MACHINE LEARNING BASED APPROACH FOR FUNCTIONAL ANNOTATION OF LYTIC POLYSACCHARIDE MONOOXYGENASES

Enrolment No – **151507**

Name of the student – **Pulkit Anupam Srivastava**

Name of Supervisor – **Dr. Ragothaman M. Yennamalli**

May 2019

Submitted in partial fulfillment of the requirement

for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

BIOINFORMATICS

DEPARTMENT OF BIOTECHNOLOGY AND BIOINFORMATICS,

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT, SOLAN 173234, HIMACHAL PRADESH, INDIA

ACKNOWLEDGEMENT

"Our dreams can come true if we have the courage to pursue them."

This work presents a part of my dream that would not be possible to attain without constant support and encouragement from many panjandrums who influenced my actions, thoughts and behavior during the course of this project.

Firstly, I would like to extend my sincere gratitude to my final year project supervisor **Dr. Ragothaman M. Yennamalli** for providing me with an opportunity to conduct research work in my area of interest. Also, his continuous guidance, efforts, and invertible suggestions throughout the duration of the project was blessing in disguise.

I would like to extend my heartiest gratitude to **Dr. Jayashree Ramana** whose teaching greatly influenced me and increased my interest in the field of machine learning. This project and all my future investigations in the field of machine learning will be a result of hard work she endowed in me to build conceptual understanding of the subject. She will always be a great inspiration in my life and a person whom I would look as an idol.

I give my greatest acknowledgement to **my Parents** for all round support during my studies. The endless sacrifices from my parents have made me what I am today.

I also thank my friends **Parul, Anmol**, **Parushi, Shweta** and my seniors **Pallavi Raj**, **Gorky**, **Abhishek Guleria**, **Preeti**, **Ankita Sharma**, **Rohit Shukla**, **Arvind, Monika, Siddhant Kalra, Shubham Mittal, Rahul Pramjeet,** and **Ankush Bansal** for their encouragement and constant support. I am lucky enough to have friends like them who stood beside me and motivated me during my studies and made this work possible.

DECLARATION BY THE STUDENT

I hereby declare that the work reported in the B.Tech. project report entitled "**Machine Learning based approach for functional annotation of Lytic Polysaccharide Monooxygenases**" submitted at Jaypee University of Information Technology, Waknaghat, India, is an authentic record of my work carried out under the supervision of Dr. Ragothaman M. Yennamalli. I have not submitted this work elsewhere for any other degree or diploma.

(Signature of the Student)

Pulkit Anupam Srivastava (151507)

Department of Biotechnology and Bioinformatics,

Jaypee University of Information Technology,

Waknaghat, India

Date:

CERTIFICATE

This is to certify that project report entitled "**Machine Learning based approach for functional annotation of Lytic Polysaccharide Monooxygenases**", submitted by Pulkit Anupam Srivastava is in its partial fulfillment for the award of degree of Bachelor of Technology in Bioinformatics to Jaypee University of Information Technology Waknaghat, Solan (H.P.), India is an authentic record of candidate's own work carried out by him under my supervision.

This work has not been submitted partially or fully to any other university or institution in order to achieve any award or any other degree.

Dr. Ragothaman M. Yennamalli

Assistant Professor (Grade II),

Department of Biotechnology and Bioinformatics,

Jaypee University of Information Technology,

Waknaghat, Dist. Solan (173234), Himachal Pradesh, India

TABLE OF CONTENTS

ABSTRACT 1

INTRODUCTION 2

General Background 2

Hypothesis 4

MATERIALS AND METHODS 5

Method overview 5

Dataset collection and cleaning 5

Positive and negative set construction 5

Sequence based feature generation 6

Model generation for each feature 8

Consensus approach 10

Validation of models 10

RESULTS AND DISCUSSION 11

CONCLUSION 13

REFERENCES 14

APPENDIX 17

List of descriptors having a Gini index of ≥ 0.6 in each feature-set for AA9. 17

List of descriptors having a Gini index of ≥ 0.6 in each feature-set for AA10. 18

Python script to filter out noise from sequence data 19

Python script to generate training and test dataset (AA10 family) 20

R script to generate protein sequence feature-sets for training dataset (AA10 family) 21

R script to generate protein sequence feature-sets for test dataset (AA10 family) 22

Python script to generate optimized models for each feature using modified Huber as loss function in SGD (AA10 family) 23

Python script to generate optimized models for each feature using hinge as loss function in SGD (AA10 family) 25

Python script to generate optimized models for each feature using log as loss function in SGD (AA10 family) 27

Python script to generate optimized models for each feature using radial basis function as kernel in SVC (AA10 family) 29

Python script to generate optimized model for each feature using feature based neural network (AA10 family) 31

Python script to generate optimized model using long short-term memory units (AA10 family) 33

Python script uploaded on GitHub for Data Cleaning 35

Python script uploaded on GitHub for feature extraction 36

Python script uploaded on GitHub for feature-based neural network prediction 37

Python script uploaded on GitHub for long short-term memory (LSTMs) based prediction 39

LIST OF FIGURES

Figure 1. Schematic workflow implemented to develop machine learning method for prediction of LPMO family (AA9 and AA10).

Figure 2. Schematic workflow for prediction of potential LPMO sequence using protein sequences in fasta format.

Figure 3. Framework of the neural network implemented in both approaches.

LIST OF TABLES

Table 1. Distribution of sequences (positive and negative set) into training and test set for construction AA9 and AA10 models.

Table 2. Performance of various machine learning algorithms and dbCAN2 for AA9 and AA10 models on validation and independent set.

ABSTRACT

Lytic polysaccharide monooxygenases (LPMO), a family of copper-dependent oxidative enzymes, boost the degradation of crystalline polysaccharides, such as cellulose and chitin, by breaking an internal glycosidic bond thereby exposing the polymer for further degradation. Recently, the sequence diversity of LPMOs has increased significantly, with newer sequences identified in organisms across the tree of life. Accurate functional assignment of yet unknown sequences into LPMOs family is an important step towards production of enzymatic mixture adept at efficiently degrading recalcitrant polysaccharides. While, multiple experimental methods are used for accurate identification of LPMOs, a computational method that can accurately classify sequences into LPMOs is needed to match the sequences generated. Thus, to screen potential LPMOs, we developed a machine learning based tool that employs two different approaches to functionally classify a given protein sequence(s) as belonging to LPMO family or not. As proof of concept, we worked on classifying sequences belonging to either AA9 or AA10 family of LPMO. The first approach uses traditional neural network based prediction after calculating sequence features. The second approach uses bi-directional long short-term memory (LSTM) units, a type of recurrent neural network, which extracts important features directly from sequence and utilizes an internal state, i.e., memory, to process input data. The optimized model trained from both the approaches was cross validated on a validation set to test the precision and recall. Specifically, feature-based traditional neural network approach was able to correctly discriminate AA9 LPMO sequences from non-AA9 LPMOs with a recall of 96.4%, precision of 100% and AA10 LPMO sequences from non-AA10 LPMOs with a recall of 86.9%, precision of 100%. On the other hand, LSTM had a recall of 93.4%, precision of 90.7% on AA9 dataset and recall of 91.7%, precision of 89.6% on AA10 dataset. Further, we validated our method with an independent benchmark set of LPMO sequences, where we observed significant precision and recall compared to dbCAN2, an existing HMM-profile based CAZyme predicting tool. The working code can be freely found at: https://github.com/PulkiD/PreDSLpmo.

Keywords: Lytic polysaccharide monooxygenases, deep neural network, long short-term memory, proteome.

INTRODUCTION

General Background

Biofuels are mooted as a sustainable source of energy that can meet the increasing energy demands of developing countries in Asia [1]. One of the steps taken is transitioning from fossil fuels (fast depleting and causing pollution) to an eco-friendly environment through production and usage of biofuels via industrial biotechnology methods. These methods explore various approaches to degrade renewable plant biomass which consist of mixed sugars and polysaccharides. However, one of the major bottlenecks is the conversion of crystalline form of complex polysaccharides like cellulose and starch to its amorphous form. This limitation can be imparted to recalcitrant nature of these polymers towards various physical, chemical, physio-chemical, thermal, and enzymatic processes thereby making them less cost effective for industrial purpose [2]. Hence, enzymes that can enhance the activity of biomass degradation offer great promise in improvement of industrial bioethanol production.

Lytic polysaccharide monooxygenases (LPMOs), a class of enzymes identified in last decade or so, has ability to directly disrupt glycosidic bond present on crystalline surfaces i.e., the region inaccessible to conventional hydrolytic enzymes have attracted scientific community [3]. Hence, makes it more attractive to the scientific community. Originally, LPMOs were originally classified as carbohydrate binding module family 33 (CBM33) in bacteria and glycoside hydrolase family 61 (GH61) in fungi [4]. However, in recent years, carbohydrate-active enzymes (CAZy) have reclassified them into auxiliary activity (AA) class of enzymes [5]. Thus, based on the substrate it acts on, LPMOs are classified into six families of AA (AA9, AA10, AA11, AA13, AA14, and AA15). Further, they are classified into different types (Type1, Type2, and Type3) on the basis of carbon they attack in glycosidic bond. Detailed explanations on the LPMOs are reported in [3, 4, 6, and 7].

Currently, LPMOs have been identified in bacteria, fungi, viruses, and eukaryotes. Besides, new LPMO sequences are being identified from already known organisms' proteomes and also from new proteomes deposited at NCBI. For example, LPMO reported from *Vibrio cholera, Bacillus* *anthracis, Bacillus cereus, Serratia marcescens, Listeria monocytogenes, Pseudomonas aeruginosa*, and *Enterococcus faecalis* [8, 9, 10, 11, and 12]. However, lack of an automated method from CAZy server [5] to identify potential LPMOs in new proteomes and metagenomes presents a major challenge.

To address the above challenge, Xu and his teammates integrated three tools in a meta-server called dbCAN2 [13] (http://cys.bios.niu.edu/dbCAN2) for automated annotation of CAZymes. The three tools works by running HMMER [14], Hotpep [15], and DIAMOND [16] against dbCAN hidden Markov model database, CAZyme pre-annotated sequence database, and CAZyme short peptide database, respectively.

Hypothesis

Since LPMOs' discovery almost a decade ago, the usual method of characterization is via experimental methods, such as High-Performance Anion-Exchange Chromatography with Pulsed Amperometric Detection (HPAE-PAD) and/or colorimetric assays. The limitations of such experimental methods are high-cost and time. Hence, automated functional annotation using a computational approach that is able to identify LPMO sequences correctly can aid in identification of sequences from proteomic or metagenomic sequence data for further characterization. Hence, using various supervised machine learning algorithm and deep learning we have tried to develop a tool for functional annotation of sub-family of LPMOs.

MATERIALS AND METHODS

Method overview

The first approach developed in our study employs a traditional neural network for classification of sequences after calculating sequence features. The features are conjoint triad, di-peptide composition, tri-peptide composition, Moran autocorrelation, Geary autocorrelation, and Normalized Moreau-Broto autocorrelation. The second approach used in our study is bi-directional long short-term memory (LSTM), a type of deep learning method that extracts important features from sequences itself by utilizing an internal state (i.e., memory) to process input data. The optimized model trained from both the approaches was cross validated on a validation set as well as on an independent benchmark set to test the precision and recall. Figure 1 illustrates the workflow used in the study.

Dataset collection and cleaning

AA9 and AA10 sequences were downloaded from CAZy [5] (accessed on 3rd August, 2018). In order to construct a negative set we took sequences from other family of AAs listed in Table 1. Sequences downloaded to construct positive set and negative set were removed if they had unrecognized residues labelled as "X" in their sequence and or were partial in nature.

In AA10 model, we used CD-HIT [17], an online tool to cluster and compare large sequences, to remove redundant sequences for building positive dataset.

Validation set for AA9 and AA10 models was constructed by downloading new sequences reported to be AA9 and AA10 by CAZy [5] as on 20th November, 2018. The number of new AA9 and AA10 sequences were 44 and 505, respectively

Positive and negative set construction

To construct the training and test set for further steps we followed the 60:40 ratio. We used 60 % of the total number of sequences from each positive and negative set to construct a training set. And, 40 % of the total number of sequences from each positive and negative set was used to construct a test set. Further, we tried to keep the ratio of positive and negative set to 1:1 ratio in both training and test set.

Sequence based feature generation

Both training and test set sequences were independently given as an input to in-house developed R script that made use of ProtR package [18] to generate sequence based physiochemical and structural protein features. They were Di-peptide, Tri-peptide, Conjoint Triad, Moran autocorrelation (AC), Geary (AC), and Normalized Moreau-Broto (AC).

- **Di-peptide composition feature-set:** This feature generates 400 dimensional descriptors and can be described as follows:

  $$f_{(r,s)} = \frac{N_{rs}}{N-1} \quad r, s = 1, 2, 3, \dots, 20$$

  where; N_{rs} is the number of di-peptide, which signifies type *r* and type *s* amino acid.
- **Tri-peptide composition feature-set:** Tri-peptide feature of ProtR generates 8000 dimensional descriptors and can be defined as follows:

  $$f_{(r,s,t)} = \frac{N_{rst}}{(N-2)} \quad r, s, t = 1, 2, \dots, 20$$

  where, N_{rst} is the number of tri-peptide, which signifies type *r*, type *s*, and type *t* amino acid.
- **Conjoint-Triad feature-set:** Protein interactions like electrostatic and hydrophobic interactions are governed by the type of amino acid present in the protein. Hence, conjoint triad makes use of one amino acid and its neighbouring amino acid to form a triad to calculate the descriptors based on amino acid biochemical classification. First, vector V and F are used to represent protein sequence, where V is the sequence feature vector having v_i (i = 1 to 343) thereby representing one of the triads. F is the counting vector where each f_i (i = 1 to 343) represents frequency v_i. Further, *d* is calculated as follows:

  $$d = \frac{f_i - \min(f_1, f_2, \dots, f_{343})}{\max(f_1, f_2, \dots, f_{343})}$$

- **Moran AC feature:** The Moran AC features are calculated on the basis of amino acid properties distributed in the given protein sequence and can be mathematically represented as:

$$I(d) = \frac{\frac{1}{(N-d)}\sum_{i=1}^{(N-d)}(P_i - \bar{P}')\left(P_{(i+d)} - \bar{P}'\right)}{\frac{1}{N}\sum_{I=1}^{N}(P_i - \bar{P}')^2} \quad d = 1, 2, \ldots, 30$$

where, *d*, *P*$_i$, and *P*$_{(i+d)}$ represents lag in the autocorrelation, property of amino acid at *i*th position, and property of amino acid at *(i + d)*th position, respectively. While $\bar{P}'$ can be calculated as:

$$\bar{P}' = \frac{\sum_{i=1}^{N} P_i}{N}$$

- **Geary AC feature:** Likewise Moran AC, Geary AC is also based on amino acid properties distributed in the given protein sequences and can be calculated as follows:

$$C\,(d) = \frac{\frac{1}{2(N-d)}\sum_{i=1}^{N-d}\left(P_i - P_{(i+d)}\right)^2}{\frac{1}{N}\sum_{i=1}^{N}(P_i - \bar{P}')^2} \quad d = 1, 2, \ldots, 30$$

where, *d*, *P*$_i$, and *P*$_{(i+d)}$ represents lag in the autocorrelation, property of amino acid at *i*th position, and property of amino acid at *(i + d)*th position, respectively. While $\bar{P}'$ can be calculated as:

$$\bar{P}' = \frac{\sum_{i=1}^{N} P_i}{N}$$

- **Normalized Moreau-Broto AC feature:** The Moreau-Broto AC feature can be denoted as:

$$AC(d) = \sum_{i=1}^{N-d} P_i P_{(i+d)} \quad d = 1, 2, \ldots, nlag$$

While the normalized Moreau-Broto autocorrelation can be calculated as follows:

$$ATS(d) = \frac{AC(d)}{(N-d)} \quad d = 1, 2, \ldots, nlag$$

where, d, P_i, $P_{(i+d)}$, and *nlag* represents lag in the autocorrelation, property of amino acid at i^{th} position, property of amino acid at $(i + d)^{th}$ position, and maximum value of lag, respectively.

Model generation for each feature

Each features-set were then used as an input file into the in-house developed Python scripts. The scripts implemented various machine learning algorithms from scikit-learn [19], keras[27], tensorflow[28] package, and machine learning libraries for Python.

- **Stochastic Gradient Descent (SGD):** The SGD tries to optimize differentiable loss function through an iterative process to come-up with least loss value that can classify the given data set with maximum accuracy. Hence, for a given loss function we tuned two parameters; number of iterations ranging from 10 to 500 with steps of 10 and regularization term alpha ranging from 1 to $1.0E^{-07}$ with steps of 10. The loss functions used are as follows:
  - **Log loss:** It gives a probabilistic classifier called logistic regression.
  - **Modified Huber loss:** A smooth loss function that can bring tolerance to outliers and probability estimates too.
  - **Hinge loss:** It is an soft margin loss that gives linear SVM.

  SGD can be implemented in the Python script through `SGDClassifier()` function in package `sklearn.linear_model`.

- **Support Vector Machine (SVM):** SVM are non-probabilistic binary linear classifier and comes under supervised learning models. In Python, `SVC()` forms its basis on libsvm. Since we only implemented Gaussian radial basis function kernel, we tuned C (a regularization parameter that provides the ability to generalized the classifier to unseen data) from 5 to 50 with steps of 5 and Gamma (is inverse of the standard deviation of Gaussian function) from 0.01 to 0.05 with steps of 0.01.

- **Neural Network:** In this type of machine learning technique, a network consists of basic units called neurons that are densely interconnected to solve a specific problem. Figure 3A demonstrates a typical neural network where an input vector is fed into the input layer which is connected to a hidden layer composed of many neurons. Neurons in hidden layers are the place where all computation such as activation function is applied. After computation, the outputs are conveyed to output layer, which on the basis of maximum probability assigns a class to the input sample.

  In our work, we implemented a 2-hidden layer neural network. The neural network is fed with descriptor vectors generated from the ProtR package; hence the number of neurons equals in the input layers equals the number of descriptors in each feature-set. The hidden layer implements ReLu [29] activation function and is composed of 30 and 15 neurons in the first and second hidden layer, respectively. Because the classification performed here is binary in nature, we used a sigmoid activation function in the output layer to calculate the probability of sequence being in either positive or negative class.
- **Long short-term memory (LSTM):** We designed a LSTM based network as portrayed in Figure 3B. Initially, all 20 amino acids had a unique integer assigned to them. Further, due to presence of proteins with varying sequence lengths, we padded the sequence to a length of 300 in the case of AA9 and 350 in the case of AA10. The padding length was chosen based on the mean value of protein sequence length in training data of AA9 and AA10. The padded sequences were then fed into the embedding layer of the neural network architecture that had neurons equal to the total length of padding sequence, i.e., 300 for AA9 and 350 for AA10. Word embeddings are generally used to increase the expressiveness of the network and hence constitutes the learning performance of the network [22, 23].

  The output from the embedding layer was fed into the bi-directional LSTM units whose output was further fed into a fully connected dense neural network (DNN). Figure 3B illustrates the network we used in our work. Optimized values of number of LSTM units in a LSTM layer, number of LSTM layer, number of neurons in dense neural network, and number of layers of DNN to attain the best F-score were 400, 1, (100, 50), and 2, respectively.

Consensus approach

In the case of feature based learning, only those sequences were labelled as potential LPMO that were predicted to be part of the family by all six feature-sets (Figure 2).

Validation of models

To perform validation of various algorithm variants and ensemble methods, maximum voting was performed to screen-out potential LPMOs from list of candidate sequences predicted as LPMO from each feature set. Figure 2 describes the workflow used for validation of AA9 and AA10 models. Performance of the various algorithm variants and ensemble approach was evaluated for test set and validation set using precision and recall. They were calculated as follows:

$$Precision = \frac{TP}{Predicted\ Condition\ Positive\ (TP + FP)}$$

$$Recall = \frac{TP}{Coniditon\ Positive\ (TP + FN)}$$

where, *TP* (True Positive): Annotated as AA9 or AA10 by CAZy and predicted as AA9 or AA10 by a given method

FP (False Positive): Predicted as AA9 or AA10 by a given method but not labelled as AA9 or AA10 by CAZy

FN (False Negative): Annotated as AA9 by CAZy but not by a given method

RESULTS AND DISCUSSION

The total number of sequences downloaded for each family was as follows: AA1, AA2, AA3, AA4, AA6, AA7, AA8, AA9, AA10, AA11, and AA13 were 3620, 543, 1052, 34, 474, 92, 108, 484, 3565, 103, and 25, respectively. The next step was filtering the sequences. After removing sequences that were either partially deposited or containing residues labelled X, the final number of sequences in all the families are as follows: AA1, AA2, AA3, AA4, AA6, AA7, AA8, AA9, AA10, AA11, and AA13 had 1067, 199, 690, 34, 471, 86, 32, 418, 3554, 96, and 25, respectively. Removal of noisy sequences from the positive and negative set drastically reduced the total number of sequences in each family.

In the case of AA10 model, we used a cut-off of 70% similarity in CD-HIT [17] to remove redundant sequences so that we were able to form a balanced training and test set for the various approaches used in the manuscript. This reduced the number of AA10 sequences from 3554 to 517 in AA10 dataset.

Though LPMOs are classified into six AA families, we considered the machine learning based model development for AA9 and AA10 families since the number of sequences in these two families are significantly higher than the other four (AA11, AA13, AA14, and AA15). Thus, the positive set had 418 sequences for AA9 and 517 sequences for AA10 family, respectively. To avoid any hindrance in prediction performance of the model, we built positive and negative set in such a way that the ratio of number of sequences in positive and negative set is 1:1 [20]. Table 1 depicts the distribution of sequences in positive and negative set.

Using ProtR [18] package in in-house developed R scripts, we generated 12 feature-set for the AA9 and AA10 sequences. In order to identify which feature is significant to the input data, we used the Gini index of Random forest in Wekav3.8 [21], where it gives values to descriptors generated from 0.1 to 1. We arbitrarily set a threshold of 0.6 and above as the criteria to select the 6 feature-set that had a value of 0.6 and above in the descriptors generated (Appendix). Specifically, the features are: conjoint triad (343 descriptors), dipeptide composition (400 descriptors), tripeptide composition (8000 descriptors), Moran autocorrelation (240 descriptors), Geary autocorrelation (240 descriptors), and Normalized Moreau-Broto autocorrelation (240 descriptors). Thus, for each sequence there were 9463 descriptors generated.

The features generated were then used as input into the various machine learning algorithms, such as Stochastic Gradient Descent (SGD), Support Vector machine (SVM), and Neural Network (NN). We made use of `GridSearchCV()`, a python function which implements "fit" and "score" method, to exhaustively search between various parameters tuned to get best classifier for SGD and SVM techniques. To trace abstract patterns from the raw data, which most likely remain undiscovered by feature based methods, we applied bi-directional long short-term memory (LSTM). The capability of LSTMs to retain information from more than 1000 time steps has led to its wide applications in bioinformatics in recent years [24, 25, and 26].

For theAA9 dataset, SGD with various loss functions (hinge, log, and modified Huber) and SVM with a radial basis function kernel had recall of 0.922, 0.904, 0.892, and 0.922, respectively on the validation set. Also, a NN with 2-hidden layers had a recall of 0.964, while LSTM gave 0.934. For the AA10 dataset, we observed similar results, where a feature-based NN with 2-hidden layers outperformed other traditional machine learning methods with recall of 0.869, while SVM with radial basis function, SGD with log loss, hinge loss, and modified Huber loss, and LSTM had recall of 0.84, 0.748, 0.806, 0.845 and 0.917, respectively (Table 2).

We further evaluated our different learning methods on independent set. As shown in Table 2, feature-based NN outperformed other learning methods on validation set of both the AA9 and AA10 datasets. Since feature-based NN had fewer false positives and false negatives, we propose feature-based NN as best model for classification of LPMOs into AA9 or AA10 family with current data available.

CONCLUSION

We have successfully collected, analyzed, and generated sequence-based functional annotation of LPMOs. Specifically, we were able to use six sequence-based physiochemical feature-set (di-peptide composition, tri-peptide composition, conjoint triad, Moran AC, Geary AC, normalized Moreau-Broto AC) in feature-based neural network. For the AA9 based model, feature-based NN gave F-score of 0.982 on validation set and 0.939 on independent set. Similarly, the F-score of our proposed method for AA10 model was 0.930 on validation set and 0.726 on independent set. While the validation set is from well-curated AA9 and AA10 sequences, five and 216 sequences were not identified by our method as AA9 and AA10, respectively.

The reason these sequences were not identified by feature-based NN approach as AA9 or AA10 LPMO is most probably because of insufficient descriptors for these sequences. Alternatively, the current used sequence based feature-set may not be able to capture the signal to identify a sequence as AA9 or AA10 LPMO. The method can be improved in the future either by using more descriptors or by using more labeled sequence data for accurate prediction through deep learning methods such as LSTMs.

REFERENCES

1. Hassan, Masjuki Hj., and Md. Abul Kalam. 2013. "An Overview Of Biofuel As A Renewable Energy Source: Development And Challenges". *Procedia Engineering* 56: 39-53.
2. Himmel, M. E., S.-Y. Ding, D. K. Johnson, W. S. Adney, M. R. Nimlos, J. W. Brady, and T. D. Foust. 2007. "Biomass Recalcitrance: Engineering Plants And Enzymes For Biofuels Production". *Science* 315 (5813): 804-807.
3. Frandsen, Kristian E. H., and Leila Lo Leggio. 2016. "Lytic Polysaccharide Monooxygenases: A Crystallographer's View On A New Class Of Biomass-Degrading Enzymes". *Iucrj*3 (6): 448-467.
4. Aachmann, Finn L, Gustav Vaaje-Kolstad, Zarah Forsberg, Åsmund Røhr, Vincent G H Eijsink, and Morten Sørlie. 2015. "Lytic Polysaccharide Monooxygenase". *Encyclopedia Of Inorganic And Bioinorganic Chemistry*, 1-13.
5. Lombard, Vincent, Hemalatha Golaconda Ramulu, Elodie Drula, Pedro M. Coutinho, and Bernard Henrissat. 2013. "The Carbohydrate-Active Enzymes Database (Cazy) In 2013". *Nucleic Acids Research* 42 (D1): D490-D495.
6. Beeson, William T., Van V. Vu, Elise A. Span, Christopher M. Phillips, and Michael A. Marletta. 2015. "Cellulose Degradation By Polysaccharide Monooxygenases". *Annual Review Of Biochemistry* 84 (1): 923-946.
7. Hemsworth, Glyn R., Esther M. Johnston, Gideon J. Davies, and Paul H. Walton. 2015. "Lytic Polysaccharide Monooxygenases In Biomass Conversion". *Trends In Biotechnology* 33 (12): 747-761.
8. Loose, Jennifer S.M., Zarah Forsberg, Marco W. Fraaije, Vincent G.H. Eijsink, and Gustav Vaaje-Kolstad. 2014. "A Rapid Quantitative Activity Assay Shows That Thevibrio Choleraecolonization Factor Gbpa Is An Active Lytic Polysaccharide Monooxygenase". *FEBS Letters*588 (18): 3435-3440.
9. Mutahir, Zeeshan, Sophanit Mekasha, Jennifer S. M. Loose, Faiza Abbas, Gustav Vaaje-Kolstad, Vincent G. H. Eijsink, and Zarah Forsberg. 2018. "Characterization And Synergistic Action Of A Tetra-Modular Lytic Polysaccharide Monooxygenase From Bacillus Cereus". *FEBS Letters*592 (15): 2562-2571.

10. Agostoni, Marco, John A. Hangasky, and Michael A. Marletta. 2017. "Physiological And Molecular Understanding Of Bacterial Polysaccharide Monooxygenases". *Microbiology And Molecular Biology Reviews* 81 (3).
11. Yang, Yalin, Juan Li, Xuewei Liu, Xingliang Pan, Junxiu Hou, Chao Ran, and Zhigang Zhou. 2017. "Improving Extracellular Production Of Serratia Marcescens Lytic Polysaccharide Monooxygenase CBP21 And Aeromonas Veronii B565 Chitinase Chi92 In Escherichia Coli And Their Synergism". *AMB Express* 7 (1).
12. Morgenstern, I., J. Powlowski, and A. Tsang. 2014. "Fungal Cellulose Degradation By Oxidative Enzymes: From Dysfunctional GH61 Family To Powerful Lytic Polysaccharide Monooxygenase Family". *Briefings In Functional Genomics* 13 (6): 471-481.
13. Zhang, Han, Tanner Yohe, Le Huang, Sarah Entwistle, Peizhi Wu, Zhenglu Yang, Peter K Busk, Ying Xu, and Yanbin Yin. 2018. "Dbcan2: A Meta Server For Automated Carbohydrate-Active Enzyme Annotation". *Nucleic Acids Research* 46 (W1): W95-W101.
14. Finn, R. D., J. Clements, and S. R. Eddy. 2011. "HMMER Web Server: Interactive Sequence Similarity Searching". *Nucleic Acids Research* 39 (suppl): W29-W37.
15. Busk, P. K., B. Pilgaard, M. J. Lezyk, A. S. Meyer, and L. Lange. 2017. "Homology To Peptide Pattern For Annotation Of Carbohydrate-Active Enzymes And Prediction Of Function". *BMC Bioinformatics* 18 (1).
16. Buchfink, Benjamin, Chao Xie, and Daniel H Huson. 2014. "Fast And Sensitive Protein Alignment Using DIAMOND". *Nature Methods* 12 (1): 59-60.
17. Huang, Ying, Beifang Niu, Ying Gao, Limin Fu, and Weizhong Li. 2010. "CD-HIT Suite: A Web Server For Clustering And Comparing Biological Sequences". *Bioinformatics* 26 (5): 680-682.
18. Xiao, Nan, Dong-Sheng Cao, Min-Feng Zhu, and Qing-Song Xu. 2015. "Protr/Protrweb: R Package And Web Server For Generating Various Numerical Representation Schemes Of Protein Sequences". *Bioinformatics* 31 (11): 1857-1859.
19. Pedregosa, Fabian, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, and Mathieu Blondel et al. 2011. "Scikit-Learn: Machine Learning In Python". *Journal Of Machine Learning Research* 12: 2825-2830.

20. Guo, Xinjian, Yilong Yin, Cailing Dong, Gongping Yang, and Guangtong Zhou. 2008. "On The Class Imbalance Problem". *2008 Fourth International Conference On Natural Computation*.
21. Holmes, G., A. Donkin, and I.H. Witten. 2018. "WEKA: A Machine Learning Workbench". *Proceedings Of ANZIIS '94 - Australian New Zealnd Intelligent Information Systems Conference*. Accessed December 9.
22. M. Habibi, L. Weber, M. Neves, D. Wiegandt and U. Leser, "Deep learning with word embeddings improves biomedical named entity recognition", Bioinformatics, vol. 33, no. 14, pp. i37-i48, 2017.
23. . Asgari and M. Mofrad, "Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics", PLOS ONE, vol. 10, no. 11, p. e0141287, 2015.
24. K. Yamada and K. Kinoshita, "De novo profile generation based on sequence context specificity with the long short-term memory network", BMC Bioinformatics, vol. 19, no. 1, 2018.
25. J. Hanson, K. Paliwal, T. Litfin, Y. Yang and Y. Zhou, "Accurate prediction of protein contact maps by coupling residual two-dimensional bidirectional long short-term memory with convolutional neural networks", Bioinformatics, 2018.
26. Y. Wang, Z. You, X. Li, T. Jiang, L. Cheng and Z. Chen, "Prediction of protein self-interactions using stacked long short-term memory from protein sequences information", BMC Systems Biology, vol. 12, no. 8, 2018.
27. F. Chollet et al. "Keras". https://github.com/keras-team/keras. 2015.
28. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". 2015.
29. A. F. M. Agarap., "Deep Learning using Rectified Linear Units (ReLU)", 2018.

APPENDIX

List of descriptors having a Gini index of ≥ 0.6 in each feature-set for AA9.

AA9	
Feature-set	**Significant descriptors (Gini index >= 0.6)**
Di-peptide composition	VQ, EF, SW, AN, FN, WE, GW, CL, LR, HN, AA, WR, KA, LM, HA, and WN
Tri-peptide composition	GHF, KSY, VPQ, KGY, EAP, NVR, PEW, AFS, CSS, SWF, TKK, SGA, WEA, MPW, QSY, FEF, ESF, FKS, MYR, SSN, HHF, EGR, INV, KKL, GGQ, RPM, VFT, SDM, NDS, QNR, SNM, QGQ, ELS, TYA, DME, SES, MQP, EGD, MLN, HCK, VNT, DDK, RVV, QEG, PSC, FPH, ERR, ILT, DVN, PHY, WQN, LGQ, RRN, KAF, SEQ, LED, SVC, WSI, GWC, QPL, AHS, GVM, VFQ, DRH, GMY, SAE, SYN, HKH, FYA, IYN, VPK, YWY, YKA, NTV, SWD, KFW, YTV, SHR, TSE, RFE, EKM, LML, EYP, RMR, EVY, PWG, DEC, YAQ, PQW, QPF, MYC, DNS, AEF, PGW, INS, WDK, YKK, KNW, TPN, LTG, WDD, QEN, QAP, HET, TWD, VQQ, GVP, YQQ, TWA, GRT, KCL, DPR, GCI, DIH, YDG, GRE, DTG, TMV, LRT, EIE, RAY, EWK, FQQ, CKL, TIK, GEI, KPW, RFF, AHG, PPP, AQP, IWG, QRF, WPF, DRS, RQW, STI, YWI, WAE, PIV, QQD, ERE, AIK, NHR, VET, NSE, PMC, DVR, IQS, EEK, SRW, EAY, CPD, RSI, QLQ, HIT, AIP, TVH, NVM, VEL, PLS, RWA, SSH, GIY, DSP, TAT, CNP, GEF, RCP, RWG, AIF, YFN, GQI, VLF, TLE, GWS, ADH, VSC, RTT, SFD, DST, VTR, FKK, RFA, EIA, DHS, TQN, NST, and PKA
Conjoint Triad	VS571, VS671, and VS762
Moran autocorrelation	BHAR880101.lag5, CIDH920105.lag3, BHAR880101.lag3, and CIDH920105.lag27
Geary autocorrelation	CIDH920105.lag12, CIDH920105.lag19, CIDH920105.lag3 CIDH920105.lag14,CIDH920105.lag7, and BHAR880101.lag5
Normalized Moreau-Broto autocorrelation	CIDH920105.lag16, CIDH920105.lag9 BHAR880101.lag6 CIDH920105.lag15, CIDH920105.lag7, and CIDH920105.lag23

List of descriptors having a Gini index of ≥ 0.6 in each feature-set for AA10.

AA10	
Feature-set	**Significant descriptors (Gini index >= 0.6)**
Di-peptide composition	QA, RN, IA, FF, KK, MD, EN, PA, and AA
Tri-peptide composition	CQP, LKV, WEY, SSQ, KVS, RSL, DNR, KPR, YFQ, LAL, WQI, RVM, TPM, RDM, MRR, REV, AVC, PQI, CWR, IRQ, VQD, EED, TIH, WGQ, PHQ, MMG, VLH, LFI, SKK, QKK, KWQ, DQH, MRG, CAH, MLF, HDN, AEF, SNE, FMH, GPN, NYQ, LNN, YFA, EGE, LFA, GHR, DTC, SYT, PYH, KGY, NVH, QIE, GKD, TPN, YND, QFT, WFY, SQK, YNP, GQN, IPA, DMI, RMG, GIW, GKN, SIE, WGP, TYR, YIR, QHA, LHC, AQL, FMS, FPP, NNC, YNK, AIC, LFL, YER, FKG, VYI, NFR, NQE, VTE, KAV, LKL, NTI, IKS, SYF, DVM, KSN, CNF, GDG, HNT, FAN, ENI, PVK, FST, IRT, LEN, ILF, PYT, LHN, KTF, DAW, AWQ, RLV, FFK, RYR, AFK, SKA, NHT, HII, QLV, AHV, EDT, ILN, PAN, QPA, YFS, RRL, HIE, TFS, LRC, PRT, RLP, WVC, MVI, MPT, LHW, NER, EFF, VRT, REY, ANH, RSP, QWD, VVR, RND, DTH, WND, KVR, WIN, KAF, LQD, LSN, LAW, NEL, FKL, GGK, FER, MHG, VQG, LGF, RNT, PSA, FTR, WES, AEP, GCF, NFD, PMR, DDL, VKE, LIE, CNA, QAD, IDC, IQN, TCV, LGS, LTV, FVA, VRI, HHP, GVG, TVW, GRR, SNI, AVK, DDE, DFR, KED, VMD, MDK, KTM, MRQ, FSS, NCL, MAS, EIS, DHP, PTS, DLK, FTF, SPS, TAG, SFR, IGV, TIW, VDP, YGN, VSA, EVE, TNS, and LPC
Conjoint Triad	VS761, VS751, VS111, and VS641
Moran autocorrelation	CIDH920105.lag17, CIDH920105.lag30, CIDH920105.lag29, and CIDH920105.lag5
Geary autocorrelation	CIDH920105.lag27, CIDH920105.lag28, CIDH920105.lag22, and CIDH920105.lag7
Normalized Moreau-Broto autocorrelation	CIDH920105.lag10, BHAR880101.lag6, and CIDH920105.lag19

Python script to filter out noise from sequence data

```
Author: Pulkit Anupam Srivastava
Date: 27 July, 2018
Version: 2.0
from Bio import SeqIO
import glob
import os
#
#Path to directory where sequences are kept
path = " ../Projects/FinalYear/Sequences/”
chcklist = list()
for filename in glob.glob(os.path.join(path, '*.fasta')):
    str2ryt_seq = ""
    Sequ = SeqIO.to_dict(SeqIO.parse(filename, "fasta"))
    for key in Sequ:
        if ("X" not in Sequ[key].seq)
        and ("partial" not in Sequ[key].format("fasta"))
        and (key not in chcklist):
            str2ryt_seq += Sequ[key].format("fasta")
            chcklist.append(key)
    with open(filename, "w+") as f:
        f.write(str2ryt_seq)
    f.close()
```

Python script to generate training and test dataset (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 28 July, 2018
from Bio import SeqIO
import random
from random import shuffle
#
#Positive Data-set
#
Sequ = SeqIO.to_dict(SeqIO.parse("../Projects/FinalYear/AAs/AA10/InitialFiles/AA10.fasta",
"fasta"))
Sequ_keysList = [i for i in Sequ]
rand_train = list()
AllNumbers = [i for i in range(0, len(Sequ_keysList))]
shuffle(AllNumbers)
rand_train = AllNumbers[:2132]
rand_test = AllNumbers[-1421:]
#
#Training file
#
with open ("../Projects/FinalYear/AAs/AA10/TrainingFile/AA10.fasta", "w+") as f_Sequ_Train:
    for i in rand_train:
        f_Sequ_Train.write(Sequ[Sequ_keysList[i]].format("fasta"))
f_Sequ_Train.close()
#
#Test File
#
with open ("../Projects/FinalYear/AAs/AA10/TestFile/AA10.fasta", "w+") as f_Sequ_Test:
    for i in rand_test:
        f_Sequ_Test.write(Sequ[Sequ_keysList[i]].format("fasta"))
f_Sequ_Test.close()
#
#Negative Data-set
#
Non_Train = {"AA1":415,"AA2":199,"AA3":415,"AA4":34,"AA6":415,
             "AA7":86,"AA8":32,"AA9":415,"AA11":96,"AA13":25}
Non_Test = {"AA1":681,"AA2":0,"AA3":275,"AA4":0,"AA6":56,
            "AA7":0,"AA8":0,"AA9":3,"AA11":0,"AA13":0}
for fil in Non_Test:
    num = Non_Test[fil]*-1
    Sequ_N =
SeqIO.to_dict(SeqIO.parse("../Projects/FinalYear/AAs/AA10/InitialFiles/NonAA10/"+fil+".fasta",
"fasta"))
    Sequ_N_keysList = [i for i in Sequ_N]
    rand_train = list()
    AllNumbers = [i for i in range(0, len(Sequ_N_keysList))]
    shuffle(AllNumbers)
    rand_train = AllNumbers[:Non_Train[fil]]
    rand_test = AllNumbers[num:]
    #
    #Trainig File
    #
    with open ("../Projects/FinalYear/AAs/AA10/TrainingFile/NonAA10.fasta", "a+") as
f_Sequ_N_Train:
        for i in rand_train:
            f_Sequ_N_Train.write(Sequ_N[Sequ_N_keysList[i]].format("fasta"))
    f_Sequ_N_Train.close()
    #
    #Test File
    #
    c=0
    with open ("../Projects/FinalYear/AAs/AA10/TestFile/NonAA10.fasta", "a+") as f_Sequ_N_Test:
        for i in rand_test:
            if (c<(num*-1)):
                f_Sequ_N_Test.write(Sequ_N[Sequ_N_keysList[i]].format("fasta"))
                c+=1
            else:
                break
    f_Sequ_N_Test.close()
```

R script to generate protein sequence feature-sets for training dataset (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 7 August, 2018
Version: 3.0
library("protr")
library("foreign")
seq_class1 = readFASTA("./Projects/FinalYear/AAs/AA10/TrainingFile/AA10.fasta")
seq_class2 = readFASTA("./Projects/FinalYear/AAs/AA10/TrainingFile/NonAA10.fasta")
#
seq_class1 = seq_class1[(sapply(seq_class1, protcheck))]
seq_class2 = seq_class2[(sapply(seq_class2, protcheck))]
#
#Feature-sets
#
func2perform=c('extractDC','extractTC','extractMoreauBroto','extractCTDC',
               'extractCTDT','extractCTDD','extractAPAAC','extractPAAC',
               'extractCTriad','extractMoran','extractGeary','extractQSO','extractSOCN')
#
for (f in func2perform) {
  class1 = t(sapply(seq_class1, f))
  class2 = t(sapply(seq_class2, f))
  #
  dir.create(file.path("./Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/", f), showWarnings = 
FALSE)
  #

outputfile_csv_class1=paste("./Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/",f,"/",f,"_AA10
.csv", sep="")

outputfile_csv_class2=paste("./Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/",f,"/",f,"_NonA
A10.csv", sep="")
  #
  write.csv(class1,outputfile_csv_class1,sep=",",row.names=TRUE)
  write.csv(class2,outputfile_csv_class2,sep=",",row.names=TRUE)
  #
  data_class1 = read.csv(outputfile_csv_class1,header=TRUE)
  colnames(data_class1)[1]<-"ID"
  data_class1$label <- "AA10"
  #
  data_class2 = read.csv(outputfile_csv_class2,header=TRUE)
  colnames(data_class2)[1]<-"ID"
  data_class2$label <- "NonAA10"
  #
  outputfile_csv=paste("./Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/",f,"/",f,".csv", 
sep="")
  mydata=rbind(data_class1, data_class2)
  write.csv(mydata,outputfile_csv,sep=",",row.names=TRUE)
}
```

R script to generate protein sequence feature-sets for test dataset (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 7 August, 2018
Version: 3.0
library("protr")
library("foreign")
seq_class1 = readFASTA("./Projects/FinalYear/AAs/AA10/TestFile/AA10.fasta")
seq_class2 = readFASTA("./Projects/FinalYear/AAs/AA10/TestFile/NonAA10.fasta")
#
seq_class1 = seq_class1[(sapply(seq_class1, protcheck))]
seq_class2 = seq_class2[(sapply(seq_class2, protcheck))]
#
#Feature-sets that had descriptors with gini index >= 0.6
#
func2perform=c('extractDC','extractTC','extractMoreauBroto',
               'extractCTriad','extractMoran','extractGeary')
#
for (f in func2perform) {
  class1 = t(sapply(seq_class1, f))
  class2 = t(sapply(seq_class2, f))
  #
  dir.create(file.path("./Projects/FinalYear/AAs/AA10/Feature_TestFiles/", f), showWarnings =
FALSE)
  #

outputfile_csv_class1=paste("./Projects/FinalYear/AAs/AA10/Feature_TestFiles/",f,"/",f,"_AA10.csv
", sep="")

outputfile_csv_class2=paste("./Projects/FinalYear/AAs/AA10/Feature_TestFiles/",f,"/",f,"_NonAA10.
csv", sep="")
  #
  write.csv(class1,outputfile_csv_class1,sep=",",row.names=TRUE)
  write.csv(class2,outputfile_csv_class2,sep=",",row.names=TRUE)
  #
  data_class1 = read.csv(outputfile_csv_class1,header=TRUE)
  colnames(data_class1)[1]<-"ID"
  data_class1$label <- "AA10"
  #
  data_class2 = read.csv(outputfile_csv_class2,header=TRUE)
  colnames(data_class2)[1]<-"ID"
  data_class2$label <- "NonAA10"
  #
  outputfile_csv=paste("./Projects/FinalYear/AAs/AA10/Feature_TestFiles/",f,"/",f,".csv", sep="")
  mydata=rbind(data_class1, data_class2)
  write.csv(mydata,outputfile_csv,sep=",",row.names=TRUE)
}
```

Python script to generate optimized models for each feature using modified Huber as loss function in SGD (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 9 November, 2018
Version: 5.0
from io import StringIO
import pandas as pd
import numpy as np
import pickle
from sklearn import grid_search
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import precision_recall_curve, average_precision_score, roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.utils.fixes import signature
from sklearn.metrics import average_precision_score
#
def saveBestModel(X_train, X_test, Y_train, Y_test, func, best_param):
    A_opt = best_param['alpha']
    P_opt = best_param['max_iter']
    classifier = SGDClassifier(alpha = A_opt, loss = "modified_huber",
                               class_weight='balanced', penalty = 'elasticnet', max_iter = P_opt)
    classifier.fit(X_train,Y_train)
    #
    #Save Best Model
    #
    ModelFileName =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_modified_huber_CV10/"+func+".sav"
    pickle.dump(classifier, open(ModelFileName, 'wb'))
    #
    Y_score = classifier.decision_function(X_test)
    #
    #Generate PR Curve
    #
    precision, recall, _ = precision_recall_curve(Y_test, Y_score)
    average_precision = average_precision_score(Y_test, Y_score)
    step_kwargs = ({'step': 'post'} if 'step' in signature(plt.fill_between).parameters else {})
    plt.step(recall, precision, color='b', alpha=0.2,where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='navy', **step_kwargs)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Precision-Recall curve')
    add_PR =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_modified_huber_CV10/PR_Curve/PR_"+func+".png"
    plt.savefig(add_PR)
    plt.close()
    #
    #Generate ROC Curve
    #
    fpr, tpr, _ = roc_curve(Y_test, Y_score)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (AUC = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    add_ROC =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_modified_huber_CV10/ROC_Curve/ROC_"+func+".pn
g"
    plt.legend(loc="lower right")
    plt.savefig(add_ROC)
    plt.close()
    #
    return classifier
#
def combinePredTest(Y_test, Y_pred, ID_Test):
    str2ryt = "Test\tPredicted\tID\n"
    for i in range(0,len(ID_Test)):
        str2ryt += Y_test[i]+"\t"+Y_pred[i]+"\t"+ID_Test[i]+"\n"
    return str2ryt
#
def getBestModel(X_train, X_test, Y_train, Y_test, ID_Test, path2dirs, func):
    Alpha = [10 ** x for x in range(-7, 1)]
    Max_iter = [x for x in range(50, 510, 10)] #max_iter
    param_grid = {'alpha': Alpha, 'max_iter': Max_iter}
    sgd = SGDClassifier(loss = "modified_huber", class_weight='balanced', penalty = 'elasticnet')
    grid_search = GridSearchCV(sgd, param_grid, cv=10)
    grid_search.fit(X_train,Y_train)
    #
    BestModel = saveBestModel(X_train, X_test, Y_train, Y_test, func, grid_search.best_params_)
    Y_pred = BestModel.predict(X_test)
    ModelStatFile = path2dirs+func+"/"+func+"_ModelStats_SGD_modified_huber_CV10.txt"
    with open(ModelStatFile, "w+") as f2:
        f2.write(classification_report(Y_test,Y_pred))
    f2.close()
    #
    PredTest = combinePredTest(Y_test, Y_pred, ID_Test)
    PredTestFile = path2dirs+func+"/"+func+"_PredTest_SGD_modified_huber_CV10.txt"
    with open(PredTestFile, "w+") as f3:
        f3.write(PredTest)
    f3.close()
    #
    str2ryt = func+"\t"+str(grid_search.best_params_['alpha'])+"\t"+
    "\t"+str(grid_search.best_params_['max_iter'])+"\n"
    with
open("../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_modified_huber_CV10/ModelStats.txt","a+"
) as f1:
        f1.write(str2ryt)
    f1.close()
    #
#
path2dirs_Train = "../Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/"
path2dirs_Test = "../Projects/FinalYear/AAs/AA10/Feature_TestFiles/"
func2change = ["extractCTriad","extractDC", "extractGeary",
               "extractMoran", "extractMoreauBroto", "extractTC"]
for i in func2change:
    CSVFile_Train = path2dirs_Train+i+"/"+i+".csv"
    CSVFile_Test = path2dirs_Test+i+"/"+i+".csv"
    #
    df = pd.read_csv(CSVFile_Train,index_col=0)
    df_Train = df.dropna()
    df_Train.reset_index(drop=True, inplace=True)
    #
    df = pd.read_csv(CSVFile_Test,index_col=0)
    df_Test = df.dropna()
    df_Test.reset_index(drop=True, inplace=True)
    #
    X_Train = df_Train.drop(['ID', 'label'], axis = 1)
    Y_Train = df_Train['label']
    X_Test = df_Test.drop(['ID', 'label'], axis = 1)
    Y_Test = df_Test['label']
    ID_Test = df_Test['ID']
    getBestModel(X_Train, X_Test, Y_Train, Y_Test, ID_Test, path2dirs_Train, i)
```

Python script to generate optimized models for each feature using hinge as loss function in SGD (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 9 November, 2018
Version: 5.0
from io import StringIO
import pandas as pd
import numpy as np
import pickle
from sklearn import grid_search
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import precision_recall_curve, average_precision_score, roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.utils.fixes import signature
from sklearn.metrics import average_precision_score
#
def saveBestModel(X_train, X_test, Y_train, Y_test, func, best_param):
    A_opt = best_param['alpha']
    P_opt = best_param['max_iter']
    classifier = SGDClassifier(alpha = A_opt, loss = "hinge",
                               class_weight='balanced', penalty = 'elasticnet', max_iter = P_opt)
    classifier.fit(X_train,Y_train)
    #
    #Save Best Model
    #
    ModelFileName = "../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_hinge_CV10/"+func+".sav"
    pickle.dump(classifier, open(ModelFileName, 'wb'))
    #
    Y_score = classifier.decision_function(X_test)
    #
    #Generate PR Curve
    #
    precision, recall, _ = precision_recall_curve(Y_test, Y_score)
    average_precision = average_precision_score(Y_test, Y_score)
    step_kwargs = ({'step': 'post'} if 'step' in signature(plt.fill_between).parameters else {})
    plt.step(recall, precision, color='b', alpha=0.2,where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='navy', **step_kwargs)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Precision-Recall curve')
    add_PR =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_hinge_CV10/PR_Curve/PR_"+func+".png"
    plt.savefig(add_PR)
    plt.close()
    #
    #Generate ROC Curve
    #
    fpr, tpr, _ = roc_curve(Y_test, Y_score)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (AUC = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    add_ROC =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_hinge_CV10/ROC_Curve/ROC_"+func+".png"
    plt.legend(loc="lower right")
    plt.savefig(add_ROC)
    plt.close()
    #
    return classifier
#
def combinePredTest(Y_test, Y_pred, ID_Test):
    str2ryt = "Test\tPredicted\tID\n"
    for i in range(0,len(ID_Test)):
        str2ryt += Y_test[i]+"\t"+Y_pred[i]+"\t"+ID_Test[i]+"\n"
    return str2ryt
#
def getBestModel(X_train, X_test, Y_train, Y_test, ID_Test, path2dirs, func):
    Alpha = [10 ** x for x in range(-7, 1)]
    Max_iter = [x for x in range(50, 510, 10)] #max_iter
    param_grid = {'alpha': Alpha, 'max_iter': Max_iter}
    sgd = SGDClassifier(loss = "hinge", class_weight='balanced', penalty = 'elasticnet')
    grid_search = GridSearchCV(sgd, param_grid, cv=10)
    grid_search.fit(X_train,Y_train)
    #
    BestModel = saveBestModel(X_train, X_test, Y_train, Y_test, func, grid_search.best_params_)
    Y_pred = BestModel.predict(X_test)
    ModelStatFile = path2dirs+func+"/"+func+"_ModelStats_SGD_hinge_CV10.txt"
    with open(ModelStatFile, "w+") as f2:
        f2.write(classification_report(Y_test,Y_pred))
    f2.close()
    #
    PredTest = combinePredTest(Y_test, Y_pred, ID_Test)
    PredTestFile = path2dirs+func+"/"+func+"_PredTest_SGD_hinge_CV10.txt"
    with open(PredTestFile, "w+") as f3:
        f3.write(PredTest)
    f3.close()
    #
    str2ryt = func+"\t"+str(grid_search.best_params_['alpha'])+"\t"+
    "\t"+str(grid_search.best_params_['max_iter'])+"\n"
    with
open("../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_hinge_CV10/ModelStats.txt","a+") as f1:
        f1.write(str2ryt)
    f1.close()
    #
#
path2dirs_Train = "../Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/"
path2dirs_Test = "../Projects/FinalYear/AAs/AA10/Feature_TestFiles/"
func2change = ["extractCTriad","extractDC", "extractGeary",
             "extractMoran", "extractMoreauBroto", "extractTC"]
for i in func2change:
    CSVFile_Train = path2dirs_Train+i+"/"+i+".csv"
    CSVFile_Test = path2dirs_Test+i+"/"+i+".csv"
    #
    df = pd.read_csv(CSVFile_Train,index_col=0)
    df_Train = df.dropna()
    df_Train.reset_index(drop=True, inplace=True)
    #
    df = pd.read_csv(CSVFile_Test,index_col=0)
    df_Test = df.dropna()
    df_Test.reset_index(drop=True, inplace=True)
    #
    X_Train = df_Train.drop(['ID', 'label'], axis = 1)
    Y_Train = df_Train['label']
    X_Test = df_Test.drop(['ID', 'label'], axis = 1)
    Y_Test = df_Test['label']
    ID_Test = df_Test['ID']
    getBestModel(X_Train, X_Test, Y_Train, Y_Test, ID_Test, path2dirs_Train, i)
```

Python script to generate optimized models for each feature using log as loss function in SGD (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 9 November, 2018
Version: 5.0
from io import StringIO
import pandas as pd
import numpy as np
import pickle
from sklearn import grid_search
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import precision_recall_curve, average_precision_score, roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.utils.fixes import signature
from sklearn.metrics import average_precision_score
#
def saveBestModel(X_train, X_test, Y_train, Y_test, func, best_param):
    A_opt = best_param['alpha']
    P_opt = best_param['max_iter']
    classifier = SGDClassifier(alpha = A_opt, loss = "log",
                               class_weight='balanced', penalty = 'elasticnet', max_iter = P_opt)
    classifier.fit(X_train,Y_train)
    #
    #Save Best Model
    #
    ModelFileName = "../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_log_CV10/"+func+".sav"
    pickle.dump(classifier, open(ModelFileName, 'wb'))
    Y_score = classifier.decision_function(X_test)
    #
    #Generate PR Curve
    #
    precision, recall, _ = precision_recall_curve(Y_test, Y_score)
    average_precision = average_precision_score(Y_test, Y_score)
    step_kwargs = ({'step': 'post'} if 'step' in signature(plt.fill_between).parameters else {})
    plt.step(recall, precision, color='b', alpha=0.2,where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='navy', **step_kwargs)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Precision-Recall curve')
    add_PR =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_log_CV10/PR_Curve/PR_"+func+".png"
    plt.savefig(add_PR)
    plt.close()
    #
    #Generate ROC Curve
    #
    fpr, tpr, _ = roc_curve(Y_test, Y_score)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (AUC = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    add_ROC =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_log_CV10/ROC_Curve/ROC_"+func+".png"
    plt.legend(loc="lower right")
    plt.savefig(add_ROC)
    plt.close()
    #
    return classifier
```

```
#
def combinePredTest(Y_test, Y_pred, ID_Test):
    str2ryt = "Test\tPredicted\tID\n"
    for i in range(0,len(ID_Test)):
        str2ryt += Y_test[i]+"\t"+Y_pred[i]+"\t"+ID_Test[i]+"\n"
    return str2ryt
#
def getBestModel(X_train, X_test, Y_train, Y_test, ID_Test, path2dirs, func):
    Alpha = [10 ** x for x in range(-7, 1)]
    Max_iter = [x for x in range(50, 510, 10)] #max_iter
    param_grid = {'alpha': Alpha, 'max_iter': Max_iter}
    sgd = SGDClassifier(loss = "log", class_weight='balanced', penalty = 'elasticnet')
    grid_search = GridSearchCV(sgd, param_grid, cv=10)
    grid_search.fit(X_train,Y_train)
    #
    BestModel = saveBestModel(X_train, X_test, Y_train, Y_test, func, grid_search.best_params_)
    Y_pred = BestModel.predict(X_test)
    ModelStatFile = path2dirs+func+"/"+func+"_ModelStats_SGD_log_CV10.txt"
    with open(ModelStatFile, "w+") as f2:
        f2.write(classification_report(Y_test,Y_pred))
    f2.close()
    #
    PredTest = combinePredTest(Y_test, Y_pred, ID_Test)
    PredTestFile = path2dirs+func+"/"+func+"_PredTest_SGD_log_CV10.txt"
    with open(PredTestFile, "w+") as f3:
        f3.write(PredTest)
    f3.close()
    #
    str2ryt = func+"\t"+str(grid_search.best_params_['alpha'])+"\t"+
    "\t"+str(grid_search.best_params_['max_iter'])+"\n"
    with open("../Projects/FinalYear/AAs/AA10/OptimisedModels/SGD_log_CV10/ModelStats.txt","a+")
as f1:
        f1.write(str2ryt)
    f1.close()
    #
#
path2dirs_Train = "../Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/"
path2dirs_Test = "../Projects/FinalYear/AAs/AA10/Feature_TestFiles/"
func2change = ["extractCTriad","extractDC", "extractGeary",
               "extractMoran", "extractMoreauBroto", "extractTC"]
for i in func2change:
    CSVFile_Train = path2dirs_Train+i+"/"+i+".csv"
    CSVFile_Test = path2dirs_Test+i+"/"+i+".csv"
    #
    df = pd.read_csv(CSVFile_Train,index_col=0)
    df_Train = df.dropna()
    df_Train.reset_index(drop=True, inplace=True)
    #
    df = pd.read_csv(CSVFile_Test,index_col=0)
    df_Test = df.dropna()
    df_Test.reset_index(drop=True, inplace=True)
    #
    X_Train = df_Train.drop(['ID', 'label'], axis = 1)
    Y_Train = df_Train['label']
    X_Test = df_Test.drop(['ID', 'label'], axis = 1)
    Y_Test = df_Test['label']
    ID_Test = df_Test['ID']
    getBestModel(X_Train, X_Test, Y_Train, Y_Test, ID_Test, path2dirs_Train, i)
```

Python script to generate optimized models for each feature using radial basis function as kernel in SVC (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 11 August, 2018
Version: 3.0
from io import StringIO
import pandas as pd
import numpy as np
import pickle
from sklearn import grid_search
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import precision_recall_curve, average_precision_score, roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.utils.fixes import signature
from sklearn.metrics import average_precision_score
#
def saveBestModel(X_train, X_test, Y_train, Y_test, func, best_param):
    C_opt = best_param['C']
    gamma_opt = best_param['gamma']
    classifier = SVC(kernel='rbf', C = C_opt, gamma = gamma_opt)
    classifier.fit(X_train,Y_train)
    #
    #Save Best Model
    #
    ModelFileName = "../Projects/FinalYear/AAs/AA10/OptimisedModels/SVC_rbf_CV10/"+func+".sav"
    pickle.dump(classifier, open(ModelFileName, 'wb'))
    #
    Y_score = classifier.decision_function(X_test)
    #
    #Generate PR Curve
    #
    precision, recall, _ = precision_recall_curve(Y_test, Y_score)
    average_precision = average_precision_score(Y_test, Y_score)
    step_kwargs = ({'step': 'post'} if 'step' in signature(plt.fill_between).parameters else {})
    plt.step(recall, precision, color='b', alpha=0.2,where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='navy', **step_kwargs)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Precision-Recall curve')
    add_PR =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SVC_rbf_CV10/PR_Curve/PR_"+func+".png"
    plt.savefig(add_PR)
    plt.close()
    #
    #Generate ROC Curve
    #
    fpr, tpr, _ = roc_curve(Y_test, Y_score)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (AUC = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    add_ROC =
"../Projects/FinalYear/AAs/AA10/OptimisedModels/SVC_rbf_CV10/ROC_Curve/ROC_"+func+".png"
    plt.legend(loc="lower right")
    plt.savefig(add_ROC)
    plt.close()
    #
    return classifier
```

```
#
def combinePredTest(Y_test, Y_pred, ID_Test):
    str2ryt = "Test\tPredicted\tID\n"
    for i in range(0,len(ID_Test)):
        str2ryt += Y_test[i]+"\t"+Y_pred[i]+"\t"+ID_Test[i]+"\n"
    return str2ryt
#
def getBestModel(X_train, X_test, Y_train, Y_test, ID_Test, path2dirs, func):
    C = [k for k in range(5, 105,5)]
    Gamma = np.arange(0.01, 0.08, 0.01)
    param_grid = {'C': C, 'gamma' : Gamma}
    grid_search = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=10)
    grid_search.fit(X_train,Y_train)
    #
    BestModel = saveBestModel(X_train, X_test, Y_train, Y_test, func, grid_search.best_params_)
    Y_pred = BestModel.predict(X_test)
    ModelStatFile = path2dirs+func+"/"+func+"_ModelStats_SVC_rbf_CV10.txt"
    with open(ModelStatFile, "w+") as f2:
        f2.write(classification_report(Y_test,Y_pred))
    f2.close()
    #
    PredTest = combinePredTest(Y_test, Y_pred, ID_Test)
    PredTestFile = path2dirs+func+"/"+func+"_PredTest_SVC_rbf_CV10.txt"
    with open(PredTestFile, "w+") as f3:
        f3.write(PredTest)
    f3.close()
    #
    str2ryt = 
func+"\t"+str(grid_search.best_params_['C'])+"\t"+"\t"+str(grid_search.best_params_['gamma'])+"\n
"
    with open("../Projects/FinalYear/AAs/AA10/OptimisedModels/SVC_rbf_CV10/ModelStats.txt","a+") 
as f1:
        f1.write(str2ryt)
    f1.close()
    #
#
path2dirs_Train = "../Projects/FinalYear/AAs/AA10/Feature_TrainingFiles/"
path2dirs_Test = "../Projects/FinalYear/AAs/AA10/Feature_TestFiles/"
func2change = ["extractCTriad","extractDC", "extractGeary",
               "extractMoran", "extractMoreauBroto", "extractTC"]
for i in func2change:
    CSVFile_Train = path2dirs_Train+i+"/"+i+".csv"
    CSVFile_Test = path2dirs_Test+i+"/"+i+".csv"
    #
    df = pd.read_csv(CSVFile_Train,index_col=0)
    df_Train = df.dropna()
    df_Train.reset_index(drop=True, inplace=True)
    #
    df = pd.read_csv(CSVFile_Test,index_col=0)
    df_Test = df.dropna()
    df_Test.reset_index(drop=True, inplace=True)
    #
    X_Train = df_Train.drop(['ID', 'label'], axis = 1)
    Y_Train = df_Train['label']
    X_Test = df_Test.drop(['ID', 'label'], axis = 1)
    Y_Test = df_Test['label']
    ID_Test = df_Test['ID']
    getBestModel(X_Train, X_Test, Y_Train, Y_Test, ID_Test, path2dirs_Train, i)
```

Python script to generate optimized model for each feature using feature based neural network (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 06 Jan, 2019
Version: 1.0
import numpy as np
import pandas as pd
from sklearn import metrics
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
#
def configureModel(numSamples, numFeatures, layer1, layer2, layer3, layer4):
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(layer1, input_shape = (numFeatures,), activation="relu"),
        #tf.keras.layers.Dense(layer2, activation="relu"),
        tf.keras.layers.Dense(layer3, activation="relu"),
        tf.keras.layers.Dense(layer4, activation="sigmoid")]) #sigmoid for binary classificaiton
    model.compile(optimizer = tf.keras.optimizers.Adam(lr = 0.001),
                  loss = 'sparse_categorical_crossentropy',
                  metrics = ['accuracy'])
    return model
#
def getBestModel(X_Train, X_Test, Y_Train, Y_Test, ID_Test, path2dirs_Train, i):
    #print (X_Test.shape,Y_Test.shape)
    model_i = configureModel(X_Train.shape[0], X_Train.shape[1], 30, 30, 15, 2)#100, 50,25,2
    history = model_i.fit(X_Train, Y_Train, batch_size = 50, epochs = 20, validation_data =
(X_Test, Y_Test))
    saveModel_dir = "../Projects/FinalYear/AAs/AA10/ANN/OptimizedModel/"+i+".h5"
    model_i.save(saveModel_dir)
    #
    history_dict = history.history
    history_dict.keys()
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    # "bo" is for "blue dot"
    plt.plot(epochs, acc, 'bo', label='Training loss')
    # b is for "solid blue line"
    plt.plot(epochs, val_acc, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
#
path2dirs_Train = "../Projects/FinalYear/AAs/AA10/ANN/Feature_TrainingFiles/"
path2dirs_Test = "../Projects/FinalYear/AAs/AA10/ANN/Feature_TestFiles/"
func2change = ["extractCTriad","extractDC", "extractGeary",
               "extractMoran", "extractMoreauBroto", "extractTC"]
for i in func2change:
    CSVFile_Train = path2dirs_Train+i+"/"+i+".csv"
    CSVFile_Test = path2dirs_Test+i+"/"+i+".csv"
    #
    df = pd.read_csv(CSVFile_Train,index_col=0, dtype={'label':str})
    df_Train = df.dropna()
    df_Train.reset_index(drop=True, inplace=True)
    #
    df = pd.read_csv(CSVFile_Test,index_col=0, dtype={'label':str})
    df_Test = df.dropna()
    df_Test.reset_index(drop=True, inplace=True)
    #
    Y_Train, Y_Test, ID_Test = pd.DataFrame(), pd.DataFrame(), pd.DataFrame()
    X_Train = df_Train.drop(['ID', 'label'], axis = 1)
    Y_Train["label"] = df_Train['label'].replace(["AA10", "NonAA10"], [0, 1])
    X_Test = df_Test.drop(['ID', 'label'], axis = 1)
    Y_Test["label"] = df_Test['label'].replace(["AA10", "NonAA10"], [0, 1])
    ID_Test["label"] = df_Test['ID'].replace(["AA10", "NonAA10"], [0, 1])
    getBestModel(X_Train, X_Test, Y_Train, Y_Test, ID_Test, path2dirs_Train, i)
    print (i)
#
```

Python script to generate optimized model using long short-term memory units (AA10 family)

```
Author: Pulkit Anupam Srivastava
Date: 24 Jan, 2019
Version: 9.0
from sklearn import metrics
from sklearn.metrics import classification_report
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding, LSTM, Bidirectional, Dropout
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import pickle
def configureModel():
    model = Sequential()
    model.add(Embedding(21, 300, input_length=350))
    model.add(Dropout(0.5))
    model.add(Bidirectional(LSTM(400, dropout=0.5, recurrent_dropout=0.5)))
    model.add(Dropout(0.5))
    model.add(Dense(100, activation="relu"))
    model.add(Dense(50, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation="sigmoid")) #sigmoid for binary classificaiton
    model.compile(optimizer = "adam",
                  loss = 'sparse_categorical_crossentropy',
                  metrics = ['accuracy'])
    model.summary()
    return model
#
def getBestModel(X_Train, X_Test, Y_Train, Y_Test):
    total_AA=21
    model_i = configureModel()#100, 50,25,2
    history = model_i.fit(X_Train, Y_Train, batch_size = 50, epochs = 60, validation_data = 
(X_Test, Y_Test))
    saveModel_dir = 
"../Projects/FinalYear/AAs/AA10/RNN/OptimizedModel/Em300_BiLSTM400_D100_D50_Ep60/Em300_BiLSTM400_
D100_D50_Ep60.h5"
    model_i.save(saveModel_dir)
    #simple_save(session,export_dir,)
    #
    predictions = model_i.predict(X_Test)
    Y_Pred = np.argmax(predictions, 1)
    print (classification_report(Y_Test,Y_Pred))
    #
    history_dict = history.history
    history_dict.keys()
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    # "bo" is for "blue dot"
    plt.plot(epochs, acc, 'bo', label='Training loss')
    # b is for "solid blue line"
    plt.plot(epochs, val_acc, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
#
#Pre-processing
```

```
#Reading training and test file
TrainFileData = pd.read_csv("../Projects/FinalYear/AAs/AA10/RNN/TrainingFile/TrainingFile.csv")
TestFileData = pd.read_csv("../Projects/FinalYear/AAs/AA10/RNN/TestFile/TestFile.csv")
#Initializing Token with num_words=21 since there are 20 amino acids
tokenizer = Tokenizer(num_words=21, char_level=True, split='')
#Training the tokenizer on training data
tokenizer.fit_on_texts(TrainFileData['Sequence'])
#Saving tokenizer for future use
TokenizerFile =
"../Projects/FinalYear/AAs/AA10/RNN/OptimizedModel/Em300_BiLSTM400_D100_D50_Ep60/tokenizer.pickle
"
with open(TokenizerFile, 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
#Converting the each amino acid to a integer
X_Train = tokenizer.texts_to_sequences(TrainFileData['Sequence'])
X_Test = tokenizer.texts_to_sequences(TestFileData['Sequence'])
#Padding the sequence to avoid mismatch in length
X_Train = pad_sequences(X_Train, maxlen=350)
X_Test = pad_sequences(X_Test, maxlen=350)
#Label
Y_Train = TrainFileData['Label']
Y_Test = TestFileData['Label']
#Calling RNN function
getBestModel(X_Train, X_Test, Y_Train, Y_Test)
```

Python script uploaded on GitHub for Data Cleaning

```
#!/usr/bin/env python3
#Author: Pulkit Anupam Srivastava
#Co-authors: Eric L. Hegg, Michigan State University
#           Brian G. Fox, University of Wisconsin-Madison
#           Ragothaman M. Yennamalli, Jaypee University of Information Technology
#Version: 1.0
#Last Modified: 26 Feb, 2019
#Description: The script filters out protein sequences with "X" amino acid.
#Input: Fasta filename containing protein sequences with unrecognized residues.
#Ouput: A fasta file with filtered out protein sequences
from Bio import SeqIO
import glob
import os
import sys
#
def genCleanFile(filename):
   path_name, file_name = os.path.split(filename)
   Sequences = SeqIO.to_dict(SeqIO.parse(filename, "fasta"))
   str2ryt_newFile = ""
   for key in Sequences:
      if ("X" not in Sequences[key].seq):
          str2ryt_newFile += Sequences[key].format("fasta")
   outfile = path_name+"/Cleaned_"+file_name
   with open(outfile, "w+") as f:
       f.write(str2ryt_newFile)
   f.close()
   return outfile
#
```

Python script uploaded on GitHub for feature extraction

```
#!/usr/bin/env Rscript
#Author: Pulkit Anupam Srivastava
#Co-authors: Eric L. Hegg, Michigan State University
#            Brian G. Fox, University of Wisconsin-Madison
#            Ragothaman M. Yennamalli, Jaypee University of Information Technology
#Version: 1.0
#Last Modified: 26 Feb, 2019
#Description: The script generate feature-sets for given protein sequence(s).
#Input: Path to fasta file. Name of fasta file.
#Output: Generates files containing descriptor value for all sequences in fasta file.
library("protr")
library("foreign")
#
args = commandArgs(TRUE)
path2dir <- args[1]
FastaFile <- args[2]
#
seq_class1 = readFASTA(FastaFile)
seq_class1 = seq_class1[(sapply(seq_class1, protcheck))]
#List of feature-sets
func2perform=c('extractMoreauBroto','extractTC','extractDC',
               'extractMoran','extractGeary','extractCTriad')
#Iteration over list of feature-sets
for (f in func2perform)
{
  class1 = t(sapply(seq_class1, f))
  #Create a folder with feature-set name
  dir.create(file.path(path2dir, f), showWarnings = FALSE)
  #
  outputfile_csv=file.path(path2dir,f,paste(f,".csv",sep=""))
  write.csv(class1,outputfile_csv,sep=",",row.names=TRUE)
  #Writes csv file having descriptor values for each protein sequence
  data_class1 = read.csv(outputfile_csv,header=TRUE)
  colnames(data_class1)[1]<-"ID"
  data_class1$label <- "?"
  mydata=rbind(data_class1)
  write.csv(mydata,outputfile_csv,sep=",",row.names=TRUE)
}
```

Python script uploaded on GitHub for feature-based neural network prediction

```
#!/usr/bin/env python
#Author: Pulkit Anupam Srivastava
#Co-authors: Eric L. Hegg, Michigan State University
#           Brian G. Fox, University of Wisconsin-Madison
#           Ragothaman M. Yennamalli, Jaypee University of Information Technology
#Version: 1.0
#Last Modified: 26 Feb, 2019
#Description: The script annotates a protein sequence as a member of given LPMO family or not.
The prediction method
#implemented in the script is feature based neural network, where descriptor value of protein
sequences for each
#fetaure-set is fed into the neural network. Once prediction is made by script for each feature-
set, the common
#protein sequences predicted as a member of LPMO family in all feature-set is further labelled as
potential LPMO.
#Input: Path to fasta file. Name of the family of LPMO.
#Output: A file with potential LPMO protein sequences.
import numpy as np
import pandas as pd
import tensorflow as tf
import glob
import os
import sys
import subprocess
import copy
from tensorflow import keras
from Bio import SeqIO
from DataCleaning import genCleanFile
#To extract IDs predicted as LPMO
def getID(df):
    ID_List = list()
    for i in range(1,len(df)):
        if (df.iloc[i].Predicted == 0):
            #print ("D")
            ID_List.append(df.iloc[i].ID)
    return ID_List
#To write a file having probability of a sequence to be LPMO or not
def combinePredTest(Y_pred, ID_Test):
    str2ryt = "Predicted\tID\n"
    for i in range(0,len(ID_Test)):
        str2ryt += str(Y_pred[i])+"\t"+ID_Test.iloc[i]+"\n"
    return str2ryt
#To annotate a sequence as member of given LPMO family
def getPrediction(X_Test, ID_Test, path, func, family):
    optimizedModel = os.path.join(sys.path[0],"Models",family,"fbdl",func+'.h5')
    new_model = keras.models.load_model(optimizedModel)
    predictions = new_model.predict(X_Test)
    Y_Pred = np.argmax(predictions, 1)
    #
    PredTest = combinePredTest(Y_Pred, ID_Test)
    PredTestFile = os.path.join(path,func,"FB_Predictions.txt")
    with open(PredTestFile, "w+") as f_prediction:
        f_prediction.write(PredTest)
    f_prediction.close()
#Input to be given while execution
path2file = sys.argv[1]
family = sys.argv[2]
path, input_filename = os.path.split(path2file)
```

```
#To filter out protein sequence having unrecognized residues
CleanedFastaFile = genCleanFile(path2file)
#To generate descriptor value of protein sequences for each feature-set
subprocess.call("Rscript "+os.path.join(sys.path[0],"ExtractFeatures.R ")+path+"
"+CleanedFastaFile, shell=True)
#List of feature-set
func2change = ["extractGeary", "extractCTriad","extractDC",
               "extractMoran", "extractMoreauBroto", "extractTC"]
#Iteration over list of feature set for prediction of LPMO family members
for i in func2change:
    CSVFile = os.path.join(path,i,i+".csv")
    df = pd.read_csv(CSVFile,index_col=0, dtype={'label':str})
    df_Test = df.dropna()
    df_Test.reset_index(drop=True, inplace=True)
    X_Test = df_Test.drop(['ID', 'label'], axis = 1)
    ID = df_Test['ID']
    getPrediction(X_Test, ID, path, i, family)
#For extracting common IDs
first_list = second_list = common_list = list()
for i in range(0,len(func2change)):
    CSVFile = os.path.join(path,func2change[i],"FB_Predictions.txt")
    df = pd.read_csv(CSVFile, sep = '\t')
    if (i == 0):
        ID_List_1 = getID(df)
        first_list = copy.copy(ID_List_1)
    else:
        second_list = getID(df)
        common_list = list()
        for j in first_list:
            if j in second_list:
                common_list.append(j)
        first_list = copy.copy(common_list)
#For extracting Fasta sequence using common ID
Sequences = SeqIO.to_dict(SeqIO.parse(CleanedFastaFile, "fasta"))
str2ryt_newFile = ""
for key in common_list:
   str2ryt_newFile += Sequences[key].format("fasta")
outSeqFile = os.path.join(path,"FB_Potential"+family+".fasta")
with open(outSeqFile, "w+") as f_potentialLPMO:
    f_potentialLPMO.write(str2ryt_newFile)
f_potentialLPMO.close()
```

Python script uploaded on GitHub for long short-term memory (LSTMs) based prediction

```
#!/usr/bin/env python
#Author: Pulkit Anupam Srivastava
#Co-authors: Eric L. Hegg, Michigan State University
#           Brian G. Fox, University of Wisconsin-Madison
#           Ragothaman M. Yennamalli, Jaypee University of Information Technology
#Version: 1.0
#Last Modified: 26 Feb, 2019
#Description: The script annotates a protein sequence as a member of given LPMO family or not.
The prediction method
#implemented in the script is long short-term based neural network.
#Input: Path to fasta file. Name of the family of LPMO.
#Output: A file with potential LPMO protein sequences.
from DataCleaning import genCleanFile
from tensorflow import keras
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from Bio import SeqIO
import pickle
import glob
import os
import sys
import numpy as np
import pandas as pd
import tensorflow as tf
#Pre-processing of fasta file for prediction
def PrepareFile(path, FastaFile):
    str2ryt = "ID,Sequence\n"
    Sequences = SeqIO.to_dict(SeqIO.parse(FastaFile, "fasta"))
    for key in Sequences:
        seq = Sequences[key].seq
        str2ryt+= str(key)+","+str(seq)+"\n"
    #Writing the pre-processed csv file
    PreparedFile = os.path.join(path, "PreparedFile.csv")
    with open(PreparedFile, "w+") as f_prepared:
        f_prepared.write((str2ryt.strip()))
    f_prepared.close()
    return PreparedFile
#To write a file having probability of a sequence to be LPMO or not
def combinePredTest(Y_Pred, ID_Test, predictions):
    str2ryt="ID\tConfidence\n"
    for i in range(len(Y_Pred)):
        if Y_Pred[i] == 1:
            str2ryt+=ID_Test.iloc[i]+"\t"+str(predictions[i][1])+"\n"
    return ((str2ryt.strip()))
#To annotate a sequence as member of given LPMO family
def getPrediction(X_Test, ID_Test, path, family):
    optimizedModel = os.path.join(sys.path[0],'Models',family,'lstm','BestModel.h5')
    new_model = keras.models.load_model(optimizedModel)
    #
    TokenizerFile = os.path.join(sys.path[0],'Models',family,'lstm','tokenizer.pickle')
    with open(TokenizerFile, 'rb') as handle:
        tokenizer = pickle.load(handle)
    X_Test = tokenizer.texts_to_sequences(X_Test)
    if ("AA9" in family):
        X_Test = pad_sequences(X_Test, maxlen=300)
    elif ("AA10" in family):
        X_Test = pad_sequences(X_Test, maxlen=350)
    #
    predictions = new_model.predict(X_Test)
    Y_Pred = np.argmax(predictions, 1)
    #
    PredTest = combinePredTest(Y_Pred, ID_Test, predictions)
    PredTestFile = os.path.join(path,"LSTM_Predictions.txt")
    with open(PredTestFile, "w+") as f_prediction:
        f_prediction.write(PredTest)
    f_prediction.close()
    return PredTest
#Input to be given while execution
path2file = sys.argv[1]
family = sys.argv[2]
path, input_filename = os.path.split(path2file)
#To filter out protein sequence having unrecognized residues
CleanedFastaFile = genCleanFile(path2file)
#For Prediciton
TestFileData = pd.read_csv(PrepareFile(path, CleanedFastaFile))
X_Test = TestFileData['Sequence']
ID_Test = TestFileData['ID']
common_list = getPrediction(X_Test, ID_Test, path, family)
#For extracting Fasta sequence using ID
Sequences = SeqIO.to_dict(SeqIO.parse(CleanedFastaFile, "fasta"))
ID_List = pd.read_csv(os.path.join(path,"LSTM_Predictions.txt"), sep = "\t")
common_list = ID_List['ID']
str2ryt_newFile = ""
for key in common_list:
   str2ryt_newFile += Sequences[key].format("fasta")
outSeqFile = os.path.join(path,"LSTM_Potential"+family+".fasta")
with open(outSeqFile, "w+") as f_potentialLPMO:
    f_potentialLPMO.write(str2ryt_newFile)
f_potentialLPMO.close()
```