function from the joblib library and call the load function with the file name. `from sklearn.externals import joblib`
`model = joblib.load(filename)`

pseAAC

In this file, we explored how standard forms of amino acid distribution may be improved by incorporating additional information such as patterns in sequences and physicochemical properties may affect the classification of protein using traditional Machine Learning models.

Running Code

Code may be run in a Jupyter notebook environment without issues. If run in a Google Colab environment, ensure training and testing data is uploaded. Warning: estimated time to run the code is around 22 hours.

Saving/Loading Model

To load the saved model, you need to import the load function from the joblib library and call the load function with the file name. `from sklearn.externals import joblib`
`model = joblib.load(filename)`

CNN

In this file, we explored using Convolutional Neural Networks as a classifier using Natural Language Processing techniques, namely embedding and amino acid integer encoding. Notably, we are using `Train_valid.fasta` and `Test_valid.fasta` here as training and testing dataset here.

Running Code

Code may be run in a Jupyter notebook environment without issues. If run in a Google Colab environment, ensure training and testing data is uploaded. Switching to GPU is recommended for faster training time. Warning: running on CPU might take total 24 hours while using GPU might take around 2 hours.

Saving/ Loading Model

The model may be saved and loaded using a H5 file. The H5 file includes everything about the model, including:

- Model weights
- Model architecture
- Model compilation details (loss and metrics)
- Model optimizer state

For later use, load and use the model directly without having to re-compile and train the model following this syntax:

```
from keras.models import load_model
   model = load_model(file_path)
```

LLM

In this file, we explored using Large Language Models as a classifier. In particular, we used a pretrained mode, ProtBert.

Running Code

Code may be run in a Jupyter notebook environment without issues. If run in a Google Colab environment, ensure training and testing data is uploaded. Warning: training might take 2.5 hours.

Saving/Loading Model

You can load in this way:

```
model = BertModel.from_pretrained(filename")
```