Faculty of Computers and Artificial intelligence
Department of Computing and Bioinformatics

جامعة القاهرة
كلية الحاسبات والذكاء الاصطناعي
قسم المعلوماتية الحيوية

# Predicting Covid-19 protein-protein interactions

Covid-19 Protein Protein Interaction Prediction

Supervised by

*Dr. Hanaa Bayomi*

*Dr. Ahmed Farouk*

*TA. Eng. Sara El-Nady*

Implemented by

*Mohamed Mahmoud Ali*
*Tarek Abd Al-Majed Omar*
*Mohamed Abd Al-Majed Omar*
*Hasnaa Ali Hassan*
*Mennatollah Mabrouk Abd El-Hamid*

Graduation Project
Academic Year 2021-2022
Documentation

**Table of content**

**List of figures:**

**List of tables:**

**List of abbreviation**

**LR:** Logistic Regression

**RF:** Random Forest

**RNN:** Recurrent Neural Network

**SVM:** Support Vector Machine

**ML:** Machine Learning

**DL:** Deep Learning

**AL:** Algorithm

**BLAST:** Basic Local Alignment Several Tool

**EIIP:** Electron-Ion Interaction Potential

**CPNR:** Complex Prime Number Representation

**PPI:** Protein-Protein Interaction

# Chapter 1 Introduction

## 1.1 Motivation

The genomic structure of the SARS-COV-2 virus, causing COVID-19 has been investigated and the proteins of the virus have been identified in the literature. The virus consists of four structural: S (surface), M (membrane), E (envelope), N (nucleocapsid), and six non-structural (orf3a, orf3b, orf6, orf7a, orf7b, and orf8) genes. The structure of the genome of the virus is given in table 2. In this study, non-structural proteins were used and the interaction information between COVID-19, and human protein pairs were obtained from BioGRID dataset. The reason for using non-structural proteins in the study is that these proteins are thought to be necessary for the replication of viral genomes. Similarly, non-structural proteins important for viral RNA synthesis and for antagonizing host antiviral immunity. Therefore, predicting or determining the interaction network of non-structural proteins is key to understanding protein interactions. Table 1 shows the interacting COVID-19 and human proteins and total protein numbers.

**Table 1** Interacting COVID-19 and human proteins and the total number of these proteins

| COVID-19 proteins | Interacting human proteins | Total number of proteins |
|---|---|---|
| orf3a | ALG5, ARL6IP6, CLCC1, HMOX1, SUN2, TRIM59, VPS11, VPS39 | 8 |
| orf3b | STOML2 | 1 |
| orf6 | MTCH1, NUO98, RAE1 | 3 |
| orf7a | HEATR3, MDN1 | 2 |
| orf8 | ADAM9, ADAMTS1, CHPF2, CHPF, CISD3, COL6A1, DNMT1, EDEM3, EMC1, ERLEC1, ERO1LB, ERP44, FBXL12, FKBP7, FKBP10, FOXRED2, GDF15, GGH, HS6ST2, HYOU1, IL17RA, INHBE, ITGB1, KDELC1, KDELC2, LOX, MFGE8, NEU1, NGLY1, NPC2, NPTX1, OS9, PCSK6, PLAT, PLD3, PLEKHF2, PLOD2, POFUT1, PUSL1, PVR, SDF2, SIL1, SMOC1, STC2, TM2D3, TOR1A, UGGT2 | 47 |

| 5UTR | orf1ab | S | ORF3a | E | M | ORF6a | ORF7a | ORF7b | ORF8 | N | ORF10 | 3UTR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Non Coding Region | Polyprotein | Surface Glycoprotein | ORF3a Protein | Envelope Protein | Membrane Glycoprotein | ORF6a Protein | ORF7a Protein | ORF7b Protein | ORF8 Protein | Nucleocapsid Phosphprotein | ORF10 Protein | Non Coding Region |

Table 2(Genomic structure of SARS-COV-2 virus)

Since no interaction of the orf7b protein was observed, the orf7b protein was not considered in this study. The protein sequences were obtained from the NCBI dataset. A total of 3201 protein sequences were considered in the study. The same protein sequences were not used multiple times, but due to the small number of data, similar proteins were used in the study with the BLAST (Basic Local Alignment Search Tool) algorithm. The main reason for the lack of data is that COVID-19 is a new disease. This reveals the main limitation of our study. We tried to overcome this problem using the BLAST algorithm. Using the BLAST algorithm, we included protein sequences with a similarity rate of 90% and above. After we get the data, we want to overcome the mapping as all the mapping methods are chemical based such as EIIP (Electron–Ion Interaction Potential), CPNR (Complex Prime Number Representation) and hydrophobicity One of the methods is character based, the other one is signal based and the another one is physicochemical based.

## 1.2 problem Definition

As the high spread of covid-19 it has been a must to know how to develop a drug by counteract the SARS-COV-2 virus. One of the ways to do that is knowing how SARS-COV-2 proteins interact with human protein. Protein–protein interactions are important to examine protein functions and pathways involved in various biological processes and to determine the cause and progression of diseases. So, when we predict the interaction, we will be able to have knowledge of how the virus contacted the host during infection, and develop new drugs. When determining interactions between proteins experimentally are expensive and time-consuming, also experimental methods are sensitive to both the environment and operational processes, false-positive and false-negative results may occur. High-throughput experimental techniques have produced a large amount of protein–protein interaction (PPI) data, but their coverage is still low and the PPI data is also very noisy. Computational prediction of PPIs can be used to discover new PPIs and identify errors in the experimental PPI data. In computational methods, first genomic sequences are mapped and

then protein interactions are predicted by classifying them with machine-learning and deep-learning approaches. Compared to experimental methods, computational methods are time efficient and can analyze the protein interactions with less equipment. Furthermore, with the recent development of technology, protein sequence information can be obtained easily.

## 1.3 Project Objective

The main objective is to compare of our mapping method is that the mapping process is performed in an algorithm-based structure (AVL-Method) to the other methods are chemical-based. Then we able to perform on these mapped proteins with machine and deep learning classifier to predict if unknown protein has interaction with covid-19 proteins.

## 1.4 Gantt chart of project time plan

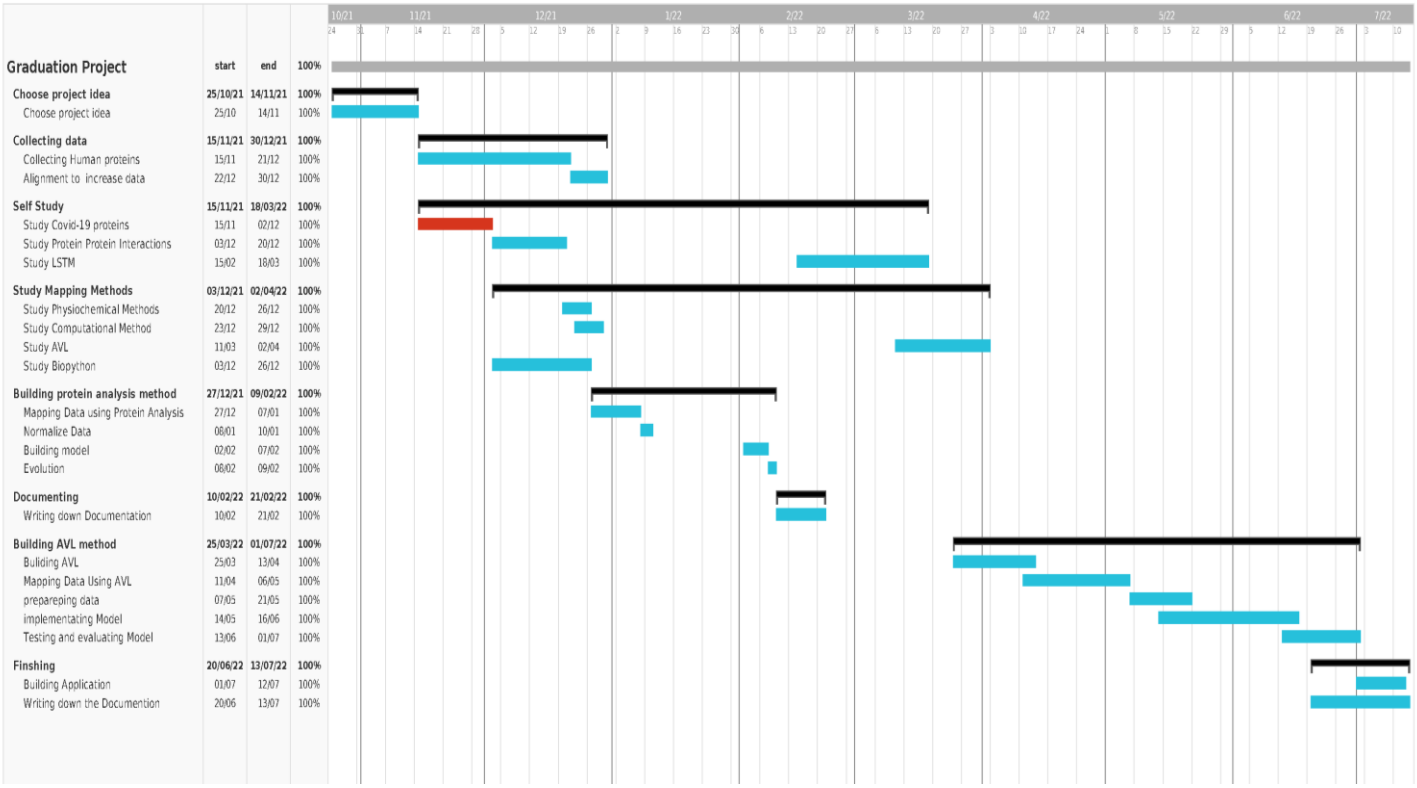



**Fig 1.1( Gantt Chart)**

## 1.5 Project development methodology

So, our roadmap to apply the computational method:

### First Method (BioPython)

1- After we get the human protein that interact with covid-19 as in -table 1.
2- but due to the small number of data, similar proteins were used in the project with the BLAST (Basic Local Alignment Search Tool) algorithm. We included protein sequences with a similarity rate of 90% and above.
3- After getting the data it is the time to mapping protein using the mentioned method, we will pass our data set to the Biopython function that will output for every input protein some physiochemical characteristics that will be our features.
4- Then, the extracted data will be normalized by standardization (Z-Score).
5- After that the data is ready to be classified. We will use 5K-fold using Machine Learning classifiers.
6- We will use three different machine learning classifiers (Logistic Regression - Support Vector Machine -Random Forest) and the accuracy of each classifier reported as shown in table 3.
7- Then evaluate the results of the classifiers.

| Classifier | Accuracy | Precession | Recall |
|---|---|---|---|
| **Logistic Regression** | **0.878** | **0.879** | **0.878** |
| **SVM** | **0.982** | **0.982** | **0.982** |
| **Random Forest** | **0.993** | **0.992** | **0.992** |

(Table 3. Evaluation table for the three classifiers).

As shown in table 3 random forest gave us the highest accuracy so we need to check if the model is biased by plotting confusion matrix as in fig1.2


**Fig1.2(Confusion Matrix of Random Forest)**

As shown in confusion matrix fig2.1 our model for protein Analysis with random forest made 99.33% accuracy without over-fitting or biased which is an accepted model and accuracy.

## Second Method (AVL)

AVL tree method, AVL tree is a kind of binary tree with a balance condition. A tree in which no node can have more than two children called a binary tree and it is useful in algorithm analysis applications. The balance of an AVL tree is determined the height of

.2

**Fig. 2** Insertion of amino acid codes to the AVL tree

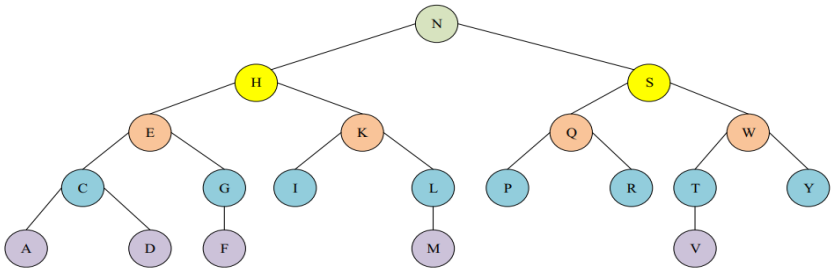


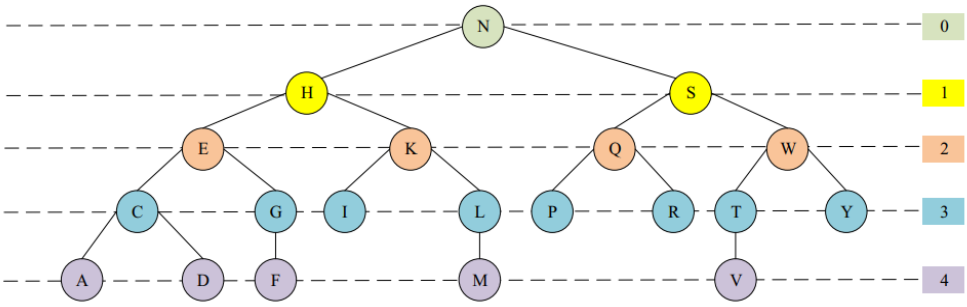

**Fig 2**

nodes. Basically, the left and right subtrees need to be at the same height. In detail, the differences between heights of right and left subtrees for each node must be less than or equal to 1. In nature, there are 20 amino acids. We add these amino acid codes to AVL tree in alphabetical order as in table 3. So, the final AVL will be as shown in Fig 2.

Table 4: Amino acids in alphabetical order.

| Amino acid | Abbreviation (3 Letters) | Abbreviation (1 Letters) |
|---|---|---|
| Alanine | Ala | A |
| Arginine | Arg | R |
| Asparagine | Asn | N |
| Aspartic Acid | Asp | D |
| Cysteine | Cys | C |
| Glutamine | Gln | Q |
| Glutamic Acid | Glu | E |
| Glycine | Gly | G |
| Histidine | His | H |
| Isoleucine | Ile | I |
| Leucine | Leu | L |
| Lysine | Lys | K |
| Methionine | Met | M |
| Phenylalanine | Phe | F |
| Proline | Pro | P |
| Serine | Ser | S |
| Threonine | Thr | T |
| Tryptophan | Trp | W |
| Tyrosine | Tyr | Y |
| Valine | Val | V |

**Fig. 3** Depth values of each amino acid




For instance: first we add Alanine (A) in the AVL tree since it is the first amino acid in alphabetical order and we insert Arginine (R), Asparagine (N), and so on. After the insertion process, we calculate the depth values of each node. Figure depicts the depth values of each node visually. We can see that only the root node has a 0-depth value. All other nodes have different depth values than root value. After calculation of the depth values as in Fig 3, we convert amino acid sequences to the numerical representations.

Let say we have a protein sequence S(n) = MHIILKFDASHN. The numerical representation of this sequence is calculated S(n) = 413332444110. We selected an AVL tree since the execution time is good and the time complexity is O(logN) in most of the cases. It is well

known fact that there are some other trees running in O(logN) time such as binary tree. In binary tree, there is no balance condition. AVL trees are often compared consistently with Red–Black trees. However, the biggest advantage of the AVL tree over Red–Black trees or even other trees is that it has a fast search process.

AVL Method Steps:

1- Mapping operation were done for each sequence one by one.
2- The mapped protein sequences were then padding to the maximum sequence Length.
3- The Structure of RNN will considered as following
    - In the first layer, a total of 64 BiRNN units were used. Then output values were calculated with ReLU activation function.
    - Dropout was performed and 25% of the data was forgotten.
    - BiRNN was used again in the second layer. The number of units in this layer is determined as 32. ReLU was again used as the activation function.
    - Dropout was performed and 25% of the data was forgotten.
    - BiRNN was used again in the third layer. The number of units in this layer is determined as 16. ReLU was again used as the activation function.
    - Dropout was performed and 25% of the data was forgotten.
    - Then the Flatten function was applied to flatten the data in matrix format.
    - Batch normalization was performed to prevent changes in data distribution.
    - After flatten and batch normalization, dropout was performed and 25% of the data was forgotten.
    - Later, fully connected layers were designed and 512 neurons were used in the first fully connected layer.
    - Classification was carried out in the second fully connected layer, and the interactions were predicted with the sigmoid function.

- SGD (Stochastic Gradient Descent) was applied as the optimizer and the learning rate and momentum values were designed to be 0.0001 and 0.9, respectively.
- For model loss, binary crossentropy was used and the model was complied with a value of 500 epochs, 64 batch_size.

4- After Padding process, interactions were classified with the RNN deep-learning model. Prediction studies based on protein interactions are generally based on two scenarios: interaction either exists or does not exist. In other words, binary classification is made. Since it is known which proteins belonging to the virus interact with which proteins belonging to humans, the output of the interacting proteins is determined as 1 and the others as 0.

5- After performing binary classification with RNN we will calculate the accuracy of the results.

We get 99.58% as we can see in fig4.

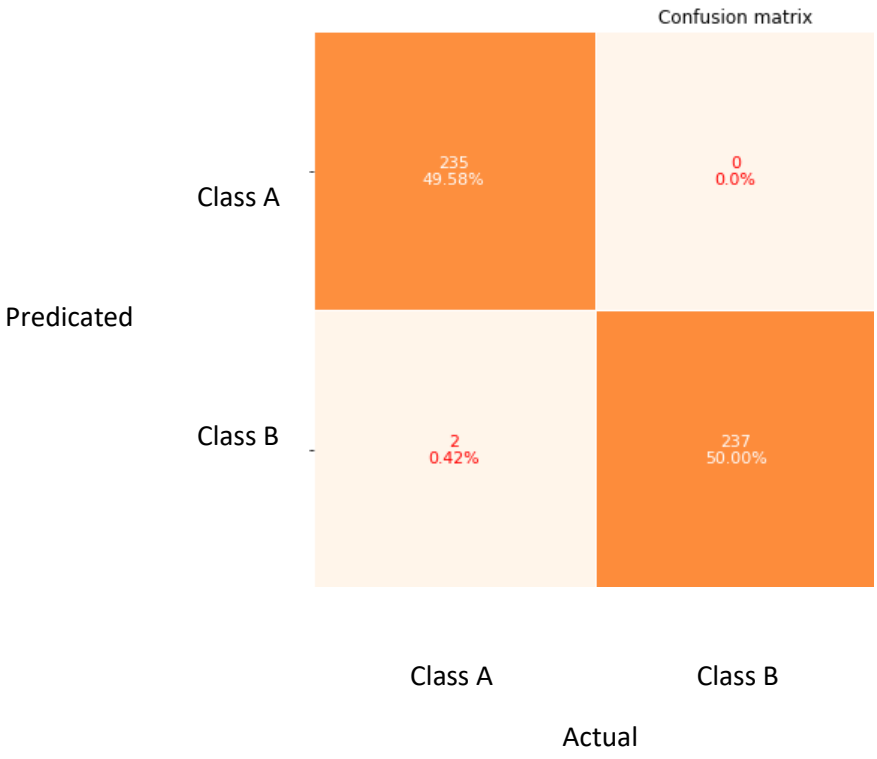



**Fig 4(Confusion Matrix For RNN)**

## 1.6 The used tools in the project (SW and HW)

### 1-Hardware:

This project does not need any hardware except ones' personal computer –on web site- or smart phone-as mobile application.

### 2-Software:

#### 1- Models:

- Languages:

| | | |
|---|---|---|
| **Python** | Python is a programming language that lets you work quickly and integrate systems more effectively. | |

- Libraries:

| | | |
|---|---|---|
| **SKLearn** | Which is probably the most useful library for machine learning in Python with efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction-which needed for **((CNN), (SVM) and (RNN))** models | |

| | | |
|---|---|---|
| **TensorFlow** | Is an end-to-end platform that makes it offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API which easy for us to build and deploy ML models. | |
| **BioPython** | The Biopython project is an open-source collection of non-commercial Python tools for computational biology and bioinformatics, created by an international association of developers. | |
| **Keras** | Is an API designed for human beings which follows best practices for reducing cognitive load and offers consistent & simple APIs, | |

- IDEs:

| | | |
|---|---|---|
| **Google Colab** | Colaboratory, or "Colab" for short, is a product from Google Research. Colab **allows anybody to write and execute arbitrary python code through the browser**, and is especially well suited to machine learning, data analysis and education. | |

| | | |
|---|---|---|
| **PyCharm** | PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated **to create a convenient environment for productive Python, web, and data science development**. | |
| **HTML** | Is an open-source UI software development kit, It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase. | |
| **CSS** | CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once | |
| **JAVASCRIPT** | JavaScript is a scripting or programming language that allows you to implement complex features on web pages | |
| **Flask** | Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. | |

**Table 5: Application Tools & libraries**

## 1.7 Report Organization (summary of the rest of the report)

**Chapter Two: Related Work:**
In chapter two, we will establish other work associated with our research. There, we show that different ways to achieve our goals. We will declare the authors of these methods, the classification method they use in their project, and off course the accuracy.

**Chapter Three: System Analysis:**
In chapter three, we will talk about project specification. It will contain descriptions of how the program will be used from a user perspective and performance details such as usability, reliability, and stability. In addition, illustrate a use case diagram that emphases our program.

**Chapter Four: System Design:**
The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces using some diagrams. System Component Diagram, System Class Diagrams, Sequence Diagrams, Project ERD, System GUI Design

**Chapter Five: Implementation and Testing:**
This phase includes screen shots of training and testing of (SVM, RF, LR) model, RNN model
Screenshots of the system running and samples of the applied test cases.

# Chapter 2: Related work

**Related work**

In study [1] radial-based functional neural networks to determine the protein–protein interaction sites. Protein sequences were converted to the numerical representations at the first stage, and frequency values of each amino acid were calculated. Then, by calculating the relative entropy, a total of 1000 features were obtained. The main differences between that study and our study are they used a physiochemical-based mapping method but we used algorithm-based mapping method, also the protein interactions were specified using SNN (Siamese Neural Network) but we will use RNN (Recurrent Neural Network). In Study [2] authors used SVM classifier to predict PPIs.

| | Study [1] | Study [2] | Our project (phase 1) |
|---|---|---|---|
| **Mapping Method** | Computational based (PSSM) | Physiochemical (Hydrophobicity amino acid classification) | Physiochemical Computational (Protein Analysis |
| **Classifier** | CNN (convolutional Neural Network) | Deep Forest | Logistic Regression<br>SVM<br>Random Forest |
| **Accuracy** | 97.75% | 71.4% | 87.8%<br>98.2%<br>99.3% |

**Table 6.1 (Related Works)**

-In study [3] they used AVL tree as a mapping method and RNN classifier we also used the same mapping method and classifier but we changed in preparing the data before passing it to the model (RNN). They normalized the data using z-score but we performed padding to the sequence before passing it to the model. They get 97.76% accuracy but we get 99.58%.

| | Study [3] | Our project (phase 2) |
|---|---|---|
| **Mapping Method** | AVL (algorithm-based) With Normalization | AVL (algorithm-based) With padding |
| **Classifier** | RNN (recurrent neural network) | RNN (recurrent neural network) |
| **Accuracy** | 97.76% | 99.58% |

**Table 6.2 (Related Works)**

# Chapter 3: System Analysis

### 3.1 Project specification:

### 3.1.1 Functional requirement

1) **Registration**: If this is the first time for the user to use the website, he/she will register.

2)**Log in**: If the user already registers before, the user will log in to the app with his user's name and password.

3)**PPI**: The user will input the sequence that he/she wants to predict the interaction of this protein with COVID-19 proteins. The PPI will be processed and transferred to phase that contains prediction process. Is protein having interaction or not? with four different models RNN, SVM, RF and LR.

4**) profile**: the user can show his history, and he could back to the history and check the results and accuracy.

### 3.1.2 Non-functional requirement

- **Performance:**

For example, Response Time, Throughput, Utilization, Static Volumetric.

- **Usability:**

Prioritize the important functions of the system based on usage patterns. Frequently used functions should be tested for usability, as should complex and critical functions.

- **Reliability:**

Users have to trust the system, even after using it for a long time. Create a requirement that data created in the system will be retained for several years without the data being changed by the system. Make it easier to monitor system performance.

- **Security:**

Security is the degree of resistance to, or protection from, harm. It's important to have this feature as a requirement.

- **Availability:**

Availability refers to a property of software that it is there and ready to carry out its task when you need it to be. This is a broad perspective and encompasses what is normally called reliability (although it may encompass additional considerations such as downtime due to periodic maintenance). In fact, availability builds upon the concept of reliability by adding the notion of recovery—that is, when the system breaks, it repairs itself.

## 3.2 Use Case Diagrams

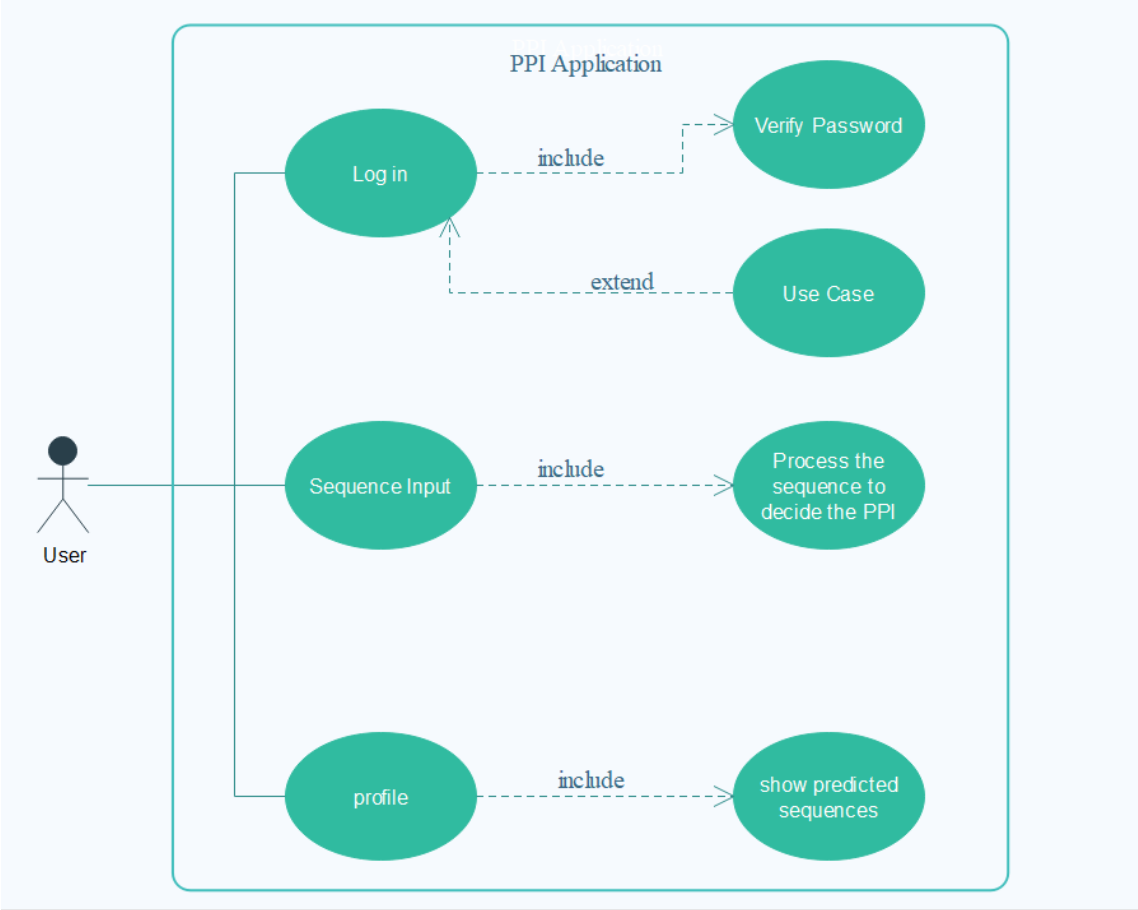



**Figure 5.1(Use Case Diagram)**

# Chapter 4: System Design

# System Design

- **System Component Diagram**

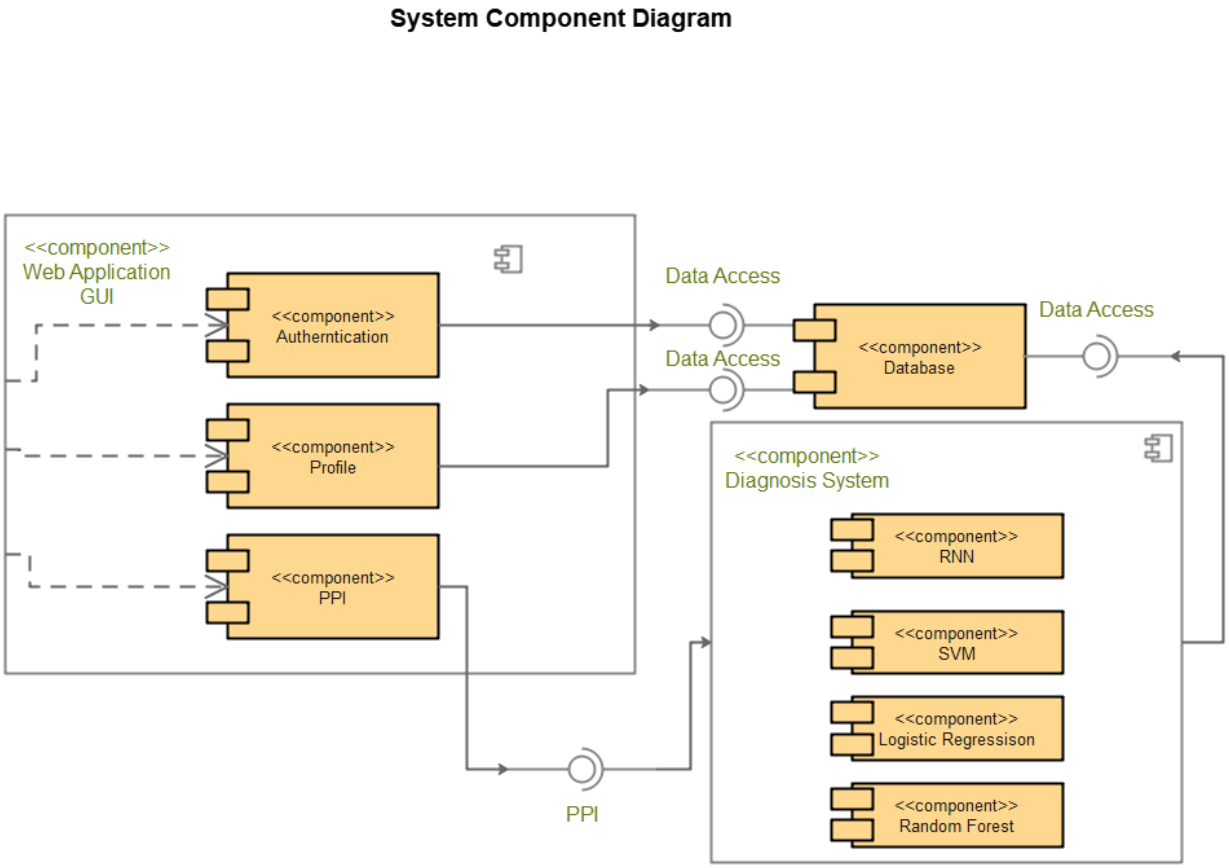



**Figure 5.2(System Component Diagram)**

- **System Class Diagrams**

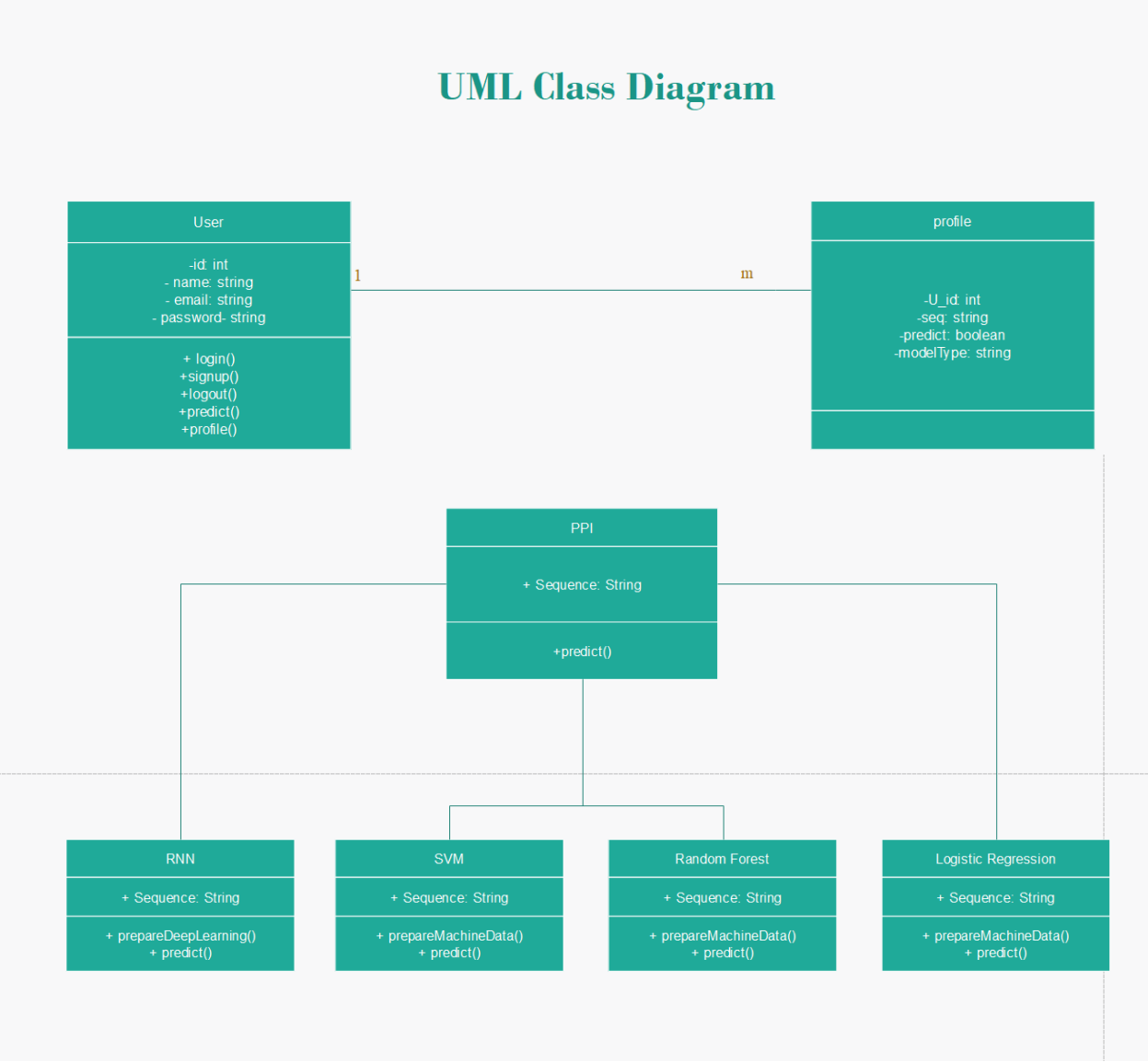



**Figure 5.3(System Class Diagram)**

- **Sequence Diagram**

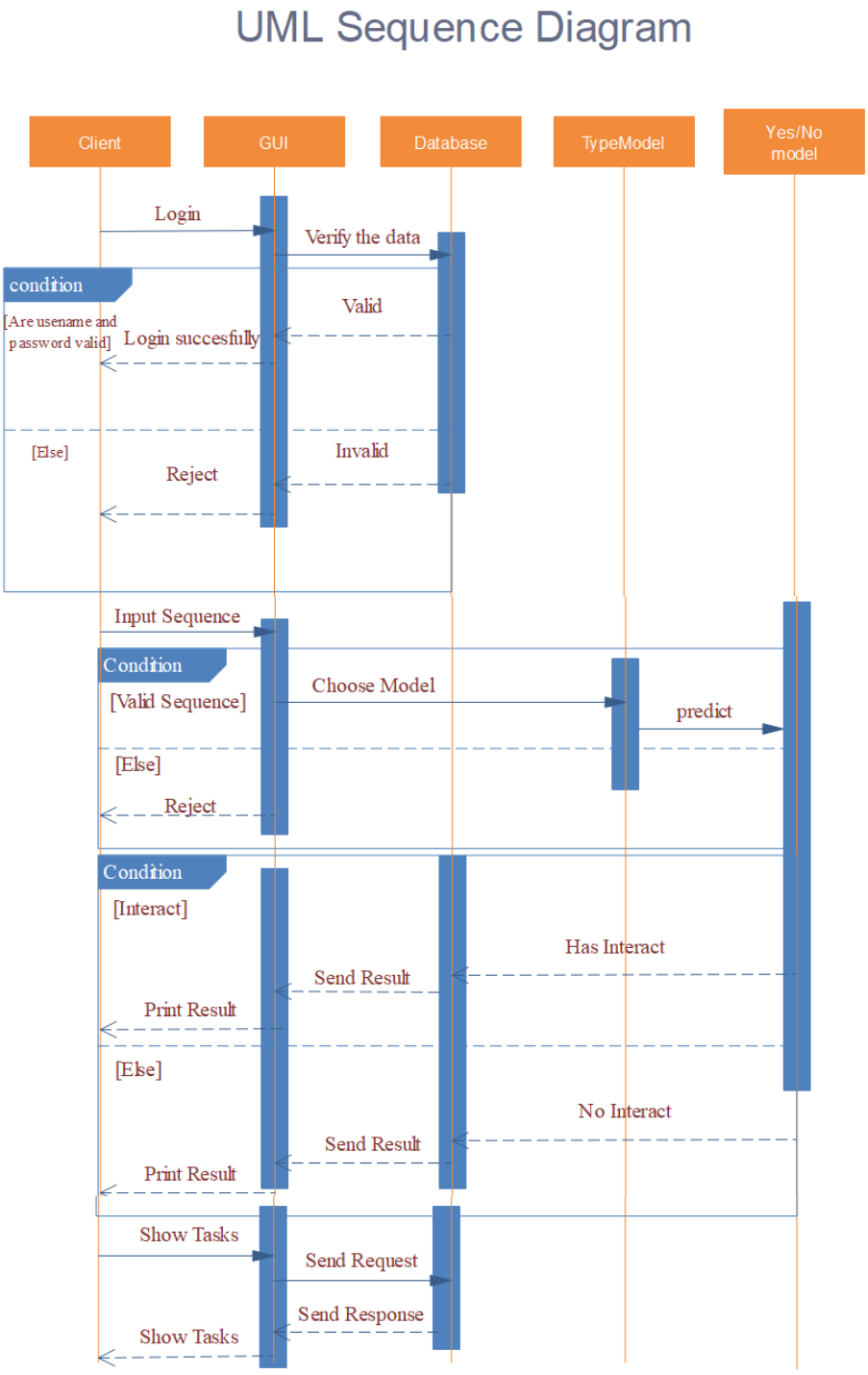



**Figure 5.4 (System Sequence Diagram)**

- **ERD**

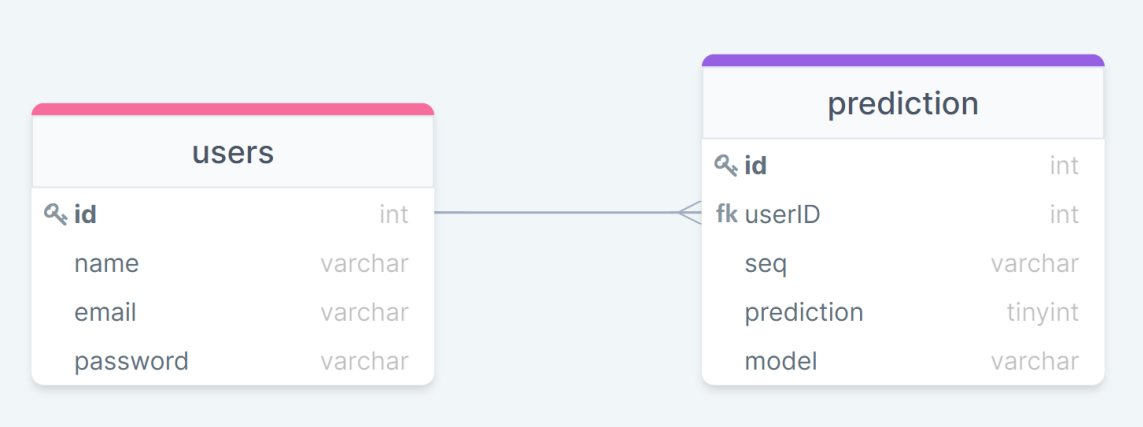



**Figure 5.5(System ERD Diagram)**

- **System GUI Design**

**Login**

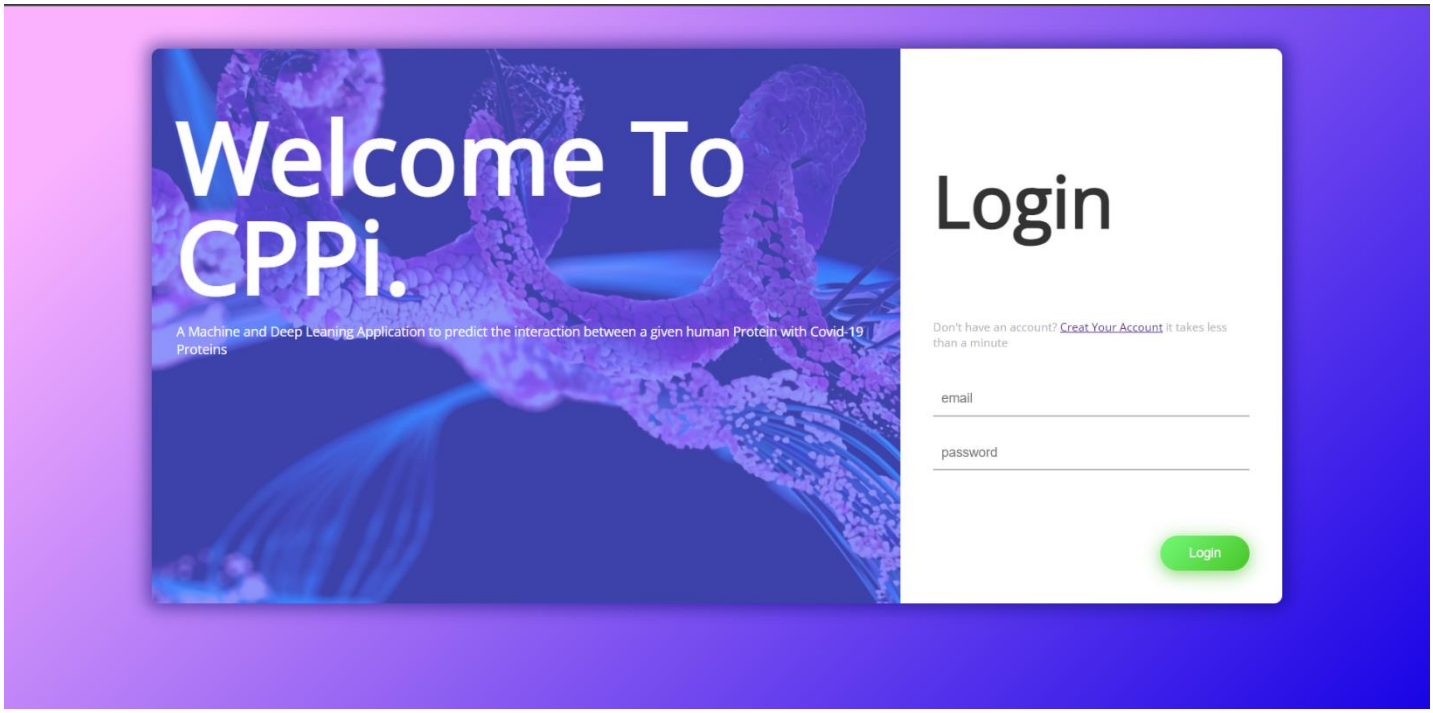



**Figure 5.6(System Login GUI)**

## Registration

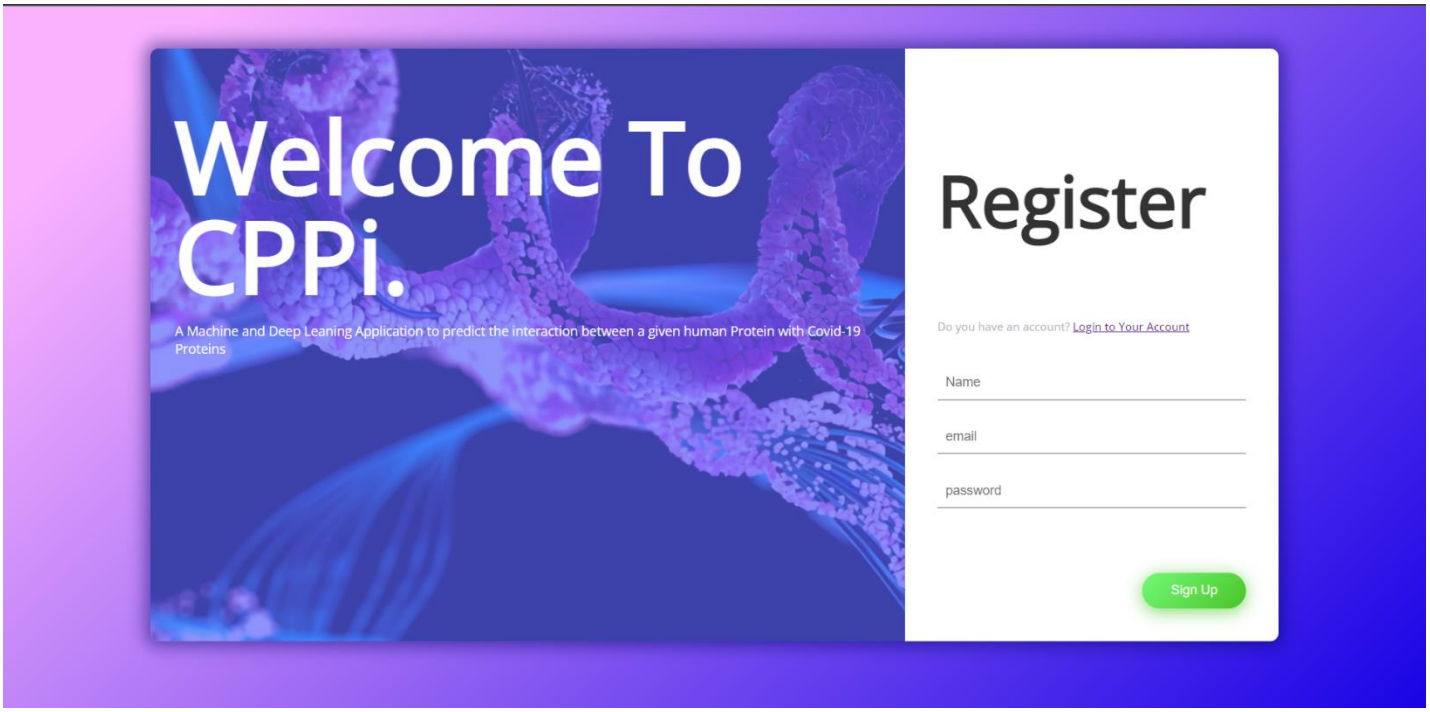



**Figure 5.7(System Register GUI)**

# Profile

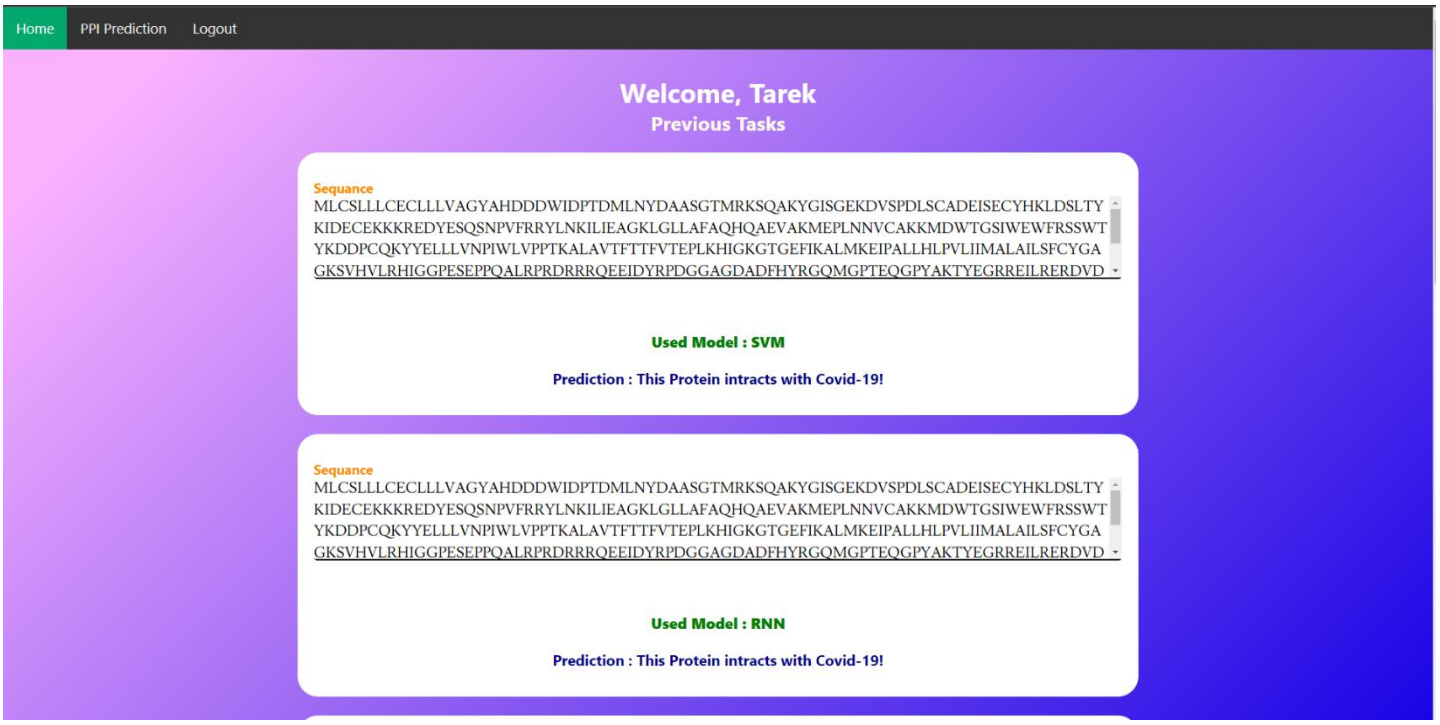



**Figure 5.8(System Profile GUI)**

## Predication process

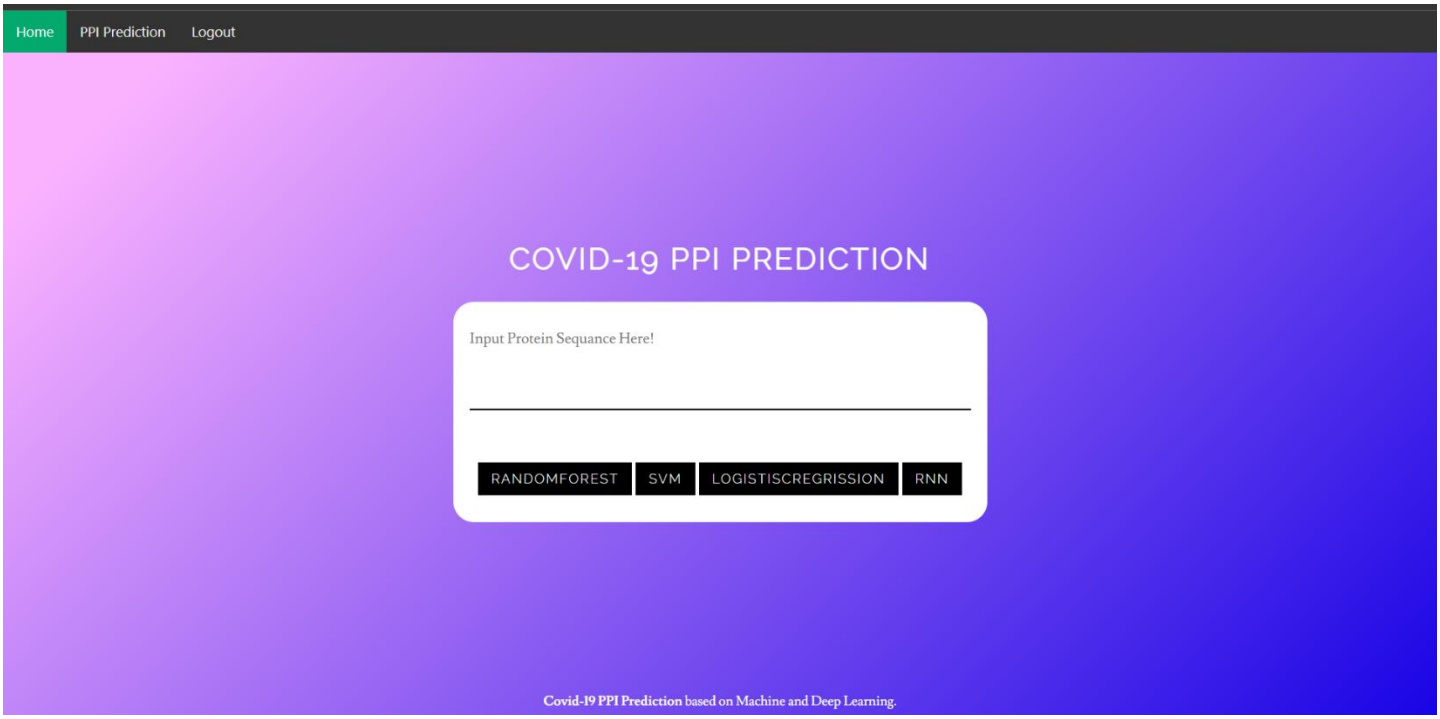


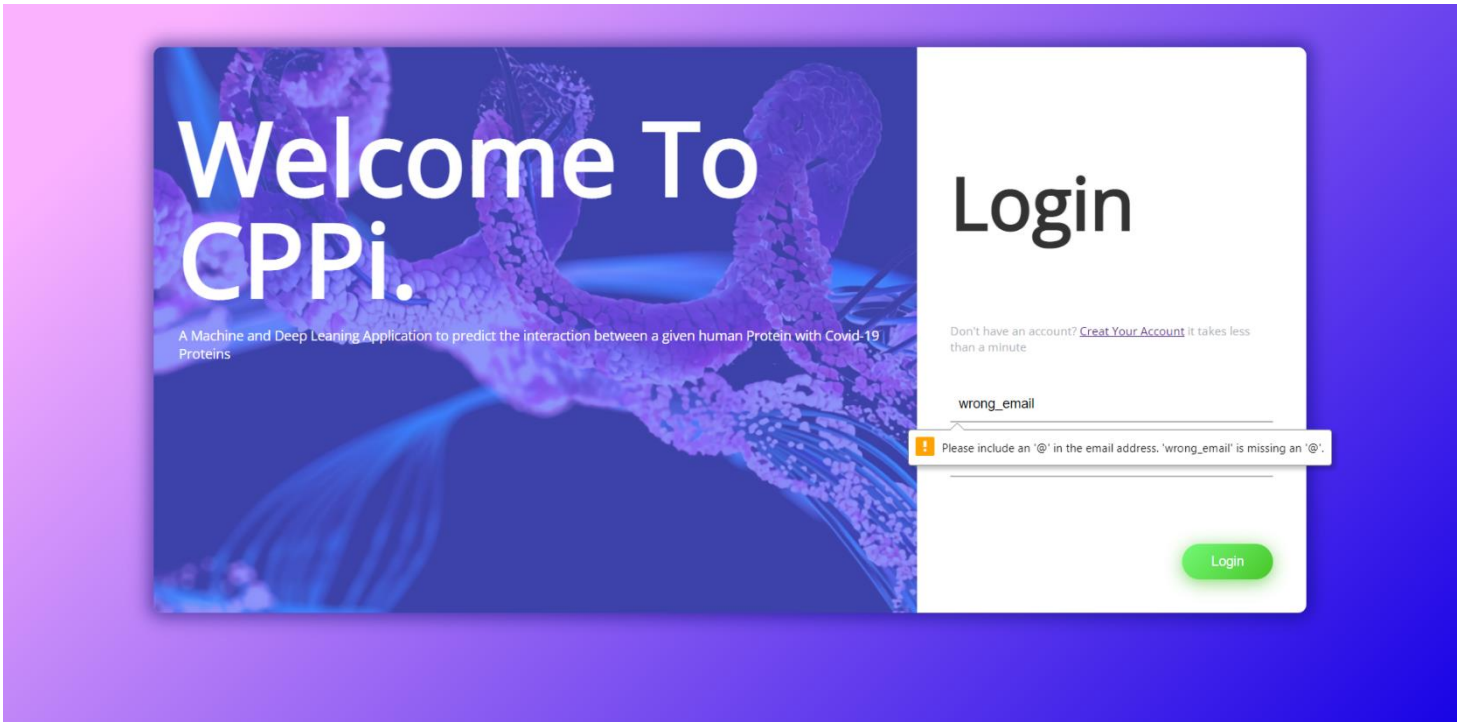

**Figure 5.9(System PPI GUI)**

# Chapter 5: Implementation and testing

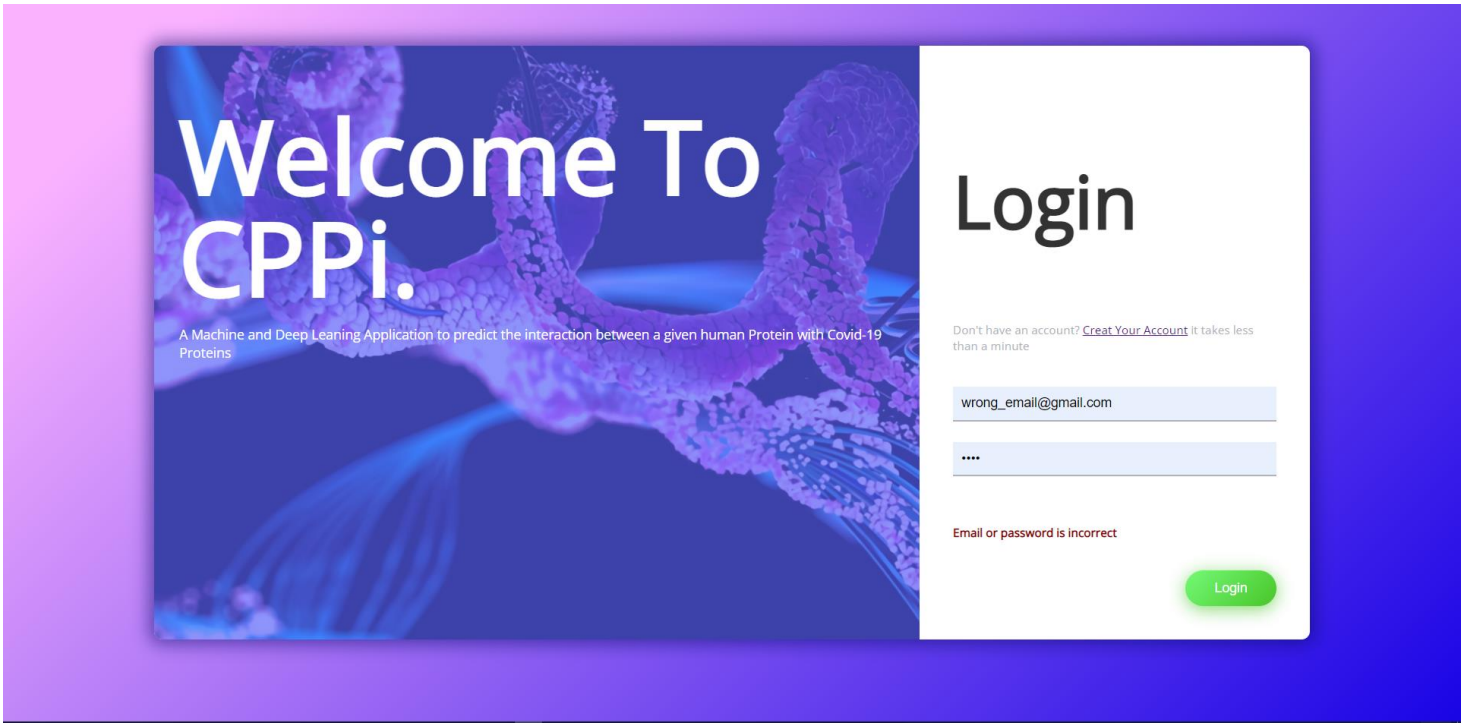
**Implementation and testing:**

- The rest of this part will be snapshots of the system scenarios including all test cases that could be happing cause by user:

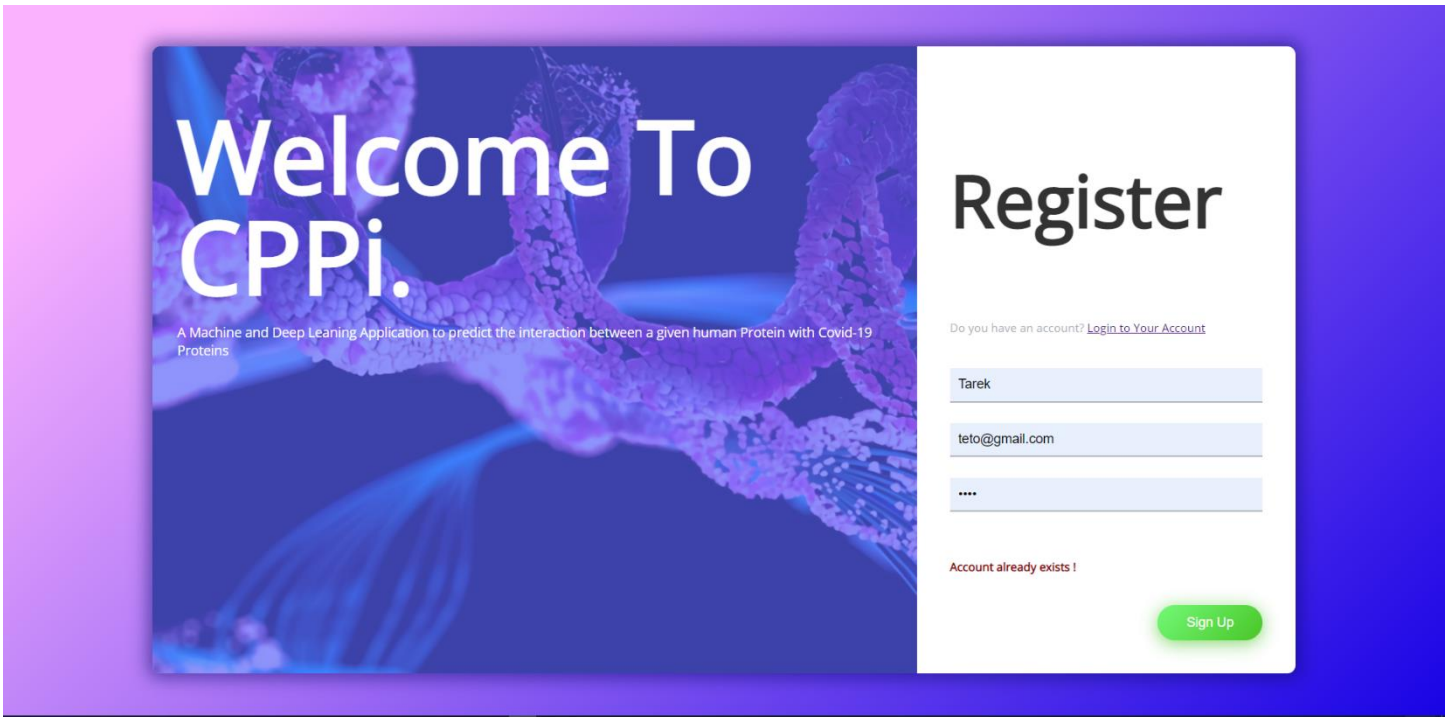
**1- Invalid email and password**
**1.1 invalid email pattern**



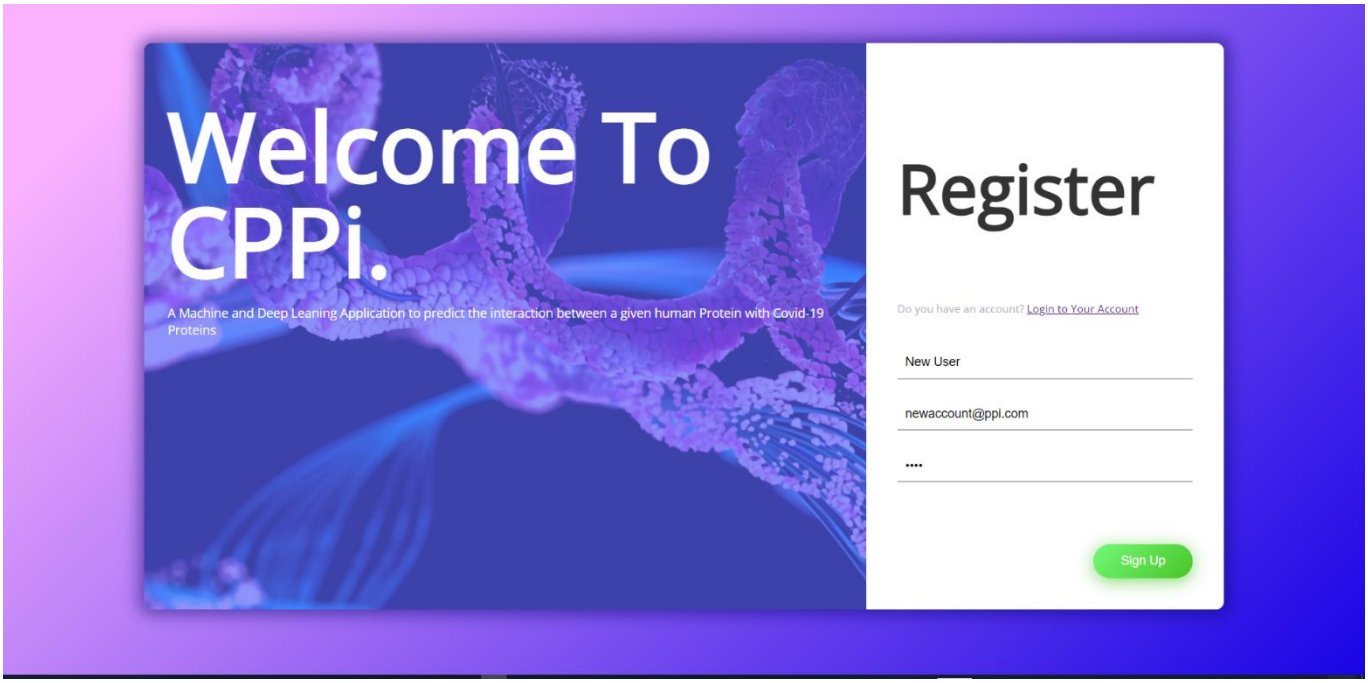
## 1.2 invalid email or password

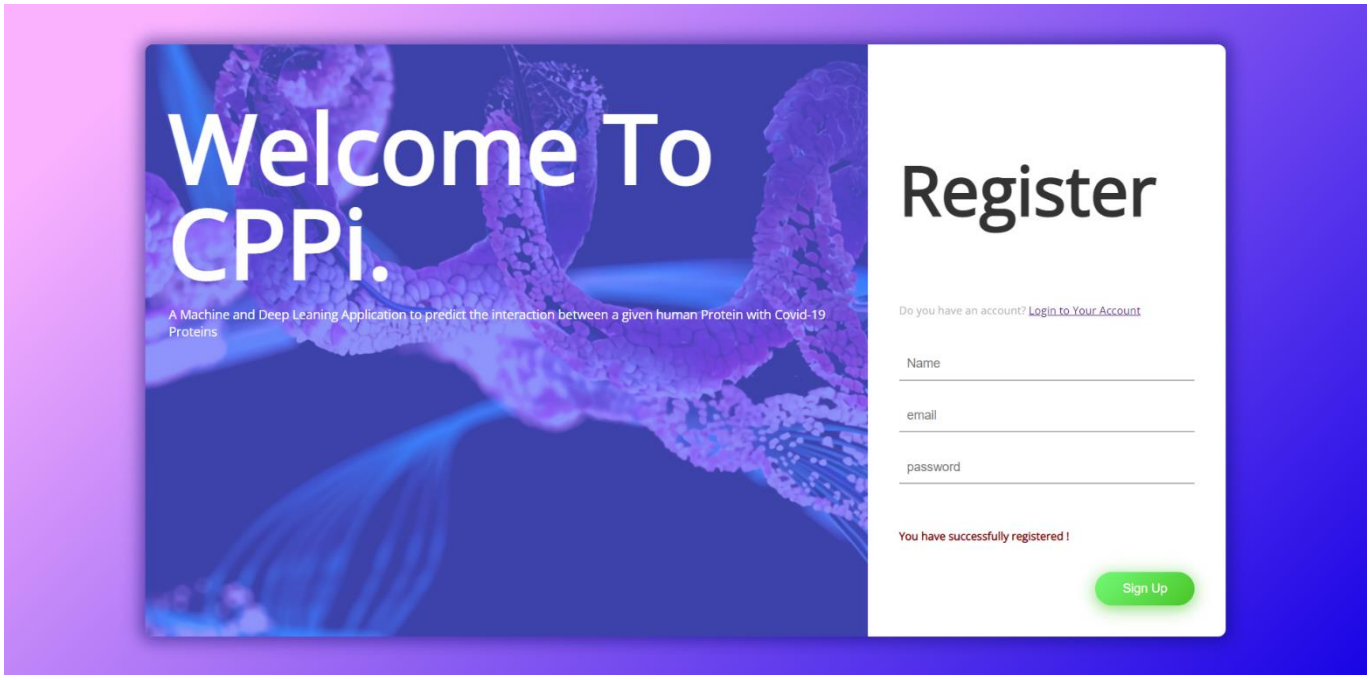



## 2- Registration with exist email

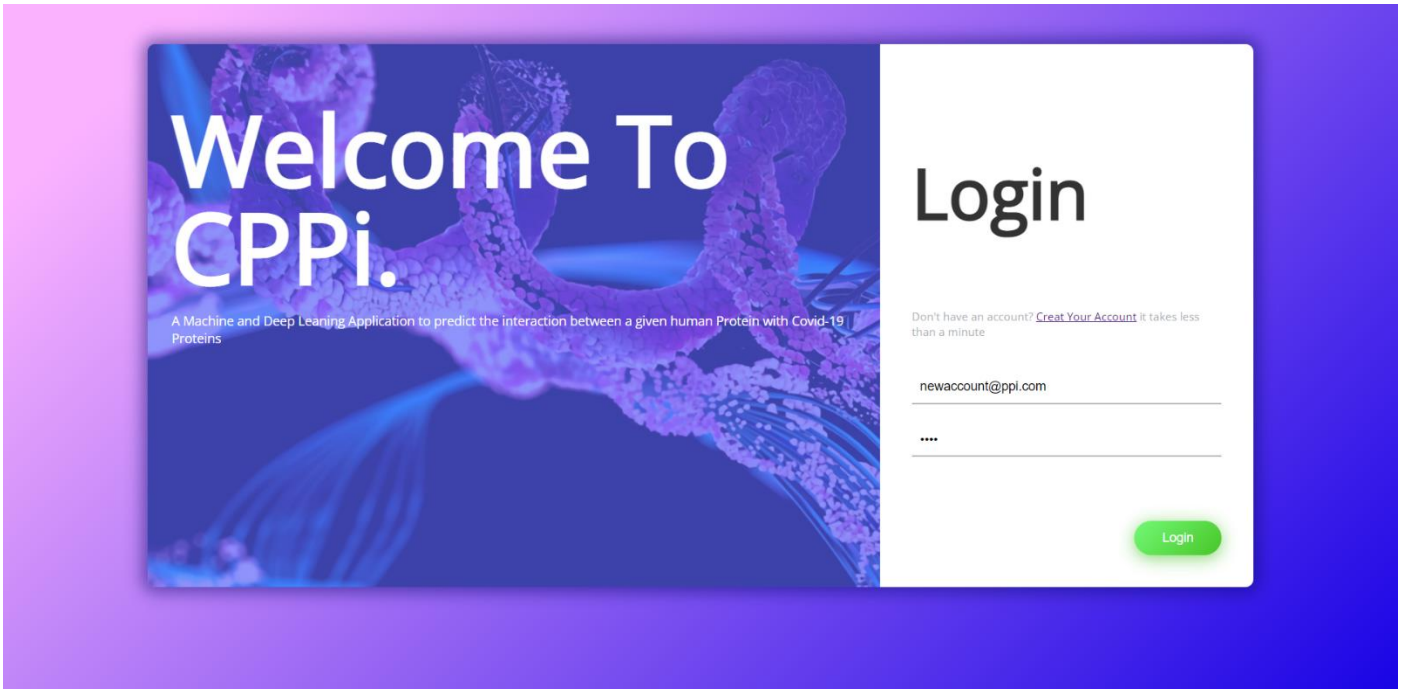


## 3- Registration Normal

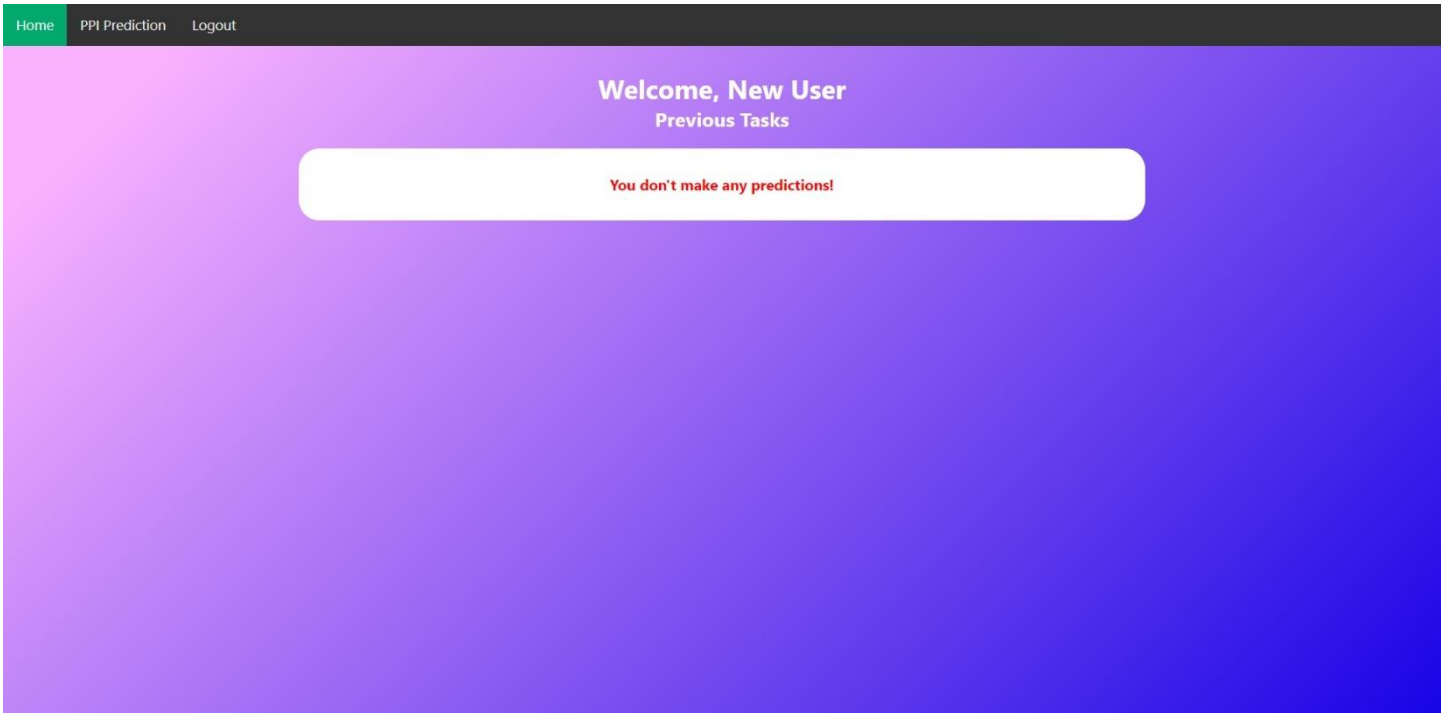


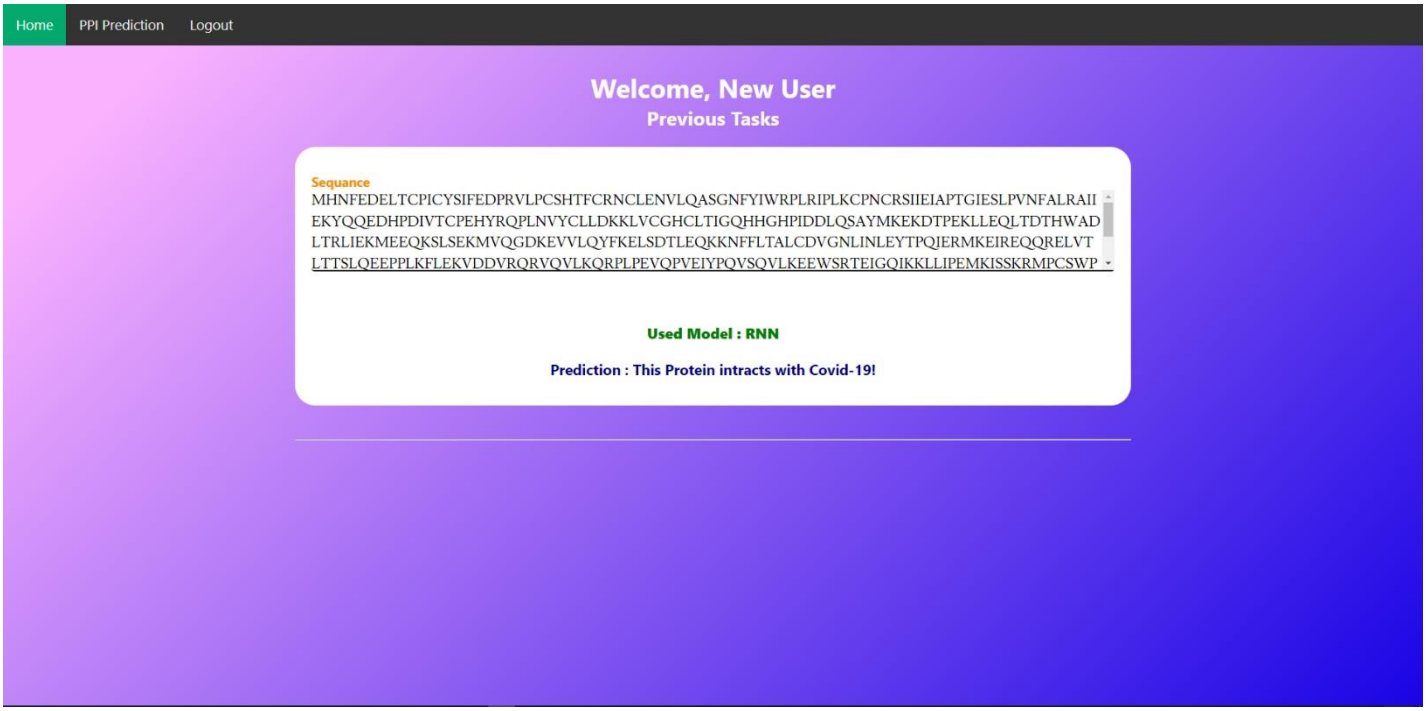
## 4- Login Normal



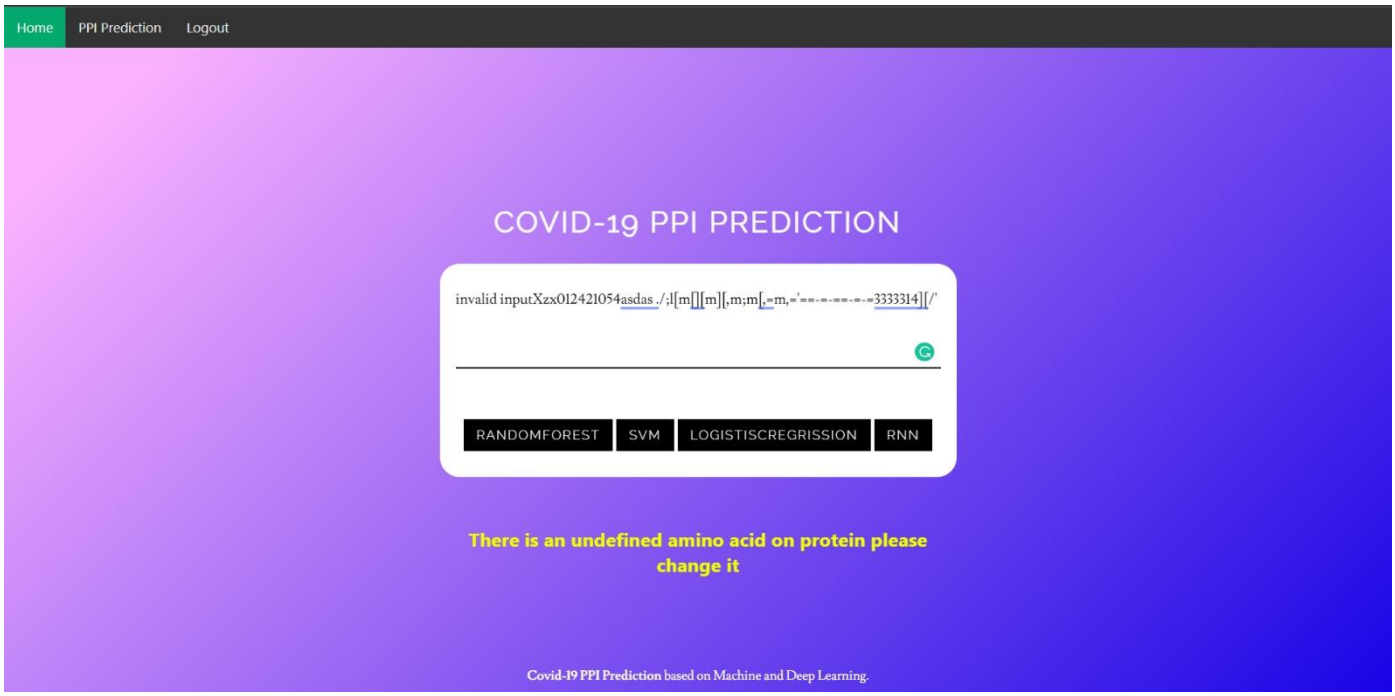

## 5- Profile Has no Tasks



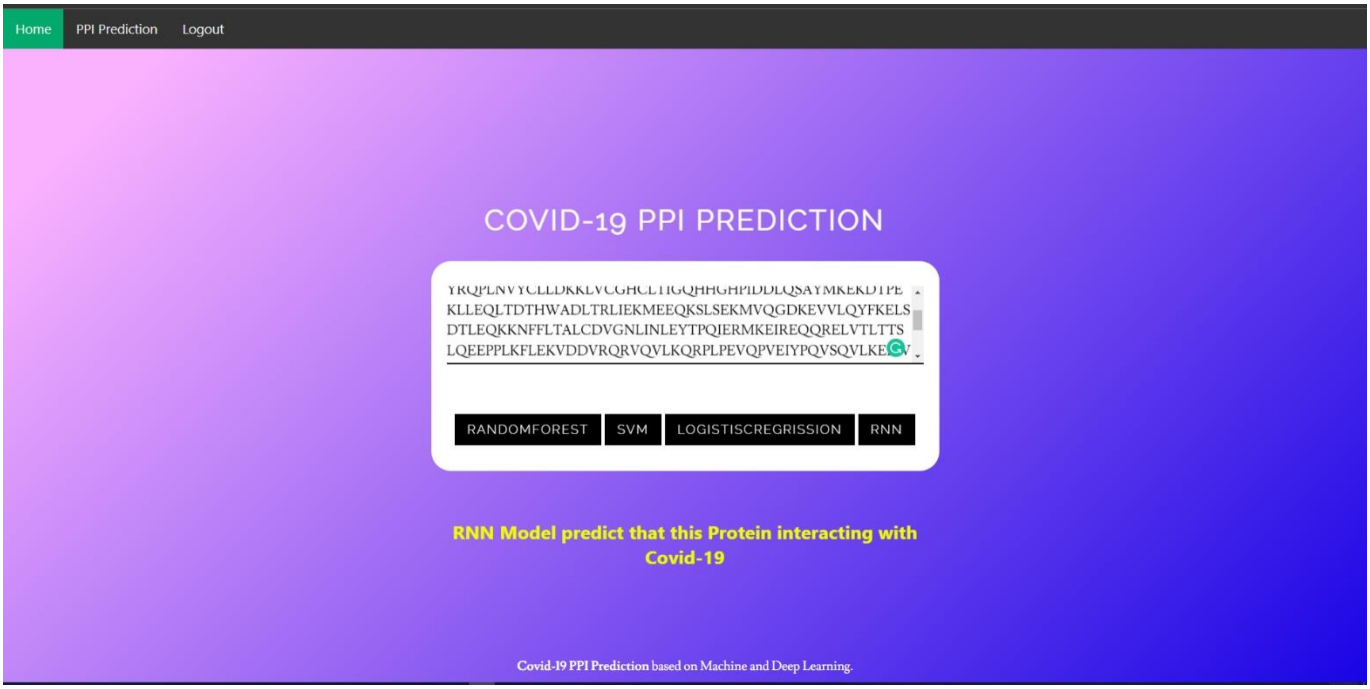
## 6- Previous Tasks




## 7- Invalid Input



## 8- Valid Sequence (Normal Case)



# Appendix

1- **Data:**

- The data consist of 5 different covid proteins:
  . orf3a
  . orf3b
  . orf6
  . orf7a
  . orf8
- Split the data (80% training and validation, 20% testing)
- Each protein contains several human proteins which is interact or not with it.
- So, each protein will contain two group:
  . positive sequences (which interact)
  . negative sequences (which not interact)

```
def mapping(proteinName, fileName, outputFile):
    df = pd.read_excel(r'COVID-19 PPI Data\\{}\\{}.xlsx'.format(proteinName, fileName))

    grouped = df.groupby(df['Interaction Output'])
    positive = grouped.get_group(1)
    negative = grouped.get_group(0)

    sequencePositive = list(SeqIO.parse('COVID-19 PPI Data\\%s\\positive.fasta' % proteinName, 'fasta'))
    sequenceNegative = list(SeqIO.parse('COVID-19 PPI Data\\%s\\negative.fasta' % proteinName, 'fasta'))

    positivesSequences = []
    negativesSequences = []
    counter = 0
    for pos in positive['Protein ID']:
        counter = 0
        for seq in sequencePositive:
            if seq.id == pos:
                positivesSequences.append(str(seq.seq))
                sequencePositive[counter].id = 0
                break
            counter += 1

    counter = 0
    for neg in negative['Protein ID']:
        counter = 0
        for seq in sequenceNegative:
            if seq.id == neg:
                negativesSequences.append(str(seq.seq))
                sequenceNegative[counter].id = 0
                break
            counter += 1

    positive['Sequence'] = positivesSequences
    negative['Sequence'] = negativesSequences

    frames = [positive, negative]
    df = pd.concat(frames)
    # create excel writer object
    writer = pd.ExcelWriter('COVID-19 PPI Data\\{}\\{}.xlsx'.format(proteinName, outputFile))
    # write dataframe to excel
    df.to_excel(writer, index=False)
    # save the Excel
    writer.save()
```

- Data before apply mapping method:

| | Protein ID | Interaction Output |
|---|---|---|
| 2 | NP_002124.1 | 1 |
| 3 | AAX36773.1 | 1 |
| 4 | AAV38587.1 | 1 |
| 5 | NP_002124.1 | 1 |
| 6 | 4WD4_A | 1 |
| 7 | ADZ76424.1 | 1 |
| 8 | XP_030861390.1 | 1 |
| 9 | AAH01491.1 | 1 |
| 10 | BAG35941.1 | 1 |
| 11 | XP_023077866.1 | 1 |
| 12 | PNJ49363.1 | 1 |
| 13 | XP_032003242.1 | 1 |
| 14 | NP_001126358.1 | 1 |
| 15 | XP_030671530.1 | 1 |
| 16 | XP_007973791.1 | 1 |
| 17 | XP_025255454.1 | 1 |
| 18 | EHH65790.1 | 1 |
| 19 | XP_011893429.1 | 1 |
| 20 | XP_033088657.1 | 1 |
| 21 | XP_021777178.1 | 1 |
| 22 | EHH20181.1 | 1 |
| 23 | XP_011834363.1 | 1 |
| 24 | XP_017359512.1 | 1 |
| 25 | XP_011834360.1 | 1 |
| 26 | XP_011813509.1 | 1 |
| 27 | XP_002743767.1 | 1 |
| 28 | XP_009232576.1 | 1 |
| 29 | XP_009232574.1 | 1 |
| 30 | XP_032125620.1 | 1 |
| 31 | XP_003932972.1 | 1 |

**Table 7.1( Data befor Mapping)**

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **Protein ID** | **raction Ou** | **Sequence** | | |
| 2 | NP_00212 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 3 | AAX36773 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 4 | AAV38587 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 5 | NP_00212 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 6 | 4WD4_A | 1 | GSHMMERPQPDSMPQDLSEALKEAT | | |
| 7 | ADZ76424 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 8 | XP_03086 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 9 | AAH01491 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 10 | BAG35941 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 11 | XP_02307 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 12 | PNJ49363 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 13 | XP_03200 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 14 | NP_00112 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 15 | XP_03067 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 16 | XP_00797 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 17 | XP_02525 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 18 | EHH65790 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 19 | XP_01189 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 20 | XP_03308 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 21 | XP_02177 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 22 | EHH20181 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 23 | XP_01183 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 24 | XP_01735 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |
| 25 | XP_01183 | 1 | MERLQPDSMPQDLSEALKEATKEVHT | | |
| 26 | XP_01181 | 1 | MKRLQPDSMPQDLSEALKEATKEVHT | | |
| 27 | XP_00274 | 1 | MERPQPDSMPQDLSEALKEATKEVHT | | |

**Table 7.2 (Data after extract sequences)**

- Chemical based mapping method**:**
  **.** extract the features of each protein using the method ProteinAnalysis

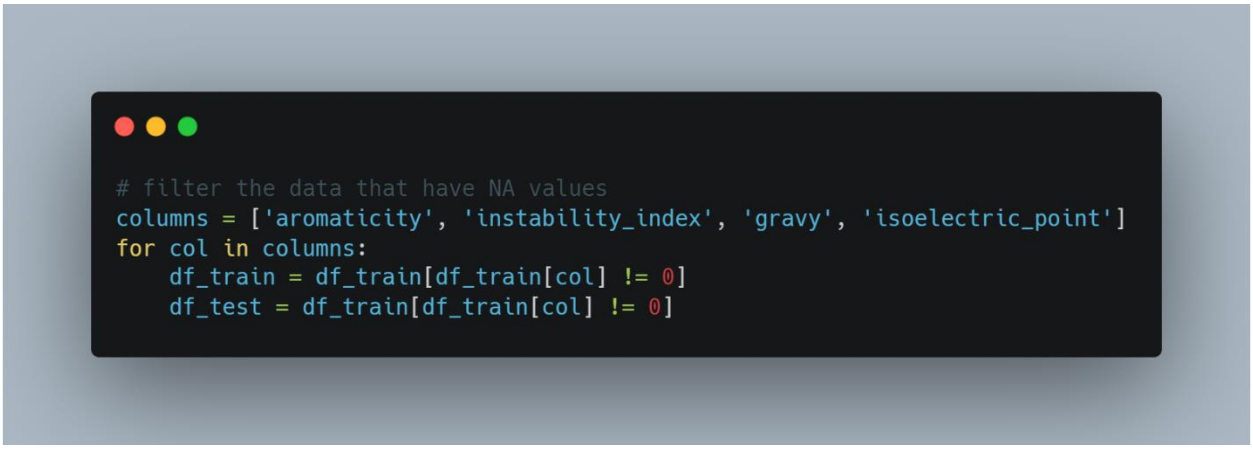
| Protein ID | raction Ou | Sequence | aromaticity | tability_inc | gravy | electric_point |
|---|---|---|---|---|---|---|
| NP_00212 | 1 | MERPQPD | 0.090278 | 60.81319 | -0.42674 | 7.885467 |
| AAX36773 | 1 | MERPQPD | 0.089965 | 60.63737 | -0.43737 | 7.044932 |
| AAV38587 | 1 | MERPQPD | 0.089965 | 60.63737 | -0.41211 | 7.885467 |
| NP_00212 | 1 | MERPQPD | 0.090278 | 60.81319 | -0.42674 | 7.885467 |
| 4WD4_A | 1 | GSHMMER | 0.089041 | 60.27671 | -0.4339 | 8.634589 |
| ADZ76424 | 1 | MERPQPD | 0.086806 | 60.71493 | -0.43333 | 7.907708 |
| XP_03086 | 1 | MERPQPD | 0.090278 | 59.2875 | -0.42222 | 7.083809 |
| AAH01491 | 1 | MERPQPD | 0.090278 | 62.60069 | -0.42222 | 7.083809 |
| BAG35941 | 1 | MERPQPD | 0.090278 | 61.48194 | -0.44549 | 7.885467 |
| XP_02307 | 1 | MERPQPD | 0.086806 | 60.64653 | -0.45382 | 7.085174 |
| PNJ49363 | 1 | MERPQPD | 0.090278 | 59.17535 | -0.41667 | 7.899069 |
| XP_03200 | 1 | MERPQPD | 0.090278 | 59.57465 | -0.39097 | 8.805817 |
| NP_00112 | 1 | MERPQPD | 0.090278 | 59.30729 | -0.41215 | 7.120584 |
| XP_03067 | 1 | MERPQPD | 0.090278 | 60.2434 | -0.4059 | 7.809458 |
| XP_00797 | 1 | MERLQPD | 0.09375 | 60.92396 | -0.42188 | 7.043681 |
| XP_02525 | 1 | MERLQPD | 0.090278 | 62.12778 | -0.43646 | 7.085174 |
| EHH65790 | 1 | MERLQPD | 0.090278 | 62.12778 | -0.44549 | 7.085174 |
| XP_01189 | 1 | MERLQPD | 0.090278 | 59.93333 | -0.42743 | 7.085174 |
| XP_03308 | 1 | MERPQPD | 0.090278 | 59.97778 | -0.41042 | 7.085174 |
| XP_02177 | 1 | MERLQPD | 0.090278 | 60.97604 | -0.45451 | 7.085174 |
| EHH20181 | 1 | MERLQPD | 0.090278 | 60.60208 | -0.44965 | 6.73101 |
| XP_01183 | 1 | MERLQPD | 0.090278 | 60.86389 | -0.41042 | 7.907386 |
| XP_01735 | 1 | MERPQPD | 0.086806 | 53.82292 | -0.44271 | 6.414303 |
| XP_01183 | 1 | MERLQPD | 0.092784 | 62.51134 | -0.40481 | 7.907386 |

**Table 7.3(Data after Mapping)**

- To extract this feature, we use getPhysicochemicalCharacteristics method.

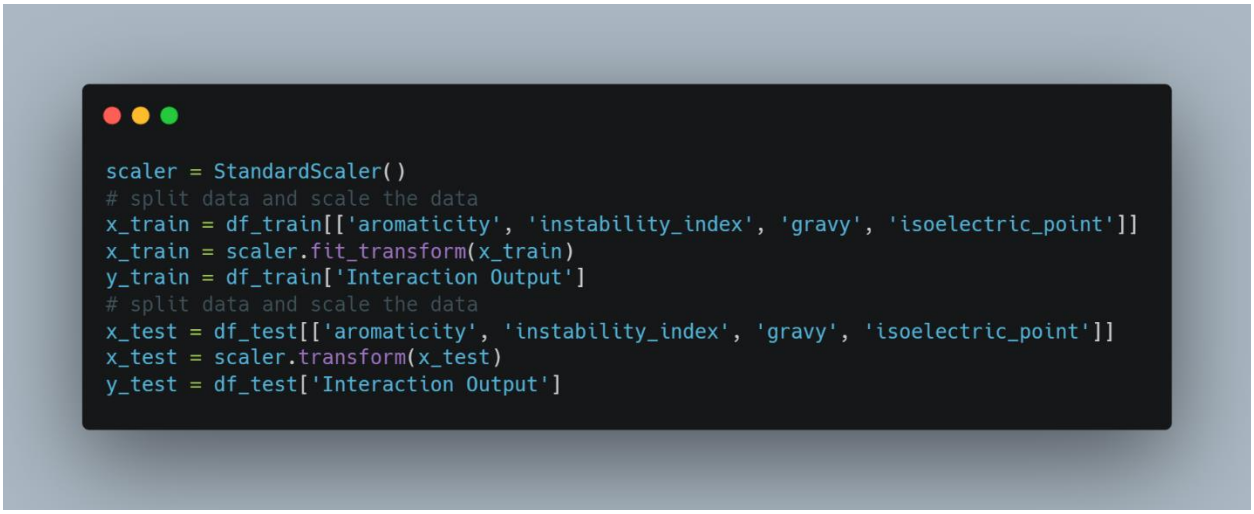
```
def getPhysicochemicalCharacteristics(proteinName, fileName, outputFile):
    aromaticity = []
    instability_index = []
    gravy = []
    isoelectric_point = []

    df = pd.read_excel(r'COVID-19 PPI Data\\{}\\{}.xlsx'.format(proteinName, fileName))
    for seq in df['Sequence']:
        sequence = ProteinAnalysis(seq)
        try:
            aromaticity.append(sequence.aromaticity())
        except:
            aromaticity.append(0)
        try:
            instability_index.append(sequence.instability_index())
        except:
            instability_index.append(0)
        try:
            gravy.append(sequence.gravy())
        except:
            gravy.append(0)
        try:
            isoelectric_point.append(sequence.isoelectric_point())
        except:
            isoelectric_point.append(0)

    df['aromaticity'] = aromaticity
    df['instability_index'] = instability_index
    df['gravy'] = gravy
    df['isoelectric_point'] = isoelectric_point

    # create excel writer object
    writer = pd.ExcelWriter('COVID-19 PPI Data\\{}\\{}.xlsx'.format(proteinName, outputFile))
    # write dataframe to excel
    df.to_excel(writer, index=False)
    # save the Excel
    writer.save()
```

- Filter data which may contain NA values



```
# filter the data that have NA values
columns = ['aromaticity', 'instability_index', 'gravy', 'isoelectric_point']
for col in columns:
    df_train = df_train[df_train[col] != 0]
    df_test = df_train[df_train[col] != 0]
```

- split and scale the data using StanderScaler method



```
scaler = StandardScaler()
# split data and scale the data
x_train = df_train[['aromaticity', 'instability_index', 'gravy', 'isoelectric_point']]
x_train = scaler.fit_transform(x_train)
y_train = df_train['Interaction Output']
# split data and scale the data
x_test = df_test[['aromaticity', 'instability_index', 'gravy', 'isoelectric_point']]
x_test = scaler.transform(x_test)
y_test = df_test['Interaction Output']
```

- apply 5-kfold for 3 different classifiers (SVM-LR-RF) and print for each one:
  . accuracy
  . precision
  . recall
  . confusion matrix

```python
def model(x_train, y_train, x_test, y_test):
    # build model
    models = [RandomForestClassifier(n_estimators=100),svm.SVC(),LogisticRegression()]
    names = ['RandomForest','SVM','LogisticRegression']
    # define cross-validation method to use
    cv = StratifiedKFold(n_splits=5, random_state=None, shuffle=True)
    scorer = MultiScorer({
        'Accuracy': (accuracy_score, {}),
        'Precision': (precision_score, {'average': 'macro'}),
        'Recall': (recall_score, {'average': 'macro'})
    })
    for model, name in zip(models, names):
        print(name)
        _ = cross_val_score(estimator = model, X = x_train, y= y_train, cv=cv,scoring=scorer)
        results = scorer.get_results()
        model.fit(x_train, y_train)
        filename = name + '.pkl'
        pickle.dump(model, open(filename, 'wb'))

        for metric_name in results.keys():
            average_score = np.average(results[metric_name])
            print('%s : %f' % (metric_name, average_score))
        print()

        y_pred = cross_val_predict(model, x_test, y_test, cv=cv)
        cf_matrix = confusion_matrix(y_test, y_pred)
        # plot the confusion matrix
        pp_matrix_from_data(y_test, y_pred)
```

- **Random forest:**

```
RandomForest
Accuracy : 0.991421
Precision : 0.991472
Recall : 0.991414
```

**Figure 6.1(Accuracy, Precision and Recall of Random Forest)**


**Figure 6.2(Confusion Matrix of Random Forest)**

- **SVM:**

```
SVM
Accuracy : 0.982469
Precision : 0.982513
Recall : 0.982462
```

**Figure 6.3(Accuracy, Precision and Recall of SVM)**

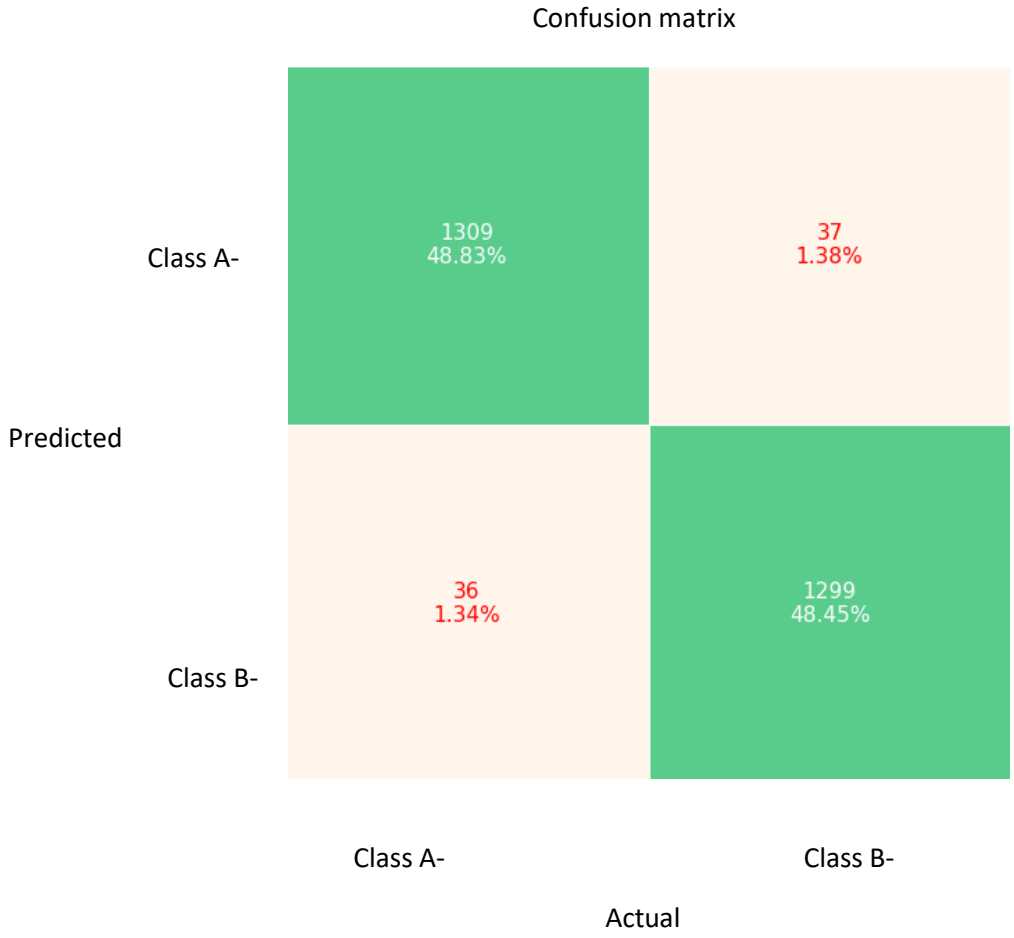



**Figure 6.4(Accuracy, Precision and Recall of SVM)**

- **Logistic Regression:**

```
LogisticRegression
Accuracy : 0.877160
Precision : 0.877649
Recall : 0.877197
```

**Figure 6.5(Accuracy, Precision and Recall of Logistic Regression)**

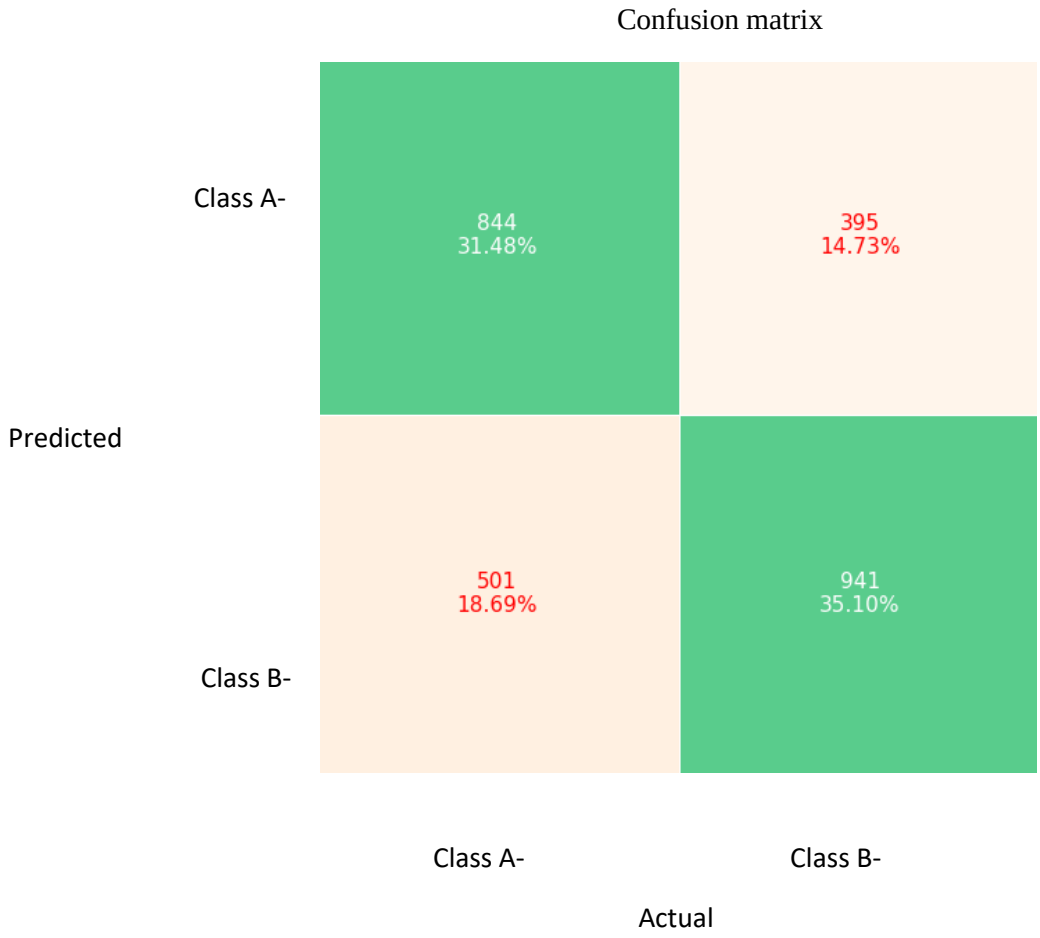



**Figure 6.6(Confusion Matrix of Logistic Regression)**

- Save three different models:

```
def model(x_train, y_train, x_test, y_test):
  ...
  names = ['RandomForest','SVM','LogisticRegression']
  filename = name + '.pkl'
  pickle.dump(model, open(filename, 'wb'))
```

- As we mention in data of phase 1 in tables 7.1 and 7.2 we will apply the same data preprocessing.

- AVL mapping method:

```python
# Create a tree node
class TreeNode(object):
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1
        self.depth = 0

class AVLTree(object):
    def __init__(self):
        self.depth={}
    # Function to insert a node
    def insert_node(self, root, key):

        # Find the correct location and insert the node
        if not root:
            return TreeNode(key)
        elif key < root.key:
            root.left = self.insert_node(root.left, key)
        else:
            root.right = self.insert_node(root.right, key)

        root.height = 1 + max(self.getHeight(root.left),
                              self.getHeight(root.right))

        # Update the balance factor and balance the tree
        balanceFactor = self.getBalance(root)
        if balanceFactor > 1:
            if key < root.left.key:
                return self.rightRotate(root)
            else:
                root.left = self.leftRotate(root.left)
                return self.rightRotate(root)

        if balanceFactor < -1:
            if key > root.right.key:
                return self.leftRotate(root)
            else:
                root.right = self.rightRotate(root.right)
                return self.leftRotate(root)

        return root
```

```python
# Function to delete a node
    def delete_node(self, root, key):

        # Find the node to be deleted and remove it
        if not root:
            return root
        elif key < root.key:
            root.left = self.delete_node(root.left, key)
        elif key > root.key:
            root.right = self.delete_node(root.right, key)
        else:
            if root.left is None:
                temp = root.right
                root = None
                return temp
            elif root.right is None:
                temp = root.left
                root = None
                return temp
            temp = self.getMinValueNode(root.right)
            root.key = temp.key
            root.right = self.delete_node(root.right,
                                          temp.key)
        if root is None:
            return root

        # Update the balance factor of nodes
        root.height = 1 + max(self.getHeight(root.left),
                              self.getHeight(root.right))

        balanceFactor = self.getBalance(root)

        # Balance the tree
        if balanceFactor > 1:
            if self.getBalance(root.left) >= 0:
                return self.rightRotate(root)
            else:
                root.left = self.leftRotate(root.left)
                return self.rightRotate(root)
        if balanceFactor < -1:
            if self.getBalance(root.right) <= 0:
                return self.leftRotate(root)
            else:
                root.right = self.rightRotate(root.right)
                return self.leftRotate(root)
        return root
```

```python
# Function to perform left rotation
    def leftRotate(self, z):
        y = z.right
        T2 = y.left
        y.left = z
        z.right = T2
        z.height = 1 + max(self.getHeight(z.left),
                           self.getHeight(z.right))
        y.height = 1 + max(self.getHeight(y.left),
                           self.getHeight(y.right))
        return y

    # Function to perform right rotation
    def rightRotate(self, z):
        y = z.left
        T3 = y.right
        y.right = z
        z.left = T3
        z.height = 1 + max(self.getHeight(z.left),
                           self.getHeight(z.right))
        y.height = 1 + max(self.getHeight(y.left),
                           self.getHeight(y.right))
        return y

    # Get the height of the node
    def getHeight(self, root):
        if not root:
            return 0
        return root.height
```

```
# Get balance factore of the node
    def getBalance(self, root):
        if not root:
            return 0
        return self.getHeight(root.left) - self.getHeight(root.right)

    def getMinValueNode(self, root):
        if root is None or root.left is None:
            return root
        return self.getMinValueNode(root.left)

    def preOrder(self, root):
        if not root:
            return
        print("{0} ".format(root.key), end="")
        self.preOrder(root.left)
        self.preOrder(root.right)

    def buildDepthDict(self,currPtr):
        if currPtr != None:
            self.depth[currPtr.key]=currPtr.depth
            self.buildDepthDict(currPtr.left)
            self.buildDepthDict(currPtr.right)

        return self.depth

    def setDepth(self,root,depth):
        if root !=None:
            root.depth=depth
            self.setDepth(root.left,depth+1)
            self.setDepth(root.right, depth + 1)

    # Print the tree
    def printHelper(self, currPtr, indent, last):
        if currPtr != None:
            sys.stdout.write(indent)
            if last:
                sys.stdout.write("R----")
                indent += "     "
            else:
                sys.stdout.write("L----")
                indent += "|    "
            print(currPtr.key)
            self.printHelper(currPtr.left, indent, False)
            self.printHelper(currPtr.right, indent, True)
```

- Build AVL for each sequence:

```
myTree = AVLTree()
root = None
nums = ['A',
for num in nums:'Q','E','G','H','I','L','K','M','F','P','S','T','W','Y','V']
    root = myTree.insert_node(root, num)
myTree.setDepth(root,0)
AVLDepth=myTree.buildDepthDict(root)
```

```
def AVLMapper(protein):
    MappedProtein=''
    for aminoAcid in protein:
        if aminoAcid =='X':
            MappedProtein='NA'
            break
        else:
            #print(AVLDepth[aminoAcid])
            MappedProtein+=str(AVLDepth[aminoAcid])
    return MappedProtein

def MappingAVL(proteinName, fileName, outputFile):
    df = pd.read_excel(r'COVID-19 PPI Data/{}/{}.xlsx'.format(proteinName, fileName))
    SequenceMapped = []
    lengths = []
    for seq in df['Sequence']:
        sequence = AVLMapper(seq)
        SequenceMapped.append(sequence)
        lengths.append(len(sequence))

    df['Mapped'] = SequenceMapped
    df['length'] = lengths
    # create excel writer object
    writer = pd.ExcelWriter('COVID-19 PPI Data/{}/{}.xlsx'.format(proteinName, outputFile))
    # write dataframe to excel
    df.to_excel(writer, index=False)
    # save the Excel
    writer.save()
```

- Data after apply method:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | **Protein ID** | **raction Ou** | **Sequence** | **Mapped** |
| 2 | NP_00212 | 1 | MERPQPD | 42332341 |
| 3 | AAX36773. | 1 | MERPQPD | 42332341 |
| 4 | AAV38587 | 1 | MERPQPD | 42332341 |
| 5 | NP_00212 | 1 | MERPQPD | 42332341 |
| 6 | 4WD4_A | 1 | GSHMMEF | 31144233 |
| 7 | ADZ76424. | 1 | MERPQPD | 42332341 |
| 8 | XP_030861 | 1 | MERPQPD | 42332341 |
| 9 | AAH01491 | 1 | MERPQPD | 42332341 |
| 10 | BAG35941 | 1 | MERPQPD | 42332341 |
| 11 | XP_023077 | 1 | MERPQPD | 42332341 |
| 12 | PNJ49363. | 1 | MERPQPD | 42332341 |
| 13 | XP_032003 | 1 | MERPQPD | 42332341 |
| 14 | NP_00112 | 1 | MERPQPD | 42332341 |
| 15 | XP_030671 | 1 | MERPQPD | 42332341 |
| 16 | XP_007973 | 1 | MERLQPD | 42332341 |
| 17 | XP_025255 | 1 | MERLQPD | 42332341 |
| 18 | EHH65790 | 1 | MERLQPD | 42332341 |
| 19 | XP_011893 | 1 | MERLQPD | 42332341 |
| 20 | XP_033088 | 1 | MERPQPD | 42332341 |
| 21 | XP_021777 | 1 | MERLQPD | 42332341 |
| 22 | EHH20181 | 1 | MERLQPD | 42332341 |
| 23 | XP_011834 | 1 | MERLQPD | 42332341 |
| 24 | XP_017359 | 1 | MERPQPD | 42332341 |
| 25 | XP_011834 | 1 | MERLQPD | 42332341 |

**Tables 8(Data after mapping using AVL)**

- Continue data pre-processing:

```
# filter the data that have NA values
df_train = df_train.dropna()
df_test = df_test.dropna()


#slicing the input
x_train = np.array(df_train[['Mapped']],dtype=object)
y_train = np.array(df_train['Interaction Output'])
x_test = np.array(df_test[['Mapped']],dtype=object)
y_test = np.array(df_test['Interaction Output'])

#Convert x_train into array contain each sequence in a list
sequencesMappedTrain = []
for seq in x_train:
    sequenceArray = []
    for index in range(len(seq[0])):
        sequenceArray.append(int(seq[0][index]))
    sequencesMappedTrain.append(sequenceArray)

sequencesMappedTest = []
for seq in x_test:
    sequenceArray = []
    for index in range(len(seq[0])):
         sequenceArray.append(int(seq[0][index]))
    sequencesMappedTest.append(sequenceArray)
```

- Build RNN model with the following structure:

```
def modelRNN(x_train, y_train):

    #Embedding
    maxlen = 5668
    x_train = pad_sequences(x_train,maxlen= maxlen,padding='post',value=-1)
    x_test = pad_sequences(x_test,maxlen = maxlen,padding='post',value=-1)

    #Prepare data for model
    x_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
    x_train= tf.cast(x_train,tf.float32)

    model = Sequential()
    model.add(Bidirectional(LSTM(64, activation='relu',return_sequences=True ,
                                 input_shape=(x_train.shape[0],x_train.shape[1]))))
    model.add(Dropout(0.25))
    model.add(Bidirectional(LSTM(32, activation='relu',return_sequences=True)))
    model.add(Dropout(0.25))
    model.add(Bidirectional(LSTM(16, activation='relu',return_sequences=True)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add((Dense(512, activation='sigmoid', )))
    model.add((Dense(1, activation='sigmoid', )))
    model.add(Flatten())
    optim=tf.keras.optimizers.SGD(learning_rate=0.0001,momentum=0.9,name='SGD')
    model.compile(loss='binary_crossentropy', optimizer=optim, metrics=['accuracy'])
    history = model.fit(x_train, y_train,
                        epochs=500,
                        batch_size=64,
                        verbose=2,callbacks = [EarlyStopping(monitor='val_acc', patience=2)])
    model.save('Rnn_model.h5')
```

- Run model:

```
model = modelRNN(sequencesMappedTrain,y_train)
```

- Save the model:

```
model.save('Rnn_model.h5')
```

- Accuracy while training:

```
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy
42/42 - 1s - loss: 0.0238 - accuracy: 0.9970 - 1s/epoch - 33ms/step
Epoch 495/500
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy
42/42 - 1s - loss: 0.0245 - accuracy: 0.9978 - 1s/epoch - 32ms/step
Epoch 496/500
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy
42/42 - 1s - loss: 0.0267 - accuracy: 0.9963 - 1s/epoch - 33ms/step
Epoch 497/500
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy
42/42 - 1s - loss: 0.0239 - accuracy: 0.9966 - 1s/epoch - 32ms/step
Epoch 498/500
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy
42/42 - 1s - loss: 0.0225 - accuracy: 0.9981 - 1s/epoch - 33ms/step
Epoch 499/500
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy
42/42 - 1s - loss: 0.0246 - accuracy: 0.9981 - 1s/epoch - 34ms/step
Epoch 500/500
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy
42/42 - 1s - loss: 0.0266 - accuracy: 0.9978 - 1s/epoch - 33ms/step
```

**Figure 7(Accuracy of RNN Model)**

- Prepare test data for testing:

```
#Prepare x_test
x_test = pad_sequences(sequencesMappedTest,maxlen = 5668,padding='post',value=-1)
x_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
x_test= tf.cast(x_test,tf.float32)
```

- Load the model and make prediction:

```
# Prediction Part
from sklearn.metrics import confusion_matrix
from keras.models import load_model
# load model from single file
model = load_model('Rnn_model.h5')
# make predictions
y_pred_first = model.predict(x_test, verbose=2)
y_pred = []
for element in y_pred_first:
  if(element >=0.1):
    y_pred.append(1)
  else:
    y_pred.append(0)
cf_matrix = confusion_matrix(y_test, y_pred)
# plot the confusion matrix
pp_matrix_from_data(y_test, y_pred)
```

- Plot confusion matrix for RNN model:

```
cf_matrix = confusion_matrix(y_test, y_pred)
# plot the confusion matrix
pp_matrix_from_data(y_test, y_pred)
```


**Figure 8(Confusion Matrix of RNN)**

## References


[1]. Chen Y, Xu J, Yang B, Zhao Y, He W (2012) A novel method for prediction of protein interaction sites based on integrated RBF neural networks. Comput Biol Med 42(4):402–407. https://doi.org/10.1016/j.compbiomed.2011.12.007

[2]. W. Zhou, H. Yan, X. Fan, and Q. Hao, "Prediction of protein-protein interactions based on molecular interface features and the support vector machine," Current Bioinformatics, vol. 8, no. 1, pp. 3–8, 2013. https://doi.org/10.2174/157489313804871597

[3]. Alakus, T.B., Turkoglu, I. A Novel Protein Mapping Method for Predicting the Protein Interactions in COVID-19 Disease by Deep Learning. Interdiscip Sci Comput Life Sci 13, 44–60 (2021). https://doi.org/10.1007/s12539-020-00405-4

-Gordon DE, Jang GM, Bouhaddou JM, Xu J, Obernier K, White KM, O'Meara MJ et al (2020) A SARS-CoV-2 protein interaction map reveals targets for drug repurposing. Nature. https://doi.org/10.1038/s41586-020-2286-9

-Khailany RA, Safdar M, Ozaslan M (2020) Genomic characterization of a novel SARS-CoV-2. Gene Rep 19:100682. https://doi.org/10.1016/j.genrep.2020.100682

-Dimitrova M, Imbert I, Kieny MP, Schuster C (2003) Protein-protein interactions between Hepatitis C virus nonstructural proteins. J Virol 77(9):5401–5414. https://doi.org/10.1128/JVI.77.9.5401-5414.2003

-Song J, Liu Y, Gao P, Hu Y, Chai Y et al (2018) Mapping the nonstructural protein interaction network of porcine reproductive and respiratory syndrome virus. J Virol 92(24):112–118. https://doi.org/10.1128/JVI.01112-18

# Predicting Covid-19 protein protein interactions

*Mohamed[m], Tarek[i], Mohamed[i], Hasnaa[a], and Mennatallah[m]*

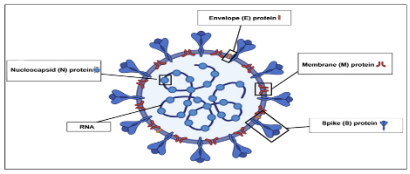
Supervisor: Dr. Hanaa , Dr.Ahmed

## Abstract

Corona virus incident occurred in Wuhan, China in December 2019, and spread rapidly to every region of the world it is vital to develop new strategies to counteract the SARS-COV-2 virus, to have knowledge of how the virus contacted the host during infection, and to develop new drugs or to reuse existing drugs. However, clinical trials are being conducted for any treatment, and both RNA and protein sequences are used effectively. One of these methods is based on protein–protein interactions. To predict protein interactions, protein sequences need to be mapped. There are various types and numbers of protein-mapping methods used in this area. In this project, an Biopython protein-mapping method and AVL tree mapping method were proposed to predict the interactions of non-structural proteins belonging to COVID-19 with other human proteins.in phase 1 we mapped the data using protein analysis method , and the data normalized and classified it with random forest which gave us 99.33% accuracy. In phase 2 we mapped data using AVL tree, protein sequence represented as a numeric according to each amino acid depth in AVL , then classified by RNN and get 99.58% accuracy which is a great for algorithm-based mapping method .

## Introduction

The genomic structure of the SARS-COV-2 virus, causing COVID-19 has been investigated and the proteins of the virus have been identified in the literature. The virus consists of four structural: S (surface), M (membrane), E (envelope), N (nucleocapsid), and six non-structural (orf3a, orf3b, orf6, orf7a, orf7b, and orf8) genes. In this project, non-structural proteins were used and the interaction information between COVID-19, and human protein pairs were obtained from BioGRID dataset. The reason for using non-structural proteins in the study is that these proteins are thought to be necessary for the replication of viral genomes. Similarly, non-structural proteins important for viral RNA synthesis and for antagonizing host antiviral immunity. Therefore, predicting or determining the interaction network of non-structural proteins is key to understanding protein interactions. Computational prediction of PPIs can be used to discover new PPIs and identify errors in the experimental PPI data. In computational methods, first genomic sequences are mapped and then protein interactions are predicted by classifying them with machine-learning and deep-learning approaches.




Covid-19 structure

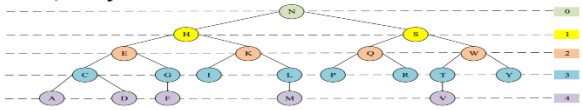
## Methods

**-First Method (BioPython):**

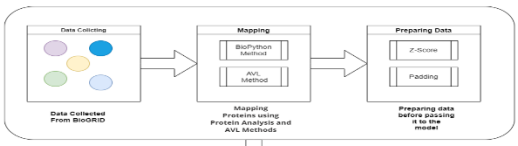
After getting the data it is the time to mapping protein using the mentioned method, we passed our data set to the Biopython function that would output for every input protein some physiochemical characteristics that would be our features. Then, the extracted data normalized by standardization (Z-Score). After that the data was ready to be classified. We used three different machine learning classifiers (Logistic Regression -Support Vector Machine -Random Forest) .

**-Second Method (AVL):**

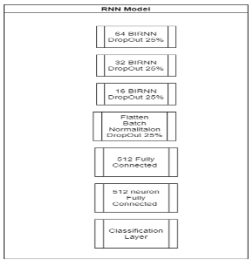
We will pass our data to AVL tree (mapping method) that gives us a numeric representation to sequences The mapped protein sequences were then padding to the maximum sequence Length. interactions were classified with the RNN deep-learning model, binary classification is made.




AVL Tree Mapping As amino acids inserted on it then assign the depth of each node to it's amino acid



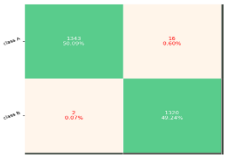

Project Design



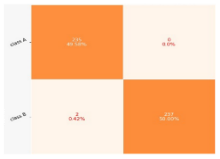

RNN Model Structure

## Primarily Design

- For Machine learning classifiers with BioPython Mapping method we got an 87.8% accuracy for Logistic Regression ,98.2% accuracy for SVM and 99.3% accuracy for Random Forest.
- For Deep Learning BiRNN with AVL Tree mapping method we got an 99.58% accuracy.
- We can say that algorithm based mapping method can be dependable to predict PPI.



Random Forest Confusion Matrix



RNN Confusion Matrix

## Conclusion

In this project, an Biopython protein-mapping method and AVL tree mapping method were proposed to predict the interactions of non-structural proteins belonging to COVID-19 with other human proteins, data collected from the BioGrid dataset. In phase 1 we mapped the data using protein analysis method , and the data normalized using z-score ,then the data is classified it with random forest which gave us 99.33% accuracy. In phase 2 we mapped data using AVL tree we managed to apply non physiochemical feature extraction (AVL) , protein sequence represented as a numeric representation according to each amino acid depth in AVL , then these sequence classified by RNN and get 99.58% accuracy which is a great for algorithm-based mapping method. Compared to experimental methods, computational methods are time efficient and can analyze the protein interactions with less equipment. Furthermore, with the recent development of technology, protein sequence information can be obtained easily.

- mohamed.mahmoud0726@gmail.com
- tarekdrias321@gmail.com
- mohamadidrees@mail.ru
- Hasnaaalirl545@gmail.com
- mennamubarak44@gmail.com

/mohamed-thesnak
/tarek-idrees-7417b2175
/mohamad-idrees-009b4a194
/hasnaa-ali-0b8528240
/manna-mubarak-734109232