# The Prediction of Big InterPro Features using Small Features

**Student**: Maryam Jalali

**Student ID: 443224**

**Study**: Data Science for Life Sciences

## Introduction

The purpose of the project is to design sci-kit machine learning standards to anticipate the operation of a protein utilizing InterPro attributes. The dataset includes a protein with related InterPro classes that represent the function. The predicted InterPro class is defined as follows:

1. Its length covers more than 90 percent of the protein's sequence length.
2. It covers the largest length of the whole sequence.

## Material and Methods

### Dataset

The dataset is the bacilli dataset. It is delivered by the Programming 3 course. A table containing 15 columns: protein accession identifier, sequence MD5 digest, sequence length, analysis medium, signature accession, signature description, start location, stop location, score, the status of the match, date of the run, InterPro accession annotation, InterPro description annotation, GO annotation, and pathway annotation.  The InterPro scan outcomes deliver information about a protein's function.

### Pipeline

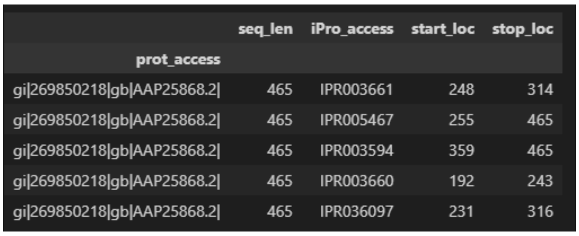
To examine the data I use Python with DASK for instructing the machine learning instances. Pyspark is utilized for analyzing the dataset.

## Results

The dataset holds some noise. Accordingly, the dataset is cleaned before executing the Machine Learning models. Invalid entries are withdrawn. These are InterPro acquisition explanations with the value '-'. It indicates that no function was seen for the related protein. Consequently, rows with a dash in the InterPro accession column are removed.

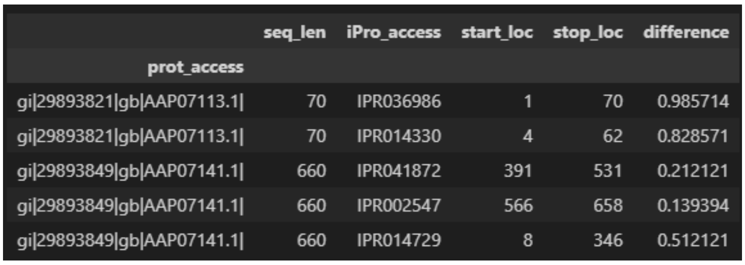
After clearing the noise, we need to develop the small elements.

First, simply four columns are necessary to generate the features: the sequence length of the InterPro feature, InterPro accession, start location of the protein, and stop location of the protein.



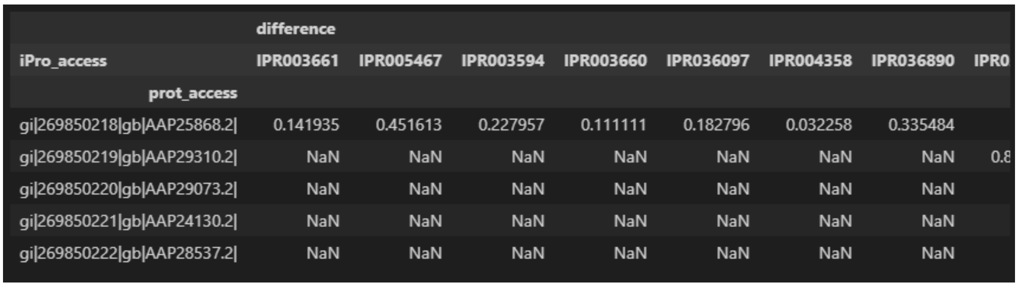
| | seq_len | iPro_access | start_loc | stop_loc |
|---|---|---|---|---|
| prot_access | | | | |
| gi\|269850218\|gb\|AAP25868.2\| | 465 | IPR003661 | 248 | 314 |
| gi\|269850218\|gb\|AAP25868.2\| | 465 | IPR005467 | 255 | 465 |
| gi\|269850218\|gb\|AAP25868.2\| | 465 | IPR003594 | 359 | 465 |
| gi\|269850218\|gb\|AAP25868.2\| | 465 | IPR003660 | 192 | 243 |
| gi\|269850218\|gb\|AAP25868.2\| | 465 | IPR036097 | 231 | 316 |

We position the protein accession identifier as the index and choose the four columns. Secondly, we figure out how large the InterPro attribute is close to the protein. We do this step with the following formula: (*stop location – start location) / sequence length* and assign the results to the *difference* column.



| | seq_len | iPro_access | start_loc | stop_loc | difference |
|---|---|---|---|---|---|
| prot_access | | | | | |
| gi\|29893821\|gb\|AAP07113.1\| | 70 | IPR036986 | 1 | 70 | 0.985714 |
| gi\|29893821\|gb\|AAP07113.1\| | 70 | IPR014330 | 4 | 62 | 0.828571 |
| gi\|29893849\|gb\|AAP07141.1\| | 660 | IPR041872 | 391 | 531 | 0.212121 |
| gi\|29893849\|gb\|AAP07141.1\| | 660 | IPR002547 | 566 | 658 | 0.139394 |
| gi\|29893849\|gb\|AAP07141.1\| | 660 | IPR014729 | 8 | 346 | 0.512121 |

After that, we choose only the InterPro accession and the difference and pivot the table on the InterPro accession, so all the cells will include the difference.

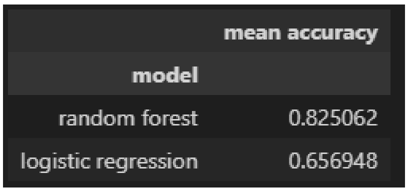


| | difference | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| iPro_access | IPR003661 | IPR005467 | IPR003594 | IPR003660 | IPR036097 | IPR004358 | IPR036890 | IPR0 |
| prot_access | | | | | | | | |
| gi\|269850218\|gb\|AAP25868.2\| | 0.141935 | 0.451613 | 0.227957 | 0.111111 | 0.182796 | 0.032258 | 0.335484 | |
| gi\|269850219\|gb\|AAP29310.2\| | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 0.8 |
| gi\|269850220\|gb\|AAP29073.2\| | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| gi\|269850221\|gb\|AAP24130.2\| | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| gi\|269850222\|gb\|AAP28537.2\| | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |

To get the labels to anticipate, we get the highest value. The features will be converted into 1s and 0s.

Now we can see the labels and the features. We utilize a label encoder to encode the labels. Then we divide the data into a train and test set.

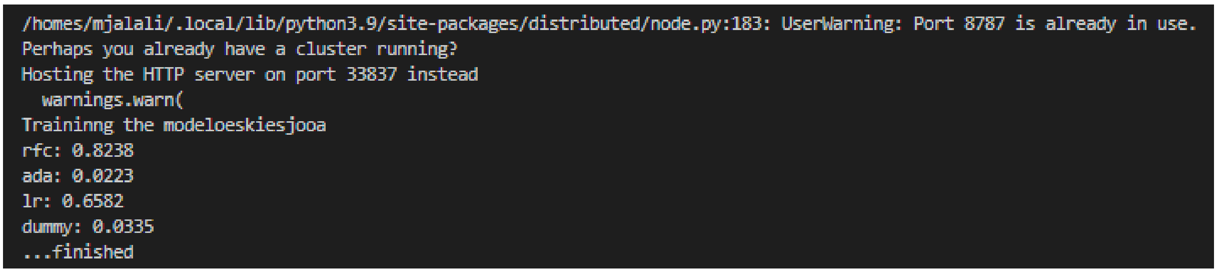
Two forms are being trained on the dataset. An accidental random forest, as it often adjusts very well to data and multi-class logistic regression, is a good baseline and a usually operating algorithm. The outcomes are as follows:



| | mean accuracy |
|---|---|
| model | |
| random forest | 0.825062 |
| logistic regression | 0.656948 |

We can see that random forest functions exceptionally well for a complex dataset.

The dummy classifier predicts the label that is most frequent. So it will always predict the label with most instances. It is a handy baseline model. Random Forest performs well as it splits the features into random subsets and trains on that subset. It averages the variances of all the trees. So in contrast to the dummy classifier, Random Forest tries to associate the features with the labels. Logistic regression is another technique that estimates the probability of a label occurring. It assumes a lack of influential outliers, while the random forest does not make such assumptions. Therefore the random forest may perform better



```
/homes/mjalali/.local/lib/python3.9/site-packages/distributed/node.py:183: UserWarning: Port 8787 is already in use.
Perhaps you already have a cluster running?
Hosting the HTTP server on port 33837 instead
  warnings.warn(
Traininng the modeloeskiesjooa
rfc: 0.8238
ada: 0.0223
lr: 0.6582
dummy: 0.0335
...finished
```

## Future directions

There are some modifications for the future:

- Employ a more applicable machine learning model that surpasses most other machine learning models like XGBoost.

- Employ deep learning methods and compare the machine learning outcomes with them.

- Use hyperparameter tuning to discover the optimal model.