NATIONAL INSTITUTE OF TECHNOLOGY RAIPUR

RAIPUR, CHHATTISGARH – 492010

DEPARTMENT OF BIOTECHNOLOGY ENGINEERING

MAJOR PROJECT ON

PROTEIN STRUCTURE & FUNCTION PREDICTION

SUBMITTED BY

HARISH GANESH R (18112020)

JAWAHAR B (18112022)

SUPERVISED BY

Dr. SHARDA BHARTI

ABSTRACT

We present a bidirectional LSTM based approach for predicting the secondary structure of a protein from its amino acid sequence. It is important to know the protein structure to determine its function. Protein structures are analysed using X-ray crystallography, NMR, and CryoEm each with its pros and cons but we know sequences of many proteins than their structures due to the cost and time taken for analysing the structures. Therefore, it's important to predict the structure of a protein from its amino acid sequence. We use the bidirectional LSTM method to predict the secondary structure of a protein from its amino acid sequence and we classify them as helix, sheet, and loop. This method gives a reasonable accuracy for predicting the protein structure. We have then incorporated a function prediction of protein from its sequence using machine learning techniques and have analysed the algorithm that provides the best prediction based on the accuracy delivered by various algorithms.

Keywords: Recurrent Neural Network, long short-term memory, Machine learning

CANDIDATE'S DECLARATION

We declare that the project entitled ‘Protein secondary structure and function prediction’ is an original piece of work carried out by the following mentioned students, under the guidance of Dr. Sharda Bharti, Assistant Professor, Department of Biotechnology Engineering. The project has not formed the basis for the award of any other degree associate-ship, fellowship, or other similar titles. The information has been collected from authentic sources. The details of the project are based on the personal evaluation made during the tenure of our minor project.

Place: Raipur

Date:

Signature of the student:

HARISH GANESH R (18112020)

JAWAHAR B (18112022)

CERTIFICATE

This is to certify that the project entitled 'Protein secondary structure and function prediction' being submitted by **Harish Ganesh R, Jawahar B** to the National Institute of Technology, Raipur for the partial fulfillment of the requirement of the degree of Bachelor in Technology in Biotechnology Engineering, is an authentic record of research work carried out under our supervision and guidance The work incorporated in this thesis has not been, to the best of my knowledge, submitted to any other university or institute for the award of a degree or diploma.

Guided By:

Dr. Sharda Bharti

Assistant Professor,

Department of Biotechnology Engineering

Counter-signed By:

Date:

Place: Raipur

ACKNOWLEDGEMENT

The authors express their deep sense of gratitude to the project supervisor Dr. Sharda Bharti, Assistant Professor, Department of Biotechnology Engineering NIT Raipur for her invaluable encouragement, helpful suggestions, and supervision throughout this course of this work. Her willingness to devote time and attention amidst numerous responsibilities is gratefully acknowledged. Above all her, patience and optimistic attitude led to carrying out this work. Our heartiest thanks to **Prof. Dr. Lata Sheo Bachan Upadhyay, Head of the Department of Biotechnology Engineering, NIT Raipur**, for her valuable guidance and consistent encouragement throughout the course of the project.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

PSSM	Position Specific-Scoring Matrix
PDB	Protein Data Bank
NMR	Nuclear Magnetic Resonance
BLAST	Basic Local Alignment Search Tool
PSI-BLAST	Position-Specific Iterative Basic Local Alignment Search Tool
RNN	Recurrent Neural Networks
BPTT	Backpropagation through time
LSTM	Long Short-Term Memory
NN	Neural Networks
RMS	Root Mean Square
MSA	Multiple Sequence Alignment
A.A	Amino Acid
ML	Machine Learning
NB	Naive Bayes

LIST OF FIGURES

CHAPTER 1

1. Introduction

Knowing protein structures and functions will enable us to understand better about various mechanisms undergone by protein. This ranges from immunology to cell signaling and enzymology. There are many diseases that occur due to mutation that results in a change of protein structure like Parkinson's disease and Alzheimer's disease. By understanding the protein structures and their mechanism, we can develop drugs that have more efficacy. Traditionally, protein structure is determined using methods like X-ray crystallization, NMR, and electron microscopy. Each comes with its own benefits and limitations. 200 million protein sequences have been collected in GenBank but only 100 000 protein structures have been deposited in Protein Data Bank, the central depository of all protein structures it is due to the high cost of protein structure determination ($100,000 per protein) and low cost of whole-genome sequencing ($1000) Predicting protein structure using artificial intelligence can speed up the process of structure determination. This can also reduce the cost and time of this process rigorously for biotechnologists. The protein function prediction enables us to know the function of newly found proteins from their amino acid sequence without spending more time on experimental methods of function determination. While techniques such as microarray analysis, RNA interference, and the yeast two-hybrid system can be used to experimentally demonstrate the function of a protein, advances in sequencing technologies have made the rate at which proteins can be experimentally characterized much slower than the rate at which new sequences become available. Thus, the annotation of new sequences through computational methods can often be done quickly and for many genes or proteins at once.

2. Problem Statement

Knowing protein structures will enable us to understand better various mechanisms undergone by protein. This ranges from immunology to cell signaling and enzymology. There are many diseases that occur due to mutation that results in a change of protein structure like Parkinson's disease and Alzheimer's disease. By understanding the protein structures and their mechanism, we can develop drugs that have more efficiency. While protein structure determination techniques such as microarray analysis, RNA interference, and the yeast two-hybrid system can be used to experimentally demonstrate the function of a protein, advances in sequencing technologies have made the rate at which proteins can be experimentally characterized much slower than the rate at which new sequences become available. Thus, the annotation of new sequences can often be done quickly and for many genes or proteins at once by using computational methods.

3. Protein secondary structure

The secondary structure of a protein is due to the folding of the polypeptide chain into different folds due to hydrogen bonding and Vander Waal forces. The primary structure is very important in defining the structure and function of the protein. Amino acids join each other through peptide bonds which are rigid i.e., they do not allow rotation of the two amino acids freely. But the alpha carbon which is bonded with NH- and C=O group have some rotation which allows arranging amino acids at different angles in limited values.

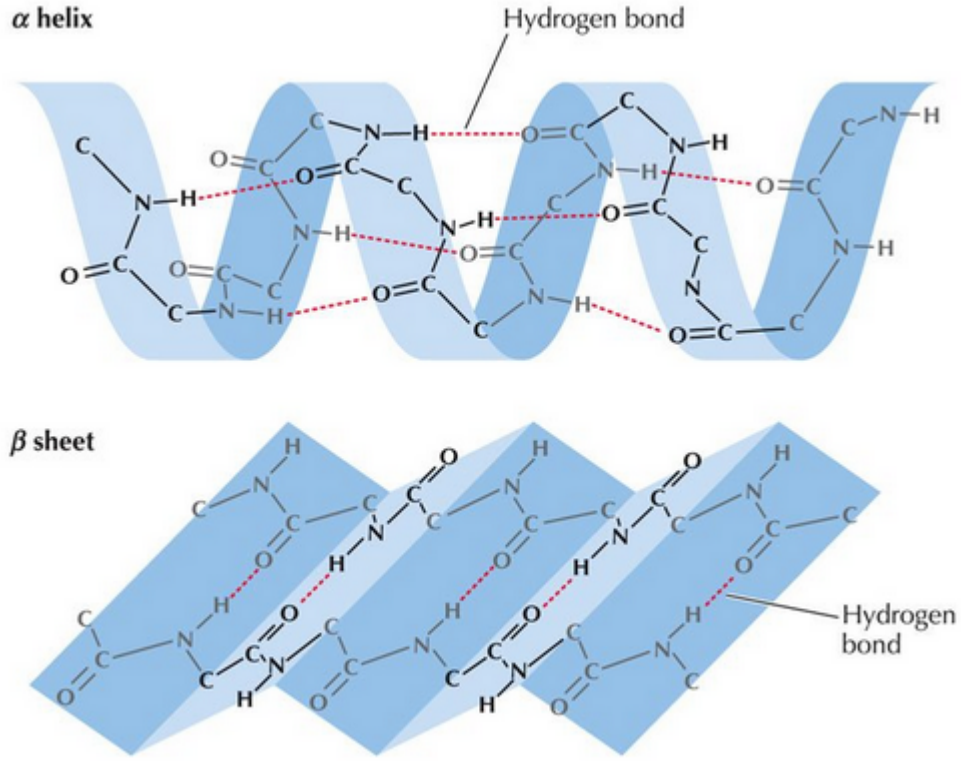
These angles are called torsion angles and help in the folding of the polypeptide chain into different secondary structure elements like α-helix, β-sheet, β pleated-sheet, and turns. These secondary structure elements are also stabilized by the forces present between amino acids located at some distance from each other.

3.1. α-Helix

The helical structure in the protein is one of the common secondary structures that exist. The α-helical structure is stabilized by the presence of the hydrogen bond formed between the peptide carbonyl group (C=O) and the peptide amide group (N-H) of the amino acid which is present four residues away. Each turn of the α-helix contains 3.6 amino acids and the helical structure rise along its axis to 5.4 Å.

3.2. β-Sheet

β-Sheets are formed when several β-strands self-assemble and are stabilized by interstrand hydrogen bonding, leading to the formation of extended amphipathic sheets in which hydrophobic side-chains point in one direction and polar side-chains in the other



(Structure of α-helix and β-Sheets)
figure 1.1

3.3. Loops and irregular elements

The third secondary structure which presents in the protein is the loop structure which joins the other secondary structure such as α-helix and strands of β-sheet. The loop structure consists of 2-6 amino acids. Coil structures are not true secondary structures but they are mostly classified as coil conformations. Coils are mostly located in a protein at places where amino acid residues do not form regular secondary structures such as α-helix or β-sheet.

4. Protein functions

The functions that proteins can perform are very varied and essential for the correct functioning of the organism, such as composing the cellular structures of living beings and making chemical reactions possible. Generally, function can be thought of as, "anything that happens to or through a protein".

5. Literature review for protein structure and function prediction

Proteins are not assembled into their native structures by a biological process, but folding is a purely physical process that depends only on the specific amino acid sequence of the protein.
There are two types of approaches: Physics-Based, Statistics Based. In physics-based, we determine protein structures by defining their potential energy at the molecular level and finding the structure that has the least amount of free energy. The methods include energy minimization, molecular dynamics, and simulated annealing. In the statistical method, physical forces acting on the molecules are not considered, we only look for the similarity and the frequency of the state like bond angles, rotamers, etc. The methods include homology-based and De novo based. In the statistical approach, we assume fixed geometry, discrete rotamers, and statistical potential.
Pauling and Corey [1] proposed in 1951 that the dominant secondary structural motifs were α-helices (hydrogen bonds between the ith and the i + 4th residues) and β-sheets (sequential hydrogen bonding between neighbouring segments rather than within a local segment).

Early methods used features derived from single-residue properties [2]. This was followed by the inclusion of neighbouring residues within a window, and later by a sequence evolution profile derived from multiple sequence alignment [3]. Sequence profiles, such as the PSSM from PSI-BLAST [4], contain conserved structural information across homologous sequences. Using sequence profiles was the main driving force for three-state accuracy going beyond 70% [5]. In particular, SPINE [6] achieved 80% by using classical neural networks trained on a large data set of 2640 non-redundant proteins with PSSM and physiochemical structural properties as its input.

Function identification from sequence resemblance can use several information sources to back up predictions based just on sequence identity levels, as well as to enhance outcomes in circumstances when the sequence homology is less than the recommended 40% identity confidence threshold by Wilson et al.[7], Todd et al.[8] and Devos & Valencia[9].

After identifying putative homologues, multiple sequence alignments allow for the classification of conserved residues, the research may provide important details about the family as a collective and the role of conserved regions, and evolutionary history can reveal whether an unknown protein cluster with a specific functional grouping.[10][11]

In principle, if an unfamiliar protein shares considerable sequence homology with a class of known function, and possesses the 'correct essential conserved residues', a function classification can be made.

CHAPTER 2

1. Dataset for structure prediction

The main dataset lists peptide sequences and their corresponding secondary structures. It is a transformation of https://cdn.rcsb.org/etl/kabschSander/ss.txt.gz downloaded from RSCB PDB into a tabular structure.

1. C: Loops and irregular elements (corresponding to the blank characters output by DSSP)
2. E: β-strand
3. H: α-helix
4. B: β-bridge
5. G: 3-helix
6. I: π-helix
7. T: Turn
8. S: Bend

For the purpose of secondary structure prediction, it is common to simplify the aforementioned eight states (Q8) into three (Q3) by merging (E, B) into E, (H, G, I) into H, and (C, S, T) into C.

1. pdb_id: the id used to locate its entry on https://www.rcsb.org/
2. chain_code: when a protein consists of multiple peptides (chains), the chain code is needed to locate a particular one.
3. seq: the sequence of the peptide
4. sst8: the eight-state (Q8) secondary structure
5. sst3: the three-state (Q3) secondary structure
6. len: the length of the peptide
7. hasnonstdaa: whether the peptide contains nonstandard amino acids (B, O, U, X, or Z).

	pdb_id	chain_code	seq	sst8	sst3	len	has_nonstd_aa
0	1A30	C	EDL	CBC	CEC	3	False
1	1B05	B	KCK	CBC	CEC	3	False
2	1B0H	B	KAK	CBC	CEC	3	False
3	1B1H	B	KFK	CBC	CEC	3	False
4	1B2H	B	KAK	CBC	CEC	3	False

(Columns in data)

Figure 2.1

	pdb_id	chain_code	seq	sst8	sst3	len	has_nonstd_aa
count	393732	393732	393732	393732	393732	393732.000000	393732
unique	139496	62	95915	329636	289917	NaN	2
top	5LUF	A	PIVQNLQGQMVHQAISPRTLNAWVKVVEEKAFSPEVIPMFSALSEG...	CCCSCCCCC	CCCCCCCCC	NaN	False
freq	62	133808	2580	285	720	NaN	386333
mean	NaN	NaN	NaN	NaN	NaN	260.212634	NaN
std	NaN	NaN	NaN	NaN	NaN	196.864409	NaN
min	NaN	NaN	NaN	NaN	NaN	3.000000	NaN
25%	NaN	NaN	NaN	NaN	NaN	131.000000	NaN
50%	NaN	NaN	NaN	NaN	NaN	223.000000	NaN
75%	NaN	NaN	NaN	NaN	NaN	336.000000	NaN
max	NaN	NaN	NaN	NaN	NaN	5037.000000	NaN

(Statistics about data)

Figure 2.2

2. Dataset for function prediction

This data set has been downloaded from RCSB PDB it contains data of protein sequences along with their specified functions.

structureId	classification	sequence
101M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
102L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSE...
102M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
103L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAK...
103M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...

(Columns in data)

Figure 2.3

StructureId: Used to identify and locate the sequence

Classification: denotes the type of function that the protein sequence carries out

Sequence: The sequence of the peptide

Data Pre-processing CHAPTER 3

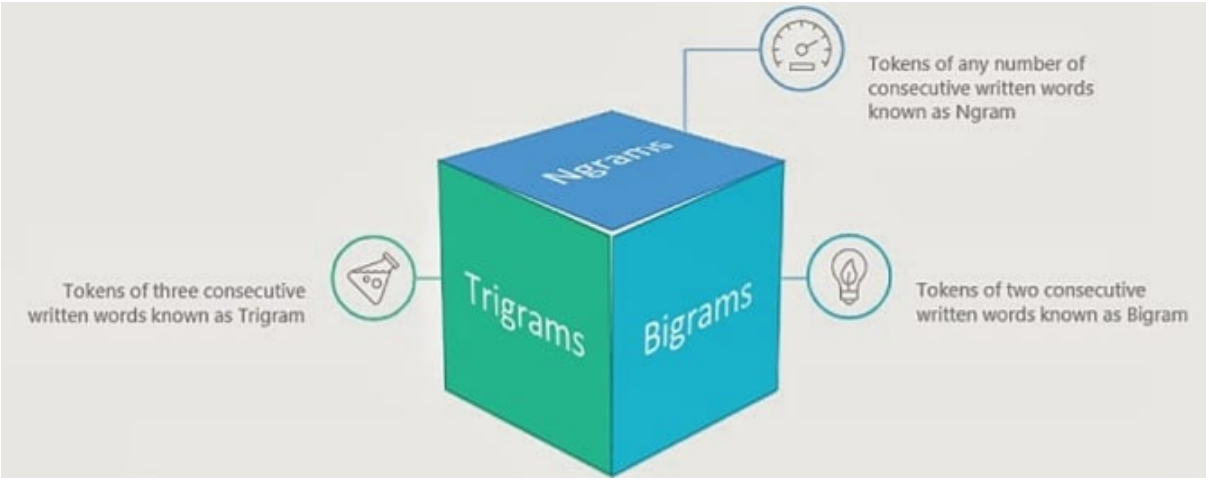
1. SEQUENCE PREPROCESSING

Pre-processing Sequence means, converting it into a format that is predictable and analyzable for the approach

1.1. Creating N-grams

They are basically a set of co-occurring words within a given window and when computing the n-grams you typically move one word forward

Given that a primary protein sequence can be treated as a string of amino acids, n-gram-based feature extraction methods can be applied to predict functionality from a sequence. n-grams from a protein sequence also offer biologically meaningful information, as each n-gram represents a protein sequence motif.



(N-Gram for words)

Figure 3.1

GSMTTFRIENVRIETINDFDMVKFDLVTDLGRVELAEHVNYDSEGD....

[GSM, SMT, MTT, TTF, TFR, FRI, RIE, IEN, ENV, ...

1.2. Tokenization

Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of a token as a part like a word is a token in a sentence, and a sentence is a token in a paragraph.

A process of breaking up text into many small pieces. Works by separating words using spaces and punctuation. Each of these smaller units is known as a token. We can use the tokenized form to determine the frequency of the word in the paragraph or text

We use tokens to convert the grams into a vector encoding the information of the n-gram in numerical

We have used this method to assign unique values to each n-gram we have created before and have converted the sequence data into machine-readable form (i.e.) numbers

1.3. Padding sequences

We include zeros to the data to keep all inputs consistent in shape. We have limited the length of the amino acid sequence to 100 in this model and we have also made the tokenized data in the equal size of a vector by adding zeros

```
array([[  89,    0,    0, ...,    0,    0,    0],
       [1913,    0,    0, ...,    0,    0,    0],
       [  47,    0,    0, ...,    0,    0,    0],
       ...,
       [  28, 1654, 1209, ...,    1,    0,    0],
       [1869,    5,    5, ...,  112,    0,    0],
       [ 120,   53,  262, ..., 2014,    0,    0]])
```

(Converted data from sequence after N gram, tokenization and padding)

figure 3.2

Shape before tokenizing and Padding

15550

Shape after tokenizing and Padding

(15550, 99)

1.4. One-hot encoding

One-hot encoding converts a letter into a vector in which a single element of the vector is set. This encoding creates a distinct feature that can be used mathematically—for example, each letter represented gets its own weight applied within the network. While in this implementation, 1 represent letters through one-hot

Human-Readable Machine-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

(Example of one hot encoding)

figure 3.3

we have used this method to convert our output data i.e the predicted structure data from C, E, H to an array of 1s and 0s.

```
array([[[0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        ...,
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.]],

       [[0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
```

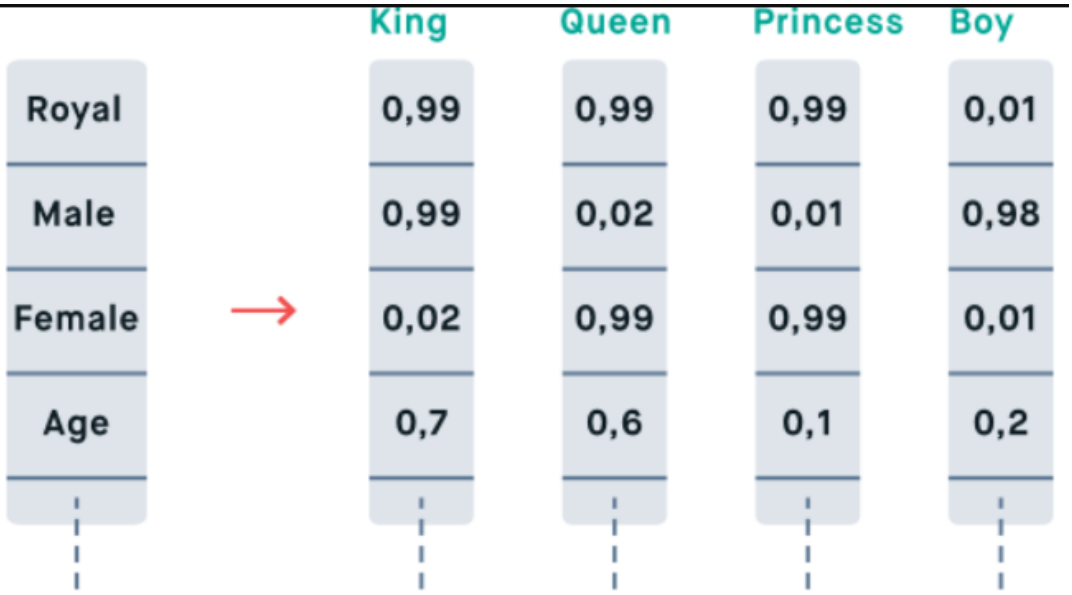
(Converted data after one-hot encoding)
figure 3.4

1.5. Word Embeddings

The geometric relationship between words in word embeddings can represent the semantic relationship between words. Words closer to each other have a strong relation compared to words away from each other.

Vectors/words closer to each other means the cosine distance or geometric distance between them is less compared to others.

There could be a vector “male to female” which represents the relation between a word and its feminine. That vector may help us in predicting “king” when “he” is used and “Queen” when she is used in the sentence.



(Example for word embeddings)
Figure 3.5

1.6. CountVectorizer

Machines cannot understand characters and words. So when dealing with text data we need to represent it in numbers to be understood by the machine. Countvectorizer is a method to convert text to numerical data.

Example

```
text = ['Hello my name is james, this is my python notebook']
```

The text is transformed into a sparse matrix as shown below.

	hello	is	james	my	name	notebook	python	this
0	1	1	1	1	1	0	0	0
1	0	1	0	1	0	1	1	1

(Example for countVectorizer)
Figure 3.6

2. SPLITTING DATA

2.1. Train Set:

The train set would contain the data which will be fed into the model. In simple terms, our model would learn from this data.

2.2. Test Set:

The test set contains the data on which we test the trained and validated model. It tells us how efficient our overall model is and how likely is it going to predict something which does not make sense.

Structure prediction splitting
30 % of the data is taken as the test set.
Shape before splitting
(15550, 99)
Shape after splitting
((10885, 99), (10885, 99, 4), (4665, 99), (4665, 99, 4))

Function prediction splitting
20 % of the data is taken as the test set.
Total size of the data-278,866
Size of training data-223,093
Size of test data-55,773

2.3. Validation Set:

The validation set is used to fine-tune the hyperparameters of the model and is considered a part of the training of the model. The model only sees this data for evaluation but does not learn from this data, providing an objective unbiased evaluation of the model.

MACHINE LEARNING CHAPTER 4

1. Introduction to ML

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory, and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain In its application across business problems, machine learning is also referred to as predictive analytic

2. Algorithm

Naive Bayes: A Naive Bayes classifier is a collection of many algorithms where all the algorithms share one common principle, and that is each feature being classified is not related to any other feature. The presence or absence of a feature does not affect the presence or absence of the other feature.

AdaBoost: AdaBoost is an ensemble learning method (also known as “meta-learning”) that was initially created to increase the efficiency of binary classifiers. AdaBoost uses an iterative approach to learn from the mistakes of weak classifiers, and turn them into strong one's

Decision Tree: It is a graphical representation of all the possible solutions to a decision based on certain conditions. Tree models where the target variable can take a finite set of values are called classification trees and the target variable can take continuous values (numbers) are called regression trees.

NEURAL NETWORK CHAPTER 5

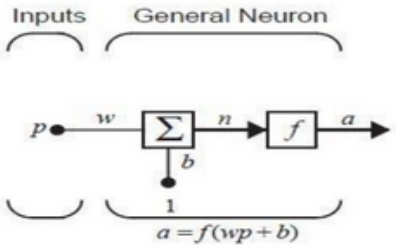
1. IMPORTANCE OF NEURAL NETWORKS

Neural networks can be applied in many fields with the advancement of technology. Some of the fields in which neural networks can be used are Aerospace, Automotive, Banking, Electronics, Entertainment, Financial, Insurance, Manufacturing, Medical, Oil and Gas, Robotics, Speech, Security, Telecommunications, and Transportation. With the advancement in computers and algorithms which are faster, it became easy in solving complex industrial problems which previously required much computation.

2. MODEL OF THE NEURON

The structure of a single input neuron is shown in which two scalar inputs w, p are multiplied together and this product is sent into the summer and bias b which is much like a weight is multiplied by an input 1, and this product this also sent into the summer. The output from the summer is called “net input”. This net input is further sent into an activation function which gives an output that is scalar in nature. This scalar output is denoted by “a”. We can correlate this model with biological neurons and the observations are given as follows:

The synaptic strength is denoted by the” w” Summation and transfer function in the neuron model is used to depict the function of the cell body Single on the axon is represented by network output “a” According to our requirement of output the type of transfer function can be chosen by us. The scalar parameters of neurons as said above “w”, “b” can be adjusted according to our requirement. In order to meet a specific goal, the transfer function can be chosen by the designer, and the scalar parameters “w”, “b” are adjusted by using learning rules. There are different types of transfer functions that can be used for different purposes to meet a specified goal. The transfer function can be a linear or nonlinear function of “n”. Different transfer functions are given below



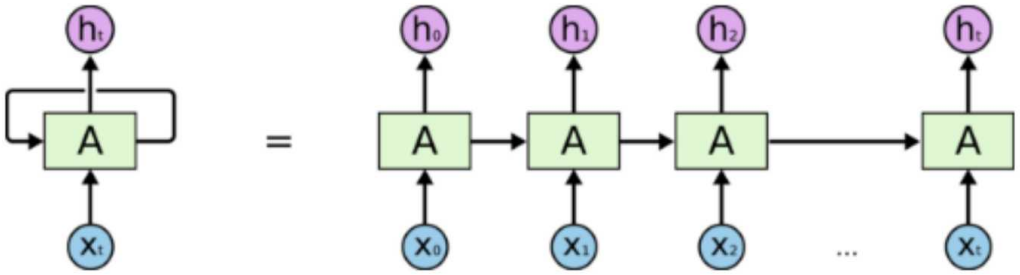
(Single neuron model)

figure 4.1

3. Modules

3.1. About RNN

A recurrent neural network (RNN) is a form of artificial neural network that works with time series or sequential data. Recurrent neural networks are characterized by their "memory," which allows them to influence current input and output by using information from previous inputs. Recurrent neural networks' performance is dependent on the sequence's prior elements. Although future events can be useful in deciding a sequence's performance, unidirectional recurrent neural networks cannot account for them in their predictions.



An unrolled recurrent neural network.

(An unrolled recurrent neural network)

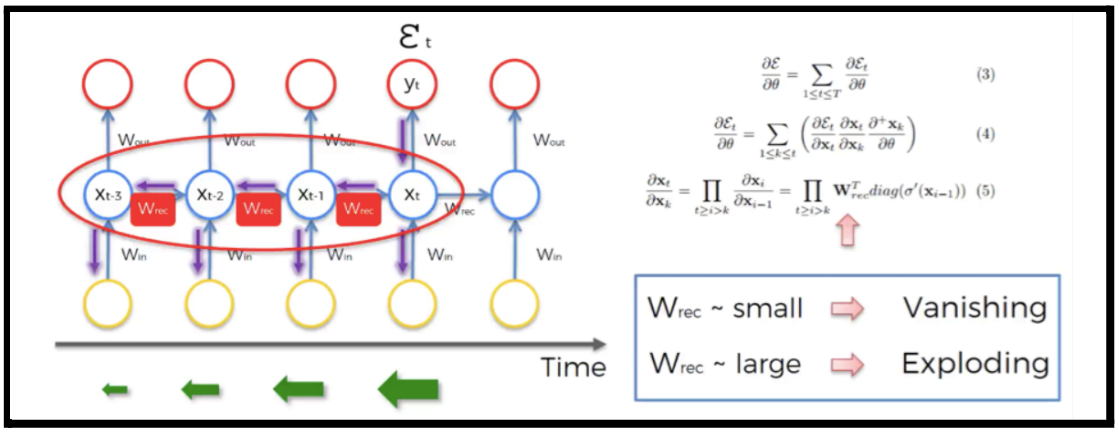
Figure 4.2

3.1.1. Principle

To calculate the gradients, recurrent neural networks use the backpropagation through time (BPTT) algorithm. Traditional backpropagation uses the same concepts as BPTT, in which the model trains itself by measuring errors from its output layer to its input layer. These calculations allow us to correctly modify and match the model's parameters.

3.1.2. Concerns

Exploding gradients and vanishing gradients are two issues that RNNs often encounter. The scale of the gradient, which is the slope of the loss function along the error curve, defines these problems. When the gradient is too small, it gets smaller and smaller, updating the weight parameters until they are insignificant—that is, zero.



(Vanishing and exploding gradient)

Figure 4.3

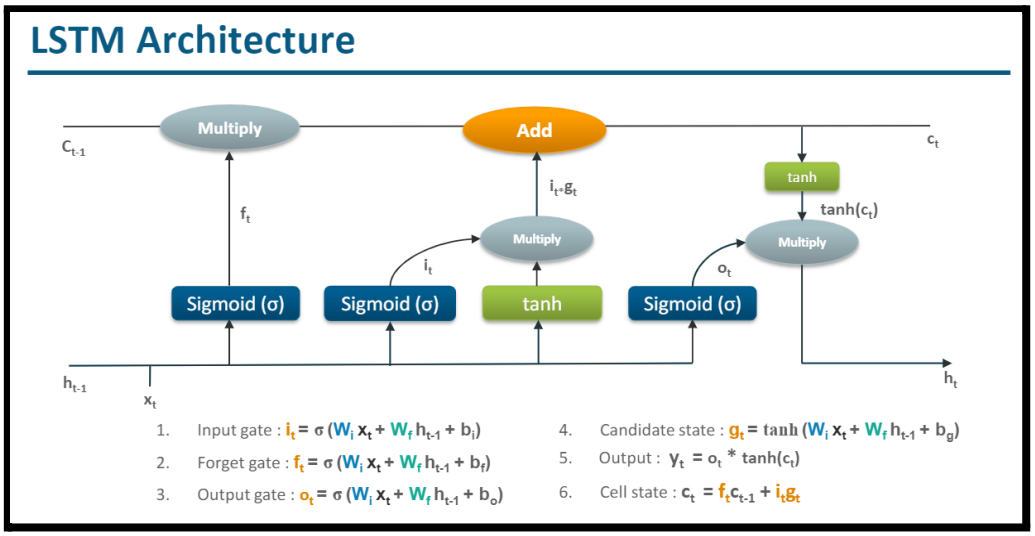
When this happens, the algorithm stops learning. When a gradient is too high, it explodes, resulting in an unstable model. The model weights will become too high in this scenario, and

they will finally be interpreted as NaN. One way to address these problems is to reduce the number of hidden layers in the neural network, removing some of the RNN model's complexity.
In our project, we used LSTM (Long Short-Term Memory) to overcome this problem which uses tensor operations called the gates.

3.2. About LSTM

The Long Short Term Memory Network (LSTMN) is an advanced RNN (sequential network) that allows information to be stored indefinitely. It can deal with the vanishing gradient problem that RNN has.

LSTM consist of three gates mainly Forget gate, Input gate, and Output gate



(LSTM Architecture)

Figure 4.4

Forget gate:
Forget gate decides what information should be retained or removed from memory. This is controlled by a sigmoid function.

For time step (t), passing input (x_t) with the previous hidden state (h_{t-1}) to forget gate

- Returns 0 if the information must be removed
- Returns 1 if the information must be retained

Input gate:
Input gate (i_t) decides what information should be stored in memory. It is controlled by a sigmoid function. It returns 1 if the information must be stored in a cell state.

Updating the cell:
The creation of a new vector (g_t) is required to hold the new information that is added to the memory. This is also known as a candidate state or internal state vector.
It is controlled by the tanh function, unlike the sigmoid function. Tanh returns values in the range of
-1 to 1.

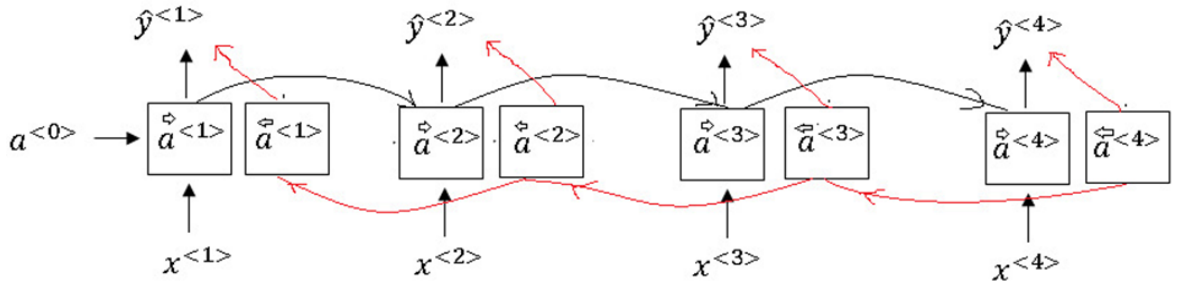
Output gate:
Output gate (o_t) decides what information should be taken from memory to provide as an output. This is controlled by the sigmoid function returns 1 if the information must be taken from the cell state to provide as output.

3.3. Bidirectional LSTM

In bidirectional LSTM, instead of training a single model, we introduce two. The first model learns the sequence of the input provided, and the second model learns the reverse of that sequence.

Since we do have two models trained, we need to build a mechanism to combine both. It is usually referred to as the Merge step. Merging can be one of the following functions:

A NN must accept the amino acid at its input with all the necessary information that it needs in order to produce the right output. Taking into account that the formation of different secondary structural elements depends on the interaction between neighboring-in-space (not necessarily in sequence) amino acid residues, we chose the BRNN architecture for its ability to encapsulate information included in the amino acid residues that are coming before and after the residue at the examined position



(Bidirectional LSTM)

Figure 4.5

3.3.1. Model

Optimizers:

Optimization algorithms or strategies are responsible for reducing the losses and providing the most accurate results possible. It is a search technique used to update weights in our model.

We have

- RMS prop - adaptive learning rate optimization method which utilizes the magnitude of recent gradients to normalize the gradients.

Objective Function:

The loss function/objective function is the evaluation of the model used by the optimizer to navigate the weight space

- categorical_crossentropy - for multi-class logarithmic loss (log loss)

Importing the library and accessing the dataset that has the required data

In [16]:
```
#importing the library and accessing the dataset that has the required data
import pandas as pd
data = pd.read_csv("2018-06-06-ss.cleaned.csv")
data.head()
```

Out[16]:

	pdb_id	chain_code	seq	sst8	sst3	len	has_nonstd_aa
0	1A30	C	EDL	CBC	CEC	3	False
1	1B05	B	KCK	CBC	CEC	3	False
2	1B0H	B	KAK	CBC	CEC	3	False
3	1B1H	B	KFK	CBC	CEC	3	False
4	1B2H	B	KAK	CBC	CEC	3	False

Checking various data columns from the dataset

In [17]:
```
#checing various data columns from the dataset
data.describe(include = "all")
```

Out[17]:

	pdb_id	chain_code	seq	sst8	sst3	len	has_nonstd_aa
count	393732	393732	393732	393732	393732	393732.000000	393732
unique	139496	62	95915	329636	289917	NaN	2
top	5LUF	A	PIVQNLQGQMVHQAISPRTLNAWVKVVEEKAFSPEVIPMFSALSEG...	CCCSCCCCC	CCCCCCCCC	NaN	False
freq	62	133808	2580	285	720	NaN	386333
mean	NaN	NaN	NaN	NaN	NaN	260.212634	NaN
std	NaN	NaN	NaN	NaN	NaN	196.864409	NaN
min	NaN	NaN	NaN	NaN	NaN	3.000000	NaN
25%	NaN	NaN	NaN	NaN	NaN	131.000000	NaN
50%	NaN	NaN	NaN	NaN	NaN	223.000000	NaN
75%	NaN	NaN	NaN	NaN	NaN	336.000000	NaN
max	NaN	NaN	NaN	NaN	NaN	5037.000000	NaN

Checking various data columns from the dataset

In [18]:
```
#getting the data that has only standard amino acids
df = data[data["has_nonstd_aa"] == False]
```

In [19]:
```
#counting the number of samples present
df.count()
```

Out[19]:
```
pdb_id          386333
chain_code      386333
seq             386333
sst8            386333
sst3            386333
len             386333
has_nonstd_aa   386333
dtype: int64
```

This is the dataset that we are using for this model after the initial pre-processing

In [20]:
```
# This is the dataset that we are using or this model
df
```

Out[20]:

	pdb_id	chain_code	seq	sst
0	1A30	C	EDL	CB
1	1B05	B	KCK	CB
2	1B0H	B	KAK	CB
3	1B1H	B	KFK	CB
4	1B2H	B	KAK	CB
...	...	...	...	
393719	5NUG	B	MSEPGGGGGEDGSAGLEVSAVQNVADVSVLQKHLRKLVPLLLEDGG...	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
393728	5J8V	A	MGDGGEGEDEVQFLRTDDEVVLQCSATVLKEQLKLCLAAEGFGNRL...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE
393729	5J8V	B	MGDGGEGEDEVQFLRTDDEVVLQCSATVLKEQLKLCLAAEGFGNRL...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE
393730	5J8V	C	MGDGGEGEDEVQFLRTDDEVVLQCSATVLKEQLKLCLAAEGFGNRL...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE
393731	5J8V	D	MGDGGEGEDEVQFLRTDDEVVLQCSATVLKEQLKLCLAAEGFGNRL...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE

386333 rows × 7 columns

```
In [20]: # This is the dataset that we are using or this model
         df
```

Out[20]:

eq	sst8	sst3	len	has_nonstd_aa
)L	CBC	CEC	3	False
:K	CBC	CEC	3	False
\K	CBC	CEC	3	False
:K	CBC	CEC	3	False
\K	CBC	CEC	3	False
...	...	...	...	...
...	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC...	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC...	4646	False
...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE...	CCCCCCCCCCCCCCCCCCCCEEEECCEECCEECCEECCEEECCEEE...	5037	False
...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE...	CCCCCCCCCCCCCCCCCCCCEEEECCEECCEECCEECCEEECCEEE...	5037	False
...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE...	CCCCCCCCCCCCCCCCCCCCEEEECCEECCEECCEECCEEECCEEE...	5037	False
...	CCCCCCCCCCCCCCCSSSCCEEEECSEETTEECCEECCEEETTEEE...	CCCCCCCCCCCCCCCCCCCCEEEECCEECCEECCEECCEEECCEEE...	5037	False

We are creating a dataset with only the required attributes for predicting the protein secondary structure. This includes the sequence of the amino acid, the length of the protein, and the structure in the three-state secondary structure and we are removing the duplicates

```
In [22]: # we are creating a dataset with only the required attributes for predicting the protein secondary structure. This include the
         # sequence of the amino acid, the lenght of the protein, and the structure in three state secondary structure
         df = df[['seq','len','sst3']][(df['len']<100)]
         df.drop_duplicates(subset='seq',inplace=True)
         df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 15550 entries, 0 to 60162
         Data columns (total 3 columns):
          #   Column  Non-Null Count  Dtype
         ---  ------  --------------  -----
          0   seq     15550 non-null  object
          1   len     15550 non-null  int64
          2   sst3    15550 non-null  object
         dtypes: int64(1), object(2)
         memory usage: 485.9+ KB
```

from the dataset.

```
In [23]: # Here we are breaking the whole sequence into trigram where the whole sequence is divided into parts of 3 and are associated
         # with its respective structure for the 3 amino acids
         def ngrams(seq,n=3):
             return ([seq[i:i+n] for i in range(len(seq)-n+1)])

         df['seqss']=df['seq'].apply(ngrams)
         df
```

Out[23]:

	seq	len	sst3	seqss
0	EDL	3	CEC	[EDL]
1	KCK	3	CEC	[KCK]
2	KAK	3	CEC	[KAK]
3	KFK	3	CEC	[KFK]
5	KMK	3	CEC	[KMK]
...	...	...	...	...
60095	GSMTTFRIENVRIETINDFDMVKFDLVTDLGRVELAEHVNYDSEGD...	99	CCCCCEEEEEEEEEEEEECCEEEEEEEEEEECCEEEEEEEEEEEECCCCC...	[GSM, SMT, MTT, TTF, TFR, FRI, RIE, IEN, ENV, ...
60139	MKRALGFLSLLVIFASLLVALSPEYGIKFGVGGEDWLKYRYTDNYY...	99	CHHHHHHHHHHHHHHHHHHHHCCCCCCCCCCECCCHHHCCHHHHHH...	[MKR, KRA, RAL, ALG, LGF, GFL, FLS, LSL, SLL, ...
60142	GSGFVNKDQIAKDVKQFYDQALQQAVVDDDANNAKAVVKTFHETLD...	99	CCCCCCHHHHHHHHHHHHHHHHHHHHHHHHCCCCHHHHHHHHHHHHHHC...	[GSG, SGF, GFV, FVN, VNK, NKD, KDQ, DQI, QIA, ...
60149	MAAAAASRAVGAKLGLREIRVHLCQRSPGSQGVRDFIVQRYVELKK...	99	CCCCCCCCCCCCCCCCCCEEEEEECCCHHHCCHHHHHHHHHHHHHHH...	[MAA, AAA, AAA, AAA, AAS, ASR, SRA, RAV, AVG, ...

We are importing TensorFlow and Keras libraries. we are tokenizing the sequence. And we do padding to the data so that all the data are of the same size.

In [24]:
```
# we are trying to get the maximum number of amino acid present in a protein in the new dataset
def max_length(series):
    l = []
    [l.append(len(s)) for s in series]
    return max(l)

maxlen = max_length(df['seq'])
maxlen
```

Out[24]: 99

In [25]:
```
# we  are importing tensorflow and keras libraries. we are tokenizing the sequence. And we do padding to the data so that
# all the data are of the same size
import tensorflow as tf
from tensorflow.keras.models import Sequential
from keras.preprocessing import text, sequence
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

tok_x = Tokenizer(lower=False)
tok_x.fit_on_texts(df['seqss'].values)
x = tok_x.texts_to_sequences(df['seqss'].values)
x = sequence.pad_sequences(x, maxlen=maxlen, padding='post')

tok_y = Tokenizer(char_level=True)
tok_y.fit_on_texts(df['sst3'].values)
y = tok_y.texts_to_sequences(df['sst3'].values)
y = sequence.pad_sequences(y, maxlen=maxlen, padding='post')
y = to_categorical(y)

x.shape, y.shape
```

Out[25]: ((15550, 99), (15550, 99, 4))

We are splitting the data into 2 parts: training data and test data. Training data is used to train the model and the test data is used to test the model with unseen data. We are splitting in a way that 30% of the data goes to test data and the rest into training data.

We can see that the data has been split into two: training data with 10885 samples and test data with 4665

In [26]:
```
# we are splitting the data into 2 parts: training data and test data. Training data is used to traint the model and the test
# data is used to test the model with unseen data. we are splitting in a way that 30% of the data goes to test data and the rest
# into training data
from sklearn.model_selection import train_test_split

x_tr, x_ts, y_tr, y_ts = train_test_split(x,y,test_size=0.3, random_state=33)
x_tr.shape, y_tr.shape, x_ts.shape, y_ts.shape
```

Out[26]: ((10885, 99), (10885, 99, 4), (4665, 99), (4665, 99, 4))

In [27]:
```
from keras.models import Sequential
from keras.layers import Embedding, Dense, TimeDistributed, Bidirectional, LSTM, CuDNNLSTM
n_ngrams = len(tok_x.word_index) + 1
n_tags = len(tok_y.word_index) + 1
```

Here, we are building the bidirectional LSTM model and giving the hyperparameters for the model to run on

In [28]:
```
# we are building the bidirectional LSTM model and giving the hyperparameters for the model to run on
lstm = Sequential()
lstm.add(Embedding(input_dim=n_ngrams,output_dim=x.shape[0]//12,input_length=maxlen))
lstm.add(Bidirectional(CuDNNLSTM(x.shape[0]//24,return_sequences=True)))
lstm.add(TimeDistributed(Dense(n_tags,input_dim=y.shape[2],activation='softmax')))
lstm.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 99, 1295)          10358705

 bidirectional_1 (Bidirectio  (None, 99, 1294)         10062144
 nal)

 time_distributed_1 (TimeDis  (None, 99, 4)            5180
 tributed)

=================================================================
Total params: 20,426,029
Trainable params: 20,426,029
Non-trainable params: 0
_________________________________________________________________
```

We are now training the above model with the training data and having the metric as accuracy to measure the model.

We are then turning 30% of the training data into a validation set.

A validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning the model's hyperparameters.

The validation dataset is different from the test dataset that is also held back from the training of the model, but is instead used to give an unbiased estimate of the skill of the final tuned model when comparing or selecting between final models.

In [29]:
```
# we are now the training the above model with the training data and having the metric as accuracy
lstm.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['acc'])
lstm.fit(x_tr,y_tr,epochs=10,validation_split=0.3,verbose=1)
```

```
Epoch 1/10
239/239 [==============================] - 44s 121ms/step - loss: 0.4620 - acc: 0.8080 - val_loss: 0.4507 - val_acc: 0.7878
Epoch 2/10
239/239 [==============================] - 31s 131ms/step - loss: 0.3093 - acc: 0.8723 - val_loss: 0.3750 - val_acc: 0.8498
Epoch 3/10
239/239 [==============================] - 42s 174ms/step - loss: 0.2567 - acc: 0.8968 - val_loss: 0.3478 - val_acc: 0.8622
Epoch 4/10
239/239 [==============================] - 45s 190ms/step - loss: 0.2163 - acc: 0.9145 - val_loss: 0.3551 - val_acc: 0.8622
Epoch 5/10
239/239 [==============================] - 51s 212ms/step - loss: 0.1827 - acc: 0.9287 - val_loss: 0.3481 - val_acc: 0.8705
Epoch 6/10
239/239 [==============================] - 53s 221ms/step - loss: 0.1533 - acc: 0.9411 - val_loss: 0.4226 - val_acc: 0.8585
Epoch 7/10
239/239 [==============================] - 54s 224ms/step - loss: 0.1280 - acc: 0.9515 - val_loss: 0.4157 - val_acc: 0.8685
Epoch 8/10
239/239 [==============================] - 60s 252ms/step - loss: 0.1069 - acc: 0.9599 - val_loss: 0.4539 - val_acc: 0.8699
Epoch 9/10
239/239 [==============================] - 55s 230ms/step - loss: 0.0910 - acc: 0.9663 - val_loss: 0.4651 - val_acc: 0.8693
Epoch 10/10
239/239 [==============================] - 58s 242ms/step - loss: 0.0779 - acc: 0.9717 - val_loss: 0.5091 - val_acc: 0.8723
```

Out[29]:
```
<keras.callbacks.History at 0x1fdb0bb3be0>
```

In [30]:
```
# we are now checking the accuracy with which the model can predict the test data. And we got the accuracy as 0.8711
evals = lstm.evaluate(x_ts,y_ts)
print('accuracy:', evals[1])
```

```
146/146 [==============================] - 8s 53ms/step - loss: 0.5256 - acc: 0.8696
accuracy: 0.8695963025093079
```

Here we are importing the library and dataset required for function prediction

•[20]:
```
#Here we are Importing Libraries and Datasets
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

•[21]:
```
# Import Datasets
df_seq=pd.read_csv('archive/pdb_data_seq.csv')
df_char=pd.read_csv('archive/pdb_data_no_dups.csv')
```

After importing the dataset we are modifying the data to make it suitable for function prediction of protein.

```
[22]: # Filter for only proteins
protein_char = df_char[df_char.macromoleculeType == 'Protein']
protein_seq = df_seq[df_seq.macromoleculeType == 'Protein']

# Select only necessary variables to join
protein_char = protein_char[['structureId','classification']]
protein_seq = protein_seq[['structureId','sequence']]
protein_seq.head()
```

[22]:

	structureId	sequence
4	101M	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
7	102L	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSE...
8	102M	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
11	103L	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAK...
12	103M	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...

Joining the both data sets of Function an sequence using structure ID

```
[23]: # Join two datasets on structureId
model_f = protein_char.set_index('structureId').join(protein_seq.set_index('structureId'))
```

```
[24]: model_f.head()
```

[24]:

structureId	classification	sequence
101M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
102L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSE...
102M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
103L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAK...
103M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...

dropping the rows with missing values

```
[25]: # Drop rows with missing values
model_f = model_f.dropna()
```

Counting the number of samples present in each type of function

Getting the functions that has more than 1000 samples for classification

```
[27]: # Get classification types where counts are over 1000
types = np.asarray(counts[(counts > 1000)].index)
```

creating a dataset that has function types which has more than 1000 samples and the final size of the dataset we get is 278,866

```
[28]: # Filter dataset's records for classification types > 1000
      data_f = model_f[model_f.classification.isin(types)]

[29]: data_f.shape[0]

[29]: 278866
```

Splitting the dataset into train and test data set

```
[30]: # Split Data
      X_train, X_test,y_train,y_test = train_test_split(data_f['sequence'], data_f['classification'], test_size = 0.2, random_state = 1)
```

Creating a Countvectorizer

```
[31]: # Create a Count Vectorizer to gather the unique elements in sequence
      vect = CountVectorizer(analyzer = 'char_wb',ngram_range = (4,4))
```

Fitting the data and transforming training and test dataset into vectorized form

```
[32]: # Fit and Transform CountVectorizer
      vect.fit(X_train)
      X_train_df = vect.transform(X_train)
      X_test_df = vect.transform(X_test)
```

Samples of features present in the countvectorizer

```
[33]: #Print a few of the features
      print(vect.get_feature_names()[-20:])
```

```
['zhhh', 'ziar', 'zigi', 'ziwz', 'zkal', 'zkky', 'zknt', 'zkyh', 'zlik', 'zlz
k', 'zpvm', 'zrgd', 'zrvi', 'ztvl', 'ztzk', 'zvbd', 'zvib', 'zvka', 'zwdl', '
zzvb']
```

Making a dictionary that stores all the accuracy values of different models and training the model with Multinomial naive Bayes algorithm and we got an accuracy of 76.385%

```
[34]: # Make a prediction dictionary to store accuracys
      prediction = dict()
      # Naive Bayes Model
      from sklearn.naive_bayes import MultinomialNB
      model = MultinomialNB()
      model.fit(X_train_df, y_train)
      NB_pred = model.predict(X_test_df)
      prediction["MultinomialNB"] = accuracy_score(NB_pred, y_test)
      print( prediction['MultinomialNB'])
```

```
0.7638505396779861
```

Training the dataset with adaboost algorithm and we got a accuracy of 18.61%

```
•[35]: # Adaboost
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier()
model.fit(X_train_df,y_train)
ADA_pred = model.predict(X_test_df)
prediction["Adaboost"] = accuracy_score(ADA_pred , y_test)
print(prediction["Adaboost"])
```

```
0.18614408147165346
```

Training the dataset with decision tree algorithm and got a accuracy of 88.64%

```
•[36]: # Make a prediction dictionary to store accuracys
prediction = dict()
# Decision Tree
from sklearn import tree
model = tree.DecisionTreeClassifier()
model.fit(X_train_df, y_train)
DT_pred = model.predict(X_test_df)
prediction["decision_tree"] = accuracy_score(DT_pred, y_test)
print( prediction['decision_tree'])
```

```
0.8863807508875103
```

Here we are making a function to predict the function of the sequence given using the pre-trained model (here the model of the decision tree is used as it shows the higher accuracy) it takes the data frame of the sequence and returns its predicted function

```
•[53]: # Function for predicting function
def function_predict(target):
    query_seq = vect.transform(target)
    fun_pred = model.predict(query_seq)
    return(print("Function of protein: ",fun_pred))
```

The function struct_and_funct_predict takes the sequence and converts it into a dataframe and passes the data frame to the function function_predict to predict its function and it also transforms the data frame to make it suitable for prediction then the transformed data is used for prediction of the structure after the predictions the data is decoded so that we can understand.

```
•[52]: # Function for predicting both structure and function
from keras.preprocessing import text, sequence
import numpy as np
def struct_and_funct_predict(value):
    seq_test =  {'seq':[value]}
    target = pd.DataFrame(seq_test)
    function_predict(target)
    target['seqss']= target['seq'].apply(ngrams)
    token = Tokenizer(lower=False)
    token.fit_on_texts(df_s['seqss'].values)
    t_seq = token.texts_to_sequences(target['seqss'].values)
    t_seq = sequence.pad_sequences(t_seq, maxlen= 99, padding='post')
    res = (lstm.predict(t_seq)).round()
    ans=np.where(res == 1)
    g=[]
    string1=""
    for i in ans[2]:
        if i == 1:
            g.append("c")
        elif i==2:
            g.append("h")
        elif i==3:
            g.append("e")
    for i in g:
        string1 = string1 + i

    return(print("secondary structure of protein: ",string1))
```

we are calling the function that we made above by passing the sequence to which we need to predict the structure and function. The resultant structure and function predictions made for the sequence are printed below.

```
•[55]: # Enter a sequence to get its secondary structure and its function
value =  "MASPQLQMALDGFKMMGEKMAQAGGDVKAMRAVMEEMATFPSAGETKCTPVNAGGVPAEWIAAPGAAD
struct_and_funct_predict(value)
```

```
Function of protein:  ['HYDROLASE']
secondary structure of protein:  ccccccccccccccccccccccccceeeeeecchheeeehhh
hhhhhhhhhcccehhheehhccceeeeccccccccchhhhhhhhhhcccccccccc
```

Result

Performance on the training set of structure prediction gives a loss of 0.0779 and an accuracy of 97.17%. Performance on the validation set gives an accuracy of 87.23 therefore the data is not overfitting the data and the accuracy on test data is 86.96%. The current accuracy for three-state secondary structure prediction is about ~85%. The exact number depends on the particular test dataset used. And in the function prediction model, we used three types of algorithm- Multinomial naive bayes, Adaboost, and decision tree. In this, we got the highest accuracy in the decision tree algorithm of 88.64%.

So, the function struct_and_funct_predict can predict the structure and function of the given protein sequence with an accuracy of 85% and 88.64% respectively.

Future Outlook

Predicting the secondary structure of the protein is just a baby step into the journey of predicting the final structure of the protein, protein interaction, etc. even the prediction of secondary structure seems to have evolved much with the addition of MSA to the prediction pipeline. Thus, the prediction of secondary structure is just a base for protein structure prediction and harnessing the power of proteins in the future. The tertiary structure prediction involves the use of MSA and DEEP MSA is the current standard tool for creating MSA and it involves predicting contact maps/distograms/organograms from the MSA. Using transformer networks with attention in place of RNN has been shown to capture more accurate long-range interaction between the protein residues.

References

[1] Pauling, Linus, and Robert B. Corey. "Configurations of polypeptide chains with favored orientations around single bonds: two new pleated sheets." Proceedings of the National Academy of Sciences of the United States of America 37.11 (1951): 729.

[2] Finkelstein, A. V., and O. B. Ptitsyn. "Statistical analysis of the correlation among amino acid residues in helical, β-stractural and non-regular regions of globular proteins." *Journal of molecular biology* 62.3 (1971): 613-624.

[3] Zvelebil, Markéta J., et al. "Prediction of protein secondary structure and active sites using the alignment of homologous sequences." *Journal of molecular biology* 195.4 (1987): 957-961.

[4] Altschul, Stephen F., et al. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." *Nucleic acids research* 25.17 (1997): 3389-3402.

[5] Rost, Burkhard, and Chris Sander. "Improved prediction of protein secondary structure by use of sequence profiles and neural networks." *Proceedings of the National Academy of Sciences* 90.16 (1993): 7558-7562.

[6] Dor, Ofer, and Yaoqi Zhou. "Achieving 80% ten‐fold cross‐validated accuracy for secondary structure prediction by large‐scale training." *Proteins: Structure, Function, and Bioinformatics* 66.4 (2007): 838-845.

[7] Wilson, Cyrus A., Julia Kreychman, and Mark Gerstein. "Assessing annotation transfer for genomics: quantifying the relations between protein sequence, structure, and function through traditional and probabilistic scores." *Journal of molecular biology* 297.1 (2000): 233-249.

[8] Todd, Annabelle E., Christine A. Orengo, and Janet M. Thornton. "Evolution of protein function, from a structural perspective." *Current opinion in chemical biology* 3.5 (1999): 548-556.

[9] Devos, Damien, and Alfonso Valencia. "Practical limits of function prediction." *Proteins: Structure, Function, and Bioinformatics* 41.1 (2000): 98-107.

[10] Hannenhalli, Sridhar S., and Robert B. Russell. "Analysis and prediction of functional sub-types from protein sequence alignments." *Journal of molecular biology* 303.1 (2000): 61-76.

[11] Gu, Xun, and Kent Vander Velden. "DIVERGE: phylogeny-based analysis for functional–structural divergence of a protein family." *Bioinformatics* 18.3 (2002): 500-501.